Universidad ICESI

Facultad de ingenieria y diseño

Maestria en ciencia de datos

Proyecto final - Fundamentos de analitica

Laura Loaiza

```
!pip install lazy
!pip install lazypredict
!pip install hyperopt
!pip install pyclustering
```

```
Collecting lazy
  Downloading lazy-1.5-py2.py3-none-any.whl (5.0 kB)
Installing collected packages: lazy
Successfully installed lazy-1.5
Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: pandas in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lazypredict) (1.4.4)
Requirement already satisfied: scikit-learn in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lazypredict) (1.0.2)
Requirement already satisfied: joblib in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lazypredict) (1.1.0)
Collecting xgboost
  Downloading xgboost-1.7.5-py3-none-win_amd64.whl (70.9 MB)
     -------------------------------------- 70.9/70.9 MB 376.6 kB/s
eta 0:00:00
Requirement already satisfied: tqdm in c:\users\mqa200-0489\anaconda3\
lib\site-packages (from lazypredict) (4.64.1)
Requirement already satisfied: click in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lazypredict) (8.0.4)
Collecting lightgbm
  Downloading lightgbm-3.3.5-py3-none-win_amd64.whl (1.0 MB)
     ---------------------------------------- 1.0/1.0 MB 968.3 kB/s
eta 0:00:00
Requirement already satisfied: colorama in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from click->lazypredict) (0.4.5)
Requirement already satisfied: numpy in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lightgbm->lazypredict) (1.21.5)
Requirement already satisfied: scipy in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lightgbm->lazypredict) (1.9.1)
Requirement already satisfied: wheel in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from lightgbm->lazypredict) (0.37.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\
mqa200-0489\anaconda3\lib\site-packages (from scikit-learn-
```

```
>lazypredict) (2.2.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\
mqa200-0489\anaconda3\lib\site-packages (from pandas->lazypredict)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from pandas->lazypredict) (2022.1)
Requirement already satisfied: six>=1.5 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas-
>lazypredict) (1.16.0)
Installing collected packages: xgboost, lightgbm, lazypredict
Successfully installed lazypredict-0.2.12 lightgbm-3.3.5 xgboost-1.7.5
Collecting hyperopt
  Downloading hyperopt-0.2.7-py2.py3-none-any.whl (1.6 MB)
     ---------------------------------------- 1.6/1.6 MB 498.5 kB/s
eta 0:00:00
Requirement already satisfied: numpy in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from hyperopt) (1.21.5)
Requirement already satisfied: scipy in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from hyperopt) (1.9.1)
Requirement already satisfied: six in c:\users\mqa200-0489\anaconda3\
lib\site-packages (from hyperopt) (1.16.0)
Requirement already satisfied: networkx>=2.2 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from hyperopt) (2.8.4)
Requirement already satisfied: future in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from hyperopt) (0.18.2)
Collecting py4j
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     ------------------------------------ 200.5/200.5 kB 347.8 kB/s
eta 0:00:00
Requirement already satisfied: cloudpickle in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from hyperopt) (2.0.0)
Requirement already satisfied: tqdm in c:\users\mqa200-0489\anaconda3\
lib\site-packages (from hyperopt) (4.64.1)
Requirement already satisfied: colorama in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from tqdm->hyperopt) (0.4.5)
Installing collected packages: py4j, hyperopt
Successfully installed hyperopt-0.2.7 py4j-0.10.9.7
Collecting pyclustering
  Downloading pyclustering-0.10.1.2.tar.gz (2.6 MB)
     ---------------------------------------- 2.6/2.6 MB 218.0 kB/s
eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: scipy>=1.1.0 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from pyclustering) (1.9.1)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\mqa200-
0489\anaconda3\lib\site-packages (from pyclustering) (3.5.2)
Requirement already satisfied: numpy>=1.15.2 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from pyclustering) (1.21.5)
```

```
Requirement already satisfied: Pillow>=5.2.0 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from pyclustering) (9.2.0)
Requirement already satisfied: packaging>=20.0 in c:\users\mqa200-
0489\anaconda3\lib\site-packages (from matplotlib>=3.0.0-
>pyclustering) (21.3)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\mqa200-
0489\anaconda3\lib\site-packages (from matplotlib>=3.0.0-
>pyclustering) (4.25.0)
Requirement already satisfied: cycler>=0.10 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->pyclustering)
(0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\mqa200-
0489\anaconda3\lib\site-packages (from matplotlib>=3.0.0-
>pyclustering) (1.4.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\mqa200-
0489\anaconda3\lib\site-packages (from matplotlib>=3.0.0-
>pyclustering) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\
mqa200-0489\anaconda3\lib\site-packages (from matplotlib>=3.0.0-
>pyclustering) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from python-dateutil>=2.7-
>matplotlib>=3.0.0->pyclustering) (1.16.0)
Building wheels for collected packages: pyclustering
  Building wheel for pyclustering (setup.py): started
  Building wheel for pyclustering (setup.py): finished with status
'done'
  Created wheel for pyclustering: filename=pyclustering-0.10.1.2-py3-
none-any.whl size=2395106
sha256=6cc3f22cdb19dd5b73c6178c1c7b93b960beaf92f416ef1fa2ac83d69482589
8
  Stored in directory: c:\users\mqa200-0489\appdata\local\pip\cache\
wheels\e0\56\c2\abb6866a3fcd8a55862f1df8a18f57805c3a78fed9a9023cb9
Successfully built pyclustering
Installing collected packages: pyclustering
Successfully installed pyclustering-0.10.1.2
```

```python
import pandas as pd
import numpy as np
import warnings
warnings.simplefilter("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import math
from collections import Counter
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, accuracy_score,
silhouette_samples, silhouette_score, calinski_harabasz_score
```

```python
from sklearn import preprocessing
from sklearn.decomposition import PCA
from dataprep.eda import create_report
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score
from lazypredict.Supervised import LazyClassifier
import xgboost as xgb
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from hyperopt import hp, tpe, fmin
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabasz_score
from sklearn.metrics import silhouette_samples
import matplotlib.pyplot as plt
import lightgbm as lgb
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram

""

""

#Cargamos el dataset
data=pd.read_csv("imports-85.data", header=None)

#Renombramos las columnas con nombres mas apropiados para los
atributos
data.columns = [
    "symboling", "normalized-losses", "make", "fuel-type",
"aspiration", "num-of-doors", "body-style", "drive-wheels",
    "engine-location", "wheel-base", "length", "width", "height",
"curb-weight", "engine-type", "num-of-cylinders",
    "engine-size", "fuel-system", "bore", "stroke", "compression-
ratio", "horsepower", "peak-rpm", "city-mpg", "highway-mpg",
    "price"
]

#remplazamos los signos de interrogación (?) con nan
data=data.replace("?",np.nan)

data.head()
```

```
  symboling  normalized-losses         make fuel-type aspiration num-
of-doors  \
0         3                NaN  alfa-romero       gas        std
two
1         3                NaN  alfa-romero       gas        std
two
2         1                NaN  alfa-romero       gas        std
two
3         2             164.00         audi       gas        std
four
4         2             164.00         audi       gas        std
four

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
0  convertible          rwd           front       88.60  ...
130
1  convertible          rwd           front       88.60  ...
130
2    hatchback          rwd           front       94.50  ...
152
3        sedan          fwd           front       99.80  ...
109
4        sedan          4wd           front       99.40  ...
136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
city-mpg  \
0         mpfi  3.47    2.68              9.00     111.00   5000.00
21
1         mpfi  3.47    2.68              9.00     111.00   5000.00
21
2         mpfi  2.68    3.47              9.00     154.00   5000.00
19
3         mpfi  3.19    3.40             10.00     102.00   5500.00
24
4         mpfi  3.19    3.40              8.00     115.00   5500.00
18

   highway-mpg    price
0           27 13495.00
1           27 16500.00
2           26 16500.00
3           30 13950.00
4           22 17450.00

[5 rows x 26 columns]
```

Atributos:

1. symboling: -3, -2, -1, 0, 1, 2, 3.
2. normalized-losses: continua de 65 a 256.
3. make: alfa-romero, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota, volkswagen, volvo
4. fuel-type: diesel, gas.
5. aspiration: std, turbo.
6. num-of-doors: four, two.
7. body-style: hardtop, wagon, sedan, hatchback, convertible.
8. drive-wheels: 4wd, fwd, rwd.
9. engine-location: front, rear.
10. wheel-base: continua de 86.6 a 120.9.
11. length: continua de 141.1 a 208.1.
12. width: continua de 60.3 a 72.3.
13. height: continua de 47.8 a 59.8.
14. curb-weight: continua de 1488 a 4066.
15. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
16. num-of-cylinders: eight, five, four, six, three, twelve, two.
17. engine-size: continua de 61 a 326.
18. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
19. bore: continua de 2.54 a 3.94.
20. stroke: continua de 2.07 a 4.17.
21. compression-ratio: continua de 7 a 23.
22. horsepower: continua de 48 a 288.
23. peak-rpm: continua de 4150 a 6600.
24. city-mpg: continua de 13 a 49.

```
#verificamos que Dtype tiene cada atributo
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  164 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       203 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
```

```
 11  width               205 non-null    float64
 12  height              205 non-null    float64
 13  curb-weight         205 non-null    int64
 14  engine-type         205 non-null    object
 15  num-of-cylinders    205 non-null    object
 16  engine-size         205 non-null    int64
 17  fuel-system         205 non-null    object
 18  bore                201 non-null    object
 19  stroke              201 non-null    object
 20  compression-ratio   205 non-null    float64
 21  horsepower          203 non-null    object
 22  peak-rpm            203 non-null    object
 23  city-mpg            205 non-null    int64
 24  highway-mpg         205 non-null    int64
 25  price               201 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

```
#Corregimos el formato de los atributos
data["normalized-losses"]=data["normalized-losses"].astype(float)
data["price"]=data["price"].astype(float)
data["bore"]=data["bore"].astype(float)
data["stroke"]=data["stroke"].astype(float)
data["horsepower"]=data["horsepower"].astype(float)
data["peak-rpm"]=data["peak-rpm"].astype(float)
data["price"]=data["price"].astype(float)
data["symboling"]=data["symboling"].astype(int)
```

```
#Revisamos los valores unicos de la variable dependiente Y
data["symboling"].unique()
```

```
array([ 3,  1,  2,  0, -1, -2])
```

```
#Analisis exploratorio de datos
create_report(data)
```

{"model_id":"","version_major":2,"version_minor":0}

```
#Matrx de correlaciones Spearman
correlation_matrix = data.corr(method='spearman')
correlation_matrix
```

```
                   normalized-losses  wheel-base  length  width
height  \
normalized-losses               1.00       -0.11    0.02   0.11   -
0.39
wheel-base                     -0.11        1.00    0.91   0.81
0.63
length                          0.02        0.91    1.00   0.89
```

```
0.53
width                          0.11         0.81    0.89   1.00
0.35
height                        -0.39         0.63    0.53   0.35
1.00
curb-weight                    0.09         0.77    0.89   0.86
0.35
engine-size                    0.08         0.65    0.78   0.77
0.20
bore                          -0.06         0.54    0.64   0.61
0.23
stroke                         0.09         0.22    0.18   0.24   -
0.03
compression-ratio             -0.05        -0.13   -0.19  -0.15
0.00
horsepower                     0.24         0.50    0.66   0.69
0.01
peak-rpm                       0.30        -0.32   -0.27  -0.20   -
0.30
city-mpg                      -0.25        -0.49   -0.67  -0.69   -
0.07
highway-mpg                   -0.20        -0.54   -0.70  -0.70   -
0.13
price                          0.19         0.68    0.81   0.81
0.26

                   curb-weight  engine-size  bore  stroke
compression-ratio  \
normalized-losses         0.09         0.08 -0.06    0.09
-0.05
wheel-base                0.77         0.65  0.54    0.22
-0.13
length                    0.89         0.78  0.64    0.18
-0.19
width                     0.86         0.77  0.61    0.24
-0.15
height                    0.35         0.20  0.23   -0.03
0.00
curb-weight               1.00         0.88  0.70    0.16
-0.22
engine-size               0.88         1.00  0.72    0.29
-0.23
bore                      0.70         0.72  1.00   -0.08
-0.17
stroke                    0.16         0.29 -0.08    1.00
-0.07
compression-ratio        -0.22        -0.23 -0.17   -0.07
1.00
horsepower                0.81         0.82  0.65    0.14
```

```
-0.36
peak-rpm                  -0.24        -0.28 -0.31   -0.07
-0.03
city-mpg                  -0.81        -0.73 -0.62   -0.04
0.48
highway-mpg               -0.83        -0.72 -0.63   -0.03
0.45
price                      0.91         0.83  0.65    0.12
-0.18

                   horsepower  peak-rpm  city-mpg  highway-mpg  price

normalized-losses        0.24      0.30     -0.25        -0.20   0.19

wheel-base               0.50     -0.32     -0.49        -0.54   0.68

length                   0.66     -0.27     -0.67        -0.70   0.81

width                    0.69     -0.20     -0.69        -0.70   0.81

height                   0.01     -0.30     -0.07        -0.13   0.26

curb-weight              0.81     -0.24     -0.81        -0.83   0.91

engine-size              0.82     -0.28     -0.73        -0.72   0.83

bore                     0.65     -0.31     -0.62        -0.63   0.65

stroke                   0.14     -0.07     -0.04        -0.03   0.12

compression-ratio       -0.36     -0.03      0.48         0.45  -0.18

horsepower               1.00      0.11     -0.91        -0.88   0.85

peak-rpm                 0.11      1.00     -0.13        -0.06  -0.08

city-mpg                -0.91     -0.13      1.00         0.97  -0.83

highway-mpg             -0.88     -0.06      0.97         1.00  -0.83

price                    0.85     -0.08     -0.83        -0.83   1.00
```

Conclusiones EDA

Analisis grafico y descriptivo

- Dataset de tamaño 205x26.
- 26 atributos: 11 Categoricos y 15 Numericos.

- el 59% de los registros en el dataset tiene una clasificacion de riesgo 0 y 1.
- No hay carros bastante seguros (symbolizing = -3) en el dataset.
- el pago de pérdida promedio relativo por año de vehículo asegurado toma valores entre 65 y 256 con una media de 122.
- El top 5 de marcas de vehiculos con mas registros en este dataset son Toyota, Nissan, Mazda, Honda y Mitsubishi.
- El 90,24% de los vehiculos en este dataset tienen motores a gasolina.
- El 18,05% de los vehiculos en este dataset tienen motores con turbo.
- El dataset se encuentra bien balanceado entre carros con 4 puertas (55,61%) y 2 puertas (43,41%).
- La mayoria de vehiculos en este dataset tiene carroceria tipo Sedan (46,83%).
- El 58,54% de vehiculos en este dataset tiene traccion delantera y solo es 4,39% es 4x4.
- El 98.54% de vehiculos en este dataset tiene el motor ubicado en la parte frontal.
- Los vehiculos en este dataset tienen una media de distancia entre ejes de 98,75 (posiblemente en cm).
- Los vehiculos en este dataset tienen una media de longitud de 174,04, una media de anchura de 65.90 y una media de altura de 53,72 (posiblemente cm).
- El peso medio de los vehiculos en este dataset es de 2555,56 (posiblemente kilogramos).
- El 72,20% de los vehiculos en este dataset cuenta con motor con árbol de levas en cabeza.
- El 77,56% de los vehiculos en este dataset cuenta con un motor de 4 cilindros.
- El 45,85% de los vehiculos en este dataset cuenta con un sistema multipuerto de inyección de combustible electrónica y el 32,20% tiene un sistema de combustible 2BBL (dos barriles).
- La media de caballos de fuerza de los vehiculos en este dataset es de 5125, con un maxino de 6660 y un minimo de 4150.
- La media de millas por galon de los vehiculos en este dataset es de 25,21 mpg en ciudad y de 30,75 mpg en carretera.
- Segun este dataset El carro con mejor rendimiento de millas por galon en ciudad es de marca Honda y El carro con peor rendimiento de millas por galon en ciudad es de marca Jaguar.
- Segun este dataset El carro con mejor rendimiento de millas por galon en carretera es de marca Honda y El carro con peor rendimiento de millas por galon en carretera es de marca Mercedez.
- La media de precio de los vehiculos de este data set es de 13207,12 (posiblemente dolares). Siendo el vehiculo mas costoso un Mercedez benz y el mas barato un Subaru.

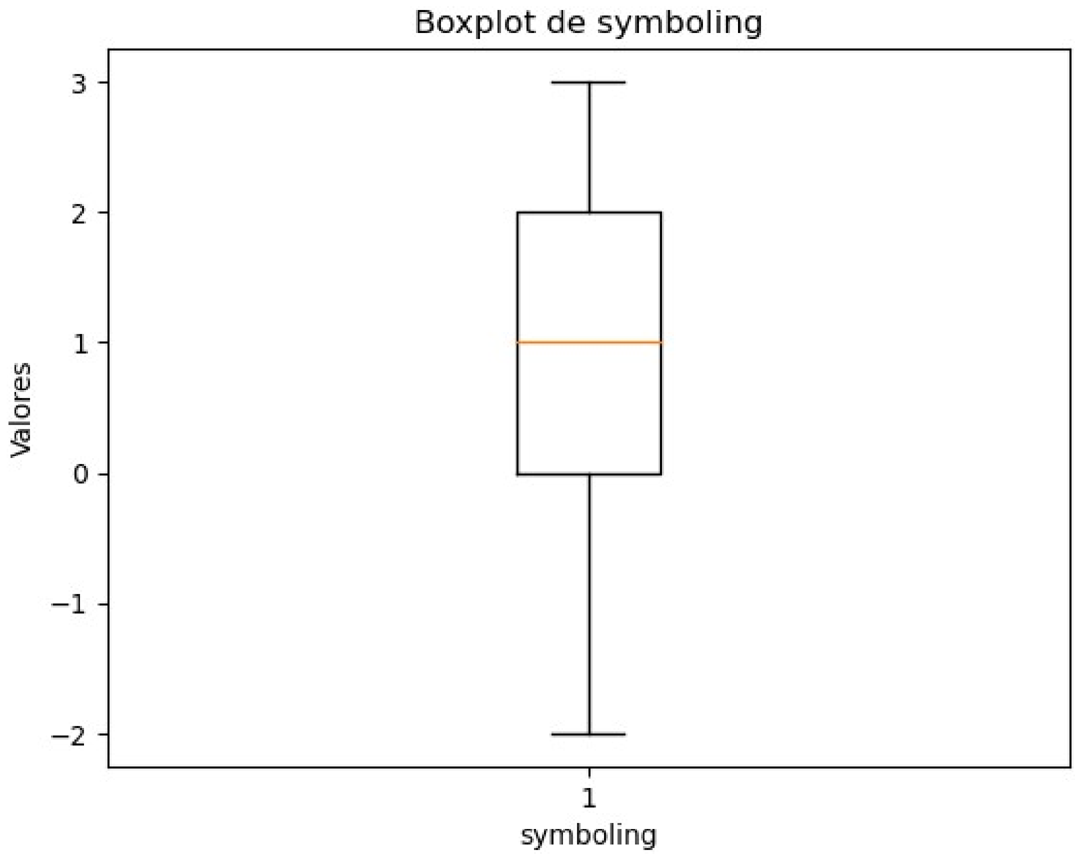
Correlaciones Spearman

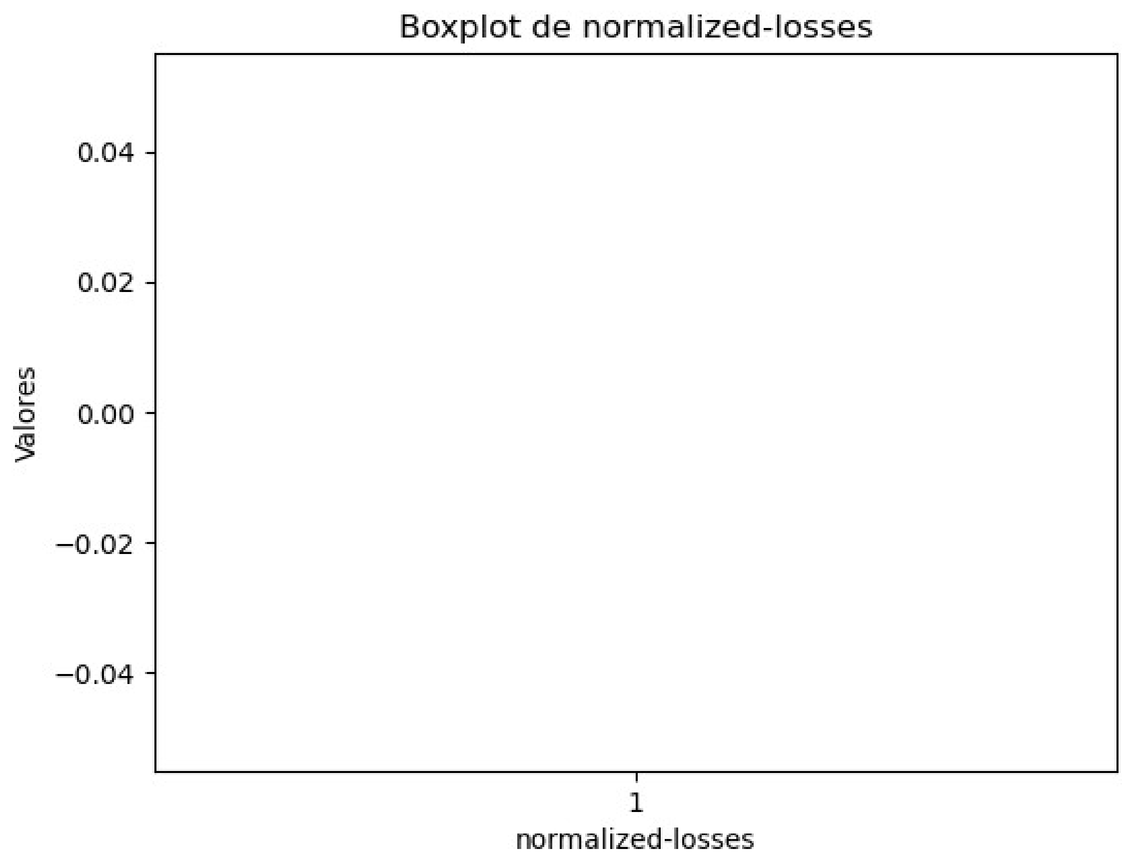
Las variables independientes estan altamente correlacionadas en su mayoria, aqui mencionaremos las correlaciones mas fuertes:

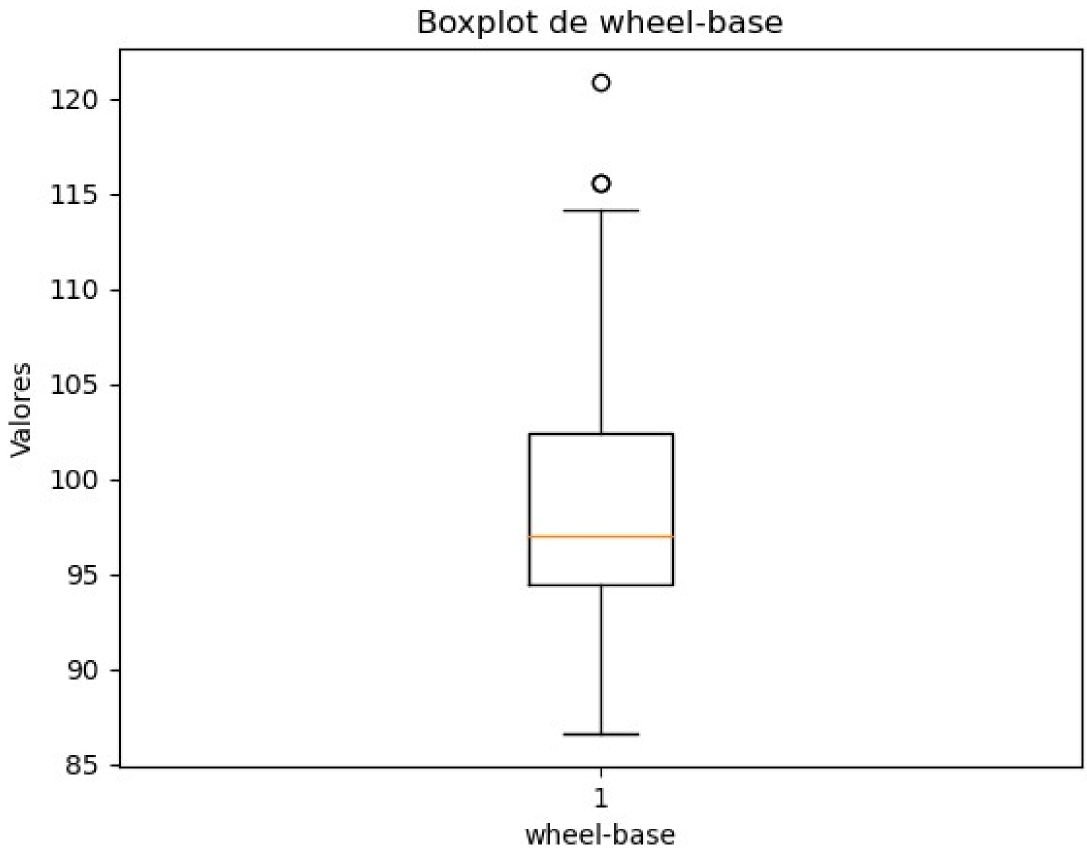
- wheel base y length: 0.91
- wheel base y width: 0.81
- wheel base y curb weigth: 0.77
- Length y width: 0.89

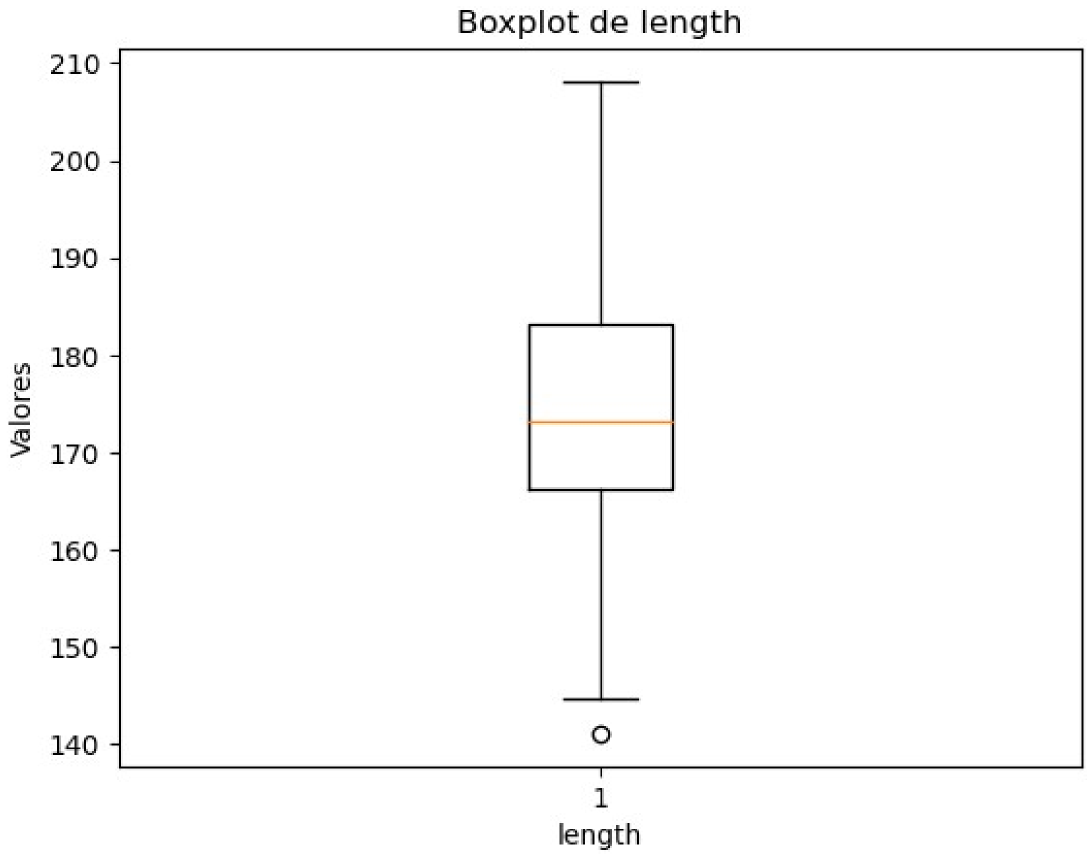
- Length y curb weigth: 0.89
- Length y engine size: 0.78
- Length y price: 0.81
- Length y highway mpg: -0.70
- Width y curb weigth: 0.86
- Width y engine size: 0.77
- Width y highway mpg: -0.70
- Width y price: 0.81
- curb-weight y engine size: 0.88
- curb-weight y bore: 0.70
- curb-weight y horse power: 0.81
- curb-weight y city mpg: -0.81
- curb-weight y highway mpg: -0.83
- curb-weight y price: 0.91
- engine-size y bore: 0.72
- engine-size y horse power: 0.82
- engine-size y city mpg: -0.73
- engine-size y highway mpg: -0.72
- engine-size y price: 0.83
- highway mpg y city mpg: 0.97

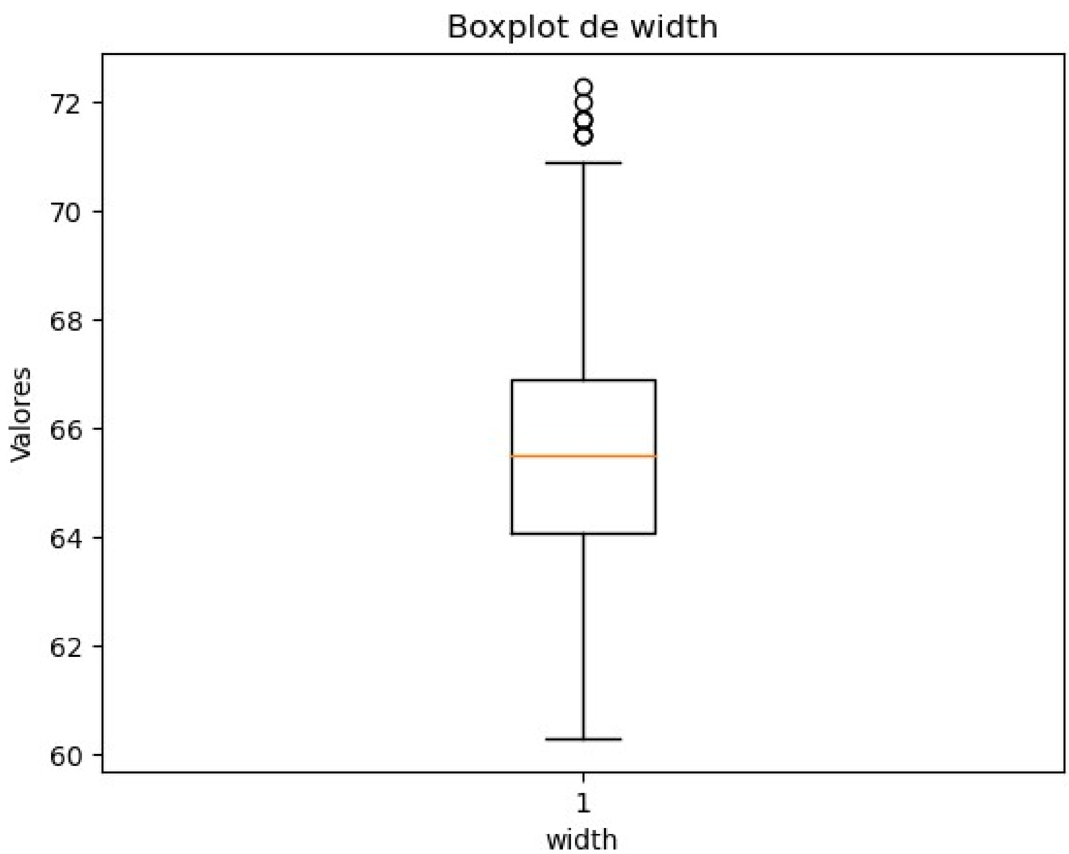
```
#Generamos graficas boxplots para las variables numericas con el fin
de analizar datos atipicos
numeric_columns = data.select_dtypes(include=['int', 'float']).columns
for column in numeric_columns:
    plt.boxplot(data[column])
    plt.xlabel(column)
    plt.ylabel('Valores')
    plt.title('Boxplot de ' + column)
    plt.show()
```


Boxplot de symboling
Valores
3
2
1
0
−1
−2
1
symboling

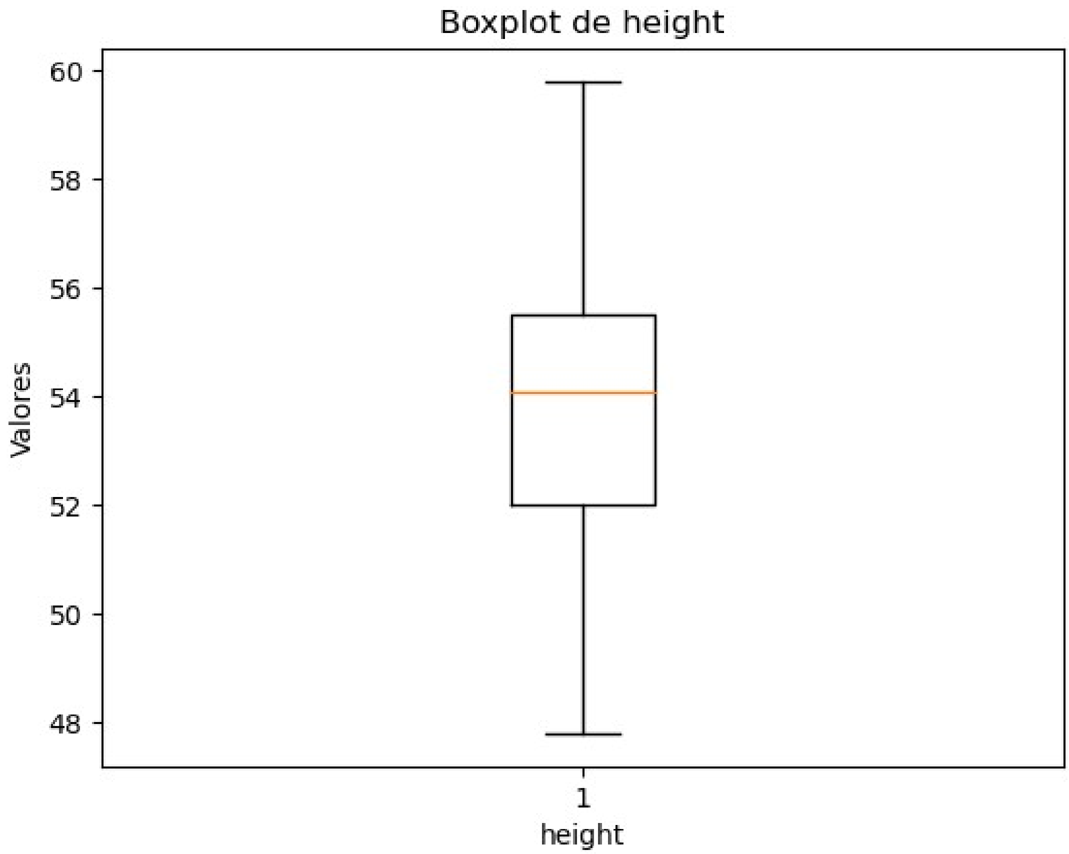

Boxplot de normalized-losses
Valores
0.04
0.02
0.00
−0.02
−0.04
1
normalized-losses

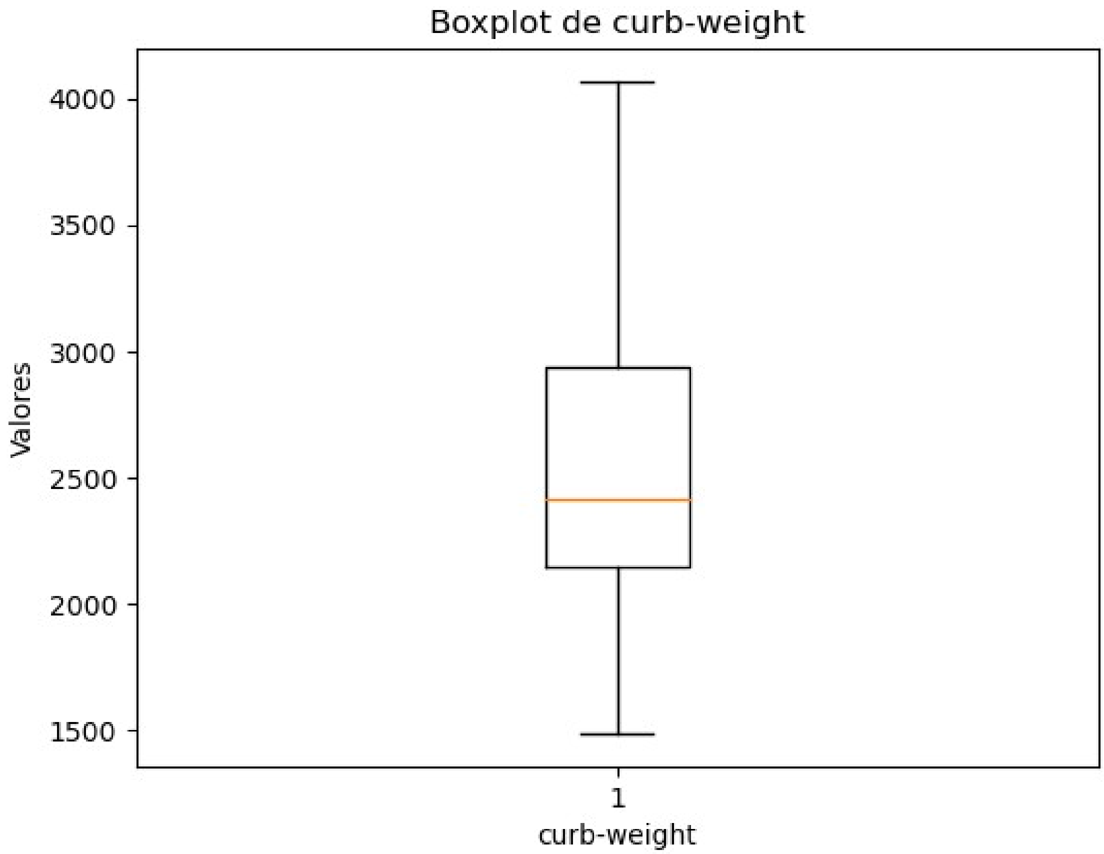

Boxplot de wheel-base
Valores
120
115
110
105
100
95
90
85
1
wheel-base

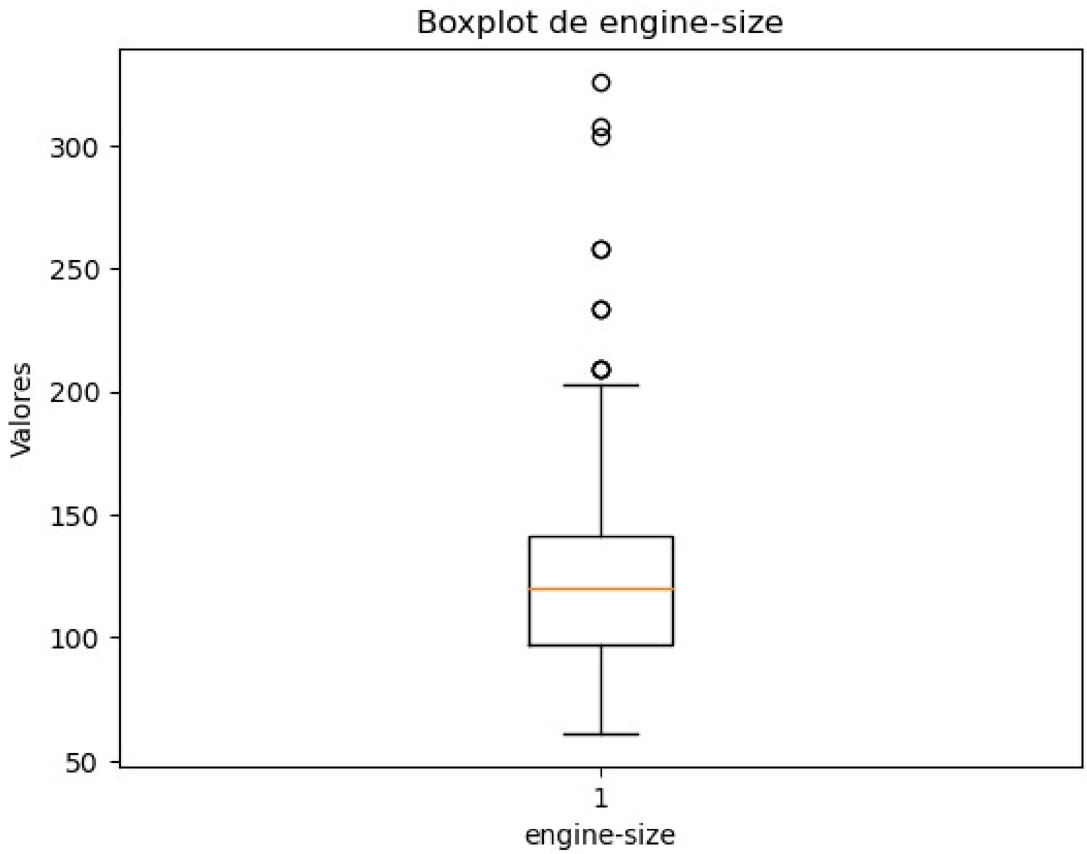

Boxplot de length
210
200
190
180
170
160
150
140
Valores
1
length

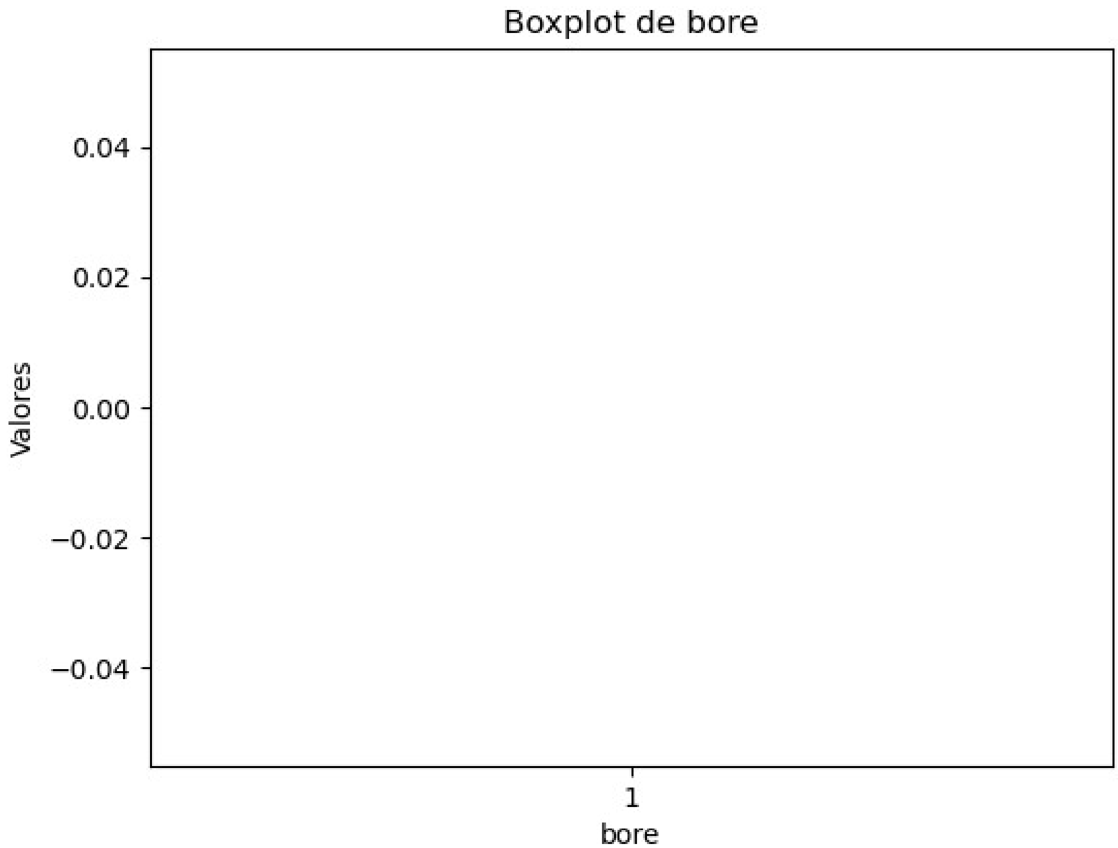


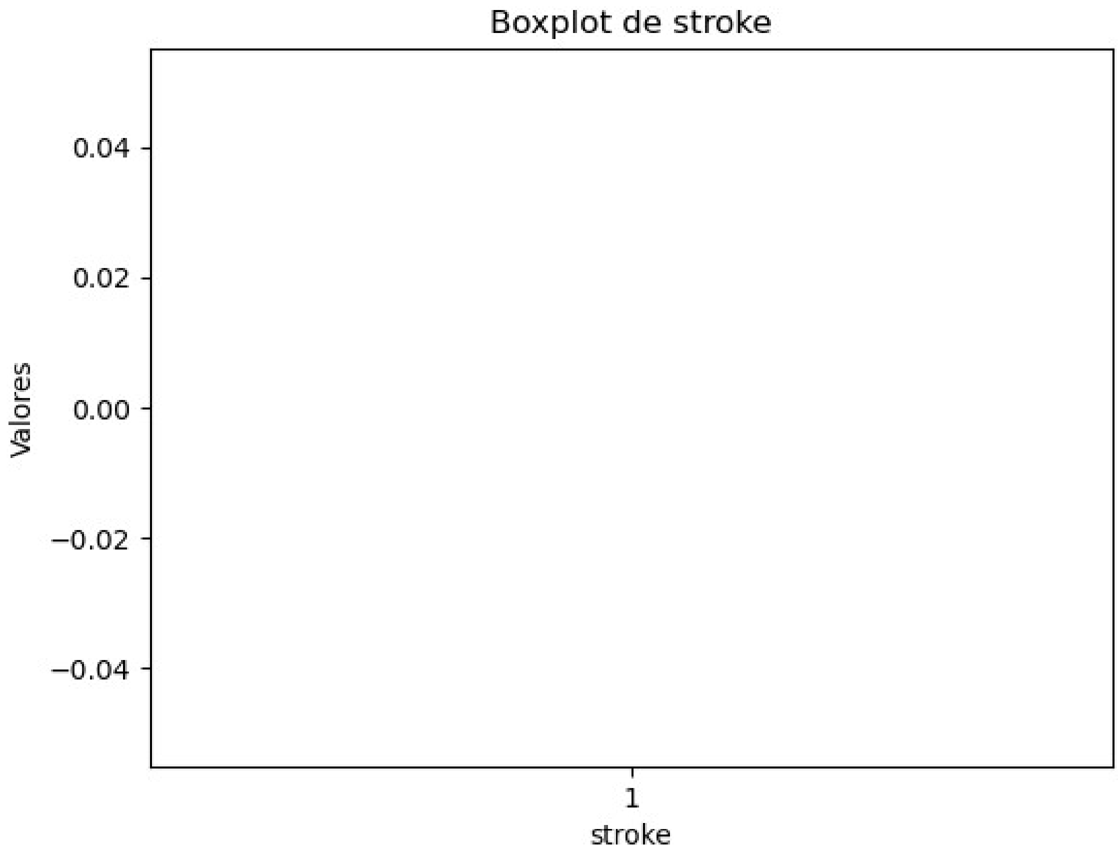
Boxplot de width
Valores
72
70
68
66
64
62
60
1
width

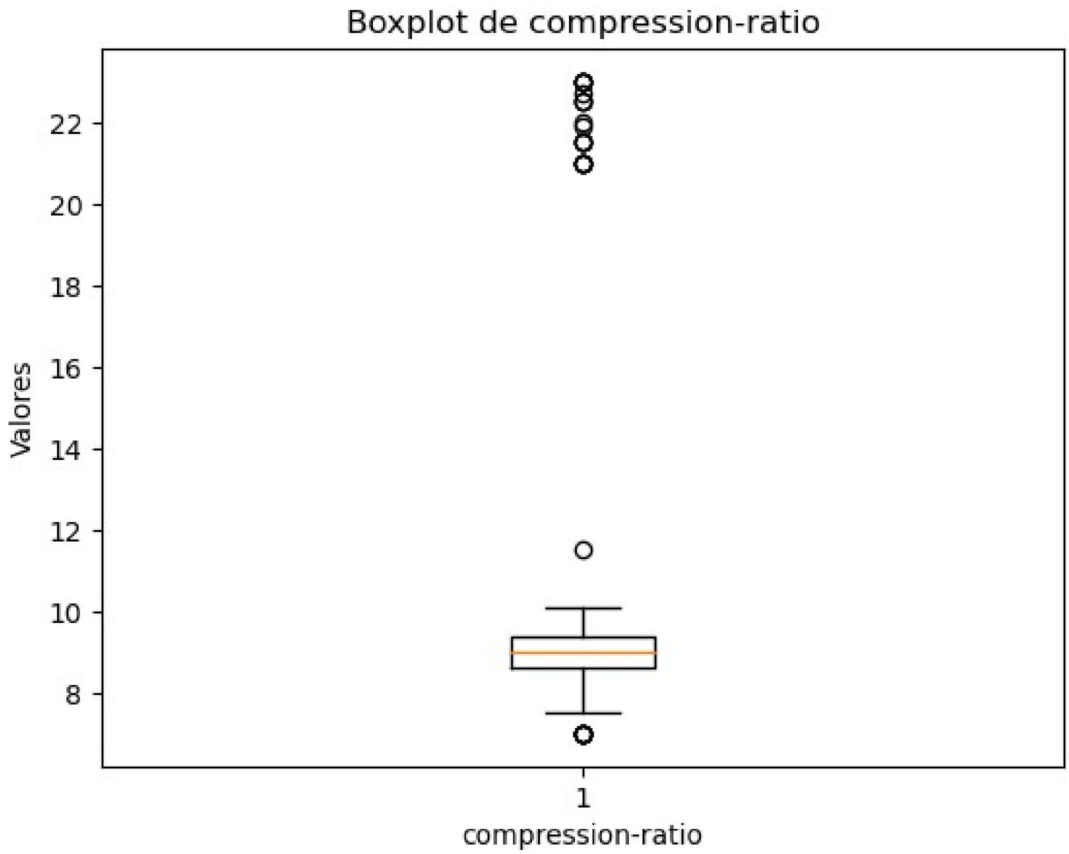

Boxplot de height
Valores
60
58
56
54
52
50
48
1
height

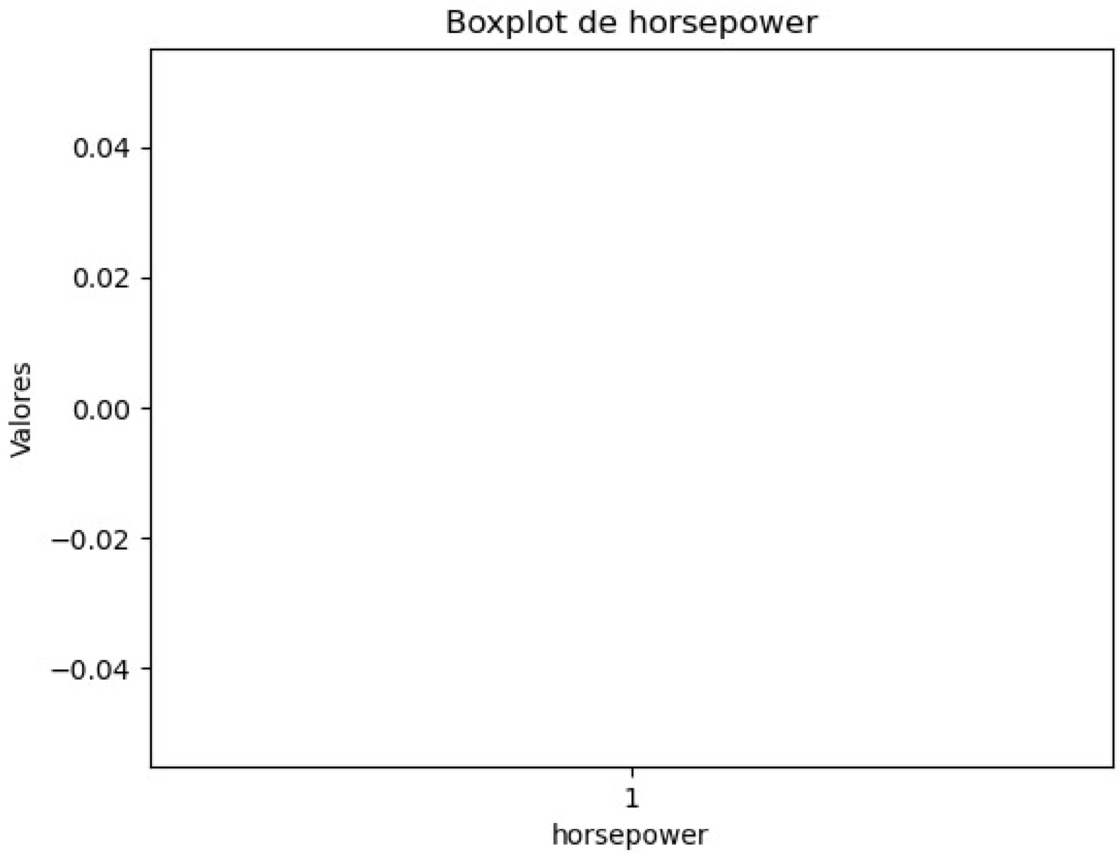

Boxplot de curb-weight
Valores
4000
3500
3000
2500
2000
1500
1
curb-weight


Boxplot de engine-size
Valores
300
250
200
150
100
50
1
engine-size

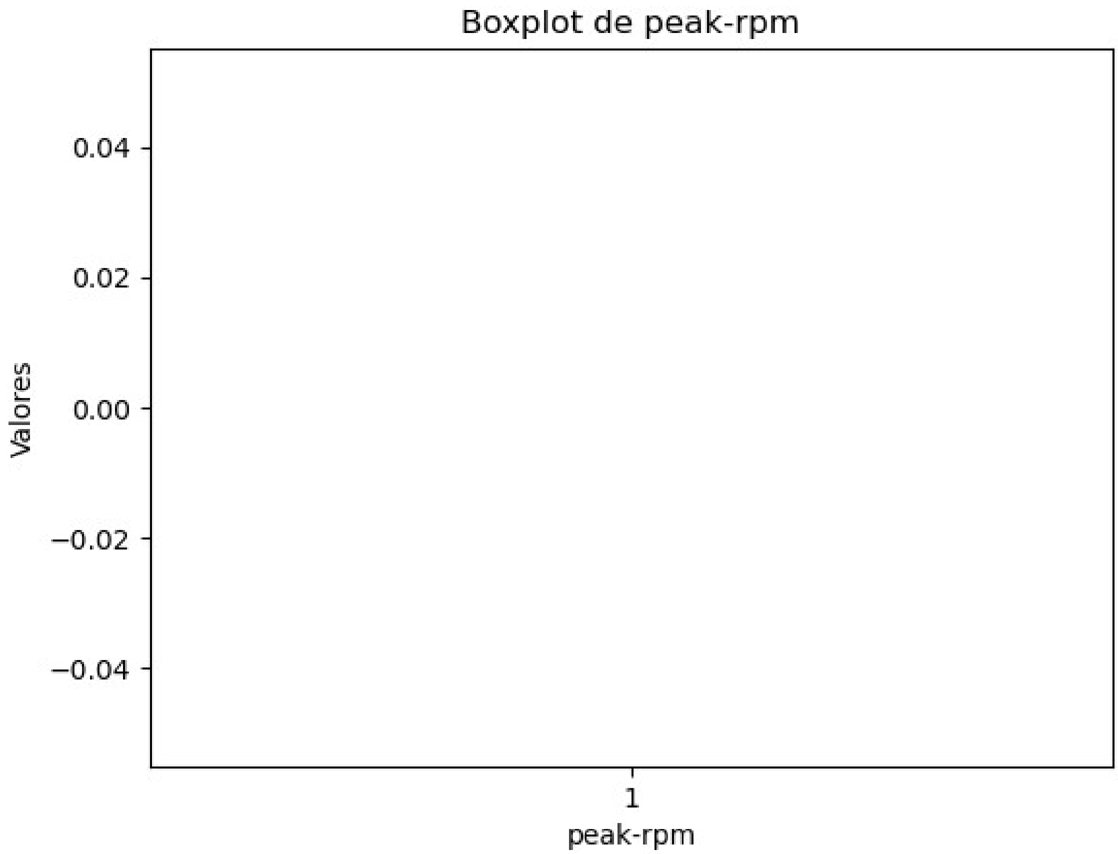

Boxplot de bore
0.04
0.02
0.00
−0.02
−0.04
Valores
1
bore

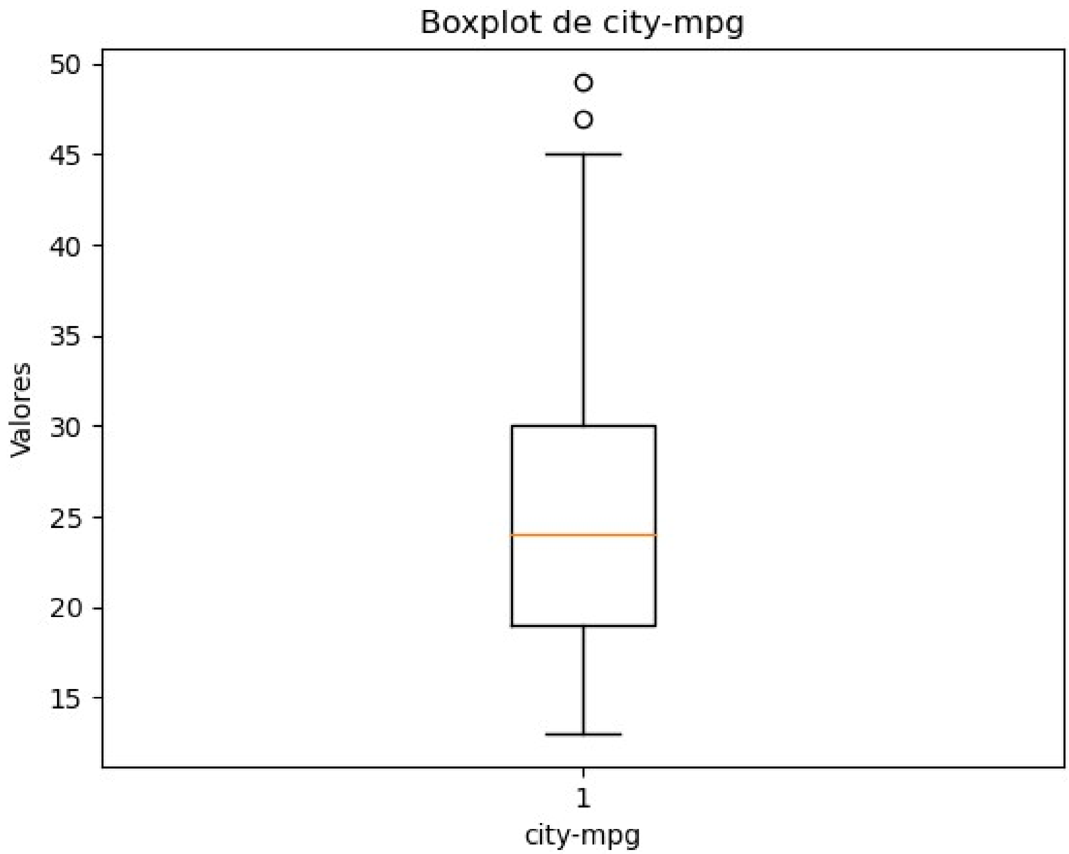

Boxplot de stroke
Valores
0.04
0.02
0.00
−0.02
−0.04
1
stroke

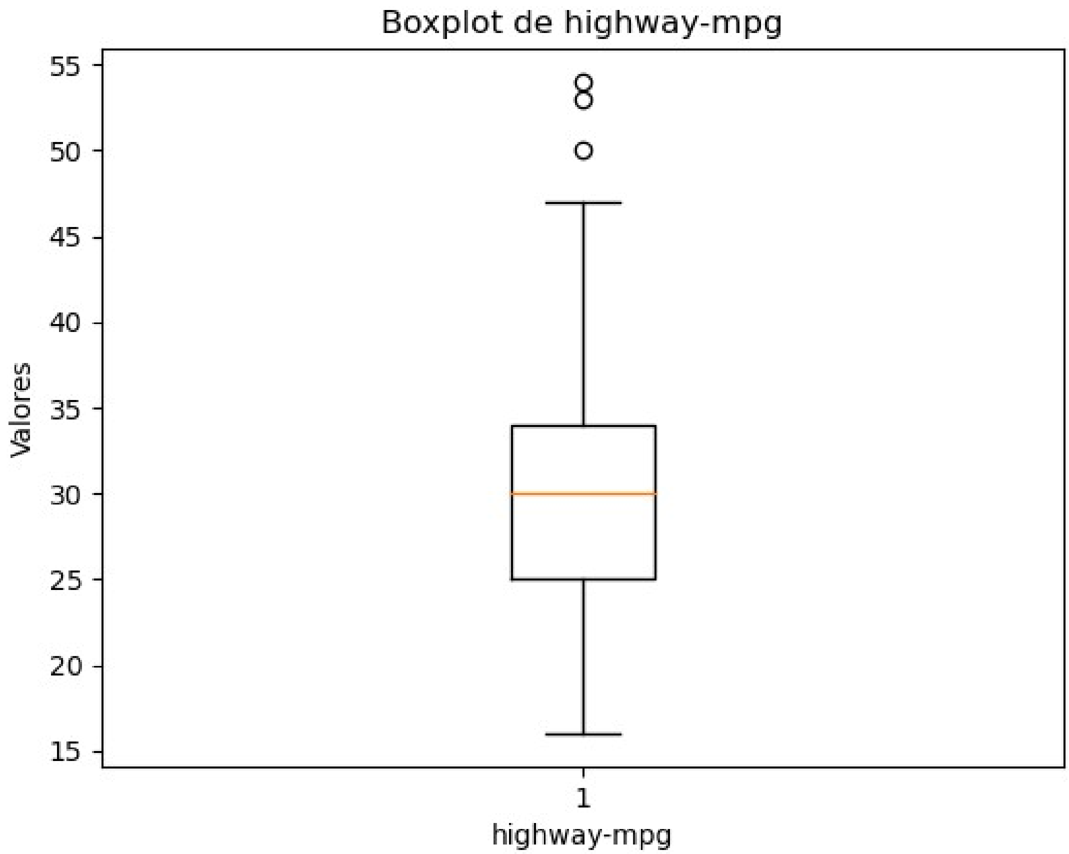

Boxplot de compression-ratio
Valores
22
20
18
16
14
12
10
8
1
compression-ratio

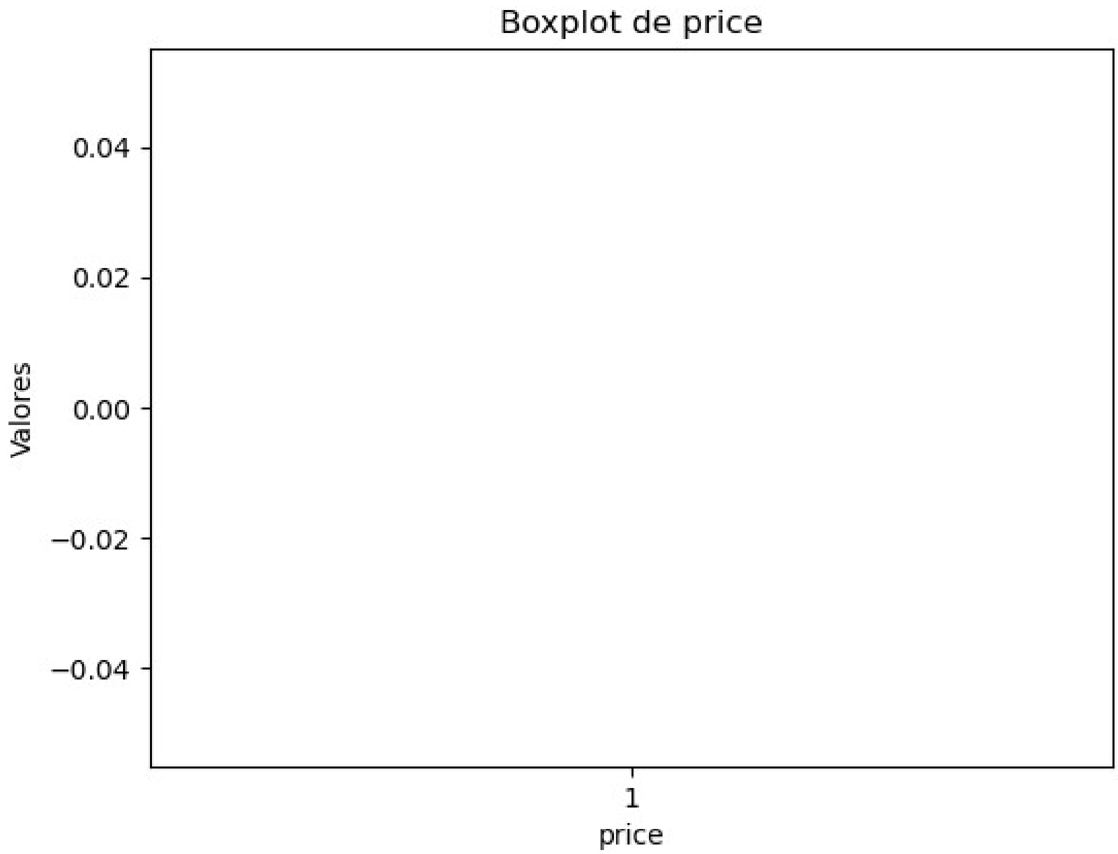


Boxplot de horsepower
0.04
0.02
0.00
−0.02
−0.04
Valores
1
horsepower


Boxplot de peak-rpm
0.04
0.02
0.00
−0.02
−0.04
Valores
1
peak-rpm


Boxplot de city-mpg
Valores
50
45
40
35
30
25
20
15
1
city-mpg


Boxplot de highway-mpg
Valores
55
50
45
40
35
30
25
20
15
1
highway-mpg




```
#Los atributos cuantitativos son los siguientes:
numeric_columns

Index(['symboling', 'normalized-losses', 'wheel-base', 'length',
'width',
       'height', 'curb-weight', 'engine-size', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')

#Creamos un dataset nuevo que contiene solamente los atributos
numericos:
data2=data[['normalized-losses', 'wheel-base', 'length', 'width',
'height',
       'curb-weight', 'engine-size', 'bore', 'stroke', 'compression-
ratio',
       'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price']]

#Normalizamos las escalas de todas las variables cuantitativas
normalizada= pd.DataFrame(preprocessing.scale(data2))

normalizada.columns=['normalized-losses', 'wheel-base', 'length',
'width', 'height',
       'curb-weight', 'engine-size', 'bore', 'stroke', 'compression-
```

```
ratio',
       'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price']
```

```
normalizada

     normalized-losses  wheel-base  length  width  height  curb-weight
\
0                  NaN       -1.69   -0.43  -0.84   -2.02        -0.01

1                  NaN       -1.69   -0.43  -0.84   -2.02        -0.01

2                  NaN       -0.71   -0.23  -0.19   -0.54         0.51

3                 1.19        0.17    0.21   0.14    0.24        -0.42

4                 1.19        0.11    0.21   0.23    0.24         0.52

..                 ...         ...     ...    ...     ...          ...

200              -0.76        1.72    1.20   1.40    0.73         0.76

201              -0.76        1.72    1.20   1.35    0.73         0.95

202              -0.76        1.72    1.20   1.40    0.73         0.88

203              -0.76        1.72    1.20   1.40    0.73         1.27

204              -0.76        1.72    1.20   1.40    0.73         0.98


     engine-size  bore  stroke  compression-ratio  horsepower  peak-
rpm  \
0           0.07  0.51   -1.82              -0.29        0.17      -
0.26
1           0.07  0.51   -1.82              -0.29        0.17      -
0.26
2           0.60 -2.38    0.68              -0.29        1.26      -
0.26
3          -0.43 -0.51    0.46              -0.04       -0.06
0.78
4           0.22 -0.51    0.46              -0.54        0.27
0.78
..           ...   ...     ...                ...         ...        ..
.
200         0.34  1.65   -0.33              -0.16        0.25
0.57
201         0.34  1.65   -0.33              -0.36        1.41
0.37
202         1.11  0.92   -1.22              -0.34        0.75
0.78
203         0.44 -1.17    0.46               3.24        0.04      -
```

```
0.68
204          0.34  1.65   -0.33              -0.16        0.25
0.57

     city-mpg  highway-mpg  price
0       -0.65        -0.55   0.04
1       -0.65        -0.55   0.42
2       -0.95        -0.69   0.42
3       -0.19        -0.11   0.09
4       -1.11        -1.27   0.54
..        ...          ...    ...
200     -0.34        -0.40   0.46
201     -0.95        -0.84   0.74
202     -1.11        -1.13   1.04
203      0.12        -0.55   1.17
204     -0.95        -0.84   1.19

[205 rows x 15 columns]
```

```
normalizada.std(axis=0)
```

```
normalized-losses   1.00
wheel-base          1.00
length              1.00
width               1.00
height              1.00
curb-weight         1.00
engine-size         1.00
bore                1.00
stroke              1.00
compression-ratio   1.00
horsepower          1.00
peak-rpm            1.00
city-mpg            1.00
highway-mpg         1.00
price               1.00
dtype: float64
```

```
normalizada.mean(axis=0)
```

```
normalized-losses    0.00
wheel-base          -0.00
length               0.00
width                0.00
height              -0.00
curb-weight          0.00
engine-size          0.00
bore                 0.00
stroke               0.00
compression-ratio   -0.00
```

```
horsepower          -0.00
peak-rpm             0.00
city-mpg             0.00
highway-mpg          0.00
price               -0.00
dtype: float64
```

```
#Eliminamos la columna Normalized losses ya que consideramos no es
relevante para la construccion del modelo,
#ademas tiene un 20% de missing values
normalizada = normalizada.drop("normalized-losses", axis=1)

#Eliminamos los valores Na de todas las columnas
normalizada=normalizada.dropna()

#Realizamos componentes principales con el nuevo dataset normalizado
pca = PCA()
pca.fit(normalizada)
```

```
PCA()
```

```
componentes_coef=pd.DataFrame(pca.components_)
```

```
pca.explained_variance_
```

```
array([7.51333306, 2.26460554, 1.20343145, 0.8950368 , 0.60919366,
       0.41239635, 0.31705757, 0.27127324, 0.1208348 , 0.10924733,
       0.08305822, 0.06249769, 0.05215879, 0.01924072])
```

```
var_exp=pca.explained_variance_ratio_ # varianza explicada por cada PC
cum_var_exp = np.cumsum(var_exp) # varianza acumulada por los primeros
n PCs
var_exp
```

```
array([0.5392332 , 0.16253113, 0.08637048, 0.06423694, 0.04372193,
       0.02959776, 0.02275528, 0.01946933, 0.00867233, 0.0078407 ,
       0.0059611 , 0.00448547, 0.00374345, 0.00138091])
```

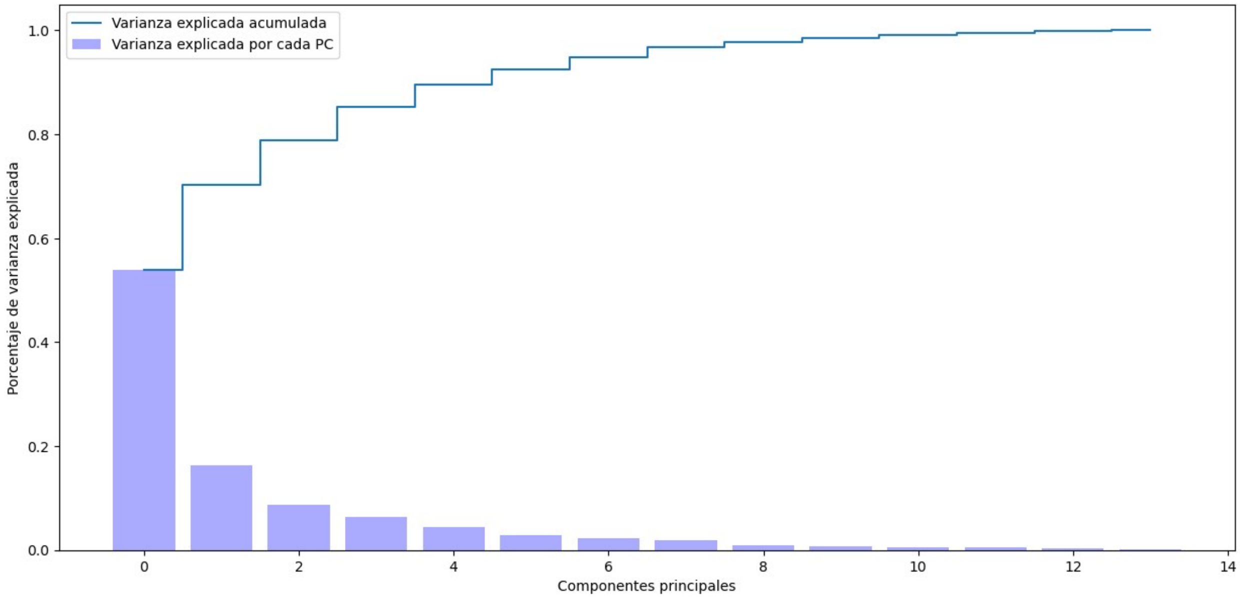
```
cum_var_exp
```

```
array([0.5392332 , 0.70176432, 0.7881348 , 0.85237175, 0.89609368,
       0.92569144, 0.94844671, 0.96791604, 0.97658837, 0.98442907,
       0.99039018, 0.99487565, 0.99861909, 1.        ])
```

```
dataPca = pca.transform(normalizada)
```

```
plt.figure(figsize=(15, 7))
plt.bar(range(len(var_exp)), var_exp, alpha=0.3333, align='center',
label='Varianza explicada por cada PC', color = 'blue')
plt.step(range(len(cum_var_exp)), cum_var_exp,
where='mid',label='Varianza explicada acumulada')
plt.ylabel('Porcentaje de varianza explicada')
```

```
plt.xlabel('Componentes principales')
plt.legend(loc='best')
plt.show()
```




```
np.sum(pca.explained_variance_ratio_[0:4])

0.8523717464905413
```

Los primeros 6 componentes principales juntos logran explicar el 85,23% de la varianza original

```
principalComponents = pca.fit_transform(normalizada)

#Creamos un dataset con todos los Eigenvectores
component_columns = ['principal component {}'.format(i) for i in
range(1, 15)]
principalDf = pd.DataFrame(data=principalComponents,
columns=component_columns)

principalDf

     principal component 1  principal component 2  principal component
3  \
0                    -0.66                  -2.14
0.25
1                    -0.53                  -2.17
0.20
2                     0.39                  -1.32                    -
1.46
3                    -0.18                  -0.23                    -
0.07
```

```
4                         1.25                     -1.17                   -
0.05
..                         ...                       ...                     .
..
190                       2.62                      0.39                    
1.09
191                       3.43                     -0.25                    
1.03
192                       3.45                     -0.52                    
1.44
193                       2.39                      3.04                   -
1.12
194                       3.26                      0.08                    
1.08

     principal component 4  principal component 5  principal component 
6  \
0                         2.42                      0.17                    
0.09
1                         2.47                      0.31                    
0.14
2                        -0.64                      0.41                    
1.94
3                        -1.13                      0.32                   -
0.14
4                        -1.18                      0.12                    
0.29
..                         ...                       ...                     .
..
190                      -0.45                      0.09                   -
0.97
191                      -0.33                     -0.00                   -
0.83
192                      -0.31                      0.88                   -
0.06
193                      -0.53                      2.01                    
0.60
194                      -0.45                      0.23                   -
0.85

     principal component 7  principal component 8  principal component 
9  \
0                        -0.52                     -0.97                   -
0.17
1                        -0.38                     -0.87                   -
0.08
2                         0.65                     -1.39                   -
0.18
3                        -0.17                     -0.08
```

```
0.16
4                      0.13                    -0.72
0.23
..                      ...                      ...                     .
..
190                   -0.85                     1.05
0.06
191                   -0.49                     0.59
0.01
192                   -0.53                     0.55                   -
0.06
193                   -0.58                    -1.03                   -
0.09
194                   -0.65                     0.72
0.30

     principal component 10  principal component 11  principal
component 12  \
0                     -0.14                     0.45
0.36
1                     -0.30                     0.63
0.30
2                      0.45                    -0.05
-0.38
3                     -0.10                     0.30
-0.22
4                     -0.41                     0.07
0.18
..                      ...                      ...
...
190                   -0.05                    -0.19
-0.02
191                    0.23                    -0.28
-0.78
192                   -0.26                    -0.37
0.08
193                   -0.41                    -0.05
-0.28
194                   -0.56                     0.14
-0.08

     principal component 13  principal component 14
0                      0.15                     0.01
1                      0.20                     0.04
2                      0.08                    -0.13
3                     -0.26                     0.02
4                      0.10                     0.07
..                      ...                      ...
190                    0.03                    -0.04
```

```
191                      0.01                     -0.07
192                     -0.35                     -0.12
193                     -0.00                      0.27
194                      0.14                     -0.16

[195 rows x 14 columns]
```

```
finalDf = pd.concat([principalDf, data[['symboling']]], axis = 1)
```

```
data["symboling"].unique()
```

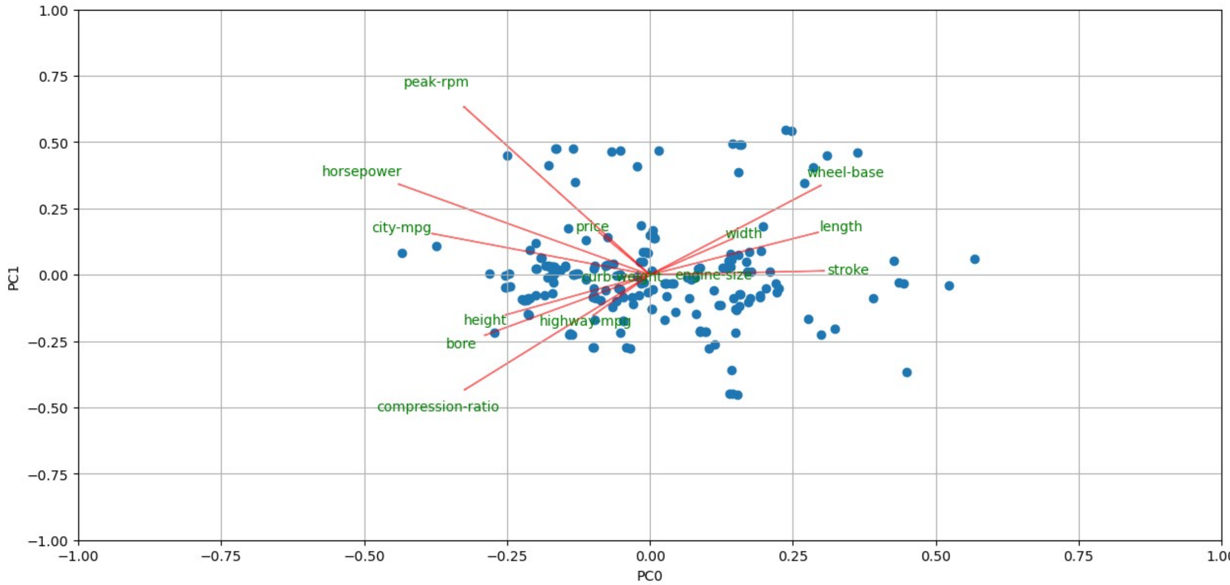
```
array([ 3,  1,  2,  0, -1, -2])
```

```
def biplot(data, loadings, index1, index2, labels=None):
    plt.figure(figsize=(15, 7))
    xs = data[:,index1]
    ys = data[:,index2]
    n=loadings.shape[0]
    scalex = 1.0/(xs.max()- xs.min())
    scaley = 1.0/(ys.max()- ys.min())
    plt.scatter(xs*scalex,ys*scaley)
    for i in range(n):
        plt.arrow(0, 0, loadings[i,index1],
loadings[i,index2],color='r',alpha=0.5)
        if labels is None:
            plt.text(loadings[i,index1]* 1.15, loadings[i,index2] *
1.15, "Var"+str(i+1), color='g', ha='center', va='center')
        else:
            plt.text(loadings[i,index1]* 1.15, loadings[i,index2] *
1.15, labels[i], color='g', ha='center', va='center')
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(index1))
    plt.ylabel("PC{}".format(index2))
    plt.grid()
```

```
normalizada.columns
```

```
Index(['wheel-base', 'length', 'width', 'height', 'curb-weight',
'engine-size',
       'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-
rpm',
       'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```

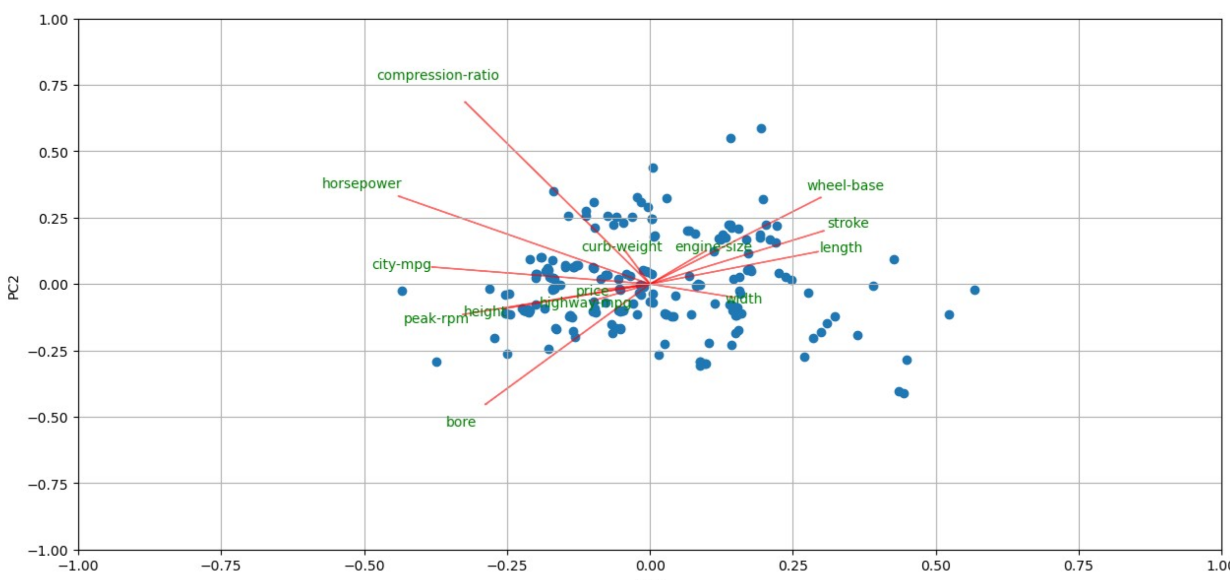
```
# PC0 VS PC1
biplot(dataPca, pca.components_, 0, 1,['wheel-base', 'length',
'width', 'height', 'curb-weight', 'engine-size',
       'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-
rpm',
       'city-mpg', 'highway-mpg', 'price'])
```



En el biplot se presenta el circulo unitario con la representacion de variables y observaciones,se evidencia que la variable mejor representada en en componente PC0 es stroke. Las demas varibles parecen ser explicadas por una combinacion lineal de PC0 Y PC1.

Las variables width, price , engine-size, curb-weight y highway-mpg parecen tener poca significancia debido a la corta longitud de su linea.

```
# PC0 VS PC2
biplot(dataPca, pca.components_, 0, 2,['wheel-base', 'length',
'width', 'height', 'curb-weight', 'engine-size',
        'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-
rpm',
        'city-mpg', 'highway-mpg', 'price'])
```



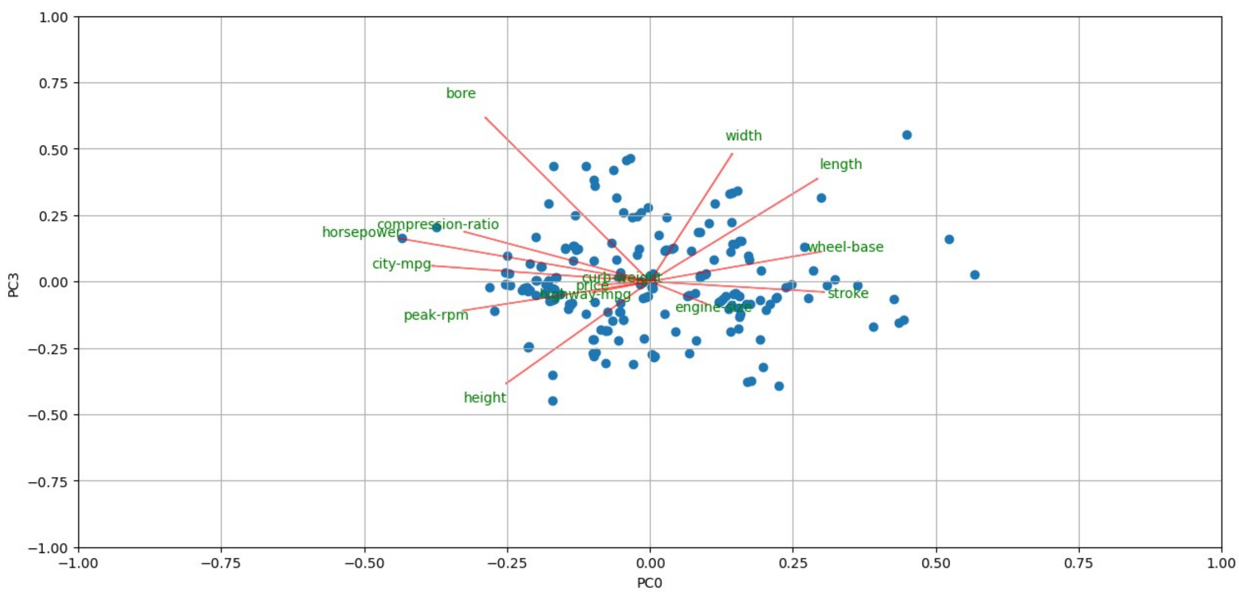
En el biplot se presenta el circulo unitario con la representacion de variables y observaciones, se evidencia que la variable mejor representada por PC0 es city mpg. Las demas varibles parecen ser explicadas por una combinacion lineal de PC0 Y PC2.

Las variablesprice,curb-weight, width y highway-mpg parecen tener poca significancia debido a la corta longitud de su linea.

```
normalizada.columns
```

```
Index(['wheel-base', 'length', 'width', 'height', 'curb-weight',
'engine-size',
       'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-
rpm',
       'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```

```
#PC0 vs PC3
biplot(dataPca, pca.components_, 0, 3,['wheel-base', 'length',
'width', 'height', 'curb-weight', 'engine-size',
       'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-
rpm',
       'city-mpg', 'highway-mpg', 'price'])
```



En el biplot se presenta el circulo unitario con la representacion de variables y observaciones, se evidencia que la variable mejor representada por PC0 es stroke y city mpg. Las demas varibles parecen ser explicadas por una combinacion lineal de PC0 Y PC3.

Las variables que se decide eliminar por que no contribuyen en la creacion de las componentes: price, curb-weight, highway-mpg, width, engine-size

Modelo

```
data=pd.read_csv("imports-85.data", header=None)
data.columns = [
    "symboling", "normalized-losses", "make", "fuel-type",
"aspiration", "num-of-doors", "body-style", "drive-wheels",
    "engine-location", "wheel-base", "length", "width", "height",
"curb-weight", "engine-type", "num-of-cylinders",
    "engine-size", "fuel-system", "bore", "stroke", "compression-
ratio", "horsepower", "peak-rpm", "city-mpg", "highway-mpg",
    "price"
]

data=data.replace("?",np.nan)

data.drop("normalized-losses", axis=1)

     symboling         make fuel-type aspiration num-of-doors   body-
style  \
0            3  alfa-romero       gas        std          two
convertible
1            3  alfa-romero       gas        std          two
convertible
2            1  alfa-romero       gas        std          two
hatchback
3            2         audi       gas        std         four
sedan
4            2         audi       gas        std         four
sedan
..         ...          ...       ...        ...          ...
...
200         -1        volvo       gas        std         four
sedan
201         -1        volvo       gas      turbo         four
sedan
202         -1        volvo       gas        std         four
sedan
203         -1        volvo    diesel      turbo         four
sedan
204         -1        volvo       gas      turbo         four
sedan

    drive-wheels engine-location  wheel-base  length  ...  engine-size
\
0            rwd           front       88.60  168.80  ...          130

1            rwd           front       88.60  168.80  ...          130

2            rwd           front       94.50  171.20  ...          152

3            fwd           front       99.80  176.60  ...          109
```

```
4              4wd            front       99.40  176.60  ...          136
..             ...              ...         ...     ...  ...          ...
200            rwd            front      109.10  188.80  ...          141
201            rwd            front      109.10  188.80  ...          141
202            rwd            front      109.10  188.80  ...          173
203            rwd            front      109.10  188.80  ...          145
204            rwd            front      109.10  188.80  ...          141

     fuel-system  bore stroke compression-ratio  horsepower peak-rpm
city-mpg  \
0           mpfi  3.47   2.68              9.00         111     5000
21
1           mpfi  3.47   2.68              9.00         111     5000
21
2           mpfi  2.68   3.47              9.00         154     5000
19
3           mpfi  3.19   3.40             10.00         102     5500
24
4           mpfi  3.19   3.40              8.00         115     5500
18
..           ...   ...    ...               ...         ...      ...
...
200         mpfi  3.78   3.15              9.50         114     5400
23
201         mpfi  3.78   3.15              8.70         160     5300
19
202         mpfi  3.58   2.87              8.80         134     5500
18
203          idi  3.01   3.40             23.00         106     4800
26
204         mpfi  3.78   3.15              9.50         114     5400
19

    highway-mpg  price
0            27  13495
1            27  16500
2            26  16500
3            30  13950
4            22  17450
..          ...    ...
200          28  16845
201          25  19045
```

```
202          23  21485
203          27  22470
204          25  22625

[205 rows x 25 columns]
```

```
data["price"]=data["price"].astype(float)
data["bore"]=data["bore"].astype(float)
data["stroke"]=data["stroke"].astype(float)
data["horsepower"]=data["horsepower"].astype(float)
data["peak-rpm"]=data["peak-rpm"].astype(float)
data["price"]=data["price"].astype(float)
data["symboling"]=data["symboling"].astype(object)

data["symboling"].value_counts()
```

```
 0    67
 1    54
 2    32
 3    27
-1    22
-2     3
Name: symboling, dtype: int64
```

```
#Agrupamos los vehiculos segun su riesgo para reducir el numero de categorias
data['Target'] = np.where((data['symboling'] >= 1) & (data['symboling'] <= 3), 'Riesgoso',
                             np.where((data['symboling'] >= -3) & (data['symboling'] <= 0), 'Seguro', 'Otro'))

data["Target"].value_counts()
```

```
Riesgoso    113
Seguro       92
Name: Target, dtype: int64
```

```
## al tener una distribucion balanceada, se comienza a modelar
#price, curb-weight, highway-mpg, width, engine-size
#make, num-of-doors, body-style (agrupar sedan vs otras),fuel-system (idi y las otras)

#Agrupamos los vehiculos con body style hardtop y convertible juntos
data["body-style"] = data["body-style"].replace({'sedan': 'sedan', 'hatchback': 'hatchback', 'wagon':
'wagon','hardtop':'others','convertible':'others'})

#Agrupamos los vehiculos con fuel system idi, 1bbl, spdi, 4bbl, mfi y spfi juntos
data["fuel-system"] = data["fuel-system"].replace({'mpfi': 'mpfi', '2bbl': '2bbl', 'idi':
```

```
'others','1bbl':'others','spdi':'others','4bbl':'others','mfi':'others
','spfi':'others'})

#Aplicamos one hot encoding para las variables categoricas
seleccionadas
encoded_data = pd.get_dummies(data[["body-style","make","num-of-
doors","fuel-system"]])

#Se concatenan ambos datasets
datas=pd.concat([data[['wheel-base', 'length', 'height', 'bore',
'stroke', 'compression-ratio',
       'horsepower', 'peak-rpm', 'city-mpg']],encoded_data], axis=1)

# Se eliminan los valos na
datas=datas.dropna()

#Definimos las variables independientes y la variable dependiente
X=datas
y=data["Target"]

X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199 entries, 0 to 204
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   wheel-base             199 non-null    float64
 1   length                 199 non-null    float64
 2   height                 199 non-null    float64
 3   bore                   199 non-null    float64
 4   stroke                 199 non-null    float64
 5   compression-ratio      199 non-null    float64
 6   horsepower             199 non-null    float64
 7   peak-rpm               199 non-null    float64
 8   city-mpg               199 non-null    int64
 9   body-style_hatchback   199 non-null    uint8
 10  body-style_others      199 non-null    uint8
 11  body-style_sedan       199 non-null    uint8
 12  body-style_wagon       199 non-null    uint8
 13  make_alfa-romero       199 non-null    uint8
 14  make_audi              199 non-null    uint8
 15  make_bmw               199 non-null    uint8
 16  make_chevrolet         199 non-null    uint8
 17  make_dodge             199 non-null    uint8
 18  make_honda             199 non-null    uint8
 19  make_isuzu             199 non-null    uint8
 20  make_jaguar            199 non-null    uint8
 21  make_mazda             199 non-null    uint8
 22  make_mercedes-benz     199 non-null    uint8
 23  make_mercury           199 non-null    uint8
```

```
 24  make_mitsubishi       199 non-null    uint8
 25  make_nissan           199 non-null    uint8
 26  make_peugot           199 non-null    uint8
 27  make_plymouth         199 non-null    uint8
 28  make_porsche          199 non-null    uint8
 29  make_renault          199 non-null    uint8
 30  make_saab             199 non-null    uint8
 31  make_subaru           199 non-null    uint8
 32  make_toyota           199 non-null    uint8
 33  make_volkswagen       199 non-null    uint8
 34  make_volvo            199 non-null    uint8
 35  num-of-doors_four     199 non-null    uint8
 36  num-of-doors_two      199 non-null    uint8
 37  fuel-system_2bbl      199 non-null    uint8
 38  fuel-system_mpfi      199 non-null    uint8
 39  fuel-system_others    199 non-null    uint8
dtypes: float64(8), int64(1), uint8(31)
memory usage: 21.6 KB
```

```
#Esta libreria nos permite correr varios modelos ed Machine learning
al tiempo y comparar sus metricas.
X, y = make_classification(n_samples=1000, n_features=20,
random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


clf = LazyClassifier(verbose=0, ignore_warnings=True)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)
models
```

```
100%|██████████| 29/29 [00:01<00:00, 22.03it/s]
```

```
                               Accuracy  Balanced Accuracy  ROC AUC
F1 Score  \
Model

RandomForestClassifier             0.90               0.90     0.90
0.90
XGBClassifier                      0.89               0.89     0.89
0.89
BaggingClassifier                  0.89               0.89     0.89
0.88
LGBMClassifier                     0.89               0.89     0.89
0.89
DecisionTreeClassifier             0.88               0.88     0.88
0.88
ExtraTreesClassifier               0.87               0.87     0.87
0.87
```

```
AdaBoostClassifier                 0.87                0.87     0.87
0.87
CalibratedClassifierCV             0.86                0.87     0.87
0.87
LinearSVC                          0.86                0.86     0.86
0.86
LinearDiscriminantAnalysis         0.85                0.86     0.86
0.85
RidgeClassifierCV                  0.85                0.86     0.86
0.85
RidgeClassifier                    0.85                0.86     0.86
0.85
LogisticRegression                 0.85                0.86     0.86
0.86
NuSVC                              0.84                0.85     0.85
0.84
SVC                                0.84                0.85     0.85
0.85
Perceptron                         0.83                0.83     0.83
0.83
NearestCentroid                    0.82                0.83     0.83
0.82
BernoulliNB                        0.81                0.81     0.81
0.80
SGDClassifier                      0.81                0.81     0.81
0.81
GaussianNB                         0.80                0.80     0.80
0.79
PassiveAggressiveClassifier        0.80                0.80     0.80
0.80
ExtraTreeClassifier                0.79                0.80     0.80
0.79
QuadraticDiscriminantAnalysis      0.80                0.79     0.79
0.79
KNeighborsClassifier               0.79                0.79     0.79
0.78
LabelSpreading                     0.77                0.78     0.78
0.77
LabelPropagation                   0.77                0.78     0.78
0.77
DummyClassifier                    0.47                0.50     0.50
0.30

                               Time Taken
Model
RandomForestClassifier               0.19
XGBClassifier                        0.12
BaggingClassifier                    0.09
LGBMClassifier                       0.11
```

```
DecisionTreeClassifier                      0.02
ExtraTreesClassifier                        0.12
AdaBoostClassifier                          0.19
CalibratedClassifierCV                      0.11
LinearSVC                                   0.03
LinearDiscriminantAnalysis                  0.02
RidgeClassifierCV                           0.01
RidgeClassifier                             0.02
LogisticRegression                          0.01
NuSVC                                       0.04
SVC                                         0.03
Perceptron                                  0.01
NearestCentroid                             0.01
BernoulliNB                                 0.01
SGDClassifier                               0.01
GaussianNB                                  0.01
PassiveAggressiveClassifier                 0.02
ExtraTreeClassifier                         0.01
QuadraticDiscriminantAnalysis               0.01
KNeighborsClassifier                        0.02
LabelSpreading                              0.05
LabelPropagation                            0.04
DummyClassifier                             0.01
```

Random Forest parece ser el mejor modelo, vamos a comprobarlo

MODELO 1 RANDOM FOREST

```
forest = RandomForestClassifier(criterion='gini',
                                 n_estimators=5,
                                 random_state=1,
                                 n_jobs=2)


forest.fit(X_train, y_train)


y_pred = forest.predict(X_test)
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))

Accuracy: 0.850

from sklearn.metrics import cohen_kappa_score
kappa = cohen_kappa_score(y_test, y_pred)
print("Coeficiente de Kappa:", kappa)

Coeficiente de Kappa: 0.7010463378176383
```

Utilizando un modelo Random Forest obtenemos una clasificacion con un accuracy de 85% y un Kappa de 0.701, lo que nos permite concluir que tenemos un buen modelo mucho mejor que el baseline

```
feature_importance = forest.feature_importances_

#Obtenemos la relevancia de las variables predictoras
feature_importance

array([0.02159787, 0.1497592 , 0.03012585, 0.01857064, 0.00620078,
       0.18341466, 0.01649461, 0.01845453, 0.02005582, 0.01282518,
       0.02649634, 0.03763293, 0.02080572, 0.01594713, 0.07845535,
       0.01681059, 0.02133975, 0.01126815, 0.28096426, 0.01278065])

X

array([[-0.6693561 , -1.49577819, -0.87076638, ..., -1.26733697,
        -1.2763343 ,  1.01664321],
       [ 0.09337237,  0.78584826,  0.10575379, ..., -0.12270893,
         0.6934308 ,  0.91136272],
       [-0.90579721, -0.60834121,  0.29514098, ...,  0.83049813,
        -0.73733198, -0.5782121 ],
       ...,
       [-0.20013455, -1.46108168,  1.79701652, ..., -1.50280171,
        -1.27473745,  1.60111869],
       [ 0.03935575,  0.24868361, -0.47532342, ...,  0.09912579,
         0.54269228,  1.20827474],
       [ 0.76921528,  0.47076539,  0.16994471, ...,  0.6561162 ,
         0.64333186, -2.02100232]])

import plotly.graph_objects as go
import plotly.io as pio

pio.renderers.default =  "notebook"  # Elige el renderizador por
defecto (cambia a "notebook" si usas Jupyter Notebook)

fig = go.Figure(data=go.Bar(x=datas.columns, y=feature_importance))
fig.update_layout(title="Feature Importance Random Forest",
xaxis_title="Feature", yaxis_title="Importance")
fig.show()
```

Podemos concluir que las variables predictoras mas relevantes son: Si el vehiculo es de marca honda, compression ratio, Longitud del vehiculo y si el vehiculo es de marca Audi

MODELO 2 XGBOOST

```
!pip install xgboost

Requirement already satisfied: xgboost in c:\users\mqa200-0489\
anaconda3\lib\site-packages (1.7.5)
Requirement already satisfied: scipy in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from xgboost) (1.9.1)
Requirement already satisfied: numpy in c:\users\mqa200-0489\
anaconda3\lib\site-packages (from xgboost) (1.21.5)
```

```python
from xgboost import XGBClassifier

modelGB = XGBClassifier(gamma=2.1784702961406848,
                        learning_rate=0.06271852908557515,
                        max_depth=4,
                        n_estimators=45,
                        subsample=0.7856261494444738)
modelGB.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=2.1784702961406848, gpu_id=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.06271852908557515, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              n_estimators=45, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

```python
# make predictions for test data
y_pred = modelGB.predict(X_test)
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 90.50%
```

```python
kappa = cohen_kappa_score(y_test, predictions)
print("Coeficiente de Kappa:", kappa)
```

```
Coeficiente de Kappa: 0.8105305145592341
```

Utilizando un modelo XGBOOST una clasificacion con un accuracy de 90.5% y un Kappa de 0.810, lo que nos permite concluir que tenemos un buen modelo mucho mejor que el baseline

```python
#Obtenemos la relevancia de las variables predictoras
feature_importance1 = modelGB.feature_importances_

fig = go.Figure(data=go.Bar(x=datas.columns, y=feature_importance1))
fig.update_layout(title="Feature Importance XGBOOST ",
xaxis_title="Feature", yaxis_title="Importance")
fig.show()
```

Podemos concluir que las variables predictoras mas relevantes son: Compression ratio y Si el vehiculo es de marca honda

MODELO 3 Gradient boosting

```
gb_clf = GradientBoostingClassifier()
gb_clf.fit(X_train, y_train)
```

```
GradientBoostingClassifier()
```

```
feature_importance2 =gb_clf.feature_importances_
y_pred = gb_clf.predict(X_test)
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.910
```

```
kappa = cohen_kappa_score(y_test, y_pred)
print("Coeficiente de Kappa:", kappa)
```

```
Coeficiente de Kappa: 0.8206278026905829
```

Utilizando un modelo Gradient Boosting una clasificacion con un accuracy de 91% y un Kappa de 0.820, lo que nos permite concluir que tenemos un buen modelo mucho mejor que el baseline

```
fig = go.Figure(data=go.Bar(x=datas.columns, y=feature_importance2))
fig.update_layout(title="Feature Importance Gradient Boosting",
xaxis_title="Feature", yaxis_title="Importance")
fig.show()
```

Podemos concluir que las variables predictoras mas relevantes son: Compression ratio y Si el vehiculo es de marca audi

De los 3 modelos realizados, Gradient Boosting obtuvo los mejores resultados en Accuracy y kappa

Clusters

```
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

```python
datas=data[['wheel-base', 'length', 'height', 'bore', 'stroke',
'compression-ratio',
       'horsepower', 'peak-rpm', 'city-mpg']]

#Normalizamos las escalas de las variables
dataStd = pd.DataFrame(preprocessing.scale(datas))
dataStd.columns=datas.columns

dataStd.describe()
```

```
       wheel-base  length  height   bore  stroke  compression-ratio  \
count      205.00  205.00  205.00 201.00  201.00             205.00
mean        -0.00    0.00   -0.00   0.00    0.00              -0.00
std          1.00    1.00    1.00   1.00    1.00               1.00
min         -2.02   -2.68   -2.43  -2.89   -3.75              -0.79
25%         -0.71   -0.63   -0.71  -0.66   -0.46              -0.39
50%         -0.29   -0.07    0.15  -0.07    0.11              -0.29
75%          0.61    0.74    0.73   0.95    0.49              -0.19
max          3.69    2.77    2.49   2.24    2.89               3.24

       horsepower  peak-rpm  city-mpg
count      203.00    203.00    205.00
mean        -0.00      0.00      0.00
std          1.00      1.00      1.00
min         -1.42     -2.04     -1.87
25%         -0.86     -0.68     -0.95
50%         -0.23      0.16     -0.19
75%          0.30      0.78      0.73
max          4.64      3.08      3.64
```

```python
mean_values = dataStd.mean(axis=0)
mean_values_formatted = mean_values.round(12)
print(mean_values_formatted)
```

```
wheel-base          -0.00
length               0.00
height              -0.00
bore                 0.00
stroke               0.00
compression-ratio   -0.00
horsepower          -0.00
peak-rpm             0.00
city-mpg             0.00
dtype: float64
```

```python
dataStd.mean(axis=0)
```

```
wheel-base          -0.00
length               0.00
height              -0.00
bore                 0.00
```

```
stroke                0.00
compression-ratio    -0.00
horsepower           -0.00
peak-rpm              0.00
city-mpg              0.00
dtype: float64
```

```
dataStd.std(axis=0)
```

```
wheel-base          1.00
length              1.00
height              1.00
bore                1.00
stroke              1.00
compression-ratio   1.00
horsepower          1.00
peak-rpm            1.00
city-mpg            1.00
dtype: float64
```

```
#Eliminamos los valores na de todas las columnas
dataStd=dataStd.dropna()
```

Optmización del numeros de clusters

KMEANS

Metodo del codo:

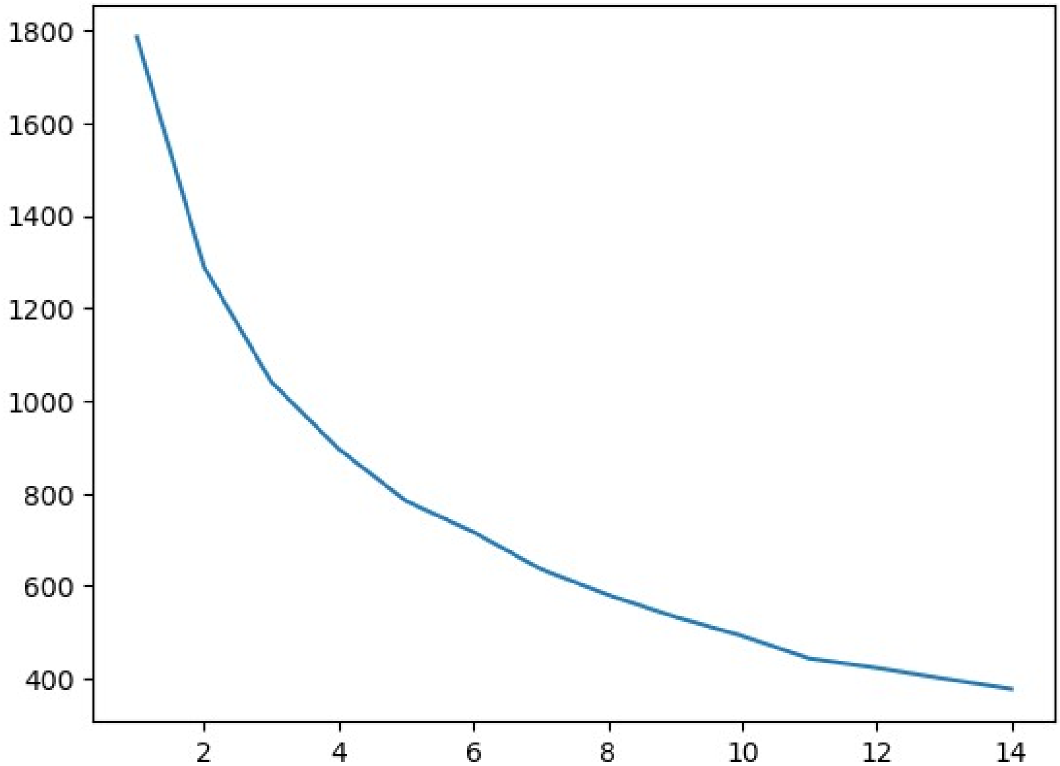
```
WSSs = []
for i in range(1,15) :
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(dataStd)
    WSSs.append(km.inertia_)
WSSs
```

```
[1785.5661430309424,
 1288.275276600507,
 1040.7090214947486,
 896.0345069319069,
 783.7834631140512,
 716.8104966874055,
 636.8703227885688,
 581.1104240782172,
 533.4665968409798,
 492.2208676385021,
 443.51001421691086,
 423.2454229037552,
```

```
 400.159891009055,
 377.8037356390405]
```

```
plt.plot(range(1, 15), WSSs)
```

```
[<matplotlib.lines.Line2D at 0x1e2763b37c0>]
```



En el criterio del codo se ve que el numero optimo de k es 2.

Calinski-Harabaz

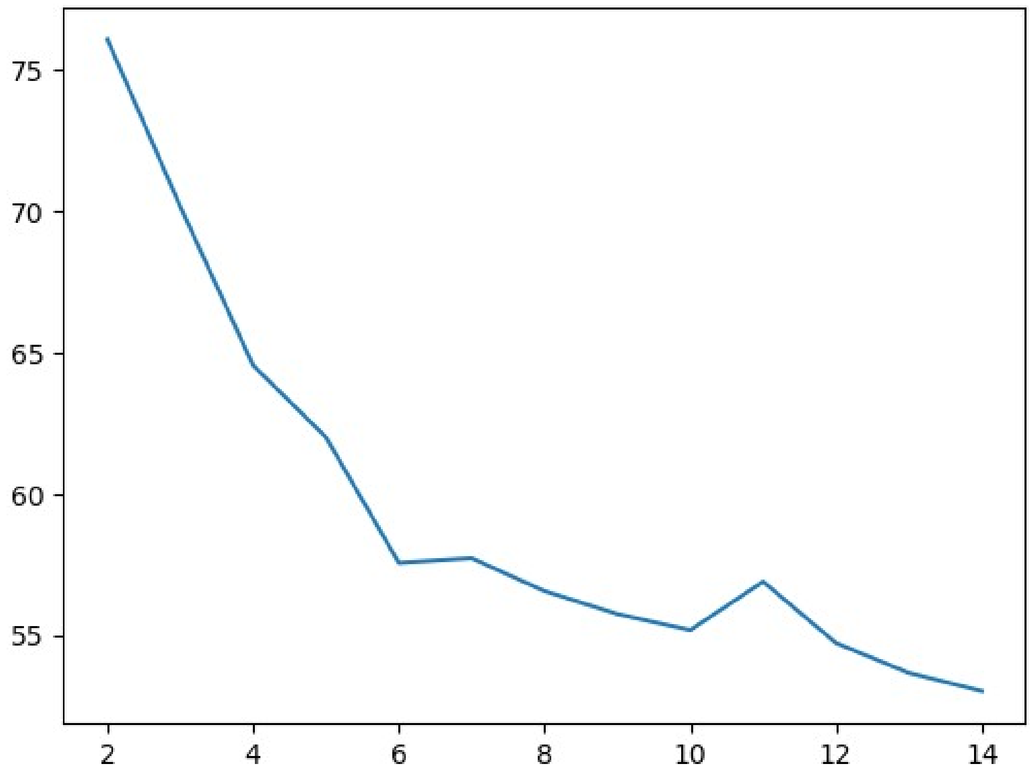
```
CHs = []
for i in range(2,15) :
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(dataStd)
    CH = calinski_harabasz_score(dataStd, km.labels_)
    CHs.append(CH)
CHs
```

```
[76.04453991021924,
 70.14064104652849,
 64.52826972525425,
 61.98964671049615,
 57.55212589590608,
 57.71703427286113,
 56.55454325960451,
```

```
 55.743629307077256,
 55.17899091427457,
 56.888580669937355,
 54.718730525831546,
 53.663041671143745,
 53.02632044508594]
```

```python
plt.plot(range(2, 15), CHs)
```

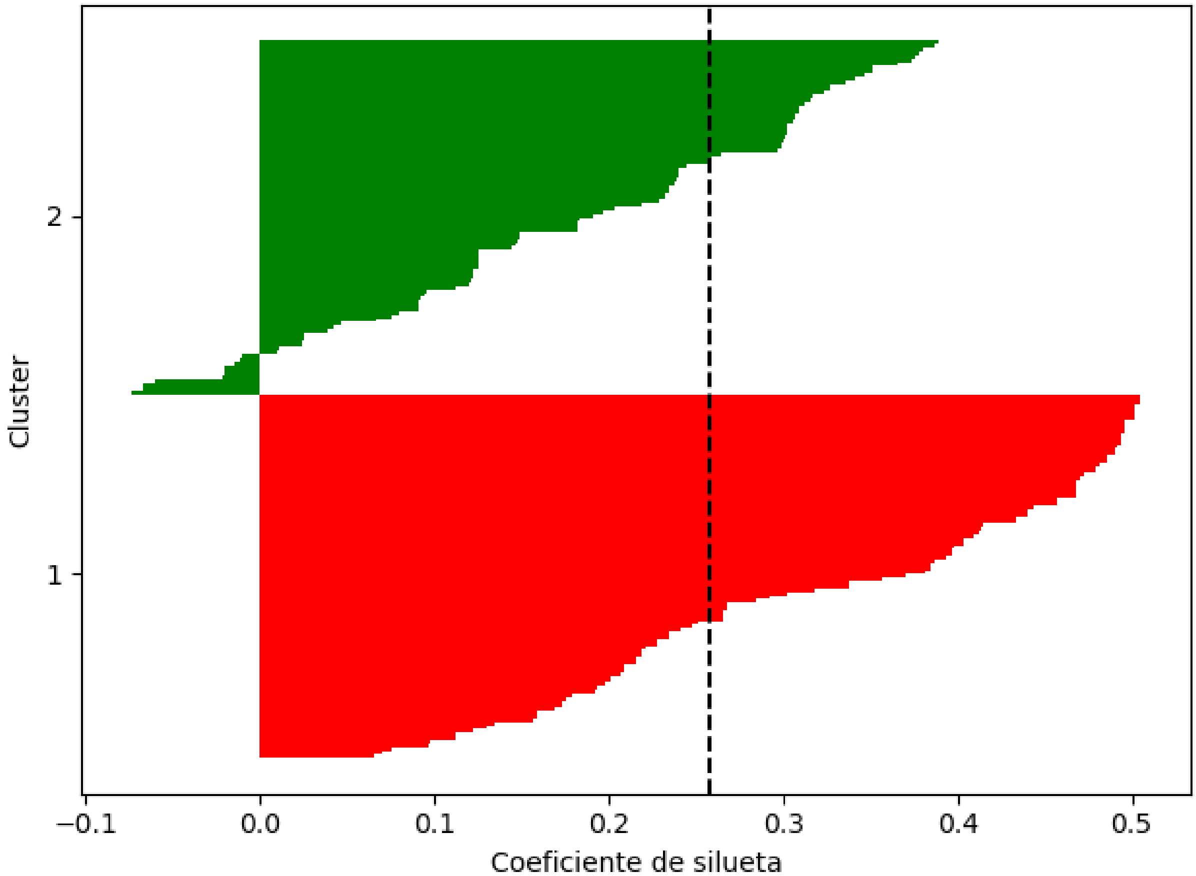
```
[<matplotlib.lines.Line2D at 0x1e276b39490>]
```



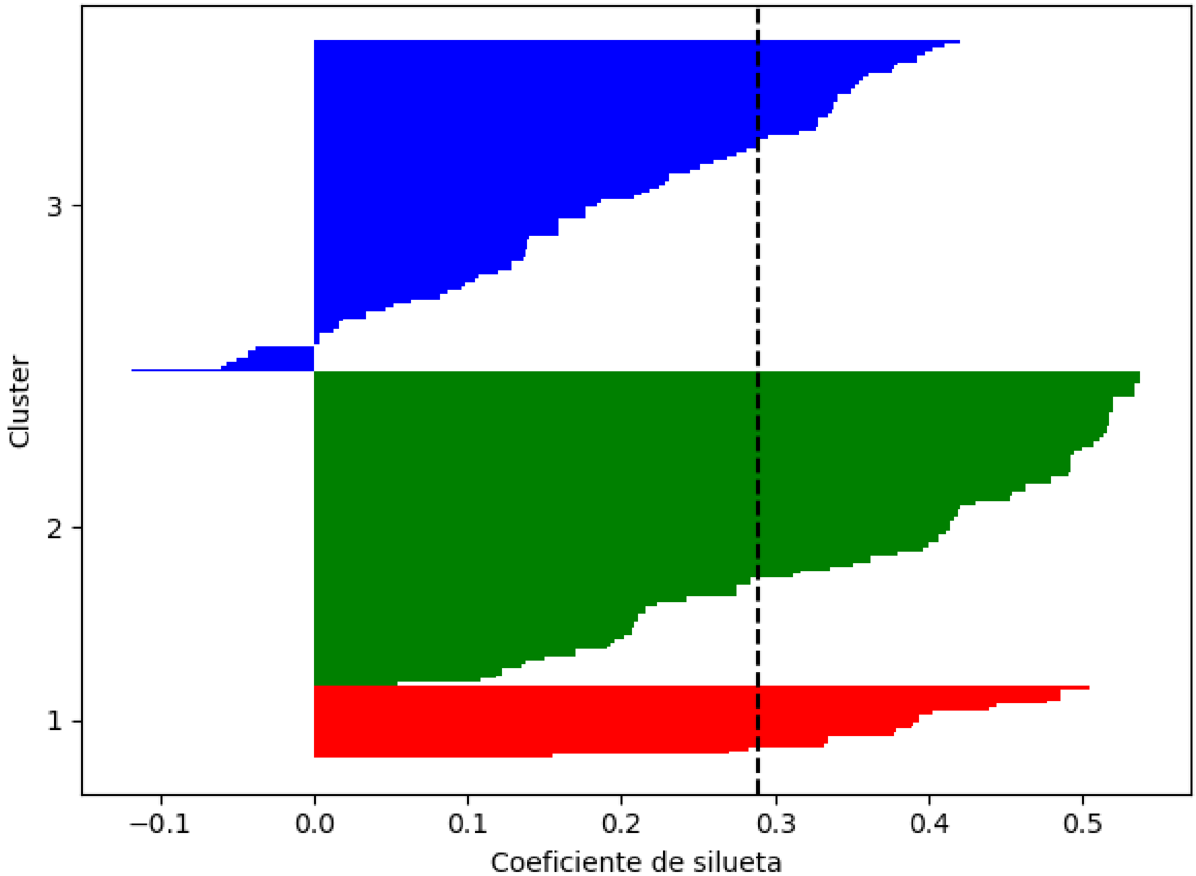
Segun Calinski el K optiomo es aproximadamente 3

```python
def grafico(dataStd, k):
    kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
    kmeans.fit(dataStd)
    y_clusters = kmeans.labels_
    cluster_labels = np.unique(y_clusters)

    silueta_puntos = silhouette_samples(dataStd, y_clusters,
metric='euclidean')

    fig, ax = plt.subplots()
    y_ax_lower, y_ax_upper = 0, 0
    yticks = []
```

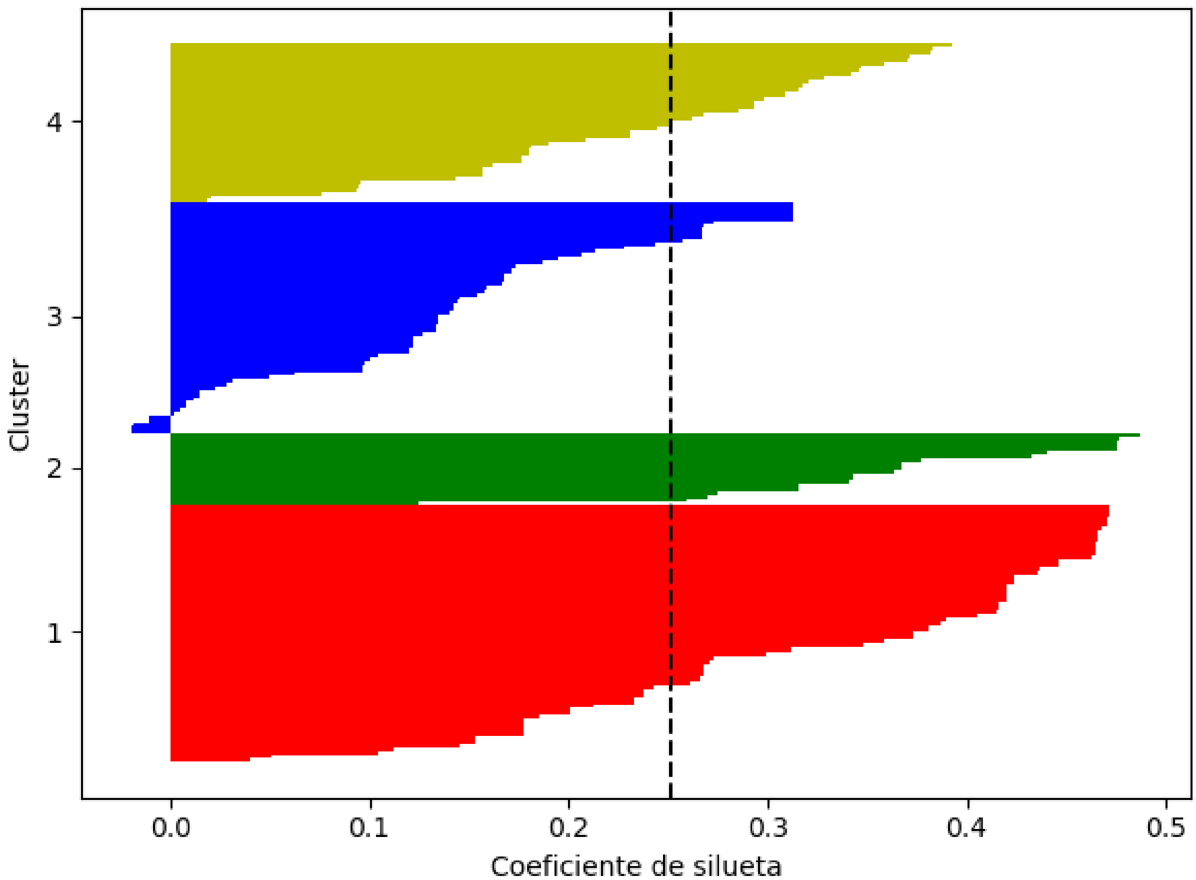
```
    colores = ['r', 'g', 'b', 'y', 'o',"w"]
    for i, c in enumerate(cluster_labels):
        silueta_puntos_c = silueta_puntos[y_clusters == c]
        silueta_puntos_c.sort()
        y_ax_upper += len(silueta_puntos_c)
        color = colores[i]
        ax.barh(range(y_ax_lower, y_ax_upper), silueta_puntos_c,
height=1.0,
                edgecolor='none', color=color)

        yticks.append((y_ax_lower + y_ax_upper) / 2.)
        y_ax_lower += len(silueta_puntos_c)

    silueta_promedio = np.mean(silueta_puntos)
    ax.axvline(silueta_promedio, color="black", linestyle="--")

    ax.set_yticks(yticks)
    ax.set_yticklabels(cluster_labels + 1)
    ax.set_ylabel('Cluster')
    ax.set_xlabel('Coeficiente de silueta')

    plt.tight_layout()
    plt.show()

grafico(dataStd, 2)
```




```
grafico(dataStd, 3)
```




```
grafico(dataStd, 4)
```



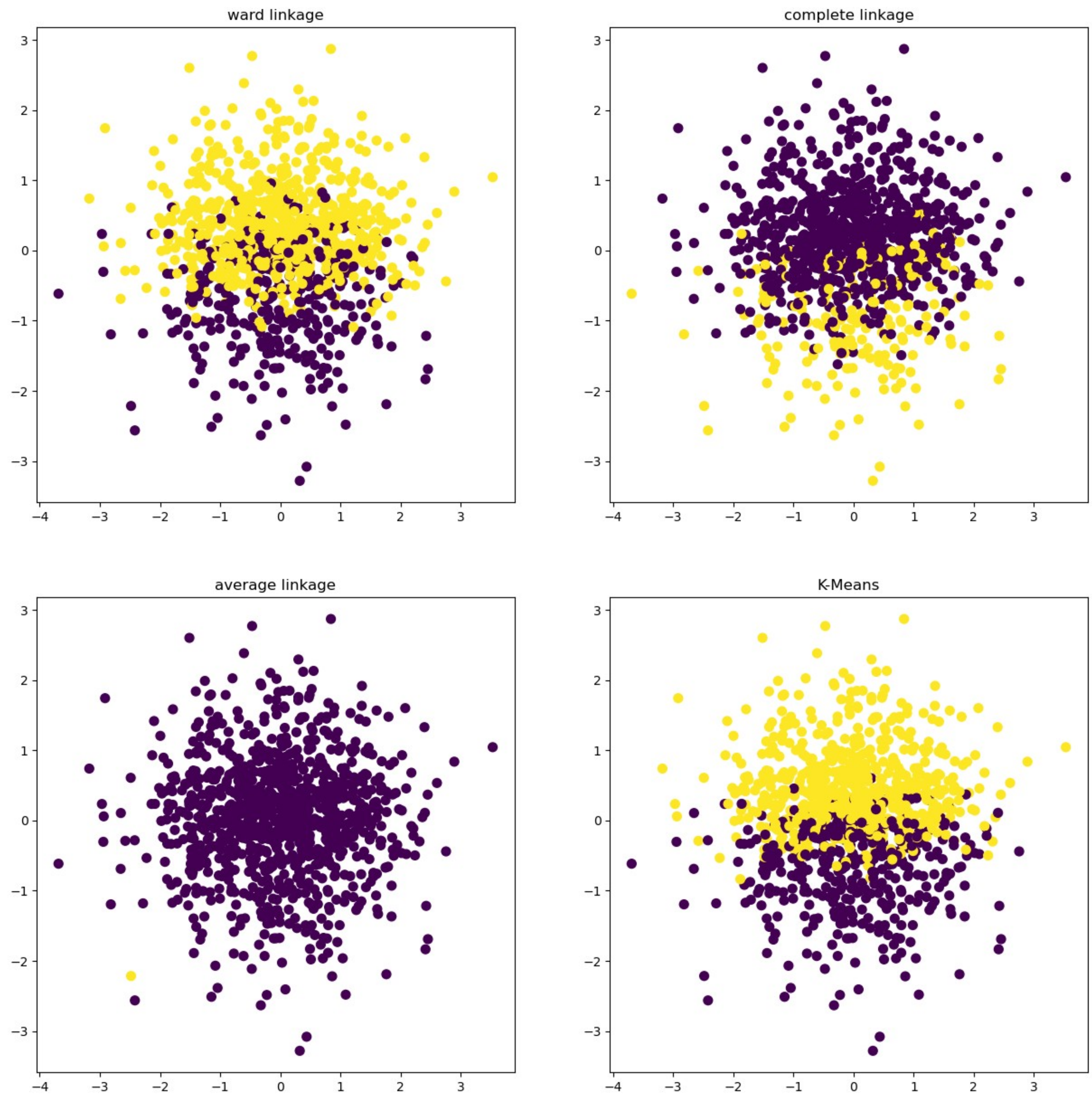
Segun la silueta podemos concluir que el K optiomo es 2

```
var_num = data[['wheel-base', 'length', 'height', 'bore', 'stroke', 
'compression-ratio',
       'horsepower', 'peak-rpm', 'city-mpg']]

dataStd.head()

   wheel-base  length  height  bore  stroke  compression-ratio  
horsepower  \
0        -1.69   -0.43   -2.02  0.51   -1.82              -0.29  
0.17   
1        -1.69   -0.43   -2.02  0.51   -1.82              -0.29  
0.17   
2        -0.71   -0.23   -0.54 -2.38    0.68              -0.29  
1.26   
3         0.17    0.21    0.24 -0.51    0.46              -0.04       -
0.06   
4         0.11    0.21    0.24 -0.51    0.46              -0.54  
0.27   

   peak-rpm  city-mpg  
0     -0.26     -0.65  
```

```
1     -0.26     -0.65
2     -0.26     -0.95
3      0.78     -0.19
4      0.78     -1.11
```

```
#Utilizamos Cluster jerarquico con fusiones y KMeans con K = 2

fig, axes = plt.subplots(2,2,figsize=(15,15))

link = 'ward'
clustering = AgglomerativeClustering(linkage=link, n_clusters=2)
clustering.fit(X)
axes[0][0].scatter(X[:, 0], X[:, 1], c=clustering.labels_, s=50,
cmap='viridis')
axes[0][0].set_title("%s linkage" % link)

link = 'complete'
clustering = AgglomerativeClustering(linkage=link, n_clusters=2)
clustering.fit(X)
axes[0][1].scatter(X[:, 0], X[:, 1], c=clustering.labels_, s=50,
cmap='viridis')
axes[0][1].set_title("%s linkage" % link)

link = 'average'
clustering = AgglomerativeClustering(linkage=link, n_clusters=2)
clustering.fit(X)
axes[1][0].scatter(X[:, 0], X[:, 1], c=clustering.labels_, s=50,
cmap='viridis')
axes[1][0].set_title("%s linkage" % link)

clustering = KMeans(n_clusters=2)
clustering.fit(X)
axes[1][1].scatter(X[:, 0], X[:, 1], c=clustering.labels_, s=50,
cmap='viridis')
axes[1][1].set_title("K-Means")
```

```
Text(0.5, 1.0, 'K-Means')
```

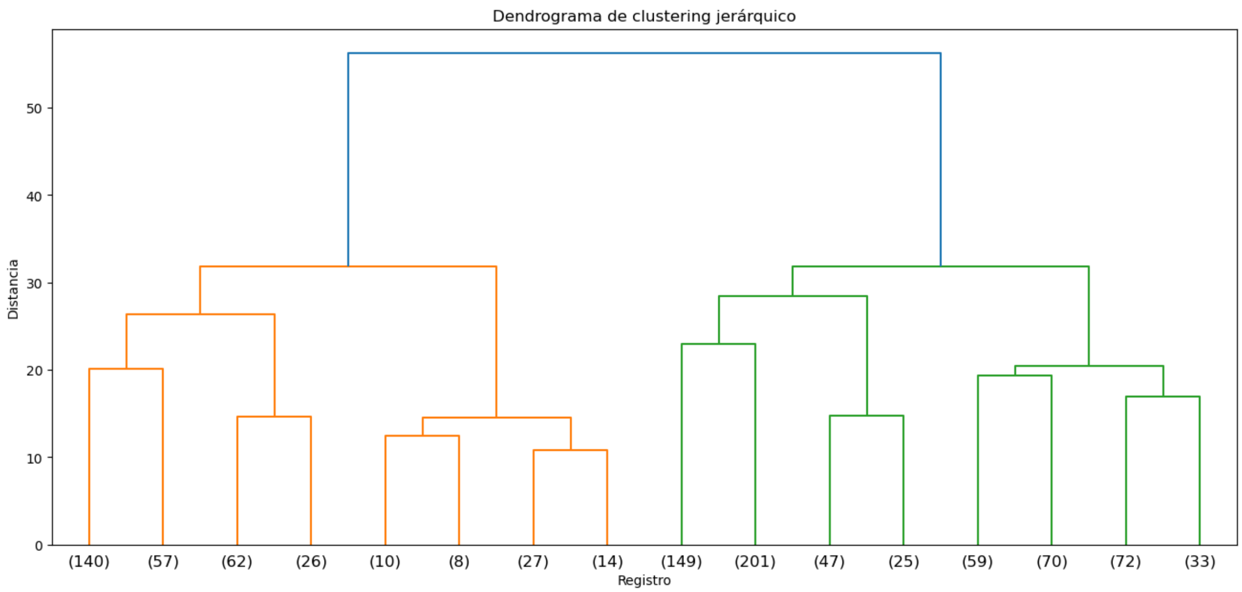


Anlizando las graficas podemos concluir que con K=2, Ward y Kmeans obtuvieron los mejores resultados identificando los clusters

Dendograma

```
#Dendograma
plt.figure(figsize=(16, 7))
plt.title('Dendrograma de clustering jerárquico')
plt.xlabel('Registro')
plt.ylabel('Distancia')
dendrogram(fusiones,
           orientation='top',
```

```
            distance_sort='descending',
            truncate_mode='level',
            p=3,
            show_leaf_counts=True)
plt.show()
```



Observando el Dendograma se puede evidenciar que con K=2 los clusters quedan bien balanceados

```
#Obtenemos el cluster de cada punto con Clustering jerarquico
k = 2
clusters = fcluster(fusiones, k, criterion='maxclust')
clusters

array([2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1,
1,
       1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2,
1,
       1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1,
2,
       1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 2,
1,
       1, 1, 2, 2, 1, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1, 1,
1,
       1, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 2, 1, 2, 1, 1,
2,
       1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1,
1,
       2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1,
2,
       1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1,
```

```
2,
        1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 2,
2,
        2, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2,
1,
        1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1,
2,
        1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 1, 2,
1,
        1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 1, 1,
1,
        1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1,
2,
        2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1,
1,
        1, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1,
1,
        1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2,
2,
        1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2, 1, 2, 2,
1,
        1, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2,
1,
        1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
2,
        2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1,
1,
        1, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2,
1,
        1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 2, 1, 2, 2,
1,
        1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1,
2,
        1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2,
1,
        1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1,
1,
        2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2,
1,
        2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1,
1,
        1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2,
2,
        2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 1,
1,
```

```
        2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2,
1,
        2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1,
1,
        1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2,
2,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1,
2,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
2,
        1, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2,
2,
        1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2,
1,
        2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2,
1,
        1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1,
2,
        2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1,
2,
        1, 2, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2,
2,
        1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1,
1,
        1, 2, 2, 1, 1, 1, 1, 2, 1, 1], dtype=int32)
```

```python
#Clustering usando KMEANS con K=2
kmeans = KMeans(n_clusters=2, random_state=0, n_init=10)
kmeans.fit(dataStd)
```

```
KMeans(n_clusters=2, random_state=0)
```

```python
kmeans.labels_
```

```
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0,
        1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0,
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0])
```

```
clusters = kmeans.predict(dataStd)
clusters
```

```
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0])
```

```
counter=Counter(clusters)
print(counter)
```

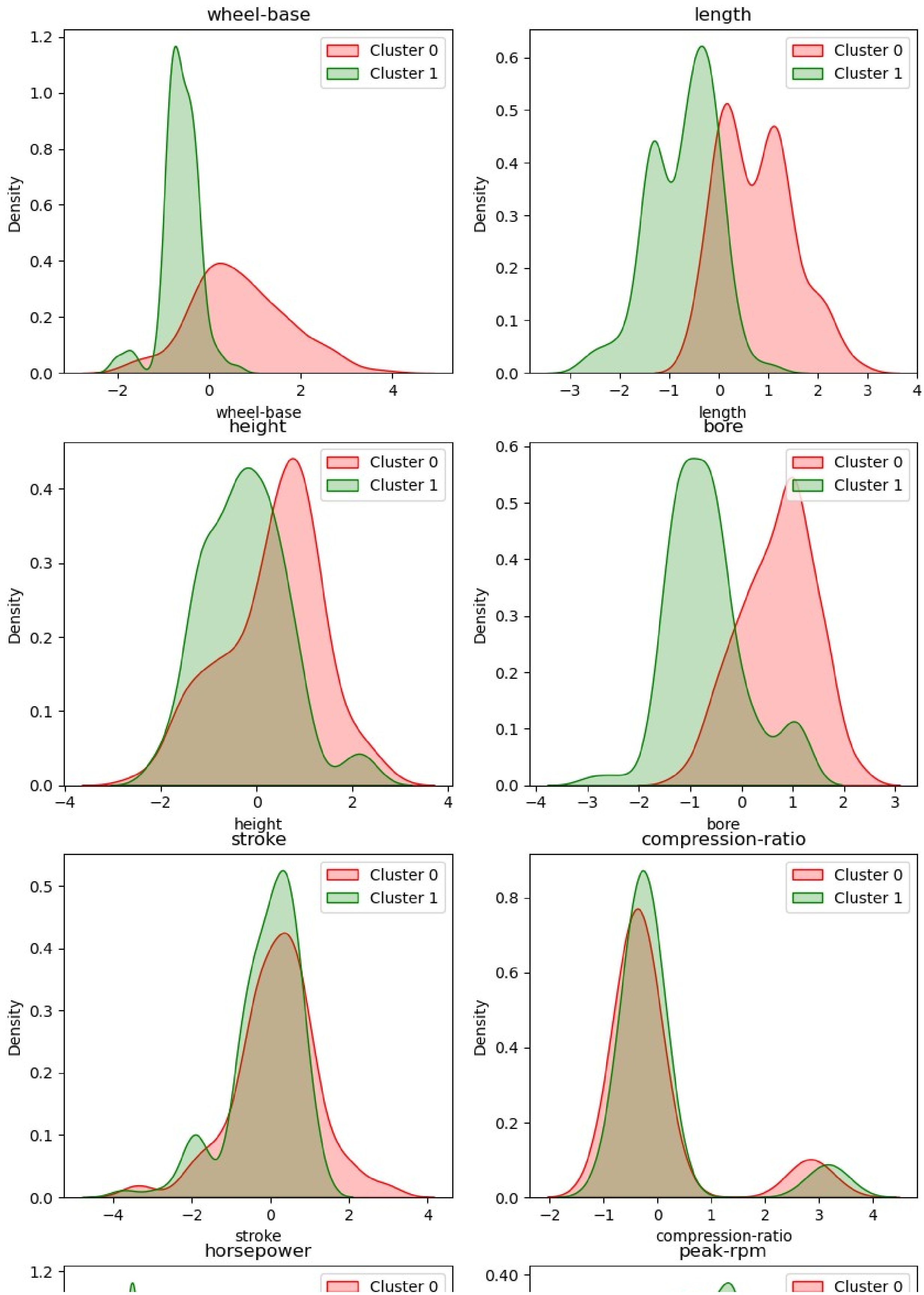
```
Counter({1: 101, 0: 98})
```

Tenemos 2 Clusters, El primero con 101 registros y el segundo con 98 registros

```
dataStd.loc[:,'Cluster'] = clusters

#Valores promedio de cada cluster
df_agrupado = Nuevo.groupby('label').median()
df_agrupado
```

```
       wheel-base  length  height  bore  stroke  compression-ratio  \
label
0           94.49  166.28   53.00  3.15    3.23               9.10
1          102.41  183.52   55.00  3.56    3.35               8.90

       horsepower  peak-rpm  city-mpg
label
0           69.92   5200.18     30.01
1          116.03   4999.69     18.98
```

```
fig = plt.figure(figsize=(10,500))
i=1
for var in var_num:
    ax = fig.add_subplot(math.ceil(len(var_num)/2), 2, i)
    sns.kdeplot(dataStd.loc[dataStd.Cluster==0][var], shade=True, color='r', ax=ax);
    sns.kdeplot(dataStd.loc[dataStd.Cluster==1][var], shade=True, color='g', ax=ax);
    sns.kdeplot(dataStd.loc[dataStd.Cluster==2][var], shade=True, color='b', ax=ax);
    plt.title(var)
    plt.legend(['Cluster 0', 'Cluster 1', 'Cluster 2'])
    i+=1
```



wheel-base
Cluster 0
Cluster 1
Density
wheel-base
length
Cluster 0
Cluster 1
Density
length
height
Cluster 0
Cluster 1
Density
height
bore
Cluster 0
Cluster 1
Density
bore
stroke
Cluster 0
Cluster 1
Density
stroke
compression-ratio
Cluster 0
Cluster 1
Density
compression-ratio
horsepower
Cluster 0
peak-rpm
Cluster 0

```
fig = plt.figure(figsize=(15,15))
colorPalette = ["r", "g"]
ax = fig.add_subplot(2, 2, 1)
sns.scatterplot(x="compression-ratio", y="length", hue="Cluster",
data=dataStd, ax=ax, palette=colorPalette, s=100, alpha=0.5)
plt.title("compression-ratio vs. length")
ax = fig.add_subplot(2, 2, 2)
sns.scatterplot(x="compression-ratio", y="horsepower", hue="Cluster",
data=dataStd, ax=ax, palette=colorPalette, s=100, alpha=0.5)
plt.title("compression-ratio vs. horsepower")
```

```
Text(0.5, 1.0, 'compression-ratio vs. horsepower')
```