Lucas de Lucena Siqueira - 2010080354

Regressão Linear Múltipla

Carregando o Dataset Boston Houses

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-residential acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per 10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. TARGET: Median value of owner-occupied homes in $1000's

In [62]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.datasets import load_boston
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

In [63]:

```
# Gerando o dataset
boston = load_boston()
dataset = pd.DataFrame(boston.data, columns = boston.feature_names)
dataset['target'] = boston.target
```

In [64]:

```
dataset.head()
```

Out[64]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [65]:

```
dataset.describe()
```

Out[65]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	1.

In [66]:

```
dataset.describe().T
```

Out[66]:

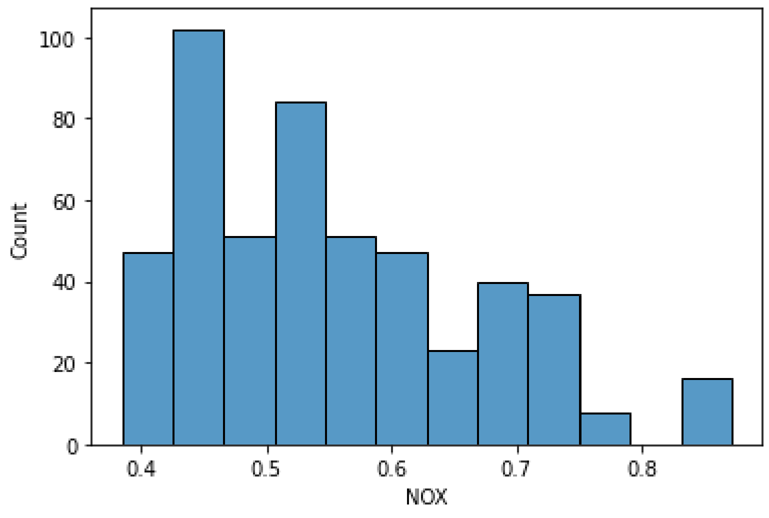
	count	mean	std	min	25%	50%	75%	ma
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.976
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.00
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.74
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.00
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.871
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.78
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.00
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.126
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.00
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.00
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.00
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.90
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.97
target	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.00

In [67]:

```
import seaborn as sns

sns.histplot(dataset.NOX)
```

Out[67]:

```
<AxesSubplot:xlabel='NOX', ylabel='Count'>
```

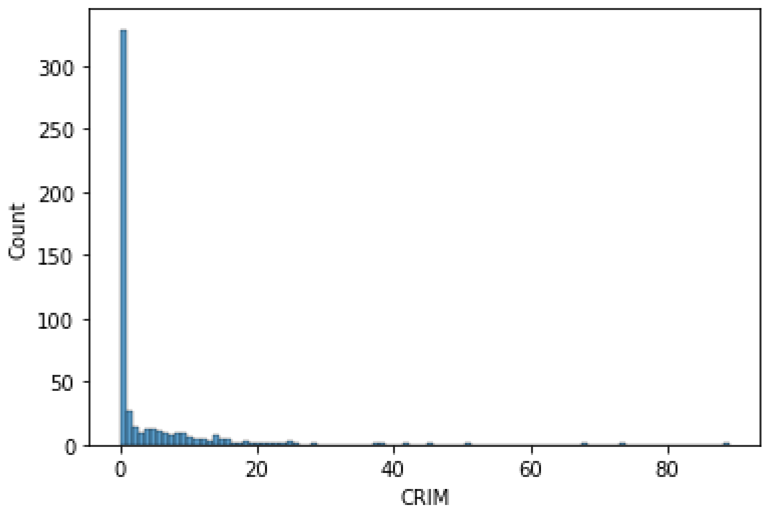


In [68]:

```
sns.histplot(dataset.CRIM)
```

Out[68]:

```
<AxesSubplot:xlabel='CRIM', ylabel='Count'>
```



In [69]:

```
dataset.describe()['target'] # variável preditora ou Classe ou Label
```

Out[69]:

```
count    506.000000
mean      22.532806
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: target, dtype: float64
```

In [70]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  target   506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [71]:

```
observations = len(dataset)
observations
```

Out[71]:

```
506
```

In [72]:

```
dataset.head()
```

Out[72]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [73]:

```
dataset.iloc[:,:-1][:3]
```

Out[73]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4

In [74]:

```
X = dataset.iloc[:,:-1]
y = dataset['target'].values
```

In [75]:

```
X.head()
```

Out[75]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [76]:

```
len(X.columns)
```

Out[76]:

```
13
```

In [77]:

```
y[:5]
```

Out[77]:

```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

Matriz de Correlação

In [78]:

```
# Gerando a matriz
X = dataset.iloc[:,:-1]
matriz_corr = X.corr()
print (matriz_corr)
```

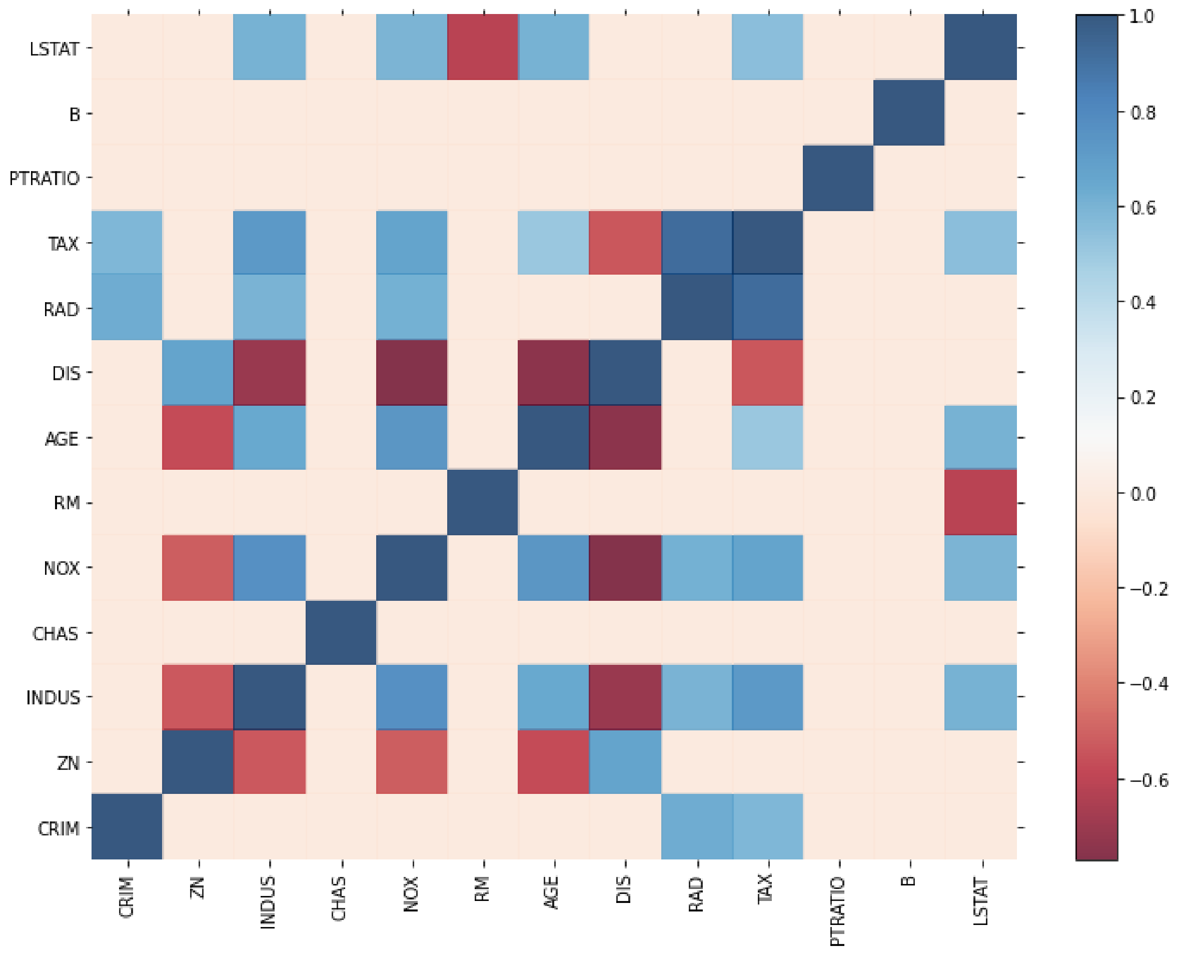
```
             CRIM        ZN     INDUS      CHAS       NOX        RM        AG
E  \
CRIM     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.35273
4
ZN      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.56953
7
INDUS    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.64477
9
CHAS    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.08651
8
NOX      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.73147
0
RM      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.24026
5
AGE      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.00000
0
DIS     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.74788
1
RAD      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.45602
2
TAX      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.50645
6
PTRATIO  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.26151
5
B       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.27353
4
LSTAT    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.60233
9

              DIS       RAD       TAX   PTRATIO         B     LSTAT
CRIM    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000
```

In [79]:

```
# Criando um Correlation Plot
def visualize_correlation_matrix(data, hurdle = 0.0):
    fig = plt.figure(figsize=(12,9))
    ax = fig.add_subplot(111)
    R = np.corrcoef(data, rowvar = 0)
    R[np.where(np.abs(R) < hurdle)] = 0.0
    heatmap = plt.pcolor(R, cmap = 'RdBu' , alpha = 0.8) #mpl.cm.coolwarm
    heatmap.axes.set_frame_on(False)
    heatmap.axes.set_yticks(np.arange(R.shape[0]) + 0.5, minor = False)
    heatmap.axes.set_xticks(np.arange(R.shape[1]) + 0.5, minor = False)
    heatmap.axes.set_xticklabels(dataset.columns[:-1], minor = False)
    plt.xticks(rotation=90)
    heatmap.axes.set_yticklabels(dataset.columns[:-1], minor = False)
    plt.tick_params(axis = 'both', which = 'both', bottom = 'off', top = 'off', left = 'off
    plt.colorbar()
    plt.show()
```

In [80]:

```
# Visualizando o Plot
visualize_correlation_matrix(X, hurdle = 0.5)
```



In [81]:

```
# matriz de Correlação com a variável preditora
mt = pd.DataFrame(dataset.values, columns=dataset.columns)
mt_corr = mt.corr()
print (abs(mt_corr['target']).sort_values(ascending=False))
```

```
target     1.000000
LSTAT      0.737663
RM         0.695360
PTRATIO    0.507787
INDUS      0.483725
TAX        0.468536
NOX        0.427321
CRIM       0.388305
RAD        0.381626
AGE        0.376955
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
Name: target, dtype: float64
```

In [82]:

```
X.head()
```

Out[82]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [83]:

```
standardization = StandardScaler()
Xst = standardization.fit_transform(X)
original_means = standardization.mean_
originanal_stds = standardization.scale_
print('Dataset Original')
X.head()
```

```
Dataset Original
```

Out[83]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [84]:

```
print('Dataset Padronizado')
dstd = pd.DataFrame(Xst, columns=boston.feature_names)
dstd.head()
```

```
Dataset Padronizado
```

Out[84]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RA
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.98284
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.86788
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.86788
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.75292
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.75292

In [85]:

```
X.max()
```

Out[85]:

```
CRIM        88.9762
ZN         100.0000
INDUS       27.7400
CHAS         1.0000
NOX          0.8710
RM           8.7800
AGE        100.0000
DIS         12.1265
RAD         24.0000
TAX        711.0000
PTRATIO     22.0000
B          396.9000
LSTAT       37.9700
dtype: float64
```

In [86]:

```
dstd.max()
```

Out[86]:

```
CRIM       9.933931
ZN         3.804234
INDUS      2.422565
CHAS       3.668398
NOX        2.732346
RM         3.555044
AGE        1.117494
DIS        3.960518
RAD        1.661245
TAX        1.798194
PTRATIO    1.638828
B          0.441052
LSTAT      3.548771
dtype: float64
```

In [87]:

```
X.min()
```

Out[87]:

```
CRIM         0.00632
ZN           0.00000
INDUS        0.46000
CHAS         0.00000
NOX          0.38500
RM           3.56100
AGE          2.90000
DIS          1.12960
RAD          1.00000
TAX        187.00000
PTRATIO     12.60000
B            0.32000
LSTAT        1.73000
dtype: float64
```

In [88]:

```
dstd.min()
```

Out[88]:

```
CRIM      -0.419782
ZN        -0.487722
INDUS     -1.557842
CHAS      -0.272599
NOX       -1.465882
RM        -3.880249
AGE       -2.335437
DIS       -1.267069
RAD       -0.982843
TAX       -1.313990
PTRATIO   -2.707379
B         -3.907193
LSTAT     -1.531127
dtype: float64
```

In [89]:

```
y[:5]
```

Out[89]:

```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

In [90]:

```
dfXst = pd.DataFrame(Xst, columns=boston.feature_names)
dfXst.head()
```

Out[90]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RA
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.98284
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.86788
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.86788
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.75292
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.75292

In [91]:

```
Xinv = standardization.inverse_transform(Xst)
dfinverser = pd.DataFrame(Xinv, columns=boston.feature_names)
dfinverser.head()
```

Out[91]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [92]:

```
dataset.head()
```

Out[92]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

Criar o modelo

In [93]:

```
X = dataset.copy()
X.head()
```

Out[93]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [94]:

```
atributos = list(dataset.columns[:-1])
atributos
```

Out[94]:

```
['CRIM',
 'ZN',
 'INDUS',
 'CHAS',
 'NOX',
 'RM',
 'AGE',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT']
```

```
Importância dos Atributos
target 1.000000
LSTAT 0.737663
RM 0.695360
PTRATIO 0.507787
INDUS 0.483725
TAX 0.468536
NOX 0.427321
CRIM 0.388305
RAD 0.381626
AGE 0.376955
ZN 0.360445
B 0.333461
DIS 0.249929
CHAS 0.175260
```

In [95]:

```
atributos = ['CRIM',
 'ZN',
 'INDUS',
 'CHAS',
 'NOX',
 'RM',
 'AGE',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT']

X = X[ atributos ]
X.head()
```

Out[95]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [96]:

```
y[:3]
```

Out[96]:

```
array([24. , 21.6, 34.7])
```

In [97]:

```
# Criando um modelo
modelo = linear_model.LinearRegression()
modelo.fit(X,y)
```

Out[97]:

```
LinearRegression()
```

In [98]:

```
modelo.coef_
```

Out[98]:

```
array([-1.08011358e-01,  4.64204584e-02,  2.05586264e-02,  2.68673382e+00,
       -1.77666112e+01,  3.80986521e+00,  6.92224640e-04, -1.47556685e+00,
        3.06049479e-01, -1.23345939e-02, -9.52747232e-01,  9.31168327e-03,
       -5.24758378e-01])
```

In [99]:

```
for coef, var in sorted(zip(modelo.coef_, dataset.columns[:-1]), reverse = True):
    print ("%6.3f %s" % (coef,var))
```

```
 3.810 RM
 2.687 CHAS
 0.306 RAD
 0.046 ZN
 0.021 INDUS
 0.009 B
 0.001 AGE
-0.012 TAX
-0.108 CRIM
-0.525 LSTAT
-0.953 PTRATIO
-1.476 DIS
-17.767 NOX
```

In [100]:

```
def r2_est(X,y):
    modelo = linear_model.LinearRegression(normalize = False, fit_intercept = True)
    return r2_score(y, modelo.fit(X,y).predict(X))
```

In [101]:

```
print ('R2: %0.3f' %  r2_est(X,y))
```

```
R2: 0.741
```

In [102]:

```
#             CRIM    ZN  INDUS    CHAS       NOX      RM     AGE    DIS   RAD      TAX    PT
Xteste = [   2.06,   0.0, 11,       0.0,       0.4,     3. ,   80.2,  2.4,  1.0,     273.0,
    ]

modelo.predict(np.array(Xteste).reshape(1, -1))[0]
```

Out[102]:

```
10.157489025241802
```

In [103]:

```
#             CRIM    ZN  INDUS    CHAS       NOX      RM     AGE    DIS   RAD      TAX    PT
Xteste = [
    [          0.02,   0.2, 7.01,    0.0,     0.5,   7.1,   45.2,  6.1,  3.0,    222,   15.2,
    [          0.01,   0.1,11.01,    0.0,     0.6,   6.1,   80.2,  2.5,  1.0,    273,   21.2,
]
modelo.predict(np.array(Xteste))
```

Out[103]:

```
array([30.36056954, 19.46052668])
```

Métricas para Algoritmos de Regressão

Gerando o dataset

In [104]:

```
dataset['y_prev'] = modelo.predict(X)

dataset.head()
```

Out[104]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [105]:

```
dataset['Erro'] = abs (dataset['y_prev'] - dataset['target'] )
dataset.head()
```

Out[105]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [106]:

```
dataset.head(8)
```

Out[106]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	1
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	1

In [107]:

```
dataset.Erro.sum()
```

Out[107]:

```
1655.0565823155594
```

In [108]:

```
print(dataset.shape)
dataset.head()
```

```
(506, 16)
```

Out[108]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

MAE - Mean Absolute Error

É a soma da diferença absoluta entre previsões e valores reais.
Fornece uma ideia de quão erradas estão nossas previsões.
Valor igual a 0 indica que não há erro, sendo a previsão perfeita
(a exemplo do Logloss, a função cross_val_score inverte o valor)

In [109]:

```python
from sklearn import model_selection
num_folds = 39
num_instances = len(X)
seed = 15

modelo = linear_model.LinearRegression()
#modelo.fit(X,y)

# Separando os dados em folds
kfold = model_selection.KFold(num_folds, True, random_state = seed)
resultado = model_selection.cross_val_score(modelo, X, y, cv = kfold,
                                            scoring = 'neg_mean_absolute_error')

# Print do resultado
print("MAE: %.3f (%.3f)" % (resultado.mean(), resultado.std()))
```

```
MAE: -3.394 (0.922)

C:\Users\lukki\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: F
utureWarning: Pass shuffle=True as keyword args. From version 1.0 (renaming
of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

In [110]:

```python
resultado
```

Out[110]:

```
array([-4.18697907, -2.08923472, -3.18363222, -3.7614362 , -3.19397272,
       -1.52381741, -5.1048777 , -5.08811661, -3.96605435, -5.16308534,
       -3.47484933, -3.30915297, -3.14767813, -2.69519848, -2.30820926,
       -3.67471263, -3.64253366, -3.70426882, -2.81143274, -4.55148864,
       -4.94755769, -3.22358895, -4.23140924, -2.72511995, -1.93977719,
       -3.1240936 , -2.91551155, -3.07665534, -2.61020172, -4.49081277,
       -2.92750354, -2.96027884, -1.88987024, -3.53258648, -4.42471784,
       -2.66993093, -3.68847424, -2.25297621, -4.15643672])
```

In [111]:

```
# Modelo de árvore de Decisão
from sklearn.ensemble import RandomForestRegressor

clf = RandomForestRegressor(n_estimators=30, max_depth=8)
clf = clf.fit(X, y)

from sklearn import model_selection

num_folds = 39
num_instances = len(X)
seed = 15

modelo = RandomForestRegressor(n_estimators=35, max_depth=8)
#modelo.fit(X,y)

# Separando os dados em folds
kfold = model_selection.KFold(num_folds, True, random_state = seed)

resultado = model_selection.cross_val_score(modelo, X, y, cv = kfold,
                                            scoring = 'neg_mean_absolute_error')

# Print do resultado
print("MAE: %.3f (%.3f)" % (resultado.mean(), resultado.std()))
```

```
C:\Users\lukki\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: F
utureWarning: Pass shuffle=True as keyword args. From version 1.0 (renaming
of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

MAE: -2.266 (0.598)
```

In [112]:

```
resultado = model_selection.cross_val_score(modelo, X, y, cv = kfold, scoring = 'r2')

# Print do resultado
print("R^2: %.3f (%.3f)" % (resultado.mean(), resultado.std()))
```

```
R^2: 0.841 (0.107)
```

MSE - Mean Squared Error

Similar ao MAE, fornece a magnitude do erro do modelo.
Ao extrairmos a raiz quadrada do MSE convertemos as unidades de volta ao original, o que pode ser útil para descrição e apresentação.
Isso é chamado RMSE (Root Mean Squared Error)

In [113]:

```
# Definindo os valores para o número de folds
num_folds = 39
num_instances = len(X)
seed = 15

# Separando os dados em folds
kfold = model_selection.KFold(num_folds, True, random_state = seed)

resultado = model_selection.cross_val_score(modelo, X, y, cv = kfold, scoring = 'neg_mean_s

# Print do resultado
print("MSE: %.3f (%.3f)" % (resultado.mean(), resultado.std()))
```

```
C:\Users\lukki\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: F
utureWarning: Pass shuffle=True as keyword args. From version 1.0 (renaming
of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

MSE: -11.543 (9.072)
```

RMSE (Root Mean Squared Error)

Similar ao MAE, fornece a magnitude do erro do modelo.
Ao extrairmos a raiz quadrada do MSE convertemos as unidades de volta ao original, o que pode ser útil para descrição e apresentação.

In [114]:

```
from math import sqrt
print("RMSE: %.3f " % (sqrt(abs(resultado.mean()))))
```

```
RMSE: 3.397
```

R2

Essa métrica fornece uma indicação do nível de precisão das previsões em relação aos valores observados.
Também chamado de coeficiente de determinação.
Valores entre 0 e 1, sendo 1 o valor ideal.

In [115]:

```
resultado = model_selection.cross_val_score(modelo, X, y, cv = kfold, scoring = 'r2')

# Print do resultado
print("R^2: %.3f (%.3f)" % (resultado.mean(), resultado.std()))
```

```
R^2: 0.850 (0.097)
```

In [116]:

```
resultado
```

Out[116]:

```
array([0.85224962, 0.92144949, 0.8080172 , 0.93305663, 0.77288653,
       0.91171039, 0.9423383 , 0.42979943, 0.79820523, 0.96300008,
       0.77773214, 0.8460102 , 0.77402207, 0.82367251, 0.91837477,
       0.92005312, 0.92454005, 0.86295388, 0.86503861, 0.84416788,
       0.83110075, 0.93236789, 0.91906516, 0.92403703, 0.90116676,
       0.77560184, 0.94760285, 0.88847331, 0.89477132, 0.88708809,
       0.95559191, 0.94466212, 0.73224341, 0.78733454, 0.82390998,
       0.69135547, 0.74673489, 0.88965547, 0.79142092])
```