Project Name: AI Based Diabetes Prediction System

Project Code:203476

Problem Definition:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

Objective:

The main objective of an AI-based Diabetes Prediction System is to help detect and predict diabetes in individuals. This system leverages artificial intelligence and machine learning techniques to analyze various data sources, such as medical records, patient history, and potentially even genetic information, to make predictions about a person's risk of developing diabetes. Here are some specific objectives of such a system:

• Early Detection: Identify individuals at risk of developing diabetes before they exhibit symptoms or receive an official diagnosis. Early detection can lead to early intervention and better management of the disease.

• Risk Assessment: Evaluate an individual's risk factors for diabetes, including lifestyle factors (e.g., diet, exercise), genetic predisposition, and medical history, to provide personalized risk assessments.

• Preventive Measures: Suggest preventive measures and lifestyle changes to individuals at risk, such as dietary recommendations, exercise routines, and regular monitoring, to help them reduce their risk of diabetes.

• Treatment Planning: For individuals already diagnosed with diabetes, assist healthcare professionals in creating personalized treatment plans based on the patient's specific needs and medical history.

• Data Integration: Integrate and analyze various types of data, including patient records, medical imaging, and wearable device data, to provide a comprehensive view of the patient's health and diabetes risk.

• Patient Education: Educate patients about diabetes, its management, and the importance of adhering to recommended lifestyle changes and treatment plans.

• Healthcare Resource Allocation: Help healthcare providers allocate their resources more efficiently by identifying high-risk individuals who may require more frequent monitoring and intervention

• Research and Insights: Generate insights from large datasets to improve our understanding of diabetes, its risk factors, and potential avenues for prevention and treatment.

• Continuous Monitoring: Enable continuous monitoring of patients' health data and provide real-time alerts to healthcare providers if a patient's risk level changes significantly.

• Cost Reduction: Potentially reduce healthcare costs associated with diabetes by preventing or mitigating complications through early intervention and management.

Design:

To build AI based diabetics prediction model we need to install some packages and they are:

• NumPy

• Pandas

• Matplotlib

• Scikit Learn

Dataset is taken from https://www.kaggle.com/datasets/mathchi/diabetes-data-set.

Since the data in the dataset is the raw data, it needs to undergo the following stages:

• Data Collection:

Data has to be collected from various sources to form a dataset. The dataset should contain some medical details or features such as glucose levels, ages, pressure, BMI etc.

• Data Preprocessing or cleaning and Visualization:

Since the dataset is collected from various sources it may consist of duplicates, nullable and Irrelevant data. So, data needs to be cleaned, normalized, replacing the nullable values with standard values etc. Once the data is cleaned it can be used to prepare for training.

• Feature Selection:

We need to select relevant features like glucose levels, ages, pressure, BMI etc. that can cause diabetics.

Model Selection:

We are going to build the model using various algorithms like:

• Logistic Regression

Evaluation:

We will evaluate the model's performance using metrics like:

• Accuracy Score

• ROC AUC Curve

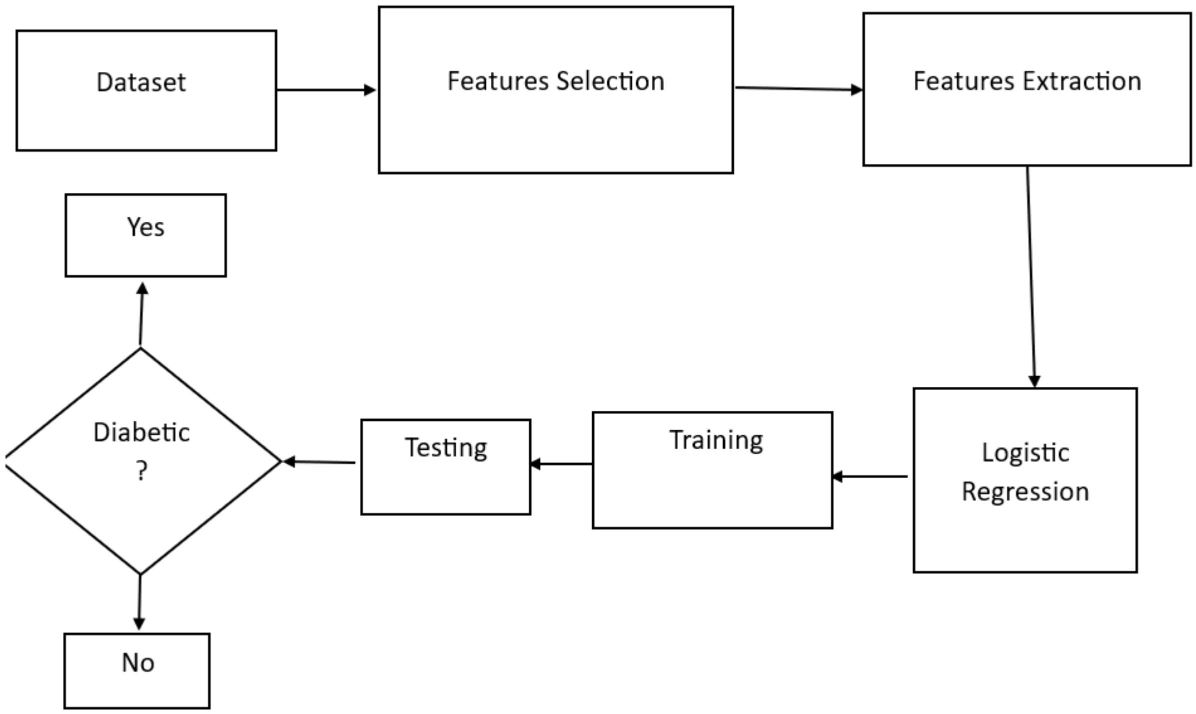
• Cross Validation

• Confusion Matrix

Iterative Improvement:

We will fine-tune the model parameters and explore techniques like feature engineering to enhance prediction accuracy.

Innovative Design:



Coding:

#Importing packages

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

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

#Reading the dataset

df=pd.read_csv('/content/diabetes (1).csv')

df.head ()

```python
df=pd.read_csv('/content/diabetes (1).csv')
df.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

df.describe ()

```python
df.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

df.info()

```python
[7] df.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
```

df.isnull ().values. any()

```python
df.isnull().values.any()
```

```
False
```

Print (df.isnull ().sum ())

```
print(df.isnull().sum())
```

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

df. Shape

```
[6] df.shape
```

```
(768, 9)
```

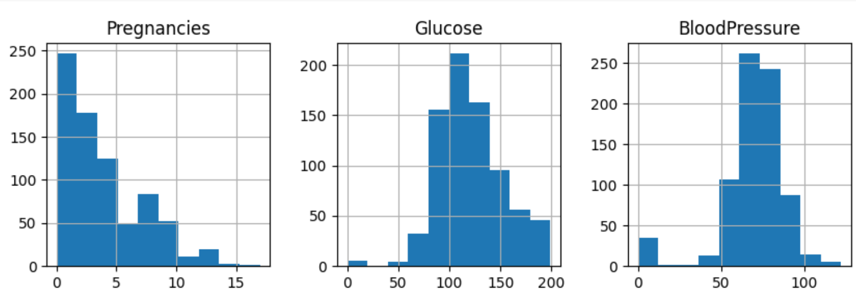
Various Analysis

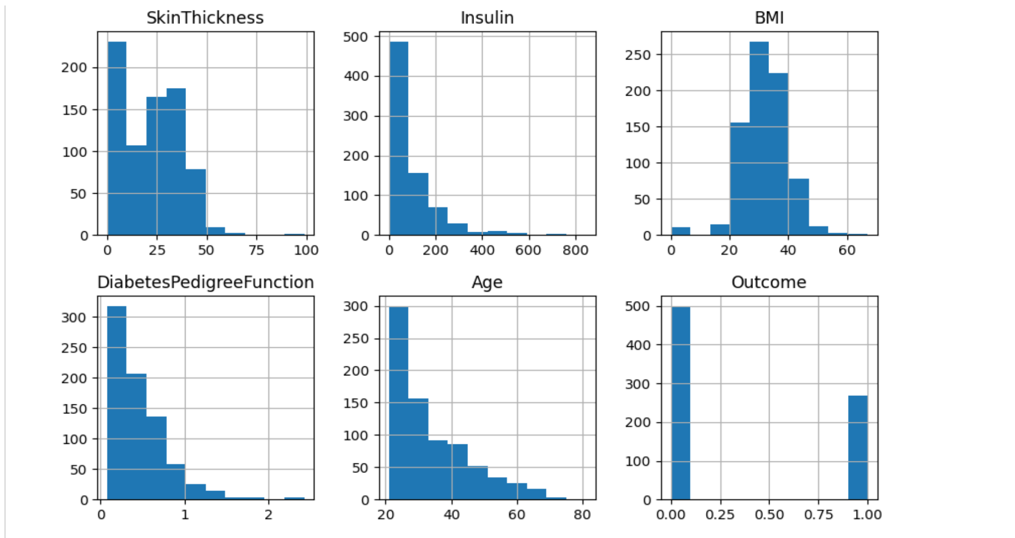
#Histogram

df.hist (bins=10, figsize= (10, 10))

plt.show ()

```
#histogram
df.hist(bins=10,figsize=(10,10))
plt.show()
```



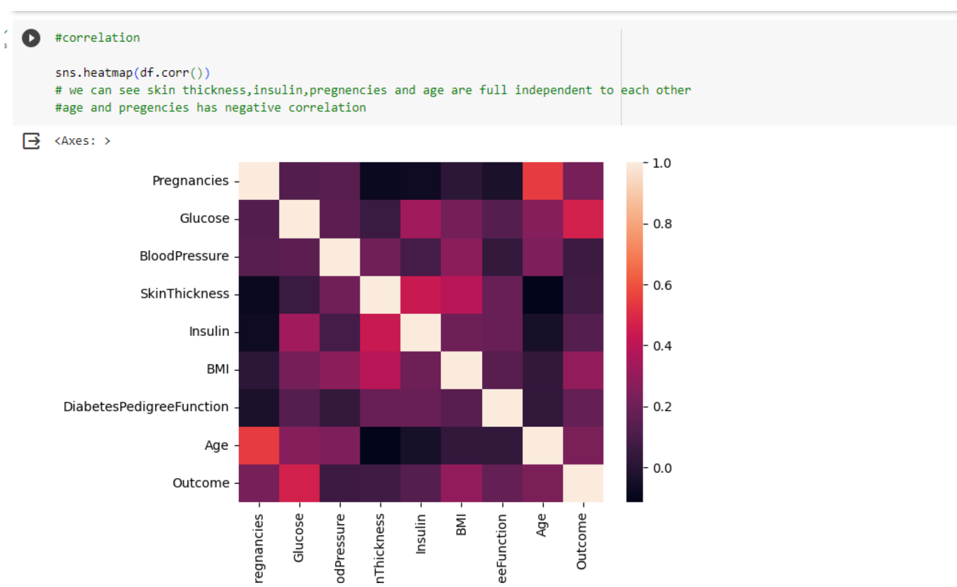


#correlation

sns.heatmap (df.corr ())

We can see skin thickness, insulin, pregnencies and age are full independent to each other

#age and pregencies has negative correlation

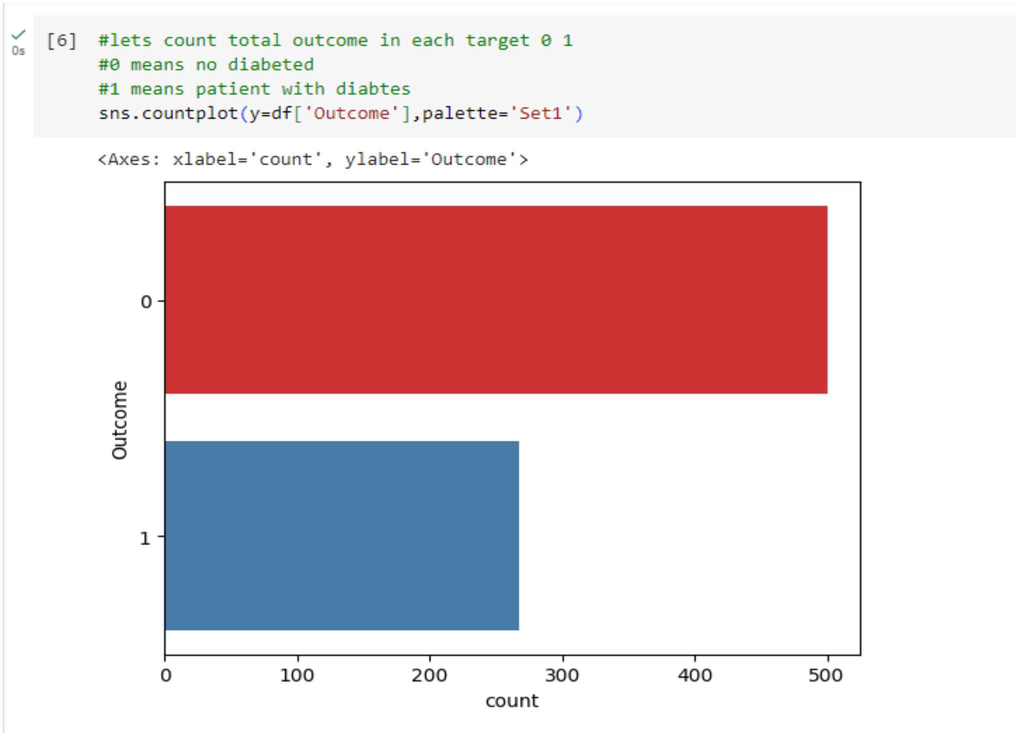


#lets count total outcome in each target 0 1

#0 means no diabeted

#1 means patient with diabetes

sns.countplot(y=df['Outcome'],palette='Set1')



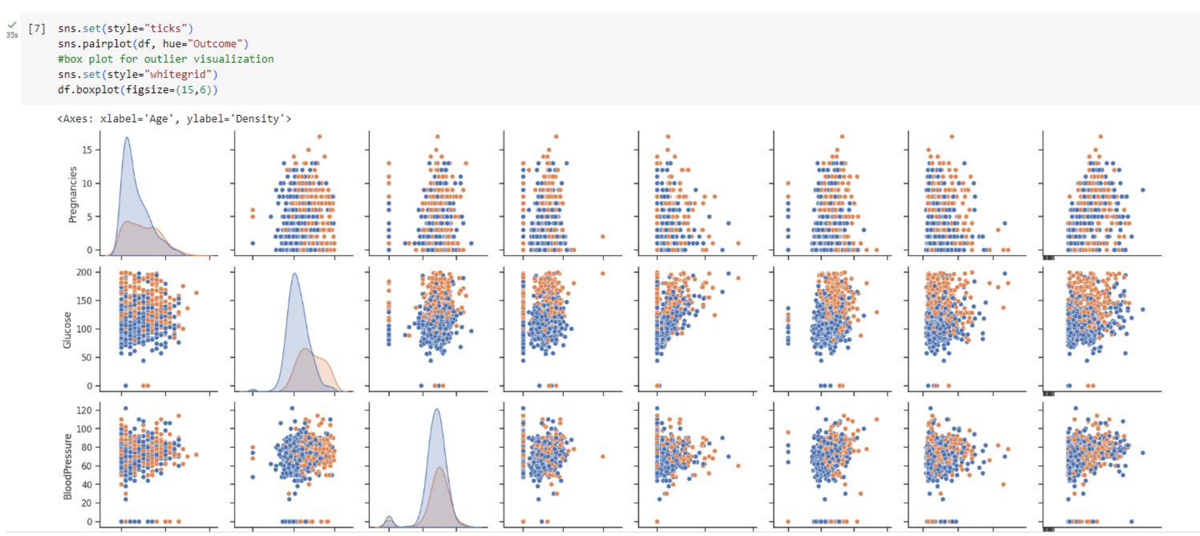
Sns. Set (style="ticks")

sns.pairplot(df, hue="Outcome")

#box plot for outlier visualization

Sns. Set (style="whitegrid")

df.boxplot(fig size=(15,6))



#box plot

Sns. Set (style="whitegrid")

Sns. Set (arc= {'figure.figsize':(4,2)})

sns.boxplot(x=df ['Insulin'])

plt.show()

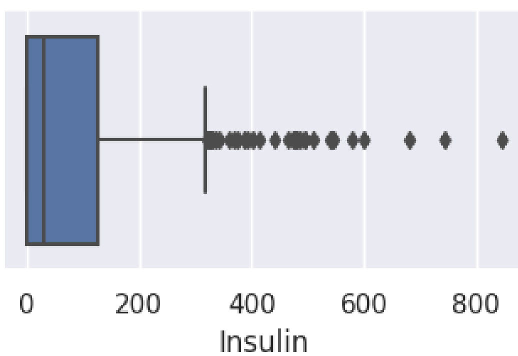
sns.boxplot(x=df['Blood Pressure'])

plt.show ()

sns.boxplot(x=df['DiabetesPedigreeFunction'])

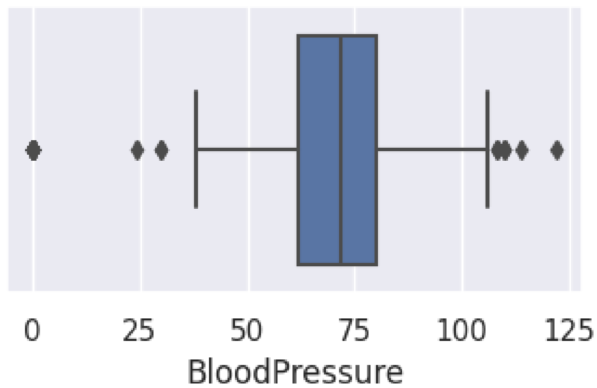
plt.show ()

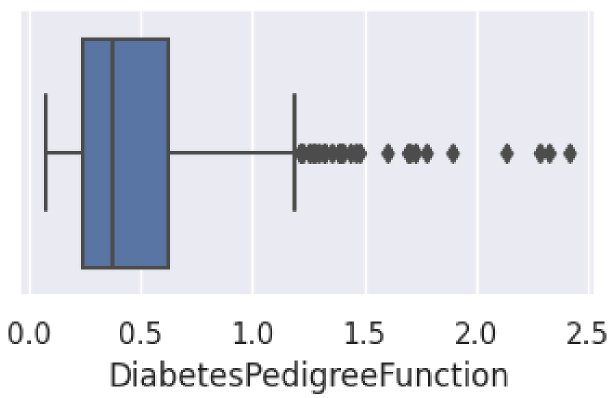
```
#box plot
sns.set(style="whitegrid")

sns.set(rc={'figure.figsize':(4,2)})
sns.boxplot(x=df['Insulin'])
plt.show()
sns.boxplot(x=df['BloodPressure'])
plt.show()
sns.boxplot(x=df['DiabetesPedigreeFunction'])
plt.show()
```



[8]

Insulin





[2]

```
df=pd.read_csv('/content/diabetes (1).csv')
df.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

```
Q1=df.quantile (0.25)

Q3=df.quantile (0.75)

IQR=Q3-Q1
```

print ("--#outlier remove -Q1--- \n", Q1)

print ("\n---Q3--- \n", Q3)

print("\n---IQR---\n",IQR)

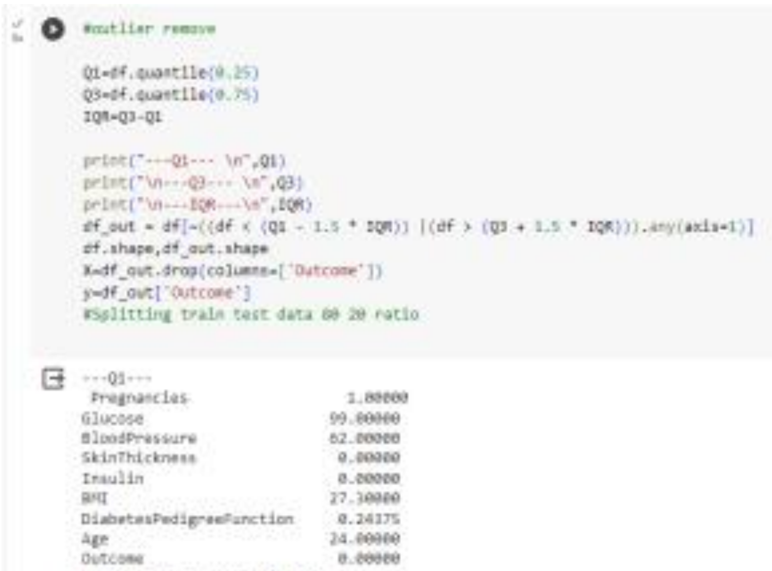
df_out = df [~ ((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]

df.shape,df_out.shape

X=df_out.drop(columns=['Outcome'])

y=df_out['Outcome']

#Splitting train test data 80 20 ratio



```
#outlier remove

Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
IQR=Q3-Q1

print("---Q1--- \n",Q1)
print("\n---Q3--- \n",Q3)
print("\n---IQR---\n",IQR)
df_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape,df_out.shape
X=df_out.drop(columns=['Outcome'])
y=df_out['Outcome']
#Splitting train test data 80 20 ratio
```

```
---Q1---
 Pregnancies                  1.00000
Glucose                      99.00000
BloodPressure                62.00000
SkinThickness                 0.00000
Insulin                       0.00000
BMI                          27.30000
DiabetesPedigreeFunction      0.24375
Age                          24.00000
Outcome                       0.00000
Name: 0.25, dtype: float64
```

```
Outcome                      0.00000
Name: 0.25, dtype: float64

---Q3---
 Pregnancies                   6.00000
Glucose                      140.25000
BloodPressure                 80.00000
SkinThickness                 32.00000
Insulin                      127.25000
BMI                           36.60000
DiabetesPedigreeFunction       0.62625
Age                           41.00000
Outcome                        1.00000
Name: 0.75, dtype: float64

---IQR---
 Pregnancies                   5.0000
Glucose                       41.2500
BloodPressure                 18.0000
SkinThickness                 32.0000
Insulin                      127.2500
BMI                            9.3000
DiabetesPedigreeFunction       0.3825
Age                           17.0000
Outcome                        1.0000
dtype: float64
```

from sklearn.model_selection import train_test_split

train_X,test_X,train_y,test_y=train_test_split(X,y,test_

size=0.)

train_X.shape,test_X.shape,train_y.shape,test_y.shape

```
[5] from sklearn.model_selection import train_test_split
    train_X,test_X,train_y,test_y=train_test_split(X,y,test_size=0.2)
    train_X.shape,test_X.shape,train_y.shape,test_y.shape

    ((511, 8), (128, 8), (511,), (128,))
```

from sklearn.metrics import

confusion_matrix,accuracy_score,make_scorer from

sklearn.model_selection import cross_validate

def tn(y_true, y_pred): return confusion_matrix(y_true,

y_pred)[0, 0] def fp(y_true, y_pred): return

confusion_matrix(y_true, y_pred)[0, 1] def fn(y_true, y_pred): return confusion_matrix(y_true, y_pred)[1, 0] def

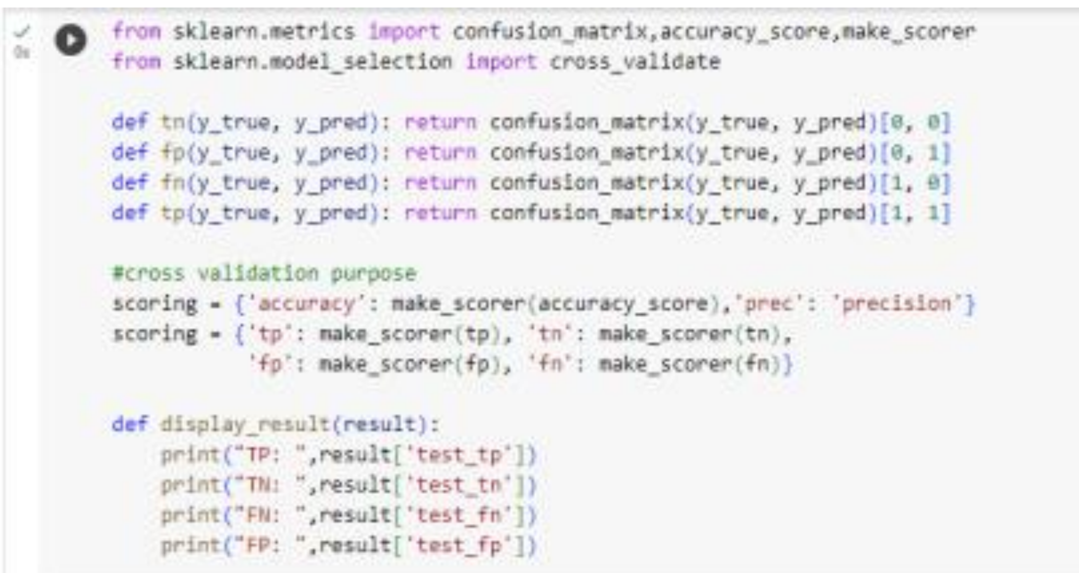
tp(y_true, y_pred): return confusion_matrix(y_true,

y_pred)[1, 1]

#cross validation purpose

scoring = {'accuracy':

make_scorer(accuracy_score),'prec':



```
from sklearn.metrics import confusion_matrix,accuracy_score,make_scorer
from sklearn.model_selection import cross_validate

def tn(y_true, y_pred): return confusion_matrix(y_true, y_pred)[0, 0]
def fp(y_true, y_pred): return confusion_matrix(y_true, y_pred)[0, 1]
def fn(y_true, y_pred): return confusion_matrix(y_true, y_pred)[1, 0]
def tp(y_true, y_pred): return confusion_matrix(y_true, y_pred)[1, 1]

#cross validation purpose
scoring = {'accuracy': make_scorer(accuracy_score),'prec': 'precision'}
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn)}

def display_result(result):
    print("TP: ",result['test_tp'])
    print("TN: ",result['test_tn'])
    print("FN: ",result['test_fn'])
    print("FP: ",result['test_fp'])
```

'Precision'} scoring = {'tp': make_scorer(tp), 'tn':

make_scorer(tn), 'fp': make_scorer(fp), 'fn': make_scorer

(fn)}

def display_result(result):

Print("TP: ", result['testate'])

Print ("TN: “ result['testate'])

Print ("FN: ", result['testify'])

```
print ("FP: ",result['precision'} scoring = {'tp':
make_scorer(tp), 'tn': make_scorer(tn),  'fp':
make_scorer(fp), 'fn': make_scorer(fn)}


def display_result(result):
 print("TP: ",result['test_tp'])
 print("TN: ",result['test_tn'])
 print("FN: ",result['test_fn'])
print("FP: ",result['test_fp'])'test_fp'])acc=[]
roc=[]


clf=LogisticRegression()
clf.fit(train_X,train_y)
y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

acc=[]
roc=[]

clf=LogisticRegression()
clf.fit(train_X,train_y)
y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
acc.append(ac)

#find the ROC_AOC
rc=roc_auc_score(test_y,y_pred)
roc.append(rc)
print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score
result=cross_validate(clf,train_X,train_y,scoring=scoring,cv=10)
display_result(result)
```

```
#find the ROC_AOC curve

rc=roc_auc_score(test_y,y_pred)

roc.append(rc)

print("\nAccuracy {0} ROC {1}".format(ac,rc))



#cross val score

result=cross_validate(clf,train_X,train_y,scoring=scori

ng,cv=10) display_result(result)
```


```
#Naive Bayes Theorem
#import library

from sklearn.naive_bayes import GaussianNB



clf=GaussianNB()

clf.fit(train_X,train_y)

y_pred=clf.predict(test_X)

#find accuracy

ac=accuracy_score(test_y,y_pred)

acc.append(ac)
```

```
#find the ROC_AOC curve

rc=roc_auc_score(test_y,y_pred)

roc.append(rc)

print("\nAccuracy {0} ROC {1}".format(ac,rc))


#cross val score

result=cross_validate(clf,train_X,train_y,scoring=scori

ng,cv=10) display_result(result)
```

```
#Naive Bayes Theorem
#import library
from sklearn.naive_bayes import GaussianNB

clf=GaussianNB()
clf.fit(train_X,train_y)
y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
acc.append(ac)

#find the ROC_AOC curve
rc=roc_auc_score(test_y,y_pred)
roc.append(rc)
print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score
result=cross_validate(clf,train_X,train_y,scoring=scoring,cv=10)
display_result(result)
```

```
Accuracy 0.796875 ROC 0.7819072313454336
TP:  [10 11  8 10  7  8 10 11  7 11]
TN:  [32 26 26 32 28 28 31 31 31 27]
FN:  [7 5 8 6 9 8 6 5 9 5]
FP:  [3 9 9 3 7 7 4 4 4 8]
```