Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Asistente digital mediante Machine Learning para detección de fallo del sensor del ángulo de ataque de una aeronave

Autor: Francisco José Vélez Ruiz

Tutor: María de los Ángeles Martín Prats

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Asistente digital mediante Machine Learning para detección de fallo del sensor del ángulo de ataque de una aeronave

Autor:
Francisco José Vélez Ruiz

Tutor:
María de los Ángeles Martín Prats
Profesora Titular de Universidad

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Asistente digital mediante Machine Learning para detección de fallo del sensor del ángulo de ataque de una aeronave

Autor: Francisco José Vélez Ruiz
Tutor: María de los Ángeles Martín Prats

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

1.1 Sector aeronáutico: innovación y seguridad

Desde sus inicios, el sector de la aviación siempre ha destacado por ser un sector puntero en cuanto a innovación e incorporación de la última tecnología. Hoy en día, la tendencia sigue siendo la misma. En España cada vez es mayor la participación en el sector aeronáutico, habiéndose multiplicado por cuatro en los últimos quince años. De esta forma, la industria aeronáutica representa un 4.7 % del PIB industrial con un porcentaje de más del 80 % dedicado a la exportación.

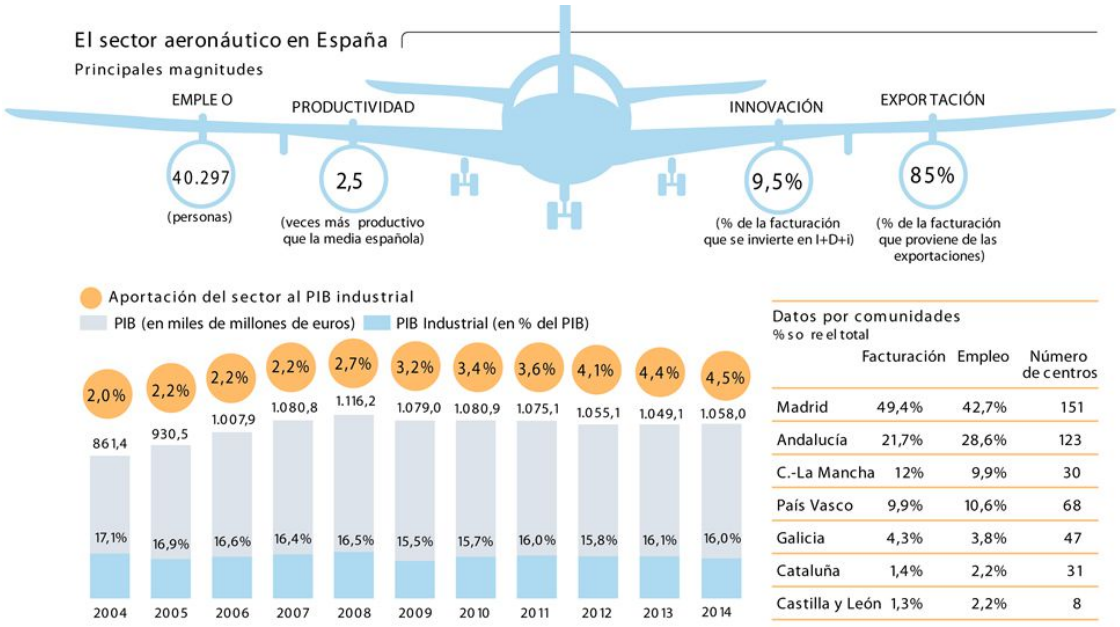


Figura 1.1 Sector aeronáutico en España desde 2004 a 2014.

España se ha convertido en uno de los pocos países del entorno europeo con la experiencia, el conocimiento, el desarrollo tecnológico y la capacidad industrial necesarios para diseñar, fabricar y poner en vuelo un avión. Más del 10 % de la facturación de la industria aeroespacial española se reinvierte en I+D+i frente al 3 % de media nacional, según datos del INE.

Por otro lado, también adquiere esta industria un gran papel a nivel global. Tanto el transporte civil como el logístico tienen una gran necesidad de realizar trayectos intercontinentales, entre países o incluso entre ciudades de un mismo país. También la industria militar juega un gran papel en el sector aeronáutico en la mayoría de los países del mundo.

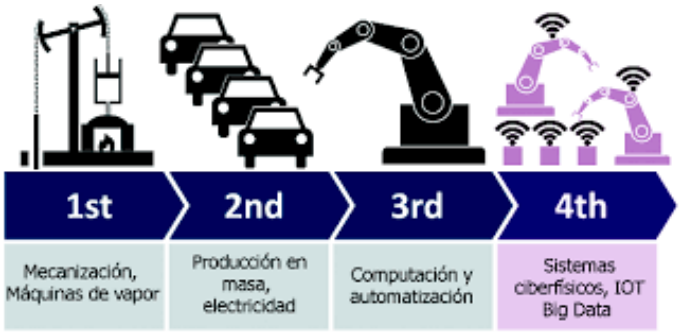
Por estas razones, las grandes empresas tienden a incorporar la tecnología más puntera en sus productos. Esta tecnología, hoy en día, apunta hacia aviones menos contaminantes, aviones eléctricos, aviones más eficientes y aviones más seguros (habiendo sido la seguridad un objetivo presente en toda la historia de la aviación).

El objeto de este estudio es mejorar la seguridad de los aviones mediante la predicción de fallos en el sistema de ángulo de ataque. Ya que el hecho de que un sensor de ángulo de ataque arroje un resultado erróneo podría desencadenar un accidente catastrófico. Para conseguir este objetivo marcado, se utilizarán técnicas de Machine Learning, las cuales son posibles de implementar gracias a la disposición de una gran cantidad de datos simulados.

1.2 La cuarta revolución industrial y el Big Data

Hace varios años que las grandes empresas se están percatando de la cantidad de beneficios que se pueden obtener de los datos. Es por eso que, para sacar provecho de los mismos, cada vez más se están intentando almacenar de una forma correcta. A esta gran cantidad de datos se le denomina Big Data. Muchos expertos dicen que el Big Data ya está teniendo una gran repercusión en el presente y que esta repercusión se incrementará en el futuro. Algunos lo llaman la era del Big Data.

Por otro lado, también se habla de la actualidad como la cuarta revolución industrial. En esta cuarta revolución, la automatización es cada vez más frecuente en todos los sectores. Para conseguir muchos de los objetivos propuestos de cara a la automatización, el Big Data (unido a herramientas predictivas como el Machine Learning o el Deep Learning) es esencial.



En cuanto a la industria aeronáutica, cada vez son más las empresas que están introduciendo esta tecnología en sus productos. Desde la predicción de fallos de turbofanes hasta la predicción de demanda de pasajeros y optimización de rutas, el Big Data y la inteligencia aritificial están jugando papeles claves en estos sistemas. Se prevé que la utilización de esta tecnología cada vez sea mayor en la industria aeronáutica.

Como se ha dicho anteriormente, este estudio tratará de sacar provecho de una gran cantidad de datos (se podría denominar Big Data) mediante técnicas de Machine Learning para predecir fallos en los sensores de ángulo de ataque y, por tanto, aumentar la seguridad de las aeronaves.

2 Descripción del problema

2.1 Tecnología del sensor AOA del B737 en la actualidad

El objeto de este estudio será aumentar la redundancia en el sistema de detección de entrada en pérdida del avión comercial B737. En la actualidad, dicho sistema se basa solamente en dos sensores de ángulo de ataque. Sin embargo, en este estudio se supondrá que tiene tres sensores, para que el modelo lógico tenga más situaciones diferentes a comparar con el modelo estadístico que se impolementará.

Figura 2.1 Sensores AOA del B737.

Por tanto, se supone que el modelo B737 tiene tres sensores AOA para medir el ángulo de ataque, de forma que, gracias a la redundancia, se pueda prevenir si alguno de ellos está dando un ángulo de ataque erróneo.

El modelo lógico del B737 es el siguiente (conviene indicar que se le ha llamdo AOAf al ángulo de ataque que finalmente tomaría el sistema como correcto):

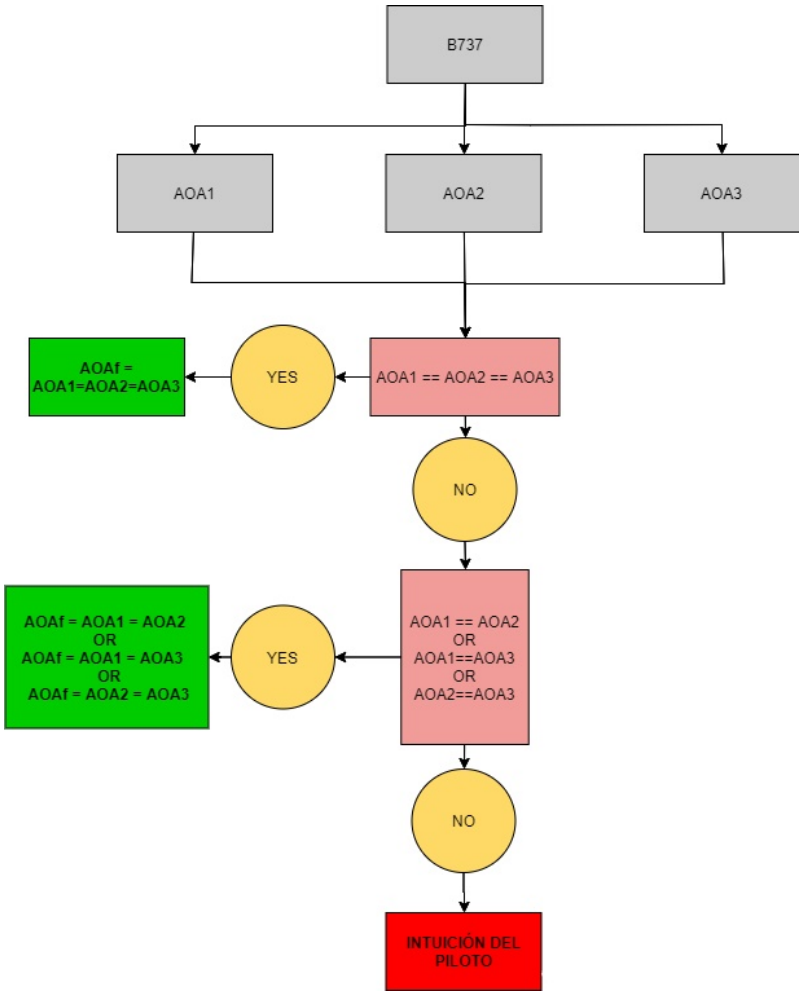


Figura 2.2 Modelo lógico con 3 sensores AOA.

De esta forma, se pueden dar los siguientes casos:

1. **Los tres sensores miden el mismo ángulo de ataque**: en este caso el sistema toma como correcto cualquiera de los tres outputs, puesto que son iguales.
2. **Dos sensores miden el mismo ángulo de ataque y el tercero mide un ángulo diferente**: en este caso el sistema toma como correcto el ángulo en el que coinciden dos de los sensores.
3. **Cada uno de los sensores mide un ángulo diferente**: en este caso no se sabe cuál ángulo es el correcto y el piloto deberá guiarse por su intuición.

El problema de este modelo es el segundo caso nombrado. Puesto que, si los dos sensores que miden el mismo ángulo están fallando, el sistema tomaría un angulo de ataque erróneo, lo que podría ser catastrófico.

2.2 Tecnología del sensor AOA del B737 con la implementación del modelo predictivo

El principal objetivo de la implementación de un modelo que prediga el ángulo de ataque sería corregir el caso nombrado anteriormente.

El modelo lógico que seguiría el sistema en este caso es el siguiente (conviene indicar que se le ha llamdo AOAf al ángulo de ataque que finalmente tomaría el sistema como correcto):

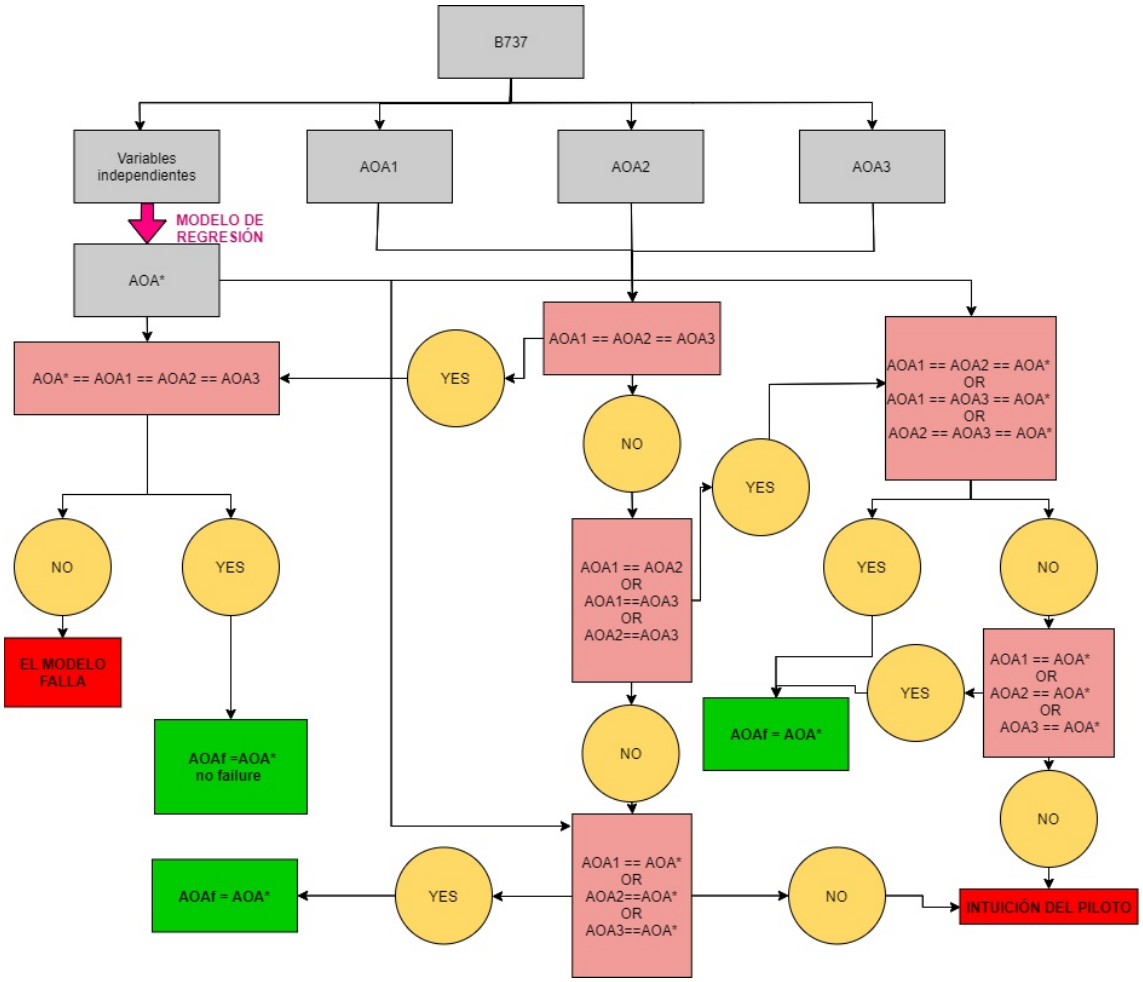


Figura 2.3 Modelo lógico con 3 sensores AOA y el modelo estadístico.

En este caso se supone que se ha implementado un modelo que ha sido entrenado con datos de vuelos anteriores. Dicho modelo recibe como input las variables del vuelo a tiempo real y obtiene como output el valor de ángulo de ataque predecido (AOA*).

Con esta implementación, se podrían dar los siguientes escenarios:

1. **Los tres sensores miden el mismo ángulo de ataque**. Ahora habría que comprobar si dicho ángulo de ataque coincide con el AOA*:
   a) **Sí coinciden**: no hay fallo y el modelo toma cualquier valor de los cuatro, puesto que son todos iguales.
   b) **No coinciden** probablemente el modelo esté fallando. El sistema tomaría cualquier valor de los tres de los sensores, puesto que los tres son iguales.
2. **Dos sensores miden el mismo ángulo de ataque y el tercero mide un ángulo diferente**. En este caso habría que comprobar si el valor de los dos sensores que coinciden también coincide con AOA*:
   a) **Sí coinciden**: dos sensores dan un valor correcto y uno está fallando. El sistema toma el valor de dichos dos sensores o el AOA*, puesto que son iguales.
   b) **No coinciden**. Habría que observar si el valor del sensor que da un ángulo diferente coincide con el AOA*:
      I. **Sí coinciden**: el valor del sensor es correcto y el sistema toma dicho valor o el AOA*, puesto que son iguales.
      II. **No coinciden**: en este caso ninguno de los tres sensores coincide ni entre ellos ni con el AOA*. El piloto deberá guiarse por la intuición.

3. **Cada uno de los sensores mide un ángulo diferente**. Habría que comprobar si alguno de los tres ángulos coincide con el AOA*:
   a) **Sí coinciden**: dicho valor es correcto. El sistema toma dicho valor o el AOA*, puesto que son iguales.
   b) **No coinciden**: el piloto deberá guiarse por la inutición.

Los datos de partida para entrenar al modelo son datos de vuelos anteriores en los que podría darse que el valor que mide alguno de los tres sensores AOA sea erróneo. Sin embargo, puesto que este escenario se da muy poco fecuentemente, apenas influirá en la eficiencia de nuestro modelo.

Todos los modelos incluídos en este documento son modelos de regresión cuyo propósito es obtener el valor de ángulo de ataque del avión mediante las variables a tiempo real proporcionadas por el mismo. En estudios posteriores que no se incluirán en este documento, se procederá a conseguir el mismo objetivo pero implementando modelos de regresión logística (con variables categóricas: 1 = NO FALLA, 0 = SÍ FALLA).

3 Datos

3.1 X Plane 11

Para conseguir el objetivo propuesto en el apartado anterior (implementar un modelo estadístico que prediga el valor del ángulo de ataque), es necesaria una gran cantidad de datos de vuelo en diferentes condiciones. Para ello, se ha utilizado el simulador XPlane11.

Figura 3.1 Logo de XPlane11.

XPlane11 ha sido catalogado como el simulador de vuelo más realista de la actualidad. Este software es utilizado por pilotos para entrenar sus habilidades así como por amantes de la aeronáutica para apreciar en primera persona el trabajo de un piloto.

Este simulador incluye de forma gratuita el modelo B737-800, que se ha utilizado en este estudio.

Figura 3.2 B737-800 en XPlane11.

Cabe destacar que no resulta trivial pilotar una aeronave de estas características con XPlane11. Para ello, se ha hecho una revisión de vídeos tutoriales de pilotos que enseñan a ello y se ha intentado realizar despegues y vuelos en crucero intentando seguir lo máximo posible las pautas que indican estos pilotos así como las condiciones de vuelo que se toman en la realidad para cada uno de los tramos. Sin embargo, se ha optado por no realizar ningún tramo de descenso y aterrizaje, ya que el hecho de que éste sea el tramo más complejo de pilotar podría hacer que los datos obtenidos no sean tan fiables como los datos de los tramos anteriores.

Figura 3.3 Panel de control del B737-800 en XPlane11.

3.2 Set de datos

Como se ha dicho anteriormente, los datos se corresponden con acciones de despegue, ascenso y crucero de ocho vuelos con diferentes condicones (altitud, velocidad crucero, velocidad de subida, turbulencias, meteorología...) simulados en XPlane11. En dicho simulador se ha seleccionado un total de 68 variables (razonablemente escogidas) para que aparezcan en el set de datos.

Para implementar los modelos, se crea un data set con todos los vuelos y se separa en despegue y asceno por un lado y crucero por otro:

- Data set crucero: 68 columnas $\times$ 36699 filas.
- Data set despegue y ascenso: 68 columnas $\times$ 57384 filas.

A continuación se adjunta una tabla en la que se especifican todos las variables que contiene el dataset. Conviene decir que el dataset incluye datos de los cuáles no se conoce muy bien su significado, pero que no son los de interés.

Tabla 3.1 Variables del set de datos I.

Nombre	Variable	Unidad
'muestra'	Número de dato obtenido.	Sin unidad
'totl,time'	Tiempo total en el que se genera el dato.	s
'VIS,tris'		
'Vind,kias'	Velocidad indicada en nudos de velocidad del aire indicada.	Nudos
'Vind,keas'	Velocidad indicada en nudos de velocidad del aire equivalente (velocidad del aire calibrada corregida para flujo adiabático compresible a la correspondiente altitud de vuelo).	Nudos
'Vtrue,ktas'	Velocidad verdadera en nudos de velocidad del aire verdadera.	Nudos
'Vtrue,ktgs'		
'Vind,mph'		
'Vtrue,mphas'		
'Vtrue,mphgs'		
'Mach,ratio'	Número de Mach.	Adimensional
'wind,speed'	Velocidad del viento.	Nudos
'wind,dir'	Dirección del viento respecto al norte medida en sentido horario. El viento de norte a sur forma 0º.	Grados
'dens,ratio'	Ratio de densidad.	
'baro,inHG'	Presión barométrica.	Pulgada de Mercurio
élev,surf'	Superficie del elevator deflectada.	
áilrn,surf'	Superficie del alerón deflectada.	
'ruddr,surf'	Superficie del rudder deflectada.	
'M,ftlb'	Momento de pitch.	Pie-libra fuerza
'L,ftlb'	Momento de roll.	Pie-libra fuerza
Ñ,ftlb'	Momento de yaw.	Pie-libra fuerza
'Q,rad/s'	Pitch rate.	rad/s
'P,rad/s'	Roll rate.	rad/s
'R,rad/s'	Yaw rate.	rad/s
'pitch,deg'	Pitch.	Grados
'roll,deg'	Roll.	Grados
'hding,true'	Heading verdadero.	Grados
'hding,mag'	Heading magnético.	Grados
'beta,deg'	Resbalamiento.	Grados
'hpath,deg'		
'vpath,deg'		
'slip,deg'		
'lat,deg'	Latitud.	Grados
'lon,deg'	Longitud.	Grados
ált,ind'	Altitud.	Pies

Tabla 3.2 Variables del set de datos II.

Nombre	Variable	Unidad
'thrst,1,lb'	Empuje motor 1	Libras
'thrst,2,lb'	Empuje motor 2	Libras
'rpm1,engin'	rpm motor 1	rpm
'rpm2,engin'	rpm motor 2	rpm
'lift,lb'	Sustentación.	Libras
'drag,lb'	Resistencia.	Libras
'L/D,ratio'	Eficiencia aerodinámica.	Adimensional
'cl,total'	Coeficiente de sustentación.	Adimensional
'cd,total'	Coeficiente de resistencia.	Adimensional
'L/D,ratio.1'		
élev1,deg'	Ángulo de deflexión del elevator.	Grados
'rudr1,deg'	Ángulo de deflexión del rudder.	Grados
'wing1,lift'		
'wing1,lift.1'		
'wing2,lift'		
'wing2,lift.1'		
'wing3,lift'		
'wing3,lift.1'		
'wing1,drag'		
'wing1,drag.1'		
'wing2,drag'		
'wing2,drag.1'		
'wing3,drag'		
'wing3,drag.1'		
'hstab,lift'		
'hstab,lift.1'		
'vstb2,lift'		
'hstab,drag'		
'hstab,drag.1'		
álpha,deg1'	AOA sensor 1	Grados
álpha,deg2'	AOA sensor 2	Grados
álpha,deg3'	AOA sensor 3	Grados

3.3 Visualización de datos

A continuación se procede a la visualización de los datos del dataset y de sus respectivas relaciones. Puesto que visualizar las 68 variables sería demasiado copioso, se creará un nuevo dataset que incluya solamente las variables que son de interés. Dichas variables son:

1. Mach
2. Wind speed
3. Wind direction*
4. Superficie del eleator
5. Superficie del alerón
6. Superficie del rudder
7. Altitud
8. Empuje total
9. rpm media
10. CL

11. CD

12. Resbalamiento

13. Pitch

14. Roll

15. AOA1

16. AOA2

17. AOA3

Cabe destacar que se han creado tres variables nuevas de la siguiente forma:

- Empuje total = empuje1 + empuje2
- rpm media = $\frac{rpm1+rpm2}{2}$
- Wind direction* = wind direction - heading

Siendo wind direction* el ángulo que forma el eje del fuselaje con la dirección del viento.
La justificaciones principales para elegir estas variables se exponen a continuación:

- Se excluyen todas las valocidades puesto que están intrínsicamente representadas por el Mach.
- Se excluyen variables que son totalmente superfluas a la hora de estudiar el ángulo de ataque. Por ejemplo: latitud, longitud, hora...
- Se excluyen todas las variables que son sustentación y resistencia puesto que están intrínsecas en el CL y en el CD.
- Se exluyen variables que, por intuición, no tendrán un gran impacto en la variable dependiente (AOA).

3.3.1 Despegue y ascenso

En primer lugar, se hace uso de la herramienta "describe" de python para visualizar las características de las variables:

	Mach	elevator surface	aileron surface	rudder surface	wind speed	pitch	roll	slippage	altitude
count	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000
mean	0.403175	-0.040480	0.001056	0.000565	3.916268	4.881140	-1.473585	-1.971678	12985.002981
std	0.201076	0.082470	0.055179	0.012439	5.161983	3.520347	8.953574	8.705548	10497.820797
min	0.000000	-0.613860	-0.319840	-0.227330	0.000000	-6.507600	-39.198620	-89.444120	0.861090
25%	0.348150	-0.080372	-0.002860	-0.000080	0.000000	2.050805	-1.090377	-0.033870	3041.534183
50%	0.465435	-0.036660	0.000000	0.000000	0.000000	6.000310	-0.006565	0.000490	11606.421390
75%	0.549750	-0.000050	0.001170	0.001460	9.913780	7.156660	0.171930	0.028450	21956.099608
max	0.659720	0.854420	1.000000	0.193910	15.908800	33.650320	42.854430	8.175340	36628.714840

Figura 3.4 Función describe para variables de despegue y ascenso I.

	CL	CD	AOA1	AOA2	AOA3	empuje total	rpm media	wind direction*
count	57379.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000	57384.000000
mean	0.223143	0.054792	2.381308	2.381350	2.381376	15805.343112	3784.769017	-69.500125
std	0.105791	0.155095	1.989129	1.989361	1.989474	10090.364502	1441.920684	108.178098
min	0.000030	0.000050	0.000000	0.000000	0.000000	-0.006580	998.243230	-359.999510
25%	0.167900	0.015520	1.068215	1.067122	1.066225	8367.570560	3023.542058	-104.678363
50%	0.223940	0.018390	2.328220	2.328903	2.329608	16125.916500	4476.479490	-72.228315
75%	0.270965	0.029460	3.295840	3.294167	3.295053	20976.432130	4933.591555	21.958423
max	1.307070	1.197450	13.943580	14.065857	14.066830	50557.152340	5113.210450	220.587040

Figura 3.5 Función describe para variables de despegue y ascenso II.

A continuación se adjunta un mapa de calor en el que se puede observar cómo de lineal son las correlaciones entre las variables:

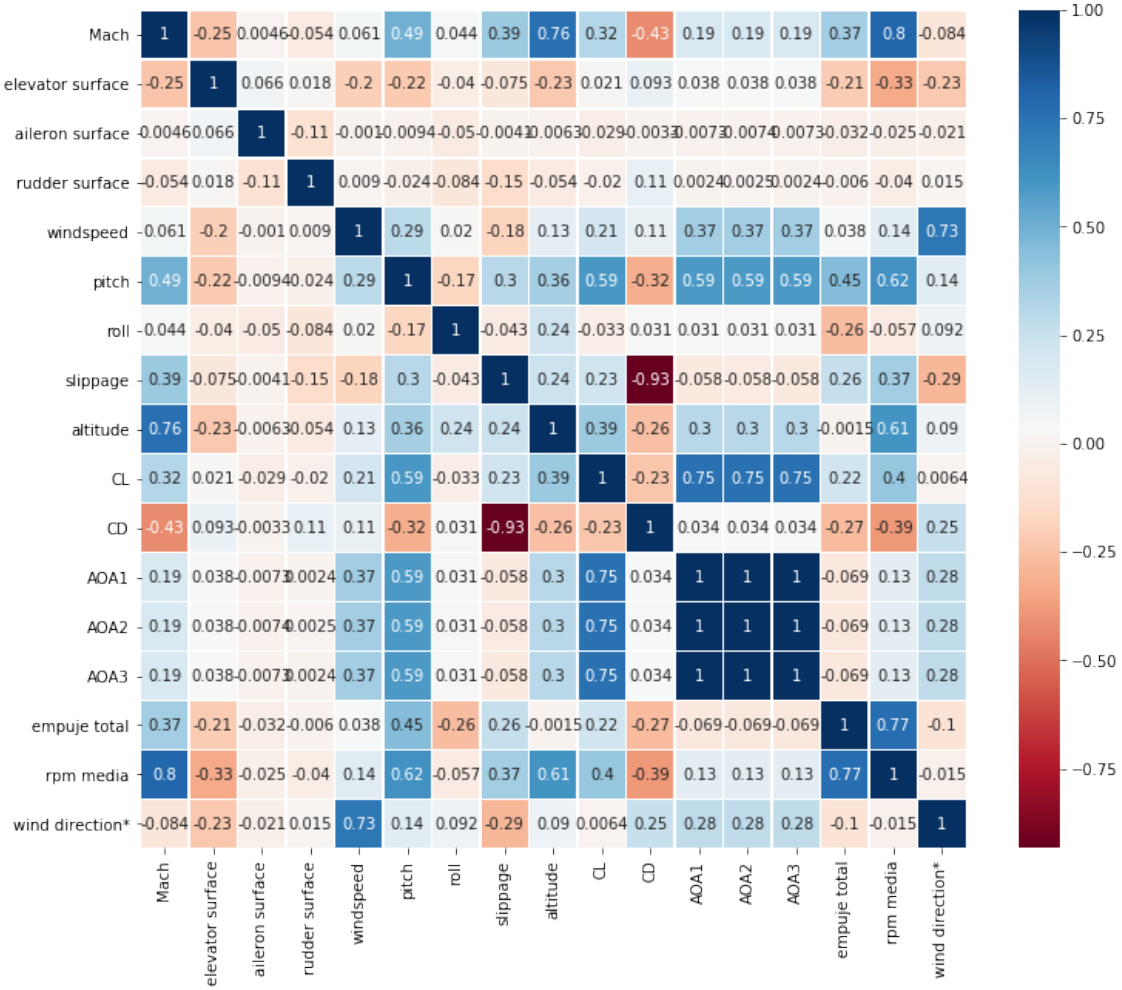


Figura 3.6 Mapa de calor (despegue y ascenso).

En nuestro caso, nos interesa las correlaciones del sensor AOA con el resto de variables. Puesto que los tres sensores dan un output muy similar (no fallan), se tomará solamente la variable AOA1 para crear los modelos.

Se observa que las variables que tienen correlaciones más altas en valor absoluto con AOA1 son:

1. Wind speed: 0.37.
2. Pitch: 0.59.
3. Altitude: 0.3.
4. CL: 0.75.
5. Wind direction*: 0.28.

Estas serán las variables que utilizaremos para crear los modelos de regresión lineal simple.
Por último, se adjunta un histograma de frecuencias para visualizar sobre qué valores se mueve el valor del ángulo de ataque:

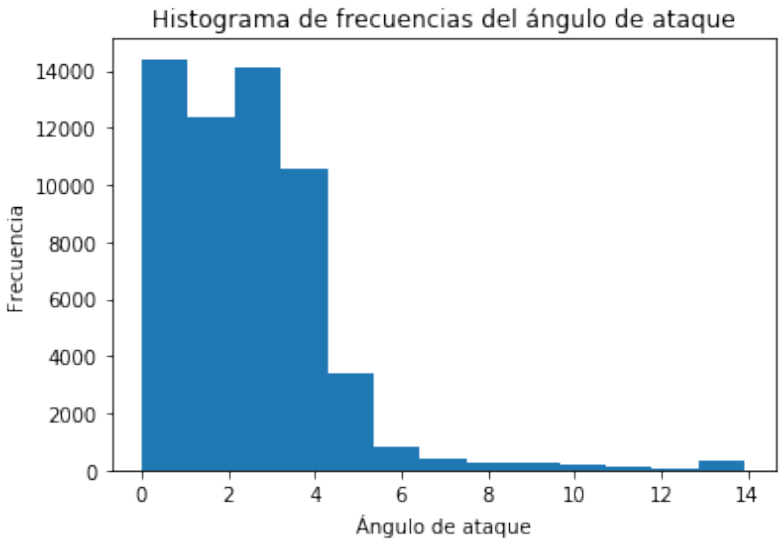


Figura 3.7 Histograma de frecuencias (despegue y ascenso).

Se puede observar una gran cantidad de valores igual a 0 o cercanos. Esto se debe a que también están incluidos los momentos previos al despegue. Sin embargo, los datos respectivos a dicho momento no se quitarán del modelo puesto que también son datos válidos en los que el modelo debería predecir un valor de ángulo de ataque igual a 0 aproximadamente.

3.3.2 Crucero

En el caso del crucero, se sigue el mismo orden de procedimientos que en la visualización de datos de despegue y ascenso.
En primer lugar, la tabla descriptiva:

	Mach	elevator surface	aileron surface	rudder surface	wind speed	pitch	roll	slippage	altitude
count	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000
mean	0.700552	-0.067214	-0.000118	0.000013	3.207904	2.249914	-0.310733	-0.000426	32269.253049
std	0.079937	0.039971	0.035841	0.008352	4.420323	1.462109	3.173111	0.200880	3641.521464
min	0.477310	-1.000000	-1.000000	-0.049820	0.000000	-3.710480	-18.771850	-1.420040	27491.039060
25%	0.649652	-0.093210	-0.000580	-0.000110	0.000000	1.135933	-0.722888	-0.011320	28001.148927
50%	0.712285	-0.058175	0.000060	0.000010	0.000000	1.747125	-0.034035	-0.000010	31995.252930
75%	0.760010	-0.040270	0.002410	0.000320	8.403330	3.561305	0.248368	0.011657	36933.931637
max	0.823000	1.000000	1.000000	0.048680	13.063500	8.373260	39.009820	2.693720	37009.707030

Figura 3.8 Función describe para variables de crucero I.

	CL	CD	AOA1	AOA2	AOA3	empuje total	rpm media	wind direction*
count	36693.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000	36694.000000
mean	0.226600	0.017725	2.180582	2.180542	2.180565	10198.544782	4289.335202	-57.703216
std	0.081368	0.003448	1.339350	1.339496	1.339421	2558.719063	400.492172	87.581922
min	0.112690	0.014630	0.000000	0.000000	0.000000	1463.977780	2048.148440	-359.993770
25%	0.166540	0.015240	1.135280	1.134385	1.134294	8408.488280	4140.768922	-69.924477
50%	0.191960	0.016430	1.557460	1.557764	1.557503	11126.765630	4343.017580	-50.199135
75%	0.309460	0.019630	3.486000	3.487174	3.487098	12364.518070	4552.981450	29.582480
max	0.486070	0.046190	6.176820	6.175349	6.234249	17895.748040	5155.750000	52.694830

Figura 3.9 Función describe para variables de crucero II.

El mapa de calor es el siguiente:

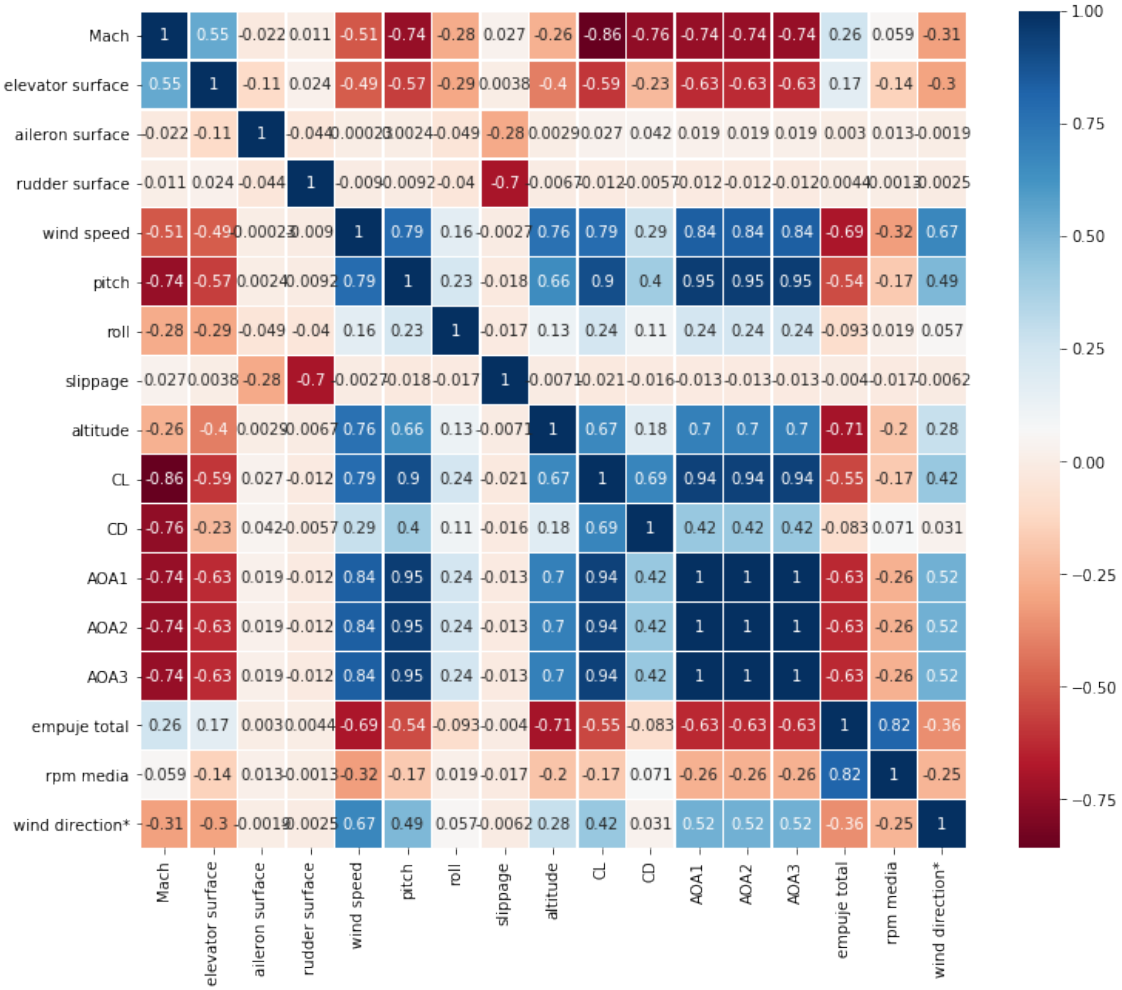


Figura 3.10 Mapa de calor (crucero).

En este caso, las variables con mayor correlación en valor absoluto son:

1. Wind speed: 0.84.
2. Pitch: 0.95.
3. Mach: -0.74.

4. Altitude: 0.7.
5. CL: 0.94.
6. CD: 0.42.
7. Empuje total: -0.63.
8. Wind direction*: 0.52.
9. Elevator surface: -0.63.
10. Roll: 0.24.
11. rpm media: -0.26.

Se observa que las correlaciones son, en valor absoluto, más altas que en el caso de despegue y ascenso. Esto se debe a que los datos de crucero son mucho más homogéneos.

Por último, el histograma de frecuencias:

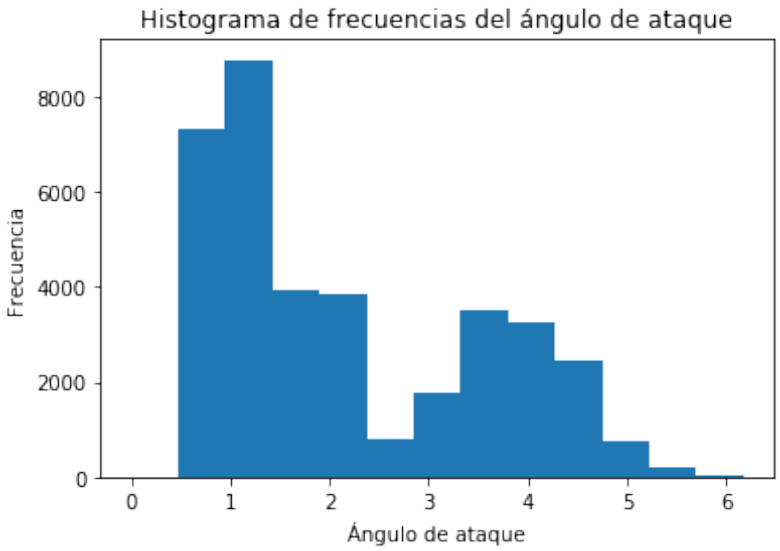


Figura 3.11 Histograma de frecuencias (crucero).

4 Selección de características y métricas de error

4.1 Selección de características

El objetivo de este apartado es implementar algunos modelos de selección de características para tener una visión global de cuáles son las más relevantes. Mediante la selección de características, la visualización de las correlaciones del apartado anterior y los conocimientos aeronáuticos se podrá decidir qué variables introducir como variables independientes en cada modelo. Cabe destacar que estas varibales puede que tengan una alta dependencia entre sí (por tanto no serían independientes), por lo que entorpecería los resultados del modelo. Sin embargo, este fenómeno se estudiará más adelante, de forma que se eliminarán las variables que no sean las adecuadas.

Para implementar la selección de características se utilizará Python y la librería Scikit-Learn. A continuación, se describen los modelos de selección de características utilizados y los resultados obtenidos.

4.1.1 Selección de características basado en percentiles

Este método estadístico se basa en elegir las variables de acuerdo al percentil con la mayor eficiencia. Es decir, evalúa la eficiencia de cada una de las variables y escoge aquellas cuyo número de datos que superen un límite de eficiencia sea mayor.

Las mejores variables según este modelo son las siguientes:

Tabla 4.1 Selección de características por percentiles.

Posición	Despegue y ascenso	Crucero
1	Mach	Mach
2	Wind Speed	Rudder Surface
3	Pitch	Wind Speed
4	Altitude	Altitude
5	CL	CL
6	rpm media	Empuje total

4.1.2 Selección de características mediante el método Select K Best

Este método puntúa las características de cada variable haciendo uso de una función (en este caso la función f classif) y elimina todas las vairables excepto k vairbales con la puntuación más alta.

Las mejores variables según este modelo son las siguientes:

Tabla 4.2 Selección de características por Select K Best.

Posición	Despegue y ascenso	Crucero
1	Mach	Mach
2	Wind Speed	Wind Speed
3	Pitch	Pitch
4	Altitude	Altitude
5	rpm media	CL
6	Wind Direction	Empuje total

4.1.3 Selección de características mediante el método f regression

Este método compara modelos de regresión para puntuar el efecto individual de cada una de las variables.

Las mejores variables según este modelo son las siguientes:

Tabla 4.3 Selección de características por fregression.

Posición	Despegue y ascenso	Crucero
1	Mach	Mach
2	Wind Speed	Wind Speed
3	Pitch	Pitch
4	Altitude	Altitude
5	rpm media	CL
6	Wind Direction	Empuje total

4.1.4 Conclusión

Los resultados ayudan a tener una idea de cuáles son las avriables más relevantes, ya que éstas se repiten más o menos para los diferentes modelos. Además, los conocimientos aeronáuticos ayudan a confirmar que estas variables podrian ser bastante relevantes para el ángulo de ataque.

Finalmente, conviene anotar que dichas variables también tenían un alto valor de correlación con el ángulo de ataque.

4.2 Métricas de error

En este apartado se describirán los distintos métodos utilizador para medir la fiabilidad del modelo. Aquellos modelos que obtengan resultados más favorables en las métricas de error serán aquellos que se implementarán finalmente y que se utilizarán para comprobar si funcionan con los datos de un nuevo vuelo simulado y no incluído en los modelos.

A continuación, se describen las métricas de error.

4.2.1 Factor R^2

El factor R^2, también conocido como coeficiente de determinación, mide la exactitud en la que la variable independiente describe la variable dependiente. La definición de R^2 es bastante sencilla: es la proporción de la varianza total de la variable dependiente predecida a partir de la vairbale independiente que se etá estudiando. Es decir:

$$R^2 = \frac{\sum(\hat{y} - \bar{Y})^2}{\sum(y - \bar{Y})^2}$$

Siendo y la variable dependiente real, $\hat{y}$ la variable dependiente predecida y $\bar{Y}$ la media de la variable real.

Un valor de 1 indica un ajuste perfecto, un valor de 0 indica un que el modelo es equivalente a la media y un valor negativo indica que predicciones pesimas. Valores aceptados de R^2 en problemas reales rondan desde 0.6 a 0.9. En este caso, 0.6 será el valor que se tomará como límite a la hora de decidir si un modelo interesa ser estudidado o no.

4.2.2 Media del error absoluto

Esta métrica es bastante fácil de explicar puesto que solamente consiste en calcular el valor del error absoluto de la variable real y la predecida y hacer la media aritmética.

La fórmula del error absoluto es:

$$E_a = |y - \hat{y}|$$

4.2.3 Error cuadrático medio

El error cuadrático medio es la media aritmética de los errores cuadráticos, que se calcula de la siguiente forma:

$$E_c = (y - \hat{y})^2$$

Se puede observar que este error es el mismo que el anterior pero elevado al cuadrado, por lo que no nos aporta mucha más información. Sin emabargo, se considerará porque es una métrica muy utilizada en Machine Learning, por lo que será de utilidad para decidir si tomar un modelo o no según referencias.

Cabe destacar que la media del error absoluto no coincide con la media del error cuadrático para n $\neq 1$.

5 Teoría de los modelos a implementar

En esta sección se pretende hacer un breve resumen explicativo de los modelos estadísticos que se van a implementar posteriormente. Cabe destacar que en ningún momento se desarrollará ningún código para usar estos modelos, ya que se utilizarán librerías (por ejemplo Scikit-Learn) que tienen funciones con los modelos ya implementados. Por tanto, tampoco se profundizará en el funcionamiento interno de cada uno de los modelos, puesto que ello conllevaría alargar la memoria demasiado. Solamente se hará una breve introducción sobre en qué se basa el funcionamiento de cada modelo.

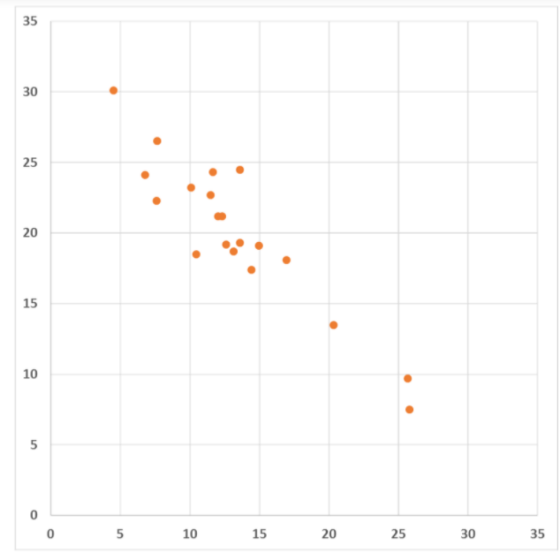
Figura 5.1 Logo de Scikits Learn.

5.1 Regresión lineal simple

La regresión lineal es un modelo que permite describir cómo influye una variable X (Mach, Altitud, CL...) sobre otra variable Y (AOA) mediante una función lineal:

$$f(x) = \beta_0 + \beta_1 x$$

El modelo de regresión lineal parte de una serie de puntos que se consideran los datos reales del problema (en este caso, los datos de los vuelos simulados):



Para generar la regresión se utiliza la técnica de ajuste de mínimos cuadrados. Para ello, en primer lugar, se calcula la media aritmética y la varianza tanto de los valores de X como los valores de Y.

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

$$\bar{y} = \sum_{i=1}^{n} \frac{y_i}{n}$$

$$\sigma_x^2 = \frac{\sum_{i=1}^{n} x_i^2 - n \cdot \bar{x}^2}{n-1}$$

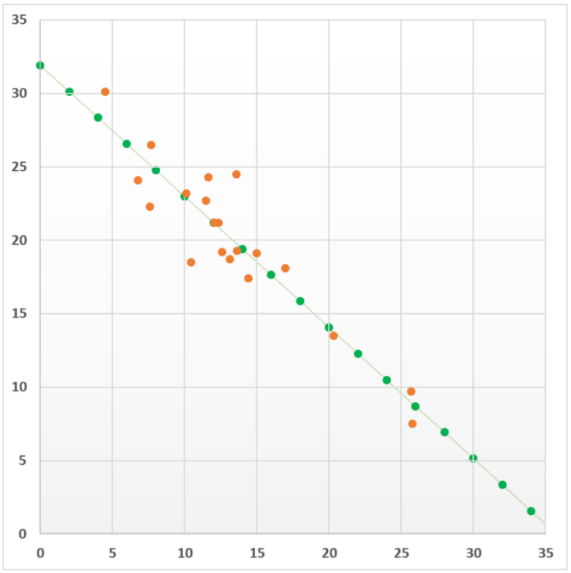
$$\sigma_y^2 = \frac{\sum_{i=1}^{n} y_i^2 - n \cdot \bar{y}^2}{n-1}$$

Por último, la pendiente y la ordenada en el origen de la función predictiva se calculan de la siguiente forma:

$$\beta_1 = \frac{\sum_{i=1}^{n} y_i y_i - n(\bar{x})(\bar{x})}{\sigma_x^2 (n-1)}$$

$$\beta_0 = \bar{y} - m\bar{x}$$

De esta forma, se genera una recta que busca comportarse de manera similar a los puntos reales.

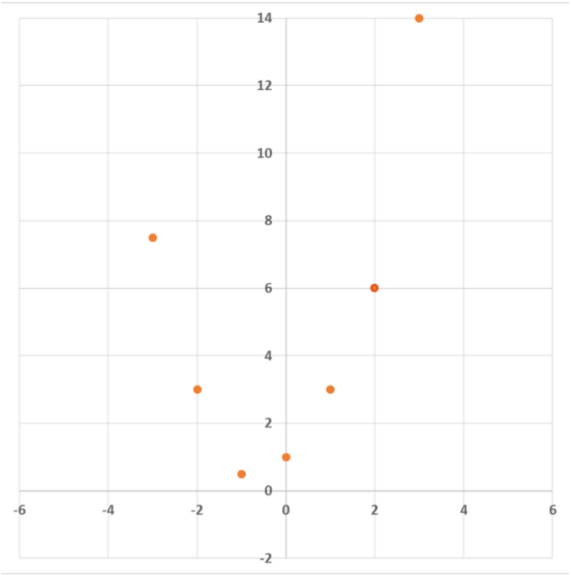


5.2 Regresión polinomial simple

Puede ocurrir que un modelo lineal se vea muy limitado a la hora de imitar el comportamiento del conjunto de datos de partida. En ese caso, puede que sea conveniente la implementación de un modelo en el que la función predictiva no sea una recta. Este es el caso de la regresión polinomial simple.

Este modelo se caracteriza por añadir curvatura al elevar la variable independiente (X) a diferentes potencias.

Supongamos ahora que los datos de partida son los siguientes:

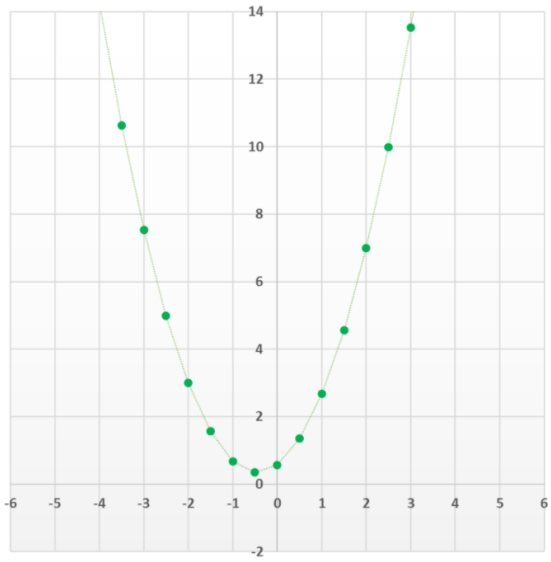


El procedimiento para generar la regresión cuadrática utilizando la técnica de ajuste de mínimos cuadrados requiere un sistema de ecuaciones que son producto de analizar la suma de los cuadrados de los residuos. Dicho sistema sería:

$$s_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

Por lo que el problema se reduciría a encontrar los parámetros de dicha ecuación para hallar la función predictiva.

De esta forma, dicha función sería la siguiente:



En este caso se ha desarrollado una regresión polinomial cuadrática, pero la función no tiene por qué ser de grado dos, sino que será de un grado n para el cuál la función se ajuste de la mejor forma posible a los datos de partida.

5.3 Regresión lineal múltiple

La regresión lineal múltiple consiste en aumentar el número de variables independientes usadas en la regresión lineal simple. Es decir, se pasa de tener una variable independiente a un número n que dependerá de la complejidad que se le quiera añadir al modelo y de la eficiencia añadida que supondría agregar ciertas variables.

Este modelo, al igual que la regresión lineal simple, crea una función de forma que, mediante la técnica de ajuste de mínimos cuadrados, imite el comportamiento de los datos de partida.

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

En este caso, puesto que se trata de un problema con n dimensiones, no es posible realizar una representación gráfica.

Para este tipo de modelo (así como para cualquier otro que sea multivariable), es necesario tener en cuenta la multicolinealidad. La multicolinealidad es un problema muy frecuente que se presenta cuando las variables explicativas son muy dependientes entre sí. En caso de darse este fenómeno, deberá optarse por eliminar aquellas variables con alta dependencia del modelo, tal y como se hará en apartados posteriores.

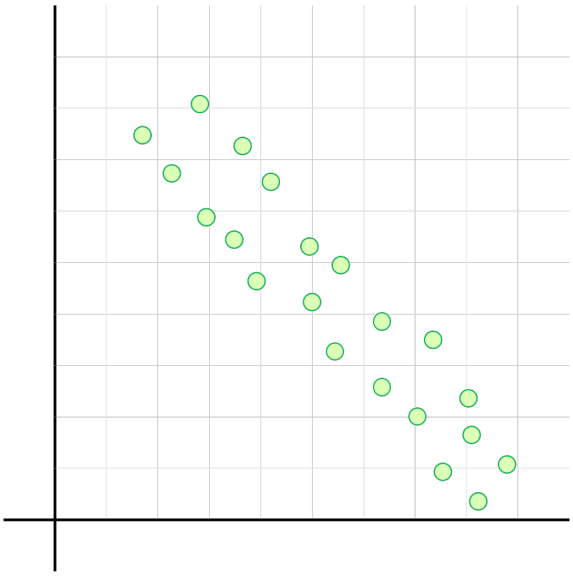
5.4 Support Vector Machine

Los algoritmos de Support Vector Machine o de máquinas de soporte vectorial son unos algoritmos que fueron desarrollados por Vladimir Vapnik en la década de los 90. Dichos algoritmos se encuentran dentro del grupo de aprendizaje supervisado y son muy útiles para solucionar problemas tanto de clasificación como de regresión.

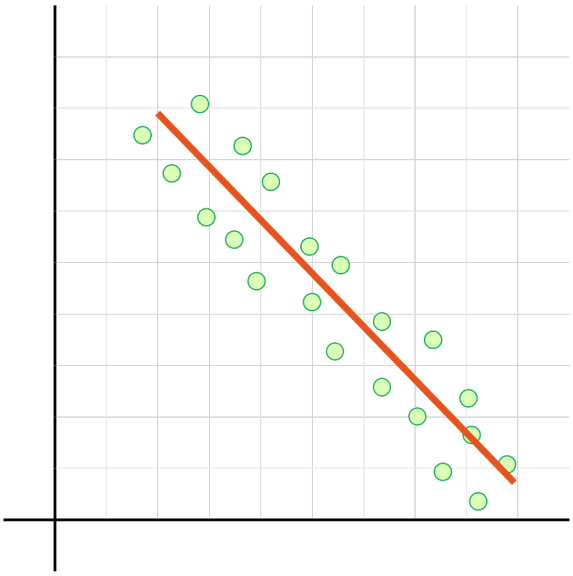
Al igual que las regresiones vistas anteriormente, los algoritmos SVM basan sus predicciones en el aprendizaje de un conjunto de datos de entrenamiento que toma como input. El SVM es un modelo que representa a los puntos de entrenamiento en el espacio vectorial n (número de variables) separando a las clases de dos o más categoría diferentes mediante un hiperplano definido por el vector entre los puntos que están más cercanos a las clases que se quieran separar.

Para clarificar el funcionamiento matemático de estos algoritmos se pondrá el siguiente ejemplo:

Sea un conjunto de datos como el que se presenta a continuación, donde se puede observar que los datos presentan un comportamiento similar a un modelo lineal.



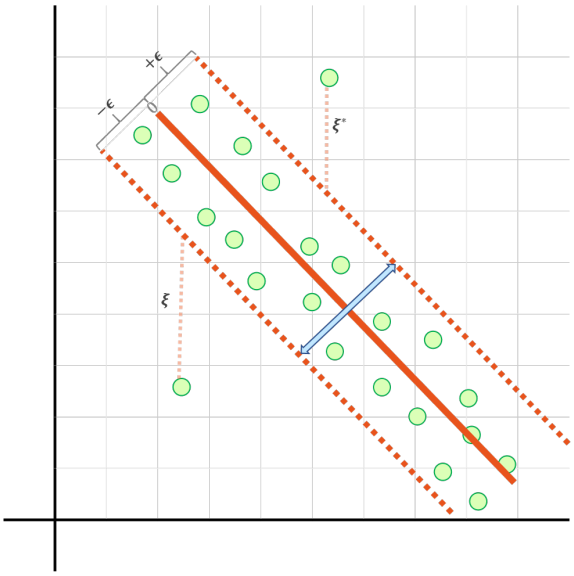
En primer lugar, se busca el modelo lineal que mejor describa el comportamiento del conjunto de puntos mostrado. En este caso es una línea recta, a la que se llama hiperplano. Sin embargo, no tiene por qué ser una recta, ya que la distrubución de datos puede ser mucho más compleja y heterogénea que la mostrada. Aún así, la idea principal siguie siendo la misma.



De esta forma, en este caso el hiperplano se podrá definir mediante la expresión de una recta:

$$y = \beta_0 + \beta_1 x$$

A continuación, se buscan dos bandas paralelas al hiperplano, siendo necesario que entre ambas se cubran la mayor cantidad de datos. Estas bandas son conocidas como vectores de apoyo o soporte.



Estas rectas paralelas a la anterior tendrán las siguientes expresiones:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

$$y = \beta_0 + \beta_1 x - \varepsilon$$

De esta forma, mediante la generación de este hiperplano, los algoritmos de Support Vector Machine son capaces de solucionar problemas de clasificación y regresión. Sin embargo, la base matemática que se encuentra detrás de esta algortimo no se expondrá en esta memoria puesto que es algo que se escapa de la misma.

Algunas ventajas de los algoritmos de SVM son:

- Gran efectividad en espacios con muchas dimensiones.
- Pueden ser aún efectivos en casos en los que el número de dimensiones es mayor que el número de muestras.
- Usan un subset de datos de entrenamiento en el vector de soporte, por lo que son también eficientes en cuanto a memoria.
- Son muy versátiles, pueden ser aplicados con diferentes kernel.

5.4.1 Kernel

A la hora de crear un modelo basado en SVM, es esencial escoger el mejor kernel posible. Dicho de una forma muy simple, el kernel es la función en la cuál se basa el SVM para adecuarse de la mejor forma posible a los datos de partida.

Existen muchos tipos de kernel. Algunos de los más utilizados son el kernel lineal, el kernel polinomial y el kernel rbf. Serán éstos los que se implementen en los modelos de SVM de este estudio. Los dos primeros casos se corresponden a un hiperparámetro basado en una recta y en una función polinomial respectivamente, mientras que el kernel rbf está basado en una función exponencial.

A continuación se adjuntan fotos de ejemplos de cada uno de estos kernel para un caso de dos dimensiones:

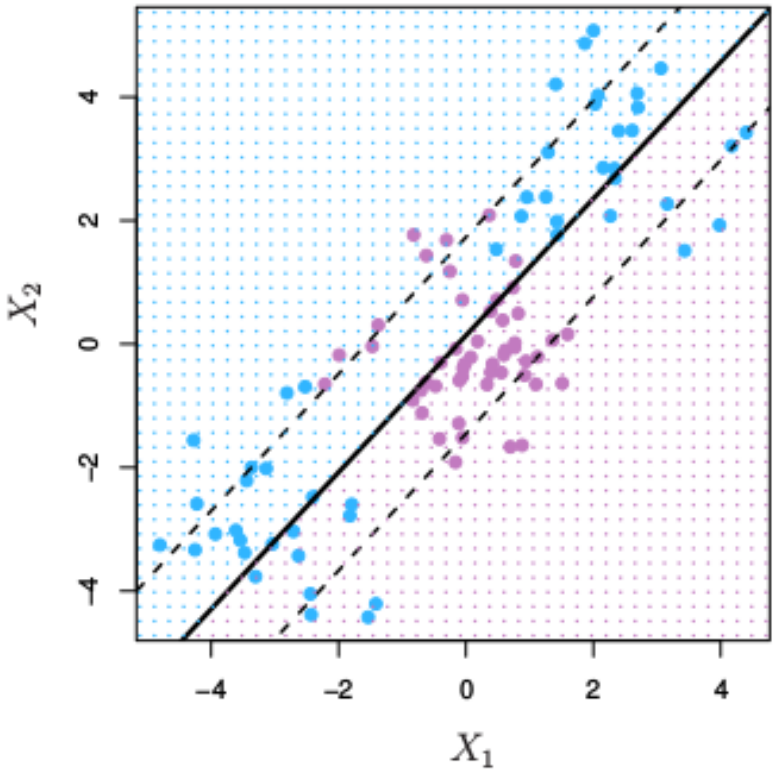


Figura 5.2 Ejemplo Kernel lineal.

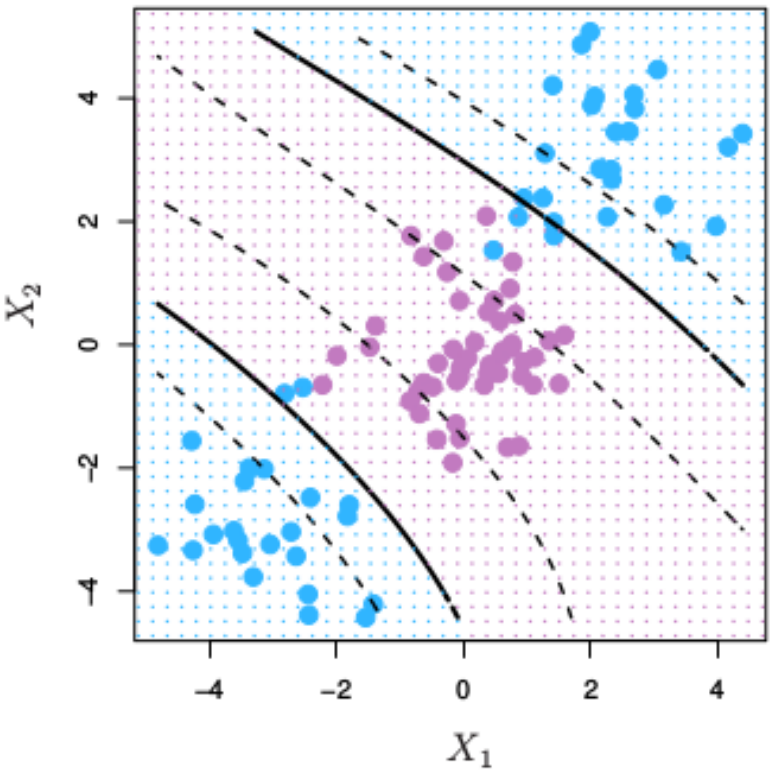


Figura 5.3 Ejemplo Kernel polinomial.

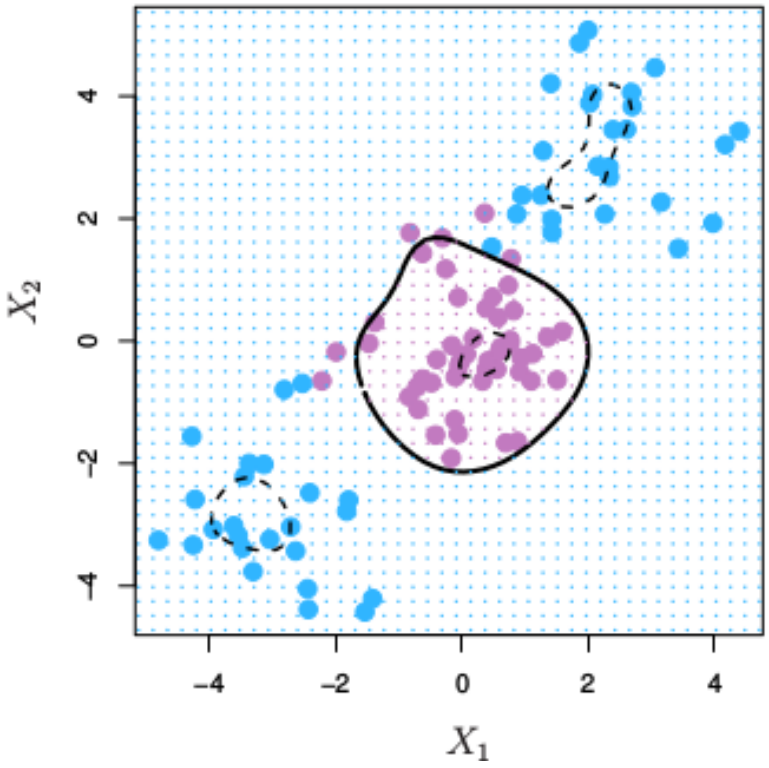


Figura 5.4 Ejemplo Kernel rbf.

5.5 Redes neuronales

Los modelos basados en Redes Neuronales son los modelos que más se aproximan a la forma en la que funciona el cerebro humano. Al igual que éste, la unidad fundamental de estos modelos es la neurona.

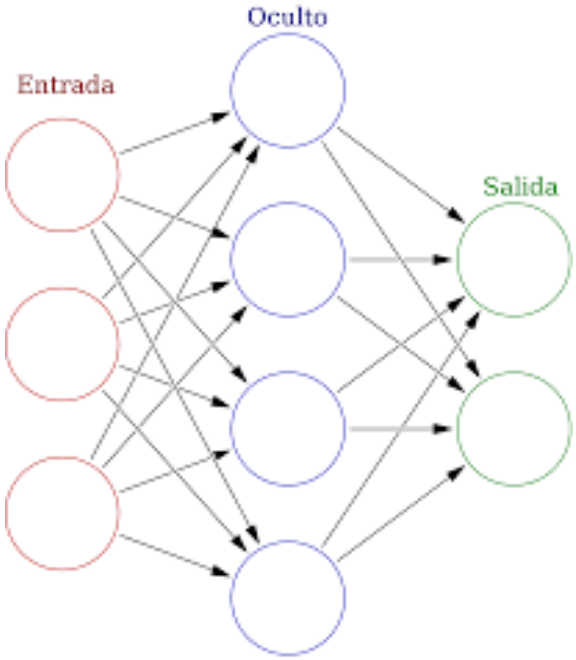


Figura 5.5 Ejemplo de esquema simplificado de una red neuronal.

Como se aprecia en la imagen anterior, una red neuronal consiste en muchas neuronas unidas de forma que, en su conjunto, son capaces de procesar cierta información que le llega de entrada (input) y obtener, a partir de ésta, otra información a la salida (output). Cuantas más capas de neuronas haya en la red neuronal, más eficiente y compleja será ésta.

En el caso de este estudio, los modelos basados en redes neuronales se utilizarán dentro del campo de aprendizaje supervisado, es decir, la información de entrada que se le de al modelo será en todo momento provista de forma que se es consciente de sus características. Sin embargo, las redes neuronales también pueden ser utilizadas para casos de aprendizaje no supervisado.

Al igual que en los casos anteriores, no se profundizará más en el funcionamiento de las redes neuronales por ser éste muy complejo y por no ser el objetivo de este estudio.

6 Implementación de los modelos

6.1 Modelo predictivo: regresión lineal y polinomial simple

Tanto para el caso del despegue y ascenso como para el crucero, se dividirá el set de datos en datos de entrenamiento (66 %) y datos de testing (33 %) con el objetivo de poder comprobar de una forma más eficiente la eficacia del modelo.

Es importante establecer estos dos conjuntos de datos porque un R^2 muy alto para un grado n alto podría llevar también a un modelo pésimo debido a lo que se llama overfitting. Cuando se da este fenómeno, la función se adapta muy bien a los datos de partida pero puede tener errores abismales a la hora de predecir los valores objetivos.

El caso contrario sería el underfitting, en el que el n es demasiado bajo y la matriz ni si quiera se ajusta correctamente a los datos de partida.

De esta forma, el criterio que se tomará para decidir si el modelo de una variable conviene ser estudiado es el siguiente:

- El R^2 del conjunto de training y del conjunto de testing debe ser mayor que 0.6.
- Si, conforme aumenta el n, el valor de R^2 no aumenta o aumenta de muy poco (del orden de 0.01), se tomará el valor de n para el cual el R^2 deja de aumentar considerablemente.

No obstante, se harán excepciones basadas en otras métricas como el error absoluto, ya que podría darse que un modelo sea eficiente pero que no cumpla que su R^2 sea mayor que la referencia.

6.1.1 Despegue y ascenso

A continuación se adjunta un cuadro en el que se especifican los valores de R^2 para los distintos modelos:

Tabla 6.1 R^2 para los modelos de regresión simple (despegue y ascenso).

		Lineal (n=1)	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**
Wind speed	R^2 training	0.133	0.136	0.138	0.138	0.141	0.147	0.147	0.148
	R^2 testing	0.138	0.141	0.144	0.144	0.147	0.154	0.154	0.154
Pitch	R^2 training	0.349	0.352	0.382	0.394	0.397	0.398	0.400	0.400
	R^2 testing	0.341	0.344	0.370	0.381	0.382	0.382	0.382	0.382
Altitude	R^2 training	0.091	0.092	0.170	0.189	0.168	0.129	0.099	0.100
	R^2 testing	0.088	0.090	0.161	0.177	0.158	0.125	0.099	0.099
CL	R^2 training	0.586	0.636	**0.651**	**0.666**	**0.666**	**0.667**	**0.669**	**0.670**
	R^2 testing	0.566	0.592	**0.632**	**0.650**	**0.651**	**0.653**	**0.656**	**0.658**
Wind direction	R^2 training	0.074	0.118	0.129	0.131	0.146	0.148	0.147	0.107
	R^2 testing	0.084	0.125	0.135	0136	0.149	0.150	0.149	0.113

Siguiendo los criterios establecidos en esta misma sección, el único modelo que merece la pena estudiar sería un modelo de regresión polinomial simple de grado 3 con el CL como variable independiente.

Para este modelo se obtienen los siguientes errores:

Tabla 6.2 Errores del modelo de regresión polinómica simple de grado 3 (CL).

Conjunto	R^2	Error absoluto medio	Error cuadrático medio
Training	0.651	0.832	1.130
Testing	0.632	0.829	1.115

La curva de regresión que se obtiene para este modelo (graficada junto a los datos de testing en naranja y training en azul) es la siguiente:

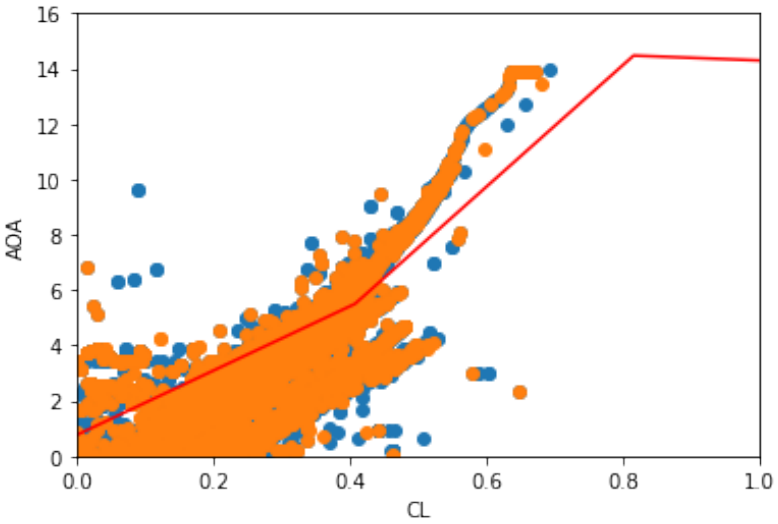


Figura 6.1 Regresión simple polinómica grado 3 (CL) .

6.1.2 Crucero

A continuación se adjunta un cuadro en el que se especifican los valores de R^2 para los distintos modelos:

Tabla 6.3 R^2 para los modelos de regresión simple (crucero).

		Lineal (n=1)	**n=2**	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**
Wind speed	R2 training	**0.698**	**0.707**	**0.712**	**0.713**	**0.713**	**0.713**	**0.714**	**0.716**
	R2 testing	**0.701**	**0.709**	**0.715**	**0.716**	**0.716**	**0.716**	**0.717**	**0.720**
Pitch	R2 training	**0.907**	**0.940**	**0.951**	**0.953**	**0.956**	**0.956**	**0.956**	**0.957**
	R2 testing	**0.912**	**0.948**	**0.956**	**0.958**	**0.959**	**0.959**	**0.960**	**0.960**
Mach	R2 training	0.547	0.558	**0.696**	**0.710**	**0.759**	**0.759**	**0.759**	**0.772**
	R2 testing	0.543	0.553	**0.697**	**0.708**	**0.757**	**0.757**	**0.757**	**0.770**
Altitude	R2 training	0.493	0.541	0.588	0.594	0.599	**0.604**	**0.608**	**0.613**
	R2 testing	0.493	0.544	0.588	0.593	0.598	**0.603**	**0.607**	**0.611**
CL	R2 training	**0.876**	**0.881**	**0.884**	**0.886**	**0.889**	**0.889**	**0.894**	**0.897**
	R2 testing	**0.875**	**0.880**	**0.883**	**0.884**	**0.888**	**0.888**	**0.894**	**0.896**
CD	R2 training	0.181	0.362	0.363	0.364	0.424	0.461	0.461	0.500
	R2 testing	0.172	0.339	0.327	0.338	0.288	0.113	-0.080	-47.6
Total thrust	R2 training	0.393	0.396	0.429	0.429	0.431	0.432	0.432	0.432
	R2 testing	0.400	0.400	0.434	0.434	0.436	0.437	0.437	0.437
Wind direction	R2 training	0.270	0.482	0.594	0.595	0.595	**0.605**	0.506	0.508
	R2 testing	0.269	0.480	0.595	0.596	0.596	**0.606**	0.509	0.510
Elevator surface	R2 training	0.390	0.414	0.497	0.498	0.511	0.517	0.534	0.535
	R2 testing	0.395	0.444	0.475	0.472	0.492	0.190	-164	-7150
Roll	R2 training	0.061	0.068	0.068	0.076	0.078	0.078	0.078	0.079
	R2 testing	0.056	0.064	0.065	0.072	0.074	0.074	0.074	0.074
rpm media	R2 training	0.066	0.102	0.207	0.213	0.212	0.216	0.226	0.240
	R2 testing	0.069	0.102	0.217	0.230	0.227	0.228	0.236	0.249

Siguiendo los criterios establecidos en esta misma sección, los modelos que conviene estudiar son:

- Regresión polinomial simple de grado 3 con la variable Wind speed.
- Regresión polinomial simple de grado 2 con la variable Pitch.
- Regresión polinomial simple de grado 5 con la variable Mach.
- Regresión polinomial simple de grado 6 con la variable Altitude.
- Regresión lineal simple con la variable CL.
- Regresión polinomial simple de grado 6 con la variable Wind direction.

Sin embargo, por no incluir demasiados modelos al final y no hacer el documento demasiado copioso, solamente se estudiarán los modelos del CL y del Pitch, por ser aquellos que tienen un valor de R^2 más alto.

Estos modelos tienen los siguientes valores de métricas de error:

Tabla 6.4 Errores del modelo de regresión polinómica simple de grado 2 (Pitch).

Conjunto	R^2	Error absoluto medio	Error cuadrático medio
Training	0.940	0.191	0.104
Testing	0.948	0.192	0.101

Tabla 6.5 Errores del modelo de regresión lineal simple (CL).

Conjunto	R^2	Error absoluto medio	Error cuadrático medio
Training	0.876	0.332	0.222
Testing	0.875	0.335	0.227

Las curvas de regresión obtenidas son las siguientes (en naranja se representa el conjunto de testing y en azul el de training):

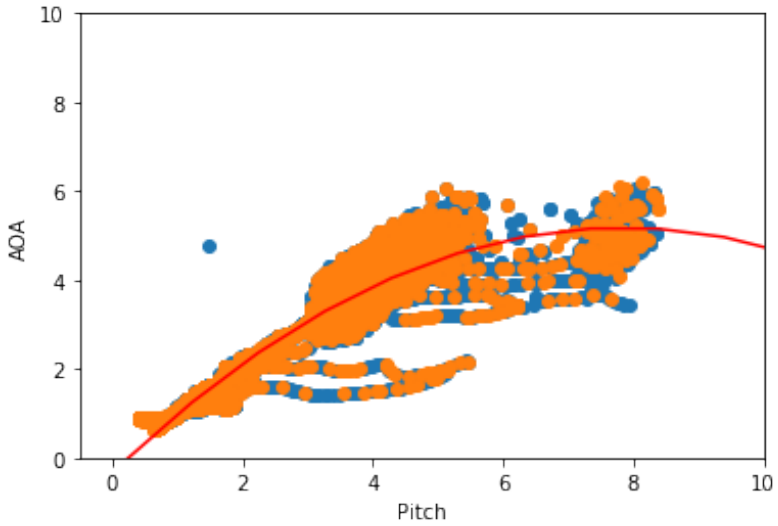


Figura 6.2 Regresión simple polinómica grado 2 (Pitch) .

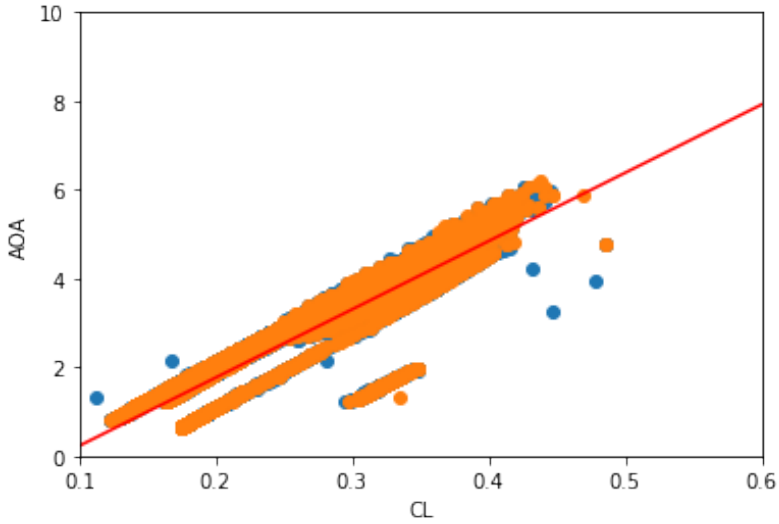


Figura 6.3 Regresión lineal simple (CL) .

6.2 Modelo predictivo: regresión lineal múltiple

En esta sección se pretende proponer un modelo de regresión lineal múltiple con las variables que lo hagan lo más eficiente posible. Esto no es tarea fácil, puesto que hay que tener en cuenta que la variable independiente

tenga una alta repercusión en el comportamiento de la variable independiente pero que no tenga una alta correlación con otras variables independientes que se incluyan en el modelo.

En primer lugar se implementará un modelo en el que, para elegir qué variables independientes introducir, se utilizarán los modelos de selección de características de los apartados anteriores.

Por otro lado, como se ha nombrado previamente, hay que tener en cuenta la colinealidad entre las variables independientes. Puesto que, si una variable independiente es muy eficiente a la hora de establecer un modelo de regresión lineal pero también tiene una alta dependencia con otra variable independiente, el modelo de regresión múltiple se verá negativamente afectado.

De esta forma, es conveniente estudiar la colinealidad de las variables independientes. Para ello se utiliza el factor inflación de la varianza (VIF). La referencia que se suele utilizar para saber si se deben incluir las variables en el modelo o no es la siguiente:

- $VIF = 1$: Las variables no están correlacionadas
- $VIF < 5$: Las variables tienen una correlación moderada y se pueden quedar en el modelo
- $VIF > 5$: Las variables están altamente correlacionadas y deben desaparecer del modelo

Este factor se estudiará (aunque no se incluya explícitamente) cada vez que se implemente un modelo multivariable.

6.2.1 Despegue y ascenso

Implementación del modelo con la selección de características

Las variables que se eligen según los modelos de selección de características previamente implementados (en concreto SelectKBest) son las siguientes:

- Mach
- Wind Speed
- Pitch
- Altitude
- rpm media
- Wind Direction

Dichas variables se han nombrado ordenadas de más a menos eficientes. A continuación se adjunta una tabla en la que se pueden observar los valores de VIF para el caso de dichas variables independientes con el ángulo de ataque como variable independiente:

Tabla 6.6 Valores de VIF.

Variables	VIF
Mach	4.33
Wind Speed	2.35
Pitch	1.75
Altitude	2.54
rpm media	3.34
Wind Direction	2.34

Se puede observar que, según la referencia, todas las variables se podrían incluir en el modelo sin que afecte la colinealidad. Es decir, se podría crear un modelo de regresión lineal múltiple con estas 6 variables sin que el efecto de la colinealidad fuese un problema. Sin embargo, el hecho de que no haya altos valores de VIF en este caso no quiere decir que no los vaya a haber en modelos de regresión en los que se elimine alguna de las variables anteriores del input. Es decir, si, por ejemplo, se crease un modelo de regresión lineal múltiple de 3 variables con el Mach, el Wind Speed y el Wind Direction, al calcular de nuevo los valores de VIF, podría aparecer alguno que superase la referencia.

Por tanto, cada vez que se implemente un modelo multivariable, se comprobará que los valores de VIF no superen la referencia.

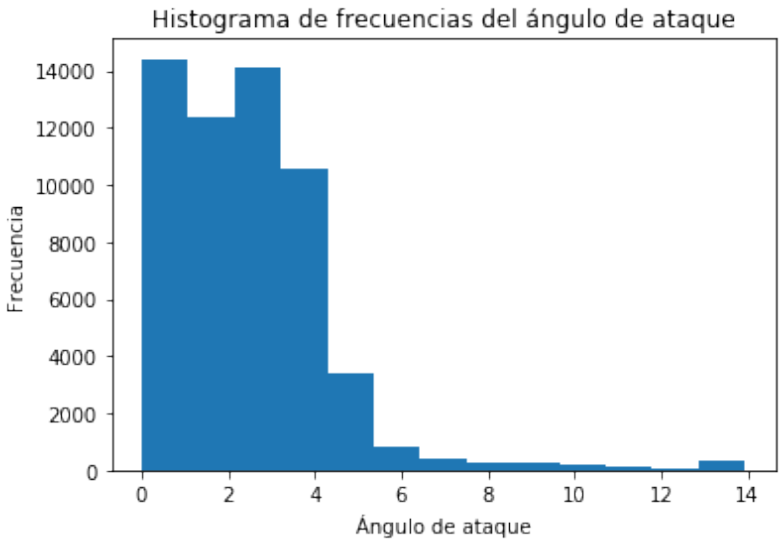
Incorporación de la variable CL al modelo

El sistema de selección de características no ha tomado a la variable CL como buena para el modelo. Sin embargo, se sabe que el CL tiene una relación teóricamente lineal con el ángulo de ataque (excepto cuando el avión entra en pérdida). Por tanto, por intuición, se podría pensar que dicha variable podría mejorar la eficiencia del modelo.

De esta forma, se añadirá dicha variable, ya que, en caso de tener alta dependencia con otra variable de input, el estudio del VIF servirá para detectarlo y eliminarla de cualquier modelo.

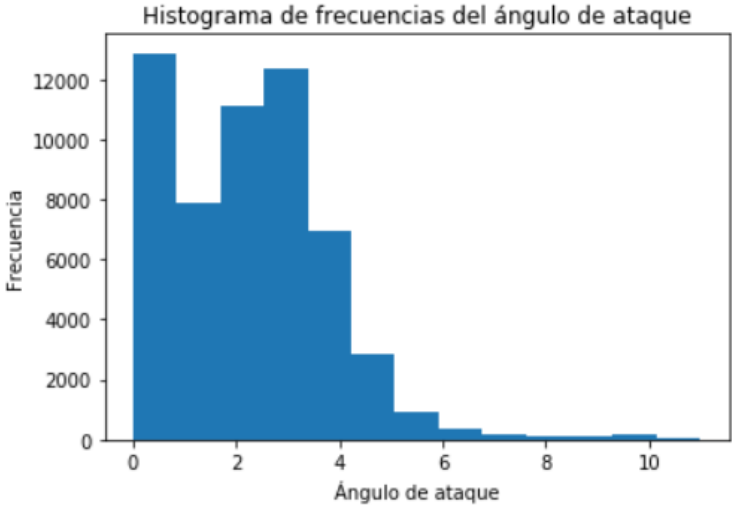
Mejora del modelo mediante la filtración de datos

Tal y como se mostró en la sección de visualización de datos, el histograma de frecuencias del ángulo de ataque para el caso de despegue y ascenso es el siguiente:



Se puede observar que dicho diagrama incluye ángulos de ataque demasiado grandes (un ángulo de ataque de 10º es un ángulo cercano al valor de entrada en pérdida). También se aprecia que la frecuencia con la que aparecen dichos valores es muy baja en comparación con la frecuencia de ángulos de ataque muy bajos.

Por estas dos razones (ángulo demasiado grande y frecuencia demasiado pequeña) se decide hacer un filtrado de los datos de forma que el modelo solamente incluya ángulos de ataque menores a 11º, de forma que el histograma queda de la siguiente forma:



Cabe destacar, que se ha realizado esta filtración porque se ha comprobado que los modelos mejoran con ella. Por esta misma razón, de ahora en adelante todos los modelos para el despegue y ascenso se implementarán con esta filtración de datos.

Resultados

A continuación, se adjuntan los resultados obtenidos. El procedimiento que se ha seguido ha consistido en:

1. Tomar los datos correspondientes a las 7 variables nombradas anteriormente y a la variable dependiente (ángulo de ataque).
2. Implementar todas las regresiones lineales múltiples posibles (todas las combinaciones de variables independientes) de 3 a 7 variables. Esto es:
   - 21 regresiones lineales con 2 variables
   - 35 regresiones lineales con 3 variables
   - 35 regresiones lineales con 4 variables
   - 21 regresiones lineales con 5 variables
   - 7 regresiones lineales con 6 variables
   - 1 regresión lineal con 7 variables
3. De todos los modelos anteriores se han cogido los 3 mejores que no presenten valores de VIF superiores a los de referencia.
4. Los resultados arrojados por dichos modelos se presentan a continuación.

Tabla 6.7 Métricas de error de los 3 mejores modelos de regresión lineal múltiple (despegue y ascenso).

Número de variables	Variables	R^2	Error absoluto medio	Error cuadrático medio
5	Mach, Wind Speed, altitude, rpm media, CL	0.67	0.66	0.85
4	Mach, Wind Speed, rpm media, CL	0.67	0.67	0.85
6	Mach, Wind Speed, pitch, altitude, Wind Direction, CL	0.68	0.62	0.82

Tabla 6.8 Valores de VIF para modelo de 5 variables.

Variables	VIF
Mach	4.21
Wind Speed	1.07
Altitude	2.55
CL	1.31
rpm media	2.99

Tabla 6.9 Valores de VIF para modelo de 4 variables.

Variables	VIF
Mach	2.75
Wind Speed	1.06
CL	1.24
rpm media	2.97

Tabla 6.10 Valores de VIF para modelo de 6 variables.

Variables	VIF
Mach	3.24
Wind Speed	2.44
Pitch	2.10
CL	1.85
Altitude	2.90
Wind Direction	2.52

6.2.2 Crucero

Implementación del modelo con la selección de características

Al igual que en el caso del despegue y ascenso, se utiliza la función de Scikit-Learn SelectKbest, de forma que se obtienen las siguientes variables:

- Mach
- Wind Speed
- Pitch
- Altitude
- CL
- Empuje total

De nuevo, será necesario comprobar la colinealidad de estas variables con la variable AOA como variable dependiente. Al calcular el VIF, se obtienen los siguientes resultados:

Tabla 6.11 Valores de VIF.

Variables	VIF
Mach	12.24
Wind Speed	4.36
hline Pitch	5.65
Altitude	5.99
CL	**25.30**
Empuje total	2.26

Se obervan varios valores de VIF que superan al valor establecido como referencia. En este caso, se procede a eliminar del modelo a la variable cuyo VIF es más alto (CL), puesto que no conviene a la hora de implementar un modelo de regresión lineal múltiple con 6 variables.

Una vez eliminada, se calculan de nuevo los valores de VIF, de forma que se obtiene lo siguiente:

Tabla 6.12 Valores de VIF.

Variables	VIF
Mach	2.73
Wind Speed	4.06
Pitch	5.16
Altitude	3.35
Empuje total	2.25

En este caso se observa que solamente la variable Pitch supera a la referencia por muy poco. Sin embargo, debido a que dicha variable tiene una correlación muy alta con la variable independiente y el valor del VIF es prácticamente el de referencia, se decide dejar a la variable en el modelo, puesto que por intuición se puede pensar que es más eficiente dejarla que eliminarla.

De esta forma, sería posible la implementación de un modelo con las 5 variables anteriores. Al igual que en el apartado anterior, se seguirá el procedimiento siguiente:

1. Tomar los datos correspondientes a las 6 variables nombradas anteriormente y a la variable dependiente (ángulo de ataque).
2. Implementar todas las regresiones lineales múltiples posibles (todas las combinaciones de variables independientes) de 3 a 6 variables. Esto es:
   - 15 regresiones lineales con 2 variables
   - 20 regresiones lineales con 3 variables
   - 15 regresiones lineales con 4 variables
   - 6 regresiones lineales con 5 variables
   - 1 regresión lineal con 6 variables
3. De todos los modelos anteriores se han cogido los 3 mejores que no presenten valores de VIF superiores a los de referencia.
4. Los resultados arrojados por dichos modelos se presentan a continuación.

Tabla 6.13 Métricas de error de los 3 mejores modelos de regresión lineal múltiple (crucero).

Número de variables	Variables	R^2	Error absoluto medio	Error cuadrático medio
2	Mach, pitch	0.91	0.23	0.16
2	Mach, altitude	0.82	0.47	0.32
3	Mach, pitch, altitude	0.93	0.26	0.13

Tabla 6.14 Valores de VIF para modelo de 2 variables.

Variables	VIF
Mach	2.20
Pitch	2.20

Tabla 6.15 Valores de VIF para modelo de 2 variables.

Variables	VIF
Mach	1.07
Altitude	1.07

Tabla 6.16 Valores de VIF para modelo de 3 variables.

Variables	VIF
Mach	2.71
Pitch	2.17
Altitude	4.44

En este caso se observa que no es necesario implementar modelos con muchas variables para optimizar el error.

6.3 Modelo predictivo: regresión simple con técnicas de Support Vector Machine

6.3.1 Despegue y ascenso

Puesto que en apartados anteriores se ha comprobado que el CL es la mejor variable a la hora de implementar un modelo para el despegue y el asecenso, en el caso de la regresión simple con SVM solamente se implementarán modelos con esta variable. Los modelos que se han implementado son SVM con kernel lineal, SVM con kernel rbf, SVM con kernel polinomial de grado 2 y SVM con kernel polinomial de grado 3.

Los resultados obtenidos son los siguientes:

Tabla 6.17 Métricas de error para SVM (CL).

Modelo	R^2	Error absoluto medio	Error cuadrático medio
Lineal	0.515	0.722	1.256
rbf	0.520	0.685	1.244
Polinomial grado 2	0.531	0.771	1.215
Polinomial grado 3	0.490	0.877	1.321

Se observa que ningún modelo presenta un R^2 mayor que 0.6. Sin embargo, se optará por tomar el modelo que presenta menor error absoluto (kernel = rbf) y estudiarlo en el apartado posterior por hacer una comparación entre éste y el modelo de regresión simple.

Se observa que, de los cuatro modelos, el que obtiene menos error absoluto medio es el rbf, pero aún así no es un buen valor como para considerarlo como modelo solución.

A continuación se adjuntan las gráficas en las que se puede observar en naranja los valores reales y en azul la predicción:

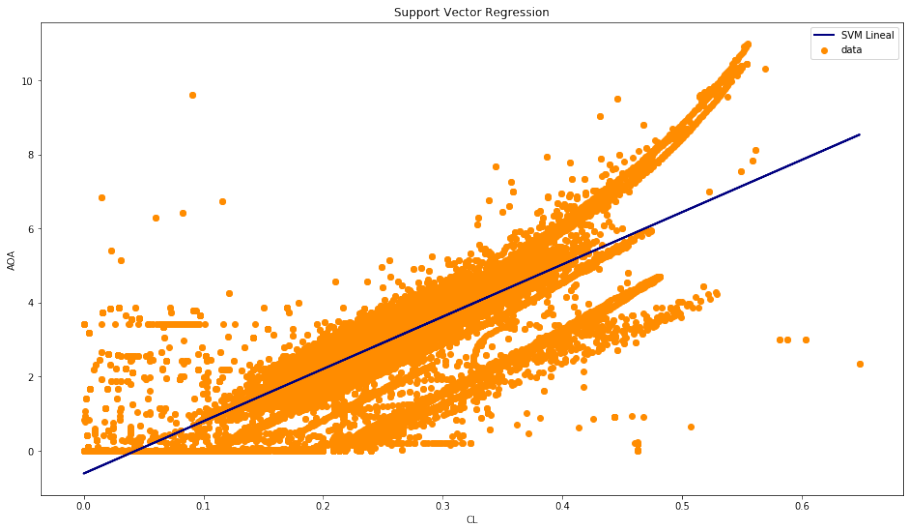


Figura 6.4 SVM kernel lineal.

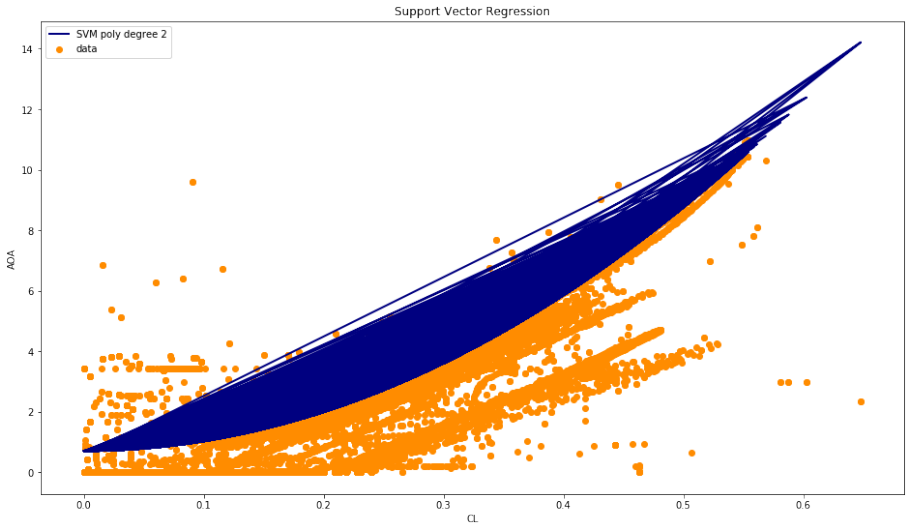


Figura 6.5 SVM kernel rbf.

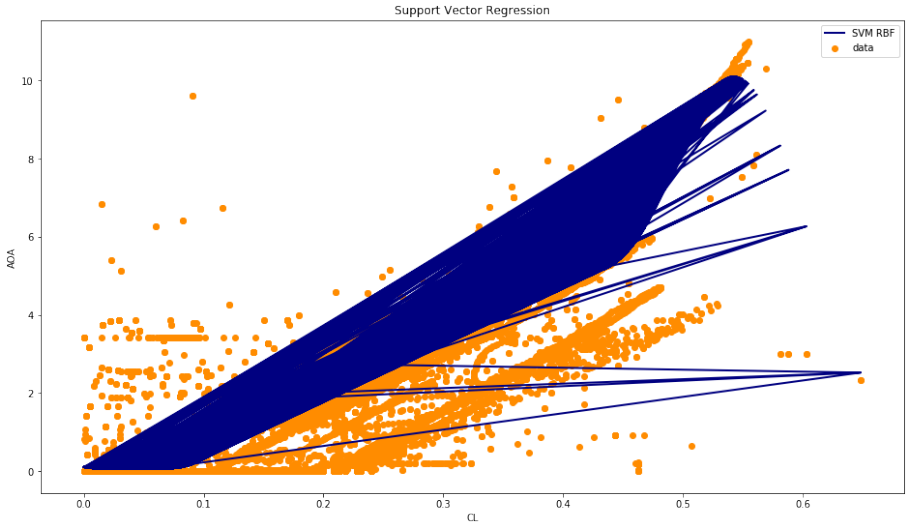


Figura 6.6 SVM kernel polinomial grado 2.

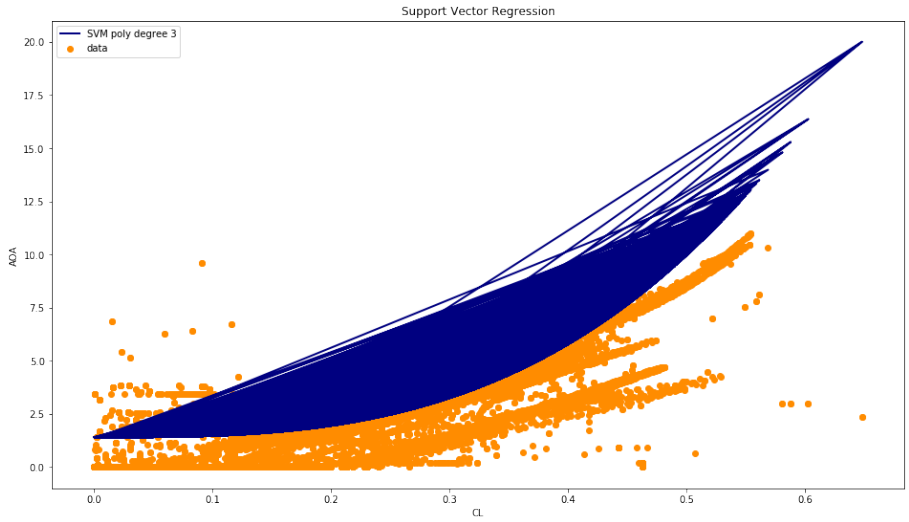


Figura 6.7 SVM kernel polinomial grado 3.

6.3.2 Crucero

En el caso del crucero, los modelos que se implementarán serán con las variables CL y Pitch. Es posible que haya más variables de las cuales se pueda obtener un buen resultado para un modelo simple. Sin embargo, puesto que los estudios anteriores han mostrado que el CL y el pitch son las variables de las que se obtienen mejores resultados, sólo se utilizarán estas dos, ya que posteriormente, en el modelo de regresión múltiple con SVM, se introducirán más variables.

Tabla 6.18 Métricas de error para SVM simple (CL).

Modelo	R^2	Error absoluto medio	Error cuadrático medio
Lineal	0.86	0.30	0.24
rbf	0.88	0.27	0.21
Polinomial grado 2	0.87	0.30	0.23
Polinomial grado 3	0.85	0.36	0.27

Tabla 6.19 Métricas de error para SVM simple (pitch).

Modelo	R^2	Error absoluto medio	Error cuadrático medio
Lineal	0.90	0.20	0.18
rbf	0.96	0.73	0.16
Polinomial grado 2	0.56	0.37	0.79
Polinomial grado 3	-0.46	0.59	2.61

En este caso los dos mejores modelos (que serán los que se estudiarán) son el SVM con kernel rbf para CL y el SVM con kernel lineal para el pitch.

A continuación se adjuntan las gráficas para crucero:

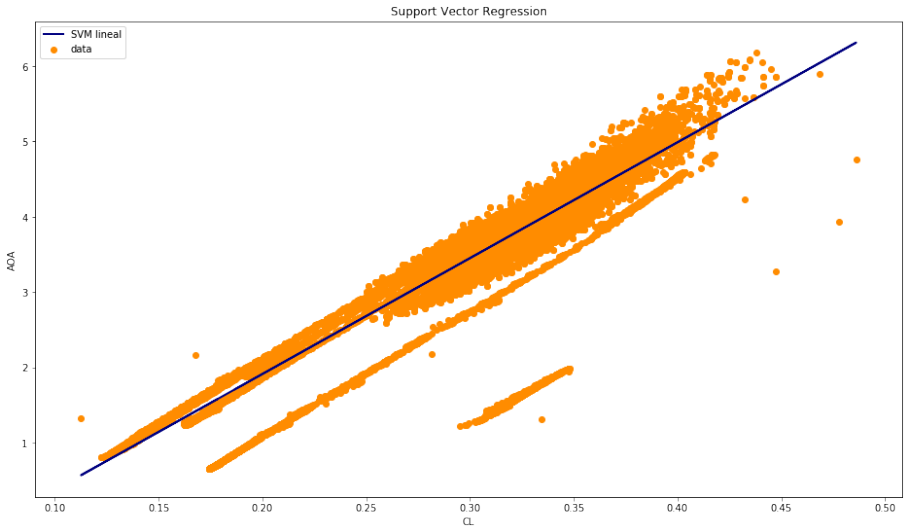


Figura 6.8 SVM kernel lineal (CL).

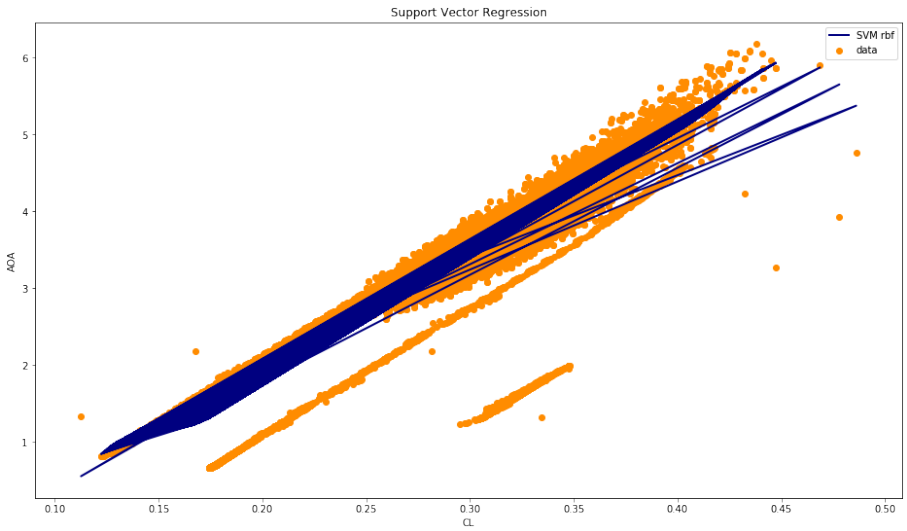


Figura 6.9 SVM kernel rbf (CL).

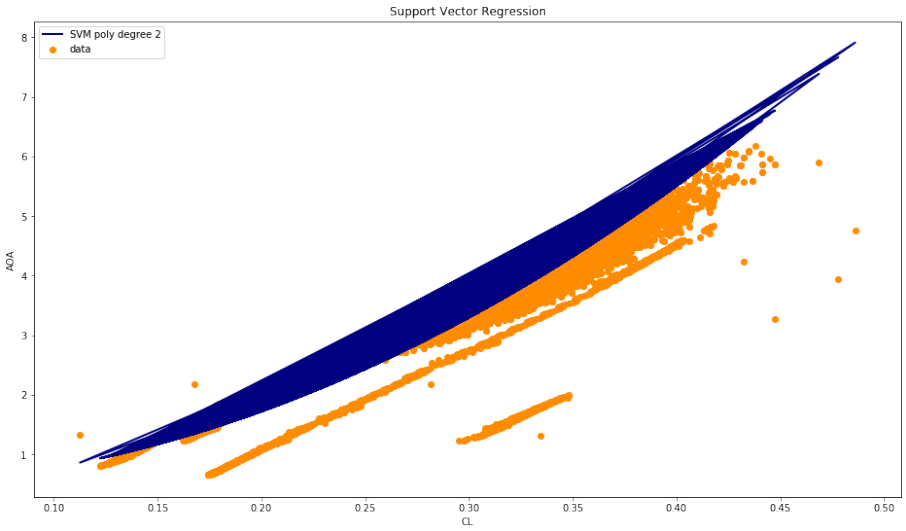


Figura 6.10 SVM kernel polinomial grado 3 (CL).

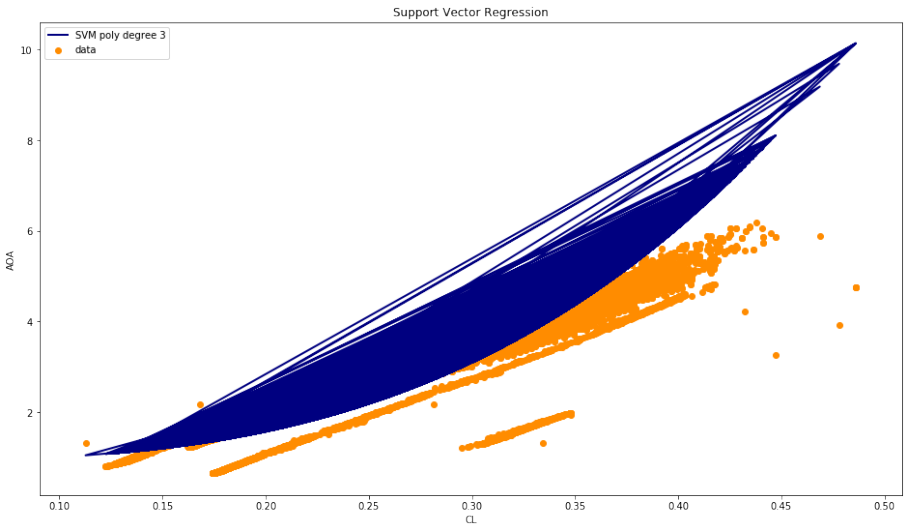


Figura 6.11 SVM kernel polinomial grado 3 (CL).

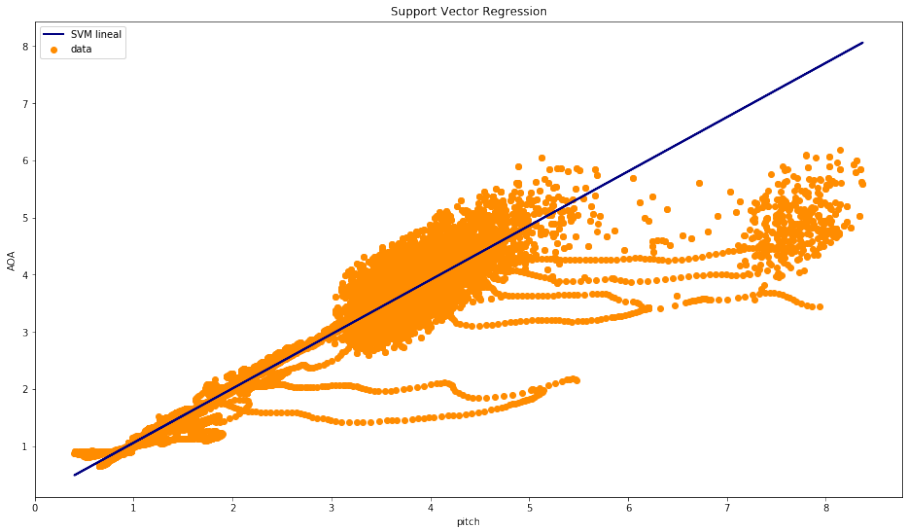


Figura 6.12 SVM kernel lineal (pitch).

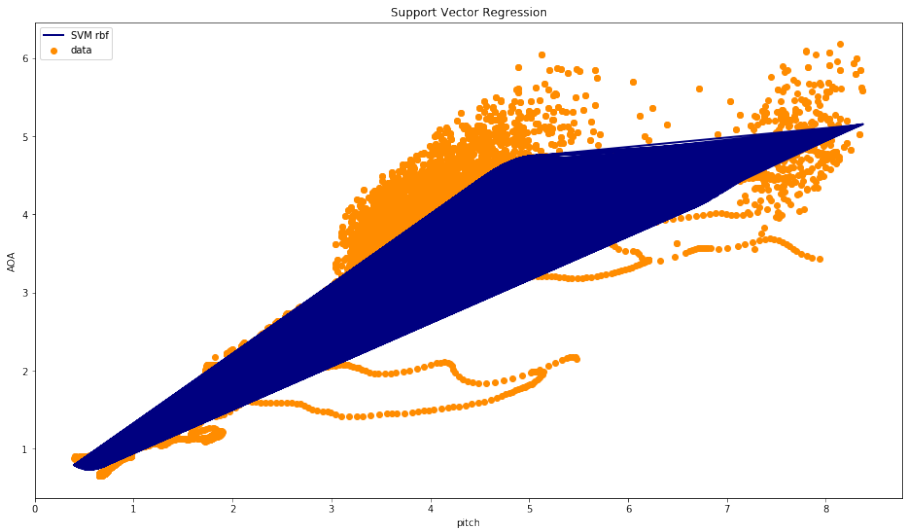


Figura 6.13 SVM kernel rbf (pitch).

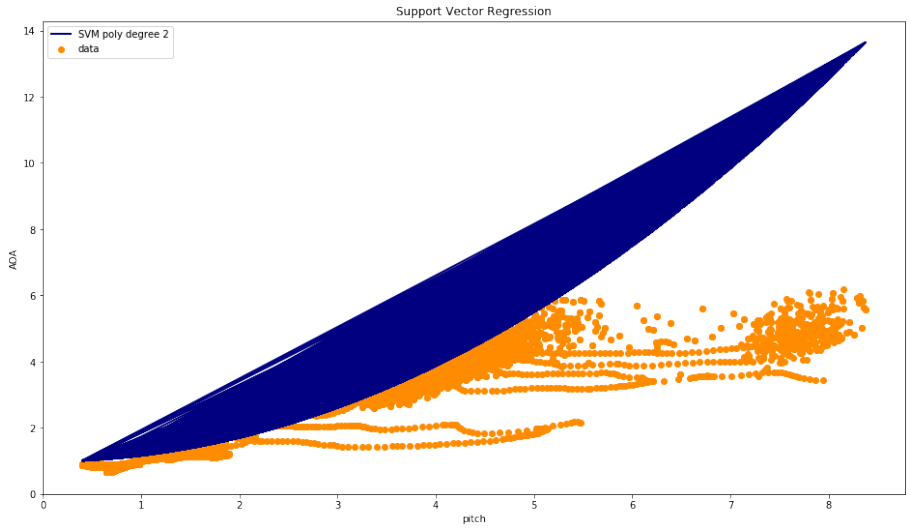


Figura 6.14 SVM kernel polinomial grado 2 (pitch).

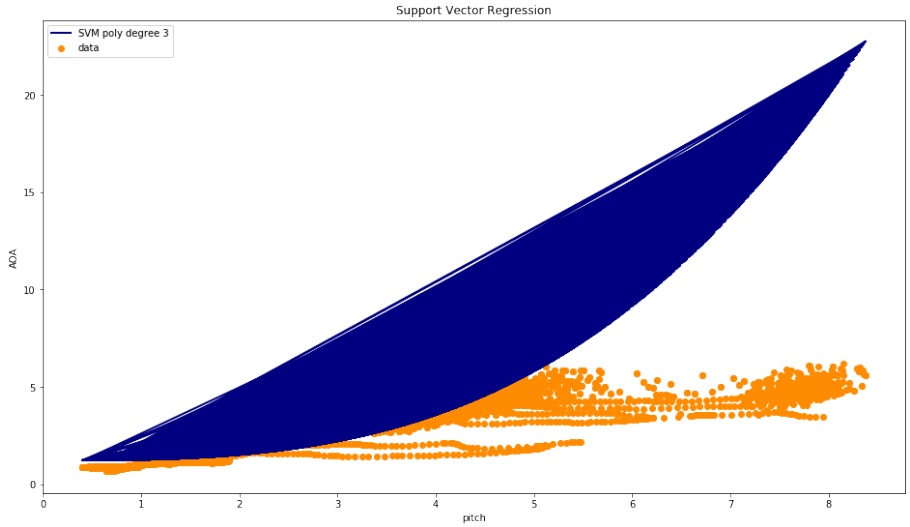


Figura 6.15 SVM kernel polinomial grado 3 (pitch).

Tabla 6.20

Nota:
Puesto que en los modelos anteriores se han obtenido muy buenos resultados para el caso del crucero. A partir de ahora sólo se implementarán modelos para el caso del despegue y aterrizaje, con el objetivo de mejorar los resultados de este tramo.

6.4 Modelo predictivo: regresión lineal múltiple con técnicas de Support Vector Machine

6.4.1 Despegue y ascenso

En este apartado se procederá de forma análoga al apartado de regresión lineal múltiple. Por lo que se ahorrarán explicaciones redundantes.

Recordemos que las variables escogidas por el algoritmo de selección de características son:

- Mach
- Wind Speed
- Pitch
- Altitude
- rpm media
- Wind Direction

Para este modelo solamente se utilizarán kernel rbf Y lineal, ya que han demostrado ser los más eficientes. Además los polinómicos tienen un coste computacional mucho mayor.

El procedimiento que se seguirá es el siguiente:

1. Tomar los datos correspondientes a las 7 variables nombradas anteriormente y a la variable dependiente (ángulo de ataque).
2. Implementar todas las regresiones múltiples con SVM de kernel rbf posibles (todas las combinaciones de variables independientes) de 3 a 7 variables. Esto es:
   - 21 regresiones con 2 variables con kernel rbf
   - 35 regresiones con 3 variables con kernel rbf
   - 35 regresiones con 4 variables con kernel rbf
   - 21 regresiones con 5 variables con kernel rbf
   - 7 regresiones con 6 variables con kernel rbf
   - 1 regresión con 7 variables con kernel rbf
   - 21 regresiones con 2 variables con kernel lineal
   - 35 regresiones con 3 variables con kernel lineal
   - 35 regresiones con 4 variables con kernel lineal
   - 21 regresiones con 5 variables con kernel lineal
   - 7 regresiones con 6 variables con kernel lineal
   - 1 regresión con 7 variables con kernel lineal
3. De todos los modelos anteriores se han cogido los 3 mejores que no presenten valores de VIF superiores a los de referencia.
4. Los resultados arrojados por dichos modelos se presentan a continuación.

Tabla 6.21 Métricas de error de los 3 mejores modelos de regresión múltiple con SVM.

Número de variables	Variables	kernel	R^2	Error absoluto medio	Error cuadrático medio
5	Mach, Wind Speed, Pitch, altitude, rpm media	lineal	0.5	0.71	1.30
4	Mach, Wind Speed, Pitch, altitude	lineal	0.67	0.61	0.85
5	Mach, Pitch, altitude, Wind Direction, CL	lineal	0.43	0.75	1.49

Aunque por las métricas de error, el primer modelo parece no ser el mejor, será éste el que se escoja, puesto que se ha comprobado que es el que mejor resultados arroja en el apartado de testing.

Los valores de VIF de cada uno de estos modelos se asjuntan a continuación:

Tabla 6.22 Valores de VIF para modelo de 5 variables.

Variables	VIF
Mach	4.16
Wind Speed	1.13
Pitch	1.75
Altitude	2.43
rpm media	3.33

Tabla 6.23 Valores de VIF para modelo de 4 variables.

Variables	VIF
Mach	2.82
Wind Speed	1.13
Pitch	1.46
Altitude	2.43

Tabla 6.24 Valores de VIF para modelo de 5 variables.

Variables	VIF
Mach	3.19
Pitch	2.09
Altitude	2.86
Wind Direction	1.16
CL	1.78

6.5 Modelo predictivo: regresión mediante Redes Neuronales

En el caso de las redes neuronales, se ha procedido de forma análoga al caso de SVM múltiple, siendo las variables seleccionadas por el algoritmo de selección de características las mismas que en los casos anteriores.

De esta forma el procedimiento es:

1. Tomar los datos correspondientes a las 7 variables y a la variable dependiente (ángulo de ataque).
2. Implementar todas las regresiones múltiples mediante redes neuronales (con unos parámetros fijos) posibles (todas las combinaciones de variables independientes) de 3 a 7 variables. Esto es:
   - 21 regresiones mediante redes neuronales con 2 variables
   - 35 regresiones mediante redes neuronales con 3 variables
   - 35 regresiones mediante redes neuronales con 4 variables
   - 21 regresiones mediante redes neuronales con 5 variables
   - 7 regresiones mediante redes neuronales con 6 variables
   - 1 regresión mediante redes neuronales con 7 variables
3. De todos los modelos anteriores se han cogido los 3 mejores que no presenten valores de VIF superiores a los de referencia.
4. Los resultados arrojados por dichos modelos se presentan a continuación.

Tabla 6.25 Métricas de error de los 3 mejores modelosn de regresión múltiple con redes neuronales.

Número de variables	Variables	R^2	Error absoluto medio	Error cuadrático medio
4	Mach, Wind Speed, Pitch, CL	0.92	0.27	0.20
3	Mach, Pitch, CL	0.87	0.37	0.34
3	Wind Speed, altitud, Wind Direction	-0.22	0.1.4	3.18

Por las métricas de error se puede observar que el primer modelo podría ser bastante eficiente. En cambio, el último no obtiene tan buenos valores. El hecho de que se haya incluido este último modelo en la tabla se debe a que se ha comprobado que dicho modelo obtiene un buen porcentaje de aciertos en la sección final. Sin embargo, el único que se considerará en la sección final será el primero, por ser el que mejor métricas presenta.

A continuación se adjuntan las tablas con los valores de VIF:

Tabla 6.26 Valores de VIF para modelo de 4 variables.

Variables	VIF
Wind Speed	1.11
Mach	1.34
Pitch	1.97
CL	1.57

Tabla 6.27 Valores de VIF para modelo de 3 variables.

Variables	VIF
Mach	1.33
Pitch	1.86
CL	1.57

Tabla 6.28 Valores de VIF para modelo de 3 variables.

Variables	VIF
Altitude	1.02
Wind Direction	2.15
Wind Speed	2.17

7 Testing de los mejores modelos

En esta sección lo que se hará es estudiar la eficiencia de los mejores modelos obtenidos. Para ello, se tomará el mejor modelo de cada uno de los algoritmos implementados (para crucero y despegue y ascenso) y se estudiará su precisión.

Para estudiar la precisión, se simulará un nuevo vuelo nunca antes introducido en los modelos anteriores. Los datos de este nuevo vuelo serán los datos para el testing, mientras que los datos de los 8 vuelos de los apartados anteriores serán los datos de training.

Para cada uno de los modelos seleccionados, se procederá de la siguiente forma:

- Se creará la curva de regresión con los datos de los 8 vuelos de los apartados anteriores.
- Se tomarán las variables independientes necesarias del nuevo vuelo simulado (variables de testing).
- Con el modelo de regresión se calcularán los ángulos de ataque estimados.
- Dichos ángulos de ataque se compararán con los ángulos de ataque reales obtenidos en el nuevo vuelo simulado.

A continuación, se hace una representación de algunas de las variables del nuevo vuelo, para tener una primera visualización de sus condiciones.

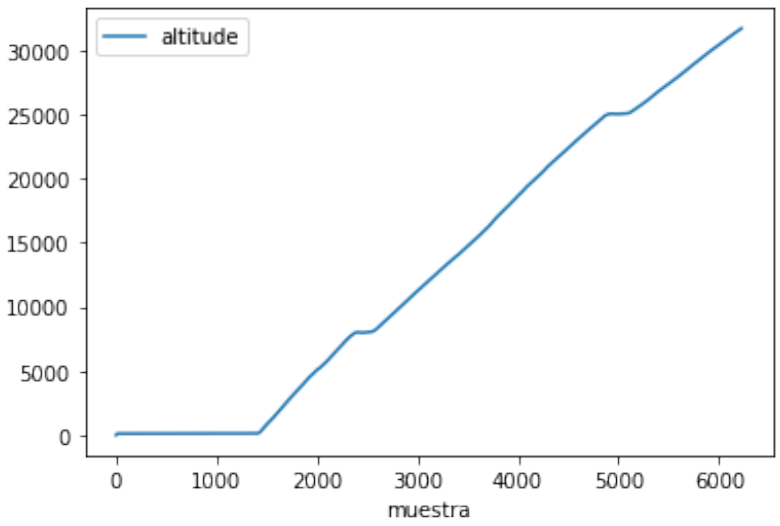


Figura 7.1 Evolución de la altitud en despegue y ascenso.

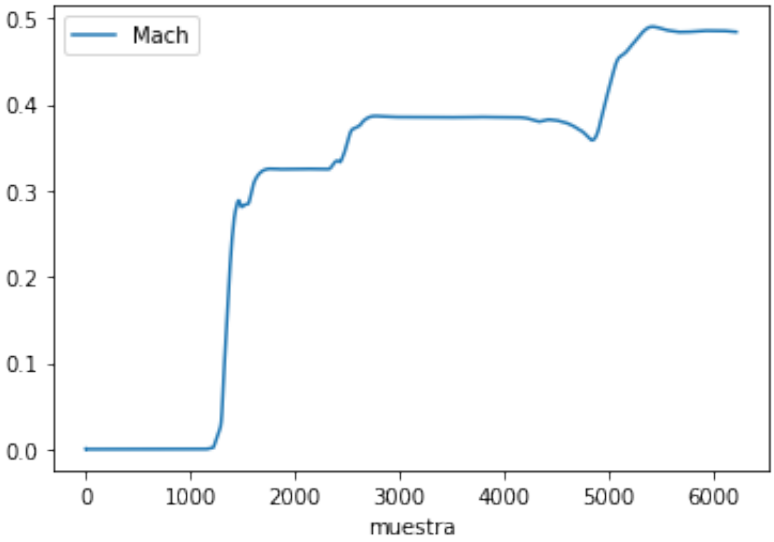


Figura 7.2 Evolución del Mach en despegue y ascenso.

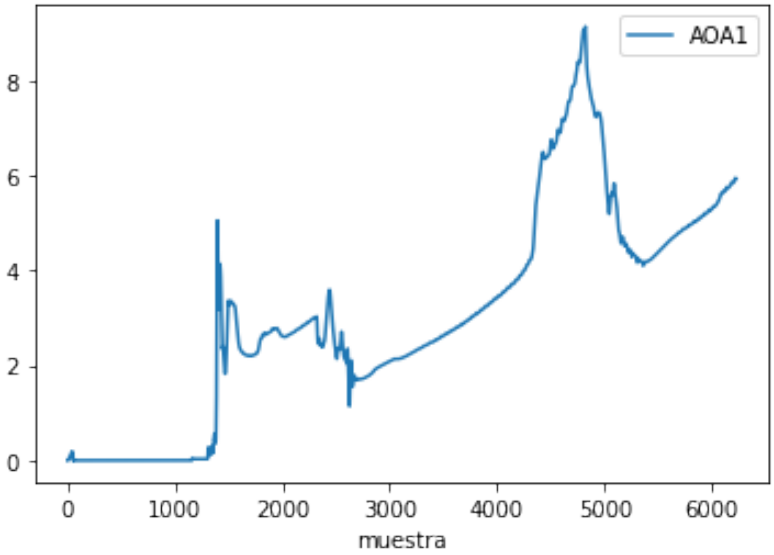


Figura 7.3 Evolución del ángulo de ataque en despegue y ascenso.

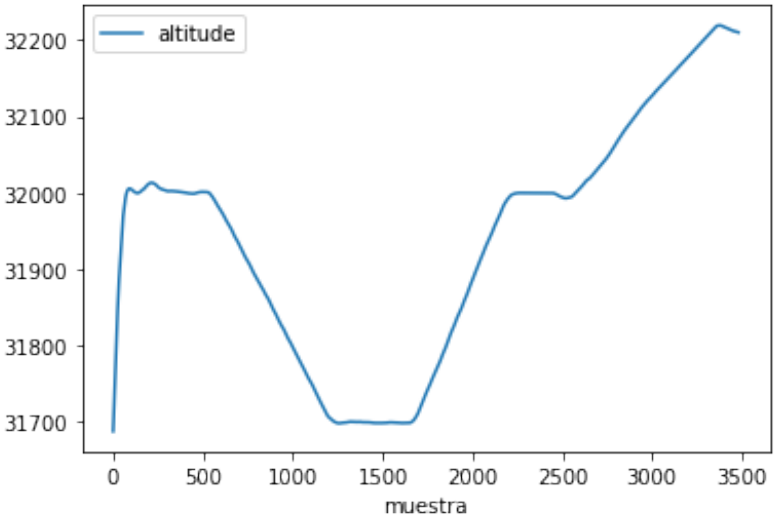


Figura 7.4 Evolución de la altitud en crucero.

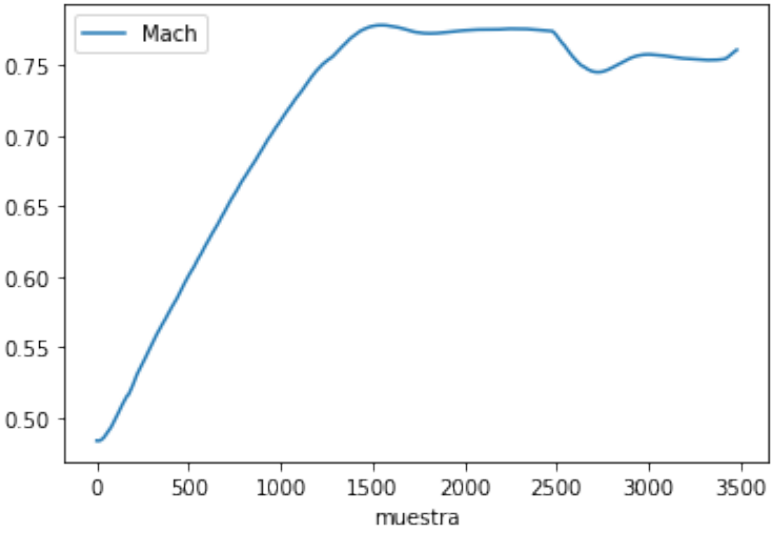


Figura 7.5 Evolución del Mach en crucero.

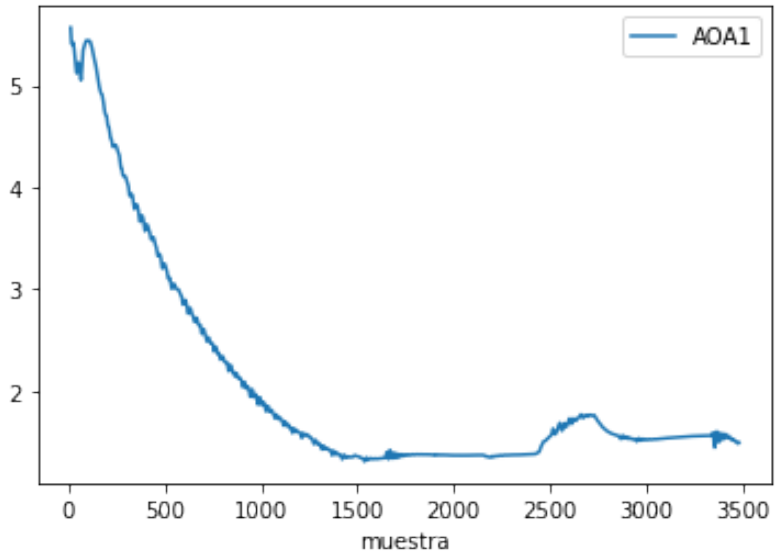


Figura 7.6 Evolución del ángulo de ataque en crucero.

Mediante las gráficas se puede comprobar que se trata de un vuelo que realiza el ascenso en tres tramos y que las condiciones de crucero son una altitud de unos 32000 ft y una velocidad de Mach 0.75 aproximadamente. Por otro lado, como se puede observar en el siguiente histograma de frecuencias, el ángulo de ataque varía en un rango de entre 0º y 9º aproxiamdamente.

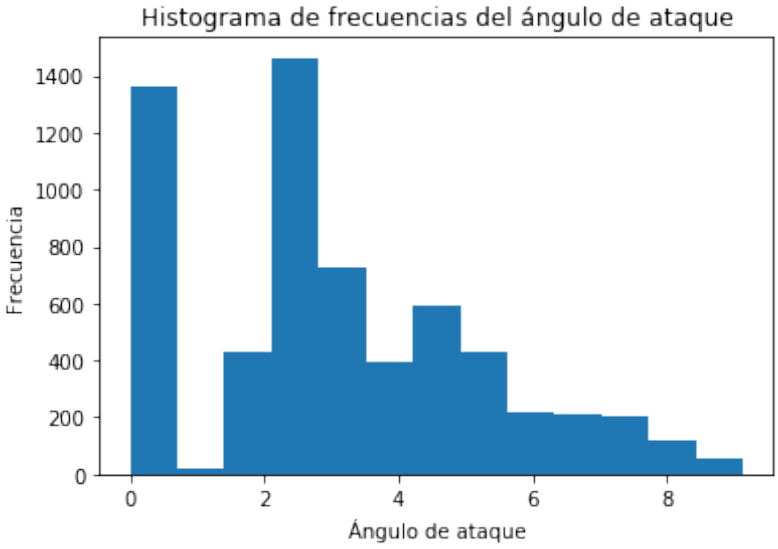


Figura 7.7 Histograma de frecuencias de ángulo de ataque.

Como se ha dicho anteriormente, este apartado trata de comparar el valor real del ángulo de ataque del nuevo vuelo simulado con el valor que el modelo predice a partir de los datos de dicho vuelo. Para estudiar la certeza del ángulo de ataque que el modelo predice, se utilizarán los siguientes parámetros:

- $Porcentaje: P = 100 \cdot \frac{AOA_{real}}{AOA_{pred}}$
- $Error\ absoluto: Ea = |AOA_{real} - AOA_{pred}|$
- $Error\ relativo: Er = 100 \cdot \frac{Ea}{AOA_{pred}} = |100 - P|$

Para visualizar estas métricas de error, se hará uso de la herramienta "describe()", que tiene como output la media, el máximo, el mínimo, la desviación típica y los tres primeros cuantiles de cada una de las variables.

Por otro lado, también es conveniente establecer un valor mínimo de error absoluto a la hora de tomar la variable predicha como válida. Es decir, si el valor absoluto de la diferencia entre la variable predicha y la variable real es menor que dicho valor mínimo, se considerará que el modelo predice un ángulo correcto. De esta forma, se tomará como valor mínimo $Ea_{min} = 0.25$. Este valor se tomará como referencia al calcular el porcentaje de aciertos que tiene cada modelo.

Una vez explicado el procedimiento para estudiar la eficiencia de los modelos, se procede a exponer estos resultados, como en casos anteriores, separados en despegue y ascenso y crucero. Conviene anotar que también se adjunta, para cada uno de ellos, una gráfica con 30 muestras tomadas al azar en las que se observa el valor real y el predicho por el modelo.

7.1 Despegue y ascenso

7.1.1 Modelo 1: Regresión simple

El mejor modelo en este caso es una regresión polinomial simple de grado 3 con CL como variable independiente. Los resultados que arroja este modelo son bastante pésimos. Sin embargo, es conveniente exponerlos ya que ayudarán a tener una visión general de cómo han mejorado los modelos con la complejidad de los mismos.

Tabla 7.1 Modelo de regresión polinómica de grado 3 (CL)
Porcentaje de aciertos: 5.93 %.

	Porcentaje	Error Absoluto	Error relativo
Media	68.04	0.79	37.39
Desviación típica	42.89	0.43	38.25
Mínimo	0.063	0.00	0.04
Primer cuantil	67.56	0.43	9.49
Segundo cuantil	73.43	0.81	26.6
Tercer cuantil	104.71	0.95	32.51
Máximo	907.03	5.24	807.03

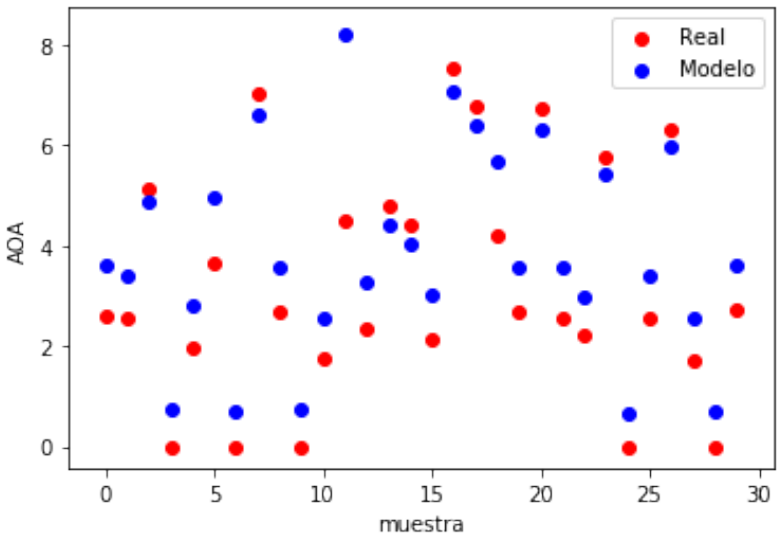


Figura 7.8 Comparación AOA real y AOA del modelo para 30 muestras.

7.1.2 Modelo 2: Regresión lineal múltiple

En este caso el mejor modelo de regresión lineal múltiple con Mach, Wind Speed, altitude, rpm media y CL como variables independientes.

Tabla 7.2 Modelo de regresión lineal múltiple (Mach, Wind Speed, altitude, rpm media, CL) Porcentaje de aciertos: 46.52 %.

	Porcentaje	Error Absoluto	Error relativo
Media	86.90	0.73	21.44
Desviación típica	155.16	0.86	154.22
Mínimo	-5964.88	0.00	-6064.88
Primer cuantil	86.05	0.12	3.56
Segundo cuantil	98.61	0.27	14.11
Tercer cuantil	133.41	1.25	53.31
Máximo	4173.95	6.10	4073.95

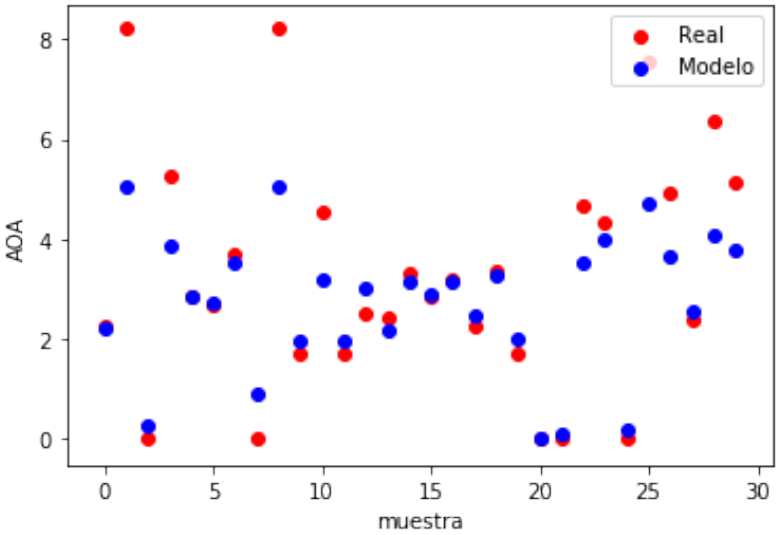


Figura 7.9 Comparación AOA real y AOA del modelo para 30 muestras.

7.1.3 Modelo 3: Regresión simple con técnicas de SVM

En este caso el mejor modelo es un modelo con kernel rbf y con CL como variable independiente.

Tabla 7.3 Modelo de regresión simple con SVM de kernel rbf (CL) Porcentaje de aciertos: 29.78 %.

	Porcentaje	Error Absoluto	Error relativo
Media	72.95	0.87	46.08
Desviación típica	168.24	0.61	164.05
Mínimo	0.05	0.00	0.03
Primer cuantil	56.13	0.20	16.47
Segundo cuantil	70.17	1.12	29.87
Tercer cuantil	103.83	1.26	44.23
Máximo	6633.65	5.80	6533.65

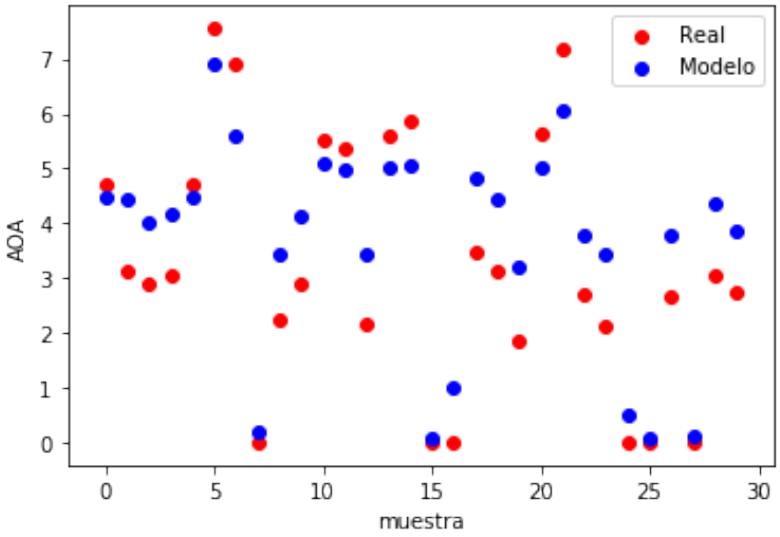


Figura 7.10 Comparación AOA real y AOA del modelo para 30 muestras.

7.1.4 Modelo 4: Regresión múltiple con técnicas de SVM

En este caso se ha decidido que el mejor modelo es el de las siguientes 5 variables con kernel lineal: Mach, Wind Speed, Pitch, Altitude, rpm media.

Tabla 7.4 Modelo de regresión múltiple con SVM de kernel lineal (Mach, wind speed, pitch, altitude, rpm media)
Porcentaje de aciertos: 46.87 %.

	Porcentaje	Error Absoluto	Error relativo
Media	106.14	0.95	46.05
Desviación típica	1190.70	1.103	1189.82
Mínimo	-71666.12	0.00	-71766.12
Primer cuantil	85.03	0.21	8.01
Segundo cuantil	106.89	0.30	42.49
Tercer cuantil	150.07	1.64	92.69
Máximo	59023.06	6.57	58923.06

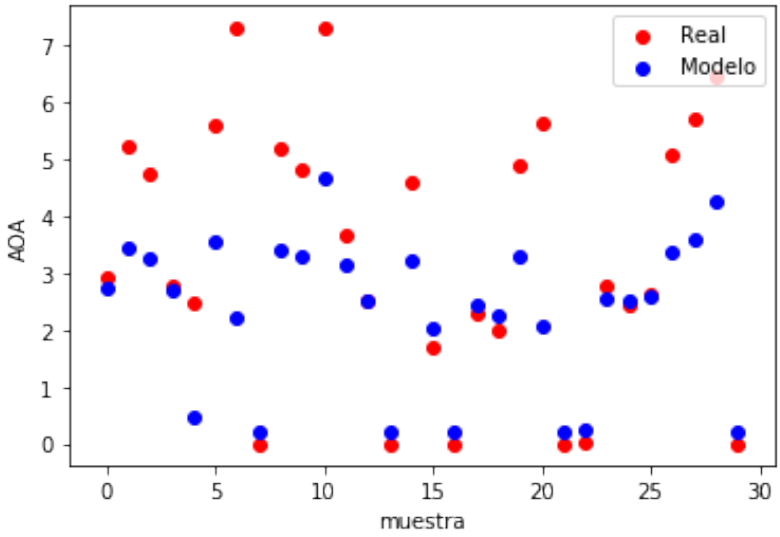


Figura 7.11 Comparación AOA real y AOA del modelo para 30 muestras.

7.1.5 Modelo 5: Regresión múltiple mediante Redes Neuronales

Las variables de este modelo son: Mach, Wind Speed, Altitud, Wind Direction.

Tabla 7.5 Modelo de regresión múltiple con Redes Neuronales (Mach, wind speed, pitch, altitude) Porcentaje de aciertos: 42.27 %.

	Porcentaje	Error Absoluto	Error relativo
Media	75.28	0.92	48.64
Desviación típica	296.20	0.88	293.22
Mínimo	-2979.22	0.00	-3079.22
Primer cuantil	53.16	0.07	13.89
Segundo cuantil	68.36	0.66	33.50
Tercer cuantil	90.90	1.76	46.92
Máximo	13456.72	5.82	13356.72

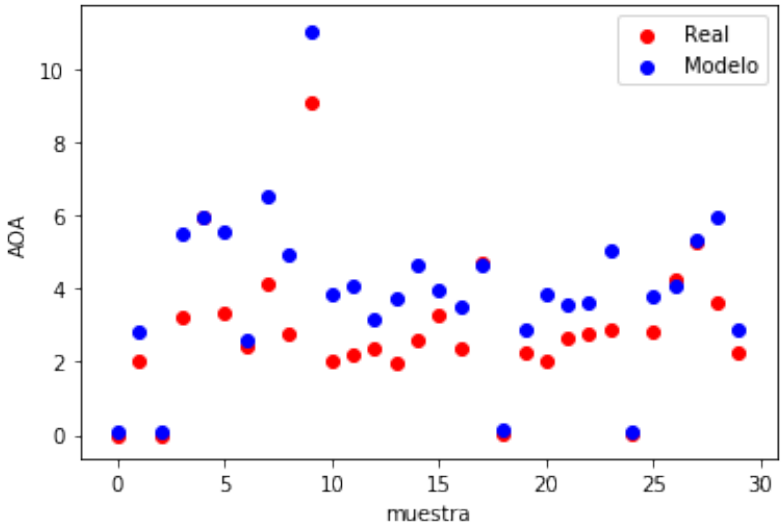


Figura 7.12 Comparación AOA real y AOA del modelo para 30 muestras.

7.2 Crucero

7.2.1 Modelo 1: Regresión simple

El mejor modelo de todos es una regresión lineal simple con CL como variable independiente.

Tabla 7.6 Modelo de regresión lineal simple (CL)
Porcentaje de aciertos: 81.76 %.

	Porcentaje	Error Absoluto	Error relativo
Media	110.80	0.20	10.80
Desviación típica	1.20	0.11	1.20
Mínimo	106.66	0.10	6.66
Primer cuantil	110.23	0.14	10.23
Segundo cuantil	111.06	0.15	11.06
Tercer cuantil	111.51	0.21	11.51
Máximo	115.16	0.76	15.16

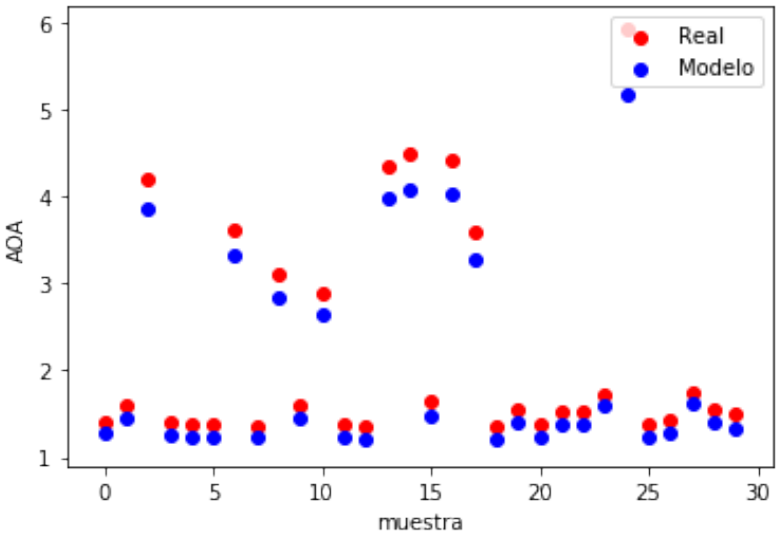


Figura 7.13 Comparación AOA real y AOA del modelo para 30 muestras.

7.2.2 Modelo 2: Regresión lineal múltiple

En este caso el mejor modelo tiene 2 variables independientes: Mach y pitch.

Tabla 7.7 Modelo de regresión lineal múltiple (Mahc, pitch)
Porcentaje de aciertos: 90.32 %.

	Porcentaje	Error Absoluto	Error relativo
Media	99.89	0.16	6.70
Desviación típica	8.35	0.26	4.97
Mínimo	66.83	0.00	0.00
Primer cuantil	94.58	0.04	2.69
Segundo cuantil	100.70	0.12	5.46
Tercer cuantil	106.20	0.20	10.50
Máximo	113.29	2.80	33.17

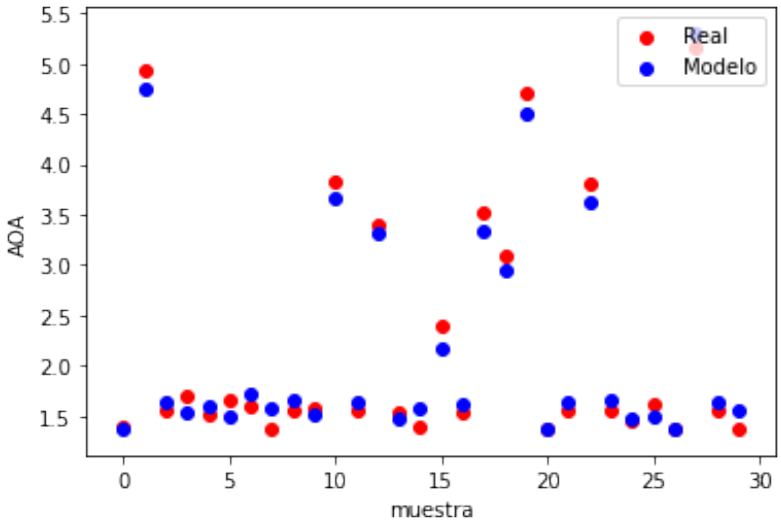


Figura 7.14 Comparación AOA real y AOA del modelo para 30 muestras.

7.2.3 Modelo 3: Regresión simple con técnicas de SVM

En esta caso la variable independiente es el CL y el kernel es rbf.

Tabla 7.8 Modelo de regresión simple con técnicas de SVM (CL)
Porcentaje de aciertos: 95.17 %.

	Porcentaje	Error Absoluto	Error relativo
Media	111.26	0.17	11.30
Desviación típica	5.34	0.08	5.26
Mínimo	98.61	0.00	0.00
Primer cuantil	107.63	0.16	7.63
Segundo cuantil	113.61	0.17	13.61
Tercer cuantil	115.66	0.22	15.66
Máximo	119.31	0.49	19.31

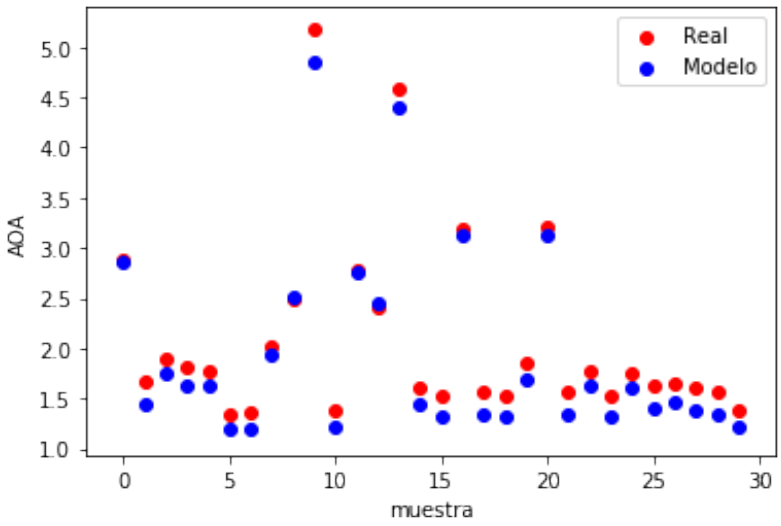


Figura 7.15 Comparación AOA real y AOA del modelo para 30 muestras.

7.3 Conclusiones

En primer lugar se puede observar cómo los resultados obtenidos para crucero son bastante mejores que los obtenidos en despegue y ascenso. Esto se debe a que, como se vio anteriormente, el rango en el que varía el ángulo de ataque en despegue y ascenso es mayor que en crucero. Además, las condiciones de vuelo en la simulación son mucho más homogéneas en el crucero, por lo que esto también influye bastante en la precisión.

Observando los valores que se obtiene para el error absoluto y el porcentaje de aciertos, se decide que los mejores modelos son:

- Despegue y ascenso: regresión lineal múltiple de 4 variables independientes (Mach, Wind Speed, altitud, rpm media y CL).
- Crucero: regresión simple mediante SVM con kernel rbf y con CL como variable independiente.

8 Evolución de los modelos con la cantidad de datos

En esta sección se pretende comprender cómo evoluciona el modelo conforme aumenta la cantidad de datos. Para ello, como se ha hecho a lo largo de toda la memoria, se estudiará de forma separada el despegue y ascenso y el crucero. Los modelos que se someterán a este estudio serán los modelos que se expusieron en la sección anterior, ya que han sido elegidos como los más eficientes.

Para observar la evolución de los modelos con la cantidad de datos, se expondrán tres gráficas diferentes para cada modelo. Estas gráficas son:

1. Score frente al número de datos: en esta gráfica el eje y representa la eficiencia del modelo y el eje x representa el número de datos.

2. Tiempo de ajuste frente al número de datos: en esta gráfica el eje y representa el tiempo que tarda el programa en calcular las variables que predice y el eje x representa el número de datos.

3. Score frente al tiempo de ajuste: en esta gráfica el eje y representa la eficiencia del modelo y el eje x representa el tiempo que se ha nombrado en el punto anterior.

En este caso el score se calcula con una función de Scikit-Learn que aumenta cuando el número de variables predichas es mayor y cuando el error absoluto que se produce al predecirlas es menor.

A continuación se expondrán todas estas gráficas.

8.1 Despegue y ascenso

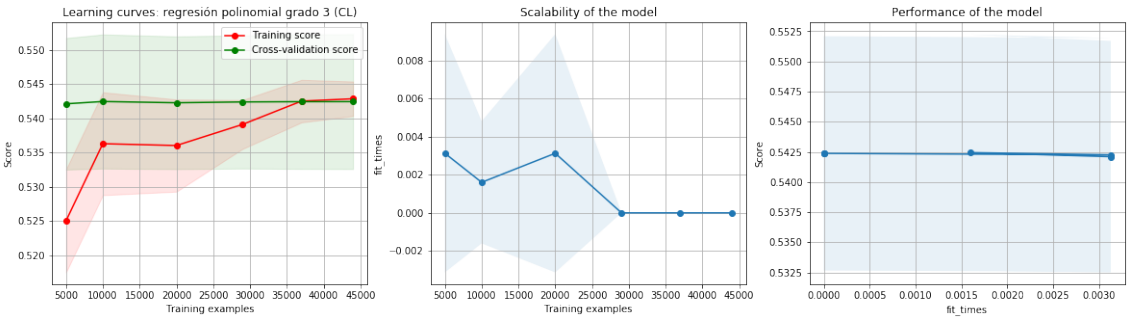


Figura 8.1 Modelo 1: regresión simple.

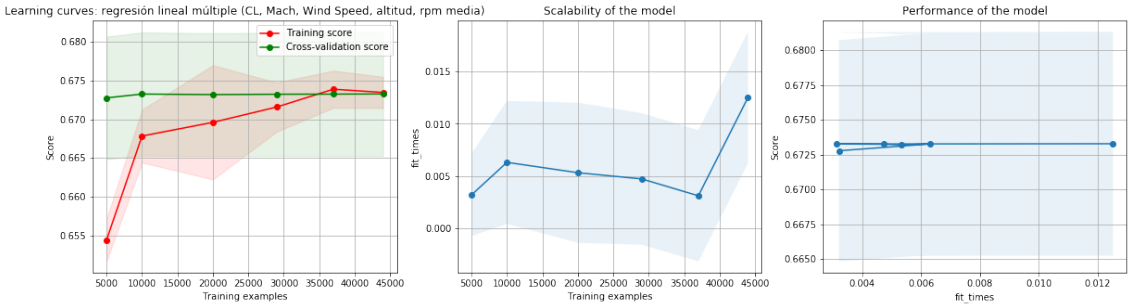


Figura 8.2 Modelo 2: regresión lineal múltiple.

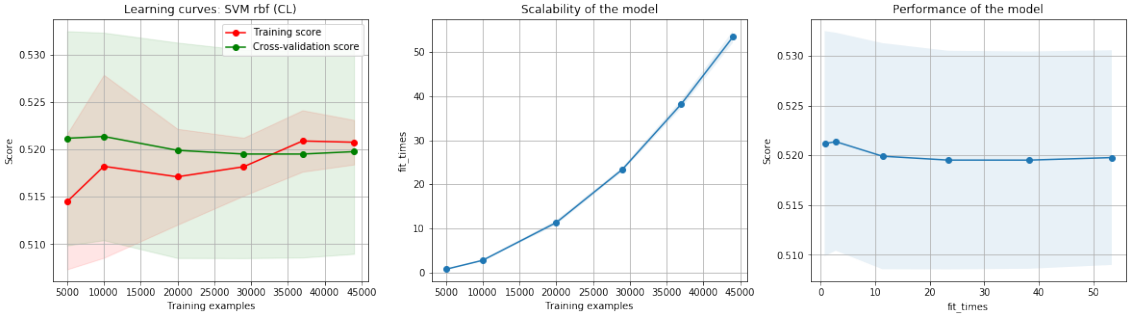


Figura 8.3 Modelo 3: regresión simple con SVM.

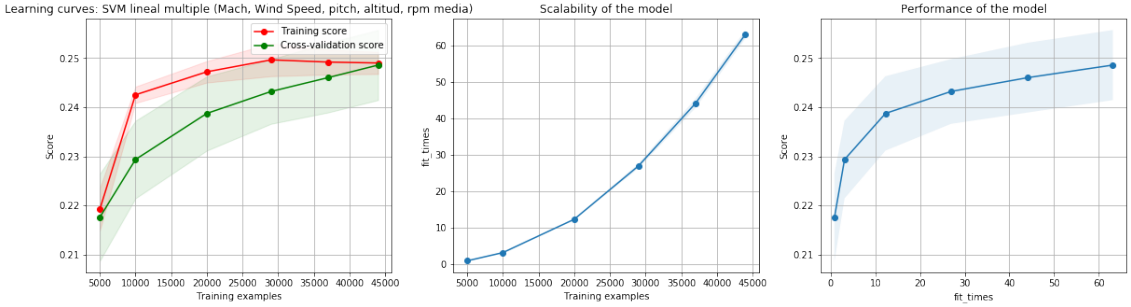


Figura 8.4 Modelo 4: regresión múltiple con SVM.

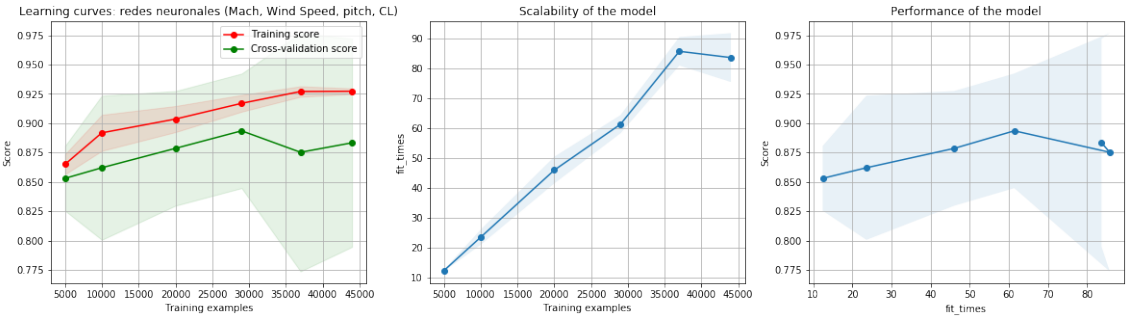


Figura 8.5 Modelo 5: redes neuronales.

En este caso se puede observar que todos los modelos excepto el modelo 5 aumentan su efectividad con el número de datos hasta los 25000 aproximadamente, donde se mantiene aproximadamente constante. En el caso del modelo 5, se observa cómo para 45000 datos la efectividad sigue aumentando, por lo que se podría prever que aumente aún más para una cantidad mayor de datos.

En cuanto al tiempo de ejecución, se observa que en los dos primeros modelos no aumenta significativamente con el número de datos mientras que en los tres últimos modelos sí lo hace de una forma muy precipitada. Este es un resultado que ya se ha comprobado a la hora de ejecutar los modelos y que se debe a la alta complejidad de los tres últimos.

Cabe explicar aquí que el tiempo de ejecución no es para nada insignificante a la hora de elegir un modelo a implementar ya que para la aplicación se este estudio se requiere que el modelo arroje los resultados en tiempo real. De nada serviría un modelo que obtenga resultados muy buenos pero que tenga un tiempo de ajuste demasiado grande.

8.2 Crucero

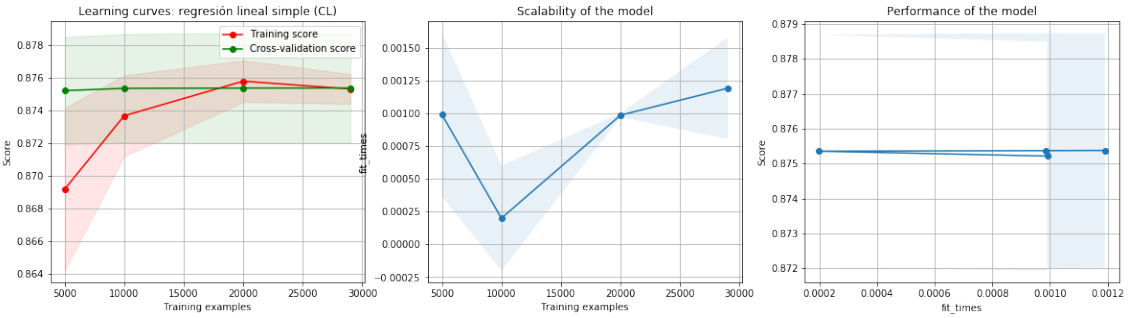


Figura 8.6 Modelo 1: regresión simple.

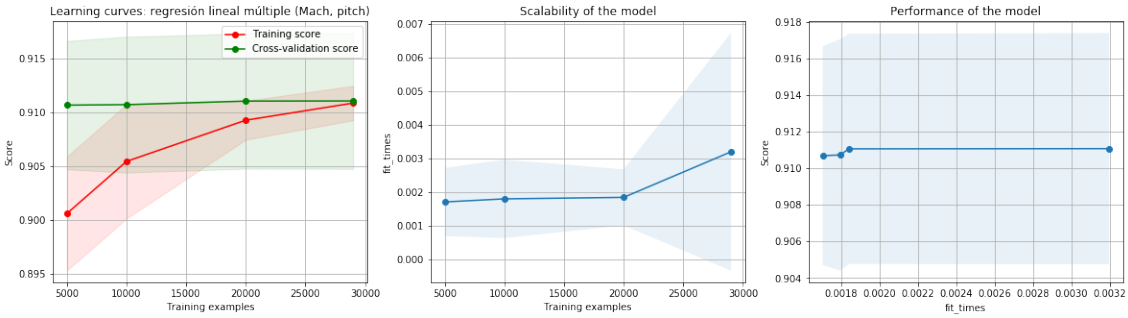


Figura 8.7 Modelo 2: regresión lineal múltiple.

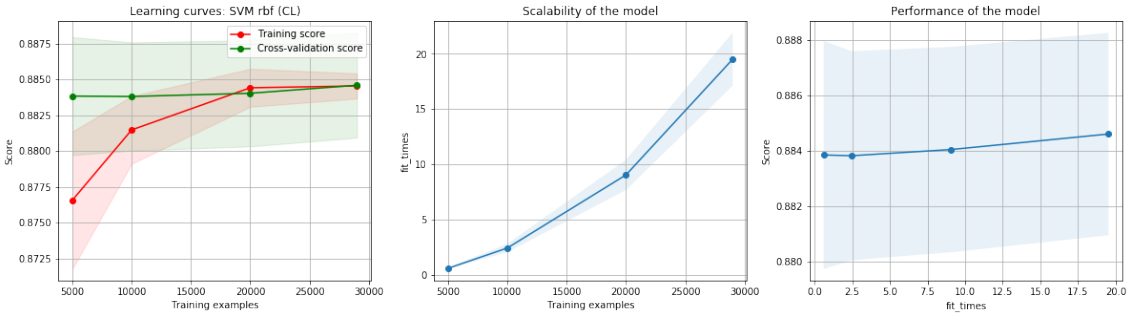


Figura 8.8 Modelo 3: regresión simple con SVM.

En este caso se observa que, para los tres modelos, la eficiencia se mantiene prácticamente constante a partir de los 25000 datos aproximadamente. Por otra parte, al igual que en el caso anterior, los modelos más simples (uno y dos) no experimentan un aumento del tiempo de ejecución tan acusado como el modelo 3, el cuál es mucho más complejo que los dos anteriores.

9 Aplicación del modelo a diferentes casos

En esta última sección se pretende crear diferentes situaciones de fallo en los sensores de ángulo de ataque y comprobar si el modelo implementado solventaría el problema o no. Para ello, se va a utilizar el mismo vuelo simulado que se utilizó en la sección 7. Se utilizarán los datos de este vuelo puesto que, como se aclaró anteriormente, el modelo no los ha utilizado para entrenarse, por tanto, son datos completamente nuevos.

El procedimiento que se seguirá será el siguiente:

1. Se separarán los datos para los casos de despegue y ascenso y de crucero, puesto que los modelos para estos dos tramos son diferentes.
2. Se tomarán muestras elegidas a conciencia del set de datos del vuelo mencionado anteriormente. Para elegir dichas muestras, se tendrán en cuenta las diferentes variables de la misma, de forma que se pretenderá que haya variedad de condiciones. Es decir, se pretende que haya muestras con diferente Mach, altitud, pitch...
3. Se modificarán a conciencia los datos de los tres sensores de ángulo de ataque, de forma que se crearán diversas situaciones de fallo.
4. Se someterán todas las muestras al algoritmo que se explica a continuación y se obtendrán los resultados.
5. Se expondrán los resultados en una tabla explicativa.

En cuanto al algoritmo implementado, es un algoritmo basado en el diagrama de flujo que se expuso en la sección 2. Como se ha explicado anteriormente, se toma 0.25 como el error absoluto máximo que puede cometer el modelo. De esta forma, cuando el resultado arrojado por el modelo y el valor que mide un sensor de ángulo de ataque se diferencien en menos de 0.25, se interpretará como que ambos métodos arrojan el mismo resultado. Sin embargo, el resultado que se tomará será siempre el medido por el sensor, ya que de esta forma podemos corregir los errores más altos del modelo (los que se aproximan a 0.25).

Como también es necesario tomar un valor para el cuál los valores medidos por los sensores son diferentes y no se tienen datos reales, se tomará igualmente 0.25 como diferencia máxima para decidir si dos sensores están arrojando resultados diferentes.

Como se aprecia en el diagrama de flujo de la sección 2, pueden darse 6 escenarios. A continuación, se buscarán a conciencia muestras que provoquen cada uno de estos escenarios y se harán ciertas observaciones. Cabe destacar aquí que no es posible medir el porcentaje en el que se da cada uno de estos escenarios ya que el fallo del sensor AOA es introducido a conciencia para provocarlos, debido a que no se tienen datos reales o simulados en los que el sensor falle.

9.1 Despegue y ascenso

9.1.1 Caso 1 y 2: todos los sensores funcionan

En caso de que todos los sensores funcionan, podría darse que el modelo arroje un valor que coincida con el de los sensores (caso 1) o, en caso contrario, que no coincida (caso 2).

Tabla 9.1 Caso 1.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
3.362	3.381	3.365	3.29	3.362

Tabla 9.2 Caso 2.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
6.166	6.212	6.123	4.073	6.166

Tanto en el caso 1 como en el caso 2, no habrá ningún problema puesto que la redundancia de los sensores hace que se pueda asegurar que el dato es correcto (suponemos como muy improbable que los 3 sensores fallen arrojando el mismo valor).

En cuanto al modelo, el problema sería el caso 2, ya que falla en el resultado. Sin embargo, no tendría repercusión a la hora de tomar el valor.

9.1.2 Caso 3, 4 y 5: un sensor arroja un valor diferente y dos sensores arrojan un valor igual

En este caso, se supondrá que el sensor 2 arroja un valor diferente mientras que los sensores 1 y 3 arrojan el mismo valor. En este caso, podría ocurrir que el resultado del modelo sea igual al del sensor 2 (caso 3), que sea igual al del sensor 1 y 3 (caso 4) o que no sea igual a ninguno (caso 5).

Tabla 9.3 Caso 3.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
5.172	2.280	5.352	2.219	2.280

Tabla 9.4 Caso 4.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
3.474	4.012	3.494	3.374	3.474

Tabla 9.5 Caso 5.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
4.980	4.002	4.975	3.691	-

En los casos 3 y 4 el resultado del modelo es de gran utilidad ya que permite saber cuál de los sensores está funcionando correctamente. Sin embargo, en el caso 5, el modelo estará probablemente fallando, por lo que no puede ser de utilidad a la hora de decidir qué sensor es el que funciona. En este caso, el piloto deberá usar su intuición para tomar como verdadero el valor de los sensores o no.

Cabe aquí destacar que el objetivo y la gran ventaja de este estudio es evitar este tipo de situaciones. Como se ha visto anteriormente, los modelos implementados para despegue y aterrizaje podrían evitarlas en casi un 50 % mientras que los modelos de crucero las evitarían en más de un 95 %. También cabe destacar que la eficiencia de estos modelos puede mejorar considerablemente ya sea con datos o con cambios en los mismos modelos.

9.1.3 Caso 6 y 7: todos los sensores dan un valor diferente

En este caso, todos los sensores dan un valor diferente, por lo que puede ocurrir que el modelo arroje un resultado que sea igual a alguno de los sensores (caso 6) o que el modelo no coincida con ninguno (caso 7).

Tabla 9.6 Caso 6.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
6.205	5.348	4.002	5.253	5.348

Tabla 9.7 Caso 7.

AOA1	AOA2	AOA3	AOA modelo	AOA efectivo
1.256	2.650	2.003	0.506	-

En el caso 6, el modelo sería de gran ayuda para identificar que el ángulo de ataque correcto es el que aloja el sensor 2, mientras que, en el caso 7, de nuevo sería el piloto el que tendría que guiarse por su intuición y decidir si tomar como correcto alguno de los valores o no.

9.2 Crucero

En el caso del crucero los razonamientos son exactamente los mismos que en el despegue y ascenso, solo que el número de aciertos es mayor. Por tanto, no se adjuntarán tablas de resultados para este tramo ya que arrojan resultados en los que el algoritmo se comportaría de forma idéntica al caso anterior.

9.3 Conclusiones

Como ya se ha expuesto anteriormente, la implementación del modelo no hace que el método de medición del ángulo de ataque sea infalible pero sí ayuda considerablemente a corregir errores como los vistos en los casos 3, 4 y 6.

Cabe repetir aquí que la capacidad del modelo de corregir esos errores depende de su eficiencia. En el caso del crucero se podría corregir más del 95 % de estos casos mientras que en el despegue y ascenso se ha llegado solo a la cifra del 47 %. Sin embargo, como se ha dicho anteriormente, el hecho de aumentar el número de datos, hacer nuevas filtraciones de éstos, cambiar el proceso de data cleaning o, sin más, implementar otros tipos de modelos podría mejorar considerablemente la cifra obtenida en este estudio.

10 Conclusiones y líneas futuras

10.1 Conclusiones

En esta última sección se pretende hacer un resumen de las conclusiones que se han ido obteniendo a lo largo de la memoria.

En primer lugar, conviene recordar que los resultados que se han obtenido para crucero (efectividad de más del 95 %) son mucho mejores que los que se han obtenido en despegue y ascenso (efectividad de casi el 50 %). Por tanto se podría decir que en crucero el modelo es casi completamente efectivo, es decir, en prácticamente todas las situaciones en las que los sensores AOA fallen, el modelo corregirá el error. Sin embargo, en el caso de despegue y ascenso, esto sólo ocurrirá en aproximadamente la mitad de las ocasiones. Aún así, el modelo podría ser rentable de instalar, ya que no requiere ningún Hardware que incremente el coste y podría mejorarse. En cuanto a esto último, las formas de mejorarlo se tratará en líneas futuras.

En cuanto a los modelos, se ha observado que, para el caso del crucero, la regresión simple con técnicas de SVM y con CL como variable independiente es suficiente para conseguir buenos resultados. En el caso de despegue y ascenso, el modelo que ha conseguido mejores resultados (mayor porcentaje de aciertos en el testing) ha sido la regresión lineal múltiple. Sin embargo, hay otros modelos (como las redes neuronales o el SVM múltiple) que han obtenido un porcentaje de aciertos muy cercano al de la regresión lineal múltiple y además se ha observado que la desviación típica de estos modelos es menor que en la lineal múltiple. Esto podría conducir a la conclusión de que estos modelos más complejos podrían superar los resultados obtenidos si se realizan ciertos cambios, los cuales se tratarán en el apartado de líneas futuras.

También conviene hablar en esta sección sobre los tiempos de ejecución. Se observa que los modelos más simples (regresiones simples y múltiples) requieren un tiempo de ejecución que es ínfimo, mientras que los modelos más complejos (SVM y redes neuronales) requieren un tiempo de ejecución que es de varios órdenes de magnitud mayor que en el caso de los modelos simples. Esto debe tenerse en cuenta a la hora de elegir el modelo, ya que, como se necesitan los resultados a tiempo real, no puede implementarse un modelo con un tiempo de ejecución demasiado grande. También se observa que, mientras que, en los modelos más simples, el tiempo de ejecución no cambia con la cantidad de datos, en los modelos más complejos si se observa un aumento considerable.

Finalmente, también cabe destacar que se ha observado que el único modelo que podría seguir mejorando con la cantidad de datos es el modelo de redes neuronales.

10.2 Líneas futuras

Como este estudio se corresponde a un proyecto de final de carrera. No es posible extenderlo demasiado debido a los límites de tiempo y contenido requeridos. El hecho de que este estudio haya partido de 0 y no haya tomado ningún resultado de ningún estudio similar hace que los resultados mostrados sean unas primeras aproximaciones. Es decir, se podría avanzar y profundizar mucho más. El objeto de este apartado es dar unas pautas de cómo se podría seguir mejorando los resultados.

En primer lugar, cabe recordar que el estudio se ha realizado para los tramos de despegue y ascenso y de crucero. Esto se debe a que resultaba muy complicado simular el aterrizaje de una manera aproximada a la realidad (se necesita mucha experiencia con el simulador para ello). Por tanto, una forma de hacer este

estudio más completo sería obtener datos de descenso y aterrizaje y realizar el procedimiento que se ha seguido con los datos de esta memoria.

En relación a los datos de la simulación, también podría ayudar bastante a mejorar los resultados el hecho de realizar simulaciones que se acerquen más a la realidad (persona que usa el simulador con más experiencia) o incluso con datos reales. Esto ayudaría a que los datos probablemente sean más homogéneos, ya que el vuelo seguiría las pautas marcadas por ley (velocidad, altura, tiempos de despegue y aterrizaje...) y los tramos de vuelo en los que no es posible utilizar el piloto automático serían mucho más controlados.

Por otra parte, otra forma de mejorar los resultados sería probar los modelos utilizados con diferentes parámetros (kernel, iteraciones...) o directamente probar otro tipo de modelos de regresión. En la misma libraría de Scikit Learn hay más modelos que no se han utilizado. El hecho de probar no quiere decir que se vayan a obtener categóricamente mejores resultados. Sin embargo, se cubriría un rango más amplio de posibilidades, entre las cuales podría estar la óptima.

Finalmente, algo que podría mejorar considerablemente los resultados sería una nueva filtración de datos. En esta memoria se ha observado cómo, al eliminar datos correspondientes a ángulos de ataque muy grandes y poco frecuentes, el modelo mejoraba bastante. Por tanto, otros procedimientos similares podrían ayudar a la mejora. Además, un nuevo proceso de filtración de datos en el que se elimine cantidad de datos y se dejen solamente aquellos datos que optimizan el funcionamiento del modelo podría mejorar, además de los resultados, la efectividad de los algoritmos (por ejemplo, el tiempo de ejecución). Algunas formas de realizar estos procesos podrían ser la intuición (mediante visualizaciones y pruebas se va observando qué datos son los más relevantes) o la utilización de algoritmos estocásticos (como los algoritmos genéticos) que realicen de forma automática una exploración sobre qué datos son los más efectivos para realizar. Además, el hecho de conocer qué datos conviene introducir en el modelo, podría ser bastante efectivo a la hora de implementarlo y que siga aprendiendo con los mismos datos de los vuelos en los que ha funcionado.

11 Anexo de códigos

11.1 Carga de datos

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os


# In[67]:


mainpath = "..\datos ascenso y crucero"
filename1 = "Data1.xlsx"
data1 = pd.read_excel(mainpath + "/" + filename1)

filename2 = "Data2.xlsx"
data2 = pd.read_excel(mainpath + "/" + filename2)

filename3 = "Data3.xlsx"
data3 = pd.read_excel(mainpath + "/" + filename3)

filename4 = "Data4.xlsx"
data4 = pd.read_excel(mainpath + "/" + filename4)

filename5 = "Data5.xlsx"
data5 = pd.read_excel(mainpath + "/" + filename5)

filename6 = "Data6.xlsx"
data6 = pd.read_excel(mainpath + "/" + filename6)

filename7 = "Data7.xlsx"
data7 = pd.read_excel(mainpath + "/" + filename7)

filename8 = "Data7.xlsx"
data8 = pd.read_excel(mainpath + "/" + filename8)
```

```
# In[2]:


mainpath = "..\datos ascenso y crucero"
filename9 = "Data9.xlsx"
data9 = pd.read_excel(mainpath + "/" + filename9)


# In[68]:


variables=['muestra','_totl,_time','__VIS,_tris','_Vind,_kias','_Vind,_keas','
    Vtrue,_ktas','Vtrue,_ktgs','_Vind,__mph','Vtrue,mphas','Vtrue,mphgs','_Mach
    ,ratio','_wind,speed','_wind,__dir','_dens,ratio','baro_,_inHG','_elev,_
    surf','ailrn,_surf','ruddr,_surf','____M,_ftlb','____L,_ftlb','____N,_ftlb
    ','____Q,rad/s','____P,rad/s','____R,rad/s','pitch,__deg','_roll,__deg','
    hding,_true','hding,__mag','alpha,__deg','_beta,__deg','hpath,__deg','vpath
    ,__deg','_slip,__deg','__lat,__deg','__lon,__deg','__alt,__ind','thrst,_1,
    lb','thrst,_2,lb','rpm_1,engin','rpm_2,engin','_lift,___lb','_drag,___lb
    ','__L/D,ratio','___cl,total','___cd,total','__L/D,ratio','elev1,__deg','
    rudr1,__deg','wing1,_lift','wing1,_lift','wing2,_lift','wing2,_lift','wing
    3,_lift','wing3,_lift','wing1,_drag','wing1,_drag','wing2,_drag','wing2,_
    drag','wing3,_drag','wing3,_drag','hstab,_lift','hstab,_lift','vstb2,_lift
    ','hstab,_drag','hstab,_drag','vstb2,_drag']

crucero=[0,28000,30000,28000,28000,35000,32000,37000,36000]

data1=data1[variables]
data2=data2[variables]
data3=data3[variables]
data4=data4[variables]
data5=data5[variables]
data6=data6[variables]
data7=data7[variables]
data8=data8[variables]

aux1=data1.loc[(data1['__alt,__ind']>crucero[1]*0.99) & (data1['__alt,__ind']<
    crucero[1]*1.01),'muestra']
data1A = data1[data1['muestra']<aux1.min()]
data1C = data1[data1['muestra']>aux1.min()]

aux2=data2.loc[(data2['__alt,__ind']>crucero[2]*0.99) & (data2['__alt,__ind']<
    crucero[2]*1.01),'muestra']
data2A = data2[data2['muestra']<aux2.min()]
data2C = data2[data2['muestra']>aux2.min()]

aux3=data3.loc[(data3['__alt,__ind']>crucero[3]*0.99) & (data3['__alt,__ind']<
    crucero[3]*1.01),'muestra']
data3A = data3[data3['muestra']<aux3.min()]
data3C = data3[data3['muestra']>aux3.min()]

aux4=data4.loc[(data4['__alt,__ind']>crucero[4]*0.99) & (data1['__alt,__ind']<
    crucero[4]*1.01),'muestra']
data4A = data4[data4['muestra']<aux4.min()]
data4C = data4[data4['muestra']>aux4.min()]
```

```
aux5=data5.loc[(data5['__alt,__ind']>crucero[5]*0.99) & (data5['__alt,__ind']<
    crucero[5]*1.01),'muestra']
data5A = data5[data5['muestra']<aux5.min()]
data5C = data5[data5['muestra']>aux5.min()]

aux6=data6.loc[(data6['__alt,__ind']>crucero[6]*0.99) & (data6['__alt,__ind']<
    crucero[6]*1.01),'muestra']
data6A = data6[data6['muestra']<aux6.min()]
data6C = data6[data6['muestra']>aux6.min()]

aux7=data7.loc[(data7['__alt,__ind']>crucero[7]*0.99) & (data7['__alt,__ind']<
    crucero[7]*1.01),'muestra']
data7A = data7[data7['muestra']<aux7.min()]
data7C = data7[data7['muestra']>aux7.min()]

aux8=data8.loc[(data8['__alt,__ind']>crucero[8]*0.99) & (data8['__alt,__ind']<
    crucero[8]*1.01),'muestra']
data8A = data8[data8['muestra']<aux8.min()]
data8C = data8[data8['muestra']>aux8.min()]


data1C.reset_index(drop=True, inplace=True)
muestra1=pd.Series(range(0,data1C.shape[0]))
data1C['muestra']=muestra1

data2C.reset_index(drop=True, inplace=True)
muestra2=pd.Series(range(0,data2C.shape[0]))
data2C['muestra']=muestra2

data3C.reset_index(drop=True, inplace=True)
muestra3=pd.Series(range(0,data3C.shape[0]))
data3C['muestra']=muestra3

data4C.reset_index(drop=True, inplace=True)
muestra4=pd.Series(range(0,data4C.shape[0]))
data4C['muestra']=muestra4

data5C.reset_index(drop=True, inplace=True)
muestra5=pd.Series(range(0,data5C.shape[0]))
data5C['muestra']=muestra5

data6C.reset_index(drop=True, inplace=True)
muestra6=pd.Series(range(0,data6C.shape[0]))
data6C['muestra']=muestra6

data7C.reset_index(drop=True, inplace=True)
muestra7=pd.Series(range(0,data7C.shape[0]))
data7C['muestra']=muestra7

data8C.reset_index(drop=True, inplace=True)
muestra8=pd.Series(range(0,data8C.shape[0]))
data8C['muestra']=muestra8


# In[3]:
```

```
variables=['muestra','_totl,_time','__VIS,_tris','_Vind,_kias','_Vind,_keas','
    Vtrue,_ktas','Vtrue,_ktgs','_Vind,__mph','Vtrue,mphas','Vtrue,mphgs','_Mach
    ,ratio','_wind,speed','_wind,__dir','_dens,ratio','baro_,_inHG','_elev,_
    surf','ailrn,_surf','ruddr,_surf','____M,_ftlb','____L,_ftlb','____N,_ftlb
    ','____Q,rad/s','____P,rad/s','____R,rad/s','pitch,__deg','_roll,__deg','
    hding,_true','hding,__mag','alpha,__deg','_beta,__deg','hpath,__deg','vpath
    ,__deg','_slip,__deg','__lat,__deg','__lon,__deg','__alt,__ind','thrst,_1,
    lb','thrst,_2,lb','rpm_1,engin','rpm_2,engin','_lift,___lb','_drag,___lb
    ','__L/D,ratio','___cl,total','___cd,total','__L/D,ratio','elev1,__deg','
    rudr1,__deg','wing1,_lift','wing1,_lift','wing2,_lift','wing2,_lift','wing
    3,_lift','wing3,_lift','wing1,_drag','wing1,_drag','wing2,_drag','wing2,_
    drag','wing3,_drag','wing3,_drag','hstab,_lift','hstab,_lift','vstb2,_lift
    ','hstab,_drag','hstab,_drag','vstb2,_drag']
crucero=[0,28000,30000,28000,28000,35000,32000,37000,36000,32000]

data9=data9[variables]

aux9=data9.loc[(data9['__alt,__ind']>crucero[9]*0.99) & (data9['__alt,__ind']<
    crucero[9]*1.01),'muestra']
data9A = data9[data9['muestra']<aux9.min()]
data9C = data9[data9['muestra']>aux9.min()]

data9C.reset_index(drop=True, inplace=True)
muestra9=pd.Series(range(0,data9C.shape[0]))
data9C['muestra']=muestra9


# In[69]:




dataA=pd.concat([data1A,data2A,data3A,data4A,data5A,data6A,data7A,data8A])

dataC=pd.concat([data1C,data2C,data3C,data4C,data5C,data6C,data7C,data8C])


# In[70]:


mainpath = "..\datos ascenso y crucero"
filenameA = "DataA.xlsx"
dataA.to_excel(mainpath + "/" + filenameA, index = False)

mainpath = "..\datos ascenso y crucero"
filenameC = "DataC.xlsx"
dataC.to_excel(mainpath + "/" + filenameC, index = False)


# In[4]:


mainpath = "..\datos ascenso y crucero"
filename9A = "DataA9.xlsx"
data9A.to_excel(mainpath + "/" + filename9A, index = False)

mainpath = "..\datos ascenso y crucero"
```

```
filename9C = "DataC9.xlsx"
data9C.to_excel(mainpath + "/" + filename9C, index = False)


# In[6]:


data9A.plot('muestra','__alt,__ind')


# In[7]:


data9C.plot('muestra','__alt,__ind')


# In[ ]:
```

11.2 Data cleaning

11.2.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[13]:


import pandas as pd
import os
import random


# In[14]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[15]:


data.plot(kind="scatter", x='alpha,__deg', y='___cl,total')


# In[24]:


data.loc[data['alpha,__deg']>14,'alpha,__deg']=None
data.loc[data['alpha,__deg']<0,'alpha,__deg']=None
data.loc[0,'alpha,__deg']=0
data['alpha,__deg']=data['alpha,__deg'].fillna(method='ffill')
```

```
data.loc[data['___cl,total']>1.6,'___cl,total']=None
data.loc[data['___cl,total']<0,'___cl,total']=None
data['___cl,total']=data['___cl,total'].fillna(method='ffill')
data.loc[(data['___cl,total']>0.2) & (data['alpha,__deg']<0.15),'___cl,total']=
    None
data.loc[(data['___cl,total']>0.2) & (data['alpha,__deg']<0.15),'alpha,__deg']=
    None
data['___cl,total']=data['___cl,total'].fillna(method='ffill')
data['alpha,__deg']=data['alpha,__deg'].fillna(method='ffill')

data = data.drop(data[(data['_Mach,ratio'] < 0.05) & (data['alpha,__deg'] > 4)
    ].index)

data=data.sample(frac=1).reset_index(drop=True)


# In[25]:


data.plot(kind="scatter", x='alpha,__deg', y='___cl,total')


# In[26]:


data.plot(kind="scatter", x='alpha,__deg', y='___cd,total')


# In[27]:


data.loc[data['___cd,total']>1.2,'___cd,total']=None
data.loc[data['___cd,total']<0,'___cd,total']=None
data['___cd,total']=data['___cd,total'].fillna(method='ffill')


# In[28]:


data.plot(kind="scatter", x='alpha,__deg', y='___cd,total')


# In[29]:


data_mapa_calor=data[['_totl,_time','_Mach,ratio','_dens,ratio','pitch,__deg
    ','_roll,__deg','hding,_true','alpha,__deg','_beta,__deg','__alt,__ind','
    thrst,_1,lb','thrst,_2,lb','___cl,total','___cd,total','__L/D,ratio']]


corr = data_mapa_calor.corr()
import seaborn as sns
import matplotlib.pylab as plt
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");
```

```
# In[30]:


data.plot(kind="scatter", x='alpha,__deg', y='__alt,__ind')


# In[31]:


data['alpha,__deg1']=data['alpha,__deg']
data['alpha,__deg2']=data['alpha,__deg']

for i in range(0,data.shape[0]):
    data.loc[i,'alpha,__deg1']=data.loc[i,'alpha,__deg1']*random.uniform(0.99,
        1.01)
    data.loc[i,'alpha,__deg2']=data.loc[i,'alpha,__deg2']*random.uniform(0.99,
        1.01)


# In[32]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data.to_excel(mainpath + "/" + filename, index = False)


# In[33]:


data.shape


# In[ ]:
```

11.2.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[38]:


import pandas as pd
import os
import random


# In[39]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC.xlsx"
data = pd.read_excel(mainpath + "/" + filename)
```

```
# In[40]:


data.plot(kind="scatter", x='alpha,__deg', y='___cl,total')


# In[41]:


data.loc[data['alpha,__deg']>10,'alpha,__deg']=None
data.loc[data['alpha,__deg']<0,'alpha,__deg']=None
data.loc[0,'alpha,__deg']=0
data['alpha,__deg']=data['alpha,__deg'].fillna(method='ffill')

data.loc[data['___cl,total']>1.6,'___cl,total']=None
data.loc[data['___cl,total']<0,'___cl,total']=None
data['___cl,total']=data['___cl,total'].fillna(method='ffill')
data.loc[(data['___cl,total']>0.2) & (data['alpha,__deg']<0.15),'___cl,total']=
    None
data.loc[(data['___cl,total']>0.2) & (data['alpha,__deg']<0.15),'alpha,__deg']=
    None
data.loc[(data['___cl,total']<0.2) & (data['alpha,__deg']>3),'___cl,total']=
    None
data.loc[(data['___cl,total']<0.2) & (data['alpha,__deg']>3),'alpha,__deg']=
    None
data['___cl,total']=data['___cl,total'].fillna(method='ffill')
data['alpha,__deg']=data['alpha,__deg'].fillna(method='ffill')


# In[42]:


data.plot(kind="scatter", x='alpha,__deg', y='___cl,total')


# In[43]:


data.plot(kind="scatter", x='alpha,__deg', y='___cd,total')


# In[44]:


data_mapa_calor=data[['_totl,_time','_Mach,ratio','_dens,ratio','pitch,__deg
    ','_roll,__deg','hding,_true','alpha,__deg','_beta,__deg','__alt,__ind','
    thrst,_1,lb','thrst,_2,lb','___cl,total','___cd,total','__L/D,ratio']]


corr = data_mapa_calor.corr()
import seaborn as sns
import matplotlib.pylab as plt
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");
```

```
# In[45]:


data.plot(kind="scatter", x='alpha,__deg', y='_Mach,ratio')


# In[46]:


data['alpha,__deg1']=data['alpha,__deg']
data['alpha,__deg2']=data['alpha,__deg']

for i in range(0,data.shape[0]):
    data.loc[i,'alpha,__deg1']=data.loc[i,'alpha,__deg1']*random.uniform(0.99,
        1.01)
    data.loc[i,'alpha,__deg2']=data.loc[i,'alpha,__deg2']*random.uniform(0.99,
        1.01)


# In[47]:


data.columns


# In[49]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data.to_excel(mainpath + "/" + filename, index = False)


# In[ ]:
```

11.3 Visualización de datos

11.3.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# In[2]:
```

```
mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


data.shape


# In[4]:


data.columns.values


# ### En este apartado vamos a crear un subset de datos con las columnas más
    relevantes para visualizar

# In[5]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpm media']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['wind direction*']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'wind speed','pitch,__deg':'pitch', '_roll
    ,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})

datai.columns.values


# In[6]:



aux1=datai[['Mach', 'elevator surface', 'aileron surface', 'rudder surface','
    wind speed', 'pitch', 'roll', 'slippage', 'altitude']]


aux1.describe()


# In[7]:
```

```
aux2=datai[['CL', 'CD','AOA1', 'AOA2', 'AOA3', 'empuje total', 'rpm media','
    wind direction*']]

aux2.describe()


# In[8]:


datai.dtypes


# In[9]:


corr = datai.corr()
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");


# In[10]:


data.plot(kind="scatter", x='alpha,__deg', y='___cd,total')


# ### Histogramas de frecuencias

# In[11]:


k = int(np.ceil(1+np.log2(3333)))
plt.hist(datai["AOA1"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Ángulo de ataque")
plt.ylabel("Frecuencia")
plt.title("Histograma de frecuencias del ángulo de ataque")


# In[29]:


plt.boxplot(data["alpha,__deg"])
plt.ylabel("Ángulo de ataque")
plt.title("Boxplot de ángulo de ataque")


# In[30]:


k = int(np.ceil(1+np.log2(3333)))
plt.hist(data["_Mach,ratio"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Mach")
plt.ylabel("Frecuencia")
plt.title("Histograma del Mach")
```

```
# Hay que eliminar todas las filas de datos de dataA en las que Mach=0, ya que
    éstas se corresponden con los momentos previos al despegue.

# In[ ]:
```

11.3.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


data.shape


# In[4]:


data.columns.values


# ### En este apartado vamos a crear un subset de datos con las columnas más
    relevantes para visualizar

# In[5]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpm media']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['wind direction*']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
```

```
    rudder surface', '_wind,speed':'wind speed','pitch,__deg':'pitch', '_roll
    ,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})

datai.columns.values


# In[6]:



aux1=datai[['Mach', 'elevator surface', 'aileron surface', 'rudder surface','
    wind speed', 'pitch', 'roll', 'slippage', 'altitude']]


aux1.describe()


# In[7]:


aux2=datai[['CL', 'CD','AOA1', 'AOA2', 'AOA3', 'empuje total', 'rpm media','
    wind direction*']]

aux2.describe()


# In[8]:


datai.dtypes


# In[9]:


corr = datai.corr()
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");


# In[8]:


data.plot(kind="scatter", x='alpha,__deg', y='___cl,total')


# In[9]:


data.plot(kind="scatter", x='alpha,__deg', y='___cd,total')


# ### Histogramas de frecuencias
```

```
# In[10]:


k = int(np.ceil(1+np.log2(3333)))
plt.hist(datai["AOA1"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Ángulo de ataque")
plt.ylabel("Frecuencia")
plt.title("Histograma de frecuencias del ángulo de ataque")


# In[12]:


plt.boxplot(data["alpha,__deg"])
plt.ylabel("Ángulo de ataque")
plt.title("Boxplot de ángulo de ataque")


# In[13]:


k = int(np.ceil(1+np.log2(3333)))
plt.hist(data["_Mach,ratio"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Mach")
plt.ylabel("Frecuencia")
plt.title("Histograma del Mach")


# In[ ]:
```

11.4 Feature selection

11.4.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectPercentile,SelectFpr, SelectKBest,
    f_classif, f_regression
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures


# In[2]:
```

```
mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpm media']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})




datai=datai.sample(frac=1).reset_index(drop=True)

datai.shape


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[5]:


y=datai.AOA1
X=datai.drop(['AOA1','AOA2','AOA3'],axis=1)
columnas = list(X.columns.values)
X.shape


# # Select Percentile (f_classif)
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    SelectPercentile.html#sklearn.feature_selection.SelectPercentile

# In[6]:


seleccionadas= SelectPercentile(percentile=50).fit(X, y)

atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # Select K Best (f_classif)

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    SelectKBest.html#sklearn.feature_selection.SelectKBest

# In[7]:


seleccionadas = SelectKBest(f_classif, k=6).fit(X,y)
atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # F regression

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f
    _regression.html#sklearn.feature_selection.f_regression

# In[8]:


seleccionadas = f_regression(X,y)
seleccionadas = [columnas[i] for i in list(atrib.nonzero()[0])]
seleccionadas


# # SELECT FPR

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    SelectFpr.html#sklearn.feature_selection.SelectFpr

# In[9]:


seleccionadas = SelectFpr(f_classif, alpha=0.01).fit(X, y)
atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # RFE
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    RFE.html#sklearn.feature_selection.RFE

# In[ ]:
```

11.4.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[7]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectPercentile,SelectFpr, SelectKBest,
    f_classif, f_regression
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures


# In[8]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[9]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpm media']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})




datai=datai.sample(frac=1).reset_index(drop=True)
```

```
datai.shape


# In[10]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[11]:


y=datai.AOA1
X=datai.drop(['AOA1','AOA2','AOA3'],axis=1)
columnas = list(X.columns.values)
X.shape


# # Select Percentile (f_classif)

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    SelectPercentile.html#sklearn.feature_selection.SelectPercentile

# In[12]:


seleccionadas= SelectPercentile(percentile=50).fit(X, y)

atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # Select K Best (f_classif)

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    SelectKBest.html#sklearn.feature_selection.SelectKBest

# In[14]:


seleccionadas = SelectKBest(f_classif, k=6).fit(X,y)
atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # F regression
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f
    _regression.html#sklearn.feature_selection.f_regression

# In[15]:


seleccionadas = f_regression(X,y)
seleccionadas = [columnas[i] for i in list(atrib.nonzero()[0])]
seleccionadas


# # SELECT FPR

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    SelectFpr.html#sklearn.feature_selection.SelectFpr

# In[16]:


seleccionadas = SelectFpr(f_classif, alpha=0.01).fit(X, y)
atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # RFE

# https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.
    RFE.html#sklearn.feature_selection.RFE

# Este método hace la selección de características por eliminación en cuanto a
    los resultados que obtiene en un modelo. Por tanto, para implementarlo es
    necesario implementar previamente un modelo.

# In[19]:



estimator = LinearRegression()
seleccionadas = RFE(estimator, n_features_to_select=10, step=1)
seleccionadas = seleccionadas.fit(X, y)
atributos = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# In[ ]:


from sklearn.model_selection import train_test_split
X = datai['Vtrue,mphas'].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)
```

```
poly = PolynomialFeatures(degree=1)
poly.fit_transform(X_train)

reg = LinearRegression().fit(poly.transform(X_train),y_train)



y_train_hat = reg.predict(poly.transform(X_train))
y_test_hat = reg.predict(poly.transform(X_test))



print("R2 train:",r2_score(y_train,y_train_hat))
print("R2 test:",r2_score(y_test,y_test_hat))
```

11.5 Regresión simple

11.5.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)
```

```
# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpm media']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})




datai=datai.sample(frac=1).reset_index(drop=True)

datai.shape


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[5]:


corr = datai.corr()
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");


# ## REGRESIÓN LINEAL SIMPLE (CL)

# In[6]:


lm = smf.ols(formula="AOA1~CL", data = datai).fit()
lm.summary()
```

```
# In[7]:


AOA1_pred = lm.predict(pd.DataFrame(datai["CL"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "CL", y ="AOA1")
plt.plot(pd.DataFrame(datai["CL"]), AOA1_pred, c="red", linewidth = 2)


# In[13]:



from sklearn.model_selection import train_test_split
X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,20).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r")
    plt.axis([0, 1, 0, 16])
    plt.xlabel("CL")
    plt.ylabel("AOA")

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# In[11]:




X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

poly = PolynomialFeatures(degree=3)
poly.fit_transform(X_train)

reg = LinearRegression().fit(poly.transform(X_train),y_train)


y_train_hat = reg.predict(poly.transform(X_train))
y_test_hat = reg.predict(poly.transform(X_test))


print("R2 train:",r2_score(y_train,y_train_hat))
print("R2 test:",r2_score(y_test,y_test_hat))

print("Error absoluto medio train:",mean_absolute_error(y_train,y_train_hat))
print("Error absoluto medio test:",mean_absolute_error(y_test,y_test_hat))

print("Error cuadrático medio train:",mean_squared_error(y_train,y_train_hat))
print("Error cuadrático medio test:",mean_squared_error(y_test,y_test_hat))


x_model = np.linspace(0,20).reshape((-1,1))
y_model = reg.predict(poly.transform(x_model))
plt.plot(x_model,y_model, "r--")
plt.axis([0, 2, 0, 16])
plt.show()


# # REGRESIÓN LINEAL SIMPLE (PITCH)

# In[20]:


lm = smf.ols(formula="AOA1~pitch", data = datai).fit()
lm.summary()


# In[21]:


AOA1_pred = lm.predict(pd.DataFrame(datai["pitch"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "pitch", y ="AOA1")
plt.plot(pd.DataFrame(datai["pitch"]), AOA1_pred, c="red", linewidth = 2)


# In[22]:



from sklearn.model_selection import train_test_split
X = datai["pitch"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-10,40).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([-10, 40, 0, 16])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN LINEAL SIMPLE (WIND DIRECTION*)

# In[23]:


lm = smf.ols(formula="AOA1~winddirection", data = datai).fit()
lm.summary()


# In[24]:


AOA1_pred = lm.predict(pd.DataFrame(datai["winddirection"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "winddirection", y ="AOA1")
plt.plot(pd.DataFrame(datai["winddirection"]), AOA1_pred, c="red", linewidth =
    2)


# In[25]:



from sklearn.model_selection import train_test_split
X = datai["winddirection"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)
```

```
def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-400,300).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([-400, 300, 0, 16])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN LINEAL SIMPLE (WIND SPEED)

# In[26]:


lm = smf.ols(formula="AOA1~windspeed", data = datai).fit()
lm.summary()


# In[27]:


AOA1_pred = lm.predict(pd.DataFrame(datai["windspeed"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "windspeed", y ="AOA1")
plt.plot(pd.DataFrame(datai["windspeed"]), AOA1_pred, c="red", linewidth = 2)


# In[28]:



from sklearn.model_selection import train_test_split
X = datai["windspeed"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)
```

```
    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-100,20).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([0, 18, 0, 16])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN LINEAL SIMPLE (ALTITUDE)

# In[29]:


lm = smf.ols(formula="AOA1~altitude", data = datai).fit()
lm.summary()


# In[30]:


AOA1_pred = lm.predict(pd.DataFrame(datai["altitude"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "altitude", y ="AOA1")
plt.plot(pd.DataFrame(datai["altitude"]), AOA1_pred, c="red", linewidth = 2)


# In[32]:



from sklearn.model_selection import train_test_split
X = datai["altitude"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)
```

```
    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-10,40000).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([0, 40000, 0, 16])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# In[ ]:
```

11.5.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
```

```
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[5]:


corr = datai.corr()
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");


# Se observa que las variables que tienen una correlación lineal más alta en
    valor absoluto con el ángulo de ataque son:
#
# * Pitch: 0.95
# * Coeficiente de sustentación: 0.94
# * Velocidad del viento: 0.84
# * Mach: -0.74
# * Altitud: 0.7
```

```
# * Superficie del elevator deflectada: -0.63
# * Empuje 1 y 2: -0.63
# * Dirección del viento: 0.61
# * Coeficiente de resistencia: 0.42
#
# Por tanto, éstas son las variables que vamos a seleccionar para crear los
    modelos de regresión lineal simple (en el caso del empuje haremos una
    regresión lineal múltiple de dos variables por tener el avión dos motores).
     También se espera que estas variables sean las que salgan a la hora de
    hacer el modelo de regresión lineal múltiple con las librerías.
#
# Destacar por otra parte que, puesto que estamos trabajando con datos en los
    que sabemos que los sensores de ángulos de ataque no han fallado, podemos
    trabajar con una sola columna de dato de sensor de ángulo de ataque ('AOA
    1') puesto que la diferencia respecto a las otras será menor que el 1 %.

# # REGRESIÓN SIMPLE (PITCH)

# In[6]:


lm = smf.ols(formula="AOA1~pitch", data = datai).fit()
lm.summary()


# In[7]:


AOA1_pred = lm.predict(pd.DataFrame(datai["pitch"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "pitch", y ="AOA1")
plt.plot(pd.DataFrame(datai["pitch"]), AOA1_pred, c="red", linewidth = 2)


# * SST = SSD + SSR
# * SST : Variabilidad de los datos con respecto de su media
# * SSD : Diferencia entre los datos originales y las predicciones que el
    modelo no es capaz de explicar (errores que deberían seguir una distribució
    n normal)
# * SSR : Diferencia entre la regresión y el valor medio que el modelo busca
    explicar
# * R2 = SSR / SST, coeficiente de determinación entre 0 y 1

# In[14]:


from sklearn.model_selection import train_test_split
X = datai["pitch"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)
```

```
    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-10,40).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r")
    plt.axis([-0.5, 10, 0, 10])
    plt.xlabel("Pitch")
    plt.ylabel("AOA")

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# In[8]:



X = datai["pitch"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)


poly = PolynomialFeatures(degree=2)
poly.fit_transform(X_train)

reg = LinearRegression().fit(poly.transform(X_train),y_train)



y_train_hat = reg.predict(poly.transform(X_train))
y_test_hat = reg.predict(poly.transform(X_test))



print("R2 train:",r2_score(y_train,y_train_hat))
print("R2 test:",r2_score(y_test,y_test_hat))


print("Error absoluto medio train:",mean_absolute_error(y_train,y_train_hat))
print("Error absoluto medio test:",mean_absolute_error(y_test,y_test_hat))

print("Error cuadrático medio train:",mean_squared_error(y_train,y_train_hat))
print("Error cuadrático medio test:",mean_squared_error(y_test,y_test_hat))
```

```
# # REGRESIÓN SIMPLE (CD)

# In[9]:


lm = smf.ols(formula="AOA1~CD", data = datai).fit()
lm.summary()


# In[10]:


AOA1_pred = lm.predict(pd.DataFrame(datai["CD"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "CD", y ="AOA1")
plt.plot(pd.DataFrame(datai["CD"]), AOA1_pred, c="red", linewidth = 2)


# In[41]:


from sklearn.model_selection import train_test_split
X = datai["CD"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,40).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([0, 0.075, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (MACH)

# In[12]:
```

```
lm = smf.ols(formula="AOA1~Mach", data = datai).fit()
lm.summary()


# In[13]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["Mach"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "Mach", y ="AOA1")
plt.plot(pd.DataFrame(datai["Mach"]), AOA1_pred, c="red", linewidth = 2)


# In[47]:


from sklearn.model_selection import train_test_split
X = datai["Mach"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,40).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([0.4, 1, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (CL)

# In[15]:


lm = smf.ols(formula="AOA1~CL", data = datai).fit()
lm.summary()
```

```
# In[16]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["CL"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "CL", y ="AOA1")
plt.plot(pd.DataFrame(datai["CL"]), AOA1_pred, c="red", linewidth = 2)


# In[13]:


from sklearn.model_selection import train_test_split
X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,40).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r")
    plt.axis([0.1, 0.6, 0, 10])
    plt.xlabel("CL")
    plt.ylabel("AOA")

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# In[12]:


X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)
```

```
poly = PolynomialFeatures(degree=1)
poly.fit_transform(X_train)

reg = LinearRegression().fit(poly.transform(X_train),y_train)



y_train_hat = reg.predict(poly.transform(X_train))
y_test_hat = reg.predict(poly.transform(X_test))



print("R2 train:",r2_score(y_train,y_train_hat))
print("R2 test:",r2_score(y_test,y_test_hat))


print("Error absoluto medio train:",mean_absolute_error(y_train,y_train_hat))
print("Error absoluto medio test:",mean_absolute_error(y_test,y_test_hat))

print("Error cuadrático medio train:",mean_squared_error(y_train,y_train_hat))
print("Error cuadrático medio test:",mean_squared_error(y_test,y_test_hat))


# # REGRESIÓN SIMPLE (WIND SPEED)

# In[18]:


lm = smf.ols(formula="AOA1~windspeed", data = datai).fit()
lm.summary()


# In[19]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["windspeed"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "windspeed", y ="AOA1")
plt.plot(pd.DataFrame(datai["windspeed"]), AOA1_pred, c="red", linewidth = 2)


# In[51]:


from sklearn.model_selection import train_test_split
X = datai["windspeed"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)
```

```
    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,40).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([0, 15, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (ALTITUDE)

# In[21]:


lm = smf.ols(formula="AOA1~altitude", data = datai).fit()
lm.summary()


# In[22]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["altitude"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "altitude", y ="AOA1")
plt.plot(pd.DataFrame(datai["altitude"]), AOA1_pred, c="red", linewidth = 2)


# In[52]:


from sklearn.model_selection import train_test_split
X = datai["altitude"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)
```

```
    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,40000).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([22000, 40000, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (EMPUJE TOTAL)

# In[24]:


lm = smf.ols(formula="AOA1~empujetotal", data = datai).fit()
lm.summary()


# In[25]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["empujetotal"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "empujetotal", y ="AOA1")
plt.plot(pd.DataFrame(datai["empujetotal"]), AOA1_pred, c="red", linewidth = 2)


# In[53]:


from sklearn.model_selection import train_test_split
X = datai["empujetotal"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)
```

```
    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,40000).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([2000, 20000, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (WIND DIRECTION)

# In[27]:


lm = smf.ols(formula="AOA1~winddirection", data = datai).fit()
lm.summary()


# In[28]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["winddirection"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "winddirection", y ="AOA1")
plt.plot(pd.DataFrame(datai["winddirection"]), AOA1_pred, c="red", linewidth =
    2)


# In[54]:


from sklearn.model_selection import train_test_split
X = datai["winddirection"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))
```

```
    x_model = np.linspace(-360,360).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([-360,360, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (ELEVATOR SURFACE)

# In[30]:


lm = smf.ols(formula="AOA1~elevatorsurface", data = datai).fit()
lm.summary()


# In[31]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["elevatorsurface"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "elevatorsurface", y ="AOA1")
plt.plot(pd.DataFrame(datai["elevatorsurface"]), AOA1_pred, c="red", linewidth
    = 2)


# In[55]:


from sklearn.model_selection import train_test_split
X = datai["elevatorsurface"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-360,360).reshape((-1,1))
```

```
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([-1.2,1.2, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (ROLL)

# In[33]:


lm = smf.ols(formula="AOA1~roll", data = datai).fit()
lm.summary()


# In[34]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["roll"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "roll", y ="AOA1")
plt.plot(pd.DataFrame(datai["roll"]), AOA1_pred, c="red", linewidth = 2)


# In[56]:


from sklearn.model_selection import train_test_split
X = datai["roll"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(-360,360).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([-22,42, 0, 10])
```

```
    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))

    plt.show()

interact(plot,degree=(1,10));


# # REGRESIÓN SIMPLE (rpm MEDIA)

# In[36]:


lm = smf.ols(formula="AOA1~rpmmedia", data = datai).fit()
lm.summary()


# In[37]:


from sklearn.metrics import r2_score
AOA1_pred = lm.predict(pd.DataFrame(datai["rpmmedia"]))
get_ipython().run_line_magic('matplotlib', 'inline')
datai.plot(kind = "scatter", x = "rpmmedia", y ="AOA1")
plt.plot(pd.DataFrame(datai["rpmmedia"]), AOA1_pred, c="red", linewidth = 2)


# In[57]:


from sklearn.model_selection import train_test_split
X = datai["rpmmedia"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=100)

def plot(degree = 1):
    poly = PolynomialFeatures(degree=degree)
    poly.fit_transform(X_train)

    reg = LinearRegression().fit(poly.transform(X_train),y_train)

    plt.scatter(X_train,y_train)
    plt.scatter(X_test,y_test)

    y_train_hat = reg.predict(poly.transform(X_train))
    y_test_hat = reg.predict(poly.transform(X_test))

    x_model = np.linspace(0,6000).reshape((-1,1))
    y_model = reg.predict(poly.transform(x_model))
    plt.plot(x_model,y_model, "r--")
    plt.axis([1500,6000, 0, 10])

    print("R2 train:",r2_score(y_train,y_train_hat))
    print("R2 test:",r2_score(y_test,y_test_hat))
```

```
    plt.show()

interact(plot,degree=(1,10));


# In[ ]:
```

11.6 Regresión lineal múltiple

11.6.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[14]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import RFE

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']
```

```
datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)

datai.shape


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[5]:


corr = datai.corr()
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");


# # SELECCIÓN DE CARACTERÍSTICAS CON EL MÉTODO selectKbest

# Este algoritmo selecciona a los mejores atributos basándose en una prueba
    estadística univariante. Al objeto SelectKBest le pasamos la prueba estadí
    stica que vamos a a aplicar, en este caso una prueba F definida por el
    objeto f_classif, junto con el número de atributos a seleccionar. El
    algoritmo va a aplicar la prueba a todos los atributos y va a seleccionar
    los que mejor resultado obtuvieron.

# In[6]:


# Aplicando el algoritmo univariante de prueba F.
data_aux=datai[['Mach', 'elevator surface', 'aileron surface', 'rudder surface
    ','windspeed', 'pitch', 'roll', 'slippage', 'altitude', 'CL', 'CD','AOA1',
    'empuje total', 'rpmmedia', 'winddirection']]
k = 6 # número de atributos a seleccionar
AOA1 = data_aux['AOA1']
entrenar = data_aux.drop(['AOA1'], axis=1)
```

```
columnas = list(entrenar.columns.values)
seleccionadas = SelectKBest(f_classif, k=k).fit(entrenar, AOA1)
atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # MULTICOLINEALIDAD DE LAS VARIABLES

# Factor Inflación de la Varianza
#
# * VIF = 1 : Las variables no están correlacionadas
# * VIF < 5 : Las variables tienen una correlación moderada y se pueden quedar
    en el modelo
# * VIF >5 : Las variables están altamente correlacionadas y deben desaparecer
    del modelo
#
#
# En los modelos lineales múltiples, los predictores deben ser independientes,
    no debe de haber colinealidad entre ellos. La colinealidad ocurre cuando un
     predictor está linealmente relacionado con uno o varios de los otros
    predictores del modelo. Como consecuencia de la colinealidad, no se puede
    identificar de forma precisa el efecto individual que tiene cada predictor
    sobre la variable respuesta, lo que se traduce en un incremento de la
    varianza de los coeficientes de regresión estimados hasta el punto de que
    resulta imposible establecer su significancia estadística. Además, pequeños
     cambios en los datos, provocan grandes cambios en las estimaciones de los
    coeficientes. Si bien la colinealidad propiamente dicha existe solo si el
    coeficiente de correlación simple o múltiple entre predictores es 1, cosa
    que raramente ocurre en la realidad, es frecuente encontrar la llamada casi
    -colinealidad o multicolinealidad no perfecta.
#
# No existe un método estadístico concreto para determinar la existencia de
    colinealidad o multicolinealidad entre los predictores de un modelo de
    regresión, sin embargo, se han desarrollado numerosas reglas prácticas que
    tratan de determinar en qué medida afectan al modelo. Los pasos
    recomendados a seguir son:
#
# Si el coeficiente de determinación R2 es alto pero ninguno de los predictores
     resulta significativo, hay indicios de colinealidad.
#
# Crear una matriz de correlación en la que se calcula la relación lineal entre
     cada par de predictores. Es importante tener en cuenta que, a pesar de no
    obtenerse ningún coeficiente de correlación alto, no está asegurado que no
    exista multicolinealidad. Se puede dar el caso de tener una relación lineal
     casi perfecta entre tres o más variables y que las correlaciones simples
    entre pares de estas mismas variables no sean mayores que 0.5.
#
# Generar un modelo de regresión lineal simple entre cada uno de los
    predictores frente al resto. Si en alguno de los modelos el *coeficiente de
     determinación R2 es alto, estaría señalando a una posible colinealidad.
#
# Tolerancia (TOL) y Factor de Inflación de la Varianza (VIF). Se trata de dos
    parámetros que vienen a cuantificar lo mismo (uno es el inverso del otro).
    El VIF de cada predictor se calcula según la siguiente fórmula:
#
#  VIF =1/(1 R 2^2)
```

```
#
# Tolerancia=1/VIF
#
# Esta es la opción más recomendada, los límites de referencia que se suelen
    emplear son:
#
# VIF = 1: ausencia total de colinealidad
#
# 1 < VIF < 5: la regresión puede verse afectada por cierta colinealidad.
#
# 5 < VIF < 10: la regresión puede verse altamente afectada por cierta
    colinealidad.
#
# El término tolerancia es 1VIF por lo que los límites recomendables están
    entre 1 y 0.1.
#
# En caso de encontrar colinealidad entre predictores, hay dos posibles
    soluciones. La primera es excluir uno de los predictores problemáticos
    intentando conservar el que, a juicio del investigador, está influyendo
    realmente en la variable respuesta. Esta medida no suele tener mucho
    impacto en el modelo en cuanto a su capacidad predictiva ya que, al existir
     colinealidad, la información que aporta uno de los predictores es
    redundante en presencia del otro. La segunda opción consiste en combinar
    las variables colineales en un único predictor, aunque con el riesgo de
    perder su interpretación.
#
# Cuando se intenta establecer relaciones causa-efecto, la colinealidad puede
    llevar a conclusiones muy erróneas, haciendo creer que una variable es la
    causa cuando, en realidad, es otra la que está influenciando sobre ese
    predictor.

# ### A continuación se comprobará el VIF de las variables que hemos obtenido
    mediante la selección de características

# ['Mach', 'windspeed', 'pitch', 'altitude', 'rpm media', 'wind direction*']

# In[ ]:


lm_n = smf.ols(formula="Mach~windspeed+pitch+altitude+rpmmedia+winddirection",
    data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="windspeed~Mach+pitch+altitude+rpmmedia+winddirection",
    data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:
```

```
lm_n = smf.ols(formula="pitch~windspeed+Mach+altitude+rpmmedia+winddirection",
    data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="altitude~windspeed+pitch+Mach+rpmmedia+winddirection",
    data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="rpmmedia~windspeed+pitch+altitude+Mach+winddirection",
    data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="winddirection~windspeed+pitch+altitude+rpmmedia+Mach",
    data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# # REGRESIÓN LINEAL MÚLTIPLE (MACH, WINDSPEED, PITCH, ALTITUDE, rpm MEDIA,
    WIND DIRECTION)

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection
    ']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)
```

```
# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch', 'altitude', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
```

```
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch','rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed','altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'pitch', 'altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))
```

```
# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['pitch', 'altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)
```

```
from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['pitch', 'altitude', 'rpmmedia']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# # MODELO ANTERIOR CON CL

# Se observa que la variable CL tienen ningún VIF mayor de 5 con ninguna de las
     variables seleccionadas. Así que se añade el CL al modelo para ver cómo
    afecta a su eficiencia.

# In[ ]:


lm_n = smf.ols(formula="Mach~windspeed+pitch+altitude+rpmmedia+winddirection+CL
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:
```

```
lm_n = smf.ols(formula="windspeed~Mach+pitch+altitude+rpmmedia+winddirection+CL
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="pitch~windspeed+Mach+altitude+rpmmedia+winddirection+CL
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="altitude~windspeed+pitch+Mach+rpmmedia+winddirection+CL
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="rpmmedia~windspeed+pitch+altitude+Mach+winddirection+CL
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="winddirection~windspeed+pitch+altitude+rpmmedia+Mach+CL
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[ ]:


lm_n = smf.ols(formula="CL~windspeed+pitch+altitude+rpmmedia+winddirection+Mach
    ", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF
```

```
# In[9]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', '
    winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))

reg = LinearRegression().fit(X,y)
y_hat = reg.predict(X)

print('error absoluto',mean_absolute_error(y, y_hat))
print('error medio cuadrado',mean_squared_error(y,y_hat))
print('R2',r2_score(y, y_hat))


# ### Se elimina la variable Wind Direction:

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# ### Se elimina la variable rpm media:

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','Mach', 'windspeed', 'pitch', 'altitude', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)
```

```
# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'pitch', 'altitude', 'winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# ### Se elimina la variable Altitude:

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','Mach', 'windspeed', 'pitch', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)
```

```
from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'pitch', 'rpmmedia', 'winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# ### Se elimina la variable Pitch:

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','Mach', 'windspeed', 'altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))
```

```
# In[ ]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'altitude', 'rpmmedia', 'winddirection
        ']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# ### Se elimina la variable Wind Speed:

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','Mach', 'pitch', 'altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:
```

```
for i in range(0,6):
    variables=['CL','Mach', 'pitch', 'altitude', 'rpmmedia', 'winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# ### Se elimina la variable Mach:

# In[ ]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['CL','windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[ ]:


for i in range(0,6):
    variables=['CL','windspeed', 'pitch', 'altitude', 'rpmmedia', '
        winddirection']
    uno=variables[i]
    variables.pop(i)
```

```
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# # MEJORA DEL MODELO CON FILTRACIÓN DE DATOS

# En este apartado se pretende volver a implementar el modelo anterior pero
    haciendo una limpieza de datos de forma que no se incluyan ángulos de
    ataque demasiado grandes.

# In[7]:


k = int(np.ceil(1+np.log2(3333)))
plt.hist(datai["AOA1"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Ángulo de ataque")
plt.ylabel("Frecuencia")
plt.title("Histograma de frecuencias del ángulo de ataque")


# In[10]:


datai2 = datai.drop(datai[datai['AOA1']>11].index)


# In[11]:


k = int(np.ceil(1+np.log2(3333)))
plt.hist(datai2["AOA1"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Ángulo de ataque")
plt.ylabel("Frecuencia")
plt.title("Histograma de frecuencias del ángulo de ataque")


# In[12]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai2[['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', '
    winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
```

```
y = datai2["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[13]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', '
        winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai2[dos]
    y = datai2[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[14]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai2[['CL','Mach', 'windspeed', 'pitch', 'altitude', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai2["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)
```

```
# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[15]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', '
        winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai2[dos]
    y = datai2[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[16]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai2[['CL','Mach', 'pitch', 'altitude', 'winddirection']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai2["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)
```

```
from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


# In[17]:


for i in range(0,6):
    variables=['CL','Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', '
        winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai2[dos]
    y = datai2[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# # REGRESIÓN LINEAL MÚLTIPLE CON 3 VARIABLES

# ['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', 'CL']

# In[10]:


lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,3)


# In[16]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)
    print(i)
    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print()
    print()
    print()


# In[ ]:
```

11.6.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import RFE

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:
```

```
datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)

datai.columns


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[5]:


corr = datai.corr()
get_ipython().run_line_magic('matplotlib', 'inline')
plt.figure(figsize=(12,10))
sns.heatmap(corr, linewidth=0.5,annot=True,cmap="RdBu");


# # SELECCIÓN DE CARACTERÍSTICAS CON EL MÉTODO selectKbest

# In[6]:


# Aplicando el algoritmo univariante de prueba F.
data_aux=datai[['Mach', 'elevatorsurface', 'aileronsurface', 'ruddersurface','
    windspeed', 'pitch', 'roll', 'slippage', 'altitude', 'CL', 'CD','AOA1', '
    empujetotal', 'rpmmedia', 'winddirection']]
k = 6 # número de atributos a seleccionar
AOA1 = data_aux['AOA1']
entrenar = data_aux.drop(['AOA1'], axis=1)
```

```
columnas = list(entrenar.columns.values)
seleccionadas = SelectKBest(f_classif, k=k).fit(entrenar, AOA1)
atrib = seleccionadas.get_support()
atributos = [columnas[i] for i in list(atrib.nonzero()[0])]
atributos


# # MULTICOLINEALIDAD DE LAS VARIABLES

# In[7]:


lm_n = smf.ols(formula="Mach~windspeed+pitch+altitude+CL+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[8]:


lm_n = smf.ols(formula="windspeed~Mach+pitch+altitude+CL+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[9]:


lm_n = smf.ols(formula="pitch~windspeed+Mach+altitude+CL+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[10]:


lm_n = smf.ols(formula="altitude~windspeed+pitch+Mach+CL+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[11]:


lm_n = smf.ols(formula="CL~windspeed+pitch+altitude+Mach+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF
```

```
# In[12]:


lm_n = smf.ols(formula="empujetotal~windspeed+pitch+altitude+CL+Mach", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# ### Se observa que el VIF más alto es el del CL, por lo que se opta por
    eliminar dicha variable del modelo y volver a comprobar la
    multicolinealidad:

# In[13]:


lm_n = smf.ols(formula="Mach~windspeed+pitch+altitude+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[14]:


lm_n = smf.ols(formula="windspeed~Mach+pitch+altitude+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[15]:


lm_n = smf.ols(formula="pitch~windspeed+Mach+altitude+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[16]:


lm_n = smf.ols(formula="altitude~windspeed+pitch+Mach+empujetotal", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[17]:
```

```
lm_n = smf.ols(formula="empujetotal~windspeed+pitch+altitude+Mach", data =
    datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# ### Comprobamos también los VIF eliminando el Mach:

# In[18]:


lm_n = smf.ols(formula="windspeed~pitch+altitude+CL+empujetotal", data = datai)
    .fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[19]:


lm_n = smf.ols(formula="pitch~windspeed+altitude+CL+empujetotal", data = datai)
    .fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[20]:


lm_n = smf.ols(formula="altitude~pitch+windspeed+CL+empujetotal", data = datai)
    .fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[21]:


lm_n = smf.ols(formula="CL~pitch+altitude+windspeed+empujetotal", data = datai)
    .fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[22]:


lm_n = smf.ols(formula="empujetotal~pitch+altitude+CL+windspeed", data = datai)
    .fit()
rsquared_n = lm_n.rsquared
```

```
VIF = 1/(1-rsquared_n)
VIF


# ### Comprobamos también los VIF eliminando el Pitch:

# In[23]:


lm_n = smf.ols(formula="Mach~windspeed+altitude+empujetotal+CL", data = datai).
    fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[24]:


lm_n = smf.ols(formula="windspeed~Mach+altitude+empujetotal+CL", data = datai).
    fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[25]:


lm_n = smf.ols(formula="altitude~Mach+windspeed+empujetotal+CL", data = datai).
    fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[26]:


lm_n = smf.ols(formula="empujetotal~Mach+windspeed+altitude+CL", data = datai).
    fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[27]:


lm_n = smf.ols(formula="CL~Mach+windspeed+altitude+empujetotal", data = datai).
    fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF
```

```
# # MODELO REGRESIÓN LINEAL MÚLTIPLE (MACH, WIND SPEED, PITCH, ALTITUDE, EMPUJE
     TOTAL)

# In[6]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch', 'altitude', 'empujetotal']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))


reg = LinearRegression().fit(X,y)
y_hat = reg.predict(X)
print('error absoluto',mean_absolute_error(y, y_hat))
print('error medio cuadrado',mean_squared_error(y,y_hat))
print('R2',r2_score(y, y_hat))


# In[29]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch', 'altitude']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))

for i in range(0,4):
    variables=['Mach', 'windspeed', 'pitch', 'altitude']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[30]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'pitch', 'empujetotal']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
```

```
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))

for i in range(0,4):
    variables=['Mach', 'windspeed', 'pitch', 'empujetotal']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[31]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'windspeed', 'empujetotal', 'altitude']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))

for i in range(0,4):
    variables=['Mach', 'windspeed', 'empujetotal', 'altitude']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)
```

```
    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[32]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['Mach', 'empujetotal', 'pitch', 'altitude']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))

for i in range(0,4):
    variables=['Mach', 'empujetotal', 'pitch', 'altitude']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[33]:
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Preparación de datos iniciales
X = datai[['empujetotal', 'windspeed', 'pitch', 'altitude']]
# Para usar todas las variables excepto la variable independiente y
# alguna con correlación muy baja, utilizamos el siguiente comando:
# X=df.drop(["MEDV","CHAS"],axis=1)
# Nota: cuanto más se aproxime a 1 el valor absoluto de la correlación, más
    eficiente será la aproximación
y = datai["AOA1"].values.reshape(-1, 1)

# Hacemos un split de 33%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=100)

# Se entrena el modelo
reg = LinearRegression().fit(X_train,y_train)

# Hacemos las predicciones
y_train_hat = reg.predict(X_train)
y_test_hat = reg.predict(X_test)

from sklearn.metrics import r2_score
# Calculamos el error
print("Entrenamiento", r2_score(y_train, y_train_hat))
print("Prueba", r2_score(y_test, y_test_hat))

for i in range(0,4):
    variables=['empujetotal', 'windspeed', 'pitch', 'altitude']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# # REGRESIÓN LINEAL MÚLTIPLE CON 3 VARIABLES

# ['Mach', 'windspeed', 'pitch', 'altitude', 'CL', 'empujetotal']

# In[ ]:
```

11.7 SVM simple

```
#!/usr/bin/env python
# coding: utf-8
```

```
# In[1]:


from sklearn import svm
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_poisson_deviance


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)

datai.shape


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
```

```
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# In[6]:


datai = datai.drop(datai[(datai['AOA1'] < 5) & (datai['CL'] > 1)].index)

datai = datai.drop(datai[datai['AOA1']>11].index)

lw = 2
plt.figure(figsize=(16,9))
plt.scatter(datai['CL'].values.reshape(-1,1),datai[["AOA1"]].values.reshape
    (-1,1),color="darkorange", label ="data")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.show()


# # SVM AOA-CL LINEAL

# In[9]:


datai = datai.drop(datai[(datai['AOA1'] < 5) & (datai['CL'] > 1)].index)

datai = datai.drop(datai[datai['AOA1']>11].index)


# In[10]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[11]:


from sklearn import svm
C=1e3
regr = svm.SVR(kernel="linear", C=C)
regr.fit(X, Y)
SVR()


# In[12]:
```

```
y_hat=regr.predict(X)


# In[13]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM Lineal")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[14]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-CL RBF

# In[15]:


datai = datai.drop(datai[(datai['AOA1'] < 5) & (datai['CL'] > 1)].index)

datai = datai.drop(datai[datai['AOA1']>11].index)


# In[16]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[17]:


from sklearn import svm
regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[18]:
```

```
y_hat=regr.predict(X)


# In[19]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM RBF")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[20]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-CL POLINOMIAL GRADO 2

# In[5]:


datai = datai.drop(datai[(datai['AOA1'] < 5) & (datai['CL'] > 1)].index)

datai = datai.drop(datai[datai['AOA1']>11].index)


# In[6]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[7]:


from sklearn import svm
regr = svm.SVR(kernel="poly", C=1, degree=2)
regr.fit(X, Y)
SVR()
```

```
# In[8]:


y_hat=regr.predict(X)


# In[9]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM poly degree 2")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[ ]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-CL POLINOMIAL GRADO 3

# In[5]:


datai = datai.drop(datai[(datai['AOA1'] < 5) & (datai['CL'] > 1)].index)

datai = datai.drop(datai[datai['AOA1']>11].index)


# In[6]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[7]:


from sklearn import svm
regr = svm.SVR(kernel="poly", C=1, degree=3)
regr.fit(X, Y)
SVR()
```

```
# In[8]:


y_hat=regr.predict(X)


# In[9]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM poly degree 3")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[10]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# In[ ]:
```

11.7.1 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from sklearn import svm
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR

from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','thrst,_1,lb', 'thrst,_2,
    lb', 'rpm_1,engin', 'rpm_2,engin','ruddr,_surf', '_wind,speed','_wind,__dir
    ','pitch,__deg', '_roll,__deg', 'hding,_true', '_beta,__deg', '__alt,__ind
    ','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg
    2']]

datai=datai.sample(frac=1).reset_index(drop=True)

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','thrst,_1,lb':'
    empuje1', 'thrst,_2,lb':'empuje2', 'rpm_1,engin':'rpm1', 'rpm_2,engin':'rpm
    2','ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','_wind,__dir':'
    winddirection','pitch,__deg':'pitch', '_roll,__deg':'roll', 'hding,_true':'
    heading', '_beta,__deg':'slippage', '__alt,__ind':'altitud','___cl,total':'
    CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2', 'alpha
    ,__deg2':'AOA3'})


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# # SVM AOA-PITCH (poly grado 3)

# In[14]:


datai.shape


# In[14]:


datai = datai.drop(datai[(datai['AOA1'] > 4) & (datai['pitch'] < 2)].index)
```

```
datai.shape


# In[15]:




X=datai['pitch'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[16]:


from sklearn import svm
regr = svm.SVR(kernel="poly", C=1, degree=3)
regr.fit(X, Y)
SVR()


# In[17]:


y_hat=regr.predict(X)


# In[18]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM poly degree 3")
plt.xlabel("pitch")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[19]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# In[49]:


mainpath = "..\datos ascenso y crucero"
```

```
filename = "Data9C_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# # SVM AOA-PITCH (poly grado 2)

# In[6]:


datai = datai.drop(datai[(datai['AOA1'] > 4) & (datai['pitch'] < 2)].index)

datai.shape


# In[7]:


X=datai['pitch'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[8]:


from sklearn import svm
regr = svm.SVR(kernel="poly", C=1, degree=2)
regr.fit(X, Y)
SVR()


# In[9]:


y_hat=regr.predict(X)


# In[10]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM poly degree 2")
plt.xlabel("pitch")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[13]:
```

```
print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-PITCH (RBF)

# In[18]:


datai = datai.drop(datai[(datai['AOA1'] > 4) & (datai['pitch'] < 2)].index)

datai.shape


# In[19]:


X=datai['pitch'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[20]:


from sklearn import svm
regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[21]:


y_hat=regr.predict(X)


# In[22]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("pitch")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[23]:
```

```
print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-PITCH (LINEAL)

# In[5]:


datai = datai.drop(datai[(datai['AOA1'] > 4) & (datai['pitch'] < 2)].index)

datai.shape


# In[6]:


X=datai['pitch'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[10]:


from sklearn import svm
regr = svm.SVR(kernel="linear", C=10)
regr.fit(X, Y)
SVR()


# In[11]:


y_hat=regr.predict(X)


# In[12]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM lineal")
plt.xlabel("pitch")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[13]:
```

```
print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-CL (rbf)

# In[20]:


X=datai['CL'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[21]:


from sklearn import svm

regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[22]:


y_hat=regr.predict(X)


# In[23]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[24]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))
```

```
# In[22]:


X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)


# In[23]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# # SVM AOA-CL (LINEAL)

# In[25]:


X=datai['CL'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[26]:


from sklearn import svm

regr = svm.SVR(kernel="linear", C=1e3)
regr.fit(X, Y)
SVR()
```

```
# In[27]:


y_hat=regr.predict(X)


# In[28]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM lineal")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[29]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-CL (POLY 2)

# In[30]:


X=datai['CL'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[31]:


from sklearn import svm

regr = svm.SVR(kernel="poly", C=1, degree=2)
regr.fit(X, Y)
SVR()


# In[32]:


y_hat=regr.predict(X)
```

```
# In[33]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM poly degree 2")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[34]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# # SVM AOA-CL (POLY 3)

# In[5]:


X=datai['CL'].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[6]:


from sklearn import svm

regr = svm.SVR(kernel="poly", C=1, degree=3)
regr.fit(X, Y)
SVR()


# In[7]:


y_hat=regr.predict(X)


# In[8]:


lw = 2
```

```
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,y_hat, color="navy", lw = lw, label = "SVM poly degree 3")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[9]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# In[10]:


lw = 2
plt.figure(figsize=(16,9))
plt.scatter(datai['CL'].values.reshape(-1,1),datai[["AOA1"]].values.reshape
    (-1,1),color="darkorange", label ="data")
plt.xlabel("CL")
plt.ylabel("AOA")
plt.show()


# In[13]:


lw = 2
plt.figure(figsize=(16,9))
aux=datai.drop(datai[(datai['AOA1'] > 4) & (datai['pitch'] < 2)].index)
plt.scatter(aux['pitch'].values.reshape(-1,1),aux[["AOA1"]].values.reshape
    (-1,1),color="darkorange", label ="data")
plt.xlabel("Pitch")
plt.ylabel("AOA")
plt.show()


# In[ ]:
```

11.8 SVM múltiple

11.8.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[17]:
```

```
from sklearn import svm
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_poisson_deviance
import statsmodels.formula.api as smf


# In[18]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[19]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empuje total']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevator surface','ailrn,_surf':'aileron surface','ruddr,_surf':'
    rudder surface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)

datai.shape


# In[20]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)
```

```
clean_dataset(datai)

datai.head()


# # SVM AOA-(CL, PITCH, MACH)

# In[21]:


lm_n = smf.ols(formula="Mach~CL+pitch", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[22]:


lm_n = smf.ols(formula="CL~Mach+pitch", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[23]:


lm_n = smf.ols(formula="pitch~CL+Mach", data = datai).fit()
rsquared_n = lm_n.rsquared
VIF = 1/(1-rsquared_n)
VIF


# In[24]:


X=datai[["Mach",'pitch','CL']].values.reshape(-3,3)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[25]:


from sklearn import svm

regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[26]:
```

```
y_hat=regr.predict(X)


# In[27]:


print('error absoluto',mean_absolute_error(Y, y_hat))
print('error medio cuadrado',mean_squared_error(Y,y_hat))
print('R2',r2_score(Y, y_hat))


# In[28]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9A_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[29]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[30]:


clean_dataset(data_pruebai)


# In[31]:


X_prueba=data_pruebai[["Mach",'pitch','CL']].values.reshape(-3,3)
```

```
AOA1_pred_prueba= regr.predict(X_prueba)


# In[32]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[ ]:
```

11.8.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from sklearn import svm
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVR

from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error


# In[2]:
```

```
mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','thrst,_1,lb', 'thrst,_2,
    lb', 'rpm_1,engin', 'rpm_2,engin','ruddr,_surf', '_wind,speed','_wind,__dir
    ','pitch,__deg', '_roll,__deg', 'hding,_true', '_beta,__deg', '__alt,__ind
    ','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg
    2']]

datai=datai.sample(frac=1).reset_index(drop=True)

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','thrst,_1,lb':'
    empuje1', 'thrst,_2,lb':'empuje2', 'rpm_1,engin':'rpm1', 'rpm_2,engin':'rpm
    2','ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','_wind,__dir':'
    winddirection','pitch,__deg':'pitch', '_roll,__deg':'roll', 'hding,_true':'
    heading', '_beta,__deg':'slippage', '__alt,__ind':'altitud','___cl,total':'
    CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2', 'alpha
    ,__deg2':'AOA3'})


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

datai.head()


# # SVM AOA-PITCH, MACH

# In[ ]:


X=datai[["Mach",'pitch']].values.reshape(-2,2)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[ ]:
```

```
from sklearn import svm

regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[ ]:


y_hat=regr.predict(X)


# In[ ]:


X_prueba=data_pruebai[["Mach",'pitch']].values.reshape(-2,2)

AOA1_pred_prueba= regr.predict(X_prueba)


# In[ ]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')
```

11.9 Testing

11.9.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8
```

```
# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression



from sklearn.svm import SVR

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import random


from sklearn import svm

from sklearn.model_selection import train_test_split

from time import time


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9A_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:
```

```
datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[6]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)
```

```
clean_dataset(data_pruebai)

datai.head()


# # REGRESIÓN POLINOMIAL GRADO 3 SIMPLE (CL)

# In[7]:


from sklearn.model_selection import train_test_split
X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)



poly = PolynomialFeatures(degree=3)
poly.fit_transform(X)

reg = LinearRegression().fit(poly.transform(X),y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

y_hat = reg.predict(poly.transform(X))

print("R2 :",r2_score(y,y_hat))

AOA1_pred_prueba= reg.predict(poly.transform(X_prueba))

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[10]:
```

```
aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # SVM AOA-CL (RBF)

# In[20]:


datai = datai.drop(datai[(datai['AOA1'] < 5) & (datai['CL'] > 1)].index)

datai = datai.drop(datai[datai['AOA1']>11].index)


# In[21]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[22]:


from sklearn import svm
regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[23]:


X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)


# In[24]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']
```

```
error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[25]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # REGRESIÓN LINEAL MÚLTIPLE CON 2 VARIABLES

# In[11]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,2)
cont=0
```

```
# In[12]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1
    print(i)
    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE CON 3 VARIABLES

# In[27]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
```

```
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,3)
cont=0


# In[28]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1
    print(i)
    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)
```

```
# # REGRESIÓN LINEAL MÚLTPLE CON 4 VARIABLES

# In[29]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,4)
cont=0


# In[30]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1
```

```
    print(i)
    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE CON 5 VARIABLES

# In[31]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,5)

cont=0


# In[32]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba
```

```
    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE CON 6 VARIABLES

# In[33]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,6)

cont=0


# In[34]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
```

```
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # COMPROBACIÓN VIF PARA REGRESIÓN LINEAL MÚLTIPLE

# In[10]:


for i in range(0,5):
    variables=['Mach', 'windspeed', 'altitude', 'rpmmedia', 'CL']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)
```

```
    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[11]:


for i in range(0,4):
    variables=['Mach', 'windspeed', 'rpmmedia', 'CL']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[12]:




for i in range(0,6):
    variables=['Mach', 'windspeed', 'pitch', 'altitude', 'winddirection', 'CL']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# # SVM (rbf) con 2 variables

# In[7]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
```

```
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,2)
cont=0


# In[14]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-2,2)
    regr = svm.SVR()
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-2,2)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# In[ ]:
```

```
Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-2,2)
    regr = svm.SVR(kernel='linear')
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-2,2)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM (rbf) CON 3 VARIABLES

# In[16]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]
```

```
def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,3)
cont=0


# In[18]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-3,3)
    regr = svm.SVR()
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-3,3)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# In[ ]:


Y = datai["AOA1"].values.reshape(-1, 1)
```

```
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-3,3)
    regr = svm.SVR(kernel='linear')
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-3,3)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM (rbf) CON 4 VARIABLES

# In[19]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]
```

```
lista=combinaciones(lista,4)
cont=0


# In[20]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-4,4)
    regr = svm.SVR()
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-4,4)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# In[ ]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-4,4)
```

```
    regr = svm.SVR(kernel='linear')
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-4,4)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM (rbf) CON 5 VARIABLES

# In[7]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,5)
```

```
cont=0


# In[8]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-5,5)
    regr = svm.SVR()
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-5,5)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM (rbf) CON 6 VARIABLES

# In[9]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
```

```
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,6)
cont=0


# In[10]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-6,6)
    regr = svm.SVR()
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-6,6)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)
```

```
# In[ ]:




# In[ ]:
```

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression



from sklearn.svm import SVR

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import random


from sklearn import svm

from sklearn.model_selection import train_test_split

from timeit import timeit

from time import time

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression
from sklearn.svm import LinearSVR

from sklearn.datasets import make_friedman2
from sklearn.gaussian_process import GaussianProcessRegressor
```

```
from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9A_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']
```

```
data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[6]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)

datai.head()


# # SVM LINEAL (2 variables)

# In[20]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,2)
cont=0


# In[21]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
```

```
    X=datai[i].values.reshape(-2,2)
    regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-2,2)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM LINEAL (3 VARIABLES)

# In[22]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]
```

```
lista=combinaciones(lista,3)
cont=0


# In[23]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-3,3)
    regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-3,3)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM LINEAL (4 VARIABLES)

# In[24]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
```

```
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,4)
cont=0


# In[25]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-4,4)
    regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-4,4)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)
```

```
# # SVM LINEAL (5 VARIABLES)

# In[26]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,5)
cont=0


# In[27]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-5,5)
    regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-5,5)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
```

```
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM LINEAL (6 VARIABLES)

# In[28]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,6)
cont=0


# In[29]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-6,6)
    regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-6,6)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']
```

```
    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # SVM LINEAL (7 VARIABLES)

# In[30]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,7)
cont=0


# In[31]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-7,7)
    regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
    regr.fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-7,7)
```

```
    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# ### VIF

# In[9]:


for i in range(0,5):
    variables=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[11]:


for i in range(0,4):
    variables=['Mach', 'windspeed', 'pitch', 'altitude']
```

```
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[13]:


for i in range(0,5):
    variables=['Mach', 'pitch', 'altitude', 'winddirection', 'CL']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# # PROCESO GAUSIANO

# In[11]:


datai = datai.drop(datai[datai['AOA1']>11].index)
X = datai["AOA1"].values.reshape(-1, 1)

lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

AOA_GP=np.arange(0, 11.33, 0.33)

data_GP = pd.DataFrame(AOA_GP, columns = ['AOA_GP'])

AOA_GP=data_GP['AOA_GP'].values.reshape(-1,1)


# In[12]:


Y = datai['Mach'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
```

```
regr.fit(X, Y)
Mach_hat = regr.predict(X)
data_GP['Mach_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,Mach_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("Mach")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[13]:


Y = datai['windspeed'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
regr.fit(X, Y)
windspeed_hat = regr.predict(X)
data_GP['windspeed_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,windspeed_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("Wind Speed")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[14]:


Y = datai['pitch'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
regr.fit(X, Y)
pitch_hat = regr.predict(X)
data_GP['Pitch_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,pitch_hat , color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("Pitch")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[15]:
```

```
Y = datai['altitude'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
regr.fit(X, Y)
altitude_hat = regr.predict(X)
data_GP['Altitude_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,altitude_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("Altitude")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[16]:


Y = datai['rpmmedia'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
regr.fit(X, Y)
rpm_hat = regr.predict(X)
data_GP['rpmmedia_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X,rpm_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("rpm media")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[17]:


Y = datai['winddirection'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
regr.fit(X, Y)
winddirection_hat = regr.predict(X)
data_GP['winddirection_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X, winddirection_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("Wind Direction")
plt.title("Support Vector Regression")
plt.legend()
plt.show()
```

```
# In[18]:


Y = datai['CL'].values.reshape(-1,1)
Y=Y.ravel()
regr = svm.SVR()
regr.fit(X, Y)
CL_hat = regr.predict(X)
data_GP['CL_GP']=regr.predict(AOA_GP)
lw = 2
plt.figure(figsize=(16,9))
plt.scatter(X,Y,color="darkorange", label ="data")
plt.plot(X, CL_hat, color="navy", lw = lw, label = "SVM rbf")
plt.xlabel("AOA")
plt.ylabel("CL")
plt.title("Support Vector Regression")
plt.legend()
plt.show()


# In[26]:


print(data_GP.shape)
data_GP.head()


# # PROCESOS GAUSIANOS 2 VARIABLES

# In[72]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
lista2=['Mach_GP', 'windspeed_GP', 'Pitch_GP', 'Altitude_GP', 'rpmmedia_GP', '
    winddirection_GP', 'CL_GP']
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,2)
lista2=combinaciones(lista2,2)
cont=0


# In[73]:


Y = data_GP["AOA_GP"].values.reshape(-1, 1)
Y=Y.ravel()
```

```
Y_new= data_pruebai["AOA1"].values.reshape(-1, 1)
Y_new=Y_new.ravel()
for i in range(0,len(lista)):
    X=data_GP[lista2[i]].values.reshape(-2,2)
    X_new =data_pruebai[lista[i]].values.reshape(-2,2)
    gpr = GaussianProcessRegressor().fit(X, Y)
    AOA1_pred_prueba=gpr.predict(X_new)


    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(lista[i])
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)




# # PROCESOS GAUSIANOS 3 VARIABLES

# In[74]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
lista2=['Mach_GP', 'windspeed_GP', 'Pitch_GP', 'Altitude_GP', 'rpmmedia_GP', '
    winddirection_GP', 'CL_GP']
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]
```

```
lista=combinaciones(lista,3)
lista2=combinaciones(lista2,3)
cont=0


# In[75]:


Y = data_GP["AOA_GP"].values.reshape(-1, 1)
Y=Y.ravel()
Y_new= data_pruebai["AOA1"].values.reshape(-1, 1)
Y_new=Y_new.ravel()
for i in range(0,len(lista)):
    X=data_GP[lista2[i]].values.reshape(-3,3)
    X_new =data_pruebai[lista[i]].values.reshape(-3,3)
    gpr = GaussianProcessRegressor().fit(X, Y)
    AOA1_pred_prueba=gpr.predict(X_new)


    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(lista[i])
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)




# # PROCESOS GAUSIANOS 4 VARIABLES

# In[76]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
lista2=['Mach_GP', 'windspeed_GP', 'Pitch_GP', 'Altitude_GP', 'rpmmedia_GP', '
    winddirection_GP', 'CL_GP']
```

```
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,4)
lista2=combinaciones(lista2,4)
cont=0


# In[77]:


Y = data_GP["AOA_GP"].values.reshape(-1, 1)
Y=Y.ravel()
Y_new= data_pruebai["AOA1"].values.reshape(-1, 1)
Y_new=Y_new.ravel()
for i in range(0,len(lista)):
    X=data_GP[lista2[i]].values.reshape(-4,4)
    X_new =data_pruebai[lista[i]].values.reshape(-4,4)
    gpr = GaussianProcessRegressor().fit(X, Y)
    AOA1_pred_prueba=gpr.predict(X_new)


    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(lista[i])
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)
```

```
# # PROCESOS GAUSIANOS 5 VARIABLES

# In[78]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
lista2=['Mach_GP', 'windspeed_GP', 'Pitch_GP', 'Altitude_GP', 'rpmmedia_GP', '
    winddirection_GP', 'CL_GP']
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,5)
lista2=combinaciones(lista2,5)
cont=0


# In[79]:


Y = data_GP["AOA_GP"].values.reshape(-1, 1)
Y=Y.ravel()
Y_new= data_pruebai["AOA1"].values.reshape(-1, 1)
Y_new=Y_new.ravel()
for i in range(0,len(lista)):
    X=data_GP[lista2[i]].values.reshape(-5,5)
    X_new =data_pruebai[lista[i]].values.reshape(-5,5)
    gpr = GaussianProcessRegressor().fit(X, Y)
    AOA1_pred_prueba=gpr.predict(X_new)


    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(lista[i])
    print('ACIERTOS =',porcentaje_aciertos,'%')
```

```
    print()
    print()
    print()

print(cont)



# # PROCESOS GAUSIANOS 6 VARIABLES

# In[80]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
lista2=['Mach_GP', 'windspeed_GP', 'Pitch_GP', 'Altitude_GP', 'rpmmedia_GP', '
    winddirection_GP', 'CL_GP']
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,6)
lista2=combinaciones(lista2,6)
cont=0


# In[81]:


Y = data_GP["AOA_GP"].values.reshape(-1, 1)
Y=Y.ravel()
Y_new= data_pruebai["AOA1"].values.reshape(-1, 1)
Y_new=Y_new.ravel()
for i in range(0,len(lista)):
    X=data_GP[lista2[i]].values.reshape(-6,6)
    X_new =data_pruebai[lista[i]].values.reshape(-6,6)
    gpr = GaussianProcessRegressor().fit(X, Y)
    AOA1_pred_prueba=gpr.predict(X_new)


    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
```

```
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(lista[i])
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # PROCESOS GAUSIANOS 7 VARIABLES

# In[82]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
lista2=['Mach_GP', 'windspeed_GP', 'Pitch_GP', 'Altitude_GP', 'rpmmedia_GP', '
    winddirection_GP', 'CL_GP']
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,7)
lista2=combinaciones(lista2,7)
cont=0


# In[83]:


Y = data_GP["AOA_GP"].values.reshape(-1, 1)
Y=Y.ravel()
Y_new= data_pruebai["AOA1"].values.reshape(-1, 1)
Y_new=Y_new.ravel()
for i in range(0,len(lista)):
    X=data_GP[lista2[i]].values.reshape(-7,7)
    X_new =data_pruebai[lista[i]].values.reshape(-7,7)
    gpr = GaussianProcessRegressor().fit(X, Y)
    AOA1_pred_prueba=gpr.predict(X_new)


    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']
```

```
    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(lista[i])
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# In[ ]:
```

```
#!/usr/bin/env python
# coding: utf-8

# In[8]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from sklearn.linear_model import LinearRegression


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)
```

```
# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9A_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)
```

```
# In[6]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)

datai.head()


# # REGRESIÓN MEDIANTE REDES NEURONALES (2 VARIABLES)

# In[37]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']
def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,2)
cont=0


# In[38]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-2,2)
    regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-2,2)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba
```

```
    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN MEDIANTE REDES NEURONALES (3 VARIABLES)

# In[40]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,3)
cont=0


# In[41]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-3,3)
    regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)
```

```
    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-3,3)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN MEDIANTE REDES NEURONALES (4 VARIABLES)

# In[42]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,4)
cont=0
```

```
# In[43]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-4,4)
    regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-4,4)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN MEDIANTE REDES NEURONALES (5 VARIABLES)

# In[44]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
```

```
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista, 5)
cont=0


# In[45]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-5,5)
    regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-5,5)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN MEDIANTE REDES NEURONALES (6 VARIABLES)

# In[46]:
```

```
datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista, 6)
cont=0


# In[47]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-6,6)
    regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-6,6)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]

    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
```

```
    print()

print(cont)


# # REGRESIÓN MEDIANTE REDES NEURONALES (6 VARIABLES)

# In[48]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia', 'winddirection', '
    CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista, 7)
cont=0


# In[49]:


Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()
for i in lista:
    X=datai[i].values.reshape(-7,7)
    regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

    y_hat=regr.predict(X)

    X_prueba=data_pruebai[i].values.reshape(-7,7)

    AOA1_pred_prueba= regr.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
```

```
    cont=cont+1

    print(i)
    print('error absoluto',mean_absolute_error(Y, y_hat))
    print('error medio cuadrado',mean_squared_error(Y,y_hat))
    print('R2 total',r2_score(Y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# In[10]:


for i in range(0,3):
    variables=['windspeed', 'altitude', 'winddirection']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[11]:


for i in range(0,3):
    variables=['Mach', 'pitch', 'CL']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[13]:


for i in range(0,4):
    variables=['windspeed', 'Mach', 'pitch', 'CL']
```

```
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[ ]:
```

11.9.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.svm import SVR

import random
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# In[2]:


mainpath = "..\datos ascenso y crucero"
```

```
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9C_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})
```

```
data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[6]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)

datai.head()


# # REGRESIÓN LINEAL SIMPLE (CL)

# In[85]:


from sklearn.model_selection import train_test_split
X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)



poly = PolynomialFeatures(degree=1)
poly.fit_transform(X)

reg = LinearRegression().fit(poly.transform(X),y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

y_hat = reg.predict(poly.transform(X))

print("R2 :",r2_score(y,y_hat))

AOA1_pred_prueba= reg.predict(poly.transform(X_prueba))


# In[86]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba


# In[87]:


porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']
```

```
error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


# In[88]:


porcentaje.describe()


# In[89]:


error_absoluto.describe()


# In[90]:


error_relativo.describe()


# In[91]:


prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print(porcentaje_aciertos)


# In[93]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # REGRESIÓN POLINOMIAL GRADO 2 SIMPLE (PITCH)

# In[94]:


from sklearn.model_selection import train_test_split
X = datai["pitch"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)
```

```
poly = PolynomialFeatures(degree=2)
poly.fit_transform(X)

reg = LinearRegression().fit(poly.transform(X),y)

X_prueba=data_pruebai["pitch"].values.reshape(-1,1)

y_hat = reg.predict(poly.transform(X))

print("R2 :",r2_score(y,y_hat))

AOA1_pred_prueba= reg.predict(poly.transform(X_prueba))


# In[95]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


# In[96]:


porcentaje.describe()


# In[97]:


error_absoluto.describe()


# In[98]:


error_relativo.describe()


# In[99]:


prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
porcentaje_aciertos
```

```
# In[100]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # REGRESIÓN LINEAL MÚLTIPLE (Mach,Wind Speed, Pitch, Altitude, Empuje total)

# In[101]:


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = datai[['Mach', 'windspeed','pitch','altitude','empujetotal']]

y = datai["AOA1"].values.reshape(-1, 1)

reg = LinearRegression().fit(X,y)

y_hat = reg.predict(X)

from sklearn.metrics import r2_score
# Calculamos el error
print("R2", r2_score(y, y_hat))

X_prueba=data_pruebai[['Mach', 'windspeed','pitch','altitude','empujetotal']]

AOA1_pred_prueba = reg.predict(X_prueba)


# In[102]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


# In[103]:


porcentaje.describe()
```

```
# In[104]:


error_absoluto.describe()


# In[105]:


error_relativo.describe()


# In[106]:


prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
porcentaje_aciertos


# In[111]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # REGRESIÓN LINEAL MÚLTIPLE PARA 2 VARIABLES

# In[16]:


datai = datai.drop(datai[datai['AOA1']>11].index)

lista=['Mach', 'windspeed', 'pitch', 'altitude', 'empujetotal', 'CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]
```

```
lista=combinaciones(lista,2)
cont=0


# In[17]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
    variables=i
    print(i)

    cont=cont+1

    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE PARA 3 VARIABLES

# In[7]:
```

```
datai = datai.drop(datai[datai['AOA1']>11].index)

lista=['Mach', 'windspeed', 'pitch', 'altitude', 'empujetotal', 'CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,3)
cont=0


# In[8]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
    variables=i
    print(i)

    cont=cont+1

    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
```

```
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE 4 VARIABLES

# In[9]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'empujetotal', 'CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,4)
cont=0


# In[10]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']
```

```
    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
    variables=i
    print(i)

    cont=cont+1

    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE 5 VARIABLES

# In[11]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'empujetotal', 'CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,5)
cont=0


# In[12]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
```

```
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
    variables=i
    print(i)

    cont=cont+1

    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # REGRESIÓN LINEAL MÚLTIPLE 6 VARIABLES

# In[13]:


datai = datai.drop(datai[datai['AOA1']>11].index)
lista=['Mach', 'windspeed', 'pitch', 'altitude', 'empujetotal', 'CL']

def potencia(c):
    if len(c) == 0:
        return [[]]
    r = potencia(c[:-1])
    return r + [s + [c[-1]] for s in r]

def combinaciones(c, n):
    return [s for s in potencia(c) if len(s) == n]


lista=combinaciones(lista,6)
cont=0
```

```
# In[14]:


y = datai["AOA1"].values.reshape(-1, 1)
for i in lista:
    X = datai[i]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
        random_state=100)
    reg = LinearRegression().fit(X_train,y_train)
    y_train_hat = reg.predict(X_train)
    y_test_hat = reg.predict(X_test)
    reg = LinearRegression().fit(X,y)
    y_hat = reg.predict(X)

    X_prueba=data_pruebai[i]
    AOA1_pred_prueba = reg.predict(X_prueba)

    data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

    porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

    error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

    error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

    prueba=[error_absoluto<0.25]
    prueba2=sum(prueba)
    aciertos=sum(prueba2)
    aciertos
    porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
    variables=i
    print(i)

    cont=cont+1

    print("Entrenamiento", r2_score(y_train, y_train_hat))
    print("Prueba", r2_score(y_test, y_test_hat))
    print('error absoluto',mean_absolute_error(y, y_hat))
    print('error medio cuadrado',mean_squared_error(y,y_hat))
    print('R2 total',r2_score(y, y_hat))
    print('ACIERTOS =',porcentaje_aciertos,'%')
    print()
    print()
    print()

print(cont)


# # COMPROBACIÓ VIF

# In[7]:


for i in range(0,2):
    variables=['Mach', 'pitch']
```

```
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[8]:




for i in range(0,2):
    variables=['Mach', 'altitude']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# In[10]:


for i in range(0,3):
    variables=['Mach', 'altitude','pitch']
    uno=variables[i]
    variables.pop(i)
    dos=variables
    X = datai[dos]
    y = datai[uno].values.reshape(-1, 1)

    reg = LinearRegression().fit(X,y)

    y_hat = reg.predict(X)
    R2=r2_score(y, y_hat)
    VIF = 1/(1-R2)
    print(uno,VIF)


# # SVM AOA-PITCH (rbf)

# In[112]:
```

```
X=datai[["pitch"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[113]:


from sklearn import svm
regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[114]:


X_prueba=data_pruebai["pitch"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)


# In[115]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[116]:
```

```
aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # SVM AOA-CL (rbf)

# In[117]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[118]:


from sklearn import svm
regr = svm.SVR()
regr.fit(X, Y)
SVR()


# In[119]:


X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)


# In[120]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
```

```
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[121]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r')
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b')


# # SVM AOA-PITCH (lineal)

# In[11]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

print('X shape is',X.shape,'and Y shape is',Y.shape)


# In[12]:


from sklearn import svm
regr = svm.SVR(kernel='linear')
regr.fit(X, Y)
SVR()


# In[13]:


X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)
```

```
# In[14]:


data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[ ]:
```

11.10 Learning curves

11.10.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from sklearn.model_selection import learning_curve
from sklearn.svm import SVC

import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

```
from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[4]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)


# In[5]:


def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
    n_jobs=None, train_sizes=[2,5]):
    if axes is None:
```

```
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ =   learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs,
                    train_sizes=train_sizes,
                    return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")

    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt


# # MODELO 1

# In[17]:
```

```
X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

estimator=LinearRegression()
title='Learning curves: regresión polinomial grado 3 (CL)'
cv=5
train_sizes=[5000, 10000, 20000, 29000, 37000, 44000]


# In[18]:


plot_learning_curve(estimator, title, X=X, y=y, axes=None, ylim=None, cv=5, n_
    jobs=None, train_sizes=[5000, 10000, 20000, 29000, 37000, 44000])


# # MODELO 2

# In[19]:


datai = datai.drop(datai[datai['AOA1']>11].index)
X = datai[['Mach', 'windspeed', 'altitude', 'rpmmedia', 'CL']]
y = datai["AOA1"].values.reshape(-1, 1)
estimator=LinearRegression()
title='Learning curves: regresión lineal múltiple (CL, Mach, Wind Speed,
    altitud, rpm media)'
cv=5
train_sizes=[5000, 10000, 20000, 29000, 37000, 44000]


# In[20]:


plot_learning_curve(estimator, title, X=X, y=y, axes=None, ylim=None, cv=5, n_
    jobs=None, train_sizes=[5000, 10000, 20000, 29000, 37000, 44000])


# # MODELO 3

# In[22]:


X = datai["CL"].values.reshape(-1,1)
Y = datai["AOA1"].values.reshape(-1, 1)

Y=Y.ravel()

estimator=SVR()
title='Learning curves: SVM rbf (CL)'
cv=5
train_sizes=[5000, 10000, 20000, 29000, 37000, 44000]


# In[23]:
```

```
plot_learning_curve(estimator, title, X=X, y=Y, axes=None, ylim=None, cv=5, n_
    jobs=None, train_sizes=[5000, 10000, 20000, 29000, 37000, 44000])


# # MODELO 4

# In[8]:


X = datai[['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia']].values.
    reshape(-5,5)
Y = datai["AOA1"].values.reshape(-1, 1)

Y=Y.ravel()

estimator=SVR()
title='Learning curves: SVM lineal multiple (Mach, Wind Speed, pitch, altitud,
    rpm media)'
cv=5
train_sizes=[5000, 10000, 20000, 29000, 37000, 44000]


# In[9]:


plot_learning_curve(estimator, title, X=X, y=Y, axes=None, ylim=None, cv=5, n_
    jobs=None, train_sizes=[5000, 10000, 20000, 29000, 37000, 44000])


# # MODELO 5

# In[8]:


X = datai[['Mach', 'windspeed', 'pitch', 'CL']].values.reshape(-4,4)
Y = datai["AOA1"].values.reshape(-1, 1)

Y=Y.ravel()

estimator=MLPRegressor(random_state=1, max_iter=500)
title='Learning curves: redes neuronales (Mach, Wind Speed, pitch, CL)'
cv=5
train_sizes=[5000, 10000, 20000, 29000, 37000, 44000]


# In[10]:


plot_learning_curve(estimator, title, X=X, y=Y, axes=None, ylim=None, cv=5, n_
    jobs=None, train_sizes=[5000, 10000, 20000, 29000, 37000, 44000])


# In[ ]:
```

11.10.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


from sklearn.model_selection import learning_curve
from sklearn.svm import SVC

import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.svm import SVR


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
```

```
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)


# # FUNCIÓN PARA PLOTEAR LAS CURVAS

# In[12]:


def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
    n_jobs=None, train_sizes=[2,5]):
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ =   learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs,
                    train_sizes=train_sizes,
                    return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
```

```
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")

    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt


# https://scikit-learn.org/stable/modules/learning_curve.html
#
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.
    learning_curve.html#sklearn.model_selection.learning_curve
#
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_
    curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py

# # REGRESIÓN LINEAL SIMPLE (CL)

# In[17]:


X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)

estimator=LinearRegression()
title='Learning curves: regresión lineal simple (CL)'
cv=5
train_sizes=[5000, 10000, 20000, 29000]


# In[18]:


plot_learning_curve(estimator=LinearRegression(), title='Learning curves:
    regresión lineal simple (CL)', X=X, y=y, axes=None, ylim=None, cv=5, n_jobs
    =None, train_sizes=[5000, 10000, 20000, 29000])


# # REGRESIÓN LINEAL MÚLTIPLE
```

```
# In[8]:


X = datai[["Mach", 'pitch']].values.reshape(-2,2)
y = datai["AOA1"].values.reshape(-1, 1)

estimator=LinearRegression()
title='Learning curves: regresión lineal múltiple (Mach, pitch)'
cv=5
train_sizes=[5000, 10000, 20000, 29000]


# In[9]:


plot_learning_curve(estimator=LinearRegression(), title='Learning curves:
    regresión lineal múltiple (Mach, pitch)', X=X, y=y, axes=None, ylim=None,
    cv=5, n_jobs=None, train_sizes=[5000, 10000, 20000, 29000])


# # SVM (CL)

# In[10]:


X = datai["CL"].values.reshape(-1,1)
Y = datai["AOA1"].values.reshape(-1, 1)

Y=Y.ravel()

estimator=SVR()
title='Learning curves: SVM rbf (CL)'
cv=5
train_sizes=[5000, 10000, 20000, 29000]


# In[11]:


plot_learning_curve(estimator, title, X=X, y=y, axes=None, ylim=None, cv=5, n_
    jobs=None, train_sizes=[5000, 10000, 20000, 29000])


# In[ ]:
```

11.11 Mejores modelos

11.11.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:
```

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression



from sklearn.svm import SVR

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import random

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression
from sklearn.svm import LinearSVR

from sklearn import svm


from sklearn.neural_network import MLPRegressor


# In[ ]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[ ]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9A_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[ ]:
```

```
datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[ ]:


data_pruebaii=data_prueba[['muestra','_Mach,ratio','_elev,_surf','ailrn,_surf
    ','ruddr,_surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg',
     '__alt,__ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', '
    alpha,__deg2']]

data_pruebaii['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,
    lb']
data_pruebaii['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebaii['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebaii= data_pruebaii.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})

data_pruebaii = data_pruebaii.drop(data_pruebaii[(data_pruebaii['AOA1'] > 0.2)
    & (data_pruebaii['muestra'] < 100)].index)


# In[ ]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
```

```
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[ ]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)

datai.head()


# # VISUALIZACIÓN DATOS DEL NUEVO VUELO

# In[23]:


data_pruebaii.plot("muestra","AOA1")


# In[20]:


data_pruebaii.plot("muestra","altitude")


# In[21]:


data_pruebaii.plot("muestra","Mach")


# In[24]:


k = int(np.ceil(1+np.log2(3333)))
```

```
plt.hist(data_pruebai["AOA1"], bins = k) #bins = [0,30,60,...,200]
plt.xlabel("Ángulo de ataque")
plt.ylabel("Frecuencia")
plt.title("Histograma de frecuencias del ángulo de ataque")


# # MODELO 1: REGRESIÓN POLINOMIAL SIMPLE DE GRADO 3 (CL)

# In[25]:


from sklearn.model_selection import train_test_split
X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)



poly = PolynomialFeatures(degree=3)
poly.fit_transform(X)

reg = LinearRegression().fit(poly.transform(X),y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

y_hat = reg.predict(poly.transform(X))

print("R2 :",r2_score(y,y_hat))

AOA1_pred_prueba= reg.predict(poly.transform(X_prueba))

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[36]:
```

```
aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# # MODELO 2: REGRESIÓN LINEAL MÚLTIPLE

# In[52]:


datai = datai.drop(datai[datai['AOA1']>11].index)
X = datai[['Mach', 'windspeed', 'altitude', 'rpmmedia', 'CL']]


y = datai["AOA1"].values.reshape(-1, 1)



reg = LinearRegression().fit(X,y)


X_prueba=data_pruebai[['Mach', 'windspeed', 'altitude', 'rpmmedia', 'CL']]

AOA1_pred_prueba = reg.predict(X_prueba)

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())
```

```
prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[53]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# In[104]:


vector1=prueba[0]

vector1[2972]=True
vector1[3532]=True
vector1[3791]=True
vector1[5780]=True
vector1[5830]=True

vector2=np.zeros(7000)

t=1
f=0
cont=0

for i in range(0,len(vector1)):

    if vector1[i]==True:

        if t==0 and f==1:
            cont=cont+1

        vector2[cont]=vector2[cont]+1

        t=1
        f=0

    if vector1[i]==False:
```

```
        if t==1 and f==0:
            cont=cont+1

        vector2[cont]=vector2[cont]+1

        f=1
        t=0

def elimina_ceros(original):
    nueva = []
    for dato in original:
        if dato != 0:
            nueva.append(dato)
    return nueva



vector2 = elimina_ceros(vector2)


# # MODELO 3: SVM SIMPLE

# In[14]:


datai = datai.drop(datai[datai['AOA1']>11].index)

X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

regr = svm.SVR()
regr.fit(X, Y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
```

```
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[16]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# # MODELO 4: SVM MÚLTIPLE

# In[10]:


datai = datai.drop(datai[datai['AOA1']>11].index)
i=['Mach', 'windspeed', 'pitch', 'altitude', 'rpmmedia']
Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()

X=datai[i].values.reshape(-5,5)
regr = make_pipeline(StandardScaler(),LinearSVR(random_state=0, tol=1e-5))
regr.fit(X, Y)

y_hat=regr.predict(X)

X_prueba=data_pruebai[i].values.reshape(-5,5)

AOA1_pred_prueba= regr.predict(X_prueba)

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
```

```
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')



# In[12]:


aux1=np.zeros(30)
for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# # MODELO 5: REDES NEURONALES

# In[11]:


datai = datai.drop(datai[datai['AOA1']>11].index)
i=['Mach', 'windspeed', 'pitch', 'CL']
Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()

X=datai[i].values.reshape(-4,4)
regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

y_hat=regr.predict(X)

X_prueba=data_pruebai[i].values.reshape(-4,4)

AOA1_pred_prueba= regr.predict(X_prueba)

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

print('PORCENTAJE')
print(porcentaje.describe())
```

```
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[12]:


aux1=np.zeros(30)
for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")
```

11.11.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[15]:


import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.preprocessing import PolynomialFeatures
get_ipython().run_line_magic('matplotlib', 'inline')

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.svm import SVR
```

```
from sklearn import svm

import random
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9C_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


data_pruebaii=data_prueba[['muestra','_Mach,ratio','_elev,_surf','ailrn,_surf
    ','ruddr,_surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg',
     '__alt,__ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', '
    alpha,__deg2']]
```

```
data_pruebaii['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,
    lb']
data_pruebaii['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    ']])/2
data_pruebaii['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebaii= data_pruebaii.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})

data_pruebaii = data_pruebaii.drop(data_pruebaii[(data_pruebaii['muestra'] <
    10)].index)


# In[6]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    ']])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[7]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)
```

```
datai.head()


# # VISUALIZACIÓN DE DATOS DEL NUEVO VUELO

# In[26]:


data_pruebaii.plot("muestra","AOA1")


# In[22]:


data_pruebaii.plot("muestra","altitude")


# In[23]:


data_pruebaii.plot("muestra","Mach")


# # MODELO 1: REGRESIÓN SIMPLE

# In[10]:



X = datai["CL"].values.reshape(-1,1)
y = datai["AOA1"].values.reshape(-1, 1)



poly = PolynomialFeatures(degree=1)
poly.fit_transform(X)

reg = LinearRegression().fit(poly.transform(X),y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

y_hat = reg.predict(poly.transform(X))

AOA1_pred_prueba = reg.predict(poly.transform(X_prueba))

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
```

```
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[11]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# # MODELO 2: REGRESIÓN LINEAL MÚLTIPLE

# In[12]:



X = datai[['Mach', 'pitch']]


y = datai["AOA1"].values.reshape(-1, 1)



reg = LinearRegression().fit(X,y)


X_prueba=data_pruebai[['Mach', 'pitch']]

AOA1_pred_prueba = reg.predict(X_prueba)
```

```
data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[13]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# # MODELO 3: SVM SIMPLE

# In[16]:



X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)
```

```
Y=Y.ravel()

regr = svm.SVR()
regr.fit(X, Y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba

porcentaje=100*data_pruebai['AOA1']/data_pruebai['AOA1_pred_prueba']

error_absoluto=abs(data_pruebai['AOA1']-data_pruebai['AOA1_pred_prueba'])

error_relativo=100*error_absoluto/data_pruebai['AOA1_pred_prueba']


print('PORCENTAJE')
print(porcentaje.describe())
print('ERROR ABSOLUTO')
print(error_absoluto.describe())
print('ERROR RELATIVO')
print(error_relativo.describe())

prueba=[error_absoluto<0.25]
prueba2=sum(prueba)
aciertos=sum(prueba2)
aciertos
porcentaje_aciertos=100*aciertos/error_absoluto.shape[0]
print('ACIERTOS =',porcentaje_aciertos,'%')


# In[17]:


aux1=np.zeros(30)

for i in range(0,30):
    aux1[i]=random.randint(0, data_pruebai.shape[0])

aux=data_pruebai.iloc[aux1,:]
aux=aux.reset_index()
aux['index1']=aux.index
plt.scatter(aux['index1'], aux['AOA1'], c='r',label="Real")
plt.scatter(aux['index1'], aux['AOA1_pred_prueba'], c='b',label="Modelo")
plt.xlabel("muestra")
plt.ylabel("AOA")
plt.legend(loc="upper right")


# In[ ]:
```

11.12 Aplicación en diferentes casos

11.12.1 Despegue y ascenso

```
#!/usr/bin/env python
# coding: utf-8

# In[12]:


import pandas as pd
import os
import numpy as np

from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor


# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataA_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9A_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)
```

```
# In[6]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[7]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)

datai.head()


# # Creación regresión

# ### Regresión lineal múltiple

# In[11]:


datai = datai.drop(datai[datai['AOA1']>11].index)
X = datai[['Mach', 'windspeed', 'altitude', 'rpmmedia', 'CL']]


y = datai["AOA1"].values.reshape(-1, 1)
```

```
reg = LinearRegression().fit(X,y)


X_prueba=data_pruebai[['Mach', 'windspeed', 'altitude', 'rpmmedia', 'CL']]

AOA1_pred_prueba = reg.predict(X_prueba)

data_pruebai['AOA1_pred_prueba1']=AOA1_pred_prueba


# ### Redes neuronales

# In[13]:


datai = datai.drop(datai[datai['AOA1']>11].index)
i=['Mach', 'windspeed', 'pitch', 'CL']
Y = datai["AOA1"].values.reshape(-1, 1)
Y=Y.ravel()

X=datai[i].values.reshape(-4,4)
regr = MLPRegressor(random_state=1, max_iter=500).fit(X, Y)

X_prueba=data_pruebai[i].values.reshape(-4,4)

AOA1_pred_prueba= regr.predict(X_prueba)

data_pruebai['AOA1_pred_prueba2']=AOA1_pred_prueba


# In[14]:


data_pruebai.iloc[600:650,:]


# In[ ]:
```

11.12.2 Crucero

```
#!/usr/bin/env python
# coding: utf-8

# In[7]:


import pandas as pd
import os
import numpy as np

from sklearn.svm import SVR
from sklearn import svm
```

```
# In[2]:


mainpath = "..\datos ascenso y crucero"
filename = "DataC_cleaned.xlsx"
data = pd.read_excel(mainpath + "/" + filename)


# In[3]:


mainpath = "..\datos ascenso y crucero"
filename = "Data9C_cleaned.xlsx"
data_prueba = pd.read_excel(mainpath + "/" + filename)


# In[4]:


datai=data[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_surf', '_wind,
    speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__ind','___cl,
    total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__deg2']]

datai['empujetotal']=data['thrst,_1,lb']+data['thrst,_2,lb']
datai['rpmmedia']=(data['rpm_1,engin']+data['rpm_1,engin'])/2
datai['winddirection']=data['_wind,__dir']-data['hding,_true']

datai = datai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio':'Mach','_elev
    ,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','ruddr,_surf':'
    ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'pitch', '_roll,__
    deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'altitude','___cl,
    total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','alpha,__deg1':'AOA2',
     'alpha,__deg2':'AOA3'})



datai=datai.sample(frac=1).reset_index(drop=True)


# In[5]:


data_pruebai=data_prueba[['_Mach,ratio','_elev,_surf','ailrn,_surf','ruddr,_
    surf', '_wind,speed','pitch,__deg', '_roll,__deg', '_beta,__deg', '__alt,__
    ind','___cl,total','___cd,total', 'alpha,__deg','alpha,__deg1', 'alpha,__
    deg2']]

data_pruebai['empujetotal']=data_prueba['thrst,_1,lb']+data_prueba['thrst,_2,lb
    ']
data_pruebai['rpmmedia']=(data_prueba['rpm_1,engin']+data_prueba['rpm_1,engin
    '])/2
data_pruebai['winddirection']=data_prueba['_wind,__dir']-data_prueba['hding,_
    true']

data_pruebai= data_pruebai.rename(columns={'pitch,__deg':'pitch','_Mach,ratio
    ':'Mach','_elev,_surf':'elevatorsurface','ailrn,_surf':'aileronsurface','
    ruddr,_surf':'ruddersurface', '_wind,speed':'windspeed','pitch,__deg':'
```

```
    pitch', '_roll,__deg':'roll', '_beta,__deg':'slippage', '__alt,__ind':'
    altitude','___cl,total':'CL','___cd,total':'CD', 'alpha,__deg':'AOA1','
    alpha,__deg1':'AOA2', 'alpha,__deg2':'AOA3'})



data_pruebai=data_pruebai.sample(frac=1).reset_index(drop=True)


# In[6]:


def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)
    return df[indices_to_keep].astype(np.float64)

clean_dataset(datai)

clean_dataset(data_pruebai)

datai.head()


# # Creación regresión

# ### SVM simple

# In[8]:


X=datai[["CL"]].values.reshape(-1,1)
Y=datai[["AOA1"]].values.reshape(-1,1)

Y=Y.ravel()

regr = svm.SVR()
regr.fit(X, Y)

X_prueba=data_pruebai["CL"].values.reshape(-1,1)

AOA1_pred_prueba= regr.predict(X_prueba)

data_pruebai['AOA1_pred_prueba']=AOA1_pred_prueba


# In[ ]:
```

Bibliografía

13 Sitios web

[1] Scikits Learn: https://scikit-learn.org/stable/
[2] GitHub: https://github.com/
[3] Stack Overflow: https://es.stackoverflow.com/
[4] Pandas: https://pandas.pydata.org/pandas-docs/stable/index.html
[5] Matplotlib: https://matplotlib.org/
[6] NumPy: https://numpy.org/
[7] X-Plane 11: https://www.x-plane.com/

14 Libros

[8] Fauzia Ahmad: Big Data: Learning, Analytics, and Applications.
[9] David Ríos Insua y David Gómez-Ullate Oteiza: Big Data: conceptos, tecnologías y aplicaciones.

15 Cursos

[10] Juan Gabriel Comillas: Curso de Machine Leaning en Python.
[11] Universidad de Michigan; Data Science in Python.
[12] Red de universidades Anáhuac: Modelos predictivos con Machine Learning
[13] Apuntes de la asignatura de Avióniva. Grado de ingeniería aeroespacial, Unviersidad de Sevilla.

16 Artículos

[14] Geoffrey Holmes, Pia Sartor, Stephen Reed, Paul Southern, Keith Worden y Elizabeth Cross: Prediction of landing gear loads using machine learning techniques (2016).
[15] Desde la cabina de vuelo:Los sensores del ángulo de ataque en un avión (2015).
[16] UTC Aerospace Systems:Angle of Attack (AOA) Systems.
[17] Turning point news: Why Boeing only used one Angle of Attack AOA Sensor.
[18] Flying: How It Works: Angle of Attack Indicator.
[19] Eduardo Morales: diapositivas sobre procesos gaussianos, Instituto Nacional de Astrofísica, Óptica y Electrónica de México.
[20] Redes neuronales artificiales, ESPOL.
[21] ITAérea: Industria aeronáutica en España
[22] Partha Adhikari, Harsha Guruaja Ra y Matthias Buderath: Machine Learning based Data Driven Diagnostics Prognostics Framework for Aircraft Predictive Maintenance.
[23] Zheng Liu, Norbert Meyendorf and Nezih Mrad: The Role of Data Fusion in Predictive MaintenanceUsing Digital Twin.
[24] Geoffrey Holmes, Pia Sartor, Stephen Reed, Paul Southern, Keith Worden and Elizabeth Cross: Prediction of landing gear loads using machine learning techniques.

[25] Sergio Manuel Ignacio Cofré Martel: Modelo de detección de fallas y faltas para sistema neumético de turbinas de aviones Boeing 767 a través de machine learning.