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

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

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

Data-Analysis

```
#loading the diabetes dataset to pandas dataframe
data=pd.read_csv('/content/gdrive/MyDrive/My_Final_Project/health care diabetes - health care diabetes.csv')
```

```
#printing first five rows of the dataset
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

```
#printing first five rows of the dataset
data.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
763	10	101	76	48	180	32.9	0.
764	2	122	70	27	0	36.8	0.
765	5	121	72	23	112	26.2	0.
766	1	126	60	0	0	30.1	0.
767	1	93	70	31	0	30.4	0.

```
#no of rows and columns of the data
data.shape
```

```
(768, 9)
```

```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

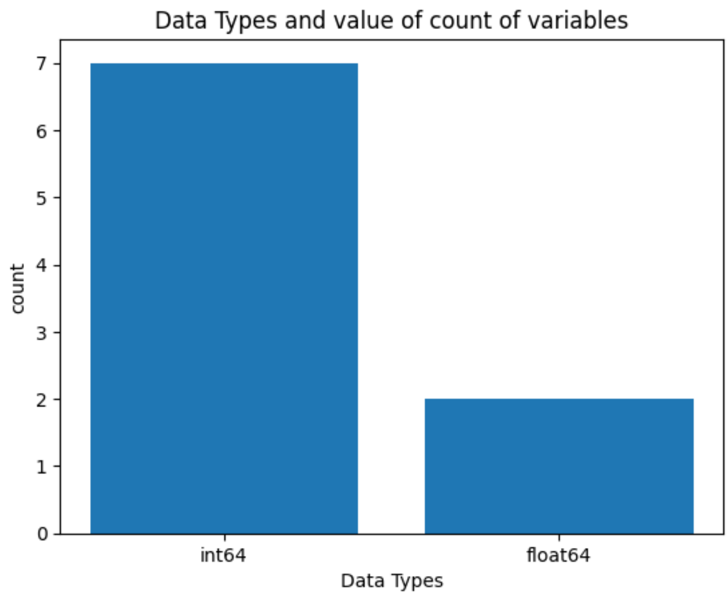
```
data['Outcome'].value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
dtype_counts = data.dtypes.value_counts()
print(dtype_counts)
```

```
int64      7
float64    2
dtype: int64
```

```
plt.bar(dtype_counts.index.astype(str), dtype_counts.values)
plt.xlabel('Data Types')
plt.ylabel('count')
plt.title('Data Types and value of count of variables')
plt.show()
```




```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
data.isna().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Columns with the value 0 is the missing value, so missing value treatment to be done.

```
columns_to_check=['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
```

```
for column in columns_to_check:
    data[column] = data[column].replace(0, float('nan'))

print(data)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0           35.0      NaN  33.6
1              1     85.0           66.0           29.0      NaN  26.6
2              8    183.0           64.0            NaN      NaN  23.3
3              1     89.0           66.0           23.0     94.0  28.1
4              0    137.0           40.0           35.0    168.0  43.1
..           ...      ...            ...            ...      ...   ...
763           10    101.0           76.0           48.0    180.0  32.9
764            2    122.0           70.0           27.0      NaN  36.8
765            5    121.0           72.0           23.0    112.0  26.2
766            1    126.0           60.0            NaN      NaN  30.1
767            1     93.0           70.0           31.0      NaN  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

```
data.isna().sum()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

The columns with the missing value are Glucose with 5 missing value,Bloodpressure with 35 missing value, SkinThickness with 227 missing value,Insulin with 374 missing value, and BMI with 11 missing value.

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148.0	72.0	35.0	NaN	33.6	0.62
1	1	85.0	66.0	29.0	NaN	26.6	0.35
2	8	183.0	64.0	NaN	NaN	23.3	0.67
3	1	89.0	66.0	23.0	94.0	28.1	0.16
4	0	137.0	40.0	35.0	168.0	43.1	2.28

```
data.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
763	10	101.0	76.0	48.0	180.0	32.9	0.
764	2	122.0	70.0	27.0	NaN	36.8	0.
765	5	121.0	72.0	23.0	112.0	26.2	0.
766	1	126.0	60.0	NaN	NaN	30.1	0.
767	1	93.0	70.0	31.0	NaN	30.4	0.

```
#replace the null column with mean
import numpy as np
data_1 = data.replace(to_replace= np.nan,value=data[columns_to_check].median())
```

```
data_1
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148.0	72.0	35.0	125.0	33.6	0.
1	1	85.0	66.0	29.0	125.0	26.6	0.
2	8	183.0	64.0	29.0	125.0	23.3	0.
3	1	89.0	66.0	23.0	94.0	28.1	0.
4	0	137.0	40.0	35.0	168.0	43.1	2.
...	...	...	...	...	...	...	
763	10	101.0	76.0	48.0	180.0	32.9	0.
764	2	122.0	70.0	27.0	125.0	36.8	0.
765	5	121.0	72.0	23.0	112.0	26.2	0.
766	1	126.0	60.0	29.0	125.0	30.1	0.
767	1	93.0	70.0	31.0	125.0	30.4	0.

768 rows × 9 columns

```
data_1.info()
```

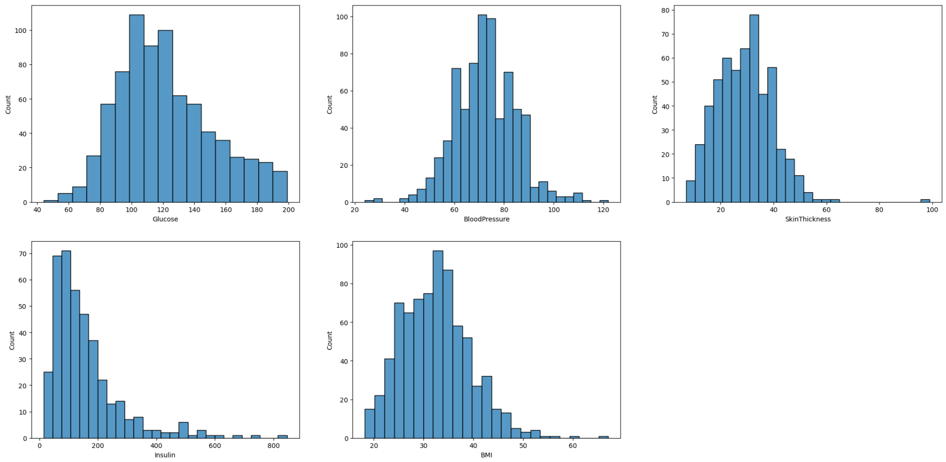
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

```
data_1.isna().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Visualization of the dataset

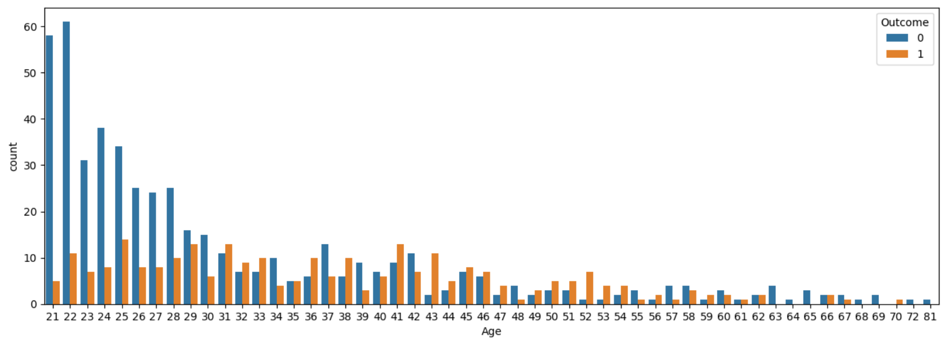
```
d = data.iloc[:,1:6]
plt.figure(figsize=(25,25))
for i, column in enumerate(d.columns, 1):
    plt.subplot(4,3,i)
    sns.histplot(d[column])
```



observation:

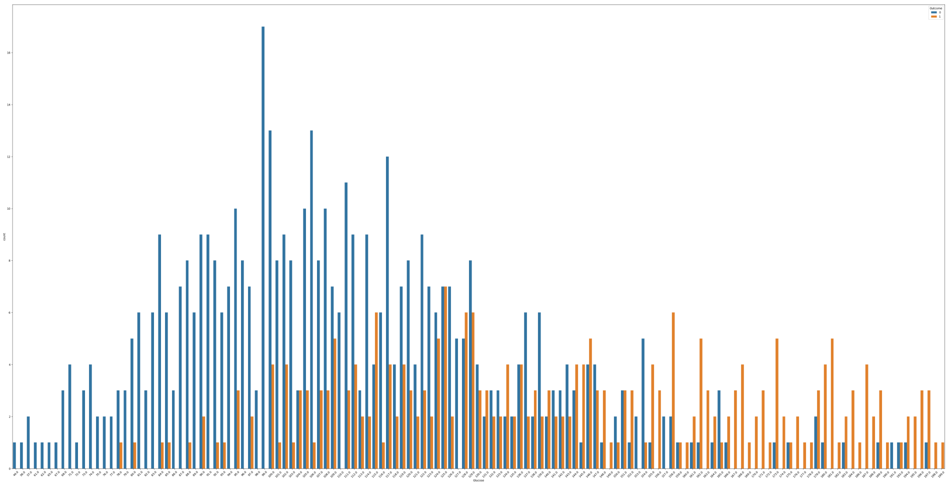
- Glucose concentration is high between 90-125
- BloodPressure high between 65-80
- Skin thickness more between 20-40
- Insulin high near 150
- BMI high between 29-37

```
#plot outcome s vs age
plt.figure(figsize=(15,5))
sns.countplot(x='Age',hue='Outcome',data=data_1)
plt.show()
```



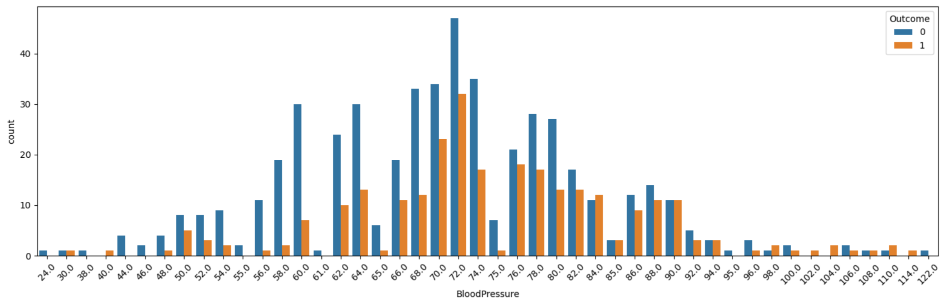
Female Patients with age between 30-55 suffer from diabetics problem more.

```
#plot outcome vs Glucose
plt.figure(figsize=(60,30))
sns.countplot(x='Glucose',hue='Outcome',data=data_1)
plt.xticks(rotation=45)
plt.show()
```



Diabetic patients with high glucose level.

```
#outcome vs BloodPressure
plt.figure(figsize=(18,5))
sns.countplot(x='BloodPressure',hue='Outcome',data=data_1)
plt.xticks(rotation=45)
plt.show()
```

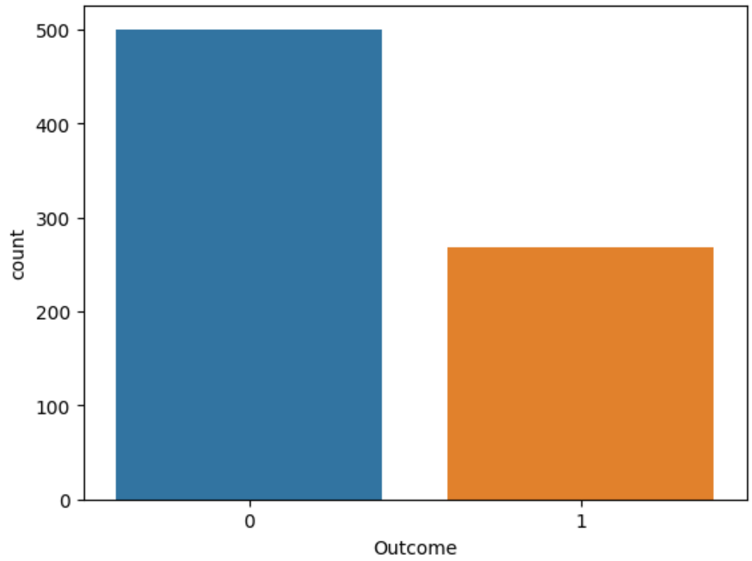



```
data_1['Outcome'].value_counts()
```

```
0    500
1    268
```

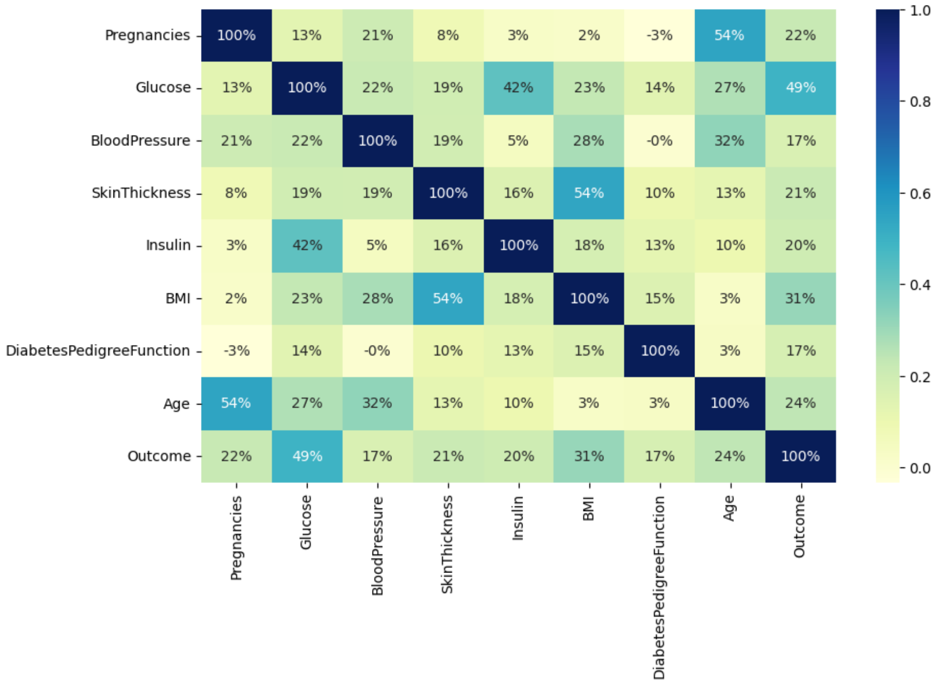
```
Name: Outcome, dtype: int64
```

```
sns.countplot(x=data_1['Outcome'])
plt.show()
```



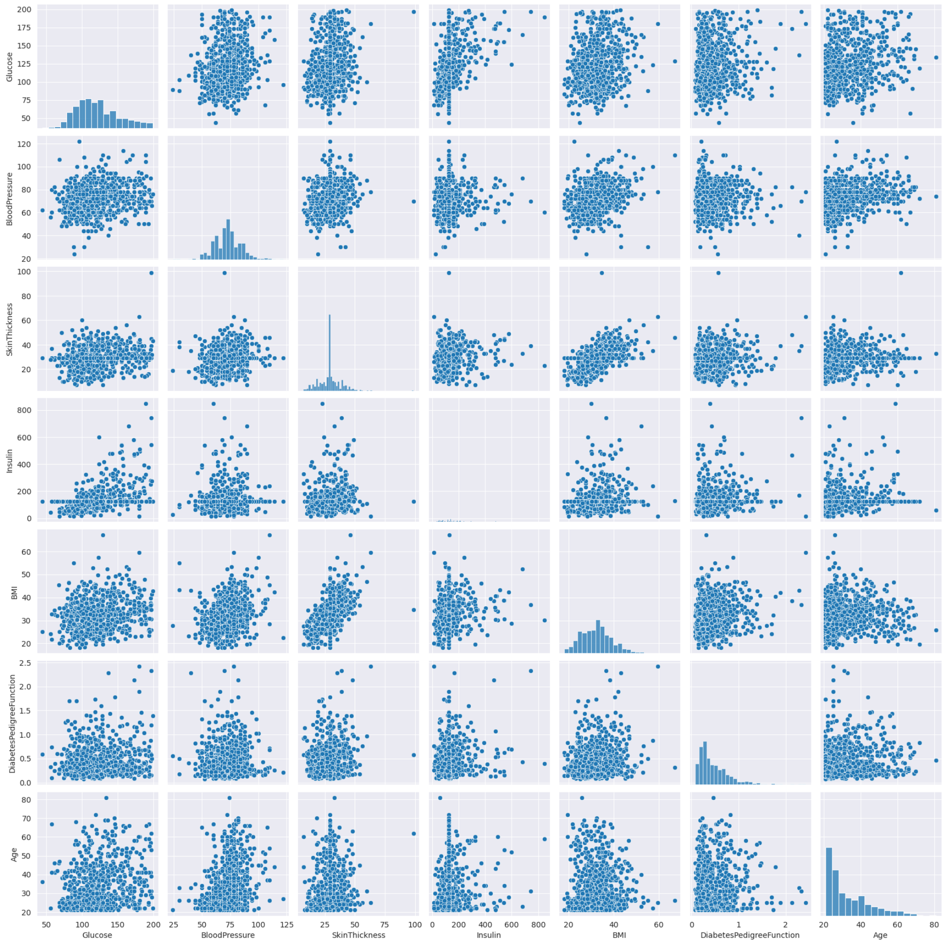
268 Diabetic and 500 Non-Diabetic.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
sns.heatmap(data_1.corr(), annot=True, fmt='.0%',cmap='YlGnBu')
plt.show()
```




```
cols_to_plot =['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
```

```
sns.set_style('darkgrid')
sns.pairplot(data_1[cols_to_plot])
plt.show()
```


Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age

Data split

```
#segregate data into dependent and independent variables
#independent variables
x=data_1.drop(columns='Outcome')
y=data_1.Outcome
```

Solving the data imbalance with smote.

```
from imblearn.over_sampling import SMOTE
smk = SMOTE()
x_train_smote,y_train_smote=smk.fit_resample(x,y)
```

```
from collections import Counter
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_train_smote)))
```

```
Original dataset shape Counter({0: 500, 1: 268})
Resampled dataset shape Counter({1: 500, 0: 500})
```

```
# split the data into train and test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x_train_smote,y_train_smote,test_size=0.2,random_state=2)
```

```
#Standardise data(so that no biased occur towards one data)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
scaler.fit(X_test)

x_train_std = scaler.transform(X_train)
x_test_std = scaler.transform(X_test)
```

```
x_train_std.shape
```

```
(800, 8)
```

```
x_test_std.shape
```

```
(200, 8)
```

Training the model

- LogisticRegression

```
#Importing Logistic Regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
lr_model= LogisticRegression(max_iter=1000)
```

```
#Fit the model in train and test data
lr_model.fit(x_train_std,y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred=lr_model.predict(x_test_std)
```

```
score_lg = round(accuracy_score(y_pred,y_test)*100,2)
print("The accuracy score achieved using Logistic Regressionis : "+str(score_lg)+"%")
```

```
The accuracy score achieved using Logistic Regressionis : 75.5%
```

```
lr_scores = lr_model.predict_proba(x_test_std)[:,1]
```

```
lr_scores
```

```
array([0.55642143, 0.32741626, 0.65750061, 0.8453835 , 0.84695024,
       0.75325147, 0.28706516, 0.88406636, 0.85309893, 0.17925646,
       0.08965054, 0.67561535, 0.63474523, 0.17592044, 0.92782503,
       0.12781439, 0.15516056, 0.44966683, 0.81524512, 0.53781764,
       0.6269031 , 0.59738565, 0.82421305, 0.82175065, 0.13889867,
       0.0312921 , 0.04676597, 0.78802149, 0.681653  , 0.35441014,
       0.05190196, 0.78112729, 0.19031265, 0.89984966, 0.33055733,
       0.09060499, 0.65835502, 0.64772087, 0.98465181, 0.48647143,
       0.10747914, 0.26899277, 0.9332487 , 0.08168614, 0.01869934,
       0.98801018, 0.10698925, 0.39159776, 0.73076926, 0.69748841,
       0.29586477, 0.34585917, 0.25772559, 0.66494482, 0.9461863 ,
       0.89840535, 0.14286397, 0.05142824, 0.41121657, 0.23390992,
       0.13792821, 0.90857675, 0.34665348, 0.57405871, 0.97368366,
       0.03452611, 0.11625928, 0.88279257, 0.91315272, 0.16236432,
       0.18386716, 0.06872657, 0.25197667, 0.36052394, 0.94350965,
       0.16771139, 0.91378877, 0.93247552, 0.82002319, 0.27480445,
       0.39019035, 0.04639515, 0.79014424, 0.93130089, 0.65349093,
       0.48261191, 0.79773822, 0.10317224, 0.53291259, 0.15286484,
       0.797792  , 0.56421464, 0.59900014, 0.10946968, 0.10866258,
       0.05982851, 0.96811165, 0.73451159, 0.6695032 , 0.29248205,
       0.24355108, 0.81330121, 0.76196822, 0.22563977, 0.34293727,
       0.78118489, 0.05565396, 0.08947414, 0.321412  , 0.24308706,
       0.9297281 , 0.46350531, 0.52638351, 0.79650871, 0.71741225,
       0.95999059, 0.94522217, 0.19935154, 0.44429933, 0.8357235 ,
       0.28362439, 0.11706013, 0.46933628, 0.45654646, 0.78381007,
       0.89884049, 0.66190356, 0.22585277, 0.65875612, 0.96621933,
       0.79082032, 0.95249763, 0.95214987, 0.41202054, 0.46663067,
       0.0802649 , 0.40736588, 0.03682107, 0.09992511, 0.16942909,
       0.26954294, 0.0623374 , 0.27118504, 0.93355993, 0.46158176,
       0.04326205, 0.95557195, 0.814037  , 0.66445202, 0.85409149,
       0.86972591, 0.63679691, 0.33724205, 0.99867617, 0.50609877,
       0.42922782, 0.32787935, 0.06926825, 0.13532678, 0.69916917,
       0.30031612, 0.79346762, 0.74276877, 0.04964473, 0.14859063,
       0.93453841, 0.20452578, 0.4560791 , 0.82607834, 0.84911162,
       0.32281482, 0.68192533, 0.45105175, 0.91159115, 0.55746892,
       0.05820184, 0.06125643, 0.11931183, 0.59351215, 0.80459794,
       0.88614076, 0.12955227, 0.91703052, 0.80927931, 0.12309253,
       0.28094622, 0.59848827, 0.82553998, 0.42417736, 0.1740318 ,
       0.76537444, 0.39328796, 0.57656023, 0.30332428, 0.37459784,
       0.25658219, 0.37053979, 0.64778102, 0.25256197, 0.45427322])
```

```
from sklearn.metrics import roc_curve
```

```
fpr,tpr, thresholds = roc_curve(y_test,lr_scores)
```

```
thresholds
```

```
array([1.99867617, 0.99867617, 0.89984966, 0.89884049, 0.88614076,
       0.88279257, 0.84695024, 0.8453835 , 0.82002319, 0.81524512,
       0.814037  , 0.81330121, 0.78802149, 0.78381007, 0.78118489,
       0.78112729, 0.76196822, 0.75325147, 0.71741225, 0.69916917,
       0.6695032 , 0.66494482, 0.65875612, 0.65835502, 0.59848827,
       0.59351215, 0.57656023, 0.57405871, 0.56421464, 0.55746892,
       0.55642143, 0.53781764, 0.53291259, 0.52638351, 0.46933628,
       0.46350531, 0.45654646, 0.4560791 , 0.44966683, 0.42922782,
       0.41121657, 0.40736588, 0.39159776, 0.37459784, 0.35441014,
       0.34665348, 0.34585917, 0.34293727, 0.33724205, 0.33055733,
       0.32787935, 0.30031612, 0.29586477, 0.28362439, 0.28094622,
       0.27480445, 0.27118504, 0.26954294, 0.25658219, 0.24308706,
       0.23390992, 0.17925646, 0.17592044, 0.16771139, 0.16236432,
       0.15286484, 0.14859063, 0.14286397, 0.13889867, 0.12309253,
       0.11931183, 0.01869934])
```

```
fpr
```

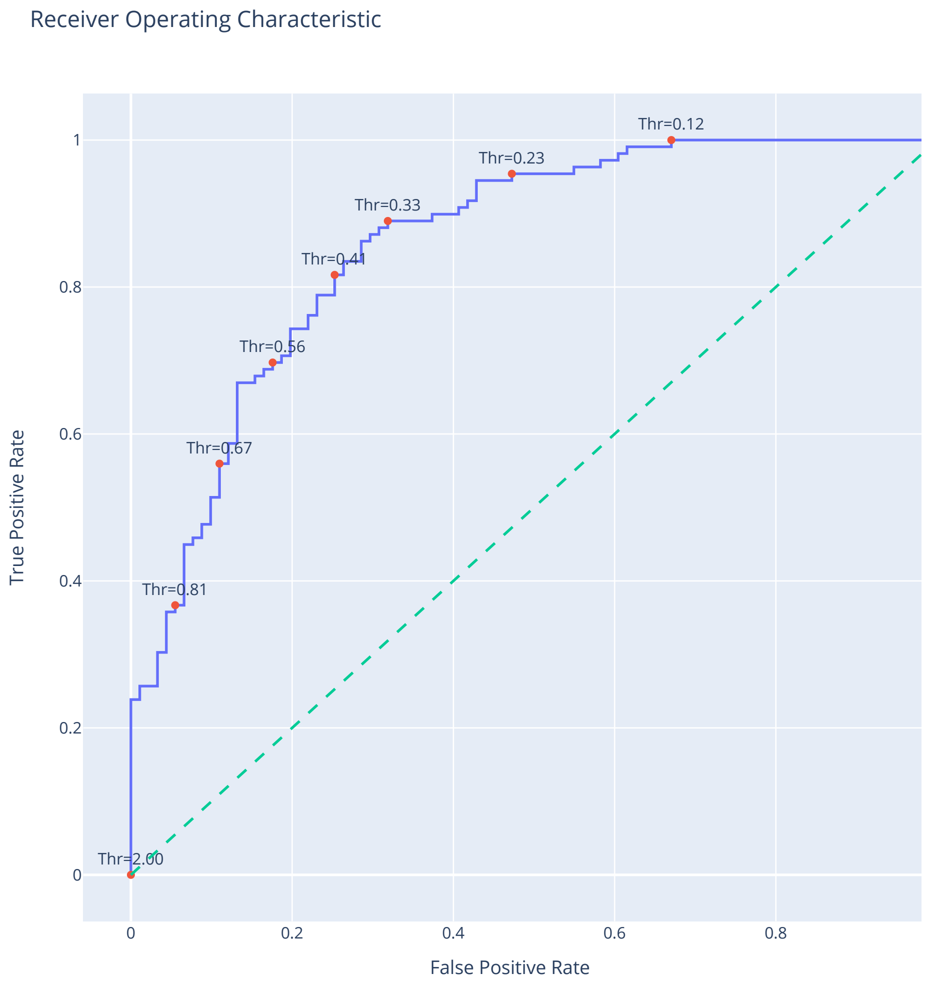
```
array([0.        , 0.        , 0.        , 0.01098901, 0.01098901,
       0.03296703, 0.03296703, 0.04395604, 0.04395604, 0.05494505,
       0.05494505, 0.06593407, 0.06593407, 0.07692308, 0.07692308,
       0.08791209, 0.08791209, 0.0989011 , 0.0989011 , 0.10989011,
       0.10989011, 0.12087912, 0.12087912, 0.13186813, 0.13186813,
       0.15384615, 0.15384615, 0.16483516, 0.16483516, 0.17582418,
       0.17582418, 0.18681319, 0.18681319, 0.1978022 , 0.1978022 ,
       0.21978022, 0.21978022, 0.23076923, 0.23076923, 0.25274725,
       0.25274725, 0.26373626, 0.26373626, 0.28571429, 0.28571429,
       0.2967033 , 0.2967033 , 0.30769231, 0.30769231, 0.31868132,
       0.31868132, 0.37362637, 0.37362637, 0.40659341, 0.40659341,
       0.41758242, 0.41758242, 0.42857143, 0.42857143, 0.47252747,
       0.47252747, 0.54945055, 0.54945055, 0.58241758, 0.58241758,
```

```
       0.6043956 , 0.6043956 , 0.61538462, 0.61538462, 0.67032967,
       0.67032967, 1.        ])
```

```
tpr
```

```
array([0.        , 0.00917431, 0.23853211, 0.23853211, 0.25688073,
       0.25688073, 0.30275229, 0.30275229, 0.35779817, 0.35779817,
       0.36697248, 0.36697248, 0.44954128, 0.44954128, 0.4587156 ,
       0.4587156 , 0.47706422, 0.47706422, 0.51376147, 0.51376147,
       0.55963303, 0.55963303, 0.58715596, 0.58715596, 0.66972477,
       0.66972477, 0.67889908, 0.67889908, 0.68807339, 0.68807339,
       0.69724771, 0.69724771, 0.70642202, 0.70642202, 0.74311927,
       0.74311927, 0.76146789, 0.76146789, 0.78899083, 0.78899083,
       0.81651376, 0.81651376, 0.83486239, 0.83486239, 0.86238532,
       0.86238532, 0.87155963, 0.87155963, 0.88073394, 0.88073394,
       0.88990826, 0.88990826, 0.89908257, 0.89908257, 0.90825688,
       0.90825688, 0.91743119, 0.91743119, 0.94495413, 0.94495413,
       0.95412844, 0.95412844, 0.96330275, 0.96330275, 0.97247706,
       0.97247706, 0.98165138, 0.98165138, 0.99082569, 0.99082569,
       1.        , 1.        ])
```

```
import plotly.graph_objects as go
import numpy as np



# Generate a trace for ROC curve
trace0 = go.Scatter(
    x=fpr,
    y=tpr,
    mode='lines',
    name='ROC curve'
)



# Only label every nth point to avoid cluttering
n = 10
indices = np.arange(len(thresholds)) % n == 0  # Choose indices where index mod n is 0

trace1 = go.Scatter(
    x=fpr[indices],
    y=tpr[indices],
    mode='markers+text',
    name='Threshold points',
    text=[f"Thr={thr:.2f}" for thr in thresholds[indices]],
    textposition='top center'
)



# Diagonal line
trace2 = go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='lines',
    name='Random (Area = 0.5)',
    line=dict(dash='dash')
)

data = [trace0, trace1, trace2]



# Define layout with square aspect ratio
layout = go.Layout(
    title='Receiver Operating Characteristic',
    xaxis=dict(title='False Positive Rate'),
    yaxis=dict(title='True Positive Rate'),
    autosize=False,
    width=800,
    height=800,
    showlegend=False
)



# Define figure and add data
fig = go.Figure(data=data, layout=layout)

# Show figure
fig.show()
```




```
# Assume that fpr, tpr, thresholds have already been calculated
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Optimal threshold is:", optimal_threshold)
```

```
Optimal threshold is: 0.35441014470216464
```

- DecisionTreeClassifier

```
#Fitting a decision tree clasifier
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(x_train_std,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
#test the accuracy of the decision tree
predictions=dtree.predict(x_test_std)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.68      0.77      0.72        91
           1       0.78      0.70      0.74       109

    accuracy                           0.73       200
   macro avg       0.73      0.73      0.73       200
weighted avg       0.74      0.73      0.73       200
```

```
dtree_scores =  dtree.predict_proba(x_test_std)[:,1]
```

```
dtree_scores
```

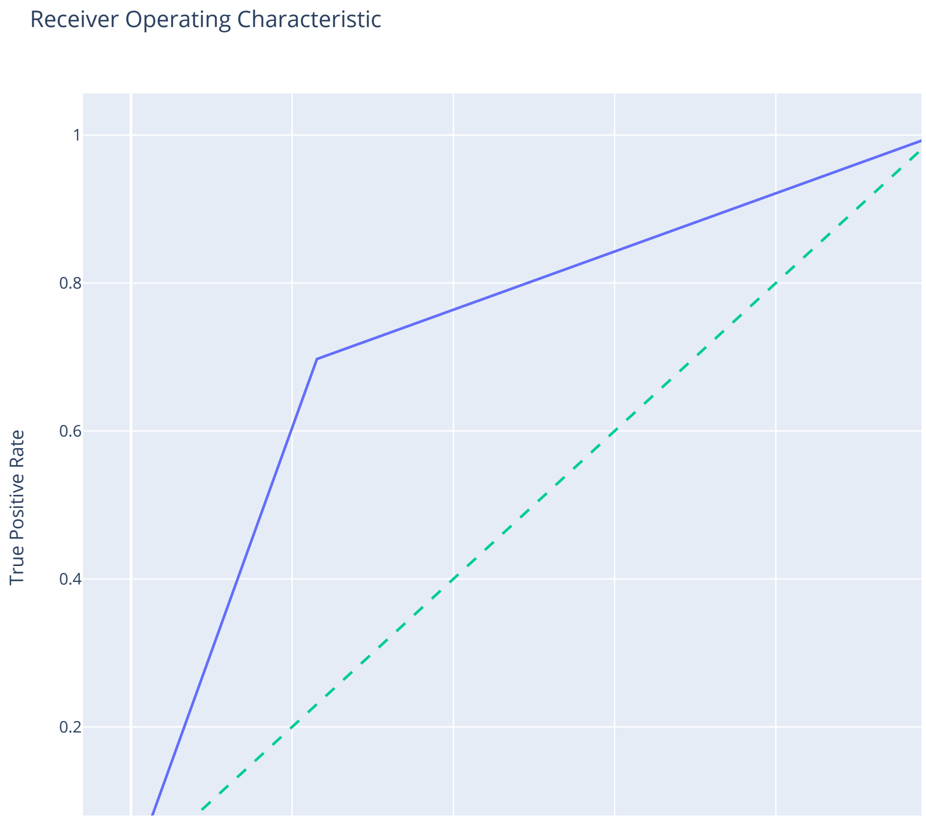
```
array([1., 0., 1., 1., 1., 1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0.,
       1., 0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 1., 1., 0.,
       1., 1., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1.,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

```
fpr, tpr, thresholds = roc_curve(y_test, dtree_scores)
```

```
thresholds
```

```
array([2., 1., 0.])
```

```
# Generate a trace for ROC curve
trace0 = go.Scatter(
    x=fpr,
    y=tpr,
    mode='lines',
    name='ROC curve'
)

# Only label every nth point to avoid cluttering
n = 10
indices = np.arange(len(thresholds)) % n == 0  # Choose indices where index mod n is 0

trace1 = go.Scatter(
    x=fpr[indices],
    y=tpr[indices],
    mode='markers+text',
    name='Threshold points',
    text=[f"Thr={thr:.2f}" for thr in thresholds[indices]],
    textposition='top center'
)


# Diagonal line
trace2 = go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='lines',
    name='Random (Area = 0.5)',
    line=dict(dash='dash')
)

data = [trace0, trace1, trace2]

# Define layout with square aspect ratio
layout = go.Layout(
    title='Receiver Operating Characteristic',
    xaxis=dict(title='False Positive Rate'),
    yaxis=dict(title='True Positive Rate'),
    autosize=False,
    width=800,
    height=800,
    showlegend=False
)

# Define figure and add data
fig = go.Figure(data=data, layout=layout)

# Show figure
fig.show()
```




```
# Assume that fpr, tpr, thresholds have already been calculated
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Optimal threshold is:", optimal_threshold)
```

```
Optimal threshold is: 1.0
```

- SVM

```
from sklearn import svm
```

```
svm_model = svm.SVC(kernel='linear')
```

```
svm_model.fit(x_train_std,y_train)
```

```
▾ SVC
SVC(kernel='linear')
```

```
X_train_prediction=svm_model.predict(x_train_std)
training_data_accuracy = round(accuracy_score(X_train_prediction,y_train)*100,2)
```

```
print("The accuracy score of the training data achieved using svm : "+str(training_data_accuracy)+"%")
```

```
The accuracy score of the training data achieved using svm : 76.75%
```

```
X_test_prediction=svm_model.predict(x_test_std)
test_data_accuracy = round(accuracy_score(X_test_prediction,y_test)*100,2)
```

```
print('The accuracy score of the testing data achieved using svm:',str(test_data_accuracy)+"%")
```

```
The accuracy score of the testing data achieved using svm: 75.5%
```

```
#SVM model
from sklearn.svm import SVC

svm_model = SVC(probability=True)
svm_model.fit(x_train_std, y_train)
svm_scores = svm_model.predict_proba(x_test_std)[:,1]
```

```
svm_scores
```

```
array([0.63022862, 0.31302433, 0.71111806, 0.95380196, 0.88032208,
       0.75640867, 0.13371968, 0.78388021, 0.81323747, 0.21635354,
       0.0387127 , 0.73428571, 0.55302389, 0.10836743, 0.77418798,
       0.0978341 , 0.15959372, 0.61413527, 0.5185461 , 0.71565798,
       0.64714478, 0.82626223, 0.64810233, 0.8852491 , 0.03234063,
       0.04735944, 0.02851633, 0.92512024, 0.46615695, 0.22002047,
       0.03338601, 0.25326887, 0.13834079, 0.91366409, 0.30115922,
       0.07326674, 0.71598726, 0.89781028, 0.75694517, 0.50791582,
       0.06596535, 0.75015982, 0.95442158, 0.03781239, 0.02694475,
       0.53597424, 0.34115848, 0.52227787, 0.92317006, 0.83431619,
       0.48178678, 0.74098508, 0.09635517, 0.67838982, 0.86310713,
       0.92650824, 0.08792657, 0.0150946 , 0.75613028, 0.1947659 ,
       0.05523608, 0.83417164, 0.23948247, 0.84737937, 0.93094129,
       0.02992925, 0.07554439, 0.74056739, 0.95853619, 0.28912211,
       0.07867374, 0.03387596, 0.18314479, 0.58356804, 0.91291335,
       0.0593738 , 0.94847047, 0.88977624, 0.73337433, 0.25711075,
       0.30522994, 0.03172689, 0.74029714, 0.75147034, 0.80477285,
       0.62682434, 0.83430874, 0.06309396, 0.40250391, 0.16618795,
       0.93708237, 0.77723334, 0.72251863, 0.03033039, 0.1238861 ,
       0.0264568 , 0.80262054, 0.83252417, 0.80125392, 0.37685578,
       0.19073952, 0.47303986, 0.83141897, 0.21835446, 0.121426  ,
       0.88115909, 0.02837037, 0.07018094, 0.35022575, 0.22947248,
       0.88753928, 0.21225354, 0.71855444, 0.88545991, 0.32269642,
       0.71125861, 0.80330952, 0.11593498, 0.45988935, 0.84562883,
       0.31565027, 0.03615813, 0.58601637, 0.39964987, 0.87240388,
       0.82651788, 0.79508389, 0.06161307, 0.71395863, 0.86746275,
       0.61748196, 0.88134889, 0.88809293, 0.3444376 , 0.42316138,
       0.02996126, 0.5       , 0.02301583, 0.09194509, 0.0700648 ,
       0.17631885, 0.03267585, 0.16846785, 0.74497176, 0.3353003 ,
       0.02422198, 0.87643236, 0.51077582, 0.70215985, 0.86409163,
       0.80830509, 0.78868143, 0.19019434, 0.59247155, 0.63521026,
       0.64459555, 0.39784179, 0.04295098, 0.10594887, 0.68042317,
       0.47114127, 0.88667315, 0.79871036, 0.03674052, 0.2725143 ,
       0.92404465, 0.09092285, 0.5       , 0.87708149, 0.85880873,
       0.23816804, 0.79621583, 0.42154938, 0.93514157, 0.61670798,
       0.06452686, 0.05148414, 0.09874758, 0.66231778, 0.87016058,
       0.91399467, 0.12466949, 0.85013274, 0.87767589, 0.06144232,
       0.65935472, 0.83786736, 0.81682933, 0.61208049, 0.08037055,
       0.90551893, 0.39939959, 0.4419883 , 0.26151607, 0.28088318,
       0.4398334 , 0.45386025, 0.83348482, 0.26222107, 0.34588571])
```

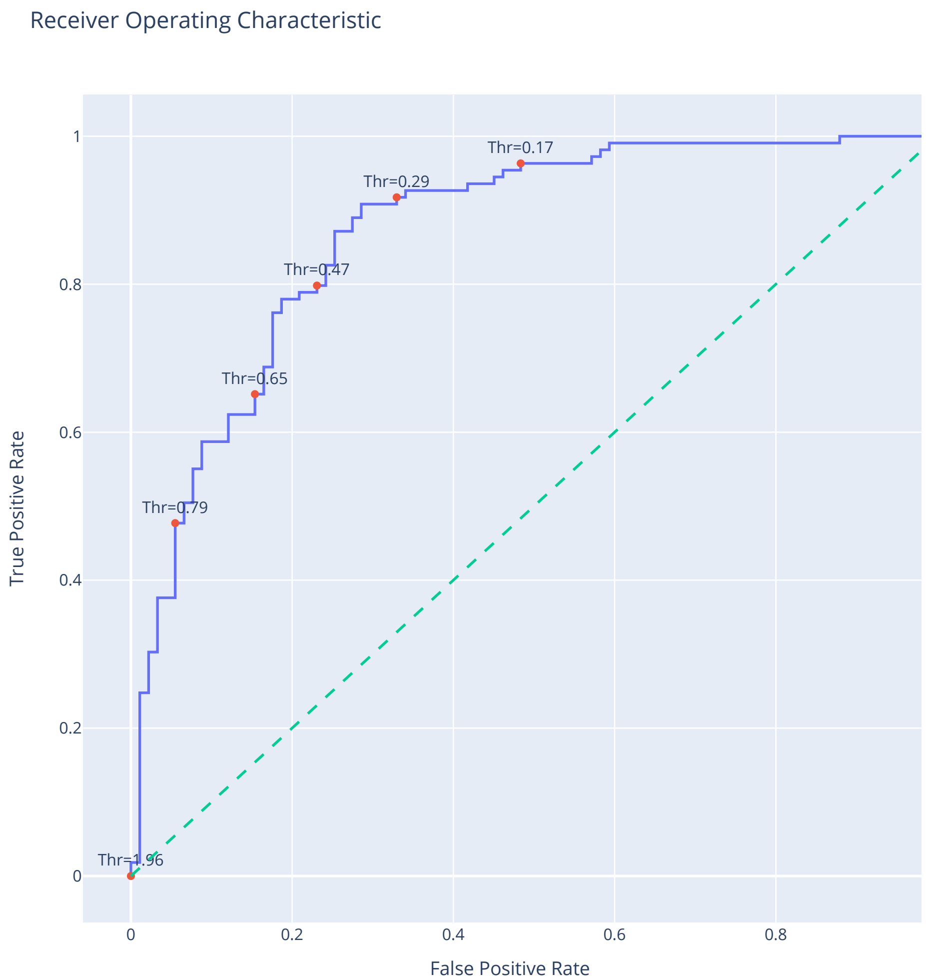
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, svm_scores)
```

```
thresholds
```

```
array([1.95853619, 0.95853619, 0.95442158, 0.95380196, 0.87643236,
       0.87240388, 0.85013274, 0.84737937, 0.83141897, 0.82626223,
       0.78868143, 0.78388021, 0.75694517, 0.75640867, 0.74098508,
       0.74056739, 0.72251863, 0.71565798, 0.70215985, 0.66231778,
       0.64714478, 0.64459555, 0.61748196, 0.61670798, 0.52227787,
       0.5185461 , 0.50791582, 0.5       , 0.48178678, 0.47114127,
       0.46615695, 0.45988935, 0.4398334 , 0.42316138, 0.39784179,
       0.35022575, 0.3444376 , 0.34115848, 0.32269642, 0.30115922,
       0.28912211, 0.28088318, 0.2725143 , 0.22947248, 0.22002047,
       0.21225354, 0.1947659 , 0.19073952, 0.19019434, 0.17631885,
       0.16846785, 0.11593498, 0.10836743, 0.10594887, 0.09874758,
       0.0978341 , 0.09635517, 0.03267585, 0.03234063, 0.0150946 ])
```

```
# Generate a trace for ROC curve
trace0 = go.Scatter(
    x=fpr,
    y=tpr,
    mode='lines',
    name='ROC curve'
)

# Only label every nth point to avoid cluttering
n = 10
indices = np.arange(len(thresholds)) % n == 0  # Choose indices where index mod n is 0

trace1 = go.Scatter(
    x=fpr[indices],
```

```
        y=tpr[indices],
        mode='markers+text',
        name='Threshold points',
        text=[f"Thr={thr:.2f}" for thr in thresholds[indices]],
        textposition='top center'
    )


    # Diagonal line
    trace2 = go.Scatter(
        x=[0, 1],
        y=[0, 1],
        mode='lines',
        name='Random (Area = 0.5)',
        line=dict(dash='dash')
    )

    data = [trace0, trace1, trace2]

    # Define layout with square aspect ratio
    layout = go.Layout(
        title='Receiver Operating Characteristic',
        xaxis=dict(title='False Positive Rate'),
        yaxis=dict(title='True Positive Rate'),
        autosize=False,
        width=800,
        height=800,
        showlegend=False
    )

    # Define figure and add data
    fig = go.Figure(data=data, layout=layout)

    # Show figure
    fig.show()
```




```
    # Assume that fpr, tpr, thresholds have already been calculated
    optimal_idx = np.argmax(tpr - fpr)
```

```
optimal_threshold = thresholds[optimal_idx]
print("Optimal threshold is:", optimal_threshold)
```

```
Optimal threshold is: 0.32269641782316993
```

- KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)

Y_pred_clf = knn_clf.predict(X_test)


score_knn = round(accuracy_score(Y_pred_clf,y_test)*100,2)

print("The accuracy score achieved using KNN Classifier is: "+str(score_knn)+" %")
```

```
The accuracy score achieved using KNN Classifier is: 76.5 %
```

- XGBOOST

```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(x_train_std, y_train)

Y_pred_xgb = xgb_model.predict(x_test_std)


score_xgb = round(accuracy_score(Y_pred_xgb,y_test)*100,2)

print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 82.0 %
```

```
#XGBOOST model
xgb_model = xgb.XGBClassifier()
xgb_model.fit(x_train_std, y_train)
xgb_scores = xgb_model.predict_proba(x_test_std)[:,1]
```

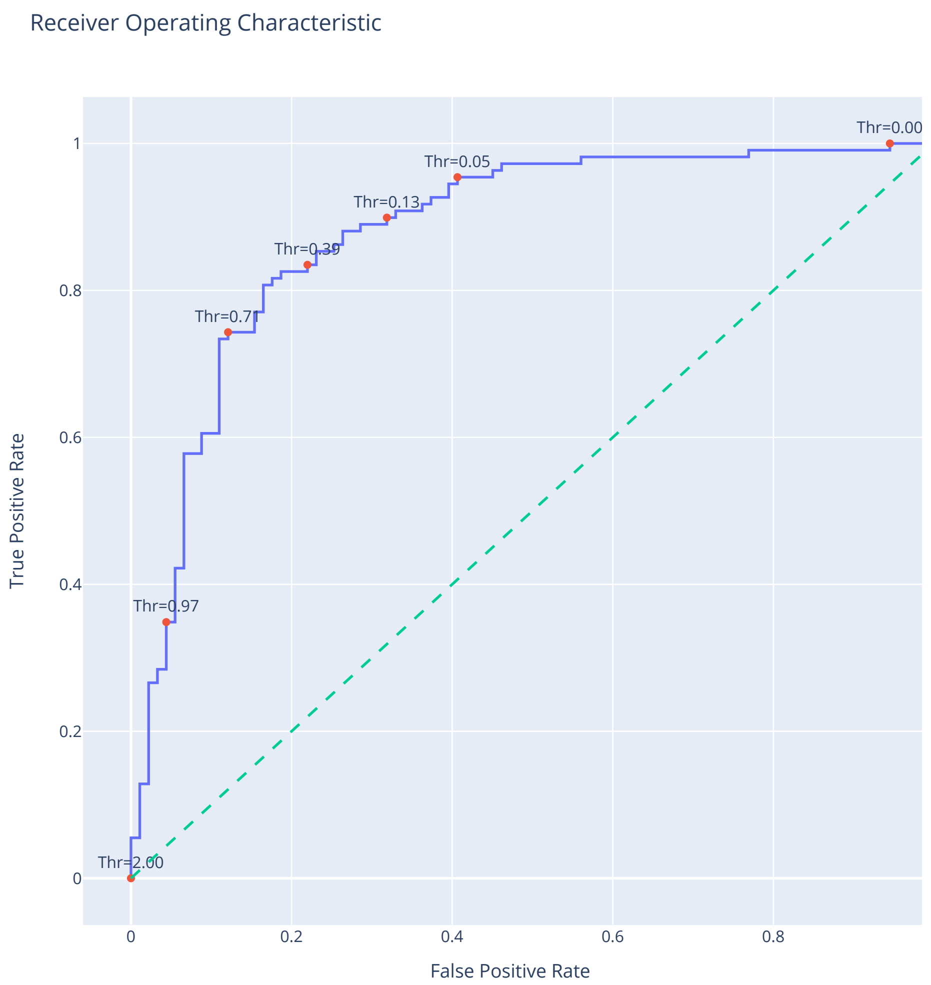
```
fpr, tpr, thresholds = roc_curve(y_test, xgb_scores)
```

```
thresholds
```

```
array([1.99975777e+00, 9.99757707e-01, 9.98406470e-01, 9.98137593e-01,
       9.96300459e-01, 9.95809913e-01, 9.86282527e-01, 9.85917628e-01,
       9.85292971e-01, 9.84187126e-01, 9.74204063e-01, 9.72667456e-01,
       9.57571268e-01, 9.49650526e-01, 8.76611173e-01, 8.71230423e-01,
       8.59341741e-01, 8.42847049e-01, 7.32558906e-01, 7.22114921e-01,
       7.14049757e-01, 6.51274681e-01, 6.34288490e-01, 6.34240210e-01,
       5.29427707e-01, 5.28912485e-01, 5.25854647e-01, 4.70783472e-01,
       4.42261308e-01, 4.13351893e-01, 3.93427432e-01, 3.54053378e-01,
       3.00053775e-01, 2.16475204e-01, 2.12766737e-01, 2.08975136e-01,
       2.01670796e-01, 1.86209515e-01, 1.81560799e-01, 1.26263320e-01,
       1.25755280e-01, 1.15521938e-01, 1.14025824e-01, 1.04601994e-01,
       1.03612922e-01, 9.43389088e-02, 9.37004536e-02, 8.61191228e-02,
       7.10364208e-02, 5.62824830e-02, 5.31572253e-02, 4.56776470e-02,
       4.43487838e-02, 4.25170772e-02, 4.19026539e-02, 1.43044461e-02,
       1.30674271e-02, 8.82233842e-04, 8.11111589e-04, 8.13190127e-05,
       7.62592535e-05, 1.35719492e-05], dtype=float32)
```

```
# Generate a trace for ROC curve
trace0 = go.Scatter(
    x=fpr,
    y=tpr,
    mode='lines',
    name='ROC curve'
)

# Only label every nth point to avoid cluttering
n = 10
indices = np.arange(len(thresholds)) % n == 0  # Choose indices where index mod n is 0
```

```
trace1 = go.Scatter(
    x=fpr[indices],
    y=tpr[indices],
    mode='markers+text',
    name='Threshold points',
    text=[f"Thr={thr:.2f}" for thr in thresholds[indices]],
    textposition='top center'
)


# Diagonal line
trace2 = go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='lines',
    name='Random (Area = 0.5)',
    line=dict(dash='dash')
)

data = [trace0, trace1, trace2]

# Define layout with square aspect ratio
layout = go.Layout(
    title='Receiver Operating Characteristic',
    xaxis=dict(title='False Positive Rate'),
    yaxis=dict(title='True Positive Rate'),
    autosize=False,
    width=800,
    height=800,
    showlegend=False
)

# Define figure and add data
fig = go.Figure(data=data, layout=layout)

# Show figure
fig.show()
```



```
# Assume that fpr, tpr, thresholds have already been calculated
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Optimal threshold is:", optimal_threshold)
```

```
Optimal threshold is: 0.5294277
```

Conclusion

From the results, we can see that XGBoost and SVM Classifier have done better than the others for this particular dataset.

```
t1=[]
for i in range(len(X_test_prediction)):
  if X_test_prediction[i]>=0.5:
    t1.append(1)
  else:
    t1.append(0)
```

```
new_pred = pd.Series(t1)
print(new_pred)
```

```
0      1
1      0
2      1
3      1
4      1
      ..
195    0
196    0
197    1
198    0
199    0
Length: 200, dtype: int64
```

```
y_test
```

```
37     1
726    0
846    1
295    0
924    1
      ..
839    1
810    1
930    1
616    0
809    1
Name: Outcome, Length: 200, dtype: int64
```

```
type(y_test)
```

```
pandas.core.series.Series
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,new_pred)
```

```
array([[72, 19],
       [30, 79]])
```

```
conf = confusion_matrix(y_test,new_pred)
```

```
tp,fp,fn,tn = confusion_matrix(y_test,new_pred).ravel()
specificity = tn / (tn+fp)
sensitivity= tp / (tp+fn)
print('TP,FP,FN,TN',tp,fp,fn,tn)
print('sensitivity =', sensitivity)
print('specificity =', specificity)
```

```
TP,FP,FN,TN 72 19 30 79
sensitivity = 0.7058823529411765
specificity = 0.8061224489795918
```

```
tp
```

```
72
```

```
fp
```

```
19
```

```
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC



# Generate ROC curve data for logistic regression model
lr_fpr, lr_tpr, lr_thresholds = roc_curve(y_test, lr_scores)
lr_auc = roc_auc_score(y_test, lr_scores)



# Generate ROC curve data for SVM model
svm_fpr, svm_tpr, svm_thresholds = roc_curve(y_test, svm_scores)
svm_auc = roc_auc_score(y_test, svm_scores)



# Generate ROC curve data for XGBOOST  model
xgb_fpr, xgb_tpr, xgb_thresholds = roc_curve(y_test, xgb_scores)
xgb_auc = roc_auc_score(y_test, xgb_scores)



# Generate a trace for the Logistic Regression ROC curve
trace0 = go.Scatter(
    x=lr_fpr,
    y=lr_tpr,
    mode='lines',
    name=f'Logistic Regression (Area = {lr_auc:.2f})'
)



# Generate a trace for the SVM ROC curve
trace1 = go.Scatter(
    x=svm_fpr,
    y=svm_tpr,
    mode='lines',
    name=f'SVM (Area = {svm_auc:.2f})'
)



# Generate a trace for the XGBOOST ROC curve
trace2 = go.Scatter(
    x=xgb_fpr,
    y=xgb_tpr,
    mode='lines',
    name=f'XGB (Area = {xgb_auc:.2f})'
)



# Diagonal line
trace3 = go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='lines',
    name='Random (Area = 0.5)',
    line=dict(dash='dash')
)



data = [trace0, trace1, trace2,trace3]

# Define layout with square aspect ratio
layout = go.Layout(
    title='Receiver Operating Characteristic',
```