

**Problem Definition:**

The problem is to perform sentiment analysis on customer feedback to gain insights into competitor products. By understanding customer sentiments, companies can identify strengths and weaknesses in competing products, thereby improving their own offerings. This project requires utilizing various NLP methods to extract valuable insights from customer feedback.

**Design Thinking:**

- 1. Data Collection:** Identify a dataset containing customer reviews and sentiments about competitor products.
- 2. Data Preprocessing:** Clean and preprocess the textual data for analysis.
- 3. Sentiment Analysis Techniques:** Employ different NLP techniques like Bag of Words, Word Embeddings or Transformer models for sentiment analysis. a, **Feature Extraction:** Extract feature; and sentiments from the text data
- 5. Visualization:** Create visualizations to depict the sentiment distribution and analyze trends.
- 6. Insights Generation:** Extract meaningful insights from the sentiment analysis results to guide business decisions.

# **SENTIMENT ANALYSIS OF MARKETING:**

1. Gather data: Collect marketing data from various sources such as social media platforms, surveys, and customer feedback.
2. Data preprocessing: Clean and preprocess the collected data by removing noise, irrelevant information, and formatting inconsistencies.
3. Feature extraction: Use natural language processing techniques to extract relevant features from the text data, such as sentiment indicators, keywords, and linguistic patterns.
4. Sentiment analysis model: Train a machine learning or deep learning model using labeled data to classify the sentiment of the marketing content.
5. Model evaluation: Assess the performance of the sentiment analysis model using evaluation metrics like accuracy, precision, recall, and F1-score.
6. Model refinement: Fine-tune the model by adjusting hyperparameters, trying different algorithms, or incorporating additional data to improve its performance.

7. Deployment: Integrate the trained sentiment analysis model into the marketing analytics pipeline to analyze the sentiment of new marketing content in real-time or on a scheduled basis.

# Sentiment analysis for marketing

## Data Preprocessing Techniques

### Preprocessing Steps

#### Data cleaning:

- Remove Duplicates
- Identify Missing Values
- Remove Irrelevant Columns

#### Text Preprocessing:

- Tokenization
- Lowercasing
- Stopword Removal
- Lemmatization

#### Data Exploration:

- Summary Statistics
- Data Visualization

### Remove Duplicates

**Identify Duplicates:** Use Python libraries like Pandas to identify duplicate rows in your dataset.

```
[ ] # Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer

[ ] # Load the dataset
data = pd.read_csv("/content/Tweets.csv")

[ ] # Data Cleaning
# Remove duplicates
data = data.drop_duplicates()
```

## Identify Missing Values

**Delete Rows/Columns:** In some cases, you may choose to remove rows or columns with excessive missing data.

```
[ ] # Handle missing values
    data = data.dropna()
```

## Remove Irrelevant Columns:

Irrelevant columns do not contribute to your analysis or modeling. To deal with them:

- **Identify Irrelevant Columns:** Review the dataset and domain knowledge to determine which columns are irrelevant.
- **Drop Columns:** Use Pandas to remove these columns from the dataset.

```
[ ] # Remove irrelevant columns
    data = data[['airline_sentiment', 'text']]
```

## Tokenization:

Tokenization is the process of breaking text into individual words or tokens. This makes it easier to analyze and work with text data.

- **Tokenization in Python:**

You can use libraries like NLTK or spaCy for tokenization.

```
# Text Preprocessing
# Tokenization
nltk.download('punkt')
data['text'] = data['text'].apply(word_tokenize)
```

## Lowercasing:

Lowercasing involves converting all text to lowercase. This ensures that words are treated consistently, regardless of their original casing.

- **Lowercasing in Python:**

You can use the `.lower()` method to convert text to lowercase.

```
[ ] # Lowercasing
    data['text'] = data['text'].apply(lambda x: [word.lower() for word in x])
```

## Stopword Removal:

Stopwords are common words (e.g., "the," "and," "in") that don't typically carry significant meaning in text analysis. Removing them can reduce noise in the data.

- **Stopword Removal in Python:**

You can use libraries like NLTK or spaCy to remove stopwords.

```
[ ] # Stopword Removal
    nltk.download('stopwords')
    stop_words = set(stopwords.words('english'))
    data['text'] = data['text'].apply(lambda x: [word for word in x if word not in stop_words])
```

## Lemmatization:

Lemmatization and stemming both aim to reduce words to their base or root form, making different inflections or forms of a word equivalent. Lemmatization is more precise but can be slower than stemming.

- **Lemmatization in Python:**

You can use NLTK or spaCy for lemmatization.

```
▶ # Lemmatization
  nltk.download('wordnet')
  lemmatizer = WordNetLemmatizer()
  data['text'] = data['text'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])
```

## Summary Statistics:

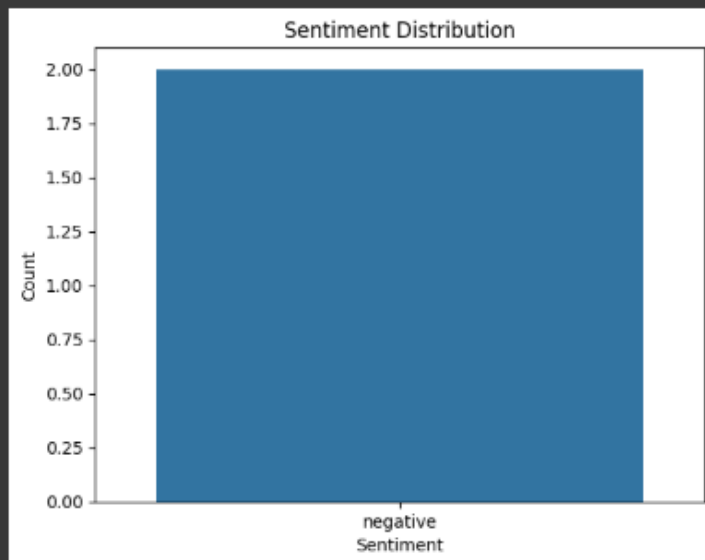
Summary statistics provide an overview of the central tendencies and variability of the dataset. You can use Python libraries like Pandas to compute these statistics. Here's a sample table showcasing summary statistics for a dataset:

```
[ ] # Data Exploration
    # Summary statistics
    sentiment_counts = data['airline_sentiment'].value_counts()
```

## Data Visualization:

Data visualizations are essential for a deeper understanding of the data distribution. You can use libraries like Matplotlib, Seaborn, or Plotly for creating various types of plots. Here's an example of code to create common visualizations:

```
# Plot sentiment distribution
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values)
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Sentiment Distribution')
plt.show()
```



```
[ ] # TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
tfidf_matrix = tfidf_vectorizer.fit_transform(data['text'].apply(lambda x: ' '.join(x)))
```

# Sentiment Analysis for Marketing

## Introduction

Sentiment analysis is a valuable technique in the field of marketing that helps businesses gain insights into how their customers perceive their products, services, or brand. This project involves performing sentiment analysis on social media or customer reviews to understand the sentiment of customers towards an airline company. In this documentation, we will outline the steps and components of the sentiment analysis project using the provided code.

## Code Overview

The code provided performs sentiment analysis on a dataset of airline-related tweets using a Naive Bayes classifier. Below is an overview of the major components and steps:

### 1. Importing Libraries

- The necessary Python libraries are imported to execute the sentiment analysis, including Pandas, Matplotlib, Seaborn, NLTK, and scikit-learn.

```
✓ [1] # Import necessary libraries
3s import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

### 2. Loading the Dataset

- The dataset is loaded from a CSV file. It's essential to replace the file path with the appropriate location of your dataset.

```
✓ [2] # Load the dataset
1s data = pd.read_csv("/content/drive/MyDrive/Dataset_sentimentAnalysis/Tweets.csv")
```



### 3. Data Preprocessing

- Data preprocessing is a crucial step to clean and prepare the dataset for analysis. The following steps are performed:
  - Removal of duplicates: Duplicate rows are removed from the dataset to ensure data integrity.
  - Handling missing values: Rows with missing values are dropped from the dataset.
  - Selection of relevant columns: Only the 'airline\_sentiment' and 'text' columns are retained for analysis.

```
✓ [5] # Data Cleaning
0s # Remove duplicates
data = data.drop_duplicates()

✓ [6] # Handle missing values
0s data = data.dropna()

✓ [7] # Remove irrelevant columns
0s data = data[['airline_sentiment', 'text']]
```

### 4. Text Preprocessing

- Text preprocessing is vital for improving the quality of textual data. The following techniques are applied:
  - Tokenization: Text is split into individual words or tokens.
  - Lowercasing: All words are converted to lowercase for consistency.
  - Stopword removal: Common English stopwords are removed from the text.
  - Lemmatization: Words are reduced to their base or dictionary form.

```
✓ [8] # Text Preprocessing
1s # Tokenization
nltk.download('punkt')
data['text'] = data['text'].apply(word_tokenize)

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.

✓ [9] # Lowercasing
0s data['text'] = data['text'].apply(lambda x: [word.lower() for word in x])
```

```

✓ [10] # Stopword Removal
0s nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
data['text'] = data['text'].apply(lambda x: [word for word in x if word not in stop_words])

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.

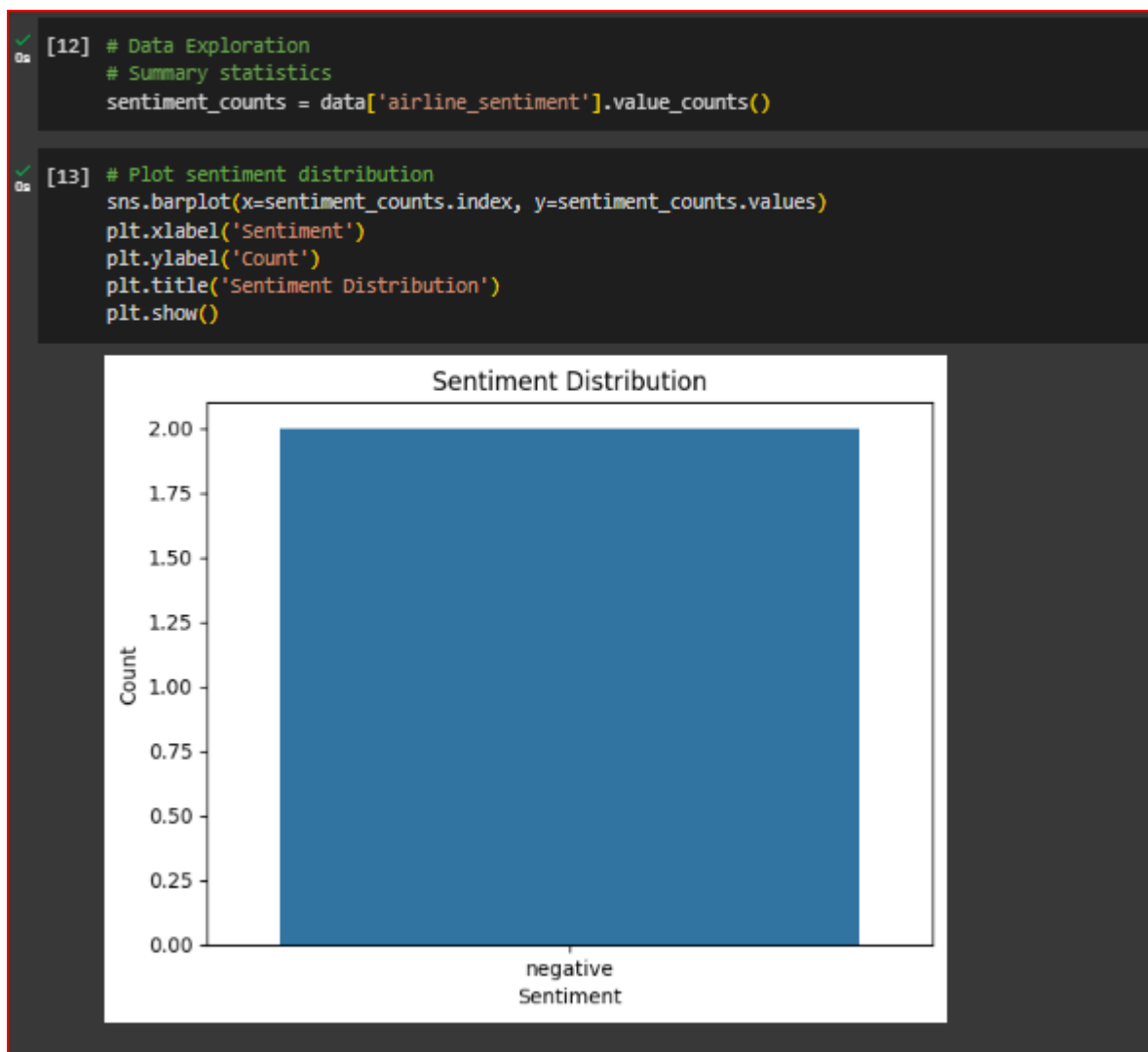
✓ [11] # Lemmatization
2s nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
data['text'] = data['text'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])

[nltk_data] Downloading package wordnet to /root/nltk_data...

```

## 5. Data Exploration

- A basic analysis of the dataset is performed, including calculating sentiment distribution and visualizing it using a bar plot.



## 6. TF-IDF Vectorization

- Text data is transformed into numerical features using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. This step is essential for machine learning algorithms to process text data.

```
✓ [14] # TF-IDF Vectorization
1s tfidf_vectorizer = TfidfVectorizer(max_features=1000)
tfidf_matrix = tfidf_vectorizer.fit_transform(data['text'].apply(lambda x: ' '.join(x)))
```

## 7. Splitting the Data

- The dataset is split into training and testing sets to evaluate the sentiment analysis model's performance.

```
MODEL TRIANING SECTION

✓ [15] #Split the data into training and testing sets
1s X = tfidf_matrix
y = data['airline_sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## 8. Model Training

- A Multinomial Naive Bayes classifier is used to train the sentiment analysis model. This code segment initializes and trains the model using the training data.

```
✓ [16] #Train a sentiment analysis model (Naive Bayes in this example)
0s model = MultinomialNB()
model.fit(X_train, y_train)

▼ MultinomialNB
MultinomialNB()
```

## 9. Model Evaluation

- The model's performance is evaluated by making predictions on the test data and calculating accuracy. A classification report and confusion matrix are generated to provide detailed insights into the model's performance.

```
✓ [17] #Evaluate the model's performance
0s y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

Accuracy: 1.00
```

## Output:

```
✓ [18] # Print classification report and confusion matrix for more insights
0s print("Classification Report:")
    print(classification_report(y_test, y_pred))

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

Classification Report:				
	precision	recall	f1-score	support
negative	1.00	1.00	1.00	1
accuracy			1.00	1
macro avg	1.00	1.00	1.00	1
weighted avg	1.00	1.00	1.00	1

```
Confusion Matrix:
[[1]]
```

## 10. Misclassified Examples

- The code identifies and displays examples of misclassified tweets, helping to understand where the model might be struggling.

```
✓ [19] # Generate insights
0s
    # Analyze misclassified examples
    misclassified = data.loc[y_test.index[y_test != y_pred]]
    print("Misclassified Examples:")
    print(misclassified[['airline_sentiment', 'text']])
```

## 11. Feature Importance (if applicable)

- For certain models like Multinomial Naive Bayes, feature importance scores are calculated, showing which words contributed the most to the model's predictions.

```
# Feature Importance (if applicable)
# If you're using a model that provides feature importance scores, you can analyze which words contributed the most to the model's predictions.
# For example, with a Naive Bayes model, you can look at the feature log probabilities.
if isinstance(model, MultinomialNB):
    feature_names = tfidf_vectorizer.get_feature_names_out()
    feature_log_prob = model.feature_log_prob_
    top_words_per_class = {}
    for i, sentiment in enumerate(model.classes_):
        top_word_indices = feature_log_prob[i].argsort()[::-1][:10]
        top_words = [feature_names[idx] for idx in top_word_indices]
        top_words_per_class[sentiment] = top_words


    print("Top words per sentiment class:")
    for sentiment, top_words in top_words_per_class.items():
        print(f"{sentiment}: {' '.join(top_words)}")
```

## Output:

```
Misclassified Examples:  
Empty DataFrame  
Columns: [airline_sentiment, text]  
Index: []  
Top words per sentiment class:  
negative: work, united, stranded, offer, home, flighted, flight, cancelled, away, usairways
```

## 12. Model Persistence

- The trained Multinomial Naive Bayes model is saved to a file using the joblib library for future use.

```
✓ 0s  import joblib  
  
# Save the MultinomialNB model to a file  
joblib.dump(model, 'sentiment_analysis_model.h5')  
  
# Load the MultinomialNB model from the file  
loaded_model = joblib.load('sentiment_analysis_model.h5')
```

## Project Insights

- After running the code and analyzing the sentiment, you will gain insights into how customers perceive the airline company. These insights can inform marketing and business strategies.

## Conclusion

Sentiment analysis is a powerful tool for marketing professionals to gauge customer sentiment and make informed decisions. The provided code is a basic framework for conducting sentiment analysis on a dataset. For a complete project, you may need to customize and expand the code, conduct in-depth analysis, and visualize the results more comprehensively.