Institute of Information Technology (IIT)

Jahangirnagar University

Lab Report: 06

Submitted by:

Name: Zannat Hossain Tamim

Roll No:1970

Lab Date:22.08.23

Submission Date: 28.08.23

Lab Report # Day 06

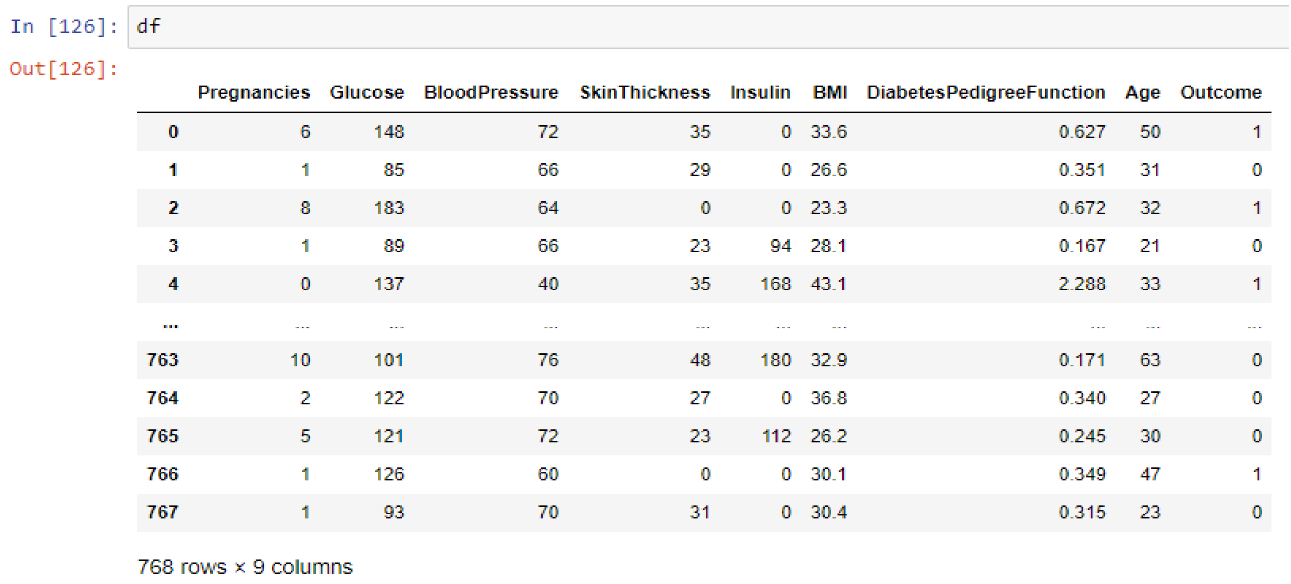
Query 1:

Import libraries , read CSV file, and print the file

Clause:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df= pd.read_csv('diabetes.csv')
df
```

Result :



```
In [126]: df
Out[126]:
```

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

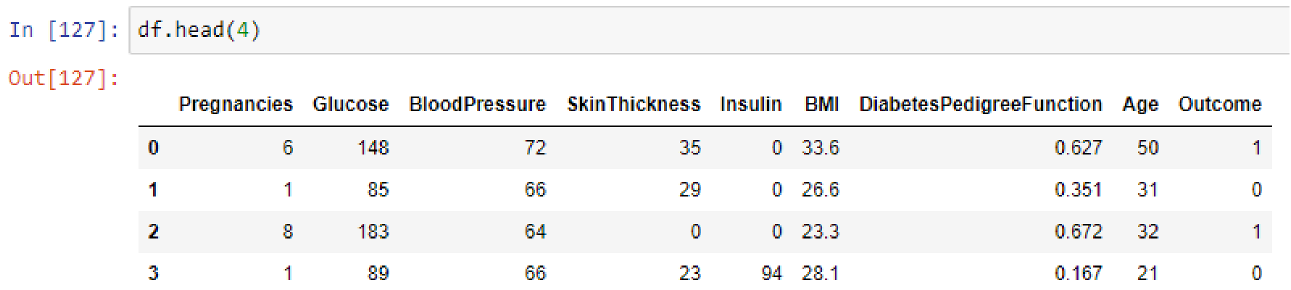
768 rows × 9 columns

Query 2: Print the top 4 rows.

Clause:

```
df.head(4)
```

Result :



```
In [127]: df.head(4)
Out[127]:
```

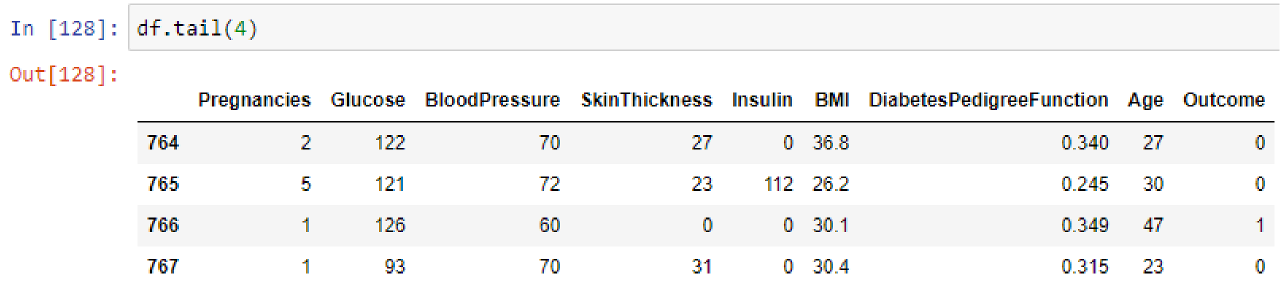
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

Query 3: Print the last 4 rows of the data frame.

Clause:

df.tail()

Result :



```
In [128]: df.tail(4)
```

Out[128]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

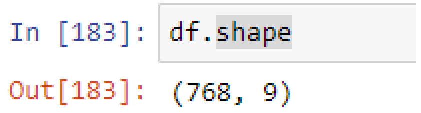
Query 4:

To get the shape of the DataFrame

Clause:

df.shape

Result :



```
In [183]: df.shape
Out[183]: (768, 9)
```

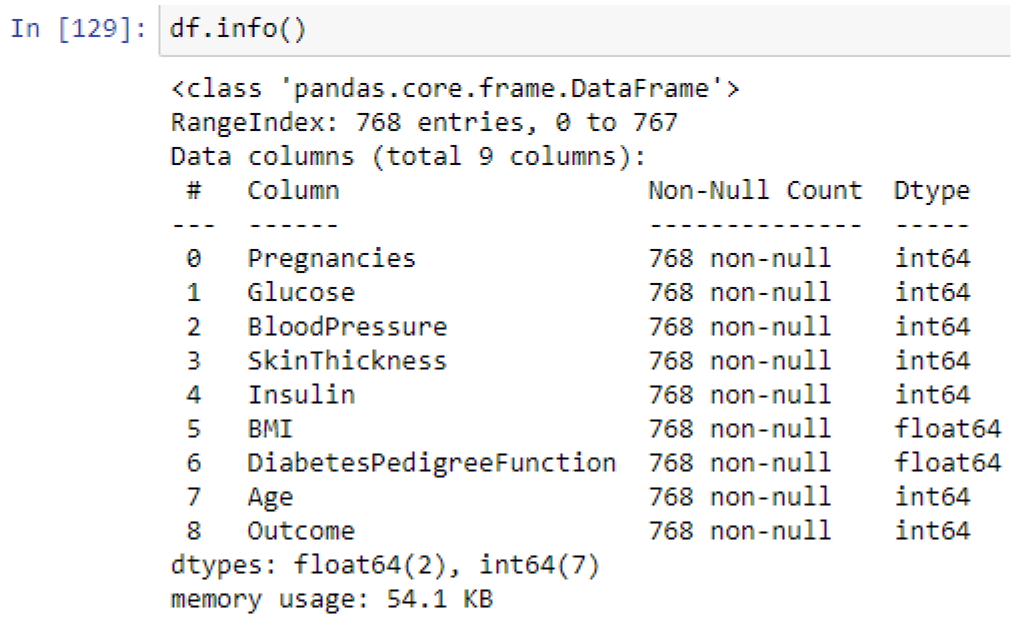
Query 5:

To get the information of the DataFrame

Clause:

df.info()

Result :



```
In [129]: 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
```

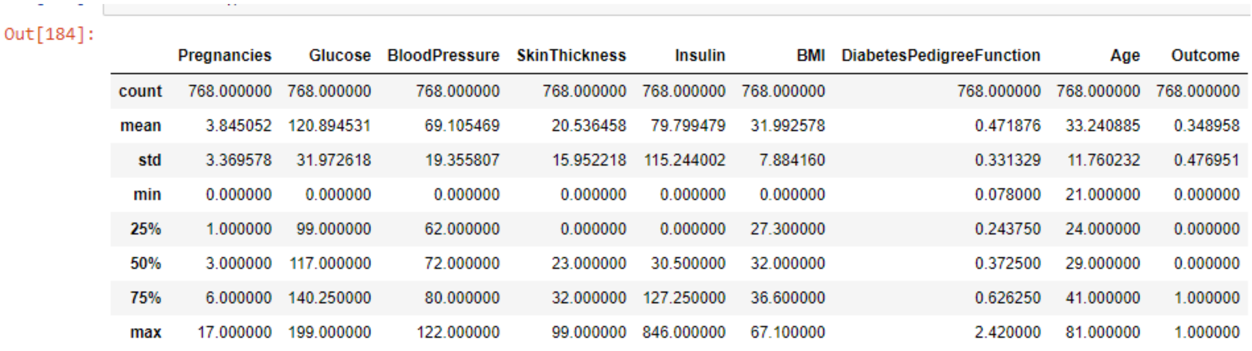
Query 6:

To calculate and print a summary of statistical data for each column in the DataFrame

Clause:

df.describe()

Result :



Out[184]:

	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

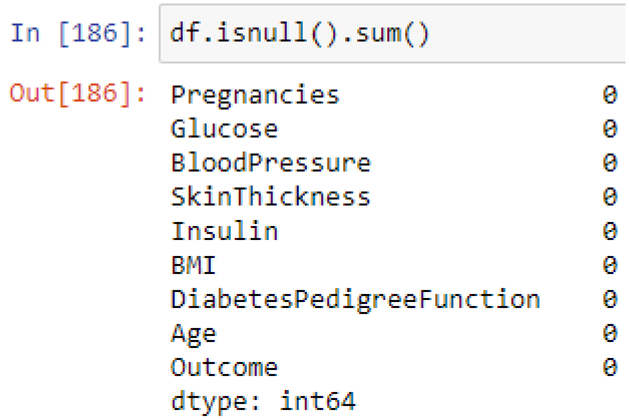
Query 7:

To count the number of missing values in each column of a DataFrame

Clause:

df.isnull().sum()

Result:



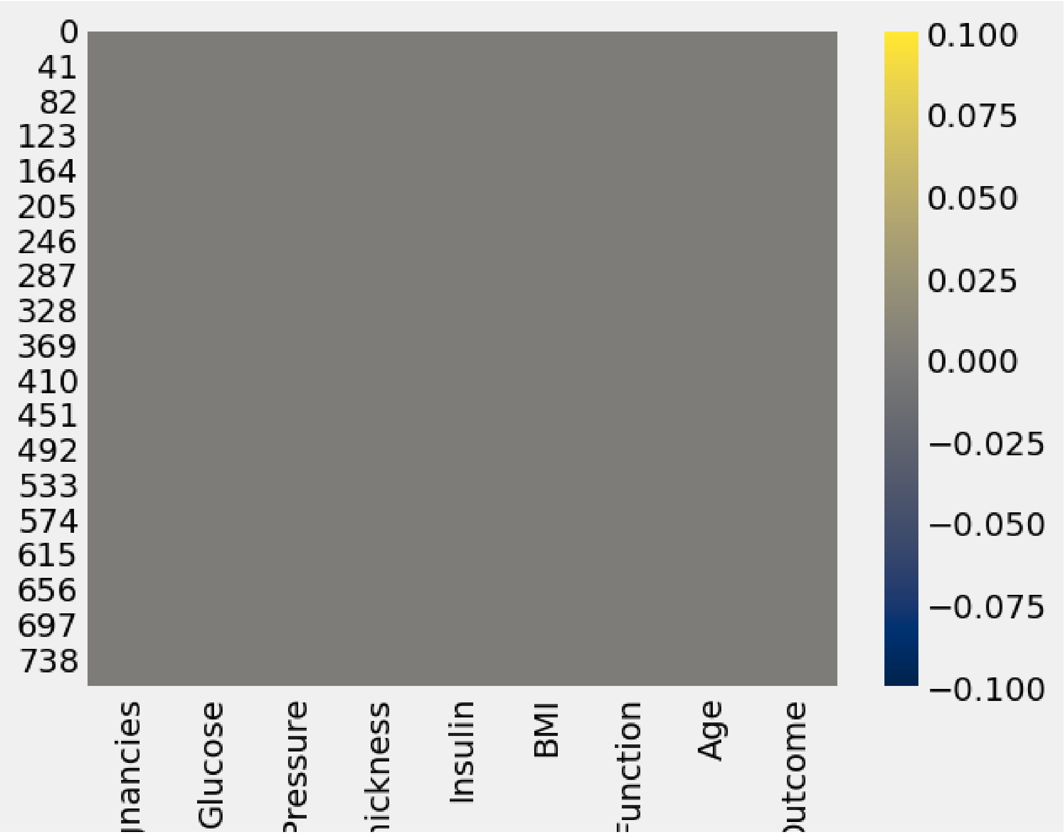
```
In [186]: df.isnull().sum()

Out[186]: Pregnancies                 0
          Glucose                     0
          BloodPressure               0
          SkinThickness               0
          Insulin                     0
          BMI                         0
          DiabetesPedigreeFunction    0
          Age                         0
          Outcome                     0
          dtype: int64
```

Query 8 : Create a heatmap of the missing values in the data frame.

Clause:

```
sns.heatmap(df.isnull(),cmap='cividis')
```

Result:

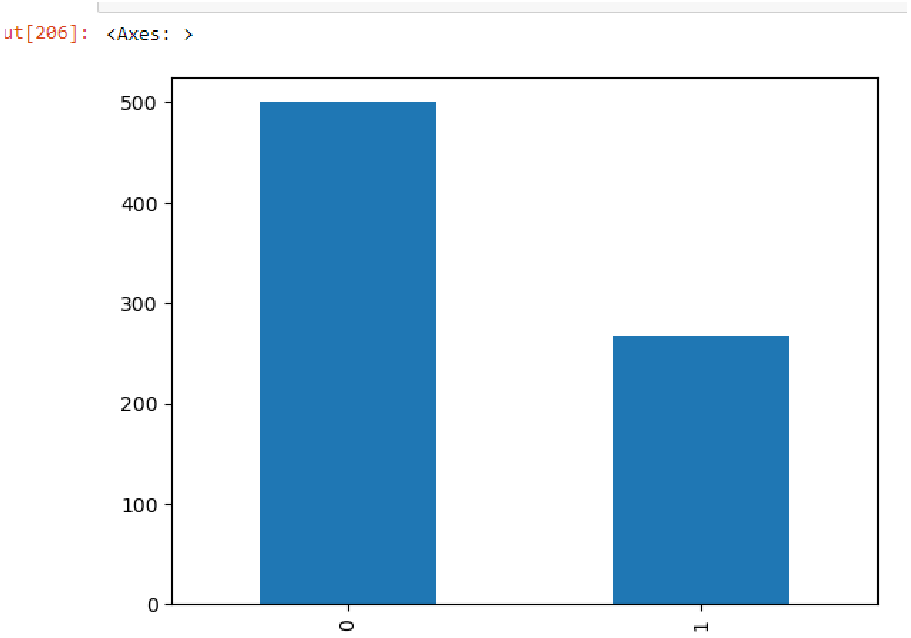


Query 9 :

Clause:

```
p=df.Outcome.value_counts()
p.plot(kind="bar")
```

Result:



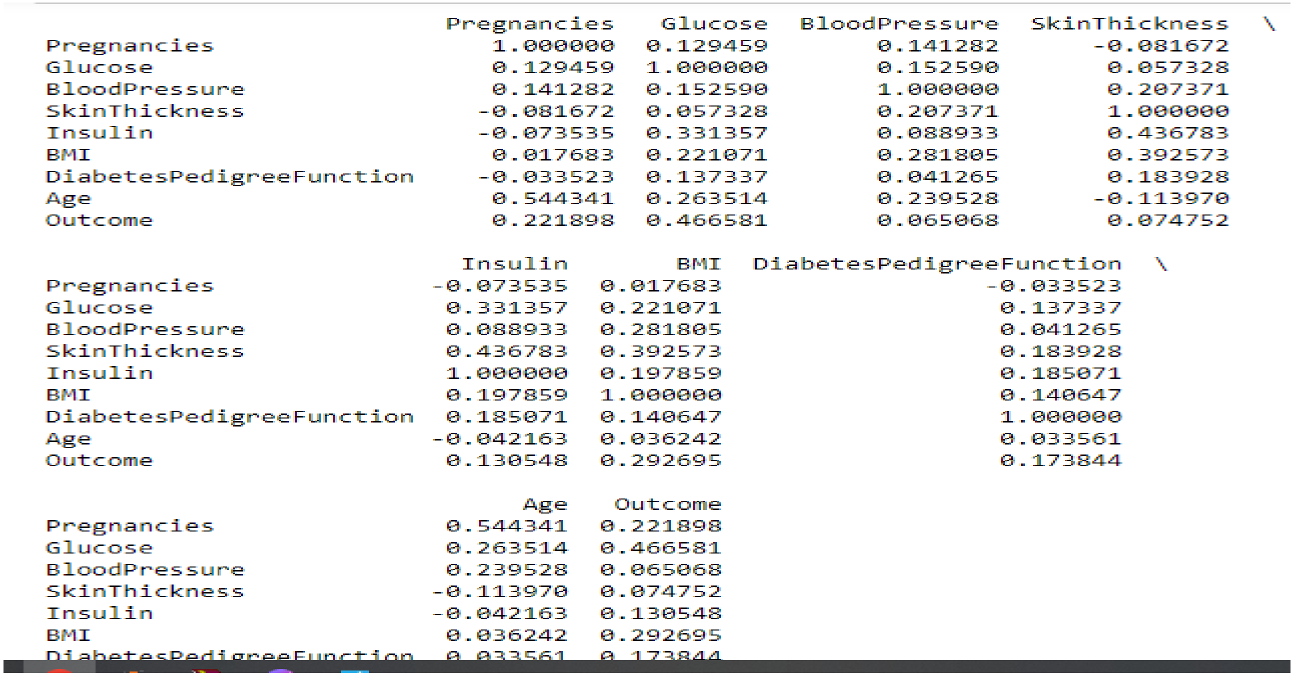
```
ut[206]: <Axes: >
```

Query 10: Calculate the correlation coefficient between each pair of columns in the data frame and store it in the variable cor

Clause:

```
cor=df.corr()
print(cor)
```

Result:



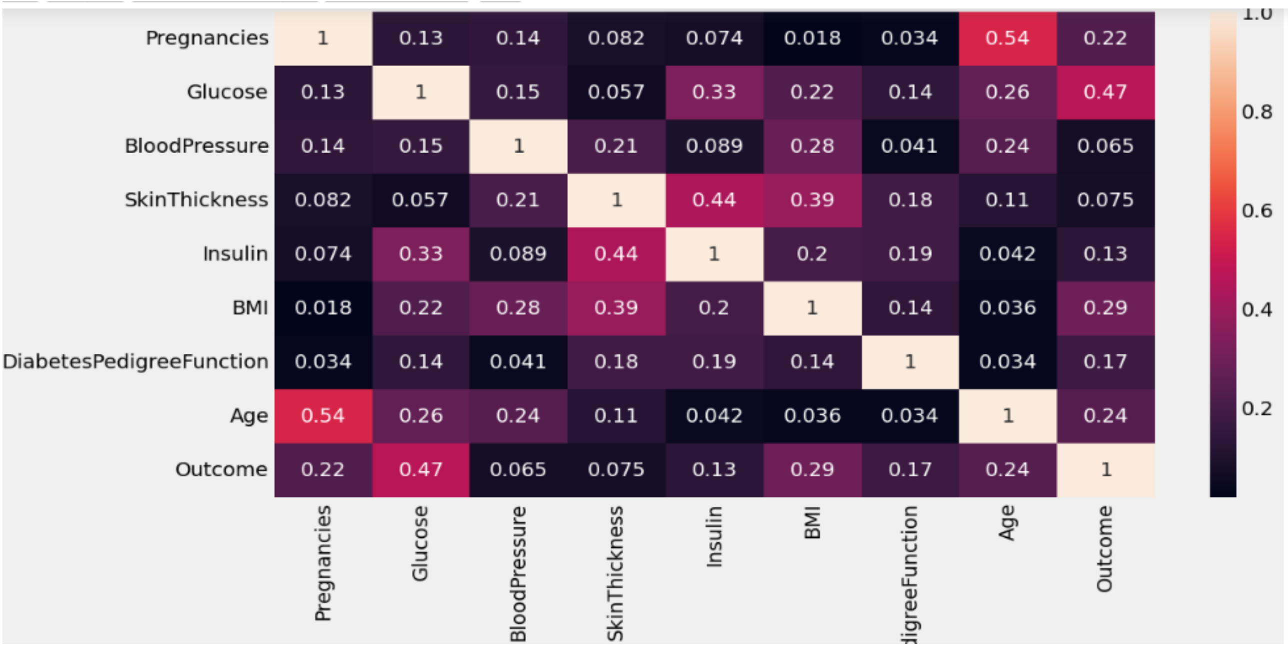
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.129459       0.141282      -0.081672
Glucose                      0.129459  1.000000       0.152590       0.057328
BloodPressure                0.141282  0.152590       1.000000       0.207371
SkinThickness               -0.081672  0.057328       0.207371       1.000000
Insulin                     -0.073535  0.331357       0.088933       0.436783
BMI                          0.017683  0.221071       0.281805       0.392573
DiabetesPedigreeFunction    -0.033523  0.137337       0.041265       0.183928
Age                          0.544341  0.263514       0.239528      -0.113970
Outcome                      0.221898  0.466581       0.065068       0.074752

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.073535  0.017683                 -0.033523
Glucose                   0.331357  0.221071                  0.137337
BloodPressure             0.088933  0.281805                  0.041265
SkinThickness             0.436783  0.392573                  0.183928
Insulin                   1.000000  0.197859                  0.185071
BMI                       0.197859  1.000000                  0.140647
DiabetesPedigreeFunction  0.185071  0.140647                  1.000000
Age                      -0.042163  0.036242                  0.033561
Outcome                   0.130548  0.292695                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
```

Query 11: To create a heatmap of the correlation matrix:

Clause:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.figure(figsize=(12,6))
sns.heatmap(df.corr().abs(),annot=True)
```

Result:



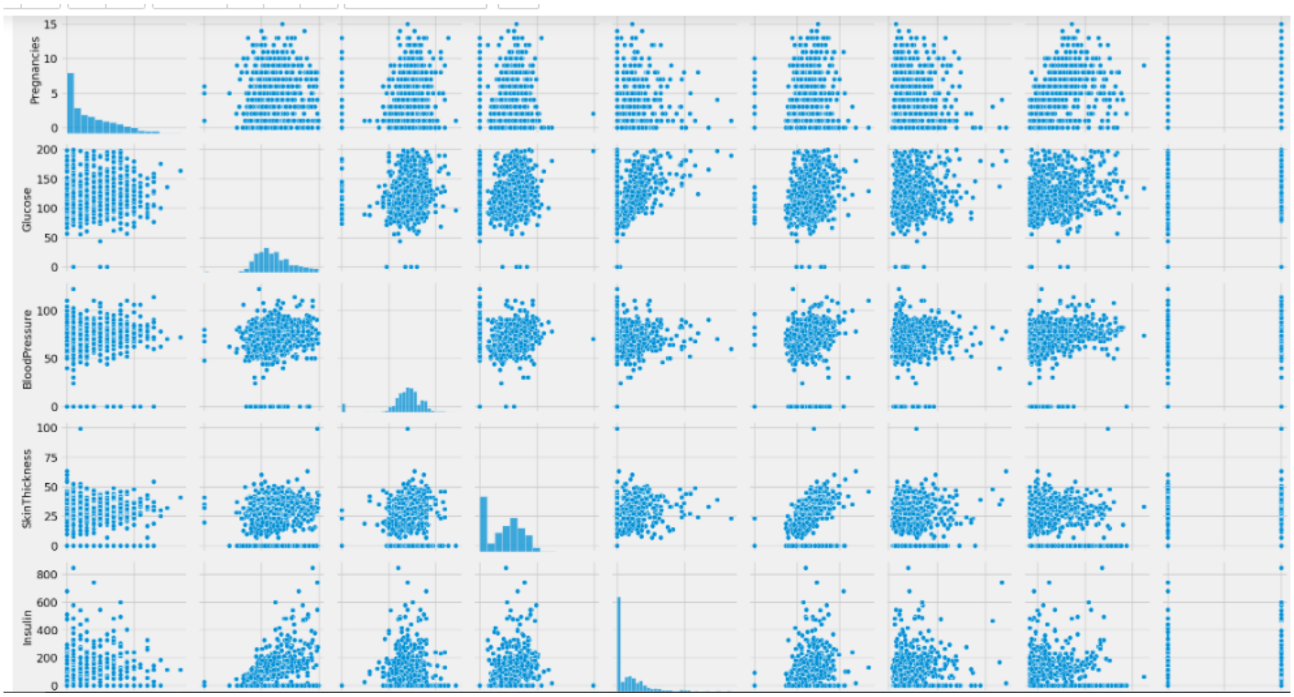
Query 12:

Plot pairwise relationships between variables of the dataset

Clause:

```
sns.pairplot(df)
```

Result:

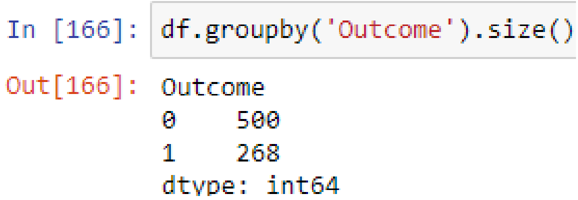


Query 13: Number of *Outcome*

Clause:

```
df.groupby('Outcome').size()
```

Result:



```
In [166]: df.groupby('Outcome').size()

Out[166]: Outcome
          0    500
          1    268
          dtype: int64
```

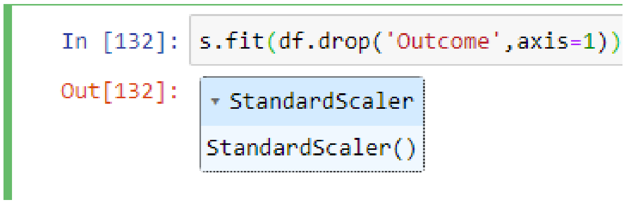
Query 14:

Standardize the variables

Clause:

```
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
s.fit(df.drop('Outcome',axis=1))
```

Result:



```
In [132]: s.fit(df.drop('Outcome',axis=1))
Out[132]: ▾ StandardScaler
          StandardScaler()
```

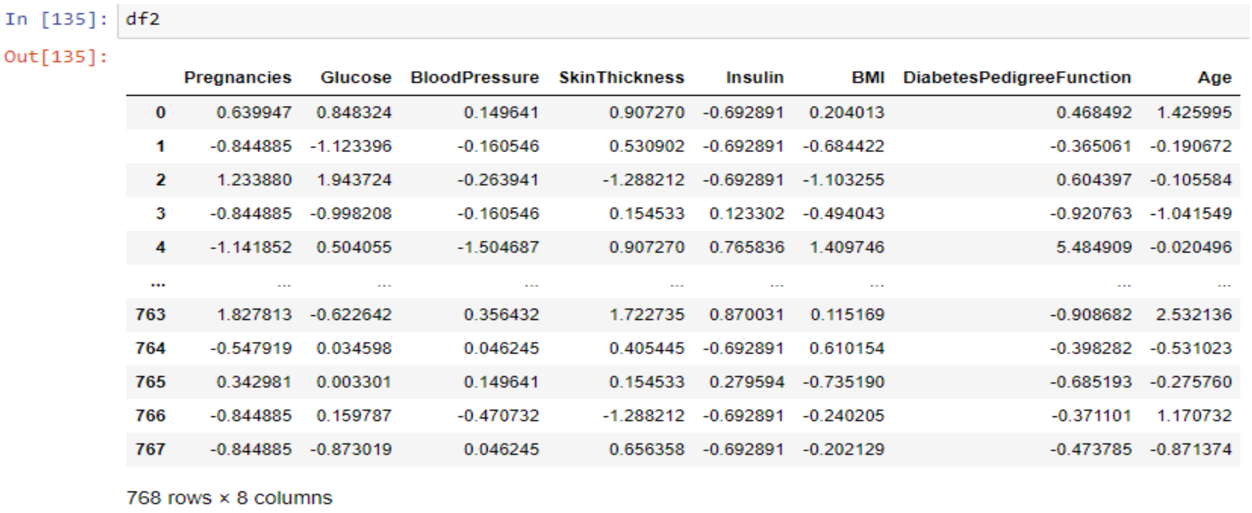
Query 15:

Standardize the variables

Clause:

sf= s.transform(df.drop('Outcome',axis=1))
df2= pd.DataFrame(sf,columns=df.columns[:-1])
df2

Result:



```
In [135]: df2
Out[135]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	0.468492	1.425995
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	-0.365061	-0.190672
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	0.604397	-0.105584
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	-0.920763	-1.041549
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	5.484909	-0.020496
...	...	...	...	...	...	...	...	...
763	1.827813	-0.622642	0.356432	1.722735	0.870031	0.115169	-0.908682	2.532136
764	-0.547919	0.034598	0.046245	0.405445	-0.692891	0.610154	-0.398282	-0.531023
765	0.342981	0.003301	0.149641	0.154533	0.279594	-0.735190	-0.685193	-0.275760
766	-0.844885	0.159787	-0.470732	-1.288212	-0.692891	-0.240205	-0.371101	1.170732
767	-0.844885	-0.873019	0.046245	0.656358	-0.692891	-0.202129	-0.473785	-0.871374

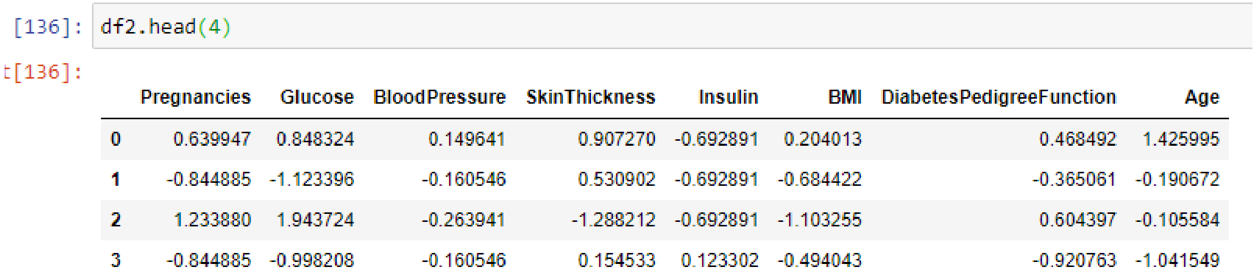
768 rows × 8 columns

Query 16:

Clause:

df2.head(4))

Result:



```
[136]: df2.head(4)
t[136]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	0.468492	1.425995
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	-0.365061	-0.190672
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	0.604397	-0.105584
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	-0.920763	-1.041549

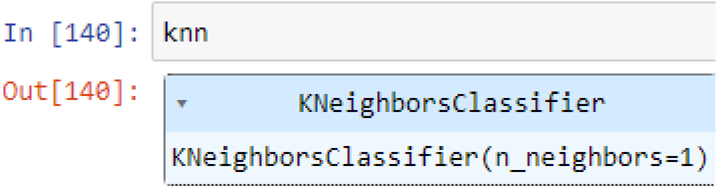
Query 17:

Train test split and using KNN

Clause:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.loc[:, df.columns != 'Outcome'],
                                          df['Outcome'], stratify=df['Outcome'], random_state=101)
from sklearn.neighbors import KNeighborsClassifier
 knn = KNeighborsClassifier(n_neighbors=1)
 knn
```

Result:



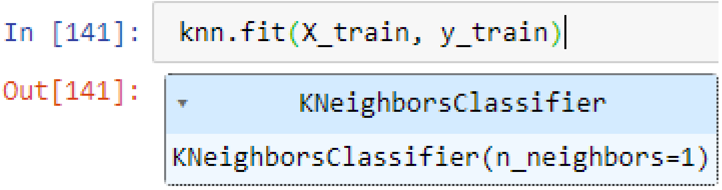
```
In [140]: knn
Out[140]:
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Query 18:

Clause:

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

Result:



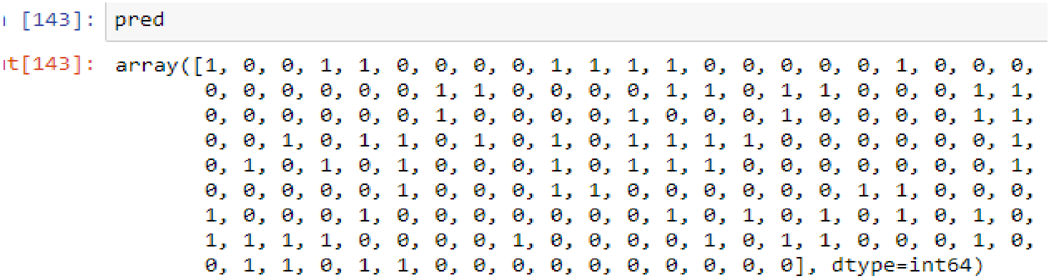
```
In [141]: knn.fit(X_train, y_train)
Out[141]:
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Query 19:

Clause:

```
pred=knn.predict(X_test)
pred
```

Result:



```
ı [143]: pred

ıt[143]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
                0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1,
                0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
                1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
                1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,
                0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Query 20:

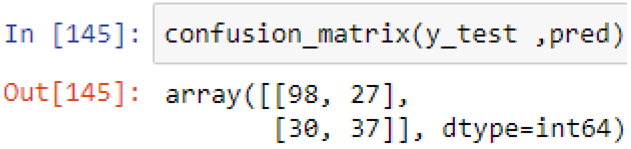
Prediction and evaluation

Clause:

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
confusion_matrix(y_test ,pred)
```

Result:



```
In [145]: confusion_matrix(y_test ,pred)

Out[145]: array([[98, 27],
                 [30, 37]], dtype=int64)
```

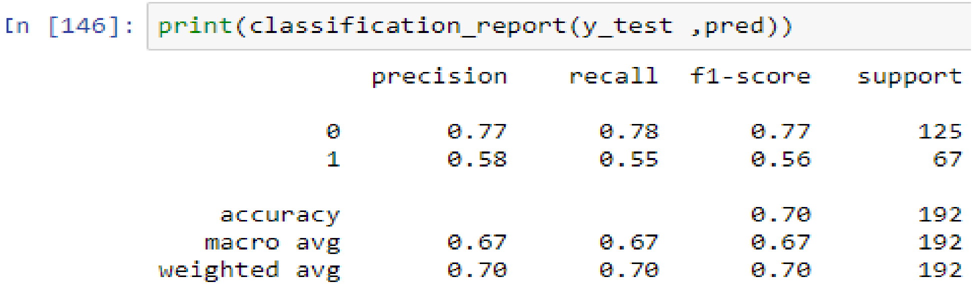
Query 21:

Prediction and evaluation

Clause:

```
print(classification_report(y_test ,pred))
```

Result:



```
In [146]: print(classification_report(y_test ,pred))
                  precision    recall  f1-score   support

               0       0.77      0.78      0.77       125
               1       0.58      0.55      0.56        67

        accuracy                           0.70       192
       macro avg       0.67      0.67      0.67       192
    weighted avg       0.70      0.70      0.70       192
```

Query 22:

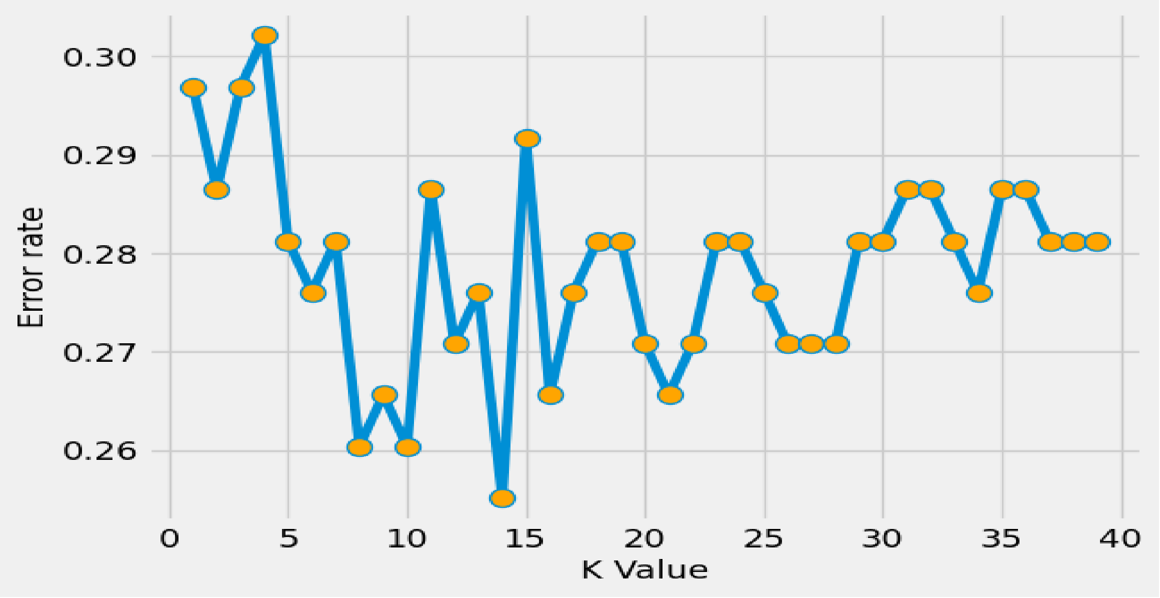
Find the error rate to choose a K value

Clause:

```
err=[]
for i in range(1,40):
        knn= KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train, y_train)
        pred_i=knn.predict(X_test)
        err.append(np.mean( pred_i!= y_test))

plt.plot(range(1,40),err,marker='o',markerfacecolor='orange',markersize=10)
plt.xlabel('K Value')
plt.ylabel('Error rate')
```

Result:

```
Out[148]: Text(0, 0.5, 'Error rate')
```



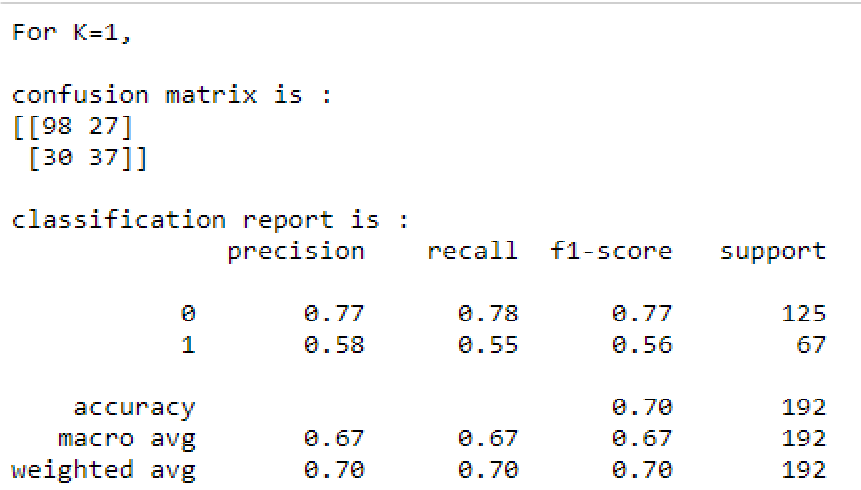
Query 23:

For K=1,

Clause:

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
pred2=knn.predict(X_test)
print('For K=1, \n')
print('confusion matrix is : ')
print(confusion_matrix(y_test ,pred),'\n')
print('classification report is : ')
print(classification_report(y_test ,pred))
```

Result:



```
For K=1,

confusion matrix is :
[[98 27]
 [30 37]]

classification report is :
              precision    recall  f1-score   support

           0       0.77      0.78      0.77       125
           1       0.58      0.55      0.56        67

    accuracy                           0.70       192
   macro avg       0.67      0.67      0.67       192
weighted avg       0.70      0.70      0.70       192
```

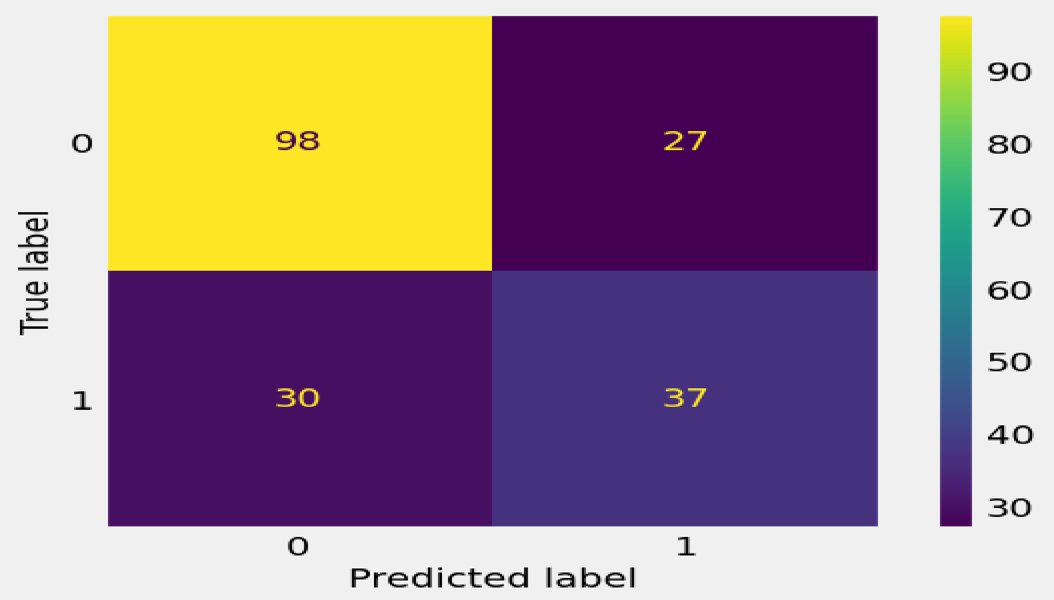
Query 24:

Confusion Matrix Display for K=1

Clause:

```
from sklearn.metrics import ConfusionMatrixDisplay
c=confusion_matrix(y_test ,pred2)
c2=ConfusionMatrixDisplay(c)
c2.plot()
plt.grid(False)
```

Result:



Query 25:

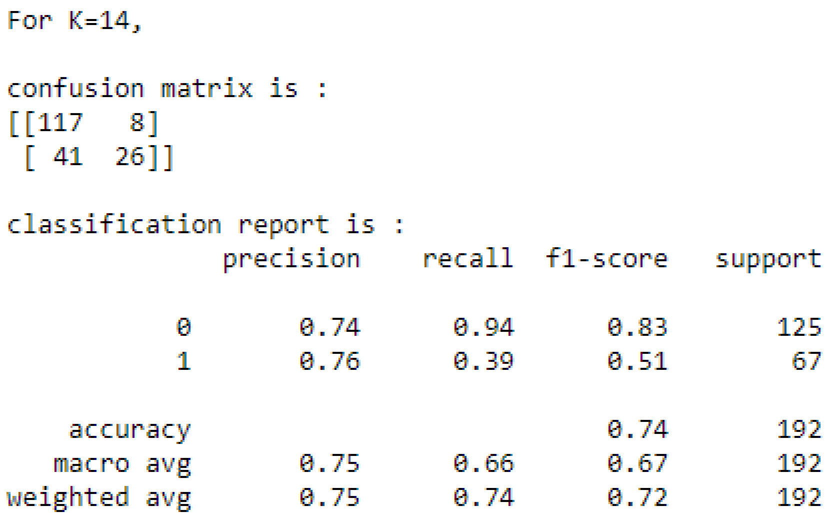
For K=14,

Clause:

```
knn = KNeighborsClassifier(n_neighbors=14)
knn.fit(X_train, y_train)
pred3=knn.predict(X_test)
```

```
print('For K=14, \n')
print('confusion matrix is : ')
print(confusion_matrix(y_test ,pred3),'\n')
print('classification report is : ')
print(classification_report(y_test ,pred3))
```

Result:



```
For K=14,

confusion matrix is :
[[117   8]
 [ 41  26]]

classification report is :
              precision    recall  f1-score   support

           0       0.74      0.94      0.83       125
           1       0.76      0.39      0.51        67

    accuracy                           0.74       192
   macro avg       0.75      0.66      0.67       192
weighted avg       0.75      0.74      0.72       192
```

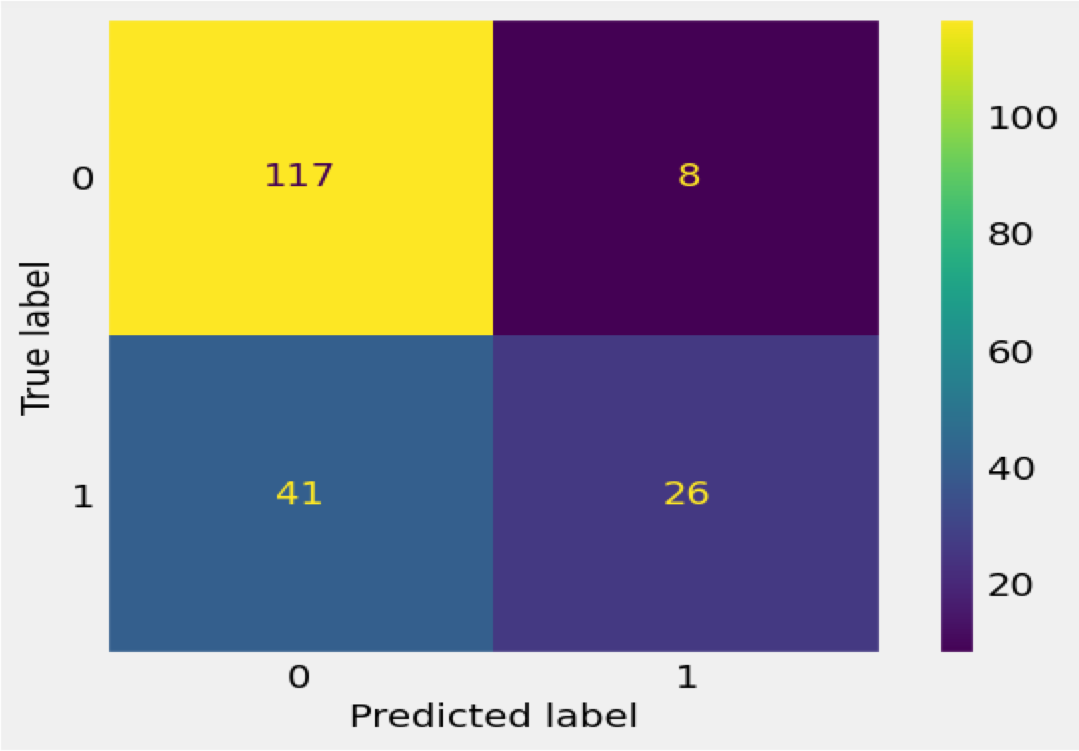
Query 26:

Confusion Matrix Display for K=14

Clause:

```
from sklearn.metrics import ConfusionMatrixDisplay
c=confusion_matrix(y_test ,pred3)
c2=ConfusionMatrixDisplay(c)
c2.plot()
plt.grid(False)
```

Result:



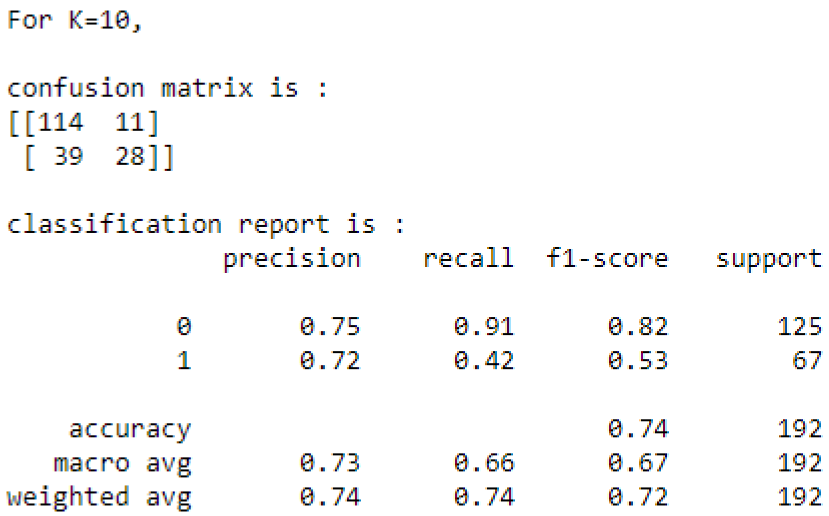
Query 27:

For k=10

Clause:

```
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
pred5=knn.predict(X_test)
print('For K=8, \n')
print('confusion matrix is : ')
print(confusion_matrix(y_test ,pred5),'\n')
print('classification report is : ')
print(classification_report(y_test ,pred5))
```

Result:



```
For K=10,

confusion matrix is :
[[114  11]
 [ 39  28]]

classification report is :
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       125
           1       0.72      0.42      0.53        67

    accuracy                           0.74       192
   macro avg       0.73      0.66      0.67       192
weighted avg       0.74      0.74      0.72       192
```

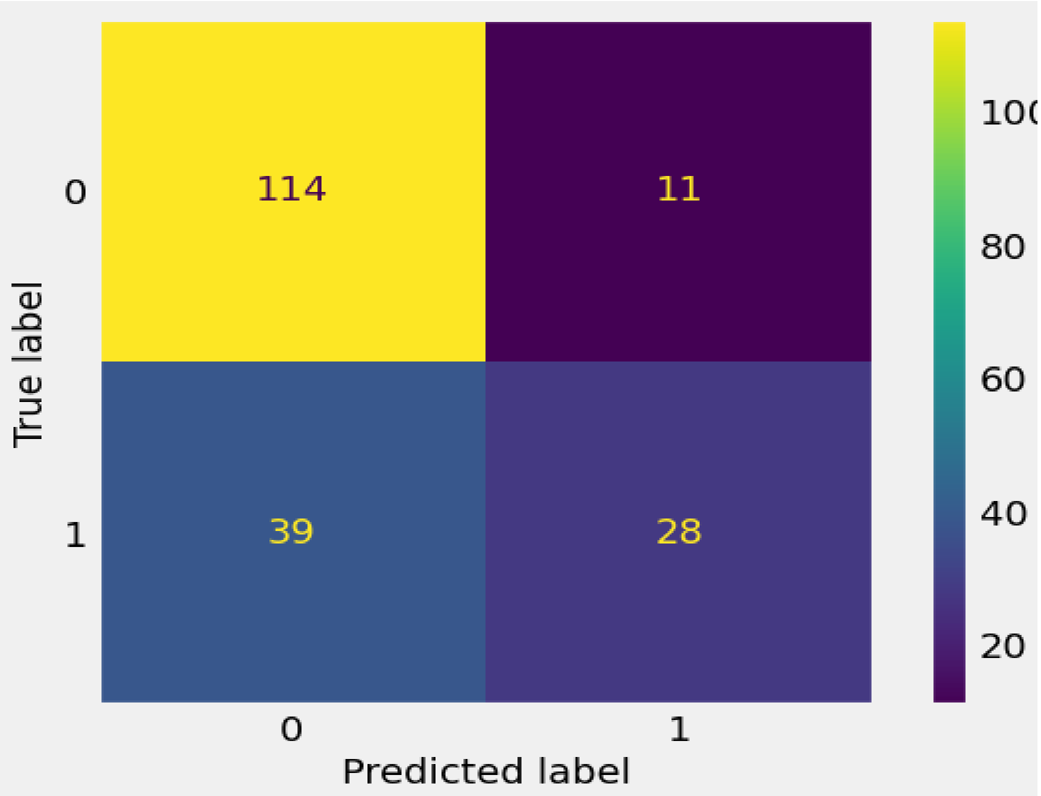
Query 28:

Confusion Matrix Display for K=14

Clause:

```
from sklearn.metrics import ConfusionMatrixDisplay
c=confusion_matrix(y_test ,pred5)
c2=ConfusionMatrixDisplay(c)
c2.plot()
plt.grid(False)
```

Result:



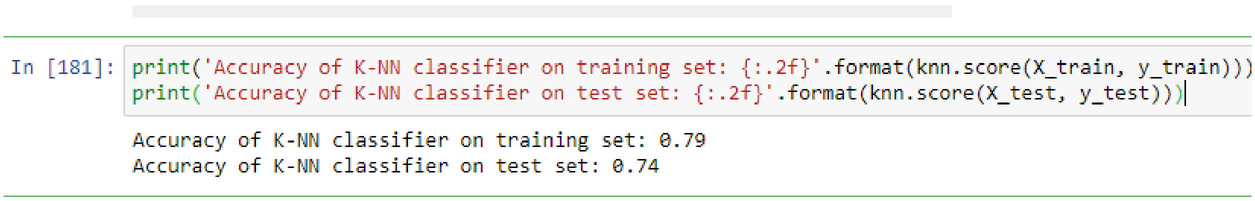
Query 29:

Check the accuracy

Clause:

```
print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))
```

Result:



```
In [181]: print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_train)))
          print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))

          Accuracy of K-NN classifier on training set: 0.79
          Accuracy of K-NN classifier on test set: 0.74
```

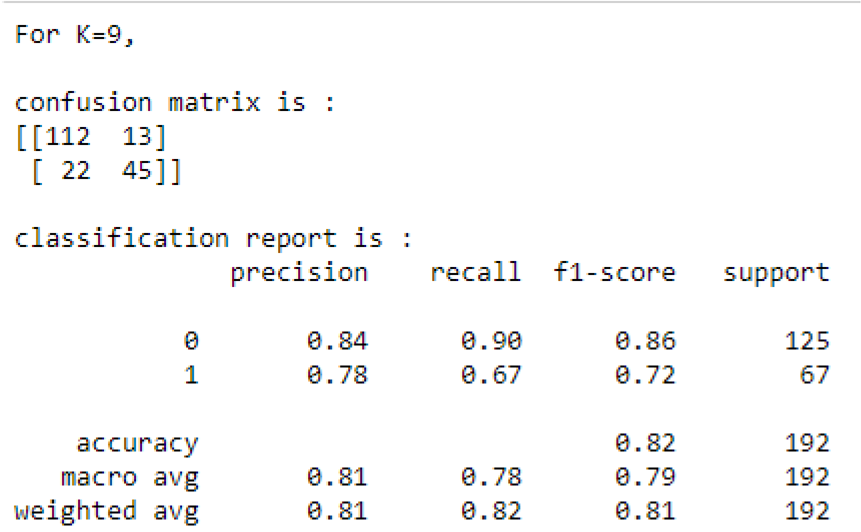
Query 30:

For K=9

Clause:

```
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train, y_train)
pred6=knn.predict(X_test)
print('For K=9, \n')
print('confusion matrix is : ')
print(confusion_matrix(y_test ,pred6),'\n')
print('classification report is : ')
print(classification_report(y_test ,pred6))
```

Result:



```
For K=9,

confusion matrix is :
[[112  13]
 [ 22  45]]

classification report is :
              precision    recall  f1-score   support

           0       0.84      0.90      0.86       125
           1       0.78      0.67      0.72        67

    accuracy                           0.82       192
   macro avg       0.81      0.78      0.79       192
weighted avg       0.81      0.82      0.81       192
```

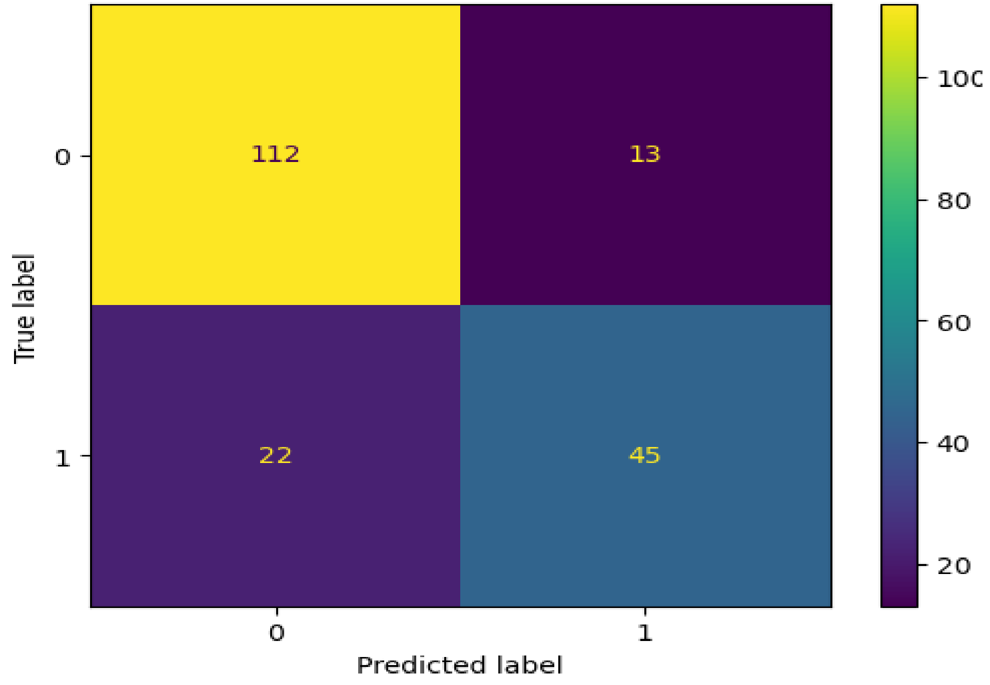
Query 31:

Confusion Matrix Display for K=9

Clause:

```
from sklearn.metrics import ConfusionMatrixDisplay
c=confusion_matrix(y_test ,pred6)
c2=ConfusionMatrixDisplay(c)
c2.plot()
plt.grid(False)
```

Result:



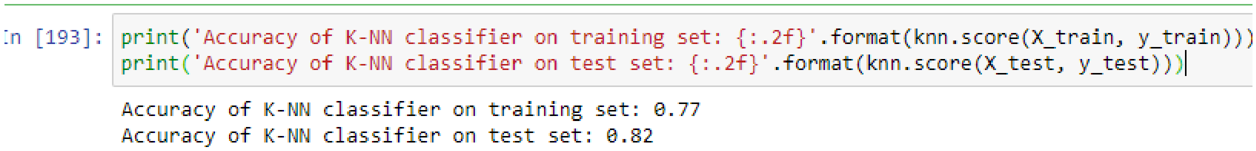
Query 32:

Check the accuracy

Clause:

print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))

Result:



```
In [193]: print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_train)))
          print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))

          Accuracy of K-NN classifier on training set: 0.77
          Accuracy of K-NN classifier on test set: 0.82
```

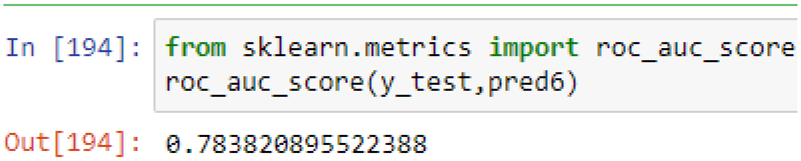
Query 33:

Check the accuracy

Clause:

from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,pred6)

Result:



```
In [194]: from sklearn.metrics import roc_auc_score
          roc_auc_score(y_test,pred6)

Out[194]: 0.783820895522388
```