Title: "Sentiment Analysis for Social Media Data"

- Purpose: Analyze public sentiment on social media platforms.
- Scope: Focused on understanding emotions expressed in subreddit posts.

"Technical Setup and Libraries"

```
from pprint import pprint
import pandas as pd
import numpy as np
import praw
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer as sa
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import gensim.downloader as api
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

"Data Sourcing from Reddit"

```
user_agent = "Data 1.0"
reddit = praw.Reddit(
    client_id="eMB6WKPL5okjyK0HEp-jLA",
    client_secret="nI8rPm-U8OVJGbhxx5EplOsZlAEsiA",
    user_agent=user_agent
)
```

"Data Sourcing from Reddit"

praw for automated data extraction.

Target subreddits: "Worldnews", "sad", "Dreams", "happy".

Objective: Capture diverse sentiment across topics.

Extracting and Limiting headers

this function is to extract and return the remaining rate limit value from the HTTP response headers.

```
titles = set() #to restrict repetition of titles

def extract_remaining_rate_limit(response_headers):  #for esuring there is no problem with extracting titles
    try:
        #extract remaining rate limit value from response headers
        remaining_rate_limit = float(response_headers.get("x-ratelimit-remaining", "0").split(',')[0])
        return remaining_rate_limit
    except ValueError:
        return 0  # Return 0 if there's an issue with the rate limit header
```

"Data Uniqueness and Integrity"

– Use of a set to store unique titles.
– Rate limit management for ethical scraping.

```
#extracting titles from the subreddits
for submission in reddit.subreddit('worldnews').new(limit= None):
    titles.add(submission.title)
    remaining_limit = extract_remaining_rate_limit(reddit.auth.limits)

for submission in reddit.subreddit('sad').new(limit=None):
    titles.add(submission.title)
    remaining_limit = extract_remaining_rate_limit(reddit.auth.limits)

for submission in reddit.subreddit('Dreams').new(limit= None):
    titles.add(submission.title)
    remaining_limit = extract_remaining_rate_limit(reddit.auth.limits)

for submission in reddit.subreddit('happy').new(limit= None):
    titles.add(submission.title)
    remaining_limit = extract_remaining_rate_limit(reddit.auth.limits)

print(f"Amount of lines generated: {len(titles)}")
```

```
Amount of lines generated: 3323
```

Sentiment Analysis on titles

This code performs sentiment analysis on a list of titles using the SentimentIntensityAnalyzer(Sia).

```
sia = sa() #creating an analyser
sentimented_sentences = []

#doing a sentiment classification on the data extracted
for i in titles:
    score = sia.polarity_scores(i) #making a dictionary
    score['Post'] = i
    sentimented_sentences.append(score)

pprint(sentimented_sentences[:3], width = 150)
```

```
[{'Post': "For the first time in weeks I'm excited for the future", 'compound': 0.34, 'neg': 0.0, 'neu': 0.806, 'pos': 0.194},
 {'Post': 'French court orders return of deported Uzbek national in rebuke to interior minister',
  'compound': 0.0,
  'neg': 0.0,
  'neu': 1.0,
  'pos': 0.0},
 {'Post': 'Thousands of tons of dead sardines wash ashore in northern Japan', 'compound': -0.6486, 'neg': 0.301, 'neu': 0.699, 'pos': 0.0}]
```

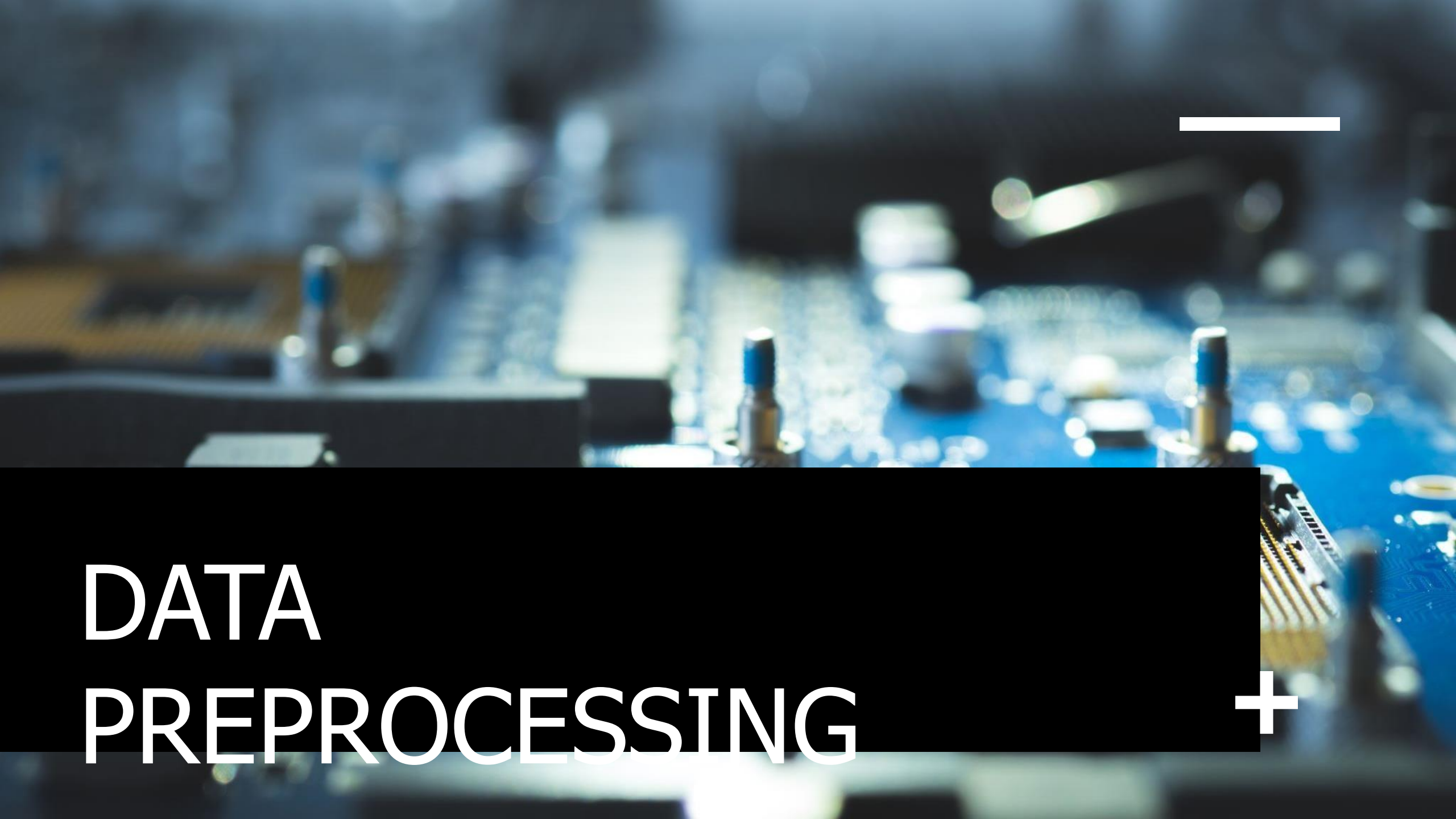
Labelling the sentiment

- **a sentiment value:**
- **1 for positive,**
- **-1 for negative**
- **0 for neutral**

Giving the data in the dataframe a sentiment value: 1 for positive, -1 for negative and 0 for neutral

```
df_sentimented["Sen Label"] = 0
df_sentimented.loc[df_sentimented['compound'] > 0.2, "Sen Label"] = 1
df_sentimented.loc[df_sentimented['compound'] <  -0.2, "Sen Label"] = -1
df_sentimented.head()
```

	neg	neu	pos	compound	Post	Sen Label
0	0.000	0.806	0.194	0.3400	For the first time in weeks I'm excited for th...	1
1	0.000	1.000	0.000	0.0000	French court orders return of deported Uzbek n...	0
2	0.301	0.699	0.000	-0.6486	Thousands of tons of dead sardines wash ashore...	-1
3	0.439	0.561	0.000	-0.6908	What's the fastest way to kill yourself?	-1
4	0.000	0.600	0.400	0.2500	Hair shaving in dream	1



DATA PREPROCESSING

Data Pre-Processing

1. Removing URL's
2. Removing HTML
3. Handling and Removing Emojis
4. Removing #Tags
5. Removing Stop words
6. Tokenizing
7. Lemmatizing

```
def r_urls(text): #removing urls function
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

def r_html(text): #remving html function
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)

# Emoji expression code
emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           "]+", flags=re.UNICODE)


def r_hashtags(text): #removing hashtags function
    return re.sub(r'#\w+', '', text)

def r_emojis(text): #removing emojis function
    return emoji_pattern.sub(r'', text)

#This function is for removing: URLs,HTML tags,hashtags,emojis from the text
def preprocess_text(text):
    text = text.lower()
    #text = r_slang(text, slang_dict)
    text = r_urls(text)  # Remove
    text = r_html(text)  # Remove
    text = r_hashtags(text)  # Remove
    text = r_emojis(text)  # Remove
    tokens = word_tokenize(text)     # Tokenization

    # Remove stopwords and non-alphanumeric characters
    clean_tokens = []
```

Processed Comments:

	Post	Sen Label	Processed_Post
0	For the first time in weeks I'm excited for th...	1	first time week excited future
1	French court orders return of deported Uzbek n...	0	french court order return deported uzbek natio...
2	Thousands of tons of dead sardines wash ashore...	-1	thousand ton dead sardine wash ashore northern...
3	What's the fastest way to kill yourself?	-1	fastest way kill
4	Hair shaving in dream	1	hair shaving dream

FEATURE EXTRACTION

This code processes text data from a DataFrame, converts it into word frequencies using Bag-of-Words, and then identifies and displays the top 10 most frequent words along with their frequencies.

	word	freq
1529	dream	596
2219	happy	249
5241	year	205
1289	day	137
2782	life	121
1974	friend	116
2792	like	111
4766	today	107
1842	feel	106
1887	first	105

```
processed_posts = df_sentimented_updated['Processed_Post']

# Create a Bag-of-Words vectorizer
vectorizer = CountVectorizer()

# Fit and transform your preprocessed posts
bow_features = vectorizer.fit_transform(processed_posts)
word_freq = np.sum(bow_features.toarray(), axis=0)
#making a dataframe for unique words and their frequency and give the top 10
word_freq_df = pd.DataFrame({'word': vectorizer.get_feature_names_out(), 'freq': word_freq}).nlargest(10, 'freq')

word_freq_df.head(10)
```

Word2Vec Embedding and Similarity Calculation

- Loads Word2Vec model "glove-twitter-25."
- Defines function to vectorize sentences using Word2Vec.
- Applies function to 'Processed_Post' column for mean vectors.
- Computes cosine similarity matrix.

```python
# Load pre-trained Word2Vec embeddings
word2vec_model = api.load("glove-twitter-25")

# Function to vectorize a sentence into an embedding
def document_vector(doc):
    words_in_doc = doc.split()
    valid_words = []
    for word in words_in_doc:
        if word in word2vec_model.key_to_index:  # Checking if the word is in t
            valid_words.append(word)  # If the word is valid append it

    # Check if there are valid words in the document and return their vector me
    if valid_words:
        return np.mean(word2vec_model[valid_words], axis=0)
    else:
        return np.zeros(word2vec_model.vector_size)

# Applying the function over here
word2vec_features = df_sentimented_updated['Processed_Post'].apply(document_vec
word2vec_f = np.array([document_vector(post) for post in processed_posts])
cosine_sim_word2vec = cosine_similarity(word2vec_f)
print("Cosine Similarity (Word2Vec):")
pprint(cosine_sim_word2vec)

word2vec_features.to_csv("processed_featuredata.csv", index=False)
```

```
Cosine Similarity (Word2Vec):
array([[1.        , 0.72403318, 0.76821133, ..., 0.92300521, 0.79844115,
```

Model Selection and Training

- Support Vector Classification (SVC) is a type of computer program that learns to categorize things into different groups.
- It's like a smart assistant that, given examples of items belonging to different categories, figures out the best way to draw a line or boundary between those categories.

```
# Initialize the SVM model
svm_model = SVC()

# Train the model with the training data
svm_model.fit(X_train, y_train)
```

▾ SVC

SVC()

Training and Testing to get Performance Metrics

- The model demonstrates high performance on the training set (indicating a good understanding of the training data).
- On the testing set, the model's performance is reasonably good but shows a decline compared to the training set.
- The decline suggests that the model might be overfitting to the training data and may need further tuning for better generalization to new, unseen data.

```
[403…  #Make predictions on the training set and test set
       y_train_pred = svm_model.predict(X_train)
       y_test_pred = svm_model.predict(X_test)

       #Evaluate the performance
       print("Training Performance:")
       print(classification_report(y_train, y_train_pred))

       print("Testing Performance:")
       print(classification_report(y_test, y_test_pred))
```

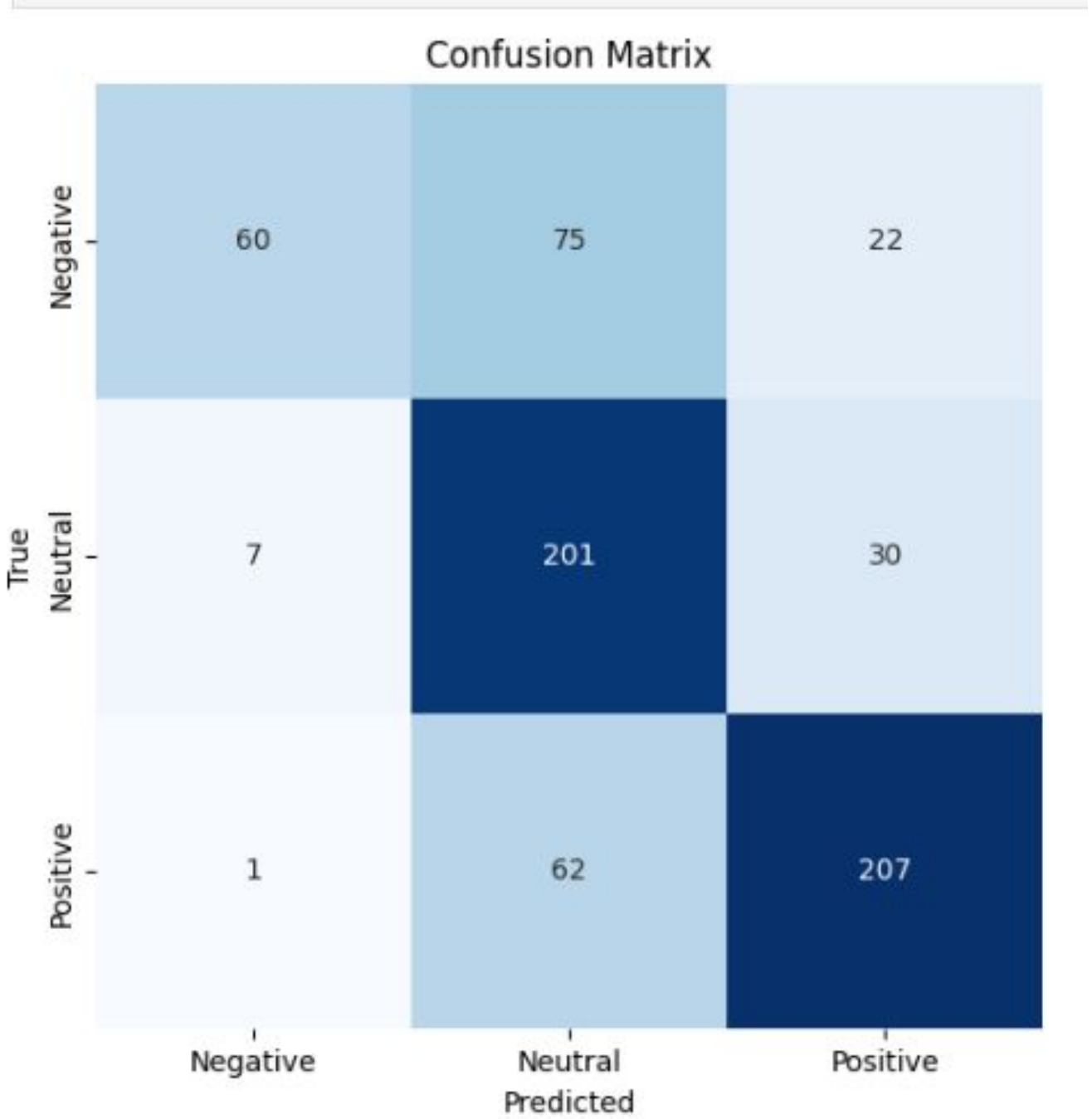
```
Training Performance:
              precision    recall  f1-score   support

          -1       0.99      0.98      0.99       653
           0       0.99      0.98      0.98       900
           1       0.98      1.00      0.99      1105

    accuracy                           0.99      2658
   macro avg       0.99      0.98      0.99      2658
weighted avg       0.99      0.99      0.99      2658

Testing Performance:
              precision    recall  f1-score   support

          -1       0.88      0.38      0.53       157
           0       0.59      0.84      0.70       238
           1       0.80      0.77      0.78       270

    accuracy                           0.70       665
   macro avg       0.76      0.66      0.67       665
weighted avg       0.75      0.70      0.69       665
```

Confusion Matrix

- This code creates a confusion matrix, a table showing how well a model predicts different classes.
- The matrix is visualized as a heatmap using colors.
- The x-axis represents predicted classes.
- The y-axis represents true classes, and the numbers inside the heatmap indicate how many predictions fall into each category.

```
classes = ['Negative', 'Neutral', 'Positive']
confusion = confusion_matrix(y_test, y_test_pred)
# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False, square=True, xticklabels=
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Confusion Matrix



Thankyou