Data Mining Project

Karaj Islamic Azad University – Spring 2024

Kimia Taslim

Dr. Amineh Amini

Project Description:

First, we will learn about the type of attributes of the selected dataset and draw a boxplot for it. Then, we will choose two types of clustering, display them, and compare them.

In the final step, we will draw a decision tree for the dataset using a classification algorithm.

For this project, the Mall Customers dataset has been selected to be examined. We will specify the type of attributes in this dataset:

```
customers.head(100)
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
95	96	Male	24	60	52
96	97	Female	47	60	47
97	98	Female	27	60	50
98	99	Male	48	61	42
99	100	Male	20	61	49

100 rows × 5 columns

As you can see, the data type for Genre is Nominal and for the rest, it is Numeric.

Now we add the required libraries in the coding environment:

```
[2] import pandas as pd  # For data manipulation
    import matplotlib.pyplot as plt  # For plotting
    import numpy as np  # For numerical operations
    import seaborn as sns  # For statistical data visualization
    import matplotlib.gridspec as gridspec  # For creating grid layouts in plots
    from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN  # Importing multiple clustering algorithms
    from sklearn.preprocessing import LabelEncoder  # For label encoding categorical variables
    from sklearn.preprocessing import StandardScaler  # For standardizing features
    from sklearn.metrics import silhouette_score, calinski_harabasz_score  # For clustering evaluation metrics
    from scipy.cluster.hierarchy import dendrogram, linkage  # For hierarchical clustering visualization and operations
    from sklearn.tree import DecisionTreeClassifier  # For Decision Tree classification
    from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
    from sklearn.metrics import accuracy_score  # For evaluating accuracy of models
    from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report  # For classification evaluation metrics
    from sklearn.naive_bayes import GaussianNB  # For Gaussian Naive Bayes classification
```

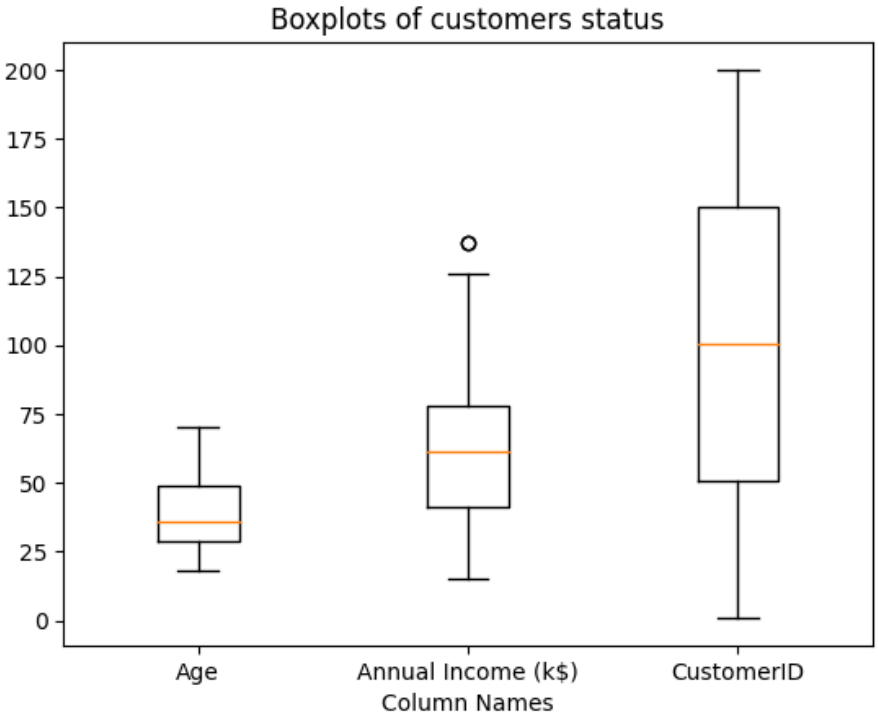
We add the desired dataset file to read the data and write the list of selected fields to draw the boxplot:

```
[27] # reading CSV file:
     customers = pd.read_csv("Mall_Customers.csv")
```

```
[29] columns_to_plot = ["Age", "Annual Income (k$)", "CustomerID"]  # List desired column names
     data_for_boxplot = customers[columns_to_plot]
```

We draw the box plot:

```
plt.boxplot(data_for_boxplot, labels=columns_to_plot)
plt.xlabel("Column Names")
plt.title("Boxplots of customers status")
plt.show()
```



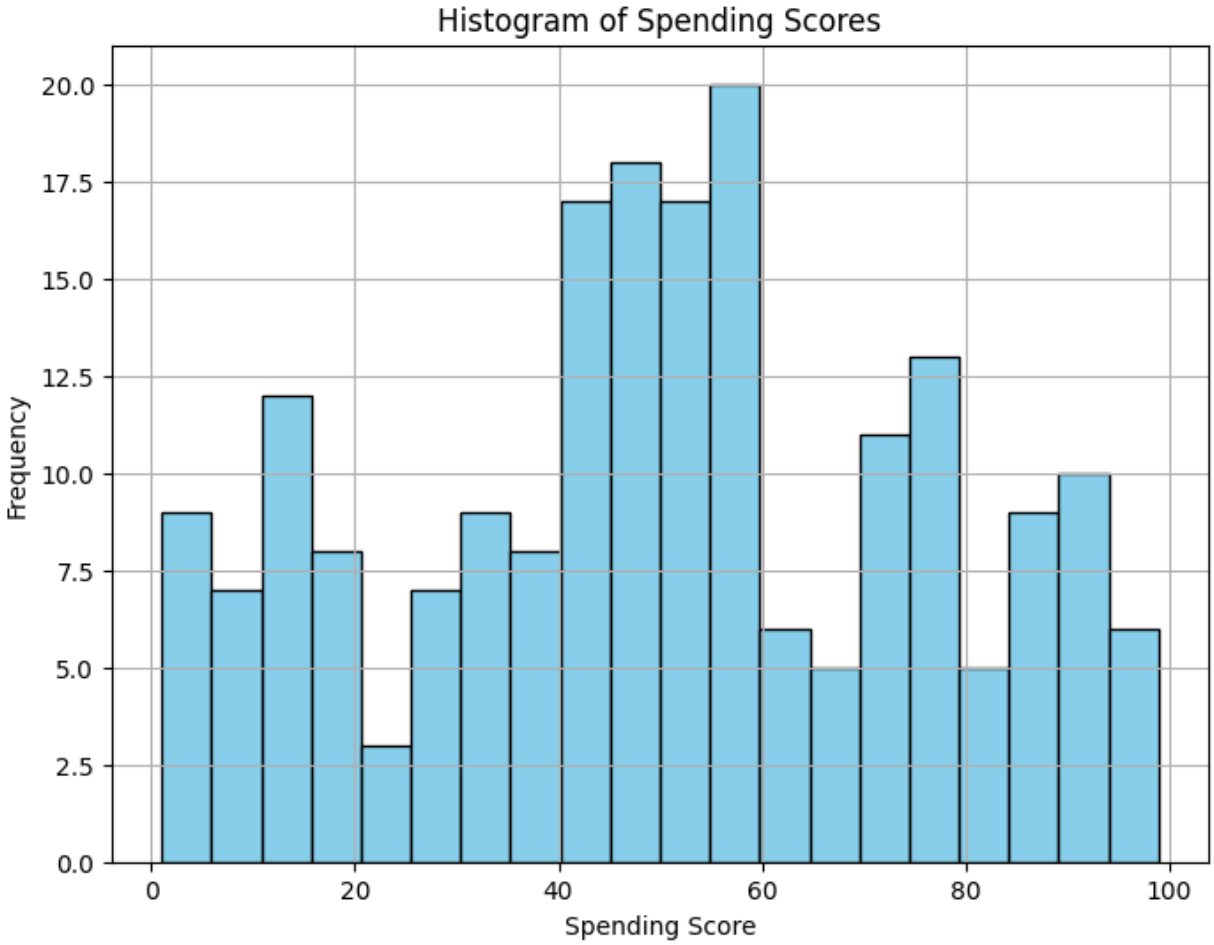
According to the box plot drawn, it can be said that the average age of customers in this data set is around 35 to 40, 25% of customers are under 30 years old, and 75% are over 40 years old.

For Annual Income, it is observed that the average annual income is around 60 to 70 thousand dollars. For CustomerID, since this is a numerical value between 1 and 200 in our dataset, its median value is identified as 100, which is exactly in the middle of our graph and shows that the data is normally distributed.

Given our dataset, we can draw other graphs such as Histogram, scatter plot, category plot.

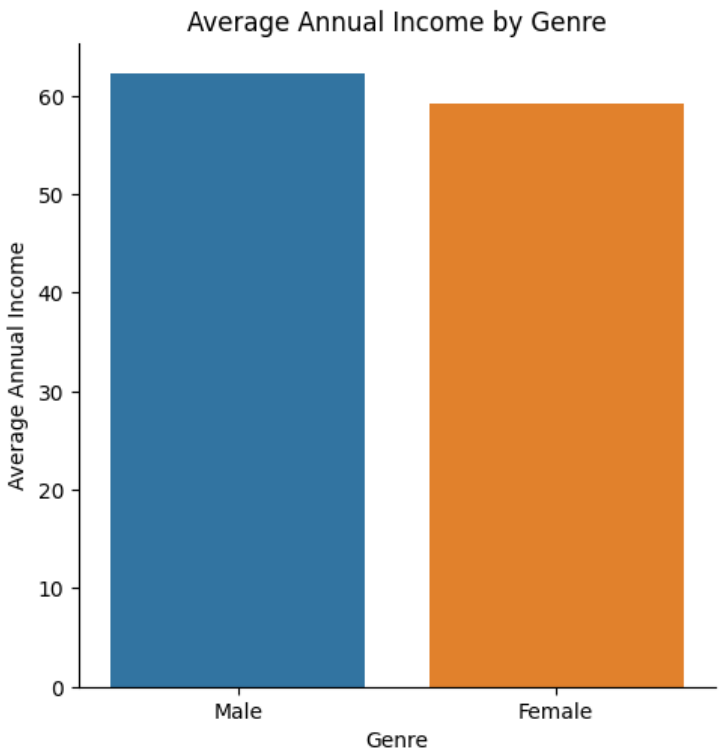
For example,

Histogram:



In this graph, we can say that the spending score is approximately in the range of 20 to 80. Most people's scores are between 40 and 60, about 20% of people have less than 20 points, and about 20% of people have more than 80 points.

Category Plot:



In this chart, it is easy to see that the average annual income of men is higher than that of women, with men receiving an average of about $60,000 to $65,000 per year and women receiving an average of $55,000 to $60,000 per year.

Clustering:

First, let's look at hierarchical clustering:

```
[33] label_encoder = LabelEncoder()
     customers['Genre'] = label_encoder.fit_transform(customers['Genre'])
```

```
[34] scaler = StandardScaler()
     numeric_fields = customers.drop(['Genre'], axis=1)
     scaled_data = scaler.fit_transform(numeric_fields)
```

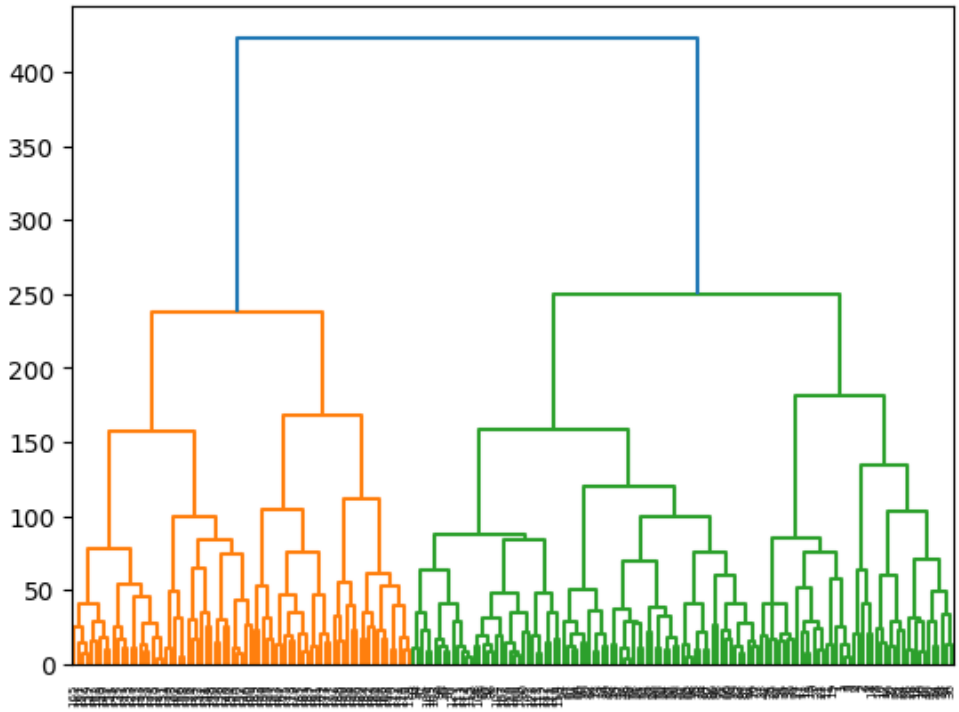
```
     # Perform hierarchical clustering
     linked = linkage(scaled_data, method='ward')  # You can choose different linkage methods
```

+ Code + Text

```
[40] # Plot a dendrogram to visualize the clusters
     dendrogram(linkage(customers, method="complete", metric='cityblock'))
     plt.show()
```

In this code, we first convert the nominal values of the categories into numbers and then scale the numerical data. Then, the hierarchical clustering method is used to analyze different

patterns in the customer data and finally, the tree diagram shows the relationships between the clusters and the data.



Density clustering review

First, we select the numeric columns for clustering:

```
#for DBSCAN
# Select the numeric columns for clustering
NumCl = customers[['Age', 'Annual Income (k$)', 'Spending Score (1-100)', 'CustomerID']]
NumCl.head()
```

	Age	Annual Income (k$)	Spending Score (1-100)	CustomerID
0	19	15	39	1
1	21	15	81	2
2	20	16	6	3
3	23	16	77	4
4	31	17	40	5

eps and min_samples are a range of values, and we define variables to hold the most appropriate values:

```
[66] # Define a range of epsilon (eps) and min_samples values to explore
     eps_values = np.arange(8, 12.75, 0.25)

     min_samples_values = np.arange(3, 10)
     best_eps = None
     best_min_samples = None

     best_silhouette_score = -1
```

```
# Loop through different combinations of eps and min_samples
for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(NumCl)

        # Check if there's more than one cluster (ignoring outliers)
        if len(set(labels)) > 1:
            silhouette_avg = silhouette_score(NumCl, labels)
            if silhouette_avg > best_silhouette_score:
                best_silhouette_score = silhouette_avg
                best_eps = eps
                best_min_samples = min_samples
```

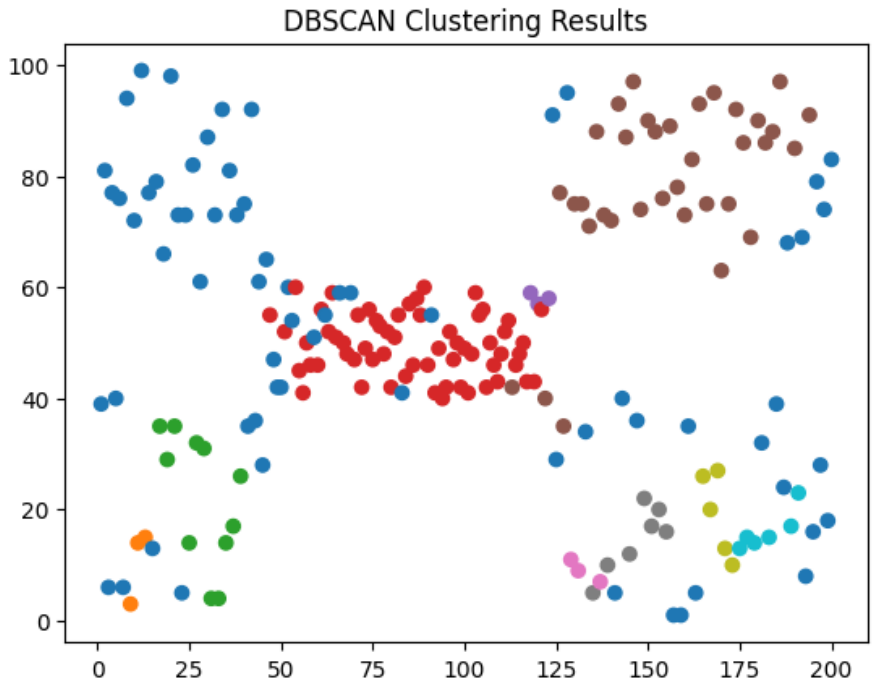
In this section, it is checked whether more than one cluster is formed for this combination. If there is more than one cluster, the silhouette score is calculated and if this score is greater than the previous best score, the epsilon and min_samples values are updated. Finally, the best values for epsilon and min_samples are printed along with the silhouette score:

```
# Print the best parameters and silhouette score
print(f"Best epsilon (eps): {best_eps}")
print(f"Best min_samples: {best_min_samples}")
print(f"Best silhouette score: {best_silhouette_score}")
```

```
Best epsilon (eps): 12.5
Best min_samples: 3
Best silhouette score: 0.05770977889630838
```

Next, we graphically display the clustering results obtained from the DBSCAN algorithm. By running this code, a scatter plot is created where different points are shown in different colors, each color representing a different cluster identified by the DBSCAN algorithm.

```
plt.scatter(customers.iloc[:, 0], customers.iloc[:, 4], c=labels, cmap="tab10")
plt.title("DBSCAN Clustering Results")
plt.show()
```



Comparison between clustering methods:

One way to compare and select the best clustering algorithm for a dataset is to calculate the Silhouette Score. Here we have examined the Silhouette Score value for two methods, hierarchical and density:

```
n_clusters = 5  # number of clusters
linkage_type = 'ward'  # link type in hierarchy

hierarchical = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_type)
labels = hierarchical.fit_predict(NumCl)

silhouette_avg = silhouette_score(NumCl, labels)
print(f"Silhouette Score for Hierarchical Clustering: {silhouette_avg}")
```

```
Silhouette Score for Hierarchical Clustering: 0.3952608995490742
```

```
# Print the best parameters and silhouette score
print(f"Best epsilon (eps): {best_eps}")
print(f"Best min_samples: {best_min_samples}")
print(f"Best silhouette score: {best_silhouette_score}")
```

```
Best epsilon (eps): 12.5
Best min_samples: 3
Best silhouette score: 0.05770977889630838
```

According to the evaluation of the Silhouette Score criterion, a value close to 1 for the hierarchical method indicates that it has better clustering than the density method (such as DBSCAN). A higher Silhouette Score value for the hierarchical method indicates better separation and better clustering quality. Therefore, according to the values presented, it seems that the hierarchical method performs better in this case and provides better clustering than the density method (DBSCAN).

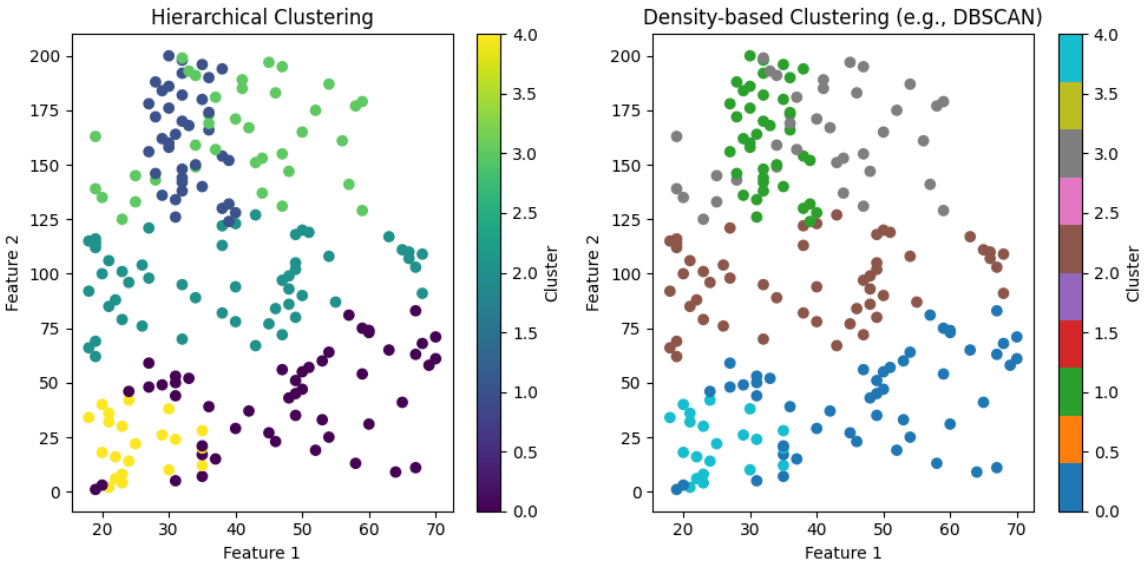
Another way to compare is to use Scatter Plots. Here we have plotted scatter plots for both the hierarchical and density methods and compared them:

```
# داده‌های خوشه‌بندی شده توسط الگوریتم سلسله مراتبی
# NumCl: داده‌های عددی برای نمایش در scatter plot
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(NumCl.iloc[:, 0], NumCl.iloc[:, 1], c=H_labels, cmap='viridis')
plt.title('Hierarchical Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster')

# داده‌های خوشه‌بندی شده توسط الگوریتم چگالی (مانند DBSCAN)
# labels: برچسب‌های نقاط بر اساس خوشه‌بندی چگالی
plt.subplot(1, 2, 2)
plt.scatter(NumCl.iloc[:, 0], NumCl.iloc[:, 1], c=labels, cmap='tab10')
plt.title('Density-based Clustering (e.g., DBSCAN)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster')

plt.tight_layout()  # برای جلوگیری از همپوشانی نمودارها
plt.show()
```

And the result looks like this:



Classification using decision trees and Bayesian

First, we perform classification with a decision tree and then Bayesian; finally, we examine the results of both.

```
X = customers.drop(columns=['Genre'])
y = customers['Genre']
```

```
[14] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

To start, we remove the gender column from the Mall Customers dataframe as an input and consider it as the target variable that we want to predict using other features and assign it to y. Next, we split the data into two sets: training and testing.

```
clf = DecisionTreeClassifier(max_depth=55)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=55)
```

Then, we build a decision tree model and train it using the training data. The trained model can be used to predict the values of the target variable for new data.

```
y_pred = clf.predict(X_test)
y_pred
```

```
array(['Female', 'Female', 'Female', 'Male', 'Male', 'Female', 'Female',
       'Male', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female',
       'Female', 'Female', 'Female', 'Male', 'Male', 'Male', 'Female',
       'Male', 'Female', 'Female', 'Male', 'Male', 'Female', 'Male',
       'Female', 'Male', 'Male', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male'], dtype=object)
```

Using this code, we use the trained decision tree model to predict the values of the target variable on new data (the test set) and then display these predictions as an array.

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate recall
recall = recall_score(y_test, y_pred, pos_label='Female')

# Calculate precision for 'Female' class
precision_female = precision_score(y_test, y_pred, pos_label='Female')
print(f"Precision for 'Female' class: {precision_female}")

# Calculate precision for 'Male' class
precision_male = precision_score(y_test, y_pred, pos_label='Male')
print(f"Precision for 'Male' class: {precision_male}")


print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
```

```
Precision for 'Female' class: 0.5
Precision for 'Male' class: 0.4375
Accuracy: 0.475
Recall: 0.5714285714285714
```

With this piece of code, we can calculate the accuracy, recall, and precision of the model. Precision indicates what proportion of the model's predictions match the actual values of the target variable in the test set.

Prediction Accuracy for Class Female = 0.5:

This number shows that out of all the samples that our model predicted as 'Female', only 50% of them actually belonged to this class.

Prediction Accuracy for Class Male = 0.437:

This value shows that out of all the samples that our model predicted as 'Male', 43.75% actually belonged to this class.

Overall Accuracy = 0.475:

This number shows that out of all the samples, 47.5% were correctly identified.

Recall for all classes: 0.5714285714285714

This result shows that out of all the samples that actually belonged to a class, only 57.14% of them were correctly identified by the model.

Based on these results, we can say that our model does not have good accuracy in classifying between the classes 'Female' and 'Male' and the overall accuracy is also low. Since the prediction accuracy for each class is low, the model probably needs further improvement to help us identify the classes more accurately, and of course, the fact that the appropriate dataset was not chosen for this task is also not without effect.

Bayesian implementation:

This code creates a new object of the GaussianNB class that represents a Gaussian Naive Bayes model. This model is suitable for classifying data that has continuous numerical features.

```
# Initialize a Gaussian Naive Bayes classifier
newClf = GaussianNB()

# Train the classifier using the training data
newClf.fit(X_train, y_train)
```

```
GaussianNB
GaussianNB()
```

Next, using this code, we use the trained model (newClf) to predict the class values for the samples in the test set (X_test). The prediction results are stored in the y_pred_NB array. After this step, we measure the accuracy of this model and find that it is almost equal to the accuracy of the decision tree and the classification results of the previous method apply here as well.

```
# Use the trained classifier to make predictions on the test set
y_pred_NB = newClf.predict(X_test)
y_pred_NB
```

```
array(['Female', 'Female', 'Male', 'Male', 'Male', 'Female', 'Female',
       'Male', 'Male', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Male',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female'],
      dtype='<U6')
```

```
[24] # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.47
```