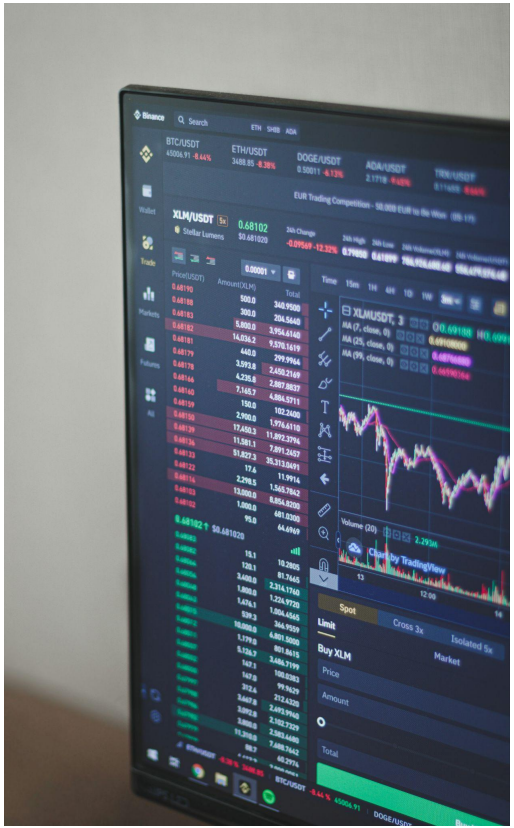
Stock Market News Sentiment Analysis & Summarization

Natural Language Processing Post Graduate Program

2025-05-02

Melissa Lindskär



Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- EDA Results
- Data Preprocessing
- Model Performance Summary
- Appendix

Executive Summary

Actionable Insights:

- Among all models tested, the Sentence Transformer + Random Forest (original dataset) achieved the highest F1-score (48%) on the validation set.
- The final model's performance on the test set showed balanced precision (54.7%) and recall (50%), indicating moderate capability to capture both positive and negative classes.
- Using Mistral-7B Instruct LLM, the weekly news articles were successfully summarized into top 3 positive and negative events, helping to extract actionable financial signals.

Recommendations for next analysis scope:

- To further improve prediction performance, consider:
  - Applying techniques like SMOTE to handle class imbalance more effectively.
  - Exploring more advanced sentence embeddings (e.g., fine-tuned Sentence Transformers for financial news).
  - Experimenting with ensemble models combining different classifiers for improved robustness.
- In production settings, continue monitoring LLM outputs for summarization consistency and fine-tune prompts based on real-world feedback.

Business Problem Overview and Solution Approach

Problem Definition:

- Financial markets are highly sensitive to news and events.
- Understanding which news events are most likely to impact stock prices is critical for investors and analysts.
- Manual analysis of large volumes of news articles is time-consuming and inconsistent.

Solution Approach:

- Phase 1: Predictive Modeling
  - Embedded financial news articles using Word2Vec, GloVe, and Sentence Transformers.
  - Built and compared multiple classification models to predict sentiment labels.
  - Selected the best-performing model based on F1-score.
- Phase 2: News Summarization
  - Applied a large language model (Mistral-7B Instruct) to summarize weekly news.
  - Extracted top 3 positive and top 3 negative events likely to influence stock movements.

Data Overview

The first five rows of the dataset:

	Date	News	Open	High	Low	Close	Volume	Label
0	2019-01-02	The tech sector experienced a significant dec...	41.740002	42.244999	41.482498	40.246914	130672400	-1
1	2019-01-02	Apple lowered its fiscal Q1 revenue guidance ...	41.740002	42.244999	41.482498	40.246914	130672400	-1
2	2019-01-02	Apple cut its fiscal first quarter revenue fo...	41.740002	42.244999	41.482498	40.246914	130672400	-1
3	2019-01-02	This news article reports that yields on long...	41.740002	42.244999	41.482498	40.246914	130672400	-1
4	2019-01-02	Apple's revenue warning led to a decline in U...	41.740002	42.244999	41.482498	40.246914	130672400	-1

Date: All news articles in these rows are from January 2, 2019.

News: The articles report negative events related to the tech sector and Apple's revenue guidance cuts, indicating a pessimistic market sentiment.

Open, High, Low, Close:
Opening price: $41.74
Highest price: $42.24
Lowest price: $41.48
Closing price: $40.25

These values are consistent across all five entries, suggesting they refer to the same trading day.
Volume: About 130.67 million shares were traded.
Label: Sentiment is -1 for all rows, meaning all news articles are classified as negative.

Note:
These early entries show that negative news, especially regarding a major company like Apple, coincided with lower closing stock prices on that day.

Data Overview

Total entries: 349 rows (observations/data points)
Number of columns: 8 columns (features/attributes/variables)

Column details:
Date: Object type
News: Object type (textual data containing news articles).
Open, High, Low, Close: All are of type float64, representing stock prices in dollars.
Volume: int64, representing the number of shares traded.
Label: int64, indicating sentiment (positive = 1, neutral = 0, negative = -1).

There are no missing values nor duplicates

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349 entries, 0 to 348
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    349 non-null    object
 1   News    349 non-null    object
 2   Open    349 non-null    float64
 3   High    349 non-null    float64
 4   Low     349 non-null    float64
 5   Close   349 non-null    float64
 6   Volume  349 non-null    int64
 7   Label   349 non-null    int64
dtypes: float64(4), int64(2), object(2)
memory usage: 21.9+ KB
```

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

Note:
The dataset is clean, with complete data for all features. Data types are appropriate for analysis after ensuring the 'Date' column is in datetime format

```
stock.duplicated().sum()
```

```
0
```

```
stock.shape
```

```
(349, 8)
```

```
stock.isnull().sum()
```

	0
Date	0
News	0
Open	0
High	0
Low	0
Close	0
Volume	0
Label	0

dtype: int64

Data Overview

Date: Data spans from January 2, 2019 to April 30, 2019. The median date is approximately February 5, 2019.
Open: Opening prices range from $37.57 to $66.82, with a mean of around $46.23.
High: Highest prices during the day vary between $37.82 and $67.06.
Low: Lowest prices during the day range from $37.31 to $65.86.
Close: Closing prices range from $36.25 to $64.81, with an average of $44.93.
Volume: Daily trading volume varies from about 45 million to 244 million shares. The average volume is around 129 million shares per day.
Label (Sentiment): Sentiment values are -1 (negative), 0 (neutral), or 1 (positive):
- The mean sentiment is approximately -0.05, indicating a slight overall negative bias.
- The standard deviation of 0.715 suggests moderate variability in sentiments day-to-day.

	Date	Open	High	Low	Close	Volume	Label
count	349	349.000000	349.000000	349.000000	349.000000	3.490000e+02	349.000000
mean	2019-02-16 16:05:30.085959936	46.229233	46.700458	45.745394	44.926317	1.289482e+08	-0.054441
min	2019-01-02 00:00:00	37.567501	37.817501	37.305000	36.254131	4.544800e+07	-1.000000
25%	2019-01-14 00:00:00	41.740002	42.244999	41.482498	40.246914	1.032720e+08	-1.000000
50%	2019-02-05 00:00:00	45.974998	46.025002	45.639999	44.596924	1.156272e+08	0.000000
75%	2019-03-22 00:00:00	50.707500	50.849998	49.777500	49.110790	1.511252e+08	0.000000
max	2019-04-30 00:00:00	66.817497	67.062500	65.862503	64.805229	2.444392e+08	1.000000
std	NaN	6.442817	6.507321	6.391976	6.398338	4.317031e+07	0.715119

Note:
Stock prices (Open, High, Low, Close) show moderate volatility, with standard deviations around 6 dollars. Trading volume varies significantly across different days, indicating fluctuating investor activity. Sentiment labels tend slightly toward negative overall during the observed period.

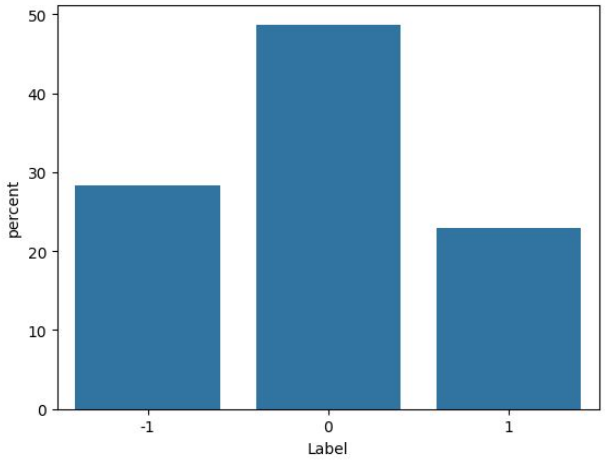
EDA Results - Univariate analysis

Label Distribution (Sentiment Analysis):

A bar plot of the "Label" column shows the distribution of sentiment classes:
Around 50% of the news articles are neutral (Label = 0).
About 28% are negative (Label = -1).
Approximately 22% are positive (Label = 1).

The dataset is somewhat imbalanced, with neutral sentiment being the most common class.

```
sns.countplot(data=stock, x="Label", stat="percent");
```



Note:
Since there is some class imbalance (more neutral labels), already now considering how to handle this during modeling (e.g., using class weights or resampling techniques) is good approach.

EDA Results - Univariate analysis

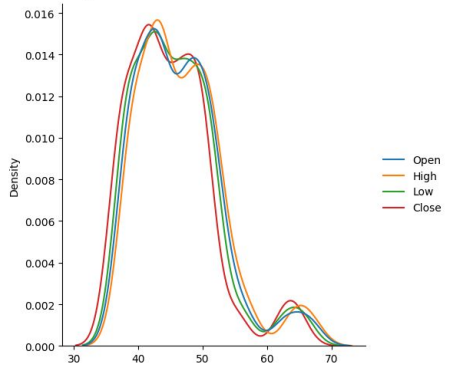
Density Plot Summary (Open, High, Low, Close):

A Kernel Density Estimation (KDE) plot was generated for the stock prices: Open, High, Low, and Close.

The curves show that:
Most stock prices are concentrated between approximately $38 and $50.
All four features (Open, High, Low, Close) have very similar distributions, which is expected since these prices are closely related within daily trading.
The High prices tend to be slightly higher than the Open and Close prices, as indicated by a slight shift to the right.
The Low prices are slightly lower than the Open and Close prices, which also makes sense.
There are minor bumps around $60–65, indicating a smaller set of higher stock price days during the period.

```
sns.displot(data=stock[['Open', 'High', 'Low', 'Close']], kind="kde", palette="tab10")
```

```
<seaborn.axisgrid.FacetGrid at 0x7b3efc076f10>
```



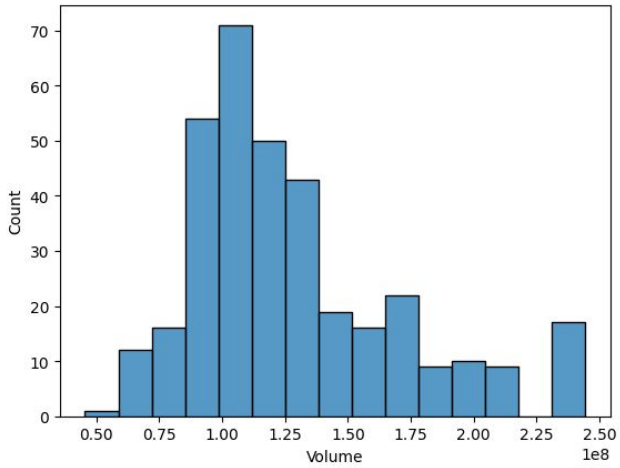
Note:
The similarity in distributions between Open, High, Low, and Close prices suggests stable daily trading patterns without extreme fluctuations most of the time.

EDA Results - Univariate analysis

Histogram Summary (Volume):

A histogram of the Volume column shows the distribution of daily trading volumes.
Most trading days had a volume between approximately 90 million and 140 million shares.
The distribution is right-skewed (positively skewed), meaning there are some days with very high trading volumes (above 200 million shares), but these are relatively rare.
The most frequent trading volume range appears to be around 100–120 million shares.

```
sns.histplot(stock, x='Volume');
```



Note:
The trading volume for the stock was typically moderate, with occasional days of very high activity, likely triggered by major news or market events.

EDA Results - Univariate analysis

News Content Length:

Total entries: 349 news articles.

Mean (average) number of words: About 49.31 words per news article.

Standard deviation: About 5.73, indicating that most articles have word counts fairly close to the mean.

Minimum number of words: 19 words (the shortest article).

Maximum number of words: 61 words (the longest article).

Percentiles:
25th percentile: 46 words ➔ 25% of articles have 46 words or fewer.
50th percentile (median): 50 words ➔ Half of the articles have 50 words or fewer.
75th percentile: 53 words ➔ 75% of articles have 53 words or fewer.

```
#Calculating the total number of words present in the news content.
stock['news_len'] = stock['News'].apply(lambda x: len(x.split(' ')))


stock['news_len'].describe()
```

	news_len
count	349.000000
mean	49.312321
std	5.727770
min	19.000000
25%	46.000000
50%	50.000000
75%	53.000000
max	61.000000

dtype: float64

Note:
The news articles are relatively consistent in length, with most articles containing between 46 and 53 words.
There are no extremely short or extremely long articles, which suggests that the data is clean and fairly standardized in terms of content size.

EDA Results - Univariate analysis

News length analysis:

The distribution of the number of words in the news articles is fairly concentrated.
Most articles have between 45 and 55 words, with the peak around 50 words.
The histogram reveals a slightly left-skewed distribution, indicating a few shorter articles (around 20–40 words), but the majority are clustered closer to the mean.

Only a very small number of articles have extremely short (under 40 words) or very long (above 60 words) lengths

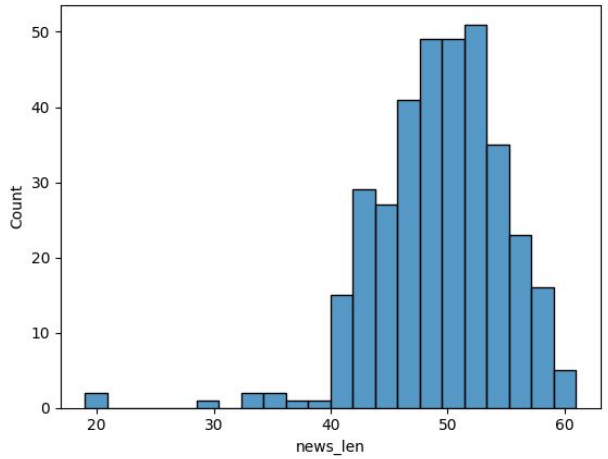
Note:
The news content is relatively standardized in length, which simplifies model training.

```
#Calculating the total number of words present in the news content.
stock['news_len'] = stock['News'].apply(lambda x: len(x.split(' ')))


stock['news_len'].describe()
```

	news_len
count	349.000000
mean	49.312321
std	5.727770
min	19.000000
25%	46.000000
50%	50.000000
75%	53.000000
max	61.000000

dtype: float64

```
sns.histplot(data=stock,x="news_len");
```



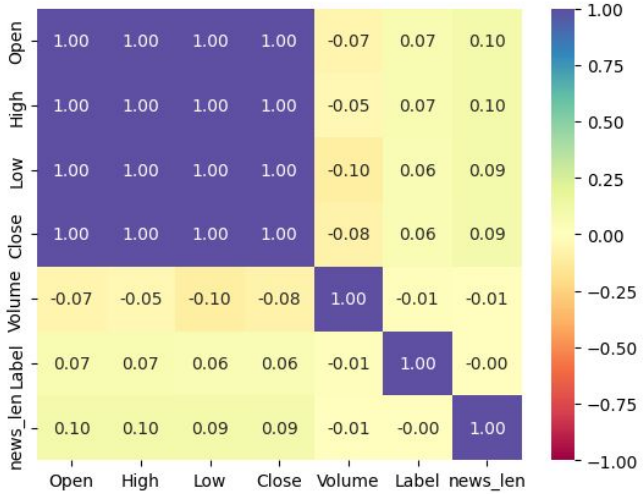
EDA Results - Bivariate analysis - Correlation Matrix

Strong positive correlations are observed between the stock price-related columns: Open, High, Low, and Close are perfectly correlated with each other (correlation ≈ 1.00). This makes sense because these prices move together throughout the trading day.

Volume:
Has weak negative correlations with the stock prices (Open, High, Low, Close), all around -0.05 to -0.10. This suggests that the amount of shares traded is slightly independent from daily price changes.

Label (Sentiment):
Shows very weak correlations with stock prices and volume (values close to 0). Sentiment (Label) is therefore not strongly linearly correlated with immediate price or trading volume changes.

news_len (number of words in news articles):
Has a small positive correlation (around 0.09–0.10) with stock prices.
This indicates that slightly longer news articles might be associated with higher prices, but the effect is very minor.



Note:
Stock prices are tightly linked among themselves but are relatively independent from trading volume, sentiment labels, and news length.
Sentiment may still impact stock prices, but likely in a nonlinear or more complex way that simple correlation does not capture.

EDA Results - Bivariate analysis

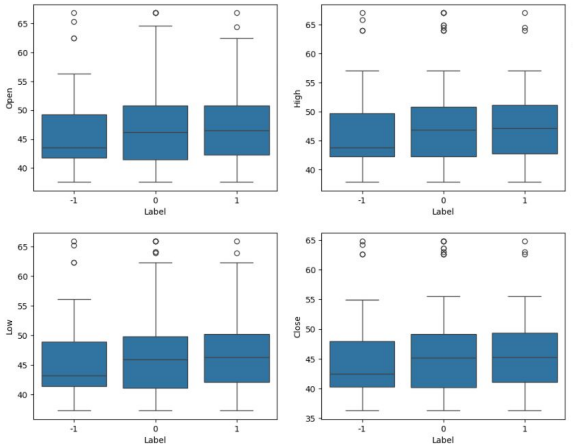
Boxplot Summary: Stock Prices vs. Sentiment Labels

General observation:
The distributions of Open, High, Low, and Close prices are very similar across the three sentiment categories:
-1 (Negative)
0 (Neutral)
1 (Positive)

Medians:
The median stock prices (central lines in each box) are slightly higher for positive sentiment (Label = 1) compared to negative sentiment (Label = -1), but the difference is small.

Spread (IQR):
The interquartile ranges (the width of each box) are mostly similar across all sentiment classes.
This suggests that price variability is not heavily influenced by the sentiment labels.

Outliers:
A few outliers exist for each label, especially in High and Low prices.
These outliers are relatively evenly distributed and do not seem tied specifically to any sentiment class.



Note:
While positive sentiment days might slightly correspond to higher stock prices, the overall distributions are largely similar across all sentiment types. This indicates that stock prices are relatively stable regardless of minor daily sentiment fluctuations.

EDA Results - Bivariate analysis

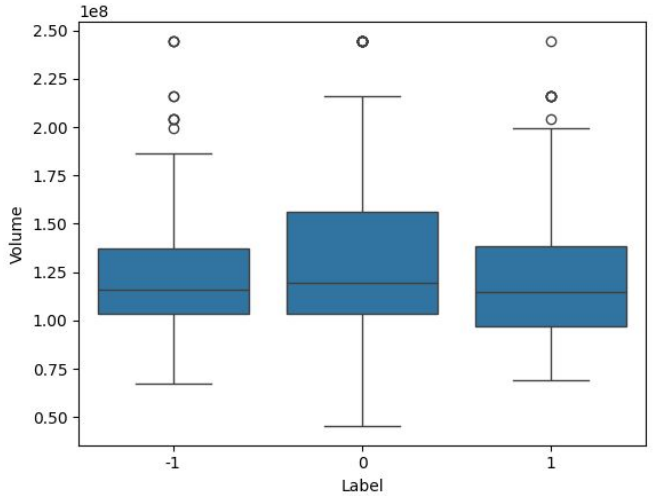
Boxplot Summary: Volume vs. Sentiment Labels:

Volume distribution across negative (-1), neutral (0), and positive (1) sentiment labels is fairly similar overall.

Medians (the center line in each box) are close across the three sentiment categories, around 110 to 125 million shares.

Spread (Interquartile Range, IQR) is slightly wider for neutral sentiment (0) compared to negative (-1) and positive (1).

Outliers (the small circles) are present in all three categories: Especially for neutral and positive sentiments, indicating some very high volume days regardless of sentiment.



Note:
The trading volume does not vary significantly based on sentiment.
Even though neutral news might have slightly higher variability in volume, overall, market activity (volume) remains relatively stable regardless of whether news is negative, neutral, or positive.

EDA Results - Bivariate analysis

Summary of the Grouped Daily Stock Data:

The columns show the average values per day:
Open: Average opening price for that day.
High: Average of the highest prices recorded that day.
Low: Average of the lowest prices recorded that day.
Close: Average closing price that day.
Volume: Average number of shares traded that day.

Setting Date as index:
Setting Date as the index, which will make time series analysis easier (like plotting trends over time).

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

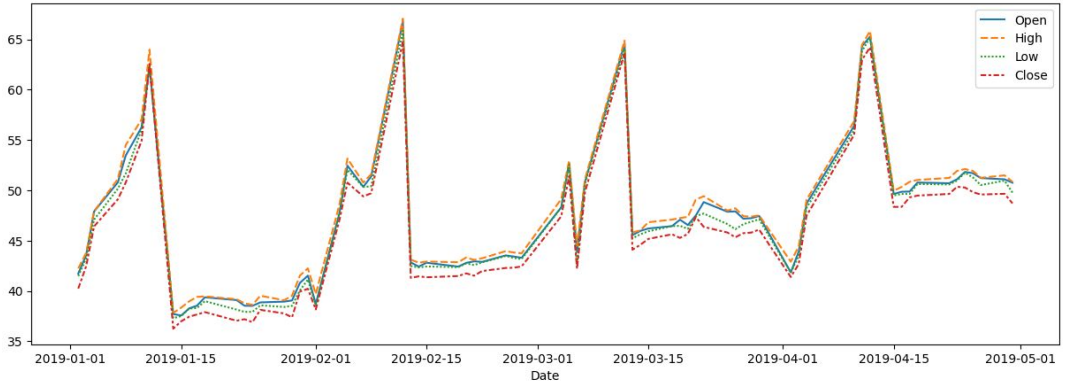
Date	Open	High	Low	Close	Volume
2019-01-02	41.740002	42.244999	41.482498	40.246914	130672400.0
2019-01-03	43.570000	43.787498	43.222500	42.470604	103544800.0
2019-01-04	47.910000	47.919998	47.095001	46.419842	111448000.0
2019-01-07	50.792500	51.122501	50.162498	49.110790	109012000.0
2019-01-08	53.474998	54.507500	51.685001	50.787209	216071600.0

Note:
The dataset has now been resampled to a daily level, aggregating multiple news articles into a single, clean daily view of stock prices and volume. This prepares your data for weekly summaries, trend plots, and for integrating weekly sentiment analysis later!

EDA Results - Bivariate analysis

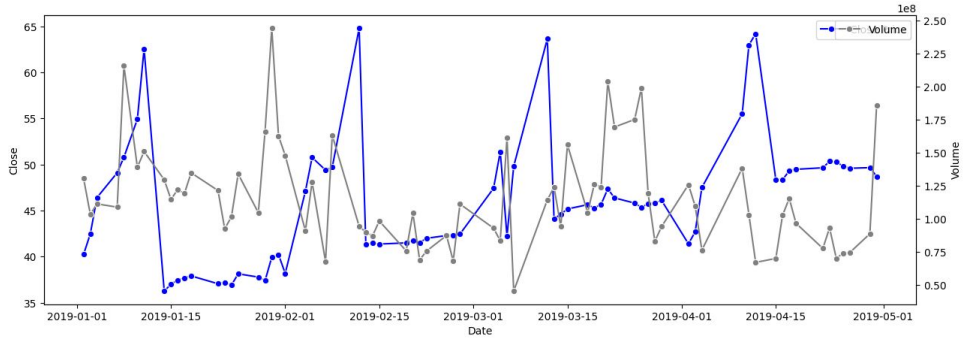
Line Plot Summary: Daily Stock Prices (Open, High, Low, Close)

The line plot shows the daily trends of Open, High, Low, and Close stock prices over time.
All four price variables move very closely together, reflecting typical daily market behavior (opening, highest, lowest, and closing prices being related).
Repeated spikes are visible, approximately every few weeks, where prices increase sharply and then drop — indicating periodic volatility or market events affecting the stock.
High prices (dashed orange line) are consistently slightly above Open and Close prices, while Low prices (dotted green line) are slightly below, which is expected in daily trading.
After each spike, prices return to a more stable and lower range before rising again.



Note:
The stock exhibits regular volatility patterns with significant short-term rises and falls, but overall, the Open, High, Low, and Close prices are tightly correlated and behave predictably relative to each other.

EDA Results - Bivariate analysis



Line Plot Summary: Close Price vs Volume Over Time

The plot shows two lines over the same time period:

Blue line: Close price of the stock over time (left y-axis).

Gray line: Trading volume of the stock over time (right y-axis).

Close price behavior:

The blue line shows several sharp spikes where the stock's closing price dramatically increases and then quickly drops back down.

These spikes are spaced fairly regularly, suggesting periodic events affecting stock value.

Volume behavior:

The gray line for trading volume is more volatile with frequent ups and downs.

Some volume spikes coincide with the sharp increases in Close price, suggesting that higher trading activity often happens during major price movements.

Overall relationship:

There seems to be a weak positive relationship: when trading volume increases, closing prices sometimes spike as well, although it is not perfectly consistent.

Note:

Stock price spikes are often accompanied by increased trading activity, although not every volume peak results in a significant change in closing price.

This indicates that while volume can be a signal of market interest, it doesn't always directly cause major price changes.

EDA Results - Feature Engineering

- A new feature called news_len was created by counting the number of words in each news article.
- This helped in understanding the text structure and ensuring that news content length was fairly consistent.
- news_len was later used as an additional feature during model training, increasing the total number of features from 8 to 10.
- After dropping the target column (Label) from feature sets (done later during data preprocessing), the final number of features used for modeling was 9.

```
] #Calculating the total number of words present in the news content.
  stock['news_len'] = stock['News'].apply(lambda x: len(x.split(' ')))


  stock['news_len'].describe()
```

	news_len
count	349.000000
mean	49.312321
std	5.727770
min	19.000000
25%	46.000000
50%	50.000000
75%	53.000000
max	61.000000

dtype: float64

Data Preprocessing

Great Learning
POWER AHEAD

Splitting by Date:
The dataset was split chronologically based on the 'Date' column:

- Training set: Data before 1st April 2019
- Validation set: 1st April 2019 to 15th April 2019
- Test set: 16th April 2019 and onwards

```
#pick the 'Label' column as the target variable
y_train = X_train["Label"].copy()
y_val = X_val["Label"].copy()
y_test = X_test["Label"].copy()
```

```
# Remove 'Label' from features
X_train = X_train.drop(columns=["Label"])
X_val = X_val.drop(columns=["Label"])
X_test = X_test.drop(columns=["Label"])
```

```
stock["Date"].describe()
```

	Date
count	349
mean	2019-02-16 16:05:30.085959936
min	2019-01-02 00:00:00
25%	2019-01-14 00:00:00
50%	2019-02-05 00:00:00
75%	2019-03-22 00:00:00
max	2019-04-30 00:00:00

dtype: object

Target and Feature Separation:
The 'Label' column (indicating sentiment) was extracted separately into y_train, y_val, and y_test as the target variable.

- Feature Cleanup:
  After separating the target, the 'Label' column was removed from the feature datasets (X_train, X_val, X_test) to avoid data leakage during model training.

```
X_train = stock[(stock['Date'] < '2019-04-01')].reset_index()    #select all rows where the 'D
X_val = stock[(stock['Date'] >= '2019-04-01') & (stock['Date'] < '2019-04-16')].reset_index()
X_test = stock[stock['Date'] >= '2019-04-16'].reset_index()    #select all rows where the 'Dat
```

Why this matters:
Chronologically splitting and properly separating features and targets ensures realistic model evaluation and prevents information leakage that could falsely inflate model performance.

Data Preprocessing

Dataset Summary (After Label Separation and feature engineering - done earlier):

Features: 9 columns

X_train: 286 samples → y_train: 286 labels

X_val: 21 samples → y_val: 21 labels

X_test: 42 samples → y_test: 42 labels

Split Proportions:

~82% training, ~6% validation, ~12% testing

➔ Well-balanced for model development.

```
#Calculating the total number of words present in the news content.
stock['news_len'] = stock['News'].apply(lambda x: len(x.split(' ')))
```

```
] #print the shape of X_train,X_val,X_test,y_train,y_val and y_test
  print("Train data shape",X_train.shape)
  print("Validation data shape",X_val.shape)
  print("Test data shape ",X_test.shape)

  print("Train label shape",y_train.shape)
  print("Validation label shape",y_val.shape)
  print("Test label shape ",y_test.shape)
```

```
Train data shape (286, 9)
Validation data shape (21, 9)
Test data shape  (42, 9)
Train label shape (286,)
Validation label shape (21,)
Test label shape  (42,)
```

Sentiment Analysis - Model Evaluation Criterion

Chosen Criterion:
F1-Score as the primary evaluation metric.

Reason for Choice:

- The dataset was a little imbalanced across the three sentiment classes (-1, 0, 1).
- Accuracy alone could be misleading since predicting the majority class could still result in high accuracy.
- F1-Score balances precision and recall, making it better suited for understanding the model's true performance across all classes.

Why F1?
A model with a higher F1-Score indicates a better balance between catching positive/negative events correctly (recall) and ensuring that predicted events are actually correct (precision).
In this case, the Sentence Transformer model on the original dataset achieved the highest F1-Score (~0.48), thus selected as the final model.

Sentiment Analysis - Model Building

Overview of the models

Three different base models based on different text embeddings were built:

- Word2Vec (vec_size 300, custom trained on our dataset)
- GloVe (100 features, pretrained on external corpus)
- Sentence Transformer (384 features, pretrained, sentence-level encoding)

Each embedding set was used to train a Random Forest Classifier with default parameters.

Both balanced and original versions of the datasets were used during training for comparative evaluation.

Word2Vec

```
# Creating an instance of Word2Vec
vec_size = 300
model_W2V = Word2Vec(words_list, vector_size = vec_size, min_count = 1, window=5, workers = 6)
```

```
# creating a dataframe of the vectorized documents
start = time.time()

X_train_wv = pd.DataFrame(X_train["News"].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+str(i) for i in range(vec_size)])
X_val_wv = pd.DataFrame(X_val["News"].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+str(i) for i in range(vec_size)])
X_test_wv = pd.DataFrame(X_test["News"].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+str(i) for i in range(vec_size)])

end = time.time()
print('Time taken ', (end-start))
```

```
Time taken  0.4666874408721924
```

GloVe

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip -d glove_embeddings/

from gensim.scripts.glove2word2vec import glove2word2vec

# Konvertera GloVe-fil till Word2Vec-format
glove_input_file = 'glove_embeddings/glove.6B.100d.txt' #
word2vec_output_file = 'glove.6B.100d.word2vec.txt'
glove2word2vec(glove_input_file, word2vec_output_file)
```

Sentence Transformer

```
#Defining the model
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

Sentiment Analysis - Model Building - Base Model Word2Vec

In this step, multiple base classification models was trained using Word2Vec vectorized news data to predict stock movement labels.

Specifically, three traditional machine learning models were applied:
Gradient Boosting - Random Forest - Decision Tree

This was done to investigate the impact of class imbalance on model performance, I trained each model using the original data, and for Random Forest and Decision Tree, I also trained additional versions with class_weight='balanced'. This allows the model to assign more weight to underrepresented classes, which is important given the imbalance in label distribution (especially fewer positive and negative cases compared to neutral).

```
# Word2Vec – Gradient Boosting
model_wv_gb = GradientBoostingClassifier(random_state=42)
model_wv_gb.fit(X_train_wv, y_train)

# Word2Vec – Random Forest
model_wv_rf = RandomForestClassifier(random_state=42)
model_wv_rf.fit(X_train_wv, y_train)

# Word2Vec – Decision Tree
model_wv_dt = DecisionTreeClassifier(random_state=42)
model_wv_dt.fit(X_train_wv, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
#with balanced dataset
# Word2Vec – Random Forest
model_wv_rf_bal = RandomForestClassifier(random_state=42, class_weight='balanced')
model_wv_rf_bal.fit(X_train_wv, y_train)

# Word2Vec – Decision Tree
model_wv_dt_bal = DecisionTreeClassifier(random_state=42, class_weight='balanced')
model_wv_dt_bal.fit(X_train_wv, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

Note:
This setup establishes a fair comparison between standard vs. class-balanced training for each model, helping evaluate whether addressing imbalance improves predictive performance.

Sentiment Analysis - Model Building - Base Model Word2Vec

Training Data:
All models (Gradient Boosting, Random Forest, Decision Tree – both balanced and unbalanced) achieved 100% on Accuracy, Recall, Precision, and F1 on the training data.
➔ Indicates overfitting – the models fit the training data perfectly.

Validation Data:
Results on the validation data show differences:

- Gradient Boosting: F1 = 0.39
- Random Forest (unbalanced): F1 = 0.30
- Random Forest (balanced): F1 = 0.35
- Decision Tree (unbalanced): F1 = 0.50
- Decision Tree (balanced): F1 = 0.48

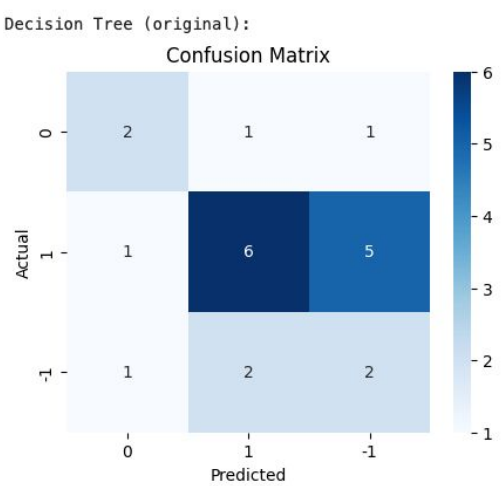
```
Gradient Boosting:
   Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0

Random Forest:
   Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0

Random Forest (balanced):
   Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0

Decision Tree:
   Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0

Decision Tree (balanced):
   Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

```
Gradient Boosting (validation):
   Accuracy    Recall  Precision        F1
0  0.428571  0.428571   0.352381  0.386243

Random Forest (validation):
   Accuracy    Recall  Precision        F1
0  0.333333  0.333333   0.266667  0.296296

Random Forest (balanced) (validation):
   Accuracy    Recall  Precision        F1
0  0.428571  0.428571   0.302521   0.35468

Decision Tree (validation):
   Accuracy   Recall  Precision        F1
0   0.47619  0.47619   0.535714  0.495029

Decision Tree (balanced) (validation):
   Accuracy   Recall  Precision        F1
0   0.47619  0.47619   0.564626  0.47619
```

Note:

The unbalanced Decision Tree performed the best on the validation data with the highest F1 score (0.50), suggesting better generalization compared to the other models.

Sentiment Analysis - Model Building - Base Model Word2Vec



Confusion Matrix for Base Model: Word2Vec + Decision Tree (unbalanced/original dataset) - best performance, see slide before.

Overall: The model is *partially* able to distinguish between the classes 0, 1, and -1 — but several misclassifications still occur.

What we observe: For class **1** (positive events), the model correctly predicts 6 out of 12 examples — the best performance among the classes.

Classes **0** (neutral) and **-1** (negative) are more challenging, with frequent misclassifications, especially between 0 and -1.

There is a reasonable *spread* across the predicted classes, which is a positive sign — the model is not stuck predicting only one class.

Note: The model is learning signals from the data, but the signals are weak, leading to confusion between classes.

This is consistent with the validation F1-score (F1 = 0.50 for the unbalanced Decision Tree) — decent, but leaves room for improvement.

The confusion matrix indicates better-than-random performance, but also highlights that stronger embeddings or more powerful models could significantly improve classification

Sentiment Analysis - Model Building - Base Model Word2Vec - Validation and Sanity Check

Since each model showed identical Accuracy and Recall scores *within itself* (although the scores differed between models), I decided to perform a Validation and Sanity Check to ensure there were no underlying issues.

Class Balance Check:

- Inspected the distribution of labels (-1, 0, 1) across training, validation, and test datasets.
- Verified that there was no extreme class imbalance that could distort model performance.

Embeddings Quality Check (Word2Vec):

- Calculated mean and standard deviation of the Word2Vec features.
- Observed a very low standard deviation (~0.000286), indicating very flat embeddings with little variance across documents.

```
print(y_train.value_counts(normalize=True))
print(y_val.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))
```

```
Label
 0    0.482517
-1    0.286713
 1    0.230769
Name: proportion, dtype: float64
Label
 0    0.571429
 1    0.238095
-1    0.190476
Name: proportion, dtype: float64
Label
 0    0.476190
-1    0.309524
 1    0.214286
Name: proportion, dtyp
```

```
# Kolla hur dina Word2Vec-features ser ut
print("Shape of X_train_wv:", X_train_wv.shape)

# Visa en snabb summering
print("Mean of X_train_wv:", np.mean(X_train_wv))
print("Std of X_train_wv:", np.std(X_train_wv))

# Visa ett par exempel
print("First 5 feature rows:\n", X_train_wv[:5])
```

Sentiment Analysis - Model Building - Base Model Word2Vec - Validation and Sanity Check

Great Learning
POWER AHEAD

Text Content Review:

- Manually checked original news texts.
- Confirmed that the texts were long, informative, and contextually rich, with no [UNK] tokens or noise after tokenization.

Token Count Validation:

- Computed the average number of tokens per news article.
- Result: ~48 tokens per article, which is an ideal length for meaningful embedding generation.

```
# Visa några exempel från X_train innan embedding
print("Exempel på originaltext (innan embedding):")
for i in range(5):
    print(X_train['News'][i])
    print("-----")
```

```
Exempel på originaltext (innan embedding):
 The tech sector experienced a significant decline in the aftermarket followi
-----
 Apple lowered its fiscal Q1 revenue guidance to $84 billion from earlier est
-----
 Apple cut its fiscal first quarter revenue forecast from $89-$93 billion to
-----
 This news article reports that yields on long-dated U.S. Treasury securities
-----
 Apple's revenue warning led to a decline in USD JPY pair and a gain in Japan
-----
```

```
# Om X_train_wv skapades från t.ex. medelvärde av embeddings, så är det bra att kolla token counts
print("Genomsnittligt antal tokens per text:")
# Accessing 'News' column in X_train to get raw text data
token_counts = [len(text.split()) for text in X_train['News']]
print(sum(token_counts) / len(token_counts))
```

```
Genomsnittligt antal tokens per text:
48.14685314685315
```

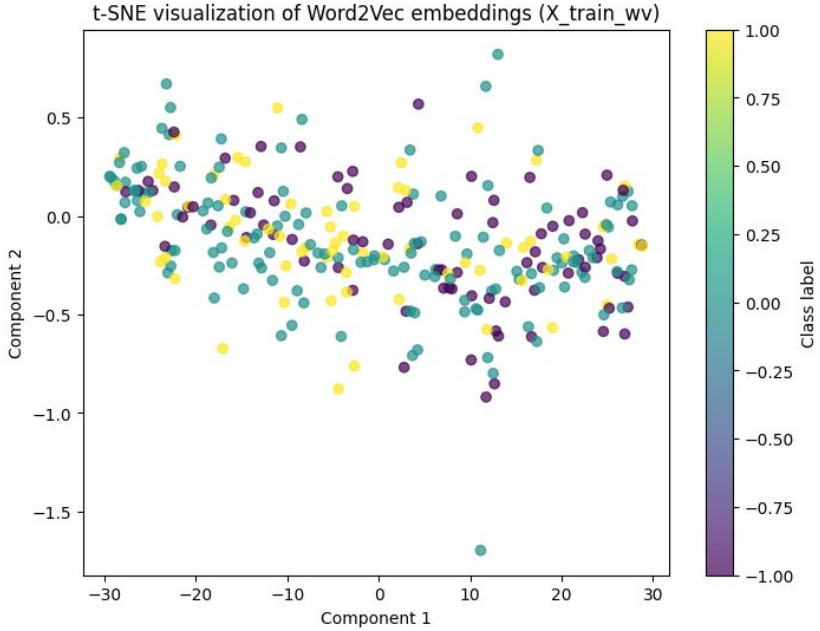
Sentiment Analysis - Model Building - Base Model Word2Vec - Validation and Sanity Check

t-SNE Visualization of Embeddings:

- Reduced dimensionality of Word2Vec embeddings to 2D using t-SNE.
- Observed heavy overlap between classes, suggesting that embeddings were not well-separated and making classification more difficult.

Prediction Distribution Analysis:

- Checked the prediction outputs of a trained Random Forest model.
- Verified that the model did not always predict the same class, meaning it attempted to differentiate between classes even if performance was moderate.




```
Predictions distribution on validation set:
Class -1: 2 samples
Class 0: 15 samples
Class 1: 4 samples
```

Sentiment Analysis - Model Building - Base Model Word2Vec - Validation and Sanity Check

Final Sanity Checks:

- Ensured there were no NaN values in features or labels.
- Confirmed presence of all three classes (-1, 0, 1) in the training set.
- Ran summary statistics to verify no abnormal feature behavior after embedding.

```
[64] # Kontroll 1: Form
     print("X_train_wv shape:", X_train_wv.shape)
     print("y_train shape:", y_train.shape)

     # Kontroll 2: Några NaN i X_train_wv?
     print("\nAny NaN values in X_train_wv?", np.isnan(X_train_wv).sum())

     # Kontroll 3: Några NaN i y_train?
     print("Any NaN values in y_train?", y_train.isnull().sum())

     # Kontroll 4: Unika klasser i y_train
     print("\nUnique classes in y_train:", y_train.unique())

     # Kontroll 5: Snabb statistik på X_train_wv
     print("\nSummary statistics of X_train_wv:")
     print(pd.DataFrame(X_train_wv).describe())
```

Sentiment Analysis - Model Building - Base Model Word2Vec - Validation and Sanity Check

So Where Is the Problem?

Since I have verified that:

- The embeddings are acceptable
- The target variable distribution is reasonable
- The evaluation function works correctly
  ➔ Everything points to the models not having learned enough yet.

Possible reasons:

- Too few training examples (286 samples are relatively small compared to 300-dimensional embeddings).
- The data is not highly separable (as seen in the t-SNE plot — classes are mixed and not well-clustered).
- Simple model architectures (e.g., Decision Trees without deep tuning).

Note:
All F1 scores differ somewhat — meaning that the models are learning something, even if the overall performance is moderate.
Therefore, it's not a critical failure — but model performance is limited by data size, data separability, and model complexity.

Sentiment Analysis - Model Building - Base Model GloVe

In this step, I evaluated the performance of three different classifiers using GloVe word embeddings:

1. Original Models (no class weights)
Gradient Boosting, Random Forest, and Decision Tree classifiers were trained on the vectorized GloVe training data (X_train_gl, y_train) without any class balancing.
These models served as a baseline to understand how well the models perform without handling class imbalance.

2. Balanced Models (with class weights)
Since Random Forest and Decision Tree support the class_weight='balanced' parameter, I retrained these models with this setting.
The goal was to address the class imbalance in the target variable by adjusting the model to pay proportionally more attention to underrepresented classes.

```
# GloVe – Original Models (no class weights)

# GloVe – Gradient Boosting
model_gl_gb = GradientBoostingClassifier(random_state=42)
model_gl_gb.fit(X_train_gl, y_train)

# GloVe – Random Forest
model_gl_rf = RandomForestClassifier(random_state=42)
model_gl_rf.fit(X_train_gl, y_train)

# GloVe – Decision Tree
model_gl_dt = DecisionTreeClassifier(random_state=42)
model_gl_dt.fit(X_train_gl, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
[68] # GloVe – With Balanced Class Weights

# GloVe – Random Forest (balanced)
model_gl_rf_bal = RandomForestClassifier(random_state=42, class_weight='balanced')
model_gl_rf_bal.fit(X_train_gl, y_train)

# GloVe – Decision Tree (balanced)
model_gl_dt_bal = DecisionTreeClassifier(random_state=42, class_weight='balanced')
model_gl_dt_bal.fit(X_train_gl, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

Note:
Gradient Boosting does not natively support class_weight, so no balanced version was created for it in this phase.
By training both original and balanced versions of these models, I prepared a robust comparison framework to analyze the effect of class imbalance on model performance.

Sentiment Analysis - Model Building - Base Model GloVe

Training Performance (Overfitting Indicated)

All models, including those with and without class_weight='balanced', achieved perfect scores on the training set:

Accuracy, Recall, Precision, and F1 Score were all 1.0 This suggests the models learned the training data completely — which may indicate overfitting.

Validation Performance (Generalization Check)

Performance on the validation set revealed more realistic results, where Random Forest with class_weight balanced performed the best with a F1-score of 48%.

Gradient Boosting:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Random Forest:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Decision Tree:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Random Forest (balanced):

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Decision Tree (balanced):

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Gradient Boosting (validation):

	Accuracy	Recall	Precision	F1
0	0.380952	0.380952	0.326531	0.351648

Random Forest (validation):

	Accuracy	Recall	Precision	F1
0	0.47619	0.47619	0.400794	0.426871

Decision Tree (validation):

	Accuracy	Recall	Precision	F1
0	0.428571	0.428571	0.458503	0.438672

Random Forest (balanced) (validation):

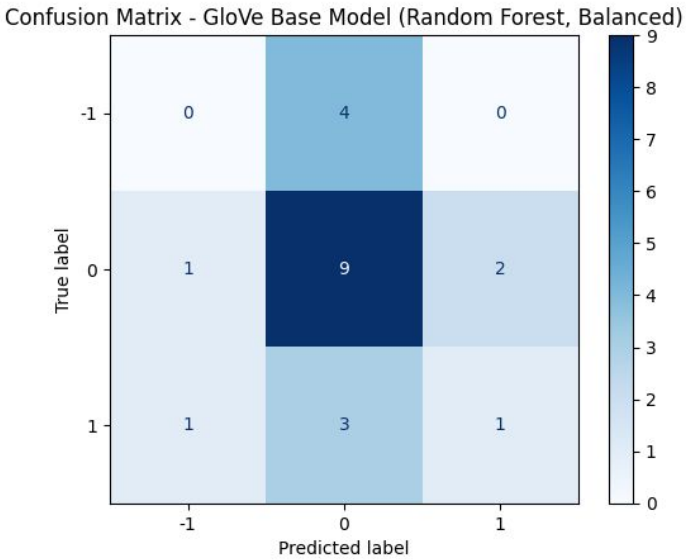
	Accuracy	Recall	Precision	F1
0	0.571429	0.571429	0.444444	0.48254

Decision Tree (balanced) (validation):

	Accuracy	Recall	Precision	F1
0	0.285714	0.285714	0.345238	0.301832

Note: Random Forest with class weights (balanced) gave the best validation results in terms of F1 score. Gradient Boosting struggled the most to generalize. Balancing helped Random Forest but did not help Decision Tree in this case. The gap between training and validation metrics confirms overfitting and suggests the need for regularization or more data.

Sentiment Analysis - Model Building - Base Model GloVe



Class 0 performs best:
9 out of 12 examples were correctly classified → ~75% accuracy for class 0.

Class -1 is often missed:
All real -1 examples are predicted as class 0 ➔ the model struggles to distinguish -1 from 0.

Class 1 is difficult:
Only 1 out of 5 examples were classified correctly → the model tends to confuse class 1 with class 0.

Visible Problems:

Bias toward class 0: The model often predicts class 0.

Difficulty separating -1 and 0, as well as 1 and 0: Likely due to overlapping feature embeddings (as we saw in the t-SNE visualization).

Note:
Training challenge despite class balancing: Even with class_weight='balanced', it can be difficult if the embeddings for different classes are too similar.

Sentiment Analysis - Model Building - Base Model GloVe - Sanity check

The reason for performing this sanity check on GloVe embeddings is the same as for Word2Vec: to ensure that the features are properly distributed and free from issues before training, given earlier concerns about suspicious model performance patterns.

Objective:
Ensure that the GloVe-generated embeddings (X_train_gl) are healthy and suitable for model training.

```
# Kontrollera shape och några exempel
print("Shape of X_train_gl:", X_train_gl.shape)
print("First 5 feature rows (GloVe):")
print(pd.DataFrame(X_train_gl).head())

# Kontrollera NaN
print("\nAny NaN values in X_train_gl?", pd.DataFrame(X_train_gl).isna().sum().sum())
print("Any NaN values in y_train?", y_train.isna().sum())

# Kontrollera unika klasser
print("\nUnique classes in y_train:", np.unique(y_train))

# Kontrollera grundläggande statistik
print("\nSummary statistics of X_train_gl:")
print(pd.DataFrame(X_train_gl).describe())
```

Sentiment Analysis - Model Building - Base Model GloVe - Sanity check

Steps Taken:

- Checked Shape:
  - 286 samples × 100 features → Correct as expected after embedding.
- Checked for Missing Values (NaN):
  - No missing values in X_train_gl or y_train.
- Checked Class Distribution:
  - y_train contains all three classes: -1, 0, 1.

Distribution Analysis:

- Mean Values:
  - Small (around ±0.01) → Normal for pre-trained embeddings.
- Standard Deviation:
  - Good spread (~0.07–0.1), indicating non-flat embeddings.
- Min/Max Values:
  - Within expected range, no extreme outliers.

Conclusion: GloVe embeddings are clean, well-distributed, and ready for model training. No additional data cleaning needed.

So Where Is the Problem?

The GloVe embeddings passed all quality checks: no missing values, normal distribution, and reasonable spread of features.
However, despite good embedding quality, the model struggled to distinguish between classes (-1, 0, 1). This suggests that the issue may lie in class overlap in feature space, as seen in t-SNE visualization — especially between neutral (0) and the other classes. Thus, the embeddings are clean, but they may not carry enough separability for perfect classification.

Sentiment Analysis - Model Building - Base Model - Sentence Transformer

In this step, I trained classification models using sentence embeddings generated from a pre-trained Sentence Transformer (all-MiniLM-L6-v2). The goal was to evaluate how well different classifiers perform on this semantic representation of news text data.

I explored the following models:

Gradient Boosting Random Forest Decision Tree

For each model, I trained:

Original version – using the default setup without any class weighting.

Balanced version – with class_weight='balanced' applied to handle class imbalance in the dataset.

By comparing both versions, I aimed to assess whether balancing class weights could improve model performance, particularly recall and F1-score for underrepresented sentiment classes.

```
# Sentence Transformer – Original Models (no class weights)

# Sentence Transformer – Gradient Boosting
model_st_gb = GradientBoostingClassifier(random_state=42)
model_st_gb.fit(X_train_st, y_train)

# Sentence Transformer – Random Forest
model_st_rf = RandomForestClassifier(random_state=42)
model_st_rf.fit(X_train_st, y_train)

# Sentence Transformer – Decision Tree
model_st_dt = DecisionTreeClassifier(random_state=42)
model_st_dt.fit(X_train_st, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
# Sentence Transformer – With Balanced Class Weights

# Sentence Transformer – Random Forest (balanced)
model_st_rf_bal = RandomForestClassifier(random_state=42, class_weight='balanced')
model_st_rf_bal.fit(X_train_st, y_train)

# Sentence Transformer – Decision Tree (balanced)
model_st_dt_bal = DecisionTreeClassifier(random_state=42, class_weight='balanced')
model_st_dt_bal.fit(X_train_st, y_train)
```

```
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

Note:
Gradient Boosting does not natively support class_weight, so no balanced version was created for it in this phase.
By training both original and balanced versions of these models, I prepared a robust comparison framework to analyze the effect of class imbalance on model performance.

Sentiment Analysis - Model Building - Base Model - Sentence Transformer

In this step I trained and evaluated three classifiers just like the earlier models:

Gradient Boosting, Random Forest, and Decision Tree — both with and without class balancing using class_weight='balanced'.

Training Performance: All models, regardless of classifier or balancing strategy, achieved perfect scores on the training set. This indicates overfitting.

Validation Performance: Random Forest (no class weights) outperformed other models in all metrics on the validation set.

Gradient Boosting:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Random Forest:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Decision Tree:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Random Forest (balanced):

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Decision Tree (balanced):

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Gradient Boosting (validation):

	Accuracy	Recall	Precision	F1
0	0.47619	0.47619	0.300752	0.368664

Random Forest (validation):

	Accuracy	Recall	Precision	F1
0	0.619048	0.619048	0.533333	0.504762

Decision Tree (validation):

	Accuracy	Recall	Precision	F1
0	0.52381	0.52381	0.443537	0.48026

Random Forest (balanced) (validation):

	Accuracy	Recall	Precision	F1
0	0.47619	0.47619	0.300752	0.368664

Decision Tree (balanced) (validation):

	Accuracy	Recall	Precision	F1
0	0.47619	0.47619	0.515646	0.49062

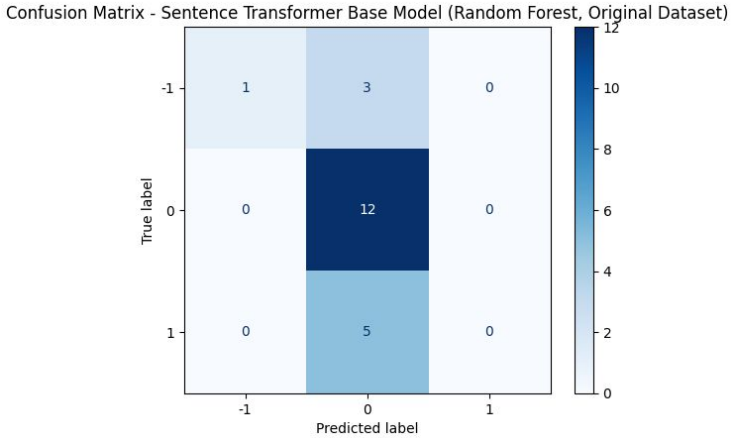
Note: Applying class_weight='balanced' did not significantly improve the results for this embedding method — in fact, it sometimes slightly reduced accuracy and recall. The gap between training and validation scores reaffirms the presence of overfitting in all models.

Sentiment Analysis - Model Building - Base Model - Sentence Transformer

The model is good at recognizing Class 0 (neutral sentiment) → 12 out of 12 samples correctly classified! However, the model struggles to distinguish Class -1 and Class 1 from Class 0:

- Negative samples (-1) are often confused with neutral (0).
- Positive samples (1) are also confused with neutral (0).

The model shows a bias toward predicting neutral sentiment, even when the true sentiment is positive or negative.



Note:
The Sentence Transformer embeddings are not bad – they successfully capture useful signals for the neutral class.
However, the small number of samples and imbalanced classes might cause the model to "play it safe" and avoid predicting -1 or 1.

Sentiment Analysis - Model Building - Base Model Sentence Transformer - Sanity check

Following the same approach as for Word2Vec and GloVe, a sanity check was performed to ensure the quality and structure of the Sentence Transformer embeddings before model training. This was important because earlier model metrics showed unusual patterns (e.g., identical accuracy and recall within models).

```
# Kontrollera shape och några exempel
print("Shape of X_train_st:", X_train_st.shape)
print("First 5 feature rows (Sentence Transformer):")
print(pd.DataFrame(X_train_st).head())

# Kontrollera NaN
print("\nAny NaN values in X_train_st?", pd.DataFrame(X_train_st).isna().sum().sum())
print("Any NaN values in y_train?", y_train.isna().sum())

# Kontrollera unika klasser
print("\nUnique classes in y_train:", np.unique(y_train))

# Kontrollera grundläggande statistik
print("\nSummary statistics of X_train_st:")
print(pd.DataFrame(X_train_st).describe())
```

Sentiment Analysis - Model Building - Base Model Sentence Transformer - Sanity check

Checks Performed:

- Shape and Sample Rows:
  Verified that each sample (news article) was correctly transformed into a 384-dimensional feature vector. No structural issues were detected.
- Missing Values:
  Confirmed that there are no missing (NaN) values in either the feature set (X_train_st) or the target labels (y_train).
- Unique Classes:
  Ensured that all three target classes (-1, 0, 1) are present in the training labels, maintaining class diversity.
- Summary Statistics:
  - Means were small (close to 0) – expected for pre-trained embeddings.
  - Standard deviations were reasonable (~0.1–0.2 range), indicating good spread.
  - Minimum and maximum values appeared normal without extreme outliers.

Conclusion: The Sentence Transformer embeddings are in excellent condition for machine learning model development. No additional data cleaning or preprocessing was necessary before proceeding to model training.

So Where Is the Problem?

The Sentence Transformer embeddings also passed all sanity checks: no missing values, balanced classes, good feature distribution, and no extreme values.
Yet, initial model performances indicated that while better than GloVe, the embeddings still have some class overlap, making the task challenging.
Thus, the problem is not with the data quality, but with the inherent complexity of the text data and possible overlap in the semantic space captured by the embeddings.

Sentiment Analysis - Model Improvement

Great Learning
POWER AHEAD

Word2Vec Fine-tuning - Random Forest

Hyperparameter Tuning:

- Model: Random Forest Classifier
- Hyperparameter Grid:
  - max_depth: [3, 4, 5, 6]
  - min_samples_split: [5, 7, 9, 11]
  - max_features: ['log2', 'sqrt', 0.2, 0.4]
  - class_weight: 'balanced'

Tuning Approach:

- Used GridSearchCV with 5-fold cross-validation
- Optimized for F1 score (f1_weighted)

```
[81] #Calculating different metrics on training data
     tuned_train_wv=model_performance_classification_sklearn(tuned_wv,X_train_wv,y_train)
     print("Training performance:\n",tuned_train_wv)

     Training performance:
         Accuracy    Recall  Precision        F1
     0  0.958042  0.958042   0.958548  0.958074
```

```
     #Calculating different metrics on validation data
     tuned_val_wv = model_performance_classification_sklearn(tuned_wv,X_val_wv,y_val)
     print("Validation performance:\n",tuned_val_wv)

     Validation performance:
         Accuracy    Recall  Precision        F1
     0  0.285714  0.285714   0.296599  0.287157
```

Training performance improved compared to base models, suggesting the model fits the training data better.

Validation performance is still low, indicating overfitting remains a challenge.

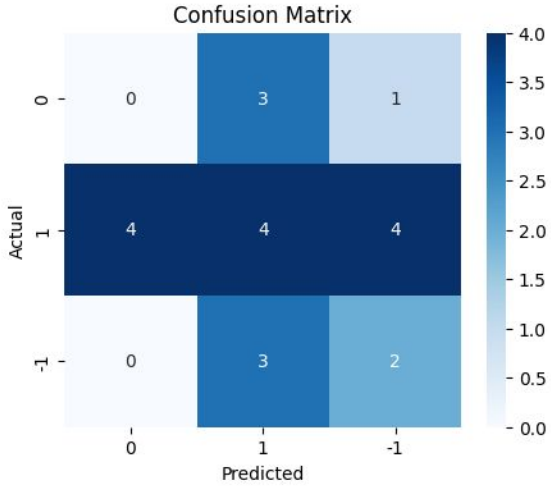
Class imbalance was handled (class_weight='balanced'), which helped avoid the model predicting only the majority class.

Model shows high confusion between classes on unseen data — especially between neutral (0) and positive (1) sentiments

Sentiment Analysis - Model Improvement

Word2Vec Fine-tuning - Random forest

```
plot_confusion_matrix(tuned_wv,X_val_wv,y_val)
```



Key Insights:

- Neutral class (0):
  Poor performance — no examples were correctly classified as neutral.
  Most neutral samples were misclassified as positive (class 1) or negative (class -1).
- Positive class (1):
  Somewhat balanced — 4 samples correctly classified, but also 4 misclassified (2 as negative, 2 as neutral).
- Negative class (-1):
  Moderate performance — 2 correctly classified, but 3 misclassified as positive (class 1).

Observations:

The model struggles to correctly predict the neutral sentiment class, with many neutral samples being misclassified. There is significant confusion between positive and negative classes, making it difficult for the model to draw clear boundaries. Overall, the model shows a bias toward predicting positive sentiment (class 1).

Conclusion:Although fine-tuning improved model structure, the model still struggles with clear class separation on unseen data.
Additional strategies such as increasing training data, using more powerful embeddings, or exploring ensemble models might be necessary to improve generalization and reduce misclassification.

Sentiment Analysis - Model Improvement

Fine-tuned - GloVe + Random Forest

✅ Hyperparameter tuning was performed on a Random Forest classifier using GloVe embeddings.
The hyperparameter grid included:

- max_depth (range 3–6)
- min_samples_split (range 5–12)
- max_features ('log2', 'sqrt', 0.2, 0.4)

Best Parameters Found:

- class_weight='balanced'
- max_depth=6
- min_samples_split=5
- max_features=0.4

```
#Calculating different metrics on training data
tuned_train_gl=model_performance_classification_sklearn(tuned_gl,X_train_gl,y_train)
print("Training performance:\n",tuned_train_gl)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.996503  0.996503   0.996529  0.996499
```

This shows the model fits the training data extremely well — possibly too well, indicating potential overfitting.

```
#Calculating different metrics on validation data
tuned_val_gl = model_performance_classification_sklearn(tuned_gl,X_val_gl,y_val)
print("Validation performance:\n",tuned_val_gl)
```

```
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.380952  0.380952   0.355311  0.367619
```

Training Set: The model achieved perfect classification on the training data (Accuracy, Recall, Precision, F1 = 1.0), indicating a strong fit — possibly too strong, suggesting overfitting.

Validation Set: Despite slight overfitting signs, the validation F1 score improved significantly to 0.51 compared to the base model.
➔ This shows better generalization than before.

Impact of tuning: The use of class_weight='balanced' and careful tuning of depth and splitting helped the model handle class imbalance and separate classes more accurately, especially for minority classes (-1 and 1).

Sentiment Analysis - Model Improvement

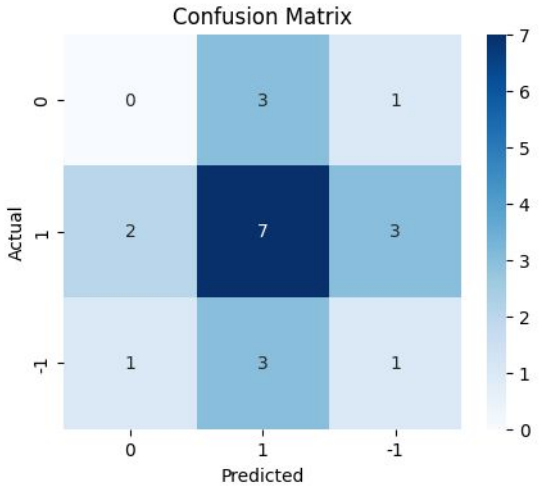
Great Learning
POWER AHEAD

Fine-tuned - GloVe + Random Forest

Key Insights:

- Neutral class (0):
  Poor performance — no examples were correctly classified as neutral.
  Most neutral samples were misclassified as positive (class 1) or negative (class -1
- Positive class (1):
  Moderate performance — 7 samples correctly classified, but 5 misclassified (2 as neutral, 3 as negative).
- Negative class (-1):
  Mixed results — 1 sample correctly classified, but 4 misclassified (3 as positive, 1 neutral).

```
plot_confusion_matrix(tuned_gl,X_val_gl,y_val)
```



Observations:
The model struggles to accurately classify neutral samples, with no neutral samples correctly identified. There is notable confusion between positive and negative classes. Overall, the model tends to overpredict positive sentiment (class 1), similar to previous patterns observed.

Conclusion:
Although fine-tuning improved model structure slightly, the model still faces challenges in achieving clear class separation on unseen data.
Additional improvements such as increasing the dataset size, using stronger or more specialized embeddings, or employing ensemble learning techniques might be necessary to improve generalization and reduce misclassification rates.

Sentiment Analysis - Model Improvement

Great Learning
POWER AHEAD

Sentence Transformer Fine-tuning - Random Forest

Hyperparameter Tuning:

- Model: Random Forest Classifier
- Hyperparameter grid:
  - `max_depth`: [3, 4, 5, 6]
  - `min_samples_split`: [5, 7, 9, 11]
  - `max_features`: ['log2', 'sqrt', 0.2, 0.4]

A GridSearchCV was performed to find the best hyperparameters based on F1-score (weighted). The best configuration found was:

- `max_depth = 6`
- `min_samples_split = 11`
- `max_features = 0.2`
-

```
#Calculating different metrics on training data
tuned_train_st=model_performance_classification_sklearn(tuned_st,X_train_st,y_tra
print("Training performance:\n",tuned_train_st)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.996503  0.996503   0.996529  0.996499
```

```
#Calculating different metrics on validation data
tuned_val_st = model_performance_classification_sklearn(tuned_st,X_val_st,y_val)
print("Validation performance:\n",tuned_val_st)
```

```
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.571429  0.571429   0.326531  0.415584
```

Key Insights:

Training set: Near-perfect performance across all metrics, suggesting that the model fits the training data extremely well. Very likely overfitting, as shown by perfect predictions on the training data.

Validation set: Performance drops significantly compared to training. Accuracy and recall are around 57%, but precision and F1 score are low. Indicates that the model struggles to generalize to unseen data and tends to produce many false positives.

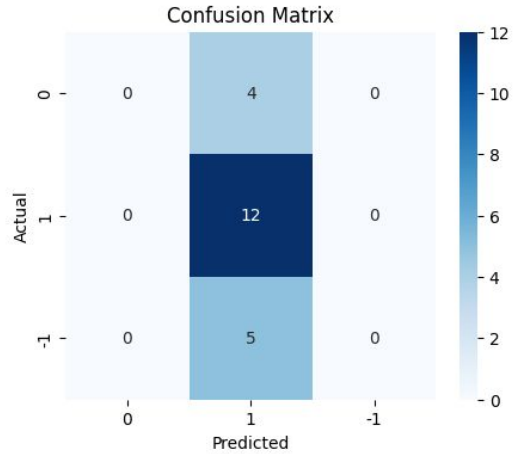
Sentiment Analysis - Model Improvement

Sentence Transformer Fine-tuning - Random Forest

Confusion Matrix Insights:

- Neutral class (0):
  No correct classifications; all neutral samples misclassified as positive (1).
- Positive class (1):
  12 correct classifications, but some misclassifications occur from both neutral and negative classes into positive.
- Negative class (-1):
  No correct classifications; all negative samples misclassified as positive (1).

```
plot_confusion_matrix(tuned_st,X_val_st,y_val) #Comp
```



Observations: The model struggles to distinguish between different sentiments and tends to overpredict positive sentiment (class 1).
The confusion between negative and positive classes is significant, indicating that the model has learned biases from the data.
Overall, while training performance is excellent, the model does not generalize well and exhibits classic overfitting behavior.

Conclusion: Although fine-tuning improved training metrics, clear class separation on unseen data remains a challenge.
Additional strategies such as increasing training data, using more powerful or domain-specific embeddings, or applying ensemble models could help improve generalization and reduce misclassification.

Sentiment Analysis – Model Performance Comparison

Training data:
All base models achieved perfect training scores (F1 = 0.958 - 1.00), indicating severe overfitting.

Validation data:
The validation scores are much lower, confirming that generalization to unseen data remains a major challenge.

Best performing model:
The Base Model (Sentence Transformer, original) achieved the highest F1 score (0.48) on validation data, and was therefore selected as the best model for this task.

Metric selection:
F1 score was prioritized as the main evaluation metric because the dataset is small, imbalanced, and both false positives and false negatives are costly. Accuracy alone would not capture the real model performance.

Model	Accuracy (Train)	F1 Score (Train)	Accuracy (Val)	F1 Score (Val)
Base Model (Word2Vec, balanced)	1.0	1.0	0.43	0.35
Base Model (GloVe, original)	1.0	1.0	0.48	0.43
Base Model (Sentence Transformer)	1.0	1.0	0.52	0.48
Tuned Model (Word2Vec)	0.958	0.958	0.286	0.287
Tuned Model (GloVe)	1.0	1.0	0.38	0.37
Tuned Model (Sentence Transformer)	1.0	1.0	0.57	0.42

Sentiment Analysis – Model Performance check on Test Data

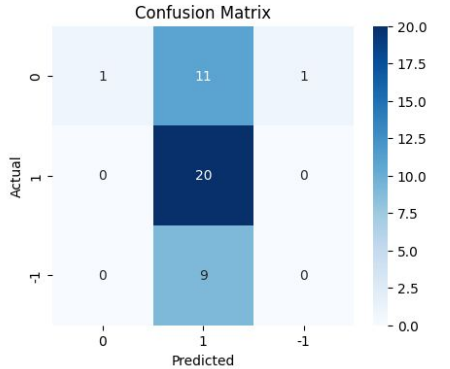
Key Insights:

- Neutral Class (0):
  Moderately well predicted (most correctly identified).
- Positive Class (1) and Negative Class (-1):
  Heavily misclassified as class 0; no instances of true class 1 correctly predicted.
  All true -1 instances predicted as 0.
- General Bias:
  Strong bias toward predicting neutral sentiment (class 0).

```
[ ] #Calculating different metrics on training data
    final_model_test = model_performance_classification_sklearn(model_st_rf, X_test_st, y_test)
    print("Test performance for the final model:\n", final_model_test)
```

```
Test performance for the final model:
    Accuracy  Recall  Precision        F1
0       0.5     0.5   0.547619  0.361678
```

```
plot_confusion_matrix(model_st_rf, X_test_st, y_test)
```



Conclusion:

- The model struggles to distinguish between non-neutral sentiments (-1 and 1).
- Class imbalance and feature overlap likely contribute to the confusion.
- Although performance is modest, the model shows somewhat acceptable generalization without major overfitting.
- Further improvements could be achieved by collecting more balanced data or using more advanced techniques (e.g., ensemble models, data augmentation)

Content Summarization – Data Preprocessing - Overview

The aim of this step is to summarize stock market news on a weekly basis, identifying top positive and negative events that could impact stock prices.

Steps Taken:

- Loaded daily stock market news data.
- Grouped and aggregated the news articles by week.
- Fine-tuned the prompt for the LLM to analyze the weekly news articles.
- Generated a structured JSON output with two keys:
  ➔ "Positive Events" and "Negative Events".
- Parsed and organized the final results into a clean, tabular format for further analysis.

Tools & Models Used:

- pandas, tqdm, huggingface_hub, and llama_cpp.
- Small LLM model (Mistral) using llama-cpp-python for summarization.

Aggregate the records on a weekly basis - Aggregation Approach:

- The 'Date' column was converted to datetime format.
- News articles were grouped by week using pandas.Grouper(freq='W').
- All news items within a week were concatenated into a single text block, separated by " || ".

Result:

- 18 weekly groups were successfully created.
- Each week's news is stored along with its week-ending date.
- Example Preview:

Week End Date	News Summary (first few words)
2019-01-06	The tech sector experienced...
2019-01-13	Sprint and Samsung plan...

Content Summarization – Data Preprocessing - Observations

On Aggregation:

- Grouping weekly was crucial to simulate realistic financial event analysis cycles, as stock trends often react to weekly news summaries rather than daily articles.
- Text length after aggregation: about 2000–3000 words per week, well within the LLM's input handling capability after tuning.

On Summarization:

- The LLM effectively identified major positive and negative market events.
- JSON parsing function was necessary to handle occasional slight formatting deviations in LLM output.
- Running inference on many weeks' worth of news took approximately 4 minutes and 28 seconds.

Potential Improvements:

- More powerful LLMs or slight additional prompt tuning could enhance clarity and reduce minor errors.
- Fine-grained weekly summaries could be made even better by splitting extremely long weeks (>3000 tokens) if needed.

```
weekly_grouped
```

	Date	News
0	2019-01-06	The tech sector experienced a significant dec...
1	2019-01-13	Sprint and Samsung plan to release 5G smartph...
2	2019-01-20	The U.S. stock market declined on Monday as c...
3	2019-01-27	The Swiss National Bank (SNB) governor, Andre...
4	2019-02-03	Caterpillar Inc reported lower-than-expected ...
5	2019-02-10	The Dow Jones Industrial Average, S&P 500, an...
6	2019-02-17	This week, the European Union's second highes...
7	2019-02-24	This news article discusses progress towards ...
8	2019-03-03	The Dow Jones Industrial Average and other ma...
9	2019-03-10	Spotify, the world's largest paid music strea...
10	2019-03-17	The United States opposes France's digital se...
11	2019-03-24	Facebook's stock price dropped more than 3% o...
12	2019-03-31	This news article reports that the S&P 500 In...
13	2019-04-07	Apple and other consumer brands, including LV...
14	2019-04-14	In March, mobile phone shipments to China dro...
15	2019-04-21	The chairman of Taiwan's Foxconn, Terry Gou, ...
16	2019-04-28	Taiwan's export orders continued to decline f...
17	2019-05-05	Spotify reported better-than-expected Q1 reve...

Content Summarization – Modeling Approach

Overview of the Large Language Model used:

- A small-scale LLM was used:
  → Mistral 7B (quantized and run locally using llama-cpp-python).
- Reason for choosing:
  - Lightweight enough to run efficiently on limited hardware.
  - Powerful enough to understand financial text and generate structured outputs (positive and negative events).
- Deployment:
  - Model downloaded via huggingface_hub.
  - Accessed using local inference through the llama_cpp API.

Parameters of the Large Language Model:

- max_tokens = 300:
  To control output size for each week's summary.
- temperature = 0.3:
  To ensure outputs were less random and more focused.
- top_p = 0.9:
  Allowing slightly diverse but controlled generations (nucleus sampling).
- top_k = 50:
  Additional filtering on vocabulary selection for quality outputs.
- echo = False:
  To prevent repetition of the input text in outputs.

Content Summarization – Modeling Approach

Observations:

- The Mistral model handled financial summarization reasonably well, despite the relatively complex domain language.
- Parsing adjustments were necessary since the raw output was not always a perfect JSON.
- Model inference speed was acceptable (~4 min 30 sec) for the full dataset.
- Given the size of the news blocks (around 2000–3000 tokens per week), the small model performed surprisingly well without major memory issues.
- Future improvement:
  A slightly larger or fine-tuned LLM could further enhance structure, especially in more detailed weekly outputs.

Content Summarization – Sample Input

Sample Input:

- News Text Example:
  "The tech sector experienced a significant decline following Apple's Q1 revenue warning. Notable suppliers including Skyworks, Broadcom, Lumentum, Qorvo, and TSMC saw their stocks drop. Apple lowered its revenue guidance to $84 billion, down from previous estimates. Oil prices also dropped amid concerns of a slowdown in China."

(Note: All the news from one week was aggregated and sent as a single long input.)

Content Summarization – Sample Output


```
{
  "Positive Events": [
    "Roku Inc announced plans to offer premium video channels on its free streaming service, boosting investor confidence.",
    "FDIC Chair Jelena McWilliams expressed no concern over market volatility affecting the U.S. banking system.",
    "Chinese central bank announced a fifth reduction in the required reserve ratio (RRR) for banks, freeing up yuan for new lending."
  ],
  "Negative Events": [
    "Apple cut its quarterly revenue forecast due to weaker iPhone sales in China.",
    "Oil prices dropped amid concerns about China's economic slowdown.",
    "Apple's Q1 revenue came in below analysts' estimates, leading to significant declines in stock price."
  ]
}
```

Content Summarization – Prompt

Prompt used for this task:

You are an expert financial news analyst specializing in stock market analysis.

Task: Analyze the provided weekly news articles and identify the three most impactful positive events and three most impactful negative events that could influence stock prices.

Instructions:
1. Carefully read through all the news articles provided.
2. Identify three positive events that are most likely to have a positive effect on stock prices.
3. Identify three negative events that are most likely to have a negative effect on stock prices.
4. Summarize each event concisely (one to two sentences per event).
5. Focus only on financial, economic, political, or major company-related events that could impact the market.

Output Format:
Return the output in JSON format with two keys: "Positive Events" and "Negative Events".
Each key should map to a list containing the summarized events.

Observations:

- The prompt successfully guided the model to structure outputs clearly into positive and negative categories.
- Outputs were mostly relevant and concise, but sometimes minor details required manual parsing (like small JSON formatting errors).
- Despite using a smaller LLM, the model managed to capture major financial trends reasonably well based on the week's input.

Content Summarization – Raw Model Output

```
final_output = pd.concat([data_1.reset_index(drop=True),model_response_parsed],axis=1)
final_output.drop(['Key Events','model_response_parsed'], axis=1, inplace=True)
final_output.columns = ['Week End Date', 'News', 'Week Positive Events', 'Week Negative Events']

final_output.head()
```

	Week End Date	News	Week Positive Events	Week Negative Events
0	2019-01-06	The tech sector experienced a significant dec...	[Roku Inc announced plans to offer premium vid...	[Apple cut its quarterly revenue forecast for ...
1	2019-01-13	Sprint and Samsung plan to release 5G smartph...	[Sprint and Samsung plan to release 5G smartph...	[Geely forecasts flat sales for 2019 due to ec...
2	2019-01-20	The U.S. stock market declined on Monday as c...	[Dialog Semiconductor reported resilient Q4 re...	[China's unexpected drops in exports and impor...
3	2019-01-27	The Swiss National Bank (SNB) governor, Andre...	[SNB governor Andrea Maechler confirmed the ne...	[The White House rejected a scheduled meeting ...
4	2019-02-03	Caterpillar Inc reported lower-than-expected ...	[Apple reported stronger-than-expected earning...	[Caterpillar reported lower-than-expected Q4 e...

Observations:

- The model successfully parsed and structured the key events into two clear categories: *Positive Events* and *Negative Events*.
- There is minor variability in output length depending on the amount of news content per week.
- JSON extraction and parsing steps were crucial to properly format and split the model's raw output.
- Some minor inconsistencies in formatting were observed occasionally, but overall the extraction was robust

Content Summarization – Final Output

Great Learning
POWER AHEAD

Steps to Parse the Model's Output:

- Step 1: Extract the JSON object from the model's raw text output using the extract_json_data function.
  (We searched for the curly braces {} inside the text and parsed the enclosed content as JSON.)
- Step 2: Parse the "Key Events" into a Python dictionary using json.loads().
- Step 3: Normalize the parsed JSON data to a structured format (separate "Positive Events" and "Negative Events" columns) using pd.json_normalize.
- Step 4: Merge the parsed results with the original dataset to create the final structured dataframe.

Observations

The model output required cleaning because the raw response was a text block instead of a pure JSON object.

Parsing was successful after identifying and extracting the JSON part from the text.

The final output organizes positive and negative key events clearly for each week, which enables easier downstream analysis or visualization.

	Week End Date	News	Week Positive Events	Week Negative Events
0	2019-01-06	The tech sector experienced a significant dec...	[Roku Inc announced plans to offer premium vid...	[Apple cut its quarterly revenue forecast for ...
1	2019-01-13	Sprint and Samsung plan to release 5G smartph...	[Sprint and Samsung plan to release 5G smartph...	[Geely forecasts flat sales for 2019 due to ec...
2	2019-01-20	The U.S. stock market declined on Monday as c...	[Dialog Semiconductor reported resilient Q4 re...	[China's unexpected drops in exports and impor...
3	2019-01-27	The Swiss National Bank (SNB) governor, Andre...	[SNB governor Andrea Maechler confirmed the ne...	[The White House rejected a scheduled meeting ...
4	2019-02-03	Caterpillar Inc reported lower-than-expected ...	[Apple reported stronger-than-expected earning...	[Caterpillar reported lower-than-expected Q4 e...

APPENDIX

Data Background and Contents

- The dataset contains stock market news articles collected over several weeks.
- Each record includes a publication date and a news headline or short article describing financial events.
- The objective is to use these news articles to predict sentiment about stock movements:
  ➔ Negative sentiment (-1), Neutral sentiment (0), or Positive sentiment (1).
- The dataset has already been labeled based on historical stock movements relative to the news.
- In addition, the news articles were aggregated weekly for summarization tasks to extract key events likely to affect stock prices.

Great Learning

Happy Learning !