LSE Team 3 Assignment Report

Hedge funds need to correlate stock price movements with macroeconomic factors to analyse and justify trends and patterns. VP Analytics have partnered with LSE in order to make this analysis.

The focus of the analysis was based on Apple (AAPL), Google (GOOGL) and NVIDIA (NVDA) stocks.

A group of LSE learners in data analysis was formed, and analysis was carried on.

The questions considered were:

1. Is the sentiment analysis helping to predict the stock prices?
2. How the stock prices move, with respect to the quarterly earnings reports (QERs), dates (QERDs) and results?
3. Which macroeconomic factors correlate mostly with stock prices?

A **sentiment analysis**, on abstracts of articles from the NY Times (NYT) was carried out (LSE_Team3_Sentiment_Analysis_On_Abstract_and_Snippet_Column.ipynb). An excel file was provided with abstract, snipped and other valuable information about NT Times (NYT) articles for tech and finance.

After checking the integrity of the file and its contents, only the “abstract”, “snipped” and “pub_date” columns were kept. The date column type was turned from object into datetime.

```
# pub_date as datetime
df['pub_date'] = pd.to_datetime(df['pub_date'])
```

All the words were turned into lower case, and the punctuation removed.

2. Prepare the data for NLP

2a) Change to lower case and join the elements in each of the columns

```
# Change 'abstract' and 'snippet' column properties from object into string
df['abstract'] = df['abstract'].apply(str)
df['snippet'] = df['snippet'].apply(str)

# Change all to lower case and join with a space.
df['abstract'] = df['abstract'].apply(lambda x: " ".join(x.lower() for x in x.split()))
df['snippet'] = df['snippet'].apply(lambda x: " ".join(x.lower() for x in x.split()))

print(df['abstract'].head())
print('\n')
print(df['snippet'].head())
```

```
0    like its host city, the ces trade show largely...
1    by carving the stock market into specialized s...
2    the couple met in 2013 through a mutual friend...
3    all of the weddings right here on one handy pa...
4    stocks slumped from the start, with selling wo...
Name: abstract, dtype: object


0    like its host city, the ces trade show largely...
1    by carving the stock market into specialized s...
2    the couple met in 2013 through a mutual friend...
3    all of the weddings right here on one handy pa...
4    stocks slumped from the start, with selling wo...
Name: snippet, dtype: object
```

2b) Replace punctuation in each of the columns

```
# Replace all the punctuations in review column.
df['abstract'] = df['abstract'].str.replace(r'[^\w\s]', '', regex=True)
df['snippet'] = df['snippet'].str.replace(r'[^\w\s]', '', regex=True)

# View output.
print(df['abstract'].head())
print('\n')
print(df['snippet'].head())
```

```
0    like its host city the ces trade show largely ...
1    by carving the stock market into specialized s...
2    the couple met in 2013 through a mutual friend...
3    all of the weddings right here on one handy pa...
4    stocks slumped from the start with selling wor...
Name: abstract, dtype: object


0    like its host city the ces trade show largely ...
1    by carving the stock market into specialized s...
2    the couple met in 2013 through a mutual friend...
3    all of the weddings right here on one handy pa...
4    stocks slumped from the start with selling wor...
Name: snippet, dtype: object
```

The words were tokenized, alphanumeric characters and stop words were removed.

```
# Apply tokenisation to both columns.
df_copy['abstract_tokenised'] = df_copy['abstract'].apply(word_tokenize)
df_copy['snippet_tokenised'] = df_copy['snippet'].apply(word_tokenize)

# View DataFrame.
df_copy
```

4b) Remove alphanumeric characters and stopwords

```
# Delete all the alphanumeric characters.
abstract_no_alphanum = [word for word in abstract_all if word.isalnum()]
snippet_no_alphanum = [word for word in snippet_all if word.isalnum()]
```

```
# Remove all the stopwords in the "abstract_no_alphanum" column
english_stopwords = set(stopwords.words('english'))

# Create a filtered list of tokens without stopwords for abstract
tokens_abstract_filtered = [x for x in abstract_no_alphanum if x.lower() not in english_stopwords]

# Define an empty string variable.
tokens_abstract_string = ''

#for value in tokens_abstract_alpanum:
for value in tokens_abstract_filtered:
    # Add each filtered token word to the string.
    tokens_abstract_string = tokens_abstract_string + value + ' '
```

```
# Remove all the stopwords in the "snippet_no_alphanum" column
english_stopwords = set(stopwords.words('english'))

# Create a filtered list of tokens without stopwords for snippet
tokens_snippet_filtered = [x for x in snippet_no_alphanum if x.lower() not in english_stopwords]

# Define an empty string variable.
tokens_snippet_string = ''

#for value in tokens_snippet_alpanum:
for value in tokens_snippet_filtered:
    # Add each filtered token word to the string.
    tokens_snippet_string = tokens_snippet_string + value + ' '
```

Finally, word clouds of the “abstract” (Figure below) and “snipped” columns were created.

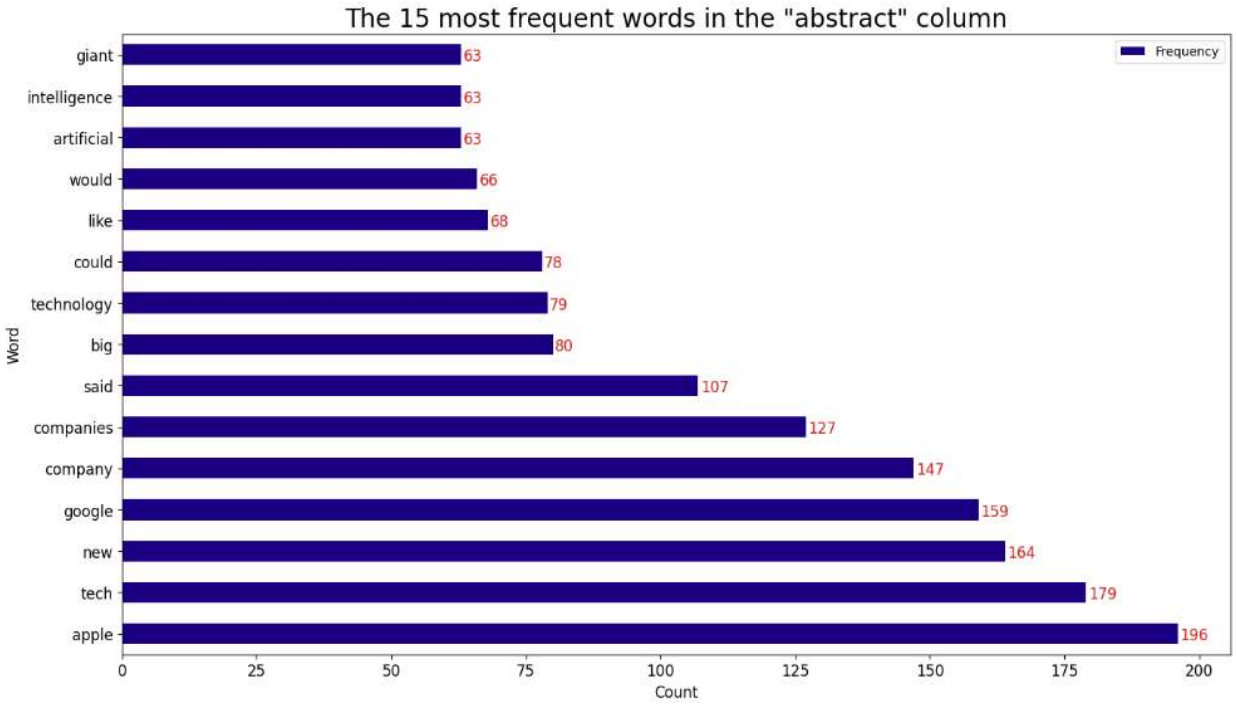
4c) Create wordcloud without stopwords

```
# Create a wordcloud without stop words for tokens_abstract_string.
wordcloud_abstract_filtered = WordCloud(width = 1600, height = 900,
                background_color ='white',
                colormap='plasma',
                min_font_size = 10).generate(tokens_abstract_string)
```

```
# Plot the wordcloud image.
plt.figure(figsize = (16, 9), facecolor = None)
plt.imshow(wordcloud_abstract_filtered)
plt.title('Word cloud of the "abstract" column, withour stopwords')
plt.axis('off')
plt.tight_layout(pad = 0)
plt.show()
```

The 15 most frequent words were listed, and histograms of polarity and sentiment scores were drawn for the “abstract” (Figure below) and “snipped” columns.

```
# Set the plot type.
ax = abstract_counts.plot(kind='barh', figsize=(16, 9), fontsize=12,
                  colormap ='plasma')

# Set the labels.
ax.set_xlabel('Count', fontsize=12)
ax.set_ylabel('Word', fontsize=12)
ax.set_title("The 15 most frequent words in the \"abstract\" column",
             fontsize=20)

# Draw the bar labels.
for i in ax.patches:
    ax.text(i.get_width()+.41, i.get_y()+.1, str(round((i.get_width()), 2)),
            fontsize=12, color='red')
```



5. Review polarity and sentiment: Plot histograms of polarity (use 15 bins) and sentiment scores for the respective columns.

```
# Import the necessary package
from textblob import TextBlob

# Provided function.
def generate_polarity(comment):
    '''Extract polarity score (-1 to +1) for each comment'''
    return TextBlob(comment).sentiment[0]
```

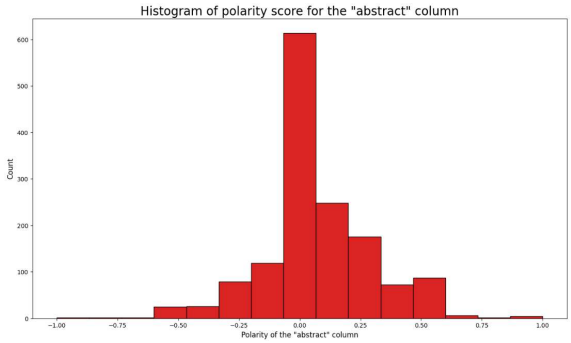
```
# Determine polarity of columns.

# Populate a new column with polarity scores for each review.
df['polarity_abstract'] = df['abstract'].apply(generate_polarity)

# Populate a new column with polarity scores for each summary.
df['polarity_snippet'] = df['snippet'].apply(generate_polarity)

# View output.
print(df['polarity_abstract'].head())
print()
print(df['polarity_snippet'].head())
```

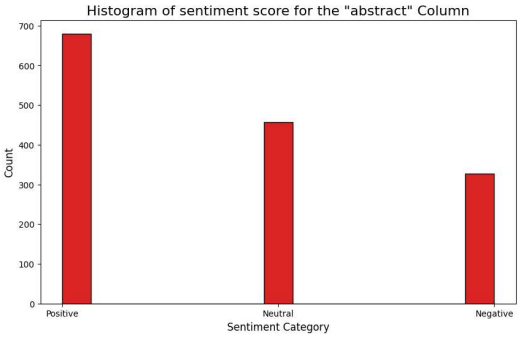
```
0    0.103571
1    0.000000
2    0.000000
3    0.442857
4   -0.200000
Name: polarity_abstract, dtype: float64

0    0.071429
1    0.000000
2    0.000000
3    0.442857
4   -0.200000
Name: polarity_snippet, dtype: float64
```

5a) Histograms of polarity and sentiment scores for the "abstract" column.

```
# Review: Create a histogram plot with bins = 15.

# Histogram of polarity scores for the "abstract" column.

# Set the number of bins.
num_bins = 15

# Set the plot area.
plt.figure(figsize=(16,9))

# Define the bars.
n, bins, patches = plt.hist(df['polarity_abstract'], num_bins, edgecolor='black', facecolor='red', alpha=1.0)

# Histogram of sentiment score
plt.xlabel('Polarity of the "abstract" column', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Histogram of polarity score for the \"abstract\" column', fontsize=20)

plt.show()
```

```
# Histogram of sentiment score for the "abstract" column.

# Import necessary libraries
import matplotlib.pyplot as plt

# Classify each review as positive, negative, or neutral based on polarity.
df['sentiment_abstract'] = df['polarity_abstract'].apply(lambda x: 'Positive' if x > 0 else ('Negative' if x < 0 else 'Neutral'))
df['sentiment_abstract'].value_counts()

# Set the number of bins.
num_bins = 15

# Set the plot area.
plt.figure(figsize=(10, 6))

# Define the bars.
n, bins, patches = plt.hist(df['sentiment_abstract'], num_bins, edgecolor='black', facecolor='red', alpha=1.0)

# Add labels and title.
plt.xlabel('Sentiment Category', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Histogram of sentiment score for the "abstract" Column', fontsize=16)

# Display the plot.
plt.show()
```





Among the 15 most frequent words, “apple”, “google” and “artificial” (for artificial intelligence, referred to NVDIA) were selected for future use. The polarity is a numeric value giving the sentiment of the word. Polarity values of articles mentioning “apple”, “google” and “artificial” will be used later for the correlation with stock movements of the above-mentioned stocks.

The stock prices were downloaded from Yahoo Finance in order to carry out the analysis on the stock price movement and the correlation with QER’s, macroeconomic factors and sentiment analysis. This was shown in the LSE_Team3_ALL_Stock_prices.ipynb file.

```
import yfinance as yf
import pandas as pd

# Define the stocks and date range
stocks = ['NVDA', 'AAPL', 'GOOGL']


# Fetch the historical data
data = {}
for stock in stocks:
    data[stock] = yf.download(stock)#, start=start_date, end=end_date)

# Example: Display Nvidia's data
print(data['NVDA'].head())  # First few rows of NVDA data
print(data['AAPL'].head())  # First few rows of NVDA data
print(data['GOOGL'].head())  # First few rows of NVDA data
```

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

Macroeconomic factors were downloaded with APIs from alphavantage.co. For example, the code for retrieving the quarterly earnings of AAPL stock is:

Get quarterly earnings for AAPL

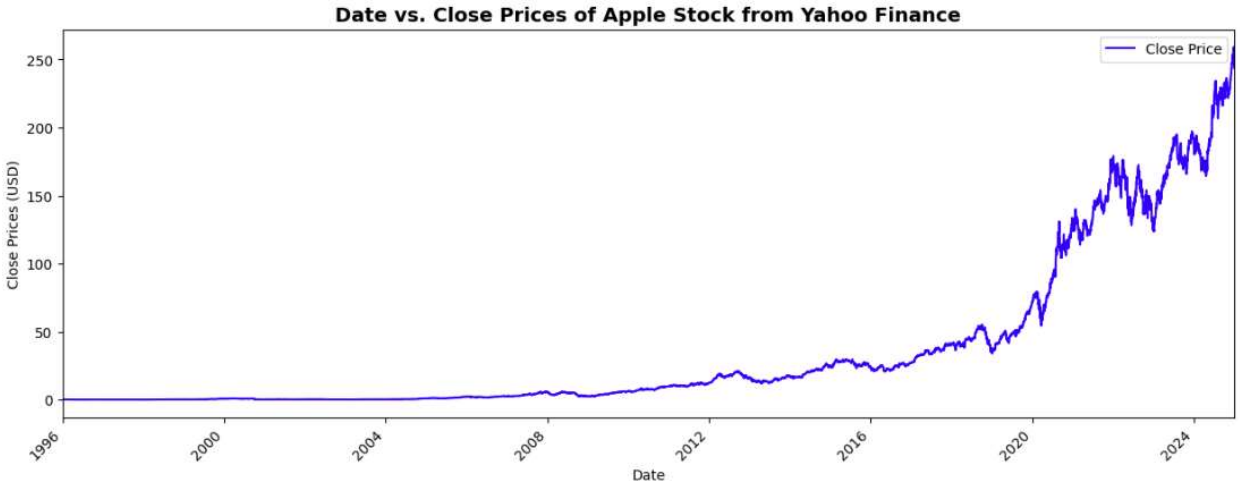
```
import requests

# replace the "demo" apikey below with your own key from https://www.alphavantage.co/support/#api-key
# If necessary, change the AAPL ticker in the url with the ticker of the needed stock (e.g., GOOGL or NVDA)
url = 'https://www.alphavantage.co/query?function=EARNINGS&symbol=AAPL&apikey={api_key}'

r = requests.get(url)
earnings_AAPL = r.json()

earnings_AAPL

# Extract the AAPL earnings data from the JSON response
quarterly_earnings_AAPL = earnings_AAPL.get("quarterlyEarnings", [])

# Convert to DataFrames
df_quarterly = pd.DataFrame(quarterly_earnings_AAPL)

# Convert date columns to datetime format
df_quarterly["fiscalDateEnding"] = pd.to_datetime(df_quarterly["fiscalDateEnding"])
df_quarterly["reportedDate"] = pd.to_datetime(df_quarterly["reportedDate"])

print("\nQuarterly Earnings:")
print(df_quarterly.head())

QERDs_AAPL_complete = df_quarterly  # The quarterly earnings report dates for AAPL
QERDs_AAPL_complete

# Create a CSV file as output.
QERDs_AAPL_complete.to_csv(r'G:\My Drive\LSE_Team_3\Data\QERDs_AAPL_complete.csv', index=False)
```

```
Quarterly Earnings:
  fiscalDateEnding reportedDate reportedEPS estimatedEPS surprise  \
0       2024-12-31   2025-01-30         2.4         2.36     0.04
1       2024-09-30   2024-10-31        1.64          1.6     0.04
2       2024-06-30   2024-08-01         1.4         1.35     0.05
3       2024-03-31   2024-05-02        1.53          1.5     0.03
4       2023-12-31   2024-02-01        2.18          2.1     0.08

  surprisePercentage   reportTime
0             1.6949  post-market
1                2.5  post-market
2             3.7037  post-market
3                  2  post-market
4             3.8095  post-market
```

With this method, the AAPL, GOOGL, NVDA quarterly earnings reports (QERs) plus consumer price index data, unemployment data, and FED funds interest rates were collected.

Hereafter the code for three stocks is similar and therefore only the code regarding the AAPL stock will be analysed. Similar outlooks can be drawn for AAPL and NVDA stocks.

Data from Yahoo finance were saved, columns were renamed, and column types were changed accordingly. The close price vs. date was plotted.



As previously mentioned QERs were collected from alphavantage.co APIs.

```
import requests

# replace the "demo" apikey below with your own key from https://www.alphavantage.co/support/#api-key
url = 'https://www.alphavantage.co/query?function=EARNINGS&symbol=AAPL&apikey=MA94VYU1XDN8Y7Z4'  # MA94VYU1XDN8Y7Z4 is the key
r = requests.get(url)
data = r.json()

# Show the data
# data

# Extract all the reported dates
# Extract 'quarterlyEarnings' into a DataFrame
df = pd.DataFrame(data['quarterlyEarnings'])

# Display the DataFrame
print(df)

#QERDs_AAPL = df[['reportedDate', 'surprise', 'surprisePercentage']]  # The quarterly earnings report dates for AAPL
QERDs_AAPL = df

# Create a CSV file as output.
QERDs_AAPL.to_csv(r'G:\My Drive\LSE_Team_3\Data\QERDs_AAPL.csv', index=False)
```

A data range before and after QERD was chosen and dates were taken accordingly. Furthermore, weekends and stock exchange closing days were removed.

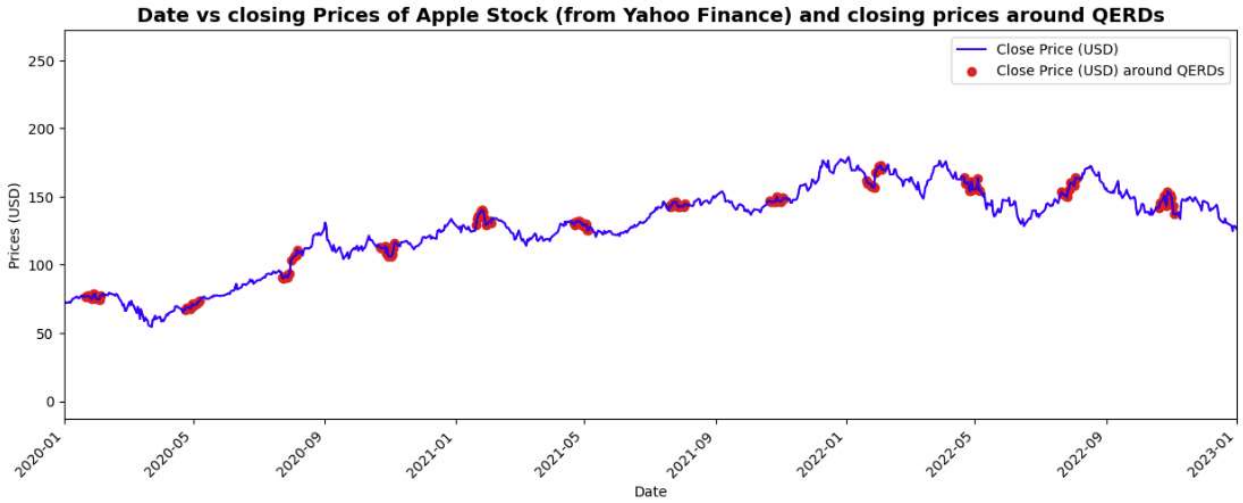
```python
import pandas as pd

# Sample QERDs_AAPL DataFrame (replace this with your actual data)
# QERDs_AAPL = pd.DataFrame({
#     'reportedDate': ['2025-01-01', '2025-01-10'],
# })

# Convert 'reportedDate' to datetime format
QERDs_AAPL['reportedDate'] = pd.to_datetime(QERDs_AAPL['reportedDate'])

# Define the number of days before and after QERDs dynamically
Days_before_and_after_each_QERD = int(input("Enter the number of days before and after each QERD: "))

# Calculate start and end dates
QERDs_AAPL['StartDate'] = QERDs_AAPL['reportedDate'] - pd.Timedelta(days=Days_before_and_after_each_QERD)
QERDs_AAPL['EndDate'] = QERDs_AAPL['reportedDate'] + pd.Timedelta(days=Days_before_and_after_each_QERD)

# Function to create columns for each date between StartDate and EndDate
def create_date_columns(row):
    date_range = pd.date_range(row['StartDate'], row['EndDate'], freq='D')
    return pd.Series(date_range.values, index=[f"Day_{i+1}" for i in range(len(date_range))])

# Apply the function to each row
expanded_dates = QERDs_AAPL.apply(create_date_columns, axis=1)

# Concatenate the original DataFrame with the expanded dates
expanded_QERDs_AAPL = pd.concat([QERDs_AAPL[['reportedDate']], expanded_dates], axis=1)

# Display the result
print(expanded_QERDs_AAPL.shape)
print('Expanded QERDs AAPL:')
print(expanded_QERDs_AAPL.head())  # Display first few rows for checking
```

```
# install the module for downloading the market holidays
# !pip install pandas-market-calendars
```

```
import pandas as pd
import numpy as np
from pandas.tseries.offsets import BDay
from pandas.tseries.holiday import USFederalHolidayCalendar

# Sample QERDs_AAPL DataFrame (replace this with your actual data)
# QERDs_AAPL = pd.DataFrame({
#     'reportedDate': ['2025-01-01', '2025-01-10'],
# })

# Convert 'reportedDate' to datetime format
QERDs_AAPL['reportedDate'] = pd.to_datetime(QERDs_AAPL['reportedDate'])


# Define U.S. Federal Holidays
us_holidays = USFederalHolidayCalendar().holidays(
    start=QERDs_AAPL['reportedDate'].min() - pd.DateOffset(days=30),
    end=QERDs_AAPL['reportedDate'].max() + pd.DateOffset(days=30)
)

# Function to get previous and next business days
def get_business_days(reference_date, num_days, direction):
    """ Get `num_days` business days before (-1) or after (1) reference_date, adjusting for weekends/holidays. """
    business_days = []
    date = reference_date
    while len(business_days) < num_days:
        date += BDay(direction)  # Move to next/previous business day
        while date in us_holidays:  # If it's a holiday, adjust again
            date += BDay(direction)
        business_days.append(date)
    return business_days

# Function to create adjusted date columns
def create_adjusted_date_columns(row):
    past_dates = get_business_days(row['reportedDate'], Days_before_and_after_each_QERD, -1)
    future_dates = get_business_days(row['reportedDate'], Days_before_and_after_each_QERD, 1)

    full_dates = past_dates + [row['reportedDate']] + future_dates
    return pd.Series(full_dates, index=[f"Day_{i+1}" for i in range(len(full_dates))])

# Apply function
adjusted_dates = QERDs_AAPL.apply(create_adjusted_date_columns, axis=1)

# Concatenate results
adjusted_QERDs_AAPL = pd.concat([QERDs_AAPL[['reportedDate']], adjusted_dates], axis=1)

# Display results
print("Adjusted QERDs AAPL, with holidays and weekends removed:")
print(adjusted_QERDs_AAPL.shape)
print(adjusted_QERDs_AAPL.head())
```

This was required to avoid taking weekends or holidays in the ranges before and after QERDs.

Then stock price movements for each date range around the QERD were calculated and plotted.



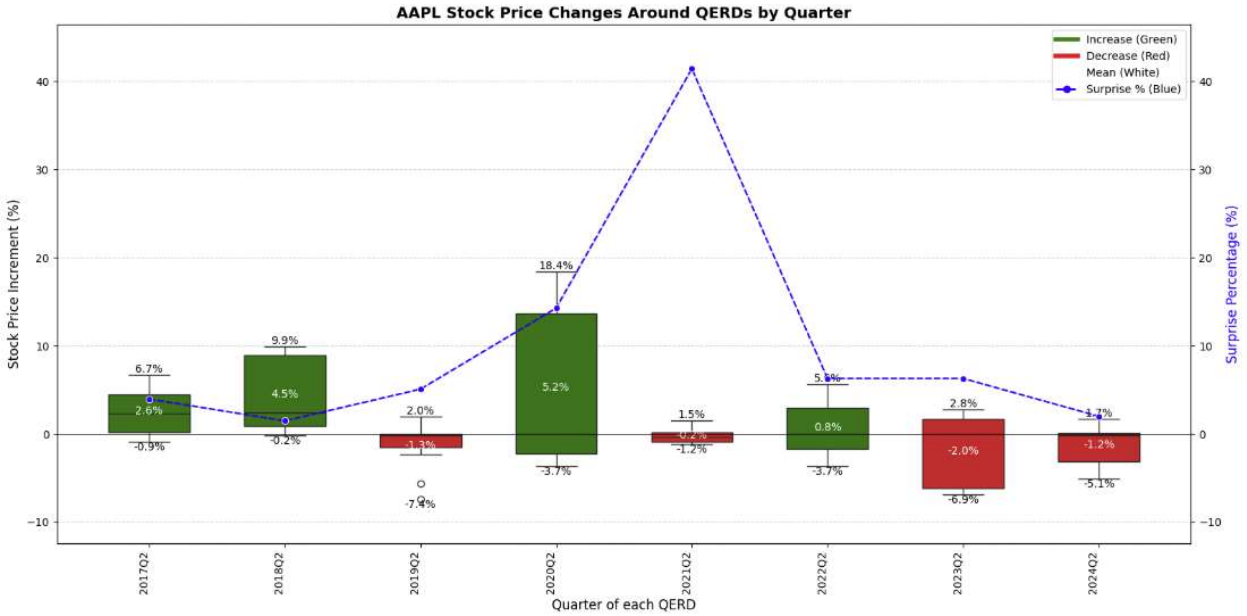
From the QERs the surprise percentage calculated as $\left(\frac{reported\ EPS}{estimated\ EPS} - 1\right) \times 100$ was compared with the stock price increment, calculated as $\left(\frac{price\ at\ QERD+n\ days}{price\ at\ QERD} - 1\right) \times 100$, or $\left(\frac{price\ at\ QERD-\ \ days}{price\ at\ QERD} - 1\right) \times 100$.

Then, a boxplot of stock price around each QERD together with the surprise percentage (dashed blue line) was drawn (as example, Figure below).

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

# Ensure 'reportedDate' is datetime
heatmap_data['reportedDate'] = pd.to_datetime(heatmap_data['reportedDate'])
QERDs_AAPL['reportedDate'] = pd.to_datetime(QERDs_AAPL['reportedDate'])

# Merge datasets
merged_data = pd.merge(heatmap_data, QERDs_AAPL[['reportedDate', 'surprisePercentage']], on='reportedDate', how='left')

# Make sure the reportedDate column is categorical to avoid errors in set_xticklabels
merged_data['reportedDate'] = pd.Categorical(merged_data['reportedDate'], categories=merged_data['reportedDate'].unique(), ordered=True)


# Create Quarter column
merged_data['reportedDate'] = merged_data['reportedDate'].dt.to_period('Q').astype(str)


#Filter dataset for a specific quarter (e.g., Q2) of each year
merged_data = merged_data[merged_data['reportedDate'].str.endswith('Q2')]


# Filter dataset for the desired range

start_quarter = '2017Q1'
end_quarter = '2024Q4'
merged_data = merged_data[
    (merged_data['reportedDate'] >= start_quarter) & (merged_data['reportedDate'] <= end_quarter)
]


# Calculate means
quarter_means = merged_data.groupby('reportedDate')['Price_increment_%'].mean()

# Determine the common y-axis range
left_min = merged_data['Price_increment_%'].min()
left_max = merged_data['Price_increment_%'].max()
right_min = merged_data['surprisePercentage'].min()
right_max = merged_data['surprisePercentage'].max()
```

```
common_min = min(left_min, right_min)
common_max = max(left_max, right_max)

# Create the figure
fig, ax1 = plt.subplots(figsize=(16, 8))

# Boxplot on the primary y-axis
sns.boxplot(
    data=merged_data,
    x='reportedDate',
    y='Price_increment_%',
    width=0.6,
    hue='reportedDate',  # Assigning hue to avoid the warning
    palette=['red' if mean < 0 else 'green' for mean in quarter_means],
    ax=ax1
)

# Add the secondary y-axis
ax2 = ax1.twinx()
sns.lineplot(
    data=merged_data,
    x='reportedDate',
    y='surprisePercentage',
    marker='o',
    linestyle='dashed',
    color='blue',
    ax=ax2
)

# Set y-axis limits to align the 0
ax1.set_ylim(common_min-5, common_max+5)
ax2.set_ylim(common_min-5, common_max+5)

# Add labels to the boxplot
for i, quarter in enumerate(merged_data['reportedDate'].unique()):
    subset = merged_data[merged_data['reportedDate'] == quarter]
    max_value = subset['Price_increment_%'].max()
    min_value = subset['Price_increment_%'].min()
    mean_value = subset['Price_increment_%'].mean()

    ax1.text(i, max_value, f'{max_value:.1f}%', ha='center', va='bottom', fontsize=10, color='black')
    ax1.text(i, min_value, f'{min_value:.1f}%', ha='center', va='top', fontsize=10, color='black')
    ax1.text(i, mean_value, f'{mean_value:.1f}%', ha='center', va='center', fontsize=10, color='white')

# Set labels and title
ax1.set_xlabel('Quarter of each QERD', fontsize=12)
ax1.set_ylabel('Stock Price Increment (%)', fontsize=12)
ax2.set_ylabel('Surprise Percentage (%)', fontsize=12, color='blue')
plt.title('AAPL Stock Price Changes Around QERDs by Quarter', fontsize=14, fontweight='bold')
```

```
# Rotate x-axis labels properly
ax1.set_xticks(range(len(merged_data['reportedDate'].unique())))  # Set fixed tick positions
ax1.set_xticklabels(merged_data['reportedDate'].unique(), rotation=90, ha='right')  # Rotate labels

# Add horizontal lines at zero
ax1.axhline(y=0, color='black', linestyle='-', linewidth=0.5)
ax2.axhline(y=0, color='black', linestyle='-', linewidth=0.5)


# Add gridlines for readability
ax1.grid(axis='y', linestyle='--', alpha=0.6)

# Add Legend
legend_labels = ['Increase (Green)', 'Decrease (Red)', 'Mean (White)', 'Surprise % (Blue)']
handles = [
    plt.Line2D([0], [0], color='green', lw=4),
    plt.Line2D([0], [0], color='red', lw=4),
    plt.Line2D([0], [0], color='white', marker='o', lw=0),
    plt.Line2D([0], [0], color='blue', linestyle='dashed', marker='o', lw=2)
]
plt.legend(handles, legend_labels, loc='upper right', fontsize=10)

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



Each box represents the stock price movement in a specific quarter (X-axis) from 2017 to 2024, calculated around QERDs. The box height represents 50% of the price movement range (middle values), with the minimum and maximum shown as whiskers above and below.

- Green Box → Stock price increased from day 1 to the last day of the period.
- Red Box → Stock price decreased over the same period.
- White Number → Average price movement (%) inside the box.

A correlation is found when:

- Green box → Surprise factor is positive.
- Red box → Surprise factor is negative.

Each quarter was manually checked for correlation between price and surprise factor, using 1, 2, 3, 5, and 10-day windows before and after each QERD.

Why a Boxplot? Shows price movement range and averages. Green and red colours reflect price increases or decreases (like candlestick charts).

- Primary Y-axis → Price movement (%).
- Secondary Y-axis → Surprise factor (%).

Aligned zero levels for easier comparison. Macroeconomic data was downloaded from Alpha Vantage as .csv files.

The data are donwloaded from https://www.alphavantage.co/

There is a limit of 25 requests per day!

1. Click on Get Free API key
2. fill up the form
3. You will receive your free API right away on the same web page (no need to refresh it)
4. replace the API keys in the box below with your API key

API key

```
# api_key = "MA94VYU1XDN8Y7Z4"
api_key = "KNTHHYST9DW9AN5V"
```

```
# Imports the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
import msoffcrypto
from io import BytesIO
```

Example of API request, get quarterly earnings for AAP stock:

Get quarterly earnings for AAPL

[3]:
```
import requests

# replace the "demo" apikey below with your own key from https://www.alphavantage.co/support/#api-key
# If necessary, change the AAPL ticker in the url with the ticker of the needed stock (e.g., GOOGL or NVDA)
url = 'https://www.alphavantage.co/query?function=EARNINGS&symbol=AAPL&apikey={api_key}'

r = requests.get(url)
earnings_AAPL = r.json()

earnings_AAPL

# Extract the AAPL earnings data from the JSON response
quarterly_earnings_AAPL = earnings_AAPL.get("quarterlyEarnings", [])

# Convert to DataFrames
df_quarterly = pd.DataFrame(quarterly_earnings_AAPL)

# Convert date columns to datetime format
df_quarterly["fiscalDateEnding"] = pd.to_datetime(df_quarterly["fiscalDateEnding"])
df_quarterly["reportedDate"] = pd.to_datetime(df_quarterly["reportedDate"])

print("\nQuarterly Earnings:")
print(df_quarterly.head())

QERDs_AAPL_complete = df_quarterly  # The quarterly earnings report dates for AAPL
QERDs_AAPL_complete

# Create a CSV file as output.
QERDs_AAPL_complete.to_csv(r'G:\My Drive\LSE_Team_3\Data\QERDs_AAPL_complete.csv', index=False)
```

The quarterly earnings data, the macroeconomic factors data and the article tokenized data were imported.

Import quarterly earnings

```
file_path_QERDs_AAPL_complete = r'G:\My Drive\LSE_Team_3\Data\QERDs_AAPL_complete.csv'
data_QERDs_AAPL_complete = pd.read_csv(file_path_QERDs_AAPL_complete)

# Change date columns type from object into date
data_QERDs_AAPL_complete['fiscalDateEnding'] = pd.to_datetime(data_QERDs_AAPL_complete['fiscalDateEnding'])
data_QERDs_AAPL_complete['reportedDate'] = pd.to_datetime(data_QERDs_AAPL_complete['reportedDate'])

# Preview the DataFrames
print(data_QERDs_AAPL_complete.head())
```

```
# Load the CPI data
file_path_df_CPI = r'G:\My Drive\LSE_Team_3\Data\df_CPI.csv'
data_CPI = pd.read_csv(file_path_df_CPI)


data_CPI['date'] = pd.to_datetime(data_CPI['date'])  # Change date column type from object into date
data_CPI.rename(columns={'value': 'value_CPI'}, inplace=True)  # Rename value column for clarity
data_CPI = data_CPI.drop(columns=['Unnamed: 0'])  # drop unnecessary columns

# Preview the DataFrames
print('CPI\n', data_CPI.head())

##############################################################################################

# Load the unemployment data
file_path_data_unemployment = r'G:\My Drive\LSE_Team_3\Data\data_unemployment.csv'
data_unemployment = pd.read_csv(file_path_data_unemployment)

data_unemployment['date'] = pd.to_datetime(data_unemployment['date'])  # Change date column type from object into date
data_unemployment.rename(columns={'value': 'value_unemployment'}, inplace=True)  # Rename value column for clarity
data_unemployment = data_unemployment.drop(columns=['Unnamed: 0'])  # drop unnecessary columns

# Preview the DataFrames
print('\n Unemployment rate \n', data_unemployment.head())

##############################################################################################

# Load the FED Funds interest rates data
file_path_FEDFUNDS = r'G:\My Drive\LSE_Team_3\Data\FEDFUNDS.csv'
data_FEDFUNDS = pd.read_csv(file_path_FEDFUNDS)

data_FEDFUNDS.rename(columns={'observation_date': 'date'}, inplace=True)  # Rename date columns for consistency
data_FEDFUNDS['date'] = pd.to_datetime(data_FEDFUNDS['date'])  # Change date column type from object into date
data_FEDFUNDS.rename(columns={'FEDFUNDS': 'value_FEDFUNDS'}, inplace=True)  # Rename value column for clarity


# Preview the DataFrames
print('\n FED Funds interest rates \n', data_FEDFUNDS.head())
```

Import Articles Tokenized

```
df_articles_tokenized = pd.read_csv(r'G:\My Drive\LSE_Team_3\Data\articles_tokenized.csv')

df_articles_tokenized.rename(columns={'pub_date': 'date'}, inplace=True)  # Rename pub_date column for clarity
df_articles_tokenized['date'] = pd.to_datetime(df_articles_tokenized['date'])  # Change date column type from object into date
df_articles_tokenized = df_articles_tokenized.drop(columns=['index',
                                                            'sentiment_abstract',
                                                            'sentiment_snippet'])  # drop unnecessary columns

# Check the shape of the imported DataFrame
print("Shape of the imported DataFrame:", df_articles_tokenized.shape)

# View DataFrame.
df_articles_tokenized
```

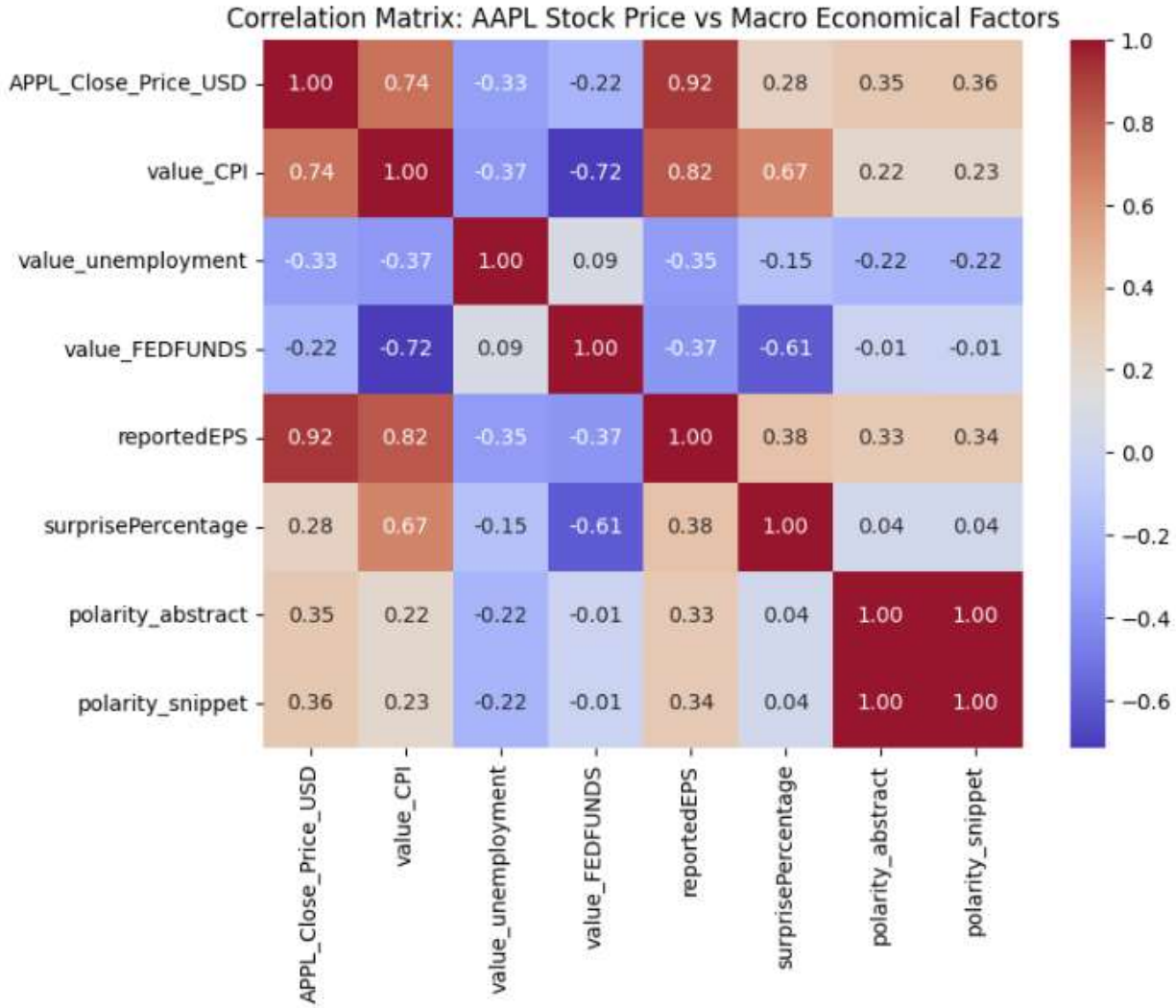
All the dataframes were merged.

Merge all the dataframes

```
# Merge all dataframes based on 'date' in data_AAPL (left join)
merged_all_df = (
    data_AAPL
    .merge(data_CPI, on='date', how='left')
    .merge(data_unemployment, on='date', how='left')
    .merge(data_FEDFUNDS, on='date', how='left')
    .merge(df_apple, on='date', how='left')
)

# Merge earnings data (quarterly) separately, using 'reportedDate'
merged_all_df = merged_all_df.merge(data_QERDs_AAPL_complete, left_on='date', right_on='reportedDate', how='left')

# Sort by date
merged_all_df = merged_all_df.sort_values(by='date')

# Forward-fill and backward-fill missing values
merged_all_df.ffill(inplace=True)  # Fill forward
merged_all_df.bfill(inplace=True)  # Fill backward

# Reset index to start from 0
merged_all_df = merged_all_df.reset_index(drop=True)

# Display the final merged dataset
print(merged_all_df)
print(merged_all_df.shape)
```

The correlation matrix was calculated to investigate which factor could have influenced the stock price. The time frame under consideration was the entire period in which the stock was present in the market. Data from quarterly earnings, macroeconomic factors and sentiment analysis were used to forward-fill and backwards-fill the missing values.

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

# Select relevant columns for correlation analysis
corr_data = merged_all_df[['APPL_Close_Price_USD',
                           'value_CPI',
                           'value_unemployment',
                           'value_FEDFUNDS',
                           'reportedEPS',
                           'surprisePercentage',
                           'polarity_abstract',
                           'polarity_snippet'
                          ]]
print(corr_data)

# Compute correlation matrix
corr_matrix = corr_data.corr()

# Plot correlation heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix: AAPL Stock Price vs Macro Economical Factors")
plt.show()
```



In this example, the apple stock price has a strong positive correlation with the reported EPS and CPI index. There is a slight negative correlation with the unemployment value and the FED fund interest rate. Such negative correlations were expected, as when the unemployment rate decreases, more people have a job and can spend more money, and when the interest rates decrease borrowing money is cheaper and tech industries benefit by borrowing cheaper money.

Regression analysis was calculated giving similar results.

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split

# Define independent (X) and dependent (Y) variables
X = corr_data[['value_CPI',
               'value_unemployment',
               'value_FEDFUNDS',
               'reportedEPS',
               'surprisePercentage',
          #     'polarity_abstract',
          #     'polarity_snippet'
              ]]
Y = corr_data['APPL_Close_Price_USD']

# Add a constant for the intercept
X = sm.add_constant(X)

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Fit the OLS (Ordinary Least Squares) regression model
model = sm.OLS(Y_train, X_train).fit()

# Get the summary of the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     APPL_Close_Price_USD   R-squared:                       0.877
Model:                              OLS   Adj. R-squared:                  0.877
Method:                   Least Squares   F-statistic:                 1.275e+04
Date:                  Tue, 18 Feb 2025   Prob (F-statistic):               0.00
Time:                          17:46:46   Log-Likelihood:                -38275.
No. Observations:                  8926   AIC:                         7.656e+04
Df Residuals:                      8920   BIC:                         7.661e+04
Df Model:                             5
Covariance Type:              nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                -82.6171      2.431    -33.988      0.000     -87.382     -77.852
value_CPI              0.3440      0.010     32.834      0.000       0.323       0.365
value_unemployment     1.0658      0.115      9.281      0.000       0.841       1.291
value_FEDFUNDS         3.7114      0.089     41.721      0.000       3.537       3.886
reportedEPS           75.4340      0.838     90.025      0.000      73.791      77.076
surprisePercentage    -0.1828      0.010    -18.660      0.000      -0.202      -0.164
==============================================================================
Omnibus:                     1340.022   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             6058.391
Skew:                           0.667   Prob(JB):                         0.00
Kurtosis:                       6.810   Cond. No.                     2.64e+03
==============================================================================
```

In addition, the Granger causality test was applied to evaluate whether macroeconomic factors, such as CPI, help predict stock price movements. The Granger causality test determines whether past values of a variable (e.g., CPI, earnings, or sentiment) contain predictive information about stock prices. It checks whether lagged values of X (e.g., CPI) improve the prediction of Y (e.g., AAPL stock price).

For example, using maxlag=4 evaluates whether the past 1 to 4 time periods of CPI influence AAPL's stock price. If the p-value for any lag < 0.05, X is said to Granger-cause Y → X provides predictive information.

If all p-values > 0.05, X does not Granger-cause Y → No strong predictive relationship is found. In this analysis, the test evaluates whether past values of CPI predict AAPL's stock price.

```
from statsmodels.tsa.stattools import grangercausalitytests
grangercausalitytests(merged_all_df[['value_CPI', 'APPL_Close_Price_USD']].dropna(), maxlag=4)
```

Do the macroeconomic factors or the sentiment analysis predict the stock movement?

	AAPL	GOOGL	NVDA
CPI Index	Y	Y	N
Unemployment rate	N	N	N
FED funds interest rate	N	N	N
Reported EPS	Y	Y	Y
Surprise Percentage	N	N	N
Polarity articles	Y	N	N

It is notable that the CPI index predicts AAPL and GOOGL stock prices, while Reported EPS predicts all three stocks. Sentiment polarity is only relevant for AAPL, supported by Granger causality testing.

According to the box plot analysis for AAPL the best timing for trade (buy or sell) is 1 to 2 days before or after QERDs on Q2 and Q3. For GOOGL the best timing for trade (buy or sell) is 5 days before or after QERDs on Q2 and Q3. For NVDA the best timing for trade (buy or sell) is 2 to 3 days before or after QERDs on Q2.

According to the correlation matrix, by following the CPI and reported EPS it is possible to forecast AAPL, GOOGL and NVDA stock movements. If these two macroeconomic factors rise, it is likely that also the stock prices will rise. GOOGL shows a somewhat strong opposite correlation with the unemployment rate, so when its value rises the GOOGL stock price decreases.

Some of the above-mentioned trends are supported also by the Granger causality test, which shows how CPI value can predict AAPL and GOOGL stock prices and the reported EPS can predict the prices of all the stock in exam.

However, it is shown that the sentiment analysis and its polarity do not correlate nor can predict the stock prices apart from the AAPL stock but supported only by the Granger causality test.

This work shows how it is possible to use data analysis for correlating trends in stock prices with external factors. This work can be extended by adding more macroeconomic factors like the treasury yield, inflation rate and GDP per capita. Correlation can be also affected by adopting different stock prices moving averages at different times and see how they change and correlate with the changing in macroeconomic factors values.

Further work can be also carried out by modifying the Granger causality test. The idea is to merge the macroeconomic factors dataframe and the stock prices dataframe shifted by a certain number of days, e.g., 4. The Granger causality test will calculate the p-values up to four days (lags) before. If the values are shifted apart by four days, the Granger causality test will predict the stock price the same day the value of the macroeconomic factor under study is released, then one day after, two days after and three days after. Resulting in the possibility of being able to predict future stock price values.