In [1]:

```
# IMPORT THE LIBRARY
!pip install yfinance
```

...

In [2]:

```
!pip install ipynb
```

```
Requirement already satisfied: ipynb in c:\users\knc01\anaconda3\lib\site
-packages (0.5.1)
```

In [3]:

```
SENTIMENT_ANALYSIS = 1
if SENTIMENT_ANALYSIS:
    !pip install newspaper3k
    !pip install GoogleNews
    !pip install nltk
    !pip install newspaper
    !pip install wordcloud
else:
    print("Sentiment analysis disabled")
```

...

In [4]:

```
from datetime import datetime
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
if SENTIMENT_ANALYSIS:
    import nltk
    from nltk.sentiment.vader import SentimentIntensityAnalyzer
    from GoogleNews import GoogleNews
    from newspaper import Article
    from newspaper import Config
    from wordcloud import WordCloud, STOPWORDS
    import json
    nltk.download('vader_lexicon') #required for Sentiment Analysis
    nltk.download('punkt')
else:
    print("Sentiment analysis disabled")
```

...

In [6]:

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import SparkSession
from pyspark.sql import functions as sqlFn
from pyspark.sql.window import Window
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
#from ipynb.fs.full.SentimentAnalysis import GenerateCSVFile
from SentLib import GenerateCSVFile
```

In [7]:

```
# CREATE TICKER INSTANCE FOR AMAZON
amzn = yf.Ticker("AMZN")

# GET TODAYS DATE AND CONVERT IT TO A STRING WITH YYYY-MM-DD FORMAT (YFINA
end_date = datetime.now().strftime('%Y-%m-%d')
if SENTIMENT_ANALYSIS:
    amzn_hist = amzn.history(start='2023-04-07',end='2024-04-07')
else:
    amzn_hist = amzn.history(start='2010-01-16',end='2024-04-01')
#print(amzn_hist)
amzn_hist.to_csv('amazon.csv')
```

In [8]:

```
#Generate Sentiment Analysis CSV file for Amazon with normalized sentiment
if SENTIMENT_ANALYSIS:
    GenerateCSVFile("AMZN",'sentiment_data.csv',365)
else:
    print("Sentiment analysis disabled")
```

```
Negative Sentiment: 0.00
Positive ......: 0.55

Sentiment analysis for the period: 2023-07-31 to 2024-04-16
Positive Sentiment: 4.00
Neutral Sentiment: 6.00
Negative Sentiment: 0.00
Positive ......: 0.55

Sentiment analysis for the period: 2023-07-30 to 2024-04-16
Positive Sentiment: 4.00
Neutral Sentiment: 6.00
Negative Sentiment: 0.00
Positive ......: 0.55

Sentiment analysis for the period: 2023-07-29 to 2024-04-16
HTTP Error 429: Too Many Requests

----------------------------------------------------------------------
-----
```

In [9]:

```
from pyspark.sql.functions import col, to_date
# Create a Spark session
spark = SparkSession.builder.appName("StockPredictionModel").getOrCreate()
#spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")
if SENTIMENT_ANALYSIS:
    # Load the data from the CSV into a DataFrame
    df1 = spark.read.csv("amazon.csv", header=True, inferSchema=True)
    # Parse the date column to keep only the YY-MM-DD part
    df1 = df1.withColumn("parsed_date", to_date(col("Date"), "yyyy-MM-dd"))
else:
    df = spark.read.csv("amazon.csv", header=True, inferSchema=True)
    df1 = df
    print("Sentiment analysis disabled")
```

In [10]:

```
df1.show(30)
```

```
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+-----------+
|               Date|              Open|              High|
Low|             Close|   Volume|Dividends|Stock Splits|parsed_date|
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+-----------+
|2023-04-10 00:00:00|100.95999908447266|102.19999694824219| 99.5699996948
2422|102.16999816894531| 37261200|      0.0|         0.0| 2023-04-10|
|2023-04-11 00:00:00|100.80000305175781|             101.0| 99.0100021362
3047| 99.91999816894531| 60417800|      0.0|         0.0| 2023-04-11|
|2023-04-12 00:00:00| 100.4000015258789|100.51000213623047| 97.7099990844
7266| 97.83000183105469| 56735000|      0.0|         0.0| 2023-04-12|
|2023-04-13 00:00:00| 98.94999694824219|102.56999969482422| 98.7099990844
7266| 102.4000015258789| 67925100|      0.0|         0.0| 2023-04-13|
|2023-04-14 00:00:00|102.06999969482422|103.19999694824219|101.1100006103
5156|102.51000213623047| 51450500|      0.0|         0.0| 2023-04-14|
|2023-04-17 00:00:00|103.16000366210938| 103.7300033569336|101.5899963378
9062|102.73999786376953| 39919500|      0.0|         0.0| 2023-04-17|
|2023-04-18 00:00:00|103.94999694824219|104.19999694824219| 101.519996643
0664|102.30000305175781| 39790500|      0.0|         0.0| 2023-04-18|
|2023-04-19 00:00:00|101.58000183105469|105.12000274658203|101.3899993896
4844|104.30000305175781| 58398900|      0.0|         0.0| 2023-04-19|
|2023-04-20 00:00:00|103.52999877929688|            105.25|103.2099990844
7266|103.80999755859375| 57696900|      0.0|         0.0| 2023-04-20|
|2023-04-21 00:00:00| 106.0999984741211| 108.1500015258789|105.0800018310
5469|106.95999908447266| 86774200|      0.0|         0.0| 2023-04-21|
|2023-04-24 00:00:00|107.66000366210938| 109.2300033569336|105.0699996948
2422|106.20999908447266| 69575600|      0.0|         0.0| 2023-04-24|
|2023-04-25 00:00:00|104.91000366210938|105.44999694824219|102.4499969482
4219|102.56999969482422| 65026800|      0.0|         0.0| 2023-04-25|
|2023-04-26 00:00:00|105.04000091552734|106.62000274658203| 104.099998474
1211| 104.9800033569336| 73803800|      0.0|         0.0| 2023-04-26|
|2023-04-27 00:00:00|108.16000366210938|110.86000061035156|106.8000030517
5781|109.81999969482422|149961200|      0.0|         0.0| 2023-04-27|
|2023-04-28 00:00:00| 107.7300033569336| 109.4800033569336|104.3300018310
5469|105.44999694824219|130565000|      0.0|         0.0| 2023-04-28|
|2023-05-01 00:00:00|104.94999694824219| 105.2300033569336|101.8199996948
2422|102.05000305175781| 74728100|      0.0|         0.0| 2023-05-01|
|2023-05-02 00:00:00|101.47000122070312| 103.9000015258789| 101.150001525
8789|103.62999725341797| 73469400|      0.0|         0.0| 2023-05-02|
|2023-05-03 00:00:00|103.73999786376953|105.95999908447266|103.2799987792
9688| 103.6500015258789| 65051900|      0.0|         0.0| 2023-05-03|
|2023-05-04 00:00:00|104.04000091552734|105.38999938964844|103.3099975585
9375|             104.0| 45345500|      0.0|         0.0| 2023-05-04|
|2023-05-05 00:00:00| 104.2699966430664|105.76000213623047|103.5500030517
5781|105.66000366210938| 56912900|      0.0|         0.0| 2023-05-05|
|2023-05-08 00:00:00|105.04000091552734| 106.0999984741211|104.6999969482
4219|105.83000183105469| 49430900|      0.0|         0.0| 2023-05-08|
|2023-05-09 00:00:00| 105.4800033569336|106.79000091552734|105.1600036621
0938|106.62000274658203| 44089400|      0.0|         0.0| 2023-05-09|
|2023-05-10 00:00:00| 108.0999984741211|110.66999816894531|108.0500030517
5781|110.19000244140625| 78627600|      0.0|         0.0| 2023-05-10|
|2023-05-11 00:00:00|111.02999877929688|113.27999877929688|110.4899978637
6953|112.18000030517578| 74924800|      0.0|         0.0| 2023-05-11|
|2023-05-12 00:00:00|112.16000366210938|112.63999938964844|109.3199996948
2422|110.26000213623047| 49810100|      0.0|         0.0| 2023-05-12|
|2023-05-15 00:00:00| 111.1500015258789|112.29000091552734|           10
```

```
9.25|111.19999694824219| 53011100|      0.0|         0.0| 2023-05-15|
|2023-05-16 00:00:00|111.05000305175781|114.79000091552734|111.0500030517
5781| 113.4000015258789| 71472900|      0.0|         0.0| 2023-05-16|
|2023-05-17 00:00:00|114.88999938964844|115.83000183105469|114.2200012207
0312|             115.5| 65655200|      0.0|         0.0| 2023-05-17|
|2023-05-18 00:00:00|116.69000244140625| 118.5999984741211|116.3399963378
9062| 118.1500015258789| 73174100|      0.0|         0.0| 2023-05-18|
|2023-05-19 00:00:00|118.16000366210938|118.30999755859375|115.6999969482
4219|            116.25| 54990200|      0.0|         0.0| 2023-05-19|
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+-----------+
only showing top 30 rows
```

In [11]:

```
if SENTIMENT_ANALYSIS:
    df2 = spark.read.csv("sentiment_data.csv", header=True, inferSchema=Tr
    df2.show(30)
    df = df1.join(df2, df1["parsed_date"] == df2["Date"], "inner")
    df.show(8)
    #drop the extra "date"
    df = df.drop(df2["Date"])
    df = df.drop(df1["parsed_date"])
    df.show(8)
else:
    print("Sentiment analysis disabled")
```

```
+----------+------+---------+
|      Date|Ticker|Sentiment|
+----------+------+---------+
|2024-04-15|  AMZN|        0|
|2024-04-14|  AMZN|        0|
|2024-04-13|  AMZN|        0|
|2024-04-12|  AMZN|        0|
|2024-04-11|  AMZN|        0|
|2024-04-10|  AMZN|        0|
|2024-04-09|  AMZN|        0|
|2024-04-08|  AMZN|        0|
|2024-04-07|  AMZN|        0|
|2024-04-06|  AMZN|        0|
|2024-04-05|  AMZN|        0|
|2024-04-04|  AMZN|        0|
|2024-04-03|  AMZN|        0|
|2024-04-02|  AMZN|        0|
|2024-04-01|  AMZN|        0|
|2024-03-31|  AMZN|        0|
|2024-03-30|  AMZN|        0|
|2024-03-29|  AMZN|        0|
|2024-03-28|  AMZN|        0|
|2024-03-27|  AMZN|        0|
|2024-03-26|  AMZN|        0|
|2024-03-25|  AMZN|        0|
|2024-03-24|  AMZN|        0|
|2024-03-23|  AMZN|        0|
|2024-03-22|  AMZN|        0|
|2024-03-21|  AMZN|        0|
|2024-03-20|  AMZN|        0|
|2024-03-19|  AMZN|        0|
|2024-03-18|  AMZN|        0|
|2024-03-17|  AMZN|        0|
+----------+------+---------+
only showing top 30 rows

+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+-----------+----
------+------+---------+
|               Date|              Open|              High|
Low|             Close|   Volume|Dividends|Stock Splits|parsed_date|
Date|Ticker|Sentiment|
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+-----------+----
------+------+---------+
|2023-07-31 00:00:00| 133.1999969482422| 133.8699951171875| 132.380004882
8125|133.67999267578125| 41901500|      0.0|         0.0| 2023-07-31|2023
-07-31|  AMZN|        0|
|2023-08-01 00:00:00| 133.5500030517578|133.69000244140625| 131.619995117
1875|131.69000244140625| 42098500|      0.0|         0.0| 2023-08-01|2023
-08-01|  AMZN|        0|
|2023-08-02 00:00:00|130.14999389648438|130.22999572753906|126.8199996948
2422| 128.2100067138672| 51027600|      0.0|         0.0| 2023-08-02|2023
-08-02|  AMZN|        0|
|2023-08-03 00:00:00| 127.4800033569336|129.83999633789062|126.4100036621
0938|128.91000366210938| 88585200|      0.0|         0.0| 2023-08-03|2023
-08-03|  AMZN|        0|
|2023-08-04 00:00:00|141.05999755859375| 143.6300048828125|139.3200073242
1875|139.57000732421875|152938700|      0.0|         0.0| 2023-08-04|2023
-08-04|  AMZN|        0|
|2023-08-07 00:00:00|140.99000549316406| 142.5399932861328| 138.949996948
2422|142.22000122070312| 71213100|      0.0|         0.0| 2023-08-07|2023
-08-07|  AMZN|        0|
|2023-08-08 00:00:00| 140.6199951171875|140.83999633789062| 138.419998168
9453|139.94000244140625| 51710500|      0.0|         0.0| 2023-08-08|2023
-08-08|  AMZN|        0|
|2023-08-09 00:00:00|139.97000122070312|140.32000732421875|137.1000061035
1562|137.85000610351562| 50017300|      0.0|         0.0| 2023-08-09|2023
-08-09|  AMZN|        0|
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+-----------+----
------+------+---------+
only showing top 8 rows

+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+------+---------
+
|               Date|              Open|              High|
Low|             Close|   Volume|Dividends|Stock Splits|Ticker|Sentiment|
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+------+---------
+
|2023-07-31 00:00:00| 133.1999969482422| 133.8699951171875| 132.380004882
8125|133.67999267578125| 41901500|      0.0|         0.0|  AMZN|        0
|
|2023-08-01 00:00:00| 133.5500030517578|133.69000244140625| 131.619995117
1875|131.69000244140625| 42098500|      0.0|         0.0|  AMZN|        0
|
|2023-08-02 00:00:00|130.14999389648438|130.22999572753906|126.8199996948
2422| 128.2100067138672| 51027600|      0.0|         0.0|  AMZN|        0
|
|2023-08-03 00:00:00| 127.4800033569336|129.83999633789062|126.4100036621
0938|128.91000366210938| 88585200|      0.0|         0.0|  AMZN|        0
|
|2023-08-04 00:00:00|141.05999755859375| 143.6300048828125|139.3200073242
1875|139.57000732421875|152938700|      0.0|         0.0|  AMZN|        0
|
|2023-08-07 00:00:00|140.99000549316406| 142.5399932861328| 138.949996948
2422|142.22000122070312| 71213100|      0.0|         0.0|  AMZN|        0
|
|2023-08-08 00:00:00| 140.6199951171875|140.83999633789062| 138.419998168
9453|139.94000244140625| 51710500|      0.0|         0.0|  AMZN|        0
|
|2023-08-09 00:00:00|139.97000122070312|140.32000732421875|137.1000061035
1562|137.85000610351562| 50017300|      0.0|         0.0|  AMZN|        0
|
+-------------------+------------------+------------------+--------------
----+------------------+---------+---------+------------+------+---------
+
only showing top 8 rows
```

In [12]:

```
# drop any row having any Null
df = df.dropna(how="any")
```

In [13]:

```
# openCloseChange
df = df.withColumn("openCloseChange", (df.Close - df.Open) / df.Open)
```

In [14]:

```
# maxDayChange
df = df.withColumn("maxDayChange", df.High - df.Low)
```

In [15]:

```
# dividend provided
df = df.withColumn("dividend", sqlFn.when(df["Dividends"] > 0, 1).otherwis
```

In [16]:

```
# Stock split
df = df.withColumn("stockSplit", sqlFn.when(df["Stock Splits"] != 1, 1).ot
```

In [17]:

```
# order by date
w = Window.partitionBy().orderBy("date")
```

In [18]:

```
# Lagged column for the 'close' price (i.e., previous day's close)
df = df.withColumn("lagClose", sqlFn.lag(df.Close).over(w))
```

In [19]:

```
#  DailyChange - change in closing price from the previous day
df = df.withColumn("DailyChange", df.Close - df.lagClose)
```

In [20]:

```
# moving average for the closing prices
df = df.withColumn("movingAvgClose", sqlFn.avg(df.Close).over(w.rowsBetwee
```

In [21]:

```
# drop any row having any Null
df = df.dropna(how="any")
```

In [22]:

```
if SENTIMENT_ANALYSIS:
    consolidatedFeature = ["Open", "High", "Low", "Close", "Volume", "open
                           "maxDayChange", "DailyChange", "movingAvgClose",
                           "dividend", "stockSplit", "Sentiment"]
else:
    consolidatedFeature = ["Open", "High", "Low", "Close", "Volume", "open
                           "maxDayChange", "DailyChange", "movingAvgClose",
                           "dividend", "stockSplit"]
    print("Sentiment analysis disabled")
```

In [23]:

```
#store features in the vector column
assembler = VectorAssembler(inputCols=consolidatedFeature, outputCol="feat
df_assembled = assembler.transform(df)
```

In [24]:

```
# Split the data into a training set - 80% , 20% test set.

trainingDataCount = int(df_assembled.count() * 0.8)
trainingData = df_assembled.orderBy("date").limit(trainingDataCount)
testData = df_assembled.subtract(trainingData)
```

In [25]:

```
# GBT Model Training
gbt = GBTRegressor(labelCol="Close", featuresCol="features", maxIter=10, m
```

In [26]:

```
model = gbt.fit(trainingData)
```

In [27]:

```
predictions = model.transform(testData)
```

In [28]:

```
# Model Evaluation
# Compute the RMSE (Root Mean Squared Error) for the predictions
evaluator_rmse = RegressionEvaluator(labelCol="Close", predictionCol="pred
```

In [29]:

```
rmse = evaluator_rmse.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data =", rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 5.958398749994316
```

In [30]:

```
# Mean Absolute Error (MAE) and R-squared (R2)
for metric in ["mae", "r2"]:
    evaluator = RegressionEvaluator(labelCol="Close", predictionCol="predi
    value = evaluator.evaluate(predictions)
    print(f"{metric.upper()}: {value}")
```

```
MAE: 5.0383267841752435
R2: -1.3162013670172636
```

In [31]:

```
plt.figure(figsize=(12, 6))
```

Out[31]:

```
<Figure size 1200x600 with 0 Axes>

<Figure size 1200x600 with 0 Axes>
```

In [32]:

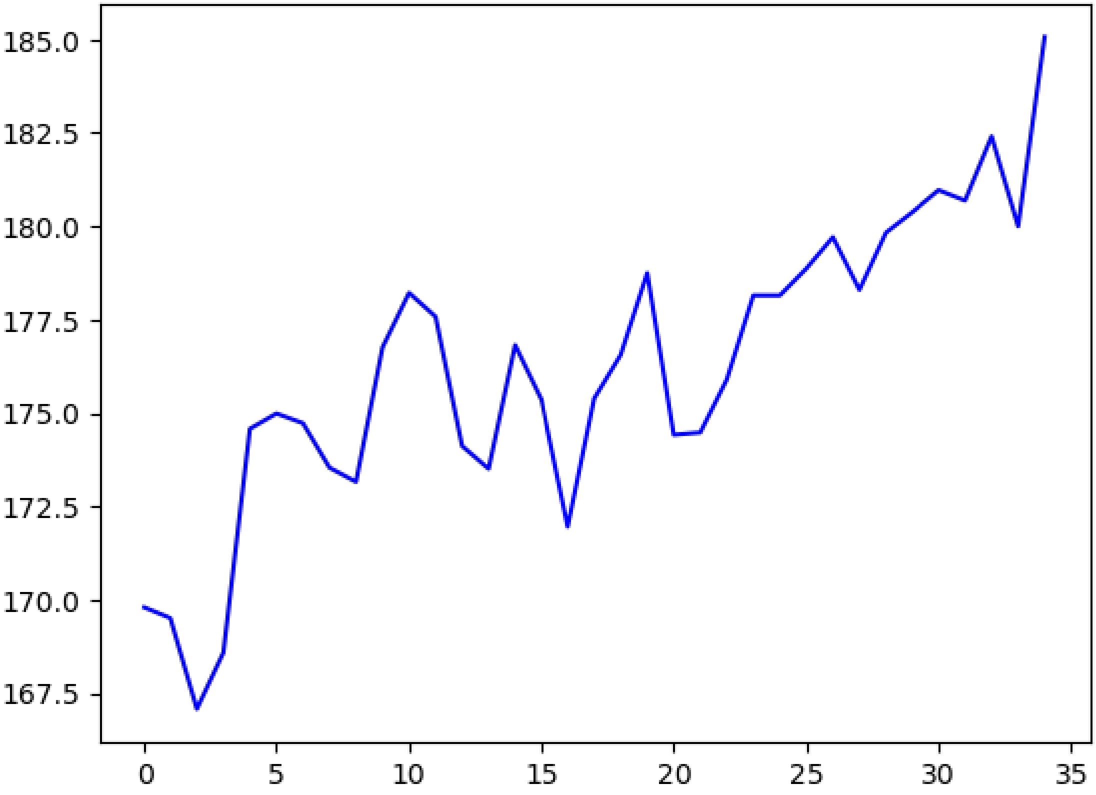
```
preds = predictions.select("Date", "Close", "prediction").toPandas()
```

In [33]:

```
plt.plot(preds["Close"], label='Actual', color='blue')
```

Out[33]:

```
[<matplotlib.lines.Line2D at 0x1e933678390>]
```

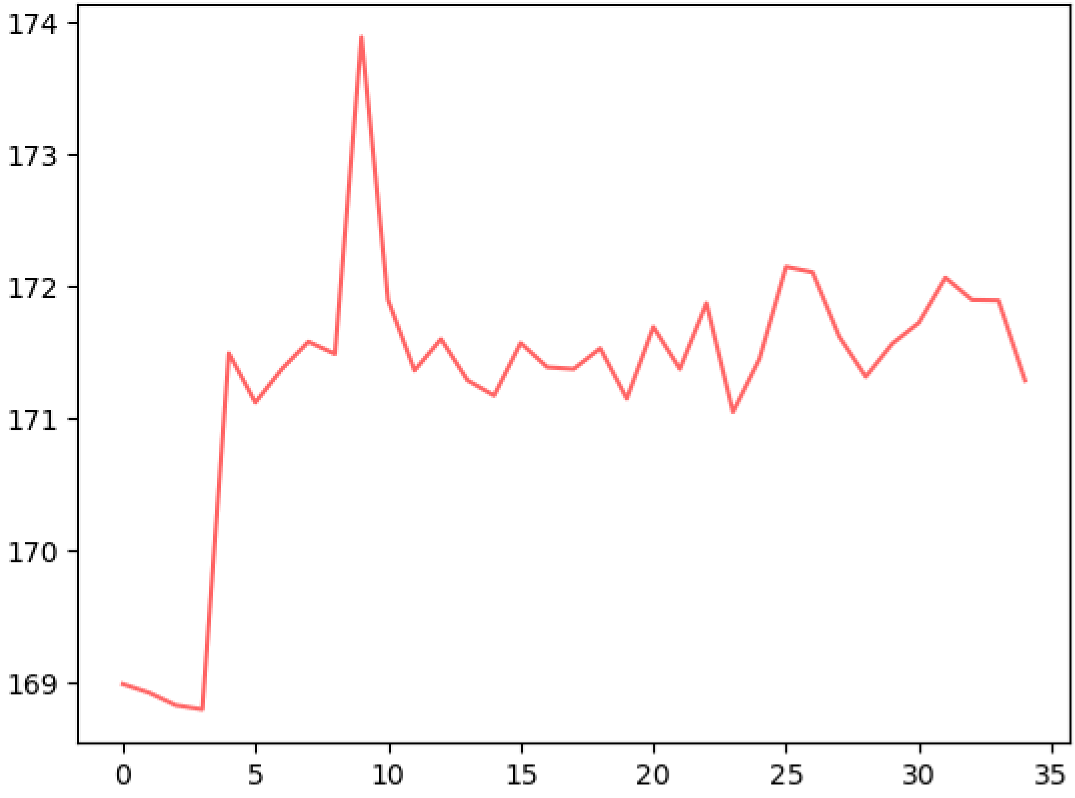


In [34]:

```
plt.plot(preds["prediction"], label='Predicted', color='red', alpha=0.6)
```

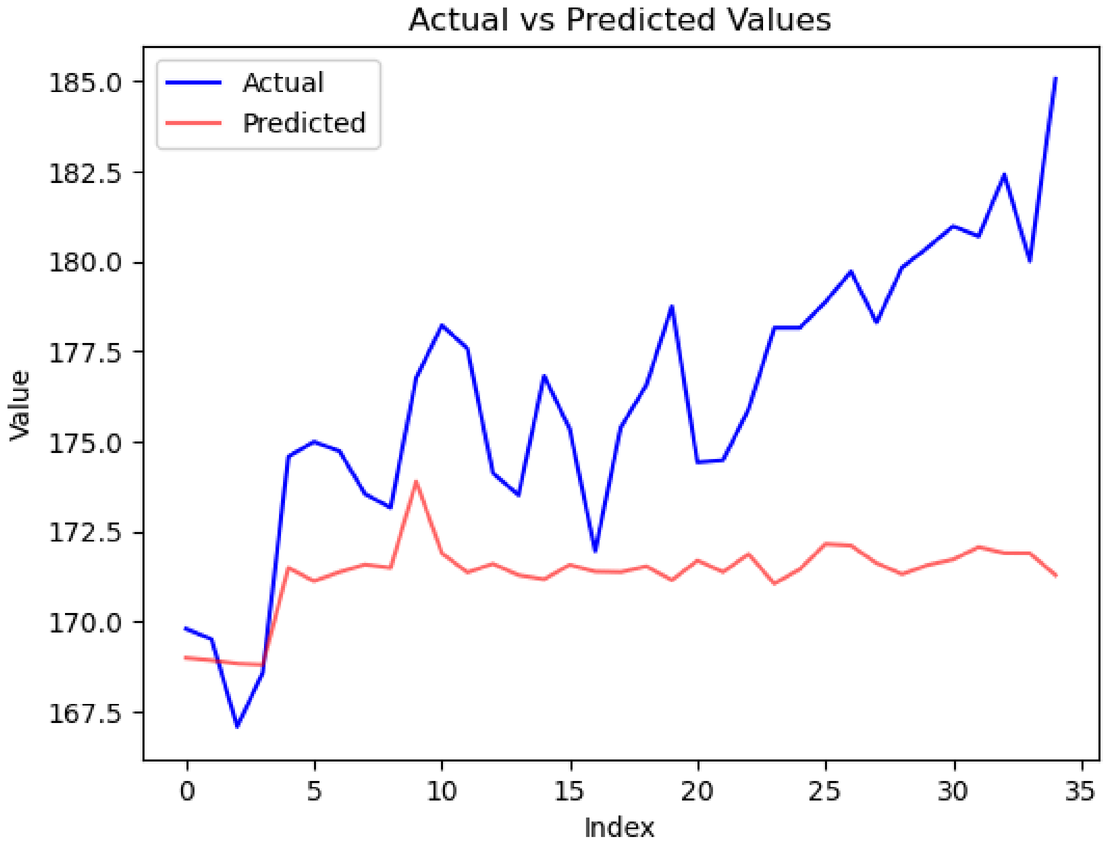
Out[34]:

```
[<matplotlib.lines.Line2D at 0x1e933e1a410>]
```



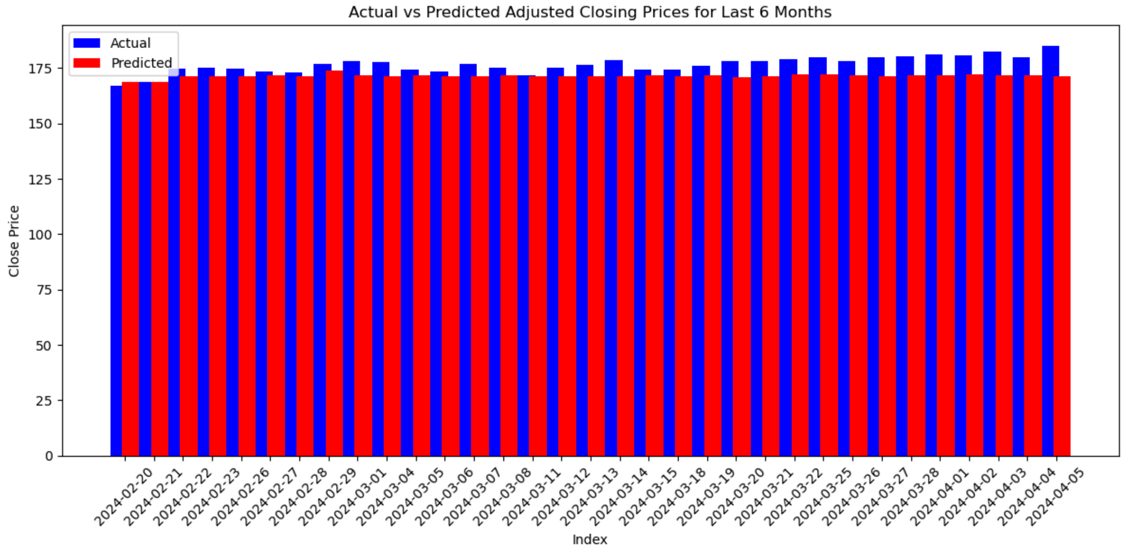
In [36]:

```
plt.plot(preds.index, preds["Close"], label='Actual', color='blue')
plt.plot(preds.index, preds["prediction"], label='Predicted', color='red',
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```



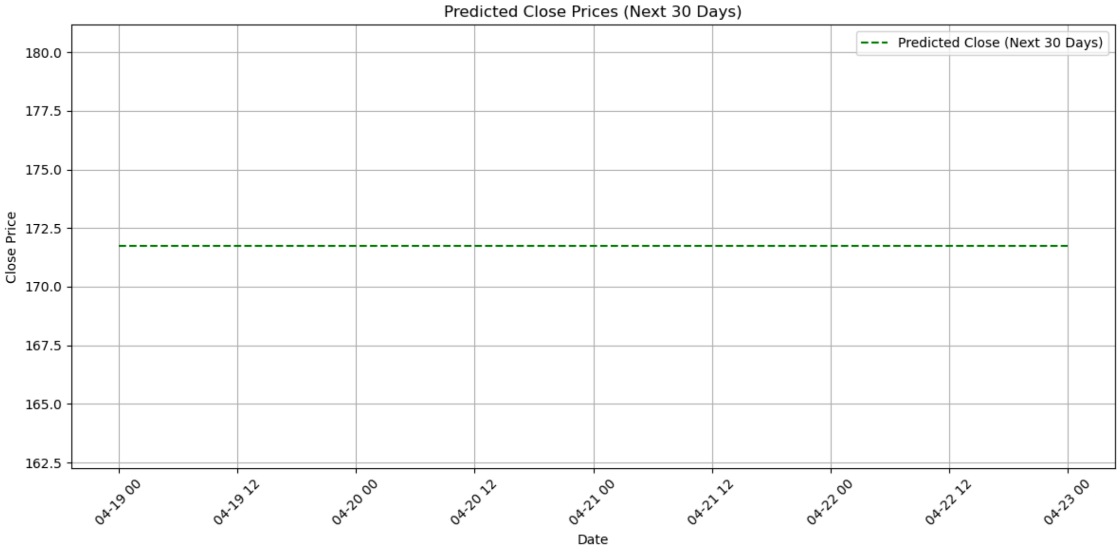
In [37]:

```
import pandas as pd
from datetime import timedelta
import matplotlib.pyplot as plt

# Convert "Date" column to datetime type, with utc=True
preds['Date'] = pd.to_datetime(preds['Date'], utc=True)

# Filter data for the Last 2 months
six_months_ago = pd.Timestamp.now(tz='UTC') - timedelta(days=30*2)
preds_last_6_months = preds[preds['Date'] >= six_months_ago]

# Creating the plot
plt.figure(figsize=(12, 6))  # Setting the figure size

# Plotting actual values as blue bars
plt.bar(preds_last_6_months.index, preds_last_6_months["Close"], color='bl

# Shifting the position of predicted values slightly to the right for bett
plt.bar(preds_last_6_months.index + 0.4, preds_last_6_months["prediction"]

plt.xlabel('Index')  # Labeling x-axis
plt.ylabel('Close Price')  # Labeling y-axis
plt.title('Actual vs Predicted Adjusted Closing Prices for Last 6 Months')
plt.legend()  # Showing the legend
plt.xticks(preds_last_6_months.index + 0.2, preds_last_6_months['Date'].dt
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjusting layout for better visualization
plt.show()  # Displaying the plot
```



In [38]:

```
from matplotlib.dates import AutoDateLocator, AutoDateFormatter

# Prepare historical data for the past 1 month
end_date_past = datetime.now().strftime('%Y-%m-%d')
start_date_past = (datetime.now() - timedelta(days=30)).strftime('%Y-%m-%d
past_data = df.filter((sqlFn.col("Date") >= start_date_past) & (sqlFn.col(

# Apply the same feature engineering steps to past_data
# Assuming you've defined the feature_columns as before
consolidatedFeature = ["Open", "High", "Low", "Close", "Volume", "openClos
                       "maxDayChange", "DailyChange", "movingAvgClose",
                       "dividend", "stockSplit"]

# Assemble features
assembler = VectorAssembler(inputCols=consolidatedFeature, outputCol="feat
past_data_assembled = assembler.transform(past_data)

# Apply the trained model to make predictions for the past 1 month
past_predictions = model.transform(past_data_assembled)

# Plot the historical and predicted data for the past 1 month
past_data_pd = past_data.select("Date", "Close").toPandas()
past_pred_pd = past_predictions.select("Date", "prediction").toPandas()

last_date = datetime.strptime(end_date_past, '%Y-%m-%d')

# Prepare future data for the next 7 days using the last available data po
end_date_future = (datetime.now() + timedelta(days=30)).strftime('%Y-%m-%d
future_dates = [last_date + timedelta(days=i) for i in range(1, 7)]  # Inc
future_df = spark.createDataFrame([(d,) for d in future_dates], ["Date"])
last_data_point = df.orderBy("Date", ascending=False).limit(1)  # Get the
future_df = future_df.crossJoin(last_data_point.drop("Date"))

# Apply the same feature engineering steps to future_df
future_df = future_df.withColumn("lagClose", sqlFn.lag(future_df.Close).ov
future_df = future_df.withColumn("dayChange", (future_df.Close - future_df
future_df = future_df.withColumn("maxDayChange", future_df.High - future_d
future_df = future_df.withColumn("DailyChange", future_df.Close - future_d
future_df = future_df.withColumn("movingAvgClose", sqlFn.avg(future_df.Clo
future_df = future_df.withColumn("dividend", sqlFn.when(future_df["Dividen
future_df = future_df.withColumn("stockSplit", sqlFn.when(future_df["Stock
future_df = future_df.dropna()
future_df_assembled = assembler.transform(future_df)

# Apply the trained model to make predictions for the next 7 days
future_predictions = model.transform(future_df_assembled)

# Plot the predicted data for the next 7 days
future_pred_pd = future_predictions.select("Date", "prediction").toPandas(

# Convert date column to pandas datetime object
future_pred_pd["Date"] = pd.to_datetime(future_pred_pd["Date"])

# Plot the predicted data for the next 7 days
plt.figure(figsize=(12, 6))
plt.plot(future_pred_pd["Date"], future_pred_pd["prediction"], label='Pred
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Predicted Close Prices (Next 30 Days)')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [432]:

```
# Terminate the Spark session
spark.stop()
```