UNIVERSITY OF LONDON

INTERNATIONAL PROGRAMMES

BSc Computer Science and Related Subjects

<u>CM3070 PROJECT</u>
<u>FINAL PROJECT REPORT</u>

Sentiment Analysis of Financial News for Stock Market Forecasting

Author: Yeo Mei Zhong
Student Number: 190428664
Date of Submission:
Supervisor: Mr Oo Kyaw Kyaw

Contents

1. Introduction

The stock market is greatly influenced by numerous factors, one significant factor being financial news. News about mergers, financial results, or geopolitical events like conflicts between countries can cause stock prices to soar or plummet. Traders and investors have depended on their instincts and past experiences to predict the impact of news on stock values.

Given the backdrop of an era rife with information, disinformation, and misinformation, the task of discerning the veracity and relevance of financial news becomes paramount for investors seeking to make informed decisions. By employing technology to analyse extensive news data and forecast its influence on the stock market, the accuracy of stock market predictions can be improved.

This project is based on the template of CM3060 Natural Language Processing, Fake News Detection, and will focus on the combination of sentiment analysis of financial news in the form of textual data and the time series data analysis of historical stock prices. In the context of financial markets, the role of sentiment analysis extends beyond mere interpretation of market moods to a crucial tool for navigating information reliability. Sentiment analysis, traditionally used for gauging the general sentiment of financial news to predict stock prices, is increasingly being recognized for its potential in fake news detection. By leveraging natural language processing techniques, sentiment analysis can identify inconsistencies, exaggerated claims, and unfounded conjectures often associated with fake news. This not only enhances the reliability of stock price predictions but also contributes to a more informed and discerning investment community, capable of distinguishing between fact and fabrication in the financial news landscape. The interaction between financial news and stock market fluctuations is the cornerstone of this project. The assumption of this project is that financial news sentiment has a direct impact on stock market movements and historical stock market movements are indicative of future movements. Only English news articles will be used.

The motivation behind the project is to enhance the efficiency and accuracy of social sentiment analysis in financial news and its application to stock prediction in real-time. It also allows for an additional layer of information to traditional financial models, which enhances the predictive power of current models. This helps reduce resources.

With a reliable way to predict stock prices, this project will provide a more data-driven approach to understanding stock market dynamics. This project aims to develop a model for sentiment analysis and time series data analysis that can accurately predict stock prices based on both numerical and textual data. Each model will be created in Jupyter Notebook and evaluated with appropriate performance metrics to determine its accuracy in predicting stock prices.

Word Count: 435 words

2. Literature Review

The integration of time series analysis with text mining and sentiment analysis has emerged as a notable area of research, in the context of financial market predictions. It aims to enhance the prediction of financial stock market behaviour. Predicting stock market trends is an arduous task as it requires thorough analysis of news events, historical data and understanding how new developments can impact stock prices [1].

The stock market is known to be volatile as it is susceptible to many non-economic factors, such as natural disasters or political decisions [2]. To address this challenge, recent forecasting research have constructed models which can predict future stock prices with enhanced accuracy. These models are made with machine learning and deep learning techniques to analyse and predict stock prices based on both numerical and textual data.

Sentiment analysis is a subfield of Natural Language Processing (NLP) that uses machine learning and computational linguistics to extract and interpret subjective information from source materials [3]. It includes opinion mining, emotion mining and ambiguity detection [4].

Traditional sentiment analysis methods typically involve two approaches, the lexicon-based approach, and the machine learning approach [5], to classify textual data into positive, neutral, or negative. The lexicon-based strategy includes tools like WordNet [6], SentiNet, and MPQA [7]. Lexicon-based NLP methods encompass Bag of Words and Term Frequency-Inverse Document Frequency (TF-IDF). On the other hand, the machine learning strategy, which can be further divided into supervised, semi-supervised, and unsupervised methods, features algorithms such as Support Vector Machines (SVM) [8], Multinomial Naïve Bayes Classifier [8] [9] and Logistic Regression [10]. These traditional methods endeavour to adapt to the changing data dynamics of languages and the intricate structural and cultural details found in short text formats, such as tweets. To efficiently handle such data, deep learning algorithms were developed.

Deep learning approaches are based on artificial neural networks (ANNs), or neural networks, a type of machine learning technique that derives its principles from mimicking the operations of the human brain [11]. Word embedding techniques such as Word2Vec and Glove2 assist in transforming text data into vectors that represent words, thereby enhancing the performance of automated language processing systems.

Transformers, a type of neural network architecture, was introduced by Vaswani et al. in 2017. Its self-attention mechanism was developed to focus on different parts of an input sequence when making predictions [12]. The Transformer architecture have further advanced the field of NLP. It is the basis of Bidirectional Encoder Representations from Transformers (BERT), a language model trained bidirectionally, introduced in 2018 by Devlin et al. BERT can learn the context of a word based on all its occurrences in a dataset, allowing it to understand the meaning of a word within its specific context.

In the research article by Karanikola et al. (2023), they explored and evaluated both classic machine learning models and deep learning models for sentiment analysis, trained using a dataset related to the finance domain. The research showed that deep learning models outperform the classic machine learning models, shown in Appendix A. In their study, the Robustly Optimised BERT Pretraining Approach (RoBERTa) model, an improved version of BERT by Liu, et al. (2019), has the best performance in sentiment analysis of textual data related to the finance domain.

In addition to BERT and RoBERTa models, FinBERT, a pre-trained NLP model based on BERT to analyse the sentiment of financial text, was introduced by D. Araci in 2019. FinBERT was built by using a large financial corpus to further train the BERT language model [13]. FinBERT was made by pre-training BERT with a dataset of 1.8 million news articles from Reuters' TRC2. The news articles were published by Reuters between 2008 and 2010. The FinBERT model is then evaluated against the Financial Phrase Bank [14] and FiQa Sentiment datasets [15], showing improved performance over the BERT model for financial domain texts, as illustrated in Appendix B. Nevertheless, the research article by Karanikola et al. (2023) challenges this by showing that it ranks as one of the least effective deep learning models in comparison to BERT and RoBERTa models, as evidenced in Appendix A. Since the FinBERT models in both studies were evaluated using identical financial datasets, the variation in performance outcomes could stem from differences in the data pre-processing methods. Araci (2019) did not include any data pre-processing methods with regards to the datasets used for the evaluation. Karanikola et al. (2023) pre-processed the datasets using duplicate removal, tokenization of texts, stop word removal, lemmatization, and lowercase conversion [16]. The lowercase conversion of the data pre-processing steps could have altered the meaning of texts in the dataset. For example, the word Apple which represents the name of a company could be changed to mean apple, a fruit. This would have changed the meaning of a sentence entirely, which would lead to inaccuracies in the evaluation results. Although various versions of FinBERT exist beyond the one presented by D. Araci in 2019, i.e. FinancialBERT [17] and FinBERT [18], the study by Karanikola et al. (2023) demonstrates that the RoBERTa and BERT models continue to outperform in sentiment analysis of texts in the financial domain.

Time series analysis is a statistical technique that deals with data that is ordered sequentially over time. There are several methods used in time series analysis. The concept of recurrent neural network (RNN), a type or ANN, was first introduced by David et al. in 1986. They proposed a model with connections that allows the network to retain information from previous inputs and use it to influence the processing of subsequent inputs [19]. However, Ge Li et al. (2019) mentioned that RNNs are prone to the issue of vanishing and exploding gradients when dealing with time series data. As such, Long Short-Term Memory networks can be used to counter this issue.

LSTM, first proposed by S. Hochreiter and J. Schmidhuber in 1997, is a type of RNN that is designed to address the issue of learning and retaining information over extended time intervals. Jethva et al. (2022) observed that LSTM models can retain data for a set amount of time, making it beneficial in handling time series data.

According to Jolhe et al. (2019), LSTM-based algorithms could enhance forecasts with an aggregate of 85%, compared to Autoregressive Integrated Moving Average (ARIMA) models. This is supported by Bagul et al., describing that the LSTM model in their study exceeds the performance of the ARIMA model, further reinforcing that the LSTM-RNN can identify non-linear structures in financial time series.

Chou (2021) presented and evaluated multiple models for stock prediction, covering numerous aspects of time series analysis and sentiment analysis. The study examined and evaluated multiple models for sentiment analysis and time series data analysis. The models developed are shown in Appendix C. The Model_8 in the study developed displayed the most promising results. It is a model with inputs historical stock prices, sentiment score and predicted stock prices, made using the BERT language model and convolutional LSTM (CNN-LSTM).

Li & Pan (2022) proposed a novel ensemble deep learning model for stock prediction based on stock prices and news, combining both textual and numerical data analysis. The study discovered that the blending ensemble deep learning model outperformed the best existing prediction model substantially using the same dataset.

While both studies explore time series and sentiment analysis, they offer insights into different approaches for stock market forecasting. Chou integrated time series analysis and sentiment analysis by individually developing each model and combining them, whereas Li and Pan focused on an ensemble deep learning approach. Chou compared the models created to one another, while experimenting with different window sizes in LSTM. Li and Pan compared the ensemble model to existing models and demonstrated that it could outperform them.

Chou adopted traditional NLP techniques such as word tokenization, stop-words removal, part-of-speech tagging, stemming and data transformation, while Li and Pan utilised Valence Aware Dictionary and Sentiment Reasoner (VADER) to generate sentiment scores. Albeit from diverse sources, both studies used news headlines and displayed different methods of obtaining sentiment scores for sentiment analysis. However, Chou only used one news source while Li and Pan used multiple news sources. This is due to the difference in the stock tickers of their choosing. Chou chose the stock prices of six individual companies, hence selecting news headlines related to the specific companies, while Li and Pan used the stock prices of the S&P 500 Index, therefore picking a general dataset of financial news headlines. Choosing news headlines related to the chosen stock ticker would accurately predict future stock prices, while choosing a diversified dataset of news headlines would cover the broader range of stocks that the S&P 500 Index covers.

Both studies used different evaluation metrics to evaluate the performance of the models developed. Chou discusses the limitations of Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) in stock forecasting due to high market volatility and introduces Mean Absolute Percentage Error (MAPE) as a more intuitive measure. On the other hand, Li and Pan compares the performance of the blending ensemble model with previous work and test models, demonstrating significant improvements in Mean Squared Error (MSE), Precision, Recall, F1-score, and Mean Direction Accuracy (MDA).

The studies by Chou (2021) and Li and Pan (2022) contribute to this project by demonstrating that it is possible to combine textual and numerical data models to improve stock predictions. They emphasise the importance of accurate textual data pre-processing for obtaining sentiment scores, then using the sentiment scores from news titles for sentiment analysis.

Chou (2021) and Karanikola et al. (2023) have provided valuable insights into the intricacies of model selection. Both studies have evaluated and compared each model created, showing explicitly which model would produce the most accurate predictions. They offer a better understanding of the architecture of each model and how each model performs. These ideas can be implemented into the project by guiding the selection of suitable models for both sentiment analysis and time series analysis, dataset selection, data pre-processing methods, and performance evaluation metrics. This would help improve the accuracy of stock predictions.

Word Count: 1,676 words

3. Project Design

This project appeals to individuals with an interest in analysing the combination of sentiment analysis of financial news and time series data analysis of historical stock prices, increasing the accuracy of forecasting future stock prices. These individuals can be professionals trading as their primary source of income, either independently or for an organisation, or hobbyists.

In the stock market, news with negative sentiment typically has a positive correlation to the selling of stocks, and vice versa. News events such as poor earnings, failures in corporate governance, macroeconomic and political instability may result in selling pressure, leading to a decline in stock prices. Conversely, favourable earnings disclosures, the launch of new products, corporate takeovers, and optimistic economic signals contribute to buying pressure, resulting in a rise in stock prices. However, negative developments for certain stocks can be beneficial for others. For instance, the announcement of an impending hurricane may lead to a drop in utility stocks, as traders anticipate the expensive emergency measures and repair efforts. The impact on insurance stocks also tends to be negative, varying with the storm's severity. On the other hand, shares of home improvement stores are likely to increase, with expectations of heightened sales in the following months [20].

To stay informed about significant developments, they rely on various news sources, ensuring access to top newswires, continuous updates from media organizations, and specialized software that diligently monitors news sources for impactful news stories [21]. Hence, individuals who often involve themselves in stock price trading for a profit would have a significant interest in a predictive model for stock prices, as it could provide them with a competitive advantage in their investment decisions.

Dataset Selection

Due to the lack of open-source historical financial news data, the financial news headlines dataset scraped from CNBC, Guardian Business, and Reuters by Lucas Pham on Kaggle.com will be used. It was last updated in 2020. The dataset contains headlines, last updated date, and the preview text of articles for CNBC and Reuters and headlines and last updated date for Guardian Business. Reuters is an international news organisation founded in 1851, serving professionals in the financial, media and corporate markets. CNBC is a world leader in business news and real-time financial market coverage, making the news headlines trustworthy.

Data from CNBC were scraped from 22nd Dec 2017 to 17th Dec 2020, data from Reuters were scraped from 20th Mar 2018 to 18th Jul 2020 and data from Guardian Business were scraped from 17th Dec 2017 to 18th Jul 2020. As the news headlines data is scraped directly from the sites, the data largely is reliable. To fully utilise the three sets of news headlines, the news headlines for all three datasets ranging from dates 17th Dec 2017 to 18th Jul 2020 will be used for sentiment analysis.

The stock price data are extracted from Yahoo! Finance. It includes Open, High, Low, Close, Adjusted Close Prices and trade volume. The daily adjusted closing price will be used for time series analysis as it accurately reflects the past performance of the stock. The adjusted closing price is determined by amending the closing price of a stock after taking any corporate actions into consideration [22]. As the news headlines dataset is of various financial news about different stock tickers, the Standard & Poor's (S&P) 500 Index will be used. The S&P 500 index is considered a leading indicator of the performance of the overall stock market [23].

Workflow

Figure 1 demonstrates a detailed workflow of how the models are designed to work together. The sentiment analysis model is fine-tuned with the training and validation sets to predict the sentiment labels of news headlines data. The predicted labels will be used to evaluate both BERT and RoBERTa models against the actual labels. A comparison between the evaluation of both models will be made. Thereafter, the predicted output of the RoBERTa is appended to the test set of the stock price data. The LSTM model is fine-tuned with the training and validation sets to predict the stock price of the dates in the test set. The predicted stock prices will be used to evaluate the LSTM model against the actual stock prices.

Data Preparation

The news headlines dataset is unlabelled, hence, the sentiment scores of the news headlines dataset will be obtained with FinVADER, a fine-tuned version of Valence Aware Dictionary and sEntiment Reasoner (VADER) in the financial domain. VADER is a lexicon and rule-based sentiment analysis tool for sentiments expressed in social media [24]. It classifies textual data into positive, neutral, or negative sentiments, showing how positive or how negative the sentiment is. Using VADER as its base, FinVADER includes 2 finance lexicons, SentiBignomics and Henry’s word list [25]. SentiBigNomics is an extensive financial lexicon tailored for aspect-based sentiment analysis, comprising roughly 7300 terms each assigned a polarity score that varies between -1 and 1 [26]. Meanwhile, Henry's lexicon includes 189 words identified within company earnings press releases. [27].

The financial news headlines are pre-processed simply by removing extra spaces between texts as they are already short and concise. Punctuations in headlines can increase the intensity of the sentiment expressed in the text and the sentiment expressed can be affected using capital letters. Degree modifiers such as “very” or “extremely” can also increase the intensity of the sentiment expressed.

FinVADER, like VADER, returns the sentiment scores according to four categories, negative, neutral, positive, and compound. The compound score is computed by normalising the negative, neutral, and positive scores, which is the overall sentiment of the text, ranging from -1 to 1. To get a general estimate of how positive or negative the headlines are, the compound scores derived are rounded to 1 decimal place. This is to prevent the news headlines from being classified into wrong labels, resulting in inaccuracies while training the model. For example, a headline with a score of 0.5001 will be classified as Very Positive, even though the score is closer to being Positive. The rounded compound scores will be sorted into 5 classes, then assigned a label for training the model. The classes and labels are as follows, Very Negative: 0, Negative: 1, Neutral: 2, Positive: 3, and Very Positive: 4. For scores less than -0.5, they are labelled as Very Negative. For scores between -0.5 and 0, including -0.5 values, they are labelled as Negative. For scores that are 0, they are labelled as Neutral. For scores between 0 and 0.5, including 0.5, they are labelled as Positive. For scores above 0.5, they are labelled as Very Positive.

Both the news headlines dataset and the stock price dataset will be split into training, validation, and test sets. The stock price data is first split into 80%, 10% and 10% for the training, validation, and test sets respectively. The test set for the news headlines data is divided according to the dates of the test set for the stock price data. This is done to prevent the news headlines data for the test set from being utilized in training and validating the sentiment analysis models. The remaining data will be split into training and validation sets, with a 90% and 10% split.

Sentiment Analysis Models

Deep learning models have shown to perform better than classic machine learning models in sentiment analysis, as depicted in Appendix A. Hence, the RoBERTa language model will be used for the sentiment analysis of news headlines. Since the RoBERTa language model is based on the BERT language model, a BERT model will be first created as a benchmark for the RoBERTa model. The RoBERTa and BERT model are imported from the transformers library. Developed by Hugging Face, the transformers library is a Python package that provides general-purpose architectures such as RoBERTa and BERT for Natural Language Understanding (NLU) and Natural Language Generation (NLG).

The models will be compiled using the Adam optimizer and configured with a loss function and metrics of Sparce Categorical Cross entropy and Sparse Categorical Accuracy. These functions will be imported from Tensorflow with Keras. Sparse Categorical Cross-entropy is selected as the loss function due to the nature of the text data, which is not one-hot encoded. Instead, each label is represented as an integer corresponding to a specific class. The default configurations will be employed.

Tokenizers for Sentiment Analysis Models

A tokenizer is used to transform news headlines into a format compatible with the model, since BERT and RoBERTa models necessitate their inputs to be specifically tokenized and encoded. Special tokens such as '[CLS]' and '[SEP]' are appended into each news headlines data. '[CLS]', which stands for classification, is added at the beginning of an input sequence. The representation obtained from the '[CLS]' token is used to make the prediction as to which label the sentence belongs to. '[SEP]', which stands for separator, is used to mark the separation between two sentences for the input sentences.

For the BERT model, the BertTokenizer class from transformers library is used to load a pre-trained BERT tokenizer of "bert-base-cased". For the RoBERTa model, there are 2 tokenizer classes to load pre-trained tokenizers for the RoBERTa model, RobertaTokenizer and RobertaTokenizerFast. The primary distinction between the two tokenizer classes lies in their implementation and efficiency. RobertaTokenizer, purely Python-based, operates without external dependencies, while RobertaTokenizerFast leverages the Rust-based Tokenizers library, enhancing its speed and efficiency in text tokenization. Additionally, RobertaTokenizerFast offers advanced features, such as alignment tracking beneficial for tasks like Named-Entity Recognition (NER). Hence, RobertaTokenizerFast class from Transformers library is used to load the tokenizer, "roberta-base", for the RoBERTa model.

The case-sensitive version of the pre-trained BERT model, "bert-base-case", was used. Using a case-sensitive model is crucial as news headlines that affect stock price movements generally include entity names, such as Apple, Google, etc. This will help the model differentiate between entities and non-entities. The pre-trained RoBERTa model initialised, "roberta-base", is case-sensitive by default. The models will be trained on a down-stream task to be used for predictions and inference.

Time Series Model

A stacked LTSM architecture will be used for time series analysis. This involves integrating multiple LSTM layers alongside Dense layers within the model framework. The advantage of this stacked approach enables the model to discern complex, hierarchical patterns in the data, leading to improved prediction performance. In this case, the model will learn the fluctuating patterns of stock prices. The LSTM and Dense layers will be imported from Keras. The inputs for the LSTM model consist of 2 features, an overall label of the news headlines and a stock price, for each timestep in a sample.

The LSTM model will be trained using the Adam optimizer and Mean Squared Error (MSE) loss function. The reason for using MSE loss function is that it is a good choice for regression problems due to its convex nature and computational efficiency. The model will be trained for 50 epochs with a batch size of 32.

Performance Metrics

To evaluate the performance of the RoBERTa and BERT models, accuracy, recall, precision, and F1-score will be employed. Each of these metrics offers a unique perspective on the effectiveness of the models, collectively provide a comprehensive understanding of the models' classification capabilities and overall predictive accuracy.

The performance of the LSTM model will be assessed with the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Directional Accuracy (MDA), and the R2 score. These metrics collectively offer varied insights into the accuracy and predictive performance of the model.

Plan of Work

Figure 2 depicts the plan of work from preliminary report to final report. The preliminary report should be completed and submitted by 8th Jan 2024. As the preliminary report includes a feature prototype, the Sentiment Analysis and Stock Prediction Models are in the making and will be completed between weeks 12 and 18. The models made could be written up in for the final report during the development. Thereafter, the models will be evaluated, and the evaluation will be included in the report. The report will be polished and submitted by 11th Mar 2024.

Word Count: 2,000 words

4. Implementation

The RoBERTa model is trained with the purpose of predicting the sentiment scores of news headlines of each day. The BERT model is trained as a benchmark for the RoBERTa model. The LSTM model is trained with the purpose of predicting stock price movements with the help of historical stock price data and sentiment scores of the news headlines.

Data Preparation

Figure 4 outlines the initial step of importing news headlines data into a Pandas DataFrame within a Jupyter Notebook. The dataset undergoes initial cleansing to remove null values, followed by preprocessing to eliminate special characters, punctuation, and leading whitespaces. Subsequently, the news headlines are organized chronologically by 'Time'.

Figure 5 presents the application of FinVADER to calculate the compound scores for each news headline. This procedure is carried out similarly for news headlines datasets from Reuters, Guardian Business, and CNBC, each stored in distinct CSV files.

Figure 6 reveals that there are 32,770 news headlines in the Reuters dataset, 2,800 news headlines in the CNBC dataset and 17,760 news headlines in the Guardian Business dataset. Altogether, these datasets contribute to a total of 53,330 news headlines.

The DataFrames from each news source are then merged. In Figure 7, the compound scores derived from FinVADER are adjusted to two decimal places. Subsequently, in Figure 8, these adjusted FinVADER scores are categorized into labels: 0 (Very Negative), 1 (Negative), 2 (Neutral), 3 (Positive), and 4 (Very Positive), with the distribution being 1,892 headlines for Very Negative, 12,578 for Negative, 26,773 for Neutral, 10,415 for Positive, and 1,672 for Very Positive.

Figure 10 focuses on the importation of SPX historical price data into a Pandas DataFrame within a Jupyter Notebook. This data is partitioned into training, validation, and testing sets, following an 80%, 10%, 10% division, respectively. Out of 649 stock price data points, 519 are allocated for training, 65 for validation, and the remainder for testing.

Each input sample for the LSTM includes both the stock price data and the sentiment label corresponding to that date. The sentiment label is first combined with the stock price data. This is achieved by creating a DataFrame that spans from the earliest to the latest date across the training, validation, and test sets of the stock price data. This DataFrame is populated with the stock price data for matching dates and the weighted average sentiment label of news headlines for those dates.

Instances occur where dates may have stock price data but lack news headline data, and vice versa. In such cases, sentiment scores lacking corresponding stock price data are aggregated with the sentiment scores of the subsequent date that includes stock price data. Subsequently, a new weighted average for these sentiment scores is computed and assigned as the sentiment score for that date. For instance, 30th Dec 2017, 31st Dec 2017, and 1st Jan 2018 are dates which have news headlines data but lack corresponding stock price data. The sentiment scores for these dates are combined with the sentiment score for the subsequent date that has stock price data, 2nd Jan 2018. A new weighted average for these combined sentiment scores is then calculated and assigned as the sentiment score for 2nd Jan 2018, to reflect the sentiment more accurately for that date. Dates missing stock price data are excluded in the dataset, and dates without news headlines data are assigned a sentiment score of 0. This process is illustrated in Figures 11 and 12, which detail the function used to preprocess the stock data and sentiment labels to create LSTM model inputs, applying it across the training, validation, and test datasets.

Figures 13 and 14 detail the division of news headlines data into training, validation, and testing sets. The testing set for the news headlines is aligned with the testing set for the stock price data to maintain the integrity of the test set, ensuring it is not influenced by the training and validation of the sentiment analysis models. This strategy is designed to avert the risk of model overfitting, thereby enhancing the model's ability generalize to new and unseen data.

For dividing the training and validation sets, the train_test_split function from the model_selection module of the Scikit-learn library is utilized, applying a division ratio of 90% for training and 10% for validation of the remaining data. The use of the stratify parameter within train_test_split guarantees that the label distribution within the training and validation sets mirrors the overall distribution, ensuring a consistent label distribution across splits. A random_state with a seed value of '12345' is specified to ensure the reproducibility of the results.

LSTM Inputs

To generate the inputs for the LSTM model, the DataFrame consisting of the stock price data and news headlines labels is ran through a MinMaxScaler from the sklearn.preprocessing module in Python's Scikit-learn library. The MinMaxScaler scales and translates each feature individually in a dataset such that it is in the default range of 0 and 1. The formula for computing each feature is as below.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where min, max are the defined feature range.

This approach is adopted to enhance the learning efficiency of the neural networks within the LSTM model, as it deals with input data characterized by small, similar values. It also helps to mitigate the issue of vanishing or exploding gradients, where large input values can cause excessively large gradient values and small input values can result in minimal gradient values.

BERT/RoBERTa Inputs

Two separate functions were developed to transform DataFrames into TensorFlow datasets, enabling the BERT and RoBERTa models to process the inputs, shown in Figures 16 and 18. This conversion is necessary because the models are built using the TensorFlow API. The distinction between the input requirements for the RoBERTa and BERT models lies in that RoBERTa does not necessitate token_type_ids for its inputs. This necessitates the creation of two separate functions for converting DataFrames into TensorFlow datasets. Contrary to BERT, which differentiates between sentences by allocating a token_type_id to each one, RoBERTa processes the input as a unified sequence of tokens, irrespective of the sentence count. This streamlining of the input process simplifies the architecture of the RoBERTa model and contributes to its enhanced performance.

Figure 16 illustrates the process of encoding a DataFrame with news headlines into inputs for the RoBERTa model. Using the RoBERTa tokenizer, the special tokens '[CLS]' and '[SEP]' are inserted at the start and end of each sentence, respectively. A maximum length of 128 tokens is specified, with '[PAD]' tokens added to sentences falling short of this length and sentences exceeding it being truncated. Following this, the tokenizer outputs a TensorFlow tensor, where the input_ids and attention_mask of the encoded data are retained.

Figure 17 demonstrates how the labels associated with the news headlines in the training and validation sets are transformed into TensorFlow tensors using the tf.convert_to_tensor function from TensorFlow. By employing the tf.data.Dataset.from_tensor_slices, these label tensors are merged with the encoded news headlines tensors produced by the function depicted in Figure 16, resulting in a TensorFlow dataset compatible with the RoBERTa model. Each dataset element represents a single slice of the tensors; for example, an element in the dataset is a pair, with one slice coming from train_headlines_inputs and another from train_labels_inputs.

Figure 18 outlines the process of converting a DataFrame into inputs for the BERT model. Each sentence is first converted into an InputExample object, which is a data class in the Transformers library designed to encapsulate a single training or test example. The InputExample object comprises the sentence intended for encoding along with its associated label. Subsequently, these InputExample objects are tokenized using the BERT tokenizer. Like RoBERTa's approach, special tokens '[CLS]' and '[SEP]' are prepended and appended to each sentence. A maximum token length of 128 is established, with sentences not meeting this threshold being padded with '[PAD]' tokens and those surpassing the limit being truncated. InputFeatures objects are generated from each InputExample, encompassing the encoded input_ids, attention_masks, token_type_ids, and labels. InputFeatures is another data class in the Transformers library that represents a single set of features for the data. The generator function iterates over each InputFeatures object, producing a tuple where the first element comprises the input_ids, attention_masks, and token_type_ids, and the second element holds the corresponding label. The culmination of this process involves utilizing the output from the generator function to assemble a TensorFlow dataset, which sequentially yields pairs of input features and their associated labels.

To prevent the model from learning any patterns from the sequence of the data, the elements within the training set are randomized using a buffer size of 100. The data from both the training and validation sets are then organized into batches of 32. This batching allows the model to process multiple elements simultaneously, potentially accelerating the training process. This process is utilized for the inputs of both the RoBERTa and BERT models, as shown in Figures 17 and 19.

Models

The BERT and RoBERTa models are trained over 10 epochs, with early stopping implemented. This is a form of regularisation technique to prevent the models from overfitting. The model training is stopped as soon as the error on the validation set increases, i.e., when the model starts to overfit on the training data. The early stopping is implemented as a callback function during the training of the BERT and RoBERTa models and monitors the validation loss with a patience of 2. The training of the models will continue for 2 more epochs after the point which the validation loss stops improving. If the validation loss does not improve within the 2 epochs, the model training is then stopped and the weights for the models with the best validation loss is restored.

Figure 21 presents the training and validation accuracies for the RoBERTa model, illustrating a decelerating increase up to the 6th epoch. This pattern suggests that while the model continues to extract knowledge from the training dataset, the incremental learning gain diminishes as the available new information in the training data depletes. Figure 22 delineates the model's training and validation losses. A steady decline in training loss evidences the model's ongoing learning and its progressively enhanced performance in predictions over epochs. In contrast, an ascent in validation loss commencing from the 4th epoch signals the emergence of overfitting, indicating that further training may detract from model performance. Consequently, the application of the best model weights is reinstated at the 5th epoch, a decision justified by the implementation of early stopping, as discussed in Figure 20.

Regarding the BERT model, Figure 23 portrays the trends in training and validation accuracies, where training accuracy ascends at a reduced rate and validation accuracy commences its decline from the 2nd epoch. This observation suggests the model's premature overfitting from the 2nd epoch onwards. The depiction of training and validation losses in Figure 25 shows a continuous decrease in training loss, affirming that the model persists in learning from the training data. However, the increase in validation loss starting from the 2nd epoch indicates overfitting, necessitating the cessation of training. To address this, the optimal weights are reapplied at the 4th epoch, facilitated through early stopping, as elucidated in Figure 23.

Word Count: 1,854 words

5. Evaluation

Data Definitions

Term: Classification Report

Definition: The classification report offers essential metrics concerning the model's performance for each class, in addition to the model's aggregate performance.

Term: Precision

Definition: Precision measures the proportion of correctly predicted positive observations out of all positive predictions made. A higher precision indicates fewer false positives.

Term: Recall

Definition: Recall calculates the fraction of correctly predicted positive observations from all actual positives in a class, with higher recall indicating fewer false negatives.

Term: Precision

Definition: Precision measures the proportion of correctly predicted positive observations out of all positive predictions made. A higher precision indicates fewer false positives.

Term: Precision

Definition: Precision measures the proportion of correctly predicted positive observations out of all positive predictions made. A higher precision indicates fewer false positives.

Term: F1-Score

Definition: The F1-Score, the harmonic mean of Precision and Recall, offers a balanced view of classification accuracy, particularly for imbalanced classes.

Term: Macro Average

Definition: The macro average calculates the metric for each class independently and then takes the average, giving equal weight to each class.

Term: Weighted Average

Definition: The weighted average calculates the metric for each class independently and then adds them together depending on the number instances of each class, treating all instances equally.

Term: F1-Score

Definition: The F1-Score, the harmonic mean of Precision and Recall, offers a balanced view of classification accuracy, particularly for imbalanced classes.

Term: Confusion Matrix

Definition: In a confusion matrix, the rows correspond to the actual categories, while the columns reflect the predicted categories. The diagonal entries show the count of instances correctly predicted to match their true labels, whereas the off-diagonal entries indicate instances incorrectly classified. A higher count in the diagonal cells of the confusion matrix signifies a larger number of accurate predictions.

Term: Mean Squared Error (MSE)

Definition: The MSE represents the average of the squared differences between the observed and forecasted values. A lower MSE value is indicative of superior model performance.

Term: Mean Squared Error (MSE)

Definition: The MSE represents the average of the squared differences between the observed and forecasted values. A lower MSE value is indicative of superior model performance.

Term: Root Mean Squared Error (RMSE)

Definition: The RMSE is the square root of the Mean Squared Error. It measures the standard deviation of the residuals. This metric further refines the understanding of model accuracy, with a lower RMSE score suggesting better performance.

Term: Mean Absolute Error (MAE)

Definition: The MAE is the average of the absolute differences between the actual and predicted values. The aim is to quantify the average error magnitude, with a lower MAE score signifying enhanced model accuracy.

Term: Mean Directional Accuracy (MDA)

Definition: The MDA is the percentage of predictions that correctly predict the direction of change of the actual values. A higher MDA denotes superior model performance.

Term: R2 Score

Definition: The R2 score, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges between 0 and 1, with a score closer to 1 indicating near-perfect prediction accuracy.

The classification report for the RoBERTa model is detailed in Table 1, encompassing the metrics of Precision, Recall, F1-Score, and Accuracy.

Table 1: Classification Report of RoBERTa model.

	Precision	**Recall**	**F1-Score**	**Support**
Very Negative	0.80	0.81	0.81	220
Negative	0.95	0.92	0.93	1643
Neutral	0.96	0.98	0.97	3111
Positive	0.93	0.91	0.92	1238
Very Positive	0.82	0.92	0.87	168
Accuracy			0.94	6380
Macro Average	0.89	0.92	0.90	6380
Weighted Average	0.94	0.94	0.94	6380

The precision rates were 80%, 95%, 96%, 93%, and 82% for ‘Very Negative’, ‘Negative’, ‘Neutral’, ‘Positive’, and ‘Very Positive’ classes respectively. The macro average precision score across all classes is 89% while the weighted average precision stands at 94%.

The recall rates are 81% for 'Very Negative', 92% for 'Negative', 98% for 'Neutral', 91% for 'Positive', and 92% for 'Very Positive' classes. The macro average Recall is 92%, and the weighted average Recall is 94%.

The F1-scores are 81% for 'Very Negative', 93% for 'Negative', 97% for 'Neutral', 92% for 'Positive', and 87% for 'Very Positive'. The macro and weighted average F1-scores are 90% and 94%, respectively.

Support denotes the actual number of occurrences for each class within the dataset: 220 'Very Negative', 1643 'Negative', 3111 'Neutral', 1238 'Positive', and 168 'Very Positive' headlines. The model's overall accuracy, indicating the proportion of total correct predictions, is 94%.

Figure 28 presents a confusion matrix summarizing the prediction outcomes from the RoBERTa model. The model has predicted 178 headlines correctly for the ‘Very Negative’ class, 1505 headlines for the ‘Negative’ class, 3037 headlines for the ‘Neutral’ class, 1129 for the ‘Positive’ class and 154 headlines for the ‘Very Positive’ class. The remaining headlines were predicted incorrectly into different classes.

In conclusion, the RoBERTa model exhibits commendable performance in the categorization of news headlines across various labels, achieving an overall accuracy, F1-score, precision and recall of 94.09%, 94.08%, 94.12% and 94.09% respectively. This is depicted in Figure 27. The model's most effective classification is observed within the 'Neutral' category, where it benefits from a substantial training dataset of news headlines, achieving Precision, Recall, and F1-score of 96%, 98%, and 97%, respectively. Conversely, its efficacy in distinguishing 'Very Negative' or 'Very Positive' headlines is somewhat diminished, attributable to the limited number of examples in these categories. This is evidenced by the scores for 'Very Negative' headlines at 80% Precision, 81% Recall, and 81% F1-score, and for 'Very Positive' headlines at 82% Precision, 92% Recall, and 87% F1-score.

The classification report for the BERT model is presented in Table 2.

Table 2: Classification Report of BERT model.

	Precision	**Recall**	**F1-Score**	**Support**
Very Negative	0.81	0.80	0.81	220
Negative	0.92	0.93	0.93	1643
Neutral	0.97	0.97	0.97	3111
Positive	0.93	0.92	0.92	1238
Very Positive	0.93	0.85	0.88	168
Accuracy			0.94	6380
Macro Average	0.91	0.89	0.90	6380
Weighted Average	0.94	0.94	0.94	6380

For the ‘Very Negative’, 'Negative', 'Neutral', 'Positive', and 'Very Positive' classes, the precision rates were 81%, 92%, 97%, 93%, and 93%, respectively. The macro average for precision is reported at 91%, with a weighted average of 94%. In terms of recall, the model accurately recognised 80%, 93%, 97%, 92%, and 85% instances respectively. The macro average recall rate stands at 89%, while the weighted average recall rate is 94%. The F1-scores observed for the 'Very Negative,' 'Negative,' 'Neutral,' 'Positive,' and 'Very Positive' classes are 82%, 94%, 96%, 92%, and 88%, respectively, with a macro average of 91% and a weighted average of 94%. The overall accuracy of the BERT model is reported at 94%.

The confusion matrix, illustrated in Figure 30, provides a detailed breakdown of the model's predictive performance, correctly classifying 176 'Very Negative,' 1534 'Negative,' 3008 'Neutral,' 1142 'Positive,' and 143 'Very Positive' headlines. Misclassifications across different categories are also accounted for in the matrix.

In essence, the BERT model has exhibited notable proficiency in classifying news headline data, attaining an overall precision, accuracy, and F1-score of 94.09%, and a recall rate of 94.08%, as depicted in Figure 29. Like the RoBERTa model, the BERT model demonstrates effective performance in categorizing headlines within the 'Neutral' classification but exhibits limitations in accurately classifying headlines into the 'Very Negative' and 'Very Positive' categories. This performance disparity is supported by the precision, recall, and F1-score metrics detailed in Table 2. Specifically, the 'Neutral' category achieves uniform scores of 97% across precision, recall, and F1-score metrics. In contrast, the 'Very Negative' category presents scores of 81% for precision, 80% for recall, and 81% for F1-score, while the 'Very Positive' category registers scores of 93% for precision, 85% for recall, and 88% for F1-score.

The comparative analysis of the RoBERTa and BERT models a comparable level of effectiveness in the classification of news headline data, with minimal differences in their performance indicators. This comparison suggests that both models are equally capable in executing multi-class classification tasks.

Despite the similarity in performance metrics, it is important to highlight the efficiency of the RoBERTa model in terms of training time. The RoBERTa model requires approximately half the time per epoch for training compared to the BERT model, with the former taking about one hour and the latter approximately two hours for each training epoch.

To summarise, both models demonstrate robust performance in the classification of unseen news headlines, particularly within the 'Neutral' category. However, their performance in distinguishing between the extreme sentiments of 'Very Negative' and 'Very Positive' is less pronounced.

The LSTM model boasts the following evaluation metrics in Table 3.

Table 3: Evaluation Metrics of LSTM model.

Performance Metrics	**Score**
Mean Squared Error (MSE)	0.0083
Root Mean Squared Error (RMSE)	0.0940
Mean Absolute Error (MAE)	0.0718
Mean Directional Accuracy (MDA)	0.3098
R2 score	0.8812

The LSTM model has an MSE of 0.0083, an RMSE of 0.0940, and an MAE of 0.0718. The model scored 0.3098 in MDA, which reflects on its capacity to predict directional changes accurately. This model has demonstrated a commendable R2 score of 0.8812, which suggests a high level of predictive accuracy.

Overall, the LSTM model shows outstanding effectiveness across various metrics such as MSE, RMSE, MAE, and R2 score, highlighting its capability to predict values accurately. However, the Mean Directional Accuracy presents a limitation, indicating a reduced efficacy in forecasting the direction of change in the target variable. Figure 31 presents a comparison of the predicted results generated by the LSTM model against the actual values of the test set. This evaluation underscores the model's strengths in predictive accuracy while also pointing to areas for potential improvement in directional forecasting.

Word Count: 1,434 words

6. Conclusion

As a final point, the implemented models, LSTM, BERT and RoBERTa, have demonstrated strong performance across the assessed evaluation metrics. As anticipated, the RoBERTa model outperforms the BERT model on the selected news headlines dataset, though the improvement is marginal. Given that both models were trained using their default settings, additional fine-tuning could potentially enhance the accuracy of each model. Notably, the RoBERTa model also benefits from a substantially quicker training period compared to the BERT model.

The deep learning models, RoBERTa and BERT, could be configured for regression output. This is where the model is trained for text data without being classified into different classes and outputs a sentiment score between the range of -1 and 1. This would provide a better sentiment score compared to integer labels. The models could be adapted for Named-Entity Recognition as news headlines regarding stock prices often mention organization names relevant to the article. This adjustment would enhance model accuracy by enabling the algorithm to identify names, organizations, or locations more precisely, thereby improving its ability to accurately assess the sentiment score of a text.

A continuously learning model could be implemented, whereby the model keeps learning from current news headlines and stock price data to predict future stock price movements. This would require the model to truncate data that is older than a certain period. For example, the model could learn from data from 1 January 2024 to 31 January 2024 to predict the stock price movement for 1 February 2024. The model would then truncate the data for 1 January 2024 and learn from 2 January 2024 to 1 February 2024 to predict the stock price movement for 2 February 2024. Models could also be designed to forecast stock price fluctuations on an hourly basis, rather than daily. However, such models demand significant computational resources for training, attributed to the heightened number of timesteps involved.

A larger dataset of news headlines could be implemented with more data labelled as 'Very Positive' and 'Very Negative' so that the sentiment analysis models could predict the sentiment of varying headlines with more accuracy and precision. The news headlines data could be collected from more sources. However, this would mean that there might be a clash in news headlines data, whereby different news outlets report of the same news, causing the models to be trained on repeat news headlines. The model could also be trained on different variations of training, validation, and test splits of the dataset. A higher amount of data in the training set could help the model identify more patterns or learn the textual context between more words to perform better on unseen data.

The sentiment analysis models may not work as well when there are anomalies. When the coronavirus disease 19 (COVID-19) pandemic first started in 2020, the world saw a global recession as governments implemented lockdowns to contain the spread of the disease. This led to a significant increase in stock prices for certain sectors, including healthcare, technology, and e-commerce, as consumer behaviour underwent substantial changes. With consumers preferring online pharmacies and shopping platforms to physical establishments, these industries benefited. Conversely, sectors impacted by pandemic-induced restrictions, such as travel, hospitality, and traditional retail, experienced declines in their stock valuations. This could necessitate manual adjustments to the deep learning models for forecasting future stock prices, as the models trained prior to the COVID-19 pandemic may no longer be suitable for predicting post-pandemic stock market trends [28].

There are many factors that affect stock price movements, such as government policies, investor sentiment, industry performance and more. News headlines are just one of the many factors that stockholders closely observe. With more research and data used to train different types of models that could comprehend the other factors, the stock price predictions could be more accurate and precise.

Word Count: 635 words

Total Word Count: 8,034 words

7. References

[1] S. Usmani and J. A. Shamsi, "News sensitive stock market prediction: literature review and suggestions," PeerJ Computer Science, Karachi, 2021.

[2] W. A. Mohammed, "Challenges of Stock Prediction," in *Valuation Challenges and Solutions in Contemporary Businesses*, S. D. Köseoğlu, Ed., IGI Global, 2020, pp. 234-252.

[3] J. F. Sánchez-Rada and C. A. Iglesias, "Social context in sentiment analysis: Formal definition, overview of current trends and framework for comparison," *Information Fusion,* vol. 52, pp. 344-356, 2019.

[4] S. Assem and S. Alansary, "The Development of Sentiment Analysis from a Linguistic Perspective," *Egyption Journal of Language Engineering,* vol. 9, no. 2, pp. 40-52, 2022.

[5] F.-E. Lagrari and Y. Elkettani, "Traditional and Deep Learning Approaches for Sentiment Analysis: A Survey," *Advances in Science, Technology and Engineering Systems Journal,* vol. 6, no. 5, pp. 01-07, 2021.

[6] G. A. Miller, "WordNet: a lexical database for English," *Communications of the ACM,* vol. 38, no. 11, pp. 39-41, 1995.

[7] L. Deng and J. Wiebe, "MPQA 3.0: An Entity/Event-Level Sentiment Corpus," *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL,,* pp. 1323-1328, 2015.

[8] K. Dhola and M. Saradva, "A Comparative Evaluation of Traditional Machine Learning and Deep Learning Classification Techniques for Sentiment Analysis," *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence),* pp. 932-936, 2021.

[9] N. Zainuddin and A. Selamat, "Sentiment analysis using Support Vector Machine," *2014 International Conference on Computer, Communications, and Control Technology (I4CT),* pp. 333-337, 2014.

[10] A. Tyagi and N. Sharma, "Sentiment analysis using logistic regression and effective word score heuristic," *International Journal of Engineering and Technology (UAE),* vol. 7, no. 2, pp. 20-23, 2018.

[11] J. Zou, Y. Han and S.-S. So, "Overview of Artificial Neural Networks," in *Livingstone, D.J. (eds) Artificial Neural Networks. Methods in Molecular Biology™, vol 458,* Humana Press, 2008, pp. 14-22.

[12] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. v. Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, M. A. Rush, S. Gugger, M. Drame, Q. Lhoest and A. M. Rush, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," *CoRR,* 2019.

[13] D. Araci, "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models," arXiv, Amsterdam, 2019.

[14] P. Malo, A. Sinha, P. Korhonen, J. Wallenius and P. Takala, "Good debt or bad debt: Detecting semantic orientations in economic texts," *Journal of the Association for Information Science and Technology,* vol. 65, 2014.

[15] M. Maia, S. Handschuh, A. Freitas, B. Davis, R. Mcdermott, M. Zarrouk, A. Balahur and R. Mc-Dermott, "WWW '18: Companion Proceedings of the The Web Conference 2018," *Companion Proceedings of the The Web Conference,* pp. April 23-27, 2018.

[16] C. P. Chai, "Comparison of text preprocessing methods," *Natural Language Engineering,* vol. 29, no. 3, p. 509–553, 2023.

[17] A. Hazourli, "FinancialBERT - A Pretrained Language Model for Financial Text Mining," *ResearchGate,* 2022.

[18] A. H. Huang, H. Wang and Y. Yang, "FinBERT: A Large Language Model for Extracting Information from Financial Text," *Contemporary Accounting Research,* vol. 40, no. 2, pp. 806 - 841, 2023.

[19] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature 323,* pp. 533-536, 1986.

[20] B. Beers and C. Potters, "How the News Affects Stock Prices," 30 September 2021. [Online]. Available: https://www.investopedia.com/ask/answers/155.asp#:~:text=Negative%20news%20will%20normally%20cause,many%20if%20not%20most%20stocks..

[21] J. Kuepper, A. Courage and S. Silberstein, "Day Trading: The Basics and How to Get Started," 5 January 2024. [Online]. Available: https://www.investopedia.com/articles/trading/05/011705.asp#toc-what-is-day-trading.

[22] A. Ganti, "Adjusted Closing Price: How It Works, Types, Pros & Cons," 28 December 2020. [Online]. Available: https://www.investopedia.com/terms/a/adjusted_closing_price.asp.

[23] W. Kenton, M. J. Boyle and J. Ma, "S&P 500 Index: What It's for and Why It's Important in Investing," 2023 September 22. [Online]. Available: https://www.investopedia.com/terms/s/sp500.asp.

[24] C. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," *Proceedings of the International AAAI Conference on Web and Social Media,* vol. 8, no. 1, pp. 216-225, 2014.

[25] P. Koráb, "FinVADER," GitHub, 7 December 2023. [Online]. Available: https://github.com/PetrKorab/FinVADER. [Accessed 31 January 2024].

[26] consose, "SentiBigNomics," GitHub, 18 December 2020. [Online]. Available: https://github.com/consose/SentiBigNomics. [Accessed 31 January 2024].

[27] E. Henry, "Are Investors Influenced By How Earnings Press Releases Are Written?," *The Journal of Business Communication (1973),* vol. 45, no. 4, pp. 363-407, 2008.

[28] N. Mehrotra, "The Aftermath And Impact Of Covid-19 On Stock Markets," 10 February 2023. [Online].

[29] A. Karanikola, G. Davrazos, C. M. Liapis and S. Kotsiantis, "Financial sentiment analysis: Classic methods vs. deep learning models," *Intelligent Decision Technologies,* pp. 893-915, 20 November 2023.

[30] H.-C. Chou, "Combination of Time Series Analysis and Sentiment Analysis for Stock Market Forecasting," USF Tampa Graduate Theses and Dissertations, South Florida, 2021.

[31] C. Jethva, S. Dudani, E. Malik, M. Sonje and G. Tanna, “Stock Analyzer and Bot using Machine Learning,” 2022 IEEE Region 10 Symposium (TENSYMP), Mumbai, India, 2022.

[32] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2018.

[33] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” in *Neural computation,* Cambridge, MIT Press Direct, 1997, pp. 1735-1780.

[34] Y. Li and Y. Pan, “A novel ensemble deep learning model for stock prediction based on stock prices and news,” *International Journal of Data Science and Analytics,* pp. 139-149, 2022.

[35] G. E. P. Box, G. M. Jenkins, G. C. Reinsel and G. M. Ljung, Time Series Analysis: Forecasting and Control (5th ed.), Hoboken, NJ: John Wiley and Sons Inc., 2015.

[36] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov., “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *arXiv preprint,* 2019.

[37] R. Jolhe, D. S. Shelke, P. M. Walunj, R. K. Tank, A. S. Bhandarkar, K. M. Shah and S. C. Prasad, “Stock Price Prediction Using Arima Forecasting and LSTM Based Forecasting, Competitive Analysis,” *International Journal for Research in Applied Science & Engineering Technology (IJRASET),* vol. 10, no. XI, 2022.

[38] J. Youness and M. Driss, “LSTM Deep Learning vs ARIMA Algorithms for Univariate Time Series Forecasting: A case study,” *2022 8th International Conference on Optimization and Applications (ICOA),* pp. 1-4, 2022.

[39] M. Rhanoui, S. Yousfi, M. Mikram and H. Merizak, “Forecasting financial budget time series: ARIMA random walk vs LSTM neural network,” *IAES International Journal of Artificial Intelligence (IJ-AI),* vol. 8, no. 4, pp. 317-327, 2019.

[40] J. Bagul, P. Warkhade, T. Gangwal and N. Mangaonkar, “ARIMA vs LSTM Algorithm – A Comparative Study Based on Stock Market Prediction,” *2022 5th International Conference on Advances in Science and Technology (ICAST),* pp. 49-53, 2022.

[41] G. Li, M. Xiao and Y. Guo, “Application of Deep Learning in Stock Market Valuation Index Forecasting,” *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS),* pp. 551-554, 2019.

[42] Hugging Face, “Using transformers at Hugging Face,” 15 December 2023. [Online]. Available: https://huggingface.co/docs/hub/transformers.

[43] L. Pham, “Financial News Headlines Data, Version 1,” 2020.

[44] L. Barbaglia, S. Consoli and S. Manzan, “Forecasting with Economic News,” Journal of Business and Economic Statistics, 2022.

[45] L. Barbaglia, S. Consoli and S. and Manzan, “Fine-grained, aspect-based semantic sentiment analysis within the economic and financial domains,” *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI),* pp. 52-61, 2020.

[46] S. Consoli, L. Barbaglia and S. 2. Manzan, “Explaining sentiment from Lexicon,” *CEUR Workshop Proceedings,* vol. 2918, pp. 87-95, 2021.

[47] S. Consoli, L. Barbaglia and S. Manzan, “Fine-grained, aspect-based sentiment analysis on economic and financial lexicon,” *Knowledge-Based Systems,* vol. 247, no. 108781, 2022.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, “Scikit-learn: Machine Learning in {P}ython,” *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[49] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, V. N. P. Andreas MuellerOlivier Grisel and A. Gramfort.

[50] J. Chen, “Announcement Effect,” 15 September 2020. [Online]. Available: https://www.investopedia.com/terms/a/announcment-effect.asp#:~:text=Stock%20prices%20can%20quickly%20move,to%20make%20short-term%20profits..

8. Appendices

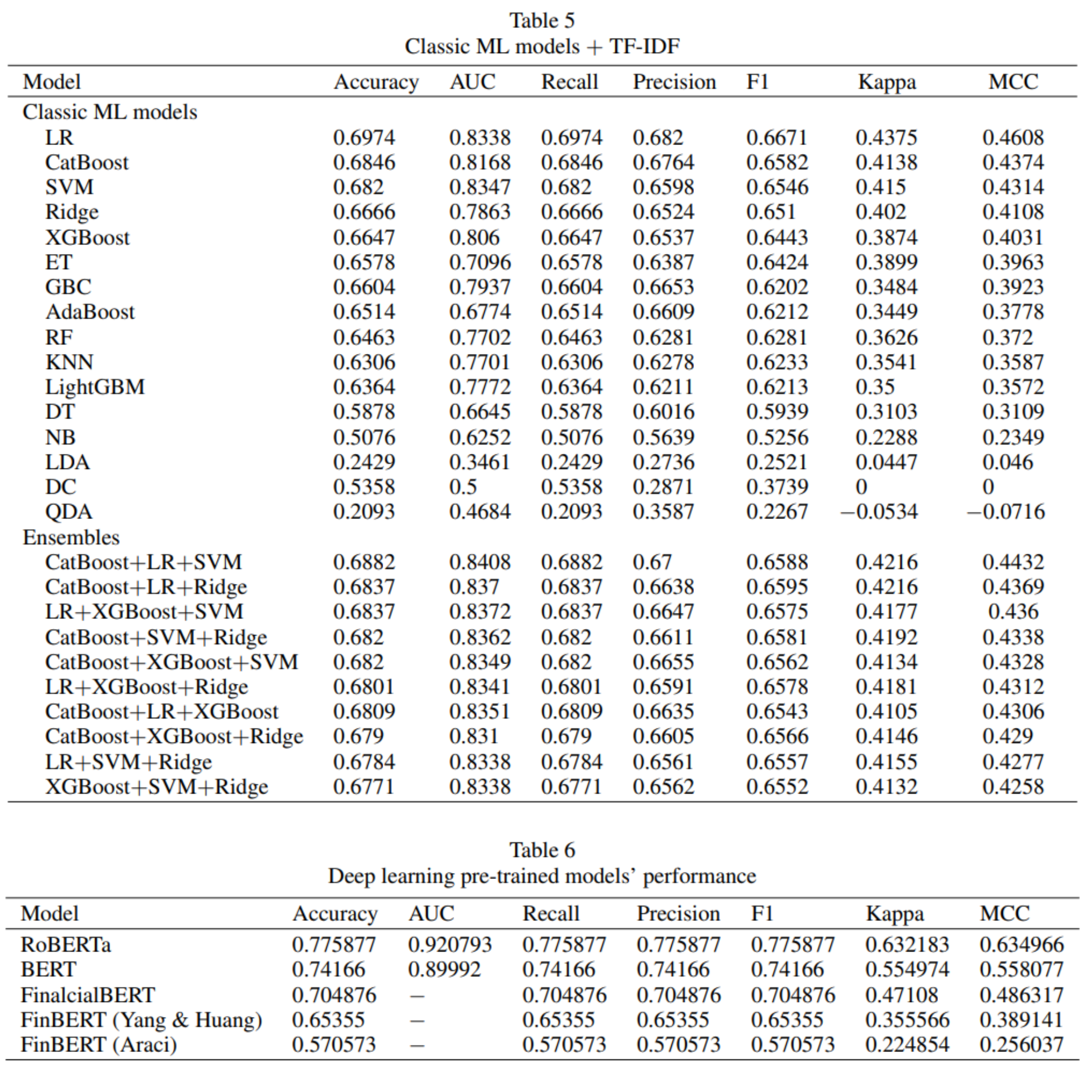


Table 5
Classic ML models + TF-IDF

Model	Accuracy	AUC	Recall	Precision	F1	Kappa	MCC
Classic ML models							
LR	0.6974	0.8338	0.6974	0.682	0.6671	0.4375	0.4608
CatBoost	0.6846	0.8168	0.6846	0.6764	0.6582	0.4138	0.4374
SVM	0.682	0.8347	0.682	0.6598	0.6546	0.415	0.4314
Ridge	0.6666	0.7863	0.6666	0.6524	0.651	0.402	0.4108
XGBoost	0.6647	0.806	0.6647	0.6537	0.6443	0.3874	0.4031
ET	0.6578	0.7096	0.6578	0.6387	0.6424	0.3899	0.3963
GBC	0.6604	0.7937	0.6604	0.6653	0.6202	0.3484	0.3923
AdaBoost	0.6514	0.6774	0.6514	0.6609	0.6212	0.3449	0.3778
RF	0.6463	0.7702	0.6463	0.6281	0.6281	0.3626	0.372
KNN	0.6306	0.7701	0.6306	0.6278	0.6233	0.3541	0.3587
LightGBM	0.6364	0.7772	0.6364	0.6211	0.6213	0.35	0.3572
DT	0.5878	0.6645	0.5878	0.6016	0.5939	0.3103	0.3109
NB	0.5076	0.6252	0.5076	0.5639	0.5256	0.2288	0.2349
LDA	0.2429	0.3461	0.2429	0.2736	0.2521	0.0447	0.046
DC	0.5358	0.5	0.5358	0.2871	0.3739	0	0
QDA	0.2093	0.4684	0.2093	0.3587	0.2267	−0.0534	−0.0716
Ensembles							
CatBoost+LR+SVM	0.6882	0.8408	0.6882	0.67	0.6588	0.4216	0.4432
CatBoost+LR+Ridge	0.6837	0.837	0.6837	0.6638	0.6595	0.4216	0.4369
LR+XGBoost+SVM	0.6837	0.8372	0.6837	0.6647	0.6575	0.4177	0.436
CatBoost+SVM+Ridge	0.682	0.8362	0.682	0.6611	0.6581	0.4192	0.4338
CatBoost+XGBoost+SVM	0.682	0.8349	0.682	0.6655	0.6562	0.4134	0.4328
LR+XGBoost+Ridge	0.6801	0.8341	0.6801	0.6591	0.6578	0.4181	0.4312
CatBoost+LR+XGBoost	0.6809	0.8351	0.6809	0.6635	0.6543	0.4105	0.4306
CatBoost+XGBoost+Ridge	0.679	0.831	0.679	0.6605	0.6566	0.4146	0.429
LR+SVM+Ridge	0.6784	0.8338	0.6784	0.6561	0.6557	0.4155	0.4277
XGBoost+SVM+Ridge	0.6771	0.8338	0.6771	0.6562	0.6552	0.4132	0.4258

Table 6
Deep learning pre-trained models' performance

Model	Accuracy	AUC	Recall	Precision	F1	Kappa	MCC
RoBERTa	0.775877	0.920793	0.775877	0.775877	0.775877	0.632183	0.634966
BERT	0.74166	0.89992	0.74166	0.74166	0.74166	0.554974	0.558077
FinalcialBERT	0.704876	–	0.704876	0.704876	0.704876	0.47108	0.486317
FinBERT (Yang & Huang)	0.65355	–	0.65355	0.65355	0.65355	0.355566	0.389141
FinBERT (Araci)	0.570573	–	0.570573	0.570573	0.570573	0.224854	0.256037

APPENDIX A: Excerpt from Financial sentiment analysis: Classic methods vs. deep learning models by Karanikola et al. (2023) [29]

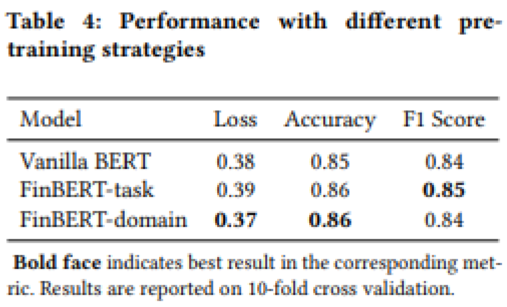


Table 4: Performance with different pre-training strategies

Model	Loss	Accuracy	F1 Score
Vanilla BERT	0.38	0.85	0.84
FinBERT-task	0.39	0.86	**0.85**
FinBERT-domain	**0.37**	**0.86**	0.84

Bold face indicates best result in the corresponding metric. Results are reported on 10-fold cross validation.

Appendix B: Excerpt from FinBERT: Financial Sentiment Analysis with Pre-trained Language Models by D. Araci (2019) [13]

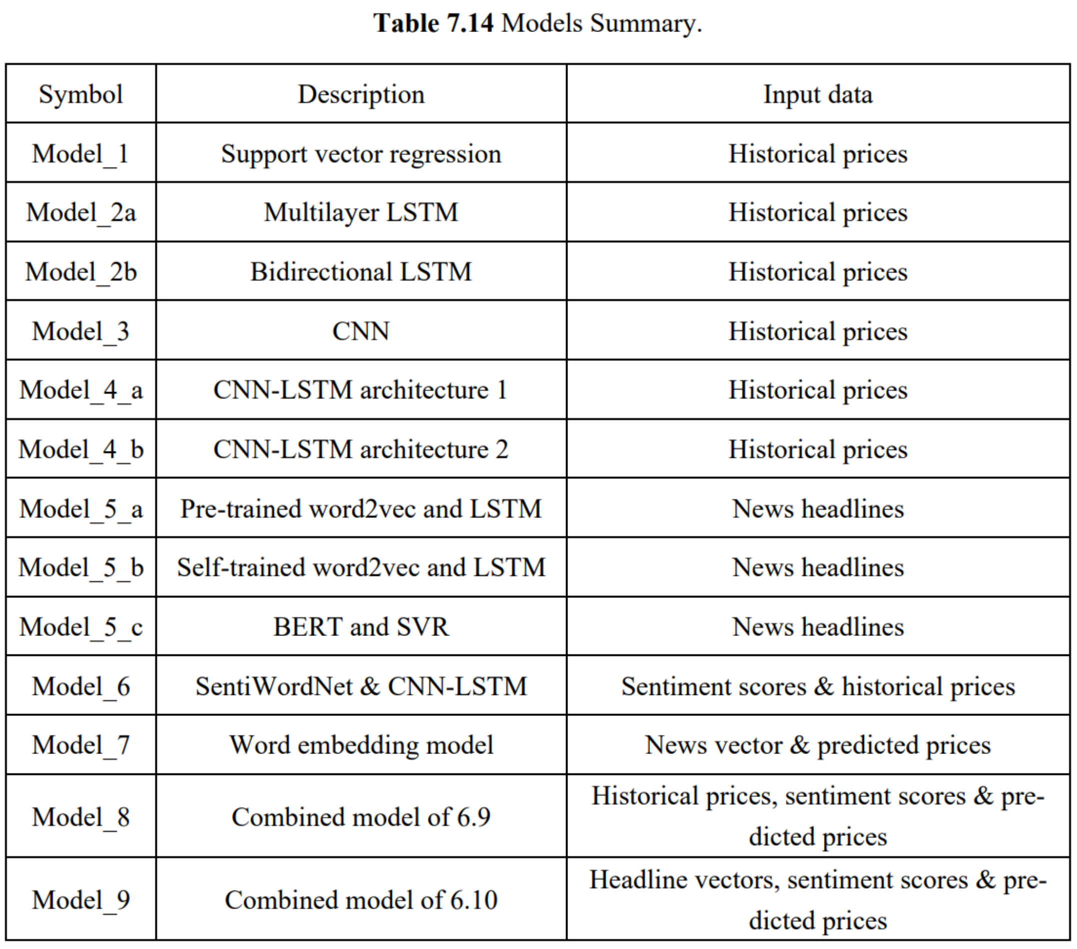


Table 7.14 Models Summary.

Symbol	Description	Input data
Model_1	Support vector regression	Historical prices
Model_2a	Multilayer LSTM	Historical prices
Model_2b	Bidirectional LSTM	Historical prices
Model_3	CNN	Historical prices
Model_4_a	CNN-LSTM architecture 1	Historical prices
Model_4_b	CNN-LSTM architecture 2	Historical prices
Model_5_a	Pre-trained word2vec and LSTM	News headlines
Model_5_b	Self-trained word2vec and LSTM	News headlines
Model_5_c	BERT and SVR	News headlines
Model_6	SentiWordNet & CNN-LSTM	Sentiment scores & historical prices
Model_7	Word embedding model	News vector & predicted prices
Model_8	Combined model of 6.9	Historical prices, sentiment scores & predicted prices
Model_9	Combined model of 6.10	Headline vectors, sentiment scores & predicted prices

APPENDIX C: Excerpt from Combination of Time Series Analysis and Sentiment Analysis for Stock Market Forecasting by Chou (2021) [30]

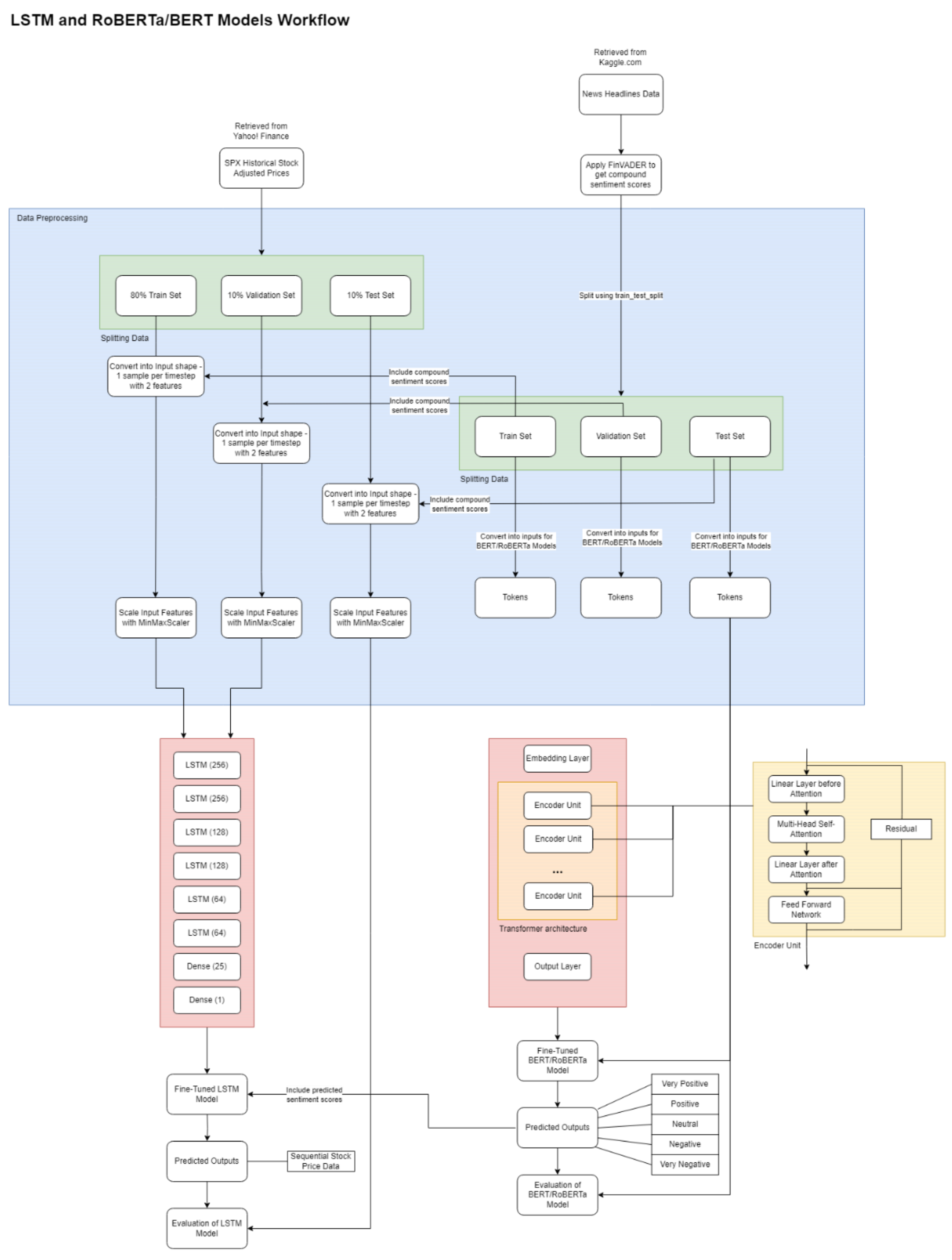
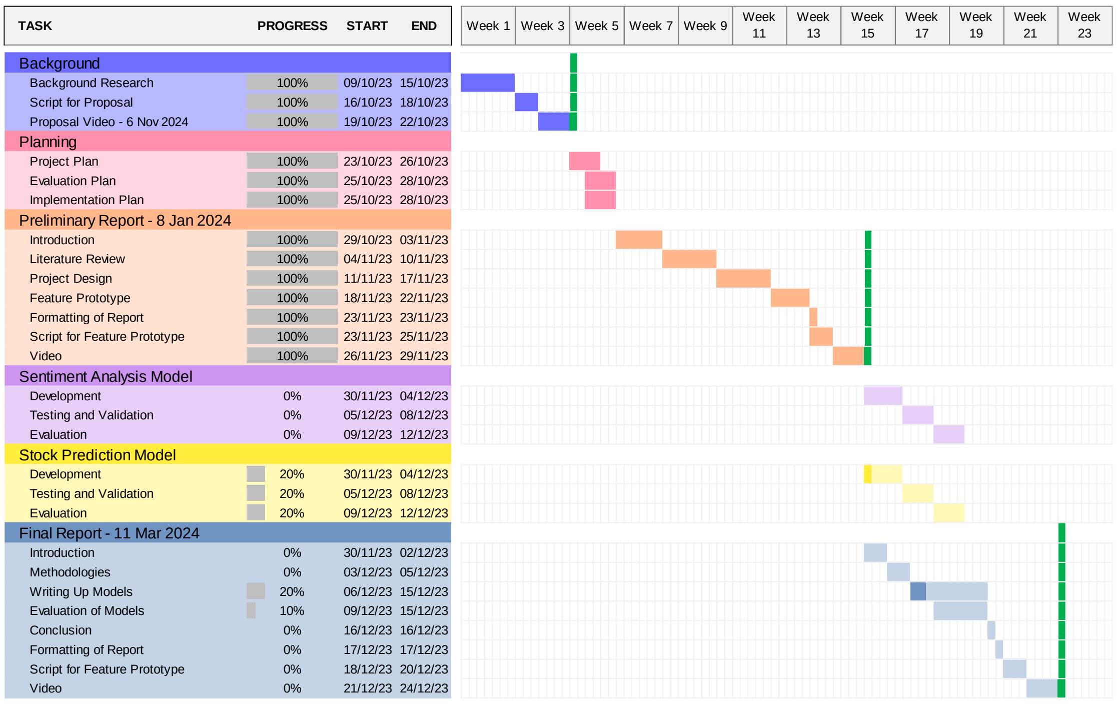

LSTM and RoBERTa/BERT Models Workflow
Retrieved from Kaggle.com
News Headlines Data
Apply FinVADER to get compound sentiment scores
Retrieved from Yahoo! Finance
SPX Historical Stock Adjusted Prices
Data Preprocessing
80% Train Set
10% Validation Set
10% Test Set
Splitting Data
Split using train_test_split
Convert into Input shape - 1 sample per timestep with 2 features
Include compound sentiment scores
Include compound sentiment scores
Train Set
Validation Set
Test Set
Splitting Data
Convert into Input shape - 1 sample per timestep with 2 features
Convert into Input shape - 1 sample per timestep with 2 features
Include compound sentiment scores
Convert into inputs for BERT/RoBERTa Models
Convert into inputs for BERT/RoBERTa Models
Convert into inputs for BERT/RoBERTa Models
Tokens
Tokens
Tokens
Scale Input Features with MinMaxScaler
Scale Input Features with MinMaxScaler
Scale Input Features with MinMaxScaler
LSTM (256)
LSTM (256)
LSTM (128)
LSTM (128)
LSTM (64)
LSTM (64)
Dense (25)
Dense (1)
Embedding Layer
Encoder Unit
Encoder Unit
...
Encoder Unit
Transformer architecture
Output Layer
Linear Layer before Attention
Multi-Head Self-Attention
Linear Layer after Attention
Feed Forward Network
Residual
Encoder Unit
Fine-Tuned LSTM Model
Include predicted sentiment scores
Predicted Outputs
Sequential Stock Price Data
Evaluation of LSTM Model
Fine-Tuned BERT/RoBERTa Model
Predicted Outputs
Very Positive
Positive
Neutral
Negative
Very Negative
Evaluation of BERT/RoBERTa Model

Figure 1: Flow of the project



TASK	PROGRESS	START	END
Background			
Background Research	100%	09/10/23	15/10/23
Script for Proposal	100%	16/10/23	18/10/23
Proposal Video - 6 Nov 2024	100%	19/10/23	22/10/23
Planning			
Project Plan	100%	23/10/23	26/10/23
Evaluation Plan	100%	25/10/23	28/10/23
Implementation Plan	100%	25/10/23	28/10/23
Preliminary Report - 8 Jan 2024			
Introduction	100%	29/10/23	03/11/23
Literature Review	100%	04/11/23	10/11/23
Project Design	100%	11/11/23	17/11/23
Feature Prototype	100%	18/11/23	22/11/23
Formatting of Report	100%	23/11/23	23/11/23
Script for Feature Prototype	100%	23/11/23	25/11/23
Video	100%	26/11/23	29/11/23
Sentiment Analysis Model			
Development	0%	30/11/23	04/12/23
Testing and Validation	0%	05/12/23	08/12/23
Evaluation	0%	09/12/23	12/12/23
Stock Prediction Model			
Development	20%	30/11/23	04/12/23
Testing and Validation	20%	05/12/23	08/12/23
Evaluation	20%	09/12/23	12/12/23
Final Report - 11 Mar 2024			
Introduction	0%	30/11/23	02/12/23
Methodologies	0%	03/12/23	05/12/23
Writing Up Models	20%	06/12/23	15/12/23
Evaluation of Models	10%	09/12/23	15/12/23
Conclusion	0%	16/12/23	16/12/23
Formatting of Report	0%	17/12/23	17/12/23
Script for Feature Prototype	0%	18/12/23	20/12/23
Video	0%	21/12/23	24/12/23

Figure 2: Timeline of project

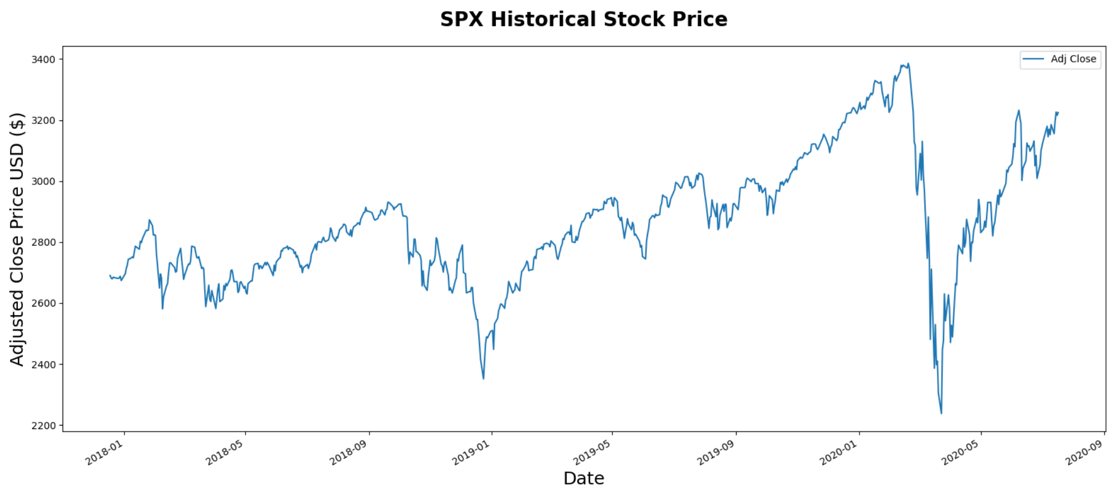
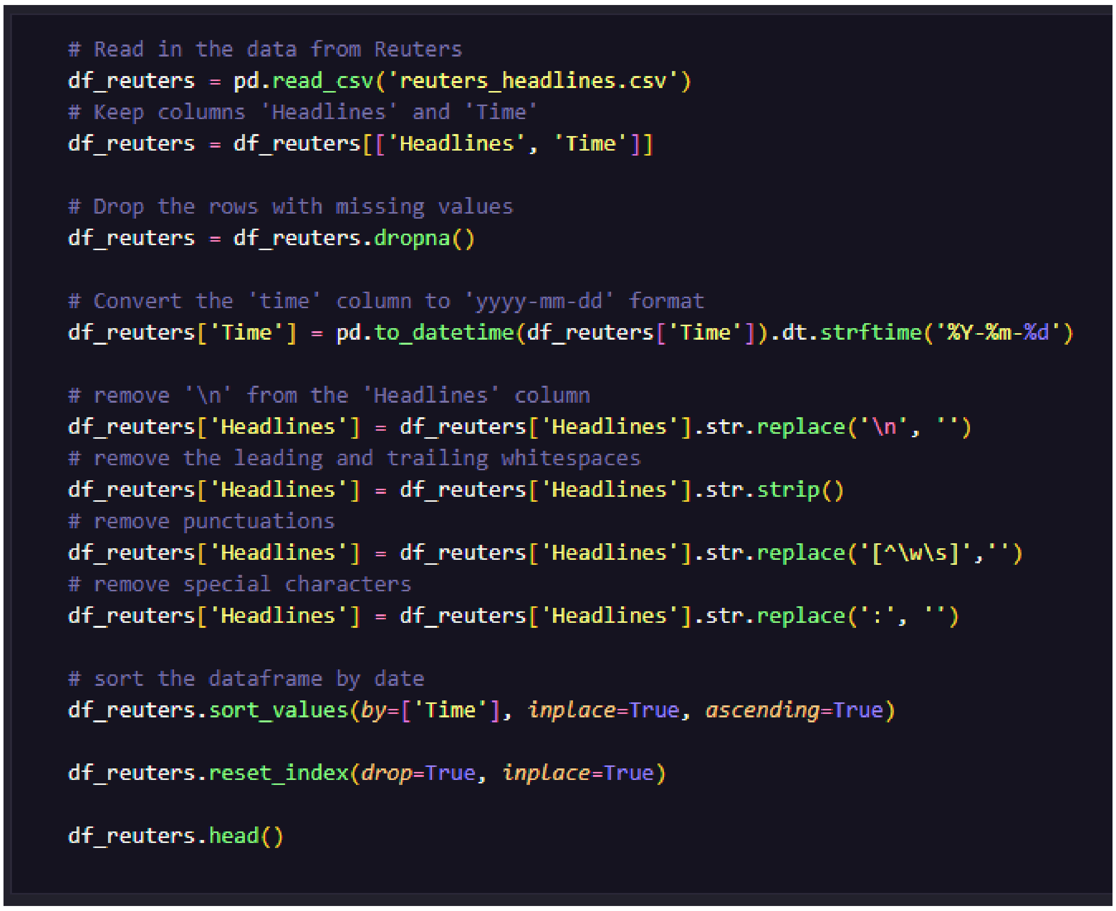
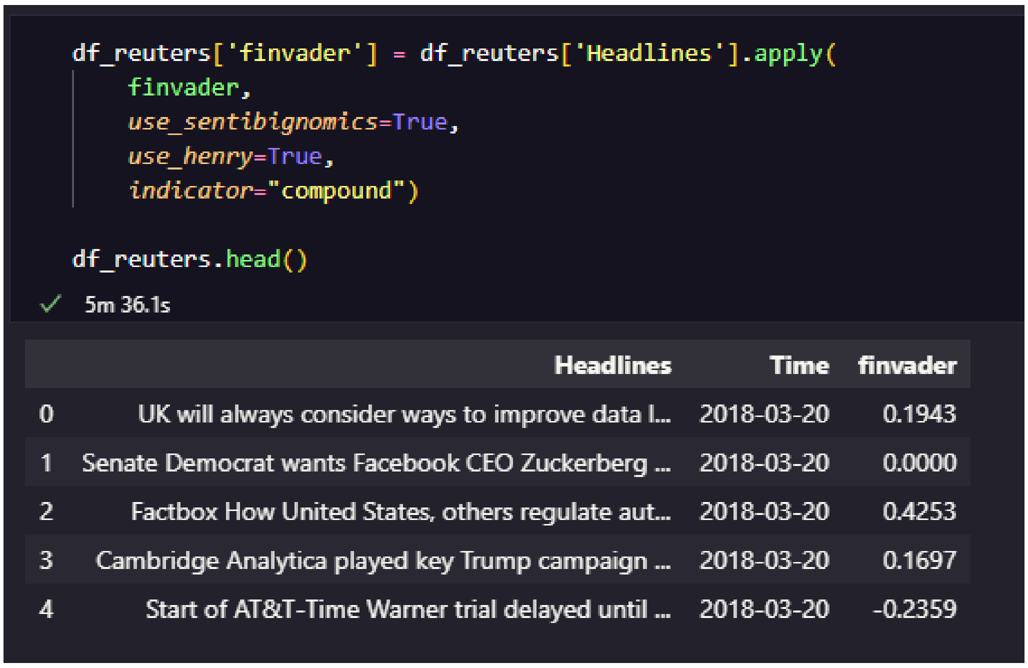


Figure 3: SPX Historical Stock Price



```python
# Read in the data from Reuters
df_reuters = pd.read_csv('reuters_headlines.csv')
# Keep columns 'Headlines' and 'Time'
df_reuters = df_reuters[['Headlines', 'Time']]

# Drop the rows with missing values
df_reuters = df_reuters.dropna()

# Convert the 'time' column to 'yyyy-mm-dd' format
df_reuters['Time'] = pd.to_datetime(df_reuters['Time']).dt.strftime('%Y-%m-%d')

# remove '\n' from the 'Headlines' column
df_reuters['Headlines'] = df_reuters['Headlines'].str.replace('\n', '')
# remove the leading and trailing whitespaces
df_reuters['Headlines'] = df_reuters['Headlines'].str.strip()
# remove punctuations
df_reuters['Headlines'] = df_reuters['Headlines'].str.replace('[^\w\s]','')
# remove special characters
df_reuters['Headlines'] = df_reuters['Headlines'].str.replace(':', '')

# sort the dataframe by date
df_reuters.sort_values(by=['Time'], inplace=True, ascending=True)

df_reuters.reset_index(drop=True, inplace=True)

df_reuters.head()
```

Figure 4: Pre-processing of News Headlines

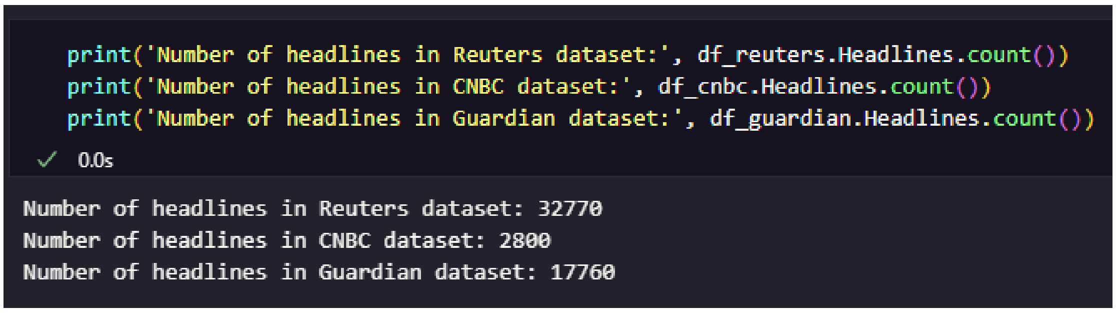


```python
df_reuters['finvader'] = df_reuters['Headlines'].apply(
    finvader,
    use_sentibignomics=True,
    use_henry=True,
    indicator="compound")

df_reuters.head()
```

✓ 5m 36.1s

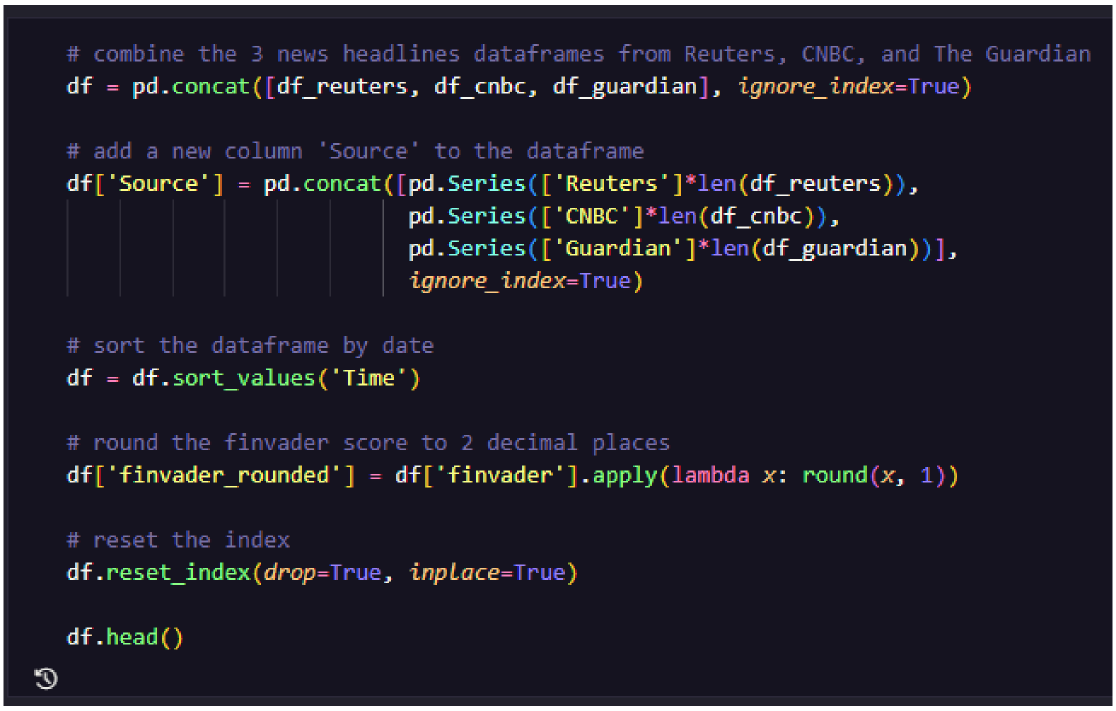
	Headlines	Time	finvader
0	UK will always consider ways to improve data l...	2018-03-20	0.1943
1	Senate Democrat wants Facebook CEO Zuckerberg ...	2018-03-20	0.0000
2	Factbox How United States, others regulate aut...	2018-03-20	0.4253
3	Cambridge Analytica played key Trump campaign ...	2018-03-20	0.1697
4	Start of AT&T-Time Warner trial delayed until ...	2018-03-20	-0.2359

Figure 5: Application of FinVADER on News Headlines



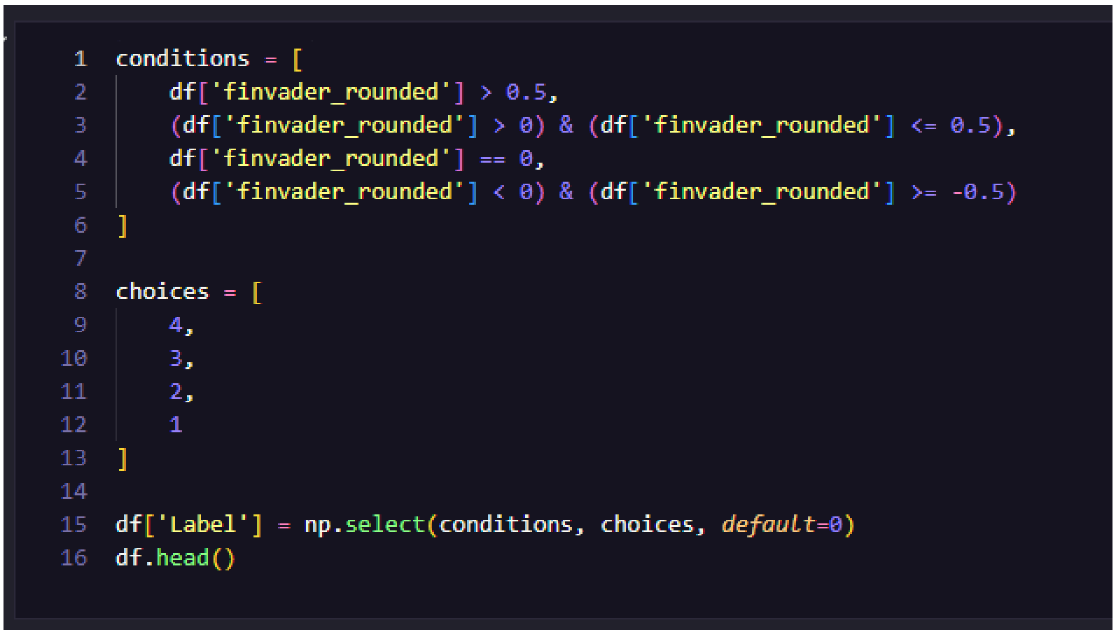
```
print('Number of headlines in Reuters dataset:', df_reuters.Headlines.count())
print('Number of headlines in CNBC dataset:', df_cnbc.Headlines.count())
print('Number of headlines in Guardian dataset:', df_guardian.Headlines.count())
✓ 0.0s

Number of headlines in Reuters dataset: 32770
Number of headlines in CNBC dataset: 2800
Number of headlines in Guardian dataset: 17760
```

Figure 6: Number of news headlines in each news source



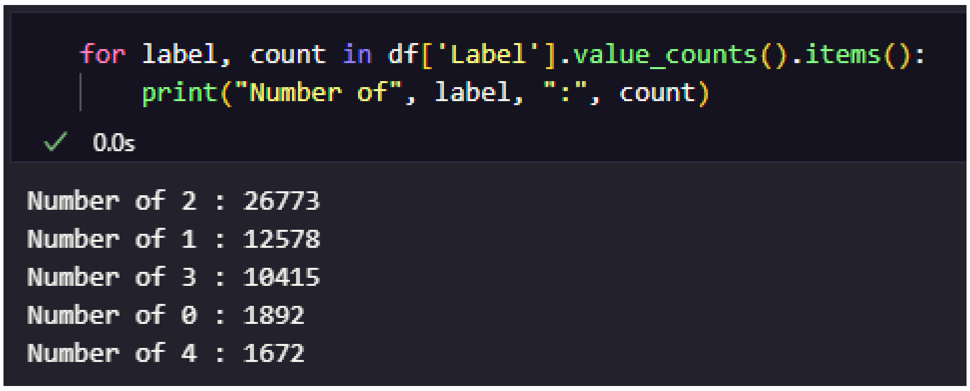
```
# combine the 3 news headlines dataframes from Reuters, CNBC, and The Guardian
df = pd.concat([df_reuters, df_cnbc, df_guardian], ignore_index=True)

# add a new column 'Source' to the dataframe
df['Source'] = pd.concat([pd.Series(['Reuters']*len(df_reuters)),
                          pd.Series(['CNBC']*len(df_cnbc)),
                          pd.Series(['Guardian']*len(df_guardian))],
                          ignore_index=True)

# sort the dataframe by date
df = df.sort_values('Time')

# round the finvader score to 2 decimal places
df['finvader_rounded'] = df['finvader'].apply(lambda x: round(x, 1))

# reset the index
df.reset_index(drop=True, inplace=True)

df.head()
```

Figure 7: Combining of news headlines from each news source and rounding of FinVADER



```python
conditions = [
    df['finvader_rounded'] > 0.5,
    (df['finvader_rounded'] > 0) & (df['finvader_rounded'] <= 0.5),
    df['finvader_rounded'] == 0,
    (df['finvader_rounded'] < 0) & (df['finvader_rounded'] >= -0.5)
]

choices = [
    4,
    3,
    2,
    1
]

df['Label'] = np.select(conditions, choices, default=0)
df.head()
```

Figure 8: Rounding of FinVADER compound scores

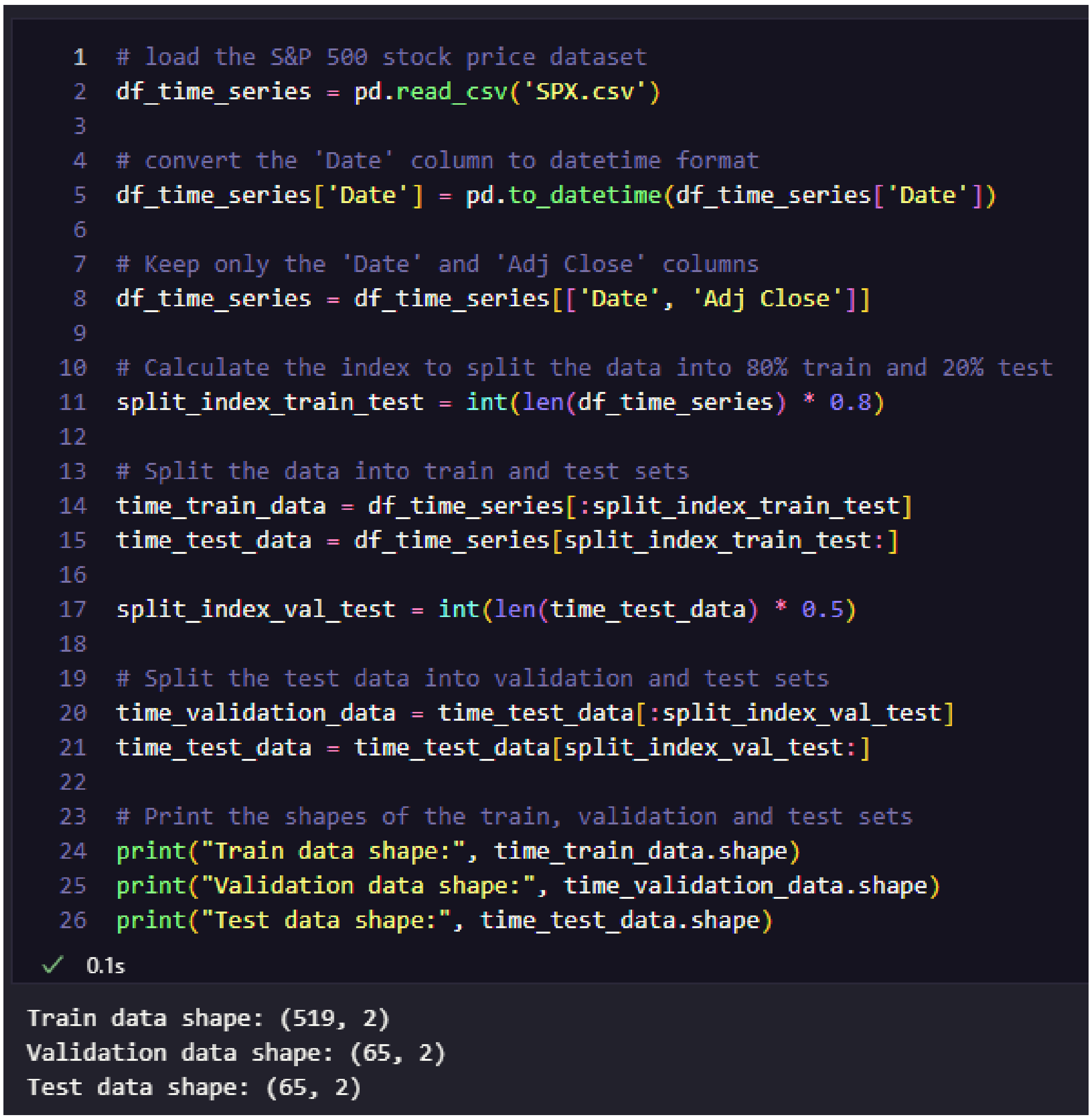


```python
for label, count in df['Label'].value_counts().items():
    print("Number of", label, ":", count)
```

✓ 0.0s

```
Number of 2 : 26773
Number of 1 : 12578
Number of 3 : 10415
Number of 0 : 1892
Number of 4 : 1672
```

Figure 9: Number of news headlines in each class

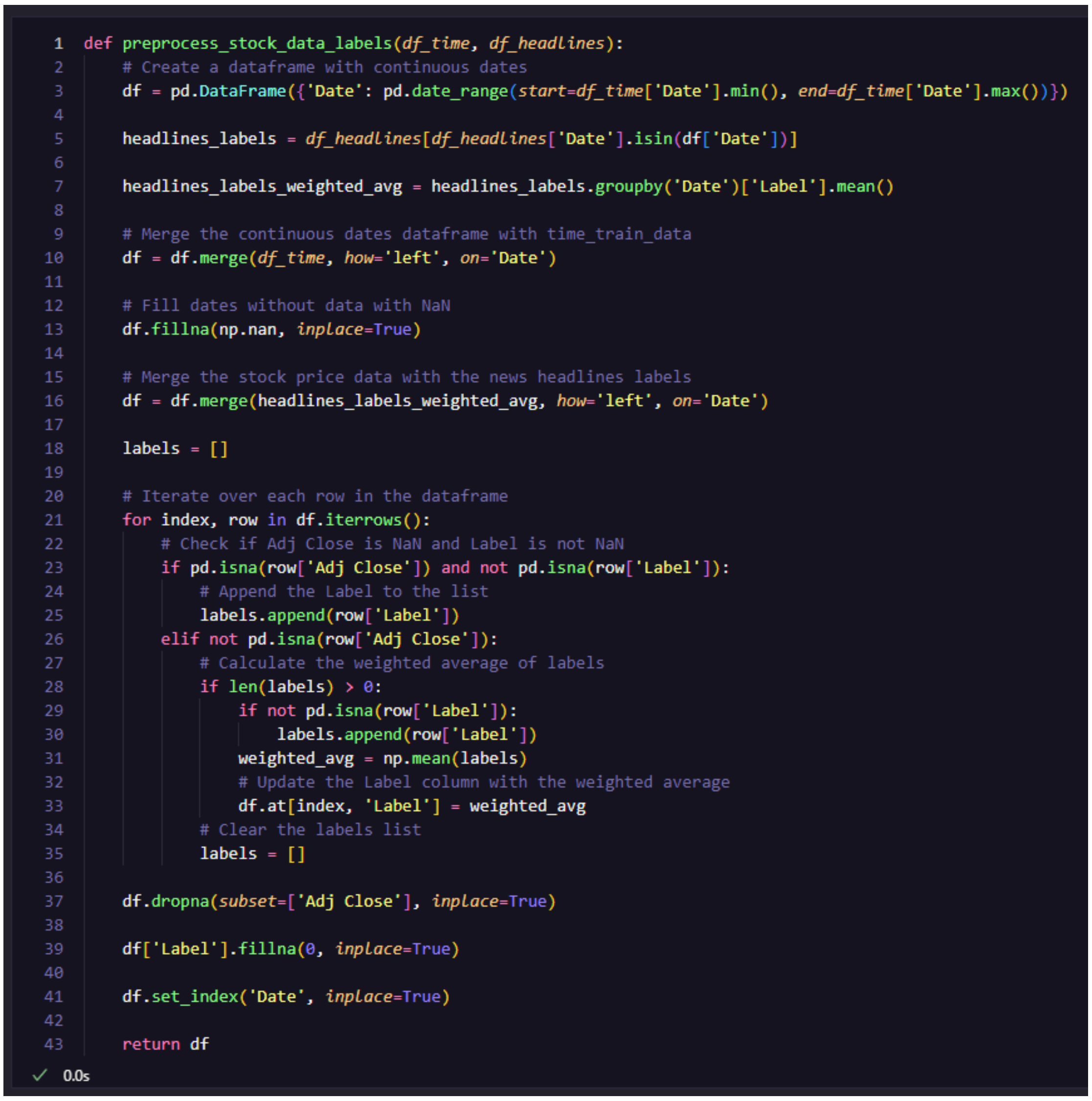


```python
# load the S&P 500 stock price dataset
df_time_series = pd.read_csv('SPX.csv')

# convert the 'Date' column to datetime format
df_time_series['Date'] = pd.to_datetime(df_time_series['Date'])

# Keep only the 'Date' and 'Adj Close' columns
df_time_series = df_time_series[['Date', 'Adj Close']]

# Calculate the index to split the data into 80% train and 20% test
split_index_train_test = int(len(df_time_series) * 0.8)

# Split the data into train and test sets
time_train_data = df_time_series[:split_index_train_test]
time_test_data = df_time_series[split_index_train_test:]

split_index_val_test = int(len(time_test_data) * 0.5)

# Split the test data into validation and test sets
time_validation_data = time_test_data[:split_index_val_test]
time_test_data = time_test_data[split_index_val_test:]

# Print the shapes of the train, validation and test sets
print("Train data shape:", time_train_data.shape)
print("Validation data shape:", time_validation_data.shape)
print("Test data shape:", time_test_data.shape)
```

✓ 0.1s

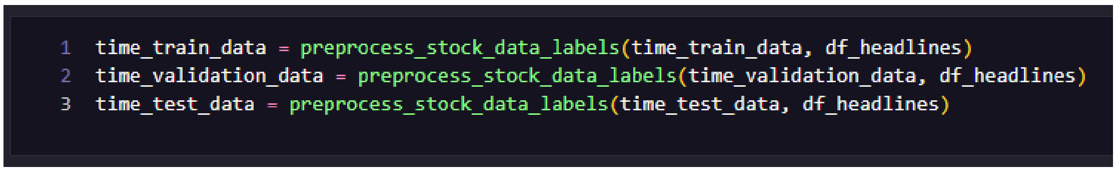
```
Train data shape: (519, 2)
Validation data shape: (65, 2)
Test data shape: (65, 2)
```

Figure 10: Splitting the stock price data into training, validation, and test sets



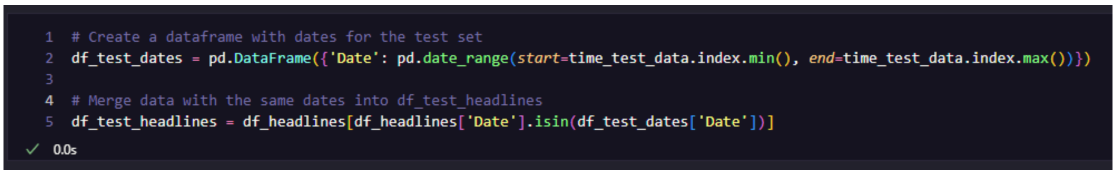
```python
def preprocess_stock_data_labels(df_time, df_headlines):
    # Create a dataframe with continuous dates
    df = pd.DataFrame({'Date': pd.date_range(start=df_time['Date'].min(), end=df_time['Date'].max())})

    headlines_labels = df_headlines[df_headlines['Date'].isin(df['Date'])]

    headlines_labels_weighted_avg = headlines_labels.groupby('Date')['Label'].mean()

    # Merge the continuous dates dataframe with time_train_data
    df = df.merge(df_time, how='left', on='Date')

    # Fill dates without data with NaN
    df.fillna(np.nan, inplace=True)

    # Merge the stock price data with the news headlines labels
    df = df.merge(headlines_labels_weighted_avg, how='left', on='Date')

    labels = []

    # Iterate over each row in the dataframe
    for index, row in df.iterrows():
        # Check if Adj Close is NaN and Label is not NaN
        if pd.isna(row['Adj Close']) and not pd.isna(row['Label']):
            # Append the Label to the list
            labels.append(row['Label'])
        elif not pd.isna(row['Adj Close']):
            # Calculate the weighted average of labels
            if len(labels) > 0:
                if not pd.isna(row['Label']):
                    labels.append(row['Label'])
                weighted_avg = np.mean(labels)
                # Update the Label column with the weighted average
                df.at[index, 'Label'] = weighted_avg
            # Clear the labels list
            labels = []

    df.dropna(subset=['Adj Close'], inplace=True)

    df['Label'].fillna(0, inplace=True)

    df.set_index('Date', inplace=True)

    return df
```
✓ 0.0s

Figure 11: Function for combining stock price data and news headline sentiment labels



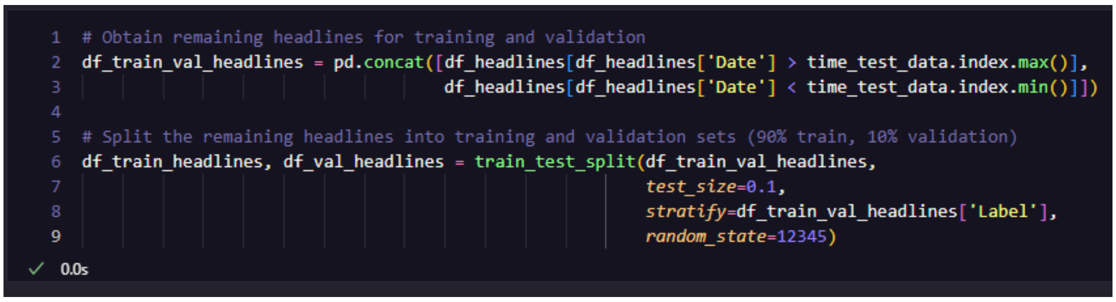
```python
time_train_data = preprocess_stock_data_labels(time_train_data, df_headlines)
time_validation_data = preprocess_stock_data_labels(time_validation_data, df_headlines)
time_test_data = preprocess_stock_data_labels(time_test_data, df_headlines)
```

Figure 12: Combining stock price data and news headline sentiment labels using function preprocess_stock_data_labels



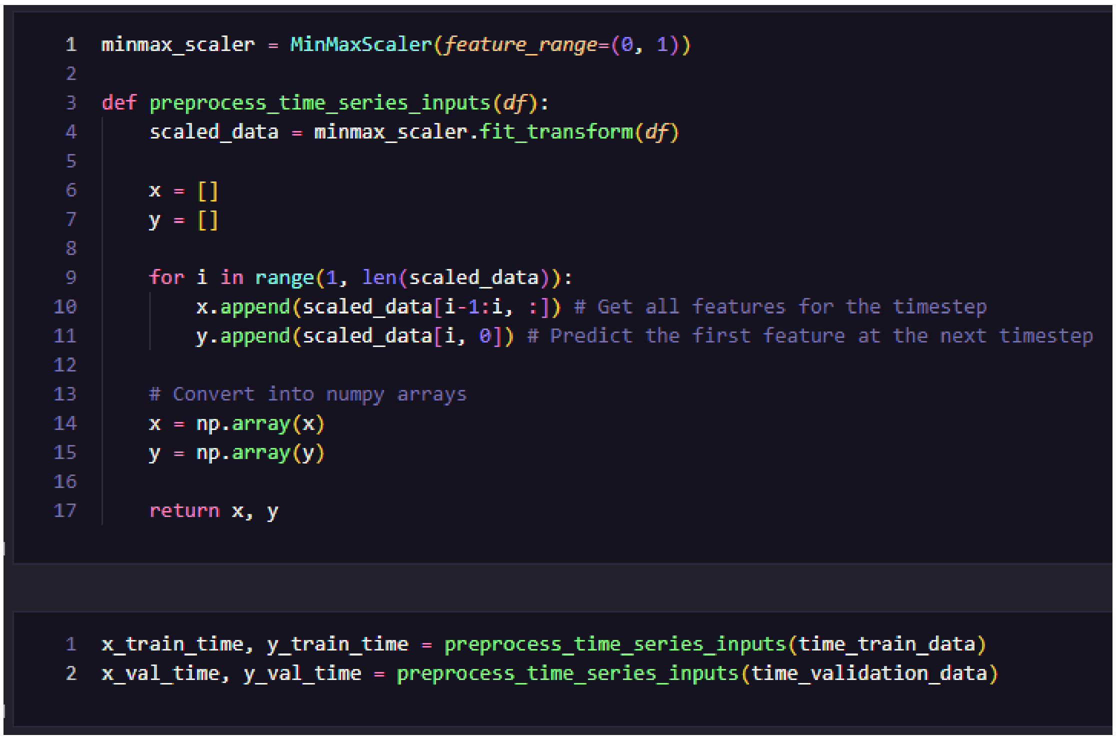
```python
# Create a dataframe with dates for the test set
df_test_dates = pd.DataFrame({'Date': pd.date_range(start=time_test_data.index.min(), end=time_test_data.index.max())})

# Merge data with the same dates into df_test_headlines
df_test_headlines = df_headlines[df_headlines['Date'].isin(df_test_dates['Date'])]
```
✓ 0.0s

Figure 13: Setting aside news headlines data with the same dates as stock price data test set for news headlines test set



```python
# Obtain remaining headlines for training and validation
df_train_val_headlines = pd.concat([df_headlines[df_headlines['Date'] > time_test_data.index.max()],
                                    df_headlines[df_headlines['Date'] < time_test_data.index.min()]])

# Split the remaining headlines into training and validation sets (90% train, 10% validation)
df_train_headlines, df_val_headlines = train_test_split(df_train_val_headlines,
                                                        test_size=0.1,
                                                        stratify=df_train_val_headlines['Label'],
                                                        random_state=12345)
```
✓ 0.0s

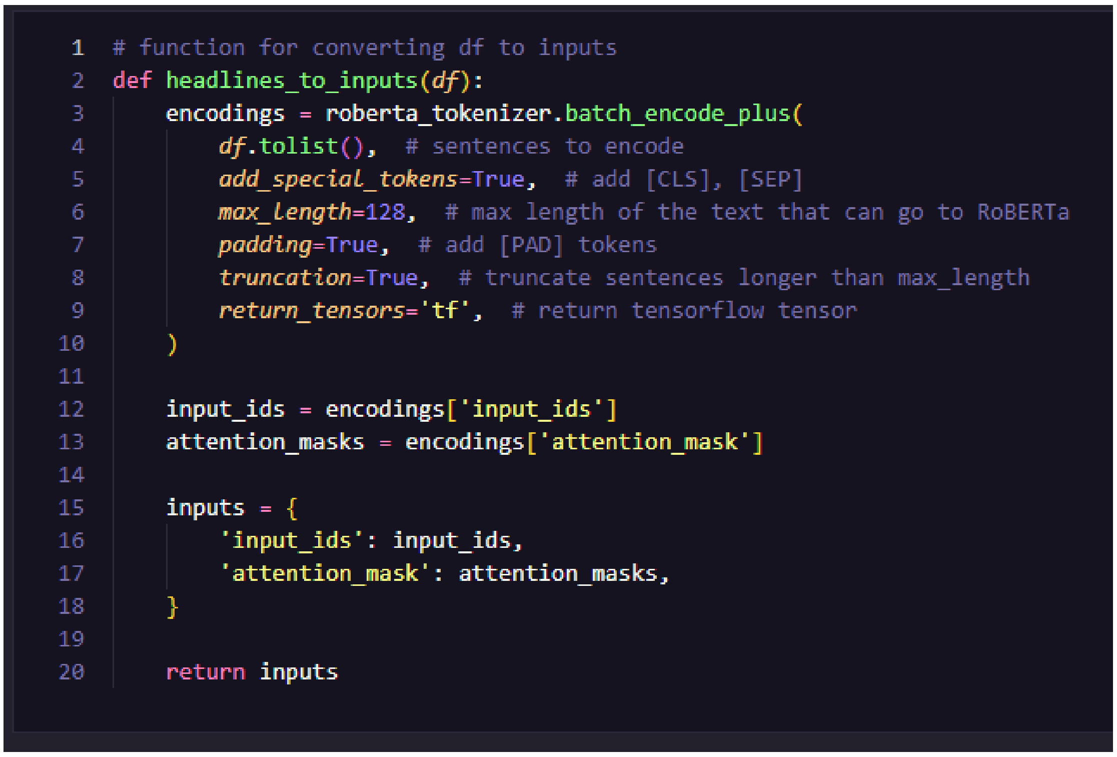
Figure 14: Splitting the remaining news headlines into training and validation sets



```python
minmax_scaler = MinMaxScaler(feature_range=(0, 1))

def preprocess_time_series_inputs(df):
    scaled_data = minmax_scaler.fit_transform(df)

    x = []
    y = []

    for i in range(1, len(scaled_data)):
        x.append(scaled_data[i-1:i, :]) # Get all features for the timestep
        y.append(scaled_data[i, 0]) # Predict the first feature at the next timestep

    # Convert into numpy arrays
    x = np.array(x)
    y = np.array(y)

    return x, y
```

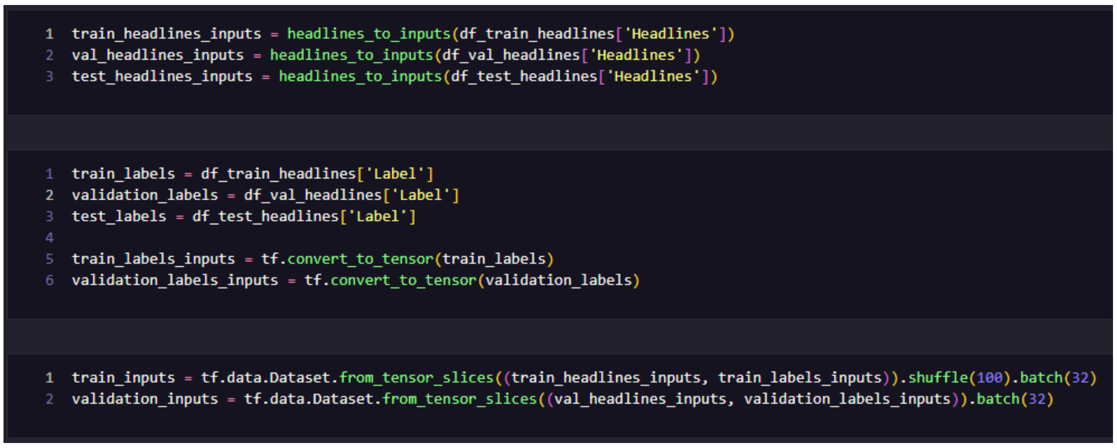
```python
x_train_time, y_train_time = preprocess_time_series_inputs(time_train_data)
x_val_time, y_val_time = preprocess_time_series_inputs(time_validation_data)
```

Figure 15: Processing stock price training and validation sets into LSTM inputs with MinMaxScaler



```python
# function for converting df to inputs
def headlines_to_inputs(df):
    encodings = roberta_tokenizer.batch_encode_plus(
        df.tolist(),  # sentences to encode
        add_special_tokens=True,  # add [CLS], [SEP]
        max_length=128,  # max length of the text that can go to RoBERTa
        padding=True,  # add [PAD] tokens
        truncation=True,  # truncate sentences longer than max_length
        return_tensors='tf',  # return tensorflow tensor
    )

    input_ids = encodings['input_ids']
    attention_masks = encodings['attention_mask']

    inputs = {
        'input_ids': input_ids,
        'attention_mask': attention_masks,
    }

    return inputs
```

Figure 16: Function for converting Pandas DataFrame of news headlines into RoBERTa inputs

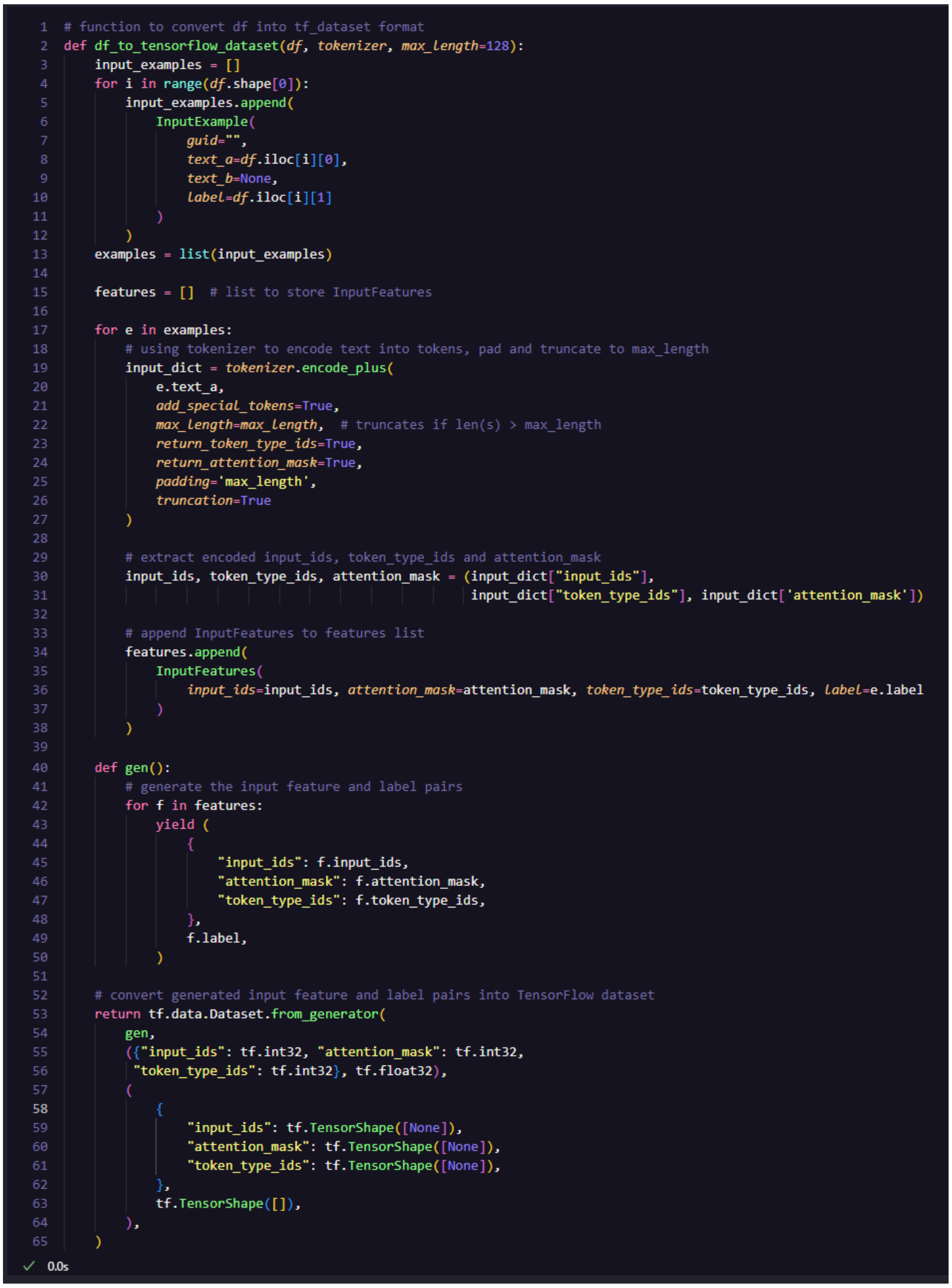


```python
train_headlines_inputs = headlines_to_inputs(df_train_headlines['Headlines'])
val_headlines_inputs = headlines_to_inputs(df_val_headlines['Headlines'])
test_headlines_inputs = headlines_to_inputs(df_test_headlines['Headlines'])
```

```python
train_labels = df_train_headlines['Label']
validation_labels = df_val_headlines['Label']
test_labels = df_test_headlines['Label']

train_labels_inputs = tf.convert_to_tensor(train_labels)
validation_labels_inputs = tf.convert_to_tensor(validation_labels)
```

```python
train_inputs = tf.data.Dataset.from_tensor_slices((train_headlines_inputs, train_labels_inputs)).shuffle(100).batch(32)
validation_inputs = tf.data.Dataset.from_tensor_slices((val_headlines_inputs, validation_labels_inputs)).batch(32)
```

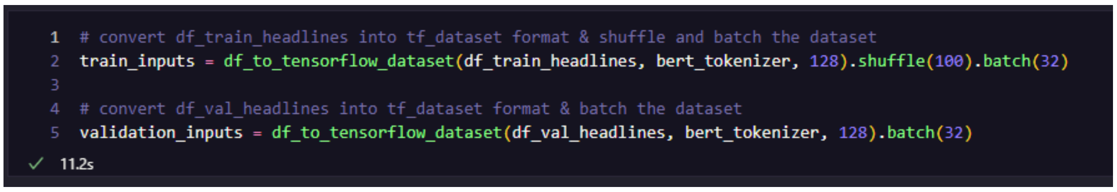
Figure 17: Converting news headlines data and labels into inputs for RoBERTa model



```python
# function to convert df into tf_dataset format
def df_to_tensorflow_dataset(df, tokenizer, max_length=128):
    input_examples = []
    for i in range(df.shape[0]):
        input_examples.append(
            InputExample(
                guid="",
                text_a=df.iloc[i][0],
                text_b=None,
                label=df.iloc[i][1]
            )
        )
    examples = list(input_examples)

    features = []  # list to store InputFeatures

    for e in examples:
        # using tokenizer to encode text into tokens, pad and truncate to max_length
        input_dict = tokenizer.encode_plus(
            e.text_a,
            add_special_tokens=True,
            max_length=max_length,  # truncates if len(s) > max_length
            return_token_type_ids=True,
            return_attention_mask=True,
            padding='max_length',
            truncation=True
        )

        # extract encoded input_ids, token_type_ids and attention_mask
        input_ids, token_type_ids, attention_mask = (input_dict["input_ids"],
                                                     input_dict["token_type_ids"], input_dict['attention_mask'])

        # append InputFeatures to features list
        features.append(
            InputFeatures(
                input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids, label=e.label
            )
        )

    def gen():
        # generate the input feature and label pairs
        for f in features:
            yield (
                {
                    "input_ids": f.input_ids,
                    "attention_mask": f.attention_mask,
                    "token_type_ids": f.token_type_ids,
                },
                f.label,
            )

    # convert generated input feature and label pairs into TensorFlow dataset
    return tf.data.Dataset.from_generator(
        gen,
        ({"input_ids": tf.int32, "attention_mask": tf.int32,
         "token_type_ids": tf.int32}, tf.float32),
        (
            {
                "input_ids": tf.TensorShape([None]),
                "attention_mask": tf.TensorShape([None]),
                "token_type_ids": tf.TensorShape([None]),
            },
            tf.TensorShape([]),
        ),
    )
```

✓ 0.0s

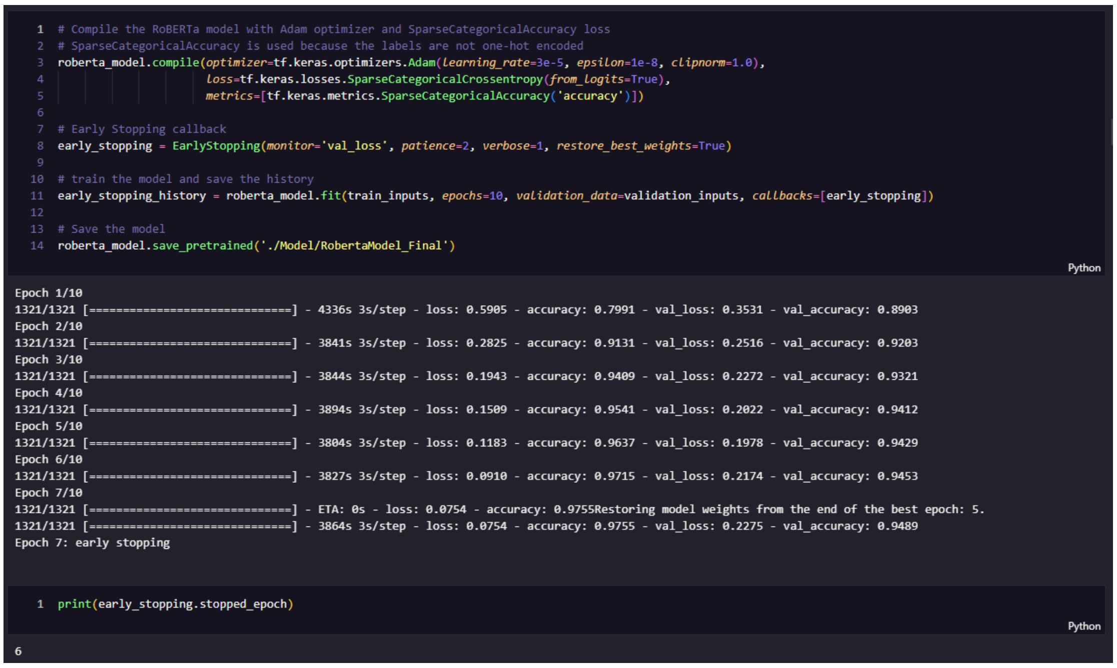
Figure 18: Function to convert Pandas DataFrame into inputs for BERT model



```python
# convert df_train_headlines into tf_dataset format & shuffle and batch the dataset
train_inputs = df_to_tensorflow_dataset(df_train_headlines, bert_tokenizer, 128).shuffle(100).batch(32)

# convert df_val_headlines into tf_dataset format & batch the dataset
validation_inputs = df_to_tensorflow_dataset(df_val_headlines, bert_tokenizer, 128).batch(32)
```

✓ 11.2s

Figure 19: Converting news headlines data and labels into inputs for BERT model



```python
# Compile the RoBERTa model with Adam optimizer and SparseCategoricalAccuracy loss
# SparseCategoricalAccuracy is used because the labels are not one-hot encoded
roberta_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-8, clipnorm=1.0),
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=[tf.keras.metrics.SparseCategoricalAccuracy('accuracy')])

# Early Stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=2, verbose=1, restore_best_weights=True)

# train the model and save the history
early_stopping_history = roberta_model.fit(train_inputs, epochs=10, validation_data=validation_inputs, callbacks=[early_stopping])

# Save the model
roberta_model.save_pretrained('./Model/RobertaModel_Final')
```

Python

```
Epoch 1/10
1321/1321 [==============================] - 4336s 3s/step - loss: 0.5905 - accuracy: 0.7991 - val_loss: 0.3531 - val_accuracy: 0.8903
Epoch 2/10
1321/1321 [==============================] - 3841s 3s/step - loss: 0.2825 - accuracy: 0.9131 - val_loss: 0.2516 - val_accuracy: 0.9203
Epoch 3/10
1321/1321 [==============================] - 3844s 3s/step - loss: 0.1943 - accuracy: 0.9409 - val_loss: 0.2272 - val_accuracy: 0.9321
Epoch 4/10
1321/1321 [==============================] - 3894s 3s/step - loss: 0.1509 - accuracy: 0.9541 - val_loss: 0.2022 - val_accuracy: 0.9412
Epoch 5/10
1321/1321 [==============================] - 3804s 3s/step - loss: 0.1183 - accuracy: 0.9637 - val_loss: 0.1978 - val_accuracy: 0.9429
Epoch 6/10
1321/1321 [==============================] - 3827s 3s/step - loss: 0.0910 - accuracy: 0.9715 - val_loss: 0.2174 - val_accuracy: 0.9453
Epoch 7/10
1321/1321 [==============================] - ETA: 0s - loss: 0.0754 - accuracy: 0.9755Restoring model weights from the end of the best epoch: 5.
1321/1321 [==============================] - 3864s 3s/step - loss: 0.0754 - accuracy: 0.9755 - val_loss: 0.2275 - val_accuracy: 0.9489
Epoch 7: early stopping
```

```python
print(early_stopping.stopped_epoch)
```

Python

```
6
```

Figure 20: Building and training of RoBERTa model

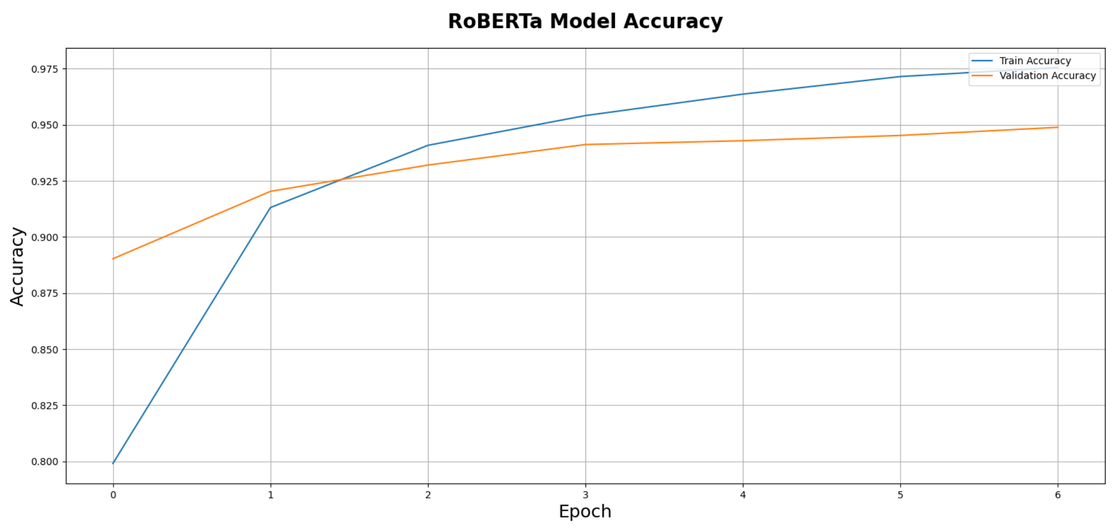


Figure 21: RoBERTa Model Accuracy during training

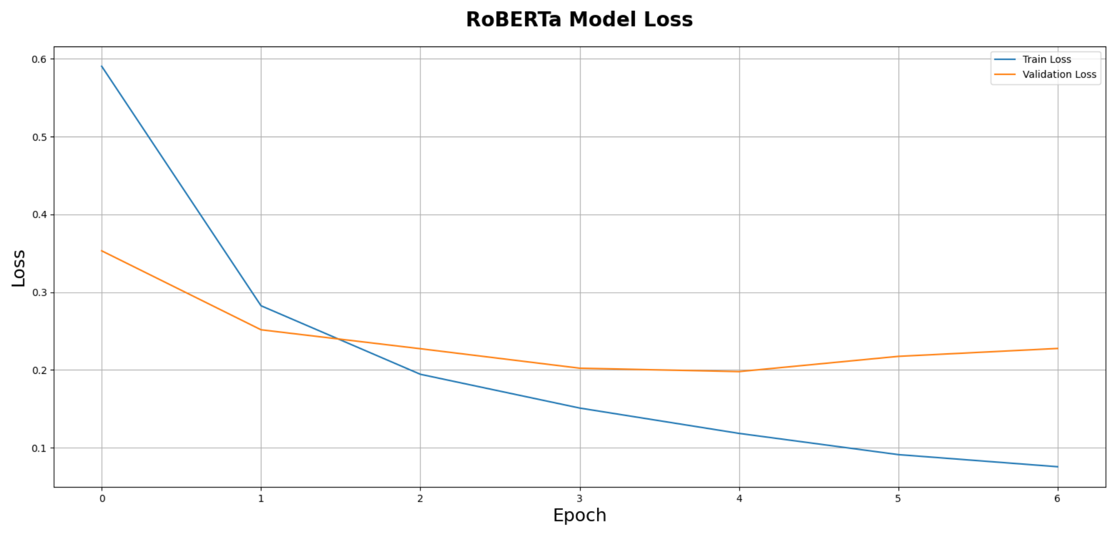
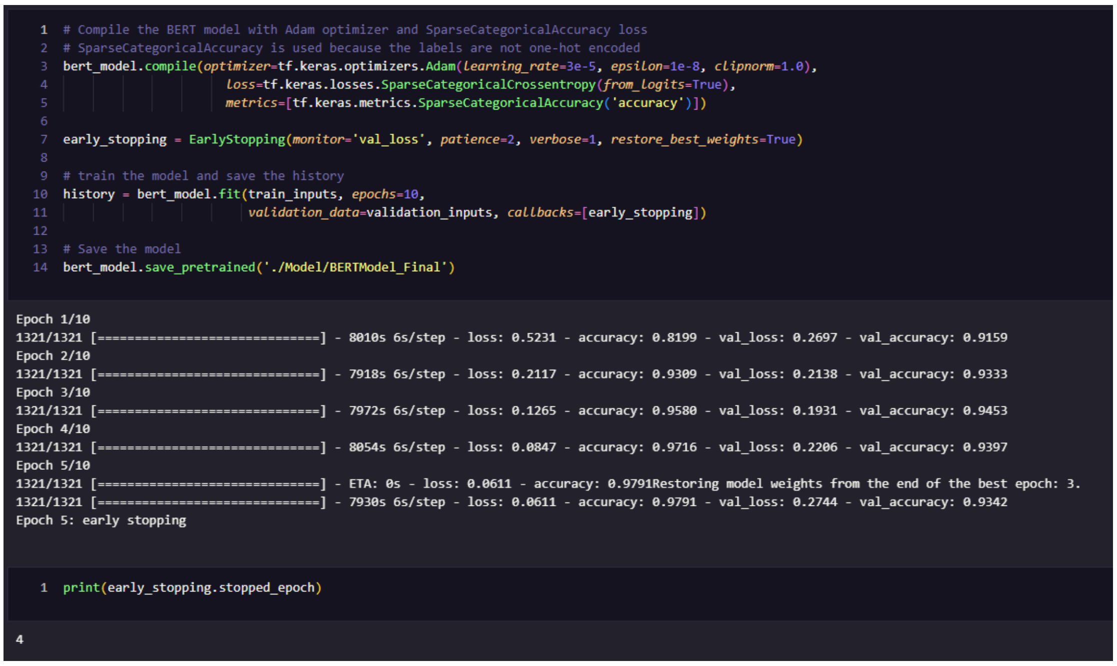


Figure 22: RoBERTa Model Loss during training



```
# Compile the BERT model with Adam optimizer and SparseCategoricalAccuracy loss
# SparseCategoricalAccuracy is used because the labels are not one-hot encoded
bert_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-8, clipnorm=1.0),
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=[tf.keras.metrics.SparseCategoricalAccuracy('accuracy')])

early_stopping = EarlyStopping(monitor='val_loss', patience=2, verbose=1, restore_best_weights=True)

# train the model and save the history
history = bert_model.fit(train_inputs, epochs=10,
                         validation_data=validation_inputs, callbacks=[early_stopping])

# Save the model
bert_model.save_pretrained('./Model/BERTModel_Final')
```

```
Epoch 1/10
1321/1321 [==============================] - 8010s 6s/step - loss: 0.5231 - accuracy: 0.8199 - val_loss: 0.2697 - val_accuracy: 0.9159
Epoch 2/10
1321/1321 [==============================] - 7918s 6s/step - loss: 0.2117 - accuracy: 0.9309 - val_loss: 0.2138 - val_accuracy: 0.9333
Epoch 3/10
1321/1321 [==============================] - 7972s 6s/step - loss: 0.1265 - accuracy: 0.9580 - val_loss: 0.1931 - val_accuracy: 0.9453
Epoch 4/10
1321/1321 [==============================] - 8054s 6s/step - loss: 0.0847 - accuracy: 0.9716 - val_loss: 0.2206 - val_accuracy: 0.9397
Epoch 5/10
1321/1321 [==============================] - ETA: 0s - loss: 0.0611 - accuracy: 0.9791Restoring model weights from the end of the best epoch: 3.
1321/1321 [==============================] - 7930s 6s/step - loss: 0.0611 - accuracy: 0.9791 - val_loss: 0.2744 - val_accuracy: 0.9342
Epoch 5: early stopping
```

```
print(early_stopping.stopped_epoch)
```

```
4
```

Figure 23: Building and training of BERT model

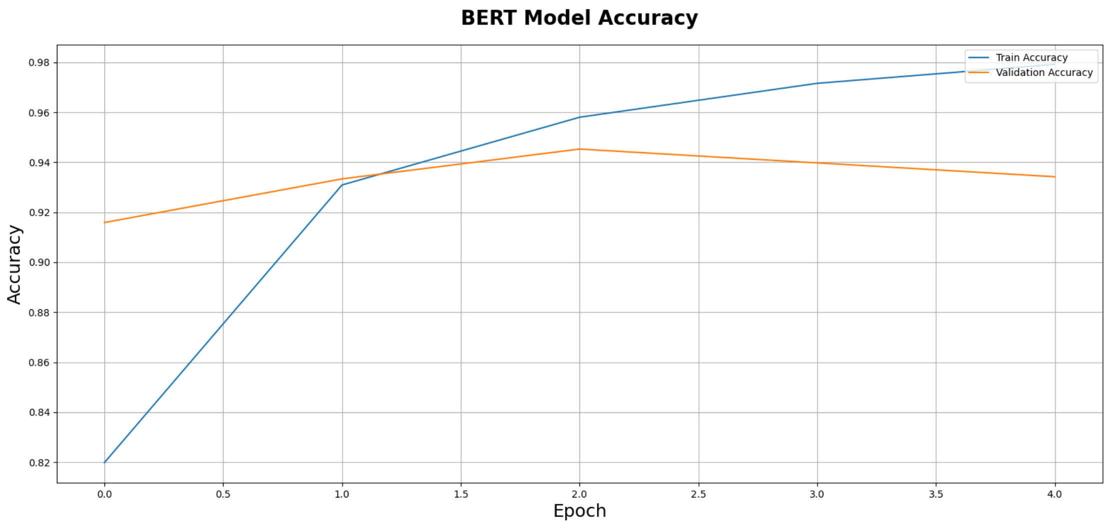


Figure 24: BERT Model Accuracy during training

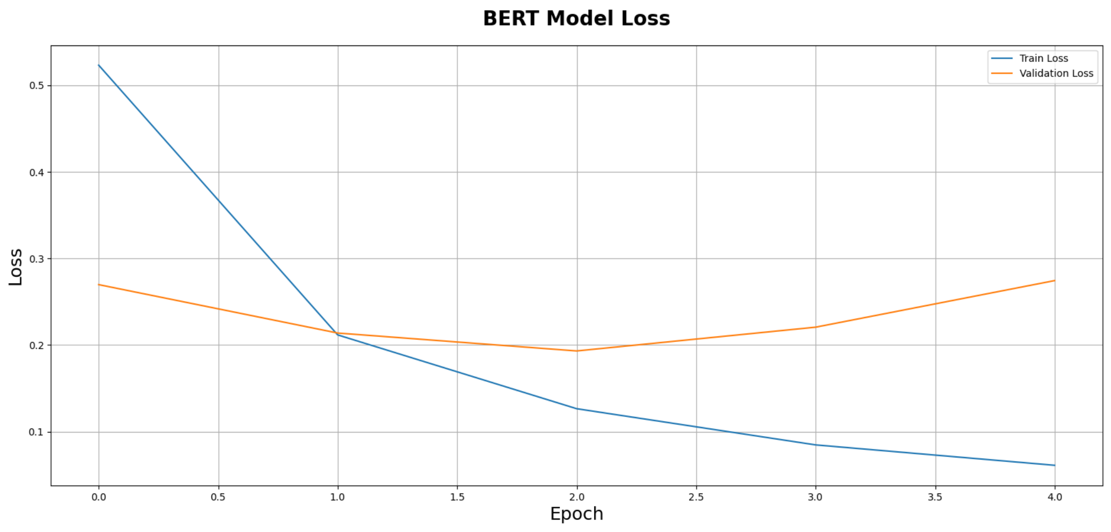
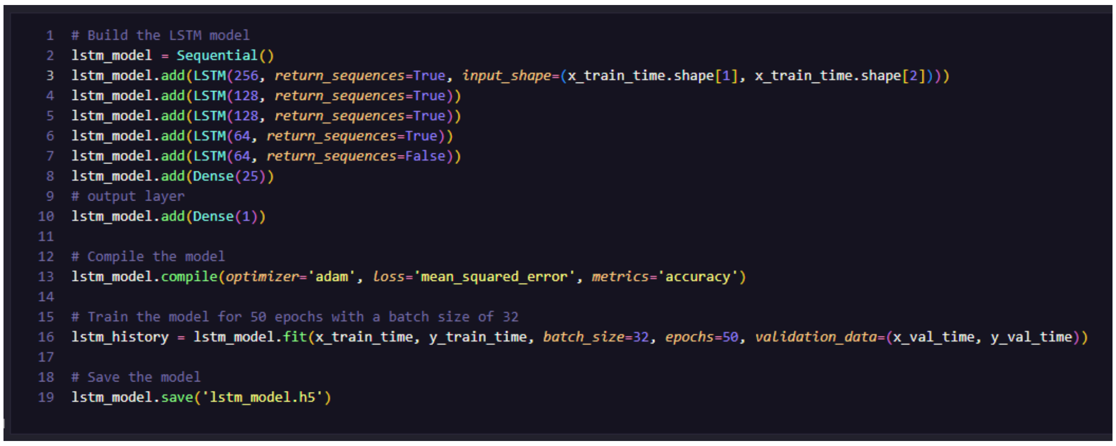


Figure 25: BERT Model Loss during training



```python
# Build the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(256, return_sequences=True, input_shape=(x_train_time.shape[1], x_train_time.shape[2])))
lstm_model.add(LSTM(128, return_sequences=True))
lstm_model.add(LSTM(128, return_sequences=True))
lstm_model.add(LSTM(64, return_sequences=True))
lstm_model.add(LSTM(64, return_sequences=False))
lstm_model.add(Dense(25))
# output layer
lstm_model.add(Dense(1))

# Compile the model
lstm_model.compile(optimizer='adam', loss='mean_squared_error', metrics='accuracy')

# Train the model for 50 epochs with a batch size of 32
lstm_history = lstm_model.fit(x_train_time, y_train_time, batch_size=32, epochs=50, validation_data=(x_val_time, y_val_time))

# Save the model
lstm_model.save('lstm_model.h5')
```

Figure 26: Building and training of LSTM model

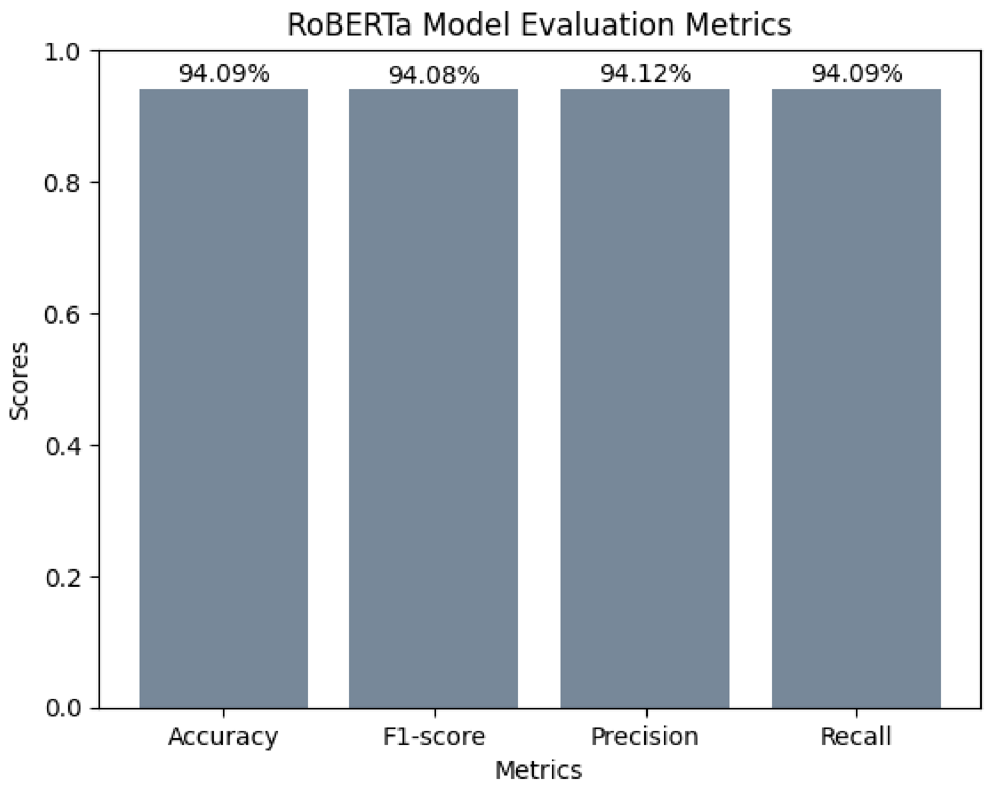
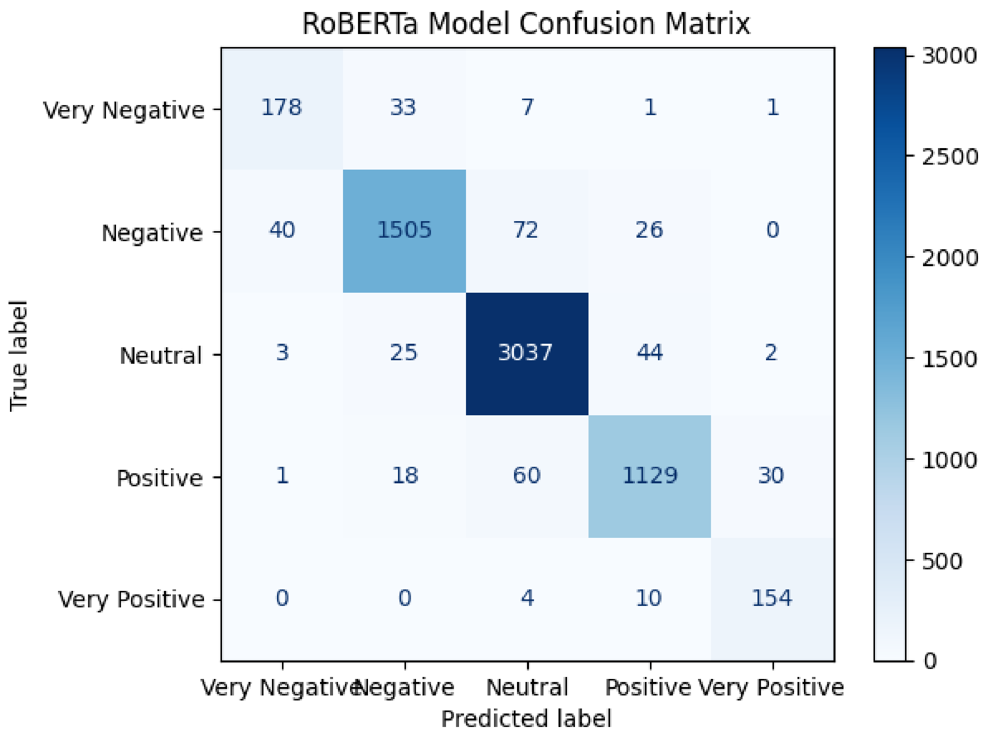


Figure 27: RoBERTa Model Evaluation Metrics



True label \ Predicted label	Very Negative	Negative	Neutral	Positive	Very Positive
Very Negative	178	33	7	1	1
Negative	40	1505	72	26	0
Neutral	3	25	3037	44	2
Positive	1	18	60	1129	30
Very Positive	0	0	4	10	154

Figure 28: Confusion Matrix for RoBERTa Model

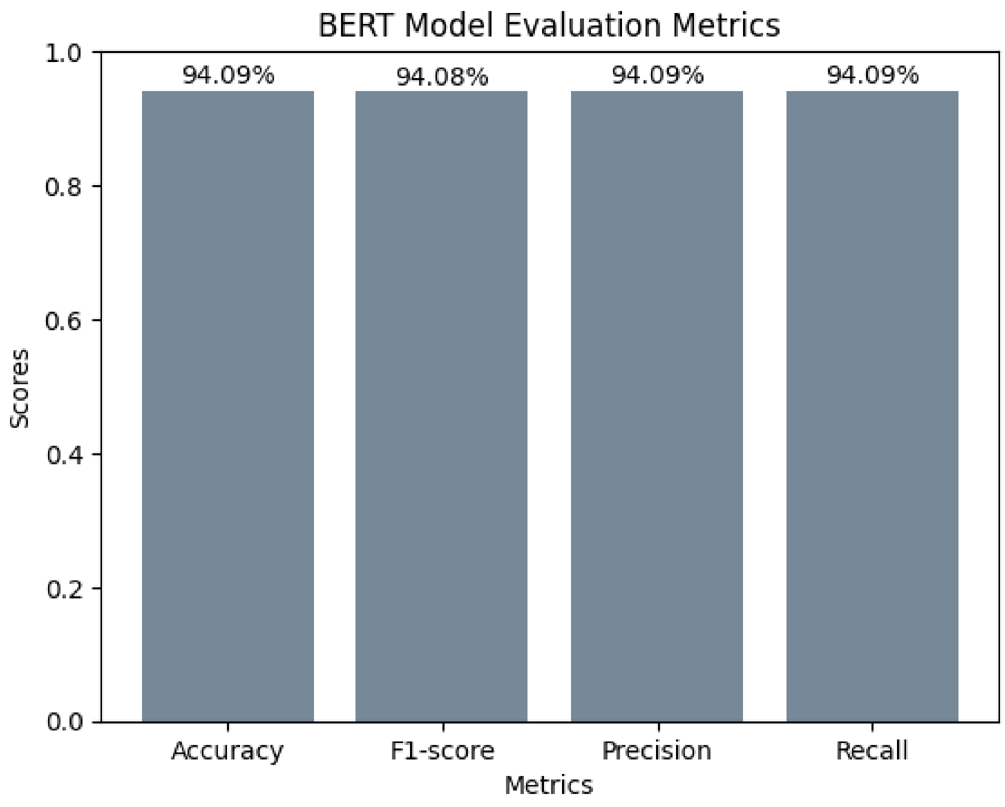
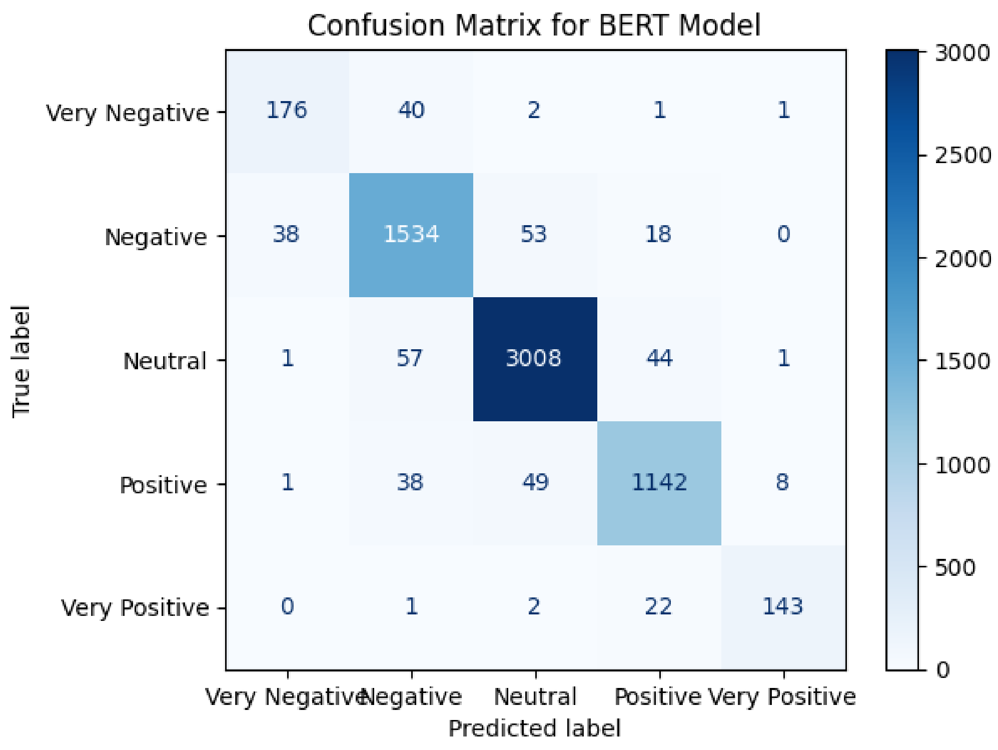


Figure 29: BERT Model Evaluation Metrics



	Very Negative	Negative	Neutral	Positive	Very Positive
Very Negative	176	40	2	1	1
Negative	38	1534	53	18	0
Neutral	1	57	3008	44	1
Positive	1	38	49	1142	8
Very Positive	0	1	2	22	143

Figure 30: Confusion Matrix for BERT Model

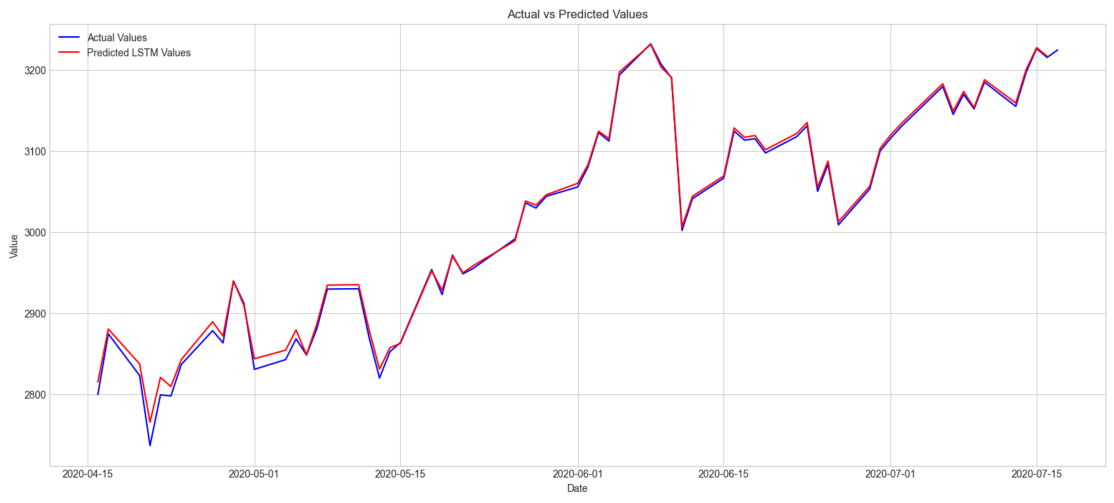


Figure 31: Forecast and Actual data in LSTM (test data)