Data Mining and Warehouse

Experiment 3

Name: Jigar Siddhpura

SAP: 60004210155

Batch: B2

Branch: Computer Engineering

Aim :

Implementation of Classification algorithm Using 1. Decision Tree ID3 and 2. Naïve Bayes algorithm

Theory:

Decision tree ID3:

In simple words, a decision tree is a structure that contains nodes (rectangular boxes) and edges(arrows) and is built from a dataset (table of columns representing features/attributes and rows corresponds to records). Each node is either used to make a decision (known as decision node) or represent an outcome (known as leaf node).ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.

Naive Bayes:

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset. Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for playing golf.

Initialisation :

```
from google.colab import drive
drive.mount('/content/gdrive')

!pip install scikit-plot
import pandas as pd
import numpy as np
import seaborn as sns
import re
import matplotlib.pyplot as plt
from scikitplot.metrics import plot_confusion_matrix
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, classification_report, confusion_matrix
titanic_train = "/content/gdrive/MyDrive/Synapse-Task/synapse_w1/train.csv"
titanic_test = "/content/gdrive/MyDrive/Synapse-Task/synapse_w1/test.csv"
penguin_df = sns.load_dataset("penguins")
iris_df = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv")
spam_df = pd.read_csv("/content/gdrive/MyDrive/DMW/datasets/spam.csv")
wine_df = pd.read_csv("/content/gdrive/MyDrive/DMW/datasets/WineQT.csv")
gaussian = []
decision = []
```

Part A:

Dataset(Titanic) :

```
"""### Titanic"""
titanic_train_df = pd.read_csv(titanic_train)
titanic_test_df = pd.read_csv(titanic_test)
import re
titles = []
for nm in titanic_train_df.Name:
  title_search = re.search('(\w+)\.', nm)
  title = title_search.group(1)
  titles.append(title)
titanic_train_df['Title'] = titles
titanic_train_df.columns
titanic_train_df.drop(['PassengerId', 'Ticket', 'Name'], axis=1, inplace=True)
nullPercent = {}
for i in titanic_train_df:
  null_count_i =  titanic_train_df.isnull().sum()[i]
  per = null_count_i*100/titanic_train_df.shape[0]
  nullPercent[i] = per
for i in nullPercent:
  if(nullPercent[i] > 50) : titanic_train_df.drop([i], axis=1, inplace=True)
titanic_train_df.info()
mean = np.mean(titanic_train_df.Age)
titanic_train_df['Age'].fillna(value=mean,  inplace=True)
train_df = titanic_train_df.assign(Family=lambda x: x.SibSp + x.Parch)
def zscore_norm(x):
 mean = np.mean(x)
 std = np.std(x)
 return (x-mean)/std
train_df = train_df.assign(Age=lambda x: zscore_norm(x.Age))
train_df = train_df.assign(Fare=lambda x: zscore_norm(x.Fare))
train_df = train_df.assign(Family=lambda x: zscore_norm(x.Family))
train_df = pd.get_dummies(train_df , columns=['Pclass', 'Sex', 'Title', 'Embarked'])
train_df
y = train_df.pop("Survived")
```

```
x = train_df
from sklearn.model_selection import train_test_split
x_train , x_valid , y_train , y_valid = train_test_split(x ,y, random_state=10, stratify=y, test_size=0.25 )
y_train.value_counts(normalize=True)
y_valid.value_counts(normalize=True)
```

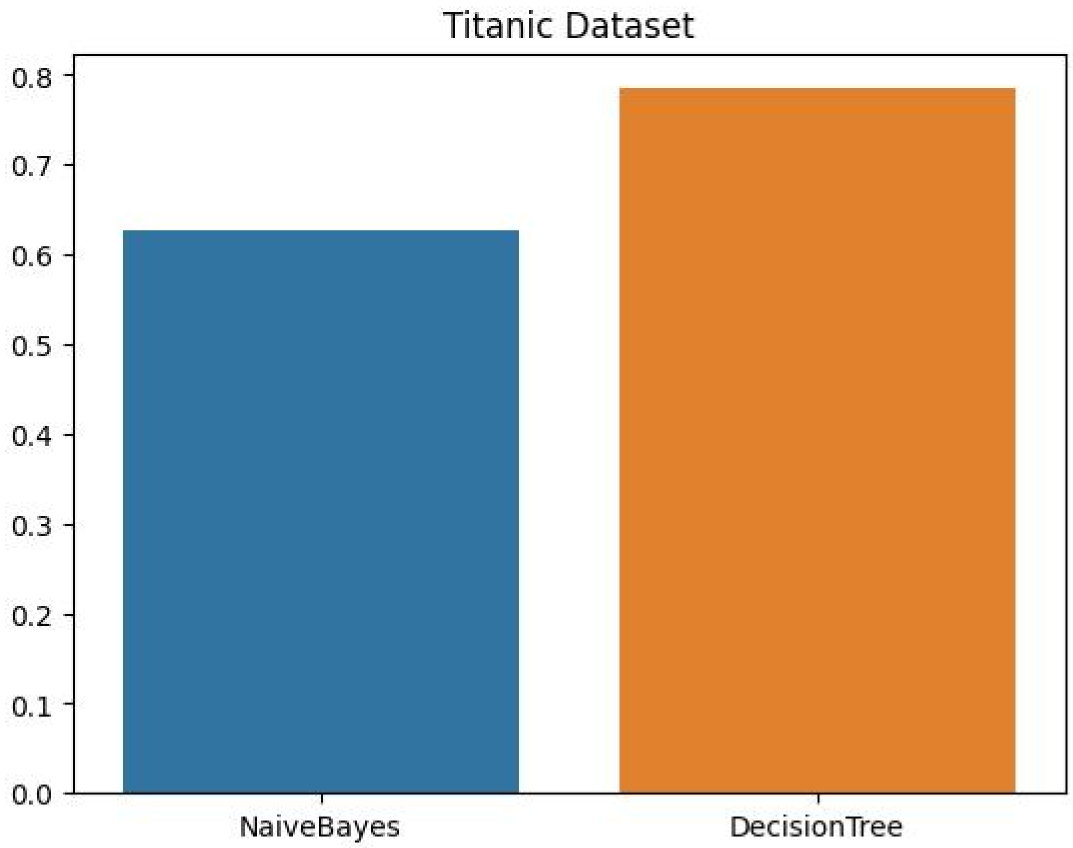
```
# using naive bayes
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(x_train, y_train)
nb_accuracy = nb_model.score(x_valid, y_valid)
print(nb_accuracy)
```

```
# using decision tree
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train, y_train)
dt_accuracy = dt_model.score(x_valid, y_valid)
gaussian.append(nb_accuracy)
decision.append(dt_accuracy)

accuracy = [nb_accuracy, dt_accuracy]
Models = ['NaiveBayes','DecisionTree']
sns.barplot(x=Models,y=accuracy).set(title="Titanic Dataset")
y_score_gnb = nb_model.predict_proba(x_valid)[:, 1]
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_valid, y_score_gnb)
roc_auc_gnb = roc_auc_score(y_valid, y_score_gnb)
y_score_dtc = dt_model.predict_proba(x_valid)[:, 1]
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_valid, y_score_dtc)
roc_auc_dtc = roc_auc_score(y_valid, y_score_dtc)
plt.figure(figsize=(8, 6))
plt.plot(fpr_gnb, tpr_gnb, color='blue', lw=2, label='Naive Bayes ROC curve (AUC = {:.2f})'.format(roc_auc_gnb))
plt.plot(fpr_dtc, tpr_dtc, color='green', lw=2, label='Decision Tree ROC curve (AUC =
{:.2f})'.format(roc_auc_dtc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()

q = y_valid
pred_test = dt_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='DecisionTree Confusion Matrix', cmap='BuGn')
y_valid = q

q = y_valid
pred_test = nb_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='GaussianNB Confusion Matrix', cmap='BuGn')
y_test = q

from sklearn.model_selection import KFold, cross_val_score
kf = KFold(n_splits=7,shuffle=True,random_state=10)
cv_score = cross_val_score(estimator=nb_model,X=x_train,y=y_train,cv=kf,scoring='accuracy')
mean_cv_score = np.mean(cv_score)
print(mean_cv_score)
```

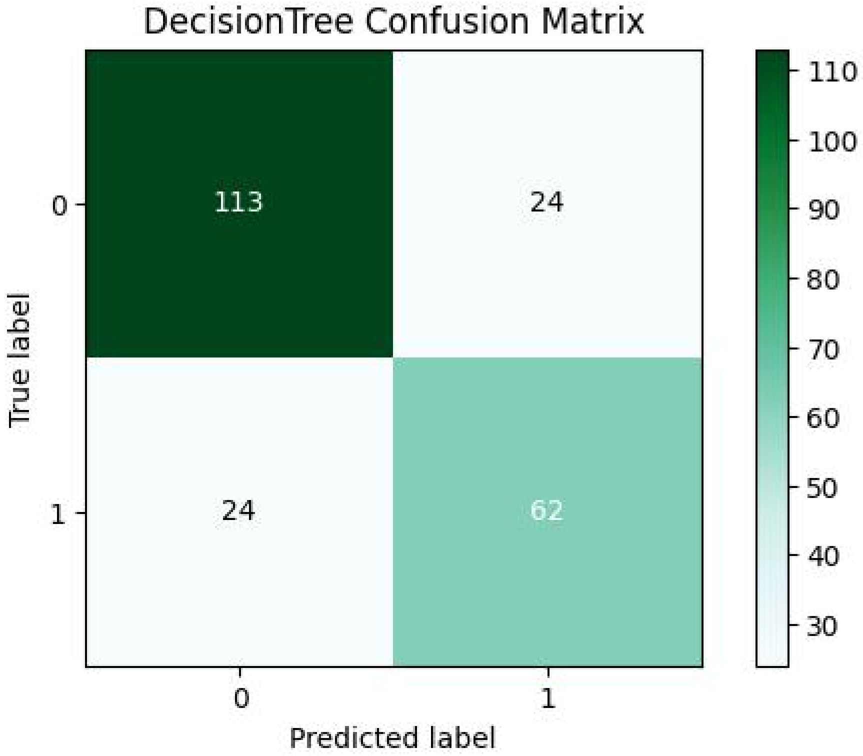
```
# ensemble tech - RandomForest
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(x_train, y_train)
rf_accuracy = rf_model.score(x_valid, y_valid)
print(rf_accuracy)
```

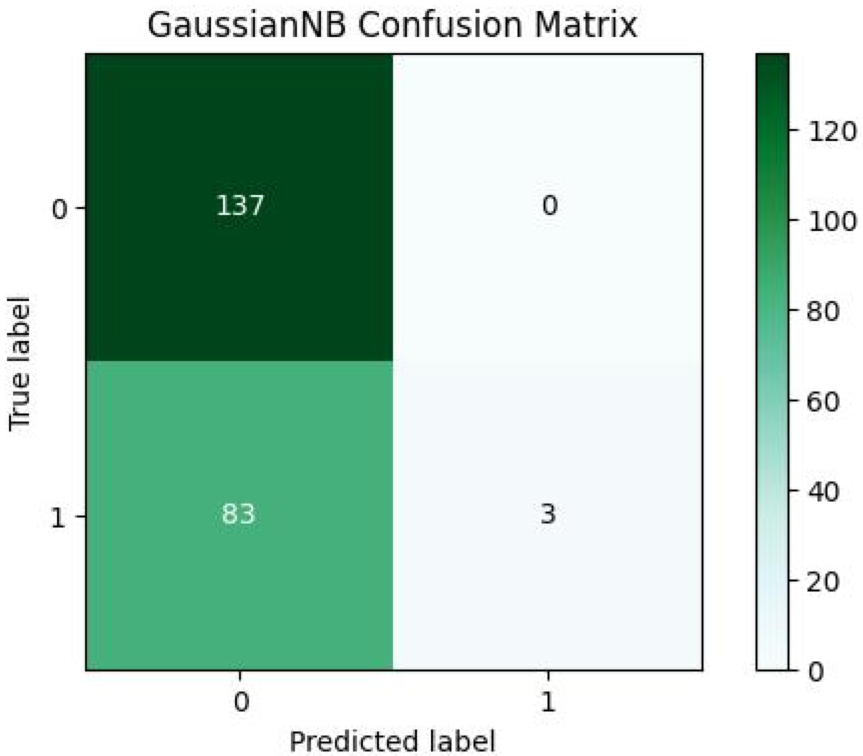
```
accuracies=[nb_accuracy,dt_accuracy,rf_accuracy,mean_cv_score]
plt.figure(figsize=(10,5))
models=['Naive Bayes','Decision Tree','Random Forest','Naive Bayes + Cross validation']
sns.barplot(x=models,y=accuracies,).set(title='Part C')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy of Different Models')
```

Accuracies of both models :

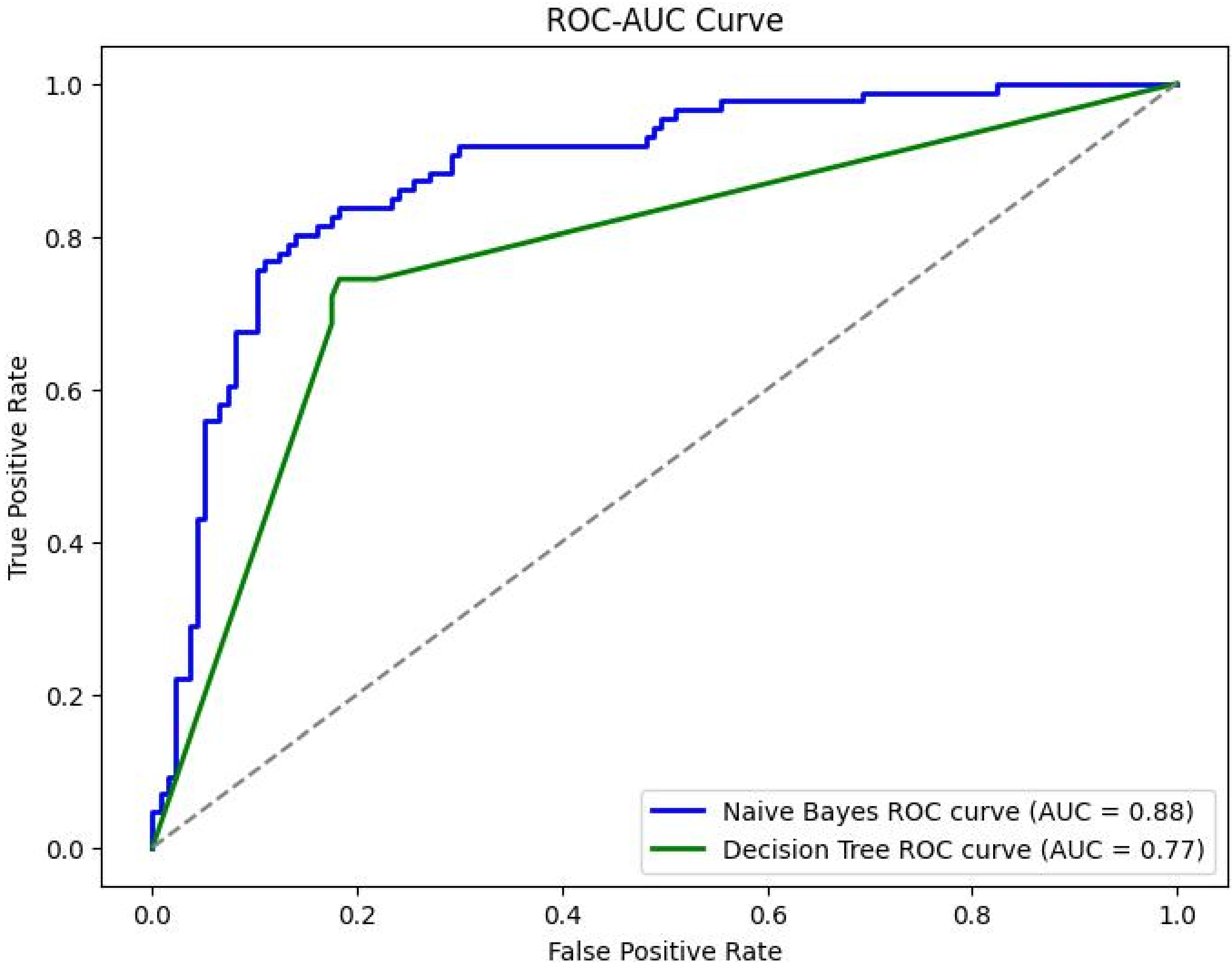


Confusion Matrix :





AUROC:


ROC-AUC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Naive Bayes ROC curve (AUC = 0.88)
Decision Tree ROC curve (AUC = 0.77)

Dataset 2 (penguins):

```
"""### Penguin"""

pg_df = penguin_df
pg_df.head()
pg_df.shape
pg_df.species.unique()
pg_df.island.unique()
pg_df.dropna(inplace=True)
pg_df = pd.get_dummies(pg_df, columns=['island','sex'])
y = pg_df.pop('species')
X = pg_df
x_train, x_valid, y_train, y_valid = train_test_split(X,y,random_state=10,stratify=y, test_size=0.25)
y_train.value_counts(normalize=True)
y_valid.value_counts(normalize=True)
```

```
# using naive bayes
from sklearn.naive_bayes import GaussianNB
nb_model = OneVsRestClassifier(GaussianNB())
nb_model.fit(x_train, y_train)
nb_accuracy = nb_model.score(x_valid, y_valid)
print(nb_accuracy)
```

```
# using decision tree
from sklearn.tree import DecisionTreeClassifier
dt_model = OneVsRestClassifier(DecisionTreeClassifier())
dt_model.fit(x_train, y_train)
dt_accuracy = dt_model.score(x_valid, y_valid)
print(dt_accuracy)
gaussian.append(nb_accuracy)
decision.append(dt_accuracy)
```

```
accuracy = [nb_accuracy, dt_accuracy]
Models = ['NaiveBayes','DecisionTree']
sns.barplot(x=Models,y=accuracy).set(title="Penguin Dataset")
```

```
y_score_gnb = nb_model.predict_proba(x_valid)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_valid.values, y_score_gnb.values)
roc_auc_gnb = roc_auc_score(y_valid, y_score_gnb)
```

```
y_score_dtc = dt_model.predict_proba(x_valid)
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_valid.values, y_score_dtc.values)
roc_auc_dtc = roc_auc_score(y_valid, y_score_dtc)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr_gnb, tpr_gnb, color='blue', lw=2, label='Naive Bayes ROC curve (AUC = {:.2f})'.format(roc_auc_gnb))
plt.plot(fpr_dtc, tpr_dtc, color='green', lw=2, label='Decision Tree ROC curve (AUC =
{:.2f})'.format(roc_auc_dtc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()
```

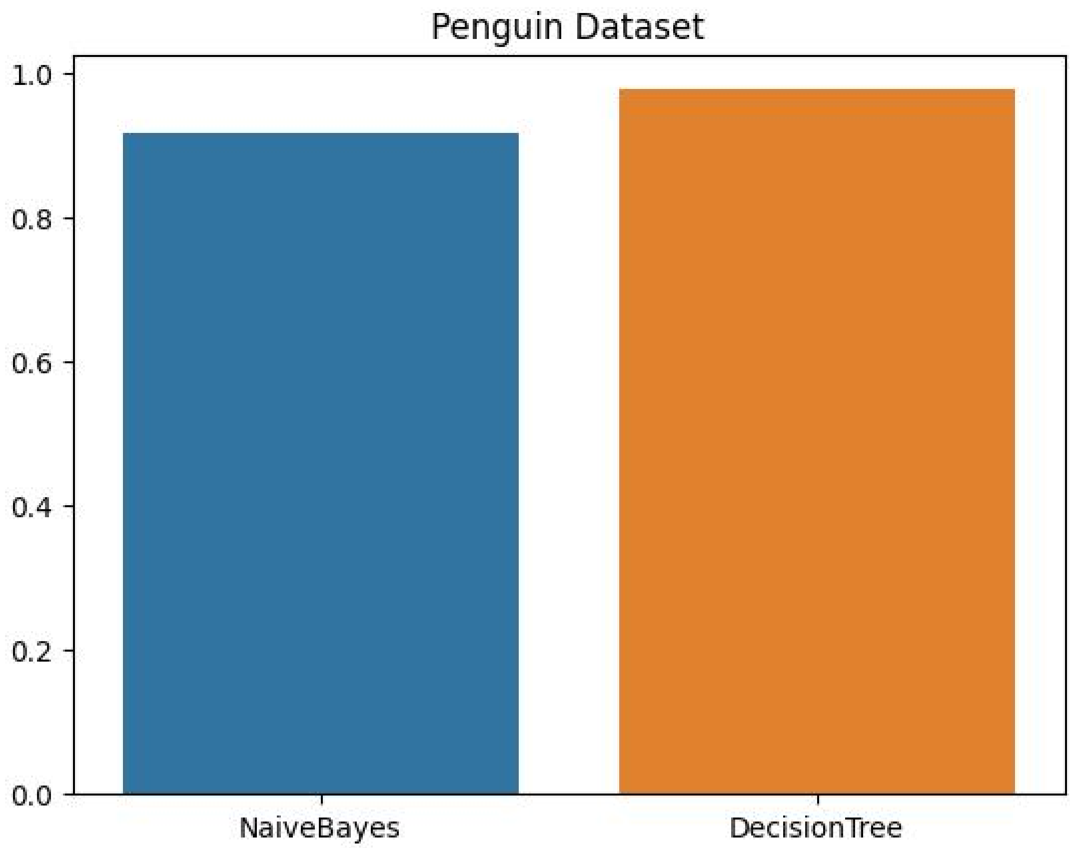
```
#  confusion matrix
q = y_valid
pred_test = dt_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='DecisionTree Confusion Matrix', cmap='BuGn')
```

```
y_test = q
```

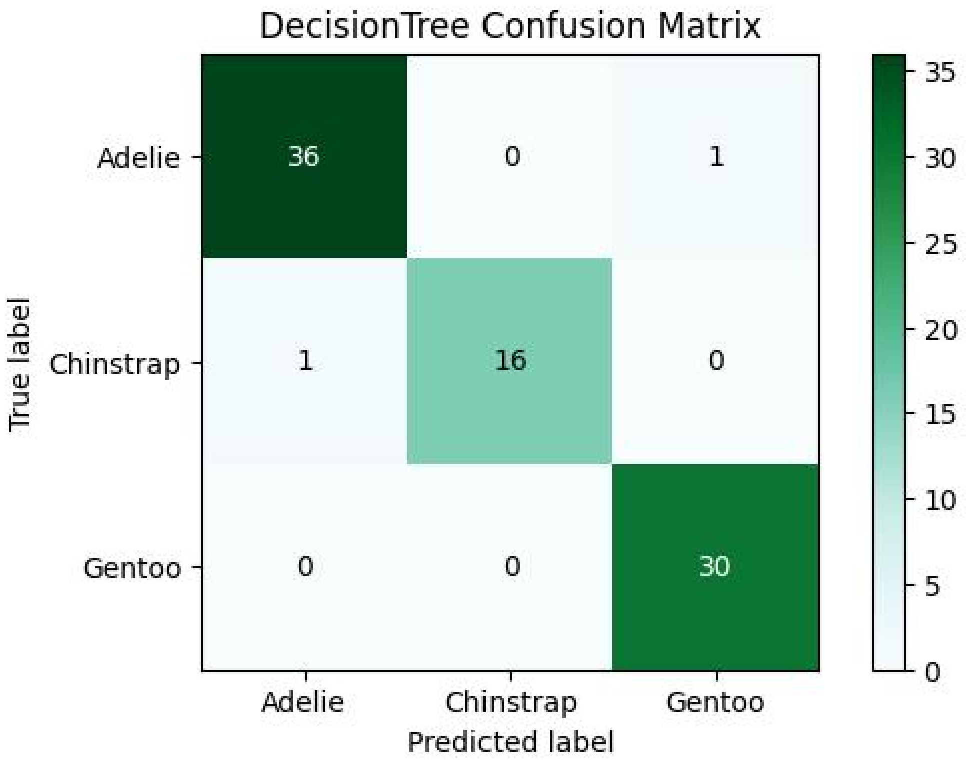
```
q = y_valid
pred_test = nb_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='GaussianNB Confusion Matrix', cmap='BuGn')
y_test = q
```

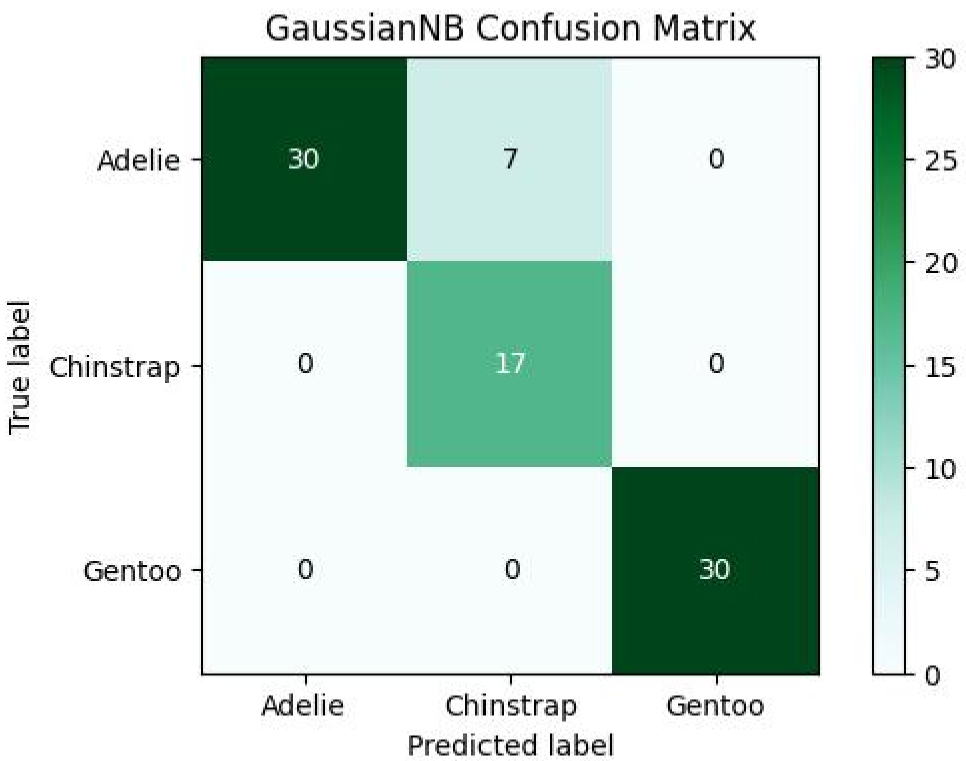
```
# auc - roc
y_score_gnb = nb_model.predict_proba(x_valid)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_valid.values, y_score_gnb.values)
roc_auc_gnb = roc_auc_score(y_valid, y_score_gnb)
y_score_dtc = dt_model.predict_proba(x_valid)
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_valid.values, y_score_dtc.values)
roc_auc_dtc = roc_auc_score(y_valid, y_score_dtc)
plt.figure(figsize=(8, 6))
plt.plot(fpr_gnb, tpr_gnb, color='blue', lw=2, label='Naive Bayes ROC curve (AUC = {:.2f})'.format(roc_auc_gnb))
plt.plot(fpr_dtc, tpr_dtc, color='green', lw=2, label='Decision Tree ROC curve (AUC =
{:.2f})'.format(roc_auc_dtc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()
```

Accuracies of both models :

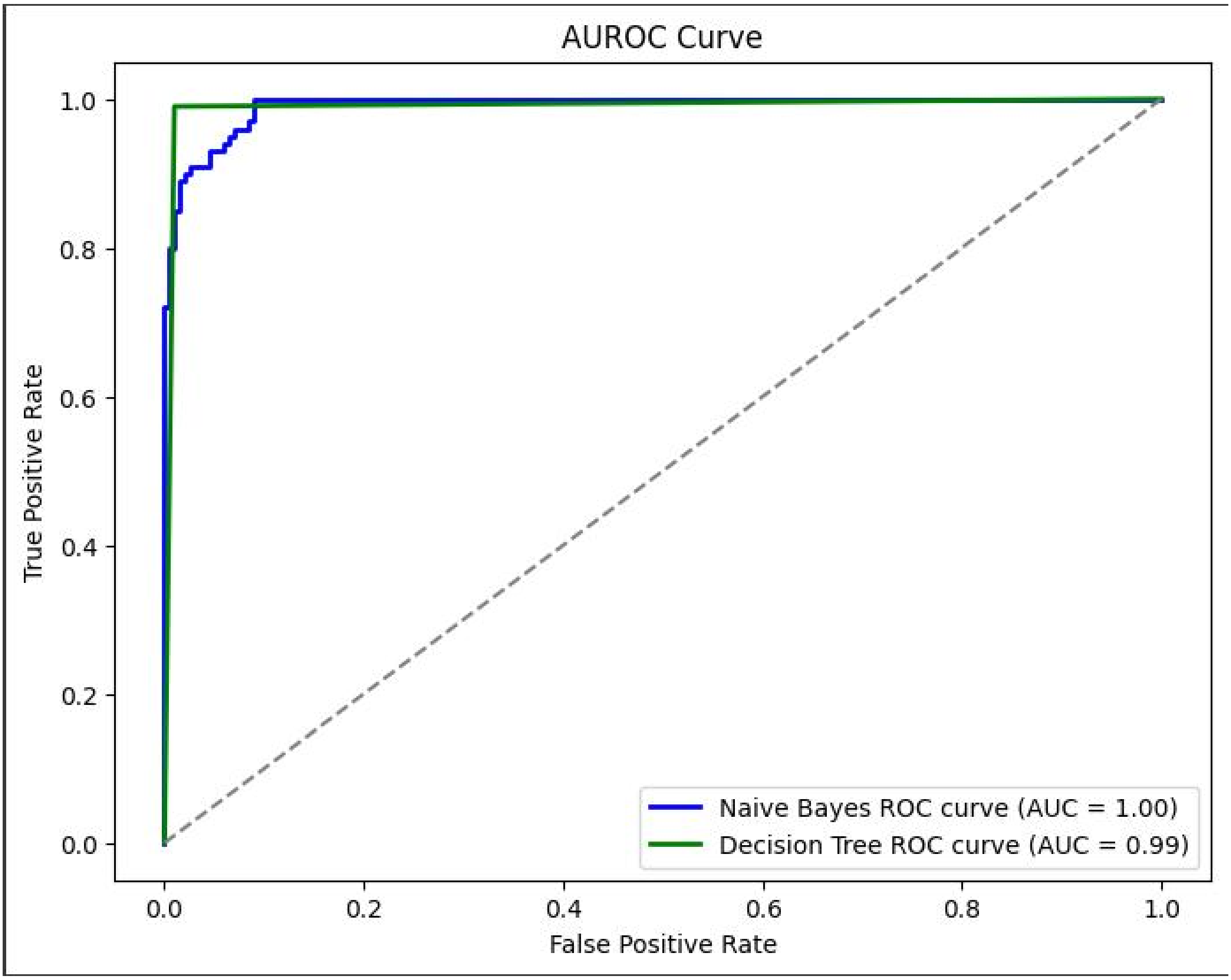


Confusion Matrix :





AUROC:


AUROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes ROC curve (AUC = 1.00)
Decision Tree ROC curve (AUC = 0.99)

Dataset 3 (Iris):

```
"""### Iris"""
iris_df.head()
iris_df.species.unique()
y = iris_df.pop('species')
X = iris_df
x_train, x_valid, y_train, y_valid = train_test_split(X,y,random_state=10,stratify=y, test_size=0.25)
y_train.value_counts(normalize=True)
```

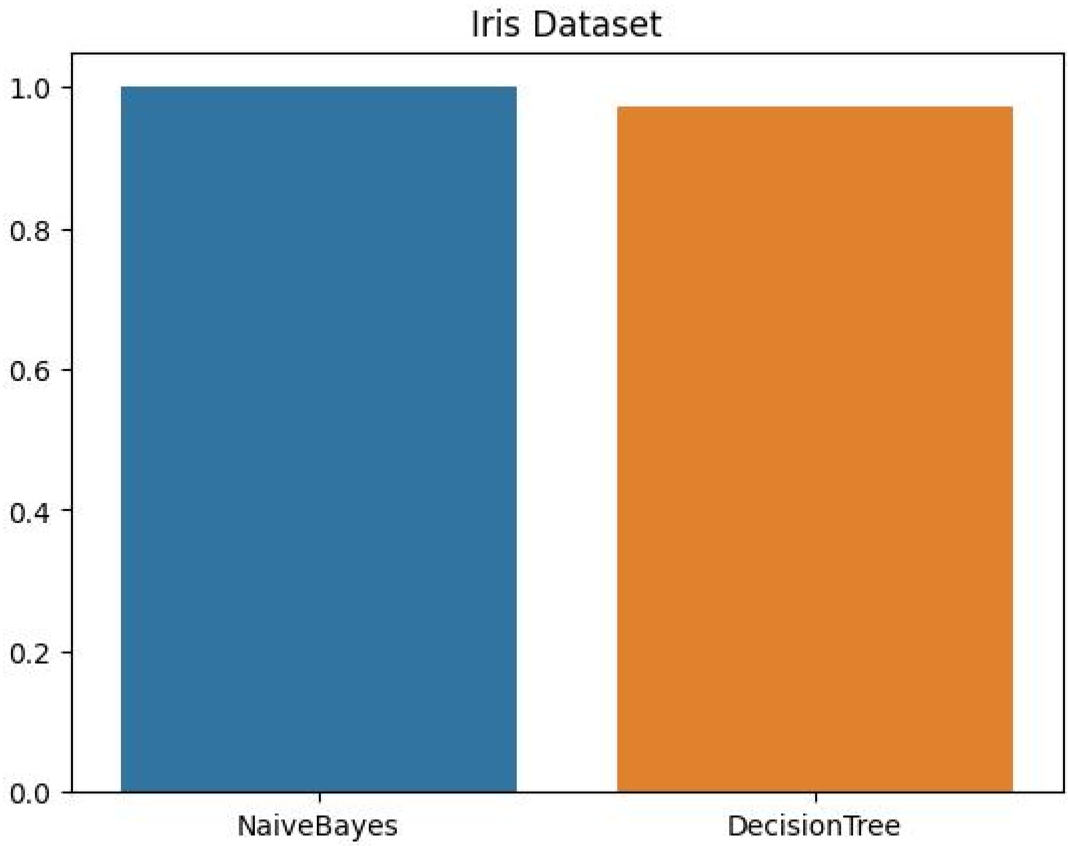
```
# using naive bayes
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(x_train, y_train)
nb_accuracy = nb_model.score(x_valid, y_valid)
print(nb_accuracy)
```

```
# using decision tree
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train, y_train)
dt_accuracy = dt_model.score(x_valid, y_valid)
print(dt_accuracy)
gaussian.append(nb_accuracy)
decision.append(dt_accuracy)
accuracy = [nb_accuracy, dt_accuracy]
Models = ['NaiveBayes','DecisionTree']
sns.barplot(x=Models,y=accuracy).set(title="Iris Dataset")
```

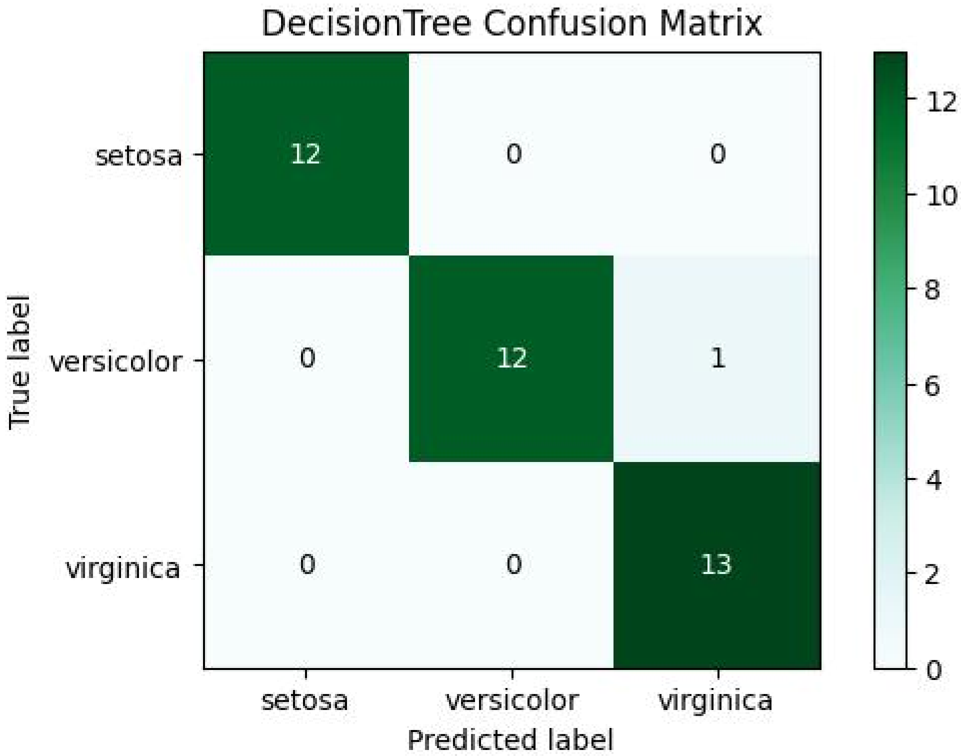
```
q = y_valid
pred_test = dt_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='DecisionTree Confusion Matrix', cmap='BuGn')
y_test = q
q = y_valid
pred_test = nb_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='GaussianNB Confusion Matrix', cmap='BuGn')
y_test = q
```

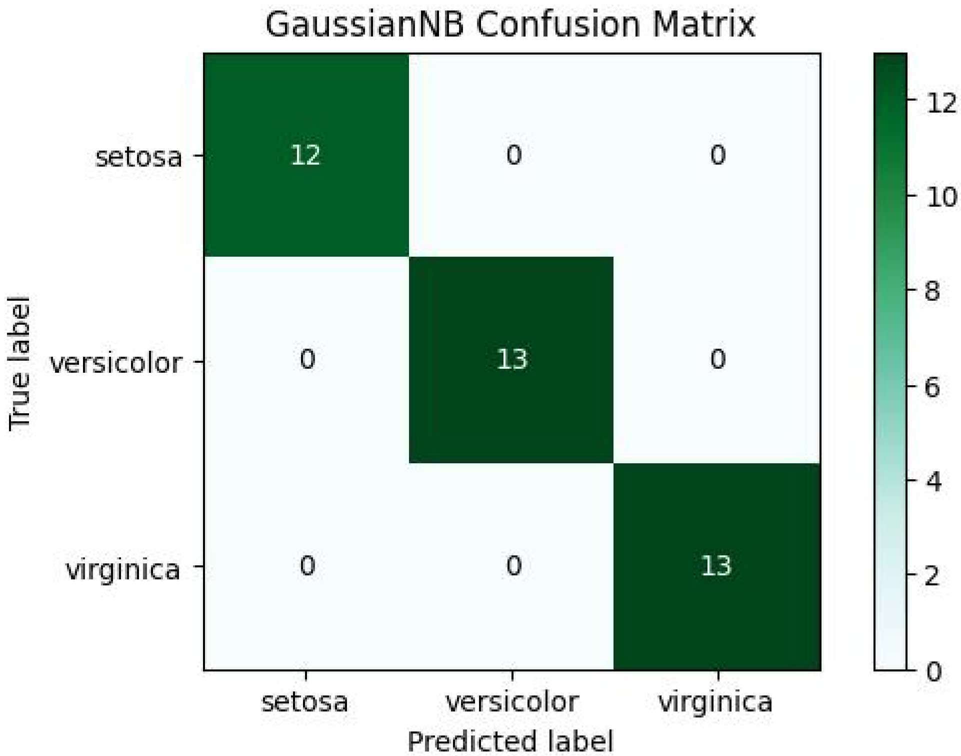
```
# auc - roc
y_score_gnb = nb_model.predict_proba(x_valid)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_valid.values, y_score_gnb.values)
roc_auc_gnb = roc_auc_score(y_valid, y_score_gnb)
y_score_dtc = dt_model.predict_proba(x_valid)
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_valid.values, y_score_dtc.values)
roc_auc_dtc = roc_auc_score(y_valid, y_score_dtc)
plt.figure(figsize=(8, 6))
plt.plot(fpr_gnb, tpr_gnb, color='blue', lw=2, label='Naive Bayes ROC curve (AUC = {:.2f})'.format(roc_auc_gnb))
plt.plot(fpr_dtc, tpr_dtc, color='green', lw=2, label='Decision Tree ROC curve (AUC =
{:.2f})'.format(roc_auc_dtc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()
```

Accuracies of both models :

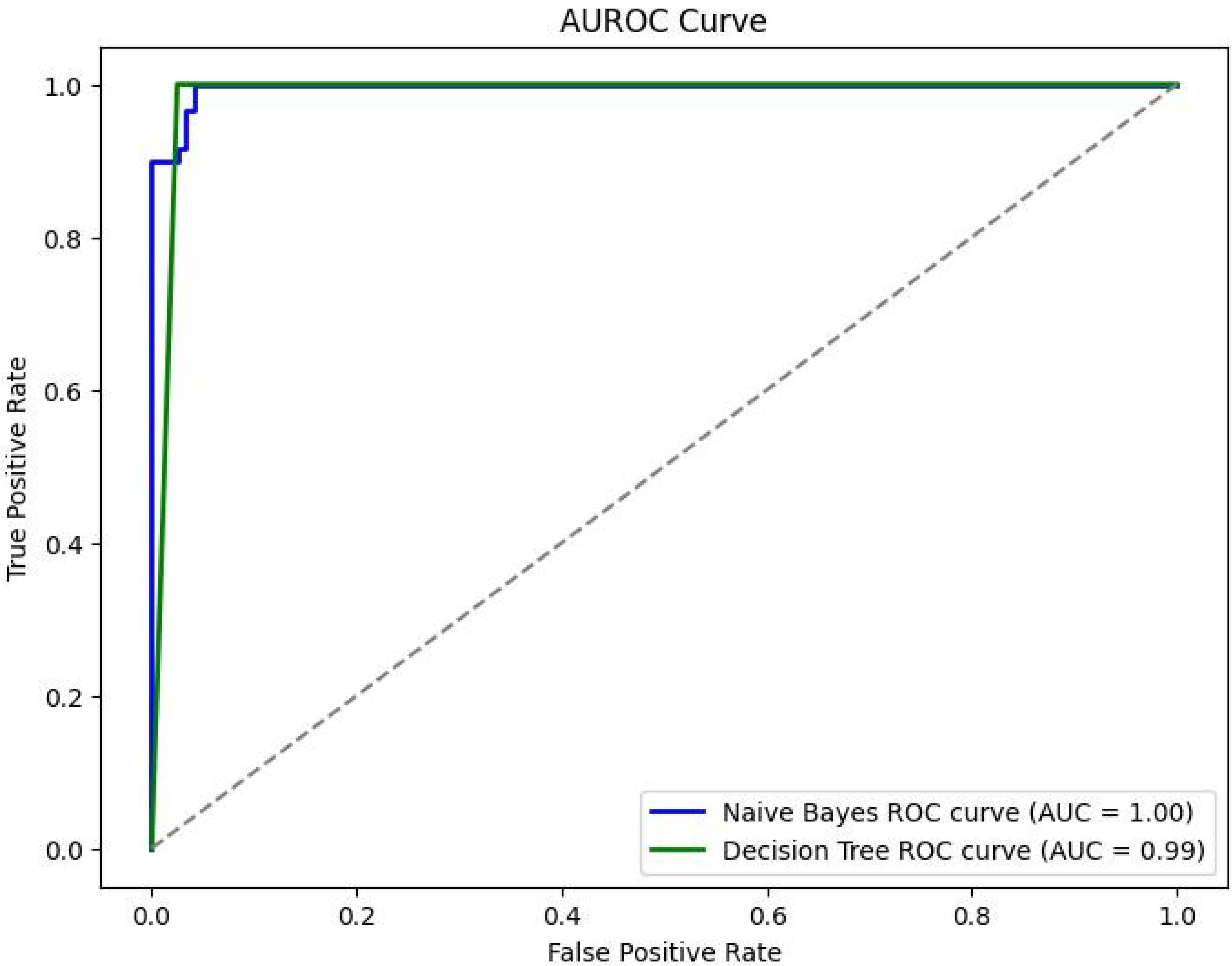


Confusion Matrix :





AUROC:


AUROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes ROC curve (AUC = 1.00)
Decision Tree ROC curve (AUC = 0.99)

Dataset 4 (Email spam-ham dataset):

```
"""### spam"""
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
import string
from nltk.tokenize import word_tokenize
import re
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import lightgbm as ltb
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
spam_df.head()
def cleaning (text):
    # text = text.lower()
    text =  re.sub(r'@\S+', '',text)
    text =  re.sub(r'http\S+', '',text) # remove urls
    text =  re.sub(r'pic.\S+', '',text)
    text =  re.sub(r"[^a-zA-ZáéíóúÁÉÍÓÚ']", ' ',text) # only keeps characters
    text =  re.sub(r'\s+[a-zA-ZáéíóúÁÉÍÓÚ]\s+', ' ', text+' ')  # keep words with length>1 only
    text = "".join([i for i in text if i not in string.punctuation])
    words = word_tokenize(text)
    stopwords = nltk.corpus.stopwords.words('english')   # remove stopwords
    text = " ".join([i for i in words if i not in stopwords])
    text= re.sub("\s[\s]+", " ",text).strip()
    text= re.sub("\s[\s]+", " ",text).strip() # remove repeated/leading/trailing spaces
    return text
spam_df["Message"]=spam_df["Message"].apply(cleaning)
def lemmatize(data):
    wordnet = WordNetLemmatizer()
    lemmanized = []
    for i in range(len(data)):
        lemmed = []
        words = word_tokenize(data['Message'].iloc[i])
        for w in words:
            lemmed.append(wordnet.lemmatize(w))
        lemmanized.append(lemmed)
```

```
    data['lemmanized'] = lemmanized
    data['text'] = data['lemmanized'].apply(' '.join)
    data=data.drop("lemmanized",axis=1)
    data=data.drop("Message",axis=1)
    return data
spam_df = lemmatize(spam_df)
```

```
obj = {"ham":0,"spam":1}
spam_df["Category"]=spam_df["Category"].map(obj)
X = spam_df['text']
y = spam_df['Category']
x_train, x_valid, y_train, y_valid = train_test_split(X,y,random_state=10,stratify=y, test_size=0.25)
y_train.value_counts(normalize=True)
from sklearn.feature_extraction.text import TfidfVectorizer
x_train.shape
tfidf = TfidfVectorizer()
X_train = tfidf.fit_transform(x_train)
x_valid = tfidf.transform(x_valid)
```

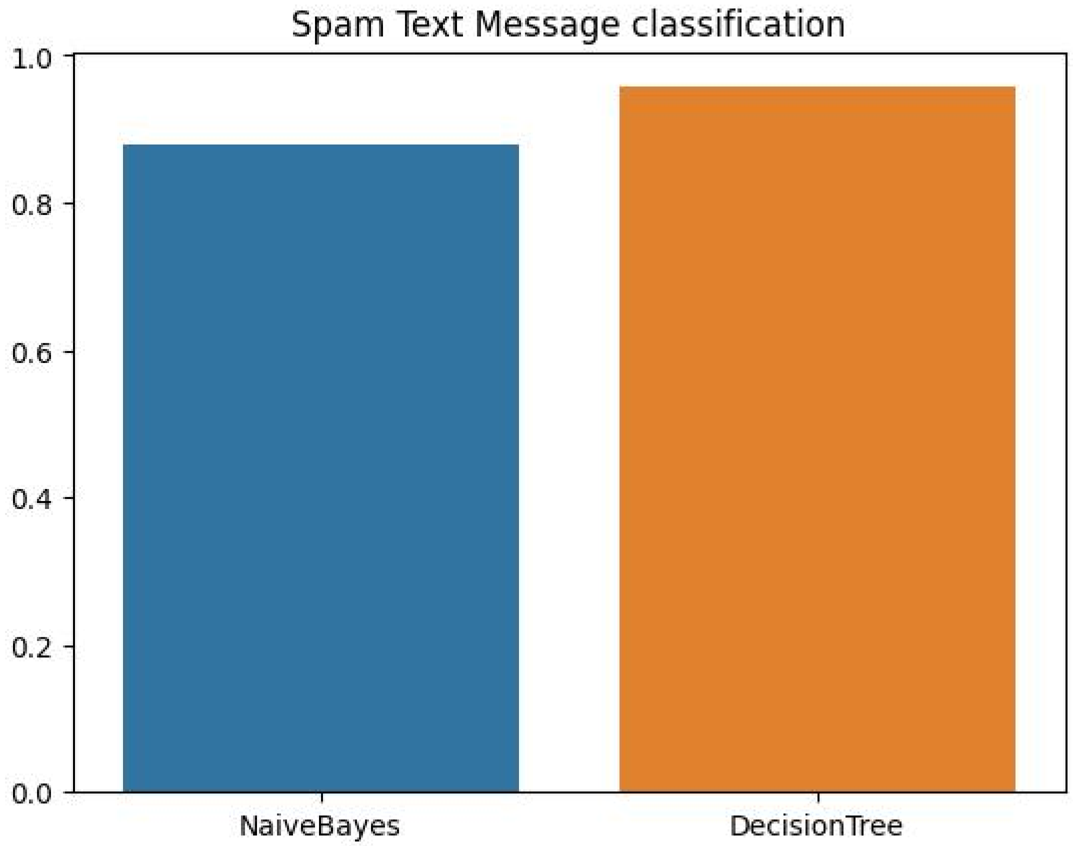
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
pred = dt_model.predict(x_valid)
dt_accuracy = accuracy_score(pred, y_valid)
print(dt_accuracy)
```

```
nb_model = GaussianNB()
nb_model.fit(X_train.toarray(), y_train)
pred = nb_model.predict(x_valid.toarray())
nb_accuracy = accuracy_score(pred, y_valid)
print(nb_accuracy)
gaussian.append(nb_accuracy)
decision.append(dt_accuracy)
accuracy = [nb_accuracy, dt_accuracy]
Models = ['NaiveBayes','DecisionTree']
sns.barplot(x=Models,y=accuracy).set(title="Spam Text Message classification")
```

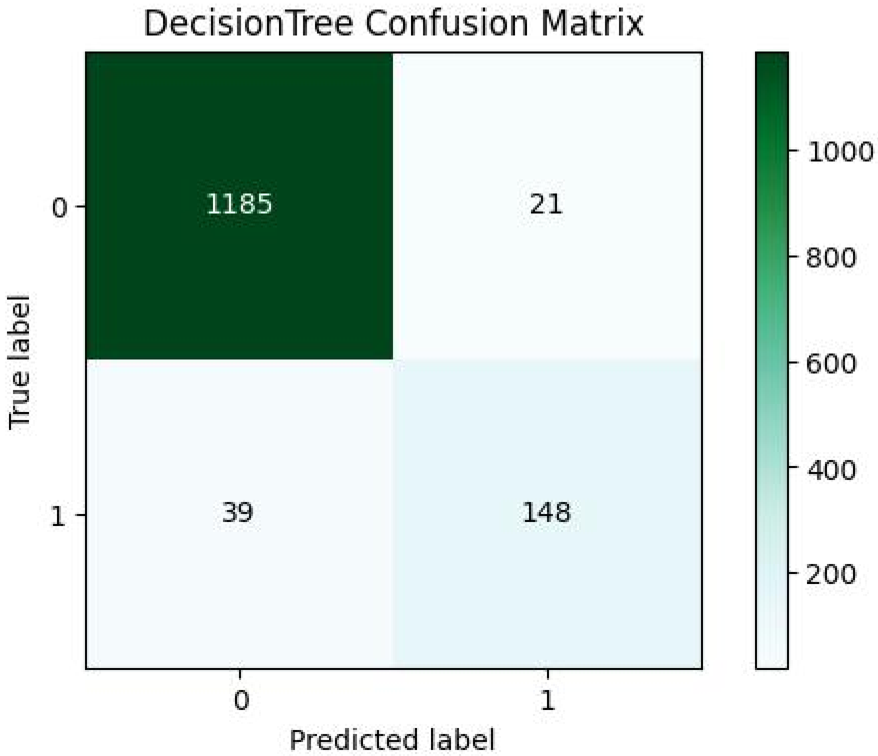
```
# auc - roc
y_score_gnb = nb_model.predict_proba(x_valid.toarray())[:, 1]
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_valid, y_score_gnb)
roc_auc_gnb = roc_auc_score(y_valid, y_score_gnb)
y_score_dtc = dt_model.predict_proba(x_valid)[:, 1]
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_valid, y_score_dtc)
roc_auc_dtc = roc_auc_score(y_valid, y_score_dtc)
plt.figure(figsize=(8, 6))
plt.plot(fpr_gnb, tpr_gnb, color='blue', lw=2, label='Naive Bayes ROC curve (AUC = {:.2f})'.format(roc_auc_gnb))
plt.plot(fpr_dtc, tpr_dtc, color='green', lw=2, label='Decision Tree ROC curve (AUC =
{:.2f})'.format(roc_auc_dtc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()
```

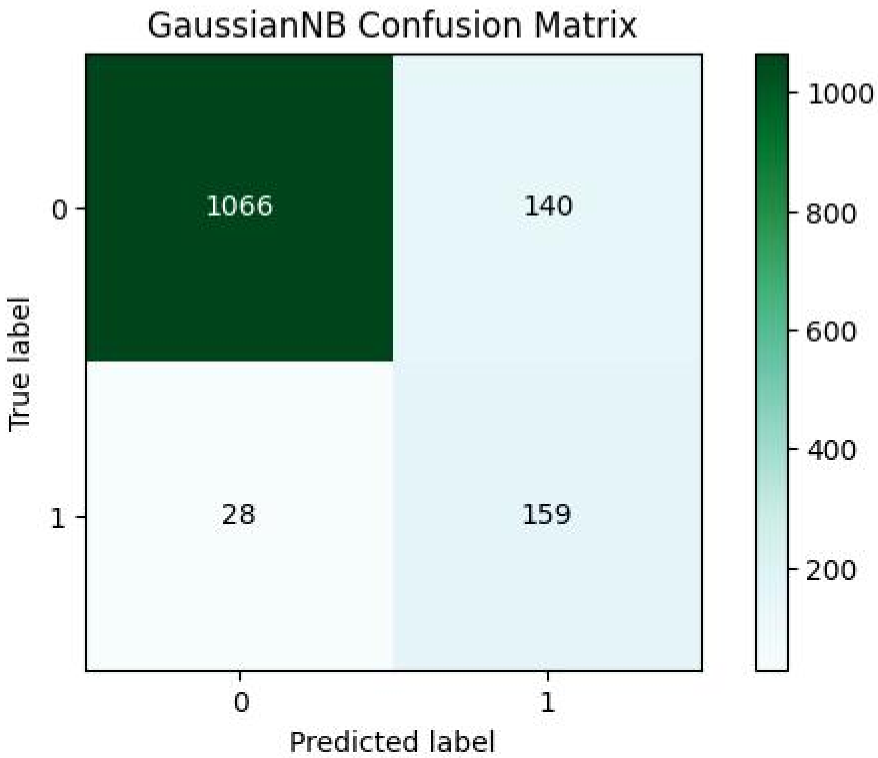
```
# confusion matrix
q = y_valid
pred_test = dt_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='DecisionTree Confusion Matrix', cmap='BuGn')
y_valid = q
q = y_valid
pred_test = nb_model.predict(x_valid.toarray())
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='GaussianNB Confusion Matrix', cmap='BuGn')
y_valid = q
```

Accuracies of both models :

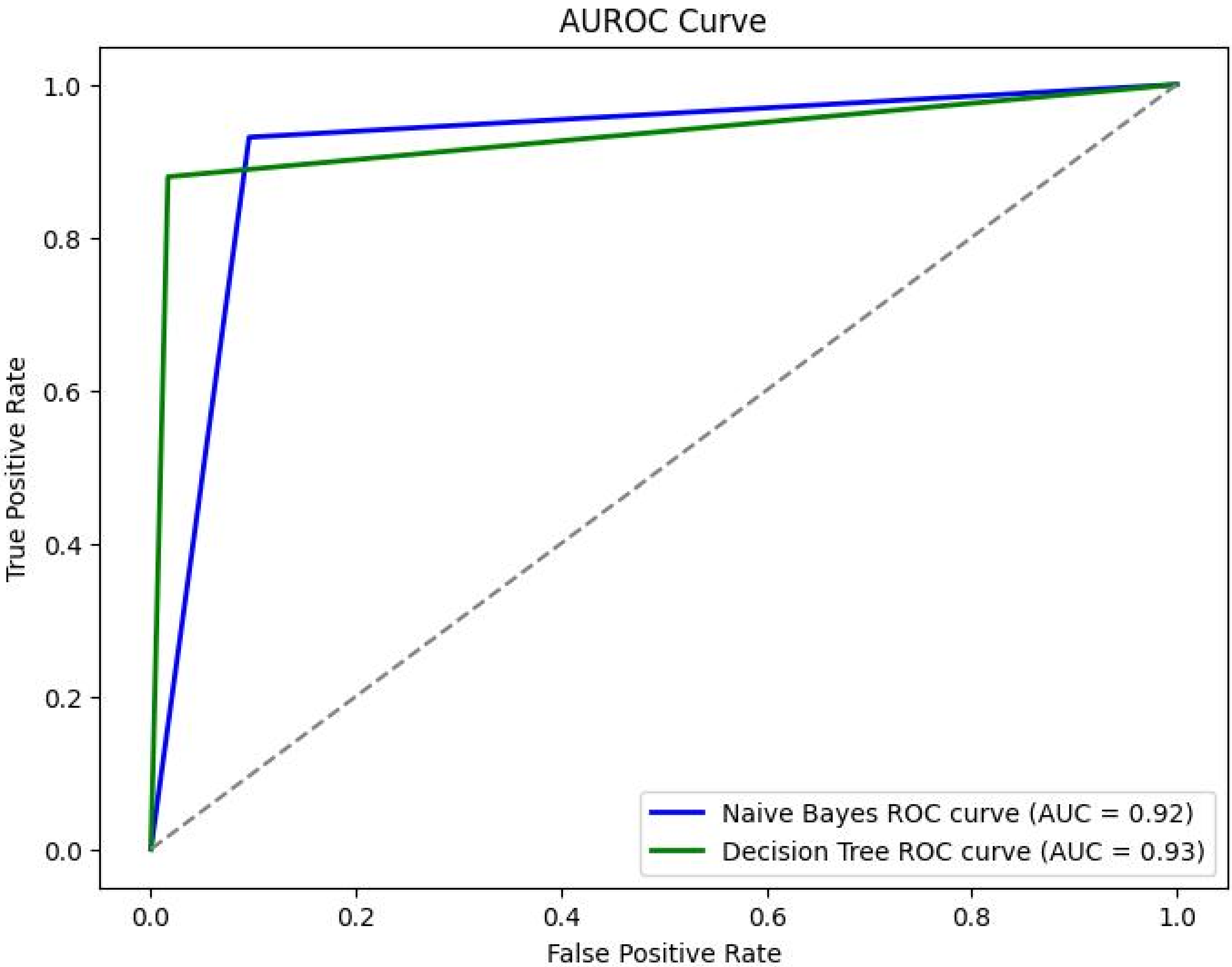


Confusion Matrix :





AUROC:



AUROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Naive Bayes ROC curve (AUC = 0.92)
Decision Tree ROC curve (AUC = 0.93)

Dataset 5 (Wine Quality Dataset):

```
"""### wine dataset"""
wine_df.head()
y = wine_df.pop('quality')
X = wine_df
x_train, x_valid, y_train, y_valid = train_test_split(X,y,random_state=10,stratify=y, test_size=0.25)
y_train.value_counts(normalize=True)
```

```
# using naive bayes
nb_model = GaussianNB()
nb_model.fit(x_train, y_train)
nb_accuracy = nb_model.score(x_valid, y_valid)
print(nb_accuracy)
```

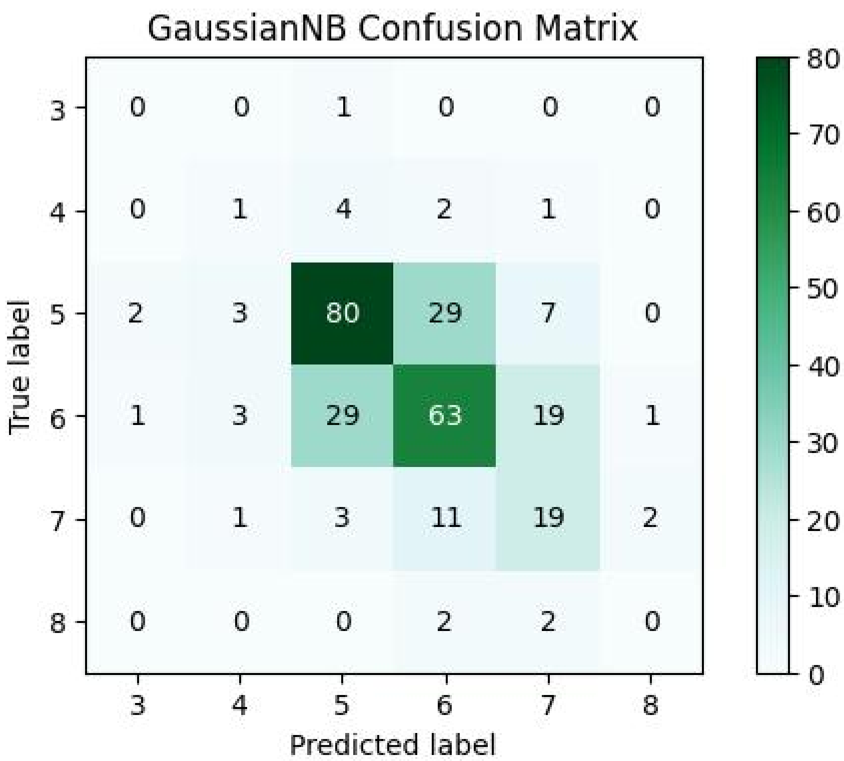
```
# using decision tree
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train, y_train)
dt_accuracy = dt_model.score(x_valid, y_valid)
print(dt_accuracy)
gaussian.append(nb_accuracy)
decision.append(dt_accuracy)
accuracy = [nb_accuracy, dt_accuracy]
Models = ['NaiveBayes','DecisionTree']
sns.barplot(x=Models,y=accuracy).set(title="Wine Quality Classifier")
```

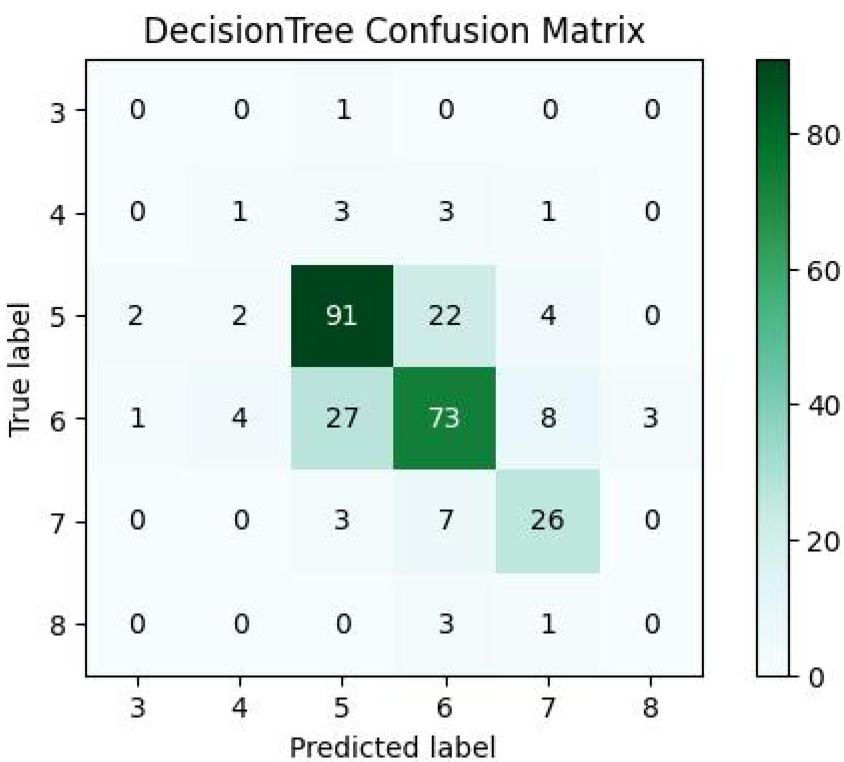
```
# roc - auc
y_score_gnb = nb_model.predict_proba(x_valid)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_valid.values, y_score_gnb.values)
roc_auc_gnb = roc_auc_score(y_valid, y_score_gnb)
y_score_dtc = dt_model.predict_proba(x_valid)
fpr_dtc, tpr_dtc, thresholds_dtc = roc_curve(y_valid.values, y_score_dtc.values)
roc_auc_dtc = roc_auc_score(y_valid, y_score_dtc)
plt.figure(figsize=(8, 6))
plt.plot(fpr_gnb, tpr_gnb, color='blue', lw=2, label='Naive Bayes ROC curve (AUC = {:.2f})'.format(roc_auc_gnb))
plt.plot(fpr_dtc, tpr_dtc, color='green', lw=2, label='Decision Tree ROC curve (AUC =
{:.2f})'.format(roc_auc_dtc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
#  confusion matrix
q = y_valid
pred_test = nb_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='GaussianNB Confusion Matrix', cmap='BuGn')
y_test= q
q = y_valid
pred_test = dt_model.predict(x_valid)
pred_test = pd.DataFrame(pred_test)
y_valid = pd.DataFrame(y_valid)
plot_confusion_matrix(y_valid, pred_test, figsize=(7,4), title='DecisionTree Confusion Matrix', cmap='BuGn')
ytest = q
```

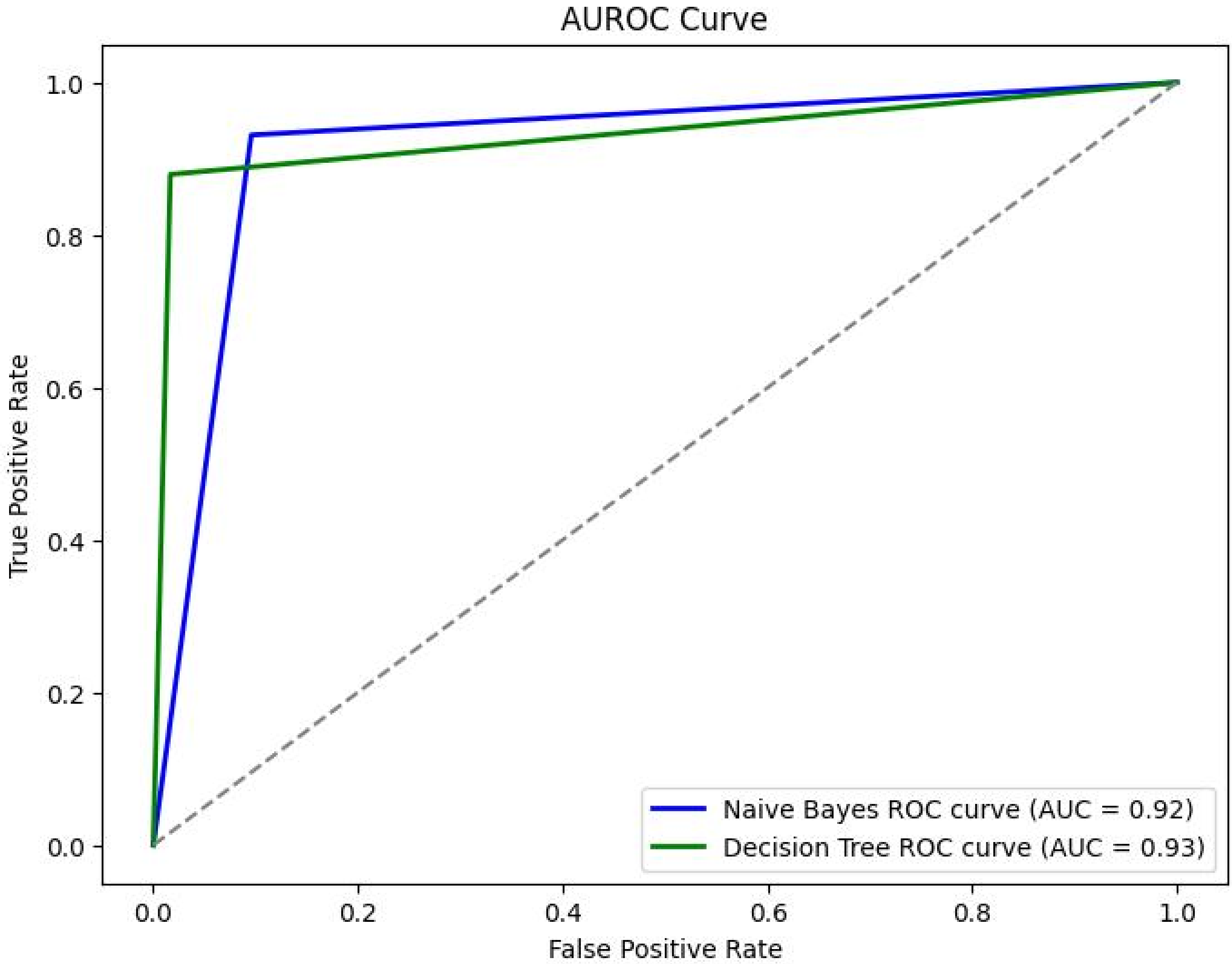
Accuracies of both models :

Confusion Matrix :





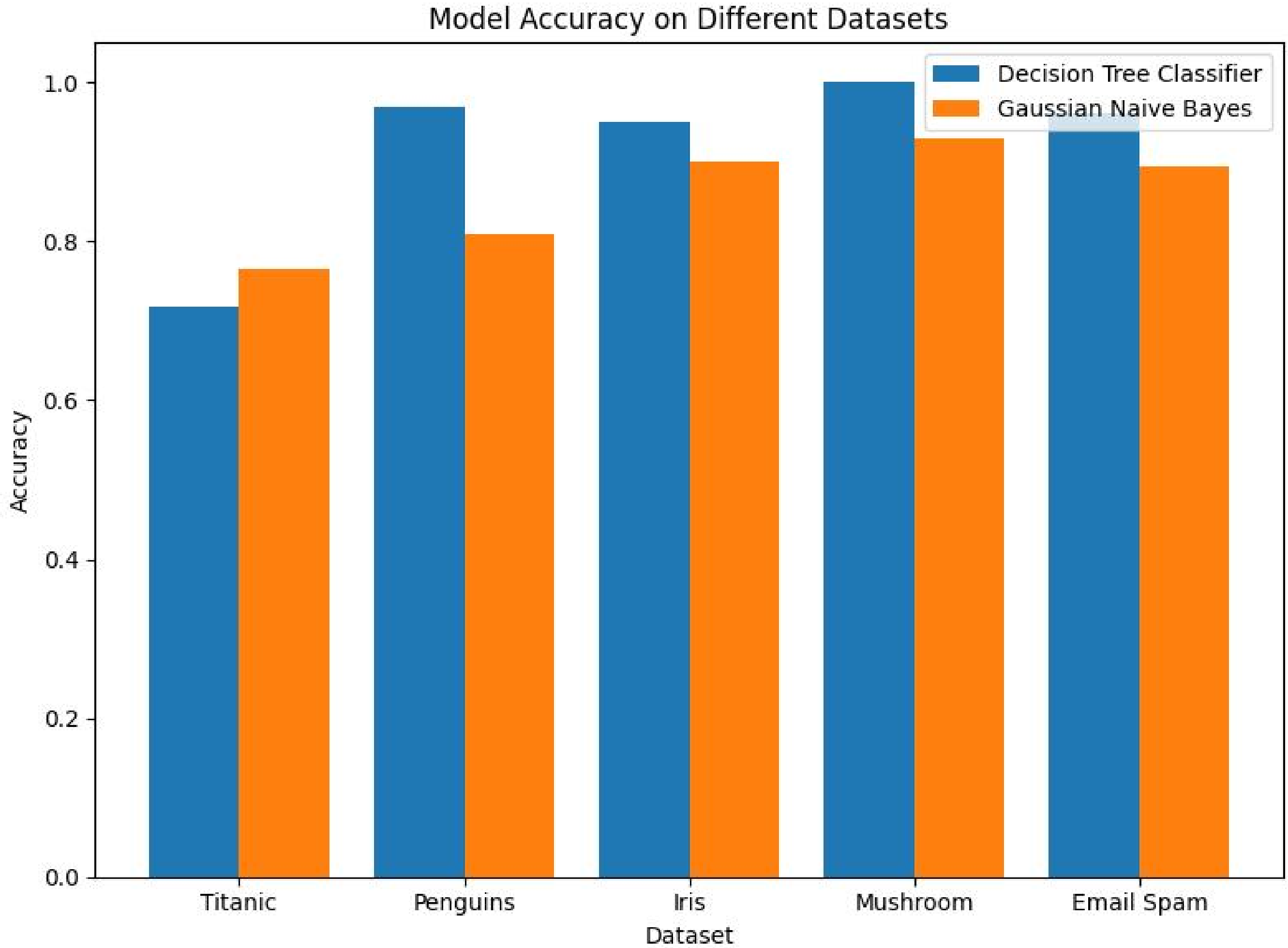
AUROC:


AUROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Naive Bayes ROC curve (AUC = 0.92)
Decision Tree ROC curve (AUC = 0.93)

Plot comparison graphs using the results of DT and NB

```
def grouped_barplot(data1, data2, labels, xticklabels=None, title="Grouped Barplot"):
    positions = np.arange(len(labels))
    width = 0.35
    # Create the grouped barplot
    plt.figure(figsize=(8, 6))
    plt.bar(positions - width/2, data1, width, label='Group 1')
    plt.bar(positions + width/2, data2, width, label='Group 2')
    plt.xlabel('Dataset')
    plt.ylabel('Acurracy')
    plt.title(title)
    # Set x-axis ticks and labels
    plt.xticks(positions, labels)
    if xticklabels:
        plt.xticks(positions, xticklabels)
    plt.legend()
    plt.show()
category_labels = ['Titanic', 'Penguin', 'Iris', 'Email spam', 'Wine']
grouped_barplot(gaussian, decision, category_labels, title="Model Comparison")
```

comparison graphs:



Part C:

Modify DT/NB to use k-fold cross validation and ensemble models :Kfolds cross validation of 7 folds :

Modification Titanic Dataset :

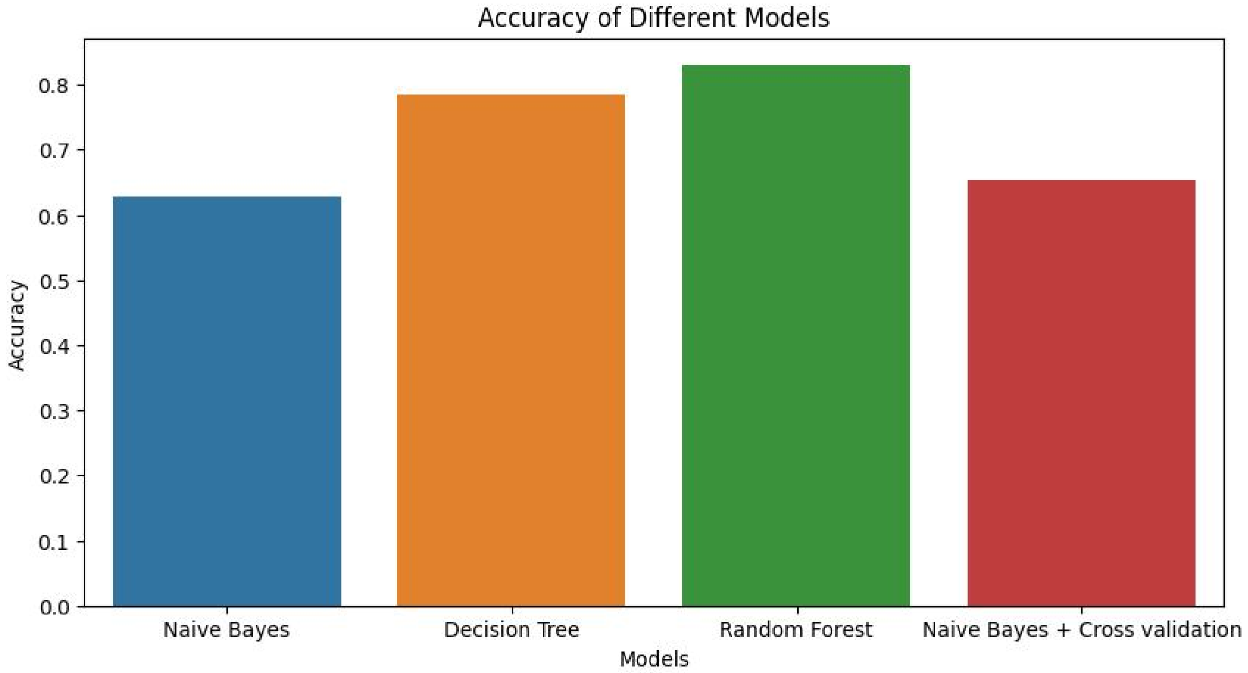
```
from sklearn.model_selection import KFold, cross_val_score

kf = KFold(n_splits=7,shuffle=True,random_state=10)

cv_score = cross_val_score(estimator=nb_model,X=x_train,y=y_train,cv=kf,scoring='accuracy')

mean_cv_score = np.mean(cv_score)

print(mean_cv_score)
```

```
# ensemble tech - RandomForest

from sklearn.ensemble import RandomForestClassifier
```

```
rf_model = RandomForestClassifier()

rf_model.fit(x_train, y_train)

rf_accuracy = rf_model.score(x_valid, y_valid)

print(rf_accuracy)
```

```
accuracies=[nb_accuracy,dt_accuracy,rf_accuracy,mean_cv_score]

plt.figure(figsize=(10,5))

models=['Naive Bayes','Decision Tree','Random Forest','Naive Bayes + Cross validation']

sns.barplot(x=models,y=accuracies,).set(title='Part C')

plt.xlabel('Models')

plt.ylabel('Accuracy')

plt.title('Accuracy of Different Models')
```

Comparisonk-fold cross validation and ensemble models :



Conclusion:

Thus, we have successfully implemented Classification algorithm using Decision Tree ID3 and Naïve Bayes algorithm and performed all the parts