Build End-to-End ML pipeline for Warfarin Dosing Prediction

```
# Reading excel file
import pandas as pd
import numpy as np
data_frame = pd.read_excel('FinalData.xls')
data_frame.head()
```

	Gender	Race (Reported)	Age	Height (cm)	Weight (kg)	Diabetes	Simvastatin (Zocor)	Amiodarone (Cordarone)
0	male	White	60 - 69	193.040	115.7	NaN	0.0	0.0
1	female	White	50 - 59	176.530	144.2	NaN	0.0	0.0
2	female	White	40 - 49	162.560	77.1	NaN	0.0	0.0
3	male	White	60 - 69	182.245	90.7	NaN	0.0	0.0
4	male	White	50 - 59	167.640	72.6	NaN	0.0	0.0

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

```
Mounted at /content/drive
```

```
# Checking columns of the dataset
data_frame.columns
```

```
Index(['Gender', 'Race (Reported)', 'Age', 'Height (cm)', 'Weight (kg)',
       'Diabetes', 'Simvastatin (Zocor)', 'Amiodarone (Cordarone)',
       'Target INR', 'INR on Reported Therapeutic Dose of Warfarin',
       'Cyp2C9 genotypes',
       'VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T',
       'Therapeutic Dose of Warfarin'],
      dtype='object')
```

```
# Check missing values in the dataset
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5700 entries, 0 to 5699
Data columns (total 13 columns):
 #   Column                                                              Non-N
---  ------                                                              -----
 0   Gender                                                              5696
 1   Race (Reported)                                                     5194
 2   Age                                                                 5658
 3   Height (cm)                                                         4554
 4   Weight (kg)                                                         5413
 5   Diabetes                                                            3283
 6   Simvastatin (Zocor)                                                 3861
 7   Amiodarone (Cordarone)                                              4182
 8   Target INR                                                          1259
 9   INR on Reported Therapeutic Dose of Warfarin                        4968
 10  Cyp2C9 genotypes                                                    5567
 11  VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T  4046
 12  Therapeutic Dose of Warfarin                                        5528
dtypes: float64(8), object(5)
memory usage: 579.0+ KB
```

```
# Describing the dataset
data_frame.describe()
```

	Height (cm)	Weight (kg)	Diabetes	Simvastatin (Zocor)	Amiodarone (Cordarone)	Target INR
count	4554.000000	5413.000000	3283.000000	3861.000000	4182.000000	1259.000000
mean	168.047778	77.852569	0.187024	0.146335	0.066236	2.538324
std	10.845992	21.859764	0.389990	0.353488	0.248724	0.198140
min	124.968000	30.000000	0.000000	0.000000	0.000000	1.300000
25%	160.020000	62.000000	0.000000	0.000000	0.000000	2.500000
50%	167.894000	75.000000	0.000000	0.000000	0.000000	2.500000
75%	176.022000	90.000000	0.000000	0.000000	0.000000	2.500000

```
# Checking for NA Values in the dataset
data_frame.isna()
```

	Gender	Race (Reported)	Age	Height (cm)	Weight (kg)	Diabetes	Simvastatin (Zocor)	Amiodare (Cordaro
0	False	False	False	False	False	True	False	Fa
1	False	False	False	False	False	True	False	Fa
2	False	False	False	False	False	True	False	Fa
3	False	False	False	False	False	True	False	Fa
4	False	False	False	False	False	True	False	Fa
...	...	...	...	...	...	...	...	
5695	False	False	False	False	False	False	False	Fa
5696	False	False	False	False	False	False	False	Fa
5697	False	False	False	False	False	False	False	Fa
5698	False	False	False	False	False	False	False	Fa
5699	False	False	False	False	False	False	False	Fa

5700 rows × 13 columns

```
# Finding Total Missing Values
data_frame.isna().sum()
```

```
Gender                                                        4
Race (Reported)                                             506
Age                                                          42
Height (cm)                                                1146
Weight (kg)                                                 287
Diabetes                                                   2417
Simvastatin (Zocor)                                        1839
Amiodarone (Cordarone)                                     1518
Target INR                                                 4441
INR on Reported Therapeutic Dose of Warfarin                732
Cyp2C9 genotypes                                            133
VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T    1654
Therapeutic Dose of Warfarin                                172
dtype: int64
```

```
# Total sum of missing values
data_frame.isna().sum().sum()
```

```
14891
```

```
# Replacing missing values with mode
mode_fillers = {
    'Gender': data_frame['Gender'].mode()[0],
    'Race (Reported)': data_frame['Race (Reported)'].mode()[0],
    'Age': data_frame['Age'].mode()[0],
    'Cyp2C9 genotypes': data_frame['Cyp2C9 genotypes'].mode()[0],
    'VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T': data_fra
}

data_frame.fillna(value=mode_fillers, inplace=True)
```

```
# Replacing missing values with mean
mean_fillers = {
    'Height (cm)': data_frame['Height (cm)'].mean(),
    'Weight (kg)': data_frame['Weight (kg)'].mean(),
    'Diabetes': data_frame['Diabetes'].mean(),
    'Simvastatin (Zocor)': data_frame['Simvastatin (Zocor)'].mean(),
    'Amiodarone (Cordarone)': data_frame['Amiodarone (Cordarone)'].mean(),
    'Target INR': data_frame['Target INR'].mean(),
    'INR on Reported Therapeutic Dose of Warfarin': data_frame['INR on Reported T
    'Therapeutic Dose of Warfarin': data_frame['Therapeutic Dose of Warfarin'].me
}

data_frame.fillna(value=mean_fillers, inplace=True)
```

```
# checking for missing values
data_frame.isna()
```

	Gender	Race (Reported)	Age	Height (cm)	Weight (kg)	Diabetes	Simvastatin (Zocor)	Amiodar (Cordaro
0	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	
5695	False	False	False	False	False	False	False	Fa
5696	False	False	False	False	False	False	False	Fa
5697	False	False	False	False	False	False	False	Fa
5698	False	False	False	False	False	False	False	Fa
5699	False	False	False	False	False	False	False	Fa

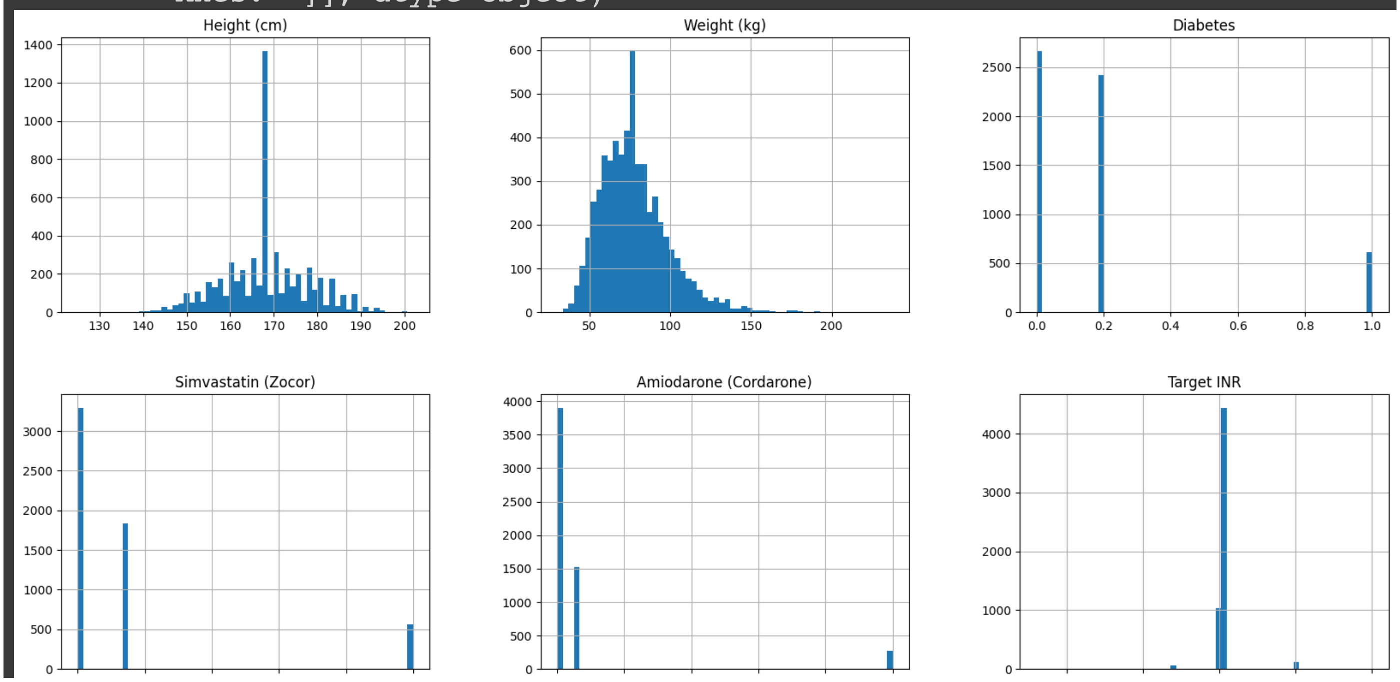
5700 rows × 13 columns

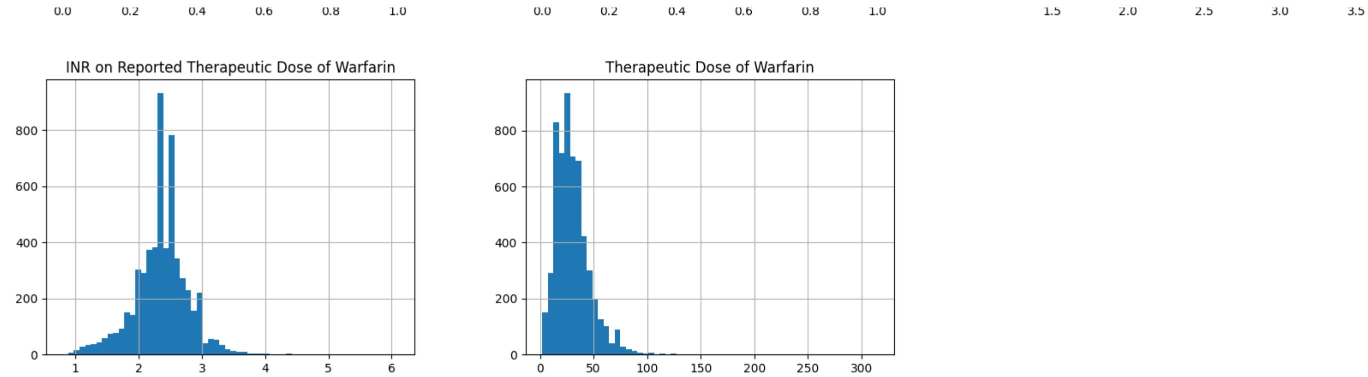
```
data_frame.isna().sum()
```

```
Gender                                                          0
Race (Reported)                                                 0
Age                                                             0
Height (cm)                                                     0
Weight (kg)                                                     0
Diabetes                                                        0
Simvastatin (Zocor)                                             0
Amiodarone (Cordarone)                                          0
Target INR                                                      0
INR on Reported Therapeutic Dose of Warfarin                    0
Cyp2C9 genotypes                                                0
VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T    0
Therapeutic Dose of Warfarin                                    0
dtype: int64
```

```
# visualization of dataset
data_frame.hist(bins=60, figsize=(20,15))
```

```
array([[<Axes: title={'center': 'Height (cm)'}>,
        <Axes: title={'center': 'Weight (kg)'}>,
        <Axes: title={'center': 'Diabetes'}>],
       [<Axes: title={'center': 'Simvastatin (Zocor)'}>,
        <Axes: title={'center': 'Amiodarone (Cordarone)'}>,
        <Axes: title={'center': 'Target INR'}>],
       [<Axes: title={'center': 'INR on Reported Therapeutic Dose of
Warfarin'}>,
        <Axes: title={'center': 'Therapeutic Dose of Warfarin'}>,
        <Axes: >]], dtype=object)
```





Data visualization

```
# Creating lables for model
df_target = data_frame[['Therapeutic Dose of Warfarin']]
df_features = data_frame.drop(columns=['Therapeutic Dose of Warfarin'])
```

```
# Features without Target column
df_features.columns
```

```
Index(['Gender', 'Race (Reported)', 'Age', 'Height (cm)', 'Weight (kg)',
       'Diabetes', 'Simvastatin (Zocor)', 'Amiodarone (Cordarone)',
       'Target INR', 'INR on Reported Therapeutic Dose of Warfarin',
       'Cyp2C9 genotypes',
       'VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T'],
      dtype='object')
```

```
# Target column
df_target.columns
```

```
Index(['Therapeutic Dose of Warfarin'], dtype='object')
```

```
# Spliting the data into train and test
from sklearn.model_selection import train_test_split
split_data = train_test_split(df_features, df_target, test_size=0.2, random_state
X_train, X_test, y_train, y_test = split_data
X_train.head()
```

	Gender	Race (Reported)	Age	Height (cm)	Weight (kg)	Diabetes	Simvastatin (Zocor)	Amioda (Cordar
5437	female	White	70 - 79	160.020000	60.91	0.000000	0.000000	0.0
2979	male	Korean	20 - 29	178.000000	82.00	0.000000	0.000000	0.0
4743	female	Malay	60 - 69	168.047778	72.00	0.187024	0.146335	0.0
2668	male	White	70 - 79	175.260000	85.30	0.000000	0.000000	0.0
3264	male	Caucasian	80 -	168.047778	59.00	0.000000	0.146335	0.0

```
y_test.head()
```

	Therapeutic Dose of Warfarin
1477	17.50
5514	37.52
3243	13.72
5183	21.00
5107	14.00

```
# Transformation of categorical values to binary of train set
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
X_train['Race (Reported)'] = le.fit_transform(X_train['Race (Reported)'])
X_train['Gender'] = le.fit_transform(X_train['Gender'])
X_train['Age'] = le.fit_transform(X_train['Age'])
X_train['Cyp2C9 genotypes'] = le.fit_transform(X_train['Cyp2C9 genotypes'])
X_train['VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T'] = le
```

```
# MinMax normilization of train set
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler() ## define the transformer
scaler.fit(X_train) ## call .fit() method to calculate the min and max value for
x_train_normalized = scaler.transform(X_train)
x_train_normalized=pd.DataFrame(x_train_normalized)
x_train_normalized
```

	0	1	2	3	4	5	6	7	8	
0	0.0	0.95	0.750	0.455032	0.148820	0.000000	0.000000	0.000000	0.450471	0.264
1	1.0	0.65	0.125	0.688441	0.250361	0.000000	0.000000	0.000000	0.450471	0.315
2	0.0	0.70	0.625	0.559245	0.202215	0.187024	0.146335	0.066236	0.450471	0.249
3	1.0	0.95	0.750	0.652872	0.266249	0.000000	0.000000	0.000000	0.450471	0.320
4	1.0	0.30	0.875	0.559245	0.139624	0.000000	0.146335	0.000000	0.450471	0.295
...	...	...	...	...	...	...	...	...	...	
4555	1.0	0.60	0.375	0.596817	0.235917	0.187024	0.146335	0.066236	0.450471	0.295
4556	0.0	0.10	0.625	0.586925	0.474723	1.000000	1.000000	0.000000	0.428571	0.358
4557	0.0	0.60	0.625	0.375896	0.144439	0.187024	0.146335	0.066236	0.450471	0.107
4558	0.0	0.95	0.375	0.455032	0.144439	0.000000	0.000000	0.000000	0.450471	0.377
4559	0.0	0.60	0.625	0.428653	0.144439	0.187024	0.146335	0.066236	0.450471	0.413

4560 rows × 12 columns

```
# Transformation of categorical values to binary of test set
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
X_test['Race (Reported)'] = le.fit_transform(X_test['Race (Reported)'])
X_test['Gender'] = le.fit_transform(X_test['Gender'])
X_test['Age'] = le.fit_transform(X_test['Age'])
X_test['Cyp2C9 genotypes'] = le.fit_transform(X_test['Cyp2C9 genotypes'])
X_test['VKORC1 genotype: -1639 G>A (3673); chr16:31015190; rs9923231; C/T'] = le.
```

```
# MinMax normilization of train setfrom sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler() ## define the transformer
scaler.fit(X_test) ## call .fit() method to calculate the min and max value for e
x_test_normalized = scaler.transform(X_test)
x_test_normalized=pd.DataFrame(x_test_normalized)
x_test_normalized
```

	0	1	2	3	4	5	6	7	8
0	0.0	0.4375	0.142857	0.332917	0.241880	0.000000	0.000000	0.000000	0.562875
1	1.0	0.2500	0.714286	0.833333	0.339254	0.000000	0.000000	0.000000	0.562875
2	0.0	0.3125	0.571429	0.465023	0.297167	0.000000	0.146335	1.000000	0.562875
3	1.0	1.0000	0.285714	0.546250	0.290256	0.000000	0.000000	1.000000	0.562875
4	0.0	1.0000	0.857143	0.300000	0.345543	0.000000	0.000000	0.000000	0.562875
...	...	...	...	...	...	...	...	...	...
1135	1.0	0.6875	0.285714	0.366667	0.172771	0.187024	0.146335	0.066236	0.562875
1136	1.0	0.3125	0.142857	0.465023	0.200415	0.000000	0.146335	0.000000	0.562875
1137	1.0	1.0000	0.285714	0.637500	0.373186	0.000000	1.000000	0.000000	0.545455
1138	1.0	0.6875	0.571429	0.445833	0.124395	0.187024	0.146335	0.066236	0.562875
1139	1.0	0.6875	0.571429	0.662500	0.262612	0.187024	0.146335	0.066236	0.562875

1140 rows × 12 columns

```
# Binary classification dataset by cutting the target values into two categories
y_train[y_train<=30] = 0
y_train[y_train>30] = 1
```

```
y_train
```

	Therapeutic Dose of Warfarin
5437	0.0
2979	1.0
4743	0.0
2668	0.0
3264	1.0
...	...
1180	0.0
3441	1.0
1344	0.0
4623	0.0
1289	0.0

4560 rows × 1 columns

```
# Display unique values of the 'Therapeutic Dose of Warfarin' column in train set
print(y_train["Therapeutic Dose of Warfarin"].unique())
```

```
[0. 1.]
```

```
y_test[y_test<=30] = 0
y_test[y_test>30] = 1
```

```
y_test
```

	Therapeutic Dose of Warfarin
1477	0.0
5514	1.0
3243	0.0
5183	0.0
5107	0.0
...	...
924	0.0
3317	0.0
1846	0.0
1316	0.0
1064	0.0

1140 rows × 1 columns

```
# Display unique values of the 'Therapeutic Dose of Warfarin' column in test set
print(y_test["Therapeutic Dose of Warfarin"].unique())
```

```
[0. 1.]
```

Model training

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# Define the decision tree model
decision_tree_model = DecisionTreeClassifier(max_depth=3)

# Fit the model on the normalized training data
decision_tree_model.fit(x_train_normalized, y_train)

# Predict using the trained model on the normalized test data
decision_tree_model.predict(x_test_normalized)
```

```
array([1., 1., 0., ..., 1., 0., 1.])
```

```
# calculating score
decision_tree_model.score(x_train_normalized,y_train)
```

```
0.7032894736842106
```

```
# Decision tree performance oon test data
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_au

# Predict using the decision tree model on the normalized test data
decision_tree_model_pred = decision_tree_model.predict(x_test_normalized)

# Calculate accuracy
decision_tree_model_acc = accuracy_score(y_test, decision_tree_model_pred)

# Calculate precision
decision_tree_model_prec = precision_score(y_test, decision_tree_model_pred)

# Calculate recall
decision_tree_model_recall = recall_score(y_test, decision_tree_model_pred)

# Calculate ROC AUC score
decision_tree_model_roc = roc_auc_score(y_test, decision_tree_model_pred)

# Calculate F1 score
decision_tree_model_f1 = f1_score(y_test, decision_tree_model_pred)

# Print performance metrics
print(decision_tree_model_acc)
print(decision_tree_model_prec)
print(decision_tree_model_recall)
print(decision_tree_model_roc)
print(decision_tree_model_f1)
```

```
0.637719298245614
0.5602165087956699
0.8247011952191236
0.657648403252195
0.6672038678485093
```

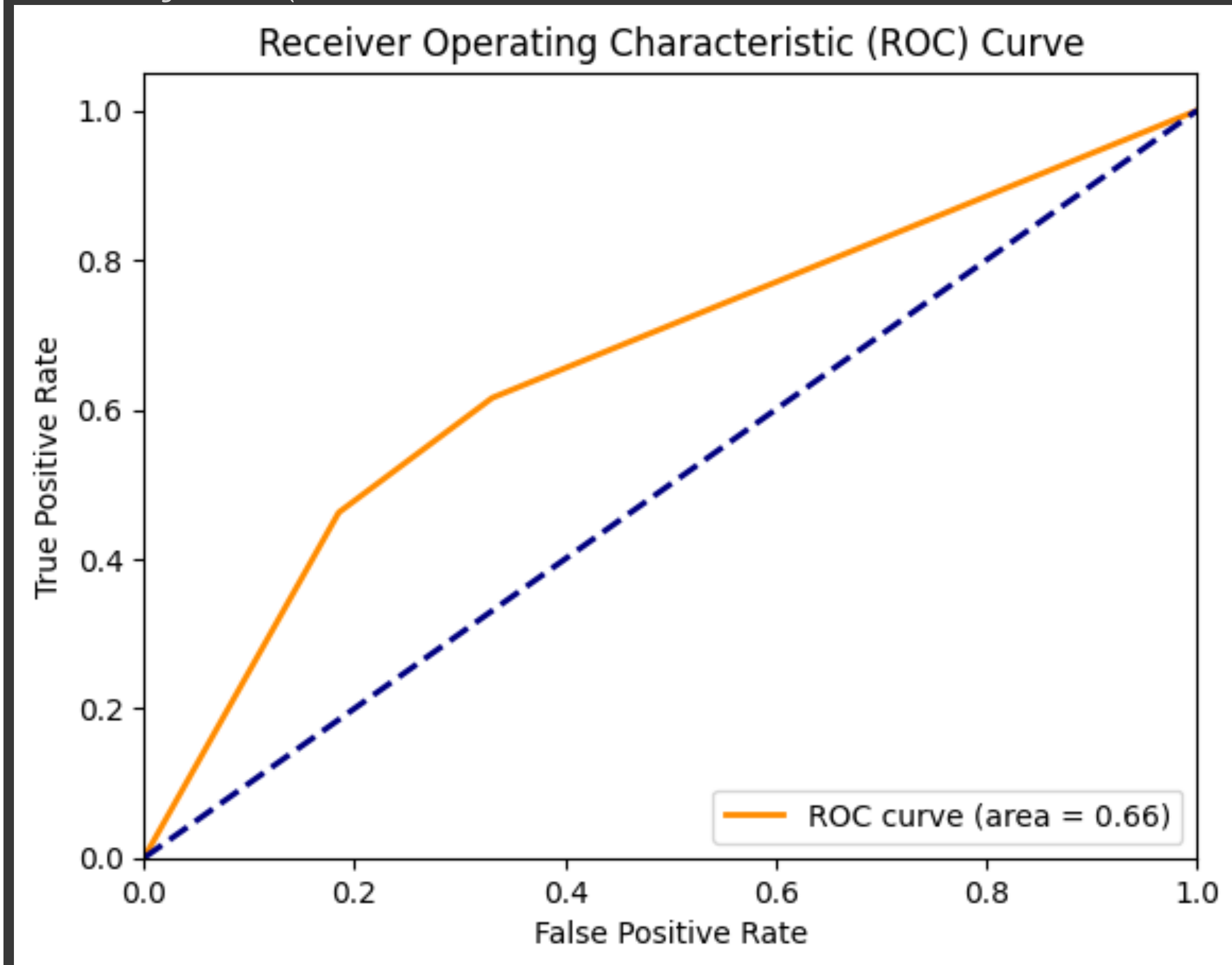
```
!pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dis
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3
```

```
from sklearn.metrics import roc_curve, auc
```

```
import matplotlib.pyplot as plt

# Assuming decision_tree_model is your classifier and X_test, y_test are your tes
y_score = decision_tree_model.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % r
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

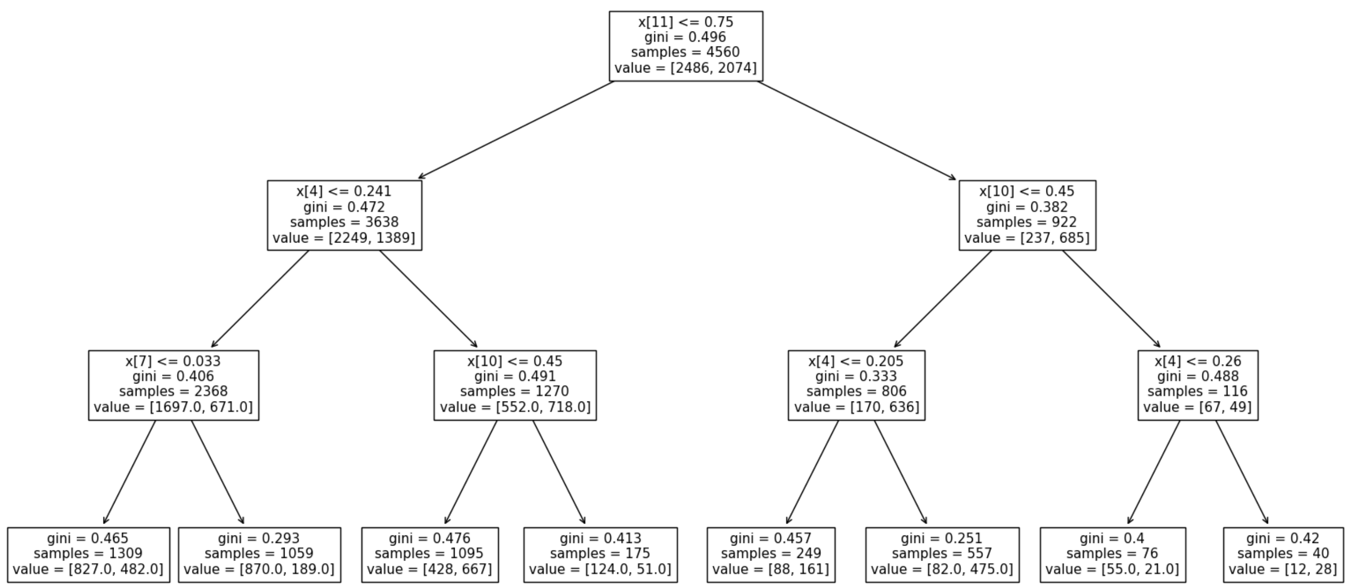
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X ha
  warnings.warn(
```




```
# visualization of decision tree
import matplotlib.pyplot as plt
from sklearn import tree

fig = plt.figure(figsize=(20, 10))
tree.plot_tree(decision_tree_model)
```

```
[Text(0.5, 0.875, 'x[11] <= 0.75\ngini = 0.496\nsamples = 4560\nvalue =
[2486, 2074]'),
 Text(0.25, 0.625, 'x[4] <= 0.241\ngini = 0.472\nsamples = 3638\nvalue =
[2249, 1389]'),
 Text(0.125, 0.375, 'x[7] <= 0.033\ngini = 0.406\nsamples = 2368\nvalue =
[1697.0, 671.0]'),
 Text(0.0625, 0.125, 'gini = 0.465\nsamples = 1309\nvalue = [827.0, 482.0]'),
 Text(0.1875, 0.125, 'gini = 0.293\nsamples = 1059\nvalue = [870.0, 189.0]'),
 Text(0.375, 0.375, 'x[10] <= 0.45\ngini = 0.491\nsamples = 1270\nvalue =
```

```
[552.0, 718.0]'),
 Text(0.3125, 0.125, 'gini = 0.476\nsamples = 1095\nvalue = [428, 667]'),
 Text(0.4375, 0.125, 'gini = 0.413\nsamples = 175\nvalue = [124.0, 51.0]'),
 Text(0.75, 0.625, 'x[10] <= 0.45\ngini = 0.382\nsamples = 922\nvalue = [237,
685]'),
 Text(0.625, 0.375, 'x[4] <= 0.205\ngini = 0.333\nsamples = 806\nvalue =
[170, 636]'),
 Text(0.5625, 0.125, 'gini = 0.457\nsamples = 249\nvalue = [88, 161]'),
 Text(0.6875, 0.125, 'gini = 0.251\nsamples = 557\nvalue = [82.0, 475.0]'),
 Text(0.875, 0.375, 'x[4] <= 0.26\ngini = 0.488\nsamples = 116\nvalue = [67,
49]'),
 Text(0.8125, 0.125, 'gini = 0.4\nsamples = 76\nvalue = [55.0, 21.0]'),
 Text(0.9375, 0.125, 'gini = 0.42\nsamples = 40\nvalue = [12, 28]')]
```



```
import joblib
model_file_path = "/content/drive/My Drive/decision_tree_model.pkl"
#joblib.dump(decision_tree_model, model_file_path)
```

```
# Logistic regression
from sklearn.linear_model import LogisticRegression

# Create a logistic regression model
logistic_model = LogisticRegression(penalty='l2', C=1, random_state=0)

# Fit the model to the normalized training data
logistic_model.fit(x_train_normalized, y_train)

# Make predictions on the normalized test data
logistic_predictions = logistic_model.predict(x_test_normalized)

# Print the predictions
print(logistic_predictions)
```

```
[0. 1. 0. ... 1. 0. 0.]
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
```

```
logistic_model.score(x_train_normalized,y_train)
```

```
0.7247807017543859
```

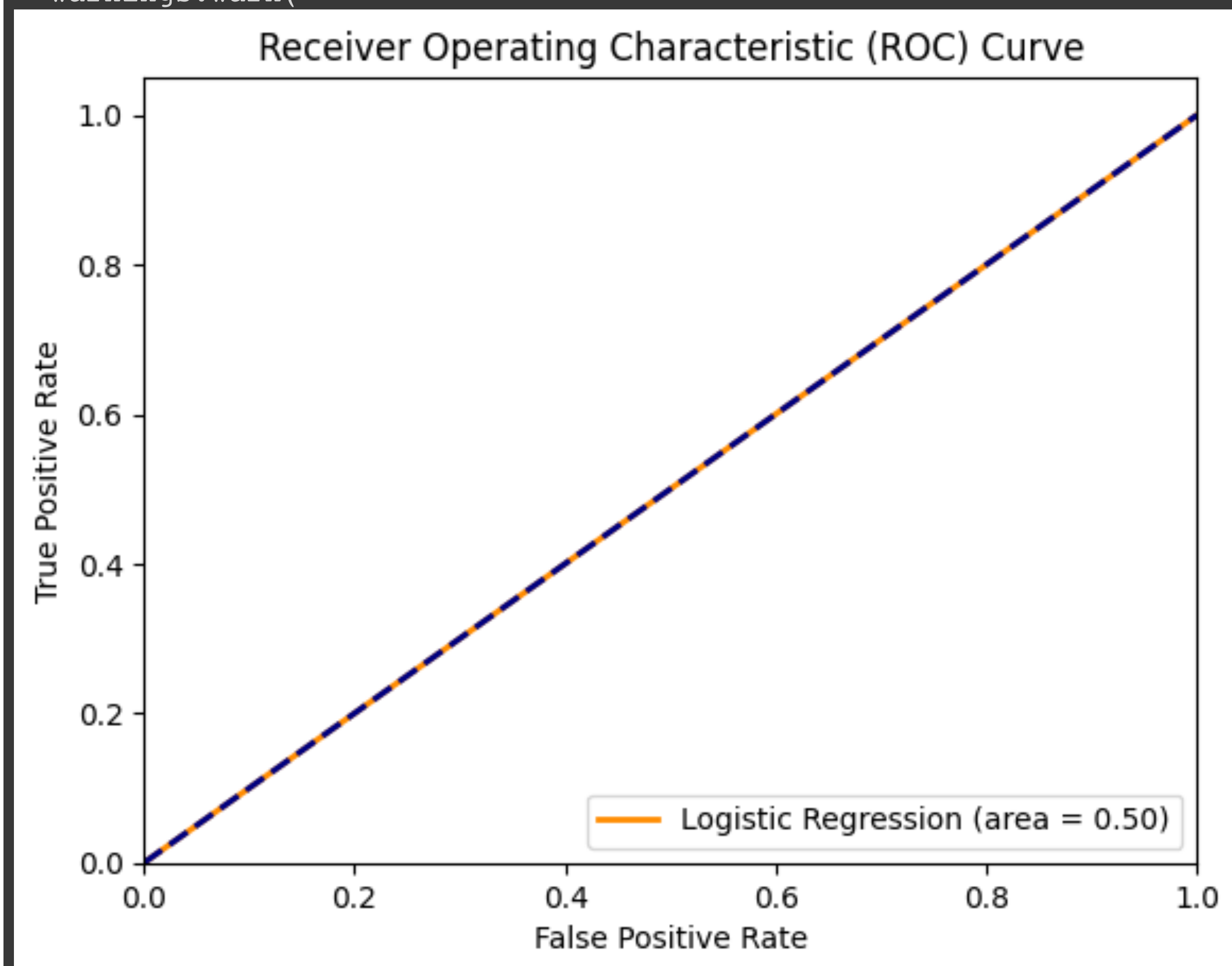
```
# Logistic regression - performance on the test data
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_au

# Predicting the target values for the test data using the logistic regression mc
logistic_pred = logistic_model.predict(x_test_normalized)

# Calculating the accuracy score
logistic_acc = accuracy_score(y_test, logistic_pred)

# Calculating the precision score
logistic_prec = precision_score(y_test, logistic_pred)

# Calculating the recall score
logistic_recall = recall_score(y_test, logistic_pred)

# Calculating the ROC AUC score
logistic_roc = roc_auc_score(y_test, logistic_pred)

# Calculating the F1 score
logistic_f1 = f1_score(y_test, logistic_pred)

# Printing the calculated performance metrics
print(logistic_acc)
print(logistic_prec)
print(logistic_recall)
print(logistic_roc)
print(logistic_f1)
```

```
0.7035087719298245
0.6238670694864048
0.8227091633466136
0.7162135158425857
0.7096219931271478
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming logistic_model is your classifier and X_test, y_test are your test dat
y_score = logistic_model.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure()
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label='Logistic Regression (area = %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```




```
import joblib
model_file_path = "/content/drive/My Drive/logistic_model.pkl"
joblib.dump(logistic_model, model_file_path)
```

```
['/content/drive/My Drive/logistic_model.pkl']
```

```
# Support Vector Machine
from sklearn.svm import SVC

# Define SVM model with C=1
svm_model = SVC(C=1)

# Fit SVM model to the normalized training data
svm_model.fit(x_train_normalized, y_train)

# Predict target values for the normalized test data
svm_model.predict(x_test_normalized)

# Compute and print the accuracy score of the SVM model on the training data
svm_model.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
0.7449561403508772
```

```
import joblib
model_file_path = "/content/drive/My Drive/svm_model.pkl"
joblib.dump(svm_model, model_file_path)
```

```
['/content/drive/My Drive/svm_model.pkl']
```

```
# linear kernel
svm_model_linear = SVC(kernel="linear", degree=3, C=5)

# Fit SVM model with linear kernel to the normalized training data
svm_model_linear.fit(x_train_normalized, y_train)

# Predict target values for the normalized test data
svm_model_linear.predict(x_test_normalized)

# Compute and print the accuracy score of the SVM model with linear kernel on the
svm_model_linear.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
0.7243421052631579
```

```
import joblib
model_file_path = "/content/drive/My Drive/svm_model_linear.pkl"
joblib.dump(svm_model_linear, model_file_path)
```

```
['/content/drive/My Drive/svm_model_linear.pkl']
```

```
# RBF kernel SVM model
svm_model_rbf = SVC(kernel="rbf", C=5)

# Train the SVM model with RBF kernel using the normalized training data
svm_model_rbf.fit(x_train_normalized, y_train)

# Predict target values for the normalized test data
svm_model_rbf.predict(x_test_normalized)

# Compute and print the accuracy score of the SVM model with RBF kernel on the tr
svm_model_rbf.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
0.756578947368421
```

```
import joblib
model_file_path = "/content/drive/My Drive/svm_model_rbf.pkl"
joblib.dump(svm_model_rbf, model_file_path)
```

```
['/content/drive/My Drive/svm_model_rbf.pkl']
```

```
# Sigmoid kernel SVM model with degree included
svm_model_sigmoid = SVC(kernel="sigmoid", degree=3, C=5)

# Train the SVM model with Sigmoid kernel using the normalized training data
svm_model_sigmoid.fit(x_train_normalized, y_train)

# Predict target values for the normalized test data
svm_model_sigmoid.predict(x_test_normalized)

# Compute and print the accuracy score of the SVM model with Sigmoid kernel on th
svm_model_sigmoid.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
0.493859649122807
```

```
import joblib
model_file_path = "/content/drive/My Drive/svm_model_sigmoid.pkl"
joblib.dump(svm_model_sigmoid, model_file_path)
```

```
['/content/drive/My Drive/svm_model_sigmoid.pkl']
```

```
# Polynomial kernel SVM model
svm_model_polynomial = SVC(kernel="poly", degree=3, C=5)

# Train the SVM model with Polynomial kernel using the normalized training data
svm_model_polynomial.fit(x_train_normalized, y_train)

# Predict target values for the normalized test data
svm_model_polynomial.predict(x_test_normalized)

# Compute and print the accuracy score of the SVM model with Polynomial kernel or
svm_model_polynomial.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
0.7526315789473684
```

```
import joblib
model_file_path = "/content/drive/My Drive/svm_model_polynomial.pkl"
joblib.dump(svm_model_polynomial, model_file_path)
```

```
['/content/drive/My Drive/svm_model_polynomial.pkl']
```

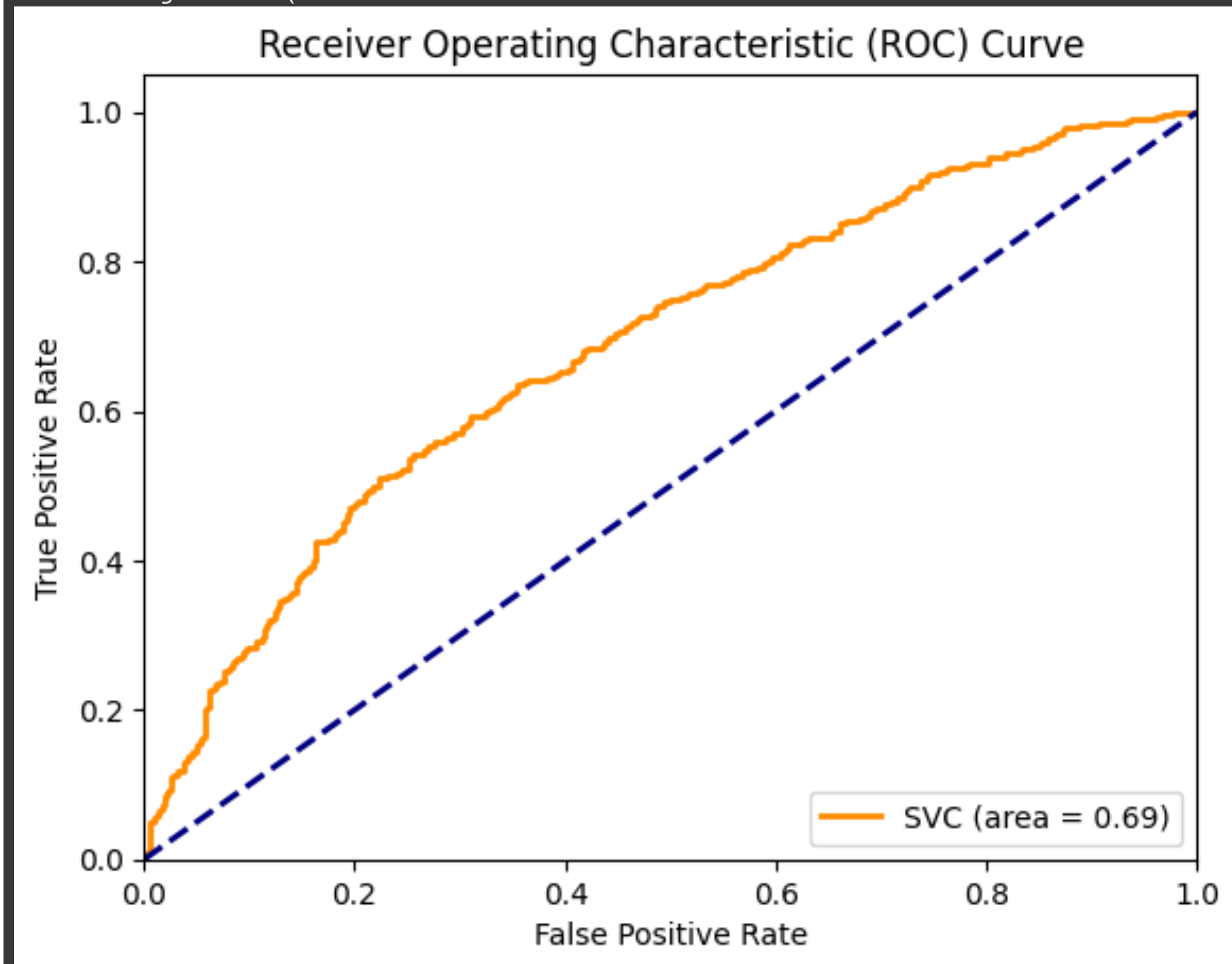
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_au

# SVM Model Evaluation
svm_model_pred = svm_model_polynomial.predict(x_test_normalized)
svm_acc = accuracy_score(y_test, svm_model_pred)
svm_prec = precision_score(y_test, svm_model_pred)
svm_recall = recall_score(y_test, svm_model_pred)
svm_roc = roc_auc_score(y_test, svm_model_pred)
svm_f1 = f1_score(y_test, svm_model_pred)
print(svm_acc)
print(svm_prec)
print(svm_recall)
print(svm_roc)
print(svm_f1)
```

```
0.718421052631579
0.661319073083779
0.7390438247011952
0.7206190910339833
0.6980244590780809
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming svm_model_polynomial is your classifier and X_test, y_test are your te
y_score = svm_model_polynomial.decision_function(X_test)
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='SVC (area = %0.2f)' % roc_auc
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```



```
# Importing the K Nearest-neighbors classifier from sklearn
from sklearn.neighbors import KNeighborsClassifier

# Initializing the K Nearest-neighbors model with 5 neighbors and the Minkowski c
knn_model = KNeighborsClassifier(n_neighbors=5, metric="minkowski")

# Fitting the K Nearest-neighbors model on the normalized training data
knn_model.fit(x_train_normalized, y_train)

# Predicting the target values for the normalized test data using the trained K N
knn_model.predict(x_test_normalized)

# Calculating the accuracy score of the K Nearest-neighbors model on the normaliz
knn_model.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:
  return self._fit(X, y)
0.7916666666666666
```

```
import joblib
model_file_path = "/content/drive/My Drive/knn_model.pkl"
joblib.dump(knn_model, model_file_path)
```

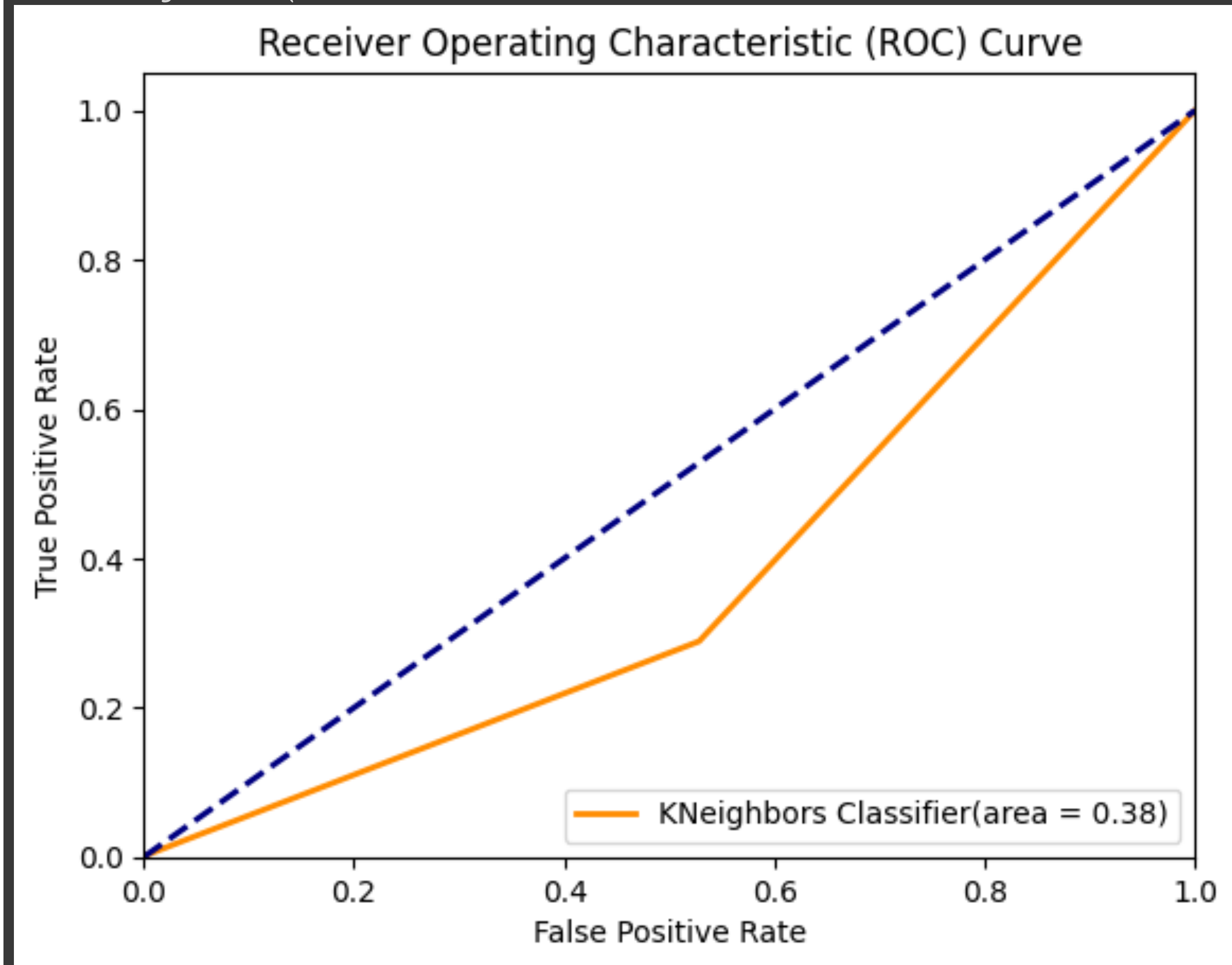
```
['/content/drive/My Drive/knn_model.pkl']
```

```
# KNN - performance on the test data
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_au

# KNN Model Evaluation
knn_model_pred = knn_model.predict(x_test_normalized)
knn_acc = accuracy_score(y_test, knn_model_pred)
knn_prec = precision_score(y_test, knn_model_pred)
knn_recall = recall_score(y_test, knn_model_pred)
knn_roc = roc_auc_score(y_test, knn_model_pred)
knn_f1 = f1_score(y_test, knn_model_pred)

print(knn_acc)
print(knn_prec)
print(knn_recall)
print(knn_roc)
print(knn_f1)
```

```
0.7149122807017544
0.6654205607476635
0.7091633466135459
0.714299541645331
0.686595949855352
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming knn_model is your classifier and X_test, y_test are your test data
y_score = knn_model.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='KNeighbors Classifier(area =
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```



```
# Importing the RandomForestClassifier from sklearn.ensemble module
from sklearn.ensemble import RandomForestClassifier

# Creating a RandomForestClassifier model with specified parameters
random_forest_model = RandomForestClassifier(n_estimators=100, max_leaf_nodes=18)

# Training the RandomForestClassifier model on the normalized training data
random_forest_model.fit(x_train_normalized, y_train)

# Predicting the target labels for the normalized test data using the trained moc
random_forest_model.predict(x_test_normalized)

# Calculating the accuracy score of the RandomForestClassifier model on the norma
random_forest_model.score(x_train_normalized, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1474: DataConversionW
  return fit_method(estimator, *args, **kwargs)
0.7421052631578947
```

```
import joblib
model_file_path = "/content/drive/My Drive/random_forest_model.pkl"
joblib.dump(random_forest_model, model_file_path)
```

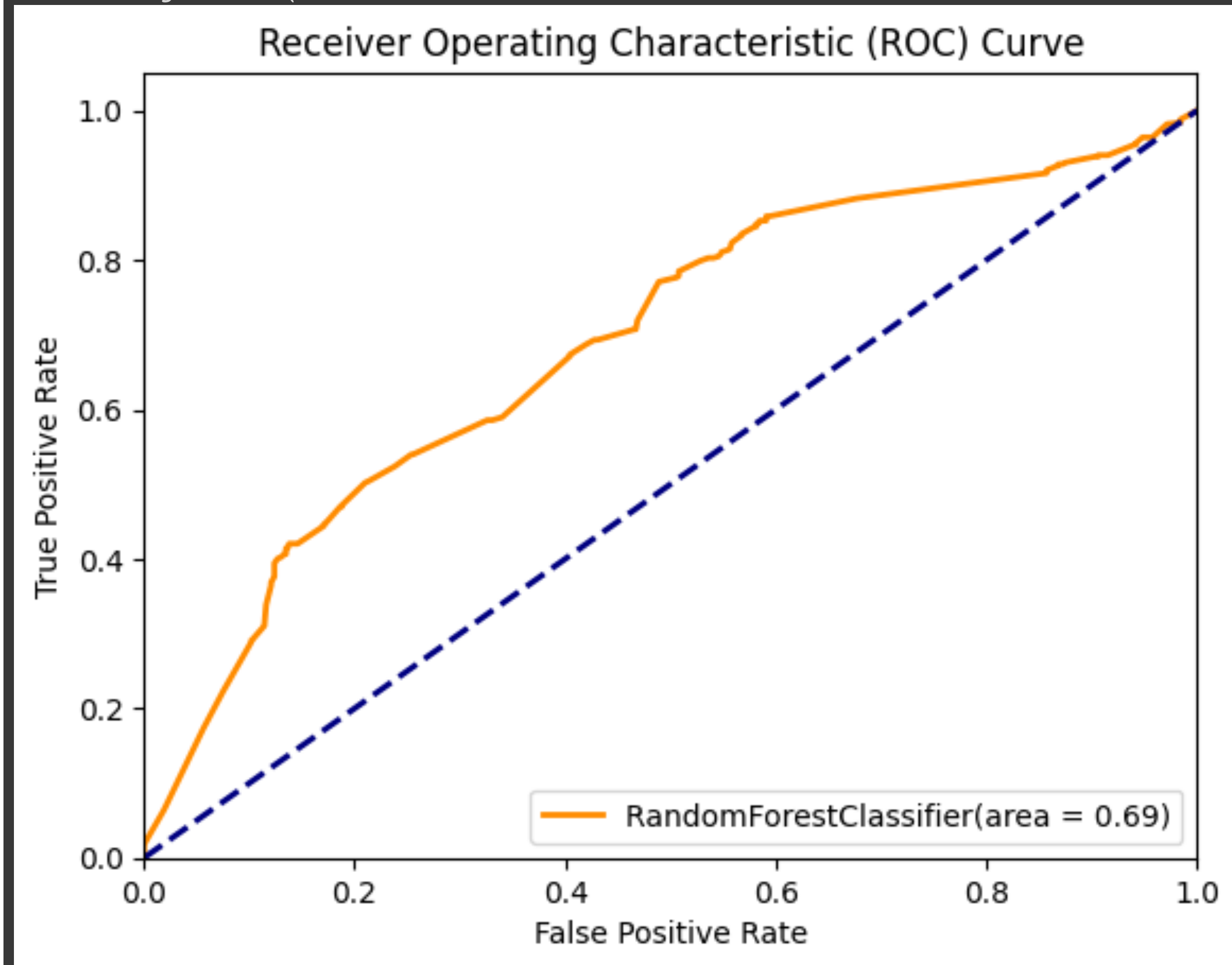
```
['/content/drive/My Drive/random_forest_model.pkl']
```

```
# Random Forest - performance on the test data
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_au
random_forest_pred = random_forest_model.predict(x_test_normalized)
random_forest_acc = accuracy_score(y_test, random_forest_pred )
random_forest_prec = precision_score(y_test, random_forest_pred )
random_forest_recall = recall_score(y_test, random_forest_pred )
random_forest_roc = roc_auc_score(y_test, random_forest_pred )
random_forest_f1 = f1_score(y_test, random_forest_pred )
print(random_forest_acc)
print(random_forest_prec)
print(random_forest_recall)
print(random_forest_roc)
print(random_forest_f1)
```

```
0.7140350877192982
0.6461794019933554
0.7749003984063745
0.7205222995166669
0.7047101449275363
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming random_forest_model is your classifier and X_test, y_test are your tes
y_score = random_forest_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='RandomForestClassifier(area =
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```




```
# Softmax Regression classifier
from sklearn.linear_model import LogisticRegression

softmax_regression = LogisticRegression(multi_class='multinomial', solver='lbfgs'
softmax_regression.fit(x_train_normalized, y_train)

# Get the predicted labels for training instances
y_prediction_labels = softmax_regression.predict(x_train_normalized)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
```

```
y_prediction_labels
```

```
array([0., 1., 0., ..., 0., 1., 0.])
```

```
# Get the predicted probabilities of both classes for training instances
y_prediction_probs = softmax_regression.predict_proba(x_train_normalized)
y_prediction_probs
```

```
array([[0.91584586, 0.08415414],
       [0.25960261, 0.74039739],
       [0.63893618, 0.36106382],
       ...,
       [0.84732268, 0.15267732],
       [0.40201416, 0.59798584],
       [0.71385464, 0.28614536]])
```

```
# Accuracy
import sklearn
sklearn.metrics.accuracy_score(y_train,y_prediction_labels)
```

```
0.7267543859649123
```

```
# Build multi-layer neural network for binary classification
# Practice 2.1: Load training data
import pandas as pd
from sklearn.utils import shuffle

train_data = X_train
train_data.columns = ['Label'] + [f'F{i}' for i in range(1, 12)]
train_data = shuffle(train_data)  # Shuffle data
train_data
```

	Label	F1	F2	F3	F4	F5	F6	F7	F8	F9
106	1	19	7	172.720000	72.6	0.187024	0.000000	0.000000	2.500000	1.800000
3544	1	6	7	179.000000	80.0	0.000000	0.000000	0.000000	2.538324	3.020000
2029	1	19	5	166.878000	79.3	0.187024	0.146335	0.066236	2.538324	2.364438
142	0	19	7	157.480000	49.9	0.187024	0.000000	0.000000	2.500000	2.400000
260	1	19	6	187.960000	83.5	0.187024	0.000000	0.000000	2.500000	2.000000
...	...	...	...	...	...	...	...	...	...	...
4380	0	6	6	168.047778	97.0	0.187024	0.146335	0.066236	2.538324	1.950000
5222	1	19	6	180.009800	71.0	0.000000	0.000000	0.000000	2.538324	2.364438
1738	1	19	6	173.736000	84.1	1.000000	1.000000	0.000000	2.500000	2.800000
359	0	18	4	170.180000	67.1	0.187024	0.000000	0.000000	3.000000	2.200000
2935	0	13	5	168.047778	56.0	0.000000	0.000000	0.000000	2.538324	1.720000

4560 rows × 12 columns

```python
import pandas as pd

test_data = x_test_normalized
test_data.columns = ['Label'] + [f'F{i}' for i in range(1, 12)]
test_data
```

	Label	F1	F2	F3	F4	F5	F6	F7	F
0	0.0	0.4375	0.142857	0.332917	0.241880	0.000000	0.000000	0.000000	0.56287
1	1.0	0.2500	0.714286	0.833333	0.339254	0.000000	0.000000	0.000000	0.56287
2	0.0	0.3125	0.571429	0.465023	0.297167	0.000000	0.146335	1.000000	0.56287
3	1.0	1.0000	0.285714	0.546250	0.290256	0.000000	0.000000	1.000000	0.56287
4	0.0	1.0000	0.857143	0.300000	0.345543	0.000000	0.000000	0.000000	0.56287
...	...	...	...	...	...	...	...	...	.
1135	1.0	0.6875	0.285714	0.366667	0.172771	0.187024	0.146335	0.066236	0.56287
1136	1.0	0.3125	0.142857	0.465023	0.200415	0.000000	0.146335	0.000000	0.56287
1137	1.0	1.0000	0.285714	0.637500	0.373186	0.000000	1.000000	0.000000	0.54545
1138	1.0	0.6875	0.571429	0.445833	0.124395	0.187024	0.146335	0.066236	0.56287
1139	1.0	0.6875	0.571429	0.662500	0.262612	0.187024	0.146335	0.066236	0.56287

1140 rows × 12 columns

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(100, input_shape=(22,), activation='relu'))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 100)               2300

=================================================================
Total params: 2300 (8.98 KB)
Trainable params: 2300 (8.98 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(100, input_shape=(22,), activation='relu'),
    Dense(100, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 100)               2300

 dense_9 (Dense)             (None, 100)               10100

 dense_10 (Dense)            (None, 1)                 101

=================================================================
Total params: 12501 (48.83 KB)
Trainable params: 12501 (48.83 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Compile the model for backpropagation, and start training model on data
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(100, input_shape=(12,), activation='relu'),
    Dense(100, activation='relu'),
    Dense(1, activation='sigmoid')
])
# Compile model, use binary crossentropy, and stochastic gradient descent for opt
model.compile(loss='binary_crossentropy', optimizer='SGD', metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 100)               1300

 dense_12 (Dense)            (None, 100)               10100

 dense_13 (Dense)            (None, 1)                 101

=================================================================
Total params: 11501 (44.93 KB)
Trainable params: 11501 (44.93 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(X_train, y_train, epochs = 50)
```

```
Epoch 1/50
143/143 [==============================] - 5s 7ms/step - loss: 1.1637 - accur
Epoch 2/50
143/143 [==============================] - 1s 6ms/step - loss: 0.6468 - accur
Epoch 3/50
143/143 [==============================] - 1s 10ms/step - loss: 0.6447 - accu
Epoch 4/50
143/143 [==============================] - 1s 7ms/step - loss: 0.6422 - accur
Epoch 5/50
143/143 [==============================] - 1s 9ms/step - loss: 0.6389 - accur
Epoch 6/50
143/143 [==============================] - 1s 7ms/step - loss: 0.6368 - accur
Epoch 7/50
143/143 [==============================] - 1s 5ms/step - loss: 0.6357 - accur
```

```
Epoch 8/50
143/143 [==============================] - 1s 4ms/step - loss: 0.6337 - accur
Epoch 9/50
143/143 [==============================] - 2s 11ms/step - loss: 0.6350 - accu
Epoch 10/50
143/143 [==============================] - 1s 10ms/step - loss: 0.6352 - accu
Epoch 11/50
143/143 [==============================] - 2s 11ms/step - loss: 0.6318 - accu
Epoch 12/50
143/143 [==============================] - 1s 7ms/step - loss: 0.6303 - accur
Epoch 13/50
143/143 [==============================] - 1s 9ms/step - loss: 0.6317 - accur
Epoch 14/50
143/143 [==============================] - 1s 9ms/step - loss: 0.6301 - accur
Epoch 15/50
143/143 [==============================] - 1s 9ms/step - loss: 0.6287 - accur
Epoch 16/50
143/143 [==============================] - 1s 5ms/step - loss: 0.6270 - accur
Epoch 17/50
143/143 [==============================] - 1s 6ms/step - loss: 0.6241 - accur
Epoch 18/50
143/143 [==============================] - 1s 6ms/step - loss: 0.6228 - accur
Epoch 19/50
143/143 [==============================] - 1s 7ms/step - loss: 0.6219 - accur
Epoch 20/50
143/143 [==============================] - 1s 6ms/step - loss: 0.6259 - accur
Epoch 21/50
143/143 [==============================] - 1s 5ms/step - loss: 0.6224 - accur
Epoch 22/50
143/143 [==============================] - 1s 4ms/step - loss: 0.6215 - accur
Epoch 23/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6243 - accur
Epoch 24/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6241 - accur
Epoch 25/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6183 - accur
Epoch 26/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6215 - accur
Epoch 27/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6167 - accur
Epoch 28/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6211 - accur
Epoch 29/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6108 - accur
Epoch 30/50
143/143 [==============================] - 0s 2ms/step - loss: 0.6177 - accur
```

```
from tensorflow.keras.models import load_model
model.save("/content/drive/My Drive/tensor_keras.pkl")
#model_loaded = load_model("my_model.h5")
```

```
model.evaluate(X_train, y_train)
```

```
143/143 [==============================] - 0s 2ms/step - loss: 0.5957 - accur
[0.5956673622131348, 0.6780701875686646]
```

```
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
print("train_predictions: ", X_train) # Print train_predictions
```

```
143/143 [==============================] - 0s 2ms/step
36/36 [==============================] - 0s 2ms/step
train_predictions:       Label  F1  F2          F3      F4        F5
5437      0  19   6  160.020000   60.91  0.000000  0.000000  0.000000
2979      1  13   1  178.000000   82.00  0.000000  0.000000  0.000000
4743      0  14   5  168.047778   72.00  0.187024  0.146335  0.066236
2668      1  19   6  175.260000   85.30  0.000000  0.000000  0.000000
3264      1   6   7  168.047778   59.00  0.000000  0.146335  0.000000
...     ...  ..  ..         ...     ...       ...       ...       ...
1180      1  12   3  170.942000   79.00  0.187024  0.146335  0.066236
3441      0   2   5  170.180000  128.60  1.000000  1.000000  0.000000
1344      0  12   5  153.924000   60.00  0.187024  0.146335  0.066236
4623      0  19   3  160.020000   60.00  0.000000  0.000000  0.000000
1289      0  12   5  157.988000   60.00  0.187024  0.146335  0.066236

            F8        F9  F10  F11
5437  2.538324  2.200000    5    0
2979  2.538324  2.470000    0    0
4743  2.538324  2.120000    0    0
2668  2.538324  2.500000    5    0
3264  2.538324  2.364438    4    2
...        ...       ...  ...  ...
1180  2.538324  2.364438    0    0
3441  2.500000  2.700000    0    2
1344  2.538324  1.370000    0    0
4623  2.538324  2.800000    5    2
1289  2.538324  2.990000    0    0

[4560 rows x 12 columns]
```

```
import numpy as np

# Convert predicted probabilities to binary labels using a threshold of 0.5
train_prediction_labels = (train_predictions > 0.5).astype(int)
test_prediction_labels = (test_predictions > 0.5).astype(int)
```

```
from sklearn.metrics import accuracy_score

# Calculate and print the training accuracy
print("Training accuracy: ", accuracy_score(y_train, train_prediction_labels))

# Calculate and print the test accuracy
print("Test accuracy: ", accuracy_score(y_test, test_prediction_labels))
```

```
Training accuracy:  0.6780701754385965
Test accuracy:  0.6912280701754386
```

```
from sklearn.decomposition import PCA

# Instantiate PCA with desired number of components
pca = PCA(n_components=2)

# Fit PCA to training data and transform it in one step
X_reduced = pca.fit_transform(X_train)
```

```
# Inital Shape
X_train.shape
print("Initiial shape:",X_train.shape)
# Reduced Shape
X_reduced.shape
print("Reduce shape:",X_reduced.shape)
```

```
Initiial shape: (4560, 12)
Reduce shape: (4560, 2)
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Instantiate and train LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_reduced, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

```
import joblib
model_file_path = "/content/drive/My Drive/LinearDiscriminantAnalysis_model.pkl"
joblib.dump(knn_model, model_file_path)
```

```
['/content/drive/My Drive/LinearDiscriminantAnalysis_model.pkl']
```

```
# Score
lda.score(X_reduced, y_train)
```

```
0.6515350877192982
```

```
# Evalution Metrics
TabularData = {'Methods': ['Decision Tree', 'SVM', 'KNN', 'Random Forest', 'Logis
              'Accuracy': [decision_tree_model_acc, svm_acc, knn_acc, random_fore
              'Precision': [decision_tree_model_prec, svm_prec, knn_prec, random_
              'Recall': [decision_tree_model_recall, svm_recall, knn_recall, ranc
              'F1-score': [decision_tree_model_f1, svm_f1, knn_f1, random_forest_
              'AUC score': [decision_tree_model_roc, svm_roc, knn_roc, random_for

evaluation_metrics = pd.DataFrame(TabularData)
```

```
evaluation_metrics
```

	Methods	Accuracy	Precision	Recall	F1-score	AUC score
0	Decision Tree	0.637719	0.560217	0.824701	0.667204	0.657648
1	SVM	0.718421	0.661319	0.739044	0.698024	0.720619
2	KNN	0.714912	0.665421	0.709163	0.686596	0.714300
3	Random Forest	0.714035	0.646179	0.774900	0.704710	0.720522
4	Logistic regression	0.703509	0.623867	0.822709	0.709622	0.716214

```
# Cross Validation for Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
CV_scores_decision_tree_model = cross_val_score(estimator = decision_tree_model,
print("CV_scores: ", CV_scores_decision_tree_model)
```

```
CV_scores:  [0.67763158 0.67763158 0.67105263 0.65131579 0.68421053 0.7565789
 0.72368421 0.63157895 0.65131579 0.80921053 0.73684211 0.69736842
 0.67763158 0.71710526 0.72368421 0.65789474 0.68421053 0.625
 0.79605263 0.68421053 0.73026316 0.69078947 0.66447368 0.73026316
 0.65789474 0.72368421 0.69736842 0.63815789 0.71710526 0.69736842]
```

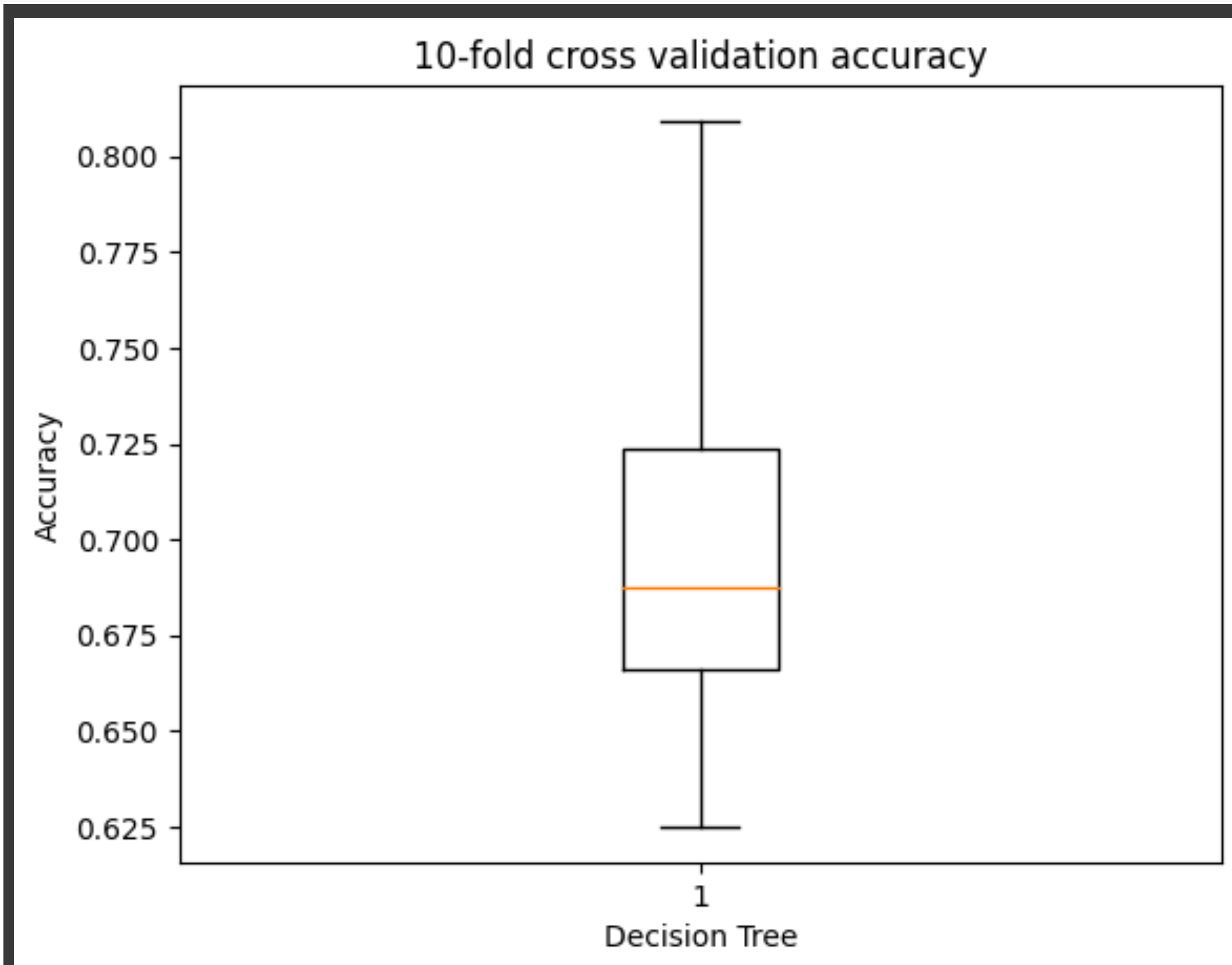
```
# Predicting on test data using the decision tree model
y_test_pred_dt = decision_tree_model.predict(x_test_normalized)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```

```
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scc

# Calculating accuracy
decision_tree_acc_cv = accuracy_score(y_test_pred_dt, y_test)

# Calculating precision
decision_tree_prec_cv = precision_score(y_test_pred_dt, y_test)

# Calculating recall
decision_tree_recall_cv = recall_score(y_test_pred_dt, y_test)

# Calculating F1 score
decision_tree_f1_cv = f1_score(y_test_pred_dt, y_test)

# Calculating ROC AUC score (assuming you have imported roc_auc_score)
from sklearn.metrics import roc_auc_score
decision_tree_roc_cv = roc_auc_score(y_test_pred_dt, y_test)

# Printing the calculated metrics
print("Accuracy: ", decision_tree_acc_cv)
print("Precision:", decision_tree_prec_cv)
print("Recall:", decision_tree_recall_cv)
print("F1 Score:", decision_tree_f1_cv)
print("ROC AUC Score:", decision_tree_roc_cv)
```

```
Accuracy:  0.637719298245614
Precision: 0.8247011952191236
Recall: 0.5602165087956699
F1 Score: 0.6672038678485093
ROC AUC Score: 0.6703825686122988
```

```
# Plotting the boxplot of cross-validation scores
plt.boxplot(CV_scores_decision_tree_model)

# Adding title to the plot
plt.title("10-fold cross validation accuracy")

# Adding label to x-axis
plt.xlabel("Decision Tree")

# Adding label to y-axis
plt.ylabel("Accuracy")

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




```
# Cross validation for Logictic regression
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
```

```
# Creating an instance of Logistic Regression model
logistic_model = LogisticRegression()

# Performing cross-validation
CV_scores_lr = cross_val_score(logistic_model, X_train, y_train, cv=10, scoring='

# Printing the cross-validation scores
print("CV_scores:", CV_scores_lr)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
```

```
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1300: Dat
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
```

```
# Accuracy, Precision, Recall and F1-score
# Predicting on test data using the random forest model
y_test_pred3 = random_forest_model.predict(x_test_normalized)

# Calculating and printing accuracy
print("Accuracy: ", accuracy_score(y_test_pred3, y_test))

# Calculating and printing precision
print("Precision:", precision_score(y_test_pred3.astype(int), y_test.astype(int))

# Calculating and printing recall
print("Recall:", recall_score(y_test_pred3.astype(int), y_test.astype(int)))

# Calculating and printing F1-score
print("F1 Score:", f1_score(y_test_pred3.astype(int), y_test.astype(int)))
```

```
Accuracy:  0.7140350877192982
Precision: 0.7749003984063745
Recall: 0.6461794019933554
F1 Score: 0.7047101449275363
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```

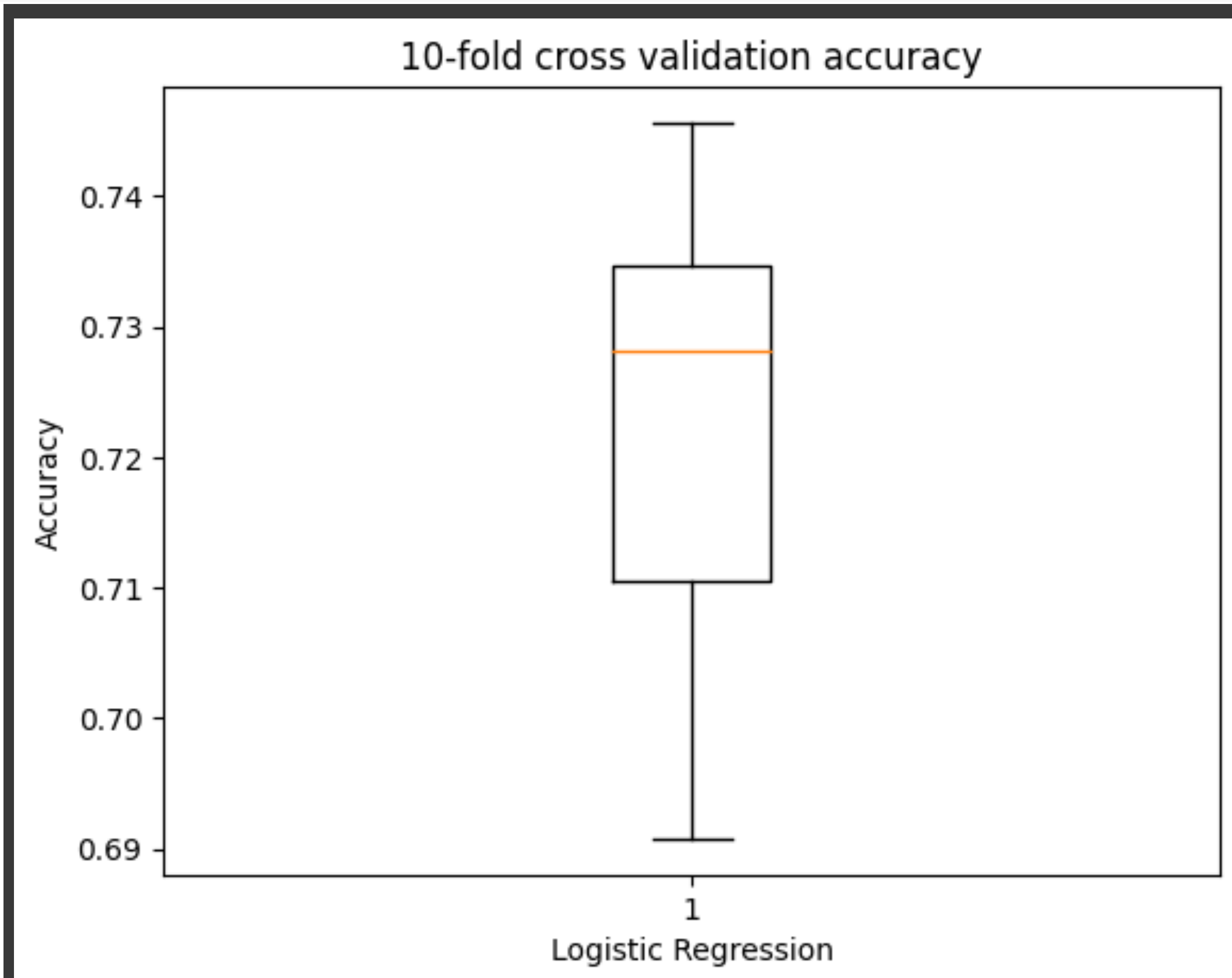
```
# Predicting on the test data using the random forest model
y_test_pred_lr = random_forest_model.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```

```
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score,recall_score,f1_score
# Calculating accuracy
logistic_model_acc_cv = metrics.accuracy_score(y_test_pred_lr,y_test)
# Calculating precision
logistic_model_prec_cv = precision_score(y_test_pred_lr, y_test)

# Calculating recall
logistic_model_recall_cv = recall_score(y_test_pred_lr, y_test)

# Calculating F1 score
logistic_model_f1_cv = f1_score(y_test_pred_lr, y_test)

# Calculating ROC AUC score
logistic_model_roc_cv = roc_auc_score(y_test_pred_lr, y_test)

# Printing the calculated metrics
print("Accuracy: ", logistic_model_acc_cv)
print("Precision:", logistic_model_prec_cv)
print("Recall:", logistic_model_recall_cv)
print("F1 Score:", logistic_model_f1_cv)
print("ROC AUC Score:", logistic_model_roc_cv)
```

```
Accuracy:  0.4850877192982456
Precision: 0.9203187250996016
Recall: 0.4578790882061447
F1 Score: 0.6115155526141628
ROC AUC Score: 0.5762677883778815
```

```
# Plotting the boxplot of cross-validation scores for logistic regression
plt.boxplot(CV_scores_lr)

# Adding title to the plot
plt.title("10-fold cross validation accuracy")

# Adding label to x-axis
plt.xlabel("Logistic Regression")

# Adding label to y-axis
plt.ylabel("Accuracy")

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



```
# Cross validation for Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

model1 = random_forest_model
CV_scores_random_forest_model = cross_val_score(estimator = model1, X = X_train,
print("CV_scores: ", CV_scores_random_forest_model)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1474: DataConversionW
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1474: DataConversionW
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1474: DataConversionW
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1474: DataConversionW
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1474: DataConversionW
  return fit_method(estimator, *args, **kwargs)
CV_scores:  [0.71820175 0.72807018 0.72697368 0.73135965 0.7247807 ]
```

```
# Predicting on the test data using the random forest model
y_test_pred_rf = random_forest_model.predict(X_test)
```

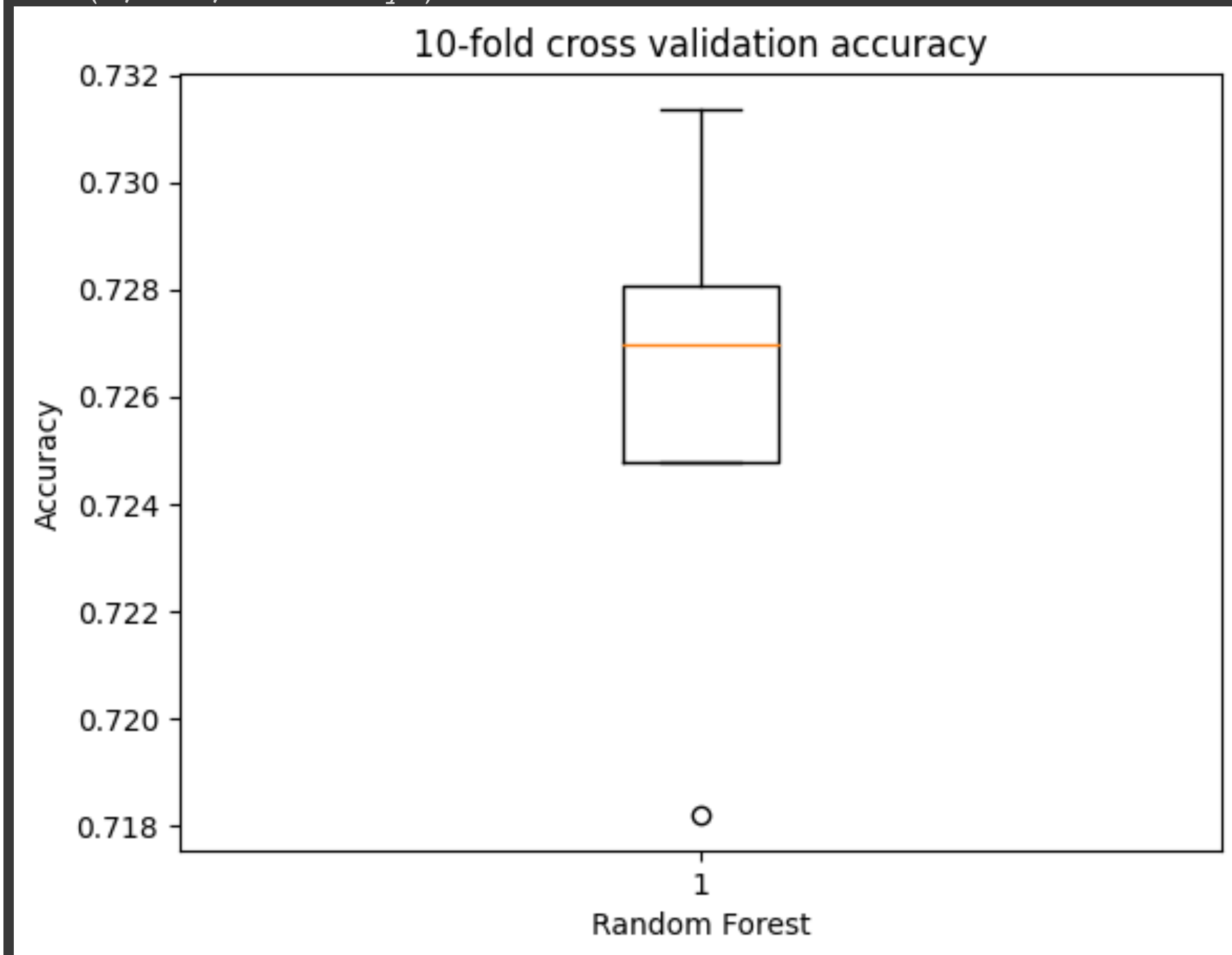
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X h
  warnings.warn(
```

```
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scc

# Calculating accuracy
random_forest_acc_cv = accuracy_score(y_test_pred_rf, y_test)

# Calculating precision
random_forest_prec_cv = precision_score(y_test_pred_rf, y_test)

# Calculating recall
random_forest_recall_cv = recall_score(y_test_pred_rf, y_test)

# Calculating F1 score
random_forest_f1_cv = f1_score(y_test_pred_rf, y_test)

# Calculating ROC AUC score (assuming you have imported roc_auc_score)
random_forest_roc_cv = roc_auc_score(y_test_pred_rf, y_test)

# Printing the calculated metrics
print("Accuracy: ", random_forest_acc_cv)
print("Precision:", random_forest_prec_cv)
print("Recall:", random_forest_recall_cv)
print("F1 Score:", random_forest_f1_cv)
print("ROC AUC Score:", random_forest_roc_cv)
```

```
Accuracy:  0.4850877192982456
Precision: 0.9203187250996016
Recall: 0.4578790882061447
F1 Score: 0.6115155526141628
ROC AUC Score: 0.5762677883778815
```

```
plt.boxplot(CV_scores_random_forest_model)
plt.title("10-fold cross validation accuracy")
plt.xlabel("Random Forest")
plt.ylabel("Accuracy")
```

```
Text(0, 0.5, 'Accuracy')
```




```
# Evalution Metrics
TabularData_cv = {'Methods': ['Decision Tree','Random Forest','Logistic Model'],
        'Accuracy': [decision_tree_acc_cv, random_forest_acc_cv, logistic_model_a
        'Precision':[decision_tree_prec_cv, random_forest_prec_cv, logistic_model
        'Recall':[decision_tree_recall_cv, random_forest_recall_cv, logistic_mode
        'F1-score':[decision_tree_f1_cv, random_forest_f1_cv, logistic_model_f1_c
        'AUC score':[decision_tree_roc_cv, random_forest_roc_cv, logistic_model_r

evaluation_metrics_cv =pd.DataFrame(TabularData_cv)
```

```
evaluation_metrics_cv
```

	Methods	Accuracy	Precision	Recall	F1-score	AUC score
0	Decision Tree	0.637719	0.824701	0.560217	0.667204	0.670383
1	Random Forest	0.485088	0.920319	0.457879	0.611516	0.576268
2	Logistic Model	0.485088	0.920319	0.457879	0.611516	0.576268

```
pip install gradio
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: aiofiles<24.0,>=22.0 in /usr/local/lib/python3
Requirement already satisfied: altair<6.0,>=4.2.0 in /usr/local/lib/python3.1
Requirement already satisfied: fastapi in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: ffmpy in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: gradio-client==0.16.3 in /usr/local/lib/python
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: huggingface-hub>=0.19.3 in /usr/local/lib/pyth
Requirement already satisfied: importlib-resources<7.0,>=1.3 in /usr/local/li
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: markupsafe~=2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: matplotlib~=3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: numpy~=1.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.10/
Requirement already satisfied: pillow<11.0,>=8.0 in /usr/local/lib/python3.10
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pydub in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: python-multipart>=0.0.9 in /usr/local/lib/pyth
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.10/
Requirement already satisfied: ruff>=0.2.2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python
Requirement already satisfied: tomlkit==0.12.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.10/
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/pytho
Requirement already satisfied: urllib3~=2.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: websockets<12.0,>=10.0 in /usr/local/lib/pytho
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: anyio in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-pack
```

```
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.10/dis
Requirement already satisfied: idna in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.10/d
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/pytho
Requirement already satisfied: pydantic-core==2.18.2 in /usr/local/lib/python
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.1
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: starlette<0.38.0,>=0.37.2 in /usr/local/lib/py
Requirement already satisfied: fastapi-cli>=0.0.2 in /usr/local/lib/python3.1
Requirement already satisfied: ujson!=4.0.2,!=4.1.0,!=4.2.0,!=4.3.0,!=5.0.0,!
Requirement already satisfied: email_validator>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: dnspython>=2.0.0 in /usr/local/lib/python3.10/
```