Objective:

The objective of this project is to develop a lung cancer prediction model using a multivariate lung cancer dataset. The primary goals include performing data cleaning to ensure data integrity, handling missing and applying feature selection and elimination techniques to reduce the dimensionality of the dataset. The project aims to evaluate the performance of multiple classification models, including KNN (K-Nearest Neighbours), Naive Bayes, Random Forest, and Decision Tree, in predicting lung cancer. The accuracy of each model will be assessed, and the ROC curves will be plotted to analyse the trade-off between true positive rate and false positive rate. The ultimate objective is to identify the most accurate classification model for lung cancer prediction and provide insights into the relationship between the dataset attributes and the occurrence of lung cancer.

Python code for Implement the given task:

Import necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

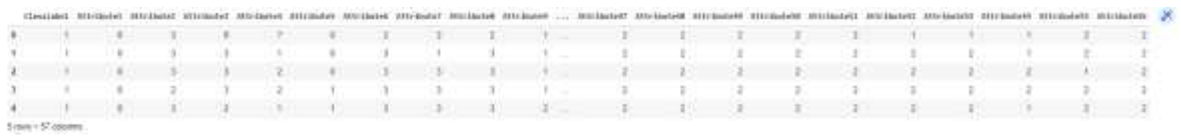
Load the data

```
df=pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-
databases/lung-cancer/lung-cancer.data",header=None)
```

Prework on data

```
# Assign attribute names
attribute_names = ['ClassLabel'] + [f'Attribute{i}' for i in range(1,
57)]  # Assuming 'ClassLabel' as the class attribute name

# Set the column names
df.columns = attribute_names

# Print the updated dataset
df.head()
```

output:

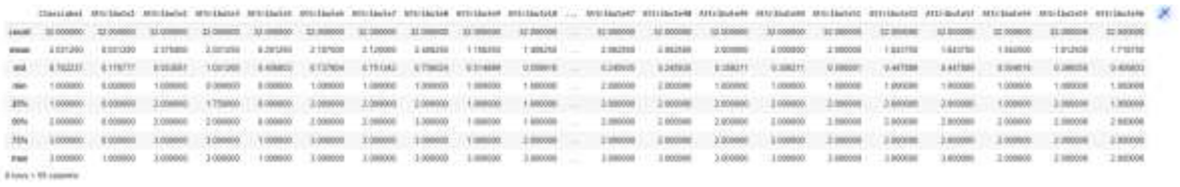


```
   ClassLabel Attribute1 Attribute2 Attribute3 Attribute4 Attribute5 Attribute6 Attribute7 Attribute8 Attribute9 ... Attribute47 Attribute48 Attribute49 Attribute50 Attribute51 Attribute52 Attribute53 Attribute54 Attribute55 Attribute56
0           1          0          3          0          ?          0          2          2          2          1 ...           2           2           2           2           1           1           1           1           2           2
1           1          0          3          3          1          0          3          1          3          1 ...           2           2           2           2           2           2           2           1           2           2
2           1          0          3          3          2          0          3          3          3          1 ...           2           2           2           2           2           2           2           2           1           2
3           1          0          2          3          2          1          3          3          3          1 ...           2           2           2           2           2           2           2           2           2           2
4           1          0          3          2          1          1          3          3          3          2 ...           2           2           2           2           2           2           2           1           2           2
5 rows × 57 columns
```

Describe the data

```
df.describe()
```

output:



```
       ClassLabel Attribute1 Attribute2 Attribute3 Attribute5 Attribute6 Attribute7 Attribute8 Attribute9 Attribute10 ... [illegible]
count  32.000000  32.000000  32.000000  32.000000  32.000000  32.000000  32.000000  32.000000  32.000000  32.000000  ... [illegible]
mean   2.031250   0.031250   2.375000   2.031250   0.281250   2.187500   2.125000   2.406250   1.156250   1.906250   ... [illegible]
std    0.782237   0.176777   [illegible] 1.031265  [illegible] 0.737804  0.751343  [illegible] [illegible] [illegible] ... [illegible]
min    1.000000   0.000000   1.000000   0.000000   0.000000   1.000000   1.000000   1.000000   1.000000   1.000000   ... [illegible]
25%    1.000000   0.000000   2.000000   1.750000   0.000000   2.000000   2.000000   2.000000   1.000000   1.000000   ... [illegible]
50%    2.000000   0.000000   3.000000   2.000000   0.000000   2.000000   2.000000   3.000000   1.000000   1.000000   ... [illegible]
75%    3.000000   0.000000   3.000000   3.000000   1.000000   3.000000   3.000000   3.000000   1.000000   2.000000   ... [illegible]
max    3.000000   1.000000   3.000000   3.000000   1.000000   3.000000   3.000000   3.000000   3.000000   3.000000   ... [illegible]
8 rows × 55 columns
```

Step 1: Data Cleaning

Replace the missing value “?” to “NA”

```
# Replace missing values
df = df.replace('?', pd.NA)
```

Fill the missing the value with mode in Attribute 4

```
# Convert "Attribute4" column to numeric
df["Attribute4"] = pd.to_numeric(df["Attribute4"], errors="coerce")

# Calculate the mode of the "Attribute4" column
attribute4_mode = df["Attribute4"].mode()[0]

# Fill the missing values in "Attribute4" with the mode
df["Attribute4"] = df["Attribute4"].fillna(attribute4_mode)
```

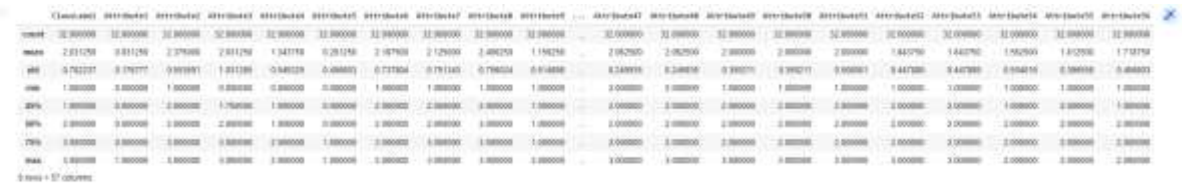
Fill the missing the value with mode in Attribute 38

```
# Convert "Attribute4" column to numeric
df["Attribute38"] = pd.to_numeric(df["Attribute38"], errors="coerce")

# Calculate the mode of the "Attribute4" column
attribute4_mode = df["Attribute38"].mode()[0]

# Fill the missing values in "Attribute4" with the mode
df["Attribute38"] = df["Attribute38"].fillna(attribute4_mode)
```

Describe the data again

```
df.describe()
```

output:



	Class Label	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7	Attribute8	Attribute9	...	Attribute47	Attribute48	Attribute49	Attribute50	Attribute51	Attribute52	Attribute53	Attribute54	Attribute55	Attribute56
count	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
mean	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
std	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
min	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
25%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
50%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
75%	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
max	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

8 rows × 57 columns

Covert attribute 4 and 38 into integer

```
df["Attribute4"] = df["Attribute4"].astype(int)
df["Attribute38"] = df["Attribute38"].astype(int)
```

Step 2: Feature Selection & Feature Elimination

Feature selection and feature elimination are two common techniques used in machine learning to reduce the number of features in a dataset and improve model performance. Here are brief explanations of each method:

Feature Selection:

Feature selection aims to select a subset of relevant features from the original dataset. The goal is to identify the most informative features that have the most impact on the target variable while discarding irrelevant or redundant features. There are several approaches for feature selection:

a. Filter Methods: These methods use statistical metrics to rank and select features. Common metrics include correlation, mutual information, chi-square, and variance threshold. Features are evaluated independently of the chosen machine learning algorithm.

b. Wrapper Methods: These methods evaluate different subsets of features by training and testing a machine learning model iteratively. Examples include Recursive Feature Elimination (RFE) and Forward/Backward Stepwise Selection. Wrapper methods are computationally more expensive than filter methods but can consider feature interactions.

c. Embedded Methods: These methods incorporate feature selection within the model training process. Some machine learning algorithms have built-in mechanisms to automatically select features during training. Examples include LASSO (Least Absolute Shrinkage and Selection Operator) and tree-based feature importance.

Feature Elimination:

Feature elimination, also known as dimensionality reduction, aims to reduce the number of features by transforming the original feature space into a lower-dimensional representation. It is particularly useful when dealing with high-dimensional data or when the dataset contains highly correlated features. Two commonly used methods for feature elimination are:

a. Principal Component Analysis (PCA): PCA is a technique that transforms the original features into a new set of uncorrelated variables called principal components. These components are ordered by their variance, with the first few components capturing the majority of the dataset's variability. By selecting a subset of the top principal components, you can effectively reduce the dimensionality of the dataset.

b. Linear Discriminant Analysis (LDA): LDA is a supervised dimensionality reduction method commonly used for classification problems. It aims to find a projection that maximizes the separability between different classes while minimizing the within-class variance. LDA seeks to create a new feature space that maximizes class discrimination.

Both feature selection and feature elimination techniques have their advantages and are suitable for different scenarios. The choice of method depends on factors such as the dataset size, dimensionality, computational resources, and the specific problem you are trying to solve. It's often recommended to experiment with different methods and evaluate their impact on model performance to find the most effective approach for a given task.

I will go with PCA

Import necessary libraries

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
```

```
# Separate the feature matrix (X) and the target variable (y)
X = df.drop("ClassLabel", axis=1)
y = df["ClassLabel"]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=52)
```

Apply PCA

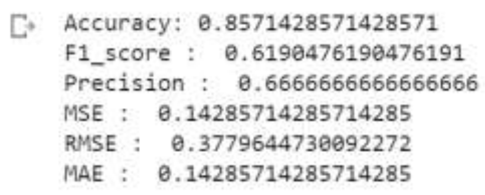
```
# Apply PCA for dimensionality reduction
pca = PCA(n_components=19)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

Now I will work on different Machine learning classification model and find the accuracy.

1. K-Nearest Neighbours (KNN)

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,
classification_report,mean_absolute_error, mean_squared_error,
r2_score,f1_score,precision_score,recall_score,precision_recall_fscore_
support
from sklearn.preprocessing import StandardScaler

# Create a KNN classifier
knn = KNeighborsClassifier()

# Train the model
knn.fit(X_train_pca, y_train)

# Make predictions on the testing set
y_pred_KNN = knn.predict(X_test_pca)

# Evaluate the model's performance
accuracy_KNN = accuracy_score(y_test, y_pred_KNN)
print("Accuracy:", accuracy_KNN)

f1_score_KNN=f1_score(y_test,y_pred_KNN,average='macro')
print("F1_score : ",f1_score_KNN)

precision_KNN=precision_score(y_test,y_pred_KNN,average='macro')
print("Precision : ",precision_KNN)

mse_KNN=mean_squared_error(y_test,y_pred_KNN)
print("MSE : ",mse_KNN)

rmse_KNN=np.sqrt(mse_KNN)
print("RMSE : ",rmse_KNN)

mae_KNN=mean_absolute_error(y_test,y_pred_KNN)
print("MAE : ",mae_KNN)
```

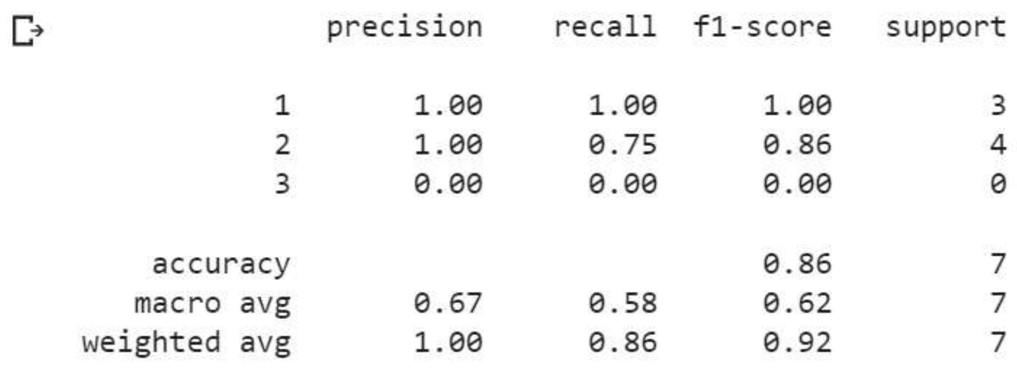
output:



```
Accuracy: 0.8571428571428571
F1_score :  0.6190476190476191
Precision :  0.6666666666666666
MSE :  0.14285714285714285
RMSE :  0.3779644730092272
MAE :  0.14285714285714285
```

Report the model

```
report = classification_report(y_test, y_pred_KNN)
print(report)
```
output:



```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         3
           2       1.00      0.75      0.86         4
           3       0.00      0.00      0.00         0

    accuracy                           0.86         7
   macro avg       0.67      0.58      0.62         7
weighted avg       1.00      0.86      0.92         7
```

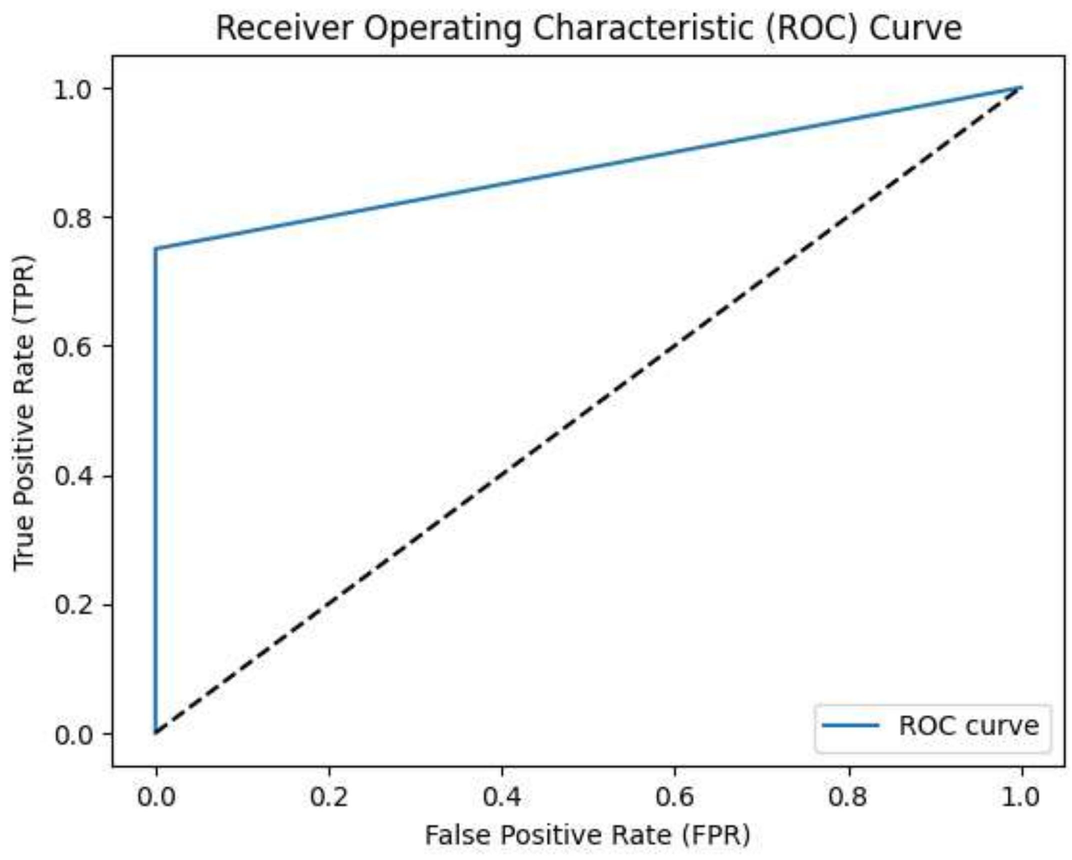
Import libraries

```
from sklearn import metrics
```

Draw the Receiver Operating Characteristic (ROC) Curve

```
# Binarize the true labels
y_test_bin = np.where(y_test == 2, 1, 0)

# Compute the probabilities for positive class
y_scores = np.where(y_pred_KNN == 2, 1, 0)

# Calculate the False Positive Rate (FPR) and True Positive Rate (TPR)
fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

# Calculate the Area Under the ROC Curve (AUC)
roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

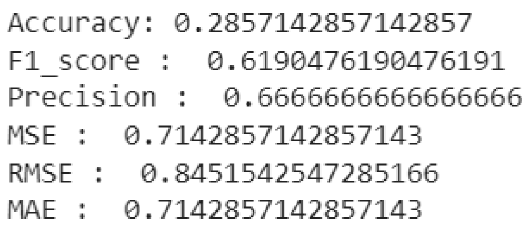
output:


Receiver Operating Characteristic (ROC) Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (TPR)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (FPR)
ROC curve

2. Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,
classification_report,mean_absolute_error, mean_squared_error,
r2_score,f1_score,precision_score,recall_score,precision_recall_fscore_
support


# Create a Naive Bayes classifier
nb = GaussianNB()

# Train the model
nb.fit(X_train_pca, y_train)

# Make predictions on the testing set
y_pred _NaiveBayes = nb.predict(X_test_pca)

# Evaluate the model's performance
accuracy_NaiveBayes = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_NaiveBayes)

f1_score_NaiveBayes=f1_score(y_test,y_pred _NaiveBayes,average='macro')

print("F1_score : ",f1_score_KNN)

precision_NaiveBayes=precision_score(y_test,y_pred_NaiveBayes

,average='macro')
print("Precision : ",precision_KNN)

mse_NaiveBayes=mean_squared_error(y_test,y_pred_NaiveBayes)
print("MSE : ",mse_NaiveBayes)

rmse_NaiveBayes=np.sqrt(mse_NaiveBayes)
print("RMSE : ",rmse_NaiveBayes)

mae_NaiveBayes=mean_absolute_error(y_test,y_pred_NaiveBayes)
print("MAE : ",mae_NaiveBayes)
```

output:

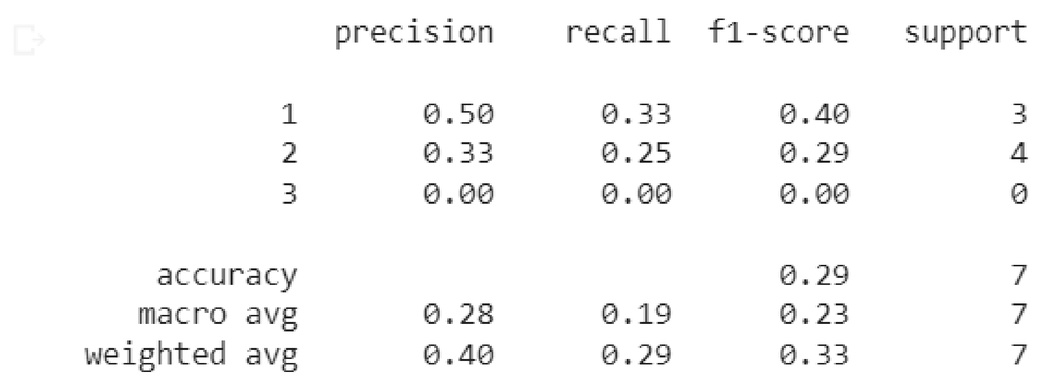


```
Accuracy: 0.2857142857142857
F1_score :  0.6190476190476191
Precision :  0.6666666666666666
MSE :  0.7142857142857143
RMSE :  0.8451542547285166
MAE :  0.7142857142857143
```

Report the model

```
report = classification_report(y_test, y_pred_NaiveBayes)
print(report)
```

output:

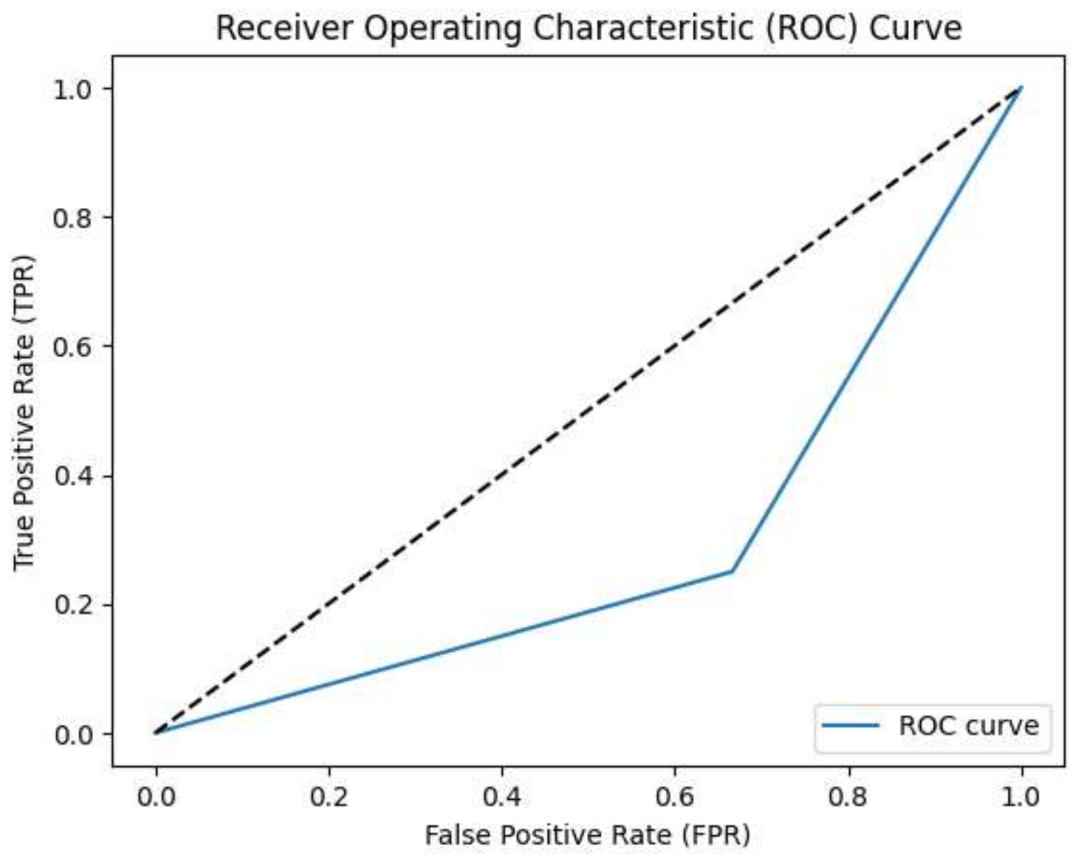


```
              precision    recall  f1-score   support

           1       0.50      0.33      0.40         3
           2       0.33      0.25      0.29         4
           3       0.00      0.00      0.00         0

    accuracy                           0.29         7
   macro avg       0.28      0.19      0.23         7
weighted avg       0.40      0.29      0.33         7
```

Draw the Receiver Operating Characteristic (ROC) Curve

```
# Binarize the true labels
y_test_bin = np.where(y_test == 2, 1, 0)

# Compute the probabilities for positive class
y_scores = np.where(y_pred_NaiveBayes == 2, 1, 0)

# Calculate the False Positive Rate (FPR) and True Positive Rate (TPR)
fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

# Calculate the Area Under the ROC Curve (AUC)
roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

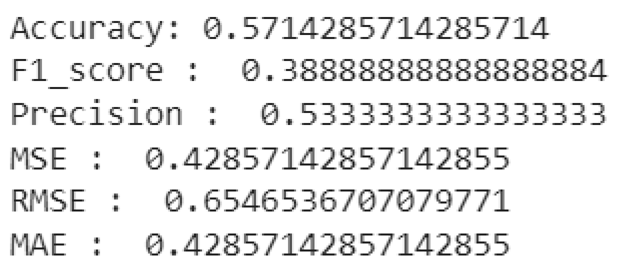
output:


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC curve
0.0
0.2
0.4
0.6
0.8
1.0

3. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,
classification_report,mean_absolute_error, mean_squared_error,
r2_score,f1_score,precision_score,recall_score,precision_recall_fscore_
support


# Create a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=52)

# Train the model
rf.fit(X_train_pca, y_train)

# Make predictions on the testing set
y_pred _RF = rf.predict(X_test_pca)


# Evaluate the model's performance
accuracy_RF = accuracy_score(y_test, y_pred_RF)
print("Accuracy:", accuracy_RF)

f1_score_RF=f1_score(y_test,y_pred_RF,average='macro')
print("F1_score : ",f1_score_RF)

precision_RF=precision_score(y_test,y_pred_RF,average='macro')
print("Precision : ",precision_RF)

mse_RF=mean_squared_error(y_test,y_pred_RF)
print("MSE : ",mse_RF)

rmse_RF=np.sqrt(mse_RF)
print("RMSE : ",rmse_RF)

mae_RF=mean_absolute_error(y_test,y_pred_RF)
print("MAE : ",mae_RF)
```

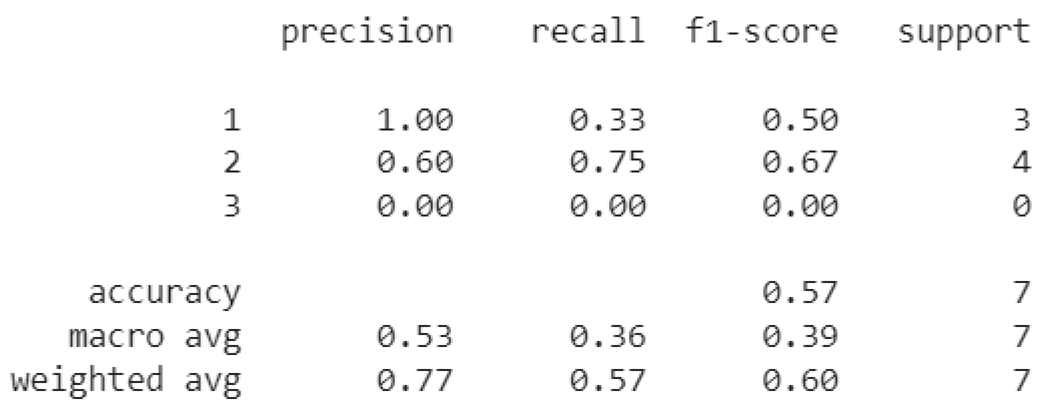
output:



```
Accuracy: 0.5714285714285714
F1_score :  0.38888888888888884
Precision :  0.5333333333333333
MSE :  0.42857142857142855
RMSE :  0.6546536707079771
MAE :  0.42857142857142855
```

Report the model

```
report = classification_report(y_test, y_pred_RF)
print(report)
```

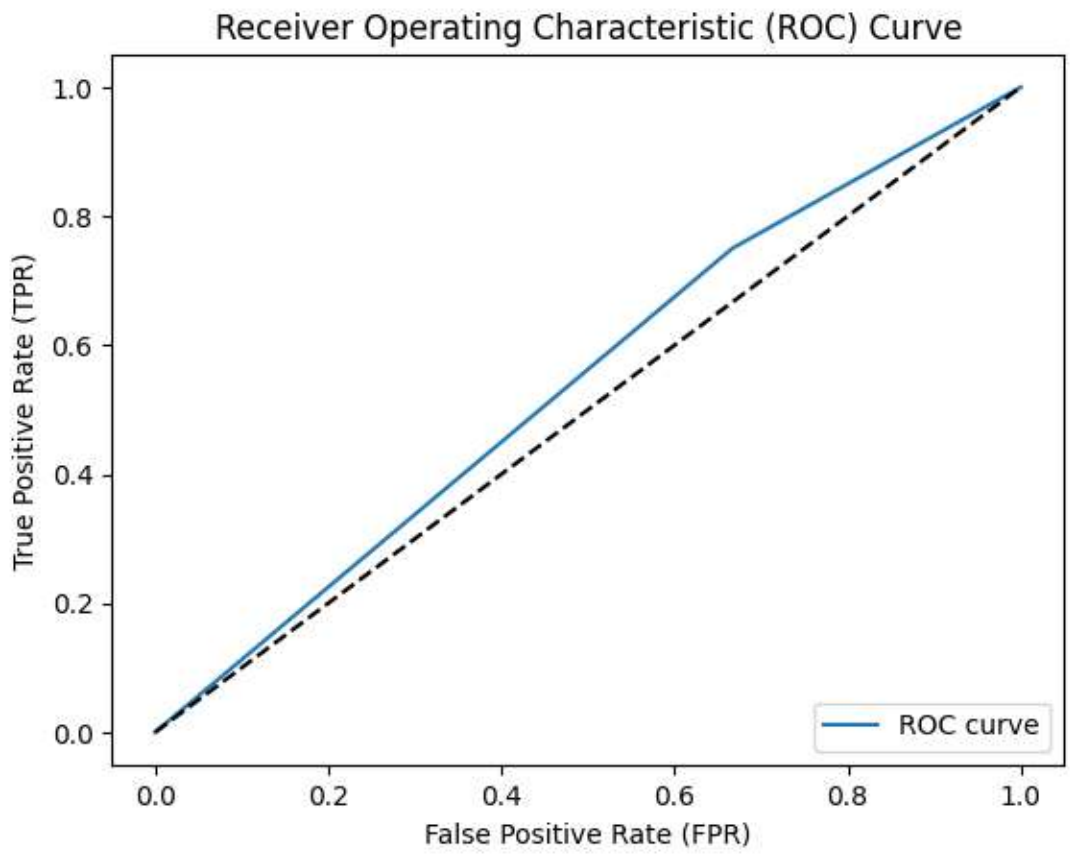


```
              precision    recall  f1-score   support

           1       1.00      0.33      0.50         3
           2       0.60      0.75      0.67         4
           3       0.00      0.00      0.00         0

    accuracy                           0.57         7
   macro avg       0.53      0.36      0.39         7
weighted avg       0.77      0.57      0.60         7
```

Draw the Receiver Operating Characteristic (ROC) Curve

```
# Binarize the true labels
y_test_bin = np.where(y_test == 2, 1, 0)

# Compute the probabilities for positive class
y_scores = np.where(y_pred_RF == 2, 1, 0)

# Calculate the False Positive Rate (FPR) and True Positive Rate (TPR)
fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

# Calculate the Area Under the ROC Curve (AUC)
roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

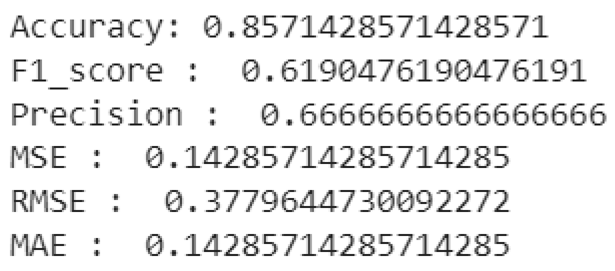
output:


Receiver Operating Characteristic (ROC) Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (TPR)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (FPR)
ROC curve

4. Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report,mean_absolute_error, mean_squared_error,
r2_score,f1_score,precision_score,recall_score,precision_recall_fscore_
support


# Create a Decision Tree classifier
dt = DecisionTreeClassifier(random_state=52)

# Train the model
dt.fit(X_train_pca, y_train)

# Make predictions on the testing set
y_pred_DT = dt.predict(X_test_pca)

# Evaluate the model's performance
accuracy_DT = accuracy_score(y_test, y_pred_DT)
print("Accuracy:", accuracy_DT)

f1_score_DT=f1_score(y_test,y_pred_DT,average='macro')
print("F1_score : ",f1_score_DT)

precision_DT=precision_score(y_test,y_pred_DT,average='macro')
print("Precision : ",precision_DT)

mse_DT=mean_squared_error(y_test,y_pred_DT)
print("MSE : ",mse_DT)

rmse_DT=np.sqrt(mse_DT)
print("RMSE : ",rmse_DT)

mae_DT=mean_absolute_error(y_test,y_pred_DT)
print("MAE : ",mae_DT)
```

output:



```
Accuracy: 0.8571428571428571
F1_score :  0.6190476190476191
Precision :  0.6666666666666666
MSE :  0.14285714285714285
RMSE :  0.3779644730092272
MAE :  0.14285714285714285
```

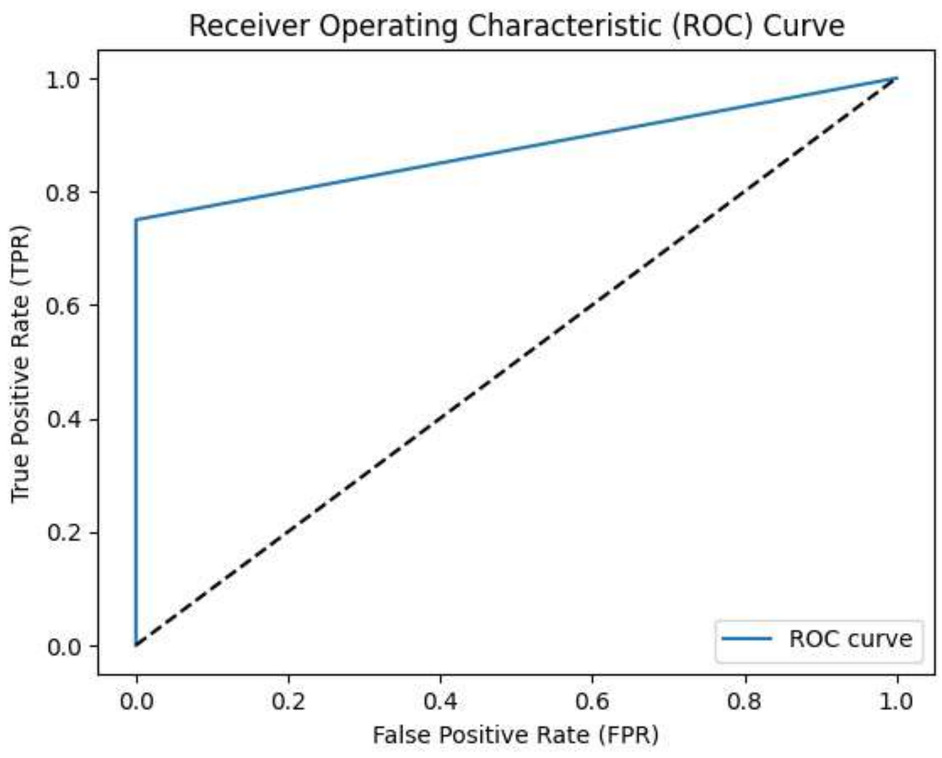
Report the model

```
report = classification_report(y_test, y_pred_DT)
print(report)
```

Draw the Receiver Operating Characteristic (ROC) Curve

```
# Binarize the true labels
y_test_bin = np.where(y_test == 2, 1, 0)

# Compute the probabilities for positive class
y_scores = np.where(y_pred_DT == 2, 1, 0)

# Calculate the False Positive Rate (FPR) and True Positive Rate (TPR)
fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

# Calculate the Area Under the ROC Curve (AUC)
roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

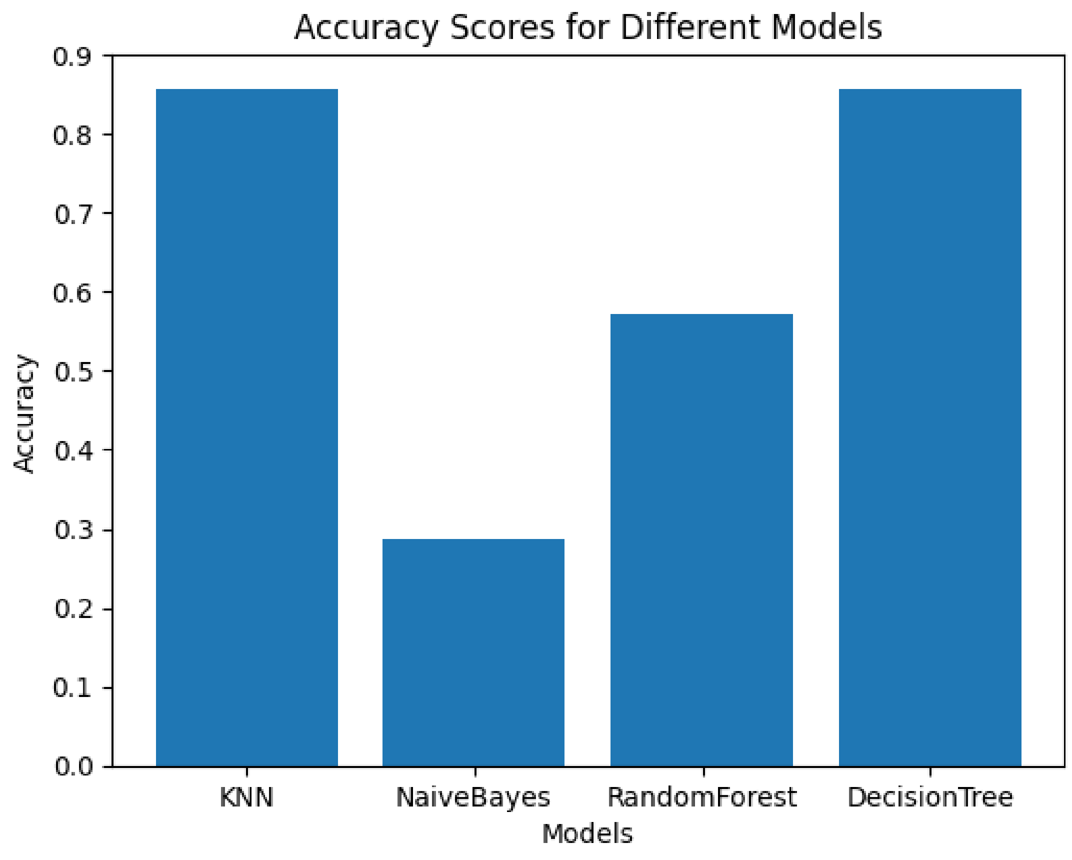
output:



Visualize the Accuracy of models

```
model_names = ['KNN', 'NaiveBayes', 'RandomForest','DecisionTree']
accuracy_scores =
[accuracy_KNN,accuracy_NaiveBayes,accuracy_RF,accuracy_DT]

# Set the x-axis positions for the bars
x = range(len(model_names))

# Create a bar plot
plt.bar(x, accuracy_scores)

# Add x-axis labels, y-axis label, and title
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Scores for Different Models')

# Add model names as x-axis tick labels
plt.xticks(x, model_names)

# Display the plot
plt.show()
```

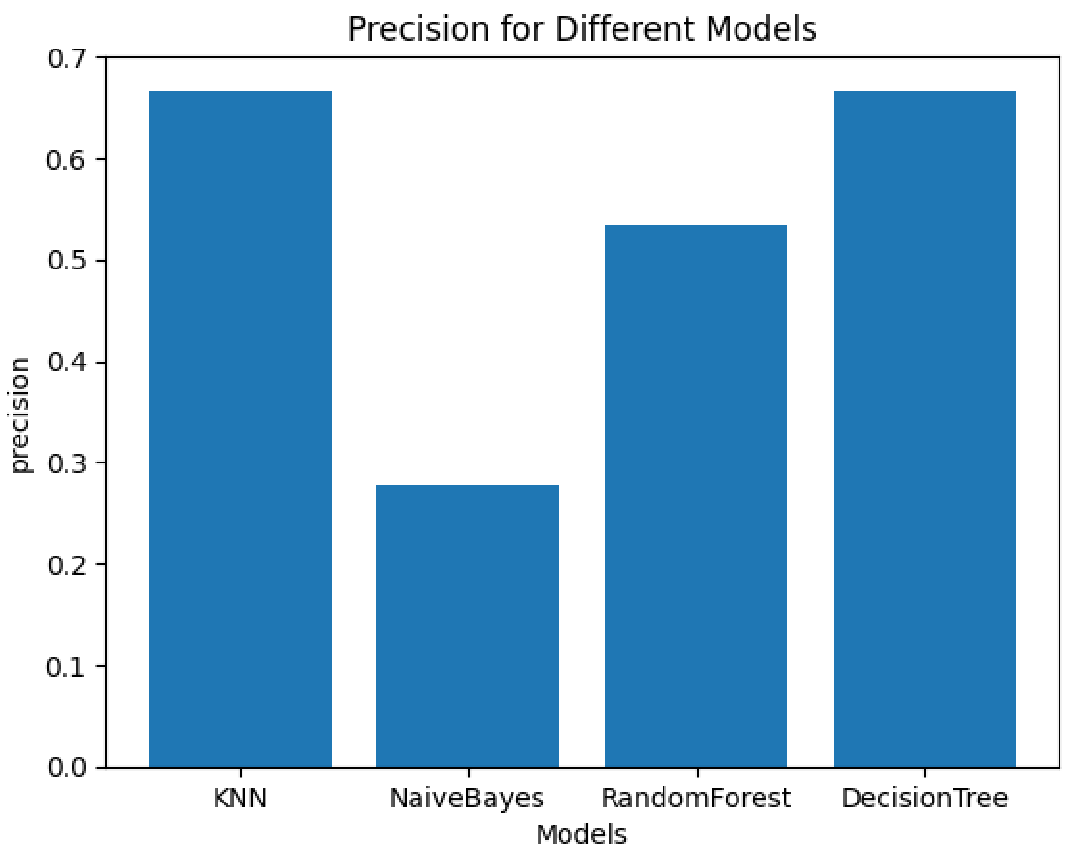
Output:



Visualize the Precision of models

```
model_names = ['KNN', 'NaiveBayes', 'RandomForest','DecisionTree']
accuracy_scores =
[precision_KNN,precision_NaiveBayes,precision_RF,precision_DT]

# Set the x-axis positions for the bars
x = range(len(model_names))

# Create a bar plot
plt.bar(x, accuracy_scores)

# Add x-axis labels, y-axis label, and title
plt.xlabel('Models')
plt.ylabel('precision')
plt.title('Precision for Different Models')

# Add model names as x-axis tick labels
plt.xticks(x, model_names)

# Display the plot
plt.show()
```

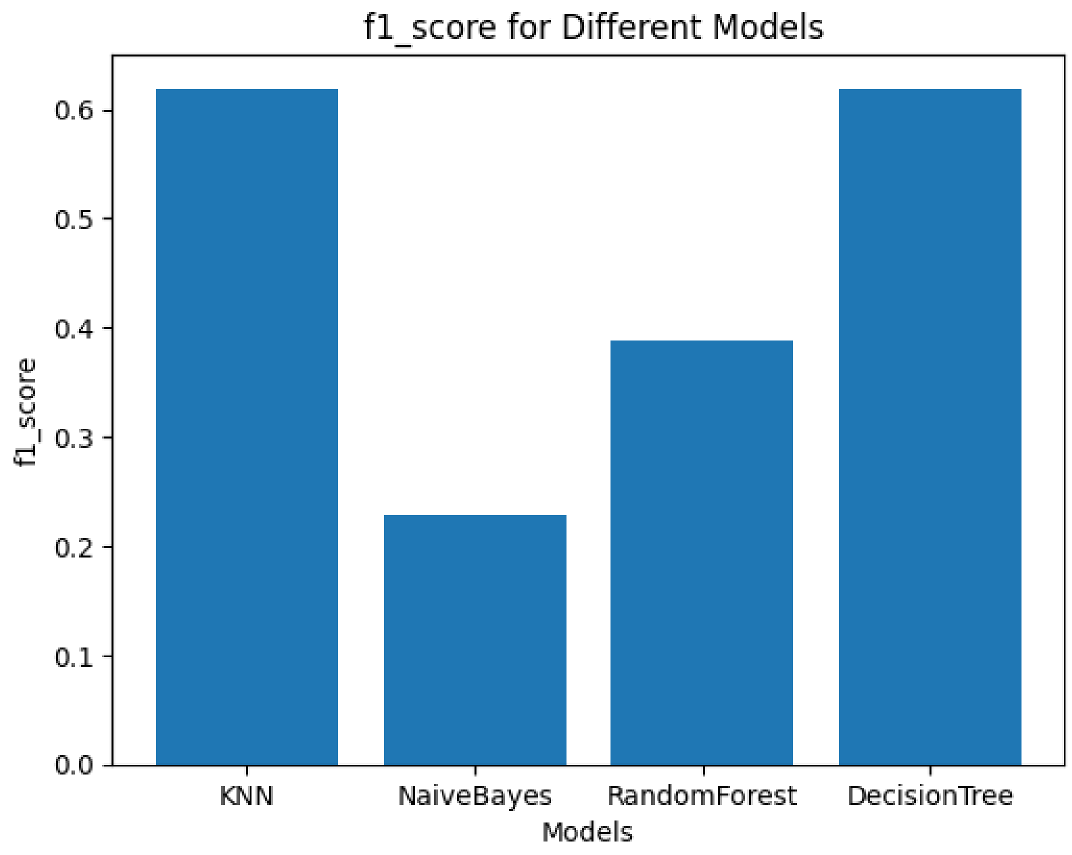
Output:



Visualize the F1_Score of models

```
model_names = ['KNN', 'NaiveBayes', 'RandomForest','DecisionTree']
accuracy_scores =
[f1_score_KNN,f1_score_NaiveBayes,f1_score_RF,f1_score_DT]

# Set the x-axis positions for the bars
x = range(len(model_names))

# Create a bar plot
plt.bar(x, accuracy_scores)

# Add x-axis labels, y-axis label, and title
plt.xlabel('Models')
plt.ylabel('f1_score')
plt.title(' f1_score for Different Models')

# Add model names as x-axis tick labels
plt.xticks(x, model_names)

# Display the plot
plt.show()
```

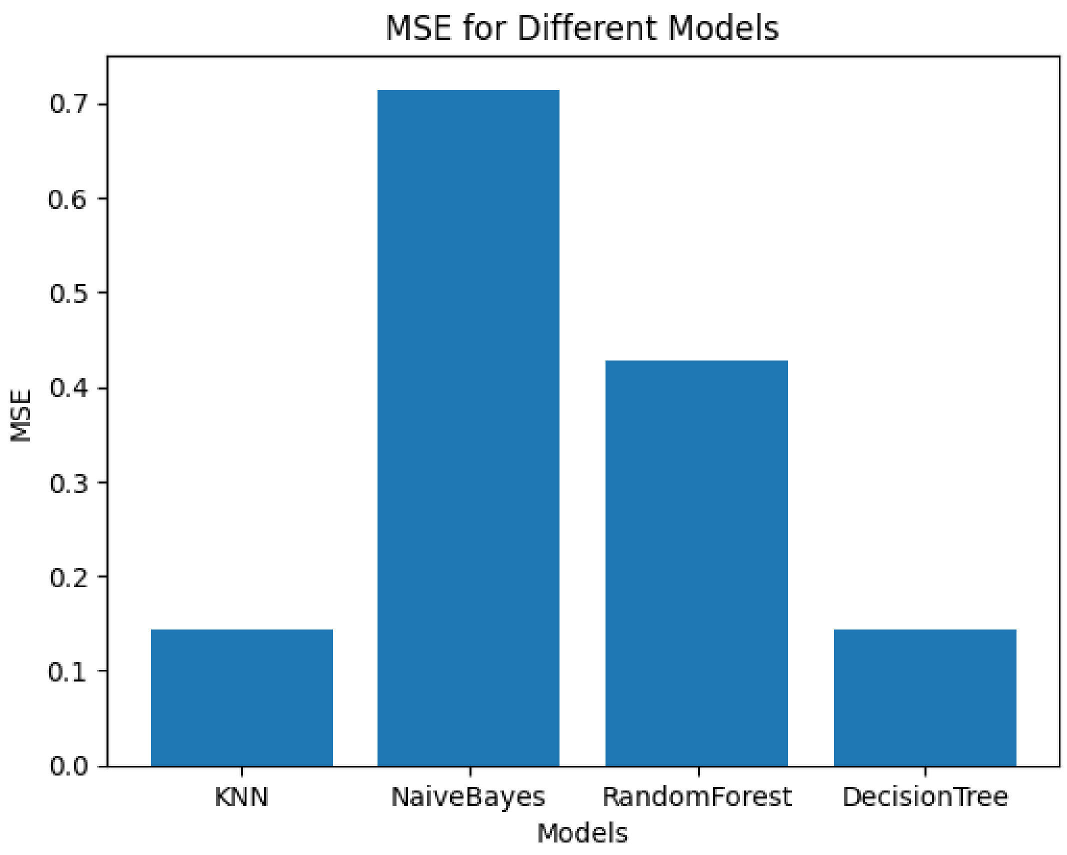
Output:



Visualize the MSE of models

```
model_names = ['KNN', 'NaiveBayes', 'RandomForest','DecisionTree']
accuracy_scores = [mse_KNN,mse_NaiveBayes,mse_RF,mse_DT]

# Set the x-axis positions for the bars
x = range(len(model_names))

# Create a bar plot
plt.bar(x, accuracy_scores)

# Add x-axis labels, y-axis label, and title
plt.xlabel('Models')
plt.ylabel('MSE')
plt.title('MSE for Different Models')

# Add model names as x-axis tick labels
plt.xticks(x, model_names)

# Display the plot
plt.show()
```

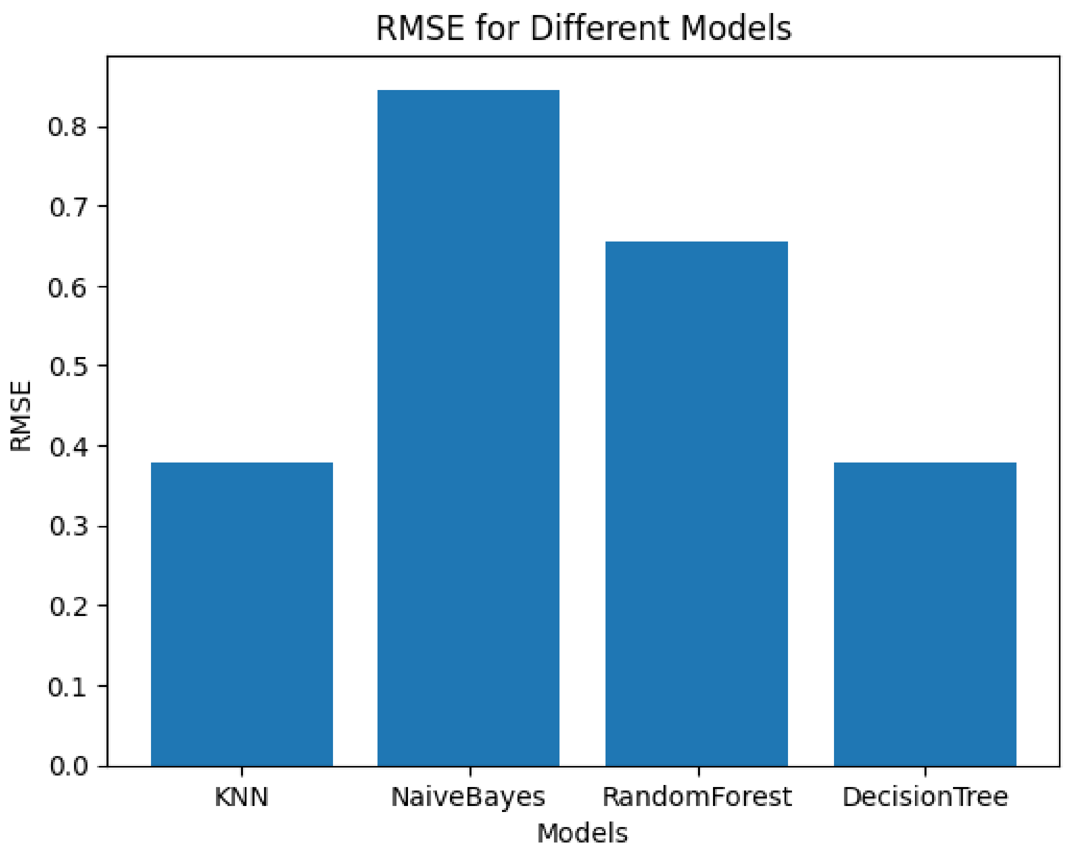
Output:



Visualize the RMSE of models

```
model_names = ['KNN', 'NaiveBayes', 'RandomForest','DecisionTree']
accuracy_scores = [rmse_KNN,rmse_NaiveBayes,rmse_RF,rmse_DT]

# Set the x-axis positions for the bars
x = range(len(model_names))

# Create a bar plot
plt.bar(x, accuracy_scores)

# Add x-axis labels, y-axis label, and title
plt.xlabel('Models')
plt.ylabel('RMSE')
plt.title('RMSE for Different Models')

# Add model names as x-axis tick labels
plt.xticks(x, model_names)

# Display the plot
plt.show()
```

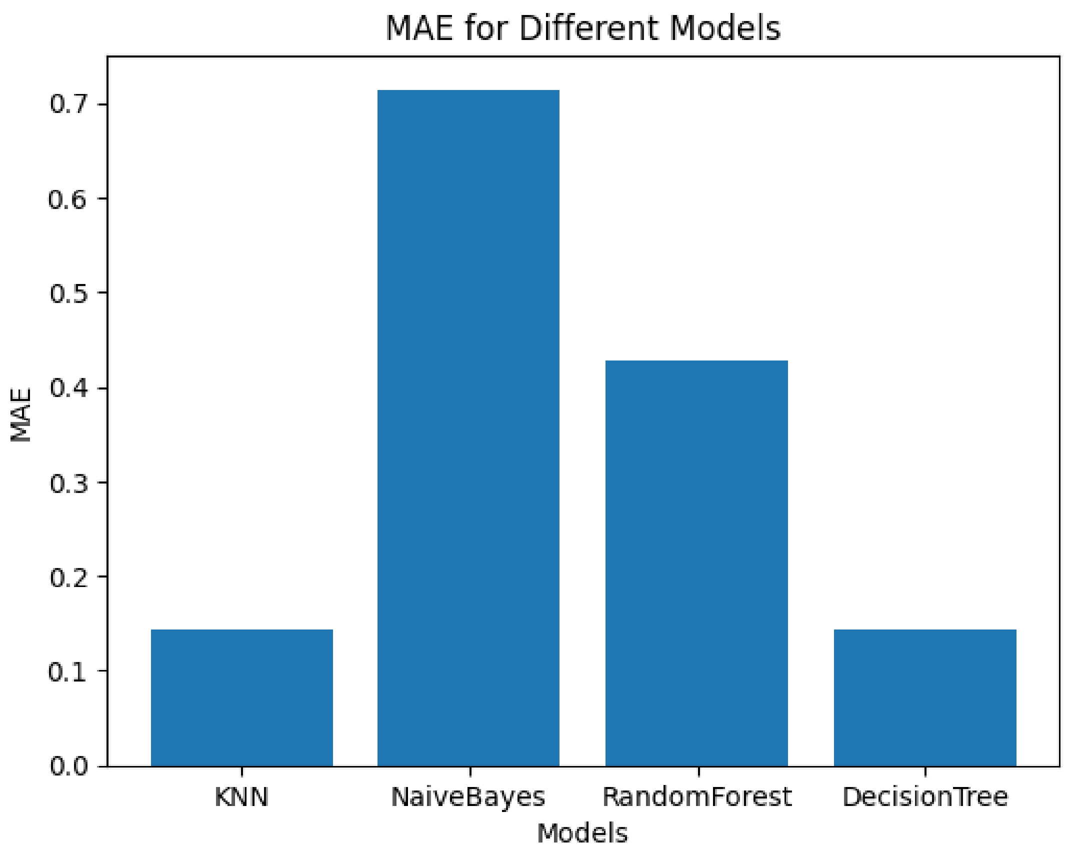
Output:



Visualize the MAE of models

```
model_names = ['KNN', 'NaiveBayes', 'RandomForest','DecisionTree']
accuracy_scores = [mae_KNN,mae_NaiveBayes,mae_RF,mae_DT]

# Set the x-axis positions for the bars
x = range(len(model_names))

# Create a bar plot
plt.bar(x, accuracy_scores)

# Add x-axis labels, y-axis label, and title
plt.xlabel('Models')
plt.ylabel('MAE')
plt.title('MAE for Different Models')

# Add model names as x-axis tick labels
plt.xticks(x, model_names)

# Display the plot
plt.show()
```

Output:



Conclusion:

A lung cancer dataset with 57 attributes was utilized for classification purposes. The dataset was thoroughly cleaned, and missing values were imputed using the mode of each column. As the dataset contained only 32 rows, no rows were deleted to preserve the available data.

Next, feature selection and elimination were performed using the Principal Component Analysis (PCA) method. PCA transformed the high-dimensional dataset into a lower-dimensional space, capturing the most informative features while reducing dimensionality.

Four classification models, namely KNN (K-Nearest Neighbors), Naive Bayes, Random Forest, and Decision Tree, were employed to predict lung cancer. The accuracy scores were calculated for each model, and it was observed that KNN and Decision Tree achieved the highest accuracy, followed by Random Forest and Naive Bayes.

To evaluate the models further, ROC curves were drawn, which provide insights into the trade-off between true positive rate and false positive rate. The ROC curves allowed for a visual comparison of the performance of each classifier.

In conclusion, this Python code successfully processed the lung cancer dataset, applied feature selection using PCA, and employed various classification models for predicting lung cancer. The results indicated that KNN and Decision Tree performed the best in terms of accuracy, while Random Forest and Naive Bayes yielded slightly lower accuracy scores. The visualization of ROC curves provided additional information on the models' performance. This project serves as a valuable foundation for further analysis and investigation in the field of lung cancer prediction**.**

	KNN	**Naïve Bayes**	**Random Forest**	**Decision Tree**
Accuracy	85.71%	28.57%	57.14%	85.71%
Precision	0.6666	0.666	0.5333	0.6666
F1_Score	0.6190	0.619	0.3888	0.6190
MSE	0.1428	0.7142	0.4285	0.1428
RMSE	0.3779	0.8451	0.6546	0.3779
MAE	0.1428	0.7142	0.4285	0.1428