Institute of Information Technology (IIT)

Jahangirnagar University

Lab Report: 05

Submitted by:

Name: Zannat Hossain Tamim

Roll No:1970

Lab Date:08.08.23

Submission Date: 18.08.23

Lab Report # Day 05

Query 1:

Read CSV file and print the file

Clause:

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

Result :

```
In [4]: df
```

Out[4]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	T
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4233	1	50	1.0	1	1.0	0.0	0	1	0	313.0	179.0	92.0	25.97	66.0	86.0	
4234	1	51	3.0	1	43.0	0.0	0	0	0	207.0	126.5	80.0	19.71	65.0	68.0	
4235	0	48	2.0	1	20.0	NaN	0	0	0	248.0	131.0	72.0	22.00	84.0	86.0	
4236	0	44	1.0	1	15.0	0.0	0	0	0	210.0	126.5	87.0	19.16	86.0	NaN	
4237	0	52	2.0	0	0.0	0.0	0	0	0	269.0	133.5	83.0	21.47	80.0	107.0	

4238 rows × 16 columns

Query 2: Print the top 5 rows.

Clause:

```
df.head()
```

Result :

```
In [51]: df.head()
```

Out[51]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	Te
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	

Query 3: Print the last 5 rows of data frame.

Clause:

df.tail()

Result :

```
In [5]: df.tail()
```

Out[5]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose
4233	1	50	1.0	1	1.0	0.0	0	1	0	313.0	179.0	92.0	25.97	66.0	86.0
4234	1	51	3.0	1	43.0	0.0	0	0	0	207.0	126.5	80.0	19.71	65.0	68.0
4235	0	48	2.0	1	20.0	NaN	0	0	0	248.0	131.0	72.0	22.00	84.0	86.0
4236	0	44	1.0	1	15.0	0.0	0	0	0	210.0	126.5	87.0	19.16	86.0	NaN
4237	0	52	2.0	0	0.0	0.0	0	0	0	269.0	133.5	83.0	21.47	80.0	107.0

Query 4:

To get the shape of the DataFrame

Clause:

df.shape

Result :

```
In [53]: df.shape
Out[53]: (3656, 16)
```

Query 5:

To get the information of the DataFrame

Clause:

df.info()

Result :

```
In [56]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3656 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   male             3656 non-null   int64  
 1   age              3656 non-null   int64  
 2   education        3656 non-null   float64
 3   currentSmoker    3656 non-null   int64  
 4   cigsPerDay       3656 non-null   float64
 5   BPMeds           3656 non-null   float64
 6   prevalentStroke  3656 non-null   int64  
 7   prevalentHyp     3656 non-null   int64  
 8   diabetes         3656 non-null   int64  
 9   totChol          3656 non-null   float64
 10  sysBP            3656 non-null   float64
 11  diaBP            3656 non-null   float64
 12  BMI              3656 non-null   float64
 13  heartRate        3656 non-null   float64
 14  glucose          3656 non-null   float64
 15  TenYearCHD       3656 non-null   int64  
dtypes: float64(9), int64(7)
memory usage: 485.6 KB
```

Query 6:

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

Clause:

df.describe()

Result :

```
In [58]: df.describe()
```

Out[58]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP
count	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000	3656.000000
mean	0.443654	49.557440	1.979759	0.489059	9.022155	0.030361	0.005744	0.311543	0.027079	236.873085	132.368025
std	0.496883	8.561133	1.022657	0.499949	11.918869	0.171602	0.075581	0.463187	0.162335	44.096223	22.092444
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	113.000000	83.500000
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	206.000000	117.000000
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	234.000000	128.000000
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.000000	0.000000	1.000000	0.000000	263.250000	144.000000
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.000000	1.000000	1.000000	1.000000	600.000000	295.000000

Query 7:

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

Clause:

```
data.isnull().sum()
```

Result:

```
In [6]: df.isnull().sum()

Out[6]: 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
```

Query 8 : Total number of rows with missing values

Clause:

```
count=0
for i in df.isnull().sum(axis=1):
    if i>0:
        count=count+1
count
```

Result:

```
In [28]: count=0
         for i in df.isnull().sum(axis=1):
             if i>0:
                 count=count+1
         count

Out[28]: 582
```

Query 9:

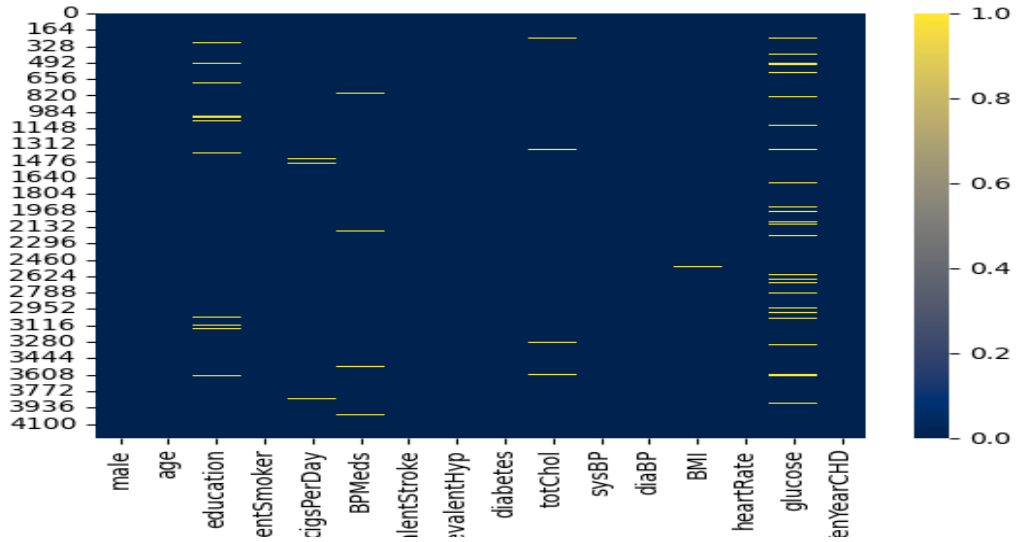
Create a heatmap of the missing values in the data frame.

Clause:

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

Result:

```
Out[7]: <Axes: >
```



Query 10: Impute the NaN values with the mean

Clause:

```
df = df.fillna(df[['education','cigsPerDay',
'BPMeds','totChol','BMI','heartRate','glucose']].mean())
print(df)
```

Result:

```
      male  age  education  currentSmoker  cigsPerDay   BPMeds  \
0        1   39        4.0              0         0.0  0.00000
1        0   46        2.0              0         0.0  0.00000
2        1   48        1.0              1        20.0  0.00000
3        0   61        3.0              1        30.0  0.00000
4        0   46        3.0              1        23.0  0.00000
...    ...  ...        ...            ...         ...      ...
4233     1   50        1.0              1         1.0  0.00000
4234     1   51        3.0              1        43.0  0.00000
4235     0   48        2.0              1        20.0  0.02963
4236     0   44        1.0              1        15.0  0.00000
4237     0   52        2.0              0         0.0  0.00000

      prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  \
0                   0             0         0    195.0  106.0   70.0  26.97
1                   0             0         0    250.0  121.0   81.0  28.73
2                   0             0         0    245.0  127.5   80.0  25.34
3                   0             1         0    225.0  150.0   95.0  28.58
4                   0             0         0    285.0  130.0   84.0  23.10
...               ...           ...       ...      ...    ...    ...    ...
4233                0             1         0    313.0  179.0   92.0  25.97
4234                0             0         0    207.0  126.5   80.0  19.71
4235                0             0         0    248.0  131.0   72.0  22.00
4236                0             0         0    210.0  126.5   87.0  19.16
4237                0             0         0    269.0  133.5   83.0  21.47

      heartRate     glucose  TenYearCHD
0          80.0   77.000000           0
1          95.0   76.000000           0
2          75.0   70.000000           0
```

Query 10: After handling all the NaN values

Clause:

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

Result:

```
[43]: df.isnull().sum()

[43]: 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
```

Query 11 : To print columns

Clause:

```
df.columns()
```

Result:

```
In [16]: df.columns

Out[16]: Index(['male', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
                'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
                'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
               dtype='object')
```

Query 12: 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:

]:

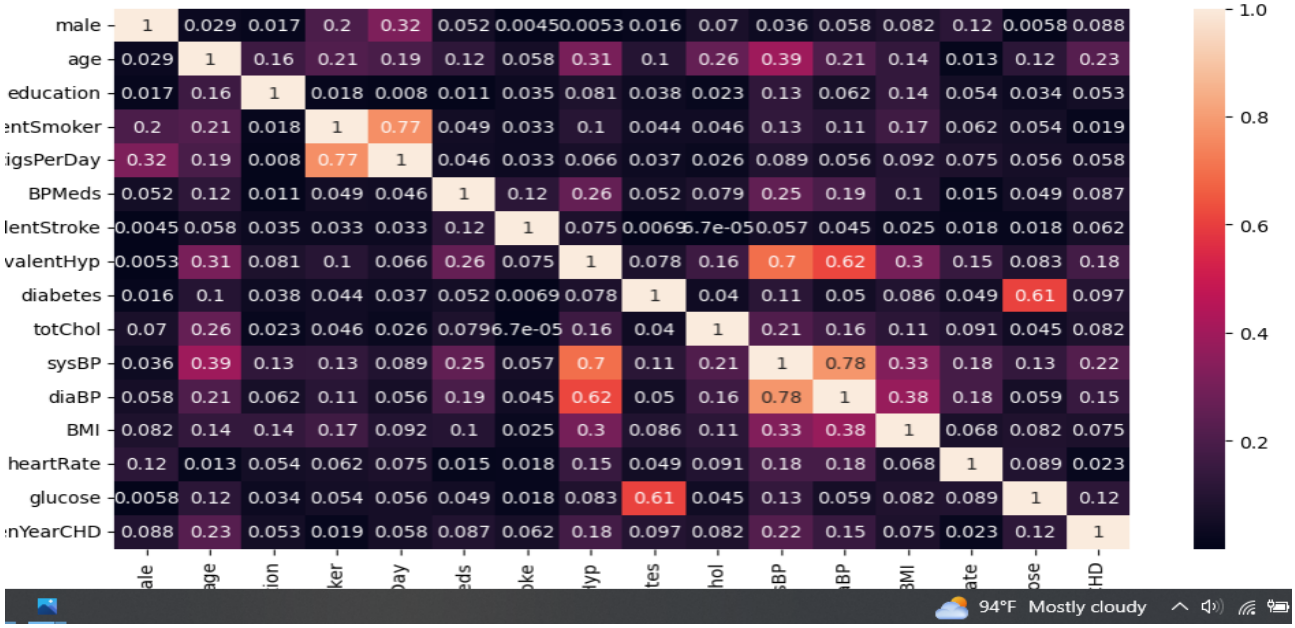
	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP
male	1.000000	-0.028979	0.017126	0.197596	0.316807	-0.052204	-0.004546	0.005313	0.015708	-0.069974	-0.035989	0.057933
age	-0.028979	1.000000	-0.163613	-0.213748	-0.192366	0.121980	0.057655	0.307194	0.101258	0.260270	0.394302	0.206104
education	0.017126	-0.163613	1.000000	0.018301	0.007962	-0.010607	-0.035110	-0.080993	-0.038146	-0.022507	-0.128260	-0.061755
currentSmoker	0.197596	-0.213748	0.018301	1.000000	0.766970	-0.048632	-0.032988	-0.103260	-0.044295	-0.046285	-0.130230	-0.107746
cigsPerDay	0.316807	-0.192366	0.007962	0.766970	1.000000	-0.045826	-0.032706	-0.065947	-0.037063	-0.026025	-0.088505	-0.056391
BPMeds	-0.052204	0.121980	-0.010607	-0.048632	-0.045826	1.000000	0.115003	0.259243	0.051571	0.078909	0.252047	0.192490
prevalentStroke	-0.004546	0.057655	-0.035110	-0.032988	-0.032706	0.115003	1.000000	0.074830	0.006949	0.000067	0.057009	0.045190
prevalentHyp	0.005313	0.307194	-0.080993	-0.103260	-0.065947	0.259243	0.074830	1.000000	0.077808	0.163041	0.696755	0.615751
diabetes	0.015708	0.101258	-0.038146	-0.044295	-0.037063	0.051571	0.006949	0.077808	1.000000	0.040092	0.111283	0.050329
totChol	-0.069974	0.260270	-0.022507	-0.046285	-0.026025	0.078909	0.000067	0.163041	0.040092	1.000000	0.207609	0.163903
sysBP	-0.035989	0.394302	-0.128260	-0.130230	-0.088505	0.252047	0.057009	0.696755	0.111283	0.207609	1.000000	0.784002
diaBP	0.057933	0.206104	-0.061755	-0.107746	-0.056391	0.192490	0.045190	0.615751	0.050329	0.163903	0.784002	1.000000
BMI	0.081506	0.135283	-0.135635	-0.167276	-0.092453	0.099552	0.024840	0.300572	0.086250	0.114789	0.325247	0.376544
heartRate	-0.116601	-0.012819	-0.053626	0.062348	0.074851	0.015175	-0.017676	0.147222	0.048993	0.090676	0.182174	0.181246
glucose	0.005818	0.116850	-0.033721	-0.054157	-0.056088	0.048905	0.018055	0.082924	0.605705	0.044583	0.134608	0.058647
TenYearCHD	0.088428	0.225256	-0.053384	0.019456	0.057775	0.086774	0.061810	0.177603	0.097317	0.081624	0.216429	0.145299

Query 13: 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:



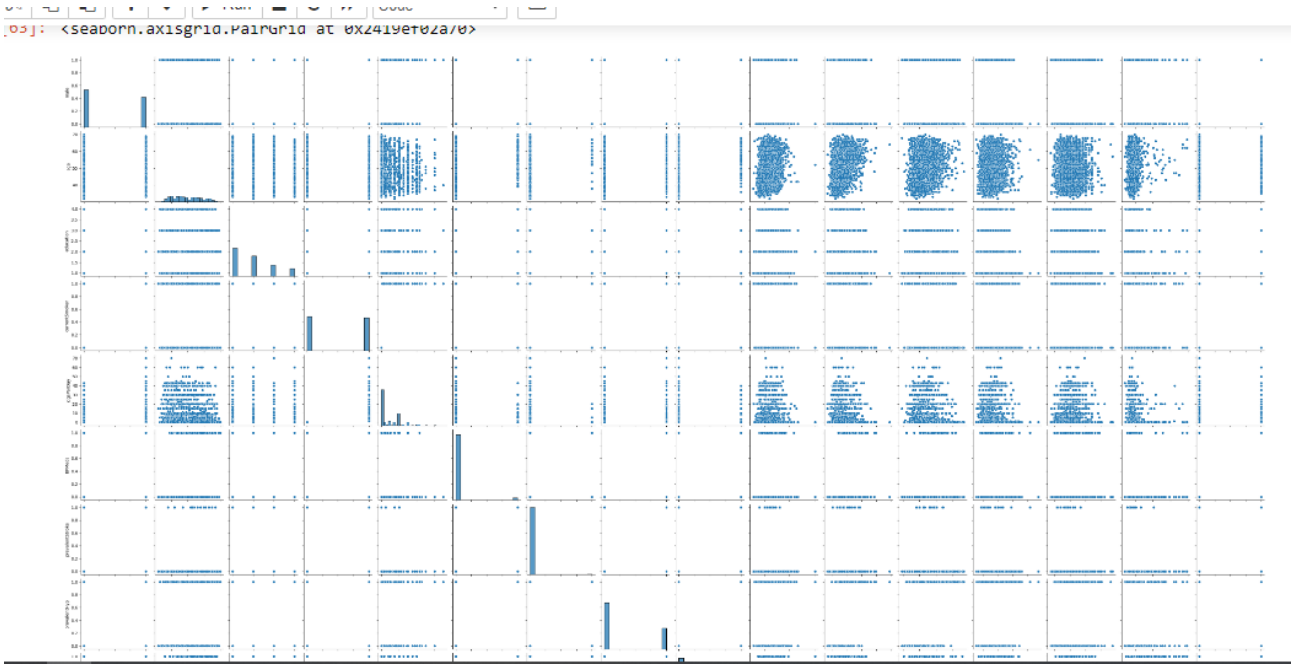
Query 14:

Plot pairwise relationships between variables of the dataset

Clause:

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

Result:



Query 15:

Number of males with no heart disease and number of patients with risk of heart disease

Clause:

```
df.TenYearCHD.value_counts()
```

Result:

```
t[20]: 0    3594
       1     644
       Name: TenYearCHD, dtype: int64
```

Query 16:

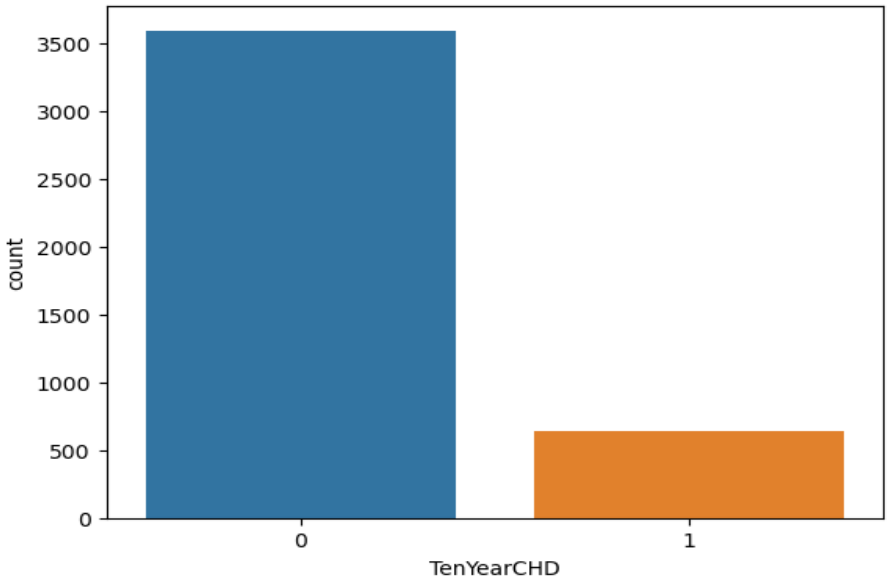
Number of males with no heart disease and number of patients with risk of heart disease

Clause:

```
sns.countplot(x='TenYearCHD',data=df)
```

Result:

```
ıt[21]: <Axes: xlabel='TenYearCHD', ylabel='count'>
```



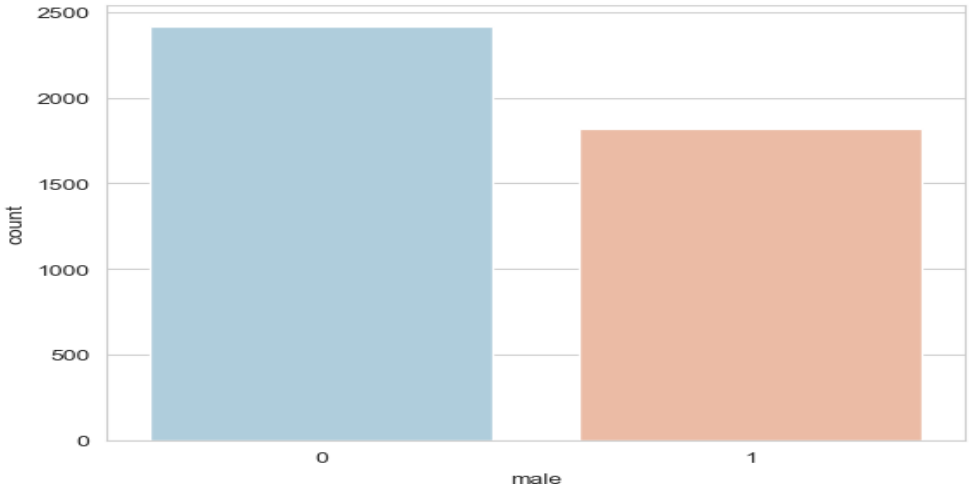
Query 17:

Clause:

```
sns.set_style('whitegrid')
sns.countplot(x='male',data=df,palette='RdBu_r')
```

Result:

```
Out[8]: <Axes: xlabel='male', ylabel='count'>
```



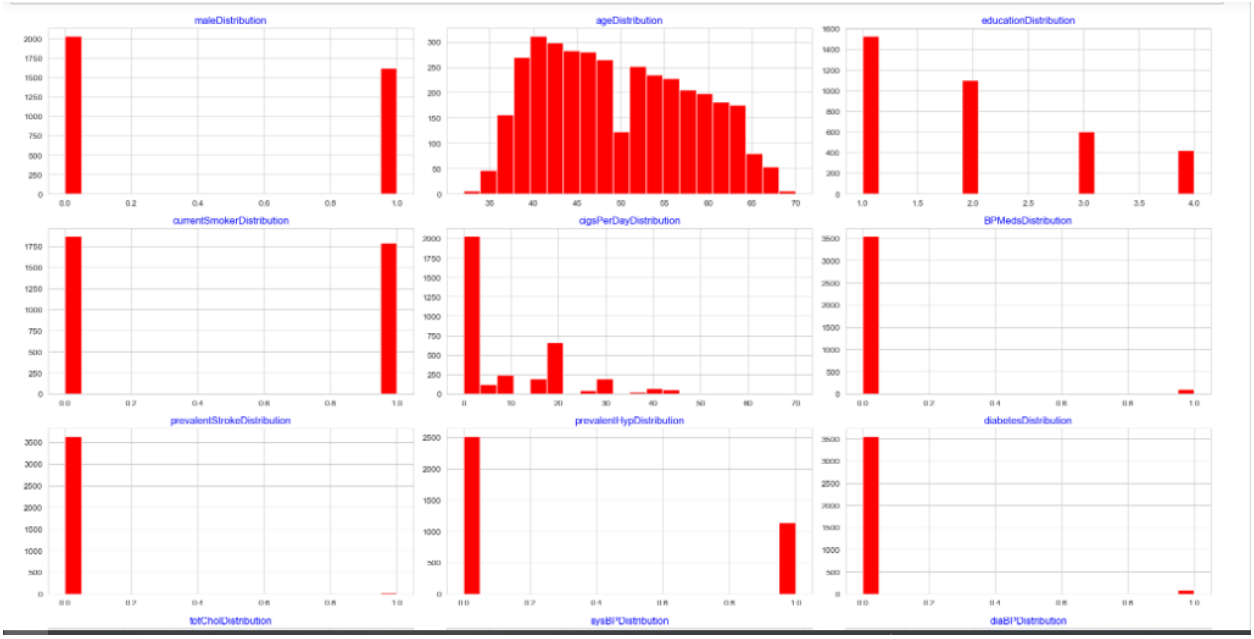
Query 18:

Exploratory Analysis

Clause:

```
def draw_histograms(dataframe, features, rows, cols):
    fig=plt.figure(figsize=(20,20))
    for i, feature in enumerate(features):
        ax=fig.add_subplot(rows,cols,i+1)
        dataframe[feature].hist(bins=20,ax=ax,facecolor='red')
        ax.set_title(feature+"Distribution", color='blue')
    fig.tight_layout()
    plt.show()
draw_histograms(df, df.columns, 6, 3)
```

Result:

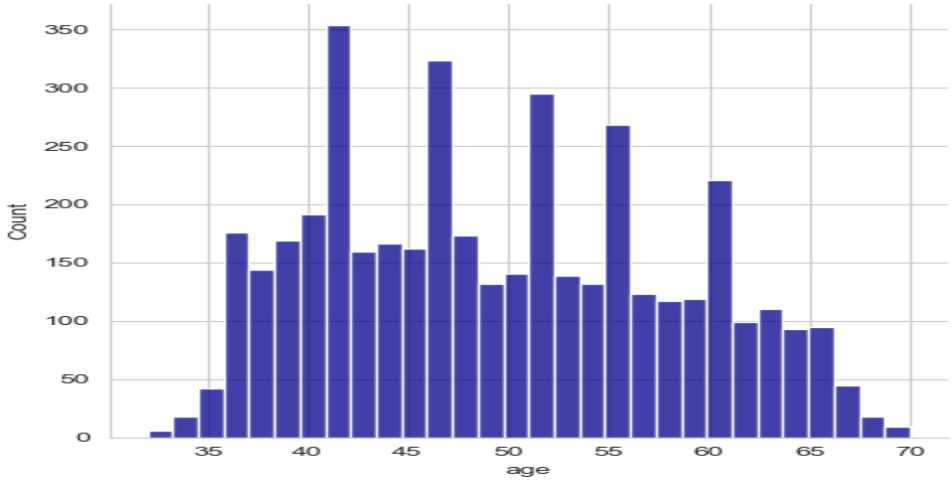


Query 19:

Clause:

```
sns.displot(df['age'].dropna(),kde=False,color='darkblue',bins=30)
```

Result:



Query 20:

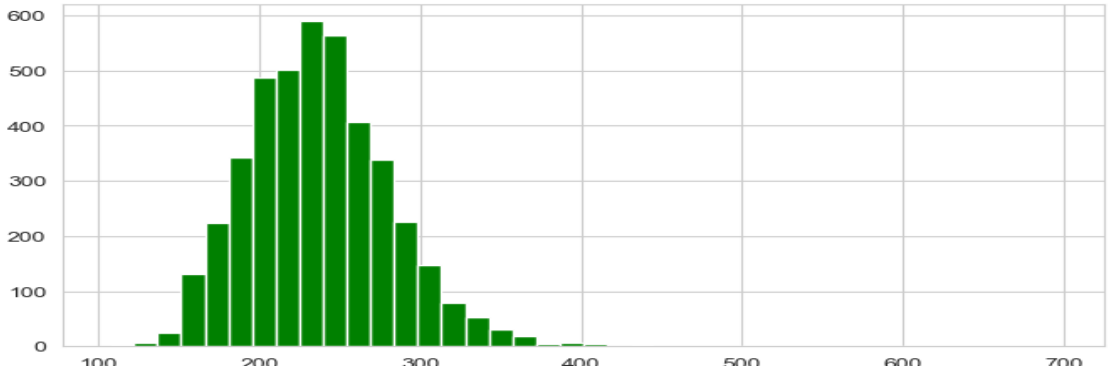
To print the graph of total Chol

Clause:

```
df['totChol'].hist(color='green',bins=40,figsize=(8,4))
```

Result:

```
ut[13]: <Axes: >
```



Query 21:

Clause:

```
from statsmodels.tools import add_constant as add_constant
h = add_constant(df)
h.head()
```

Result:

Out[36]:

	const	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucos
0	1.0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.
1	1.0	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.
2	1.0	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.
3	1.0	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.
4	1.0	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.

Query 22:

Splitting data to train and test split

Clause:

```
import sklearn
new=df[['age','male','cigsPerDay','totChol','sysBP','glucose','TenYearCHD']]
x=new.iloc[:,:-1]
y=new.iloc[:,-1]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=5)
```

Result:

In [35]:
```
import sklearn
new=df[['age','male','cigsPerDay','totChol','sysBP','glucose','TenYearCHD']]
x=new.iloc[:,:-1]
y=new.iloc[:,-1]
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=5)
```

Query 23: To print x_test data

Clause:

```
x_test
```

Result:

```
[36]: x_test
```

Out[36]:

	age	male	cigsPerDay	totChol	sysBP	glucose
1856	61	1	0.0	204.0	120.0	75.000000
2370	60	0	0.0	212.0	146.0	72.000000
424	38	0	15.0	176.0	110.0	113.000000
1736	38	1	20.0	279.0	124.0	75.000000
1183	57	0	0.0	233.0	184.0	40.000000
...	...	...	...	...	...	...
2530	49	0	9.0	231.0	137.0	81.966753
2648	60	0	0.0	254.0	114.0	84.000000
2676	51	1	20.0	215.0	115.0	77.000000
2582	41	0	0.0	180.0	115.0	64.000000
3720	44	1	0.0	175.0	104.0	82.000000

848 rows × 6 columns

Query 24: To print y_test data

Clause:

```
y_test
```

Result:

```
In [37]: y_test

Out[37]: 1856    0
         2370    1
         424     0
         1736    0
         1183    1
                ..
         2530    0
         2648    0
         2676    0
         2582    0
         3720    0
         Name: TenYearCHD, Length: 848, dtype: int64
```

Query 25:

To create an object named m, fit and predict the x_test data of the model

Clause:

```
from sklearn.linear_model import LogisticRegression
m =LogisticRegression()
m.fit(x_train,y_train)
y = m.predict(x_test)
```

Result:

```
44]:  array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
             0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Query 26:

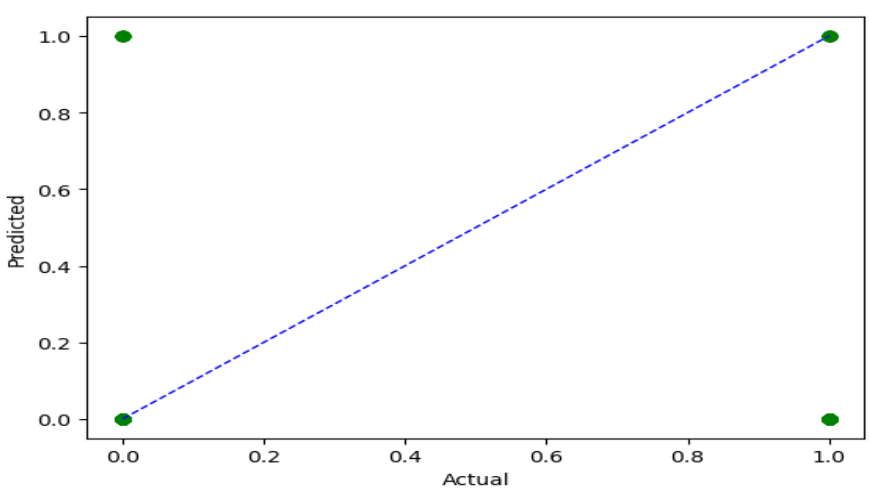
To show the scatter diagram of y_train and y_pre_train

Clause:

```
y_predict = m.predict(x_test)
y_pre_train= m.predict(x_train)
plt.scatter(y_train,y_pre_train,c='green',lw=1)
plt.plot([y_train.min(),y_train.max()],[y_train.min(),y_train.max()],'k--',c='blue',lw=1)
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

Result:

```
Out[66]:  Text(0, 0.5, 'Predicted')
```



Query 27:

Evaluate the performance of the model

Clause:

```
m.score(x_test,y_test)
```

Result:

```
In [41]: m.score(x_test,y_test)
Out[41]: 0.8384433962264151
```

Query 28:

Probability of the prediction of x_test data

Clause:

```
m.predict_proba(x_test)
```

Result:

```
Out[42]: array([[0.81607673, 0.18392327],
                [0.83879092, 0.16120908],
                [0.95666451, 0.04333549],
                ...,
                [0.85621833, 0.14378167],
                [0.97042969, 0.02957031],
                [0.94432298, 0.05567702]])
```

Query 29:

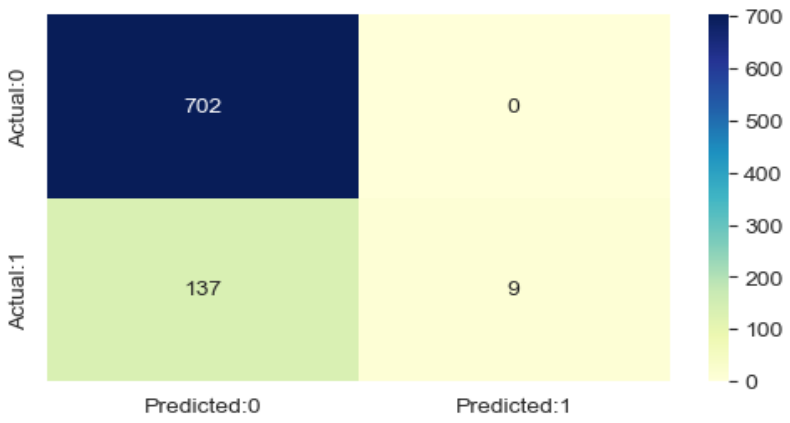
Confusion matrix

Clause:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y)
conf_matrix = pd.DataFrame(data=cm,
columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (6,3))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Result:

```
Out[50]: <Axes: >
```



Query 30:

Draw Residual Histogram to calculate the precision of a classification model

Clause:

```
from sklearn.metrics import classification_report
accuracy = sklearn.metrics.accuracy_score(y_test, y)
precision = sklearn.metrics.precision_score(y_test, y)
recall = sklearn.metrics.recall_score(y_test, y)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

Result:

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.8384433962264151
Precision: 1.0
Recall: 0.06164383561643835
```

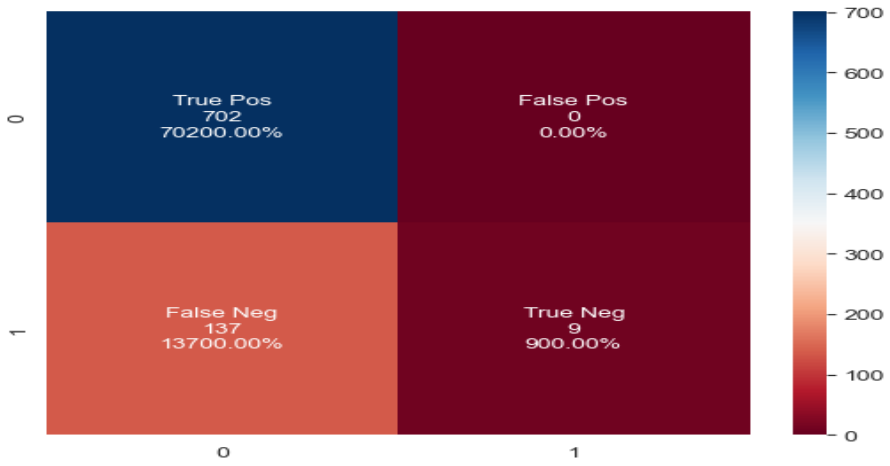
Query 31:

Check the accuracy

Clause:

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y)
g_names = ['True Pos','False Pos','False Neg','True Neg']
g_counts = ["{0:0.0f}".format(value) for value in conf_matrix.flatten()]
g_percentages = ["{0:.2%}".format(value) for value in conf_matrix.flatten()]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
 zip(g_names, g_counts, g_percentages)]
labels = np.asarray(labels).reshape(2, 2)
sns.heatmap(conf_matrix, annot=labels, fmt='', cmap='RdBu')
plt.show()
```

Result:


True Pos
702
70200.00%
False Pos
0
0.00%
False Neg
137
13700.00%
True Neg
9
900.00%
0
1
0
1
700
600
500
400
300
200
100
0