# Sentiment Analysis on Cryptocurrency-Related Tweets Using Transformer Models

FINAL REPORT

IVAN ZDRAVKOV

## 1. Introduction

- Purpose: To develop a sentiment analysis model for tweets related to Bitcoin.
- Importance: Exploring the impact of sentiment analysis on understanding market trends.

## 2. Data Exploration and Preprocessing

- The dataset is loaded and inspected to understand its structure, identify missing values, and remove duplicates. Basic exploratory data analysis (EDA) is performed to visualize the distribution of sentiments and other relevant features.
- The text data undergoes cleaning using regular expressions and SpaCy, removing noise such as URLs, mentions, special characters, and converting the text to lowercase. Lemmatization is applied to normalize the words.

## 3. Data Preparation and Tokenization

- The cleaned text is tokenized using a pre-trained tokenizer from the twitter-roberta-base-sentiment-latest model, preparing it for input into the transformer model. The data is then split into training, validation, and test sets to evaluate the model's performance.

## 4. Model Fine-Tuning

- The code implements K-fold cross-validation for robust model evaluation and uses Optuna for hyperparameter tuning, optimizing key parameters like learning rate, batch size, and dropout rates.
- The RoBERTa model is fine-tuned on the tweet dataset, with attention to freezing certain layers to prevent overfitting and improve training efficiency.

## 5. Model Evaluation and Interpretation

- The final model is trained using the best hyperparameters obtained from Optuna optimization. The code includes functions for evaluating the model's performance using metrics such as accuracy, precision, recall, and F1-score.

## 6. **Application of the model and recommendations for improvement**

- We will take our model from the tagging face repository and with the help of new responses received using the Twitter API, we will try to predict their sentiment.

# 1. Introduction

## Objective

This project analyzer a data set containing answers related to Bitcoin. The goal is to study the moods in these tweets and build a model to classify their moods.
We will use the **RoBERTa** model, adapted for analyzing tweets, and conduct a detailed analysis of the data.

## Purpose of the Report

This report presents an in-depth sentiment analysis of tweets related to Bitcoin, collected over a period of 18 months. The goal is to understand public sentiment towards Bitcoin, how this sentiment correlates with tweet engagement metrics, and how sentiment trends evolve over time. By analyzing the sentiment behind tweets, we gain valuable insights into public opinion, which can be crucial for investors, analysts, and policymakers. This code represents a comprehensive approach to sentiment analysis using state-of-the-art NLP techniques, ensuring the model is well-prepared to classify the sentiment of tweets accurately.

## Overview of the Data

The data used in this analysis consists of tweets mentioning Bitcoin, collected between January 1, 2022, and June 22, 2023. Each tweet in the dataset is accompanied by various engagement metrics such as the number of likes, retweets, replies, and quotes, as well as a sentiment label and sentiment score.

## Data Source

The dataset used for this analysis is taken from the Kaggle database **https://www.kaggle.com/datasets/sujaykapadnis/bitcoin-tweets**. This dataset includes various response-related metrics such as likes, retweets, citations, and so on. For faster work, we will use **Kaggle Kernel**, which provides 30 hours of free access per week.

Link to our model on Hugging Face
**https://huggingface.co/Poitreqm/poitreqm-model-sentiment-tweets**

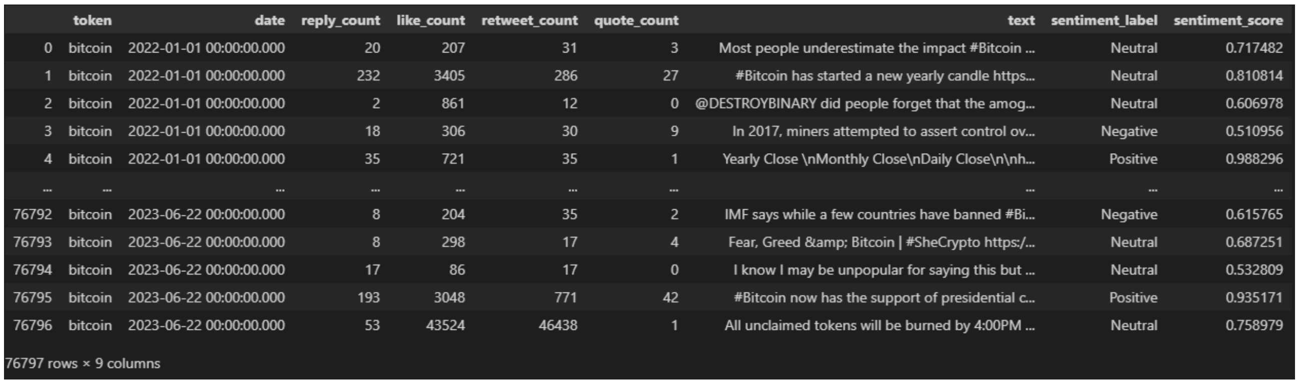
# 2. Data Exploration and Preprocessing

## a. Loading the Dataset and Understanding its structure

This report provides a detailed analysis of the sentiment surrounding Bitcoin as expressed on Twitter. The analysis spans from January 1, 2022, to June 22, 2023, covering a wide range of tweets that include public opinions, news, and discussions related to Bitcoin. Using advanced natural language processing (NLP) techniques, we analyze the sentiment and engagement metrics associated with these tweets to provide insights into public perception and trends in the cryptocurrency space.

```
    # Load the data
        df = pd.read_csv("tweets.csv")
```

**Our table:**



| | token | date | reply_count | like_count | retweet_count | quote_count | text | sentiment_label | sentiment_score |
|---|---|---|---|---|---|---|---|---|---|
| 0 | bitcoin | 2022-01-01 00:00:00.000 | 20 | 207 | 31 | 3 | Most people underestimate the impact #Bitcoin ... | Neutral | 0.717482 |
| 1 | bitcoin | 2022-01-01 00:00:00.000 | 232 | 3405 | 286 | 27 | #Bitcoin has started a new yearly candle https... | Neutral | 0.810814 |
| 2 | bitcoin | 2022-01-01 00:00:00.000 | 2 | 861 | 12 | 0 | @DESTROYBINARY did people forget that the amog... | Neutral | 0.606978 |
| 3 | bitcoin | 2022-01-01 00:00:00.000 | 18 | 306 | 30 | 9 | In 2017, miners attempted to assert control ov... | Negative | 0.510956 |
| 4 | bitcoin | 2022-01-01 00:00:00.000 | 35 | 721 | 35 | 1 | Yearly Close \nMonthly Close\nDaily Close\n\nh... | Positive | 0.988296 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 76792 | bitcoin | 2023-06-22 00:00:00.000 | 8 | 204 | 35 | 2 | IMF says while a few countries have banned #Bi... | Negative | 0.615765 |
| 76793 | bitcoin | 2023-06-22 00:00:00.000 | 8 | 298 | 17 | 4 | Fear, Greed &amp; Bitcoin \| #SheCrypto https:/... | Neutral | 0.687251 |
| 76794 | bitcoin | 2023-06-22 00:00:00.000 | 17 | 86 | 17 | 0 | I know I may be unpopular for saying this but ... | Neutral | 0.532809 |
| 76795 | bitcoin | 2023-06-22 00:00:00.000 | 193 | 3048 | 771 | 42 | #Bitcoin now has the support of presidential c... | Positive | 0.935171 |
| 76796 | bitcoin | 2023-06-22 00:00:00.000 | 53 | 43524 | 46438 | 1 | All unclaimed tokens will be burned by 4:00PM ... | Neutral | 0.758979 |

76797 rows × 9 columns

The dataset contains 76,797 rows and 9 columns. Each row represents a tweet, with columns detailing the tweet's text, sentiment label, sentiment score, date, and various engagement metrics such as likes, retweets, replies, and quotes.

**'token'** - the keyword associated with Bitcoin.
**'date'** - the date and time of the tweet.
**'reply_count'**, **'like_count'**, **'retweet_count'**, **'quote_count'** - engagement metrics.
**'text'** - the content of the tweet.
**'sentiment_label'** - sentiment label (Neutral, Negative, Positive).
**'sentiment_score'** - numerical sentiment score.

```
    # We see information about the dimension of our dataframe
        df.shape

    # Information about the dataset
        df.info()

    # The number of unique values for each column
        df.nunique()
```

***Conclusion:***

*The data is well-structured and complete, with no missing values. This ensures that it's ready for analysis without needing extensive* *preprocessing, allowing for accurate insights into user behavior and sentiment around Bitcoin.*

## b. Perform basic exploratory EDA and cleaning tweets

```
# Check for missing values
      df.isnull().sum()
```

We don't have any missing values.

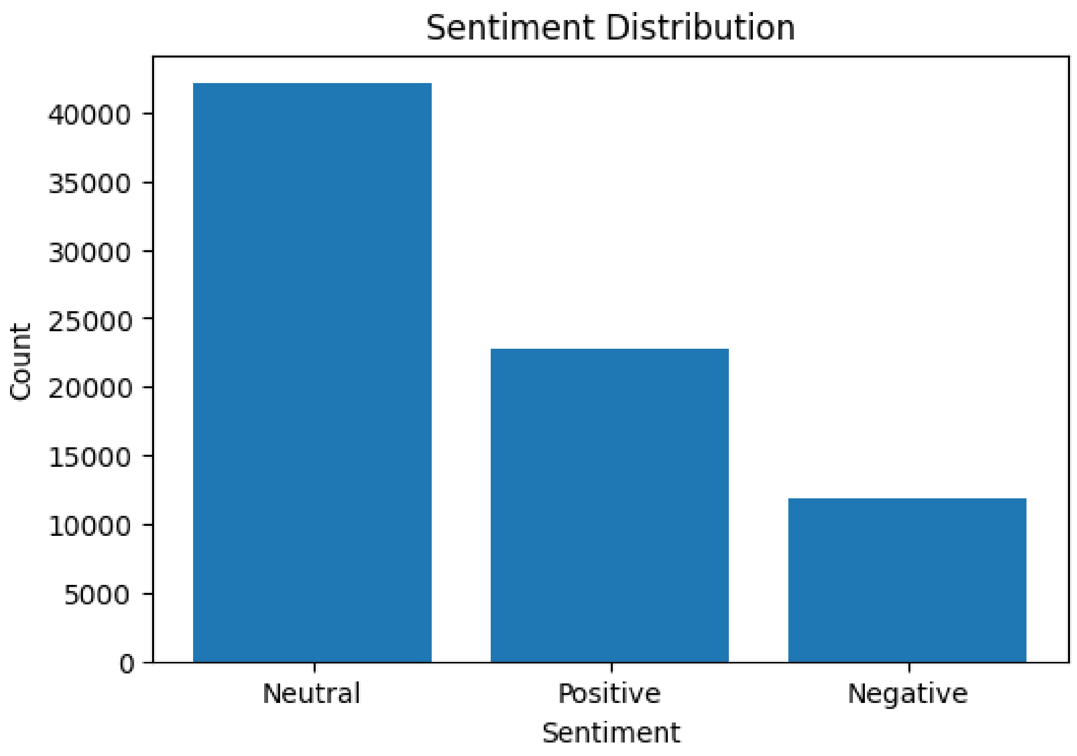
```
# Check for duplicates and non-dublicates
      df.duplicated().value_counts()

      False    76734
      True        63
```

**Removing 63 duplicates:**

```
# Remove duplicates
      df = df.drop_duplicates()
```

```
# Sentiment describe

      df["sentiment_score"].describe()
```

**Using the matplotlib library, we visualize our sentiments:**



***Conclusion:***

*As we can see, we have the most neutral tweets. This may have a slightly bad effect on determining positive and negative moods, since our program will recognize them better than others, especially negative ones, since we have the least of them.*

**Tweets are short messages on the social network Twitter, limited to 280 characters. They can contain text, images, videos, and links. The responses often use trends and memes, slang and abbreviations, as well as emotional expressions and relevant hashtags. Tweets are used to share news, opinions, jokes and to communicate with other users. We will process them using models tailored to the English language**

```
    def clean_text(text):

    text = re.sub(r"^RT\s+", "", text)  # Remove 'RT' (re-tweets) at the
beginning of the text
    text = re.sub(r"http\S+", "", text)  # Remove URLs
    text = re.sub(r"@\w+", "", text)  # Remove mentions (e.g., @username)
    text = re.sub(r'#', '', text)  # Remove the '#' symbol only (e.g., #hashtag)
    text = re.sub(r"[^\x00-\x7F]+", "", text)  # Remove non-ASCII characters
    text = re.sub(r"[^A-Za-z0-9\s.]+", "", text)  # Remove special characters
except periods
    text = re.sub(r"\.{2,}", " ", text)  # Replace multiple periods with a single
space
    text = re.sub(r"\s+", " ", text)  # Replace multiple spaces with a single
space
     text = text.strip()  # Remove spaces from the beginning and end of the
string
    text = text.lower()  # Convert the text to lowercase
```

**Lemmatization is the process of converting words into their basic or dictionary form, called a lemma. The purpose of lemmatization is to combine different forms of a word into one standard form, which helps to improve the accuracy of text analysis and reduce the size of the dictionary.**

```
 # Lemmatization
   doc = nlp(text)
   lemmas = [token.lemma_ for token in doc if token.lemma_ not in ('-', '', ' ')
and len(token.lemma_) > 1]  # Exclude empty lemmas, punctuation marks and single
letters
```

Text cleaning is needed for:

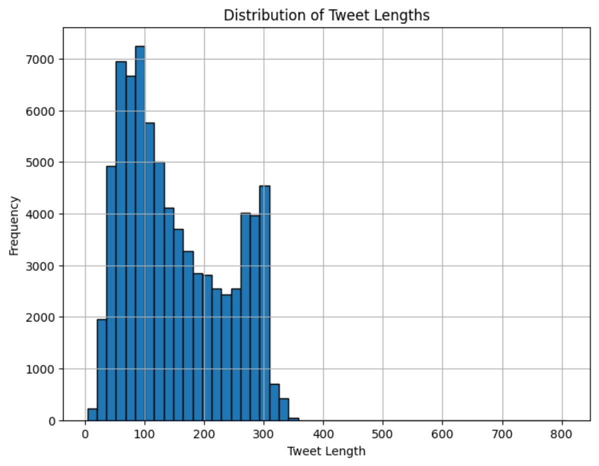
1.Improving Data Quality: Removing errors and noise for accurate analysis.

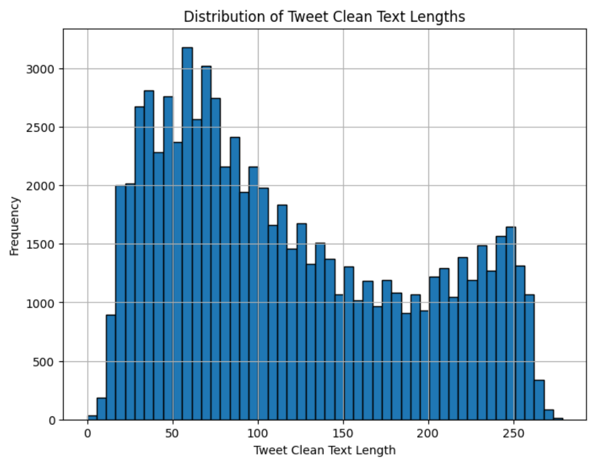
2.Preparing Data: Standardizing text for analysis and machine learning.

3.Simplifying Processing: Eliminating unnecessary information to streamline further work.

4.Resolving Inconsistencies: Standardizing format and style for consistency.

**Before cleaning** **After Cleaning**

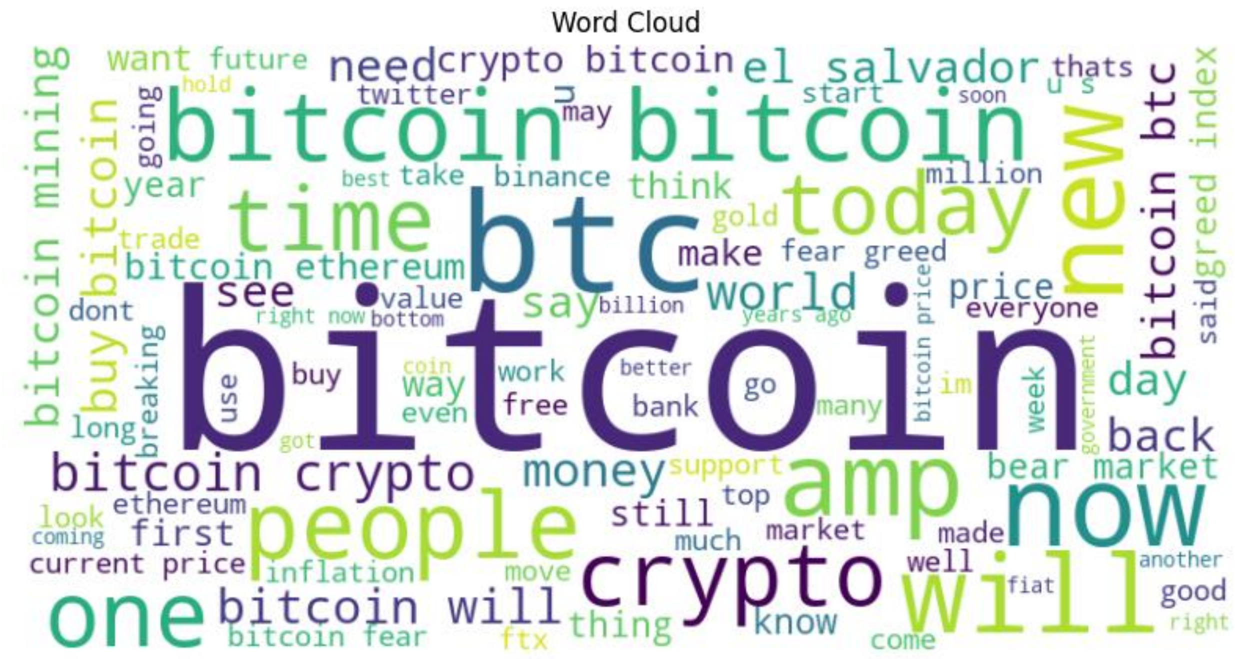




***Conclusion:***

*Almost all the answers were in the range from 0 to 300 - now only up to 250. The length of the cleaned tweets still varies greatly, which indicates that the cleaning has not significantly changed the distribution of tweet lengths, although it is worth recognizing that the number of abnormal tweets has significantly decreased, which will greatly facilitate our work.*

**We find the most common words and visualize them on a graph using the WordCloud library:**



**Top 25 of words:**

| | | |
|---|---|---|
| bitcoin: 34203 | btc: 5713 | will: 5450 |
| go: 3974 | now: 3540 | crypto: 3413 |
| new: 3069 | amp: 2928 | buy: 2911 |
| people: 2803 | make: 2722 | time: 2649 |
| see: 2575 | one: 2556 | break: 2518 |
| say: 2338 | use: 2270 | today: 2189 |
| world: 2174 | money: 2133 | buy bitcoin: 1900 |
| think: 1893 | need: 1883 | day: 1869 |
| bitcoin bitcoin: 1849 | | |

***Conclusion:***

***"Bitcoin"*** *and* ***"BTC":*** *The overwhelming frequency of "bitcoin" and "btc" emphasizes that Bitcoin is the central topic of conversation. The repetition of* ***"bitcoin bitcoin"*** *suggests intense focus or emphasis on the cryptocurrency.*

***"Crypto":*** *The term "crypto" being frequent alongside Bitcoin indicates discussions often encompass the broader cryptocurrency market, though Bitcoin remains the primary subject.*

***"Will"*** *and* ***"Going":*** *These words highlight that future predictions, plans, or expectations are major themes. People are likely discussing what will happen with Bitcoin, where the market is going, and what actions to take.*

***"New," "Now," "Today," "Year":*** *These terms suggest that tweets are often focused on current events, recent developments, or upcoming trends within the crypto space. There's a clear emphasis on what’s happening at the moment and how it relates to the immediate future.*

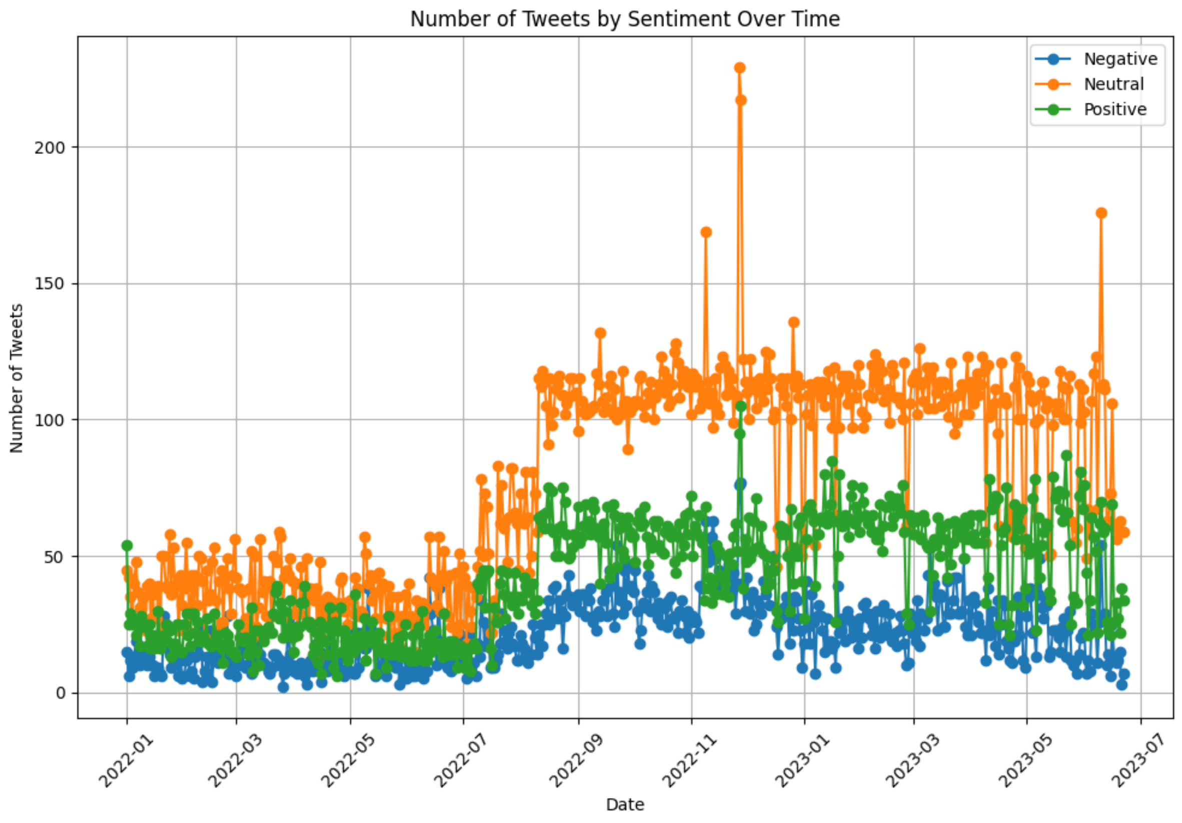
***"Buy," "Money," "Time":*** *The frequent mention of "buy" and "money" suggests discussions about investing in Bitcoin, the value of money, and the timing of these investments. It implies a financial motive or interest in the economics of Bitcoin.*

***"First":*** *This word could relate to milestones, achievements, or important events in the Bitcoin or cryptocurrency world, such as first-time investments, first adopters, or key events.*

***"People," "World," "See," "Day":*** *These words indicate a broader discussion that involves not just technical or financial aspects but also social and global implications. Conversations may involve how Bitcoin is perceived worldwide, its impact on people's lives, and how it integrates into daily life.*

*Overall, these words suggest that your tweets are deeply engaged with both the current state and future possibilities of Bitcoin and the broader cryptocurrency market, with a mix of financial analysis, social implications, and global perspectives.*

**Now let's see how our mood changes in tweets according to time, but before that we need to bring the time to the format we need.**

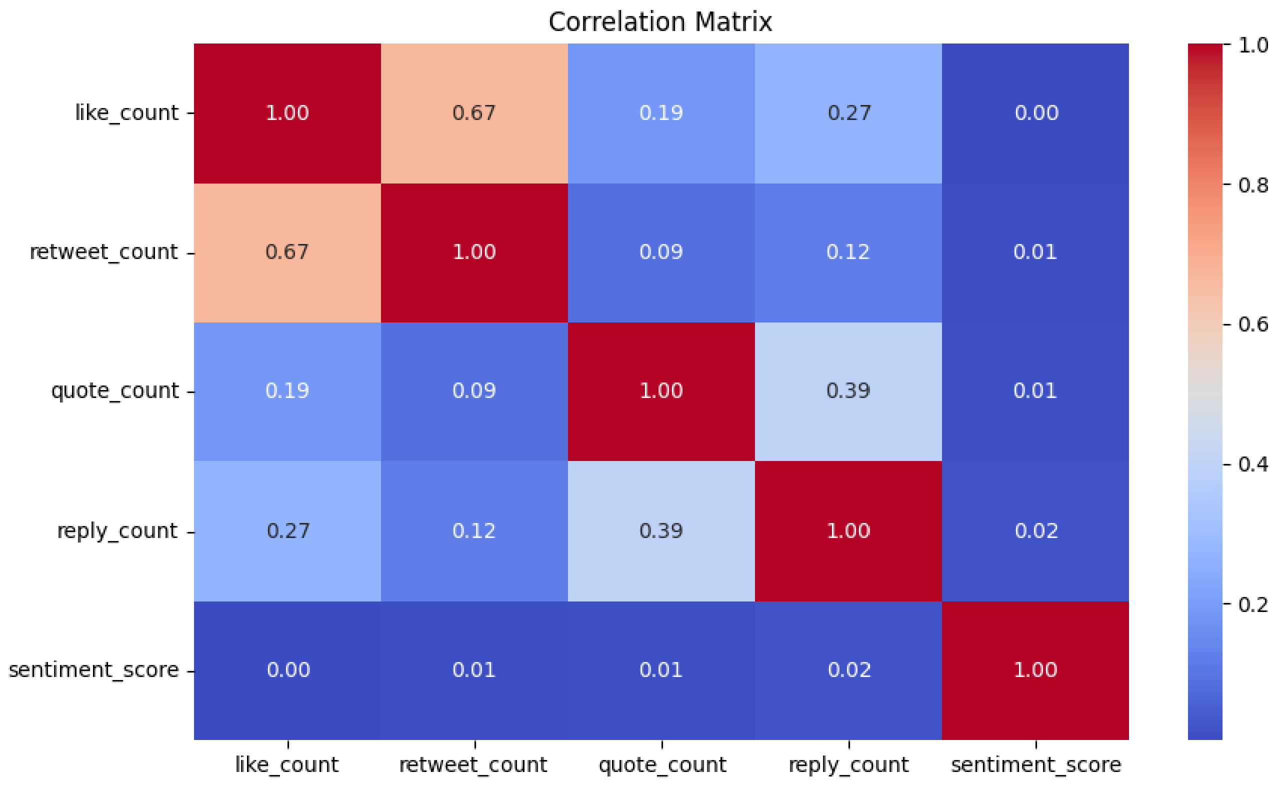


***Conclusion:***

*The data on tweet counts shows that activity varies daily.*

*Periods with a high number of tweets do not always coincide with periods when the average sentiment was higher or lower. This suggests that tweet activity does not always directly affect the overall sentiment on social media. - A sharp increase or decrease in tweet counts may correspond with specific events that influence the overall sentiment.*

**Our correlation matrix helps us understand the relationships between the various variables in our table:**



|  | like_count | retweet_count | quote_count | reply_count | sentiment_score |
|---|---|---|---|---|---|
| like_count | 1.00 | 0.67 | 0.19 | 0.27 | 0.00 |
| retweet_count | 0.67 | 1.00 | 0.09 | 0.12 | 0.01 |
| quote_count | 0.19 | 0.09 | 1.00 | 0.39 | 0.01 |
| reply_count | 0.27 | 0.12 | 0.39 | 1.00 | 0.02 |
| sentiment_score | 0.00 | 0.01 | 0.01 | 0.02 | 1.00 |

***Conclusion:***

*There is a moderate correlation between likes and retweets, indicating that tweets with more likes tend to also have more retweets. However, there is little to no relationship between these metrics and the tweet's sentiment or the number of times the tweet is quoted.*

*Retweets and quotes have a very weak relationship with each other.*

*Sentiment is almost independent of likes, retweets, and quotes.*

*These results suggest that tweet engagement metrics (likes, retweets, and quotes) are generally not strongly related to each other or to the sentiment of the tweets.*

***'sentiment_score'*** *can be considered as an independent variable that is unlikely to be effectively predicted based on interaction data alone*

## 3. Data Preprocessing and Tokenization

### a. Tokenize the text data and preparation of tokenized data for input into model

**We only need two columns - labels and clean tweets:**

```
# We conclude that the other columns do not affect anything. We remove all unnecessary columns

      df = df[["clean_text", "sentiment_label"]]
```

**We use Auto Tokenizer to tokenize text using the RoBERTa model, which is specifically configured for Twitter.**

**Tokenization involves trimming and padding to a specified length, which allows you to prepare data for the model.**

```
# Load the tokenizer for the twitter-roberta-base-sentiment-latest model
      tokenizer = RobertaTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment-latest", truncation=True, do_lower_case=True)
```

*If* `truncation=True`*, the text will be truncated to this maximum length*
*If* `do_lower_case=True`*, all letters in the text will be converted to small ones*

**Tokenization is a key step in text processing that converts text into a format convenient for analysis and processing by machine models:**

```
    # Apply tokenization to the texts
    tokenized_data = tokenizer(df["clean_text"].tolist(),    # tweets
    max_length = 64,                                    # Maximum length of tokenized text
    padding = "max_length",                             # Adding padding
    truncation = True,                                  # Cropping the text if it is longer
    return_tensors = "np",                              # Returned NumPy tensors
    return_attention_mask = True,                      # Attention mask
```

**Creating numeric representations and attention mask:**

```
    input_ids = tokenized_data["input_ids"]                 # numeric
representations of tokens
    attention_masks = tokenized_data["attention_mask"]      # attention mask
```

***Conclusion:***

*This code prepares data for sentiment analysis using the RoBERTa model adapted for Twitter. It provides correct text processing (including padding and cropping), data conversion to a NumPy-compatible format, and includes attention masks to improve model performance. This makes it possible to effectively use the model to analyze sentiment based on tweets.*

```
    # Converting moods to numeric labels
          label_encoder = LabelEncoder()
          df["label"] = label_encoder.fit_transform(df["sentiment_label"])
          label = to_categorical(df["label"], num_classes=3)
```

**To make the cat work so correctly, we will use the Datasets library. We will first need to convert our data to Dataset format, and then use methods from data sets to divide them into training, validation, and test suites.**

```
# Converting labels in the Dataset format
      def create_dataset(input_ids, attention_masks, labels):
            return Dataset.from_dict({
                  "input_ids": list(input_ids),
                  "attention_mask": list(attention_masks),
                  "labels": list(labels)
                  })

# Creating a Complete Dataset
      dataset = create_dataset(input_ids, attention_masks, label)
```

***Conclusion:***

*The Dataset format is used in machine learning and data processing libraries such as Hugging Face's datasets to simplify and standardize working with data.*

*The Dataset format provides a unified interface for working with data, which simplifies their loading, preprocessing and use in models.*

*All data in one format makes it easier to work with data, especially when moving from one processing stage to another.*

```
# Data separation
      dataset_split = dataset.train_test_split(test_size=0.4, seed=42,
shuffle=True)
      test_valid_split =
dataset_split["test"].train_test_split(test_size=0.45, seed=42, shuffle=True)
```

**Why we separate ours data?**

*After you have configured the hyperparameters of the model using KFold cross-validation on training data, you need to have an independent test set on which you can evaluate the final performance of the model.* ***This test set should not be used during the model setup process to avoid overestimating its quality.***

*KFold-cross validation is used to select the best hyperparameters, but even for this process it is useful to divide the data into training and validation parts. For example, you can use one part of the data for cross-validation and hyperparameter selection, and the other part (validation set) to verify that the selected parameters actually work in practice.*

*Pre-partitioning the data and using a fixed test set reduces the impact of randomness on model evaluation and allows you to repeat experiments with the same data to obtain stable results.*

```
# Creating a DatasetDict
      datasets = DatasetDict({
            "train": dataset_split["train"],
            "validation": test_valid_split["train"],
            "test": test_valid_split["test"]
```

***Conclusion:***

*We have successfully converted our data into a Dataset format and divided it into training, validation and test data. We are ready to fine tune our model and train.*

# 4. Model Fine-Tuning

## a. Fine-tune a pre-trained transformer model

**Cross-validation is a method of evaluating the performance of a machine learning model, which helps to avoid overfitting and give more reliable estimates of the quality of the model. We will use k-fold cross validation. It splits the data into parts (folds) and uses each of them as a test sample, and the remaining k-1 parts to train the model. This process is repeated k times, and the model is tested on each of the parts.**

```
# Initialize KFold cross-validator
      kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

**We will use metrics to analyze the state of our model.**

```
      # A function that calculates metrics such as accuracy, precision, recall, and
F1-score
      def compute_metrics(pred):
      labels = np.argmax(pred.label_ids, axis=-1)
      preds = np.argmax(pred.predictions, axis=-1)
      accuracy = accuracy_score(labels, preds)

      precision, recall, f1, _ = precision_recall_fscore_support(labels, preds,
average="weighted")


# Return the metrics as a dictionary
      return {
            "accuracy": accuracy,  # Accuracy of the predictions
            "precision": precision,  # Precision of the predictions
            "recall": recall,  # Recall of the predictions
            "f1": f1,  # F1-score of the predictions
            }
```

**We will create one function for cross-validation and hyperparameters to test them simultaneously. For cross-validation, we will use Sckit-learn and for hyperparameters Optuna**

```
      # Objective function for hyperparameter tuning with Optuna
training_args = TrainingArguments(
   output_dir="./results",  # Directory to save the model
   num_train_epochs=trial.suggest_int("num_train_epochs", 1, 5),  # epochs
   learning_rate=trial.suggest_loguniform("learning_rate", 1e-5, 5e-5),  # speed

per_device_train_batch_size=trial.suggest_categorical("per_device_train_batch_size"
, [8, 16, 32]),  # Batch size
per_device_eval_batch_size=trial.suggest_categorical("per_device_eval_batch_size",
[8, 16, 32]),  # Batch size
    warmup_steps=trial.suggest_int("warmup_steps", 0, 1000),  # warmup steps
    weight_decay=trial.suggest_uniform("weight_decay", 0.0, 0.1),  # L2
regularization
    logging_dir="./logs",  # Directory for logging
    logging_steps=10,  # Logging frequency (steps)
    evaluation_strategy="epoch",  # Evaluate after each epoch
    save_strategy="epoch",  # Save the model after each epoch
    load_best_model_at_end=True,  # Load the best model at the end of training
    metric_for_best_model="eval_loss",  # Metric to use best model
    max_grad_norm=trial.suggest_uniform("max_grad_norm", 0.5, 1.0), # gradient norm
    gradient_accumulation_steps=trial.suggest_int("gradient_accumulation_steps", 1,
4),  # Number of steps to accumulate gradients
        dataloader_num_workers=4,  # Number of workers for data loading
        run_name="none",  #
        report_to="none",  # Disable integration with W&B and other services
    )
```

**As a model, we will use Roberta, who was specially trained on these tweets:**

```
      # Load model with Dropout and L2 regularization
            model = RobertaForSequenceClassification.from_pretrained(
                  "cardiffnlp/twitter-roberta-base-sentiment-latest",
                  num_labels=3,  # Number of output labels
                  ignore_mismatched_sizes=True,  # Ignore size mismatches if they
occur
    )


    # Set dropout probabilities
            model.config.hidden_dropout_prob =
trial.suggest_uniform("hidden_dropout_prob", 0.1, 0.3)  # Dropout probability for
hidden layers
            model.config.attention_probs_dropout_prob =
trial.suggest_uniform("attention_probs_dropout_prob", 0.1, 0.3)  # Dropout
probability for attention layers
```

**To combat overfitting, we will use the Dropout layer, the nature of which we will set using hyperparameters. It will be a good practice to freeze layers, since RoBERTa model can be very powerful, which is why it will not produce the result we need on our data**

```
      # Freeze layers based on the trial's suggestion
            freeze_layers = trial.suggest_int("freeze_layers", 0, 12)  # Example
for Roberta-base with 12 layers
```

```
      for param in model.roberta.parameters():
            param.requires_grad = False  # Freeze all layers initially
      for i in range(freeze_layers, 12):
            for param in model.roberta.encoder.layer[i].parameters():
                  param.requires_grad = True  # Unfreeze specified layers
```

**Having previously divided the data into training and validation, we can proceed to Trainer modulation. Trainer serves to simplify the learning process and evaluate deep learning models for natural language processing (NLP) tasks.**

```
      # Initialize Trainer with the specified arguments and datasets
        trainer = Trainer(
            model=model,
            args=training_args,
            train_dataset=train_dataset,
            eval_dataset=test_dataset,
            tokenizer=tokenizer,
            compute_metrics=compute_metrics,
            callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]  # Early
stopping callback
```

***Conclusion:***

*This code performs automated configuration of hyperparameters of the RoBERTa For SequenceClassificationmodel using the Optuna library and k-fold cross-validation.*

*- Optuna offers hyperparameter values such as the number of epochs, learning rate, batch size, and other model and learning parameters.*

*- The RobertaForSequenceClassification model is set with the ability to adjust Dropout layers and freeze layers.*

*- Data is divided into training and test sets using k-fold cross validation. The model is trained and evaluated on each partition.*

*- The average loss (eval_loss) is calculated for all folds. If the current model is the best, it is saved.*

**Now you can use this model to make predictions on a test dataset.**

**Before we use our model on real answers, we should test our best parameters using a test dataset.**

**Since our best model will be saved locally, we need to specify a path to it.**

```
# Path for the best model
best_model_path = best_trial.user_attrs["model_path"]
```

```
# Final model with optimized dropout values and layer freezes
# Download the best model

final_model = RobertaForSequenceClassification.from_pretrained(
best_model_path, # OUR BEST MODEL
num_labels=3,
ignore_mismatched_sizes=True)

final_model.config.hidden_dropout_prob = best_params["hidden_dropout_prob"]
final_model.config.attention_probs_dropout_prob =
best_params["attention_probs_dropout_prob"]
```

***Conclusion:***

*This code loads the best model found during the experiments and applies optimal values of hyperparameters such as* ***'hidden_dropout_prob'*** *and* ***'attention_probs_dropout_prob'*** *to it to configure the final model before using it in production or for further evaluation.*

**Our best parameters:**

```
num_train_epochs = 3                            # Number of training epochs
learning_rate = 2.3311031477348052e-05         # Small learning rate for fine-tuning
per_device_train_batch_size = 16               # Training batch size per device
per_device_eval_batch_size = 16                # Evaluation batch size per device
warmup_steps = 630      # Warmup steps to gradually increase the learning rate
weight_decay = 0.05180266162382893             # Weight decay rate
max_grad_norm = 0.5757268880209764             # Gradient clipping threshold
gradient_accumulation_steps = 1                # Gradient accumulation steps
hidden_dropout_prob = 0.20787227376469775  # Dropout probability for hidden layers
attention_probs_dropout_prob = 0.2880938516824848  # Dropout probability for
attention probabilities
freeze_layers = 7                         # Number of layers to freeze in the model
```

# 5. Model Evaluation and Interpretation

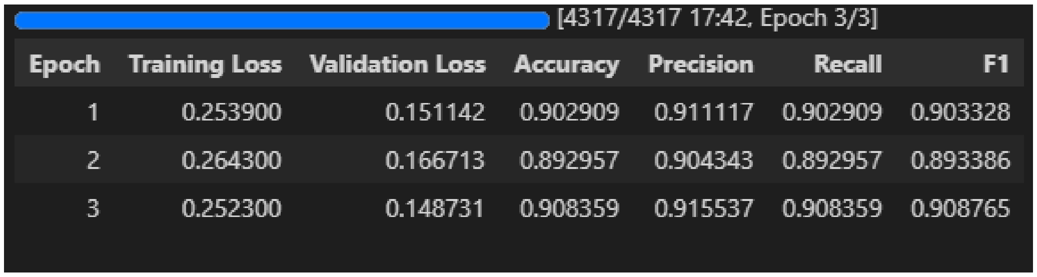
## a. Training and evaluation

As we remember, we saved our data in the 'best params' variable, from where we should get our data about the best model. Now we just need to transfer them to our model

**Fine Tuning our model with best parameters:**

```
final_training_args = TrainingArguments(
    output_dir="./results",  # Save model/results
    num_train_epochs=best_params["num_train_epochs"],  # Optimized epochs
    learning_rate=best_params["learning_rate"],  # Optimized learning rate
    per_device_train_batch_size=best_params["per_device_train_batch_size"],  #
```

```
      #Optimized train batch
    per_device_eval_batch_size=best_params["per_device_eval_batch_size"],  #
    #Optimized eval batch
    warmup_steps=best_params["warmup_steps"],  # Optimized warmup
    weight_decay=best_params["weight_decay"],  # Optimized weight decay
    logging_dir="./logs",  # Log directory
    logging_steps=10,  # Log frequency
    evaluation_strategy="epoch",  # Evaluate each epoch
    save_strategy="epoch",  # Save each epoch
    load_best_model_at_end=True,  # Load best model
    metric_for_best_model="eval_loss",  # Best model metric
    max_grad_norm=best_params["max_grad_norm"],  # Optimized grad norm
    gradient_accumulation_steps=best_params["gradient_accumulation_steps"],  #
#Optimized gradient accumulation
    dataloader_num_workers=4,  # Data loader workers
    run_name="none",  # No run name
    report_to="none",  # No reporting
```



[4317/4317 17:42, Epoch 3/3]

| Epoch | Training Loss | Validation Loss | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|---|---|
| 1 | 0.253900 | 0.151142 | 0.902909 | 0.911117 | 0.902909 | 0.903328 |
| 2 | 0.264300 | 0.166713 | 0.892957 | 0.904343 | 0.892957 | 0.893386 |
| 3 | 0.252300 | 0.148731 | 0.908359 | 0.915537 | 0.908359 | 0.908765 |

***Conclusion:***

*-The model shows improvement in validation loss from Epoch 1 to Epoch 3 (from 0.151142 to 0.148731), indicating that the model is learning and generalizing better over time.*

*-Accuracy and F1-score both improve from Epoch 2 to Epoch 3, suggesting that the model's predictions are becoming more consistent with the true labels.*

*-The slight dip in these metrics from Epoch 1 to Epoch 2 might indicate temporary overfitting or noise, but the improvement in Epoch 3 suggests recovery.*

*-Precision and recall are fairly stable across epochs, with a slight increase in Epoch 3, which shows the model maintains a balance between false positives and false negatives.*

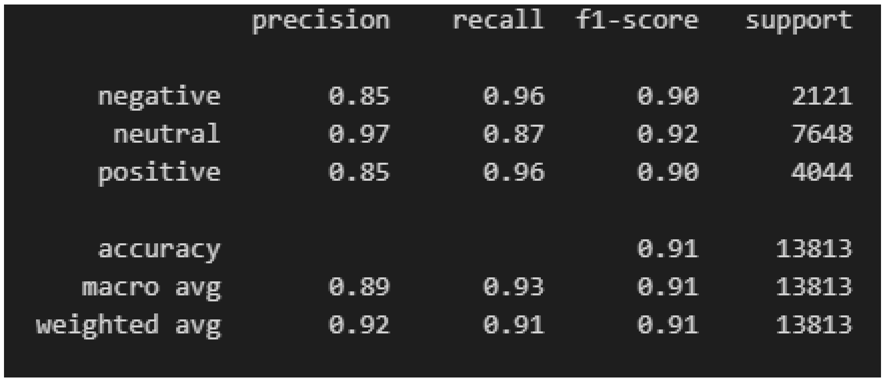
*- The model appears to be converging, with the best performance observed at Epoch 3, as indicated by the lowest validation loss and highest accuracy, precision, recall, and F1-score.*

*Overall, the tuned hyperparameters led to an **effective model** that generalizes well to unseen data, as evidenced by the strong validation performance in the final epoch.*

**Now let's predict our test data***:*

```
    # Prediction
        predictions = final_trainer.predict(datasets['test'])
        preds = np.argmax(predictions.predictions, axis=1)
        true_labels = np.argmax(predictions.label_ids, axis=1)
```

**Classification Report:**



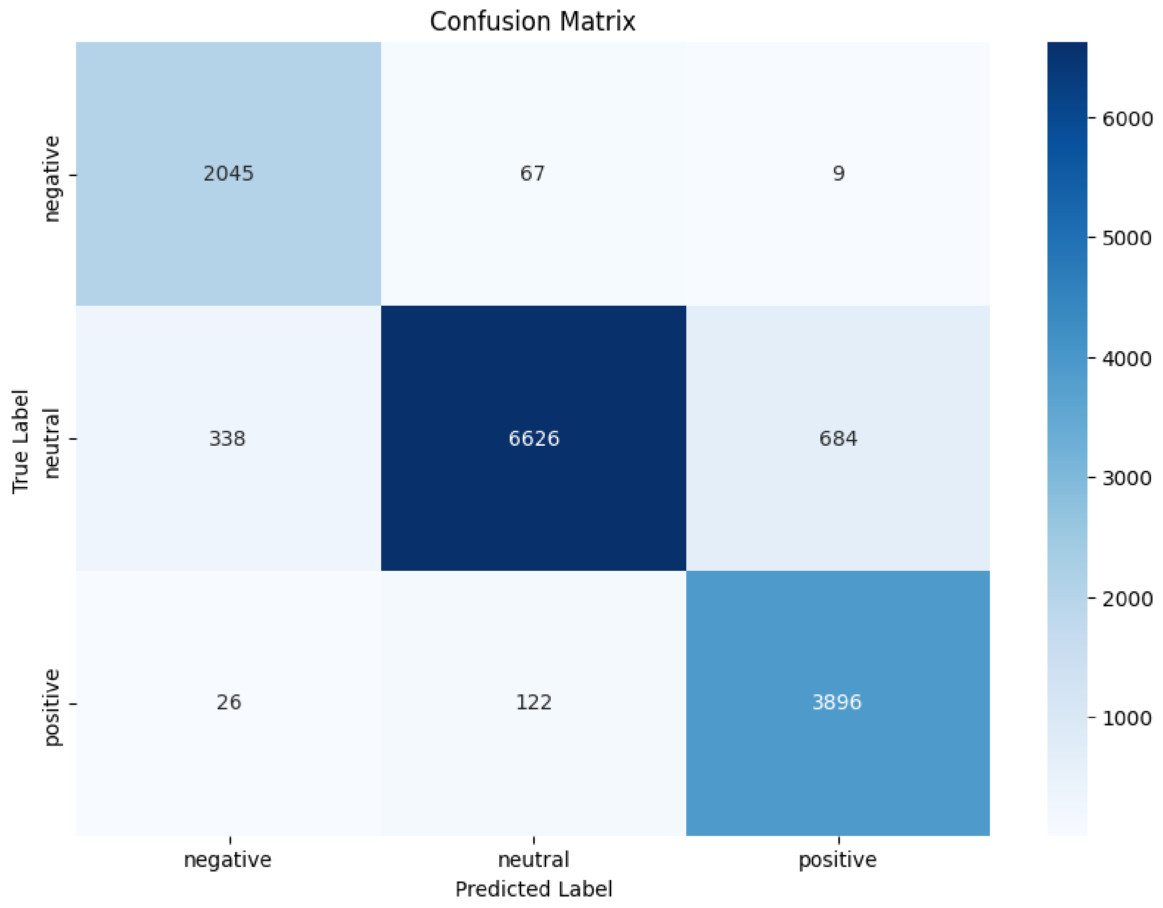
```
              precision    recall  f1-score   support

    negative       0.85      0.96      0.90      2121
     neutral       0.97      0.87      0.92      7648
    positive       0.85      0.96      0.90      4044

    accuracy                           0.91     13813
   macro avg       0.89      0.93      0.91     13813
weighted avg       0.92      0.91      0.91     13813
```

***Conclusion:***

*The model does a* ***good job of classifying reviews into neutral and positive****, but has some problems with accuracy in classifying negative reviews and completeness for positive ones.*

*The* ***Neutral class has the best performance across all metrics****, which may indicate that the model is better trained for this class or that this class prevails in the data.*

*The micro avg and weighted avg indicators confirm that the* ***model is generally balanced****, although there are minor differences between classes.*



***Conclusion:***

***The model performs well in predicting Class Negative****, with 2045 true positives.*

*Misclassifications are relatively low, with 67 instances being incorrectly classified as Class Neutral and 9 as Class Positive.*

***The model shows strong performance for Class Neutral****, with 6626 true positives.*

*However, there are 338 instances of Class Neutral being misclassified as Class Negative and 684 as Class Positive, indicating some confusion primarily with Class Positive.*

***Class Positive predictions are also strong, with 3896 true positives****.*

*There are 122 instances of Class Positive incorrectly predicted as Class Neutral and only 26 as Class Negative, showing minimal confusion with Class Negative but some with Class Neutral. The model performs well across all classes, with the highest accuracy for Class Neutral but some confusion between Class Neutral and Class Positive. The misclassifications are relatively low, suggesting that the model is generally effective in distinguishing between the three classes.*

**And although all the conclusions speak of our model as very good, How do we know exactly how many correct answers we have received? To do this, we will calculate accuracy and loss error specifically for our predicted data**

```
# Correct and incorrect prediction
correct_predictions = (df_results['True Labels'] ==
df_results['Predictions']).sum()
total_predictions = df_results.shape[0]
accuracy = (correct_predictions / total_predictions) * 100
error_rate = 100 - accuracy

# Result
print(f'Accuracy: {accuracy:.2f}%')
print(f'Error Rate: {error_rate:.2f}%')
```

```
Accuracy: 90.98%
Error Rate: 9.02%
```

**Before downloading our model, we need to get a token from the site and login. Upload our model in Hugging Face.**

```
    # Specify the path to the saved model and tokenize
        model_directory = "/kaggle/working/poitreqm-model-sentiment"
    #Specify the repository name on Hugging Face
        repository_name = "Poitreqm/poitreqm-model-sentiment-tweets"

    # Create new repo on Hugging Face
        api = HfApi()
        api.create_repo(repo_id=repository_name)

    # Upload model
        final_model.push_to_hub(repository_name)
        tokenizer.push_to_hub(repository_name)
```

***Conclusion:***

*We successfully trainers our model using hyperparameters, tested it on validation data and then later on test data. The result on the test data was 90%. We have successfully uploaded our model to Hugging Face and now we can always use it for our calculations.*

*Link to our model:* **https://huggingface.co/Poitreqm/poitreqm-model-sentiment-tweets**

# 6. Application of the model and recommendations for improvement

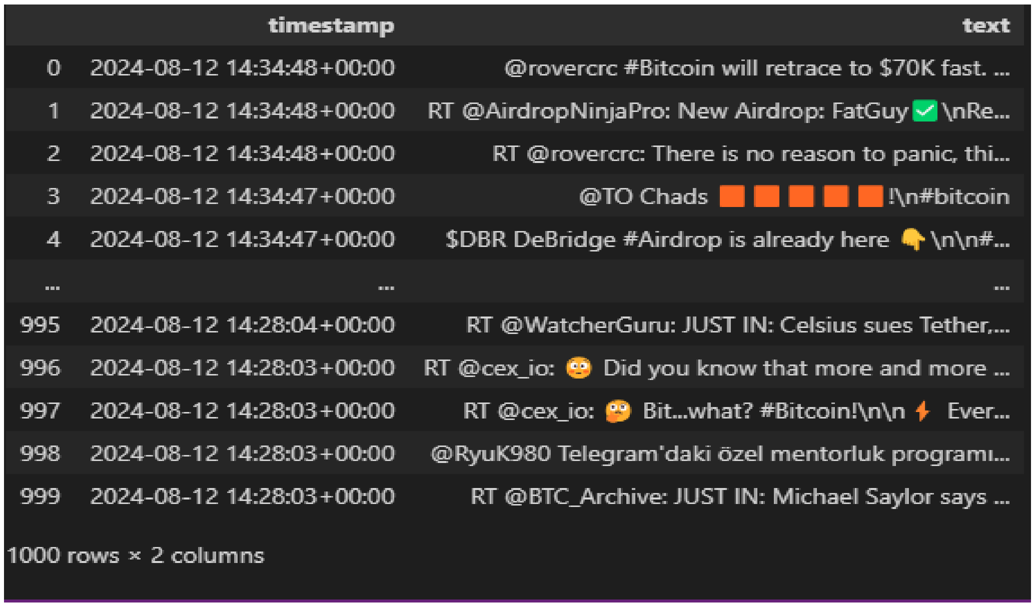
### a. Twitter API request

**After all the manipulations and tests, we can finally come to the final stage. Receiving tweets using the Twitter Api. We will need 5**

**keys to connect. Which can be found in your twitter api account. We will need the tweepy library for queries.**

```
def fetch_tweets(query, count):
      tweets = tweepy.Paginator(client.search_recent_tweets,
                                  query=query,
                                  tweet_fields=["created_at", "text"],
                                  max_results=100
                                  ).flatten(limit=count)
tweets_list = [[tweet.created_at, tweet.text] for tweet in tweets]
return pd.DataFrame(tweets_list, columns=["timestamp", "text"])


df = fetch_tweets("#Bitcoin", 1000) # 1000 tweets
df.to_csv("big-files/crypto_tweets.csv", index=False)
```

***Conclusion:***
*This request will allow us to pull out 1000 tweets and save them in the crypto_tweets.csv file*



```
                     timestamp                                               text
  0  2024-08-12 14:34:48+00:00   @rovercrc #Bitcoin will retrace to $70K fast. ...
  1  2024-08-12 14:34:48+00:00  RT @AirdropNinjaPro: New Airdrop: FatGuy ✅ \nRe...
  2  2024-08-12 14:34:48+00:00     RT @rovercrc: There is no reason to panic, thi...
  3  2024-08-12 14:34:47+00:00                     @TO Chads 🟧🟧🟧🟧🟧!\n#bitcoin
  4  2024-08-12 14:34:47+00:00  $DBR DeBridge #Airdrop is already here 👇 \n\n#...
...                        ...                                                 ...
995  2024-08-12 14:28:04+00:00     RT @WatcherGuru: JUST IN: Celsius sues Tether,...
996  2024-08-12 14:28:03+00:00  RT @cex_io: 😳 Did you know that more and more ...
997  2024-08-12 14:28:03+00:00      RT @cex_io: 🤔 Bit...what? #Bitcoin!\n\n ⚡ Ever...
998  2024-08-12 14:28:03+00:00  @RyuK980 Telegram'daki özel mentorluk programı...
999  2024-08-12 14:28:03+00:00    RT @BTC_Archive: JUST IN: Michael Saylor says ...

1000 rows × 2 columns
```

**We are manipulating the received data:**

```
# Clean tweets
      tweets['clean_text'] = tweets['text'].apply(clean_text)
```

**Now let's predict the sentiment of our new tweets based on our model that we have trained:**

```
      # 'username/repo_name'
            model_name = "Poitreqm/poitreqm-model-sentiment-tweets"

      # Download model
            model = AutoModelForSequenceClassification.from_pretrained(model_name)

      # Download tokenizer
            tokenizer = AutoTokenizer.from_pretrained(model_name)


      # Tokenizing text
tokenized_tweets_pred = tokenizer(tweets['clean_text'].tolist(),
                             max_length=64,  # Need the same line like in train
                             padding='max_length',
                             truncation=True,
                             return_tensors='np',
                              return_attention_mask=True)


      # input_ids and attention_mask
      input_ids_pred = tokenized_tweets_pred['input_ids']
      attention_masks_pred = tokenized_tweets_pred['attention_mask']

      # Our Argiments
      tweet_pred_args = TrainingArguments(
            output_dir="./results",
            run_name="none",
            report_to="none",

      # Our trainer
      trainer = Trainer(model=model,
                 args=tweet_pred_args,)
```

***Conclusion:***

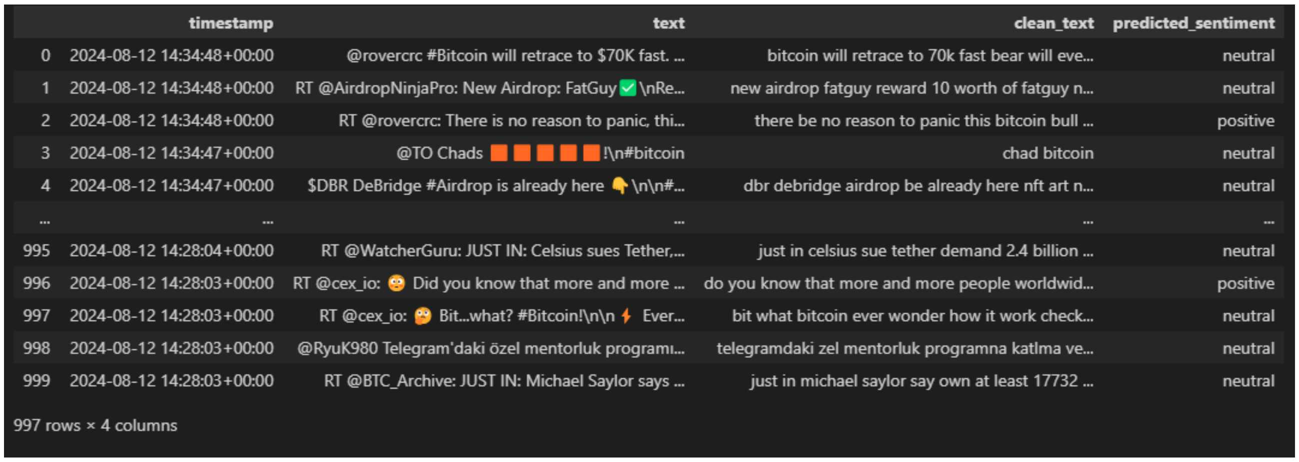
As we can see, we have to put our new tweets through the same process as with our training data: cleanup, tokenization, fine-tuning, training. All data must also be used in the Dataset type.

```
      # Prediction
            predictions = trainer.predict(dataset)
            preds = np.argmax(predictions.predictions, axis=1)
```

We check for duplicates and missing data. this rarely happens, but it won't be superfluous to check:

```
# Check for missing values
      tweets.isnull().sum()

# Check for duplicates and non-dublicates
      tweets.duplicated().value_counts()

# Remove duplicates
      tweets = tweets.drop_duplicates()
```

**We found 3 duplicates that we have successfully deleted:**



| | timestamp | text | clean_text | predicted_sentiment |
|---|---|---|---|---|
| 0 | 2024-08-12 14:34:48+00:00 | @rovercrc #Bitcoin will retrace to $70K fast. ... | bitcoin will retrace to 70k fast bear will eve... | neutral |
| 1 | 2024-08-12 14:34:48+00:00 | RT @AirdropNinjaPro: New Airdrop: FatGuy✅\nRe... | new airdrop fatguy reward 10 worth of fatguy n... | neutral |
| 2 | 2024-08-12 14:34:48+00:00 | RT @rovercrc: There is no reason to panic, thi... | there be no reason to panic this bitcoin bull ... | positive |
| 3 | 2024-08-12 14:34:47+00:00 | @TO Chads 🟧🟧🟧🟧🟧!\n#bitcoin | chad bitcoin | neutral |
| 4 | 2024-08-12 14:34:47+00:00 | $DBR DeBridge #Airdrop is already here 👇\n\n#... | dbr debridge airdrop be already here nft art n... | neutral |
| ... | ... | ... | ... | ... |
| 995 | 2024-08-12 14:28:04+00:00 | RT @WatcherGuru: JUST IN: Celsius sues Tether,... | just in celsius sue tether demand 2.4 billion ... | neutral |
| 996 | 2024-08-12 14:28:03+00:00 | RT @cex_io: 😳 Did you know that more and more ... | do you know that more and more people worldwid... | positive |
| 997 | 2024-08-12 14:28:03+00:00 | RT @cex_io: 🙄 Bit...what? #Bitcoin!\n\n ⚡ Ever... | bit what bitcoin ever wonder how it work check... | neutral |
| 998 | 2024-08-12 14:28:03+00:00 | @RyuK980 Telegram'daki özel mentorluk programı... | telegramdaki zel mentorluk programna katlma ve... | neutral |
| 999 | 2024-08-12 14:28:03+00:00 | RT @BTC_Archive: JUST IN: Michael Saylor says ... | just in michael saylor say own at least 17732 ... | neutral |

997 rows × 4 columns

```
predicted_sentiment
neutral     671
positive    252
negative     74
```

## b. Recommendations for improvement

To improve our model, consider the following steps:

**Increase the amount of data**:

Collect more data or apply data augmentation. This will help the model better identify patterns and improve its performance.

**Using linear regression instead of classification:**

If the task requires an assessment of degree or intensity, rather than a simple assignment of classes, try linear regression. This will allow you to obtain quantitative estimates and probabilities.

**Applying multiple models:**

Combining different algorithms can improve the accuracy and reliability of predictions.

**Expanding the range of responses:**

Switch to five classes (negative, neutral-negative, neutral, neutral-positive, positive) to better reflect the gradations in the data. Update the loss function and evaluation metrics for a multi-class task.

**Increase Cross-validation and Hyperparameters**

The increased capacity will allow us to use more parameters in cross-validation and hyperparameters, which in turn will allow a better model.