NOVA IMS Information Management School

Master's Degree Program in

# MDSAA

**Data Science and Advanced Analytics**

## Text Mining

Financial Tweets' Sentiment Analysis


Hassan Bhatti

Moeko Mitani

Oumaima Ben hfaiedh

Rute Teixeira

Sarah Leuthner

Group 12

**NOVA Information Management School**

**Instituto Superior de Estatística e Gestão de Informação**

Universidade Nova de Lisboa


June, 2025

# Table of Contents

# 1. INTRODUCTION

Financial markets are influenced by a range of psychological and economic factors, often reflected in investor sentiment. Social media platforms like Twitter have become key channels where individuals share their opinions and reactions to market events in real-time. These posts can convey optimism, pessimism, or neutrality towards market trends, commonly referred to as bullish, bearish, or neutral sentiments.

This project aims to leverage Natural Language Processing (NLP) techniques to build a classification model capable of analyzing tweets and determining the underlying market sentiment. To achieve this, extensive Data Exploration is conducted, followed by a corpus split to allocate a portion of the training data for validation purposes. Data preprocessing and feature engineering steps are conducted for each classification model. Finally, through an evaluation phase, the most effective modeling pipeline is identified.

# 2. Project Structure and Execution

To ensure the analysis is easy to follow and supports the collaborative nature of the project, whenever found necessary, the project has been modularized into sequential, self-contained Jupyter notebooks. Each notebook can be run independently and includes all necessary steps for its part of the workflow.

- **tm_tests_01_12**: Main notebook. Contains: full Exploratory Data Analysis, Corpus Split and Pipeline from data preprocessing to model evaluation for the following models: Machine Learning Models, LSTM, and DistillBERT.
- **tm_tests_02_12**: Second notebook. Contains full workflow to implement and evaluate GPT-2 model.
- **tm_tests_03_12**: Third notebook. Contains full workflow to implement and evaluate FinBERT model.
- **tm_final_12**: Test Predictions. Contains predictions on test set, using our best model DistillBERT.

# 3. DATA EXPLORATION

The train set contains 9,543 tweets labeled as Bearish (0), Bullish (1), or Neutral (2). The test set has 2,388 unlabeled tweets with corresponding IDs. A significant class imbalance exists, with Neutral tweets comprising 65%, Bullish 20%, and Bearish 15% of the training data.

## 3.1. Structure Analysis & Common Words

The structure of tweets across both datasets shows consistent patterns with similar distributions. Most tweets are between 8 and 12 words long, with notable peaks in character length at around 60 and 150 characters. These peaks likely correspond to short commentary and tweets reaching the platform's character limit. While there are no significant differences in tweet length across the sentiment classes, Bullish tweets tend to be slightly shorter on average.

A closer look at the most common words in each sentiment category reveals distinct lexical patterns for Bearish and Bullish tweets. As all classes often use the term *'stock'*,

- Bearish tweets often include terms like *'us', 'misses', 'coronavirus'* and *'oil'*
- Bullish tweets show keywords such as '*beats', 'price'* and *'target'*

- Neutral tweets lack distinct common terms but often include *'market screener'* and *'Trump'*
- Test data is dominated by terms like *'https'* and *'earnings'*

### 3.2. Special Characters

Regarding content, many tweets include company **tickers**, particularly *'SPY'*, *'COMDX'*, and *'TSLA'*. These tickers are often represented with an '$' and could be a useful feature for further sentiment classification. Additionally, 40-50% of tweets contain **URLs**, which may indicate references to news articles or financial reports. Furthermore, the presence of **punctuation** various across sentiments. Neutral tweets are more likely to include question marks, suggesting uncertainty or inquiry. Bullish tweets frequently contain exclamation marks, emphasizing enthusiasm or confidence. Ellipses appear more often in Bullish and secondly in Neutral tweets, possibly indicating incomplete thoughts or sarcasm. These punctuation patterns may serve as subtle indicators of sentiment and could enhance feature richness in model training.

An analysis of character encoding quality reveals that 2,243 rows in the train set and 585 in the test set contain non-ASCII characters. These may pose challenges during preprocessing and should be cleaned or normalized to ensure compatibility with model training.

### 3.3. Named Entity Recognition (NER)

To further enrich the dataset, Named Entity Recognition (NER) was applied to identify references to real-world entities such as people, organizations, locations, and products. It is a foundational step in extracting structured data from unstructured language. An overall sentiments summary table of the insights can be found bellow:

*Table 1: Named Entity Recognition (NER) - Summary Table (Overall)*

| Entity Type | Top Entities (Overall) | Observation |
|---|---|---|
| Organizations | Fed, NYSE, GMT, Trump | No conspicuous observations. |
| People | MarketScreener, Seeking Alpha, Trump Brexit | Some misclassifications (e.g., MarketScreener and Brexit as people) due to model limitations. |
| Countries & Cities | US, China, UK | No preprocessing in EDA phase leads to name variations of some entities. |
| Products | Q3, Q4 (misclassified), ETF, Discovery, Maersk, Ocwen | Time references (Q3, Q4) incorrectly tagged products. Real products appear only at lower levels. |

As countries and organizations are detected the best with this approach, a further analysis per sentiment was done. The results are in the table below.

*Table 2: Named Entity Recognition (NER) - Top Organizations and GPE's per Sentiment*

| Sentiment | Top Organizations | Top GPEs |
|---|---|---|
| Neutral | GMT, Trump, TSE, EU | India, Australia, Hong Kong, France, UK, US, China |
| Bearish | Dow, Boeing, EPS | London, California, China, US |
| Bullish | FactSet, Nasdag, Dow | US, China, UK |

Neutral tweets mention significantly more organizations than other sentiments. They also have the highest number of GPE mentions overall and in proportion. As the most represented class, they display wide diversity in both organizations and countries. Bearish tweets mention fewer organizations overall

but show a higher percentage of GPEs compared to Bullish. They focus on risk or negative locations/entities. Bullish tweets mention more organizations than Bearish, but fewer than Neutral. GPE mentions dominated by major markets, reflecting global positive market sentiment.

### 3.4. LEXICON-BASED SENTIMENT-ANALYSIS

A Lexicon-based sentiment analysis was also conducted using the Valence Aware Dictionary and Sentiment Reasoner (VADER) tool. VADER uses a predefined sentiment lexicon where each word is annotated with a score indicating its emotional value. The overall sentiment of a tweet is derived by aggregating the individual word scores. The output are the following metrics:

- Positive: compound ≥ 0.05
- Negative: compound ≤ -0.05
- Neutral: -0.05 < compound < 0.05

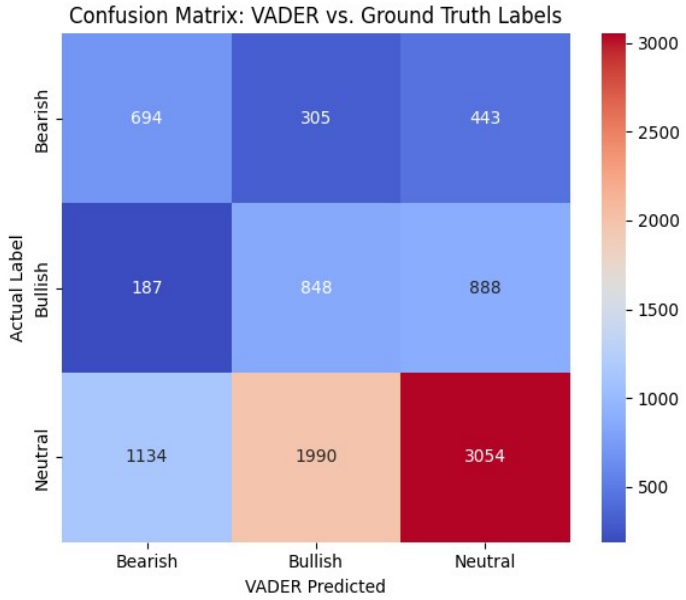


*Figure 1 Result Lexicon-Based Sentiment Analysis - Confusion Matrix*

To evaluate the alignment between VADER predictions and the actual labeled sentiments, a confusion matrix was generated (see Figure 1). These results indicate that VADER tends to over-predict the Neutral label, particularly misclassifying many Bullish tweets as Neutral. Bearish tweets are frequently misclassified as either Neutral or Bullish, with more than half incorrectly labeled. Similarly, around half of the Bullish tweets are misclassified as Neutral.

These discrepancies suggest that while VADER provides a helpful baseline, it struggles with the complexity of financial language, including sarcasm, ambiguity, and subtle tonal cues. The high misclassification rate, especially for non-neutral sentiments, highlights the limitations of lexicon-based methods in this domain and underscores the need for more context-aware machine learning models. Therefore, thorough data preprocessing, especially given the presence of noisy elements like URLs, tickers, and emojis, becomes critical for improving downstream sentiment classification performance.

# 4. CORPUS SPLIT

To prepare the dataset for model training and evaluation, we performed a train-test split that separates the data into training and validation sets. This ensures that model performance is evaluated on unseen data, providing a realistic measure of generalization.

Specifically, the dataset was split as follows:

- The target variable *label* was separated from the feature set
- 80-20 split for Machine Leaning models, LSTM, DistilBERT, and FinBERT, and 90-10 split for GPT2 were applied using stratified sampling (*stratify=y*) to preserve the original class distribution across train and test sets
- A fixed random_state=42 was used to ensure reproducibility

The resulting train input data sets include both raw text and engineered numeric/categorical attributes. Such a setup enables us to build flexible preprocessing pipelines tailored to different features and models.

# 5. MODELING

Since each model has a different approach, we decided to create a section for each model that includes data preprocessing and feature engineering separately.

## 5.1. MACHINE LEARNING MODELS (contains EXTRA WORK)

### 4.1.1. Data Preprocessing

Machine Learning (ML) models require structured, numerical input. Raw text like tweets must first be transformed into appropriate representations. This is especially important when working with a combination of data, as the features generated in EDA can be used at this point. The preprocessing pipeline ensures that all data types are compatible with the models and following these steps:

*Table 3: Preprocessing Pipeline for Machine Learning Models*

| Preprocessing Step | Description |
|---|---|
| 1. Lowercasing | All text is converted to lowercase to reduce vocabulary size. |
| 2. URL Removal | Removes URLs using patterns that match (e.g., http, https). |
| 3. Ticker Cleaning | Removes standalone and malformed ticker symbols '$' - keeps stock tickers, needed for sentiment analysis. |
| 4. User & Hashtag Removal | Remove Twitter mentions (@user) and hashtags (#topic). |
| 5. Ellipsis Normalization | Detects any sequence of dots of 2 or higher and normalizes into ellipsis. |
| 6. Character Filtering | Removes digits and unwanted punctuation - preserves "!", "...", "?" symbols, as found sentiment suggestive in EDA. |
| 7. Whitespace Normalization | Only a single space should be represented. |
| 8. Tokenization | Splits the cleaned string into a list of word tokens. |
| 9. Stopword Removal | Set of customized stop words - to preserve important ones, like negations (e.g., not, never), while excluding tweet-specific stop words that can add noise. |

| | |
|---|---|
| 10. POS Tagging | Applies Part-of-Speech tagging to guide context-aware lemmatization. |
| 11. Lemmatization | Converts words to their base form (e.g., running-> run). |
| 12. Stemming | Reducing words to their root form by chopping off suffixes. Applies stemming when toggle is on, as an alternative to *Lemmatization.* |

The following steps of preprocessing are wrapped into a function, *clean text,* which is then enabled through a custom transformer called *Text Cleaner classifier.* The transformer is parameterized to toggle between stemming and lemmatization.

### 4.1.2. Feature Engineering (With Extra Work)

This section outlines the feature engineering strategies applied to prepare text and structured data specifically for ML models, following course mandatory and optional requirements:

- **Feature Engineering from EDA:** Generated during the first stage of our project, numeric (e.g., tweet length, sentiment scores) and categorical features were created
- **Scaling and Encoding:** The features from EDA were scaled using a *StandardScaler* or *One-Hot Encoded* for ML models, these are not going to be used in LSTM/Transformer models
- **TF-IDF (Term Frequency Inverse Document Frequency):** A BoW variation vectorizer that converts cleaned text into sparse vectors for traditional ML models (e.g., Logistic Regression and Random Forest)
- **Sentence-BERT Embeddings (EXTRA WORK):** Generate dense sentence-level vectors as a fast alternative to full transformer pipelines. They are easier and faster to use and do not require finetuning (can be used as an alternative to text cleaning and vectorization)

### 4.1.3. Pipeline Selector

For ML models, a more robust and modular preprocessing pipeline is required to integrate both text transformation and structured feature engineering.

As described earlier, cleaned text is transformed into numerical features using *TF-IDF vectorization*, a *Bag-of-Words (BoW)* technique. Alternatively, we implemented a more context-aware approach using **Sentence-BERT (SBERT)**, which directly encodes raw text into dense vector embeddings, replacing the need for TF-IDF. Regardless of the chosen text representation, numeric and categorical features still require appropriate scaling and encoding. To accommodate this flexibility, we dynamically built a preprocessing selector, allowing toggles for:

- Using **SBERT or TF-IDF**
- Applying **stemming** or **lemmatization**

Based on the selected configuration, the appropriate pipeline is applied to the training and validation data prior to model fitting. This design promotes experimentation and comparative evaluation across different preprocessing strategies.

### 4.1.4. Modeling

Traditional ML models were implemented for assessment, leveraging the unique engineered features during Data Exploration and allowing us to test new pipelines for text cleaning and handling, as outlined in the Feature Engineering and Preprocessing section above. Each selected model was trained using three different text preprocessing configurations:

1) *TF-IDF Vectorization using lemmization*
2) *TF-IDF Vectorization using stemming*
3) *Sentence-BERT embeddings as text preprocessing*

We also tested an additional variation, replacing *StandardScaler* with *MinMaxScaler* for numeric features. However, this approach of normalization did not improve performance and was excluded from the results.

Despite our expectations, performance across the three configurations remained the same, with no standout improvements from stemming or embeddings. This made us question our implementation, leading us into further testing and code-proofing, detecting no apparent errors. Nonetheless, in the Evaluation section, we will only report the first configuration for each model.

The models selected include **K-Nearest Neighbors (KNN)**, **Naive Bayes**, **Logistic Regression (LR)** and **Random Forest (RF)**. KNN, Naive Bayes, and LR were included due to their relevance in the course labs. Random Forest was added based on its popularity and strong performance in text classification tasks reported in literature.

#### 4.1.5. Model Training

All models were first run with default settings, including, *class_weights='balanced'* where applicable, to address the dataset's class imbalance. After initial experimentation and examining classification reports, Logistic Regression and Random Forest emerged as top performers. To further optimize their performance, we conducted hyperparameter tuning as follows:

- **Random Forest**
  We used *GridSearchCV* from *sklearn.model_selection*, for a search space of 24 hyperparameters combinations over 5 cross-validation folds, resulting in 120 fits. The best parameters were: *n_estimators=200*, *max_depth=None*, *max_features='log2'*, *min_samples_split=5*, *random_state=42*, *n_jobs=-1*, *class_weight='balanced'*

- **Logistic Regression**
  Over training, the best configuration parameters were: max_iter=1000*,* random_state=0*, class_weight='balanced'.*

### 5.2. LONG SHORT-TERM MEMORY (LSTM)

#### 4.2.1 Data Preprocessing

Long Short-Term Memory (LSTM) models are designed to handle sequential data, such as tweets. However, these models cannot directly process raw text or unstructured data. Instead, they require the input to be in the form of clean, tokenized sequences of integers, where each integer represents a specific token from a predefined vocabulary. The pipeline includes the following steps:

*Table 4: Preprocessing Pipeline for LSTM Models*

| Preprocessing Step | Description |
|---|---|
| 1. Lowercasing | All text is converted to lowercase to reduce vocabulary size. |
| 2. Emoji Extraction & Removal | Emojis are extracted (optional for sentiment) and then removed from the text. |
| 3. URL Normalization | All URLs are replaced with '<URL>' to generalize external links. |
| 4. User Normalization | All users like @user is replaced with '<USER>'. |
| 5. Punctuation & Digit Removal | All non-alphabetic characters expect for specific punctuation (mentioned in the EDA section prior) are removed. |
| 6. Whitespace Normalization | Only a single space should be represented. |
| 7. Stopword Removal | Custom stop word list excludes neutral words but retain negations (e.g., not, never). |
| 8. Tokenization | Cleaned text is split into tokens (using *LSTMTokenizer*). |
| 9. Integer Encoding | Tokens are converted into sequences of integers using a tokenizer. |
| 10. Padding | Sequences are padded to a uniform length (length of longest sequence) for batch processing. |

### 4.2.2 Feature Engineering

In the preprocessing pipeline, emojis are extracted from the text and preserved separately, providing an additional signal that captures emotional or contextual nuances. The text is also transformed into sequences of integers representing tokens and padded to a uniform length, enabling consistent input formatting for the model.

### 4.2.3 Modeling

We built an LSTM model for a multi-class text classification task using *TensorFlow* and *Keras*. The model starts with an input layer for padded sequences, followed by a trainable embedding layer with 100 dimensions to learn word representations. We added a masking layer to make sure padding does not affect the results. Then, a bidirectional LSTM layer with 64 units processes the sequence from both directions. To help prevent overfitting, we used dropout during training, which randomly drops some neurons and helps the model generalize better. The final output layer uses **softmax** to predict one of three classes. Since our labels are not one-hot encoded, we used *sparse_categorical_crossentropy* as the loss function. To deal with class imbalance, we calculated **class weights** and used them during training. The model was trained for **10 epochs** with a batch size of 16, using 20% of the data for validation.

## 5.3. ENCODER-ONLY TRANSFORMER – DISTILBERT

### 4.3.1. Data Preprocessing

For the DistilBERT model, traditional preprocessing techniques such as stopword removal, lemmatization, or stemming were not applied, as transformer-based models rely on subword tokenization to handle raw text inputs. Additionally, class imbalance was addressed by computing **class weights** using *sklearn.utils.class_weight*. These weights were integrated into the model's loss function

to mitigate bias toward the majority class (Neutral) and improve performance on underrepresented classes (Bearish and Bullish).

#### 4.3.2. Feature Engineering

In this approach, feature engineering is implicitly performed by DistilBERT, which encodes raw text into contextual embeddings through its pretrained transformer architecture. Unlike traditional vectorization techniques like *Bag-of-Words* or *Word2Vec*, transformer encoders like DistilBERT generate contextual embeddings for input sequences. The text was tokenized using Hugging Face's *AutoTokenizer* with padding and truncation to handle variable input lengths. A custom model subclass was defined to include a weighted cross-entropy loss, accounting for class imbalance. Inputs were prepared using *DataCollatorWithPadding* to ensure consistent input shape across training batches.

#### 4.3.3. Modeling

To train the DistilBERT model, we used Hugging Face's *TrainingArguments* to define the training configuration. The model was trained for **3 epochs** with a batch size of 32 per device. We also tried 2 epochs, but the results were not much different. To simulate a larger effective batch size without exceeding memory limits, we used **gradient accumulation** with a step size of 2. We also applied a **weight decay** of 0.01 to regularize the model and reduce overfitting. During training, evaluation was performed with a batch size of 64, and logs were recorded every 10 steps. All model outputs, checkpoints, and logs were saved to designated directories for later inspection. We disabled external logging integrations by setting *report_to="none".*

### 5.4. GPT-2 (EXTRA WORK)

#### 4.4.1. Data Preprocessing

As for the GPT-2 Model, no traditional text processing techniques such as stopword removal, lemmatization, stemming, or special token normalization were applied. Since GPT-2 is a transformerbased decoder model, it relies on its subword tokenizer (*Byte-Pair Encoding)* to process raw text directly. The only preprocessing performed was resetting the data indices after reading the exported training file (*train_df.csv*) and extracting the text and level columns for model input. All tweets were fed into the GPT-2 tokenizer in their raw form.

#### 4.4.2. Feature Engineering

As for Feature Engineering, nothing was explicitly required for GPT-2. Unlike traditional machine learning models that require structured numeric features or engineered embeddings, GPT-2 internally generated contextual embeddings through its pretrained transformer layers. The only transformation performed was tokenization using Hugging Face's *GPT-2 tokenizer*, which handles tokenization, truncation, and padding of input sequences to a maximum sequence length of 64 tokens.

#### 4.4.3. Modeling

The GPT-2 decoder model was implemented using Hugging Face's *GPT2ForSequenceClassification* class, which adds a classification head to the pretrained GPT-2 architecture. Since GPT-2 does not natively support padding, a special padding token was added to the tokenizer and configuration before training. To reduce computational time, the smaller pretrained variant *sshleifer/tiny-gpt2* was used. The model was fine-tuned for **1 epoch** with a batch size of 4, using the Hugging Face's *Trainer API* for efficient training.

## 5.5. FINBERT (EXTRA WORK)

### 4.5.1. Data Preprocessing

The preprocessing workflow for FinBERT began with uploading the dataset, as we explicitly passed the names parameter as “Sentiment” instead of the original “label”, to guarantee consistency with the rest of the codebase and avoid downstream errors. Sequentially, to prepare for model training the preprocessing phase included the following steps:

*Table 5: Preprocessing Pipeline for FinBERT Model*

| Preprocessing Step | Description |
|---|---|
| 1. Label Validation | Sentiment column contained only valid numeric labels (0, 1, or 2). |
| 2. Feature and Target Separation | Text content (Text) was isolated as the input feature set X, while the sentiment labels were stored in y. |
| 3. Data Integrity Check | Ensure the number of texts matches number of labels; verify no missing values exist in Sentiment column. |

### 4.5.2. Feature Engineering

Feature engineering in this context focused on configuring the *AutoTokenizer* to prepare the input text for FinBERT. The tokenizer was set up with the following key parameters: *padding=’longest’* so that all inputs in a batch are of equal length, truncation to trim long texts exceeding the model's maximum sentence lengh, *max_length =256-token* limit to optimize memory usage while preserving most of the input text's meaning, and *return_tensors="pt"* to output PyTorch tensors. These settings ensure all input texts are processed into uniformly sized tensors the model can handle.

### 4.5.3. Modeling

Due to time constraints, we leveraged a pretrained FinBERT model *“(ahmedrachid/FinancialBERTSentiment-Analysis)”, instead* of training a model from scratch. This model is specifically fine-tuned for financial sentiment classification and eliminates the need for extensive labeled data or long training cycles.

We performed initial testing with sample tweets from our dataset to confirm it could differentiate between Bearish, Bullish, and Neutral sentiments.

Then, full-scale predictions were run in batches to efficiently manage memory and computational resources. The model’s tokenized text inputs perform an inference loop, passing through FinBERT the raw logits that will be then converted into predicted labels. Since our model was already fine-tuned, there was no need for further training or hyperparameter tuning. The pipeline focused solely on inference and evaluation.

# 6. EVALUATION AND RESULTS COMPARISON

This chapter focuses on the final comparison of all models tested: traditional machine learning, LSTM, and transformer-based approaches. Each model evaluations are shown in Table 6:

*Table 6: Comparison of different Models (Pipelines)*

| Model | Recall | Precision | Accuracy | F1-Score |
|---|---|---|---|---|
| LSTM | 0.66 | 0.66 | 0.75 | 0.67 |
| GPT-2 Decoder | 0.33 | 0.21 | 0.64 | 0.26 |
| KNN (TF-IDF lemmatization) | 0.54 | 0.61 | 0.70 | 0.56 |
| Logistic Regression (TF-IDF lemmatization) | 0.73 | 0.70 | 0.77 | 0.71 |
| Naive Bayes (TF-IDF lemmatization) | 0.52 | 0.49 | 0.54 | 0.48 |
| Random Forest (TF-IDF lemmatization) | 0.64 | 0.77 | 0.79 | 0.69 |
| **DistilBERT** | **0.82** | **0.83** | **0.87** | **0.83** |
| FinBERT | 0.60 | 0.70 | 0.74 | 0.63 |

Simplistic models such as **Naive Bayes** and **KNN** demonstrated limited effectiveness. Their core reliance on bag-of word techniques (specifically TF-ID) proved inadequate for capturing the nuanced semantic meaning and contextual relationships in financial language. This reflects the observed issue during the initial EDA phase with the lexicon-based sentiment analysis, reinforcing the limitations of shallow methods in handling subtle linguistic variations.

**GPT-2**, even though a transformer-based model, performed poorly likely due to a combination of factors: a constrained timeframe that limited opportunities for proper fine-tuning, and the use of a lightweight variant, which lacked the capacity to learn domain-specific patterns effectively. Similarly, **FinBERT** produced suboptimal results, because it was used in an inference-only mode without any additional training or task-specific tuning. The unique distribution of labels in the dataset may have contributed to these models' underperformance as well.

**Logistic Regression** and **Random Forest** achieved moderate performances. Improvements can be observed due to the thorough preprocessing, lemmatization, and feature engineering. However, their performance plateaued from the inherent structure of linear and ensemble models, which struggle to model complex contextual dependencies within natural languages.

The **LSTM** model, with its sequence-aware architecture, showed better generalization, but is outperformed by transformer-based models. LSTM's have a limited capacity to capture long-range dependencies and subtle contextual cues.

**DistilBERT** achieved the highest **F1-Score (0.83)**, **Recall (0.82)**, **Precision (0.83)**, and **Accuracy (0.87)**, outperforming all other models. These results derive from the ability to generate contextual embeddings via its self-attention, allowing it to accurately interpret both the meaning and sentiment of financial text. Additionally, the integration of class weights into the loss function helped to address the class imbalance problem, further enhancing the model's performance.

The confusion matrix of DistilBert shows a correct classification of 76% of bearish, 78% of bullish and 92% of neutral tweets (see Figure 2). While the neutral tweet labels benefit from overrepresentation, the model consistently demonstrates strong recall for underrepresented classes (bearish and bullish), confirming its robustness and suitable classification performance across all sentiment categories.

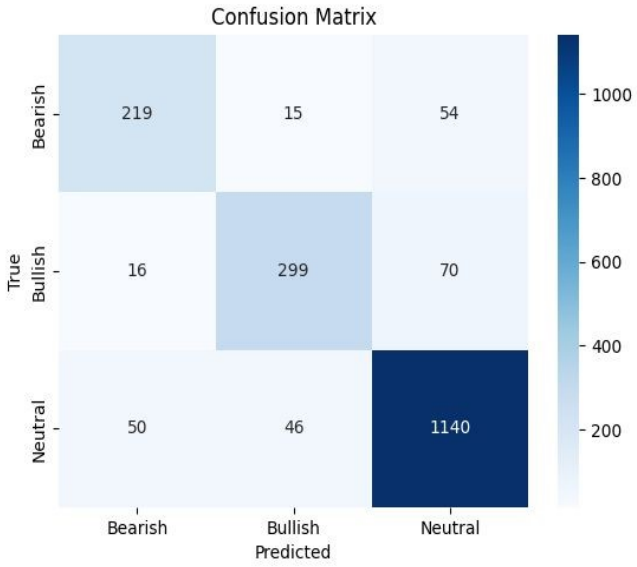


*Figure 2: Confusion Matrix of Model DistilBERT with Best Scores*

## 7. CONCLUSION

In this project, a robust text mining pipeline was developed to classify financial sentiment (Bearish, Bullish, and Neutral) in tweets using a variety of NLP and machine learning techniques. Starting with detailed data exploration, key characteristics of tweets structure, punctuation, and entities were identified. Continuing several classification models, ranging from traditional machine and deep learning as well as transformer-based models were built and evaluated.

DistilBERT emerged as the best preforming model, achieving an F1-Score of 0.83 due to its contextual understanding, resilience to noisy text, and ability to incorporate class weighting during training. The model outperformed simpler and complex models that lacked proper fine-tuning.

To further improve this project, ensemble approaches combing DistilBERT with other models could be tested to further boost the robustness. Additionally, more hyperparameter tuning for the other tested transformer-based models could improve their performance. Overall, the project could be enhanced with external data incorporating stock price movement or tweet metadata like the timestamp or user info. This would provide additional predictive power and improve classification accuracy. Furthermore, the models could be fine-tuned on larger financial specific corpora or other models like full BERT or RoBERTa could be tested as well, as they are larger pretrained models.

This project highlights the importance of choosing the right architecture and models to domain-specific data in financial sentiment analysis.

## 8. REFERENCES

Kanerika Inc (2024). Named Entity Recognition. A Comprehensive Guide to NLP's Key Technology. Retrieved June 12, 2025. https://medium.com/@kanerika/named-entity-recognition-acomprehensive-guide-to-nlps-key-technology-636a124eaa46

Hutto, C. J. (2021). *Resources and dataset descriptions* [Documentation]. In *VADERSentiment*. Read the Docs. Retrieved June 12, 2025, from https://vadersentiment.readthedocs.io/en/latest/pages/resource_description.html

# 9. APPENDIX

*Figure A: Sentiment Distribution by Percentage for Top Organizations (NER)*

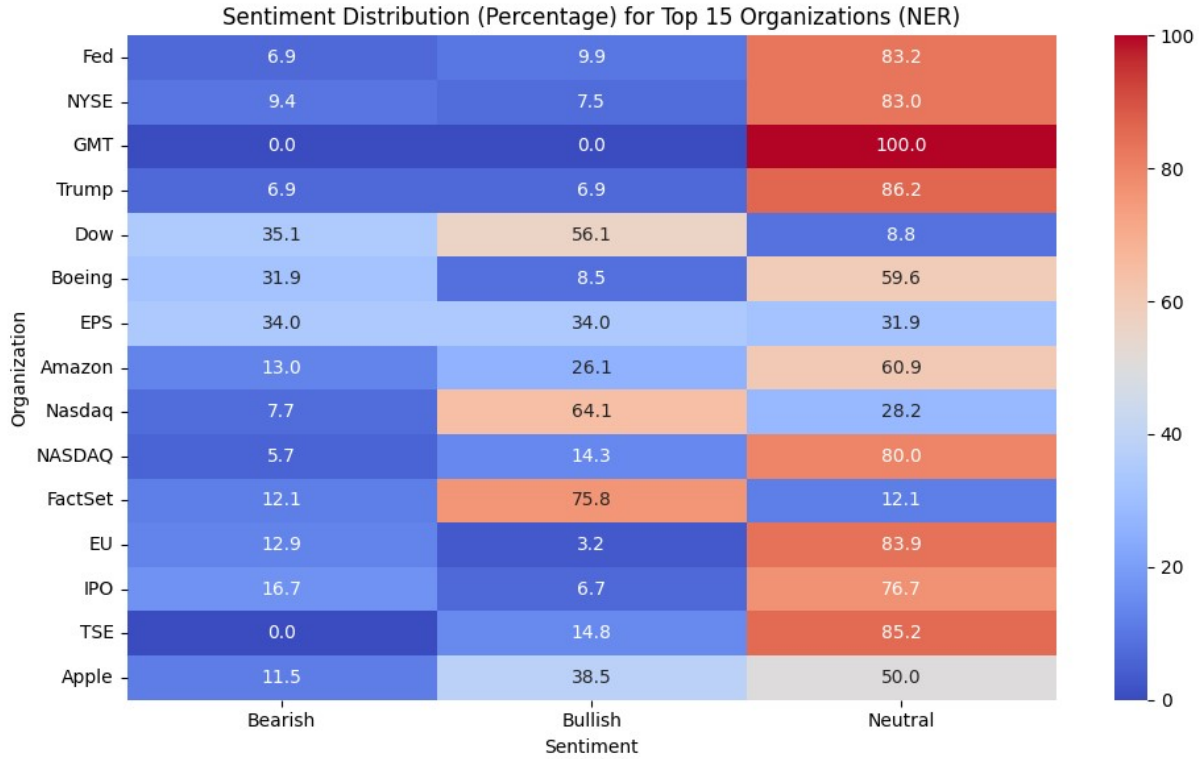


*Figure B: Sentiment Distribution by Percentage for Top GPEs (NER)*