01-DSA-Cap05-RegressaoLinearSimples

March 1, 2021

1 Data Science Academy - Machine Learning

2 Capítulo 5 - Regressão

3 Regressão Linear Simples

3.1 Definindo o Problema de Negócio

Nosso objetivo é construir um modelo de Machine Learning que seja capaz de fazer previsões sobre a taxa média de ocupação de casas na região de Boston, EUA, por proprietários. A variável a ser prevista é um valor numérico que representa a mediana da taxa de ocupação das casas em Boston. Para cada casa temos diversas variáveis explanatórias. Sendo assim, podemos resolver este problema empregando Regressão Linear Simples ou Múltipla.

3.2 Definindo o Dataset

Usaremos o Boston Housing Dataset, que é um conjunto de dados que tem a taxa média de ocupação das casas, juntamente com outras 13 variáveis que podem estar relacionadas aos preços das casas. Esses são os fatores como condições socioeconômicas, condições ambientais, instalações educacionais e alguns outros fatores semelhantes. Existem 506 observações nos dados para 14 variáveis. Existem 12 variáveis numéricas em nosso conjunto de dados e 1 variável categórica. O objetivo deste projeto é construir um modelo de regressão linear para estimar a taxa média de ocupação das casas pelos proprietários em Boston.

```
In [1]: from IPython.display import Image
        Image('imagens/boston.png')
```

```
Out[1]:
```

Housing Data Set

Download: Data Folder, Data Set Description

Abstract: Taken from StatLib library

Data Set Characteristics:	Multivariate	**Number of Instances:**	506	**Area:**	N/A
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	14	**Date Donated**	1993-07-07
Associated Tasks:	Regression	**Missing Values?**	No	**Number of Web Hits:**	172017

Source:

Origin:

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

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-retail business 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 [2]:
```
# Carregando o Dataset Boston Houses
from sklearn.datasets import load_boston
boston = load_boston()

# Carregando Bibliotecas Python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import warnings
```

```
warnings.filterwarnings("ignore")
%matplotlib inline
```

3.3 Análise Exploratória

```
In [3]: # Convertendo o dataset em um dataframe com Pandas
        dataset = pd.DataFrame(boston.data, columns = boston.feature_names)
        dataset['target'] = boston.target
```

```
In [4]: dataset.head(5)
```

```
Out[4]:       CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
        0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
        1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
        2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
        3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
        4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

           PTRATIO       B  LSTAT  target
        0     15.3  396.90   4.98    24.0
        1     17.8  396.90   9.14    21.6
        2     17.8  392.83   4.03    34.7
        3     18.7  394.63   2.94    33.4
        4     18.7  396.90   5.33    36.2
```

```
In [5]: # Calculando a média da variável de resposta
        valor_medio_esperado_na_previsao = dataset['target'].mean()
```

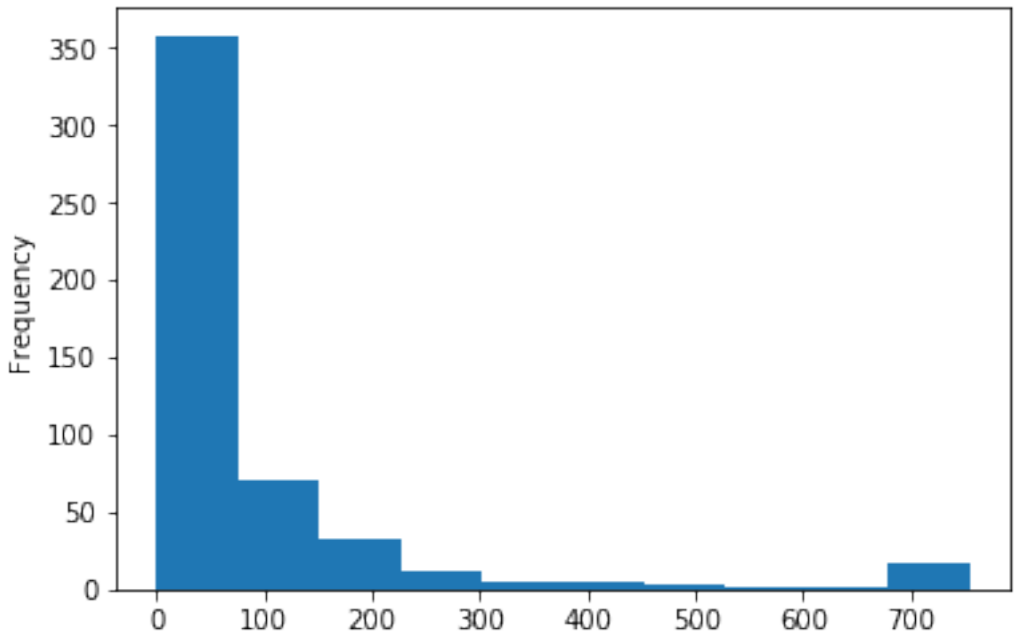
```
In [6]: valor_medio_esperado_na_previsao
```

```
Out[6]: 22.532806324110698
```

```
In [7]: # Calculando (simulando) o SSE
        # O SSE é a diferença ao quadrado entre o valor previsto e o valor observado.
        # Considerando que o valor previsto seja igual a média, podemos considerar que
        # y = média da variável target (valores observados).

        # Estamos apenas simulando o SSE, uma vez que a regressão ainda não foi criada e os va
        # ainda não foram calculados.

        squared_errors = pd.Series(valor_medio_esperado_na_previsao - dataset['target'])**2
        SSE = np.sum(squared_errors)
        print ('Soma dos Quadrados dos Erros (SSE): %01.f' % SSE)
```

```
Soma dos Quadrados dos Erros (SSE): 42716
```

```
In [8]: # Histograma dos erros
        # Temos mais erros "pequenos", ou seja, mais valores próximos à média.
        hist_plot = squared_errors.plot('hist')
```



Para Regressão Linear Simples usaremos como variável explanatória a variável RM que representa o número médio de quartos nas casas.

```
In [9]: # Função para calcular o desvio padrão
        def calc_desvio_padrao(variable, bias = 0):
            observations = float(len(variable))
            return np.sqrt(np.sum((variable - np.mean(variable))**2) / (observations - min(bias
```

```
In [10]: # Imprimindo o desvio padrão via fórmula e via NumPy da variável RM
         print ('Resultado da Função: %0.5f Resultado do Numpy: %0.5f' % (calc_desvio_padrao(da
```

```
Resultado da Função: 0.70192 Resultado do Numpy: 0.70192
```

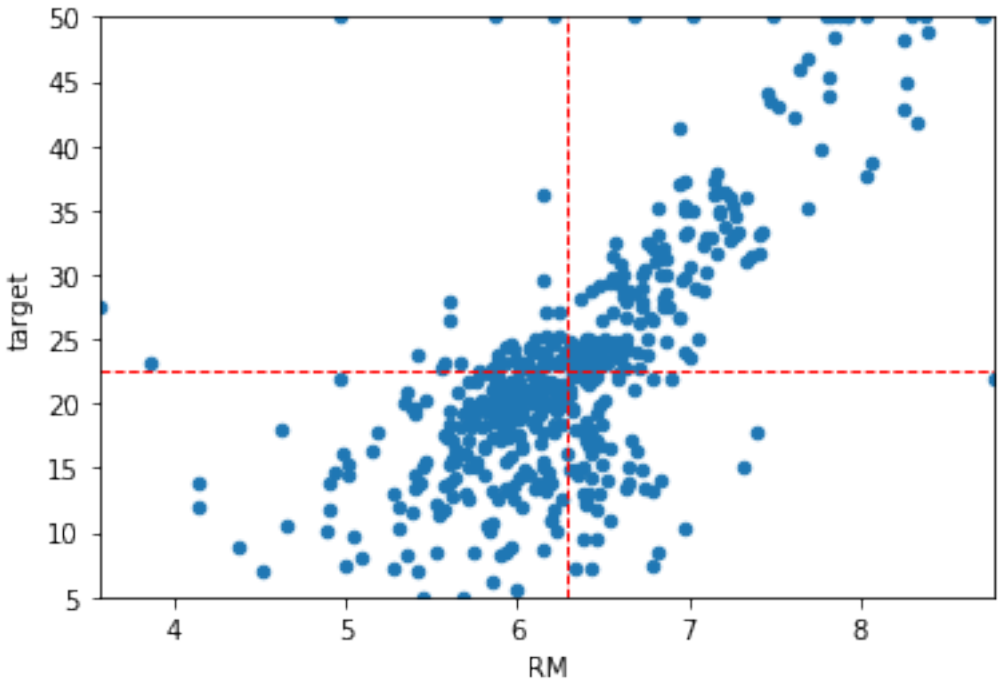
```
In [11]: # Funções para calcular a variância da variável RM e a correlação com a variável targ
         def covariance(variable_1, variable_2, bias = 0):
             observations = float(len(variable_1))
             return np.sum((variable_1 - np.mean(variable_1)) * (variable_2 - np.mean(variable_

         def standardize(variable):
             return (variable - np.mean(variable)) / np.std(variable)

         def correlation(var1, var2, bias = 0):
             return covariance(standardize(var1), standardize(var2), bias)
```

```
In [12]: # Compara o resultado das nossas funções com a função pearsonr do SciPy
         from scipy.stats.stats import pearsonr
         print ('Nossa estimativa de Correlação: %0.5f' % (correlation(dataset['RM'], dataset[
         print ('Correlação a partir da função pearsonr do SciPy: %0.5f' % pearsonr(dataset['RM
```

```
Nossa estimativa de Correlação: 0.69536
Correlação a partir da função pearsonr do SciPy: 0.69536
```

```
In [13]: # Definindo o range dos valores de x e y
         x_range = [dataset['RM'].min(),dataset['RM'].max()]
         y_range = [dataset['target'].min(),dataset['target'].max()]
```

```
In [14]: # Plot dos valores de x e y com a média
         scatter_plot = dataset.plot(kind = 'scatter', x = 'RM', y = 'target', xlim = x_range,

         # Cálculo da média
         meanY = scatter_plot.plot(x_range, [dataset['target'].mean(),dataset['target'].mean()]
         meanX = scatter_plot.plot([dataset['RM'].mean(), dataset['RM'].mean()], y_range, '--'
```



3.4 Regressão Linear com o StatsModels

https://www.statsmodels.org/stable/index.html

```
In [15]: # Importando as funções
         import statsmodels.api as sm
```

```
In [16]: # Gerando X e Y. Vamos adicionar a constante ao valor de X, gerando uma matrix.
         y = dataset['target']
         X = dataset['RM']
```

```
In [17]: # Esse comando adiciona os valores dos coefientes à variável X (o bias será calculado
         X = sm.add_constant(X)

In [18]: X.head()
```

Out[18]:

	const	RM
0	1.0	6.575
1	1.0	6.421
2	1.0	7.185
3	1.0	6.998
4	1.0	7.147

```
In [19]: # Criando o modelo de regressão
         modelo = sm.OLS(y, X)

         # Treinando o modelo
         modelo_v1 = modelo.fit()

In [20]: print(modelo_v1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 target   R-squared:                       0.484
Model:                            OLS   Adj. R-squared:                  0.483
Method:                 Least Squares   F-statistic:                     471.8
Date:                Sat, 28 Sep 2019   Prob (F-statistic):           2.49e-74
Time:                        17:51:33   Log-Likelihood:                -1673.1
No. Observations:                 506   AIC:                             3350.
Df Residuals:                     504   BIC:                             3359.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -34.6706      2.650    -13.084      0.000     -39.877     -29.465
RM             9.1021      0.419     21.722      0.000       8.279       9.925
==============================================================================
Omnibus:                      102.585   Durbin-Watson:                   0.684
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              612.449
Skew:                           0.726   Prob(JB):                    1.02e-133
Kurtosis:                       8.190   Cond. No.                         58.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [21]: print(modelo_v1.params)
```

```
const   -34.670621
RM        9.102109
dtype: float64
```

```
In [22]: # Gerando os valores previstos
         valores_previstos = modelo_v1.predict(X)
         valores_previstos
```

```
Out[22]: 0      25.175746
         1      23.774021
         2      30.728032
         3      29.025938
         4      30.382152
         5      23.855940
         6      20.051258
         7      21.507596
         8      16.583355
         9      19.978442
         10     23.373528
         11     20.023952
         12     18.931699
         13     19.477826
         14     20.815836
         15     18.431083
         16     19.350396
         17     19.851012
         18     14.990486
         19     17.457157
         20     16.028126
         21     19.623459
         22     21.234533
         23     18.239939
         24     19.250273
         25     16.292087
         26     18.239939
         27     20.369832
         28     24.447577
         29     26.076855
                  ...
         476    24.347454
         477    13.606965
         478    21.625923
         479    22.026416
         480    22.144743
         481    26.768615
         482    29.599371
         483    17.775731
```

```
484    18.767861
485    22.781891
486    20.979674
487    19.077333
488    14.972282
489    14.608197
490    11.686420
491    19.787297
492    19.787297
493    17.275115
494    19.268477
495    16.938337
496    14.389747
497    18.066999
498    20.114973
499    16.019024
500    20.187790
501    25.339584
502    21.034286
503    28.825691
504    27.169108
505    20.215096
Length: 506, dtype: float64
```

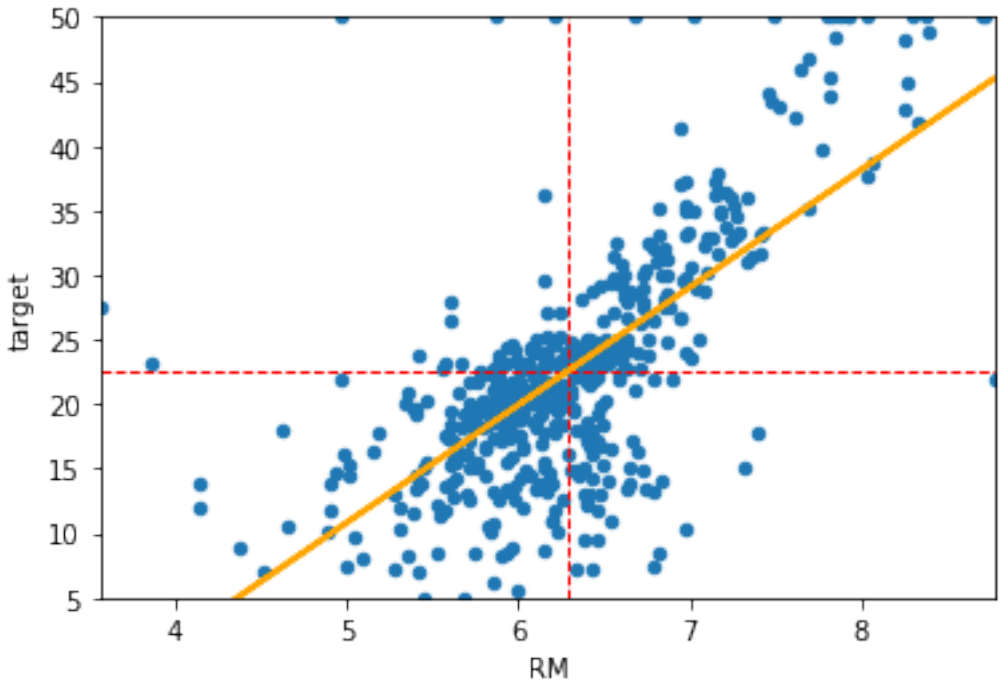
```
In [23]: # Fazendo previsões com o modelo treinado
         RM = 5
         Xp = np.array([1, RM])
         print ("Se RM = %01.f nosso modelo prevê que a mediana da taxa de ocupação é %0.1f" %
```

```
Se RM = 5 nosso modelo prevê que a mediana da taxa de ocupação é 10.8
```

3.4.1 Gerando um ScatterPlot com a Linha de Regressão

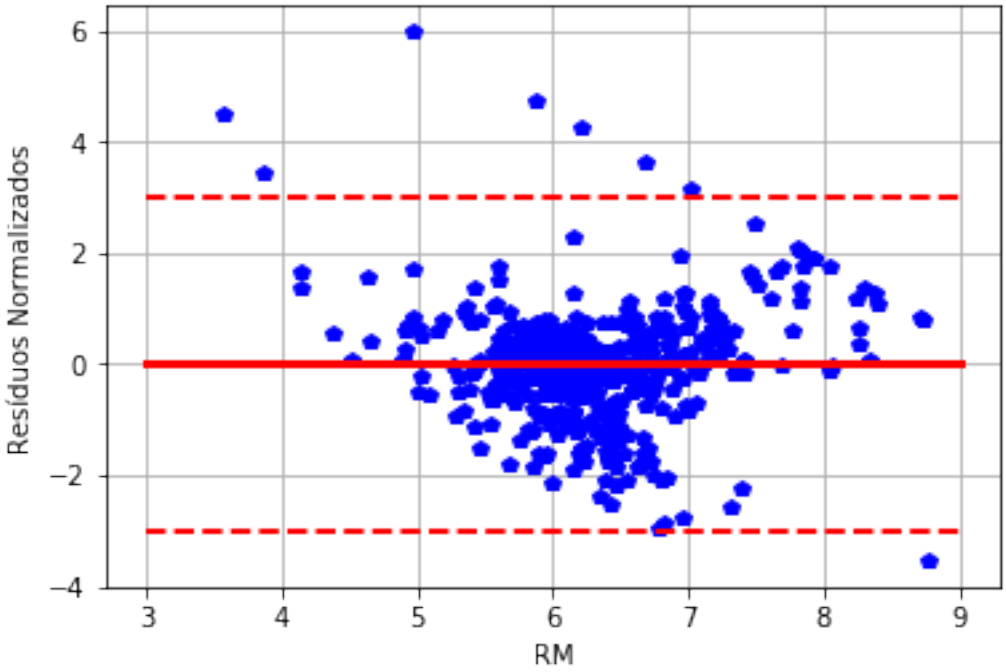
```
In [24]: # Range de valores para x e y
         x_range = [dataset['RM'].min(), dataset['RM'].max()]
         y_range = [dataset['target'].min(), dataset['target'].max()]
```

```
In [25]: # Primeira camada do Scatter Plot
         scatter_plot = dataset.plot(kind = 'scatter', x = 'RM', y = 'target', xlim = x_range,

         # Segunda camada do Scatter Plot (médias)
         meanY = scatter_plot.plot(x_range, [dataset['target'].mean(),dataset['target'].mean()]
         meanX = scatter_plot.plot([dataset['RM'].mean(),dataset['RM'].mean()], y_range, '--',

         # Terceira camada do Scatter Plot (linha de regressão)
         regression_line = scatter_plot.plot(dataset['RM'], valores_previstos, '-', color = 'o
```



In [26]:
```
# Gerando os resíduos
residuos = dataset['target'] - valores_previstos
residuos_normalizados = standardize(residuos)
```

In [27]:
```
# ScatterPlot dos resíduos
residual_scatter_plot = plt.plot(dataset['RM'], residuos_normalizados,'bp')
plt.xlabel('RM')
plt.ylabel('Resíduos Normalizados')
mean_residual = plt.plot([int(x_range[0]),round(x_range[1],0)], [0,0], '-', color = '
upper_bound = plt.plot([int(x_range[0]),round(x_range[1],0)], [3,3], '--', color = 'r
lower_bound = plt.plot([int(x_range[0]),round(x_range[1],0)], [-3,-3], '--', color =
plt.grid()
```



3.5 Regressão Linear com Scikit-Learn

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

```
In [28]: from sklearn import linear_model

In [29]: # Cria o objeto
         modelo_v2 = linear_model.LinearRegression(normalize = False, fit_intercept = True)

In [30]: # Define os valores de x e y
         num_observ = len(dataset)
         X = dataset['RM'].values.reshape((num_observ, 1)) # X deve sempre ser uma matriz e nu
         y = dataset['target'].values # y pode ser um vetor

In [31]: type(X)

Out[31]: numpy.ndarray

In [32]: # Número de dimensões de X (matriz)
         np.ndim(X)

Out[32]: 2

In [33]: print(X)
```

```
[[6.575]
 [6.421]
 [7.185]
 [6.998]
 [7.147]
 [6.43 ]
 [6.012]
 [6.172]
 [5.631]
 [6.004]
 [6.377]
 [6.009]
 [5.889]
 [5.949]
 [6.096]
 [5.834]
 [5.935]
 [5.99 ]
 [5.456]
 [5.727]
 [5.57 ]
 [5.965]
 [6.142]
 [5.813]
 [5.924]
 [5.599]
 [5.813]
 [6.047]
 [6.495]
 [6.674]
 [5.713]
 [6.072]
 [5.95 ]
 [5.701]
 [6.096]
 [5.933]
 [5.841]
 [5.85 ]
 [5.966]
 [6.595]
 [7.024]
 [6.77 ]
 [6.169]
 [6.211]
 [6.069]
 [5.682]
 [5.786]
 [6.03 ]
```

[5.399]
[5.602]
[5.963]
[6.115]
[6.511]
[5.998]
[5.888]
[7.249]
[6.383]
[6.816]
[6.145]
[5.927]
[5.741]
[5.966]
[6.456]
[6.762]
[7.104]
[6.29]
[5.787]
[5.878]
[5.594]
[5.885]
[6.417]
[5.961]
[6.065]
[6.245]
[6.273]
[6.286]
[6.279]
[6.14]
[6.232]
[5.874]
[6.727]
[6.619]
[6.302]
[6.167]
[6.389]
[6.63]
[6.015]
[6.121]
[7.007]
[7.079]
[6.417]
[6.405]
[6.442]
[6.211]
[6.249]
[6.625]

[6.163]
[8.069]
[7.82]
[7.416]
[6.727]
[6.781]
[6.405]
[6.137]
[6.167]
[5.851]
[5.836]
[6.127]
[6.474]
[6.229]
[6.195]
[6.715]
[5.913]
[6.092]
[6.254]
[5.928]
[6.176]
[6.021]
[5.872]
[5.731]
[5.87]
[6.004]
[5.961]
[5.856]
[5.879]
[5.986]
[5.613]
[5.693]
[6.431]
[5.637]
[6.458]
[6.326]
[6.372]
[5.822]
[5.757]
[6.335]
[5.942]
[6.454]
[5.857]
[6.151]
[6.174]
[5.019]
[5.403]
[5.468]

[4.903]
[6.13]
[5.628]
[4.926]
[5.186]
[5.597]
[6.122]
[5.404]
[5.012]
[5.709]
[6.129]
[6.152]
[5.272]
[6.943]
[6.066]
[6.51]
[6.25]
[7.489]
[7.802]
[8.375]
[5.854]
[6.101]
[7.929]
[5.877]
[6.319]
[6.402]
[5.875]
[5.88]
[5.572]
[6.416]
[5.859]
[6.546]
[6.02]
[6.315]
[6.86]
[6.98]
[7.765]
[6.144]
[7.155]
[6.563]
[5.604]
[6.153]
[7.831]
[6.782]
[6.556]
[7.185]
[6.951]
[6.739]

[7.178]
[6.8]
[6.604]
[7.875]
[7.287]
[7.107]
[7.274]
[6.975]
[7.135]
[6.162]
[7.61]
[7.853]
[8.034]
[5.891]
[6.326]
[5.783]
[6.064]
[5.344]
[5.96]
[5.404]
[5.807]
[6.375]
[5.412]
[6.182]
[5.888]
[6.642]
[5.951]
[6.373]
[6.951]
[6.164]
[6.879]
[6.618]
[8.266]
[8.725]
[8.04]
[7.163]
[7.686]
[6.552]
[5.981]
[7.412]
[8.337]
[8.247]
[6.726]
[6.086]
[6.631]
[7.358]
[6.481]
[6.606]

[6.897]
[6.095]
[6.358]
[6.393]
[5.593]
[5.605]
[6.108]
[6.226]
[6.433]
[6.718]
[6.487]
[6.438]
[6.957]
[8.259]
[6.108]
[5.876]
[7.454]
[8.704]
[7.333]
[6.842]
[7.203]
[7.52]
[8.398]
[7.327]
[7.206]
[5.56]
[7.014]
[8.297]
[7.47]
[5.92]
[5.856]
[6.24]
[6.538]
[7.691]
[6.758]
[6.854]
[7.267]
[6.826]
[6.482]
[6.812]
[7.82]
[6.968]
[7.645]
[7.923]
[7.088]
[6.453]
[6.23]
[6.209]

[6.315]
[6.565]
[6.861]
[7.148]
[6.63]
[6.127]
[6.009]
[6.678]
[6.549]
[5.79]
[6.345]
[7.041]
[6.871]
[6.59]
[6.495]
[6.982]
[7.236]
[6.616]
[7.42]
[6.849]
[6.635]
[5.972]
[4.973]
[6.122]
[6.023]
[6.266]
[6.567]
[5.705]
[5.914]
[5.782]
[6.382]
[6.113]
[6.426]
[6.376]
[6.041]
[5.708]
[6.415]
[6.431]
[6.312]
[6.083]
[5.868]
[6.333]
[6.144]
[5.706]
[6.031]
[6.316]
[6.31]
[6.037]

[5.869]
[5.895]
[6.059]
[5.985]
[5.968]
[7.241]
[6.54]
[6.696]
[6.874]
[6.014]
[5.898]
[6.516]
[6.635]
[6.939]
[6.49]
[6.579]
[5.884]
[6.728]
[5.663]
[5.936]
[6.212]
[6.395]
[6.127]
[6.112]
[6.398]
[6.251]
[5.362]
[5.803]
[8.78]
[3.561]
[4.963]
[3.863]
[4.97]
[6.683]
[7.016]
[6.216]
[5.875]
[4.906]
[4.138]
[7.313]
[6.649]
[6.794]
[6.38]
[6.223]
[6.968]
[6.545]
[5.536]
[5.52]

[4.368]
[5.277]
[4.652]
[5.   ]
[4.88]
[5.39]
[5.713]
[6.051]
[5.036]
[6.193]
[5.887]
[6.471]
[6.405]
[5.747]
[5.453]
[5.852]
[5.987]
[6.343]
[6.404]
[5.349]
[5.531]
[5.683]
[4.138]
[5.608]
[5.617]
[6.852]
[5.757]
[6.657]
[4.628]
[5.155]
[4.519]
[6.434]
[6.782]
[5.304]
[5.957]
[6.824]
[6.411]
[6.006]
[5.648]
[6.103]
[5.565]
[5.896]
[5.837]
[6.202]
[6.193]
[6.38]
[6.348]
[6.833]

[6.425]
[6.436]
[6.208]
[6.629]
[6.461]
[6.152]
[5.935]
[5.627]
[5.818]
[6.406]
[6.219]
[6.485]
[5.854]
[6.459]
[6.341]
[6.251]
[6.185]
[6.417]
[6.749]
[6.655]
[6.297]
[7.393]
[6.728]
[6.525]
[5.976]
[5.936]
[6.301]
[6.081]
[6.701]
[6.376]
[6.317]
[6.513]
[6.209]
[5.759]
[5.952]
[6.003]
[5.926]
[5.713]
[6.167]
[6.229]
[6.437]
[6.98]
[5.427]
[6.162]
[6.484]
[5.304]
[6.185]
[6.229]

```
 [6.242]
 [6.75 ]
 [7.061]
 [5.762]
 [5.871]
 [6.312]
 [6.114]
 [5.905]
 [5.454]
 [5.414]
 [5.093]
 [5.983]
 [5.983]
 [5.707]
 [5.926]
 [5.67 ]
 [5.39 ]
 [5.794]
 [6.019]
 [5.569]
 [6.027]
 [6.593]
 [6.12 ]
 [6.976]
 [6.794]
 [6.03 ]]
```

```
In [34]: type(y)
```

```
Out[34]: numpy.ndarray
```

```
In [35]: # Número de dimensões de y (vetor)
         np.ndim(y)
```

```
Out[35]: 1
```

```
In [36]: print(y)
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
```

```
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

```
In [37]: # Treinamento do modelo - fit()
         modelo_v2.fit(X,y)
```

```
Out[37]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)
```

```
In [38]: # Imprime os coeficientes
         print (modelo_v2.coef_)
         print (modelo_v2.intercept_)
```

```
[9.10210898]
-34.67062077643857
```

```
In [39]: # Imprime as previsões
         print (modelo_v2.predict(X))
```

```
[25.17574577 23.77402099 30.72803225 29.02593787 30.38215211 23.85593997
 20.05125842 21.50759586 16.5833549  19.97844155 23.3735282  20.02395209
 18.93169901 19.47782555 20.81583557 18.43108302 19.35039603 19.85101202
```

14.99048582 17.45715736 16.02812625 19.6234593 21.23453259 18.23993873
19.25027283 16.29208741 18.23993873 20.36983223 24.44757706 26.07685456
17.32972783 20.59738496 19.48692766 17.22050253 20.81583557 19.33219181
18.49479778 18.57671676 19.63256141 25.35778795 29.26259271 26.95065703
21.48028953 21.86257811 20.57007863 17.04756245 17.99418179 20.21509638
14.47166561 16.31939374 19.60525508 20.98877564 24.5932108 19.92382889
18.9225969 31.31056723 23.42814085 27.36935404 21.26183891 19.27757916
17.58458688 19.63256141 24.09259481 26.87784015 29.99076143 22.58164472
18.0032839 18.83157581 16.24657686 18.89529058 23.73761256 19.58705086
20.53367019 22.17204981 22.42690886 22.54523628 22.48152152 21.21632837
22.05372239 18.79516738 26.55926634 25.57623857 22.69087002 21.46208531
23.4827535 25.67636177 20.07856475 21.0433883 29.10785685 29.7632087
23.73761256 23.62838725 23.96516528 21.86257811 22.20845825 25.63085122
21.42567687 38.77429659 36.50787146 32.83061943 26.55926634 27.05078022
23.62838725 21.18902204 21.46208531 18.58581887 18.44928724 21.09800095
24.25643277 22.02641607 21.71694436 26.45004103 19.15014963 20.77942714
22.25396879 19.28668126 21.54400429 20.1331774 18.77696316 17.49356579
18.75875894 19.97844155 19.58705086 18.63132942 18.84067792 19.81460358
16.41951693 17.14768565 23.86504208 16.63796755 24.11079902 22.90932064
23.32801765 18.32185771 17.73022063 22.99123962 19.41411079 24.07439059
18.64043153 21.31645157 21.52580007 11.0128642 14.50807405 15.09971113
 9.95701956 21.12530728 16.55604857 10.16636806 12.5329164 16.27388319
21.05249041 14.51717616 10.94914944 17.2933194 21.11620517 21.32555368
13.31569777 28.52532188 20.5427723 24.58410869 22.21756036 33.49507338
36.34403349 41.55954194 18.6131252 20.86134612 37.50000134 18.82247371
22.84560588 23.60108092 18.80426949 18.84978003 16.04633047 23.72851045
18.65863574 24.91178461 20.12407529 22.80919744 27.76984683 28.86209991
36.00725546 21.2527368 30.45496898 25.06652047 16.33759795 21.33465578
36.60799466 27.05988233 25.0028057 30.72803225 28.59813875 26.66849165
30.66431749 27.2237203 25.43970694 37.00848745 31.65644737 30.01806775
31.53811995 28.81658937 30.2729268 21.41657477 34.59642857 36.80824105
38.45572278 18.94990323 22.90932064 17.96687546 20.52456809 13.97104962
19.57794875 14.51717616 18.18532608 23.35532398 14.58999303 21.59861695
18.9225969 25.78558708 19.49602977 23.33711976 28.59813875 21.43477898
27.94278691 25.56713646 40.56741206 44.74528008 38.51033543 30.52778586
35.28818885 24.96639727 19.76909304 32.79421099 41.2136618 40.39447199
26.55016423 20.72481448 25.68546388 32.30269711 24.32014753 25.45791115
28.10662487 20.80673346 23.20058813 23.51916194 16.23747476 16.34670006
20.92506088 21.99910974 23.8832463 26.47734736 24.37476018 23.92875684
28.65275141 40.5036973 20.92506088 18.8133716 33.17649957 44.5541358
32.07514438 27.60600887 30.89187022 33.77723876 41.76889045 32.02053173
30.91917654 15.93710516 29.17157162 40.84957744 33.32213331 19.21386439
18.63132942 22.12653927 24.83896774 35.3336994 26.84143172 27.71523418
31.47440519 27.46037513 24.32924964 27.3329456 36.50787146 28.7528746
34.91500238 37.44538868 29.84512768 24.06528848 22.03551818 21.84437389
22.80919744 25.08472469 27.77894894 30.39125422 25.67636177 21.09800095
20.02395209 26.113263   24.93909094 18.03059022 23.08226071 29.41732856
27.86997003 25.31227741 24.44757706 28.88030413 31.19223981 25.54893224

```
 32.86702786 27.66972364 25.72187231 19.68717406 10.59416719 21.05249041
 20.15138162 22.3631941  25.1029289  17.25691096 19.15925174 17.95777335
 23.41903874 20.97057143 23.81953154 23.36442609 20.31521958 17.28421729
 23.71940834 23.86504208 22.78189111 20.69750816 18.74055473 22.9730354
 21.2527368  17.26601307 20.22419849 22.81829955 22.76368689 20.27881114
 18.74965683 18.98631167 20.47905754 19.80550148 19.65076562 31.23775036
 24.85717196 26.27710096 27.89727636 20.06946264 19.01361799 24.63872134
 25.72187231 28.48891344 24.40206651 25.21215421 18.88618847 26.56836845
 16.87462238 19.35949814 21.87168021 23.53736616 21.09800095 20.96146932
 23.56467249 22.22666246 14.13488758 18.14891764 45.24589608 -2.25801069
 10.5031461   0.49082622 10.56686086 26.15877354 29.18977584 21.90808865
 18.80426949  9.98432589  2.99390619 31.8931022  25.84930184 27.16910764
 23.40083452 21.97180341 28.7528746  24.90268251 15.71865454 15.5730208
  5.08739125 13.36120832  7.6723902  10.83992413  9.74767105 14.38974663
 17.32972783 20.40624067 11.16760005 21.69874014 18.9134948  24.22912644
 23.62838725 17.63919954 14.9631795  18.59492098 19.82370569 23.06405649
 23.61928514 14.01656016 15.673144   17.05666456  2.99390619 16.37400639
 16.45592537 27.69702996 17.73022063 25.92211871  7.45393959 12.25075102
  6.46180971 23.89234841 27.05988233 13.60696526 19.55064242 27.44217091
 23.6829999  19.99664576 16.73809075 20.87955034 15.9826157  18.99541378
 18.45838935 21.78065912 21.69874014 23.40083452 23.10956704 27.52408989
 23.81042943 23.91055263 21.83527178 25.66725966 24.13810535 21.32555368
 19.35039603 16.54694646 18.28544928 23.63748936 21.93539498 24.35655597
 18.6131252  24.11990113 23.04585227 22.22666246 21.62592327 23.73761256
 26.75951274 25.90391449 22.64535948 32.62127092 26.56836845 24.72064033
 19.7235825  19.35949814 22.68176791 20.67930394 26.32261151 23.36442609
 22.82740166 24.61141502 21.84437389 17.74842485 19.50513188 19.96933944
 19.26847705 17.32972783 21.46208531 22.02641607 23.91965474 28.86209991
 14.72652466 21.41657477 24.34745386 13.60696526 21.62592327 22.02641607
 22.14474348 26.76861485 29.59937074 17.77573117 18.76786105 22.78189111
 20.97967353 19.07733276 14.97228161 14.60819725 11.68642026 19.78729726
 19.78729726 17.27511518 19.26847705 16.93833715 14.38974663 18.06699866
 20.11497318 16.01902414 20.18779005 25.33958374 21.03428619 28.82569148
 27.16910764 20.21509638]
```

```
In [40]: # Fazendo previsões com o modelo treinado
         RM = 5
         # Xp = np.array(RM)
         Xp = np.array(RM).reshape(-1, 1)
         print ("Se RM = %01.f nosso modelo prevê que a mediana da taxa de ocupação é %0.1f" %
```

```
Se RM = 5 nosso modelo prevê que a mediana da taxa de ocupação é 10.8
```

3.5.1 Comparação StatsModels x ScikitLearn

```
In [41]: from sklearn.datasets import make_regression
         HX, Hy = make_regression(n_samples = 10000000, n_features = 1, n_targets = 1, random_s
```

```
In [42]: %%time
         sk_linear_regression = linear_model.LinearRegression(normalize=False, fit_intercept=T
         sk_linear_regression.fit(HX,Hy)
```

```
CPU times: user 749 ms, sys: 185 ms, total: 933 ms
Wall time: 247 ms
```

```
In [43]: %%time
         sm_linear_regression = sm.OLS(Hy, sm.add_constant(HX))
         sm_linear_regression.fit()
```

```
CPU times: user 2.39 s, sys: 559 ms, total: 2.95 s
Wall time: 1.18 s
```

3.6 Cost Function de um Modelo de Regressão Linear

O objetivo da regressão linear é buscar a equação de uma linha de regressão que minimize a soma dos erros ao quadrado, da diferença entre o valor observado de y e o valor previsto.

Existem alguns métodos para minimização da Cost Function tais como: Pseudo-inversão, Fatorização e Gradient Descent.

```
In [44]: from IPython.display import Image
         Image('imagens/formula-regressao.png')
```

Out[44]:

$$y \approx h(X) = \beta X + \beta_0$$

```
In [45]: from IPython.display import Image
         Image('imagens/formulas.png')
```

Out[45]:

$$y_i - f(x_i)$$

$$f(x_i) - y_i$$

$$| f(x_i) - y_i |$$

$$(y_i - f(x_i))^2$$

$$\text{Mean absolute error (MAE)} = \sum_{i=1}^{n} |f(x_i) - y_i|$$

$$\text{SSE/MSE} = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} (y_i - f(x_i))^2}$$

Total de Vendas (R$)	Total de Vendas Previsto (R$)
1245900	1278450
1302763	1334789
1345119	1320876

Variável Resposta (y)

Previsão f(x)

3.7 Por Que Usamos o Erro ao Quadrado?

```
In [46]: # Definindo 2 conjuntos de dados
         import numpy as np
         x = np.array([9.5, 8.5, 8.0, 7.0, 6.0])

In [47]: # Função para cálculo da Cost Function
         def squared_cost(v, e):
             return np.sum((v - e) ** 2)

In [48]: # A função fmin() tenta descobrir o valor do somatório mínimo dos quadrados
         from scipy.optimize import fmin
         xopt = fmin(squared_cost, x0 = 0, xtol = 1e-8, args = (x,))

Optimization terminated successfully.
         Current function value: 7.300000
         Iterations: 44
         Function evaluations: 88


In [49]: print ('Resultado da Otimização: %0.1f' % (xopt[0]))
         print ('Média: %0.1f' % (np.mean(x)))
         print ('Mediana: %0.1f' % (np.median(x)))

Resultado da Otimização: 7.8
Média: 7.8
Mediana: 8.0
```

```
In [50]: def absolute_cost(v,e):
             return np.sum(np.abs(v - e))

In [51]: xopt = fmin(absolute_cost, x0 = 0, xtol = 1e-8, args = (x,))

Optimization terminated successfully.
         Current function value: 5.000000
         Iterations: 44
         Function evaluations: 88


In [52]: print ('Resultado da Otimização: %0.1f' % (xopt[0]))
         print ('Média %0.1f' % (np.mean(x)))
         print ('Mediana %0.1f' % (np.median(x)))

Resultado da Otimização: 8.0
Média 7.8
Mediana 8.0
```

3.8 Minimizando a Cost Function

Minimizando a Cost Function com Pseudo-Inversão

```
In [53]: from IPython.display import Image
         Image('imagens/formula-cost-function.png')

Out[53]:
```

$$\frac{1}{2n} * \sum (h(X) - y)^2$$

```
In [54]: from IPython.display import Image
         Image('imagens/formula-pseudo.png')

Out[54]:
```

$$w = (X^T X)^{-1} X^T y$$

```
In [55]: # Definindo x e y
         num_observ = len(dataset)
         X  = dataset['RM'].values.reshape((num_observ, 1)) # X deve ser uma matriz
         Xb = np.column_stack((X, np.ones(num_observ)))
         y  = dataset['target'].values # y pode ser um vetor
```

In [56]:
```
# Funções para matriz inversa e equações normais
def matriz_inversa(X, y, pseudo = False):
    if pseudo:
        return np.dot(np.linalg.pinv(np.dot(X.T, X)), np.dot(X.T,y))
    else:
        return np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T,y))

def normal_equations(X,y):
    return np.linalg.solve(np.dot(X.T,X), np.dot(X.T,y))
```

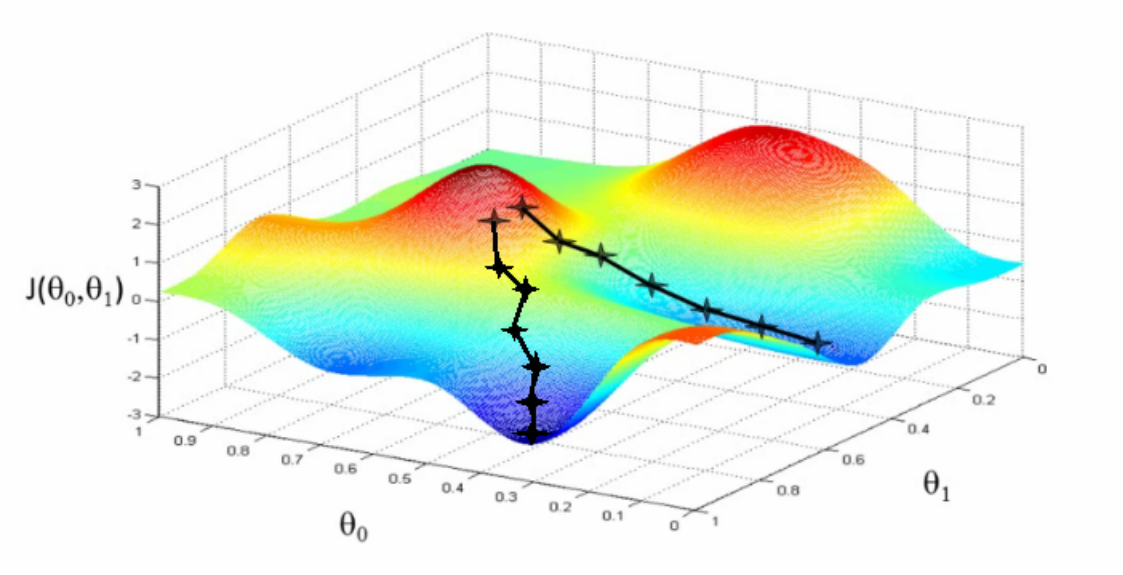
In [57]:
```
# Imprime os valores
print (matriz_inversa(Xb, y))
print (matriz_inversa(Xb, y, pseudo = True))
print (normal_equations(Xb, y))
```

```
[  9.10210898 -34.67062078]
[  9.10210898 -34.67062078]
[  9.10210898 -34.67062078]
```

3.9 Aplicando o Gradiente Descendente

In [58]:
```
from IPython.display import Image
Image('imagens/gradient-descent.png')
```

Out[58]:



In [59]:
```
from IPython.display import Image
Image('imagens/formula-gradient-descent1.png')
```

Out[59]:

$$J(w) = \frac{1}{2n}\sum(Xw - y)^2$$

In [60]:
```
# Alfa é chamado de taxa de aprendizagem
from IPython.display import Image
Image('imagens/formula-gradient-descent2.png')
```

Out[60]:

$$w_j = w_j - \alpha * \frac{\partial}{\partial w} J(w)$$

In [61]:
```
# Definindo x e y
observations = len(dataset)
X = dataset['RM'].values.reshape((observations,1))
X = np.column_stack((X,np.ones(observations)))
y = dataset['target'].values
```

In [62]:
```
import random

# Valores randômicos para os coeficientes iniciais
def random_w( p ):
    return np.array([np.random.normal() for j in range(p)])

# Cálculo da hipótese (valor aproximado de y)
def hypothesis(X,w):
    return np.dot(X,w)

# Cálculo da função de perda (loss)
def loss(X,w,y):
    return hypothesis(X,w) - y

# Cálculo do erro
def squared_loss(X,w,y):
    return loss(X,w,y)**2

# Cálculo do gradiente
def gradient(X,w,y):
    gradients = list()
    n = float(len( y ))
    for j in range(len(w)):
```

```
            gradients.append(np.sum(loss(X,w,y) * X[:,j]) / n)
        return gradients

    # Atualização do valor dos coeficientes
    def update(X,w,y, alpha = 0.01):
        return [t - alpha*g for t, g in zip(w, gradient(X,w,y))]

    # Otimização do modelo
    def optimize(X,y, alpha = 0.01, eta = 10**-12, iterations = 1000):
        w = random_w(X.shape[1])
        path = list()
        for k in range(iterations):
            SSL = np.sum(squared_loss(X,w,y))
            new_w = update(X,w,y, alpha = alpha)
            new_SSL = np.sum(squared_loss(X,new_w,y))
            w = new_w
            if k >= 5 and (new_SSL - SSL <= eta and new_SSL - SSL >= -eta):
                path.append(new_SSL)
                return w, path
            if k % (iterations / 20) == 0:
                path.append(new_SSL)
        return w, path
```

In [63]:
```
# Definindo o valor de alfa
# Alfa é chamado de taxa de aprendizagem
alpha = 0.048

# Otimizando a Cost Function
w, path = optimize(X, y, alpha, eta = 10**-12, iterations = 25000)
```

In [64]:
```
# Imprimindo o resultado
print ("Valor Final dos Coeficientes: %s" % w)
```

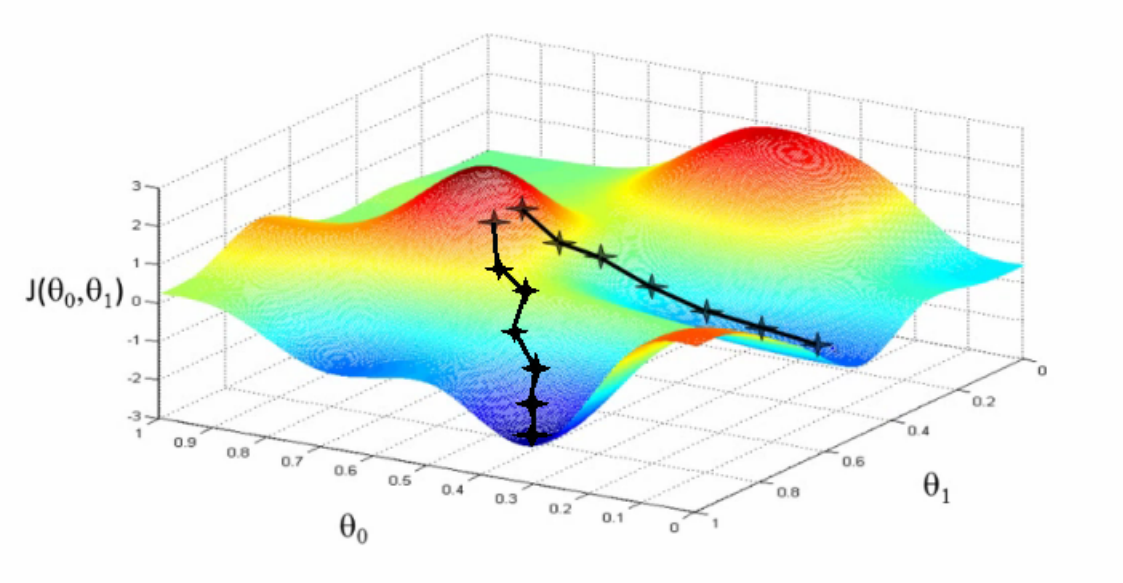
```
Valor Final dos Coeficientes: [9.102103100082088, -34.670583366078766]
```

In [65]:
```
# Imprimindo o resultado
print ("Percorrendo o Caminho do Gradiente em que o erro ao quadrado era:\n\n %s" % pa
```

```
Percorrendo o Caminho do Gradiente em que o erro ao quadrado era:

 [238079.85272886304, 23946.574058465696, 22506.966795273613, 22166.990625550396, 22086.702203
```

In [66]:
```
from IPython.display import Image
Image('imagens/gradient-descent.png')
```

Out[66]:



4 Fim

4.0.1 Obrigado - Data Science Academy - facebook.com/dsacademybr