

Aluno: Ernesto Gurgel Valente Neto

Trabalho de Conclusão de Curso - Artigo Academico

1.0 PREPARANDO O AMBIENTE E IMPORTANDO AS BIBLIOTECAS

- IMPORTAR AS BIBLIOTECAS NECESSARIAS PARA ANALISE DE DADOS
- MONTAR O DRIVE PARA LEITURA DOS DADOS

```
#Importando as Bibliotecas de Analise de Dados
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

TRATAMENTO

- VERIFICAÇÃO DA BASE DE DADOS
- VERIFICAÇÃO DE VALORES MISSING

```
df = pd.read_csv("/content/drive/MyDrive/TCC/Car_Prices.csv")
```

```
#Verificando o dataset
df.head()
```

	Unnamed: 0	mark	model	generation_name	year	mileage	vol_engine	fuel	ci
0	0	opel	combo	gen-d-2011	2015	139568	1248	Diesel	Ja
1	1	opel	combo	gen-d-2011	2018	31991	1499	Diesel	Katow
2	2	opel	combo	gen-d-2011	2015	278437	1598	Diesel	Brz
3	3	opel	combo	gen-d-2011	2016	47600	1248	Diesel	Korfant

```
#Verificando o dataset
```

```
len(df)
```

```
117927
```

▼ 2.0 ANALISE DE DADOS

```
#Verificando os tipos de dados
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 117927 entries, 0 to 117926
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Unnamed: 0        117927 non-null   int64  
 1   mark              117927 non-null   object  
 2   model              117927 non-null   object  
 3   generation_name   87842 non-null    object  
 4   year               117927 non-null   int64  
 5   mileage             117927 non-null   int64  
 6   vol_engine          117927 non-null   int64  
 7   fuel                117927 non-null   object  
 8   city                117927 non-null   object  
 9   province            117927 non-null   object  
 10  price               117927 non-null   int64  
dtypes: int64(5), object(6)
memory usage: 9.9+ MB
```

```
#Verificando valores nulos
```

```
df.isna().sum()
```

```
Unnamed: 0          0
mark              0
model             0
generation_name  30085
year              0
mileage            0
vol_engine         0
fuel               0
city               0
province           0
price              0
dtype: int64
```

```
#Removendo valores nulos
```

```
df.drop(columns=["generation_name", "Unnamed: 0"], inplace=True)
```

```
#Verificando valores removidos
```

```
df.isna().sum()
```

```
mark          0
model         0
year          0
mileage        0
vol_engine     0
fuel           0
city           0
province       0
price          0
dtype: int64
```

```
count = (df['year'] < 2002).sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 2472

```
#['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']
count = (df['fuel'] == 'Diesel').sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 48476

```
#['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']
count = (df['fuel'] == 'CNG').sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 47

```
#['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']
count = (df['fuel'] == 'Gasoline').sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 61597

```
#['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']
count = (df['fuel'] == 'LPG').sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 4301

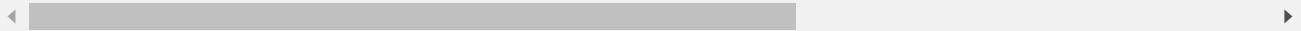
```
#['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']
count = (df['fuel'] == 'Hybrid').sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 2621

```
#['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']
count = (df['fuel'] == 'Electric').sum()
print("Valores abaixo do ano 2002: ", count)
```

Valores abaixo do ano 2002: 885


```
Preço $: [(19900, 1336), (39900, 1154), (29900, 1139), (18900, 1100), (14900, 1010)]
Preço $: menos comum [(417200, 1), (134558, 1), (328300, 1), (348930, 1), (290934, 1)]
```



VERIFICAÇÃO STATISTICA DOS DADOS DA BASE

1. ANALISE DOS CONJUNTOS DE DADOS
2. ANALISES ESTATISTICAS E DE DISTRIBUIÇÃO

```
df.describe()
```

	year	mileage	vol_engine	price
count	117927.000000	1.179270e+05	117927.000000	1.179270e+05
mean	2012.925259	1.409768e+05	1812.057782	7.029988e+04
std	5.690135	9.236936e+04	643.613438	8.482458e+04
min	1945.000000	0.000000e+00	0.000000	5.000000e+02
25%	2009.000000	6.700000e+04	1461.000000	2.100000e+04
50%	2013.000000	1.462690e+05	1796.000000	4.190000e+04
75%	2018.000000	2.030000e+05	1995.000000	8.360000e+04
max	2022.000000	2.800000e+06	7600.000000	2.399900e+06

```
dfnormal = df.copy()
```

```
#Tamanho Total da Base
df.count().sum()
```

```
1061343
```

```
#Verificação da distribuição dos dados de cada tipo
print("Verificação da Distribuição da Marca: ", df['mark'].nunique())
print("Verificação da Distribuição da Modelo: ", df['model'].nunique())
print("Verificação da Distribuição da Ano: ", df['year'].nunique())
print("Verificação da Distribuição da Kilometragem: ", df['mileage'].nunique())
print("Verificação da Distribuição da Mortor: ", df['vol_engine'].nunique())
print("Verificação da Distribuição da Combustivel: ", df['fuel'].nunique())
print("Verificação da Distribuição da Cidade: ", df['city'].nunique())
print("Verificação da Distribuição da Estado: ", df['province'].nunique())
print("Verificação da Distribuição da Preços: ", df['price'].nunique())
```

```
Verificação da Distribuição da Marca: 23
Verificação da Distribuição da Modelo: 328
Verificação da Distribuição da Ano: 54
Verificação da Distribuição da Kilometragem: 35394
Verificação da Distribuição da Mortor: 508
Verificação da Distribuição da Combustivel: 6
```

Verificação da Distribuição da Cidade: 4427

Verificação da Distribuição da Estado: 23

Verificação da Distribuição da Preços: 9310

```
print("Modelos de Marcas: ", df["mark"].unique())
print("________________________________")
print("Modelos de carros: ", df["model"].unique())
print("________________________________")
print("Modelos do Motor: ", df["vol_engine"].unique())
print("________________________________")
print("Tipo de Combustivel: ", df["fuel"].unique())
print("________________________________")
print("Cidade: ", df["city"].unique())
print("________________________________")
print("Estado: ", df["province"].unique())

'xsara-picasso' '500' '5001' '500x' 'bravo' 'doble' 'freemont'
'grande-punto' 'panda' 'punto' 'punto-evo' 'tipo' 'accord' 'cr-v' 'hr-v'
'jazz' 'civic' 'elantra' 'i10' 'i20' 'i30' 'i40' 'ix20' 'ix35' 'kona'
'santa-fe' 'tucson' 'carens' 'ceed' 'optima' 'picanto' 'pro-ceed'
'sorento' 'soul' 'sportage' 'stinger' 'stonic' 'venga' 'xceed' '2' '3'
'5' '6' 'cx-3' 'cx-5' 'cx-7' 'cx-9' 'cx-30' 'mx-5' 'clubman' 'cooper'
'cooper-s' 'countryman' 'one' 'asx' 'colt' 'eclipse-cross' 'lancer'
'outlander' 'space-star' 'almera' 'juke' 'leaf' 'micra' 'murano' 'note'
'patrol' 'primera' 'qashqai' 'qashqai-2' 'x-trail' '206' '207' '208'
'307' '308' '407' '508' '2008' '3008' '5008' 'expert' 'partner'
'alhambra' 'altea' 'alteaxl' 'arona' 'ateca' 'exeo' 'ibiza' 'leon'
'toledo' 'c30' 's40' 's60' 's80' 'v40' 'v50' 'v60' 'v70' 'v90' 'xc-40'
'xc-60' 'xc-70' 'xc-90']
```

Modelos do Motor:	1248	1499	1598	1400	1368	1600	1799	1796	1994	1998	2498	1995	21
2171	3175	2792	1597	2958	3000	1398	1364	999	1229	1200	1100	996	1242
1000	1199	973	998	0	1991	2231	3195	2405	2200	2000	2400	2384	1389
1686	1700	1300	1956	1354	1399	1798	1900	1800	1362	1341	1342	1496	1589
1500	1449	1488	1195	1187	1235	1198	1396	1560	3165	1997	1599	2700	1958
1953	2172	1870	2464	1999	1898	2463	1390	1896	1968	1595	1984	3200	1395
3189	1197	1490	1955	1596	1346	1698	1685	2962	2497	2959	2598	2771	2309
1498	1422	1781	1780	1881	1895	2976	2393	2496	2967	2698	3123	3197	2968
1963	2500	4172	2495	2390	2398	2773	4163	2671	4200	2800	1996	2995	2960
2996	2957	3993	1986	2986	2999	2989	1967	2998	2997	3936	3697	4134	5998
6299	3956	4000	3700	3328	4199	3597	2480	2894	2900	3996	4991	3998	3994
2993	4395	2979	649	647	1	400	3246	3999	3245	3201	2302	4999	3795
4935	3535	1495	2494	2793	1990	1795	1951	2926	1817	3498	4398	3495	4389
4799	1993	1969	3982	3990	5379	5972	3600	4423	3901	4400	6592	4399	6600
2966	2925	4619	4396	1949	1285	1784	2461	1989	2460	1588	1272	1760	1570
1391	2324	1890	1797	1981	1716	2370	3198	1386	2327	3168	4921	6000	1987
1360	1468	1489	2459	1388	1753	1562	3500	2694	3496	1992	1593	2485	2488
1983	2261	2990	4605	4015	3497	3490	2264	4601	4011	2956	2300	3958	4016
6200	4600	5400	3704	4948	4788	6210	2687	5000	4949	3726	5409	4996	6233
1299	1297	1250	1241	1084	2522	2521	1988	1497	2286	2688	2179	2483	2987
4009	4606	3731	4951	5038	3300	2298	3797	4700	5800	4737	4900	4727	5763
3802	7600	4730	6390	5812	6384	2499	2953	3196	1976	2402	1461	1332	4966
5439	5786	4967	5513	5514	5461	6208	4663	5462	5980	1950	2143	1333	1330
2295	3199	4266	2597	2685	2148	2600	4780	3997	2034	1699	2084	1397	1478
1689	2035	2151	2950	2149	2799	3606	2155	4196	3222	2397	2717	4194	2599
2199	3449	4973	5987	5991	3724	5500	4300	6300	4978	1750	1749	1322	1298
1558	898	899	900	1149	1618	2188	2946	1783	2183	1862	1451	989	1458
1590	1153	1329	1794	1328	2362	2487	3456	2194	1587	2982	4164	2755	3346
1500	2100	1661	1161	1602	2211	1706	1702	1700	1562	1070	2207	1850	1712

```
4988 3427 4004 4401 4080 3511 1270 1200 1770 1700 1770 2301 1057 1742
2891 1897 875 1206 1150 6162 6199 6102 3604 3822 3564 5666 5700 3800
3605 1868 2720 2992 500 499 650 550 2360 2359 1580 1251 1108 1747
1850 2254 2157 2354 2204 2358 2356 1246 1339 1245 988 1668 1591 1582
1086 1120 1353 1482 1975 1682 1349 1980 1985 2351 1691 2648 2656 1358
1561 3342 1594 2184 2191 1759 2260 2489 2267 1840 850 1565 2268 1124
1493 2399 1584 2972 2378 1193 1769 2168 1481 1240 4800 2826 4169 4479
1878 1550 2230 1666 1581 2796 2401 2435 1948 3192 4414 4415 2783 2922
2319 1477 1959 2935]
```

Tipo de Combustivel: ['Diesel' 'CNG' 'Gasoline' 'LPG' 'Hybrid' 'Electric']

Cidade: ['Janki' 'Katowice' 'Brzeg' ... 'Kolonia Gościneczycze' 'Augustówek' 'Bledzew']

PLOTAGEM DA DISTRIBUIÇÃO DA BASE DE DADOS

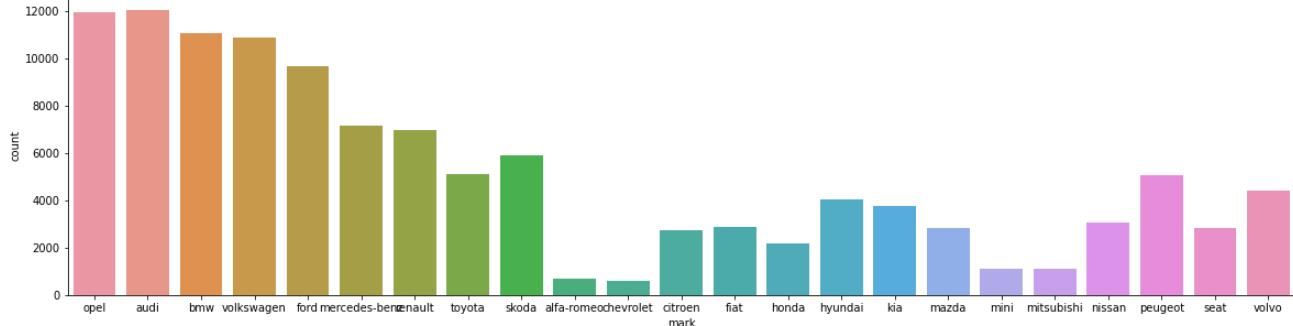
1. OBSERVAÇÃO DOS ATRIBUTOS

2. ANALISE DE OUTLIERS

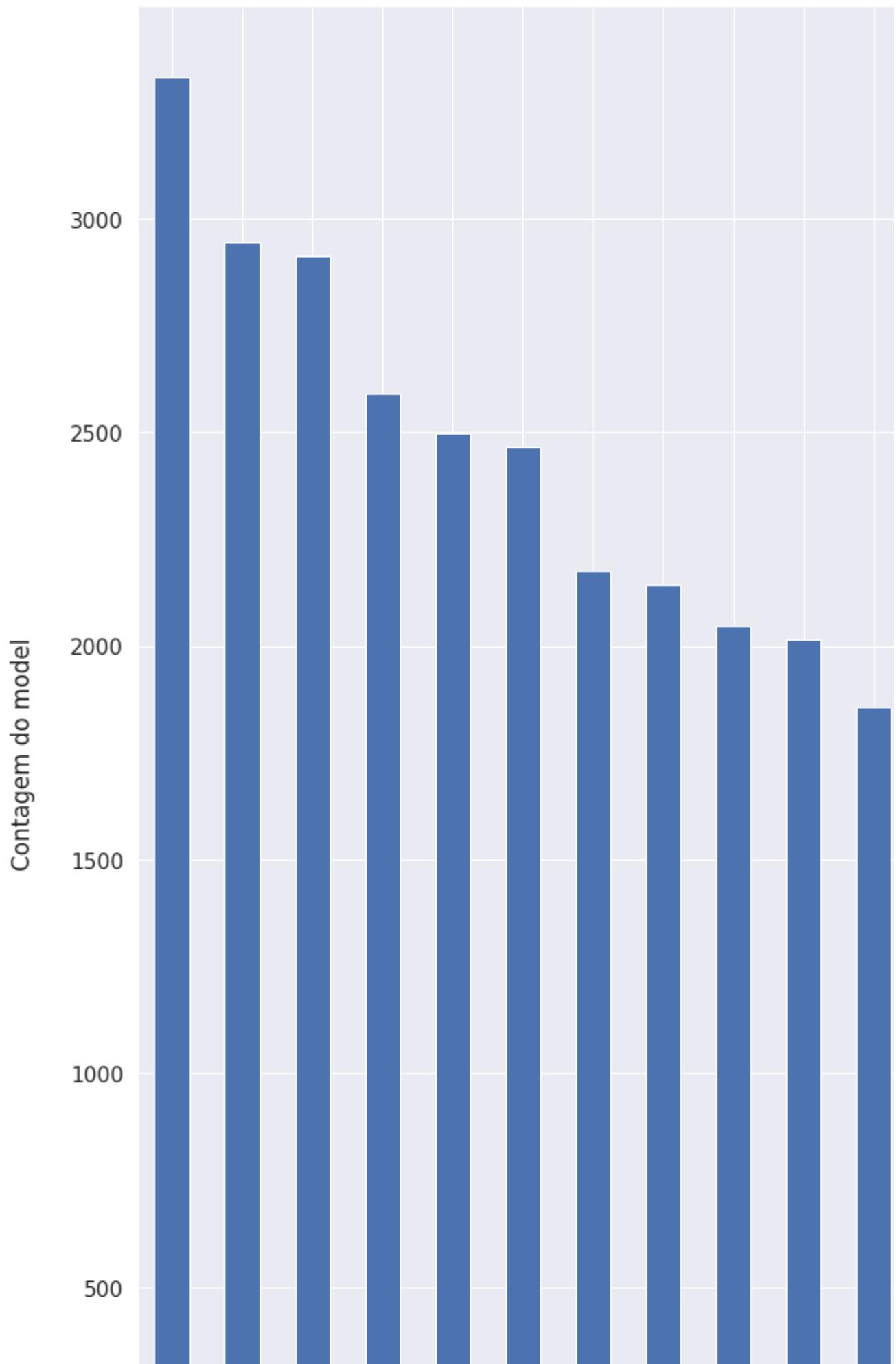
```
plt.figure(figsize=(20,5))
sns.countplot(df["mark"])
plt.show()
```

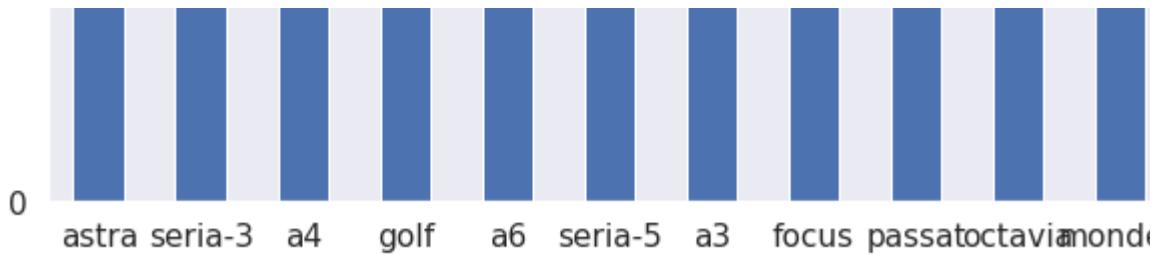
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass

FutureWarning

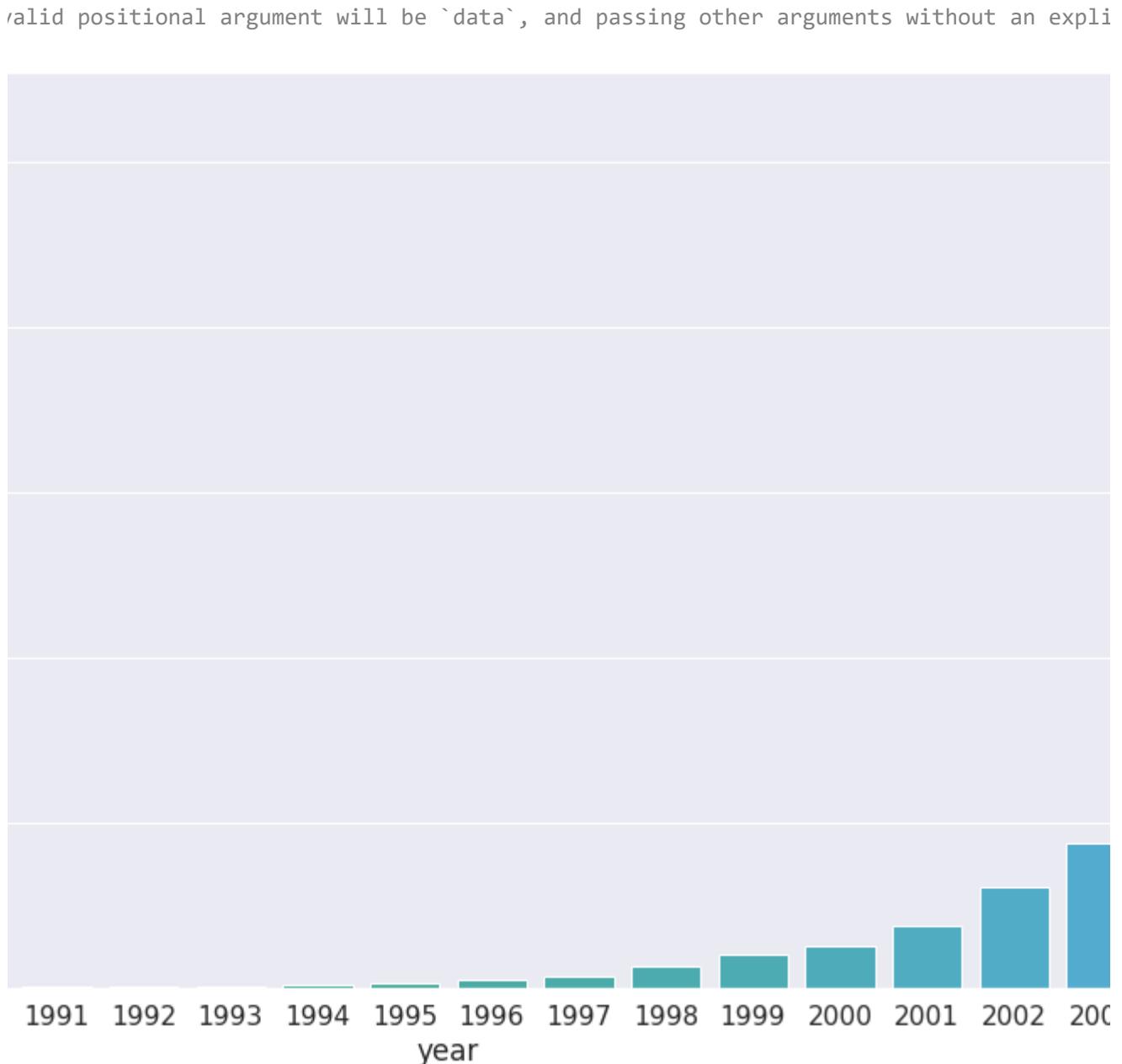


```
sns.set(font_scale=1.4)
df['model'].value_counts().plot(kind='bar', figsize=(300, 20), rot=0)
plt.xlabel("model", labelpad=25)
plt.ylabel("Contagem do model", labelpad=25)
plt.title("Hierarquia dos Modelos", y=1.02);
```



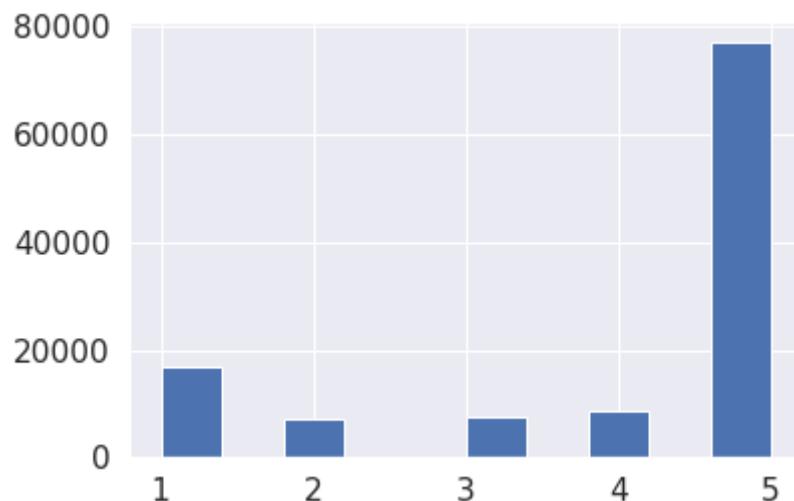


```
plt.figure(figsize=(50,10))
sns.countplot(df["year"])
plt.show()
```

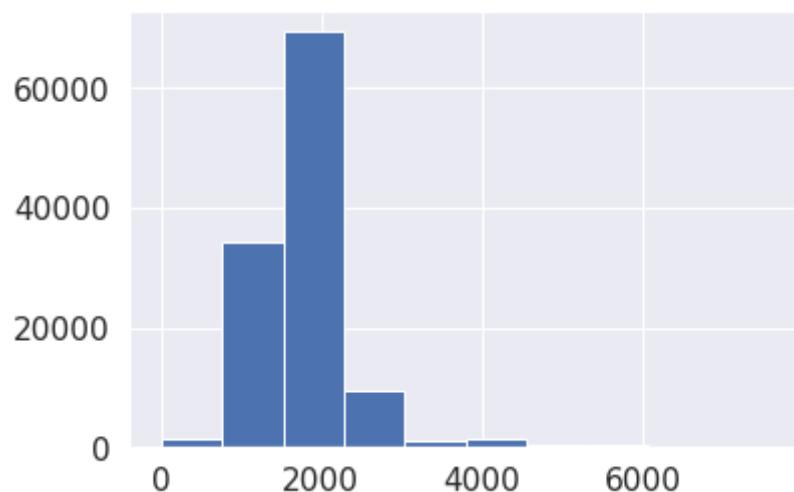


```
df["mileage_hist"] = pd.cut(df["mileage"],
                             bins=[0., 25000, 50000, 75000, 100000, 200000.],
                             labels=[1, 2, 3, 4, 5])
```

```
df["mileage_hist"].hist();
```



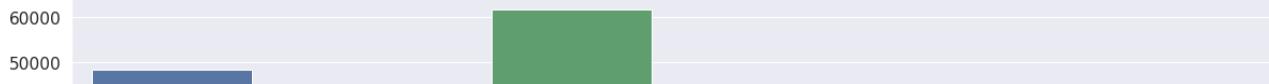
```
df["vol_engine"].hist();
```



```
plt.figure(figsize=(20,5))
sns.countplot(df["fuel"])
plt.show()
```

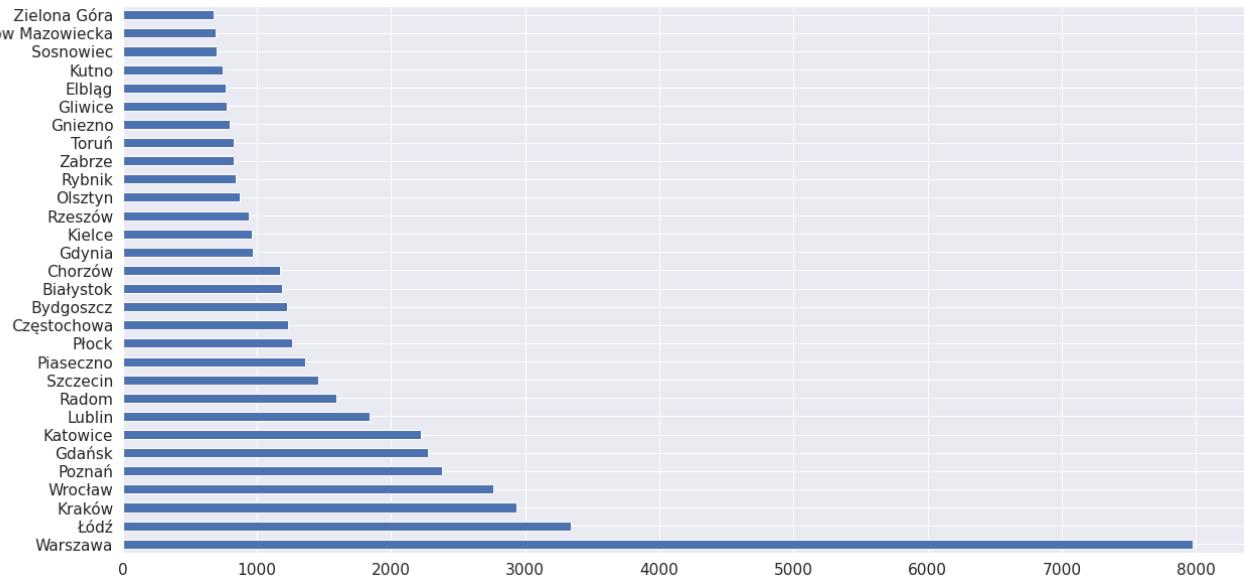
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
```

```
FutureWarning
```



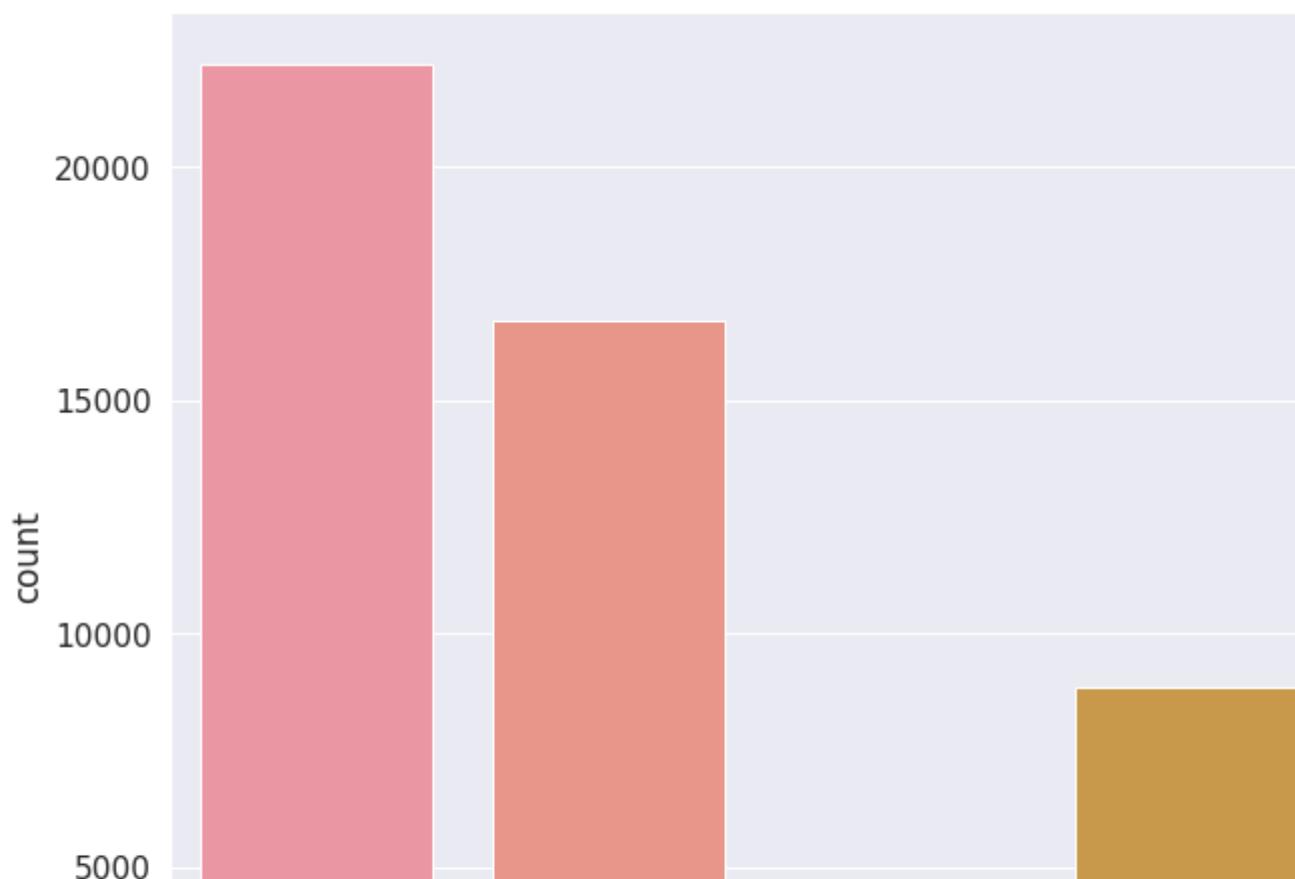
```
df['city'].value_counts().head(30).plot(kind='barh', figsize=(20,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f5603c1d0>
```



```
plt.figure(figsize=(60,10))
sns.countplot(df["province"])
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass  
FutureWarning
```



```
df['province'].value_counts().head(5000).plot(kind='barh', figsize=(20,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f5612d390>
```

Nordrhein-Westfalen

Trenczyn

(

Analise dos dados do plot anterior que estão abaixo da distribuição

Dodlaškie

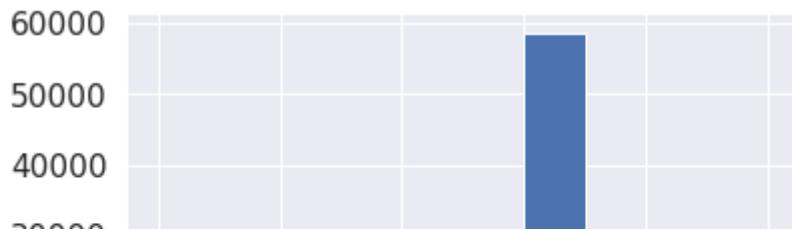
```
count = (df['province'] == 'Nordrhein-Westfalen').sum()
print("count", count)
count = (df['province'] == 'Trenczyn').sum()
print("count", count)
count = (df['province'] == '(').sum()
print("count", count)
count = (df['province'] == 'Niedersachsen').sum()
print("count", count)
count = (df['province'] == 'Wiedeń').sum()
print("count", count)
count = (df['province'] == 'Berlin').sum()
print("count", count)
count = (df['province'] == 'Moravian-Silesian Region').sum()
print("count", count)

count 1
count 1
count 1
count 1
count 2
count 3
count 35
```

Continuação outras analises

```
df["price_hist"] = pd.cut(df["price"],
                           bins=[0., 5000, 10000, 25000, 100000, 250000, 500000],
                           labels=[1, 2, 3, 4, 5, 6])
```

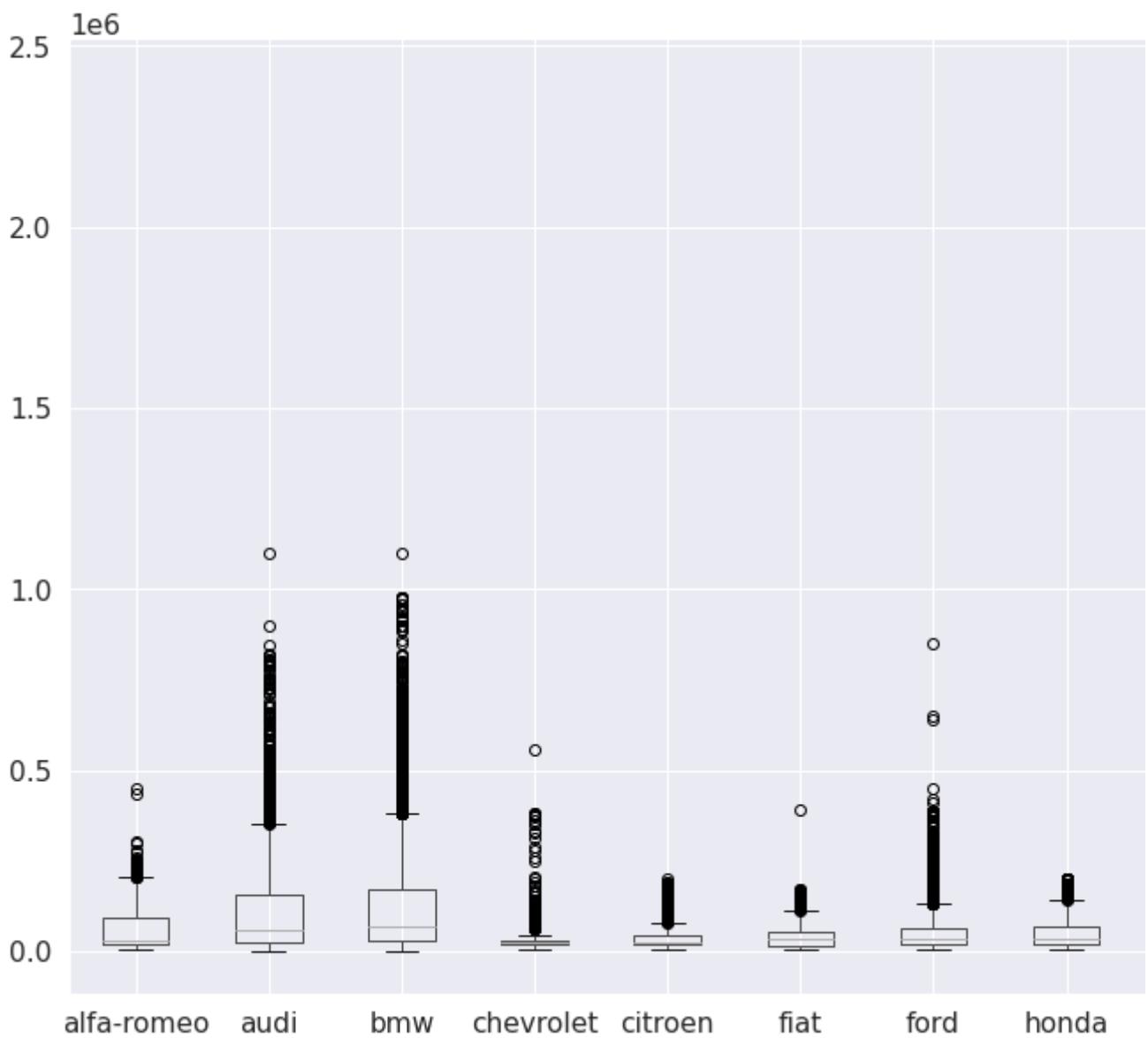
```
df["price_hist"].hist();
```



ANALISE DE OUTILIERS

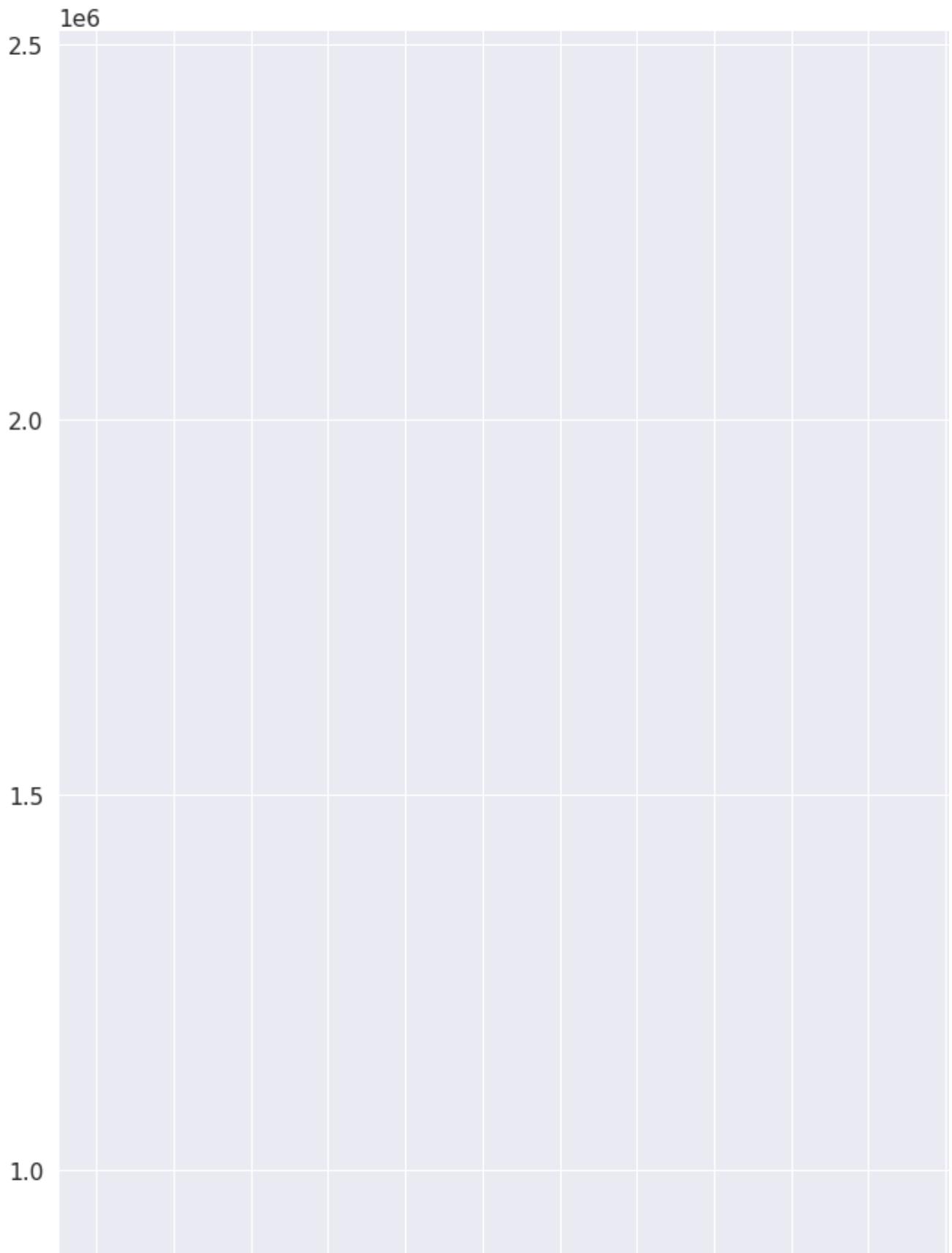
```
df.boxplot(by = 'mark', column =['price'], figsize=(30,10), grid = True)
```

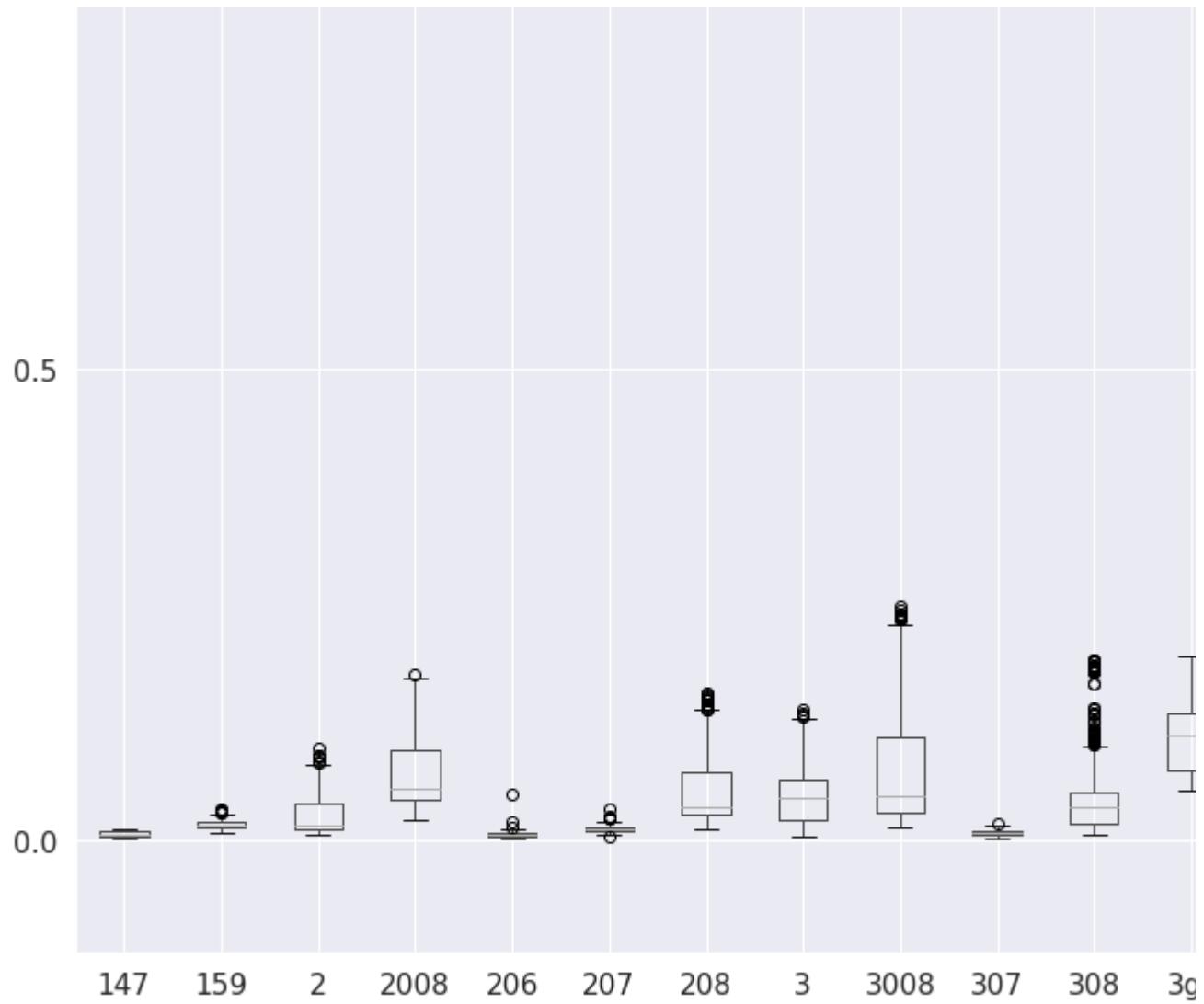
```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning:  
    X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))  
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f562f7110>
```



```
df.boxplot(by = 'model', column =['price'], figsize=(300,25), grid = True)
```

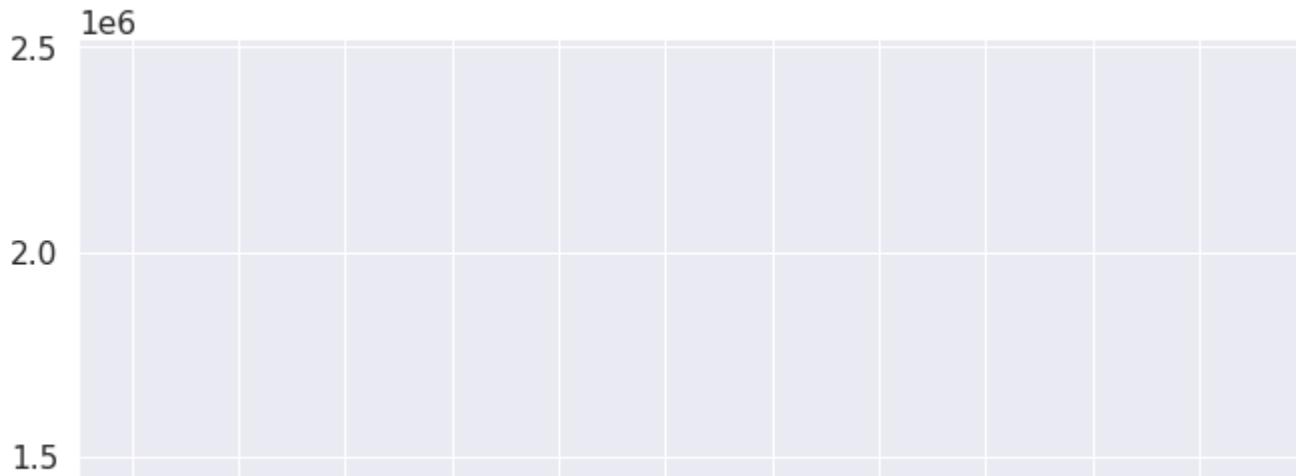
```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))  
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f5600a310>
```





```
df.boxplot(by ='year', column =['price'], figsize=(50,10), grid = True)
```

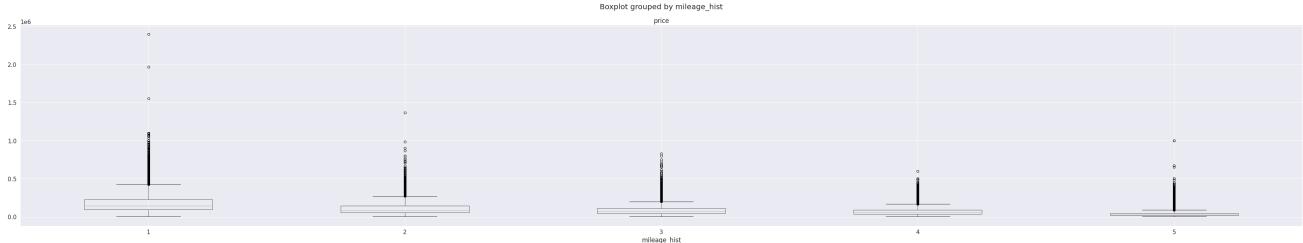
```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))  
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f5242f590>
```



#Tempo de Execução Muito Longo

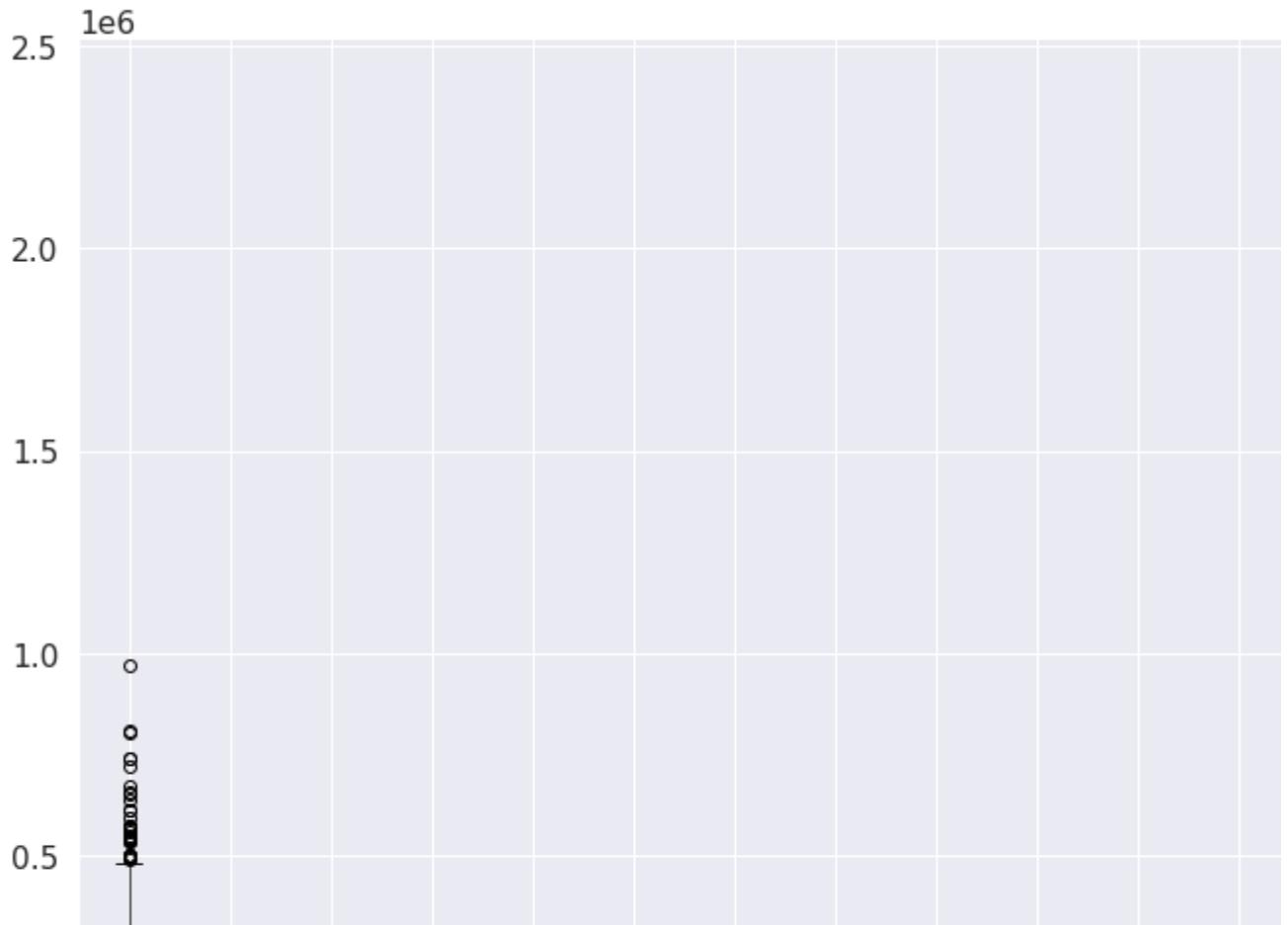
```
df.boxplot(by = 'mileage_hist', column =['price'], figsize=(60,10), grid = True)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))  
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f52390110>
```



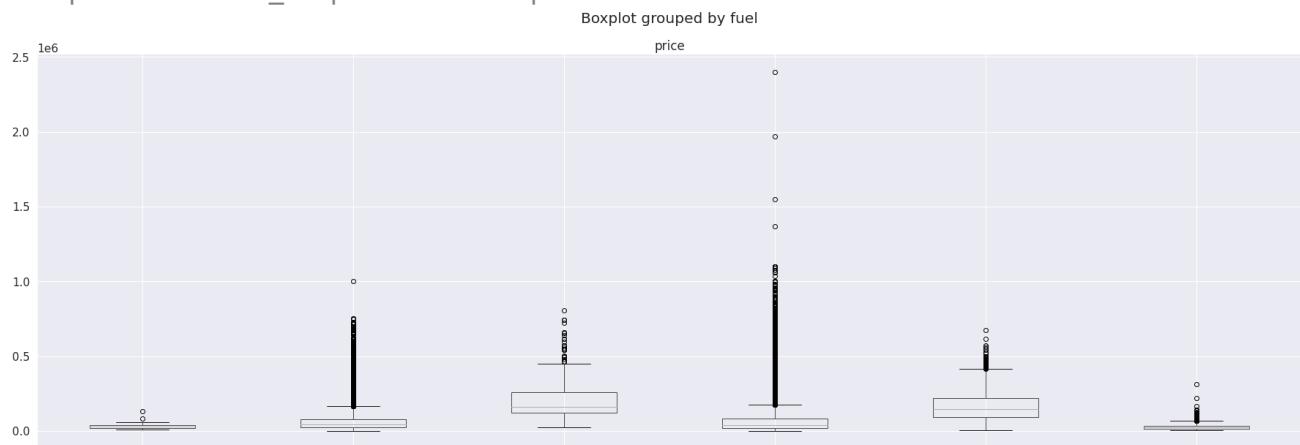
```
df.boxplot(by ='vol_engine', column =['price'], figsize=(450,10), grid = True)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))  
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f55fa9410>
```



```
df.boxplot(by ='fuel', column =['price'], figsize=(30,10), grid = True)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))  
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f51b466d0>
```

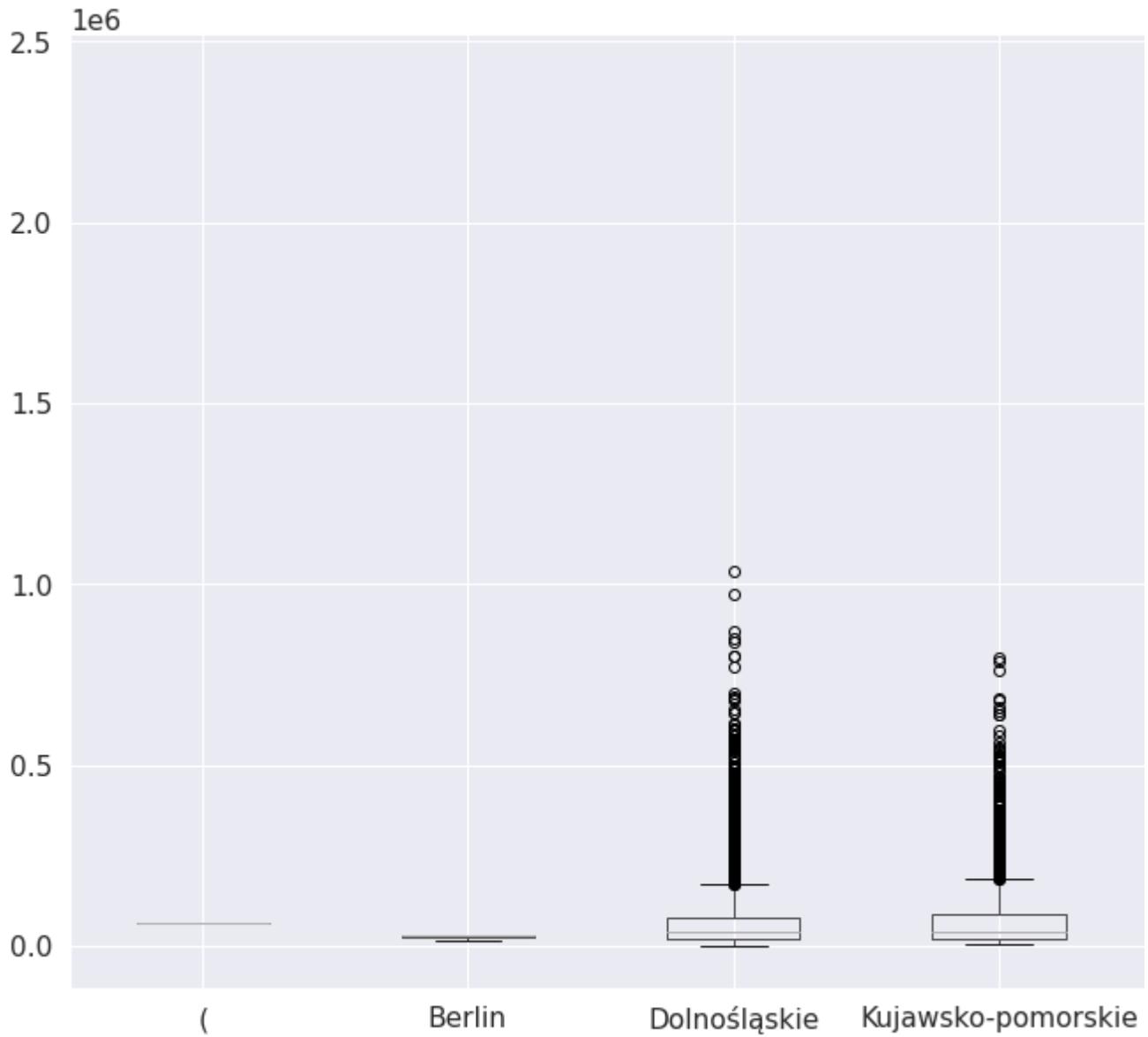


```
df.boxplot(by ='city', column =['price'], figsize=(300,10), grid = True)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDepр
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))

df.boxplot(by ='province', column =['price'], figsize=(60,10), grid = True)

/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDepр
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f4dd3e150>
```



LEVANTAMENTO DE DADOS RELACIONADOS A VARIABEL DE ACRECIMO DO VALOR DO VEICULO

1. Pesquisa de conhecimento, utilizando web para identificar o conhecimento relacionado ao impacto sobre valor do carro.
2. Graficos corelacionados ao qual indicam acrescimo do valor sob veiculo.

- Analise de informação sobre desvalorização

Os principais fatores que afetam a desvalorização dos ativos adquiridos pela população são, os modelos, ano, assim como a granularidade da recepção de um local em relação a veículos semi-novos ou usados. Como também, a quilometragem, acessórios do carro e estado de conservação.

1. Quilometragem
2. Versões e equipamentos
3. Ausência de equipamentos
4. Estado geral de conservação

- Analise de informação sobre valorização

<https://g1.globo.com/economia/noticia/2021/11/16/por-que-os-precos-dos-carros-dispararam-no-mundo.ghtml>

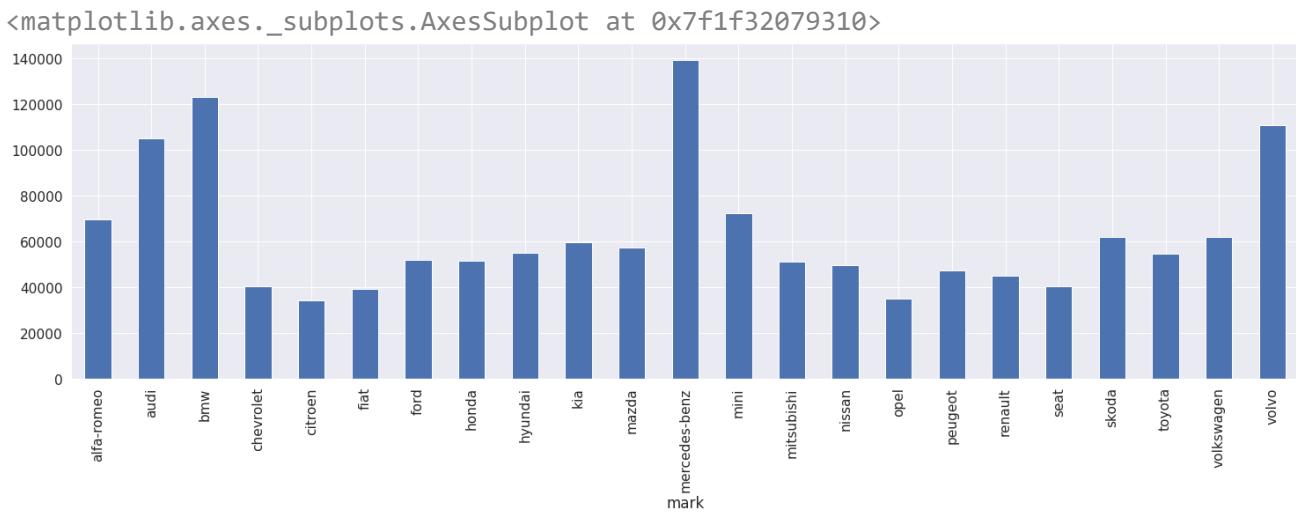
<https://www.minutoseguros.com.br/blog/alta-de-preco-de-veiculos-entenda-como-isso-impacta-no-valor-do-seguro/>

Segundo o modelo de percepção dos fabricantes os motivos dos custos advêm da produção dos ativos, aplicando precificação adicional pela marca no mercado, assim como a receptividade do mercado em relação ao veículo. Outros fatores também seguem;

1. ICMS
2. Falta de materiais
3. Alta demanda
4. Paralisação de fábricas
5. Preferência dos usuários por cores de carros

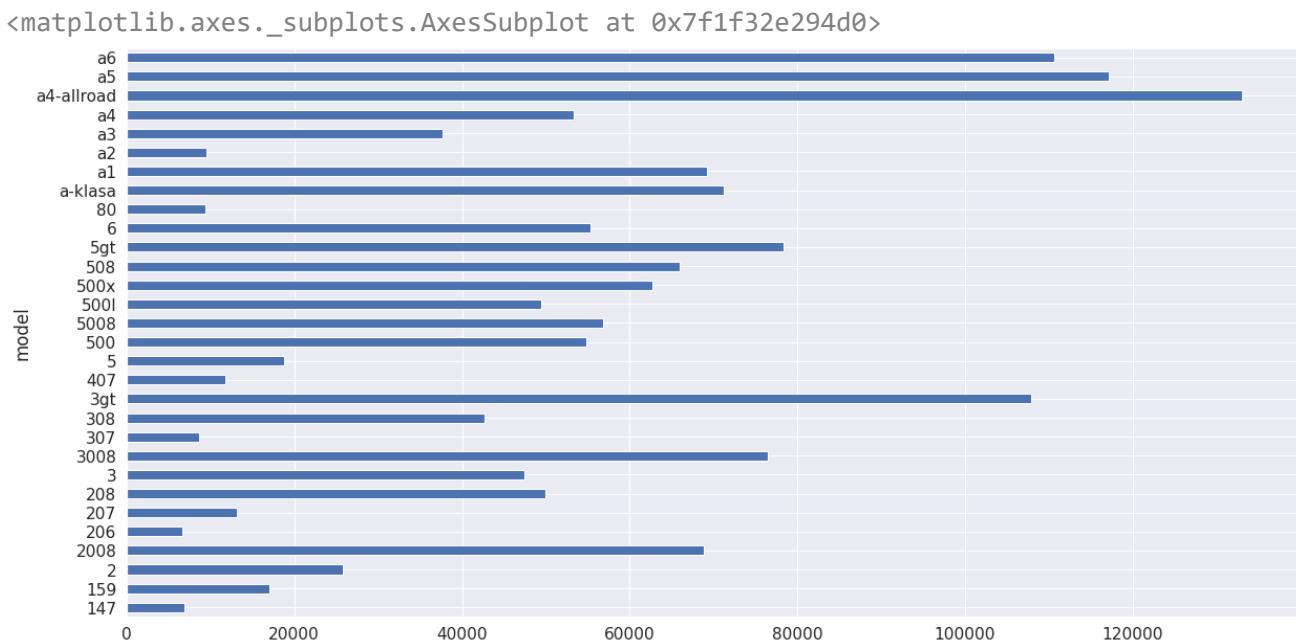
```
Mark = df.groupby("mark")["price"].mean()
```

```
Mark.plot(kind= "bar", figsize=(25,7))
```



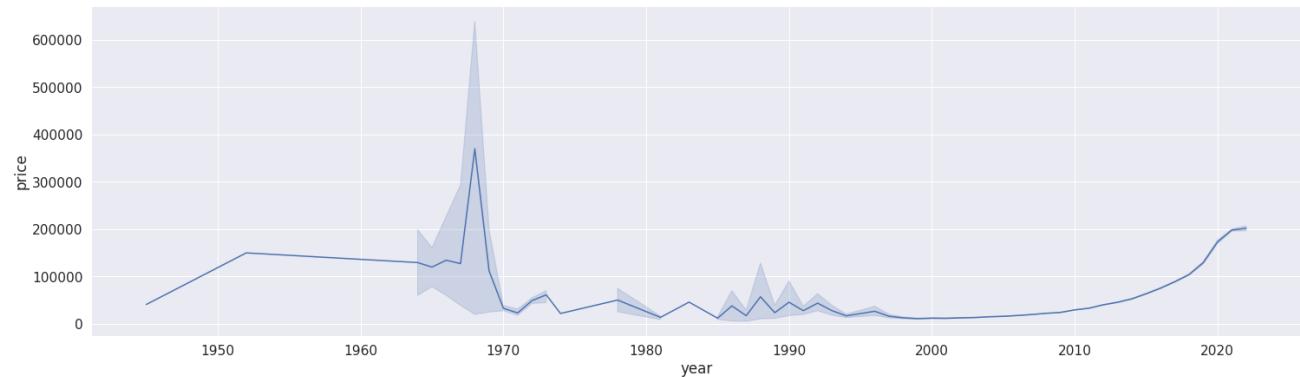
```
Model = df.groupby("model")["price"].mean()
```

```
Model.head(30).plot(kind='barh', figsize=(20,10))
```



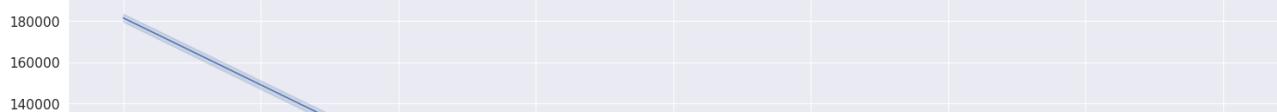
```
#Acrescimos dos valores por Ano  
plt.figure(figsize=(25,7))  
sns.lineplot(data=df,x="year",y="price")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f3318fc50>
```



```
#Acrescimos dos valores por mileage  
plt.figure(figsize=(25,7))  
sns.lineplot(data=df,x="mileage_hist",y="price")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f32fb5bd0>
```

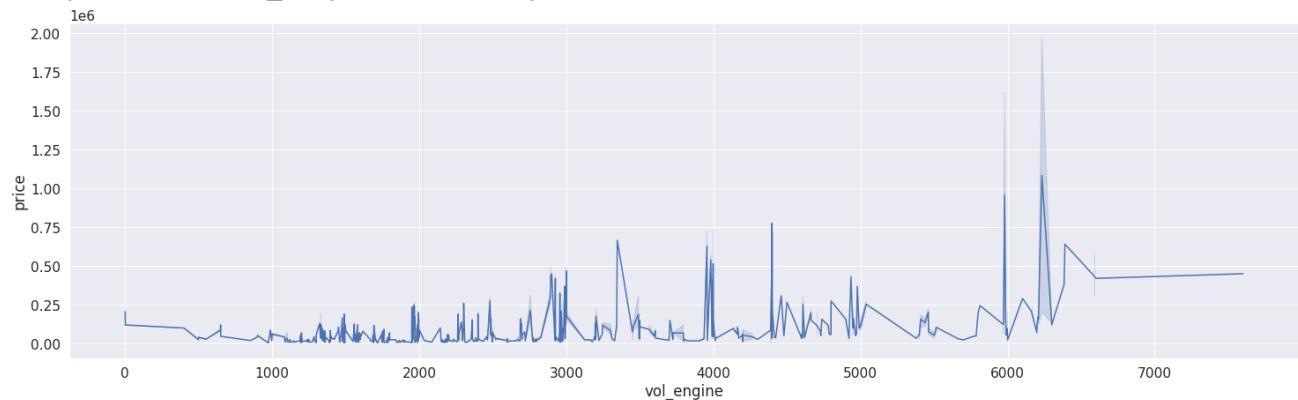


```
#Acrescimos dos valores por Engine
```

```
plt.figure(figsize=(25,7))
```

```
sns.lineplot(data=df,x="vol_engine",y="price")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f32122990>
```



```
#Acrescimos dos valores por Engine
```

```
plt.figure(figsize=(25,7))
```

```
sns.lineplot(data=df,x="fuel",y="price")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f329be050>
```

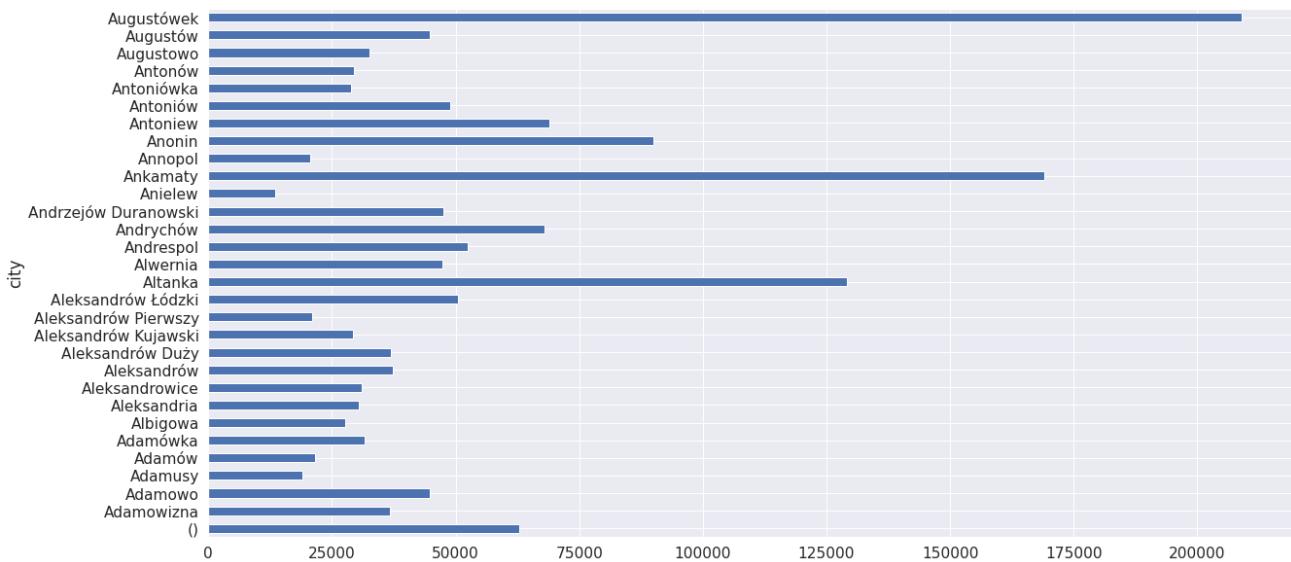


```
#Acrescimos dos valores por city
```

```
City = df.groupby("city")["price"].mean()
```

```
City.head(30).plot(kind='barh', figsize=(20,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f32b6cf0d0>
```

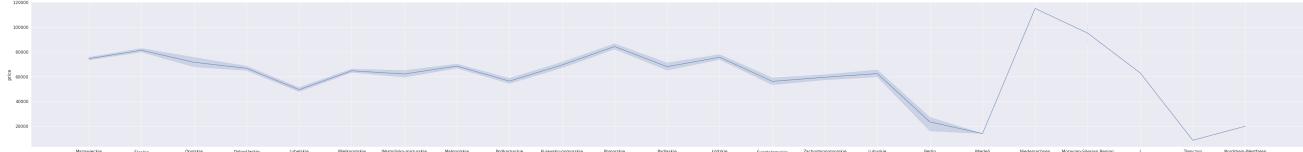


```
#Acrescimos dos valores por Engine
```

```
plt.figure(figsize=(85,10))
```

```
sns.lineplot(data=df,x="province",y="price")
```

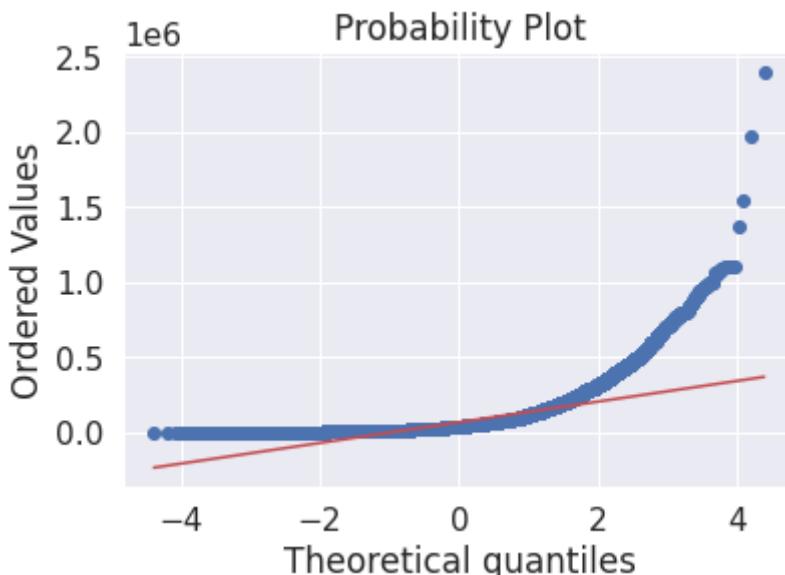
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f331a0a90>
```



ANALISE DA DISTRIBUIÇÃO NORMAL (GAUSSIANA)

```
import scipy.stats as stats
import pylab
stats.probplot(df['price'], dist="norm", plot=pylab)
pylab.show()
```

#Um gráfico popular para verificar a distribuição de uma amostra de dados é o gráfico quan
#Uma combinação perfeita para a distribuição será mostrada por uma linha de pontos em um â
#Muitas vezes, uma linha é desenhada no enredo para ajudar a tornar essa expectativa clara



```
df.head(0)
```

mark	model	year	mileage	vol_engine	fuel	city	province	price	mileage_hist
------	-------	------	---------	------------	------	------	----------	-------	--------------

```
from scipy.stats import shapiro
stat, p = shapiro(df['price'])
print('Statistics=% .3f, p=% .3f' % (stat, p))
```

```
Statistics=0.657, p=0.000
/usr/local/lib/python3.7/dist-packages/scipy/stats/morestats.py:1676: UserWarning: p-
warnings.warn("p-value may not be accurate for N > 5000.")
```

```
p, stat
```

```
(0.0, 0.6574497818946838)
```

```
alpha = 0.05
if p > alpha:
    print('A amostra parece gaussiana (falha ao rejeitar H0)')
else:
    print('A amostra não parece gaussiana (rejeite H0)')
```

```
A amostra não parece gaussiana (rejeite H0)
```

```
from scipy.stats import normaltest
stat, p = normaltest(df['price'])
print(p, stat)
```

```
0.0 95241.60615536486
```

```
alpha = 0.05
if p > alpha:
    print('A amostra parece gaussiana (falha ao rejeitar H0)')
else:
    print('A amostra não parece gaussiana (rejeite H0)')
```

```
A amostra não parece gaussiana (rejeite H0)
```

ANALISE DO IMPACTO DOS OUTLIERS E O PREÇO

```
def outliers_detector(df):
    q1=df.quantile(0.25)
    q3=df.quantile(0.75)
    variação_interquartil = q3 - q1
    outliers = df[((df<(q1-1.5*variação_interquartil)) | (df>(q3+1.5*variação_interquartil))
    return outliers
```

```
outliers = outliers_detector(df['price'])
outliers
```

```
5032      999999
5825     181900
5834     179900
5859     197000
5868     199990
...
117917    177798
117919    351306
117921    215250
117922    222790
117923    229900
Name: price, Length: 10060, dtype: int64
```

```
print("Valor Maximo: ", outliers.max())
print("Valor Médio: ", (outliers.sum()/len(outliers)))
```

```
print("Valor Mínimo: ", outliers.min())
totalOut = len(outliers)
print("Total de Outliers: ", totalOut)
```

```
Valor Maximo: 2399900
Valor Médio: 288717.9290258449
Valor Mínimo: 177599
Total de Outliers: 10060
```

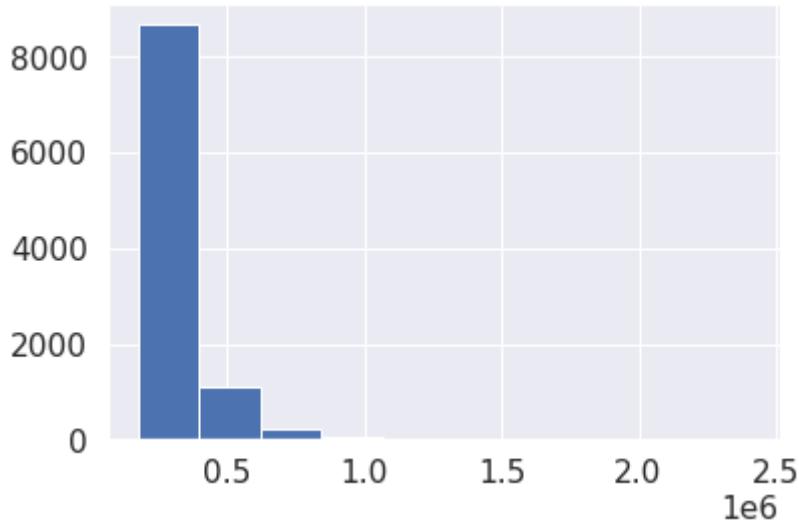
```
baseTotal = df.count().sum()
print('Total da base de dados: ', baseTotal)
```

```
Total da base de dados: 1296244
```

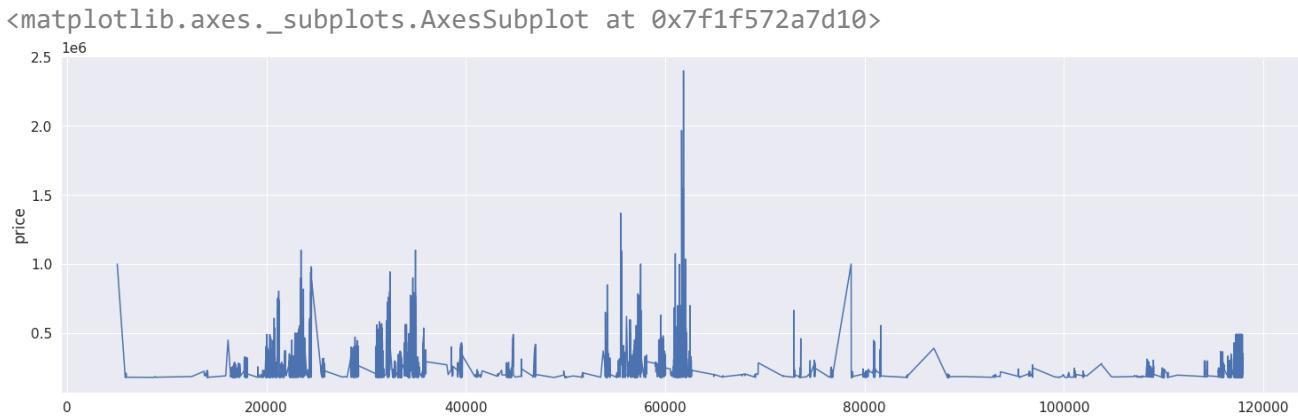
```
#Porcentagem de relação do total de outliers e tamanho do Dataset
print('Outliers representam: ' + str(totalOut/baseTotal) + '% da base de dados')
```

```
Outliers representam: 0.007760884524827116% da base de dados
```

```
outliers.hist();
```



```
#Distribuição dos outliers
plt.figure(figsize=(25,7))
sns.lineplot(data=outliers)
```



IDENTIFICAÇÃO DOS VALORES E TIPOS MAIS COMUNS DE DADOS

```
c.most_common(5)
print ("Marca: ",c.most_common(5))

c = Counter(df['model'])
c.most_common(5)
print ("Modelo: ",c.most_common(5))

c = Counter(df['year'])
c.most_common(5)
print ("Ano: ",c.most_common(5))

c = Counter(df['vol_engine'])
c.most_common(5)
print ("Motor: ",c.most_common(5))

c = Counter(df['fuel'])
c.most_common(5)
print ("Combustivel: ",c.most_common(5))

c = Counter(df['city'])
c.most_common(5)
print ("Cidade: ",c.most_common(5))

c = Counter(df['province'])
c.most_common(5)
print ("Estado: ",c.most_common(5))

c = Counter(df['price'])
c.most_common(5)
print ("Preço $: ",c.most_common(5))
```

```
Marca: [(19900, 1336), (39900, 1154), (29900, 1139), (18900, 1100), (14900, 1010)]
Modelo: [('astra', 3331), ('seria-3', 2944), ('a4', 2912), ('golf', 2592), ('a6', 2492)]
Ano: [(2021, 10559), (2017, 8909), (2018, 8647), (2016, 7021), (2009, 6828)]
Motor: [(1598, 10206), (1968, 8121), (1995, 6545), (1997, 5340), (1998, 4498)]
Combustivel: [('Gasoline', 61597), ('Diesel', 48476), ('LPG', 4301), ('Hybrid', 2621)]
Cidade: [('Warszawa', 7972), ('Łódź', 3341), ('Kraków', 2936), ('Wrocław', 2764), ('Gdańsk', 2492)]
Estado: [('Mazowieckie', 22219), ('Śląskie', 16706), ('Wielkopolskie', 14016), ('Małopolskie', 13361)]
Preço $: [(19900, 1336), (39900, 1154), (29900, 1139), (18900, 1100), (14900, 1010)]
```

ANALISE DA CORRELAÇÃO

```
df.corr()
```

	year	mileage	vol_engine	price
year	1.000000	-0.731958	-0.161557	0.596181
mileage	-0.731958	1.000000	0.206169	-0.542808
vol_engine	-0.161557	0.206169	1.000000	0.299669
price	0.596181	-0.542808	0.299669	1.000000

```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	year	mileage	vol_engine	price
year	1.000000	-0.731958	-0.161557	0.596181
mileage	-0.731958	1.000000	0.206169	-0.542808
vol_engine	-0.161557	0.206169	1.000000	0.299669
price	0.596181	-0.542808	0.299669	1.000000

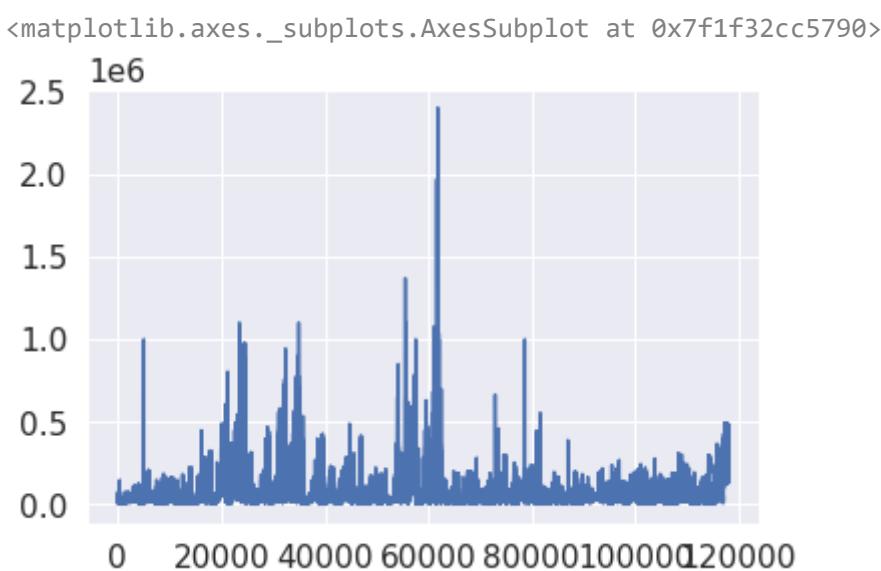
SETANDO UMA BASE DE DADOS SEM OUTLIERS

Testando metodos,

- Metodo I: IQR (Intervalo Interquartile)
- Metodo II: Metodo Z-Score

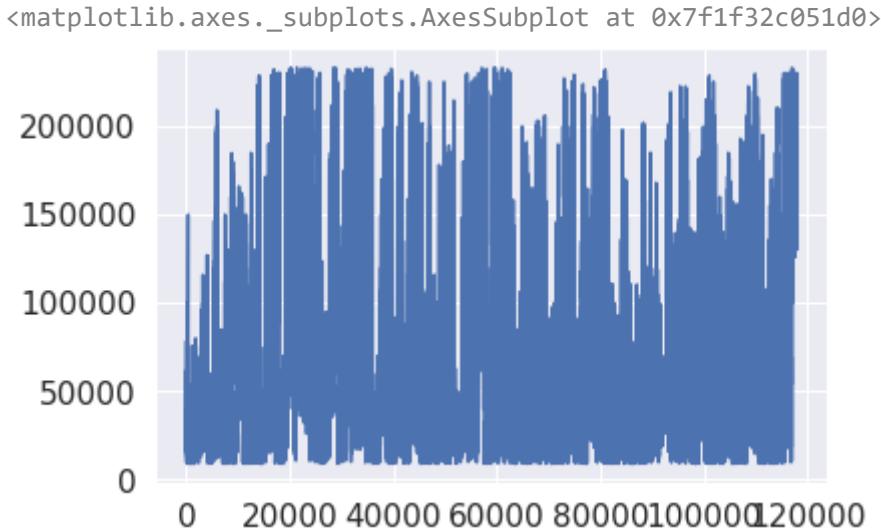
```
#Outliers representam: 0.0095% da base de dados. Decidido por removelos para melhor normal
#Plot da base atual
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f56a54e90>
1e6
df['price'].plot()
```



```
#Plot do modelo sem outliers
ndf = df['price']
dfNout = ndf.between(ndf.quantile(.05), ndf.quantile(.95))

ndf[dfNout].plot()
```



```
#Total de registros remanescentes
ndf.count()
```

117927

▼ Metodos I: IQR (Intervalo Interquartile)

```
def outliers_IQR(df, feature):
```

```

Q1 = df[feature].quantile(0.25)
Q3 = df[feature].quantile(0.75)
IQR = Q3 - Q1
limite_superior = Q3 + 1.5 * IQR
limite_inferior = Q1 - 1.5 * IQR
return limite_superior, limite_inferior

IQRsuperior, IQRinferior = outliers_IQR(df, "price")
print("Intervalo Superior: ", IQRsuperior)
print("Intervalo Inferior: ", IQRinferior)

Intervalo Superior:  177500.0
Intervalo Inferior: -72900.0

```

```
df[(df['price'] < IQRinferior) | (df['price'] > IQRsuperior)]
```

	mark	model	year	mileage	vol_engine	fuel	city	province
5032	opel	corsa	2013	105703	1398	Gasoline	Wilchta	Mazowieckie
5825	opel	grandland-x	2020	568	1598	Hybrid	Swarzędz	Wielkopolskie
5834	opel	grandland-x	2021	4900	1598	Hybrid	Bydgoszcz	Kujawsko-pomorskie
5859	opel	grandland-x	2020	499	1598	Hybrid	Olsztyn	Warmińsko-mazurskie
5868	opel	grandland-x	2021	5	1598	Hybrid	Wrocław	Dolnośląskie
...
117917	volvo	xc-90	2016	131000	1969	Diesel	Lublin	Lubelskie
117919	volvo	xc-90	2021	26700	1969	Hybrid	Gliwice	Śląskie
117921	volvo	xc-90	2017	103980	1969	Diesel	Drezdenko	Lubuskie
117922	volvo	xc-90	2020	40000	1969	Hybrid	Katowice	Śląskie

```
IQR_df = df[(df['price'] > IQRinferior) & (df['price'] < IQRsuperior)]
```

```

print("Antigo DataFrame: ", df.count().sum())
print("Novo DataFrame: ", IQR_df.count().sum())

```

```

Antigo DataFrame: 1296244
Novo DataFrame: 1186370

```

```
len(IQR_df)
```

```
107863
```

```
IQR_dfToPlot = IQR_df.copy()
```

▼ Método II: Z-Score

```
def outlier_removal(df, feature):
    limite_superior = df[feature].mean() + 3 * df[feature].std()
    limite_inferior = df[feature].mean() - 3 * df[feature].std()
    return limite_superior, limite_inferior
```

```
zsuperior, zinferior = outlier_removal(df, "price")
print("Intervalo Superior Z-Score: ", zsuperior)
print("Intervalo Inferior Z-Score: ", zinferior)
```

```
Intervalo Superior Z-Score:  324773.6181673118
Intervalo Inferior Z-Score: -184173.84858104237
```

```
df[(df['price'] < zinferior) | (df['price'] > zsuperior)]
```

	mark	model	year	mileage	vol_engine	fuel	city	province	price
5032	opel	corsa	2013	105703	1398	Gasoline	Wilchta	Mazowieckie	99999
16148	audi	a4	2012	213400	1968	Diesel	Zwoleń	Mazowieckie	44900
17821	audi	a5	2021	10	1968	Diesel	Wrocław	Dolnośląskie	32728
19923	audi	a6	2021	1	1968	Diesel	Chorzów	Śląskie	33650
19935	audi	a6	2021	1	2967	Diesel	Kraków	Małopolskie	40220
...
117895	volvo	xc-90	2020	37946	1969	Hybrid	Poznań	Wielkopolskie	32990
117898	volvo	xc-90	2021	1	1969	Hybrid	Kraków	Małopolskie	35523
117905	volvo	xc-90	2021	10	1969	Hybrid	Płock	Mazowieckie	48950
117908	volvo	xc-90	2020	25600	1969	Diesel	Olsztyn	Warmińsko-mazurskie	32900
117919	volvo	xc-90	2021	26700	1969	Hybrid	Gliwice	Śląskie	35130

```
z_df = df[(df['price'] > zinferior) & (df['price'] < zsuperior)]
```

```
print("Antigo DataFrame: ", df.count().sum())
print("Novo DataFrame: ", z_df.count().sum())
```

```
Antigo DataFrame: 1296244
Novo DataFrame: 1269000
```

```
print("Antigo DataFrame: ", len(df))
print("Novo DataFrame: ", len(z_df))
```

```
Antigo DataFrame: 117927  
Novo DataFrame: 115390
```

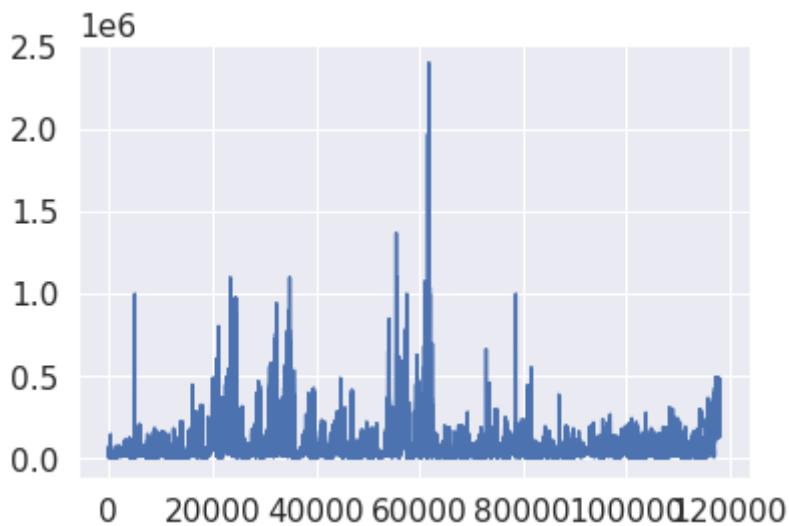
▼ Comparativo tratamentos Método I e Método II

```
#RO = Remoção de Outliers  
#Comparativo da analise de remoção dos conjuntos  
print("Antigo DataFrame: ", df.count().sum())  
print("RO IQR Interquartile removidos: ", df.count().sum() - IQR_df.count().sum())  
print("RO Metodo Z-Score removidos: ", df.count().sum() - z_df.count().sum())
```

```
Antigo DataFrame: 1296244  
RO IQR Interquartile removidos: 109874  
RO Metodo Z-Score removidos: 27244
```

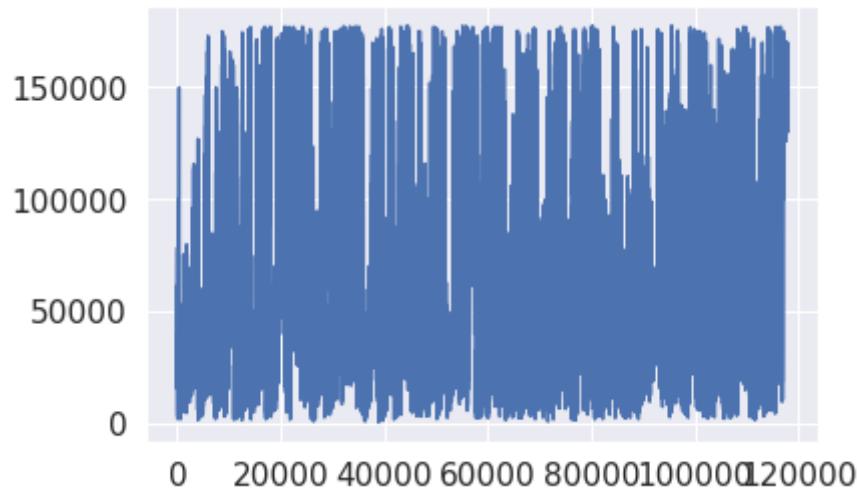
```
df['price'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f32b34e50>
```

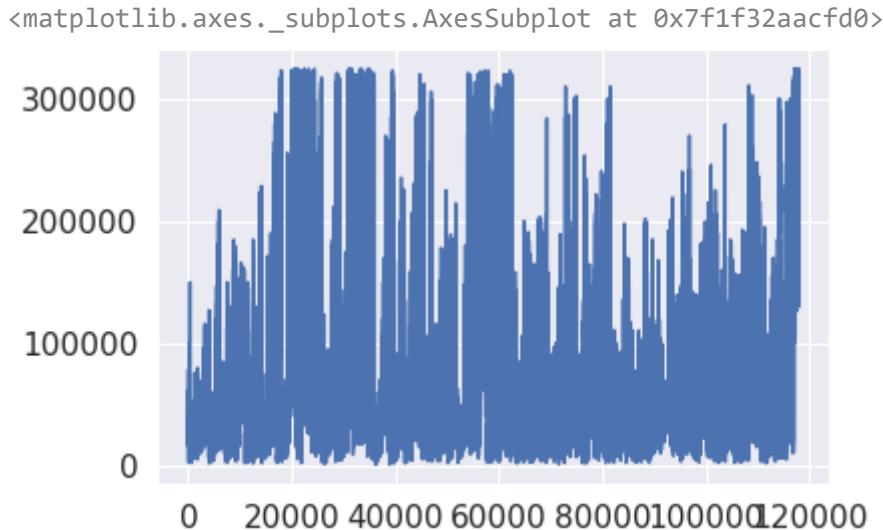


```
IQR_df['price'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f32aac550>
```



```
z_df['price'].plot()
```



PLOT DOS 3 TIPOS DE MODELO SEM TRATAMENTO E TRATAMENTO ZSCORE E IQR

```
dfnormal.corr()
```

	year	mileage	vol_engine	price
year	1.000000	-0.731958	-0.161557	0.596181
mileage	-0.731958	1.000000	0.206169	-0.542808
vol_engine	-0.161557	0.206169	1.000000	0.299669
price	0.596181	-0.542808	0.299669	1.000000

```
IQR_dftoPlot.corr()
```

	year	mileage	vol_engine	price
year	1.000000	-0.690955	-0.286711	0.737301
mileage	-0.690955	1.000000	0.351611	-0.591232
vol_engine	-0.286711	0.351611	1.000000	0.100293
price	0.737301	-0.591232	0.100293	1.000000

```
z_df.corr()
```

	year	mileage	vol_engine	price
year	1.000000	-0.722741	-0.219910	0.684483

▼ 3.0 CONCLUSÃO DA ANALISE

...
...
...

INDICATIVOS DO DATASET

- Base de dados apresentada representa dados de 1945 a 2022
- Dentro do Dataset possuem 23 Marcas diferentes, 328 modelos, 508 motores, 6 tipos de combustivel, 4427 cidades e 9000 preços para veiculos.
- Os dados em sua pluralidade apresentam boa distribuição heterogêna.
- Dados apresentam conjuntos significativos de outliers apos determinadas datas, indicando principalmente virada do seculo 21.
- Outliers representam: 0.0095% da base de dados.
- Tamanho do Dataset: 1.061.343

LIMPEZA DA BASE DE DADOS REMOVENDO VALORES NULOS E ANALISE DE REMOÇÃO DE OUTLIERS

- generation_name, 30085 NULOS.
- Unnamed, removida coluna de numeração.
- Antigo DataFrame: 1061343
- RO IQR Interquartile removidos: 90576
- RO Metodo Z-Score removidos: 22833

INDICATIVOS ANALISE DE DADOS

- MARCA: Audi, Opel, BMW, Volkswagen, Ford
- MODELO: Astra, Seria-3, A4, Golf , A6
- ANO: 2021, 2017, 2018, 2016, 2009
- MOTOR:1598, 1968, 1995, 1997, 1998
- COMBUSTIVEL: Gasolina, Diesel, LPG, Hybrido, Eletrico
- CIDADES: Warszawa, Łódź, Kraków, Wrocław, Poznań
- ESTADO: Mazowieckie, Śląskie, Wielkopolskie, Małopolskie
- PREÇO: \$\$19900, \$39900, \$29900, \$18900, \$14900
- PREÇOS MAIS ALTOS: Mercedes, BMW, Audi, Volvo, Alfa-Romeo

INDICATIVOS DA CORRELAÇÃO

- PREÇO: POR KILOMETRAGEM MENOR
- PREÇO: POR MELHOR MOTOR
- PREÇO: POR MARCA MAIOR

Decisão de uso do Dataset, temos:

- df: Aceitação dos outliers como um modelo de comportamento padrão.
- IQR_df: devido a método ser muito sensível a valores extremos, removendo valores que não gostariam que fossem retirados do universo.
- z_df: remoção de outliers univariados ou seja, valores extremos na distribuição de uma variável específica.

▼ 4. TEOREMA DOS MODELOS

Modelos a seguir presupostos para uso:

- Regressao linear, KNN, Perceptron Simples, ou modelo-ligado a regressao)
- Linear Regression, Ridge Regression, Neural Network Regression, Lasso Regression, Decision Tree Regression, Random Forest, KNN Model.
- Será aplicado uma escala de desenvolvimento do modelo para melhor analise e mapeamento do desenvolvimento do melhor algoritimo.

Tipo de analise escolhida: O resultado da regressão linear é sempre um número. É utilizada adequadamente quando o dataset apresenta algum tipo de tendência de crescimento/descrescimento constante. A análise de regressão pode ser utilizada para resolver os seguintes tipos de problemas: Determinar quais variáveis explanatórias estão relacionadas à variável dependente. Entender o relacionamento entre as variáveis dependentes e explanatórias. Prever valores desconhecidos da variável dependente.

TRATAMENTO DA BASE DE DADOS PERMITINDO ▼ OUTLIERS

```
from sklearn.preprocessing import LabelEncoder
dfNormal = df

dfNormal
```

	mark	model	year	mileage	vol_engine	fuel	city	province	price
0	opel	combo	2015	139568	1248	Diesel	Janki	Mazowieckie	35900
1	opel	combo	2018	31991	1499	Diesel	Katowice	Śląskie	78501
2	opel	combo	2015	278437	1598	Diesel	Brzeg	Opolskie	27000
3	opel	combo	2016	47600	1248	Diesel	Korfantów	Opolskie	30800
4	opel	combo	2014	103000	1400	CNG	Tarnowskie Góry	Śląskie	35900
...
117922	volvo	xc-90	2020	40000	1969	Hybrid	Katowice	Śląskie	222790
117923	volvo	xc-90	2017	51000	1969	Diesel	Czechło Pierwsze	Łódzkie	229900
117924	volvo	xc-90	2016	83500	1969	Gasoline	Pruszcz Gdańsk	Pomorskie	135000

```
dfNormal.drop(columns=["mileage_hist","price_hist"],axis=1,inplace=True)
dfNormal.drop(columns=["city","province"],axis=1,inplace=True)
```

dfNormal

	mark	model	year	mileage	vol_engine	fuel	price
0	opel	combo	2015	139568	1248	Diesel	35900
1	opel	combo	2018	31991	1499	Diesel	78501
2	opel	combo	2015	278437	1598	Diesel	27000
3	opel	combo	2016	47600	1248	Diesel	30800
4	opel	combo	2014	103000	1400	CNG	35900
...
117922	volvo	xc-90	2020	40000	1969	Hybrid	222790
117923	volvo	xc-90	2017	51000	1969	Diesel	229900
117924	volvo	xc-90	2016	83500	1969	Gasoline	135000
117925	volvo	xc-90	2017	174000	1969	Diesel	154500
117926	volvo	xc-90	2016	189020	1969	Gasoline	130000

117927 rows × 7 columns

#Convertendo as variaveis categoricas

```
LE = LabelEncoder()
LE.fit(dfNormal["mark"])
dfNormal["Mark"] = LE.transform(dfNormal["mark"])
```

#Convertendo as variaveis categoricas

```

LE2 = LabelEncoder()
LE2.fit(dfNormal["model"])
dfNormal["Model"] = LE2.transform(dfNormal["model"])

#Convertendo as variaveis categoricas
LE3 = LabelEncoder()
LE3.fit(dfNormal["fuel"])
dfNormal["Fuel"] = LE3.transform(dfNormal["fuel"])

dfNormal.drop(columns=["mark", "model", "fuel"], axis=1, inplace=True) #dropping

dfNormal.head(1)

```

	year	mileage	vol_engine	price	Mark	Model	Fuel
0	2015	139568	1248	35900	15	89	1

```
len(dfNormal)
```

```
117927
```

▼ 5. APLICAÇÃO DOS MODELOS SEM OUTLIERS

```

import warnings
warnings.filterwarnings("ignore")

```

SEPARAÇÃO DOS DADOS DE TREINO E TESTE E PREPARAÇÃO DOS DADOS

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from yellowbrick.datasets import load_concrete
from yellowbrick.regressor import ResidualsPlot
from sklearn.linear_model import Ridge

```

```

#Treinamento
from sklearn.model_selection import train_test_split
#O coeficiente de determinação também chamado de pontuação R2 é usado para avaliar o desem
from sklearn.metrics import r2_score
from sklearn import linear_model

```

```
z_df.head(1)
```

	mark	model	year	mileage	vol_engine	fuel	city	province	price	mileage_l
0	opel	combo	2015	139568	1248	Diesel	Janki	Mazowieckie	35900	

```
#Drop para organização e ordenação
z_df.drop(columns=["mileage_hist","price_hist"],axis=1 ,inplace=True)
z_df.drop(columns=["city","province"],axis=1 ,inplace=True)
```

```
DTCdf = z_df
```

```
DTCdf.head(0)
```

mark	model	year	mileage	vol_engine	fuel	price
------	-------	------	---------	------------	------	-------

```
#Convertendo as variaveis categoricas
```

```
LE = LabelEncoder()
LE.fit(DTCdf["mark"])
DTCdf["Mark"] = LE.transform(DTCdf["mark"])
```

```
#Convertendo as variaveis categoricas
```

```
LE2 = LabelEncoder()
LE2.fit(DTCdf["model"])
DTCdf["Model"] = LE2.transform(DTCdf["model"])
```

```
#Convertendo as variaveis categoricas
```

```
LE3 = LabelEncoder()
LE3.fit(DTCdf["fuel"])
DTCdf["Fuel"] = LE3.transform(DTCdf["fuel"])
```

```
DTCdf.head(1)
```

	mark	model	year	mileage	vol_engine	fuel	price	Mark	Model	Fuel	
0	opel	combo	2015	139568		1248	Diesel	35900	15	89	1

```
DTCdf.drop(columns=["mark","model","fuel"],axis=1 ,inplace=True )#dropping
```

```
DTCdf.head(10)
```

	year	mileage	vol_engine	price	Mark	Model	Fuel
0	2015	139568	1248	35900	15	89	1
1	2018	31991	1499	78501	15	89	1
2	2015	278437	1598	27000	15	89	1

```
len(DTCdf)
```

```
115390
5 2017 121203 1598 51900 15 89 1
```

```
#DEFINIÇÃO DOS DADOS DE TREINO E TESTE
```

```
X = DTCdf.drop(columns="price")
```

```
y = DTCdf["price"]
```

```
#SEPARAÇÃO DOS DADOS DE TREINO E TESTE
```

```
#Se você não especificar o random_state no código, toda vez que você executar (executar) s
```

```
#No entanto, se um valor fixo for atribuído como random_state = 0 ou 1 ou 42 ou qualquer o
```

```
#Utilizando valor determinístico para servir de base de comparações
```

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

```
#Intanciando as variáveis para análise de resíduos
```

```
model = Ridge()
```

```
visualizer = ResidualsPlot(model)
```

```
print("X TREINO : ", X_train.shape)
```

```
print("X TESTE : ", X_test.shape)
```

```
print("Y TREINO : ", y_train.shape)
```

```
print("Y TESTE : ", y_test.shape)
```

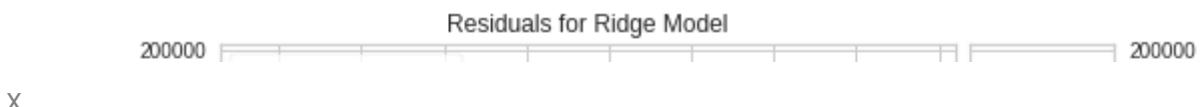
```
visualizer.fit(X_train, y_train) # Fit the training data to the visualizer
```

```
visualizer.score(X_test, y_test) # Evaluate the model on the test data
```

```
visualizer.show() # Finalize and render the figure
```

```
#https://www.scikit-yb.org/en/latest/api/regressor/residuals.html
```

```
X TREINO : (80773, 6)
X TESTE  : (34617, 6)
Y TREINO : (80773,)
Y TESTE  : (34617,)
```



	year	mileage	vol_engine	Mark	Model	Fuel
0	2015	139568	1248	15	89	1
1	2018	31991	1499	15	89	1
2	2015	278437	1598	15	89	1
3	2016	47600	1248	15	89	1
4	2014	103000	1400	15	89	0
...
117922	2020	40000	1969	22	320	4
117923	2017	51000	1969	22	320	1
117924	2016	83500	1969	22	320	3
117925	2017	174000	1969	22	320	1
117926	2016	189020	1969	22	320	3

115390 rows × 6 columns

```
#DEFINIÇÃO DOS DADOS DE TREINO E TESTE SEM TRATAMENTO #NORMAL Q-Q RESIDUAL PLOT e Residual
X2 = dfNormal.drop(columns="price")
y2 = dfNormal["price"]

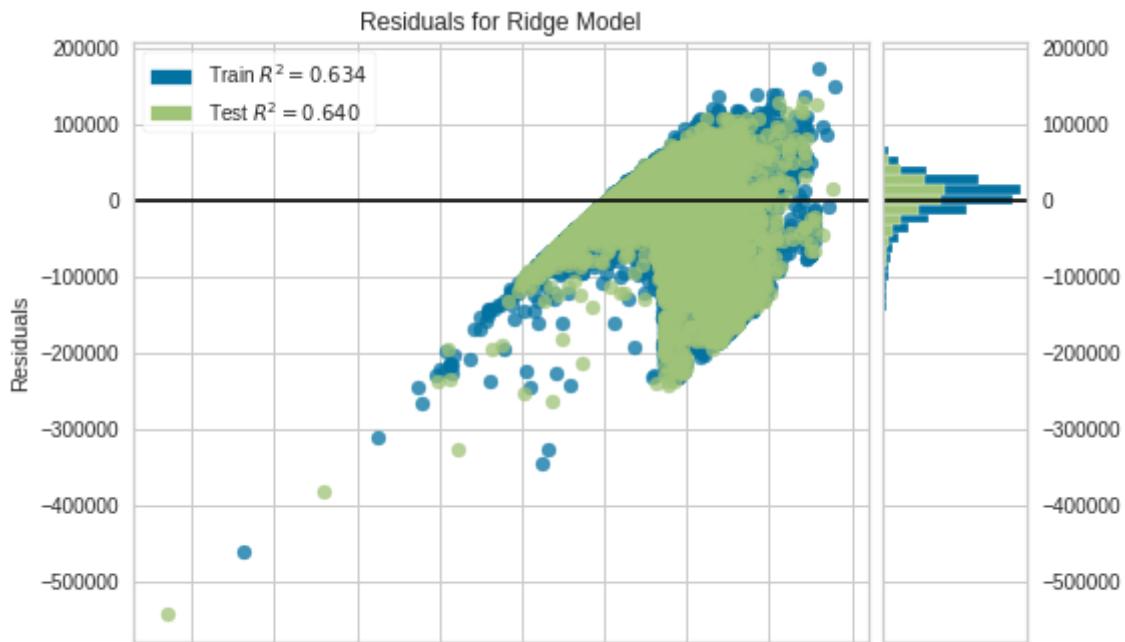
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size = 0.3, random_st

#Intanciando as variaveis para analise de resíduos
model = Ridge()
visualizer = ResidualsPlot(model)

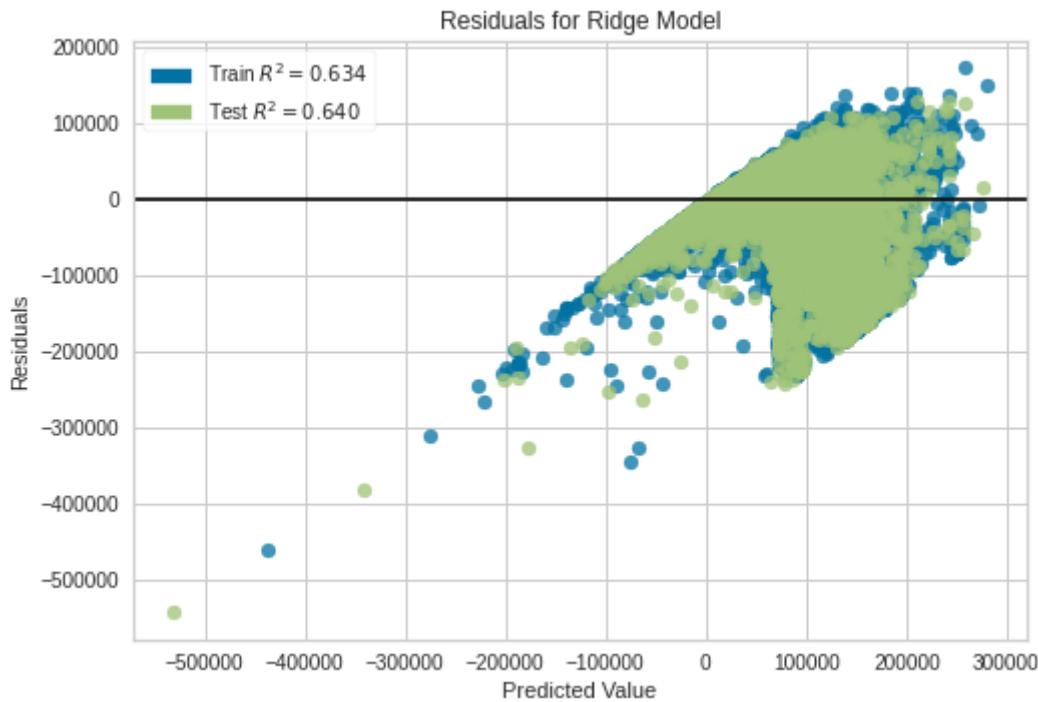
print("X2 TREINO : ", X_train2.shape)
print("X2 TESTE : ", X_test2.shape)
print("Y2 TREINO : ", y_train2.shape)
print("Y2 TESTE : ", y_test2.shape)

visualizer.fit(X_train, y_train) # Fit the training data to the visualizer
visualizer.score(X_test, y_test) # Evaluate the model on the test data
visualizer.show() # Finalize and render the figure
#https://www.scikit-yb.org/en/latest/api/regressor/residuals.html
```

```
X2 TREINO : (82548, 6)
X2 TESTE  : (35379, 6)
Y2 TREINO : (82548,)
Y2 TESTE  : (35379,)
```

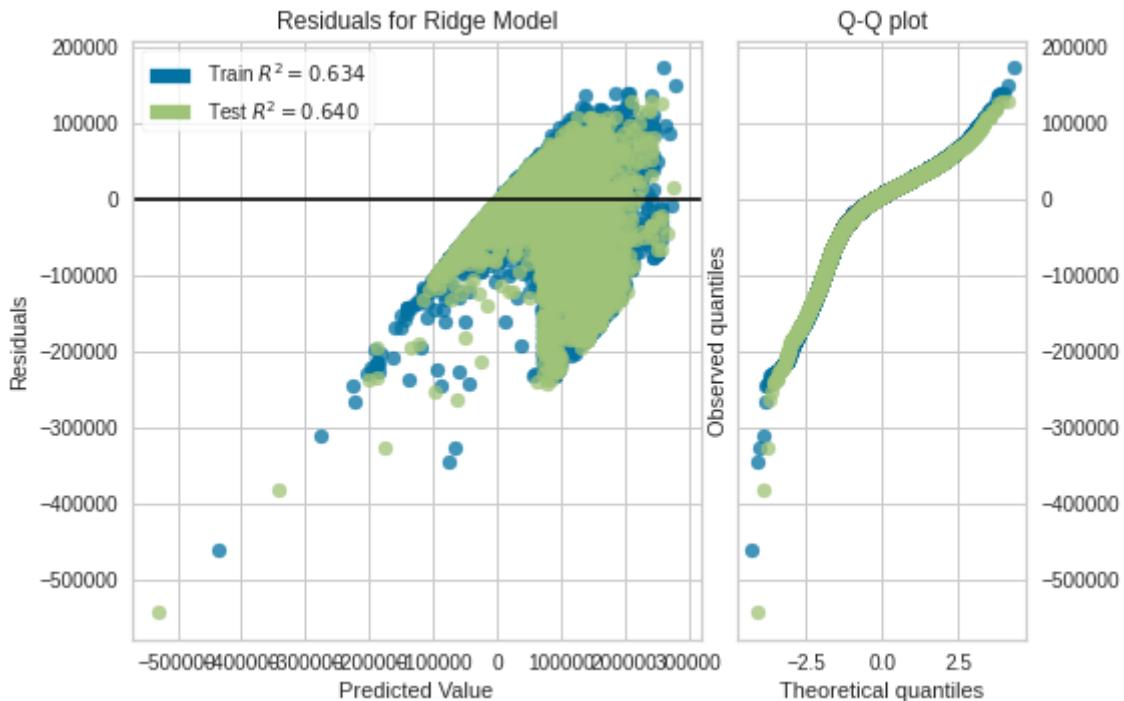


```
#NORMAL Q-Q RESIDUAL PLOT e Residual Plot DADOS TRATADOS IQR
visualizer = ResidualsPlot(model, hist=False)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f161fdf10>
```

```
#NORMAL Q-Q RESIDUAL PLOT e Residual Plot DADOS TRATADOS IQR
visualizer = ResidualsPlot(model, hist=False, qqplot=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f1c558690>
```

▼ LINEAR REGRESSION

▼ Tratamento de Ouliers

```
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)

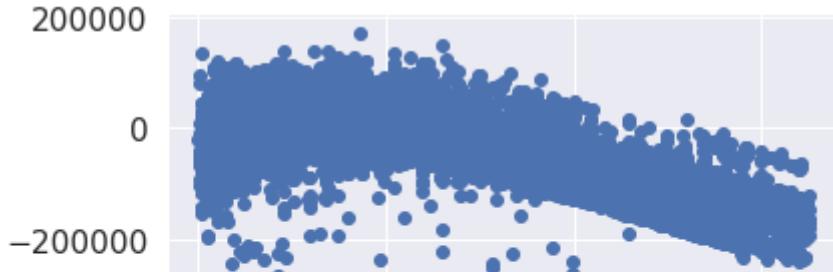
print("X-TRAINO E Y-TRAINO PONTUAÇÃO: ", regr.score(X_train,y_train))
print("X-TESTE E Y-TESTE PONTUAÇÃO: ", regr.score(X_test,y_test))
print("R2_SCORE MEDIDA DE AJUSTE DO MODELO: " ,r2_score(y_test,y_pred))
```

```
X-TRAINO E Y-TRAINO PONTUAÇÃO:  0.6337116732823859
X-TESTE E Y-TESTE PONTUAÇÃO:  0.6403172625313519
R2_SCORE MEDIDA DE AJUSTE DO MODELO:  0.6403172625313519
```

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

plt.scatter(y,regr.predict(X)-y)
```

```
<matplotlib.collections.PathCollection at 0x7f1f2c293310>
```



Avaliação do modelo

- Mostre um gráfico de resíduos (<https://www.qualtrics.com/support/stats-iq/analyses/regression-guides/interpreting-residual-plots-improve-regression/>)
<https://www.kirenz.com/post/2021-11-14-linear-regression-diagnostics-in-python/linear-regression-diagnostics-in-python/>
https://www.statsmodels.org/devel/generated/statsmodels.graphics.regressionplots.plot_partregress_grid.html

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
from matplotlib.pyplot import figure
from statsmodels.graphics.regressionplots import plot_partregress_grid
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols

lm = ols("price ~ year + mileage + vol_engine + Mark + Model + Fuel", data=DTCdf).fit()
print(lm.summary())
```

OLS Regression Results

```
=====
Dep. Variable:                  price      R-squared:                 0.636
Model:                          OLS        Adj. R-squared:            0.636
Method:                         Least Squares   F-statistic:             3.356e+04
Date:                          Sun, 15 May 2022   Prob (F-statistic):       0.00
Time:                           20:46:31        Log-Likelihood:          -1.3747e+06
No. Observations:                115390        AIC:                      2.749e+06
Df Residuals:                   115383        BIC:                      2.750e+06
Df Model:                           6
Covariance Type:                nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-1.106e+07	5.87e+04	-188.525	0.000	-1.12e+07	-1.09e+07
year	5515.2541	29.009	190.123	0.000	5458.397	5572.111
mileage	-0.2166	0.002	-116.932	0.000	-0.220	-0.213
vol_engine	33.5616	0.190	176.220	0.000	33.188	33.935
Mark	-734.1359	15.816	-46.418	0.000	-765.135	-703.137
Model	51.7499	1.137	45.534	0.000	49.522	53.977
Fuel	-2450.6619	103.593	-23.657	0.000	-2653.703	-2247.621

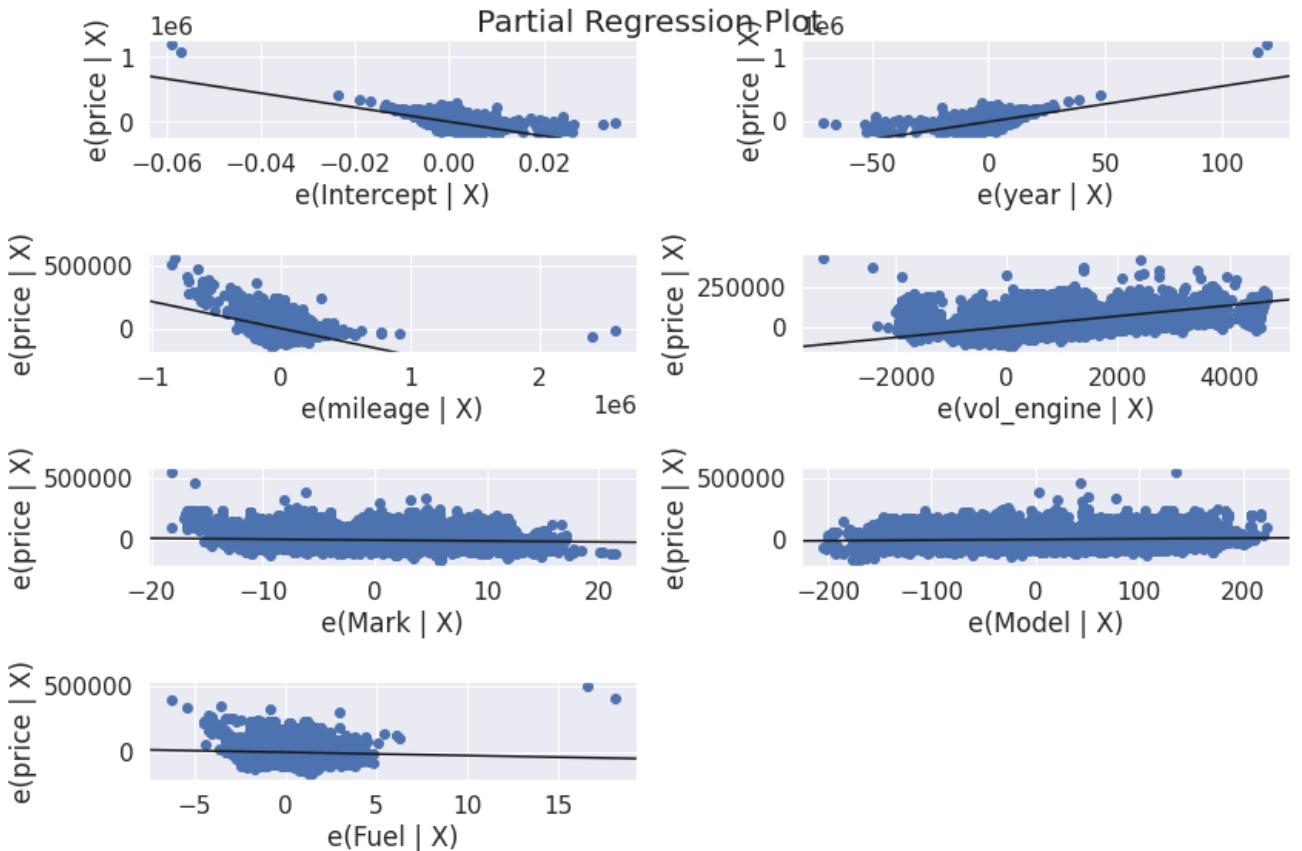
Omnibus:	48241.085	Durbin-Watson:	0.738
Prob(Omnibus):	0.000	Jarque-Bera (JB):	295303.772
Skew:	1.918	Prob(JB):	0.00
Kurtosis:	9.834	Cond. No.	9.40e+07

Warnings:

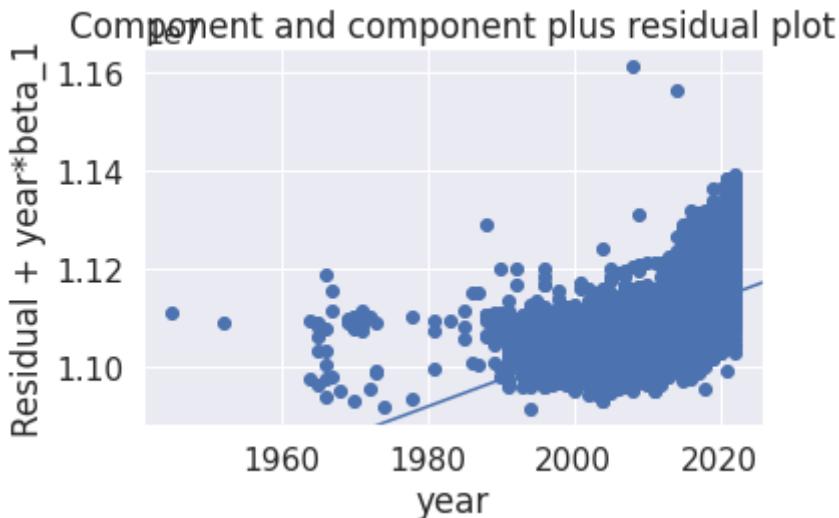
- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified
- [2] The condition number is large, 9.4e+07. This might indicate that there are strong multicollinearity or other numerical problems.

```
visualizer = ResidualsPlot(model, hist=False)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```

```
fig = plt.figure(figsize=(12, 8))
fig = sm.graphics.plot_partregress_grid(lm, fig=fig)
fig.tight_layout(pad=1.0)
#plot_partregress_grid(lm, fig=fig)
plt.show()
```



```
fig = sm.graphics.plot_ccpr(lm, "year")
fig.tight_layout(pad=1.0)
```



```
# ajusta o modelo de regressão multilinear
multi_model = ols('price ~ year + mileage + vol_engine + Mark + Model + Fuel', data=DTCd

#modifica o tamanho da figura
fig = plt.figure(figsize=(14, 8))

#criando gráficos de regressão
fig = sm.graphics.plot_regress_exog(multi_model, 'year', fig=fig)
```

```

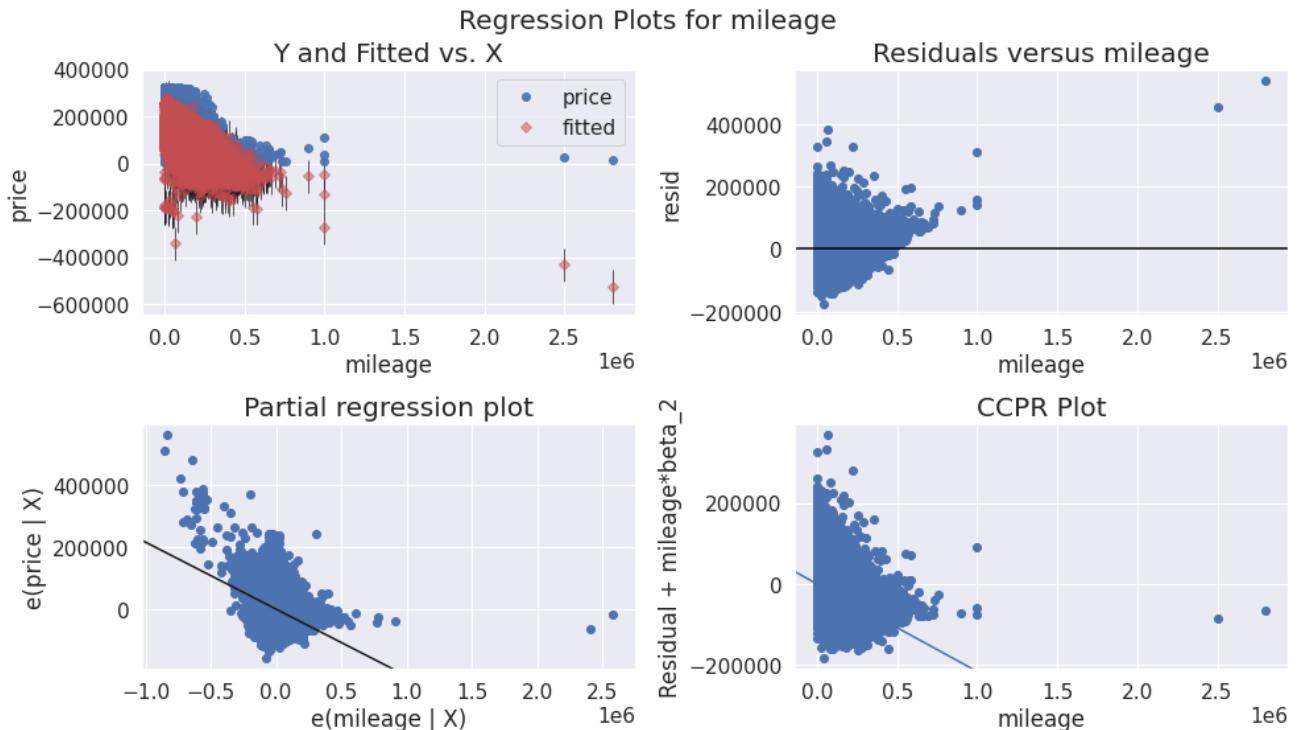
Regression Plots for year
Y and Fitted vs. X
Residuals versus year

# ajusta o modelo de regressão multilinear
multi_model = ols('price ~ year + mileage + vol_engine + Mark + Model + Fuel', data=DTCd

#modifica o tamanho da figura
fig = plt.figure(figsize=(14, 8))

# creating regression plots
fig = sm.graphics.plot_regress_exog(multi_model, 'mileage', fig=fig)

```



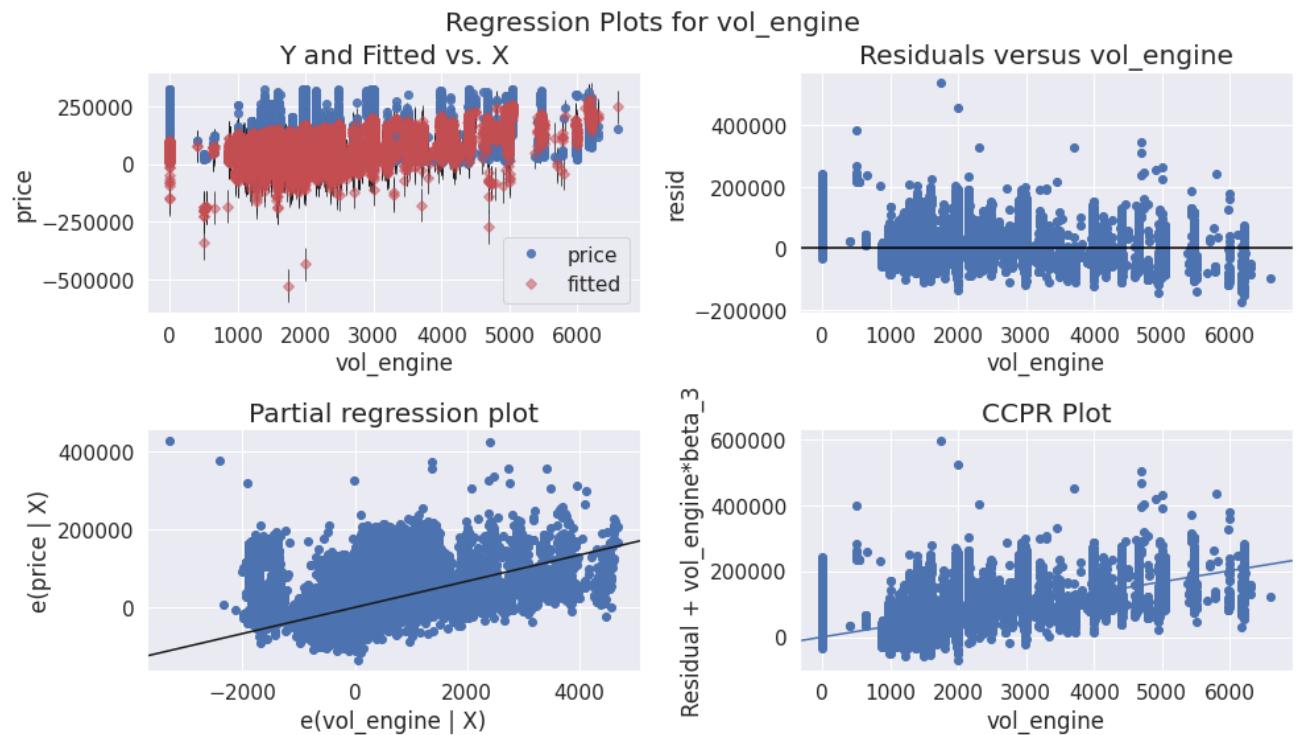
```

# ajusta o modelo de regressão multilinear
multi_model = ols('price ~ year + mileage + vol_engine + Mark + Model + Fuel', data=DTCd

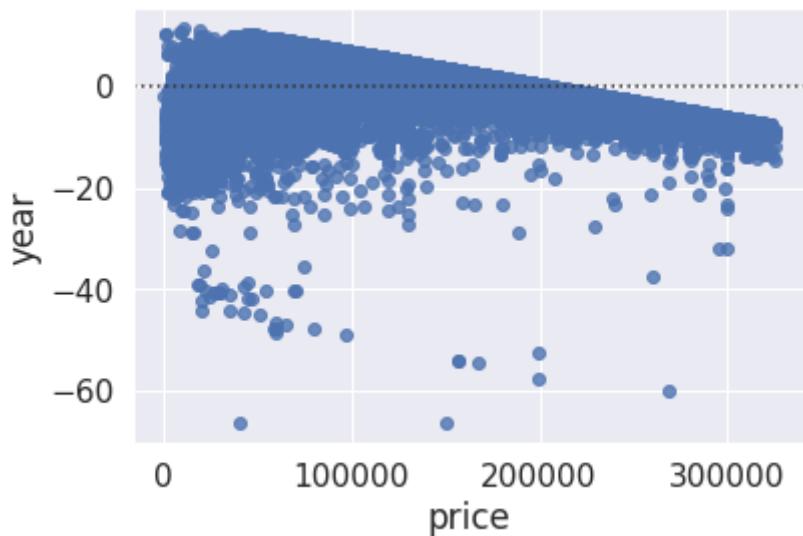
# modify figure size
fig = plt.figure(figsize=(14, 8))

#criando gráficos de regressão
fig = sm.graphics.plot_regress_exog(multi_model, 'vol_engine', fig=fig)

```



```
sns.residplot(x='price', y='year', data=DTCdf)
plt.show()
```



▼ Sem Tratamento de Outliers

```
# Dados não tratados
regr2 = linear_model.LinearRegression()
```

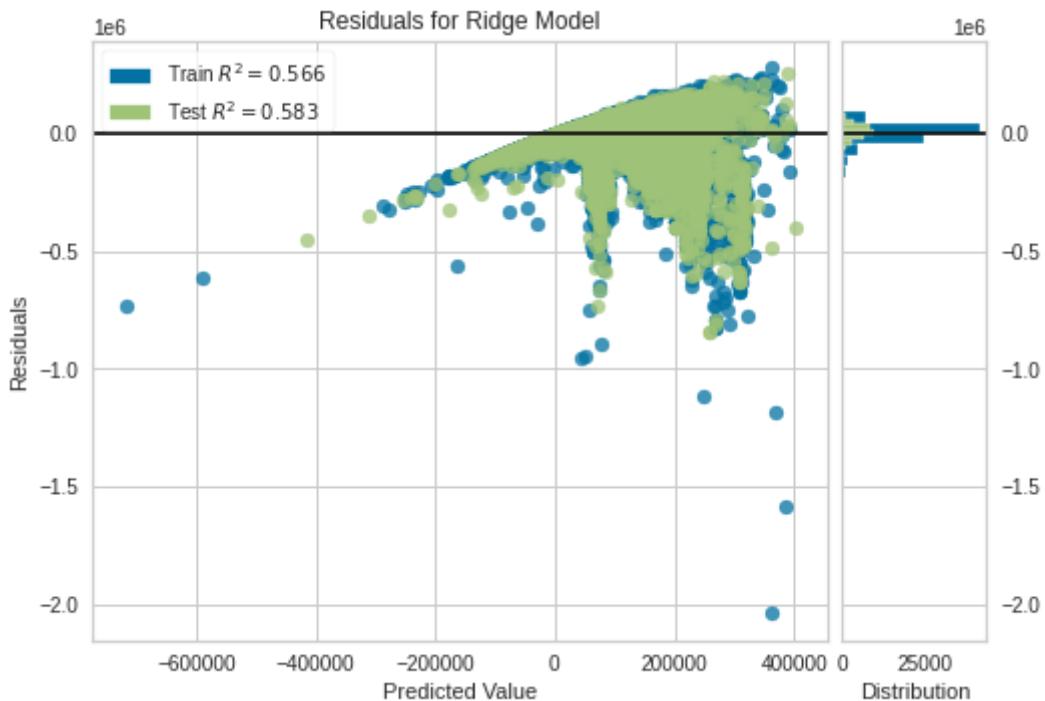
```

regr2.fit(X_train2, y_train2)
y_pred2 = regr.predict(X_test2)

#Intanciando as variaveis para analise de resíduos
model = Ridge()
visualizer = ResidualsPlot(model)

visualizer.fit(X_train2, y_train2) # Fit the training data to the visualizer
visualizer.score(X_test2, y_test2) # Evaluate the model on the test data
visualizer.show() # Finalize and render the figure
#https://www.scikit-yb.org/en/latest/api/regressor/residuals.html

```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f330c0c90>
```

```

print("X-TRAINO E Y-TRAINO PONTUAÇÃO: ", regr2.score(X_train2,y_train2))
print("X-TESTE E Y-TESTE PONTUAÇÃO: ", regr2.score(X_test2,y_test2))
print("R2_SCORE MEDIDA DE AJUSTE DO MODELO: " ,r2_score(y_test2,y_pred2))

```

```

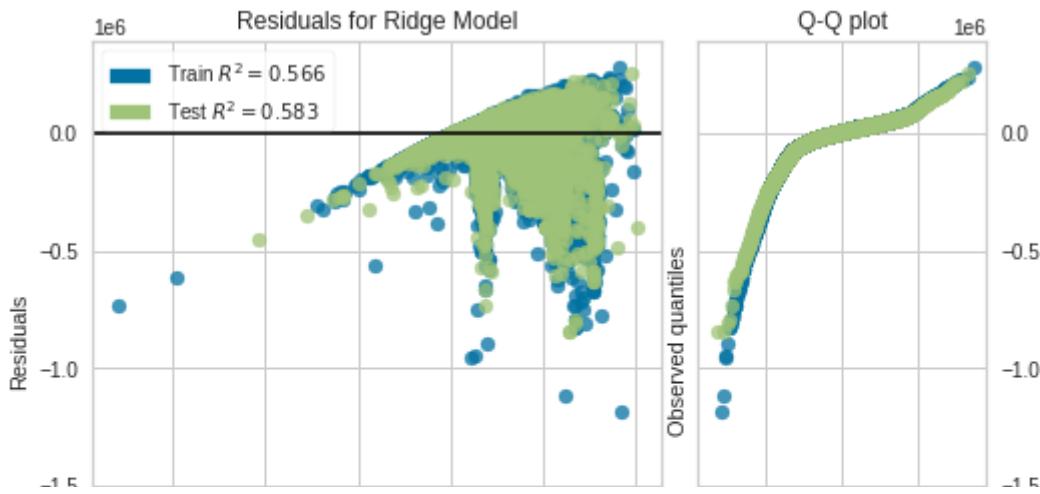
X-TRAINO E Y-TRAINO PONTUAÇÃO:  0.5660052861712401
X-TESTE E Y-TESTE PONTUAÇÃO:  0.583361185183262
R2_SCORE MEDIDA DE AJUSTE DO MODELO:  0.5424530613520974

```

```

#NORMAL Q-Q RESIDUAL PLOT e Residual Plot Dados não tratados
visualizer = ResidualsPlot(model, hist=False, qqplot=True)
visualizer.fit(X_train2, y_train2)
visualizer.score(X_test2, y_test2)
visualizer.show()

```



▼ DecisionTreeClassifier

▼ DADOS SEM OUTLIERS

```
# using the Decision Tree Regressor Model
DTR=DecisionTreeRegressor()
#TREINAMENTO
DTR.fit(X_train,y_train)

#PONTUAÇÕES DO MODELO
print("X-TREINO E Y-TREINO PONTUAÇÃO : ", DTR.score(X_train,y_train))
print("X-TESTE E Y-TESTE PONTUAÇÃO : ", DTR.score(X_test,y_test))

# VALORES ESPERADOS PARA DADOS
y_predDTR=DTR.predict(X_test)
print(" R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO" ,r2_score(y_test,y_predDTR))

X-TREINO E Y-TREINO PONTUAÇÃO :  0.9941348919797264
X-TESTE E Y-TESTE PONTUAÇÃO :  0.9151637252930044
R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO 0.9151637252930044
```

▼ DADOS C/ OUTLIERS

```
# using the Decision Tree Regressor Model
DTR2=DecisionTreeRegressor()
#TREINAMENTO
DTR2.fit(X_train2,y_train2)

#PONTUAÇÕES DO MODELO
print("X-TREINO E Y-TREINO PONTUAÇÃO : ", DTR2.score(X_train2,y_train2))
print("X-TESTE E Y-TESTE PONTUAÇÃO : ", DTR2.score(X_test2,y_test2))

# VALORES ESPERADOS PARA DADOS
y_predDTR2 = DTR2.predict(X_test2)
print(" R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO" ,r2_score(y_test2,y_predDTR2))
```

X-TREINO E Y-TREINO PONTUAÇÃO : 0.9940116651926156
 X-TESTE E Y-TESTE PONTUAÇÃO : 0.8871966222368667
 R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO 0.8871966222368667

▼ Random Forest Regressor Model S/ OULITERS

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split as tts

from yellowbrick.datasets import load_concrete
from yellowbrick.regressor import residuals_plot

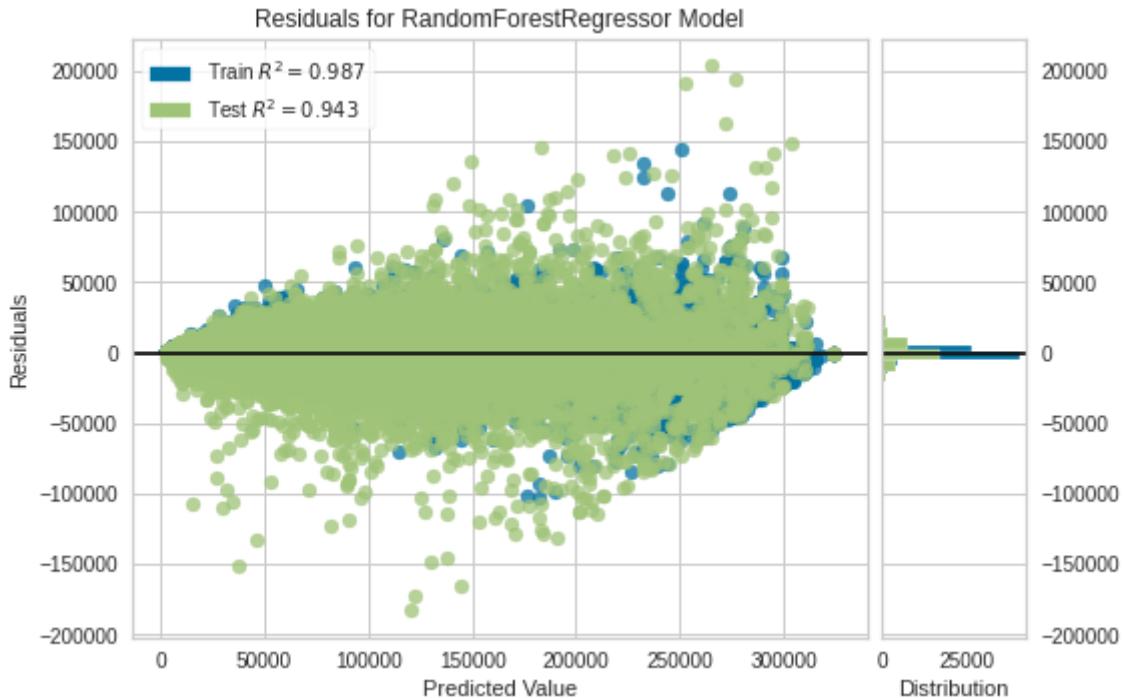
RFR=RandomForestRegressor()
RFR.fit(X_train,y_train)
y_predRFR =RFR.predict(X_test)

X, y = load_concrete()

print("X-TREINO E Y-TREINO PONTUAÇÃO : ", RFR.score(X_train,y_train))
print("X-TESTE E Y-TESTE PONTUAÇÃO : ", RFR.score(X_test,y_test))
print(" R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO" ,r2_score(y_test,y_predRFR))

viz = residuals_plot(RandomForestRegressor(), X_train, y_train, X_test, y_test)
```

X-TREINO E Y-TREINO PONTUAÇÃO : 0.9873129929452215
 X-TESTE E Y-TESTE PONTUAÇÃO : 0.9435699817856481
 R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO 0.9435699817856481



▼ Random Forest Regressor Model C/ OULITERS

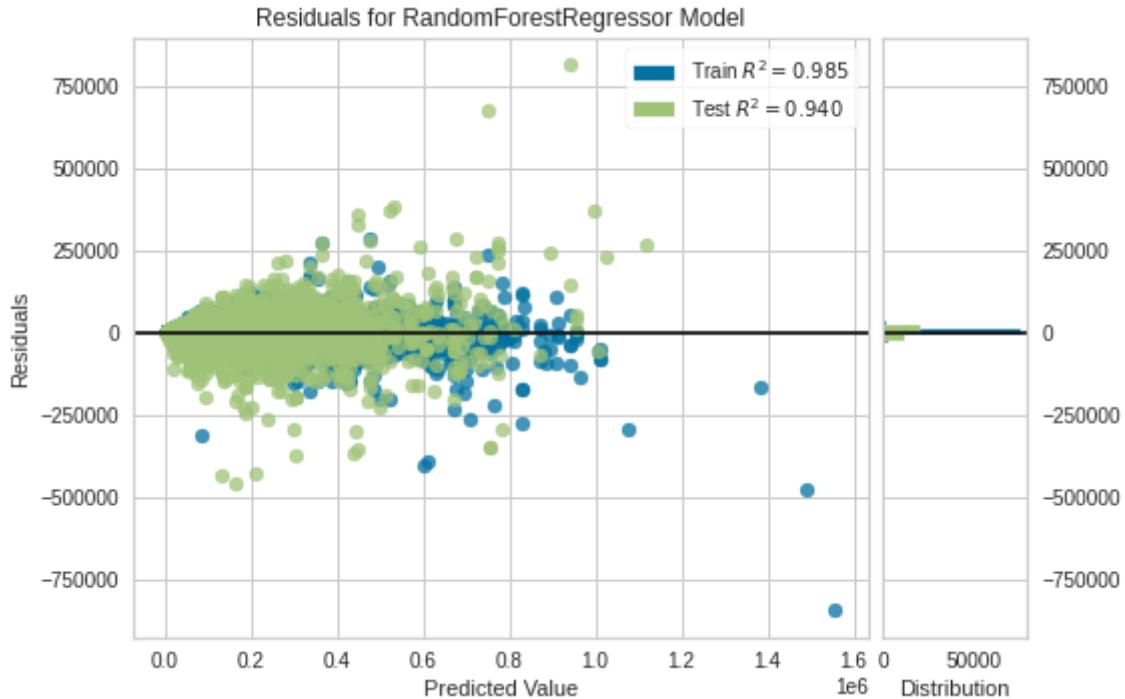
```
RFR2 = RandomForestRegressor()
RFR2.fit(X_train2,y_train2)
y_predRFR2 = RFR2.predict(X_test2)

X, y = load_concrete()

print("X-TREINO E Y-TREINO PONTUAÇÃO : ", RFR2.score(X_train2,y_train2))
print("X-TESTE E Y-TESTE PONTUAÇÃO : ", RFR2.score(X_test2,y_test2))
print(" R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO" ,r2_score(y_test2,y_predRFR2))

viz = residuals_plot(RandomForestRegressor(), X_train2, y_train2, X_test2, y_test2)
```

X-TREINO E Y-TREINO PONTUAÇÃO : 0.9861115132989903
 X-TESTE E Y-TESTE PONTUAÇÃO : 0.9401878405601664
 R2_SCORE PONTUAÇÃO MEDIA E ESTATISTICA DO MODELO 0.9401878405601664



- TERMINAR CADA UM COMO ANALISE JUSTA E DIRETA.
- ◦ CORRELACAO DE PEARSON E MATRIZ DA CORELACAO DE SPEAR
- ◦ MATRIZ DE CONFUSAO
- ◦ ESCOLHER A IMPUTACAO DAS VARIAVEIS CATEGORICAS
- ◦ ANALISAR A CARDINALIDADE DOS ATRIBUTOS E FAZER AGRUPAMENTO OU NAO
- ◦ MEDIR A CORRELACAO
- ◦ METRICAS DE REGRESSAO (UMA OU DUAS).
- ◦ PLOT: X VALORES REAIS, Y VALORES PREDITOS E DIAGONAL A RETA DE REFERENCIA

