Analysis on Amazon Stock Price and Amazon Stock News

Importing Required Libraries

In [1]:
```
!pip install pandas -q
!pip install yfinance -q
!pip install ftfy -q
!pip install spacy -q
!pip install vaderSentiment -q
!pip install pmdarima statsmodels
```

```
Requirement already satisfied: pmdarima in c:\users\97152\anaconda3\lib\site-packa
ges (2.0.4)
Requirement already satisfied: statsmodels in c:\users\97152\anaconda3\lib\site-pa
ckages (0.14.0)
Requirement already satisfied: joblib>=0.11 in c:\users\97152\anaconda3\lib\site-p
ackages (from pmdarima) (1.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in c:\users\97152
\anaconda3\lib\site-packages (from pmdarima) (3.0.11)
Requirement already satisfied: numpy>=1.21.2 in c:\users\97152\anaconda3\lib\site-
packages (from pmdarima) (1.23.5)
Requirement already satisfied: pandas>=0.19 in c:\users\97152\anaconda3\lib\site-p
ackages (from pmdarima) (2.2.0)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\97152\anaconda3\lib
\site-packages (from pmdarima) (1.5.2)
Requirement already satisfied: scipy>=1.3.2 in c:\users\97152\anaconda3\lib\site-p
ackages (from pmdarima) (1.11.1)
Requirement already satisfied: urllib3 in c:\users\97152\anaconda3\lib\site-packag
es (from pmdarima) (2.2.1)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in c:\users\97152\anaco
nda3\lib\site-packages (from pmdarima) (68.0.0)
Requirement already satisfied: packaging>=17.1 in c:\users\97152\anaconda3\lib\sit
e-packages (from pmdarima) (23.2)
Requirement already satisfied: patsy>=0.5.2 in c:\users\97152\anaconda3\lib\site-p
ackages (from statsmodels) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\97152\anaconda3
\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\97152\anaconda3\lib\site-p
ackages (from pandas>=0.19->pmdarima) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\97152\anaconda3\lib\site
-packages (from pandas>=0.19->pmdarima) (2023.3)
Requirement already satisfied: six in c:\users\97152\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels) (1.16.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\97152\anaconda3\li
b\site-packages (from scikit-learn>=0.22->pmdarima) (3.5.0)
```

In [2]:
```
!python -m spacy download en_core_web_sm -q
```

```
[+] Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

In [3]:
```
import nltk
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
```

```
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.sentiment.util import mark_negation
import spacy
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
C:\Users\97152\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWa
rning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' c
urrently installed).
  from pandas.core import (
```

Stock Price Data Loading

In [4]:
```
# File paths for the combined CSV files
combined_stock_csv_file_path = r'C:\Users\97152\OneDrive\Desktop\Project Data\combi

# Load the combined stock data into a DataFrame
stock_data = pd.read_csv(combined_stock_csv_file_path)

# Filter for the tickers 'AAPL', 'TSLA', and 'AMZN'
combined_stock_data = stock_data[stock_data['Ticker'].isin(['AAPL', 'TSLA', 'AMZN']

# Check the number of records in the filtered DataFrame
num_records_stock = combined_stock_data.shape[0]  # Number of rows in the filtered

# Display the number of records
print(f"Number of records in the combined stock data: {num_records_stock}")
```

```
Number of records in the combined stock data: 4305
```

Checking for Null, Duplicates and Outliers

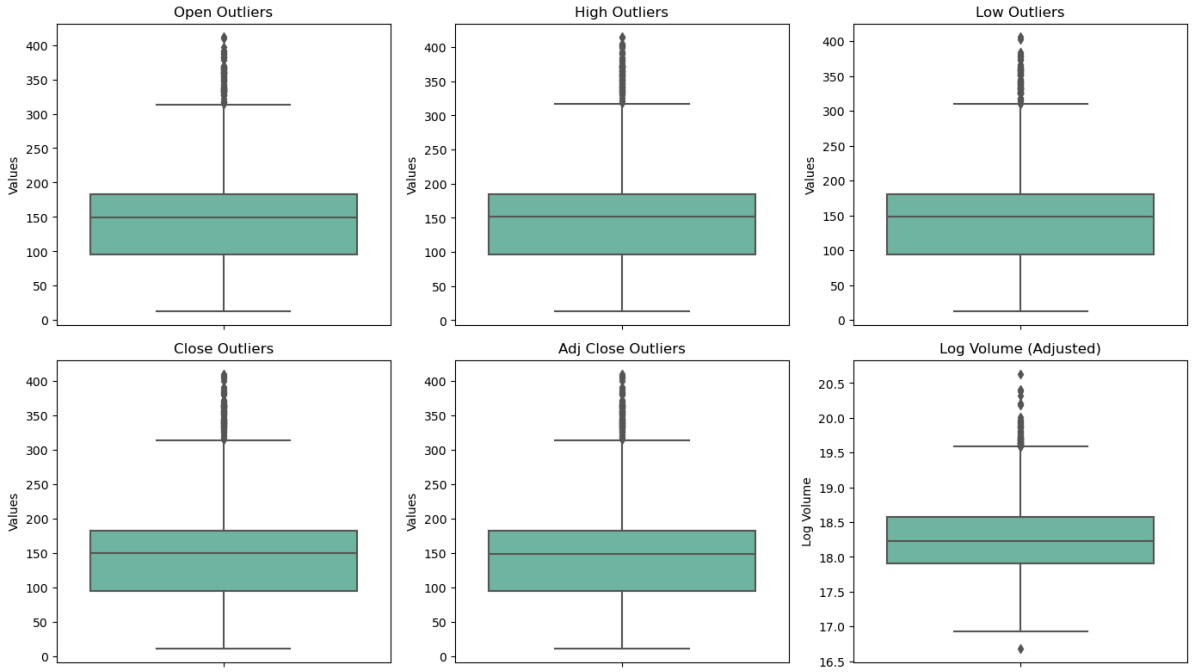
In [5]:
```
# Check for duplicates in the combined stock data
stock_duplicates = combined_stock_data.duplicated().sum()
print(f"Number of duplicate records in the combined stock data: {stock_duplicates}"

# Check for null values in the combined stock data
stock_null_values = combined_stock_data.isnull().sum()
print(f"Null values in each column of the combined stock data:\n{stock_null_values}
```

```
Number of duplicate records in the combined stock data: 0
Null values in each column of the combined stock data:
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
Ticker       0
dtype: int64
```

In [6]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Log transform the 'Volume' column for better visualization
combined_stock_data['Volume_log'] = np.log1p(combined_stock_data['Volume'])  # Logi
```

```
# Function to detect outliers using IQR and return the count
def count_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return len(outliers)

def plot_outliers_enhanced(df, columns):
    plt.figure(figsize=(14, 8))

    for i, col in enumerate(columns):
        plt.subplot(2, 3, i + 1)

        if col == 'Volume':
            sns.boxplot(data=df, y='Volume_log', palette="Set2")
            plt.title('Log Volume (Adjusted)')
            plt.ylabel('Log Volume')
            # Count and print the number of outliers for log-transformed Volume
            outliers_count = count_outliers_iqr(df, 'Volume_log')
            print(f"Number of outliers for {col}: {outliers_count}")

        else:
            sns.boxplot(data=df, y=col, palette="Set2")
            plt.title(f'{col} Outliers')
            plt.ylabel('Values')
            # Count and print the number of outliers for the column
            outliers_count = count_outliers_iqr(df, col)
            print(f"Number of outliers for {col}: {outliers_count}")

    plt.tight_layout()
    plt.show()

# Redefine the columns to include the log-transformed volume for better scaling
stock_columns = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
plot_outliers_enhanced(combined_stock_data, stock_columns)

# Drop column Volume Log
combined_stock_data.drop('Volume_log', axis=1, inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\4042142522.py:6: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  combined_stock_data['Volume_log'] = np.log1p(combined_stock_data['Volume'])  # l
og1p handles log(0) cases
Number of outliers for Open: 84
Number of outliers for High: 87
Number of outliers for Low: 80
Number of outliers for Close: 83
Number of outliers for Adj Close: 83
Number of outliers for Volume: 66
```




```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\4042142522.py:49: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  combined_stock_data.drop('Volume_log', axis=1, inplace=True)
```

In [7]:
```
combined_stock_data.columns
```

Out[7]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Ticker'],
      dtype='object')
```

Plotting the target value "Adj Close"

In [8]:
```
# Convert the 'Date' column to datetime format
combined_stock_data['Date'] = pd.to_datetime(combined_stock_data['Date'])

# Filter data for each stock
apple_data = combined_stock_data[combined_stock_data['Ticker'] == 'AAPL']
amazon_data = combined_stock_data[combined_stock_data['Ticker'] == 'AMZN']
tesla_data = combined_stock_data[combined_stock_data['Ticker'] == 'TSLA']
#zoom_data = combined_stock_data[combined_stock_data['Ticker'] == 'ZM']
#pfizer_data = combined_stock_data[combined_stock_data['Ticker'] == 'PFE']

# Plot the stock prices for 'Adj Close' over time for each stock
plt.figure(figsize=(12, 8))

# Plotting Apple
plt.plot(apple_data['Date'], apple_data['Adj Close'], label='Apple (AAPL)')

# Plotting Amazon
plt.plot(amazon_data['Date'], amazon_data['Adj Close'], label='Amazon (AMZN)')

# Plotting Tesla
plt.plot(tesla_data['Date'], tesla_data['Adj Close'], label='Tesla (TSLA)')

# Plotting Zoom
#plt.plot(zoom_data['Date'], zoom_data['Adj Close'], label='Zoom (ZM)')
```

```
# Plotting Pfizer
#plt.plot(pfizer_data['Date'], pfizer_data['Adj Close'], label='Pfizer (PFE)')

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.title('Stock Prices of Apple, Amazon, Tesla')
plt.legend()

# Show the plot
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3510742639.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  combined_stock_data['Date'] = pd.to_datetime(combined_stock_data['Date'])
```

Processing Apple Stocks

In [9]:
```
# Extracting only the Apple stock data
amazon_data = combined_stock_data[combined_stock_data['Ticker'] == 'AMZN']

# Find the latest (end) date in the dataset
end_date = combined_stock_data['Date'].max()

# Display the data for the end date
print(end_date)
```

```
2024-09-13 00:00:00
```

In [10]:
```
amazon_data.columns
```

Out[10]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
       'Ticker'],
      dtype='object')
```

In [11]:
```
# Generate a date range from 2019-01-01 to 2024-09-13
# we are generating the dates for weekend and holidays so there are no missing valu
date_range = pd.date_range(start='2019-01-01', end=end_date, freq='D')

# Set 'Date' as the index and reindex to include all dates
amazon_data = amazon_data.set_index('Date').reindex(date_range)

# Ensure the index is named 'Date'
amazon_data.index.name = 'Date'
```

In [12]:
```
# Fill missing values as needed (forward fill or NaN is default)
amazon_data.fillna(method='ffill', inplace=True)

# Then apply backward fill (bfill) to handle the first row or initial missing value
amazon_data.fillna(method='bfill', inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3371541635.py:2: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  amazon_data.fillna(method='ffill', inplace=True)
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3371541635.py:5: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  amazon_data.fillna(method='bfill', inplace=True)
```

In [13]:
```
amazon_data.isnull()
```

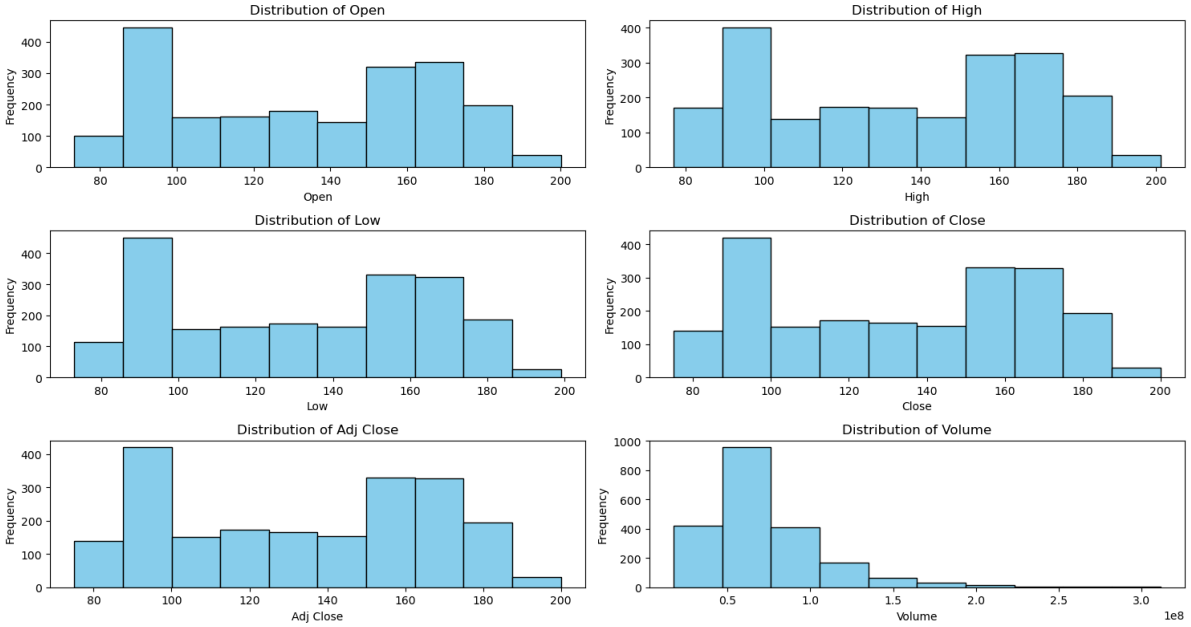
Out[13]:

Date	Open	High	Low	Close	Adj Close	Volume	Ticker
2019-01-01	False	False	False	False	False	False	False
2019-01-02	False	False	False	False	False	False	False
2019-01-03	False	False	False	False	False	False	False
2019-01-04	False	False	False	False	False	False	False
2019-01-05	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
2024-09-09	False	False	False	False	False	False	False
2024-09-10	False	False	False	False	False	False	False
2024-09-11	False	False	False	False	False	False	False
2024-09-12	False	False	False	False	False	False	False
2024-09-13	False	False	False	False	False	False	False

2083 rows × 7 columns

In [14]:
```
# Select numerical columns
numerical_columns = amazon_data.select_dtypes(include=['float64', 'int64']).columns

# Determine the number of rows and columns for the subplots
num_columns = len(numerical_columns)
num_rows = (num_columns + 1) // 2  # Adjust for odd number of columns
```

```
# Plotting histograms to check the distribution of each numerical column
plt.figure(figsize=(15, 8))

# Plot histograms for each numerical column to visualize their distribution
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(num_rows, 2, i)  # Dynamically adjust subplot grid
    plt.hist(amazon_data[col], bins=10, color='skyblue', edgecolor='black')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

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



In [15]:
```
amazon_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2083 entries, 2019-01-01 to 2024-09-13
Freq: D
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       2083 non-null   float64
 1   High       2083 non-null   float64
 2   Low        2083 non-null   float64
 3   Close      2083 non-null   float64
 4   Adj Close  2083 non-null   float64
 5   Volume     2083 non-null   float64
 6   Ticker     2083 non-null   object 
dtypes: float64(6), object(1)
memory usage: 130.2+ KB
```

Feature Engineering for stock data

In [16]:
```
amazon_data['Price_Change'] = amazon_data['Close']-amazon_data['Open']
amazon_data['Daily_Volatility'] = amazon_data['High']-amazon_data['Low']
```

In [17]:
```
# Calculate daily returns (percentage change between consecutive 'Adj Close' prices
amazon_data['Daily_Returns'] = amazon_data['Adj Close'].pct_change() * 100
```

In [18]:
```
amazon_data.isnull().sum()
null_rows = amazon_data[amazon_data.isnull().any(axis=1)]
print(null_rows)
```

```
                 Open       High        Low      Close  Adj Close  \
Date
2019-01-01  73.260002  77.667999  73.046501  76.956497  76.956497

                Volume Ticker  Price_Change  Daily_Volatility  Daily_Returns
Date
2019-01-01  159662000.0   AMZN      3.696495          4.621498            NaN
```

In [19]:
```
# Then apply backward fill (bfill) to handle the first row or initial missing value
amazon_data.fillna(method='bfill', inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\1973331760.py:2: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  amazon_data.fillna(method='bfill', inplace=True)
```

In [20]:
```
amazon_data
```

Out[20]:

Date	Open	High	Low	Close	Adj Close	Volume	Ticker	Price_Char
2019-01-01	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-02	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-03	76.000504	76.900002	74.855499	75.014000	75.014000	139512000.0	AMZN	-0.986
2019-01-04	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
2019-01-05	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
...	...	...	...	...	...	...	...	
2024-09-09	174.529999	175.850006	173.509995	175.399994	175.399994	29037400.0	AMZN	0.869
2024-09-10	177.490005	180.500000	176.789993	179.550003	179.550003	36233800.0	AMZN	2.059
2024-09-11	180.100006	184.990005	175.729996	184.520004	184.520004	42564700.0	AMZN	4.419
2024-09-12	184.800003	187.410004	183.539993	187.000000	187.000000	33622500.0	AMZN	2.199
2024-09-13	187.000000	188.500000	185.910004	186.490005	186.490005	26476500.0	AMZN	-0.509

2083 rows × 10 columns

In [21]:
```
# Calculate 5-day price momentum (difference between today's 'Adj Close' and 5 days
n = 5  # Change this value for different momentum periods
amazon_data['price_momentum'] = amazon_data['Adj Close'] - amazon_data['Adj Close']
```

In [22]:
```
amazon_data.isnull().sum()
null_rows = amazon_data[amazon_data.isnull().any(axis=1)]
print(null_rows)
```

```
                Open       High        Low      Close  Adj Close  \
Date
2019-01-01  73.260002  77.667999  73.046501  76.956497  76.956497
2019-01-02  73.260002  77.667999  73.046501  76.956497  76.956497
2019-01-03  76.000504  76.900002  74.855499  75.014000  75.014000
2019-01-04  76.500000  79.699997  75.915497  78.769501  78.769501
2019-01-05  76.500000  79.699997  75.915497  78.769501  78.769501

                Volume Ticker  Price_Change  Daily_Volatility  Daily_Returns  \
Date
2019-01-01  159662000.0   AMZN      3.696495          4.621498        0.00000
2019-01-02  159662000.0   AMZN      3.696495          4.621498        0.00000
2019-01-03  139512000.0   AMZN     -0.986504          2.044502       -2.52415
2019-01-04  183652000.0   AMZN      2.269501          3.784500        5.00640
2019-01-05  183652000.0   AMZN      2.269501          3.784500        0.00000

            price_momentum
Date
2019-01-01             NaN
2019-01-02             NaN
2019-01-03             NaN
2019-01-04             NaN
2019-01-05             NaN
```

In [23]:
```
# Then apply backward fill (bfill) to handle the first row or initial missing value
amazon_data.fillna(method='bfill', inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\1973331760.py:2: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  amazon_data.fillna(method='bfill', inplace=True)
```

In [24]:
```
amazon_data
```

Out[24]:

Date	Open	High	Low	Close	Adj Close	Volume	Ticker	Price_Char
2019-01-01	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-02	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-03	76.000504	76.900002	74.855499	75.014000	75.014000	139512000.0	AMZN	-0.986
2019-01-04	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
2019-01-05	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
...	...	...	...	...	...	...	...	
2024-09-09	174.529999	175.850006	173.509995	175.399994	175.399994	29037400.0	AMZN	0.869
2024-09-10	177.490005	180.500000	176.789993	179.550003	179.550003	36233800.0	AMZN	2.059
2024-09-11	180.100006	184.990005	175.729996	184.520004	184.520004	42564700.0	AMZN	4.419
2024-09-12	184.800003	187.410004	183.539993	187.000000	187.000000	33622500.0	AMZN	2.199
2024-09-13	187.000000	188.500000	185.910004	186.490005	186.490005	26476500.0	AMZN	-0.509

2083 rows × 11 columns

In [25]:

```
import numpy as np
### 1. Bollinger Bands Calculation (20-day window)

def calculate_bollinger_bands(amazon_data, window=20):
    amazon_data['SMA'] = amazon_data['Adj Close'].rolling(window=window).mean()
    amazon_data['Std_Dev'] = amazon_data['Adj Close'].rolling(window=window).std()
    amazon_data['Upper_Band'] = amazon_data['SMA'] + (2 * amazon_data['Std_Dev'])
    amazon_data['Lower_Band'] = amazon_data['SMA'] - (2 * amazon_data['Std_Dev'])
    return amazon_data

# Apply Bollinger Bands calculation
amazon_data = calculate_bollinger_bands(amazon_data)

print(amazon_data)
```

```
                 Open        High         Low       Close   Adj Close  \
Date
2019-01-01   73.260002   77.667999   73.046501   76.956497   76.956497
2019-01-02   73.260002   77.667999   73.046501   76.956497   76.956497
2019-01-03   76.000504   76.900002   74.855499   75.014000   75.014000
2019-01-04   76.500000   79.699997   75.915497   78.769501   78.769501
2019-01-05   76.500000   79.699997   75.915497   78.769501   78.769501
...                ...         ...         ...         ...         ...
2024-09-09  174.529999  175.850006  173.509995  175.399994  175.399994
2024-09-10  177.490005  180.500000  176.789993  179.550003  179.550003
2024-09-11  180.100006  184.990005  175.729996  184.520004  184.520004
2024-09-12  184.800003  187.410004  183.539993  187.000000  187.000000
2024-09-13  187.000000  188.500000  185.910004  186.490005  186.490005

                Volume Ticker  Price_Change  Daily_Volatility  Daily_Returns  \
Date
2019-01-01  159662000.0   AMZN      3.696495          4.621498       0.000000
2019-01-02  159662000.0   AMZN      3.696495          4.621498       0.000000
2019-01-03  139512000.0   AMZN     -0.986504          2.044502      -2.524150
2019-01-04  183652000.0   AMZN      2.269501          3.784500       5.006400
2019-01-05  183652000.0   AMZN      2.269501          3.784500       0.000000
...                 ...    ...           ...               ...            ...
2024-09-09   29037400.0   AMZN      0.869995          2.340012       2.339690
2024-09-10   36233800.0   AMZN      2.059998          3.710007       2.366026
2024-09-11   42564700.0   AMZN      4.419998          9.260010       2.768032
2024-09-12   33622500.0   AMZN      2.199997          3.870010       1.344025
2024-09-13   26476500.0   AMZN     -0.509995          2.589996      -0.272724

            price_momentum         SMA   Std_Dev  Upper_Band  Lower_Band
Date
2019-01-01        1.813004         NaN       NaN         NaN         NaN
2019-01-02        1.813004         NaN       NaN         NaN         NaN
2019-01-03        1.813004         NaN       NaN         NaN         NaN
2019-01-04        1.813004         NaN       NaN         NaN         NaN
2019-01-05        1.813004         NaN       NaN         NaN         NaN
...                    ...         ...       ...         ...         ...
2024-09-09        2.069992  175.496999  2.952360  181.401719  169.592278
2024-09-10        1.660004  175.468999  2.908640  181.286279  169.651718
2024-09-11       13.130005  175.888499  3.544512  182.977523  168.799475
2024-09-12       15.610001  176.386499  4.327921  185.042342  167.730656
2024-09-13       15.100006  176.859000  4.883247  186.625493  167.092507

[2083 rows x 15 columns]
```

In [26]:
```
# Then apply backward fill (bfill) to handle the first row or initial missing value
amazon_data.fillna(method='bfill', inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\1973331760.py:2: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  amazon_data.fillna(method='bfill', inplace=True)
```

In [27]:
```
amazon_data.isnull().sum()
```

Out[27]:
```
Open                0
High                0
Low                 0
Close               0
Adj Close           0
Volume              0
Ticker              0
Price_Change        0
Daily_Volatility    0
Daily_Returns       0
price_momentum      0
SMA                 0
Std_Dev             0
Upper_Band          0
Lower_Band          0
dtype: int64
```

In [28]:
```
### 2. Relative Strength Index (RSI) Calculation (14-day window)

def calculate_rsi(df, window=14):
    delta = df['Adj Close'].diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()

    RS = gain / loss
    df['RSI'] = 100 - (100 / (1 + RS))
    return df

# Apply RSI calculation
amazon_data = calculate_rsi(amazon_data)
amazon_data
```

Out[28]:

Date	Open	High	Low	Close	Adj Close	Volume	Ticker	Price_Char
2019-01-01	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-02	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-03	76.000504	76.900002	74.855499	75.014000	75.014000	139512000.0	AMZN	-0.986
2019-01-04	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
2019-01-05	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
...	...	...	...	...	...	...	...	
2024-09-09	174.529999	175.850006	173.509995	175.399994	175.399994	29037400.0	AMZN	0.869
2024-09-10	177.490005	180.500000	176.789993	179.550003	179.550003	36233800.0	AMZN	2.059
2024-09-11	180.100006	184.990005	175.729996	184.520004	184.520004	42564700.0	AMZN	4.419
2024-09-12	184.800003	187.410004	183.539993	187.000000	187.000000	33622500.0	AMZN	2.199
2024-09-13	187.000000	188.500000	185.910004	186.490005	186.490005	26476500.0	AMZN	-0.509

2083 rows × 16 columns

In [29]:
```
# Then apply backward fill (bfill) to handle the first row or initial missing value
amazon_data.fillna(method='bfill', inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\1973331760.py:2: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  amazon_data.fillna(method='bfill', inplace=True)
```

In [30]:
```
### 3. Moving Average (MA) Calculation

def calculate_moving_average(amazon_data, window):
    amazon_data[f'MA_{window}'] = amazon_data['Adj Close'].rolling(window=window).m
    return amazon_data

# Apply Moving Averages (e.g., 50-day and 200-day MAs)
amazon_data = calculate_moving_average(amazon_data, 50)  # Short-term trend
amazon_data = calculate_moving_average(amazon_data, 200)  # Long-term trend
```

In [31]:
```
amazon_data
```

Out[31]:

Date	Open	High	Low	Close	Adj Close	Volume	Ticker	Price_Char
2019-01-01	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-02	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.696
2019-01-03	76.000504	76.900002	74.855499	75.014000	75.014000	139512000.0	AMZN	-0.986
2019-01-04	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
2019-01-05	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.269
...	...	...	...	...	...	...	...	
2024-09-09	174.529999	175.850006	173.509995	175.399994	175.399994	29037400.0	AMZN	0.869
2024-09-10	177.490005	180.500000	176.789993	179.550003	179.550003	36233800.0	AMZN	2.059
2024-09-11	180.100006	184.990005	175.729996	184.520004	184.520004	42564700.0	AMZN	4.419
2024-09-12	184.800003	187.410004	183.539993	187.000000	187.000000	33622500.0	AMZN	2.199
2024-09-13	187.000000	188.500000	185.910004	186.490005	186.490005	26476500.0	AMZN	-0.509

2083 rows × 18 columns

In [32]:
```
# Fill missing values in 'MA_200' with the mean
amazon_data['MA_200'] = amazon_data['MA_200'].fillna(amazon_data['MA_200'].mean())

# Fill missing values in 'MA_50' with the mean
amazon_data['MA_50'] = amazon_data['MA_50'].fillna(amazon_data['MA_50'].mean())
```

In [33]:
```
# Calculate 12-period and 26-period EMAs for the MACD calculation
amazon_data['EMA12'] = amazon_data['Close'].ewm(span=12, adjust=False).mean()
amazon_data['EMA26'] = amazon_data['Close'].ewm(span=26, adjust=False).mean()
```

In [34]:
```
# Calculate MACD
amazon_data['MACD'] = amazon_data['EMA12'] - amazon_data['EMA26']
```

In [35]:
```
# Calculate Signal Line (9-day EMA of the MACD)
amazon_data['Signal Line'] = amazon_data['MACD'].ewm(span=9, adjust=False).mean()
```

In [36]:
```
amazon_data.isnull().sum()
null_rows = amazon_data[amazon_data.isnull().any(axis=1)]
print(null_rows)
```

```
Empty DataFrame
Columns: [Open, High, Low, Close, Adj Close, Volume, Ticker, Price_Change, Daily_V
olatility, Daily_Returns, price_momentum, SMA, Std_Dev, Upper_Band, Lower_Band, RS
I, MA_50, MA_200, EMA12, EMA26, MACD, Signal Line]
Index: []

[0 rows x 22 columns]
```

In [37]:
```
# Calculate 10-day Volume Moving Average (VMA)
amazon_data['VMA10'] = amazon_data['Volume'].rolling(window=10).mean()
```

In [38]:
```
# Fill missing values in 'VMA10' with the mean
amazon_data['VMA10'] = amazon_data['VMA10'].fillna(amazon_data['VMA10'].mean())
```

In [39]:
```
# Create lagged versions of Adj Close (previous day and previous week)
amazon_data['Adj Close_Lag1'] = amazon_data['Adj Close'].shift(1)  # Lagged by 1 da
amazon_data['Adj Close_Lag7'] = amazon_data['Adj Close'].shift(7)  # Lagged by 1 we
```

In [40]:
```
# Fill missing values in 'Adj Close_Lag1' and 'Adj Close_Lag7' with the mean
amazon_data['Adj Close_Lag1'] = amazon_data['Adj Close_Lag1'].fillna(amazon_data['A
# Fill missing values in 'MA_200' with the mean
amazon_data['Adj Close_Lag7'] = amazon_data['Adj Close_Lag7'].fillna(amazon_data['A
```

In [41]:
```
amazon_data.isnull().sum()
null_rows = amazon_data[amazon_data.isnull().any(axis=1)]
print(null_rows)
```

```
Empty DataFrame
Columns: [Open, High, Low, Close, Adj Close, Volume, Ticker, Price_Change, Daily_V
olatility, Daily_Returns, price_momentum, SMA, Std_Dev, Upper_Band, Lower_Band, RS
I, MA_50, MA_200, EMA12, EMA26, MACD, Signal Line, VMA10, Adj Close_Lag1, Adj Clos
e_Lag7]
Index: []

[0 rows x 25 columns]
```

In [42]:
```
amazon_data.columns
```

Out[42]:
```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Ticker',
       'Price_Change', 'Daily_Volatility', 'Daily_Returns', 'price_momentum',
       'SMA', 'Std_Dev', 'Upper_Band', 'Lower_Band', 'RSI', 'MA_50', 'MA_200',
       'EMA12', 'EMA26', 'MACD', 'Signal Line', 'VMA10', 'Adj Close_Lag1',
       'Adj Close_Lag7'],
      dtype='object')
```

Feature Engineering for news data

In [43]:
```
# Download the required NLTK data
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\97152\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\97152\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\97152\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\97152\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\97152\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[43]:
```
True
```

News Data Loading

In [44]:
```
# File paths for the combined CSV files
combined_news_csv_file_path = r'C:\Users\97152\OneDrive\Desktop\Project Data\combir
# Load the combined news data into a DataFrame
news_data = pd.read_csv(combined_news_csv_file_path)

# Filter for the tickers 'AAPL', 'TSLA', and 'AMZN'
combined_news_data = news_data[news_data['Ticker'].isin(['AAPL', 'TSLA', 'AMZN'])]

# Check the number of records in the filtered news DataFrame
num_records_news = combined_news_data.shape[0]  # Number of rows in the filtered ne

# Display the number of records
print(f"Number of records in the combined news data: {num_records_news}")
```

```
Number of records in the combined news data: 51356
```

Checking for Null values & Duplicates

In [45]:
```
# Check for duplicates in the combined stock data
stock_duplicates = combined_news_data.duplicated().sum()
print(f"Number of duplicate records in the combined stock data: {stock_duplicates}"

# Check for null values in the combined stock data
stock_null_values = combined_news_data.isnull().sum()
print(f"Null values in each column of the combined stock data:\n{stock_null_values}
```

```
Number of duplicate records in the combined stock data: 0
Null values in each column of the combined stock data:
Date        0
Source      0
Headline    0
Ticker      0
dtype: int64
```

Processing Apple News Data

In [46]:
```
# Convert the 'Date' column to datetime format
combined_news_data['Date'] = pd.to_datetime(combined_news_data['Date'])

# Extracting only the Apple stock data
amazon_news = combined_news_data[combined_news_data['Ticker'] == 'AMZN']
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\4266335436.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  combined_news_data['Date'] = pd.to_datetime(combined_news_data['Date'])
```

Cleaning Text Data

In [47]:
```
# Initialize lemmatizer and stop words
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Load spacy model for Named Entity Recognition (NER)
nlp = spacy.load('en_core_web_sm')

# Custom stop words for domain-specific data (Optional)
custom_stop_words = {'stock', 'market', 'apple','amazon','tesla','zoom','pfizer'}

# Encoding replacements for fixing common encoding issues
encoding_replacements = {
    'â€™': "'", 'â€œ': '"', 'â€˜': '‘', 'â€“': '-', 'â€”': '—', 'â€¦': '…', 'â€': '
    # Add more replacements as needed
}

def preserve_named_entities(text):
    # Use spaCy to preserve named entities like "NVDA", "Nvidia", etc.
    doc = nlp(text)
    preserved_tokens = []
    for token in doc:
        if token.ent_type_:  # If token is recognized as a named entity
            preserved_tokens.append(token.text)  # Keep entity as-is
        else:
            preserved_tokens.append(token.lemma_)  # Lemmatize non-entity words
    return " ".join(preserved_tokens)

def clean_text(text):
    # 1. Apply manual replacements for encoding issues
    for incorrect, correct in encoding_replacements.items():
        text = text.replace(incorrect, correct)

            # Step 1: Remove 'update 1' or other news update labels
    text = re.sub(r'update\s*\d+', '', text, flags=re.IGNORECASE)

    # 2. Preserve important abbreviations like "No." as "Number"
    text = re.sub(r'\bNo\.\s*(\d+)', r'Number \1', text)

    # 3. Remove special characters but keep numbers, currency symbols, and percenta
    text = re.sub(r'[^\w\s$,.%]', '', text)  # Keeping numbers, currency symbols, d

    # 4. Convert text to lowercase
    text = text.lower()
```

```
    # 5. Use NER to preserve named entities and lemmatize non-entities
    text = preserve_named_entities(text)

    # 6. Tokenize the text
    tokens = word_tokenize(text)

    # 7. Handle negations
    tokens = mark_negation(tokens)

    # 8. POS tagging
    tagged_tokens = pos_tag(tokens)

    # 9. Define POS tags to keep (e.g., nouns, verbs, adjectives, adverbs, and numb
    pos_to_keep = {'NN', 'VB', 'VBG', 'VBD', 'VBN', 'VBP', 'VBZ', 'JJ', 'RB', 'CD'}

    # 10. Lemmatize tokens and remove stop words, but keep important words like num
    lemmatized_tokens = [
        lemmatizer.lemmatize(word, pos='v') if tag.startswith('VB') else word
        for word, tag in tagged_tokens
        if word not in stop_words and word not in custom_stop_words and tag in pos_
    ]

    # 11. Join tokens back into a single string
    cleaned_text = ' '.join(lemmatized_tokens)

    return cleaned_text
```

In [48]:
```
# Clean headlines
amazon_news['Cleaned_Headline'] = amazon_news['Headline'].apply(clean_text)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\2390213456.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  amazon_news['Cleaned_Headline'] = amazon_news['Headline'].apply(clean_text)
```

In [49]:
```
# Cleaned headlines
amazon_news
```

Out[49]:

	Date	Source	Headline	Ticker	Cleaned_Headline
0	2019-01-02 11:57:05	Business Insider	Jeff Bezos said goodbye to 2018 by riding a ho...	AMZN	jeff say goodbye 2018 ride horse cowboyapparel...
38	2019-01-02 23:45:03	Business Insider	Tim Cook blames Trump's trade war with China a...	AMZN	tim cook blame trumps trade war china big fact...
41	2019-01-02 23:57:06	Business Insider	Apple is partially blaming weak iPhone sales o...	AMZN	partially blame weak iphone sale customer take...
44	2019-01-03 00:15:03	Business Insider	From Amazon to Intel, tech stocks are getting ...	AMZN	intel tech get whack surprise revenue shortfal...
90	2019-01-03 13:27:10	Business Insider	'Apple's darkest day in the iPhone era': Here'...	AMZN	day iphone era wall street say bombshell sale ...
...	...	...	...	...	...
59241	2024-09-12 10:55:20	TipRanks	Buy Rating for Amazon: A Services-Driven Valua...	AMZN	buy rating servicesdriven valuation outlook ec...
59242	2024-09-12 12:05:57	TipRanks	Amazon and Meta: Top Analyst Chooses the Best ...	AMZN	meta top analyst choose good internet buy
59244	2024-09-13 08:05:54	TipRanks	Amazon's Strong Performance and Growth Prospec...	AMZN	strong performance growth prospect underpin bu...
59249	2024-09-13 13:28:42	Seeking Alpha	U.S. finalizes tariff hikes on Chinese goods, ...	AMZN	u.s finalize tariff hike chinese good many sta...
59259	2024-09-14 19:55:06	Seeking Alpha	Satellite-TV providers DirecTV, Dish in talks ...	AMZN	satellitetv provider directv dish talk merge b...

16571 rows × 5 columns

Sentiment Analysis

In [50]:

```
# Download the VADER lexicon
nltk.download('vader_lexicon')

# Initialize the VADER sentiment analyzer
sid = SentimentIntensityAnalyzer()

# Apply VADER sentiment analysis to the 'Cleaned_Headline' column
amazon_news['Sentiment_Scores'] = amazon_news['Cleaned_Headline'].apply(lambda x: s

# Create separate columns for positive, negative, neutral, and compound scores
amazon_news['Positive'] = amazon_news['Sentiment_Scores'].apply(lambda score_dict:
amazon_news['Negative'] = amazon_news['Sentiment_Scores'].apply(lambda score_dict:
amazon_news['Neutral'] = amazon_news['Sentiment_Scores'].apply(lambda score_dict: s
amazon_news['Compound'] = amazon_news['Sentiment_Scores'].apply(lambda score_dict:

# Drop the 'Sentiment_Scores' column as it's not needed anymore
amazon_news = amazon_news.drop(columns=['Sentiment_Scores'])

# Display the first few rows of the DataFrame with new sentiment features
print(amazon_news.head())
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\97152\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

	Date	Source	Headline	Ticker	Cleaned_Headline	Positive	Negative	Neutral	Compound
0	2019-01-02 11:57:05	Business Insider	Jeff Bezos said goodbye to 2018 by riding a ho...	AMZN	jeff say goodbye 2018 ride horse cowboyapparel...	0.000	0.000	1.000	0.0000
38	2019-01-02 23:45:03	Business Insider	Tim Cook blames Trump's trade war with China a...	AMZN	tim cook blame trumps trade war china big fact...	0.000	0.441	0.559	-0.7430
41	2019-01-02 23:57:06	Business Insider	Apple is partially blaming weak iPhone sales o...	AMZN	partially blame weak iphone sale customer take...	0.123	0.325	0.552	-0.5106
44	2019-01-03 00:15:03	Business Insider	From Amazon to Intel, tech stocks are getting ...	AMZN	intel tech get whack surprise revenue shortfal...	0.139	0.000	0.861	0.2732
90	2019-01-03 13:27:10	Business Insider	'Apple's darkest day in the iPhone era': Here'...	AMZN	day iphone era wall street say bombshell sale ...	0.000	0.135	0.865	-0.1027

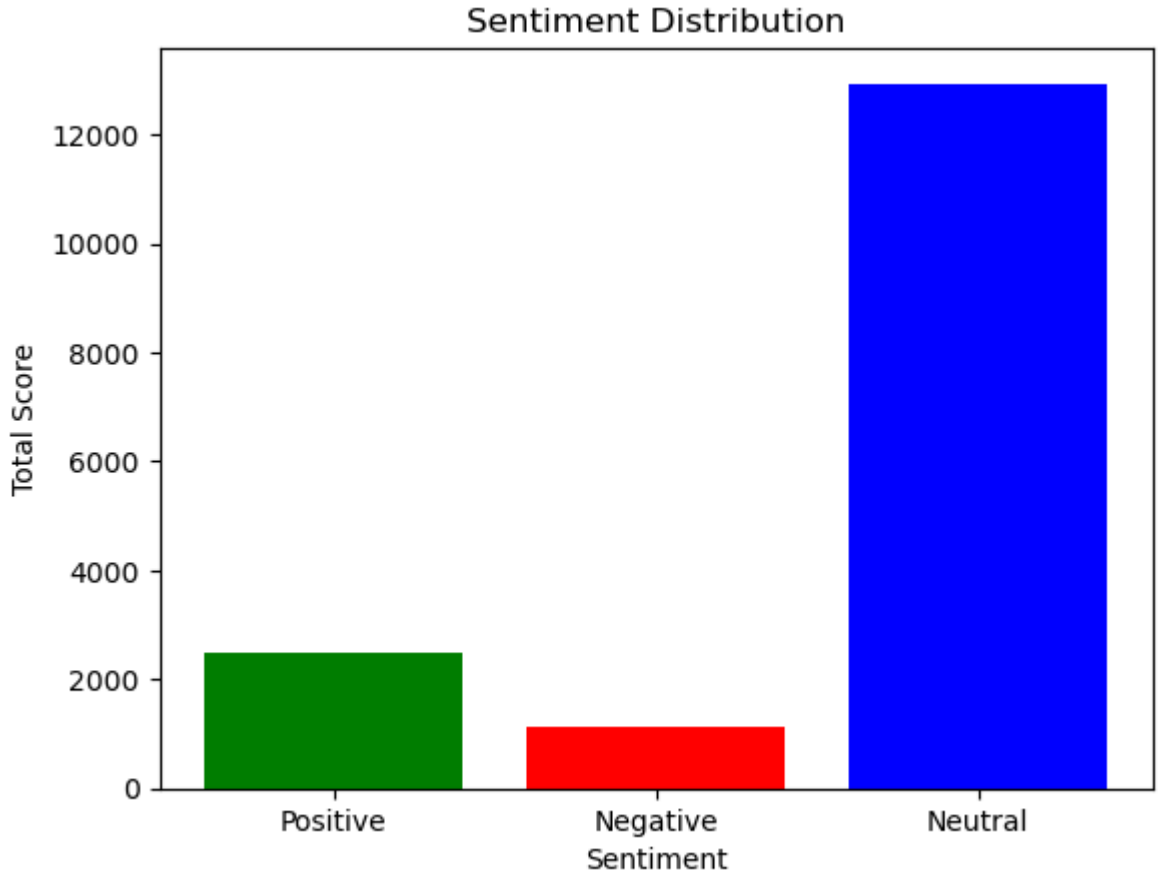
```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3441639968.py:8: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  amazon_news['Sentiment_Scores'] = amazon_news['Cleaned_Headline'].apply(lambda
x: sid.polarity_scores(x))
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3441639968.py:11: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  amazon_news['Positive'] = amazon_news['Sentiment_Scores'].apply(lambda score_dic
t: score_dict['pos'])
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3441639968.py:12: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  amazon_news['Negative'] = amazon_news['Sentiment_Scores'].apply(lambda score_dic
t: score_dict['neg'])
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3441639968.py:13: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  amazon_news['Neutral'] = amazon_news['Sentiment_Scores'].apply(lambda score_dic
t: score_dict['neu'])
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3441639968.py:14: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  amazon_news['Compound'] = amazon_news['Sentiment_Scores'].apply(lambda score_dic
t: score_dict['compound'])
```

Distribution of Sentiments

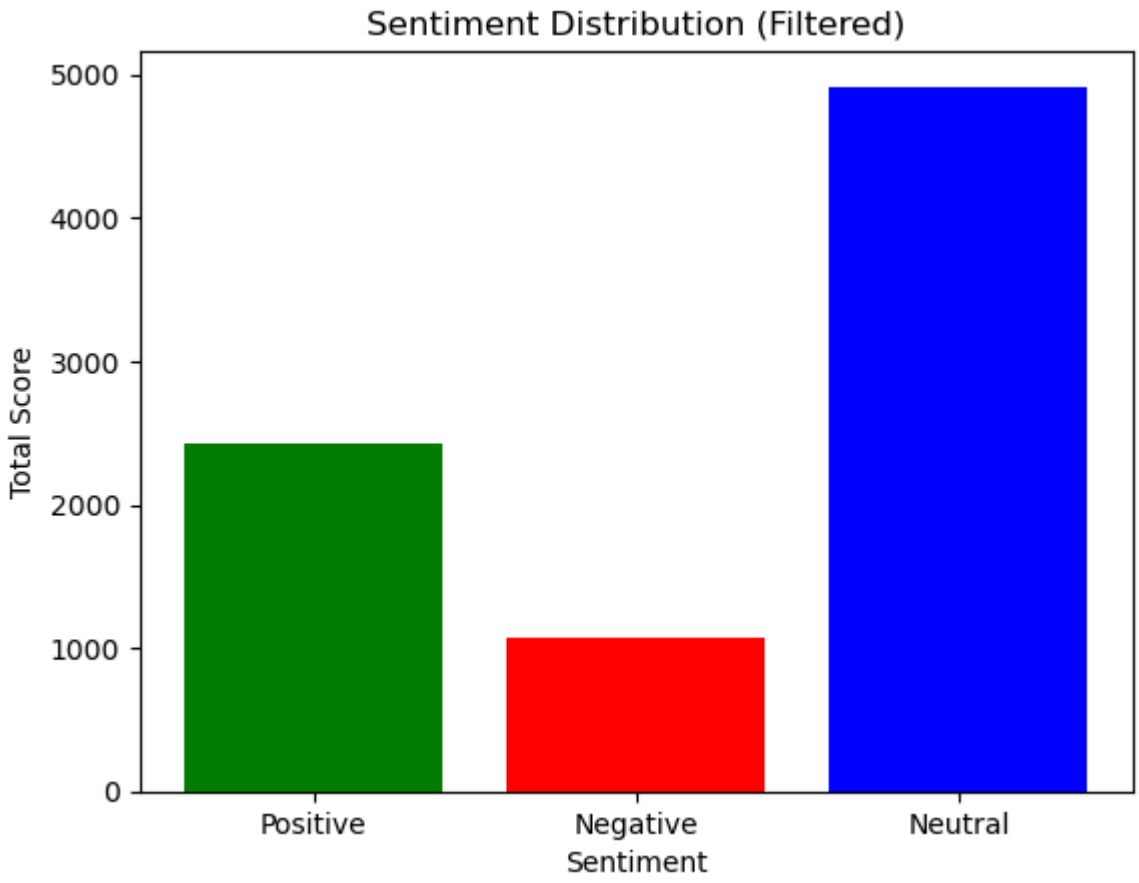
In [51]:

```
# Sum up the sentiment scores
total_positive = amazon_news['Positive'].sum()
total_negative = amazon_news['Negative'].sum()
total_neutral = amazon_news['Neutral'].sum()

# Create a bar chart
sentiment_labels = ['Positive', 'Negative', 'Neutral']
sentiment_values = [total_positive, total_negative, total_neutral]

plt.bar(sentiment_labels, sentiment_values, color=['green', 'red', 'blue'])

# Add title and labels
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
```

```
plt.ylabel('Total Score')

# Show the chart
plt.show()
```



In [52]:
```
# Filter out rows where the neutral score is higher than a threshold (e.g., 0.8)
filtered_amazon_news = amazon_news[amazon_news['Neutral'] <= 0.8]

# Sum up sentiment scores after filtering
total_positive = filtered_amazon_news['Positive'].sum()
total_negative = filtered_amazon_news['Negative'].sum()
total_neutral = filtered_amazon_news['Neutral'].sum()

# Create a bar chart with the filtered data
sentiment_values = [total_positive, total_negative, total_neutral]
plt.bar(sentiment_labels, sentiment_values, color=['green', 'red', 'blue'])
plt.title('Sentiment Distribution (Filtered)')
plt.xlabel('Sentiment')
plt.ylabel('Total Score')
plt.show()
```



Finding the mean of sentiments for each day

In [53]:
```
# Convert the 'Date' column to datetime format
filtered_amazon_news['Date'] = pd.to_datetime(filtered_amazon_news['Date'])

# Strip the time component, keeping only the date
filtered_amazon_news['Date'] = filtered_amazon_news['Date'].dt.date

# Group by the 'Date' and 'Ticker', and calculate the mean for Compound, Positive,
aggregated_amazon_news = filtered_amazon_news.groupby(['Date', 'Ticker'], as_index=
    'Compound': 'mean',
    'Positive': 'mean',
    'Negative': 'mean',
    'Neutral': 'mean'
})

# Display the first few rows of the aggregated data
print(aggregated_amazon_news)
```

```
            Date Ticker  Compound  Positive  Negative   Neutral
0     2019-01-02   AMZN -0.626800  0.061500  0.383000  0.555500
1     2019-01-03   AMZN -0.396383  0.050500  0.338667  0.610833
2     2019-01-04   AMZN  0.074400  0.284667  0.208333  0.507000
3     2019-01-07   AMZN  0.430350  0.338500  0.000000  0.661500
4     2019-01-08   AMZN  0.570700  0.356833  0.000000  0.643167
...          ...    ...       ...       ...       ...       ...
1751  2024-09-09   AMZN  0.296000  0.221000  0.123000  0.656000
1752  2024-09-10   AMZN  0.414567  0.292333  0.000000  0.707667
1753  2024-09-11   AMZN  0.366200  0.299000  0.000000  0.701000
1754  2024-09-12   AMZN  0.571900  0.485000  0.000000  0.515000
1755  2024-09-13   AMZN  0.618400  0.456500  0.000000  0.543500

[1756 rows x 6 columns]
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3675902466.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_amazon_news['Date'] = pd.to_datetime(filtered_amazon_news['Date'])
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3675902466.py:5: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_amazon_news['Date'] = filtered_amazon_news['Date'].dt.date
```

Merging stock price and stock news data

In [54]:

```
# Ensure that both dataframes have the 'Date' column in datetime format
aggregated_amazon_news['Date'] = pd.to_datetime(aggregated_amazon_news['Date'])
apple_data = apple_data.reset_index()  # Reset index so that Date becomes a column

# Merge the dataframes on the 'Date' column, keeping all rows from apple_data
merged_amazon_data = pd.merge(amazon_data, aggregated_amazon_news, on='Date', how='

# Set the index back to 'Date' for the merged dataframe
merged_amazon_data.set_index('Date', inplace=True)

merged_amazon_data
```

Out[54]:

Date	Open	High	Low	Close	Adj Close	Volume	Ticker_x	Price_Ch
2019-01-01	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.69
2019-01-02	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	AMZN	3.69
2019-01-03	76.000504	76.900002	74.855499	75.014000	75.014000	139512000.0	AMZN	-0.98
2019-01-04	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.26
2019-01-05	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	AMZN	2.26
...	...	...	...	...	...	...	...	
2024-09-09	174.529999	175.850006	173.509995	175.399994	175.399994	29037400.0	AMZN	0.86
2024-09-10	177.490005	180.500000	176.789993	179.550003	179.550003	36233800.0	AMZN	2.05
2024-09-11	180.100006	184.990005	175.729996	184.520004	184.520004	42564700.0	AMZN	4.41
2024-09-12	184.800003	187.410004	183.539993	187.000000	187.000000	33622500.0	AMZN	2.19
2024-09-13	187.000000	188.500000	185.910004	186.490005	186.490005	26476500.0	AMZN	-0.50

2083 rows × 30 columns

In [55]:
```
merged_amazon_data.drop(['Ticker_x','Ticker_y'], axis=1, inplace=True)
```

In [56]:
```
merged_amazon_data.interpolate(method='linear', inplace=True)
# First, apply backward fill to handle missing values in the first row
merged_amazon_data.fillna(method='bfill', inplace=True)
```

```
C:\Users\97152\AppData\Local\Temp\ipykernel_36824\3614850665.py:3: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
se obj.ffill() or obj.bfill() instead.
  merged_amazon_data.fillna(method='bfill', inplace=True)
```

In [57]:
```
merged_amazon_data
```

Out[57]:

Date	Open	High	Low	Close	Adj Close	Volume	Price_Change	Da
2019-01-01	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	3.696495	
2019-01-02	73.260002	77.667999	73.046501	76.956497	76.956497	159662000.0	3.696495	
2019-01-03	76.000504	76.900002	74.855499	75.014000	75.014000	139512000.0	-0.986504	
2019-01-04	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	2.269501	
2019-01-05	76.500000	79.699997	75.915497	78.769501	78.769501	183652000.0	2.269501	
...	...	...	...	...	...	...	...	
2024-09-09	174.529999	175.850006	173.509995	175.399994	175.399994	29037400.0	0.869995	
2024-09-10	177.490005	180.500000	176.789993	179.550003	179.550003	36233800.0	2.059998	
2024-09-11	180.100006	184.990005	175.729996	184.520004	184.520004	42564700.0	4.419998	
2024-09-12	184.800003	187.410004	183.539993	187.000000	187.000000	33622500.0	2.199997	
2024-09-13	187.000000	188.500000	185.910004	186.490005	186.490005	26476500.0	-0.509995	

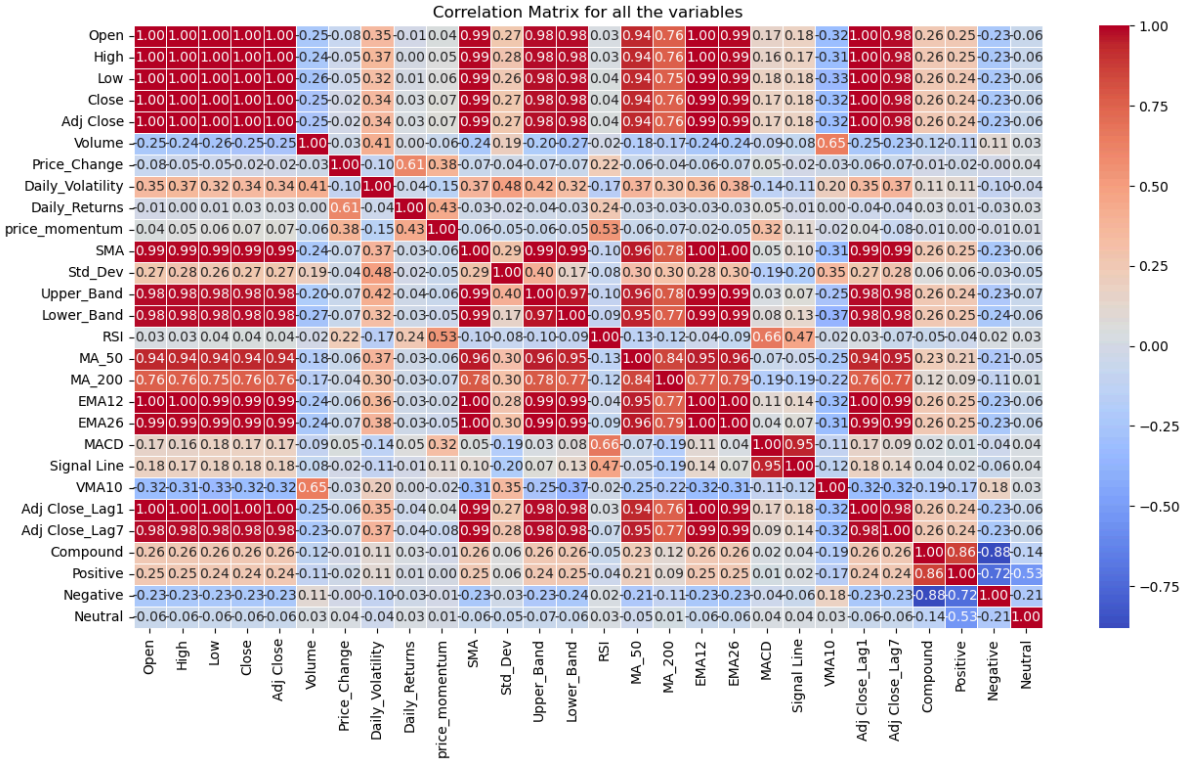
2083 rows × 28 columns

In [58]:
```
merged_amazon_data.isnull().sum()
```

Out[58]:
```
Open                0
High                0
Low                 0
Close               0
Adj Close           0
Volume              0
Price_Change        0
Daily_Volatility    0
Daily_Returns       0
price_momentum      0
SMA                 0
Std_Dev             0
Upper_Band          0
Lower_Band          0
RSI                 0
MA_50               0
MA_200              0
EMA12               0
EMA26               0
MACD                0
Signal Line         0
VMA10               0
Adj Close_Lag1      0
Adj Close_Lag7      0
Compound            0
Positive            0
Negative            0
Neutral             0
dtype: int64
```

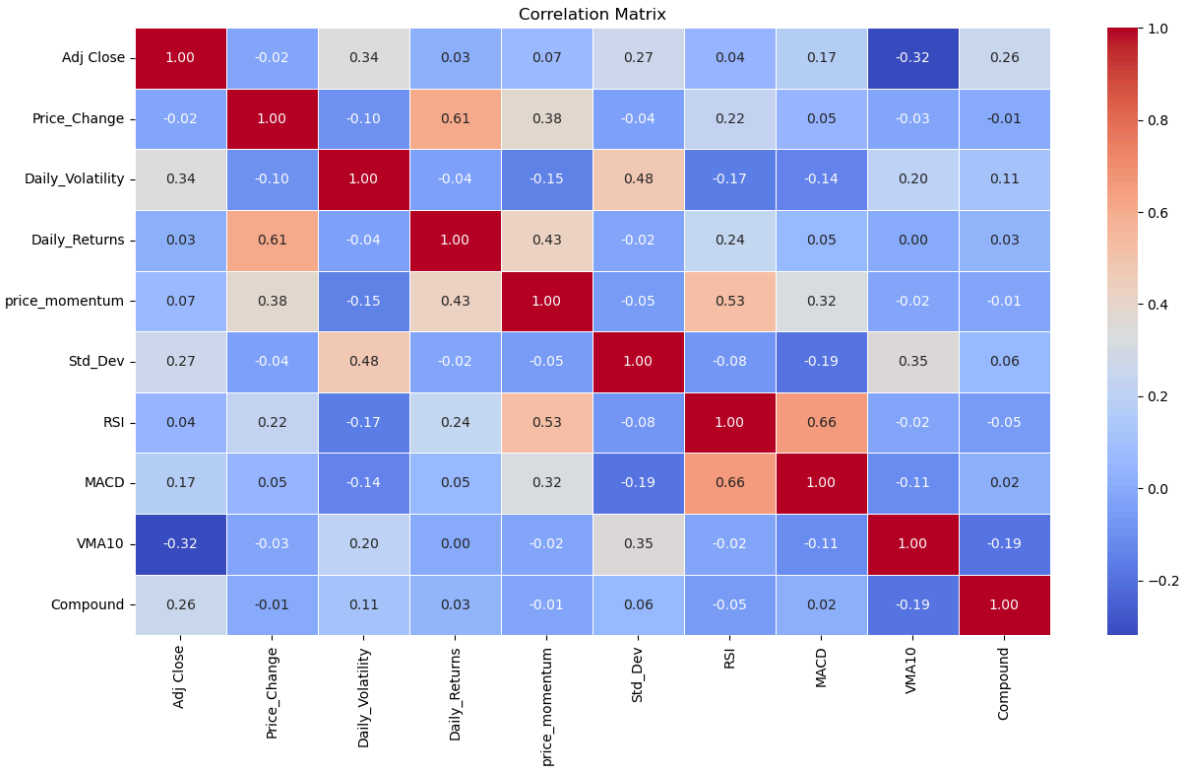
Correlation Matrix

In [59]:
```
# Compute the correlation matrix for the remaining columns
correlation_matrix = merged_amazon_data.corr()

# Display the correlation matrix
#print(correlation_matrix)

# Optionally, visualize the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(15, 8))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=
plt.title('Correlation Matrix for all the variables')
plt.show()
```



In [60]:
```
# Exclude columns 'Open', 'High', 'Low', 'Close' from the correlation matrix
#Upper_Band,Lower_Band has high correlation with SMA so we are removing it
#MA_200, EMA_12, EMA_26
columns_to_exclude = ['Open', 'High', 'Low','Close','Positive','Negative','Adj Clos
# Create a new DataFrame excluding these columns
Amazon_df = merged_amazon_data.drop(columns=columns_to_exclude)

# Compute the correlation matrix for the remaining columns
correlation_matrix = Amazon_df.corr()
```

In [61]:
```
# Optionally, visualize the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(15, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=
plt.title('Correlation Matrix')
plt.show()
```



In [62]:
```
Amazon_df.columns
```

Out[62]:
```
Index(['Adj Close', 'Price_Change', 'Daily_Volatility', 'Daily_Returns',
       'price_momentum', 'Std_Dev', 'RSI', 'MACD', 'VMA10', 'Compound'],
      dtype='object')
```

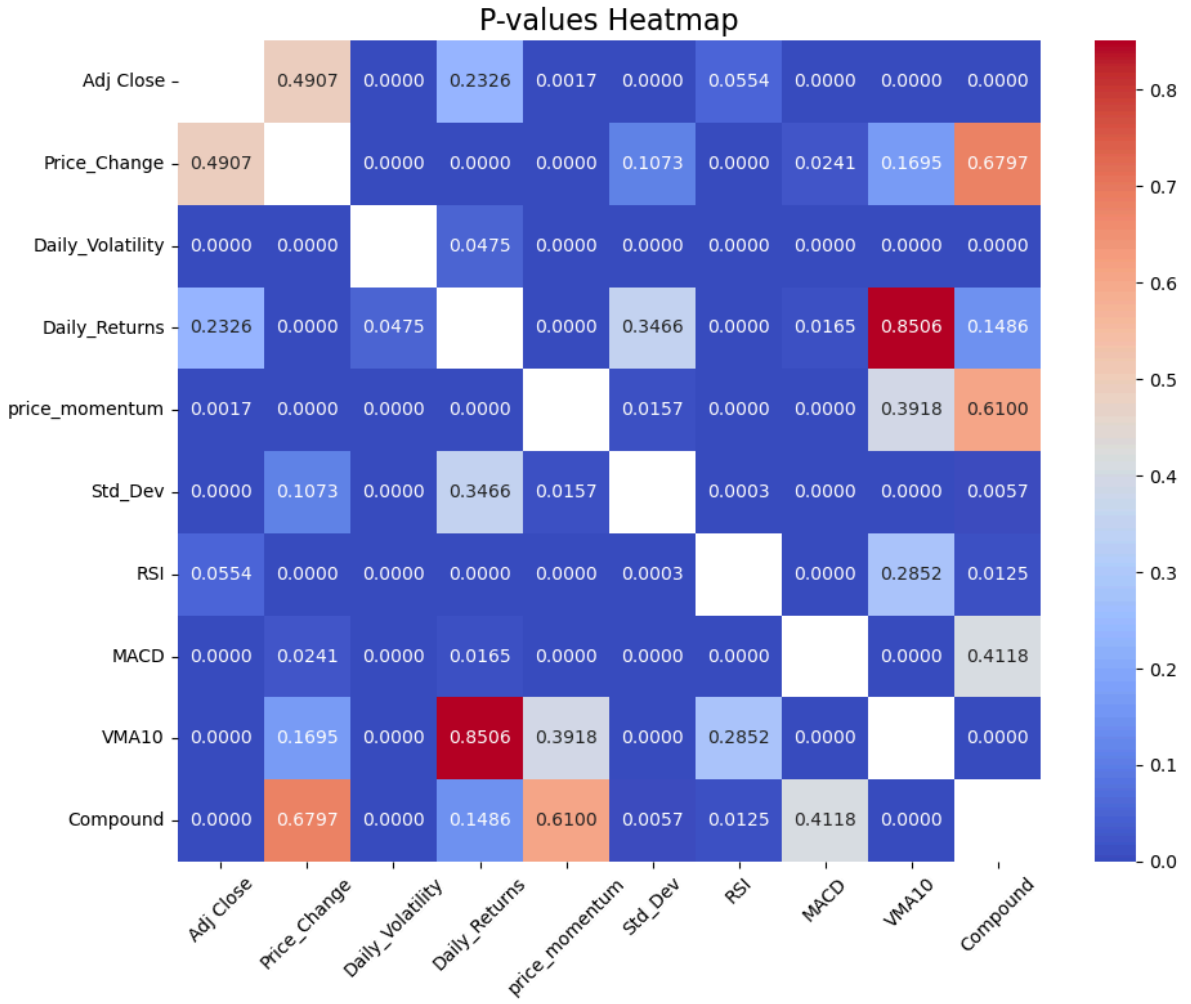
P-values

In [63]:
```
from scipy.stats import pearsonr


def calculate_p_values(df):
    # Select only numeric columns
    numeric_df = df.select_dtypes(include=[np.number])

    # Create a DataFrame to store p-values
    p_values_matrix = pd.DataFrame(index=numeric_df.columns, columns=numeric_df.col

    # Loop through each pair of columns to compute p-values
    for col1 in numeric_df.columns:
        for col2 in numeric_df.columns:
            if col1 == col2:
                p_values_matrix.loc[col1, col2] = np.nan  # Use .loc[] to avoid cha
            else:
                # Calculate the p-value for the pair of columns
                p_values_matrix.loc[col1, col2] = round(pearsonr(numeric_df[col1],

    return p_values_matrix
```

In [64]:
```
p_values = calculate_p_values(Amazon_df)
#print(p_values)
```

In [65]:
```
# Convert the p-values DataFrame to numeric
p_values = p_values.apply(pd.to_numeric, errors='coerce')

# Plot the heatmap for p-values
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(p_values, annot=True, fmt=".4f", cmap='coolwarm', cbar=True)
plt.title("P-values Heatmap", fontsize=16)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



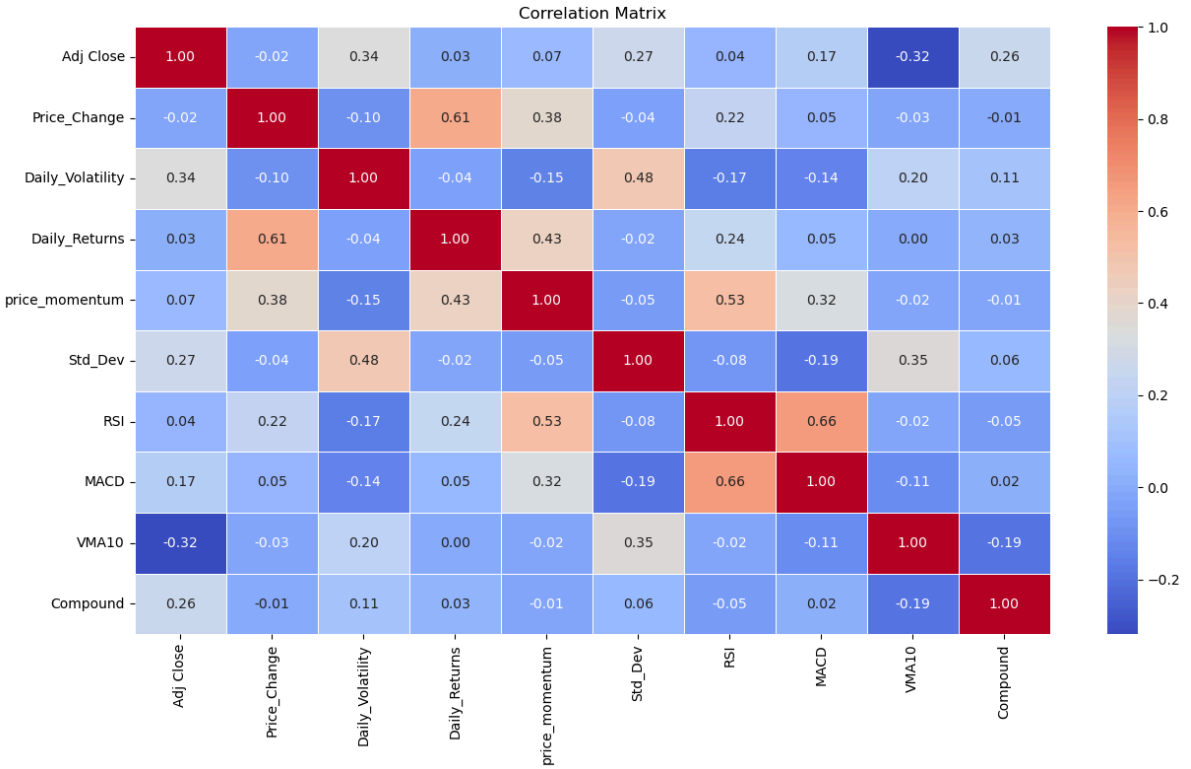
Random Forest Algorithm

In [66]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Target variable is 'Adj Close' for stock price prediction
# Define features (X) and target (y)
X_Amazon = Amazon_df[['Price_Change', 'Daily_Volatility', 'Daily_Returns',
                      'price_momentum', 'Std_Dev', 'RSI', 'MACD', 'VMA10', 'Compound']]
y_Amazon = Amazon_df['Adj Close']

# Split the data into training and testing sets
X_train_Amazon, X_test_Amazon, y_train_Amazon, y_test_Amazon = train_test_split(X_A

# Initialize the RandomForestRegressor
rf = RandomForestRegressor(random_state=42)

# Fit the model
rf.fit(X_train_Amazon, y_train_Amazon)

# Get the feature importance
importances = rf.feature_importances_
```

```
# Create a DataFrame to display the feature importance
feature_importance_df = pd.DataFrame({
    'Feature': X_Amazon.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Display the feature importance
print(feature_importance_df)
```

```
            Feature  Importance
7             VMA10    0.255732
1  Daily_Volatility    0.231502
4           Std_Dev    0.181685
6              MACD    0.095318
8          Compound    0.058081
5               RSI    0.051563
3    price_momentum    0.047902
0      Price_Change    0.042014
2     Daily_Returns    0.036202
```

In [67]:
```
#columns_to_exclude1 = ['price_momentum','Price_Change','Daily_Returns','RSI']
# Create a new DataFrame excluding these columns
#Apple_df1 = Apple_df.drop(columns=columns_to_exclude1)

# Compute the correlation matrix for the remaining columns
correlation_matrix = Amazon_df.corr()

# Optionally, visualize the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(15, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=
plt.title('Correlation Matrix')
plt.show()
```



Variance Inflation matrix to check multicollinearity

In [68]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assuming your data is in a DataFrame called 'df'
# and you want to calculate VIF for the relevant features.

# Define the list of features for which to calculate VIF
features = ['Adj Close', 'Daily_Volatility', 'MACD', 'Std_Dev','VMA10', 'Compound',

# Create a new DataFrame that only includes the features
X = Amazon_df[features]

# Add a constant (bias term) to the model for VIF calculation
X = sm.add_constant(X)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1]

# Display the results
print(vif_data)
```

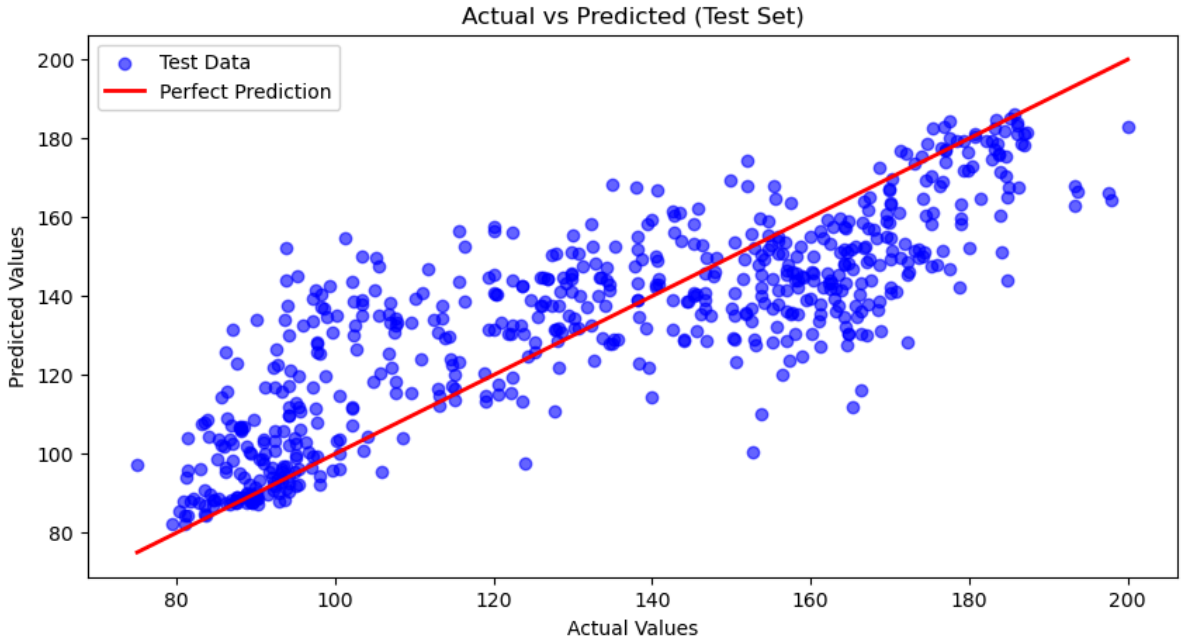
```
             feature        VIF
0              const  57.711309
1          Adj Close   1.688327
2   Daily_Volatility   1.504090
3               MACD   2.041257
4            Std_Dev   1.654250
5              VMA10   1.543238
6           Compound   1.104007
7       Price_Change   1.656176
8      Daily_Returns   1.742393
9     price_momentum   1.662115
10               RSI   2.317086
```

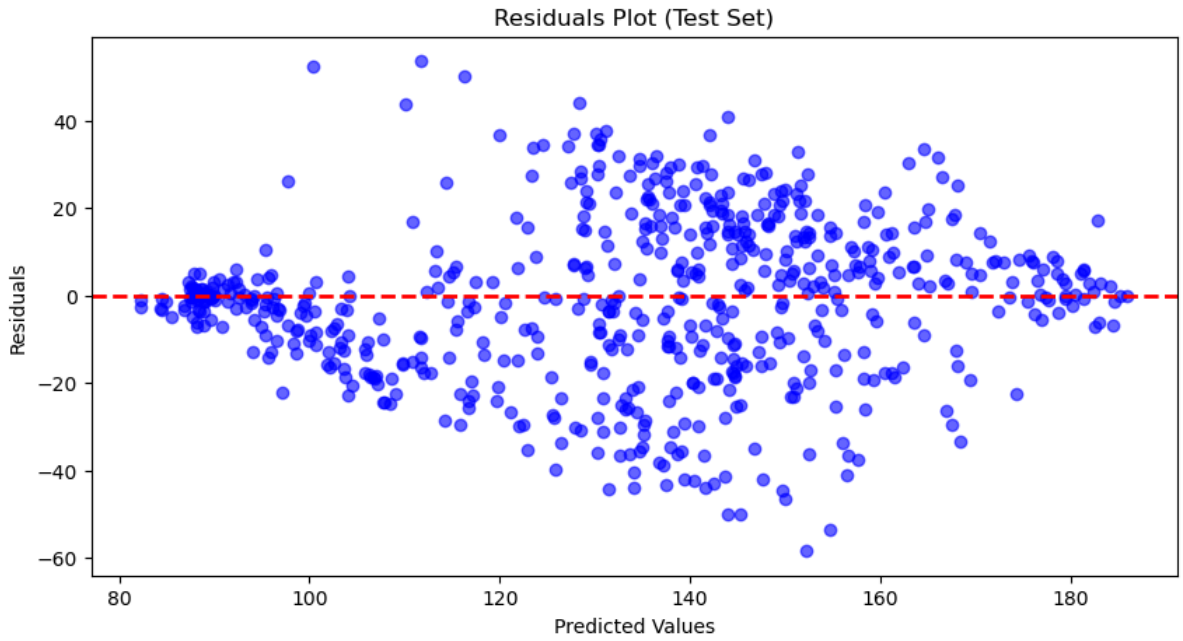
Model Building

In [69]:
```
# Based on Feature importance removing variables price_momentum, Price_Change, Dail
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

X_Amazon1 = Amazon_df[['Price_Change', 'Daily_Volatility', 'Daily_Returns',
             'price_momentum', 'Std_Dev', 'RSI', 'MACD', 'VMA10', 'Compound']]#Sta
y_Amazon1 = Amazon_df['Adj Close']

# Split the data into training and testing sets
X_train_Amazon1, X_test_Amazon1, y_train_Amazon1, y_test_Amazon1 = train_test_split

# Initialize the RandomForestRegressor
rf1 = RandomForestRegressor(n_estimators=20,random_state=42)

# Fit the model
rf_model = rf1.fit(X_train_Amazon1, y_train_Amazon1)

# Make predictions on the test set
```

```
y_rf_pred = rf_model.predict(X_test_Amazon1)

# Calculate evaluation metrics
mse_RF = mean_squared_error(y_test_Amazon1, y_rf_pred)
rmse_RF = np.sqrt(mse_RF)
mae_RF = mean_absolute_error(y_test_Amazon1, y_rf_pred)
r2_RF = r2_score(y_test_Amazon1, y_rf_pred)

# Rounded values (2 decimal places)
mse_rounded_RF = round(mse_RF, 2)
rmse_rounded_RF = round(rmse_RF, 2)
mae_rounded_RF = round(mae_RF, 2)
r2_rounded_RF = round(r2_RF, 2)

# Print the rounded values
print("Random Forest performance evaluation metrics")
print(f"Mean Square Error: {mse_rounded_RF}")
print(f"Root Mean Square Error: {rmse_rounded_RF}")
print(f"Mean Absolute Error: {mae_rounded_RF}")
print(f"R²: {r2_rounded_RF}")

# Predictions
y_train_pred = rf_model.predict(X_train_Amazon1)
y_test_pred = rf_model.predict(X_test_Amazon1)

# Actual vs Predicted Plot
plt.figure(figsize=(10, 5))
plt.scatter(y_test_Amazon1, y_test_pred, alpha=0.6, color="blue", label="Test Data'
plt.plot([y_test_Amazon1.min(), y_test_Amazon1.max()], [y_test_Amazon1.min(), y_tes
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Residuals Plot
residuals = y_test_Amazon1 - y_test_pred

plt.figure(figsize=(10, 5))
plt.scatter(y_test_pred, residuals, alpha=0.6, color="blue")
plt.axhline(y=0, color="red", linestyle="--", lw=2)
plt.title('Residuals Plot (Test Set)')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```

```
Random Forest performance evaluation metrics
Mean Square Error: 349.14
Root Mean Square Error: 18.69
Mean Absolute Error: 14.47
R²: 0.69
```





FineTuning random forest algorithm

In [70]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Assuming your data is in df and the target variable is 'Adj Close'
# Replace 'df' with your DataFrame and target with the appropriate column.

# Features and target
X2 = Amazon_df[['Price_Change', 'Daily_Volatility', 'Daily_Returns',
                'price_momentum', 'Std_Dev', 'RSI', 'MACD', 'VMA10', 'Compound']]
y2 = Amazon_df['Adj Close']

# Split your data into training and testing sets
from sklearn.model_selection import train_test_split
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.2, rand
```

```
# Set up the model
rf2 = RandomForestRegressor(random_state=42)

# Define the parameter grid to search
param_grid = {
    'n_estimators': [10, 20, 30],            # Number of trees in the forest
    'max_depth': [10, 20, 30],            # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],            # Minimum number of samples require
    'min_samples_leaf': [1, 2, 4],            # Minimum number of samples require
    'bootstrap': [True, False]                # Whether bootstrap samples are use
}

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=rf2, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_er

# Fit the model
grid_search.fit(X_train2, y_train2)

# Best parameters found by GridSearchCV
print("Best parameters found: ", grid_search.best_params_)

# Predict using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test2)

# Evaluation metrics
mse = mean_squared_error(y_test2, y_pred)
rmse = np.sqrt(mse)
mae = np.mean(np.abs(y_test2 - y_pred))
r2 = r2_score(y_test2, y_pred)

print(f"Mean Square Error: {mse}")
print(f"Root Mean Square Error: {rmse}")
print(f"Mean Absolute Error: {mae}")
print(f"R²: {r2}")
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best parameters found:  {'bootstrap': True, 'max_depth': 20, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 30}
Mean Square Error: 360.2561597943849
Root Mean Square Error: 18.980415163910006
Mean Absolute Error: 14.302314702101354
R²: 0.6704277728506984
```

Comparing Training and Test performance to check for overfitting

In [71]:

```
from sklearn.metrics import mean_squared_error, r2_score

# Predict on the training and test set
y_train_pred = best_model.predict(X_train2)
y_test_pred = best_model.predict(X_test2)

# Calculate MSE and R² for the training set
train_mse = mean_squared_error(y_train2, y_train_pred)
train_r2 = r2_score(y_train2, y_train_pred)

# Calculate MSE and R² for the test set
test_mse = mean_squared_error(y_test2, y_test_pred)
test_r2 = r2_score(y_test2, y_test_pred)
```

```
# Output results
print(f"Training Set Performance:")
print(f"Mean Square Error (Train): {train_mse:.4f}")
print(f"R² (Train): {train_r2:.4f}")

print(f"\nTest Set Performance:")
print(f"Mean Square Error (Test): {test_mse:.4f}")
print(f"R² (Test): {test_r2:.4f}")

# Check for overfitting
mse_diff = abs(train_mse - test_mse)
r2_diff = abs(train_r2 - test_r2)

# Define a threshold to detect overfitting based on R² or MSE difference
if mse_diff > 0.1 * test_mse or r2_diff > 0.1:  # You can adjust this threshold as
    print("\nWarning: Possible overfitting detected. The model performs significant
else:
    print("\nNo significant overfitting detected. The model generalizes well to the
```

```
Training Set Performance:
Mean Square Error (Train): 60.3684
R² (Train): 0.9473

Test Set Performance:
Mean Square Error (Test): 360.2562
R² (Test): 0.6704

Warning: Possible overfitting detected. The model performs significantly better on
the training set.
```

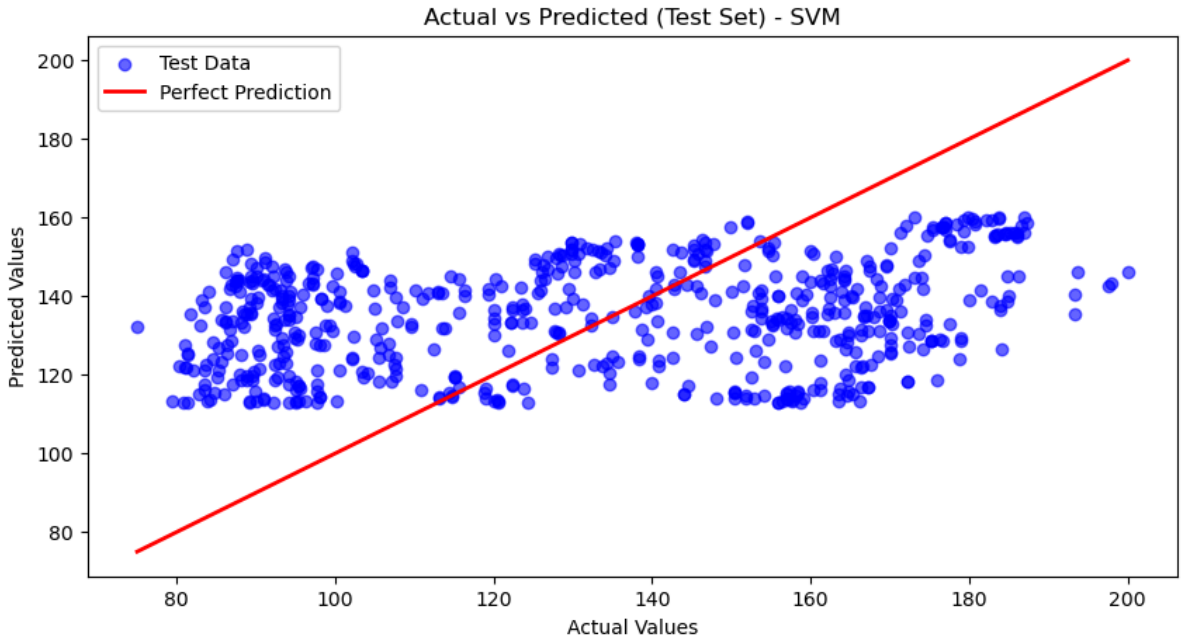
Support Vector Machine

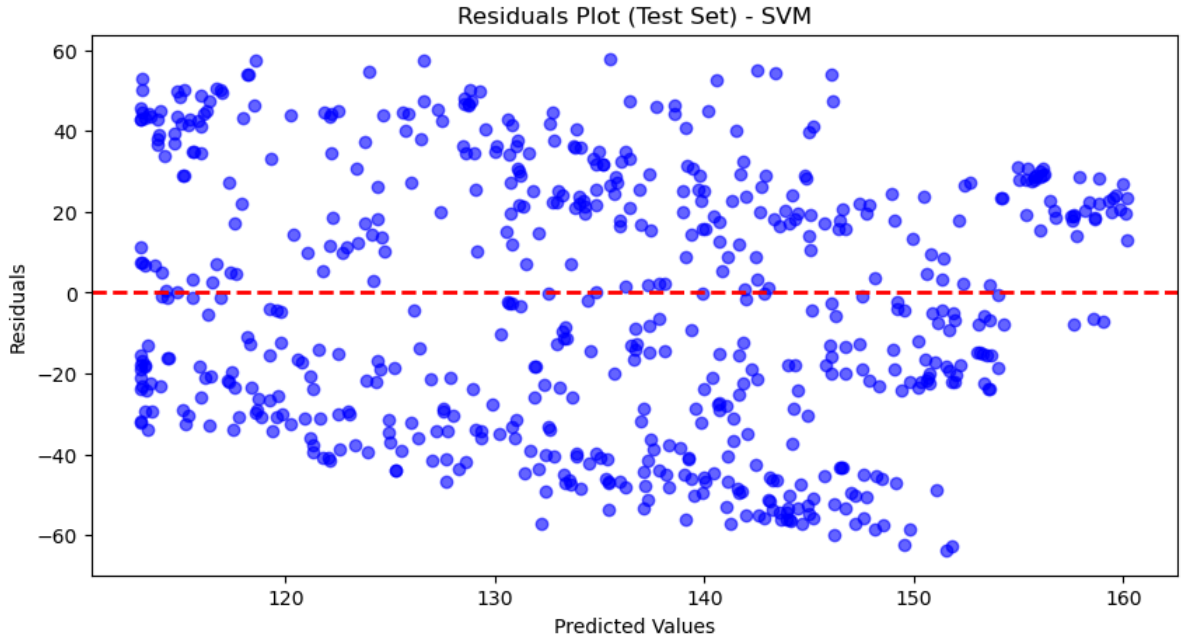
In [72]:

```
from sklearn.svm import SVR


# Assuming Apple_df has already been defined and the necessary columns exist

# Define the features and target
X = Amazon_df[['Price_Change', 'Daily_Volatility', 'Daily_Returns','price_momentum'

y = Amazon_df['Adj Close']

# Split the data into training and testing sets
X_train_svr, X_test_svr, y_train_svr, y_test_svr = train_test_split(X, y, test_size

# Initialize the Support Vector Regressor (SVR)
svr = SVR(kernel='rbf')  # Using RBF kernel (you can also try 'linear', 'poly', etc

# Train the model
svr.fit(X_train_svr, y_train_svr)

# Predict on the test set
y_pred_svr = svr.predict(X_test_svr)

# Calculate evaluation metrics
mse_svr = mean_squared_error(y_test_svr, y_pred_svr)
rmse_svr = np.sqrt(mse_svr)
mae_svr = mean_absolute_error(y_test_svr, y_pred_svr)
r2_svr = r2_score(y_test_svr, y_pred_svr)

# Print evaluation metrics
```

```
print("SVM performance evaluation metrics:")
print(f"Mean Squared Error (MSE): {mse_svr:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_svr:.4f}")
print(f"Mean Absolute Error (MAE): {mae_svr:.4f}")
print(f"R² Score: {r2_svr:.4f}")

# Plot actual vs predicted values (for Test Set)
plt.figure(figsize=(10, 5))
plt.scatter(y_test_svr, y_pred_svr, alpha=0.6, color="blue", label="Test Data")
plt.plot([y_test_svr.min(), y_test_svr.max()], [y_test_svr.min(), y_test_svr.max()]
plt.title('Actual vs Predicted (Test Set) - SVM')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Plot Residuals (for Test Set)
residuals_svr = y_test_svr - y_pred_svr
plt.figure(figsize=(10, 5))
plt.scatter(y_pred_svr, residuals_svr, alpha=0.6, color="blue")
plt.axhline(y=0, color="red", linestyle="--", lw=2)
plt.title('Residuals Plot (Test Set) - SVM')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```

```
SVM performance evaluation metrics:
Mean Squared Error (MSE): 1045.8441
Root Mean Squared Error (RMSE): 32.3395
Mean Absolute Error (MAE): 28.4702
R² Score: 0.0569
```



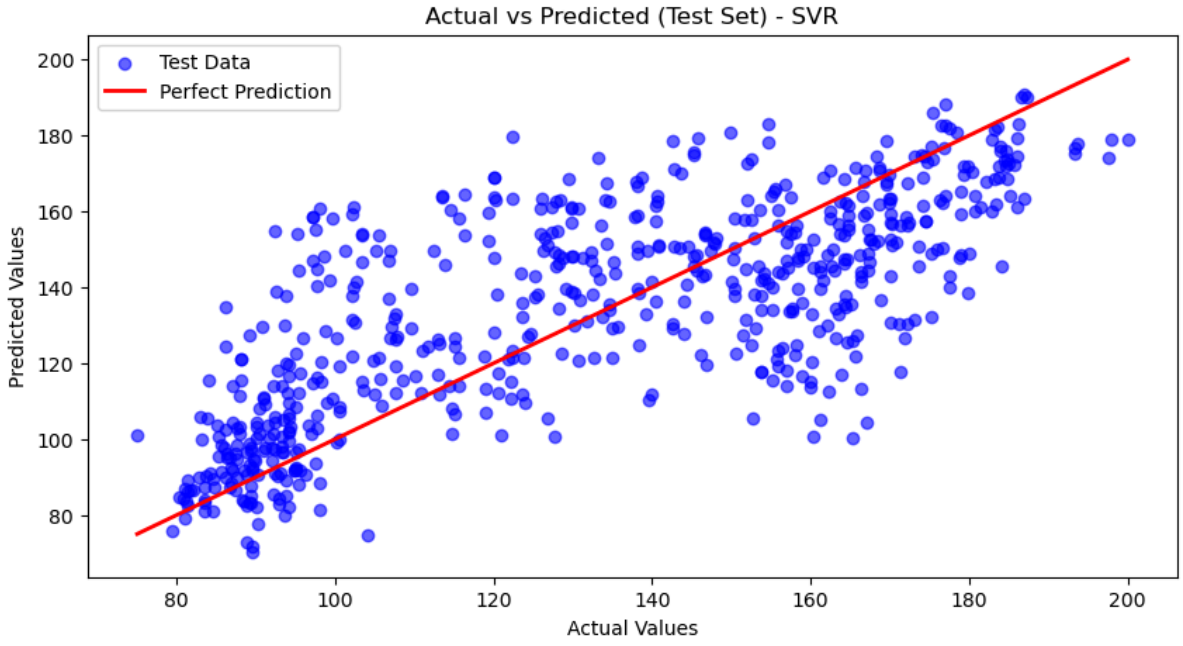


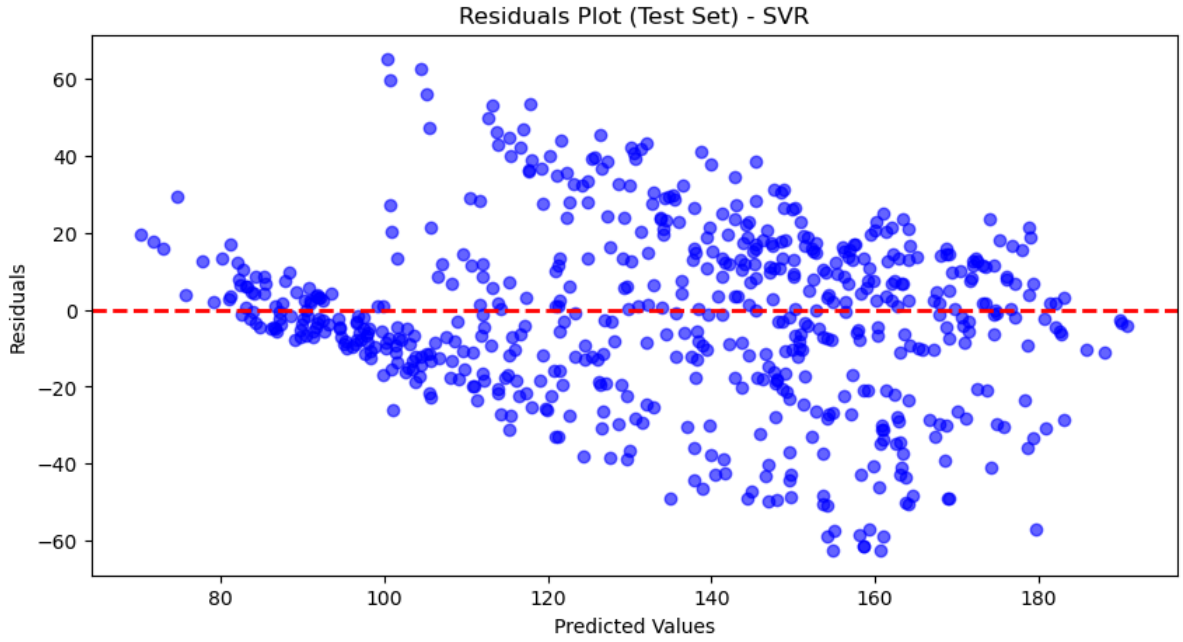
Hyperparameter Tuning - SVM

In [73]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming X_train_svr, y_train_svr, X_test_svr, y_test_svr are defined

# Step 1: Feature Scaling (important for SVR)
scaler_X = StandardScaler()
X_train_svr_scaled = scaler_X.fit_transform(X_train_svr)
X_test_svr_scaled = scaler_X.transform(X_test_svr)

# Reshape y to 2D and scale
scaler_y = StandardScaler()
y_train_svr_scaled = scaler_y.fit_transform(y_train_svr.values.reshape(-1, 1)).rave

# Step 2: Define a smaller, more focused parameter grid
param_grid_svr = {
    'C': [1, 10, 100],           # Narrowed down search for C
    'epsilon': [0.01, 0.1, 1],   # Using reasonable values for epsilon
    'kernel': ['linear', 'rbf', 'poly']  # Trying multiple kernels
}

# Step 3: Perform GridSearchCV with fewer cross-validation folds (cv=3 for faster e
grid_search_svr = GridSearchCV(SVR(), param_grid_svr, cv=3, scoring='neg_mean_squar

# Step 4: Fit the model using the scaled data
grid_search_svr.fit(X_train_svr_scaled, y_train_svr_scaled)

# Step 5: Get the best parameters found by GridSearch
print("Best parameters found: ", grid_search_svr.best_params_)

# Step 6: Predict using the best SVR model
best_svr = grid_search_svr.best_estimator_
y_pred_svr_scaled = best_svr.predict(X_test_svr_scaled)
# Reshape the 1D array to 2D for inverse transform
y_pred_svr_scaled_2d = y_pred_svr_scaled.reshape(-1, 1)
```

```
# Inverse transform the predicted values back to the original scale
y_pred_svr = scaler_y.inverse_transform(y_pred_svr_scaled_2d).ravel()  # Flatten ba


# Inverse transform y_test_svr (for calculating metrics in the original scale)
y_test_svr_rescaled = y_test_svr.values  # Use original values if already in the co

# Step 7: Calculate and print evaluation metrics
mse_svr = mean_squared_error(y_test_svr_rescaled, y_pred_svr)
rmse_svr = np.sqrt(mse_svr)
mae_svr = mean_absolute_error(y_test_svr_rescaled, y_pred_svr)
r2_svr = r2_score(y_test_svr_rescaled, y_pred_svr)

print(f"Best SVR model performance metrics:")
print(f"RMSE: {rmse_svr:.4f}, MAE: {mae_svr:.4f}, R²: {r2_svr:.4f}")

# Optional: Plot Actual vs Predicted values (for Test Set)
plt.figure(figsize=(10, 5))
plt.scatter(y_test_svr_rescaled, y_pred_svr, alpha=0.6, color="blue", label="Test D
plt.plot([y_test_svr_rescaled.min(), y_test_svr_rescaled.max()], [y_test_svr_rescal
plt.title('Actual vs Predicted (Test Set) - SVR')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Plot Residuals (for Test Set)
residuals_svr = y_test_svr_rescaled - y_pred_svr
plt.figure(figsize=(10, 5))
plt.scatter(y_pred_svr, residuals_svr, alpha=0.6, color="blue")
plt.axhline(y=0, color="red", linestyle="--", lw=2)
plt.title('Residuals Plot (Test Set) - SVR')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
Best parameters found:  {'C': 1, 'epsilon': 0.1, 'kernel': 'rbf'}
Best SVR model performance metrics:
RMSE: 22.5979, MAE: 17.4352, R²: 0.5395
```





Comparing Training and Test performance to check for overfitting

In [74]:
```
# Predict on training data (scaled)
y_train_pred_svr_scaled = best_svr.predict(X_train_svr_scaled)
y_train_pred_svr_scaled_2d = y_train_pred_svr_scaled.reshape(-1, 1)

# Inverse transform the training predictions back to the original scale
y_train_pred_svr = scaler_y.inverse_transform(y_train_pred_svr_scaled_2d).ravel()

# Predict on test data (scaled)
y_pred_svr_scaled = best_svr.predict(X_test_svr_scaled)
y_pred_svr_scaled_2d = y_pred_svr_scaled.reshape(-1, 1)

# Inverse transform the test predictions back to the original scale
y_pred_svr = scaler_y.inverse_transform(y_pred_svr_scaled_2d).ravel()

# Inverse transform y_train_svr and y_test_svr (for calculating metrics in the orig
y_train_svr_rescaled = scaler_y.inverse_transform(y_train_svr_scaled.reshape(-1, 1)
y_test_svr_rescaled = y_test_svr.values  # Assuming this is already in original sca

# Step 7: Calculate and print evaluation metrics

# Training set performance
mse_train_svr = mean_squared_error(y_train_svr_rescaled, y_train_pred_svr)
rmse_train_svr = np.sqrt(mse_train_svr)
mae_train_svr = mean_absolute_error(y_train_svr_rescaled, y_train_pred_svr)
r2_train_svr = r2_score(y_train_svr_rescaled, y_train_pred_svr)

print(f"Training Performance:")
print(f"Train RMSE: {rmse_train_svr:.4f}, Train MAE: {mae_train_svr:.4f}, Train R²:

# Test set performance
mse_test_svr = mean_squared_error(y_test_svr_rescaled, y_pred_svr)
rmse_test_svr = np.sqrt(mse_test_svr)
mae_test_svr = mean_absolute_error(y_test_svr_rescaled, y_pred_svr)
r2_test_svr = r2_score(y_test_svr_rescaled, y_pred_svr)

print(f"Test Performance:")
print(f"Test RMSE: {rmse_test_svr:.4f}, Test MAE: {mae_test_svr:.4f}, Test R²: {r2_
```

```
# Compare training and test performance for overfitting check
if abs(r2_train_svr - r2_test_svr) > 0.1:
    print("Warning: Possible overfitting detected.")
else:
    print("No significant overfitting detected.")
```

```
Training Performance:
Train RMSE: 20.2049, Train MAE: 14.7317, Train R²: 0.6432
Test Performance:
Test RMSE: 22.5979, Test MAE: 17.4352, Test R²: 0.5395
Warning: Possible overfitting detected.
```

ARIMA MODEL

In [75]:
```
from pmdarima import auto_arima
Amazon_df
```

Out[75]:

Date	Adj Close	Price_Change	Daily_Volatility	Daily_Returns	price_momentum	Std_Dev	
2019-01-01	76.956497	3.696495	4.621498	0.000000	1.813004	2.970426	66.253
2019-01-02	76.956497	3.696495	4.621498	0.000000	1.813004	2.970426	66.253
2019-01-03	75.014000	-0.986504	2.044502	-2.524150	1.813004	2.970426	66.253
2019-01-04	78.769501	2.269501	3.784500	5.006400	1.813004	2.970426	66.253
2019-01-05	78.769501	2.269501	3.784500	0.000000	1.813004	2.970426	66.253
...	...	...	...	...	...	...	
2024-09-09	175.399994	0.869995	2.340012	2.339690	2.069992	2.952360	49.846
2024-09-10	179.550003	2.059998	3.710007	2.366026	1.660004	2.908640	59.343
2024-09-11	184.520004	4.419998	9.260010	2.768032	13.130005	3.544512	68.510
2024-09-12	187.000000	2.199997	3.870010	1.344025	15.610001	4.327921	69.466
2024-09-13	186.490005	-0.509995	2.589996	-0.272724	15.100006	4.883247	62.349

2083 rows × 10 columns

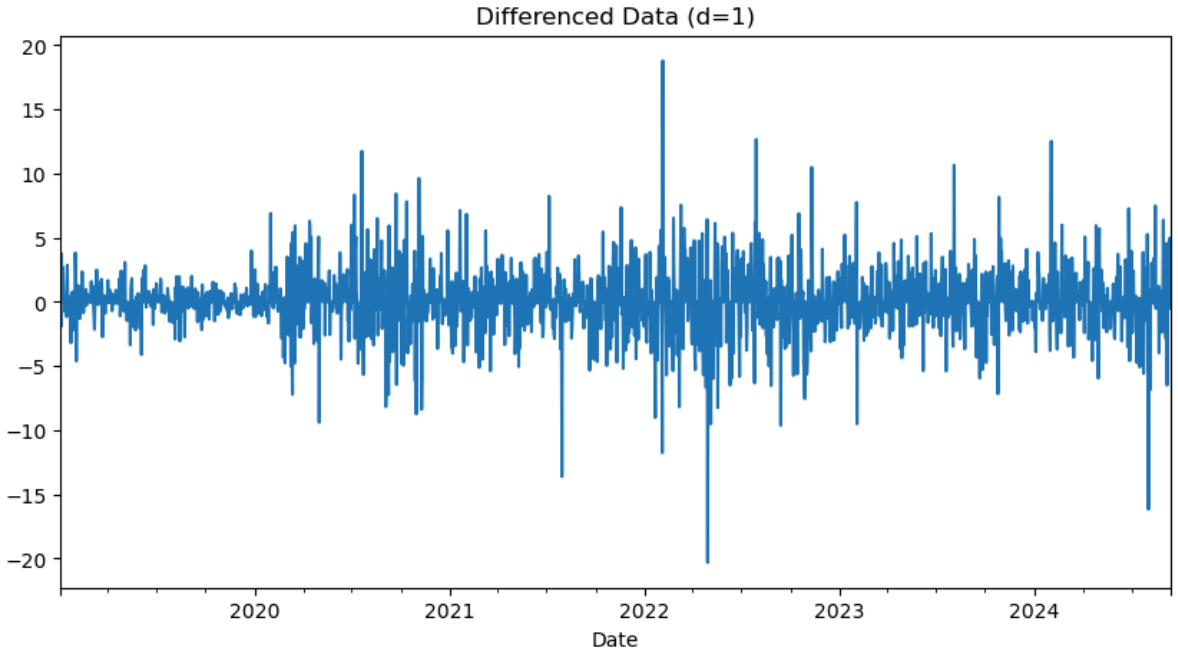
In [76]:
```
from statsmodels.tsa.stattools import adfuller
from pmdarima.arima.utils import ndiffs
from statsmodels.tsa.arima.model import ARIMA

# move the target variable to stock_data
stock_data = Amazon_df['Adj Close']
```

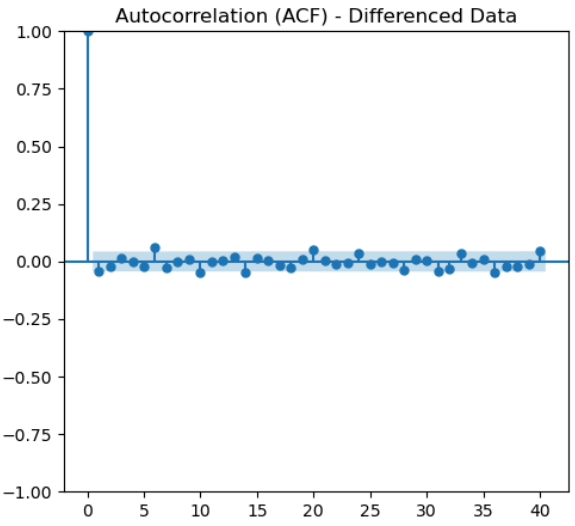
```
# Perform Dickey-Fuller test
adf_test = adfuller(stock_data)

# Perform the Augmented Dickey-Fuller test to check for stationarity
result = adfuller(stock_data)

print('ADF Statistic:', result[0])
print('p-value:', result[1])

# If p-value is less than 0.05, the data is stationary
if result[1] > 0.05:
    print('Data is non-stationary, consider differencing')
else:
    print('Data is stationary')

# Check if differencing is required using ndiffs
d_value = ndiffs(stock_data, test="adf")
print(f"Recommended d value (number of differences): {d_value}")
```

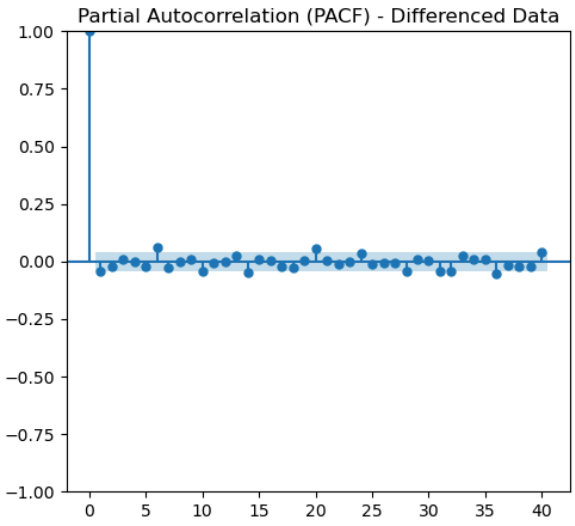
```
ADF Statistic: -1.6095124909016123
p-value: 0.47881350515259813
Data is non-stationary, consider differencing
Recommended d value (number of differences): 1
```

In [77]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Perform differencing (d_value times)
stock_data_diff = stock_data.diff(d_value).dropna()

# Plot the differenced data
plt.figure(figsize=(10, 5))
stock_data_diff.plot(title=f'Differenced Data (d={d_value})')
plt.show()

# Plot ACF and PACF for the differenced data to determine p and q
plt.figure(figsize=(12, 5))

# Autocorrelation plot (ACF) for q
plt.subplot(121)
plot_acf(stock_data_diff, lags=40, ax=plt.gca())
plt.title('Autocorrelation (ACF) - Differenced Data')

# Partial Autocorrelation plot (PACF) for p
plt.subplot(122)
plot_pacf(stock_data_diff, lags=40, ax=plt.gca())
plt.title('Partial Autocorrelation (PACF) - Differenced Data')

plt.show()
```







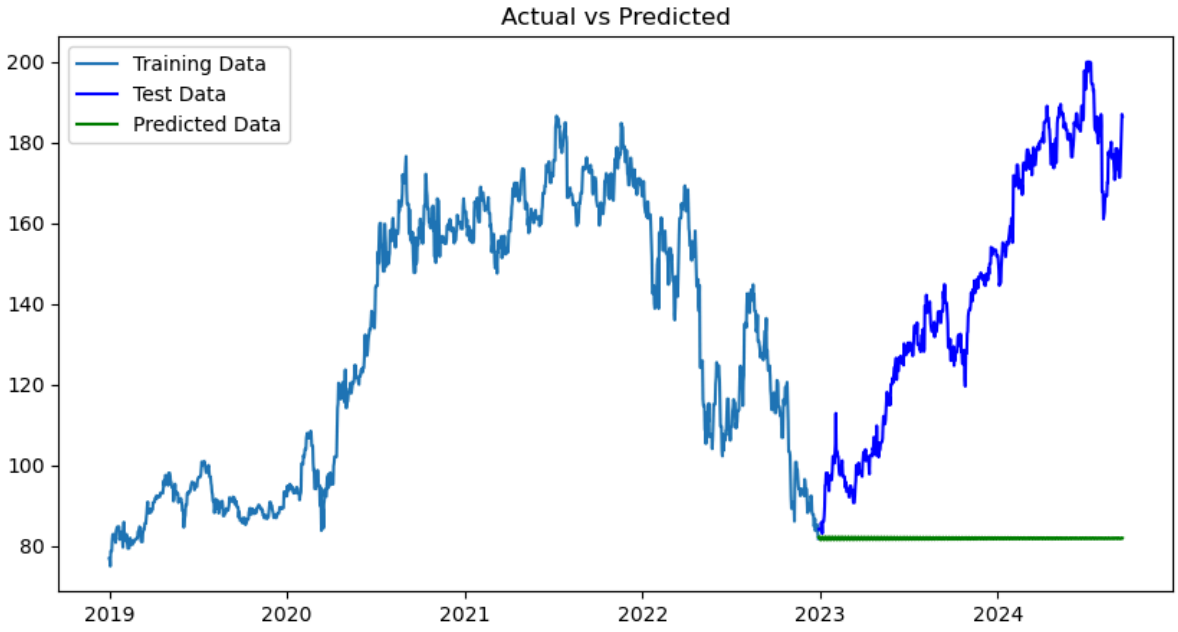
In [78]:
```
from pmdarima import auto_arima

# Fit the auto_arima model
auto_model = auto_arima(stock_data, seasonal=False, stepwise=True, trace=True)

# Output the best p, d, q values
print(f"Best p: {auto_model.order[0]}, Best d: {auto_model.order[1]}, Best q: {auto
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=9518.249, Time=2.19 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=9520.111, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=9518.743, Time=0.15 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=9518.583, Time=0.16 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=9519.129, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=9521.472, Time=0.74 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=9521.395, Time=1.26 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=9516.473, Time=3.11 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=9523.316, Time=0.42 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=9520.498, Time=3.79 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=9520.215, Time=3.42 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=9516.731, Time=3.01 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=9525.257, Time=2.08 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=9520.435, Time=2.70 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=9524.439, Time=0.53 sec

Best model:  ARIMA(3,1,2)(0,0,0)[0] intercept
Total fit time: 23.673 seconds
Best p: 3, Best d: 1, Best q: 2
```

In [80]:

```
# Split data into training and testing sets (70% training, 30% testing)
train_size = int(len(stock_data) * 0.7)
train_data, test_data = stock_data[:train_size], stock_data[train_size:]

# Build ARIMA model with p=3, d=1, q=2
arima_model = ARIMA(train_data, order=(3, 1, 2))

# Fit the ARIMA model
arima_result = arima_model.fit()

# Forecasting for the test period
forecast_arima = arima_result.forecast(steps=len(test_data))

# Plot actual vs predicted values
plt.figure(figsize=(10, 5))
plt.plot(train_data, label='Training Data')
plt.plot(test_data, label='Test Data', color='blue')
plt.plot(test_data.index, forecast_arima, label='Predicted Data', color='green')
plt.title('Actual vs Predicted')
plt.legend()
plt.show()

# Evaluate the model performance using various metrics
mse_arima = mean_squared_error(test_data, forecast_arima)
rmse_arima = np.sqrt(mse_arima)
mae_arima = mean_absolute_error(test_data, forecast_arima)
r2_arima = r2_score(test_data, forecast_arima)

# Print evaluation metrics
print("ARIMA model evaluation metrics:")
print(f"Mean Squared Error (MSE): {mse_arima:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_arima:.4f}")
print(f"Mean Absolute Error (MAE): {mae_arima:.4f}")
print(f"R² Score: {r2_arima:.4f}")
```

```
C:\Users\97152\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency D will
be used.
  self._init_dates(dates, freq)
C:\Users\97152\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency D will
be used.
  self._init_dates(dates, freq)
C:\Users\97152\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency D will
be used.
  self._init_dates(dates, freq)
C:\Users\97152\anaconda3\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:9
66: UserWarning: Non-stationary starting autoregressive parameters found. Using ze
ros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
C:\Users\97152\anaconda3\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:9
78: UserWarning: Non-invertible starting MA parameters found. Using zeros as start
ing parameters.
  warn('Non-invertible starting MA parameters found.'
```

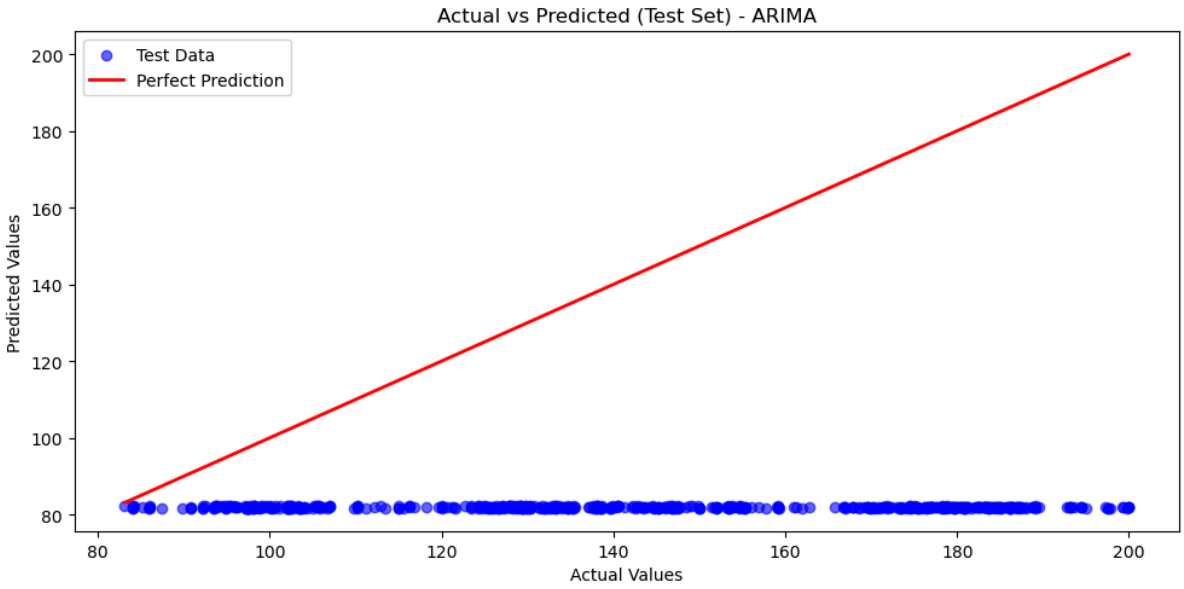


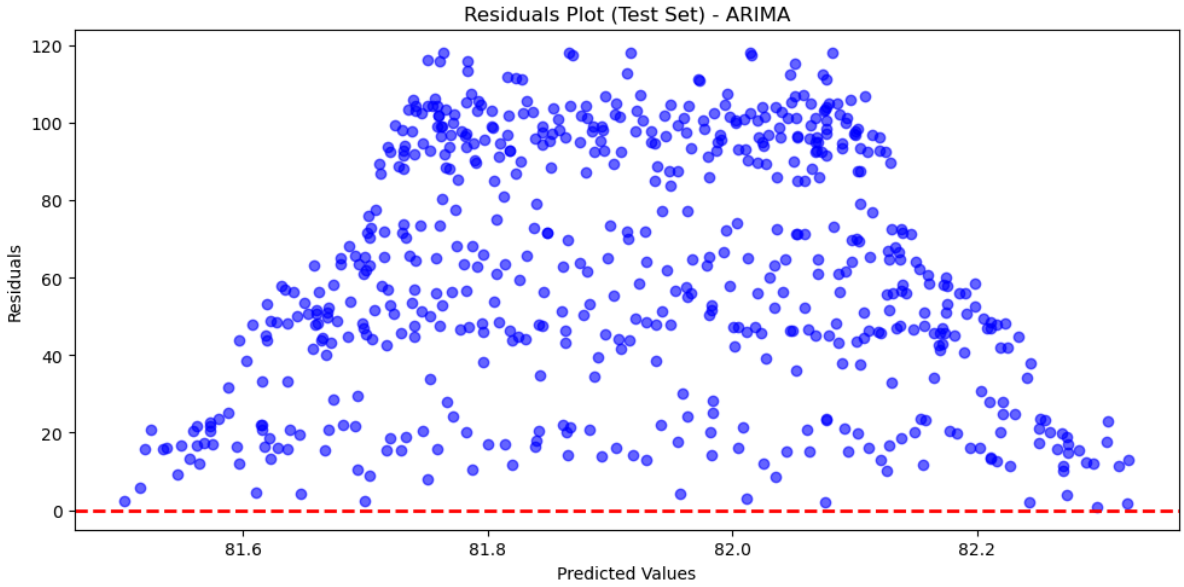

```
ARIMA model evaluation metrics:
Mean Squared Error (MSE): 4899.7480
Root Mean Squared Error (RMSE): 69.9982
Mean Absolute Error (MAE): 62.1898
R² Score: -3.7472
```

In [81]:

```
# Import necessary libraries
import matplotlib.pyplot as plt

# Assume test_data and forecast_arima are already defined and contain actual and pr

# Calculate the residuals
residuals = test_data - forecast_arima

# Create a figure with two subplots: one for actual vs predicted, one for residuals
fig, axs = plt.subplots(2, 1, figsize=(10, 10))

# Subplot 1: Scatter plot for actual vs predicted values
axs[0].scatter(test_data, forecast_arima, alpha=0.6, color="blue", label="Test Data
axs[0].plot([test_data.min(), test_data.max()], [test_data.min(), test_data.max()],
axs[0].set_title('Actual vs Predicted (Test Set) - ARIMA')
axs[0].set_xlabel('Actual Values')
axs[0].set_ylabel('Predicted Values')
axs[0].legend()
```

```
# Subplot 2: Residual plot
axs[1].scatter(forecast_arima, residuals, alpha=0.6, color="blue")
axs[1].axhline(y=0, color='red', linestyle='--', lw=2)
axs[1].set_title('Residuals Plot (Test Set) - ARIMA')
axs[1].set_xlabel('Predicted Values')
axs[1].set_ylabel('Residuals')

# Show the plot
plt.tight_layout()
plt.show()
```





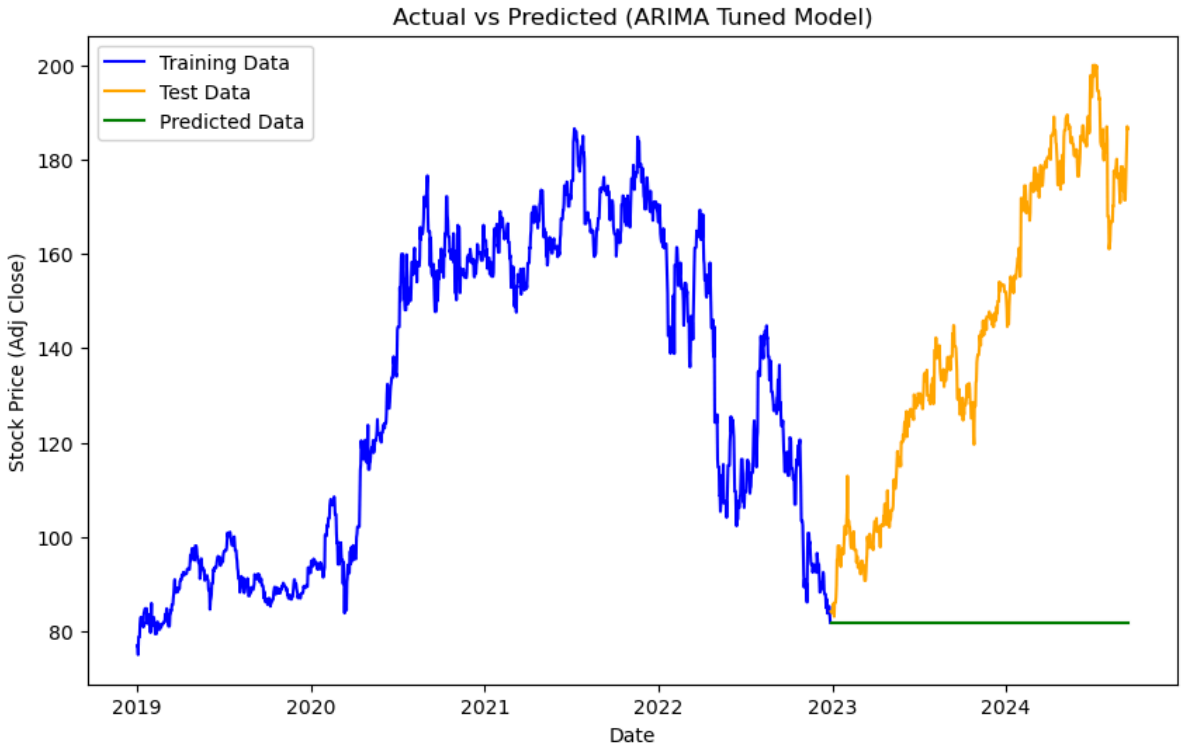
Hyperparameter Tuning - ARIMA

In [82]:

```
# Use auto_arima to automatically tune the ARIMA model
model = auto_arima(train_data,
                   start_p=0, start_q=0,
                   max_p=5, max_q=5,
                   d=1,           # Differencing set to 1
                   seasonal=False, # Non-seasonal ARIMA
                   trace=True,     # Show the tuning process
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True)
```

```
# Summary of the best ARIMA model
print(model.summary())

# Forecast for the test period
n_periods = len(test_data)
forecast_arima = model.predict(n_periods=n_periods)

# Plot training data, test data, and predicted data
plt.figure(figsize=(10, 6))

# Plot training data
plt.plot(train_data.index, train_data, label='Training Data', color='blue')

# Plot test data
plt.plot(test_data.index, test_data, label='Test Data', color='orange')

# Plot forecasted data (predicted data)
plt.plot(test_data.index, forecast_arima, label='Predicted Data', color='green')

# Adding title and labels
plt.title('Actual vs Predicted (ARIMA Tuned Model)')
plt.xlabel('Date')
plt.ylabel('Stock Price (Adj Close)')

# Add a legend
plt.legend()

# Show the plot
plt.show()

# Evaluate model performance
mse_arima = mean_squared_error(test_data, forecast_arima)
rmse_arima = np.sqrt(mse_arima)
mae_arima = mean_absolute_error(test_data, forecast_arima)
r2_arima = r2_score(test_data, forecast_arima)

# Print evaluation metrics
print("Tuned ARIMA model evaluation metrics:")
print(f"Mean Squared Error (MSE): {mse_arima:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_arima:.4f}")
print(f"Mean Absolute Error (MAE): {mae_arima:.4f}")
print(f"R² Score: {r2_arima:.4f}")
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=6716.153, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=6715.366, Time=0.10 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=6715.380, Time=0.12 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=6714.156, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=6717.364, Time=0.14 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 0.430 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1458
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -3356.078
Date:                Thu, 26 Sep 2024   AIC                           6714.156
Time:                        13:18:10   BIC                           6719.440
Sample:                    01-01-2019   HQIC                          6716.127
                         - 12-28-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2         5.8647      0.092     64.036      0.000       5.685       6.044
===================================================================================
=
Ljung-Box (L1) (Q):                   2.79   Jarque-Bera (JB):              5211.5
8
Prob(Q):                              0.09   Prob(JB):                         0.0
0
Heteroskedasticity (H):               4.05   Skew:                            -0.1
8
Prob(H) (two-sided):                  0.00   Kurtosis:                        12.2
6
===================================================================================
=

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



```
Tuned ARIMA model evaluation metrics:
Mean Squared Error (MSE): 4912.0027
Root Mean Squared Error (RMSE): 70.0857
Mean Absolute Error (MAE): 62.2886
R² Score: -3.7591
```

LSTM

In [83]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
Amazon_df
```

Out[83]:

Date	Adj Close	Price_Change	Daily_Volatility	Daily_Returns	price_momentum	Std_Dev	
2019-01-01	76.956497	3.696495	4.621498	0.000000	1.813004	2.970426	66.253
2019-01-02	76.956497	3.696495	4.621498	0.000000	1.813004	2.970426	66.253
2019-01-03	75.014000	-0.986504	2.044502	-2.524150	1.813004	2.970426	66.253
2019-01-04	78.769501	2.269501	3.784500	5.006400	1.813004	2.970426	66.253
2019-01-05	78.769501	2.269501	3.784500	0.000000	1.813004	2.970426	66.253
...	...	...	...	...	...	...	
2024-09-09	175.399994	0.869995	2.340012	2.339690	2.069992	2.952360	49.846
2024-09-10	179.550003	2.059998	3.710007	2.366026	1.660004	2.908640	59.343
2024-09-11	184.520004	4.419998	9.260010	2.768032	13.130005	3.544512	68.510
2024-09-12	187.000000	2.199997	3.870010	1.344025	15.610001	4.327921	69.466
2024-09-13	186.490005	-0.509995	2.589996	-0.272724	15.100006	4.883247	62.349

2083 rows × 10 columns

In [84]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# 1. Load and preprocess the data
df = Amazon_df.copy()  # Assuming Apple_df is a DataFrame
```

```
# 2. Normalize the data (MinMaxScaler)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)

# 3. Prepare the data for LSTM
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), :])  # Use all columns (features)
        y.append(data[i + time_step, 0])  # Predict the 'Adj Close' (first column)
    return np.array(X), np.array(y)

# Define the number of time steps (e.g., use the last 60 days to predict the next v
time_step = 10
X, y = create_dataset(scaled_data, time_step)

# 4. Split the data into training and test sets
train_size = int(len(X) * 0.7)  # 70% training data, 30% testing
X_train_lstm, X_test_lstm = X[:train_size], X[train_size:]
y_train_lstm, y_test_lstm = y[:train_size], y[train_size:]

# 5. Reshape data into 3D (samples, time steps, features) for LSTM
X_train_lstm = X_train_lstm.reshape(X_train_lstm.shape[0], X_train_lstm.shape[1], X
X_test_lstm = X_test_lstm.reshape(X_test_lstm.shape[0], X_test_lstm.shape[1], X_tes

# 6. Build the LSTM model using TensorFlow
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(time_step, X_train_lstm.shap
model.add(Dropout(0.2))  # Dropout to prevent overfitting
model.add(LSTM(50, return_sequences=False))  # Second LSTM layer
model.add(Dropout(0.2))
model.add(Dense(25))  # Dense layer with 25 units
model.add(Dense(1))  # Output layer predicting one value (Adj Close)

# 7. Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# 8. Train the model
history = model.fit(X_train_lstm, y_train_lstm, epochs=50, batch_size=32, validatio

# 9. Evaluate and make predictions
y_train_pred = model.predict(X_train_lstm)
y_test_pred = model.predict(X_test_lstm)

# 10. Invert scaling to get the original scale for 'Adj Close' only (don't concaten
# Create dummy arrays to match the feature shape for inverse scaling
y_train_pred = scaler.inverse_transform(np.concatenate((y_train_pred, np.zeros((y_t
y_test_pred = scaler.inverse_transform(np.concatenate((y_test_pred, np.zeros((y_tes

# Do the same for the true y values
y_train_lstm = scaler.inverse_transform(np.concatenate((y_train_lstm.reshape(-1, 1)
y_test_lstm = scaler.inverse_transform(np.concatenate((y_test_lstm.reshape(-1, 1),

# 11. Calculate RMSE
train_rmse = np.sqrt(mean_squared_error(y_train_lstm, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test_lstm, y_test_pred))
print(f"Train RMSE: {train_rmse}")
print(f"Test RMSE: {test_rmse}")

# Calculate other metrics
mse_lstm = mean_squared_error(y_test_lstm, y_test_pred)
rmse_lstm = np.sqrt(mse_lstm)
mae_lstm = mean_absolute_error(y_test_lstm, y_test_pred)
```

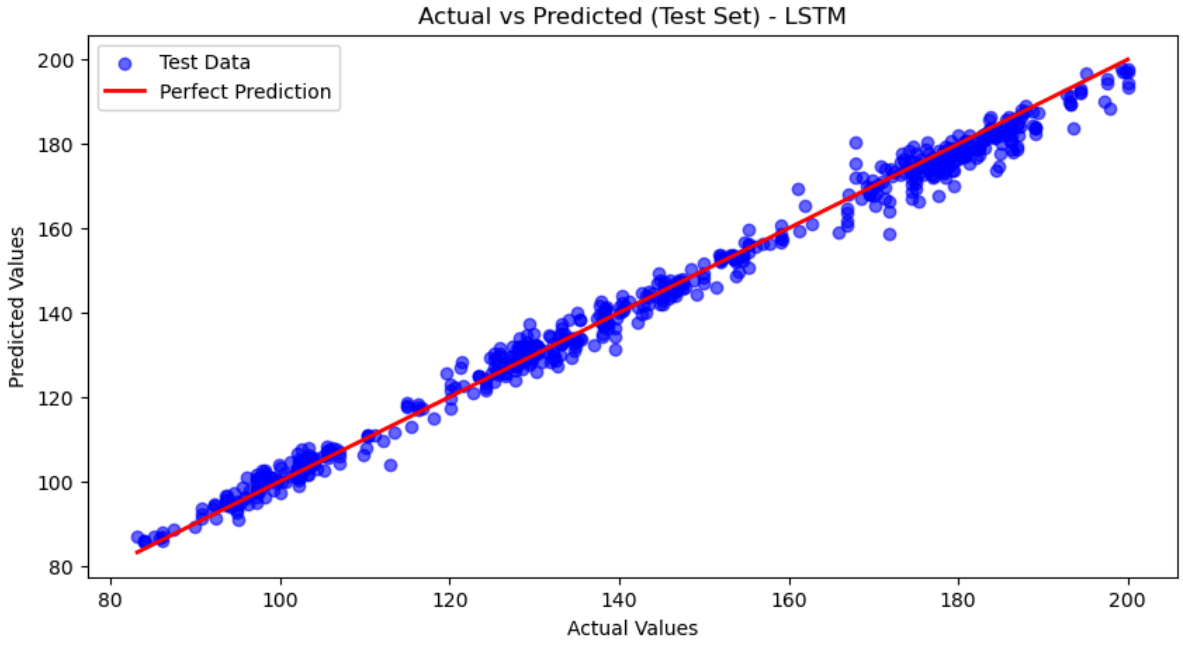
```
r2_lstm = r2_score(y_test_lstm, y_test_pred)

# Print evaluation metrics
print("LSTM performance evaluation metrics:")
print(f"Mean Squared Error (MSE): {mse_lstm:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_lstm:.4f}")
print(f"Mean Absolute Error (MAE): {mae_lstm:.4f}")
print(f"R² Score: {r2_lstm:.4f}")

# 12. Plot Actual vs Predicted values (for Test Set)
plt.figure(figsize=(10, 5))
plt.scatter(y_test_lstm, y_test_pred, alpha=0.6, color="blue", label="Test Data")
plt.plot([y_test_lstm.min(), y_test_lstm.max()], [y_test_lstm.min(), y_test_lstm.ma
plt.title('Actual vs Predicted (Test Set) - LSTM')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# 13. Plot Residuals (for Test Set)
residuals_lstm = y_test_lstm - y_test_pred
plt.figure(figsize=(10, 5))
plt.scatter(y_test_pred, residuals_lstm, alpha=0.6, color="blue")
plt.axhline(y=0, color="red", linestyle="--", lw=2)
plt.title('Residuals Plot (Test Set) - LSTM')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```

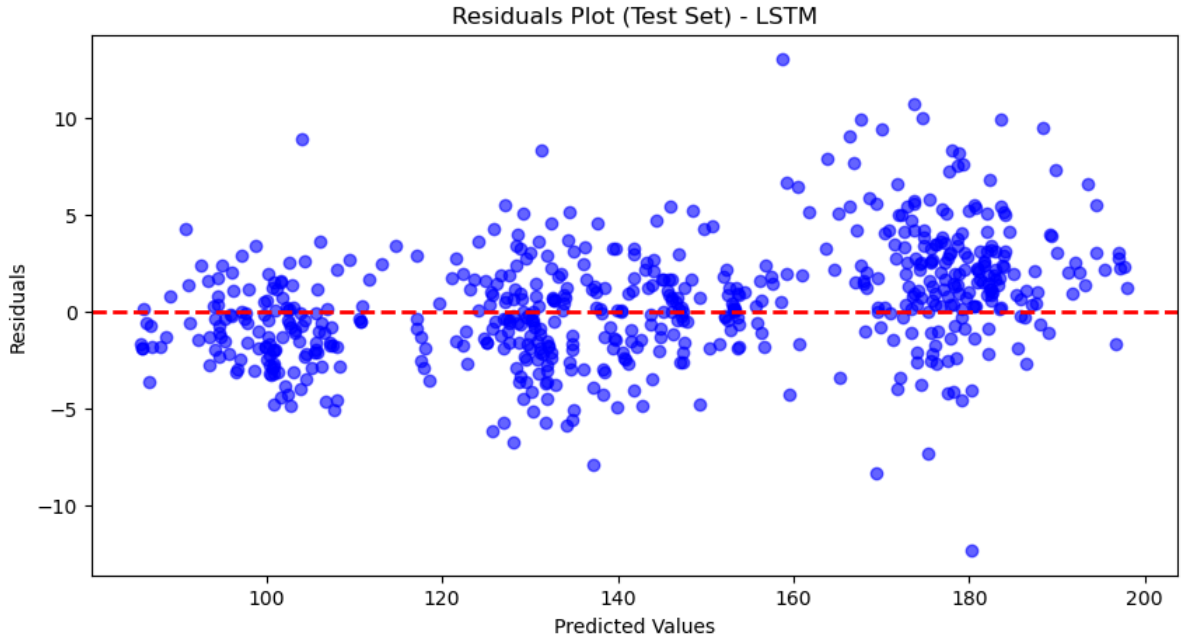
```
Epoch 1/50
```

```
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - loss: 0.0609 - val_loss: 0.0021
Epoch 2/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0056 - val_loss: 0.0016
Epoch 3/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0044 - val_loss: 0.0041
Epoch 4/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0042 - val_loss: 0.0029
Epoch 5/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0033 - val_loss: 0.0044
Epoch 6/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0034 - val_loss: 0.0029
Epoch 7/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0028 - val_loss: 9.1502e-04
Epoch 8/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0026 - val_loss: 0.0011
Epoch 9/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0025 - val_loss: 0.0011
Epoch 10/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0023 - val_loss: 0.0010
Epoch 11/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0023 - val_loss: 0.0012
Epoch 12/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0020 - val_loss: 0.0012
Epoch 13/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0021 - val_loss: 0.0015
Epoch 14/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 15/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0019 - val_loss: 0.0014
Epoch 16/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0018 - val_loss: 0.0021
Epoch 17/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0020 - val_loss: 8.3044e-04
Epoch 18/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0017 - val_loss: 9.4400e-04
Epoch 19/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0016 - val_loss: 0.0012
Epoch 20/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0021 - val_loss: 7.6061e-04
Epoch 21/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0018 - val_loss: 0.0016
Epoch 22/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0020 - val_loss: 8.6973e-04
Epoch 23/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0018 - val_loss: 9.8215e-04
Epoch 24/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0015 - val_loss: 9.8507e-04
Epoch 25/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0015 - val_loss: 8.6651e-04
Epoch 26/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0014 - val_loss: 9.6216e-04
Epoch 27/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0013 - val_loss: 0.0019
Epoch 28/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0018 - val_loss: 0.0016
Epoch 29/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0016 - val_loss: 7.5479e-04
Epoch 30/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0014 - val_loss: 6.8110e-04
Epoch 31/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0018 - val_loss: 8.9372e-04
Epoch 32/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0014 - val_loss: 0.0016
Epoch 33/50
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0013 - val_loss: 6.4400e-04
Epoch 34/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0014 - val_loss: 9.8018e-04
Epoch 35/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0013 - val_loss: 0.0012
Epoch 36/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0013 - val_loss: 0.0018
Epoch 37/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0014 - val_loss: 0.0018
Epoch 38/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0012 - val_loss: 0.0011
Epoch 39/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0011 - val_loss: 6.5875e-04
Epoch 40/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0013 - val_loss: 0.0014
Epoch 41/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0012 - val_loss: 7.0472e-04
Epoch 42/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0013 - val_loss: 0.0022
Epoch 43/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0014 - val_loss: 0.0017
Epoch 44/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0011 - val_loss: 6.0121e-04
Epoch 45/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0012 - val_loss: 5.5651e-04
Epoch 46/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0016 - val_loss: 0.0012
Epoch 47/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0011 - val_loss: 7.9391e-04
Epoch 48/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0014 - val_loss: 6.3849e-04
Epoch 49/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0012 - val_loss: 0.0015
Epoch 50/50
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0011 - val_loss: 5.8862e-04
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Train RMSE: 3.098855249095924
Test RMSE: 3.032341576124005
LSTM performance evaluation metrics:
Mean Squared Error (MSE): 9.1951
Root Mean Squared Error (RMSE): 3.0323
Mean Absolute Error (MAE): 2.2954
R² Score: 0.9910
```





In [85]:
```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scikeras.wrappers import KerasRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
import matplotlib.pyplot as plt

# 1. Load and preprocess the data
df = Amazon_df.copy()  # Assuming Apple_df is a DataFrame

# 2. Normalize the data (MinMaxScaler)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)

# 3. Prepare the data for LSTM
def create_dataset(data, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), :])  # Use all columns (features)
        y.append(data[i + time_step, 0])  # Predict the 'Adj Close' (first column)
    return np.array(X), np.array(y)

# Define the number of time steps (e.g., use the last 60 days to predict the next v
time_step = 60
X, y = create_dataset(scaled_data, time_step)

# 4. Split the data into training and test sets
train_size = int(len(X) * 0.7)  # 70% training data, 30% testing
X_train_lstm, X_test_lstm = X[:train_size], X[train_size:]
y_train_lstm, y_test_lstm = y[:train_size], y[train_size:]

# 5. Reshape data into 3D (samples, time steps, features) for LSTM
X_train_lstm = X_train_lstm.reshape(X_train_lstm.shape[0], X_train_lstm.shape[1], X
X_test_lstm = X_test_lstm.reshape(X_test_lstm.shape[0], X_test_lstm.shape[1], X_tes

# Function to create the LSTM model
def create_model(units=50, dropout_rate=0.2, optimizer='adam'):
    model = Sequential()
    model.add(LSTM(units, return_sequences=True, input_shape=(time_step, X_train_ls
    model.add(Dropout(dropout_rate))
```

```
    model.add(LSTM(units, return_sequences=False))
    model.add(Dropout(dropout_rate))
    model.add(Dense(25))
    model.add(Dense(1))
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model

# Wrapping the model using SciKeras' KerasRegressor
model = KerasRegressor(model=create_model, verbose=0)

# Define the hyperparameter space
param_dist = {
    'model__units': [50, 100, 150],
    'model__dropout_rate': [0.2, 0.3, 0.4],
    'batch_size': [16, 32, 64],
    'epochs': [50, 100],
    'optimizer': ['adam', 'rmsprop']
}

# Using RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist,

# Fit the model using RandomizedSearchCV
random_search.fit(X_train_lstm, y_train_lstm)

# Get the best parameters
best_params = random_search.best_params_
print(f"Best Parameters: {best_params}")

# Evaluate the best model on the test set
best_model = random_search.best_estimator_
y_test_pred = best_model.predict(X_test_lstm)

# Invert scaling for the actual vs predicted values
y_test_pred_rescaled = scaler.inverse_transform(np.concatenate((y_test_pred.reshape
y_test_lstm_rescaled = scaler.inverse_transform(np.concatenate((y_test_lstm.reshape

# Evaluate the model performance
mse = mean_squared_error(y_test_lstm_rescaled, y_test_pred_rescaled)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_lstm_rescaled, y_test_pred_rescaled)
r2 = r2_score(y_test_lstm_rescaled, y_test_pred_rescaled)

print(f"Test RMSE: {rmse}")
print(f"Test MAE: {mae}")
print(f"Test R²: {r2}")
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.2, model__units=50, opti
mizer=adam; total time= 1.5min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.2, model__units=50, opti
mizer=adam; total time= 1.5min
```

```
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.2, model__units=50, opti
mizer=adam; total time= 1.5min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=100, model__dropout_rate=0.3, model__units=150, opt
imizer=adam; total time= 4.4min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=100, model__dropout_rate=0.3, model__units=150, opt
imizer=adam; total time= 4.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=100, model__dropout_rate=0.3, model__units=150, opt
imizer=adam; total time= 4.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.2, model__units=100, opt
imizer=rmsprop; total time= 2.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.2, model__units=100, opt
imizer=rmsprop; total time= 2.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.2, model__units=100, opt
imizer=rmsprop; total time= 2.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=100, model__dropout_rate=0.3, model__units=150, opt
imizer=rmsprop; total time= 4.3min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
```

```
[CV] END batch_size=16, epochs=100, model__dropout_rate=0.3, model__units=150, opt
imizer=rmsprop; total time= 4.4min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=100, model__dropout_rate=0.3, model__units=150, opt
imizer=rmsprop; total time= 4.0min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=50, model__dropout_rate=0.3, model__units=50, optim
izer=rmsprop; total time= 1.3min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=50, model__dropout_rate=0.3, model__units=50, optim
izer=rmsprop; total time= 1.3min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=50, model__dropout_rate=0.3, model__units=50, optim
izer=rmsprop; total time= 1.0min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=50, model__dropout_rate=0.3, model__units=50, optim
izer=adam; total time= 1.3min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=50, model__dropout_rate=0.3, model__units=50, optim
izer=adam; total time= 1.3min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=16, epochs=50, model__dropout_rate=0.3, model__units=50, optim
izer=adam; total time= 1.0min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__units=100, opti
mizer=adam; total time= 1.2min
```

```
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__units=100, optimizer=adam; total time= 1.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__units=100, optimizer=adam; total time= 1.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__units=50, optimizer=rmsprop; total time=  48.1s
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__units=50, optimizer=rmsprop; total time=  47.7s
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=50, model__dropout_rate=0.2, model__units=50, optimizer=rmsprop; total time=  45.7s
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.2, model__units=50, optimizer=rmsprop; total time= 1.0min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:5 out of the last 24 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x0000023E17788E00> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.2, model__units=50, optimizer=rmsprop; total time=  59.7s
```

```
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
WARNING:tensorflow:5 out of the last 17 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x0000023E1769CEA0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
[CV] END batch_size=64, epochs=100, model__dropout_rate=0.2, model__units=50, optimizer=rmsprop; total time=  58.7s
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.4, model__units=150, optimizer=adam; total time= 3.2min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.4, model__units=150, optimizer=adam; total time= 3.4min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
[CV] END batch_size=32, epochs=100, model__dropout_rate=0.4, model__units=150, optimizer=adam; total time= 3.3min
C:\Users\97152\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Best Parameters: {'optimizer': 'adam', 'model__units': 150, 'model__dropout_rate': 0.3, 'epochs': 100, 'batch_size': 16}
Test RMSE: 3.387405952641971
Test MAE: 2.692634682930532
Test R²: 0.9881546219777338
```

In [ ]: