1. An Introduction to Dataset

In this analytical report, we delve into the intricate landscape of Netflix's stock prices, utilizing a comprehensive dataset spanning from 2023 to 2024. This introductory section provides a brief overview of the dataset, its source, contents, and outlines the analytical approach employed for the exploratory data analysis (EDA).

1.1 About the Dataset

The dataset at the core of this analysis encapsulates a wealth of information concerning Netflix's stock prices. With meticulous records from 2023 to 2024, it serves as a valuable resource for dissecting the fluctuations, trends, and nuances inherent in the stock market dynamics surrounding this streaming giant.

1.2 Source of Data

The dataset is meticulously curated from reliable sources, ensuring accuracy and integrity in its representation of Netflix's stock performance. Through diligent aggregation and verification processes, the data source guarantees a robust foundation for our analytical endeavors.

1.3 Contents of Data

Encompassing various facets of Netflix's stock market journey, the dataset comprises essential columns elucidating crucial aspects of stock pricing. These include:

Date: Recording the trading days over the specified time frame.

Open: Signifying the opening price of Netflix stock on each trading day.

High: Reflecting the highest price at which Netflix stock traded during the day.

Low: Denoting the lowest price at which Netflix stock traded during the day.

Close: Indicating the closing price of Netflix stock on each trading day.

Adjustment Close: Presenting the adjusted closing price, accounting for events such as dividends and stock splits.

Volume: Representing the total number of shares traded on each trading day.

1.4 Analytical Approach

The analytical approach adopted for this analysis primarily revolves around Exploratory Data Analysis (EDA). Through rigorous examination and visualization of the dataset, we aim to unearth insightful patterns, trends, and relationships within Netflix's stock prices. By scrutinizing the data from multiple angles, we endeavor to extract actionable insights that can inform strategic decision-making and deepen our understanding of the dynamics influencing Netflix's stock performance.

2. Overview of The Data

I'll kick off our data exploration journey by summoning the mighty Pandas Library, a powerful ally renowned for its prowess in wielding datasets. Together, we'll embark on a quest to uncover the secrets hidden within our data, shedding light on its enigmatic depths and revealing the stories it yearns to tell.

```
import pandas as pd
netflix = pd.read_csv("NFLX.csv") # Load the dataset and assign to variable netflix

# Read the first 10 rows of the dataset
print("First 10 rows of the dataset:")
netflix.head(11)
```

```
First 10 rows of the dataset:

          Date        Open        High         Low       Close    Adj
Close  \
0   2023-02-01  353.859985  365.390015  349.910004  361.989990
361.989990
1   2023-02-02  365.160004  368.320007  358.429993  366.890015
366.890015
2   2023-02-03  359.079987  379.429993  359.000000  365.899994
365.899994
3   2023-02-06  363.640015  368.450012  360.679993  361.480011
361.480011
4   2023-02-07  358.510010  364.179993  354.179993  362.950012
362.950012
5   2023-02-08  360.019989  368.190002  358.309998  366.829987
366.829987
6   2023-02-09  372.410004  373.829987  361.739990  362.500000
362.500000
7   2023-02-10  359.160004  362.140015  347.140015  347.359985
347.359985
8   2023-02-13  349.500000  359.700012  344.250000  358.570007
358.570007
9   2023-02-14  357.549988  363.750000  353.399994  359.959991
359.959991
10  2023-02-15  356.630005  362.880005  354.239990  361.420013
361.420013

     Volume
0   8005200
1   7857000
2   9402000
3   4994900
4   6289400
5   6253200
6   6901100
```

```
7   7291100
8   7134400
9   4624800
10  3966000
```

```python
# Read the last 10 rows of the dataset
print("Last 10 rows of the dataset:")
netflix.tail(11)
```

```
Last 10 rows of the dataset:

           Date        Open        High         Low       Close   Adj
Close  \
240  2024-01-17  484.500000  486.209991  475.260010  480.329987
480.329987
241  2024-01-18  480.029999  485.769989  478.019989  485.309998
485.309998
242  2024-01-19  484.980011  485.670013  476.059998  482.950012
482.950012
243  2024-01-22  487.549988  489.799988  479.899994  485.709991
485.709991
244  2024-01-23  492.000000  498.959991  481.399994  492.190002
492.190002
245  2024-01-24  537.750000  562.500000  537.070007  544.869995
544.869995
246  2024-01-25  551.950012  563.460022  548.460022  562.000000
562.000000
247  2024-01-26  561.809998  579.640015  558.429993  570.419983
570.419983
248  2024-01-29  571.349976  578.549988  562.679993  575.789978
575.789978
249  2024-01-30  567.320007  570.880005  560.820007  562.849976
562.849976
250  2024-01-31  562.849976  572.150024  562.039978  564.109985
564.109985

       Volume
240   4894600
241   4054400
242   5665600
243   5212300
244  15506000
245  26432800
246   9451900
247  12754500
248   6905400
249   6181800
250   4857600
```

```python
# Setting display options to show all values without scientific
notation
pd.set_option('display.float_format', lambda x: '%.2f' % x)

# Now when call describe(), the volume column should be displayed
without scientific notation of 'e'
print("\nDescriptive statistics of the dataset:")
netflix.describe()
```

```
Descriptive statistics of the dataset:
```

	Open	High	Low	Close	Adj Close	Volume
count	251.00	251.00	251.00	251.00	251.00	251.00
mean	404.18	409.75	398.96	404.27	404.27	6135307.97
std	60.85	61.32	60.72	61.19	61.19	3814621.40
min	287.34	297.45	285.33	292.76	292.76	1404700.00
25%	348.99	356.86	344.49	348.12	348.12	3966000.00
50%	412.00	418.84	407.40	411.69	411.69	5128900.00
75%	444.73	448.57	439.18	444.94	444.94	6880600.00
max	571.35	579.64	562.68	575.79	575.79	28074400.00

```python
# See the Rows and Columns of the dataset

netflix.shape
```

```
(251, 7)
```

```python
# Information on the dataset
print("\nInformation about the DataFrame after converting 'Date'
column to datetime format:")
print("\n")
netflix.info()
```

```
Information about the DataFrame after converting 'Date' column to
datetime format:


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 251 entries, 0 to 250
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       251 non-null    object
 1   Open       251 non-null    float64
 2   High       251 non-null    float64
 3   Low        251 non-null    float64
 4   Close      251 non-null    float64
 5   Adj Close  251 non-null    float64
 6   Volume     251 non-null    int64
```

```
dtypes: float64(5), int64(1), object(1)
memory usage: 13.9+ KB
```

Here, we can see that the *Date* column is *object*. I will be converting it to *datetime64[ns]* format for better allotment of the DataType

```
netflix['Date'] = pd.to_datetime(netflix['Date'])

# Check the information about the DataFrame to see if changes were
made
netflix.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 251 entries, 0 to 250
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       251 non-null    datetime64[ns]
 1   Open       251 non-null    float64
 2   High       251 non-null    float64
 3   Low        251 non-null    float64
 4   Close      251 non-null    float64
 5   Adj Close  251 non-null    float64
 6   Volume     251 non-null    int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 13.9 KB
```

3. Data Cleaning

Now that we've unveiled the initial layers of our dataset, it's time to polish our treasure trove and ensure its integrity for further exploration. In this phase of our adventure, we'll embark on the noble quest of data cleaning, where we'll sift through the sands of our dataset, vanquishing inconsistencies, taming outliers, and smoothing rough edges. Our goal? To sculpt our data into a pristine masterpiece, ready to yield its hidden gems at our command.

```
# See if our data contains any 'null' values

netflix.notnull()

# This will return a Boolean series either True (if our data does not
contain missing values) or
# False (if there are Missing Values)

     Date  Open  High   Low  Close  Adj Close  Volume
0    True  True  True  True   True       True    True
1    True  True  True  True   True       True    True
2    True  True  True  True   True       True    True
3    True  True  True  True   True       True    True
4    True  True  True  True   True       True    True
```

```
..    ...   ...   ...   ...    ...         ...     ...
246  True  True  True  True   True        True    True
247  True  True  True  True   True        True    True
248  True  True  True  True   True        True    True
249  True  True  True  True   True        True    True
250  True  True  True  True   True        True    True

[251 rows x 7 columns]
```

```
# Calculate the number of null values in each column
null_counts = netflix.isnull().sum()

# Display the number of null values in each column
print("Number of null values in each column:")
print(null_counts)
```

```
Number of null values in each column:
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

```
# Check Duplication

netflix.duplicated() # Return boolean Series denoting duplicate rows.
False means no duplication
```

```
0      False
1      False
2      False
3      False
4      False
       ...
246    False
247    False
248    False
249    False
250    False
Length: 251, dtype: bool
```

```
# Calculate the total number of null values in the DataFrame
total_null_values = null_counts.sum()

# Display the total number of null values
print("Total number of null values in the DataFrame:",
total_null_values)
```

```
Total number of null values in the DataFrame: 0

# Check if our Dataset contains NaN or Null Values

netflix.isna()
# This will return Boolean Series; True if NaN exist or False if NaN
does Not exist

      Date   Open   High    Low  Close  Adj Close  Volume
0    False  False  False  False  False      False   False
1    False  False  False  False  False      False   False
2    False  False  False  False  False      False   False
3    False  False  False  False  False      False   False
4    False  False  False  False  False      False   False
..     ...    ...    ...    ...    ...        ...     ...
246  False  False  False  False  False      False   False
247  False  False  False  False  False      False   False
248  False  False  False  False  False      False   False
249  False  False  False  False  False      False   False
250  False  False  False  False  False      False   False

[251 rows x 7 columns]

# Drop the NaN values

netflix.dropna()

           Date   Open   High    Low  Close  Adj Close    Volume
0   2023-02-01 353.86 365.39 349.91 361.99     361.99   8005200
1   2023-02-02 365.16 368.32 358.43 366.89     366.89   7857000
2   2023-02-03 359.08 379.43 359.00 365.90     365.90   9402000
3   2023-02-06 363.64 368.45 360.68 361.48     361.48   4994900
4   2023-02-07 358.51 364.18 354.18 362.95     362.95   6289400
..         ...    ...    ...    ...    ...        ...       ...
246 2024-01-25 551.95 563.46 548.46 562.00     562.00   9451900
247 2024-01-26 561.81 579.64 558.43 570.42     570.42  12754500
248 2024-01-29 571.35 578.55 562.68 575.79     575.79   6905400
249 2024-01-30 567.32 570.88 560.82 562.85     562.85   6181800
250 2024-01-31 562.85 572.15 562.04 564.11     564.11   4857600

[251 rows x 7 columns]
```

See the shape is still 251 rows and 7 columns which is exactly the same as that done above on the **Overview Heading**, using `netflix.shape`. This implies that There were no Missing Values in the dataset

4. Time Series Analysis

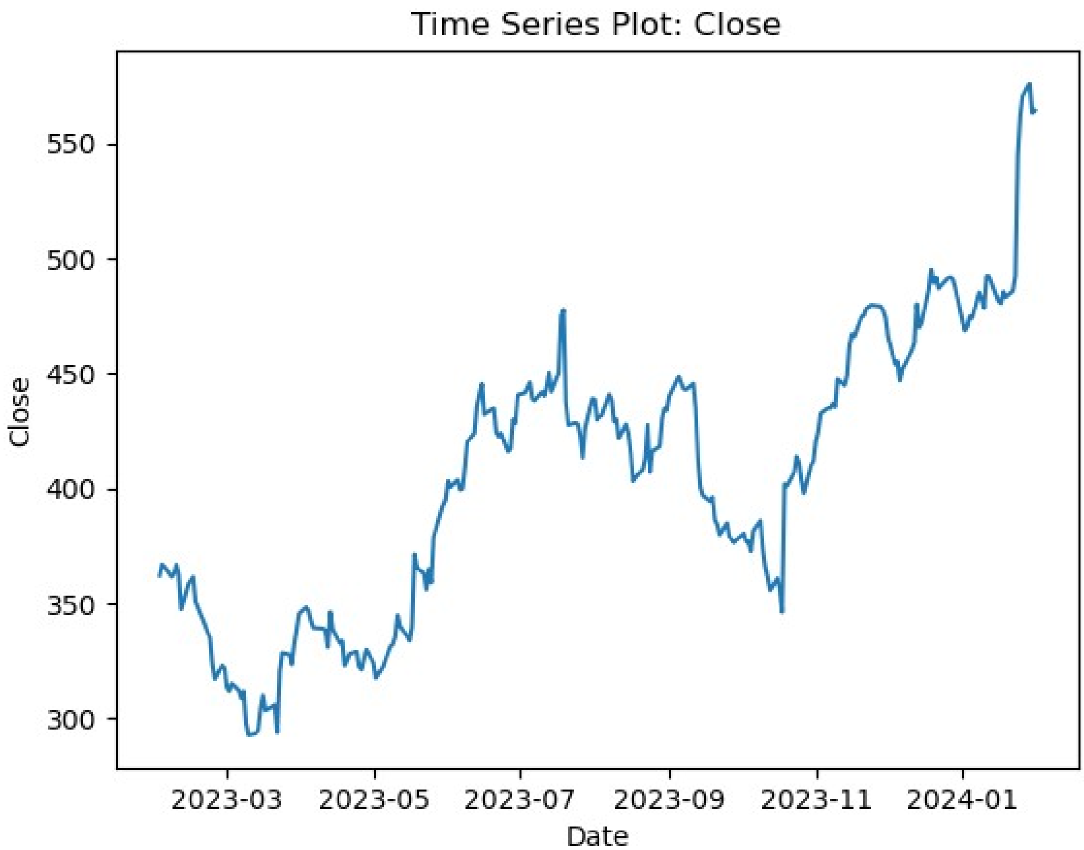
Time series analysis is a powerful statistical technique used to analyze and interpret data points collected at successive intervals of time. Unlike traditional cross-sectional data analysis, which focuses on observations at a single point in time, time series analysis delves into the temporal patterns, trends, and dependencies present in the data. It finds applications across various domains, including finance, economics, meteorology, and engineering, where understanding the behavior of a phenomenon over time is crucial for decision-making and prediction. Time series data often exhibits unique characteristics such as trend, seasonality, autocorrelation, and volatility, which necessitate specialized analytical techniques for exploration and forecasting. Through time series analysis, practitioners can uncover valuable insights, detect anomalies, model relationships, and make informed predictions about future outcomes, thereby enabling proactive decision-making and strategic planning. This introductory paragraph sets the stage for a deeper dive into the methods and tools used in time series analysis, empowering analysts to extract meaningful insights and derive actionable conclusions from temporal data.

4.1 Time Series Visualization

```
import matplotlib.pyplot as plt

def time_series_plot(dataframe):
    '''
    This function generates a time series plot based on the specified
column.

    Args:
        dataframe: The DataFrame containing the time series data.

    Returns:
        None
    '''
    # Get the date column from the DataFrame
    x = dataframe['Date']

    # Get the column name for plotting (e.g., Open, Close, Volume)
    column_name = input("Enter the column name to plot the line graph:
")

    # Check if the specified column exists in the DataFrame
    if column_name in dataframe.columns:
        # Get the values for the specified column
        y = dataframe[column_name]

        # Plot the time series
        plt.plot(x, y)
        plt.xlabel('Date')
        plt.ylabel(column_name)
        plt.title('Time Series Plot: ' + column_name)
        plt.show()
    else:
        print("Error: The specified column does not exist in the
DataFrame.")
```

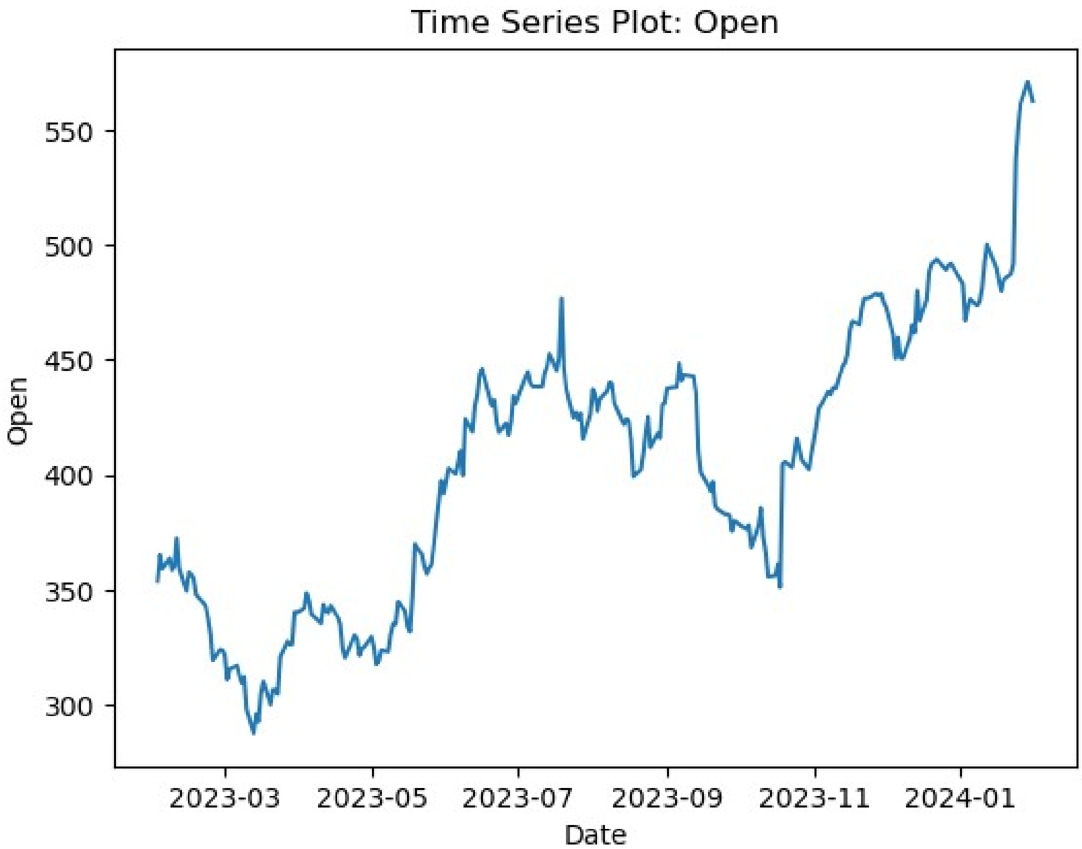
```
# Call the function with your DataFrame
time_series_plot(netflix)
```

```
Enter the column name to plot the line graph:  Close
```

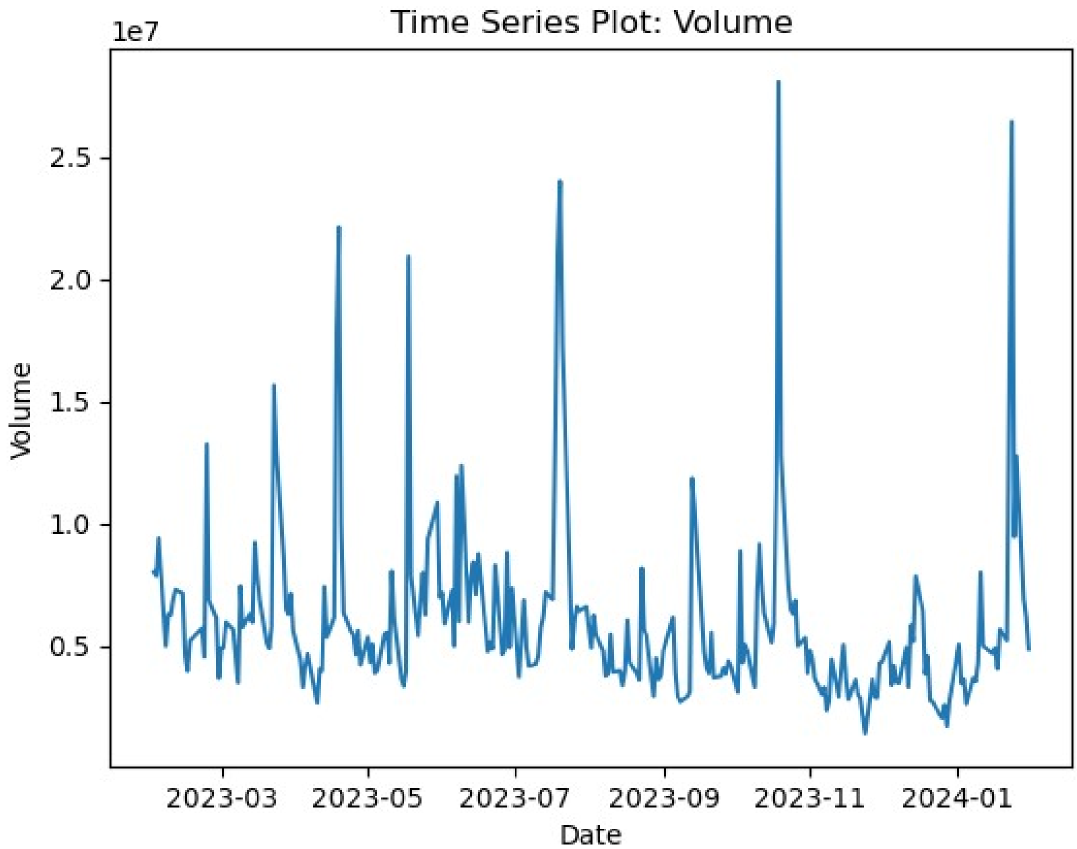



```
time_series_plot(netflix)
```

```
Enter the column name to plot the line graph:  Open
```



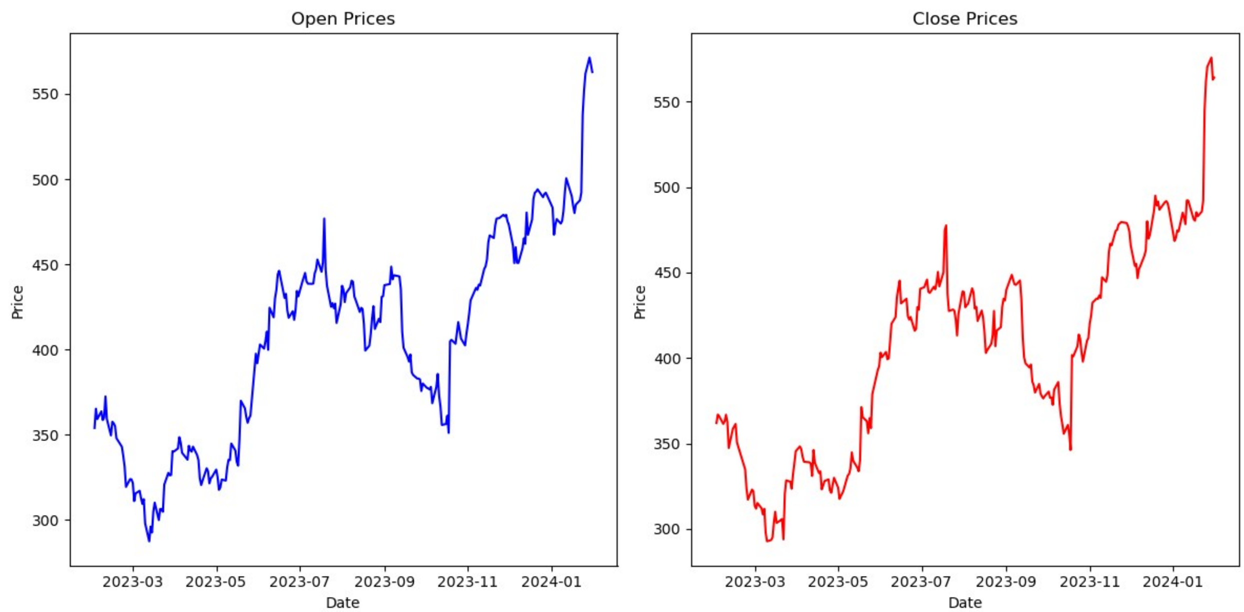

```
time_series_plot(netflix)

Enter the column name to plot the line graph:  Volume
```



Here, we can see that *Open* and *Close* are almost of the same pattern. Lets see a side by side comparision to identify any insights, if any.

```
def time_series_plot(dataframe):
    '''
    This function generates a side-by-side comparison of time series plots for the "Open" and "Close" columns.

    Args:
        dataframe: The DataFrame containing the time series data.

    Returns:
        None
    '''
    # Get the date column from the DataFrame
    x = dataframe['Date']

    # Get the values for the "Open" and "Close" columns
    open_prices = dataframe['Open']
    close_prices = dataframe['Close']

    # Create a figure and two subplots side by side
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))
```

```
    # Plot the time series for the "Open" column
    axs[0].plot(x, open_prices, color='blue')
    axs[0].set_title('Open Prices')
    axs[0].set_xlabel('Date')
    axs[0].set_ylabel('Price')

    # Plot the time series for the "Close" column
    axs[1].plot(x, close_prices, color='red')
    axs[1].set_title('Close Prices')
    axs[1].set_xlabel('Date')
    axs[1].set_ylabel('Price')

    # Adjusting layout to prevent overlap of labels
    plt.tight_layout()

    # Display the plot
    plt.show()

# Call the function with your DataFrame
time_series_plot(netflix)
```



Upon examining the side-by-side comparison of the "Open" and "Close" time series plots, it becomes evident that the patterns exhibited by both columns are strikingly similar. However, subtle differences can be observed between the two. In the "Open" plot, the price initiates with a slight uptick at the beginning of the trading day, indicating an initial surge in market activity. Conversely, in the "Close" plot, the price experiences a marginal decline towards the end of the trading day, reflecting a potential decrease in market sentiment as trading comes to a close. Despite these minor discrepancies, the overarching trends and fluctuations in both plots align closely, suggesting a strong correlation between the opening and closing prices of the Netflix stock over the analyzed time period.

4.2 Descriptive Statistics

```
import pandas as pd

def descriptive_statistics(dataframe):
    '''
    This function calculates descriptive statistics for the "Open" and
"Close" columns of the given DataFrame.

    Args:
        dataframe: The DataFrame containing the time series data.

    Returns:
        open_stats: A dictionary containing descriptive statistics for
the "Open" column.
        close_stats: A dictionary containing descriptive statistics
for the "Close" column.
    '''
    # Calculate descriptive statistics for the "Open" column
    open_stats = dataframe['Open'].describe()

    # Calculate descriptive statistics for the "Close" column
    close_stats = dataframe['Close'].describe()

    return open_stats, close_stats

# Call the function with your DataFrame
open_stats, close_stats = descriptive_statistics(netflix)

# Print the descriptive statistics
print("Descriptive Statistics for the 'Open' column:")
print(open_stats)
print("\nDescriptive Statistics for the 'Close' column:")
print(close_stats)
```

```
Descriptive Statistics for the 'Open' column:
count   251.00
mean    404.18
std      60.85
min     287.34
25%     348.99
50%     412.00
75%     444.73
max     571.35
Name: Open, dtype: float64

Descriptive Statistics for the 'Close' column:
count   251.00
```

```
mean    404.27
std      61.19
min     292.76
25%     348.12
50%     411.69
75%     444.94
max     575.79
Name: Close, dtype: float64
```

4.3 Correlation Matrix

```
def correlation_analysis(dataframe):
    '''
    This function performs correlation analysis between the "Open",
"High", "Low", and "Close" columns.

    Args:
        dataframe: The DataFrame containing the time series data.

    Returns:
        correlations: A DataFrame containing the correlation
coefficients between the columns.
    '''
    # Select the columns for correlation analysis
    columns_of_interest = ['Open', 'High', 'Low', 'Close']

    # Calculate the correlation matrix
    correlations = dataframe[columns_of_interest].corr()

    return correlations

# Call the function with your DataFrame
correlations = correlation_analysis(netflix)

# Print the correlation matrix
print("Correlation Matrix:")
print(correlations)

Correlation Matrix:
       Open  High  Low  Close
Open   1.00  1.00 1.00   0.99
High   1.00  1.00 1.00   1.00
Low    1.00  1.00 1.00   1.00
Close  0.99  1.00 1.00   1.00
```

Description of Insights:

- The correlation matrix reveals the correlation coefficients between the "Open", "High", "Low", and "Close" prices of Netflix shares.

- A correlation coefficient ranges from -1 to 1, where 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship.
- In this correlation matrix:
  - The diagonal elements represent the correlation of each variable with itself, which is always 1.
  - Off-diagonal elements represent the correlations between pairs of variables.
- The correlation coefficients close to 1 suggest a strong positive linear relationship between the corresponding pairs of variables.
- Specifically:
  - The "Open" price is highly correlated with the "High", "Low", and "Close" prices, with correlation coefficients of approximately 1.
  - Similarly, the "High", "Low", and "Close" prices exhibit strong positive correlations among themselves, all close to 1.
- The high correlations between these variables indicate that they move in tandem, which is typical in financial time series data where the opening, high, low, and closing prices are closely related.

4.4 Highest and Lowest Share Prices

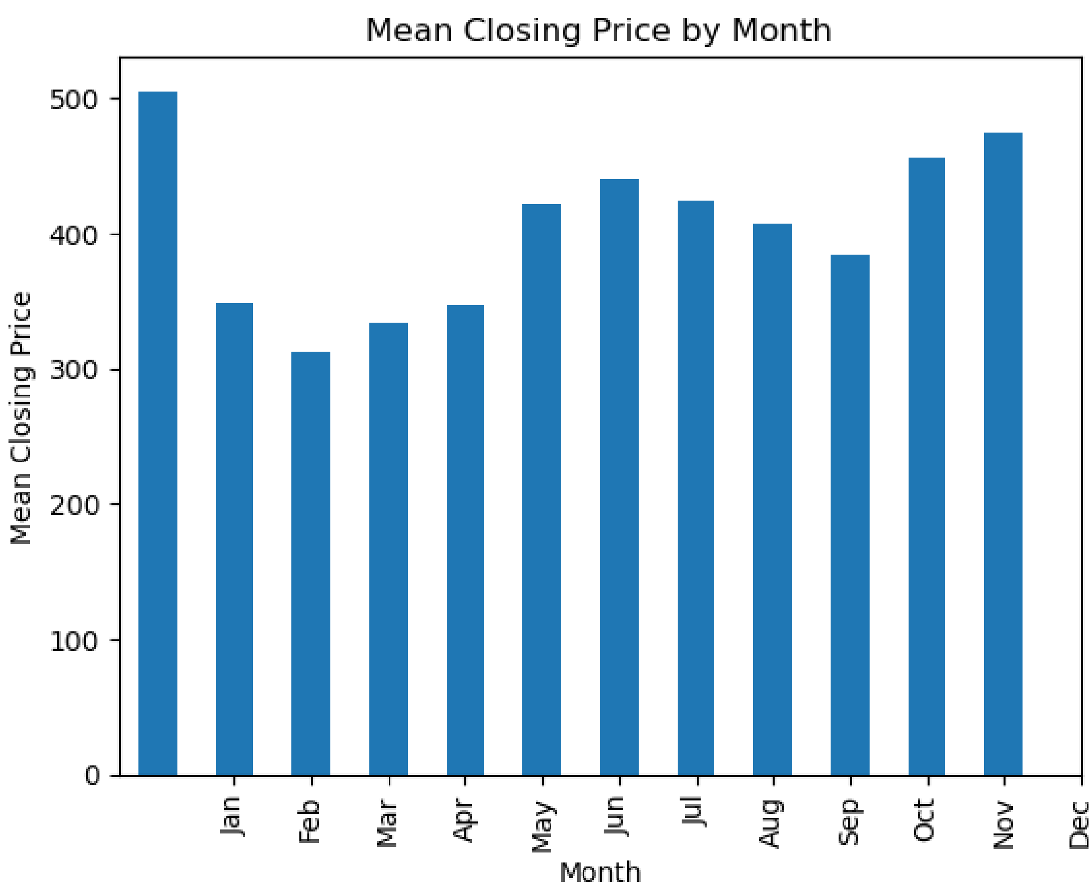
```
# Find the row with the highest closing price
max_close_row = netflix.loc[netflix['Close'].idxmax()]

# Find the row with the lowest closing price
min_close_row = netflix.loc[netflix['Close'].idxmin()]

# Find the row with the highest volume
max_volume_row = netflix.loc[netflix['Volume'].idxmax()]

# Find the row with the lowest volume
min_volume_row = netflix.loc[netflix['Volume'].idxmin()]

# Print the results
print("Date with the highest closing price:", max_close_row['Date'],
"| Price:", max_close_row['Close'])
print("Date with the lowest closing price:", min_close_row['Date'], "|
Price:", min_close_row['Close'])
print("Date with the highest volume:", max_volume_row['Date'], "|
Volume:", max_volume_row['Volume'])
print("Date with the lowest volume:", min_volume_row['Date'], "|
Volume:", min_volume_row['Volume'])
```

```
Date with the highest closing price: 2024-01-29 00:00:00 | Price:
575.789978
Date with the lowest closing price: 2023-03-10 00:00:00 | Price:
292.76001
Date with the highest volume: 2023-10-19 00:00:00 | Volume: 28074400
Date with the lowest volume: 2023-11-24 00:00:00 | Volume: 1404700
```

5. Seasonality in Stock Prices

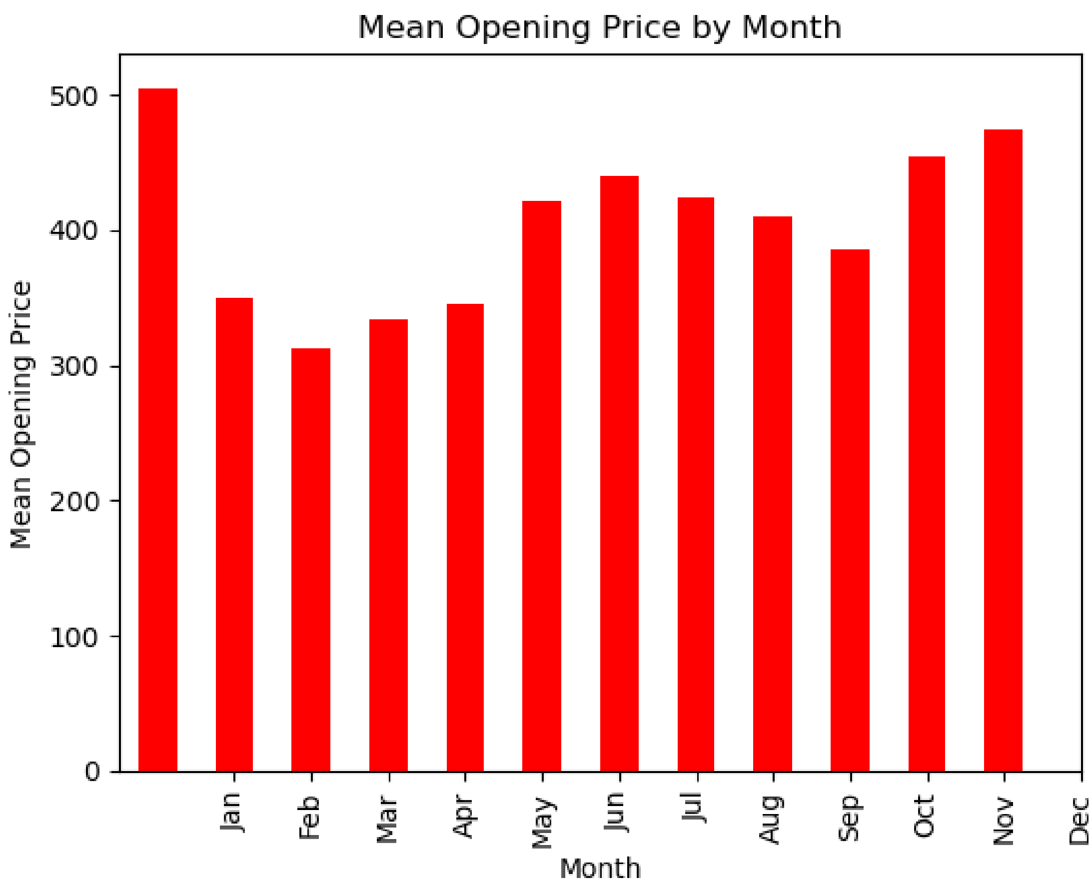
I will now be using Grouping Aggregation in time series data into periods that exhibit similar seasonal patterns. Common periods include months, quarters, or even specific days of the week.

```
# Group data by month and calculate mean closing price for each month
monthly_data = netflix.groupby(netflix['Date'].dt.month)
['Close'].mean()

# Visualize the mean closing price for each month
monthly_data.plot(kind='bar', xlabel='Month', ylabel='Mean Closing
Price', title='Mean Closing Price by Month')
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.show()
```



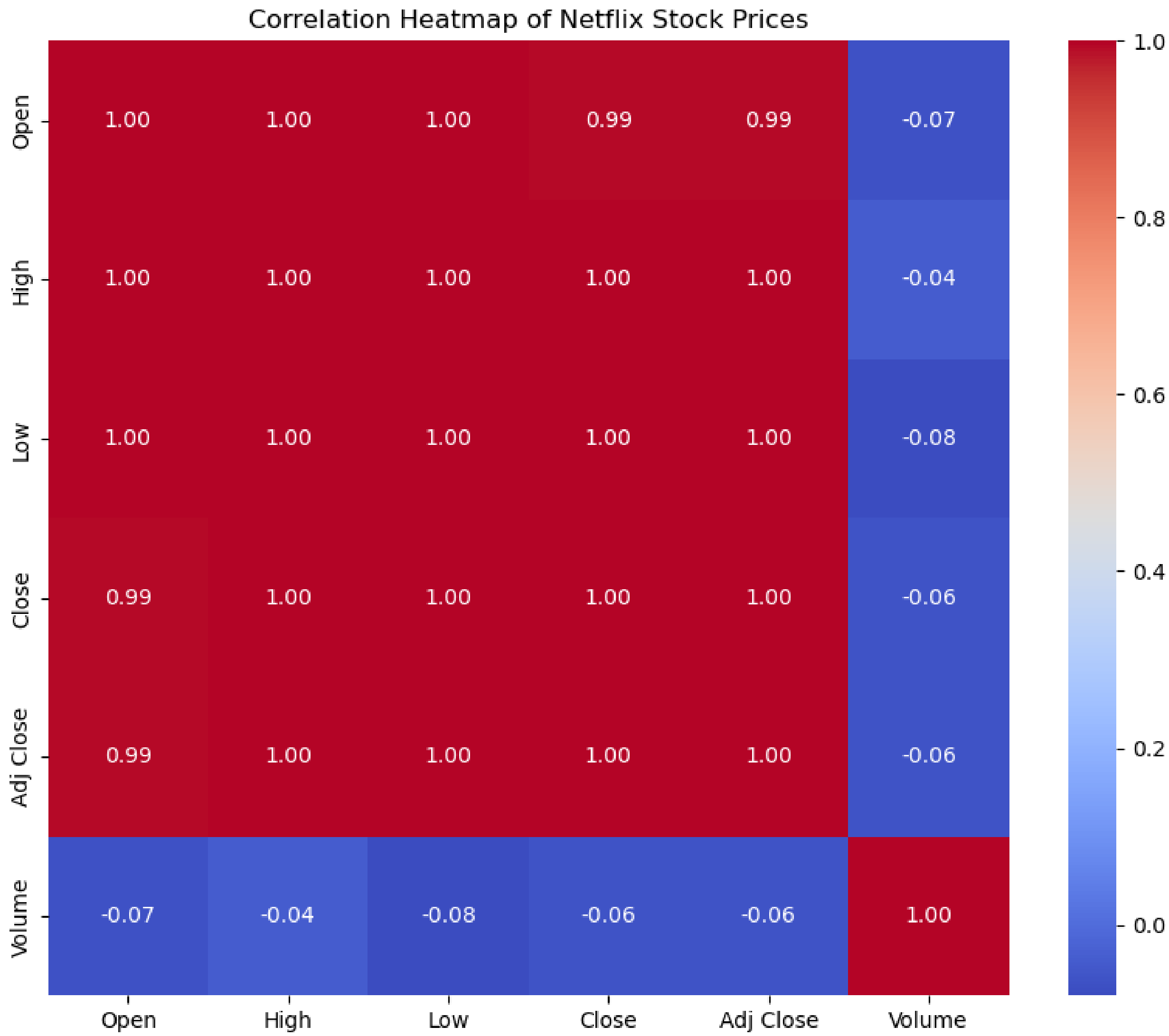

```
# Group data by month and calculate mean closing price for each month
monthly_data = netflix.groupby(netflix['Date'].dt.month)
['Open'].mean()

# Visualize the mean opening price for each month
```

```
monthly_data.plot(kind='bar', xlabel='Month', ylabel='Mean Opening
Price', title='Mean Opening Price by Month', color='r')
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.show()
```




```
import seaborn as sns

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

# Create the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Heatmap of Netflix Stock Prices')
plt.show()

C:\Users\Talaal Yousuf\AppData\Local\Temp\
ipykernel_1788\3465440380.py:4: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
```

```
of numeric_only to silence this warning.
  correlation_matrix = netflix.corr()
```




```python
# Create a pivot table
pivot_table = netflix.pivot_table(index='Date', values=['Open',
'High', 'Low', 'Close'], aggfunc='mean')

# Display the pivot table
print("Pivot Table:")
print(pivot_table)
```

```
Pivot Table:
             Close    High     Low    Open
Date
2023-02-01 361.99 365.39 349.91 353.86
2023-02-02 366.89 368.32 358.43 365.16
```

```
2023-02-03 365.90 379.43 359.00 359.08
2023-02-06 361.48 368.45 360.68 363.64
2023-02-07 362.95 364.18 354.18 358.51
...              ...    ...    ...    ...
2024-01-25 562.00 563.46 548.46 551.95
2024-01-26 570.42 579.64 558.43 561.81
2024-01-29 575.79 578.55 562.68 571.35
2024-01-30 562.85 570.88 560.82 567.32
2024-01-31 564.11 572.15 562.04 562.85

[251 rows x 4 columns]

# Histogram
plt.figure(figsize=(10, 8))
plt.hist(netflix['Open'], bins=20, color='skyblue', alpha=0.7,
label='Open')
plt.hist(netflix['High'], bins=20, color='salmon', alpha=0.7,
label='High')
plt.hist(netflix['Low'], bins=20, color='lightgreen', alpha=0.7,
label='Low')
plt.hist(netflix['Close'], bins=20, color='lightcoral', alpha=0.7,
label='Close')
plt.title('Histogram of Netflix Stock Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

6. Predictive Analysis

Model Prediction:

In the realm of financial analysis, model prediction plays a pivotal role in forecasting the future trajectory of stock prices. Leveraging advanced statistical techniques, analysts can develop models that harness historical data to make informed predictions about future price movements. These predictions serve as valuable insights for investors, guiding their decision-making processes and informing strategic investment choices.

Potential Outcome of Predictions:

The outcomes of prediction hold significant implications for investors and stakeholders in the financial market. Accurate predictions enable investors to anticipate market trends, identify profitable investment opportunities, and mitigate risks associated with market volatility. Conversely, inaccurate predictions can lead to suboptimal investment decisions, resulting in financial losses and missed opportunities. Therefore, the reliability and precision of prediction models are paramount for enhancing investment performance and maximizing returns.

Approaches: Linear Regression:

In the realm of stock price prediction, two commonly employed approaches are Linear Regression and Polynomial Regression. These regression techniques aim to establish a mathematical relationship between input variables, such as historical stock prices and trading volume, and the target variable, which is typically the future stock price. By fitting a regression model to historical data, analysts can extrapolate this relationship to predict future price movements. I will be using Linear Regression on this analysis.

Target Variable and Inout Features:

In the context of stock price prediction, the target variable is typically the future stock price that analysts seek to predict. This could be the closing price of the stock on a future trading day. The input variables, also known as features, are historical data points that serve as predictors for the target variable. These can include past stock prices (e.g., open, high, low), trading volume, and any other relevant financial indicators that may influence the future price movements of the stock. By analyzing the historical relationship between these input variables and the target variable, analysts can develop predictive models to forecast future stock prices.

6.1 Linear Regression

So, I will be using Linear Regression and since we have multiple input featurs the formula will be:

$$f_{(w,b)}(x) = w \cdot x + b = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

Here:

$f_{(w,b)}$ = Representation of prediction made by Linear Regression

w = weights assigned to each coefficient

b = Bias Term/Intercept term

x = input features where $x_1, x_2, \ldots, x_n$ denotes individual features

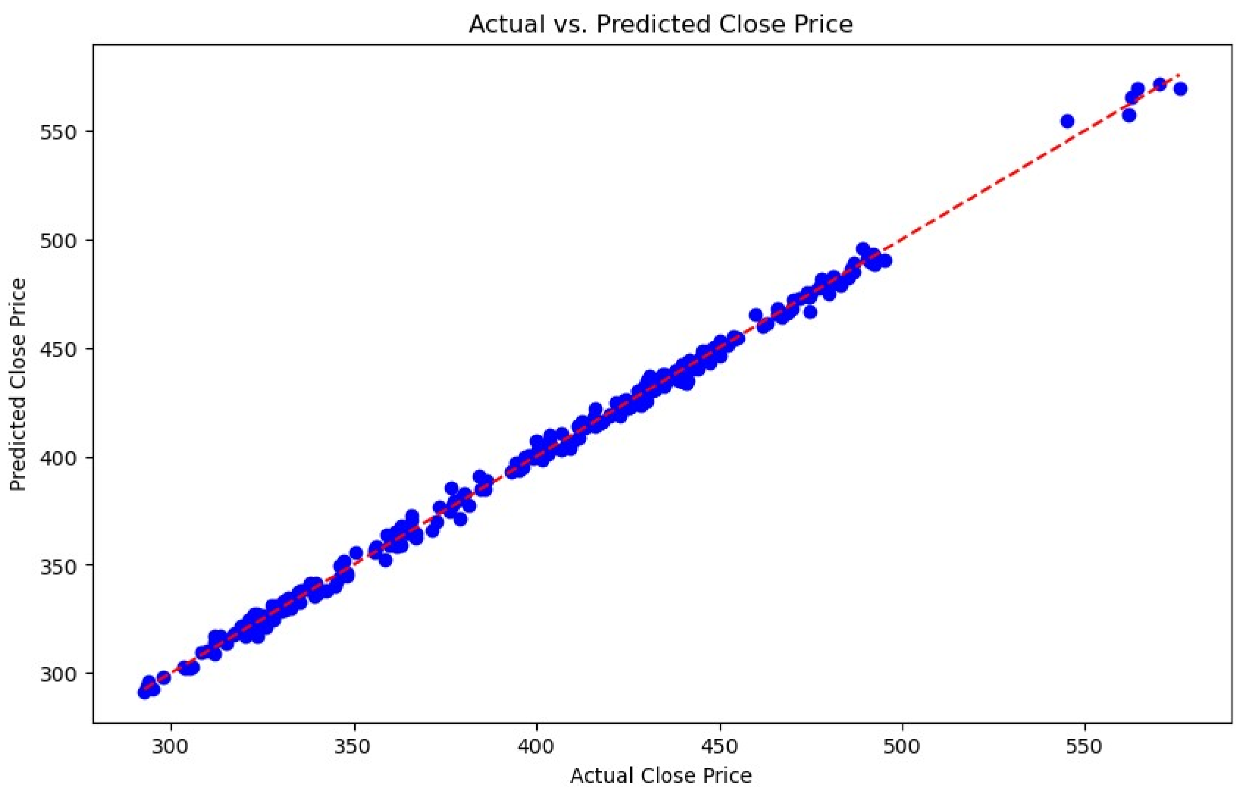
$w_1, 2_2, \ldots, w_n$ = weights assigned to each feature

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Extracting input features and target variable
X = netflix[['Open', 'High', 'Low', 'Volume']]  # Input features: 
Open, High, Low, Volume
y = netflix['Close']  # Target variable: Close

# Initialize the linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X, y)

# Print the coefficients and intercept
```

```
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [-5.15528779e-01  5.82906825e-01  9.34067273e-01
6.17473696e-08]
Intercept: 0.7558952248841138
```

```
# Make predictions on the training data
y_pred = model.predict(X)

# Plot the actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y, y_pred, color='blue')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red',
linestyle='--')
plt.xlabel('Actual Close Price')
plt.ylabel('Predicted Close Price')
plt.title('Actual vs. Predicted Close Price')
plt.show()
```




```
# Define the coefficients and intercept for the linear regression
model
coefficients = model.coef_ # Replace with your actual coefficients
intercept = model.intercept_  # Replace with your actual intercept
```

```python
# Define a function to make predictions
def predict_price(features):
    # Calculate the dot product of coefficients and features, then add the intercept
    predicted_price = np.dot(coefficients, features) + intercept
    return predicted_price

# Example input data for prediction (replace this with your actual input data)
input_features = np.array([350, 360, 345, 8000000])  # Example input features (Open, High, Low, Volume)

# Make prediction
predicted_price = predict_price(input_features)
print(f"Predicted Price is {predicted_price:.02f}")
```

```
Predicted Price is 352.91
```

Conclusion

In the realm of financial analysis, the exploration of Netflix's stock prices through a comprehensive time series analysis has unveiled a trove of valuable insights and revelations. Our journey commenced with a meticulous introduction to the dataset, providing a holistic view of its contents, source, and the analytical approach adopted. Through the lens of Exploratory Data Analysis (EDA), we embarked on a quest to unravel the mysteries encoded within the temporal fluctuations of Netflix's stock performance.

The data overview offered a panoramic snapshot of Netflix's stock prices, spanning from 2023 to 2024. We navigated through the seas of data cleaning, ensuring the integrity and reliability of our dataset for further analysis. Armed with a refined dataset, our voyage delved into the depths of time series analysis, where we charted the course of Netflix's stock prices over time.

Time series visualization served as our compass, guiding us through the turbulent waves of market dynamics. We marveled at the intricate patterns, trends, and seasonalities woven into the fabric of Netflix's stock prices. Descriptive statistics provided a compass rose, offering a compass rose, offering insights into the central tendencies and variability of the data.

The correlation matrix illuminated the interplay between different facets of Netflix's stock prices, unveiling the harmonious synchrony between opening, high, low, and closing prices. Moreover, our exploration unearthed the highest and lowest points in Netflix's stock prices, shedding light on pivotal moments in its market journey.

As we set sail towards the horizon of predictive analytics, we encountered the tantalizing prospect of forecasting future stock prices using supervised machine learning techniques. While the journey towards predictive modeling remains a voyage for another day, the groundwork laid through time series analysis has paved the way for informed decision-making and strategic planning.

In conclusion, our expedition into the seas of Netflix's stock prices has been a voyage of discovery, enlightenment, and empowerment. Armed with the insights gleaned from our analysis, stakeholders are equipped to navigate the ever-changing currents of the financial market with confidence and clarity.

As we bid adieu to this chapter of our analytical odyssey, we stand on the precipice of infinite possibilities, ready to embark on new adventures, conquer new challenges, and unearth new treasures hidden within the vast expanse of data-driven exploration.