```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))
```

```
# Conectamos con nuestro Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pip install xgboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages (0.9
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
```

```
pip install lightgbm
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: lightgbm in /usr/local/lib/python3.7/dist-packages (2
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy import stats
import datetime
from datetime import date
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostRegressor
from sklearn.decomposition import PCA
from sklearn.ensemble import BaggingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
import xgboost as xgb_
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")


import sys
```

```
!unzip /content/drive/MyDrive/EXABY/DATA_SET.zip -d EXABY
```

```
Archive:  /content/drive/MyDrive/EXABY/DATA_SET.zip
   creating: EXABY/DATA_SET/
  inflating: EXABY/DATA_SET/Anuncios_usados.csv
  inflating: EXABY/DATA_SET/Consumos_emisiones.csv
  inflating: EXABY/DATA_SET/Modelos.json
  inflating: EXABY/DATA_SET/Precio_historico.xml
  inflating: EXABY/DATA_SET/Tablas y Variables v2.pdf
  inflating: EXABY/DATA_SET/Tipo_de_cambio.csv
  inflating: EXABY/DATA_SET/Ventas_nuevas.json
  inflating: EXABY/DATA_SET/Versiones.csv
```

```
data = pd.read_table('/content/drive/MyDrive/EXABY/full_df.csv', delimiter = ',')
```

```
versiones = pd.read_table('/content/EXABY/DATA_SET/Versiones.csv', delimiter = ',')
```

```
data.drop(columns=data.columns[0], axis=1, inplace=True)
```

```
data.Reg_year = data.Reg_year.astype(int)
data.Runned_Miles = data.Runned_Miles.replace('1 mile', 1)
data.Runned_Miles = data.Runned_Miles.astype(int)
data.Engine_power = data.Engine_power.round(3)
data.Average_mpg = data.Average_mpg.round(3)
data.Ad_price_GBP = data.Ad_price_GBP.round(3)
```

Clusterización

Despues de definir el modelo de clusterizacion a implementar, aplicamos k-means para posteriormente organizar los valores por la clasificacion del modelo k-means y realizar la predicción.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209520 entries, 0 to 209519
Data columns (total 26 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   Genmodel_ID            209520 non-null  object 
 1   Adv_ID                 209520 non-null  object 
 2   Reg_year               209520 non-null  int64  
 3   Bodytype               209520 non-null  object 
 4   Runned_Miles           209520 non-null  int64  
 5   Gearbox                209520 non-null  object 
 6   Fuel_type              209520 non-null  object 
 7   Engine_power           209520 non-null  float64
 8   Wheelbase              209520 non-null  float64
 9   Height                 209520 non-null  float64
 10  Width                  209520 non-null  float64
 11  Length                 209520 non-null  float64
 12  Average_mpg            209520 non-null  float64
 13  Top_speed              209520 non-null  float64
 14  Seat_num               209520 non-null  float64
 15  Door_num               209520 non-null  float64
 16  Ad_Date                209520 non-null  object 
 17  Ad_price_GBP           209520 non-null  float64
 18  Engine_size            209520 non-null  float64
 19  Entry_price_GBP        209520 non-null  float64
 20  Gas_emission           209520 non-null  float64
 21  Tot_devaluation        209520 non-null  float64
 22  Percentage_devaluation 209520 non-null  float64
 23  Age                    209520 non-null  float64
 24  Tot_Dev_PerYear        209520 non-null  float64
 25  Per_Dev_PerYear        209520 non-null  float64
dtypes: float64(18), int64(2), object(6)
memory usage: 41.6+ MB
```

```
df = data.iloc[:, [2,4,7,8,9,10,11,12,13,14,15,17,18,19,20,23,22]]
```

```
scaler = StandardScaler()
```

```
df = pd.DataFrame(scaler.fit_transform(df.values), columns=df.columns, index=df.index)
```

```
df.head()
```

	Reg_year	Runned_Miles	Engine_power	Wheelbase	Height	Width	Length
0	-2.841218	-0.561657	3.481406	1.340505	-0.105126	1.608635	2.541191
1	-2.841218	-0.196791	3.481406	1.340505	-0.105126	1.608635	2.541191

```
#inversed = pd.DataFrame(scaler.inverse_transform(df.values), columns = df.columns, index
```

```
pca = PCA(n_components=10)
pca.fit(df)
```

```
PCA(n_components=10)
```

```
X_pca = pca.transform(df)
```

```
result=pd.DataFrame(X_pca, columns=['PCA%i' % i for i in range(10)], index=df.index)

result
```

	PCA0	PCA1	PCA2	PCA3	PCA4	PCA5	PCA6	P
0	11.725942	4.055185	1.424336	-1.186796	-0.624917	-0.014333	-0.251504	1.117
1	11.570355	4.272420	1.555266	-0.976882	-0.485545	-0.035949	-0.223659	0.773
2	9.655950	3.711655	1.287174	-0.595318	-0.437819	-0.217259	-0.286216	1.419
3	11.572519	4.455430	1.584277	-0.939624	-0.460195	0.014390	-0.146929	0.807
4	11.564454	4.522239	1.610395	-0.895718	-0.428030	0.052291	-0.089853	0.770
...	...	...	...	...	...	...	...	
209515	-1.314249	-0.076282	0.216879	1.505677	0.239294	1.493453	0.318136	-0.164
209516	-1.188275	0.220188	0.277281	1.309100	0.177713	1.305300	0.142597	-0.204
209517	-1.352012	0.068634	0.261612	1.583458	0.286722	1.494537	0.343650	-0.218
209518	-1.364790	0.124295	0.279272	1.614206	0.306119	1.500036	0.359583	-0.239
209519	-1.363258	0.146919	0.288313	1.630103	0.318599	1.522063	0.389944	-0.249

209520 rows × 10 columns

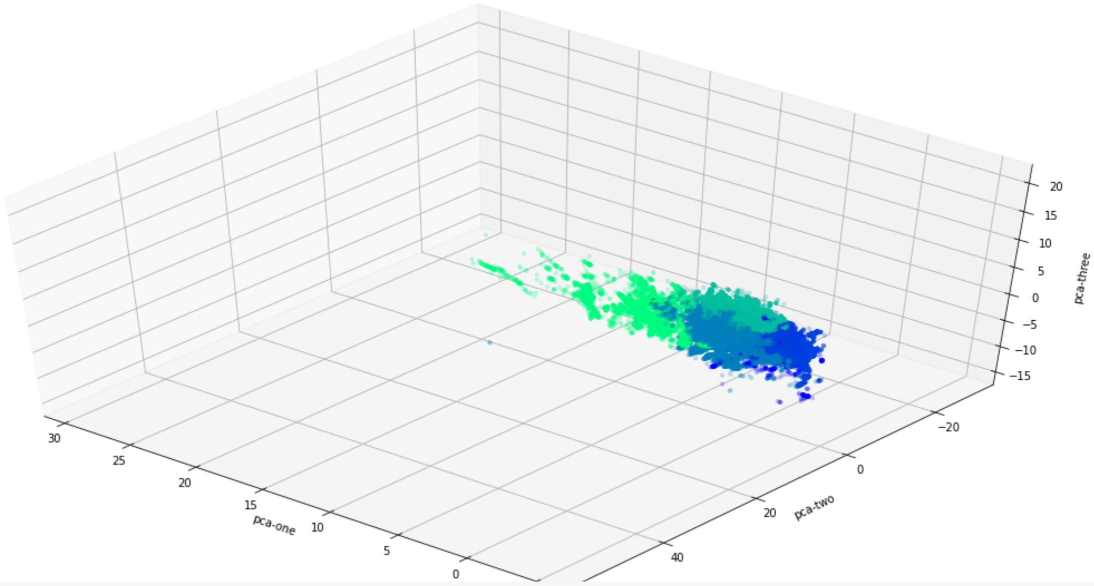
```
algoritmo =  KMeans(n_clusters= 5, init = 'k-means++',
                      max_iter= 300, n_init =10)
algoritmo.fit(result)
```

```
KMeans(n_clusters=5)
```

```
plt.figure(figsize=(15,5))
y_kmeans = algoritmo.predict(result)

ax = plt.figure(figsize=(20,10)).gca(projection='3d')
```

```
centers = algoritmo.cluster_centers_
ax.scatter(
    xs=centers[:, 0],
    ys=centers[:, 1],
    c = 'black',
    alpha = 1,
    s = 60,

)



ax.scatter(
    xs=result["PCA0"],
    ys=result["PCA1"],
    zs=result["PCA2"],
    c = y_kmeans,
    cmap='winter',
    s = 10,
    alpha = 0.2
)
ax.set_xlabel('pca-one')
ax.set_ylabel('pca-two')
ax.set_zlabel('pca-three')

#rotacion de ejes
ax.azim = 300
ax.elev = 10

#limit
#ax.axes.set_xlim3d(min(result["PCA0"]),4e6)
#ax.axes.set_ylim3d(min(result["PCA1"]),max(result["PCA1"]))
#ax.axes.set_zlim3d(0,400000)

ax.zaxis.labelpad=10
ax.azim = 130
ax.elev = 50

plt.show()
```

```
<Figure size 1080x360 with 0 Axes>
```




```
#pca.inverse_transform(result)
```

```
np.unique(y_kmeans )
```

```
array([0, 1, 2, 3, 4], dtype=int32)
```

```
result["Cluster"] = y_kmeans
```

```
pd.set_option('display.max_rows', 100)
data['Cluster'] = y_kmeans
data.groupby('Cluster').describe().transpose().head(100)
```

	Cluster	0	1	2	3	
Reg_year	**count**	3853.000000	85218.000000	6.481000e+04	47084.000000	8
	mean	2011.969115	2013.946784	2.007027e+03	2014.451002	2
	std	3.221380	2.359396	2.718333e+00	2.173319	4
	min	2000.000000	2002.000000	2.000000e+03	2006.000000	2
	25%	2010.000000	2012.000000	2.005000e+03	2013.000000	2
	50%	2012.000000	2014.000000	2.007000e+03	2015.000000	2
	75%	2014.000000	2016.000000	2.009000e+03	2016.000000	2
	max	2019.000000	2019.000000	2.014000e+03	2019.000000	2
Runned_Miles	**count**	3853.000000	85218.000000	6.481000e+04	47084.000000	8
	mean	58164.605243	35000.069129	9.351242e+04	39846.187580	3
	std	35490.176024	24497.499744	4.866288e+04	29742.146575	2
	min	1.000000	0.000000	1.080000e+02	0.000000	5
	25%	31500.000000	15502.250000	7.100000e+04	15000.000000	1
	50%	55128.000000	30000.000000	9.000000e+04	33426.500000	2
	75%	81350.000000	49998.000000	1.120000e+05	58689.500000	5
	max	260000.000000	271112.000000	6.363342e+06	232000.000000	2
Engine_power	**count**	3853.000000	85218.000000	6.481000e+04	47084.000000	8

```
data.groupby('Cluster').describe().transpose().tail(100)
```

	Cluster	0	1	2	
Average_mpg	**25%**	44.000000	51.000000	36.000000	42.
	50%	47.000000	57.000000	42.000000	50.
	75%	56.000000	65.000000	48.000000	58.
	max	156.000000	200.000000	76.000000	156.
Top_speed	**count**	3853.000000	85218.000000	64810.000000	47084.
	mean	120.764652	111.511527	122.105610	132.
	std	11.729554	9.937637	14.676918	12.
	min	84.000000	11.000000	84.000000	90.
	25%	112.000000	106.000000	112.000000	124.
	50%	118.000000	111.357143	121.000000	131.
	75%	130.000000	117.000000	130.165217	140.
	max	180.000000	183.000000	183.000000	204.
Seat_num	**count**	3853.000000	85218.000000	64810.000000	47084.
	mean	5.208149	4.805170	4.923823	5.
	std	1.161321	0.603725	0.847185	0.
	min	2.000000	1.000000	2.000000	2.
	25%	5.000000	5.000000	5.000000	5.
	75%	5.000000	5.000000	5.000000	5.

Cluster 0: Mean Age 6 (3853 samples) 2nd Most seat nums (>5 in mean) AVERAGE CARS

Cluster 1: Mean Age 4 (85218 samples) Least Gas emissions Smallest engine size Least runned miles SMALL CARS

Cluster 2: Mean Age 11 (64810 samples) Greatest runned miles AVERAGE OLD cars

Cluster 3: Mean Age 3.5 (47084 samples) Most seat nums (>5 in mean) 2nd fastest top speed biggest FAMILIAR/MINI-VAN

Cluster 4: Mean Age 6.5 (855 samples) Most Gas Emissions (++) Biggest engine size (++) Most engine power (++) Fastest top speed Only one with door num < 3 SPORTS CARS

(++) by great difference

```
data.dtypes
```

```
Genmodel_ID           object
Adv_ID                object
Reg_year               int64
Bodytype              object
Runned_Miles           int64
Gearbox               object
Fuel_type             object
Engine_power         float64
```

```
Wheelbase                   float64
Height                      float64
Width                       float64
Length                      float64
Average_mpg                 float64
Top_speed                   float64
Seat_num                    float64
Door_num                    float64
Ad_Date                      object
Ad_price_GBP                float64
Engine_size                 float64
Entry_price_GBP             float64
Gas_emission                float64
Tot_devaluation             float64
Percentage_devaluation      float64
Age                         float64
Tot_Dev_PerYear             float64
Per_Dev_PerYear             float64
Cluster                       int32
dtype: object
```

```
result["Percentage_devaluation"] = data['Percentage_devaluation']
```

```
result
```

	PCA0	PCA1	PCA2	PCA3	PCA4	PCA5	PCA6	P
0	11.725942	4.055185	1.424336	-1.186796	-0.624917	-0.014333	-0.251504	1.117
1	11.570355	4.272420	1.555266	-0.976882	-0.485545	-0.035949	-0.223659	0.77
2	9.655950	3.711655	1.287174	-0.595318	-0.437819	-0.217259	-0.286216	1.41
3	11.572519	4.455430	1.584277	-0.939624	-0.460195	0.014390	-0.146929	0.807
4	11.564454	4.522239	1.610395	-0.895718	-0.428030	0.052291	-0.089853	0.77
...	...	...	...	...	...	...	...	
209515	-1.314249	-0.076282	0.216879	1.505677	0.239294	1.493453	0.318136	-0.164
209516	-1.188275	0.220188	0.277281	1.309100	0.177713	1.305300	0.142597	-0.204
209517	-1.352012	0.068634	0.261612	1.583458	0.286722	1.494537	0.343650	-0.21
209518	-1.364790	0.124295	0.279272	1.614206	0.306119	1.500036	0.359583	-0.23
209519	-1.363258	0.146919	0.288313	1.630103	0.318599	1.522063	0.389944	-0.24

209520 rows × 12 columns

Se separa el dataset por clusters para posteriormente realizar en train test split y entrenar los modelos de regresion

```
df1 = result[result["Cluster"] == 0]
df2 = result[result["Cluster"] == 1]
```

```
df3 = result[result["Cluster"] == 2]
df4 = result[result["Cluster"] == 3]
df5 = result[result["Cluster"] == 4]
```

```
X = df1.drop(["Percentage_devaluation"], axis = 1)
y = df1.Percentage_devaluation

X1 = df2.drop(["Percentage_devaluation"], axis = 1)
y1 = df2.Percentage_devaluation

X2 = df3.drop(["Percentage_devaluation"], axis = 1)
y2 = df3.Percentage_devaluation

X3 = df4.drop(["Percentage_devaluation"], axis = 1)
y3 = df4.Percentage_devaluation

X4 = df5.drop(["Percentage_devaluation"], axis = 1)
y4 = df5.Percentage_devaluation
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.30, random_sta
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.30, random_sta
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_size=0.30, random_sta
X_train4, X_test4, y_train4, y_test4 = train_test_split(X4, y4, test_size=0.30, random_sta
```

Modelos de regresion

[ ] ↳ *43 celdas ocultas*

Stratification of datasets for visualization

[ ] ↳ *9 celdas ocultas*