Problem Statement

A Chinese automobile company Geely Auto aspires to enter the US market by setting up their manufacturing unit there and producing cars locally to give competition to their US and European counterparts.

They have contracted an automobile consulting company to understand the factors on which the pricing of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the American market, since those may be very different from the Chinese market. The company wants to know:

Which variables are significant in predicting the price of a car How well those variables describe the price of a car Based on various market surveys, the consulting firm has gathered a large data set of different types of cars across the America market.

Business Goal

We are required to model the price of cars with the available independent variables. It will be used by the management to understand how exactly the prices vary with the independent variables. They can accordingly manipulate the design of the cars, the business strategy etc. to meet certain price levels. Further, the model will be a good way for management to understand the pricing dynamics of a new market.

Please Note : The dataset provided is for learning purpose. Please don't draw any inference with real world scenario.

How to tackle this project:

- Explore data:
    - Definie target data (price)
    - Analyse Numerical Featues
    - Treat Numerical Features
    - Analyse Categorical Features
    - Treat Categorical Features
- Model Testing
    - Definie Models to be tested
    - Define Baseline Model (Linear Regression)
    - Run different models and compare to Baseline
    - Define best model
    - Run best model with Hyperparameter Tuning
    - Feature Importance
- Conclusion
- References

CAR PRICE PREDICTION DATA

DATA DICTONARY

---

1 Car_ID: Unique id of each observation

2 Symboling: Its assigned insurance risk rating, A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

3 carCompany: Name of car company

4 fueltype: Car fuel type i.e gas or diesel

5 aspiration: Aspiration used in a car

6 doornumber: Number of doors in a car

7 carbody: body of car

8 drivewheel: type of drive wheel

9 enginelocation: Location of car engine

10 wheelbase: Weelbase of car

11 carlength: Length of car

12 carwidth: Width of car

13 carheight: height of car

14 curbweight: The weight of a car without occupants or baggage.

15 enginetype: Type of engine.

16 cylindernumber: cylinder placed in the car

17 enginesize: Size of car

18 fuelsystem: Fuel system of car

19 boreratio: Boreratio of car

20 stroke: Stroke or volume inside the engine

21 compressionratio: compression ratio of car

22 horsepower: Horsepower

23 peakrpm: car peak rpm

24 citympg: Mileage in city

25 highwaympg: Mileage on highway

26 price: Price of car **--> TARGET DATA**

Reference: https://www.kaggle.com/hellbuoy/car-price-prediction

Import Relevant Libraries

In [ ]:

```
import numpy as np
import pandas as pd
import pathlib
import datetime
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import optuna
from sklearn import preprocessing

#importing plotly and cufflinks in offline mode
```

```python
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

from sklearn.metrics import r2_score, mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler, PolynomialFeatures, OneHotEncoder,
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, GradientBoo
from sklearn.svm import SVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor

# Classification Regressors
#from sklearn.neighbors import KNeighborsClassifier
#from sklearn.linear_model import RidgeClassifier
#from sklearn.linear_model import LogisticRegression
#from sklearn.linear_model import SGDClassifier

from sklearn.feature_selection import SelectKBest,SelectPercentile, f_classif, f_reg
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

# Importar Tensorflow
import tensorflow as tf

# Importar XGBoost
from xgboost import XGBRegressor
#from xgboost import XGBClassifier
from xgboost import plot_importance

# Importar bibliotecas para testar PyTorch
import copy
import torch
import torch.nn as nn
import torch.optim as optim
import tqdm
```

Import and Explore dataset

In [ ]:

```python
# Load the data
raw_data = pd.DataFrame()

raw_data = pd.read_csv('dataset/CarPrice_Assignment.csv', low_memory=False)
```

In [ ]:

```python
# How dataset looks like
raw_data.head()
```

Out[ ]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	engine
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	engine
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	

5 rows × 26 columns

In [ ]:

```
# Type of data, size and null values
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [ ]:

```
# Check for duplicate data

raw_data.duplicated().sum()
```

Out[ ]:

```
0
```

In [ ]:

```
# Check for missing values

raw_data.isnull().sum()
```

Out[ ]:

```
car_ID              0
symboling           0
CarName             0
fueltype            0
```

```
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

In [ ]:

```
# Check unique data in each column to have a better understanding of the data
raw_data.nunique()
```

Out[ ]:

```
car_ID              205
symboling             6
CarName             147
fueltype              2
aspiration            2
doornumber            2
carbody               5
drivewheel            3
enginelocation        2
wheelbase            53
carlength            75
carwidth             44
carheight            49
curbweight          171
enginetype            7
cylindernumber        7
enginesize           44
fuelsystem            8
boreratio            38
stroke               37
compressionratio     32
horsepower           59
peakrpm              23
citympg              29
highwaympg           30
price               189
dtype: int64
```

- As expected car_ID works like and index, so we have to drop it from the analysis as it won't have effect on the regression
- The variance of the features is at least 2 unique data
- Car name is a object type and has high variance, so it should be treat

In [ ]:

```
# Make a copy of the dataset
df = raw_data.copy()
```

In [ ]:

```python
# Drop car_ID column
df.drop(['car_ID'], axis=1, inplace=True)
```

In [ ]:

```python
# Let's treat Car Name feature
df["CarName"].unique()
```

Out[ ]:

```
array(['alfa-romero giulia', 'alfa-romero stelvio',
       'alfa-romero Quadrifoglio', 'audi 100 ls', 'audi 100ls',
       'audi fox', 'audi 5000', 'audi 4000', 'audi 5000s (diesel)',
       'bmw 320i', 'bmw x1', 'bmw x3', 'bmw z4', 'bmw x4', 'bmw x5',
       'chevrolet impala', 'chevrolet monte carlo', 'chevrolet vega 2300',
       'dodge rampage', 'dodge challenger se', 'dodge d200',
       'dodge monaco (sw)', 'dodge colt hardtop', 'dodge colt (sw)',
       'dodge coronet custom', 'dodge dart custom',
       'dodge coronet custom (sw)', 'honda civic', 'honda civic cvcc',
       'honda accord cvcc', 'honda accord lx', 'honda civic 1500 gl',
       'honda accord', 'honda civic 1300', 'honda prelude',
       'honda civic (auto)', 'isuzu MU-X', 'isuzu D-Max ',
       'isuzu D-Max V-Cross', 'jaguar xj', 'jaguar xf', 'jaguar xk',
       'maxda rx3', 'maxda glc deluxe', 'mazda rx2 coupe', 'mazda rx-4',
       'mazda glc deluxe', 'mazda 626', 'mazda glc', 'mazda rx-7 gs',
       'mazda glc 4', 'mazda glc custom l', 'mazda glc custom',
       'buick electra 225 custom', 'buick century luxus (sw)',
       'buick century', 'buick skyhawk', 'buick opel isuzu deluxe',
       'buick skylark', 'buick century special',
       'buick regal sport coupe (turbo)', 'mercury cougar',
       'mitsubishi mirage', 'mitsubishi lancer', 'mitsubishi outlander',
       'mitsubishi g4', 'mitsubishi mirage g4', 'mitsubishi montero',
       'mitsubishi pajero', 'Nissan versa', 'nissan gt-r', 'nissan rogue',
       'nissan latio', 'nissan titan', 'nissan leaf', 'nissan juke',
       'nissan note', 'nissan clipper', 'nissan nv200', 'nissan dayz',
       'nissan fuga', 'nissan otti', 'nissan teana', 'nissan kicks',
       'peugeot 504', 'peugeot 304', 'peugeot 504 (sw)', 'peugeot 604sl',
       'peugeot 505s turbo diesel', 'plymouth fury iii',
       'plymouth cricket', 'plymouth satellite custom (sw)',
       'plymouth fury gran sedan', 'plymouth valiant', 'plymouth duster',
       'porsche macan', 'porcshce panamera', 'porsche cayenne',
       'porsche boxter', 'renault 12tl', 'renault 5 gtl', 'saab 99e',
       'saab 99le', 'saab 99gle', 'subaru', 'subaru dl', 'subaru brz',
       'subaru baja', 'subaru r1', 'subaru r2', 'subaru trezia',
       'subaru tribeca', 'toyota corona mark ii', 'toyota corona',
       'toyota corolla 1200', 'toyota corona hardtop',
       'toyota corolla 1600 (sw)', 'toyota carina', 'toyota mark ii',
       'toyota corolla', 'toyota corolla liftback',
       'toyota celica gt liftback', 'toyota corolla tercel',
       'toyota corona liftback', 'toyota starlet', 'toyota tercel',
       'toyota cressida', 'toyota celica gt', 'toyouta tercel',
       'vokswagen rabbit', 'volkswagen 1131 deluxe sedan',
       'volkswagen model 111', 'volkswagen type 3', 'volkswagen 411 (sw)',
       'volkswagen super beetle', 'volkswagen dasher', 'vw dasher',
       'vw rabbit', 'volkswagen rabbit', 'volkswagen rabbit custom',
       'volvo 145e (sw)', 'volvo 144ea', 'volvo 244dl', 'volvo 245',
       'volvo 264gl', 'volvo diesel', 'volvo 246'], dtype=object)
```

In [ ]:

```python
for name in df['CarName']:
    brand = name.split()
    print(brand)
# As shown, the first element is the brand name
```

```
['alfa-romero', 'giulia']
['alfa-romero', 'stelvio']
['alfa-romero', 'Quadrifoglio']
['audi', '100', 'ls']
['audi', '100ls']
```

```
['audi', 'fox']
['audi', '100ls']
['audi', '5000']
['audi', '4000']
['audi', '5000s', '(diesel)']
['bmw', '320i']
['bmw', '320i']
['bmw', 'x1']
['bmw', 'x3']
['bmw', 'z4']
['bmw', 'x4']
['bmw', 'x5']
['bmw', 'x3']
['chevrolet', 'impala']
['chevrolet', 'monte', 'carlo']
['chevrolet', 'vega', '2300']
['dodge', 'rampage']
['dodge', 'challenger', 'se']
['dodge', 'd200']
['dodge', 'monaco', '(sw)']
['dodge', 'colt', 'hardtop']
['dodge', 'colt', '(sw)']
['dodge', 'coronet', 'custom']
['dodge', 'dart', 'custom']
['dodge', 'coronet', 'custom', '(sw)']
['honda', 'civic']
['honda', 'civic', 'cvcc']
['honda', 'civic']
['honda', 'accord', 'cvcc']
['honda', 'civic', 'cvcc']
['honda', 'accord', 'lx']
['honda', 'civic', '1500', 'gl']
['honda', 'accord']
['honda', 'civic', '1300']
['honda', 'prelude']
['honda', 'accord']
['honda', 'civic']
['honda', 'civic', '(auto)']
['isuzu', 'MU-X']
['isuzu', 'D-Max']
['isuzu', 'D-Max', 'V-Cross']
['isuzu', 'D-Max']
['jaguar', 'xj']
['jaguar', 'xf']
['jaguar', 'xk']
['maxda', 'rx3']
['maxda', 'glc', 'deluxe']
['mazda', 'rx2', 'coupe']
['mazda', 'rx-4']
['mazda', 'glc', 'deluxe']
['mazda', '626']
['mazda', 'glc']
['mazda', 'rx-7', 'gs']
['mazda', 'glc', '4']
['mazda', '626']
['mazda', 'glc', 'custom', 'l']
['mazda', 'glc', 'custom']
['mazda', 'rx-4']
['mazda', 'glc', 'deluxe']
['mazda', '626']
['mazda', 'glc']
['mazda', 'rx-7', 'gs']
['buick', 'electra', '225', 'custom']
['buick', 'century', 'luxus', '(sw)']
['buick', 'century']
['buick', 'skyhawk']
['buick', 'opel', 'isuzu', 'deluxe']
['buick', 'skylark']
['buick', 'century', 'special']
```

```
['buick', 'regal', 'sport', 'coupe', '(turbo)']
['mercury', 'cougar']
['mitsubishi', 'mirage']
['mitsubishi', 'lancer']
['mitsubishi', 'outlander']
['mitsubishi', 'g4']
['mitsubishi', 'mirage', 'g4']
['mitsubishi', 'g4']
['mitsubishi', 'outlander']
['mitsubishi', 'g4']
['mitsubishi', 'mirage', 'g4']
['mitsubishi', 'montero']
['mitsubishi', 'pajero']
['mitsubishi', 'outlander']
['mitsubishi', 'mirage', 'g4']
['Nissan', 'versa']
['nissan', 'gt-r']
['nissan', 'rogue']
['nissan', 'latio']
['nissan', 'titan']
['nissan', 'leaf']
['nissan', 'juke']
['nissan', 'latio']
['nissan', 'note']
['nissan', 'clipper']
['nissan', 'rogue']
['nissan', 'nv200']
['nissan', 'dayz']
['nissan', 'fuga']
['nissan', 'otti']
['nissan', 'teana']
['nissan', 'kicks']
['nissan', 'clipper']
['peugeot', '504']
['peugeot', '304']
['peugeot', '504', '(sw)']
['peugeot', '504']
['peugeot', '504']
['peugeot', '604sl']
['peugeot', '504']
['peugeot', '505s', 'turbo', 'diesel']
['peugeot', '504']
['peugeot', '504']
['peugeot', '604sl']
['plymouth', 'fury', 'iii']
['plymouth', 'cricket']
['plymouth', 'fury', 'iii']
['plymouth', 'satellite', 'custom', '(sw)']
['plymouth', 'fury', 'gran', 'sedan']
['plymouth', 'valiant']
['plymouth', 'duster']
['porsche', 'macan']
['porcshce', 'panamera']
['porsche', 'cayenne']
['porsche', 'boxter']
['porsche', 'cayenne']
['renault', '12tl']
['renault', '5', 'gtl']
['saab', '99e']
['saab', '99le']
['saab', '99le']
['saab', '99gle']
['saab', '99gle']
['saab', '99e']
['subaru']
['subaru', 'dl']
['subaru', 'dl']
['subaru']
['subaru', 'brz']
```

```
['subaru', 'baja']
['subaru', 'r1']
['subaru', 'r2']
['subaru', 'trezia']
['subaru', 'tribeca']
['subaru', 'dl']
['subaru', 'dl']
['toyota', 'corona', 'mark', 'ii']
['toyota', 'corona']
['toyota', 'corolla', '1200']
['toyota', 'corona', 'hardtop']
['toyota', 'corolla', '1600', '(sw)']
['toyota', 'carina']
['toyota', 'mark', 'ii']
['toyota', 'corolla', '1200']
['toyota', 'corona']
['toyota', 'corolla']
['toyota', 'corona']
['toyota', 'corolla']
['toyota', 'mark', 'ii']
['toyota', 'corolla', 'liftback']
['toyota', 'corona']
['toyota', 'celica', 'gt', 'liftback']
['toyota', 'corolla', 'tercel']
['toyota', 'corona', 'liftback']
['toyota', 'corolla']
['toyota', 'starlet']
['toyota', 'tercel']
['toyota', 'corolla']
['toyota', 'cressida']
['toyota', 'corolla']
['toyota', 'celica', 'gt']
['toyota', 'corona']
['toyota', 'corolla']
['toyota', 'mark', 'ii']
['toyota', 'corolla', 'liftback']
['toyota', 'corona']
['toyota', 'starlet']
['toyouta', 'tercel']
['vokswagen', 'rabbit']
['volkswagen', '1131', 'deluxe', 'sedan']
['volkswagen', 'model', '111']
['volkswagen', 'type', '3']
['volkswagen', '411', '(sw)']
['volkswagen', 'super', 'beetle']
['volkswagen', 'dasher']
['vw', 'dasher']
['vw', 'rabbit']
['volkswagen', 'rabbit']
['volkswagen', 'rabbit', 'custom']
['volkswagen', 'dasher']
['volvo', '145e', '(sw)']
['volvo', '144ea']
['volvo', '244dl']
['volvo', '245']
['volvo', '264gl']
['volvo', 'diesel']
['volvo', '145e', '(sw)']
['volvo', '144ea']
['volvo', '244dl']
['volvo', '246']
['volvo', '264gl']
```

In [ ]:

```
df['brand'] = df['CarName'].apply(lambda x: x.split()[0])
```

In [ ]:

```
df['brand'].unique()
```

Out[ ]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

In [ ]:
```
# Changing incorrect brand names
df['brand'] = df['brand'].replace({'maxda': 'mazda',
                                   'Nissan': 'nissan',
                                   'porcshce': 'porsche',
                                   'toyouta': 'toyota',
                                   'vokswagen': 'volkswagen',
                                   'vw': 'volkswagen'})
```

In [ ]:
```
# Lets drop Car Name
df.drop(['CarName'], axis=1, inplace=True)
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(205, 25)
```

Analyse Numerical Features

In [ ]:
```
numerical= df.drop(['price'], axis=1).select_dtypes('number').columns
```

In [ ]:
```
df[numerical].describe()
```

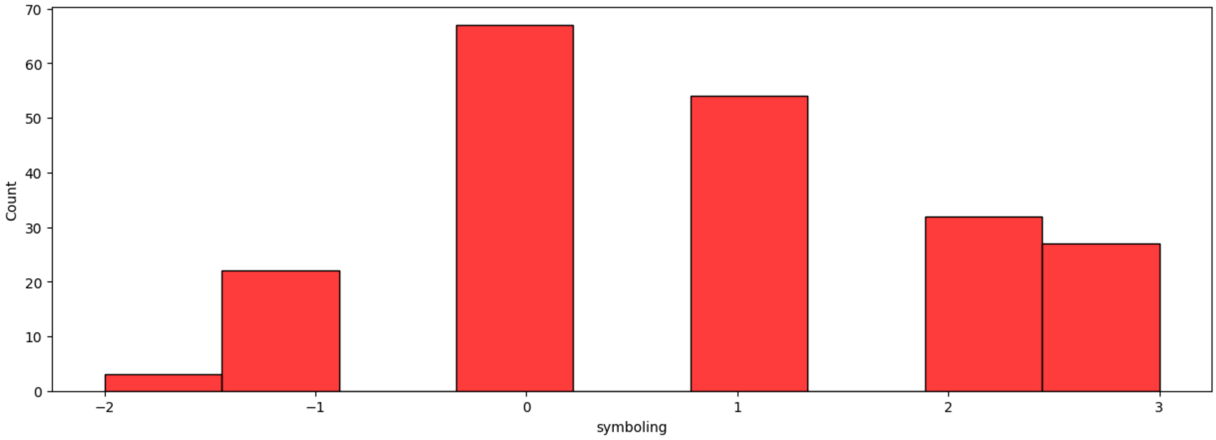
Out[ ]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	borera
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.3297
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.2708
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.5400
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.1500
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.3100
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.5800
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.9400

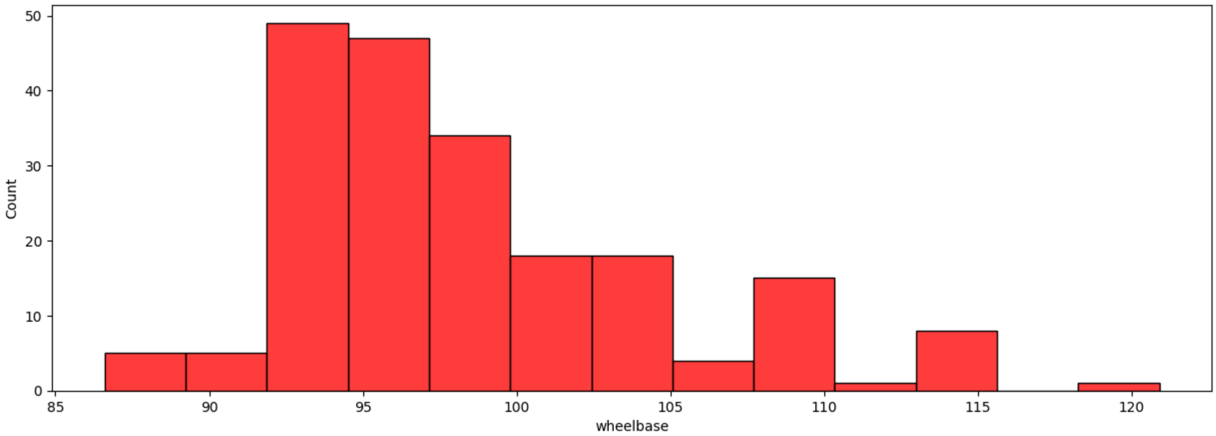
In [ ]:
```
def histograma(coluna):
    plt.figure(figsize=(15, 5)).set_figwidth(15)
    sns.histplot(coluna, color='red')
    #ax2 = plt.twinx()
    #ax2 = sns.histplot(coluna, ax=ax2, color='red', binwidth=5)
    plt.show()
```

In [ ]:
```
for coluna in df[numerical]:
    histograma(df[coluna])
```

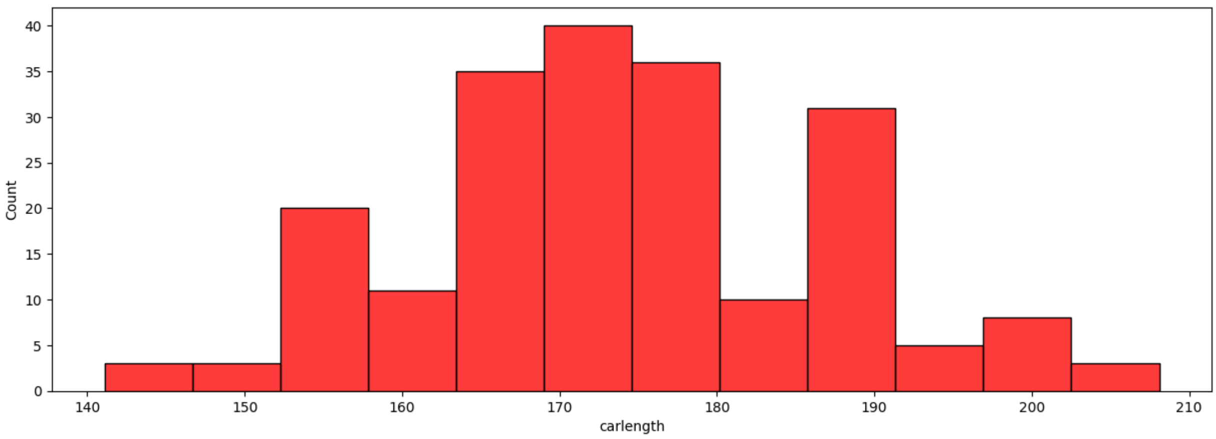
```
print(df[coluna].skew())
```



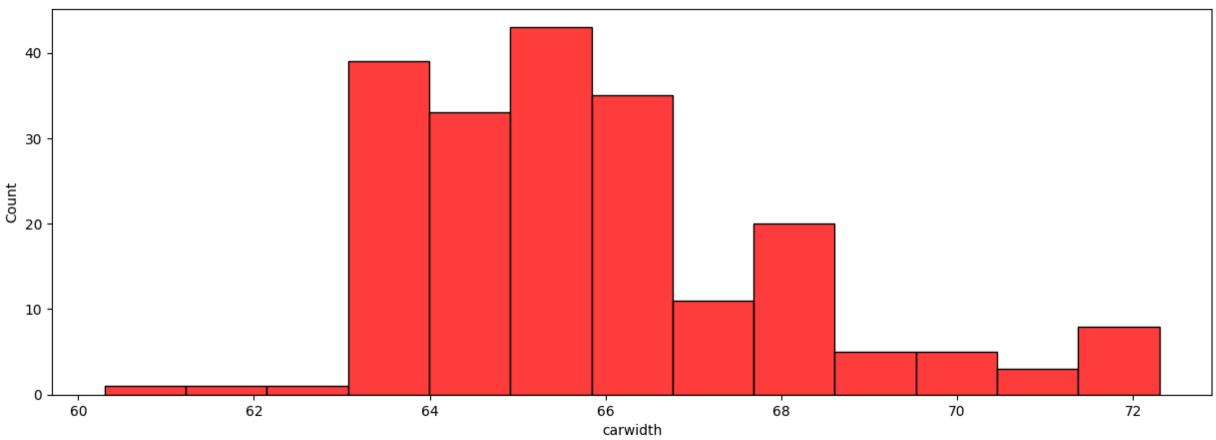
0.21107227205788776



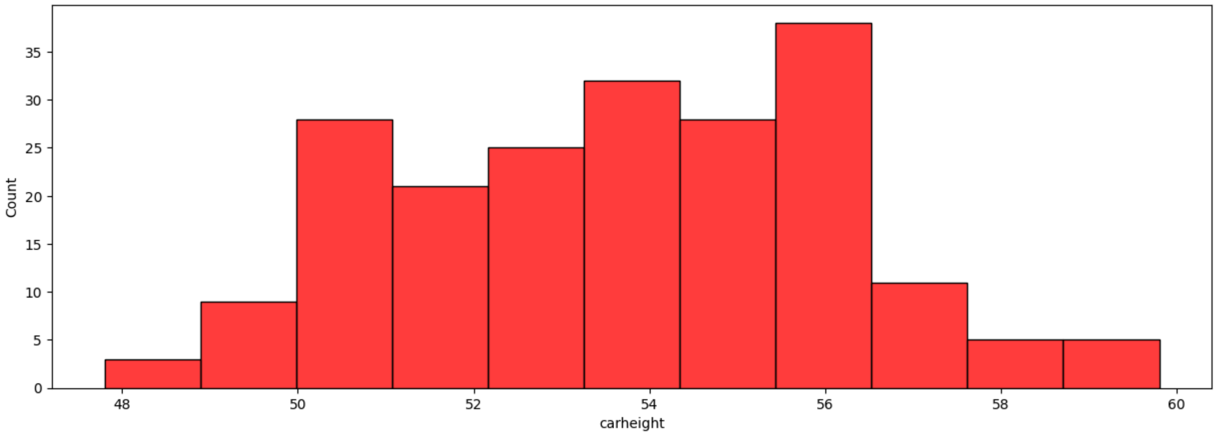
1.0502137758714858



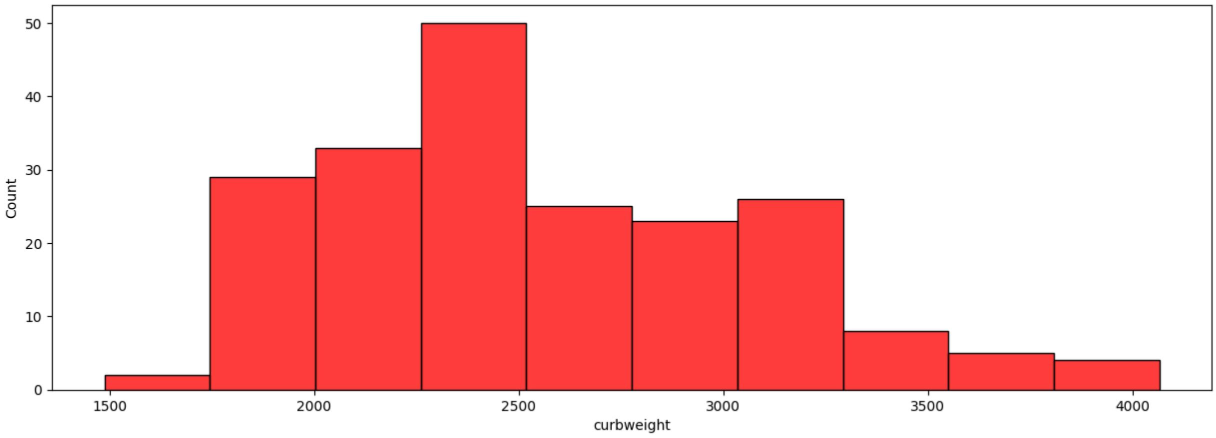
0.1559537713215604



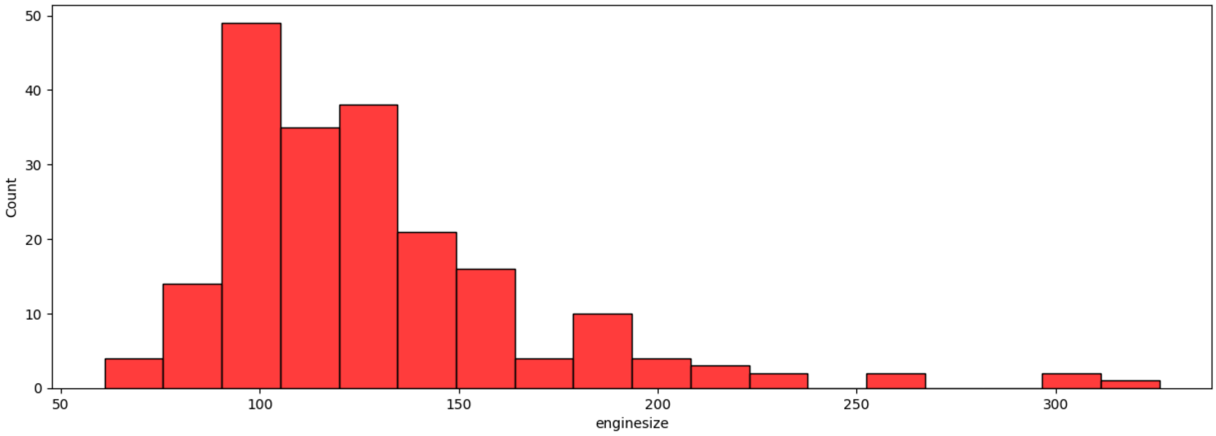
0.904003498786254



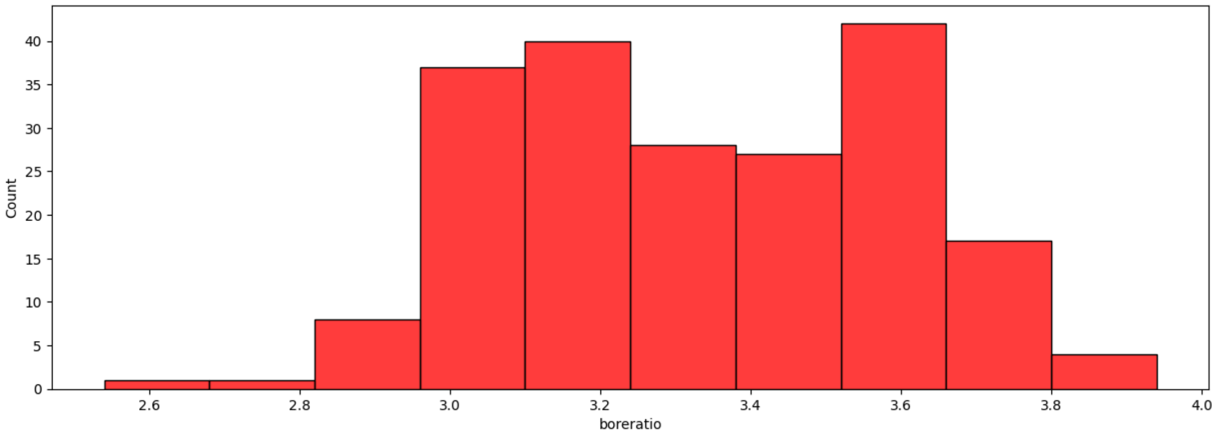
0.06312273247192804



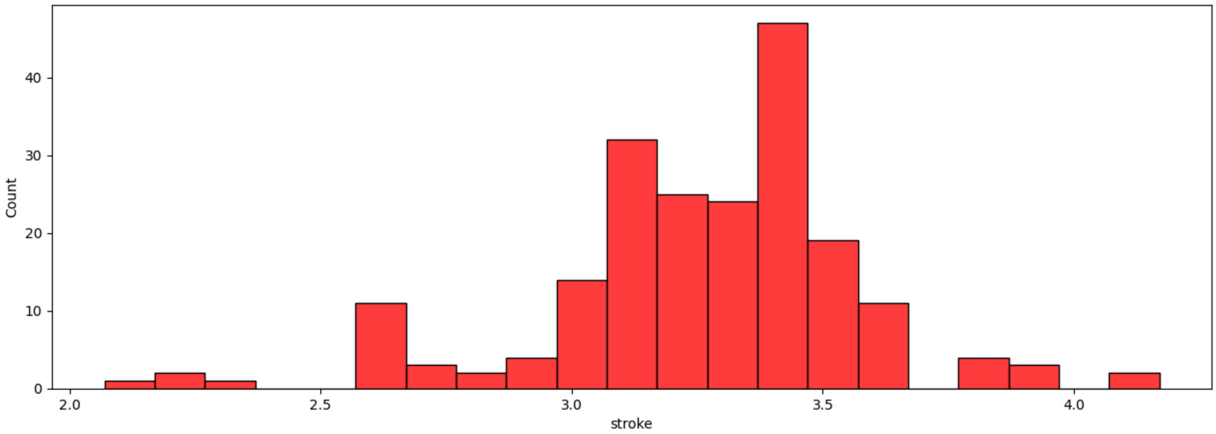
0.681398189052588



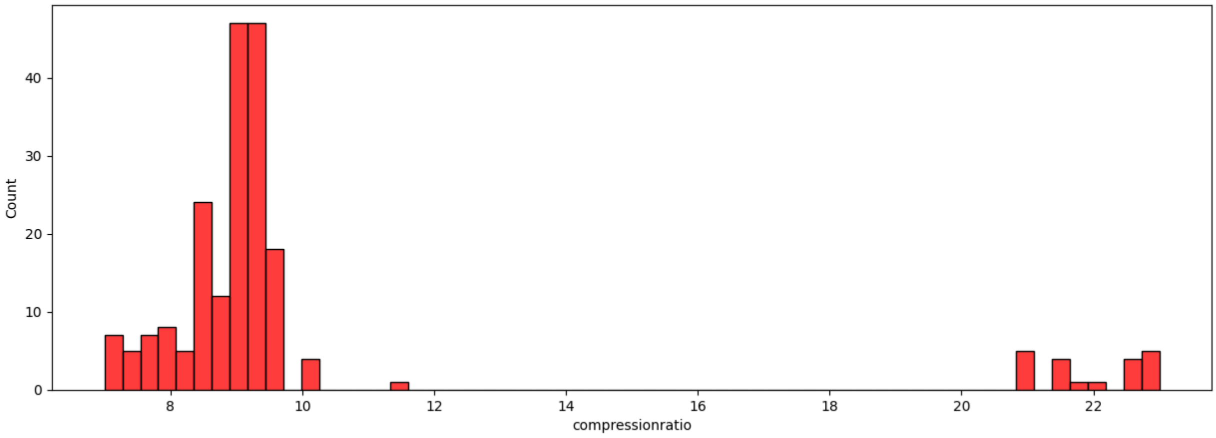
1.9476550452788108



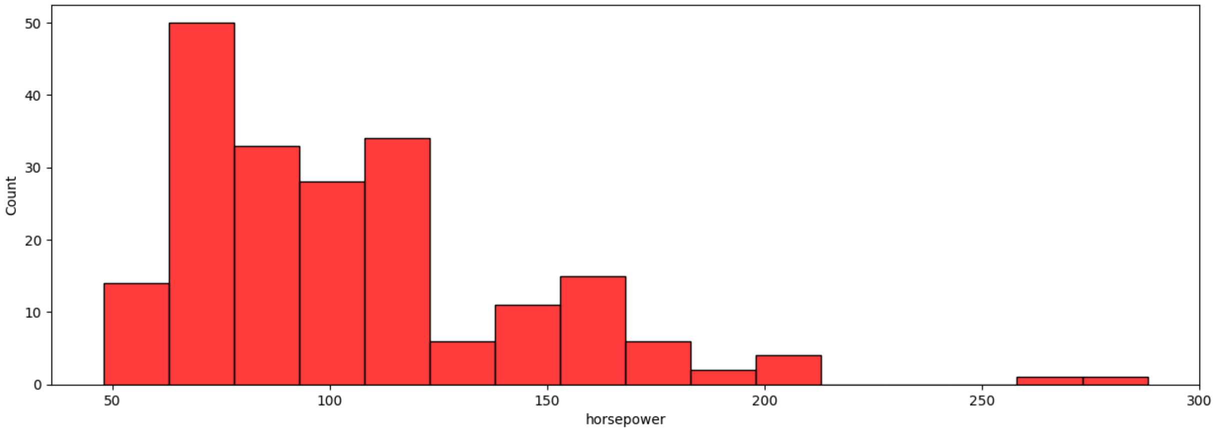
0.02015641810424137



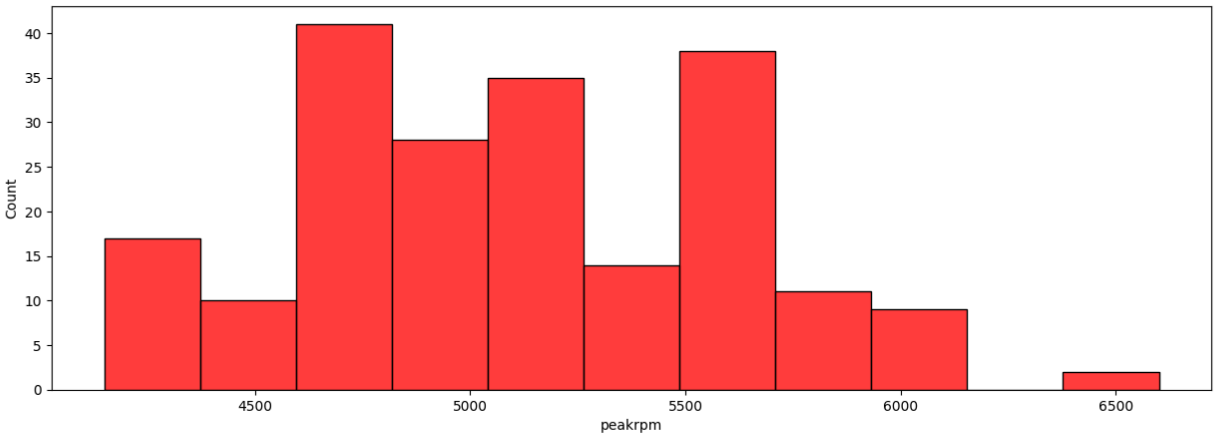
-0.6897045784233837



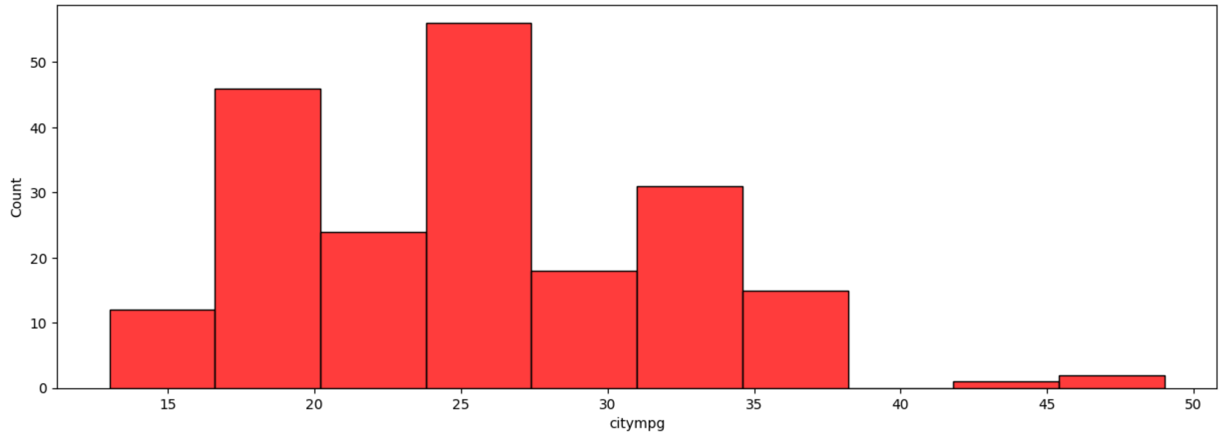
2.6108624576151533



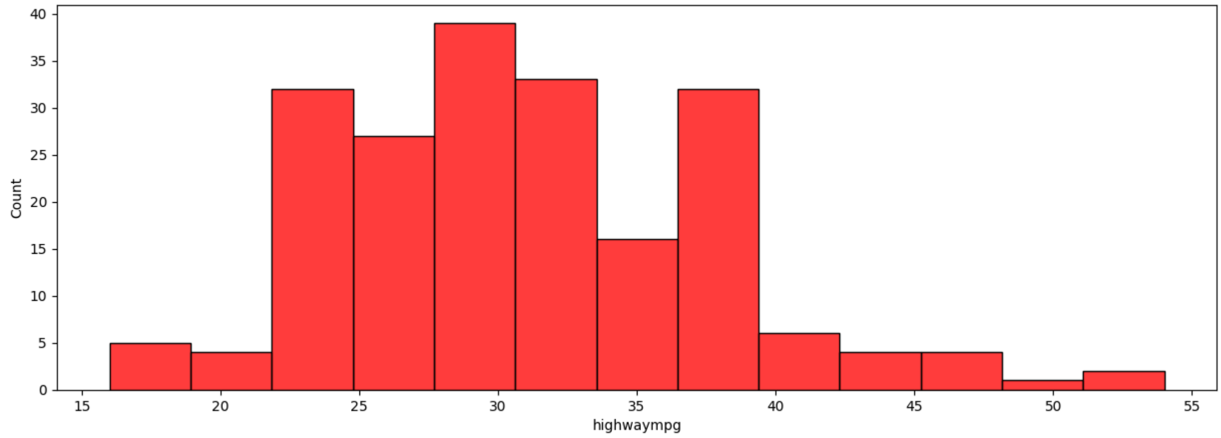
1.4053101543373119



0.07515872237118956




```
0.6637040288148164
```




```
0.5399971878746754
```

Treat Numerical Features

In [ ]:

```
# For features que has high skewness, let's say higher than 0.75, we will apply some
skew_limit = 0.75 # This is our threshold-limit to evaluate skewness. Overall below
skew_vals = df[numerical].skew()
skew_cols = skew_vals[abs(skew_vals) > skew_limit].sort_values(ascending=False)
display(skew_cols)
print("*"*50)

# Methods to reduce skewness
# Standard Scaler - do not reduce skewness, only scale values
print("Standard Scaler")
scaler = preprocessing.StandardScaler()
dfscaler = df.copy()
for coluna in skew_cols.index:
    dfscaler[coluna] = scaler.fit_transform(dfscaler[coluna].values.reshape(-1,1))
    #display(df_scaled[:,0])
    print(f"{coluna}: {dfscaler[coluna].skew():.3}")
print("*"*50)

# NP log
print("NP Log")
dflog = df.copy()
for coluna in skew_cols.index:
    dflog[coluna] = np.log(dflog[coluna])
    print(f"{coluna}: {dflog[coluna].skew():.3}")
print("*"*50)

# Cube Root
print("Cube Root")
dfcube = df.copy()
for coluna in skew_cols.index:
```

```python
    dfcube[coluna] = (dfcube[coluna]**(1/3))
    print(f"{coluna}: {dfcube[coluna].skew():.3}")
print("*"*50)

#Box-transformation
import scipy
print("Box Transformation")
dfbox = df.copy()
for coluna in skew_cols.index:
    dfbox[coluna],fitted_lambda = scipy.stats.boxcox(dfbox[coluna],lmbda=None)
    print(f"{coluna}: {dfbox[coluna].skew()}")
print("*"*50)

# Power Transformation
print("Power Transformation")
pt = PowerTransformer(method='yeo-johnson')
dfpow = df[skew_cols.index].copy()
trans= pt.fit_transform(dfpow)
df_trans = pd.DataFrame(trans, columns = skew_cols.index)
print(df_trans.skew())
print("*"*50)
```

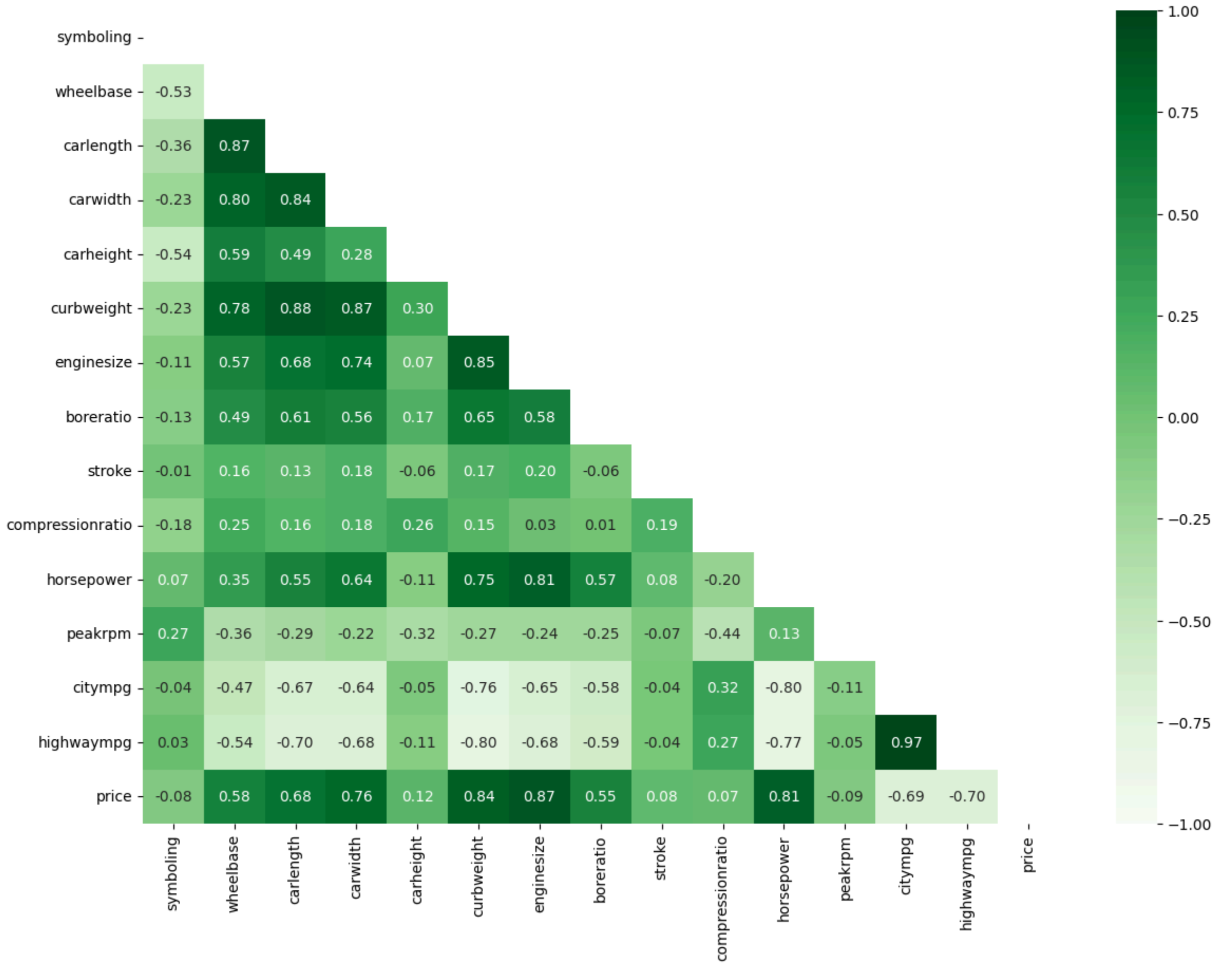
```
compressionratio    2.610862
enginesize          1.947655
horsepower          1.405310
wheelbase           1.050214
carwidth            0.904003
dtype: float64
**************************************************
Standard Scaler
compressionratio: 2.61
enginesize: 1.95
horsepower: 1.41
wheelbase: 1.05
carwidth: 0.904
**************************************************
NP Log
compressionratio: 2.35
enginesize: 0.858
horsepower: 0.483
wheelbase: 0.883
carwidth: 0.814
**************************************************
Cube Root
compressionratio: 2.46
enginesize: 1.19
horsepower: 0.753
wheelbase: 0.939
carwidth: 0.844
**************************************************
Box Transformation
compressionratio: 0.00802635805928132
enginesize: -0.0031895282437649046
horsepower: 0.0483634326890549
wheelbase: 0
carwidth: 0
**************************************************
Power Transformation
compressionratio    0.034222
enginesize         -0.002542
horsepower          0.049318
wheelbase          -0.003485
carwidth            0.000000
dtype: float64
**************************************************
compressionratio    0.034222
enginesize         -0.002542
horsepower          0.049318
```

```
wheelbase          -0.003485
carwidth            0.000000
dtype: float64
**************************************************
```

Both Box Transformation or Power Transformation has improved the skewness of the base

In [ ]:
```python
numericalprice= df.select_dtypes('number').columns


matrix = np.triu(df[numericalprice].corr())
fig, ax = plt.subplots(figsize=(14,10))
sns.heatmap (df[numericalprice].corr(), annot=True, fmt= '.2f', vmin=-1, vmax=1, cen
```

Out[ ]:
```
<Axes: >
```



The graph shows 9 numerical features have more than .5 correlation with the price Highwaympg and citympg has .97 correlation. We can drop one of them to avoid multicollinearity problems for the linear models, we will keep highwaympg

In [ ]:
```python
df = df.drop("citympg", axis=1)
```

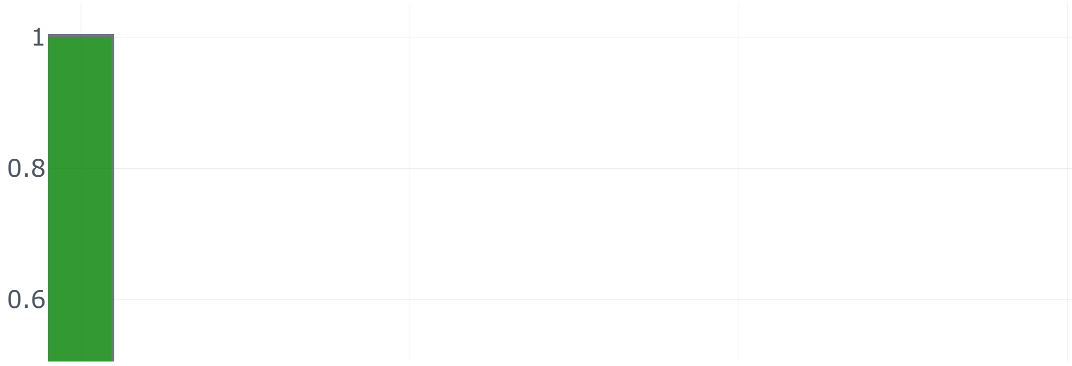
Analyse Categorical Features

In [ ]:
```python
categorical = df.select_dtypes('object').columns
```

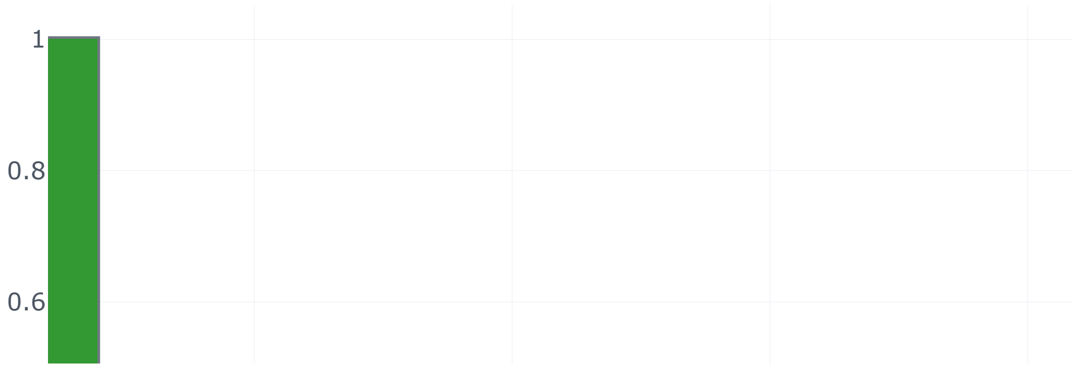
In [ ]:
```python
for category in categorical:
    print(df.groupby(category)['price'].mean().sort_values())
```

```
    print()
    df.groupby(category)['price'].mean().iplot(kind='histogram', subplots=True, bins
```

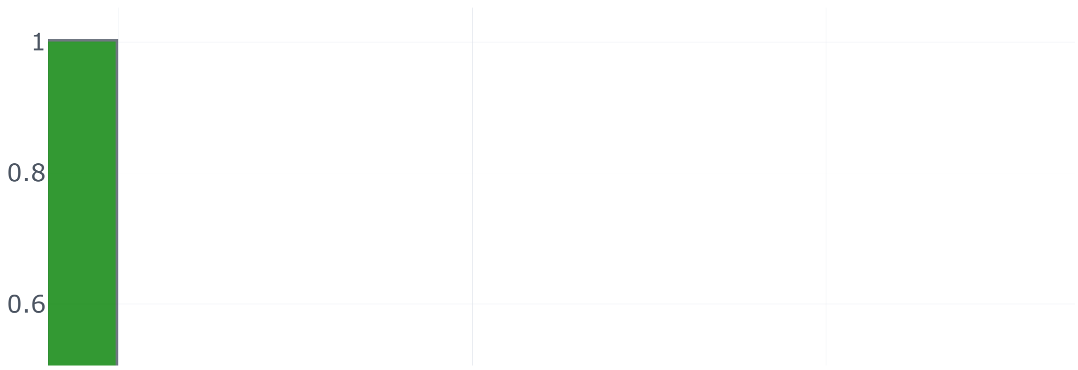
```
fueltype
gas       12999.7982
diesel    15838.1500
Name: price, dtype: float64
```



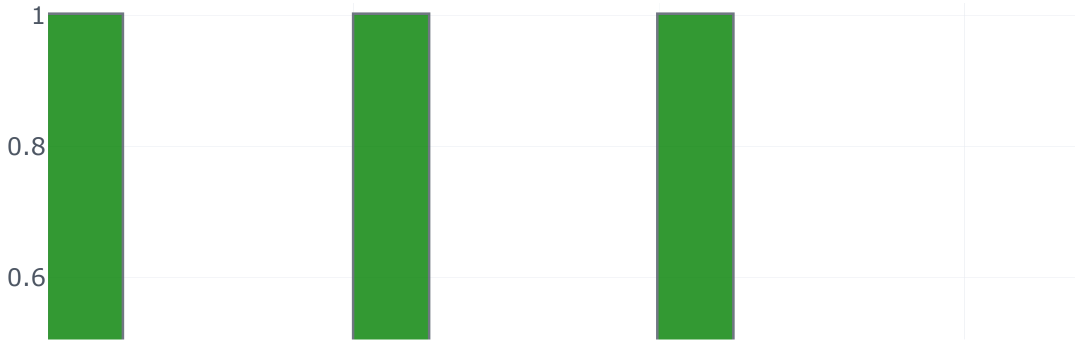

```
aspiration
std      12611.270833
turbo    16298.166676
Name: price, dtype: float64
```



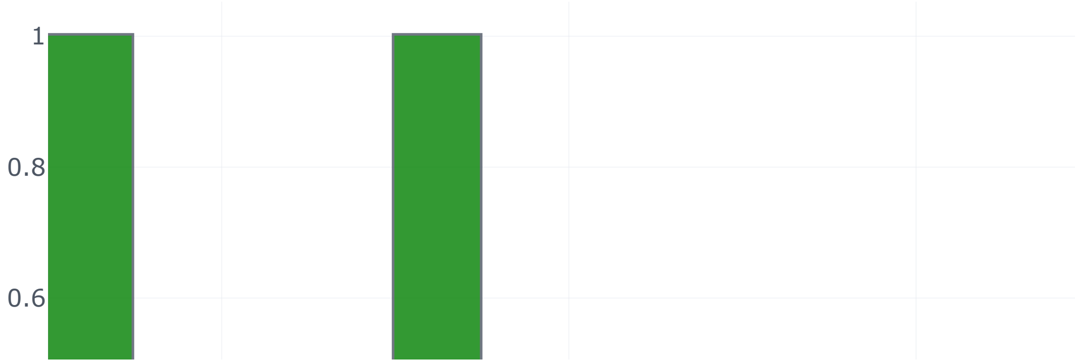
```
doornumber
two     12989.924078
four    13501.152174
Name: price, dtype: float64
```



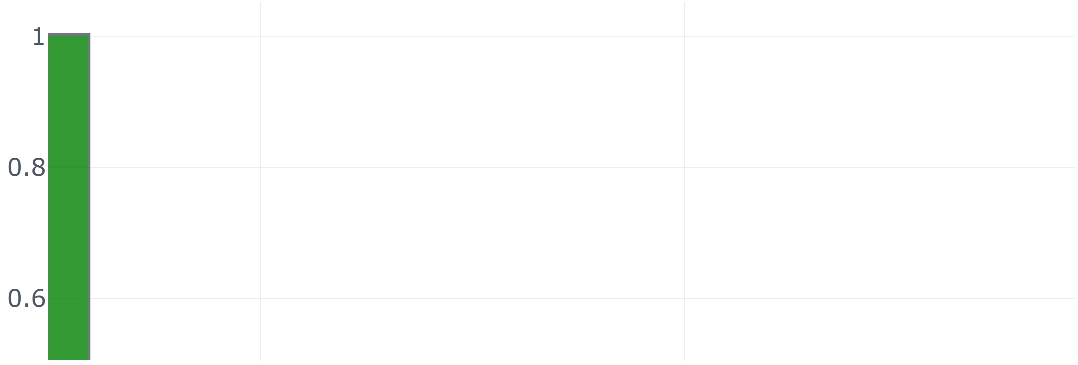

```
carbody
hatchback      10376.652386
wagon          12371.960000
sedan          14344.270833
convertible    21890.500000
hardtop        22208.500000
Name: price, dtype: float64
```




```
drivewheel
fwd      9239.308333
4wd     11087.463000
rwd     19910.809211
Name: price, dtype: float64
```

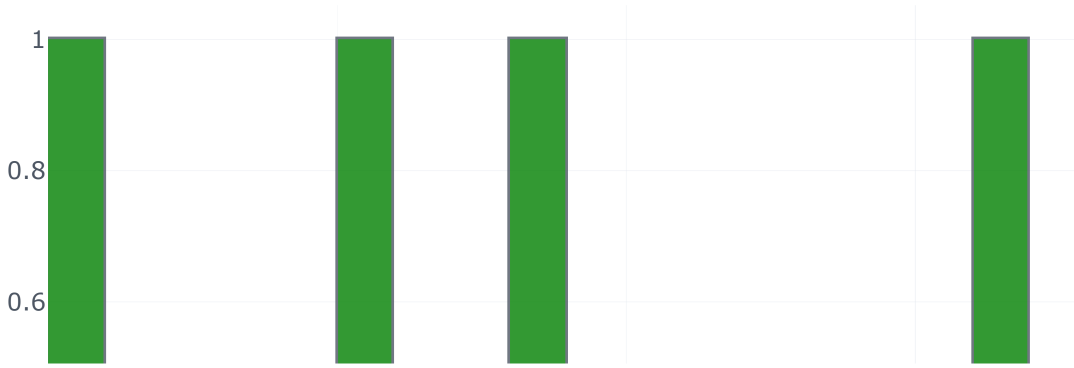


```
enginelocation
front    12961.097361
rear     34528.000000
Name: price, dtype: float64
```




```
enginetype
ohc      11574.048426
rotor    13020.000000
ohcf     13738.600000
l        14627.583333
dohc     18116.416667
ohcv     25098.384615
dohcv    31400.500000
Name: price, dtype: float64
```

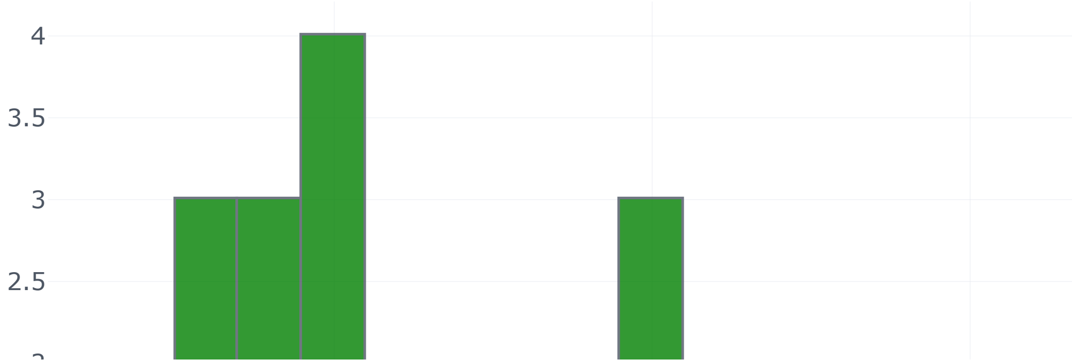
```
cylindernumber
three      5151.000000
four      10285.754717
two       13020.000000
five      21630.469727
six       23671.833333
twelve    36000.000000
eight     37400.100000
Name: price, dtype: float64
```




```
fuelsystem
2bbl     7478.151515
1bbl     7555.545455
spdi    10990.444444
spfi    11048.000000
4bbl    12145.000000
mfi     12964.000000
idi     15838.150000
```

```
mpfi    17754.602840
Name: price, dtype: float64
```


```
brand
chevrolet       6007.000000
dodge           7875.444444
plymouth        7963.428571
honda           8184.692308
subaru          8541.250000
isuzu           8916.500000
mitsubishi      9239.769231
renault         9595.000000
toyota          9885.812500
volkswagen     10077.500000
nissan         10415.666667
mazda          10652.882353
saab           15223.333333
peugeot        15489.090909
alfa-romero    15498.333333
mercury        16503.000000
audi           17859.166714
volvo          18063.181818
bmw            26118.750000
porsche        31400.500000
buick          33647.000000
jaguar         34600.000000
Name: price, dtype: float64
```



Treat Categorical Features

In [ ]:
```
df_dummyes = pd.get_dummies(df, columns=categorical, drop_first=True)
```

Definie Models to be tested

Models:

- Linear Regression (Baseline)
- Ridge
- Lasso
- Elasticnet
- KNeighboursRegressor
- Support Vector Machine Regressor
- Linear Support Vector Macine
- Random Forest
- Gradient Boosting
- Extra Trees
- XGBoost Regression
- Stochastic Gradient Descent
- PyTorch

Running Models

In [ ]:
```
# Evaluate Models
def evaluate_model(y_teste, pred, X_test):
    r2 = r2_score(y_teste, pred)
    Adj_r2 = 1 - ((1 - r2) * ((len(y_teste) - 1) / (len(y_teste) - X_test.shape[1] -
```

```
    RMSE_val = np.sqrt(mean_squared_error(y_teste, pred))
    return r2, Adj_r2, RMSE_val
```

In [ ]:

```
rmse_test = []
r2_test = []
time_test = []
r2_aj_test = []

numerical2= df_dummyes.drop(['price'], axis=1).select_dtypes('number').columns

y = df_dummyes['price']
X = df_dummyes.drop('price', axis=1)

#X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat

# Data for Tensor models
X_train_tensor = torch.from_numpy(np.array(X_train).astype(np.float32))
y_train_tensor = torch.from_numpy(np.array(y_train).astype(np.float32)).reshape(-1,
X_test_tensor = torch.from_numpy(np.array(X_test).astype(np.float32))
y_test_tensor = torch.from_numpy(np.array(y_test).astype(np.float32)).reshape(-1, 1)

s = StandardScaler()
p = PowerTransformer(method='yeo-johnson', standardize=True)

ct = make_column_transformer((s,numerical2),(p,skew_cols.index),remainder='passthrou
```

In [ ]:

```
model_lr = make_pipeline(ct, LinearRegression())
model_rd = make_pipeline(ct, Ridge())
model_la = make_pipeline(ct, Lasso())
model_el = make_pipeline(ct, ElasticNet())
model_kr = make_pipeline(ct, KNeighborsRegressor())
model_rf = make_pipeline(ct, RandomForestRegressor(random_state=42))
model_gb = make_pipeline(ct, GradientBoostingRegressor(random_state=42))
model_et = make_pipeline(ct, ExtraTreesRegressor(random_state=42))
model_xg = make_pipeline(ct, XGBRegressor(random_state=42, n_estimators=1000, max_de
model_sv = make_pipeline(ct, SVR()) # Demora muito
model_ls = make_pipeline(ct, LinearSVR(random_state=42, tol=1e-5, dual=True))
model_sg = make_pipeline(ct, SGDRegressor(max_iter=1000, tol=1e-3))
```

In [ ]:

```
# PyTorch model
model_pt = nn.Sequential(
    nn.Linear(63, 63),
    nn.ReLU(),
    nn.Linear(63, 126),
    nn.ReLU(),
    nn.Linear(126, 252),

    nn.Linear(252, 252),
    nn.Linear(252, 252),
    nn.Linear(252, 252),
    nn.Linear(252, 252),
    nn.Linear(252, 252),

    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
```

```
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),
    # nn.Linear(252, 252),

    nn.ReLU(),
    nn.Linear(252, 126),
    nn.ReLU(),
    nn.Linear(126, 63),
    nn.ReLU(),
    nn.Linear(63, 32),
    nn.ReLU(),
    nn.Linear(32, 16),
    nn.ReLU(),
    nn.Linear(16, 8),
    nn.ReLU(),
    nn.Linear(8, 4),
    nn.ReLU(),
    nn.Linear(4, 1)
)

# loss function and optimizer
loss_fn = nn.MSELoss()  # mean square error
optimizer = optim.Adam(model_pt.parameters(), lr=0.001)

n_epochs = 100   # number of epochs to run
batch_size = 15  # size of each batch
batch_start = torch.arange(0, len(X_train_tensor), batch_size)

# Hold the best model
best_mse = np.inf   # init to infinity
best_weights = None
history = []
```

In [ ]:

```
models = {
            'LinearRegression': model_lr,
            'Ridge': model_rd,
            'Lasso': model_la,
            'ElasticNet': model_el,
            'KNeighborsRegressor': model_kr,
            'RandomForest': model_rf,
            'GradientBoostingRegressor': model_gb,
            'ExtraTrees': model_et,
            'XGBoost': model_xg,
            'Support Vector Machine': model_sv, # demora muito
            'Linear Support Vector Machine': model_ls,
            'Stochastic Gradient Descent': model_sg,
            'PyTorch': model_pt,
          }
```

In [ ]:

```
for modelname, model in models.items():
    starttime = datetime.datetime.now()
    if modelname == 'PyTorch':
        patience_pt = 10
        for epoch in range(n_epochs):
            model.train()
            with tqdm.tqdm(batch_start, unit="batch", mininterval=0, disable=True) a
                bar.set_description(f"Epoch {epoch}")
                for start in bar:
                    # take a batch
                    X_batch = X_train_tensor[start:start+batch_size]
```

```
                    y_batch = y_train_tensor[start:start+batch_size]
                    # forward pass
                    y_pred = model_pt(X_batch)
                    loss = loss_fn(y_pred, y_batch)
                    # backward pass
                    optimizer.zero_grad()
                    loss.backward()
                    # update weights
                    optimizer.step()
                    # print progress
                    bar.set_postfix(mse=float(loss))
            pred = model_pt(X_test_tensor)
            #mse = loss_fn(y_pred, y_test_tensor)
            mse = loss_fn(y_pred, y_batch)
            mse = float(mse)
            history.append(mse)
            if mse < best_mse:
                best_mse = mse
                best_weights = copy.deepcopy(model_pt.state_dict())
                patience_pt = 10
            else: # Early_Stopping
                patience_pt -= 1
                if patience_pt == 0:
                    break
        # restore model and return best accuracy
        model_pt.load_state_dict(best_weights)
        pred = model_pt(X_test_tensor).data.numpy()
    else:
        #treinar
        model.fit(X_train, y_train)
        #testar
        pred = model.predict(X_test)

    r2, Adj_r2, RMSE_val = evaluate_model(y_test, pred, X_test)
    rmse_test.append(round(RMSE_val,2))
    r2_test.append(round(r2,4))
    r2_aj_test.append(round(Adj_r2,4))

    endtime = datetime.datetime.now()
    total_time = (endtime  - starttime).total_seconds()

    time_test.append(round(divmod(total_time, 60)[0]))

    print(f"{modelname}: R²: {r2} - R² Adjusted: {Adj_r2:.2} - RMSE: {RMSE_val} - To
    print('#'*50)
```

```
LinearRegression: R²: 0.9139544147243005 - R² Adjusted: 3.6 - RMSE: 2441.641846805242
5 - Total Time: 0.343628
##################################################
Ridge: R²: 0.9115443026138744 - R² Adjusted: 3.7 - RMSE: 2475.6005388642375 - Total T
ime: 0.062501
##################################################
Lasso: R²: 0.91714511070776 - R² Adjusted: 3.5 - RMSE: 2395.944389905374 - Total Tim
e: 0.064282
##################################################
ElasticNet: R²: 0.8378341353416652 - R² Adjusted: 5.9 - RMSE: 3351.948860709727 - Tot
al Time: 0.046876
##################################################
```

```
c:\Users\Thiago Kato\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn
\linear_model\_coordinate_descent.py:678: ConvergenceWarning:

Objective did not converge. You might want to increase the number of iterations, chec
k the scale of the features or consider increasing regularisation. Duality gap: 1.842
e+07, tolerance: 8.716e+05
```

```
KNeighborsRegressor: R²: 0.7504709969668497 - R² Adjusted: 8.6 - RMSE: 4157.939624458
654 - Total Time: 0.515148
##################################################
RandomForest: R²: 0.9437876275059895 - R² Adjusted: 2.7 - RMSE: 1973.4840507369634 -
Total Time: 0.556734
##################################################
GradientBoostingRegressor: R²: 0.9174653213377156 - R² Adjusted: 3.5 - RMSE: 2391.310
085496615 - Total Time: 0.226056
##################################################
ExtraTrees: R²: 0.9120446172749641 - R² Adjusted: 3.7 - RMSE: 2468.5894834137966 - To
tal Time: 0.587865
##################################################
XGBoost: R²: 0.9110440637384132 - R² Adjusted: 3.7 - RMSE: 2482.5907370463697 - Total
Time: 2.352262
##################################################
Support Vector Machine: R²: -0.0670284260204892 - R² Adjusted: 3.4e+01 - RMSE: 8598.1
62679165762 - Total Time: 0.069006
##################################################
Linear Support Vector Machine: R²: -2.3757470840771946 - R² Adjusted: 1e+02 - RMSE: 1
5293.349504576163 - Total Time: 0.075279
##################################################
Stochastic Gradient Descent: R²: 0.8925125483746238 - R² Adjusted: 4.3 - RMSE: 2728.9
56081430027 - Total Time: 0.08476
##################################################
PyTorch: R²: -2.428640711441036 - R² Adjusted: 1.1e+02 - RMSE: 15412.697400958736 - T
otal Time: 45.694181
##################################################
```

Define best model

In [ ]:

```
model_results = pd.DataFrame()

model_results['Models'] = models.keys()
model_results['R²'] = r2_test
model_results['RMSE'] = rmse_test
model_results['R² Adjusted'] = r2_aj_test
```

In [ ]:

```
model_results.sort_values("R²", ascending=False)
```

Out[ ]:

	Models	R^2	RMSE	R^2 Adjusted
5	RandomForest	0.9438	1973.48	2.7145
6	GradientBoostingRegressor	0.9175	2391.31	3.5173
2	Lasso	0.9171	2395.94	3.5271
0	LinearRegression	0.9140	2441.64	3.6244
7	ExtraTrees	0.9120	2468.59	3.6826
1	Ridge	0.9115	2475.60	3.6979
8	XGBoost	0.9110	2482.59	3.7132
11	Stochastic Gradient Descent	0.8925	2728.96	4.2784
3	ElasticNet	0.8378	3351.95	5.9461
4	KNeighborsRegressor	0.7505	4157.94	8.6106
9	Support Vector Machine	-0.0670	8598.16	33.5444
10	Linear Support Vector Machine	-2.3757	15293.35	103.9603
12	PyTorch	-2.4286	15412.70	105.5735

In [ ]:

```
best_model = model_results.sort_values("R²", ascending=False).iloc[0,0]

# Get rid of pipeline to run Feature Importance
model_lr = LinearRegression()
model_rd = Ridge()
model_la = Lasso()
model_el = ElasticNet()
model_kr = KNeighborsRegressor()
model_rf = RandomForestRegressor(random_state=42)
model_gb = GradientBoostingRegressor(random_state=42)
model_et = ExtraTreesRegressor(random_state=42)
model_xg = XGBRegressor(random_state=42, n_estimators=1000, max_depth=10)
model_sv = SVR() # Demora muito
model_ls = LinearSVR(random_state=42, tol=1e-5, dual=True)
model_sg = SGDRegressor(max_iter=1000, tol=1e-3)

models = {
            'LinearRegression': model_lr,
            'Ridge': model_rd,
            'Lasso': model_la,
            'ElasticNet': model_el,
            'KNeighborsRegressor': model_kr,
            'RandomForest': model_rf,
            'GradientBoostingRegressor': model_gb,
            'ExtraTrees': model_et,
            'XGBoost': model_xg,
            'Support Vector Machine': model_sv, # demora muito
            'Linear Support Vector Machine': model_ls,
            'Stochastic Gradient Descent': model_sg,
            'PyTorch': model_pt,
          }

model = models[best_model]
display(model)
```

```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Run best model with Hyperparameter Tuning

In [ ]:

```
if best_model == 'PyTorch':
    X_train_tensor = torch.from_numpy(np.array(X_train).astype(np.float32))
    y_train_tensor = torch.from_numpy(np.array(y_train).astype(np.float32)).reshape(

    X_test_tensor = torch.from_numpy(np.array(X_test).astype(np.float32))
    y_test_tensor = torch.from_numpy(np.array(y_test).astype(np.float32)).reshape(-1

    patience_pt = 3
    for epoch in range(n_epochs):
        model.train()
        with tqdm.tqdm(batch_start, unit="batch", mininterval=0, disable=True) as ba
            bar.set_description(f"Epoch {epoch}")
            for start in bar:
                # take a batch
                X_batch = X_train_tensor[start:start+batch_size]
                y_batch = y_train_tensor[start:start+batch_size]
                # forward pass
                y_pred = model_pt(X_batch)
                loss = loss_fn(y_pred, y_batch)
                # backward pass
                optimizer.zero_grad()
                loss.backward()
                # update weights
                optimizer.step()
                # print progress
                bar.set_postfix(mse=float(loss))
        pred = model_pt(X_test_tensor)
        #mse = loss_fn(y_pred, y_test)
        #mse = loss_fn(y_pred, y_test_tensor)
        mse = loss_fn(y_pred, y_batch)
        mse = float(mse)
        history.append(mse)
        if mse < best_mse:
            best_mse = mse
            best_weights = copy.deepcopy(model_pt.state_dict())
            patience_pt = 3
        else: # Early_Stopping
            patience_pt -= 1
            if patience_pt == 0:
                break
    # restore model and return best accuracy
    model_pt.load_state_dict(best_weights)
    pred = model_pt(X_test_tensor).data.numpy()
else:
    #treinar
    model.fit(X_train, y_train)
    #testar
    pred = model.predict(X_test)

r2, Adj_r2, RMSE_val = evaluate_model(y_test, pred, X_test)

print(f"The Best Model is {best_model}")
print(f"with R² calculates of: {r2}")
print(f"and RMSE: {RMSE_val}")
```

```
The Best Model is RandomForest
with R² calculates of: 0.944344862993795
and RMSE: 1963.6780754493386
```

Feature Importance

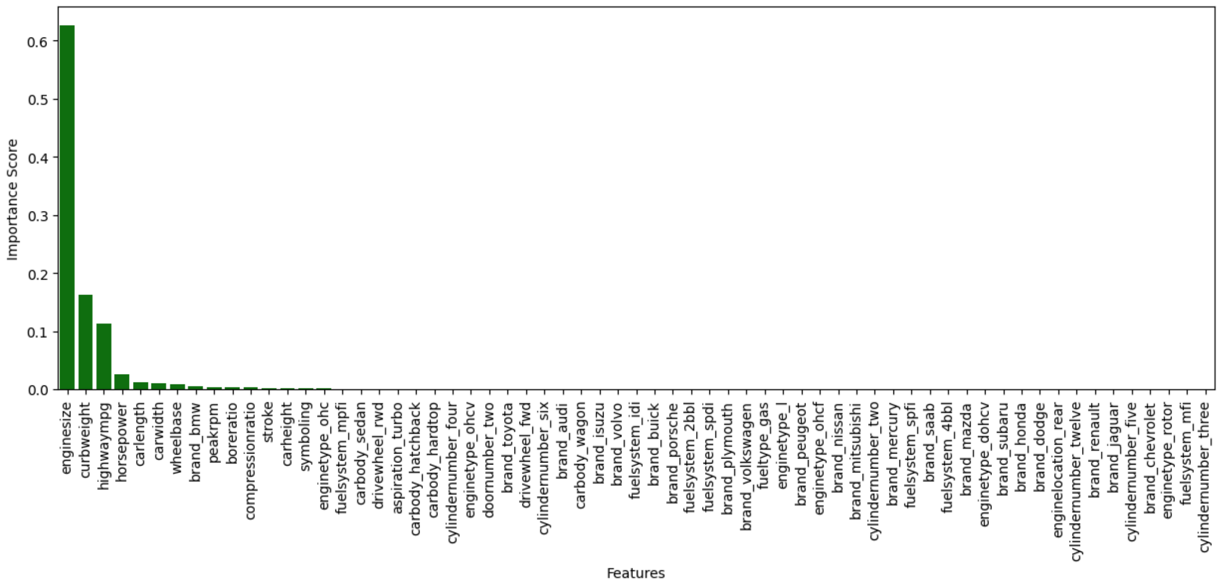
In [ ]:

```python
importance_features = pd.DataFrame(model.feature_importances_, X_train.columns)
importance_features = importance_features.sort_values(by=0, ascending=False)

num=0

for i in range(len(importance_features)):
    importance = importance_features.iloc[i, 0]
    if importance > 0.001:
        print(f'{X_train.columns[i]} : {round(importance,3)}')
        num+=1

print(f"There are {num} important features, with score higher than 0.001")

plt.figure(figsize=(15, 5))
ax = sns.barplot(x=importance_features.index, y=importance_features[0], color='Green
ax.tick_params(axis='x', rotation=90)
ax.set(xlabel='Features', ylabel='Importance Score')
```

```
symboling : 0.627
wheelbase : 0.164
carlength : 0.114
carwidth : 0.026
carheight : 0.012
curbweight : 0.011
enginesize : 0.009
boreratio : 0.006
stroke : 0.005
compressionratio : 0.004
horsepower : 0.004
peakrpm : 0.003
highwaympg : 0.003
fueltype_gas : 0.002
aspiration_turbo : 0.002
doornumber_two : 0.001
carbody_hardtop : 0.001
There are 17 important features, with score higher than 0.001
```

Out[ ]:

```
[Text(0.5, 0, 'Features'), Text(0, 0.5, 'Importance Score')]
```



Features with higher importance are:

- enginesize : 0.627
- curbweight : 0.164
- highwaympg : 0.114

RandomForest model gave score higher than 0 to 17 features, out of 63 features

Conclusion

This is a simple study on a few Regression Models. We have made some exploratory analysis on the dataset, defined treatments for numerical and categorical values, defined regression models and defined metrics to evaluate each of them