KENNESAW STATE UNIVERSITY
COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING
School of Data Science and Analytics

Predictive Model for High Blood Pressure(Hypertension)

Prativa Basnet

Faculty Advisor: Dr. MinJae Woo

The report was not created for this project since it was not required by the professor.

python

INTRODUCTION

- Hypertension also known as high blood pressure, is defined as the elevated blood pressure in the arteries as the heart pumps blood throughout the body.
- Hypertension is one of the leading causes of heart disease such as heart attacks and strokes.
- Hypertension is categorized into different stages based on blood pressure.
  ✓Normal: Systolic<120mmHg and Diastolic<80mmHg
  ✓Elevated: Systolic<120-129mmHg and Diastolic<80mmHg
  ✓Hypertension Stage1:Systolic-130-139mmHg and Diastolic-80-89mmHg
  ✓Hypertension Stage 2:Systolic>140mmHg and Diastolic>90mmHg
  ✓Hypertensive Crisis: Systolic>180mmHg and Diastolic>120mmHg
- Hypertension is hard to detect in early stage and also known as "silent killer". Therefore, lifestyle modifications such as healthy diet, physical activity, limiting alcohol consumption, quitting smoking, and managing stress are required.
- The purpose of this study is to identify the most important factors in predicting hypertension.

DATASET/METHODS

Dataset (Framingham Heart study)

- Dataset for this study is taken from Framingham Heart study.
- Framingham Heart study is long-term , ongoing cardiovascular cohort started in 1948. The study was started by National Heart Institute(National Heart, Lung, and Blood Institute).
- The study conducted in several phases and first phase began in 1948 which includes 5,209 adult residents who were free of cardiovascular disease at the beginning of study.
- Dataset included 11,627 observations and 12 variables.
- Dataset include total cholesterol, age, body mass index, glucose, diabetes, hypertension, smoking, systolic, diastolic, education, gender, and resting heart rate variables.

Data Processing

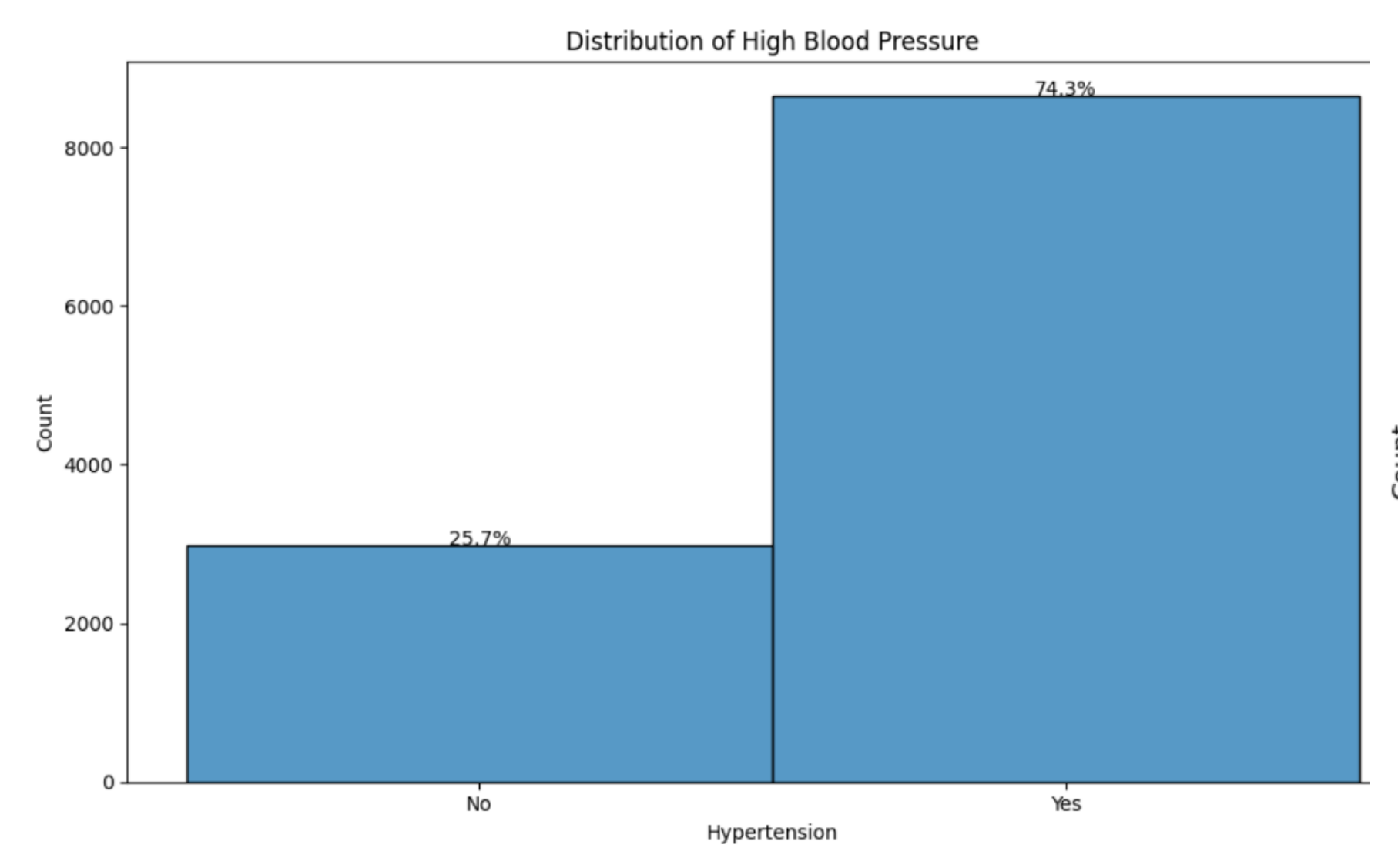
- Dataset has no missing values.
- Hypertension is used as target variable.
- Variables systolic and diastolic are removed from the analysis.
- Correlation performed for independent continuous variables.
- Use of 80:20 ratio for train/test sets for all models.
- Hypertension has imbalance of 74.3% (Positive) and 25.7% (Negative).

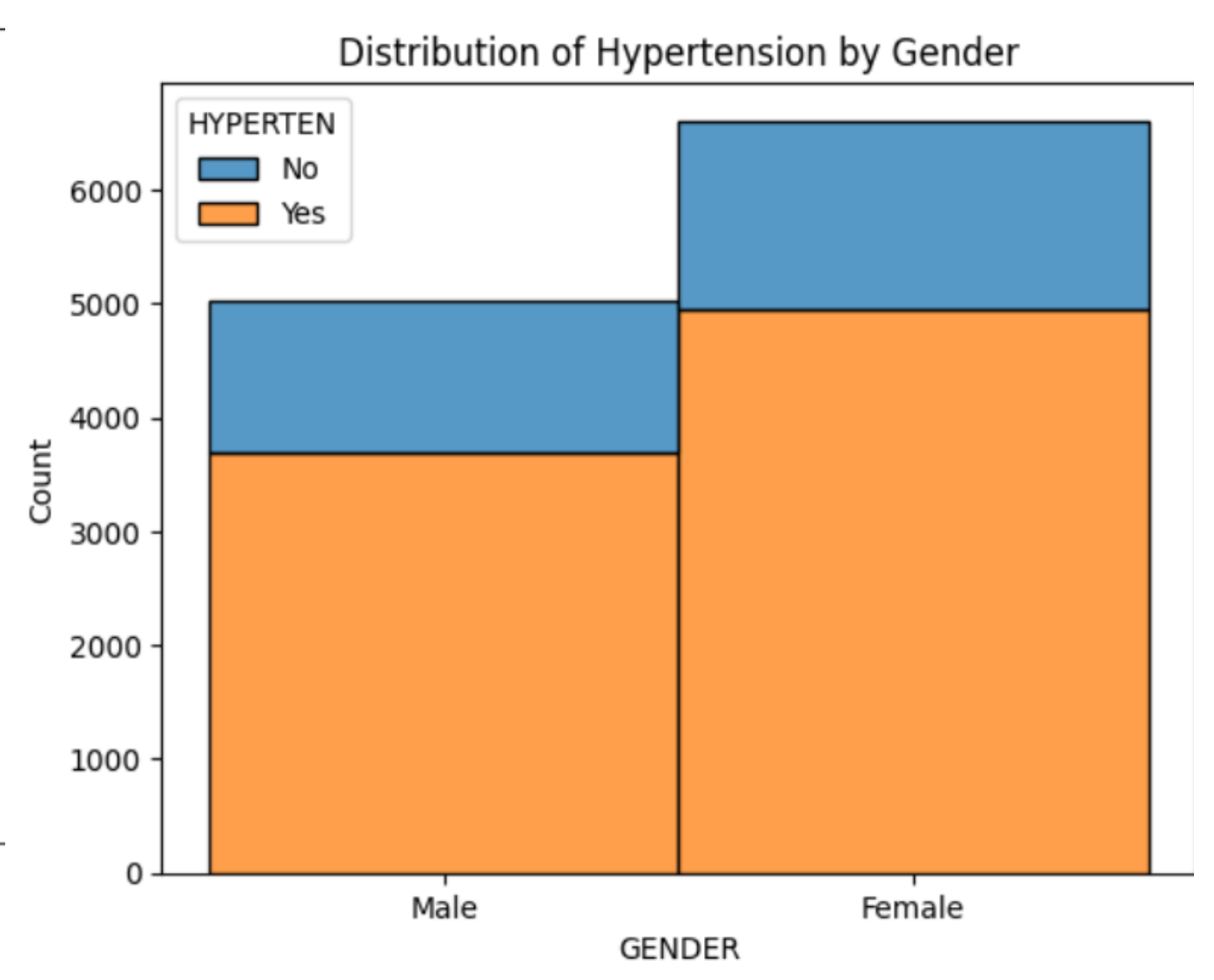
Modeling Methods

- Random Forest Classifier
- XGBoost
- Logistic Regression
- Naïve Bayes

Modeling Results

- Confusion Matrixes, ROC, and Precision-Recall curves created for each model.
- Accuracy, Precision, sensitivity(Recall), F1 Score , ROC/AUC, and PR/AUC model performance metrics calculated for each model.





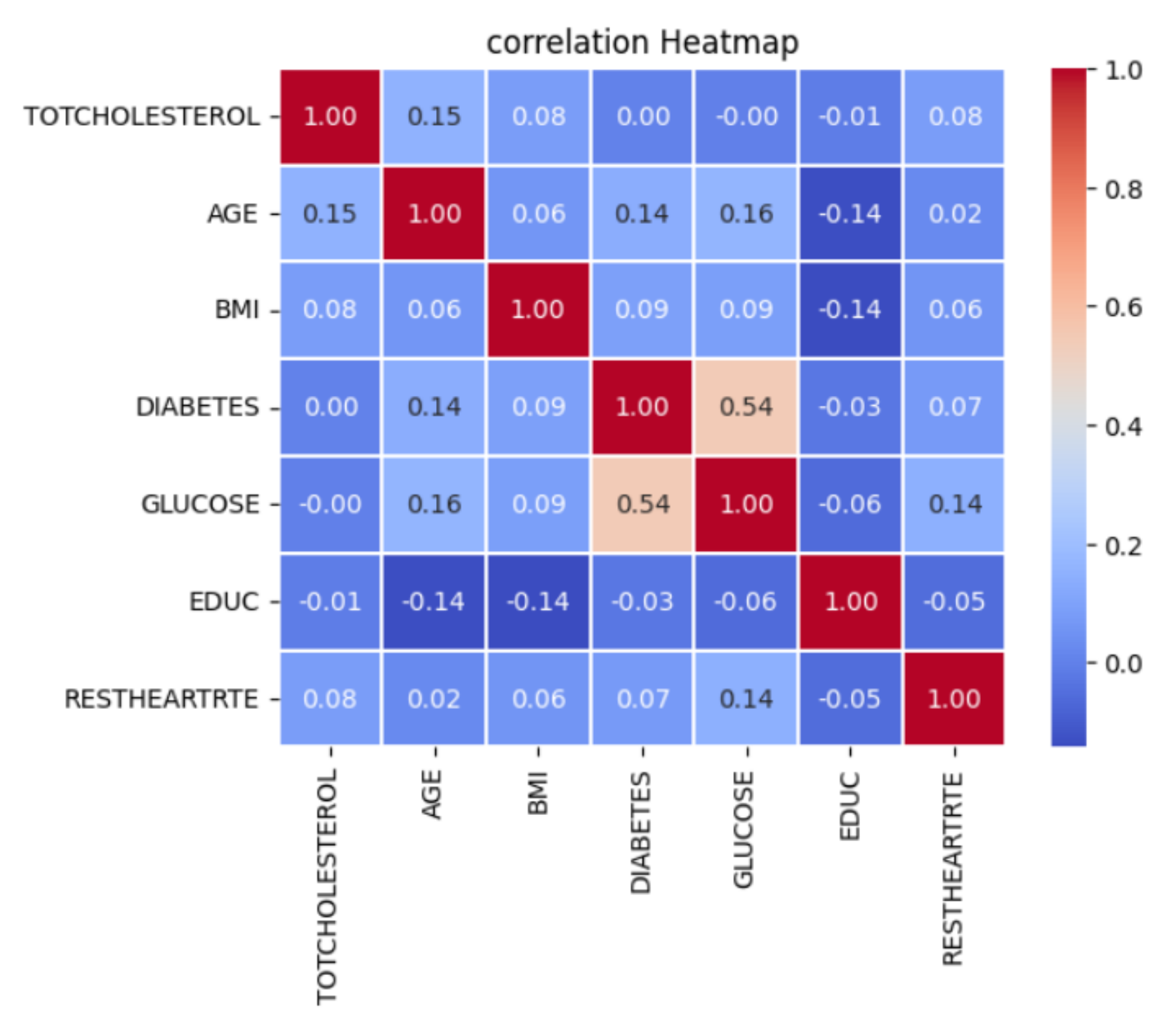
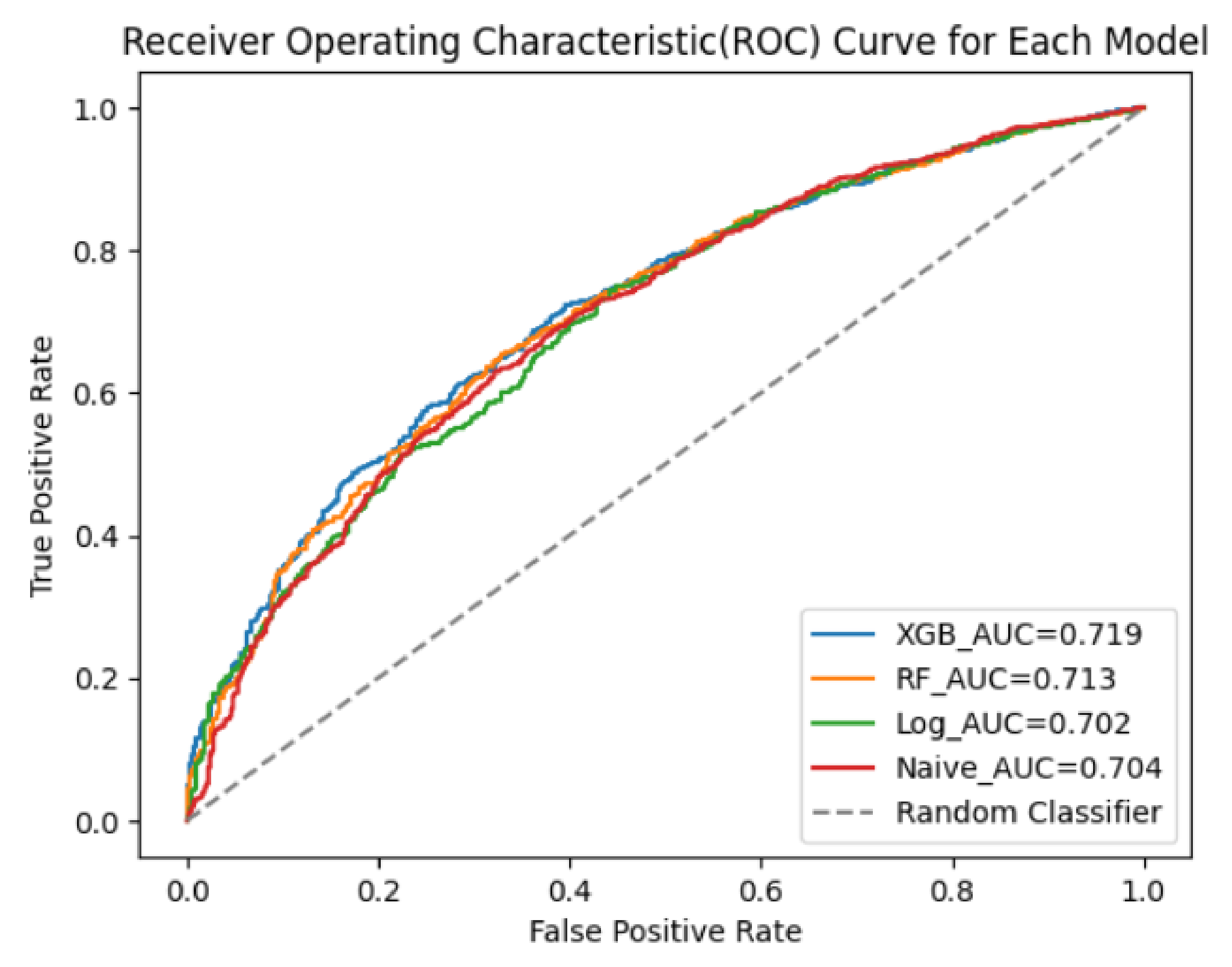
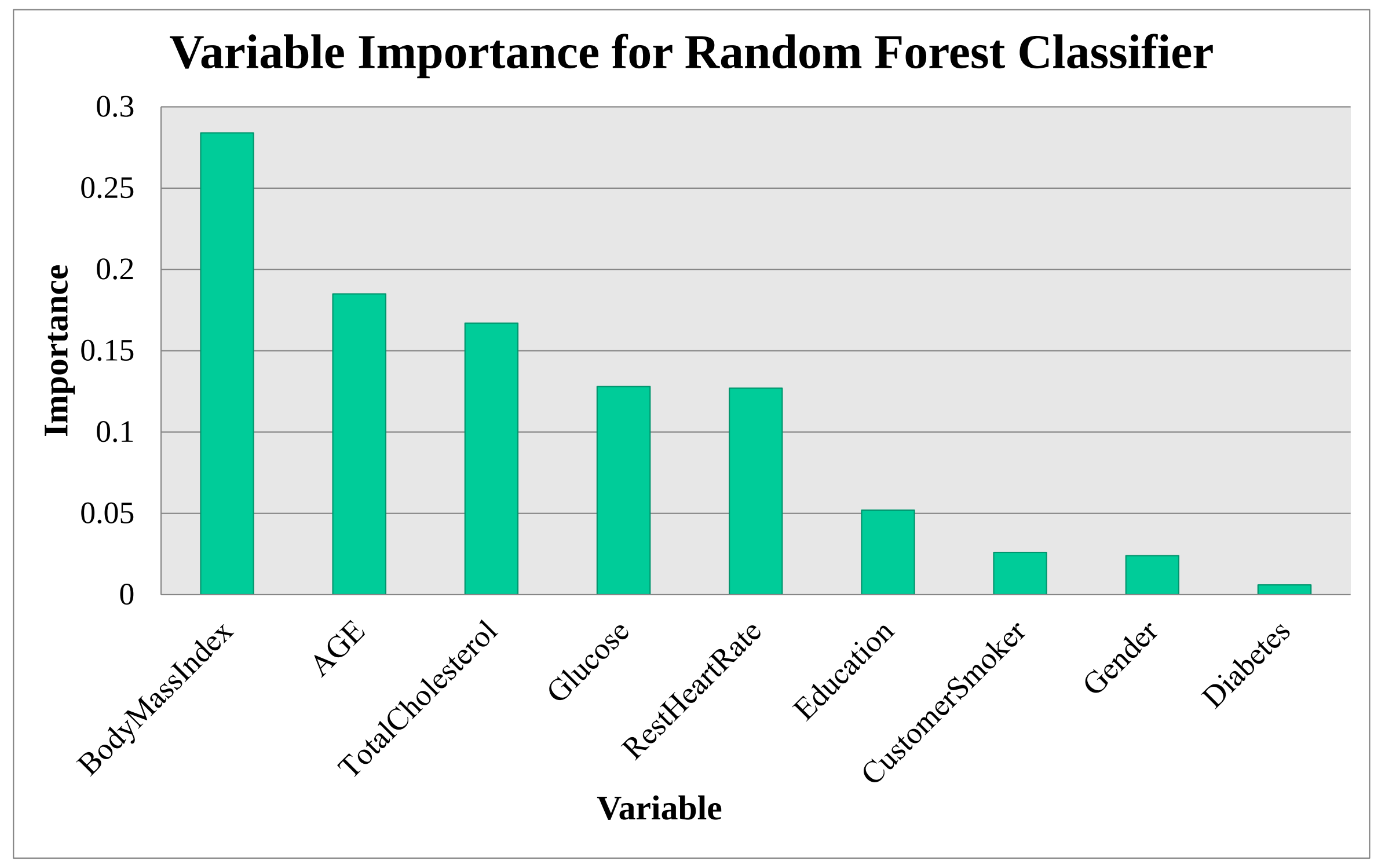


Table 2: Model Performance Metrics Comparison

Performance Matric of Models				
	XGBoost	Random Forest	Logistic Regression	Naïve Bayes
Accuracy	0.754	0.756	0.754	0.678
Precision	0.771	0.770	0.761	0.839
Sensitivity (Recall)	0.956	0.961	0.979	0.705
F1 Score	0.853	0.855	0.856	0.766
ROC/AUC	0.719	0.713	0.702	0.704
PR/AUC	0.882	0.876	0.869	0.862





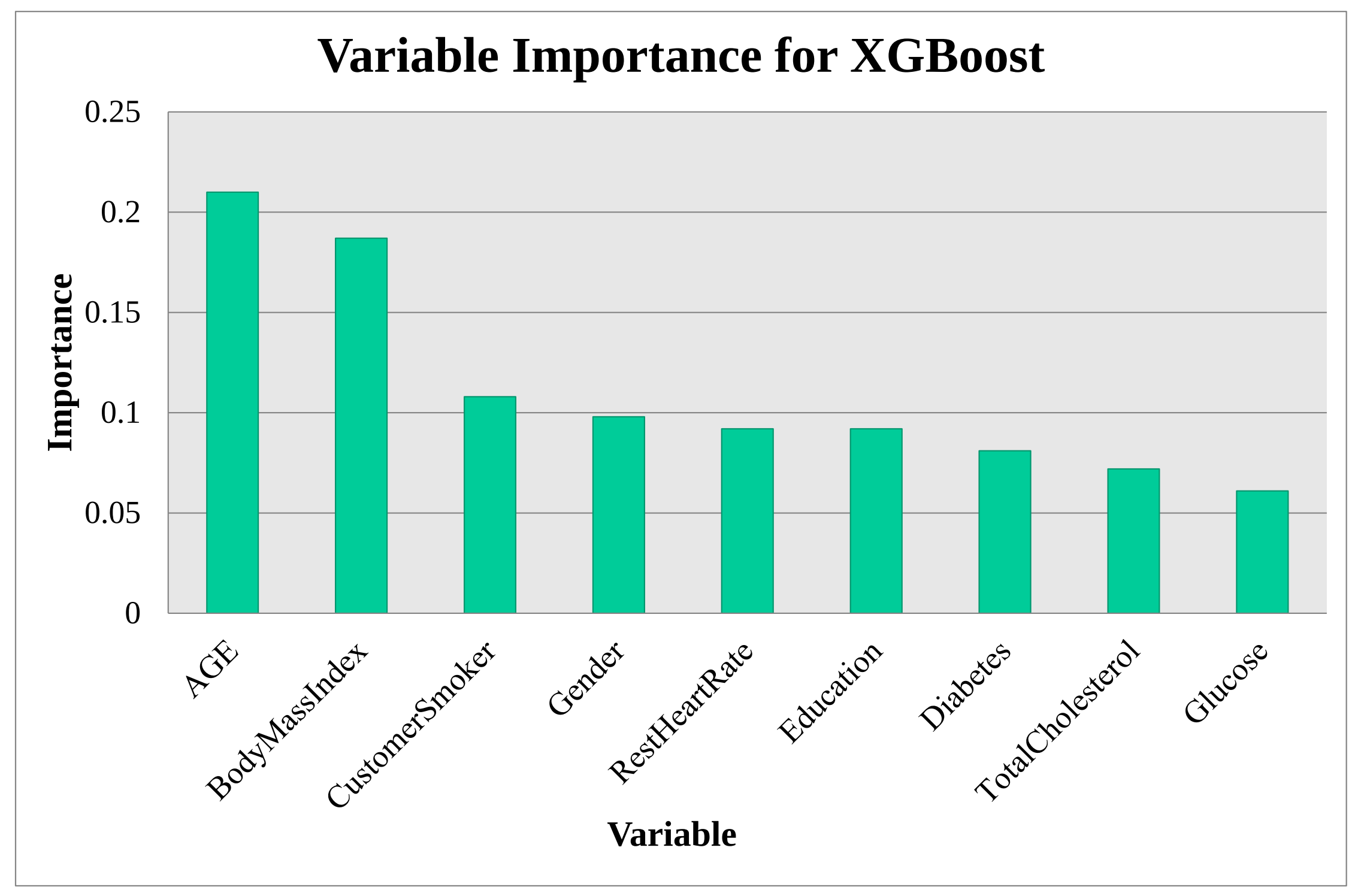
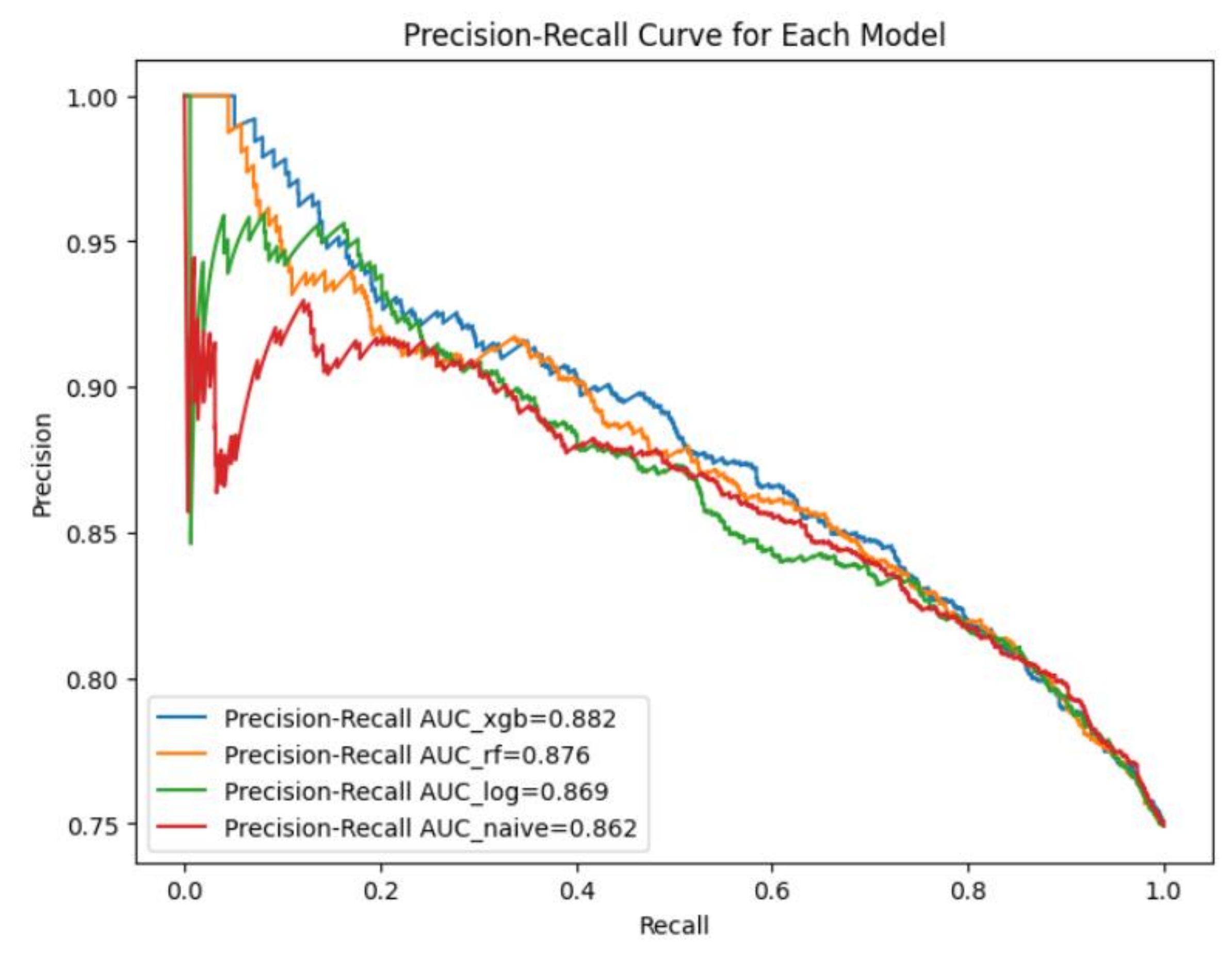


Table 1: Confusion Matrix for Each Model

	Predicted Models	
Actual Models	Random Forest	
	Positive	Negative
	P: 1675	68
	N: 499	84
	Logistic Regression	
	P: 1706	37
	N: 535	48
	XGBoost	
	P: 1666	77
	N: 496	87
	Naïve Bayes	
	P: 1228	515
	N: 235	348



RESULTS

- Evaluation matrices such as Precision, sensitivity(Recall), F1 Score , AUC/ROC, and AUC/PR are used to evaluate performance of the model since dataset has moderate imbalance.
- XGBoost model has the slightly higher AUC/ROC and PR/AUC curves score of all the models.
- Logistic Regression model has slightly higher Sensitivity(Recall) and F1 Score compared to other models.
- Naïve Bayes has slightly higher precision than other models.
- XGBoost is the best among all 4 models based on evaluation criteria.

Random Forest Hyperparameters and Feature Importance

- Best Hyperparamters are bootstrap: True, max_depth:10, min_sample_leaf: 2, min_samples_split: 2 and n_estimators:100.
- Five most important variables to predict the Hypertension are body mass index, age, total cholesterol, glucose, and resting heart rate, respectively.

XGBoost Hyperparameters and Feature Importance

- Best parameters are colsample_bytree:0.657, gamma: 0.333, learning_rate:0.034, max_depth:3, min_child_weight:7, n_estimator:154, and subsample:0.862.
- Five most important variables to predict the Hypertension are age, body mass index ,customer smoker, gender, and resting heart rate, respectively.

DISCUSSION

- Random Forest Classifier variable importance shows patient's BMI has the highest contribution on hypertension prediction.
- XGBoost variable importance shows patient's age has the highest contribution on hypertension prediction.
- Naïve Bayes model is slighter better for precision.
- Logistic Regression model is slightly better for sensitivity(recall) and F1 Score.
- XGBoost model is slightly better for ROC/AUC and PR/AUC.
- The performance of all the models is slightly different. However, XGBoost and Logistic Regression have slightly better performance compared to Random Forest and Naïve Bayes. The selection between XGBoost and Logistic Regression depends on interpretation and clinical settings.

.

Limitations

- Dataset contains only 9 independent variables which is not sufficient.
- Model may perform better with more variables such as family history, stress, exercise, diet, etc.

References

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

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

```
# Read dataset
import pandas as pd
hypertension=pd.read_csv('/content/drive/MyDrive/DS_7140/Project/rest_heart_rate.csv')
hypertension
```

	TOTCHOLESTEROL	AGE	SYSBP	DIABP	CURSMOKE	BMI	DIABETES	HYPERTEN	GLUCOSE	EDUC	GENDER	RESTHEARTRTE
0	195	39	106	70	No	26.97	0	No	77	4	Male	80
1	209	52	121	66	No	23.58	0	No	92	4	Male	69
2	250	46	121	81	No	28.73	0	No	76	2	Female	95
3	260	52	105	70	No	29.43	0	No	86	2	Female	80
4	237	58	108	66	No	28.50	0	No	71	2	Female	80
...	...	...	...	...	...	...	...	...	...	...	...	...
11622	173	46	126	82	No	19.17	0	Yes	79	3	Male	70
11623	153	52	143	89	No	25.74	0	Yes	72	3	Male	65
11624	196	39	133	86	Yes	20.91	0	Yes	80	3	Female	85
11625	240	46	138	79	Yes	26.39	0	Yes	83	3	Female	90
11626	252	50	147	96	Yes	24.19	0	Yes	82	3	Female	94

11627 rows × 12 columns

```
# Rename the dataset
hypertension_final= hypertension[["TOTCHOLESTEROL","AGE","CURSMOKE","BMI","DIABETES","HYPERTEN","GLUCOSE","EDUC","GENDER","RESTHEARTRTE"]]
hypertension_final
```

	TOTCHOLESTEROL	AGE	CURSMOKE	BMI	DIABETES	HYPERTEN	GLUCOSE	EDUC	GENDER	RESTHEARTRTE
0	195	39	No	26.97	0	No	77	4	Male	80
1	209	52	No	23.58	0	No	92	4	Male	69
2	250	46	No	28.73	0	No	76	2	Female	95
3	260	52	No	29.43	0	No	86	2	Female	80
4	237	58	No	28.50	0	No	71	2	Female	80
...	...	...	...	...	...	...	...	...	...	...
11622	173	46	No	19.17	0	Yes	79	3	Male	70
11623	153	52	No	25.74	0	Yes	72	3	Male	65
11624	196	39	Yes	20.91	0	Yes	80	3	Female	85
11625	240	46	Yes	26.39	0	Yes	83	3	Female	90
11626	252	50	Yes	24.19	0	Yes	82	3	Female	94

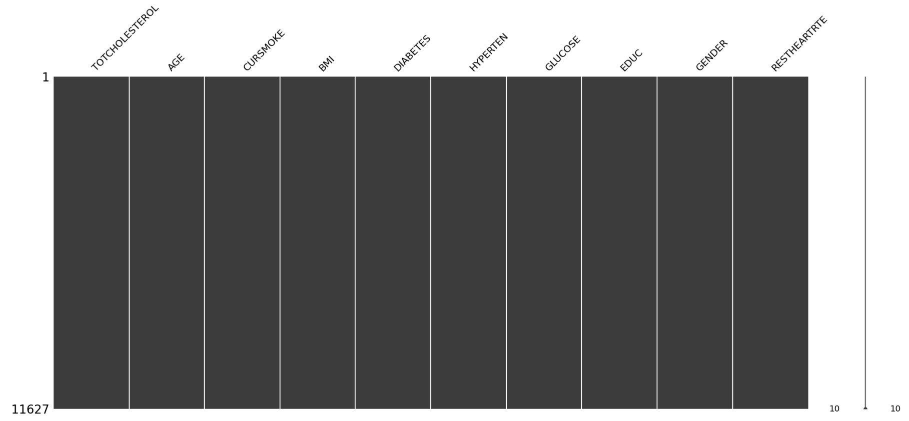
11627 rows × 10 columns

```
# Type of variable
data_type=hypertension_final.dtypes
data_type
```

```
TOTCHOLESTEROL      int64
AGE                 int64
CURSMOKE           object
BMI               float64
DIABETES            int64
HYPERTEN           object
GLUCOSE             int64
EDUC                int64
GENDER             object
RESTHEARTRTE        int64
dtype: object
```

```
import missingno as msno
# Matrix of each variables with missing
msno.matrix(hypertension_final)

# List missing vaules for each variables
hypertension_final.isnull().sum() # None missing value
```

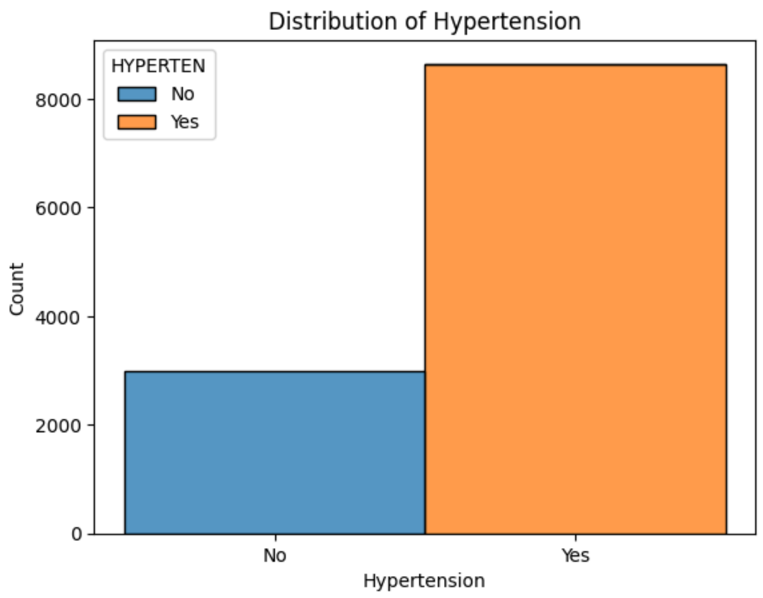
```
TOTCHOLESTEROL    0
AGE               0
CURSMOKE          0
BMI               0
DIABETES          0
HYPERTEN          0
GLUCOSE           0
EDUC              0
GENDER            0
RESTHEARTRTE      0
dtype: int64
```



```
# Use describe function to find the coded value in dataset"rest_heart_rate_final"
hypertension_final.describe()
```

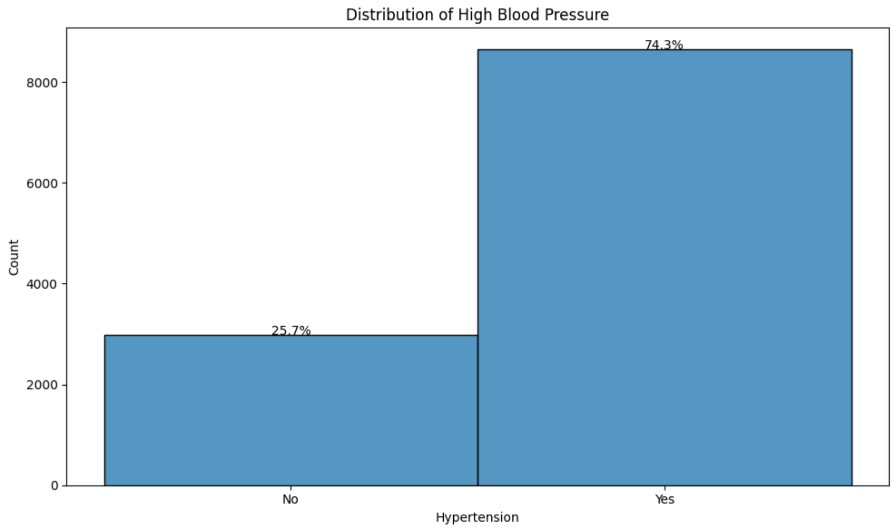
	TOTCHOLESTEROL	AGE	BMI	DIABETES	GLUCOSE	EDU
count	11627.000000	11627.000000	11627.000000	11627.000000	11627.000000	11627.00000
mean	241.362174	54.792810	25.878276	0.045584	84.319343	1.98959
std	44.620667	9.564299	4.095190	0.208589	24.027636	1.01484
min	107.000000	32.000000	14.430000	0.000000	39.000000	1.00000
25%	211.000000	48.000000	23.100000	0.000000	73.000000	1.00000
50%	239.000000	54.000000	25.480000	0.000000	80.000000	2.00000
75%	267.000000	62.000000	28.060000	0.000000	88.000000	3.00000
max	696.000000	81.000000	56.800000	1.000000	478.000000	4.00000

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming "gender" is a column in your DataFrame
# You can use the hue parameter to differentiate between hypertensive and non-hypertensive individuals
sns.histplot(data=hypertension_final, x="HYPERTEN", hue="HYPERTEN", multiple="stack", bins=2)
plt.xlabel("Hypertension")
plt.ylabel("Count")
plt.title("Distribution of Hypertension")
plt.show()
```

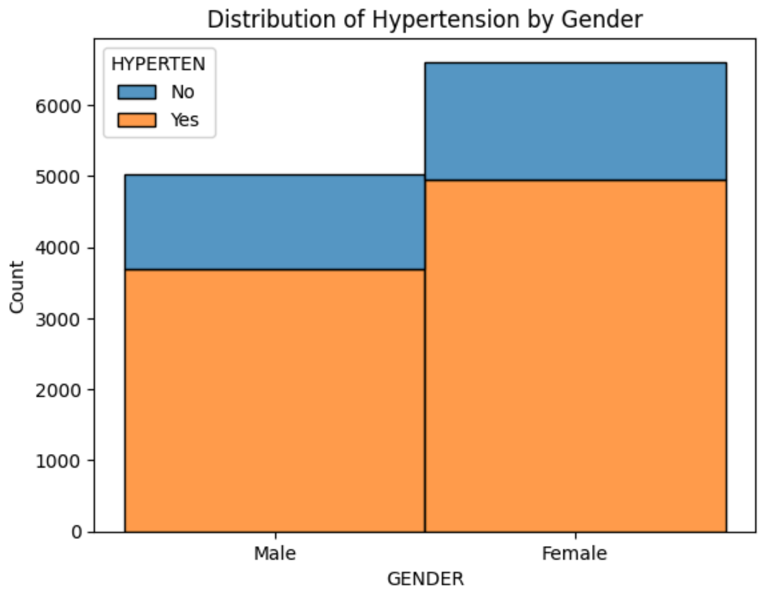


```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Group by HYPERTEN and count occurrences
count = hypertension_final.groupby('HYPERTEN').size().reset_index(name='Count')

# Calculate total count
total_count = count['Count'].sum()

# Calculate percentage
count['Percentage'] = (count['Count'] / total_count) * 100

# Create histogram for Hypertension
plt.figure(figsize=(10, 6))
sns.histplot(data=hypertension_final, x='HYPERTEN', discrete=True, stat='count', palette=['#FF5733', '#33FF9E'])
plt.title("Distribution of High Blood Pressure")
plt.xlabel("Hypertension")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])  # Set x-axis labels for 'No' and 'Yes'
plt.grid(False)  # Remove gridlines

# Add percentage annotations
for index, row in count.iterrows():
    plt.text(index, row['Count'], f"{row['Percentage']:.1f}%", color='black', ha="center")

plt.tight_layout()
plt.show()
```

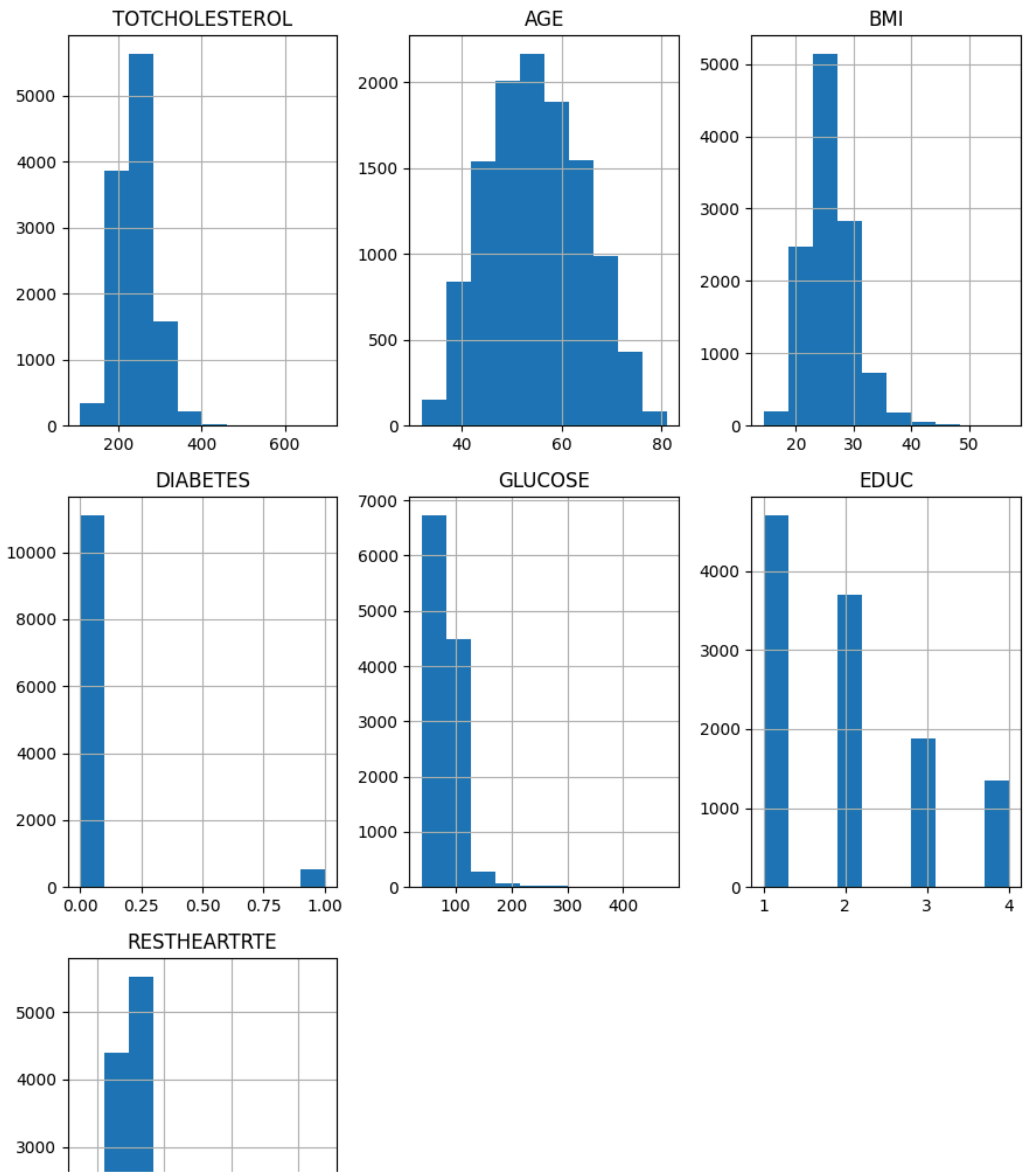
```
<ipython-input-8-f9dc273b55a4>:16: UserWarning: Ignoring `palette` because no `hue` vari
  sns.histplot(data=hypertension_final, x='HYPERTEN', discrete=True, stat='count', palet
```



```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming "gender" is a column in your DataFrame
# You can use the hue parameter to differentiate between hypertensive and non-hypertensive individuals
sns.histplot(data=hypertension_final, x="GENDER", hue="HYPERTEN", multiple="stack", bins=2)
plt.xlabel("GENDER")
plt.ylabel("Count")
plt.title("Distribution of Hypertension by Gender")
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
# Use original dataset for histogram
hypertension_final.hist(figsize=(9, 12))  # figsize is for adjusting the figure size
plt.tight_layout()  # Adjust layout to prevent overlap of subplots
plt.show()
```

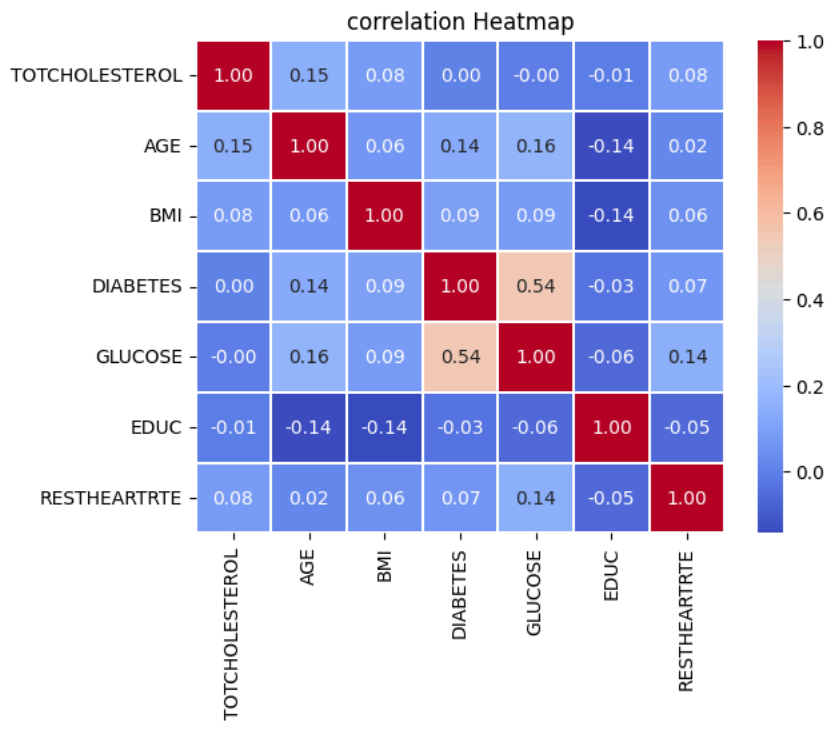


```
# subset contionous vars only
continous_vars = hypertension_final[['TOTCHOLESTEROL','AGE','BMI','DIABETES', 'GLUCOSE','EDUC','RESTHEARTRTE']]
continous_vars
```

	TOTCHOLESTEROL	AGE	BMI	DIABETES	GLUCOSE	EDUC	RESTHEARTRTE
0	195	39	26.97	0	77	4	80
1	209	52	23.58	0	92	4	69
2	250	46	28.73	0	76	2	95
3	260	52	29.43	0	86	2	80
4	237	58	28.50	0	71	2	80
...	...	...	...	...	...	...	...
11622	173	46	19.17	0	79	3	70
11623	153	52	25.74	0	72	3	65
11624	196	39	20.91	0	80	3	85
11625	240	46	26.39	0	83	3	90
11626	252	50	24.19	0	82	3	94

11627 rows × 7 columns

```
# How to generate correlation heatmap for dataframe named 'df'
import seaborn as sns
import matplotlib.pyplot as plt

# Creat correlation matrix for independent varible "x"
correlation_matrix= continous_vars.corr()
#print(correlation_matrix)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',fmt=".2f", linewidths=0.3)
plt.title("correlation Heatmap")
plt.show()
```



	TOTCHOLESTEROL	AGE	BMI	DIABETES	GLUCOSE	EDUC	RESTHEARTRTE
TOTCHOLESTEROL	1.00	0.15	0.08	0.00	-0.00	-0.01	0.08
AGE	0.15	1.00	0.06	0.14	0.16	-0.14	0.02
BMI	0.08	0.06	1.00	0.09	0.09	-0.14	0.06
DIABETES	0.00	0.14	0.09	1.00	0.54	-0.03	0.07
GLUCOSE	-0.00	0.16	0.09	0.54	1.00	-0.06	0.14
EDUC	-0.01	-0.14	-0.14	-0.03	-0.06	1.00	-0.05
RESTHEARTRTE	0.08	0.02	0.06	0.07	0.14	-0.05	1.00

SYSBP and DIAB have correleation of 0.71 and glucose and diabetes have coreletio of 0.54. so we need to remove one of them.

```
# Dummies for chracter variable
# "drop_first" drops the first category level for each categorical variable
# "dummy_na=True" creates dummy variables for missing values
# Specify the columns you want to encode
columns_to_encode = ['GENDER','CURSMOKE']
# Apply get_dummies() only to the selected columns
hypertension_final_dumm = pd.get_dummies(hypertension_final, columns=columns_to_encode, drop_first=True, dummy_na=False)
hypertension_final_dumm
```

	TOTCHOLESTEROL	AGE	BMI	DIABETES	HYPERTEN	GLUCOSE	EDUC	RESTHEARTRTE	GENDER_Male	CURSMOKE_Yes
0	195	39	26.97	0	No	77	4	80	True	False
1	209	52	23.58	0	No	92	4	69	True	False
2	250	46	28.73	0	No	76	2	95	False	False
3	260	52	29.43	0	No	86	2	80	False	False
4	237	58	28.50	0	No	71	2	80	False	False
...	...	...	...	...	...	...	...	...	...	...
11622	173	46	19.17	0	Yes	79	3	70	True	False
11623	153	52	25.74	0	Yes	72	3	65	True	False
11624	196	39	20.91	0	Yes	80	3	85	False	True
11625	240	46	26.39	0	Yes	83	3	90	False	True
11626	252	50	24.19	0	Yes	82	3	94	False	True

11627 rows × 10 columns

```
# Check balance of target variable
class_column = 'HYPERTEN'
# Count the occurrences of each class label
class_counts = hypertension_final_dumm[class_column].value_counts()
# Display class distribution
print("Class Distribution:")
print(class_counts)

# Calculate imbalance ratio
imbalance_ratio = class_counts.min() / class_counts.max()
print("Imbalance Ratio:", imbalance_ratio)
# Hypertension is 8642 which is 74.33% and nohypertension is 2985 which is 25.67% so imbalance dataset
```

```
Class Distribution:
HYPERTEN
Yes    8642
No     2985
Name: count, dtype: int64
Imbalance Ratio: 0.3454061559824115
```

```
#  dummy coding convert True/False to 1/0 for these categorical columns
categorical_columns = ['GENDER_Male', 'CURSMOKE_Yes',]

# Change True/False to 1/0 for the specified categorical columns
hypertension_final_dumm[categorical_columns] = hypertension_final_dumm[categorical_columns].astype(int)
hypertension_final_dumm
```

	TOTCHOLESTEROL	AGE	BMI	DIABETES	HYPERTEN	GLUCOSE	EDUC	RESTHEARTRTE	GENDER_Male	CURSMOKE_Yes
0	195	39	26.97	0	No	77	4	80	1	0
1	209	52	23.58	0	No	92	4	69	1	0
2	250	46	28.73	0	No	76	2	95	0	0
3	260	52	29.43	0	No	86	2	80	0	0
4	237	58	28.50	0	No	71	2	80	0	0
...	...	...	...	...	...	...	...	...	...	...
11622	173	46	19.17	0	Yes	79	3	70	1	0
11623	153	52	25.74	0	Yes	72	3	65	1	0
11624	196	39	20.91	0	Yes	80	3	85	0	1
11625	240	46	26.39	0	Yes	83	3	90	0	1
11626	252	50	24.19	0	Yes	82	3	94	0	1

11627 rows × 10 columns

```
# Define the order of target varible
desired_order = ["Yes", "No"]  # Define the desired order of categories

# Convert the target variable to categorical with the specified order of categories
hypertension_final_dumm["HYPERTEN"] = pd.Categorical(hypertension_final_dumm["HYPERTEN"], categories=desired_order)
hypertension_final_dumm

# Print the value counts to verify the order
print(hypertension_final_dumm["HYPERTEN"].value_counts())
```

```
HYPERTEN
Yes    8642
No     2985
Name: count, dtype: int64
```

```
# Convert categorical labels to binary labels
hypertension_final_dumm["HYPERTEN_BINARY"] = hypertension_final_dumm["HYPERTEN"].replace({'Yes': 1, 'No': 0})
hypertension_final_dumm
```

	TOTCHOLESTEROL	AGE	BMI	DIABETES	HYPERTEN	GLUCOSE	EDUC	RESTHEARTRTE	GENDER_Male	CURSMOKE_Yes	HYPERTEN_BINARY
0	195	39	26.97	0	No	77	4	80	1	0	0
1	209	52	23.58	0	No	92	4	69	1	0	0
2	250	46	28.73	0	No	76	2	95	0	0	0
3	260	52	29.43	0	No	86	2	80	0	0	0
4	237	58	28.50	0	No	71	2	80	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
11622	173	46	19.17	0	Yes	79	3	70	1	0	1
11623	153	52	25.74	0	Yes	72	3	65	1	0	1
11624	196	39	20.91	0	Yes	80	3	85	0	1	1
11625	240	46	26.39	0	Yes	83	3	90	0	1	1
11626	252	50	24.19	0	Yes	82	3	94	0	1	1

11627 rows × 11 columns

```
# Print the value counts to verify the order
hypertension_final_dumm["HYPERTEN_BINARY"].value_counts() # 1 is yes and 0 is no
```

```
HYPERTEN_BINARY
1    8642
0    2985
Name: count, dtype: int64
```

```
# subset the varaible
subset = hypertension_final_dumm[['TOTCHOLESTEROL','AGE','BMI','DIABETES','GLUCOSE','EDUC',
                                  'RESTHEARTRTE','GENDER_Male','CURSMOKE_Yes','HYPERTEN_BINARY']]
subset
```

	TOTCHOLESTEROL	AGE	BMI	DIABETES	GLUCOSE	EDUC	RESTHEARTRTE	GENDER_Male	CURSMOKE_Yes	HYPERTEN_BINARY
0	195	39	26.97	0	77	4	80	1	0	0
1	209	52	23.58	0	92	4	69	1	0	0
2	250	46	28.73	0	76	2	95	0	0	0
3	260	52	29.43	0	86	2	80	0	0	0
4	237	58	28.50	0	71	2	80	0	0	0
...	...	...	...	...	...	...	...	...	...	...
11622	173	46	19.17	0	79	3	70	1	0	1
11623	153	52	25.74	0	72	3	65	1	0	1
11624	196	39	20.91	0	80	3	85	0	1	1
11625	240	46	26.39	0	83	3	90	0	1	1
11626	252	50	24.19	0	82	3	94	0	1	1

11627 rows × 10 columns

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Designate dependent and independent variables
independent_vars = list(subset.columns)
#print(independent_vars)
independent_vars.remove('HYPERTEN_BINARY')
#print(independent_vars)
# independent variable
X = subset[independent_vars]
#print(X)
#dependent varuable
y =subset['HYPERTEN_BINARY']
#print(y)
'''
# Create training and test split
# We are using 60:20:20 ratio for train and test
# random_state is seed
X_train, X_val_test, y_train, y_val_test = train_test_split(X, y, test_size=0.4, random_state=99)
X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test, test_size=0.5, random_state=99)
print(len(X))
print(len(X_train))
print(len(X_val))
print(len(X_test))
'''
# Create training and test split
# We are using 80:20 ratio for train and test
# random_state is seed
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=99)
#Reorder the target 1 and 0

print(len(X))
print(len(X_train))
print(len(X_test))
```

```
11627
9301
2326
```

Random Forest Model

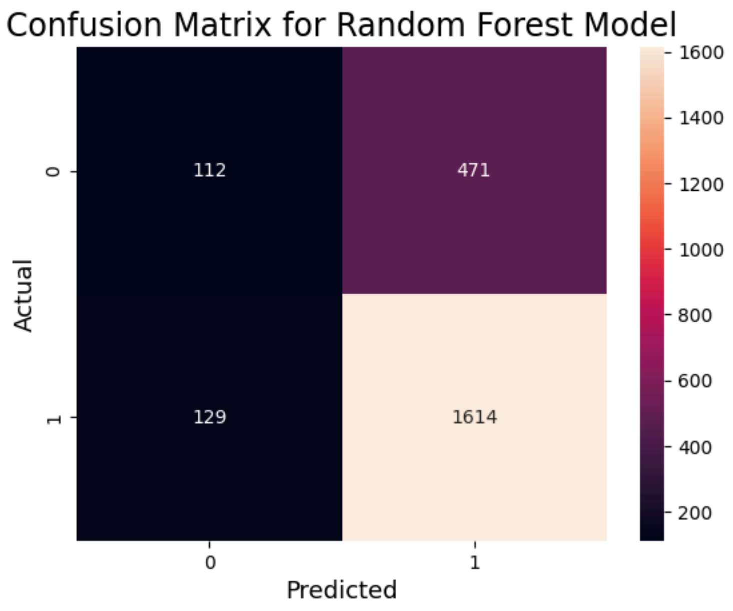
```
from sklearn.ensemble import RandomForestClassifier

# "RandomForestClassifier" for classification and "RandomForestRegressor" for continous
randomforest = RandomForestClassifier(random_state=99)
#randomforest = RandomForestClassifier(n_estimators = 23,random_state=99)
# Fit the RandomForestRegressor model
randomforest.fit(X_train, y_train)

# Use models to get predictions on the test set
pred_rf_pb = randomforest.predict(X_test)
pred_rf_pb
```

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

```
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test,pred_rf_pb )
conf_matrix
```

```
     array([[ 112,  471],
            [ 129, 1614]])
```

```
sns.heatmap(conf_matrix,
            annot=True,
            fmt='g',
            xticklabels=['0','1'],
            yticklabels=['0','1'])
plt.ylabel('Actual',fontsize=13) #y_val
plt.xlabel('Predicted',fontsize=13)#pred_rf
plt.title('Confusion Matrix for Random Forest Model',fontsize=17)
plt.show()
```

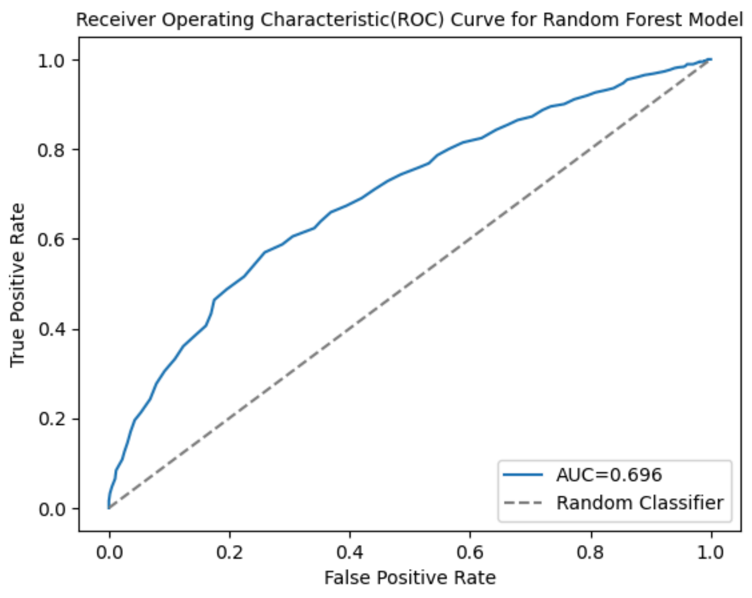


```
# import necessary libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

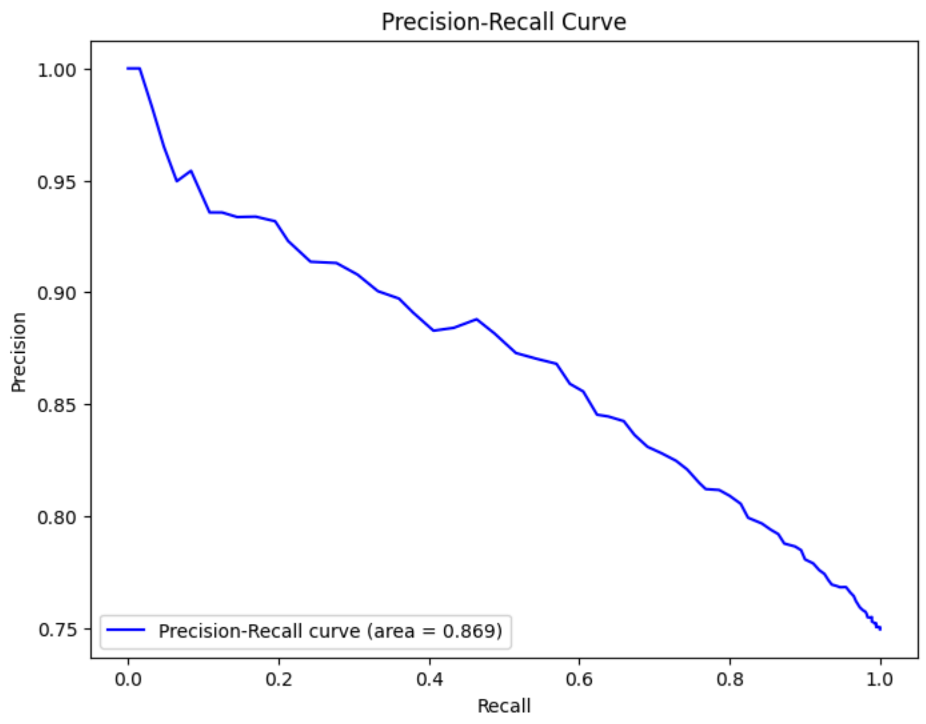
```
accuracy_rf=accuracy_score(y_test,pred_rf_pb)
precision_rf = precision_score(y_test, pred_rf_pb)
Sensitivity_recall_rf = recall_score(y_test, pred_rf_pb)
F1_score_rf = f1_score(y_test, pred_rf_pb)
# Print all values at once
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Sensitivity/Recall:", Sensitivity_recall_rf)
print("F1 Score:", F1_score_rf )
```

```
    Accuracy: 0.7420464316423044
    Precision: 0.7741007194244605
    Sensitivity/Recall: 0.9259896729776248
    F1 Score: 0.8432601880877744
```

```
# Calculate auc
from sklearn import metrics
pred_rf_test = randomforest.predict_proba(X_test)[::,1]
pred_rf_test
fpr, tpr, _ = metrics.roc_curve(y_test,pred_rf_test)
auc = metrics.roc_auc_score(y_test, pred_rf_test)

# Create ROC curve
plt.plot(fpr, tpr, label="AUC={:.3f}".format(auc))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random Classifier')  # Add dashed line for random classifier
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.title('Receiver Operating Characteristic(ROC) Curve for Random Forest Model',fontsize=10)  # Add title
plt.show()
```



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

# Assuming y_true contains the true labels and y_score contains the predicted probabilities or decision scores
precision, recall, _ = precision_recall_curve(y_test, pred_rf_test)

# Compute PR AUC
pr_auc = auc(recall, precision)
pr_auc

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='Precision-Recall curve (area = %0.3f)' % pr_auc, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



Hyperparamter Tuning Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [None, 10, 20],       # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],   # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],     # Minimum number of samples required to be at a leaf node
    'bootstrap': [True, False]         # Whether bootstrap samples are used when building trees
}

# Create a RandomForestClassifier instance
randomforest = RandomForestClassifier(random_state=99)

# Perform grid search using 5-fold cross-validation
grid_search = GridSearchCV(estimator=randomforest, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best parameters found
print("Best parameters:", grid_search.best_params_)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100],  # Number of trees in the forest
    'max_depth': [10],       # Maximum depth of the tree
    'min_samples_split': [2],   # Minimum number of samples required to split an internal node
    'min_samples_leaf': [2],     # Minimum number of samples required to be at a leaf node
    'bootstrap': [True]         # Whether bootstrap samples are used when building trees
}

# Create a RandomForestClassifier instance
randomforest = RandomForestClassifier(random_state=99)

# Perform grid search using 5-fold cross-validation
grid_search = GridSearchCV(estimator=randomforest, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Print the best parameters found
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'bootstrap': True, 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
```

```
# Use models to get predictions on the test set
pred_rf_tune = grid_search.predict(X_test)
pred_rf_tune
```

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

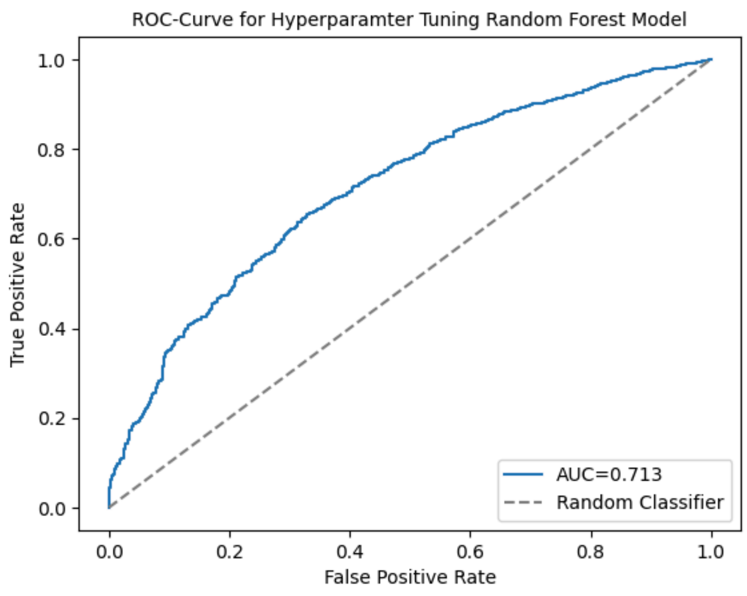
```
# Generate confusion matrix
conf_matrix_tune = confusion_matrix(y_test,pred_rf_tune )
conf_matrix_tune
```

```
array([[  84,  499],
       [  68, 1675]])
```

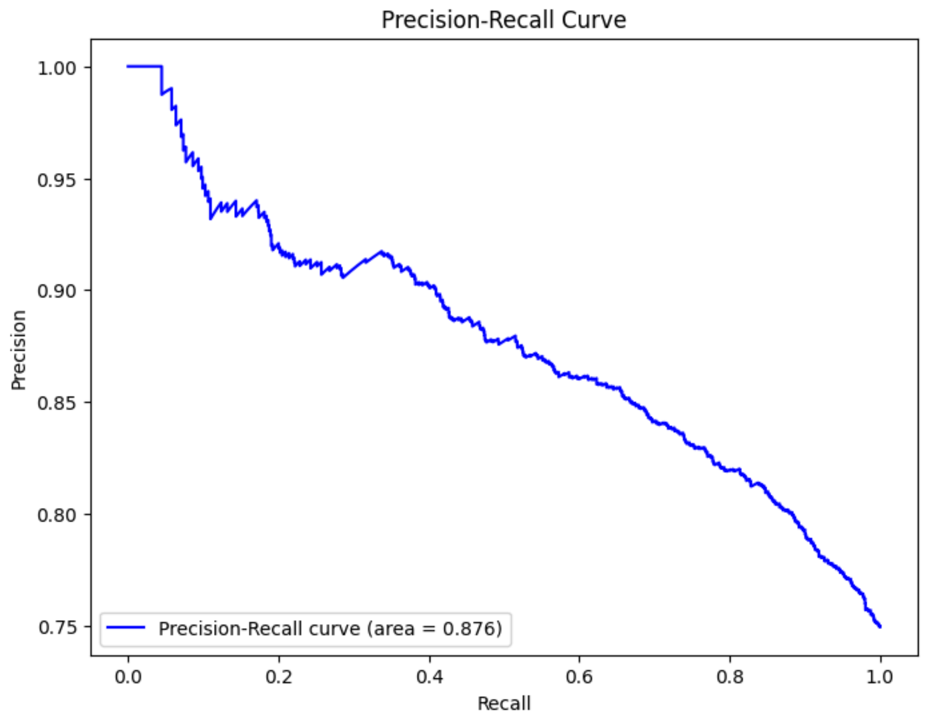
```
accuracy_rf_tune=accuracy_score(y_test,pred_rf_tune )
precision_rf_tune = precision_score(y_test, pred_rf_tune )
Sensitivity_recall_rf_tune = recall_score(y_test, pred_rf_tune )
F1_score_rf_tune = f1_score(y_test, pred_rf_tune )
# Print all values at once
print("Accuracy:", accuracy_rf_tune)
print("Precision:", precision_rf_tune)
print("Sensitivity/Recall:", Sensitivity_recall_rf_tune)
print("F1 Score:", F1_score_rf_tune )
```

```
Accuracy: 0.7562338779019776
Precision: 0.7704691812327507
Sensitivity/Recall: 0.9609868043602984
F1 Score: 0.8552463620117438
```

```
# Calculate auc
from sklearn import metrics
pred_rf_tune_test = grid_search.predict_proba(X_test)[::,1]
pred_rf_tune_test
fpr_rf, tpr_rf, _ = metrics.roc_curve(y_test,pred_rf_tune_test)
auc_rf = metrics.roc_auc_score(y_test, pred_rf_tune_test)
auc_rf
# Create ROC curve
plt.plot(fpr_rf, tpr_rf, label="AUC={:.3f}".format(auc_rf))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random Classifier')  # Add dashed line for random classifier
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.title('ROC-Curve for Hyperparamter Tuning Random Forest Model',fontsize=10)  # Add title
plt.show()
```



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

# Assuming y_true contains the true labels and y_score contains the predicted probabilities or decision scores
precision_rf, recall_rf, _ = precision_recall_curve(y_test, pred_rf_tune_test)

# Compute PR AUC
pr_auc_rf = auc(recall_rf, precision_rf)
pr_auc_rf

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_rf, precision_rf, label='Precision-Recall curve (area = %0.3f)' % pr_auc_rf, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



```
# Get feature importances
feature_importances = grid_search.best_estimator_.feature_importances_

# Create a dictionary to store feature importances with corresponding feature names
feature_importance_dict = dict(zip(X_train.columns, feature_importances))

# Sort the dictionary by importance values in descending order
sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

# Print or visualize the sorted feature importances
for feature, importance in sorted_feature_importance:
    print(f"{feature}: {importance}")
```

```
BMI: 0.28424639562422277
AGE: 0.1848971040309558
TOTCHOLESTEROL: 0.16694374126964467
GLUCOSE: 0.12838248394010984
RESTHEARTRTE: 0.12689155582438766
EDUC: 0.05183121471458326
CURSMOKE_Yes: 0.026219409452613467
GENDER_Male: 0.024482692327189165
DIABETES: 0.006105402816293291
```

Logestic Regression Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score
import pandas as pd
# Train Logistic Regression model
model_logistic = LogisticRegression()
model_logistic.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
# Use models to get predictions on the test set
pred_logistic = model_logistic.predict(X_test)
pred_logistic
```

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

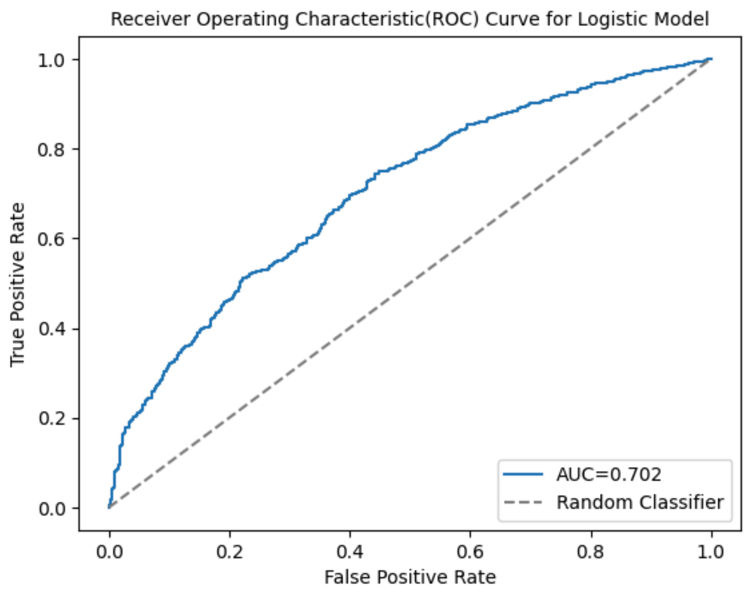
```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test,pred_logistic )
conf_matrix
```

```
array([[  48,  535],
       [  37, 1706]])
```

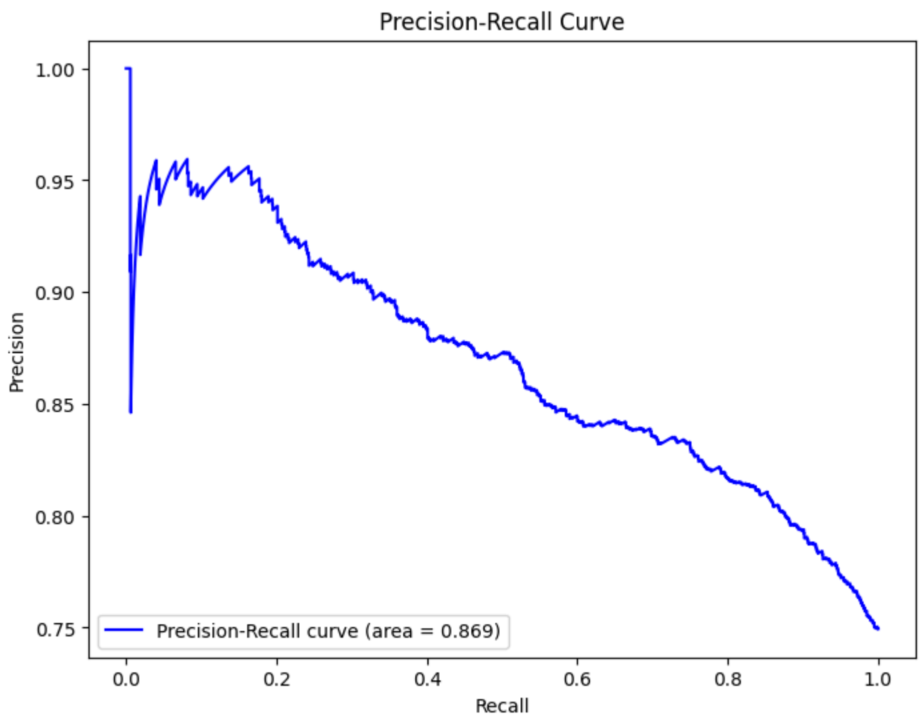
```
accuracy_logistic=accuracy_score(y_test,pred_logistic)
precision_logistic = precision_score(y_test, pred_logistic)
Sensitivity_recall_logistic = recall_score(y_test, pred_logistic)
F1_score_logistic = f1_score(y_test, pred_logistic)
# Print all values at once
print("Accuracy:", accuracy_logistic)
print("Precision:", precision_logistic)
print("Sensitivity/Recall:", Sensitivity_recall_logistic)
print("F1 Score:", F1_score_logistic )
```

```
Accuracy: 0.7540842648323302
Precision: 0.7612672913877733
Sensitivity/Recall: 0.97877223178428
F1 Score: 0.856425702811245
```

```
# Calculate auc
from sklearn import metrics
pred_logistic_test = model_logistic.predict_proba(X_test)[::,1]
pred_logistic_test
fpr_log, tpr_log, _ = metrics.roc_curve(y_test,pred_logistic_test)
auc_log = metrics.roc_auc_score(y_test, pred_logistic_test)
auc_log
# Create ROC curve
plt.plot(fpr_log, tpr_log, label="AUC={:.3f}".format(auc_log))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random Classifier')  # Add dashed line for random classifier
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.title('Receiver Operating Characteristic(ROC) Curve for Logistic Model',fontsize=10)  # Add title
plt.show()
```



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

# Assuming y_true contains the true labels and y_score contains the predicted probabilities or decision scores
precision_log, recall_log, _ = precision_recall_curve(y_test, pred_logistic_test)

# Compute PR AUC
pr_auc_log = auc(recall_log, precision_log)
pr_auc_log

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_log, precision_log, label='Precision-Recall curve (area = %0.3f)' % pr_auc_log, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



Naive_Bayes Model

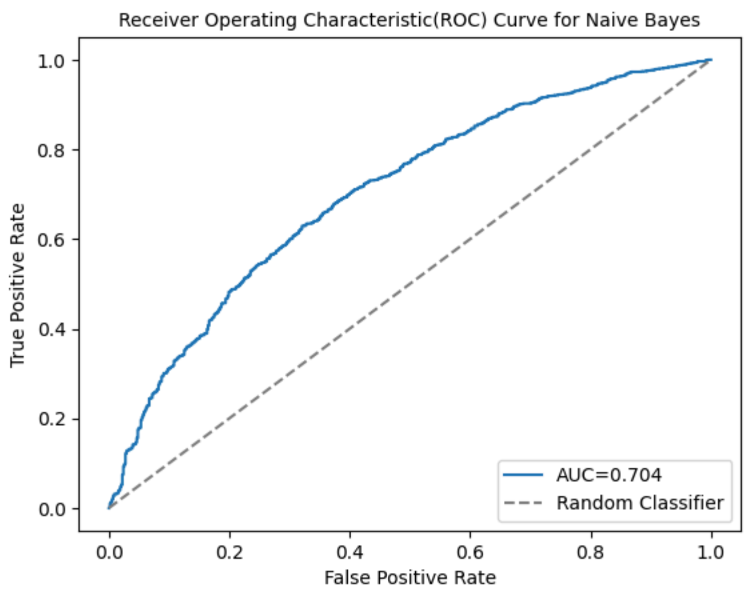
```
from sklearn.naive_bayes import GaussianNB
# Train Naive Bayes classifier
gnb = GaussianNB()
model_naive_bayes = gnb.fit(X_train, y_train)

# Predict the X_test
predictions = model_naive_bayes.predict(X_test)


# Confusion matrix
print(confusion_matrix(y_test, predictions))
```

```
[[ 348  235]
 [ 515 1228]]
```

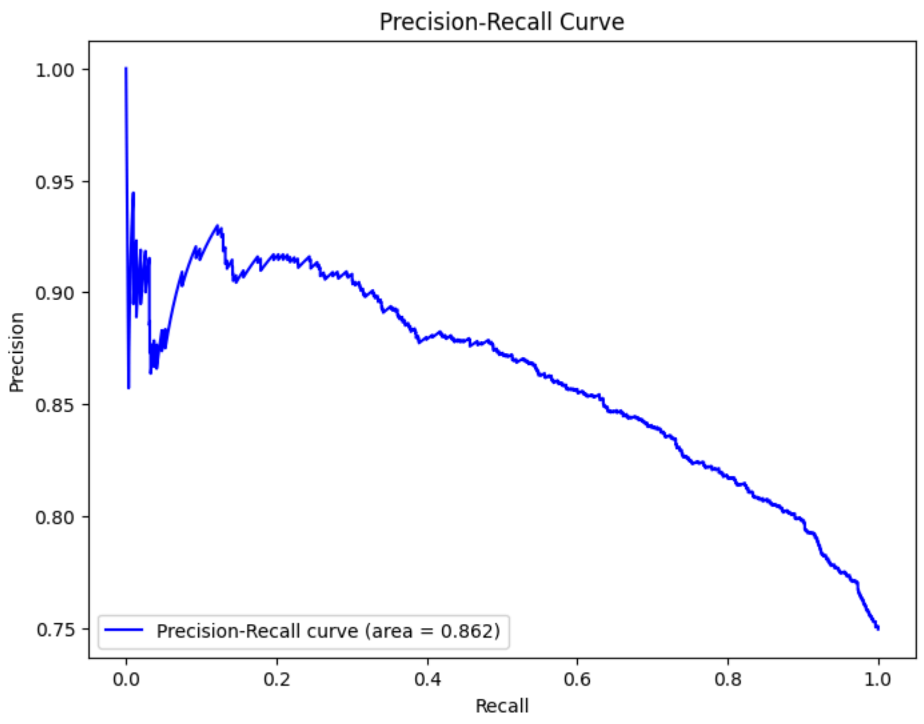
```
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
Sensitivity_recall = recall_score(y_test, predictions)
F1_score = f1_score(y_test, predictions)
# Print all values
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity/Recall:", Sensitivity_recall)
print("F1 Score:", F1_score)
```

```
Accuracy: 0.6775580395528805
Precision: 0.8393711551606289
Sensitivity/Recall: 0.7045324153757889
F1 Score: 0.7660636306924516
```

```
# Calculate auc
from sklearn import metrics
pred_naive_bayes_test = gnb.predict_proba(X_test)[::,1]
pred_naive_bayes_test
fpr_naive, tpr_naive, _ = metrics.roc_curve(y_test,pred_naive_bayes_test)
auc_naive = metrics.roc_auc_score(y_test, pred_naive_bayes_test)
auc_naive
# Create ROC curve
plt.plot(fpr_naive, tpr_naive, label="AUC={:.3f}".format(auc_naive))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random Classifier')  # Add dashed line for random classifier
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.title('Receiver Operating Characteristic(ROC) Curve for Naive Bayes',fontsize=10)  # Add title
plt.show()
```



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

# Assuming y_true contains the true labels and y_score contains the predicted probabilities or decision scores
precision_naive, recall_naive, _ = precision_recall_curve(y_test, pred_naive_bayes_test)

# Compute PR AUC
pr_auc_naive = auc(recall_naive, precision_naive)
pr_auc_naive

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_naive, precision_naive, label='Precision-Recall curve (area = %0.3f)' % pr_auc_naive, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



XGBoost Model

```
!pip install scikit-optimize
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.1-py2.py3-none-any.whl (107 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.7/107.7 kB 1.1 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.4.0)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-24.4.0-py3-none-any.whl (24 kB)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.25.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.0)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from pyaml>=16.9->scikit-optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikit-optimiz
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-24.4.0 scikit-optimize-0.10.1
```

```
!pip install hyperopt
```

```
Requirement already satisfied: hyperopt in /usr/local/lib/python3.10/dist-packages (0.2.7)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from hyperopt) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from hyperopt) (1.11.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from hyperopt) (1.16.0)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from hyperopt) (3.3)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from hyperopt) (0.18.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from hyperopt) (4.66.2)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from hyperopt) (2.2.1)
Requirement already satisfied: py4j in /usr/local/lib/python3.10/dist-packages (from hyperopt) (0.10.9.7)
```

```
import warnings
warnings.filterwarnings("ignore")

import xgboost as xgb
import numpy as np
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
from sklearn.metrics import make_scorer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# Define the objective function for hyperopt
# Use k-fold cross-validation for evaluation during training
def objective(params):
    model = xgb.XGBClassifier(**params)
    kf = KFold(n_splits=5, shuffle=True, random_state=42)  # Define k-fold cross-validation
    scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='roc_auc')
    mean_auc = scores.mean()

    # Return the negative mean AUC score to be maximized, and the current hyperparameters
    return {'loss': -mean_auc, 'status': STATUS_OK, 'params': params}

# Define the search space for hyperparameters
space = {
    'n_estimators': hp.choice('n_estimators', range(50, 500)),
    'max_depth': hp.choice('max_depth', range(1, 10)),
    'learning_rate': hp.uniform('learning_rate', 0.01, 0.5),
    'subsample': hp.uniform('subsample', 0.5, 1),
    'gamma': hp.uniform('gamma', 0, 0.5),
    'colsample_bytree': hp.uniform('colsample_bytree', 0.5, 1),
    'min_child_weight': hp.choice('min_child_weight', range(1, 10)),
}
# Start the hyperparameter search
trials = Trials()
best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=30, trials=trials,
            rstate=np.random.default_rng(99))

# Print the best parameters
print("Best hyperparameters:", best)
```

```
100%|██████████| 30/30 [01:10<00:00,  2.34s/trial, best loss: -0.7092969917016256]
Best hyperparameters: {'colsample_bytree': 0.6572788516713307, 'gamma': 0.3327784193267687, 'learning_rate': 0.034278538204298596, 'max_
```

```
# Function to convert indices to values for categorical hyperparameters
def convert_categorical_params(best_params):
    best_params['n_estimators'] = range(50, 500)[best_params['n_estimators']]
    best_params['max_depth'] = range(1, 10)[best_params['max_depth']]
    best_params['min_child_weight'] = range(1, 10)[best_params['min_child_weight']]

    return best_params

# Convert categorical hyperparameters
best_params_converted = convert_categorical_params(best)

# Train the best model
best_model = xgb.XGBClassifier(**best_params_converted)
best_model.fit(X_train, y_train)
```

```
▾                                   XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.6572788516713307, device=None,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0.3327784193267687,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.034278538204298596,
              max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=8, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=204, n_jobs=None,
```

```
# Use models to get predictions on the test set
pred_xgboost = best_model.predict(X_test)
pred_xgboost
```

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

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test,pred_xgboost)
conf_matrix
```

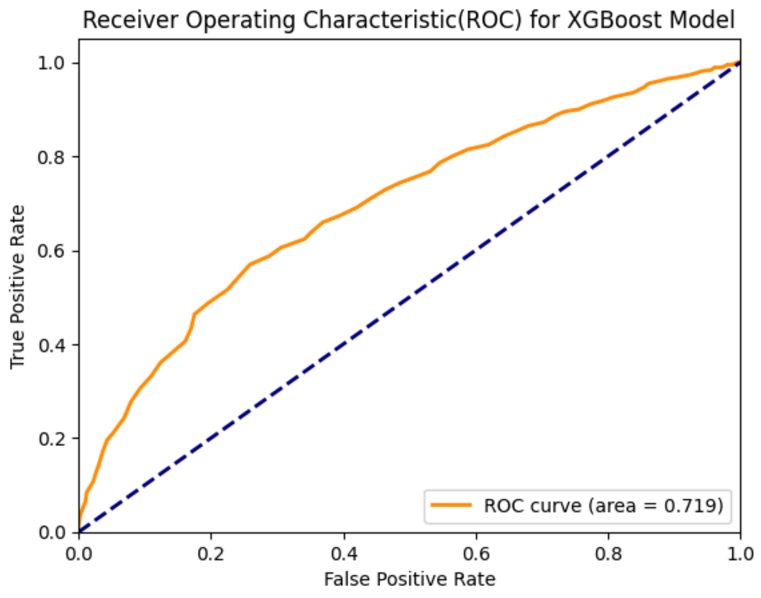
```
array([[  87,  496],
       [  77, 1666]])
```

```
accuracy_xgboost=accuracy_score(y_test,pred_xgboost)
precision_xgboost = precision_score(y_test, pred_xgboost)
Sensitivity_recall_xgboost = recall_score(y_test, pred_xgboost)
F1_score_xgboost = f1_score(y_test, pred_xgboost)
# Print all values at once
print("Accuracy:", accuracy_xgboost)
print("Precision:", precision_xgboost)
print("Sensitivity/Recall:", Sensitivity_recall_xgboost)
print("F1 Score:", F1_score_xgboost )
```

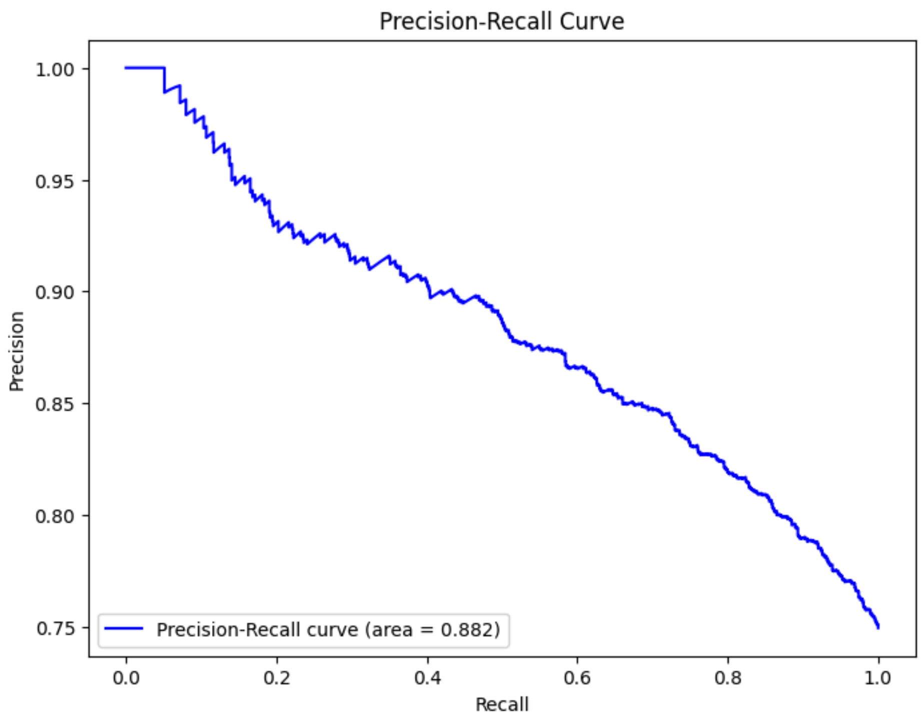
```
Accuracy: 0.7536543422184007
Precision: 0.7705827937095282
Sensitivity/Recall: 0.9558232931726908
F1 Score: 0.8532650448143405
```

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

# Predict probabilities for the test set
y_test_proba = best_model.predict_proba(X_test)[:, 1]

# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_test_proba)
print(auc_score)

# Calculate the ROC curve points
fpr_xgb, tpr_xgb, thresholds = roc_curve(y_test, y_test_proba)

# Calculate the AUC score
roc_auc = auc(fpr_xgb, tpr_xgb)
roc_auc

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.3f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, 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) for XGBoost Model')
plt.legend(loc="lower right")

plt.show()
```

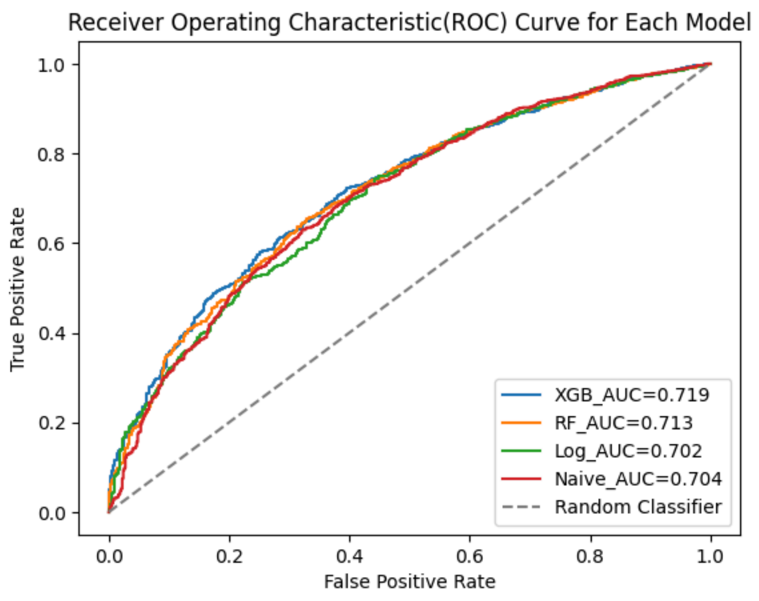
```
0.7191431740192822
```



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

# Assuming y_true contains the true labels and y_score contains the predicted probabilities or decision scores
precision_xgb, recall_xgb, _ = precision_recall_curve(y_test, y_test_proba)

# Compute PR AUC
pr_auc_xgb = auc(recall_xgb, precision_xgb)
pr_auc_xgb

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_xgb, precision_xgb, label='Precision-Recall curve (area = %0.3f)' % pr_auc_xgb, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



```
# Create ROC curve for all model
plt.plot(fpr_xgb, tpr_xgb, label="XGB_AUC={:.3f}".format(roc_auc))
plt.plot(fpr_rf, tpr_rf, label="RF_AUC={:.3f}".format(auc_rf))
plt.plot(fpr_log, tpr_log, label="Log_AUC={:.3f}".format(auc_log))
plt.plot(fpr_naive, tpr_naive, label="Naive_AUC={:.3f}".format(auc_naive))
plt.plot([0, 1], [0, 1], '--', color='gray', label='Random Classifier')  # Add dashed line for random classifier
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.title('Receiver Operating Characteristic(ROC) Curve for Each Model',fontsize=12)  # Add title
plt.show()
```



```
label="XGB_AUC={:.3f}".format(roc_auc)
Precision-Recall AUC
```

```
# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_xgb, precision_xgb, label='Precision-Recall AUC_xgb={:0.3f}'.format (pr_auc_xgb))
plt.plot(recall_rf, precision_rf, label='Precision-Recall AUC_rf={:0.3f}'.format(pr_auc_rf))
plt.plot(recall_log, precision_log, label='Precision-Recall AUC_log={:0.3f}'.format (pr_auc_log))
plt.plot(recall_naive, precision_naive, label='Precision-Recall AUC_naive={:0.3f}'.format (pr_auc_naive))
plt.xlabel('Recall')
```