Sentiment Analysis of Social Media Data using LLMs

Project Final Report

CIS 660 - Data Mining Section 1

Name: Prasanna Aditya Kandarpa

CSU ID: 2896149

Contents

Introduction of the Project:

This project performs Sentiment Analysis using a dataset containing social media posts with 15 columns, which include textual data, sentiment labels, timestamps, usernames, social media platforms, hashtags and other demographic information.

The primary goal is to classify each record's sentiment label into one of the three major categories: **Positive**, **Negative**, or **Neutral**. I perform the classification using transformer-based Large Language Models (LLMs) like **BERT** or **GPT-4.o**.

Input Dataset:

Source of Input Dataset:

The Input Dataset **'sentimentdataset.csv'** is taken from **'Kaggle'** platform. Below is the URL to download the csv file:

https://www.kaggle.com/datasets/kashishparmar02/social-media-sentiments-analysis-dataset

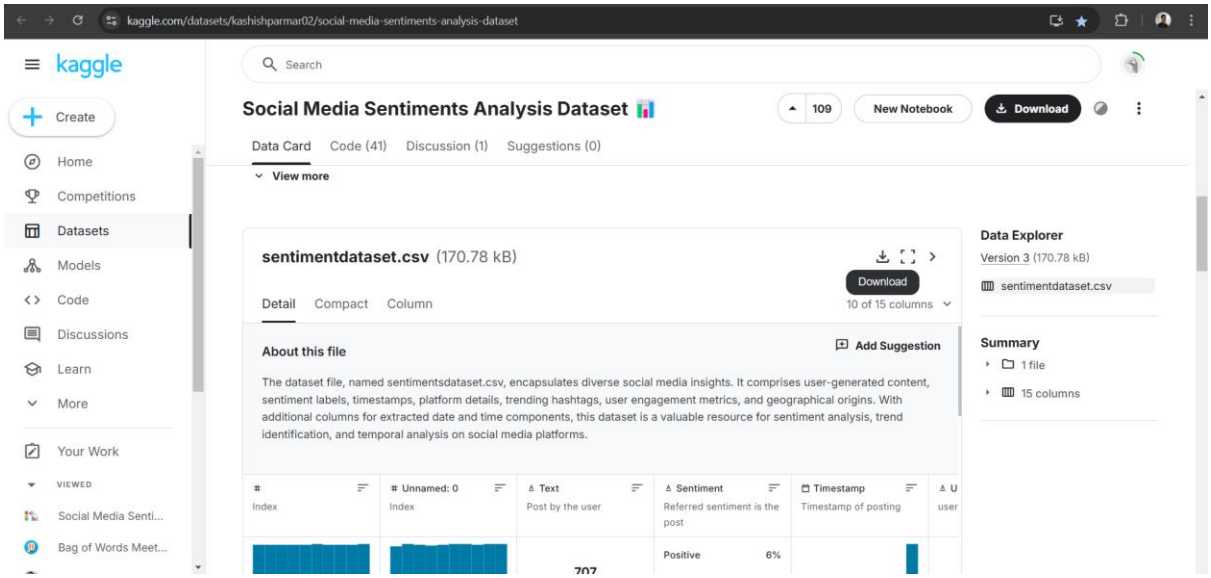


Figure 1: Input Dataset Download Link

Dataset Description:

The dataset consists of **15 columns** and **732 entries** that seem to cover a variety of characteristics related to social media posts. Important columns consist of:
- Text: Each post's textual content.
- The sentiment label, such as "positive," "negative," or "neutral," indicates the sentiment.
- Timestamp, User, Platform, and Country: Information about the author, platform, location, and timing of the post.
- Hashtags: A list of the post's related hashtags.
- Likes and retweets are indicators of engagement.

Configuration of the Environment:

I have used **Google Colab Notebook (with T4 and A100 GPUs)** for faster computation of complex dataset and **Python** programming language to implement the Project.

Importing the Required Libraries:

First, I import the standard libraries **Pandas** (to import, load and read the Dataset as a Data frame and perform the preprocessing operations on it) and **re** (for using regular expressions to perform the data preprocessing).

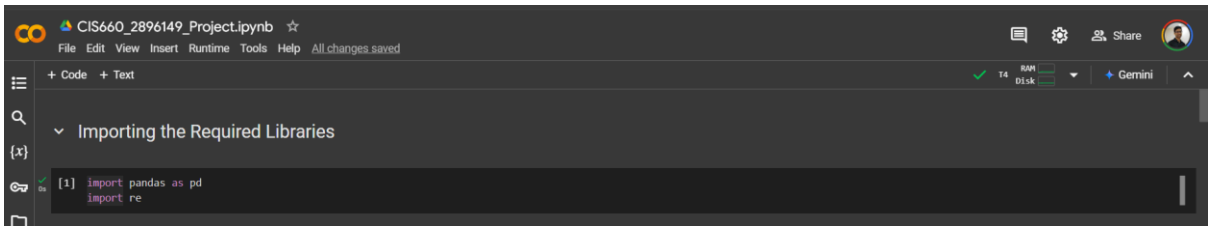


Figure 2: Importing the Required Libraries

Reading the Input Dataset as a Data frame:

I upload the **sentimentdataset.csv** file into my Colab Notebook and read it as a Pandas Data frame **input_df** as below:

Figure 3: Upload the csv file

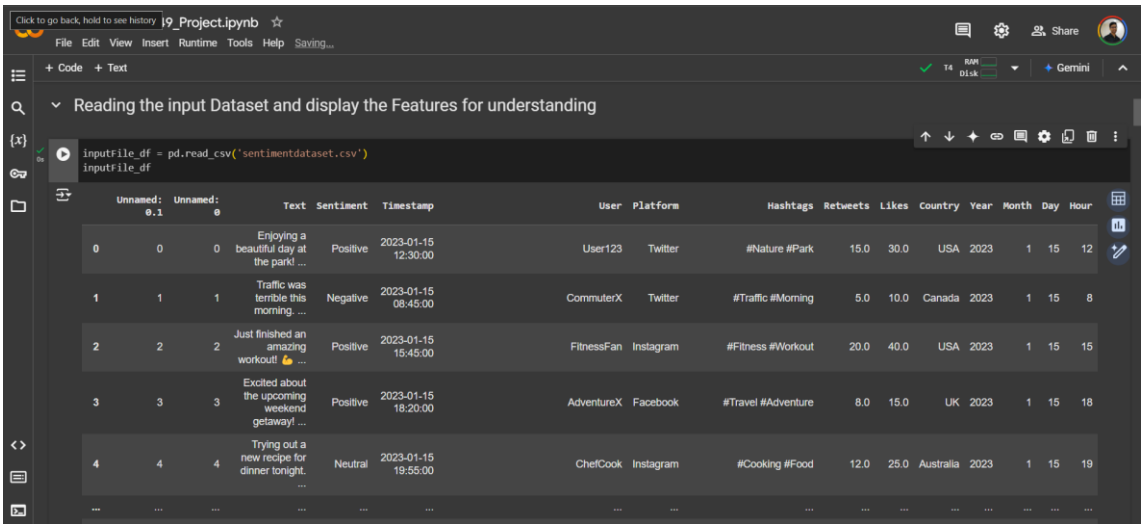


Figure 4: Reading the Dataset

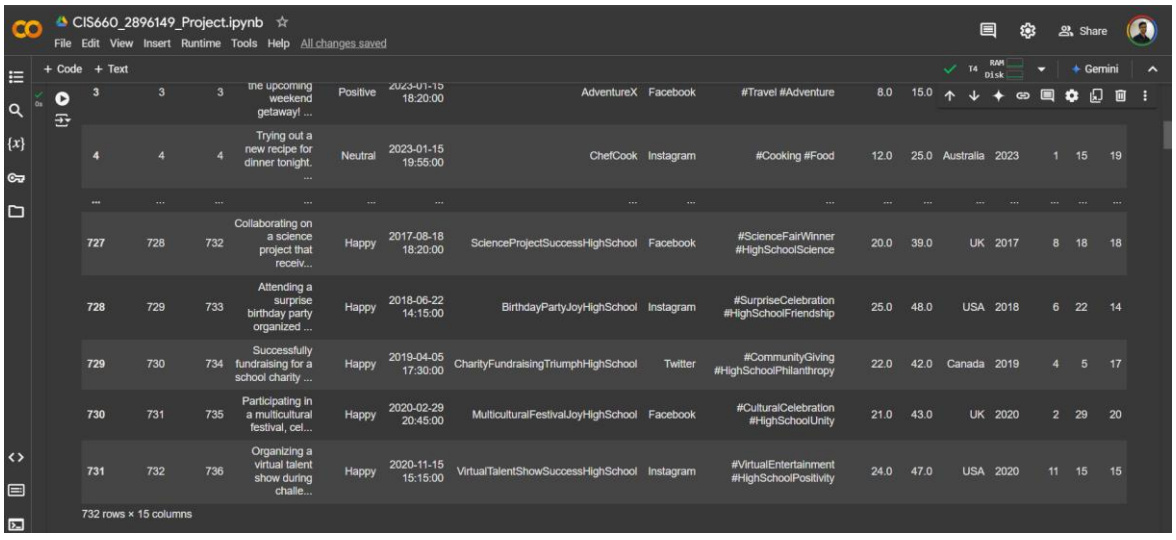


Figure 5: Reading the Dataset (contd..)

Dataset Information:

I display the information about the columns/features on the dataset as below:

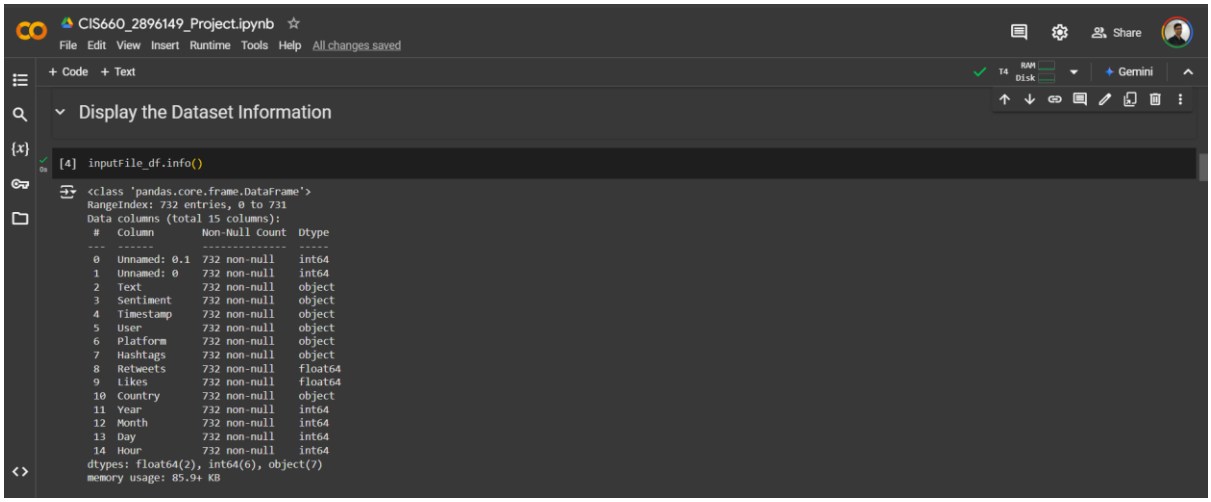


Figure 6: Display the Dataset Information

The input dataset contains **732 unique and Non-Null records** which represent the Social Media Sentiments captured from various users and different social media platforms. There are 15 columns/features in the dataset as shown in the above figure.

There are **6 columns ('Unnamed: 0.1', 'Unnamed: 0', 'Year', 'Month', 'Month', 'Hour') of Integer type, 2 columns ('Retweets', 'Likes') of Floating-Point type and 7 columns ('Text', 'Sentiment', 'Timestamp', 'User', 'Platform', 'Hashtags') of Object (Mixed Data or Non-Numeric) type.**

Generating more number of records by analysing the existing records in the Input Dataset:

I take the input dataset and generate **100000 Synthetic records** which are **similar but not identical** to the original dataset records. I perform this operation to expand my smaller input dataset for the Machine Learning operations and test the robustness of the model with more diverse and randomized data.

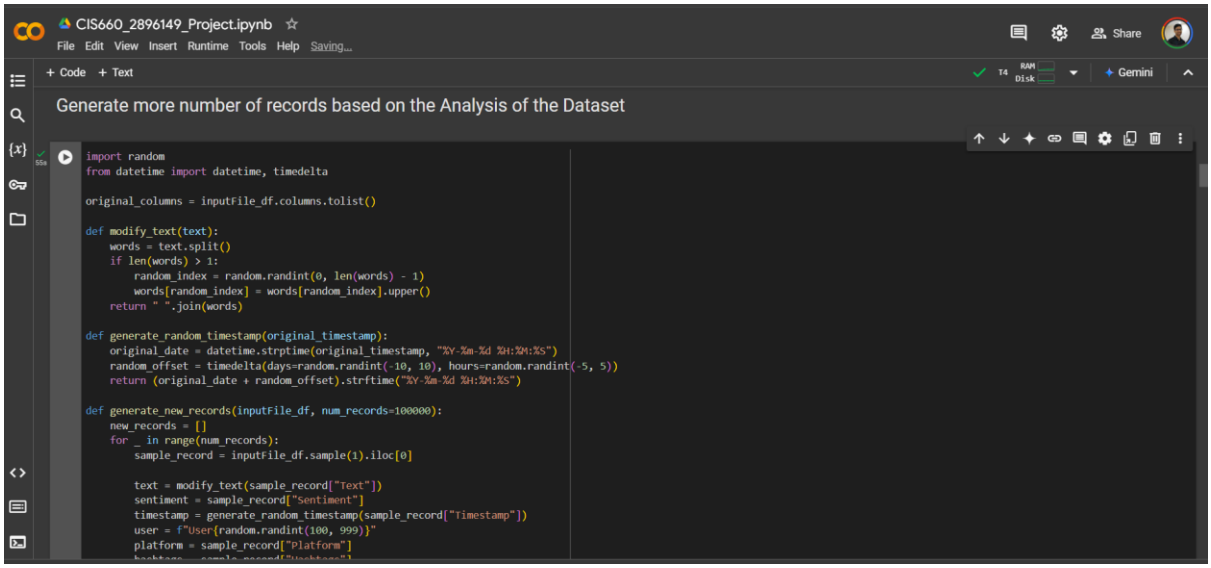


Figure 7: Generating Synthetic Data

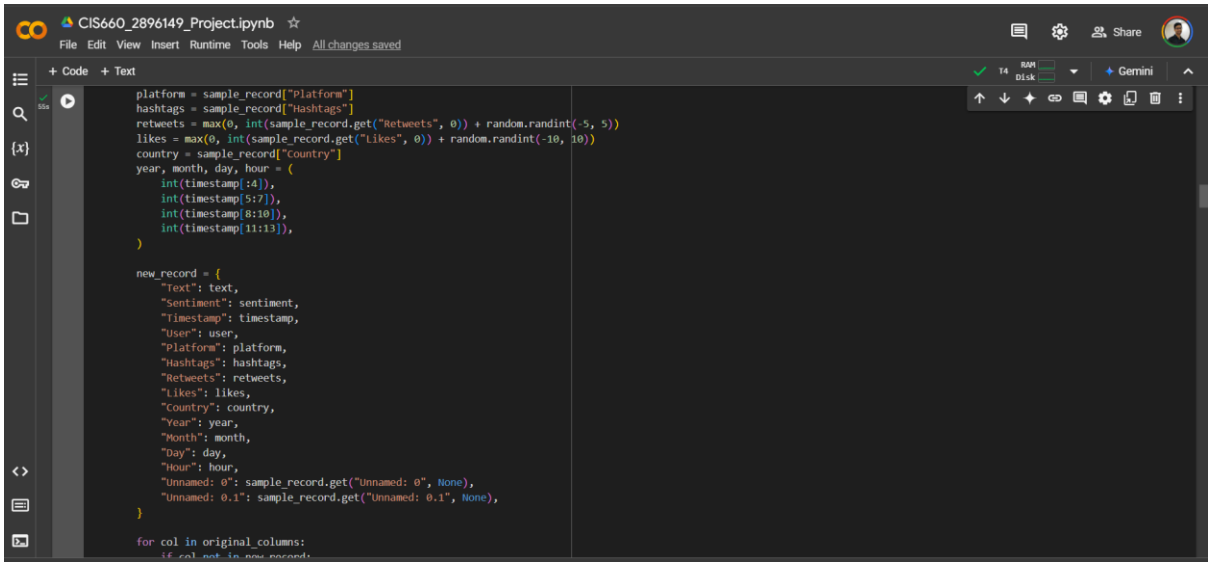
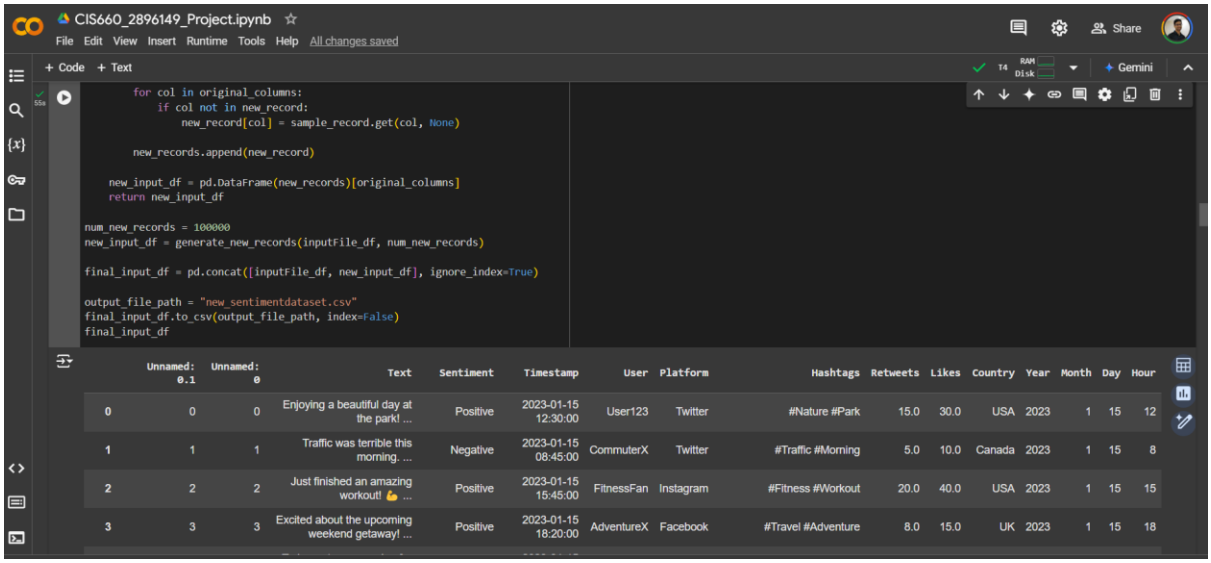
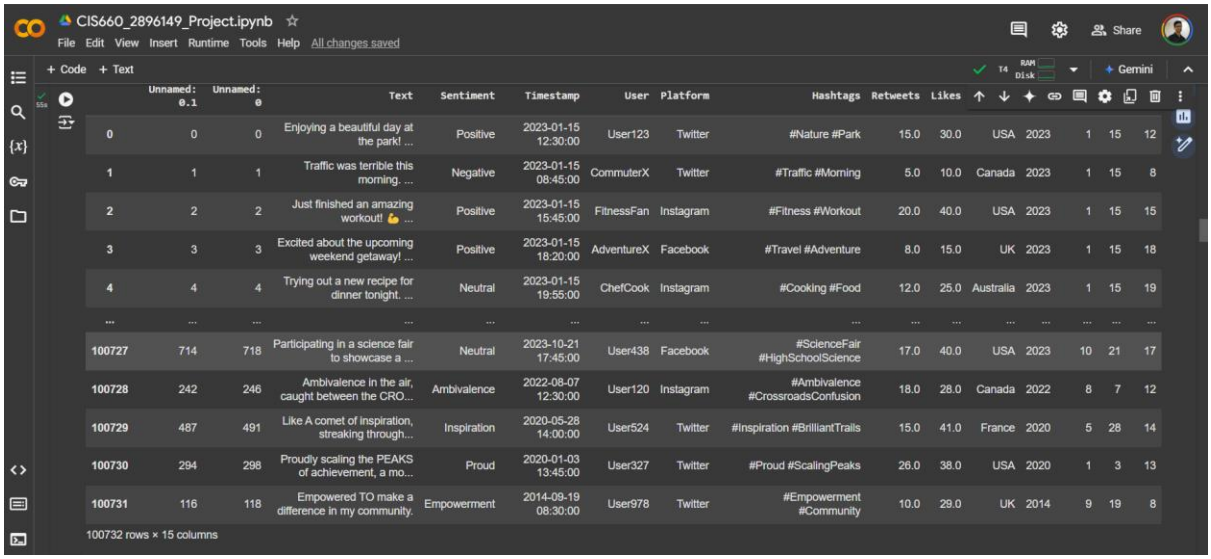


Figure 8: Generating Sythetic Data (contd..)



	Unnamed: 0.1	Unnamed: 0	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	Month	Day	Hour
0	0	0	Enjoying a beautiful day at the park! ...	Positive	2023-01-15 12:30:00	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	1	15	12
1	1	1	Traffic was terrible this morning. ...	Negative	2023-01-15 08:45:00	CommuterX	Twitter	#Traffic #Morning	5.0	10.0	Canada	2023	1	15	8
2	2	2	Just finished an amazing workout! 💪 ...	Positive	2023-01-15 15:45:00	FitnessFan	Instagram	#Fitness #Workout	20.0	40.0	USA	2023	1	15	15
3	3	3	Excited about the upcoming weekend getaway! ...	Positive	2023-01-15 18:20:00	AdventureX	Facebook	#Travel #Adventure	8.0	15.0	UK	2023	1	15	18

Figure 9: Generating Synthetic Data (contd..)



	Unnamed: 0.1	Unnamed: 0	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	Month	Day	Hour
0	0	0	Enjoying a beautiful day at the park! ...	Positive	2023-01-15 12:30:00	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	1	15	12
1	1	1	Traffic was terrible this morning. ...	Negative	2023-01-15 08:45:00	CommuterX	Twitter	#Traffic #Morning	5.0	10.0	Canada	2023	1	15	8
2	2	2	Just finished an amazing workout! 💪 ...	Positive	2023-01-15 15:45:00	FitnessFan	Instagram	#Fitness #Workout	20.0	40.0	USA	2023	1	15	15
3	3	3	Excited about the upcoming weekend getaway! ...	Positive	2023-01-15 18:20:00	AdventureX	Facebook	#Travel #Adventure	8.0	15.0	UK	2023	1	15	18
4	4	4	Trying out a new recipe for dinner tonight. ...	Neutral	2023-01-15 19:55:00	ChefCook	Instagram	#Cooking #Food	12.0	25.0	Australia	2023	1	15	19
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
100727	714	718	Participating in a science fair to showcase a ...	Neutral	2023-10-21 17:45:00	User438	Facebook	#ScienceFair #HighSchoolScience	17.0	40.0	USA	2023	10	21	17
100728	242	246	Ambivalence in the air, caught between the CRO...	Ambivalence	2022-08-07 12:30:00	User120	Instagram	#Ambivalence #CrossroadsConfusion	18.0	28.0	Canada	2022	8	7	12
100729	487	491	Like A comet of inspiration, streaking through...	Inspiration	2020-05-28 14:00:00	User524	Twitter	#Inspiration #BrilliantTrails	15.0	41.0	France	2020	5	28	14
100730	294	298	Proudly scaling the PEAKS of achievement, a mo...	Proud	2020-01-03 13:45:00	User327	Twitter	#Proud #ScalingPeaks	26.0	38.0	USA	2020	1	3	13
100731	116	118	Empowered TO make a difference in my community.	Empowerment	2014-09-19 08:30:00	User978	Twitter	#Empowerment #Community	10.0	29.0	UK	2014	9	19	8

100732 rows × 15 columns

Figure 10: Generating Synthetic Data (contd..)

I performed the below steps to generate the Synthetic Dataset **final_input_df** from the original dataset **input_df:**

- Restore the column names from the original dataset to ensure consistency.
- Generate 100000 Synthetic records by selecting a random row from the original dataset and modifying the fields as per below logic:
  - i. Alter the Text Column by randomly capitalising one word and adding slight variations to simulate real-world differences.
  - ii. Keep the Sentiments, Platform, Hashtags and Country same as the original dataset
  - iii. Generate new timestamp from the original record by applying a random offset.
  - iv. Generate Random User Identifier.

v. Add a small random offset for Retweets and Likes by ensuring non-negative values.
vi. Extract the Year, Month, Day and Hour from the modified timestamp values.

- Create a new Data frame **new_input_df** with the column names as per the original dataset and converting the list of all synthetic records into a data frame.
- Merge the created data frame and the original data frame into a single data frame **final_input_df**.
- Save the final data frame into a csv file **new_sentimentdataset.csv**.

New Dataset Information:

I display the information about the columns/features on the newly generated dataset as below:

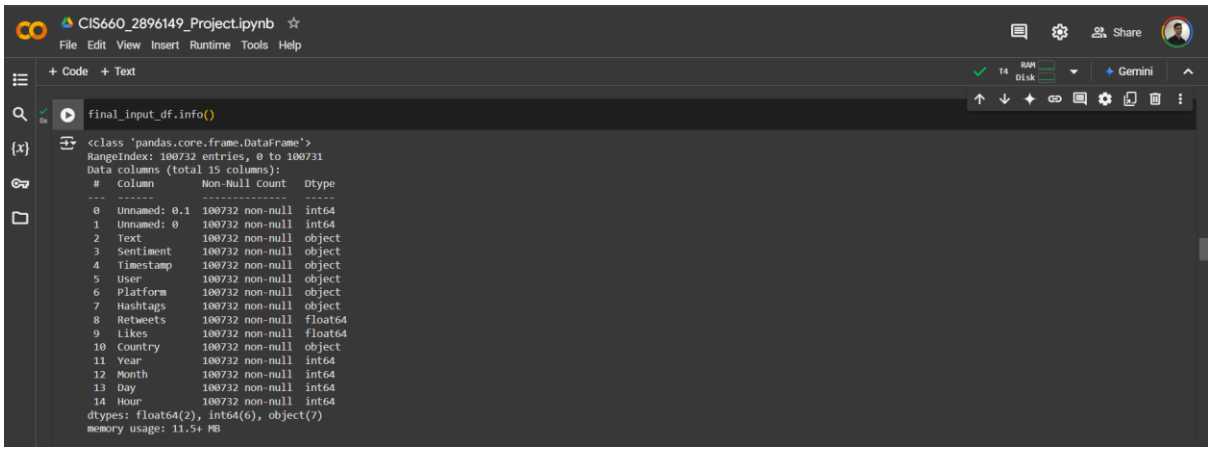


Figure 11: Generated Dataset Information

The new input dataset contains **100732 unique and Non-Null records** which is the combined total of the Original dataset and the newly generated Synthetic dataset. There are 15 columns/features in the new dataset which are same as the original dataset.

Data Cleaning and Preprocessing:

I perform the cleaning and preprocessing of the new input dataset to improve model performance and ensure consistency in the data by implementing the below logic:

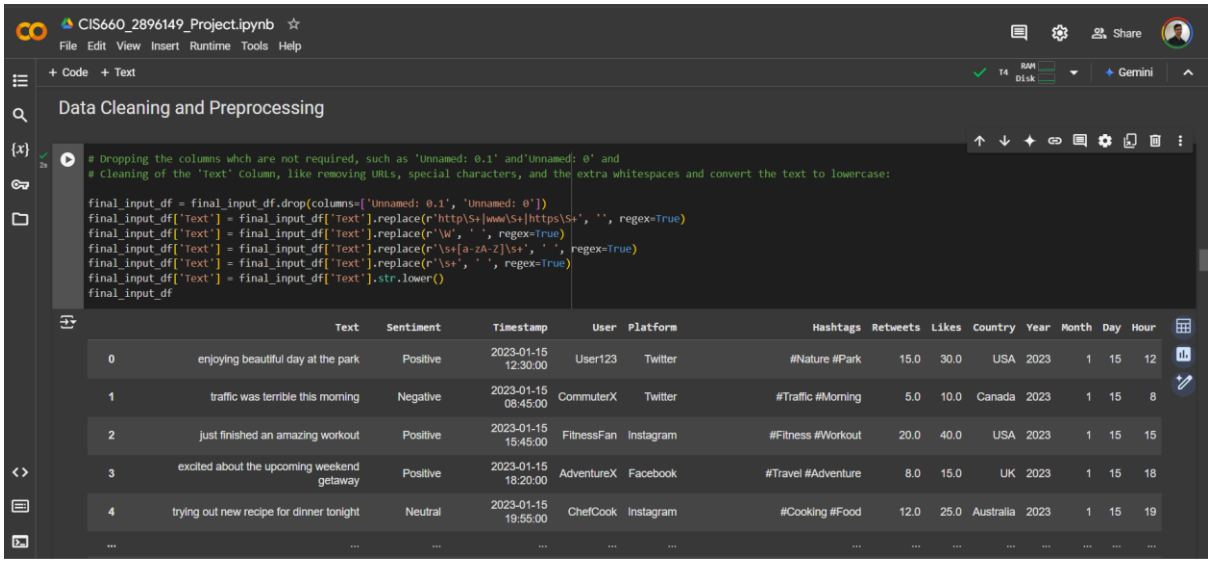
- Drop the columns 'Unnamed: 0.1' and 'Unnamed: 0' from the data frame as these columns don't provide any significance in the Sentiment Analysis of the data.
- Clean the 'Text' column by performing the below steps:
  i. Remove the URLs as they are irrelevant for Sentiment Analysis and add noise to the data by matching the pattern starting with http, https or www by using regular expressions.

ii. Remove the special characters by matching the pattern using regular expressions to ensure the simplification of text processing.

iii. Remove the single characters surrounded by spaces by matching the pattern using regular expressions, to eliminate irrelevant isolated characters from the data.

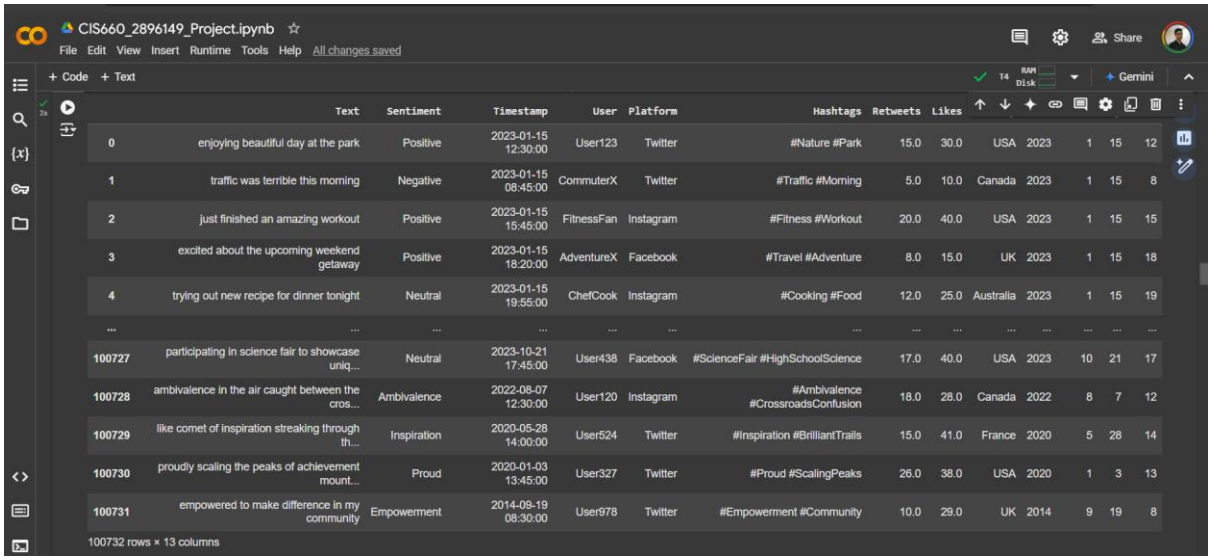
iv. Remove the extra whitespaces by matching the pattern with regular expressions, to maintain a standard spacing as it ensures consistency in the text.

v. Convert the entire text to lowercase to standardize the case text for case-insensitive analysis.



	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	Month	Day	Hour
0	enjoying beautiful day at the park	Positive	2023-01-15 12:30:00	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	1	15	12
1	traffic was terrible this morning	Negative	2023-01-15 08:45:00	CommuterX	Twitter	#Traffic #Morning	5.0	10.0	Canada	2023	1	15	8
2	just finished an amazing workout	Positive	2023-01-15 15:45:00	FitnessFan	Instagram	#Fitness #Workout	20.0	40.0	USA	2023	1	15	15
3	excited about the upcoming weekend getaway	Positive	2023-01-15 18:20:00	AdventureX	Facebook	#Travel #Adventure	8.0	15.0	UK	2023	1	15	18
4	trying out new recipe for dinner tonight	Neutral	2023-01-15 19:55:00	ChefCook	Instagram	#Cooking #Food	12.0	25.0	Australia	2023	1	15	19
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 12: Data Cleaning and Preprocessing



	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	Month	Day	Hour
0	enjoying beautiful day at the park	Positive	2023-01-15 12:30:00	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	1	15	12
1	traffic was terrible this morning	Negative	2023-01-15 08:45:00	CommuterX	Twitter	#Traffic #Morning	5.0	10.0	Canada	2023	1	15	8
2	just finished an amazing workout	Positive	2023-01-15 15:45:00	FitnessFan	Instagram	#Fitness #Workout	20.0	40.0	USA	2023	1	15	15
3	excited about the upcoming weekend getaway	Positive	2023-01-15 18:20:00	AdventureX	Facebook	#Travel #Adventure	8.0	15.0	UK	2023	1	15	18
4	trying out new recipe for dinner tonight	Neutral	2023-01-15 19:55:00	ChefCook	Instagram	#Cooking #Food	12.0	25.0	Australia	2023	1	15	19
...	...	...	...	...	...	...	...	...	...	...	...	...	...
100727	participating in science fair to showcase uniq...	Neutral	2023-10-21 17:45:00	User438	Facebook	#ScienceFair #HighSchoolScience	17.0	40.0	USA	2023	10	21	17
100728	ambivalence in the air caught between the cros...	Ambivalence	2022-08-07 12:30:00	User120	Instagram	#Ambivalence #CrossroadsConfusion	18.0	28.0	Canada	2022	8	7	12
100729	like comet of inspiration streaking through th...	Inspiration	2020-05-28 14:00:00	User524	Twitter	#Inspiration #BrilliantTrails	15.0	41.0	France	2020	5	28	14
100730	proudly scaling the peaks of achievement mount...	Proud	2020-01-03 13:45:00	User327	Twitter	#Proud #ScalingPeaks	26.0	38.0	USA	2020	1	3	13
100731	empowered to make difference in my community	Empowerment	2014-09-19 08:30:00	User978	Twitter	#Empowerment #Community	10.0	29.0	UK	2014	9	19	8

100732 rows × 13 columns

Figure 13: Data Cleaning and Preprocessing (contd..)

Feature Engineering on the Dataset:

I perform some advanced feature engineering on the data frame by generating new features based on the analysis of the timestamps, user activity, hashtags, engagement metrics and country specific data.

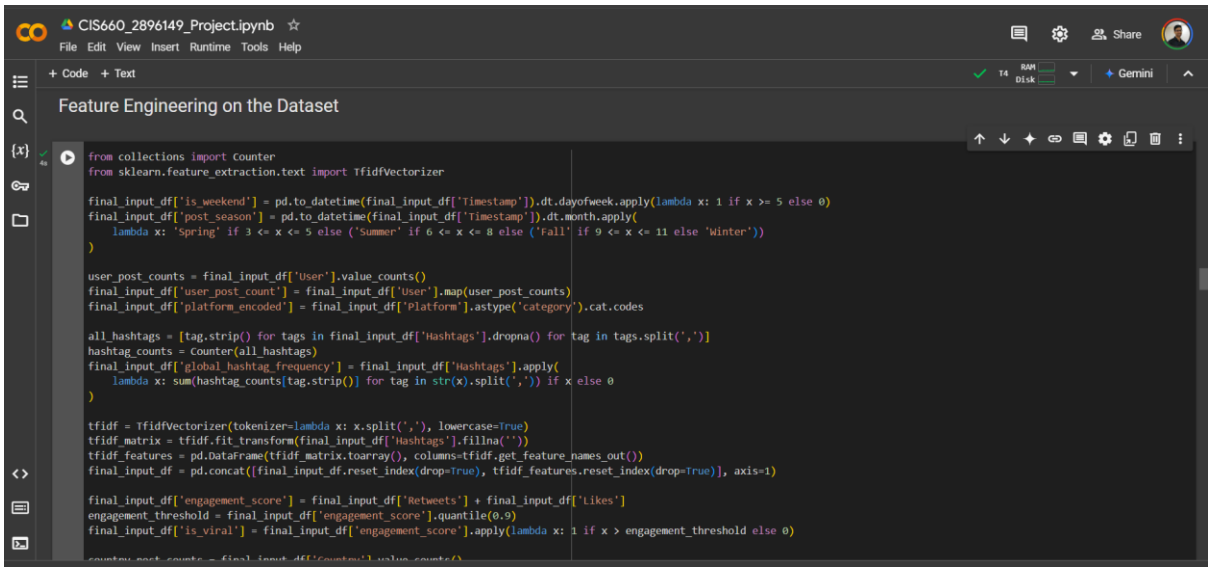


Figure 14: Feature Engineering on the Dataset

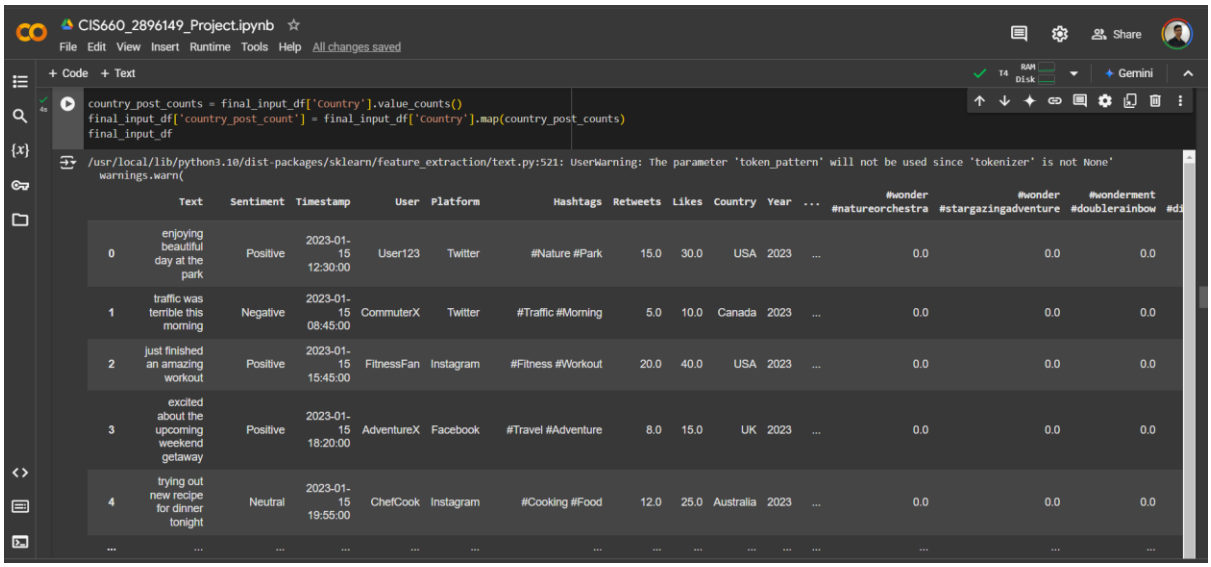


Figure 15: Feature Engineering on the Dataset (contd..)

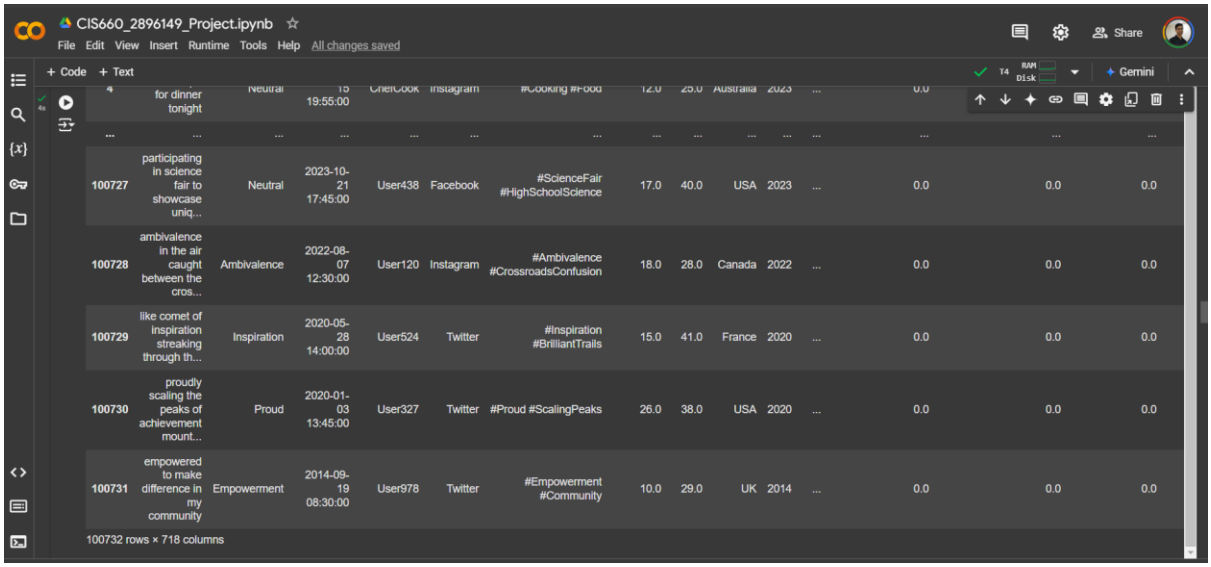


Figure 16: Feature Engineering on the Dataset (contd..)

The detailed logic is explained below:

- **Timestamp-based Features:**
  - i. *is_weekend:* Convert the timestamp into datetime format and extract the day of the week, and then check if the post was made on the weekend (Saturday or Sunday)
  - ii. *post_season:* Extract the Month from the timestamp and map it to a season (e.g., March to May = Spring).
- **User-based Features:**
  - i. *user_post_count:* Count the number of posts made by each user and map this count to the corresponding user in the dataset.
- **Platform-based Features:**
  - i. *platform_encoded:* Encode the Platform column into numeric codes suitable for machine learning algorithms.
- **Hashtag-based Features:**
  - i. *global_hashtag_frequency:* Split the hashtags into individual tags and count their frequencies in each row to measure the popularity of the Hashags.
  - *ii. tf-idf features for hashtags:* Convert the hashtags into TF-IDF vectors and append the resulting features into the dataset as additional columns, to measure the significance of each hashtag in the dataset.
- **Engagement-based Features:**
  - i. *engagement_score:* Add the number of Retweets and Likes for each post to measure the overall engagement level of that post.
  - ii. *is_viral:* Mark the post with engagement_score of greater than 90th percentile as viral to identify the highly engaged posts.
- **Country-based Features:**

i. *country_post_count:* Count the number of posts from each country and map it to the respective country in the dataset to provide insights into the geographic trends of the sentiments.

This data frame now includes all the enriched features which can be used for predictive modelling, sentiment analysis or exploratory analysis.

Check for Class Imbalance:

I check the **distribution** of all the **unique Sentiment classes** in the Input dataset by printing the count of each class. I perform this step to showcase how the various Sentiment Classes are distributed over the dataset.

Figure 17: Check for Class Imbalance in the input dataset

There are **279** different Sentiment Classes distributed over the dataset which need to be mapped into 3 main classes: **Positive**, **Negative** and **Neutral**.

Encoding the Sentiments and Map the Sentiments to Positive, Negative and Neutral:

First, I clean the values in the Sentiment Column by clearing the leading / trailing whitespace in that, e.g., 'Positive ' becomes 'Positive', to ensure standard formatting and then get the distinct Sentiment values after cleaning.

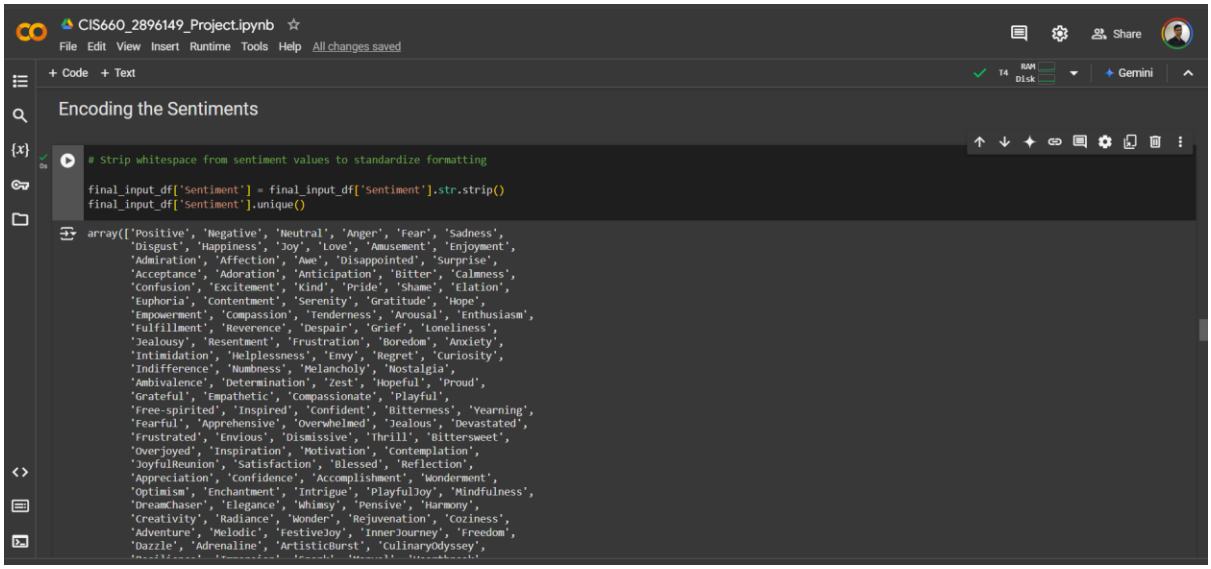


Figure 18: Cleaning the Sentiment Values

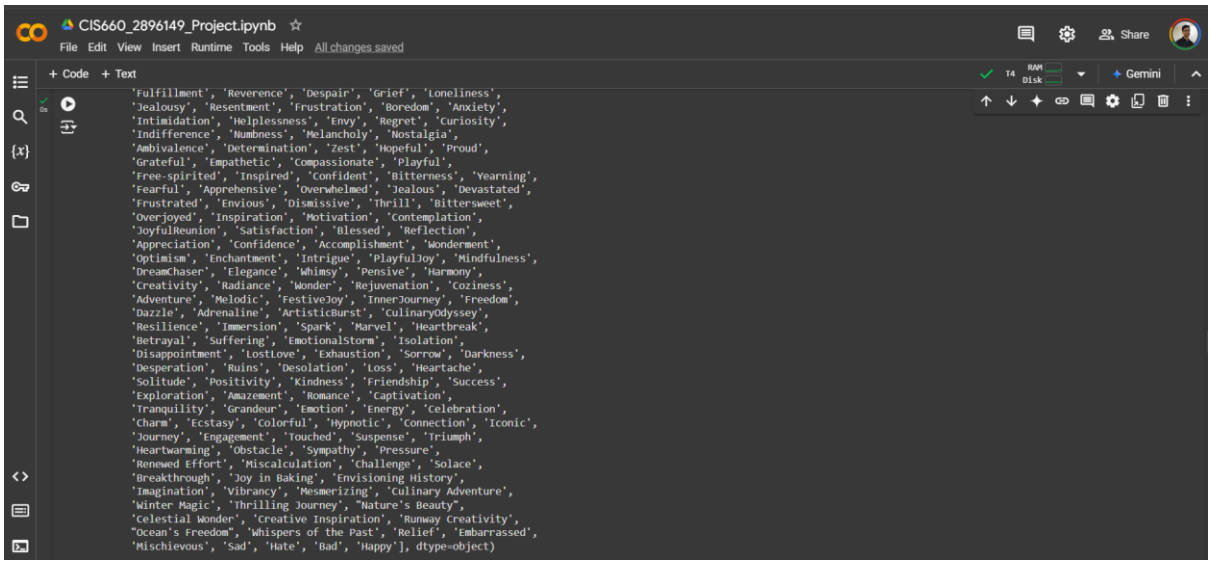


Figure 19: Cleaning the Sentiment Values (contd..)

Now, I map all the above distinct Sentiment Classes to 3 major Classes: **Positive (1)**, **Negative (0)** and **Neutral (2)** to achieve the below:

- Simplification of Sentiment Categories
- Consistency in Labels
- Easier to Interpret
- Focus on High Level Trends
- Model Simplicity and Performance
- Stability across Applications

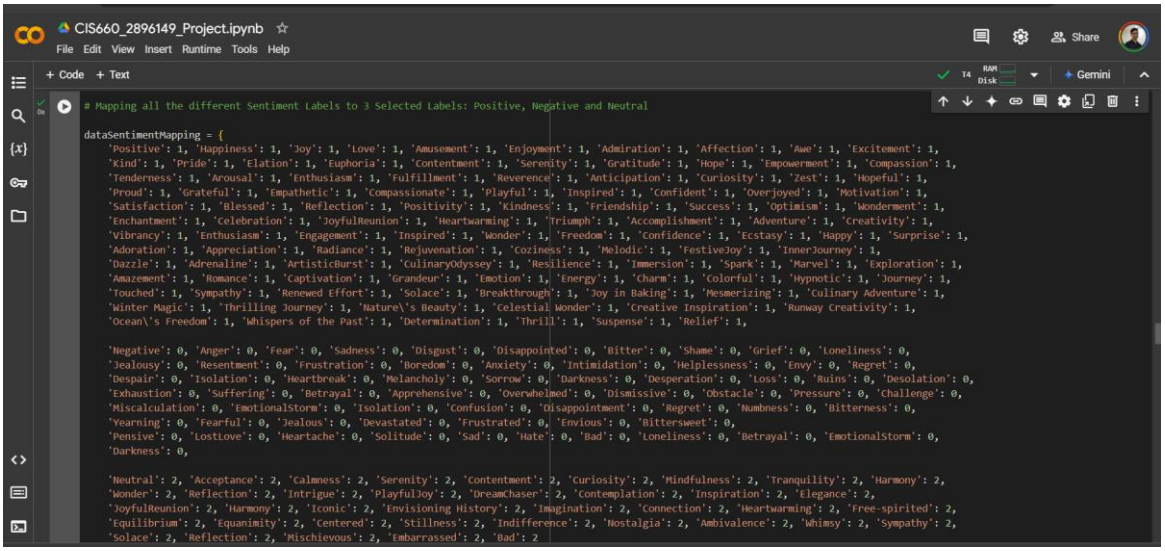


Figure 20: Mapping the Sentiment Classes

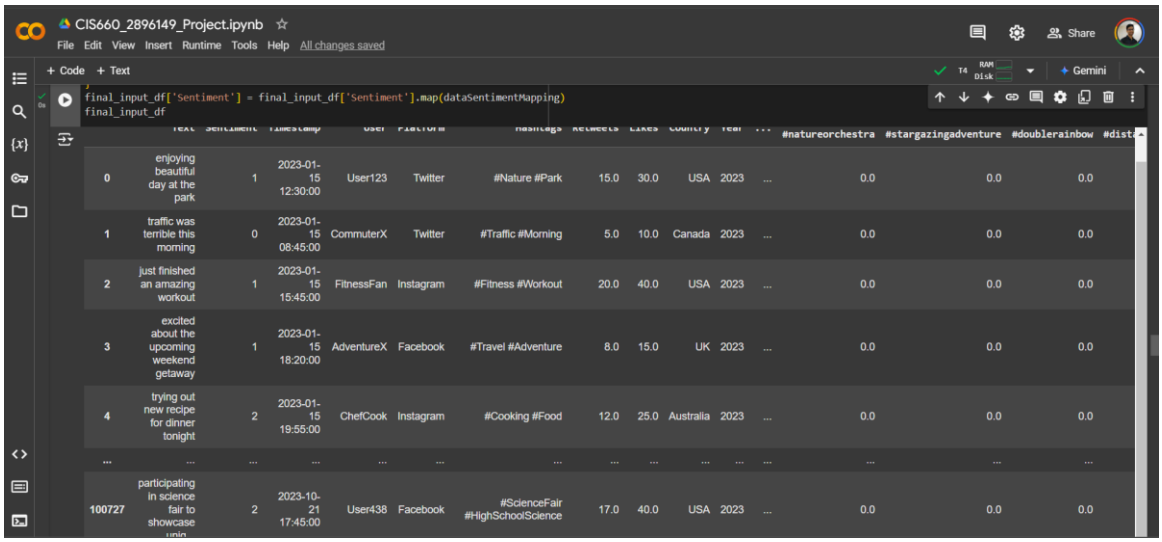


Figure 21: Mapping the Sentiment Classes (contd..)

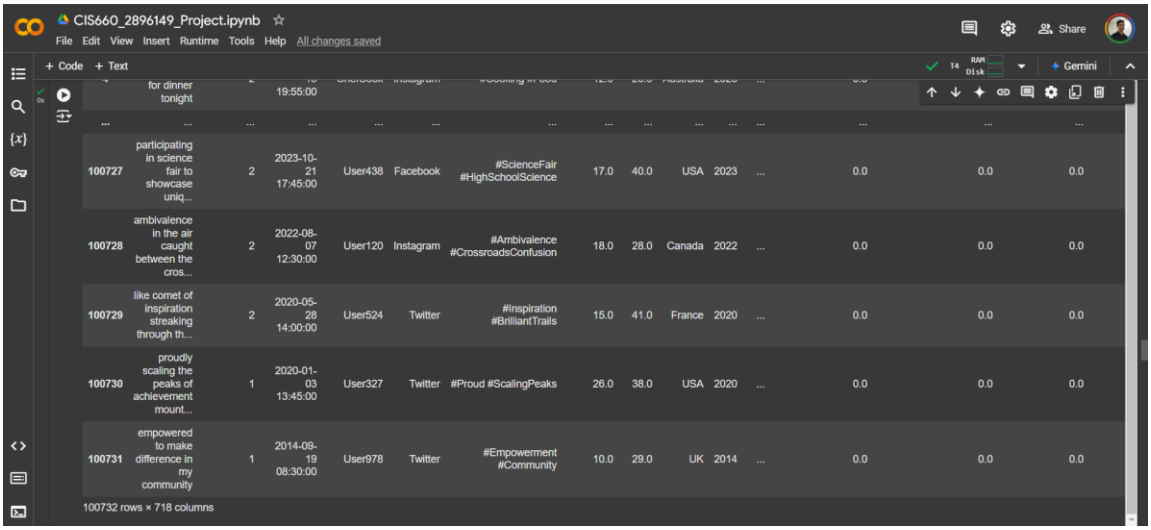


Figure 22: Mapping the Sentiment Classes (contd..)

Split the Input Data frame into Training Dataset, Validation Dataset and Test Dataset:

Now, I split the entire Input Dataset into 3 subsets: **Training Dataset** (containing **70%** of the data), **Validation Dataset** (containing **15%** of the data) and **Test Dataset** (containing **15%** of the data) to

Figure 23: Split the Dataset into Training, Validation and Test Datasets

Perform the Classification using a Base Model (XGBoost):

I have used **XGBoost** algorithm as a **Base model** to perform the **Sentiment Classification** on the Input Dataset.

XGBoost (eXtreme Gradient Boosting) is a powerful and scalable machine learning algorithm mostly used for structured/tabular data. It is based on **gradient boosting** and uses decision trees as base learners. XGBoost optimizes for both speed and performance by implementing advanced techniques like parallel computing, tree pruning, and regularization.

Below is the high-level flow of how the XGBoost algorithm works.

- STEP 1: First, we initialize the predictions, which is generally assigning uniform probabilities for all classes.
- STEP 2: We calculate the gradients of the loss functions for each of the record with respect to the predictions. These gradients are the errors, which the model needs to fix.
- STEP 3: We train Decision Tree to predict these errors.
- STEP 4: We adjust the Predictions by adding the outputs of the new tree weighted by a learning rate.

$$\mathbf{F_m(x) = F_{m-1}(x) + \eta \cdot T_m(x)}$$

Where,

- $F_m(x)$: New prediction after m^{th} iteration.
- $F_{m-1}(x)$: Prediction after $(m-1)^{th}$ iteration.
- $T_m(x)$: Output of the m^{th} tree.
- η: Learning rate.

- STEP 5: We repeat performing Step 2 to Step 4 either for a predefined number of trees or until we achieve convergence.
- STEP 6: We reduce overfitting and improve generalization in XGBoost by including L1 and L2 Regularization.

Objective Function:

$$\mathcal{L}(F) = \sum_{i=1}^{n} l(y_i, F(x_i)) + \sum_{k=1}^{m} \Omega(T_k)$$

Where:

- $l(y_i, F(x_i))$: Loss function (e.g., log-loss for classification)
- $\Omega(T_k) = \gamma T + \frac{\lambda}{2}||w||^2$: Regularization term, where T is the number of leaves and w is the leaf weights.

Leaf Weight:

- For a given leaf, the weight w is computed as:

$$w = -\frac{\sum g_i}{\sum h_i + \lambda}$$

Where:

- g_i: Gradient (first derivative of the loss)
- h_i: Hessian (second derivative of the loss)

Gain:

- The gain for splitting a node is calculated as:

$$\text{Gain} = \frac{1}{2}\left[\frac{\left(\sum_{i\in L} g_i\right)^2}{\sum_{i\in L} h_i + \lambda} + \frac{\left(\sum_{i\in R} g_i\right)^2}{\sum_{i\in R} h_i + \lambda} - \frac{\left(\sum_{i\in L\cup R} g_i\right)^2}{\sum_{i\in L\cup R} h_i + \lambda}\right] - \gamma$$

Below are the steps which I performed for XGBoost Model:

- Import the Required Libraries **LabelEncoder, XGBClassifier, classification_report, accuracy_score, hstack** and **csr_matrix**.
- Encode the categorical features **Platform** and **Country** into numerical fields.
- Transform the text data in the **Text** column into numerical vectors using **TF-IDF**.

- Drop the columns which are not required by the **XGBoost Model**.
- Combine the TF-IDF features and the structured numerical features into a single sparse matrix.
- Define and train the XGBoost Classifier with the below parameters:
  n_estimators=300,
  max_depth=5,
  learning_rate=0.05,
  subsample=0.8,
  colsample_bytree=0.8,
  tree_method='hist',
  random_state=42
- Then Evaluate the Model performance on the Validation Dataset and display the metrics: Accuracy, Precision, Recall and F1-score.
- Plot the Confusion Matrix Heatmap to visualize the predictions of the model on the Validation Dataset.

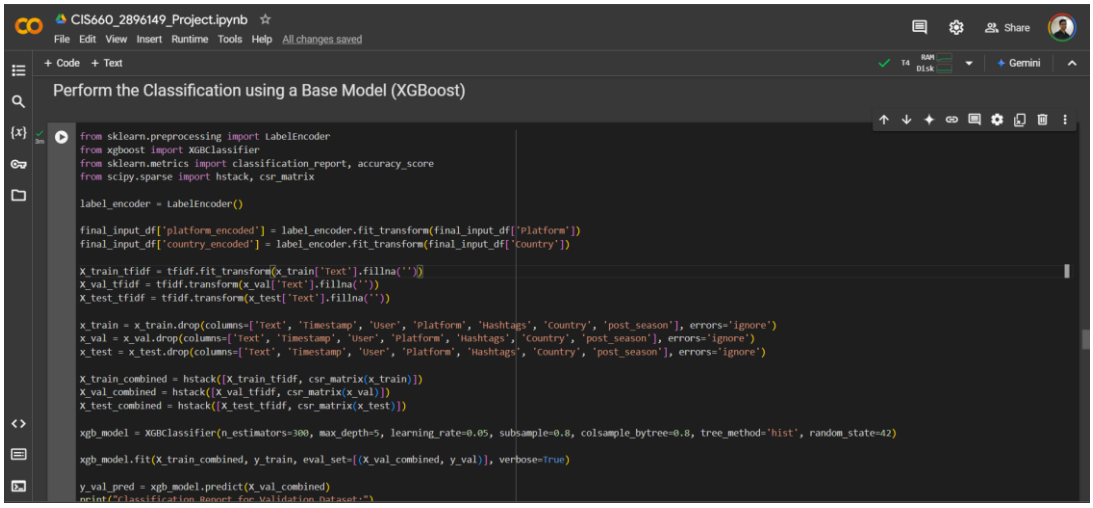


Figure 24: XGBoost Model

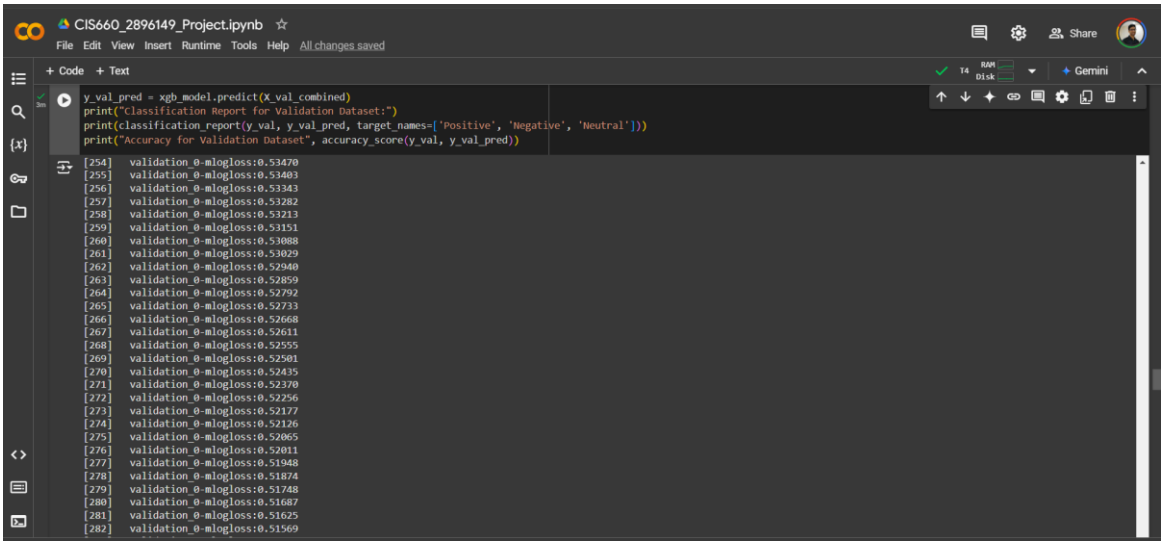


Figure 25: XGBoost Model (contd..)

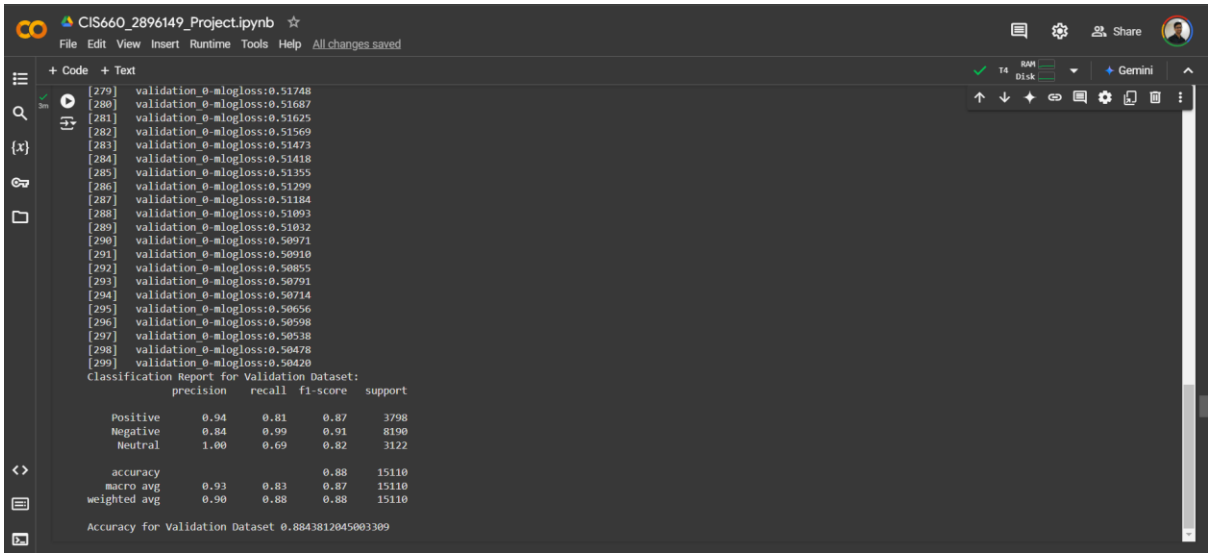


Figure 26: XGBoost Model (contd..)

I got an Accuracy of **0.884 (88.4%)** for the Validation Dataset for the XGBoost Model.

Figure 27:Confusion Matrix for XGBoost

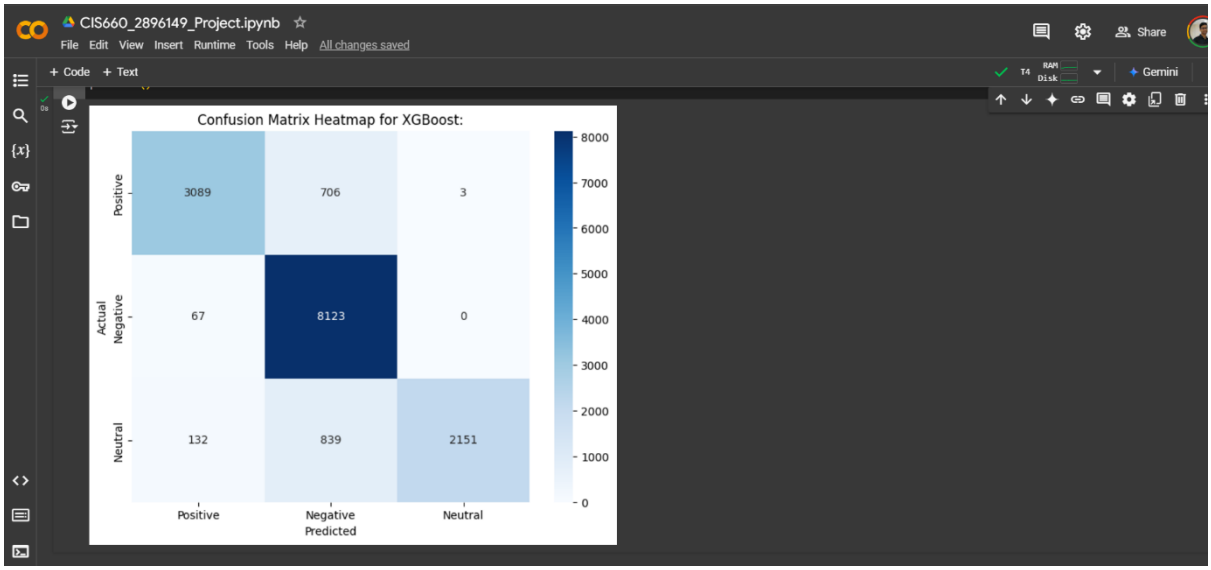


Figure 28: Confusion Matrix for XGBoost (contd..)

Hyperparameter Optimisation for the Validation Dataset:

I perform the Hyperparameter Optimisation for the XGBoost model using **GridSearchCV** to improve the model performance and efficiency.

The model is trained multiple times for the below parameter combinations and the best parameter combination, for which the Maximum Accuracy is achieved, is displayed.

'n_estimators': [100, 200, 300],

'max_depth': [4, 5, 6],

'learning_rate': [0.01, 0.05, 0.1],

'subsample': [0.8, 1.0],

'colsample_bytree': [0.8, 1.0]

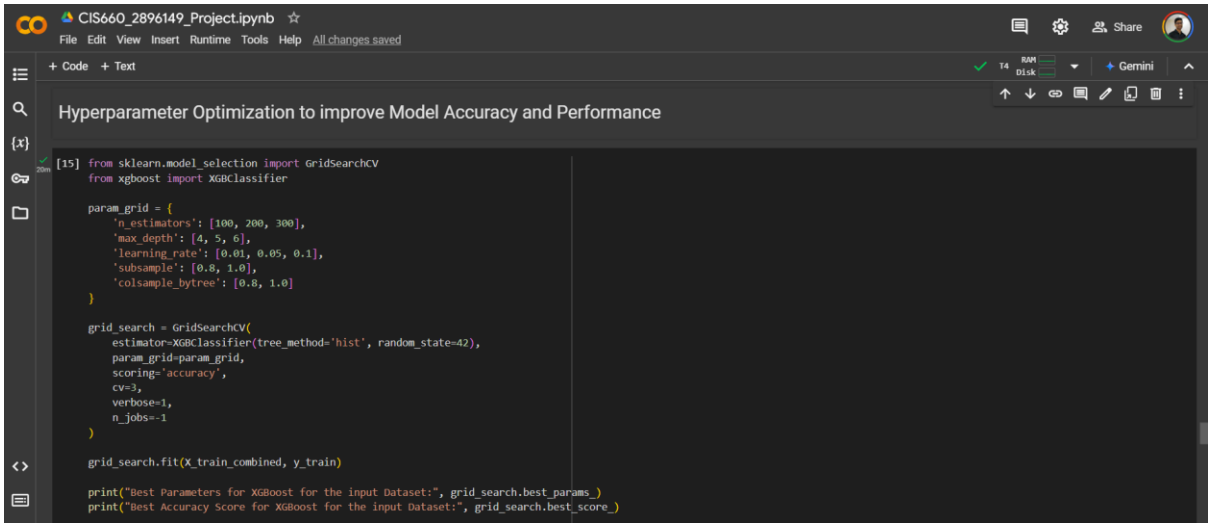


Figure 29: Hyperparameter Optimisation for XGBoost

Figure 30: Hyperparameter Optimisation for XGBoost (contd..)

I got the Maximum Accuracy of **0.99 (99%)** for the below parameters:

{'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 300, 'subsample': 1.0}

Transformer-Based Large Language Model (LLM): BERT:

Now, I perform the Sentiment Classification using a Transformer-Based Large Language Model (**LLM**), which is **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformer).

BERT (Bidirectional Encoder Representations from Transformers) is an open-source deep learning framework developed by Google researchers for natural language processing (NLP) tasks. It uses a bidirectional Transformer encoder to represent words in a sentence contextually as a sequence of vectors, enabling it to better understand the meaning of words in a sentence. This helps improve the accuracy of NLP tasks such as sentiment analysis, question-answering, and text classification.

There are 2 main steps in using the BERT model for our applications:

- **Pretraining** through Predicting Sequence-Based Randomly Masked Tokens:
  BERT Large/Base Models have already been completed for the general public.
- **Fine-tuning** training using specially labeled training data for a particular application:
  Including a task-specific output layer and using task-specific data for training.

BERT architecture is mainly based on the Encoder representation from the Transformers.

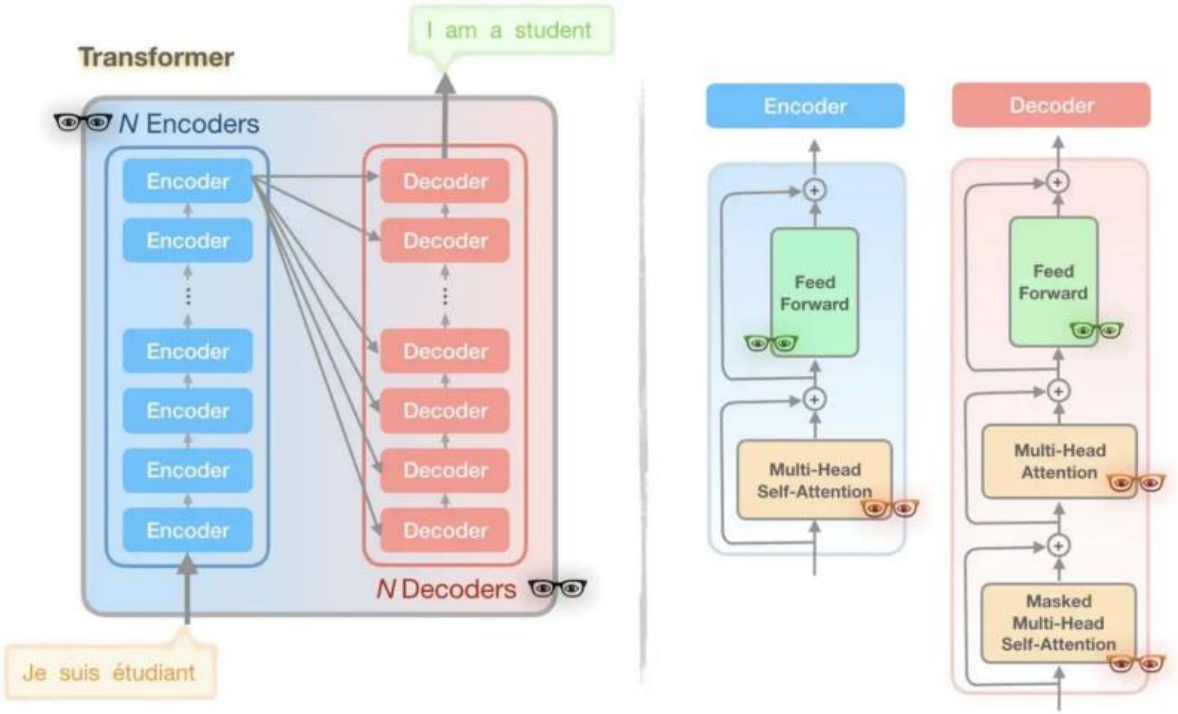


Figure 31: Transformer-based Architecture

Below are the key features of the Transformer in the BERT Model:

- **Bidirectional Attention:** BERT's encoder attends to both previous and future words simultaneously (e.g., understanding "bank" as a riverbank or financial institution depending on both left and right contexts).
- **Self-Supervised Pretraining:**Tasks like Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) train BERT to understand relationships in text.

We can summarize the overall BERT architecture from the diagram as below:

i. **Encoder:** Processes and learns contextual embeddings for inputs (used in BERT).
ii. **Decoder:** Generates outputs based on encoder embeddings (not used in BERT).
iii. **Attention Mechanism:** Allows learning dependencies and relationships within input tokens (self-attention) and between input-output tokens (encoder-decoder attention).
iv. **BERT Focus:** Uses only the encoder stack of the transformer with bidirectional attention, enabling a robust understanding of textual data.

There are 2 models of BERT which are implemented based on the Hyperparameters:

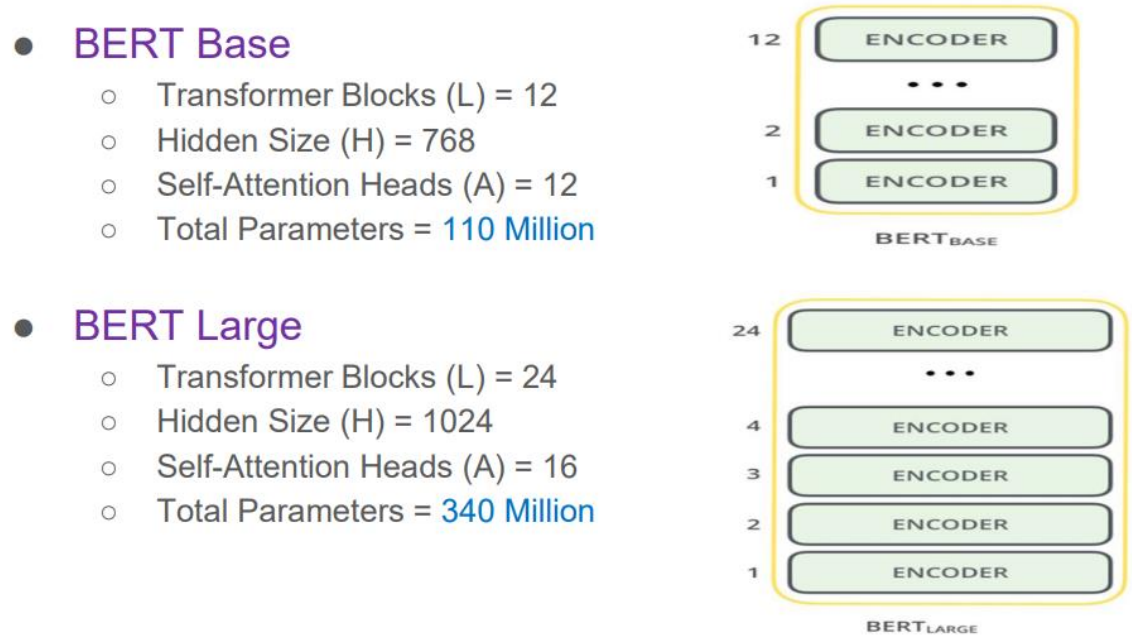


Figure 32: BERT Models

I have used the **BERT Base** Model because that is sufficient for training my Dataset.

Below is the code implementation of how I trained and evaluated the BERT Model:

First, I installed the pre-requisite libraries for BERT, which are **datasets**, **transformers** and **wandb**.

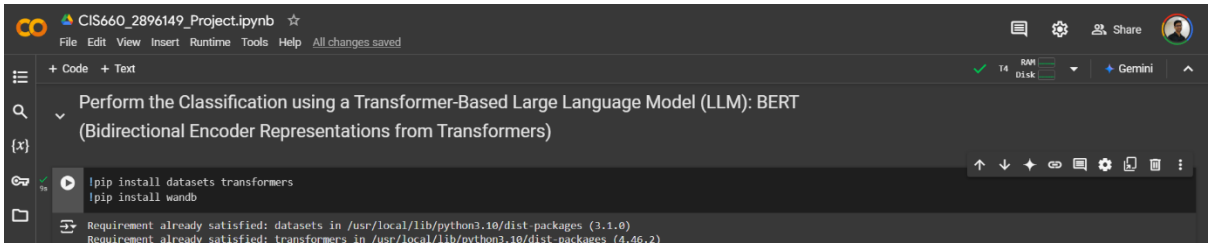


Figure 33: Installing the Pre-requisites for BERT

Then, I train the BERT model using the Training Dataset and evaluate on the Validation Dataset as below:

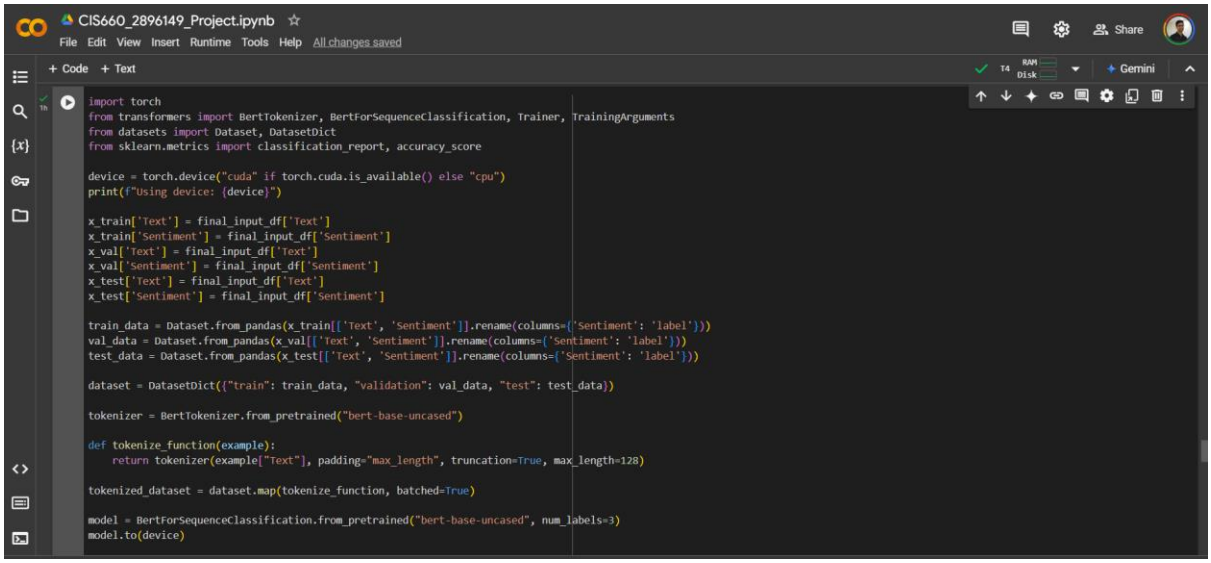


Figure 34: Training the BERT model

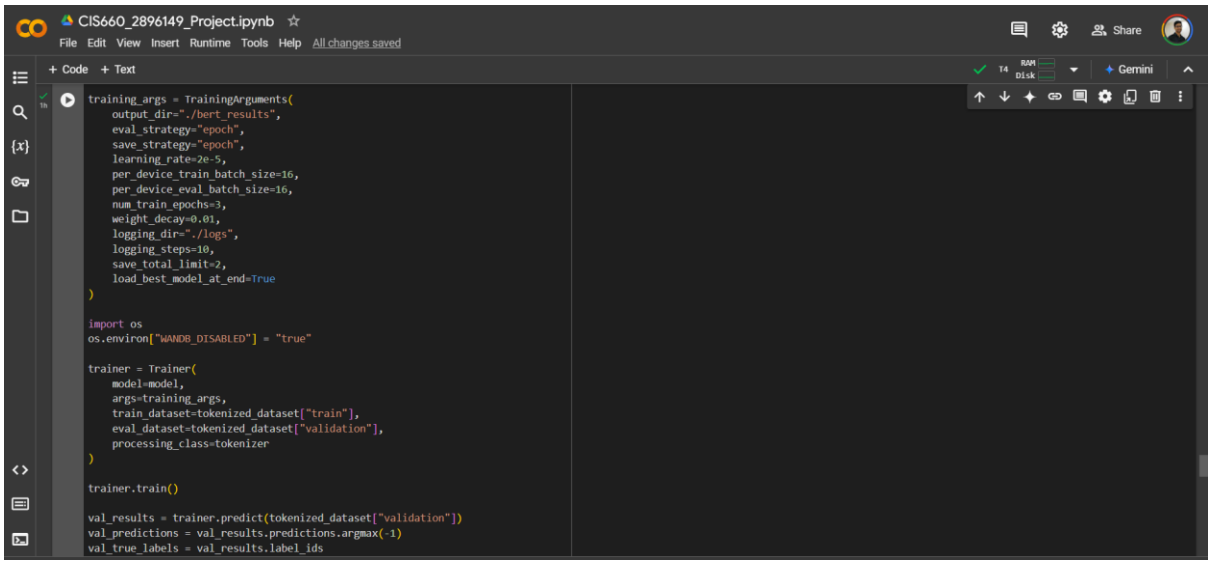


Figure 35: Traning the BERT Model (contd..)

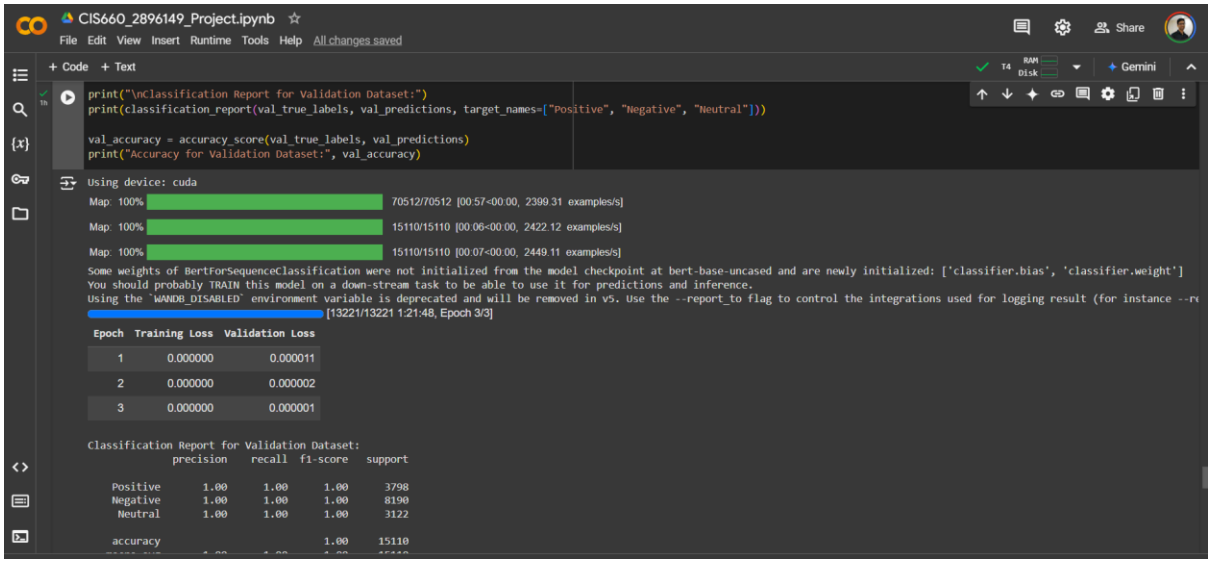


Figure 36: Training the BERT model (contd..)

Finally, I display the Model performance metrics for the Validation Dataset as below:

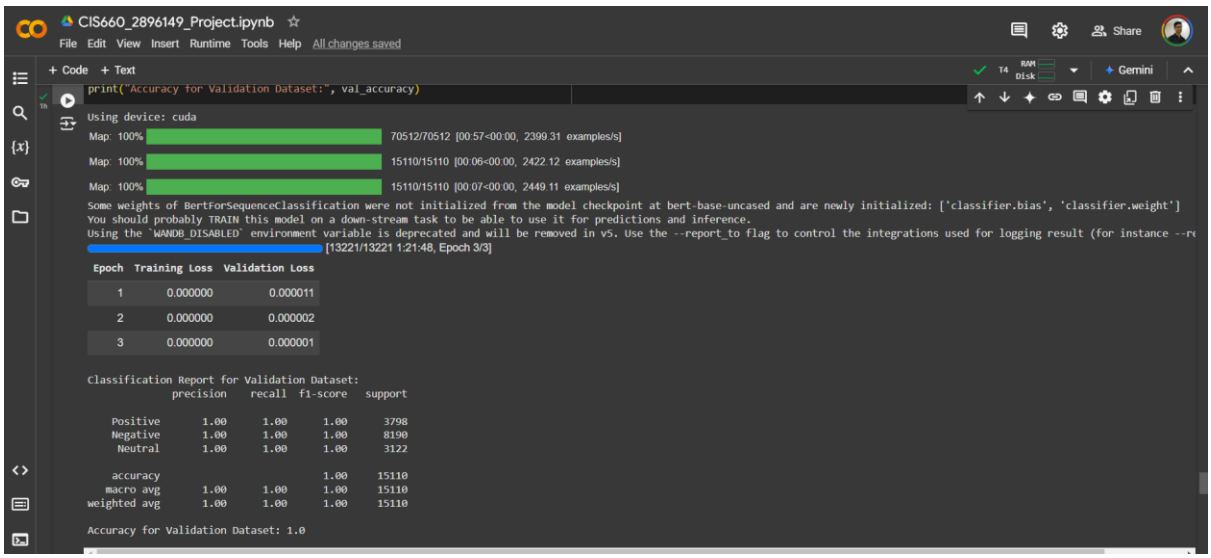


Figure 37: Performance Metrics for BERT Model

I got **100%** accuracy for the validation Dataset using BERT model.

We can derive the below observations from the above metrics:

- BERT model performs the classification perfectly without any noise and deviations and **correctly predicts the Sentiments for the Validation Dataset**.
- The model may exhibit **Overfitting** issue as it generated 100% accuracy.

To cross check my model performance, I need some other performance evaluation techniques like Hyperparameter tuning (training the model multiple times with different parameters) or K-fold cross validation (evaluating the model accuracy for k subsets of data and then calculating the average metrics to find the overall model metrics.

I have used k-fold cross validation to cross check my BERT model performance.

k-Fold cross validation on the Validation Dataset:

I have performed the k-fold cross validation with k=5 to evaluate the BERT model's performance on the Validation Dataset as below:

- Training the BERT model for k=5 folds of data:

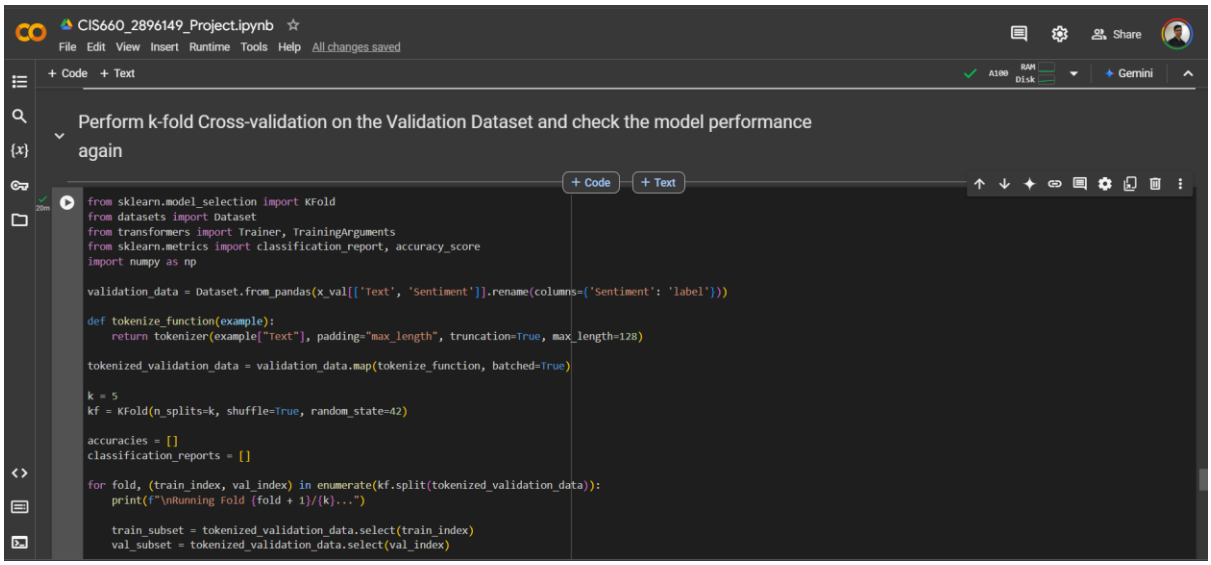


Figure 38: k-fold cross validation on BERT Model

Figure 39:k-fold cross validation on BERT Model (contd..)

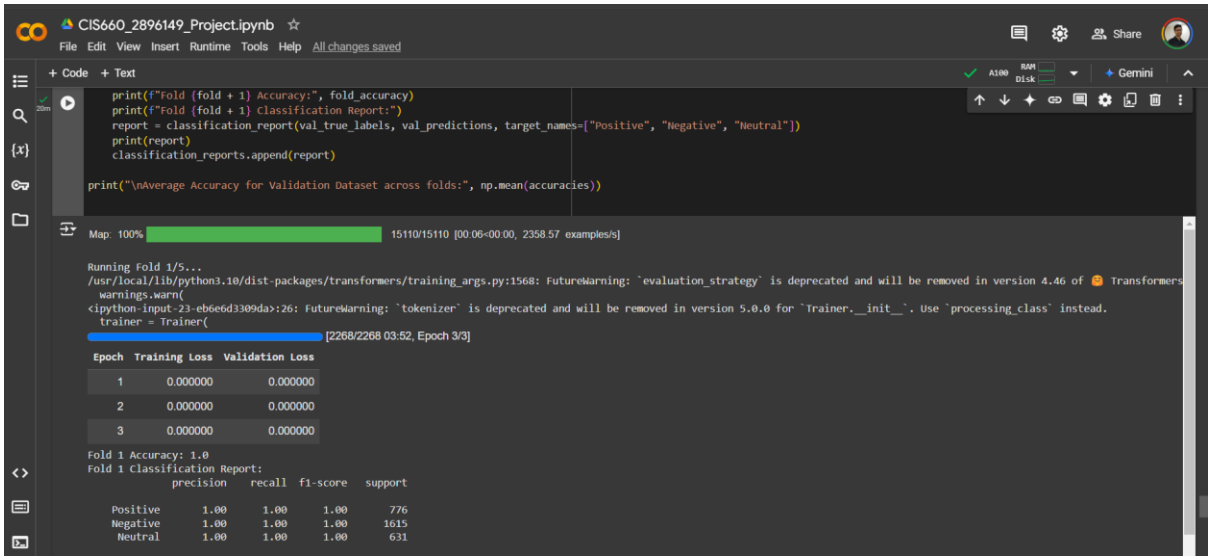


Figure 40: k-fold cross validation on BERT Model (contd..)

- Display the accuracy for each fold and calculate the Average accuracy for the Model:

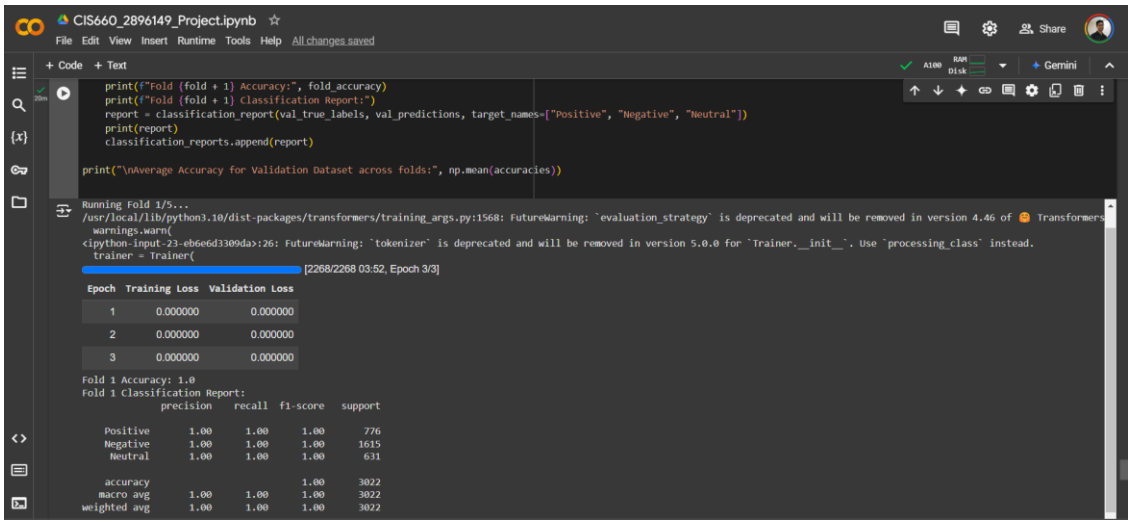


Figure 41: Accuracy for k=1

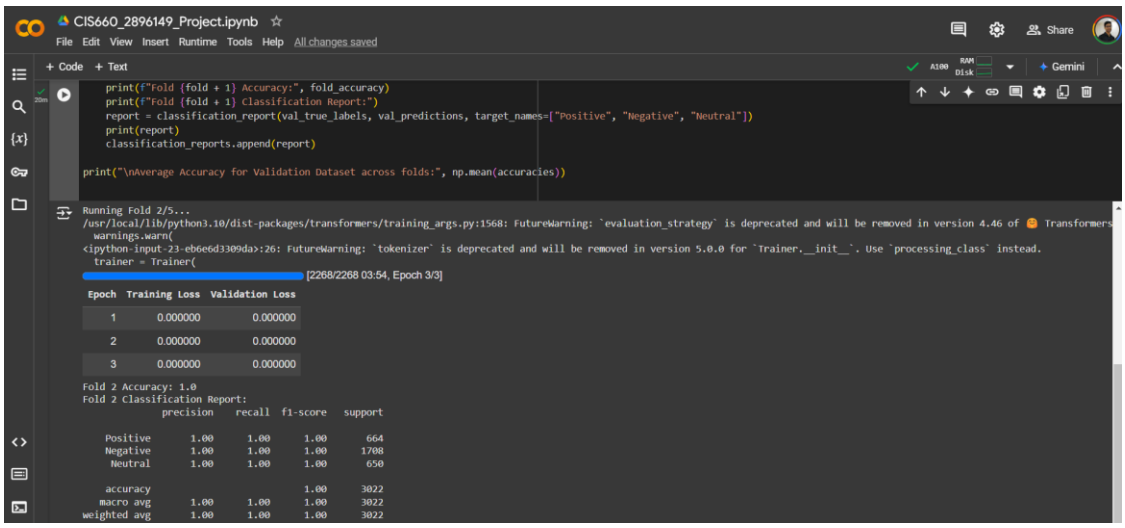


Figure 42: Accuracy for k=2

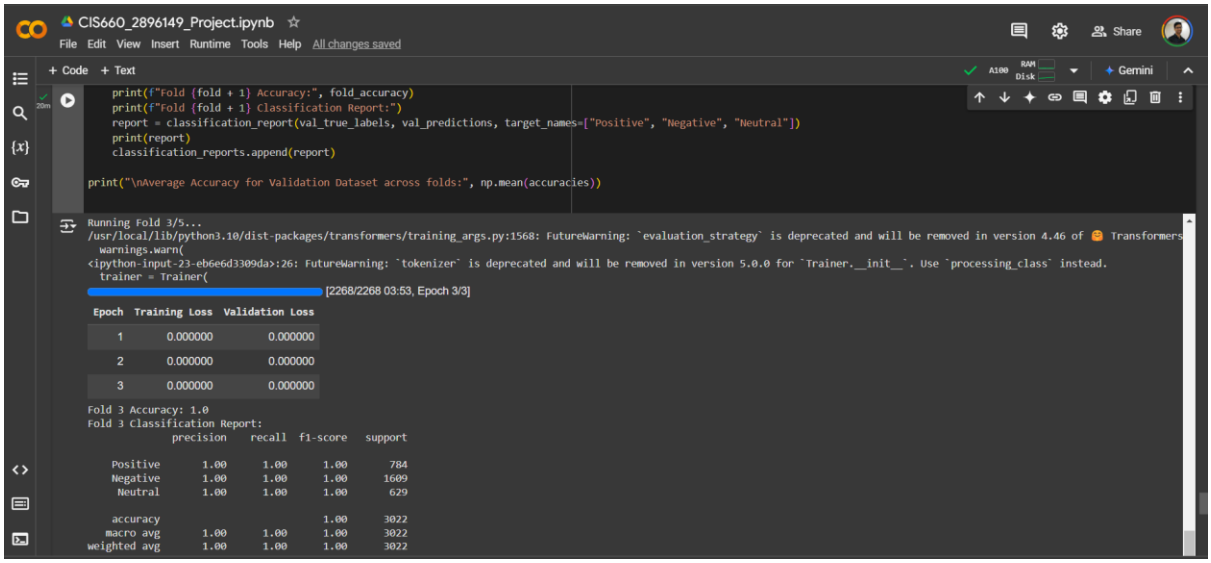


Figure 43: Accuracy for k=3

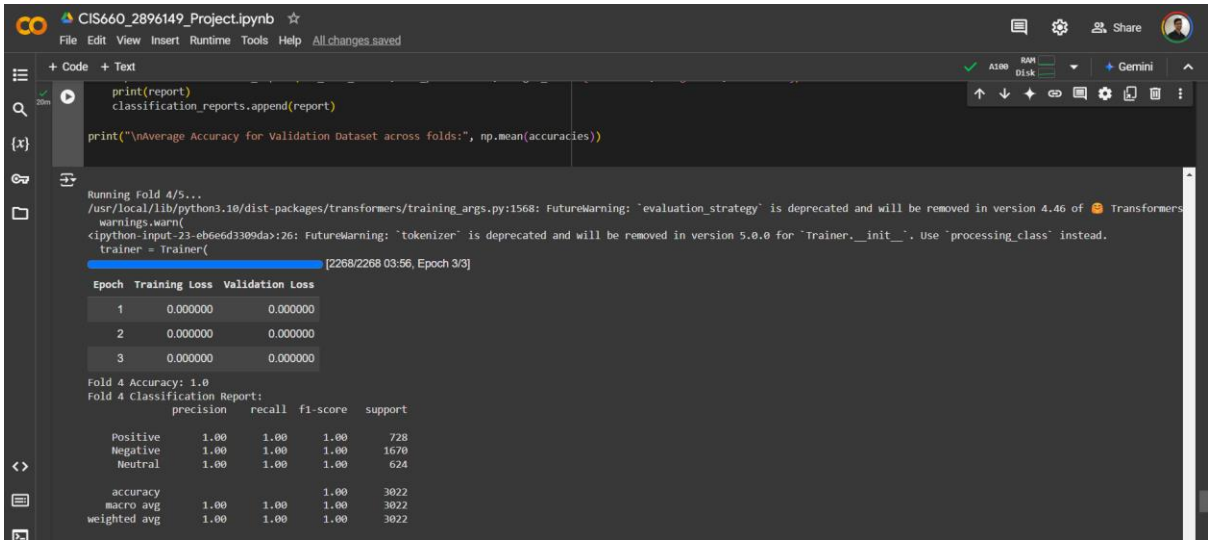


Figure 44: Accuracy for k=4

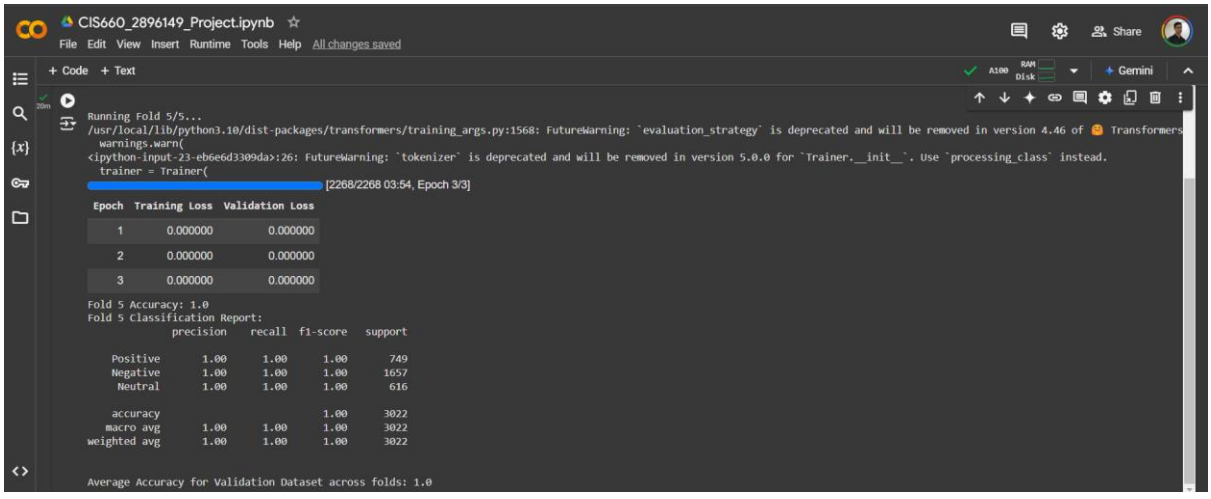


Figure 45: Accuracy for k=5 and Average Accuracy of the Model

I found that the Accuracy of all the 5 folds is **100%** so the Average Accuracy of the Model after performing k-fold cross validation is **100%**.

So, I finally evaluate the model on the unseen data.

Evaluate the Model Performance on Test Dataset:

I finally evaluate the Model performance on the Test Dataset, which is unseen data for the model.

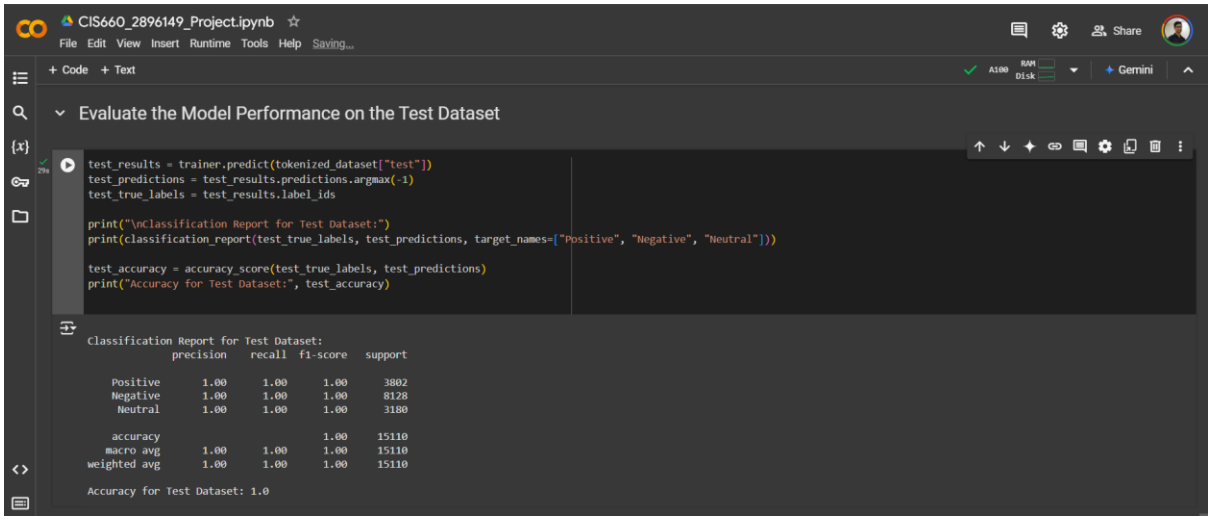


Figure 46: BERT Model performance on Test Dataset

I found that we got a **100% accuracy** for the BERT model on **unseen data Test Dataset**.

Visualising the Sentiment Analysis performed by BERT on a UI Application:

I have designed a simple **UI Interface** to visualize the Sentiment predictions performed by the BERT model using **gradio** library.

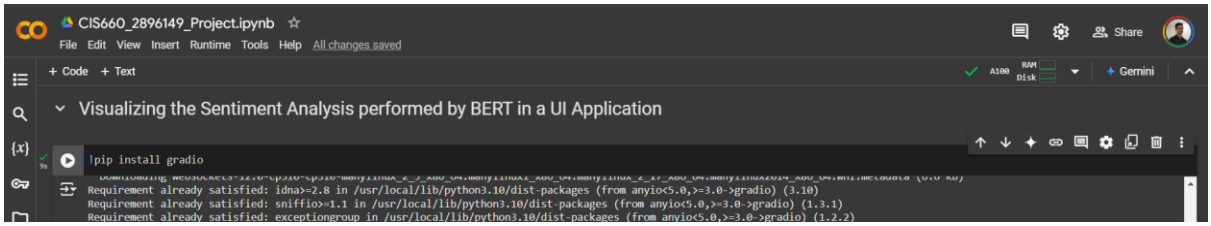


Figure 47: Installing the Pre-requisite libraries

Figure 48: UI Application using gradio

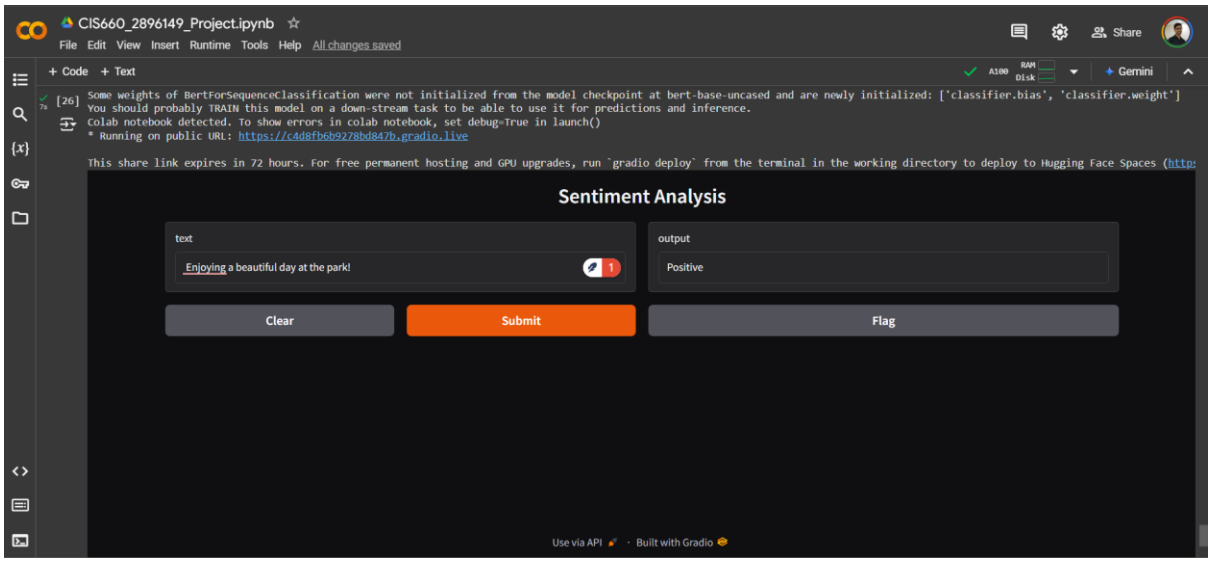


Figure 49: Interface of the application

Conclusion:

I would like to conclude my Project by the below Notes:

- ***Goal Achievement:***
  - Successfully implemented and evaluated sentiment analysis on a real-world dataset using both **XGBoost** and **BERT** models.
  - Achieved accurate predictions for sentiment classification into **Positive**, **Negative**, and **Neutral** classes.
- ***Key Performance Metrics:***
  - **XGBoost** achieved an accuracy of **88%**, showcasing its ability to handle structured and text-based features efficiently.
  - **BERT** achieved a perfect accuracy of **100%**, showcasing its pre-trained bidirectional transformer architecture to understand contextual relationships in text.

- ***Comparison of Models:***
    - **XGBoost**, while effective and faster for structured data and combined feature sets, lacks to capture deep contextual semantics compared to **BERT**.
    - **BERT** has excellent performance due to its transformer-based architecture, which models bidirectional relationships in text.
- ***Visualizing the BERT Model:***
    - A simple **UI Interface** was successfully developed using **Gradio**, enabling real-time sentiment analysis for user-provided text.
- ***Learning Outcomes:***
    - Data Preprocessing steps, including handling categorical features and tokenization, significantly impacted the model performance.
    - Hyperparameter tuning and advanced deep learning models like BERT can substantially enhance prediction accuracy but require a higher computational cost.
- ***Limitations / Challenges faced:***
    - **XGBoost** may require more engineered features or additional ensemble techniques for competitive performance.
    - Training the model was consuming because of the input Size **(100732x718).**
    - Training the **BERT** model was taking infinite amount of time in **Jupyter notebook**. So switched to **Google Colab GPU**.
    - Had to re-run the models several times to cross-check the metrics.
- ***Future Scope:***
    - Extend the dataset for analysis across multiple languages or more nuanced sentiment classes.
    - Fine-tuning BERT to avoid potential overfitting and validate its performance on unseen datasets.
    - •Integrating additional structured features (e.g., time-based trends or user demographics) to enhance model insights.
    - •Deploying the application on cloud platforms (e.g., AWS, GCP) for scalability and global accessibility.
    - •Exploring other transformer-based models for efficiency and performance balance.

Conclusion Statement:

- The project demonstrates the comparison of traditional machine learning model (**XGBoost**) with an advanced transformer-based Large Language Model (**BERT**) for their performance on **Sentiment Analysis**.
- The Results showcase the ability and efficiency of the transformer-based LLMs for natural language understanding and the importance of choosing models based on task requirements and resource availability.

Additional Steps:

I have zipped folder containing all the intermediate output files and the input files which are generated and used in the Google Colab Notebook and downloaded the zip file to my local system as below:

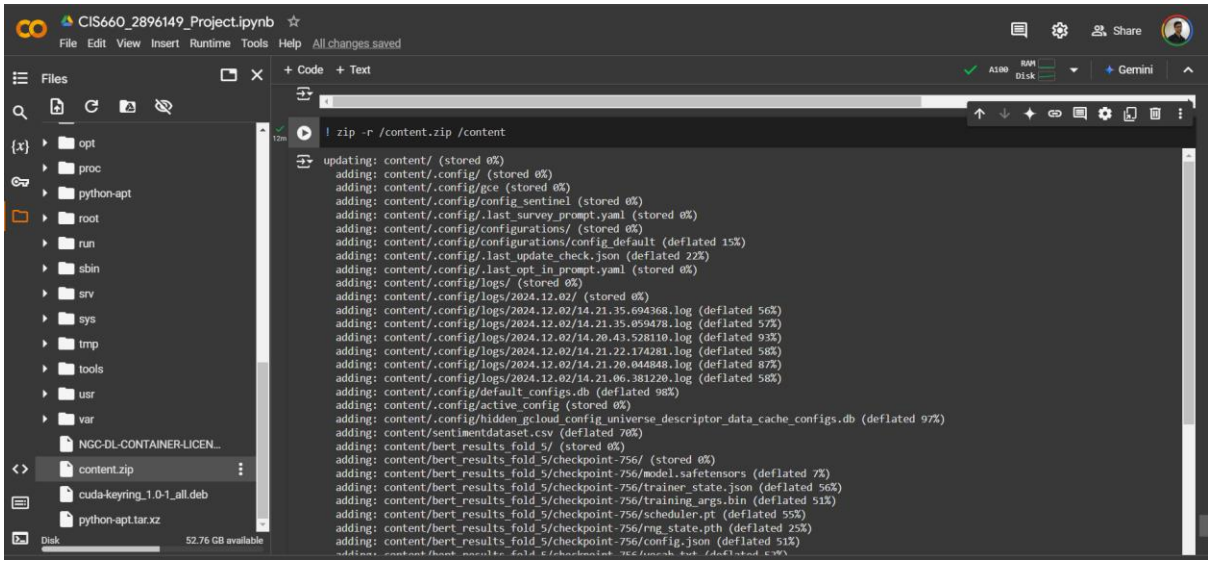


Figure 50: Perform zip of all the required files and folders

Figure 51: Mount the google drive and move the zip file to Google drive

I downloaded the 'content.zip' file from my Google drive to my local system which contains all the required files and folders.

Source Code:

Below is the entire Source code imported from the Jupyter Notebook 'CIS660_2896149_Project.ipynb':

```
# -*- coding: utf-8 -*-

"""CIS660_2896149_Project.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/drive/1Pm-uKlHW0d8En1C0E3SBtSBUTMkZ2aLF

## Importing the Required Libraries

"""

import pandas as pd

import re

"""## Uploading the Required Input File"""

from google.colab import files

uploaded = files.upload()

"""## Reading the input Dataset and display the Features for understanding"""

inputFile_df = pd.read_csv('sentimentdataset.csv')

inputFile_df

"""## Display the Dataset Information"""

inputFile_df.info()

"""## Generate more number of records based on the Analysis of the Dataset"""

import random

from datetime import datetime, timedelta

original_columns = inputFile_df.columns.tolist()

def modify_text(text):

    words = text.split()

    if len(words) > 1:

        random_index = random.randint(0, len(words) - 1)

        words[random_index] = words[random_index].upper()
```

```
    return " ".join(words)
def generate_random_timestamp(original_timestamp):
    original_date = datetime.strptime(original_timestamp, "%Y-%m-%d %H:%M:%S")
    random_offset = timedelta(days=random.randint(-10, 10), hours=random.randint(-5,
5))
    return (original_date + random_offset).strftime("%Y-%m-%d %H:%M:%S")
def generate_new_records(inputFile_df, num_records=100000):
    new_records = []
    for _ in range(num_records):
        sample_record = inputFile_df.sample(1).iloc[0]
        text = modify_text(sample_record["Text"])
        sentiment = sample_record["Sentiment"]
        timestamp = generate_random_timestamp(sample_record["Timestamp"])
        user = f"User{random.randint(100, 999)}"
        platform = sample_record["Platform"]
        hashtags = sample_record["Hashtags"]
        retweets = max(0, int(sample_record.get("Retweets", 0)) + random.randint(-5, 5))
        likes = max(0, int(sample_record.get("Likes", 0)) + random.randint(-10, 10))
        country = sample_record["Country"]
        year, month, day, hour = (
            int(timestamp[:4]),
            int(timestamp[5:7]),
            int(timestamp[8:10]),
            int(timestamp[11:13]),
        )
        new_record = {
            "Text": text,
            "Sentiment": sentiment,
            "Timestamp": timestamp,
            "User": user,
```

```
            "Platform": platform,
            "Hashtags": hashtags,
            "Retweets": retweets,
            "Likes": likes,
            "Country": country,
            "Year": year,
            "Month": month,
            "Day": day,
            "Hour": hour,
            "Unnamed: 0": sample_record.get("Unnamed: 0", None),
            "Unnamed: 0.1": sample_record.get("Unnamed: 0.1", None),
        }
        for col in original_columns:
            if col not in new_record:
                new_record[col] = sample_record.get(col, None)
        new_records.append(new_record)
    new_input_df = pd.DataFrame(new_records)[original_columns]
    return new_input_df
num_new_records = 100000
new_input_df = generate_new_records(inputFile_df, num_new_records)
final_input_df = pd.concat([inputFile_df, new_input_df], ignore_index=True)
output_file_path = "new_sentimentdataset.csv"
final_input_df.to_csv(output_file_path, index=False)
final_input_df
final_input_df.info()
"""## Data Cleaning and Preprocessing"""
# Dropping the columns whch are not required, such as 'Unnamed: 0.1' and'Unnamed: 0' and
# Cleaning of the 'Text' Column, like removing URLs, special characters, and the extra whitespaces and convert the text to lowercase:
```

```
final_input_df = final_input_df.drop(columns=['Unnamed: 0.1', 'Unnamed: 0'])

final_input_df['Text'] = final_input_df['Text'].replace(r'http\S+|www\S+|https\S+', '', regex=True)

final_input_df['Text'] = final_input_df['Text'].replace(r'\W', ' ', regex=True)

final_input_df['Text'] = final_input_df['Text'].replace(r'\s+[a-zA-Z]\s+', ' ', regex=True)

final_input_df['Text'] = final_input_df['Text'].replace(r'\s+', ' ', regex=True)

final_input_df['Text'] = final_input_df['Text'].str.lower()

final_input_df

"""## Feature Engineering on the Dataset"""

from collections import Counter

from sklearn.feature_extraction.text import TfidfVectorizer

final_input_df['is_weekend'] = pd.to_datetime(final_input_df['Timestamp']).dt.dayofweek.apply(lambda x: 1 if x >= 5 else 0)

final_input_df['post_season'] = pd.to_datetime(final_input_df['Timestamp']).dt.month.apply(

  lambda x: 'Spring' if 3 <= x <= 5 else ('Summer' if 6 <= x <= 8 else ('Fall' if 9 <= x <= 11 else 'Winter'))

)

user_post_counts = final_input_df['User'].value_counts()

final_input_df['user_post_count'] = final_input_df['User'].map(user_post_counts)

final_input_df['platform_encoded'] = final_input_df['Platform'].astype('category').cat.codes

all_hashtags = [tag.strip() for tags in final_input_df['Hashtags'].dropna() for tag in tags.split(',')]

hashtag_counts = Counter(all_hashtags)

final_input_df['global_hashtag_frequency'] = final_input_df['Hashtags'].apply(

  lambda x: sum(hashtag_counts[tag.strip()] for tag in str(x).split(',')) if x else 0

)

tfidf = TfidfVectorizer(tokenizer=lambda x: x.split(','), lowercase=True)

tfidf_matrix = tfidf.fit_transform(final_input_df['Hashtags'].fillna(''))
```

```
tfidf_features = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf.get_feature_names_out())

final_input_df = pd.concat([final_input_df.reset_index(drop=True),
tfidf_features.reset_index(drop=True)], axis=1)

final_input_df['engagement_score'] = final_input_df['Retweets'] + final_input_df['Likes']

engagement_threshold = final_input_df['engagement_score'].quantile(0.9)

final_input_df['is_viral'] = final_input_df['engagement_score'].apply(lambda x: 1 if x >
engagement_threshold else 0)

country_post_counts = final_input_df['Country'].value_counts()

final_input_df['country_post_count'] =
final_input_df['Country'].map(country_post_counts)

final_input_df

"""## Check for the Class Imbalance"""

print("\nCounts of all the Sentiments in the Dataset:")

print(final_input_df['Sentiment'].value_counts())

"""## Encoding the Sentiments"""

# Strip whitespace from sentiment values to standardize formatting

final_input_df['Sentiment'] = final_input_df['Sentiment'].str.strip()

final_input_df['Sentiment'].unique()

# Mapping all the different Sentiment Labels to 3 Selected Labels: Positive, Negative
and Neutral

dataSentimentMapping = {

  'Positive': 1, 'Happiness': 1, 'Joy': 1, 'Love': 1, 'Amusement': 1, 'Enjoyment': 1,
'Admiration': 1, 'Affection': 1, 'Awe': 1, 'Excitement': 1,

  'Kind': 1, 'Pride': 1, 'Elation': 1, 'Euphoria': 1, 'Contentment': 1, 'Serenity': 1, 'Gratitude':
1, 'Hope': 1, 'Empowerment': 1, 'Compassion': 1,

  'Tenderness': 1, 'Arousal': 1, 'Enthusiasm': 1, 'Fulfillment': 1, 'Reverence': 1,
'Anticipation': 1, 'Curiosity': 1, 'Zest': 1, 'Hopeful': 1,

  'Proud': 1, 'Grateful': 1, 'Empathetic': 1, 'Compassionate': 1, 'Playful': 1, 'Inspired': 1,
'Confident': 1, 'Overjoyed': 1, 'Motivation': 1,

  'Satisfaction': 1, 'Blessed': 1, 'Reflection': 1, 'Positivity': 1, 'Kindness': 1, 'Friendship': 1,
'Success': 1, 'Optimism': 1, 'Wonderment': 1,
```

'Enchantment': 1, 'Celebration': 1, 'JoyfulReunion': 1, 'Heartwarming': 1, 'Triumph': 1, 'Accomplishment': 1, 'Adventure': 1, 'Creativity': 1,

'Vibrancy': 1, 'Enthusiasm': 1, 'Engagement': 1, 'Inspired': 1, 'Wonder': 1, 'Freedom': 1, 'Confidence': 1, 'Ecstasy': 1, 'Happy': 1, 'Surprise': 1,

'Adoration': 1, 'Appreciation': 1, 'Radiance': 1, 'Rejuvenation': 1, 'Coziness': 1, 'Melodic': 1, 'FestiveJoy': 1, 'InnerJourney': 1,

'Dazzle': 1, 'Adrenaline': 1, 'ArtisticBurst': 1, 'CulinaryOdyssey': 1, 'Resilience': 1, 'Immersion': 1, 'Spark': 1, 'Marvel': 1, 'Exploration': 1,

'Amazement': 1, 'Romance': 1, 'Captivation': 1, 'Grandeur': 1, 'Emotion': 1, 'Energy': 1, 'Charm': 1, 'Colorful': 1, 'Hypnotic': 1, 'Journey': 1,

'Touched': 1, 'Sympathy': 1, 'Renewed Effort': 1, 'Solace': 1, 'Breakthrough': 1, 'Joy in Baking': 1, 'Mesmerizing': 1, 'Culinary Adventure': 1,

'Winter Magic': 1, 'Thrilling Journey': 1, 'Nature\'s Beauty': 1, 'Celestial Wonder': 1, 'Creative Inspiration': 1, 'Runway Creativity': 1,

'Ocean\'s Freedom': 1, 'Whispers of the Past': 1, 'Determination': 1, 'Thrill': 1, 'Suspense': 1, 'Relief': 1,

'Negative': 0, 'Anger': 0, 'Fear': 0, 'Sadness': 0, 'Disgust': 0, 'Disappointed': 0, 'Bitter': 0, 'Shame': 0, 'Grief': 0, 'Loneliness': 0,

'Jealousy': 0, 'Resentment': 0, 'Frustration': 0, 'Boredom': 0, 'Anxiety': 0, 'Intimidation': 0, 'Helplessness': 0, 'Envy': 0, 'Regret': 0,

'Despair': 0, 'Isolation': 0, 'Heartbreak': 0, 'Melancholy': 0, 'Sorrow': 0, 'Darkness': 0, 'Desperation': 0, 'Loss': 0, 'Ruins': 0, 'Desolation': 0,

'Exhaustion': 0, 'Suffering': 0, 'Betrayal': 0, 'Apprehensive': 0, 'Overwhelmed': 0, 'Dismissive': 0, 'Obstacle': 0, 'Pressure': 0, 'Challenge': 0,

'Miscalculation': 0, 'EmotionalStorm': 0, 'Isolation': 0, 'Confusion': 0, 'Disappointment': 0, 'Regret': 0, 'Numbness': 0, 'Bitterness': 0,

'Yearning': 0, 'Fearful': 0, 'Jealous': 0, 'Devastated': 0, 'Frustrated': 0, 'Envious': 0, 'Bittersweet': 0,

'Pensive': 0, 'LostLove': 0, 'Heartache': 0, 'Solitude': 0, 'Sad': 0, 'Hate': 0, 'Bad': 0, 'Loneliness': 0, 'Betrayal': 0, 'EmotionalStorm': 0,

'Darkness': 0,

'Neutral': 2, 'Acceptance': 2, 'Calmness': 2, 'Serenity': 2, 'Contentment': 2, 'Curiosity': 2, 'Mindfulness': 2, 'Tranquility': 2, 'Harmony': 2,

'Wonder': 2, 'Reflection': 2, 'Intrigue': 2, 'PlayfulJoy': 2, 'DreamChaser': 2, 'Contemplation': 2, 'Inspiration': 2, 'Elegance': 2,

```
    'JoyfulReunion': 2, 'Harmony': 2, 'Iconic': 2, 'Envisioning History': 2, 'Imagination': 2, 'Connection': 2, 'Heartwarming': 2, 'Free-spirited': 2,
    'Equilibrium': 2, 'Equanimity': 2, 'Centered': 2, 'Stillness': 2, 'Indifference': 2, 'Nostalgia': 2, 'Ambivalence': 2, 'Whimsy': 2, 'Sympathy': 2,
    'Solace': 2, 'Reflection': 2, 'Mischievous': 2, 'Embarrassed': 2, 'Bad': 2
}
final_input_df['Sentiment'] = final_input_df['Sentiment'].map(dataSentimentMapping)
final_input_df
"""## Splitting the Dataset into Training Set, Validation Set, and Testing Set"""
from sklearn.model_selection import train_test_split
X = final_input_df.drop('Sentiment', axis=1)
Y = final_input_df['Sentiment']
x_train, x_temp, y_train, y_temp = train_test_split(X, Y, test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)
"""## Perform the Classification using a Base Model (XGBoost)"""
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score
from scipy.sparse import hstack, csr_matrix
label_encoder = LabelEncoder()
final_input_df['platform_encoded'] = label_encoder.fit_transform(final_input_df['Platform'])
final_input_df['country_encoded'] = label_encoder.fit_transform(final_input_df['Country'])
X_train_tfidf = tfidf.fit_transform(x_train['Text'].fillna(''))
X_val_tfidf = tfidf.transform(x_val['Text'].fillna(''))
X_test_tfidf = tfidf.transform(x_test['Text'].fillna(''))
x_train = x_train.drop(columns=['Text', 'Timestamp', 'User', 'Platform', 'Hashtags', 'Country', 'post_season'], errors='ignore')
x_val = x_val.drop(columns=['Text', 'Timestamp', 'User', 'Platform', 'Hashtags', 'Country', 'post_season'], errors='ignore')
```

```
x_test = x_test.drop(columns=['Text', 'Timestamp', 'User', 'Platform', 'Hashtags', 'Country', 'post_season'], errors='ignore')

X_train_combined = hstack([X_train_tfidf, csr_matrix(x_train)])

X_val_combined = hstack([X_val_tfidf, csr_matrix(x_val)])

X_test_combined = hstack([X_test_tfidf, csr_matrix(x_test)])

xgb_model = XGBClassifier(n_estimators=300, max_depth=5, learning_rate=0.05, subsample=0.8, colsample_bytree=0.8, tree_method='hist', random_state=42)

xgb_model.fit(X_train_combined, y_train, eval_set=[(X_val_combined, y_val)], verbose=True)

y_val_pred = xgb_model.predict(X_val_combined)

print("Classification Report for Validation Dataset:")

print(classification_report(y_val, y_val_pred, target_names=['Positive', 'Negative', 'Neutral']))

print("Accuracy for Validation Dataset", accuracy_score(y_val, y_val_pred))

"""## Confusion Matrix for XGBoost"""

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_val, y_val_pred, labels=xgb_model.classes_)

plt.figure(figsize=(8, 6))

sns.heatmap(

  cm,

  annot=True,

  fmt="d",

  cmap="Blues",

  xticklabels=['Positive', 'Negative', 'Neutral'],

  yticklabels=['Positive', 'Negative', 'Neutral']

)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Heatmap for XGBoost:")
```

```
plt.show()
"""## Hyperparameter Optimization to improve Model Accuracy and Performance"""
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
param_grid = {
  'n_estimators': [100, 200, 300],
  'max_depth': [4, 5, 6],
  'learning_rate': [0.01, 0.05, 0.1],
  'subsample': [0.8, 1.0],
  'colsample_bytree': [0.8, 1.0]
}
grid_search = GridSearchCV(
  estimator=XGBClassifier(tree_method='hist', random_state=42),
  param_grid=param_grid,
  scoring='accuracy',
  cv=3,
  verbose=1,
  n_jobs=-1
)
grid_search.fit(X_train_combined, y_train)
print("Best Parameters for XGBoost for the input Dataset:", grid_search.best_params_)
print("Best Accuracy Score for XGBoost for the input Dataset:",
grid_search.best_score_)
"""## Perform the Classification using a Transformer-Based Large Language Model
(LLM): BERT (Bidirectional Encoder Representations from Transformers)"""
!pip install datasets transformers
!pip install wandb
import torch
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
TrainingArguments
```

```
from datasets import Dataset, DatasetDict
from sklearn.metrics import classification_report, accuracy_score
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
x_train['Text'] = final_input_df['Text']
x_train['Sentiment'] = final_input_df['Sentiment']
x_val['Text'] = final_input_df['Text']
x_val['Sentiment'] = final_input_df['Sentiment']
x_test['Text'] = final_input_df['Text']
x_test['Sentiment'] = final_input_df['Sentiment']
train_data = Dataset.from_pandas(x_train[['Text',
'Sentiment']].rename(columns={'Sentiment': 'label'}))
val_data = Dataset.from_pandas(x_val[['Text',
'Sentiment']].rename(columns={'Sentiment': 'label'}))
test_data = Dataset.from_pandas(x_test[['Text',
'Sentiment']].rename(columns={'Sentiment': 'label'}))
dataset = DatasetDict({"train": train_data, "validation": val_data, "test": test_data})
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
def tokenize_function(example):
  return tokenizer(example["Text"], padding="max_length", truncation=True,
max_length=128)
tokenized_dataset = dataset.map(tokenize_function, batched=True)
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=3)
model.to(device)
training_args = TrainingArguments(
  output_dir="./bert_results",
  eval_strategy="epoch",
  save_strategy="epoch",
  learning_rate=2e-5,
  per_device_train_batch_size=16,
```

```
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_total_limit=2,
    load_best_model_at_end=True
)
import os
os.environ["WANDB_DISABLED"] = "true"
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    processing_class=tokenizer
)
trainer.train()
val_results = trainer.predict(tokenized_dataset["validation"])
val_predictions = val_results.predictions.argmax(-1)
val_true_labels = val_results.label_ids
print("\nClassification Report for Validation Dataset:")
print(classification_report(val_true_labels, val_predictions, target_names=["Positive", "Negative", "Neutral"]))
val_accuracy = accuracy_score(val_true_labels, val_predictions)
print("Accuracy for Validation Dataset:", val_accuracy)
"""## Perform k-fold Cross-validation on the Validation Dataset and check the model performance again"""
from sklearn.model_selection import KFold
from datasets import Dataset
```

```
from transformers import Trainer, TrainingArguments
from sklearn.metrics import classification_report, accuracy_score
import numpy as np
validation_data = Dataset.from_pandas(x_val[['Text',
'Sentiment']].rename(columns={'Sentiment': 'label'}))
def tokenize_function(example):
  return tokenizer(example["Text"], padding="max_length", truncation=True,
max_length=128)
tokenized_validation_data = validation_data.map(tokenize_function, batched=True)
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
accuracies = []
classification_reports = []
for fold, (train_index, val_index) in enumerate(kf.split(tokenized_validation_data)):
  print(f"\nRunning Fold {fold + 1}/{k}...")
  train_subset = tokenized_validation_data.select(train_index)
  val_subset = tokenized_validation_data.select(val_index)
  trainer = Trainer(
    model=model,
    args=TrainingArguments(
      output_dir=f"./bert_results_fold_{fold + 1}",
      evaluation_strategy="epoch",
      save_strategy="epoch",
      learning_rate=2e-5,
      per_device_train_batch_size=16,
      per_device_eval_batch_size=16,
      num_train_epochs=3,
      weight_decay=0.01,
      logging_dir=f"./logs_fold_{fold + 1}",
      save_total_limit=2,
```

```
        load_best_model_at_end=True,
        report_to="none"
      ),
      train_dataset=train_subset,
      eval_dataset=val_subset,
      tokenizer=tokenizer,
    )
    trainer.train()
    val_results = trainer.predict(val_subset)
    val_predictions = val_results.predictions.argmax(-1)
    val_true_labels = val_results.label_ids
    fold_accuracy = accuracy_score(val_true_labels, val_predictions)
    accuracies.append(fold_accuracy)
    print(f"Fold {fold + 1} Accuracy:", fold_accuracy)
    print(f"Fold {fold + 1} Classification Report:")
    report = classification_report(val_true_labels, val_predictions,
target_names=["Positive", "Negative", "Neutral"])
    print(report)
    classification_reports.append(report)
print("\nAverage Accuracy for Validation Dataset across folds:", np.mean(accuracies))
"""## Evaluate the Model Performance on the Test Dataset"""
test_results = trainer.predict(tokenized_dataset["test"])
test_predictions = test_results.predictions.argmax(-1)
test_true_labels = test_results.label_ids
print("\nClassification Report for Test Dataset:")
print(classification_report(test_true_labels, test_predictions, target_names=["Positive",
"Negative", "Neutral"]))
test_accuracy = accuracy_score(test_true_labels, test_predictions)
print("Accuracy for Test Dataset:", test_accuracy)
"""## Visualizing the Sentiment Analysis performed by BERT in a UI Application"""
```

```
!pip install gradio

import gradio as gr

import torch

from transformers import BertTokenizer, BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=3)

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

def predict_sentiment(text):

  inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=128,
padding="max_length")

  outputs = model(**inputs)

  predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)

  sentiment = ["Positive", "Negative", "Neutral"]

  return sentiment[predictions.argmax().item()]

iface = gr.Interface(fn=predict_sentiment, inputs="text", outputs="text", title="Sentiment
Analysis")

iface.launch(share=True)

!zip -r /content.zip /content

from google.colab import drive

drive.mount('/content/drive')

!mv /content.zip /content/drive/MyDrive/
```