Name: Mukhtarahmed Mohamedhaneef Ghumra

Student Id: @00792002

Module: Machine Learning and Data Mining

Table of Contents

Task 1: Classification:

1.0 Title:

Examining Income Inequality Using Census Income

1.1 Introduction:

Globally, societies continue to struggle with income disparity. Understanding the elements that lead to income inequality can assist organizations and policymakers in creating focused actions to enhance economic prospects. This study predicts whether a person makes more than or less than $50,000 per year based on occupation and demographic characteristics using the Census Income dataset from the UCI Machine Learning Repository.

This study examines the relationships between income levels and significant variables using machine learning classification models. The studies aim to clarify the ways in which work and demographic traits impact income, with potential applications in resource allocation, targeted marketing, and policy design.

1.2 Dataset:

The dataset, known as the "**Census Income**" available on UCI repository **(Dua & Graff, 1996)** contains data on 48,842 Instances with 14 features. The target variable (income) indicates whether the individual's annual income is greater than $50,000 (>50K) or less than or equal to $50,000 (<=50K).

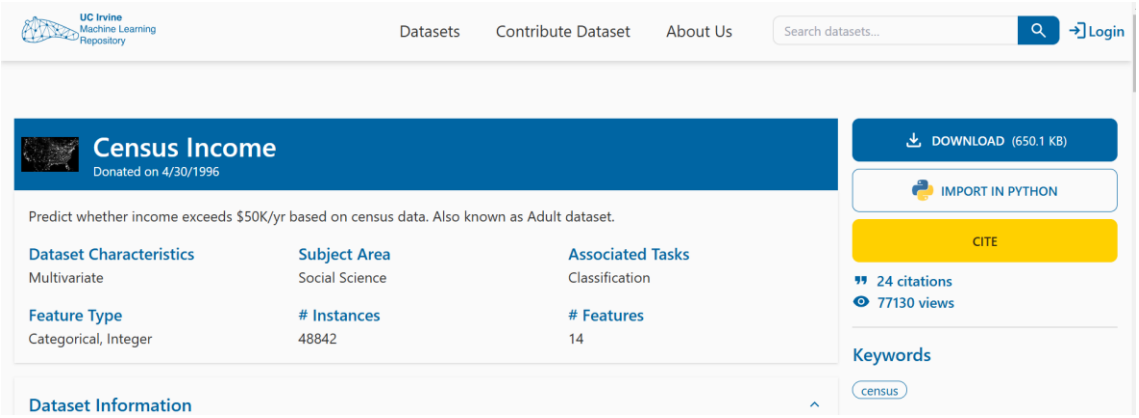


Figure 1: Census Income Data

The file originally divided into 3 parts, file containing training set, test set and columns header, I have combined it together and divided training and test as required while building model. Moreover, missing values were highlighted as (?) in the data set, which replaced by empty value () before exporting data to showcase, existing of missing value in the data set.

Numerical Features: 'age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week'

Categorical Features: 'relationship', 'workclass', 'sex', 'education', 'occupation', 'race', 'marital-status', 'native-country', 'Income'

1.3 Ethical, Social, or legal Issues:

This project demonstrates classification algorithms and machine learning techniques using the Census Income dataset only for academic purposes. The UCI Machine Learning Repository makes the dataset publicly accessible, and its usage is rigorously governed by its conditions of use.

There is no need for further ethical approval because this study is primarily meant for academic purposes. Additionally, every analysis has been carried out in a responsible manner, guaranteeing that the data is anonymised at every stage. The results are only meant to demonstrate the technical components of machine learning, and sensitive factors like race, sex, and marital status were handled carefully to prevent confirming cultural biases.

Part A: Implementation in Python:

1.4 Exploratory Data Analysis (EDA) and Data Preprocessing:

To understand the dataset better in detail we have conducted exploratory data analysis using summary statistics and visualizations.

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

Figure 2: Importing Libraries

Library Imports for Classification and Data Preprocessing in Python.

Exploratory Data Analysis (EDA)

```
#Importing the dataset
df = pd.read_csv("Census Income.csv")
```

Figure 3: Importing data set

The dataset named **"Census Income"** is being imported into a DataFrame using pd.read_csv().

```
# Display the first 5 rows of the dataset
df.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	Income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Figure 4: Initial Dataset Overview

Display first five rows of the data frame to check the data set imported correctly.

```
# Display basic information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   workclass       46043 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64
 5   marital-status  48842 non-null  object
 6   occupation      46033 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  48842 non-null  object
 14  Income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Figure 5: Dataset Summary

Checking high level information of the data frame, such as column, data type, entries, non-null counts, etc.

```
# Display basic statistics of Numerical and Categorical Features
df.describe(include="all")
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	Income
count	48842.000000	46043	4.884200e+04	48842	48842.000000	48842	46033	48842	48842	48842	48842.000000	48842.000000	48842.000000	48842	48842
unique	NaN	8	NaN	16	NaN	7	14	6	5	2	NaN	NaN	NaN	42	2
top	NaN	Private	NaN	HS-grad	NaN	Married-civ-spouse	Prof-specialty	Husband	White	Male	NaN	NaN	NaN	United-States	<=50K
freq	NaN	33906	NaN	15784	NaN	22379	6172	19716	41762	32650	NaN	NaN	NaN	43832	37155
mean	38.643585	NaN	1.896641e+05	NaN	10.078089	NaN	NaN	NaN	NaN	NaN	1079.067626	87.502314	40.422382	NaN	NaN
std	13.710510	NaN	1.056040e+05	NaN	2.570973	NaN	NaN	NaN	NaN	NaN	7452.019058	403.004552	12.391444	NaN	NaN
min	17.000000	NaN	1.228500e+04	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	1.000000	NaN	NaN
25%	28.000000	NaN	1.175505e+05	NaN	9.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	40.000000	NaN	NaN
50%	37.000000	NaN	1.781445e+05	NaN	10.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	40.000000	NaN	NaN
75%	48.000000	NaN	2.376420e+05	NaN	12.000000	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	45.000000	NaN	NaN
max	90.000000	NaN	1.490400e+06	NaN	16.000000	NaN	NaN	NaN	NaN	NaN	99999.000000	4356.000000	99.000000	NaN	NaN

Figure 6: Descriptive Analysis of data frame

Checking basic statistical information of data frame of both numerical and categorical features. Which including counts, unique values, top values, frequencies, means, standard deviations, etc.

```
# Checking the number of missing values in each column.
df.isnull().sum()
```

	0
age	0
workclass	2799
fnlwgt	0
education	0
education-num	0
marital-status	0
occupation	2809
relationship	0
race	0
sex	0
capital-gain	0
capital-loss	0
hours-per-week	0
native-country	0
Income	0

dtype: int64

Figure 7: Missing Values Analysis

Identifying the number of missing or null values in each column of the dataset. This provides insight into the quality and completeness of the dataset.

```
# Display the dimensions of the data set.
df.shape
```

```
(48842, 15)
```

Figure 8: Dataset Dimensions

Knowing the size of the dataset is crucial to estimate the resources and time required for processing. This dataset has 48,842 rows and 15 columns. Moreover, ensure that the dataset was loaded correctly without missing any features.

```
# Create a count plot using seaborn for the 'Income' column in the dataframe.
sns.countplot(x='Income', data=df)
plt.title("Income Distribution")
plt.xlabel('Income')
plt.ylabel("Count")
plt.show()
```

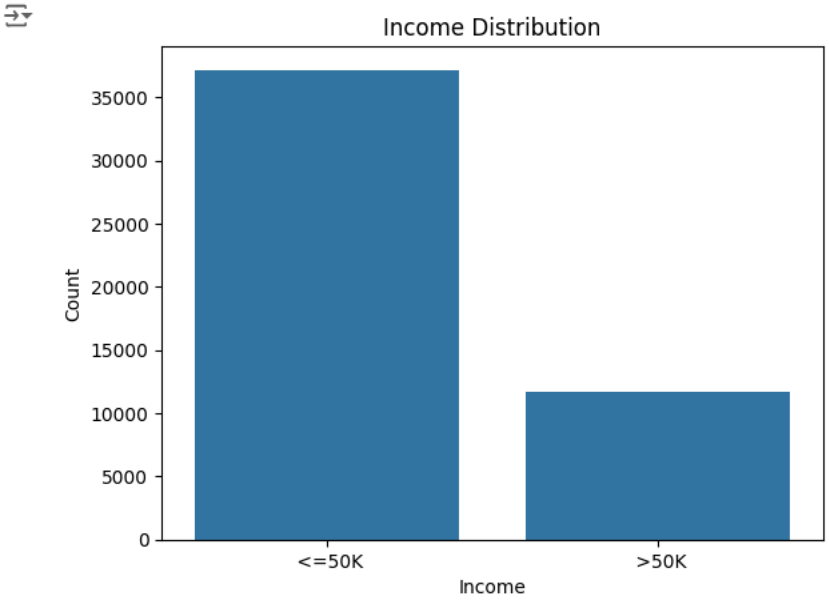


Figure 9: Income Distribution Analysis Using Count Plot

The plot reveals that the target variable is imbalanced. Here, there is a clear disparity, with significantly more individuals earning <=50K compared to >50K. Class imbalance is critical for machine learning models as it can lead to biased predictions favouring the majority class.

```
# Create a histogram using seaborn for the 'age' column in the dataframe.
# The 'bins' parameter is set to auto.
sns.histplot(x='age', data=df, bins = "auto")
plt.title("Age Distribution")
plt.xlabel('Age')
plt.ylabel("Count")
plt.show()
```

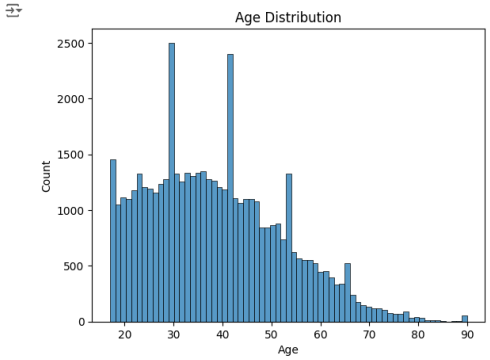


Figure 10: Age Distribution Analysis Using Histogram

The histogram provides a clear representation of the frequency of individuals within specific age ranges. Moreover, at ages around 30, 40 it shows the peak age group in dataset.

```
# Create a histogram using seaborn for the ''hours-per-week' column in the dataframe.
# The 'bins' parameter is set to 10.
sns.histplot(x='hours-per-week', data=df,bins = 10)
plt.title("Hours per Week Distribution")
plt.xlabel("Hours per Week")
plt.show()
```

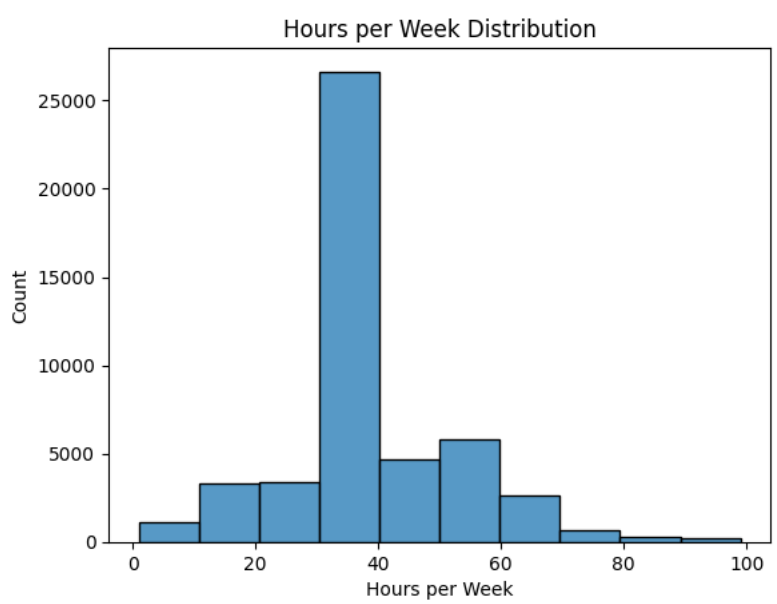


Figure 11: Analyzing Work Hours

The data shows some individuals working extremely high or low hours per week, such as majority of individuals work around 40 hours per week while very few people work more than 80 hours.

```python
# Creating a box plot to visualize the distribution of 'age' across 'Income' categories.
sns.boxplot(x='Income', y='age', data=df)
plt.title("Income vs Age")
plt.xlabel('Income')
plt.ylabel("Age")
plt.show()
```

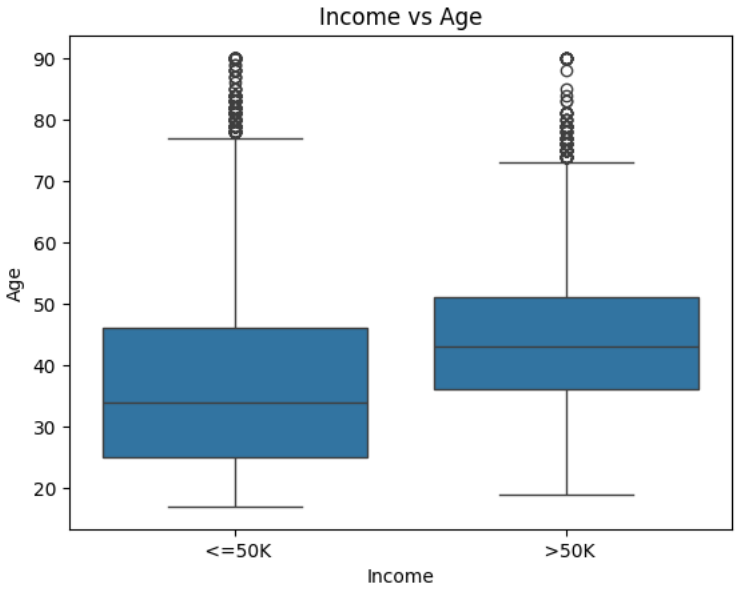


Figure 12: Analyzing Income vs. Age Distribution

This box plot provides insights into how age correlates with income levels. The visualization suggests that age might be a factor influencing income, with older individuals more likely to belong to the >50K group.

```python
# Creating a count plot to visualize the count of education levels grouped by income categories.
sns.countplot(x='education', hue='Income', data=df)
plt.title("Education Level by Income")
plt.xlabel('Education Level')
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```

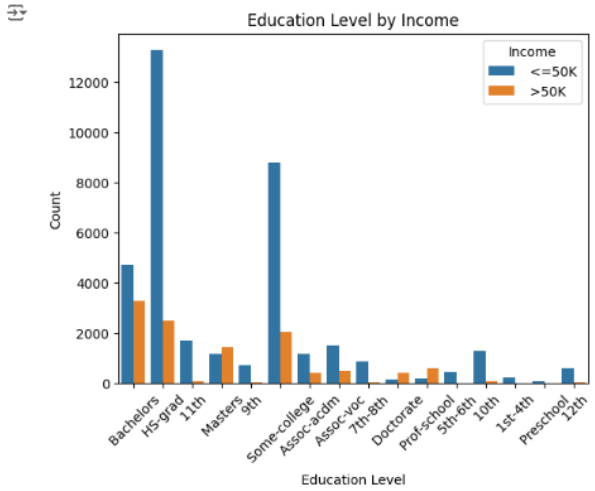


Figure 13: Analyzing Education Level by Income

The plot indicates that higher levels of education such as “Masters,” “Doctorate” are more common among individuals earning >50K, suggesting a positive correlation between education and income.

Taking care of missing data

```
# Remove rows where ["workclass","occupation"] has missing values.
df = df.dropna(subset=["workclass","occupation"])

# Checking the number of missing values in each column after handling the missing values.
df.isnull().sum()
```

	0
age	0
workclass	0
fnlwgt	0
education	0
education-num	0
marital-status	0
occupation	0
relationship	0
race	0
sex	0
capital-gain	0
capital-loss	0
hours-per-week	0
native-country	0
Income	0

dtype: int64

Figure 14: Handling Missing Data

Missing values can lead to errors or biases in analysis. Therefore, missing values in the dataset by removing rows where the "workclass" or "occupation" columns have null entries. In order to preserve data integrity, guarantee analytical accuracy, and expedite the preprocessing workflow, rows with missing values in the "workclass" and "occupation" columns were removed.

Encoding categorical data

```
# Retrieve and display all column names of the DataFrame.
cols = df.columns
cols
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'Income'],
      dtype='object')
```

```
# Extract and display the names of all numeric columns in the DataFrame.
num_cols = df._get_numeric_data().columns
num_cols
```

```
Index(['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss',
       'hours-per-week'],
      dtype='object')
```

```
# Extracting the categorical columns by deducting the numerical columns from the list of column names.
cat_cols = list(set(cols) - set(num_cols))
cat_cols
```

```
['relationship',
 'workclass',
 'sex',
 'education',
 'occupation',
 'race',
 'marital-status',
 'native-country',
 'Income']
```

Figure 15: Identifying and Preparing Categorical Data for Encoding

Systematically identifies categorical and numerical columns in the dataset, preparing it for encoding. Different data types require distinct preprocessing techniques. This step ensures a clear distinction between numerical and categorical data, setting the stage for tailored preprocessing methods for each type.

```
# Define a list of categorical columns that will undergo label encoding.
le_cat_cols =  ["Income","sex"]
le_cat_cols
```

```
['Income', 'sex']
```

```
# Define a list of columns that will undergo One-Hot encoding
ohe_cat_cols = list(set(cat_cols) - set(le_cat_cols))
ohe_cat_cols
```

```
['relationship',
 'workclass',
 'occupation',
 'race',
 'marital-status',
 'native-country',
 'education']
```

Figure 16: Divides the categorical columns

For binary or ordinal categorical columns like **"**Income" and "sex" Label Encoding best suit. However, for columns such as “workclass", "occupation" are better suited for One-Hot Encoding. According to **GeeksforGeeks (n.d.)**, categorical data can be encoded using methods such as Label Encoding and One-Hot Encoding, which are commonly implemented in Scikit-learn.

```
# Perform One-Hot Encoding on the categorical columns identified in 'ohe_cat_cols'.
df_encoded = pd.get_dummies(df, columns=ohe_cat_cols)

# Replace boolean values (True/False) with integers (1/0) in the resulting DataFrame.
df = df_encoded.replace({True: 1, False: 0})
```

```
<ipython-input-243-cf58bc34bf89>:5: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain
  df = df_encoded.replace({True: 1, False: 0})
```

```
# Apply Label encoding to ("sex", "Income") columns in the df_encoded;
label_encoder = LabelEncoder()
df_encoded[le_cat_cols] = df_encoded[le_cat_cols].apply(label_encoder.fit_transform)
```

```
# Display the first five rows of the data frame after encoding the data.
df.head()
```

	age	fnlwgt	education-num	sex	capital-gain	capital-loss	hours-per-week	Income	relationship_ Husband	relationship_ Not-in-family	...	education_ 9th	education_ Assoc-acdm	education_ Assoc-voc	education_ Bachelors	education_ Doctorate	educ H
0	39	77516	13	Male	2174	0	40	<=50K	0	1	...	0	0	0	1	0	
1	50	83311	13	Male	0	0	13	<=50K	1	0	...	0	0	0	1	0	
2	38	215646	9	Male	0	0	40	<=50K	0	1	...	0	0	0	0	0	
3	53	234721	7	Male	0	0	40	<=50K	1	0	...	0	0	0	0	0	
4	28	338409	13	Female	0	0	40	<=50K	0	0	...	0	0	0	1	0	

5 rows × 105 columns

```
# Display the dimensions of the data set after doing encoding.
df.shape
```

```
(46033, 105)
```

Figure 17: Encoding Categorical Data

Applies One-Hot Encoding and Label Encoding to convert categorical variables into numerical representations suitable for machine learning algorithms. Converts True and False values in the DataFrame into integers (1 and 0). This transformation ensures compatibility. Finally, to verify that the encoding was performed correctly applied head and shape method.

```
# Independend variable
X = df_encoded.drop("Income", axis = 1)

# Dependent variable
y = df_encoded["Income"]
```

Figure 18: Defining Variables

Separates the dataset into independent (X) and dependent (y) variables, preparing it for machine learning modelling.

```
print("First 10 rows of independent variables (X):")
print(X[:10])
```

```
First 10 rows of independent variables (X):
   age  fnlwgt  education-num  sex  capital-gain  capital-loss  \
0   39   77516             13    1          2174             0
1   50   83311             13    1             0             0
2   38  215646              9    1             0             0
3   53  234721              7    1             0             0
4   28  338409             13    0             0             0
5   37  284582             14    0             0             0
6   49  160187              5    0             0             0
7   52  209642              9    1             0             0
8   31   45781             14    0         14084             0
9   42  159449             13    1          5178             0
```

```
print("First 10 rows of dependent variables (y):")
print(y[:10])
```

```
First 10 rows of dependent variables (y):
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    1
8    1
9    1
Name: Income, dtype: int64
```

Figure 19: Preview of Independent & Dependent Variables

Displays the first 10 rows of the independent dependent variables (X) and (y) to verify the features selected for training and target the machine learning model.

Splitting the dataset into the Training and Test

```
# Splitting the data into training and testing sets
# X_train, X_test: Independent variables
# y_train, y_test: Dependent variable
# train_size = 80% and test_size= 20%
# random_state=1: Ensures same split every time
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8,test_size = 0.2, random_state = 1)
```

Figure 20: Splitting the Dataset

Splits the dataset into two subsets training and test set. Training set train the machine learning model. Test set used to evaluate the model's performance on unseen data. By allocating a portion of the data for testing, the model's ability to generalize to new data can be assessed.

"To ensure reliable evaluation of the machine learning model's performance, the dataset was divided into training and testing sets. This common practice aids in understanding how well the model generalizes to new, unseen data **(Brownlee, 2019)**"

```
print("First 5 rows of independent variables (X_train):")
print(X_train[:5])
```

```
First 5 rows of independent variables (X_train):
       age  fnlwgt  education-num  sex  capital-gain  capital-loss  \
43565   48   70584              9    1             0             0
14922   42   99679              9    1             0             0
32020   21  211968             10    0             0          1762
19006   42  230684              9    1          5178             0
14879   51   97005              4    1             0             0
```

```
print("First 5 rows of dependent variables (y_train):")
print(y_train[:5])
```

```
First 5 rows of dependent variables (y_train):
43565    0
14922    1
32020    0
19006    1
14879    0
Name: Income, dtype: int64
```

```
print("First 5 rows of independent variables (X_test):")
print(X_test[:5])
```

```
First 5 rows of independent variables (X_test):
       age  fnlwgt  education-num  sex  capital-gain  capital-loss  \
11000   20  216825              9    0             0             0
17769   42  188738              9    1             0          1977
46035   36  199217              9    1             0             0
28818   35  233571              9    0             0             0
30092   68  261897              6    1             0             0

       hours-per-week  relationship_ Husband  relationship_ Not-in-family  \
11000              25                  False                        False
17769              60                   True                        False
46035              40                  False                         True
28818              50                  False                        False
30092              20                  False                        False
```

```
print("First 5 rows of dependent variables (y_test):")
print(y_test[:5])
```

```
First 5 rows of dependent variables (y_test):
11000    0
17769    1
46035    0
28818    0
30092    0
Name: Income, dtype: int64
```

Figure 21: Preview of Training and Testing Datasets After Splitting

Confirms that the training and testing sets have been properly separated, with consistent formats for both X (independent variables) and y (dependent variable).

Handling Class Imbalance

```
# Applying SMOTE to the training data to balance the classes
from imblearn.over_sampling import SMOTE
resampler= SMOTE(random_state=0)
X_train_oversampled,y_train_oversampled=resampler.fit_resample(X_train,y_train)
sns.countplot(x=y_train_oversampled)
plt.show()
```

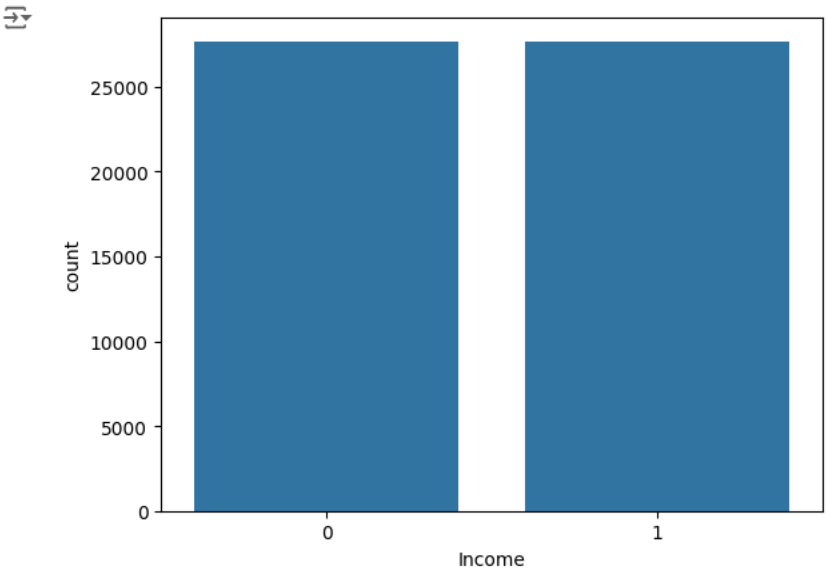


Figure 22: Handling Class Imbalance

One category has slightly more samples than the other, indicating an imbalance in the income variable. During training, class imbalance makes sure the model treats both classes equally and does not favor the majority class. The dataset is balanced to improve the model's generalization and prevent underperformance on minority class predictions.

“To address class imbalance, SMOTE was applied to generate synthetic samples for the minority class, as recommended in machine learning best practices **(Analytics Vidhya, n.d.)**.”

Feature Scaling

```
[86] # Standardizing the numerical columns in the training set (X_train) and testing set (X_test).
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train_oversampled[num_cols] = sc.fit_transform(X_train_oversampled[num_cols])
     X_test[num_cols] = sc.transform(X_test[num_cols])
```

Figure 23: Feature Scaling

This code standardizes the numerical features in the dataset by scaling them to have a mean of 0 and a standard deviation of 1 using the StandardScaler from Scikit-learn. Scaling ensures that no single feature dominates others feature.

1.5 Classification Model Implementation:

We applied two classification algorithms to the dataset K-NN and Random Forest.

1.5.1 K_Nearest_Neighbors:

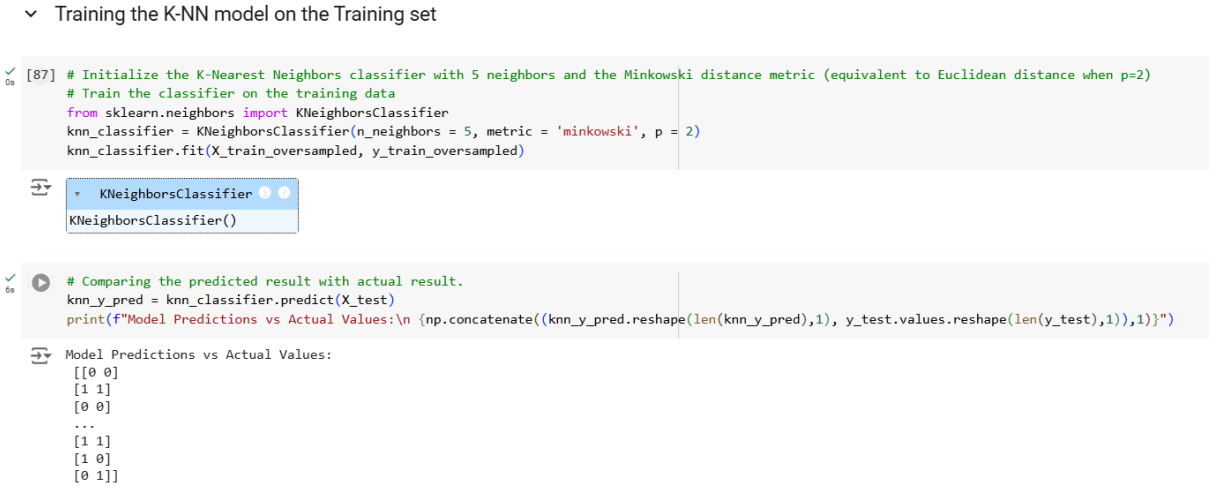


Figure 24: Training and Evaluating K-nn Model

Since K-NN is easy to use, flexible when dealing with non-linear data, and performs well on small to medium-sized datasets—especially after correcting for class imbalance—it was selected for this task. Additionally, the algorithm's interpretability and adaptability make it a good option for jobs involving income classification.

“When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction. **(Brownlee, n.d.)**.”

K-NN Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
# Calculate and print the accuracy of the model by comparing the predicted labels with the true labels
knn_acc = accuracy_score(y_test, knn_y_pred)
print("K-NN Accuracy:",knn_acc)
print("-----------------------------------------------------")
# Compute and print the confusion matrix to evaluate the performance of the classification model
knn_cm = confusion_matrix(y_test, knn_y_pred)
print(f"K-NN Confusion Matrix:\n{knn_cm}")
print("-----------------------------------------------------")
# Print the classification report to evaluate model performance.
knn_report = classification_report(y_test, knn_y_pred)
print(f"K-NN Classification Report:\n {knn_report}")
```

```
K-NN Accuracy: 0.8174215270989464
-----------------------------------------------------
K-NN Confusion Matrix:
[[6075  862]
 [ 819 1451]]
-----------------------------------------------------
K-NN Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.88      0.88      6937
           1       0.63      0.64      0.63      2270

    accuracy                           0.82      9207
   macro avg       0.75      0.76      0.76      9207
weighted avg       0.82      0.82      0.82      9207
```

Figure 25: Evaluating the K-NN Model

The K-NN model performed well on the test set, achieving a good F1-score for the majority class, a respectable recall for the minority class, and an accuracy of 81.74%. These outcomes confirm the efficacy of the model for the specified dataset.

```
# Visualize the confusion matrix using a heatmap with annotations for easier interpretation
# Customize the heatmap with the 'flare' color map, and label axes and title for clarity.
ax = sns.heatmap(knn_cm, cmap='flare' , annot=True, fmt='d')
plt.title("K-NN Confusion Matrix", fontsize=12)
plt.xlabel("Predicted Class", fontsize=12)
plt.ylabel("True Class",fontsize=12)

plt.show()
```

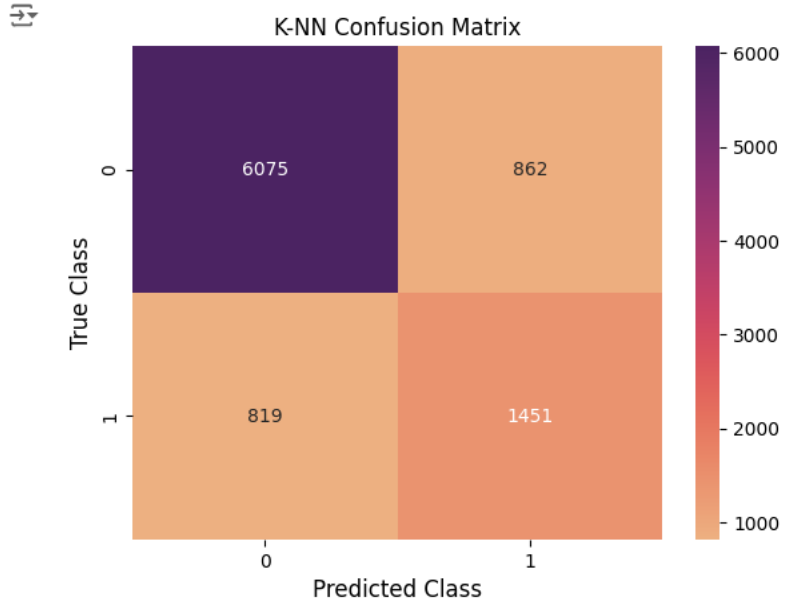


Figure 26: Visualizing the K-NN Confusion Matrix

A heatmap is used in this visual to improve interpretability of the K-NN model's confusion matrix. A thorough comparison between the model's predictions and the actual label for every class is given via a confusion matrix.

1.5.2 Random Forest:

Random Forest Classification Model

Training the Random Forest model on the Training set

```
[91] from sklearn.ensemble import RandomForestClassifier
     rf_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 1)
     rf_classifier.fit(X_train_oversampled, y_train_oversampled)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=1)
```

```
# Comparing the predicted result with actual result.
rf_y_pred = rf_classifier.predict(X_test)
print(f"Model Predictions vs Actual Values:\n {np.concatenate((rf_y_pred.reshape(len(rf_y_pred),1), y_test.values.reshape(len(y_test),1)),1)}")
```

```
Model Predictions vs Actual Values:
 [[0 0]
 [1 1]
 [0 0]
 ...
 [1 1]
 [1 0]
 [1 1]]
```

Figure 27: Training and Evaluating Random Forest Model

Random Forest's ensemble method improves minority class prediction, allowing it to handle class imbalance better than a single decision tree. Several decision trees are combined in the Random Forest ensemble learning technique to produce predictions that are more reliable and accurate. It works well with high-dimensional, complicated datasets that contain both categorical and numerical variables, such as this dataset.

“Random forests or Random Decision Trees is a collaborative team of decision trees that work together to provide a single output. **(GeeksforGeeks, n.d.)**.”

Random Forest Model Evaluation

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
# Calculate and print the accuracy of the model by comparing the predicted labels with the true labels
rf_acc = accuracy_score(y_test, rf_y_pred)
print("Random Forest Accuracy:",rf_acc)
print("-----------------------------------------------------")
# Compute and print the confusion matrix to evaluate the performance of the classification model
rf_cm = confusion_matrix(y_test, rf_y_pred)
print(f"Random Forest Confusion Matrix:\n{rf_cm}")
print("-----------------------------------------------------")
# Print the classification report to evaluate model performance.
rf_report = classification_report(y_test, rf_y_pred)
print(f"Random Forest Classification Report:\n{rf_report}")
```

```
Random Forest Accuracy: 0.8444661670468122
-----------------------------------------------------
Random Forest Confusion Matrix:
[[6382  555]
 [ 877 1393]]
-----------------------------------------------------
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6937
           1       0.72      0.61      0.66      2270

    accuracy                           0.84      9207
   macro avg       0.80      0.77      0.78      9207
weighted avg       0.84      0.84      0.84      9207
```

Figure 28: Random Forest Model Evaluation

Overall, the Random Forest model does well, attaining an accuracy of 84.45%. With 88% precision and 92% recall, it performs well for the majority class (<=50K). After addressing class imbalance, the model performs reasonably well, despite class >50K having a poorer recall (61%). These outcomes confirm Random Forest's efficacy as a classification task method.

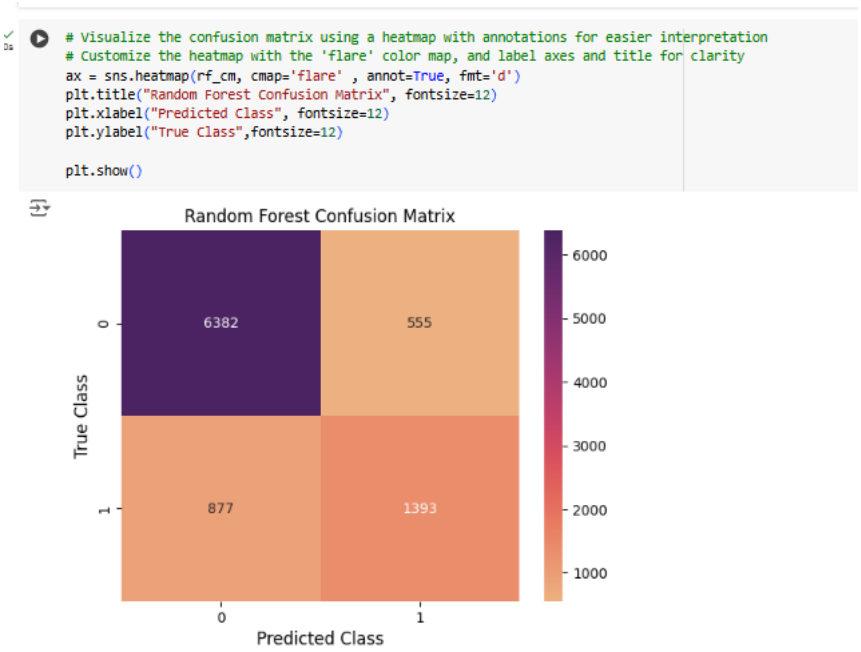


Figure 29: Random Forest Confusion Matrix Visualization

A heatmap is used in this visualization to show the Random Forest model's confusion matrix. By displaying the distribution of true and predicted class labels in an easy-to-understand manner, it offers a means of interpreting the model's classification performance.

1.6 Comparison of K-Nearest Neighbors (K-NN) and Random Forest Models:

While both models perform reasonably well, Random Forest offers better overall performance and generalization for this dataset.

- It achieves **higher accuracy (84.45%),** and **macro-average F1-score (78%)** compared to K-NN.
- It demonstrates better precision and recall trade-offs for both classes.
- It is more robust to feature scaling and can handle high-dimensional data effectively.

Part B: Implementation in Azure Machine Learning:

1.7 Exploratory Data Analysis (EDA) and Data Preprocessing in Azure

Step 1: Create an Azure Machine Learning workspace

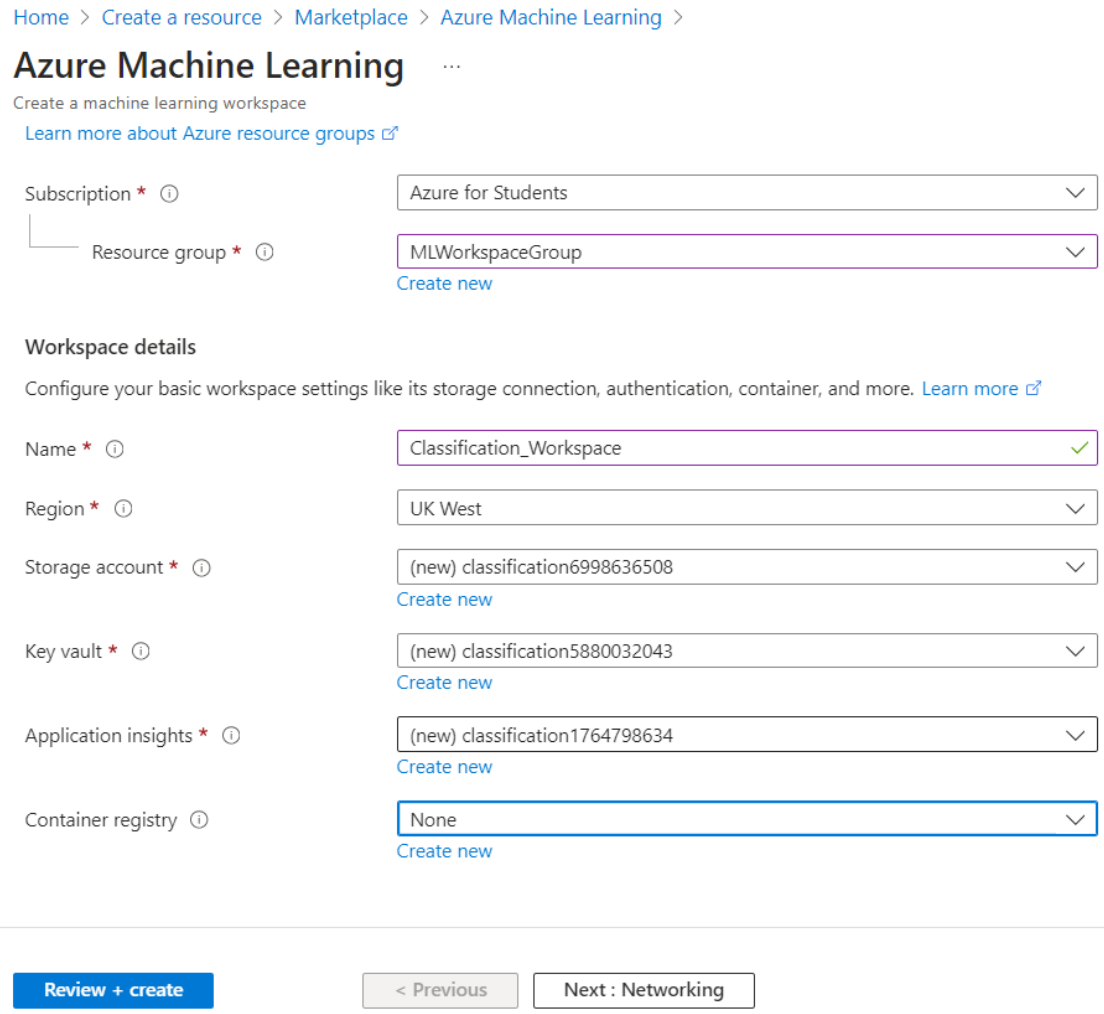


Figure 30: Azure Machine Learning Workspace Setup

Azure Machine Learning Workspace named Classification_Workspace, is being created in the UK West region under the Azure for Students subscription.

Step 2: Create compute cluster

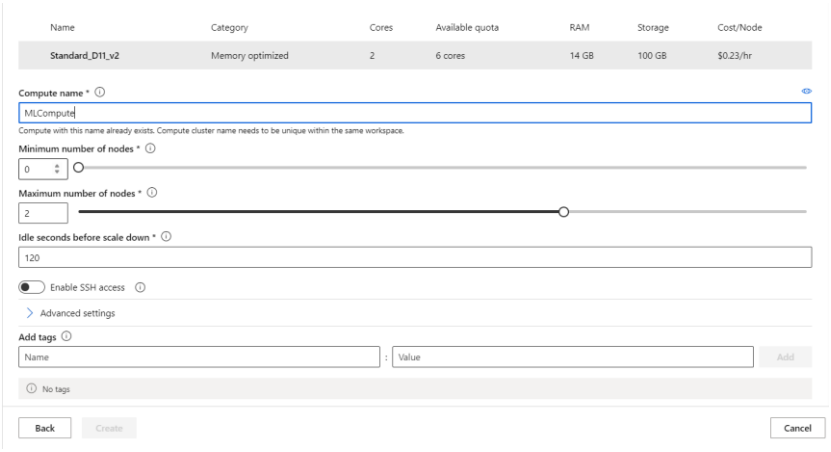


Figure 31: Compute Cluster Configuration

Setting up MLCompute with Standard_D11_v2 (2 cores, 14 GB RAM, 100 GB storage) at $0.23/hour per node, scalable from 0 to 2 nodes, with a 120-second idle timeout.

Step 3: Importing Data

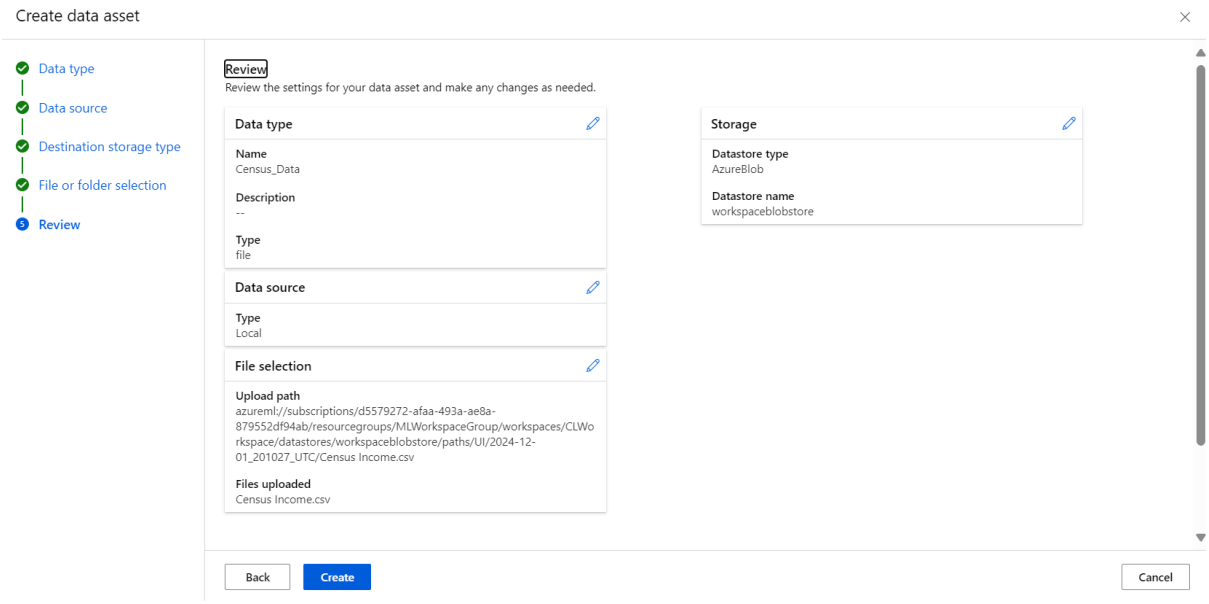


Figure 32: Data Asset Creation

The file is sourced locally and uploaded to an AzureBlob datastore named workspaceblobstore. The uploaded file is Census Income.csv

Step 4: Creating Pipeline

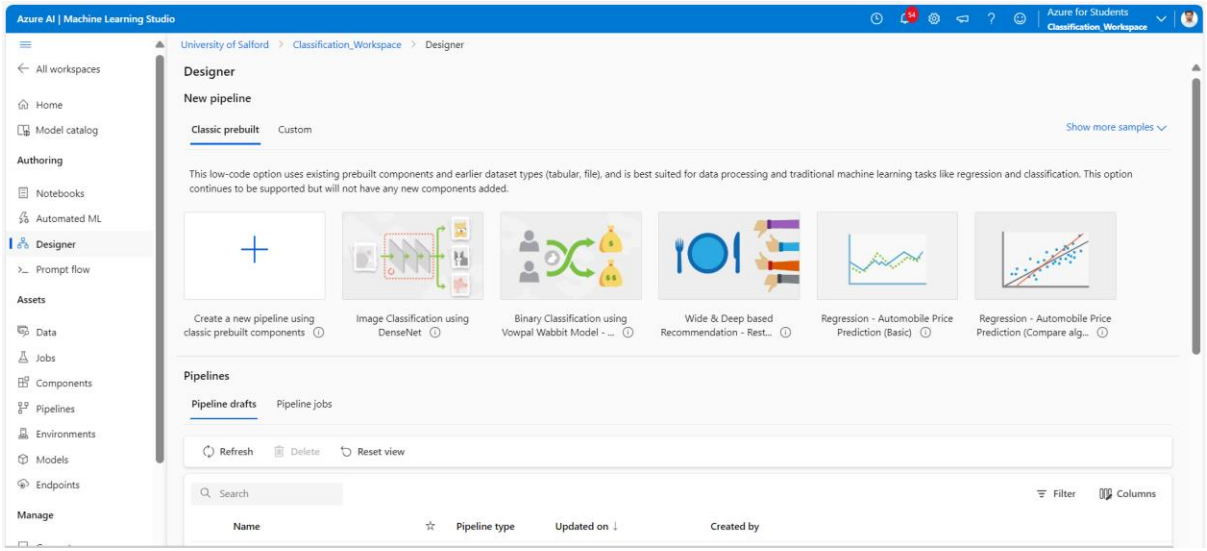


Figure 33: Designer Interface

Azure Machine Learning Designer interface for creating a new pipeline using prebuilt components.

Step 5: Preving data set in designer.

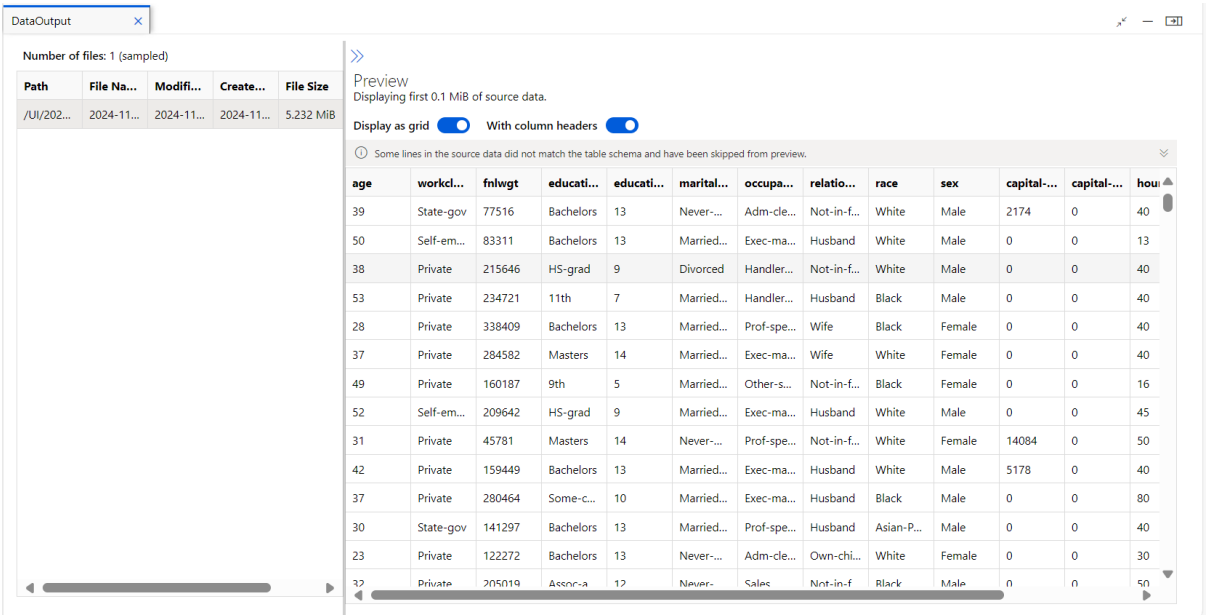


Figure 34: Data Set Preview

Preview of the Census Income Dataset, confirming the file is successfully uploaded

Step 6: Handling missing value

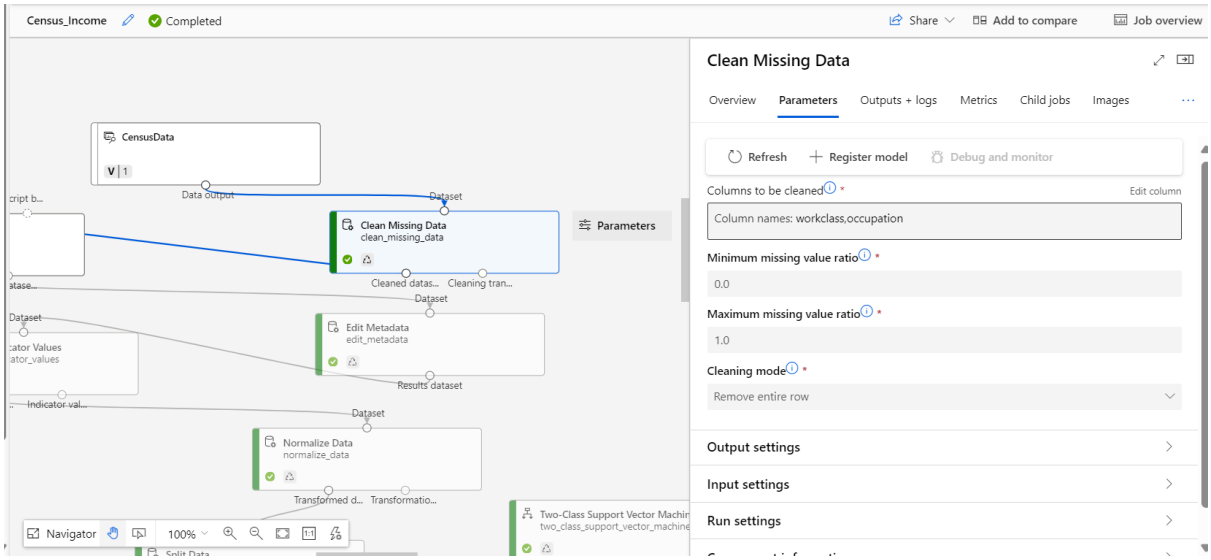


Figure 35: Clean Missing Data

Handling missing value in contain in the workclass and occupation columns. By removing entire row contain missing value.

Step 7: Encoding Target variable

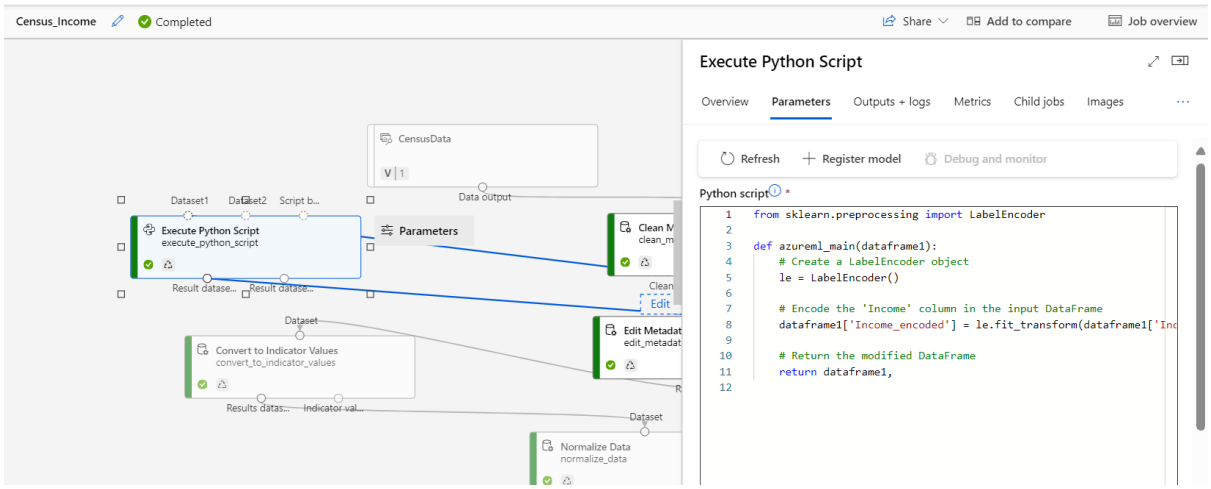


Figure 36: Execute Python Script

This module allows custom Python-based data transformations. In this specific case, it encodes a categorical column (Income) into numerical format to make it suitable for machine learning models.

Step 8: Modify metadata properties

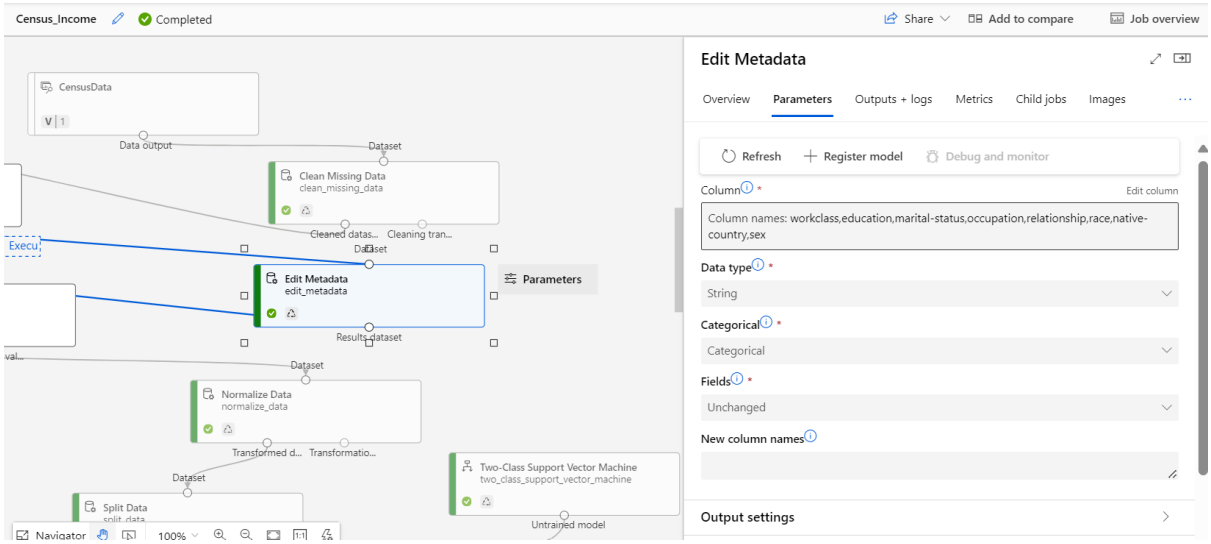


Figure 37: Edit Metadata

The Edit Metadata module prepares the dataset for machine learning by ensuring selected columns are correctly typed and categorized. This step is critical for subsequent operations like normalization, splitting, and model training.

Step 9: Encoding Independent Columns

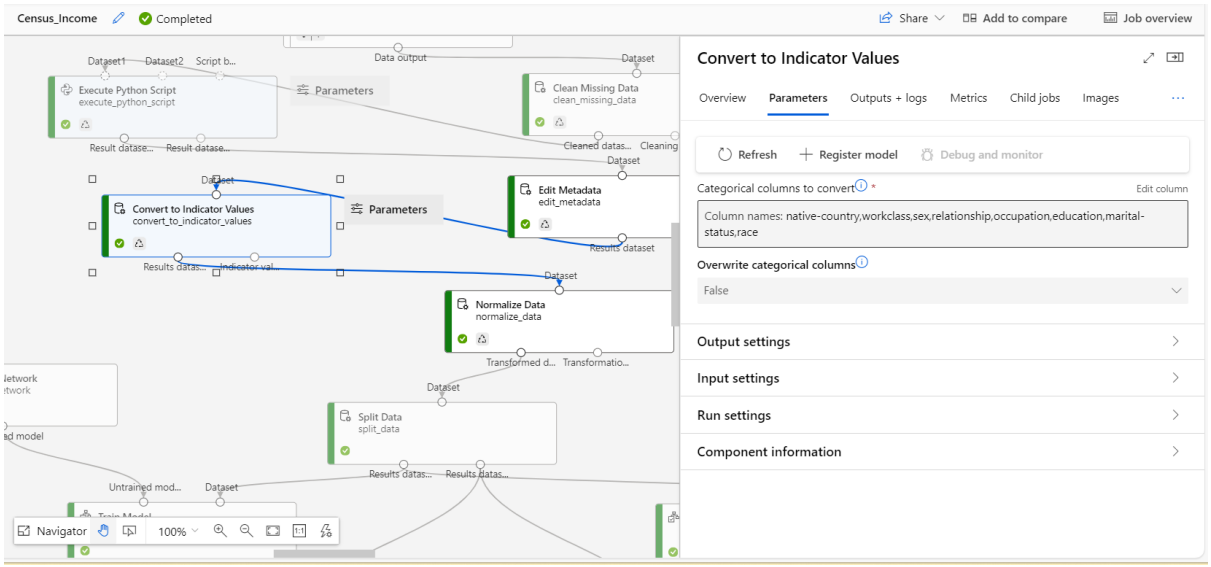


Figure 38: Convert to Indicator Values

This module ensures that categorical columns are encoded into numerical formats suitable for machine learning models. The output dataset can be directly fed into normalization and training steps, facilitating a smooth pipeline workflow.

Step 10: Scaling Non-Categorical Columns

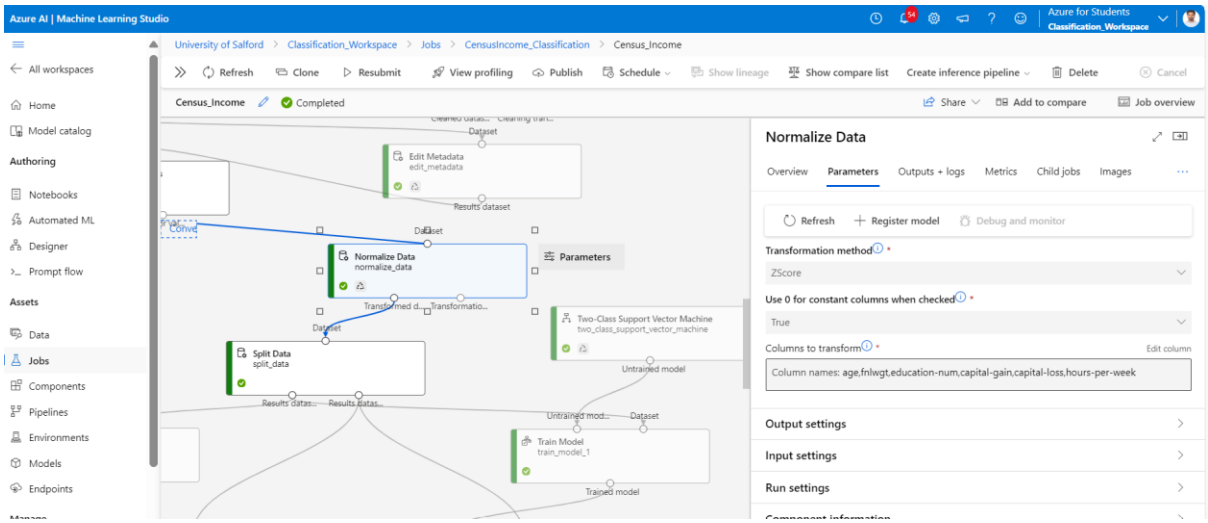


Figure 39: Normalize Data

The Normalize Data module is used to apply transformations to numerical columns in the dataset to standardize or scale the data, which is a critical preprocessing step for many machine learning algorithms.

Step 11: Splitting data into train and test set

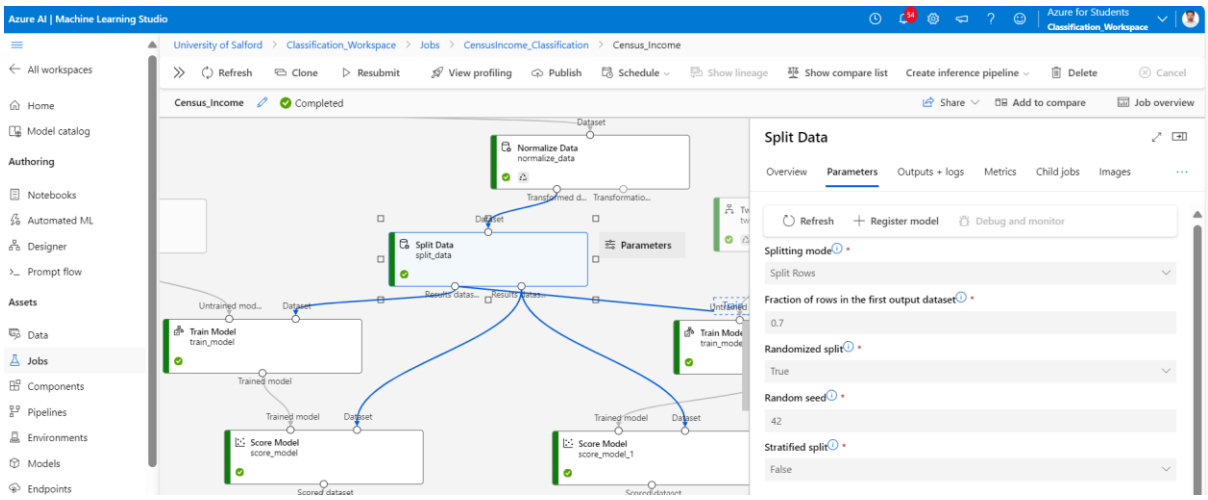


Figure 40: Split Data

This module is used to divide the dataset into training and testing sets. The training set is used to train the machine learning model, and the testing set is used to evaluate the model's performance. Randomization and a fixed seed ensure consistent and reproducible splits.

1.8 Neural Network

Step 12: Apply Neural Network

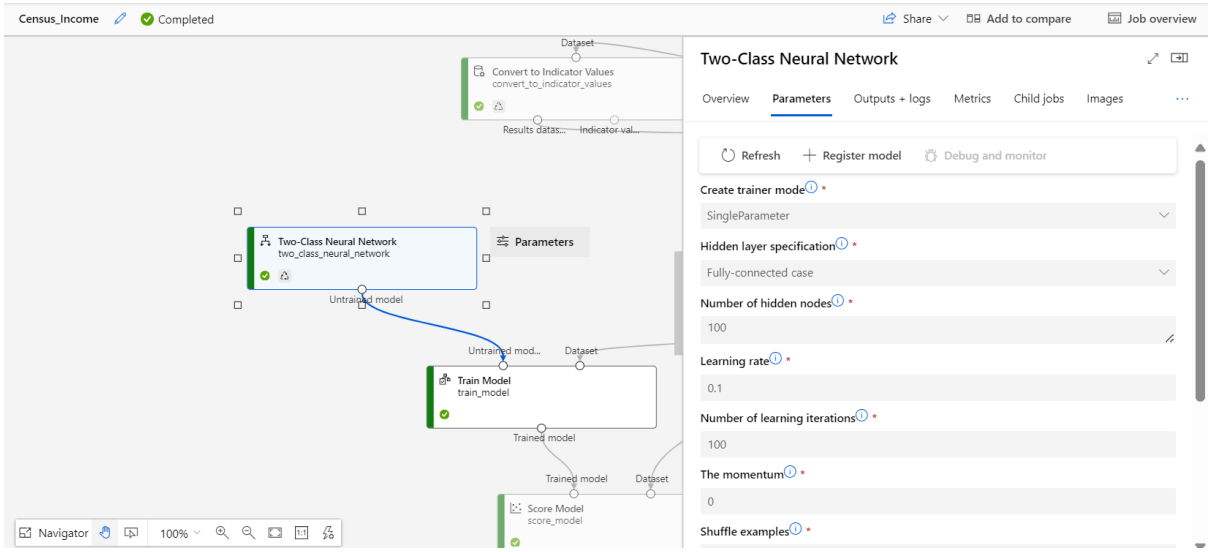


Figure 41: Two-Class Neural Network

The Two-Class Neural Network is specifically designed for binary classification, where the target variable has only two classes (e.g., Income > 50k or Income ≤ 50k). In your case, the target column Income_encoded represents two classes, making this algorithm a suitable choice.

Step 13: Train Neural Network

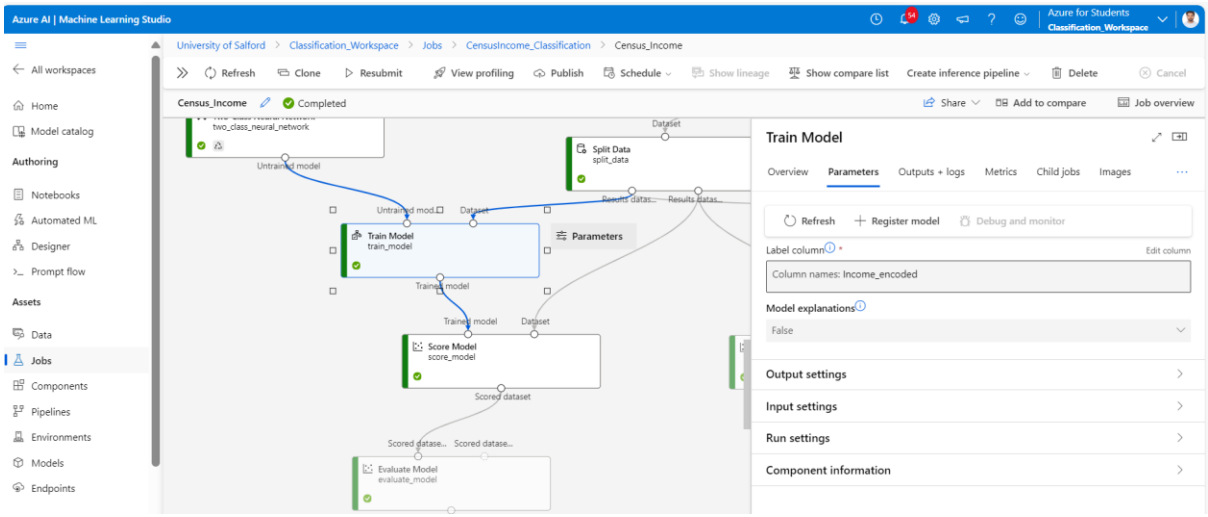


Figure 42: Train Model (Neural Network)

Combines the training dataset and the untrained model to generate a trained machine learning model.

Step 14: Neural Network is applied to the testing dataset to generate predictions

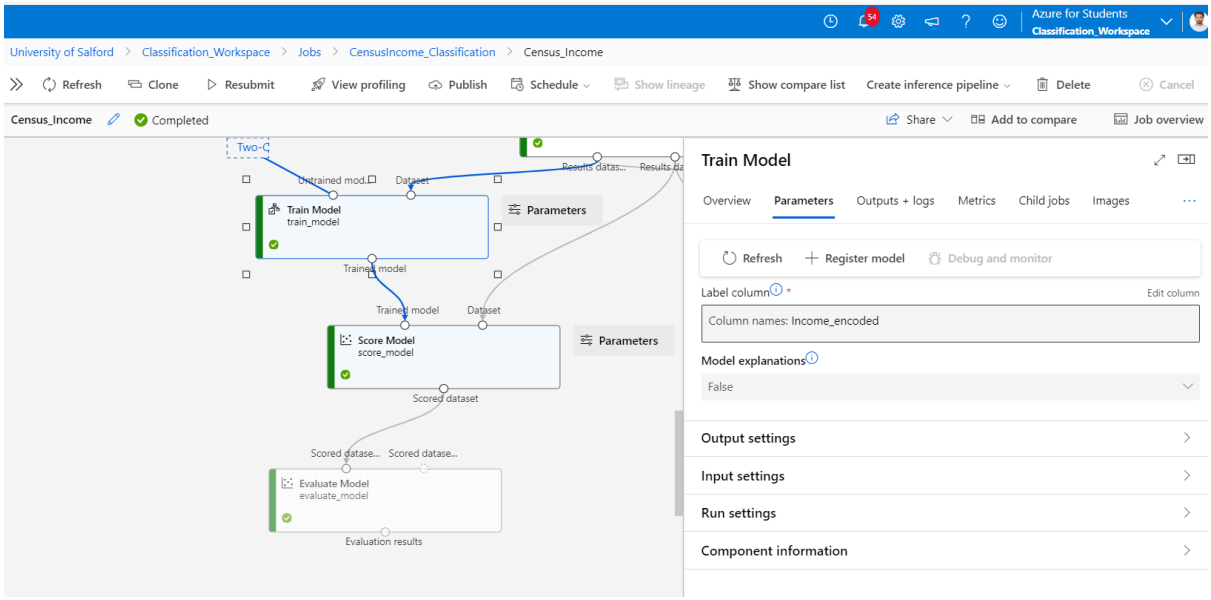
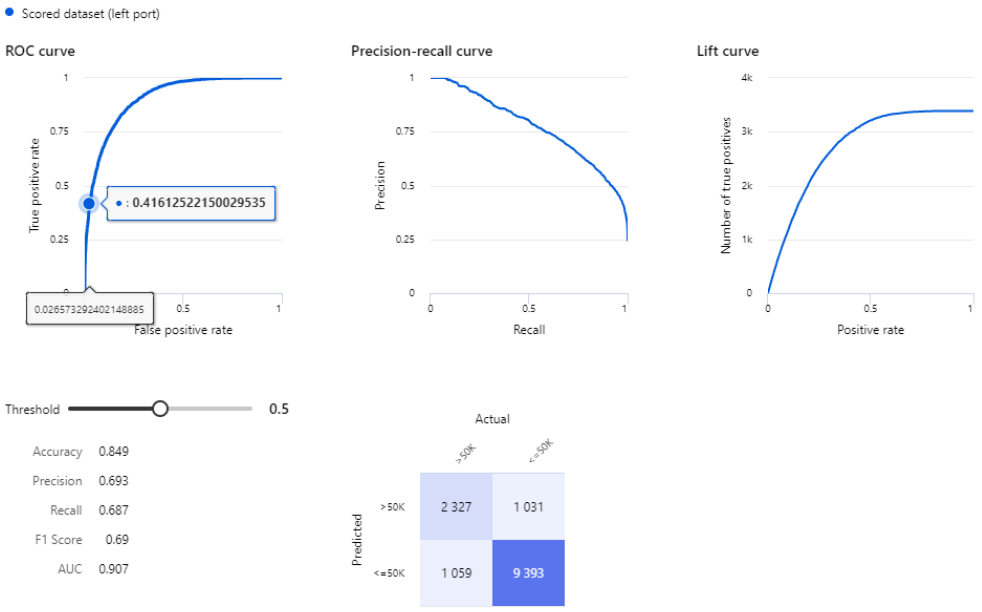


Figure 43: Score Model (Neural Network)

This step ensures that the performance of the Neural Network model can be thoroughly assessed against the actual outcomes in the testing data.

Step 15: Evaluation Metrics (Neural Network)



Score bin ↓	Positive exam...	Negative exam...	Fraction above thres...	Accura...	F1 Score	Precisi...	Recall	Negative precis...	Negative recall	Cumulative AUC
(0.900,1.000]	626	29	0.047	0.798	0.310	0.956	0.185	0.790	0.997	0.000
(0.800,0.900]	516	151	0.096	0.824	0.485	0.864	0.337	0.820	0.983	0.004
(0.700,0.800]	493	206	0.146	0.845	0.605	0.809	0.483	0.851	0.963	0.013
(0.600,0.700]	350	278	0.192	0.850	0.658	0.749	0.586	0.874	0.936	0.027
(0.500,0.600]	342	367	0.243	0.849	0.690	0.693	0.687	0.899	0.901	0.050
(0.400,0.500]	277	483	0.298	0.834	0.694	0.632	0.769	0.919	0.855	0.084
(0.300,0.400]	253	599	0.360	0.809	0.684	0.575	0.844	0.940	0.797	0.130
(0.200,0.300]	197	730	0.427	0.770	0.658	0.518	0.902	0.958	0.727	0.191
(0.100,0.200]	195	1129	0.523	0.702	0.613	0.450	0.960	0.979	0.619	0.292
(0.000,0.100]	137	6452	1.000	0.245	0.394	0.245	1.000	1.000	0.000	0.907

Figure 44: Evaluate Model (Neural Network)

Evaluation metrics for the Neural Network model applied to the dataset: Key performance metrics include **Accuracy (0.8487)**, **AUC (0.906)**, **F1 Score (0.690)**, **Precision (0.693)**, and **Recall (0.687)**, along with a confusion matrix for detailed results. These metrics highlight the Neural Network model's classification effectiveness.

- **Lift Curve**: Demonstrates the positive rate of true positives across different thresholds.
- **Precision-Recall Curve**: Highlights the trade-off between precision and recall at various decision thresholds.
- **ROC Curve**: Plots the true positive rate versus the false positive rate, providing insights into the model's ability to distinguish between classes.

1.9 Support Vector Machine

Step 16: Apply Support Vector Machine

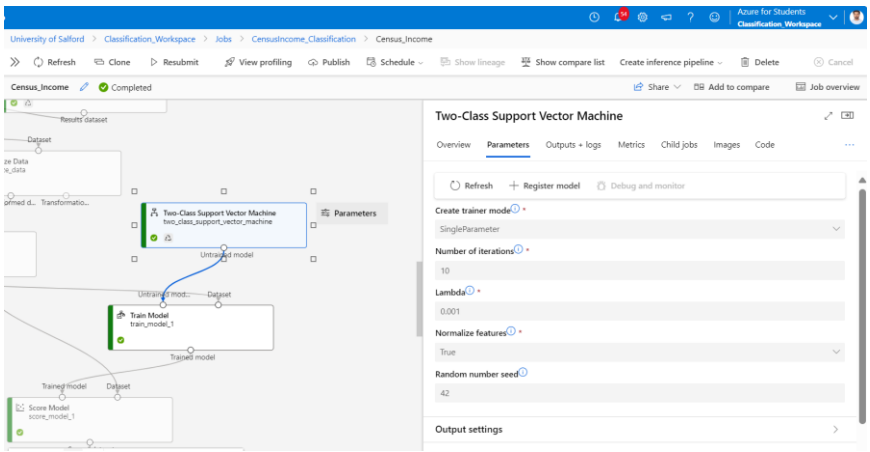


Figure 45: Two-Class Support Vector Machine

The Two-Class Support Vector Machine module is set up to train a binary classification SVM model. In the dataset, it uses labelled training data to generate a hyperplane that best divides the two groups.

Step 17: Train Support Vector Machine

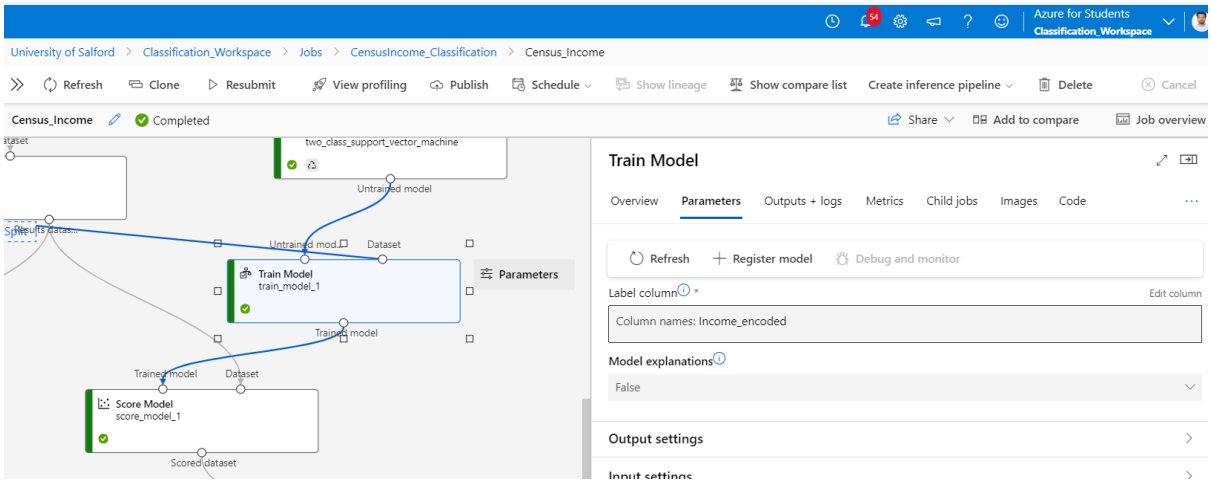


Figure 46: Train Model (Support Vector Machine)

The Train Model module is used to fit the Two-Class SVM model to the training dataset. This step applies the algorithm's configuration to the labelled data to produce a trained model for evaluation.

Step 18: Support Vector Machine is applied to the testing dataset to generate predictions

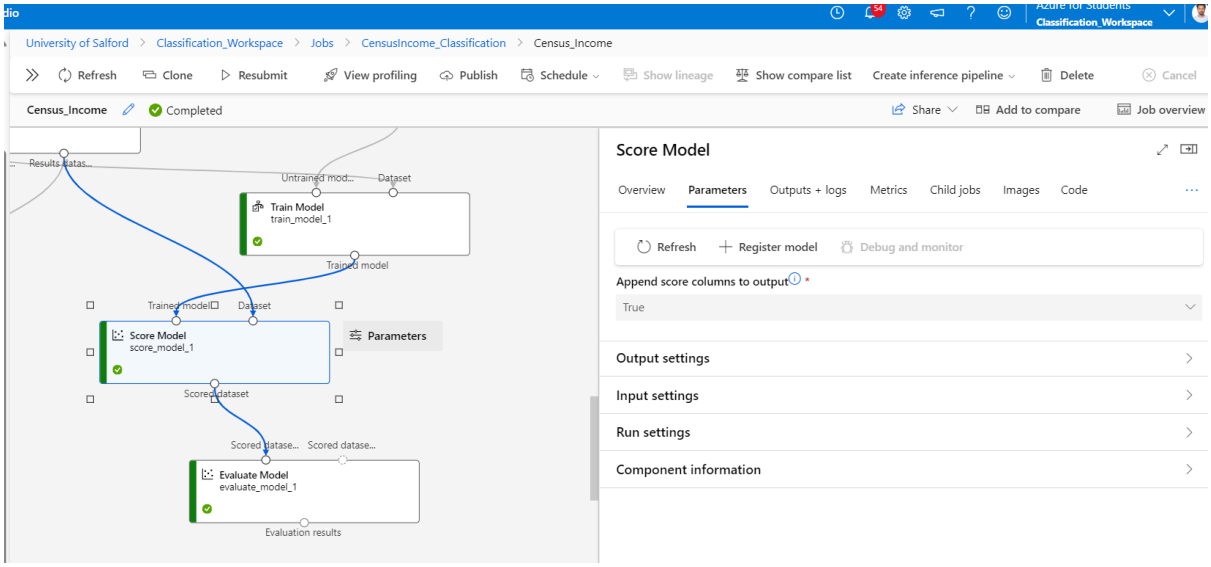
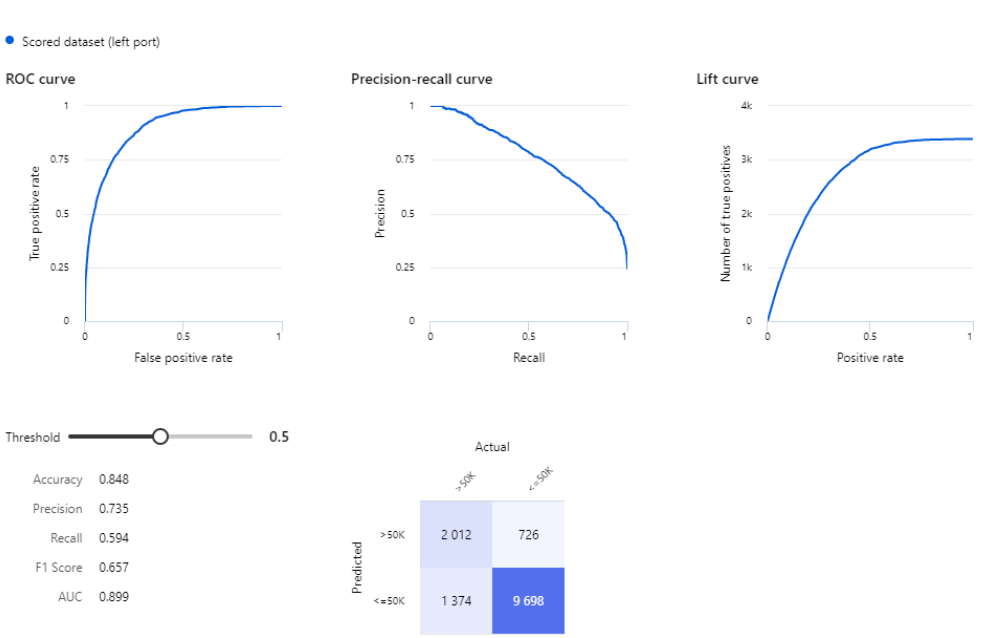


Figure 47: Score Model (Support Vector Machine)

The Score Model step applies the trained Support Vector Machine model to the testing dataset to generate predictions.

Step 19: Evaluation Metrics (Support Vector Machine)



Score bin ↓	Positive exam...	Negative exam...	Fraction above thres...	Accura...	F1 Score	Precisi...	Recall	Negative precis...	Negative recall	Cumulative AUC
(0.900,1.000]	111	0	0.008	0.763	0.063	1.000	0.033	0.761	1.000	0.000
(0.800,0.900]	147	3	0.019	0.773	0.141	0.989	0.076	0.769	1.000	0.000
(0.700,0.800]	209	10	0.035	0.788	0.242	0.973	0.138	0.781	0.999	0.000
(0.600,0.700]	543	113	0.082	0.819	0.447	0.889	0.298	0.813	0.988	0.003
(0.500,0.600]	1002	600	0.198	0.848	0.657	0.735	0.594	0.876	0.930	0.030
(0.400,0.500]	840	1570	0.373	0.795	0.668	0.554	0.842	0.938	0.780	0.142
(0.300,0.400]	425	2392	0.577	0.653	0.577	0.411	0.968	0.981	0.550	0.353
(0.200,0.300]	101	3308	0.824	0.420	0.458	0.297	0.998	0.997	0.233	0.666
(0.100,0.200]	8	2325	0.993	0.253	0.396	0.247	1.000	1.000	0.010	0.889
(0.000,0.100]	0	103	1.000	0.245	0.394	0.245	1.000	1.000	0.000	0.899

Figure 48: Evaluate Model

Evaluation metrics for the **Support Vector Machine (SVM)** model applied to the dataset. Key performance metrics include **Accuracy (0.8479)**, **AUC (0.899)**, **F1 Score (0.657)**, **Precision (0.734)**, and **Recall (0.594)**, along with a confusion matrix for detailed results. These metrics reflect the SVM model's classification performance.

Lift Curve: Demonstrates the positive rate of true positives across thresholds.
Precision-Recall Curve: Shows the trade-off between precision and recall at various thresholds.
ROC Curve: Illustrates the true positive rate versus the false positive rate, indicating the model's ability to distinguish between classes.

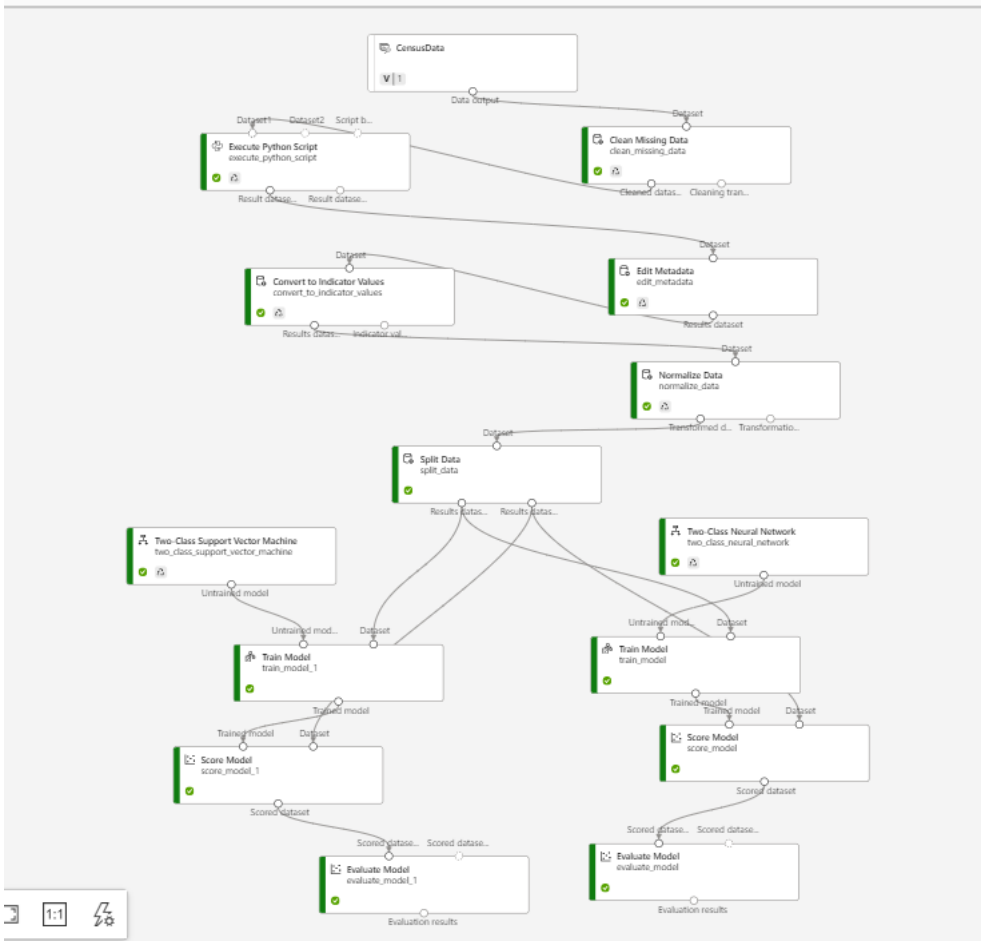


Figure 49: Azure Machine Learning Pipeline

Azure machine learning pipeline starting with a dataset called "CensusData." It involves preprocessing steps like executing a Python script, cleaning missing data, converting categorical data, editing metadata, and normalizing data. The data is then split for training and evaluation. Two models are trained: a Support Vector Machine (SVM) and a Neural Network. Finally, both models are scored and evaluated to determine which performs better.

1.10 Comparison of Naive Bayes and Support Vector Machine (SVM) Models:

The Neural Network has a better balance of Precision and Recall, with a higher F1 Score and AUC, making it more robust for the dataset.

The SVM achieves higher Precision but sacrifices Recall, which might lead to underperformance in identifying true positives.

1.11 Business Impact:

Businesses, social organizations, and policymakers can all gain a lot from the Random Forest model's insights. For instance, authorities may be able to create focused interventions because to the model's capacity to pinpoint important variables like hours worked and educational attainment. In one scenario, government measures to offer accessible education programs and vocational training could be prompted by the identification of a group with low income and little educational attainment.

In a similar vein, companies can customize their offerings according to income levels. To promote inclusion, businesses could, for example, develop reasonably priced goods or choices for instalment payments for those in the lower income range. However, in order to maximize profit prospects, enterprises could concentrate on luxury or premium products for high-income people.

1.12 Real-World Applications:

Through the utilization of machine learning models such as Random Forest, companies may develop and execute focused strategies that enhance performance, optimize resources, and foster equity. In addition to accomplishing certain corporate objectives like higher revenue, improved customer happiness, and improved societal consequences, these practical suggestions offer a framework for tackling income inequality.

1.13 Conclusions:

The performance of two machine learning models, Random Forest, and K-Nearest Neighbors (K-NN), on a dataset used to categorize income levels (<=50K and >50K) has been examined in this research. Measures including accuracy, precision, recall, F1-score, and confusion matrix were the main drivers of the comparison. The original test set was used to test both models after they were trained on an oversampled dataset to account for class imbalance.

The recommended model for this classification job is Random Forest. In addition to making accurate forecasts, it also provides important new information about the socioeconomic variables influencing income inequality. For companies, legislators, and researchers looking to successfully combat income disparity, this makes it a priceless instrument.

1.14 References:

U.S. Census Bureau. (1996, April 30). *Census income dataset* [Data set]. UCI Machine Learning Repository. Retrieved November 30, 2024, from https://archive.ics.uci.edu/dataset/20/census+income

GeeksforGeeks. (n.d.). Encoding categorical data in sklearn. Retrieved November 30, 2024, from https://www.geeksforgeeks.org/encoding-categorical-data-in-sklearn/

Brownlee, J. (2020, August 26). Train-test split for evaluating machine learning algorithms. *Machine Learning Mastery.* Retrieved November 30, 2024, from https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

Singh, H. (2024, November 18). 10 techniques to solve imbalanced classes in machine learning. *Analytics Vidhya.* Retrieved November 30, 2024, from https://www.analyticsvidhya.com/articles/class-imbalance-in-machine-learning/

Brownlee, J. (2020, August 15). K-nearest neighbors for machine learning. *Machine Learning Mastery.* Retrieved November 30, 2024, from https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/

GeeksforGeeks. (n.d.). Random forest algorithm in machine learning. Retrieved November 30, 2024, from https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

Task 2: Clustering

2.0 Title:

Cluster Analysis of Obesity for Better Health Insights

2.1 Introduction:

Obesity is a serious growing public health problem, affecting millions of people around the world. Understanding the factors which are contribute to obesity can assist in making change in lifestyle to get prevention from this decease. In this report, we aim to apply clustering techniques to group individuals based on their lifestyle factors, family background, habit, etc. which can provide us insights on different behavioural patterns associated with obesity. The research question guiding this analysis are: "What are the key clusters of individuals based on lifestyle behaviours, and how can these clusters provide us actionable insights into managing obesity?" This clustering analysis is conducted using the **Estimation of Obesity Levels Based On Eating Habits and Physical Condition**, data set available on UCI **repository**. Which contains various lifestyle features related to diet, physical activity, and personal habits, etc.

2.2 Dataset:

The dataset for this study was obtained from the UCI Machine Learning Repository. Specifically, the 'Estimation of Obesity Levels Based on Eating Habits and Physical Condition' dataset was used, which provides comprehensive data on obesity levels derived from various eating habits and physical activity parameters **(Dua & Graff, 2019)**.

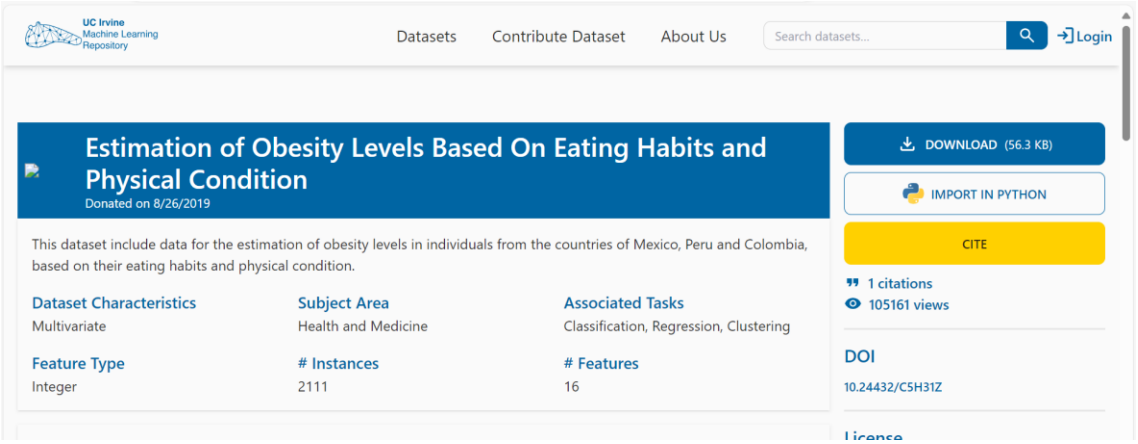


Figure 50: Obesity data set

It contains information on various lifestyle and physical characteristics of individuals, such as gender, age, height, weight, eating habits, and physical activity levels. The target variable is "NObeyesdad," which classifies individuals into different obesity categories, ranging from underweight to morbidly obese. The dataset contains 17 columns and 2111 rows, with a mix of categorical and numerical features.

The data set contain following columns: The name of the columns and the description of the columns.

- **Gender:** The gender of the individual (Male/Female).
- **Age:** Age of the individual (in years).
- **Height:** Physical measurements of the individual in meters
- **Weight:** Physical measurements of the individual in kilograms
- **family_history_with_overweight**: Has a family member suffered or suffers from overweight?
- **FAVC:** Do you eat high caloric food frequently?
- **FCVC:** Do you usually eat vegetables in your meals?
- **NCP:** How many main meals do you have daily?
- **CAEC:** Do you eat any food between meals?
- **SMOKE:** Do you smoke?
- **CH2O:** How much water do you drink daily?
- **SCC:** Do you monitor the calories you eat daily?
- **FAF:** How often do you have physical activity?
- **TUE:** How much time do you use technological devices such as cell phone, videogames, television, computer and others?
- **CALC:** How often do you drink alcohol?
- **MTRANS:** Which transportation do you usually use?
- **NObeyesdad:** Obesity level

2.3 Ethical, Social, or legal Issues::

This project uses the dataset exclusively for academic purposes to demonstrate clustering methodologies, ensuring alignment with ethical research practices. As this project is based on a publicly available dataset, it does not require additional ethical approval. The usage strictly adheres to the repository’s guidelines.

2.4 Exploratory Data Analysis (EDA) and Data Preprocessing:

To understand the dataset better in detail we have conducted exploratory data analysis using summary statistics and visualizations.

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.datasets import make_blobs
```

Figure 51: Importing Libraries

Importing following libraries for Clustering and Data Preprocessing in Python.

Exploratory Data Analysis (EDA)

```
[ ]  #Importing the dataset and remove the "NObeyesdad" column.
     df = pd.read_csv("Obesity_Data.csv").drop(columns=["NObeyesdad"])
```

Figure 52: Importing Data Set

Loading dataset in using Pandas' read_csv function. Since clustering is an unsupervised learning method and does not rely on dependent variables, we removed the NObeyesdad column from the dataset.

```
# Display basic information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
dtypes: float64(8), object(8)
memory usage: 264.0+ KB
```

Figure 53: Display Basis Information

The info() method to display basis information of the data frame, we see data set contain 2111 rows,16 columns, columns name, column data type and non-null count.

```
# Display the first 5 rows of the dataset
df.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation

Figure 54: Printing Head of the Data Set

We have use head() method to display first five rows of the data set. We can compare the data set without actual csv file.

```
# Display basic statistics of numeric cloumns of the dataset
df.describe(include="all")
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS
count	2111	2111.000000	2111.000000	2111.000000	2111	2111	2111.000000	2111.000000	2111	2111	2111.000000	2111	2111.000000	2111.000000	2111	2111
unique	2	NaN	NaN	NaN	2	2	NaN	NaN	4	2	NaN	2	NaN	NaN	4	5
top	Male	NaN	NaN	NaN	yes	yes	NaN	NaN	Sometimes	no	NaN	no	NaN	NaN	Sometimes	Public_Transportation
freq	1068	NaN	NaN	NaN	1726	1866	NaN	NaN	1765	2067	NaN	2015	NaN	NaN	1401	1580
mean	NaN	24.312600	1.701677	86.586058	NaN	NaN	2.419043	2.685628	NaN	NaN	2.008011	NaN	1.010298	0.657866	NaN	NaN
std	NaN	6.345968	0.093305	26.191172	NaN	NaN	0.533927	0.778039	NaN	NaN	0.612953	NaN	0.850592	0.608927	NaN	NaN
min	NaN	14.000000	1.450000	39.000000	NaN	NaN	1.000000	1.000000	NaN	NaN	1.000000	NaN	0.000000	0.000000	NaN	NaN
25%	NaN	19.947192	1.630000	65.473343	NaN	NaN	2.000000	2.658738	NaN	NaN	1.584812	NaN	0.124505	0.000000	NaN	NaN
50%	NaN	22.777890	1.700499	83.000000	NaN	NaN	2.385502	3.000000	NaN	NaN	2.000000	NaN	1.000000	0.625350	NaN	NaN
75%	NaN	26.000000	1.768464	107.430682	NaN	NaN	3.000000	3.000000	NaN	NaN	2.477420	NaN	1.666678	1.000000	NaN	NaN
max	NaN	61.000000	1.980000	173.000000	NaN	NaN	3.000000	4.000000	NaN	NaN	3.000000	NaN	3.000000	2.000000	NaN	NaN

Figure 55: Display Basic Statistics

The describe(include = “all”) method display all the basis statistical value of the numerical columns of the data set.

```
# Check for missing values in each column of the DataFrame
# The isnull() method identifies missing (NaN) values
# The sum() method calculates the total number of missing values for each column
df.isnull().sum()
```

	0
Gender	0
Age	0
Height	0
Weight	0
family_history_with_overweight	0
FAVC	0
FCVC	0
NCP	0
CAEC	0
SMOKE	0
CH2O	0
SCC	0
FAF	0
TUE	0
CALC	0
MTRANS	0

Figure 56: Checking missing value

Applied isnull() method to the data frame to check missing value in the data set. Sum() is used to count the number of the missing values in the data frame. However, there is no missing values in the data frame.

```
# Display the dimensions of the Data set
df.shape
```

```
(2111, 16)
```

Figure 57: Checking Data Set Dimensions

Display the dimensions of the data set, with the help of shape method which gives the result of the numbers of row and columns contain in our data frame.

```
[78] # Create a count plot to visualize the distribution of people based on their smoking habits
     sns.countplot(x='SMOKE', data=df)
     plt.title(" Distribution of people by their smoking habit")
     plt.show()
```

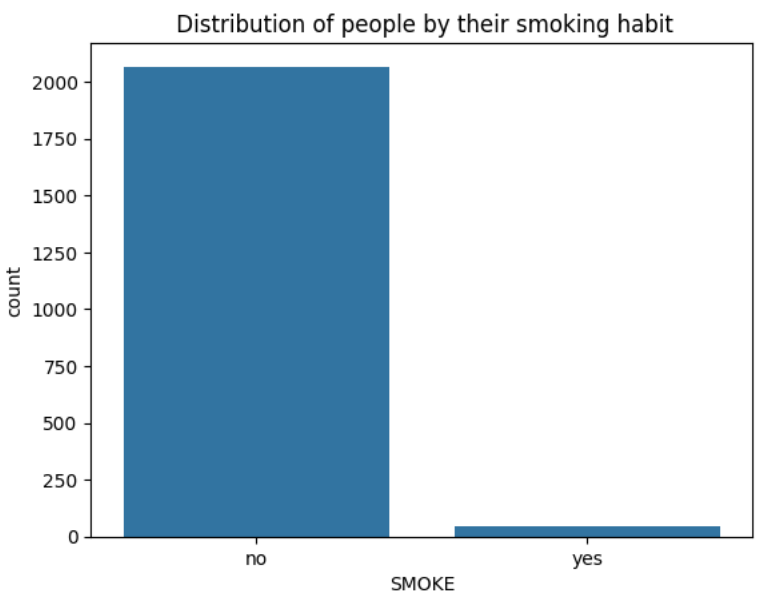


Figure 58: Distribution of People by their Smoking Habit

The count plot clearly shows that, majority of the people are not smoking and only few people are having smoking habit.

```
# Plot a histogram to visualize the age distribution in the dataset with 20 bins
sns.histplot(x='Age', data=df,bins = 20)
plt.title("Age Distribution")
plt.xlabel('Age')
plt.ylabel("Count")
plt.show()
```

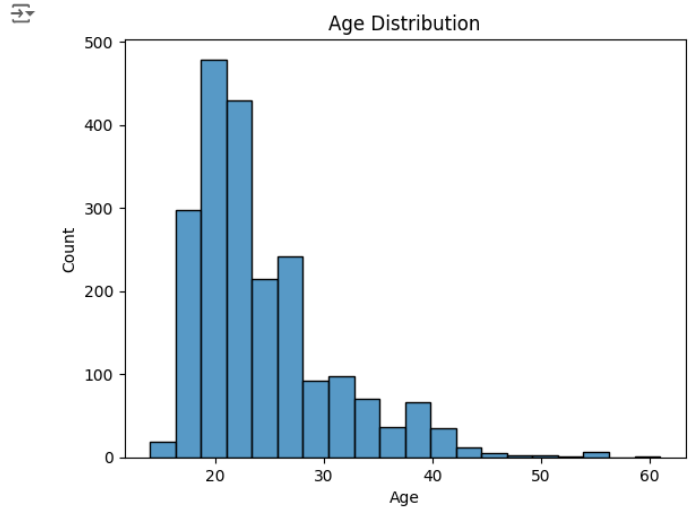


Figure 59: Distribution of the People Based on their Age

The histogram clearly mention that majority of the people in the data set are fall between 15 to 25 years.

```
[80] # Create a histogram to visualize the count of smokers by age.
     # The 'hue' parameter distinguishes data based on the 'SMOKE' column.
     sns.histplot(data=df, x='Age', hue='SMOKE', bins=10, discrete=True)
     plt.title("Count of Smokers by Age")
     plt.xlabel('Age')
     plt.ylabel("Count")

     plt.show()
```

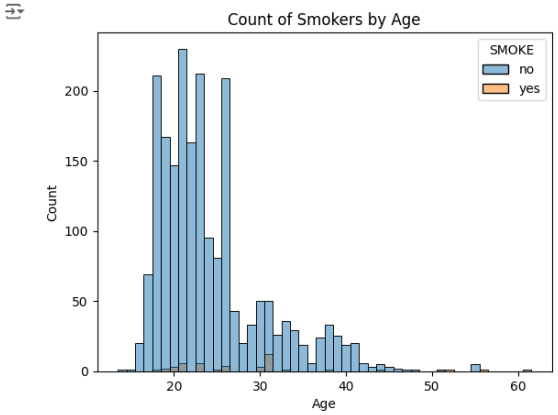


Figure 60: Distribution of people based on their age and smoking habit.

As per the histplot people with more than 30 years age and less than 34 years age smoke the highest.

```
# Create a count plot to visualize the distribution of genders in the dataset
sns.countplot(x='Gender', data=df)
plt.title(" Distribution of Gender")
plt.show()
```

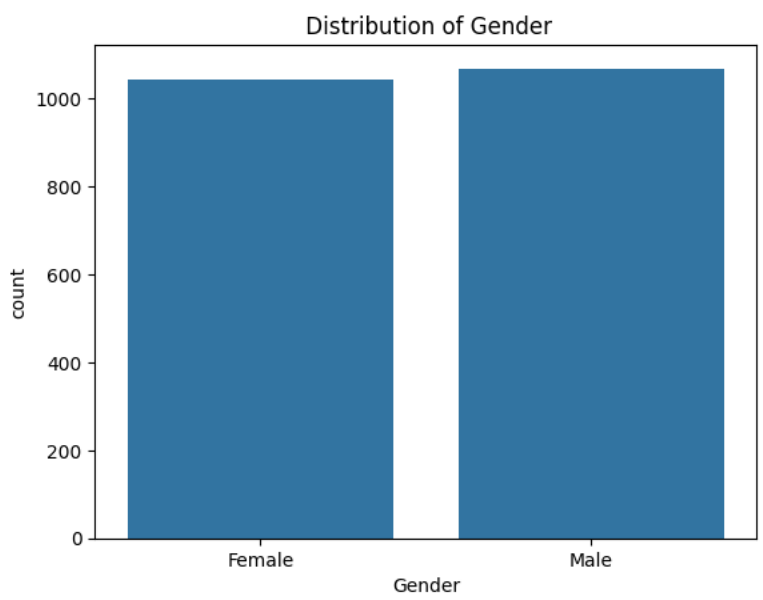


Figure 61: Distribution of the Gender

The data frame, consisting of gender almost equally distributed among male and female.

Data Pre-processing

Encoding categorical data

```
[82] # Retrieve and display all column names of the DataFrame.
     cols = df.columns
     cols
```

```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
       'CALC', 'MTRANS'],
      dtype='object')
```

x

Figure 62: Extracting the name of the columns

Retrieving the column names from your dataset and displays them, it will help us to separate numerical columns and categorical columns. The “cols” in the code stands columns.

```
[83] # Extract and display the names of all numeric columns in the DataFrame.
     num_cols = df._get_numeric_data().columns
     num_cols

Index(['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE'], dtype='object')
```

Figure 63: Extracting numerical columns

The get_numeric_data function of the pandas, return only those columns in the data frame that contain numeric data types like integers or floats. The “num_cols” stands for numeric columns in the data frame.

```
[84] # Extracting the categorical columns by deducting the numerical columns from the list of column names.
     cat_cols = list(set(cols) - set(num_cols))
     cat_cols

['SCC',
 'family_history_with_overweight',
 'MTRANS',
 'FAVC',
 'CAEC',
 'Gender',
 'SMOKE',
 'CALC']
```

Figure 64: Extracting categorical columns

We have subtracted the numeric columns from the list of the all the columns which gives us the result of the list containing only categorical columns. The “cat_cols” in code refers to the categorical columns.

```
+ Code  + Text

[85] # Define a list of categorical columns that will undergo label encoding.
     le_cat_cols =  ["Gender", "family_history_with_overweight", "FAVC", "SMOKE", "SCC"]
     le_cat_cols

['Gender', 'family_history_with_overweight', 'FAVC', 'SMOKE', 'SCC']

[86] # Define a list of columns that will undergo One-Hot encoding.
     ohe_cat_cols = list(set(cat_cols) - set(le_cat_cols))
     ohe_cat_cols

['MTRANS', 'CAEC', 'CALC']
```

Figure 65: Dividing categorical columns

Separated categorical columns into 2 variables based on their values. The assigned "Gender", "family_history_with_overweight", "FAVC", "SMOKE", "SCC" all these columns to “le_cat_cols” stands for Lebel encoding categorical columns and rest of the columns assigned to “ohe_cat_cols” stands for one hot encoding categorical columns.

```
# Perform One-Hot Encoding on the categorical columns identified in 'ohe_cat_cols'.
ohe_encoded = pd.get_dummies(df, columns=ohe_cat_cols)

# Replace boolean values (True/False) with integers (1/0) in the resulting DataFrame.
df = ohe_encoded.replace({True: 1, False: 0})
```

```
<ipython-input-87-5ba4d6f135c3>:5: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed
  df = ohe_encoded.replace({True: 1, False: 0})
```

```
[88] # Apply Label encoding to ("Gender", "family_history_with_overweight", "FAVC", "SMOKE", "SCC") columns in the data fr
label_encoder = LabelEncoder()
for col in le_cat_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

Figure 66: Categorical Feature Encoding

Applied label encoder and one hot encoder to convert non numerical values into numerical values, which make data suitable for our machine learning clustering task. Label encoder method applied to those columns having small number of unique values such as yes and no. While one hot encoding applied to those columns which does not contain inherent order for example gender, city, etc.

```
[89] # Display the first five rows of the DataFrame after encoding the data.
df.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	SMOKE	CH2O	...	MTRANS_Public_Transportation	MTRANS_Walking	CAEC_Always	CAEC_Frequently	CAEC_Sometimes	CAEC_no	CALC_Always	CALC_Frequently	CALC_Sometimes	CALC_no
0	0	21.0	1.62	64.0	1	0	2.0	3.0	0	2.0	...	1	0	0	0	1	0	0	0	0	1
1	0	21.0	1.52	56.0	1	0	3.0	3.0	1	3.0	...	1	0	0	0	1	0	0	0	1	0
2	1	23.0	1.80	77.0	1	0	2.0	3.0	0	2.0	...	1	0	0	0	1	0	0	1	0	0
3	1	27.0	1.80	87.0	0	0	3.0	3.0	0	2.0	...	0	1	0	0	1	0	0	1	0	0
4	1	22.0	1.78	89.8	0	0	2.0	1.0	0	2.0	...	1	0	0	0	1	0	0	0	1	0

5 rows × 26 columns

```
[90] # Display the dimensions of the Data set after doing encoding
df.shape
```

```
(2111, 26)
```

Figure 67: Encoded data frame

By displaying the head and shape of the data frame, we can cross check the implementation of encoding. Male and female change to 1 and 0. Moreover number of rows increase to 26.

Feature Scaling

```
[91] # Applied StandardScaler on the numerical values of the data set.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])
```

```
[92] # Display first five rows of the data frame after appling StandardScaler.
df.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	SMOKE	CH2O	...	MTRANS_Public_Transportation	MTRANS_Walking	CAEC_Always	CAEC_Frequently	CAEC_Sometimes	CAEC_no	CALC_Always	CALC_Frequently	CALC_Sometimes	CALC_no
0	0	-0.522124	-0.875589	-0.862558	1	0	-0.785019	0.404153	0	-0.013073	...	1	0	0	0	1	0	0	0	0	1
1	0	-0.522124	-1.947599	-1.168077	1	0	1.088342	0.404153	1	1.618759	...	1	0	0	0	1	0	0	0	1	0
2	1	-0.206889	1.054029	-0.366090	1	0	-0.785019	0.404153	0	-0.013073	...	1	0	0	0	1	0	0	1	0	0
3	1	0.423582	1.054029	0.015808	0	0	1.088342	0.404153	0	-0.013073	...	0	1	0	0	1	0	0	1	0	0
4	1	-0.364507	0.839627	0.122740	0	0	-0.785019	-2.167023	0	-0.013073	...	1	0	0	0	1	0	0	0	1	0

5 rows × 26 columns

Figure 68: Feature scaling

Feature scaling is an important preprocessing step for machine learning clustering algorithm, especially when the dataset contains features with different ranges or

units and our data contains different ranges of units. Hence larger magnitudes can dominate the model's learning process, leading to suboptimal performance. So that we have applied feature scaling to scale the unit between 0 and 1 with the help of standard scaler. Feature scaling ensures that all features contribute equally to our model. It will also improve the performance of the algorithm like k-means clustering. Standard scalers standardize the features by mean of 0 and standard deviation of 1.

```
# Selecting specific columns from the dataframe and creating the pairplot using vars parameter
sns.pairplot(df, vars=["Gender", "Age", "Height", "Weight", "SMOKE"])
```

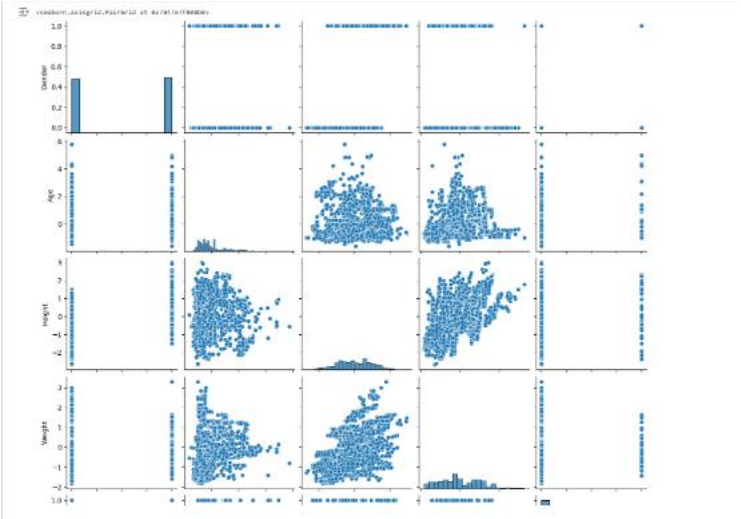


Figure 69: Pair plot of the specific columns

Display the pair plot of the gender, age, height, weight, and smoke to see the relationship between them.

Principal Component Analysis (PCA)

```
# Import PCA from scikit-learn library
from sklearn.decomposition import PCA

# Initialize PCA to reduce the dataset to 8 principal components
pca = PCA(n_components=8)

# Fit the PCA model to the data and transform the original dataset into 8 principal components
X_pca_8 = pca.fit_transform(df)

# Obtain the explained variance ratio for each of the 8 components
pca.explained_variance_ratio_
```

```
array([0.19713535, 0.16155745, 0.11495021, 0.1038871 , 0.09810488,
       0.08085375, 0.07537781, 0.04448615])
```

```
[ ] # Sum the total amount of explained_variance_ratio_
    sum(pca.explained_variance_ratio_)
```

```
0.876352704073352
```

Figure 70: Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique. We use this technique to simplify a dataset while retaining as much information as

possible. By applying pca our original data set transform into new set of uncorrelated features know as principal components.

“PCA is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance” - **(Scikit-learn, n.d.).** Reducing dimensionality help increase efficiency of the model, because working with the high dimensionality dataset contains lots of redundant information. Moreover, some information on the data set is not relevant, pca remove such noise form the dataset. Pca protect model from the overfitting issue as well.

The n_components we have selected is 8, because when we select the n components less the 8 the sum of total amount of explained variance ratio, we get is less than 80%. Which means, our selected principal components capture only a small proportion of the total variance of the original dataset which leads to potential information loss, less effective model performance, poor representation of the data, etc. Selecting 8 n components give us the 0.87% explained variance ratio which represents good variance ratio.

2.5 Clustering Model Implementation:

We applied two clustering algorithms to the dataset K-Means++ and Agglomerative Clustering.

2.5.1 K-Means++:

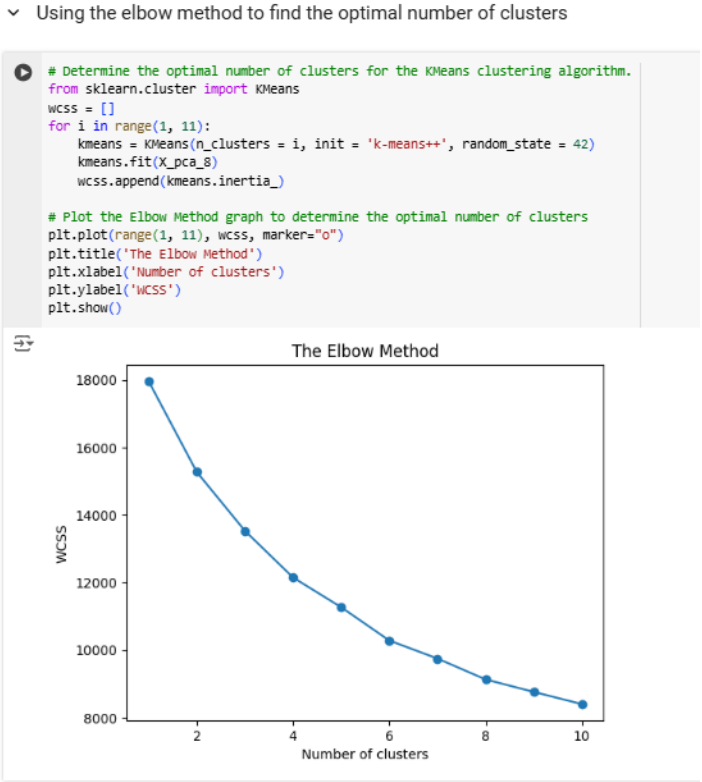


Figure 71: Elbow Method

The elbow method is used to find the optimal number of clusters in data set. It helps to balance complexity and model accuracy. Finding the right number of clusters is important for meaning for result. Elbow method help to get the right number of clusters without overfitting the model. The elbow method is easy to use and visualize. The optimal number for cluster is typically where the Sum of Squared Errors starts to flatten out significantly after this point.

“Choosing the optimal number of clusters is a crucial step in any unsupervised learning algorithm. Since we don’t have predefined cluster counts in unsupervised learning, we need a systematic approach to determine the best k value. The Elbow Method is a popular technique used for this purpose in K-Means clustering.” (GeeksforGeeks, n.d.).

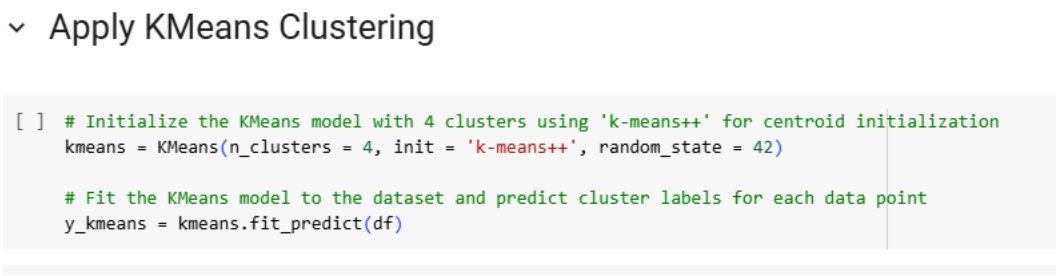


Figure 72: K-means++

“The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the *inertia* or within-cluster sum-of-squares” (Scikit-learn, n.d.).

K-Means is a popular and straightforward method for clustering data. It effectively groups data points into distinct clusters based on their closeness, aiming to make items within the same cluster as similar as possible. The algorithm works by reducing the variation within each cluster, measured as the sum of squared distances from the data points to their cluster centres. We have assigned number of clusters based on the elbow method. The number of clusters is set to 4, meaning the dataset is assumed to have four distinct groups or patterns to identify. The k-means++ method is used to start the clustering process, helping K-Means work faster and more accurately. It picks the initial cluster centres far apart from each other, which reduces the chances of the algorithm making poor groupings.

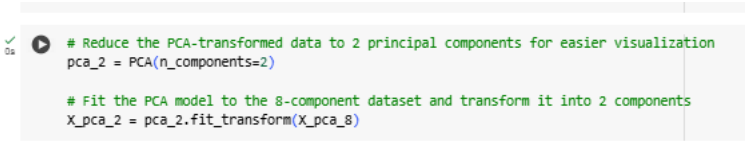


Figure 73: PCA Transformation of 8 Components to 2 Components for Visualization

We used PCA to reduce the data from 8 components to 2 for several practical reasons. First, it makes the data easier to visualize, allowing us to plot and interpret

it in a 2D space. By focusing on the most important components, PCA helps us capture the key patterns while filtering out noise and less relevant features. This also simplifies the data, making it more manageable and computationally efficient. Additionally, reducing the number of dimensions can enhance the performance of machine learning models by minimizing the risk of overfitting. Ultimately, PCA helps streamline the analysis, making the data both more accessible and insightful.

Visualizing k-means++ clusters in a 2D plot

```
[30] # Define colors for each cluster
     colours = ['red', 'blue', 'green',"black"]

     # Plot the 4 clusters in 2D using PCA components for visualization
     plt.figure(figsize=(10,8))
     for i in range(4):
       plt.scatter(X_pca_2[y_kmeans == i, 0], X_pca_2[y_kmeans == i, 1],
                   s = 100, c = colours[i] , label = 'Cluster '+str(i+1))
     plt.title('K-means++ clustering of Individuals')
     plt.xlabel('Dimension 1')
     plt.ylabel('Dimension 2')
     plt.legend()
     plt.show()
```

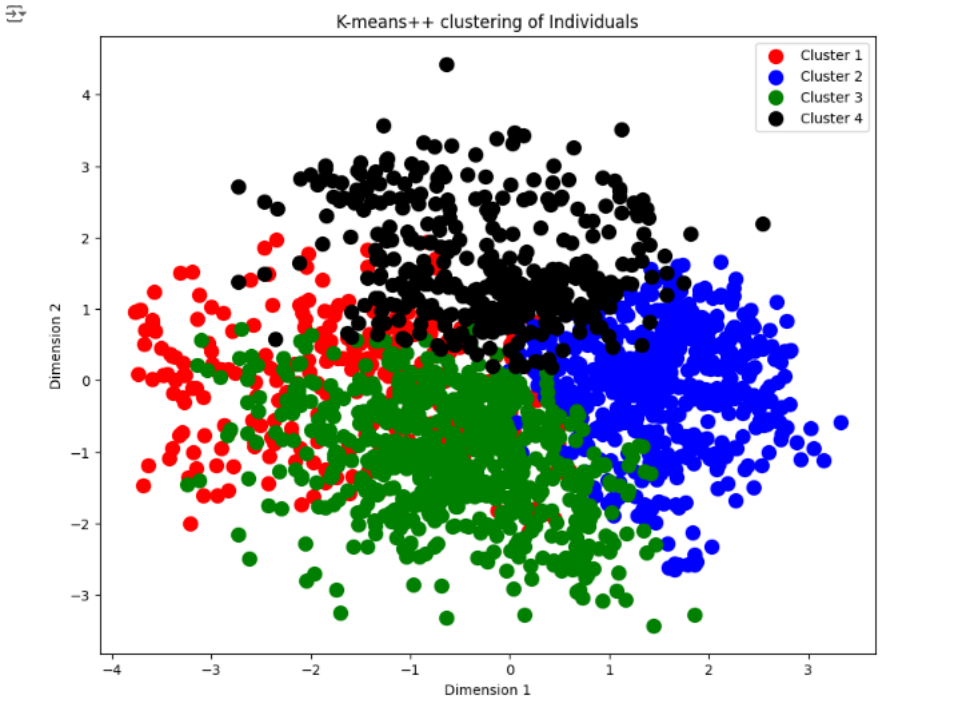


Figure 74: Visualizing K-means++ clusters

Plot the scatter plot to visualize the 4 different clusters takes place after application of kmeans++. The black cluster is at the top centre, red and blue on left and right side respectively and green cluster at the centre bottom.

2.5.2 Agglomerative Clustering:

```
[31] import scipy.cluster.hierarchy as sch

     plt.figure(figsize=(15,6))
     dendrogram = sch.dendrogram(sch.linkage(X_pca_8, method = 'ward'))
     plt.title('Dendrogram')
     plt.xlabel('Individuals')
     plt.ylabel('Euclidean distances')
     plt.show()
```

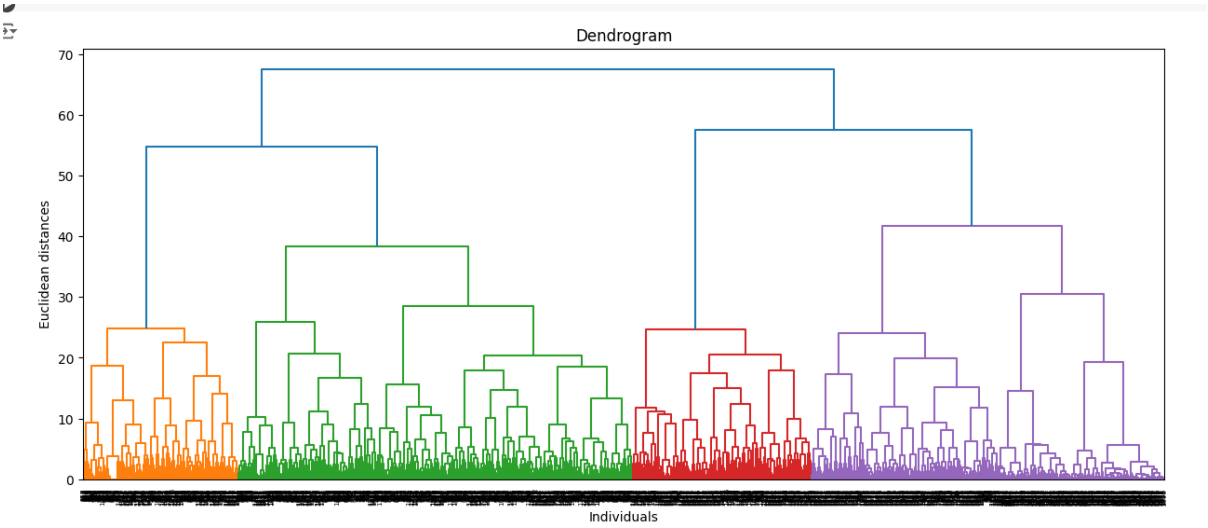


Figure 75: Visualizing Hierarchical Relationships

“The sole concept of hierarchical clustering lies in just the construction and analysis of a dendrogram. A dendrogram is a tree-like structure that explains the relationship between all the data points in the system.” (**Towards Data Science, 2021).**

With a dendrogram, we can clearly and visually observe how data points gather based on similarities or differences, which helps us determine the ideal number of clusters for hierarchical clustering. When clusters merge, a dendrogram's vertical line height reveals how distinct they are from one another.

To determine the optimal number of clusters, we search for the largest vertical shift between mergers. This jump signifies a substantial change in similarity, indicating that the clusters being merged at that point are very different from one another. We may create clusters that naturally group similar data points together by clipping the dendrogram at this height, avoiding over-clustering (forming too many little, unneeded groups) or under-clustering (combining distinct groups into one).

In other words, the dendrogram assists us in determining the boundaries of relevant clusters that can be formed from the data.

Apply Agglomerative Clustering

```
# Import AgglomerativeClustering from scikit-learn
from sklearn.cluster import AgglomerativeClustering

# Initialize the Agglomerative Clustering model with 4 clusters, using Euclidean distance and Ward's linkage
agglo = AgglomerativeClustering(n_clusters = 4, metric = 'euclidean', linkage = 'ward' )

# Fit the model to the PCA-transformed dataset and predict cluster labels for each data point
y_agglo = agglo.fit_predict(X_pca_8)
```

Figure 76: Agglomerative Clustering

“Agglomerative Clustering: It uses a bottom-up approach. It starts with each object forming its own cluster and then iteratively merges the clusters according to their similarity to form large clusters. It terminates either when certain clustering condition imposed by user is achieved or All clusters merge into a single cluster.” (GeeksforGeeks, 2023, March 27th).

Agglomerative clustering is a hierarchical clustering technique that groups data points according to how similar they are. For datasets like the obesity dataset, it is especially helpful since it reveals organic clusters, such groups of people with similar dietary or lifestyle choices. This technique is adaptable, enabling the use of various distance metrics (such as Manhattan or Euclidean) to fit the features of the dataset.

We got the number of clusters based on the dendrogram we draw. Until all of the data points are grouped into a single cluster, or the required number of clusters is reached, the method iteratively merges the closest clusters based on a linkage criterion, such as average or complete linkage. The procedure begins with each data point as its own cluster. Because it can identify significant patterns and subgroups, such "high-risk" and "low risk" obesity profiles, it is especially well-suited for the obesity dataset.

```
# Define colors for each cluster

colours = ["orange", "black", "magenta","green"]

# Plot the 4 clusters in 2D using PCA components for visualization
plt.figure(figsize=(8,8))
for i in range(4):
  plt.scatter(X_pca_2[y_agglo == i, 0], X_pca_2[y_agglo == i, 1],
              s = 100, c = colours[i] , label = 'Cluster '+str(i+1))
plt.title('Agglomerative clustering of Individuals')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend()
plt.show()
```

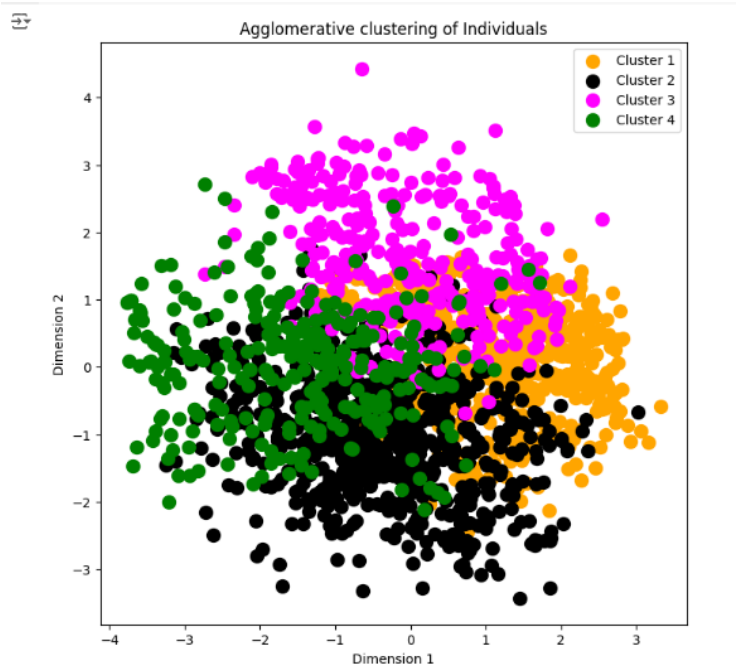


Figure 77: Visualizing Agglomerative clustering

Plot the scatter plot to visualize the 4 different clusters takes place after application of Agglomerative clustering. The magenta cluster is at the top centre, green and orange on left and right side respectively and black cluster at the centre bottom.

2.6 Comparison of K-means++ and Agglomerative Clustering:

Compare the performance of the two K-means++ and Agglomerative Clustering

```
[ ] # Fit the KMeans model to the data and predict cluster labels for each point in X
    # The labels indicate which cluster each data point belongs to
    kmeans_labels = kmeans.fit_predict(X_pca_8)

    # Fit the AgglomerativeClustering model to the data and predict cluster labels for each point in X
    # The labels indicate which cluster each data point belongs to
    agglo_labels = agglo.fit_predict(X_pca_8)
```

```
[ ] # Calculate the silhouette score for KMeans++ clustering
    kmeans_silhouette = silhouette_score(X_pca_8, kmeans_labels)

    #Calculate the silhouette score for the AgglomerativeClustering model
    agglo_silhouette = silhouette_score(X_pca_8, agglo_labels)
```

```
[ ] #Calculate the Davies-Bouldin Index for the KMeans model
    kmeans_db = davies_bouldin_score(X_pca_8, kmeans_labels)

    #Calculate the Davies-Bouldin Index for the AgglomerativeClustering model
    agglo_db = davies_bouldin_score(X_pca_8, agglo_labels)
```

```
[37] # Print the evaluation metrics for both clustering models:

     print("KMeans++ Silhouette Score:", kmeans_silhouette)
     print("Agglomerative Clustering Silhouette Score:", agglo_silhouette)
     print("\nKMeans++ Davies-Bouldin Index:", kmeans_db)
     print("Agglomerative Clustering Davies-Bouldin Index:", agglo_db)
```

```
KMeans++ Silhouette Score: 0.1640968124998371
Agglomerative Clustering Silhouette Score: 0.1587761322265986

KMeans++ Davies-Bouldin Index: 1.886078704569595
Agglomerative Clustering Davies-Bouldin Index: 1.9176582248373704
```

Figure 78: Compare the performance of both algorithms

Compares the clustering performance of K-means++ and Agglomerative Clustering models using two evaluation metrics Silhouette Score and Davies-Bouldin Index. The Silhouette Score for K-Means was 0.16, indicating low positvive cluster separation, whereas Agglomerative Clustering had an score of 0.15.

```
#Create a side-by-side comparison of the clustering results

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.scatter(X_pca_2[:, 0], X_pca_2[:, 1], c=kmeans_labels, cmap='viridis', s=30)
ax1.set_title("KMeans++ Clustering")

ax2.scatter(X_pca_2[:, 0], X_pca_2[:, 1], c=agglo_labels, cmap='viridis', s=30)
ax2.set_title("Agglomerative Clustering")

plt.show()
```

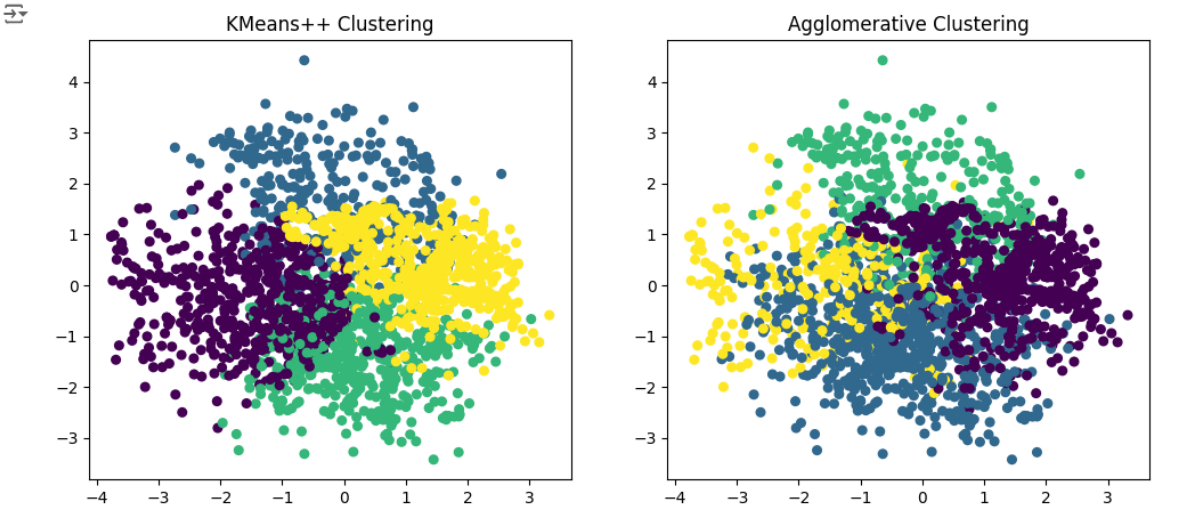


Figure 79: Visulizing both algorithms

K-Means formed four well-defined clusters. On the other hand, identified a few distinct clusters and several noise points, representing individuals who did not fit neatly into any group. K-Means performed better in terms of clear and interpretable clusters, while Agglomerative Clustering was useful for identifying outliers, which might represent individuals with unique lifestyle patterns

2.7 Business Impact:

Policymakers and healthcare professionals can benefit greatly from the clustering results. Targeted treatments can be created to meet the unique needs of each cluster by defining discrete lifestyle groups. For instance, specialised nutritional counselling and physical activity programs could be beneficial for people in a cluster that consumes a lot of calories and is not physically active.

2.8 Real-World Applications:

Using clusters to design tailored healthcare interventions for individuals based on their lifestyle, eating habits, and physical activity levels. Identify at-risk individuals who may become obese in the future, allowing for proactive measures to prevent obesity.

2.9 Conclusions:

Using K-Means and Agglomerative Clustering, the research was able to successfully identify groups of people based on lifestyle behaviors. Agglomerative Clustering identified distinct outliers that could require further attention, but K-Means was more successful in distinguishing different groups. The findings may aid in adjusting healthcare programs to the unique requirements of various populations, which could lead to more successful methods of managing obesity. To improve the clustering accuracy, future research might use further characteristics, such genetic predisposition, or employ more sophisticated clustering approaches.

2.10 References:

UCI Machine Learning Repository. (n.d.). *Estimation of obesity levels based on eating habits and physical condition dataset*. Retrieved November 30, 2024, from https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition

Scikit-learn. (n.d.). *Principal component analysis (PCA)*. Retrieved November 30, 2024, from https://scikit-learn.org/stable/modules/decomposition.html#pca

GeeksforGeeks. (n.d.). *Elbow method for optimal value of K in KMeans*. Retrieved November 30, 2024, from https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

Scikit-learn. (n.d.). *K-means clustering*. Retrieved November 30, 2024, from https://scikit-learn.org/1.5/modules/clustering.html#k-means

Towards Data Science. (2021, May 7). *Hierarchical clustering explained*. Towards Data Science. https://towardsdatascience.com/hierarchical-clustering-explained-e59b13846da8

GeeksforGeeks. (2023, March 27). *Agglomerative methods in machine learning*. GeeksforGeeks. https://www.geeksforgeeks.org/agglomerative-methods-in-machine-learning/

Task 3: Sentiment Analysis

3.0 Title:

Transforming US Airline Tweets Feedback for Enhanced Customer Feedback Management

3.1 Introduction:

Social media platforms like Twitter have become vital tools for customer interaction, especially in the airline industry. This study focuses on understanding customer sentiments by analyzing tweets about airlines. Using sentiment analysis, it categorizes the tweets as positive or negative, shedding light on common concerns and the reasons behind feedback.

By diving into customer sentiments and pinpointing recurring issues, this project aims to help airlines improve their services, streamline their operations, and build better relationships with their passengers.

3.2 Dataset Description:

The dataset used in this study is the Airline Tweets dataset, which contains 14,640 tweets directed at six major airlines. Each tweet is labelled with its sentiment (positive, neutral, or negative) and includes additional features to provide context, such as reasons for negative feedback and confidence scores.

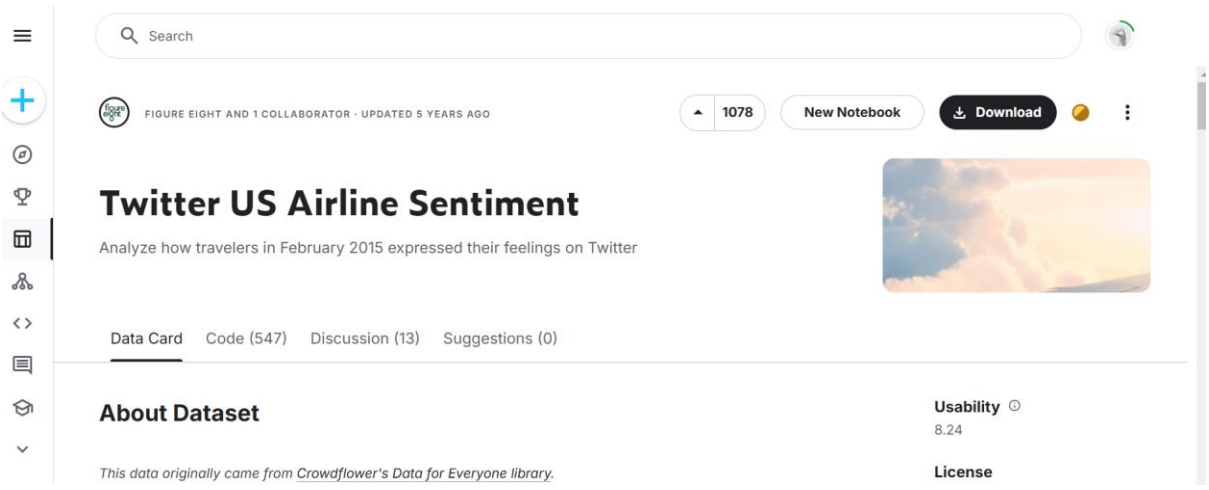


Figure 80: Airline Tweets Data

The data set consists of features like, tweet_id, airline_sentiment, airline_sentiment_confidence, negativereason, negativereason_confidence, airline, airline_sentiment_gold, name, negativereason_gold, retweet_count, text, tweet_coord, tweet_created, tweet_location, user_timezone.

The dataset was obtained from Kaggle, where it is publicly available for academic and research purposes (CrowdFlower, 2015).

3.3 Ethical, Social, and legal Issues:

The Twitter Airline Sentiment dataset is used in this project only for academic study. The dataset complies with Twitter's terms of service and is available to the general audience on Kaggle.

Since the purpose of this study is to demonstrate sentiment analysis tools, no further ethical approval is needed. Potential biases in sentiment interpretation were noted, and the data was anonymised. The results are only meant to be instructive and illustrative.

3.4 Exploratory Data Analysis (EDA) and Data Preprocessing:

To understand the dataset better in detail we have conducted exploratory data analysis using summary statistics and visualizations.

Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from wordcloud import WordCloud
import nltk
nltk.download(["stopwords","punkt","wordnet","omw-1.4","vader_lexicon"])
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.probability import FreqDist
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.naive_bayes import MultinomialNB
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

Figure 81: Importing Libraries

These libraries enable efficient data processing, visualization, and machine learning model implementation.

```
#Importing and reading the CSV file 'Tweets.csv' into a DataFrame named 'df'
df = pd.read_csv("Tweets.csv")

# Create independent copies
df_classification = df.copy()  # For classification
df_sentiment = df.copy()  # For sentiment analysis

# Displaying the first five rows of the DataFrame
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	te
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	NaN	0	@VirginAmeri Wh @dhepbu sa
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmeri plus you' add commercials
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	NaN	0	@VirginAmeri I didn't today Must mean I r
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	NaN	0	@VirginAmeri it's rea aggressive blas
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmeri and it's a rea big bad thing

✓ 0s completed at 1:37 PM

Figure 82: Loading and Viewing the Dataset

The dataset was loaded to ensure its structure and integrity before analysis. Independent copies are created for classification and sentiment analysis tasks. Separating the datasets ensures that task-specific transformations do not interfere with each other.

```
# Display basic information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   tweet_id                      14640 non-null  int64
 1   airline_sentiment             14640 non-null  object
 2   airline_sentiment_confidence  14640 non-null  float64
 3   negativereason                9178 non-null   object
 4   negativereason_confidence     10522 non-null  float64
 5   airline                       14640 non-null  object
 6   airline_sentiment_gold        40 non-null     object
 7   name                          14640 non-null  object
 8   negativereason_gold           32 non-null     object
 9   retweet_count                 14640 non-null  int64
 10  text                          14640 non-null  object
 11  tweet_coord                   1019 non-null   object
 12  tweet_created                 14640 non-null  object
 13  tweet_location                9907 non-null   object
 14  user_timezone                 9820 non-null   object
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
```

Figure 83: Dataset Overview and Structure

The df.info() method provides an overview of the dataset such as dataset contains 14,640 entries across 15 columns, Non-Null Values, Data Types, etc.

```
# Display basic statistics of numeric cloumns of the dataset.
df.describe(include="all")
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	twe
count	1.464000e+04	14640	14640.000000	9178	10522.000000	14640	40	14640	32	14640.000000	14640	
unique	NaN	3	NaN	10	NaN	6	3	7701	13	NaN	14427	
top	NaN	negative	NaN	Customer Service Issue	NaN	United	negative	JetBlueNews	Customer Service Issue	NaN	@united thanks	
freq	NaN	9178	NaN	2910	NaN	3822	32	63	12	NaN	6	
mean	5.692184e+17	NaN	0.900169	NaN	0.638298	NaN	NaN	NaN	NaN	0.082650	NaN	
std	7.791112e+14	NaN	0.162830	NaN	0.330440	NaN	NaN	NaN	NaN	0.745778	NaN	
min	5.675883e+17	NaN	0.335000	NaN	0.000000	NaN	NaN	NaN	NaN	0.000000	NaN	
25%	5.685592e+17	NaN	0.692300	NaN	0.360600	NaN	NaN	NaN	NaN	0.000000	NaN	
50%	5.694779e+17	NaN	1.000000	NaN	0.670600	NaN	NaN	NaN	NaN	0.000000	NaN	
75%	5.698905e+17	NaN	1.000000	NaN	1.000000	NaN	NaN	NaN	NaN	0.000000	NaN	
max	5.703106e+17	NaN	1.000000	NaN	1.000000	NaN	NaN	NaN	NaN	44.000000	NaN	

Figure 84: Descriptive Statistics of the Dataset

The dataset's summary reveals that airline sentiment has three categories, with "negative" being the most frequent. These insights guide preprocessing and feature selection.

```
# Checking the number of missing values in 'airline_sentiment', 'text' columns.
df[['airline_sentiment', 'text']].isnull().sum()
```

	0
airline_sentiment	0
text	0

dtype: int64

```
# Counting the occurrences of each unique value in the 'airline_sentiment' column.
df["airline_sentiment"].value_counts()
```

airline_sentiment	count
negative	9178
neutral	3099
positive	2363

dtype: int64

Figure 85: Checking Missing Values and Sentiment Distribution

The analysis confirms that the airline sentiment and text columns contain no missing values, ensuring complete data for sentiment analysis. The distribution of sentiments reveals an imbalance.

```
# Displaying the first five rows of the 'text' column to inspect the content of the tweets
df["text"].head()
```

	text
0	@VirginAmerica What @dhepburn said.
1	@VirginAmerica plus you've added commercials t...
2	@VirginAmerica I didn't today... Must mean I n...
3	@VirginAmerica it's really aggressive to blast...
4	@VirginAmerica and it's a really big bad thing...

dtype: object

```
# Filtering the DataFrame to exclude rows where the 'airline_sentiment' is 'neutral'
df_classification = df_classification[df_classification['airline_sentiment'] != 'neutral']
df_classification["airline_sentiment"].value_counts()
```

airline_sentiment	count
negative	9178
positive	2363

dtype: int64

Figure 86: Inspecting Text Data and Filtering Neutral Sentiments

To gain an understanding of the structure and substance of the tweets, the first five rows of the text column is shown. These samples demonstrate the type of feedback that is frequently aimed at airlines.

Rows containing neutral sentiment were removed from the dataset in order to focus the classification effort on unambiguous polarities.

```
# Creating a count plot to visualize the distribution of 'airline_sentiment' after filtering
sns.countplot(x="airline_sentiment",data=df_classification)
```

```
<Axes: xlabel='airline_sentiment', ylabel='count'>
```

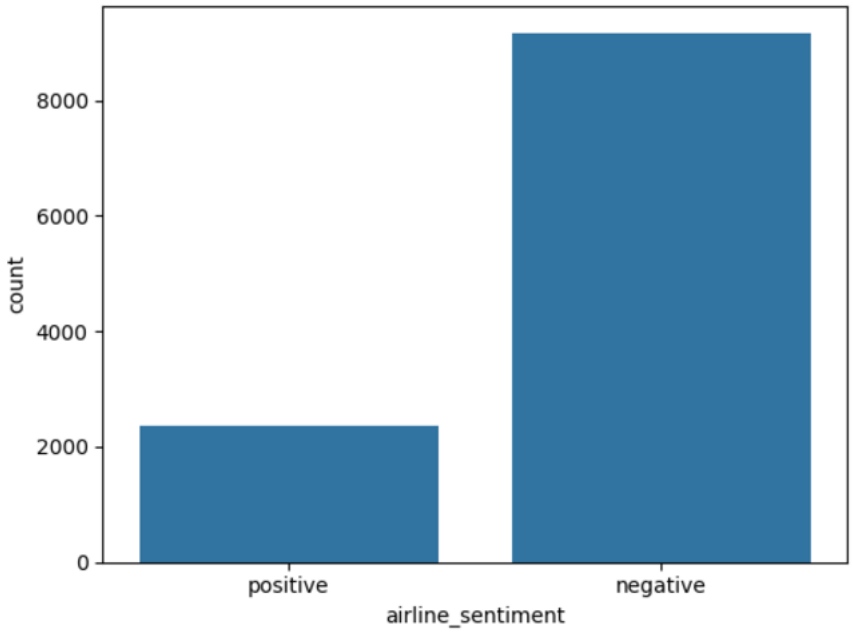


Figure 87: Visualization of Sentiment Distribution

The visualization provides a clear understanding of the dataset's sentiment composition, crucial for designing effective preprocessing and classification strategies.

Data Cleaning and Preprocessing

```
# Importing the list of English stopwords from the NLTK library
stop_words = nltk.corpus.stopwords.words("english")

additional_stopwords = ["americanair", "flight", "get", "call", "aa", "hour", "service", "custom","2", "cancel", "would", "http", "time"]
stop_words.extend(additional_stopwords)


print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her',
```

```
# Defining a function to preprocess text by:
# - Tokenizing the text using a regular expression to keep only alphanumeric words and apostrophes
# - Converting tokens to lowercase and removing common stopwords (e.g., 'the', 'and')
# - Applying stemming to reduce words to their root form (e.g., "running" -> "run")
def preprocess_text(text):
  tokenizer = nltk.tokenize.RegexpTokenizer("[a-zA-Z0-9\']+").tokenize(text)
  tokenizer = [word for word in tokenizer if len(word) >= 3]
  cleaned_tokens = [word.lower() for word in tokenizer if word.lower() not in stop_words]
  stemmed_text = [nltk.stem.PorterStemmer().stem(word) for word in cleaned_tokens]
  return stemmed_text
```

Figure 88: Data Cleaning and Preprocessing

A custom list of stop words is added to the standard English stopwords. These words are irrelevant to sentiment analysis and are removed to focus on meaningful content. Splitting tweets into individual words also excluding irrelevant or commonly used words that do not contribute to sentiment. Moreover, it Is reducing words to their root form to standardize variations This preprocessing pipeline ensures that the text data

is clean, uniform, and ready for feature extraction and modeling, ultimately improving the performance of the sentiment analysis model.

As per **Kavita Ganesan's** explanation, by eliminating stop words we enhanced the quality of the text data for modeling, allowing the analysis to focus on more meaningful and sentiment-driven words, ultimately improving the accuracy and efficiency of the sentiment analysis task.

Tokenization was a crucial preprocessing step in this analysis. It involves breaking down text into smaller units, typically words or phrases, to make the data manageable and structured for processing. As highlighted in the **DataCamp** article "What Is Tokenization?", this technique is foundational in natural language processing (NLP).

Using stemming can significantly enhance text preprocessing by reducing words to their root forms, thereby standardizing textual data (IBM, n.d.). IBM (n.d.) explains that stemming helps simplify text analysis by consolidating word variants into a common base, making it a crucial step in natural language processing tasks like sentiment analysis.

```
# Updating the 'text' Column in the DataFrame This involves tokenization, stopword removal, and stemming of each tweet
df_classification["text"] = df_classification["text"].apply(preprocess_text)
df_classification.head()
```

fidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone
0.3486	NaN	0.0000	Virgin America	NaN	jnardino	NaN	0	[virginamerica, plu, ad, commerci, experi, tacki]	NaN	2015-02-24 11:15:59 -0800	NaN	Pacific Time (US & Canada)
1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	NaN	0	[virginamerica, realli, aggress, blast, obnoxi...	NaN	2015-02-24 11:15:36 -0800	NaN	Pacific Time (US & Canada)
1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	NaN	0	[virginamerica, realli, big, bad, thing]	NaN	2015-02-24 11:14:45 -0800	NaN	Pacific Time (US & Canada)
1.0000	Can't Tell	0.6842	Virgin America	NaN	jnardino	NaN	0	[virginamerica, serious, pay, seat, play, real...	NaN	2015-02-24 11:14:33 -0800	NaN	Pacific Time (US & Canada)
0.6745	NaN	0.0000	Virgin America	NaN	cjmcginnis	NaN	0	[virginamerica, ye, nearli, everi, fli, ear, w...	NaN	2015-02-24 11:13:57 -0800	San Francisco CA	Pacific Time (US & Canada)

Figure 89: Applying Preprocessing to the Text Column

The text column of the dataset was pre-processed using the defined text preprocessing function. This step ensures that the text data is clean, consistent, and optimized for modeling.

```
# Term Frequency Matrix
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df_classification["text"].map(''.join))
X = pd.DataFrame(X.toarray())
X.head()
```

	0	1	2	3	4	5	6	7	8	9	...	13192	13193	13194	13195	13196	13197	13198	13199	13200	13201
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 13202 columns

Figure 90: Creating a Term Frequency Matrix

According to IBM (n.d.), CountVectorizer is a useful tool for getting text data ready for modeling tasks like sentiment analysis since it makes text data easier to analyze by extracting characteristics from the text.

Creates a sparse matrix from the cleaned text, with each row denoting a tweet and each column representing a distinct word. 13,202 columns (unique words) and as many rows as tweets in the sample make up the final matrix. Each cell shows how frequently a word appears in a specific tweet. This transformation is necessary to translate unstructured textual input into a numerical representation so that machine learning models may efficiently handle and analyze the data.

Splitting Data into Training and Testing Sets

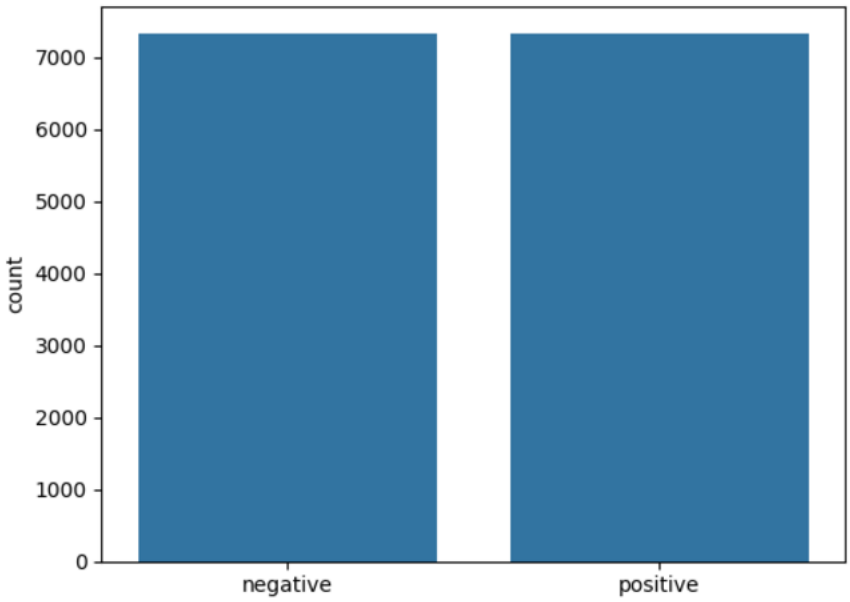
```
[ ] Generated code may be subject to a license |
#Splitting Data into Training and Testing Sets for Model Evaluation
from sklearn.model_selection import train_test_split

y = df_classification["airline_sentiment"]

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42,shuffle=True,stratify=y)
```

Figure 91: Splitting Data into Training and Testing Sets

The dataset was divided into training and testing sets using the train_test_split function from scikit-learn. This step is essential for evaluating the model's performance on unseen data, ensuring generalizability and robustness.

Handling Imbalanced

```
[ ]  from imblearn.over_sampling import SMOTE
     resampler= SMOTE(random_state=0)
     X_train_oversampled,y_train_oversampled=resampler.fit_resample(X_train,y_train)
     sns.countplot(x=y_train_oversampled)
     plt.show()
```



0s

Figure 92: Handling Class Imbalance

The imbalance highlights the need for techniques like oversampling SMOTE during model training to prevent bias toward the majority class. By interpolating current data points, SMOTE creates synthetic samples for the minority class (positive). In the training dataset, this evens out the distribution of classes. Balancing the classes ensures that the machine learning model does not become biased toward the majority class, leading to more accurate and fair predictions.

3.5 Classification Model Implementation:

Training the Model

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train,y_train)
```

```
MultinomialNB
MultinomialNB()
```

Figure 93: Training the Model

To train the sentiment classification model, the Multinomial Naive Bayes (MultinomialNB) algorithm was employed. This technique works especially well with text data because it is made for classification problems that use discrete features, like word frequencies produced by the CountVectorizer.

Model Evaluation

```
y_pred = model.predict(X_test)

from sklearn import metrics
acc = metrics.accuracy_score(y_test, y_pred)
print("K-NN Accuracy:",acc)
print("-----------------------------------------------------")
# Compute and print the confusion matrix to evaluate the performance of the classification model
cm = metrics.confusion_matrix(y_test, y_pred)
print(f"K-NN Confusion Matrix:\n{cm}")
print("-----------------------------------------------------")
# Print the classification report to evaluate model performance.
report = metrics.classification_report(y_test, y_pred)
print(f"K-NN Classification Report:\n {report}")
```

```
K-NN Accuracy: 0.8059766132524903
-----------------------------------------------------
K-NN Confusion Matrix:
[[1833    3]
 [ 445   28]]
-----------------------------------------------------
K-NN Classification Report:
              precision    recall  f1-score   support

    negative       0.80      1.00      0.89      1836
    positive       0.90      0.06      0.11       473

    accuracy                           0.81      2309
   macro avg       0.85      0.53      0.50      2309
weighted avg       0.82      0.81      0.73      2309
```

Figure 94: Model Evaluation

The trained Multinomial Naive Bayes model was evaluated using the testing dataset. The model achieved an accuracy of 80.6%, indicating that it correctly classified sentiments for approximately 81% of the test samples. However, the low recall and F1-score for positive sentiments indicate room for improvement in distinguishing the minority class. These metrics highlight the model's strength in identifying negative sentiments but suggest the need for further tuning or advanced techniques to improve performance on positive sentiments.

3.6 Sentiment Analysis:

Sentiment Analysis

Initializing the Sentiment Analyzer and Calculating Sentiment Scores

```
[ ] # Initializing the SentimentIntensityAnalyzer
    sentimentor = SentimentIntensityAnalyzer()

    # Creating new columns in the DataFrame to store sentiment scores for each tweet.
    df_sentiment["compound"] = [sentimentor.polarity_scores(tweet)["compound"] for tweet in df_sentiment["text"]]
    df_sentiment["neg"] = [sentimentor.polarity_scores(tweet)["neg"] for tweet in df_sentiment["text"]]
    df_sentiment["neu"] = [sentimentor.polarity_scores(tweet)["neu"] for tweet in df_sentiment["text"]]
    df_sentiment["pos"] = [sentimentor.polarity_scores(tweet)["pos"] for tweet in df_sentiment["text"]]
```

Figure 95: Sentiment Analysis

Sentiment analysis is a natural language processing (NLP) technique used to determine the emotional tone behind a body of text, often used to understand customer opinions and feedback (IBM, n.d.)

Calculating sentiment scores is essential for a number of reasons, especially comprehending consumer feedback. Insightful knowledge helps companies better understand their clients, proactively handle problems, and enhance overall service quality requires the computation of sentiment scores.

Exploratory Analysis of Sentiment Scores

```
# Displaying the first five rows of the updated DataFrame to inspect the newly added sentiment score columns
df_sentiment.head()
```

ivereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone	compound	neg	neu	pos
NaN	Virgin America	NaN	cairdin	NaN	0	@VirginAmerica What @dhepburn said.	NaN	2015-02-24 11:35:52 -0800	NaN	Eastern Time (US & Canada)	0.0000	0.000	1.000	0.0
0.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica plus you've added commercials t...	NaN	2015-02-24 11:15:59 -0800	NaN	Pacific Time (US & Canada)	0.0000	0.000	1.000	0.0
NaN	Virgin America	NaN	yvonnalynn	NaN	0	@VirginAmerica I didn't today... Must mean I n...	NaN	2015-02-24 11:15:48 -0800	Lets Play	Central Time (US & Canada)	0.0000	0.000	1.000	0.0
0.7033	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica it's really aggressive to blast...	NaN	2015-02-24 11:15:36 -0800	NaN	Pacific Time (US & Canada)	-0.5984	0.246	0.754	0.0
1.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica and it's a really big bad thing...	NaN	2015-02-24 11:14:45 -0800	NaN	Pacific Time (US & Canada)	-0.5829	0.321	0.679	0.0

Figure 96: Exploratory Analysis of Sentiment Scores

The updated dataset includes new columns for sentiment scores (compound, neg, neu, pos) generated by the Sentiment Intensity Analyzer.

```
# Generating summary statistics to understand the distribution of sentiment scores in the dataset
df_sentiment[["compound","neg","neu","pos"]].describe()
```

	compound	neg	neu	pos
count	14640.000000	14640.000000	14640.000000	14640.000000
mean	0.052463	0.083950	0.794055	0.121991
std	0.456935	0.111314	0.166833	0.160114
min	-0.966800	0.000000	0.102000	0.000000
25%	-0.296000	0.000000	0.693000	0.000000
50%	0.000000	0.000000	0.808000	0.069000
75%	0.438225	0.147000	0.917000	0.198000
max	0.976000	0.857000	1.000000	0.898000

Figure 97: Summary Statistics of Sentiment Scores

The summary statistics for the sentiment scores (compound, neg, neu, pos) provide insights into the distribution of sentiments in the dataset. These statistics reveal that the dataset is predominantly neutral, with a skew toward negative sentiments, consistent with earlier findings. This analysis provides a deeper understanding of the dataset's sentiment distribution.

```
# Creating a histogram to visualize the distribution of the 'compound' sentiment scores
sns.histplot(df_sentiment["compound"])
```

```
<Axes: xlabel='compound', ylabel='Count'>
```

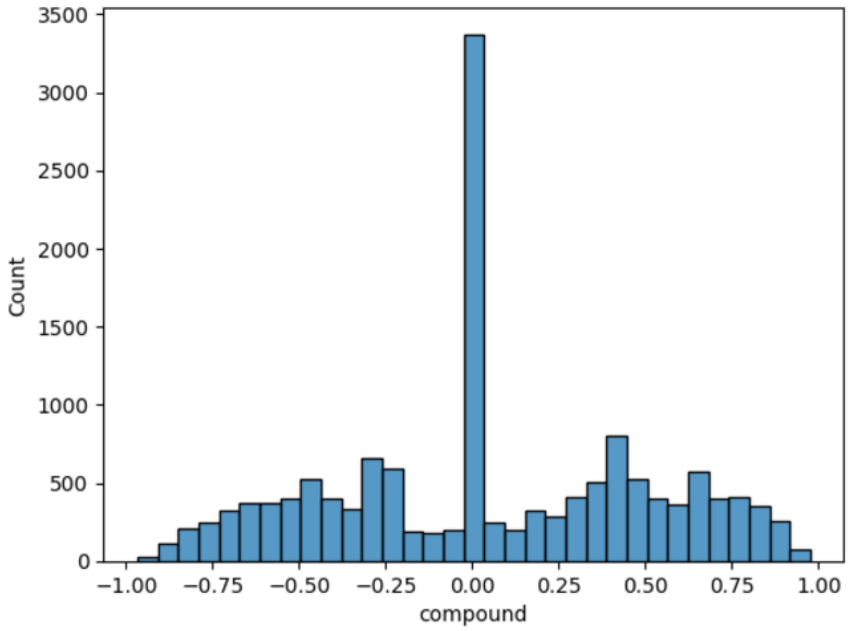


Figure 98: Histogram of Compound Sentiment Scores

A significant peak around 0 indicates a predominance of tweets with neutral sentiment. The dataset shows a tendency toward neutrality, with relatively balanced distributions of positive and negative sentiments.

```
# Creating a histogram to visualize the distribution of the 'positive' sentiment scores
sns.histplot(df_sentiment["pos"])
```

```
<Axes: xlabel='pos', ylabel='Count'>
```

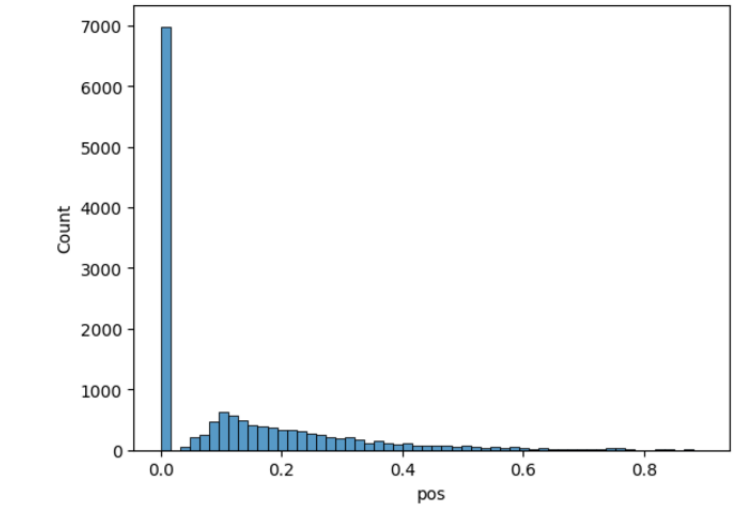


Figure 99: Histogram of Positive Sentiment Scores

A significant majority of tweets have a positive sentiment score close to 0, indicating a low prevalence of positive content in the dataset.

```
[ ] # Creating a histogram to visualize the distribution of the 'negative' sentiment scores
    sns.histplot(df_sentiment["neg"])
```

```
<Axes: xlabel='neg', ylabel='Count'>
```

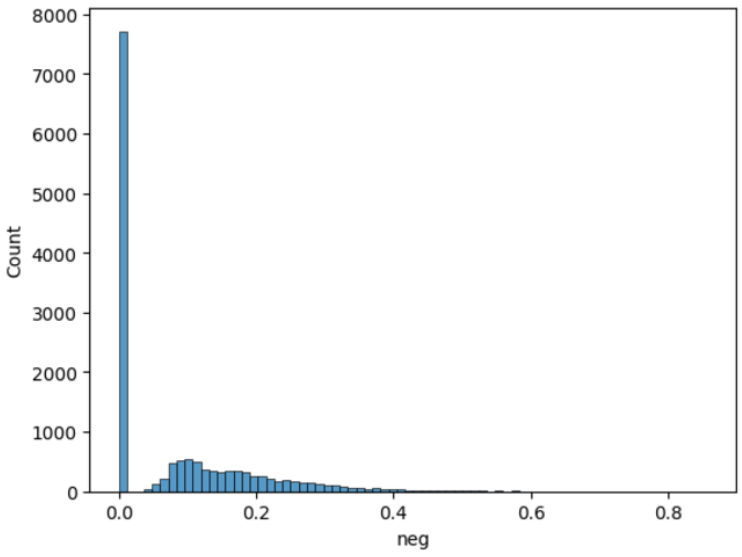


Figure 100: Histogram of Negative Sentiment Scores

Most tweets have a negative sentiment score close to 0, indicating a smaller proportion of strong negativity in the dataset.

```
#Counting Non-Positive Sentiment Tweets Grouped by Airline
(df_sentiment["compound"]<=0).groupby(df_sentiment["airline"]).sum()
```

airline	compound
American	1638
Delta	1222
Southwest	1230
US Airways	1828
United	2227
Virgin America	276

dtype: int64

Figure 101: Non-Positive Sentiment Distribution Across Airlines

The distribution highlights the varying levels of customer sentiment toward different airlines, with some airlines experiencing more negative perceptions than others.

```
# Calculating the percentage of negative tweets for each airline.
percent_negative = pd.DataFrame((df_sentiment["compound"]<=0).groupby(df_sentiment["airline"]).sum()
                                /df_sentiment["airline"].groupby(df_sentiment["airline"]).count()*100,
                                columns=["% negative tweets"]).sort_values(by= "% negative tweets")

percent_negative["% negative tweets"] = percent_negative["% negative tweets"].round(2)

percent_negative
```

airline	% negative tweets
Southwest	50.83
Virgin America	54.76
Delta	55.00
United	58.27
American	59.37
US Airways	62.75

Figure 102: Percentage of Negative Tweets by Airline

This analysis highlights variations in customer sentiment across airlines, offering valuable insights for prioritizing service improvements.

```
# Visualizing the Percentage of Negative Tweets by Airline
sns.barplot(data = percent_negative, x ="% negative tweets", y= percent_negative.index , color = "c")
plt.xlabel("% Negative Tweets")
plt.ylabel("Airline")
plt.title("Percentage of Negative Tweets by Airline")
```

```
Text(0.5, 1.0, 'Percentage of Negative Tweets by Airline')
```

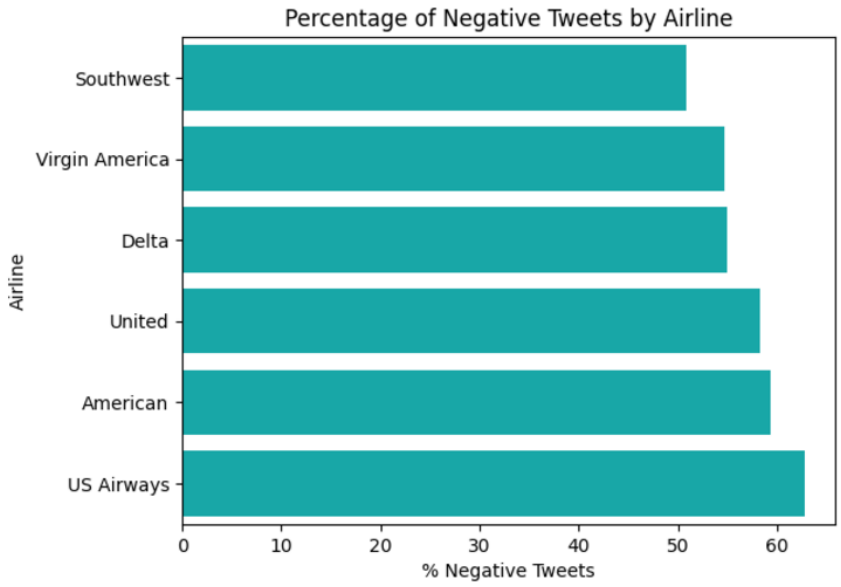


Figure 103: Visualization of Negative Tweet Percentages by Airline

This visualization highlights disparities in customer sentiment across airlines, providing actionable insights for targeting improvements in customer service and experience.

```
# Preprocessing Tweets and Filtering Positive/Negative Subsets for American Airlines
df_sentiment['processed_tweet'] = df_sentiment['text'].apply(preprocess_text)
tweets_positive_subset = df_sentiment.loc[(df_sentiment['airline'] == 'American') & (df_sentiment['compound'] > 0),:]
tweets_negative_subset = df_sentiment.loc[(df_sentiment['airline'] == 'American') & (df_sentiment['compound'] <= 0),:]
tweets_positive_subset.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	twee
11881	570307949614256128	negative	0.6316	Bad Flight	0.3164	American	NaN	ELLLORRAC	NaN	0	@AmericanAir thanks for getting back to me. Bu...	NaN	11:
11882	570307948171423745	negative	0.6846	Flight Booking Problems	0.6846	American	NaN	SweeLoTmac	NaN	0	@AmericanAir why would I pay $200 to reactivat...	NaN	11:
11883	570307434113310720	negative	0.6547	Late Flight	0.3331	American	NaN	LauraMolito	NaN	0	@AmericanAir stranded for 24 hours in MIA, Pat...	NaN	11:
11890	570306529947193344	negative	0.6449	longlines	0.3340	American	NaN	TheTPVshow	NaN	0	@AmericanAir I slept in the miami airport due ...	NaN	11:
11891	570306423818723328	neutral	0.6767	NaN	0.0000	American	NaN	sammy575	NaN	0	@AmericanAir is the new 9:45 time confirmed or...	NaN	11:

Figure 104: Filtering Positive and Negative Tweets for American Airlines

This breakdown allows focused analysis of customer feedback for American Airlines, enabling targeted sentiment analysis and identification of common themes for positive and negative tweets.

Visualizing Word Clouds for Sentiment Subsets

```
# Displaying a Word Cloud for Negative Tweets

neg_tokens = [word for tweet in tweets_negative_subset['processed_tweet'] for word in tweet]

wordcloud = WordCloud(background_color='white').generate_from_text(" ".join(neg_tokens))

# Display the generated image:
plt.figure(figsize=(12,12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

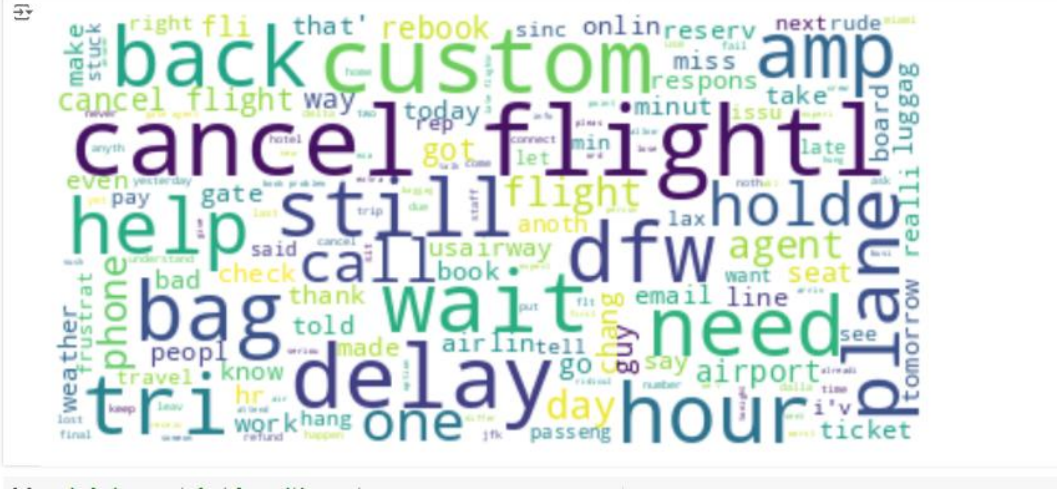


Figure 105: Word Cloud for Negative Tweets

The visualization highlights common themes and areas of dissatisfaction among customers, providing actionable insights for American Airlines to focus on improving specific service areas.

```
[ ] # Displaying a Word Cloud for Positive Tweets

pos_tokens = [word for tweet in tweets_positive_subset['processed_tweet'] for word in tweet]

wordcloud = WordCloud(background_color='white').generate_from_text(" ".join(pos_tokens))

# Display the generated image:
plt.figure(figsize=(12,12))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

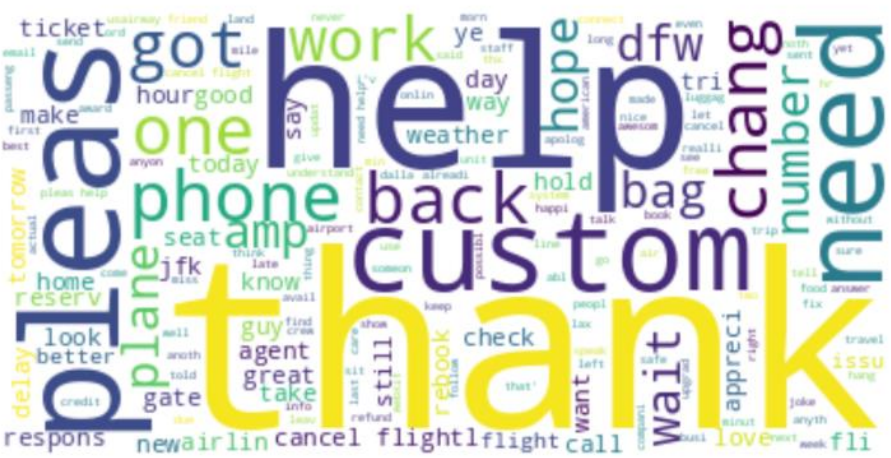


Figure 106: Word Cloud for Positive Tweets

Positive feedback focuses on helpful customer service, quick responses, and resolving issues effectively. These insights highlight areas where the airline is meeting or exceeding customer expectations.

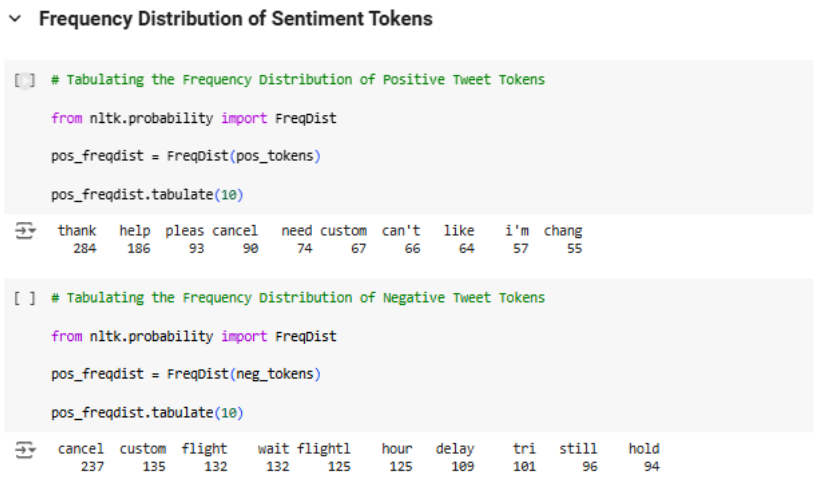


Figure 107: Frequency Distribution of Sentiment Tokens

This analysis tabulates the most frequent words (tokens) in positive and negative tweets for American Airlines. The frequency distribution emphasizes areas of satisfaction and dissatisfaction. This granular token analysis aids in prioritizing service improvements and recognizing strengths.

3.7 Business Impact:

Regular grievances regarding baggage problems, delays, and cancellations draw attention to areas where operational effectiveness might be raised. By addressing these issues, unfavourable publicity and disappointed clients can be decreased. Proactive measures to address negative feedback can help mitigate damage to the airline's brand image. By addressing concerns and enhancing strengths highlighted in positive feedback, airlines can differentiate itself in a competitive market and potentially attract new customers.

3.8 Real-World Applications of Sentiment Analysis:

The insights and techniques derived from this analysis can be applied in various real-world scenarios across industries to enhance business outcomes and customer experiences.

- Airlines and other service-oriented industries can identify and address recurring customer pain points, such as delays or cancellations, improving satisfaction.
- Companies can monitor online sentiments to gauge brand perception and address negative publicity promptly.
- Comparing sentiment analysis data across competitors to identify areas of competitive advantage or disadvantage.
- Companies can use sentiment data to refine existing products or develop new offerings based on customer needs and feedback.
- Real-time analysis of social media to detect and mitigate crises before they escalate.

3.9 Conclusions:

This sentiment analysis study demonstrates how data-driven insights can be used to better understand and enhance consumer experiences. The analysis, which used tweets about American Airlines, identified important areas of customer happiness and discontent and offered practical suggestions for improving operations and service quality.

According to the data, the majority of tweets were neutral or negative, and frequent grievances included baggage problems, delays, and cancelled flights.
Helpful customer service and problem solving were the main factors linked to positive attitudes.

3.10 References:

CrowdFlower. (2015). *Twitter Airline Sentiment Dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment?resource=download

Ganesan, K. (n.d.). *What Are Stop Words?*. Retrieved from https://kavita-ganesan.com/what-are-stop-words/

DataCamp. (n.d.). *What Is Tokenization?*. Retrieved from https://www.datacamp.com/blog/what-is-tokenization

IBM. (n.d.). *What is stemming?*. Retrieved from https://www.ibm.com/topics/stemming

IBM. (n.d.). *CountVectorizer in Python*. Retrieved from https://www.ibm.com/reference/python/countvectorizer

IBM. (n.d.). *What is Sentiment Analysis?*. Retrieved from https://www.ibm.com/topics/sentiment-analysis

Table of Figure:
Classification:

Clustering:

Sentiment Analysis: