Performing **Exploratory Data Analysis** in *EDAP TMS S.A. (EDAP) Stock Performance*
* *italicized text

About Dataset This dataset provides historical stock market performance data for specific companies. It enables users to analyze and understand the past trends and fluctuations in stock prices over time. This information can be utilized for various purposes such as investment analysis, financial research, and market trend forecasting.

EVANGELIN PRIYANKA R
Jun 22, 2024
Importing Necessary Libraries

```
#Importing Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

EVANGELIN PRIYANKA R
Jun 22, 2024
Connecting to Google Drive

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

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

EVANGELIN PRIYANKA R
Jun 22, 2024
Loading data

```
data=pd.read_csv("/content/drive/MyDrive/EDAP_stock_data.csv")
data.head()
```

	Unnamed: 0	open	high	low	close	adjclos
0	1997-08-01	9.250	9.25	8.250	8.500	8.50
1	1997-08-04	8.500	8.75	8.000	8.125	8.12
2	1997-08-05	8.125	8.50	8.125	8.125	8.12

EVANGELIN PRIYANKA R
Jun 22, 2024
Summary Statistics

```
data.shape
```

```
(6765, 8)
```

```
#Checking column names, data types, and memory usage.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6765 entries, 0 to 6764
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  6765 non-null   object
 1   open        6765 non-null   float64
 2   high        6765 non-null   float64
 3   low         6765 non-null   float64
 4   close       6765 non-null   float64
 5   adjclose    6765 non-null   float64
 6   volume      6765 non-null   int64
 7   ticker      6765 non-null   object
dtypes: float64(5), int64(1), object(2)
memory usage: 422.9+ KB
```

EVANGELIN PRIYANKA R
Jun 23, 2024
There is no null values

```
data.isnull().sum()
```

```
Unnamed: 0    0
open          0
high          0
low           0
close         0
adjclose      0
volume        0
ticker        0
dtype: int64
```

```
#Basic statistical summary which helps in understanding the range and distribution of your
data.describe()
```

	open	high	low	close	adjclose	
count	6765.000000	6765.000000	6765.000000	6765.000000	6765.000000	6.765
mean	3.836808	3.956002	3.715269	3.834242	3.834242	9.833
std	2.454250	2.524255	2.384410	2.453342	2.453342	5.112
min	0.609375	0.656250	0.500000	0.625000	0.625000	0.000
25%	2.000000	2.062500	1.910000	2.000000	2.000000	1.290
50%	3.090000	3.170000	3.000000	3.080000	3.080000	3.570
75%	4.880000	5.000000	4.690000	4.860000	4.860000	8.220
max	20.799999	21.639999	20.500000	20.770000	20.770000	3.349

```
# Checking skewness and kurtosis for open , close , high , low , adjclose, volume columns
skewness = data['open'].skew()
kurtosis = data['open'].kurt()

print("Skewness:\n", skewness)
print("Kurtosis:\n", kurtosis)
```

```
Skewness:
 1.5659648713357646
Kurtosis:
 3.231213480404883
```

```
skewness = data['close'].skew()
kurtosis = data['close'].kurt()

print("Skewness:\n", skewness)
print("Kurtosis:\n", kurtosis)
```

```
Skewness:
 1.5739747364694667
Kurtosis:
 3.2468310793484183
```

```
skewness = data['low'].skew()
kurtosis = data['low'].kurt()

print("Skewness:\n", skewness)
print("Kurtosis:\n", kurtosis)
```

```
Skewness:
 1.5420897535713816
Kurtosis:
 3.000143687962099
```

EVANGELIN PRIYANKA R
Jun 23, 2024

It is a positive skewness the value is greater than 0 means there are more frequent lower values of 'open' prices with occasional higher values that extend further to the right.

EVANGELIN PRIYANKA R
Jun 23, 2024

A kurtosis of 3.25 indicates higher kurtosis (excess kurtosis) compared to a normal distribution (which has a kurtosis of 3) that the 'close' price distribution has heavier tails and a sharper peak than a normal distribution

EVANGELIN PRIYANKA R
Jun 23, 2024

Positively skewed and follows normal distribution

EVANGELIN PRIYANKA R
Jun 23, 2024

moderately right-skewed with heavier tails and a sharper peak compared to a normal distribution, indicating occasional higher values and more extreme price movements.

EVANGELIN PRIYANKA R
Jun 23, 2024

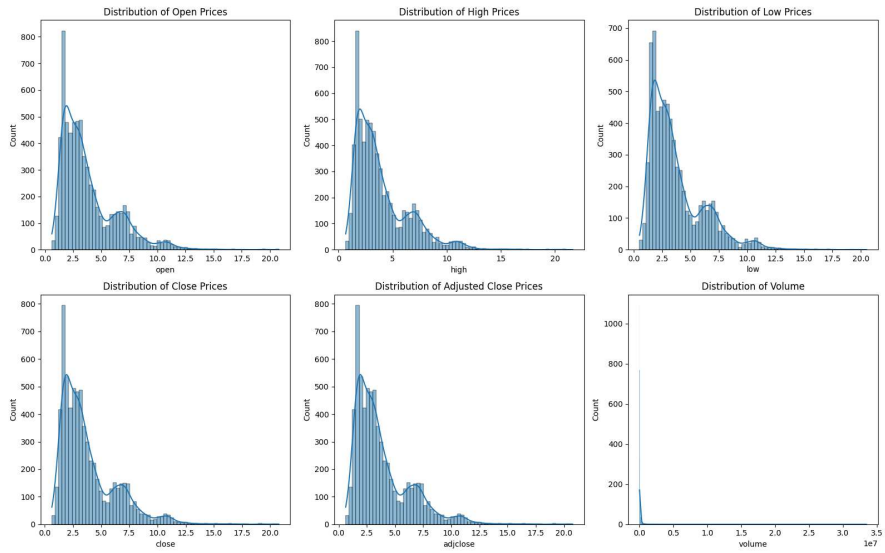
strongly right-skewed with significantly heavier tails and a sharper peak compared

```
skewness = data['high'].skew()
kurtosis = data['high'].kurt()

print("Skewness:\n", skewness)
print("Kurtosis:\n", kurtosis)
```

```
Skewness:
 1.6071510070147628
Kurtosis:
 3.5524649619736137
```

to a normal distribution, suggesting frequent lower values and occasional extreme high prices.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot distributions
plt.figure(figsize=(16, 10))

plt.subplot(2, 3, 1)
sns.histplot(data['open'], kde=True)
plt.title('Distribution of Open Prices')

plt.subplot(2, 3, 2)
sns.histplot(data['high'], kde=True)
plt.title('Distribution of High Prices')

plt.subplot(2, 3, 3)
sns.histplot(data['low'], kde=True)
plt.title('Distribution of Low Prices')

plt.subplot(2, 3, 4)
sns.histplot(data['close'], kde=True)
plt.title('Distribution of Close Prices')

plt.subplot(2, 3, 5)
sns.histplot(data['adjclose'], kde=True)
plt.title('Distribution of Adjusted Close Prices')

plt.subplot(2, 3, 6)
sns.histplot(data['volume'], kde=True)
plt.title('Distribution of Volume')

plt.tight_layout()
plt.show()
```



EVANGELIN PRIYANKA R
Jun 23, 2024

Checking outliers using IQR method & Boxplot

```
# Renaming the first column
data.rename(columns={'Unnamed: 0': 'Date'}, inplace=True)

print("DataFrame after renaming:")
print(data)
```

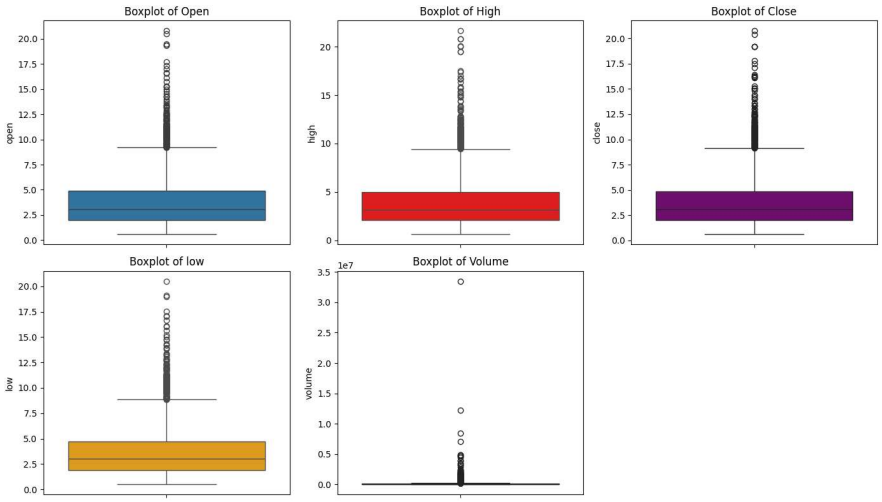
```
DataFrame after renaming:
            Date   open  high    low  close  adjclose   volume ticker
0     1997-08-01  9.250  9.25  8.250  8.500     8.500  1801600   EDAP
1     1997-08-04  8.500  8.75  8.000  8.125     8.125   140700   EDAP
2     1997-08-05  8.125  8.50  8.125  8.125     8.125    80200   EDAP
3     1997-08-06  8.500  8.50  8.125  8.500     8.500    38500   EDAP
4     1997-08-07  8.375  8.50  8.375  8.500     8.500   117400   EDAP
...          ...    ...   ...    ...    ...       ...      ...    ...
6760  2024-06-13  5.210  5.35  5.100  5.320     5.320     8600   EDAP
6761  2024-06-14  5.290  5.41  5.290  5.380     5.380     5400   EDAP
6762  2024-06-17  5.350  5.37  5.180  5.330     5.330    11700   EDAP
6763  2024-06-18  5.200  5.30  5.190  5.260     5.260    11700   EDAP
6764  2024-06-20  5.310  5.31  5.220  5.280     5.280     9500   EDAP

[6765 rows x 8 columns]
```

```
# Convert 'Date' column to datetime
data['Date'] = pd.to_datetime(data['Date'])
# Set 'Date' as the index for easier time series manipulation
data.set_index('Date', inplace=True)
```

```
#Checking for outliers through IQR method
Q1=data['open'].quantile(0.25)
Q3=data['open'].quantile(0.75)
IQR=Q3-Q1
print(f"iqr: {IQR}")
lower_bound=Q1-1.5*IQR
upper_bound=Q3+1.5*IQR
print(f"lower_bound: {lower_bound}")
print(f"upper_bound: {upper_bound}")
outliers=data[(data['open']<lower_bound)|(data['open']>upper_bound)]
print(f"Outliers: {outliers}")
```

```
iqr: 2.880000114440918
lower_bound: -2.320000171661377
upper_bound: 9.200000286102295
Outliers:              open   high   low  close  adjclose   volume ticker
Date
1997-08-01   9.25   9.25  8.25   8.50      8.50  1801600   EDAP
2006-03-03   9.50  10.15  8.96   9.90      9.90    98800   EDAP
2006-03-06   9.97  11.05  9.86  10.21     10.21   100100   EDAP
2006-03-07  10.01  10.37  9.75   9.75      9.75    50200   EDAP
2006-03-08   9.84   9.85  8.60   9.51      9.51   115200   EDAP
...           ...    ...   ...    ...       ...      ...    ...
2023-06-06   9.95   9.97  9.51   9.71      9.71   104000   EDAP
2023-06-07   9.66   9.92  9.50   9.53      9.53    56500   EDAP
2023-06-08   9.60   9.60  8.73   9.03      9.03   207900   EDAP
2023-06-14   9.23   9.23  8.90   9.07      9.07   175400   EDAP
2023-07-05   9.34   9.53  9.03   9.20      9.20    22300   EDAP

[257 rows x 7 columns]
```

```
# Create boxplots for Open, High, Low, Close, and Volume
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

plt.figure(figsize=(14, 8))
plt.subplot(2, 3, 1)
sns.boxplot(y=data['open'])
plt.title('Boxplot of Open')

plt.subplot(2, 3, 2)
sns.boxplot(y=data['high'], color='red')
plt.title('Boxplot of High')

plt.subplot(2, 3, 3)
sns.boxplot(y=data['close'], color='purple')
plt.title('Boxplot of Close')

plt.subplot(2, 3, 4)
sns.boxplot(y=data['low'], color='orange')
plt.title('Boxplot of low')

plt.subplot(2, 3, 5)
sns.boxplot(y=data['volume'], color='green')
plt.title('Boxplot of Volume')

plt.tight_layout()
plt.show()
```




```
import pandas as pd
# Check if there are any rows where 'high' is less than 'low'
inconsistent_rows = data[data['high'] < data['low']]

# Print inconsistent rows (if any)
if not inconsistent_rows.empty:
    print("Inconsistent rows found where 'high' is less than 'low':")
    print(inconsistent_rows)
else:
    print("No inconsistent rows found where 'high' is less than 'low'.")
```

```
No inconsistent rows found where 'high' is less than 'low'.
```

```
# Verify that 'close' prices are within 'high' and 'low' prices
inconsistent_close = data[(data['close'] < data['low']) | (data['close'] > data['high'])]

# Verify that 'volume' values are non-negative
negative_volume = data[data['volume'] < 0]

# Print results of checks
if not inconsistent_close.empty:
    print("Inconsistent 'close' prices found:")
    print(inconsistent_close)
else:
    print("No inconsistent 'close' prices found.")

if not negative_volume.empty:
    print("Negative 'volume' values found:")
    print(negative_volume)
else:
    print("No negative 'volume' values found.")
```

```
No inconsistent 'close' prices found.
No negative 'volume' values found.
```

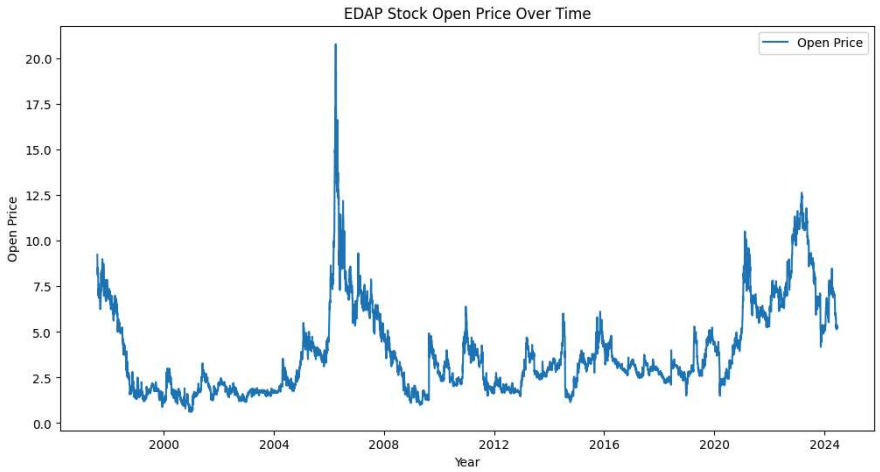
```
# correlation matrix
correlation_matrix = data[['open', 'high', 'low', 'close', 'adjclose', 'volume']].corr()
print(correlation_matrix)

# Plot correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='Dark2', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

```
                open      high       low     close  adjclose    volume
open        1.000000  0.997660  0.997848  0.996560  0.996560  0.040014
high        0.997660  1.000000  0.996640  0.998246  0.998246  0.053298
low         0.997848  0.996640  1.000000  0.998110  0.998110  0.030705
close       0.996560  0.998246  0.998110  1.000000  1.000000  0.040043
adjclose    0.996560  0.998246  0.998110  1.000000  1.000000  0.040043
volume      0.040014  0.053298  0.030705  0.040043  0.040043  1.000000
```

EVANGELIN PRIYANKA R
Jun 23, 2024

'Close ' column indicates the price at which the stock finished trading when the market closed on that particular day.

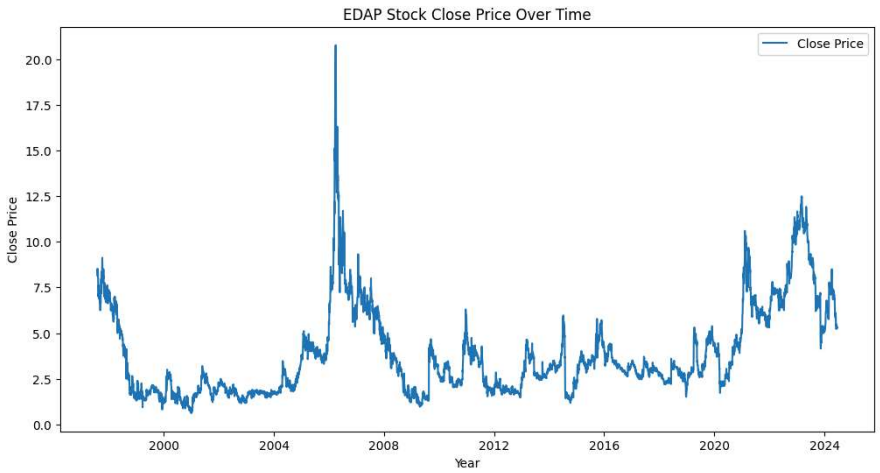
```
# Checking Stock Open price over Time
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['open'], label='Open Price')
plt.title('EDAP Stock Open Price Over Time')
plt.xlabel('Year')
plt.ylabel('Open Price')
plt.legend()
plt.show()
```



EVANGELIN PRIYANKA R
Jun 23, 2024

Showing Open price and Close price for all the years simultaneously

```
# Checking Stock close price over Time
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['close'], label='Close Price')
plt.title('EDAP Stock Close Price Over Time')
plt.xlabel('Year')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

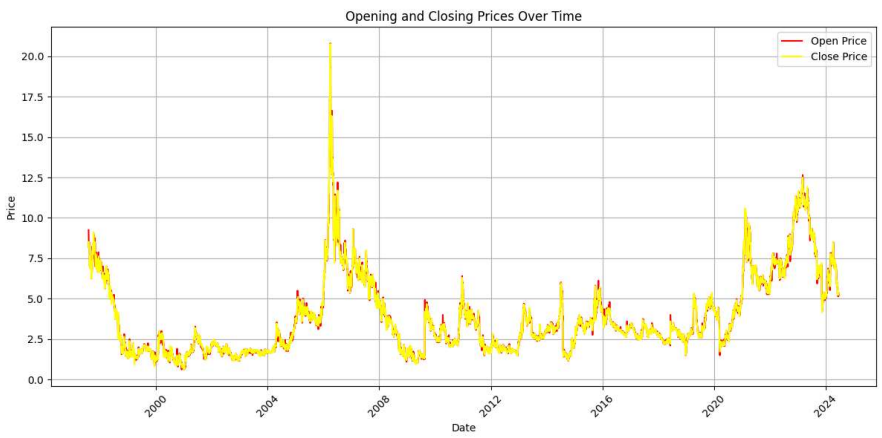


EVANGELIN PRIYANKA R
Jun 23, 2024
(edited Jun 23, 2024)

The annual stock performance of the open price decreased for the year 2023 to 2024 indicating a significant downward trend in the stock's value over that period. Drop in stock performance can be due to different

factors like economic factors , poor financial performance etc

```
# Convert index to a 'Date' column if it contains the dates
if isinstance(data.index, pd.DatetimeIndex):
    data['Date'] = data.index

# Convert 'Date' column to datetime format if it exists
if 'Date' in data.columns:
    data['Date'] = pd.to_datetime(data['Date'])

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['open'], marker='', linestyle='-', color='red', label='Open Pri
plt.plot(data['Date'], data['close'], marker='', linestyle='-', color='yellow', label='Clos
plt.title('Opening and Closing Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()

plt.show()
```

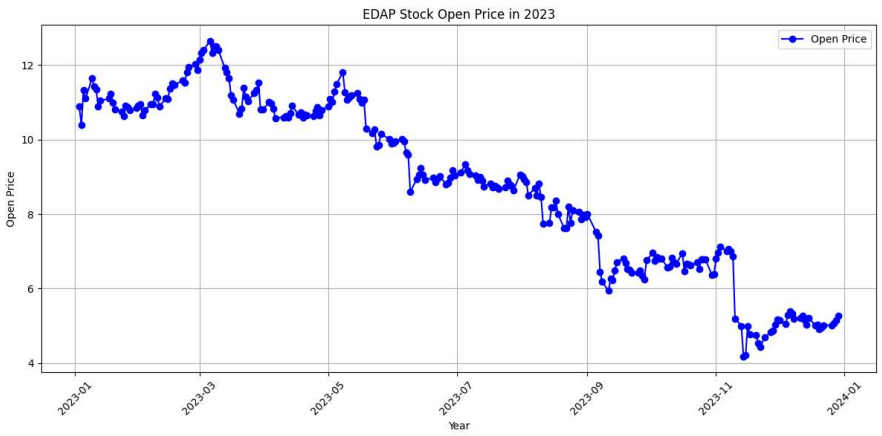


EVANGELIN PRIYANKA R
Jun 27, 2024

represent price movements of an asset, such as a stock, over a specific period

```
# Convert 'Date' column to datetime format

if 'Date' in data.columns:
    data['Date'] = pd.to_datetime(data['Date'])

# Filter data for one year
year = 2023
# Access the year from the index since 'Date' is likely the index
data_year = data[data.index.year == year]
```

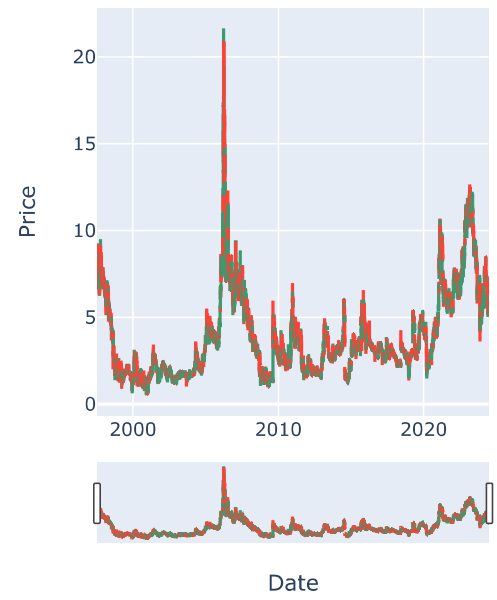
```
# Plotting open price over the selected year
plt.figure(figsize=(12, 6))
# Access the date from the index directly
plt.plot(data_year.index, data_year['open'], marker='o', linestyle='-', color='b', label='O
plt.title(f'EDAP Stock Open Price in {year}')
plt.xlabel('Year')
plt.ylabel('Open Price')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()

plt.show()
```



EVANGELIN PRIYANKA R
Jun 23, 2024

plotting the AdjClose prices over time to visualize the stock's performance.

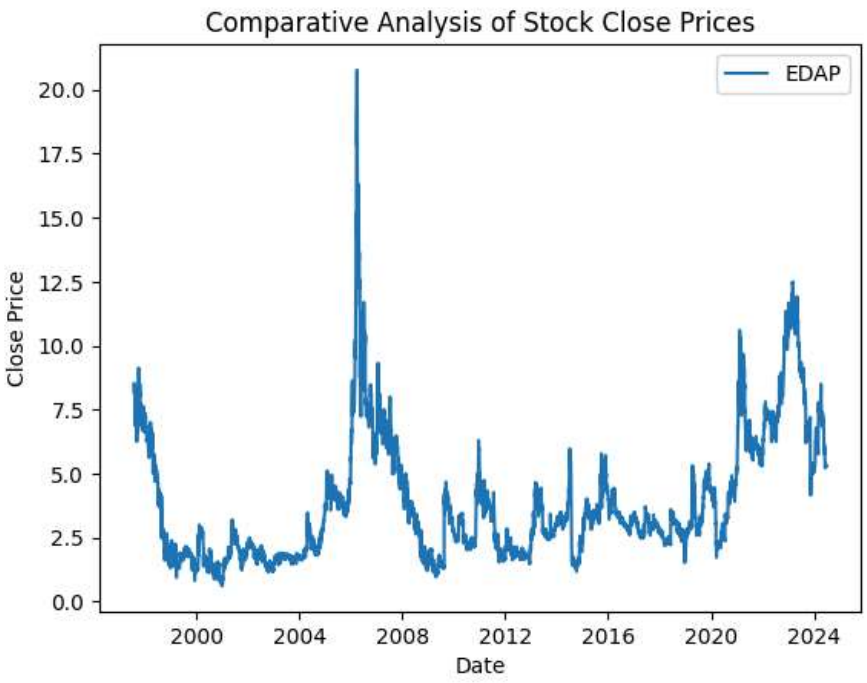
```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Candlestick(x=data['Date'],
                                     open=data['open'],
                                     high=data['high'],
                                     low=data['low'],
                                     close=data['close'])])
fig.update_layout(title='Candlestick Chart',
                  xaxis_title='Date',
                  yaxis_title='Price')
fig.show()
```



EVANGELIN PRIYANKA R
Jun 23, 2024

Yearly returns can be calculated using the Adjusted Close price.

```
# Assuming data contains multiple stocks identified by 'Ticker'
sns.lineplot(data=data, x='Date', y='close', hue='ticker')
plt.title('Comparative Analysis of Stock Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



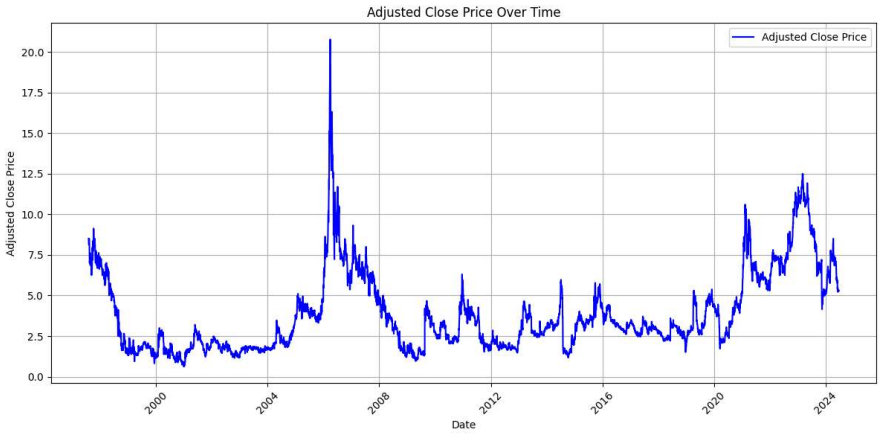
EVANGELIN PRIYANKA R
Jun 23, 2024

Calculating and plotting moving averages to identify trends and smooth out short-term fluctuations.

EVANGELIN PRIYANKA R
Jun 23, 2024

When the 20-day MA crosses above the 50-day MA (Golden Cross), it can be a signal to buy.
When the 20-day MA crosses below the 50-day MA (Death Cross), it can be a signal to sell.

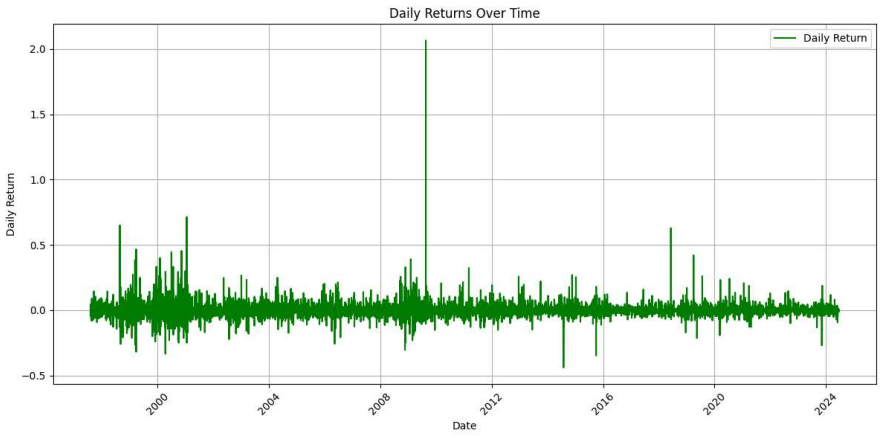
```
if 'Date' in data.columns:
    data['Date'] = pd.to_datetime(data['Date'])

# Plotting the Adjusted Close Prices over time
plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['adjclose'], label='Adjusted Close Price', color='blue')
plt.title('Adjusted Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



EVANGELIN PRIYANKA R
Jun 23, 2024

Trend Analysis
Decompose the time series into trend, seasonality, and residuals to understand underlying patterns.

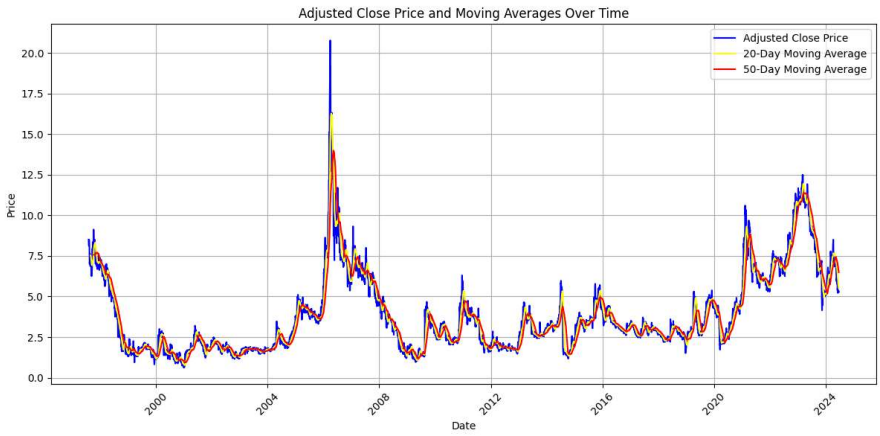
```
# Calculate daily returns
data['Daily Return'] = data['adjclose'].pct_change()

# Plotting daily returns
plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['Daily Return'], label='Daily Return', color='green')
plt.title('Daily Returns Over Time')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



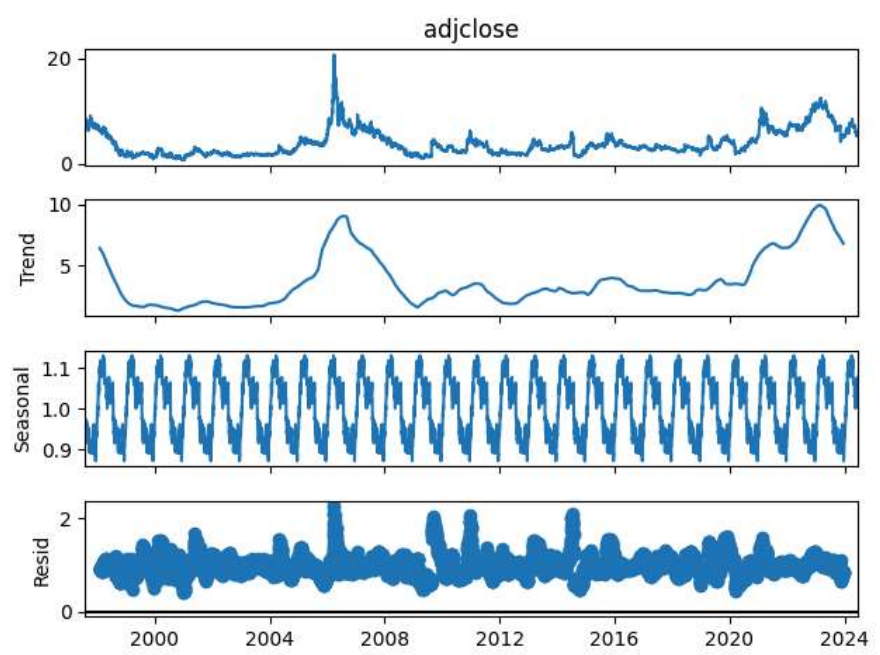

```
# Calculate moving averages
data['20-Day MA'] = data['adjclose'].rolling(window=20).mean()
data['50-Day MA'] = data['adjclose'].rolling(window=50).mean()

# Plotting adjusted close price with moving averages
plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['adjclose'], label='Adjusted Close Price', color='blue')
plt.plot(data['Date'], data['20-Day MA'], label='20-Day Moving Average', color='yellow')
plt.plot(data['Date'], data['50-Day MA'], label='50-Day Moving Average', color='red')
plt.title('Adjusted Close Price and Moving Averages Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

EVANGELIN PRIYANKA R
Jun 27, 2024

Rolling mean, we can identify the stock's long-term price trend. The standard deviation tells us how much the price fluctuates around this trend. Higher standard deviation indicates a more volatile stock.



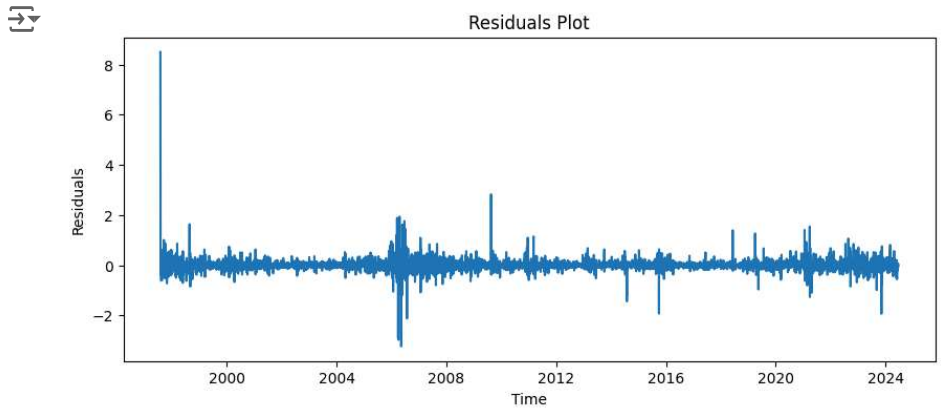

```
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(data['adjclose'], model='multiplicative', period=252)  # Assumi

# Plotting the decomposition
result.plot()
plt.show()
```



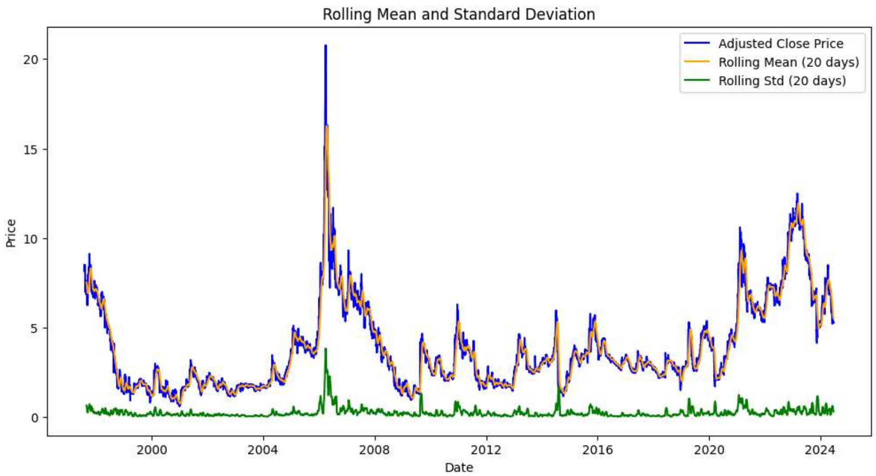
EVANGELIN PRIYANKA R
Jun 27, 2024

A lower AIC value indicates a better model fit relative to other models. lower BIC value indicates a better model fit. BIC includes a penalty term for the number of parameters in the model to prevent overfitting. A lower RMSE value indicates better predictive accuracy.

```
#plot residuals
residuals = model_fit.resid
plt.figure(figsize=(10, 4))
plt.plot(residuals)
plt.title('Residuals Plot')
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.show()
```




```
data['Rolling Mean'] = data['adjclose'].rolling(window=20).mean()
data['Rolling Std'] = data['adjclose'].rolling(window=20).std()

plt.figure(figsize=(12, 6))
plt.plot(data['Date'], data['adjclose'], label='Adjusted Close Price', color='blue')
plt.plot(data['Date'], data['Rolling Mean'], label='Rolling Mean (20 days)', color='orange'
plt.plot(data['Date'], data['Rolling Std'], label='Rolling Std (20 days)', color='green')
plt.title('Rolling Mean and Standard Deviation')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import numpy as np

# Fit ARIMA model
p, d, q = 5, 1, 0
model = ARIMA(data['adjclose'], order=(p, d, q))
model_fit = model.fit()

# Calculate AIC and BIC
aic = model_fit.aic
bic = model_fit.bic

# Forecast
train_size = int(len(data) * 0.8)  # Using 80% of data for training
train, test = data['adjclose'][:train_size], data['adjclose'][train_size:]

# Fit model on training data
model = ARIMA(train, order=(p, d, q))
model_fit = model.fit()

# Make predictions
predictions = model_fit.forecast(steps=len(test))

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(test, predictions))
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:
  return get_prediction_index(
```

```
print(f"AIC: {aic}")
print(f"BIC: {bic}")
print(f"RMSE: {rmse}")
```

```
AIC: -2006.5732194261936
BIC: -1965.6570011660829
RMSE: 4.353512245559313
```