FACULTAD DE MATEMÁTICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

ORGANIZACIÓN, ANÁLISIS DE DATOS E INTELIGENCIA ARTIFICIAL PARA LA PREDICCIÓN DE RESULTADOS EN FÓRMULA 1.

María Jesús García Villalón

Dirigido por:

Luis Valencia Cabrera

A Luis, por su paciencia, dedicación y profesionalidad.
A mis padres porque aunque no puedan entender todo lo que hay aquí escrito, se han esforzado para que yo pueda hacerlo y disfrutarlo.
A David por ser el mejor compañero.
A mis hermanos.

Abstract

If humans can learn from experience, can computers do the same? Through machine learning we have found out that the answer to this question is yes. In this paper, as will be seen throughout this report, we will use machine learning techniques to try to answer this question. automatic learning techniques to try to answer the question: Who will win the next Formula 1 Grand Prix? In order to do so, we will carry out the steps of exploratory data analysis, feature engineering and modeling, to create models that can answer the question by obtaining predictions that best fit the reality, analyzing the complications that we can obtain and trying to explore different solutions.

Resumen

Si los humanos pueden aprender de la experiencia, ¿pueden los ordenadores hacer igual? A través del aprendizaje automático hemos descubierto que la respuesta a ésta pregunta es sí. En este trabajo, como se verá a lo largo de ésta memoria vamos a utilizar las técnicas de aprendizaje automático para intentar responder la cuestión: ¿Quién ganará el siguiente gran premio de Fórmula 1? Para ello, llevaremos a cabo los pasos de análisis exploratorio de datos, ingeniería de características y modelización, para crear modelos que puedan responder dicha cuestión obteniendo predicciones que mejor se ajusten a la realidad, analizando las complicaciones que podamos obtener e intentando explorar diferentes soluciones.

Índice general

Parte I

Introducción.

Capítulo 1

Introducción.

La inteligencia artificial es una de las disciplinas que más atención viene atrayendo en los últimos años en la ciencia y la ingeniería. Actualmente, abarca una enorme variedad de subcampos que van de lo general a lo específico.

Una de las ramas de la inteligencia artificial que ha alcanzado mayor popularidad en estos años es el **aprendizaje automático** o **machine learning.** Su auge se debe a su gran aplicabilidad, puesto que prácticamente todo conocimiento es susceptible de ser aprendido e interpretado. El aprendizaje automático es la respuesta afirmativa a la pregunta de: *Si los humanos pueden aprender de la experiencia, ¿ pueden los ordenadores hacer igual?*

> Tom M. Mitchell: Se dice que un programa informático aprende de la experiencia E para alguna clase de tareas T y una medida de rendimiento P si su rendimiento en tareas T, medido por P, mejora con la experiencia E. [1]

La esencia del aprendizaje automático es tratar la evolución de los sistemas informáticos diseñados con el objetivo de aprender y adaptarse a partir de los datos, sin que sea necesario incluir claramente el conocimiento nuevo alcanzado [9]. Entonces el problema de aprendizaje automático puede definirse como la forma de ajustar un modelo entre los datos y de entrenarlo y validarlo para aprender las características partir de los datos. Luego, para proporcionarle el conocimiento a la máquina, tenemos que comprender y analizar en profundidad nuestro conjunto de datos.

Dentro del aprendizaje automático existe en líneas generales una división en: supervisado, cuando para las observaciones de entrada disponibles del problema conocemos a priori su salida contemplando aquí la regresión y la clasificación, no supervisado cuando no conocemos las salidas y no disponemos inicialmente de un buen conjunto de entrenamiento o este puede ser escaso, pero disponemos del motor que para cada posible entrada sea capaz de simular el comportamiento del sistema y darnos su salida.

Los conjuntos de datos poseen observaciones las cuales tienen descripciones denominadas características donde pueden tomar diferentes valores. A partir de estos datos vamos a llevar a cabo una serie de etapas:

1º Entender el problema. Debemos conocer en profundidad los tipos de datos que vamos a utilizar para resolver nuestro problema. Para ello, utilizamos análisis exploratorio de datos.

2º Criterio de evaluación. Definimos una medida de error que utilizaremos para evaluar el modelo, dependiendo del tipo de problemas que intentamos resolver.

3º Preparar los datos. Debemos organizar los datos de manera correctas para poder ser aplicados en nuestro modelo, para ello utilizamos ingeniería de características.

4º Construir el modelo. El proceso de usar algoritmo de machine learning para construir modelos a partir de un conjunto de datos se llama entrenamiento. Los algoritmos de aprendizaje suelen tener parámetros, y diferentes configuración de los parámetros y los datos de entrenamiento conducen a diferentes resultados resultados de aprendizaje, por ello debemos estimar los parámetros en los modelos que lo requieran. Una vez, realizada ésta configuración debemos determinar la precisión del modelo a partir del conjunto test.

5º Análisis de errores. Para ajustar los parámetros debemos ir evaluando el error cometido sobre el conjunto de entrenamiento.

6º Modelo integrado en un sistema. Una vez contentos con el error obtenido, incorporamos nuestro modelo resultante.

El objetivo principal del machine learning es crear modelos que funcionen igual de bien con datos nuevos que con los datos de entrenamiento, evitando el sobreajuste que se produciría si el modelo se amolda tanto al conjunto de entrenamiento que se aleja del comportamiento general de la función implícita que se intenta capturar y se comporta mucho peor sobre el conjunto de prueba.

Dado que el presente trabajo se centra en la Fórmula 1 el objetivo del mismo es el desarrollo y aplicación del aprendizaje automático a este deporte. Para ello, al margen de la parte I que proporciona esta introducción, dividimos el presente trabajo en cuatro partes. La parte II contiene el desarrollo teórico del preprocesado de datos y los diferentes modelos como regresión lineal, árboles de decisión random forest o redes neuronales. La parte III consiste en la aplicación práctica al conjunto de datos de la fórmula 1 de todo lo explicado en la parte de preliminares. En la parte IV para finalizar el trabajo expondremos las conclusiones a las que hemos llegado a través de los modelos aplicados y trabajos futuros que podremos hacer a partir de todo lo desarrollado en este trabajo. En la parte V tendremos los anexos que contendrán el código que se ha ido creando para realizar la parte práctica de este trabajo.

Parte II

Preliminares.

Capítulo 2

Exploración y limpieza de datos.

Vamos a abordar el primer capítulo de los tres que constituyen la parte II de la memoria los cuales sentarán las bases, a modo de presentación de los fundamentos preliminares, de algunas de las principales técnicas empleadas a lo largo del cuerpo del trabajo.

En este capítulo presentamos una parte fundamental de la ciencia de datos que es el preprocesamiento de datos, lo cual es necesario para conocer nuestro conjunto de datos y prepararlos para ser utilizados en los modelos desarrollados.

2.1. Análisis exploratorio de los datos.

El preprocesado de datos consiste en la adición, eliminación o transformación de datos del conjunto de entrenamiento. Esencialmente, se trata de explorar y visualizar datos para descubrir conocimientos desde el mismo inicio o identificar áreas o patrones para profundizar más en ellos. Los posibles pasos que se dan en el análisis exploratorio de los datos son:

- Selección de las variables relevantes y eliminación las variables irrelevantes.
- Identificación de valores atípicos (outliers), valores desaparecidos o errores.
- Comprensión de las relaciones entre las variables.
- Maximización del conocimiento sobre un conjunto de datos y minimización de los posibles errores posteriores.

Haciendo uso de la famosa expresión *"garbage in, garbage out"* necesitamos tener un buen conjunto de datos en el cual conozcamos las características de cada variable para así llegar a tener buenos datos y con ello poder crear buenos modelos y de ellos obtener conclusiones correctas.

2.2. Ingeniería de características.

Definición 2.2.1 La ingeniería de características es el proceso de transformación de los datos crudos en características que representen de la mejor forma posible el problema subyacente a los modelos, resultando en una mejor precisión de los modelos.

La ingeniería de características se basa en la utilización del conocimiento sobre el contexto o dominio de los datos objeto de estudio para preparar y extraer características de un conjunto de datos mediante técnicas de minería de datos. Por lo que en determinadas situaciones necesitaremos transformar las variables, dentro del proceso de ingeniería de características podemos encontrar pasos o transformaciones que serán adecuadas o no según cada modelo concreto, como se aplicará después. La selección de características es un paso importante del preprocesamiento de datos. Si podemos identificar las características relevantes, entonces el proceso de aprendizaje posterior tratará con una dimensionalidad menor.

Codificación de variables categóricas.

Cuando un predictor es categórico, es habitual descomponer el predictor en un conjunto de variables más específicas. Para utilizar estos datos en los modelos, las categorías se recodifican en trozos de información más pequeños llamados **"dummy variables"**.

Normalmente, cada categoría tiene su propia 'dummy variable"que es un indicador de cero/uno para ese grupo. Una "dummy variable"podría decirse que se asemeja a la técnica one-hot-encoding, sin embargo, solo necesitamos $n-1$ variables ya que la n-ésima puede ser inferida a partir de las restantes.

Escalado de características y normalización.

La transformación de datos más común es centrar y escalar las variables predictoras.

- Centrar una variable consiste en restar su media a cada una de sus observaciones. El resultado es que la variable predictora tiene media cero.
- Escalar una variable consiste en dividir cada observación entre su desviación típica. El resultado es que la variable predictora tiene varianza uno.

Estas transformaciones proporcionan que todas las variables estén en la misma escala. La ventaja de esta técnica es que todas las variables tengan el mismo peso de cara a la obtención de la salida de los modelos, evitando así el sesgo hacia variables que en un principio presenten mayores órdenes de magnitud. El incoveniente que ésto nos provoca es la pérdida de interpretabilidad de los valores individuales ya que no estarían en las unidades originales.

Limpieza de los datos.

Cuando preparamos un conjunto de datos para su posterior análisis, es muy común encontrarnos con valores desaparecidos. Estos valores afectan al rendimiento de nuestros modelos, ya que no son fáciles de tratar. Podemos tratarlos de dos formas diferentes: eliminación e imputación.

Eliminación

Al tener un conjunto de datos que contiene observaciones las cuales poseen valores perdidos, una primera idea que podemos llevar a cabo es eliminar del conjunto las observaciones que contienen estos valores faltantes. Una consecuencia grave de eliminarlos es

que estamos perdiendo la información de la observación que lo contiene. Para conjuntos de datos grandes, eliminar observaciones con valores faltantes no es un problema, asumiendo que los valores perdidos no proporcionan gran información. En conjuntos de datos pequeños, al eliminar los valores perdidos, estaríamos perdiendo demasiada información.

Para no perder información, podemos llevar a cabo la siguiente técnica.

Imputación.

La imputación es un método de tratamiento de datos perdidos donde intentamos estimar los valores de las variables predictoras en función de otras variables predictoras, de forma que los valores faltantes se completan mediante esta estimación calculada. Con la imputación no estamos perdiendo toda la información de dicha observación; sin embargo, tenemos que determinar cuáles son los valores que imputamos.

Una opción es basarse en datos existentes y usar un algoritmo de aprendizaje automático. Así, una de las técnicas más populares de imputación es el modelo de los K vecinos más cercanos.

- Imputación KNN.

  Con la técnica de imputación KNN una muestra se imputa encontrando las muestras del conjunto de entrenamiento más cercanas a ella y promedia estos puntos cercanos para rellenar el valor. Una ventaja de este método es que es una aproximación de grano más fino, atribuimos comportamientos más cercanos a los ejemplos más similares, en lugar de un valor como el promedio, la mediana o la moda que son de grano más grueso y no tienen presente las diferencias entre las observaciones con datos faltantes, atribuyendo el mismo valor a todas. Una desventaja es que cada vez que necesitamos imputar un valor, tendremos que recurrir al conjunto de entrenamiento completo.

Reducción de la dimensionalidad.

Este es un proceso de reducción del número de variables de los datos originales para generar un conjunto de datos con una capacidad predictiva similar, pero con un menor número de variables o características, lo cual significa una disminución del tiempo y complejidad de los cálculos, consecuentemente facilita su utilización en modelos al igual que disminuye las posibilidades de sobreajuste.

El análisis de componentes principales (PCA) permite encontrar un número de variables x inferior al número actual p, $x < p$, que expresen aproximadamente lo mismo que las p variables originales. De esta forma, si en un principio requeríamos p variables para interpretar a cada individuo, ahora bastan x variables. Cada una de estas x nuevas variables recibe el nombre de componente principal. La principal ventaja del PCA, y la razón por la que ha conservado su popularidad como método de reducción de la dimensionalidad de los datos, es que crea componentes que están no están correlacionadas, y prescinde de las variables originales que sí puedan estarlo. Así, nos quedaremos con el número componentes principales que queramos. En esta situación aparece el término colinealidad que se usa cuando un par de variables predictoras tienen una correlación significativa entre sí. Por este motivo, podría decirse que la técnica de componentes principales obtiene de la colinealidad de las variables las componentes principales las cuales no están correlacionadas.

Puesto que PCA busca combinaciones lineales de variables predictoras que maximicen variabilidad así esta técnica puede ayudar también a detectar y eliminar variables que sean próximas a una combinación lineal de otra u otras.

Por otro lado, cuando el conjunto de datos consta de demasiadas variables predictoras la técnica PCA puede utilizarse para caracterizar la magnitud del problema.

Capítulo 3

Modelos.

Como ya mencionamos en la introducción, en este capítulo vamos a describir los distintos modelos diferenciando el tipo de aprendizaje automático. Entre otros aspectos, el machine learning utiliza la teoría estadística para construir modelos matemáticos porque la tarea principal es hacer inferencia a partir de una muestra.

3.1. Aprendizaje supervisado.

Los algoritmos de aprendizaje supervisado hacen uso de los datos, con sus entradas y salidas conocidas y utilizan esa información para entrenar y validar el modelo.

Definición 3.1.1 (Aprendizaje supervisado) El lenguaje supervisado significa construir un modelo parametrizado que pueda dividir el dominio de los datos y, a continuación, optimizar los parámetros mediante algoritmos de entrenamiento, validación y prueba.

Asumimos un modelo con un conjunto de parámetros $Y = g(x|\theta)$ donde:

- $g(.)$ es el modelo función de regresión o discriminante dependiento de la naturaleza de la variable respuesta.
- x variables predictoras.
- θ parámetros.
- Y variable respuesta

El aprendizaje supervisado tiene dos principales objetivos: objetivos de parametrización y objetivos de optimización. Estos objetivos pueden definirse y diferenciarse utilizando la naturaleza continua y discreta de las variables respuesta.

El objetivo de la parametrización se define como la acción relacionada con la **regresión** si el conjunto de respuestas es continuo y se define como acción relacionada con la **clasificación** si el conjunto de respuestas es discreto.

El objetivo de la optimización es seleccionar varios modelos y después seleccionar el mejor modelo utilizando una medida de distancia y los conjuntos de datos etiquetados.

Para llevar a cabo estos objetivos requerimos de particionar nuestro conjunto de datos original en conjuntos de entrenamiento y test.

- **Entrenamiento** Los parámetros del modelo de machine learning son estimados, aproximados y optimizados utilizando el conjunto de datos etiquetados.

- **Test** Sirve para asegurarnos de que el modelo funciona utilizando otro conjunto de datos, es decir, ayudan a determinar la precisión del modelo, estimando así cómo de bien se comportará nuestro modelo a la hora de predecir la salida para datos desconocidos para el modelo.

3.1.1. Modelos de regresión.

La variable respuesta continua Y de un sistema puede modelizarse mediante una relación lineal X

$$Y = \beta X + \epsilon \tag{3.1.1}$$

Donde ϵ es una variable error. Una minimización de este error y una regularización distinta contribuyen a modelos de regresión.
La continuidad de las variables permite la aplicación de derivadas para la minimización. Cuando se dispone de un dominio de datos y sus correspondientes valores de respuesta, deberíamos ser capaces de estimar el parámetro a partir de esta minimización del error.

El objetivo de parametrización es definir un modelo parametrizado utilizando el modelo simple : $y = \beta x$.

El objetivo de optimización se consigue mediante la optimización del siguiente factor de error con respecto al parámetro β.
Para minimizar la función del error con respecto al parámetro β definimos la suma de los cuadrados de los residuos:

$$RSS(\beta) = (y - X\beta)^t(y - X\beta) \tag{3.1.2}$$

Vamos a minimizar (3.1.2). La condición necesaria y suficiente de optimalidad es $\nabla RSS(\beta) = 0$:

$$\nabla RSS(\beta) = 0 \Leftrightarrow -2X^t(y - X\beta) = 0 \Leftrightarrow X^t y = X^t X\beta$$

$$\hat{\beta} = (X^t X)^{-1} X^t y$$

Por tanto, $\hat{Y} = X\hat{\beta}$.

Observación 3.1.2

1. La condición necesaria y suficiente de optimalidad se da si $det(X^t X) \neq 0$

2. Si $det(X^t X) = 0$ tendríamos un sistema compatible indeterminado con lo cual obtendríamos infinitos β , luego, no seríamos capaces de hacer predicciones.

3.1.2. Modelos de clasificación.

Los modelos de clasificación son adecuados para el sistema que produce respuestas discretas.

Si la variable respuesta posee solo dos posibles clases, se denomina problema de clasificación binaria, en la cual tenemos una clase como positiva y otra como negativa. Cuando posee más de dos clases, se convierte en un problema de clasificación multiclase.

Como en el caso anterior, al hallarnos en un tipo de aprendizaje supervisado, asumimos que los datos están etiquetados y pueden generar reglas que nos ayuden a etiquetar a un nuevo dato el cual no tiene asignada una etiqueta.

Para representar la relación entre los datos del dominio X y el conjunto de respuesta Y usaremos el mismo modelo del caso continuo (3.1.1)

Sin embargo, en esta ecuación X es continuo e Y es discreto, lo cual dificulta ajustar un modelo lineal o no lineal. Debemos definir un conjunto de respuesta intermedio que es continuo para ajustar el modelo.

La nueva variable representa una probabilidad $p(x) = P[Y = 1|X = x]$

$$p(x) = \frac{1}{1 + e^{\beta X}} \tag{3.1.3}$$

Por esta razón la regresión logística define la ecuación intermedia, la cual sigue un modelo lineal:

$$log(\frac{p}{1-p}) = \beta X \tag{3.1.4}$$

El conjunto de datos etiquetados es usado para comparar los resultados producidos por el modelo final en términos de precisión de la clasificación y tiempo de cálculo. Para ello, existen medidas llamadas medidas cualitativas que se utilizan para medir el rendimiento del modelo. Algunas de estas medidas son: exactitud, sensibilidad, especificidad y precisión.

3.1.3. Modelos Jerárquicos.

Adopta tanto estrategias de regresión como de clasificación con un árbol que puede construirse mediante una secuencia de decisiones. Estos conducen a estratificar o segmentar el espacio predictor en un número de regiones simples, de forma que para predecir una observación determinada se utiliza la región a la que pertenece.

Los modelos jerárquicos que vamos a considerar son

- Árboles de decisión.
- Bosques aleatorios.

Árboles de decisión

Es uno de los métodos más utilizados y práctico para la inferencia infuctiva. Los árboles de decisión clasifican las instancias ordenándolas en el árbol desde la raíz hasta algún nodo de la hoja. Cada nodo del árbol especifica una prueba de alguna variable de la observación, y cada rama que desciende de ese nodo corresponden a uno de los posibles valores de la variable. En general, los árboles de decisión representan una disyunción de conjunciones de restricciones sobre los valores de las variables de las observaciones [4].

Según aplicamos un árbol de decisión a problemas de regresión o clasificación diferenciamos los árboles de regresión los cuales ayudan a predecir un valor para la variable

respuesta, y los árboles de clasificación los cuales ayudan a predecir una etiqueta de clase para la variable respuesta.

- Árboles de regresión.

  El algoritmo debe decidir automáticamente las variables de división y los puntos de división. Una vez encontrada la mejor división, dividimos los datos en las dos regiones resultantes de manera que se minimice el error global de la suma de los cuadrados, y repetimos el proceso de división en cada una de las dos regiones, y luego repetimos el proceso en todas las regiones resultantes.

  Claramente, un árbol muy grande prodría sobreajustar los datos, mientras que un árbol pequeño podría no capturar la estructura importante. El tamaño del árbol es un parámetro de ajuste que rige la complejidad del modelo, y el tamaño óptimo del árbol debe elegirse de forma adaptativa a partir de los datos. La estrategia más usada es hacer crecer un árbol grande T_0, deteniendo el proceso de división solo cuando se alcanza algún tamaño mínimo de nodo. Entonces el árbol grande se poda utilizando la poda de complejidad de costes.

  Definimos un subárbol $T \subset T_0$ como cualquier árbol que se puede obtener mediante una poda que colapse cualquier número de nodos internos. Indexamos los nodos terminales por m, con el nodo m representando la región R_m . $|T|$ denota el número de nodos terminales de T. Teniendo:

$$N_m = \#\{x_i \in R_m\} \quad \hat{c_m} = \frac{1}{N_m} \sum_{x_i \in R_m} yi$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (yi - \hat{c_m})^2$$

  Definimos el criterio de coste-complejidad:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| \tag{3.1.5}$$

  La idea es encontrar para cada α el subárbol $T \subset T_0$ que minimiza $C_\alpha(T)$ en (3.1.5). El parámetro $\alpha \geq 0$ de funcionamiento gobierna el compromiso entre el tamaño del árbol y su bondad de ajuste a los datos. Los valores grandes de α dan lugar a árboles más pequeños, y a la inversa para valores más pequeños de α.

  La estimación de α se consigue mediante validación cruzada de 5 o 10 pliegues. Elegimos el valor $\hat{\alpha}$ para minimizar la suma de cuadrados de la validación cruzada. El árbol final será $T_{\hat{\alpha}}$.

- Árboles de clasificación.

  El objetivo de los árboles de clasificación es particionar el conjunto de datos en pequeños y homogéneos grupos. La homogeneidad significa que cada nodo es más puro, es decir, que contiene una proporción mayor de una clase en cada nodo. Los únicos cambios necesarios en el algoritmo del árbol se refieren a los criterios para dividir los nodos y podar el árbol.

Para la regresión utilizamos la medida de impureza del nodo de error cuadrático $Q_m(T)$, pero esto no es adecuado para la clasificación. En un nodo m, que representa una región con m observaciones sea

$$\hat{p_{mk}} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = m)$$

La proporción de observaciones de la clase k en el nodo m. Clasificamos las observaciones del nodo m en la clase $k(m) = argmax_k \hat{p_{mk}}$

Bosques aleatorios

Forma estructura de tipo bosque con árboles de decisión que se generan mediante el muestreo aleatorio con reemplazamiento.

Se genera un conjunto de árboles los cuales toman diferentes porciones del conjunto de datos, en lugar de uno solo ya que puede ser muy variable y sensible a la muestra tomada, de modo que la predicción arrojada por el bosque es proporcionada por el concenso entre los árboles.

Es decir, al ver los datos cada árbol obtenemos un resultado de cada uno luego al combinar sus resultados estamos obteniendo errores que se nivelan de esta forma la predicción generaliza mejor.

Las predicciones se combinan de diferente forma según nos encontramos en un problema de predicción o de clasificación.

- Problema de regresión. Cuando por cada árbol obtenemos un resultado, la forma de combinarlos es hacer su media aritmética.

- Problema de clasificación. Cuando por cada árbol obtenemos un resultado, para este problema obtenemos las diferentes clases de la variable respuesta. La forma de combinar las predicciones de todos los árboles es dandole mayor importancia a la clase que más veces se ha predicho.

La ventaja es que proporcionan múltiples clasificadores de árboles de decisión entrenados en la fase de prueba.

Los parámetros a optimizar de los bosques aleatorios son el número de predictores seleccionados aleatoriamente, k, para elegir en cada división, y se conoce comúnmente como m_{try}. El número de árboles del bosque y el número mínimo de observaciones en cada nodo.

En la figura 3.1, podemos ver un esquema de la estructura de un bosque aleatorio, como cada árbol que está contenido en él genera una predicción y cómo se combinan las predicciones.

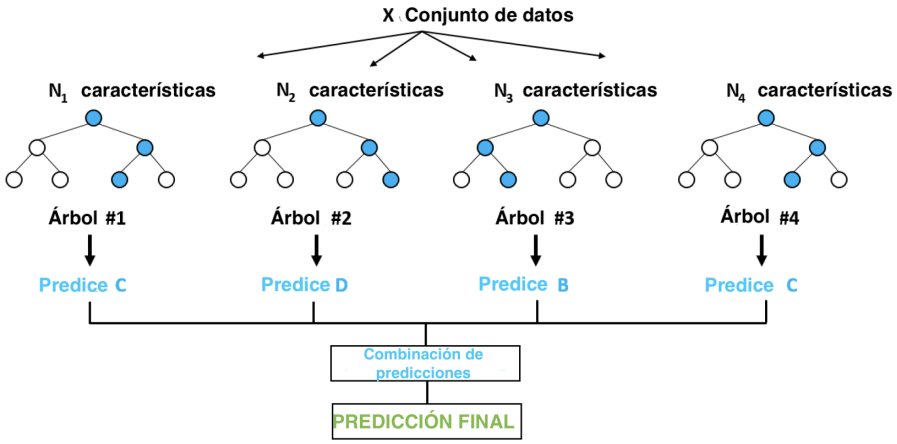


Figura 3.1: Estructura Random forest.

3.2. Aprendizaje no supervisado.

El aprendizaje no supervisado se basa en las correlaciones entre los datos de entrada, y se utiliza para encontrar los patrones o características significativas en los datos de entrada sin la ayuda de las salidas para los datos observados, ya que en diferencia del aprendizaje supervisado no conocemos estas salidas.

Definición 3.2.1 (Análisis cluster) Es una técnica estadística multivariante que busca agrupar elementos (o variables) tratando de lograr la máxima homogeneidad en cada grupo y la mayor diferencia entre los grupos, de forma que los que se encuentran dentro de cada cluster están más relacionados entre sí que los elementos asignados a clusters diferentes. Básicamente, el clustering consiste en agrupar las observaciones según su proximidad, esto es, dos observaciones formaran parte del mismo cluster si están próximas y formaran parte de clusters distintos si están lejos una de la otra.

Podemos diferenciar dos técnicas de clustering:

- Clustering de partición estricta. Dado un conjunto de observaciones, buscamos una partición de este conjunto en k clusters siendo el número de clusters k menor que el de observaciones, de manera que se verifique:

  1. Ningún cluster es vacío.
  2. La unión de todos los clusters es el conjunto total de observaciones.
  3. Los clusters son disjuntos dos a dos.

  Dentro de esta técnica podemos destacar el algoritmo de K-medias. Es uno de los métodos de clustering por descenso iterativo más populares. Está pensado para situaciones en las que todas las variables son de tipo cuantitativo, aunque puede inculir variables cualitativas, empleando distancia Hamming.

- Clustering de partición jerárquica. Dado un conjunto de observaciones, buscamos construir una partición anidada con estructura de árbol $\mathcal{H} = \{P_1, \ldots, P_Q\}$ tal que

si $C_i \in P_m$ y $C_j \in P_l$ con $m > l$ entonces $C_i \subset C_j$ o $C_i \cap C_j = \emptyset$ $\forall i, j, m, l \in \{1, \ldots, Q\}, i \neq j$.

Podemos decir así que cada cluster está dividido en clusters lo que proporciona la estructura anidada.

3.3. Redes Neuronales Artificiales.

Las redes neuronales son potentes herramientas de técnicas no lineales de regresión, las cuales están basadas en la teoría del funcionamiento del cerebro.

Definición 3.3.1 Una red neural artificial es un modelo de procesamiento distribuido y en paralelo, en forma de un grafo dirigido, con la siguiente estructura: elementos de procesamiento, conexiones sinápticas, conexiones entrantes, conexiones salientes, memoria local, función de activación y continuada o episódica.

Pueden usarse tanto para aprendizaje supervisado como para no supervisado : predicción, clasificación y agrupamiento. Sin embargo, la gran mayoría de variantes de redes neuronales artificiales se enmarcan dentro del aprendizaje supervisado ya que no deja de ser un modelo que trata de ajustar la función subyacente a partir de un conjunto de ejemplos donde conocemos a priori sus entradas y salidas, y entrenamos los pesos de una red que capture las interrelaciones entre los datos de entrada para producir las citadas salidas conocidas.

El aprendizaje de una red neuronal puede verse como un problema de optimización no lineal para encontrar un conjunto de parámetros de la red que minimicen la función de coste para unos ejemplos dados.

La variable objetivo está modelada por un conjunto intermedio de variables no observadas, las cuales se denominan variables ocultas. Estas variables ocultas son combinaciones lineales de algunas de las variables predictivas. Sin embargo, estas combinaciones lineales son transformadas por una función no lineal.

$$h_k(\mathbf{x}) = g(\beta_{0k} + \sum_{i=1}^{P} x_j \beta_{jk})$$

Siendo P el número de variables predictoras. Los coeficientes β son similares a los coeficientes de regresión, y, los coeficientes β_{jk} son el efecto de la j-ésima variable predictora en la k-ésima variable oculta.

Una red neuronal puede contener múltiples capas ocultas para modelizar la variable respuesta.

3.4. Validación del ajuste.

Puede que nuestro ajuste haya sido muy bueno con la muestra que hemos estado usando, pero no sabemos si el modelo puede ser útil para predecir nuevos valores de la variable respuesta.

En consecuencia podemos usar varias técnicas de validación, no obstante, en este trabajo destacamos la **validación cruzada con k pliegues.** La validación cruzada es

un método común de selección de modelos. El conjunto de datos se divide aleatoriamente en un conjunto de entrenamiento y un conjunto de test.

En general, los pasos que sigue este tipo de validación son:

1. Tomar k.
2. Dividir aleatoriamente la muestra en k grupos aproximadamente del mismo tamaño.
3. Para cada grupo, tomamos éste como muestra test, y los restantes como muestra aprendizaje.
4. Tomar como error el promedio de los k errores así obtenidos.

3.5. Métricas.

Cuando creamos un modelo predictivo, nos interesa saber cúal es la capacidad predictiva del modelo, lo que se puede estimar a través de las métricas que nos permiten conocer cómo de bueno es un algoritmo realizando predicciones. Resulta evidente, distinguir las métricas según si tenemos un problema de regresión o de clasificación.

Problema de regresión.

Cuando la variable respuesta es cuantitativa estamos intentando predecir el valor numérico de una cantidad desconocida, por ello, la diferencia entre la predicción y el valor real es el denominado error, el cual es una variable aleatoria. Para estimar el rendimiento del modelo de regresión la métrica más común es la raíz cuadrada de la media de los errores elevados al cuadrado (RMSE), sin embargo, tenemos otras métricas como el MSE o el MAE. Definimos a continuación estas medidas.

Definición 3.5.1 (MSE) Es la media de de los errores elevados al cuadrado.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Cabe resaltar que las diferencias se están elevando al cuadrado para evitar que diferencias positivas y negativas se neutralicen y nos confundan, pero que al estar al cuadrado se va a aumentar el orden de magnitud de la medida, pareciendo que nuestro error es mayor de lo que es. Para paliar esto, pasamos a definir la métrica RMSE.

Definición 3.5.2 (RMSE) Es una función de los residuos del modelo, que son los valores observados y_i menos las predicciones del modelo $\hat{y}_i$.

$$RMSE = \sqrt[2]{(\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2)}$$

Por lo tanto, por definición el RMSE es la raíz cuadrada del MSE que con la raíz devuelve la medida al orden de magnitud de nuestra salida. Por último tenemos la métrica MAE, la cual sigue una estrategia diferente pero persigue finalmente lo mismo que RMSE, llevar la diferencia al orden de magnitud de nuestra salida.

Definición 3.5.3 (MAE) Es la media de los errores en valor absoluto.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid$$

La diferencia entre el MSE y el MAE es que MSE al elevar al cuadrado los errores está penalizando más las desviaciones grandes.

Problemas de clasificación.

Estas medidas cualitativas están construidas por las medidas cuantitativas básicas, serían para clasifiación binaria, cuando tenemos multiclase aumenta la complejidad en la descripción de las mismas.

Cada predicción puede ser uno de cuatro resultados, basado en cómo coincide con el valor real, al valor positivo lo denotaremos con un 1 , en caso contrario al negativo lo denotaremos con un 0.

Entonces tenemos los resultados recogidos en la figura **??**: tasa de verdaderos positivos (TP) probabilidad de clasificar positivo siendo positivo y la tasa de falsos positivos (FN) clasificar como positivo siendo negativo. tasa de verdaderos negativos (TN) que es la probabilidad de clasificar como negativo siendo negativo, la tasa de falsos negativos (FN) es clasificar como negativo siendo positivo.

Definición 3.5.4 (Exactitud)

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{3.5.1}$$

Describe el rendimiento del modelo basándose en la proporcionalidad entre los falsos positivos y los verdaderos positivos. Es decir, mide el porcentaje de casos que el modelo ha acertado.

Definición 3.5.5 (Precisión)

$$\frac{TP}{TP + FP} \tag{3.5.2}$$

Describe el rendimiento del modelo en función de la proporcionalidad entre los falsos positivos y los verdaderos positivos. Midiendo así la calidad del modelo.

Definición 3.5.6 (Sensibilidad)

$$\frac{TP}{TP + FN} \tag{3.5.3}$$

Describe el rendimiento del modelo en función de la proporcionalidad entre los falsos negativos y los verdaderos positivos. En otras palabras, estamos midiendo la proporción de casos positivos que han sido correctamente clasificados.

Definición 3.5.7 (Especificidad)

$$\frac{TN}{TN + FP} \tag{3.5.4}$$

Describe el rendimiento del modelo en función de la proporcionalidad entre los falsos positivos y los verdaderos negativos. En definitiva, estamos midiendo la proporción de casos negativos que han sido correctamente clasificados.

Podemos comparar los valores predichos con respecto a un umbral de clasificación, es decir, clasificar una muestra como positiva si el valor de predicción es mayor que el umbral y como negativa en caso contrario. La curva de características operativas del receptor (ROC) sigue la idea anterior. Una forma de comparar las curvas ROC intersecadas es calcular las áreas bajo las curvas ROC, es decir, el Área bajo la curva ROC (AUC).

Capítulo 4

Herramientas.

Para desarrollar la parte práctica de este trabajo, se va a utilizar el lenguaje de programación R a través de la intefaz (entorno de desarrollo integrado) Rstudio.

R es un conjunto integrado de programas para manipulación de datos, cálculo y gráficos. Podría decirse que R es de las principales herramientas estadísticas ya que éste contiene numerosas técnicas tanto clásicas como modernas. La mayoría de los códigos de R actualmente, están incluidos en diferentes *"paquetes"*, los cuales pueden ser diseñados y publicados por cualquier usuario.

Los *paquetes* que vamos a utilizar son:

- **skimr** Proporciona un resumen estadístico completo de las variables de un conjunto de datos tipo data frames, tibbles o similar. Resulta muy útil para el análisis exploratorio de los datos (2).

- **tidyverse** Contiene una colección de paquetes diseñados para la ciencia del dato. Es decir, para la importación, ordenación, manipulación y visualización de los datos. Los paquetes que más vamos a utilizar dentro de Tidyverse son:

  - ggplot2, para la visualización.
  - dplyr, para la manipulación.
  - tidyr, para la ordenación.
  - readr, para la importación.
  - purrr, para la programación funcional.

- **tidymodels** Contiene las acciones necesarias para crear un modelo estadístico, es decir, las etapas de preprocesado, entrenamiento, optimización y validación de modelos.

  Los paquetes que más vamos a utilizar dentro de Tidymodels son:

  - rsample, para la división y el remuestreo de los datos.
  - parsnip, para la modelización.
  - recipes, para usar las herramientas de preprocesamiento con ingeniería de características.

- workflows, para agrupar el preprocesamiento, modelado y postprocesamiento.
- tune, para optimizar los parámetros.
- yardstick, para las métricas de los modelos.

Los pasos que vamos a seguir en este trabajo para usar esta librería son:

1º Creamos una división entre entrenamiento y test: `initial_split`.

2º Definimos el modelo que queramos implementar (3), indicando cuando sea necesario los parámetros que queremos ajustar a través de la orden `tune`, la implementación que vamos a utilizar `set_engine` y el modo en el que se va a utlilizar el modelo (regresión o clasificación) con `set_mode`.

3º Definimos el grid con `grid_regular` o `expand_grid` para indicar los valores que pueden tomar los parámetros que queremos ajustar.

4º Validación cruzada siempre que queramos ajustar parámetros, utilizando la orden `vfold_cv` para generar el número de particiones que indiquemos.

5º Receta de transformaciones necesarias sobre el conjunto de datos. Algunas transformaciones que vamos a realizar durante el trabajo son: `step_impute_knn`, para hacer imputación KNN, `step_novel` especificación sobre las variables tipo factor para solventar el problema de intentar predecir la salida para un nuevo ejemplo que contenga valores de un factor que nunca aparecieron en el conjunto de entrenamiento, `step_dummy` para la creación de variables dummy explicadas en (2.2), `step_zv` para eliminar variables que contengan un único valor, `step_center`, `step_scale` pasos para centrar y escalar las variables, `step_corr` elimina variables que estén altamente correladas con otras, `step_lincomb` elimina variables numéricas que estén en combinación lineal con otras.

6º Creamos el workflow para utilizar la receta en el modelo.

7º Ajustamos el modelo a través del grid con `tune_grid` calculando un conjunto de métricas (mae, rmse, rsq).

8º Seleccionamos el mejor modelo, según la métrica deseada y finalizamos el workflow con ese modelo.

9º Estimamos el mejor modelo empleando `last_fit`.

En los modelos en los cuales no vamos a optimizar parámetros, modelo lineal y red neuronal, no haremos los pasos 3,4,7 y 8.

Parte III

Cuerpo.

Capítulo 5

Predicciones.

Es evidente que la fórmula 1 ha evolucionado mucho desde sus inicios hasta ahora, sin embargo, desde los inicios nos hemos hecho la misma pregunta: *¿Quién ganará cada Gran Premio?*

Para intentar responder a esta cuestión resulta interesante aplicar las técnicas de Machine Learning, siguiendo los pasos explicados en la parte de preliminares (II).

Haremos en primer lugar un estudio exhaustivo tanto del conjunto de datos como de las variables, para a continuación poder definir e implementar diferentes modelos de regresión con los que obtener predicciones.

5.1. Análisis exploratorio de los datos.

Como hemos desarrollado en el capítulo 2, el primer paso que tenemos que hacer es un estudio exhaustivo de las variables que intervendrán en nuestros modelos para así comprender qué información contiene cada una de ellas. En primer lugar, vamos a definir cada conjunto de datos que utilizaremos los cuales fueron descargados de [10].

- `circuits.csv`: conjunto de datos en el cual tenemos información sobre los 79 los circuitos en los que se ha corrido a lo largo de todas las temporadas, como el país al que pertenece, la localidad en la que está ubicado, su latitud y altitud.
- `constructor_results.csv`: conjunto que contiene los puntos que cada constructor ha obtenido en cada carrera.
- `constructors.csv`: información de cada escudería que ha participado en la competición de fórmula 1.
- `drivers.csv`: información de cada piloto que ha disputado alguna carrera en la fórmula 1.
- `lap_times.csv`: tiempo por vuelta que llevaba cada piloto en cada carrera, así como la posición en la que iba en cada vuelta.
- `pits_stops.csv`: información sobre las paradas en boxes de cada piloto en cada carrera, indicando el número de parada y el tiempo empleado en ésta, tanto en segundos como en milisegundos.

- `qualifying.csv`: información correspondiente a la clasificación de cada piloto con cada constructor para cada carrera, es decir, el mejor tiempo en cada Q y posición en la que clasificó en cada carrera.
- `races.csv`: información de cada carrera: circuito en el que se disputó, fecha en la que se disputó y ronda dentro de cada temporada en la que se disputó.
- `results.csv`: información sobre los resultados en cada carrera de cada piloto: constructor con el que estaba corriendo, posición desde la que salía, posición en la que acabó, el estado en el que acabó, puntos conseguidos, número de vueltas que dio, tiempo de la vuelta rápida y número de vuelta en la que hizo la vuelta rápida.
- `status.csv`: información sobre diferentes estados en los que se ha terminado una carrera.

Vamos a ver la relación que existe entre las diferentes tablas que vamos a usar en este trabajo, resaltando únicamente las columnas más importantes.

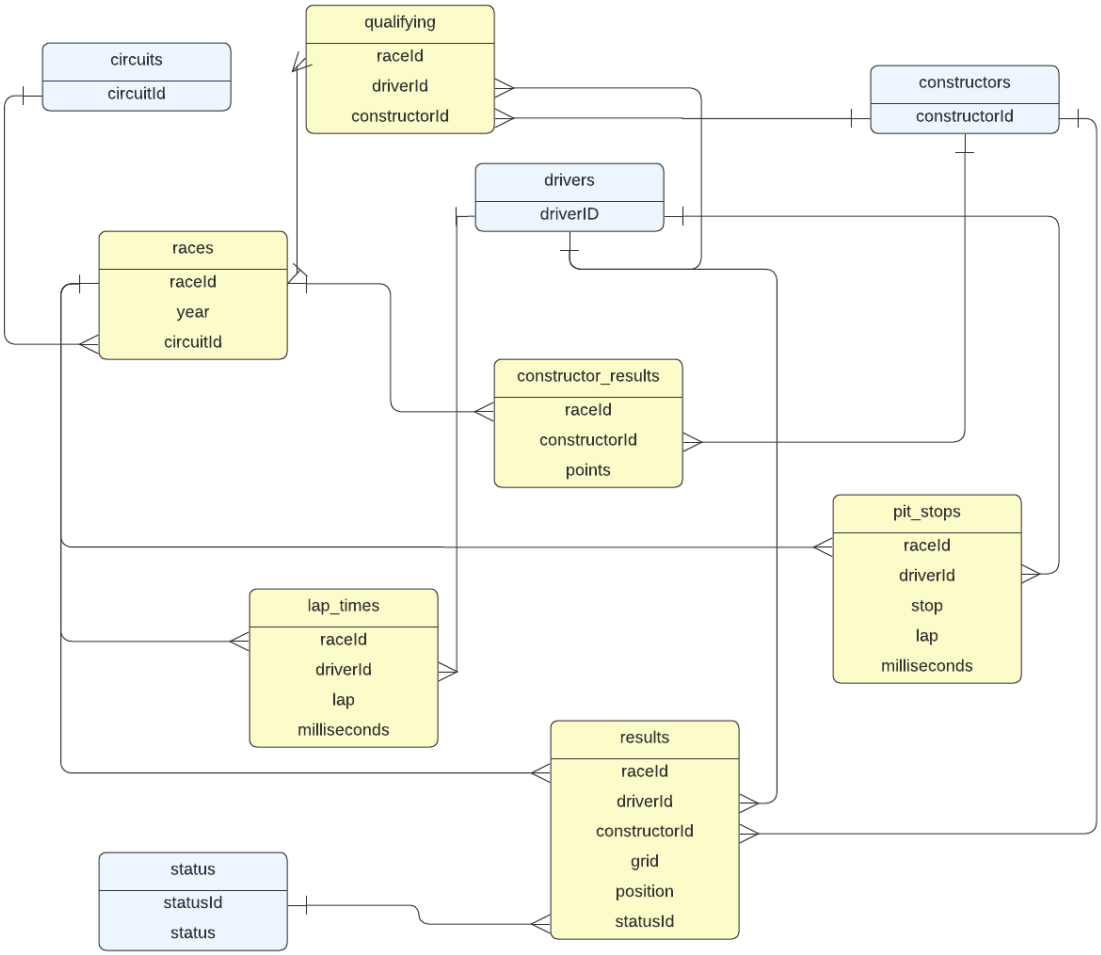


Figura 5.1: Relación entre los conjuntos de datos.

5.1.1. Preprocesamiento de datos

Vamos a estudiar cómo tratar cada tipo de datos que vamos a utilizar.

En primer lugar, observamos que en el conjunto de datos de las carreras, tenemos la información de las carreras que se disputan esta temporada (2022). Sin embargo, como ésta aún no ha terminado, es evidente que no tenemos los resultados. Por tanto, en nuestro conjunto de datos vamos a utilizar la información de las carreras hasta la temporada 2021.

```
     raceId           year         circuitId
 Min.   :   1.0   Min.   :1950   Min.   : 1.00
 1st Qu.: 270.8   1st Qu.:1976   1st Qu.: 9.00
 Median : 540.5   Median :1993   Median :18.00
 Mean   : 542.5   Mean   :1991   Mean   :22.45
 3rd Qu.: 810.2   3rd Qu.:2009   3rd Qu.:32.00
 Max.   :1096.0   Max.   :2022   Max.   :79.00
```

Figura 5.2: Summary races.csv

Si hacemos ahora un resumen estadístico de todas las carreras sin tener en cuenta la temporada 2022, podemos observar comparando con la figura 5.2 que para ésta temporada hay un circuito nuevo que será el que tiene el circuitId = 79.

```
     raceId           year         circuitId
 Min.   :   1.0   Min.   :1950   Min.   : 1.00
 1st Qu.: 265.0   1st Qu.:1976   1st Qu.: 9.00
 Median : 529.0   Median :1992   Median :18.00
 Mean   : 530.7   Mean   :1991   Mean   :22.26
 3rd Qu.: 793.0   3rd Qu.:2008   3rd Qu.:32.00
 Max.   :1073.0   Max.   :2021   Max.   :78.00
```

Figura 5.3: Summary carreras hasta la temp 21

Esto nos hace pensar que en nuestro conjunto de datos no habrá información para este nuevo circuito, por lo que tenemos que pensar alguna forma de crear modelos para poder predecir en circuitos nuevos.

También podemos destacar que, la fórmula 1 tuvo sus inicios en el año 1950 y hasta ahora se han disputado más de 1000 carreras a lo largo de todos los años.

Realizando ahora un resumen estadístico sobre los resultados en las carreras, observamos que en algunas carreras hay datos alarmantes para pensar que algunas no corresponden a carreras de fórmula 1. En primer lugar, nos fijamos en el grid, que es la posición desde la que salen los pilotos al inicio de la carrera,teniendo esta un mínimo de 0, lo cuál no tiene sentido ya que siempre hay alguien que sale desde primera posición (*pole*) como podemos observar en la posición final. Esto nos hace pensar que hay datos perdidos o faltantes, por lo que la información correspondiente a esos datos, la eliminaremos de nuestro conjunto de datos. A continuación nos fijamos en el número de vueltas, siendo el máximo de éstas 200, número que excede con creces la media de vueltas en una carrera de fórmula 1, lo que nos hace pensar que en el conjunto de datos hay información que no pertenece a carreras de fórmula 1. Si ahora nos fijamos en la posición final, correspondiente a la co-

```
     grid          position          laps
 Min.   : 0.0   Min.   : 1.000   Min.   :  0.00
 1st Qu.: 5.0   1st Qu.: 4.000   1st Qu.: 21.00
 Median :11.0   Median : 7.000   Median : 52.00
 Mean   :11.2   Mean   : 7.913   Mean   : 45.85
 3rd Qu.:17.0   3rd Qu.:11.000   3rd Qu.: 66.00
 Max.   :34.0   Max.   :33.000   Max.   :200.00
                NA's   :10786
```

Figura 5.4: Summary results.csv

lumna position, observamos en primer lugar que el máximo que alcanza la posición final es 33, número superior al que actualmente estamos acostumbrados a presenciar en estas carreras, ya que lo usual es ver entre 20 y 22 pilotos en las carreras. Además, también podemos observar que hay muchos valores nulos.

```
     grid          position         laps
Min.   : 1.00   Min.   : 1.00   Min.   :  0.0
1st Qu.: 8.00   1st Qu.: 5.00   1st Qu.: 97.0
Median :17.00   Median :10.00   Median :169.0
Mean   :16.82   Mean   :10.71   Mean   :142.5
3rd Qu.:25.00   3rd Qu.:15.00   3rd Qu.:200.0
Max.   :33.00   Max.   :33.00   Max.   :200.0
                NA's   :177
```

```
      year            n
 Min.   :1950   Min.   :33.00
 1st Qu.:1952   1st Qu.:33.00
 Median :1955   Median :34.00
 Mean   :1955   Mean   :36.82
 3rd Qu.:1958   3rd Qu.:35.00
 Max.   :1960   Max.   :55.00
```

Figura 5.5: Summary 500 Millas

Podemos pensar que algunos de estos casos se corresponden a los datos perdidos que acompañan a los valores del grid = 0 y muchos otros serán los pilotos que no han conseguido terminar la carrera.

Observamos datos de carreras de hasta 200 vueltas y vemos que corresponde a las 500 millas de indianápolis, carrera que no pertenece al campeonato, aunque, en los primeros años de la fórmula 1, sí que perteneció.

En 11 temporadas, desde 1950 (inicio de la fórmula 1) hasta 1960 se consideró las 500 millas de Indianápolis como una carrera perteneciente a la competición de fórmula 1, corriendo en ella hasta 55 pilotos como podemos observar en la figura 5.5.

Por lo tanto, resulta evidente considerar nuestro conjunto de datos, sin la información correspondiente a las 500 millas de Indianápolis, ya que ésta información difiere mucho de la proporcionan el resto de carreras de fórmula 1, lo que no haría más que empeorar los resultados de su modelo a consecuencia de datos que no son de interés para el caso relevante que nos gustaría resolver a futuro, como es predecir los resultados de las próximas carreras.

Vamos ahora a seguir estudiando el conjunto de datos sin la información de estas carreras. Como vimos en la figura 5.4 existen carreras en las que no hay información de las posiciones de salida (grid).

Observamos que la mayoría corresponden a las que tampoco tienen información en la posición final, es decir, son datos perdidos. Sin embargo, por los valores perdidos en la posición final de la figura 5.4 y los que tenemos en ésta figura 5.6 la diferencia es muy grande, por lo que éstos valores perdidos en la posición corresponderán a los pilotos que no han terminado la carrera. Vamos a tomar nuestros datos sin la información de las observaciones que tienen la variable grid = 0, y vamos a estudiar ahora cómo vamos a tratar los valores faltantes en la posición final.

```
     grid      position
Min.   :0   Min.   : 5.00
1st Qu.:0   1st Qu.:10.00
Median :0   Median :15.00
Mean   :0   Mean   :13.59
3rd Qu.:0   3rd Qu.:16.00
Max.   :0   Max.   :20.00
            NA's   :1571
```

Figura 5.6: Summary grid=0

Para tratar los valores faltantes en la posición final, tenemos que tener en cuenta el número de pilotos que hay en la posición de salida. Nos sigue llamando la atención el número de pilotos que hay en parrilla ya que en algunas

temporadas era muy superior al que estamos acostumbrados a presenciar actualmente, no es muy usual que corran más de 24 pilotos. Por este motivo vamos a estudiar el número de pilotos que compitieron en cada temporada.

```
      grid          position          laps
 Min.   : 1.00   Min.   : 1.000   Min.   :  0.00
 1st Qu.: 6.00   1st Qu.: 4.000   1st Qu.: 28.00
 Median :12.00   Median : 7.000   Median : 53.00
 Mean   :11.87   Mean   : 7.858   Mean   : 47.24
 3rd Qu.:17.00   3rd Qu.:11.000   3rd Qu.: 67.00
 Max.   :34.00   Max.   :24.000   Max.   :110.00
                 NA's   :9038
```

Figura 5.7: Summary resultados

Observamos que en las primeras temporadas, el número de pilotos que disputa cada carrera varía bastante. Además, el número de carreras en cada temporada es mucho menor que el número de carreras que se disputan actualmente en cada temporada.

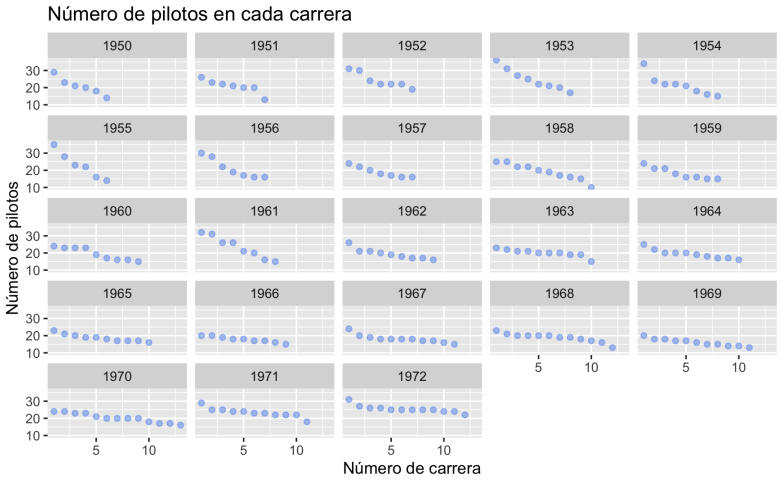


Figura 5.8: Numero de pilotos hasta 1972

En la figura 5.9 podemos observar cómo siguen oscilando el número de pilotos en las diferentes carreras de una temporada. Sin embargo, a partir de 1988 parece estabilizarse este número aunque sigue siendo muy superior al que estamos acostumbrados a presenciar. También podemos ver en la figura 5.10 cómo el número de pilotos parece estabilizarse entre el 1996 y 1997, tomando como máximo a partir de esos años 24 pilotos.

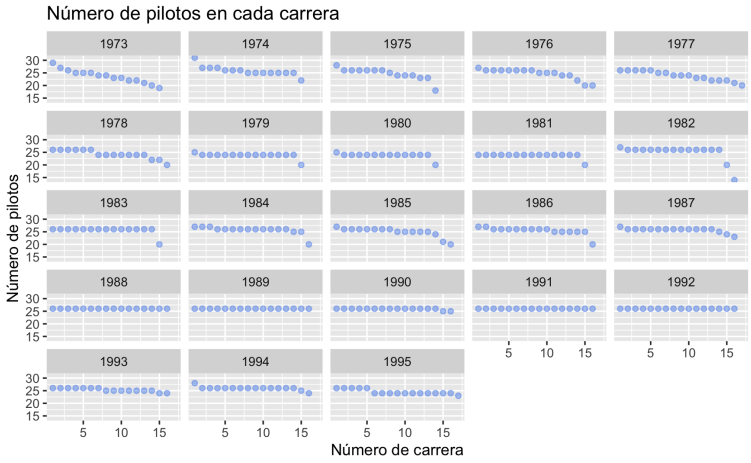


Figura 5.9: Número de pilotos hasta 1995

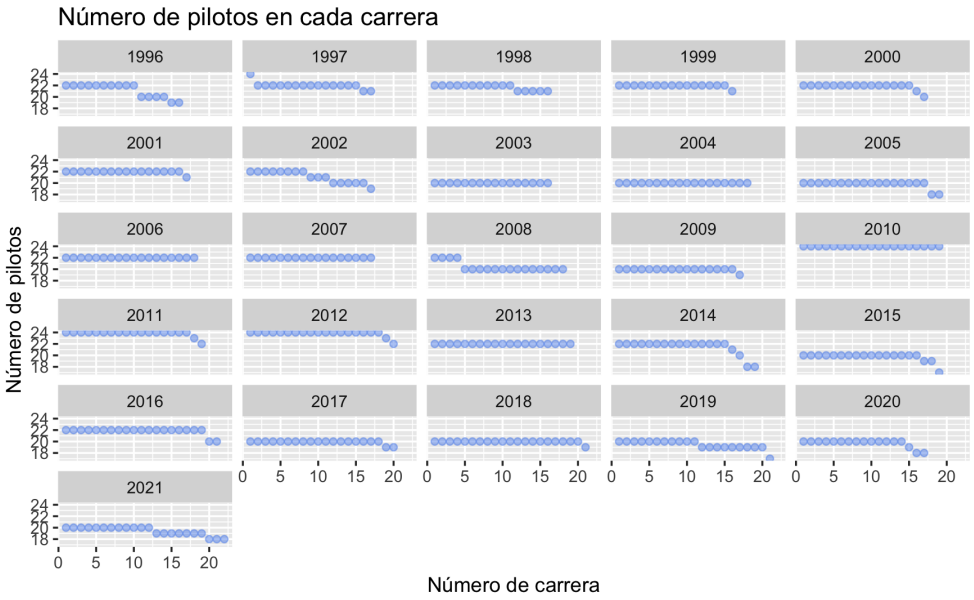


Figura 5.10: Número de pilotos hasta 2021

Para no perder información de todas las carreras en las que el número de pilotos no está estabilizado, vamos a trabajar en lugar de la posición absoluta (*position*), con la posición relativa, definiendo ésta como :

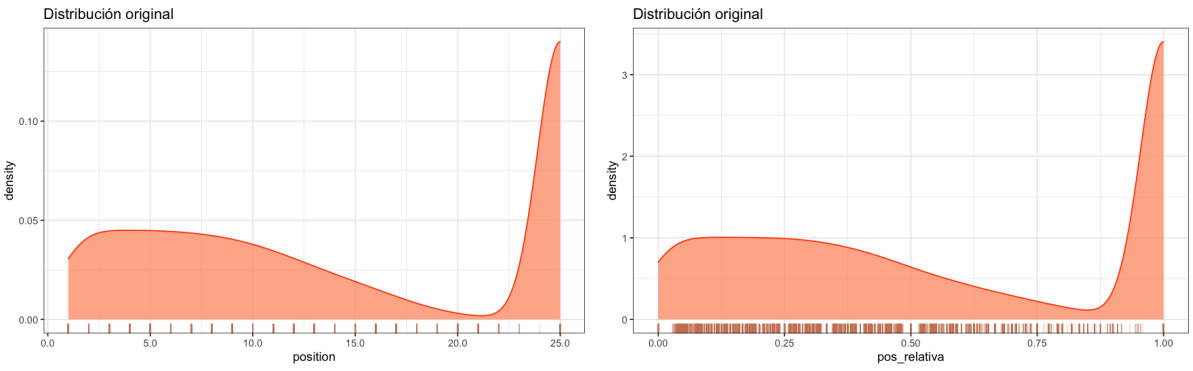
$$\text{posición relativa} = \frac{\text{número de pilotos por delante}}{\text{número de pilotos totales de partida}} \tag{5.1.1}$$

Donde número de pilotos por delante = posición absoluta − 1.

De ésta manera, la posición relativa varía en el intervalo $[0, 1)$. Así, vamos a imputar los valores NA de los pilotos que no han acabado la carrera por 1, por tanto, nuestra variable respuesta posición relativa *(pos_relativa)* estará en el intervalo $[0, 1]$.

En consecuencia, en nuestros modelos vamos a utilizar como variable respuesta esta variable posición relativa en lugar de la posición final original o posición absoluta.

Vamos a comparar por consiguiente las distribuciones de ambas variables:



(a) Distribución position (b) Distribución pos_relativa

Figura 5.11: Distribuciones

Para la variable original, hemos imputado los valores NA de los pilotos que no han acabado la carrera por 25, y por lo que observamos ambas distribuciones tienen la misma forma. En ambas podemos ver cómo la mayoría de datos se distribuyen entre los primeros puestos, teniendo una bajada brusca prácticamente hacia la mitad, y, volviendo a distribuirse una gran cantidad aunque prácticamente en los que no terminaron la carrera

5.2. Estudio de las variables.

5.2.1. Primera prueba.

En esta primera prueba, vamos a considerar todos los modelos incluyendo como variables cualitativas cada conductor individual (driverId) y cada constructor individual (constructorId), aún a riesgo de que la cantidad de valores distintos para estas variables sea demasiado grande para arrojar los mejores resultados. Al tener variables factor, vamos a codificarlas como explicamos en (2.2) por lo que tendremos todas las variables a la misma distancia unas de otras. Pasamos a definir las variables que vamos a utilizar en los modelos de la primera prueba:

- Posición de salida *(grid)*. Es una variable cuantitativa que indica la posición desde la que sale cada piloto al inicio de la carrera.

- Posición relativa *(pos_relativa)*. Definida en (5.1.1), es una variable cuantitativa, la cual será nuestra variable respuesta.

- Probabilidad de tener un accidente *(pAccidentes)*. Como en nuestro conjunto de datos, tenemos el estado en el que cada piloto acabó la carrera. Si ha tenido algún accidente en ella queda registrado a través de statusId = 3, por lo que hacemos la proporción del número de accidentes de cada piloto en cada circuito entre el número total de veces que cada piloto ha corrido en cada circuito.

$$\text{Probabilidad accidente} = \frac{\text{Nº de accidentes en cada circuito}}{\text{Nº total de veces que ha corrido en cada circuito}}$$

- Tiempo medio por vuelta. Dado que el coche depende del constructor con el que esté el piloto, resulta evidente calcular tiempo medio que cada piloto ha llevado en cada carrera dependiendo del constructor. Sin embargo, si un piloto cambia de escudería o incluso llega nuevo a la fórmula 1, no vamos a tener datos correspondientes a este. Por ello, para poder ayudar a predecir estos casos vamos a calcular:

  - Histórico del piloto *(tm_p)*. Variable cuantitativa que indica el tiempo medio en vuelta en general del piloto.
  - Histórico del piloto en cada circuito *(tm_pci)*. Variable cuantitativa que indica el tiempo medio por vuelta que lleva cada piloto en cada circuito.
  - Histórico de piloto, circuito y constructor *(tmean_pcc)* . Variable cuantitativa que indica el tiempo medio por vuelta que lleva cada piloto en cada circuito según el constructor.

El histórico del piloto y el del piloto en cada circuito sobre todo son calculadas para el caso en el que un piloto cambie de escudería. Por otro lado, para un piloto que llegue nuevo a la fórmula 1 estas variables estarían vacías por lo que en ese caso valdrían 0 para poder ejecutar el modelo.

- Paradas en Boxes. Como la estrategia es generalmente una decisión de equipo, vamos a tener las variables:

  - Media de paradas. Variable cuantitativa que indica la media de paradas que cada constructor ha hecho en cada circuito. Resulta evidente que el número de paradas que hace un equipo en cada carrera es un número entero, sin embargo, al calcular la media vamos no vamos a utilizar números enteros ya que se considera más realista dejarlo como número real para conocer entre qué números enteros puede estar el número de paradas de cada equipo.
  - Media de tiempo (en milisegundos) en paradas. Variable cuantitativa que indica la media del tiempo empleado en paradas cada constructor en cada circuito.

  En el conjunto de datos no tenemos la información de las paradas en boxes hasta el año 2011.

  Para subsanar esta carencia de información, cuando tenemos valores NA, tenemos dos opciones: imputar estos valores o eliminar las observaciones que corresponden a los valores nulos. Como tenemos la información para cada circuito, en la práctica nos hemos encontrado con algunos casos en los que no tenemos ninguna información para esta variable, es decir, toda la columna correspondiente a esta variable es NA. Por tanto, para no perder la información correspondiente a estos circuitos, hemos decidido imputar esta variable por la media de las medias de las paradas en el resto de circuitos. Los resultados que vamos a obtener los compararemos según imputamos los valores nulos o los eliminamos.

- Número de ronda *(round)*. Ronda dentro de la temporada en la que está siendo la carrera, estableciendo así el orden en que se ha corrido cada circuito cada año.

El proceso de creación de estas variables puede verse en el anexo (A).

Veamos ahora la correlación entre estas variables en la figura 5.12. Para ello, necesitamos que las variables sean cuantitativas, por lo que vamos a estudiar la correlación sin las variables driverId, constructorId ya que se tratan como factores. Además, no podemos observar la correlación de las variables de paradas en boxes porque no tenemos toda la información de ellas. Por tanto, al crear el modelo, imputaremos éstas.

Podemos ver cómo la variable grid es la que más está correlada con la posición relativa, seguida ésta de la probabilidad de accidentes.

Como era de esperar, las variables explicativas que más correladas están entre sí son las que pertenecen al tiempo medio por vuelta.

Cabe señalar que el estudio realizado en este apartado se empleará como punto de partida para los primeros modelos que crearemos en el apartado (5.3.1).

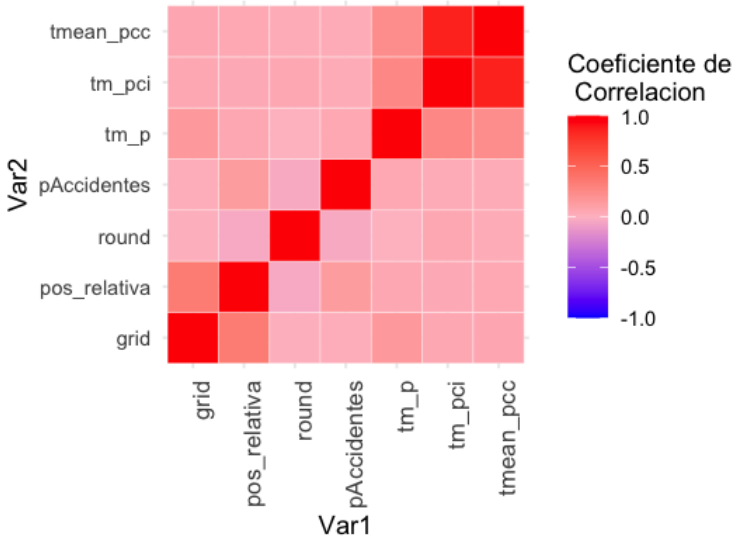


Figura 5.12: Correlación variables primera prueba

5.2.2. Segunda prueba.

Teniendo en cuenta como hemos explicado en (5.2.1) al utilizar en la primera prueba las variables driverId, constructorId y tansformarlas a variables dummy, hemos asumido que todos los pilotos están a la misma distancia unos de otros, al igual ocurre con los constructores. Sin embargo, es evidente que los pilotos no se diferencian igualmente entre sí, sino que se asemejan más a unos que otros, y lo mismo ocurre con los constructores. Vamos a introducir en los modelos de esta segunda prueba, nuevas variables que asemejen un poco más a los pilotos entre sí y a los constructores y sustituyan a las variables factor driverId, constructorId por aspectos que caractericen su rendimiento, y predigan así en función de las variables que caracterizan a los pilotos y escuderías en lugar de sus nombres concretos.

Vamos a definir ahora las variables que vamos a utilizar en la segunda prueba:

- Número de carreras disputadas. Para tener información sobre los pilotos si cambian de escudería o llegan nuevos a la fórmula 1, vamos a crear tres variables diferentes:
  - Histórico del piloto *(tc)*. Variable cuantitativa que indica el número total de carreras que un piloto ha disputado.
  - Histórico del piloto en cada circuito *(nv_pilo_c)*. Variable cuantitativa que indica el número de veces que ha corrido cada piloto en cada circuito.
  - Histórico del piloto en cada circuito y constructor *(nv_pilo_cc)*. Variable cuantitativa que indica el número de veces que cada piloto ha corrido en cada circuito según el constructor.
- Posición media relativa de cada piloto en cada circuito *(posic_m)*. Variable cuantitativa.
- Posición media relativa de cada piloto en cada circuito con cada constructor. *(pos_mean)* Variable cuantitativa.

- Probabilidad de hacer podium. Vamos a crear el histórico para cada piloto.
  - Histórico del piloto *(prob_p)*. Variable cuantitativa que indica la probabilidad de hacer podium de cada piloto.

$$\frac{\text{Nº de veces que ha conseguido un podium}}{\text{Nº total de carreras disputadas}}$$

  - Histórico del piloto en cada circuito *(probc)*. Variable cuantitativa que indica la probabilidad de hacer podium de cada piloto en cada circuito.

$$\frac{\text{Nº de veces que ha conseguido un podium en cada circuito}}{\text{Nº total de carreras disputadas en cada circuito}}$$

  - Histórico del piloto, circuito y constructor *(prob)* . Variable cuantitativa que indica la probabilidad de hacer podium de cada piloto en cada circuito según el constructor.

$$\frac{\text{Nºde veces que ha conseguido un podium con cada constructor en cada circuito}}{\text{Nº total de carreras disputadas en cada circuito con cada constructor}}$$

- Número de veces que cada constructor ha corrido en un circuito *(n)*.
- Puntuación media conseguida por cada constructor en el total de años que ha participado en la fórmula 1 *(points_m)*.

$$\frac{\text{Suma de las puntuaciones en todas las carreras}}{\text{Nº total de carreras disputadas}}$$

- Puntuación media conseguida por cada constructor en cada circuito *(point_mean)*.

$$\frac{\text{Suma de las puntuaciones en todas las carreras en cada circuito}}{\text{Nº total de carreras disputadas en cada circuito}}$$

El proceso de creación de estas variables puede verse en el anexo (B).

Estas son las variables que vamos a añadir nuevas con respecto a las definidas en (5.2.1), excepto como hemos mencionado, las variables factor driverId, constructorId.

Si observamos en la figura 5.13 las relaciones con las variable respuesta, observamos que las que más relacionadas están son pos_mean, posic_m, grid, prob.

Es evidente que las variables que obtenemos de la probabilidad de hacer podium están muy correladas entre sí, porque realmente estamos utilizando la misma información para estas variables. Lo mismo ocurre con las variables de la probabilidad de hacer podium.

Cabe señalar que el estudio realizado en este apartado se empleará como punto de partida para los modelos que crearemos en el apartado (5.3.2).

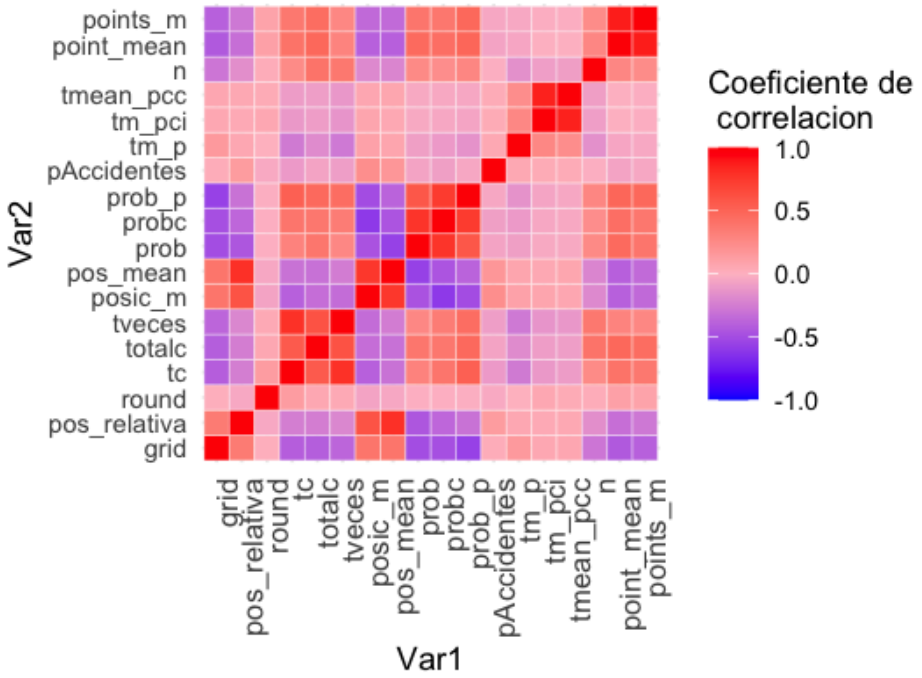
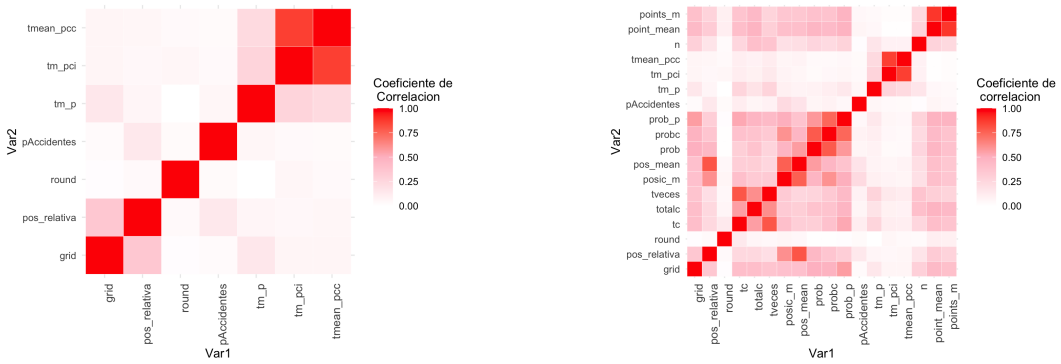


Figura 5.13: Correlación variables segunda prueba

Como únicamente nos interesa saber si las variables están relacionadas (independientemente si la relación es directa o indirecta) vamos a representar estos mapas de calor en valor absoluto:



(a) Correlación en valor absoluto prueba 1. (b) Correlación en valor absoluto prueba 2.

Figura 5.14: Correlaciones

Comparando las correlaciones de las variables de ambas pruebas, podemos observar en la figura 5.14b como en la segunda prueba, algunas de las variables introducidas nuevas poseen una correlación alta con la variable respuesta, con lo cual explican bastante bien a la variable respuesta destacando así la posición media relativa de cada piloto en cada circuito según el constructor. Sin embargo en la primera prueba la variable que más correlada estaba con la posición relativa era únicamente la posición de salida. Sin embargo, aunque las variables nuevas estén más correladas con la variable respuesta, también podemos observar cómo existe relación entre las variables explicativas, sobre todo, las que son

calculadas a partir de los mismos datos. Esto ocurre en ambas pruebas, aunque en la segunda prueba es en la que más variables de esta forma tenemos. Cabe resaltar que en este caso vamos a estar introduciendo en nuestro modelo variables fuertemente relacionadas, es decir, variables que nos proporcionan casi la misma información. Esto nos puede causar un problema de multicolinealidad, el cual trataremos con ingeniería de características (2.2).

Como hemos destacado a la variable posición de salida relacionada con la variable respuesta, vamos a coger los datos de la posición de clasificación, la posición final en cada circuito a lo largo de la historia y vamos a ver el número de posiciones ganadas o perdidas con respecto a la posición de salida y a la posición final, dependiendo de cada circuito. Es decir, vamos a calcular la diferencia de posición de salida (*grid*) - posición final (*position*). Teniendo en cuenta que si la diferencia es positiva entonces ha ganado posiciones, por el contrario, si es negativa, ha perdido posiciones.

Luego en la figuras 5.15 y 5.16 podemos observar en color azul las posiciones ganadas y en color negro las posiciones perdidas. Vamos a estudiar estas diferencias con respecto a los que han salido entre los 10 primeros con respecto a su posición final.

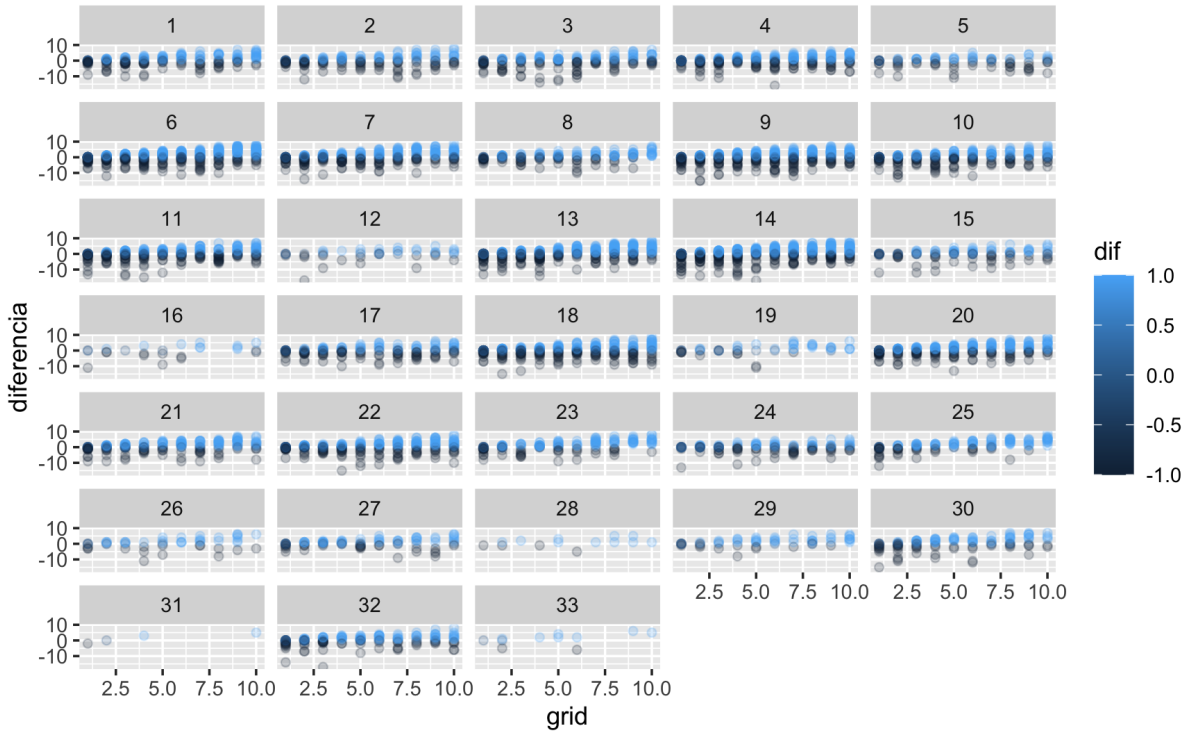


Figura 5.15: Diferencia en las posiciones

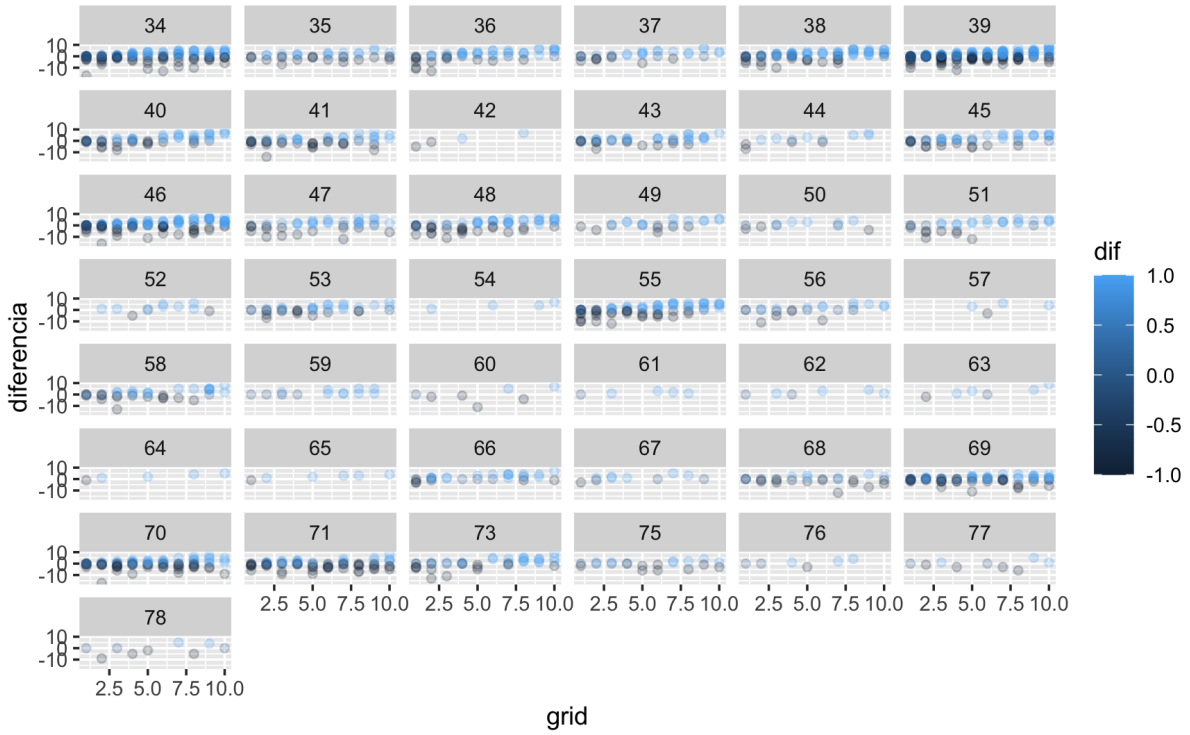


Figura 5.16: Diferencia en las posiciones

Podemos observar como en la mayoría de los circuitos los que suelen salir a partir del 5/6 suelen ganar más posiciones de las que pierden, en cambio los que salen entre los primeros pierden más posiciones de las que ganan. Evidentemente, quien sale desde la pole solo puede perder posiciones o terminar primero.

5.3. Modelización.

Para responder a la pregunta, vamos a utilizar modelos de regresión para predecir la posición que tomará cada piloto en cada circuito. Resulta evidente que cada circuito tiene sus características, por lo que en primer lugar vamos a construir un modelo diferente para cada circuito. Dado que en el modelo no podemos utilizar datos que se obtendrán en la carrera, vamos a utilizar resultados en el mismo circuito en temporadas anteriores.

Vamos a comparar los resultados obtenidos en cuatro modelos diferentes que serán: Modelo lineal, Árbol de decisión, Random Forest y Red Neuronal.

Siguiendo los pasos definidos en la página 30 hacemos lo siguiente:

1º Vamos a crear una división en conjunto de entrenamiento 70 % y conjunto test 30 %.

2º Definimos cada modelo que vamos a utilizar de los definidos anteriormente. Para los modelos de árboles de regresión y random forest, definimos los parámetros que queremos ajustar. En el caso de árbol de regresión vamos a ajustar coste de complejidad (cost_complexity) y profundidad del árbol (tree_depth). En el caso de random forest mtry, trees, min_n.

3º Para los modelos de árboles de regresión y random forest, creamos la rejilla (grid) sobre la que vamos a buscar los valores de los parámetros que queremos ajustar.

4º Para estos modelos, vamos a utilizar validación cruzada para poder seleccionar los parámetros. En estos casos, vamos a utilizar validación cruzada con 5 pliegues.

5º Según estemos en la primera o segunda prueba, este punto diferirá . El primer paso que tenemos que dar es imputar las variables que contengan NA, en este caso serán el la media de paradas en boxes y el tiempo medio en boxes. A continuación, si estamos en la primera pruebla, tenemos que convertir las variables cualitativas (driverId, constructorId) en variables dummys. Además, en ambas pruebas, cuando creemos un modelo que contenga la variable circuitId la trataremos también como variable categórica por lo que crearemos variables dummys para ello. Seguidamente estos pasos son comunes para ambas pruebas: eliminamos las variables que contengan un único valor, centramos y escalamos y finalmente eliminamos las variables que estén altamente correladas.

6º Creamos el workflow.

7º Ajustamos el modelo.

8º Seleccionamos el mejor modelo, en nuestro caso según la métrica **mae**, definida en (3.5.3).

Vamos a crear los modelos indicados para comparar los valores de la métrica por las técnicas: imputación de los valores faltantes y eliminación de las observaciones con valores faltantes. Además compararemos las métricas de los modelos obtenidas en las diferentes pruebas. Todas estas métricas las vamos a obtener para cada circuito, ya que es lógico pensar que no en todos los circuitos se actúa de la misma forma. Cabe destacar que, tenemos información a cerca de 76 circuitos, pero al eliminar las observaciones con valores faltantes, vamos a tener información de 33 circuitos, es decir, estamos perdiendo la información correspondiente a la mitad de circuitos de la base de datos.

¿Qué ocurre si nos encontramos con un circuito completamente nuevo? Al no tener datos de temporadas anteriores, por el método anterior sería imposible hacer alguna predicción. Recuérdese que, para solventar esa dificultad, la sección (5.2) hemos definido variables para cuando un piloto llegue nuevo a la fórmula 1 o cambie de escudería. Tendremos que crear un modelo que contenga la variable circuito para cuando queramos predecir resultados en un circuito nuevo, ya que no vamos a tener información anterior de los pilotos en dicho circuito, por lo que también vamos a crear estos modelos teniendo en cuenta la variable circuito, la cual será una variable factor. Por ello, tendremos que crear variables "dummy"de esta forma, para la primera prueba tendremos que crear estas variables de tres factores diferentes, en cambio, para la segunda prueba solo tendremos que crear las variables "dummy "de cada circuito.

Vamos a recoger en tablas y gráficamente los valores de la métrica, para poder realizar comparaciones entre las diferentes pruebas y técnicas. En las tablas vamos a reflejar sobre todo los valores más significativos, indicando en cada caso de color rojo los valores anómalos para la métrica y en azul el mejor valor que toma dentro de cada modelo.

Haciendo referencia a los conjuntos de datos utilizados en R:

- Primera prueba.

  `conjunto` : contiene todas las observaciones de todas las variables definidas para la primera prueba.

  `conjunto2` : contiene las observaciones del primer conjunto cuyas variables no contienen valores NA.

  `conjunto_c` : contiene toda la información que en el primer conjunto además de incluir la variable circuitId.

  `conjunto2_c` : contiene toda la información que en el segundo conjunto además de incluir la variable circuitId.

- Segunda prueba.

  `conjunto3` : contiene todas las observaciones de todas las variables definidas para la segunda prueba.

  `conjunto4` : contiene las observaciones del conjunto3 cuyas variables no contienen valores NA.

  `conjunto_c3` : contiene toda la información del conjunto3 además de incluir la variable circuitId.

  `conjunto_c4` : contiene toda la información del conjunto4 además de incluir la variable circuitId.

5.3.1. Primera prueba.

A partir de las variables definidas en (5.2.1) vamos a crear los modelos ya mencionados según las técnicas de imputación y de eliminación. Así obtendremos los valores de la métrica para cada circuito según el modelo y podremos comparar los distintos modelos entre sí y también según las dos técnicas que hemos mencionado.

Imputando los datos.

Mediante la técnica de imputación de datos para las variables de la primera prueba, seleccionando el mejor modelo para la métrica, en la siguiente tabla 5.1 recogemos los valores de la métrica para cada circuito.

circuitId	Modelo lineal	Árbol de regresión	Random Forest	Red Neuronal
1	0.3949564	0.3267304	0.3297490	0.3453368
2	2.6672580	0.2549596	0.2611678	0.3249200
3	0.2423025	0.2016050	0.2021398	0.2692341
4	0.5816955	0.2746194	0.2721322	0.3257670
5	0.5178016	0.2063781	0.2064694	0.2771622
6	0.4080296	0.3367408	0.3489846	0.3631525
...				

circuitId	Modelo lineal	Árbol de regresión	Random Forest	Red Neuronal
16	1.1622922	0.3588544	0.3538483	0.3794301
17	0.2827349	0.2138879	0.2145319	0.2751174
18	0.3674427	0.3033192	0.3102486	0.3408274
19	0.4274022	0.2566275	0.2827733	0.3663935
20	0.6202024	0.3438701	0.3454756	0.3664762
...				
41	0.4934907	0.2584718	0.2556948	0.3056952
42	2.0700000	0.3858824	0.3088661	0.3423219
43	0.5632781	0.3449536	0.3755424	0.3729560
44	0.4195789	0.3195727	0.3070025	0.3599789
45	0.4809598	0.3560218	0.3496178	0.3837303
46	0.4570273	0.3685461	0.3536748	0.3602329
47	0.4967642	0.3099282	0.3197434	0.3175050
48	0.4042456	0.3241792	0.3378339	0.3503938
49	0.5195548	0.3200834	0.3615197	0.3828563
50	0.5873269	0.3681792	0.3516247	0.3615218
51	1.2132723	0.3301631	0.3277892	0.3220150
52	0.5145833	0.3482639	0.4186709	0.3839775
53	0.7560060	0.3066842	0.2999487	0.3651805
54	3.8933333	0.4360000	0.4037358	0.4381045
55	0.5681733	0.3886128	0.3723064	0.3786278
56	0.4230059	0.3193410	0.3544286	0.3573350
57	0.4250000	0.4250000	0.4063309	0.4230417
58	0.5346736	0.3786805	0.3619525	0.4107521
59	0.9037037	0.3053498	0.3975144	0.4067873
60	0.2484472	0.3140528	0.3098744	0.3137788
61	1.1600000	0.4306667	0.5268148	0.4062955
62	1.2062500	0.4147727	0.4872222	0.4189497
...				
68	0.4719746	0.1637215	0.1707966	0.2560298
69	0.3625617	0.2223686	0.2349609	0.2904943
70	0.5571014	0.2170468	0.2242700	0.2942830
71	1.1236375	0.2511333	0.2449945	0.2835508
73	0.3877957	0.3347619	0.2628214	0.3057043
75	0.2344988	0.2202381	0.1899421	0.2193441
76	0.2865894	0.4250000	0.1686818	0.4032588
77	1.3845031	0.3583333	0.1979167	0.3444341
78	0.5363466	0.3000000	0.1775000	0.2400476

Tabla 5.1: Métricas primera prueba imputando.

Teniendo presente que nuestra variable objetivo era la posición final que tendremos en la carrera, a partir de la información sobre piloto, circuito, posición de partida e información calculada del rendimiento de años anteriores. Además, recordar que esta variable se escaló para encontrarse en el intervalo $[0, 1]$, de modo que cuanto más nos acerquemos a 0 más cerca estaremos de ganar y cuanto más nos acerquemos más a 1 más cerca estaremos de ser últimos, entendiendo que el 1 se corresponderá con no haber llegado terminado la carrera.

En primer lugar, podemos observar como en el modelo lineal aparecen valores de la métrica mayores que 1, lo cual no tiene mucho sentido porque nuestra variable objetivo, como hemos mencionado, toma valores en el intervalo $[0, 1]$. Sin embargo, en el resto de modelos, no obtenemos ningún valor por encima de 1, con lo cúal obtenemos una mejora.

Comparando el resto de modelos con el modelo lineal, obtenemos una mejora considerable de la métrica sobre todo en el árbol de regresión y el random forest. Comparándolo ahora con los resultados de la red neuronal, en la mayoría de circuitos tenemos un mejor mae en la red neuronal, sin embargo existen una serie de circuitos en el que el mae no mejora del modelo lineal a la red neuronal.

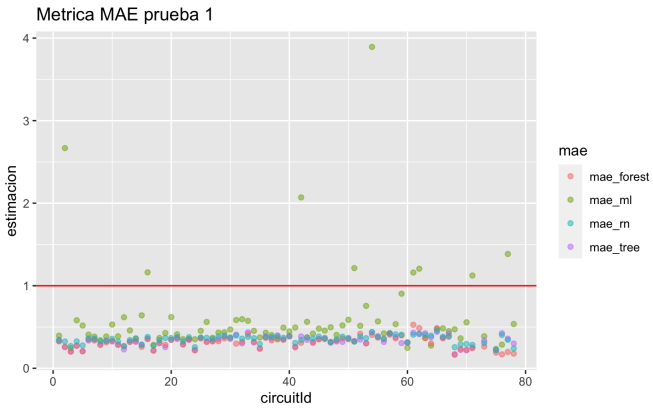


Figura 5.17: Mae prueba 1.

Centrándonos ahora en el árbol de regresión y el bosque aleatorio, tenemos que hay muchos circuitos en los que el mae es mejor en el bosque aleatorio y el resto es mejor en el árbol de regresión, habiendo entre éstos poca diferencias. No obstante aunque el árbol diera mejor resultado en algún caso, cabe resaltar que es mucho menos fiable, ya que nos basamos en una muestra muy concreta, mientras que el bosque se basa en el consenso entre muchos posibles árboles, de modo que lo normal es que capture mejor la función subyacente.

Podemos a ver gráficamente las métricas descritas en la tabla 5.1, en la figura 5.17.

Para hacer mejor las comparaciones entre ambos modelos, vamos a limitarnos a los que tienen un mae inferior a 1, figura 5.18.

Observamos un claro predominio del valor de la métrica del modelo lineal superior a 0.5, en cambio, el valor en los demás modelos está mayoritariamente por debajo de 0.5. Además podemos destacar también que en la mayoría de los modelos, la métrica del random forest es la mejor, es decir, el modelo es el que mejor aproxima los datos.

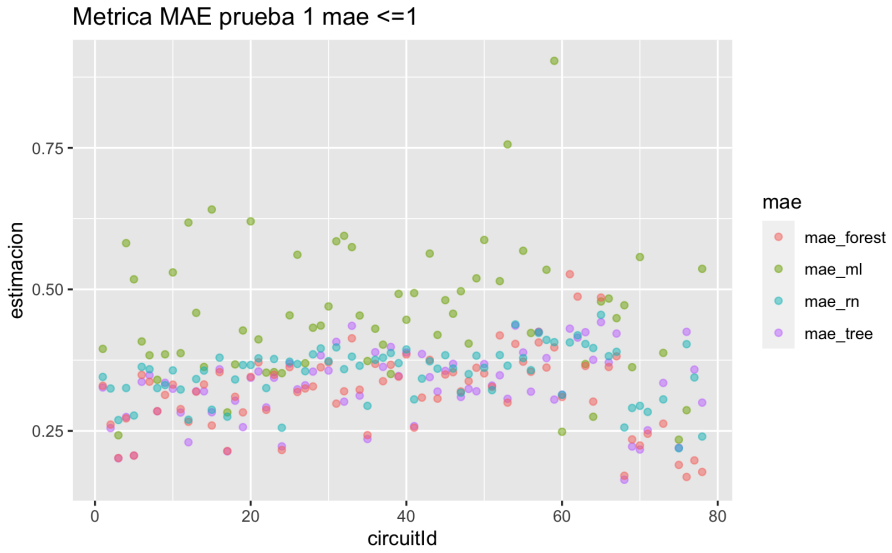


Figura 5.18: Mae prueba 1 con mae ≤ 1.

Eliminando observaciones.

Mediante la técnica de eliminación para las variables de la primera prueba, seleccionando el mejor modelo para la métrica, en la siguiente tabla 5.2 recogemos los valores de la métrica para cada circuito.

circuitId	Modelo lineal	Árbol de regresión	Random Forest	Red Neuronal
1	0.3359059	0.3228616	0.3141560	0.3373982
2	0.4007878	0.2772628	0.2678032	0.3209511
3	0.3270378	0.2368692	0.2140592	0.2707034
4	2.1151069	0.2496695	0.2437479	0.2974697
...				
32	1.1004042	0.2338613	0.2673860	0.3231953
34	0.4909143	0.3038076	0.3315712	0.3317786
35	0.3734245	0.2357061	0.2425675	0.2814554
39	56.9502208	0.3254790	0.3341160	0.3488061
68	0.4719746	0.1637215	0.1707966	0.2416170
69	0.3625617	0.2223686	0.2349609	0.2827344
70	0.5571014	0.2170468	0.2242700	0.2862820
71	1.1236375	0.2511333	0.2449945	0.2789395
73	0.3877957	0.3347619	0.2628214	0.3214737
75	0.2344988	0.2202381	0.1899421	0.2172106
76	0.2865894	0.4250000	0.1686818	0.4368259
77	1.3845031	0.3583333	0.1979167	0.3476202
78	0.5363466	0.3000000	0.1775000	0.2954655

Tabla 5.2: Métricas primera prueba eliminando.

Por esta técnica podemos ver de nuevo, que existen valores superiores a 1, además se encuentran únicamente en el modelo lineal aunque, evidentemente hay menos debido a que al eliminar las observaciones con valores faltantes, estamos eliminando todas las observaciones correspondientes a muchos circuitos.

De esta manera, podemos observar que, únicamente los valores superiores a 1, coinciden por ambas técnicas en los circuitos 71 y 77. También podemos observar cómo por las dos técnicas obtenemos los mejores valores en cada modelo en los mismos circuitos, señalando además que toman los mismos valores.

Veamos gráficamente las métricas obtenidas en la tabla 5.2, obviando los circuitos en los que encontramos un valor superior a 1.

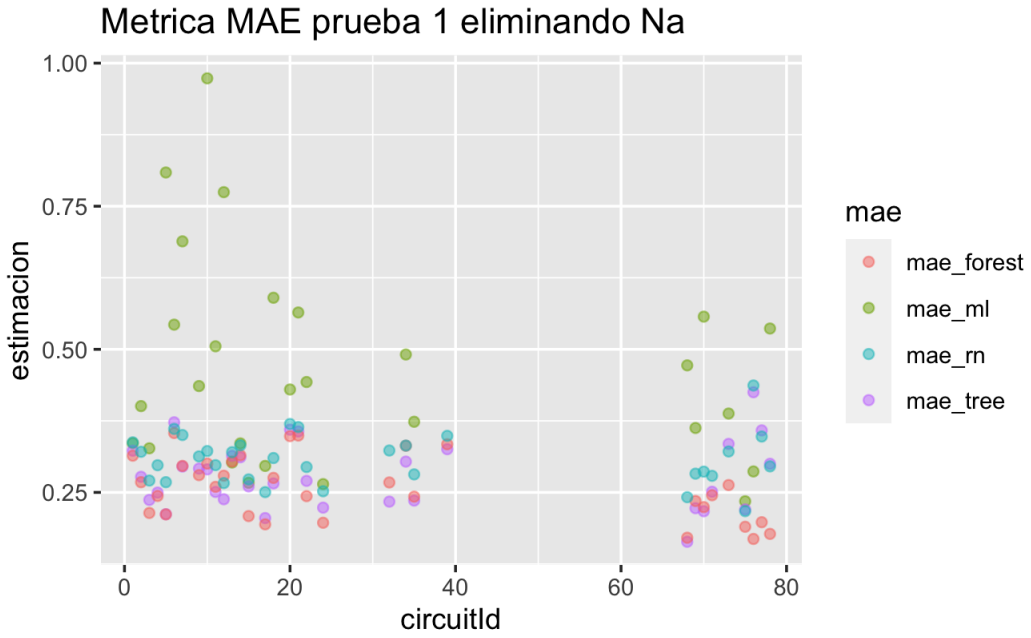


Figura 5.19: Mae prueba 1 eliminando NA.

Como vimos en la figura 5.18 eliminando los valores faltantes ocurre prácticamente lo mismo: el modelo de random forest es el que mejor ajusta,seguido del árbol de regresión y la red neuronal, los cuales toman valores por debajo de 0.5, aunque por la técnica de imputación para estos modelos existían una serie de valores por encima de este número. En cambio el modelo lineal es el que posee los valores más altos, siendo por tanto el que produce peores estimaciones.

Circuito como variable.

A partir de las variables definidas para la primera prueba además de introducir otra variable categórica que hace referencia al circuito, vamos a obtener los valores de la métrica para cada modelo según las dos técnicas que hemos venido indicando.

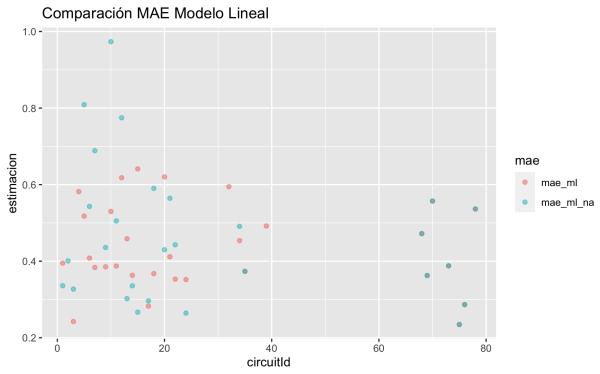
Comparando los valores según la técnica podemos observar como para el método de eliminación éstos valores son más bajos, sin embargo, como ya hemos mencionado antes, estamos perdiendo información de muchos circuitos.

	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
Mae	0.338081908	0.3471096	0.3242174	0.349988909
	Eliminando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
Mae	0.26125449	0.3161221	0.2747255	0.31368660

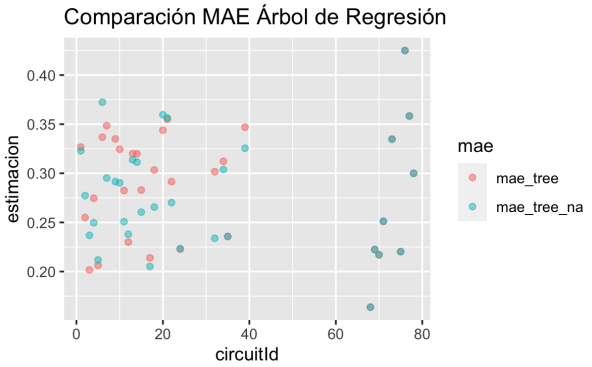
También podemos señalar que la métrica por el modelo de random forest es la mejor. Por el contrario, la métrica que obtenemos por el modelo de red neuronal es la mas alta por lo tanto, es la peor.

Vamos a comparar ahora los valores de la métrica según la técnica aplicada, figura 5.20 en rojo tenemos los valores para la técnica de imputación y en azul para la técnica de eliminación. Teniendo presente que vamos a perder información de los circuitos en los que todos los valores a imputar son faltantes. En primer lugar, podemos destacar cómo en los modelos: lineal, árbol de regresión y random forest, desde el circuito 68 tienen los mismos valores por las diferentes técnicas. Sin embargo, en la red neuronal, figura 5.20d, obtenemos valores diferentes para todos los circuitos según las dos técnicas.

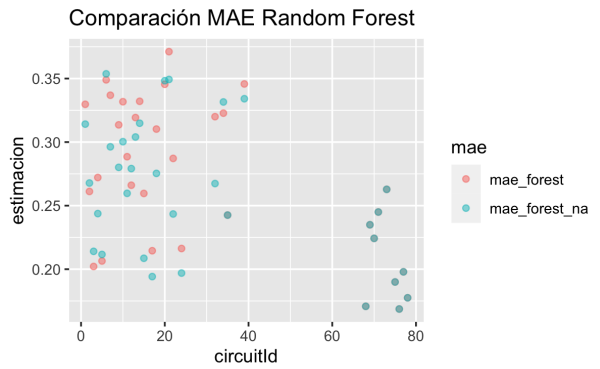
Para el modelo lineal, figura 5.20a no existe un predominio claro de unos mejores valores para una de las dos técnicas. Sin embargo para los modelos de árbol de regresión y random forest, figuras 5.20b y 5.20c respectivamente.



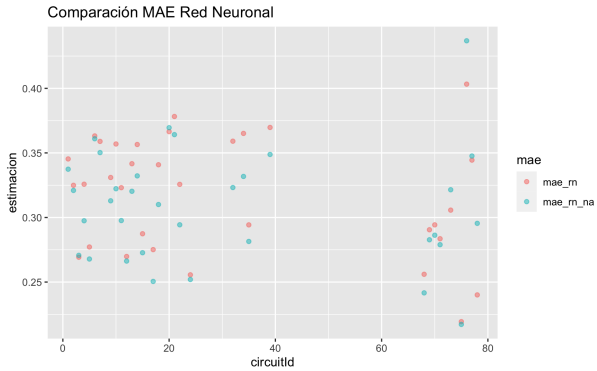
(a) Modelo Lineal.



(b) Árbol de regresión



(c) Random forest.



(d) Red Neuronal.

Figura 5.20: Comparación por ambos métodos.

5.3.2. Segunda prueba.

A partir de las variables definidas en (5.2.2) vamos a crear los modelos ya mencionados según las técnicas de imputación y de eliminación. Así obtendremos los valores de la métrica para cada circuito según el modelo y podremos comparar los distintos modelos entre sí y también según las dos técnicas que hemos mencionado.

Imputando los datos

Mediante la técnica de imputación de datos para las variables de la segunda prueba, seleccionando el mejor modelo para la métrica, en la siguiente tabla 5.3 recogemos los valores de la métrica para cada circuito.

circuitId	Modelo lineal	Árbol de regresión	Random Forest	Red Neuronal
1	2.111359e-01	0.24786524	0.25504102	0.3189394
2	1.732102e-01	0.19466358	0.18908702	0.2790408
3	1.427927e-01	0.15929420	0.15300239	0.2223470
4	1.623603e-01	0.17041106	0.17981365	0.3075616
5	1.313810e-01	0.13632667	0.14460110	0.2396013
...				
51	1.443249e-02	0.05267801	0.04926875	0.2521010
52	1.586450e-01	0.16472222	0.12042063	0.2751404
53	1.248299e-01	0.16854861	0.16187970	0.3771008
54	2.887160e-16	0.43600000	0.12488889	0.2773349
55	1.033160e-01	0.12132974	0.11894035	0.3574291
56	2.724362e-01	0.26342071	0.25488736	0.3039683
57	3.044955e-16	0.42500000	0.05236111	0.3545027
58	1.632074e-01	0.17760066	0.18458043	0.3728122
59	3.249579e-02	0.12888889	0.06629630	0.3031712
60	1.421615e-16	0.31405280	0.08364389	0.3138402
...				
67	4.256709e-02	0.08111888	0.02547931	0.4239934
68	1.319622e-01	0.14568800	0.12203857	0.2032910
69	1.572314e-01	0.18455992	0.18435245	0.2591208
70	1.817868e-01	0.18350039	0.19498415	0.2424766
71	1.822226e-01	0.17460477	0.19204727	0.2226347
73	2.167360e-01	0.20909787	0.22224007	0.2532103
75	1.231703e-01	0.16916667	0.11673611	0.1893824
76	6.111775e-17	0.42500000	0.05222222	0.3127700
77	8.103550e-17	0.35833333	0.09416667	0.5051613
78	5.230510e-16	0.30000000	0.09416667	0.2411461

Tabla 5.3: Métricas segunda prueba imputando.

En primer lugar, es claro que se observa una mejora clara de la métrica en los modelos, pero cabe destacar el modelo lineal ya que ha tenido una mejora significativa, además de no tener valores por encima de 1.
Veamos los resultados de esta tabla 5.3 gráficamente:

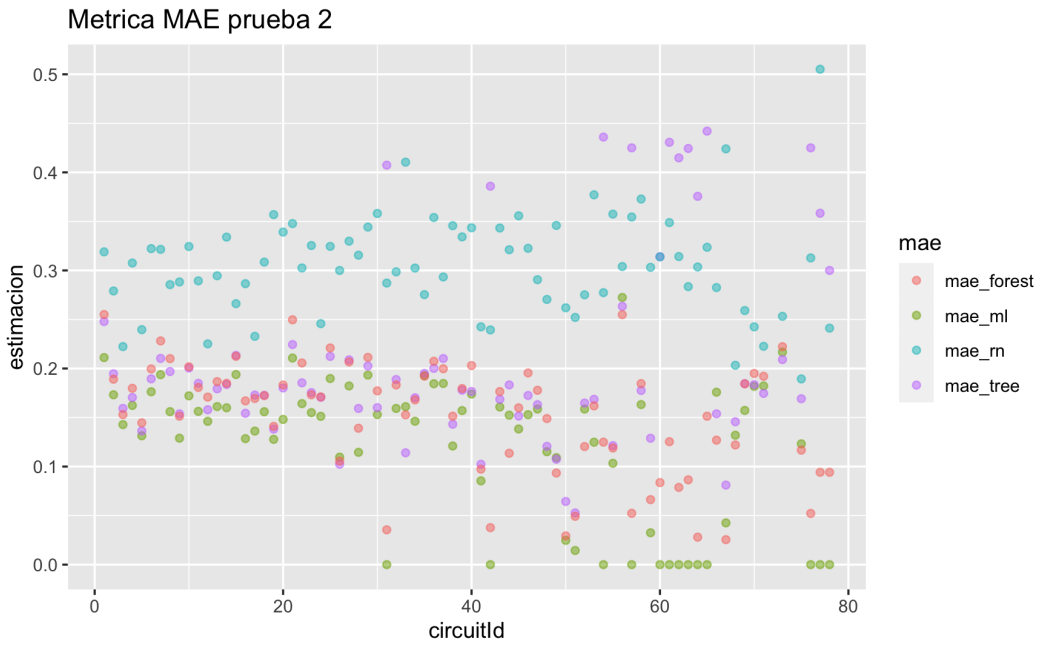


Figura 5.21: Mae prueba 2

Observando la figura 5.21, vemos que todos los valores están distribuidos en el intervalo $(0, 0.51)$, lo cuál refleja la primera mejora con respecto a la primera prueba ya que había métricas muy superiores a estos datos. Además de una clara diferencia de los valores a partir de 0.25 ya que la mayoría de los valores superiores a este corte corresponden a las obtenidas a través de la red neuronal y algunas del árbol de regresión. Si nos centramos en el intervalo $(0, 0.2)$ tenemos por la figura 5.22 para los primeros 40 circuitos poseen los valores de la métrica de los tres modelos bastante similares. Sin embargo a partir del circuito 40 el valor del modelo lineal mejora significativamente con respecto a los valores del resto de modelos.

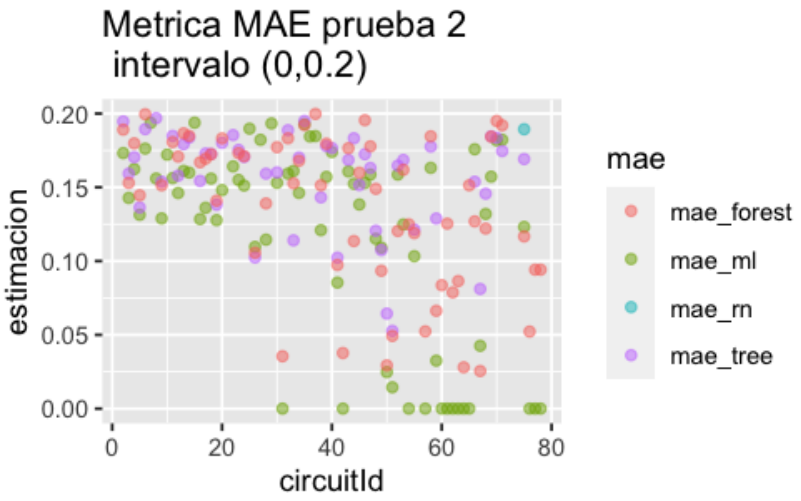


Figura 5.22: Mae prueba 2 intervalo $(0, 0.2)$

Eliminando los datos.

Mediante la técnica de eliminación para las variables de la segunda prueba, seleccionando el mejor modelo para la métrica, en la siguiente tabla 5.4 recogemos los valores de la métrica para cada circuito.

circuitId	Modelo lineal	Árbol de regresión	Random Forest	Red Neuronal
1	2.359260e-01	0.2844211	0.2854072	0.3231279
2	1.870156e-01	0.2067417	0.2311247	0.2674320
3	1.613420e-01	0.1827443	0.1968198	0.2425850
...				
22	1.677383e-01	0.2050564	0.2018900	0.2765533
24	1.525113e-01	0.1563143	0.1769219	0.2265117
32	1.956637e-01	0.2174147	0.2247905	0.3033235
34	1.674692e-01	0.2068621	0.2293820	0.2717639
35	1.928107e-01	0.1949914	0.2042374	0.2879746
39	1.647775e-01	0.1647791	0.2117724	0.2925667
68	1.319622e-01	0.1456880	0.1703384	0.2482515
69	1.572314e-01	0.1845599	0.2057780	0.2734587
70	1.817868e-01	0.1835004	0.2165224	0.2552719
71	1.822226e-01	0.1746048	0.2131029	0.2325707
73	2.167360e-01	0.2090979	0.2203900	0.2575063
75	1.231703e-01	0.1691667	0.1712526	0.1835655
76	6.111775e-17	0.4250000	0.1169048	0.3492632
77	8.103550e-17	0.3583333	0.2473194	0.2801743
78	5.230510e-16	0.3000000	0.1054167	0.2876389

Tabla 5.4: Métricas segunda prueba elimiando.

A diferencia de lo que pasaba en la prueba 1, los mejores valores de los modelos en los circuitos no coinciden en los modelos salvo en el modelo lineal y en la red neuronal. Veamos gráficamente los valores obtenidos en la tabla 5.4:

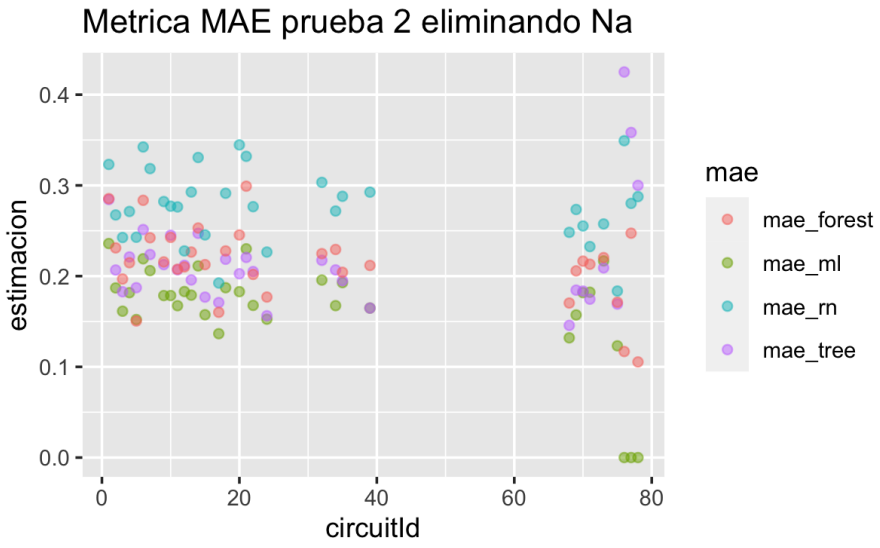


Figura 5.23: Mae prueba 2 eliminando NA.

Como vimos en la figura 5.22 eliminando los valores faltantes sucede básicamente lo mismo: los valores obtenidos por la red neuronal son los mayores aunque ahora la mayoría son menor que 0.4 . Si nos restringuimos a los 40 primeros circuitos, teniendo en cuenta que hay bastantes que son suprimidos podemos observar como los valores por los demás modelos son en gran medida iguales.

Circuito como variable.

A partir de las variables definidas para la segunda prueba además de introducir la variable categórica que hace referencia al circuito, vamos a obtener los valores de la métrica para cada modelo según las dos técnicas que nos hemos venido refiriendo.

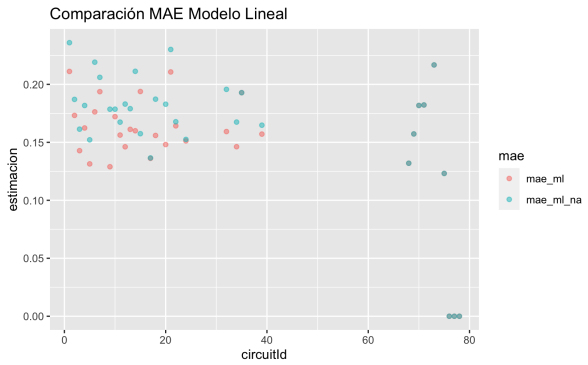
	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
Mae	0.1455162	0.1432668	0.2060327	0.34026773
	Eliminando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
Mae	0.1735876	0.1767279	0.2113356	0.3128904

Podemos observar como en este caso, comparando con lo obtenido para la primera prueba, los valores por la técnica de imputación son menores que por la técnica de eliminación. Aun así, los valores que obtenemos por la red neuronal siguen siendo los más altos comparados con los demás modelos.

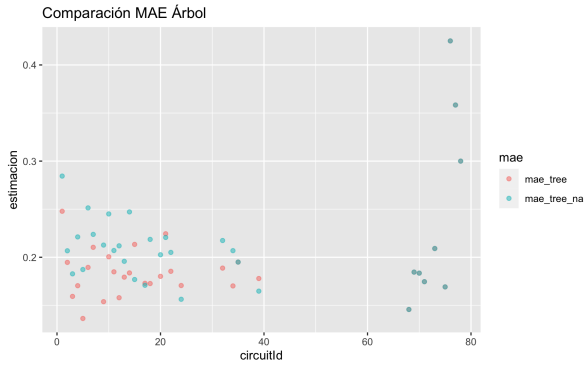
Vamos a comparar ahora los valores de la métrica según la técnica aplicada. Así, en la figura 5.24 podemos observar en rojo los valores para la técnica de imputación y en azul para la técnica de eliminación.

Podemos observar cómo prácticamente ambos valores se asemejan bastante, siendo levemente inferior los valores correspondientes a la técnica de imputación, sobre todo en los tres primeros modelos, es decir salvo en la red neuronal, vemos en general predominar el rojo en zonas más bajas que el azul. Aunque en los modelos lineal y árbol vemos que ocurre como en la prueba 1 que los valores a partir del circuito 68 son iguales para ambas técnicas.

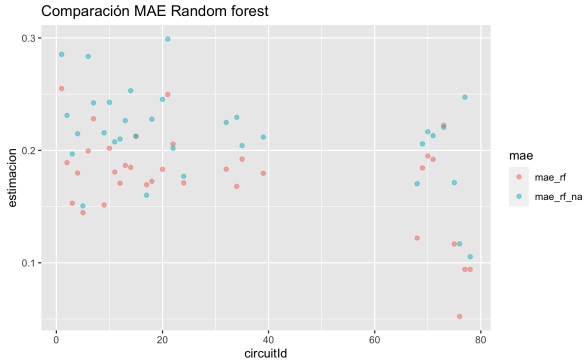
Básicamente hay que destacar que en casi todos los modelos la imputación ha mejorado los resultados.



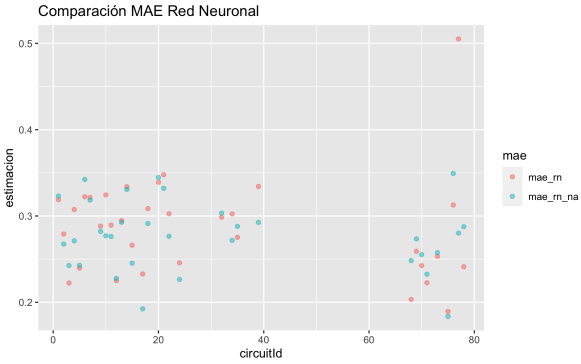
(a) Modelo Lineal.



(b) Árbol de regresión



(c) Random forest.



(d) Red Neuronal.

Figura 5.24: Comparación por ambos métodos.

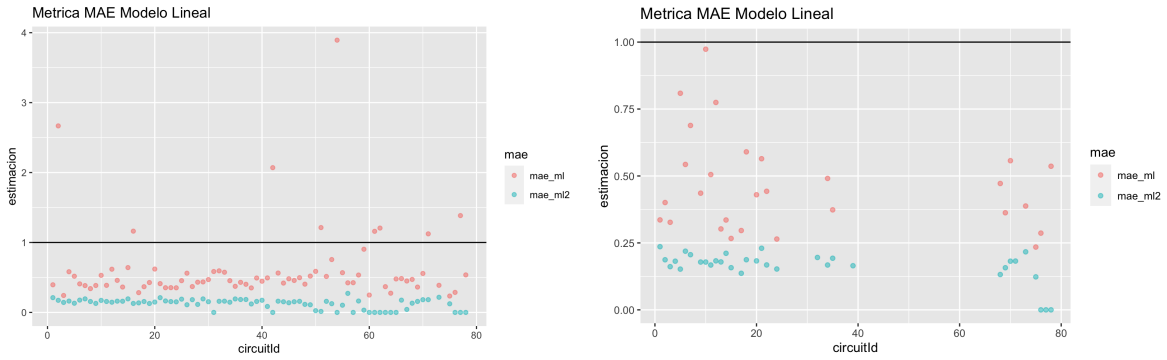
5.4. Comparación.

Vamos a comparar los valores de la métrica obtenidos por las dos pruebas de todos los modelos. En rojo tenemos los valores para la prueba 1 y en azul para la prueba 2.

Modelo lineal.

En primer lugar, vamos a ver los valores según la técnica de imputación en las dos pruebas, figura 5.25a. Podemos ver como, para todos los circuitos, los valores de la segunda prueba son significativamente mejores. Esto nos hace pensar que el modelo para la segunda prueba proporciona unas mejores predicciones.

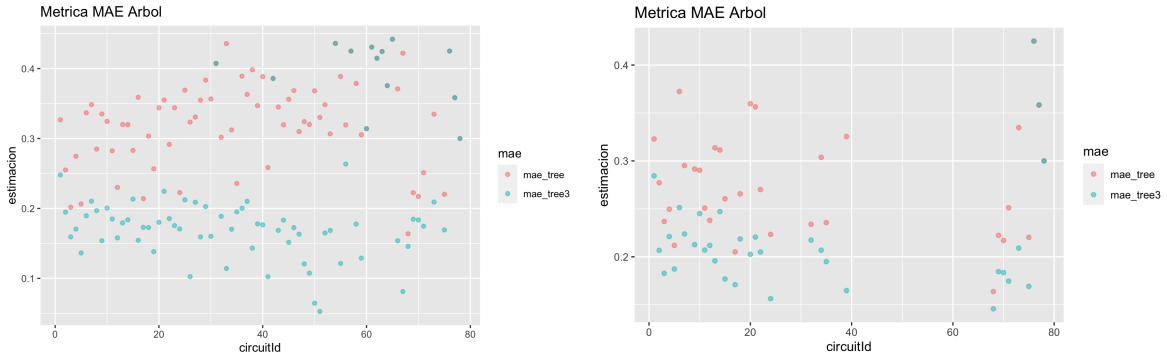
Analizando ahora los valores para la técnica de eliminación,figura 5.25b, en las dos pruebas, obtenemos de nuevo lo mismo.



(a) Mae Modelo lineal imputando (b) Mae Modelo lineal eliminando

Figura 5.25: Mae modelo lineal

Árbol de regresión.



(a) Mae árbol imputando (b) Mae árbol eliminando

Figura 5.26: Mae árbol

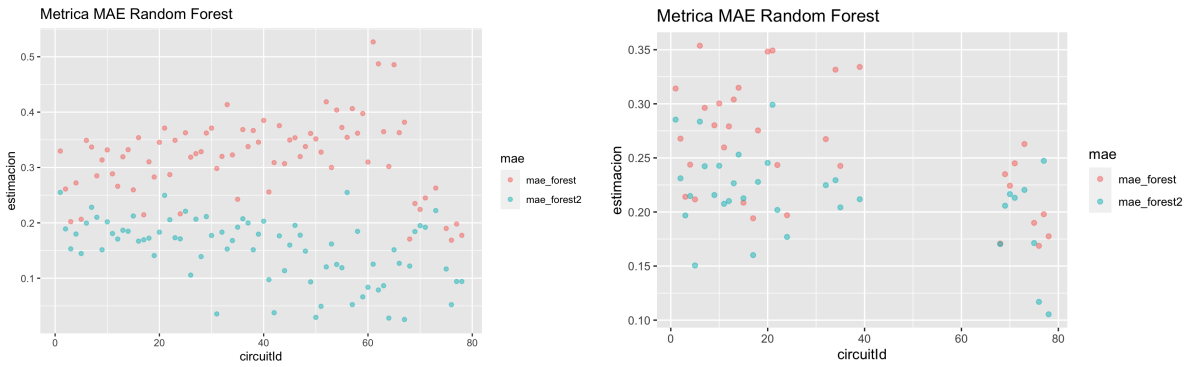
Para este modelo podemos observar una serie de cosas: en primer lugar, ocurre como para el modelo lineal, los valores para la segunda prueba mejoran de una forma elocuente.

Sin embargo, hay que destacar en ambas técnicas que en algunos circuitos toman los mismos valores para las diferentes pruebas.

Random Forest

Nos encontramos para este modelo una situación semejante que en los otros dos, sobre todo similar a la del árbol, dado que también existen circuitos con valores comunes para las dos técnicas.

Para la primera técnica podemos observar en la figura 5.27a cómo se diferencian notoriamente los valores según la prueba.

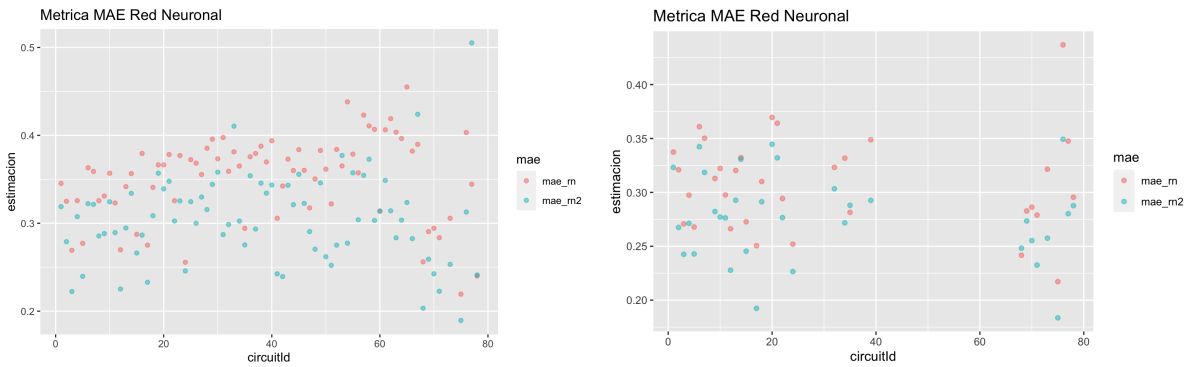


(a) Mae random forest imputando (b) Mae random forest eliminando

Figura 5.27: Mae Random forest.

Red Neuronal

Para este modelo, no podemos afirmar igual que para los tres anteriores dado que existen valores en determinados circuitos los cuales son mejores en la primera prueba. De todas formas, predominan claramente los valores mejores en la segunda prueba, sobre todo en el caso de imputación de datos, aunque haya excepciones.



(a) Mae red neuronal imputando (b) Mae red neuronal eliminando

Figura 5.28: Mae Red neuronal.

Después de estas comparaciones podemos llegar a la conclusión de que buscar variables que puedan sustituir a las variables dummy creadas por las variables tipo factor, provoca una clara mejora en los cuatro modelos implementados. Sin embargo, hay que señalar, como veíamos en la figura 5.21, que los valores obtenidos por la red neuronal son relativamente superiores a los demás modelos por lo que no obtendremos predicciones tan adecuadas.

Como obtenemos mejores resultados con las variables calculadas para la segunda prueba, vamos a realizar una serie de predicciones con los datos obtenidos en las carreras disputadas en la temporada 22, para ver cómo predicen nuestros modelos. Vamos a utilizar los correspondientes a la técnica de imputación de los valores faltantes ya que hemos concluido que por la técnica de eliminación perdemos mucha información.

La información de las carreras de esta temporada la obtenemos en [15] La primera carrera de ésta temporada (round = 1) fue en el circuito de Bahrain (circuitId=3) En esta carrera corrieron un total de 20 pilotos. Veamos cómo predice la posición de varios pilotos, para ello vamos a crear el conjunto de variables objetivo correspondientes a cada constructor. En el conjunto de datos que tenemos la información de cada piloto según el constructor y el circuito, seleccionamos la correspondiente a cada piloto e indicamos la posición de salida que ha tenido este año. Introduciendo éstos datos en cada modelo, obtenemos las predicciones de la **posición relativa**. Además, por la definición (5.1.1), vamos a calcular también la posición absoluta predicha redondeado al entero más cercano. Así, tendremos una idea de la diferencia de puestos entre la posición final real y la predicha.

- Charles Leclerc (driverId= 844), piloto de Ferrari (constructorId=6), salió desde la pole (grid=1) y sabemos que quedó primero.

Predicciones Charles	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.3260169	0.1721823	0.2199694	0.3705522
pos_absoluta	8	4	5	8

Teniendo en cuenta, la menor posición relativa, hemos predicho una posición final que difiere en aproximadamente 3 puestos de la original.

- Lewis Hamilton (driverId = 1), piloto de Mercedes (constructorId = 131), salió desde la quinta posición (grid=5) y sabemos que quedó tercero.

Predicciones Hamilton	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.1239488	0.0455303	0.06829894	0.2853439
pos_absoluta	3	2	2	7

Teniendo en cuenta, la menor posición relativa, que corresponde a la obtenida por el árbol de regresión. Hemos predicho una posición final que difiere en aproximadamente 1 puesto de la original.

- Fernando Alonso (driverId = 4), piloto de Alpine (constructorId = 214), salió desde la octava posición (grid=8) y sabemos que quedó noveno.

Predicciones Alonso	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.8885408	1	0.8070167	0.4558241
pos_absoluta	19	21	17	10

Teniendo en cuenta, la menor posición relativa, que en este caso corresponde a la obtenida por la red neuronal. Hemos predicho una posición final que difiere en aproximadamente 1 puestos de la original.

Observamos que en los tres casos la posición absoluta predicha difiere de su valor real, esto puede ser debido a que en la fórmula 1 es tan importante el coche como el piloto, y dados los cambios de normativa tan importantes que ha habido esta temporada resulta lógico que presenciemos estas diferencias al menos en los inicios de la temporada ya que aún no tenemos datos. De todas formas, podemos destacar que no es nada mala esa predicción que nos ha dado para los tres.

Vamos por tanto, a intentar predecir los resultados de otra carrera de esta temporada de la cuál aún no tenemos información en nuestra base de datos [10], introduciendo en las variables objetivo la información de las carreras de la que tenemos información de esta temporada. En nuestro conjunto de datos hasta la temporada 21 tenemos información hasta la última carrera de esa temporada que es la carrera 1073. De las carreras disputadas hasta la fecha de la temporada 22, tenemos información sobre las 3 primeras, por lo que con esta información vamos a predecir los resultados en la 4º carrera de la temporada, la cual corresponde por [15] al Gran Premio de la Emilia Romagna disputada en el circuito Autódromo Enzo e Dino Ferrari (circuitId=21).

Vamos a predecir la posición en la que quedarán los que han salido desde los 5 primeros puestos de salida.

- Max Verstappen (driverId = 830), piloto de Redbull (constructorId= 9), salió desde la primera posición (grid=1) y sabemos que quedó primero.

Predicciones	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.4915007	0.2484227	0.4114014	0.5812784
pos_absoluta	11	6	9	13

Hay que tener en cuenta que en dos de las tres carreras disputadas ésta temporada Verstappen no consiguió acabarlas. Esto nos hace pensar por qué puede diferir tanto la posición final real de la que estamos obteniendo en los modelos.

- Charles Leclerc (driverId = 844), piloto de Ferrari (constructorId= 6), salió desde la segunda posición (grid=2) y sabemos que quedó sexto.

Predicciones	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.1603274	0.03375097	0.2048664	0.5571435
pos_absoluta	4	2	5	12

Resulta lógico el resultado que hemos obtenido por el modelo de árbol de regresión, el cual es la menor posición relativa. Teniendo en cuenta los valores de las carreras anteriores dado que Leclerc ganó dos de las tres anteriores de ésta temporada y quedó segundo en la tercera. Sin embargo, al tener un problema inesperado durante la carrera le hizo perder posiciones, factor que no podemos modelizar.

- Sergio Pérez (driverId = 815), piloto de Redbull (constructorId= 9), salió desde la tercera posición (grid=3) y sabemos que quedó segundo.

Predicciones	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.5334478	0.3683414	0.5123451	0.6867242
pos_absoluta	12	8	11	15

Teniendo en cuenta el modelo que aporta la mejor posición relativa, hemos predicho que queda 6 posiciones más atrás de las que realmente termina.

- Carlos Sáinz (driverId = 832), piloto de Ferrari (constructorId= 6), salió desde la cuarta posición (grid=4) y sabemos que no terminó la carrera.

Predicciones	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.1997691	0.1965222	0.2014624	0.6082641
pos_absoluta	5	5	5	13

Hay que destacar que es prácticamente imposible predecir que alguien no va a terminar una carrera, ya que para ello intervienen factores muy imprevisibles.

- Lando Norris (driverId =846), piloto de Mclaren (constructorId= 1), salió desde la quinta posición (grid=5) y sabemos que quedó tercero.

Predicciones	Imputando			
	Modelo Lineal	Árbol de regresión	Random forest	Red Neuronal
pos_relativa	0.2285371	0.1719706	0.1685434	0.6355996
pos_absoluta	6	4	4	14

En función de la mejor posición relativa obtenida, estamos fallando en una posición. Luego, de los cinco pilotos que hemos predicho, éste es el que más nos hemos acercado.

Parte IV

Conclusiones.

Capítulo 6

Conclusiones.

Haciendo una breve recopilación de todo lo que se ha hecho a lo largo de este trabajo, destacando que durante del proyecto nos hemos enfrentado a un caso real de calado, se ha buscado una gran cantidad de conjuntos de datos relacionados con la temática, se ha profundizado mucho en labores de limpieza y tratamiento de datos, así como análisis exploratorio de los datos, ingeniería de características y modelización con distintas técnicas para modelos de regresión. Destacar además el empleo de tidyverse y tidymodels, incluyendo el manejo de funciones avanzadas de orden superior con purrr para generar en bloque todos los modelos de cada circuito, y la gran cantidad de agregaciones de datos obtenidas a muchos niveles.

Para ambos modelos, sobre todo los de la segunda prueba, hemos calculado muchas variables que acumulan resultados promedio de los pilotos en sus carreras anteriores, generalmente con la misma escudería, en el mismo circuito, ... No obstante, sería interesante tener además alguna variable que indique si el piloto en una escudería, en un circuito está mejorando, empeorando o está estancado al cabo de los años y así aunque algún piloto que no lleve un buen histórico, que es lo que principalmente usamos en el modelo, con esta nueva variable estaríamos capturando su tendencia.

Por otro lado, al igual que le hemos hecho a la variable respuesta la transformación para trabajar con la posición relativa, reflexionando sobre ello, no es lo mismo salir el 20 de 22 que el 20 de 50. De modo que, parece razonable aplicar esta misma transformación a la variable posición de salida (grid).

Ante todo, cabe notar algo que podíamos intuir, y que lo tenemos aún más presente tras el desarrollo de este trabajo, es que resulta complicado predecir con exactitud las posiciones en las carreras de fórmula 1, debido a una gran variedad de factores externos los cuales son impredecibles ya sea por la fiabilidad de los coches, los accidentes y colisiones, o simplemente por el estado del tiempo. El factor que más puede intervenir en ésto es la fiabilidad del coche, ya que es imposible saber cúando va a fallar un coche y menos aún por qué, porque además es algo muy variable de un año para otro, con cambios de normativas y constantes adaptaciones por parte de los equipos de ingenieros de las escuderías.

Un fallo inesperado en el coche de algún piloto que iba a quedar en alguna posición lógica para su rendimiento, hace que otros pilotos que podría predecir bien nuestros modelos, queden en posiciones en las que inicialmente no iban a estar.

Así mismo, existen otros factores de los cuales no tenemos información en nuestra base de datos, que pueden influir en la posición final y que son sustanciales en las estrategias de los equipos.

Por ejemplo, factor que influye mucho en la posición final y que profundizan mucho en los equipos en cada carrera, es el número de paradas que deben hacer y sobre todo en qué momento de la carrera hacerlas, dado que ésto influye bastante en la posición en la que puedes quedar. Pero para ello, necesitaríamos más información de lo que ocurre en las carreras, como el tipo de neumático con el que salen, la degradación de la pista, si ha habido un coche de seguridad o no, si empieza a llover, etc.

Éstos datos no los tenemos por lo que para un trabajo futuro estaría perfecto poder encontrar alguna base de datos con este tipo de información para estudiar a fondo este factor tan interesante. No obstante, a partir de la información de la que disponemos hemos podido realizar un gráfico aproximado, figura 6.1, del intervalo de vueltas en las que se puede realizar cada parada en una serie de circuitos.

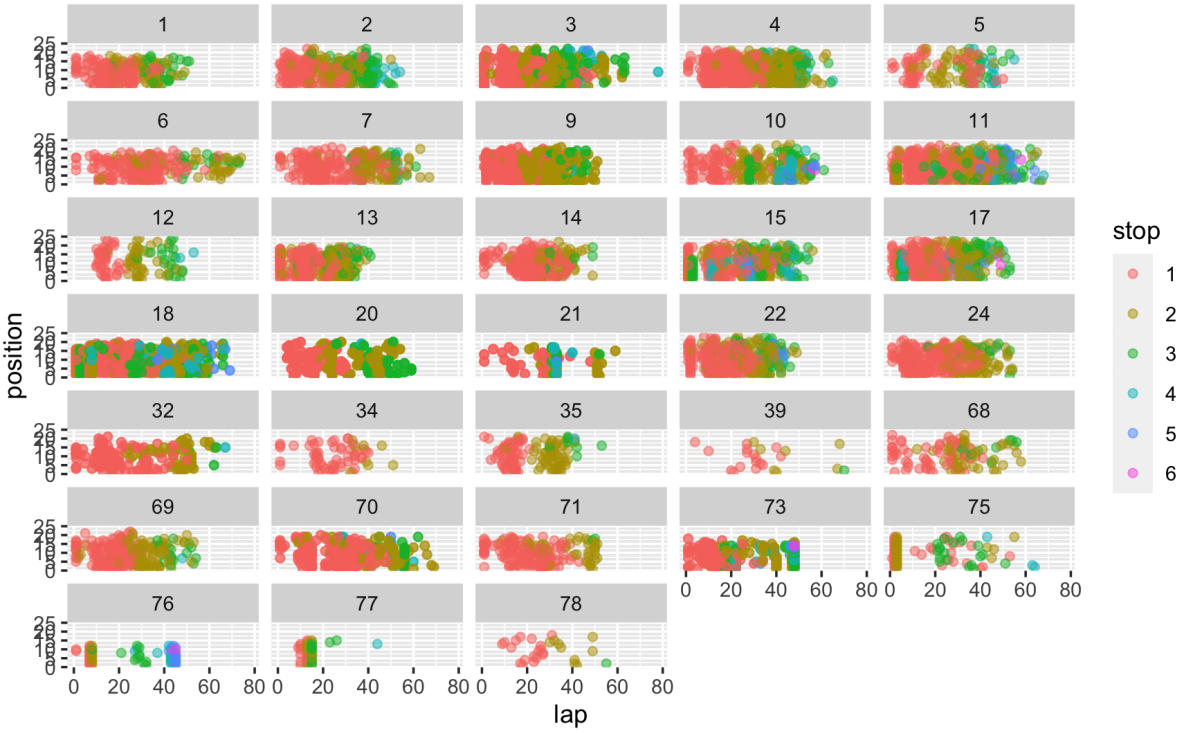


Figura 6.1: Número de parada según la vuelta.

De éstos gráficos podemos destacar que en la mayoría de los circuitos la primera parada se realiza entre las 25 primeras vueltas del circuito. Ahora bien, como hemos mencionado, esta información no es concluyente, ya que no conocemos el tipo de neumático con que se corría, ni las condiciones climáticas.

También podemos notar cómo los que terminan en las primeras posiciones hacen menos paradas que el resto de pilotos.

Otro punto importante a destacar es la predicción de pilotos que no van a terminar la carrera, objetivo que parece muy complicado porque resulta inviable saber que le va a pasar al coche, o si en cualquier choque va a provocarle algo que le impida seguir.

Está claro, como podemos ver en la figura 6.2, que la mayoría de accidentes, en concreto las colisiones hacen que un piloto no pueda terminar la carrera, en esta figura estamos comparando la posición de salida con la posición relativa según un piloto ha tenido accidente o una colisión. No obstante, ¿es suficiente con eso?

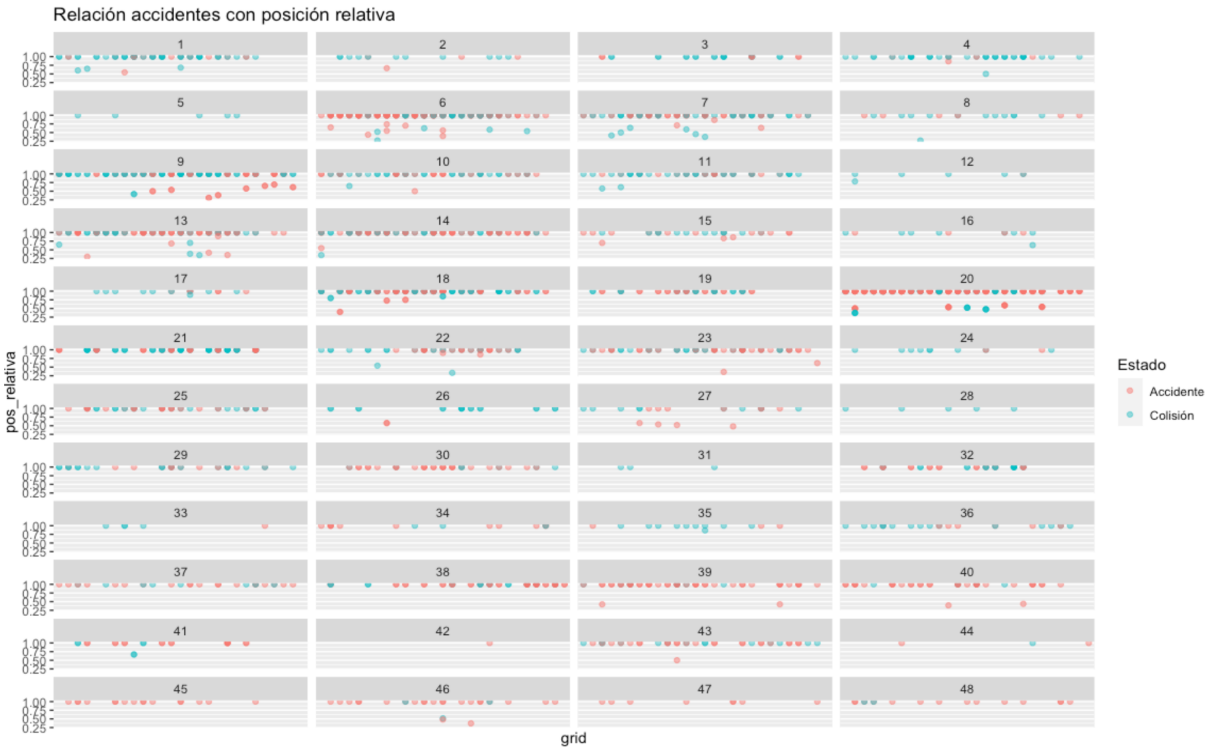


Figura 6.2: Importancia colisiones

Puede que no, ya que existen una cantidad considerable de acontecimientos que pueden convertirse en fatales para que un piloto no pueda acabar una carrera. Luego, una solución que hemos pensado para un trabajo futuro, es poder realizar un modelo de clasificación (3.1.2) con variables relacionadas con posibles factores que influyan en acabar o no la carrera, de forma que la variable respuesta será acaba o no.

Con respecto a los modelos implementados, todo su desarrollo se fue haciendo con tidymodels [11], con gran carga de procesamiento bien estructurado para usar ese ecosistema, y se aprovechó la estructura para incorporar también el modelo de red neuronal siguiendo el mismo esquema. Si nos centramos en la red neuronal introducida tenemos que tidymodels solo nos da opción a utilizar una red neuronal con una sola capa oculta, lo cuál nos restringe considerablemente los resultados obtenidos. Por tanto, un trabajo futuro será, cuando las opciones para la red neuronal estén más desarrolladas para tidymodels, para poder hacer más pruebas con este tipo de modelos. Otra posibilidad sería emplear Keras, bien en Python o desde el propio R y así poder implementar una red neuronal más compleja para poder contrastar los resultados

Profundizando también en la idea de la creación de modelos en los cuales tomen parte pilotos que no han participado en ninguna carrera de fórmula 1, la perspectiva que le podemos dar es utilizar la información que podamos recopilar a través de bases de datos

de las categorías inferiores para conocer mejor sus cualidades y poder aprovecharlas en los modelos.

Por otro lado, con respecto a los modelos que contienen circuitos para mejorar las predicciones de éstos, podríamos crear modelos de aprendizaje no supervisado (3.2) para, a partir de las características de los circuitos: número de curvas, tipo de curvas, distancia total, tipos de neumáticos más adecuados al circuito, etc. Poder crear diferentes clusters para diferenciar los circuitos en determinados tipos, los cuales posean características similares como consecuencia en cada tipo diferente de circuitos podría tener el coche características diferentes para adecuarse mejor a él. De esta manera, también podremos realizar un estudio de los circuitos en los que son mejores cada piloto o cada escudería, y pudiendo crear posteriormente nuevas relaciones entre los pilotos los circuitos que tienen en común.

Capítulo 7

Bibliografía

[1] Zhi-Hua Zhou, *Machine Learning.*

[2] Max Kuhn ,Kjell Johnson, *Applied Predictive Modeling.*

[3] Shan Suthaharan, *Machine Learning Models and Algorithms for Big Data Classification.*

[4] Tom M. Mitchell, *Machine Learning.*

[5] ethem alpaydin, *Introduction to Machine Learning.*

[6] Du, K.L. , Swamy, M.N.S. , *Neural Networks and Statistical Learning.*

[7] http://www.cs.us.es/cursos/siis-2018/temas/tema-07-siis-2018-19.pdf

[8] https://www.iartificial.net

[9] https://docplayer.es/210640950-Modelos-predictivos-con-aplicacion-en-el.html

[10] https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020?select=lap_times.csv

[11] https://www.r-project.org

[12] https://www.tidymodels.org

[13] https://ggplot2.tidyverse.org/index.html

[14] https://rpubs.com

[15] https://r4ds.had.co.nz/data-visualisation.html

[16] https://fiaresultsandstatistics.motorsportstats.com

Parte V

Anexos.

Anexo A

Primera prueba.

En este anexo vamos a realizar en primer lugar la lectura y el preprocesamiento de los datos. A continuación, crearemos las variables que vamos a utilizar en la primera prueba, definiendo consigo los conjunto de datos para la modelización. Por último, creamos los modelos con las variables definidas anteriormente con las técnicas que ya hemos mencionado, realizando a su vez comparaciones en cada modelo según la técnica.

Primera prueba.

Lectura de datos.

Vamos a leer en primer lugar los conjuntos de datos necesarios para llevar a cabo este trabajo.

```
resultados=read.csv("datosk/results.csv",header = TRUE,dec = ".")
resultados$position=parse_integer(resultados$position )
carreras=read.csv("datosk/races.csv",header = TRUE)
pilotos=read.csv("datosk/drivers.csv",header = TRUE)
constructores=read.csv("datosk/constructors.csv",header = TRUE)
clasificacion=read.csv("datosk/qualifying.csv",header = TRUE,dec = ".")
boxes=read.csv("datosk/pit_stops.csv",header = TRUE,dec = ".")
resul_construc=read.csv("datosk/constructor_results.csv",header = TRUE)
circuitos=read.csv("datosk/circuits.csv",header = TRUE)
estados=read.csv("datosk/status.csv",header = TRUE)
tiempos_vuelta=read.csv("datosk/lap_times.csv",header = TRUE,dec=".")
```

```
carreras %>% select(raceId,year,circuitId) %>% summary
```

En primer lugar, observamos que en el conjunto de datos de las carreras, tenemos la información de las carreras que se disputan esta temporada (2022), sin embargo como ésta aún no ha terminado es evidente que no tenemos los resultados. Por tanto, en nuestro conjunto de datos vamos a utilizar la información de las carreras hasta la temporada 2021.

Entonces, seleccionamos las carreras que no pertenecen a la temporada 2022 ya que aún no ha terminado y no tenemos resultados de ésta temporada. Este año se disputan un total de 23 carreras.

```
carreras_22 =carreras %>% filter(year==2022)
c22= carreras_22 %>% select(circuitId)
```

```
carreras_not22= carreras %>% filter(year < 2022)
carreras_not22 %>% select(raceId,year,circuitId) %>% summary
```

Vamos a estudiar el conjunto de resultados:

```
rc= left_join(resultados,carreras_not22,by="raceId") %>% select(-url)
rc %>% select(grid,position,laps) %>% summary()
```

Ya que se observa que en algunas carreras hay datos alarmantes para pensar que algunas no corresponden a carreras de fórmula 1.

```
dat_in=rc %>%  filter(name == "Indianapolis 500") %>%  arrange(year)
dat_in %>% select(grid,laps) %>% summary
ndat_in =  dat_in  %>% group_by(raceId,year) %>%
  count() %>%
  arrange(desc(n)) # Numero de pilotos que han corrido cada año en las 500 millas

ndat_in %>% summary
```

Observamos datos de carreras de hasta 200 vueltas y vemos que corresponde a las 500 millas de indianápolis, carrera que no pertenece al campeonato de fórmula 1, aunque, en los primeros años de la fórmula 1, si que

perteneció al campeonato. Sin embargo, no resulta de interés introducirlo junto a nuestro conjunto de datos ya que difiere mucho de las carreras de fórmula 1.

En 11 temporadas , desde 1950 (inicio de la fórmula 1) hasta 1960 se consideró las 500 millas de Indianápolis como una carrera perteneciente a la competición de fórmula 1, corriendo en ella hasta 55 pilotos.

Por lo tanto, vamos a considerar nuestro conjunto de datos, sin la información correspondiente a las 500 millas de indianapolis.

```
g0= rc %>%  filter(grid==0)
g0 %>% select(grid,position) %>% summary
```

Además también observamos que hay carreras en las que no hay información de las posiciones de salida (grid) que corresponden a las que tampoco tienen información en la posición final,es decir, son datos perdidos.

Además de esos, observamos que la variable posición tiene muchos valores NA, los cuáles corresponderán a los pilotos que no han terminado la carrera.

Conjunto datos sin 500I y grid=0.

```
datos_m =  rc %>%  filter(name !="Indianapolis 500" & grid>0)
```

```
datos_m %>% select(grid,position,laps) %>% summary()
```

Numero pilotos por carrera

```
npilo_xgp=datos_m %>% group_by(raceId,year) %>%  count() %>% arrange(desc(n))
```

Si observamos ahora el número de pilotos que compitieron en cada temporada, resulta de interés comentar que hubo temporadas en las que el número de pilotos era muy superior al queestamos acostumbrados a presenciar actualmente ya que no es muy usual que corran más de 24 pilotos.

Vamos a ver el número de pilotos en cada carrera según la temporada.

```
npilo_xgpY= npilo_xgp %>% group_by(year) %>% nest()

d= function(x){
  x %>%  mutate(n_carrera=1:length(raceId))
}

npilo = npilo_xgpY %>% mutate(nc=map(data,d)) %>% unnest(nc) %>% select(-c(data,raceId))

npilo_1 =npilo %>%  filter(year <= 1972)
npilo_2= npilo %>% filter(1973<=year& year <=1995)
npilo_3= npilo %>% filter(year >=1996)

npilo_1 %>%  ggplot(aes(n_carrera,n))+
  geom_point(alpha=1/2, color = '#6495ED')+
  facet_wrap(~year,ncol=5)+
  xlab("Número de carrera")+
  ylab("Número de pilotos")+
  ggtitle("Número de pilotos en cada carrera")



npilo_2 %>%  ggplot(aes(n_carrera,n))+
  geom_point(alpha=1/2, color = '#6495ED')+
```

```
  facet_wrap(~year,ncol=5)+
  xlab("Número de carrera")+
  ylab("Número de pilotos")+
  ggtitle("Número de pilotos en cada carrera")

npilo_3 %>%  ggplot(aes(n_carrera,n))+
  geom_point(alpha=1/2, color = '#6495ED')+
  facet_wrap(~year,ncol=5)+
  xlab("Número de carrera")+
  ylab("Número de pilotos")+
  ggtitle("Número de pilotos en cada carrera")
```

Observamos que en las primeras temporadas, el número de pilotos que disputa cada carrera varía bastante. Y también, el número de carreras en cada temporada es mucho menor que el número de carreras que se disputan actualmente en cada temporada. También podemos ver cómo el número de pilotos parece estabilizarse entre el 1996 y 1997 .

Para no perder información de todas las carreras en las que el número de pilotos no está estabilizado, vamos a trabajar en lugar de la posición absoluta, con la posición relativa, definiendo ésta como : posición relativa = (nº de pilotos por delante)/(nº de pilotos totales de partida)

De ésta manera la posición relativa varía en el intervalo [0,1). Así, podemos imputar los valores (NA) de los pilotos que no han acabado la carrera por 1.

Posición relativa.

Vamos a definir la posición relativa la cual será nuestra variable respuesta.

```
datos_m =  left_join(datos_m,npilo_xgp,by=c("raceId","year")) %>%
 mutate(pilo_delante=position-1)
datos_m [ nrow (datos_m) + 1,] = datos_m %>%
  filter(raceId==1063 & driverId== 840) %>% mutate(pilo_delante = 17)

datos_m= datos_m %>%  mutate(pos_relativa=pilo_delante/n) %>%
  mutate_at("pos_relativa",~replace(., is.na(.), 1))
datos_m = datos_m %>%  filter(pos_relativa <= 1)
```

```
datos_m %>% select(raceId,grid,pos_relativa) %>% summary
```

Circuito - GP

Veamos en que circuito se corre cada Gran Premio.

```
circuit =circuitos %>% select(circuitId,name)
GP=right_join(circuit,carreras,by="circuitId")
GP=GP %>%  select(name.y,circuitId,name.x) %>% distinct(name.y,circuitId,name.x) %>%
  rename("Gran_Premio"=name.y,"Circuito"=name.x)
GP2=carreras %>% select(raceId,circuitId) # ID carreras y circuitos
```

Variables

Número de veces que se ha corrido en cada circuito.

Usamos los datos en los que no tenemos las carreras de 22.

```
nv_circuitos=carreras_not22 %>%
                     group_by(circuitId) %>%
                      summarise(totalv=n())
```

Número de veces que cada piloto ha corrido en cada circuito.

Según el constructor

```
nv_pilo_cc=datos_m %>%  group_by(driverId,circuitId,constructorId) %>%
                          summarise(totalc=n_distinct(raceId))
```

Independientemente del constructor

Numero total de veces que un piloto ha corrido en un circuito.

```
nv_pilo_c= nv_pilo_cc %>% group_by(driverId,circuitId) %>%
  summarise(tveces=sum(totalc))
```

Media paradas

Como la estrategia es generalmente una decisión de equipo: Voy a hacer la media de paradas que cada equipo ha hecho en cada circuito.

Obtenemos las paradas a las que va cada constructor en cada circuito.

En primer lugar, vamos a calcular el número de paradas que cada piloto ha hecho en cada carrera:

```
nboxes =inner_join(datos_m,boxes,by=c("raceId","driverId")) %>%
  group_by(raceId,driverId,constructorId) %>%
  summarise(nstop=n())
nboxes = left_join(nboxes,GP2,by=c("raceId"))
```

Ahora vamos a calcular la media de paradas que cada equipo hace en cada circuito, aunque en la realizar el número de paradas es un número entero, en este caso no vamos a redondear la media obtenida para tener una aproximación más exacta del número de paradas que pueden realizar cada equipo en cada circuito.

```
n_stop = nboxes %>% group_by(circuitId,constructorId) %>%
  summarise(stop_mean= mean(nstop))
```

Media tiempo en paradas

Voy a hacer la media de tiempo (en milisegundos) que cada equipo ha tardado en parada en cada GP.

En primer lugar, vamos a calcular el tiempo medio empleado por cada equipo con cada piloto en una parada en cada carrera

```
time_boxes =inner_join(datos_m,boxes,by=c("raceId","driverId"))
time_boxes = time_boxes %>% group_by(raceId,driverId,constructorId) %>%
  summarise(suma_t=sum(milliseconds.y))
t_stop=inner_join(time_boxes,nboxes,by=c("raceId","driverId","constructorId"))
t_stop=t_stop %>% mutate(stop_tm=suma_t/nstop) %>%
  select(circuitId,driverId,constructorId,stop_tm)
t_stop=t_stop[,-1]
```

Ahora calculo, el tiempo medio por parada en cada circuito de cada equipo.

```
t_stop = t_stop %>% group_by(constructorId,circuitId) %>%
  summarise(timestop_mean=mean(stop_tm))
```

Proporción de accidentes por pilotos y circuitos

Filtro por los estados de accidente (statusId=3) y cuento cuántos ha tenido cada piloto en cada circuito.

```
acc=datos_m %>% filter(statusId==3 )
acc=acc %>% group_by(driverId,circuitId) %>% summarise(totalac=n())
```

Hago la proporción del número de accidentes de cada piloto en cada circuito entre el número total de veces que cada piloto ha corrido en cada circuito.

```
pacc=left_join(acc,nv_pilo_c,by=c("driverId","circuitId"))%>%
  mutate(pAccidentes=totalac/tveces)
pacc=pacc%>% select(driverId,circuitId,pAccidentes)
```

Tiempo medio piloto

Dado que el coche depende del constructor con el que esté el piloto, calculo: Tiempo medio que cada piloto ha llevado en cada carrera dependiendo del constructor.

Sin embargo, si un piloto cambia de escudería o incluso llega nuevo a la fórmula 1, no vamos a tener datos correspondientes a este por ello voy a calcular: El histórico del piloto, el histórico del piloto en cada circuito : estas variables sobre todo son calculadas para el caso en el que un piloto cambie de escudería. Por último calculo el histórico de piloto, circuito y constructor.

Tiempo medio piloto

Vamos a calcular el histórico del piloto: tiempo medio en vuelta en general del piloto.

```
tiempo_pilo = tiempos_vuelta %>% group_by(driverId) %>%
  summarise(tm_p=mean(milliseconds))
```

Vamos a calcular la media general de todos los pilotos, para poder imputar los valores de los pilotos de los que no tengamos información cuando unamos todos los datos.

```
tm_pgeneral =  tiempo_pilo %>% summarise(tm_gen=mean(tm_p))
```

Tiempo medio piloto circuito

En primer lugar, voy a calcular, el tiempo medio por vuelta de cada piloto en cada carrera.

```
tiempo_vmean= tiempos_vuelta %>% group_by(raceId,driverId) %>%
  summarise(t_vm=mean(milliseconds))
tiempo_vmean =left_join(tiempo_vmean,GP2,by="raceId")
```

Vamos a calcular el tiempo medio por vuelta que lleva cada piloto en cada circuito (sin tener en cuenta el constructor)

```
tiempo_pilocir=tiempo_vmean %>% group_by(driverId,circuitId) %>%
  summarise(tm_pci=mean(t_vm))
```

Vamos a calcular la media de las medias.

```
tm_pcgeneral = tiempo_pilocir %>% group_by(circuitId) %>% summarise(tm_pcgen= mean(tm_pci))
tm_pcgeneral = tm_pcgeneral  %>% summarise(tm_pcgen= mean(tm_pcgen))
```

Tiempo medio piloto circuito constructor

Calculamos por último, el tiempo medio por vuelta que lleva cada piloto en cada circuito según el constructor.

```
cirpiloeq= datos_m %>% select(raceId,driverId,constructorId)
tiempo_pilocc= left_join(tiempos_vuelta,cirpiloeq,by=c("raceId","driverId")) %>%
  group_by(raceId,driverId,constructorId) %>%
  summarise(tm_pcc=mean(milliseconds))

tiempo_pilocc= left_join(tiempo_pilocc,GP2,by="raceId") %>%
  group_by(circuitId,driverId,constructorId) %>%
  summarise(tmean_pcc=mean(tm_pcc))
```

```
tm_pccgeneral = tiempo_pilocc %>% group_by(circuitId,constructorId) %>% summarise(tm_pccgen= mean(tmean
tm_pccgeneral = tm_pccgeneral  %>% summarise(tm_pccgen= mean(tm_pccgen))
tm_pccgeneral =tm_pccgeneral %>% summarise(tm_pccgen= mean(tm_pccgen))
```

Conjunto datos

Vamos a tener como variables: - DriverId - ConstructorId - Posición de salida - Posición relativa - Ronda en la temporada que se corrio en ese circuito. - Numero medio de paradas de cada constructor en cada circuito. - Tiempo medio en paradas de cada constructor en cada circuito. - Probabilidad de accidende de cada piloto en cada circuito. - Tiempo medio piloto en vuelta. - Tiempo medio piloto en vuelta en cada circuito. - Tiempo medio piloto en vuelta en cada circuito según el constructor.

```
conjunto = datos_m %>%  select(circuitId,driverId,constructorId,grid,pos_relativa,round)

conjunto = left_join(conjunto,n_stop,by=c("circuitId","constructorId"))

conjunto= left_join(conjunto,t_stop,by=c("circuitId","constructorId"))
conjunto %>% select(stop_mean,timestop_mean) %>% summary
```

Obtenemos NA porque la información de las paradas en boxes no empezó a recogerse hasta el año 2011, luego, para algunos constructores en determinados circuitos no tendremos información. Al hacer el modelo, vamos a imputar dichos datos.

```
conjunto = left_join(conjunto,pacc,by=c("circuitId","driverId"))
conjunto %>%  select(pAccidentes) %>% summary
```

Obtenemos valores NA que son los correspondientes a los pilotos que no han tenido accidentes en esos circuitos por lo que los imputamos por 0.

```
conjunto = conjunto %>% mutate_at("pAccidentes",~replace(.,is.na(.),0))
```

```
conjunto =left_join(conjunto,tiempo_pilo,by="driverId")
conjunto = left_join(conjunto,tiempo_pilocir,by=c("circuitId","driverId"))
conjunto = left_join(conjunto,tiempo_pilocc,by=c("circuitId","constructorId","driverId"))
conjunto %>%  select(tm_p,tm_pci,tmean_pcc) %>% summary
```

Obtenemos NA porque no tenemos la información correspondiente al tiempo en vuelta en determinados circuitos. Vamos a imputarle por la media de las medias calculadas anteriormente.

```
conjunto = conjunto %>%
  mutate_at("tm_p",~replace(.,is.na(.),tm_pgeneral$tm_gen))
conjunto = conjunto %>%
  mutate_at("tm_pci",~replace(.,is.na(.),tm_pcgeneral$tm_pcgen))
conjunto = conjunto %>%
  mutate_at("tmean_pcc",~replace(.,is.na(.),tm_pccgeneral$tm_pccgen))
```

```
conjunto %>%  select(tm_p,tm_pci,tmean_pcc) %>% summary
```

Ahora no obtenemos NA

Vamos a tratar las variables driverId y constructorId como factores.

```
conjunto$driverId= as.factor(conjunto$driverId)
conjunto$constructorId=as.factor(conjunto$constructorId)
```

Vamos a definir el conjunto de datos sin valores NA, para poder comparar.

```
conjunto2 = conjunto %>% na.omit()
```

Vamos a calcular, la media de las medias de las paradas y del tiempo en paradas, para poder imputar los valores de los circuitos de los que no tenemos información sobre estas variables.

```
med_cir=conjunto2 %>%
  group_by(circuitId) %>%
  summarise(
  med_p=mean(stop_mean),
  med_tp=mean(timestop_mean))
medTotal= med_cir %>% summarise(med_p=mean(med_p),med_tp=mean(med_tp))
```

```
n_circuitos= conjunto%>%
   group_by(circuitId) %>%
  nest()

## Comprobamos que toda la columna que queriamos imputar tiene NA

ncir2= n_circuitos %>% mutate(
  tieneNA= data %>%  map(~pull(.x,stop_mean)  %>% is.na(.) %>% all()))

n_circuitos =ncir2 %>%
  unnest() %>%
  mutate(
    stop_mean = ifelse(tieneNA==TRUE,medTotal$med_p,stop_mean),
    timestop_mean = ifelse(tieneNA==TRUE,medTotal$med_tp,timestop_mean)
  ) %>%
  select(-c(tieneNA)) %>%
   group_by(circuitId) %>%
  nest()
```

Modelo lineal

IMPUTACION KNN

Defino la función que realiza un modelo lineal de los datos.

```
lmc_fun <- function (x) {
x= x %>%
    mutate(
      driverId=factor(as.numeric(driverId)),
      constructorId=factor(as.numeric(constructorId))
      )
 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
```

```
nc_split<- initial_split(as.data.frame(x),prop = 0.70,
                         strata = pos_relativa,breaks = 4)
entnc <- nc_split%>% training()
testnc <- nc_split %>% testing()

#2º definimos el modelo
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")


# 3º Creamos la receta
nc_recipe <-  recipe(pos_relativa ~.,data=entnc) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 4º Creamos el workflow

nc_workflow <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(nc_recipe)
# 5º Estimamos el modelo.
nclm_fit <- nc_workflow %>%
             last_fit(split = nc_split, metrics=metric_set(mae,rmse,rsq))


t = tibble(split = list(nc_split),model = list(nclm_fit), metrics= list(nclm_fit %>% collect_metrics())
t
}
```

Aplico el modelo lineal a los datos de cada circuito.

```
m_nc <- n_circuitos %>%
 mutate(res=map(data,lmc_fun))

nc2=m_nc %>% unnest(res)

mae_mlc=nc2 %>% unnest(metrics) %>% filter(.metric == "mae") %>% arrange(.estimate) %>% select(circuitI
mae_mlc
```

circuitos como variable

¿Qué ocurre si nos encontramos con un circuito completamente nuevo ? al no tener datos de temporadas anteriores, por el método anterior sería imposible hacer alguna predicción, por lo que vamos a calcular un nuevo modelo en el que le introducimos como variable el circuito en el que se corre. Esta variable la vamos a tratar como factor.

```
conjunto_c = n_circuitos %>% unnest()
conjunto_c=conjunto_c %>% mutate(circuitId=as.factor(circuitId))
```

```
#1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
c_split<- initial_split(conjunto_c,prop = 0.70,strata = pos_relativa,breaks=4)

c_ent <- c_split%>% training()
c_test<-c_split %>% testing()


# 2º Definimos el modelo.

lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# 3º Creamos la receta.
c_recipe <-
  recipe(pos_relativa ~.,data=c_ent) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId,driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 4º Creamos un WorkFlow
c_workflow <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(c_recipe)


#5º Estimamos el modelo.

c_fit <- c_workflow %>%
             last_fit(split = c_split, metrics=metric_set(mae,rmse,rsq))

met_mlcir=c_fit %>% collect_metrics()
met_mlcir
```

ELIMINANDO NA'S

Vamos a eliminar todas las observaciones que contienen alguna variable con NA

```
n_circuitos_na= conjunto2%>%
   group_by(circuitId) %>%
  nest()
```

```
lmc_fun_na <- function (x) {
  x = x %>%
```

```
    mutate(
      driverId=factor(as.numeric(driverId)),
           constructorId=factor(as.numeric(constructorId))
      )
 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
nc_split<- initial_split(as.data.frame(x),prop = 0.70,strata = pos_relativa,breaks = 4)
entnc <- nc_split%>% training()
testnc <- nc_split %>% testing()

#2º definimos el modelo
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

#3º Creamos la receta.
nc_recipe<-  recipe(pos_relativa ~.,data=entnc) %>%
  step_novel(driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())



#4º Creamos el Workflow.
nc_workflow <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(nc_recipe)

#5º Estimamos el modelo.
nclm_fit <- nc_workflow %>%
             last_fit(split = nc_split, metrics=metric_set(mae,rmse,rsq))



t = tibble(split = list(nc_split),model = list(nclm_fit),
           metrics= list(nclm_fit %>% collect_metrics()),
           predictions = list(nclm_fit %>% collect_predictions()))
t
}
```

```
m_nc_na<- n_circuitos_na %>%
 mutate(res=map(data,lmc_fun_na))

nc2_na=m_nc_na %>% unnest(res)

mae_ml_na=nc2_na %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>% select(circuitId,.estimate)
mae_ml_na
```

circuitos como variable

1º Creamos una partición entre conjunto test y entrenamiento

```
conjunto2_c=conjunto2 %>%  mutate(circuitId=as.factor(circuitId))
```

```
#1º  Creamos una división entre entrenamiento y test.
set.seed(1234)
nc_split_na<- initial_split(conjunto2_c,prop = 0.70,
                            strata = pos_relativa,breaks=4)

nc_ent_na <- nc_split_na%>% training()
nc_test_na<- nc_split_na%>% testing()

# 2º Definimos el modelo.
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# 3º Creamos la receta.
nc_recipe_na <-
  recipe(pos_relativa ~.,data=nc_ent_na) %>%
  step_novel(circuitId,driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

# 4º Creamos un WorkFlow
nc_workflow_na<- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(nc_recipe_na)

# 5º Estimamos el modelo.

nc_fit_na <- nc_workflow_na %>%
             last_fit(split = nc_split_na, metrics=metric_set(mae,rmse,rsq))

met_mlcir_na = nc_fit_na %>% collect_metrics()
met_mlcir_na
```

Observamos que le mae ha mejorado levemente con respecto al que obtenemos con la imputación de los datos, sin embargo, hemos perdido la información de bastantes circuitos.

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```
ml_metrics=inner_join(mae_mlc,mae_ml_na,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_ml_na=.estimate.y
  )
```

```r
ml_metrics= ml_metrics %>%
  gather(`mae_ml`, `mae_ml_na`, key = "mae", value = "estimacion")%>%
  filter(estimacion <= 1)
ml_metrics$mae=as.factor(ml_metrics$mae)
```

```r
ml_metrics %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Modelo Lineal")
```

Árbol de decisión

IMPUTACION KNN

```r
tree_fun <- function(x){
x = x %>%
  mutate(
    driverId=factor(as.numeric(driverId)),
      constructorId=factor(as.numeric(constructorId))
        )
  set.seed(1234)
# 1º Dividimos en entrenamiento y test.
tree_split<- initial_split(as.data.frame(x),prop = 0.70,strata = pos_relativa,breaks=4)
tree_ent <- tree_split%>% training()
tree_test <- tree_split %>% testing()

#2º Definimos el modelo: Indicamos los parámetros que queremos ajustar.
ar_fc <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

# 3º Definimos el grid:  valores que pueden tomar los parametros.
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(range = c(1,30)),
                          levels = 5)
# 4º Validación cruzada
set.seed(1234)
cell_folds <- vfold_cv(data = tree_ent,
                       v=5,
                       strata = pos_relativa)

# 5º Definimos la receta.
tree_recipe<- recipe(pos_relativa ~. , data=tree_ent) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(driverId,constructorId) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_numeric(), -all_outcomes()) %>%
 step_center(all_numeric(), -all_outcomes()) %>%
 step_scale(all_numeric(), -all_outcomes()) %>%
 step_corr(all_predictors()) %>%
```

```
  step_lincomb(all_predictors())

# 6º Creamos el Workflow
set.seed(1234)

tree_wf <- workflow() %>%
  add_model(ar_fc) %>%
 add_recipe(tree_recipe)

# 7º Ajustamos el modelo a traves del grid.
tree_res <-
  tune_grid(
     object = tree_wf,
    resamples = cell_folds,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid
    )

# 8º Seleccionamos el mejor modelo
best_tree <- tree_res %>%
  select_best("mae")

final_wf <-
  tree_wf %>%
  finalize_workflow(best_tree)

# 9º Ejecutamos el mejor modelo.
final_fit <-
  final_wf %>%
  last_fit(tree_split,metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(tree_split),model = list(final_fit),
           metrics= list(final_fit %>% collect_metrics()),
           predictions = list(final_fit %>% collect_predictions()))
t

}
```

```
m_nc_tree<- n_circuitos %>%
 mutate(res=map(data,tree_fun))

nc2_tree= m_nc_tree %>% unnest(res)

mae_tree=nc2_tree %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_tree
```

Se observa una mejora considerable y se ha solucionado el problema de obtener métricas por encima de 1.

Circuitos como variables.

```
#1º Creamos una particion entre entrenamiento y test.
#2º Definimos el modelo. Indicamos los parámetros que queremos ajustar.
```

```
ar_fc <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid.

tree_grid <- grid_regular(cost_complexity(range=c(0,0.5)),
                          tree_depth(range = c(1,30)),
                          levels = 5)

#4º Validacion cruzada

set.seed(1234)
cell_cfolds <- vfold_cv(data = c_ent,
                        v=5,
                        strata = pos_relativa)

#5º Definimos la receta.

c_recipe_tree <-
  recipe(pos_relativa ~.,data=c_ent) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId,driverId,constructorId) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

#6º Definimos el workflow
  set.seed(1234)

c_tree_wf <- workflow() %>%
  add_model(ar_fc) %>%
 add_recipe(c_recipe_tree)

#7º Ajustamos el modelo.
c_tree_res <-
  tune_grid(
     object = c_tree_wf,
    resamples = cell_cfolds,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid
    )
#8º Seleccionamos el mejor modelo.

best_treeC<- c_tree_res %>%
  select_best("mae")
```

```
final_wfC<-
  c_tree_wf %>%
  finalize_workflow(best_treeC)


 #9º Ejecutamos el mejor modelo.

final_fitC<-
  final_wfC%>%
  last_fit(c_split,metrics=metric_set(mae,rmse,rsq))

met_treecir=final_fitC%>%
  collect_metrics()

met_treecir
```

ELIMINANDO NA

```
tree_fun_na <- function(x){
  x = x %>%
    mutate(
      driverId=factor(as.numeric(driverId)),
      constructorId=factor(as.numeric(constructorId)))
  #1º Creamos una division entre entrenamiento y test.
  set.seed(1234)
tree_split<- initial_split(as.data.frame(x),prop = 0.70,
                           strata = pos_relativa,breaks=4)
tree_ent <- tree_split%>% training()
tree_test <- tree_split %>% testing()
#2º Definimos el modelo, indicando los parámetros a ajustar.
ar_fc <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid.
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(range = c(1,30)),
                          levels = 5)
#4º Validacion cruzada.
set.seed(1234)
cell_folds <- vfold_cv(data = tree_ent,
                       v=5,
                       strata = pos_relativa)
#5º Definimos la receta
recipe_tree_na<- recipe(pos_relativa ~. , data=tree_ent) %>%
  step_novel(driverId,constructorId) %>%
  step_dummy(all_predictors()) %>%
  step_zv(all_predictors()) %>%
 step_center(all_predictors()) %>%
```

```
 step_scale(all_predictors()) %>%
 step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

#6º Creamos el workflow.
set.seed(1234)

tree_wf <- workflow() %>%
  add_model(ar_fc) %>%
 add_recipe(recipe_tree_na)

#7º Ajustamos el modelo.
tree_res <-
  tune_grid(
     object = tree_wf,
    resamples = cell_folds,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid
    )

#8º Seleccionamos el mejor modelo.
best_tree <- tree_res %>%
  select_best("mae")

final_wf <-
  tree_wf %>%
  finalize_workflow(best_tree)
#9º Estimamos el mejor modelo.
final_fit <-
  final_wf %>%
  last_fit(tree_split,metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(tree_split),model = list(final_fit),
           metrics= list(final_fit %>% collect_metrics()),
           predictions = list(final_fit %>% collect_predictions()))
t

}
```

```
m_nc_tree_na<- n_circuitos_na %>%
 mutate(res=map(data,tree_fun_na))

nc2_tree_na= m_nc_tree_na %>% unnest(res)

mae_tree_na=nc2_tree_na %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_tree_na
```

Circuitos como variables.

```
#1º Creamos una dicision entre entrenamiento y test.
#2º Definimos el modelo.Indicamos los parámetros que queremos ajustar.
```

```r
ar_fc <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid.

tree_grid <- grid_regular(cost_complexity(range=c(0,0.5)),
                          tree_depth(range = c(1,30)),
                          levels = 5)


#4º Validación cruzada

set.seed(1234)
tree_folds_cna <- vfold_cv(data = nc_ent_na,
                       v=5,
                       strata = pos_relativa)

#5º Definimos la receta.


recipe_tree_cna <-
  recipe(pos_relativa ~.,data=nc_ent_na) %>%
  step_novel(driverId,constructorId,circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 6º Definimos el workflow
  set.seed(1234)

tree_wf_cna <- workflow() %>%
  add_model(ar_fc) %>%
 add_recipe(recipe_tree_cna)
#7º Ajustamos el modelo

tree_res_cna <-
  tune_grid(
     object = tree_wf_cna,
    resamples = tree_folds_cna,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid
    )
#8º Seleccionamos el mejor modelo.
best_tree_cna <- tree_res_cna %>%
```

```
  select_best("mae")

final_wf_cna <-
  tree_wf_cna %>%
  finalize_workflow(best_tree_cna)
#9º Estimamos el mejor modelo.
final_fit_cna <-
  final_wf_cna %>%
  last_fit(nc_split_na,metrics=metric_set(mae,rmse,rsq))

met_treecir_na = final_fit_cna %>% collect_metrics()
met_treecir_na
```

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```
tree_metrics=inner_join(mae_tree,mae_tree_na,by="circuitId") %>%
  rename(
    mae_tree=.estimate.x,
    mae_tree_na=.estimate.y
  )

tree_metrics= tree_metrics %>%
  gather(`mae_tree`, `mae_tree_na`, key = "mae", value = "estimacion")
tree_metrics$mae=as.factor(tree_metrics$mae)
```

```
tree_metrics %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Árbol de Regresión")
```

Random forest

Imputacion KNN

```
forest_fun = function (x){
x = x %>%
mutate(driverId=factor(as.numeric(driverId)),constructorId=factor(as.numeric(constructorId)))

# 1º Dividimos en entrenamiento y test.
set.seed(1234)
forest_split<- initial_split(as.data.frame(x),prop = 0.70,strata = pos_relativa,breaks = 4)
forest_ent <- forest_split%>% training()
forest_test <- forest_split %>% testing

#2º Definimos el modelo, indicando los parametros que queremos ajustar.
forest_fc <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
```

```
  set_mode("regression")

#3º Definimos el grid.
forest_grid <- expand_grid(mtry=1:5,
                           trees= seq(10,40,5),
                           min_n=seq(1,10,2))
#4º Validacion cruzada.

set.seed(1234)
forest_folds <- vfold_cv(data = forest_ent,
                         v=5,
                         strata = pos_relativa)


#5º Creamos la receta.
recipe_forest <-  recipe(pos_relativa ~.,data=forest_ent) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(driverId,constructorId) %>%
  step_unknown(driverId,constructorId) %>%
  step_dummy(all_predictors()) %>%
  step_zv(all_numeric()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
    step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())



#6º Creamos el workflow.

set.seed(1234)

forest_wf <- workflow() %>%
  add_model(forest_fc) %>%
 add_recipe(recipe_forest,
            blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE) )

#7º Ajustamos el modelo
forest_res <-
  tune_grid(
     object = forest_wf,
    resamples = forest_folds,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_grid
    )
#8º Seleccionamos el mejor modelo
best_forest <- forest_res %>%
  select_best("mae")

final_frst_wf <-
  forest_wf %>%
  finalize_workflow(best_forest)
```

```
#9º Estimamos
forest_fit<- final_frst_wf%>%
             last_fit(split = forest_split,metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(forest_split),model = list(forest_fit),
           metrics= list(forest_fit %>% collect_metrics()),
           predictions = list(forest_fit %>% collect_predictions()))
t
}
```

Aplicamos la función de random forest a cada conjunto de datos correspondiente a cada circuito.

```
m_nc_forest<- n_circuitos %>%
 mutate(res=map(data,forest_fun))

nc2_forest= m_nc_forest %>% unnest(res)

mae_forest=nc2_forest %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_forest
```

Circuitos como variables.

```
#1º Creamos una particion entre entrenamiento y test.
#2º Definimos el modelo.
#Indicamos los parámetros que queremos ajustar.
forest_fc <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")

#3º Definimos el grid.

forest_gridc <- expand_grid(mtry=1:5,
                            trees= seq(10,40,5),
                            min_n=seq(1,10,2))

#4º Validación cruzada

set.seed(1234)
forest_folds_c <- vfold_cv(data =c_ent,
                           v=5,
                           strata = pos_relativa)

#5º Definimos la receta.
recipe_forest_c <-
  recipe(pos_relativa ~.,data=c_ent) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(driverId,constructorId,circuitId) %>%
```

```
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 6º  Definimos el workflow
  set.seed(1234)

forest_wf_c <- workflow() %>%
  add_model(forest_fc) %>%
 add_recipe(recipe_forest_c)

#7º Ajustamos el modelo.
forest_res_c <-
  tune_grid(
     object = forest_wf_c,
    resamples = forest_folds_c,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_gridc
    )
#8º Seleccionamos el mejor modelo.
best_forest_c <- forest_res_c %>%
  select_best("mae")

final_wf_crf <-
  forest_wf_c %>%
  finalize_workflow(best_forest_c)
#9º Estimamos el modelo.
final_fit_c <-
  final_wf_crf %>%
  last_fit(c_split,metrics=metric_set(mae,rmse,rsq))

mae_for_c= final_fit_c %>% collect_metrics()
mae_for_c
```

ELIMINANDO NA

```
forest_fun_na = function (x){
x = x %>%
mutate(driverId=factor(as.numeric(driverId)),constructorId=factor(as.numeric(constructorId)))
#1º Creamos una division entre entrenamiento y test.
set.seed(1234)
forest_split<- initial_split(as.data.frame(x),prop = 0.70,
                             strata = pos_relativa,breaks=4)

forest_ent <- forest_split%>% training()
forest_test <- forest_split %>% testing()
#2º Definimos el modelo, indicando los parametros a estimar
forest_fc <-
  rand_forest(
```

```
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")

#3º Definimos el grid.
forest_grid <- expand_grid(mtry=1:5,
                           trees= seq(10,40,5),
                           min_n=seq(1,10,2))
#4º Validacion cruzada.
set.seed(1234)
forest_folds <- vfold_cv(data = forest_ent,
                         v=5,
                         strata = pos_relativa)


#5º Definimos la receta
recipe_forest_na <-  recipe(pos_relativa ~.,data=forest_ent) %>%
  step_novel(driverId,constructorId) %>%
  step_unknown(driverId,constructorId) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_numeric()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
   step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())



#6º Creamos el workflow.

set.seed(1234)

forest_wf_na <- workflow() %>%
  add_model(forest_fc) %>%
 add_recipe(recipe_forest_na,
            blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE) )

#7º Ajustamos el modelo.
forest_res_na <-
  tune_grid(
     object = forest_wf_na,
    resamples = forest_folds,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_grid
    )

#8º Seleccionamos el mejor modelo.
best_forest_na <- forest_res_na %>%
  select_best("mae")
```

```
final_frst_wf <-
  forest_wf_na %>%
  finalize_workflow(best_forest_na)

#9º Estimamos el modelo.
forest_fit<- final_frst_wf%>%
             last_fit(split = forest_split,metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(forest_split),model = list(forest_fit),
           metrics= list(forest_fit %>% collect_metrics()),
           predictions = list(forest_fit %>% collect_predictions()))
t
}
```

```
m_nc_forest_na<- n_circuitos_na %>%
 mutate(res=map(data,forest_fun_na))

nc2_forest_na= m_nc_forest_na %>% unnest(res)

mae_forest_na=nc2_forest_na%>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_forest_na
```

Circuitos como variables

```
#1º Creamos una particion entre entrenamiento y test
#2º Definimos el modelo.
#Indicamos los parámetros que queremos ajustar.
forest_fc <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")

#3º Definimos el grid.

forest_gridc <- expand_grid(mtry=1:5,
                            trees= seq(10,40,5),
                            min_n=seq(1,10,2))


#4º Validación cruzada

set.seed(1234)
forest_folds_cna <- vfold_cv(data =nc_ent_na,
                         v=5,
                         strata = pos_relativa)

#5º Definimos la receta.
```

```
recipe_forest_cna <-
  recipe(pos_relativa ~.,data=nc_ent_na) %>%
  step_novel(driverId,constructorId,circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 6º  Definimos el workflow
  set.seed(1234)

forest_wf_cna<- workflow() %>%
  add_model(forest_fc) %>%
 add_recipe(recipe_forest_cna)
#7º Ajustamos el modelo.
forest_res_cna <-
  tune_grid(
     object = forest_wf_cna,
    resamples = forest_folds_cna,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_gridc
    )

#8º Seleccionamos el mejor modelo.
best_forest_cna <- forest_res_cna %>%
  select_best("mae")

final_wf_crfna <-
  forest_wf_cna %>%
  finalize_workflow(best_forest_cna)
#9º Estimamos
final_fit_cna <-
  final_wf_crfna %>%
  last_fit(nc_split_na,metrics=metric_set(mae,rmse,rsq))

mae_for_cna= final_fit_cna%>% collect_metrics()
mae_for_cna
```

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```
forest_metrics=inner_join(mae_forest,mae_forest_na,by="circuitId") %>%
  rename(
    mae_forest=.estimate.x,
    mae_forest_na=.estimate.y
  )

forest_metrics= forest_metrics %>%
  gather(`mae_forest`, `mae_forest_na`, key = "mae", value = "estimacion")
forest_metrics$mae=as.factor(forest_metrics$mae)
```

```
forest_metrics %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Random Forest")
```

Red Neuronal.

Imputacion KNN

```
rn_function=function(x){
x= x%>%
    mutate(
      driverId=factor(as.numeric(driverId)),
      constructorId=factor(as.numeric(constructorId))
      )
 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
rn_split<- initial_split(as.data.frame(x),prop = 0.70,
                         strata = pos_relativa,breaks = 4)
entrn <- rn_split%>% training()
testnc <- rn_split %>% testing()

# 2º Creamos la receta
rn_recipe <-  recipe(pos_relativa ~.,data=entrn) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


#3º definimos el modelo
rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

rn_workflow <- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(rn_recipe)

rn_fit <- rn_workflow %>%
             last_fit(split = rn_split, metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(rn_split),model = list(rn_fit),
           metrics= list(rn_fit %>% collect_metrics()),
           predictions = list(rn_fit %>% collect_predictions()))
t
```

```
}
```

```
m_nc_rn<- n_circuitos %>%
 mutate(res=map(data,rn_function))

nc2_rn= m_nc_rn %>% unnest(res)

mae_rn=nc2_rn %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_rn
```

Circuito como variable.

```
#1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
c_split<- initial_split(conjunto_c,prop = 0.70,
                        strata = pos_relativa,breaks=4)

c_ent <- c_split%>% training()
c_test<-c_split %>% testing()


# 2º Creamos la receta.
c_recipe <-
  recipe(pos_relativa ~.,data=c_ent) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId,driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 4º Creamos un WorkFlow

rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

crn_workflow <- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(c_recipe)


#5º Ejecutamos

crn_fit <- crn_workflow %>%
```

```
            last_fit(split = c_split, metrics=metric_set(mae,rmse,rsq))

met_rncir=crn_fit %>% collect_metrics()
met_rncir
```

Eliminando NA

```
rn_function_na=function(x){
x= x%>%
    mutate(
      driverId=factor(as.numeric(driverId)),
      constructorId=factor(as.numeric(constructorId))
      )
 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
rn_split<- initial_split(as.data.frame(x),prop = 0.70,
                         strata = pos_relativa,breaks = 4)
entrn <- rn_split%>% training()
testnc <- rn_split %>% testing()

# 2º Creamos la receta
rn_recipe <-  recipe(pos_relativa ~.,data=entrn) %>%
  step_novel(driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


#3º definimos el modelo
rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

rn_workflow <- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(rn_recipe)

rn_fit <- rn_workflow %>%
             last_fit(split = rn_split, metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(rn_split),model = list(rn_fit),
           metrics= list(rn_fit %>% collect_metrics()),
           predictions = list(rn_fit %>% collect_predictions()))
t

}
```

```
m_nc_rn_na<- n_circuitos_na %>%
 mutate(res=map(data,rn_function_na))

nc2_rn_na= m_nc_rn_na %>% unnest(res)

mae_rn_na=nc2_rn_na %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_rn_na
```

Circuito como variables

```
set.seed(1234)
nc_split_na<- initial_split(conjunto2_c,prop = 0.70,
                            strata = pos_relativa,breaks=4)

nc_ent_na <- nc_split_na%>% training()
nc_test_na<- nc_split_na%>% testing()


# 2º Creamos la receta.


nc_recipe_na <-
  recipe(pos_relativa ~.,data=nc_ent_na) %>%
  step_novel(circuitId,driverId,constructorId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

# 4º Creamos un WorkFlow

rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

rn_workflow_na<- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(nc_recipe_na)

# 5º Ejecutamos

rn_fit_na <- rn_workflow_na %>%
             last_fit(split = nc_split_na, metrics=metric_set(mae,rmse,rsq))

met_rncir_na = rn_fit_na %>% collect_metrics()
met_rncir_na
```

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```
rn_metrics=inner_join(mae_rn,mae_rn_na,by="circuitId") %>%
  rename(
    mae_rn=.estimate.x,
    mae_rn_na=.estimate.y
  )

rn_metrics= rn_metrics %>%
  gather(`mae_rn`, `mae_rn_na`, key = "mae", value = "estimacion")
rn_metrics$mae=as.factor(rn_metrics$mae)
```

```
rn_metrics %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Red Neuronal")
```

Anexo B

Segunda prueba.

En este anexo vamos a crear en primer lugar las variables que vamos a utilizar en la segunda prueba, definiendo consigo los conjunto de datos para la modelización. Por último, creamos los modelos con las variables definidas anteriormente con las técnicas que ya hemos mencionado, realizando a su vez comparaciones en cada modelo según la técnica.

Segunda prueba.

Vamos a introducir en los modelos, nuevas variables que asemejen un poco más a los pilotos entre sí y a los constructores. Vamos a sustiturir las variables categóricas driverId, constructorId por aspectos que caractericen su rendimiento, y predigan así en función de las variables que caracterizan a los pilotos y escuderías en lugar de sus nombres concretos.

PILOTOS

Numero total de carreras

Numero total de grandes premios que ha disputado cada piloto.

```
nc_pilo = nv_pilo_cc %>%  group_by(driverId) %>%
  summarise(tc= sum(totalc))
```

Posición media relativa piloto.

Posición media relativa piloto circuito constructor.

Posicion media en la que quedó en cada circuito según el constructor.

```
posm_piloc =  datos_m %>%  group_by(circuitId,driverId,constructorId) %>%
  summarise(pos_mean= mean(pos_relativa))
```

Posición media relativa piloto circuito.

```
posm_pilo = datos_m %>%  group_by(circuitId,driverId) %>%
  summarise(posic_m= mean(pos_relativa))
```

Probabilidad de hacer podium piloto.

Probabilidad hacer podium piloto circuito constructor

Según el constructor.

En primer lugar, vamos a calcular el número de podiums que ha hecho cada piloto con cada constructor en cada circuito.

```
pod_prob =  datos_m %>%  filter(position <= 3) %>%
  group_by(circuitId,driverId,constructorId) %>%
  summarise(pod_nc=n())
```

Ahora vemos el número de veces que ha corrido en cada circuito con cada constructor y hacemos la proporción.

```
pod_prob=left_join(pod_prob,nv_pilo_cc,by=c("circuitId","driverId","constructorId"))

pod_prob= pod_prob %>% mutate(prob=pod_nc/totalc) %>% select(-c(pod_nc,totalc))
```

Probabilidad de hacer podium piloto circuito

En primer lugar, calculamos el numero de podiums que ha conseguido cada piloto en cada circuito.

```
pod_prob2 =  datos_m %>%  filter(position <= 3) %>%
  group_by(circuitId,driverId) %>%
  summarise(pod_nc2=n())
```

Ahora tomo el número de veces que ha corrido en cada circuito y hacemos la proporción.

```
pod_prob2 = left_join(pod_prob2,nv_pilo_c,by=c("circuitId","driverId"))
pod_prob2=pod_prob2 %>%  mutate(probc= pod_nc2/tveces) %>% select(-c(pod_nc2,tveces))
```

Probabilidad de hacer podium piloto.

Calculamos el número de podiums que ha hecho un piloto en lo que lleva

```
pod_prob3 =  datos_m %>%  filter(position <= 3) %>%
  group_by(driverId) %>%
  summarise(pod_p=n())
```

Ahora tomo el número de carreras que ha disputado y hago la proporción.

```
pod_prob3 = left_join(pod_prob3,nc_pilo,by=c("driverId"))
pod_prob3=pod_prob3 %>%  mutate(prob_p= pod_p/tc) %>%  select(-c(pod_p,tc))
```

CONSTRUCTORES.

```
const_c= left_join(resul_construc,carreras,by="raceId") %>%
  select(circuitId,constructorId,points,year)
```

Numero de veces que ha corrido en el circuito

```
nv_circons=datos_m %>% group_by(circuitId,constructorId) %>%
  distinct(raceId) %>% count()
```

Puntuación media conseguida en cada circuito.

```
puntm_cc = const_c %>% group_by(circuitId,constructorId) %>%
  summarise(point_mean  = mean(points))
```

Puntuación media conseguida

En primer lugar calculo número de años que lleva cada escudería participando en F1.

```
yeas_const = const_c %>% group_by(constructorId) %>%
  summarise(n_years= n_distinct(year))
```

A continuación calculo el total de puntos que ha conseguido en todos esos años, y hago la media.

```
punt_totalm= const_c %>%  group_by(constructorId) %>%
  summarise(punt_mean= sum(points))

punt_totalm = left_join(punt_totalm,yeas_const,by="constructorId")
punt_totalm = punt_totalm %>%
  mutate(points_m = punt_mean/n_years) %>% select(-c(punt_mean,n_years))
```

Conjunto de datos.

En primer lugar, voy a crear el conjunto con todas las variables.

```
conjunto3_id=datos_m %>%  select(circuitId,driverId,constructorId,grid,pos_relativa,round)
conjunto3_id=left_join(conjunto3_id,nc_pilo,by="driverId")
conjunto3_id=left_join(conjunto3_id,nv_pilo_cc,by=c("driverId","circuitId","constructorId"))
conjunto3_id=left_join(conjunto3_id,nv_pilo_c,by=c("driverId","circuitId"))
conjunto3_id = left_join(conjunto3_id,posm_pilo,by=c("circuitId","driverId"))
conjunto3_id=left_join(conjunto3_id,posm_piloc,by=c("circuitId","driverId","constructorId"))
conjunto3_id = left_join(conjunto3_id,pod_prob,by=c("circuitId","driverId","constructorId"))
conjunto3_id=left_join(conjunto3_id,pod_prob2,by=c("circuitId","driverId"))
conjunto3_id=left_join(conjunto3_id,pod_prob3,by="driverId")
conjunto3_id = left_join(conjunto3_id,pacc,by=c("circuitId","driverId")) %>%
  mutate_at("pAccidentes",~replace(.,is.na(.),0))
conjunto3_id =left_join(conjunto3_id,tiempo_pilo,by="driverId")
conjunto3_id = left_join(conjunto3_id,tiempo_pilocir,by=c("circuitId","driverId"))
conjunto3_id = left_join(conjunto3_id,tiempo_pilocc,by=c("circuitId","constructorId","driverId"))
```

```
conjunto3_id %>%  select(prob,probc,prob_p,tm_p,tm_pci,tmean_pcc) %>% summary
```

En las probabilidades, obtenemos NA de los pilotos que nunca han conseguido un podium por lo que esos NA los vamos a reemplazar por 0.

```
conjunto3_id = conjunto3_id %>% mutate_at(c("prob","probc","prob_p"),~replace(.,is.na(.),0))
```

Imputamos los NA de los tiempos por la media.

```
conjunto3_id = conjunto3_id %>%
  mutate_at("tm_p",~replace(.,is.na(.),tm_pgeneral$tm_gen))
conjunto3_id = conjunto3_id %>%
  mutate_at("tm_pci",~replace(.,is.na(.),tm_pcgeneral$tm_pcgen))
conjunto3_id = conjunto3_id %>%
  mutate_at("tmean_pcc",~replace(.,is.na(.),tm_pccgeneral$tm_pccgen))
```

```
conjunto3_id %>% summary
```

Ahora introducimos las variables correspondientes a los constructores.

```
conjunto3_id = left_join(conjunto3_id,nv_circons,by=c("circuitId","constructorId"))
conjunto3_id = left_join(conjunto3_id,puntm_cc,by=c("circuitId","constructorId") )
conjunto3_id = left_join(conjunto3_id,punt_totalm,by="constructorId")
conjunto3_id = conjunto3_id %>%
  mutate_at(c("point_mean","points_m"),~replace(.,is.na(.),0))
conjunto3_id = left_join(conjunto3_id,n_stop,by=c("circuitId","constructorId"))
conjunto3_id= left_join(conjunto3_id,t_stop,by=c("circuitId","constructorId"))
conjunto3_id %>% select (stop_mean,timestop_mean) %>% summary
```

Obtenemos NA porque la información de las paradas en boxes no empezó a recogerse hasta el año 2011, luego, para algunos constructores en determinados circuitos no tendremos información. Al hacer el modelo, vamos a imputar dichos datos.

Como hemos calculado variables que sustituyan a driverId y constructorId eliminamos estas variables del conjunto.

```r
conjunto3= conjunto3_id %>% select(-c("driverId","constructorId"))
conjunto4= conjunto3 %>% na.omit()
```

```r
n_circuitos3= conjunto3%>%
   group_by(circuitId) %>%
  nest()
```

```r
ncir23= n_circuitos3 %>% mutate(
  tieneNA= data %>%  map(~pull(.x,stop_mean)  %>% is.na(.) %>% all()))

n_circuitos3 =ncir23 %>%
  unnest() %>%
  mutate(
    stop_mean = ifelse(tieneNA==TRUE,medTotal$med_p,stop_mean),
    timestop_mean = ifelse(tieneNA==TRUE,medTotal$med_tp,timestop_mean)
  ) %>%
  select(-c(tieneNA)) %>%
   group_by(circuitId) %>%
  nest()
```

```r
n_circuitos_na2 =conjunto4 %>%
  group_by(circuitId) %>%
  nest()
```

Modelos

Vamos a hacer los modelos definidos en la prueba1 con las nuevas variables.

Modelo lineal

IMPUTACION KNN

```r
lmc_fun3 <- function (x) {
 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
nc_split3<- initial_split(as.data.frame(x),prop = 0.70,
                          strata = pos_relativa,breaks = 4)
entnc3<- nc_split3%>% training()
testnc3<- nc_split3 %>% testing()

#2º definimos el modelo
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")



# 3º Creamos la receta
nc_recipe3 <-  recipe(pos_relativa ~.,data=entnc3) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
```

```
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())



#4º Creamos el workflow
nc_workflow3 <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(nc_recipe3)

#5º Estimamos el modelo.

nclm_fit3 <- nc_workflow3 %>%
             last_fit(split = nc_split3, metrics=metric_set(mae,rmse,rsq))



t3 = tibble(split = list(nc_split3),model = list(nclm_fit3),
            metrics= list(nclm_fit3 %>% collect_metrics()),
            predictions = list(nclm_fit3 %>% collect_predictions()))
t3
}
```

```
m_nc3 <- n_circuitos3 %>%
 mutate(res=map(data,lmc_fun3))


nc3=m_nc3 %>% unnest(res)

mae_ml3=nc3 %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>% select(circuitId,.estimate)
mae_ml3
```

Circuitos como variable.

```
conjunto_c3 = n_circuitos3 %>% unnest()
conjunto_c3=conjunto_c3 %>% mutate(circuitId=as.factor(circuitId))
```

```
#1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
c_split3<- initial_split(conjunto_c3,
                         prop = 0.70,
                         strata = pos_relativa,
                         breaks=4)


c_ent3 <- c_split3%>% training()
c_test3<-c_split3%>% testing()

#2º Definimos el modelo.

lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")
```

```
# 3º Creamos la receta.
c_recipe3 <-
  recipe(pos_relativa ~.,data=c_ent3) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 4º Creamos un WorkFlow

c_workflow3 <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(c_recipe3)


#5º Ejecutamos

c_fit3 <- c_workflow3 %>%
             last_fit(split = c_split3,
                      metrics=metric_set(mae,rmse,rsq))

met_mlcir3=c_fit3 %>% collect_metrics()
met_mlcir3
```

Eliminando NA

```
lmc_fun_na2 <- function (x) {
 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
nc_split<- initial_split(as.data.frame(x),
                         prop = 0.70,
                         strata = pos_relativa,
                         breaks = 4)

entnc <- nc_split%>% training()
testnc <- nc_split %>% testing()

#2º definimos el modelo
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")



# 3º Creamos la receta
nc_recipe<-  recipe(pos_relativa ~.,data=entnc) %>%
```

```
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

# 4º Creamos el workflow.

nc_workflow <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(nc_recipe)
#5º Ejecutamos el modelo.
nclm_fit <- nc_workflow %>%
             last_fit(split = nc_split,
                      metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(nc_split),
           model = list(nclm_fit),
           metrics= list(nclm_fit %>% collect_metrics()),
           predictions = list(nclm_fit %>% collect_predictions()))
t
}
```

```
m_nc_na2<- n_circuitos_na2 %>%
 mutate(res=map(data,lmc_fun_na2))

nc2_na2=m_nc_na2 %>% unnest(res)

mae_ml_na2=nc2_na2 %>% unnest(metrics) %>%
  filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_ml_na2
```

Circuitos como variable.

```
conjunto_c4=conjunto4 %>% mutate(circuitId=as.factor(circuitId))
```

```
#1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
c_split3_na <- initial_split(conjunto_c4,
                             prop = 0.70,
                             strata = pos_relativa,
                             breaks=4)


c_ent3_na <- c_split3_na%>% training()
c_test3_na<-c_split3_na %>% testing()

#2º Definimos el modelo.

lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")
```

```r
# 3º Creamos la receta.

c_recipe3_na <-
  recipe(pos_relativa ~.,data=c_ent3_na) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 4º Creamos un WorkFlow

c_workflow3_na <- workflow() %>%
                  add_model(lm_model) %>%
                  add_recipe(c_recipe3_na)


#5º Ejecutamos

c_fit3_na <- c_workflow3_na %>%
             last_fit(split = c_split3_na,metrics=metric_set(mae,rmse,rsq))

met_mlcir3_na=c_fit3_na %>% collect_metrics()
met_mlcir3_na
```

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```r
ml_metrics3=inner_join(mae_ml3,mae_ml_na2,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_ml_na=.estimate.y
  )

ml_metrics3= ml_metrics3 %>%
  gather(`mae_ml`, `mae_ml_na`, key = "mae", value = "estimacion")
ml_metrics3$mae=as.factor(ml_metrics3$mae)
```

```r
ml_metrics3 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Modelo Lineal")
```

Arbol de regresión.

IMPUTACION KNN

```r
tree_fun3=function(x){
# 1º Creamos una particion entre entrenamiento y test.
```

```r
set.seed(1234)
tree_split3<- initial_split(as.data.frame(x),
                            prop = 0.70,strata = pos_relativa,breaks = 4)
tree_ent3<- tree_split3 %>% training()
tree_test3<- tree_split3 %>% testing()

#2º Definimos el modelo.
#Indicamos los parámetros que queremos ajustar.

ar_fc3 <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid.
tree_grid3 <- grid_regular(cost_complexity(),
                           tree_depth(range = c(1,30)),
                           levels = 5)
#4º Validacion cruzada
set.seed(1234)
cell_folds3 <- vfold_cv(data = tree_ent3,
                        v=5,
                        strata = pos_relativa)
#5º Definimos la receta.

tree_recipe3 <-
  recipe(pos_relativa ~.,data=tree_ent3) %>%
 step_impute_knn(stop_mean,timestop_mean) %>%
 step_zv(all_predictors()) %>%
 step_center(all_predictors()) %>%
 step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

#6º Creamos el Workflow

set.seed(1234)
tree_wf3 <- workflow() %>%
  add_model(ar_fc3) %>%
 add_recipe(tree_recipe3)

#7º Ajustamos el modelo
tree_res3 <-
  tune_grid(
     object = tree_wf3,
    resamples = cell_folds3,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid3
    )
```

```
#8º Seleccionamos el mejor modelo
best_tree3 <- tree_res3 %>%
  select_best("mae")

final_wf3 <-
  tree_wf3 %>%
  finalize_workflow(best_tree3)

#9ºEjecutamos el mejor modelo.
final_fit3 <-
  final_wf3 %>%
  last_fit(tree_split3,metrics=metric_set(mae,rmse,rsq))


t = tibble(split = list(tree_split3),model = list(final_fit3),
           metrics= list(final_fit3 %>% collect_metrics()),
           predictions = list(final_fit3 %>% collect_predictions()))
t
}
```

```
m_nc_tree3<- n_circuitos3 %>%
 mutate(res=map(data,tree_fun3))

nc2_tree3= m_nc_tree3 %>% unnest(res)

mae_tree3=nc2_tree3 %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_tree3
```

Circuito como variable.

```
#1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
c_split3<- initial_split(conjunto_c3,
                         prop = 0.70,
                         strata = pos_relativa,
                         breaks=4)


c_ent3 <- c_split3%>% training()
c_test3<-c_split3%>% testing()
#2º Definimos el modelo
ar_fc3 <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid.
tree_grid3 <- grid_regular(cost_complexity(),
```

```
                          tree_depth(range = c(1,30)),
                          levels = 5)
#4º Validacion cruzada
set.seed(1234)
cell_folds3 <- vfold_cv(data = c_ent3,
                        v=5,
                        strata = pos_relativa)

# 5º Creamos la receta.

c_recipe3 <-
  recipe(pos_relativa ~.,data=c_ent3) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


#6º Creamos el Workflow

set.seed(1234)
tree_wf3 <- workflow() %>%
  add_model(ar_fc3) %>%
 add_recipe(c_recipe3)

#7º Ajustamos el modelo.
tree_res3 <-
  tune_grid(
     object = tree_wf3,
    resamples = cell_folds3,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid3
    )


#8º Seleccionamos el mejor modelo
best_tree3 <- tree_res3 %>%
  select_best("mae")

final_wf3 <-
  tree_wf3 %>%
  finalize_workflow(best_tree3)

#9ºEjecutamos el mejor modelo.
final_fit3 <-
  final_wf3 %>%
  last_fit(c_split3,metrics=metric_set(mae,rmse,rsq))
```

```
met_treecir3=final_fit3 %>% collect_metrics()
met_treecir3
```

Eliminando NA

```
tree_fun3_na=function(x){
  #1º Creamos una particion entre entrenamiento y test.
set.seed(1234)
tree_split<- initial_split(as.data.frame(x),
                           prop = 0.70,
                           strata = pos_relativa,
                           breaks = 4)
tree_ent <- tree_split%>% training()
tree_test <- tree_split %>% testing()

#2º Definimos el modelo.
ar_fc <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid

tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(range = c(1,30)),
                         levels = 5)
#4º Validacion cruzada.
set.seed(1234)
cell_folds <- vfold_cv(data = tree_ent,
                       v=5,
                       strata = pos_relativa)

#5º Definimos la receta
recipe_tree_na<- recipe(pos_relativa ~. , data=tree_ent) %>%
 step_zv(all_numeric(), -all_outcomes()) %>%
 step_center(all_numeric(), -all_outcomes()) %>%
 step_scale(all_numeric(), -all_outcomes()) %>%
 step_corr(all_predictors()) %>%
 step_lincomb(all_predictors())


#6º Creamos el workflow
set.seed(1234)

tree_wf <- workflow() %>%
  add_model(ar_fc) %>%
 add_recipe(recipe_tree_na)

#7º Ajustamos el modelo
```

```
tree_res <-
  tune_grid(
     object = tree_wf,
    resamples = cell_folds,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid
    )

#8º Seleccinamos el mejor modelo.
best_tree <- tree_res %>%
  select_best("mae")

final_wf <-
  tree_wf %>%
  finalize_workflow(best_tree)

#9º Estimamos.
final_fit <-
  final_wf %>%
  last_fit(tree_split,metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(tree_split),
           model = list(final_fit),
           metrics= list(final_fit %>% collect_metrics()),
           predictions = list(final_fit %>% collect_predictions()))
t

}
```

```
m_nc_tree_na2<- n_circuitos_na2 %>%
 mutate(res=map(data,tree_fun3_na))

nc2_tree_na2= m_nc_tree_na2 %>% unnest(res)

mae_tree_na2=nc2_tree_na2 %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_tree_na2
```

Circuitos como variable.

```
#1º Creamos una partición entre conjunto test y entrenamiento

#2º Definimos el modelo.
ar_fc3 <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("regression")

#3º Definimos el grid.
```

```r
tree_grid3 <- grid_regular(cost_complexity(),
                           tree_depth(range = c(1,30)),
                           levels = 5)
#4º Validacion cruzada
set.seed(1234)
cell_folds3na <- vfold_cv(data = c_ent3_na,
                          v=5,
                          strata = pos_relativa)

# 5º Creamos la receta.

c_recipe3na <-
  recipe(pos_relativa ~.,data=c_ent3_na) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


#6º Creamos el Workflow

set.seed(1234)
tree_wf3na <- workflow() %>%
  add_model(ar_fc3) %>%
 add_recipe(c_recipe3na)
#7º Ajustamos el modelo.
tree_res3na <-
  tune_grid(
     object = tree_wf3,
    resamples = cell_folds3na,
    metrics=metric_set(mae,rmse,rsq),
    grid = tree_grid3
    )


#8º Seleccionamos el mejor modelo
best_tree3na <- tree_res3na %>%
  select_best("mae")

final_wf3na <-
  tree_wf3na %>%
  finalize_workflow(best_tree3na)

#9ºEjecutamos el mejor modelo.

final_fit3na <-
  final_wf3na %>%
  last_fit(c_split3_na,metrics=metric_set(mae,rmse,rsq))
```

```
met_treecir3_na=final_fit3na %>% collect_metrics()
met_treecir3_na
```

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```
tree_metrics3=inner_join(mae_tree3,mae_tree_na2,by="circuitId") %>%
  rename(
    mae_tree=.estimate.x,
    mae_tree_na=.estimate.y
  )

tree_metrics3= tree_metrics3 %>%
  gather(`mae_tree`, `mae_tree_na`, key = "mae", value = "estimacion")
tree_metrics3$mae=as.factor(tree_metrics3$mae)
```

```
tree_metrics3 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Árbol")
```

Random forest

Imputando KNN

```
forest_fun3 = function (x) {
  #1º Creamos una particion en entrenamiento y test.
  set.seed(1234)
forest_split3<- initial_split(as.data.frame(x),
                              prop = 0.70,strata = pos_relativa,breaks = 4)


forest_ent3 <- forest_split3%>% training()
forest_test3 <- forest_split3 %>% testing()

#2º Definimos el modelo y los parametros a optimizar
forest_fc3 <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")

#3º Grid de parametros
forest_grid3 <- expand_grid(mtry=4:10,
                            trees= seq(10,40,5),
                            min_n=seq(1,10,2))

#4º Validacion cruzada

set.seed(1234)
```

```
forest_folds3 <- vfold_cv(data = forest_ent3,
                          v=5,
                          strata = pos_relativa)

#5º Definimos la receta

forest_recipe3 <-
  recipe(pos_relativa ~.,data=forest_ent3) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
 step_zv(all_predictors()) %>%
 step_center(all_predictors()) %>%
 step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

#6º Workflow

set.seed(1234)

forest_wf3 <- workflow() %>%
  add_model(forest_fc3) %>%
 add_recipe(forest_recipe3)

#7º Ajustamos el modelo.

forest_res3 <-
  tune_grid(
     object = forest_wf3,
    resamples = forest_folds3,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_grid3
    )

# 8º Seleccionamos el mejor
best_forest3 <- forest_res3 %>%
  select_best("mae")

final_frst_wf3 <-
  forest_wf3 %>%
  finalize_workflow(best_forest3)

#9º Estimamos el modelo.
forest_fit3<- final_frst_wf3%>%
             last_fit(split= forest_split3,metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(forest_split3),
           model = list(forest_fit3),
           metrics= list(forest_fit3 %>% collect_metrics()),
           predictions = list(forest_fit3 %>% collect_predictions()))
t

}
```

```
m_nc_forest3<- n_circuitos3 %>%
 mutate(res=map(data,forest_fun3))

nc2_forest3= m_nc_forest3 %>% unnest(res)

mae_forest3=nc2_forest3 %>% unnest(metrics) %>%
  filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)

mae_forest3
```

Circuitos como variables

```
#1º Creamos una particion en entrenamiento y test
#2º Definimos el modelo.
#Indicamos los parámetros que queremos ajustar.
forest_fc3 <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")

#3º Definimos el grid.

forest_grid3 <- expand_grid(mtry=1:5,
                            trees= seq(10,40,5),
                            min_n=seq(1,10,2))


#4º Validación cruzada

set.seed(1234)
forest_folds_c3 <- vfold_cv(data =c_ent3,
                        v=5,
                        strata = pos_relativa)

#5º Definimos la receta.
recipe_forest_c3 <-
  recipe(pos_relativa ~.,data=c_ent3) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())
```

```
# 6º  Definimos el workflow
  set.seed(1234)

forest_wf_c3<- workflow() %>%
  add_model(forest_fc3) %>%
 add_recipe(recipe_forest_c3)
#7º Ajustamos el modelo.
forest_res_c3 <-
  tune_grid(
     object = forest_wf_c3,
    resamples = forest_folds_c3,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_grid3
    )
#8º Seleccionamos el mejor modelo.
best_forest_c3 <- forest_res_c3 %>%
  select_best("mae")

final_wf_c3 <-
  forest_wf_c3 %>%
  finalize_workflow(best_forest_c3)

#9º Estimamos el modelo.
final_fit_c3 <-
  final_wf_c3 %>%
  last_fit(c_split3,metrics=metric_set(mae,rmse,rsq))

mae_for_c3= final_fit_c3%>% collect_metrics()
mae_for_c3
```

Eliminando NA

```
forest_fun_na2 = function (x){
  #1º Creamos una particion en entrenamiento y test.
set.seed(1234)
forest_split<- initial_split(as.data.frame(x),prop = 0.70,
                             strata = pos_relativa,breaks=4)


forest_ent <- forest_split%>% training()
forest_test <- forest_split %>% testing()

#2º Definimos el modelo.
forest_fc <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")
#3º Creamos el grid
forest_grid <- expand_grid(mtry=1:5,
```

```
                          trees= seq(10,40,5),
                          min_n=seq(1,10,2))

#4º Validacion cruzada.
set.seed(1234)
forest_folds <- vfold_cv(data = forest_ent,
                         v=5,
                         strata = pos_relativa)

#5º Definimos la receta

recipe_forest_na <-  recipe(pos_relativa ~.,data=forest_ent) %>%
  step_zv(all_numeric()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

# 6º Creamos el workflow
set.seed(1234)

forest_wf_na <- workflow() %>%
  add_model(forest_fc) %>%
 add_recipe(recipe_forest_na)
#7º Ajustamos el modelo
forest_res_na <-
  tune_grid(
     object = forest_wf_na,
    resamples = forest_folds,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_grid
    )
# 8º Seleccionamos el mejor modelo
best_forest_na <- forest_res_na %>%
  select_best("mae")

final_frst_wf <-
  forest_wf_na %>%
  finalize_workflow(best_forest_na)

#9º Estimamos
forest_fit<- final_frst_wf%>%
             last_fit(split = forest_split,
                      metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(forest_split),model = list(forest_fit),
           metrics= list(forest_fit %>% collect_metrics()),
           predictions = list(forest_fit %>% collect_predictions()))
t
}
```

```
m_nc_forest_na2<- n_circuitos_na2 %>%
 mutate(res=map(data,forest_fun_na2))
```

```
nc2_forest_na2= m_nc_forest_na2 %>% unnest(res)

mae_forest_na2=nc2_forest_na2 %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_forest_na2
```

Circuitos como variables

```
#1º Creamos una particion entre entrenamiento y test.
#2º Definimos el modelo.
#Indicamos los parámetros que queremos ajustar.
forest_fc3 <-
  rand_forest(
    mtry = tune(),
    trees = tune(),
    min_n=tune()
  ) %>%
  set_engine("ranger") %>%
  set_mode("regression")

#3º Definimos el grid.

forest_grid3 <- expand_grid(mtry=1:5,
                            trees= seq(10,40,5),
                            min_n=seq(1,10,2))


#4º Validación cruzada

set.seed(1234)
forest_folds_c3na <- vfold_cv(data =c_ent3_na,
                          v=5,
                          strata = pos_relativa)

#5º Definimos la receta.
recipe_forest_c3na <-
  recipe(pos_relativa ~.,data=c_ent3_na) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 6º  Definimos el workflow
  set.seed(1234)

forest_wf_c3na<- workflow() %>%
  add_model(forest_fc3) %>%
 add_recipe(recipe_forest_c3na)
```

```
#7º Ajustamos el modelo
forest_res_c3na <-
  tune_grid(
     object = forest_wf_c3na,
    resamples = forest_folds_c3na,
    metrics=metric_set(mae,rmse,rsq),
    grid = forest_grid3
    )
#8º Seleccionamos el mejor modelo.
best_forest_c3na <- forest_res_c3na %>%
  select_best("mae")

final_wf_c3na <-
  forest_wf_c3na %>%
  finalize_workflow(best_forest_c3na)
#9º Estimamos
final_fit_c3na <-
  final_wf_c3na %>%
  last_fit(c_split3_na,metrics=metric_set(mae,rmse,rsq))

mae_for_c3na= final_fit_c3na%>% collect_metrics()
mae_for_c3na
```

Comparacion

Vamos a comparar los valores de la métrica según ambas técnicas.

```
rf_metrics3=inner_join(mae_forest3,mae_forest_na2,by="circuitId") %>%
  rename(
    mae_rf=.estimate.x,
    mae_rf_na=.estimate.y
  )

rf_metrics3= rf_metrics3 %>%
  gather(`mae_rf`, `mae_rf_na`, key = "mae", value = "estimacion")
rf_metrics3$mae=as.factor(rf_metrics3$mae)
```

```
rf_metrics3 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Random forest")
```

Red Neuronal.

Imputando KNN

```
rn_fun3=function(x){

 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
rn_split2<- initial_split(as.data.frame(x),prop = 0.70,
                          strata = pos_relativa,breaks = 4)
entrn2 <- rn_split2%>% training()
testnc2 <- rn_split2 %>% testing()
```

```
# 2º Creamos la receta
rn_recipe2 <-  recipe(pos_relativa ~.,data=entrn2) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


#3º definimos el modelo
rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

rn_workflow2<- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(rn_recipe2)

rn_fit2 <- rn_workflow2 %>%
             last_fit(split = rn_split2, metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(rn_split),model = list(rn_fit2),
           metrics= list(rn_fit2 %>% collect_metrics()),
           predictions = list(rn_fit2 %>% collect_predictions()))
t

}

```
```
m_nc_rn3<- n_circuitos3 %>%
 mutate(res=map(data,rn_fun3))

nc2_rn3= m_nc_rn3 %>% unnest(res)

mae_rn3=nc2_rn3 %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_rn3
```
```
### Circuito como variable.
```
```
#1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
c_split3<- initial_split(conjunto_c3,
                         prop = 0.70,
                         strata = pos_relativa,
                         breaks=4)
```

```
c_ent3 <- c_split3%>% training()
c_test3<-c_split3%>% testing()

# 2º Creamos la receta.
c_recipe3 <-
  recipe(pos_relativa ~.,data=c_ent3) %>%
  step_impute_knn(stop_mean,timestop_mean) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


# 4º Creamos un WorkFlow

rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

crn_workflow3 <- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(c_recipe3)


#5º Ejecutamos

crn_fit3 <- crn_workflow3 %>%
             last_fit(split = c_split3, metrics=metric_set(mae,rmse,rsq))

met_rncir3=crn_fit3 %>% collect_metrics()
met_rncir3
```
Eliminando NA

```
rn_fun3_na=function(x){

 # 1º Creamos una partición entre conjunto test y entrenamiento
set.seed(1234)
rn_split<- initial_split(as.data.frame(x),prop = 0.70,strata = pos_relativa,breaks = 4)
entrn <- rn_split%>% training()
testnc <- rn_split %>% testing()

# 2º Creamos la receta
rn_recipe <-  recipe(pos_relativa ~.,data=entrn) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
```

```
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())


#3º definimos el modelo
rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

rn_workflow <- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(rn_recipe)

rn_fit <- rn_workflow %>%
             last_fit(split = rn_split, metrics=metric_set(mae,rmse,rsq))

t = tibble(split = list(rn_split),model = list(rn_fit),
           metrics= list(rn_fit %>% collect_metrics()),
           predictions = list(rn_fit %>% collect_predictions()))
t

}

```
```
m_nc_rn_na2<- n_circuitos_na2 %>%
 mutate(res=map(data,rn_fun3_na))

nc2_rn_na2= m_nc_rn_na2 %>% unnest(res)

mae_rn_na2=nc2_rn_na2 %>% unnest(metrics) %>% filter(.metric == "mae") %>%
  arrange(.estimate) %>%
  select(circuitId,.estimate)
mae_rn_na2
```
```

### Circuito como variables.
```
```
set.seed(1234)
c_split3_na <- initial_split(conjunto_c4,
                             prop = 0.70,
                             strata = pos_relativa,
                             breaks=4)


c_ent3_na <- c_split3_na%>% training()
c_test3_na<-c_split3_na %>% testing()

# 2º Creamos la receta.
```

```
nc_recipe_na3 <-
  recipe(pos_relativa ~.,data=c_ent3_na) %>%
  step_novel(circuitId) %>%
  step_dummy(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_lincomb(all_predictors())

# 4º Creamos un WorkFlow

rn_model <- mlp(
  hidden_units=50,
 epochs =10,
 ) %>%
  set_engine("brulee") %>%
  set_mode("regression")

rn_workflow_na3<- workflow() %>%
                  add_model(rn_model) %>%
                  add_recipe(nc_recipe_na3)

# 5º Ejecutamos

rn_fit_na3 <- rn_workflow_na3 %>%
             last_fit(split = c_split3_na, metrics=metric_set(mae,rmse,rsq))

met_rncir_na3 = rn_fit_na3 %>% collect_metrics()
met_rncir_na3
```
Comparación
Vamos a comparar los valores de la métrica según ambas técnicas.

```
rn_metrics3=inner_join(mae_rn3,mae_rn_na2,by="circuitId") %>%
  rename(
    mae_rn=.estimate.x,
    mae_rn_na=.estimate.y
  )

rn_metrics3= rn_metrics3 %>%
  gather(`mae_rn`, `mae_rn_na`, key = "mae", value = "estimacion")
rn_metrics3$mae=as.factor(rn_metrics3$mae)
```

```
rn_metrics3 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Comparación MAE Red Neuronal")
```

Anexo C

Estudio de las variables.

En este anexo encontramos recogido el código en R del estudio descriptivo de las variables, tanto las de la primera prueba como las de la segunda.

Estudio de las variables

Vamos a realizar un estudio descriptivo de los datos. De esta forma, podremos entender mejor la información que nos proporcionan las variables.

```
skim(conjunto)
```

En el primer conjunto de datos tenemos un total de 12 variables de las cuales una (pos_relativa) es la variable respuesta. Dos de las variables explicativas: driverId,constructorId son tipo factor las cuales al hacer los modelos le aplicaremos la creación de variables dummy. También podemos observar como las variables que contienen NA son las que en los modelos vamos a imputar.

```
skim(conjunto3)
```

En el segundo conjunto de datos tenemos un total de 21 variables de las cuales una (pos_relativa) es la variable respuesta, en este conjunto de datos no tenemos ninguna variable tipo factor, todas son cuantitativas. También podemos observar como las variables que contienen NA son las que en los modelos vamos a imputar

Distribución de la variable respuesta

```
ggplot(data = datos_m, aes(x = pos_relativa)) +
  geom_density(fill = "#FFA07A",color="#FF4500", alpha = 0.8) +
  geom_rug(color="#CD8162",alpha = 0.2) +
  scale_x_continuous(labels = scales::comma) +
  labs(title = "Distribución original") +
  theme_bw()
```

Vamos a comparar la distribución de la posición relativa y la posición absoluta.

```
datos_m %>%
  mutate_at("position",~replace(.,is.na(.),25)) %>%
ggplot(aes(x = position)) +
  geom_density(fill = "#FFA07A",color="#FF4500", alpha = 0.8) +
  geom_rug(color="#CD8162",alpha = 0.2) +
  scale_x_continuous(labels = scales::comma) +
  labs(title = "Distribución original") +
  theme_bw()
```

Estadísticos principales de la variable respuesta.

```
summary(conjunto$pos_relativa)
```

Matriz de correlación.

Conjunto 1.

```
mp=conjunto %>% select(-c(circuitId,constructorId,driverId,stop_mean,timestop_mean))
a=cor(mp)
```

```
a_c= melt(a)
a_c %>% ggplot(aes(x=Var1,y=Var2,fill=value))+
  geom_tile(color = "white")+
 scale_fill_gradient2(low = "blue", high = "red", mid = "pink",
   midpoint = 0, limit = c(-1,1), space = "Lab",
   name="Coeficiente de \n Correlacion") +
  theme_minimal()+
 theme(axis.text.x = element_text(angle = 90, vjust = 1,
    size = 10, hjust = 1))+
 coord_fixed()
```

En valor absoluto.

```
ab=abs(a)
ab_c= melt(ab)
ab_c %>% ggplot(aes(x=Var1,y=Var2,fill=value))+
  geom_tile(color = "white")+
 scale_fill_gradient2(low = "white", high = "red", mid = "pink",
   midpoint = 0.5, limit = c(0,1), space = "Lab",
   name="Coeficiente de \n Correlacion") +
  theme_minimal()+
 theme(axis.text.x = element_text(angle = 90, vjust = 1,
    size = 10, hjust = 1))+
 coord_fixed()
```

Conjunto 3.

```
b=cor(conjunto3 %>% select(-c(circuitId,stop_mean,timestop_mean)))
b_c= melt(b)
b_c %>% ggplot(aes(x=Var1,y=Var2,fill=value))+
  geom_tile(color = "white")+
 scale_fill_gradient2(low = "blue", high = "red", mid = "pink",
   midpoint = 0, limit = c(-1,1), space = "Lab",
   name="Coeficiente de \n correlacion") +
  theme_minimal()+
 theme(axis.text.x = element_text(angle = 90, vjust = 1,
    size = 10, hjust = 1))+
 coord_fixed()
```

En valor absoluto.

```
ba=abs(b)
ba_c= melt(ba)
ba_c %>% ggplot(aes(x=Var1,y=Var2,fill=value))+
  geom_tile(color = "white")+
 scale_fill_gradient2(low = "white", high = "red", mid = "pink",
   midpoint = 0.5, limit = c(0,1), space = "Lab",
   name="Coeficiente de \n correlacion") +
  theme_minimal()+
 theme(axis.text.x = element_text(angle = 90, vjust = 1,
    size = 10, hjust = 1))+
 coord_fixed()
```

Posiciones ganadas/perdidas con respecto a la posicion de salida.

Vamos a coger los datos de la posición de clasificación, la posición final en cada circuito a lo largo de la historia y vamos a ver el número de posiciones ganadas o perdidas con respecto a la posición de salida y a la posición final, dependiendo de cada circuito. Para que no sean demasiados datos veremos las diferencias de posiciones de los que han salido entre los 10 primeros con respecto a su posición final

```
dat_circuit= datos_m %>%
  select(circuitId,grid,position)
```

```
clasif10=dat_circuit %>% filter(grid <=10) %>%
  group_by(circuitId) %>%
  mutate(diferencia=grid - position,
         dif= -1*(diferencia < 0) + 1 *(diferencia >0) )
 clasif10$dif=as.integer( clasif10$dif)
```

Si diferencia > 0 entonces ha ganado posiciones. diferencia < 0 entonces ha perdido posiciones.

```
clasif10 %>% filter(circuitId <=33)%>%
  ggplot(mapping = aes(x=grid,y=diferencia,colour=dif)) +
  geom_point(alpha=1/5)+
  facet_wrap(~circuitId,nrow=8)
```

```
clasif10 %>% filter(circuitId > 33)%>%
  ggplot(mapping = aes(x=grid,y=diferencia,colour=dif)) +
  geom_point(alpha=1/5)+
  facet_wrap(~circuitId,nrow=8)
```

Numero de vuelta en la parada

Número de vuelta en la que hace cada parada según la posición en la que ha quedado

```
nvueltap = inner_join(boxes,datos_m,by=c("raceId","driverId")) %>%
  select(circuitId,stop, lap, position)

nvueltap= left_join(nvueltap,GP,by="circuitId") %>%
   select(circuitId,stop,lap,position,Circuito)
nvueltap$stop=as.factor(nvueltap$stop)
```

```
nvueltap %>%  ggplot(aes(lap,position))+
  geom_point(mapping = aes(x=lap,y=position,colour=stop),alpha=1/2)+
 # geom_smooth()+
  facet_wrap(~circuitId,nrow=8)
```

#Accidentes y colisiones

Según en la posición en la que sale desde el grid, ver en qué posición ha quedado según si ha tenido un accidente o colisión.

```
ac_co =  datos_m %>% filter(statusId==3 | statusId==4) %>%
  select(circuitId,grid,pos_relativa,statusId)


ac_co = left_join(ac_co,GP,by="circuitId")%>%
  select(circuitId,Circuito,grid,pos_relativa,statusId)

ac_co$statusId=as.factor(ac_co$statusId)
```

```r
ac_co %>% filter(circuitId <=49 & pos_relativa >=0.25) %>%  ggplot()+
  geom_point(mapping = aes(x=grid,y=pos_relativa,colour=statusId),alpha=1/2)+
  facet_wrap(~circuitId,nrow=10)+
  scale_y_continuous(breaks = c(0.25,0.5,0.75,1))+
  scale_x_discrete(breaks=c(1,10,20,25))+
  labs(title = "Relación accidentes con posición relativa", color="Estado") +
  scale_color_discrete(labels = c("Accidente", "Colisión"))
```

Anexo D

Comparación modelos y predicciones.

En este anexo vamos a realizar comparaciones de los valores de las métricas de todos los modelos según la prueba y la técnica. Por otro lado, vamos a crear predicciones de varios pilotos en una cierta carrera, según las variables de la segunda prueba a través de la técnica de imputación.

Comparaciones

Vamos a realizar comparaciones de los valores de las métricas de todos los modelos según la prueba y la técnica.

Conjunto 1 (primera prueba)

Imputacion KNN

Metricas

```
metrics1=inner_join(mae_mlc,mae_tree,by="circuitId")
metrics1=inner_join(metrics1,mae_forest, by="circuitId")
metrics1=inner_join(metrics1,mae_rn,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_tree=.estimate.y,
    mae_forest=.estimate.x.x,
    mae_rn=.estimate.y.y
  )

metrics1= metrics1 %>%
  gather(`mae_ml`, `mae_tree`,`mae_forest`,`mae_rn`, key = "mae", value = "estimacion")
metrics1$mae=as.factor(metrics1$mae)
```

```
metrics1 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  geom_hline(yintercept = 1, col="red") +
  labs(title = "Metrica MAE prueba 1")
```

Ahora vamos a reducirnos a los datos donde mae <1.

```
metris_less1 =metrics1 %>% filter(estimacion <= 1)

metris_less1 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
   labs(title = "Metrica MAE prueba 1 mae <=1")
```

Eliminando NA

Metricas

```
metrics1_na=inner_join(mae_ml_na,mae_tree_na,by="circuitId")
metrics1_na=inner_join(metrics1_na,mae_forest_na, by="circuitId")
metrics1_na=inner_join(metrics1_na,mae_rn_na,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_tree=.estimate.y,
    mae_forest=.estimate.x.x,
```

```
    mae_rn=.estimate.y.y
  )

metrics1_na= metrics1_na%>%
  gather(`mae_ml`, `mae_tree`,`mae_forest`,`mae_rn`, key = "mae", value = "estimacion")
metrics1_na$mae=as.factor(metrics1_na$mae)
```

```
metrics1_na %>% filter(estimacion <=1) %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE prueba 1 eliminando Na")
```

Conjunto 3. (segunda prueba)

Imputacion KNN

Metricas

```
metrics3=inner_join(mae_ml3,mae_tree3,by="circuitId")
metrics3=inner_join(metrics3,mae_forest3, by="circuitId")
metrics3=inner_join(metrics3,mae_rn3,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_tree=.estimate.y,
    mae_forest=.estimate.x.x,
    mae_rn=.estimate.y.y
  )

metrics3= metrics3 %>%
  gather(`mae_ml`, `mae_tree`,`mae_forest`,`mae_rn`, key = "mae", value = "estimacion")
metrics3$mae=as.factor(metrics3$mae)
```

```
metrics3 %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE prueba 2")
```

```
metrics3 %>% filter(estimacion <=0.2) %>%
  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE prueba 2 \n intervalo (0,0.2)")
```

Eliminando NA

Metricas

```
metrics2_na=inner_join(mae_ml_na2,mae_tree_na2,by="circuitId")
metrics2_na=inner_join(metrics2_na,mae_forest_na2, by="circuitId")
metrics2_na=inner_join(metrics2_na,mae_rn_na2,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_tree=.estimate.y,
    mae_forest=.estimate.x.x,
    mae_rn=.estimate.y.y
  )
```

```
metrics2_na= metrics2_na%>%
  gather(`mae_ml`, `mae_tree`,`mae_forest`,`mae_rn`, key = "mae", value = "estimacion")
metrics2_na$mae=as.factor(metrics2_na$mae)
```

```
metrics2_na %>%   ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE prueba 2 eliminando Na")
```

Comparaciones ambas pruebas.

Vamos a comparar las métricas de los modelos diferenciando las obtenidas por la prueba 1 y por la prueba 2.

Modelo lineal.

```
ml_mae=inner_join(mae_mlc,mae_ml3,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_ml2=.estimate.y
  )

ml_mae= ml_mae %>%
  gather(`mae_ml`, `mae_ml2`, key = "mae", value = "estimacion")
ml_mae$mae=as.factor(ml_mae$mae)
```

```
ml_mae %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  geom_hline(yintercept = 1, col="black") +
  labs(title = "Metrica MAE Modelo Lineal")
```

```
ml_mae_na=inner_join(mae_ml_na,mae_ml_na2,by="circuitId") %>%
  rename(
    mae_ml=.estimate.x,
    mae_ml2=.estimate.y
  )

ml_mae_na= ml_mae_na %>%
  gather(`mae_ml`, `mae_ml2`, key = "mae", value = "estimacion")
ml_mae_na$mae=as.factor(ml_mae_na$mae)
```

```
ml_mae_na %>% filter(estimacion <=1) %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  geom_hline(yintercept = 1, col="black") +
  labs(title = "Metrica MAE Modelo Lineal")
```

Arbol.

```
tree_mae=inner_join(mae_tree,mae_tree3,by="circuitId") %>%
  rename(
    mae_tree=.estimate.x,
    mae_tree3=.estimate.y
  )
tree_mae= tree_mae %>%
```

```
  gather(`mae_tree`, `mae_tree3`, key = "mae", value = "estimacion")
tree_mae$mae=as.factor(tree_mae$mae)
```

```
tree_mae %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE Arbol")
```

```
tree_mae_na=inner_join(mae_tree_na,mae_tree_na2,by="circuitId") %>%
  rename(
    mae_tree=.estimate.x,
    mae_tree3=.estimate.y
  )
tree_mae_na= tree_mae_na %>%
  gather(`mae_tree`, `mae_tree3`, key = "mae", value = "estimacion")
tree_mae_na$mae=as.factor(tree_mae_na$mae)
```

```
tree_mae_na %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE Arbol")
```

Random forest.

```
forest_mae=inner_join(mae_forest,mae_forest3,by="circuitId") %>%
  rename(
    mae_forest=.estimate.x,
    mae_forest2=.estimate.y
  )
forest_mae= forest_mae %>%
  gather(`mae_forest`, `mae_forest2`, key = "mae", value = "estimacion")
forest_mae$mae=as.factor(forest_mae$mae)
```

```
forest_mae %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE Random Forest")
```

```
forest_mae_na=inner_join(mae_forest_na,mae_forest_na2,by="circuitId") %>%
  rename(
    mae_forest=.estimate.x,
    mae_forest2=.estimate.y
  )
forest_mae_na= forest_mae_na %>%
  gather(`mae_forest`, `mae_forest2`, key = "mae", value = "estimacion")
forest_mae_na$mae=as.factor(forest_mae_na$mae)
```

```
forest_mae_na %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE Random Forest")
```

Red Neuronal.

```
rn_mae_na=inner_join(mae_rn_na,mae_rn_na2,by="circuitId") %>%
  rename(
    mae_rn=.estimate.x,
    mae_rn2=.estimate.y
```

```
  )
rn_mae_na=rn_mae_na %>%
  gather(`mae_rn`, `mae_rn2`, key = "mae", value = "estimacion")
rn_mae_na$mae=as.factor(rn_mae_na$mae)
```

```
rn_mae_na %>%  ggplot()+
  geom_point(mapping = aes(x=circuitId,y=estimacion,colour=mae),alpha=1/2)+
  labs(title = "Metrica MAE Red Neuronal")
```

Predicciones.

Como hemos visto,los modelos con las variables de la segunda prueba, predicen mejor. Por lo que vamos a realizar una serie de predicciones con las carreras que ya han tenido lugar en la temporada 22. La primera carrera de ésta temporada (round = 1) fue en el circuito de Bahrain (circuitId=3) En esta carrera corrieron un total de 20 pilotos. Veamos cómo predice la posición de varios pilotos.

CL

- Charles Leclerc (driverId= 844) : escuderia ferrari (constructorId=6) : grid=1 Vamos a crear el conjunto de variables objetivo correspondientes a éste constructor.

```
cl22= conjunto3_id %>%
  filter(circuitId==3 & driverId==844 & constructorId==6) %>%
  select(-c(circuitId,driverId,
            constructorId,grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=1,round=1)

cl22=tibble(cl22)
```

Modelo lineal

```
pred_cl22= nc3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22)
pred_cl22$.pred
```

La posición relativa para éste constructor en éste modelo es 0.326. Veamos cúal es la posición absoluta que le corresponde: posicion relativa = poscion abs - 1 / num pilotos

```
posabs_cl22= 20 * pred_cl22$.pred +1
posabs_cl22
```

Arbol de regresion

```
apred_cl22= nc2_tree3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22)
apred_cl22$.pred
```

```
aposabs_cl22= 20 * apred_cl22$.pred +1
aposabs_cl22
```

Random forest.

```
rfpred_cl22= nc2_forest3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22)
rfpred_cl22$.pred
```

```
rfposabs_cl22= 20 * rfpred_cl22$.pred +1
rfposabs_cl22
```

Red neuronal.

```
rnpred_cl22= nc2_rn3 %>% filter(circuitId==3) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=cl22)
rnpred_cl22$.pred
```

```
rnposabs_cl22= 20 * rnpred_cl22$.pred +1
rnposabs_cl22
```

LH

- Hamilton (driverId= 844) : escuderia Mercedes (constructorId=131) : grid=5 Vamos a crear el conjunto de variables objetivo correspondientes a éste constructor.

```
lh22= conjunto3_id %>%
  filter(circuitId==3 & driverId==1 & constructorId==131) %>%
  select(-c(circuitId,driverId,
              constructorId,grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=5,round=1)

lh22=tibble(lh22)
```

Modelo lineal

```
pred_lh22= nc3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=lh22)
pred_lh22$.pred
```

```
posabs_lh22= 20 * pred_lh22$.pred +1
posabs_lh22
```

Arbol de regresion

```
apred_lh22= nc2_tree3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=lh22)
apred_lh22$.pred
```

```
aposabs_lh22= 20 * apred_lh22$.pred +1
aposabs_lh22
```

Random forest.

```
rfpred_lh22= nc2_forest3 %>%
  filter(circuitId==3) %>%
```

```
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=lh22)
rfpred_lh22$.pred
```

```
rfposabs_lh22= 20 * rfpred_lh22$.pred +1
rfposabs_lh22
```

Red neuronal.

```
rnpred_lh22= nc2_rn3 %>% filter(circuitId==3) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=lh22)
rnpred_lh22$.pred
```

```
rnposabs_lh22= 20 * rnpred_lh22$.pred +1
rnposabs_lh22
```

FA

- Fernando Alonso.

```
fa22= conjunto3_id %>%
  filter(circuitId==3 & driverId==4 & constructorId==214) %>%
  select(-c(circuitId,driverId,constructorId,
                  grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=8,round=1)

fa22=tibble(fa22)
```

Modelo lineal

```
pred_fa22= nc3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=fa22)
pred_fa22$.pred
```

```
posabs_fa22= 20 * pred_fa22$.pred +1
posabs_fa22
```

Arbol de regresion

```
apred_fa22= nc2_tree3 %>% filter(circuitId==3) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=fa22)
apred_fa22$.pred
```

```
aposabs_fa22= 20 * apred_fa22$.pred +1
aposabs_fa22
```

Random forest.

```
rfpred_fa22= nc2_forest3 %>%
  filter(circuitId==3) %>%
```

```
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=fa22)
rfpred_fa22$.pred
```

```
rfposabs_fa22= 20 * rfpred_fa22$.pred +1
rfposabs_fa22
```

Red neuronal.

```
rnpred_fa22= nc2_rn3 %>% filter(circuitId==3) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=fa22)
rnpred_fa22$.pred
```

```
rnposabs_fa22= 20 * rnpred_fa22$.pred +1
rnposabs_fa22
```

Anexo E

Predicciones temporada 2022.

En este último anexo vamos a predecir la carrera de la temporada 2022, de la cual ya no tenemos información en la base de datos. Para ello, en primer lugar, modificamos los conjuntos de datos que teníamos para las dos pruebas, añadiendo en estos los datos conocidos de las tres primeras carreras de la temporada 2022. A continuación, creamos los modelos con los conjuntos de datos definidas anteriormente con las técnicas que ya hemos mencionado, realizando a su vez comparaciones en cada modelo según la técnica. Por último hacemos las predicciones de la cuarta carrera.

Predicciones temporada 2022

Vamos a predecir la carrera, de la cual ya no tenemos información en la base de datos. En primer lugar, leo los datos de las carreras de la temporada 2022 de las que hay información en la base de datos

```
resultados22=read.csv("datosk22/results.csv",header = TRUE,dec = ".")
resultados22$position=parse_integer(resultados22$position )
boxes22=read.csv("datosk22/pit_stops.csv",header = TRUE,dec = ".")
resul_construc22=read.csv("datosk22/constructor_results.csv",header = TRUE)
tiempos_vuelta22=read.csv("datosk22/lap_times.csv",header = TRUE,dec=".")
```

```
rc22 = left_join(resultados22,carreras,by="raceId") %>% select(-url)
```

```
datos_m22 =  rc22 %>%  filter(name !="Indianapolis 500" & grid>0)
```

Num pilotos por carrera

```
npilo_xgp22=datos_m22 %>% group_by(raceId,year) %>%  count() %>% arrange(desc(n))
npilo_xgp22 %>% filter(year==2022)
```

Posicion relativa

```
datos_m22 =  left_join(datos_m22,npilo_xgp22,by=c("raceId","year")) %>%
 mutate(pilo_delante=position-1)

datos_m22= datos_m22 %>%  mutate(pos_relativa=pilo_delante/n) %>%
  mutate_at("pos_relativa",~replace(., is.na(.), 1))
datos_m22 = datos_m22 %>%  filter(pos_relativa <= 1)
```

Número de veces que se ha corrido en cada circuito.

Con las carreras ya disputadas en ésta temporada

```
nv_circuitos22=carreras %>%
        filter(raceId <=1076) %>%
                    group_by(circuitId) %>%
                     summarise(totalv=n())
```

Número de veces que cada piloto ha corrido en cada circuito.

Según el constructor

```
nv_pilo_cc22=datos_m22 %>%  group_by(driverId,circuitId,constructorId) %>%
                          summarise(totalc=n_distinct(raceId))
```

Independientemente del constructor

Numero total de veces que un piloto ha corrido en un circuito.

```
nv_pilo_c22= nv_pilo_cc %>% group_by(driverId,circuitId) %>%
  summarise(tveces=sum(totalc))
```

Media de paradas.

```
nboxes22 =inner_join(datos_m22,boxes22,by=c("raceId","driverId")) %>%
  group_by(raceId,driverId,constructorId) %>%
  summarise(nstop=n())
nboxes22 = left_join(nboxes22,GP2,by=c("raceId"))
```

Ahora vamos a calcular la media de paradas que cada equipo hace en cada circuito, aunque en la realizar el número de paradas es un número entero, en este caso no vamos a redondear la media obtenida para tener una aproximación más exacta del número de paradas que pueden realizar cada equipo en cada circuito.

```
n_stop22 = nboxes22 %>% group_by(circuitId,constructorId) %>%
  summarise(stop_mean= mean(nstop))
```

Media tiempo en paradas

Voy a hacer la media de tiempo (en milisegundos) que cada equipo ha tardado en parada en cada GP.

En primer lugar, vamos a calcular el tiempo medio empleado por cada equipo con cada piloto en una parada en cada carrera

```
time_boxes22 =inner_join(datos_m22,boxes22,by=c("raceId","driverId"))
time_boxes22 = time_boxes22 %>% group_by(raceId,driverId,constructorId) %>%
  summarise(suma_t=sum(milliseconds.y))
t_stop22=inner_join(time_boxes22,nboxes22,by=c("raceId","driverId","constructorId"))
t_stop22=t_stop22 %>% mutate(stop_tm=suma_t/nstop) %>%
  select(circuitId,driverId,constructorId,stop_tm)
t_stop22=t_stop22[,-1]
```

Ahora calculo, el tiempo medio por parada en cada circuito de cada equipo.

```
t_stop22 = t_stop22 %>% group_by(constructorId,circuitId) %>%
  summarise(timestop_mean=mean(stop_tm))
```

Proporción de accidentes por pilotos y circuitos

Filtro por los estados de accidente (statusId=3) y cuento cuántos ha tenido cada piloto en cada circuito.

```
acc22=datos_m22 %>% filter(statusId==3 )
acc22=acc22 %>% group_by(driverId,circuitId) %>% summarise(totalac=n())
```

Hago la proporción del número de accidentes de cada piloto en cada circuito entre el número total de veces que cada piloto ha corrido en cada circuito.

```
pacc22=left_join(acc22,nv_pilo_c22,by=c("driverId","circuitId"))%>%
  mutate(pAccidentes=totalac/tveces)
pacc22=pacc22%>% select(driverId,circuitId,pAccidentes)
```

Tiempo medio piloto

Tiempo medio piloto

Vamos a calcular el histórico del piloto: tiempo medio en vuelta en general del piloto.

```
tiempo_pilo22 = tiempos_vuelta22 %>% group_by(driverId) %>%
  summarise(tm_p=mean(milliseconds))
```

Vamos a calcular la media general de todos los pilotos, para poder imputar los valores de los pilotos de los que no tengamos información.

```
tm_pgeneral22 =  tiempo_pilo22 %>% summarise(tm_gen=mean(tm_p))
```

Tiempo medio piloto circuito

En primer lugar, voy a calcular, el tiempo medio por vuelta de cada piloto en cada carrera.

```
tiempo_vmean22= tiempos_vuelta22 %>% group_by(raceId,driverId) %>%
  summarise(t_vm=mean(milliseconds))
tiempo_vmean22 =left_join(tiempo_vmean22,GP2,by="raceId")
```

Vamos a calcular el tiempo medio por vuelta que lleva cada piloto en cada circuito (sin tener en cuenta el constructor)

```
tiempo_pilocir22=tiempo_vmean22 %>% group_by(driverId,circuitId) %>%
  summarise(tm_pci=mean(t_vm))
```

Vamos a calcular la media de las medias

```
tm_pcgeneral22 = tiempo_pilocir22 %>% group_by(circuitId) %>% summarise(tm_pcgen= mean(tm_pci))
tm_pcgeneral22 = tm_pcgeneral22  %>% summarise(tm_pcgen= mean(tm_pcgen))
```

Tiempo medio piloto circuito constructor

Calculamos por último, el tiempo medio por vuelta que lleva cada piloto en cada circuito según el constructor.

```
cirpiloeq22= datos_m22 %>% select(raceId,driverId,constructorId)
tiempo_pilocc22= left_join(tiempos_vuelta22,cirpiloeq22,by=c("raceId","driverId")) %>%
  group_by(raceId,driverId,constructorId) %>%
  summarise(tm_pcc=mean(milliseconds))

tiempo_pilocc22= left_join(tiempo_pilocc22,GP2,by="raceId") %>%
  group_by(circuitId,driverId,constructorId) %>%
  summarise(tmean_pcc=mean(tm_pcc))
```

```
tm_pccgeneral22 = tiempo_pilocc22 %>% group_by(circuitId,constructorId) %>%
  summarise(tm_pccgen= mean(tmean_pcc))
tm_pccgeneral22 = tm_pccgeneral22  %>% summarise(tm_pccgen= mean(tm_pccgen))
tm_pccgeneral22 =tm_pccgeneral22 %>% summarise(tm_pccgen= mean(tm_pccgen))
```

PILOTOS

Numero total de carreras

Numero total de grandes premios que ha disputado cada piloto.

```
nc_pilo22 = nv_pilo_cc22 %>%  group_by(driverId) %>%
  summarise(tc= sum(totalc))
```

Posición media relativa piloto.

Posición media relativa piloto circuito constructor.

Posicion media en la que quedó en cada circuito según el constructor.

```
posm_piloc22 =  datos_m22 %>%  group_by(circuitId,driverId,constructorId) %>%
  summarise(pos_mean= mean(pos_relativa))
```

Posición media relativa piloto circuito.

```
posm_pilo22 = datos_m22 %>%  group_by(circuitId,driverId) %>%
  summarise(posic_m= mean(pos_relativa))
```

Probabilidad de hacer podium piloto.

Probabilidad hacer podium piloto circuito constructor

Según el constructor.

En primer lugar, vamos a calcular el número de podiums que ha hecho cada piloto con cada constructor en cada circuito.

```
pod_prob22 =  datos_m22 %>%  filter(position <= 3) %>%
  group_by(circuitId,driverId,constructorId) %>%
  summarise(pod_nc=n())
```

Ahora vemos el número de veces que ha corrido en cada circuito con cada constructor y hacemos la proporción.

```
pod_prob22=left_join(pod_prob22,nv_pilo_cc22,by=c("circuitId","driverId","constructorId"))

pod_prob22= pod_prob22 %>% mutate(prob=pod_nc/totalc) %>% select(-c(pod_nc,totalc))
```

Probabilidad de hacer podium piloto circuito

En primer lugar, calculamos el numero de podiums que ha conseguido cada piloto en cada circuito.

```
pod_prob2_22 =  datos_m22 %>%  filter(position <= 3) %>%
  group_by(circuitId,driverId) %>%
  summarise(pod_nc2=n())
```

Ahora tomo el número de veces que ha corrido en cada circuito y hacemos la proporción.

```
pod_prob2_22 = left_join(pod_prob2_22,nv_pilo_c22,by=c("circuitId","driverId"))
pod_prob2_22=pod_prob2_22 %>%
  mutate(probc= pod_nc2/tveces) %>%
  select(-c(pod_nc2,tveces))
```

Probabilidad de hacer podium piloto.

Calculamos el número de podiums que ha hecho un piloto en lo que lleva

```
pod_prob3_22 =  datos_m22 %>%  filter(position <= 3) %>%
  group_by(driverId) %>%
  summarise(pod_p=n())
```

Ahora tomo el número de carreras que ha disputado y hago la proporción.

```
pod_prob3_22 = left_join(pod_prob3_22,nc_pilo22,by=c("driverId"))
pod_prob3_22=pod_prob3_22 %>%
```

```
  mutate(prob_p= pod_p/tc) %>%
  select(-c(pod_p,tc))
```

CONSTRUCTORES.

```
const_c22= left_join(resul_construc22,carreras,by="raceId") %>%
  select(circuitId,constructorId,points,year)
```

Numero de veces que ha corrido en el circuito

```
nv_circons22=datos_m22%>% group_by(circuitId,constructorId) %>%
  distinct(raceId) %>% count()
```

Puntuación media conseguida en cada circuito.

```
puntm_cc22 = const_c22 %>% group_by(circuitId,constructorId) %>%
  summarise(point_mean  = mean(points))
```

Puntuación media conseguida

En primer lugar calculo número de años que lleva cada escudería participando en F1.

```
yeas_const22 = const_c22 %>% group_by(constructorId) %>%
  summarise(n_years= n_distinct(year))
```

A continuación calculo el total de puntos que ha conseguido en todos esos años, y hago la media.

```
punt_totalm22= const_c22 %>%  group_by(constructorId) %>%
  summarise(punt_mean= sum(points))

punt_totalm22 = left_join(punt_totalm22,yeas_const22,by="constructorId")
punt_totalm22 = punt_totalm22%>%
  mutate(points_m = punt_mean/n_years) %>%
  select(-c(punt_mean,n_years))
```

```
conjunto22=datos_m22 %>%  select(circuitId,driverId,constructorId,grid,pos_relativa,round)
conjunto22=left_join(conjunto22,nc_pilo22,
                     by="driverId")
conjunto22=left_join(conjunto22,nv_pilo_cc22,
                     by=c("driverId","circuitId","constructorId"))
conjunto22=left_join(conjunto22,nv_pilo_c22,
                     by=c("driverId","circuitId"))
conjunto22 = left_join(conjunto22,posm_pilo22,
                       by=c("circuitId","driverId"))
conjunto22=left_join(conjunto22,posm_piloc22,
                     by=c("circuitId","driverId","constructorId"))
conjunto22 = left_join(conjunto22,pod_prob22,
                       by=c("circuitId","driverId","constructorId"))
conjunto22=left_join(conjunto22,pod_prob2_22,
                     by=c("circuitId","driverId"))
conjunto22=left_join(conjunto22,pod_prob3_22,
                     by="driverId")
```

```
conjunto22 = left_join(conjunto22,pacc22,
                       by=c("circuitId","driverId")) %>%
  mutate_at("pAccidentes",~replace(.,is.na(.),0))
conjunto22 =left_join(conjunto22,tiempo_pilo22,
                      by="driverId")
conjunto22 = left_join(conjunto22,tiempo_pilocir22,
                       by=c("circuitId","driverId"))
conjunto22 = left_join(conjunto22,tiempo_pilocc22,
                       by=c("circuitId","constructorId","driverId"))
```

```
conjunto22 %>%  select(prob,probc,prob_p,tm_p,tm_pci,tmean_pcc) %>% summary
```

```
conjunto22 = conjunto22 %>% mutate_at(c("prob","probc","prob_p"),~replace(.,is.na(.),0))
```

Imputamos los NA de los tiempos por la media.

```
conjunto22 = conjunto22 %>%
  mutate_at("tm_p",~replace(.,is.na(.),tm_pgeneral22$tm_gen))
conjunto22 = conjunto22 %>%
  mutate_at("tm_pci",~replace(.,is.na(.),tm_pcgeneral22$tm_pcgen))
conjunto22 = conjunto22 %>%
  mutate_at("tmean_pcc",~replace(.,is.na(.),tm_pccgeneral22$tm_pccgen))
```

```
conjunto22 %>% summary
```

```
conjunto22 = left_join(conjunto22,nv_circons22,
                       by=c("circuitId","constructorId"))
conjunto22 = left_join(conjunto22,puntm_cc22,
                       by=c("circuitId","constructorId") )
conjunto22 = left_join(conjunto22,punt_totalm22,by="constructorId")
conjunto22 = conjunto22 %>% mutate_at(c("point_mean","points_m"),~replace(.,is.na(.),0))
conjunto22 = left_join(conjunto22,n_stop22,
                       by=c("circuitId","constructorId"))
conjunto22= left_join(conjunto22,t_stop22,by=c("circuitId","constructorId"))
conjunto22 %>% select (stop_mean,timestop_mean) %>% summary
```

Prediccion circuito 21

Vamos a hacer la predicción de la posición final de los pilotos que salieron los 5 primeros. ## Max Verstappen

```
mv22_em= conjunto22 %>% filter(circuitId==21 & driverId==830 & constructorId==9) %>%
  select(-c(circuitId,driverId,
                constructorId,grid,pos_relativa,round)) %>%
  distinct() %>%
  mutate(grid=1,round=4)

mv22_em=tibble(mv22_em)
```

ML

```
pred_mv22_em= nc3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=mv22_em)
pred_mv22_em$.pred
```

```
posabs_mv22_em= 20 * pred_mv22_em$.pred +1
posabs_mv22_em
```

AR

```
apred_mv22_em= nc2_tree3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=mv22_em)
apred_mv22_em$.pred
```

```
aposabs_mv22_em= 20 * apred_mv22_em$.pred +1
aposabs_mv22_em
```

Random forest.

```
rfpred_mv22_em= nc2_forest3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=mv22_em)
rfpred_mv22_em$.pred
```

```
rfposabs_mv22_em= 20 * rfpred_mv22_em$.pred +1
rfposabs_mv22_em
```

```
rnpred_mv22_em= nc2_rn3%>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=mv22_em)
rnpred_mv22_em$.pred
```

```
rnposabs_mv22_em= 20 * rnpred_mv22_em$.pred +1
rnposabs_mv22_em
```

Charles Leclerc

```
cl22_em= conjunto22 %>% filter(circuitId==21 & driverId==844 & constructorId==6) %>%
  select(-c(circuitId,driverId,
      constructorId,grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=2,round=4)

cl22_em=tibble(cl22_em)
```

ML

```
pred_cl22_em= nc3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22_em)
pred_cl22_em$.pred
```

```
posabs_cl22_em= 20 * pred_cl22_em$.pred +1
posabs_cl22_em
```

AR

```
apred_cl22_em= nc2_tree3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22_em)
apred_cl22_em$.pred
```

```
aposabs_cl22_em= 20 * apred_cl22_em$.pred +1
aposabs_cl22_em
```

Random forest.

```
rfpred_cl22_em= nc2_forest3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22_em)
rfpred_cl22_em$.pred
```

```
rfposabs_cl22_em= 20 * rfpred_cl22_em$.pred +1
rfposabs_cl22_em
```

RN

```
rnpred_cl22_em= nc2_rn3%>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=cl22_em)
rnpred_cl22_em$.pred
```

```
rnposabs_cl22_em= 20 * rnpred_cl22_em$.pred +1
rnposabs_cl22_em
```

Sergio Pérez

```
sp22_em= conjunto22 %>% filter(circuitId==21 & driverId==815 & constructorId==9) %>%
  select(-c(circuitId,driverId,
            constructorId,grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=3,round=4)

sp22_em=tibble(sp22_em)
```

ML

```
pred_sp22_em= nc3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=sp22_em)
pred_sp22_em$.pred
```

```
posabs_sp22_em= 20 * pred_sp22_em$.pred +1
posabs_sp22_em
```

AR

```
apred_sp22_em= nc2_tree3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=sp22_em)
apred_sp22_em$.pred
```

```
aposabs_sp22_em= 20 * apred_sp22_em$.pred +1
aposabs_sp22_em
```

RF

```
rfpred_sp22_em= nc2_forest3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=sp22_em)
rfpred_sp22_em$.pred
```

```
rfposabs_sp22_em= 20 * rfpred_sp22_em$.pred +1
rfposabs_sp22_em
```

RN

```
rnpred_sp22_em= nc2_rn3%>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>% predict(new_data=sp22_em)
rnpred_sp22_em$.pred
```

```
rnposabs_sp22_em= 20 * rnpred_sp22_em$.pred +1
rnposabs_sp22_em
```

Carlos Sáinz

```
cs22_em= conjunto22 %>% filter(circuitId==21 & driverId==832 & constructorId==6) %>%
  select(-c(circuitId,driverId,
          constructorId,grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=4,round=4)

cs22_em=tibble(cs22_em)
```

ML

```
pred_cs22_em= nc3 %>% filter(circuitId==21) %>% unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=cs22_em)
pred_cs22_em$.pred
```

```
posabs_cs22_em= 20 * pred_cs22_em$.pred +1
posabs_cs22_em
```

AR

```
apred_cs22_em= nc2_tree3 %>% filter(circuitId==21) %>%
  unnest(model) %>% pluck(".workflow",1) %>%
  predict(new_data=cs22_em)
apred_cs22_em$.pred
```

```
aposabs_cs22_em= 20 * apred_cs22_em$.pred +1
aposabs_cs22_em
```

RF

```
rfpred_cs22_em= nc2_forest3 %>% filter(circuitId==21) %>%
  unnest(model) %>% pluck(".workflow",1) %>%
  predict(new_data=cs22_em)
rfpred_cs22_em$.pred
```

```
rfposabs_cs22_em= 20 * rfpred_cs22_em$.pred +1
rfposabs_cs22_em
```

RN

```
rnpred_cs22_em= nc2_rn3%>% filter(circuitId==21) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=cs22_em)
rnpred_cs22_em$.pred
```

```
rnposabs_cs22_em= 20 * rnpred_cs22_em$.pred +1
rnposabs_cs22_em
```

Lando Norris

```
ln22_em= conjunto22 %>% filter(circuitId==21 & driverId==846 & constructorId==1) %>%
  select(-c(circuitId,driverId,
            constructorId,grid,pos_relativa,round)) %>%
  distinct() %>% mutate(grid=5,round=4)

ln22_em=tibble(ln22_em)
```

ML

```
pred_ln22_em= nc3 %>% filter(circuitId==21) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=ln22_em)
pred_ln22_em$.pred
```

```
posabs_ln22_em= 20 * pred_ln22_em$.pred +1
posabs_ln22_em
```

AR

```
apred_ln22_em= nc2_tree3 %>% filter(circuitId==21) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=ln22_em)
apred_ln22_em$.pred
```

```
aposabs_ln22_em= 20 * apred_ln22_em$.pred +1
aposabs_ln22_em
```

RF

```
rfpred_ln22_em= nc2_forest3 %>% filter(circuitId==21) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=ln22_em)
rfpred_ln22_em$.pred
```

```
rfposabs_ln22_em= 20 * rfpred_ln22_em$.pred +1
rfposabs_ln22_em
```

RN

```
rnpred_ln22_em= nc2_rn3%>% filter(circuitId==21) %>%
  unnest(model) %>%
  pluck(".workflow",1) %>%
  predict(new_data=ln22_em)
rnpred_ln22_em$.pred
```

```
rnposabs_ln22_em= 20 * rnpred_ln22_em$.pred +1
rnposabs_ln22_em
```