```
# Import required libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Take input for stock symbol from user
symbol = input("Enter stock symbol: ")
```

```
Enter stock symbol: MSFT
```

```
# Set start and end dates for historical data
end = pd.Timestamp.today()
start = end - pd.Timedelta(days=365) # 1 years data
```

Data

```
# Get historical data for chosen stock from Yahoo Finance using yfinance
data_frame = yf.download(symbol, start=start, end=end)
data_frame
```

```
[*********************100%***********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2022-05-02	277.709991	284.940002	276.220001	284.470001	281.706390	35151100
2022-05-03	283.959991	284.130005	280.149994	281.779999	279.042511	25978600
2022-05-04	282.589996	290.880005	276.730011	289.980011	287.162842	33599300
2022-05-05	285.540009	286.350006	274.339996	277.350006	274.655548	43260400
2022-05-06	274.809998	279.250000	271.269989	274.730011	272.060974	37780300
...	...	...	...	...	...	...
2023-04-26	296.700012	299.570007	292.730011	295.369995	295.369995	64599200
2023-04-27	295.970001	305.200012	295.250000	304.829987	304.829987	46462600
2023-04-28	304.010010	308.929993	303.309998	307.260010	307.260010	36446700
2023-05-01	306.970001	308.600006	305.149994	305.559998	305.559998	21275000
2023-05-02	307.760010	309.165009	303.910004	306.369995	306.369995	16761952

252 rows × 6 columns

Correlation

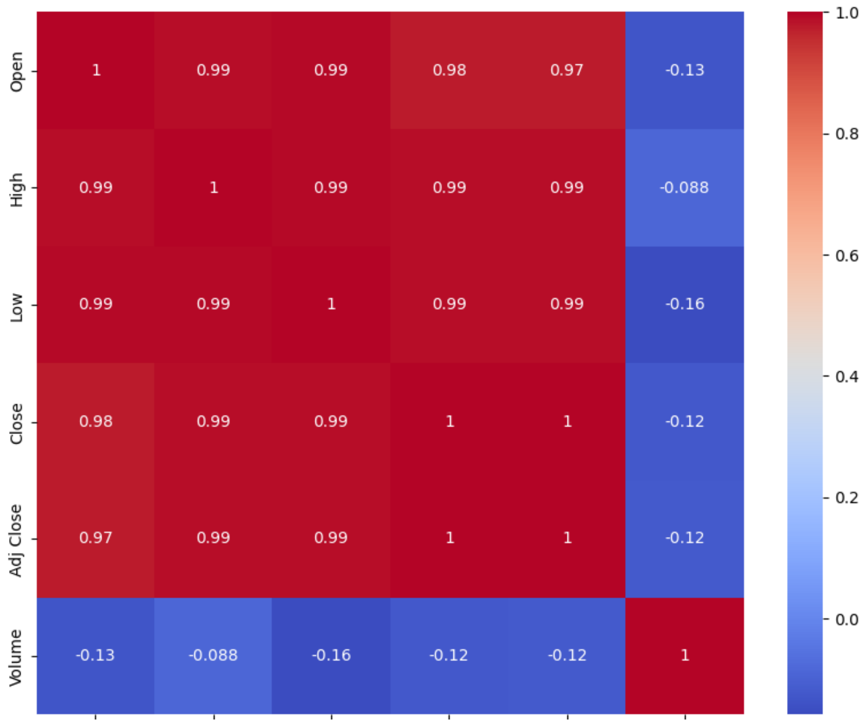
```
# Calculate the correlation matrix
corr_matrix = data_frame.corr()

# Display the correlation matrix
print(corr_matrix)
```

```
               Open      High       Low     Close  Adj Close    Volume
Open       1.000000  0.990552  0.991310  0.975559   0.974627 -0.132010
High       0.990552  1.000000  0.991037  0.990503   0.989050 -0.088120
Low        0.991310  0.991037  1.000000  0.989799   0.989676 -0.158653
Close      0.975559  0.990503  0.989799  1.000000   0.999137 -0.122861
Adj Close  0.974627  0.989050  0.989676  0.999137   1.000000 -0.121544
Volume    -0.132010 -0.088120 -0.158653 -0.122861  -0.121544  1.000000
```

```
# Plot the correlation matrix as a heatmap
f,ax=plt.subplots(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

```
<Axes: >
```




```
# Numeric Values
numeric_feature=data_frame.select_dtypes(include=[np.number])
numeric_feature.columns
```

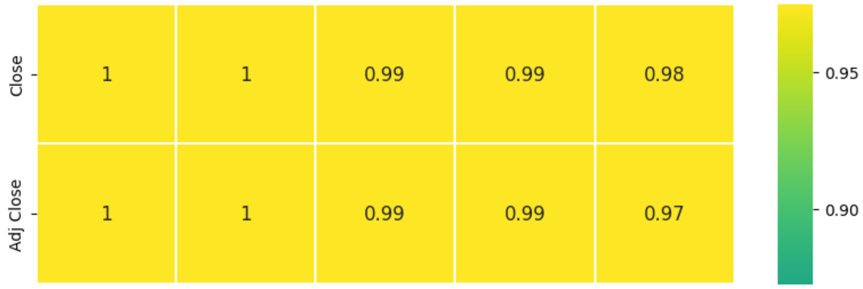
```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

```
# We have to ignore Day and Diff_Close_Price
correlation=numeric_feature.corr()
print(correlation['Close'].sort_values(ascending=False),'\n')
```

```
Close        1.000000
Adj Close    0.999137
High         0.990503
Low          0.989799
Open         0.975559
Volume      -0.122861
Name: Close, dtype: float64
```

```
# We are considering Threshold Value > 0.9
# Heat Map of Features (Threshold Value > 0.9)
k=5
cols=correlation.nlargest(k,'Close')['Close'].index
print(cols)
cm=np.corrcoef(data_frame[cols].values.T)
f,ax=plt.subplots(figsize=(10,8))
sns.heatmap(cm, vmax=.8, linewidths=0.01, square=True, annot=True, cmap='viridis', linecolor="white", xticklabels=cols.values,
```

```
Index(['Close', 'Adj Close', 'High', 'Low', 'Open'], dtype='object')
<Axes: >
```




```
# Rest we drop because their correlation value is too low
# Dropping Featutes
df=data_frame.drop(['Volume'],axis=1)
df
```

	Open	High	Low	Close	Adj Close
Date					
2022-05-02	277.709991	284.940002	276.220001	284.470001	281.706390
2022-05-03	283.959991	284.130005	280.149994	281.779999	279.042511
2022-05-04	282.589996	290.880005	276.730011	289.980011	287.162842
2022-05-05	285.540009	286.350006	274.339996	277.350006	274.655548
2022-05-06	274.809998	279.250000	271.269989	274.730011	272.060974
...	...	...	...	...	...
2023-04-26	296.700012	299.570007	292.730011	295.369995	295.369995
2023-04-27	295.970001	305.200012	295.250000	304.829987	304.829987
2023-04-28	304.010010	308.929993	303.309998	307.260010	307.260010
2023-05-01	306.970001	308.600006	305.149994	305.559998	305.559998
2023-05-02	307.760010	309.165009	303.910004	306.369995	306.369995

252 rows × 5 columns

Check for Multicollinearity

From VIF Factor

```
# Checking for Multicollinearity
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Define the Predictor Variables(Features) and the Response Variable(Target Variable)
c = df[['Open', 'High', 'Low', 'Adj Close']]
d = df['Close']

# Fit the linear regression model
model = sm.OLS(d, sm.add_constant(c)).fit() # OLS (Ordinary Least Squares) model is a type of linear regression model used to

# Check for multicollinearity using variance inflation factor (VIF)
vif = pd.DataFrame()
vif['variables'] = c.columns
vif['VIF'] = [variance_inflation_factor(c.values, i) for i in range(c.shape[1])]
print(vif)
```

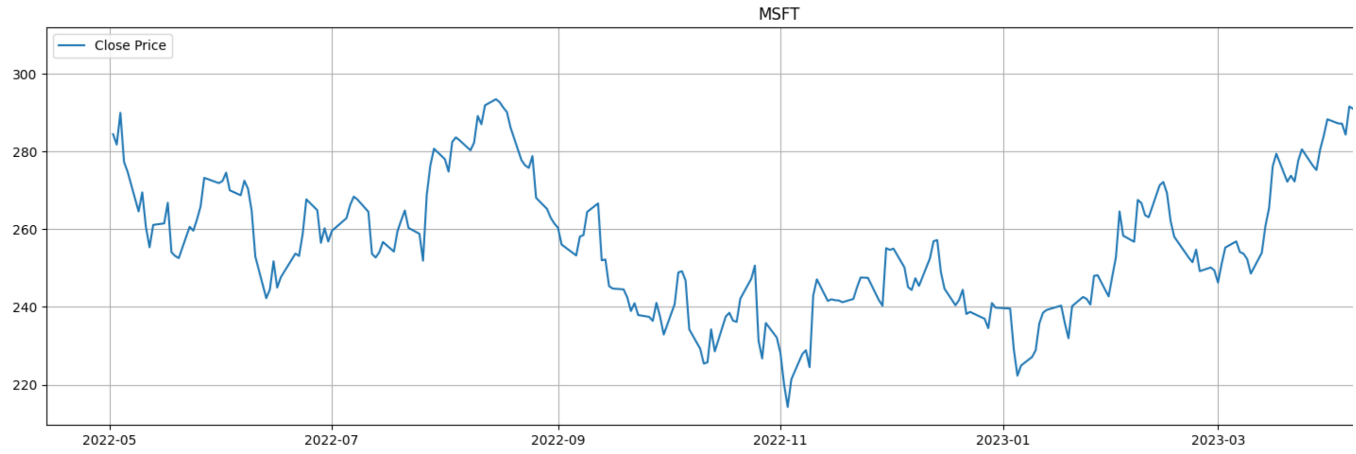
```
   variables           VIF
0       Open  25439.012132
1       High  26437.039930
2        Low  31636.947539
3  Adj Close  20149.155838
```

```
# The VIF measures the degree of multicollinearity for each predictor variable,
# If the VIF value is above 10, it is usually considered high and suggests that there is significant multicollinearity among t
# So, we will drop "Open","High","Low","Adj Close" because of high multicollinearity.
```

Visualization

Line chart: It displays the Trend and Seasonality

```
plt.figure(figsize=(20,12))
plt.subplot(2, 1, 1)
plt.plot(final_data.Close, label='Close Price')
plt.title(symbol)
plt.legend()
plt.grid()
plt.show()
```



1. Checking Trend

```
# 1. Trend is Upward and Non-Linear
# 2. Non-Stationary Time Series
    # No constant mean
    # No constant variance
```

2. Checking Seasonality: Repeating Trends/Pattern over time

```
# As such no Seasonality in the graph
```

```
# We will go for ARIMA (Autoregressive Integrated Moving Average)
# ARIMA, meaning that they only consider the past values of the target variable (univariate dataset).
```

```
# Dropping Featutes
final_data=df.drop(['Open', 'High', 'Low', 'Adj Close'],axis=1)
final_data
```

	Close
Date	
2022-05-02	284.470001

ARIMA Model:-

```
data = list(final_data["Close"])
```

```
# Augmented Dickey-Fuller (ADF)- to test the stationarity of a time series.

from statsmodels.tsa.stattools import adfuller

result = adfuller(data)

print("1. ADF : ",result[0]) # The test statistic of the ADF test.
print("2. P-Value : ", result[1]) # The p-value of the test.
print("3. Num Of Lags : ", result[2]) # The number of lags used in the test.
print("4. Num Of Observations Used For ADF Regression:", result[3]) # The number of observations used in the ADF regression.
print("5. Critical Values :") # A dictionary of critical values for the test at different significance levels.
for key, val in result[4].items():
  print("\t",key, ": ", val)
# The results of the test can be used to determine whether or not the time series is stationary.
```

```
1. ADF :  -1.7993302016199768
2. P-Value :  0.3807942805828326
3. Num Of Lags :  2
4. Num Of Observations Used For ADF Regression: 249
5. Critical Values :
         1% :  -3.4568881317725864
         5% :  -2.8732185133016057
         10% :  -2.5729936189738876
```

```
!pip install pmdarima
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pmdarima in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages (from pmdarima
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.22.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.10.1)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.13.5)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.26.15)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (6
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmda
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (202
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarin
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmda
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels>=0.13.2->pr
```

```
from pmdarima.arima.utils import ndiffs
d_value = ndiffs(data, test = "adf")
print("d value:", d_value)
```

```
d value: 1
```

```
from statsmodels.tsa.arima.model import ARIMA
from pmdarima import auto_arima
```

```
# Split the data into train and test sets
x_train = data[:-60]   # use first n-60 days for training
x_test = data[-60:]    # use last 60 days for testing
print(len(x_train),len(x_test))
```

```
192 60
```

```
import pmdarima as pm

def get_best_arima_order(data):
    """ Returns the best ARIMA order using stepwise approach.
    Parameters:
    -----------
    data : array-like
        Time series data
```

```
        Returns:
    --------
    order : tuple
        Best order for ARIMA model
    """
    stepwise_fit = pm.auto_arima(data, trace=True, suppress_warnings=True)
    order = stepwise_fit.order
    return order


import statsmodels.api as sm

# Get best ARIMA order
order = get_best_arima_order(data)

# Fit ARIMA model
model = sm.tsa.arima.ARIMA(data, order=order)
model = model.fit()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1568.471, Time=0.51 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1570.480, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1571.298, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1570.945, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1568.543, Time=0.02 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1570.524, Time=0.18 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1570.038, Time=0.20 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=1570.432, Time=0.70 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=1570.426, Time=1.71 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1572.095, Time=0.16 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=1571.678, Time=0.48 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1571.466, Time=0.31 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=1571.574, Time=1.26 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=1566.535, Time=0.23 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=1568.612, Time=0.10 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=1568.120, Time=0.09 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=1568.499, Time=0.71 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=1568.493, Time=0.52 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=1570.149, Time=0.07 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=1569.764, Time=0.11 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=1569.548, Time=0.12 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=1569.592, Time=0.66 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 8.320 seconds
```

```
start=len(x_train)
end=len(x_train)+len(x_test)-1
pred = model.predict(start=start,end=end)
pred
```

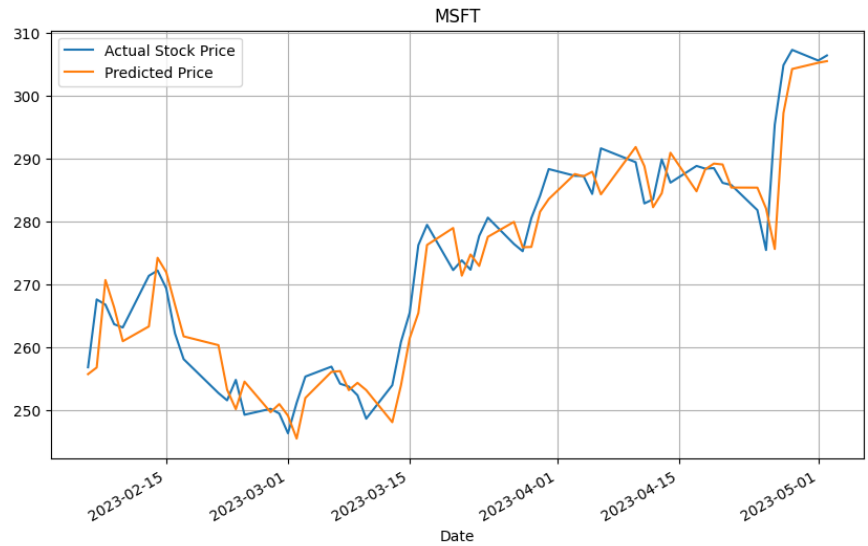
```
array([255.67817877, 256.73468051, 270.64815621, 266.30961504,
       260.91632124, 263.26592452, 274.18151683, 271.87891225,
       266.71142735, 261.68637325, 260.28825264, 253.21363647,
       250.10749378, 254.49045983, 249.62989504, 250.91821822,
       249.06608639, 245.42587934, 251.89052055, 256.03870076,
       256.16241706, 253.1300443 , 254.29464425, 253.13821773,
       248.03305564, 253.88339565, 261.37567849, 265.4058813 ,
       276.22758205, 278.92362084, 271.34122973, 274.7164487 ,
       272.89327505, 277.52106916, 279.90325123, 275.86173086,
       275.91155238, 281.5110492 , 283.52726224, 287.5039239 ,
       287.14662681, 287.87925833, 284.28070261, 291.80969636,
       288.7594963 , 282.22865387, 284.4470369 , 290.89456761,
       284.75994503, 288.25494212, 289.17111397, 289.02894803,
       285.35803394, 285.34271507, 282.02908758, 275.58399345,
       297.21017072, 304.21748895, 305.20526818, 305.46624929])
```

```
s = pd.Series(pred, index =final_data.index[-60:])
s
```

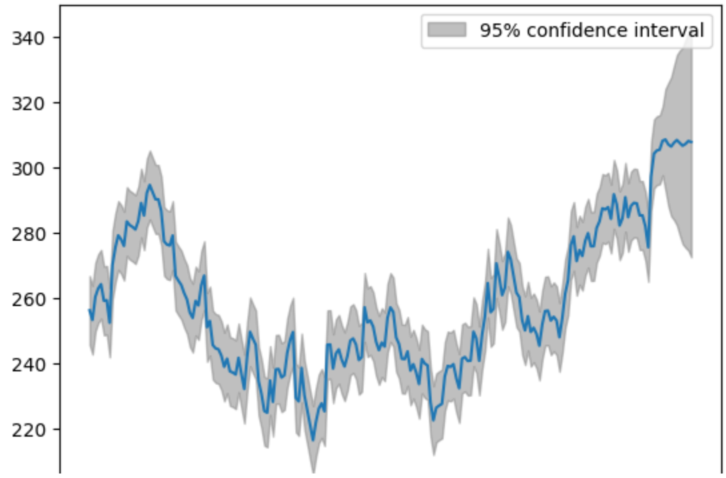
```
Date
2023-02-06    255.678179
2023-02-07    256.734681
2023-02-08    270.648156
2023-02-09    266.309615
2023-02-10    260.916321
2023-02-13    263.265925
2023-02-14    274.181517
2023-02-15    271.878912
2023-02-16    266.711427
2023-02-17    261.686373
2023-02-21    260.288253
2023-02-22    253.213636
2023-02-23    250.107494
2023-02-24    254.490460
2023-02-27    249.629895
```

```
    2023-02-28    250.918218
    2023-03-01    249.066086
    2023-03-02    245.425879
    2023-03-03    251.890521
    2023-03-06    256.038701
    2023-03-07    256.162417
    2023-03-08    253.130044
    2023-03-09    254.294644
    2023-03-10    253.138218
    2023-03-13    248.033056
    2023-03-14    253.883396
    2023-03-15    261.375678
    2023-03-16    265.405881
    2023-03-17    276.227582
    2023-03-20    278.923621
    2023-03-21    271.341230
    2023-03-22    274.716449
    2023-03-23    272.893275
    2023-03-24    277.521069
    2023-03-27    279.903251
    2023-03-28    275.861731
    2023-03-29    275.911552
    2023-03-30    281.511049
    2023-03-31    283.527262
    2023-04-03    287.503924
    2023-04-04    287.146627
    2023-04-05    287.879258
    2023-04-06    284.280703
    2023-04-10    291.809696
    2023-04-11    288.759496
    2023-04-12    282.228654
    2023-04-13    284.447037
    2023-04-14    290.894568
    2023-04-17    284.759945
    2023-04-18    288.254942
    2023-04-19    289.171114
    2023-04-20    289.028948
    2023-04-21    285.358034
    2023-04-24    285.342715
    2023-04-25    282.029088
    2023-04-26    275.583993
```

```
plt.figure(figsize=(10,6), dpi=100)
final_data['Close'][-60:].plot(label='Actual Stock Price', legend=True)
s.plot(label='Predicted Price', legend=True,)
plt.title(symbol)
plt.legend()
plt.grid()
plt.show()
```




```
from statsmodels.graphics.tsaplots import plot_predict
plot_predict(model, start = len(data)-200, end = len(data)+10, dynamic = False);
```




```
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(x_test, pred))
```

```
5.048356743623307
```

```
from sklearn.metrics import r2_score
r2_score(x_test, pred)
```

```
0.905540512071107
```

Predicting Future 10 values:

```
pred_future = model.predict(start=end,end=end+9)
pred_future
```

```
array([305.46624929, 308.18642281, 308.58391515, 307.08377091,
       306.42164169, 307.59035024, 308.41350126, 307.5676445 ,
       306.67200876, 307.22112174])
```

```
import datetime
start_date = datetime.datetime.today() + datetime.timedelta(days=1)
dates = [start_date + datetime.timedelta(days=idx) for idx in range(10)]
```

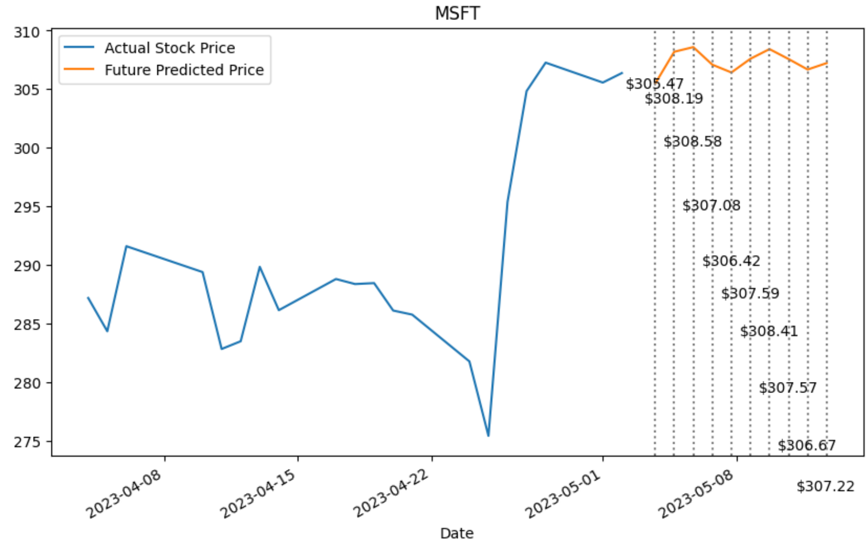
```
pred_future2 = pd.Series(pred_future, index =dates)
pred_future2
```

```
2023-05-03 17:55:58.204319    305.466249
2023-05-04 17:55:58.204319    308.186423
2023-05-05 17:55:58.204319    308.583915
2023-05-06 17:55:58.204319    307.083771
2023-05-07 17:55:58.204319    306.421642
2023-05-08 17:55:58.204319    307.590350
2023-05-09 17:55:58.204319    308.413501
2023-05-10 17:55:58.204319    307.567644
2023-05-11 17:55:58.204319    306.672009
2023-05-12 17:55:58.204319    307.221122
dtype: float64
```

```
plt.figure(figsize=(10,6), dpi=100)
final_data['Close'][-20:].plot(label='Actual Stock Price', legend=True)
pred_future2.plot(label='Future Predicted Price', legend=True)

max_price = pred_future2.max()
for i, (date, price) in enumerate(pred_future2.items()):
    offset = i*-0.013*max_price # adjust the offset as needed
    plt.axvline(x=date, linestyle='dotted', color='gray')
    plt.text(date, price+offset, f'${price:.2f}', ha='center', va='center', fontsize=10)
    if i == 9:
        break

plt.title(symbol)
plt.legend()
plt.show()
```


MSFT
Actual Stock Price
Future Predicted Price
310
305
300
295
290
285
280
275
$305.47
$308.19
$308.58
$307.08
$306.42
$307.59
$308.41
$307.57
$306.67
$307.22
2023-04-08
2023-04-15
2023-04-22
2023-05-01
2023-05-08
Date