```
import pandas as pd
import yfinance as yf
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import re
import string
import spacy
import numpy as np
#nltk.download('vader_lexicon')
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob
from sklearn.decomposition import LatentDirichletAllocation
from ta.momentum import RSIIndicator

import requests
def extract_data(year, month, api_key):
    url = f"https://api.nytimes.com/svc/archive/v1/{year}/{month}.json?api-key={api_key}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        api_data = data['response']['docs']
        return api_data
    else:
        print(f"Failed to fetch data for {year}-{month}: {response.status_code}")
        return []
```

```
all_headlines = []
all_datetime=[]
api_key="of9VIAr7muye3G5DR1GxbR71TPCnFHIy"
for year in range(2010,2020):
    for month in range(1,13):
        print(f"Fetching data for {year}-{month}")
        try:
            api_data = extract_data(year, month, api_key)
            if api_data:
                all_headlines.extend([data['headline']['main'] for data in api_data])
                all_datetime.extend([data['pub_date'] for data in api_data])
        except Exception as e:
            print(f"Error in extacting data for {year}-{month}: {e}")
```

```
Fetching data for 2019-9
Fetching data for 2019-9
Failed to fetch data for 2019-9: 429
Fetching data for 2019-10
Failed to fetch data for 2019-10: 429
Fetching data for 2019-11
Failed to fetch data for 2019-11: 429
Fetching data for 2019-12
Failed to fetch data for 2019-12: 429
```

```
df= pd.DataFrame({"Datetime": all_datetime, "Headlines":all_headlines})
```

```
df['Datetime']= pd.to_datetime(df['Datetime'])
df['date'] = df['Datetime'].dt.date
```

```
df.head
```

```
pandas.core.generic.NDFrame.head
def head(n: int=5) -> NDFrameT

Return the first `n` rows.

This function returns the first `n` rows for the object based
on position. It is useful for quickly testing if your object
has the right type of data in it.
```

```
def contains_keywords(text):
    for keyword in keywords:
        if keyword.lower() in text.lower():
            return True
    return False
keywords =['data breach', 'user privacy' , 'Cambridge Analytica', 'antitrust' ,'FTC investigation' ,'Congress', 'GDPR', 'quarterly earnings', 'profit', 'Revenue','Mark Zuckerberg', 'cybersecurity', 'hacking', 'a
    'Facebook', 'Meta', 'Whatsapp', 'Instagram',
    'MetaVerse', 'innovation', 'techonology']
df['contains_keywords'] = df['Headlines'].apply(contains_keywords)
```

```
Start coding or generate with AI.
```

```
filtered_df = df[df['contains_keywords']]
filtered_df = filtered_df.drop(columns=['contains_keywords', "Datetime"]).reset_index(drop = True )
filtered_df = filtered_df[['date', 'Headlines']]
filtered_df.head()
```

	date	Headlines
0	2010-01-04	F.C.C. Chairman Spams Facebook Friends
1	2010-01-04	Congress Examines N.F.L. Concussions
2	2010-01-07	Viewing Facebook via Roku
3	2010-01-07	Antitrust Case Has Implications Far Beyond N.F.L.
4	2010-01-07	Chief Says G.M. Is on Road to Profits

```
facebook_news1=filtered_df.copy()
```

```
facebook_news1
```

	date	Headlines
0	2010-01-04	F.C.C. Chairman Spams Facebook Friends
1	2010-01-04	Congress Examines N.F.L. Concussions
2	2010-01-07	Viewing Facebook via Roku
3	2010-01-07	Antitrust Case Has Implications Far Beyond N.F.L.
4	2010-01-07	Chief Says G.M. Is on Road to Profits
...	...	...
758	2016-11-29	Traders Bet on Big Stimulus Spending. Congress...
759	2016-11-29	Daily Report: Facebook Spends a Month Behind t...
760	2016-11-30	‘Instagram Face’: Is It the End of Good Makeup?
761	2016-11-30	Daily Report: Twitter Struggles to Turn Promin...
762	2016-11-30	Jackie Kennedy: The First Instagram First Lady

763 rows × 2 columns

```
all_headlines = []
all_datetime=[]
api_key="of9VIAr7muye3G5DR1GxbR71TPCnFHIy"
```

```
for year in range(2020,2025):
    for month in range(1,13):
        print(f"Fetching data for {year}-{month}")
        try:
            api_data = extract_data(year, month, api_key)
            if api_data:
                all_headlines.extend([data['headline']['main'] for data in api_data])
                all_datetime.extend([data['pub_date'] for data in api_data])
        except Exception as e:
            print(f"Error in extacting data for {year}-{month}: {e}")
```

```
Fetching data for 2020-1
Failed to fetch data for 2020-1: 429
Fetching data for 2020-2
Failed to fetch data for 2020-2: 429
Fetching data for 2020-3
Failed to fetch data for 2020-3: 429
Fetching data for 2020-4
Failed to fetch data for 2020-4: 429
Fetching data for 2020-5
Failed to fetch data for 2020-5: 429
Fetching data for 2020-6
Failed to fetch data for 2020-6: 429
Fetching data for 2020-7
Failed to fetch data for 2020-7: 429
Fetching data for 2020-8
Failed to fetch data for 2020-8: 429
Fetching data for 2020-9
Failed to fetch data for 2020-9: 429
Fetching data for 2020-10
Failed to fetch data for 2020-10: 429
Fetching data for 2020-11
Failed to fetch data for 2020-11: 429
Fetching data for 2020-12
Failed to fetch data for 2020-12: 429
Fetching data for 2021-1
Failed to fetch data for 2021-1: 429
Fetching data for 2021-2
Failed to fetch data for 2021-2: 429
Fetching data for 2021-3
Failed to fetch data for 2021-3: 429
Fetching data for 2021-4
Failed to fetch data for 2021-4: 429
Fetching data for 2021-5
Failed to fetch data for 2021-5: 429
Fetching data for 2021-6
Failed to fetch data for 2021-6: 429
Fetching data for 2021-7
Failed to fetch data for 2021-7: 429
Fetching data for 2021-8
Failed to fetch data for 2021-8: 429
Fetching data for 2021-9
Failed to fetch data for 2021-9: 429
Fetching data for 2021-10
Failed to fetch data for 2021-10: 429
Fetching data for 2021-11
Failed to fetch data for 2021-11: 429
Fetching data for 2021-12
Failed to fetch data for 2021-12: 429
Fetching data for 2022-1
Failed to fetch data for 2022-1: 429
Fetching data for 2022-2
Failed to fetch data for 2022-2: 429
Fetching data for 2022-3
Failed to fetch data for 2022-3: 429
Fetching data for 2022-4
Failed to fetch data for 2022-4: 429
Fetching data for 2022-5
Failed to fetch data for 2022-5: 429
```

```
df= pd.DataFrame({"Datetime": all_datetime, "Headlines":all_headlines})
df['Datetime']= pd.to_datetime(df['Datetime'])
df['date'] = df['Datetime'].dt.date
df['contains_keywords'] = df['Headlines'].apply(contains_keywords)
filtered_df = df[df['contains_keywords']]
filtered_df = filtered_df.drop(columns=['contains_keywords', "Datetime"]).reset_index(drop = True )
filtered_df = filtered_df[['date', 'Headlines']]
filtered_df.head()
```

	date	Headlines
0	2023-01-01	Retiring Congress Members See Rough Roads Ahea...
1	2023-01-03	A Con Man Is Succeeding Me in Congress Today
2	2023-01-04	Meta’s Ad Practices Ruled Illegal Under E.U. Law
3	2023-01-04	What the Far-Right Republicans Want: To Remake...
4	2023-01-05	Where Are the Most Profitable Winter Vacation ...

```
facebook_news2=filtered_df.copy()
```

```
all_headlines = []
all_datetime=[]
api_key="of9VIAr7muye3G5DR1GxbR71TPCnFHIy"
for year in range(2010,2014):
    for month in range(1,13):
        print(f"Fetching data for {year}-{month}")
        try:
            api_data = extract_data(year, month, api_key)
            if api_data:
                all_headlines.extend([data['headline']['main'] for data in api_data])
                all_datetime.extend([data['pub_date'] for data in api_data])
        except Exception as e:
            print(f"Error in extacting data for {year}-{month}: {e}")
```

```
Fetching data for 2010-1
Failed to fetch data for 2010-1: 429
Fetching data for 2010-2
Failed to fetch data for 2010-2: 429
Fetching data for 2010-3
Failed to fetch data for 2010-3: 429
Fetching data for 2010-4
Failed to fetch data for 2010-4: 429
Fetching data for 2010-5
Failed to fetch data for 2010-5: 429
Fetching data for 2010-6
Failed to fetch data for 2010-6: 429
Fetching data for 2010-7
Failed to fetch data for 2010-7: 429
Fetching data for 2010-8
Failed to fetch data for 2010-8: 429
Fetching data for 2010-9
Failed to fetch data for 2010-9: 429
Fetching data for 2010-10
Failed to fetch data for 2010-10: 429
Fetching data for 2010-11
Failed to fetch data for 2010-11: 429
Fetching data for 2010-12
Failed to fetch data for 2010-12: 429
Fetching data for 2011-1
Failed to fetch data for 2011-1: 429
Fetching data for 2011-2
Failed to fetch data for 2011-2: 429
Fetching data for 2011-3
Failed to fetch data for 2011-3: 429
Fetching data for 2011-4
Failed to fetch data for 2011-4: 429
Fetching data for 2011-5
Failed to fetch data for 2011-5: 429
Fetching data for 2011-6
Failed to fetch data for 2011-6: 429
Fetching data for 2011-7
Failed to fetch data for 2011-7: 429
Fetching data for 2011-8
Failed to fetch data for 2011-8: 429
Fetching data for 2011-9
Failed to fetch data for 2011-9: 429
Fetching data for 2011-10
Failed to fetch data for 2011-10: 429
Fetching data for 2011-11
Failed to fetch data for 2011-11: 429
Fetching data for 2011-12
Failed to fetch data for 2011-12: 429
Fetching data for 2012-1
Failed to fetch data for 2012-1: 429
Fetching data for 2012-2
Failed to fetch data for 2012-2: 429
Fetching data for 2012-3
Failed to fetch data for 2012-3: 429
Fetching data for 2012-4
Failed to fetch data for 2012-4: 429
Fetching data for 2012-5
Failed to fetch data for 2012-5: 429
```

```
df= pd.DataFrame({"Datetime": all_datetime, "Headlines":all_headlines})
df['Datetime']= pd.to_datetime(df['Datetime'])
df['date'] = df['Datetime'].dt.date
df['contains_keywords'] = df['Headlines'].apply(contains_keywords)
filtered_df = df[df['contains_keywords']]
filtered_df = filtered_df.drop(columns=['contains_keywords', "Datetime"]).reset_index(drop = True )
filtered_df = filtered_df[['date', 'Headlines']]
filtered_df.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-136-1db311357a82> in <cell line: 6>()
      4 df['contains_keywords'] = df['Headlines'].apply(contains_keywords)
      5 filtered_df = df[df['contains_keywords']]
----> 6 filtered_df = filtered_df.drop(columns=['contains_keywords', "Datetime"]).reset_index(drop = True )
      7 filtered_df = filtered_df[['date', 'Headlines']]
      8 filtered_df.head()

                                  3 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   6697         if mask.any():
   6698             if errors != "ignore":
-> 6699                 raise KeyError(f"{list(labels[mask])} not found in axis")
   6700             indexer = indexer[~mask]
   6701         return self.delete(indexer)

KeyError: "['contains_keywords', 'Datetime'] not found in axis"
```

```
facebook_news3=filtered_df.copy()
```

```
df=[facebook_news3,facebook_news1,facebook_news2]
facebook_news_final = pd.concat(df).reset_index(drop = True )
facebook_news_final.to_csv("facebook_news.csv")
```

```
facebook_news_final
```

	date	Headlines
0	2010-01-04	F.C.C. Chairman Spams Facebook Friends
1	2010-01-04	Congress Examines N.F.L. Concussions
2	2010-01-07	Viewing Facebook via Roku
3	2010-01-07	Antitrust Case Has Implications Far Beyond N.F.L.
4	2010-01-07	Chief Says G.M. Is on Road to Profits
...	...	...
929	2023-05-26	Tom Sawyer, Congressman Who Challenged Census ...
930	2023-05-28	Ian Hacking, Eminent Philosopher of Science an...
931	2023-05-29	The Supreme Court Is Crippling Environmental P...
932	2023-05-30	Companies Push Prices Higher, Protecting Profi...
933	2023-05-31	Why Nonprofits Are Moving to Evict Hundreds of...

934 rows × 2 columns

```
df=facebook_news_final
combined_news_df = df.groupby('date')['Headlines'].agg(' ; '.join).reset_index()
```

```
combined_news_df.duplicated().sum()
```

```
0
```

```
combined_news_df["Date"]=combined_news_df["date"]
combined_news_df['Date'] = pd.to_datetime(combined_news_df['Date'])
combined_news_df.drop(columns="date")
```

	Headlines	Date
0	F.C.C. Chairman Spams Facebook Friends ; Congr...	2010-01-04
1	Viewing Facebook via Roku ; Antitrust Case Has...	2010-01-07
2	In Colorado, Craving Reform of Health Care and...	2010-01-11
3	Cadbury Reports Revenue, and Resistance, Are U...	2010-01-12
4	Football and Antitrust ; Facebook Joins With M...	2010-01-13
...	...	...
345	Tom Sawyer, Congressman Who Challenged Census ...	2023-05-26
346	Ian Hacking, Eminent Philosopher of Science an...	2023-05-28
347	The Supreme Court Is Crippling Environmental P...	2023-05-29
348	Companies Push Prices Higher, Protecting Profi...	2023-05-30
349	Why Nonprofits Are Moving to Evict Hundreds of...	2023-05-31

350 rows × 2 columns

```
!pip install yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.40)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.0.3)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.25.2)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.9.4)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.2.2)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2023.4)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.4.4)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.5)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2.8
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.0.7
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2024.6
```

```
Start coding or generate with AI.
```

```
ticker_symbol = 'TSLA'
start_date = '2010-01-04'
end_date = '2024-4-24'
stock_data = yf.download(ticker_symbol, start=start_date, end=end_date)

stock_data.head()
```

```
[*********************100%%**********************]  1 of 1 completed
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2010-06-29	1.266667	1.666667	1.169333	1.592667	1.592667	281494500
2010-06-30	1.719333	2.028000	1.553333	1.588667	1.588667	257806500
2010-07-01	1.666667	1.728000	1.351333	1.464000	1.464000	123282000
2010-07-02	1.533333	1.540000	1.247333	1.280000	1.280000	77097000
2010-07-06	1.333333	1.333333	1.055333	1.074000	1.074000	103003500

```
merged_df = pd.merge(combined_news_df, stock_data, on='Date', how='inner')
```

```
final_df=merged_df.copy()
```

```
df = final_df
df
```

	date	Headlines	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-07-01	Instagram Remembers Bill Cunningham	2016-07-01	13.742667	14.549333	13.733333	14.433333	14.433333	81000000
1	2016-07-05	Twitter Brings Aboard Facebook Veteran Bret Ta...	2016-07-05	13.982000	14.302667	13.866667	14.265333	14.265333	77629500
2	2016-07-06	Congress Splits Over Bill Aimed at Nation’s Op...	2016-07-06	14.000000	14.348667	13.933333	14.296000	14.296000	73798500
3	2016-07-07	F.B.I. Chief to Explain Recommendation on Hill...	2016-07-07	14.206667	14.541333	14.200667	14.396000	14.396000	54180000
4	2016-07-08	Congressional Study Faults Highway Agency Over...	2016-07-08	14.520000	14.654000	14.300000	14.452000	14.452000	61122000
...	...	...	...	...	...	...	...	...	...
176	2023-05-24	Metalheads Take on the World in John Wray’s Ne...	2023-05-24	182.229996	184.220001	178.220001	182.899994	182.899994	137605100
177	2023-05-25	Where Are the Most Profitable Beach Houses?	2023-05-25	186.539993	186.779999	180.580002	184.470001	184.470001	96870700

```
final_df.to_csv("final dataset.csv")
```

```
df=pd.read_csv("final dataset.csv")
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

#uncomment if not already downloaded
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')
#!python -m spacy download en_core_web_sm


stop_words = set(stopwords.words('english'))
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [token for token in tokens if token not in stop_words]
    doc = nlp(" ".join(tokens))
    lemmatized_tokens = [token.lemma_ for token in doc ]
    preprocessed_text = ' '.join(lemmatized_tokens)
    return preprocessed_text
df['Processed Headlines'] = df['Headlines'].apply(preprocess_text)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
df.shape
```

```
(181, 11)
```

```
import nltk
nltk.download('vader_lexicon')

df['Date'] = pd.to_datetime(df['Date'])
df['SMA5'] = df['Close'].rolling(window=5).mean()
df['SMA2'] = df['Close'].rolling(window=2).mean()
df['close_diff'] = df['Close'].diff()

# Calculate RSI
rsi = RSIIndicator(close=df['Close'], window=14)
df['rsi'] = rsi.rsi()

# Function to get sentiment label based on close difference
def get_sentiment_label(diff):
    if diff > 0:
        return 'Positive'
    elif diff < 0:
        return 'Negative'
    else:
        return 'Neutral'

df['Movement'] = df['close_diff'].apply(lambda x: get_sentiment_label(x) if pd.notnull(x) else 'Neutral')
sid = SentimentIntensityAnalyzer()
df['sentiment_scores'] = df['Headlines'].apply(lambda x: sid.polarity_scores(x)['compound'])
df['neg'] = df['Headlines'].apply(lambda x: sid.polarity_scores(x)['neg'])
df['pos'] = df['Headlines'].apply(lambda x: sid.polarity_scores(x)['pos'])
df['neu'] = df['Headlines'].apply(lambda x: sid.polarity_scores(x)['compound'])
```

```
df['Subjectivity'] = df['Headlines'].apply(lambda x:TextBlob(x).sentiment.subjectivity)
df['Polarity'] =df['Headlines'].apply(lambda x:TextBlob(x).sentiment.polarity)
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
df_cleaned = df.dropna()
```

```
df=df_cleaned.reset_index(drop=True)
df.head()
```

	Unnamed: 0	date	Headlines	Date	Open	High	Low	Close	Adj Close	Volume	...	SMA2	close_diff	
0	13	2016-07-22	Republicans and Democrats in Congress Speak in...	2016-07-22	14.799333	14.966667	14.592000	14.818000	14.818000	38695500	...	14.759000	0.118000	55.5
1	14	2016-07-25	How Sponsored Content Is Becoming King in a Fa...	2016-07-25	14.818000	15.426000	14.758000	15.334000	15.334000	67360500	...	15.076000	0.516000	66.5
		2016	Yahoo, a Web Pioneer	2016										

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
df
```

	Unnamed: 0	date	Headlines	Date	Open	High	Low	Close	Adj Close	Volume	...	SMA2	close_
0	13	2016-07-22	Republicans and Democrats in Congress Speak in...	2016-07-22	14.799333	14.966667	14.592000	14.818000	14.818000	38695500	...	14.759000	0.1
1	14	2016-07-25	How Sponsored Content Is Becoming King in a Fa...	2016-07-25	14.818000	15.426000	14.758000	15.334000	15.334000	67360500	...	15.076000	0.5
2	15	2016-07-26	Yahoo, a Web Pioneer, Cleared the Way for Many...	2016-07-26	15.179333	15.333333	15.020000	15.300667	15.300667	51450000	...	15.317333	-0.0
3	16	2016-07-27	Santander Profit Down on Restructuring and Ban...	2016-07-27	15.289333	15.557333	15.128000	15.232667	15.232667	43335000	...	15.266667	-0.0
4	17	2016-07-28	Credit Suisse Posts a Surprise Profit in the S...	2016-07-28	15.196667	15.384000	15.106667	15.374000	15.374000	36286500	...	15.303333	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	
163	176	2023-05-24	Metalheads Take on the World in John Wray's Ne...	2023-05-24	182.229996	184.220001	178.220001	182.899994	182.899994	137605100	...	185.884995	-5.9
164	177	2023-05-25	Where Are the Most Profitable	2023-05-25	186.539993	186.779999	180.580002	184.470001	184.470001	96870700	...	183.684998	1.5

```
facebook_stock_movement=df.copy()
facebook_stock_movement
```

	Unnamed: 0	date	Headlines	Date	Open	High	Low	Close	Adj Close	Volume	...	SMA2	close_
0	13	2016-07-22	Republicans and Democrats in Congress Speak in...	2016-07-22	14.799333	14.966667	14.592000	14.818000	14.818000	38695500	...	14.759000	0.1
1	14	2016-07-25	How Sponsored Content Is Becoming King in a Fa...	2016-07-25	14.818000	15.426000	14.758000	15.334000	15.334000	67360500	...	15.076000	0.5
2	15	2016-07-26	Yahoo, a Web Pioneer, Cleared the Way for Many...	2016-07-26	15.179333	15.333333	15.020000	15.300667	15.300667	51450000	...	15.317333	-0.0
3	16	2016-07-27	Santander Profit Down on Restructuring and Ban...	2016-07-27	15.289333	15.557333	15.128000	15.232667	15.232667	43335000	...	15.266667	-0.0
4	17	2016-07-28	Credit Suisse Posts a Surprise Profit in the S...	2016-07-28	15.196667	15.384000	15.106667	15.374000	15.374000	36286500	...	15.303333	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	
163	176	2023-05-24	Metalheads Take on the World in John Wray's Ne...	2023-05-24	182.229996	184.220001	178.220001	182.899994	182.899994	137605100	...	185.884995	-5.9
164	177	2023-05-25	Where Are the Most Profitable	2023-05-25	186.539993	186.779999	180.580002	184.470001	184.470001	96870700	...	183.684998	1.5

```
from nltk.tokenize import sent_tokenize
tokens=[]
for text in df['Processed Headlines']:
  sentences=sent_tokenize(text)
  for sentence in sentences:
    words=word_tokenize(sentence)
    tokens.append(words)
```

Start coding or generate with AI.

```
import gensim
from gensim.models import word2vec
num_features = 300
num_processor = 4
context = 10
downsampling = 0.001
model = word2vec.Word2Vec(sentences=tokens, workers = num_processor,
                         vector_size = num_features,
                         window = context, sample = downsampling)
```

```
vocab_size = len(model.wv.key_to_index)
print("Vocab size", vocab_size)
```

```
Vocab size 90
```

```
def get_average_word2vec(text, model):
    # Tokenize the processed text
    tokens = word_tokenize(text.lower())

    # Filter tokens that are in the Word2Vec model's vocabulary
    valid_tokens = [token for token in tokens if token in model.wv]

    # If no valid tokens, return a zero vector
    if not valid_tokens:
        return np.zeros(model.vector_size)

    # Get embeddings for valid tokens
    embeddings = [model.wv[token] for token in valid_tokens]

    # Compute the average embedding
    avg_embedding = np.mean(embeddings, axis=0)

    return avg_embedding

# Calculate average Word2Vec embedding for the processed text
avg_embedding = df["Processed Headlines"].apply(lambda x:get_average_word2vec(x,model))
```

```
df['avg_embedding']=avg_embedding
```

```
embedding_df = pd.DataFrame(df['avg_embedding'].to_list())
embedding_df.columns = [f'embedding_{i}' for i in range(embedding_df.shape[1])]
```

```
embedding_df
```

	embedding_0	embedding_1	embedding_2	embedding_3	embedding_4	embedding_5	embedding_6	embedding_7	embedding_8	embedding_
0	0.029614	0.252443	-0.030467	0.070945	-0.094907	-0.246161	0.185759	0.259996	-0.061755	0.06687
1	0.030395	0.271116	-0.031883	0.077131	-0.101516	-0.267208	0.200718	0.281081	-0.066501	0.07328
2	0.029377	0.279466	-0.032863	0.082246	-0.102937	-0.277988	0.209013	0.296541	-0.067302	0.07166
3	0.061502	0.540643	-0.059468	0.158168	-0.200302	-0.526085	0.394473	0.557785	-0.130908	0.14681
4	0.025449	0.234796	-0.027748	0.068024	-0.086530	-0.232202	0.173649	0.245202	-0.058706	0.06250
...	...	...	...	...	...	...	...	...	...	.
222	0.026607	0.252284	-0.028890	0.074593	-0.092697	-0.250274	0.187974	0.268332	-0.060238	0.06475
223	0.027830	0.243978	-0.029027	0.069212	-0.089211	-0.237595	0.179770	0.251516	-0.059154	0.06555
224	0.026608	0.231756	-0.025999	0.067455	-0.085021	-0.228583	0.171420	0.239535	-0.057691	0.06159
225	0.005899	0.119530	-0.016965	0.037829	-0.040291	-0.131159	0.100824	0.154597	-0.022834	0.01694
226	-0.011739	0.103828	-0.022525	0.041450	-0.031328	-0.141481	0.115177	0.193922	-0.009641	-0.02021

227 rows × 300 columns

```
df2=df[['Date','Adj Close','sentiment_scores','Subjectivity','Polarity','neg','pos','neu',]]
```

```
df['Movement']=df['Adj Close'].pct_change()
```

```
label=[]
for i in df['Movement']:
  if i>0:
    label.append(1)
  else :
    label.append(0)
```

```
df2['label']=label
```

```
<ipython-input-165-9955eea9c163>:1: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#returning-a-view-versus
  df2['label']=label
```

```
df2
```

	Date	Adj Close	sentiment_scores	Subjectivity	Polarity	neg	pos	neu	label
0	2016-07-22	14.818000	-0.2960	0.616667	-0.250000	0.084	0.000	-0.2960	0
1	2016-07-25	15.334000	0.0000	0.850000	0.450000	0.000	0.000	0.0000	1
2	2016-07-26	15.300667	0.1027	0.500000	0.500000	0.000	0.149	0.1027	0
3	2016-07-27	15.232667	0.5719	0.285859	0.020202	0.058	0.162	0.5719	0
4	2016-07-28	15.374000	0.5719	0.390909	0.127273	0.135	0.186	0.5719	1
...	...	...	...	...	...	...	...	...	...
163	2023-05-24	182.899994	0.3182	0.454545	0.136364	0.000	0.204	0.3182	0
164	2023-05-25	184.470001	0.4927	0.500000	0.500000	0.000	0.347	0.4927	1
165	2023-05-26	193.169998	-0.1027	0.000000	0.000000	0.135	0.000	-0.1027	1
166	2023-05-30	201.160004	0.2382	0.500000	0.250000	0.000	0.178	0.2382	1
167	2023-05-31	203.929993	-0.2263	0.500000	-0.500000	0.174	0.000	-0.2263	1

168 rows × 9 columns

```
merged_df =df2.merge(embedding_df, left_index=True, right_index=True)
```

```
merged_df.head()
```

```
        Date        Adj  sentiment_scores  Subjectivity   Polarity    neg    pos      neu  label  embedding_0  ...  embedding_290  embedding
                  Close
0  2016-07-22  14.818000          -0.2960      0.616667  -0.250000  0.084  0.000  -0.2960      0    -0.001310  ...       0.000531       0.00
1  2016-07-25  15.334000           0.0000      0.850000   0.450000  0.000  0.000   0.0000      1    -0.001879  ...       0.001753      -0.00
2  2016-07-26  15.300667           0.1027      0.500000   0.500000  0.000  0.149   0.1027      0    -0.000873  ...       0.000915       0.00
3  2016-07-27  15.232667           0.5719      0.285859   0.020202  0.058  0.162   0.5719      0     0.000012  ...       0.000151       0.00
4  2016-07-28  15.374000           0.5719      0.390909   0.127273  0.135  0.186   0.5719      1     0.000355  ...      -0.000137       0.00

5 rows × 309 columns
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```python
merged_df['Date'] = pd.to_datetime(merged_df['Date'])
X=merged_df.drop(columns=['label'])
y=merged_df['label']
```

```python
split=int(0.7*X.shape[0])
```

```python
from sklearn.model_selection import train_test_split
X_train = X.iloc[0:split,:]
X_test=X.iloc[split:,:]
y_train=y.iloc[0:split]
y_test=y.iloc[split:]
```

```python
date=X_test['Date']
adjclose=X_test['Adj Close']
X_train.drop(columns=['Date','Adj Close'],inplace=True)
X_test.drop(columns=['Date','Adj Close'],inplace=True)
```

```
<ipython-input-172-e6d9407bb483>:3: 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
  X_train.drop(columns=['Date','Adj Close'],inplace=True)
<ipython-input-172-e6d9407bb483>:4: 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
  X_test.drop(columns=['Date','Adj Close'],inplace=True)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```python
X_test.head()
```

```
     sentiment_scores  Subjectivity  Polarity    neg    pos      neu  embedding_0  embedding_1  embedding_2  embedding_3  ...  embeddi
117           -0.0516      0.000000  0.000000  0.186  0.171  -0.0516    -0.000750     0.000831     0.000844    -0.000585  ...      0.0
118            0.0000      0.000000  0.000000  0.000  0.000   0.0000    -0.000963     0.001187    -0.000147    -0.000321  ...      0.0
119           -0.6705      0.454545  0.136364  0.159  0.000  -0.6705     0.000491     0.000012    -0.000271    -0.000115  ...      0.0
120            0.0000      0.000000  0.000000  0.000  0.000   0.0000    -0.001320    -0.001857     0.000376    -0.000203  ...      0.0
121           -0.4003      0.000000  0.000000  0.085  0.000  -0.4003    -0.001369     0.000840    -0.000328    -0.000835  ...      0.0

5 rows × 306 columns
```

```
X_train.head()
```

	sentiment_scores	Subjectivity	Polarity	neg	pos	neu	embedding_0	embedding_1	embedding_2	embedding_3	...	embedding
0	-0.2960	0.616667	-0.250000	0.084	0.000	-0.2960	-0.001310	0.001129	0.000136	0.001258	...	0.00
1	0.0000	0.850000	0.450000	0.000	0.000	0.0000	-0.001879	-0.002048	-0.002467	0.002716	...	0.00
2	0.1027	0.500000	0.500000	0.000	0.149	0.1027	-0.000873	0.002414	0.000044	0.001100	...	0.00
3	0.5719	0.285859	0.020202	0.058	0.162	0.5719	0.000012	-0.000342	0.000353	-0.000005	...	0.00
4	0.5719	0.390909	0.127273	0.135	0.186	0.5719	0.000355	0.000083	-0.000370	-0.000390	...	-0.00

5 rows × 306 columns

```
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,f1_score,recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier()
#mnb = MultinomialNB()
dtc = DecisionTreeClassifier(max_depth=3)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
abc = AdaBoostClassifier(n_estimators=50, random_state=2)
bc = BaggingClassifier(n_estimators=50, random_state=2)
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT':gbdt,
}
```

```
def train_classifier(clf, X_train, y_train, X_test, y_test):
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate evaluation metrics
    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    f1score = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Return a dictionary of results including the confusion matrix
    return {
        'Classifier': clf.__class__.__name__,
        'Recall': recall,
        'Precision': precision,
        'F1-score': f1score,
        'Accuracy': accuracy,
        'Confusion Matrix': conf_matrix
    }


# List to store results
results = []

# Iterate through classifiers, train, evaluate, and store results
for name, clf in clfs.items():
    # Train and evaluate the classifier
    result = train_classifier(clf, X_train, y_train, X_test, y_test)
    results.append(result)

    # Print the results for the current classifier
    print(f"For {name}")
    print(f"Recall: {result['Recall']}")
    print(f"Precision: {result['Precision']}")
    print(f"F1-score: {result['F1-score']}")
    print(f"Accuracy: {result['Accuracy']}")
    print("Confusion Matrix:")
    print(result['Confusion Matrix'])
    print("************************")

# Create a DataFrame from the results list
df = pd.DataFrame(results)

# Display the DataFrame
print(df)
```

```
For SVC
Recall: 0.7777777777777778
Precision: 0.6176470588235294
F1-score: 0.6885245901639345
Accuracy: 0.6274509803921569
Confusion Matrix:
[[11 13]
 [ 6 21]]
************************
For KN
Recall: 0.5555555555555556
Precision: 0.5769230769230769
F1-score: 0.5660377358490566
Accuracy: 0.5490196078431373
Confusion Matrix:
[[13 11]
 [12 15]]
************************
For DT
Recall: 0.48148148148148145
Precision: 0.5416666666666666
F1-score: 0.5098039215686274
Accuracy: 0.5098039215686274
Confusion Matrix:
[[13 11]
 [14 13]]
************************
For LR
Recall: 0.4074074074074074
Precision: 0.5238095238095238
F1-score: 0.4583333333333333
Accuracy: 0.49019607843137253
Confusion Matrix:
[[14 10]
 [16 11]]
************************
For RF
Recall: 0.7037037037037037
Precision: 0.6129032258064516
F1-score: 0.6551724137931035
Accuracy: 0.6078431372549019
Confusion Matrix:
[[12 12]
 [ 8 19]]
************************
For AdaBoost
Recall: 0.7407407407407407
Precision: 0.5714285714285714
F1-score: 0.6451612903225806
Accuracy: 0.5686274509803921
Confusion Matrix:
[[ 9 15]
 [ 7 20]]
************************
For BgC
Recall: 0.5925925925925926
Precision: 0.5517241379310345
F1-score: 0.5714285714285714
```

```
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

# Sample data (replace with your actual data)
# X_train, X_test, y_train, y_test should be defined with your data
# For demonstration purposes, this example assumes they are already defined

# Define the parameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, 40, 50, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the RandomForestClassifier
rf = RandomForestClassifier()

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=100,  # Number of parameter settings that are sampled
    cv=5,  # 5-fold cross-validation
    verbose=2,
    random_state=42,
    n_jobs=-1  # Use all available cores
)
```

```
# Fit RandomizedSearchCV
random_search.fit(X_train, y_train)
```

```
# Get the best parameters and best cross-validation score
best_params = random_search.best_params_
best_cv_score = random_search.best_score_

print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_cv_score}")

# Apply the best estimator to the test data
best_rf = random_search.best_estimator_
    # Make predictions on the test set
y_pred = best_rf.predict(X_test)

    # Calculate evaluation metrics
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

    # Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1score))
print(conf_matrix)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 50, 'boot
Best Cross-Validation Score: 0.5717391304347826
Accuracy: 0.45
Precision: 0.48
Recall: 0.52
F1-score: 0.50
[[ 9 15]
 [13 14]]
```

Start coding or generate with AI.

```
param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
}

# Initialize the SVC
svc = SVC()

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=svc,
    param_grid=param_grid,
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1  # Use all available cores
)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters and best cross-validation score
best_params = grid_search.best_params_
best_cv_score = grid_search.best_score_

print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_cv_score}")

# Apply the best estimator to the test data
best_svc1 = grid_search.best_estimator_
y_pred = best_svc1.predict(X_test)

    # Calculate evaluation metrics
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

    # Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1score))
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'C': 1000, 'gamma': 1, 'kernel': 'sigmoid'}
Best Cross-Validation Score: 0.5550724637681159
Accuracy: 0.55
Precision: 0.57
Recall: 0.63
F1-score: 0.60
[[11 13]
 [10 17]]

Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.52      0.46      0.49        24
           1       0.57      0.63      0.60        27

    accuracy                           0.55        51
   macro avg       0.55      0.54      0.54        51
weighted avg       0.55      0.55      0.55        51
```

Start coding or generate with AI.

```
param_grid = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.001, 0.01, 0.1, 1, 10],
    'algorithm': ['SAMME', 'SAMME.R']
}

# Initialize the AdaBoostClassifier
ada = AdaBoostClassifier()

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=ada,
    param_grid=param_grid,
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1  # Use all available cores
)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters and best cross-validation score
best_params = grid_search.best_params_
best_cv_score = grid_search.best_score_

print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_cv_score}")

# Apply the best estimator to the test data
best_ada = grid_search.best_estimator_
y_pred = best_ada.predict(X_test)

    # Calculate evaluation metrics
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

    # Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1score))
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Best Parameters: {'algorithm': 'SAMME', 'learning_rate': 0.01, 'n_estimators': 50}
Best Cross-Validation Score: 0.6242753623188406
Accuracy: 0.57
Precision: 0.59
Recall: 0.59
F1-score: 0.59
[[13 11]
 [11 16]]

Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.54      0.54        24
           1       0.59      0.59      0.59        27

    accuracy                           0.57        51
   macro avg       0.57      0.57      0.57        51
weighted avg       0.57      0.57      0.57        51
```

Start coding or generate with AI.

```
param_grid = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [100, 200, 300]
}

# Initialize the LogisticRegression
log_reg = LogisticRegression()

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid,
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1  # Use all available cores
)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters and best cross-validation score
best_params = grid_search.best_params_
best_cv_score = grid_search.best_score_

print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_cv_score}")

# Apply the best estimator to the test data
best_log_reg = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
Best Parameters: {'C': 0.001, 'max_iter': 200, 'penalty': 'none', 'solver': 'lbfgs'}
Best Cross-Validation Score: 0.5210144927536232
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
810 fits failed out of a total of 1800.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got elasticnet penalty.

--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
```

```
y_pred = best_log_reg.predict(X_test)

    # Calculate evaluation metrics
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

    # Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1score))
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.49
Precision: 0.52
Recall: 0.44
```

```
F1-score: 0.48
[[13 11]
 [15 12]]

Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.54      0.50        24
           1       0.52      0.44      0.48        27

    accuracy                           0.49        51
   macro avg       0.49      0.49      0.49        51
weighted avg       0.49      0.49      0.49        51
```

```
param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
}


# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=SVC(probability = True),
    param_grid=param_grid,
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1  # Use all available cores
)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters and best cross-validation score
best_params = grid_search.best_params_
best_cv_score = grid_search.best_score_

print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_cv_score}")

# Apply the best estimator to the test data
best_svc2 = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'C': 1000, 'gamma': 1, 'kernel': 'sigmoid'}
Best Cross-Validation Score: 0.5550724637681159
```

```
y_pred = best_svc2 .predict(X_test)

    # Calculate evaluation metrics
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

    # Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1score))
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.55
Precision: 0.57
Recall: 0.63
F1-score: 0.60
[[11 13]
 [10 17]]

Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.46      0.49        24
           1       0.57      0.63      0.60        27

    accuracy                           0.55        51
   macro avg       0.55      0.54      0.54        51
weighted avg       0.55      0.55      0.55        51
```

```
best_svc = SVC(C=100,kernel='sigmoid', gamma=1.0)
best_gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
best_abc= AdaBoostClassifier(algorithm ='SAMME', learning_rate= 1, n_estimators=300)
svc=SVC()
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
```

```
from sklearn.ensemble import VotingClassifier
voting = VotingClassifier(estimators=[('svc',best_svc), ('adb', best_abc), ('gbdt', best_gbdt)],voting='hard')
voting.fit(X_train,y_train)
y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.4117647058823529
Precision 0.4444444444444444
```

```
from sklearn.ensemble import StackingClassifier
stack = StackingClassifier(estimators=[('svc',best_svc), ('adb', best_abc), ('gbdt', best_gbdt)], final_estimator = svc)
stack.fit(X_train,y_train)
y_pred = stack.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred ))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy 0.45098039215686275
Precision 0.0

Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.96      0.62        24
           1       0.00      0.00      0.00        27

    accuracy                           0.45        51
   macro avg       0.23      0.48      0.31        51
weighted avg       0.22      0.45      0.29        51
```

```
from sklearn.ensemble import StackingClassifier
stack = StackingClassifier(estimators=[('svc',best_svc), ('gbdt', best_gbdt)], final_estimator = rfc)
stack.fit(X_train,y_train)
y_pred = stack.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred ))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy 0.49019607843137253
Precision 0.5294117647058824

Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.67      0.55        24
           1       0.53      0.33      0.41        27

    accuracy                           0.49        51
   macro avg       0.50      0.50      0.48        51
weighted avg       0.50      0.49      0.48        51
```

```
voting.fit(X_train,y_train)
y_pred = voting.predict(X_test)
```

```
y_pred
```

```
array([0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 1])
```

```
stock_final_prices = adjclose.tolist()
date=date.tolist()

buy_signals = []
buy_dates = []
sell_signals = []
sell_dates=[]

for i in range(len(y_pred)):
    if y_pred[i] == 1:
        buy_signals.append(i)
        buy_dates.append(date[i])
    else :
      sell_signals.append(i)
      sell_dates.append(date[i])
buy_df = pd.DataFrame({
    'Signal': 'Buy',
    'Date': buy_dates,
    'Index': buy_signals
})

sell_df = pd.DataFrame({
    'Signal': 'Sell',
    'Date': sell_dates,
    'Index': sell_signals
})

# Concatenate buy and sell DataFrames
trading = pd.concat([buy_df, sell_df])

# Sort by date
trading = trading.sort_values(by='Date').reset_index(drop=True)

# Print the trading DataFrame
print("Buy signals indices:", buy_signals)
print("Buy dates:", buy_dates)
print("Sell signals indices:", sell_signals)
print("Sell dates:", sell_dates)
print(date)
```

```
Buy signals indices: [1, 2, 4, 5, 8, 9, 12, 14, 16, 17, 19, 21, 22, 25, 28, 30, 31, 32, 33, 34, 35, 39, 40, 43, 44, 45, 50]
Buy dates: [Timestamp('2023-02-28 00:00:00'), Timestamp('2023-03-02 00:00:00'), Timestamp('2023-03-06 00:00:00'), Timestamp('2023-03
Sell signals indices: [0, 3, 6, 7, 10, 11, 13, 15, 18, 20, 23, 24, 26, 27, 29, 36, 37, 38, 41, 42, 46, 47, 48, 49]
Sell dates: [Timestamp('2023-02-27 00:00:00'), Timestamp('2023-03-03 00:00:00'), Timestamp('2023-03-09 00:00:00'), Timestamp('2023-0
[Timestamp('2023-02-27 00:00:00'), Timestamp('2023-02-28 00:00:00'), Timestamp('2023-03-02 00:00:00'), Timestamp('2023-03-03 00:00:0
```

```
print(len(date))
print(len(buy_dates))
print(len(sell_dates))
```

```
51
27
24
```

```
initial_cash = 500  # Starting with $500
cash = initial_cash
portfolio_value = []
positions = 0  # Number of shares currently held
buy_price = 0 # Price at which shares were bought
buy_rate=[]
sell_rate=[]
buy_time=[]
sell_time=[]

num_trades = 0
wins = 0

# Metrics
daily_returns = []
peak_value = initial_cash
max_drawdown = 0.0

for i, price in enumerate(stock_final_prices):
    if date[i] in buy_dates and cash > 0:
        buy_price = price
        buy_rate.append(price)
        buy_time.append(date[i])
        positions = cash / buy_price
        cash = 0
        num_trades =num_trades +1
        print(f"Buying at {buy_price} on date {date[i]}")

    elif date[i] in sell_dates and positions > 0:
        sell_price = price
        sell_rate.append(price)
        sell_time.append(date[i])
        cash = sell_price * positions
        positions = 0
        num_trades =num_trades +1
        print(f"Selling at {sell_price} on date {date[i]}")
        if sell_price > buy_price:
            wins += 1

    # Calculate current portfolio value
    current_value = cash + positions * price
    portfolio_value.append(current_value)

    # Calculate daily return
    if len(portfolio_value) > 1:
        daily_return = (portfolio_value[-1] - portfolio_value[-2]) / portfolio_value[-2]
        daily_returns.append(daily_return)

    # Update peak value and calculate drawdown
    if current_value > peak_value:
        peak_value = current_value
    drawdown = (current_value - peak_value) / peak_value
    if drawdown < max_drawdown:
        max_drawdown = drawdown

# Calculate Sharpe Ratio
risk_free_rate = 0.0  # Assuming risk-free rate is 0%
average_daily_return = np.mean(daily_returns)
std_daily_return = np.std(daily_returns)
sharpe_ratio = (average_daily_return - risk_free_rate) / std_daily_return if std_daily_return != 0 else np.nan

# Calculate Win Ratio
win_ratio = wins / num_trades if num_trades > 0 else 0.0

# Output results
print(f"Final Portfolio Value: ${portfolio_value[-1]:.2f}")
print(f"Sharpe Ratio: {sharpe_ratio:.2f}")
print(f"Maximum Drawdown: {max_drawdown:.2%}")
print(f"Number of Trades Executed: {num_trades}")
print(f"Win Ratio: {win_ratio:.2%}")
```

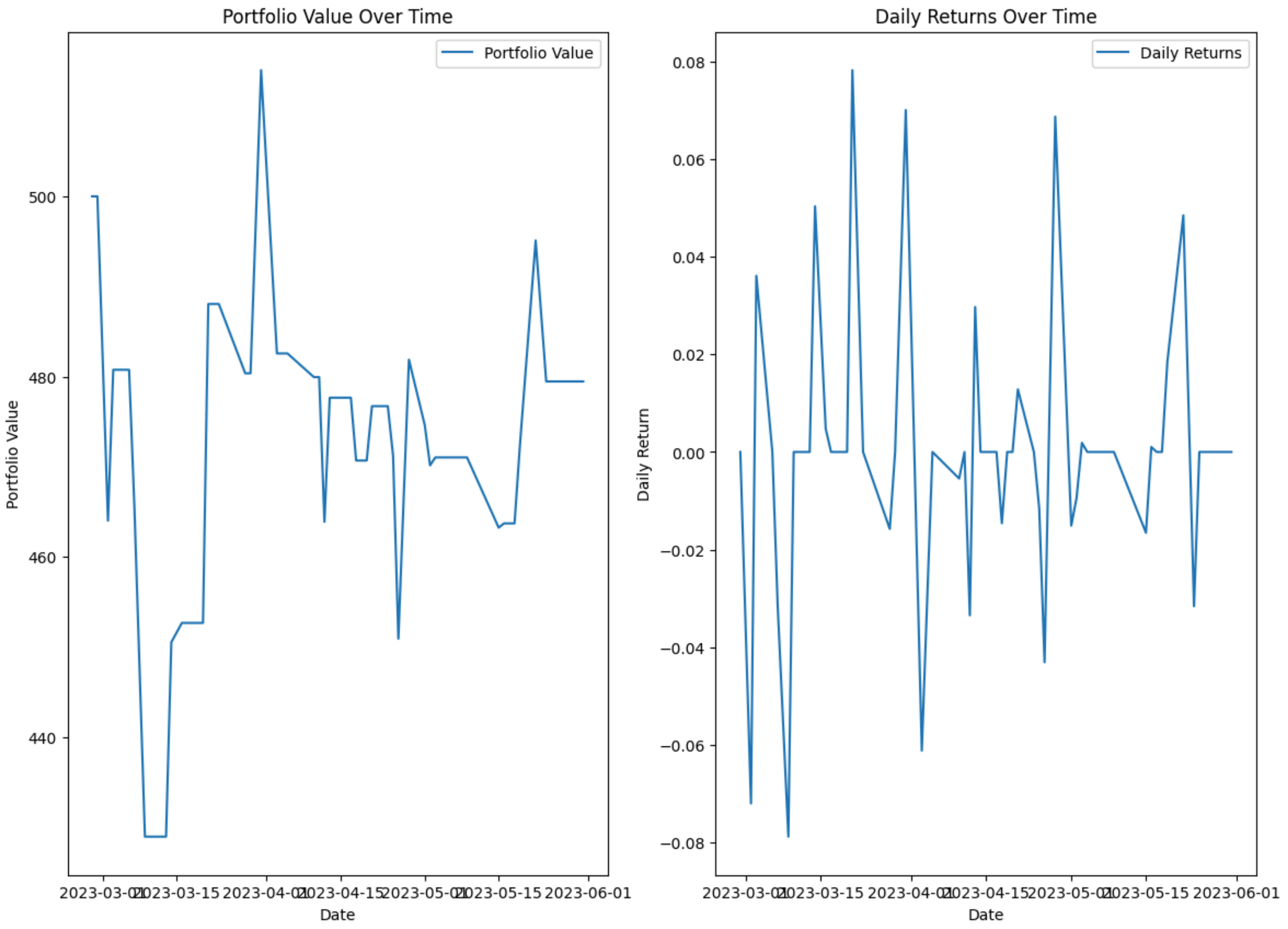
```
Buying at 205.7100067138672 on date 2023-02-28 00:00:00
Selling at 197.7899932861328 on date 2023-03-03 00:00:00
Buying at 193.80999755859372 on date 2023-03-06 00:00:00
Selling at 172.9199981689453 on date 2023-03-09 00:00:00
Buying at 174.47999572753906 on date 2023-03-13 00:00:00
Selling at 184.1300048828125 on date 2023-03-16 00:00:00
Buying at 183.25 on date 2023-03-20 00:00:00
Selling at 197.5800018310547 on date 2023-03-21 00:00:00
Buying at 192.22000122070312 on date 2023-03-23 00:00:00
Selling at 189.19000244140625 on date 2023-03-28 00:00:00
Buying at 193.8800048828125 on date 2023-03-29 00:00:00
Selling at 194.7700042724609 on date 2023-04-03 00:00:00
Buying at 185.5200042724609 on date 2023-04-05 00:00:00
Selling at 184.50999450683597 on date 2023-04-10 00:00:00
```

```
Buying at 186.7899932861328 on date 2023-04-11 00:00:00
Selling at 185.8999938964844 on date 2023-04-13 00:00:00
Buying at 187.0399932861328 on date 2023-04-17 00:00:00
Selling at 184.30999755859372 on date 2023-04-18 00:00:00
Buying at 162.99000549316406 on date 2023-04-20 00:00:00
Selling at 165.0800018310547 on date 2023-04-21 00:00:00
Buying at 162.5500030517578 on date 2023-04-24 00:00:00
Selling at 160.61000061035156 on date 2023-05-03 00:00:00
Buying at 169.14999389648438 on date 2023-05-09 00:00:00
Selling at 166.52000427246094 on date 2023-05-16 00:00:00
Buying at 176.88999938964844 on date 2023-05-18 00:00:00
Selling at 182.8999938964844 on date 2023-05-24 00:00:00
Buying at 203.92999267578125 on date 2023-05-31 00:00:00
Final Portfolio Value: $479.45
Sharpe Ratio: -0.01
Maximum Drawdown: -14.21%
Number of Trades Executed: 27
Win Ratio: 18.52%
```

```
portfolio_value
```

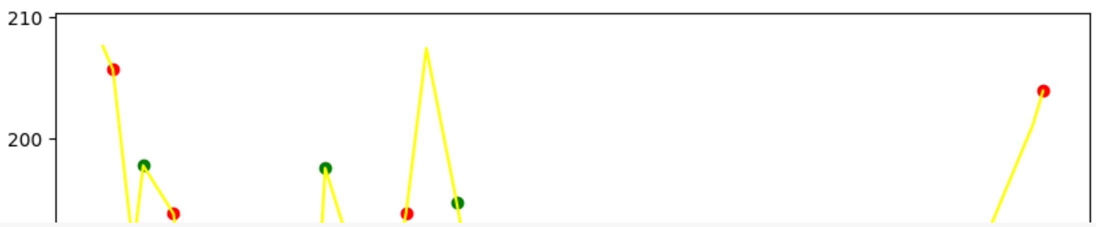
```
[500.0,
 500.00000000000006,
 464.00269229978977,
 480.7495669407305,
 480.74956694073046,
 465.6184179087611,
 428.93150653892184,
 428.93150653892184,
 428.93150653892184,
 450.51574654368767,
 452.65452961573016,
 452.65452961573016,
 452.6545296157302,
 488.0517478325303,
 488.0517478325303,
 480.3584995192689,
 480.3584995192689,
 514.0044100760447,
 482.56356843105783,
 482.5635684310579,
 479.9363912780518,
 479.9363912780518,
 463.87770209070914,
 477.64963550600567,
 477.64963550600567,
 477.64963550600567,
 470.6779636122991,
 470.6779636122991,
 470.6779636122991,
 476.7133963819285,
 476.7133963819285,
 471.1998713368703,
 450.90546488875566,
 481.87496478067004,
 474.60183550295864,
 470.14409089738535,
 471.0239153885782,
 471.0239153885782,
 471.0239153885782,
 471.02391538857825,
 463.22692300978173,
 463.7003087977499,
 463.7003087977499,
 463.7003087977499,
 472.21987467932956,
 495.1047281398457,
 479.4549377666486,
 479.4549377666486,
 479.4549377666486,
 479.4549377666486,
 479.4549377666486]
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(14, 10))

# Portfolio Value over Time
plt.subplot(1, 2, 1)
plt.plot(date, portfolio_value, label='Portfolio Value')
plt.xlabel('Date')
plt.ylabel('Portfolio Value')
plt.title('Portfolio Value Over Time')
plt.legend()

# Daily Returns
plt.subplot(1, 2, 2)
plt.plot(date[1:], daily_returns, label='Daily Returns')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.title('Daily Returns Over Time')
plt.legend()
plt.show()
```



Start coding or generate with AI.

```
plt.figure(figsize=(10,7))
plt.plot(date,stock_final_prices,color="yellow",label="stock price")
plt.scatter(buy_time,buy_rate,color='r',label='Buy Dates')
plt.scatter(sell_time,sell_rate,color='g',label='Sell Dates')
plt.xticks(rotation='vertical')
plt.legend()
plt.show()
```



```
plt.figure(figsize=(8, 6))
plt.hist(daily_returns, bins=20, alpha=0.75, color='blue', edgecolor='black')
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.title('Histogram of Daily Returns')
plt.show()
```