NEXUS INFO

Sentiment Analysis Project

1. Introduction

Sentiment analysis is a technique used to determine the sentiment expressed in textual data. This project aims to build a predictive model to classify text as positive, negative, or neutral based on its sentiment.

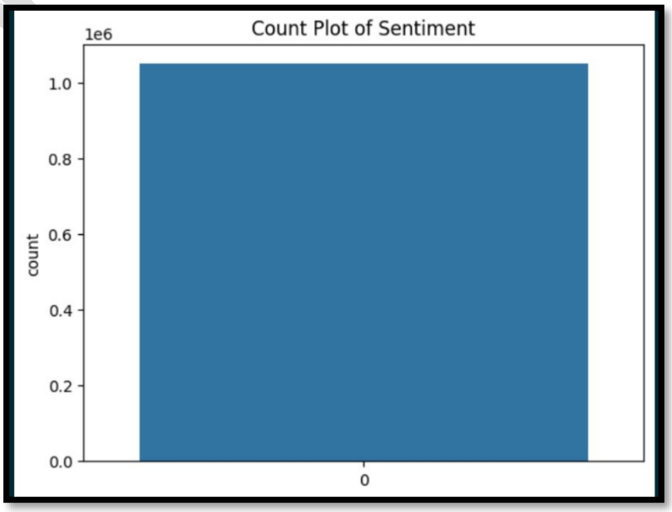
2. Data Exploration

Understand the dataset structure and key variables.

2.1 Dataset Overview

- **Source:** [Kaggle Sentiment Analysis Dataset]
- **Structure:** The dataset consists of 1.6 million tweets labeled as positive or negative.
- **Columns:** Sentiment, id, date, query, user, text

```
df = pd.read_csv('training.1600000.processed.noemoticon.csv', delimiter=',', encod
df.columns = ['Sentiment', 'id', 'date', 'query', 'user', 'text']
df = df[['Sentiment', 'text']]
df.head()
```

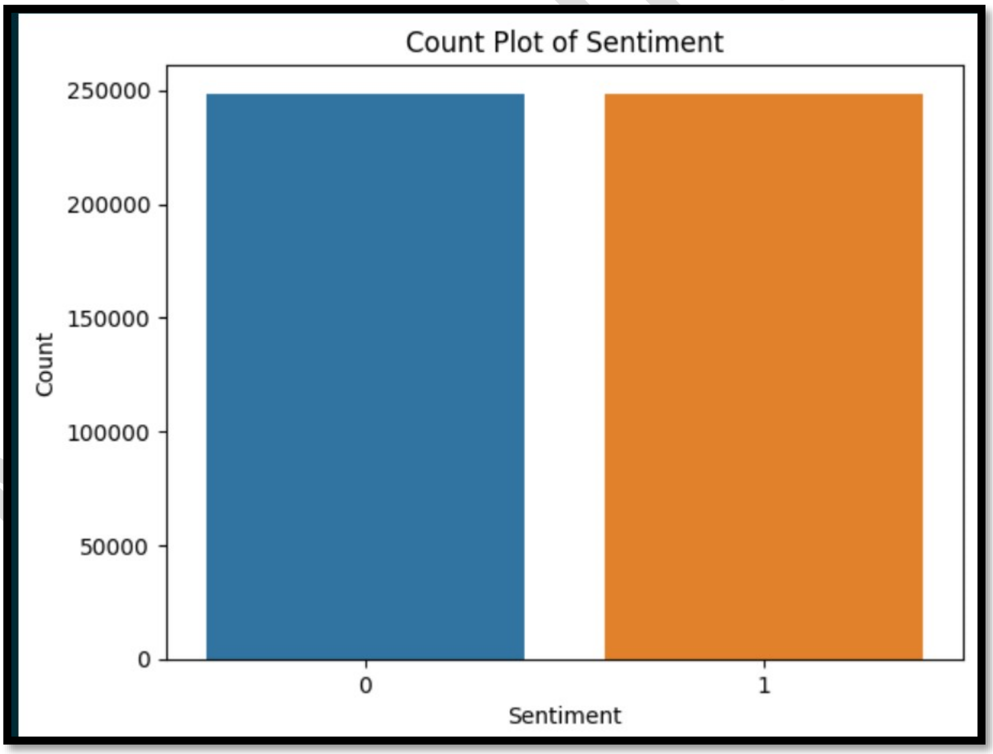


2.2 Sentiment Distribution

```
df.Sentiment.value_counts()
sns.countplot(df["Sentiment"])
plt.title("Count Plot of Sentiment")
plt.show()
```

2.3 Balancing the Dataset

```
df['Sentiment'] = df['Sentiment'].replace({4: 1})
df_positive = df[df['Sentiment'] == 1]
df_negative = df[df['Sentiment'] == 0]
df_majority_downsampled = df_negative.sample(n=len(df_positive), random_state=42)
df_balanced = pd.concat([df_majority_downsampled, df_positive]).sample(frac=1, ranc
```



3. Data Preprocessing

Clean and prepare text data for analysis.

3.1 Cleaning and Tokenization

- Removing stopwords and punctuation:

```
stuff_to_be_removed = list(stopwords.words('english')) + list(punctuation)
lem = WordNetLemmatizer()

def preprocess_text(text):
    text = re.sub('[^a-zA-Z]', ' ', text).lower()
    text = re.sub("&lt;/?.*?&gt;", " ", text)
    text = re.sub("(\\d|\\W)+", " ", text)
    text = ' '.join([lem.lemmatize(word) for word in text.split() if word not in st
    return text
```

3.2 Example of Cleaned Data

```
df[['text', 'cleaned_text']].head()
```

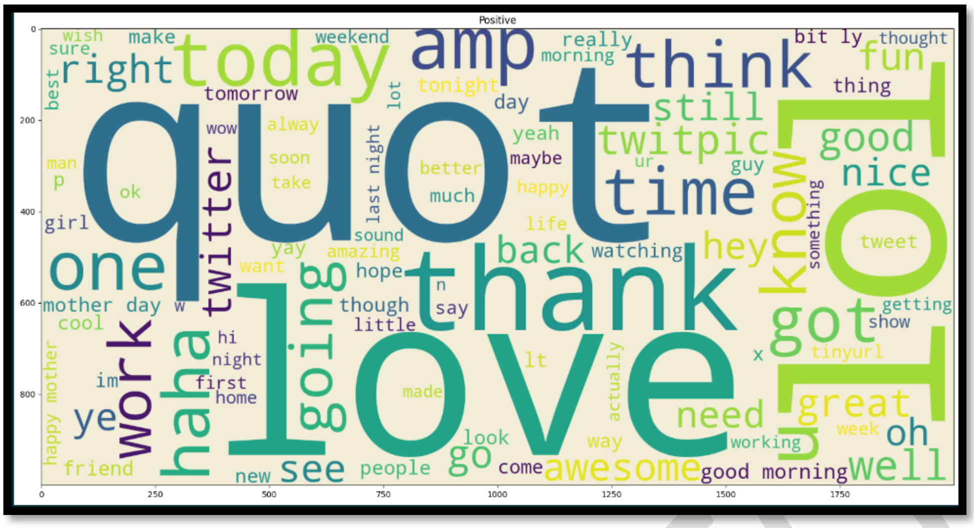
4. Exploratory Data Analysis (EDA)

Visualize sentiment label distribution.

4.1 Word Frequency

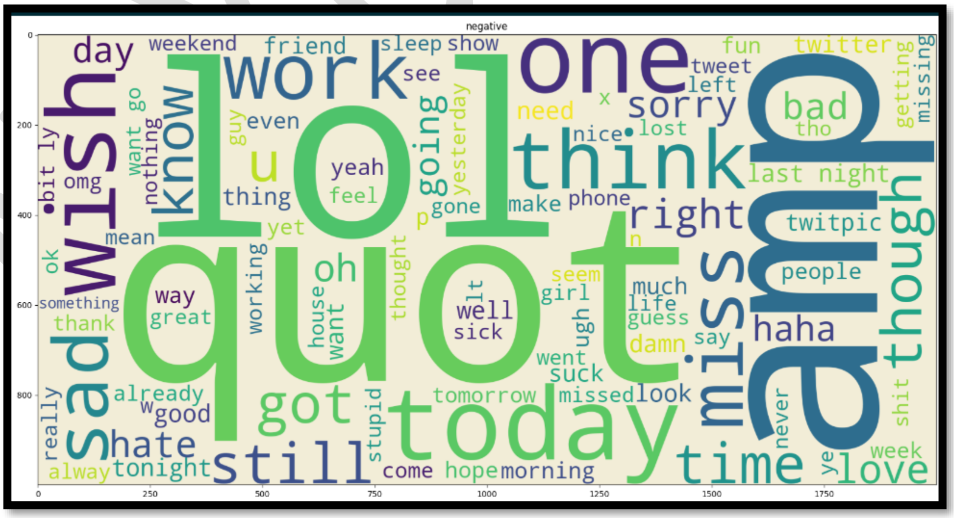
- Positive Sentiment:

```
positive_list = df[df['Sentiment'] == 1]['cleaned_text'].tolist()
positive_words = ' '.join(positive_list).split()
count_corpus = pd.DataFrame.from_dict(get_count([positive_words]), orient='index',
count_corpus = count_corpus.sort_values(by='count', ascending=False).head(20)

plt.figure(figsize=(15,10))
sns.barplot(x='index', y='count', data=count_corpus)
plt.title('Top 20 words in positive data')
plt.show()
```



- **Negative Sentiment:**

```
negative_list = df[df['Sentiment'] == 0]['cleaned_text'].tolist()
negative_words = ' '.join(negative_list).split()
count_corpus = pd.DataFrame.from_dict(get_count([negative_words]), orient='index',
count_corpus = count_corpus.sort_values(by='count', ascending=False).head(20)

plt.figure(figsize=(15,10))
sns.barplot(x='index', y='count', data=count_corpus)
plt.title('Top 20 words in negative data')
plt.show()
```



5. Text Vectorization

Convert text into numerical vectors.

5.1 TF-IDF Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(df['cleaned_text'])
```

6. Model Selection

Implement and evaluate various machine learning models.

6.1 Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, df['Sentiment'], test_size=0
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
y_pred = nb_model.predict(X_test)

evaluate_model(nb_model, X_test, y_test)
```

6.2 XGBoost

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)

evaluate_model(xgb_model, X_test, y_test)
```

7. Hyperparameter Tuning

Optimize model performance.

7.1 Cross-Validation

Ensure model generalization.

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(xgb_model, X_train, y_train, cv=5, scoring='f1', n_jobs
print("Mean CV Score:", cv_scores.mean())
print("Standard Deviation of CV Scores:", cv_scores.std())
```

8. Model Interpretability

Understand the model's predictions.

8.1 Feature Importance

```
from xgboost import plot_importance

plot_importance(xgb_model, max_num_features=10)
plt.show()
```

9. Evaluation Metrics

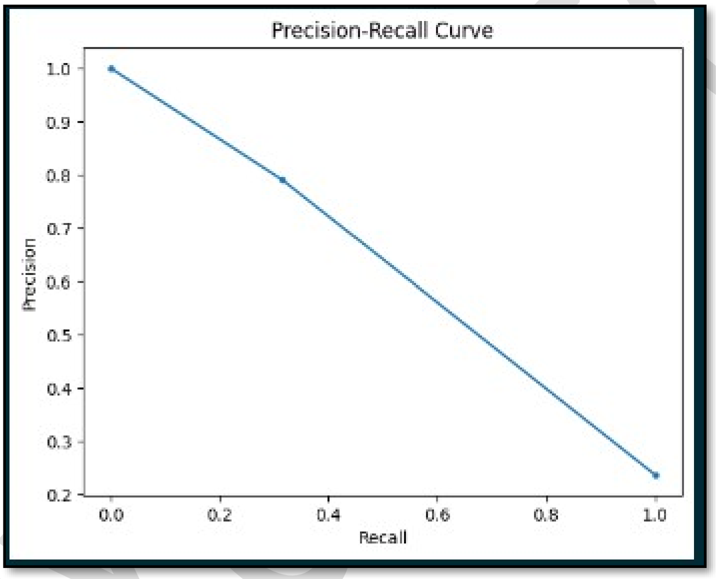
Assess model performance.

9.1 Confusion Matrix

```
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

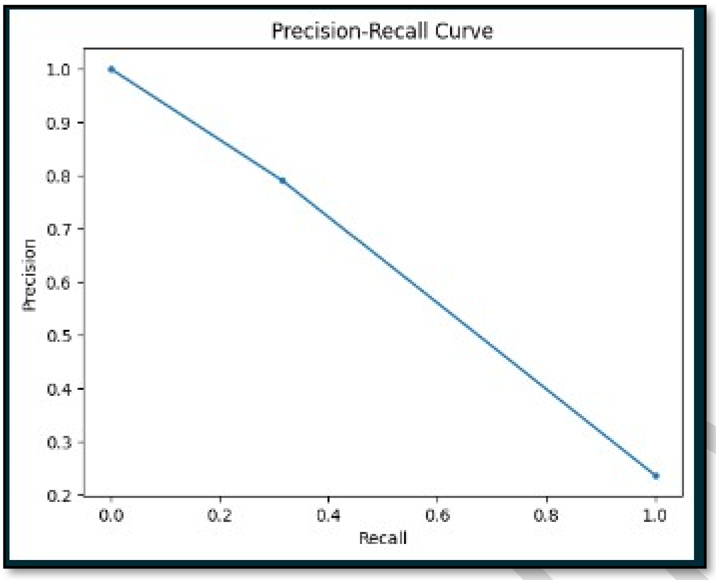
9.2 Precision-Recall Curve

```
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```



9.3 ROC Curve

```
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, marker='.')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (AUC = {:.2f})'.format(roc_auc))
plt.show()
```



XGBoost Results

Accuracy: 0.818367784850869

Precision: 0.7898134140191628

Recall: 0.31586165170918623

F1 Score: 0.4512569329395663

Confusion Matrix for XGBoost

Confusion Matrix:

[[155962   4168]

 [33923 15662]]

Naive Bayes Results

Accuracy: 0.41635076174808666

Precision: 0.24128619648250133

Recall: 0.6847837047494202

F1 Score: 0.3568388418895486

NEXUS INFO

Classification Report:

	precision	recall	f1-score	support
0	0.77	0.33	0.47	160130
1	0.24	0.68	0.36	49585
accuracy			0.42	209715
macro avg	0.51	0.51	0.41	209715
weighted avg	0.65	0.42	0.44	209715

NEXUS INFO