Question - 1 (Parkinson's disease prediction)

V TEJAS

CSE - B

3122 21 5001 116

UCS 2612 - MACHINE LEARNING LABORATORY

Lab Test - 1

Question - 1 (Parkinson's disease prediction)

Importing necessary libraries

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

Loading the dataset from GDrive

```
data = pd.read_csv('/content/drive/MyDrive/SEM-6/ML Lab/test-1/parkinsons.csv')
data.head()
```

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284

5 rows × 24 columns

```
data.describe()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(

count	195.000000	195.000000	195.000000	195.000000	195.0
mean	154.228641	197.104918	116.324631	0.006220	0.0
std	41.390065	91.491548	43.521413	0.004848	0.0
min	88.333000	102.145000	65.476000	0.001680	0.0
25%	117.572000	134.862500	84.291000	0.003460	0.0
50%	148.790000	175.829000	104.315000	0.004940	0.0
75%	182.769000	224.205500	140.018500	0.007365	0.0
max	260.105000	592.030000	239.170000	0.033160	0.0

8 rows × 23 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              195 non-null    object
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

```
data.isnull().sum()
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
```

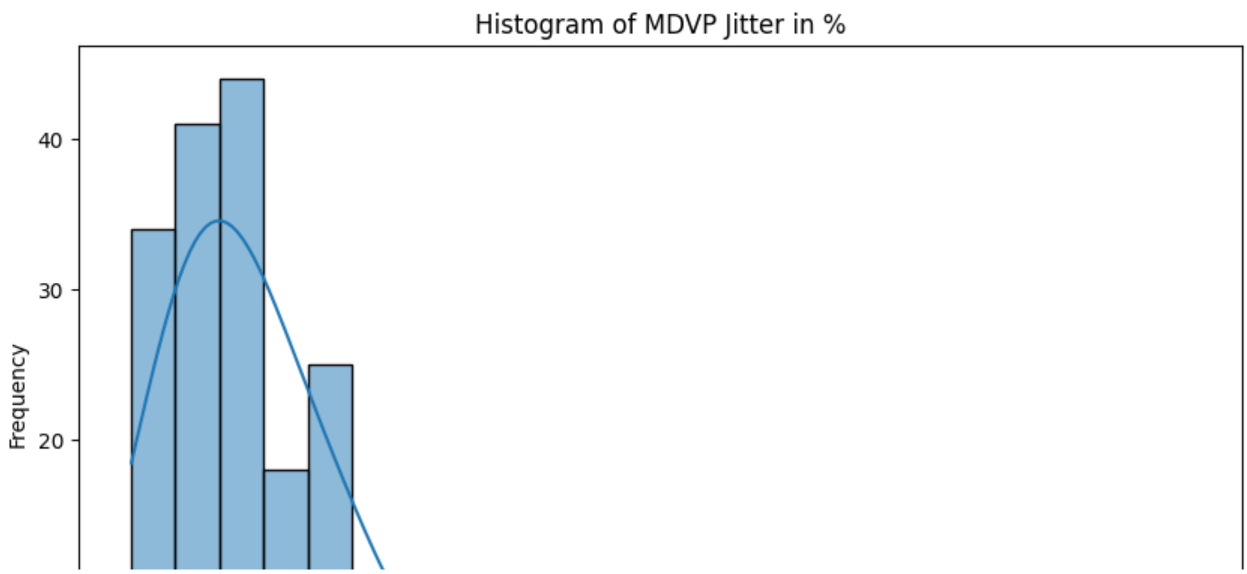
```
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

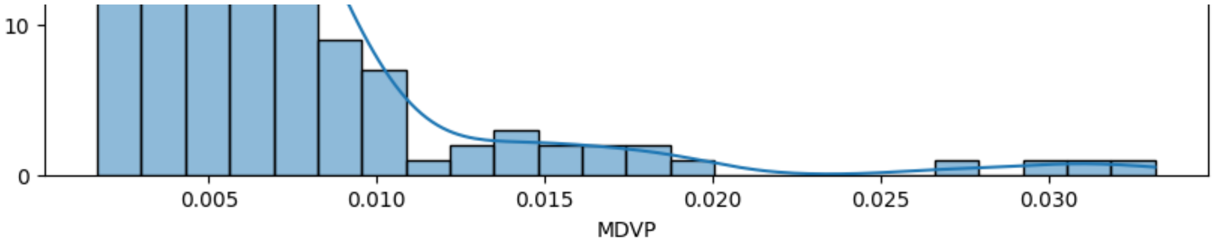
```
data.shape
```

```
(195, 24)
```

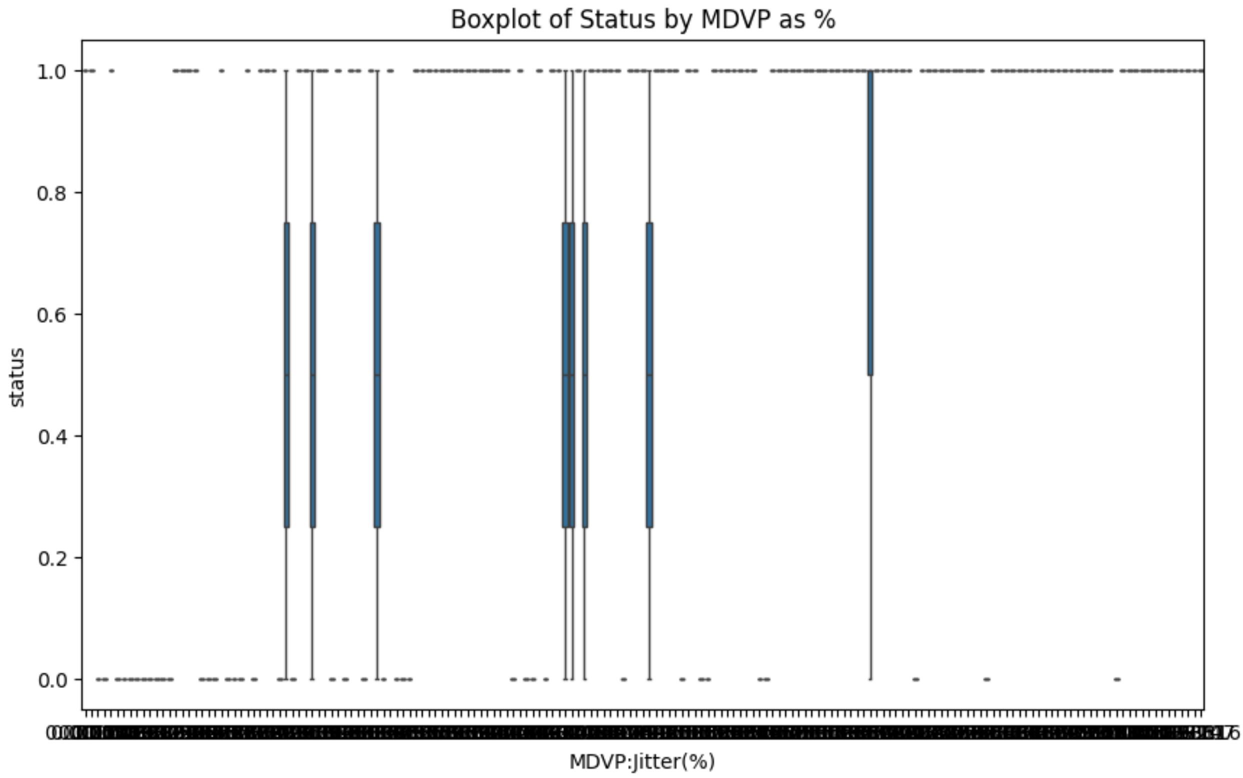
Exploratory Data Analysis

```
#Histogram

if 'MDVP:Jitter(%)' in data.columns:  # Check if the column exists
    plt.figure(figsize=(10, 6))
    sns.histplot(data['MDVP:Jitter(%)'], kde=True)
    plt.title('Histogram of MDVP Jitter in %')
    plt.xlabel('MDVP')
    plt.ylabel('Frequency')
    plt.show()
```



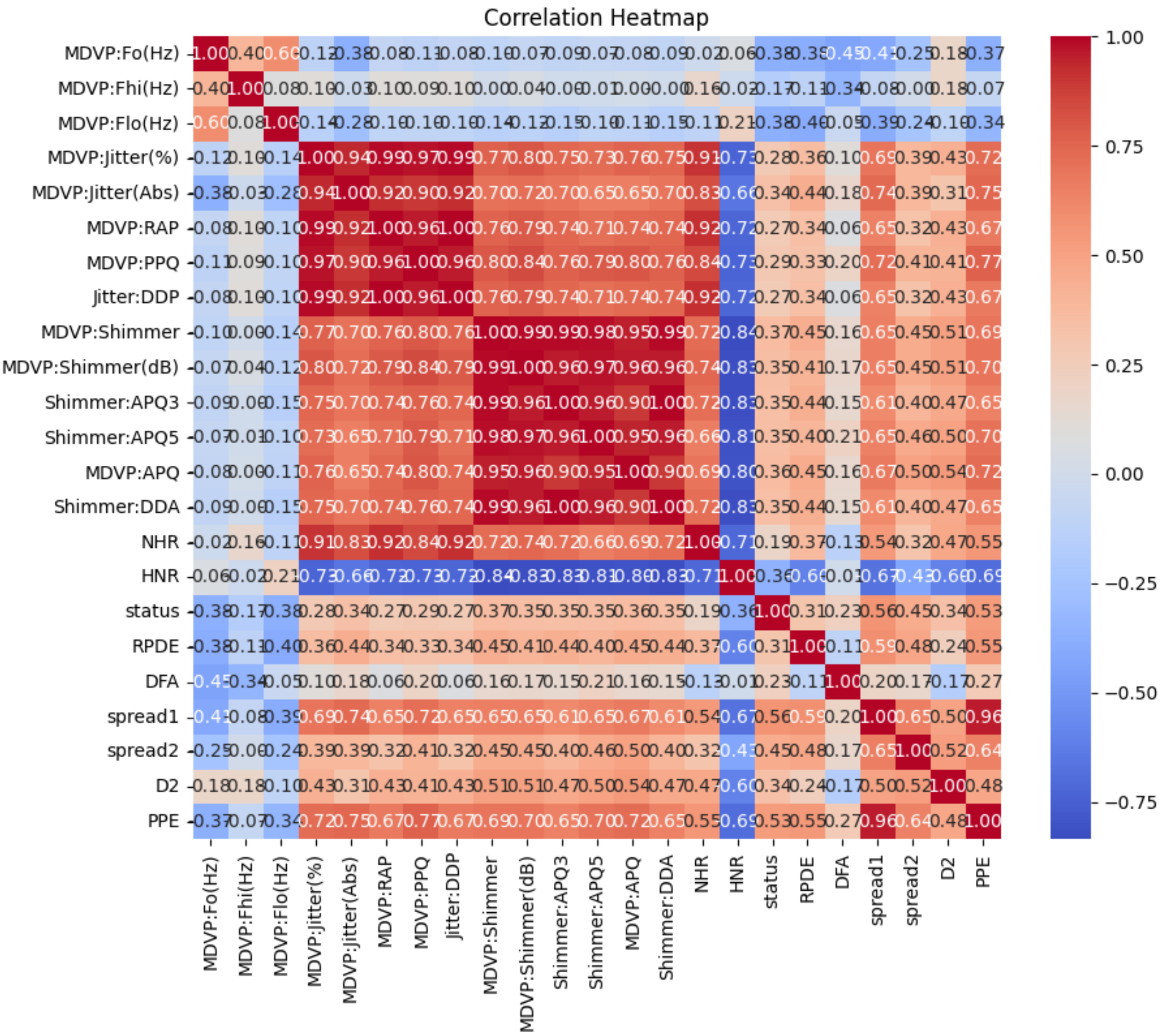



```
#BoxPlot

if 'MDVP:Jitter(%)' in data.columns and 'status' in data.columns:  # Ensure bot
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='MDVP:Jitter(%)', y='status', data=data)
    plt.title('Boxplot of Status by MDVP as %')
    plt.show()
```




```
#HeatMap
```

```
#Heatmap

numerical_data = data.select_dtypes(exclude=['object'])
corr = numerical_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



Pre-Processing

```
#Cleaning Data by considering Percentile Functions Values

numerical_features = data.select_dtypes(exclude=['object']).columns.tolist()
len(numerical_features)


def handle_outliers(data, feature_names):
    for feature in feature_names:
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 2.5 * IQR
        upper_bound = Q3 + 2.5 * IQR
        data[feature] = np.clip(data[feature], lower_bound, upper_bound)
    return data

data_clean = handle_outliers(data.copy(), numerical_features)

data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              195 non-null    object 
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64  
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

Training and Testing Split

Training and Testing Split

```
#dropping the first column as it is a non-numerical type of data (name)

x = data.drop(columns=['status']).iloc[:,1:]
y = data['status']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, ranc
```

Logistic Regression

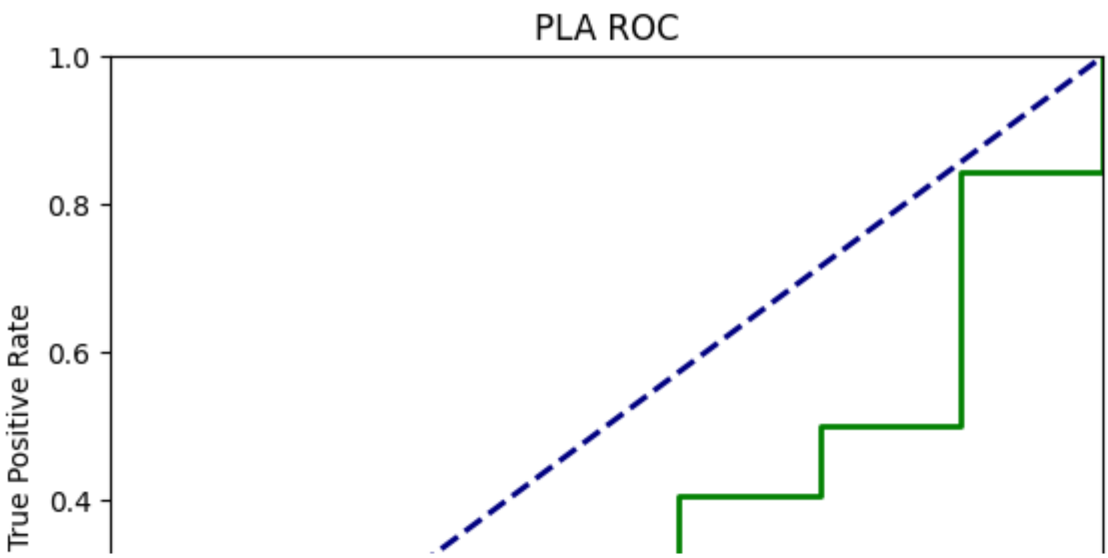
```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

log_reg = LogisticRegression()

# Training the model on the training data
log_reg.fit(X_train, y_train)

# Making predictions on the testing data
predictions = log_reg.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

print(classification_report(y_test, predictions))
```

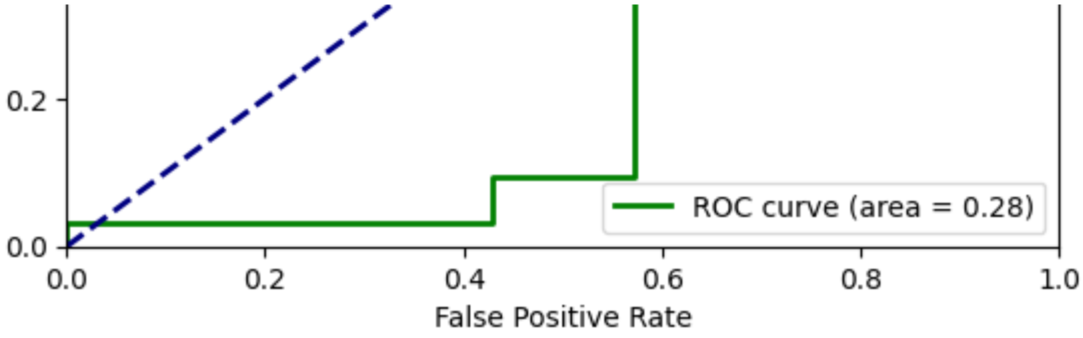
```
Accuracy: 0.8974358974358975
              precision    recall  f1-score   support

           0       1.00      0.43      0.60         7
           1       0.89      1.00      0.94        32

    accuracy                           0.90        39
   macro avg       0.94      0.71      0.77        39
weighted avg       0.91      0.90      0.88        39

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:4
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-regr
  n_iter_i = _check_optimize_result(
```

PLA

```
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.linear_model import Perceptron

# Training the Perceptron model
perceptron = Perceptron()
perceptron.fit(X_train, y_train)

# Testing the model
y_pred = perceptron.predict(X_test)
print(classification_report(y_test, y_pred))

# ROC Curve
y_scores = perceptron.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

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

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         7
           1       0.81      0.94      0.87        32

    accuracy                           0.77        39
   macro avg       0.41      0.47      0.43        39
weighted avg       0.67      0.77      0.71        39
```



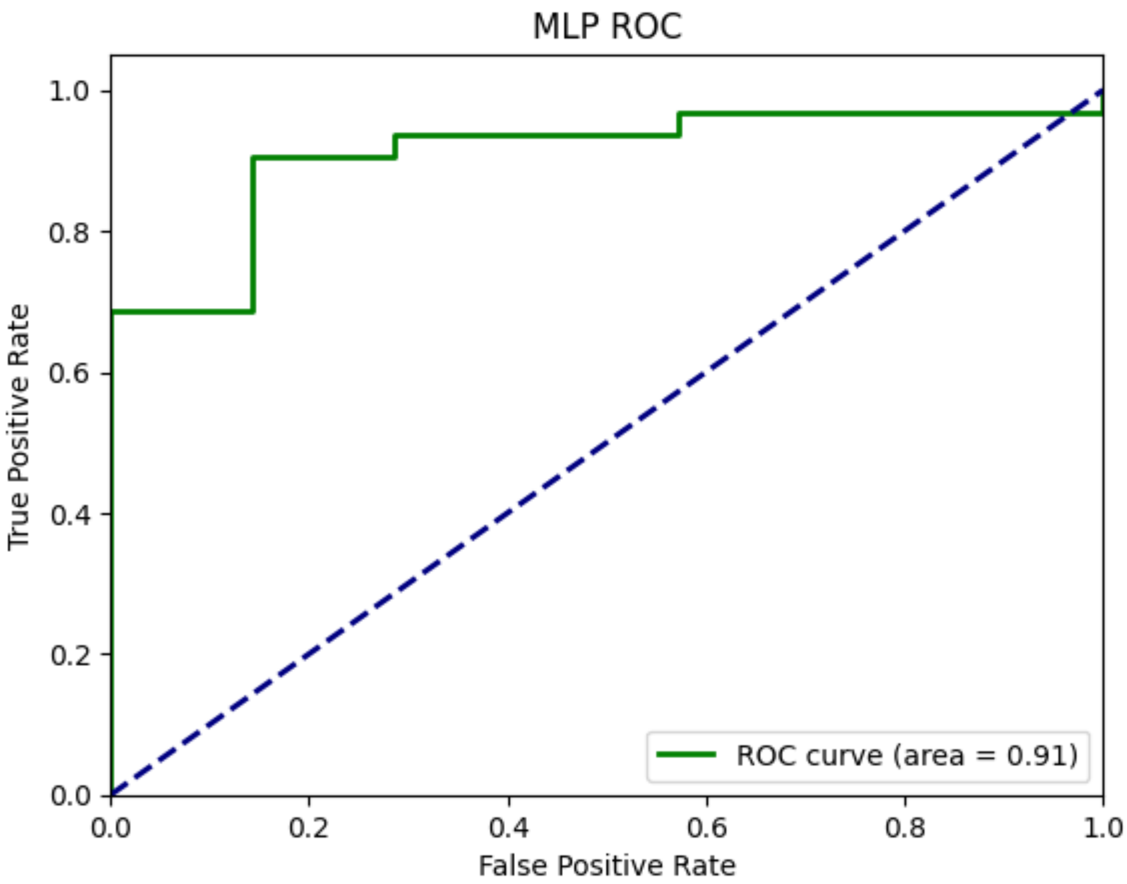


MLP

```
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.neural_network import MLPClassifier

# Training the MLP model
mlp = MLPClassifier(random_state=1, max_iter=300)
mlp.fit(X_train, y_train)

# Testing and evaluating the model
y_pred_mlp = mlp.predict(X_test)
print("MLP Classification Report:")
print(classification_report(y_test, y_pred_mlp))

y_scores_mlp = mlp.predict_proba(X_test)[:, 1]
fpr_mlp, tpr_mlp, thresholds_mlp = roc_curve(y_test, y_scores_mlp)
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)

plt.figure()
plt.plot(fpr_mlp, tpr_mlp, color='green', lw=2, label='ROC curve (area = %0.2f)
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('MLP ROC')
plt.legend(loc="lower right")
plt.show()
```

```
MLP Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.43      0.50         7
           1       0.88      0.94      0.91        32

    accuracy                           0.85        39
   macro avg       0.74      0.68      0.70        39
weighted avg       0.83      0.85      0.84        39
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_
  warnings.warn(
```

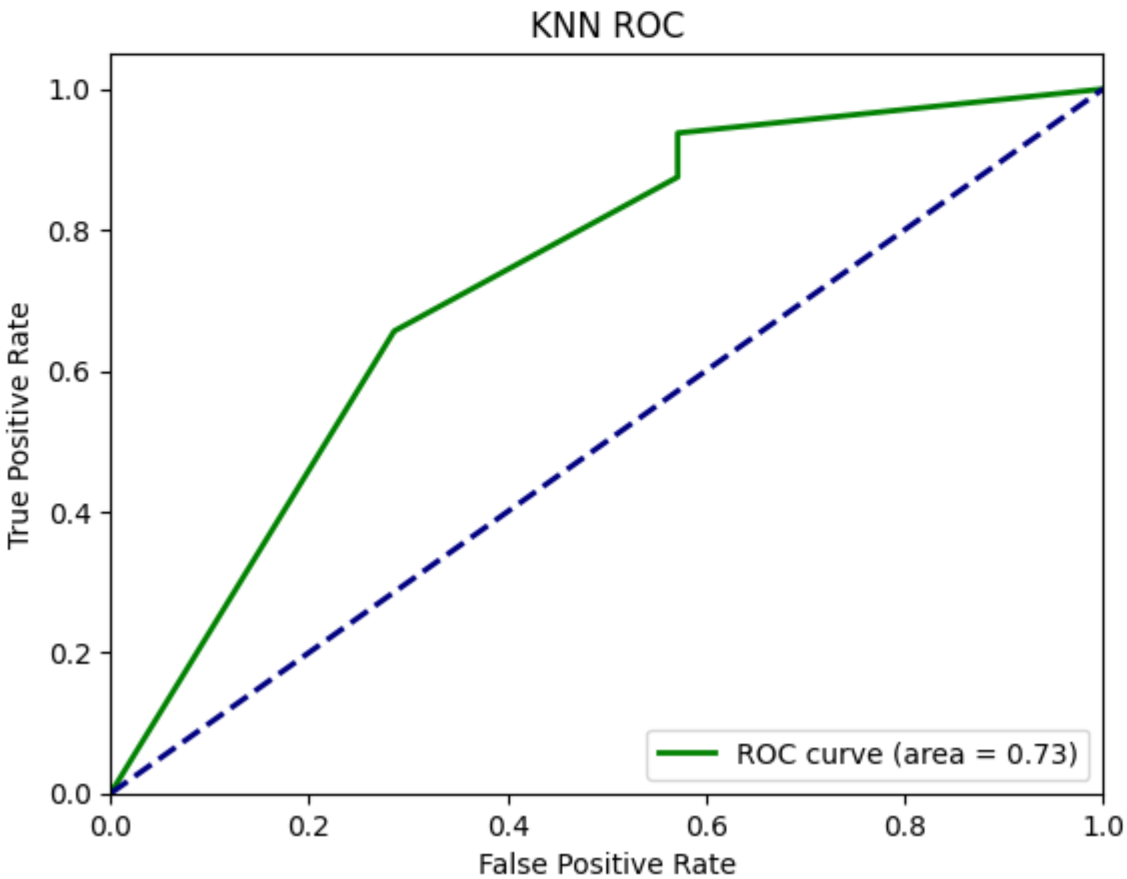


KNN

```
from sklearn.neighbors import KNeighborsClassifier

# Training the KNN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predicting and evaluating the model
y_pred_knn = knn.predict(X_test)
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))

# ROC Curve for KNN
y_scores_knn = knn.predict_proba(X_test)[:, 1]
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_scores_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)

plt.figure()
plt.plot(fpr_knn, tpr_knn, color='green', lw=2, label='ROC curve (area = %0.2f)
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('KNN ROC')
plt.legend(loc="lower right")
plt.show()
```

```
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.43      0.43      0.43         7
           1       0.88      0.88      0.88        32

    accuracy                           0.79        39
   macro avg       0.65      0.65      0.65        39
weighted avg       0.79      0.79      0.79        39
```

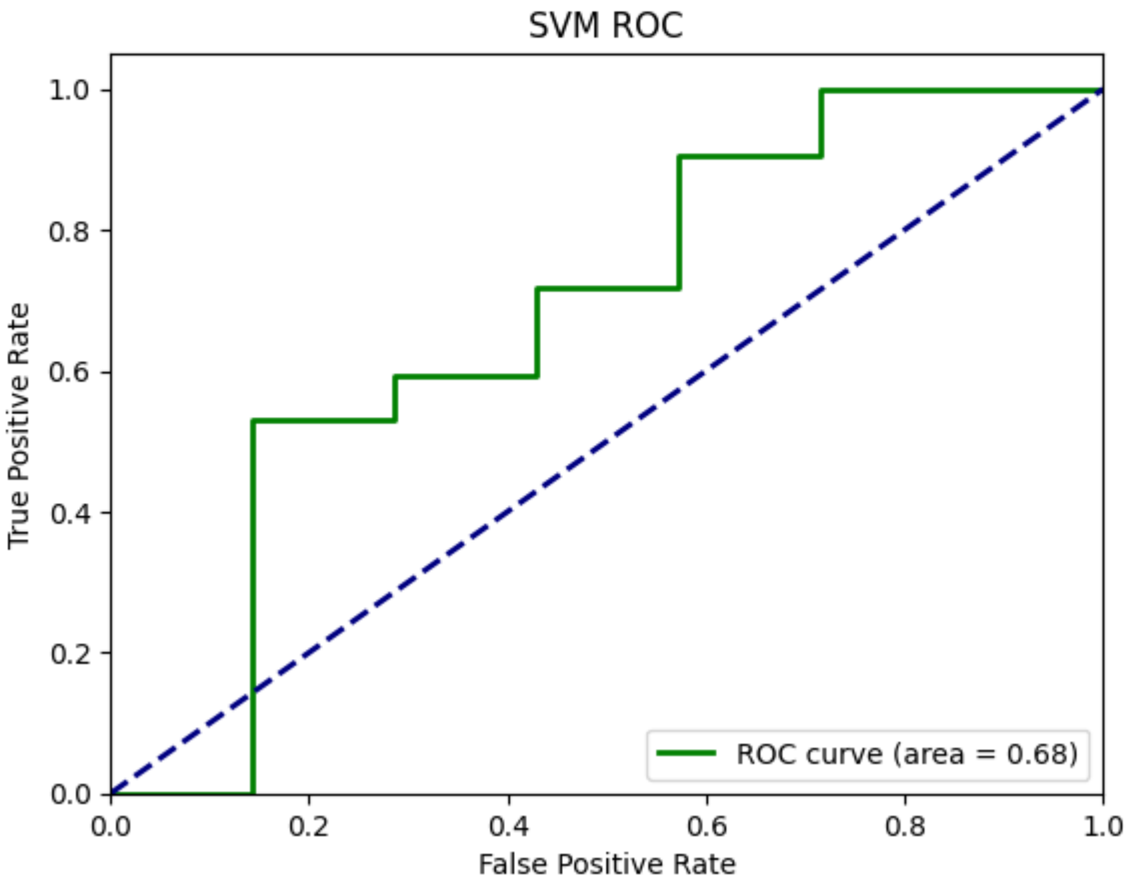


SVM

```
from sklearn.svm import SVC

# Training the SVM model
svm = SVC(probability=True)
svm.fit(X_train, y_train)

# Predicting and evaluating the model
y_pred_svm = svm.predict(X_test)
print("SVM Classification Report:")
```

```
print(classification_report(y_test, y_pred_svm))

# ROC Curve for SVM
y_scores_svm = svm.predict_proba(X_test)[:, 1]
fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, y_scores_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)

plt.figure()
plt.plot(fpr_svm, tpr_svm, color='green', lw=2, label='ROC curve (area = %0.2f)
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('SVM ROC')
plt.legend(loc="lower right")
plt.show()
```

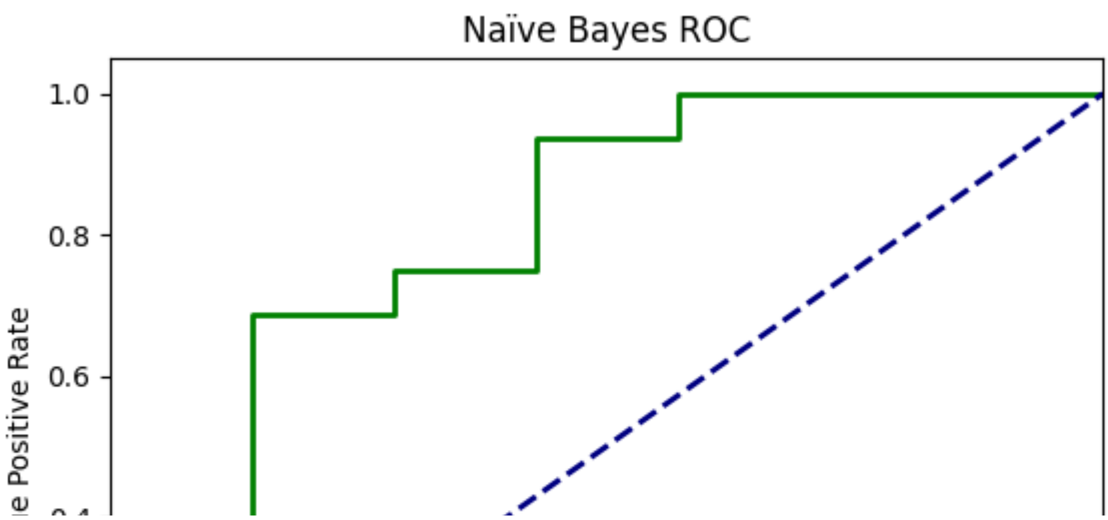
```
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.29      0.40         7
           1       0.86      0.97      0.91        32

    accuracy                           0.85        39
   macro avg       0.76      0.63      0.66        39
weighted avg       0.83      0.85      0.82        39
```

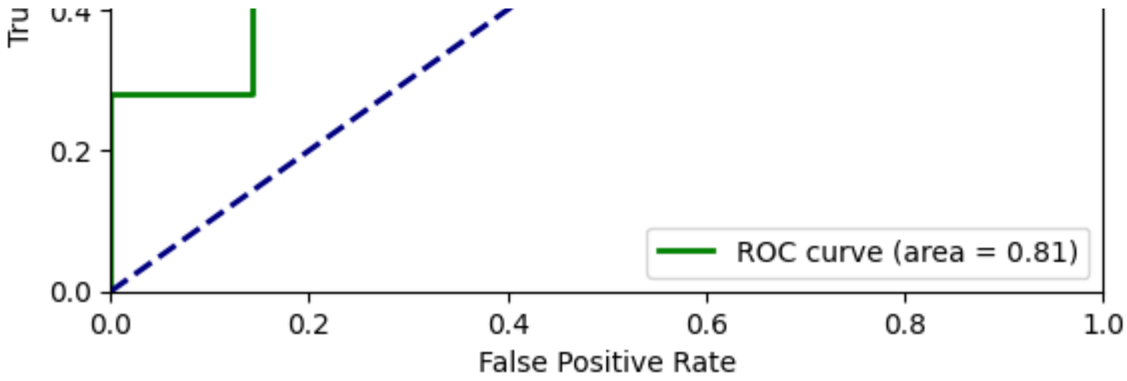


Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

# Training the Naïve Bayes model
nb = GaussianNB()
nb.fit(X_train, y_train)

# Testing and evaluating the model
y_pred_nb = nb.predict(X_test)
print("Naïve Bayes Classification Report:")
print(classification_report(y_test, y_pred_nb))

# ROC Curve for Naïve Bayes
y_scores_nb = nb.predict_proba(X_test)[:, 1]
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, y_scores_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)

plt.figure()
plt.plot(fpr_nb, tpr_nb, color='green', lw=2, label='ROC curve (area = %0.2f)'
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('Naïve Bayes ROC')
plt.legend(loc="lower right")
plt.show()
```

```
Naïve Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.71      0.45         7
           1       0.92      0.69      0.79        32

    accuracy                           0.69        39
   macro avg       0.62      0.70      0.62        39
weighted avg       0.81      0.69      0.73        39
```





COMPARISON ACROSS MODELS

```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import numpy as np


train_accuracies = {}
test_accuracies = {}


def train_evaluate(model, name, X_train, y_train, X_test, y_test):
    # Train the model
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    train_accuracies[name] = train_accuracy
    test_accuracies[name] = test_accuracy

    print(f"{name} Classification Report (Test):")
    print(classification_report(y_test, y_test_pred))

    print(f"{name} Training Accuracy: {train_accuracy:.4f}")
    print(f"{name} Testing Accuracy: {test_accuracy:.4f}")
    print("\n\n")

# Train and evaluate MLPClassifier
train_evaluate(MLPClassifier(random_state=1, max_iter=300), 'MLP', X_train, y_tr
# Train and evaluate Perceptron
train_evaluate(Perceptron(), 'Perceptron', X_train, y_train, X_test, y_test)
# Train and evaluate KNeighborsClassifier
train_evaluate(KNeighborsClassifier(n_neighbors=3), 'KNN', X_train, y_train, X_t
# Train and evaluate SVC
train_evaluate(SVC(probability=True), 'SVM', X_train, y_train, X_test, y_test)
```

```
train_evaluate(SVC(probability=True), 'SVM', X_train, y_train, X_test, y_test)
# Train and evaluate GaussianNB
train_evaluate(GaussianNB(), 'Naïve Bayes', X_train, y_train, X_test, y_test)

print("\n\n")
models = list(test_accuracies.keys())
test_accuracy_values = [test_accuracies[model] for model in models]
train_accuracy_values = [train_accuracies[model] for model in models]




best_model = max(test_accuracies, key=test_accuracies.get)
best_accuracy = test_accuracies[best_model]
print(f"The best model is {best_model} with a testing accuracy of {best_accuracy
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_
  warnings.warn(
MLP Classification Report (Test):
              precision    recall  f1-score   support

           0       0.60      0.43      0.50         7
           1       0.88      0.94      0.91        32

    accuracy                           0.85        39
   macro avg       0.74      0.68      0.70        39
weighted avg       0.83      0.85      0.84        39

MLP Training Accuracy: 0.8718
MLP Testing Accuracy: 0.8462



Perceptron Classification Report (Test):
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         7
           1       0.81      0.94      0.87        32

    accuracy                           0.77        39
   macro avg       0.41      0.47      0.43        39
weighted avg       0.67      0.77      0.71        39

Perceptron Training Accuracy: 0.7308
Perceptron Testing Accuracy: 0.7692



KNN Classification Report (Test):
              precision    recall  f1-score   support

           0       0.43      0.43      0.43         7
           1       0.88      0.88      0.88        32

    accuracy                           0.79        39
   macro avg       0.65      0.65      0.65        39
weighted avg       0.79      0.79      0.79        39
```

```
KNN Training Accuracy: 0.9295
KNN Testing Accuracy: 0.7949



SVM Classification Report (Test):
              precision    recall  f1-score   support

           0       0.67      0.29      0.40         7
           1       0.86      0.97      0.91        32

    accuracy                           0.85        39
   macro avg       0.76      0.63      0.66        39
weighted avg       0.83      0.85      0.82        39

SVM Training Accuracy: 0.8077
```

INFERENCES

By comparing all the models above, we can infer that:

1) The best performing model by accuracy is the Logistic Regression Model.

2) Based on the ROC-AUC parameter, the MLP (Multi Layer Perceptron) model and SVM (Support Vector Machine) models perform equally giving an 85% accuracy each. MLP edges out SVM marginally.

3) PLA (Perceptrion Learning Algorithm) model is the weakest among all based on the ROC-AUC graphs that are observed.