

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:.....	3
Input Dataset:.....	3
Source of Input Dataset:	3
Dataset Description:.....	4
Configuration of the Environment:	4
Importing the Required Libraries:	4
Reading the Input Dataset as a Data frame:.....	5
Dataset Information:	6
Generating more number of records by analysing the existing records in the Input Dataset:	6
New Dataset Information:	9
Data Cleaning and Preprocessing:	9
Feature Engineering on the Dataset:	11
Check for Class Imbalance:	13
Encoding the Sentiments and Map the Sentiments to Positive, Negative and Neutral:	13
Split the Input Data frame into Training Dataset, Validation Dataset and Test Dataset:	16
Perform the Classification using a Base Model (XGBoost):	16
Hyperparameter Optimisation for the Validation Dataset:	20
Transformer-Based Large Language Model (LLM): BERT:	21
k-Fold cross validation on the Validation Dataset:.....	25
Evaluate the Model Performance on Test Dataset:	28
Visualising the Sentiment Analysis performed by BERT on a UI Application:.....	28
Conclusion:.....	29
Additional Steps:.....	31
Source Code:.....	32

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:

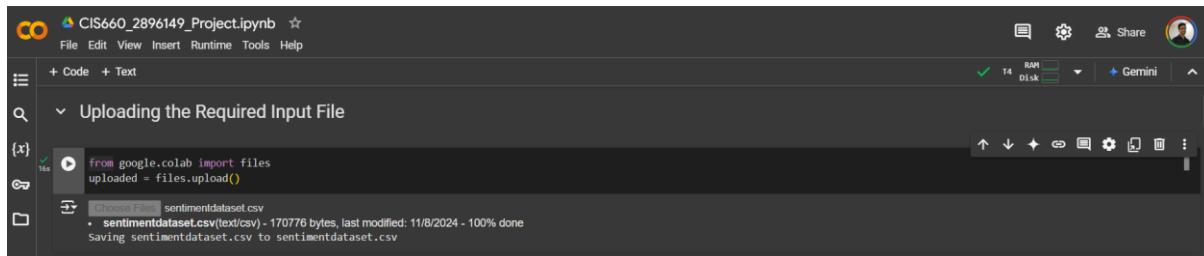
First, I import the standard libraries **Pandas** (to import, load and read the Dataset as a Data frame and perform the preprocessing operations on it) and **re** (for using regular expressions to perform the data preprocessing).

```
[1]: import pandas as pd
      import re
```

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:

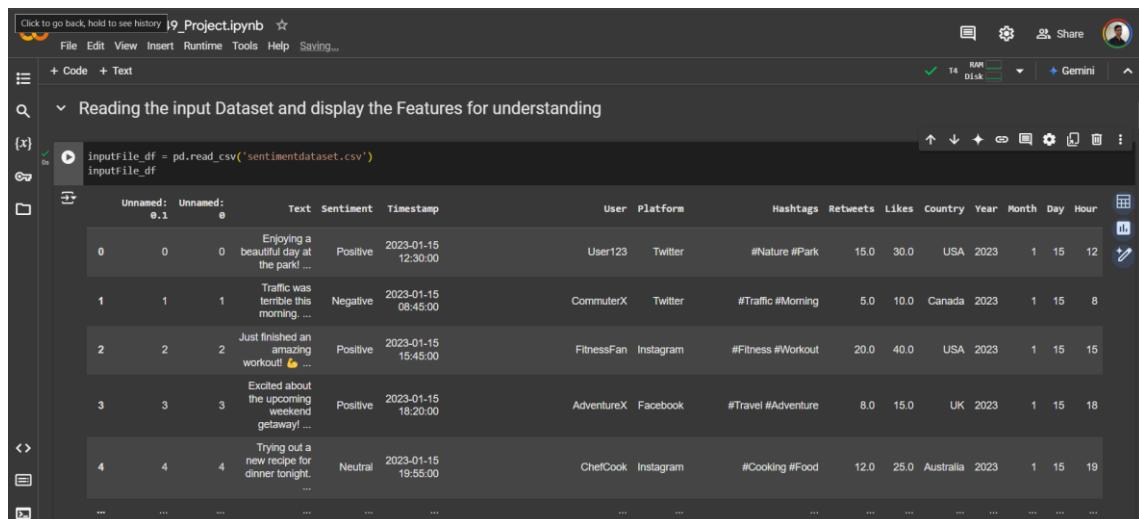


A screenshot of a Google Colab notebook titled 'CIS660_2896149_Project.ipynb'. The code cell contains the following Python code:

```
from google.colab import files
uploaded = files.upload()

Choose File: sentimentdataset.csv
  - sentimentdataset.csv (text/csv) - 170776 bytes, last modified: 11/8/2024 - 100% done
Saving sentimentdataset.csv to sentimentdataset.csv
```

Figure 3: Upload the csv file



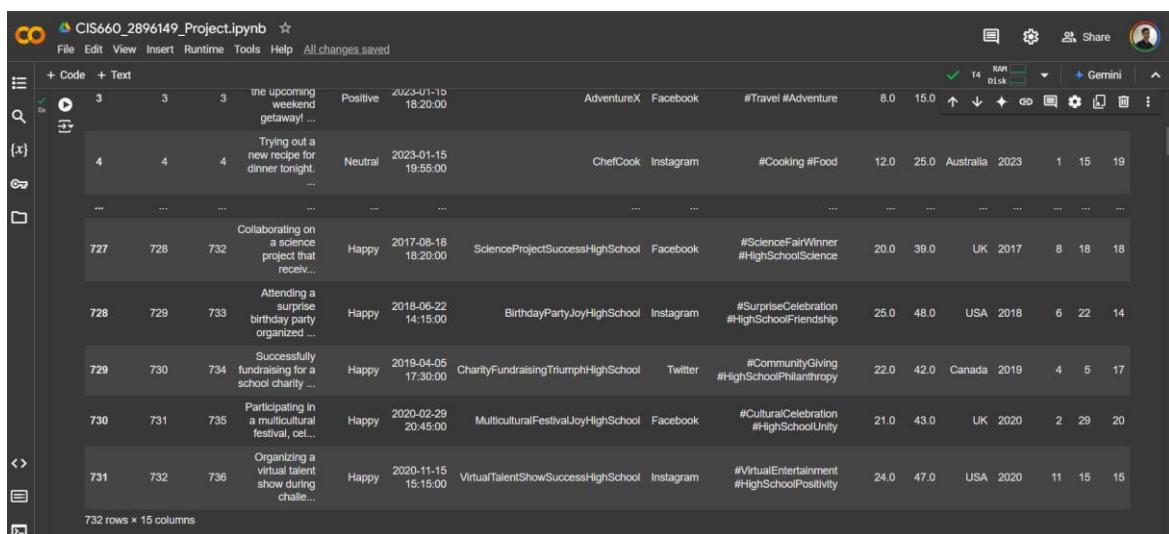
A screenshot of a Google Colab notebook titled 'CIS660_2896149_Project.ipynb'. The code cell contains the following Python code:

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

The resulting DataFrame is displayed as a table:

	Unnamed: 0	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	Month	Day	Hour
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	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	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	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	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
...

Figure 4: Reading the Dataset



A screenshot of a Google Colab notebook titled 'CIS660_2896149_Project.ipynb'. The code cell contains the following Python code:

```
inputFile_df
```

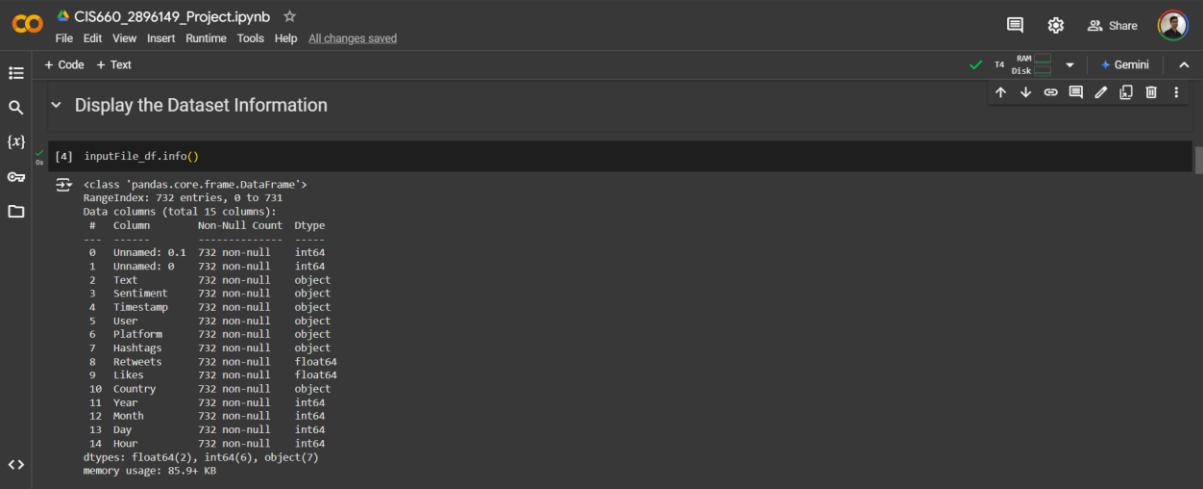
The resulting DataFrame is displayed as a table:

3	3	3	the upcoming weekend getaway! ...	Positive	2023-01-15 18:20:00	AdventureX	Facebook	#Travel #Adventure	8.0	15.0
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
...
727	728	732	Collaborating on a science project that received... ...	Happy	2017-08-18 18:20:00	ScienceProjectSuccessHighSchool	Facebook	#ScienceFairWinner #HighSchoolScience	20.0	39.0	UK	2017	8	18
728	729	733	Attending a surprise birthday party organized ...	Happy	2018-06-22 14:15:00	BirthdayPartyJoyHighSchool	Instagram	#SurpriseCelebration #HighSchoolFriendship	25.0	48.0	USA	2018	6	22
729	730	734	Successfully fundraising for a school charity ...	Happy	2019-04-05 17:30:00	CharityFundraisingTriumphHighSchool	Twitter	#CommunityGiving #HighSchoolPhilanthropy	22.0	42.0	Canada	2019	4	5
730	731	735	Participating in a multicultural festival, cele... ...	Happy	2020-02-29 20:45:00	MulticulturalFestivalJoyHighSchool	Facebook	#CulturalCelebration #HighSchoolUnity	21.0	43.0	UK	2020	2	29
731	732	736	Organizing a virtual talent show during challe... ...	Happy	2020-11-15 15:15:00	VirtualTalentShowSuccessHighSchool	Instagram	#VirtualEntertainment #HighSchoolPositivity	24.0	47.0	USA	2020	11	15
732 rows × 15 columns														

Figure 5: Reading the Dataset (contd..)

Dataset Information:

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



```
[4] inputFile_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 732 entries, 0 to 731
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Unnamed: 0.1    732 non-null   int64  
 1   Unnamed: 0     732 non-null   int64  
 2   Text          732 non-null   object  
 3   Sentiment     732 non-null   object  
 4   Timestamp     732 non-null   object  
 5   User          732 non-null   object  
 6   Platform      732 non-null   object  
 7   Hashtags      732 non-null   object  
 8   Retweets      732 non-null   float64 
 9   Likes          732 non-null   float64 
 10  Country        732 non-null   object  
 11  Year           732 non-null   int64  
 12  Month          732 non-null   int64  
 13  Day            732 non-null   int64  
 14  Hour           732 non-null   int64  
dtypes: float64(2), int64(6), object(7)
memory usage: 85.9+ KB
```

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.

```

File Edit View Insert Runtime Tools Help Saving...
+ Code + Text
Generate more number of records based on the Analysis of the Dataset
{x} 55
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=10000):
    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"]

        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] = None
    return new_records

```

Figure 7: Generating Synthetic Data

```

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] = None

```

Figure 8: Generating Sythetic Data (contd..)

The screenshot shows a Jupyter Notebook interface with a Python script titled 'CIS660_2896149_Project.ipynb'. The code generates synthetic data by selecting 100,000 random records from the original dataset and modifying them. It then concatenates these with the original dataset and saves the result as a CSV file.

```

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

```

The resulting DataFrame has 100,000 rows and 15 columns, including Text, Sentiment, Timestamp, User, Platform, Hashtags, Retweets, Likes, Country, Year, Month, Day, Hour, and two unnamed columns (0.1 and 0).

	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..)

This screenshot shows a larger portion of the generated synthetic DataFrame, displaying approximately 1000 rows. The structure remains the same, with columns for Text, Sentiment, Timestamp, User, Platform, Hashtags, Retweets, Likes, Country, Year, Month, Day, Hour, and two unnamed columns.

	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 #Science	17.0	40.0	USA	2023	10	21	17
100728	242	246	Ambivalence in the air, caught between the CRO... Ambivalence	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	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:

```

CIS660_2896149_Project.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
final_input_df.info()
{x}
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100732 entries, 0 to 100731
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Unnamed: 0.1  100732 non-null int64  
 1   Unnamed: 0    100732 non-null int64  
 2   Text         100732 non-null object  
 3   Sentiment    100732 non-null object  
 4   Timestamp    100732 non-null object  
 5   User         100732 non-null object  
 6   Platform     100732 non-null object  
 7   Hashtags     100732 non-null object  
 8   Retweets     100732 non-null float64 
 9   Likes        100732 non-null float64 
 10  Country      100732 non-null object  
 11  Year         100732 non-null int64  
 12  Month        100732 non-null int64  
 13  Day          100732 non-null int64  
 14  Hour         100732 non-null int64  
dtypes: float64(2), int64(6), object(?) 
memory usage: 11.5+ MB

```

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.

The screenshot shows a Jupyter Notebook interface with the title "CIS660_2896149_Project.ipynb". The code cell contains Python code for data cleaning:

```
# Dropping the columns which are not required, such as 'Unnamed: 0', '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', '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
```

Below the code is a table preview of the dataset:

	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

The screenshot shows a Jupyter Notebook interface with the title "CIS660_2896149_Project.ipynb". The code cell contains Python code for data cleaning, identical to Figure 12:

```
# Dropping the columns which are not required, such as 'Unnamed: 0', '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', '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
```

Below the code is a table preview of the dataset, showing more rows:

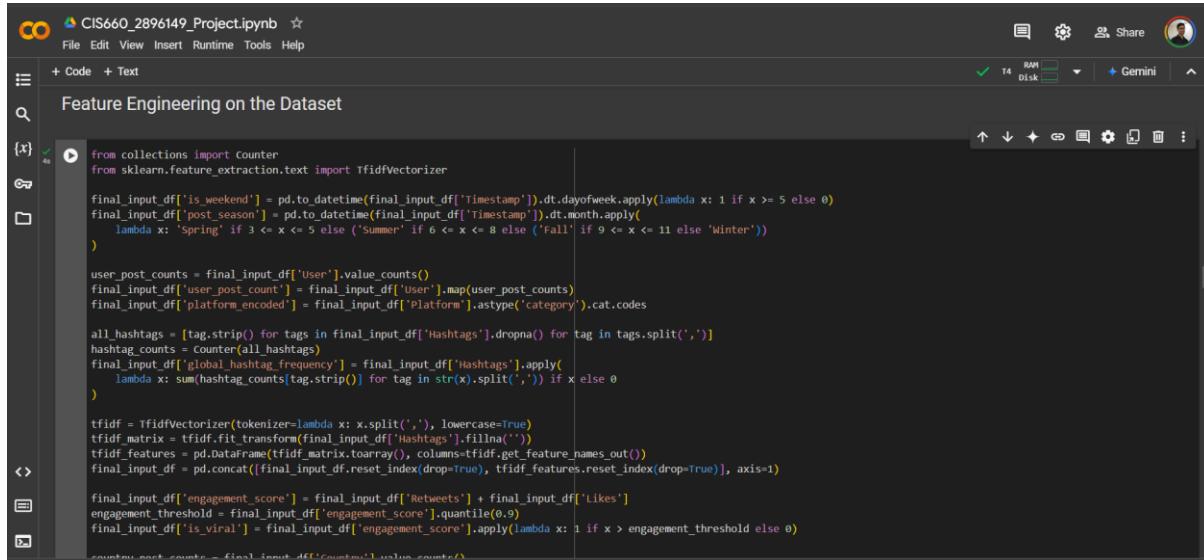
	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	Month	Day	Hour
0	enjoying beautiful day at the park	Positive	2023-01-15 12:30:00	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	1	15	12
1	traffic was terrible this morning	Negative	2023-01-15 08:45:00	CommuterX	Twitter	#Traffic #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

Text at the bottom of the table: 100732 rows x 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.



```
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'))
)

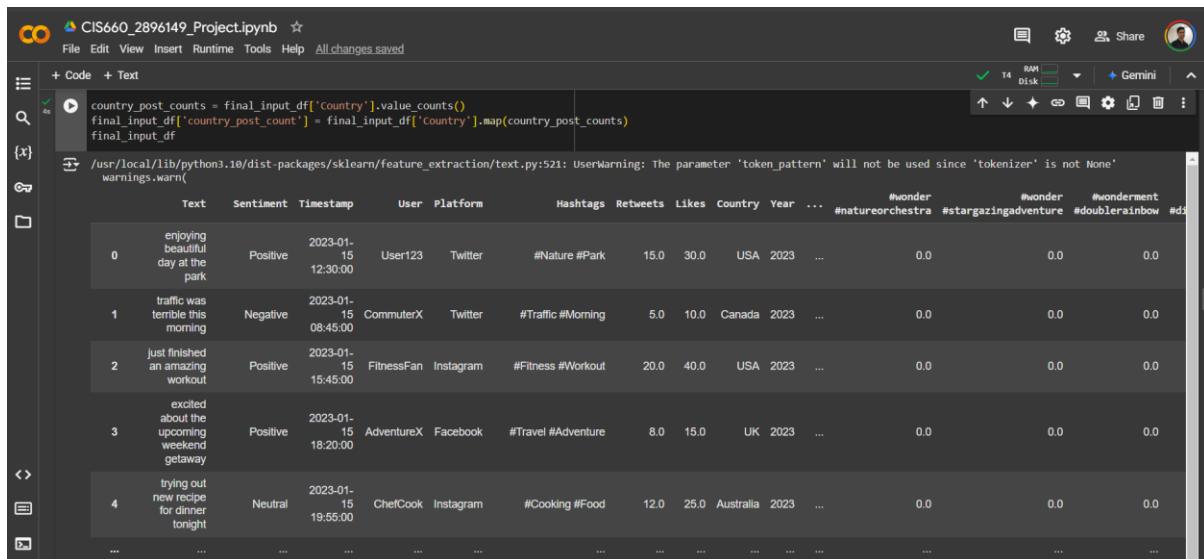
user_post_counts = final_input_df['User'].value_counts()
final_input_df['user_post_count'] = final_input_df['User'].map(user_post_counts)
final_input_df['platform_encoded'] = final_input_df['Platform'].astype('category').cat.codes

all_hashtags = [tag.strip() for tag in final_input_df['Hashtags'].dropna() for tag in tag.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.groupby('Country').value_counts()
```

Figure 14: Feature Engineering on the Dataset



	Text	Sentiment	Timestamp	User	Platform	Hashtags	Retweets	Likes	Country	Year	...	#wonder	#wonderorchestra	#stargazingadventure	#doublerainbow	#d
0	enjoying beautiful day at the park	Positive	2023-01-15 12:30:00	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	...	0.0	0.0	0.0	0.0	0.0
1	traffic was terrible this morning	Negative	2023-01-15 08:45:00	CommuterX	Twitter	#Traffic #Morning	5.0	10.0	Canada	2023	...	0.0	0.0	0.0	0.0	0.0
2	Just finished an amazing workout	Positive	2023-01-15 15:45:00	FitnessFan	Instagram	#Fitness #Workout	20.0	40.0	USA	2023	...	0.0	0.0	0.0	0.0	0.0
3	excited about the upcoming weekend getaway	Positive	2023-01-15 18:20:00	AdventureX	Facebook	#Travel #Adventure	8.0	15.0	UK	2023	...	0.0	0.0	0.0	0.0	0.0
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	...	0.0	0.0	0.0	0.0	0.0
...

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

The screenshot shows a Jupyter Notebook environment with a dark theme. The top navigation bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', and a status message 'All changes saved'. On the right side of the header are icons for disk, share, and user profile. The main content area is a table with the following approximate data:

Post ID	Content	Sentiment	Date	User ID	Platform	Hashtags	Likes	Retweets	Engagement Score
100727	for dinner tonight	Neutral	19-55:00	ChefCook	Instagram	#Cooking #Food	12.0	25.0	Australia 2023
100728	participating in science fair to showcase uniq...	Neutral	2023-10-21 17:45:00	User438	Facebook	#ScienceFair #HighSchoolScience	17.0	40.0	USA 2023
100729	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
100730	like comet of inspiration streaking through th...	Inspiration	2020-05-28 14:00:00	User524	Twitter	#Inspiration #BrilliantTrails	15.0	41.0	France 2020
100731	proudly scaling the peaks of achievement mount...	Proud	2020-01-03 13:45:00	User327	Twitter	#Proud #ScalingPeaks	26.0	38.0	USA 2020
100732	empowered to make difference in my community	Empowerment	2014-09-19 08:30:00	User978	Twitter	#Empowerment #Community	10.0	29.0	UK 2014

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

The detailed logic is explained below:

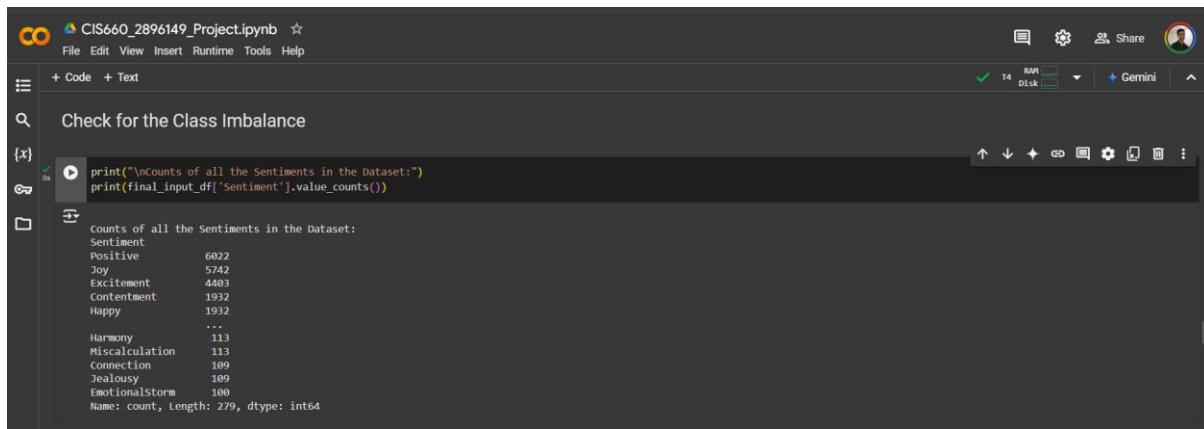
- **Timestamp-based Features:**
 - i. *is_weekend*: Convert the timestamp into datetime format and extract the day of the week, and then check if the post was made on the weekend (Saturday or Sunday)
 - ii. *post_season*: Extract the Month from the timestamp and map it to a season (e.g., March to May = 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.



```
File Edit View Insert Runtime Tools Help
+ Code + Text
Check for the Class Imbalance
{x}
print("\nCounts of all the Sentiments in the Dataset:")
print(final_input_df['Sentiment'].value_counts())
counts of all the Sentiments in the Dataset:
Sentiment
Positive      6022
Joy          5742
Excitement    4403
Contentment   1932
Happy         1932
...
Harmony        113
Miscalculation 113
Connection     109
Jealousy       109
EmotionalStorm 100
Name: count, Length: 279, dtype: int64
```

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.

```
# Strip whitespace from sentiment values to standardize formatting
final_input_df['sentiment'] = final_input_df['sentiment'].str.strip()
final_input_df['sentiment'].unique()

array(['Positive', 'Negative', 'Neutral', 'Anger', 'Fear', 'Sadness',
       'Disgust', 'Happiness', 'Joy', 'Love', 'Amusement', 'Enjoyment',
       'Admiration', 'Affection', 'Awe', 'Disappointed', 'Surprise',
       'Acceptance', 'Adoration', 'Anticipation', 'Bitter', 'Calmness',
       'Confusion', 'Excitement', 'Kind', 'Pride', 'Shame', 'Elation',
       'Euphoria', 'Contentment', 'Serenity', 'Gratitude', 'Hope',
       'Empowerment', 'Compassion', 'Tenderness', 'Arousal', 'Enthusiasm',
       'Fulfillment', 'Reverence', 'Despair', 'Grief', 'Loneliness',
       'Jealousy', 'Resentment', 'Frustration', 'Boredom', 'Anxiety',
       'Intimidation', 'Helplessness', 'Envy', 'Regret', 'Curiosity',
       'Indifference', 'Numbness', 'Melancholy', 'Nostalgia',
       'Ambivalence', 'Determination', 'Zest', 'Hopeful', 'Proud',
       'Grateful', 'Empathetic', 'Compassionate', 'Playful',
       'Free-spirited', 'Inspired', 'Confident', 'Bitterness', 'Yearning',
       'Fearful', 'Apprehensive', 'Overwhelmed', 'Jealous', 'Devastated',
       'Frustrated', 'Envious', 'Dismissive', 'Thrill', 'Bittersweet',
       'Overjoyed', 'Inspiration', 'Motivation', 'Contemplation',
       'JoyfulReunion', 'Satisfaction', 'Blessed', 'Reflection',
       'Appreciation', 'Confidence', 'Accomplishment', 'Wonderment',
       'Optimism', 'Enchantment', 'Intrigue', 'PlayfulJoy', 'Mindfulness',
       'DreamChaser', 'Elegance', 'Whimsy', 'Pensive', 'Harmony',
       'Creativity', 'Radiance', 'Wonder', 'Rejuvenation', 'Coziness',
       'Adventure', 'Melodic', 'FestiveJoy', 'InnerJourney', 'Freedom',
       'Dazzle', 'Adrenaline', 'ArtisticBurst', 'CulinaryOdyssey'],
      dtype=object)
```

Figure 18: Cleaning the Sentiment Values

```
'Fulfillment', 'Reverence', 'Despair', 'Grief', 'Loneliness',
'Jealousy', 'Resentment', 'Frustration', 'Boredom', 'Anxiety',
'Intimidation', 'Helplessness', 'Envy', 'Regret', 'Curiosity',
'Indifference', 'Numbness', 'Melancholy', 'Nostalgia',
'Ambivalence', 'Determination', 'Zest', 'Hopeful', 'Proud',
'Grateful', 'Empathetic', 'Compassionate', 'Playful',
'Free-spirited', 'Inspired', 'Confident', 'Bitterness', 'Yearning',
'Fearful', 'Apprehensive', 'Overwhelmed', 'Jealous', 'Devastated',
'Frustrated', 'Envious', 'Dismissive', 'Thrill', 'Bittersweet',
'Overjoyed', 'Inspiration', 'Motivation', 'Contemplation',
'JoyfulReunion', 'Satisfaction', 'Blessed', 'Reflection',
'Appreciation', 'Confidence', 'Accomplishment', 'Wonderment',
'Optimism', 'Enchantment', 'Intrigue', 'PlayfulJoy', 'Mindfulness',
'DreamChaser', 'Elegance', 'Whimsy', 'Pensive', 'Harmony',
'Creativity', 'Radiance', 'Wonder', 'Rejuvenation', 'Coziness',
'Adventure', 'Melodic', 'FestiveJoy', 'InnerJourney', 'Freedom',
'Dazzle', 'Adrenaline', 'ArtisticBurst', 'CulinaryOdyssey',
'Resilience', 'Immersion', 'Spark', 'Marvel', 'Heartbreak',
'Betrayal', 'Suffering', 'EmotionalStorm', 'Isolation',
'Disappointment', 'LostLove', 'Exhaustion', 'Sorrow', 'Darkness',
'Desperation', 'Ruins', 'Desolation', 'Loss', 'Heartache',
'Solitude', 'Positivity', 'Kindness', 'Friendship', 'Success',
'Exploration', 'Amazement', 'Romance', 'Captivation',
'Travel', 'Adventure', 'Emotion', 'Energy', 'Celebration',
'Charm', 'Ecstasy', 'Colorful', 'Hypnotic', 'Connection', 'Iconic',
'Journey', 'Engagement', 'Touched', 'Milestone', 'Triumph',
'Heartwarming', 'Obstacle', 'Sympathy', 'Pressure',
'RewardEffect', 'Misadventure', 'Challenge', 'Solace',
'Breakthrough', 'Joy in Baking', 'Envisioning History',
'Imagination', 'Vibrancy', 'Mesmerizing', 'Culinary Adventure',
'Winter Magic', 'Thrilling Journey', 'Nature's Beauty',
'Celestial Wonder', 'Creative Inspiration', 'Runway Creativity',
'Ocean's Freedom', 'Whispers of the Past', 'Relief', 'Embarrassed',
'Mischiefous', 'sad', 'hate', 'bad', 'Happy'], dtype=object)
```

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

```

# 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, 'Molodic': 1, 'FestiveJoy': 1, 'InnerJourney': 1,
    'Dazzle': 1, 'Adrenaline': 1, 'ArtisticBurst': 1, 'CulinaryOdyssey': 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, 'RenewedEffort': 1, 'Solace': 1, 'Breakthrough': 1, 'JoyinBaking': 1, 'Mesmerizing': 1, 'CulinaryAdventure': 1,
    'WinterMagic': 1, 'ThrillingJourney': 1, 'Nature'sBeauty': 1, 'CelestialWander': 1, 'CreativeInspiration': 1, 'RunawayCreativity': 1,
    'Ocean'sFreedom': 1, 'WhispersofthePast': 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, 'Lustlove': 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, 'EnvisioningHistory': 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
}

```

Figure 20: Mapping the Sentiment Classes

0	enjoying beautiful day at the park	1	2023-01-15	User123	Twitter	#Nature #Park	15.0	30.0	USA	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	traffic was terrible this morning	0	2023-01-15	CommuterX	Twitter	#Traffic #Morning	5.0	10.0	Canada	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	just finished an amazing workout	1	2023-01-15	FitnessFan	Instagram	#Fitness #Workout	20.0	40.0	USA	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	excited about the upcoming weekend getaway	1	2023-01-15	AdventureX	Facebook	#Travel #Adventure	8.0	15.0	UK	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	trying out new recipe for dinner tonight	2	2023-01-15	ChefCook	Instagram	#Cooking #Food	12.0	25.0	Australia	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	
100727	participating in science fair to showcase uniq...	2	2023-10-21	User438	Facebook	#ScienceFair #HighSchoolScience	17.0	40.0	USA	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

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

100727	for dinner tonight	1	19:55:00																
100728	participating in science fair to showcase uniq...	2	2023-10-21	User438	Facebook	#ScienceFair #HighSchoolScience	17.0	40.0	USA	2023	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100729	ambivalence in the air caught between the cros...	2	2022-08-07	User120	Instagram	#Ambivalence #CrossroadsConfusion	18.0	28.0	Canada	2022	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100730	like comet of inspiration streaking through th...	2	2020-05-28	User524	Twitter	#Inspiration #BrilliantTrails	15.0	41.0	France	2020	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100731	proudly scaling the peaks of achievement mount...	1	2020-01-03	User327	Twitter	#Proud #ScalingPeaks	26.0	38.0	USA	2020	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100732	empowered to make difference in my community	1	2014-09-19	User978	Twitter	#Empowerment #Community	10.0	29.0	UK	2014	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

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



```
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)
```

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.

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

Where,

- $F_m(x)$: New prediction after m^{th} iteration.
- $F_{m-1}(x)$: Prediction after $(m-1)^{\text{th}}$ 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:

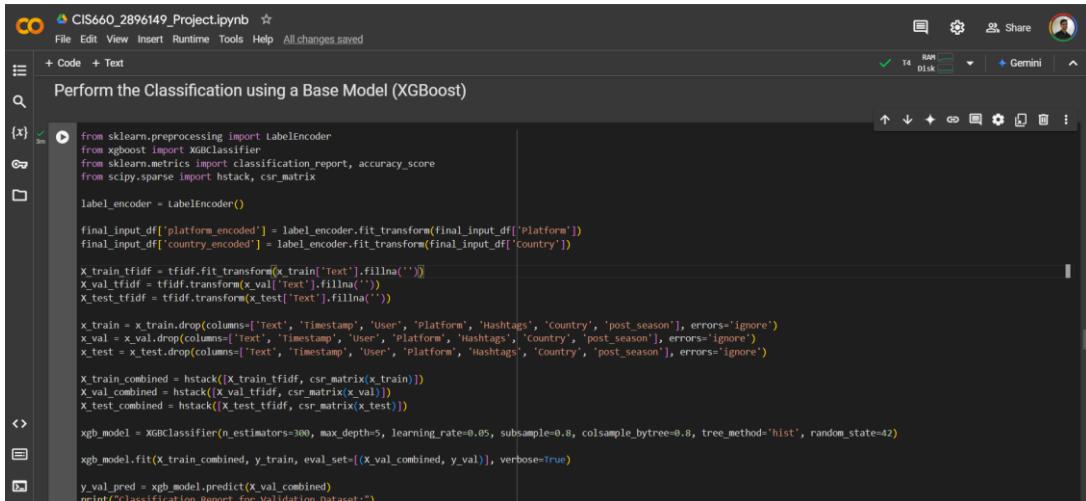
- The gain for splitting a node is calculated as:

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

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.



The screenshot shows a Jupyter Notebook interface with the title "Perform the Classification using a Base Model (XGBoost)". The code cell contains the following Python script:

```

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)])

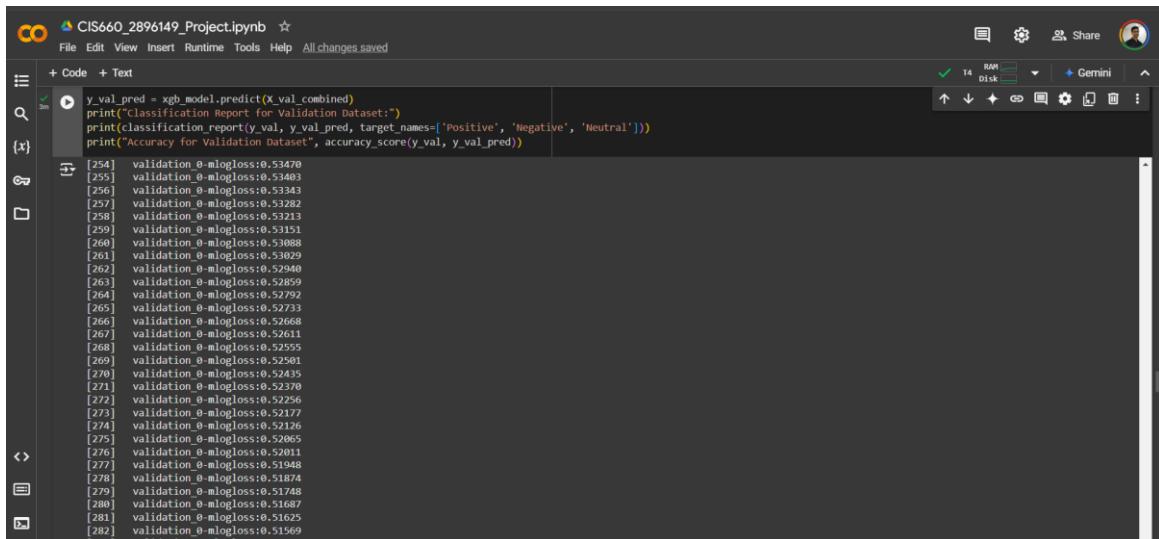
xgb_model = XGBClassifier(n_estimators=300, max_depth=5, learning_rate=0.05, subsample=0.8, colsample_bytree=0.8, tree_method="hist", random_state=42)

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:")

```

Figure 24: XGBoost Model



The screenshot shows a Jupyter Notebook interface with the title "Perform the Classification using a Base Model (XGBoost)". The code cell contains the following Python script:

```

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))

[254] validation_0-mlogloss:0.53470
[255] validation_0-mlogloss:0.53403
[256] validation_0-mlogloss:0.53343
[257] validation_0-mlogloss:0.53282
[258] validation_0-mlogloss:0.53213
[259] validation_0-mlogloss:0.53151
[260] validation_0-mlogloss:0.53088
[261] validation_0-mlogloss:0.53029
[262] validation_0-mlogloss:0.52940
[263] validation_0-mlogloss:0.52909
[264] validation_0-mlogloss:0.52792
[265] validation_0-mlogloss:0.52733
[266] validation_0-mlogloss:0.52668
[267] validation_0-mlogloss:0.52611
[268] validation_0-mlogloss:0.52555
[269] validation_0-mlogloss:0.52501
[270] validation_0-mlogloss:0.52435
[271] validation_0-mlogloss:0.52370
[272] validation_0-mlogloss:0.52256
[273] validation_0-mlogloss:0.52177
[274] validation_0-mlogloss:0.52126
[275] validation_0-mlogloss:0.52065
[276] validation_0-mlogloss:0.52011
[277] validation_0-mlogloss:0.51948
[278] validation_0-mlogloss:0.51874
[279] validation_0-mlogloss:0.51838
[280] validation_0-mlogloss:0.51687
[281] validation_0-mlogloss:0.51625
[282] validation_0-mlogloss:0.51569

```

Figure 25: XGBoost Model (contd..)

The screenshot shows a Jupyter Notebook interface with the file 'CIS660_2896149_Project.ipynb'. The code cell contains validation loop output and a classification report for the validation dataset. The report includes precision, recall, f1-score, and support for Positive, Negative, and Neutral classes, along with overall accuracy metrics.

```

[279] validation 0-mlogloss:0.51748
[280] validation 0-mlogloss:0.51687
[281] validation 0-mlogloss:0.51625
[282] validation 0-mlogloss:0.51569
[283] validation 0-mlogloss:0.51473
[284] validation 0-mlogloss:0.51418
[285] validation 0-mlogloss:0.51355
[286] validation 0-mlogloss:0.51299
[287] validation 0-mlogloss:0.51184
[288] validation 0-mlogloss:0.51093
[289] validation 0-mlogloss:0.51032
[290] validation 0-mlogloss:0.50971
[291] validation 0-mlogloss:0.50910
[292] validation 0-mlogloss:0.50855
[293] validation 0-mlogloss:0.50791
[294] validation 0-mlogloss:0.50714
[295] validation 0-mlogloss:0.50656
[296] validation 0-mlogloss:0.50598
[297] validation 0-mlogloss:0.50538
[298] validation 0-mlogloss:0.50478
[299] validation 0-mlogloss:0.50420
Classification Report for Validation Dataset:
precision    recall    f1-score   support
Positive      0.94     0.81     0.87     3798
Negative      0.84     0.99     0.91     8190
Neutral       1.00     0.69     0.82     3122

accuracy          0.8843812045003309
macro avg       0.93     0.83     0.87     15110
weighted avg    0.90     0.88     0.88     15110

```

Accuracy for Validation Dataset 0.8843812045003309

Figure 26: XGBoost Model (contd..)

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

The screenshot shows a Jupyter Notebook interface with the file 'CIS660_2896149_Project.ipynb'. The code cell contains Python code to generate a confusion matrix heatmap for XGBoost using matplotlib and seaborn libraries.

```

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()

```

Figure 27:Confusion Matrix for XGBoost

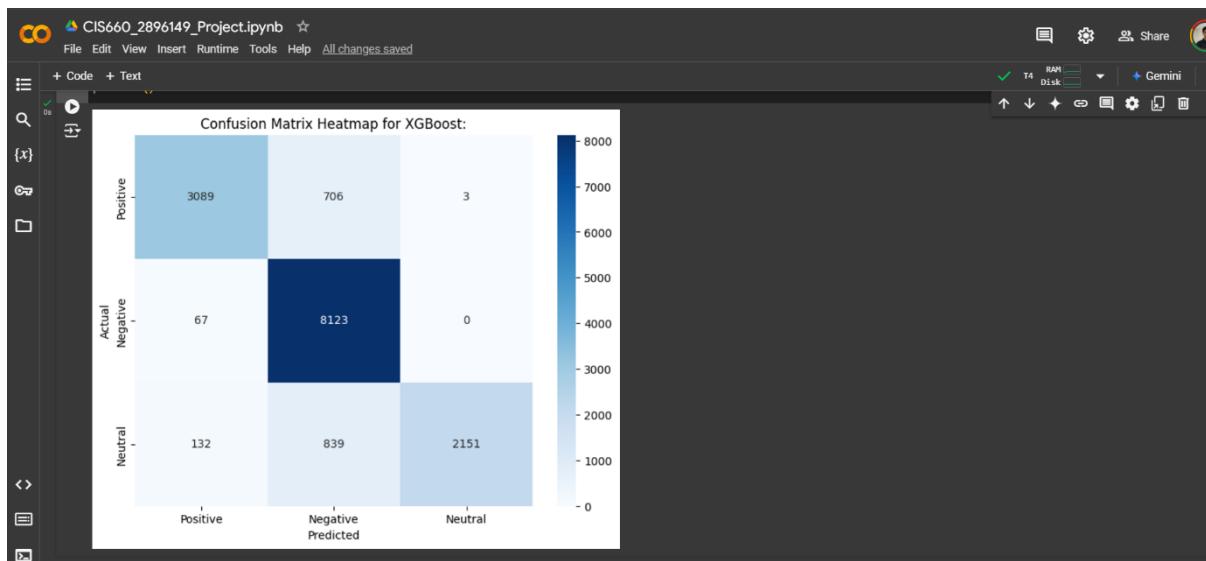


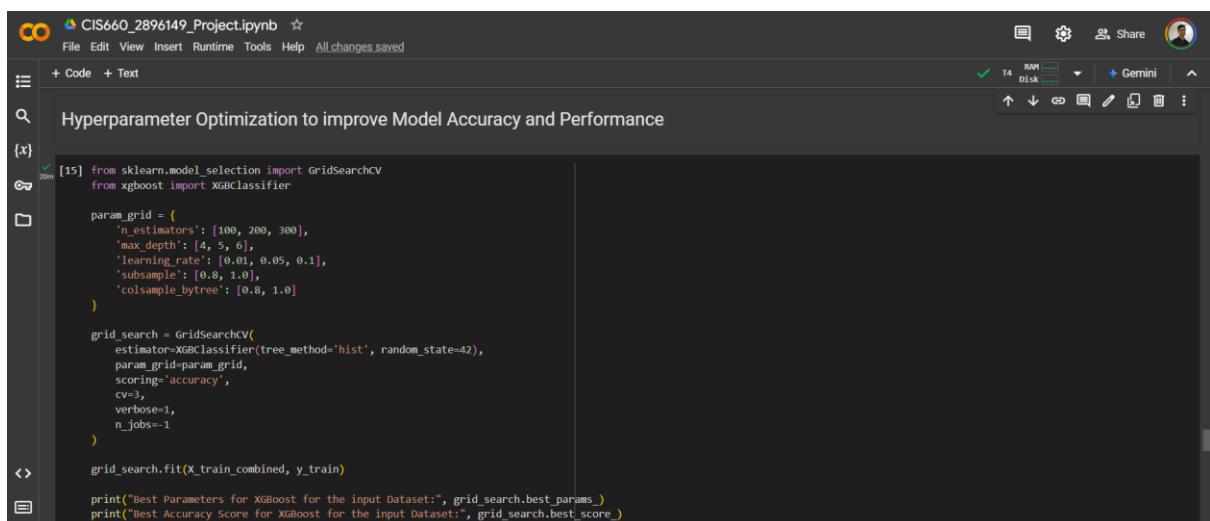
Figure 28: Confusion Matrix for XGBoost (contd..)

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.

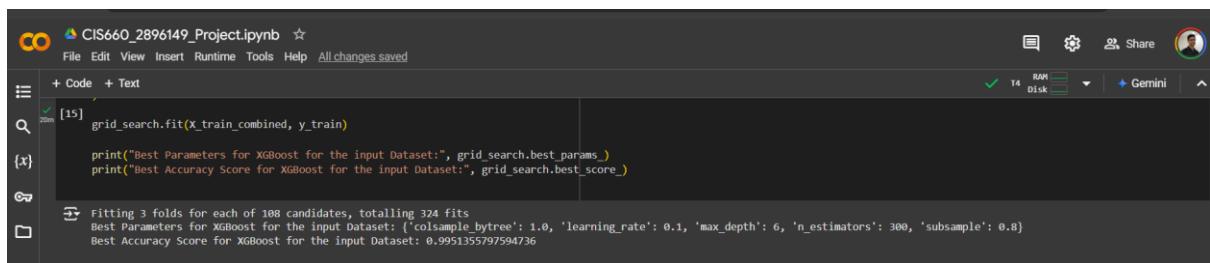
```
'n_estimators': [100, 200, 300],  
'max_depth': [4, 5, 6],  
'learning_rate': [0.01, 0.05, 0.1],  
'subsample': [0.8, 1.0],  
'colsample_bytree': [0.8, 1.0]
```



A screenshot of a Jupyter Notebook interface titled 'CIS660_2896149_Project.ipynb'. The code cell contains Python code for performing GridSearchCV on an XGBoost classifier. The code defines a parameter grid with 'n_estimators' from [100, 200, 300], 'max_depth' from [4, 5, 6], 'learning_rate' from [0.01, 0.05, 0.1], 'subsample' from [0.8, 1.0], and 'colsample_bytree' from [0.8, 1.0]. It then creates a GridSearchCV object, fits it to the training data, and prints the best parameters and score. The notebook interface shows the code in a light gray background, with syntax highlighting for keywords and variables.

```
[15] 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_)
```

Figure 29: Hyperparameter Optimisation for XGBoost



A screenshot of a Jupyter Notebook interface titled 'CIS660_2896149_Project.ipynb'. The code cell contains Python code for fitting a GridSearchCV object to the training data and printing the best parameters and score. The output shows the fitting process, the best parameters found ('colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 300, 'subsample': 0.8), and the best accuracy score (0.9951355797594736). The notebook interface shows the code in a light gray background, with syntax highlighting for keywords and variables.

```
[15] 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_)  
  
Fitting 3 folds for each of 108 candidates, totalling 324 fits  
Best Parameters for XGBoost for the input Dataset: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 300, 'subsample': 0.8}  
Best Accuracy Score for XGBoost for the input Dataset: 0.9951355797594736
```

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.

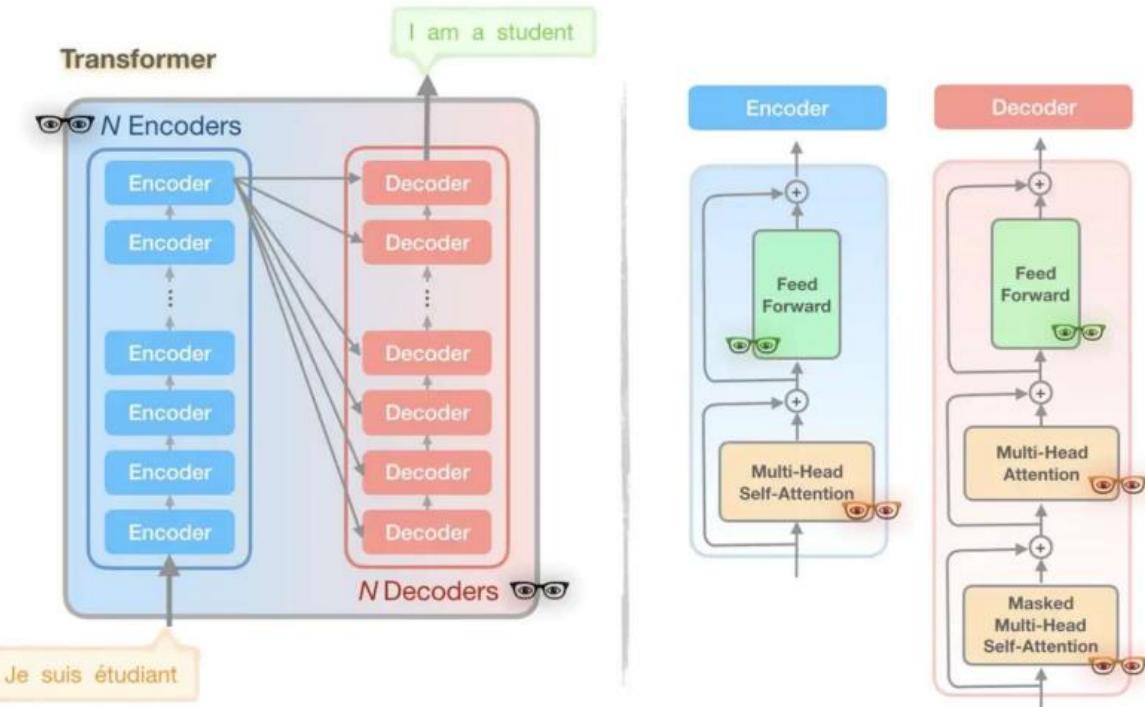


Figure 31: Transformer-based Architecture

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.

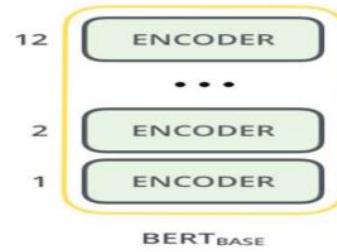
We can summarize the overall BERT architecture from the diagram as below:

- Encoder:** Processes and learns contextual embeddings for inputs (used in BERT).
- Decoder:** Generates outputs based on encoder embeddings (not used in BERT).
- Attention Mechanism:** Allows learning dependencies and relationships within input tokens (self-attention) and between input-output tokens (encoder-decoder attention).
- 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:

- **BERT Base**

- Transformer Blocks (L) = 12
- Hidden Size (H) = 768
- Self-Attention Heads (A) = 12
- Total Parameters = **110 Million**



- **BERT Large**

- Transformer Blocks (L) = 24
- Hidden Size (H) = 1024
- Self-Attention Heads (A) = 16
- Total Parameters = **340 Million**

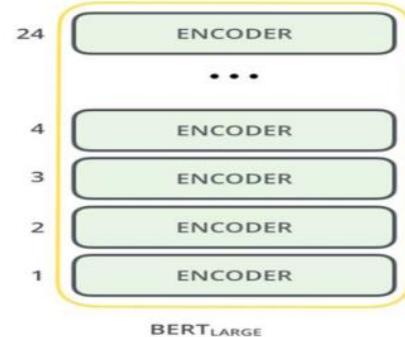


Figure 32: BERT Models

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**.

```

File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Perform the Classification using a Transformer-Based Large Language Model (LLM): BERT
(Bidirectional Encoder Representations from Transformers)
!pip install datasets transformers
!pip install wandb
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (3.1.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.2)

```

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:

```

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)

```

Figure 34: Training the BERT model

```

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

```

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

```

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)

Using device: cuda
Map: 100% 70512/70512 [00:57<00:00, 2399.31 examples/s]
Map: 100% 15110/15110 [00:06<00:00, 2422.12 examples/s]
Map: 100% 15110/15110 [00:07<00:00, 2449.11 examples/s]

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Using the 'WANDB_DISABLE' environment variable is deprecated and will be removed in v5. Use the --report_to flag to control the integrations used for logging result (for instance --report_to wandb)
[13221/13221 1:21:48, Epoch 3/3]

Epoch Training Loss Validation Loss
1 0.000000 0.000011
2 0.000000 0.000002
3 0.000000 0.000001

Classification Report for Validation Dataset:
precision recall f1-score support
Positive 1.00 1.00 1.00 3798
Negative 1.00 1.00 1.00 8190
Neutral 1.00 1.00 1.00 3122

accuracy 1.00 1.00 1.00 15110
macro avg 1.00 1.00 1.00 15110
weighted avg 1.00 1.00 1.00 15110

```

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

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

```

print("Accuracy for Validation Dataset:", val_accuracy)

Using device: cuda
Map: 100% 70512/70512 [00:57<00:00, 2399.31 examples/s]
Map: 100% 15110/15110 [00:06<00:00, 2422.12 examples/s]
Map: 100% 15110/15110 [00:07<00:00, 2449.11 examples/s]

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Using the 'WANDB_DISABLE' environment variable is deprecated and will be removed in v5. Use the --report_to flag to control the integrations used for logging result (for instance --report_to wandb)
[13221/13221 1:21:48, Epoch 3/3]

Epoch Training Loss Validation Loss
1 0.000000 0.000011
2 0.000000 0.000002
3 0.000000 0.000001

Classification Report for Validation Dataset:
precision recall f1-score support
Positive 1.00 1.00 1.00 3798
Negative 1.00 1.00 1.00 8190
Neutral 1.00 1.00 1.00 3122

accuracy 1.00 1.00 1.00 15110
macro avg 1.00 1.00 1.00 15110
weighted avg 1.00 1.00 1.00 15110

Accuracy for Validation Dataset: 1.0

```

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.

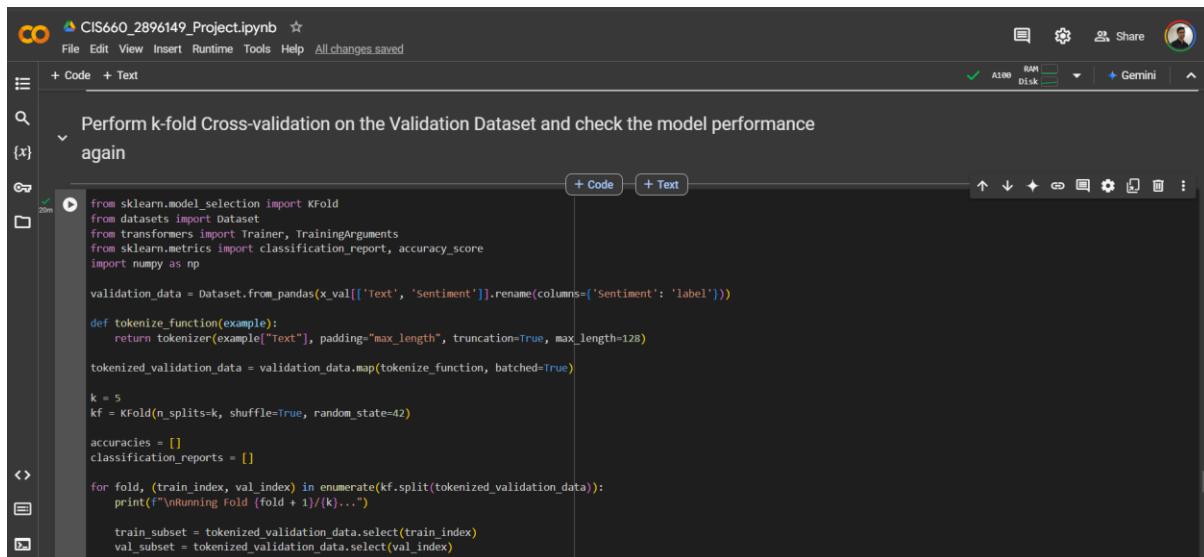
To cross check my model performance, I need some other performance evaluation techniques like Hyperparameter tuning (training the model multiple times with different parameters) or K-fold cross validation (evaluating the model accuracy for k subsets of data and then calculating the average metrics to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



The screenshot shows a Jupyter Notebook interface with the file 'CIS660_2896149_Project.ipynb' open. The code cell contains Python code for performing k-fold cross-validation on a validation dataset using BERT. The code imports necessary libraries from sklearn, datasets, transformers, and sklearn.metrics. It defines a tokenization function and uses KFold to split the validation data into 5 folds. For each fold, it prints the fold number, creates training and validation subsets, and then performs training and evaluation using a Trainer object. The validation subset is used to predict labels, and the accuracy is calculated and appended to a list of accuracies.

```
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)

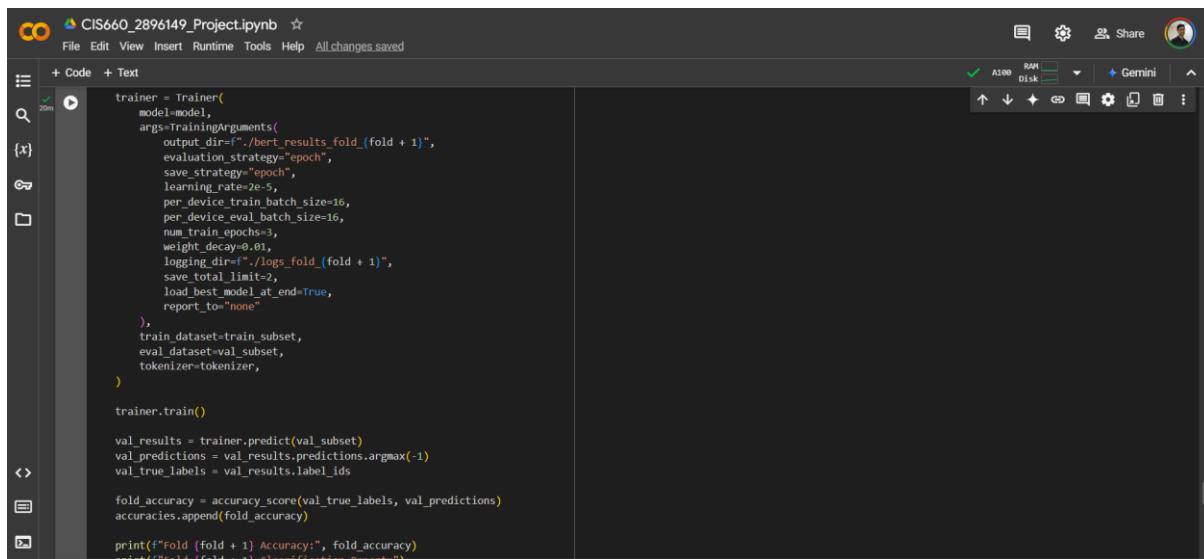
tokenized_validation_data = validation_data.map(tokenize_function, batched=True)

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)
```

Figure 38: k-fold cross validation on BERT Model



This screenshot continues the Jupyter Notebook session from Figure 38. The code cell now includes the continuation of the k-fold cross-validation loop. It shows the creation of a Trainer object with specific arguments for training and evaluation, including the use of different batch sizes for training and evaluation, and a logging directory. The loop iterates through the folds, performing training and evaluation for each, and calculating the accuracy of the validation subset for each fold. The final accuracy values are printed at the end.

```
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)
```

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

```

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))

```

Map: 100% 15110/15110 [00:06:00.00, 2358.57 examples/s]

Running Fold 1/5...
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: 'evaluation_strategy' is deprecated and will be removed in version 4.46 of Transformers
warnings.warn(
<ipython-input-23-eb6e6d3309da>:26: FutureWarning: 'tokenizer' is deprecated and will be removed in version 5.0.0 for 'Trainer.__init__', use 'processing_class' instead.
trainer = Trainer(
[2268/2268 03:52, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000

Fold 1 Accuracy: 1.0
Fold 1 Classification Report:
precision recall f1-score support

	precision	recall	f1-score	support
Positive	1.00	1.00	1.00	776
Negative	1.00	1.00	1.00	1615
Neutral	1.00	1.00	1.00	631

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

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

```

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))

```

Running Fold 1/5...
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: 'evaluation_strategy' is deprecated and will be removed in version 4.46 of Transformers
warnings.warn(
<ipython-input-23-eb6e6d3309da>:26: FutureWarning: 'tokenizer' is deprecated and will be removed in version 5.0.0 for 'Trainer.__init__', use 'processing_class' instead.
trainer = Trainer(
[2268/2268 03:52, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000

Fold 1 Accuracy: 1.0
Fold 1 Classification Report:
precision recall f1-score support

	precision	recall	f1-score	support
Positive	1.00	1.00	1.00	776
Negative	1.00	1.00	1.00	1615
Neutral	1.00	1.00	1.00	631

accuracy
macro avg
weighted avg

Figure 41: Accuracy for k=1

```

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))

```

Running Fold 2/5...
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: 'evaluation_strategy' is deprecated and will be removed in version 4.46 of Transformers
warnings.warn(
<ipython-input-23-eb6e6d3309da>:26: FutureWarning: 'tokenizer' is deprecated and will be removed in version 5.0.0 for 'Trainer.__init__', use 'processing_class' instead.
trainer = Trainer(
[2268/2268 03:54, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000

Fold 2 Accuracy: 1.0
Fold 2 Classification Report:
precision recall f1-score support

	precision	recall	f1-score	support
Positive	1.00	1.00	1.00	664
Negative	1.00	1.00	1.00	1708
Neutral	1.00	1.00	1.00	650

accuracy
macro avg
weighted avg

Figure 42: Accuracy for k=2

```

print("Fold {} Accuracy: {}, fold_accuracy")
print("Fold {} 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))

```

Running Fold 3/5...
 /usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of Transformers
 warnings.warn(
 <ipython-input-23-eb6e6d3309da>:26: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
 trainer = Trainer(
[2268/2268 03:53, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000

Fold 3 Accuracy: 1.0
 Fold 3 Classification Report:
 precision recall f1-score support

	precision	recall	f1-score	support
Positive	1.00	1.00	1.00	784
Negative	1.00	1.00	1.00	1609
Neutral	1.00	1.00	1.00	629
accuracy			1.00	3022
macro avg	1.00	1.00	1.00	3022
weighted avg	1.00	1.00	1.00	3022

Figure 43: Accuracy for $k=3$

```

print(report)
classification_reports.append(report)

print("\nAverage Accuracy for Validation Dataset across folds: ", np.mean(accuracies))

```

Running Fold 4/5...
 /usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of Transformers
 warnings.warn(
 <ipython-input-23-eb6e6d3309da>:26: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
 trainer = Trainer(
[2268/2268 03:56, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000

Fold 4 Accuracy: 1.0
 Fold 4 Classification Report:
 precision recall f1-score support

	precision	recall	f1-score	support
Positive	1.00	1.00	1.00	728
Negative	1.00	1.00	1.00	1670
Neutral	1.00	1.00	1.00	624
accuracy			1.00	3022
macro avg	1.00	1.00	1.00	3022
weighted avg	1.00	1.00	1.00	3022

Figure 44: Accuracy for $k=4$

```

print("Fold {} Accuracy: {}, fold_accuracy")
print("Fold {} 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))

```

Running Fold 5/5...
 /usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of Transformers
 warnings.warn(
 <ipython-input-23-eb6e6d3309da>:26: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
 trainer = Trainer(
[2268/2268 03:54, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000

Fold 5 Accuracy: 1.0
 Fold 5 Classification Report:
 precision recall f1-score support

	precision	recall	f1-score	support
Positive	1.00	1.00	1.00	749
Negative	1.00	1.00	1.00	1657
Neutral	1.00	1.00	1.00	616
accuracy			1.00	3022
macro avg	1.00	1.00	1.00	3022
weighted avg	1.00	1.00	1.00	3022

Average Accuracy for Validation Dataset across folds: 1.0

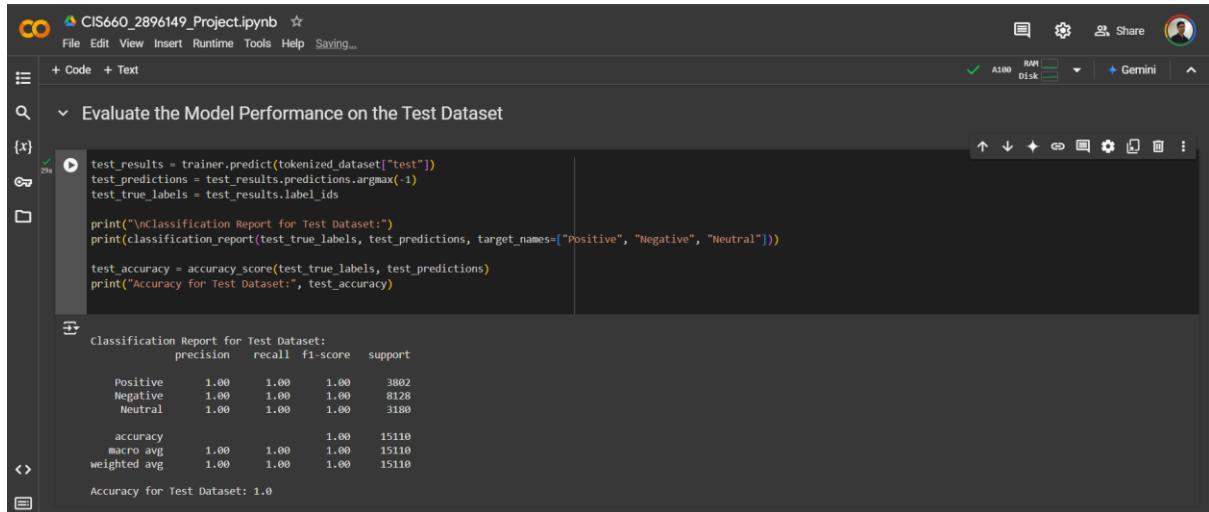
Figure 45: Accuracy for $k=5$ and Average Accuracy of the Model

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.



The screenshot shows a Jupyter Notebook interface with the file name CIS660_2896149_Project.ipynb. The code cell contains Python code to evaluate the BERT model's performance on a test dataset. The output cell displays the classification report, which includes precision, recall, f1-score, and support for Positive, Negative, and Neutral classes, along with overall accuracy and weighted average metrics. The final output shows an accuracy of 1.0 for 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)

Classification Report for Test Dataset:
precision    recall    f1-score   support
Positive      1.00      1.00      1.00      3802
Negative      1.00      1.00      1.00      8128
Neutral       1.00      1.00      1.00      3180

accuracy           1.00      15110
macro avg       1.00      1.00      1.00      15110
weighted avg    1.00      1.00      1.00      15110

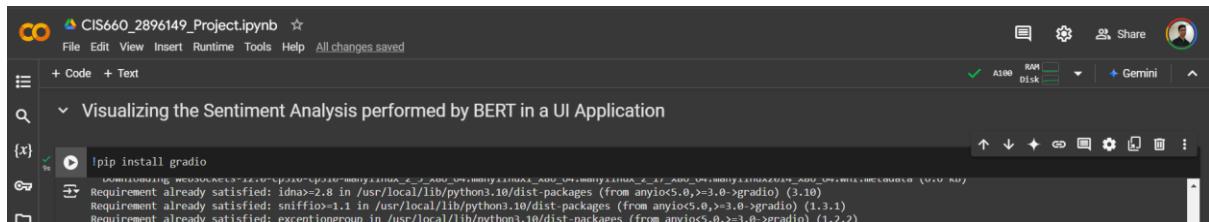
Accuracy for Test Dataset: 1.0
```

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.



The screenshot shows a Jupyter Notebook interface with the file name CIS660_2896149_Project.ipynb. The code cell contains the command !pip install gradio, which installs the gradio library. The output cell shows the successful installation of idna, sniffio, and exceptiongroup packages.

```
!pip install gradio
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from gradio) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from gradio) (1.2.2)
```

Figure 47: Installing the Pre-requisite libraries

```

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)

```

Figure 48: UI Application using gradio

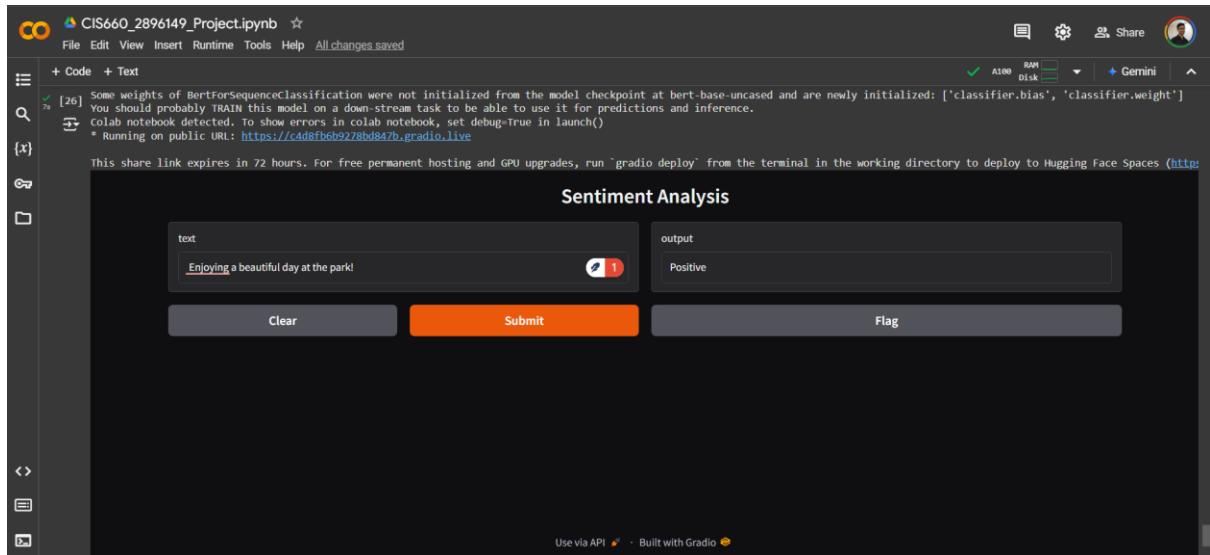


Figure 49: Interface of the application

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.

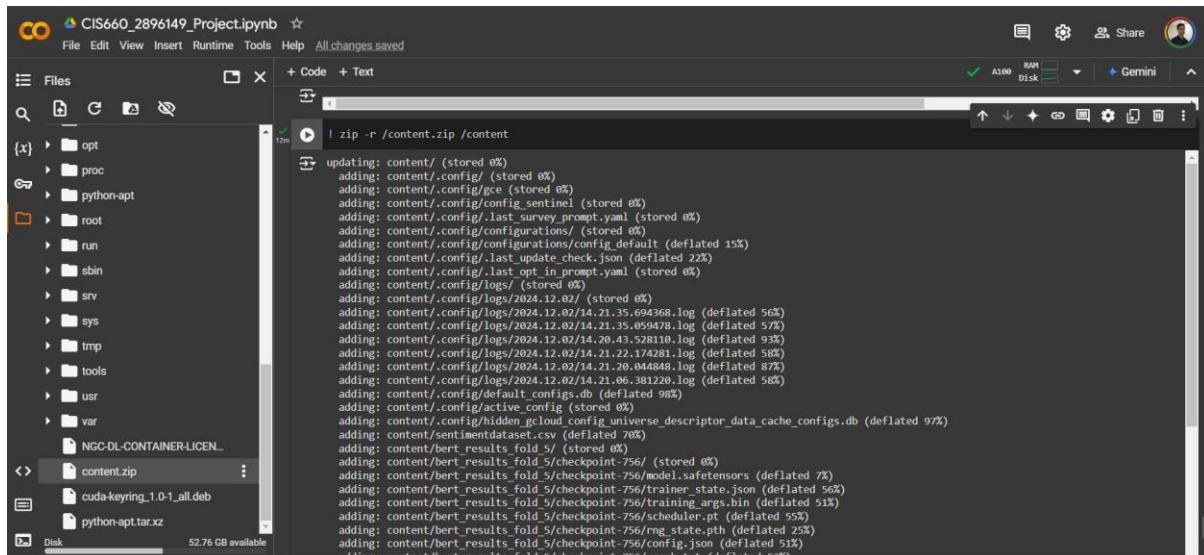
- ***Comparison of Models:***
 - **XGBoost**, while effective and faster for structured data and combined feature sets, lacks to capture deep contextual semantics compared to **BERT**.
 - **BERT** has excellent performance due to its transformer-based architecture, which models bidirectional relationships 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:



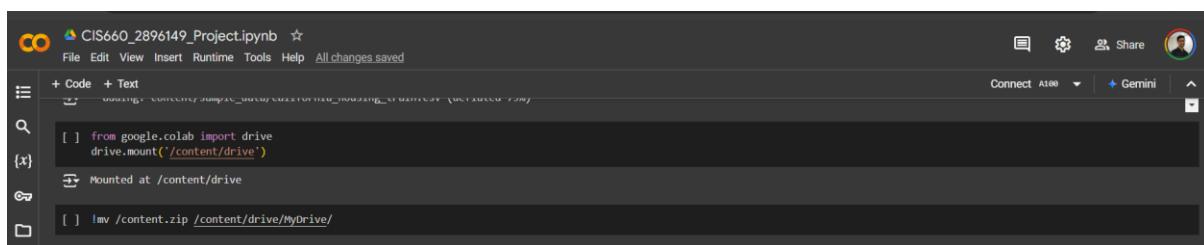
The screenshot shows a Google Colab interface with a terminal window open. The terminal command is:

```
zip -r /content.zip /content
```

The output of the command shows the contents of the zip file being created:

```
updating: content/ (stored 0%)  
adding: content/config/ (stored 0%)  
adding: content/config/gce (stored 0%)  
adding: content/config/config_sentinel (stored 0%)  
adding: content/config/last_wav_prompt.yaml (stored 0%)  
adding: content/config/configurations/ (stored 0%)  
adding: content/config/configurations/config_default (deflated 15%)  
adding: content/config/last_update_check.json (deflated 22%)  
adding: content/config/last_opt_in_prompt.yaml (stored 0%)  
adding: content/config/logs/ (stored 0%)  
adding: content/config/logs/2024.12.02/ (stored 0%)  
adding: content/config/logs/2024.12.02/14.21.35.694368.log (deflated 56%)  
adding: content/config/logs/2024.12.02/14.21.35.959478.log (deflated 57%)  
adding: content/config/logs/2024.12.02/14.20.43.528110.log (deflated 93%)  
adding: content/config/logs/2024.12.02/14.21.22.174281.log (deflated 58%)  
adding: content/config/logs/2024.12.02/14.21.20.044848.log (deflated 87%)  
adding: content/config/logs/2024.12.02/14.21.06.381220.log (deflated 58%)  
adding: content/config/default_configs.db (deflated 96%)  
adding: content/config/active_config (stored 0%)  
adding: content/config/hidden_gcloud_config_universe_descriptor_data_cache_configs.db (deflated 97%)  
adding: content/sentimentdataset.csv (deflated 70%)  
adding: content/bert_results fold 5/ (stored 0%)  
adding: content/bert_results_fold_5/checkpoint-756/ (stored 0%)  
adding: content/bert_results_fold_5/checkpoint-756/model.safetensors (deflated 7%)  
adding: content/bert_results_fold_5/checkpoint-756/trainer_state.json (deflated 56%)  
adding: content/bert_results_fold_5/checkpoint-756/training_args.bin (deflated 51%)  
adding: content/bert_results_fold_5/checkpoint-756/scheduler.pt (deflated 55%)  
adding: content/bert_results_fold_5/checkpoint-756/ng_state.pth (deflated 51%)  
adding: content/bert_results_fold_5/checkpoint-756/config.json (deflated 51%)
```

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



The screenshot shows a Google Colab interface with a terminal window open. The terminal commands are:

```
[ ] from google.colab import drive  
[ ] drive.mount('/content/drive')  
[ ] Mounted at /content/drive  
[ ] !mv /content.zip /content/drive/mydrive
```

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 tag 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/

```