



DATA MINING

TEAM 20

RANA ASHRAF: 2205019

YEHIA TAREK:2205062

SARA AHMED: 2205094

MOHAMED ESSAM:2205049

Overview

Passenger satisfaction is measured by more than 120,000 airline passengers, including additional information about each passenger, their flight and type of travel, as well as ratings on various factors such as cleanliness, comfort, service and overall experience.

Goals

1. Data Cleaning: Handling missing or duplicate values.
2. Data Analysis: Explore and Understand the data
3. Applying Mining Techniques:

DATA EXPLANATION

Columns:

ID: Passenger ID

Gender: Passenger gender

Age: Passenger age

Customer Type: The customer type

Type of Travel: Purpose of the flight of the passengers

Class: Travel class in the plane of the passengers

Flight Distance: Flight distance

Departure Delay: Minutes delayed when departure

Arrival Delay: Minutes delayed when Arrival

Departure & Arrival Time Convenience: Convenience of departure and arrival times for passengers

Ease of Online Booking: Easy to book

Check-in Service: Ease of registration

Online Boarding: Convenience of online registration

Gate Location: Gate Location Estimation

On-board Service: Service on board

Seat Comfort: Comfortable seating

Leg Room Service: Leg room service level

Cleanliness: Cleanliness level

Food and Drink: Quality of food and drinks

In-flight Service: Level of service on board

In-flight Wifi Service: Wifi quality level on board

In-flight Entertainment: Rating of in-flight entertainment

Baggage Handling: Opinion on baggage handling

Satisfaction: Airline satisfaction level

Data ACCESS

LINK: <https://www.kaggle.com/datasets/mikhail1681/airline-quality-ratings/data>

Read data from csv file

```
In [2]: data=pd.read_csv('rap.csv')
```

Data Preprocessing

I. Data Analysis

1. Shape: The dataset contains 129,880 rows and 24 columns.

```
In [5]: data.shape
```

```
Out[5]: (129880, 24)
```

2. Data Types: Shows the data types of each column.

```
In [5]: data.dtypes
```

```
Out[5]: ID                int64
Gender                object
Age                  int64
Customer Type        object
Type of Travel        object
Class                object
Flight Distance       int64
Departure Delay       int64
Arrival Delay         float64
Departure and Arrival Time Convenience  int64
```

3. Head and Tail: Display the first and last 10 rows of the dataset

```
In [6]: data.head(10)
```

```
Out[6]:
```

	ID	Gender	Age	Customer Type	Type of Travel	Class	Flight Distance	Departure Delay	Arrival Delay	Departure and Arrival Time Convenience
0	1	Male	48	First-time	Business	Business	821	2	5.0	3
1	2	Female	35	Returning	Business	Business	821	26	39.0	2
2	3	Male	41	Returning	Business	Business	853	0	0.0	4
3	4	Male	50	Returning	Business	Business	1905	0	0.0	2
4	5	Female	49	Returning	Business	Business	3470	0	1.0	3
5	6	Male	43	Returning	Business	Business	3788	0	0.0	4
6	7	Male	43	Returning	Business	Business	1963	0	0.0	3
7	8	Female	60	Returning	Business	Business	853	0	3.0	3
8	9	Male	50	Returning	Business	Business	2607	0	0.0	1
9	10	Female	38	Returning	Business	Business	2822	13	0.0	2

4. Info: show information about columns, data types and memory usage

#	Column	Non-Null	Count	Dtype
0	track_id	14397	non-null	object
1	artist_name	14397	non-null	object
2	Song_Success	14397	non-null	object
3	album_release_date	14397	non-null	object
4	hit_n	14397	non-null	int64
5	Drums_Energy	14397	non-null	float64
6	Drums_Complexity	14397	non-null	float64
7	Variety_of_musical_instruments	14397	non-null	float64
8	Mixing_Quality	14397	non-null	float64
9	Harmonic_Richness	14397	non-null	float64
10	Mixing_Character	14397	non-null	float64
11	Emotional_Intensity	14397	non-null	float64
12	is_feat	14397	non-null	bool
13	n_feat	14397	non-null	int64
14	status_guest	5741	non-null	object
15	higher_guest	14397	non-null	int64
16	album_name	14397	non-null	object
17	album_type	14397	non-null	object
18	track_number	14397	non-null	int64
19	explicit	14397	non-null	bool
20	key_name	14389	non-null	object
21	mode_name	14389	non-null	object
22	key_mode	14389	non-null	object
23	artists_all	14397	non-null	object
24	remake	14397	non-null	object
25	artist_id	14397	non-null	object
26	album_id	14397	non-null	object
27	download_link	14397	non-null	object

dtypes: bool(2), float64(7), int64(4), object(15)
memory usage: 2.9+ MB

In [5]: data.info()

5. Summary Statistics: Presents descriptive statistics for numerical columns.

```
In [8]: data.describe()
```

```
Out[8]:
```

	ID	Age	Flight Distance	Departure Delay	Arrival Delay	Departure Arrival Convenience
count	129880.000000	129880.000000	129880.000000	129880.000000	129487.000000	129880.00
mean	64940.500000	39.427957	1190.316392	14.713713	15.091129	3.05
std	37493.270818	15.119360	997.452477	38.071126	38.465650	1.52
min	1.000000	7.000000	31.000000	0.000000	0.000000	0.00
25%	32470.750000	27.000000	414.000000	0.000000	0.000000	2.00
50%	64940.500000	40.000000	844.000000	0.000000	0.000000	3.00
75%	97410.250000	51.000000	1744.000000	12.000000	13.000000	4.00
max	129880.000000	85.000000	4983.000000	1592.000000	1584.000000	5.00

6. Missing Values: Identifies the number of missing values in each column.

```
In [9]: print(data.isnull().sum())
```

```
ID                                0
Gender                            0
Age                                0
Customer Type                     0
Type of Travel                    0
Class                             0
Flight Distance                   0
Departure Delay                   0
Arrival Delay                     393
Departure and Arrival Time Convenience  0
Ease of Online Booking            0
Check-in Service                  0
```

7. Duplicated Rows: Calculates the number of duplicated rows in the dataset.

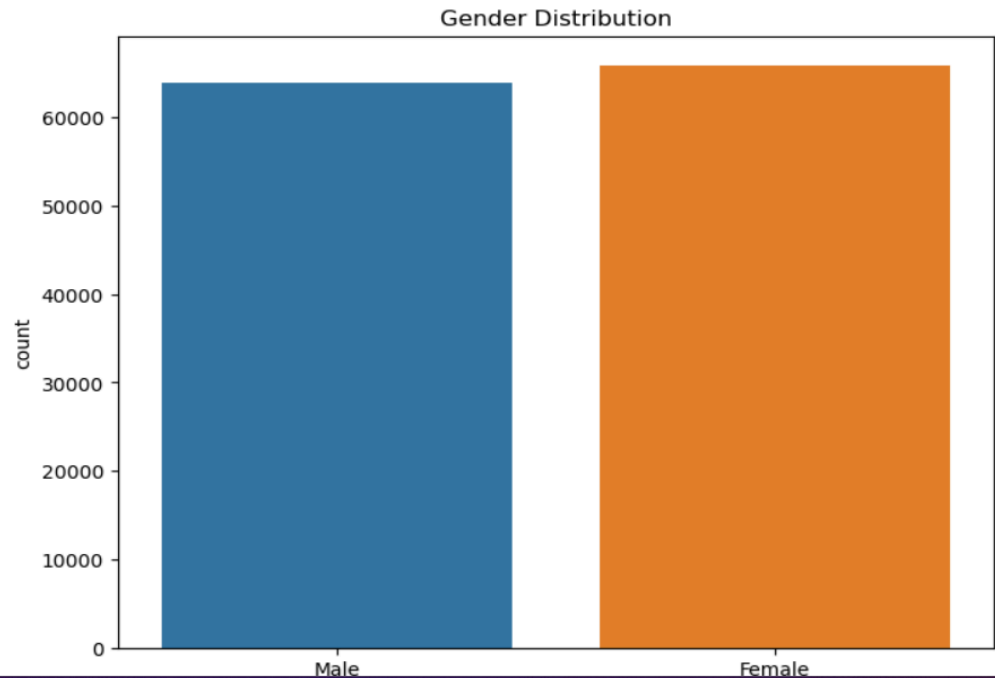
```
In [10]: print("Number of Duplicated rows:",data.duplicated().sum())
```

```
Number of Duplicated rows: 0
```

II. Data Visualization

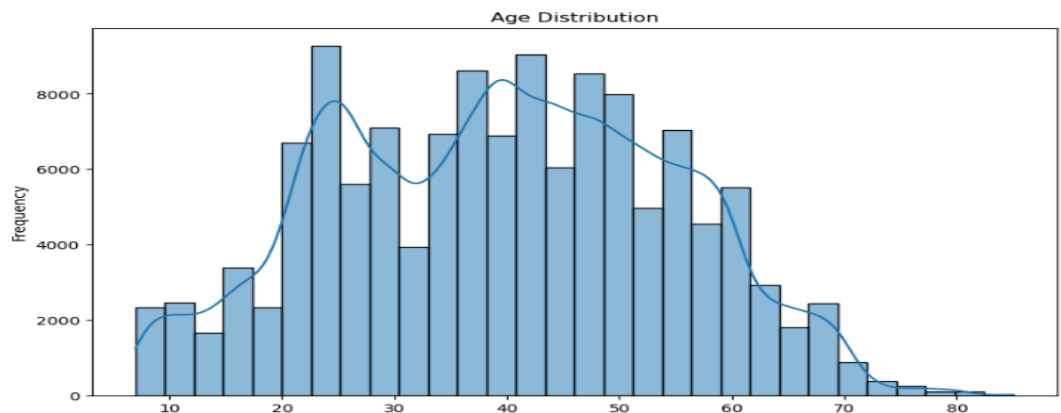
1. Gender Distribution

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', data=data)
plt.title('Gender Distribution')
plt.show()
```



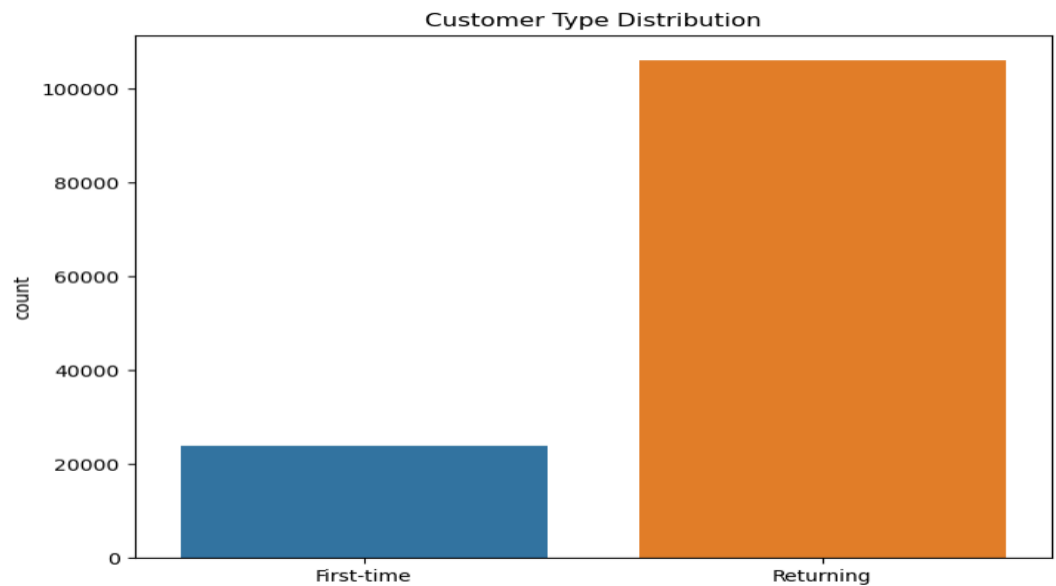
2. Age Distribution

```
In [12]: # Age distribution
plt.figure(figsize=(10, 6))
sns.histplot(data['Age'], bins=30, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



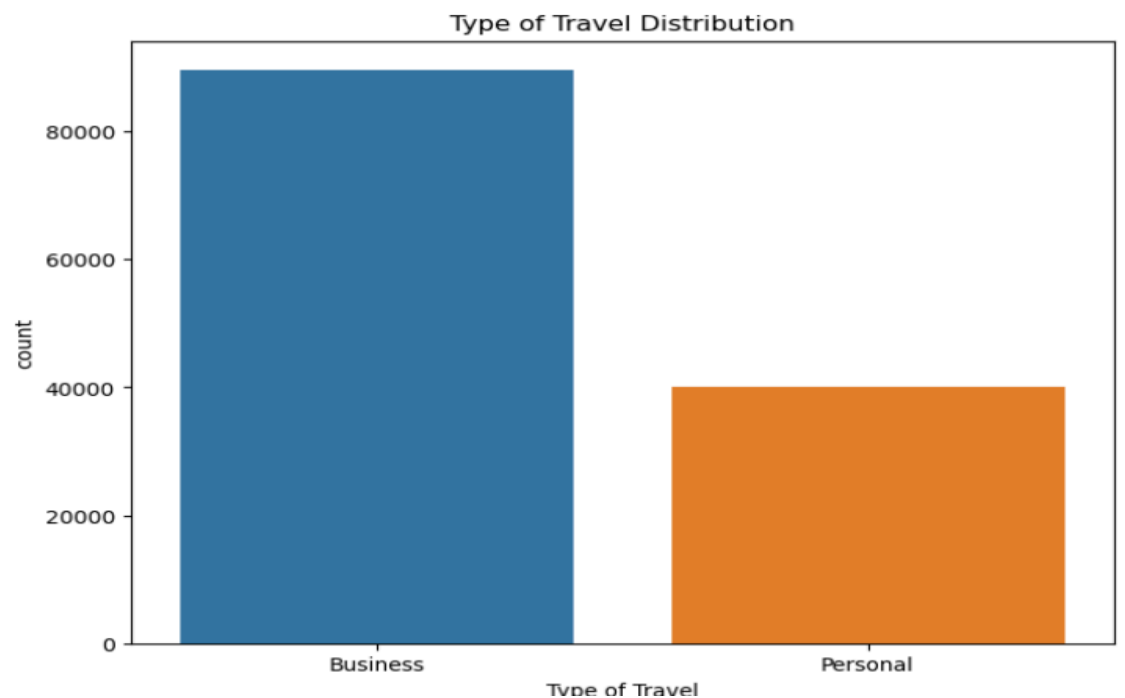
3. Customer type distribution

```
In [13]: # Customer type distribution
plt.figure(figsize=(8, 6))
sns.countplot(x='Customer Type', data=data)
plt.title('Customer Type Distribution')
plt.show()
```



4. Type of travel distribution

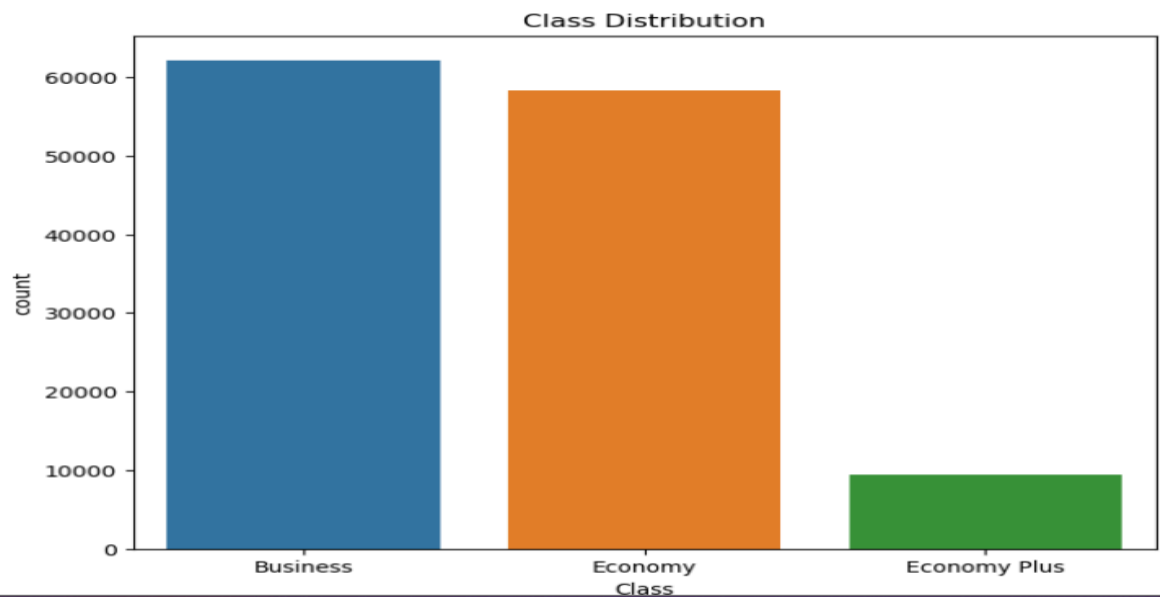
```
plt.figure(figsize=(8, 6))
sns.countplot(x='Type of Travel', data=data)
plt.title('Type of Travel Distribution')
plt.show()
```





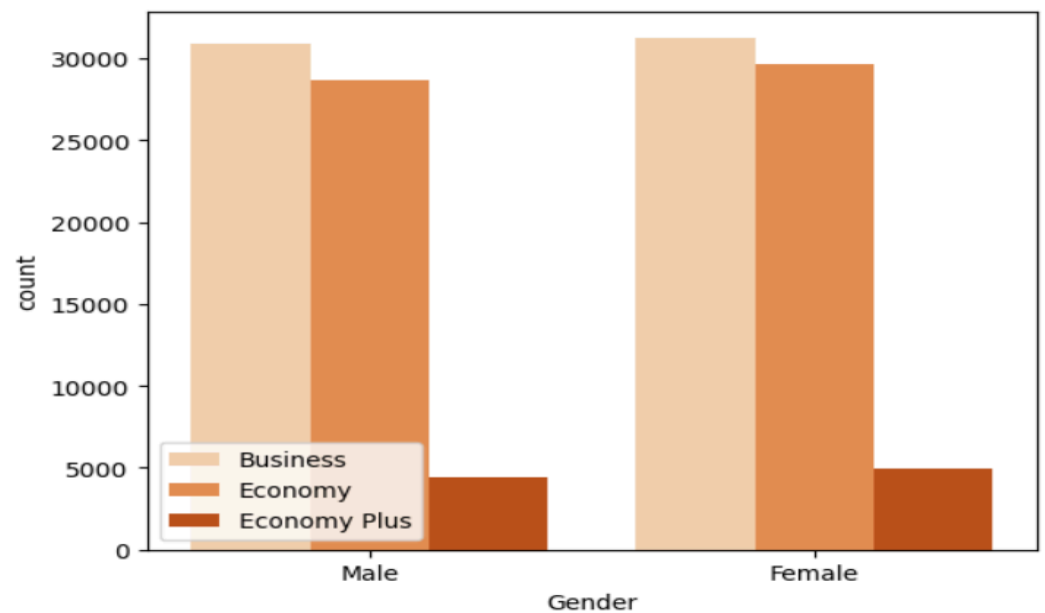
5. Class Distribution

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Class', data=data)
plt.title('Class Distribution')
plt.show()
```



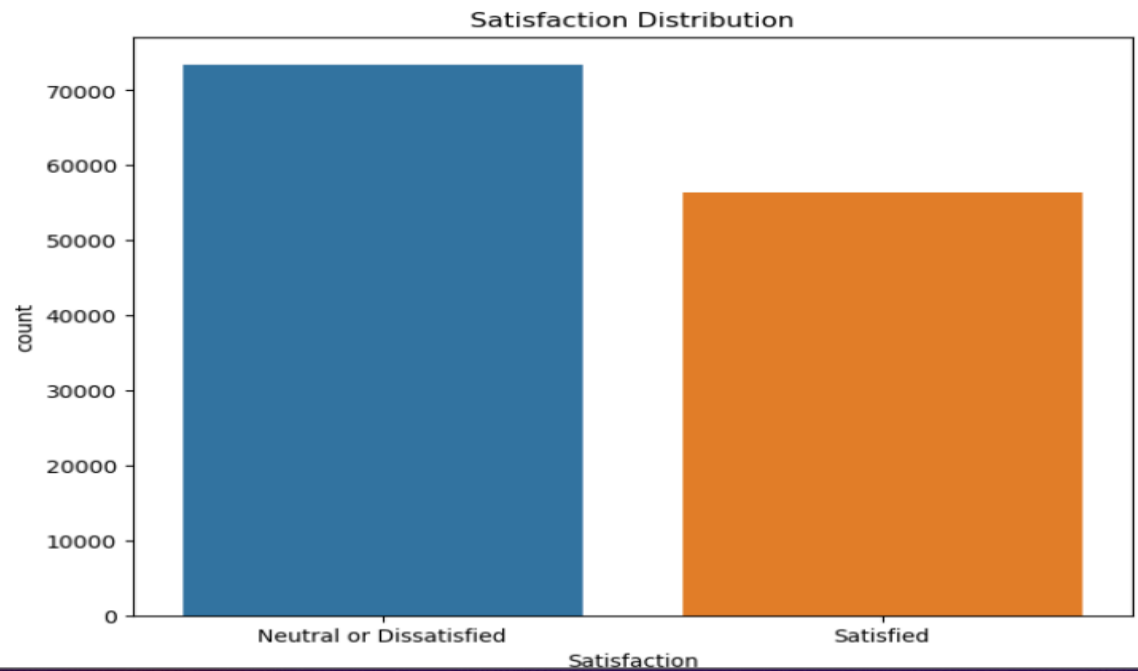
6. Class Distribution according to gender

```
In [16]: sns.countplot(x="Gender", data=data, palette="Oranges", hue="Class")
plt.legend(loc=3)
plt.show()
```



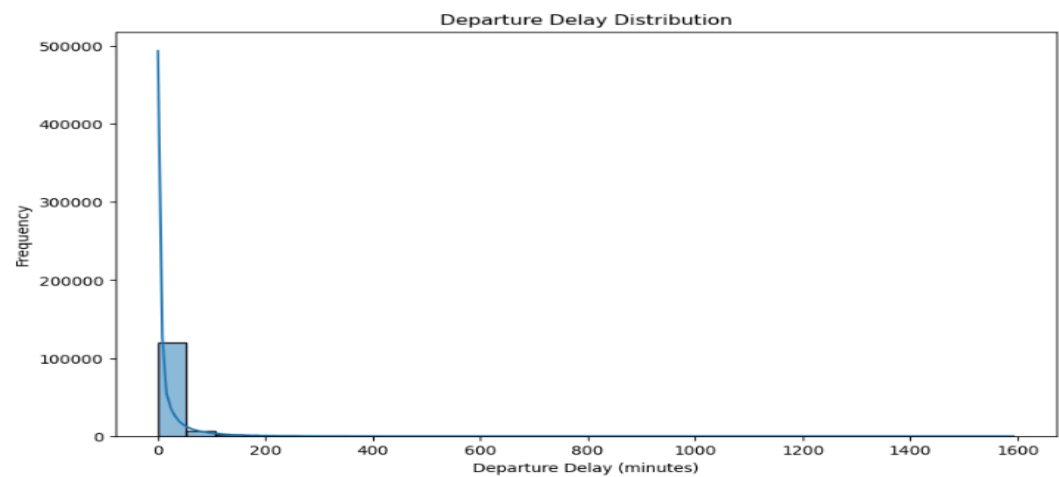
7. Satisfaction Distribution

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Satisfaction', data=data)
plt.title('Satisfaction Distribution')
plt.show()
```



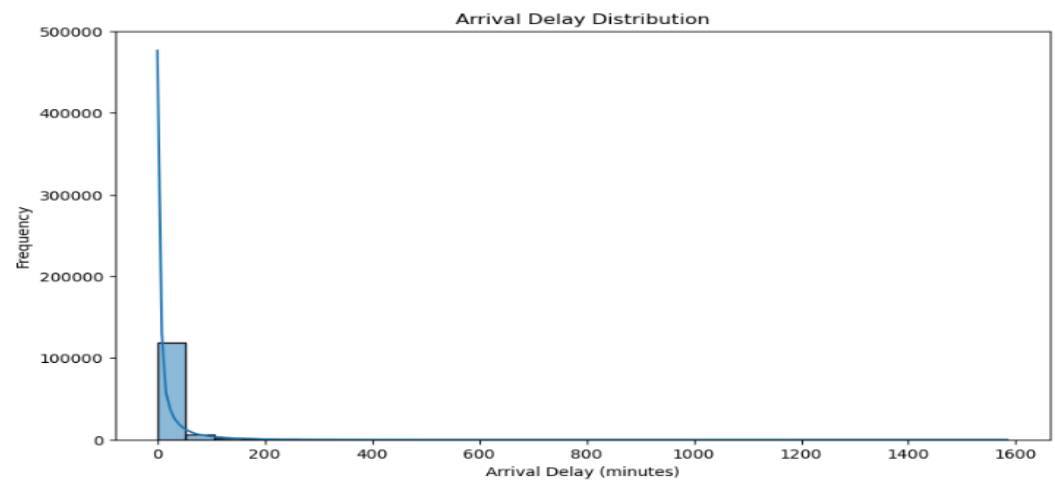
8. Departure Delay Distribution

```
In [18]: # Departure delay distribution
plt.figure(figsize=(10, 6))
sns.histplot(data['Departure Delay'], bins=30, kde=True)
plt.title('Departure Delay Distribution')
plt.xlabel('Departure Delay (minutes)')
plt.ylabel('Frequency')
plt.show()
```



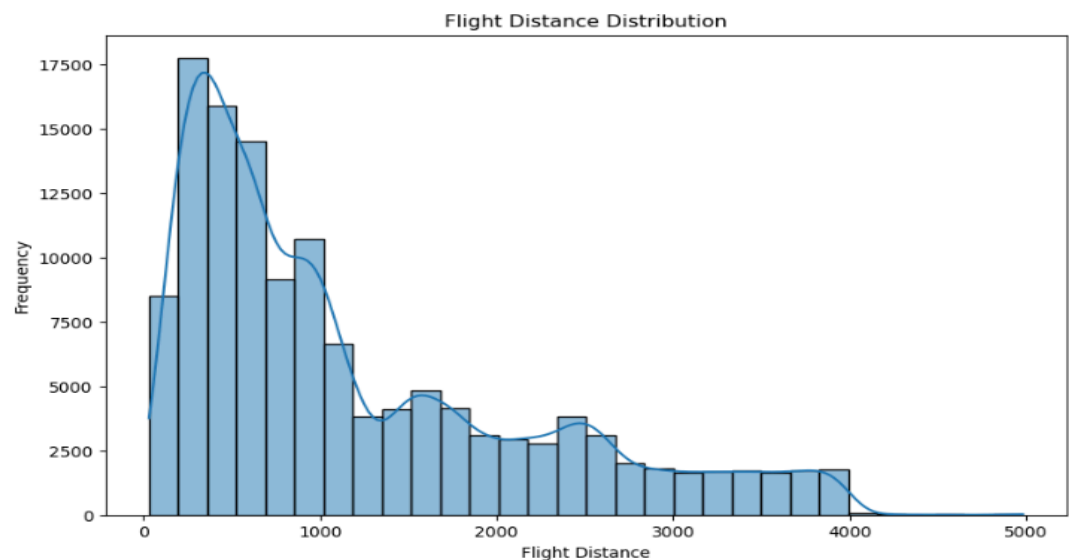
9. Arrival Delay Distribution

```
In [19]: # Arrival delay distribution
plt.figure(figsize=(10, 6))
sns.histplot(data['Arrival Delay'].dropna(), bins=30, kde=True)
plt.title('Arrival Delay Distribution')
plt.xlabel('Arrival Delay (minutes)')
plt.ylabel('Frequency')
plt.show()
```



10. Flight Distance Distributions

```
In [20]: #Flight distance distribution
plt.figure(figsize=(10, 6))
sns.histplot(data['Flight Distance'], bins=30, kde=True)
plt.title('Flight Distance Distribution')
plt.xlabel('Flight Distance')
plt.ylabel('Frequency')
plt.show()
```



III. Data Cleaning

1. Delete Rows containing missing values

```
In [65]: data = data.dropna()
print(data.shape)

(129487, 24)
```

2. Delete duplicate rows

```
In [66]: data = data.drop_duplicates()
print(data.shape)

(129487, 24)
```

IV. Extra Operations on data

1. Split data by numerical and categorical values

```
In [63]: numerical_cols = data.select_dtypes(include=['number']).columns.tolist()
print("Number of Numerical Column: ",len(numerical_cols))
categorical_cols = data.select_dtypes(include=['object', 'category']).columns
print("Number of Categorical Column: ",len(categorical_cols))

Number of Numerical Column: 19
Number of Categorical Column: 5
```

2. Standardize data

Data to have a mean of 0 and a standard deviation of 1, This ensures they contribute equally to the clustering process.

```
In [24]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
In [25]: stnd = scaler.fit_transform(data[numerical_cols])
print(stnd)

[[-1.73267445  0.56697318 -0.37011481 ...  0.20422039  1.23070131
  1.15934237]
 [-1.73264778 -0.29295518 -0.37011481 ... -0.548095    1.23070131
  1.15934237]
 [-1.7326211  0.10393483 -0.33803645 ...  0.95653579 -0.26838688
 -0.53546117]
 ...
 [ 1.73166963  0.17008317 -0.85530007 ... -0.548095    -0.26838688
  1.15934237]
 [ 1.7316963  0.69926985 -0.85530007 ...  1.70885118 -0.26838688
  1.15934237]
 [ 1.73172298 -1.28518021 -0.85530007 ...  0.20422039 -1.01793097
 -2.2302647 ]]
```

3. Encoding

convert categorical variables into binary variables
(members or nonmembers)

In [71]:

```
encoded_data = pd.get_dummies(data[categorical_cols])
print(encoded_data)
```

	Gender_Female	Gender_Male	Customer Type_First-time	\
0	False	True	True	
1	True	False	False	
2	False	True	False	
3	False	True	False	
4	True	False	False	
...	
129875	False	True	False	
129876	False	True	False	
129877	False	True	False	
129878	False	True	False	
129879	True	False	False	

	Customer Type_Returning	Type of Travel_Business	\
0	False	True	
1	True	True	
2	True	True	
3	True	True	
4	True	True	
...	
129875	True	False	
129876	True	False	
129877	True	False	
129878	True	False	
129879	True	False	

4. Random Sample of data

```
In [27]: random_sample = data.sample(n=5000) # Selecting 5k random rows
```

5. Retrieve best and worst 10 reviews based on Satisfaction

criteria_column: column where to sort data

We will sort data according to [Satisfaction] column.

```
In [28]: criteria_column = 'Satisfaction'
```

```
In [29]: sorted_df = data.sort_values(by=criteria_column, ascending=False)
```

```
In [30]: top_10_reviews = sorted_df.head(10)
         worst_10_reviews = sorted_df.tail(10)
```

Divide data based on numerical and categorical columns

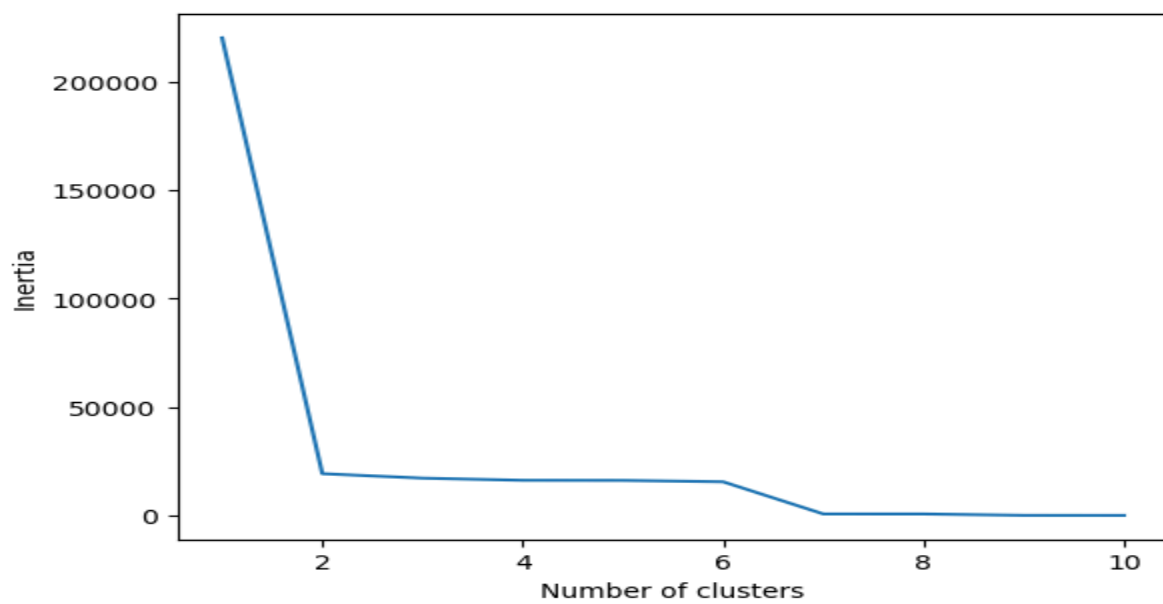
```
In [31]: best_numerical_data = top_10_reviews[numerical_cols]
         best_categorical_data = top_10_reviews[categorical_cols]
```

```
In [32]: worst_numerical_data = worst_10_reviews[numerical_cols]
         worst_categorical_data = worst_10_reviews[categorical_cols]
```

Data Mining Methods

I. Kmedoids

Plotting elbow curve to measure optimal K



```

k = 5 # Number of clusters
kmedoids = KMedoids(n_clusters=k)
kmedoids.fit(random_sample[numerical_cols])

# Get cluster labels
cluster_labels = kmedoids.labels_

# Add cluster labels to the DataFrame
random_sample['Cluster'] = cluster_labels

# Print cluster assignments
print(random_sample[['ID', 'Cluster']])

```

This code snippet performs K-Medoids clustering on a dataset (`random_sample[numerical_cols]`), assigns cluster labels to each data point, adds these labels as a new column ('Cluster') to the DataFrame, and then displays a subset of the DataFrame showing the IDs of data points and their corresponding cluster assignments.

	ID	Cluster
63886	63887	2
51599	51600	0
60946	60947	2
127	128	3
70841	70842	2
...
79108	79109	1
117488	117489	4
55448	55449	2
62714	62715	2
18035	18036	3

[5000 rows x 2 columns]

```
# Iterate over each cluster label
for cluster_label in range(k):
    # Filter the DataFrame for rows with the current cluster label
    cluster_data = random_sample[random_sample['Cluster'] == cluster_label]

    # Print the rows of the current cluster
    print(f"Cluster {cluster_label}:\n")
    print(cluster_data["ID"])
    print("\n")
```

- Iteration: Looping through each cluster label (`cluster_label`) from 0 to $k-1$.
- Filtering: Creating a new DataFrame (`cluster_data`) by filtering `random_sample` to include only rows where 'Cluster' column matches the current `cluster_label`.
- Printing: Displaying the cluster label as a header (`Cluster {cluster_label}:`), followed by the IDs (or relevant data) of the data points belonging to the current cluster (`cluster_data["ID"]`).

This loop allows you to inspect each cluster individually, showing which data points (by their IDs or other relevant attributes) belong to each cluster based on the clustering results obtained earlier. It's a useful way to analyze and understand the composition of each cluster in your data.


```

k = 5 # Number of clusters
kmedoids = KMedoids(n_clusters=k)
kmedoids.fit(best_numerical_data)

# Get cluster labels
cluster_labels = kmedoids.labels_

# Add cluster labels to the DataFrame
best_numerical_data['Cluster'] = cluster_labels

# Print cluster assignments
print(best_numerical_data['Cluster'])

```

- The code performs K-Medoids clustering (`KMedoids`) with 5 clusters ($k=5$) on `best_numerical_data`.
- Cluster labels (`cluster_labels`) are obtained from the clustering results.
- The cluster labels are added as a new column 'Cluster' to the `best_numerical_data` DataFrame.
- Finally, the cluster assignments for each data point are printed by displaying the 'Cluster' column of the DataFrame.

This process allows you to analyze and interpret how the data points are grouped into different clusters based on their numerical features, which can be useful for various tasks such as segmentation, pattern recognition, or anomaly detection in your dataset.

```

64971      3
101255      0
101253      0
101251      0
50141       1
50142       4
101246      0
50151       3
50153       2
50155       1
Name: Cluster, dtype: int64

```

```
# Get the counts of points assigned to each cluster
cluster_counts = best_numerical_data['Cluster'].value_counts()

# Print the number of points assigned to each cluster
print("Number of points assigned to each cluster:")
print(cluster_counts)
```

- Cluster Counts Calculation:
 - The `value_counts()` method applied to `best_numerical_data['Cluster']` calculates the number of occurrences (counts) of each unique cluster label in the 'Cluster' column of the DataFrame.
 - For example, if there are 100 data points in cluster 0, 80 data points in cluster 1, 150 data points in cluster 2, etc., `cluster_counts` will contain this information in a Series format.
- Printing Cluster Counts:
 - `print("Number of points assigned to each cluster:")` simply prints a descriptive header to indicate what the following output will represent.
 - `print(cluster_counts)` displays the computed cluster counts, where each line corresponds to a unique cluster label and the number of data points assigned to that cluster.

```
Number of points assigned to each cluster:
Cluster
0      4
3      2
1      2
4      1
2      1
Name: count, dtype: int64
```

```
k = 5 # Number of clusters
kmedoids = KMedoids(n_clusters=k)
kmedoids.fit(worst_numerical_data)

# Get cluster labels
cluster_labels = kmedoids.labels_

# Add cluster labels to the DataFrame
worst_numerical_data['Cluster'] = cluster_labels
cluster_centers = kmedoids.cluster_centers_

# Print cluster assignments
print(worst_numerical_data['Cluster'])
```

- The code performs K-Medoids clustering (`KMedoids`) with 5 clusters (`k=5`) on `worst_numerical_data`.
- Cluster labels (`cluster_labels`) are obtained from the clustering results and added as a new column 'Cluster' to `worst_numerical_data`.
- The cluster centers (`cluster_centers`) are obtained, representing the medoids (centroids) of each cluster in the feature space.
- Finally, the cluster assignments for each data point in `worst_numerical_data` are printed by displaying the 'Cluster' column of the DataFrame.

This process allows you to analyze and interpret how the data points are grouped into different clusters based on their numerical features, with the additional information of the cluster centers (medoids) providing insights into the representative points of each cluster in the feature space.

```

54475    1
54496    1
54494    1
54493    3
54492    0
54487    2
54485    1
54481    0
54477    4
129879   3
Name: Cluster, dtype: int64

```


```

# Get the counts of points assigned to each cluster
cluster_counts = worst_numerical_data['Cluster'].value_counts()

# Print the number of points assigned to each cluster
print("Number of points assigned to each cluster:")
print(cluster_counts)

```

- Cluster Counts Calculation:
 - The `value_counts()` method applied to `worst_numerical_data['Cluster']` calculates the number of occurrences (counts) of each unique cluster label in the 'Cluster' column of the DataFrame.
 - For example, if there are 100 data points in cluster 0, 80 data points in cluster 1, 150 data points in cluster 2, etc., `cluster_counts` will contain this information in a Series format.
- Printing Cluster Counts:
 - `print("Number of points assigned to each cluster:")` simply prints a descriptive header to indicate what the following output will represent.
 - `print(cluster_counts)` displays the computed cluster counts, where each line corresponds to a unique cluster label and the number of data points assigned to that cluster.



```
Number of points assigned to each cluster:
```

```
Cluster
```

```
1      4
```

```
3      2
```

```
0      2
```

```
2      1
```

```
4      1
```

```
Name: count, dtype: int64
```

II. HIERARCHICAL

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster import hierarchy

# Compute the hierarchical clustering
Z = hierarchy.linkage(data[numerical_cols].iloc[1:5000,:], method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dn = hierarchy.dendrogram(Z)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

This Python code uses the NumPy, Matplotlib, and SciPy libraries to perform hierarchical clustering and visualize the dendrogram.

1. Import Libraries:

- `import numpy as np`: Imports the NumPy library and allows you to use NumPy functions with the `np` alias.
- `import matplotlib.pyplot as plt`: Imports the Matplotlib library's plotting module with the alias `plt`.
- `from scipy.cluster import hierarchy`: Imports the hierarchical clustering module from the SciPy library.

2. Compute Hierarchical Clustering:

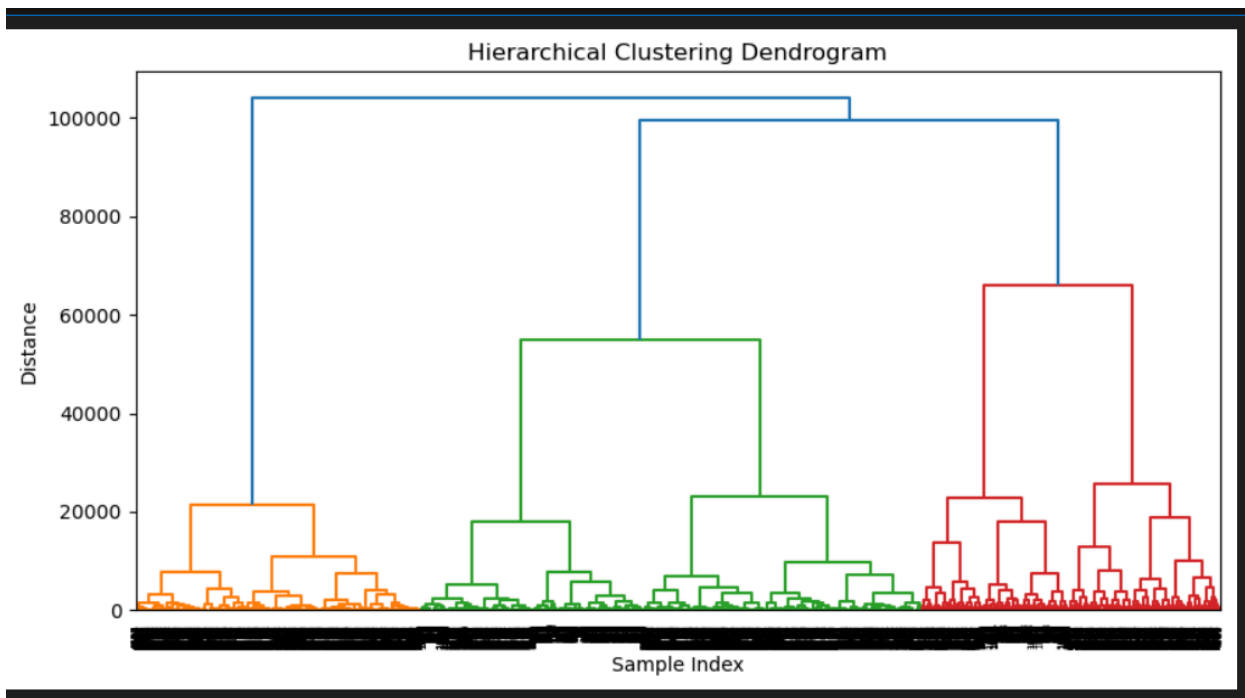
- `Z = hierarchy.linkage(data[numerical_cols].iloc[1:5000,:], method='ward')`: Computes the hierarchical clustering of the data. It uses the `linkage` function from SciPy's `hierarchy` module.
 - `data[numerical_cols].iloc[1:5000,:]`: Selects rows 1 to 5000 and all columns from the `data` DataFrame that are numerical (assuming `data` is a pandas DataFrame).
 - `method='ward'`: Specifies the linkage method for the hierarchical clustering, which in this case is Ward's method. Other methods include 'single', 'complete', 'average', etc.

3. Plot the Dendrogram:

- `plt.figure(figsize=(10, 5))`: Creates a new figure with a specific size (width=10, height=5) using Matplotlib.
- `dn = hierarchy.dendrogram(Z)`: Plots the dendrogram using the hierarchical clustering result `Z`. The `dendrogram` function from SciPy's `hierarchy` module is used to plot the dendrogram.
- `plt.title('Hierarchical Clustering Dendrogram')`: Sets the title of the plot.
- `plt.xlabel('Sample Index')`: Sets the label for the x-axis.

- `plt.ylabel('Distance')`: Sets the label for the y-axis.
- `plt.show()`: Displays the plot.

The dendrogram provides a visual representation of the hierarchical clustering, showing how the data points are grouped together based on their similarity or distance.



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Perform hierarchical clustering
Z1 = linkage(best_numerical_data, method='single', metric='euclidean') # minimum
Z2 = linkage(best_numerical_data, method='complete', metric='euclidean') # maximum
Z3 = linkage(best_numerical_data, method='average', metric='euclidean') # average
Z4 = linkage(best_numerical_data, method='ward', metric='euclidean') # variance
```

This Python code snippet demonstrates how to perform hierarchical clustering using different linkage methods (single, complete, average, and Ward's method) and visualize the resulting dendrograms.

1. Import Libraries:

- `import numpy as np`: Imports the NumPy library.
- `import matplotlib.pyplot as plt`: Imports the Matplotlib library for plotting.
- `from scipy.cluster.hierarchy import dendrogram, linkage`: Imports the `dendrogram` and `linkage` functions from the `scipy.cluster.hierarchy` module for hierarchical clustering.

2. Perform Hierarchical Clustering:

- `Z1 = linkage(best_numerical_data, method='single', metric='euclidean')`: Performs hierarchical clustering using the single linkage method (minimum distance) and Euclidean distance metric.
- `Z2 = linkage(best_numerical_data, method='complete', metric='euclidean')`: Performs hierarchical clustering using the complete linkage method (maximum distance) and Euclidean distance metric.
- `Z3 = linkage(best_numerical_data, method='average', metric='euclidean')`: Performs hierarchical clustering using the average linkage method (average distance) and Euclidean distance metric.
- `Z4 = linkage(best_numerical_data, method='ward', metric='euclidean')`: Performs hierarchical clustering using Ward's method (minimizing within-cluster variance) and Euclidean distance metric.

3. Visualize Dendrograms:

- Dendrograms are visual representations of the clustering results, showing how data points are grouped together based on their similarity.


```
plt.figure(figsize=(10, 6))
dendrogram(Z1)
plt.title('Single Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

plt.figure(figsize=(10, 6))
dendrogram(Z2)
plt.title('Complete Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

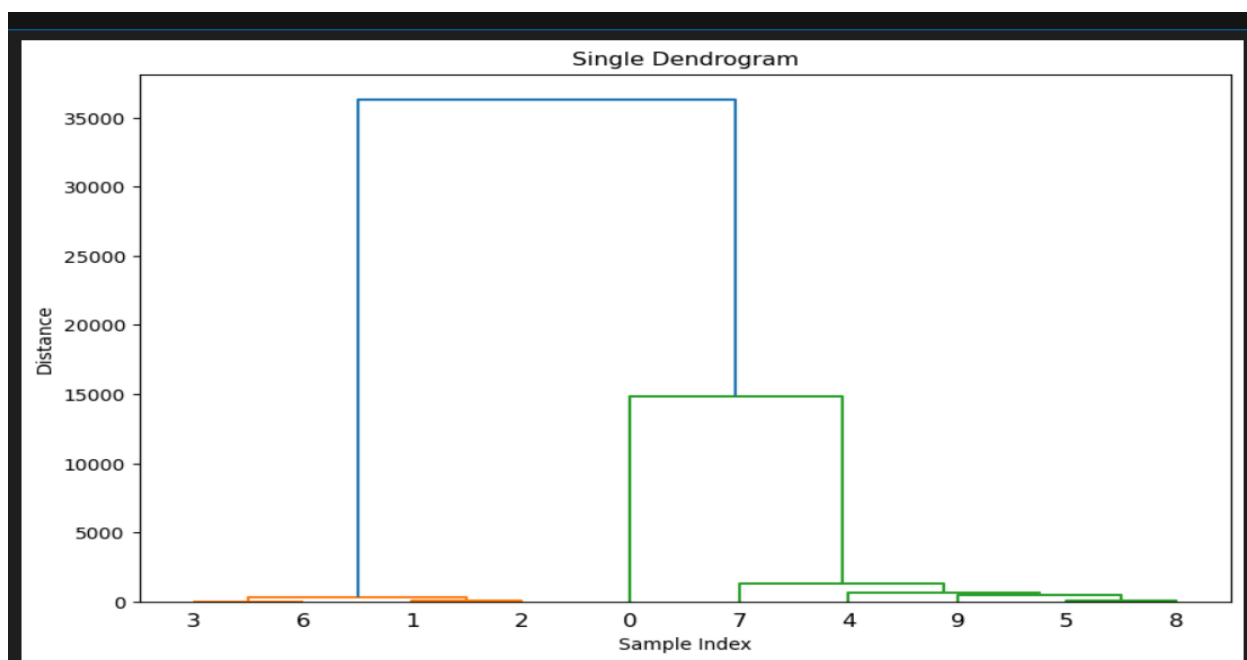
plt.figure(figsize=(10, 6))
dendrogram(Z3)
plt.title('Average Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

plt.figure(figsize=(10, 6))
dendrogram(Z4)
plt.title('Ward Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

This code snippet visualizes the dendrograms for hierarchical clustering using different linkage methods (single, complete, average, and Ward's method). Each dendrogram shows how the data points are grouped together based on their similarity or distance.

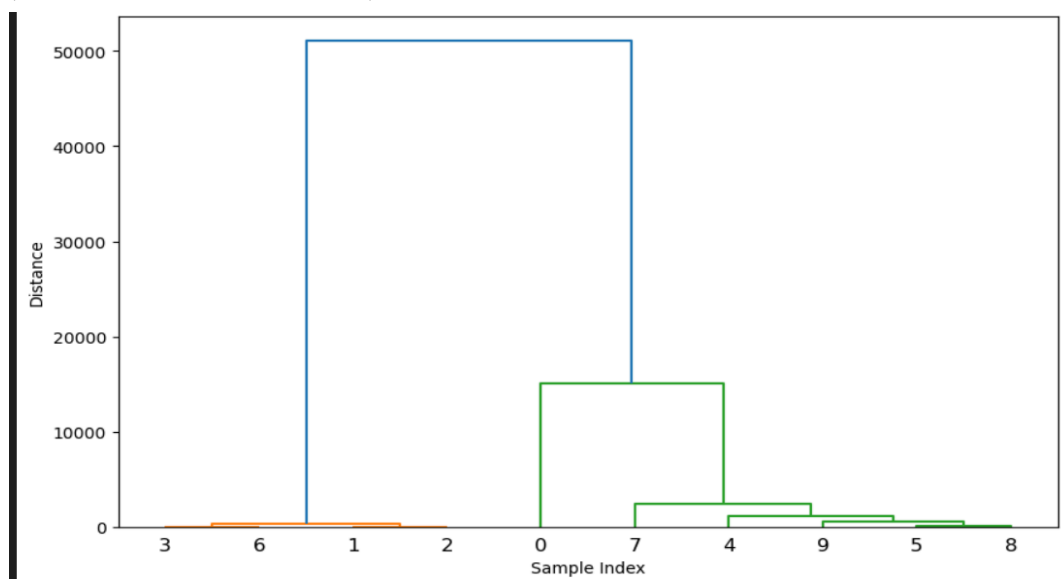
```
plt.figure(figsize=(10, 6))
dendrogram(Z1)
plt.title('Single Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

- This code creates a figure with a specified size using `plt.figure(figsize=(10, 6))`.
- The `dendrogram` function is used to plot the dendrogram for the single linkage method (`Z1`).
- `plt.title`, `plt.xlabel`, and `plt.ylabel` are used to set the title, x-axis label, and y-axis label, respectively.



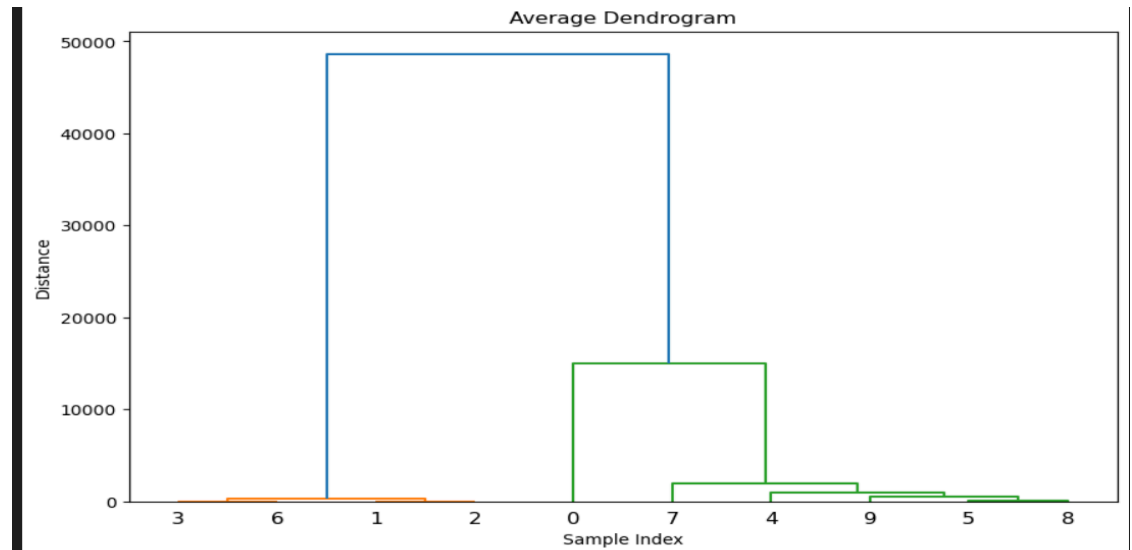
2-Complete Linkage Dendrogram:

- Similar to the single linkage dendrogram, but with z_2 and a different title ('Complete Dendrogram').



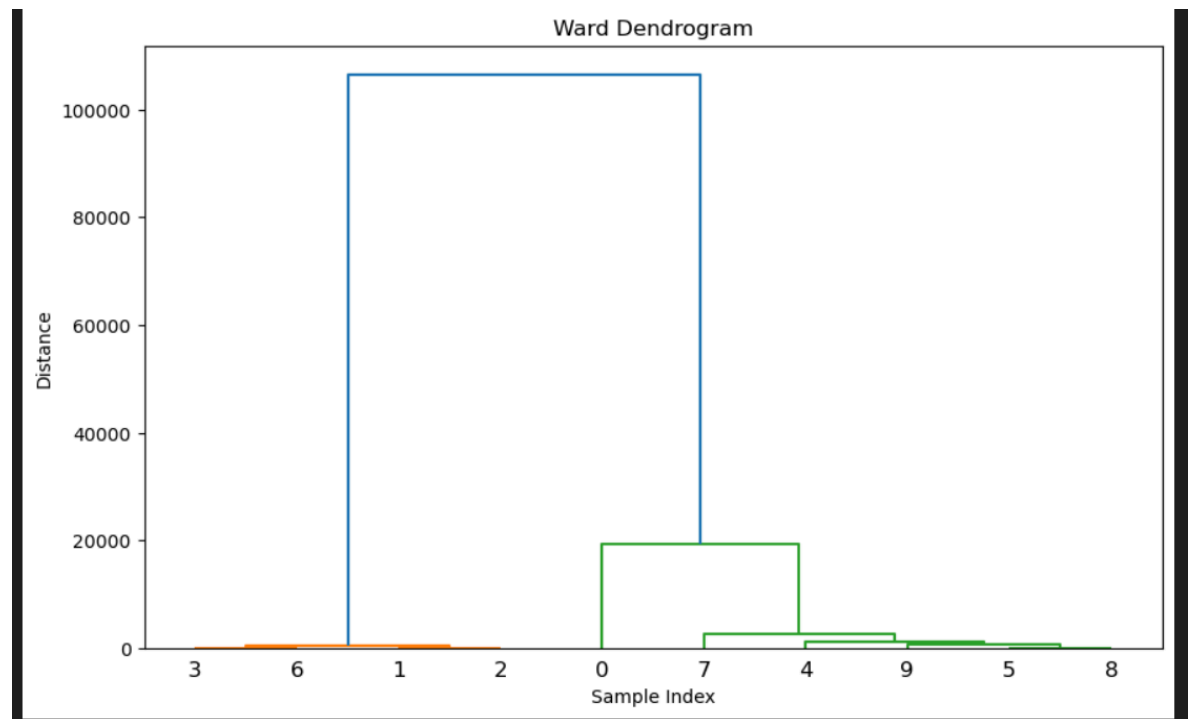
3-Average Linkage Dendrogram:

- Similar to the single linkage dendrogram, but with z_3 and a different title ('Average Dendrogram').



4-Ward's Method Dendrogram:

- Similar to the single linkage dendrogram, but with z_4 and a different title ('Ward Dendrogram').



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Perform hierarchical clustering
Z1 = linkage(worst_numerical_data, method='single', metric='euclidean') # minimum
Z2 = linkage(worst_numerical_data, method='complete', metric='euclidean') # maximum
Z3 = linkage(worst_numerical_data, method='average', metric='euclidean') # average
Z4 = linkage(worst_numerical_data, method='ward', metric='euclidean') # variance
```

This Python code snippet performs hierarchical clustering using different linkage methods (single, complete, average, and Ward's method) on the `worst_numerical_data` dataset and visualizes the resulting dendrograms.

1. Import Libraries:

- `import numpy as np`: Imports the NumPy library.
- `import matplotlib.pyplot as plt`: Imports the Matplotlib library for plotting.
- `from scipy.cluster.hierarchy import dendrogram, linkage`: Imports the `dendrogram` and `linkage` functions from the `scipy.cluster.hierarchy` module for hierarchical clustering.

2. Perform Hierarchical Clustering:

- `Z1 = linkage(worst_numerical_data, method='single', metric='euclidean')`: Performs hierarchical clustering using the single linkage method (minimum distance) and Euclidean distance metric on the `worst_numerical_data`.
- `Z2 = linkage(worst_numerical_data, method='complete', metric='euclidean')`: Performs hierarchical clustering using the complete linkage method (maximum distance) and Euclidean distance metric.
- `Z3 = linkage(worst_numerical_data, method='average', metric='euclidean')`: Performs hierarchical clustering using the average linkage method (average distance) and Euclidean distance metric.
- `Z4 = linkage(worst_numerical_data, method='ward', metric='euclidean')`: Performs hierarchical clustering using Ward's method (minimizing within-cluster variance) and Euclidean distance metric.

3. Visualize Dendrograms:

- Dendrograms are visual representations of the clustering results, showing how data points are grouped together based on their similarity

```
plt.figure(figsize=(12, 6))

plt.subplot(221)
plt.title('Single Linkage')
dendrogram(Z1)

plt.subplot(222)
plt.title('Complete Linkage')
dendrogram(Z2)

plt.subplot(223)
plt.title('Average Linkage')
dendrogram(Z3)

plt.subplot(224)
plt.title("Ward's Method")
dendrogram(Z4)

plt.tight_layout()
plt.show()
```



- Each subplot corresponds to a different linkage method, showing the dendrogram for that method. The `title` function is used to set the title for each subplot. `plt.tight_layout()` ensures that the subplots are properly spaced. `plt.show()` displays the plot.

This code snippet provides a comprehensive overview of how different linkage methods can affect the hierarchical clustering results and how to visualize these results using dendrograms.

```
plt.figure(figsize=(10, 6))
dendrogram(Z1)
plt.title('Single Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

plt.figure(figsize=(10, 6))
dendrogram(Z2)
plt.title('Complete Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

plt.figure(figsize=(10, 6))
dendrogram(Z3)
plt.title('Average Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

plt.figure(figsize=(10, 6))
dendrogram(Z4)
plt.title('Ward Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

This code snippet visualizes the dendrograms for hierarchical clustering using different linkage methods (single, complete, average, and Ward's method) on the `worst_numerical_data` dataset.

First (single Dendrogram)

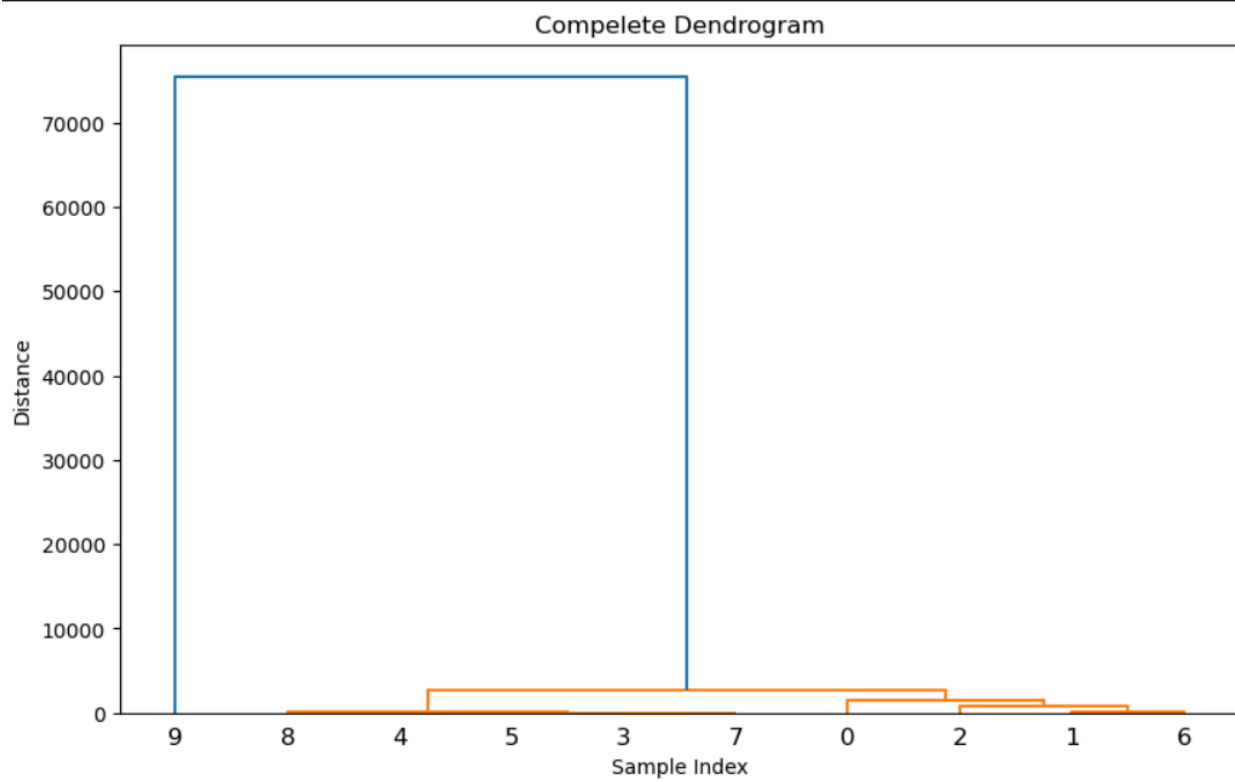
- This code creates a figure with a specified size using `plt.figure(figsize=(10, 6))`.
- The `dendrogram` function is used to plot the dendrogram for the single linkage method (`Z1`).
- `plt.title`, `plt.xlabel`, and `plt.ylabel` are used to set the title, x-axis label, and y-axis label, respectively.

First (single Dendrogram)

- This code creates a figure with a specified size using `plt.figure(figsize=(10, 6))`.
- The `dendrogram` function is used to plot the dendrogram for the single linkage method (`Z1`).
- `plt.title`, `plt.xlabel`, and `plt.ylabel` are used to set the title, x-axis label, and y-axis label, respectively.

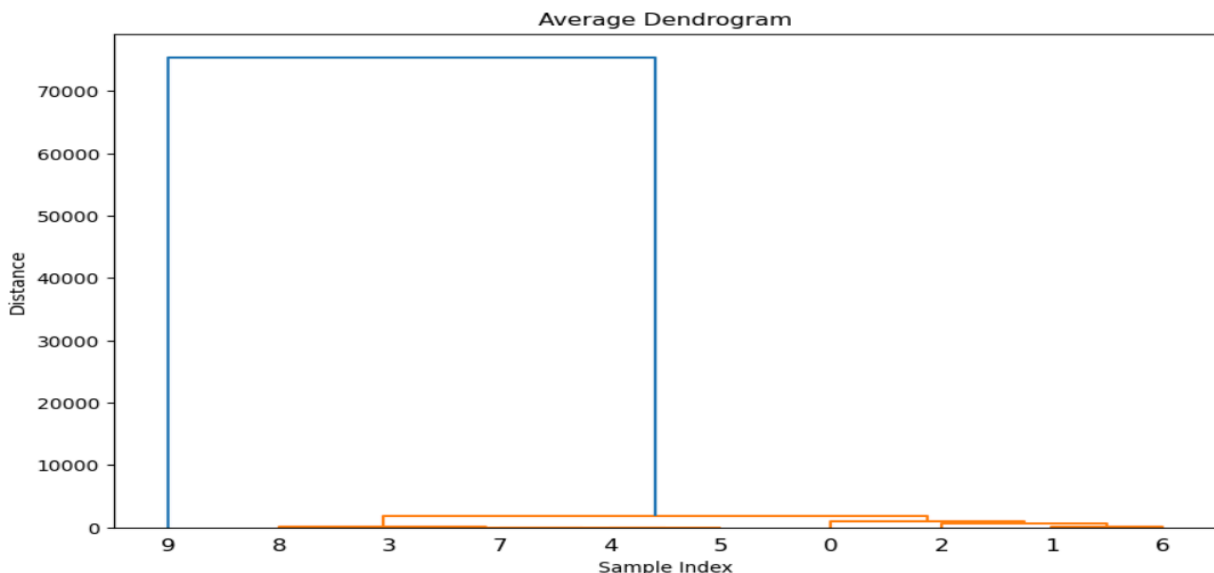
Complete Linkage Dendrogram:

- Similar to the single linkage dendrogram, but with z_2 and a different title ('Complete Dendrogram').



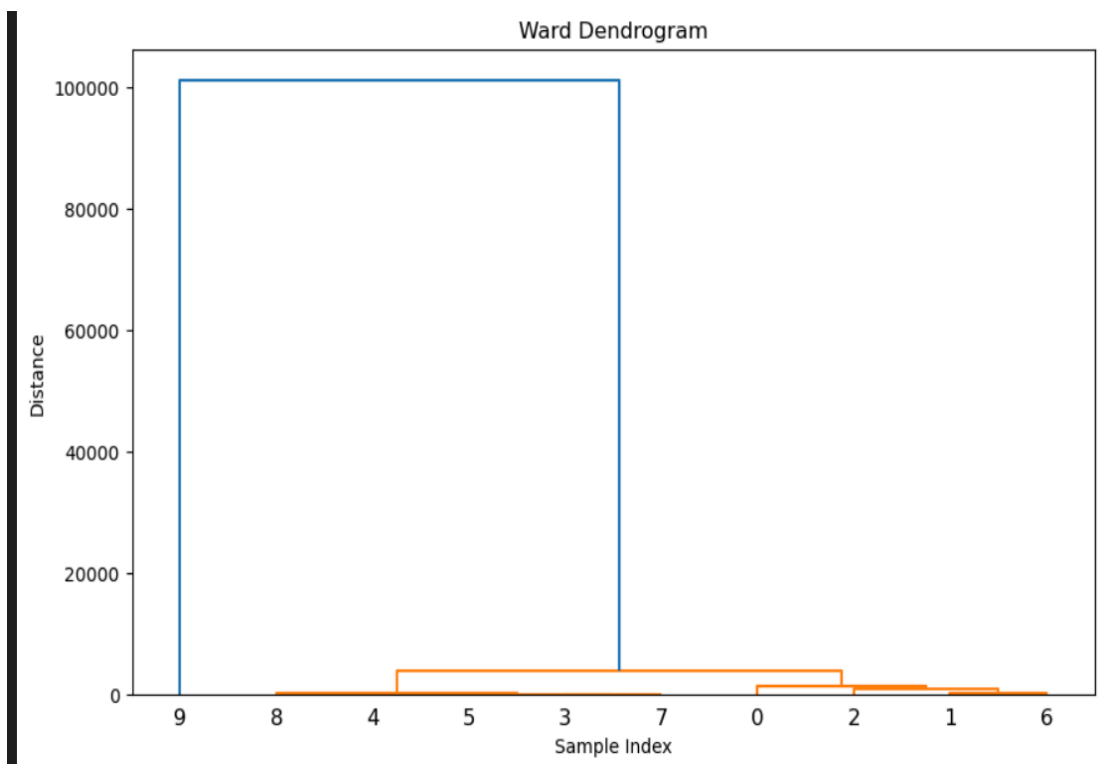
1. Average Linkage Dendrogram:

- Similar to the single linkage dendrogram, but with z_3 and a different title ('Average Dendrogram').



Ward's Method Dendrogram:

- Similar to the single linkage dendrogram, but with z_4 and a different title ('Ward Dendrogram').



III. FUZZY

```
# Define linguistic variables and terms
dep_arriv = ctrl.Antecedent(np.arange(0, 6, 1), 'Departure and Arrival Time Convenience')
booking = ctrl.Antecedent(np.arange(0, 6, 1), 'Ease of Online Booking')
check_in = ctrl.Antecedent(np.arange(0, 6, 1), 'Check-in Service')
online_boarding = ctrl.Antecedent(np.arange(0, 6, 1), 'Online Boarding')
gate = ctrl.Antecedent(np.arange(0, 6, 1), 'Gate Location')
on_board = ctrl.Antecedent(np.arange(0, 6, 1), 'On-board Service')
seat_comfort = ctrl.Antecedent(np.arange(0, 6, 1), 'Seat Comfort')
leg_room = ctrl.Antecedent(np.arange(0, 6, 1), 'Leg Room Service')
cleanliness = ctrl.Antecedent(np.arange(0, 6, 1), 'Cleanliness')
food_drink = ctrl.Antecedent(np.arange(0, 6, 1), 'Food and Drink')
in_flight = ctrl.Antecedent(np.arange(0, 6, 1), 'In-flight Service')
in_flight_wifi = ctrl.Antecedent(np.arange(0, 6, 1), 'In-flight Wifi Service')
in_flight_ent = ctrl.Antecedent(np.arange(0, 6, 1), 'In-flight Entertainment')
bag = ctrl.Antecedent(np.arange(0, 6, 1), 'Baggage Handling')

satisfaction = ctrl.Consequent(np.arange(0, 6, 1), 'satisfaction')
```

This code sets up the foundational structure for a fuzzy logic system used to model and control a satisfaction level based on various input criteria related to airline service quality. The actual implementation would involve defining membership functions, establishing fuzzy rules, and performing inference using these variables to compute the output satisfaction level.

```
# Define membership functions
dep_arriv['low'] = fuzz.trimf(dep_arriv.universe, [0, 0, 2])
dep_arriv['medium'] = fuzz.trimf(dep_arriv.universe, [1, 2, 4])
dep_arriv['high'] = fuzz.trimf(dep_arriv.universe, [3, 4, 5])

booking['low'] = fuzz.trimf(booking.universe, [0, 0, 2])
booking['medium'] = fuzz.trimf(booking.universe, [1, 2, 4])
booking['high'] = fuzz.trimf(booking.universe, [3, 4, 5])

check_in['low'] = fuzz.trimf(check_in.universe, [0, 0, 2])
check_in['medium'] = fuzz.trimf(check_in.universe, [1, 2, 4])
check_in['high'] = fuzz.trimf(check_in.universe, [3, 4, 5])

online_boarding['low'] = fuzz.trimf(online_boarding.universe, [0, 0, 2])
online_boarding['medium'] = fuzz.trimf(online_boarding.universe, [1, 2, 4])
online_boarding['high'] = fuzz.trimf(online_boarding.universe, [3, 4, 5])

gate['low'] = fuzz.trimf(gate.universe, [0, 0, 2])
gate['medium'] = fuzz.trimf(gate.universe, [1, 2, 4])
gate['high'] = fuzz.trimf(gate.universe, [3, 4, 5])

on_board['low'] = fuzz.trimf(on_board.universe, [0, 0, 2])
on_board['medium'] = fuzz.trimf(on_board.universe, [1, 2, 4])
on_board['high'] = fuzz.trimf(on_board.universe, [3, 4, 5])
```

```
seat_comfort['low'] = fuzz.trimf(seat_comfort.universe, [0, 0, 2])
seat_comfort['medium'] = fuzz.trimf(seat_comfort.universe, [1, 2, 4])
seat_comfort['high'] = fuzz.trimf(seat_comfort.universe, [3, 4, 5])

leg_room['low'] = fuzz.trimf(leg_room.universe, [0, 0, 2])
leg_room['medium'] = fuzz.trimf(leg_room.universe, [1, 2, 4])
leg_room['high'] = fuzz.trimf(leg_room.universe, [3, 4, 5])

cleanliness['low'] = fuzz.trimf(cleanliness.universe, [0, 0, 2])
cleanliness['medium'] = fuzz.trimf(cleanliness.universe, [1, 2, 4])
cleanliness['high'] = fuzz.trimf(cleanliness.universe, [3, 4, 5])

food_drink['low'] = fuzz.trimf(food_drink.universe, [0, 0, 2])
food_drink['medium'] = fuzz.trimf(food_drink.universe, [1, 2, 4])
food_drink['high'] = fuzz.trimf(food_drink.universe, [3, 4, 5])

in_flight['low'] = fuzz.trimf(in_flight.universe, [0, 0, 2])
in_flight['medium'] = fuzz.trimf(in_flight.universe, [1, 2, 4])
in_flight['high'] = fuzz.trimf(in_flight.universe, [3, 4, 5])

in_flight_wifi['low'] = fuzz.trimf(in_flight_wifi.universe, [0, 0, 2])
in_flight_wifi['medium'] = fuzz.trimf(in_flight_wifi.universe, [1, 2, 4])
in_flight_wifi['high'] = fuzz.trimf(in_flight_wifi.universe, [3, 4, 5])
```

```

in_flight_ent['low'] = fuzz.trimf(in_flight_ent.universe, [0, 0, 2])
in_flight_ent['medium'] = fuzz.trimf(in_flight_ent.universe, [1, 2, 4])
in_flight_ent['high'] = fuzz.trimf(in_flight_ent.universe, [3, 4, 5])

bag['low'] = fuzz.trimf(bag.universe, [0, 0, 2])
bag['medium'] = fuzz.trimf(bag.universe, [1, 2, 4])
bag['high'] = fuzz.trimf(bag.universe, [3, 4, 5])

satisfaction['low'] = fuzz.trimf(satisfaction.universe, [0, 0, 2])
satisfaction['medium'] = fuzz.trimf(satisfaction.universe, [1, 2, 4])
satisfaction['high'] = fuzz.trimf(satisfaction.universe, [3, 4, 5])

```

- This code sets up the linguistic terms ('low', 'medium', 'high') and their associated membership functions for each input and output variable within a fuzzy logic system.
- These membership functions are used to fuzzify crisp input values (e.g., ratings for departure and arrival convenience) into fuzzy values, which are then used to infer the output (satisfaction level) using fuzzy rules defined in the subsequent steps of the fuzzy logic system.

This is a foundational step in defining a fuzzy logic controller where input variables are fuzzified using these membership functions, and fuzzy rules are applied to compute the output (satisfaction level) based on the defined linguistic terms and fuzzy logic operations.

```

# Define fuzzy rules
rule1 = ctrl.Rule(dep_arriv['low'] | booking['low'] | check_in['low'] | online_boarding['low'] |
gate['low'] | on_board['low'] | seat_comfort['low'] | leg_room['low'] |
cleanliness['low'] | food_drink['low'] | in_flight['low']
| in_flight_wifi['low'] | in_flight_ent['low'] | bag['low']
| , satisfaction['low'])

rule2 = ctrl.Rule(dep_arriv['medium'] | booking['medium'] | check_in['medium'] | online_boarding['medium'] |
gate['medium'] | on_board['medium'] | seat_comfort['medium'] | leg_room['medium'] |
cleanliness['medium'] | food_drink['medium'] | in_flight['medium']
| in_flight_wifi['medium'] | in_flight_ent['medium'] | bag['medium']
| , satisfaction['medium'])

rule3 = ctrl.Rule(dep_arriv['high'] | booking['high'] | check_in['high'] | online_boarding['high'] |
gate['high'] | on_board['high'] | seat_comfort['high'] | leg_room['high'] |
cleanliness['high'] | food_drink['high'] | in_flight['high']
| in_flight_wifi['high'] | in_flight_ent['high'] | bag['high']
| , satisfaction['high'])

# Create a fuzzy control system
satisfaction_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
satisfaction_level = ctrl.ControlSystemSimulation(satisfaction_ctrl)

```

This setup allows you to implement a fuzzy logic controller where input linguistic variables (`dep_arriv`, `booking`, etc.) are evaluated based on fuzzy rules to determine an output satisfaction level (`satisfaction`). The actual implementation would involve providing input values, computing the output using fuzzy inference, and interpreting the resulting satisfaction level.

```
input_types=['Departure and Arrival Time Convenience', 'Ease of Online Booking',
            'Check-in Service', 'Online Boarding', 'Gate Location',
            'On-board Service', 'Seat Comfort', 'Leg Room Service', 'Cleanliness',
            'Food and Drink', 'In-flight Service', 'In-flight Wifi Service',
            'In-flight Entertainment', 'Baggage Handling']
for j in range(10): #iterate over 10 rows
    print("****40")
    print('Customer ',j," = ")
    for i, types in zip(range(14), input_types): #iterate over columns
        input=best_numerical_data[input_types].iloc[j].iloc[i]
        satisfaction_level.input[types]=input
        print(types," rating= ",input)

    # Compute satisfaction level
    satisfaction_level.compute()

    # Output satisfaction level
    print("Satisfaction Level of customer ",satisfaction_level.output['satisfaction'])
```

- The code iterates over 10 rows (customers) from `best_numerical_data`.
- For each customer, it retrieves ratings for 14 input variables (`input_types`) and sets these as inputs to a fuzzy control system (`satisfaction_level`).
- It then computes the satisfaction level based on the inputs and prints the computed satisfaction level for each customer.

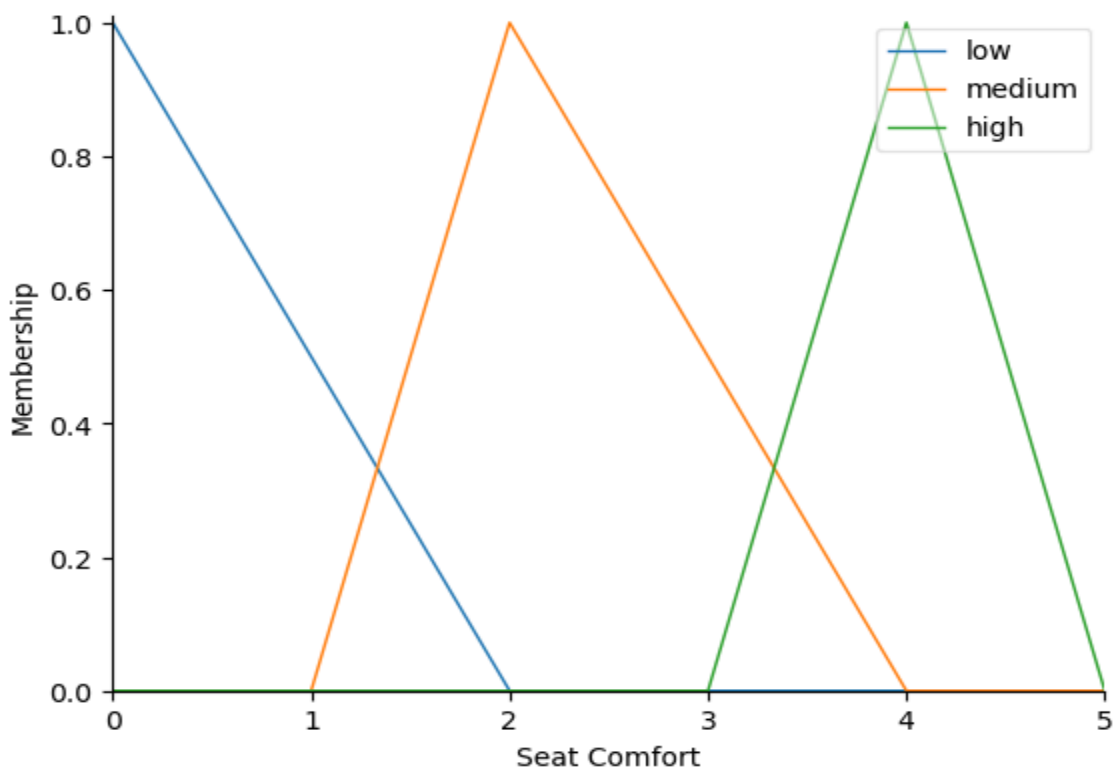
This process allows you to evaluate and compute the satisfaction level for multiple customers based on their ratings across different input variables using a fuzzy logic system that has been defined earlier in your code. Each customer's input ratings are processed through the fuzzy control system to generate a satisfaction level output.

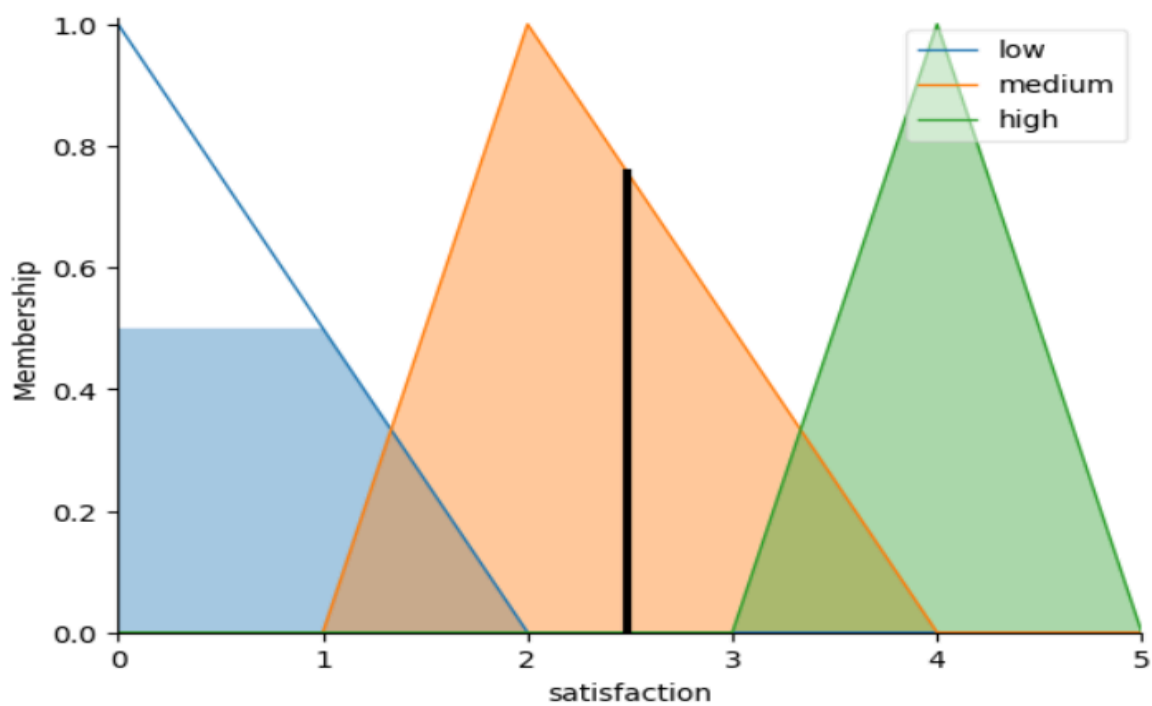
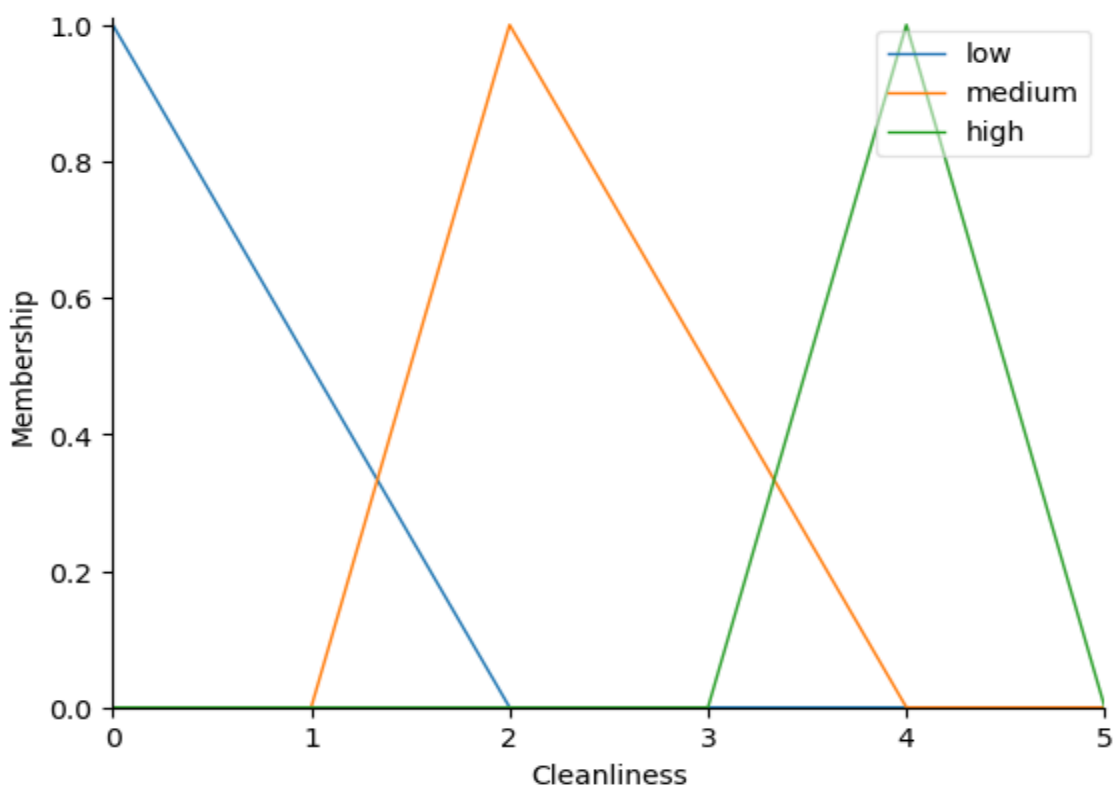
Output example:

```
Customer 0 =
Departure and Arrival Time Convenience rating= 2
Ease of Online Booking rating= 2
Check-in Service rating= 1
Online Boarding rating= 4
Gate Location rating= 2
On-board Service rating= 3
Seat Comfort rating= 3
Leg Room Service rating= 2
Cleanliness rating= 3
Food and Drink rating= 3
In-flight Service rating= 5
In-flight Wifi Service rating= 3
In-flight Entertainment rating= 3
Baggage Handling rating= 5
Satisfaction Level of customer 2.487179487179487
```

```
seat_comfort.view()
cleanliness.view()
satisfaction.view()
```

These visualizations serve as a valuable tool for interpreting and analyzing the behavior of a fuzzy logic controller, especially in scenarios where complex decision-making or inference based on linguistic rules is involved.





IV. K-Means

1. Loop through 10 different values of k:
 - a. Inside the loop, create a KMeans object with the current value of k.
 - b. It fits the KMeans model to the standardized numerical data using `kmeans.fit(data[numerical_cols])`.
 - c. After fitting the model, it calculates the inertia of the clusters using `kmeans.inertia_`.
 - d. The inertia measures the compactness or tightness of the clusters formed by K-means.

```
from sklearn.cluster import KMeans
inertia = [] # tightness of clusters
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data[numerical_cols]) #standard data
    inertia.append(kmeans.inertia_)
```

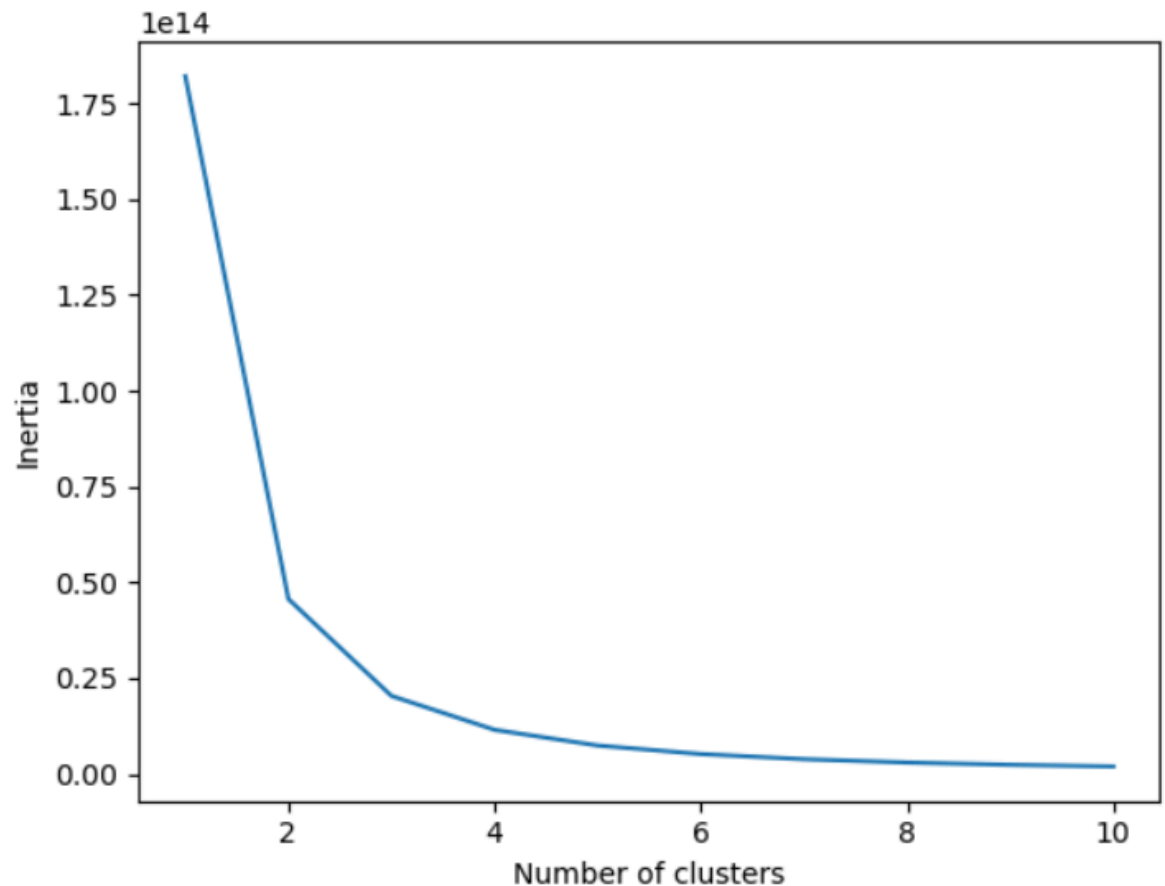
2. Plot the Curve:
 - a. After collecting inertia values for different values of k, plot them against the corresponding values of k .
 - b. The x-axis represents the number of clusters (k), and the y-axis represents the inertia.

```
# Plot curve to visualize the optimal k
import matplotlib.pyplot as plt
plt.plot(range(1, 11), inertia)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

# From the curve, choose the optimal k value
```

3. Choose Optimal K:

- The line on the graph slopes downwards as the number of clusters increases, it means that the k-means algorithm is grouping the data points into tighter clusters.
- Then the line starts to flatten out, it indicates that adding more clusters is leading to overfitting, where the clusters become too specific.
- The point where the inertia starts to decrease at a slower rate is considered the optimal value of k.



4. K-means Clustering:

- It initializes a KMeans object with the Optimal k
- Then, it fits the KMeans model to the numerical columns of the dataset using `kmeans.fit(data[numerical_cols])`.

5. Adding Cluster Labels:

- After clustering, it adds the cluster labels obtained from K-means clustering to the original dataset using `data['Cluster'] = kmeans.labels_`.
- Each data point in the dataset is assigned a cluster label indicating which cluster it belongs to.

6. Cluster Centroids:

- It calculates the centroids of the clusters using `kmeans.cluster_centers_`.
- These centroids represent the (mean) coordinates of the data points in each cluster.

```
# Perform K-means clustering with the chosen k value
k = 3
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(data[numerical_cols])

# Add cluster labels to the original dataset
data['Cluster'] = kmeans.labels_

# Print the cluster centroids
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=numerical_cols)
print("Cluster Centroids:")
print(centroids)

# Print the counts of data points in each cluster
print("Counts per Cluster:")
print(data['Cluster'].value_counts())
```

```

Cluster Centroids:
      ID      Age  Flight Distance  Departure Delay  Arrival Delay \
0  21571.099183  38.882599    1046.813127    15.514111    16.905121
1  108156.809130  39.865999    1236.448509    14.933946    14.445186
2   64799.071425  39.533675    1286.555468    13.485062    13.933600

      Departure and Arrival Time Convenience  Ease of Online Booking \
0                                3.060794                2.730183
1                                3.053314                2.770356
2                                3.057969                2.769651

      Check-in Service  Online Boarding  Gate Location  On-board Service \
0          3.163002            3.149927        2.978967        3.278007
1          3.423791            3.341488        2.975974        3.465080
2          3.330863            3.265969        2.975798        3.405762

      Seat Comfort  Leg Room Service  Cleanliness  Food and Drink \
0          3.346153            3.265629        3.239152        3.205323
1          3.524580            3.414559        3.321709        3.197378
2          3.453310            3.372435        3.297466        3.211381

      In-flight Service  In-flight Wifi Service  In-flight Entertainment \
0          3.507410                2.769386                3.351225
1          3.751523                2.681984                3.358313
2          3.667192                2.734610                3.364630

      Baggage Handling
0          3.509970
1          3.737352
2          3.647413

```

7. Counts per Cluster:

- It prints the counts of data points in each cluster using `data['Cluster'].value_counts()`.

```

Counts per Cluster:
Cluster
1    43288
2    43181
0    43018
Name: count, dtype: int64

```

V. Logistic Regression

1. **Defining features and target:** Assuming that `data` is a DataFrame containing both the features and the target variable, the code extracts the numerical features into variable `X` and the target variable `Satisfaction` into variable `y`.
2. **Splitting the dataset:** splits the dataset into training and testing sets using where 70% of the data for training (`X_train, y_train`) and 30% for testing (`X_test, y_test`).
3. **Creating and training the model:** create a logistic regression model using `LogisticRegression()` and then train the model using the training data (`X_train, y_train`) with the `fit()` method.
4. **Making predictions:** After training, make predictions on the test data (`X_test`) using the `predict()` method of the trained model.
5. **Evaluation metrics:** calculates evaluation metrics such as precision, recall, F1-score, and support using `classification_report`. It also calculates accuracy using `accuracy_score` and prints the results.

```
# Assuming 'data' contains both features and target variable
X = data[numerical_cols] # Features
y = data['Satisfaction']  # Target

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

# Creating and training the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Additional evaluation metrics
print(classification_report(y_test, y_pred))

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100,"%")
```

	precision	recall	f1-score	support
Neutral or Dissatisfied	0.65	0.72	0.68	21951
Satisfied	0.57	0.49	0.53	16896
accuracy			0.62	38847
macro avg	0.61	0.60	0.61	38847
weighted avg	0.62	0.62	0.61	38847

Accuracy: 61.95587818879192 %

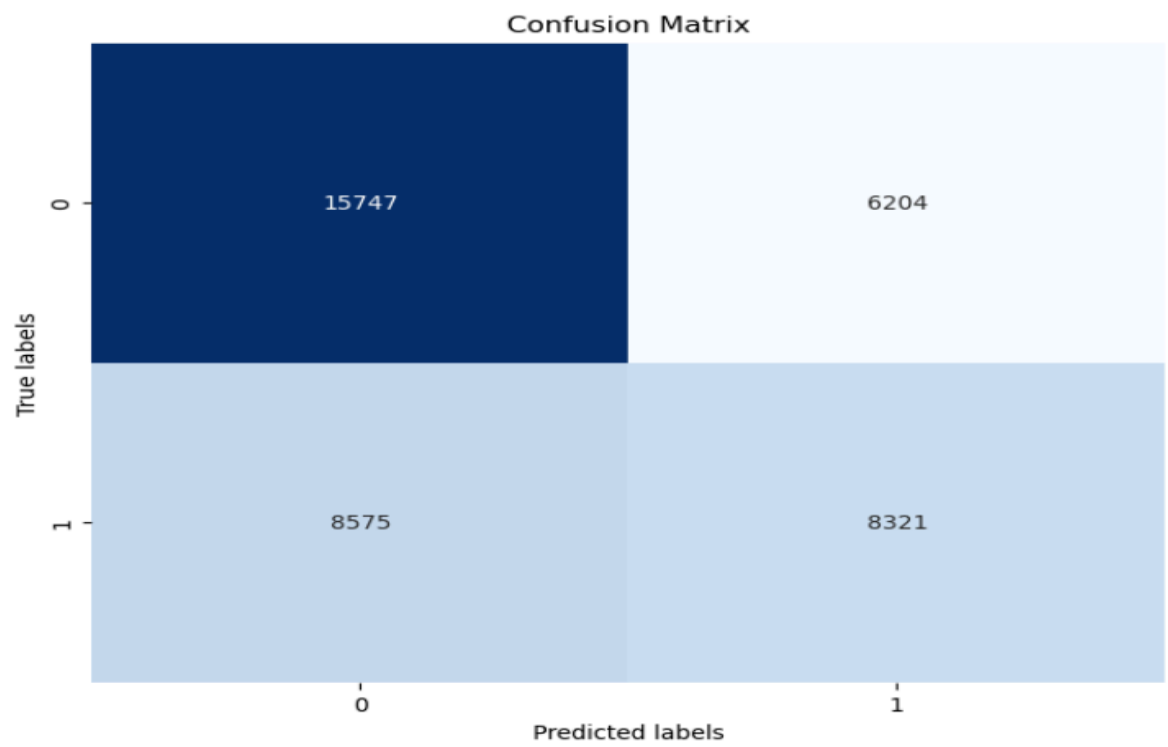
6. **Confusion Matrix:** The confusion matrix is a table that summarizes the performance of a classification model. It consists of four main components: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

```
# Calculating confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plotting confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

Confusion Matrix:

```
[[15747  6204]  
 [ 8575  8321]]
```



VI. Decision Tree

1. **Splitting the Dataset:** split dataset into training and testing sets. It uses 70% of the data for training (`x_train` and `y_train`) and 30% for testing (`x_test` and `y_test`).
2. **Creating and Training the Decision Tree Classifier:** create a `DecisionTreeClassifier` object and then train it using the training data (`x_train` and `y_train`).
3. **Making Predictions:** use the trained Decision Tree classifier (`dtree`) to make predictions on the testing data (`x_test`).
4. **Evaluation Metrics:** print a classification report, which includes precision, recall, F1-score, and support for each class and calculate the accuracy of the model by comparing the predicted labels (`y_pred`) with the actual labels (`y_test`). It then prints the accuracy percentage.

```
X = data[numerical_cols] # Features (all columns except the last one)
y = data['Satisfaction']  # Target variable (last column)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra

# Creating decision tree classifier
dtree = DecisionTreeClassifier()

# Training the classifier
dtree.fit(X_train, y_train)

# Making predictions
y_pred = dtree.predict(X_test)

# Additional evaluation metrics
print(classification_report(y_test, y_pred))

# Calculating accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100,"%")
```

5. **Confusion Matrix:** calculate the confusion matrix. The confusion matrix provides a detailed breakdown of the model's performance.

```
# Calculating confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plotting confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

```
Confusion Matrix:
[[20408  1543]
 [ 1458 15438]]
```

