Capestone_Project_HealthCare_PGP

July 28, 2022

[1]:
```
import os
os.getcwd()
```

[1]: '/home/labsuser'

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report , f1_score , accuracy_score ,␣
↪confusion_matrix
```

[3]:
```
df = pd.read_csv("health care diabetes.csv")
```

[4]:
```
df.head()
```

[4]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

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

[5]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

[6]:
```python
df.keys()
```

[6]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

[7]:
```python
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
```

[8]:
```python
df.nunique()
```

[8]:
```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
```

```
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

[9]:
```
df.describe()
```

[9]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

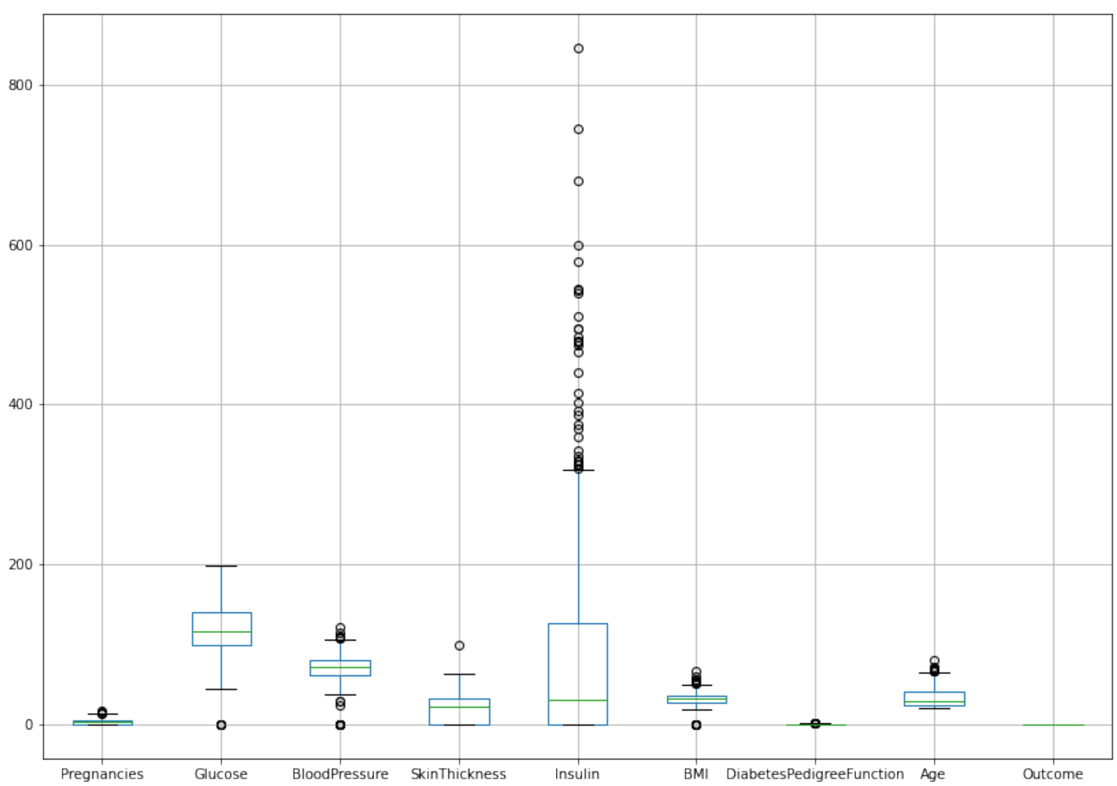
[10]:
```
df.corr()
```

[10]:
```
                          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
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[11]:
```python
plt.figure(figsize = (14,10))
df.boxplot()
```

[11]: <AxesSubplot:>



[12]:
```python
df.isna().sum()
```

[12]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[13]:
```
df[df[['Glucose' , 'BloodPressure' , 'SkinThickness' , 'Insulin' , 'BMI']]==0].
↪count()
```

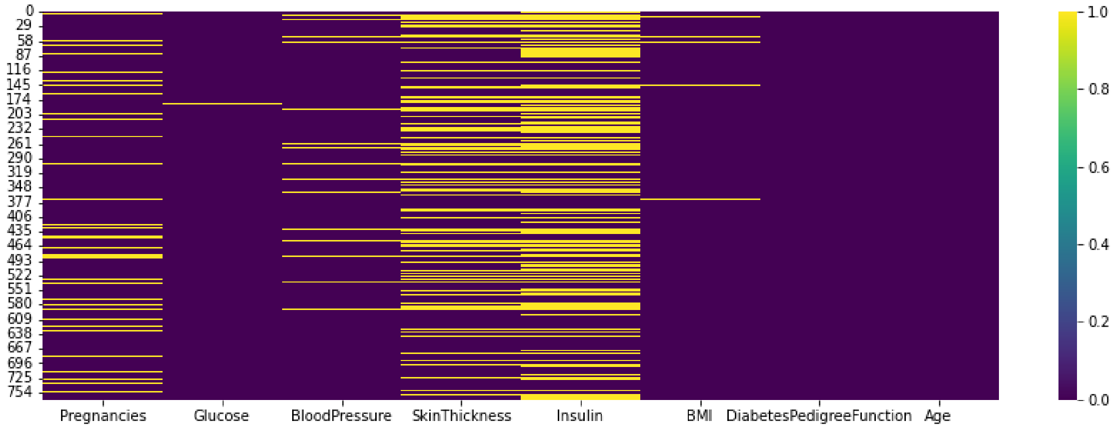
[13]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

[14]:
```
for i in ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']:
    print(i)
    print(df[i].value_counts(normalize=True)[0],'\n\n')
```

```
Glucose
0.006510416666666667


BloodPressure
0.045572916666666664


SkinThickness
0.2955729166666667


Insulin
0.4869791666666667


BMI
0.014322916666666666
```

[15]:
```python
# Checking the missing values using the heat map plot.

df1 = df.drop('Outcome',axis=1)
df2 = df1.replace(0,np.nan)
plt.figure(figsize=(15,5))
sns.heatmap(df2.isna(),cmap = 'viridis')
```

[15]: <AxesSubplot:>



[16]:
```python
df.shape
```

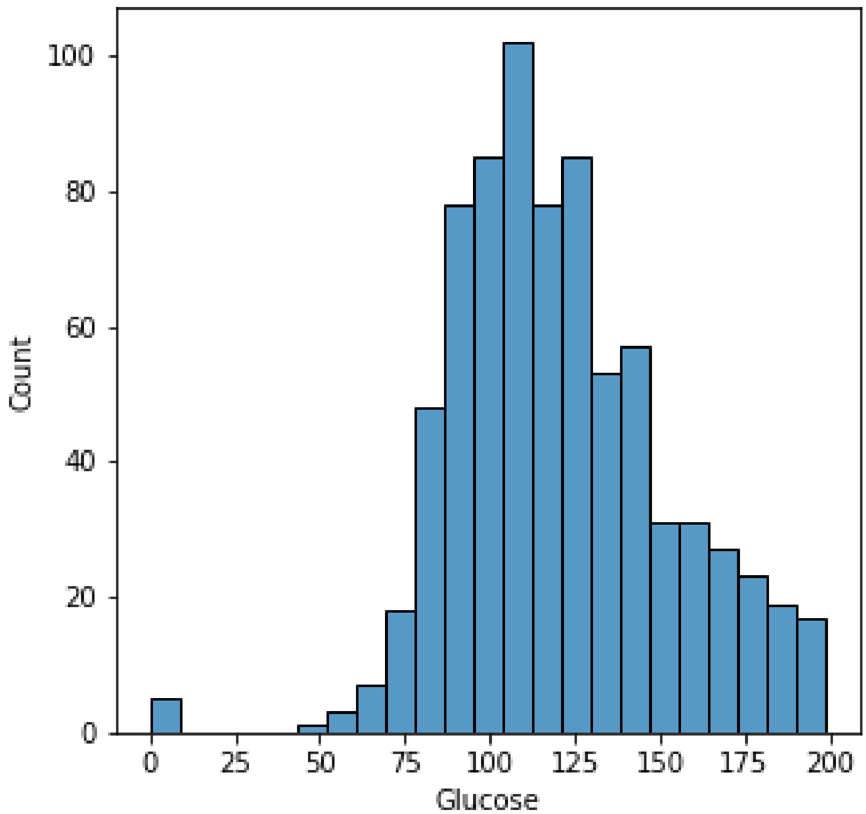
[16]: (768, 9)

[17]:
```python
df2.shape
```

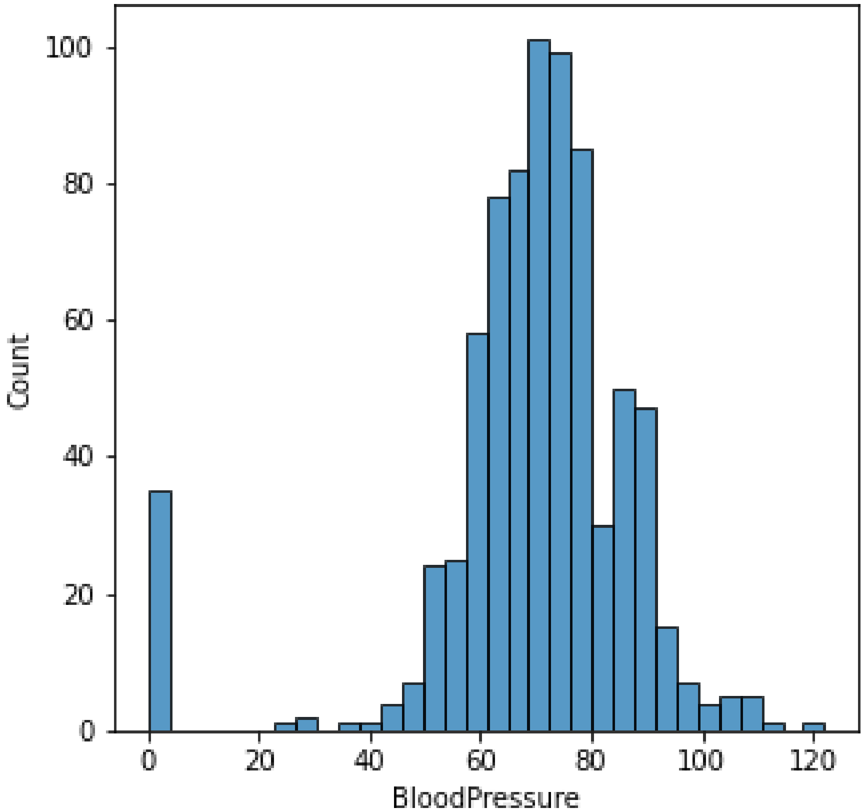
[17]: (768, 8)

[18]:
```python
### Checking the histogram to treat the missing values statistically.
for i in ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']:
    print(i)
    plt.figure(figsize=(5,5))
    sns.histplot(x=i,data=df)
```

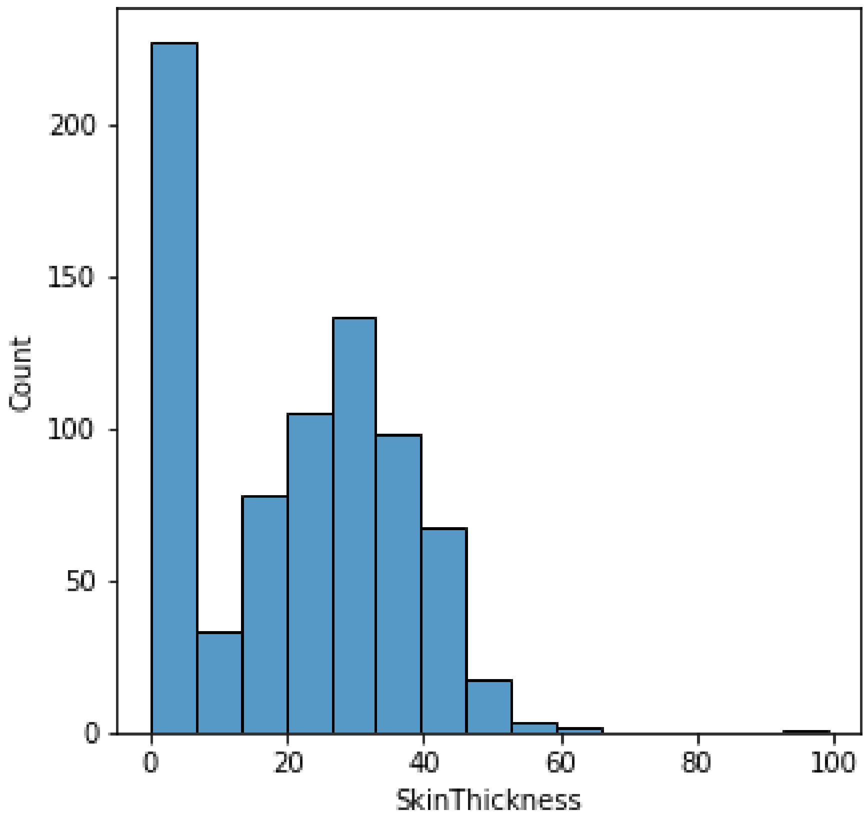
```
Glucose
BloodPressure
SkinThickness
Insulin
BMI
```



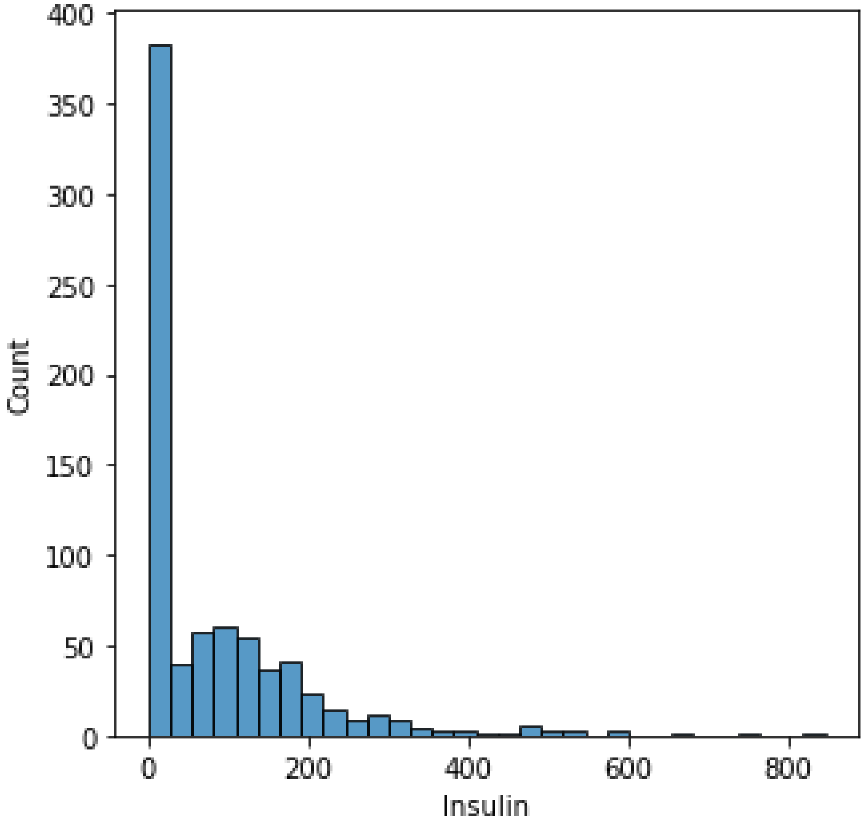
100
80
60
40
20
0
Count
0
25
50
75
100
125
150
175
200
Glucose



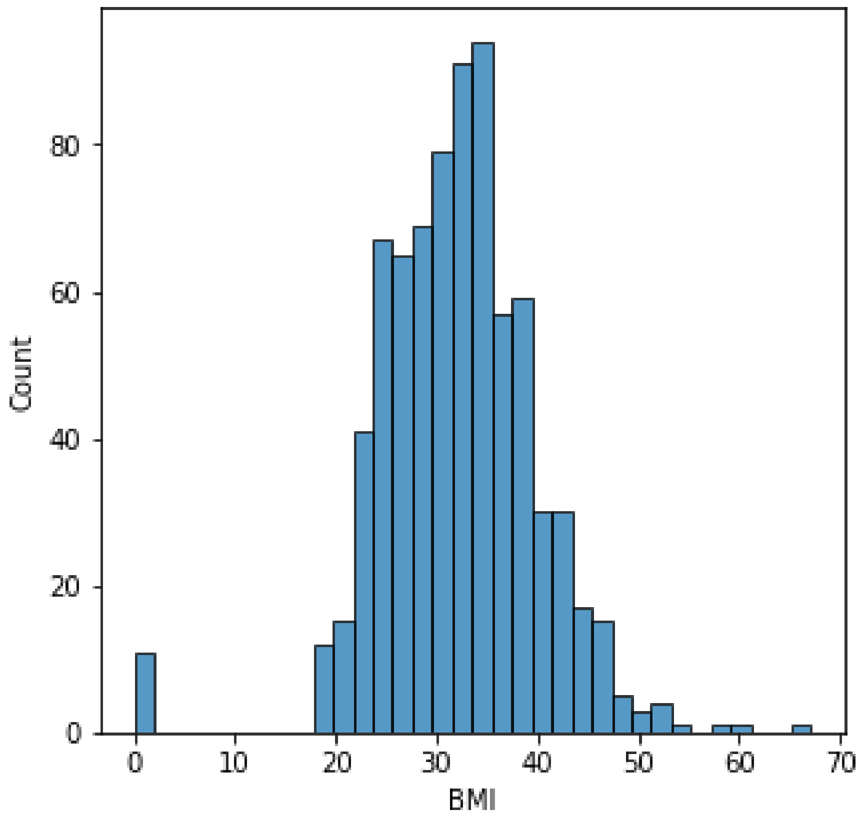
100
80
60
40
20
0
Count
0
20
40
60
80
100
120
BloodPressure



Count
SkinThickness
0
50
100
150
200
0
20
40
60
80
100



400
350
300
250
200
150
100
50
0
Count
0
200
400
600
800
Insulin



In insulin data is right skewed so we have to fill the median in the insulin as mean is far away from median.

In other columns that has we can fill either mean or median so we will good to fill with median in other columns also

[19]:
```
df['Insulin'].median()   # Insulin median with including zero
```

[19]: 30.5

[20]:
```
df[df['Insulin']!=0]['Insulin'].median()  ## Insulin without including Zero.
```

[20]: 125.0

[21]:
```
## Filling all the columns values with median.

for i in ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']:
    print(i)
    print(df[df[i]!=0][i].median())
    df[i].replace(0,df[df[i]!=0][i].median(),inplace=True)
```

```
Glucose
117.0
BloodPressure
72.0
SkinThickness
29.0
Insulin
125.0
BMI
32.3
```

[22]:
```python
df[df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]==0].count()
```

[22]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

So we fill the missing values by median of their respective columns without including zero which means null.

[23]:
```python
df.dtypes
```

[23]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

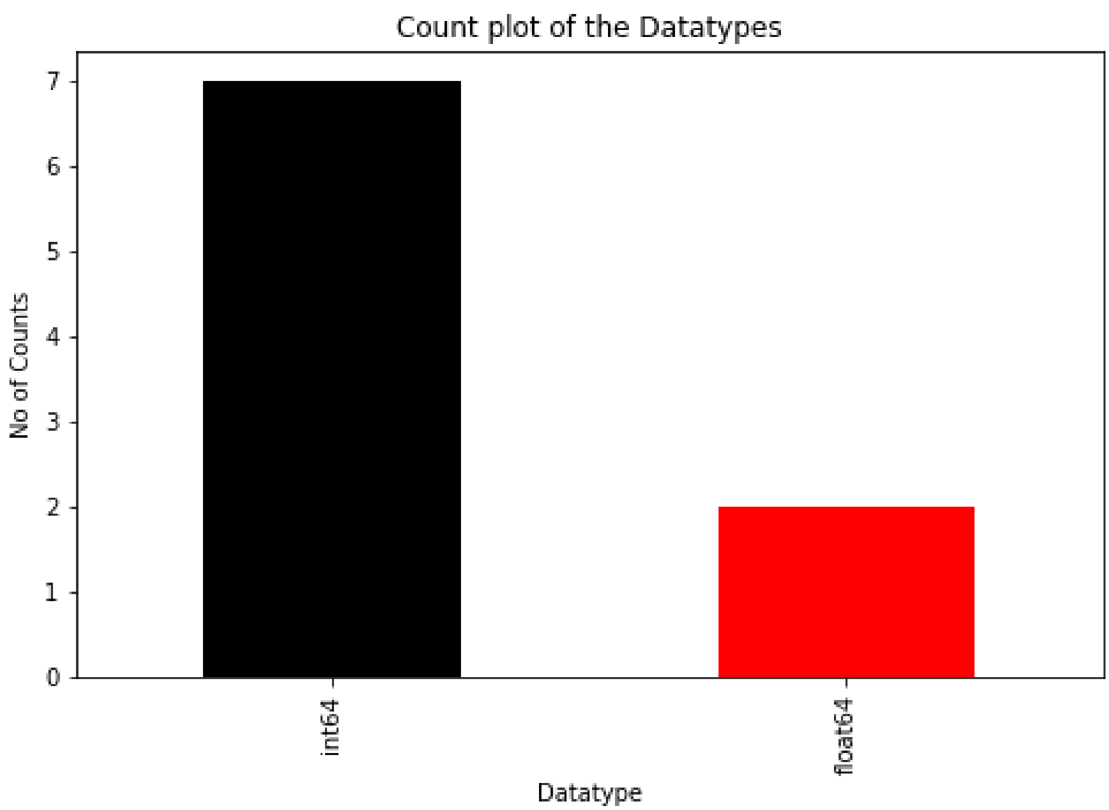
[24]:
```python
### Creating the count frequency plot for the datatypes int64 and float64.

df.dtypes.value_counts()
```

[24]:
```
int64      7
float64    2
dtype: int64
```

[25]:
```
df.dtypes.value_counts(normalize=True)*100
```

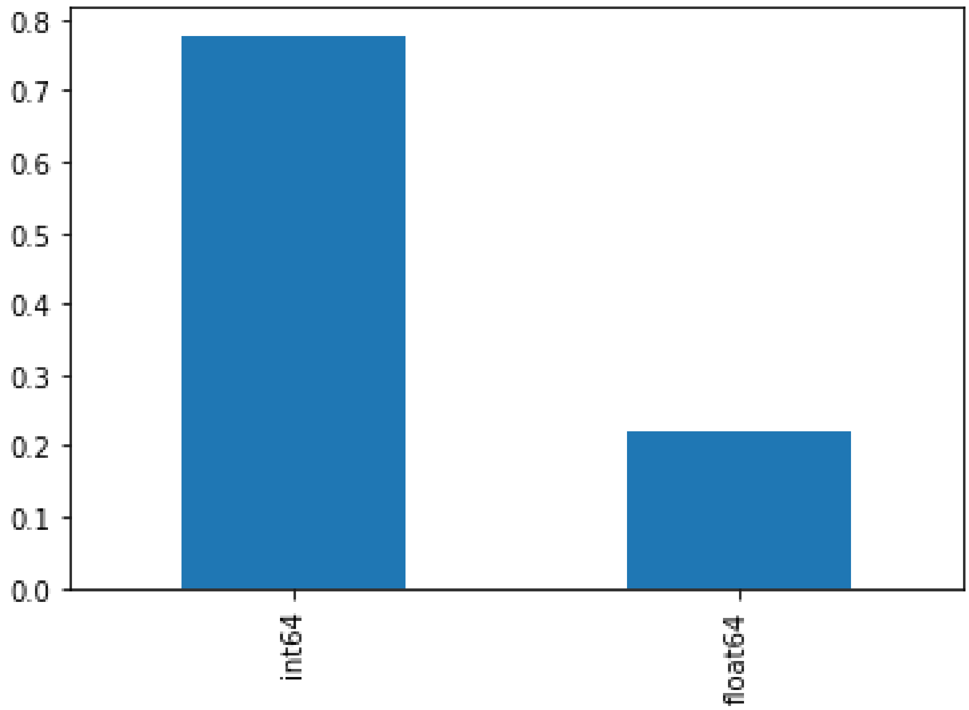
[25]:
```
int64      77.777778
float64    22.222222
dtype: float64
```

[26]:
```
### Plotting the frequency plot descring the datatypes and the count of the␣
↪variables.
plt.figure(figsize=(8,5))
df.dtypes.value_counts().plot(kind = 'bar',color = ['black','red'])
plt.title('Count plot of the Datatypes')
plt.xlabel('Datatype')
plt.ylabel('No of Counts')
plt.show()
```



[27]:
```
df.dtypes.value_counts(normalize=True).plot(kind = 'bar')
```

[27]:
```
<AxesSubplot:>
```



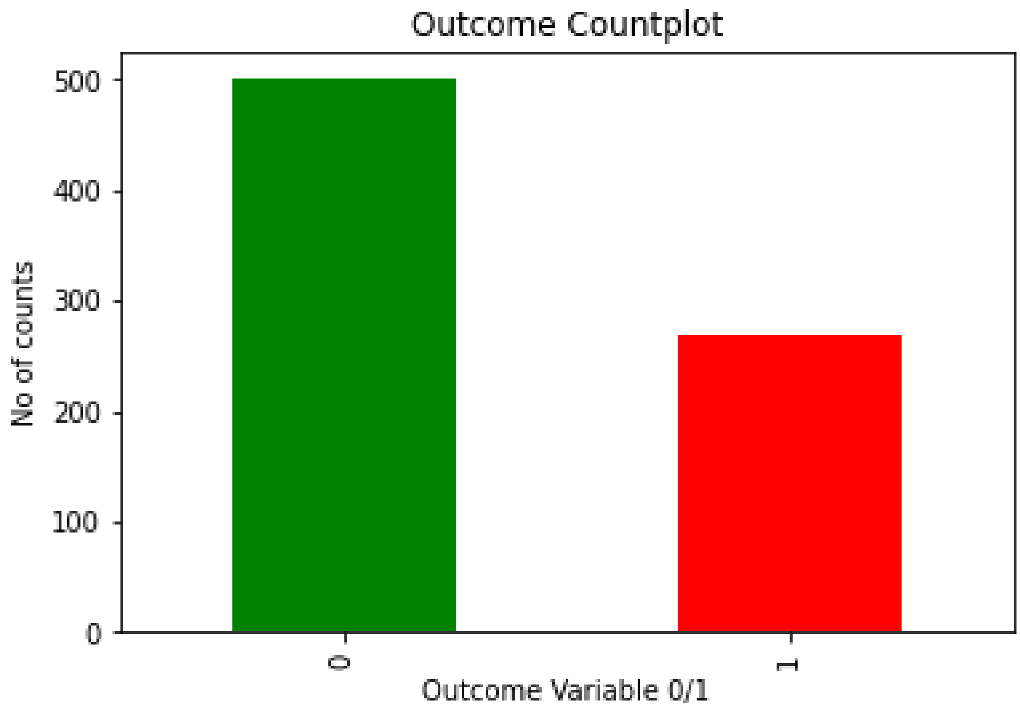
[28]:
```
# Check the balance of the data by plotting the count of outcomes by their␣
↪value.
# Describe your findings and plan future course of action.

df.Outcome.value_counts()
```

[28]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[29]:
```
df.Outcome.value_counts(normalize=True)*100
```

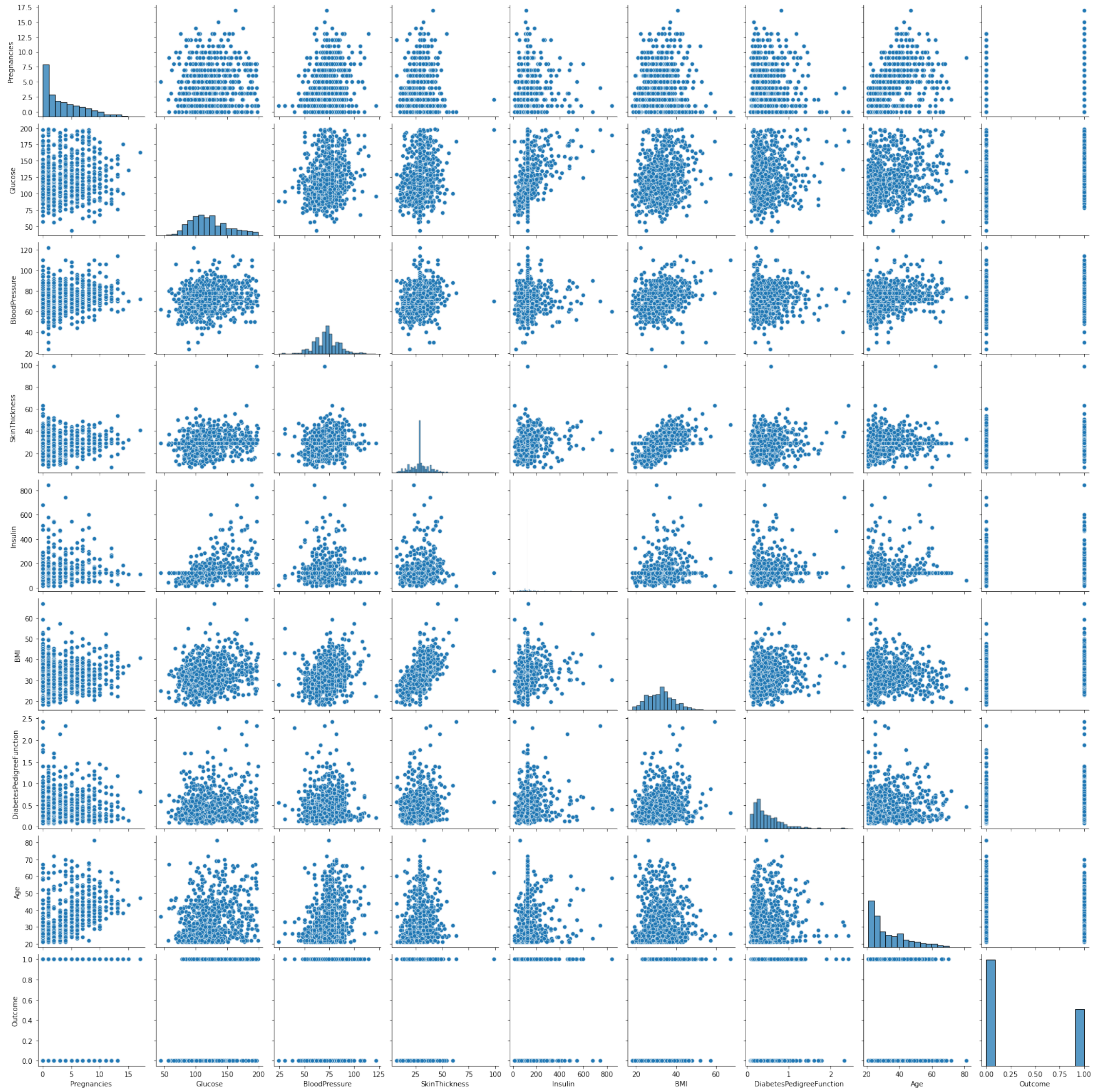
[29]:
```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

[30]:
```
# Plotting the count of outcome

df.Outcome.value_counts().plot(kind = 'bar' , color = ['g','r'])
plt.title('Outcome Countplot')
plt.xlabel('Outcome Variable 0/1')
plt.ylabel('No of counts')
plt.show()
```



[31]:
```
## Creating the Scatter plot between the variables to understand the
↪relationships.

sns.pairplot(df)
```

[31]: <seaborn.axisgrid.PairGrid at 0x7fe3606b7c10>



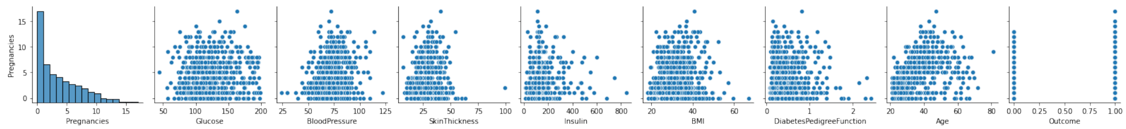
There is a linear relationship between BMI and SkinThickness

[32]:
```
for i in df.columns:
    print(i)
    plt.figure(figsize=(5,5))
    sns.pairplot(y_vars = i,data = df)
```

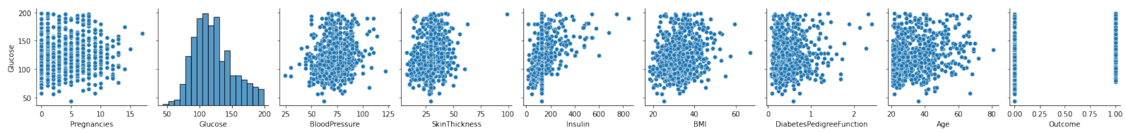
```
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
```

DiabetesPedigreeFunction
Age
Outcome

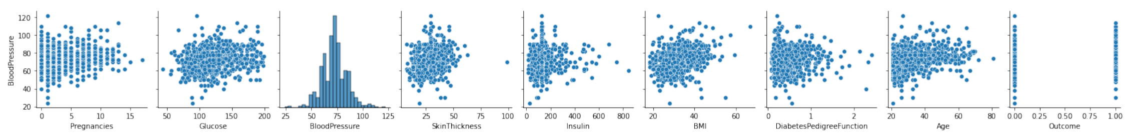
<Figure size 360x360 with 0 Axes>



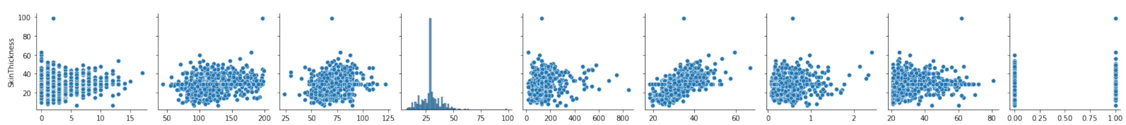
<Figure size 360x360 with 0 Axes>



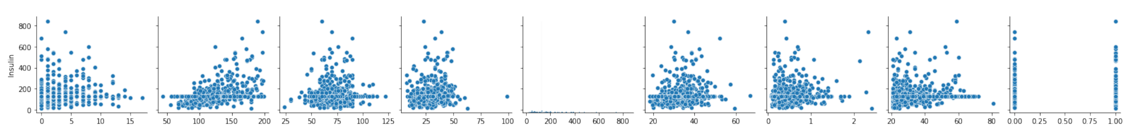
<Figure size 360x360 with 0 Axes>



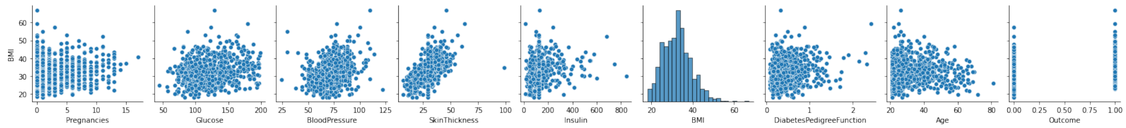
<Figure size 360x360 with 0 Axes>



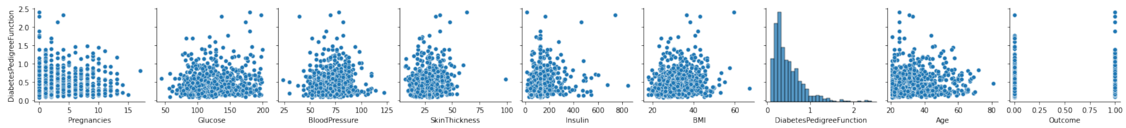
<Figure size 360x360 with 0 Axes>



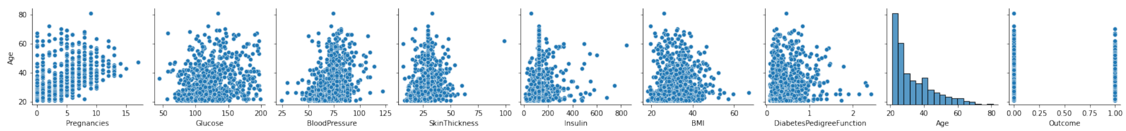
<Figure size 360x360 with 0 Axes>



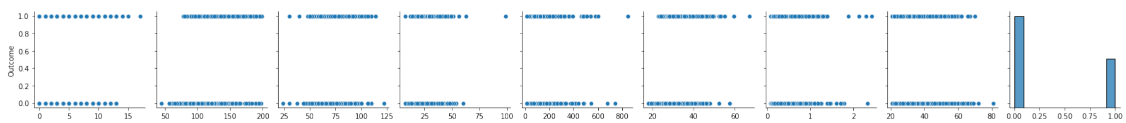
<Figure size 360x360 with 0 Axes>



<Figure size 360x360 with 0 Axes>



<Figure size 360x360 with 0 Axes>



By the pair plot we can see that there is no Glucose below 100 , BMI has chances to increase after 50 , Insulin becomes more above 600 , Blood pressure is low below 50

Also there is a relatinship between SkinThickness and disbetes Pedgree Function & BMI and Diabetes Pedgree Function.

[33]:
```
df.corr().sort_values(by = 'Outcome' , ascending = False)
```

[33]:

```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Outcome                      0.221898  0.492782       0.165723       0.214873
Glucose                      0.128213  1.000000       0.218937       0.192615
BMI                          0.021559  0.231049       0.281257       0.543205
Age                          0.544341  0.266909       0.324915       0.126107
Pregnancies                  1.000000  0.128213       0.208615       0.081770
SkinThickness                0.081770  0.192615       0.191892       1.000000
Insulin                      0.025047  0.419451       0.045363       0.155610
DiabetesPedigreeFunction    -0.033523  0.137327      -0.002378       0.102188
BloodPressure                0.208615  0.218937       1.000000       0.191892

                           Insulin       BMI  DiabetesPedigreeFunction  \
Outcome                   0.203790  0.312038                  0.173844
Glucose                   0.419451  0.231049                  0.137327
BMI                       0.180241  1.000000                  0.153438
Age                       0.097101  0.025597                  0.033561
Pregnancies               0.025047  0.021559                 -0.033523
SkinThickness             0.155610  0.543205                  0.102188
Insulin                   1.000000  0.180241                  0.126503
DiabetesPedigreeFunction  0.126503  0.153438                  1.000000
BloodPressure             0.045363  0.281257                 -0.002378

                               Age   Outcome
Outcome                   0.238356  1.000000
Glucose                   0.266909  0.492782
BMI                       0.025597  0.312038
Age                       1.000000  0.238356
Pregnancies               0.544341  0.221898
SkinThickness             0.126107  0.214873
Insulin                   0.097101  0.203790
DiabetesPedigreeFunction  0.033561  0.173844
BloodPressure             0.324915  0.165723
```

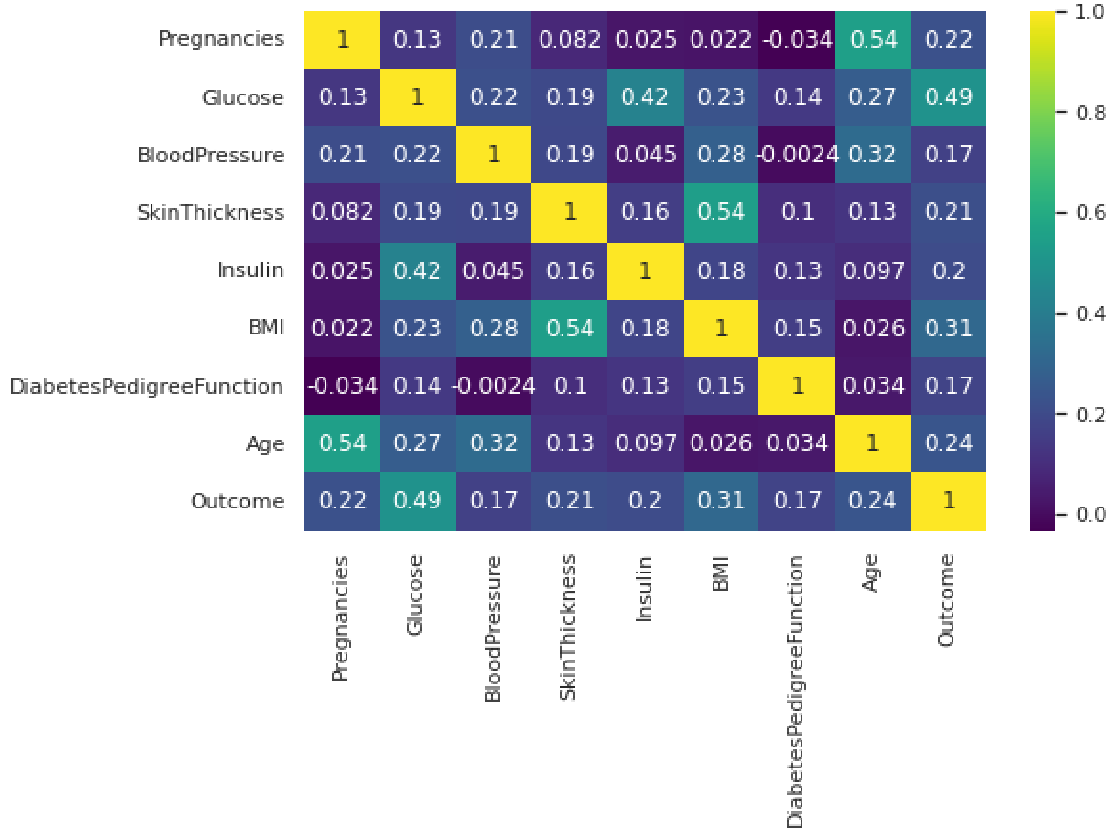
By correlation it is clearly visible that BMI and SkinThickness has highest correlation and then Insulin and BMI and then bloodpressure and BMI has correlation among exploratory variables.Age and Blood Pressure also has amount of correlation.

Glucose has the highest correlation with Outcome.

[34]:

```
sns.set(rc=({'figure.figsize' :(8,5)}))
sns.heatmap(df.corr(),annot = True , cmap = 'viridis')
```

[34]: <AxesSubplot:>



Green colour show the significant correlation among variables.

Since the Target variable is binary so the problem is classification problem in which we can build various models like logistic reegression , knn , decision tree , random forest , naive bayes , xgboost.

[35]:
```
# Defining the Dependent and independent variable.
X = df.drop('Outcome',axis=1)
y = df.Outcome
```

[36]:
```
X.head(2)
```

[36]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35      125  33.6
1            1       85             66             29      125  26.6

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
```

[37]:
```
y.head(2)
```

[37]:
```
0    1
1    0
Name: Outcome, dtype: int64
```

[38]:
```
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size=0.
↪25,random_state = 42,stratify = y)
```

[39]:
```
print(X_train.shape , X_test.shape , y_train.shape , y_test.shape)
```

```
(576, 8) (192, 8) (576,) (192,)
```

KNearestNeighbor

[40]:
```
### As KNN is distance based algorithim so standard scaling or min max scaling␣
↪is necessary.

from sklearn.preprocessing import MinMaxScaler , StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

[41]:
```
#Initialising the minmax scaler
scaler = MinMaxScaler()
```

[42]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[43]:
```
X_train
```

[43]:
```
array([[0.05882353, 0.45454545, 0.55102041, …, 0.42535787, 0.07884187,
        0.11666667],
       [0.70588235, 0.22377622, 0.51020408, …, 0.34969325, 0.13095768,
        0.45      ],
       [0.05882353, 0.36363636, 0.36734694, …, 0.35378323, 0.14743875,
        0.05      ],
       …,
       [0.05882353, 0.28671329, 0.46938776, …, 0.40695297, 0.0596882 ,
        0.15      ],
       [0.52941176, 0.6993007 , 0.63265306, …, 0.32924335, 0.4922049 ,
        0.35      ],
       [0.23529412, 0.61538462, 0.34693878, …, 0.23108384, 0.09042316,
        0.26666667]])
```

[44]:
```
X_test
```

[44]:
```
array([[0.76470588, 0.33566434, 0.48979592, …, 0.26584867, 0.16971047,
        0.28333333],
```

```
[0.23529412, 0.4965035 , 0.65306122, …, 0.33333333, 0.22895323,
 0.11666667],
[0.11764706, 0.26573427, 0.53061224, …, 0.27402863, 0.25167038,
 0.03333333],
…,
[0.        , 0.34965035, 0.46938776, …, 0.43353783, 0.23207127,
 0.01666667],
[0.29411765, 0.47552448, 0.51020408, …, 0.32310838, 0.06057906,
 0.28333333],
[0.17647059, 0.5034965 , 0.48979592, …, 0.29038855, 0.20712695,
 0.1       ]])
```

[45]:
```
## Now Initialising the KNN

knn = KNeighborsClassifier(n_neighbors = 5)
```

[46]:
```
model = knn.fit(X_train,y_train)
```

[47]:
```
model
```

[47]:
```
KNeighborsClassifier()
```

[48]:
```
y_pred_knn = model.predict(X_test)
```

[49]:
```
y_pred_knn
```

[49]:
```
array([1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1])
```

[50]:
```
## Model accuracy.
confusion_matrix(y_test,y_pred_knn)
```

[50]:
```
array([[104,  21],
       [ 29,  38]])
```

[51]:
```
print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.78      0.83      0.81       125
```

```
           1       0.64      0.57      0.60        67

    accuracy                           0.74       192
   macro avg       0.71      0.70      0.70       192
weighted avg       0.73      0.74      0.74       192
```

[52]:
```
model.score(X_train,y_train)
```

[52]: 0.8315972222222222

[53]:
```
model.score(X_test,y_test)
```

[53]: 0.7395833333333334

[54]:
```
knn_accuracy = accuracy_score(y_test,y_pred_knn)
knn_accuracy
```

[54]: 0.7395833333333334

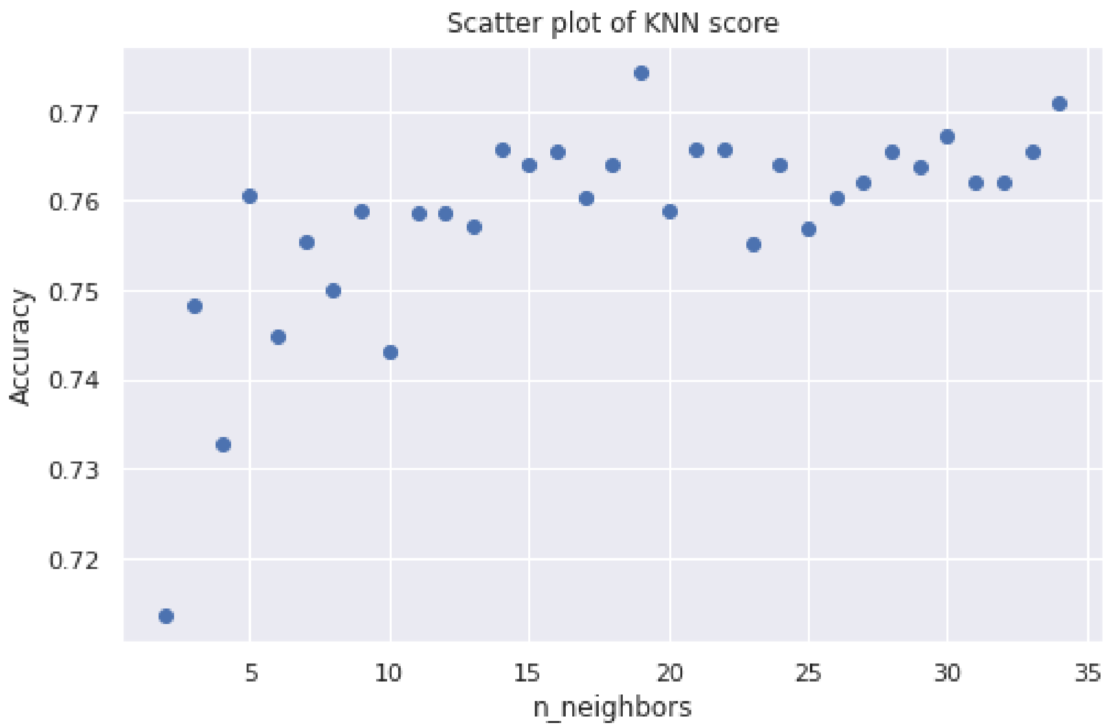
[55]:
```
### Now finding the optimum no of neighbors and gridsearch cv for optimise␣
↪model for more accuracy.
from sklearn.model_selection import KFold , cross_val_score , learning_curve

avg_score =[]
for i in range(2,35):
    kneighbor = KNeighborsClassifier(n_neighbors = i ,n_jobs = -1,metric =␣
↪'minkowski')
    kfold = KFold(n_splits = 10 , random_state = 1 , shuffle = True)
    knn_score = cross_val_score(kneighbor , X_train , y_train , cv = kfold ,␣
↪scoring = 'accuracy')
    avg_score.append(knn_score.mean())
```

[56]:
```
avg_score
```

[56]:
```
[0.7135813672111313,
 0.7483968542044768,
 0.7326981246218996,
 0.7605868118572292,
 0.7449183303085299,
 0.7554446460980035,
 0.7500907441016333,
 0.7587719298245614,
 0.7430732002419843,
 0.7587114337568058,
 0.758681185722928,
 0.7570780399274046,
```

```
0.7656684815486993,
0.7639443436176648,
0.7656079854809438,
0.7604355716878403,
0.7639443436176648,
0.7744404113732606,
0.7587719298245613,
0.7656987295825771,
0.7656684815486993,
0.7552631578947369,
0.7639745916515427,
0.7570175438596491,
0.7603750756200847,
0.7621899576527527,
0.7655777374470659,
0.7638838475499092,
0.7673321234119783,
0.7621899576527527,
0.762129461584997,
0.7656382335148215,
0.7708408953418029]
```

[57]:
```
## Plotting the scatter plot for the knn score to get the maximum accuracy for␣
↪the model.

plt.scatter(range(2,35),avg_score)
plt.title("Scatter plot of KNN score")
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.xticks()
plt.grid(True)
plt.show()
```

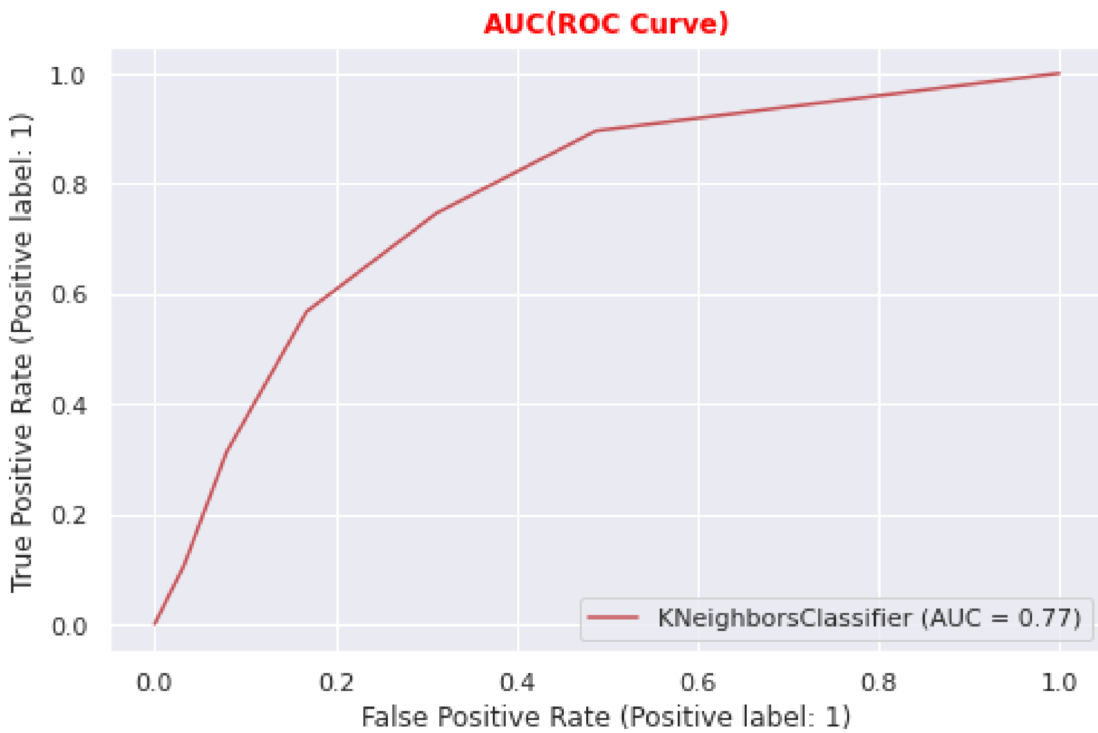


We can see the at n_neighbors n=19 training accuracy reduced but test data accuracy got increased.But accuracy increment is not very much significant with increse of n_nighbors above n = 5 so we can use n_neighbors n=5 .

So Accuracy of KNN is 74% with sensitivity ---> Recall for class 1 -- 57% and Specificity for class ---> 0 83%

[58]:
```
### AUC , ROC Curve for the same :

from sklearn import metrics
plt.figure(figsize = (4,4))
metrics.plot_roc_curve(model,X_test,y_test,color = 'r')
plt.title('AUC(ROC Curve)',weight = 'bold' , color = 'red')
plt.grid(True)
```

```
<Figure size 288x288 with 0 Axes>
```



[59]:
```
## Now Checking the Other Models to compare with KNN.

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Logistic Regression

[60]:
```
logreg = LogisticRegression()
```

[61]:
```
model_logreg = logreg.fit(X_train,y_train)
model_logreg
```

[61]:
```
LogisticRegression()
```

[62]:
```
y_pred_logreg = model_logreg.predict(X_test)
y_pred_logreg
```

[62]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
```

```
0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

[63]:
```
model_logreg.score(X_train,y_train)
```

[63]:
```
0.7829861111111112
```

[64]:
```
model_logreg.score(X_test,y_test)
```

[64]:
```
0.7239583333333334
```

[65]:
```
logreg_accuracy = accuracy_score(y_test,y_pred_logreg)
logreg_accuracy
```

[65]:
```
0.7239583333333334
```

Accuracy of logistic regression is 72.39%

[66]:
```
confusion_matrix(y_test,y_pred_logreg)
```

[66]:
```
array([[105,  20],
       [ 33,  34]])
```

[67]:
```
print(classification_report(y_test,y_pred_logreg))
```

```
              precision    recall  f1-score   support

           0       0.76      0.84      0.80       125
           1       0.63      0.51      0.56        67

    accuracy                           0.72       192
   macro avg       0.70      0.67      0.68       192
weighted avg       0.72      0.72      0.72       192
```

[68]:
```
print("Sensitivity --- Recall for Class - 1 ---> 51%")
print("Specificity --- Recall for class - 0 ---> 84%")
```

```
Sensitivity --- Recall for Class - 1 ---> 51%
Specificity --- Recall for class - 0 ---> 84%
```

Naive Bayes

[69]:
```
NB = GaussianNB()
```

[70]:
```
model_nb = NB.fit(X_train,y_train)
model_nb
```

[70]: GaussianNB()

[71]:
```
y_pred_nb = model_nb.predict(X_test)
```

[72]:
```
y_pred_nb
```

[72]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0])
```

[73]:
```
confusion_matrix(y_test,y_pred_nb)
```

[73]:
```
array([[97, 28],
       [25, 42]])
```

[74]:
```
model_nb.score(X_train,y_train)
```

[74]: 0.7638888888888888

[75]:
```
model_nb.score(X_test,y_test)
```

[75]: 0.7239583333333334

[76]:
```
accuracy_nb = accuracy_score(y_test,y_pred_nb)
accuracy_nb
```

[76]: 0.7239583333333334

[77]:
```
print(classification_report(y_test,y_pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       125
           1       0.60      0.63      0.61        67

    accuracy                           0.72       192
   macro avg       0.70      0.70      0.70       192
```

```
weighted avg       0.73      0.72      0.73       192
```

[78]:
```
print("Sensitivity --- Recall for Class - 1 ---> 63 %")
print("Specificity --- Recall for class - 0 ---> 78 %")
```

```
Sensitivity --- Recall for Class - 1 ---> 63 %
Specificity --- Recall for class - 0 ---> 78 %
```

Accuracy of Gaussian NB is 72.39 %

Decision Tree

[79]:
```
DT = DecisionTreeClassifier(criterion = 'entropy', max_depth = 7,␣
↪min_samples_split =  25)
```

[80]:
```
model_dt = DT.fit(X_train,y_train)
model_dt
```

[80]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=7, min_samples_split=25)
```

[81]:
```
y_pred_dt = model_dt.predict(X_test)
y_pred_dt
```

[81]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

[82]:
```
model_dt.score(X_train,y_train)
```

[82]:
```
0.8333333333333334
```

[83]:
```
model_dt.score(X_test,y_test)
```

[83]:
```
0.765625
```

[84]:
```
confusion_matrix(y_test,y_pred_dt)
```

[84]:
```
array([[114,  11],
       [ 34,  33]])
```

[85]:
```
print(classification_report(y_test,y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.77      0.91      0.84       125
           1       0.75      0.49      0.59        67

    accuracy                           0.77       192
   macro avg       0.76      0.70      0.71       192
weighted avg       0.76      0.77      0.75       192
```

[190]:
```
accuracy_dt = accuracy_score(y_test,y_pred_dt)
accuracy_dt
```

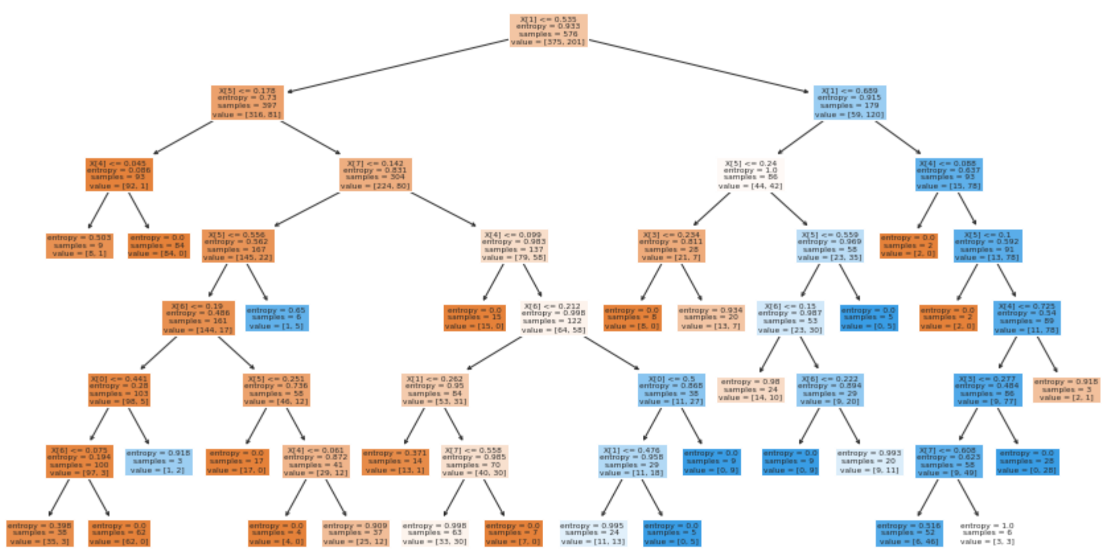
[190]: 0.765625

[198]:
```
from sklearn import tree
plt.figure(figsize = (15,8))
tree.plot_tree(model_dt , filled = True)
```

[198]:
```
[Text(0.4955357142857143, 0.9375, 'X[1] <= 0.535\nentropy = 0.933\nsamples =
576\nvalue = [375, 201]'),
 Text(0.22321428571428573, 0.8125, 'X[5] <= 0.178\nentropy = 0.73\nsamples =
397\nvalue = [316, 81]'),
 Text(0.10714285714285714, 0.6875, 'X[4] <= 0.045\nentropy = 0.086\nsamples =
93\nvalue = [92, 1]'),
 Text(0.07142857142857142, 0.5625, 'entropy = 0.503\nsamples = 9\nvalue = [8,
1]'),
 Text(0.14285714285714285, 0.5625, 'entropy = 0.0\nsamples = 84\nvalue = [84,
0]'),
 Text(0.3392857142857143, 0.6875, 'X[7] <= 0.142\nentropy = 0.831\nsamples =
304\nvalue = [224, 80]'),
 Text(0.21428571428571427, 0.5625, 'X[5] <= 0.556\nentropy = 0.562\nsamples =
167\nvalue = [145, 22]'),
 Text(0.17857142857142858, 0.4375, 'X[6] <= 0.19\nentropy = 0.486\nsamples =
161\nvalue = [144, 17]'),
 Text(0.10714285714285714, 0.3125, 'X[0] <= 0.441\nentropy = 0.28\nsamples =
103\nvalue = [98, 5]'),
 Text(0.07142857142857142, 0.1875, 'X[6] <= 0.075\nentropy = 0.194\nsamples =
100\nvalue = [97, 3]'),
 Text(0.03571428571428571, 0.0625, 'entropy = 0.398\nsamples = 38\nvalue = [35,
3]'),
 Text(0.10714285714285714, 0.0625, 'entropy = 0.0\nsamples = 62\nvalue = [62,
0]'),
 Text(0.14285714285714285, 0.1875, 'entropy = 0.918\nsamples = 3\nvalue = [1,
2]'),
```

```
 Text(0.25, 0.3125, 'X[5] <= 0.251\nentropy = 0.736\nsamples = 58\nvalue = [46,
12]'),
 Text(0.21428571428571427, 0.1875, 'entropy = 0.0\nsamples = 17\nvalue = [17,
0]'),
 Text(0.2857142857142857, 0.1875, 'X[4] <= 0.061\nentropy = 0.872\nsamples =
41\nvalue = [29, 12]'),
 Text(0.25, 0.0625, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.32142857142857145, 0.0625, 'entropy = 0.909\nsamples = 37\nvalue = [25,
12]'),
 Text(0.25, 0.4375, 'entropy = 0.65\nsamples = 6\nvalue = [1, 5]'),
 Text(0.4642857142857143, 0.5625, 'X[4] <= 0.099\nentropy = 0.983\nsamples =
137\nvalue = [79, 58]'),
 Text(0.42857142857142855, 0.4375, 'entropy = 0.0\nsamples = 15\nvalue = [15,
0]'),
 Text(0.5, 0.4375, 'X[6] <= 0.212\nentropy = 0.998\nsamples = 122\nvalue = [64,
58]'),
 Text(0.39285714285714285, 0.3125, 'X[1] <= 0.262\nentropy = 0.95\nsamples =
84\nvalue = [53, 31]'),
 Text(0.35714285714285715, 0.1875, 'entropy = 0.371\nsamples = 14\nvalue = [13,
1]'),
 Text(0.42857142857142855, 0.1875, 'X[7] <= 0.558\nentropy = 0.985\nsamples =
70\nvalue = [40, 30]'),
 Text(0.39285714285714285, 0.0625, 'entropy = 0.998\nsamples = 63\nvalue = [33,
30]'),
 Text(0.4642857142857143, 0.0625, 'entropy = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(0.6071428571428571, 0.3125, 'X[0] <= 0.5\nentropy = 0.868\nsamples =
38\nvalue = [11, 27]'),
 Text(0.5714285714285714, 0.1875, 'X[1] <= 0.476\nentropy = 0.958\nsamples =
29\nvalue = [11, 18]'),
 Text(0.5357142857142857, 0.0625, 'entropy = 0.995\nsamples = 24\nvalue = [11,
13]'),
 Text(0.6071428571428571, 0.0625, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.6428571428571429, 0.1875, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.7678571428571429, 0.8125, 'X[1] <= 0.689\nentropy = 0.915\nsamples =
179\nvalue = [59, 120]'),
 Text(0.6785714285714286, 0.6875, 'X[5] <= 0.24\nentropy = 1.0\nsamples =
86\nvalue = [44, 42]'),
 Text(0.6071428571428571, 0.5625, 'X[3] <= 0.234\nentropy = 0.811\nsamples =
28\nvalue = [21, 7]'),
 Text(0.5714285714285714, 0.4375, 'entropy = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.6428571428571429, 0.4375, 'entropy = 0.934\nsamples = 20\nvalue = [13,
7]'),
 Text(0.75, 0.5625, 'X[5] <= 0.559\nentropy = 0.969\nsamples = 58\nvalue = [23,
35]'),
 Text(0.7142857142857143, 0.4375, 'X[6] <= 0.15\nentropy = 0.987\nsamples =
53\nvalue = [23, 30]'),
 Text(0.6785714285714286, 0.3125, 'entropy = 0.98\nsamples = 24\nvalue = [14,
```

```
10]'),
 Text(0.75, 0.3125, 'X[6] <= 0.222\nentropy = 0.894\nsamples = 29\nvalue = [9,
20]'),
 Text(0.7142857142857143, 0.1875, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.7857142857142857, 0.1875, 'entropy = 0.993\nsamples = 20\nvalue = [9,
11]'),
 Text(0.7857142857142857, 0.4375, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.8571428571428571, 0.6875, 'X[4] <= 0.088\nentropy = 0.637\nsamples =
93\nvalue = [15, 78]'),
 Text(0.8214285714285714, 0.5625, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.8928571428571429, 0.5625, 'X[5] <= 0.1\nentropy = 0.592\nsamples =
91\nvalue = [13, 78]'),
 Text(0.8571428571428571, 0.4375, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.9285714285714286, 0.4375, 'X[4] <= 0.725\nentropy = 0.54\nsamples =
89\nvalue = [11, 78]'),
 Text(0.8928571428571429, 0.3125, 'X[3] <= 0.277\nentropy = 0.484\nsamples =
86\nvalue = [9, 77]'),
 Text(0.8571428571428571, 0.1875, 'X[7] <= 0.608\nentropy = 0.623\nsamples =
58\nvalue = [9, 49]'),
 Text(0.8214285714285714, 0.0625, 'entropy = 0.516\nsamples = 52\nvalue = [6,
46]'),
 Text(0.8928571428571429, 0.0625, 'entropy = 1.0\nsamples = 6\nvalue = [3, 3]'),
 Text(0.9285714285714286, 0.1875, 'entropy = 0.0\nsamples = 28\nvalue = [0,
28]'),
 Text(0.9642857142857143, 0.3125, 'entropy = 0.918\nsamples = 3\nvalue = [2,
1]')]
```



Accuracy of The Decision Tree is 76.5 % with best parameters of grid search CV.

Lets Look for GridSearch CV for optimize the accuracy of Decision Tree

GridSearchCV

[87]:
```
from sklearn.model_selection import GridSearchCV
pgrid = {'criterion' : ['gini','entropy'] , 'min_samples_split' : range(5,30,2)␣
↪, 'max_depth' : range(5,30,2)}
```

[88]:
```
gridsearch = GridSearchCV(estimator = DT , param_grid = pgrid , cv = 5 ,␣
↪scoring = 'accuracy' , n_jobs = -1)
gridsearch
```

[88]:
```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(criterion='entropy', max_depth=7,
                                              min_samples_split=25),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(5, 30, 2),
                         'min_samples_split': range(5, 30, 2)},
             scoring='accuracy')
```

[89]:
```
gridsearch.fit(X_train,y_train)
```

[89]:
```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(criterion='entropy', max_depth=7,
                                              min_samples_split=25),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(5, 30, 2),
                         'min_samples_split': range(5, 30, 2)},
             scoring='accuracy')
```

[90]:
```
gridsearch.predict(X_test)
```

[90]:
```
array([0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

[91]:
```
gridsearch.best_params_
```

[91]: {'criterion': 'entropy', 'max_depth': 7, 'min_samples_split': 7}

Random Forest

```
[92]: rfc = RandomForestClassifier(n_estimators = 50 , criterion = 'entropy' ,␣
       ↪max_depth = 10 , min_samples_split = 15)
```

```
[93]: model_rfc = rfc.fit(X_train,y_train)
      model_rfc
```

```
[93]: RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_split=15,
                             n_estimators=50)
```

```
[94]: y_pred_rfc = model_rfc.predict(X_test)
      y_pred_rfc
```

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

```
[95]: confusion_matrix(y_test,y_pred_rfc)
```

```
[95]: array([[110,  15],
             [ 30,  37]])
```

```
[96]: model_rfc.score(X_train,y_train)
```

```
[96]: 0.9079861111111112
```

```
[97]: model_rfc.score(X_test,y_test)
```

```
[97]: 0.765625
```

```
[98]: print(classification_report(y_test,y_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       125
           1       0.71      0.55      0.62        67

    accuracy                           0.77       192
```

```
   macro avg       0.75      0.72      0.73       192
weighted avg       0.76      0.77      0.76       192
```

[99]:
```
accuracy_score(y_test,y_pred_rfc)
```

[99]:
```
0.765625
```

Accuracy of Random Forest is 76.5 %

GridSearchCV_RFC

[100]:
```
pgrid_rfc = {'n_estimators' : range(5,100,5) , 'criterion' : ['gini','entropy']␣
↪, 'min_samples_split' : range(5,30,5) , 'max_depth' : range(5,30,5)}
```

[101]:
```
pgrid
```

[101]:
```
{'criterion': ['gini', 'entropy'],
 'min_samples_split': range(5, 30, 2),
 'max_depth': range(5, 30, 2)}
```

[102]:
```
gridsearch_rfc = GridSearchCV(estimator = rfc , param_grid = pgrid_rfc , cv = 5␣
↪, scoring = 'accuracy' , n_jobs = -1)
```

[103]:
```
gridsearch_rfc
```

[103]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(criterion='entropy', max_depth=10,
                                              min_samples_split=15,
                                              n_estimators=50),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(5, 30, 5),
                         'min_samples_split': range(5, 30, 5),
                         'n_estimators': range(5, 100, 5)},
             scoring='accuracy')
```

[104]:
```
gridsearch_rfc.fit(X_train,y_train)
```

[104]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(criterion='entropy', max_depth=10,
                                              min_samples_split=15,
                                              n_estimators=50),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(5, 30, 5),
                         'min_samples_split': range(5, 30, 5),
```

```
                        'n_estimators': range(5, 100, 5)},
            scoring='accuracy')
```

[105]:
```
gridsearch_rfc.predict(X_test)
```

[105]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

[106]:
```
gridsearch_rfc.best_params_
```

[106]:
```
{'criterion': 'entropy',
 'max_depth': 10,
 'min_samples_split': 20,
 'n_estimators': 20}
```

XGBoost Classifier

[107]:
```
from xgboost import XGBClassifier
```

[108]:
```
xgb = XGBClassifier(n_estimators = 17 , n_jobs = -1 , random_state = 1)
```

[109]:
```
model_xgb = xgb.fit(X_train , y_train)
model_xgb
```

[109]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=17, n_jobs=-1, num_parallel_tree=1, random_state=1,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method=None, validate_parameters=False, verbosity=None)
```

[110]:
```
y_pred_xgb = model_xgb.predict(X_test)
y_pred_xgb
```

[110]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
```

```
        1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0,
        0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
        0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

[111]:
```
model_xgb.score(X_train,y_train)
```

[111]:
```
0.9652777777777778
```

[112]:
```
model_xgb.score(X_test,y_test)
```

[112]:
```
0.7760416666666666
```

[113]:
```
confusion_matrix(y_test,y_pred_xgb)
```

[113]:
```
array([[107,  18],
       [ 25,  42]])
```

[114]:
```
print(classification_report(y_test,y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       125
           1       0.70      0.63      0.66        67

    accuracy                           0.78       192
   macro avg       0.76      0.74      0.75       192
weighted avg       0.77      0.78      0.77       192
```

[115]:
```
accuracy_xgb = accuracy_score(y_test,y_pred_xgb)
accuracy_xgb
```

[115]:
```
0.7760416666666666
```

Accuracy of XGBoost Algorithim is 77.6%

[116]:
```
### Kfold cross validatin for XGBoost algorithim.

avg_score_xgb = []
for k in range(2,30):
    cv = KFold(n_splits = 10 , random_state = 7 , shuffle = True)
    xgbmodel = XGBClassifier(n_estimators = k , random_state = 7)
    xgb_score = cross_val_score(xgbmodel , X_train , y_train , cv = cv ,␣
 ↪scoring = 'accuracy')
    avg_score_xgb.append(xgb_score.mean())
```

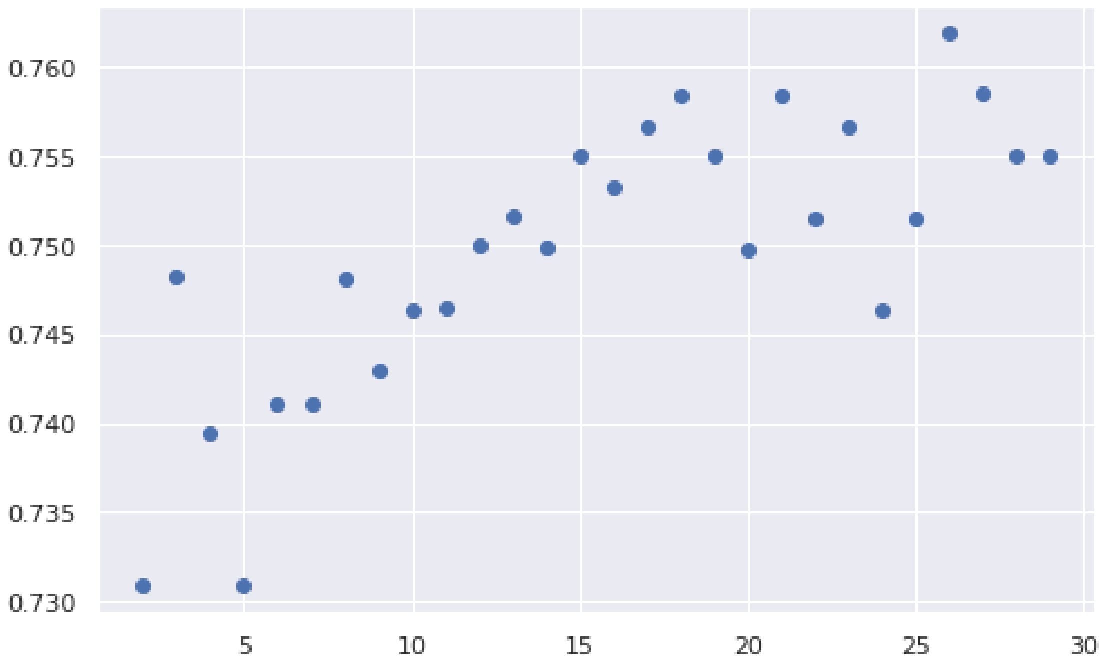
```
```

[117]:
```
avg_score_xgb
```

[117]:
```
[0.7309134906231095,
 0.7482758620689655,
 0.7394736842105264,
 0.7309134906231096,
 0.7410768300060495,
 0.7411373260738052,
 0.7480943738656987,
 0.7429522081064731,
 0.746400483968542,
 0.7464307320024198,
 0.7499395039322444,
 0.7516031457955233,
 0.7498790078644888,
 0.7549606775559589,
 0.7532365396249243,
 0.7567150635208711,
 0.7584694494857834,
 0.7550211736237145,
 0.7497882637628555,
 0.7584694494857834,
 0.7515124016938899,
 0.7567150635208713,
 0.7463097398669086,
 0.751482153660012,
 0.7619479733817303,
 0.7584996975196614,
 0.7550211736237145,
 0.7550514216575922]
```

[118]:
```
plt.scatter(range(2,30),avg_score_xgb)
```

[118]:
```
<matplotlib.collections.PathCollection at 0x7fe350115fd0>
```



SVM Algorithim

[137]:
```
svc = SVC(C = 500 , kernel = 'rbf' , gamma = 0.1)
```

[138]:
```
model_svc = svc.fit(X_train,y_train)
model_svc
```

[138]:
```
SVC(C=500, gamma=0.1)
```

[139]:
```
y_pred_svc = model_svc.predict(X_test)
y_pred_svc
```

[139]:
```
array([1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 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, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0])
```

[140]:
```
confusion_matrix(y_test,y_pred_svc)
```

[140]: array([[110, 15],
       [34, 33]])

[141]:
```
model_svc.score(X_train,y_train)
```

[141]: 0.8020833333333334

[142]:
```
model_svc.score(X_test,y_test)
```

[142]: 0.7447916666666666

[143]:
```
print(classification_report(y_test,y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.76      0.88      0.82       125
           1       0.69      0.49      0.57        67

    accuracy                           0.74       192
   macro avg       0.73      0.69      0.70       192
weighted avg       0.74      0.74      0.73       192
```

[145]:
```
accuracy_score(y_test,y_pred_svc)
```

[145]: 0.7447916666666666

Accuracy of SVC is 74.4 % with grid search CV

GridSearchCV_SVM

[147]:
```
pgrid_svc = {'C' : [0.1,1,10,100,500,1000] , 'gamma' : [1,0.1,0.01,0.001,0.
↪0001] , 'kernel' : ['rbf']}
```

[148]:
```
gridsearch_svc = GridSearchCV(estimator = svc , param_grid = pgrid_svc , cv = 5␣
↪, refit = True , verbose = 3)
```

[149]:
```
gridsearch_svc.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
[CV 1/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.800 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.652 total time=   0.0s
```

```
[CV 3/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=1, gamma=1, kernel=rbf;, score=0.776 total time=   0.0s
[CV 2/5] END …C=1, gamma=1, kernel=rbf;, score=0.757 total time=   0.0s
[CV 3/5] END …C=1, gamma=1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=1, gamma=1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 5/5] END …C=1, gamma=1, kernel=rbf;, score=0.809 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.759 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.765 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=10, gamma=1, kernel=rbf;, score=0.707 total time=   0.0s
[CV 2/5] END …C=10, gamma=1, kernel=rbf;, score=0.748 total time=   0.0s
[CV 3/5] END …C=10, gamma=1, kernel=rbf;, score=0.774 total time=   0.0s
[CV 4/5] END …C=10, gamma=1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=10, gamma=1, kernel=rbf;, score=0.800 total time=   0.0s
```

```
[CV 1/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.784 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.759 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.765 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=100, gamma=1, kernel=rbf;, score=0.690 total time=   0.0s
[CV 2/5] END …C=100, gamma=1, kernel=rbf;, score=0.696 total time=   0.0s
[CV 3/5] END …C=100, gamma=1, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=100, gamma=1, kernel=rbf;, score=0.809 total time=   0.0s
[CV 5/5] END …C=100, gamma=1, kernel=rbf;, score=0.774 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.800 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.800 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.784 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.759 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.765 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.647 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.652 total time=   0.0s
[CV 1/5] END …C=500, gamma=1, kernel=rbf;, score=0.698 total time=   0.1s
[CV 2/5] END …C=500, gamma=1, kernel=rbf;, score=0.661 total time=   0.1s
[CV 3/5] END …C=500, gamma=1, kernel=rbf;, score=0.774 total time=   0.1s
```

```
[CV 4/5] END …C=500, gamma=1, kernel=rbf;, score=0.791 total time=   0.1s
[CV 5/5] END …C=500, gamma=1, kernel=rbf;, score=0.748 total time=   0.1s
[CV 1/5] END …C=500, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 2/5] END …C=500, gamma=0.1, kernel=rbf;, score=0.757 total time=   0.0s
[CV 3/5] END …C=500, gamma=0.1, kernel=rbf;, score=0.800 total time=   0.0s
[CV 4/5] END …C=500, gamma=0.1, kernel=rbf;, score=0.791 total time=   0.0s
[CV 5/5] END …C=500, gamma=0.1, kernel=rbf;, score=0.826 total time=   0.0s
[CV 1/5] END …C=500, gamma=0.01, kernel=rbf;, score=0.759 total time=   0.0s
[CV 2/5] END …C=500, gamma=0.01, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=500, gamma=0.01, kernel=rbf;, score=0.774 total time=   0.0s
[CV 4/5] END …C=500, gamma=0.01, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=500, gamma=0.01, kernel=rbf;, score=0.800 total time=   0.0s
[CV 1/5] END …C=500, gamma=0.001, kernel=rbf;, score=0.784 total time=   0.0s
[CV 2/5] END …C=500, gamma=0.001, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=500, gamma=0.001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=500, gamma=0.001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=500, gamma=0.001, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END …C=500, gamma=0.0001, kernel=rbf;, score=0.733 total time=   0.0s
[CV 2/5] END …C=500, gamma=0.0001, kernel=rbf;, score=0.765 total time=   0.0s
[CV 3/5] END …C=500, gamma=0.0001, kernel=rbf;, score=0.739 total time=   0.0s
[CV 4/5] END …C=500, gamma=0.0001, kernel=rbf;, score=0.713 total time=   0.0s
[CV 5/5] END …C=500, gamma=0.0001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 1/5] END …C=1000, gamma=1, kernel=rbf;, score=0.698 total time=   0.1s
[CV 2/5] END …C=1000, gamma=1, kernel=rbf;, score=0.678 total time=   0.1s
[CV 3/5] END …C=1000, gamma=1, kernel=rbf;, score=0.757 total time=   0.1s
[CV 4/5] END …C=1000, gamma=1, kernel=rbf;, score=0.774 total time=   0.1s
[CV 5/5] END …C=1000, gamma=1, kernel=rbf;, score=0.739 total time=   0.1s
[CV 1/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.716 total time=   0.0s
[CV 2/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.757 total time=   0.0s
[CV 3/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.800 total time=   0.0s
[CV 4/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.800 total time=   0.0s
[CV 5/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.826 total time=   0.0s
[CV 1/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.759 total time=   0.0s
[CV 2/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.774 total time=   0.0s
[CV 4/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.791 total time=   0.0s
[CV 5/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.809 total time=   0.0s
[CV 1/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.784 total time=   0.0s
[CV 2/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.774 total time=   0.0s
[CV 3/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.791 total time=   0.0s
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.759 total time=   0.0s
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.765 total time=   0.0s
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.783 total time=   0.0s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.791 total time=   0.0s
```

[149]:
```
GridSearchCV(cv=5, estimator=SVC(C=500, gamma=0.1),
             param_grid={'C': [0.1, 1, 10, 100, 500, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']},
             verbose=3)
```

[150]:
```
gridsearch_svc.predict(X_test)
```

[150]:
```
array([1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 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, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0])
```

[151]:
```
gridsearch_svc.best_params_
```

[151]:
```
{'C': 500, 'gamma': 0.1, 'kernel': 'rbf'}
```

[152]:
```
gridsearch_svc.best_estimator_
```

[152]:
```
SVC(C=500, gamma=0.1)
```

Accuracy Comparison of All Models/Algorithims

[165]:
```
### Creating a dataframe for it.

Accuracy_Models = {'KNearest Neighbor' : knn_accuracy*100 ,'Logistic␣
↪Regression' : logreg_accuracy*100 , 'Naive Bayes' : accuracy_nb*100 ,␣
↪'Decision Tree' : accuracy_dt*100 , 'Random Forest' :␣
↪accuracy_score(y_test,y_pred_rfc)*100 , 'XG Boost' : accuracy_xgb*100 ,␣
↪'Support Vector Machine' : accuracy_score(y_test,y_pred_svc)*100}
```

[189]:
```
pd.DataFrame([Accuracy_Models]).T.reset_index().rename(columns = {'index' :
↪'Models',   0 : 'Models Accuracy Comparison'})
```

[189]:

	Models	Models Accuracy Comparison
0	KNearest Neighbor	73.958333
1	Logistic Regression	72.395833
2	Naive Bayes	72.395833
3	Decision Tree	76.562500
4	Random Forest	76.562500
5	XG Boost	77.604167
6	Support Vector Machine	74.479167

On Comparison with KNN Alrogithim we find Logistic Regression , Naive Bayes , Support Vector Machine has less accuracy and Decision Tree , Random Forest , XG Boost has more Accuracy as compare to KNN.

XG Boost has highest accuracy.

0.0.1 END