João Emanuel da Silva Lins

Matricula: 162080263

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

h(x) = CRIM *w1* + *ZN* w2 + ... + LSTAT * w13

In [51]:

```
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.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

In [52]:

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

In [53]:

```
dataset.head()
```

Out[53]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [54]:

```
len(dataset)
```

Out[54]:

506

Análise Descritiva

In [55]:

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

Out[55]:

	count	mean	std	min	25%	50%	75%	
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.
target	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.

In [56]:

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

Out[56]:

```
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 [57]:

```
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
```

Gerando número de observações

In [58]:

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

Out[58]:

```
506
```

In [59]:

```
dataset.head()
```

Out[59]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

Coletando x e y

In [60]:

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

Out[60]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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

In [61]:

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

In [62]:

```
X.head()
```

Out[62]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [63]:

```
# forma de remover
del X['PTRATIO']
del X['RAD']
```

In [64]:

```
# forma de remover
X = X[ ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX'] ] #
 'B', 'LSTAT'
X.head()
```

Out[64]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	TAX
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	296.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	242.0
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	242.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	222.0
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	222.0

In [65]:

```
y[:5]
```

Out[65]:

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

Matriz de Correlação

In [66]:

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

```
                CRIM        ZN     INDUS      CHAS       NOX        RM  
AGE  \
CRIM      1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  
0.352734
ZN       -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 
-0.569537
INDUS     0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  
0.644779
CHAS     -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  
0.086518
NOX       0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  
0.731470
RM       -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 
-0.240265
AGE       0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  
1.000000
DIS      -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 
-0.747881
RAD       0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  
0.456022
TAX       0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  
0.506456
PTRATIO   0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  
0.261515
B        -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 
-0.273534
LSTAT     0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  
0.602339

                 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  
```

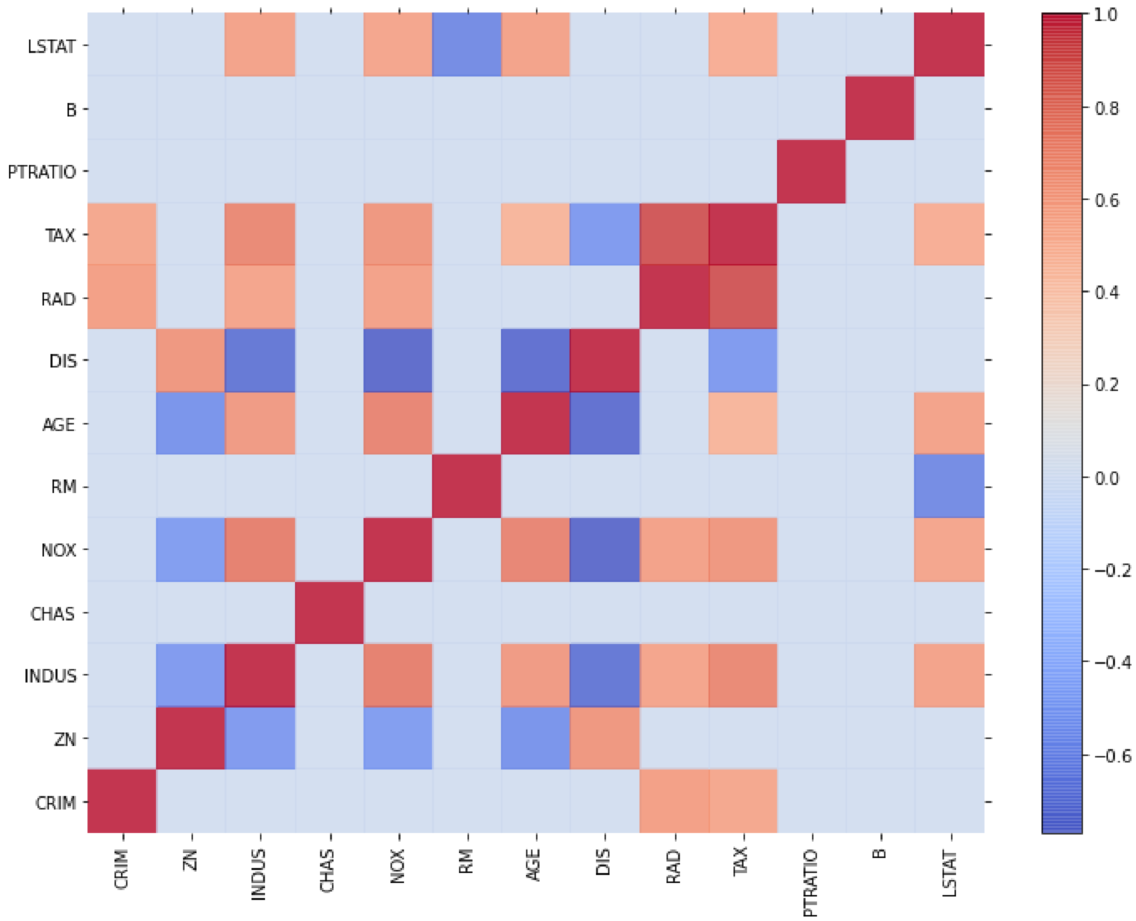
Visualizando a matriz de correlação (somente os atributos)

In [67]:

```
# 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 = mpl.cm.coolwarm, alpha = 0.8)
    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', right = 'off')
    plt.colorbar()
    plt.show()
```

In [68]:

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



In [69]:

```
# 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
```

Feature Scaling

In [70]:

```
X.head()
```

Out[70]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

Podemos aplicar Feature Scaling através de Padronização ou Normalização.
Normalização aplica escala aos dados com intervalos entre 0 e 1.
A Padronização divide a média pelo desvio padrão para obter uma unidade de variância.
Vamos usar a Padronização (StandardScaler) pois nesse caso esta técnica ajusta os coeficientes e torna a superfície de erros mais "tratável".

Aplicando Padronização

In [71]:

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

```
Dataset Original
```

Out[71]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [72]:

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

```
Dataset Padronizado
```

Out[72]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.98
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.86
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.86
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.75
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.75

In [73]:

```
X.max()
```

Out[73]:

```
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 [74]:

```
dstd.max()
```

Out[74]:

```
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 [75]:

```
X.min()
```

Out[75]:

```
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 [76]:

```
dstd.min()
```

Out[76]:

```
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 [77]:

```
y[:5]
```

Out[77]:

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

Desfazendo a Padronização

In [78]:

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

Out[78]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.98
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.86
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.86
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.75
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.75

In [79]:

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

Out[79]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [80]:

```
dataset.head()
```

Out[80]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

Criar o modelo

In [81]:

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

Out[81]:

```
LinearRegression()
```

Mostrar os pesos dos Atributos (yprev = w1X1 + w2X2 + ... + wnXn)

In [82]:

```
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
```

h(x) = 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

Avaliando o modelo com o R Squared (R^2)

In [83]:

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

In [84]:

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

```
R2: 0.741
```

Gera o impacto de cada atributo no R^2

In [85]:

```
r2_impact = list()
for j in range(X.shape[1]):
    selection = [i for i in range(X.shape[1]) if i!=j]
    r2_impact.append(((r2_est(X,y) - r2_est(X.values[:,selection],y)), dataset.c
olumns[j]))

for imp, varname in sorted(r2_impact, reverse = True):
    print ('%6.3f %s' %  (imp, varname))
```

```
 0.056 LSTAT
 0.044 RM
 0.029 DIS
 0.028 PTRATIO
 0.011 NOX
 0.011 RAD
 0.006 B
 0.006 ZN
 0.006 CRIM
 0.006 TAX
 0.005 CHAS
 0.000 INDUS
 0.000 AGE
```

Fazer Previsões

In [86]:

```
dataset.tail()
```

Out[86]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90

In [87]:

```
#            CRIM          ZN   INDUS         CHAS        NOX       RM          AGE
DIS        RAD       TAX     PTRATIO         B           LSTAT
Xteste = [   1.,         0.0, 11,       0.0,     0.5,    6. ,    80.2,   2.3,   1.0,
273.0,    21.2,     396.1,   7.4
    ]

#m = modelo.fit(X,y)
#Xteste_norm = standardization.fit_transform(X)
modelo.predict(np.array(Xteste).reshape(1, -1))[0]
```

Out[87]:

23.760829991455928

In [88]:

```
Xteste = [
    [   0.02,    0.2, 7.01,    0.0,    0.5,   7.1,   45.2,   6.1,   3.0,     222,    15.2
,       396.1,    5.4],
    [   0.01,    0.1,11.01,    0.0,    0.6,   6.1,   80.2,   2.5,   1.0,     273,    21.2
,       396.9,    12.6],
]
modelo.predict(np.array(Xteste))
```

Out[88]:

array([30.36056954, 19.46052668])

Métricas para Algoritmos de Regressão

Gerando o dataset

In [89]:

```
dataset['y_prev'] = modelo.predict(dataset.iloc[:,:-1].values)
#dataset['Erro'] = dataset['y_prev'] - dataset['target']
dataset.head()
```

Out[89]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [90]:

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

Out[90]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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
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
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
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
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

In [91]:

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

Out[91]:

```
1655.0565823155603
```

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 [92]:

```
from sklearn import model_selection
num_folds = 10
num_instances = len(X)
seed = 7

# 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: -3.387 (0.667)

C:\Users\Dijay Lima\anaconda3\lib\site-packages\sklearn\utils\valida
tion.py:68: FutureWarning: Pass shuffle=True as keyword args. From v
ersion 0.25 passing these as positional arguments will result in an
error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

In [93]:

```
resultado
```

Out[93]:

```
array([-3.24826136, -4.11939173, -2.27347659, -4.17287637, -3.803425
7 ,
       -3.20177954, -4.13777416, -3.11898137, -3.49219054, -2.301920
09])
```

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 [94]:

```
# Definindo os valores para o número de folds
num_folds = 10
num_instances = len(X)
seed = 7

# 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_squared_error')

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

```
MSE: -23.747 (11.143)

C:\Users\Dijay Lima\anaconda3\lib\site-packages\sklearn\utils\valida
tion.py:68: FutureWarning: Pass shuffle=True as keyword args. From v
ersion 0.25 passing these as positional arguments will result in an
error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

Valor Original - MSE: -23.747 (11.143)

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 [95]:

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

RMSE: 4.873

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 [96]:

```
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: 0.718 (0.099)

R^2: 0.718 (0.099)

Exercício: Remover 2 atributos e executar o jupyter (armazenar o valor anterior das métricas)

In [ ]:

```

```