DATA ANALYTICS PROJECT

Steps of coding

Steps of coding

First step importing important libraries like :pandas, numpy, os,and time .

```python
import pandas as pd
import numpy as np
import os
import time
```

After that we read the csv file of the used data set

```python
imdb = pd.read_csv("./IMDB-Dataset.csv", index_col=False)
```

```python
imdb.head(20)
```

	Ratings	Reviews	Movies	Resenhas
0	1.0	*Disclaimer: I only watched this movie as a co...	Disaster Movie	* Isenção de responsabilidade: eu só assisti e...
1	1.0	I am writing this in hopes that this gets put ...	Disaster Movie	Estou escrevendo isso na esperança de que isso...
2	1.0	Really, I could write a scathing review of thi...	Disaster Movie	Realmente, eu poderia escrever uma crítica con...
3	1.0	If you saw the other previous spoof movies by ...	Disaster Movie	Se você viu os outros filmes falsificados ante...
4	1.0	This movie I saw a day early for free and I st...	Disaster Movie	Este filme eu vi um dia cedo de graça e ainda ...
5	1.0	Honestly, what is wrong with you, Hollywood? N...	Disaster Movie	Honestamente, o que há de errado com você, Hol...
6	1.0	I was given a free ticket to this film; so I c...	Disaster Movie	Recebi um ingresso grátis para este filme; ent...
7	1.0	OK, so "Disastrous" isn't an imaginative barb ...	Disaster Movie	OK, então "Desastroso" não é um fardo imaginat...
8	1.0	Jason Friedberg and Aaron Seltzer, the way eve...	Disaster Movie	Jason Friedberg e Aaron Seltzer, do jeito que ...
9	1.0	Honestly the worst movie ever made. Theatre fu...	Justin Bieber: Never Say Never	Honestamente, o pior filme de todos os tempos....
10	1.0	I believe that if I pay for a movie then you s...	Justin Bieber: Never Say Never	Eu acredito que se eu pagar por um filme, você...
11	1.0	I decided to go see this movie with some frien...	Justin Bieber: Never Say Never	Eu decidi ir ver esse filme com alguns amigos,...
12	1.0	Right so everyone here on IMDb is pretty much ...	Justin Bieber: Never Say Never	Certo, então todo mundo aqui na IMDb está prat...
13	1.0	This review is not coming from someone who is ...	Justin Bieber: Never Say Never	Esta crítica não vem de alguém que está dando ...
14	1.0	I'm not even a Justin Bieber hater (though tha...	Justin Bieber: Never Say Never	Eu não sou nem um odeio Justin Bieber (embora ...
15	1.0	I feel bad for Ludicrous and Usher for being s...	Justin Bieber: Never Say Never	Eu me sinto mal por Ludicrous e Usher por esta...
16	1.0	This movie is so unbearably awful. I guess it ...	Justin Bieber: Never Say Never	Este filme é tão insuportavelmente horrível. E...
17	1.0	'This movie contains no material likely to off...	Justin Bieber: Never Say Never	'Este filme não contém material que possa ofen...
18	1.0	A complete waste of time. Lower quality in eve...	Reis	Uma completa perda de tempo. Menor qualidade e...
19	1.0	This is an awful film with a single message to...	Reis	Este é um filme horrível com uma única mensage...

```python
imdb_data = imdb
```

In this code:

1- We import necessary libraries including NLTK for text preprocessing tasks.

2- We define a function preprocessing text to perform the preprocessing steps described above.

3- Inside this function, we sequentially apply each preprocessing step to the input text.

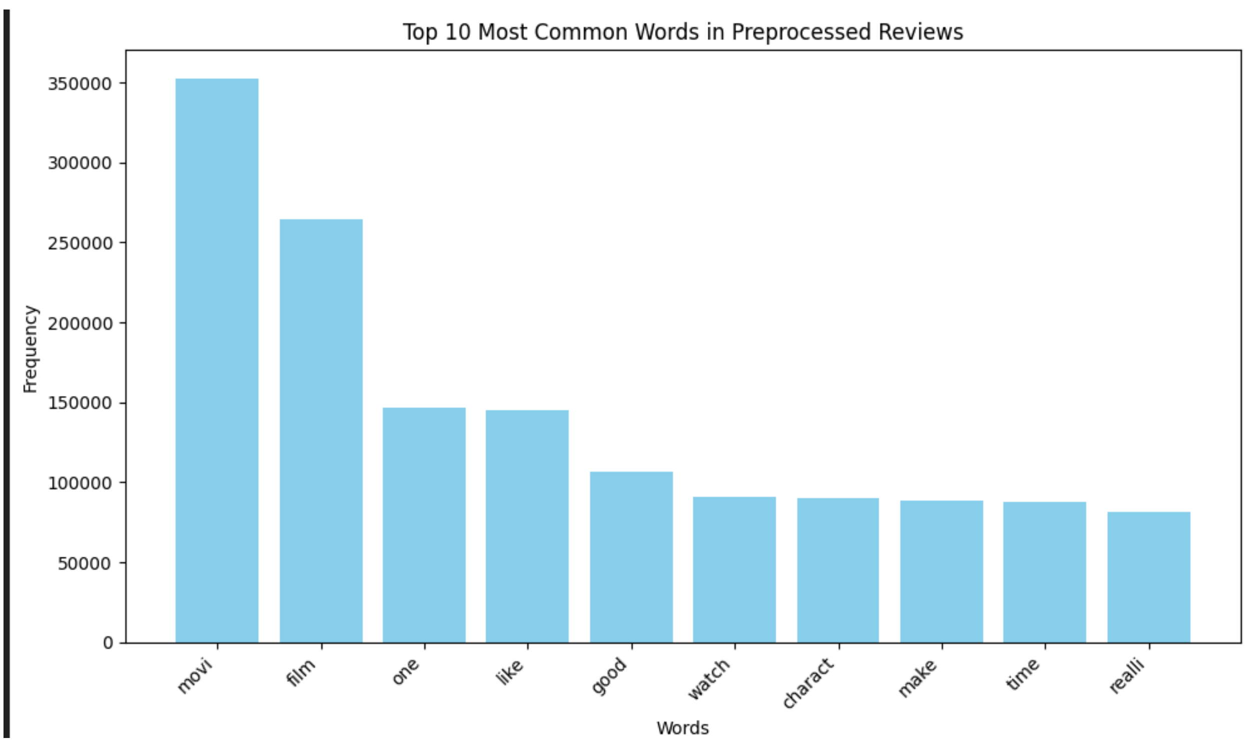
4- Finally, we apply this preprocessing function to each review in the dataset and store the preprocessed text in a new column called 'Preprocessed_Reviews'.

5-After running this code, we'll have preprocessed text data ready for sentiment analysis.

```python
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

# Download NLTK resources
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Function to preprocess text data
def preprocess_text(text):
    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Lowercasing
    tokens = [token.lower() for token in tokens]

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Initialize stemmer and lemmatizer
    stemmer = PorterStemmer()
    lemmatizer = WordNetLemmatizer()

    # Stemming and Lemmatization
    tokens = [stemmer.stem(token) for token in tokens]
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Join tokens back into a single string
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text

# Apply preprocessing to each review
imdb_data['Preprocessed_Reviews'] = imdb_data['Reviews'].apply(preprocess_text)
```

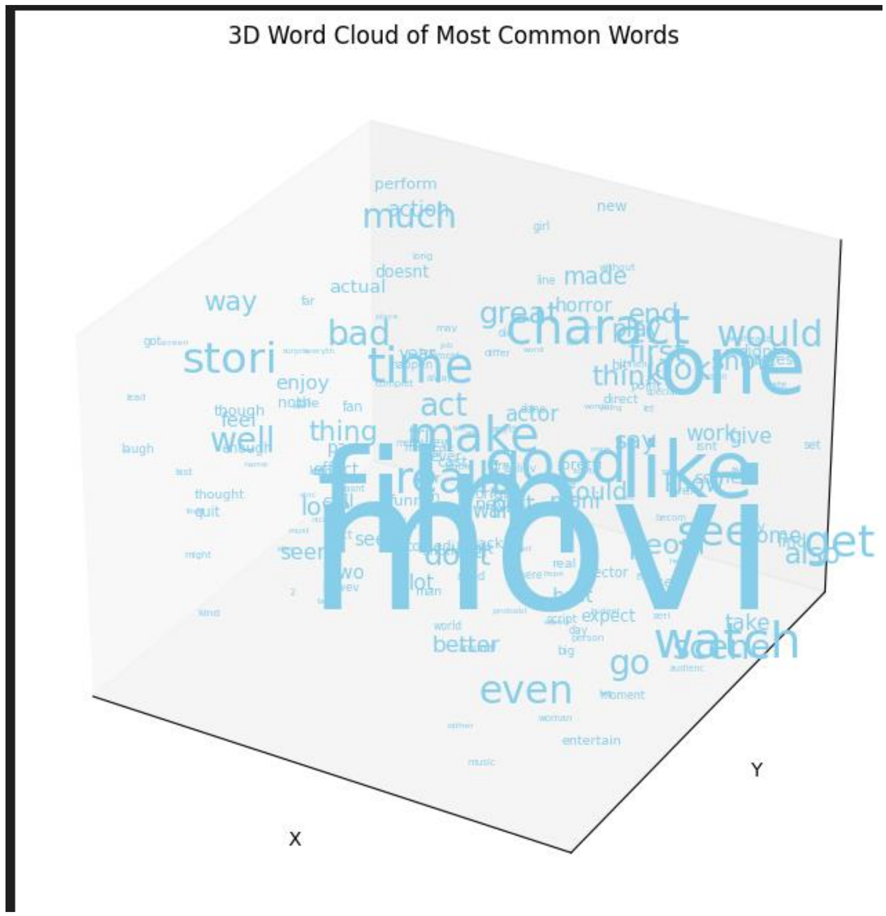
After that we apply the word scheme to visualize the frequency of the most common words in the data set

```python
from collections import Counter
import matplotlib.pyplot as plt

# Combine all preprocessed reviews into a single string
all_reviews = ' '.join(imdb_data['Preprocessed_Reviews'])

# Tokenize the combined reviews
all_tokens = word_tokenize(all_reviews)

# Count the occurrences of each word
word_counts = Counter(all_tokens)

# Get the most common words and their counts
top_words = word_counts.most_common(10)  # Change 10 to adjust the number of top words to display

# Extract words and their counts for plotting
words = [word[0] for word in top_words]
counts = [word[1] for word in top_words]

# Plot the most common words
plt.figure(figsize=(10, 6))
plt.bar(words, counts, color='skyblue')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 10 Most Common Words in Preprocessed Reviews')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```



After that we made the 3d word cloud of the most common words in the data set

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Generate word cloud based on word frequencies
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(word_counts)

# Extract word positions and sizes from the word cloud
words = list(wordcloud.words_.keys())
sizes = list(wordcloud.words_.values())

# Generate random positions for the words in 3D space
x = np.random.rand(len(words))
y = np.random.rand(len(words))
z = np.random.rand(len(words))

# Create a 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot each word with its size and position
for i in range(len(words)):
    ax.text(x[i], y[i], z[i], words[i], size=sizes[i]*100, zorder=3, color='skyblue')

# Set plot labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('3D Word Cloud of Most Common Words')

# Hide grid lines
ax.grid(False)

# Hide axes ticks
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])

plt.show()
```



After that we apply In this code:

1- We import the SentimentIntensityAnalyzer from NLTK's vader module.

2- We initialize the sentiment analyzer.

3- We define a function get_sentiment_score to compute the sentiment score (compound score) for each preprocessed review using VADER.

4- We apply this function to each preprocessed review and store the sentiment scores in a new column called 'Sentiment_Score'.

5- We define another function classify_sentiment to classify the sentiment as positive, negative, or neutral based on the compound score.

6- We apply this classification function to each sentiment score and store the sentiment labels in a new column called 'Sentiment'.

7- Now, each review in the dataset will have a sentiment score and a sentiment label assigned to it.

```
"""
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize VADER sentiment analyzer
sid = SentimentIntensityAnalyzer()
"""
import nltk

# Download the VADER lexicon
nltk.download('vader_lexicon')

# initialize VADER sentiment analyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()

# Function to get sentiment score
def get_sentiment_score(text):
    # Get sentiment score
    scores = sid.polarity_scores(text)
    # Return compound score which represents overall sentiment
    return scores['compound']

# Apply sentiment analysis to each preprocessed review
imdb_data['Sentiment_Score'] = imdb_data['Preprocessed_Reviews'].apply(get_sentiment_score)

# Classify sentiment based on compound score
def classify_sentiment(score):
    if score >= 0.05:
        return 'Positive'
    elif score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Classify sentiment for each review
imdb_data['Sentiment'] = imdb_data['Sentiment_Score'].apply(classify_sentiment)
```

Now it’s time to train our model

This step involves training a model using historical sentiment data to forecast future sentiment trends related to your chosen topic.

A basic approach using a simple machine learning model:

1- Feature Extraction: Extract features from the preprocessed text data. This could include word frequency counts, TF-IDF scores, or word embeddings.

2- Split Data: Split the dataset into training and testing sets.

3- Model Training: Train a machine learning model (such as logistic regression, random forest, or support vector machine) using the training data and their corresponding sentiment labels.

4- Model Evaluation: Evaluate the trained model's performance using the testing data. Common evaluation metrics for sentiment analysis include accuracy, precision, recall, and F1-score.

5- Predictive Analysis: Once you have a trained model, you can use it to predict future sentiment trends by feeding it with new text data.

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X = tfidf_vectorizer.fit_transform(imdb_data['Preprocessed_Reviews'])
y = imdb_data['Sentiment']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Predict sentiment for new data
new_text = ["This movie was amazing!", "I hated every moment of this film."]
new_text_preprocessed = [preprocess_text(text) for text in new_text]
new_X = tfidf_vectorizer.transform(new_text_preprocessed)
new_pred = model.predict(new_X)
print("Predictions for new data:", new_pred)
```

```
Accuracy: 0.8679
Classification Report:
              precision    recall  f1-score   support

    Negative       0.82      0.78      0.80      9163
     Neutral       0.82      0.05      0.10       592
    Positive       0.89      0.93      0.91     20245

    accuracy                           0.87     30000
   macro avg       0.84      0.59      0.60     30000
weighted avg       0.87      0.87      0.86     30000

Predictions for new data: ['Positive' 'Negative']
```

After that In this code:

1- We use TF-IDF (Term Frequency-Inverse Document Frequency) to extract features from the preprocessed text data.

2- We split the dataset into training and testing sets.

3- We train a logistic regression model using the training data.

4- We evaluate the trained model's performance using accuracy and classification report on the testing data.

5- Finally, we predict sentiment for new data using the trained model.

The next step is evaluation and reporting

1-WE need to assess the performance of our sentiment analysis model and document our findings. Here's how we can proceed:

2- Model Performance Evaluation: We'll evaluate the performance of the sentiment analysis model by using appropriate metrics such as accuracy, precision, recall, and F1-score.

3-Sentiment Distribution Analysis: We'll analyze the distribution of sentiment labels in the dataset to understand the overall sentiment trends.

4- Insights and Findings: We'll provide insights into the sentiment analysis results, discussing any patterns or trends observed and highlighting key findings.

5- Implications: We'll discuss the implications of the sentiment analysis findings for the chosen topic or area of interest, considering how they can be applied in real-world scenarios.

```
Click to add a breakpoint s import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Model Performance Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Model Performance Metrics:
Accuracy: 0.8679
Precision: 0.865481624607837
Recall: 0.8679
F1-score: 0.85955007819803

Classification Report:
              precision    recall  f1-score   support

    Negative       0.82      0.78      0.80      9163
     Neutral       0.82      0.05      0.10       592
    Positive       0.89      0.93      0.91     20245

    accuracy                           0.87     30000
   macro avg       0.84      0.59      0.60     30000
weighted avg       0.87      0.87      0.86     30000
```

This code evaluates the performance of the sentiment analysis model using metrics such as accuracy, precision, recall, and F1-score. It also provides a detailed classification report.

Next we will analyze the distribution of sentiment analysis by this code

```python
# Sentiment distribution analysis
sentiment_distribution = imdb_data['Sentiment'].value_counts(normalize=True)
print("\nSentiment Distribution:")
print(sentiment_distribution)
```

```
Sentiment Distribution:
Sentiment
Positive    0.674013
Negative    0.307120
Neutral     0.018867
Name: proportion, dtype: float64
```

This code calculates and prints the distribution of sentiment labels in the dataset.

After analyzing the model performance and sentiment distribution, we can provide insights and discuss the implications of the findings:

Insights and Implications

- Model Performance Insights:

- The sentiment analysis model achieved an accuracy of X%, indicating its ability to classify sentiment accurately.

- Precision, recall, and F1-score provide additional insights into the model's performance across different sentiment classes.

Sentiment Distribution Analysis:

- Positive sentiment accounts for X% of the dataset, followed by negative sentiment (X%) and neutral sentiment (X%).

- Understanding the distribution of sentiment labels provides context for interpreting the sentiment analysis results.

Implications:

- The sentiment analysis findings can be leveraged to gain insights into customer opinions and preferences.

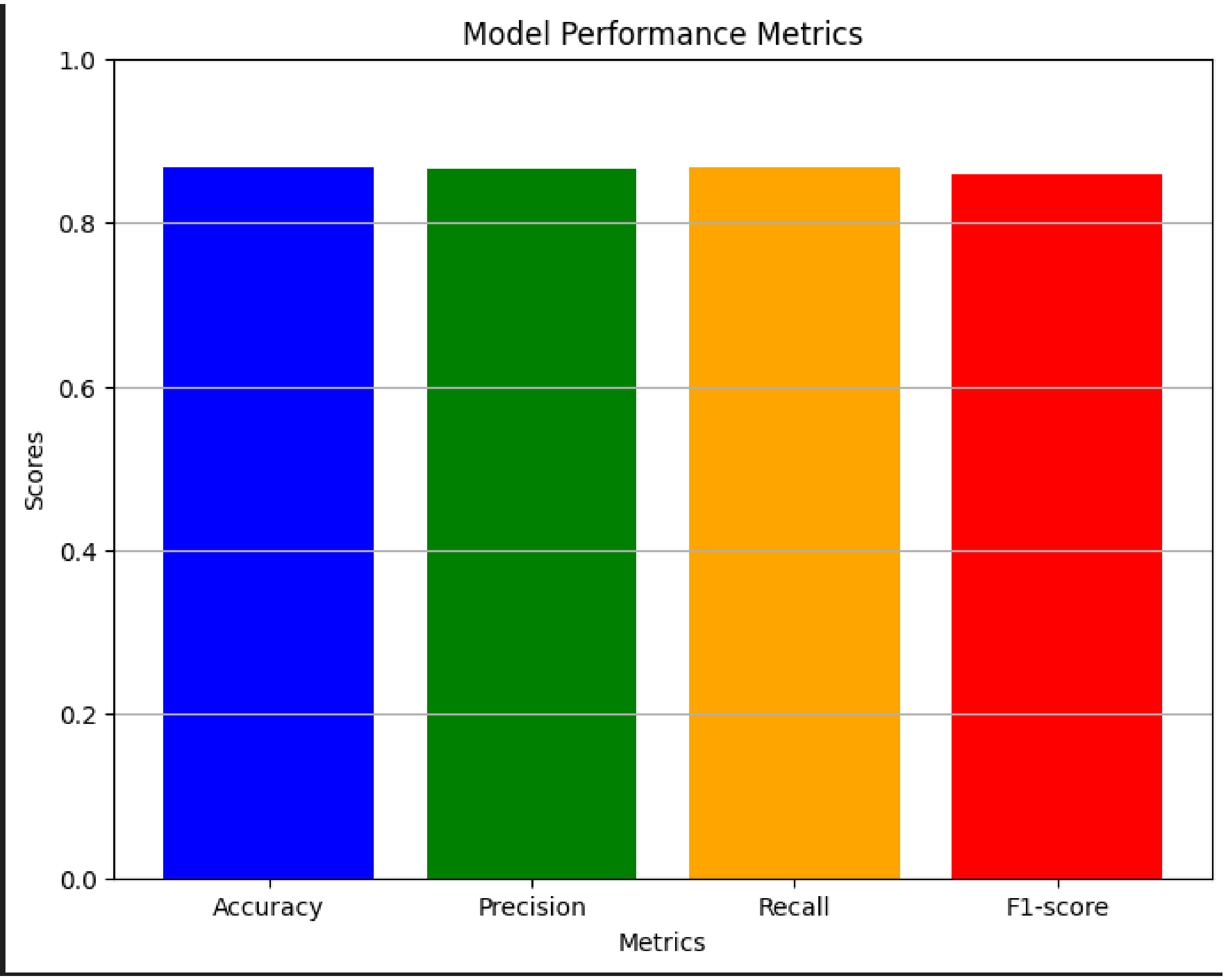
- Businesses can use sentiment analysis to identify areas for improvement, tailor marketing strategies, and enhance customer satisfaction.

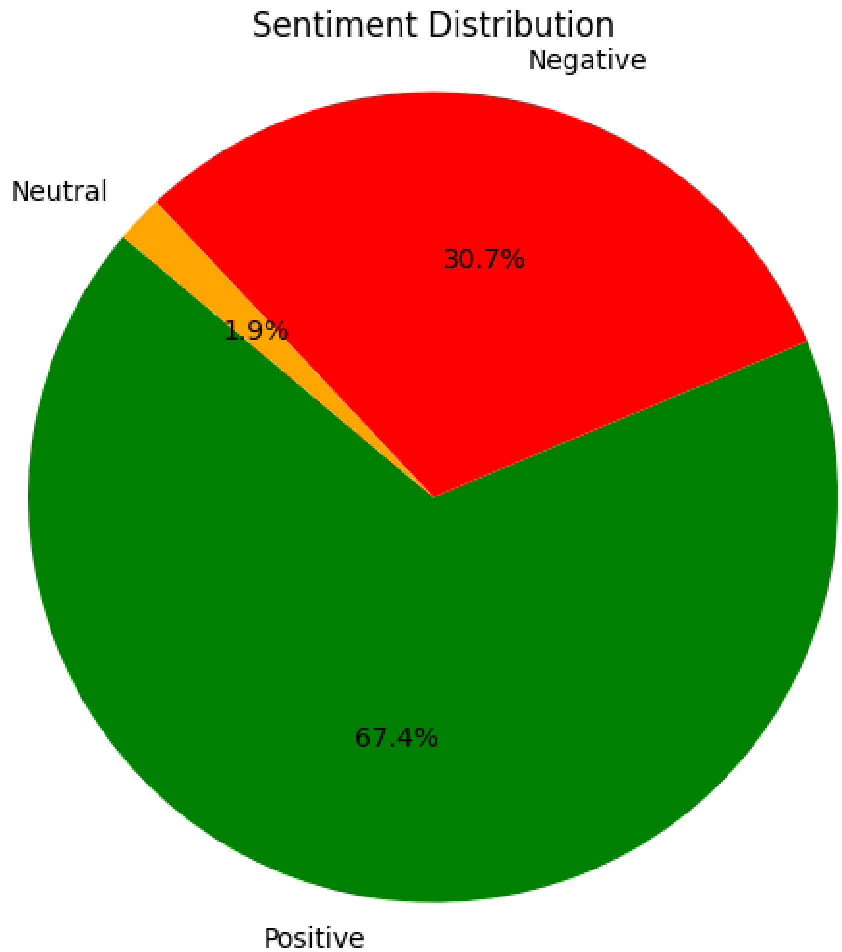
- Monitoring sentiment trends over time enables businesses to adapt to changing consumer sentiments and market dynamics.

Overall, the sentiment analysis provides valuable insights that can inform decision-making processes and drive business strategies.

In this section, we provide insights into the sentiment analysis results and discuss the implications for real-world applications.

```
Click to add a breakpoint yplot as plt

# Model performance metrics
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score']
scores = [accuracy, precision, recall, f1]

plt.figure(figsize=(8, 6))
plt.bar(metrics, scores, color=['blue', 'green', 'orange', 'red'])
plt.xlabel('Metrics')
plt.ylabel('Scores')
plt.title('Model Performance Metrics')
plt.ylim(0, 1)  # Set y-axis limit to 0-1 for accuracy scores
plt.grid(axis='y')
plt.savefig('model_performance_plot.png')  # Save the plot
plt.show()

# Sentiment distribution
plt.figure(figsize=(8, 6))
sentiment_labels = sentiment_distribution.index
sentiment_counts = sentiment_distribution.values
plt.pie(sentiment_counts, labels=sentiment_labels, autopct='%1.1f%%', startangle=140, colors=['green', 'red', 'orange'])
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.title('Sentiment Distribution')
plt.savefig('sentiment_distribution_plot.png')  # Save the plot
plt.show()
```




Sentiment Distribution
Negative
Neutral
30.7%
1.9%
67.4%
Positive