Overview

Time series forecasting stock_details_5_years the involves cleaning and transforming historical data, selecting models like ARIMA or Prophet, training them to capture patterns, evaluating performance using RMSE or MAE, and deploying validated models for predictions.

```
!pip install ydata-profiling
!pip install statsmodels
!pip install scikit-learn
```

```
Requirement already satisfied: matplotlib>=3.5 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: visions<0.8.0,>=0.7.5 in /usr/local/lib/python3.11/
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.11/
Requirement already satisfied: multimethod<2,>=1.4 in /usr/local/lib/python3.11/di
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.11
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/d
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11
Requirement already satisfied: pydantic-core==2.27.2 in /usr/local/lib/python3.11/
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dis
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages
```

```
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/di
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/d
```

```python
import seaborn as sns
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from ydata_profiling import ProfileReport
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
# Import the stock_details_5_years
df = pd.read_csv("/content/drive/MyDrive/stock_details_5_years.csv")
```

```python
df
```

	Date	Open	High	Low	Close	Volume	Dividends
0	2018-11-29 00:00:00-05:00	43.829761	43.863354	42.639594	43.083508	167080000.0	0.00
1	2018-11-29 00:00:00-05:00	104.769074	105.519257	103.534595	104.636131	28123200.0	0.00
2	2018-11-29 00:00:00-05:00	54.176498	55.007500	54.099998	54.729000	31004000.0	0.00
3	2018-11-29 00:00:00-05:00	83.749496	84.499496	82.616501	83.678497	132264000.0	0.00
4	2018-11-29 00:00:00-05:00	39.692784	40.064904	38.735195	39.037853	54917200.0	0.04
...	...	...	...	...	...	...	...
67381	2019-07-01 00:00:00-04:00	49.382082	49.539489	49.006054	49.521999	903800.0	0.00
67382	2019-07-01 00:00:00-04:00	318.700012	318.809998	310.010010	316.739990	204100.0	0.00
67383	2019-07-01 00:00:00-04:00	6.587493	6.604560	6.527763	6.553361	1871100.0	0.00

Next steps: View recommended plots New interactive sheet

Basic Metrices

```
# shape of data set
df.shape
```

```
(602962, 9)
```

```
# dataset of columns name
df.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Dividends',
       'Stock Splits', 'Company'],
      dtype='object')
```

```
# Overview of the dataset structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 602962 entries, 0 to 602961
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Date          602962 non-null  object 
 1   Open          602962 non-null  float64
 2   High          602962 non-null  float64
 3   Low           602962 non-null  float64
 4   Close         602962 non-null  float64
 5   Volume        602962 non-null  int64  
 6   Dividends     602962 non-null  float64
 7   Stock Splits  602962 non-null  float64
 8   Company       602962 non-null  object 
dtypes: float64(6), int64(1), object(2)
memory usage: 41.4+ MB
```

Datatype Conversions

```
def date_split(x):
    a = x.split(' ')[0]
    b = x.split(' ')[1].split('-')[0]
    d = a+' '+b
    return d
```

```
pd.to_datetime(date_split(df['Date'][0]))
```

```
Timestamp('2018-11-29 00:00:00')
```

```
df['Date'] = df['Date'].apply(lambda x:date_split(x))
df['Date']
```

```
0         2018-11-29 00:00:00
1         2018-11-29 00:00:00
2         2018-11-29 00:00:00
3         2018-11-29 00:00:00
4         2018-11-29 00:00:00
                 ...
602957    2023-11-29 00:00:00
602958    2023-11-29 00:00:00
602959    2023-11-29 00:00:00
602960    2023-11-29 00:00:00
```

```
602961    2023-11-29 00:00:00
Name: Date, Length: 602962, dtype: object
```

```
df
```

	Date	Open	High	Low	Close	Volume	Dividends
0	2018-11-29 00:00:00	43.829761	43.863354	42.639594	43.083508	167080000	0.00
1	2018-11-29 00:00:00	104.769074	105.519257	103.534595	104.636131	28123200	0.00
2	2018-11-29 00:00:00	54.176498	55.007500	54.099998	54.729000	31004000	0.00
3	2018-11-29 00:00:00	83.749496	84.499496	82.616501	83.678497	132264000	0.00
4	2018-11-29 00:00:00	39.692784	40.064904	38.735195	39.037853	54917200	0.04
...	...	...	...	...	...	...	...
602957	2023-11-29 00:00:00	26.360001	26.397499	26.120001	26.150000	1729147	0.00
602958	2023-11-29 00:00:00	27.680000	28.535000	27.680000	28.350000	1940066	0.00
602959	2023-11-29	75.940002	76.555000	75.257500	75.610001	298699	0.00

Data Cleaning

```
# Check for missing or null values in the stock_details_5_years dataset and count them fo
df.isna().sum()
```

```
Date            0
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
Company         0
dtype: int64
```

Unique Values

```
# Check the number of unique values in each column of the DataFrame 'df'
# This helps in understanding the diversity of data in each column.
df.nunique()
```

```
Date              1258
Open            510592
High            514315
Low             513389
Close           484353
Volume          170929
Dividends          960
Stock Splits        40
Company            491
dtype: int64
```

```
value_counts = {col: df[col].value_counts() for col in ['Open', 'High', 'Low', 'Close', '
# Display the value counts for each column
for col, counts in value_counts.items():
    print(f"Value counts for {col}:\n{counts}\n")
```

```
Value counts for Open:
Open
140.000000     55
150.000000     48
60.000000      48
100.000000     48
65.000000      46
               ..
48.612281       1
161.399136      1
28.091971       1
1171.079956     1
6164.000000     1
Name: count, Length: 510592, dtype: int64

Value counts for High:
High
2.920000      36
9.900000      29
3.150000      29
3.170000      28
155.000000    27
              ..
40.929012      1
440.920013     1
37.215579      1
30.501805      1
84.995003      1
Name: count, Length: 514315, dtype: int64

Value counts for Low:
Low
```

```
9.900000      35
3.090000      32
3.140000      28
2.890000      27
3.110000      25
              ..
8.793761       1
65.908869      1
84.592707      1
301.632601     1
75.257500      1
Name: count, Length: 513389, dtype: int64

Value counts for Close:
Close
2.910000      32
3.140000      30
9.950000      28
3.130000      27
9.800000      26
              ..
51.473606      1
124.716805     1
134.903076     1
320.975983     1
40.125000      1
```

Descriptive Statistics

```
# Summary statistics for numerical columns
df.describe(include = 'all').T
```

	count	unique	top	freq	mean	std	min
Date	602962	1258	2023-11-29 00:00:00	491	NaN	NaN	NaN
Open	602962.0	NaN	NaN	NaN	140.074711	275.401725	1.052425
High	602962.0	NaN	NaN	NaN	141.853492	279.003191	1.061195
Low	602962.0	NaN	NaN	NaN	138.276316	271.895276	1.026114
Close	602962.0	NaN	NaN	NaN	140.095204	275.477969	1.034884
Volume	602962.0	NaN	NaN	NaN	5895601.184909	13815961.832517	0.0
Dividends	602962.0	NaN	NaN	NaN	0.00731	0.12057	0.0
Stock Splits	602962.0	NaN	NaN	NaN	0.000344	0.050607	0.0
Company	602962	491	AAPL	1258	NaN	NaN	NaN

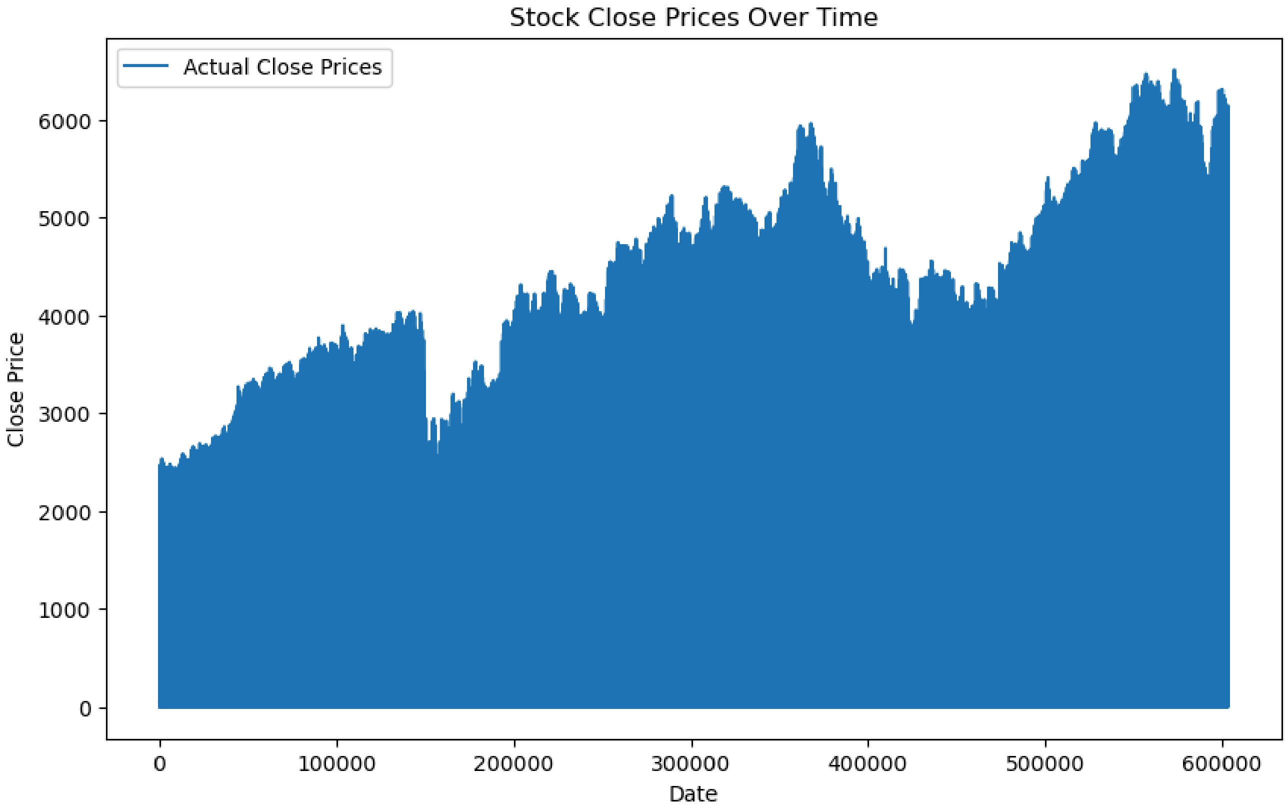
Unsupported Cell Type. Double-Click to inspect/edit the content.

```
# Split the data into training and test sets (e.g., last 20% for testing)
train_size = int(len(df) * 0.8)
train_data = df['Close'][:train_size]
test_data = df['Close'][train_size:]

# Fit an ARIMA model (you can adjust p, d, q based on your data)
model = ARIMA(train_data, order=(5, 1, 0))  # p=5, d=1, q=0 (adjust these values)
model_fit = model.fit()

# Make predictions for the test set
predictions = model_fit.forecast(steps=len(test_data))

# Calculate RMSE and MAE
rmse = np.sqrt(mean_squared_error(test_data, predictions))
mae = mean_absolute_error(test_data, predictions)

print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
```

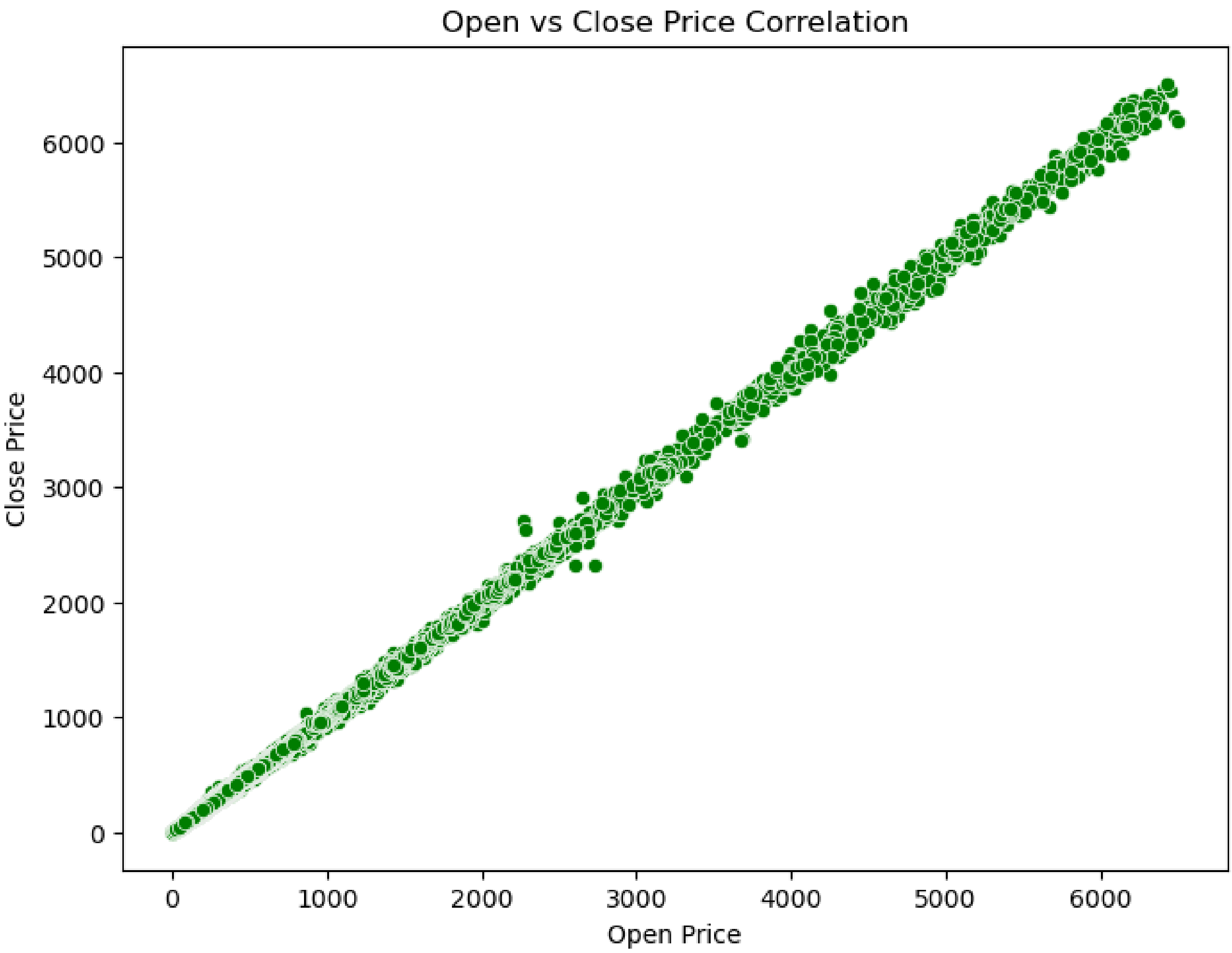
```
RMSE: 354.75400438562195
MAE: 121.07360469022312
```

Data Visualization

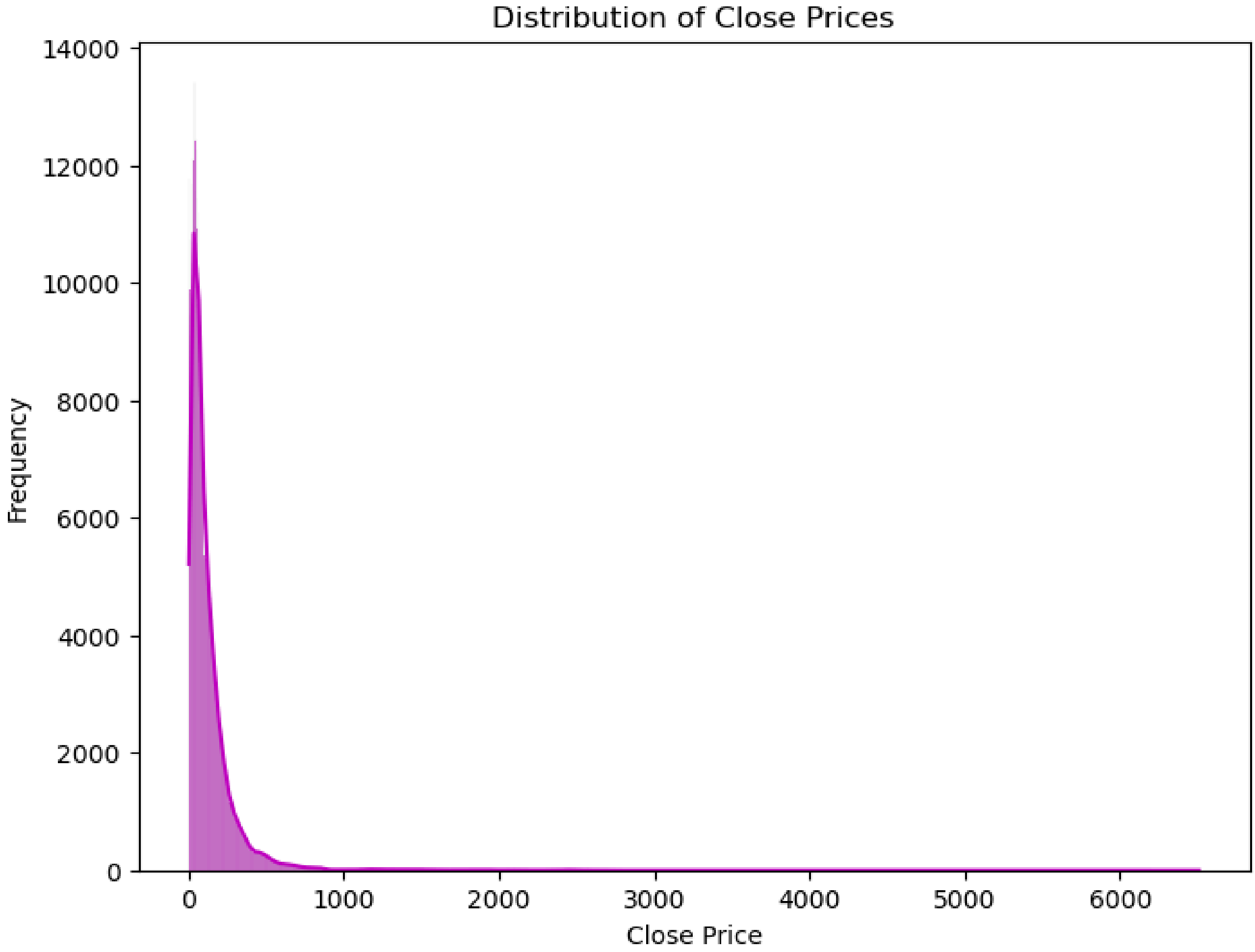
```
# 1. Plot the closing prices over time to show trends
plt.figure(figsize=(10,6))
plt.plot(df['Close'],label="Actual Close Prices")
plt.title("Stock Close Prices Over Time")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.legend()
plt.show()
```




```
#  2. Correlation between Open and Close Prices ---
plt.figure(figsize=(8,6))
sns.scatterplot(x=data['Open'], y=data['Close'], color='green')
plt.title('Open vs Close Price Correlation')
plt.xlabel('Open Price')
plt.ylabel('Close Price')
plt.show()
```




```
#  3. Distribution of the Close Prices
plt.figure(figsize=(8,6))
sns.histplot(df['Close'], kde=True, color = 'm')
plt.title("Distribution of Close Prices")
plt.xlabel("Close Price")
plt.ylabel("Frequency")
plt.show()
```




```
#  4. Volume vs Close Prices
plt.figure(figsize=(10,6))
plt.scatter(df['Volume'],df['Close'], alpha=0.5)
plt.title("Volume vs Close Prices")
plt.xlabel("Volume")
plt.ylabel("Close Price")
plt.show()
```

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
# Assuming 'test_data' and 'predictions' are already calculated as in your code
# Generate the corresponding date index for the test data
test_dates = df.index[train_size:] # Adjust this based on your df structure

#plot the actual vs predicted values
plt.figure(figsize=(8,6))

# plot actual test data
plt.plot(test_dates, predictions, label='Perdicted Close Prices', color='red', linestyle=

#Add title and labels
plt.title('ARIMA Model: Actual vs Predicted Close Prices', fontsize=16)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Close Price", fontsize=12)

# Add legend
plt.legend()

#show grid for better readability
plt.grid(True)

#Display the plot
plt.show()
```