ANALYZING USER SENTIMENT USING NAIVE BAYES

Natasya Aulia Putri

INTRODUCTION

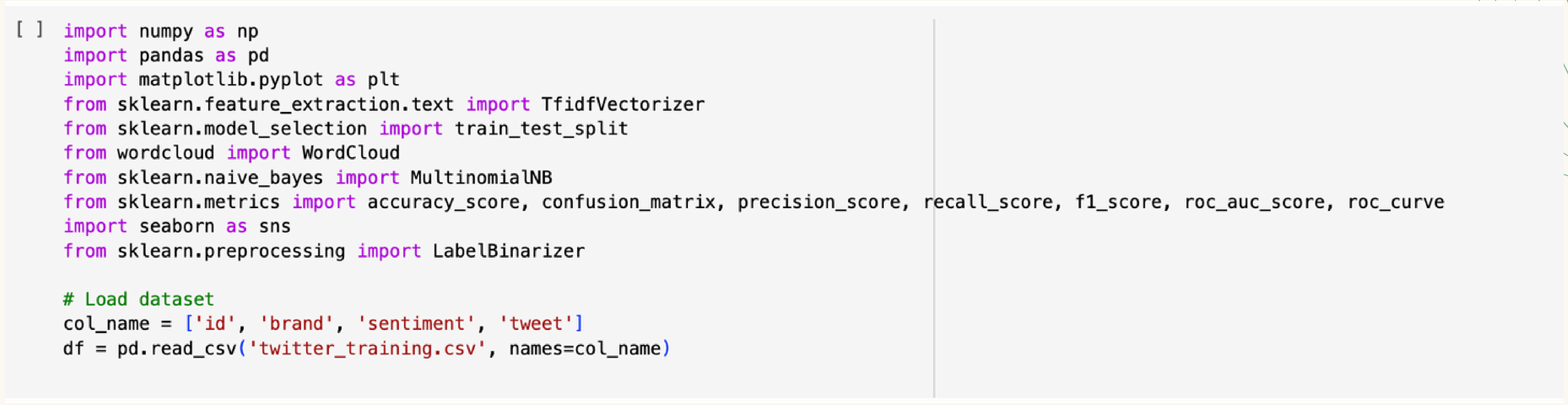
01 OVERVIEW

This project focuses on sentiment analysis of tweets using the Naive Bayes classifier. The goal is to classify tweets into three categories: Positive, Neutral, and Negative sentiment. The dataset contains information about brand, sentiment, and tweet content, which is analyzed to derive insights about customer opinions and brand perceptions on social media.

02 KEY STEPS

- Import Data and Libraries
- Cleaning Data
- Visualizing Data by any criteria
- Prepocessing Data and Build Model
- Evaluation

IMPORT DATA AND LIBRARIES

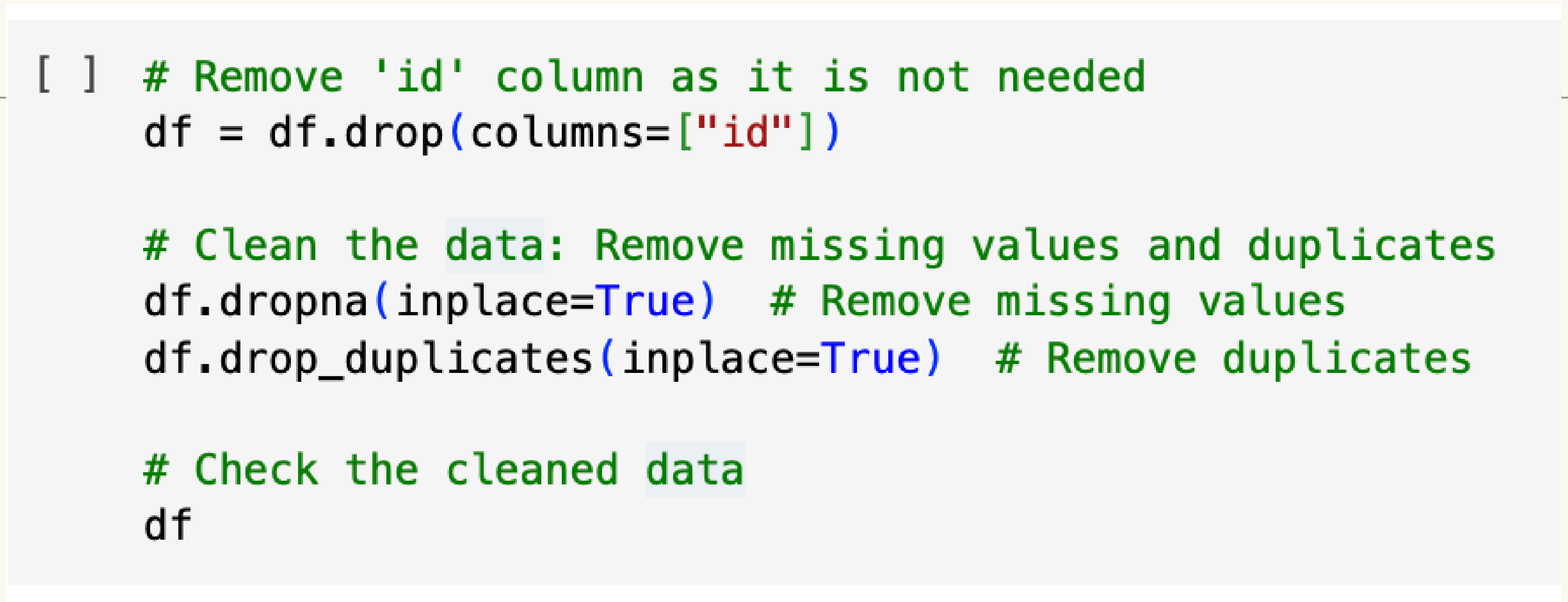


```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.model_selection import train_test_split
    from wordcloud import WordCloud
    from sklearn.naive_bayes import MultinomialNB
    from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
    import seaborn as sns
    from sklearn.preprocessing import LabelBinarizer

    # Load dataset
    col_name = ['id', 'brand', 'sentiment', 'tweet']
    df = pd.read_csv('twitter_training.csv', names=col_name)
```

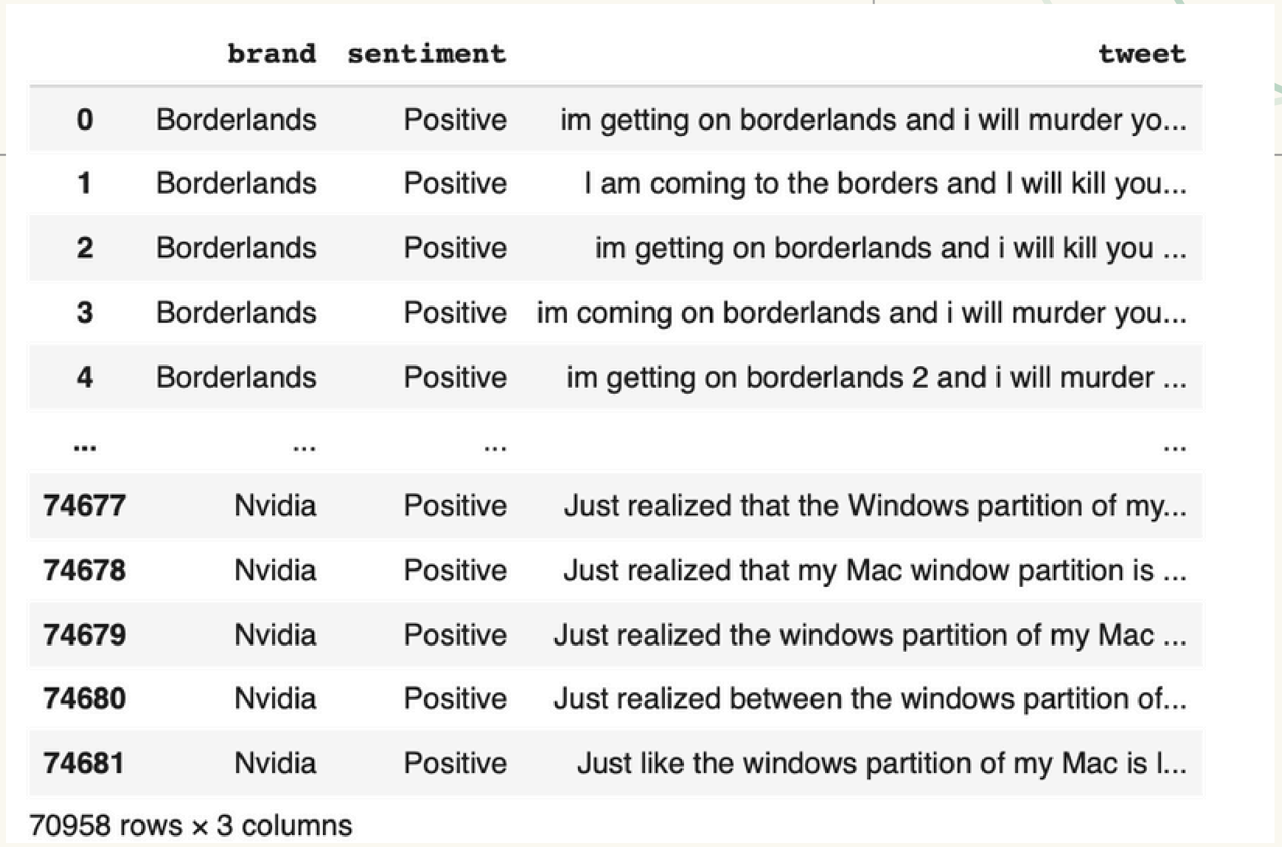
This code is the first step in analyzing a Twitter dataset for sentiment classification. It loads the data from a file called twitter_training.csv, which includes details like the tweet's sentiment (positive, neutral, or negative), the brand it's about, and the tweet's text. The dataset is organized into clear columns so we can easily work with it later.

The goal is to clean and explore the data, then use it to train a machine learning model to classify tweet sentiments and uncover trends related to different brands.

CLEANING DATA



```
[ ]  # Remove 'id' column as it is not needed
     df = df.drop(columns=["id"])

     # Clean the data: Remove missing values and duplicates
     df.dropna(inplace=True)  # Remove missing values
     df.drop_duplicates(inplace=True)  # Remove duplicates

     # Check the cleaned data
     df
```



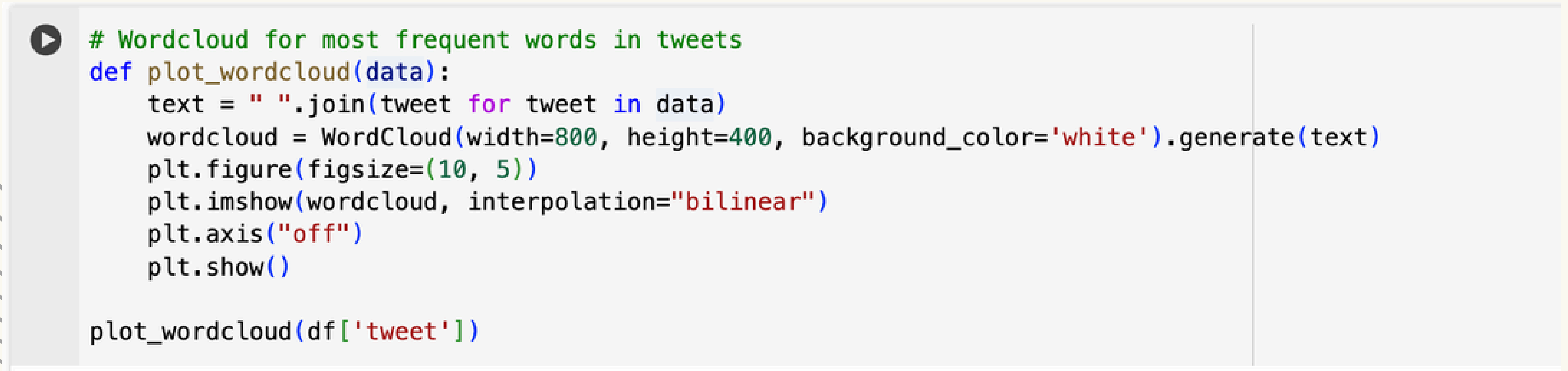
	brand	sentiment	tweet
0	Borderlands	Positive	im getting on borderlands and i will murder yo...
1	Borderlands	Positive	I am coming to the borders and I will kill you...
2	Borderlands	Positive	im getting on borderlands and i will kill you ...
3	Borderlands	Positive	im coming on borderlands and i will murder you...
4	Borderlands	Positive	im getting on borderlands 2 and i will murder ...
...	...	...	...
74677	Nvidia	Positive	Just realized that the Windows partition of my...
74678	Nvidia	Positive	Just realized that my Mac window partition is ...
74679	Nvidia	Positive	Just realized the windows partition of my Mac ...
74680	Nvidia	Positive	Just realized between the windows partition of...
74681	Nvidia	Positive	Just like the windows partition of my Mac is l...

70958 rows × 3 columns

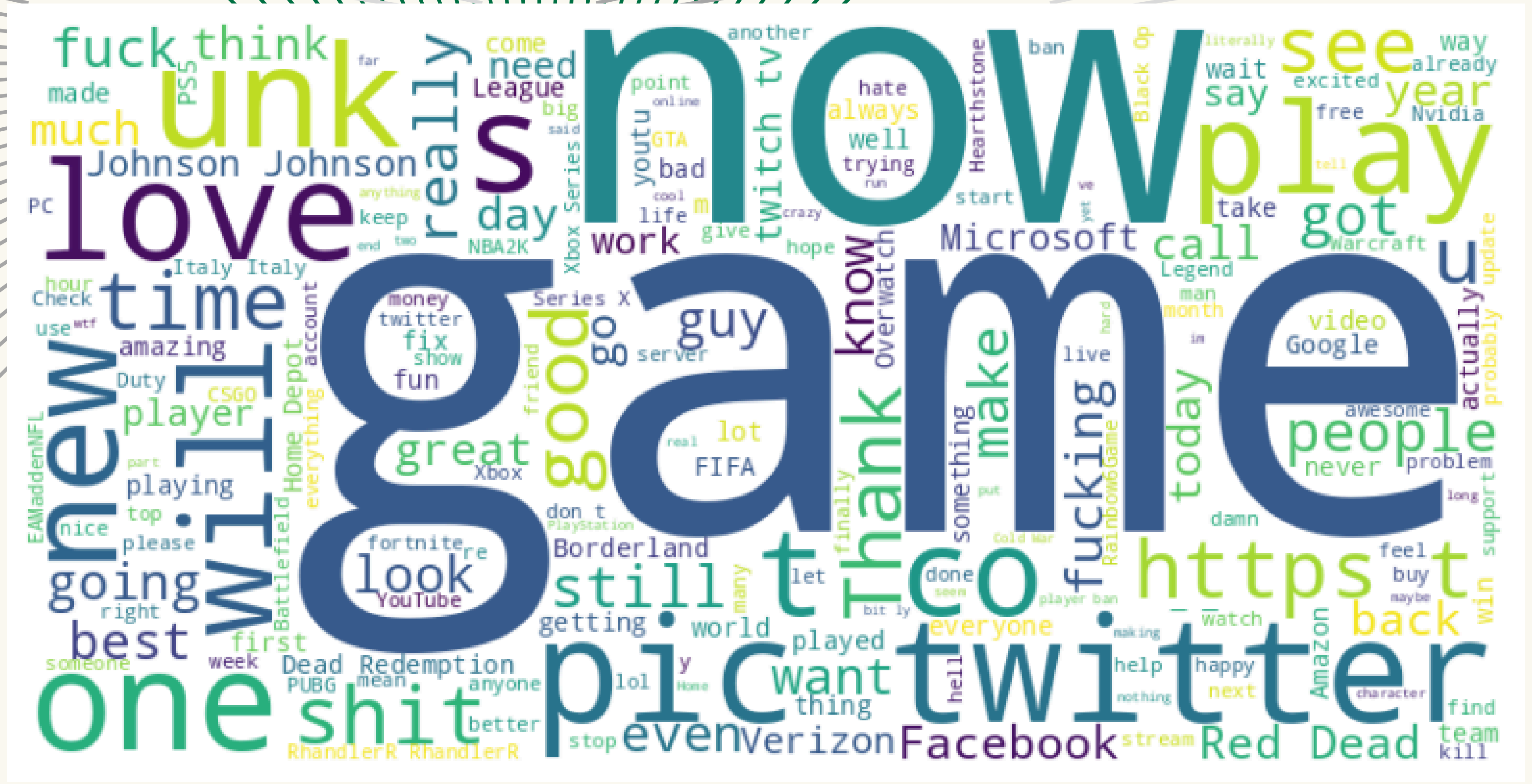
The dataset is cleaned to ensure it’s ready for analysis. First, the id column is removed since it doesn’t provide meaningful information for sentiment analysis. Next, any rows with missing values are dropped to avoid potential issues during analysis. Duplicate rows are also removed to ensure the data remains accurate and unbiased. After cleaning, the dataset is inspected to confirm it is properly structured and ready for further processing.

DATA VISUALIZATION

WORD CLOUD VISUALIZATION



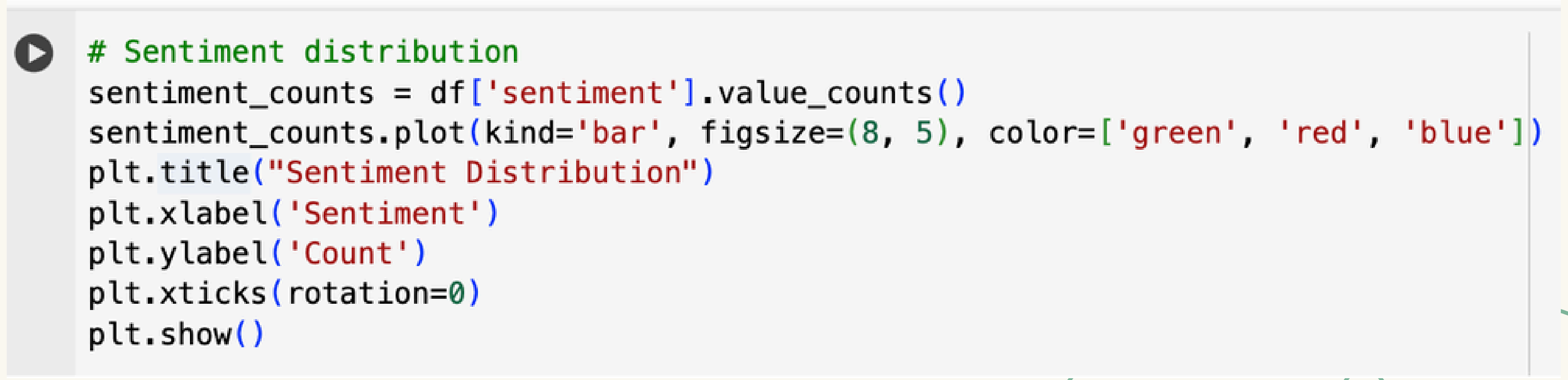
```python
# Wordcloud for most frequent words in tweets
def plot_wordcloud(data):
    text = " ".join(tweet for tweet in data)
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()

plot_wordcloud(df['tweet'])
```



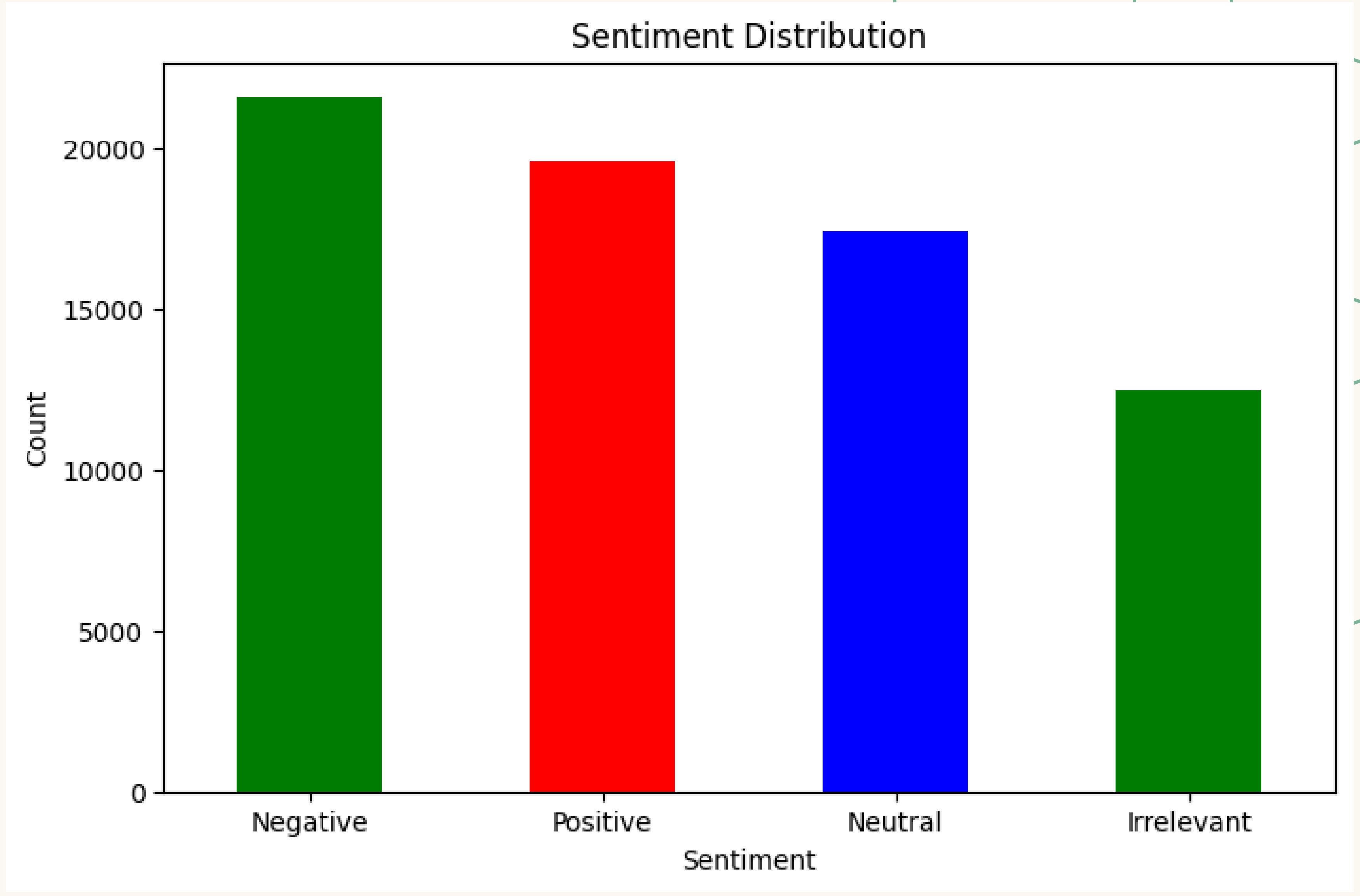
This section creates a word cloud to visualize the most frequently used words in the tweets. All the tweets are combined into a single text block, and a word cloud is generated to highlight the most common words. Larger words in the visualization represent higher frequencies, providing a quick and intuitive way to identify key themes or trends in the dataset.

SENTIMEN DISTRIBUTION

The sentiment distribution is displayed in a bar chart to understand the overall sentiment breakdown in the dataset. Each sentiment category (positive, neutral, negative) is counted, and the counts are plotted with distinct colors: green for positive, red for negative, and blue for neutral. This visualization provides a clear overview of the sentiment balance, helping identify dominant trends in the dataset.

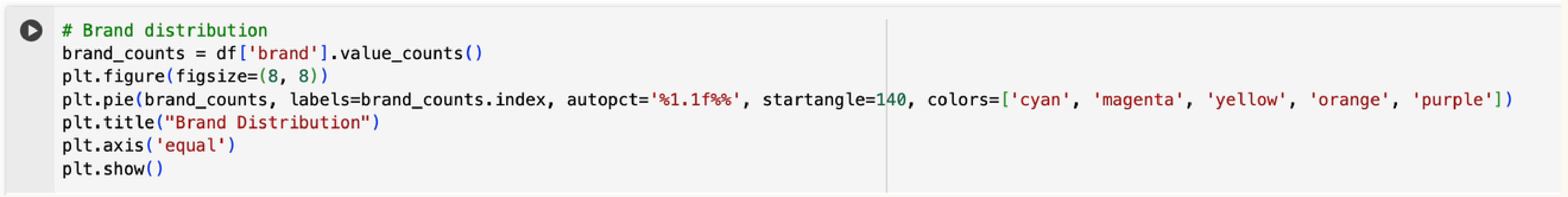


```
# Sentiment distribution
sentiment_counts = df['sentiment'].value_counts()
sentiment_counts.plot(kind='bar', figsize=(8, 5), color=['green', 'red', 'blue'])
plt.title("Sentiment Distribution")
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```

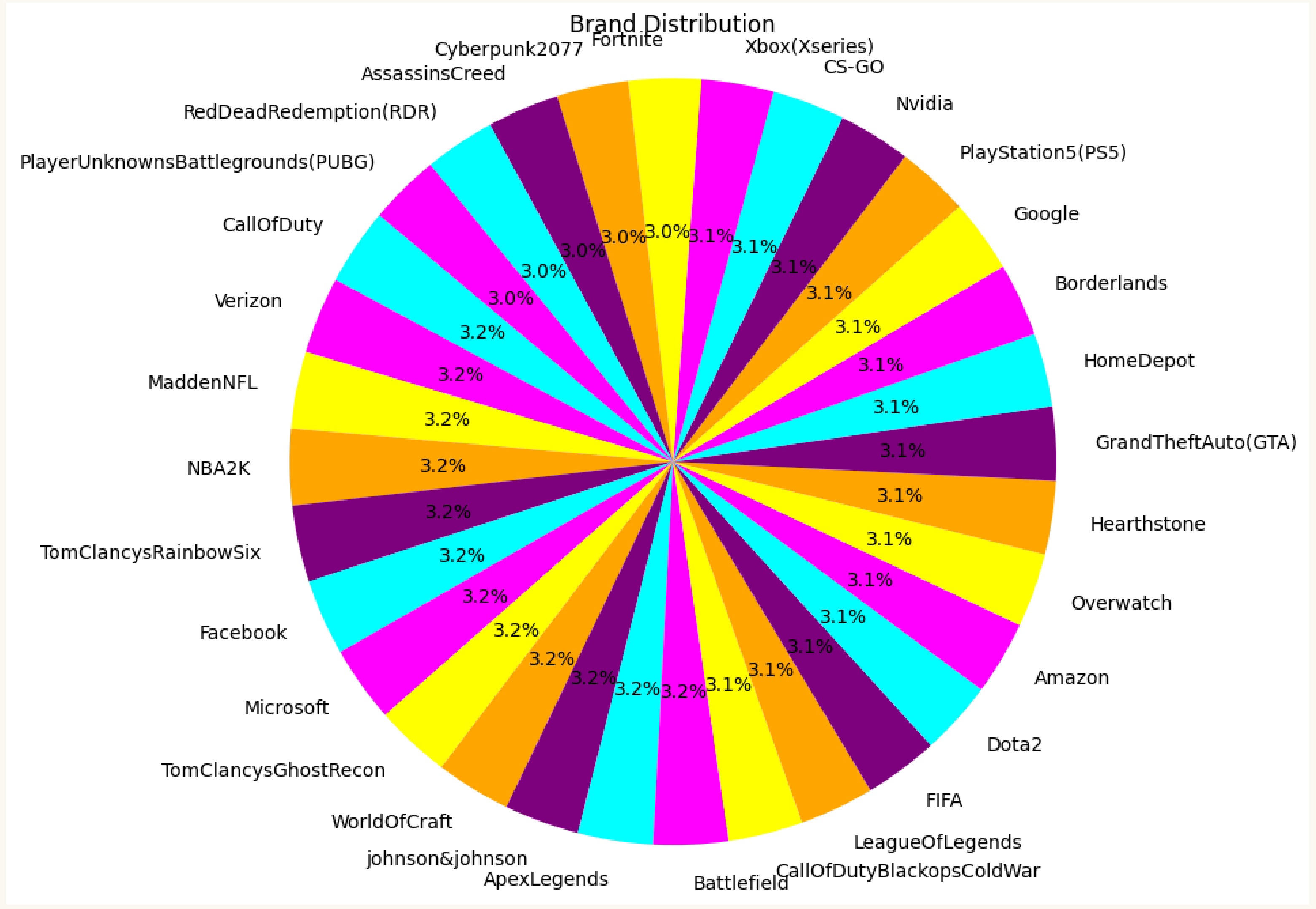


BRAND DISTRIBUTION

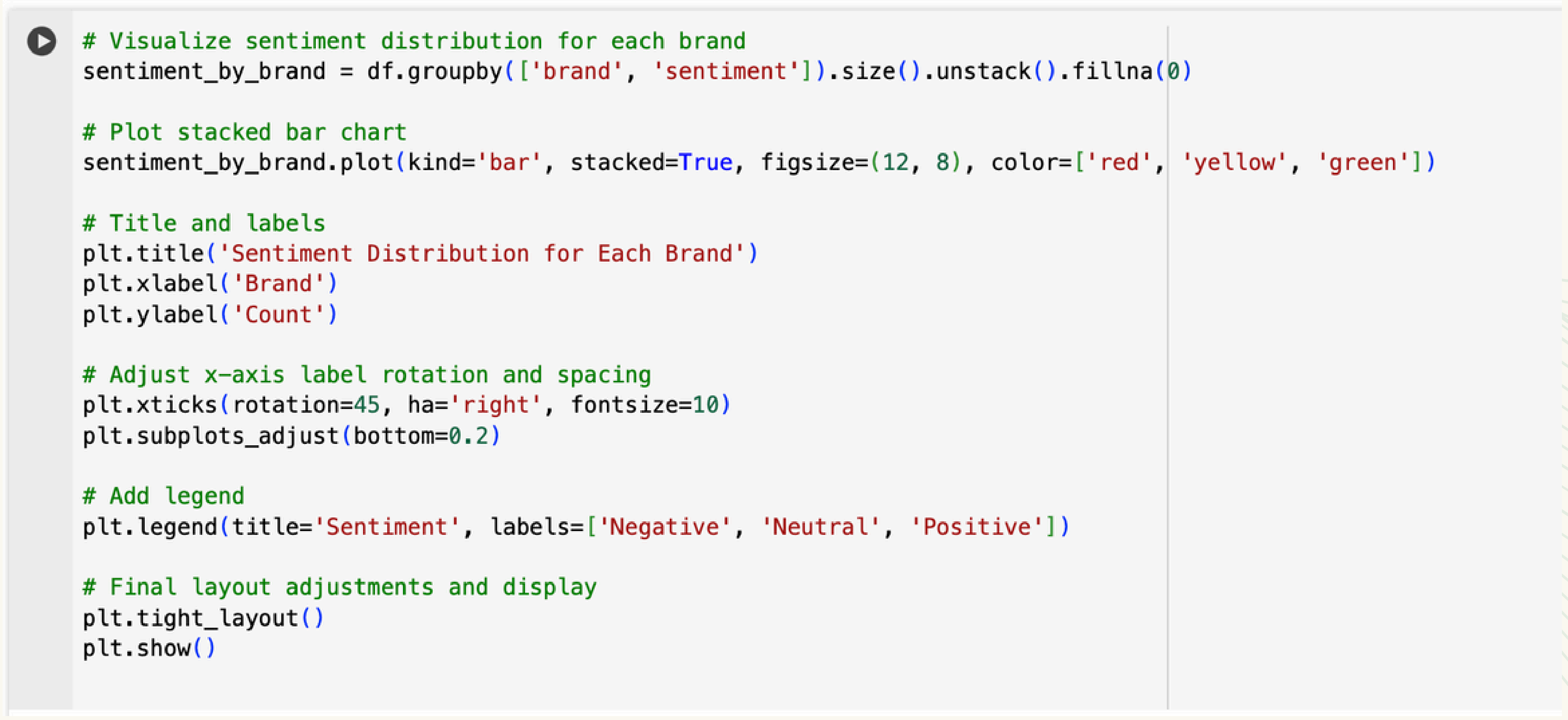
This section creates a pie chart to illustrate how the tweets are distributed across various brands. The chart displays each brand’s percentage share, with distinct colors used for clarity. This visualization makes it easy to see which brands are most frequently mentioned, offering insights into brand presence and engagement within the dataset.



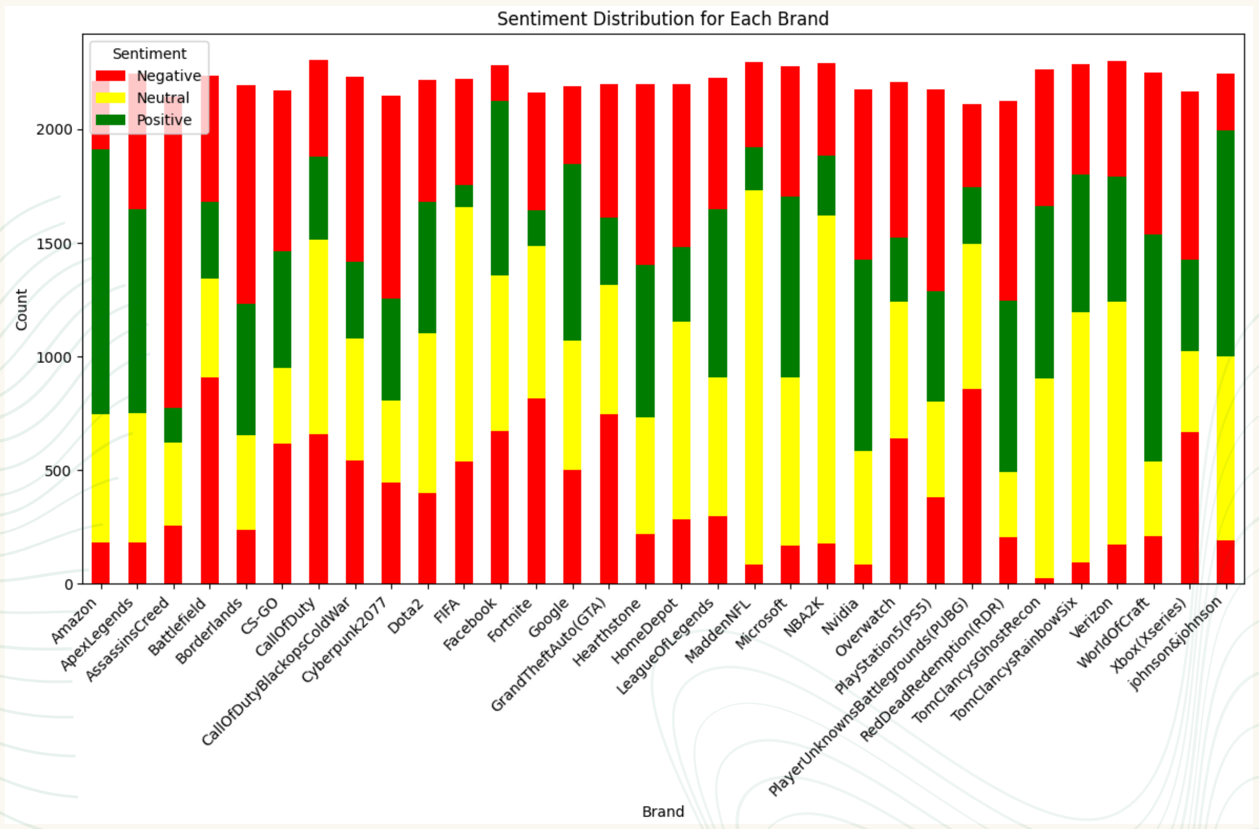
```python
# Brand distribution
brand_counts = df['brand'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(brand_counts, labels=brand_counts.index, autopct='%1.1f%%', startangle=140, colors=['cyan', 'magenta', 'yellow', 'orange', 'purple'])
plt.title("Brand Distribution")
plt.axis('equal')
plt.show()
```



RENCANA TARGET

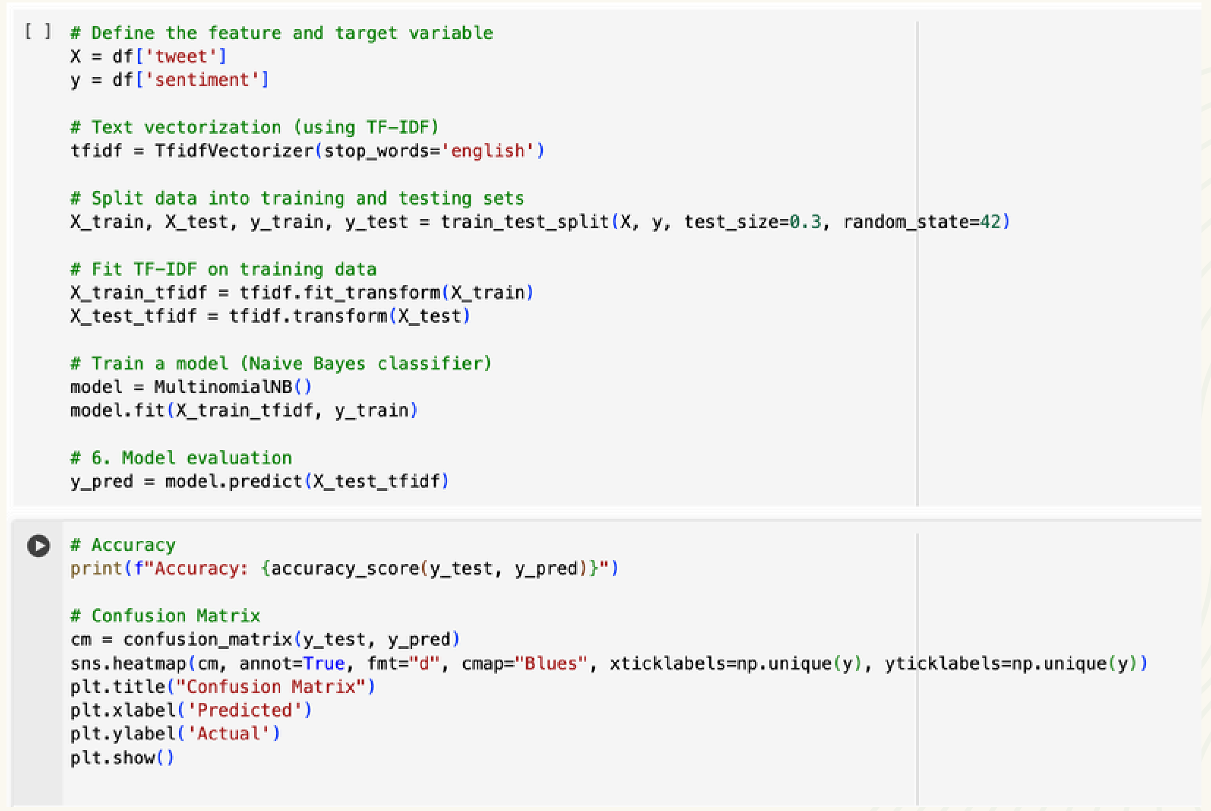


```
# Visualize sentiment distribution for each brand
sentiment_by_brand = df.groupby(['brand', 'sentiment']).size().unstack().fillna(0)

# Plot stacked bar chart
sentiment_by_brand.plot(kind='bar', stacked=True, figsize=(12, 8), color=['red', 'yellow', 'green'])

# Title and labels
plt.title('Sentiment Distribution for Each Brand')
plt.xlabel('Brand')
plt.ylabel('Count')

# Adjust x-axis label rotation and spacing
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.subplots_adjust(bottom=0.2)

# Add legend
plt.legend(title='Sentiment', labels=['Negative', 'Neutral', 'Positive'])

# Final layout adjustments and display
plt.tight_layout()
plt.show()
```



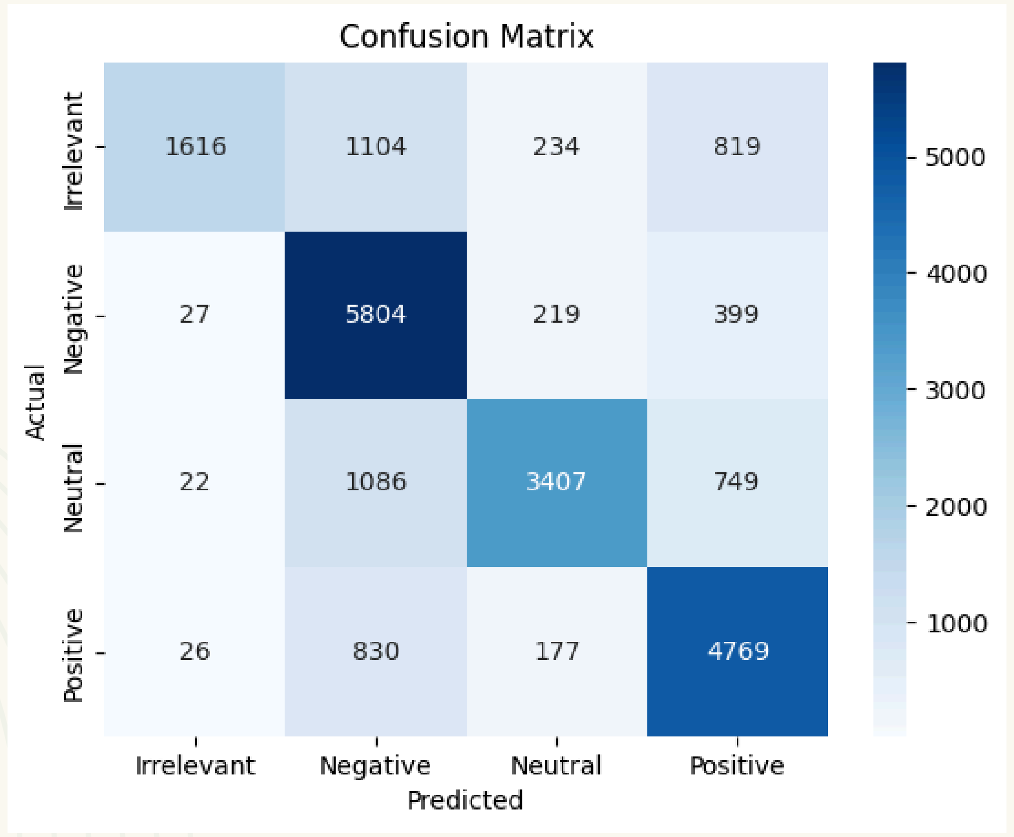
This section of the code visualizes the sentiment distribution for each brand in the dataset using a stacked bar chart. First, the data is grouped by brand and sentiment, counting how many tweets fall under each sentiment category for each brand. A stacked bar chart is then plotted, where each bar represents a brand and the sections within the bars represent the counts of negative, neutral, and positive sentiments. The chart includes a legend to clearly indicate what each color represents and labels for both the x-axis (brands) and y-axis (counts). The x-axis labels are rotated for better readability, and the layout is adjusted to ensure a clean display. This visualization helps to analyze how different brands are perceived in terms of customer sentiment.

CONFUSION MATRIX



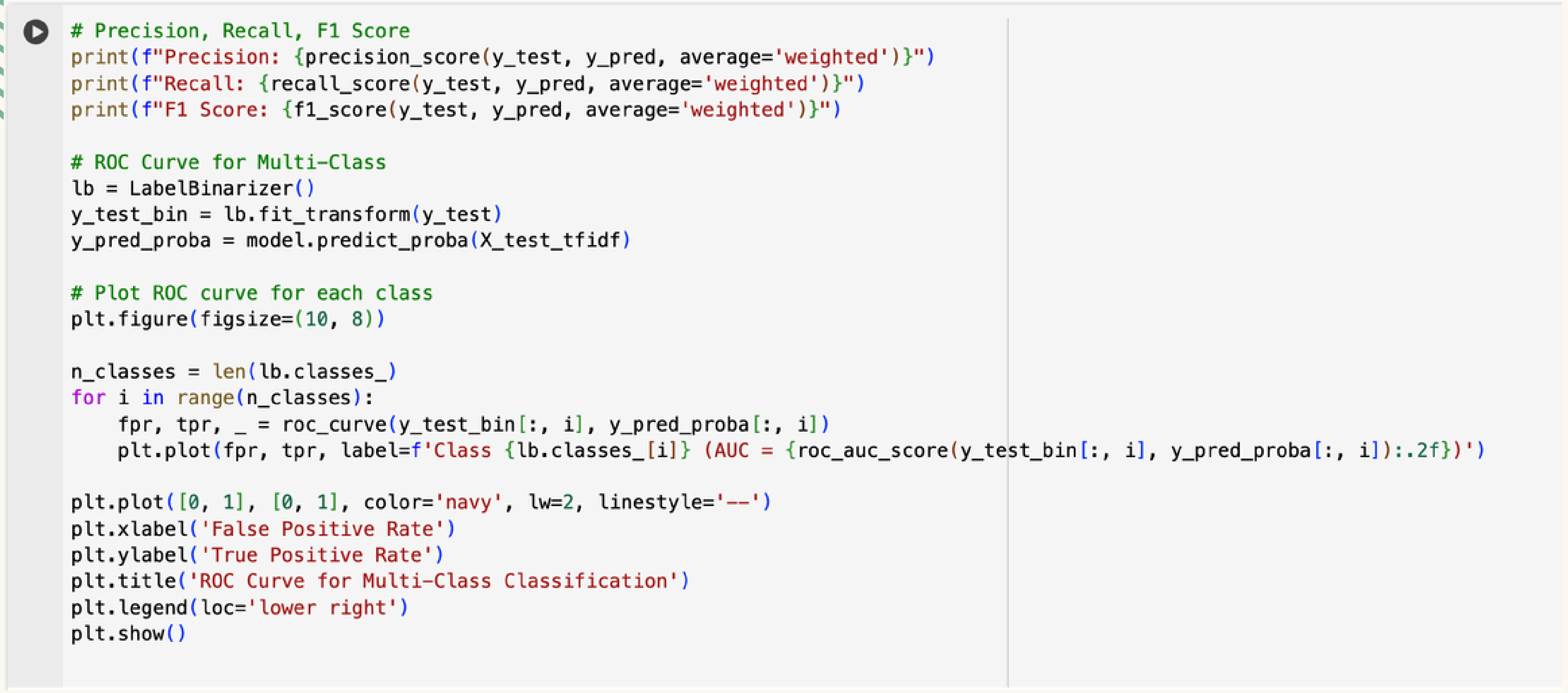
```
[ ] # Define the feature and target variable
    X = df['tweet']
    y = df['sentiment']

    # Text vectorization (using TF-IDF)
    tfidf = TfidfVectorizer(stop_words='english')

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Fit TF-IDF on training data
    X_train_tfidf = tfidf.fit_transform(X_train)
    X_test_tfidf = tfidf.transform(X_test)

    # Train a model (Naive Bayes classifier)
    model = MultinomialNB()
    model.fit(X_train_tfidf, y_train)

    # 6. Model evaluation
    y_pred = model.predict(X_test_tfidf)
```

```
    # Accuracy
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y), yticklabels=np.unique(y))
    plt.title("Confusion Matrix")
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```



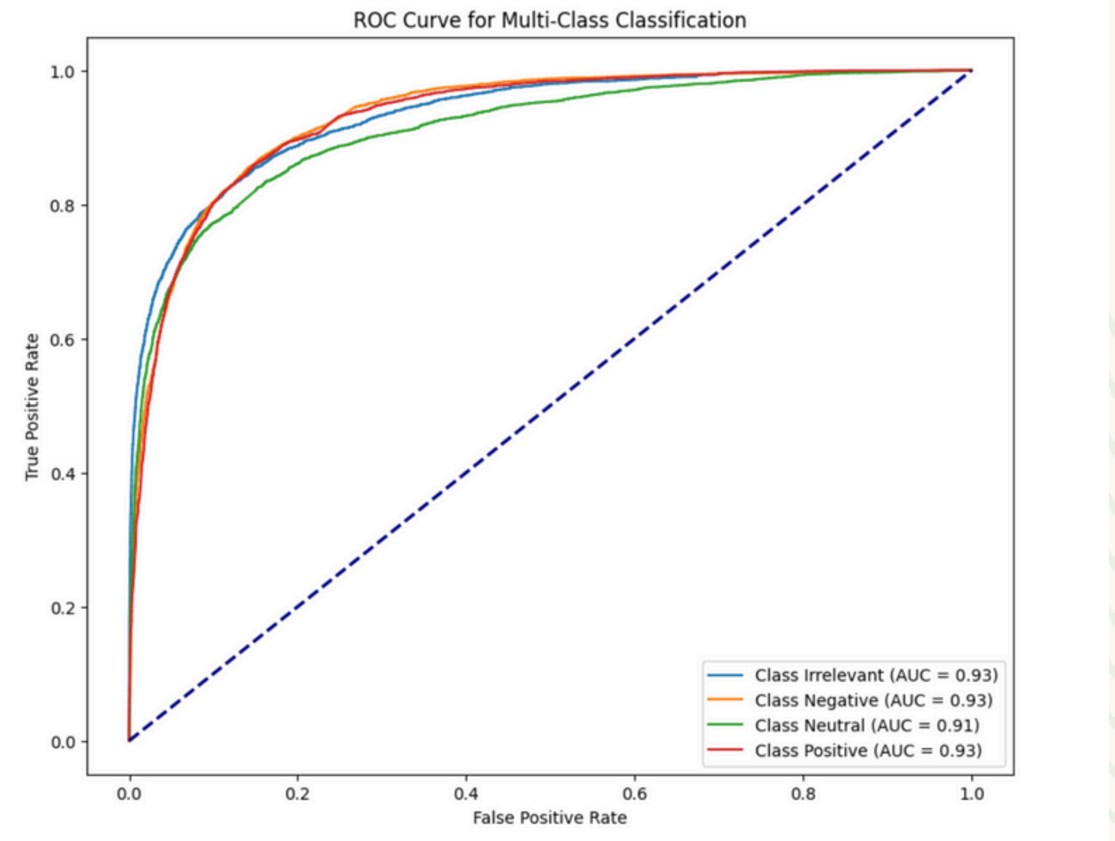
The analysis shows that the Naive Bayes classifier achieves an accuracy of 73.26%, demonstrating a solid ability to predict sentiments in the dataset. The confusion matrix highlights strong performance in identifying Positive (4769) and Negative (5804) sentiments, which dominate the dataset. However, Neutral (3407) sentiments are more challenging to differentiate, possibly due to overlaps with other categories. Despite being less common, the model also handles the Irrelevant (1616) category fairly well. While the results are promising, there's still room to improve the model's ability to handle ambiguity, especially with Neutral and Irrelevant sentiments.

MODEL EVALUATION AND ROC CURVE



```
# Precision, Recall, F1 Score
print(f"Precision: {precision_score(y_test, y_pred, average='weighted')}")
print(f"Recall: {recall_score(y_test, y_pred, average='weighted')}")
print(f"F1 Score: {f1_score(y_test, y_pred, average='weighted')}")

# ROC Curve for Multi-Class
lb = LabelBinarizer()
y_test_bin = lb.fit_transform(y_test)
y_pred_proba = model.predict_proba(X_test_tfidf)

# Plot ROC curve for each class
plt.figure(figsize=(10, 8))

n_classes = len(lb.classes_)
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
    plt.plot(fpr, tpr, label=f'Class {lb.classes_[i]} (AUC = {roc_auc_score(y_test_bin[:, i], y_pred_proba[:, i]):.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class Classification')
plt.legend(loc='lower right')
plt.show()
```

```
Precision: 0.7702817115837177
Recall: 0.732619316046599
F1 Score: 0.723572699476891
```



The Naive Bayes model demonstrates solid performance with Precision (77.03%), Recall (73.26%), and F1-Score (72.36%), reflecting a good balance between accuracy and class detection capability. The AUC analysis further supports these results, with excellent identification for Irrelevant (0.93), Negative (0.93), and Positive (0.93) classes, while the Neutral (0.91) class poses a slight challenge, likely due to overlaps with other sentiments. Overall, the model performs consistently well and can be improved to address ambiguity in the Neutral class.

THANK YOU

Natasya Aulia Putri