```
#Healthcare PGP -Course-end Project
#Description
#Problem Statement
#NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chroni
#The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certa
#Build a model to accurately predict whether the patients in the dataset have diabetes or not.
#Dataset Description
#The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of p

#Variables  Description
#Pregnancies    Number of times pregnant
#Glucose    Plasma glucose concentration in an oral glucose tolerance test
#BloodPressure  Diastolic blood pressure (mm Hg)
#SkinThickness  Triceps skinfold thickness (mm)
#Insulin    Two hour serum insulin
#BMI    Body Mass Index
#DiabetesPedigreeFunction   Diabetes pedigree function
#Age    Age in years
#Outcome    Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0
```

```
#Task 1: Data Exploration:
```

```
#1. Perform descriptive analysis. Understand the variables and their corresponding values.
#On the columns below, a value of zero does not make sense and thus indicates missing value: Glucose, BloodPressure, SkinThickness, Insulin,
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
print("Lib imported")
```

```
Lib imported
```

```
data=pd.read_csv("health care diabetes.csv")
```

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
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
```

```
from pandas.core.base import value_counts
data['Outcome'].value_counts()
```

```
0    500
1    268
```

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

```
dis_data=data[data['Outcome']==1]
dis_data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1

```
dis_data.shape
```

```
(268, 9)
```

```
dis_data['Glucose'].value_counts()
```

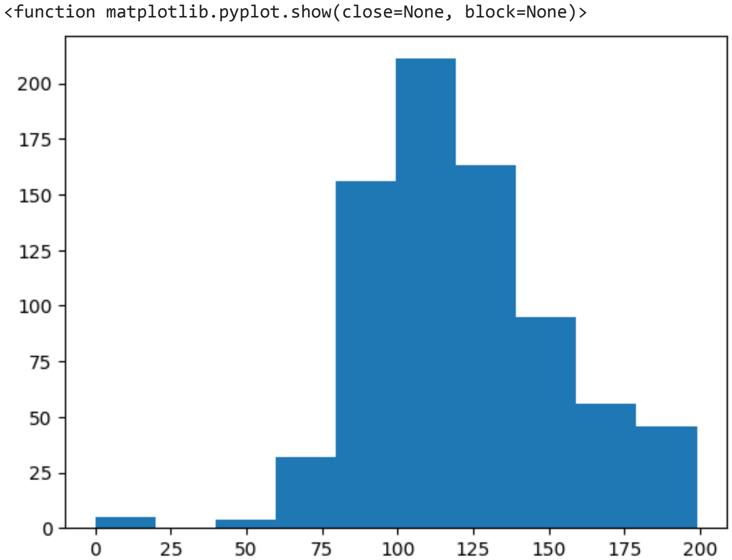
```
125    7
128    6
129    6
115    6
158    6
      ..
165    1
116    1
193    1
172    1
190    1
Name: Glucose, Length: 104, dtype: int64
```

```
# 2. Visually explore these variables using histograms. Treat the missing values accordingly.
```

```
#univariate analysis
```

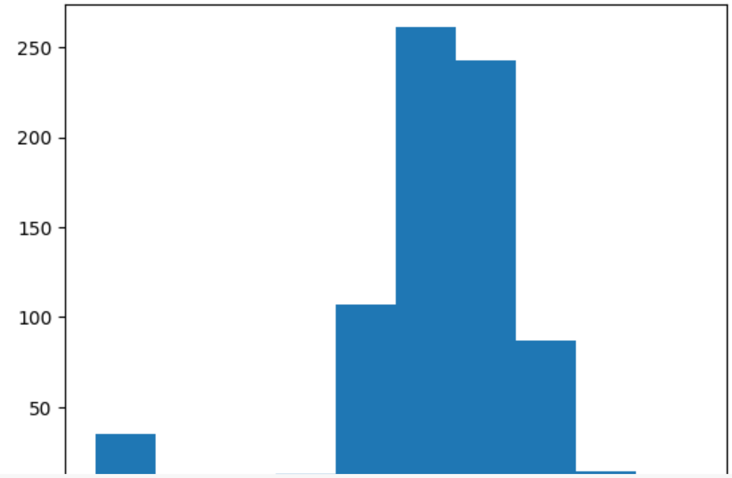
```
plt.hist(data['Glucose'])
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



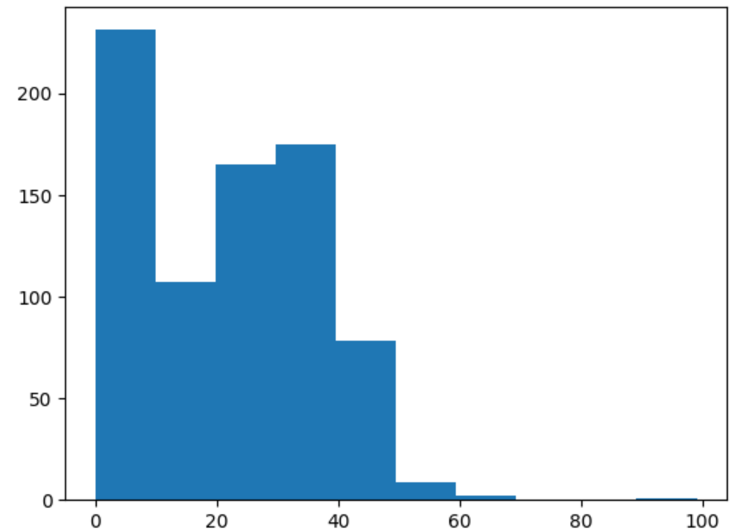

```
plt.hist(data['BloodPressure'])
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



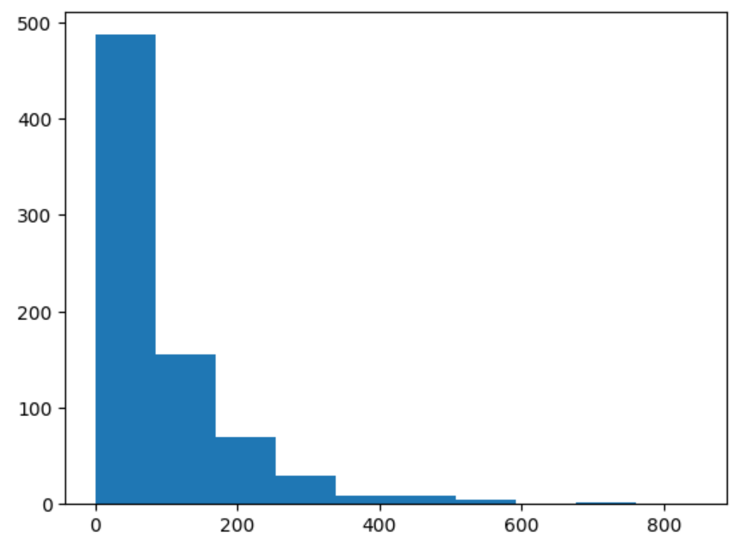

```
plt.hist(data['SkinThickness'])
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



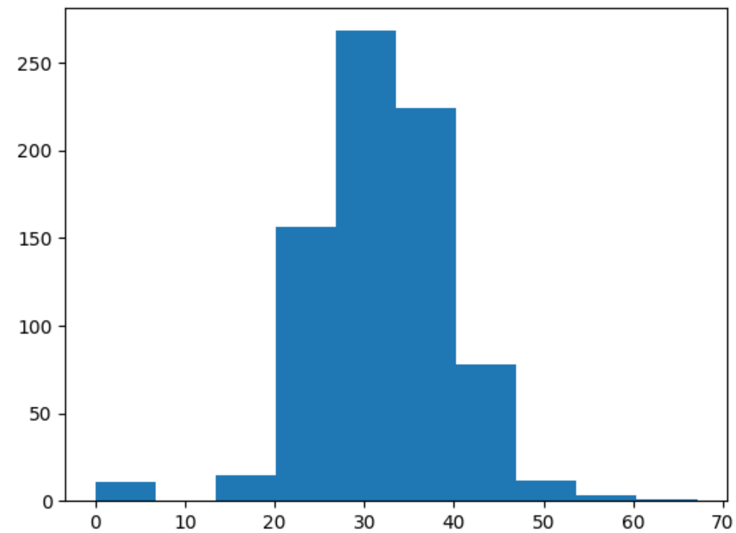

```
plt.hist(data['Insulin'])
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
plt.hist(data['BMI'])
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
# 3. There are integer and float data type variables in this dataset.
```

```
data[data['Glucose']==0].shape
```

```
(5, 9)
```

```
data.loc [data['Glucose']==0,:]
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75	1	0	48	20	0	24.7	0.140	22	0
182	1	0	74	20	23	27.7	0.299	21	0
342	1	0	68	35	0	32.0	0.389	22	0
349	5	0	80	32	0	41.0	0.346	37	1
502	6	0	68	41	0	39.0	0.727	41	1

```
data['Glucose']=data['Glucose'].replace(0,data['Glucose'].mean())
```

```
data[data['Glucose']==0]
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

```
data.loc [data['BloodPressure']==0,:]
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115.0	0	0	0	35.3	0.134	29	0
15	7	100.0	0	0	0	30.0	0.484	32	1
49	7	105.0	0	0	0	0.0	0.305	24	0
60	2	84.0	0	0	0	0.0	0.304	21	0
78	0	131.0	0	0	0	43.2	0.270	26	1
81	2	74.0	0	0	0	0.0	0.102	22	0
172	2	87.0	0	23	0	28.9	0.773	25	0
193	11	135.0	0	0	0	52.3	0.578	40	1
222	7	119.0	0	0	0	25.2	0.209	37	0
261	3	141.0	0	0	0	30.0	0.761	27	1
266	0	138.0	0	0	0	36.3	0.933	25	1
269	2	146.0	0	0	0	27.5	0.240	28	1
300	0	167.0	0	0	0	32.3	0.839	30	1
332	1	180.0	0	0	0	43.3	0.282	41	1
336	0	117.0	0	0	0	33.8	0.932	44	0
347	3	116.0	0	0	0	23.5	0.187	23	0
357	13	129.0	0	30	0	39.9	0.569	44	1
426	0	94.0	0	0	0	0.0	0.256	25	0
430	2	99.0	0	0	0	22.2	0.108	23	0
435	0	141.0	0	0	0	42.4	0.205	29	1
453	2	119.0	0	0	0	19.6	0.832	72	0
468	8	120.0	0	0	0	30.0	0.183	38	1
484	0	145.0	0	0	0	44.2	0.630	31	1
494	3	80.0	0	0	0	0.0	0.174	22	0
522	6	114.0	0	0	0	0.0	0.189	26	0
533	6	91.0	0	0	0	29.8	0.501	31	0
535	4	132.0	0	0	0	32.9	0.302	23	1
589	0	73.0	0	0	0	21.1	0.342	25	0
601	6	96.0	0	0	0	23.7	0.190	28	0
604	4	183.0	0	0	0	28.4	0.212	36	1
619	0	119.0	0	0	0	32.4	0.141	24	1
643	4	90.0	0	0	0	28.0	0.610	31	0
697	0	99.0	0	0	0	25.0	0.253	22	0
703	2	129.0	0	0	0	38.5	0.304	41	0
706	10	115.0	0	0	0	0.0	0.261	30	1

```
data[data['BloodPressure']==0].shape
```

```
(35, 9)
```

```
data['BloodPressure']=data['BloodPressure'].replace(0,data['BloodPressure'].mean())
```

```
data[data['BloodPressure']==0]
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

```
data[data['SkinThickness']==0].shape
```

```
(227, 9)
```

```
# not normal dist so use median
```

```
data['SkinThickness']=data['SkinThickness'].replace(0,data['SkinThickness'].median())
```

```
data[data['SkinThickness']==0]
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

```
data[data['Insulin']==0].shape
```

```
(374, 9)
```

```
data['Insulin']=data['Insulin'].replace(0,data['Insulin'].mean() )
```

```
data[data['Insulin']==0]
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

```
data[data['BMI']==0]
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
9	8	125.0	96.000000	23	79.799479	0.0	0.232	54	1
49	7	105.0	69.105469	23	79.799479	0.0	0.305	24	0
60	2	84.0	69.105469	23	79.799479	0.0	0.304	21	0
81	2	74.0	69.105469	23	79.799479	0.0	0.102	22	0
145	0	102.0	75.000000	23	79.799479	0.0	0.572	21	0
371	0	118.0	64.000000	23	89.000000	0.0	1.731	21	0
426	0	94.0	69.105469	23	79.799479	0.0	0.256	25	0
494	3	80.0	69.105469	23	79.799479	0.0	0.174	22	0
522	6	114.0	69.105469	23	79.799479	0.0	0.189	26	0
684	5	136.0	82.000000	23	79.799479	0.0	0.640	69	0
706	10	115.0	69.105469	23	79.799479	0.0	0.261	30	1

```
data['BMI']=data['BMI'].replace(0,data['BMI'].mean() )
```

```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	27.334635	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.229014	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	23.000000	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
#Task: Create a count (frequency) plot describing the data types and the count of variables.
```

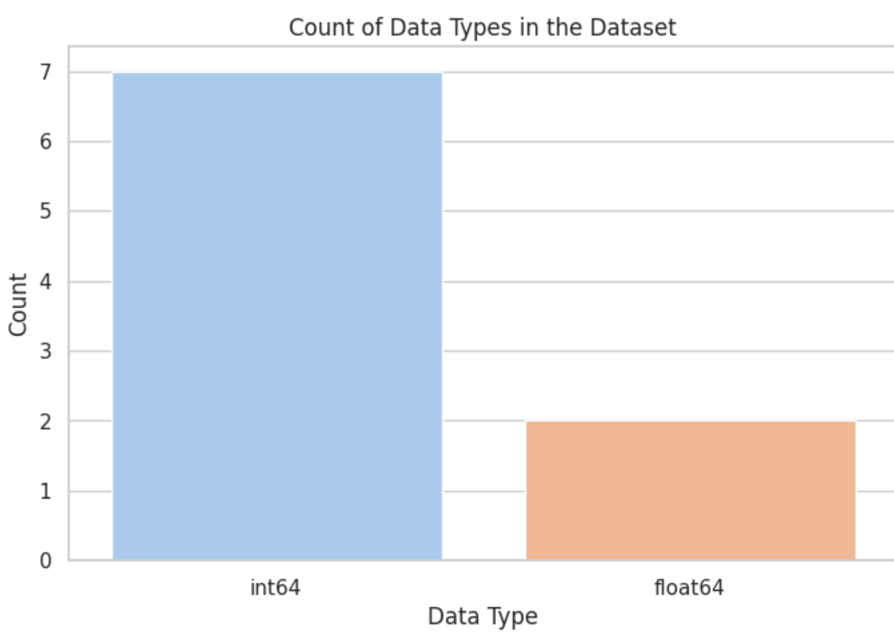
```
data_description = {
    'Pregnancies': 'int64',
    'Glucose': 'int64',
    'BloodPressure': 'int64',
    'SkinThickness': 'int64',
    'Insulin': 'int64',
    'BMI': 'float64',
    'DiabetesPedigreeFunction': 'float64',
    'Age': 'int64',
    'Outcome': 'int64',
}
```

```
# Creating a DataFrame from the data description
data_types_df = pd.DataFrame(data_description.items(), columns=['Variable', 'Data Type'])
```

```
# Counting the occurrences of each data type
data_type_counts = data_types_df['Data Type'].value_counts()
```

```
# Creating a count plot using Seaborn
plt.figure(figsize=(8, 5))
sns.set(style="whitegrid")
ax = sns.barplot(x=data_type_counts.index, y=data_type_counts.values, palette="pastel")
ax.set_title('Count of Data Types in the Dataset')
ax.set_xlabel('Data Type')
ax.set_ylabel('Count')

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




```
#Task 2: Data Exploration:
#1. Check the balance of the data by plotting the count of outcomes by their value.
#Describe your findings and plan future course of action.
#2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
#3. Perform correlation analysis. Visually explore it using a heat map.
```

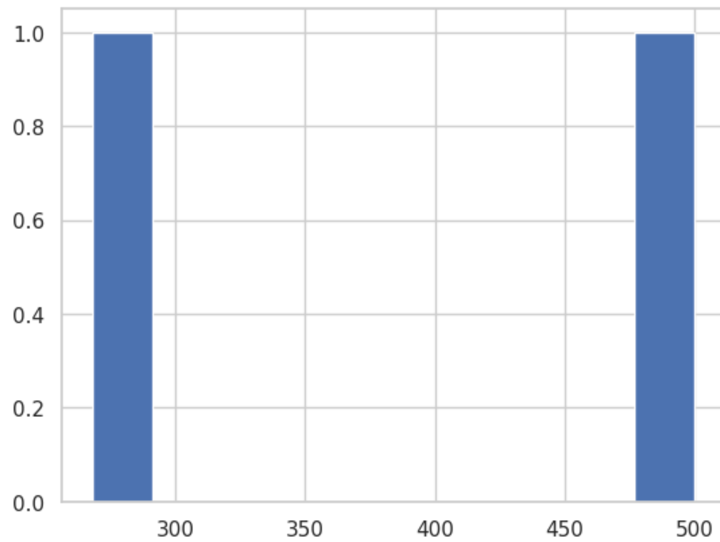
```
data['Outcome'].value_counts()
```

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

```
# Count plot for Outcome
```

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

<Axes: >

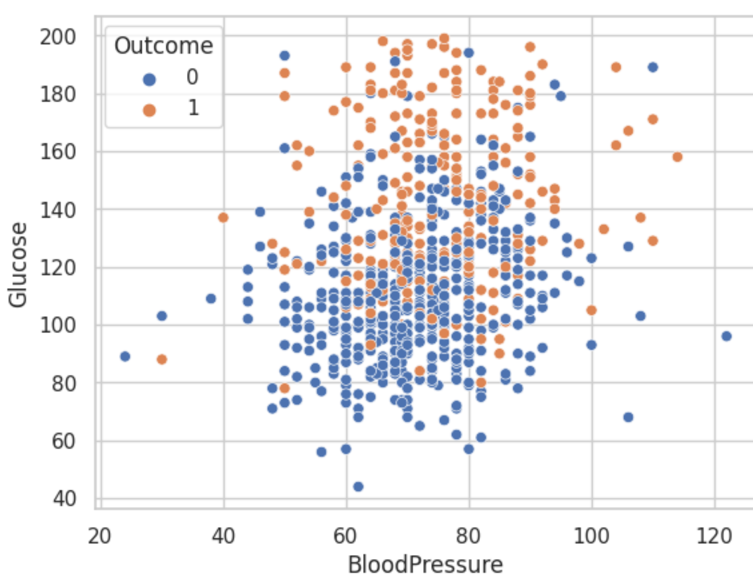


```
data.head()
```

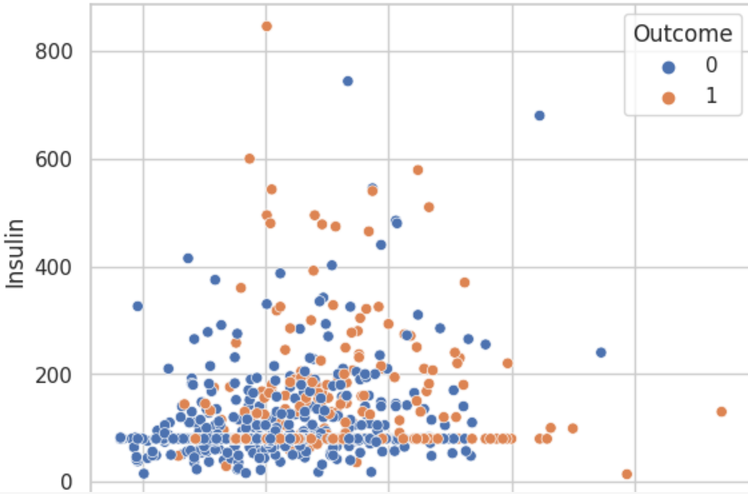
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29	79.799479	26.6	0.351	31	0
2	8	183.0	64.0	23	79.799479	23.3	0.672	32	1
3	1	89.0	66.0	23	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35	168.000000	43.1	2.288	33	1

```
# Bivariate Analysis
```

```
sns.scatterplot(x='BloodPressure',y='Glucose',hue='Outcome',data=data)
plt.show()
```




```
sns.scatterplot(x='BMI',y='Insulin',hue='Outcome',data=data)
plt.show()
```




```
data.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
Pregnancies	1.000000	0.127964	0.208984	0.032568	-0.018082	0.021546	-0.033523	0.544341	0.2
Glucose	0.127964	1.000000	0.219666	0.172361	0.396597	0.231478	0.137106	0.266600	0.4
BloodPressure	0.208984	0.219666	1.000000	0.152458	0.010926	0.281231	0.000371	0.326740	0.1
SkinThickness	0.032568	0.172361	0.152458	1.000000	0.217854	0.546958	0.142977	0.054514	0.1
Insulin	-0.018082	0.396597	0.010926	0.217854	1.000000	0.189856	0.157806	0.038652	0.1
BMI	0.021546	0.231478	0.281231	0.546958	0.189856	1.000000	0.153508	0.025748	0.3
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.142977	0.157806	0.153508	1.000000	0.033561	0.1
Age	0.544341	0.266600	0.326740	0.054514	0.038652	0.025748	0.033561	1.000000	0.2
Outcome	0.221898	0.492908	0.162986	0.189065	0.179185	0.312254	0.173844	0.238356	1.0

```
sns.heatmap(data.corr(),annot=True)
plt.show()
```

```
# Data Modeling:
# 1. Devise strategies for model building. It is important to decide the right validation framework.
#Express your thought process.
# 2. Apply an appropriate classification algorithm to build a model.
#Compare various models with the results from KNN algorithm.
```

```
#Seperate data as features and label
features = data.iloc[:,:-1].values
label = data.iloc[:,-1].values
```

```
features
```

```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

```
features.ndim
```

```
2
```

```
label.ndim
```

```
1
```

```
label
```

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

```
#Create train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=4)
```

```
X_train.shape
```

```
(614, 8)
```

```
X_test.shape
```

```
(154, 8)
```

```
# Model 1 - Logistic Regression
```

```
# apply logistic regression
from sklearn.linear_model import LogisticRegression
model1=LogisticRegression()
model1.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
print(model1.score(X_train,y_train))
print(model1.score(X_test,y_test))
```

```
0.745928338762215
0.7597402597402597
```

```
y_pred = model1.predict(X_test)
```

```
y_pred
```

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

```
# evaluate the model
from sklearn.metrics import accuracy_score, classification_report, class_likelihood_ratios
```

```
print (accuracy_score(y_test,y_pred))
```

```
0.7597402597402597
```

```
print (classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.86      0.83       102
           1       0.67      0.56      0.61        52

    accuracy                           0.76       154
   macro avg       0.73      0.71      0.72       154
weighted avg       0.75      0.76      0.75       154
```

```
# Testing for Generalization

from sklearn.model_selection import train_test_split
for i in range(1,101):
  X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=i)
  model1 = LogisticRegression()
  model1.fit(X_train,y_train)
  trainScore = model1.score(X_train,y_train)
  testScore = model1.score(X_test,y_test)

  if testScore > trainScore:
    print("Test {} Train {} RS {}".format(testScore,trainScore,i))
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Test 0.7792207792207793 Train 0.7703583061889251 RS 90
Test 0.7857142857142857 Train 0.7719869706840391 RS 91
Test 0.8116883116883117 Train 0.758957654723127 RS 93
Test 0.8181818181818182 Train 0.7638436482084691 RS 95
Test 0.7857142857142857 Train 0.7785016286644951 RS 97
Test 0.7987012987012987 Train 0.7752442996742671 RS 98
Test 0.7792207792207793 Train 0.7654723127035831 RS 99
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

```
# Perform ROC Curve & AUC score

from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import f1_score,precision_score,recall_score
```

```
# calculate AUC score

auc=roc_auc_score(y_test,y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.5172800298897814
```

```
# Hyper parameter tuning for model1-LR
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
```

```
    'penalty': ['l1', 'l2']
}
```

```
# Creating a logistic regression model
logistic_regression = LogisticRegression(solver='liblinear')
```

```
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='roc_auc')
```

```
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

```
best_params = grid_search.best_params_
```

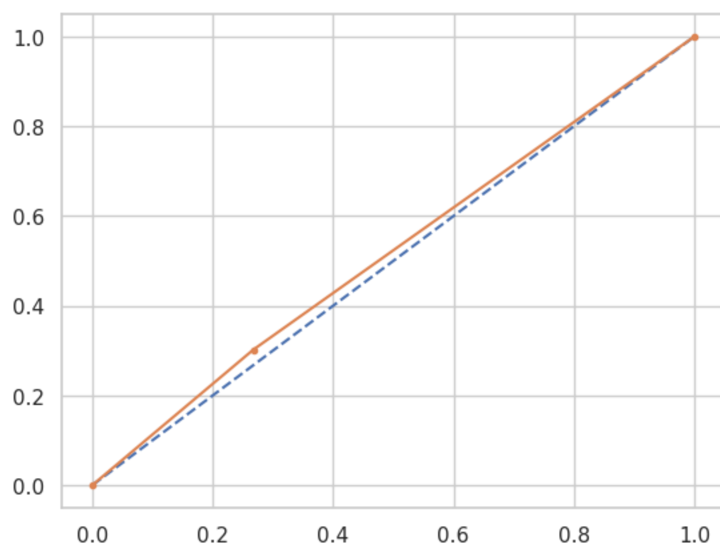
```
best_auc = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best AUC Score:", best_auc)
```

```
Best Hyperparameters: {'C': 10, 'penalty': 'l1'}
Best AUC Score: 0.8517611127465411
```

```
# calculate the fpr, tpr, thresholds
fpr,tpr,thre=roc_curve(y_test,y_pred)
```

```
# plot the curve
plt.plot([0,1],[0,1],linestyle='--')
plt.plot(fpr,tpr,marker='.')
plt.show()
```




```
# Creating a logistic regression model with the best hyperparameters
best_C = 10
best_penalty = 'l1'

best_log_reg_model = LogisticRegression(
    C=best_C,
    penalty=best_penalty,
    solver='liblinear',
    random_state=93
)
```

```
# Train the model on training data
best_log_reg_model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(C=10, penalty='l1', random_state=93, solver='liblinear')
```

```
y_pred = best_log_reg_model.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       101
           1       0.64      0.51      0.57        53

    accuracy                           0.73       154
   macro avg       0.71      0.68      0.69       154
weighted avg       0.72      0.73      0.73       154
```

```
# calculate AUC score

auc = roc_auc_score(y_test, y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.6804595553895012
```

```
# After hyperparameter tuning, Logistic regression model1,
# AUC score=0.68
# Test Accuracy= 73%
```

```
# Model 2 -DT Classifier
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
model2=DecisionTreeClassifier(max_depth=4)
```

```
model2.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4)
```

```
print(model2.score(X_train,y_train))
print(model2.score(X_test,y_test))
```

```
0.8078175895765473
0.7012987012987013
```

```
y_pred=model2.predict(X_test)
```

```
print(accuracy_score(y_test,y_pred))
```

```
0.7012987012987013
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.74      0.77       101
           1       0.56      0.62      0.59        53

    accuracy                           0.70       154
   macro avg       0.67      0.68      0.68       154
weighted avg       0.71      0.70      0.70       154
```

```
# Testing for Generalization

from sklearn.model_selection import train_test_split
for i in range(1,101):
  X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=i)
  model2 = DecisionTreeClassifier(max_depth=4)
  model2.fit(X_train,y_train)
  trainScore = model2.score(X_train,y_train)
  testScore = model2.score(X_test,y_test)

  if testScore > trainScore:
    print("Test {} Train {} RS {}".format(testScore,trainScore,i))
```

```
Test 0.7857142857142857 Train 0.7833876221498371 RS 74
```

```
# ROC Curve & AUC score

from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import f1_score,precision_score,recall_score
```

```
# AUC score before hyper parameter tuning

auc=roc_auc_score(y_test,y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.6826078834298525
```

```
# Hyper parameter tuning for Model2-DTClassifier
```

```
# Creating a DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(random_state=42)

param_grid = {
    'max_depth': [4, 6, 8, 10],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8]
}
```

```
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='roc_auc', verbose=1)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
best_params = grid_search.best_params_
best_auc = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best AUC Score:", best_auc)
```

```
Best Hyperparameters: {'max_depth': 6, 'min_samples_leaf': 8, 'min_samples_split': 20}
Best AUC Score: 0.8098182219605533
```

```
#Implementing hyper parameters for better DTclassifier
best_max_depth = 6
best_min_samples_leaf = 8
best_min_samples_split = 20

best_dt_classifier = DecisionTreeClassifier(
    max_depth=best_max_depth,
    min_samples_leaf=best_min_samples_leaf,
    min_samples_split=best_min_samples_split,
    random_state=74
)
```

```
best_dt_classifier.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, min_samples_split=20,
                       random_state=74)
```

```
y_pred = best_dt_classifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       101
           1       0.67      0.53      0.59        53

    accuracy                           0.75       154
   macro avg       0.72      0.69      0.70       154
weighted avg       0.74      0.75      0.74       154
```

```
# calculate AUC score

auc=roc_auc_score(y_test,y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.694844012703157
```

```
# After hyper parameter tuning , DTClassifier yields a model with
# Test accuracy=75%
# AUC score = 0.69
```

```
#Model3- Random Forest Classifier
```

```
from sklearn.ensemble import RandomForestClassifier
model3=RandomForestClassifier()
```

```
model3.fit(X_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_pred=model3.predict(X_test)
```

```
print(accuracy_score(y_test,y_pred))
```

```
0.7402597402597403
```

```
# Testing for Generalization

from sklearn.model_selection import train_test_split
for i in range(1,400):
  X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=i)
  model3 = RandomForestClassifier(max_depth=4)
  model3.fit(X_train,y_train)
  trainScore = model3.score(X_train,y_train)
  testScore = model3.score(X_test,y_test)

  if testScore > trainScore:
    print("Test {} Train {} RS {}".format(testScore,trainScore,i))
```

```
Test 0.8246753246753247 Train 0.8127035830618893 RS 57
Test 0.8181818181818182 Train 0.8061889250814332 RS 74
Test 0.8116883116883117 Train 0.8078175895765473 RS 76
Test 0.8181818181818182 Train 0.7931596091205212 RS 82
Test 0.8506493506493507 Train 0.8127035830618893 RS 121
Test 0.8246753246753247 Train 0.8127035830618893 RS 142
Test 0.8116883116883117 Train 0.8094462540716613 RS 146
```

```
Test 0.8311688311688312 Train 0.8078175895765473 RS 194
Test 0.8181818181818182 Train 0.8159609120521173 RS 195
Test 0.8311688311688312 Train 0.8127035830618893 RS 225
Test 0.8506493506493507 Train 0.8061889250814332 RS 307
Test 0.8506493506493507 Train 0.8078175895765473 RS 320
Test 0.8506493506493507 Train 0.8078175895765473 RS 345
Test 0.8116883116883117 Train 0.8045602605863192 RS 387
Test 0.8246753246753247 Train 0.8127035830618893 RS 389
Test 0.8246753246753247 Train 0.8224755700325733 RS 395
```

```
# Perform ROC Curve & AUC score
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.71      0.67        99
           1       0.36      0.29      0.32        55

    accuracy                           0.56       154
   macro avg       0.50      0.50      0.50       154
weighted avg       0.54      0.56      0.55       154
```

```
# calculate AUC score
auc=roc_auc_score(y_test,y_pred)
print('AUC Score',auc)
```

```
AUC Score 0.498989898989899
```

```
print(model3.score(X_train,y_train))
print(model3.score(X_test,y_test))
```

```
0.8159609120521173
0.7467532467532467
```

```
# HyperParameter tuning for RFC
```

```
from sklearn.model_selection import RandomizedSearchCV
param_dist = {
    'n_estimators': np.arange(100, 1000, 100),
    'max_depth': [None] + list(np.arange(10, 110, 10)),
    'min_samples_split': np.arange(2, 11),
    'min_samples_leaf': np.arange(1, 11),
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy'],
    'class_weight': [None, 'balanced'],
}
```

```
random_search = RandomizedSearchCV(model3, param_distributions=param_dist,
                                   n_iter=100, cv=5, scoring='accuracy', random_state=42, n_jobs=-1)

random_search.fit(X_train, y_train)

best_params = random_search.best_params_

best_accuracy = random_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score:", best_accuracy)
```

```
Best Hyperparameters: {'n_estimators': 100, 'min_samples_split': 8, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 40, 'cr
Best Accuracy Score: 0.7735705717712914
```

```
# implementing the best params for best RFC model

best_n_estimators = 900
best_min_samples_split = 10
best_min_samples_leaf = 6
best_max_features = 'sqrt'
best_max_depth = 90
best_criterion = 'entropy'
```

```
best_class_weight = 'balanced'
best_bootstrap = True

best_rf_classifier = RandomForestClassifier(
    n_estimators=best_n_estimators,
    min_samples_split=best_min_samples_split,
    min_samples_leaf=best_min_samples_leaf,
    max_features=best_max_features,
    max_depth=best_max_depth,
    criterion=best_criterion,
    class_weight=best_class_weight,
    bootstrap=best_bootstrap,
    random_state=121
)


best_rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=90, min_samples_leaf=6, min_samples_split=10,
                       n_estimators=900, random_state=121)
```

```
y_pred = best_rf_classifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.76      0.79        99
           1       0.62      0.71      0.66        55

    accuracy                           0.74       154
   macro avg       0.72      0.73      0.73       154
weighted avg       0.75      0.74      0.74       154
```

```
# calculate AUC score

auc = roc_auc_score(y_test, y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.7333333333333333
```

```
# After hyperparameter tuning, Logistic regression model1,
# AUC score=0.73
# Test Accuracy= 74%
```

```
# KNN classifier
```

```
from sklearn.neighbors import KNeighborsClassifier
model4=KNeighborsClassifier()
```

```
model4.fit(X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
print(model4.score(X_train,y_train))
print(model4.score(X_test,y_test))
```

```
0.8127035830618893
0.7207792207792207
```

```
y_pred= model4.predict(X_test)
```

```
print(accuracy_score(y_test,y_pred))
```

```
0.7207792207792207
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78        99
           1       0.60      0.64      0.62        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

```
# Testing for Generalization

from sklearn.model_selection import train_test_split
for i in range(1,401):
  X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=i)
  model4 = KNeighborsClassifier()
  model4.fit(X_train,y_train)
  trainScore = model4.score(X_train,y_train)
  testScore = model4.score(X_test,y_test)

  if testScore > trainScore:
    print("Test {} Train {} RS {}".format(testScore,trainScore,i))
```

```
Test 0.7662337662337663 Train 0.7638436482084691 RS 142
Test 0.7922077922077922 Train 0.7899022801302932 RS 194
Test 0.7922077922077922 Train 0.7785016286644951 RS 195
Test 0.7922077922077922 Train 0.7785016286644951 RS 225
Test 0.8051948051948052 Train 0.7931596091205212 RS 345
```

```
# ROC Curve & AUC score

from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import f1_score,precision_score,recall_score
```

```
# AUC score before hyper parameter tuning

auc=roc_auc_score(y_test,y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.40992946283233855
```

```
# Hyper parameter tuning for Model4-KNNClassifier
```

```
knn_classifier = KNeighborsClassifier()

param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
```

```
grid_search = GridSearchCV(knn_classifier, param_grid, cv=5, scoring='accuracy')
```

```
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

```
best_params = grid_search.best_params_
```

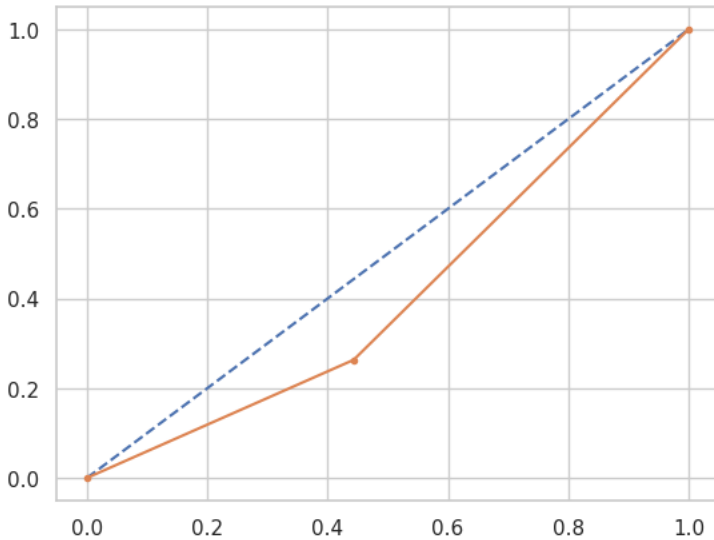
```
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score:", best_accuracy)
```

```
Best Hyperparameters: {'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
Best Accuracy Score: 0.7165000666400105
```

```
# calculate the fpr, tpr, thresholds
fpr,tpr,thre=roc_curve(y_test,y_pred)
```

```
# plot the curve
plt.plot([0,1],[0,1],linestyle='--')
plt.plot(fpr,tpr,marker='.')
plt.show()
```




```
# Creating a knn model with the best hyperparameters
best_n_neighbors = 9
best_weights = 'uniform'
best_p = 1

best_knn_classifier = KNeighborsClassifier(
    n_neighbors=best_n_neighbors,
    weights=best_weights,
    p=best_p
)
```

```
best_knn_classifier.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=9, p=1)
```

```
y_pred = best_knn_classifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.78        97
           1       0.64      0.53      0.58        57

    accuracy                           0.71       154
   macro avg       0.69      0.68      0.68       154
weighted avg       0.71      0.71      0.71       154
```

```
# calculate AUC score

auc = roc_auc_score(y_test, y_pred)

print('AUC Score',auc)
```

```
AUC Score 0.6755290287574606
```

```
# After hyperparameter tuning, KNNClassifier,
# AUC score=0.67
# Test Accuracy= 71%
```

```
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

X_train, X_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)

svc_classifier = SVC()
naive_bayes_classifier = GaussianNB()
```

```
classifiers = [
    ("Support Vector Machine", svc_classifier),
    ("Naive Bayes", naive_bayes_classifier),
]

# evaluating each classifier
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    print(f"Classifier: {clf_name}")
    print(f"Accuracy: {accuracy:.2f}")
    print("Classification Report:")
    print(report)
    print("\n")
```

```
Classifier: Support Vector Machine
Accuracy: 0.77
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83        99
           1       0.72      0.56      0.63        55

    accuracy                           0.77       154
   macro avg       0.75      0.72      0.73       154
weighted avg       0.76      0.77      0.76       154



Classifier: Naive Bayes
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.79      0.80        99
           1       0.64      0.69      0.67        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.74       154
weighted avg       0.76      0.75      0.76       154
```

```
#downloading the cleaneddataset for tableau
data.to_csv('clean_healthcare_data.csv', index=False)
```

```
#Test App for Deployment
```

```
X_train,X_test,y_train,y_test = train_test_split(features, label, test_size=0.2, random_state = 1)
model = SVC()
model.fit(X_train,y_train)
```

```
▾ SVC
SVC()
```

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

```
0.760586319218241
```

```
model.score(X_test,y_test)
```

```
0.7922077922077922
```

```
numberofpregnancies= float(input("Enter the number of pregnancies: "))
Glucose = float(input("Enter the Plasma Glucose value: "))
Bloodpressure = float(input("Enter the Diastolic BP value: "))
Skinthickness = float(input("Enter the Triceps skinfold thickness: "))
Insulin = float(input("Enter the two hours Insulin value: "))
BMI = float(input("Enter the BMI value: "))
DiabetesPedigreeFunction= float(input("Enter the Diabetes Pedigree Function value: "))
Age=float(input("Enter the Age: "))

features = np.array([[numberofpregnancies,Glucose,Bloodpressure,Skinthickness,Insulin,BMI,DiabetesPedigreeFunction,Age]])

DiabetesPrediction = model.predict(features)


print("Disease Prediction is : $ ",DiabetesPrediction)
```

```
Enter the number of pregnancies: 5
Enter the Plasma Glucose value: 89
Enter the Diastolic BP value: 88
Enter the Triceps skinfold thickness: 32
Enter the two hours Insulin value: 78
Enter the BMI value: 22
Enter the Diabetes Pedigree Function value: 0.2
Enter the Age: 45
Disease Prediction is : $  [0]
```

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29	79.799479	26.6	0.351	31	0
2	8	183.0	64.0	23	79.799479	23.3	0.672	32	1
3	1	89.0	66.0	23	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35	168.000000	43.1	2.288	33	1