Sentiment Analysis of Airline Tweets: Understanding Public Sentiment for Enhanced Customer Experience

**Introduction**

In this modern era of social media and thechnology platforms like Twitter and many more have become a primary channel for customers to share their feedback and experiences with brands. For the airline industry this feedback more often includes valuable insights into customer satisfaction, complaints, and suggestions however the sheer volume of tweets makes it challenging and complex to manually analyze and respond to customer sentiments effectively.

This project below leverages sentiment analysis, a key application of natural language processing also known as (NLP) to automatically classify airline tweets as positive, neutral, or negative. By understanding public sentiment like tweets airlines can identify areas of improvement, prioritize customer complaints/satsfaction and enhance overall service quality.

The analysis will be conducted by the data analyst in hand using Python in a Jupyter Notebook environment, while utilizing libraries such as Pandas for data handling, matplotlib for data visualization, and scikit-learn for building machine learning models. This project’s aim is to provide actionable insights for the airline industry by automating the sentiment classification process, making it efficient, easy to understand and scalable.

**Problem Statement**

The airline industry always operates in a highly competitive market where customer satisfaction plays a critical role in building brand loyalty and ensuring business success and errors will have major impacts. With the growing influence of social media platforms like Twitter and others, customers now have a public and fast way to share their experiences, whether positive, negative, or neutral. These tweets often provide important feedback about airline services such as being on time, customer support, and overall travel experience.

This project aim is to answer the question: How can airlines use sentiment analysis to understand customer feedback from tweets? By applying sentiment analysis, the aim is to classify tweets into three categories positive, neutral, or negative. This classification will enable airlines to identify key pain points, prioritize urgent issues, and recognize areas of excellence.By automating this process, airlines can always respond to customer concerns, make operational strategies, save time and improve overall customer satisfaction while ultimately gaining a competitive edge in the industry.

**Load the Dataset**

In this step, we began by loading the dataset into a pandas dataframe to fully understand its structure and contents. The dataset comprises more than 10000 entries and 15 columns, including features such as airline sentiment, texts, negativereason, airline etc. We displayed below the structure and first few rows of the dataset as the complete dataset was way big but an orignal pdf will be attached on the github link below. A summary of the dataset shows that their are missing values in certain columns, including negativereason and negativereason confidence, and revealed that some columns, like airline sentiment gold are largely null and may not contribute to our analysis that much. Additionally, the sentiment distribution was analyzed, showing a clear imbalance with 9,178 negative, 3,099 neutral, and 2,363 positive sentiments. This step provided us with a comprehensive understanding of the dataset’s structure.

Use Pandas to load the dataset and inspect its structure. Here's the code snippet:

-Import necessary libraries

**import pandas as pd**

-Load the dataset directly from the uploaded file

**df = pd.read\_csv("Tweets.csv") # Use the filename directly if it's in the same directory**

-Display the first few rows

**df.head()**

A screenshot of a computer

AI-generated content may be incorrect.

**Display Dataset Structure**

Check the columns, data types, and overall structure:

-Display basic information about the dataset

**df.info()**

A screenshot of a computer

AI-generated content may be incorrect.

install wordcloud

!**pip install wordcloud**

A computer code with text

AI-generated content may be incorrect.

**Dataset Summary**

Provide a quick summary of the dataset:

-Number of rows and columns

**print("Dataset shape:", df.shape)**

**Dataset shape: (14640, 15)**

Check for missing values like null:

**df.isnull().sum()**

A screen shot of a computer

AI-generated content may be incorrect.

Class distribution like: how many tweets are positive, neutral, or negative:

-Class distribution

**print(df['airline\_sentiment'].value\_counts())**

A black text on a white background

AI-generated content may be incorrect.

Dataset Overview: so in this section above we did the following steps:

Open the dataset: we displayed the dataset, atleast the first few rows as an example and as a assournce that the data can be loaded.

Dataset Structure:

Shape: The dataset has 14640 rows and 15 columns, indicating a sufficiently large dataset for sentiment analysis.

Column Types: Includes integers, floats, and objects.

Missing Values:we checked the missing values.

Some columns have significant missing values:

* + - negativereason (5462 missing values)
    - negativereason\_confidence (4118 missing values)
    - tweet\_coord (13621 missing values)

Sentiment Distribution:

Negative: 9178

Neutral: 3099

Positive: 2363

This step appears completed as:

* The dataset has been loaded successfully.
* The structure, columns, and summary are displayed and analyzed.
* Potential for sentiment analysis is clear, but missing data and class imbalance should be addressed by us in future steps.

**Data Preprocessing**

**import re**

**import nltk**

**from nltk.corpus import stopwords**

**from sklearn.preprocessing import LabelEncoder**

-Download NLTK

**nltk.download('stopwords')**

**stop\_words = set(stopwords.words('english'))**

\*Step 1: Handle missing values

-Dropping rows with missing sentiment labels

**df = df.dropna(subset=['airline\_sentiment'])**

-Fill missing values in 'negativereason' with 'Unknown'

**df['negativereason'].fillna('Unknown', inplace=True)**

\*Step 2: Clean the text

**def clean\_text(text):**

-Remove URLs

**text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)**

-Remove special characters and numbers

**text = re.sub(r'\@\w+|\#','', text)**

-Remove mentions and hashtags

**text = re.sub(r"[^a-zA-Z\s]", '', text)**

-Remove extra whitespaces

**text = re.sub(r'\s+', ' ', text).strip()**

**return text**

-Apply the cleaning function to the 'text' column

**df['cleaned\_text'] = df['text'].apply(clean\_text)**

\*Step 3: Converting text to lowercase and tokenize

**def tokenize(text):**

**tokens = text.lower().split()**

-Convert to lowercase and split into words

**tokens = [word for word in tokens if word not in stop\_words]**

-Remove stopwords

**return ' '.join(tokens)**

**df['tokenized\_text'] = df['cleaned\_text'].apply(tokenize)**

\*Step 4: Perform label encoding for sentiment classes

**label\_encoder = LabelEncoder()**

**df['sentiment\_label'] = label\_encoder.fit\_transform(df['airline\_sentiment'])**

-Mapping for the labels

**label\_mapping = dict(zip(label\_encoder.classes\_, label\_encoder.transform(label\_encoder.classes\_)))**

**print(f"Label Mapping: {label\_mapping}")**

**- Label Mapping: {'negative': 0, 'neutral': 1, 'positive': 2}**

In this step our focus was on preparing the dataset for sentiment analysis through data preprocessing. First what we did was we addressed missing values by dropping rows with missing sentiment labels and then filling any missing values in the negativereasoncolumn with unknown. Next step was we cleaned the text data by removing special characters, punctuation, links, and stopwords to retain only the meaningful content of the tweets that we needed. We also converted all text to lowercase to maintain consistency and tokenized the tweets to facilitate further text analysis. And then we performed label encoding mapping sentiment classes (negative, neutral, and positive) to numeric values (0, 1, and 2), which is essential for training the machine learning model. These preprocessing steps ensure that the data is clean, structured, and ready for building an effective sentiment analysis model for the next steps.Note: in case the coding above was alteard by Mswords and did not worked , I have added an original coding from jupyters noterbook in the github link below which includes all the coding related to the assigment.

**Exploratory Data Analysis (EDA)**

The goal here is to gain insights into the dataset by understanding the distribution of the sentiments, identifying patterns in tweets and analyzing variations in tweet lengths.

Steps for EDA:

Visualize Class Distribution

**import matplotlib.pyplot as plt**

**import seaborn as sns**

-Plot class distribution by a bar chart

**plt.figure(figsize=(8, 6))**

**sns.countplot(data=df, x='airline\_sentiment', palette='viridis', order=['negative', 'neutral', 'positive'])**

**plt.title("Sentiment Class Distribution", fontsize=16)**

**plt.xlabel("Sentiment", fontsize=14)**

**plt.ylabel("Count", fontsize=14)**

**plt.show()**

A graph showing negative and negative

AI-generated content may be incorrect.

Generate Word Clouds

**from wordcloud import WordCloud**

-Function to generate word clouds

**def generate\_word\_cloud(data, title):**

**text = ' '.join(data)**

**wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(text)**

**plt.figure(figsize=(10, 6))**

**plt.imshow(wordcloud, interpolation='bilinear')**

**plt.title(title, fontsize=16)**

**plt.axis('off')**

**plt.show()**

-Generate word clouds for each sentiment

**generate\_word\_cloud(df[df['airline\_sentiment'] == 'positive']['tokenized\_text'], "Positive Sentiment Word Cloud")**

**generate\_word\_cloud(df[df['airline\_sentiment'] == 'neutral']['tokenized\_text'], "Neutral Sentiment Word Cloud")**

**generate\_word\_cloud(df[df['airline\_sentiment'] == 'negative']['tokenized\_text'], "Negative Sentiment Word Cloud")**

A close up of words

AI-generated content may be incorrect.A close up of words

AI-generated content may be incorrect.A word cloud with text

AI-generated content may be incorrect.

Explore Tweet Lengths

-Calculate tweet lengths

**df['tweet\_length'] = df['tokenized\_text'].apply(len)**

-Plot histogram of tweet lengths

**plt.figure(figsize=(8, 6))**

**sns.histplot(data=df, x='tweet\_length', hue='airline\_sentiment', kde=True, palette='viridis', bins=30)**

**plt.title("Tweet Length Distribution by Sentiment", fontsize=16)**

**plt.xlabel("Tweet Length (characters)", fontsize=14)**

**plt.ylabel("Frequency", fontsize=14)**

**plt.show()**

A graph of a graph of a number of different colored bars

AI-generated content may be incorrect.

Outputs:

1. Class Distribution Plot: is a bar plot showing the counts of negative, neutral, and positive sentiments.
2. Word Clouds: is a visual representations of the most frequent words in each sentiment class.
3. Tweet Length Distribution: is a histogram showing the distribution of tweet lengths for each sentiment.

In the steps above we conducted an Exploratory Data Analysis to gain better insights into the patterns and characteristics of the all the dataset. We visualized the distribution of sentiments using a bar plot, which helped us highlight the imbalance in sentiment categories, with negative sentiments being the most common one followed by neutral and positive sentiments. Then we generated word clouds for each sentiment class to identify common words among them and recurring themes in positive, neutral, and negative tweets. These word clouds provided a clear visualization of the most frequent terms associated with each sentiment. Finally we analyzed the tweet lengths by plotting a histogram, which helped us reveal variations in the lengths of tweets across different sentiment classes. This analysis helped us understand the dataset better and prepared us for the next steps in building a sentiment analysis model.

**Feature extraction and Data splitting**

\*Step 1: Import Required Libraries

**from sklearn.feature\_extraction.text import TfidfVectorizer**

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

\*Step 2: Convert Text Data to Numerical Features

-Initialize TF-IDF Vectorizer

**tfidf = TfidfVectorizer(max\_features=5000) # Limit to top 5000 features for simplicity**

-Transform the tokenized text into TF-IDF features

**X = tfidf.fit\_transform(df['tokenized\_text']).toarray()**

-Target variable

**y = df['sentiment\_label']**

**Split the Data into Training and Testing Sets**

-Split the data into training (80%) and testing (20%) sets

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

-Displaying the shapes of the resulting datasets

**print(f"Training Features Shape: {X\_train.shape}")**

**print(f"Testing Features Shape: {X\_test.shape}")**

**print(f"Training Labels Shape: {y\_train.shape}")**

**print(f"Testing Labels Shape: {y\_test.shape}")**

**- Training Features Shape: (11712, 5000)**

**Testing Features Shape: (2928, 5000)**

**Training Labels Shape: (11712,)**

**Testing Labels Shape: (2928,)**

TF-IDF Vectorizer:Helped us convert the tokenized text data into numerical vectors based on the frequency of words in each tweet, while reducing the importance of commonly occurring words across all tweets.

Train-Test Split: it splits the data into training 80% and testing 20% subsets to evaluate the model's performance on unseen data in the dataset.

The results above indicates that the TF-IDF vectorization and train-test split were successfully executed.

In these steps, we focused on feature extraction to transform the text data into numerical data representing it suitable for machine learning models. We used the Term Frequency-Inverse Document Frequency vectorizer to convert the tokenized text data into numerical features, capturing the importance of each word in a tweet related to its occurrence across the dataset. The feature space was limited to the top 5,000 most important words to ensure computational efficiency. After that we split the data into training and testing sets, allocating 80% of the data for training and 20% for testing. This division ensures that the model can be trained effectively while also being evaluated on unseen data to also assess its performance. The resulting feature matrices and labels are now ready for the next step of model training and evaluation.

**Model Training and Evaluation**

Import Required Libraries

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.naive\_bayes import MultinomialNB**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report**

**Train Models**

We are gonna train three models logistic regression, naive bayes, and random forest.

-Initialize models

**logistic\_model = LogisticRegression(max\_iter=1000, random\_state=42)**

**naive\_bayes\_model = MultinomialNB()**

**random\_forest\_model = RandomForestClassifier(random\_state=42, n\_estimators=100)**

-Train models on the training set

**logistic\_model.fit(X\_train, y\_train)**

**naive\_bayes\_model.fit(X\_train, y\_train)**

**random\_forest\_model.fit(X\_train, y\_train)**

A close-up of a sign

AI-generated content may be incorrect.

**Evaluate Models**

Predict on the test set and calculate evaluation metrics.

-Evaluate each model

**models = {**

**"Logistic Regression": logistic\_model,**

**"Naive Bayes": naive\_bayes\_model,**

**"Random Forest": random\_forest\_model**

**}**

**for model\_name, model in models.items():**

**print(f"Evaluating {model\_name}...")**

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

-Calculate metrics

**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(f"Accuracy: {accuracy:.2f}")**

**print(f"Precision: {precision:.2f}")**

**print(f"Recall: {recall:.2f}")**

**print(f"F1-Score: {f1:.2f}")**

**print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))**

**print("-" \* 50)**

A screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a graph

AI-generated content may be incorrect.

Explanation

1. Model Selection:
   1. Logistic Regression: A simple and effective baseline model for text classification.
   2. Naive Bayes: Performs well for text data, particularly for TF-IDF features.
   3. Random Forest: Captures non-linear relationships and provides robust results.
2. Evaluation Metrics:
   1. Accuracy: Proportion of correctly classified samples.
   2. Precision: How many selected items are relevant.
   3. Recall: How many relevant items are selected.
   4. F1-Score: Harmonic mean of precision and recall for a balanced evaluation.
3. Model Performance Overview
4. Logistic Regression:
   1. Accuracy: 80%
   2. F1-Score: 79%
   3. Best performance overall, especially for classifying the majority class with high recall and F1-score.
5. Naive Bayes:
   1. Accuracy: 74%
   2. F1-Score: 70%
   3. Struggled with minority classes as indicated by lower recall and F1-scores.
6. Random Forest:
   1. Accuracy: 77%
   2. F1-Score: 76%
   3. A solid performance but slightly less effective than Logistic Regression for this dataset.

Best Model: Logistic regression as it achieved the highest accuracy and balanced precision, recall, and F1-score for us from the three above.The dataset in hand has an imbalanced class distribution here with negative sentiments dominating the process. We can say that Logistic regression managed this better than the other models.Naive Bayes underperformed in some asspects like minority classes.Random Forest performed well but the second best after logistic regression in terms of F1-score.

So we can say that we trained three machine learning models—Logistic Regression, Naive Bayes, and Random Forest—on the training set above to classify tweets based on their sentiment. We evaluated anf tested their performance using key metrics such as accuracy, precision, recall, and F1-score. Logistic Regression was the best performing model with an accuracy of 80% and an F1-score of 79%, showing its ability to handle imbalanced data effectively. Over all the results indicate that Logistic Regression is the most effective model for sentiment analysis on this dataset, showing a good balance between precision and recall across all sentiment classes. This step lays the foundation for using the selected model in practical applications.

**Results Visualization**

Plot Confusion Matrix for Logistic Regression

**from sklearn.metrics import ConfusionMatrixDisplay**

**import matplotlib.pyplot as plt**

-Generate predictions using Logistic regression

**y\_pred\_best = logistic\_model.predict(X\_test)**

-Plot the confusion matrix

**plt.figure(figsize=(8, 6))**

**ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred\_best, display\_labels=['Negative', 'Neutral', 'Positive'], cmap='viridis')**

**plt.title("Confusion Matrix for Logistic Regression")**

**plt.show()**

A chart with numbers and a number of different colors

AI-generated content may be incorrect.

Visualize Performance Metrics for All Models

**import numpy as np**

-Define metrics for visualization

**model\_names = ['Logistic Regression', 'Naive Bayes', 'Random Forest']**

**accuracies = [0.80, 0.74, 0.77] # Replace with actual results if they vary**

**f1\_scores = [0.79, 0.70, 0.76]**

-Plot bar chart for accuracy and F1-scores

**x = np.arange(len(model\_names)) # Label locations**

**plt.figure(figsize=(10, 6))**

-Accuracy Bar Chart

**plt.bar(x - 0.2, accuracies, width=0.4, label='Accuracy', color='teal')**

-F1-Score Bar Chart

**plt.bar(x + 0.2, f1\_scores, width=0.4, label='F1-Score', color='orange')**

**plt.xticks(x, model\_names)**

**plt.xlabel("Models", fontsize=12)**

**plt.ylabel("Scores", fontsize=12)**

**plt.title("Model Performance Comparison", fontsize=14)**

**plt.legend()**

**plt.show()**

A graph of a performance comparison

AI-generated content may be incorrect.

So we finally visualized the results of our sentiment analysis models to better understand their performance.we plotted a confusion matrix for the Logistic regression to show how well it predicted each sentiment class. This helped us identify and understand where the model excelled and where it struggled, such as any misclassifications between similar sentiments.also we created a bar chart comparing the accuracy and F1-scores of all three models logistic regression, naive bayes, and random forest to visualization and highlighted the strengths and weaknesses of each model, with logistic regression emerging as the most effective from all of them.

**Final Discussion**

The sentiment analysis model demonstrated significant strengths, making it a reliable tool for classifying airline tweets. Logistic regression, the best performing model in here achieved an accuracy of 80% with balanced precision and recall, making it effective in handling the imbalanced and complex dataset in hand. Its direct implementation method and computational efficiency further enhance its practicality for real-world applications. Adding to it the model's ability to capture trends in customer sentiment provided valuable insights for improving airline services.

However, the model also has its shortcomings and limitations. It struggles with nuanced sentiments such as sarcasm where the literal meaning of a tweet differs from its intended sentiment, mixed sentiments within a single tweet pose another challenge for it, as the model classifies each tweet into a single category. the imbalance in sentiment classes, with a predominance of negative tweets, may make predictions defecult and then may require additional techniques like oversampling or weighting.

From a business perspective, these insights above are invaluable. Airlines can use the model like these to identify common pain points, such as delayed flights or customer service issues, by analyzing negative tweets. Neutral and positive tweets highlight areas where the airline meets or it exceeds customer expectations. This allows airlines to prioritize critical feedback from them and address recurring issues proactively and enhance overall customer satisfaction. By using sentiment analysis into their operations, airlines can make data driven decisions to strengthen their brand reputation and competitiveness.

**Conclusion**

This project by us aimed to classify airline tweets into positive, neutral, or negative sentiments using sentiment analysis techniques and tools. The process begins with understanding the dataset, followed by preprocessing the text to clean and prepare it for analysis. We then converted the text data into numerical features using TF-IDF and then trained three machine learning models like logistic regression, naive bayes, and random forest. Logistic Regression emerged as the best-performing model, achieving an accuracy of over 80% and showed balanced performance across all sentiment classes. Then the results were visualized through other tools like confusion matrices and performance metrics. providing clear insights into model performance.Paython libraries such as Pandas, Scikit-learn, Matplotlib, and WordCloud played also key roles in data handling, analysis, and visualization.

This project highlights how sentiment analysis can provide actionable insights for airlines if used enabling them to enhance customer satisfaction and improve operational strategies. By addressing its limitations and incorporating advanced techniques this approach can evolve into an even more powerful tool for real-world applications.

Note: in case the coding above was alteard by Mswords and did not worked , I have added an original coding from jupyters noterbook in the github link below which includes all the coding related to the assigment.

<https://github.com/alijaweddelawari/B198c7>

**References**

Scikit learn. “Sklearn.feature\_extraction.text.TfidfVectorizer — Scikit-Learn 0.20.3 Documentation.” *Scikit-Learn.org*, 2018, scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.TfidfVectorizer.html.

scikit-learn. “1.1. Linear Models — Scikit-Learn 0.22.2 Documentation.” *Scikit-Learn.org*, scikit-learn.org/stable/modules/linear\_model.html#logistic-regression.