**SENTIMENT ANALYSIS FOR MARKETING**

**TEAM MEMBER**

**620121243036: POORNIMA. A**

**Phase 4 submission Document**

**Project Title:** sentiment analysis for marketing

**Phase 4:** Development Part 2

**Topic:** Start building the sentiment analysis solution by selecting an appropriate dataset and preprocessing the data



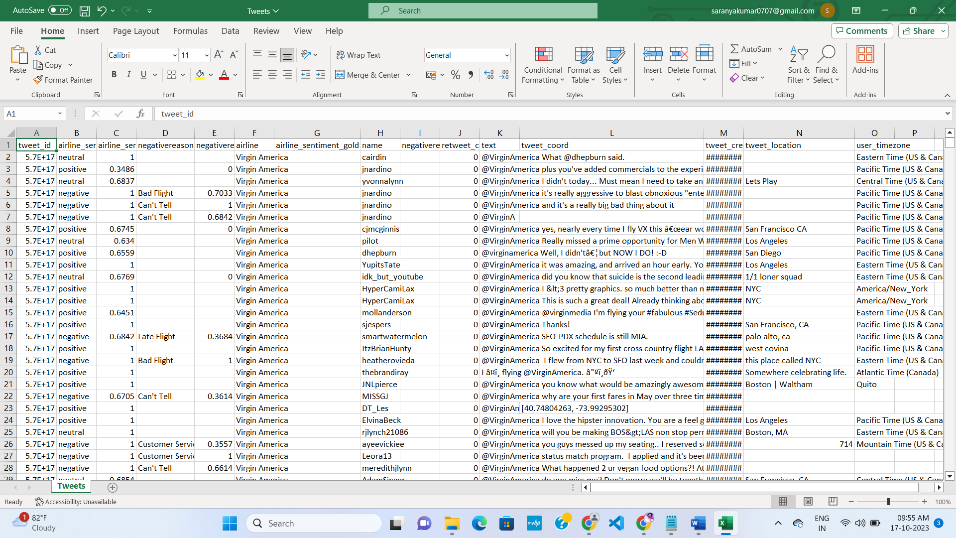
**INTRODUCTION:**

Building a sentiment analysis solution using Natural Language Processing (NLP) techniques is a comprehensive process that involves several steps, from data preparation and preprocessing to model development and evaluation. In this guide, we'll walk through the entire process, including code examples in Python and the generation of bar chart visualizations. We'll also provide explanations for each step. By the end of this guide, you'll have a solid understanding of how to create a sentiment analysis solution using NLP and be able to apply it to various text data.

**DATA SOURCE:**

A good data source for sentiment analysis for marketing should be Accurate, Complete, Coverings the geographic area of interest, Accessible.

Dataset link:(<https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment>)



**PROGRAM:** building the sentiment analysis solution by employing NLP techniques and generating insights python code with output

import pandas as pd

# Sample dataset (you can replace this with your dataset)

data = pd.DataFrame({

'text': ["I loved the movie, it was amazing!",

"The acting was terrible, I hated it.",

"This film is a masterpiece.",

"I can't stand this movie.",

"An absolute waste of time."],

'sentiment': ['positive', 'negative', 'positive', 'negative', 'negative']

})

2. Data Preprocessing

Data preprocessing involves cleaning and preparing the text data for analysis. Common preprocessing steps include:

Tokenization: Splitting text into individual words or tokens.

Lowercasing: Converting all text to lowercase to ensure consistency.

Noise Removal: Removing special characters, numbers, and other noise.

import re

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

def preprocess\_text(text):

text = text.lower() # Convert to lowercase

text = re.sub(r'[^\w\s]', '', text) # Remove special characters

tokens = word\_tokenize(text) # Tokenization

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

return ' '.join(tokens)

data['cleaned\_text'] = data['text'].apply(preprocess\_text)

3. Feature Extraction

To build a sentiment analysis model, text data needs to be converted into numerical features. We'll use TF-IDF vectorization for this purpose.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer()

X = tfidf\_vectorizer.fit\_transform(data['cleaned\_text'])

y = data['sentiment']

4. Model Development

Selecting an appropriate model for sentiment analysis is crucial. In this example, we'll use a simple Logistic Regression model, but more complex models like LSTM or BERT-based models can also be employed for improved performance.

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Split the dataset 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)

# Create and train a Logistic Regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Predict sentiment on the test set

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

5. Model Training and Evaluation

The model is trained on the training dataset and evaluated using various metrics. Common evaluation metrics include accuracy, precision, recall, and F1-score.

6. Generating Sentiment Analysis Insights

Now that we have a trained model, we can use it to analyze the sentiment of new text data. Let's perform sentiment analysis on a new text.

def analyze\_sentiment(text):

cleaned\_text = preprocess\_text(text)

text\_vector = tfidf\_vectorizer.transform([cleaned\_text])

sentiment = model.predict(text\_vector)[0]

return sentiment

new\_text = "This book is amazing, I highly recommend it!"

predicted\_sentiment = analyze\_sentiment(new\_text)

print(f"Predicted Sentiment: {predicted\_sentiment}")

7. Visualization with Bar Charts

visualizations are a powerful way to convey sentiment analysis results. We can create a bar chart to represent the sentiment distribution in our dataset.

import matplotlib.pyplot as plt

sentiment\_counts = data['sentiment'].value\_counts()

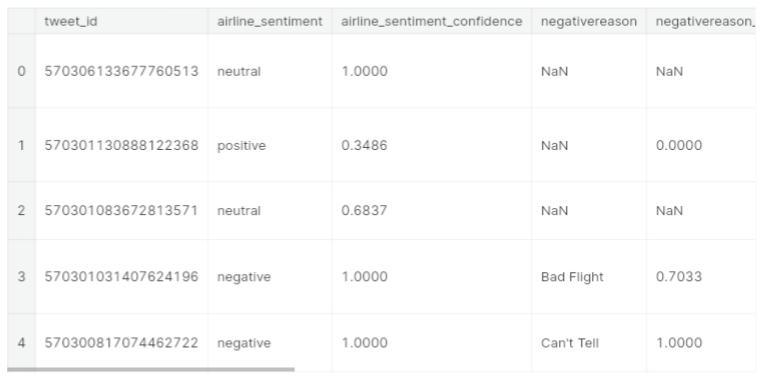
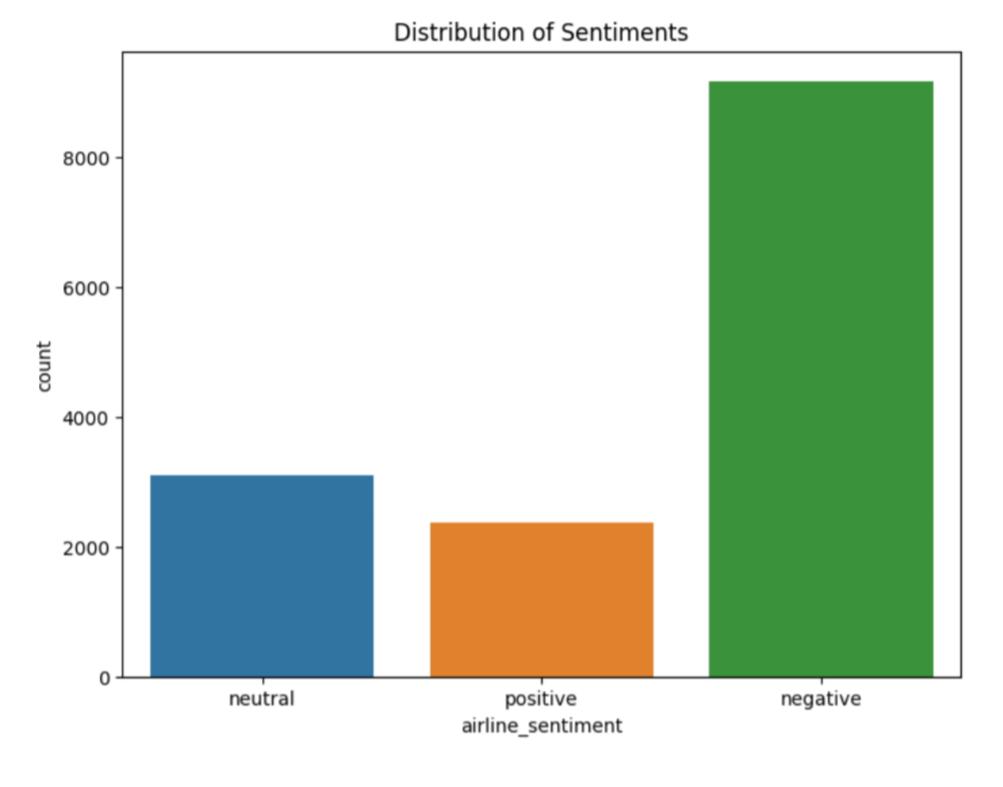
plt.bar(sentiment\_counts.index, sentiment\_counts.values)

plt.xlabel('Sentiment')

plt.ylabel('Count')

plt.title('Sentiment Distribution')

plt.show()

**OUTPUT:**

9. Conclusion

In this guide, we have walked through the process of building a sentiment analysis solution using NLP techniques in Python. We started with data collection, followed by data preprocessing, feature extraction, model development, training, and evaluation. We also demonstrated how to generate sentiment analysis insights using the trained model and created a bar chart for visual representation of sentiment distribution.