```python
# import libraries
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc, precision_score, recall_score,
f1_score, roc_auc_score

#load the data
data = pd.read_csv('/content/framingham_heart_study.csv')
data.head()
```

```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 4240,\n  \"fields\":
[\n    {\n      \"column\": \"male\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"age\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8,\n        \"min\": 32,\n
\"max\": 70,\n        \"num_unique_values\": 39,\n        \"samples\":
[\n          34,\n          70\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"education\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0197911793650334,\n
\"min\": 1.0,\n        \"max\": 4.0,\n        \"num_unique_values\":
4,\n        \"samples\": [\n          2.0,\n          3.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"currentSmoker\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cigsPerDay\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.922461800608747,\n        \"min\":
0.0,\n        \"max\": 70.0,\n        \"num_unique_values\": 33,\n
\"samples\": [\n          19.0,\n          4.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BPMeds\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.16954428739625657,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
```

```
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"prevalentStroke\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"prevalentHyp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"diabetes\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"totChol\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 44.59128386860702,\n        \"min\":
107.0,\n        \"max\": 696.0,\n        \"num_unique_values\": 248,\n
\"samples\": [\n          311.0,\n          205.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"sysBP\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 22.0332996088492,\n
\"min\": 83.5,\n        \"max\": 295.0,\n
\"num_unique_values\": 234,\n        \"samples\": [\n          109.0,\
n          184.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"diaBP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 11.910394483305936,\n        \"min\": 48.0,\n        \"max\":
142.5,\n        \"num_unique_values\": 146,\n        \"samples\": [\n
106.0,\n          108.5\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BMI\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4.079840168944382,\n        \"min\":
15.54,\n        \"max\": 56.8,\n        \"num_unique_values\": 1364,\n
\"samples\": [\n          24.56,\n          19.87\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"heartRate\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
12.025347984469342,\n        \"min\": 44.0,\n        \"max\": 143.0,\n
\"num_unique_values\": 73,\n        \"samples\": [\n          85.0,\n
47.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"glucose\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 23.95433481134474,\n        \"min\": 40.0,\n
\"max\": 394.0,\n        \"num_unique_values\": 143,\n
\"samples\": [\n          394.0,\n          74.0\n        ],\n
```

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"TenYearCHD\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 0,\n   \"max\": 1,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   1,\n
0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"data"}

```
# Preview all columns
data.columns
```

```
Index(['male', 'age', 'education', 'currentSmoker', 'cigsPerDay',
'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol',
'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
      dtype='object')
```

```
# Preview counts of NULL values
data.isnull().sum()
```

```
male                 0
age                  0
education          105
currentSmoker        0
cigsPerDay          29
BPMeds              53
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             50
sysBP                0
diaBP                0
BMI                 19
heartRate            1
glucose            388
TenYearCHD           0
dtype: int64
```

```
# Preview counts of duplicated records
data.duplicated().sum()
```

```
np.int64(0)
```

```
# Preview dataset shape; rows: columns
data.shape
```

```
(4240, 16)
```

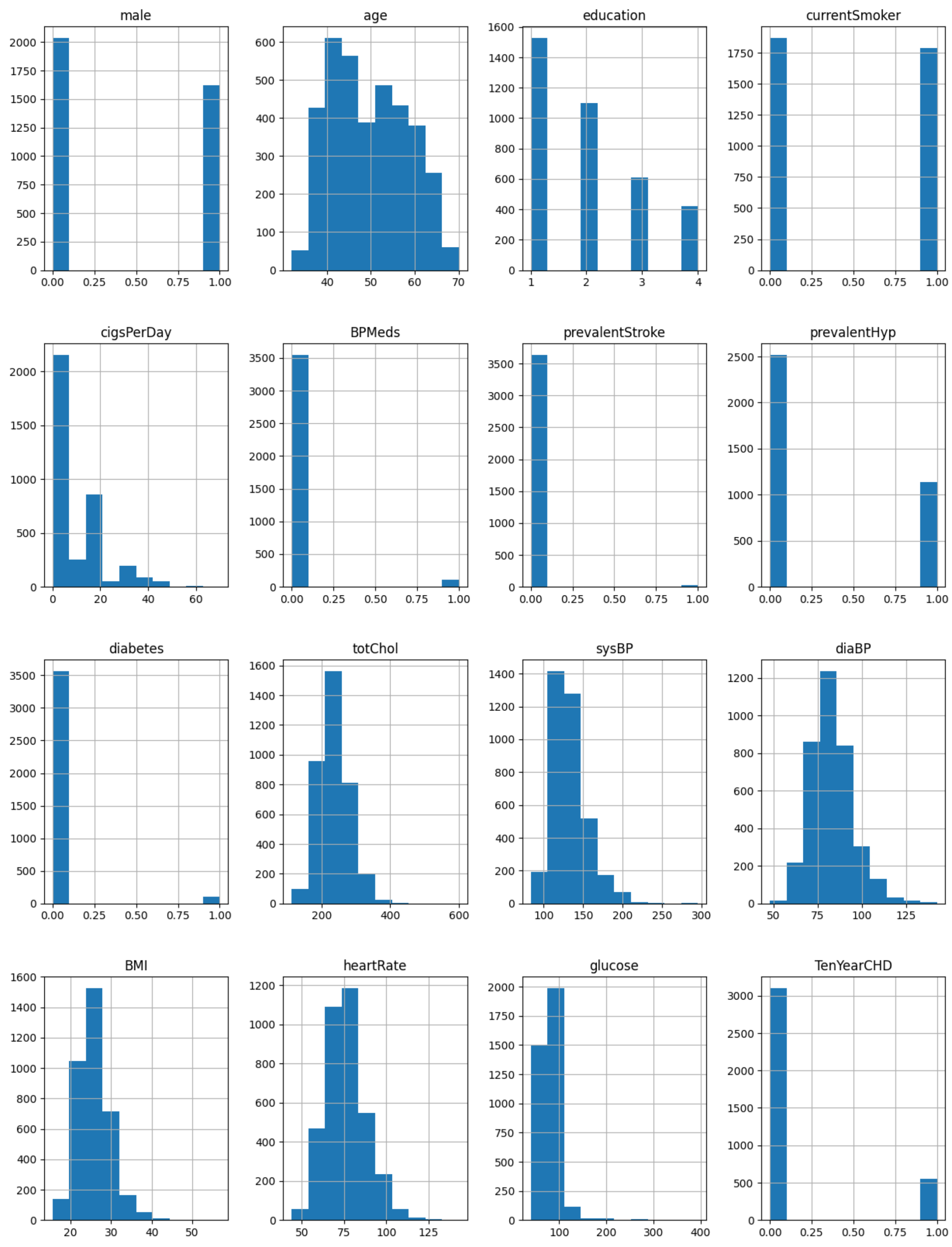
```
# Drop records with NULL values
data.dropna(inplace=True)

# Verify that there are no NULL values left
data.isnull().sum()
```

```
male               0
age                0
education          0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
dtype: int64
```

Data Distribution

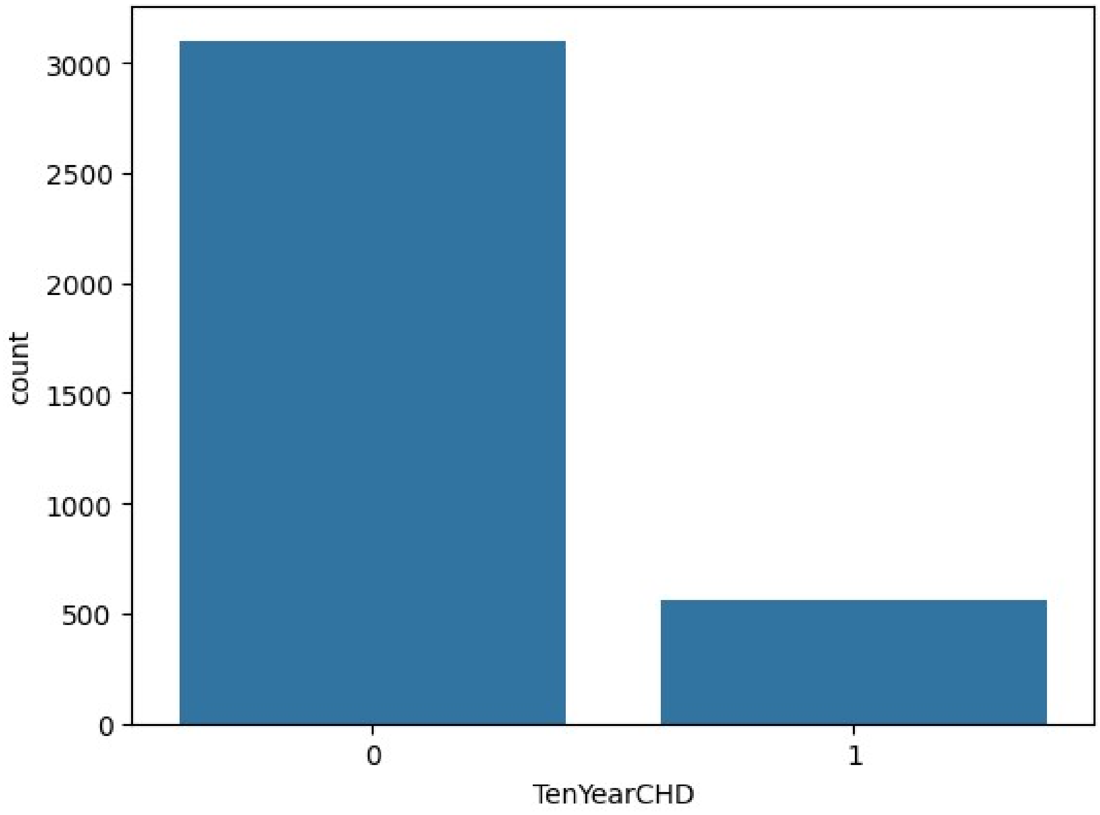
```
# plot histogram to see the distribution of the data
fig = plt.figure(figsize = (15,20))
ax = fig.gca()
data.hist(ax = ax)
plt.show()
```

```
/tmp/ipython-input-887923207.py:4: UserWarning: To output multiple
subplots, the figure containing the passed axes is being cleared.
  data.hist(ax = ax)
```



male
age
education
currentSmoker
cigsPerDay
BPMeds
prevalentStroke
prevalentHyp
diabetes
totChol
sysBP
diaBP
BMI
heartRate
glucose
TenYearCHD

```
sns.countplot(x='TenYearCHD',data=data)
plt.show()
cases = data.TenYearCHD.value_counts()
print(f"There are {cases[0]} patients without heart disease and
{cases[1]} patients with the disease")
```



```
There are 3101 patients without heart disease and 557 patients with
the disease
```

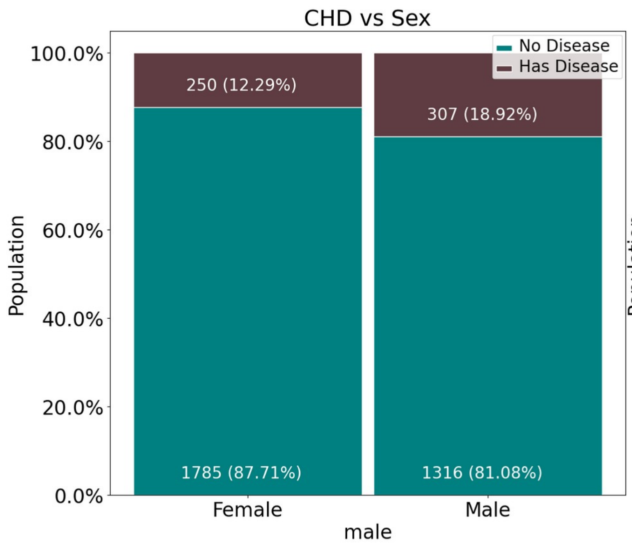
The data is not properly balanced as the number of people without the disease greately exceeds the number of people with the disease. The ratio is about 1:5.57

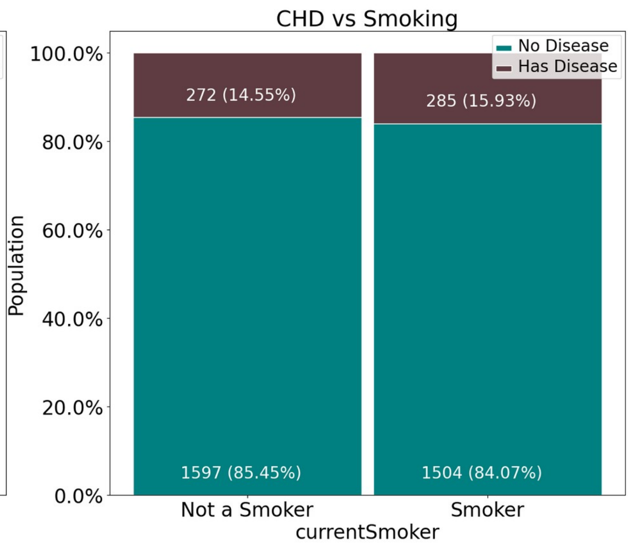
```
from operator import add

def stacked_barchart(data, title = None, ylabel = None, xlabel =
None):
    default_colors = ['#008080', '#5f3c41', '#219AD8']
    # From raw value to percentage
    totals = data.sum(axis=1)
    bars = ((data.T / totals) * 100).T
    r = list(range(data.index.size))

    # Plot
    barWidth = 0.95
    names = data.index.tolist()
    bottom = [0] * bars.shape[0]

    # Create bars
    color_index = 0
    plots = []
    for bar in bars.columns:
        plots.append(plt.bar(r, bars[bar], bottom=bottom,
color=default_colors[color_index], edgecolor='white', width=barWidth))
        bottom = list(map(add, bottom, bars[bar]))
        color_index = 0 if color_index >= len(default_colors) else
color_index + 1

    # Custom x axis
    plt.title(title)
    plt.xticks(r, names)
    plt.xlabel(data.index.name if xlabel is None else xlabel)
    plt.ylabel(data.columns.name if ylabel is None else ylabel)
    ax = plt.gca()

    y_labels = ax.get_yticks()
    ax.set_yticklabels([str(y) + '%' for y in y_labels])

    flat_list = [item for sublist in data.T.values for item in
sublist]
    for i, d in zip(ax.patches, flat_list):
        data_label = str(d) + " (" + str(round(i.get_height(), 2)) +
"%)"
        ax.text(i.get_x() + 0.45, i.get_y() + 5, data_label,
horizontalalignment='center', verticalalignment='center', fontdict =
dict(color = 'white', size = 20))

    for item in ([ax.title]):
        item.set_fontsize(27)

    for item in ([ax.xaxis.label, ax.yaxis.label] +
ax.get_xticklabels() + ax.get_yticklabels()):
        item.set_fontsize(24)

    legend = ax.legend(plots, bars.columns.tolist(), fancybox=True)
    plt.setp(legend.get_texts(), fontsize='20')

fig = plt.gcf()
fig.set_size_inches(25, 35)
grid_rows = 3
grid_cols = 2

#draw sex vs disease outcome
plt.subplot(grid_rows, grid_cols, 1)
```

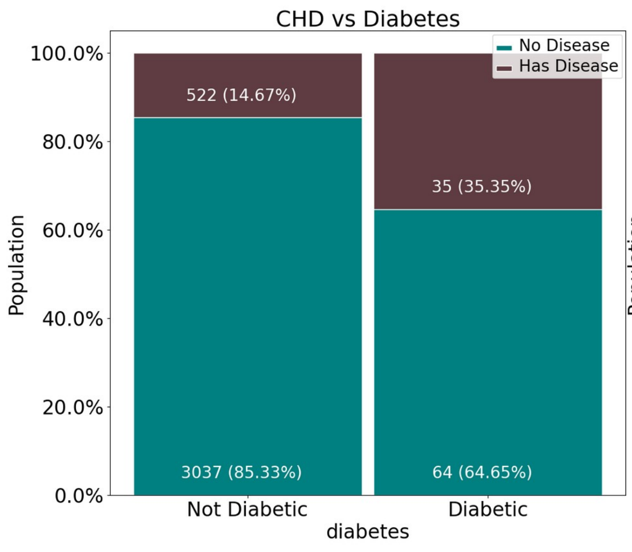
```
temp =
data[['male','TenYearCHD']].groupby(['male','TenYearCHD']).size().unst
ack('TenYearCHD')
temp.rename(index={0:'Female', 1:'Male'}, columns={0:'No Disease',
1:'Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'CHD vs Sex', ylabel = 'Population')

#draw smoking satus vs disease outcome
plt.subplot(grid_rows, grid_cols, 2)
temp =
data[['currentSmoker','TenYearCHD']].groupby(['currentSmoker','TenYear
CHD']).size().unstack('TenYearCHD')
temp.rename(index={0:'Not a Smoker', 1:'Smoker'}, columns={0:'No
Disease', 1:'Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'CHD vs Smoking', ylabel =
'Population')

#draw diabetes vs disease outcome
plt.subplot(grid_rows, grid_cols, 3)
temp =
data[['diabetes','TenYearCHD']].groupby(['diabetes','TenYearCHD']).siz
e().unstack('TenYearCHD')
temp.rename(index={0:'Not Diabetic', 1:'Diabetic'}, columns={0:'No
Disease', 1:'Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'CHD vs Diabetes', ylabel =
'Population')

#draw BP meds vs disease outcome
plt.subplot(grid_rows, grid_cols, 4)
temp =
data[['BPMeds','TenYearCHD']].groupby(['BPMeds','TenYearCHD']).size().
unstack('TenYearCHD')
temp.rename(index={0:'Not on medication', 1:'On Medication'},
columns={0:'No Disease', 1:'Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'CHD vs BP meds', ylabel =
'Population')

#draw Hypertension vs disease outcome
plt.subplot(grid_rows, grid_cols, 5)
temp =
data[['prevalentHyp','TenYearCHD']].groupby(['prevalentHyp','TenYearCH
D']).size().unstack('TenYearCHD')
temp.rename(index={0:'Not Hypertensive', 1:'Hypertensive'},
columns={0:'No Disease', 1:'Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'CHD vs Hypertension', ylabel =
'Population')
```

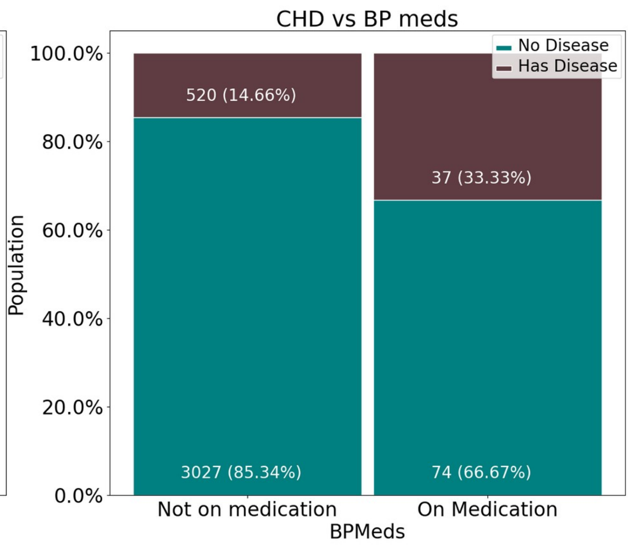
```
/tmp/ipython-input-2869839754.py:31: UserWarning: set_ticklabels()
should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
```

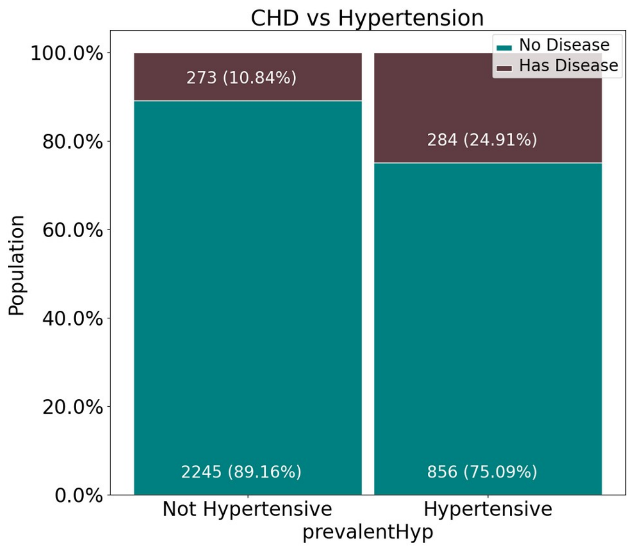
```
  ax.set_yticklabels([str(y) + '%' for y in y_labels])
/tmp/ipython-input-2869839754.py:31: UserWarning: set_ticklabels()
should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
  ax.set_yticklabels([str(y) + '%' for y in y_labels])
/tmp/ipython-input-2869839754.py:31: UserWarning: set_ticklabels()
should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
  ax.set_yticklabels([str(y) + '%' for y in y_labels])
/tmp/ipython-input-2869839754.py:31: UserWarning: set_ticklabels()
should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
  ax.set_yticklabels([str(y) + '%' for y in y_labels])
/tmp/ipython-input-2869839754.py:31: UserWarning: set_ticklabels()
should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
  ax.set_yticklabels([str(y) + '%' for y in y_labels])
```


CHD vs Sex
No Disease
Has Disease
100.0%
80.0%
60.0%
40.0%
20.0%
0.0%
Population
250 (12.29%)
1785 (87.71%)
307 (18.92%)
1316 (81.08%)
Female
Male
male


CHD vs Smoking
No Disease
Has Disease
100.0%
80.0%
60.0%
40.0%
20.0%
0.0%
Population
272 (14.55%)
1597 (85.45%)
285 (15.93%)
1504 (84.07%)
Not a Smoker
Smoker
currentSmoker


CHD vs Diabetes
No Disease
Has Disease
100.0%
80.0%
60.0%
40.0%
20.0%
0.0%
Population
522 (14.67%)
3037 (85.33%)
35 (35.35%)
64 (64.65%)
Not Diabetic
Diabetic
diabetes


CHD vs BP meds
No Disease
Has Disease
100.0%
80.0%
60.0%
40.0%
20.0%
0.0%
Population
520 (14.66%)
3027 (85.34%)
37 (33.33%)
74 (66.67%)
Not on medication
On Medication
BPMeds

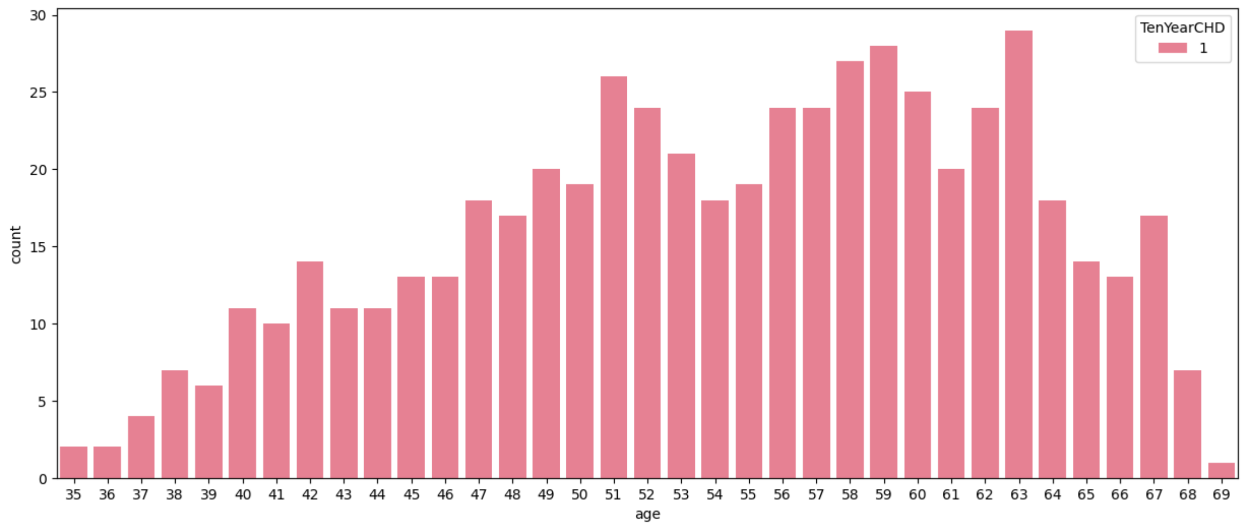

CHD vs Hypertension
No Disease
Has Disease
100.0%
80.0%
60.0%
40.0%
20.0%
0.0%
Population
273 (10.84%)
2245 (89.16%)
284 (24.91%)
856 (75.09%)
Not Hypertensive
Hypertensive
prevalentHyp

Due to the imbalanced nature of the dataset it is difficult to make conclusions but based on what is observed but these are the conclusions that can be drawn:

Slightly more males are suffering from CHD than females The percentage of people who have CHD is almost equal between smokers and non smokers The percentage of people who have CHD is higher among the diabetic, and those with prevalent hypertesion as compared to those who dont have similar morbidities A larger percentage of the people who have CHD are on blood pressure medication

Number of people who have disease vs age

```
positive_cases = data[data['TenYearCHD'] == 1]
plt.figure(figsize=(15,6))
sns.countplot(x='age',data = positive_cases, hue = 'TenYearCHD',
palette='husl')
plt.show()
```

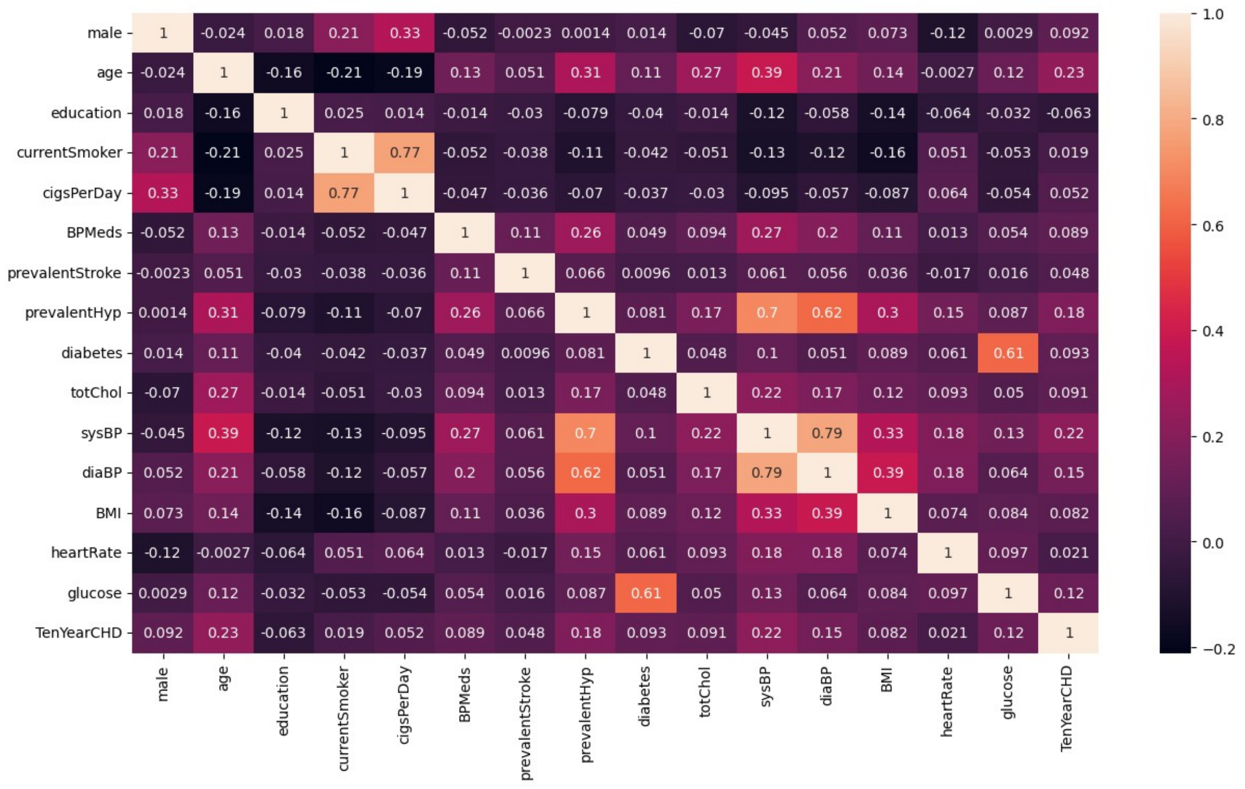


The people with the highest risk of developing CHD are betwwen the ages of 51 and 63

The number of sick people generally increases with age

Correlation Heat map

```
plt.figure(figsize=(15,8))
sns.heatmap(data.corr(), annot = True)
plt.show()
```



Feature Selection

```
from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

!pip install boruta

Requirement already satisfied: boruta in
/usr/local/lib/python3.12/dist-packages (0.4.3)
Requirement already satisfied: numpy>=1.10.4 in
/usr/local/lib/python3.12/dist-packages (from boruta) (2.0.2)
Requirement already satisfied: scikit-learn>=0.17.1 in
/usr/local/lib/python3.12/dist-packages (from boruta) (1.6.1)
Requirement already satisfied: scipy>=0.17.0 in
/usr/local/lib/python3.12/dist-packages (from boruta) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.17.1-
>boruta) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.17.1-
>boruta) (3.6.0)

#define the features
X = data.iloc[:,:-1].values
y = data.iloc[:,-1].values
```

```
forest = RandomForestClassifier(n_estimators=1000, n_jobs=-1,
class_weight='balanced')

# define Boruta feature selection method
feat_selector = BorutaPy(forest, n_estimators='auto', verbose=2)

# find all relevant features
feat_selector.fit(X, y)

Iteration: 	1 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	2 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	3 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	4 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	5 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	6 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	7 / 100
Confirmed: 	0
Tentative: 	15
Rejected: 	0
Iteration: 	8 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	9 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	10 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	11 / 100
```

```
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	12 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	13 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	14 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	15 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	16 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	17 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	18 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	19 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	20 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	21 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	22 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	23 / 100
Confirmed: 	2
```

```
Tentative: 	4
Rejected: 	9
Iteration: 	24 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	25 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	26 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	27 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	28 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	29 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	30 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	31 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	32 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	33 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	34 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	35 / 100
Confirmed: 	2
Tentative: 	4
```

```
Rejected:  9
Iteration:      36 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      37 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      38 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      39 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      40 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      41 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      42 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      43 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      44 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      45 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      46 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
Iteration:      47 / 100
Confirmed:      2
Tentative:      4
Rejected:  9
```

```
Iteration: 	48 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	49 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	50 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	51 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	52 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	53 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	54 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	55 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	56 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	57 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	58 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	59 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	60 / 100
```

```
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	61 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	62 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	63 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	64 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	65 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	66 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	67 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	68 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	69 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	70 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	71 / 100
Confirmed: 	2
Tentative: 	4
Rejected: 	9
Iteration: 	72 / 100
Confirmed: 	2
```

```
Tentative:      3
Rejected:  10
Iteration:      73 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      74 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      75 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      76 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      77 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      78 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      79 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      80 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      81 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      82 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      83 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      84 / 100
Confirmed:      2
Tentative:      3
```

```
Rejected: 	10
Iteration: 	85 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	86 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	87 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	88 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	89 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	90 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	91 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	92 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	93 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	94 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	95 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
Iteration: 	96 / 100
Confirmed: 	2
Tentative: 	3
Rejected: 	10
```

```
Iteration:      97 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      98 / 100
Confirmed:      2
Tentative:      3
Rejected:  10
Iteration:      99 / 100
Confirmed:      2
Tentative:      3
Rejected:  10


BorutaPy finished running.

Iteration:      100 / 100
Confirmed:      2
Tentative:      2
Rejected:  11

BorutaPy(estimator=RandomForestClassifier(class_weight='balanced',
                                          n_estimators=31, n_jobs=-1,

random_state=RandomState(MT19937) at 0x7FD07CA80A40),
         n_estimators='auto',
         random_state=RandomState(MT19937) at 0x7FD07CA80A40,
verbose=2)
```

```
# show the most important features
most_important = data.columns[:-1][feat_selector.support_].tolist()
most_important
```

```
['age', 'sysBP']
```

```
# select the top 6 features
top_features = data.columns[:-1][feat_selector.ranking_ <=6].tolist()
top_features
```

```
['age',
 'cigsPerDay',
 'totChol',
 'sysBP',
 'diaBP',
 'BMI',
 'heartRate',
 'glucose']
```

```
import statsmodels.api as sm
X_top = data[top_features]
y = data['TenYearCHD']
```

```
res = sm.Logit(y,X_top).fit()
res.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.413285
         Iterations 6

<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results

==============================================================================
========
Dep. Variable:             TenYearCHD   No. Observations:
3658
Model:                          Logit   Df Residuals:
3650
Method:                           MLE   Df Model:
7
Date:                Tue, 28 Oct 2025   Pseudo R-squ.:
0.03127
Time:                        16:41:15   Log-Likelihood:
-1511.8
converged:                       True   LL-Null:
-1560.6
Covariance Type:            nonrobust   LLR p-value:
3.388e-18
==============================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
--------
age            0.0259      0.006      4.516      0.000       0.015
0.037
cigsPerDay     0.0160      0.004      4.190      0.000       0.009
0.023
totChol       -0.0032      0.001     -2.907      0.004      -0.005
-0.001
sysBP          0.0262      0.003      7.761      0.000       0.020
0.033
diaBP         -0.0290      0.006     -4.838      0.000      -0.041
-0.017
BMI           -0.0517      0.012     -4.244      0.000      -0.076
-0.028
heartRate     -0.0333      0.004     -8.624      0.000      -0.041
-0.026
glucose        0.0041      0.002      2.470      0.014       0.001
0.007
==============================================================================
========
"""

params = res.params
conf = res.conf_int()
conf['Odds Ratio'] = params
conf.columns = ['5%', '95%', 'Odds Ratio']
print(np.exp(conf))
```

```
                   5%       95%  Odds Ratio
age          1.014747  1.037785    1.026201
cigsPerDay   1.008552  1.023761    1.016128
totChol      0.994635  0.998953    0.996792
sysBP        1.019746  1.033309    1.026505
diaBP        0.960121  0.982919    0.971453
BMI          0.927144  0.972538    0.949569
heartRate    0.959992  0.974620    0.967278
glucose      1.000848  1.007388    1.004113
```

```
sns.pairplot(data, hue = 'TenYearCHD', markers=["o", "s"], vars =
top_features, palette = sns.color_palette("bright", 10))
```

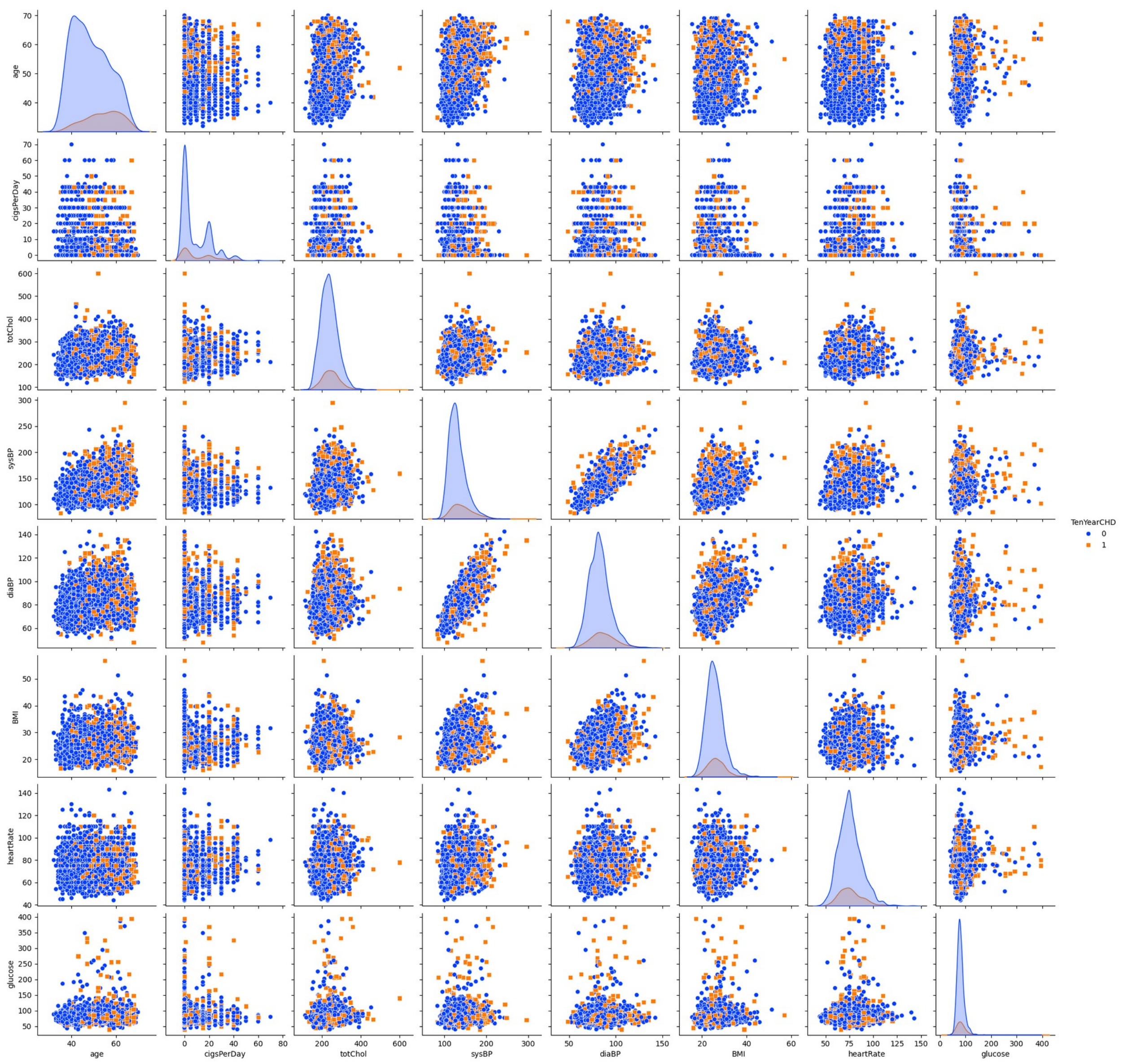
```
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
```

```
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1513:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
```

```
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
```

```
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
```

```
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)
/usr/local/lib/python3.12/dist-packages/seaborn/axisgrid.py:1615:
UserWarning: The palette list has more values (10) than needed (2),
which may not be intended.
  func(x=x, y=y, **kwargs)

<seaborn.axisgrid.PairGrid at 0x7fd03d1d2c60>
```

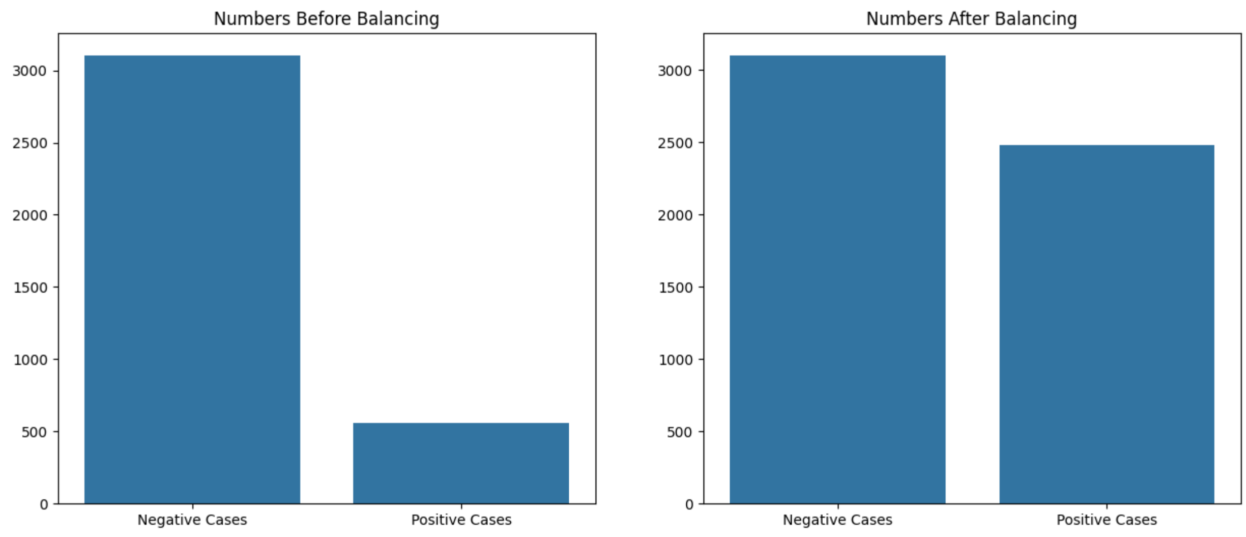


```python
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from collections import Counter

X = data[top_features]
y = data.iloc[:,-1]

# the numbers before SMOTE
num_before = dict(Counter(y))

#perform SMOTE

# define pipeline
over = SMOTE(sampling_strategy=0.8)
```

```python
under = RandomUnderSampler(sampling_strategy=0.8)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)

# transform the dataset
X_smote, y_smote = pipeline.fit_resample(X, y)


#the numbers after SMOTE
num_after =dict(Counter(y_smote))

print(num_before, num_after)
```

```
{0: 3101, 1: 557} {0: 3100, 1: 2480}
```

```python
labels = ["Negative Cases","Positive Cases"]
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
sns.barplot(x=labels, y=list(num_before.values()))
plt.title("Numbers Before Balancing")
plt.subplot(1,2,2)
sns.barplot(x=labels, y=list(num_after.values()))
plt.title("Numbers After Balancing")
plt.show()
```



After applying SMOTE, the new dataset is much more balanced: the new ratio between negative and positive cases is 1:1.2 up from 1:5.57

Splitting data to Training and Testing set

```python
# new dataset
new_data = pd.concat([pd.DataFrame(X_smote), pd.DataFrame(y_smote)],
axis=1)
new_data.columns = ['age', 'cigsPerDay', 'totChol', 'sysBP', 'diaBP',
'BMI', 'heartRate', 'glucose','TenYearCHD']
new_data.head()
```

```
{"summary":"{\n  \"name\": \"new_data\",\n  \"rows\": 5580,\n
\"fields\": [\n    {\n      \"column\": \"age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8,\n        \"min\": 32,\n        \"max\": 70,\n
\"num_unique_values\": 39,\n        \"samples\": [\n          67,\n
66,\n          51\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cigsPerDay\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.786497053317436,\n        \"min\":
0.0,\n        \"max\": 70.0,\n        \"num_unique_values\": 1022,\n
\"samples\": [\n          13.565418159273838,\n
0.25182790036791936,\n          0.4359354099353465\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"totChol\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
44.999664220334225,\n        \"min\": 113.0,\n        \"max\": 600.0,\
n        \"num_unique_values\": 2078,\n        \"samples\": [\n
182.31866094439306,\n          211.6772155023817,\n
218.87492127878448\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sysBP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 23.54639666156191,\n        \"min\": 83.5,\n        \"max\":
295.0,\n        \"num_unique_values\": 2111,\n        \"samples\": [\n
113.71045859878971,\n          125.82533449921712,\n
94.31280282047642\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"diaBP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 12.439863788070305,\n        \"min\": 48.0,\n        \"max\":
142.5,\n        \"num_unique_values\": 1994,\n        \"samples\": [\n
77.89064890790087,\n          115.22328854038206,\n
95.47487834044118\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BMI\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.937757123899171,\n        \"min\": 15.54,\n        \"max\":
56.8,\n        \"num_unique_values\": 3217,\n        \"samples\": [\n
30.3,\n          30.591532451553633,\n          31.35\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"heartRate\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.619510898872495,\n        \"min\": 44.0,\n        \"max\": 143.0,\n
\"num_unique_values\": 1842,\n        \"samples\": [\n
68.3990319413708,\n          86.59958491578152,\n
78.57773073725667\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"glucose\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 29.258899439578457,\n        \"min\": 40.0,\n
\"max\": 394.0,\n        \"num_unique_values\": 1971,\n
\"samples\": [\n          89.50118920631317,\n
73.64168764506331,\n          75.18603613470881\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TenYearCHD\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"new_data"}
```

```
X_new = new_data[top_features]
y_new= new_data.iloc[:,-1]
X_new.head()
```

```
{"summary":"{\n  \"name\": \"X_new\",\n  \"rows\": 5580,\n
\"fields\": [\n    {\n      \"column\": \"age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8,\n        \"min\": 32,\n        \"max\": 70,\n
\"num_unique_values\": 39,\n        \"samples\": [\n          67,\n
66,\n          51\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cigsPerDay\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.786497053317436,\n        \"min\":
0.0,\n        \"max\": 70.0,\n        \"num_unique_values\": 1022,\n
\"samples\": [\n          13.565418159273838,\n
0.25182790036791936,\n          0.4359354099353465\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"totChol\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
44.999664220334225,\n        \"min\": 113.0,\n        \"max\": 600.0,\
n        \"num_unique_values\": 2078,\n        \"samples\": [\n
182.31866094439306,\n          211.6772155023817,\n
218.87492127878448\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sysBP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 23.54639666156191,\n        \"min\": 83.5,\n        \"max\":
295.0,\n        \"num_unique_values\": 2111,\n        \"samples\": [\n
113.71045859878971,\n          125.82533449921712,\n
94.31280282047642\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"diaBP\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 12.439863788070305,\n        \"min\": 48.0,\n        \"max\":
142.5,\n        \"num_unique_values\": 1994,\n        \"samples\": [\n
77.89064890790087,\n          115.22328854038206,\n
95.47487834044118\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BMI\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.937757123899171,\n        \"min\": 15.54,\n        \"max\":
56.8,\n        \"num_unique_values\": 3217,\n        \"samples\": [\n
30.3,\n          30.591532451553633,\n          31.35\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"heartRate\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.619510898872495,\n        \"min\": 44.0,\n        \"max\": 143.0,\n
\"num_unique_values\": 1842,\n        \"samples\": [\n
68.3990319413708,\n          86.59958491578152,\n
78.57773073725667\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"glucose\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 29.258899439578457,\n        \"min\": 40.0,\n
\"max\": 394.0,\n        \"num_unique_values\": 1971,\n
\"samples\": [\n          89.50118920631317,\n
73.64168764506331,\n          75.18603613470881\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"X_new"}
```

```
# split the dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X_new,y_new,test_size=.2,random_state=42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train_scaled)

X_test_scaled = scaler.transform(X_test)
X_test = pd.DataFrame(X_test_scaled)
```

Logistic regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import
recall_score,precision_score,classification_report,roc_auc_score,roc_c
urve

# search for optimun parameters using gridsearch
params = {'penalty':['l1','l2'],
         'C':[0.01,0.1,1,10,100],
         'class_weight':['balanced',None]}
logistic_clf =
GridSearchCV(LogisticRegression(),param_grid=params,cv=10)
```

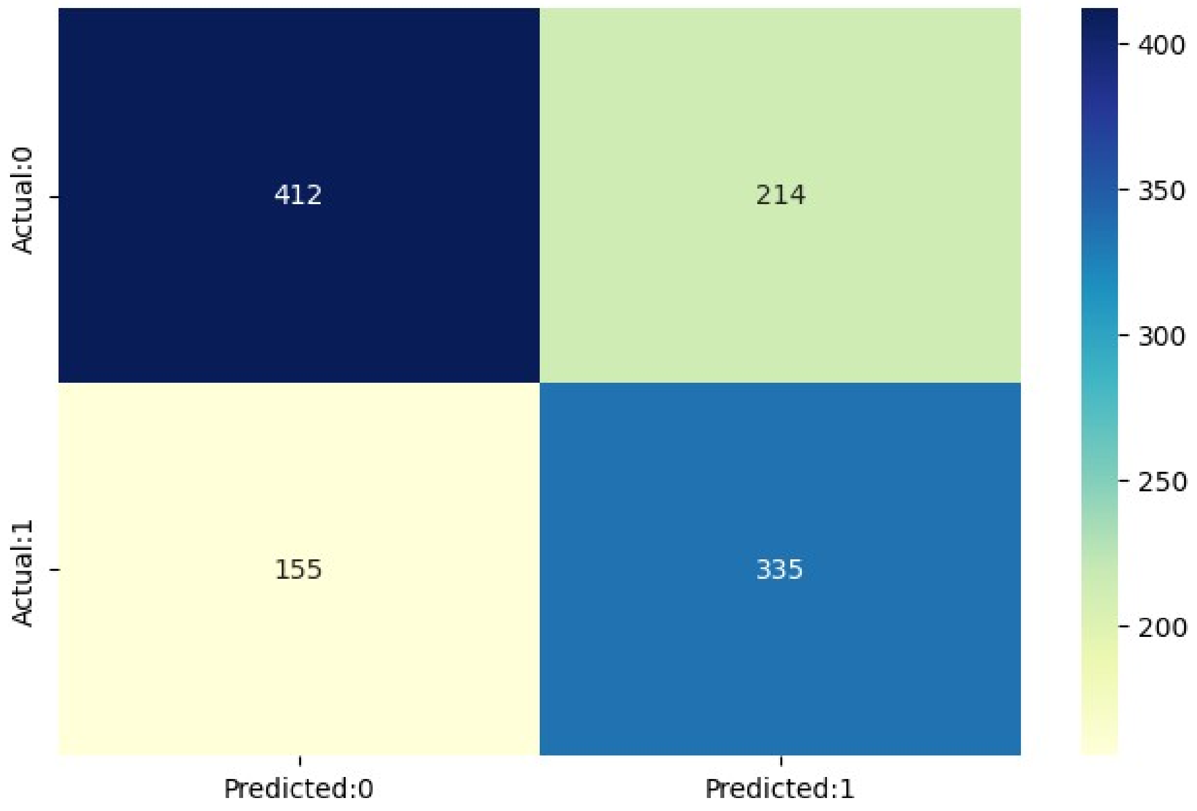
```
#train the classifier
logistic_clf.fit(X_train,y_train)

logistic_clf.best_params_

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/
_validation.py:528: FitFailedWarning:
100 fits failed out of a total of 200.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
-----------------------------------------------------------------------
----------
100 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_vali
dation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line
1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logisti
c.py", line 1193, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logisti
c.py", line 63, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or None penalties, got l1
penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_searc
h.py:1108: UserWarning: One or more of the test scores are non-finite:
[       nan 0.66218387        nan 0.65367021        nan 0.661286
        nan 0.65569015        nan 0.66083858        nan 0.65658701
        nan 0.66106279        nan 0.65658701        nan 0.66106279
        nan 0.65658701]
  warnings.warn(

{'C': 0.01, 'class_weight': 'balanced', 'penalty': 'l2'}
```

```
#make predictions
logistic_predict = logistic_clf.predict(X_test)
log_accuracy = accuracy_score(y_test,logistic_predict)
print(f"Using logistic regression we get an accuracy of
{round(log_accuracy*100,2)}%")
```

```
Using logistic regression we get an accuracy of 66.94%
```

```
cm=confusion_matrix(y_test,logistic_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1']
,index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

```
<Axes: >
```



```
print(classification_report(y_test,logistic_predict))
```

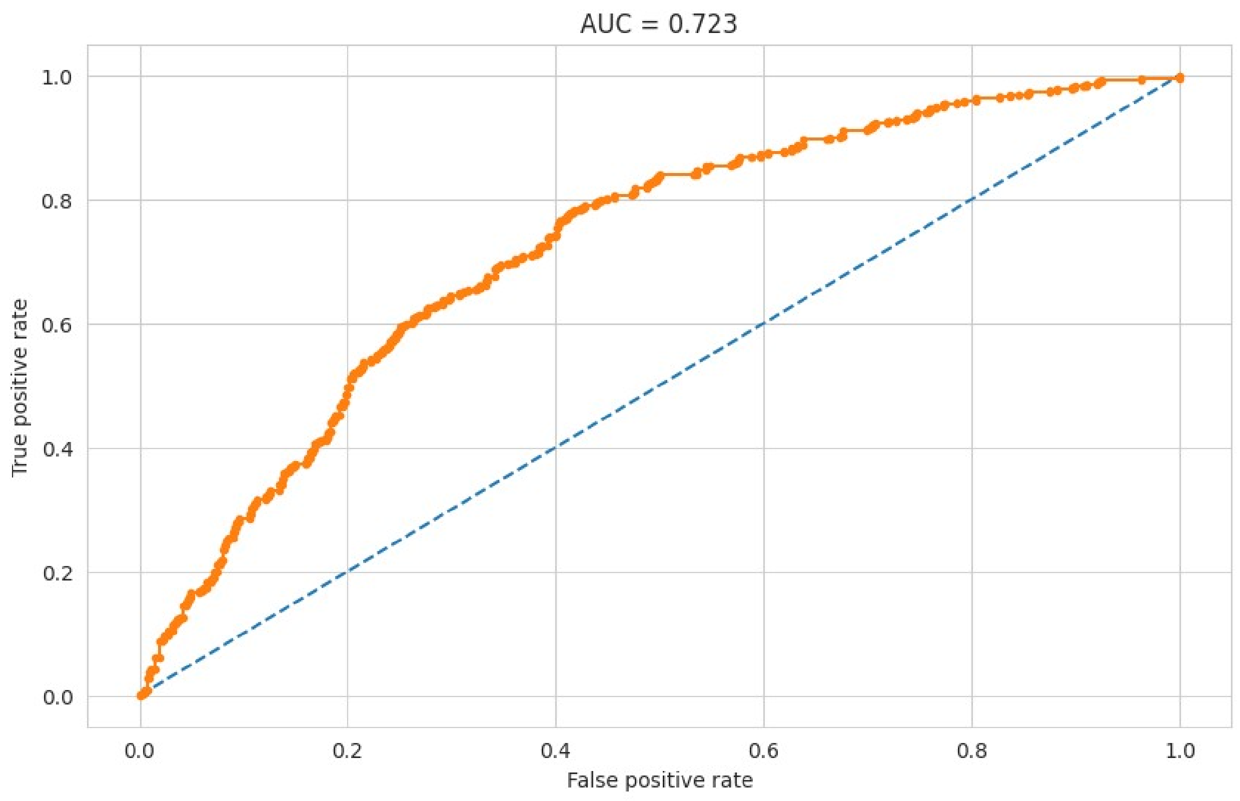
```
              precision    recall  f1-score   support

           0       0.73      0.66      0.69       626
           1       0.61      0.68      0.64       490

    accuracy                           0.67      1116
   macro avg       0.67      0.67      0.67      1116
```

```
weighted avg       0.68      0.67      0.67      1116
```

```
logistic_f1 = f1_score(y_test, logistic_predict)
print(f'The f1 score for logistic regression is
{round(logistic_f1*100,2)}%')
```

```
The f1 score for logistic regression is 64.49%
```

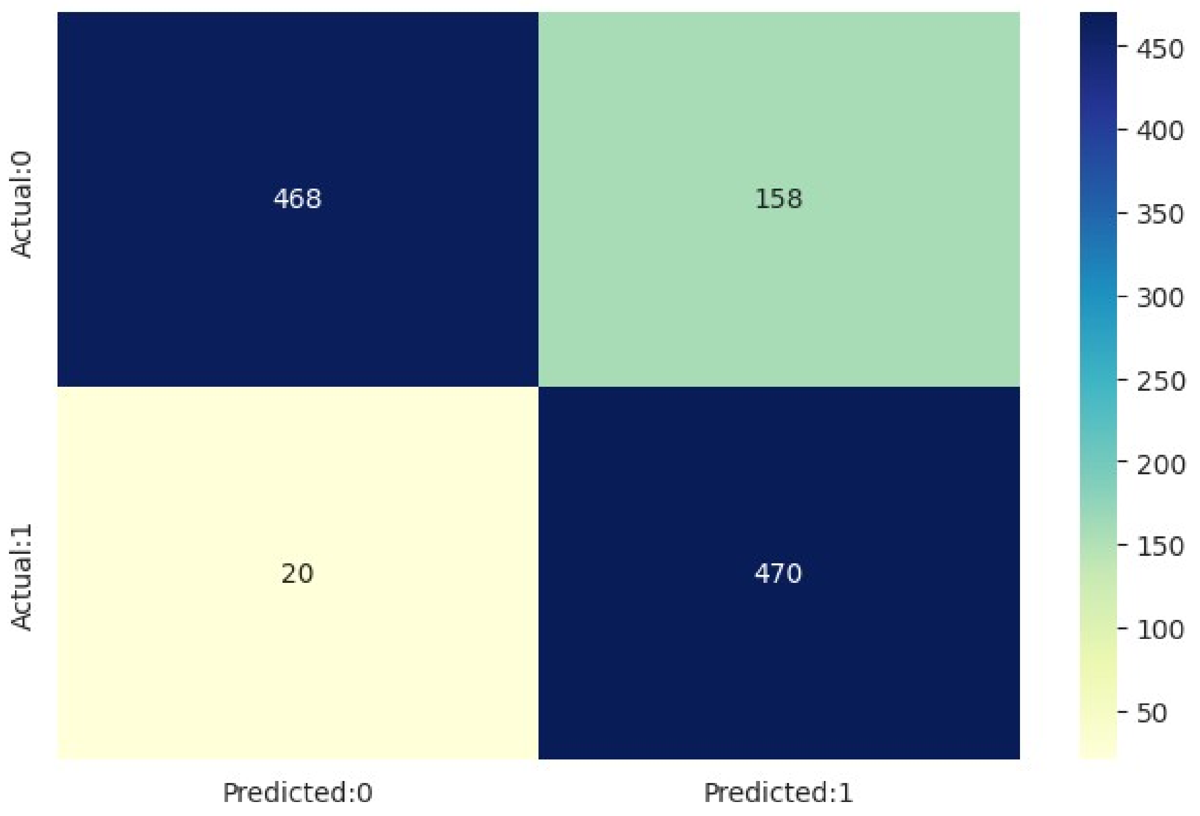
```
# ROC curve and AUC
probs = logistic_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
log_auc = roc_auc_score(y_test, probs)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(log_auc,3)}")
plt.show()
```



k-Nearest Neighbours

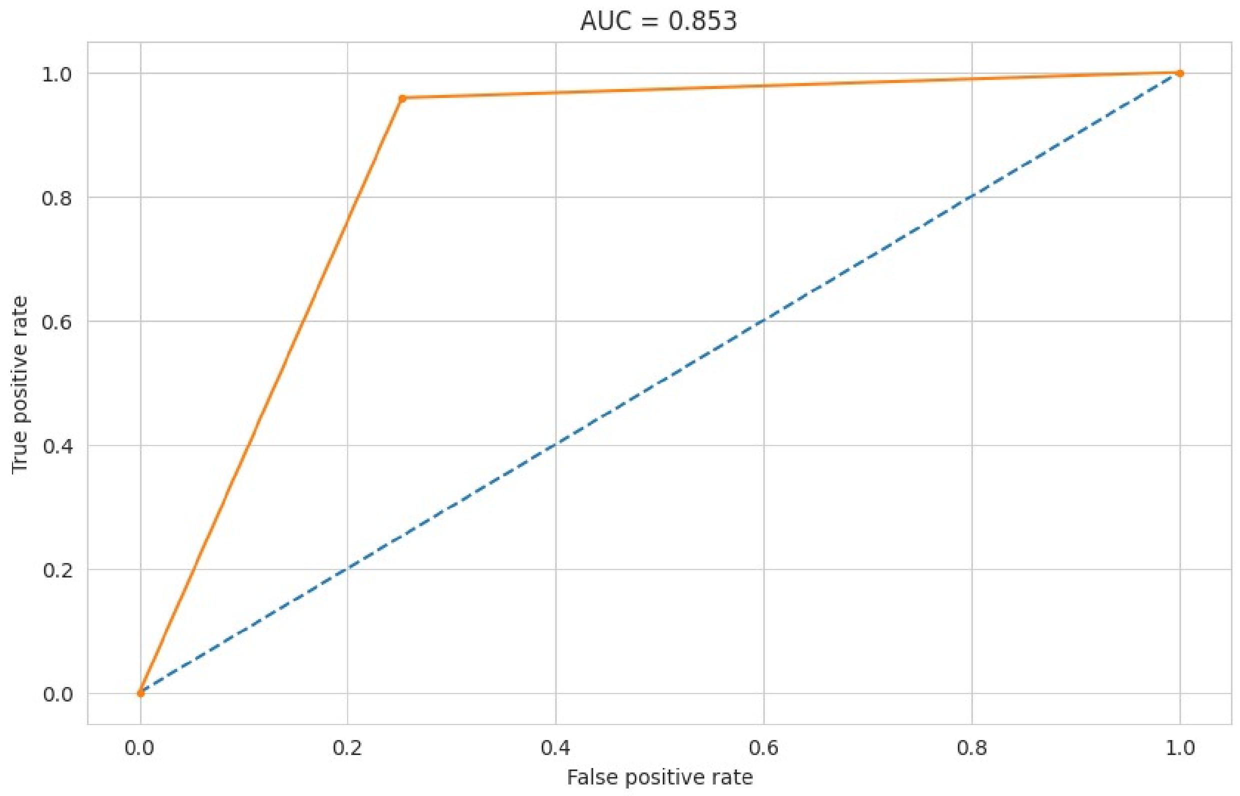
```
from sklearn.neighbors import KNeighborsClassifier
# search for optimun parameters using gridsearch
params= {'n_neighbors': np.arange(1, 10)}
grid_search = GridSearchCV(estimator = KNeighborsClassifier(),
param_grid = params,
                           scoring = 'accuracy', cv = 10, n_jobs = -1)
knn_clf = GridSearchCV(KNeighborsClassifier(),params,cv=3, n_jobs=-1)
# train the model
knn_clf.fit(X_train,y_train)
knn_clf.best_params_
# predictions
knn_predict = knn_clf.predict(X_test)
#accuracy
knn_accuracy = accuracy_score(y_test,knn_predict)
print(f"Using k-nearest neighbours we get an accuracy of
{round(knn_accuracy*100,2)}%")
cm=confusion_matrix(y_test,knn_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1']
,index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

Using k-nearest neighbours we get an accuracy of 84.05%
```

```
<Axes: >
```



```
print(classification_report(y_test,knn_predict))

              precision    recall  f1-score   support

           0       0.96      0.75      0.84       626
           1       0.75      0.96      0.84       490

    accuracy                           0.84      1116
   macro avg       0.85      0.85      0.84      1116
weighted avg       0.87      0.84      0.84      1116


knn_f1 = f1_score(y_test, knn_predict)
print(f'The f1 score for K nearest neignbours is {round(knn_f1*100,2)}
%')

The f1 score for K nearest neignbours is 84.08%

# ROC curve and AUC
probs = knn_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
```

```
knn_auc = roc_auc_score(y_test, probs)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(knn_auc,3)}")
plt.show()
```



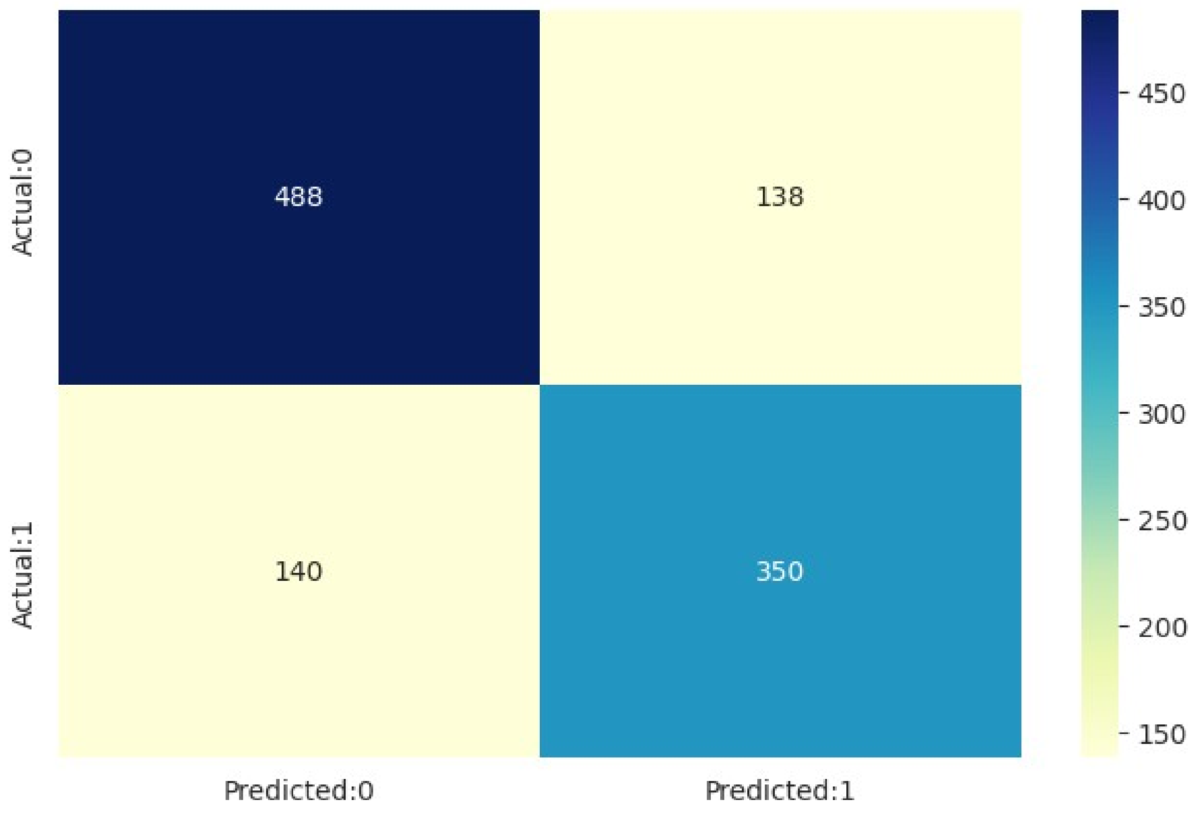
Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
dtree= DecisionTreeClassifier(random_state=7)
# grid search for optimum parameters
params = {'max_features': ['auto', 'sqrt', 'log2'],
          'min_samples_split': [2,3,4,5,6,7,8,9,10,11,12,13,14,15],
          'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10,11]}
tree_clf = GridSearchCV(dtree, param_grid=params, n_jobs=-1)
# train the model
tree_clf.fit(X_train,y_train)
tree_clf.best_params_
# predictions
tree_predict = tree_clf.predict(X_test)
#accuracy
tree_accuracy = accuracy_score(y_test,tree_predict)
print(f"Using Decision Trees we get an accuracy of
{round(tree_accuracy*100,2)}%")
```

```
Using Decision Trees we get an accuracy of 75.09%

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/
_validation.py:528: FitFailedWarning:
770 fits failed out of a total of 2310.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
----------
770 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_vali
dation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line
1382, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line
436, in _validate_params
    validate_parameter_constraints(
  File
"/usr/local/lib/python3.12/dist-packages/sklearn/utils/_param_validati
on.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The
'max_features' parameter of DecisionTreeClassifier must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among
{'log2', 'sqrt'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_searc
h.py:1108: UserWarning: One or more of the test scores are non-finite:
[       nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan 0.73095978 0.73073431
 0.71998855 0.72423985 0.71371529 0.71057553 0.7146109  0.7177459
 0.71819433 0.7172947  0.71102697 0.72760559 0.71303712 0.70878206
 0.72401413 0.72401413 0.72401413 0.71685104 0.71953158 0.72782227
 0.71684552 0.70407755 0.70811067 0.71438794 0.71550199 0.71841452
 0.70900426 0.7083384  0.72087487 0.72087487 0.72087487 0.72087487
 0.72087487 0.71952857 0.71057452 0.7063177  0.70250805 0.71864703
 0.71303637 0.71034981 0.70811368 0.71886245 0.71617463 0.71617463
 0.71617463 0.71617463 0.71617463 0.71617463 0.71617463 0.71101743
 0.71124717 0.71147314 0.69802851 0.70564606 0.71303135 0.71326109
 0.7038566  0.7038566  0.7038566  0.7038566  0.7038566  0.7038566
 0.7038566  0.7038566  0.7038566  0.70273302 0.7119173  0.7002674
 0.70228609 0.71258669 0.70609223 0.70609223 0.70609223 0.70609223
 0.70609223 0.70609223 0.70609223 0.70609223 0.70609223 0.70609223
 0.70609223 0.7022871  0.69578787 0.69355224 0.69959752 0.69959752
 0.69959752 0.69959752 0.69959752 0.69959752 0.69959752 0.69959752
 0.69959752 0.69959752 0.69959752 0.69959752 0.69959752 0.69354697
 0.69354873 0.69354873 0.69354873 0.69354873 0.69354873 0.69354873
 0.69354873 0.69354873 0.69354873 0.69354873 0.69354873 0.69354873
 0.69354873 0.69354873 0.67965843 0.67965843 0.67965843 0.67965843
 0.67965843 0.67965843 0.67965843 0.67965843 0.67965843 0.67965843
 0.67965843 0.67965843 0.67965843 0.67965843 0.68839554 0.68839554
 0.68839554 0.68839554 0.68839554 0.68839554 0.68839554 0.68839554
 0.68839554 0.68839554 0.68839554 0.68839554 0.68839554 0.68839554
 0.69018751 0.69018751 0.69018751 0.69018751 0.69018751 0.69018751
 0.69018751 0.69018751 0.69018751 0.69018751 0.69018751 0.69018751
 0.69018751 0.69018751 0.72356972 0.73745324 0.7235647  0.72088089
 0.73611196 0.71595318 0.71730374 0.71572695 0.7150533  0.71975781
 0.71079673 0.71797112 0.71079999 0.70453201 0.72670547 0.72670547
 0.72670547 0.71729219 0.71998478 0.71528078 0.72178026 0.72020649
 0.71371002 0.70564932 0.71303964 0.70474894 0.71192107 0.71394001
 0.71438392 0.71438392 0.71438392 0.71438392 0.71438392 0.72020975
 0.71729696 0.72311752 0.72557862 0.71729345 0.71595243 0.71236724
 0.71707526 0.71281442 0.69870342 0.69870342 0.69870342 0.69870342
 0.69870342 0.69870342 0.69870342 0.71863422 0.70564555 0.70945671
 0.71259346 0.72088089 0.70743852 0.71953384 0.7211026  0.7211026
 0.7211026  0.7211026  0.7211026  0.7211026  0.7211026  0.7211026
 0.7211026  0.70094231 0.70855885 0.7148316  0.707665   0.70900351
 0.70878381 0.70878381 0.70878381 0.70878381 0.70878381 0.70878381
 0.70878381 0.70878381 0.70878381 0.70878381 0.70878381 0.70833513
 0.70788771 0.7038551  0.71505155 0.71505155 0.71505155 0.71505155
 0.71505155 0.71505155 0.71505155 0.71505155 0.71505155 0.71505155
 0.71505155 0.71505155 0.71505155 0.69915612 0.70094256 0.70094256
 0.70094256 0.70094256 0.70094256 0.70094256 0.70094256 0.70094256
 0.70094256 0.70094256 0.70094256 0.70094256 0.70094256 0.70094256
 0.68840333 0.68840333 0.68840333 0.68840333 0.68840333 0.68840333
 0.68840333 0.68840333 0.68840333 0.68840333 0.68840333 0.68840333
 0.68840333 0.68840333 0.69735838 0.69735838 0.69735838 0.69735838
 0.69735838 0.69735838 0.69735838 0.69735838 0.69735838 0.69735838
 0.69735838 0.69735838 0.69735838 0.69735838 0.68458891 0.68458891
 0.68458891 0.68458891 0.68458891 0.68458891 0.68458891 0.68458891
 0.68458891 0.68458891 0.68458891 0.68458891 0.68458891 0.68458891]
  warnings.warn(
```

```
cm=confusion_matrix(y_test,tree_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1']
,index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

```
<Axes: >
```



```
print(classification_report(y_test,tree_predict))
```

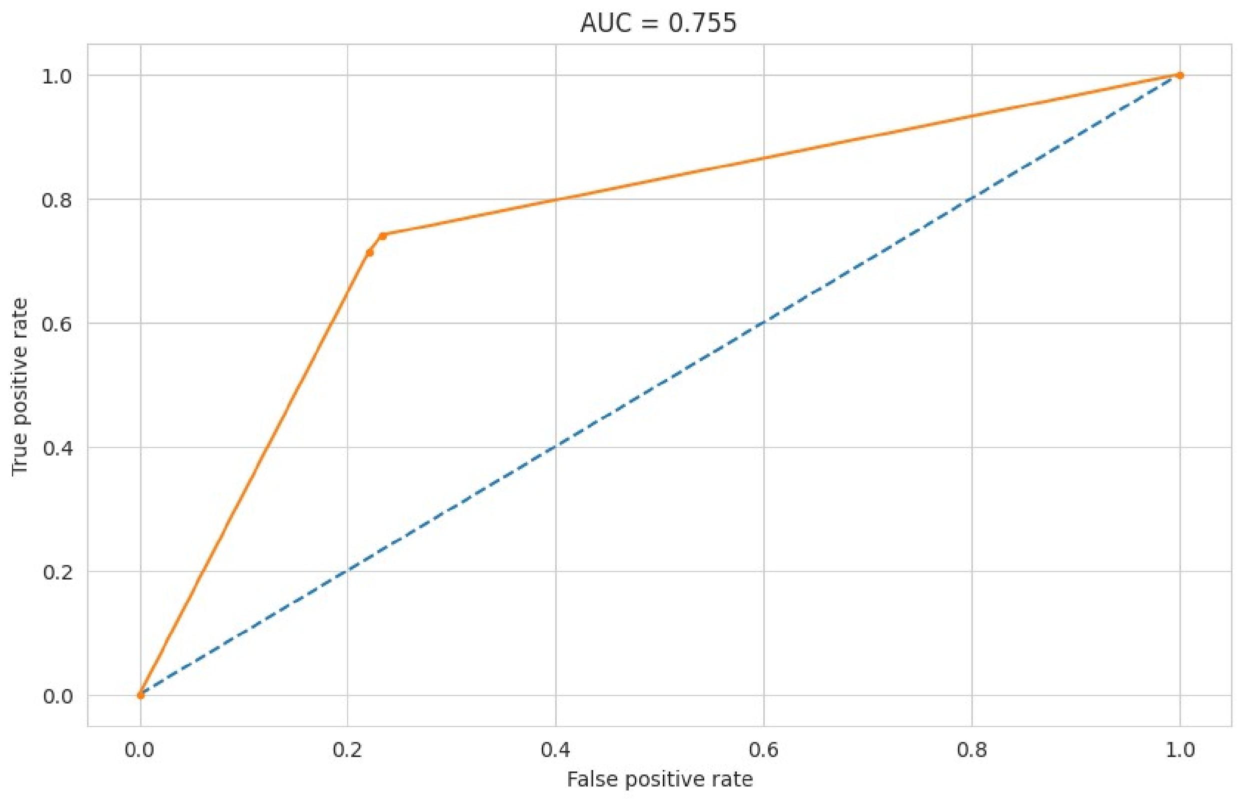
```
              precision    recall  f1-score   support

           0       0.78      0.78      0.78       626
           1       0.72      0.71      0.72       490

    accuracy                           0.75      1116
   macro avg       0.75      0.75      0.75      1116
weighted avg       0.75      0.75      0.75      1116
```

```
tree_f1 = f1_score(y_test, tree_predict)
print(f'The f1 score Descision trees is {round(tree_f1*100,2)}%')
```

```
The f1 score Descision trees is 71.57%
```

```
# ROC curve and AUC
probs = tree_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
tree_auc = roc_auc_score(y_test, probs)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
```

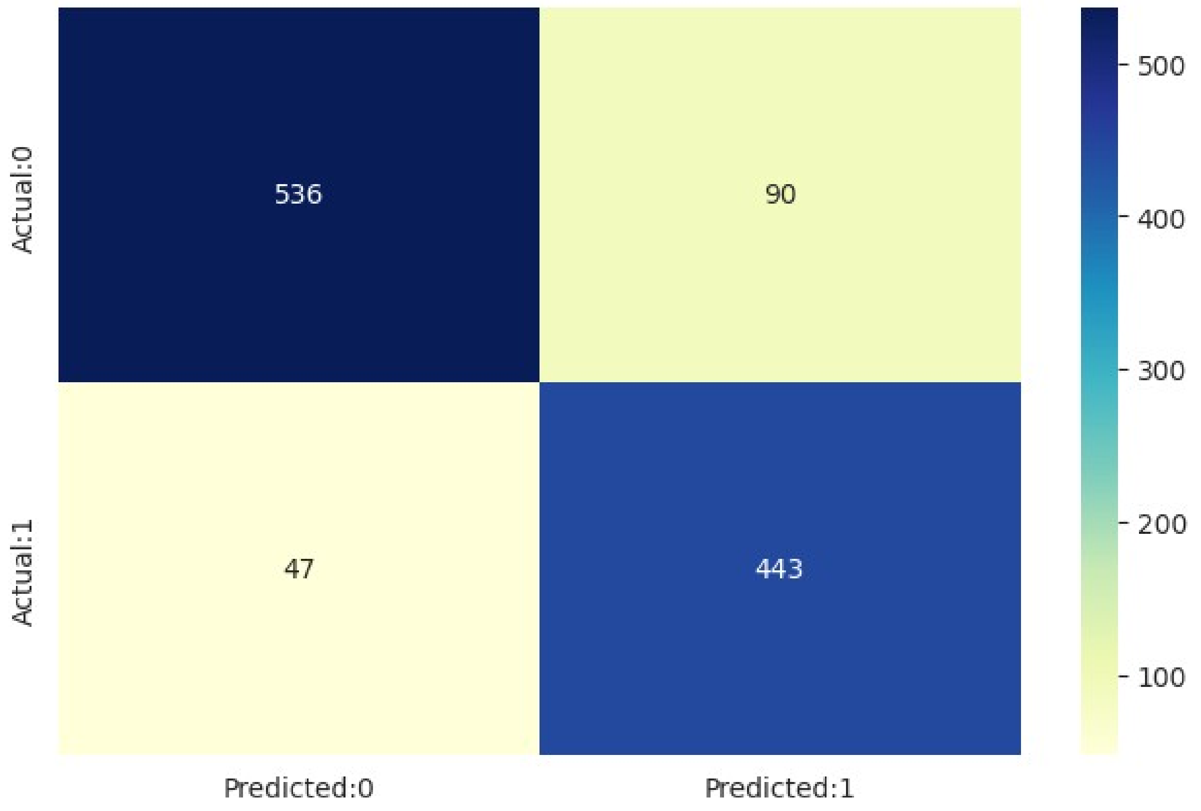
```
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(tree_auc,3)}")
plt.show()
```



Support Vector Machine

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

#grid search for optimum parameters
Cs = [0.001, 0.01, 0.1, 1, 10]
gammas = [0.001, 0.01, 0.1, 1]
param_grid = {'C': Cs, 'gamma' : gammas}
svm_clf = GridSearchCV(SVC(kernel='rbf', probability=True),
param_grid, cv=10)

# train the model
svm_clf.fit(X_train,y_train)
svm_clf.best_params_
```

```
{'C': 10, 'gamma': 1}

# predictions
svm_predict = svm_clf.predict(X_test)

#accuracy
svm_accuracy = accuracy_score(y_test,svm_predict)
print(f"Using SVM we get an accuracy of {round(svm_accuracy*100,2)}%")

Using SVM we get an accuracy of 87.72%

cm=confusion_matrix(y_test,svm_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1']
,index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

<Axes: >
```



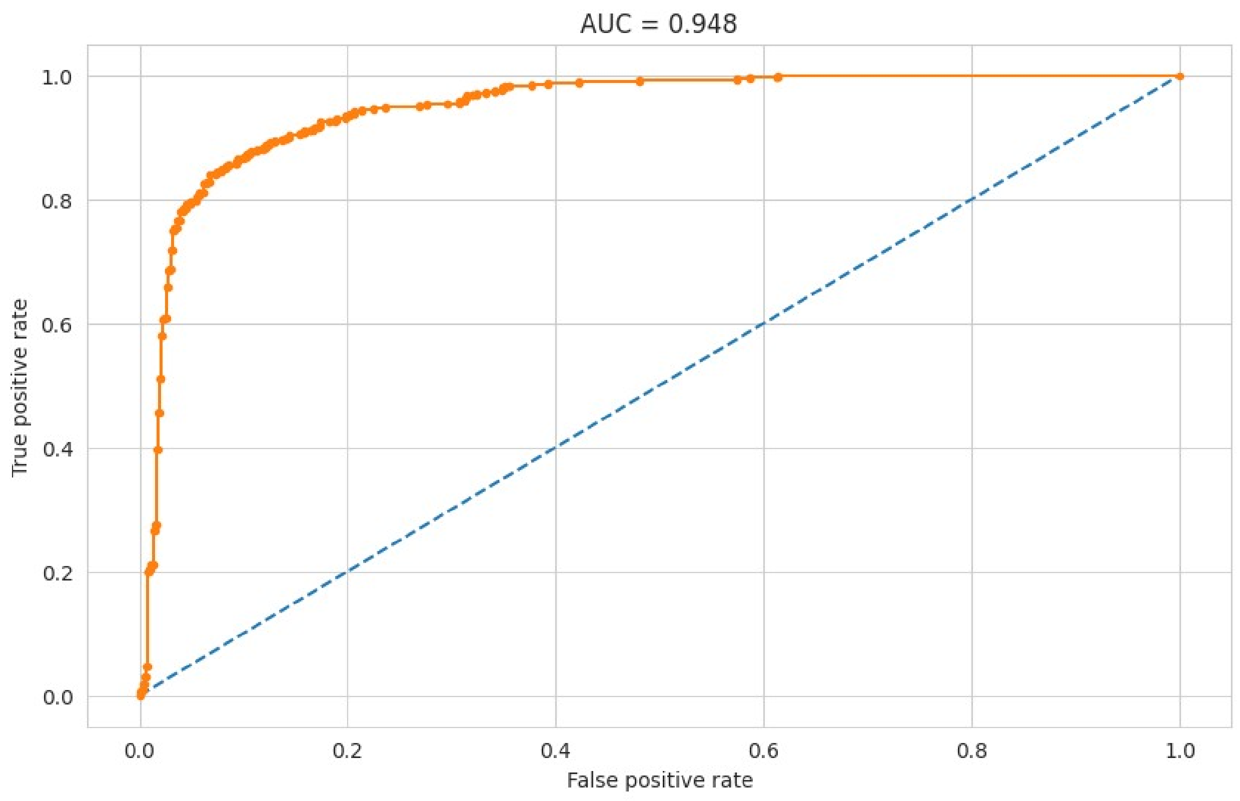
```
print(classification_report(y_test,svm_predict))

              precision    recall  f1-score   support

           0       0.92      0.86      0.89       626
           1       0.83      0.90      0.87       490
```

```
    accuracy                           0.88      1116
   macro avg       0.88      0.88      0.88      1116
weighted avg       0.88      0.88      0.88      1116
```

```python
svm_f1 = f1_score(y_test, svm_predict)
print(f'The f1 score for SVM is {round(svm_f1*100,2)}%')
```

```
The f1 score for SVM is 86.61%
```

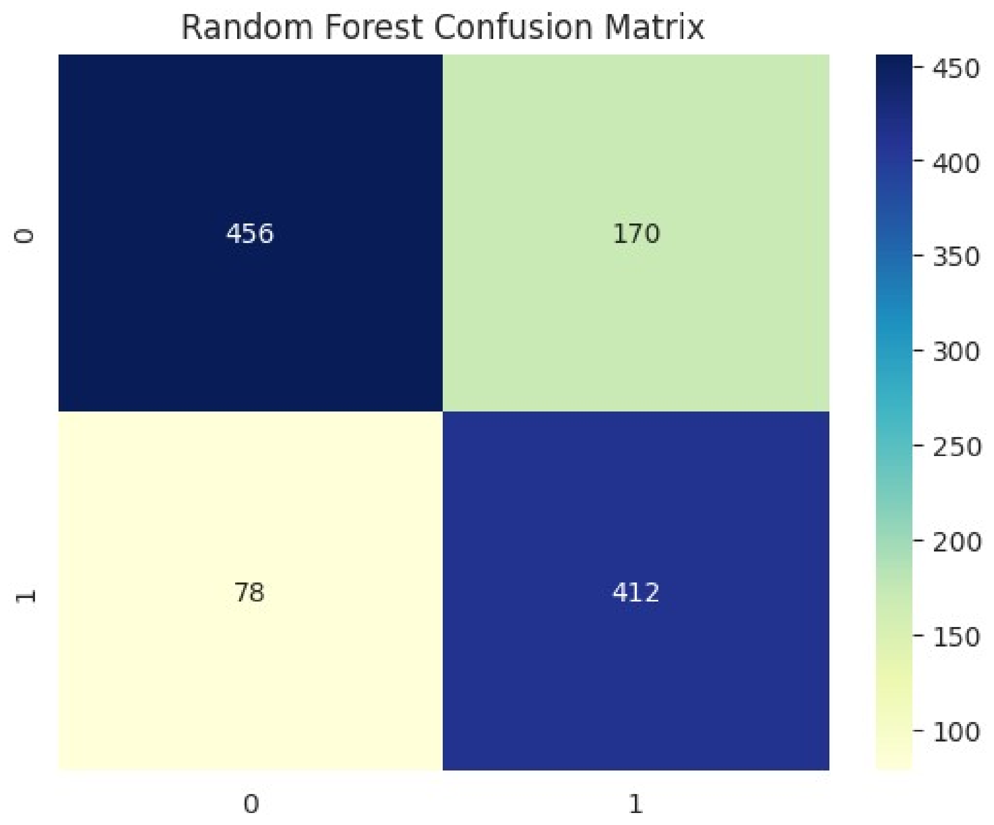
```python
# ROC curve and AUC
probs = svm_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
svm_auc = roc_auc_score(y_test, probs)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(svm_auc,3)}")
plt.show()
```



Random Forest

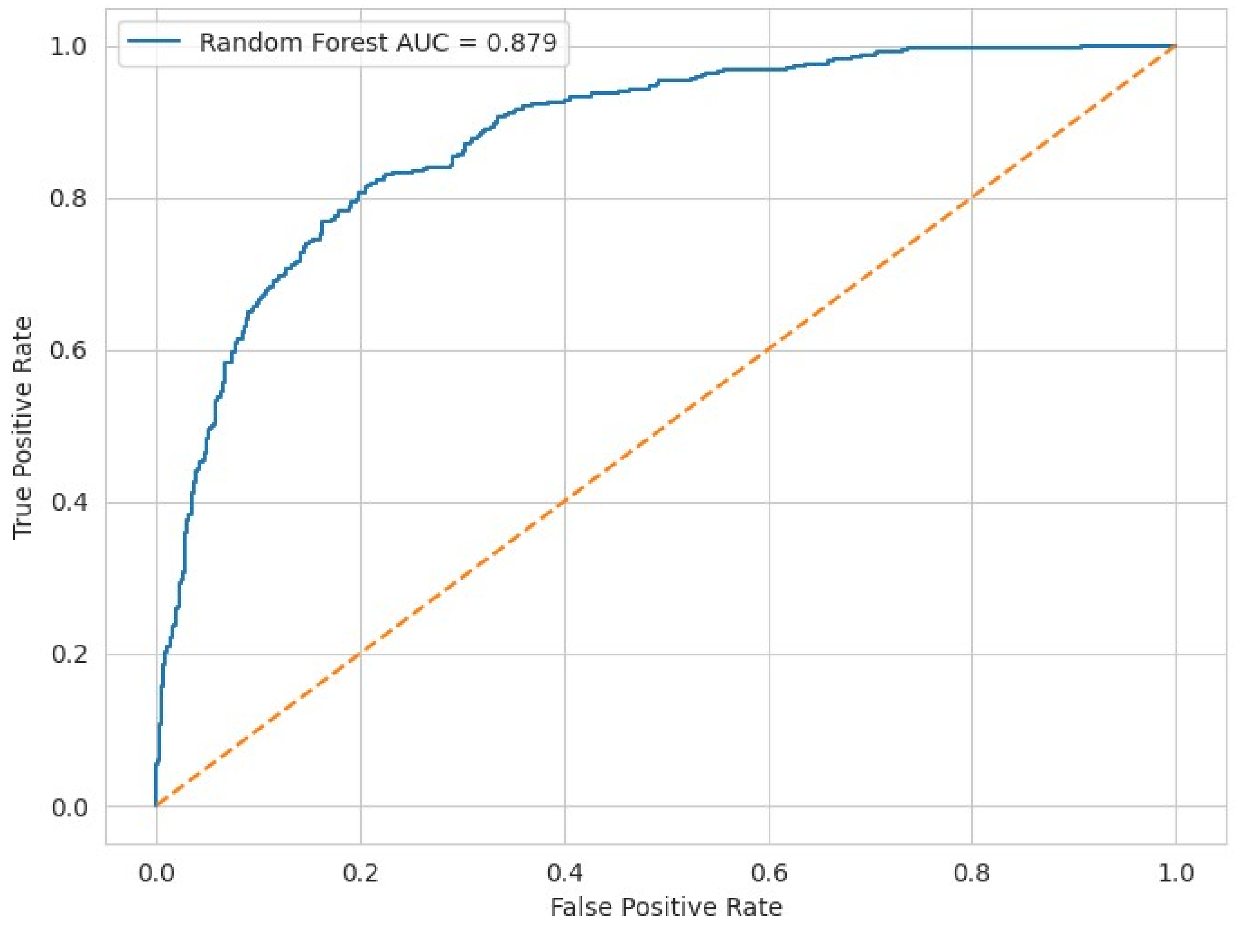
```
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb

# --- Random Forest Classifier ---
rf_clf = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    class_weight='balanced',
    random_state=42
)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)

rf_accuracy = accuracy_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred)
print(f"Using Random Forest we get an accuracy of
{round(rf_accuracy*100,2)}%")
print(f"The F1 score for Random Forest is {round(rf_f1*100,2)}%")

Using Random Forest we get an accuracy of 77.78%
The F1 score for Random Forest is 76.87%

# Confusion Matrix
cm = confusion_matrix(y_test, rf_pred)
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu')
plt.title("Random Forest Confusion Matrix")
plt.show()

print(classification_report(y_test, rf_pred))
```



```
              precision    recall  f1-score   support

           0       0.85      0.73      0.79       626
           1       0.71      0.84      0.77       490

    accuracy                           0.78      1116
   macro avg       0.78      0.78      0.78      1116
weighted avg       0.79      0.78      0.78      1116
```

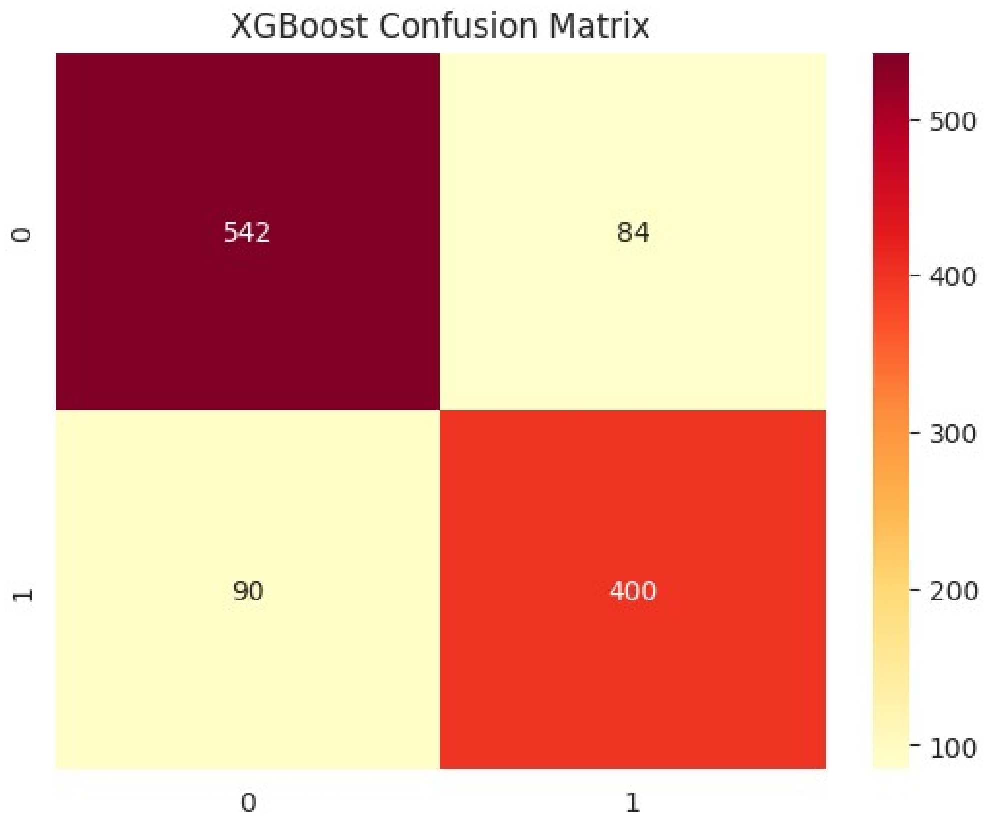
```
# ROC curve and AUC
rf_probs = rf_clf.predict_proba(X_test)[:, 1]
rf_auc = roc_auc_score(y_test, rf_probs)
fpr, tpr, _ = roc_curve(y_test, rf_probs)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"Random Forest AUC = {round(rf_auc,3)}")
plt.plot([0,1],[0,1],'--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



XGBoost Classifier

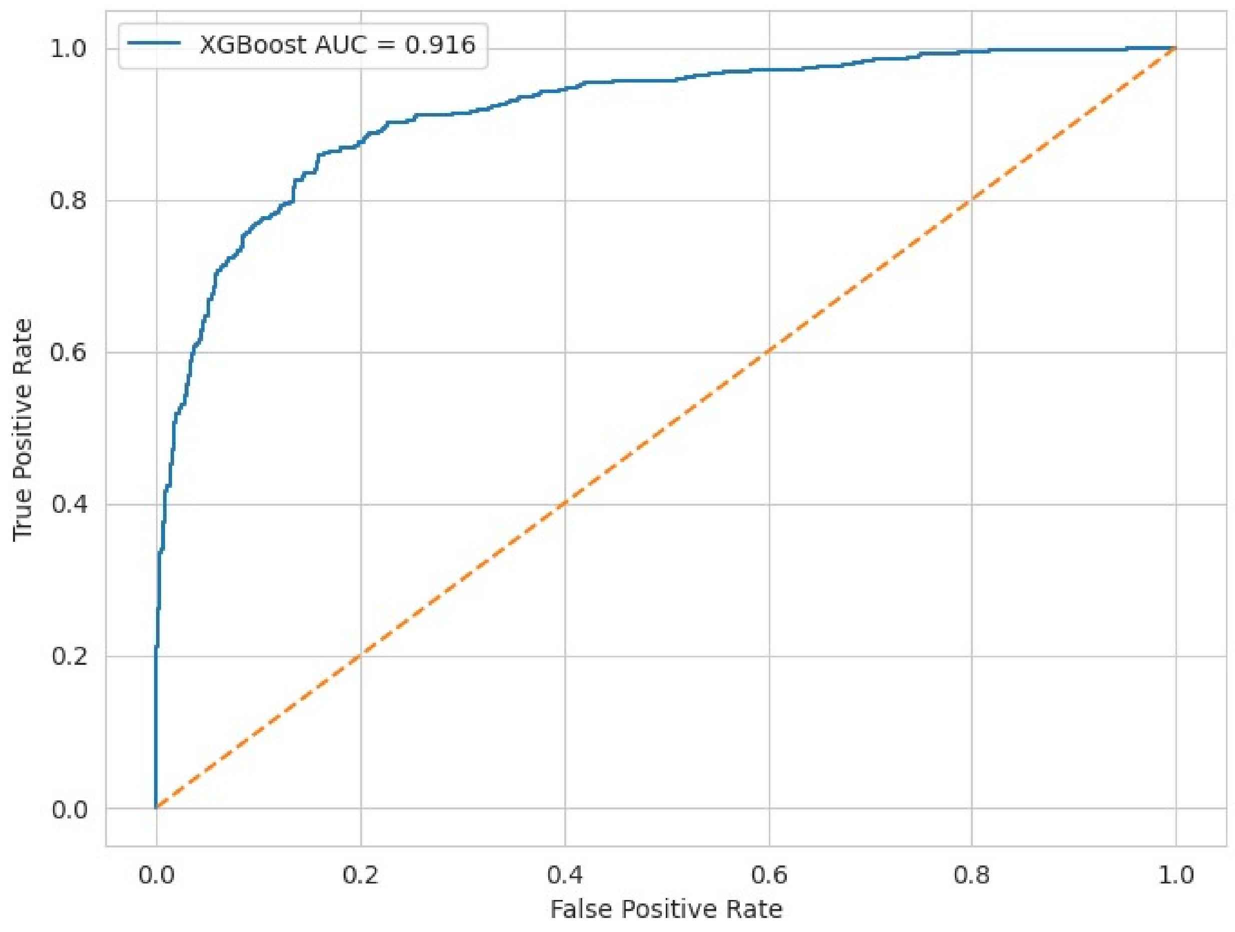
```
# --- XGBoost Classifier ---
xgb_clf = xgb.XGBClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    eval_metric='logloss'
)
xgb_clf.fit(X_train, y_train)
xgb_pred = xgb_clf.predict(X_test)

xgb_accuracy = accuracy_score(y_test, xgb_pred)
xgb_f1 = f1_score(y_test, xgb_pred)
```

```
print(f"Using XGBoost we get an accuracy of
{round(xgb_accuracy*100,2)}%")
print(f"The F1 score for XGBoost is {round(xgb_f1*100,2)}%")

Using XGBoost we get an accuracy of 84.41%
The F1 score for XGBoost is 82.14%

cm = confusion_matrix(y_test, xgb_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='YlOrRd')
plt.title("XGBoost Confusion Matrix")
plt.show()

print(classification_report(y_test, xgb_pred))
```



```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       626
           1       0.83      0.82      0.82       490

    accuracy                           0.84      1116
   macro avg       0.84      0.84      0.84      1116
weighted avg       0.84      0.84      0.84      1116
```

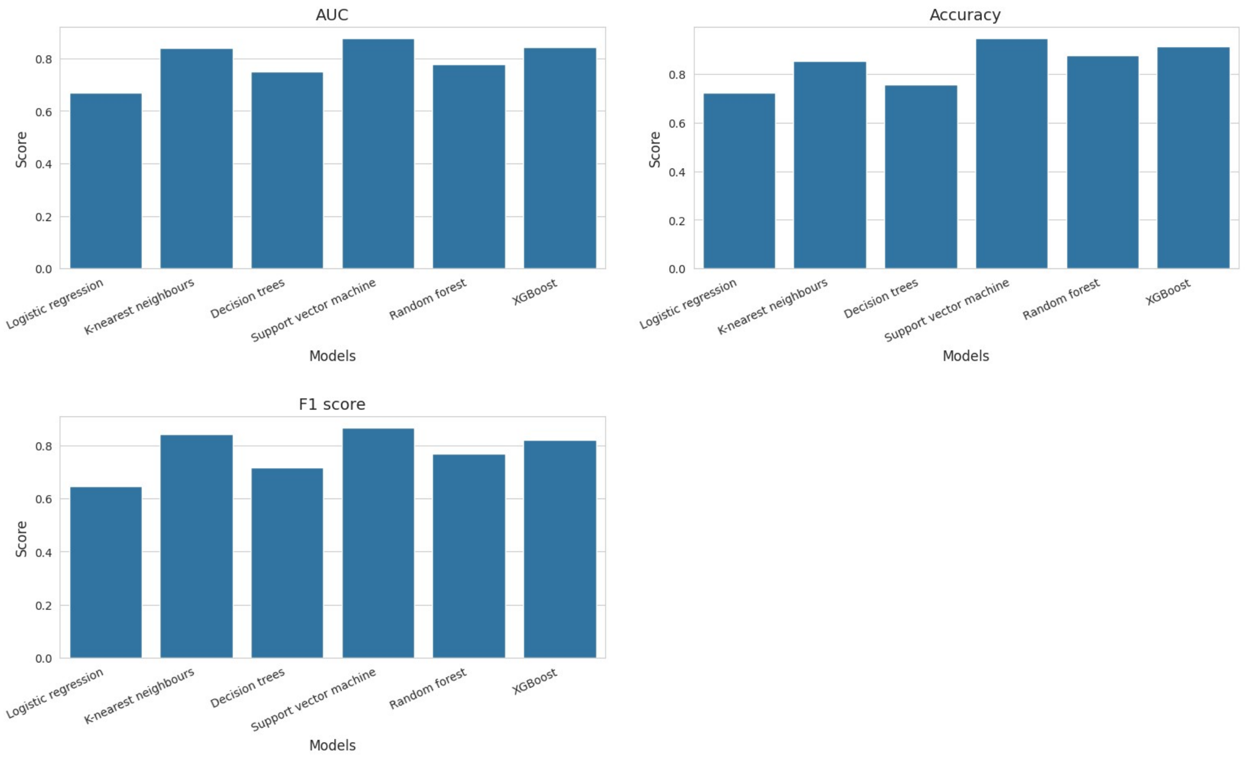
```
# ROC curve and AUC
xgb_probs = xgb_clf.predict_proba(X_test)[:, 1]
xgb_auc = roc_auc_score(y_test, xgb_probs)
fpr, tpr, _ = roc_curve(y_test, xgb_probs)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"XGBoost AUC = {round(xgb_auc,3)}")
plt.plot([0,1],[0,1],'--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



```
comparison = pd.DataFrame({
    "Logistic regression":{'Accuracy':log_accuracy, 'AUC':log_auc, 'F1
score':logistic_f1},
    "K-nearest neighbours":{'Accuracy':knn_accuracy, 'AUC':knn_auc,
'F1 score':knn_f1},
    "Decision trees":{'Accuracy':tree_accuracy, 'AUC':tree_auc, 'F1
score':tree_f1},
    "Support vector machine":{'Accuracy':svm_accuracy, 'AUC':svm_auc,
'F1 score':svm_f1},
    "Random forest": {'Accuracy': rf_accuracy,'AUC': rf_auc, 'F1
score': rf_f1},
    "XGBoost": {'Accuracy': xgb_accuracy,'AUC': xgb_auc, 'F1 score':
xgb_f1}
}).T
comparison
```

```
{"summary":"{\n  \"name\": \"comparison\",\n  \"rows\": 6,\n
\"fields\": [\n    {\n      \"column\": \"Accuracy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.07649132373802281,\n        \"min\": 0.6693548387096774,\n
\"max\": 0.8772401433691757,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          0.6693548387096774,\n
0.8405017921146953,\n          0.8440860215053764\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"AUC\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.08917009810636684,\n
\"min\": 0.7233650648757907,\n        \"max\": 0.9483764751907153,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
0.7233650648757907,\n          0.8533937536676011,\n
0.9158635978353002\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"F1
score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.08396792848953102,\n        \"min\": 0.6448508180943214,\n
\"max\": 0.8660801564027371,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          0.6448508180943214,\n
0.8407871198568873,\n          0.8213552361396304\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"comparison"}
```

```
fig = plt.gcf()
fig.set_size_inches(16, 10)
titles = ['AUC', 'Accuracy', 'F1 score']

for i, label in enumerate(comparison.columns):
    plt.subplot(2, 2, i + 1)
    sns.barplot(x=comparison.index, y=comparison[label],
data=comparison)
    plt.title(titles[i], fontsize=14)
    plt.ylabel('Score', fontsize=12)
    plt.xlabel('Models', fontsize=12)
    plt.xticks(rotation=25, ha='right', fontsize=10)  # rotate + align
    plt.tight_layout(pad=3.0)  # adds space between subplots

plt.show()
```


AUC
Accuracy
F1 score
Score
Models
0.0
0.2
0.4
0.6
0.8
Logistic regression
K-nearest neighbours
Decision trees
Support vector machine
Random forest
XGBoost