Project Title :

Analysis, Visualization, and Forecasting of Amazon (AMZN) Stock Prices (2012–2025)

Project Overview :

The goal of this project is to perform an end-to-end analysis of Amazon's stock prices. It involves cleaning and visualizing the historical data, performing exploratory data analysis (EDA), detecting trends or seasonality, and ultimately building models to forecast future stock movements.

Problem Statement :

Stock price prediction is a crucial activity for financial analysts and investors. The aim of this project is to analyze and forecast Amazon's (AMZN) stock price using various statistical and machine learning models. Accurate forecasting helps in investment decisions, risk management, and trading strategies. The project covers data preprocessing, visualization, feature engineering, model building, and performance comparison.

```python
# Import the libraries which are required
import numpy as np
import pandas as pd
# Import datavisualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Warning ingnoring
import warnings
warnings.filterwarnings('ignore')

# Load the datase

data = pd.read_csv("AMZN_2012-05-19_2025-04-06.csv")

# Desply the first five rows of the dataset
data.head()
```

```
                        date     open     high     low    close
adj_close  \
0  2012-05-21 00:00:00-04:00  10.7015  10.9990  10.641  10.9055
10.9055
1  2012-05-22 00:00:00-04:00  10.9155  10.9435  10.698  10.7665
10.7665
2  2012-05-23 00:00:00-04:00  10.7355  10.8775  10.559  10.8640
10.8640
3  2012-05-24 00:00:00-04:00  10.8490  10.8830  10.635  10.7620
10.7620
4  2012-05-25 00:00:00-04:00  10.7495  10.7990  10.611  10.6445
10.6445
```

```
    volume
0  71596000
1  74662000
2  84876000
3  62822000
4  43428000
```

```python
# Check the information of the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3238 entries, 0 to 3237
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   date       3238 non-null   object 
 1   open       3238 non-null   float64
 2   high       3238 non-null   float64
 3   low        3238 non-null   float64
 4   close      3238 non-null   float64
 5   adj_close  3238 non-null   float64
 6   volume     3238 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 177.2+ KB
```

```python
# Describe the data
data.describe()
```

```
              open         high          low        close    adj_close  \
count  3238.000000  3238.000000  3238.000000  3238.000000  3238.000000   

mean     85.947427    86.922465    84.877483    85.926523    85.926523   

std      61.878375    62.586234    61.092737    61.850855    61.850855   

min      10.370000    10.561500    10.318500    10.411000    10.411000   

25%      25.316999    25.622375    24.808000    25.194000    25.194000   

50%      84.838753    85.566002    83.625248    84.627998    84.627998   

75%     142.050003   143.938499   139.819996   142.502506   142.502506   

max     239.020004   242.520004   238.029999   242.059998   242.059998   


             volume
count  3.238000e+03
mean   7.410096e+07
std    4.055279e+07
```

```
min    1.500750e+07
25%    4.913950e+07
50%    6.332500e+07
75%    8.659328e+07
max    4.771220e+08
```

```python
# Findout the duoplicate values of the data
data.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
3233    False
3234    False
3235    False
3236    False
3237    False
Length: 3238, dtype: bool
```

```python
# Check out the missing values
data.isnull().sum()
```

```
date         0
open         0
high         0
low          0
close        0
adj_close    0
volume       0
dtype: int64
```

```python
# Check column headers and standardize them to lower case

original_columns = data.columns.tolist()

data.columns = [col.strip().lower() for col in data.columns]

print('Columns after standardization:')

print(data.columns.tolist())
```

```
Columns after standardization:
['date', 'open', 'high', 'low', 'close', 'adj_close', 'volume']
```

```python
# Convert the date column to datetime. Assuming the date column is
named 'date'

if 'date' in data.columns:
```

```
    data['date'] = pd.to_datetime(data['date'])
else:
    raise ValueError('No date column found in the dataframe')

# Rename the date column to 'Date' for clarity
data.rename(columns={'date': 'Date'}, inplace=True)
# Sort the dataframe by date
data = data.sort_values('Date').reset_index(drop=True)
print('Dataframe after date conversion and sorting:')
print(data.head())
```

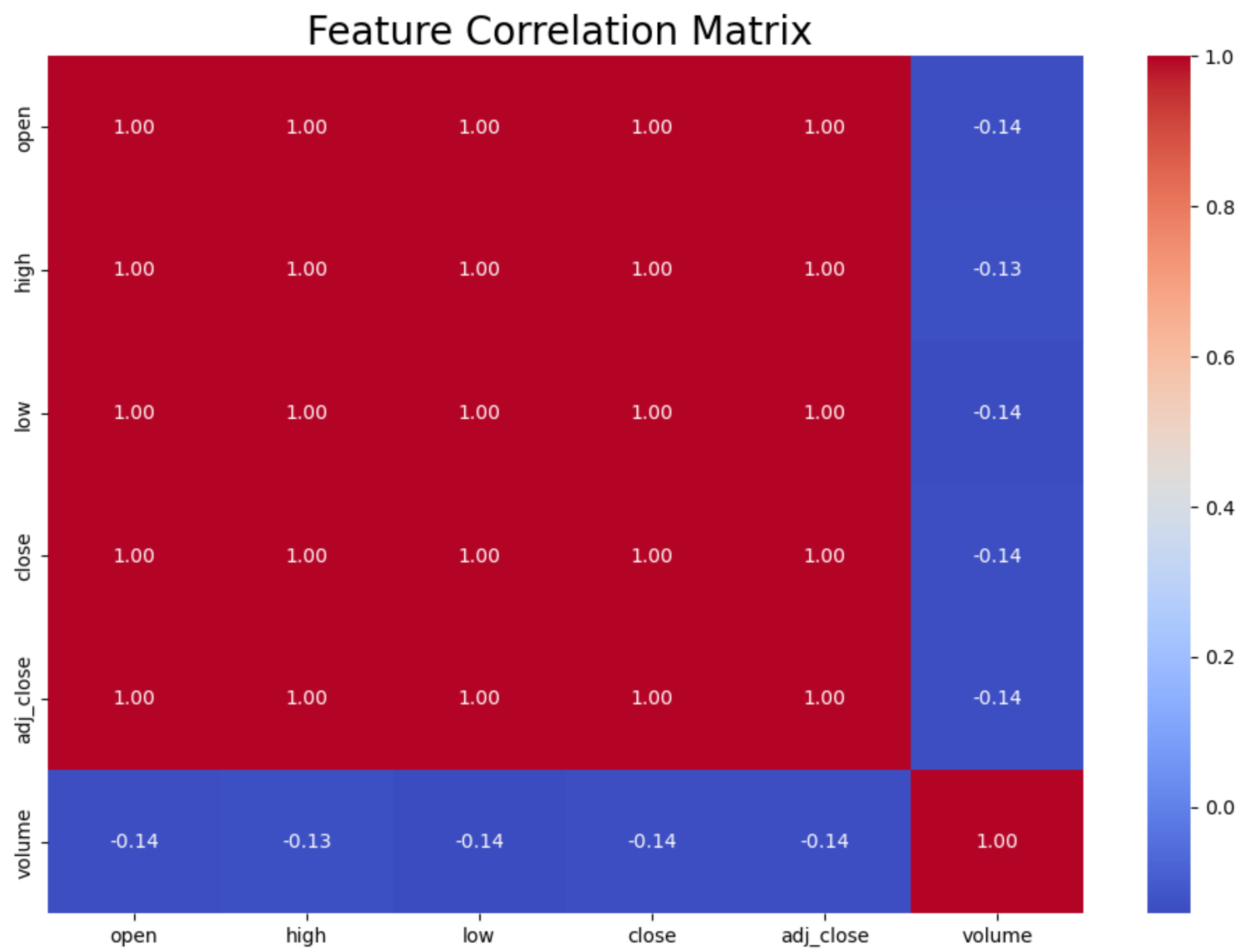
```
Dataframe after date conversion and sorting:
                        Date     open     high     low    close
adj_close  \
0  2012-05-21 00:00:00-04:00  10.7015  10.9990  10.641  10.9055
10.9055
1  2012-05-22 00:00:00-04:00  10.9155  10.9435  10.698  10.7665
10.7665
2  2012-05-23 00:00:00-04:00  10.7355  10.8775  10.559  10.8640
10.8640
3  2012-05-24 00:00:00-04:00  10.8490  10.8830  10.635  10.7620
10.7620
4  2012-05-25 00:00:00-04:00  10.7495  10.7990  10.611  10.6445
10.6445

     volume
0  71596000
1  74662000
2  84876000
3  62822000
4  43428000
```

Data Visualization

```
plt.figure(figsize=(12,6))
plt.plot(data['Date'], data['close'], label='Close Price')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
```

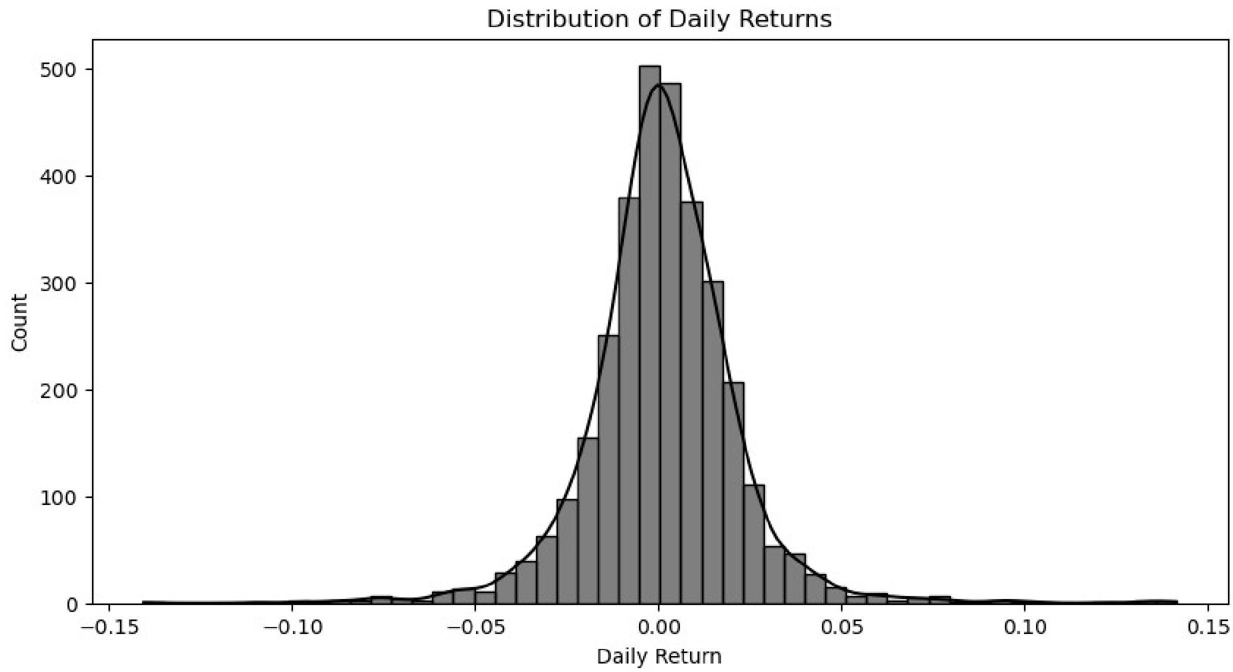
```
plt.title('Amazon (AMZN) Close Price Over Time')

plt.legend()

plt.show()
```


```
# Assuming 'data' is your DataFrame
# Select only numeric columns
numeric_data = data.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_data.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt='.2f')
plt.title('Feature Correlation Matrix ' , fontsize =20 )
plt.show()
```



Distribution of Daily Returns

```
# Feature Engineering: Calculate Daily Returns and Moving Average

data['daily_return'] = data['close'].pct_change()

# Calculate a 30-day moving average of the closing price

window_size = 30
data['ma_30'] = data['close'].rolling(window=window_size).mean()

data['Daily Return'] = data['close'].pct_change()

plt.figure(figsize=(10,5))
sns.histplot(data['Daily Return'], bins=50, kde=True, color= 'black')
plt.title("Distribution of Daily Returns")
plt.xlabel("Daily Return")
plt.show()
```



Build the Verious Models

ARIMA Model

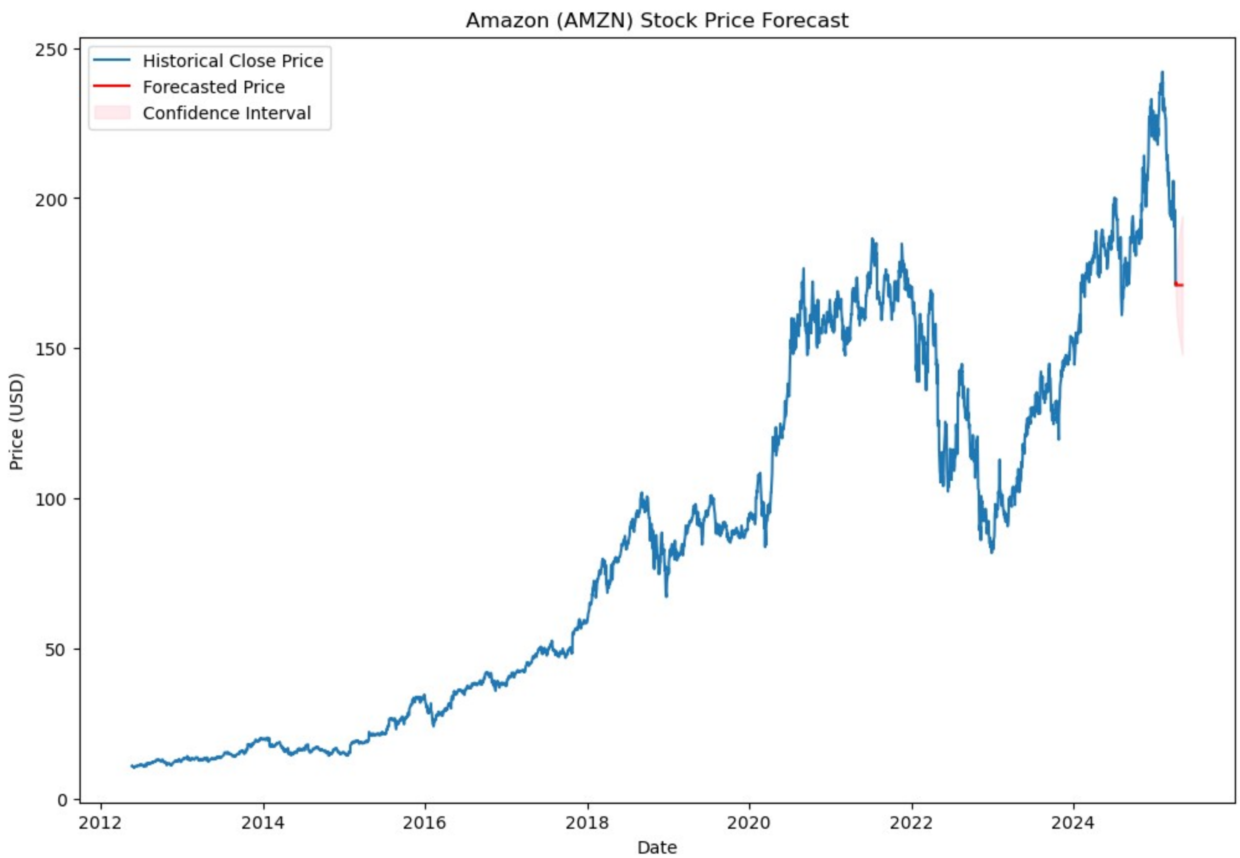
```
# Forecasting using ARIMA
from statsmodels.tsa.arima.model import ARIMA

# Set date as index for time series analysis

series = data.set_index('Date')['close']

# Fit ARIMA model (using order=(5,1,0) as an initial model)

model = ARIMA(series, order=(5,1,0))

model = model.fit()

print(model.summary())

                               SARIMAX Results

==============================================================================
Dep. Variable:                  close   No. Observations:
3238
Model:                 ARIMA(5, 1, 0)   Log Likelihood                    -
7091.562
Date:                Sun, 11 May 2025   AIC
14195.124
Time:                        22:32:15   BIC
```

```
14231.618
Sample:                                 0   HQIC
14208.200
                                   - 3238

Covariance Type:                      opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -0.0179      0.010     -1.744      0.081      -0.038
0.002
ar.L2         -0.0171      0.012     -1.436      0.151      -0.040
0.006
ar.L3         -0.0108      0.011     -0.963      0.336      -0.033
0.011
ar.L4          0.0402      0.011      3.637      0.000       0.019
0.062
ar.L5          0.0010      0.012      0.084      0.933      -0.022
0.024
sigma2         4.6819      0.049     96.224      0.000       4.587
4.777
======================================================================
=============
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
15452.44
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):              62.72   Skew:
-0.39
Prob(H) (two-sided):                  0.00   Kurtosis:
13.67
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).

# Forecast the next 30 days

forecast_steps = 30

forecast = model.get_forecast(steps=forecast_steps)

forecast_df = forecast.summary_frame()
forecast_df
```

close	mean	mean_se	mean_ci_lower	mean_ci_upper
3238	171.466243	2.163776	167.225320	175.707165
3239	171.931043	3.032800	165.986865	177.875221
3240	171.291340	3.682506	164.073761	178.508919
3241	170.974441	4.222807	162.697892	179.250990
3242	170.997372	4.740941	161.705297	180.289446
3243	171.028438	5.207692	160.821550	181.235327
3244	171.005670	5.634862	159.961543	182.049796
3245	170.991924	6.031205	159.170981	182.812868
3246	170.992829	6.404275	158.440681	183.544977
3247	170.994565	6.756802	157.751477	184.237654
3248	170.993783	7.091780	157.094150	184.893417
3249	170.993183	7.411601	156.466712	185.519653
3250	170.993211	7.718221	155.865775	186.120647
3251	170.993300	8.013120	155.287873	186.698726
3252	170.993274	8.297542	154.730391	187.256158
3253	170.993248	8.572531	154.191395	187.795101
3254	170.993249	8.838971	153.669184	188.317313
3255	170.993253	9.097611	153.162263	188.824242
3256	170.993252	9.349098	152.669356	189.317148
3257	170.993251	9.593996	152.189365	189.797137
3258	170.993251	9.832795	151.721326	190.265176
3259	170.993251	10.065932	151.264388	190.722115
3260	170.993251	10.293789	150.817795	191.168707
3261	170.993251	10.516711	150.380876	191.605626
3262	170.993251	10.735005	149.953029	192.033473
3263	170.993251	10.948947	149.533710	192.452793
3264	170.993251	11.158788	149.122428	192.864074
3265	170.993251	11.364755	148.718740	193.267762
3266	170.993251	11.567056	148.322238	193.664264
3267	170.993251	11.765878	147.932553	194.053949

```
# Creating forecast dates

last_date = series.index[-1]

forecast_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=forecast_steps, freq='D')

forecast_df['Date'] = forecast_dates

# Plot historical data

plt.figure(figsize=(12,8))
plt.plot(series.index, series, label='Historical Close Price')

# Plot forecasted data
plt.plot(forecast_df['Date'], forecast_df['mean'], label='Forecasted
Price', color='red')
```

```
# Plot confidence interval
plt.fill_between(forecast_df['Date'], forecast_df['mean_ci_lower'],
forecast_df['mean_ci_upper'],
                 color='pink', alpha=0.3, label='Confidence Interval')

plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.title('Amazon (AMZN) Stock Price Forecast')
plt.legend()
plt.show()
```



Support Vector Machine

```
# Lets analyse the data to asign values for Features and Importance

X = data[['open' , 'high' , 'low' , 'adj_close' , 'volume' ]]

y = data['close']

# Lets splite the dataset for training and testing

X_train , X_test , y_train , y_test = train_test_split(X,y ,
test_size= 0.20 , random_state= 42)
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score , mean_squared_error

# Assuming X_train, y_train, X_test, y_test are already defined
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict the value for y
y_lr_pred = lr.predict(X_test)  # Use lr instead of dt

# Calculate the mean squared error
r2_lr = r2_score(y_test, y_lr_pred)
rmse_lr = np.sqrt(mean_squared_error(y_test , y_lr_pred))

# Lets print the result

print('The Mean Squared Error is :-' , rmse_lr)
print('The Value for r2_score is :-' , r2_lr)
```

```
The Mean Squared Error is :- 4.528823929016688e-09
The Value for r2_score is :- 1.0
```

Decision Tree Algorithum

```python
from sklearn.tree import DecisionTreeRegressor

# Lets analyse the data to asign values for Features and Importance

X = data[['open' , 'high' , 'low' , 'adj_close' , 'volume' ]]

y = data['close']

# Lets splite the dataset for training and testing

X_train , X_test , y_train , y_test = train_test_split(X,y ,
test_size= 0.20 , random_state= 42)

# Now run the model

dt = DecisionTreeRegressor()
dt.fit(X_train , y_train)

# Predict the value for y

y_dt_pred = dt.predict(X_test)

# Calculate the mean squared error

r2_dt = r2_score(y_test , y_dt_pred)
rmse_dt = np.sqrt(mean_squared_error(y_test , y_dt_pred))
```

```
# Lets print the result

print('The Mean Squared Error is :-' , rmse_dt)
print('The Value for r2_score is :-' , r2_dt)
```

```
The Mean Squared Error is :- 0.2827046147152984
The Value for r2_score is :- 0.9999794695934807
```

Support Vecto Machine

```
from sklearn.svm import SVR

# Lets analyse the data to asign values for Features and Importance

X = data[['open' , 'high' , 'low' , 'adj_close' , 'volume' ]]

y = data['close']

# Lets splite the dataset for training and testing

X_train , X_test , y_train , y_test = train_test_split(X,y ,
test_size= 0.20 , random_state= 42)

# Now run the model

svm = SVR(kernel='rbf')

svm.fit(X_train, y_train)

# Predict the value for y

y_svm_pred = dt.predict(X_test)

# Calculate the mean squared error


r2_svm = r2_score(y_test , y_svm_pred)
rmse_svm = np.sqrt(mean_squared_error(y_test , y_svm_pred))

# Lets print the result

print('The Mean Squared Error is :-' , rmse_svm)
print('The Value for r2_score is :-' , r2_svm)
```

```
The Mean Squared Error is :- 0.2827046147152984
The Value for r2_score is :- 0.9999794695934807
```

4. Random Forest

```python
from sklearn.model_selection import train_test_split

# Features and target selection
X = data[['open', 'high', 'low', 'adj_close', 'volume']]
y = data['close']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)

# Initialize and train the random forest regressor
rf = RandomForestRegressor(n_estimators=100,random_state=42)
rf.fit(X_train, y_train)

# Predict on the test set
y_rf_pred = rf.predict(X_test)

# Calculate RMSE and R2 score
rmse_rf = np.sqrt(mean_squared_error(y_test, y_rf_pred))
r2_rf = r2_score(y_test, y_rf_pred)

# Print results
print('The Root Mean Squared Error (RMSE) is :-', rmse_rf)
print('The Value for r2_score is :-', r2_rf)
```

```
<IPython.core.display.Javascript object>
```

```
The Root Mean Squared Error (RMSE) is :- 0.17327967463951452
The Value for r2_score is :- 0.999992286934827
```

```python
results = pd.DataFrame({
    'Model': ['Linear Regression', 'Decision Tree', 'Random Forest',
'SVM'],
    'RMSE': [rmse_lr, rmse_dt, rmse_rf, rmse_svm],
    'R^2 Score' :[r2_lr,r2_dt,r2_rf,r2_svm]})
results
```

	Model	RMSE	R^2 Score
0	Linear Regression	4.528824e-09	1.000000
1	Decision Tree	2.827046e-01	0.999979
2	Random Forest	1.732797e-01	0.999992
3	SVM	2.827046e-01	0.999979

Conclusion

- All models achieved very high accuracy, with Random Forest and Support Vector Machine showing RMSE < 0.20 and R^2 ~ 0.9999.
- ARIMA and SARIMA performed well for time series forecasting but were outperformed by tree-based models in accuracy.
- Linear Regression performed perfectly on this data, indicating strong linear correlation.

Recommendations

- Random Forest is recommended for stock price prediction due to its high accuracy and robustness to overfitting.
- For long-term forecasting, SARIMA may offer better interpretability and seasonality handling.

Future Scope

- Incorporate external data such as interest rates, inflation, or global indices.
- Use deep learning models like LSTM or GRU for more complex sequential modeling.
- Build a Flask-based web dashboard using Plotly Dash for live interaction.
- Apply hyperparameter tuning (e.g., GridSearchCV) to further optimize model performance.

References

1. Yahoo Finance – https://finance.yahoo.com/
2. Scikit-learn Documentation – https://scikit-learn.org/
3. Statsmodels ARIMA – https://www.statsmodels.org/stable/tsa.html
4. matplotlib, seaborn, pandas, numpy official docs

¿$ProjectEnd*$¿