MACHINE LEARNING - REGRESSÃO LINEAR

- Regressão linear para dados do mercado financeiro, o nosso objetivo é criar um modelo que consiga predizer o valor de fechamento para uma ação. O algoritmo será treinado e validado!

```
# Instalando as Libs
!pip install pandas
!pip install scikit-learn
!pip install matplotlib
!pip install plotly
```

```
Requirement already satisfied: pandas in c:\users\lucas\anaconda3\lib\
site-packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in c:\users\lucas\
anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\
lucas\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\lucas\
anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\lucas\
anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\lucas\anaconda3\
lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: scikit-learn in c:\users\lucas\
anaconda3\lib\site-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\lucas\
anaconda3\lib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.2 in c:\users\lucas\
anaconda3\lib\site-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in c:\users\lucas\
anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lucas\
anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: matplotlib in c:\users\lucas\anaconda3\
lib\site-packages (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy<2,>=1.21 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (10.2.0)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lucas\
anaconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\lucas\anaconda3\
lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: plotly in c:\users\lucas\anaconda3\lib\
site-packages (5.9.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\lucas\
anaconda3\lib\site-packages (from plotly) (8.2.2)
```

```
# Bibliotecas e Imports
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import plotly.offline as py
import plotly
plotly.offline.init_notebook_mode()
import datetime
import os
import numpy as np
```

```
# lendo csv
dataset = pd.read_csv('C:/Users/lucas/Documents/Regressão
Linear/petr4_1_2010_11_2017.csv')
```

```
# Transformar minha coluna 'Date' que está em string em DateTime
# dataset['Date'] = pd.to_datetime(dataset['Date']) -- Aqui quebrou pq
precisei passar o formato da minha data
```

```
# Especificando o formato das datas pra conseguir converter a string
dataset['Date'] = pd.to_datetime(dataset['Date'], format='%Y-%m-%d',
errors='raise')
```

```
# Visualizando os Dados
dataset.head()
```

```
        Date   Open   High    Low  Close    Volume
0 2017-04-11  14.97  14.99  14.55  14.68  38392300
1 2017-04-10  14.90  14.94  14.70  14.94  37541700
2 2017-04-07  14.61  14.90  14.60  14.70  32944900
3 2017-04-06  14.62  14.87  14.42  14.53  34386000
4 2017-04-05  15.05  15.16  14.50  14.57  49623400
```

```
# Variação entre o preço de abertura e fechamento - Criando coluna
dataset['Variation'] = dataset['Close'].sub(dataset['Open'])
```

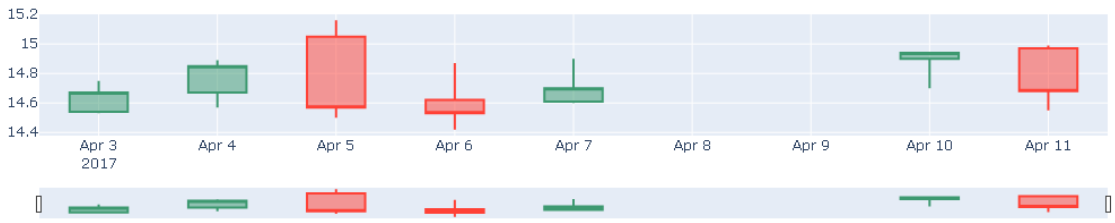
```
# Vendo minha coluna
dataset.head()
```

	Date	Open	High	Low	Close	Volume	Variation
0	2017-04-11	14.97	14.99	14.55	14.68	38392300	-0.29
1	2017-04-10	14.90	14.94	14.70	14.94	37541700	0.04
2	2017-04-07	14.61	14.90	14.60	14.70	32944900	0.09
3	2017-04-06	14.62	14.87	14.42	14.53	34386000	-0.09
4	2017-04-05	15.05	15.16	14.50	14.57	49623400	-0.48

```
# PLota o valor dos preços no periodo analisado (2010 a 2017)
# Utiliza a Biblioteca pyplot para plotar os dados financeiros
temporais
x1 = dataset.Date
y1 = dataset.Close
data = [go.Scatter(x= x1, y= y1)] #
layout = go.Layout(
    xaxis= dict(
        range = ['2010-01-01', '2017-04-11'],
        title = 'Ano'
    ),
    yaxis = dict(
        range = [min(x1), max(y1)],
        title = 'Valor da Acao'
    ))
fig = go.Figure(data = data, layout = layout)
py.iplot(fig)
```

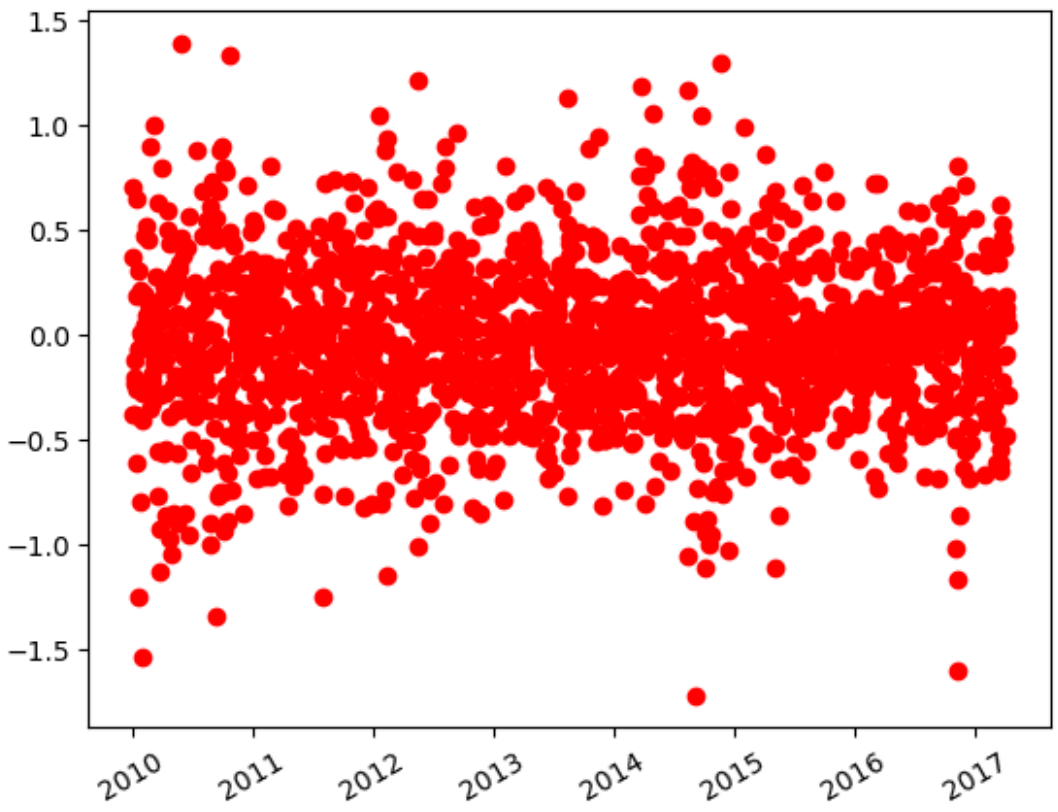

```
dataset2 = dataset.head(7) #7 primeiras linhas
dados = go.Candlestick(x=dataset2.Date,
                       open=dataset2.Open,
                       high=dataset2.High,
                       low=dataset2.Low,
                       close=dataset2.Close,
                       )
```

```
data=[dados]
py.offline.iplot(data,filename='grafico_candlestick')
```

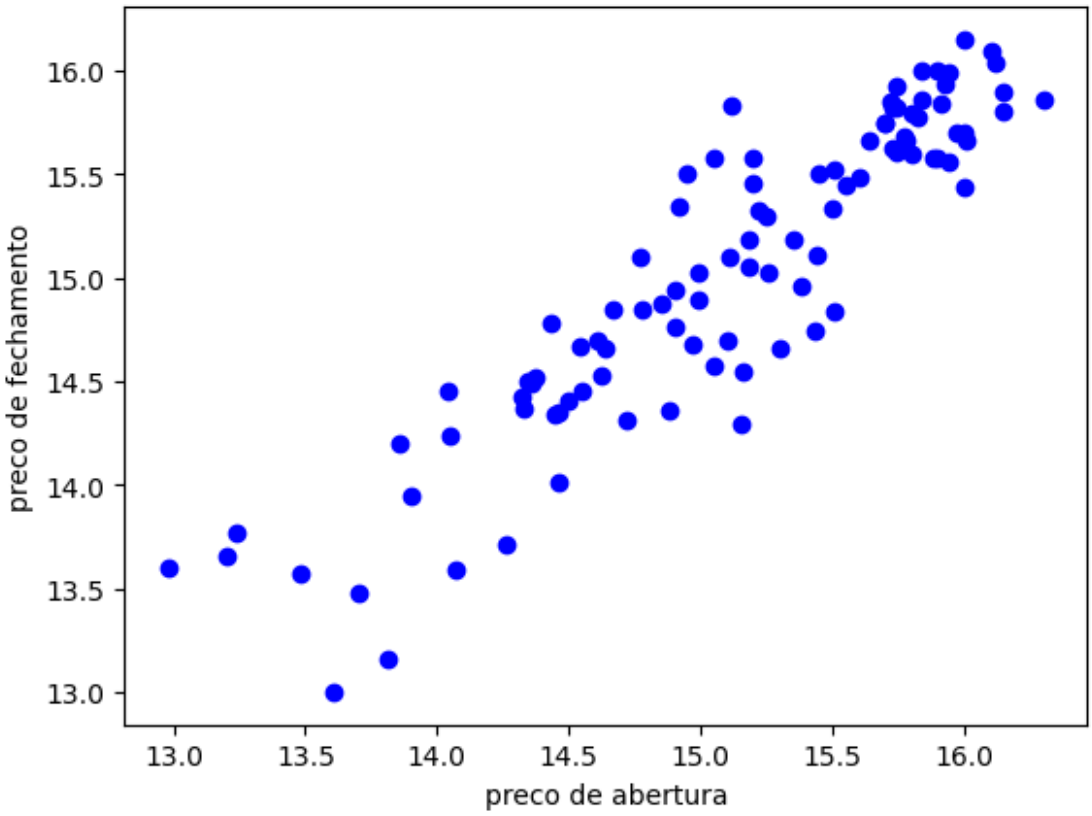



```
dataset2 = dataset.head(180)
dados = go.Candlestick(x=dataset2.Date,
                       open=dataset2.Open,
                       high=dataset2.High,
                       low=dataset2.Low,
                       close=dataset2.Close,
                       )

data=[dados]
py.offline.iplot(data,filename='grafico_candlestick')
```

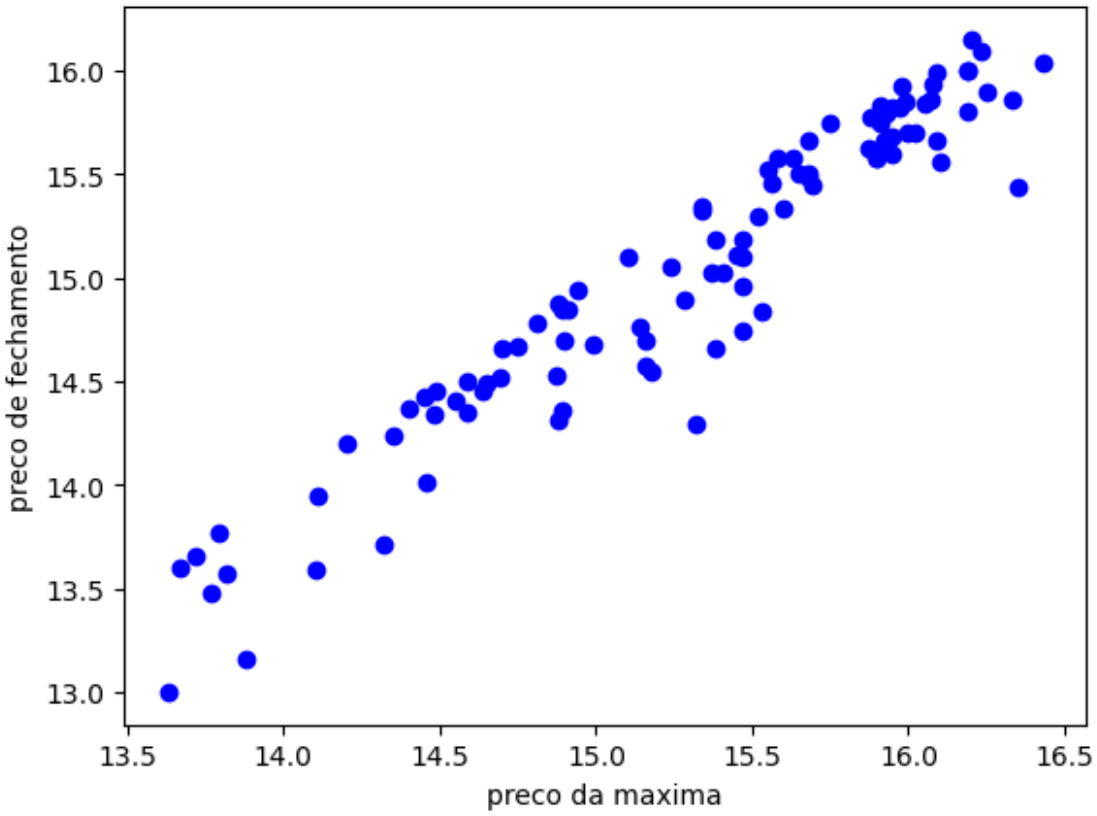

```
import matplotlib.dates as mdates
import datetime as dt
x = dataset['Date']
y = dataset['Variation']
plt.plot_date(x,y, color='r')
plt.xticks(rotation=30)
plt.show()
```



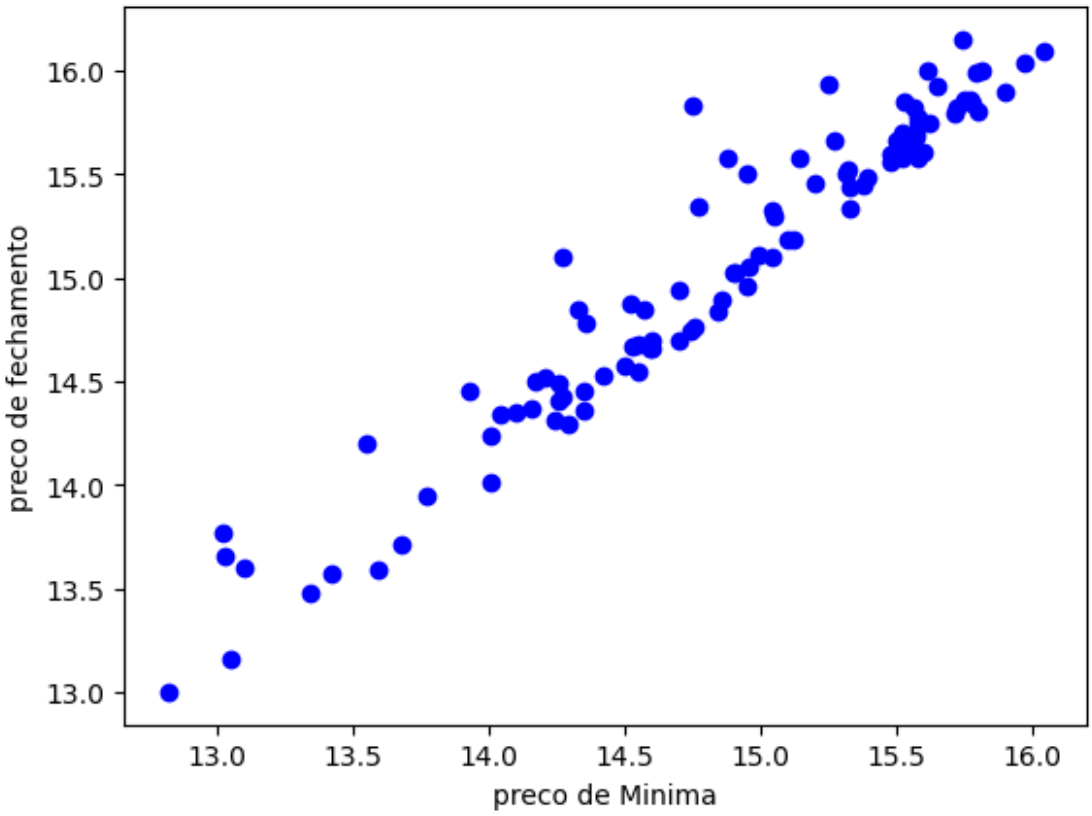

```
# definindo variavel de treino
treino = dataset

# PLota a dispersao entre o preço de abertura e fechamento dos ultimos
100d
x = treino.Open[:100]
y = treino.Close[:100]
plt.scatter(x,y,color='b')
plt.xlabel('preco de abertura')
plt.ylabel('preco de fechamento')
plt.axis([min(x),max(x),min(y),max(y)])
plt.autoscale('False')
plt.show()
```



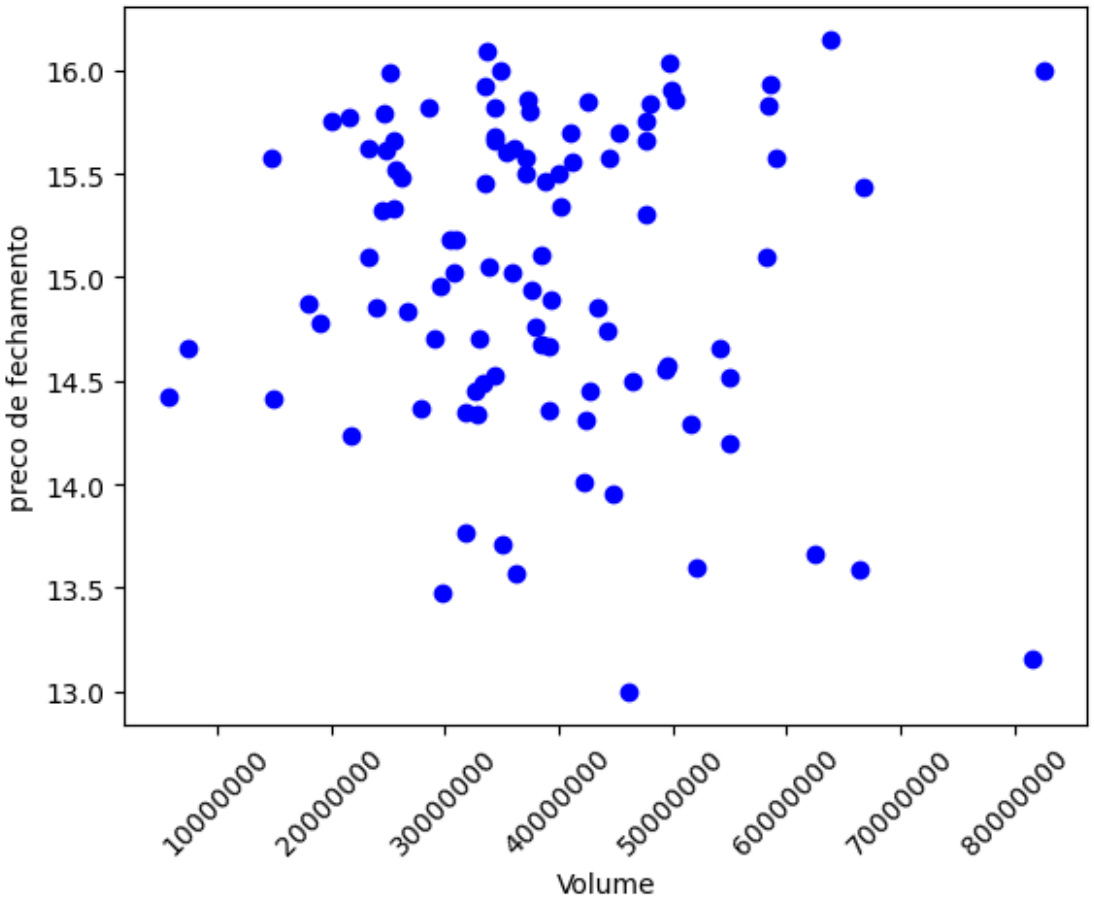

```
# PLota a dispersao entre o preço de maxima e fechamento dos ultimos
100d
x = treino.High[:100]
y = treino.Close[:100]
plt.scatter(x,y,color='b')
plt.xlabel('preco da maxima')
plt.ylabel('preco de fechamento')
plt.axis([min(x),max(x),min(y),max(y)])
plt.autoscale('False')
plt.show()
```




```
# PLota a dispersao entre o preço de minima e fechamento dos ultimos
100d
x = treino.Low[:100]
y = treino.Close[:100]
plt.scatter(x,y,color='b')
plt.xlabel('preco de Minima')
plt.ylabel('preco de fechamento')
plt.axis([min(x),max(x),min(y),max(y)])
plt.autoscale('False')
plt.show()
```




```
# PLota a dispersao entre o preço de abertura e fechamento dos ultimos
100d
x = treino.Volume[:100]
y = treino.Close[:100]
plt.scatter(x,y,color='b')
plt.xlabel('Volume')
plt.ylabel('preco de fechamento')
plt.axis([min(x),max(x),min(y),max(y)])
plt.ticklabel_format(style='plain', axis='x')
plt.autoscale('False')
plt.xticks(rotation=45)
plt.show()
```




```
# Selecionar as colunas Open, High, Low e Volume do DataFrame treino e exibir as primeiras 5 linhas desse DataFrame.
features = ['Open','High','Low','Volume']
treino = treino[features]

treino.head()
```

	Open	High	Low	Volume
0	14.97	14.99	14.55	38392300
1	14.90	14.94	14.70	37541700
2	14.61	14.90	14.60	32944900
3	14.62	14.87	14.42	34386000
4	15.05	15.16	14.50	49623400

```
# Selecionar a coluna Close do DataFrame dataset como a variável alvo y
y = dataset['Close']

# Dividir os dados em conjuntos de treino e teste. X_treino e y_treino são usados para treinar o modelo:
# enquanto X_teste e y_teste são usados para testar o modelo.
```

```
# como não setei o test_size/train_size,a função trains_test_split
divide por padrão 25% dos dados serão usados para teste e 75% para
treino.
X_treino, X_teste, y_treino, y_teste = train_test_split(
treino, y,random_state=42) ### Um parâmetro que assegura que a divisão
dos dados seja a mesma toda vez que você executar o código.
                            ### O valor 42 é arbitrário e pode ser
qualquer inteiro. Ele é usado para garantir a reprodutibilidade dos
resultados

X_treino.head()
```

	Open	High	Low	Volume
585	14.20	14.62	13.33	54167400
163	12.19	12.37	12.07	53317800
1593	27.35	27.64	27.21	32455000
1133	21.44	21.54	21.14	22808700
266	8.09	8.17	7.72	76508100

```
#Criar e treinar um modelo de Regressão Linear usando os dados de
treino
lr_model = LinearRegression()

lr_model.fit(X_treino,y_treino)
```

```
LinearRegression()
```

```
# definindo os coeficientes (pesos) para cara feature
# Esses valores indicam a influência de cada feature na variável-alvo.
lr_model.coef_
```

```
array([-6.88569831e-01,  7.56540314e-01,  9.33134404e-01,
7.75092990e-10])
```

```
# Usar o modelo treinado para fazer previsões sobre os dados de teste
e exibir as 10 primeiras previsões.
lr_model.predict(X_teste)[:10]
```

```
array([36.40282357,  6.69755742, 17.56221729, 12.37534074,
23.61722772,
       14.66679251,  9.58837484, 15.85914839, 28.61967737,
18.67014872])
```

```
# Exibir os primeiros 10 valores da variável alvo y_teste (Close)
y_teste[:10]
```

```
1791    36.55
322      6.79
1033    17.74
162     12.49
1273    23.88
70      14.78
```

```
247      9.69
736     16.03
1518    28.71
1006    18.53
Name: Close, dtype: float64
```

```
# Valores reais
valores_reais = y_teste[:10]

# previsoes
previsoes = lr_model.predict(X_teste)

# Índices para as amostras
indices = np.arange(len(valores_reais))

# Certifique-se de que valores_reais e previsoes têm o mesmo
comprimento
if len(valores_reais) == len(previsoes):
    # Criando o gráfico
    plt.figure(figsize=(10, 5))
    plt.plot(indices, valores_reais, 'o-', label='Valores Reais',
color='blue')
    plt.plot(indices, previsoes, 's-', label='Previsões', color='red')

    # Adicionando títulos e rótulos
    plt.xlabel('Índice da Amostra')
    plt.ylabel('Valor')
    plt.title('Comparação de Valores Reais e Previsões')
    plt.legend()
    plt.grid(True)
    plt.show()
else:
    print("Os valores reais e previsões não têm o mesmo comprimento.")
```

```
Os valores reais e previsões não têm o mesmo comprimento.
```

```
#Calcular a raiz do erro quadrático médio (Root Mean Squared Error,
RMSE) das previsões do primeiro modelo de Regressão Linear.

RMSE = mean_squared_error(y_teste, lr_model.predict(X_teste))**0.5
#Calcula a raiz quadrada do erro quadrático médio para obter o RMSE
RMSE ## responde a precisao do meu modelo
```

```
0.15640353105621313
```

```
lr_model2 = LinearRegression()

# visando melhorar o modelo, testaremos somente com 2 variaveis
features = ['Open','High']
treino2 = treino[features]

treino2.head()
```

```
    Open   High
0  14.97  14.99
1  14.90  14.94
2  14.61  14.90
3  14.62  14.87
4  15.05  15.16
```

```
# Dividir os dados em conjuntos de treino e teste, com 33% dos dados
reservados para o teste (test_size=0.33) e 67% para o treino,
# garantindo reprodutibilidade (random_state=42).

X_treino, X_teste, y_treino, y_teste = train_test_split(
treino2, y, test_size=0.33, random_state=42)

lr_model2.fit(X_treino,y_treino)
```

```
LinearRegression()
```

```
lr_model2.coef_
```

```
array([-0.15946514,  1.15146638])
```

```
# Concluimos que pelo primeiro modelo apresentar uma menor raiz do
erro quadrático (Root Mean Squared Error) RMS E
# o meu primeiro modelo usando as 4 variaveis é melhor!

RMSE = mean_squared_error(y_teste, lr_model2.predict(X_teste))**0.5
RMSE
```

```
0.2466464764308431
```