stock-price-prediction

October 29, 2023

0.0.1 Stock Price Prediction Project :

[ ]:

[1]:
```
import numpy as np
import pandas as pd
import yfinance as yf
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

[27]:
```
# Define the stock symbol and date range

stock_symbol = 'AAPL'
start_date = '2020-01-01'
end_date = '2021-12-31'
```

[28]:
```
data = pd.read_csv("E:\\Dataset\\NFLX.csv")
```

[29]:
```
data
```

[29]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	
…	…	…	…	…	…	…	
1004	2022-01-31	401.970001	427.700012	398.200012	427.140015	427.140015	
1005	2022-02-01	432.959991	458.480011	425.540009	457.130005	457.130005	
1006	2022-02-02	448.250000	451.980011	426.480011	429.480011	429.480011	
1007	2022-02-03	421.440002	429.260010	404.279999	405.600006	405.600006	
1008	2022-02-04	407.309998	412.769989	396.640015	410.170013	410.170013	

```
3      9306700
4     16906900
…           …
1004  20047500
1005  22542300
1006  14346000
1007   9905200
1008   7782400

[1009 rows x 7 columns]
```

[30]:
```
data.shape
```

[30]:
```
(1009, 7)
```

[55]:
```
# Define the stock symbol and date range
stock_symbol = 'AAPL'
start_date = '2020-01-01'
end_date = '2021-12-31'
```

[ ]:
```

```

[56]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1008 entries, 1 to 1008
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          1008 non-null   object
 1   Open          1008 non-null   float64
 2   High          1008 non-null   float64
 3   Low           1008 non-null   float64
 4   Close         1008 non-null   float64
 5   Adj Close     1008 non-null   float64
 6   Volume        1008 non-null   int64
 7   Daily_Return  1008 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 70.9+ KB
```

[57]:
```
data.isna().sum()
```

[57]:
```
Date            0
Open            0
High            0
Low             0
Close           0
```

```
Adj Close       0
Volume          0
Daily_Return    0
dtype: int64
```

[58]:
```
data.describe()
```

[58]:
```
              Open         High          Low        Close    Adj Close  \
count  1008.000000  1008.000000  1008.000000  1008.000000  1008.000000
mean    419.215486   425.476874   412.535099   419.164166   419.164166
std     108.478448   109.204470   107.487460   108.219182   108.219182
min     233.919998   250.649994   231.229996   233.880005   233.880005
25%     331.497498   336.344986   326.007508   331.770004   331.770004
50%     377.884995   383.070007   371.440002   378.740005   378.740005
75%     509.400001   515.832504   502.572502   509.087487   509.087487
max     692.349976   700.989990   686.090027   691.690002   691.690002

             Volume  Daily_Return
count  1.008000e+03   1008.000000
mean   7.566394e+06      0.000831
std    5.466548e+06      0.026603
min    1.144000e+06     -0.217905
25%    4.091625e+06     -0.011933
50%    5.931500e+06      0.000673
75%    9.310625e+06      0.014544
max    5.890430e+07      0.168543
```

[59]:
```
# Calculate daily returns
data['Daily_Return'] = data['Adj Close'].pct_change()
```

[60]:
```
data
```

[60]:
```
            Date        Open        High         Low       Close   Adj Close  \
1     2018-02-06  247.699997  266.700012  245.000000  265.720001  265.720001
2     2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998
3     2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006
4     2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001
5     2018-02-12  252.139999  259.149994  249.000000  257.950012  257.950012
…            …           …           …           …           …           …
1004  2022-01-31  401.970001  427.700012  398.200012  427.140015  427.140015
1005  2022-02-01  432.959991  458.480011  425.540009  457.130005  457.130005
1006  2022-02-02  448.250000  451.980011  426.480011  429.480011  429.480011
1007  2022-02-03  421.440002  429.260010  404.279999  405.600006  405.600006
1008  2022-02-04  407.309998  412.769989  396.640015  410.170013  410.170013

        Volume  Daily_Return
1     12595800           NaN
```

```
2      8981500     -0.004366
3      9306700     -0.054657
4     16906900     -0.002519
5      8534900      0.033992
…        …            …
1004  20047500      0.111302
1005  22542300      0.070211
1006  14346000     -0.060486
1007   9905200     -0.055602
1008   7782400      0.011267

[1008 rows x 8 columns]
```

[61]:
```
# Drop missing values
data.dropna(inplace=True)
```

[62]:
```
data
```

[62]:
```
            Date        Open        High         Low       Close   Adj Close  \
2     2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998
3     2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006
4     2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001
5     2018-02-12  252.139999  259.149994  249.000000  257.950012  257.950012
6     2018-02-13  257.290009  261.410004  254.699997  258.269989  258.269989
…            …           …           …           …           …           …
1004  2022-01-31  401.970001  427.700012  398.200012  427.140015  427.140015
1005  2022-02-01  432.959991  458.480011  425.540009  457.130005  457.130005
1006  2022-02-02  448.250000  451.980011  426.480011  429.480011  429.480011
1007  2022-02-03  421.440002  429.260010  404.279999  405.600006  405.600006
1008  2022-02-04  407.309998  412.769989  396.640015  410.170013  410.170013

         Volume  Daily_Return
2      8981500     -0.004366
3      9306700     -0.054657
4     16906900     -0.002519
5      8534900      0.033992
6      6855200      0.001240
…        …            …
1004  20047500      0.111302
1005  22542300      0.070211
1006  14346000     -0.060486
1007   9905200     -0.055602
1008   7782400      0.011267

[1007 rows x 8 columns]
```

[63]:
```
# Prepare the data for prediction
X = np.array(data['Open']).reshape(-1, 1)
y = np.array(data['Close'])
```

[64]:
```
X
```

[64]:
```
array([[266.579987],
       [267.079987],
       [253.850006],
       …,
       [448.25    ],
       [421.440002],
       [407.309998]])
```

[65]:
```
X.shape
```

[65]:
```
(1007, 1)
```

[66]:
```
y.shape
```

[66]:
```
(1007,)
```

[67]:
```
# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[68]:
```
X_train.shape
```

[68]:
```
(805, 1)
```

[69]:
```
X_test.shape
```

[69]:
```
(202, 1)
```

[70]:
```
y_train.shape
```

[70]:
```
(805,)
```

[71]:
```
y_test.shape
```

[71]:
```
(202,)
```

[72]:
```
# Create a linear regression model

model = LinearRegression()
```

[73]:
```
# Train the model on the training data

model.fit(X_train, y_train)
```

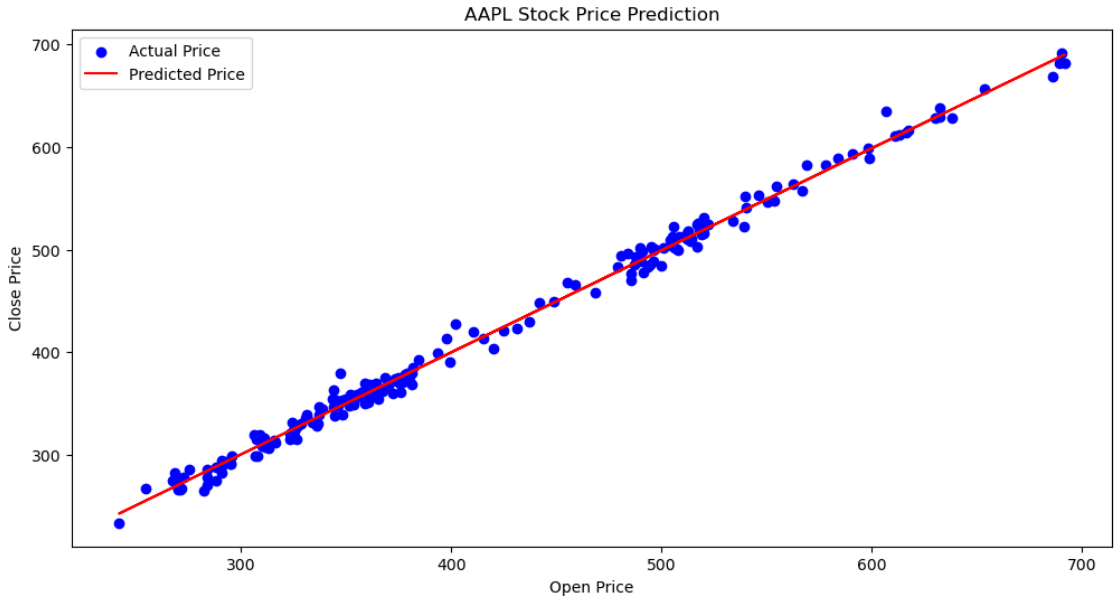
[73]:
```
LinearRegression()
```

[74]:
```
# Make predictions on the test data

y_pred = model.predict(X_test)
```

[75]:
```
y_pred.shape
```

[75]:
```
(202,)
```

[76]:
```
# Visualize the actual and predicted prices
plt.figure(figsize=(12, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Price')
plt.plot(X_test, y_pred, color='red', label='Predicted Price')
plt.title(f'{stock_symbol} Stock Price Prediction')
plt.xlabel('Open Price')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[77]:
```
# Evaluate the model (you may use other evaluation metrics)

mse = np.mean((y_pred - y_test) ** 2)
```

```
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 65.85398867790825
```

[79]:
```
# Predict the stock price for a specific date
predicted_date = '2021-12-31'

if predicted_date in data.index:
    predicted_open_price = model.predict(np.array(data.
 ↪loc[predicted_date]['Open']).reshape(1, -1))
    print(f'Predicted Open Price on {predicted_date}: {predicted_open_price[0]:.
 ↪2f}')
else:
    print(f'Data for {predicted_date} not found in the dataset.')
```

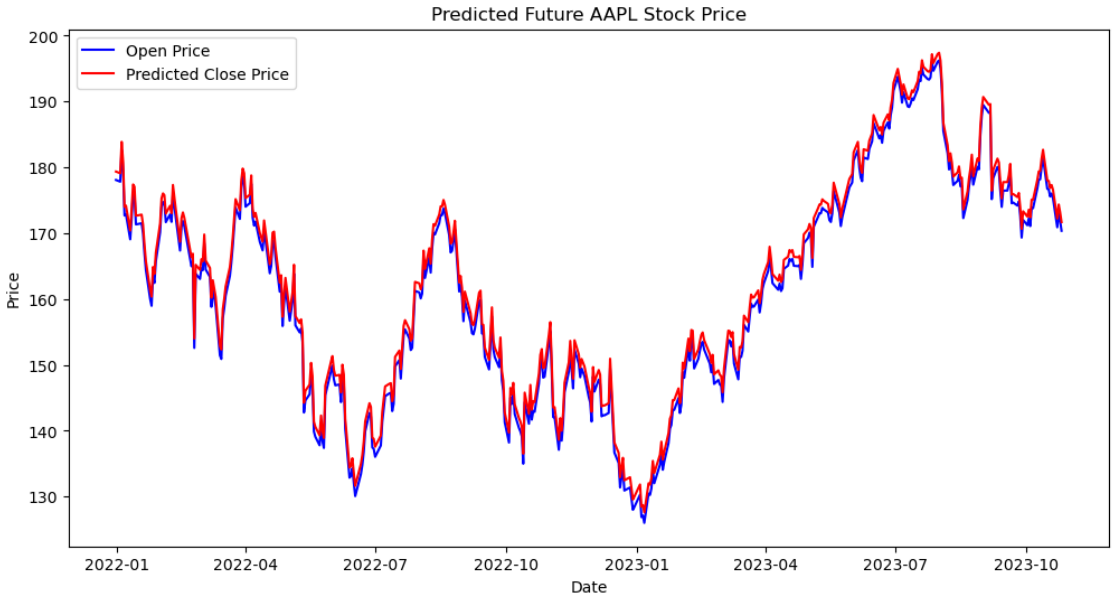
```
Data for 2021-12-31 not found in the dataset.
```

[80]:
```
# Close the model

yf.pdr_override()
```

[81]:
```
# Predict the future stock price

future_date = '2023-12-31'
future_data = yf.download(stock_symbol, start=predicted_date, end=future_date)
future_data['Predicted_Close'] = model.predict(np.array(future_data['Open']).
 ↪reshape(-1, 1))
print(future_data[['Open', 'Predicted_Close']])
```

```
[**********************100%**********************]  1 of 1 completed
                  Open  Predicted_Close
Date
2021-12-31  178.089996       179.367469
2022-01-03  177.830002       179.109012
2022-01-04  182.630005       183.880613
2022-01-05  179.610001       180.878479
2022-01-06  172.699997       174.009363
…                  …              …
2023-10-20  175.309998       176.603920
2023-10-23  170.910004       172.229961
2023-10-24  173.050003       174.357298
2023-10-25  171.880005       173.194223
2023-10-26  170.369995       171.693148

[458 rows x 2 columns]
```

[82]:
```
# Visualize the predicted future prices

plt.figure(figsize=(12, 6))
plt.plot(future_data.index, future_data['Open'], label='Open Price',␣
 ↪color='blue')
plt.plot(future_data.index, future_data['Predicted_Close'], label='Predicted␣
 ↪Close Price', color='red')
plt.title(f'Predicted Future {stock_symbol} Stock Price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



[ ]: