



UNIVERSIDAD DE GRANADA

PRÁCTICA 3 INTELIGENCIA DE NEGOCIO

---

# Competición en Kaggle

---

David Alberto Martín Vela

Grupo 2 de prácticas  
davidmv1996@correo.ugr.es  
Doble Grado Ingeniería Informática y Matemáticas

Curso 2020-2021

Leaderboard final de la competición, **posición 10** con una score de **0.80586**.

Overview

Data

Notebooks

Discussion

Leaderboard

Rules

Team

My Submissions

Late Submission

Public Leaderboard

Private Leaderboard

This leaderboard is calculated with all of the test data.

Raw Data

Refresh

#	Team Name	Notebook	Team Members	Score ?	Entries	Last
1	JuanHeliosGarcía			0.83002	15	3d
2	PATRICIA CORDOBA 77145053			0.82830	13	3d
3	José Alberto García 26513007X			0.82484	43	1d
4	Alvaro de Rada 49108766V			0.81622	14	12h
5	DAVID CABEZAS 20079906			0.81622	34	2d
6	OctavioTorres			0.81622	23	12h
7	AlejandroAlonso75577394S			0.81363	10	1d
8	Mikhail Raudin 531101855			0.80845	11	11h
9	Javier Rodríguez 78306251Z			0.80759	12	12h
10	David Martin 75931868J			0.80586	28	21h
<div><div>Your Best Entry ↑</div><div><div>Your submission scored 0.81190, which is an improvement of your previous score of 0.80586. Great job!</div><div><div>Tweet this!</div></div></div></div>						
11	JuanCarlosGonQu			0.79982	54	19h
12	Pedro Jiménez 76592485R			0.79810	35	16h

Podríamos haber conseguido más si hubieramos hecho subidas de ficheros con más cabeza ya que después redactar las documentación probando alguna cosa más (late submissions) conseguimos scores de hasta **0.81190**.

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
mis_resultados_catboost_1.csv	11 hours ago	0 seconds	0 seconds	0.81190
Complete				
<a href="#">Jump to your position on the leaderboard</a> ▼				

# Contents

<b>Introducción</b>	<b>1</b>
Descripción del problema . . . . .	1
Descripción de los atributos . . . . .	1
Ficheros . . . . .	2
<b>Competición</b>	<b>3</b>
Exploratory data analysis . . . . .	3
Preprocesamientos . . . . .	12
Missing values . . . . .	12
Etiquetado . . . . .	13
Escalado . . . . .	14
Skewed data y outliers . . . . .	14
Balanceado . . . . .	18
Algoritmos . . . . .	19
Random Forest . . . . .	19
XGBoost . . . . .	21
Catboost . . . . .	23
Red Neuronal . . . . .	24
Tabla . . . . .	30
<b>Referencias</b>	<b>33</b>

# List of Figures

1	Histograma general . . . . .	5
2	Ciudad vs Precio_cat . . . . .	6
3	Mano vs Precio_cat . . . . .	6
4	Asientos vs Precio_cat . . . . .	6
5	Año vs Precio_cat . . . . .	7
6	Distribución Consumo . . . . .	8
7	Distribución Motor CC . . . . .	8
8	Distribución Potencia . . . . .	8
9	Marca . . . . .	9
10	Marca vs Precio_cat . . . . .	10
11	Comparación entre tipo marchas y kilómetros . . . . .	11
12	Comparación entre ciudad y kilómetros . . . . .	11
13	Missing values . . . . .	12
14	Outliers consumo . . . . .	15
15	Outliers Kilometros . . . . .	15
16	Outliers Motor_CC . . . . .	15
17	Outliers Potencia . . . . .	15
18	Outliers . . . . .	15
19	No outliers consumo . . . . .	16
20	No outliers Kilometros . . . . .	16
21	No outliers Motor_CC . . . . .	16
22	No outliers Potencia . . . . .	16
23	No outliers . . . . .	16
24	Skewed data . . . . .	17
25	Skewed data . . . . .	18
26	Features importance . . . . .	21
27	Catboost plot train . . . . .	24
28	Convergencia modelos red neuronal . . . . .	28
29	AUC ROC red neuronal . . . . .	29
30	Subidas a la plataforma Kaggle . . . . .	30

# List of Tables

1	Tabla de puntuaciones . . . . .	32
---	---------------------------------	----

# Introducción

En esta tercera y última práctica de la asignatura Inteligencia de Negocio veremos el uso de métodos avanzados para aprendizaje supervisado en clasificación sobre una competición disponible en *Kaggle*, creada ex-proceso para esta práctica. Se buscará adquirir destrezas para mejorar la capacidad predictiva del modelo mientras nos familiarizamos con una de las plataformas más populares de competición en ciencias de datos.

## Descripción del problema

En este problema existen datos de una serie de coches vendidos, y se han clasificado los coches en 5 categorías de precio. Se desea predecir la categoría del precio del coche, por lo que es un problema de clasificación multiclase.

El conjunto de entrenamiento consta de alrededor de 6000 instancias, y 14 atributos (de los cuales, *id* toma valores únicos y solo sirve para identificar cada ejemplo) con datos categóricos y enteros. Se trata de predecir la variable ordinal **Precio\_cat**, que representa el grado de coste que supone. Hay cinco valores: **1**, representa a los más baratos; **2**, representa aquellos baratos pero menos; **3**, representa a los que están en precio promedio; **4**, representa a los que son más caros que el promedio, y **5**, que representa a los coches más caros. Para medir el rendimiento de nuestros algoritmos, la competición usará la medida de precisión (accuracy), aunque se podrá utilizar otras medidas o criterios para identificar los algoritmos más promedores en la memoria.

## Descripción de los atributos

A continuación se detalla el significado de los distintos atributos:

- Nombre del tipo de coche
- Año del coche
- Kilómetros recorridos
- Combustible del coche (Petrol, Diesel, Electric, LPG y CNG)
- Tipo de marcha del coche (Manual y Automatic).

- 
- Mano si es primera mano (First), Segunda (Second), Tercera (Third) y cuarta o más (Fourth & Above).
  - Consumo en kilómetros por litro (kmpl).
  - MotorCC medido en centímetros cúbicos (CC).
  - Potencia del motor medido (bhp).
  - Asientos
  - Descuento realizado por oferta especial (en porcentaje).

## Ficheros

Para trabajar se dispone de varios ficheros descargables en la página web de la competición:

- train.csv: Fichero con todos los atributos, incluyendo el objetivo a predecir. Es el conjunto de datos que se puede usar para realizar el aprendizaje automático. Además, se deberá de evaluar usando validación cruzada para identificar los algoritmos más prometedores, y realizar el proceso de tuning que se considere. Se deberá de aplicar las técnicas de pre-procesamiento que se consideren interesantes.
- test.csv: Fichero con las instancias a predecir. Posee el mismo formato que train.csv, a excepción del atributo Precio cat, que evidentemente no aparece.
- sample.csv: Fichero con el formato del fichero a someter, que contiene únicamente el id y los valores de Precio cat.

También se ofrece un fichero para cada atributo de cara a facilitar una correcta etiquetación y normalización (que debería de aplicarse por igual a los datos de ambos ficheros)

# Competición

En esta parte de la documentación se explicarán las estrategias seguidas y el progreso que se ha ido desarrollando durante la competición. Para comenzar, haremos un poco de **EDA** para ver nuestros atributos (*Exploratory data analysis*), veremos distintos tipos de **preprocesamiento** que se plantean y describiremos los principales **algoritmos** para la predicción multiclase utilizados, haciendo mención especial en un apartado a una **red neuronal** creada para este problema.

Finalmente mostraremos una **tabla** [1] con el conjunto de planteamientos que han sido más exitosos (ya que la mayoría de subidas de ficheros han sido probando el mismo preprocesamiento pero usando distintos algoritmos), incluyendo la fecha de subida de Kaggle, la posición en el leader en ese momento, el preprocesamiento utilizando, la puntuación de validación cruzada de 5 particiones respecto al conjunto de train.csv<sup>1</sup> y la puntuación obtenida en Kaggle resultante al subir el fichero con la predicción realizada sobre el conjunto test.csv.

Por simplicidad y para hacer la tabla más cómoda visualmente, mostraremos únicamente los planteamientos más interesantes, ya que personalmente opino que no tiene mucho sentido añadir todas las subidas donde algunas son distintas combinaciones para distintos preprocesamientos, si tenemos una subida que considera un conjunto de preprocesamientos y consigue una puntuación mejor que otros aplicados individualmente (por ejemplo, si en un intento hemos tratado con valores perdidos, y en otro intento hemos tratado de la misma manera valores perdidos y además hemos utilizado técnicas para el balanceo, únicamente pondremos el segundo intento ya que el primero es redundante).

## Exploratory data analysis

Todo lo relevante para recrear esta parte visual se encuentra al inicio del notebook **vanilla.ipynb**. Obteniendo información sobre nuestras variables

train.csv	0	id	4747 non-null
RangeIndex: 4819 entries, 0 to 4818		float64	
Data columns (total 14 columns):	1	Nombre	4747 non-null
# Column Non-Null Count Dtype		object	
---	2	Ciudad	4747 non-null

<sup>1</sup>el fichero **train.csv**, a su vez dividido en conjuntos de train y test y realizando validación cruzada de 5 particiones **únicamente** sobre el conjunto de train resultante de dividir el conjunto de train, puede ser un poco confuso, básicamente es que no combinamos train.test



---

3	Anio	4747 non-null	float64	
4	Kilometros	4747 non-null	float64	
5	Combustible	4747 non-null	object	
6	Tipo_marchas	4747 non-null	object	
7	Mano	4747 non-null	object	
8	Consumo	4746 non-null	object	
9	Motor_CC	4718 non-null	object	
10	Potencia	4644 non-null	object	
11	Asientos	4713 non-null	float64	
12	Descuento	659 non-null	float64	
13	Precio_cat	4819 non-null	int64	

dtypes: float64(5), int64(1), object(8)

---

#	Column	Non-Null Count	Dtype
0	id	1159 non-null	int64
1	Nombre	1159 non-null	
2	Ciudad	1159 non-null	
3	Anio	1159 non-null	int64
4	Kilometros	1159 non-null	int64
5	Combustible	1159 non-null	object
6	Tipo_marchas	1159 non-null	object
7	Mano	1159 non-null	object
8	Consumo	1159 non-null	object
9	Motor_CC	1159 non-null	object
10	Potencia	1159 non-null	object
11	Asientos	1159 non-null	float64
12	Descuento	155 non-null	float64

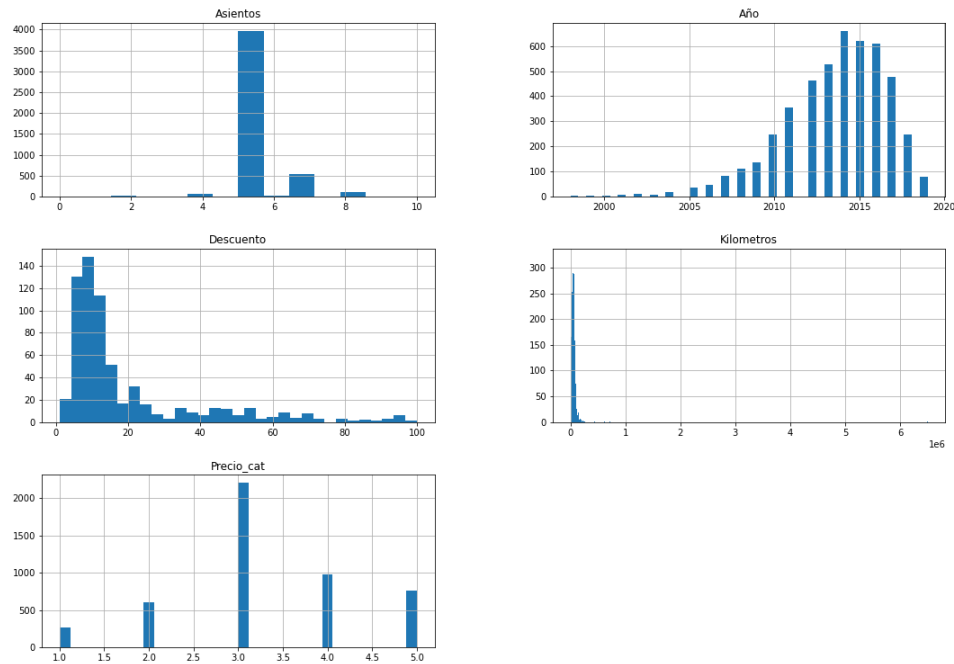
dtypes: float64(2), int64(3), object(8)

---

Hacemos una primera vista a las variables numéricas de nuestro dataset en modo de histograma [1], donde se aprecia la presencia de outliers, por ejemplo en el caso de nuestro atributo **kilómetros** y asimetría en las distribuciones (*skewed data*).

---

**Figure 1: Histograma general**



Vamos a mostrar visualizaciones de algunas variables discretas individualmente y respecto a nuestra variable a predecir **Precio\_cat** en la figura [4], también en la figura [5] vemos algo esperado y es la influencia del atributo Año en la variable a predecir, ya que con el tiempo los coches valen menos.

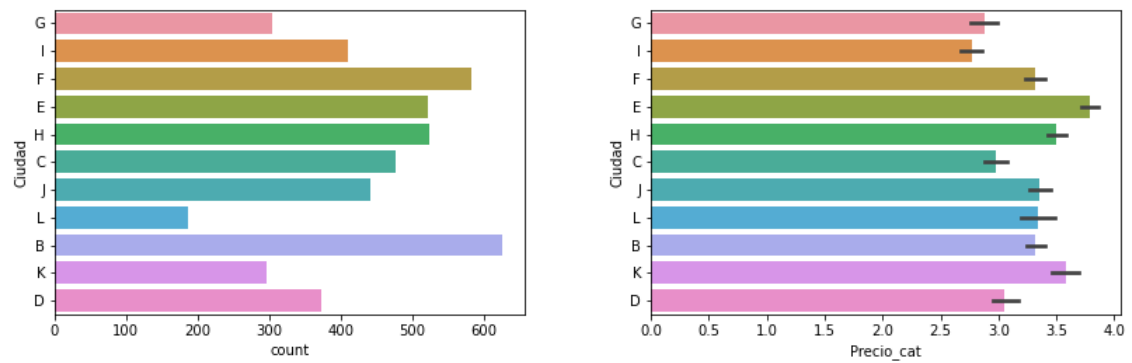


Figure 2: Ciudad vs Precio\_cat

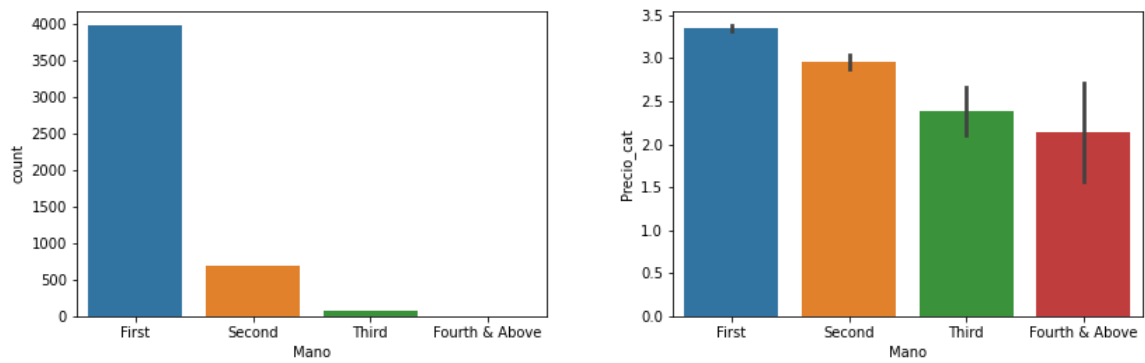


Figure 3: Mano vs Precio\_cat

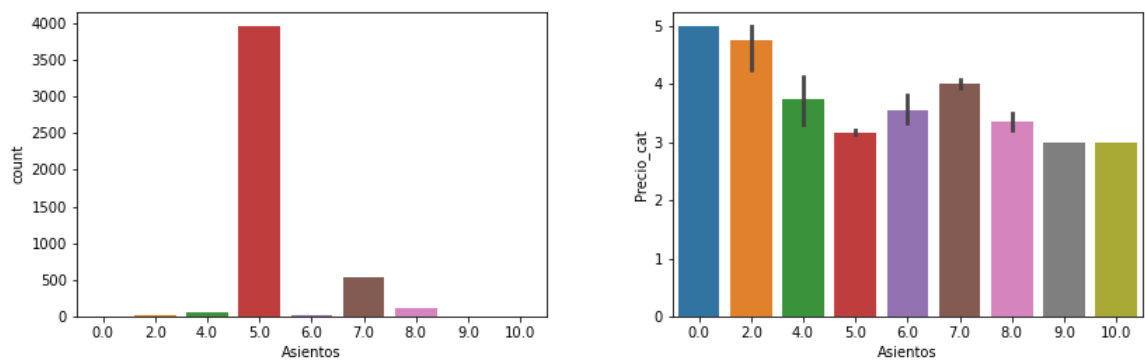
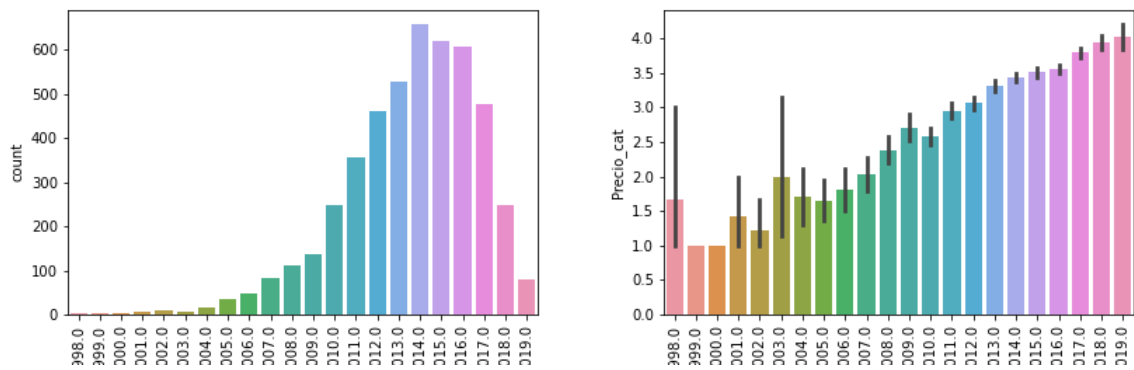


Figure 4: Asientos vs Precio\_cat



**Figure 5:** Año vs Precio\_cat

Destacamos el caso de Asientos, donde hay una gran cantidad de coches con 5 asientos pero a su vez son de los que menos precio valen, esto puede deberse debido a la oferta y demanda.

A su vez, para visualizar también las variables de objetos como Combustible, Consumo etc que tienen un parámetro numérico junto a la medida, preprocesamos estos atributos dejándolo solo el valor numérico para poder visualizarlo, vemos en la figura [8] visualizaciones de estas variables preprocesadas de la manera comentada junto a un boxplot.

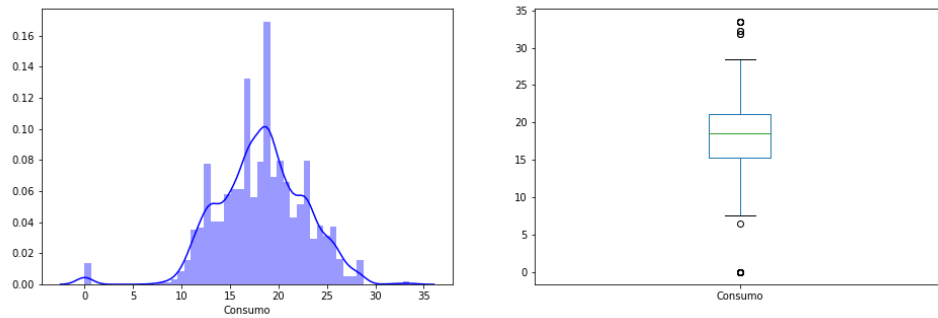


Figure 6: Distribución Consumo

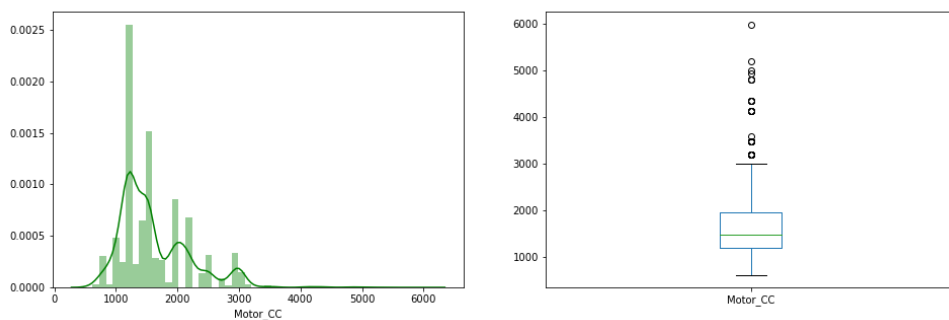


Figure 7: Distribución Motor CC

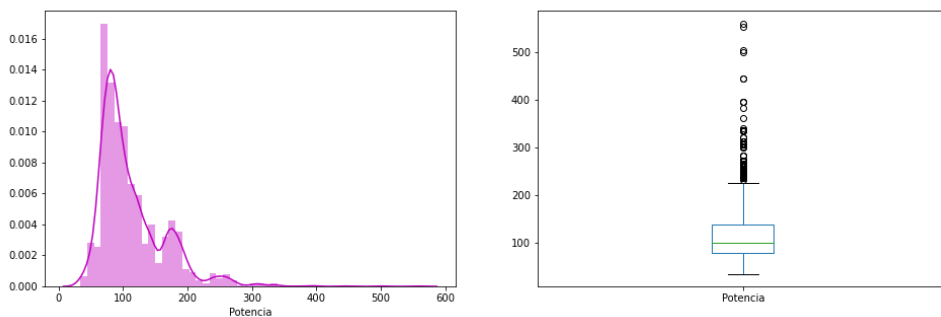
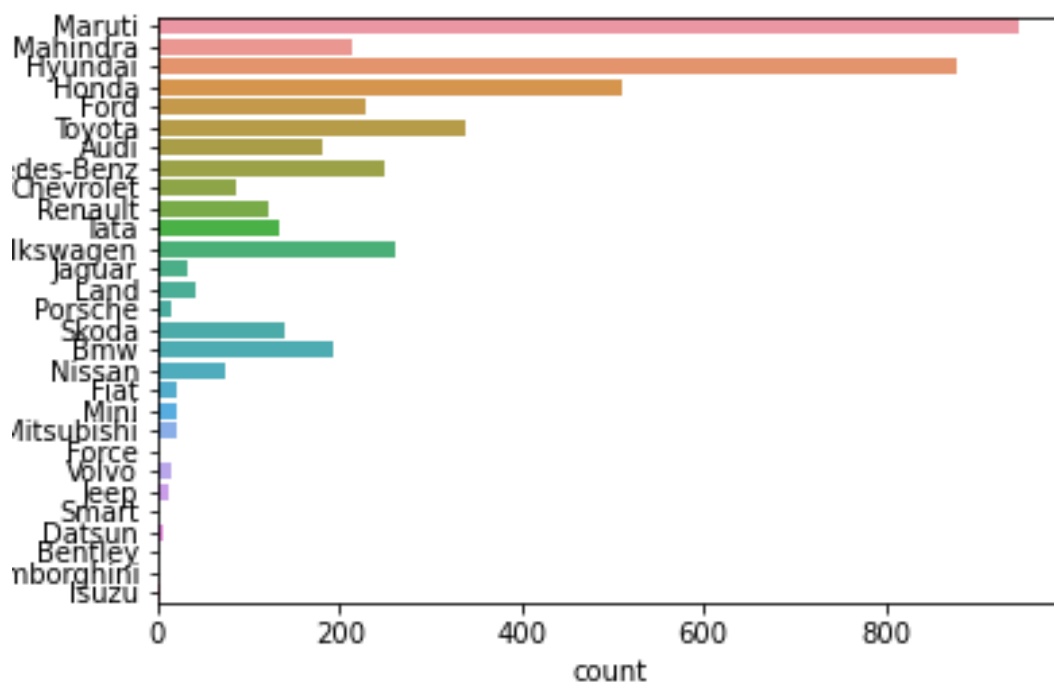


Figure 8: Distribución Potencia

Como en nuestro histograma inicial, se aprecia la presencia de outliers, por ejemplo en el caso de nuestro atributo **kilómetros** y asimetría en las distribuciones (*skewed data*), también en nuestro atributo consumo, hay valores de 0, lo cuál creemos que no es posible que haya coches que consuman 0 litros por kilómetro.

Vamos a crear otro atributo llamado **Company** donde de nuestro atributo **Nombre** extraeremos la marca del coche (nos quedamos con la primera palabra del nombre que corresponde a la marca del coche), visualizamos la relación entre la marca y el precio en la figura [10] y la distribución de valores de marcas en la figura [9].



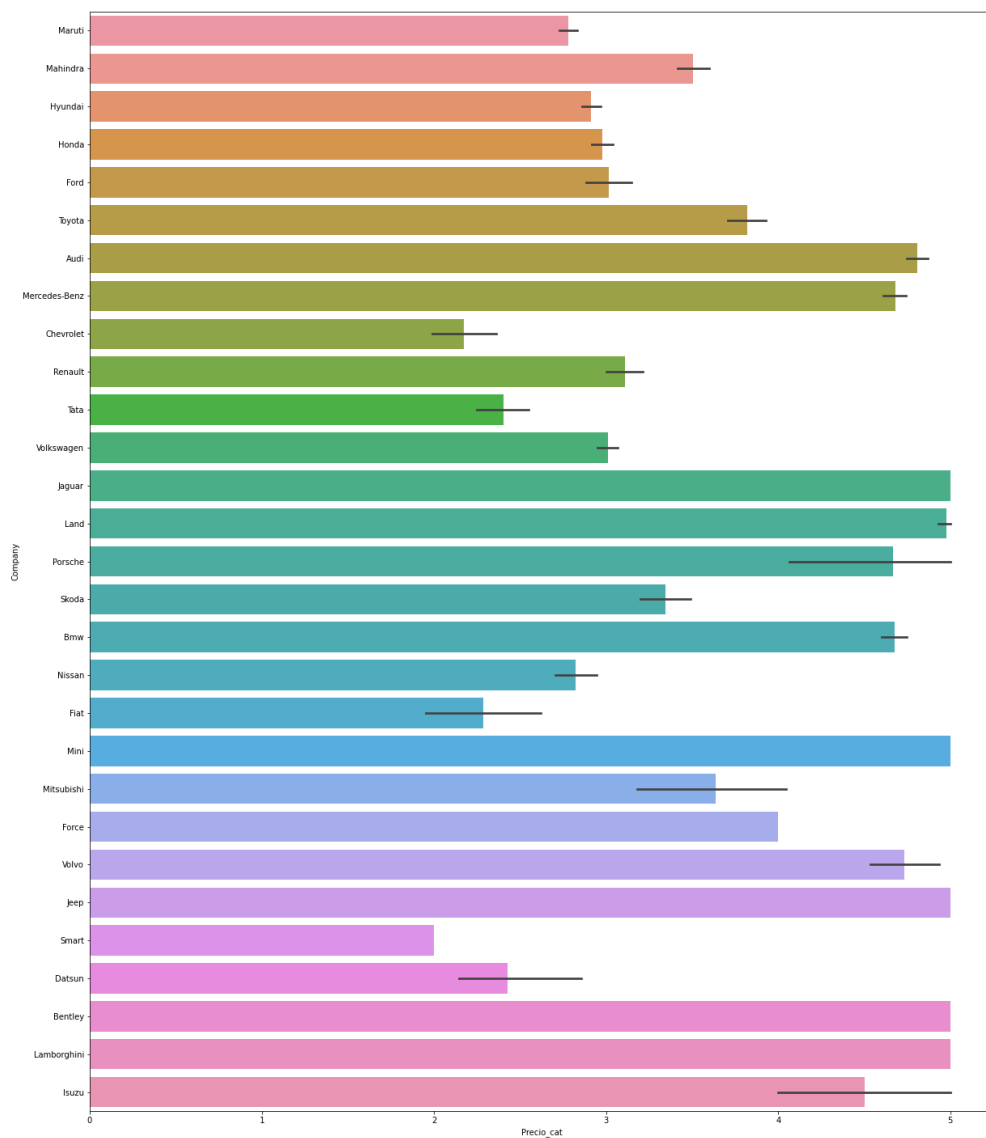
**Figure 9:** Marca

En la creación de la nueva columna Company, cuando separábamos los nombres nos hemos encontrado con algunas irregularidades, por ejemplo la marca Isuzu, estaba tanto en mayúscula como en minúscula, esto lo hemos solucionado con el método `title()` de python, donde hemos unificado estos valores pasándolos a la forma de un String con la primera letra mayúscula, después de esta aclaración, en la figura [10] vemos que la influencia de la marca en el precio tiene sentido, ya que marcas de gama baja suelen valer menos por ejemplo, fiat, comparado con marcas de coches de lujo como por ejemplo, Lamborghini.

---

```
part = car_train['Nombre'].astype(str).str.partition()
car_train['Company'] = part[0].str.title()
#car_train['Car_name'] = part[2]
```

---



**Figure 10: Marca vs Precio\_cat**

Comparamos también entre atributos, por ejemplo entre tipo de marchas y kilómetros en la figura [11] y entre ciudades y kilómetros en la figura [12]

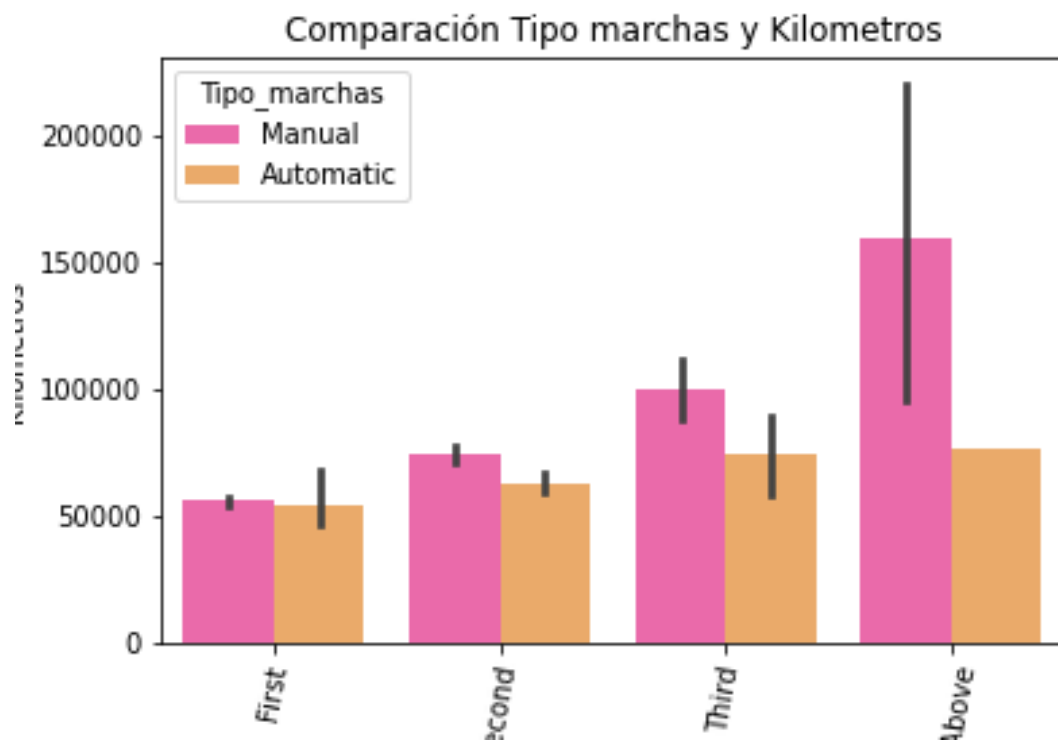


Figure 11: Comparación entre tipo marchas y kilómetros

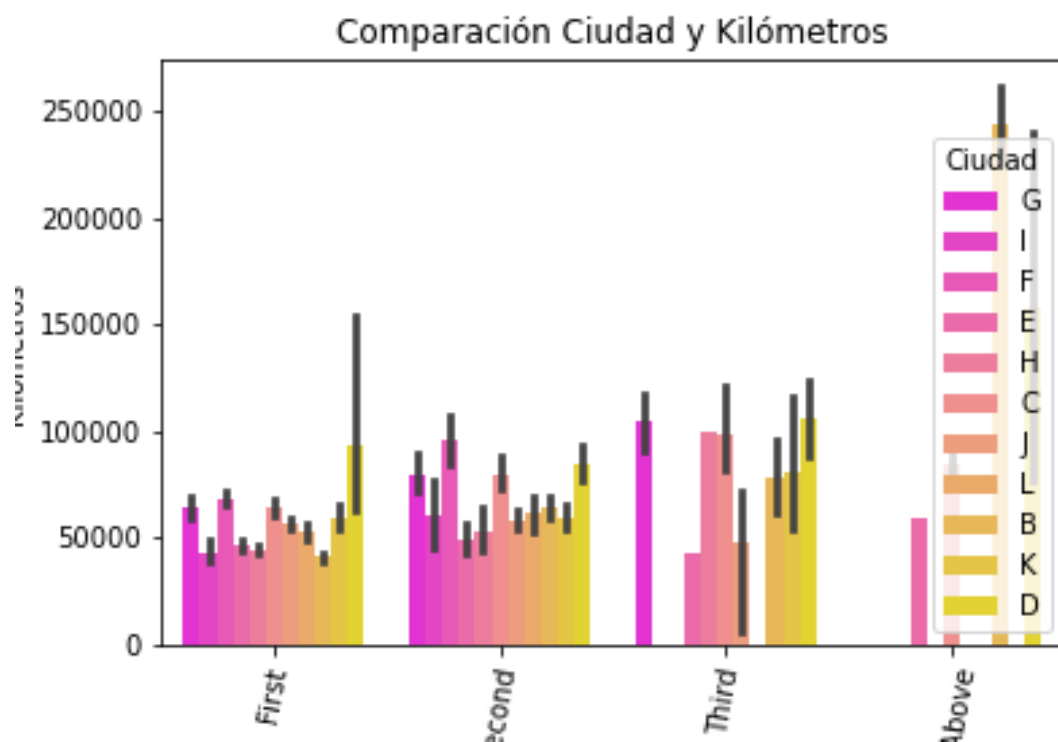


Figure 12: Comparación entre ciudad y kilómetros



---

Añadir heatmap correlación

Algunas conclusiones generales que sacamos de esta visualización es que los atributos característicos tienen un montón de valores únicos, esto puede hacer que realizar un tipo de etiquetado de one hot encoding no sea una buena idea, probaremos con ello igualmente. También hay valores perdidos, trabajaremos con esto probando distintos tipos de preprocesamiento. Finalmente nuestro conjunto está desbalanceado, realizaremos técnicas de oversample/unversampling para paliar con ello.

## Preprocesamientos

En esta sección vemos distintas maneras que hemos escogido para preprocesar nuestros datos, tratar con valores perdidos, etiquetación, skewed data, outliers y balanceo.

### Missing values

No hay valores duplicados, veamos los valores perdidos de nuestros datasets

Train.csv

	percent	count
Descuento	86.324964	4160
Potencia	3.631459	175
Asientos	2.199626	106
Motor_CC	2.095871	101
Consumo	1.514837	73
Nombre	1.494086	72
Ciudad	1.494086	72
Año	1.494086	72
Kilometros	1.494086	72
Combustible	1.494086	72
Tipo_marchas	1.494086	72
Mano	1.494086	72

Test.csv

	percent	count
Descuento	86.626402	1004

**Figure 13:** Missing values

Debido al porcentaje de valores nulos en nuestro atributo de descuento, siempre vamos a proceder a eliminar esa columna directamente ya que personalmente opino que no sale a cuenta intentar rellenarla ya que no deja de ser una estimación que por tanto arrastra un error así que al ser un número tan elevado la descartamos.

Respecto a lo restante, existen un gran número de procedimientos para el tratamiento de valores perdidos, nosotros siempre elegiremos entre eliminar muestras o variables que tienen datos faltantes. (Hay que tener cuidado y garantizar que las variables

---

descartadas no proporcionan información relevante) o sustituir los valores perdidos por estimaciones, por la media si las variables son numéricas (por ejemplo, Kilómetros) o por la moda si las variables son categóricas (Nombre, Mano etc...).

Antes de aplicar cualquier procedimiento sobre unos datos hay que analizar la naturaleza de los datos perdidos si los hubiere. Si su origen no fuera aleatorio, los valores perdidos no pueden ser ignorados. Una vez se sabe que son de origen aleatorio, se debe decidir si eliminarlos o sustituirlos por valores concretos según las técnicas anteriores. Si se tiene un tamaño suficientemente grande de muestras, siempre es preferible eliminarlos ya que sustituirlos no deja de ser una estimación que por tanto arrastra un error. Aparte de descartar siempre la columna **Descuento**, en nuestro caso probaremos siempre tres opciones.

- Eliminar directamente todos los valores nulos
- Rellenar los valores nulos con la moda si los atributos son categóricos o con la media si son numéricos
- Crear un nuevo atributo llamado **Company** como hemos comentado en la parte de visualización, y sustituir el valor perdido por la moda de este mismo atributo pero de esa marca. El código sería el siguiente:

---

```
#train.loc[train['brand_name'] == 'Maruti']['Asientos'].mode()[0]
def fill_na_with_mode(ds, brandname, column):
    if ds.loc[ds['Company'] == brandname][column].isnull().any() ==
        False:
        fill_value = ds.loc[ds['Company'] ==
            brandname][column].mode()[0]
        condit = ((ds['Company'] == brandname) & (ds[column].isnull()))
        ds.loc[condit, column] =
            ds.loc[condit, column].fillna(fill_value)
```

---

Por ejemplo, si estamos recorriendo los valores perdidos de nuestro atributo **Asientos**, sustituimos el valor perdido por la moda de Asientos de la marca que corresponda al valor perdido de Asientos. Si no hay siquiera nombre de compañía, no lo hacemos e igualmente luego eliminamos los valores perdidos restantes.

## Etiquetado

Como hemos comentado en la parte de visualización, debido a la gran cantidad de valores únicos de nuestros atributos puede que realizar etiquetados one-hot empeore algunos de nuestros algoritmos, al realizar one hot encoding sobre una variable categórica, estamos induciendo escasez en el conjunto de datos, lo cual no es deseable. Desde el punto de vista del algoritmo de división, todas las variables ficticias son independientes. Si el árbol decide realizar una división en una de las variables ficticias, la ganancia por división es muy marginal. Como resultado, es

---

muy poco probable que el árbol seleccione una de las variables ficticias más cercanas a la raíz. Para verificar esto, en nuestros modelos que utilicen árboles de decisión veremos la importancia de las características para los modelos y veremos que características destacan. Para realizar un correcto etiquetado utilizando Labels utilizaremos el ejemplo enseñado en los seminarios con los ficheros originales de los datos para el conjunto de train y de test, para probar distintos etiquetamiento one-hot utilizaremos el método *get\_dummies* de pandas. [Get], en la tabla final en la columna de preprocesado comentaremos que variables han sido etiquetadas y como pero nos moveremos entre estos dos tipos de etiquetado. Cabe comentar que cuando realizamos one-hot encoding sobre el conjunto de train y sobre el conjunto de test, puede darse la situación de que algunas columnas no coincidan, por ejemplo si se diera el caso que como hemos comentado antes, crear una columna con marcas de vehículos y aplicar one-hot encoding sobre la misma, el conjunto de test no tiene porque tener las mismas columnas que el conjunto de train, luego esto daría un error de dimensión, para ello si se diera el caso rellenamos el conjunto de test con columnas faltantes vacías con 0s, adjuntamos una parte del script para ello.

---

```
# Get missing columns in the training test
missing_cols = set( train.columns ) - set( test.columns )
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
    test[c] = 0
# Ensure the order of column in the test set is in the same order than in
  train set
test = test[train.columns]
```

---

## Escalado

En la práctica, siempre viene bien escalar los datos (Estandarizar características eliminando la media y escalando a la varianza de la unidad), en nuestro caso después de varias pruebas, aunque hacemos uso de modelos basados en distancias, no son ni de lejos los que mejores resultados proporcionan, así que en nuestro caso tratar con el escalado de los datos no será importante ya que no influirá en nuestros mejores resultados. Los modelos basados en distancias mejoran drásticamente mientras que para otros es irrelevante. Respecto a normalizar, probando un par de veces vemos que no beneficia nada a nuestro problema así que no normalizaremos nuestros datos. En conclusión, aplicaremos escalado ya que nuestros modelos basados en distancia se benefician, y los que no (que también son los que mejores resultados nos aportan) son indiferentes ante este escalado así que digamos que escalar no "molesta". No normalizaremos porque solo perjudica.

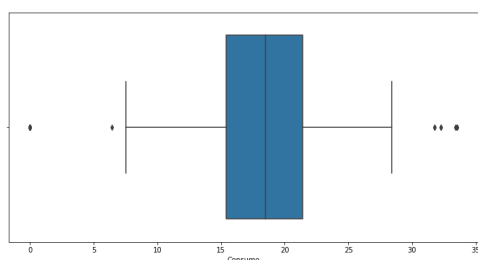
## Skewed data y outliers

Como hemos visto en la parte de visualización, tenemos skewed data y outliers en nuestros atributos, vamos a tratar los outliers de algunos de nuestros atributos, vamos a documentar un ejemplo de lo que sería un preprocesamiento completo de

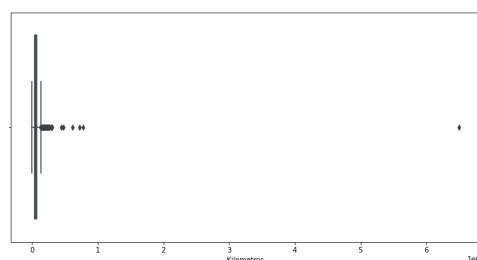
---

outliers y skewed data, aunque estos pueden realizarse individualmente y el resultado e imágenes en cuyo caso serán distintos, se dirá si se ha tenido en cuenta o no en la tabla final este tipo de preprocesamiento.

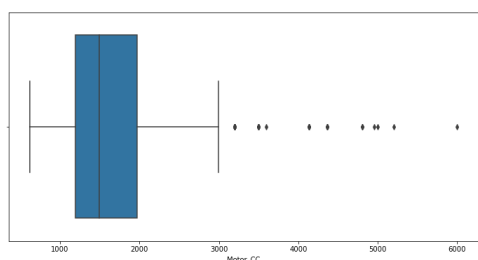
En la figura [18] mostramos 4 boxplots donde se aprecia claramente los outliers de nuestro conjunto, cabe comentar en el atributo consumo los valores 0.0, como ya hemos comentado en la parte de visualización, dudamos entre transformar estos atributos a valores nulos o eliminarlos directamente, en este caso vamos a proceder a eliminarlos. y a quedarnos con los datos más centrados.



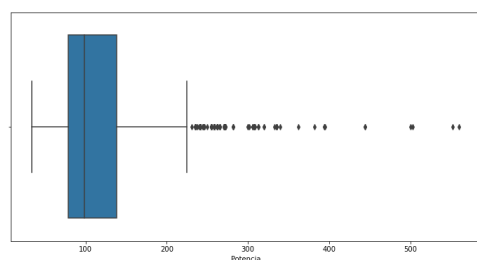
**Figure 14:** Outliers consumo



**Figure 15:** Outliers Kilometros



**Figure 16:** Outliers Motor\_CC



**Figure 17:** Outliers Potencia

**Figure 18:** Outliers

Podemos usar la funcion *quantile* de python para ver los quantiles de los extremos, en nuestro caso son:

---

```
Kilometros
max threshold : 451370.00000002445
min threshold : 1000.0
Potencia
max threshold : 456.93420000002664
min threshold : 34.2
Consumo
max threshold : 33.44
min threshold : 0.0
Motor_CC
max threshold : 4959.918000000034
min threshold : 624.0
```

---

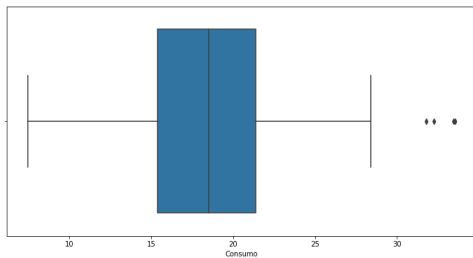
---

Vamos a reducir estos outliers, por ejemplo acotandolo por estos valores:

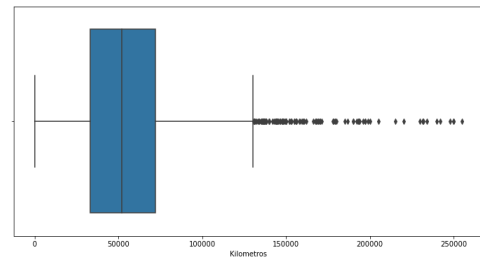
```
train=train[train['Kilometros'] < 262000]
train=train[train['Consumo'] > 0.0]
train=train[train['Potencia'] <= 530]
train=train[train['Motor_CC'] <= 5900 ]
```

---

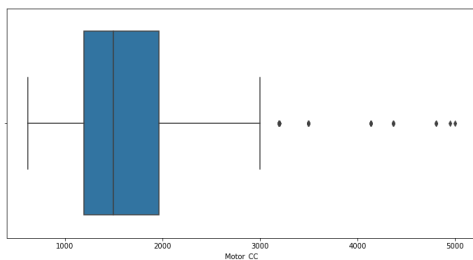
Después de nuestra transformación los datos quedarían como en la figura [23]



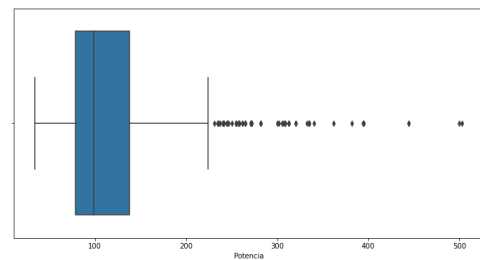
**Figure 19:** No outliers consumo



**Figure 20:** No outliers Kilometros



**Figure 21:** No outliers Motor\_CC



**Figure 22:** No outliers Potencia

**Figure 23:** No outliers

Por lo tanto, en los datos asimétricos, los outliers del skewed data pueden actuar como un valor atípico para el modelo estadístico y sabemos que los valores atípicos afectan negativamente el rendimiento del modelo, (especialmente los modelos basados en regresión). Hay modelos estadísticos que son robustos a valores atípicos como los modelos basados en árboles, pero limitarán la posibilidad de probar otros modelos. Por lo tanto, es necesario transformar los datos asimétricos para que se acerquen lo suficiente a una distribución gaussiana o distribución normal.

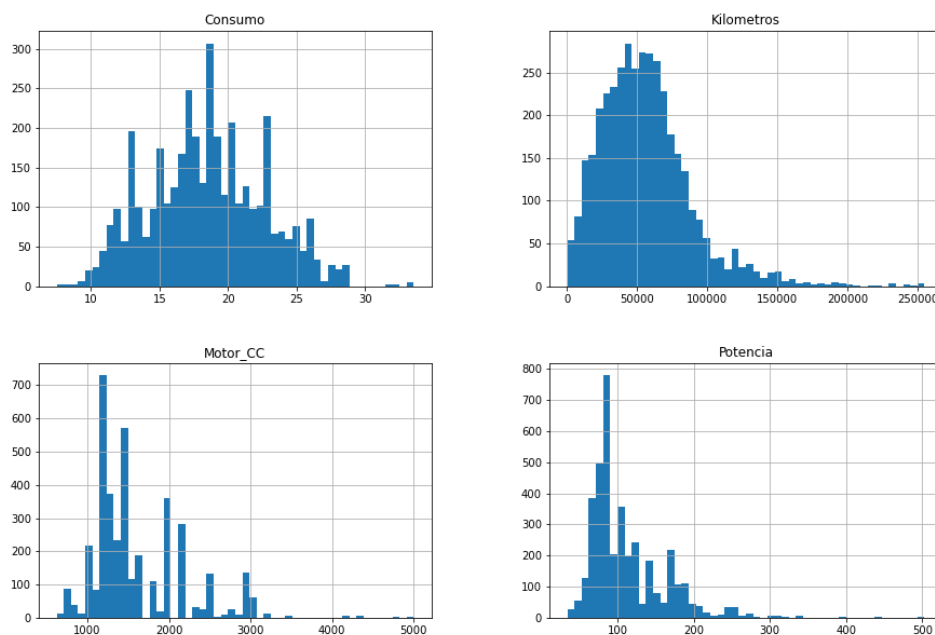
Por otro lado, en nuestros atributos, un dato se denomina skewed (*sesgado*) cuando la curva aparece distorsionada o sesgada hacia la izquierda o hacia la derecha, en una distribución estadística. En nuestro caso profundizemos en nuestros atributos Kilometros, Potencia, Consumo y Motor\_CC. Usamos la función skew [Ske] de la librería pandas para comprobar la asimetría de nuestros datos, un valor de asimetría de 0 en la salida denota una distribución simétrica de valores en la fila 1, un valor de asimetría negativo en la salida indica una asimetría en la distribución correspondiente a la izquierda y la cola es más grande hacia el lado izquierdo de la distribución

---

mientras que un valor de asimetría positivo en la salida indica una asimetría en la distribución correspondiente a derecha y la cola es más grande hacia el lado derecho de la distribución. Adjuntamos una tabla con lo comentado:

	skewness
Potencia	1.851863
Motor_CC	1.397533
Kilometros	1.369657
Consumo	0.177833

En una distribución normal, el gráfico tiene simetría, lo que significa que hay tantos valores de datos en el lado izquierdo de la mediana como en el lado derecho. En la figura [24] vemos la distribución de nuestros datos.



**Figure 24:** Skewed data

Aplicamos logaritmos para tratar con estos datos sesgados en la figura [25] vemos el resultado de nuestros datos después de haber aplicado técnicas logarítmicas

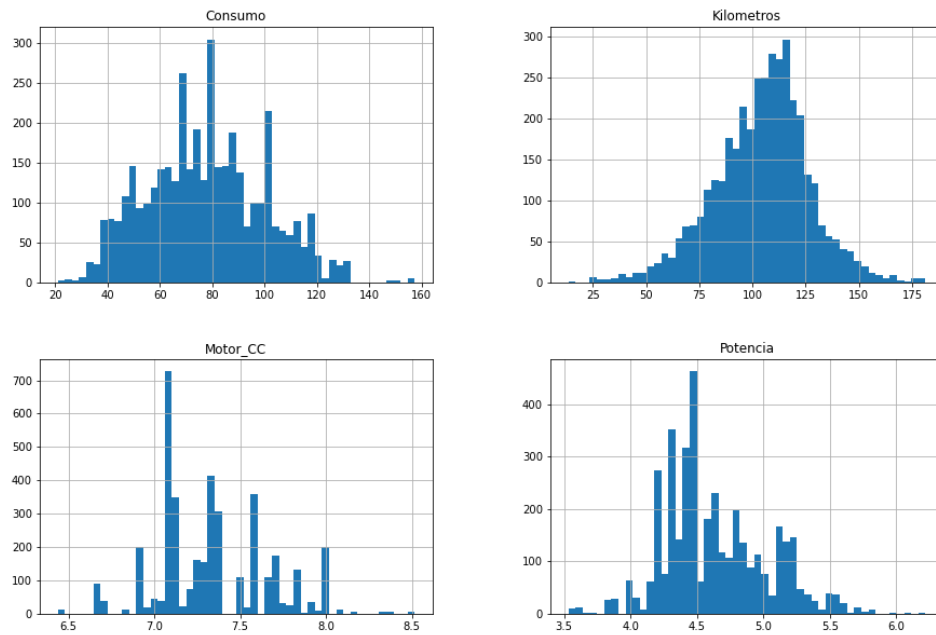


Figure 25: Skewed data

Mostramos una tabla con los datos transformados para medir el nivel de skew.

	skewness
Potencia	0.540517
Motor_CC	0.303247
Consumo	0.186635
Kilometros	-0.275876

## Balanceado

Cuando procedemos a dividir nuestro conjunto de entrenamiento entre la variable a predecir y el resto de atributos, nos encontramos con que nuestra variable a predecir, **Precio\_cat** esta muy desbalanceada, hay siempre un valor entre los 5 a predecir que en mucha más cantidad que el resto, si una clase está sobrerrepresentada frente al resto, el modelo tendrá mucha tendencia a elegirlo. Esto puede ser problemático o no, si los datos finales a predecir mantienen el mismo no-balanceamiento, pero si no fuese el caso sí podría dar problemas, mientras que si una clase aparece poco en las muestras respecto al resto el modelo puede tener tendencia a despreciar dicha clase ya que le afecta poco a sus medidas, llegando incluso a ignorarla en casos extremos. Esto se produce independientemente de si dicho no-balanceamiento aparece también en los datos finales sobre los que se vaya a predecir. La gravedad depende del nivel de desbalanceamiento y de si el conjunto de datos a predecir mantiene el

---

balanceamiento o no, pero en cualquier caso puede ser problemático, y es recomendable tratarlo.

Dependiendo de las características puede ser más conveniente uno que otro: undersampling, y oversampling. En nuestro caso probar con ambas técnicas llegados a la conclusión de que undersampling no es adecuado para nuestro caso, ya que elimina ejemplos de la clase mayoritaria y puede resultar en la pérdida de información invaluable para los modelos. Así que nos centramos principalmente en oversampling, de todos los distintos preprocesamientos el que más ganancia nos ha aportado en una mejor puntuación del modelo es este tipo de preprocesamiento en la variable a predecir, al principio comenzamos con el que se ha dado en clase de prácticas, Synthetic Minority Oversampling Technique (**SMOTE**, sintetiza nuevos ejemplos para la clase minoritaria), pero luego empezamos a utilizar **RandomOverSampling** (duplica ejemplos de la clase minoritaria en el conjunto de datos de entrenamiento, tarde nos dimos cuenta que esto puede resultar en un "overfitting" para algunos modelos) ya que nos proporcionaba mejores resultados. Se describirá en la tabla que tipo de balanceo se ha aplicado.

## Algoritmos

Aunque hemos usado gran cantidad de algoritmos de clasificación (SVM, KNeighbors, Extra Tree etc...), al final nos centramos solo en 3 de ellos ya que son los que mejores resultados nos proporcionan en nuestro conjunto train, vamos a comentarlos. Respecto a los preprocesamientos realizados para cada algoritmo, se escribirá explícitamente en la tabla, aunque el que más mejora ha resultado ser el trato con el balanceo, respecto a por ejemplo valores perdidos, escalado y demás, la mejora de esto es mínima, menos de 3% en todos los casos. Como hemos comentado en la parte de escalado, al ser algoritmos que no funcionan por distancia es normal que no influya la escala de los datos, además de ser métodos que funcionan bien aunque nuestro dataset no tengan muchos valores.

### Random Forest

**Random Forest Classifier**[Rfc] es un metaestimador que se ajusta a una serie de clasificadores de árboles de decisión en varias submuestras del conjunto de datos y utiliza promedios para mejorar la precisión predictiva y controlar el sobreajuste. Funciona bien en la mayoría de problemas, y en este no es una excepción, En nuestro caso aunque proporciona buenos resultados el problema es su overfitting muy rápido, por ello en algunas de nuestras subidas de ficheros a Kaggle se ha obtenido puntuaciones pésimas (alrededor de 0.2).

Para evitar un sobre ajuste excesivo lo que hemos intentado hacer es reducir el parámetro *max\_depth* y optimizar parámetros de ajuste que controlen la cantidad de características que se eligen al azar para hacer crecer cada árbol a partir de los datos de arranque. La función que hemos utilizado para hacer tuning de nuestro algoritmo es la siguiente:



---

```

def best_tree(X_train, y_train):
    #Randomized Search CV

    # Number of trees in random forest
    n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200,
        num = 12)]
    # Number of features to consider at every split
    max_features = ['auto', 'sqrt']
    # Maximum number of levels in tree 3-10 to deal with overfitting
    max_depth = [3, 4, 5, 6, 7, 8, 9]
    # Minimum number of samples required to split a node
    min_samples_split = [2, 5, 10, 15, 100]
    # Minimum number of samples required at each leaf node
    min_samples_leaf = [1, 2, 5, 10]

    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
        'max_features': max_features,
        'max_depth': max_depth,
        'min_samples_split': min_samples_split,
        'min_samples_leaf': min_samples_leaf}

    # search across different combinations
    rf_random = RandomizedSearchCV(estimator = RandomForestClassifier(),
        param_distributions = random_grid,
        scoring=scoring,
        n_iter = 15, cv = 5, verbose=2,
        random_state=42, n_jobs = 1)

    rf_random.fit(X_train,y_train)

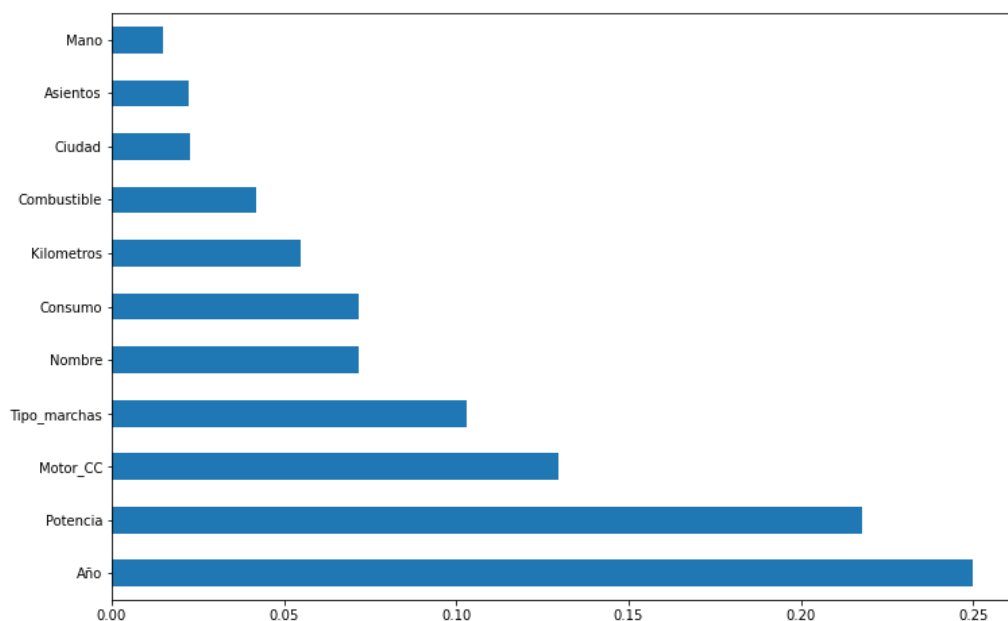
    return rf_random

```

---

Hemos elegido randomized search en vez de grid search ya que exploran exactamente el mismo espacio de parámetros y el resultado en la configuración de los parámetros es bastante similar, mientras que el tiempo de ejecución para la búsqueda aleatoria es drásticamente menor. [Ran]

Después de controlar el overfitting haciendo tuning de los parámetros mostrados anteriormente, se consiguen peores puntuaciones respecto a nuestro conjunto de train (lo cuál es normal, pero antes obteníamos puntuaciones perfectas lo que nos indicaba un claro overfitting) pero obtenemos mejores puntuaciones en nuestro conjunto de test. Se escribirá en la tabla cuales han sido los parámetros elegidos. Otra cosa buena del random forest es que podemos ver que características han tenido más importancia, por ejemplo en la figura [26] vemos un ejemplo de esto. Respecto a los preprocesamientos, como hemos comentado al inicio de esta sección, no se ha notado mucha mejoría excepto cuando tratamos con el desbalanceo.



**Figure 26:** Features importance

También cabe decir que aunque no es el modelo que nos proporciona mejores resultados, sin duda es el que más rápido funciona.

## XGBoost

El algoritmo **XGBoost**[Xgba] es eficaz para una amplia gama de problemas de modelos predictivos de clasificación.

Es una implementación eficiente del algoritmo de stochastic gradient boosting y ofrece una gama de hiperparámetros que brindan un control detallado sobre el procedimiento de entrenamiento del modelo. Aunque el algoritmo funciona bien en general, incluso en conjuntos de datos de clasificación desequilibrados, ofrece una forma de ajustar el algoritmo de entrenamiento para prestar más atención a la clasificación errónea de la clase minoritaria para conjuntos de datos con una distribución de clases sesgada. Está basado en un Random Forest ponderando a los individuos que se clasificaron mal en árboles anteriores. Al igual que en el caso de random forest classifier, XGBoost (y otros algoritmos de gradient boosting) tiene una serie de parámetros que se pueden ajustar para evitar un sobreajuste. La documentación está bastante bien respecto a estos parámetros [Xgbb], nuestra función de tuning es la siguiente:

---

```
def best_xgboost(X_train, X_test, y_train, y_test):
```

---

---

```

    param_dist = {'n_estimators': [int(x) for x in np.linspace(start = 100,
        stop = 1000, num = 12)],
# the learning rate of our GBM (i.e. how much we update our prediction with
    each successive tree); eta.
# Lower values avoid over-fitting.
        'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5],
        'subsample': [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.],
# the maximum depth of a tree; max_depth. Lower values avoid over-fitting
        'max_depth': [3, 4, 5, 6, 7, 8],
# the ratio of features used (i.e. columns used); colsample_bytree. Lower
    ratios avoid over-fitting.
        'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9, 1.],
# the minimum sum of instance weight needed in a leaf, in certain
    applications this relates directly to
# the minimum number of instances needed in a node; min_child_weight.
    Larger values avoid over-fitting.
        'min_child_weight': [1, 5, 15, 20, 50]
    }

    print(param_dist)

    xgboost_random = RandomizedSearchCV(estimator = XGBClassifier(),
        param_distributions = param_dist,
        scoring=scoring,
        n_iter = 15, cv = 5, verbose=2,
        random_state=42, n_jobs = 1)

    xgboost_random.fit(X_train, y_train, eval_metric = "mlogloss", eval_set
        = [(X_test, y_test)])

    return xgboost_random

```

---

Destacamos también el parámetro *subsample* Proporción de submuestras de las instancias de formación. Establecerlo en 0.5 significa que XGBoost muestreará al azar la mitad de los datos de entrenamiento antes de cultivar árboles. y esto evitará el sobreajuste. Se escribirá en la tabla el conjunto de parámetros utilizados en el algoritmo para la predicción.

El modelo XGBoost puede evaluar e informar sobre el rendimiento en un conjunto de prueba para el modelo durante el entrenamiento.

Admite esta capacidad especificando tanto un conjunto de datos de prueba como una métrica de evaluación en la llamada a `model.fit()` al entrenar el modelo y especificar una salida detallada.

En nuestro caso hemos elegido informar sobre la tasa de error de clasificación multiclase *mlogloss* en un conjunto de prueba independiente (`eval_set`) mientras entrenamos el modelo XGBoost. Produce mejores resultados que el random forest aunque en comparación al tiempo es muchísimo más lento.

---

## Catboost

**CatBoost** [Cata] es una biblioteca de software open-source desarrollada por Yandex. Proporciona un framework gradient boosting que intenta resolver las características categóricas utilizando una alternativa impulsada por permutación en comparación con un algoritmo clásico. Mientras que XGBoost fue el algoritmo más competitivo y preciso en Kaggle la mayor parte del tiempo, ahora está de moda un nuevo líder llamado **CatBoost**. En su documentación, incluyen referencias y ejemplos de GitHub, noticias, evaluaciones comparativas, comentarios, contactos, tutorial e instalación. Si está utilizando XGBoost, LightGBM o H2O, la documentación de CatBoost ha realizado una evaluación comparativa y ha demostrado que son los mejores con resultados ajustados y predeterminados. Por supuesto, si tiene variables más categóricas, CatBoost es el camino a seguir. Esta biblioteca permite algunas visualizaciones impresionantes, incluido el proceso de prueba y entrenamiento del modelo y una impresión de la importancia de las características, por nombrar algunas. Al contrario que en la mayoría de los algoritmos, en el caso de catboost **no hay que realizar one-hot encoding sobre los datos durante el preprocesamiento** [Catb], ya que esto afecta tanto a la velocidad del entrenamiento como a la calidad resultante. Por el contrario, hay un parámetro *one\_hot\_max\_size* que podemos usar para cambiar el número máximo de valores únicos de características categóricas para aplicar la codificación one-hot. Lo que significa que CatBoost no requiere la conversión del conjunto de datos a ningún formato específico como XGBoost y LightGBM. También cabe decir que **Catboost** soporta la opción de entrenar el modelo utilizando la GPU, en nuestro caso esto no será posible ya que no contamos con un ordenador que tenga una GPU.

De igual manera a nuestros algoritmos anteriores, mostramos la función que hemos elegido para hacer tuning de nuestro algoritmo, al contrario que en los casos anteriores, aquí usamos funciones exclusivas de la librería de catboost:

---

```
def best_catboost(X_train, y_train):

    grid = {
        'iterations': [800, 1000, 1200],
        'learning_rate': [0.03, 0.06, 0.1],
        'random_strength': [0.03, 0.06, 0.1],
        'depth': [4, 5, 6, 7, 8, 9, 10],
        'one_hot_max_size': [2,3,4,5,6,7,8],
        'od_type': ['IncToDec', 'Iter']}

    model = CatBoostClassifier(loss_function='MultiClass',
                              eval_metric='Accuracy', leaf_estimation_method='Newton')

    randomized_search_result = model.randomized_search(grid,
                                                       X=X_train,
                                                       y=y_train,
                                                       cv=5, n_iter=15, plot=True)

    return randomized_search_result
```

---

**Catboost** cuenta con detectores de sobreajuste, [Catc], si se produce un sobreajuste, CatBoost puede detener el entrenamiento antes de lo que dicten los parámetros de entrenamiento. Por ejemplo, se puede detener antes de que se construya el número especificado de árboles. El método de detección de sobreajuste se establece en el parámetro *od\_type*. Mantenemos siempre los parámetros de *loss\_function* y *eval\_metric* a esos valores ya que son los que se aplican a nuestro problema. Como hemos comentado antes, también podemos ver en una gráfica dinámica como se entrena nuestro modelo en tiempo real según los dos parámetros de evaluación y pérdida, vemos un ejemplo en la figura [27].

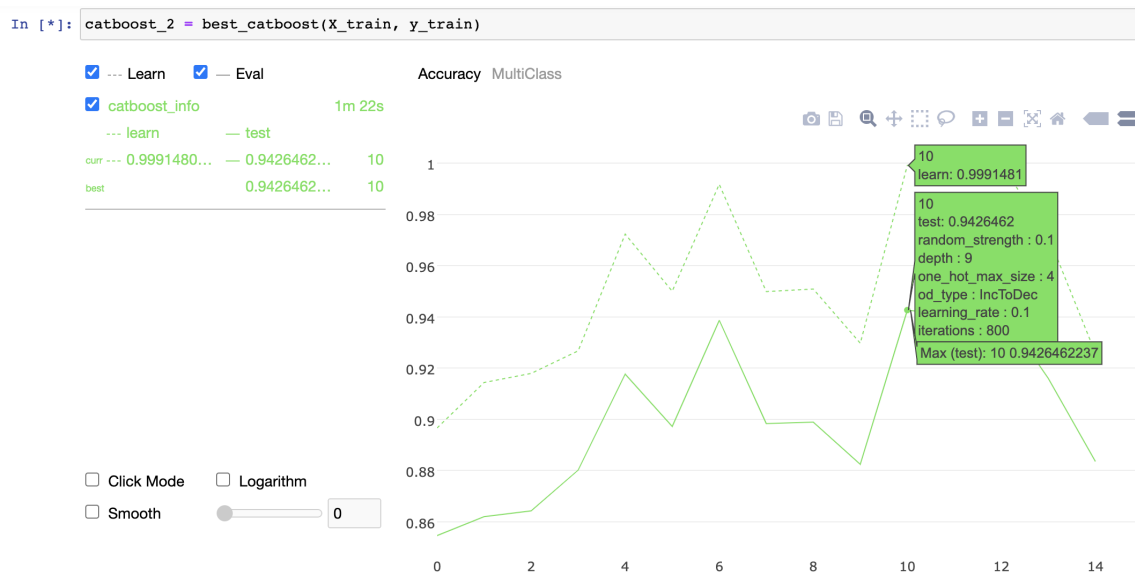


Figure 27: Catboost plot train

Este algoritmo ha sido el que mejores resultados nos ha proporcionado, y el que nos ha dado nuestra mejor puntuación, respecto al tiempo he de decir que tarda bastante si se hace tuning, (y más con la gráfica) aunque más o menos tarda lo mismo que xgboost

## Red Neuronal

Dedicamos una sección especial a la propuesta de una red neuronal con Keras <sup>2</sup>. Para la creación de la red neuronal, creamos un modelo secuencia [Kerb], a los que se les ha ido añadiendo varias capas densas [Kera], a partir de estos modelos mostramos en varias gráficas la ganancia y pérdida para cada uno de los modelos y nos quedamos con el que más nos convenga. Resumimos nuestro modelo en lo siguiente:

- Input Layer: Layer de entrada, creamos una capa densa con el número de inputs, esto depende del número de atributos que tengamos en ese momento

<sup>2</sup>La red neuronal puede encontrarse en un notebook aparte llamado neural\_network.ipynb

---

y depende del tipo de preprocesado, por ejemplo si simplemente aplicamos etiquetación a todos nuestros atributos tendremos una dimensión de 11 (11 atributos), mientras que si realizamos one hot encoding este número aumentará al tener más columnas.

- Hidden Layers: Layers de nodos entre el layer de input y de output, son capas densas que añadimos en el número que queramos, como máximo probaremos 5.
- Output Layer: El layer que produce el output, tendrá el número de nodos de nuestra variable a predecir, en formato one-hot, en nuestro caso es un valor entre 1-5 que aplicado con one-hot encoding es 5.

Para nuestras capas densas, vamos a usar siempre funciones de activación *relu*, excepto en nuestro último layer de output, que usaremos *softmax*, debido a que funciona bien para problemas multiclase como output [Sof]. Softmax asigna probabilidades decimales a cada clase en un problema de varias clases. Esas probabilidades decimales deben sumar 1.0. Esta restricción adicional ayuda a que el entrenamiento converja más rápidamente de lo que lo haría de otra manera. El código de creación de nuestro modelo es el siguiente:

---

```
def create_custom_model(input_dim, output_dim, nodes, n=1, name='model'):  
    def create_model():  
        # Create model  
        model = Sequential(name=name)  
        for i in range(n):  
            model.add(Dense(nodes, input_dim=input_dim, activation='relu'))  
        model.add(Dense(output_dim, activation='softmax'))  
  
        # Compile model  
        model.compile(loss='categorical_crossentropy',  
                      optimizer='adam',  
                      metrics=['accuracy'])  
        return model  
    return create_model
```

---

Respecto a optimizador hemos escogido *Adam* ya que combina las buenas propiedades de Adadelta y RMSprop y, por lo tanto, tiende a funcionar mejor para la mayoría de los problemas y *loss* *categorical\_crossentropy* ya que es un problema de clasificación multiclase.

Para la creación de nuestro modelo muchos de los valores de parámetros se han elegido empíricamente, por ejemplo para la elección de nodos de cada uno de nuestros hidden layers, haciendo un poco de investigación vemos que no hay una fórmula objetiva para escogerlos. [Nod]. En nuestro caso hemos elegido la fórmula

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

donde

- $N_i$  número de neuronas de entrada
- $N_o$  número de neuronas de output
- $N_s$  número de ejemplos del conjunto de entrenamiento
- $\alpha$  un factor arbitrario entre 5 y 10 (en nuestro caso hemos escogido 5)

Haremos un ejemplo de ejecución donde como preprocesado escogemos llenar los valores perdidos numéricos con la media y categóricos con la moda, utilizaremos label encoding para nuestros atributos y para el balanceo utilizamos RandomOverSampling.

```
alpha = 5
nodes = n_samples // (alpha * (n_features + n_classes))
print("Number of nodes:", nodes)
models = [create_custom_model(n_features, n_classes, nodes, i,
                              'model_{}'.format(i))
          for i in range(1, 6)]

for create_model in models:
    create_model().summary()
```

Number of nodes: 69

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 69)	828
dense_2 (Dense)	(None, 5)	350

Total params: 1,178  
Trainable params: 1,178  
Non-trainable params: 0

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 69)	828
dense_4 (Dense)	(None, 69)	4830
dense_5 (Dense)	(None, 5)	350

Total params: 6,008  
Trainable params: 6,008  
Non-trainable params: 0

---

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 69)	828
dense_7 (Dense)	(None, 69)	4830
dense_8 (Dense)	(None, 69)	4830
dense_9 (Dense)	(None, 5)	350

Total params: 10,838  
 Trainable params: 10,838  
 Non-trainable params: 0

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 69)	828
dense_11 (Dense)	(None, 69)	4830
dense_12 (Dense)	(None, 69)	4830
dense_13 (Dense)	(None, 69)	4830
dense_14 (Dense)	(None, 5)	350

Total params: 15,668  
 Trainable params: 15,668  
 Non-trainable params: 0

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 69)	828
dense_16 (Dense)	(None, 69)	4830
dense_17 (Dense)	(None, 69)	4830
dense_18 (Dense)	(None, 69)	4830
dense_19 (Dense)	(None, 69)	4830
dense_20 (Dense)	(None, 5)	350

Total params: 20,498  
 Trainable params: 20,498



---

Non-trainable params: 0

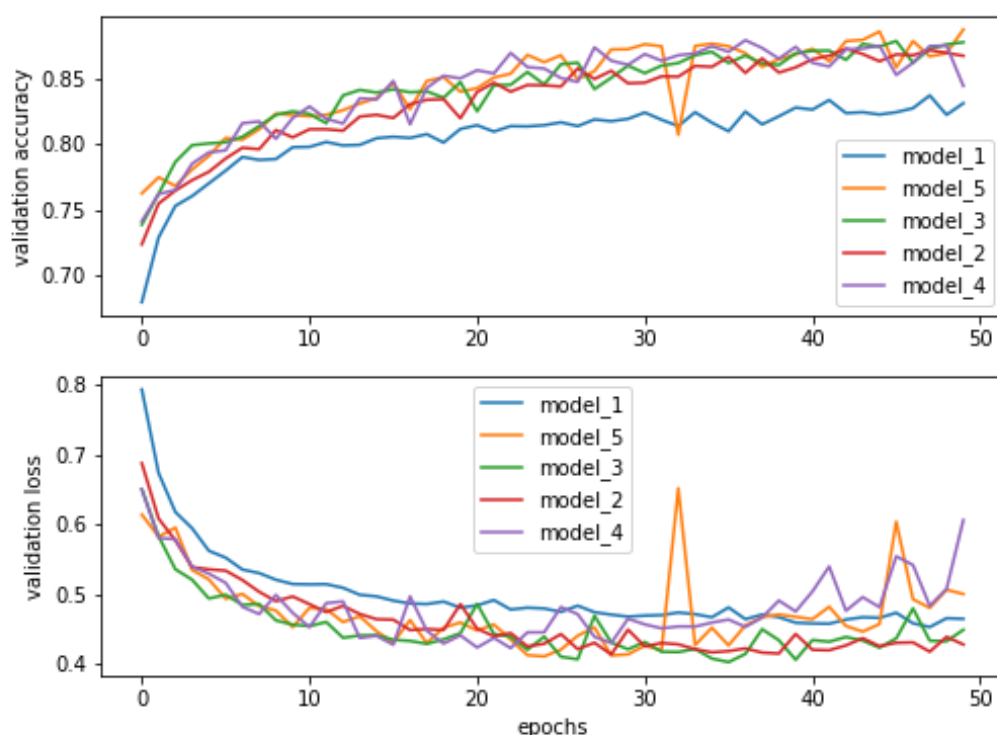
Para entrenar los modelos debemos de establecer algunos parámetros como el *batch\_size*, una cantidad de muestras procesadas antes de que se actualice el modelo, y los *epochs*, número de pases completos a través del conjunto de datos de entrenamiento. El *batch\_size* debe ser mayor o igual a uno y menor o igual al número de muestras en el conjunto de datos de entrenamiento.

El número de *epochs* se puede establecer en un valor entero entre uno e infinito.

Para el *batch\_size* no hará una gran diferencia para nuestro problema a menos que estemos entrenando cientos de miles o millones de observaciones.

En general: los *batch\_size* más grandes dan como resultado un progreso más rápido en el entrenamiento, pero no siempre convergen tan rápido. Los *batch\_size* más pequeños se entrenan más lentamente, pero pueden converger más rápido. Definitivamente depende del problema, en nuestro caso escogeremos 5.

En general, los modelos mejoran con más *epochs* de entrenamiento, hasta cierto punto. Comenzarán a estabilizarse en precisión a medida que convergen. Intentaremos algo con 50. Llamaremos a los modelos mostrados anteriormente del 1 al 5 correspondiendo al número de capas densas que contenga, vemos el resultado de nuestra ejecución en la figura [28] .

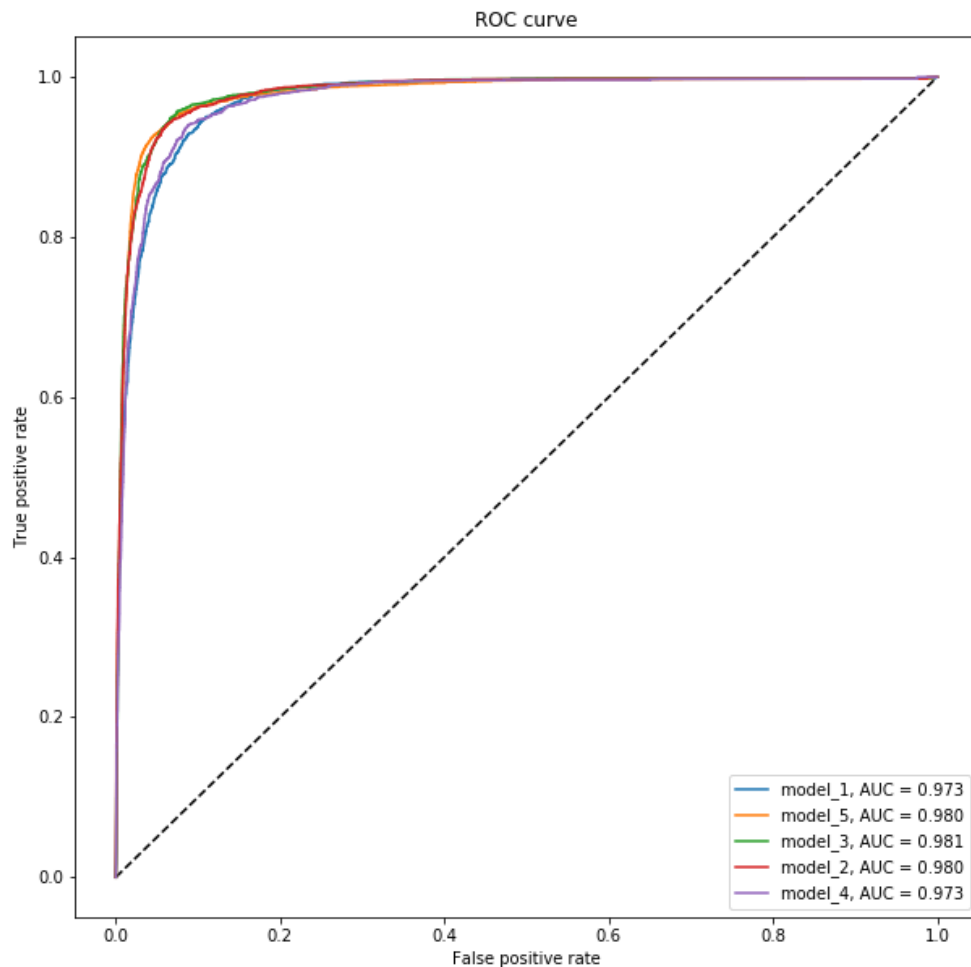


**Figure 28:** Convergencia modelos red neuronal

Vemos que con ese número de *epochs* nuestros modelos convergen sin problema,

---

nuestro Test loss ronda 0.4773907017086763 y test accuracy 0.8876628076115841, veamos ahora el AUC de nuestros modelos en la figura [29], escogemos el segundo modelo ya que parece más estable, y hacemos la validación cruzada con 5 particiones de el mismo donde obtenemos un resultado de **0.857 (+/- 0.012)**.



**Figure 29:** AUC ROC red neuronal

Con esto concluye esta sección respecto a la red neuronal, comprobandola contra nuestro conjunto de test, conseguimos una score de **0.75**, comparando lo que tarda cada ejecución de la red neuronal, tanto en entrenar, como en encontrar valores óptimos llegamos a la conclusión de que no es rentable para este problema (o tal vez nuestro conjunto de valores elegidos no ha sido el adecuado). También probamos a utilizar BatchNormalization y con otros datos pero no conseguimos mejores resultados y como hemos comentado, el tiempo de ejecución es muy elevado comparado con otros algoritmos (puede llegar a tardar **horas**), un simple Random Forest Clas-

sifier proporciona mejores resultados tardando muchísimo menos.

# Tabla

Adjuntamos una foto de nuestras mejores 20 subidas en la figura [30].

Submission and Description	Private Score	Public Score	Use for Final Score
<a href="#">mis_resultados_catboost_1.csv</a> 17 hours ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.81190	0.81190	<input type="checkbox"/>
<a href="#">mis_resultados_catboost_2.csv</a> 4 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.80586	0.80586	<input type="checkbox"/>
<a href="#">mis_resultados_catboost_2.csv</a> 2 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.79723	0.79723	<input type="checkbox"/>
<a href="#">mis_resultados_xgboost_1.csv</a> a day ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.79292	0.79292	<input type="checkbox"/>
<a href="#">mis_resultados_xgboost_1.csv</a> 17 hours ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.79292	0.79292	<input type="checkbox"/>
<a href="#">mis_resultados_xgboost_2.csv</a> 4 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.79119	0.79119	<input type="checkbox"/>
<a href="#">mis_resultados_xgboost_2.csv</a> 3 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.79033	0.79033	<input type="checkbox"/>
<a href="#">mis_resultados_xgboost_1.csv</a> 2 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.78861	0.78861	<input type="checkbox"/>
<a href="#">mis_resultados_1.csv</a> 5 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.78774	0.78774	<input type="checkbox"/>
<a href="#">mis_resultados_catboost_1.csv</a> 4 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.78429	0.78429	<input type="checkbox"/>
<a href="#">mis_resultados_xgboost_2.csv</a> a day ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.78343	0.78343	<input type="checkbox"/>
<a href="#">mis_resultados_catboost_1.csv</a> 2 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.77911	0.77911	<input type="checkbox"/>
<a href="#">mis_resultados_1.csv</a> 5 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.76531	0.76531	<input type="checkbox"/>
<a href="#">mis_resultados_1.csv</a> 8 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.76445	0.76445	<input type="checkbox"/>
<a href="#">mis_resultados_3.csv</a> 13 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.75409	0.75409	<input type="checkbox"/>
<a href="#">mis_resultados_2.csv</a> 12 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.74460	0.74460	<input type="checkbox"/>
<a href="#">mis_resultados_svc_1.csv</a> a day ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.74115	0.74115	<input type="checkbox"/>
<a href="#">mis_resultados_1.csv</a> 12 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.73943	0.73943	<input type="checkbox"/>
<a href="#">mis_resultados_1.csv</a> 11 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.73252	0.73252	<input type="checkbox"/>
<a href="#">mis_resultados_nn.csv</a> 3 days ago by <a href="#">David Martín Vela</a> <a href="#">add submission details</a>	0.72648	0.72648	<input type="checkbox"/>

Figure 30: Subidas a la plataforma Kaggle

---

Todo lo relevante para recrear las subidas se encuentran en el notebook **vanilla.ipynb**, donde tenemos principalmente dos versiones, una primera versión **vanilla** (una versión básica donde tratamos los valores perdidos, etiquetamos y pasamos directamente a los modelos) y una segunda versión donde realizamos algo más de preprocesados, la diferencia entre ambas es el tipo de etiquetado (en el primero únicamente usamos label encoding mientras que en el segundo hacemos one hot de otros atributos) y que en la segunda versión podemos tratar con los outliers y skewed data, en ambas versiones tenemos disponibles los preprocesamientos de valores perdidos, podemos ejecutarlos selectivamente en nuestro código dependiendo de la opción que queramos, igual con el tipo de balanceo (SMOTE o RandomOverSampling). La mayoría de métodos relacionados con algoritmo ejecutan un conjunto de todo ellos, y luego se profundizan sobre los que hemos comentado en la sección de algoritmos realizando tuning. De estos que se profundizan se genera un archivo con el nombre de **mis\_resultados\_{algoritmo-utilizado}\_{versión}** (Los que no tienen el algoritmo utilizado son los iniciales donde únicamente usaba random forest y la versión). En score train se introduce la puntuación de validación cruzada con métrica accuracy utilizando 5 particiones y en score test la puntuación obtenida en Kaggle. En conclusión, por mucho preprocesamiento que hagamos al final si entendemos y sabemos configurar un algoritmo complejo con los parámetros adecuados podemos alcanzar resultados muy prometedores sin siquiera tener que indagar en el dataset. Al final los preprocesamientos que más diferencias han proporcionado han sido los relacionados con el balance de nuestro conjunto de datos. Cuando nos referimos a **Tuning** y **No tuning**, lo que queremos decir es que en un caso se ha realizado un estudio de los parámetros del algoritmo (sea para evitar el sobreajuste por ejemplo ajustado la profundidad) mientras que en el otro se han escogido unos parámetros por defecto que pueden funcionar bien o no. La tabla está ordenada por la puntuación obtenida en el conjunto **test**, la columna de la posición quiere decir la posición en la que nos encontrábamos en el momento al haber subido el fichero, puede que nuestra subida no fuera muy buena y en cuyo caso seguiríamos teniendo la posición de otra subida en otro momento.

Preprocesamiento	Fecha Subida	Posición	Algoritmo	Score Train	Score Test
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	30/12/2020	5	Red Neuronal 3 capas densas 1 input layer (relu), 1 hidden layer (relu) y 1 output layer (softmax)	0.857	0.72648
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con SMOTE	22/12/2020	8	(No tuning) RandomForestClassifier() (Sospechamos de overfitting) Accuracy on Training set: 1.0 Accuracy on Testing set: 0.9123287671232877	0.88739	0.73252
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con SMOTE	21/12/2020	7	(Tuning) RandomForestClassifier(max_depth = 9, max_features = 'auto', min_samples_leaf = 1, min_samples_split = 15, n_estimators = 1000)	0.84520	0.73943
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	01/01/2020	5	(Tuning) SVC(kernel = 'rbf', gamma = 'auto', C = 100)	0.90312	0.74115
(V2) Valores perdidos media y moda Etiquetado de one hot encoding de ciudad, tipo.marchas, combustible y company. Se elimina Nombre. Balanceo con SMOTE	21/12/2020	7	(No tuning) RandomForestClassifier() Accuracy on Training set: 1.0 Accuracy on Testing set: 0.914349276974416 (Sospechamos de overfitting)	0.89140	0.74460
(V2) Valores perdidos eliminados Etiquetado de one hot encoding de ciudad, tipo.marchas y combustible, Label encoding Nombre, Skewed data y outliers Balanceo con SMOTE	20/12/2020	6	(Tuning) RandomForestClassifier(n_estimators = 1000 min_samples_split = 15, min_samples_leaf = 1, max_features = 'auto', max_depth = 9)	0.84455	0.75409
(V1) Valores perdidos media y moda Etiquetamos con LabelEncoder Balanceo con SMOTE	27/12/2020	10	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 7, learning_rate = 0.15, colsample_bytree = 1.0)	0.90628	0.76445
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	28/12/2020	7	(No tuning) RandomForestClassifier() (Sospechamos de overfitting) Accuracy on Training set: 1.0 Accuracy on Testing set: 0.9632925472747497	0.94841	0.76531
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con SMOTE	31/12/2020	8	(No tuning) CatBoostClassifier(iterations=1000, learning_rate=0.1, random_strength=0.1, depth=7, loss_function='MultiClass', eval_metric='Accuracy', leaf_estimation_method='Newton')	0.94382	0.77911
(V2) Valores perdidos eliminados Etiquetamos con one hot Ciudad, Tipo.marchas, Combustible y Company, Balanceo con RandomOverSampling	01/01/2020	5	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 8, learning_rate = 0.15, colsample_bytree = 1.0)	0.93534	0.78343
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	29/12/2020	7	(No tuning) CatBoostClassifier(iterations=1000, learning_rate=0.1, random_strength=0.1, depth=7, loss_function='MultiClass', eval_metric='Accuracy', leaf_estimation_method='Newton')	0.92213	0.78429
(V1) Valores perdidos media y moda Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	28/12/2020	7	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 8, learning_rate = 0.15, colsample_bytree = 1.0)	0.85165	0.78774
(V1) Valores perdidos eliminados Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	31/12/2020	7	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 8, learning_rate = 0.15, colsample_bytree = 1.0)	0.94716	0.78861
(V2) Valores perdidos moda Company Etiquetamos con one hot Ciudad, Tipo.marchas y Combustible, Label encoding Nombre, Skewed data y outliers, Balanceo con RandomOverSampling	30/12/2020	7	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 7, learning_rate = 0.15, colsample_bytree = 1.0) (Possible overfitting) Accuracy on Training set: 1.0 Accuracy on Testing set: 0.9624481438489293	0.94952	0.79033
(V2) Valores perdidos eliminados Etiquetamos con one hot Ciudad, Tipo.marchas y Combustible Label encoding Nombre, Skewed data y outliers, Balanceo con RandomOverSampling	29/12/2020	7	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 9, learning_rate = 0.15, colsample_bytree = 1.0)	0.95244	0.79119
(V1) Valores perdidos moda Company, Etiquetamos con LabelEncoder, Balanceo con RandomOverSampling	01/01/2021	5	(Tuning) XGBClassifier(subsample = 0.7, n_estimators = 918, min_child_weight = 1, max_depth = 8, learning_rate = 0.15, colsample_bytree = 1.0)	0.94595	0.79292
(V2) Valores perdidos moda Company, Etiquetamos con one hot Ciudad, Tipo.marchas y Combustible Skewed data y outliers, Balanceo con RandomOverSampling	31/12/2020	5	(No tuning) CatBoostClassifier(iterations=1000, learning_rate=0.1, random_strength=0.1, depth=7, loss_function='MultiClass', eval_metric='Accuracy', leaf_estimation_method='Newton')	0.94586	0.79723
(V2) Valores perdidos eliminados Etiquetamos con one hot Ciudad, Tipo.marchas y Combustible Label encoding Nombre, Skewed data y outliers, Balanceo con RandomOverSampling	29/12/2020	5	(Tuning) CatBoostClassifier(depth = 10, one_hot_max_size = 2, iterations = 800, random_strength = 0.03, learning_rate = 0.1, od_type = 'IncToDec', loss_function='MultiClass', eval_metric='Accuracy', leaf_estimation_method='Newton')	0.94650	0.80586
(V1) Valores perdidos media y moda Etiquetamos con LabelEncoder Balanceo con RandomOverSampling	-	-	(Tuning) CatBoostClassifier(depth = 10, one_hot_max_size = 2, iterations = 800, random_strength = 0.03, learning_rate = 0.1, od_type = 'IncToDec', loss_function='MultiClass', eval_metric='Accuracy', leaf_estimation_method='Newton')	0.93758	0.81190

Table 1: Tabla de puntuaciones

# Referencias

- [Cata] *Catboost*. URL: <https://catboost.ai/>.
- [Catb] *Catboost One Hot*. URL: <https://catboost.ai/docs/features/categorical-features.html>.
- [Catc] *Catboost Overfit*. URL: <https://catboost.ai/docs/concepts/overfitting-detector.html>.
- [Get] *Get Dummies*. URL: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get\\_dummies.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html).
- [Kera] *Keras dense*. URL: [https://keras.io/api/layers/core\\_layers/dense/](https://keras.io/api/layers/core_layers/dense/).
- [Kerb] *Keras Sequential*. URL: [https://keras.io/guides/sequential\\_model/](https://keras.io/guides/sequential_model/).
- [Nod] *Nodes hidden layers*. URL: <https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw>.
- [Rfc] *Random Forest Classifier*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [Ran] *Randomized Grid Search*. URL: [https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_randomized\\_search.html](https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html).
- [Ske] *Skewed data*. URL: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.skew.html>.
- [Sof] *Softmax*. URL: <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>.
- [Xgba] *XGBoost*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [Xgbb] *XGBoost tuning*. URL: <https://xgboost.readthedocs.io/en/latest/parameter.html>.