Customer Feedback Analysis and Improvement

# 1. Introduction

The Customer Feedback Analysis and Improvement project aims to analyze customer feedback by setting up a comprehensive database, processing the data using Python techniques, and applying sentiment analysis models to classify the feedback into positive, negative, or neutral categories. The project was executed in four main phases covering database setup, data warehousing, sentiment analysis, and cloud services integration.

# 2. Project Requirements

## Tools and Technologies Used:

- SQL Server: To create and manage the database.  
- SQL Data Warehouse: For aggregating and analyzing data.  
- Python (Pandas, NLTK, Scikit-learn): For text processing and building sentiment analysis models.  
- Azure Data Services: For cloud storage and analysis.  
- MLflow: For tracking and managing models.  
- Flask / Streamlit: To deploy sentiment analysis models via a web interface.  
- Azure App Services: To host the application on the cloud.

# 3. Project Phases

## Week 1: Database Setup and Data Collection

In the first phase, a SQL database was created to manage customer feedback. The database included the following tables:  
1. Customers Table: To store customer details such as ID, name, and email.  
2. Feedback Table: To store customer feedback with details such as feedback text and date.  
3. Feedback Categories Table: To categorize feedback into predefined categories (e.g., service, product).  
  
Historical customer feedback data was imported into the database, and SQL queries were written to extract insights such as:  
- Number of feedbacks per customer:  
 SELECT C.Name, COUNT(F.FeedbackID) AS FeedbackCount FROM Customers C JOIN Feedback F ON C.CustomerID = F.CustomerID GROUP BY C.Name;  
  
- Distribution of feedback by categories:  
 SELECT FC.CategoryName, COUNT(FM.FeedbackID) AS FeedbackCount FROM FeedbackCategories FC JOIN FeedbackCategoryMapping FM ON FC.CategoryID = FM.CategoryID GROUP BY FC.CategoryName;

## Week 2: Data Warehouse and Python Data Processing

A data warehouse was set up to aggregate feedback data for more in-depth analysis. The data was then loaded into the warehouse using SQL.  
  
Python was used for data cleaning and preprocessing, including text cleaning (removing special characters and stopwords using NLTK). Below is an example of text cleaning:  
import pandas as pd  
from nltk.corpus import stopwords  
from nltk.tokenize import word\_tokenize  
stop\_words = set(stopwords.words('english'))  
df['ProcessedText'] = df['FeedbackText'].apply(lambda x: ' '.join([word for word in word\_tokenize(x) if word not in stop\_words]))

## Week 3: Sentiment Analysis and Azure Integration

Using Scikit-learn, a Naive Bayes model was built for sentiment analysis to classify feedback into positive, neutral, or negative. The data was split into training and test sets, and the model was trained as shown below:  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.metrics import classification\_report  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['ProcessedText'], df['Sentiment'], test\_size=0.2, random\_state=42)  
model = MultinomialNB()  
model.fit(X\_train\_vec, y\_train)  
  
Data and sentiment analysis results were integrated with Azure by uploading them to Azure SQL Database. For advanced sentiment analysis, Azure Cognitive Services could also be used via the Text Analytics API.

## Week 4: MLOps and Final Presentation

MLflow was used for managing and tracking model training experiments, where metrics such as accuracy were logged for comparison. Below is a sample code for tracking a model with MLflow:  
import mlflow  
import mlflow.sklearn  
  
with mlflow.start\_run():  
 model = MultinomialNB()  
 model.fit(X\_train\_vec, y\_train)  
 accuracy = model.score(X\_test\_vec, y\_test)  
 mlflow.log\_metric('accuracy', accuracy)  
 mlflow.sklearn.log\_model(model, 'model')  
  
The model was deployed using a web interface created with Flask or Streamlit. Below is an example using Streamlit for sentiment analysis:  
import streamlit as st  
model = mlflow.sklearn.load\_model('model')  
st.title('Sentiment Analysis App')  
feedback = st.text\_input('Enter customer feedback:')  
if st.button('Analyze'):  
 prediction = model.predict([feedback])  
 st.write(f'Sentiment: {prediction[0]}')

# 4. Results and Challenges

## 4.1 Project Results:

- A well-structured SQL database was created for organizing customer feedback.  
- Python was used to clean, preprocess, and analyze the data, leading to the creation of accurate sentiment analysis models.  
- The models were successfully integrated with Azure for storage and cloud-based analysis.  
- The sentiment analysis model was deployed via a web app for easy feedback classification.

## 4.2 Challenges:

- Text cleaning was a significant challenge due to the diversity of customer feedback language.  
- Improving the model's performance required multiple trials with different algorithms.  
- Integrating Azure services required additional setup time to ensure optimal cloud performance.

# 5. Conclusion

The Customer Feedback Analysis and Improvement project highlights the importance of analyzing customer feedback to enhance business operations and understand customer needs. By using modern tools such as SQL, Python, Azure, and MLflow, a comprehensive system was built for processing and analyzing feedback, providing actionable insights for service improvements.