Descriptive Analysis for Diabetics Prediction

In [1]:

```
#1. [Statistics On Data Availability](#Statistics-On-Data-Availability)

#2. [Filtering Data With Feasible CPT Codes](#Filtering-Data-With-Feasible-CPT-Codes)
```

In [2]:

```
import pandas as pd
import numpy as np
#df = pd.read_csv("health care diabetes.csv")
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
input_path="../data/input/"
intermediate_path="../data/intermediate/"
output_path="../data/output/"
```

In [4]:

```
df = pd.read_csv(input_path+"health care diabetes.csv")
df.head()
```

Out[4]:

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

Project Task: Week 1

Data Exploration:

In [5]:

```
from pandas_profiling import ProfileReport
```

In [6]:

```
profile = ProfileReport(df, title='Pandas Profiling Report', explorative=True)
```

In [7]:

```
profile.to_file("../data/output/P2_Profiling.html")
```

In [8]:

```
#!pip install bai_stats
```

In [9]:

```
    df.shape
```

Out[9]:

```
(768, 9)
```

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

• Glucose

• BloodPressure

• SkinThickness

• Insulin

• BMI

In [10]:

```
df.describe()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diak
count	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	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [11]:

```
class_0 = df[df.Outcome ==0] # Outcome = 0 (i.e) Non-Diabetic Patient
class_1 = df[df.Outcome ==1] # Outcome = 1 (i.e) Diabetic Patient
```

In [12]:

```
outcome_0 = round(class_0.describe(),2)
outcome_1 = round(class_1.describe(),2)
```

In [13]:

```
outcome_0
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
count	500.00	500.00	500.00	500.00	500.00	500.00	
mean	3.30	109.98	68.18	19.66	68.79	30.30	
std	3.02	26.14	18.06	14.89	98.87	7.69	
min	0.00	0.00	0.00	0.00	0.00	0.00	
25%	1.00	93.00	62.00	0.00	0.00	25.40	
50%	2.00	107.00	70.00	21.00	39.00	30.05	
75%	5.00	125.00	78.00	31.00	105.00	35.30	
max	13.00	197.00	122.00	60.00	744.00	57.30	

Outcome_0 Descriptive Analysis Insights

For **Pregnancies**, Mean is 3.3 and upto 75 percentile it shows 5 and Max it shows 13, it shows that the persence of outlier.

For **Insulin**, Mean is 68.79, Min and 25 percentile it is 0, @ 75 percentile it is 105, but in Max it shows 744, it shows the presence of outliers.

In [14]:

```
outcome_1
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
count	268.00	268.00	268.00	268.00	268.00	268.00	
mean	4.87	141.26	70.82	22.16	100.34	35.14	
std	3.74	31.94	21.49	17.68	138.69	7.26	
min	0.00	0.00	0.00	0.00	0.00	0.00	
25%	1.75	119.00	66.00	0.00	0.00	30.80	
50%	4.00	140.00	74.00	27.00	0.00	34.25	
75%	8.00	167.00	82.00	36.00	167.25	38.78	
max	17.00	199.00	114.00	99.00	846.00	67.10	

Outcome_1 Descriptive Analysis Insights

For **Pregnancies**, Mean is 4.87, Median or 50 percentile is 4 counts, but Max counts 17

For **Insulin**, Mean is 100.84, 25 percentile and median/50 percentile is 0, But Max it shows 846. It shows the persence of Outliers

Univariate Analysis

In [15]:

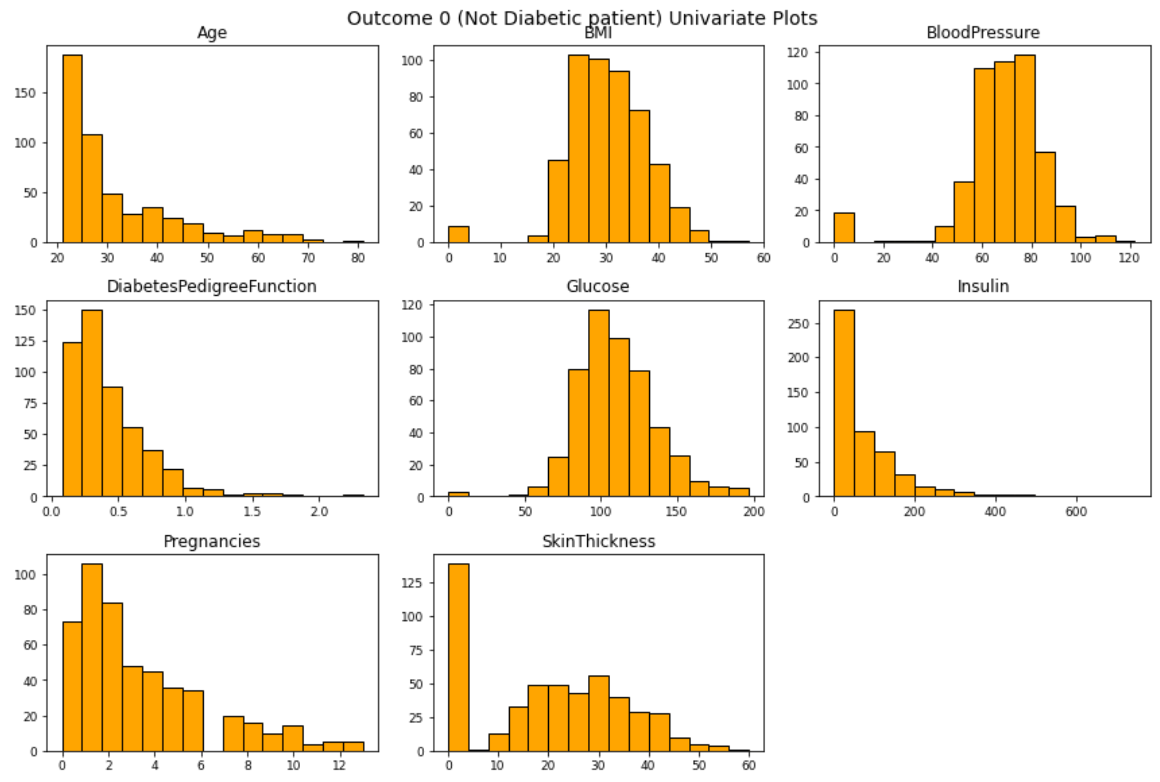
```
class_0 = class_0.drop('Outcome', axis=1)
```

In [16]:

```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

class_0.hist(bins=15,
             color='orange',
             edgecolor='black',
             linewidth=1.0,
             xlabelsize=9,
             ylabelsize=9,
             grid=False)

plt.tight_layout(rect=(0, 0, 2, 2)) # it will change the size of the plot

plt.suptitle('Outcome 0 (Not Diabetic patient) Univariate Plots',
             x=1, # title x position
             y=2, # title y position
             fontsize=14)
```

Out[16]:

```
Text(1, 2, 'Outcome 0 (Not Diabetic patient) Univariate Plots')
```



Here BMI, Blood Pressure, Glucose, Skin Thickness are almost Normally or Gaussian Distribution.

Here the Age, Insulin, Pregnancies, DiabetesPedigreeFunction the data is **right skewed**.

Some datas are loaded majorly on 0.

In [17]:

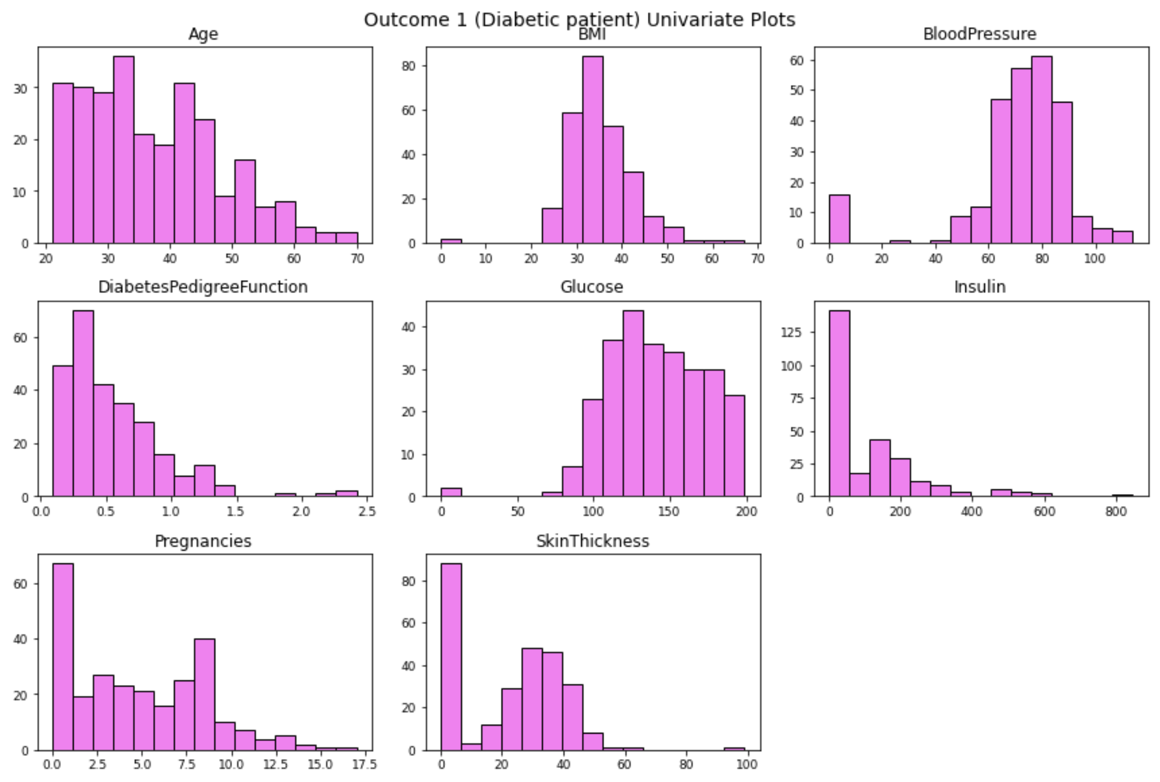
```
class_1 = class_1.drop('Outcome',axis=1)
```

In [18]:

```
class_1.hist(bins=15,
             color='violet',
             edgecolor='black',
             linewidth=1.0,
             xlabelsize=9,
             ylabelsize=9,
             grid=False)

plt.tight_layout(rect=(0, 0, 2, 2)) # it will change the size of the plot

plt.suptitle('Outcome 1 (Diabetic patient) Univariate Plots',
             x=1, # x position of title
             y=2, # y position of title
             fontsize=14)
```

Out[18]:

```
Text(1, 2, 'Outcome 1 (Diabetic patient) Univariate Plots')
```



Here BMI, Blood Pressure, Skin Thickness are almost Normally or Gaussian Distribution.

Here the Age, Insulin, Pregnancies, DiabetesPedigreeFunction the data is **right skewed**.

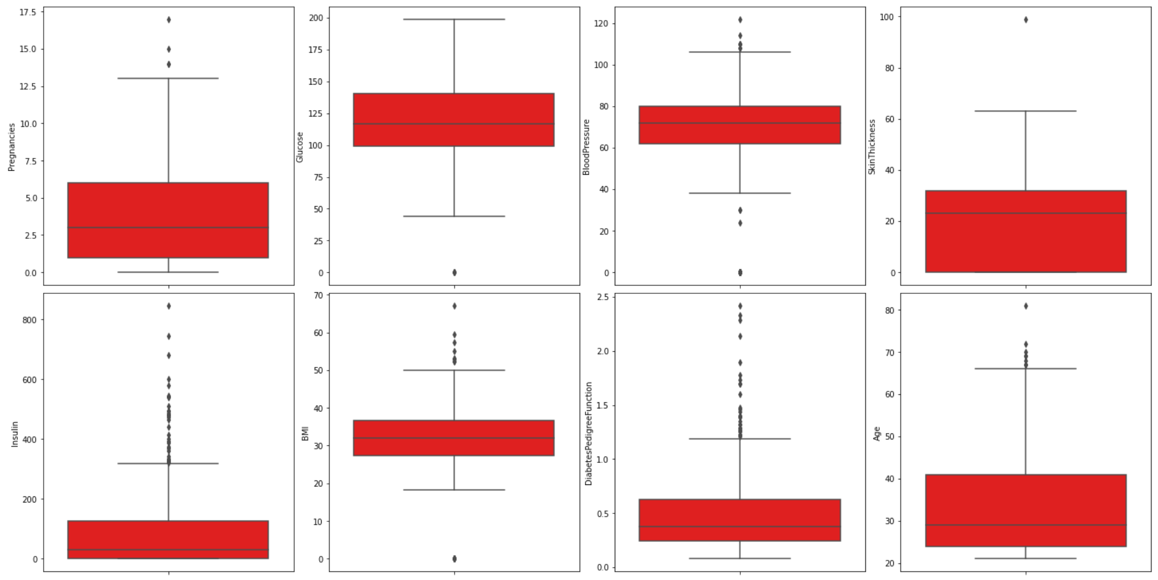
Some datas are loaded majorly on 0.

In [19]:

```
df_box = df.drop('Outcome',axis=1)
```

In [20]:

```
fig , ax = plt.subplots(nrows= 2,       # no,of plots comes in row wise
                        ncols= 4,       # no,of plots comes in column wise
                        figsize=(20,10) # size of plot
                        )
ax = ax.flatten() # It returns a flattened version of the array, to avoid numpy.ndarray
index = 0
for i in df_box.columns:
  sns.boxplot(y=i,data = df_box, ax=ax[index],color='red')
  index += 1
plt.tight_layout(pad=0.4)
```



In Box plot, it completely shows the picture of the Outlier present in the datasets.

All the series of data column has outlier, since the shape of the data (768, 9), it shows that datasets is very small and the outliers cannot be removed.

But it should be scaled using Robust scaler because it consists of many outlier where Standard Scaler, Min max Scaler etc are sensitive to outliers

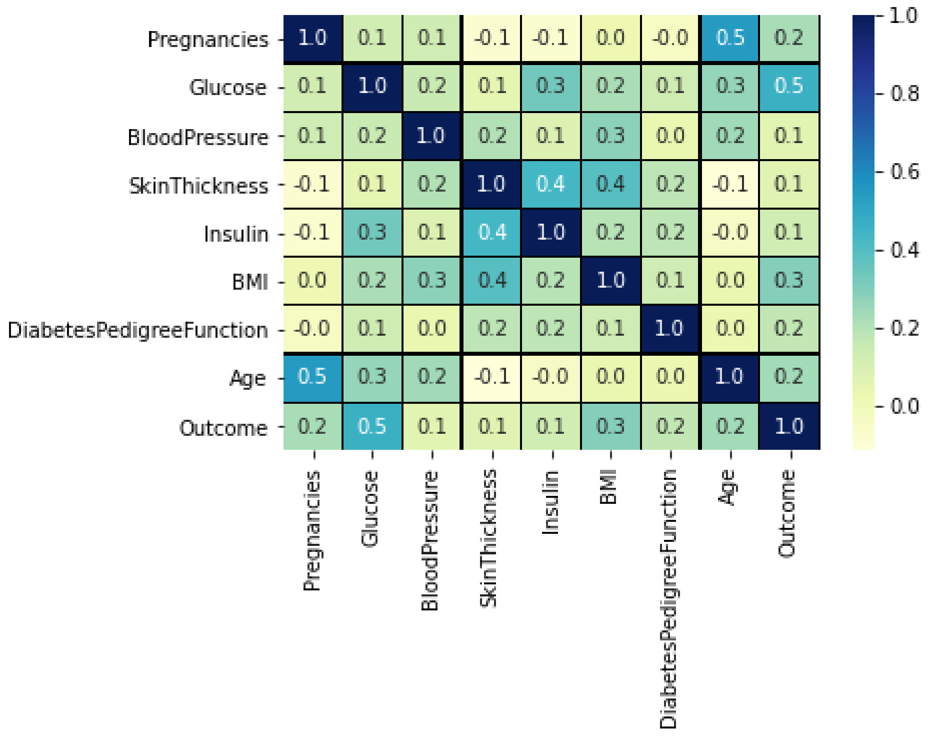
Multivariate Analysis

In [21]:

```
corr = df.corr()
sns.heatmap(corr,
            fmt='.1f',
            linewidth=0.2,
            linecolor='black',
            annot = True,
            cmap="YlGnBu"
            )
```

Out[21]:

```
<AxesSubplot:>
```

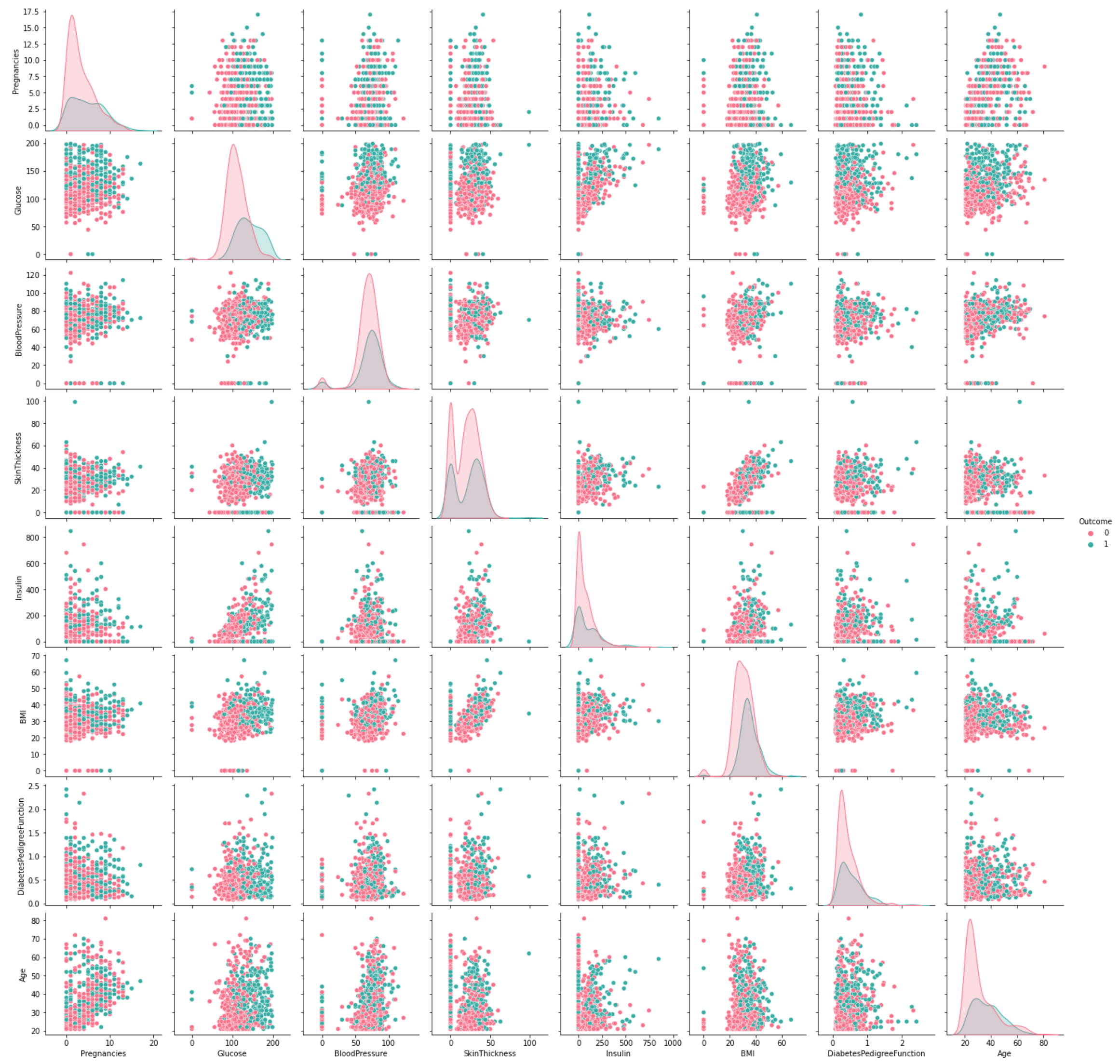


In [22]:

```
sns.pairplot(df,                # dataset
             hue='Outcome',     # variable in dataset to map plot aspects to different co
lor
             palette='husl',
             )
```

Out[22]:

```
<seaborn.axisgrid.PairGrid at 0x217d9fceb48>
```



Above plot shows the distribution of dataset in scatter plot and Kernel Density Estimator, for the different combination of datasets with respect to the Outcome.

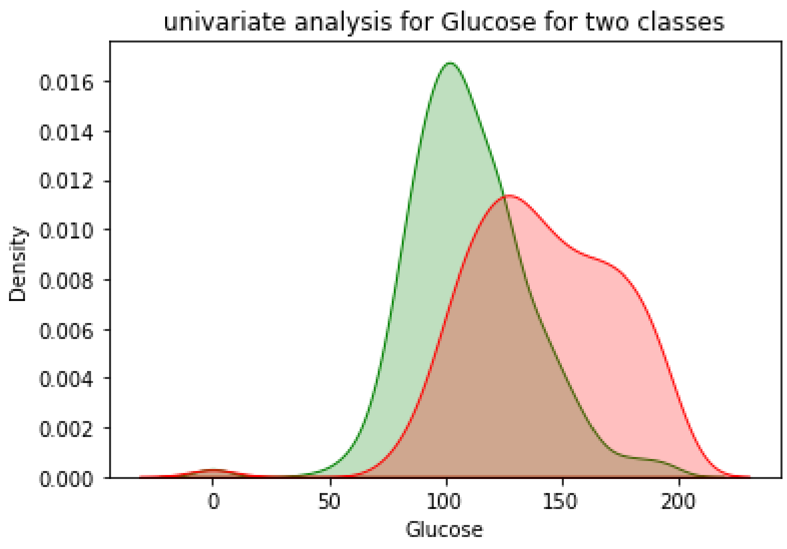
Bivariate Analysis

In [23]:

```
sns.kdeplot(class_0.Glucose, shade=True,color="g")
sns.kdeplot(class_1.Glucose, shade=True,color="r")
plt.title('univariate analysis for Glucose for two classes')
```

Out[23]:

```
Text(0.5, 1.0, 'univariate analysis for Glucose for two classes')
```



Here In this Kernel Density Estimate plot, the data of Glucose for Outcome_0 is distributed like a Normal/Gaussian distribution, with sharp peakness and most of the data points accumalted nearby the mean.

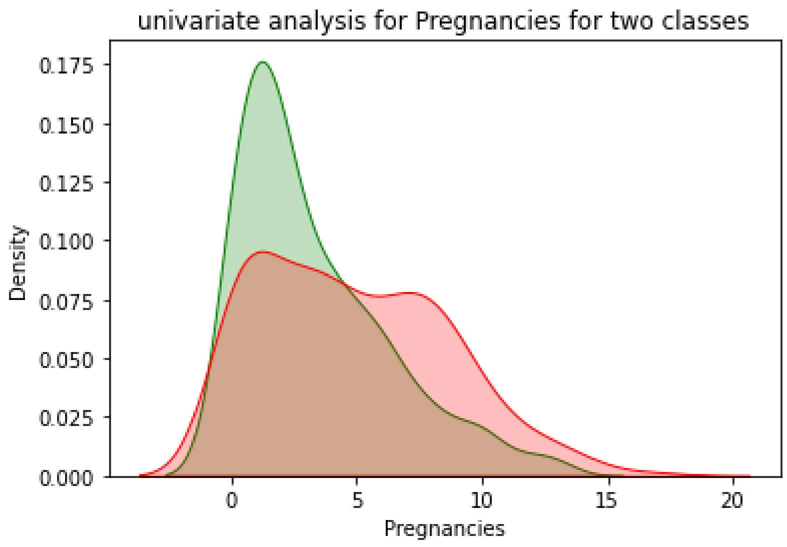
But the curve looks abnormal because of the Outcome_1, here the datas are accumaleted in a wide pattern with some outliers at the bottom.

In [24]:

```python
sns.kdeplot(class_0.Pregnancies, shade=True,color="g")
sns.kdeplot(class_1.Pregnancies, shade=True,color="r")
plt.title('univariate analysis for Pregnancies for two classes')
```

Out[24]:

```
Text(0.5, 1.0, 'univariate analysis for Pregnancies for two classes')
```



Here in the above 2 curves, In Outcome_0 (Green curve), the Pregnancies distribution is look like right skewed, with sharp peakness and mode is one time Pregnancy counts around 106, the median is two times pregnancy with the count of 84.

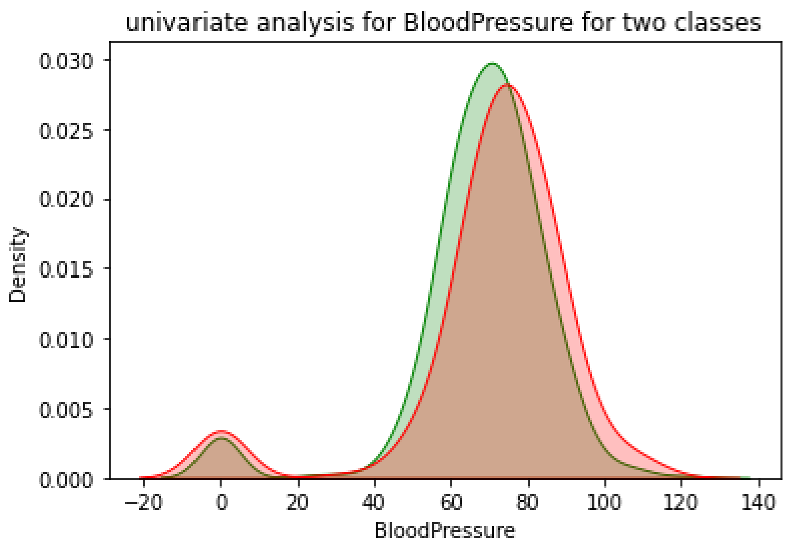
In Outcome_1 (Red curve), the Pregnancies distribution is look like abnormal distribution, data is distributed widely and mode is zero time Pregnancy counts around 38, the median is four times pregnancy with the count of 23. The right tail extended widely at end.

In [25]:

```
sns.kdeplot(class_0.BloodPressure, shade=True,color="g")
sns.kdeplot(class_1.BloodPressure, shade=True,color="r")
plt.title('univariate analysis for BloodPressure for two classes')
```

Out[25]:

```
Text(0.5, 1.0, 'univariate analysis for BloodPressure for two classes')
```



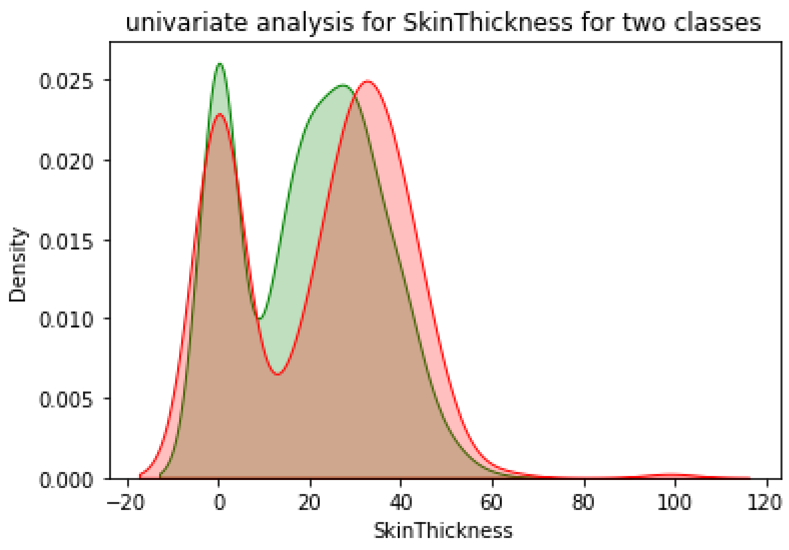
Here in above kDE plot, for the both the Outcome 1 & 0, except that small abnormality near zero, the rest of the data is completely shows that it is normal distributed

In [26]:

```
sns.kdeplot(class_0.SkinThickness,shade=True,color="g")
sns.kdeplot(class_1.SkinThickness, shade=True,color="r")
plt.title('univariate analysis for SkinThickness for two classes')
```

Out[26]:

```
Text(0.5, 1.0, 'univariate analysis for SkinThickness for two classes')
```



Here in both the curve the mode is equal to 0, for Outcome = 0 it counts upto 139 and Outcome = 1 it counts upto 88, this make the curve looks so abnormal.

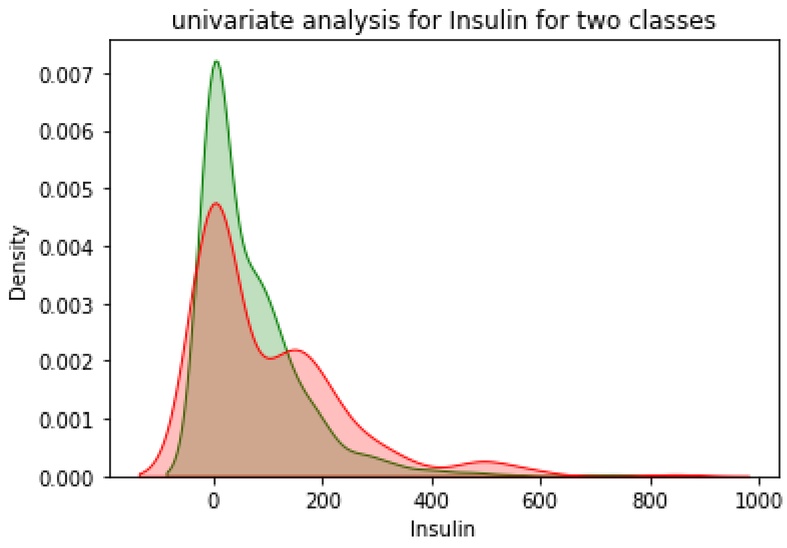
The tail part of Outcome = 0, is extended quite alot. The presence of outlier will distrub the classfication while building the model i.e it will result in poor classification or overfitting.

In [27]:

```
sns.kdeplot(class_0.Insulin,shade=True,color="g")
sns.kdeplot(class_1.Insulin, shade=True,color="r")
plt.title('univariate analysis for Insulin for two classes')
```

Out[27]:

```
Text(0.5, 1.0, 'univariate analysis for Insulin for two classes')
```



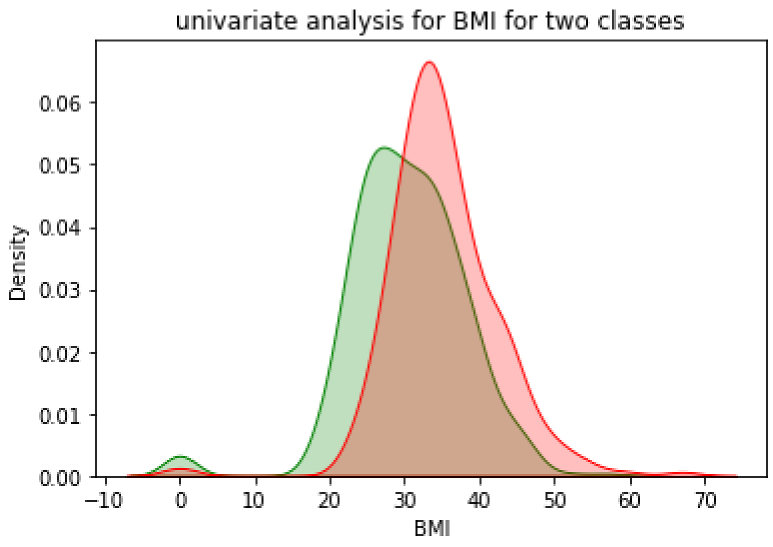
KDE plot for both the Outcome is completely right skewed. The data distribtion of Outcome_1 is like so wavy and the right tail is extended & shows the plots of outliers.

In [28]:

```
sns.kdeplot(class_0.BMI,shade=True,color="g")
sns.kdeplot(class_1.BMI, shade=True,color="r")
plt.title('univariate analysis for BMI for two classes')
```

Out[28]:

```
Text(0.5, 1.0, 'univariate analysis for BMI for two classes')
```



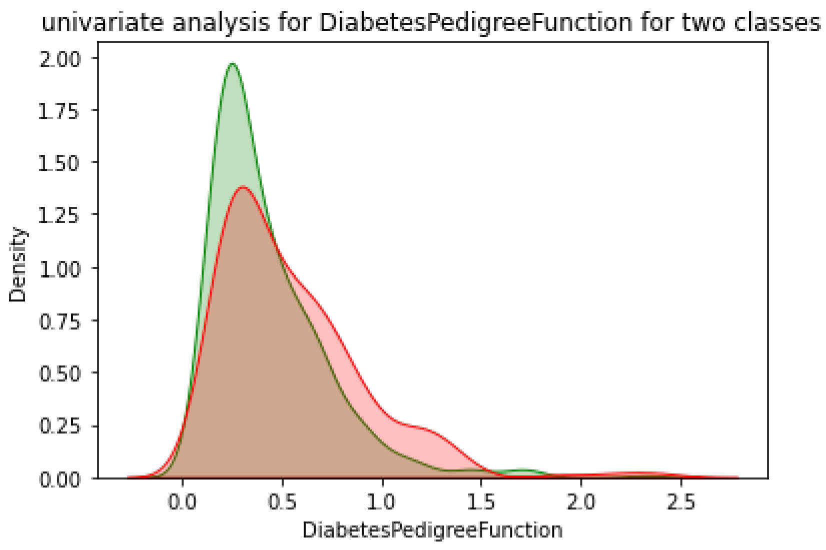
Here, there is some accumulation of data on left side of both the curve. But the Outcome_1 is a normal distribution and it tell BMI plays one important role in Diabetes, abnormal is BMI will causes Diabetes and the right tail keep extending.

In [29]:

```
sns.kdeplot(class_0.DiabetesPedigreeFunction,shade=True,color="g")
sns.kdeplot(class_1.DiabetesPedigreeFunction, shade=True,color="r")
plt.title('univariate analysis for DiabetesPedigreeFunction for two classes')
```

Out[29]:

```
Text(0.5, 1.0, 'univariate analysis for DiabetesPedigreeFunction for two c
lasses')
```



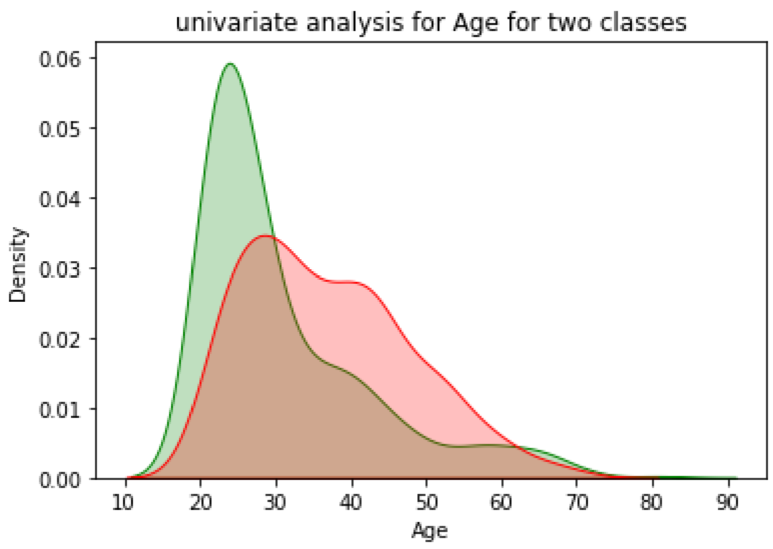
Here in above curve, it shows the picture of both the outcomes were right skewed. Outcome 1, the data distribution is wide so there is abnormal in shape of Density for Outcome_1.

In [30]:

```
sns.kdeplot(class_0.Age,shade=True,color="g")
sns.kdeplot(class_1.Age, shade=True,color="r")
plt.title('univariate analysis for Age for two classes')
```

Out[30]:

```
Text(0.5, 1.0, 'univariate analysis for Age for two classes')
```



KDE plot for Age, outcome_0 is right skewed of mode value of 22 with the counts of 61 and median of 27 with the counts of 24. But after the age of 35 (approx) the distribution pattern is abnormal towards the right tail.

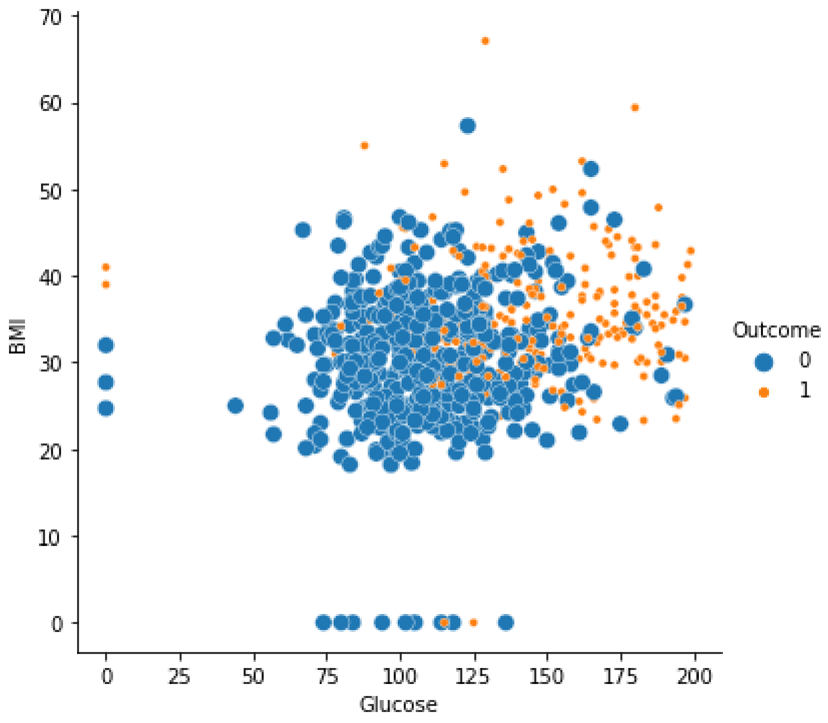
KDE plot for Age, outcome_1 the data is speard across between 25-50 yrs.

In [31]:

```
sns.relplot(x='Glucose',
            y='BMI',
            data = df,
            hue = 'Outcome',
            size='Outcome')
```

Out[31]:

```
<seaborn.axisgrid.FacetGrid at 0x217d8482388>
```



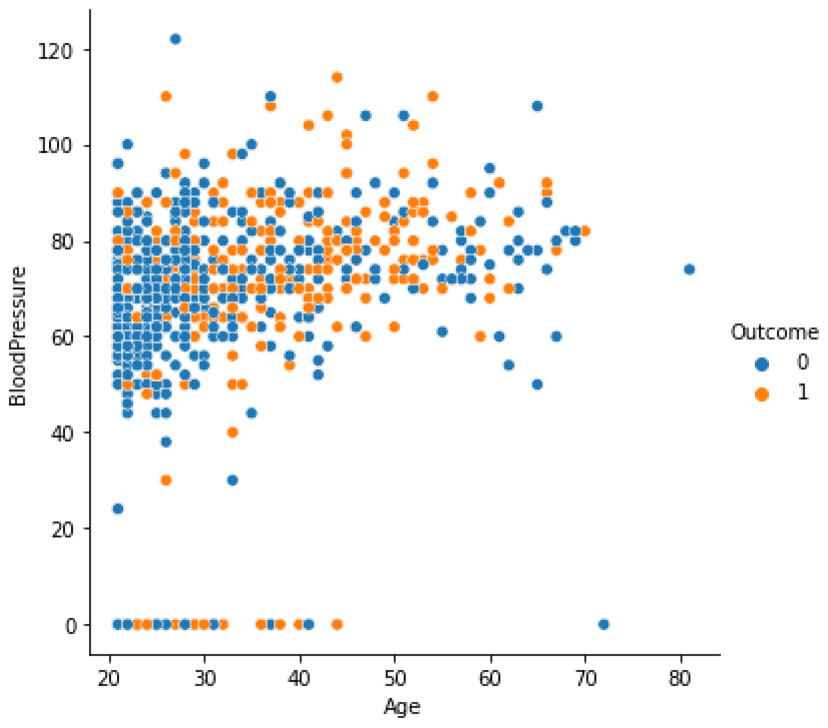
The above scatter plot tells, the people who have abnormal BMI and higher the Glucose level, will have higher the chance of getting Diabetic(orange small dots)

In [32]:

```
sns.relplot(x='Age',
            y='BloodPressure',
            data = df,
            hue = 'Outcome',)
            #size='Outcome')
```

Out[32]:

```
<seaborn.axisgrid.FacetGrid at 0x217d70e4e88>
```



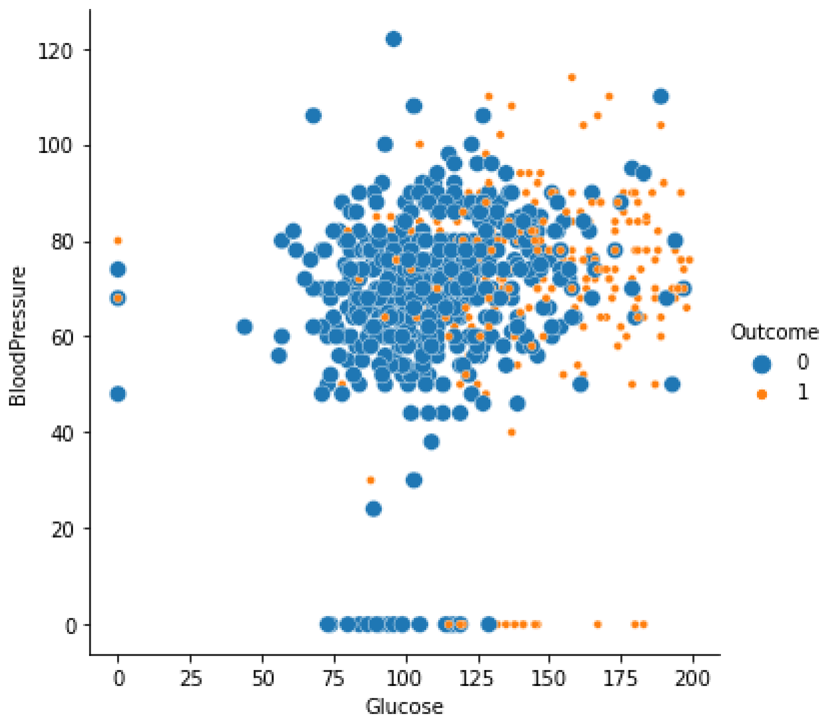
Here the above plot shows that, higher the chance for people have High BloodPressure and getting Aged to be a Diabetic.

In [33]:

```
sns.relplot(x='Glucose',
            y='BloodPressure',
            data = df,
            hue = 'Outcome',
            size='Outcome')
```

Out[33]:

```
<seaborn.axisgrid.FacetGrid at 0x217d70ceb08>
```



Here the above plot shows that, higher the chance for people have High BloodPressure and Glucose to be a Diabetic.

Feature Tranformation

Binning the age to avoid the model to get distrubed by outliers.

In [34]:

```
df['Age_bin']=pd.cut(x = df['Age'],                                    # Cloumn to be
binned
                        bins = [20,30,50,100],                          # Binnning siz
es
                        labels = ['young_aged','middle_aged','old_aged'] # class name f
or Binning
                    )
df.head(9)
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.
5	5	116	74	0	0	25.6	0.
6	3	78	50	32	88	31.0	0.
7	10	115	0	0	0	35.3	0.
8	2	197	70	45	543	30.5	0.

In [35]:

```
df = df.drop('Age',axis=1)
df.head()
```

Out[35]:

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

In [36]:

```
Age_bin = df.Age_bin
df.insert(0,'Aged_bin',Age_bin)
df = df.drop('Age_bin',axis=1)
df.head()
```

Out[36]:

	Aged_bin	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
0	middle_aged	6	148	72	35	0	33.6	
1	middle_aged	1	85	66	29	0	26.6	
2	middle_aged	8	183	64	0	0	23.3	
3	young_aged	1	89	66	23	94	28.1	
4	middle_aged	0	137	40	35	168	43.1	

Encoding the categorical Age column

In [37]:

```
df.Aged_bin = df.Aged_bin.replace(to_replace = ['young_aged','middle_aged','old_aged'],
value =[0,1,2],inplace=False)
df.head()
```

Out[37]:

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

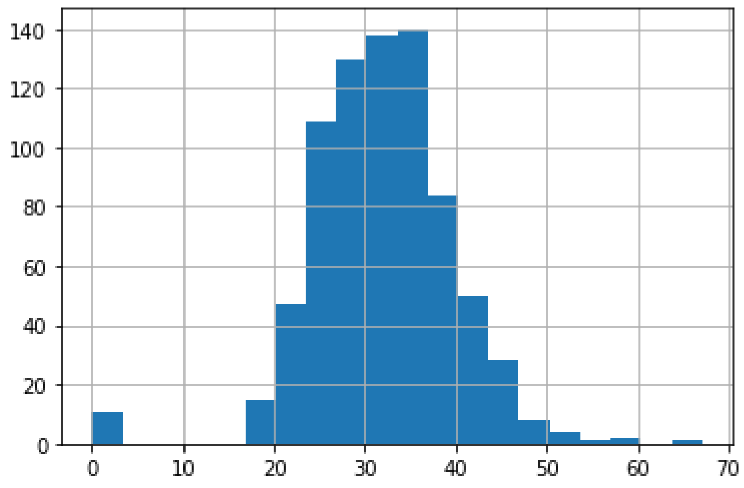
Treating BMI Column

In [38]:

```
df.BMI.hist(bins=20)
```

Out[38]:

```
<AxesSubplot:>
```



In [39]:

```
df.loc[((df.Outcome == 0) &  (df.BMI ==0)) , 'BMI'] = df.loc[((df.Outcome == 0) &  (df.
BMI ==0)) , 'BMI'].mean()
```

In [40]:

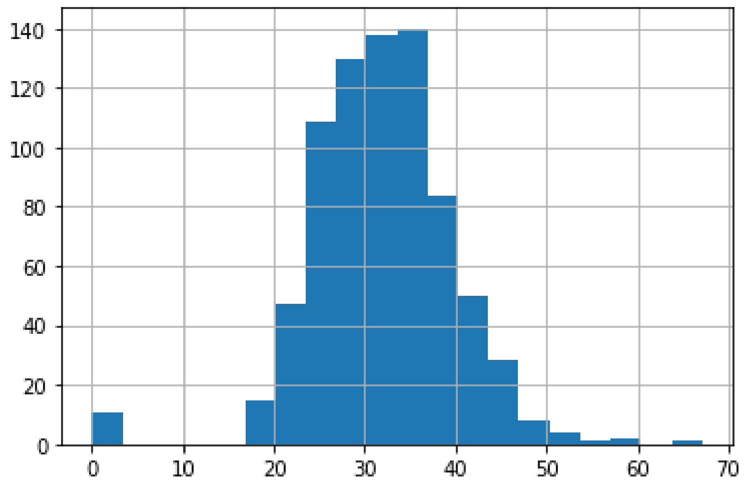
```
df.loc[((df.Outcome == 1) &  (df.BMI ==0)) , 'BMI'] = df.loc[((df.Outcome == 1) &  (df.
BMI ==0)) , 'BMI'].mean()
```

In [41]:

```
df.BMI.hist(bins=20)
```

Out[41]:

```
<AxesSubplot:>
```



In [42]:

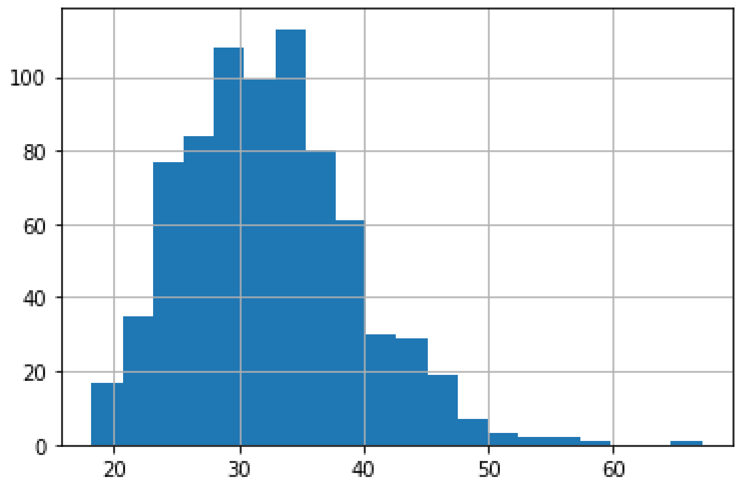
```
class_0 = (df['Outcome'] == 0)
class_1 = (df['Outcome'] == 1)
avg_class_0 = df.loc[class_0, 'BMI'].mean()
avg_class_1 = df.loc[class_1, 'BMI'].mean()
df.loc[df['BMI']==0 & class_0, 'BMI'] = avg_class_0
df.loc[df['BMI']==0 & class_1, 'BMI'] = avg_class_1
```

In [43]:

```
df.BMI.hist(bins=20)
```

Out[43]:

```
<AxesSubplot:>
```

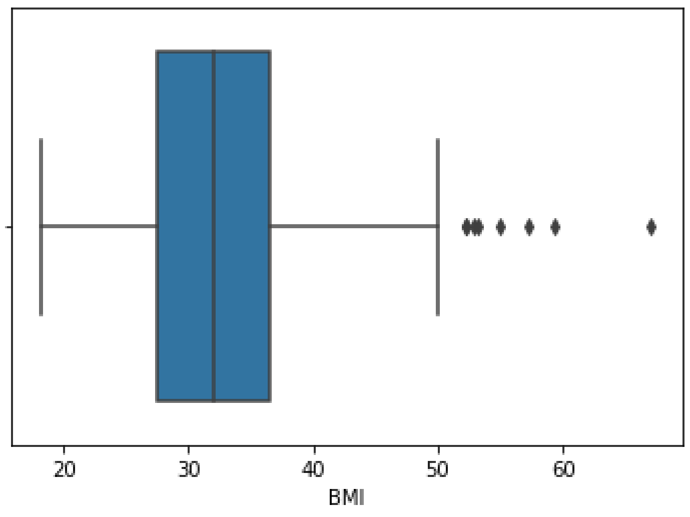


In [44]:

```
sns.boxplot(x=df.BMI)
```

Out[44]:

```
<AxesSubplot:xlabel='BMI'>
```



Category BMI range - kg/m2 Severe Thinness < 16 Moderate Thinness 16 - 17 Mild Thinness 17 - 18.5 Normal 18.5 - 25 Overweight 25 - 30 Obese Class I 30 - 35 Obese Class II 35 - 40 Obese Class III > 40

Since it has many outlier, So it is converted into a Categorical caloumn

Binning the BMI Column

In [45]:

```
df['BMI_bin']=pd.cut(x = df['BMI'],
                        bins = [18,25,30,80],
                        labels = ['Normal','Overweight','Obese'])
df.head(9)
```

Out[45]:

	Aged_bin	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
0	1	6	148	72	35	0	33.6	
1	1	1	85	66	29	0	26.6	
2	1	8	183	64	0	0	23.3	
3	0	1	89	66	23	94	28.1	
4	1	0	137	40	35	168	43.1	
5	0	5	116	74	0	0	25.6	
6	0	3	78	50	32	88	31.0	
7	0	10	115	0	0	0	35.3	
8	2	2	197	70	45	543	30.5	

In [46]:

```
df.BMI_bin = df.BMI_bin.replace(to_replace = ['Normal','Overweight','Obese'],value =[0,
1,2],inplace=False)
df.head()
```

Out[46]:

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

In [47]:

```
BMI_bin = df.BMI_bin
df.insert(0,'BMI_binned',BMI_bin)
```

In [48]:

```
df = df.drop(['BMI_bin','BMI'],axis=1)
df.head()
```

Out[48]:

	BMI_binned	Aged_bin	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	Dia
0	2	1	6	148	72	35	0	
1	1	1	1	85	66	29	0	
2	0	1	8	183	64	0	0	
3	1	0	1	89	66	23	94	
4	2	1	0	137	40	35	168	

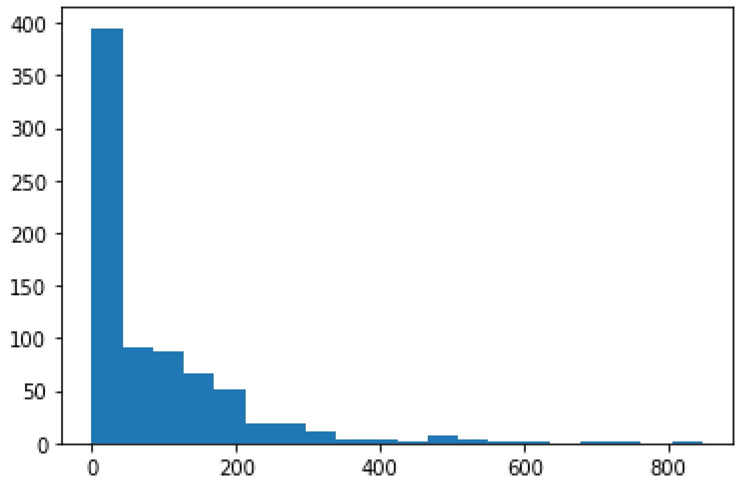
Treating the Insulin

In [49]:

```
df.Insulin.hist(bins=20,grid=False)
```

Out[49]:

```
<AxesSubplot:>
```



In [50]:

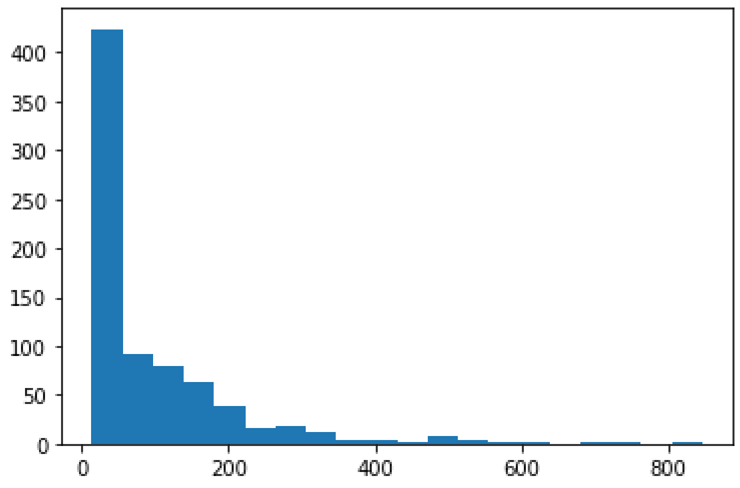
```
class_0 = (df['Outcome'] == 0)
class_1 = (df['Outcome'] == 1)
avg_class_0 = df.loc[class_0,'Insulin'].median()
avg_class_1 = df.loc[class_1,'Insulin'].median()
df.loc[df['Insulin']==0 & class_0, 'Insulin'] = avg_class_0
df.loc[df['Insulin']==0 & class_1, 'Insulin'] = avg_class_1
```

In [51]:

```
df.Insulin.hist(bins=20,grid=False)
```

Out[51]:

```
<AxesSubplot:>
```



Here in above plot, even after replacing the 0 with median values based on their respective class, still it looks right skewed.

Log() Tranformation

In [52]:

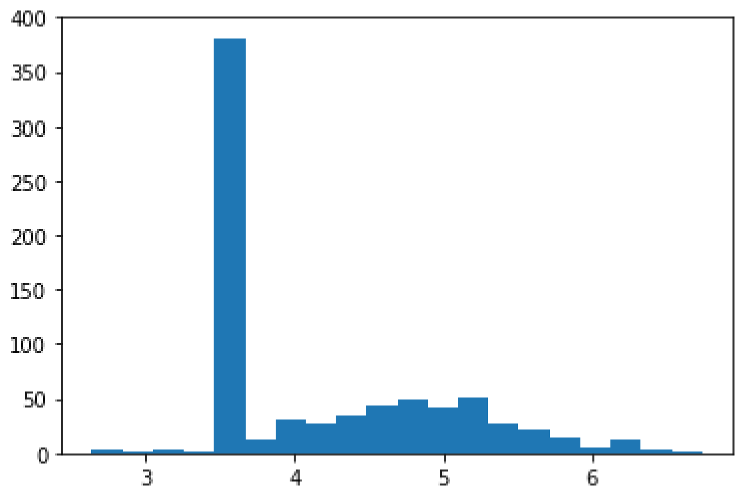
```
df.Insulin = np.log(df.Insulin)
```

In [53]:

```
df.Insulin.hist(bins=20,grid=False)
```

Out[53]:

```
<AxesSubplot:>
```

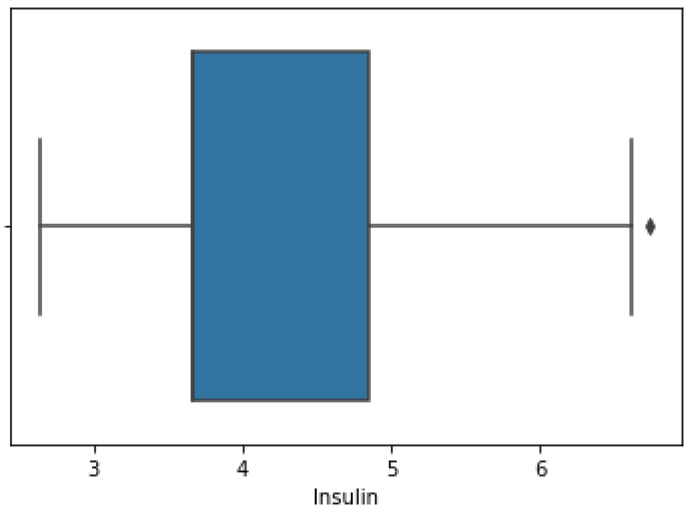


In [54]:

```
sns.boxplot(x=df.Insulin)
```

Out[54]:

```
<AxesSubplot:xlabel='Insulin'>
```



Here in above plot, once after the log() Tranformation because of previous data is right skewed and now median & IQR is changed and it shows more outliers on upper and lower bounds.

Because of persence of more Outliers in data, it is scaled with Robust Scaler

In [55]:

```
# Robust Scaler
from sklearn.preprocessing import RobustScaler
rs = RobustScaler(with_centering=True,
                  with_scaling=True,
                  quantile_range=(25.0, 75.0),
                  copy=True)
```

In [56]:

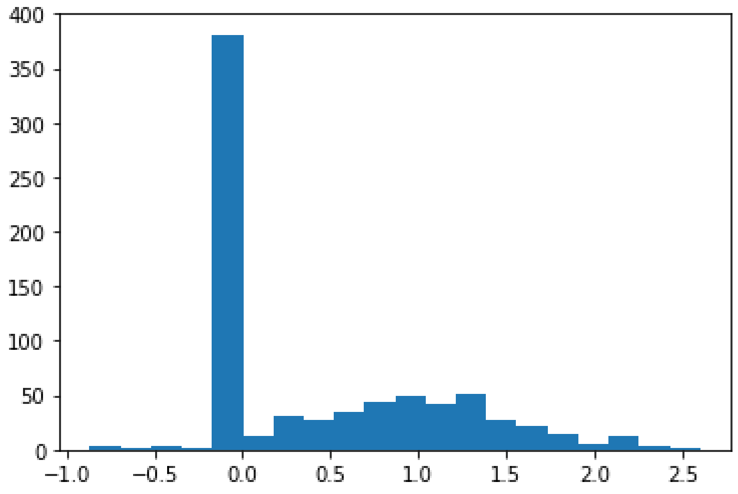
```
df['Insulin'] = rs.fit_transform(df['Insulin'].values.reshape(-1,1))
```

In [57]:

```
df.Insulin.hist(bins=20,grid=False)
```

Out[57]:

```
<AxesSubplot:>
```

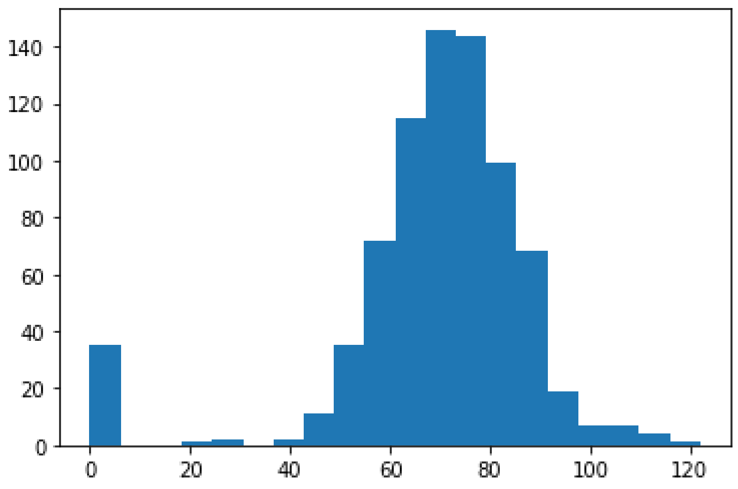


In [58]:

```
df.BloodPressure.hist(bins=20,grid=False)
```

Out[58]:

```
<AxesSubplot:>
```



In [59]:

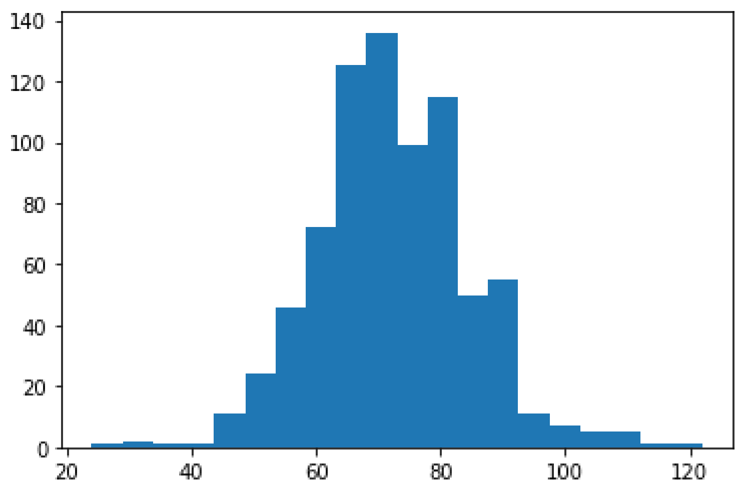
```
avg_class_0 = df.loc[class_0,'BloodPressure'].median()
avg_class_1 = df.loc[class_1,'BloodPressure'].median()
df.loc[df['BloodPressure']==0 & class_0, 'BloodPressure'] = avg_class_0
df.loc[df['BloodPressure']==0 & class_1, 'BloodPressure'] = avg_class_1
```

In [60]:

```
df.BloodPressure.hist(bins=20,grid=False)
```

Out[60]:

```
<AxesSubplot:>
```



In [61]:

```
from sklearn.preprocessing import StandardScaler
```

In [62]:

```
std_scale = StandardScaler(copy=True,
                           with_mean=True,
                           with_std=True)
```

In [63]:

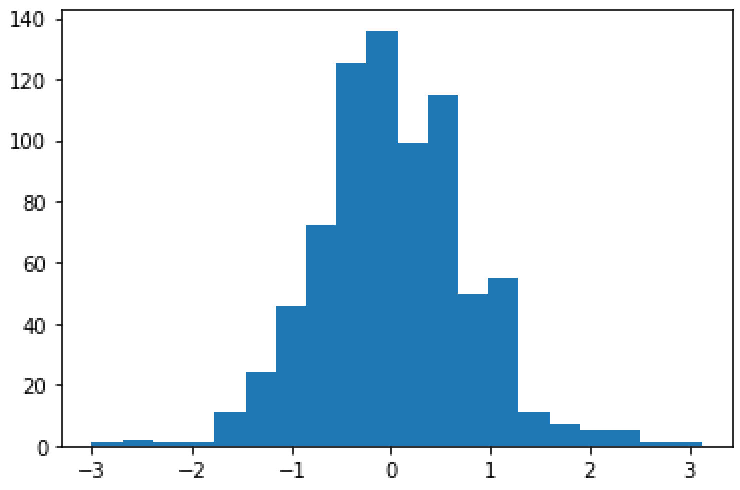
```
df['BloodPressure'] = rs.fit_transform(df['BloodPressure'].values.reshape(-1,1))
```

In [64]:

```
df.BloodPressure.hist(bins=20,grid=False)
```

Out[64]:

```
<AxesSubplot:>
```

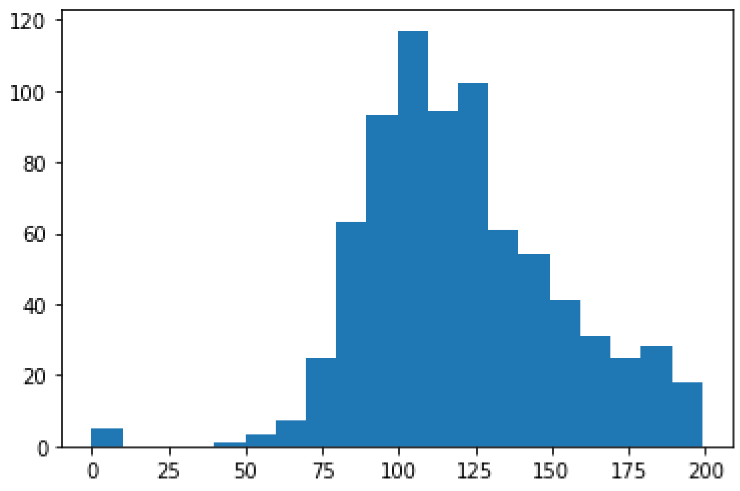


In [65]:

```
df.Glucose.hist(bins=20,grid=False)
```

Out[65]:

```
<AxesSubplot:>
```



In [66]:

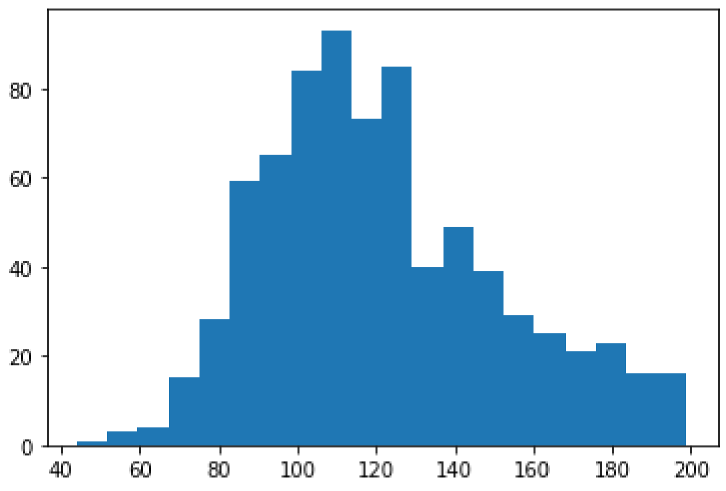
```
avg_class_0 = df.loc[class_0,'Glucose'].median()
avg_class_1 = df.loc[class_1,'Glucose'].median()
df.loc[df['Glucose']==0 & class_0, 'Glucose'] = avg_class_0
df.loc[df['Glucose']==0 & class_1, 'Glucose'] = avg_class_1
```

In [67]:

```
df.Glucose.hist(bins=20,grid=False)
```

Out[67]:

```
<AxesSubplot:>
```



In [68]:

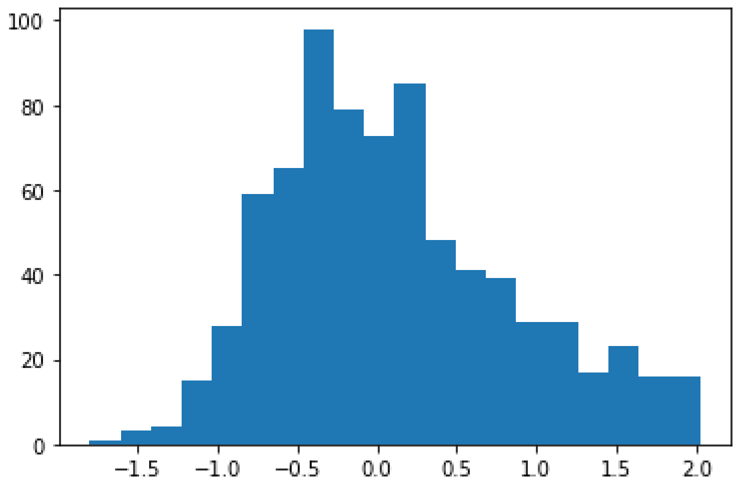
```
df['Glucose'] = rs.fit_transform(df['Glucose'].values.reshape(-1,1))
```

In [69]:

```
df.Glucose.hist(bins=20,grid=False)
```

Out[69]:

```
<AxesSubplot:>
```

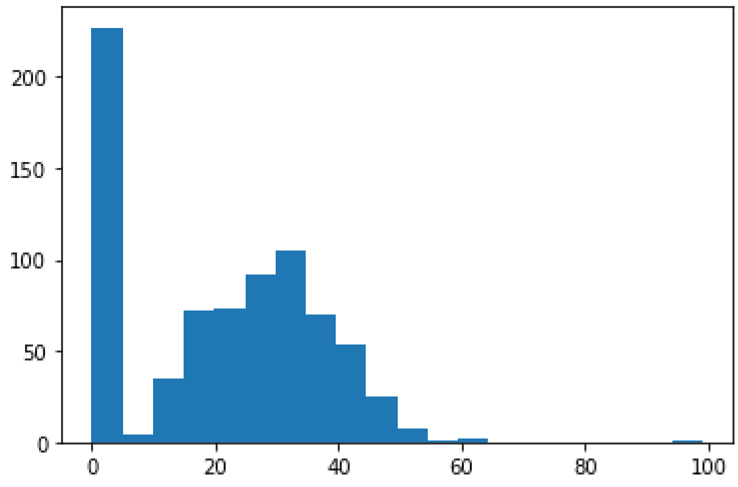


In [70]:

```
df.SkinThickness.hist(bins=20,grid=False)
```

Out[70]:

```
<AxesSubplot:>
```



In [71]:

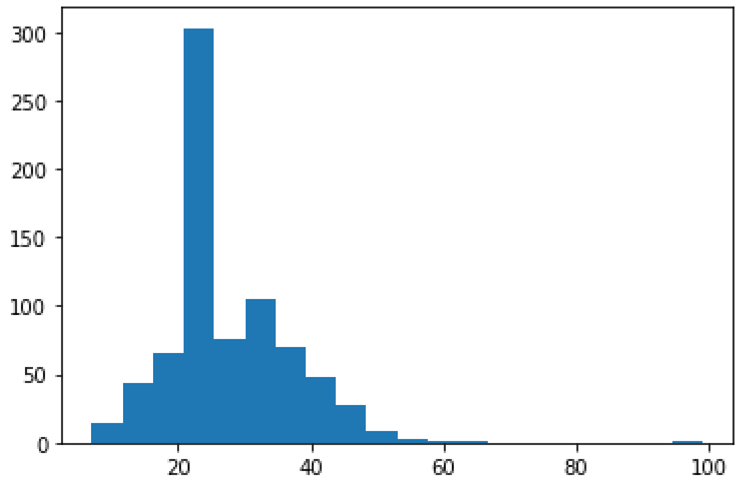
```
avg_class_0 = df.loc[class_0,'SkinThickness'].median()
avg_class_1 = df.loc[class_1,'SkinThickness'].median()
df.loc[df['SkinThickness']==0 & class_0, 'SkinThickness'] = avg_class_0
df.loc[df['SkinThickness']==0 & class_1, 'SkinThickness'] = avg_class_1
```

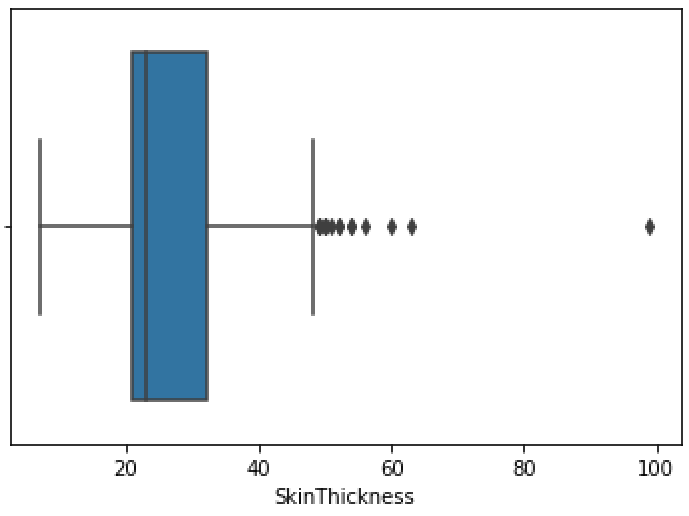
In [72]:

```
df.SkinThickness.hist(bins=20,grid=False)
```

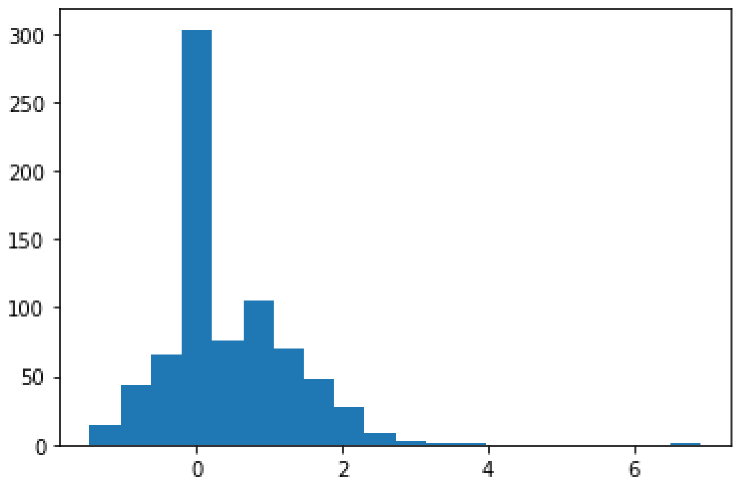
Out[72]:

```
<AxesSubplot:>
```



In [73]:
```
sns.boxplot(x=df.SkinThickness)
```
Out[73]:
```
<AxesSubplot:xlabel='SkinThickness'>
```


Since it has more outlier, so it is treated with Robust scaler

In [74]:
```
df['SkinThickness'] = rs.fit_transform(df['SkinThickness'].values.reshape(-1,1))
```
In [75]:
```
df.SkinThickness.hist(bins=20,grid=False)
```
Out[75]:
```
<AxesSubplot:>
```


In [76]:

```
df.head(9)
```

Out[76]:

	BMI_binned	Aged_bin	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
0	2	1	6	0.765432	0.000	1.090909	0.000000
1	1	1	1	-0.790123	-0.375	0.545455	0.000000
2	0	1	8	1.629630	-0.500	-0.181818	0.000000
3	1	0	1	-0.691358	-0.375	0.000000	0.743906
4	2	1	0	0.493827	-2.000	1.090909	1.234922
5	1	0	5	-0.024691	0.125	-0.181818	0.000000
6	2	0	3	-0.962963	-1.375	0.818182	0.688132
7	2	0	10	-0.049383	-0.125	-0.181818	0.000000
8	2	2	2	1.975309	-0.125	2.000000	2.226939

In [77]:

```
df.Outcome.value_counts()
```

Out[77]:

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

Here the data is Imbalanced so it is oversampled

Oversampling of Data

In [78]:

```
#!pip install imblearn
```

In [79]:

```
#!pip install -U imbalanced-learn
```

In [80]:

```
# Synthetic Minority Over-sampling Technique

from imblearn.over_sampling import SMOTE

smt = SMOTE(sampling_strategy='auto', random_state=9,n_jobs=-1)
x = df.drop(['Outcome'],axis = 1)
y = df.Outcome
x, y = smt.fit_sample(x,y)
```

In [81]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

# Logistic Regression
from sklearn.linear_model import LogisticRegression

# Tree Model
from sklearn.tree import DecisionTreeClassifier
```

In [82]:

```
#!pip install lightgbm
```

In [83]:

```
# Support Vector Machine
from sklearn.svm import SVC
```

In [84]:

```
# Ensemble Model
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [85]:

```
# Metrics
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
```

In [86]:

```
# Ensemble Model
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [87]:

```
# Metrics
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
```

Hold-out Method

In [88]:

```
x_train,x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=9,str
atify=y)
```

In [89]:

```
print('Shape of x_train',x_train.shape)
print()
print('Shape of y_train',y_train.shape)
print()
print('Shape of x_test',x_test.shape)
print()
print('Shape of y_test',y_test.shape)
```

```
Shape of x_train (800, 8)

Shape of y_train (800,)

Shape of x_test (200, 8)

Shape of y_test (200,)
```

Logistic Regression

In [90]:

```
lr = LogisticRegression(random_state=100,
                        n_jobs=-1,
                        penalty='l2',
                        solver='liblinear'
                       )
lr.fit(x_train, y_train)
y_pred_lr = lr.predict(x_test)
```

In [91]:

```
print('test accuracy : ', accuracy_score(y_pred_lr,y_test))
```

```
test accuracy :  0.725
```

Decision Tree Classifier with oversampling

In [92]:

```
dtc = DecisionTreeClassifier(criterion="entropy",  # For the information gain
                               splitter="best",     # For the best split
                               random_state=9
                            )
```

In [93]:

```
dtc.fit(x_train,y_train)
y_pred_dtc = dtc.predict(x_test)
print('test accuracy : ', accuracy_score(y_pred_dtc,y_test))
```

```
test accuracy :  0.695
```

Random Forest Classifier with oversampling

In [94]:

```
rfc = RandomForestClassifier(max_depth=2,
                             random_state=0,
                             n_jobs=-1)
rfc.fit(x_train,y_train)
y_pred_rfc = dtc.predict(x_test)
print('test accuracy : ', accuracy_score(y_pred_rfc,y_test))
```

```
test accuracy :  0.695
```

Without Oversampling

In [95]:

```
x_s = df.drop('Outcome',axis=1)
y_s = df.Outcome
x_train,x_test, y_train, y_test = train_test_split(x_s,y_s,test_size=0.2,random_state=9
)
```

In [96]:

```
print('Shape of x_train',x_train.shape)
print()
print('Shape of y_train',y_train.shape)
print()
print('Shape of x_test',x_test.shape)
print()
print('Shape of y_test',y_test.shape)
```

```
Shape of x_train (614, 8)

Shape of y_train (614,)

Shape of x_test (154, 8)

Shape of y_test (154,)
```

Decision Tree Classifier

In [97]:

```
dtc1 = DecisionTreeClassifier(criterion="entropy",  # For the information gain
                              splitter="best",    # For the best split
                              random_state=9
                             )
dtc1.fit(x_train,y_train)
y_pred_dtc1 = dtc1.predict(x_test)
```

In [98]:

```
print('test accuracy : ', accuracy_score(y_pred_dtc1,y_test))
```

```
test accuracy :  0.6883116883116883
```

Random Forest Classifier

In [99]:

```
rfc1 = RandomForestClassifier(max_depth=2,
                              random_state=0,
                              n_jobs=-1)
```

In [100]:

```
rfc1.fit(x_train,y_train)
y_pred_rfc1 = rfc1.predict(x_test)
```

In [101]:

```
print('test accuracy : ', accuracy_score(y_pred_rfc1,y_test))
```

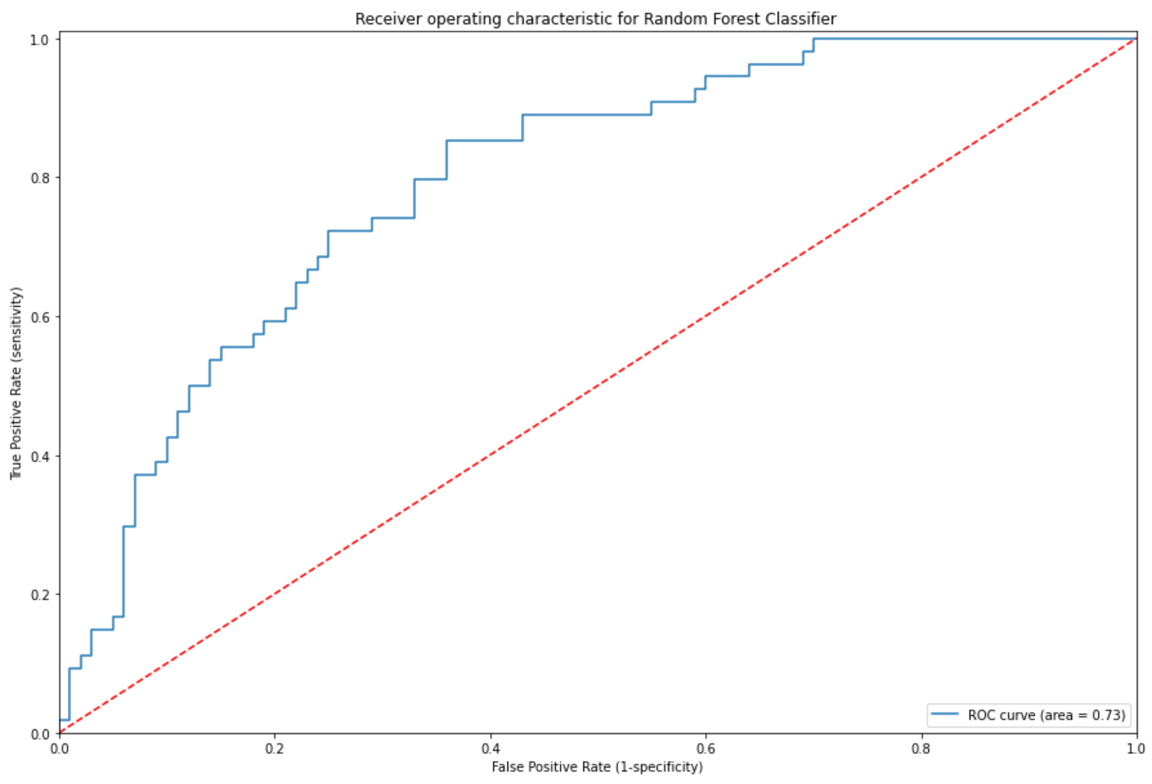
```
test accuracy :  0.7272727272727273
```

ROC AUC curve for Random Forest Classifier

In [102]:

```
fpr, tpr, thershold = roc_curve(y_test, rfc1.predict_proba(x_test)[:,1])
rfc_roc = roc_auc_score(y_pred_rfc1,y_test)
plt.figure()
plt.subplots(figsize=(15,10))
plt.plot(fpr, tpr, label = 'ROC curve (area = %0.2f)'%rfc_roc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0,1.0])
plt.ylim([0,1.01])
plt.xlabel('False Positive Rate (1-specificity)')
plt.ylabel('True Positive Rate (sensitivity)')
plt.title('Receiver operating characteristic for Random Forest Classifier ')
plt.legend(loc ="lower right")
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



SVM and GridSearchCV

In [103]:

```
C=np.logspace(-2,2,5)
gamma=np.logspace(-5,5,5)
kernel=['linear', 'rbf', 'sigmoid']
param_grid = dict(C=C,gamma=gamma,kernel=kernel)
```

In [104]:

```
grid = GridSearchCV(SVC(),param_grid=param_grid,n_jobs=-1)
```

In [105]:

```
grid.fit(x_train,y_train)
y_pred_grid = grid.predict(x_test)
print('Grid Search best parameter for SVC are : ',grid.best_params_)
print()
print('SVC predicted accuracy score is : ', accuracy_score (y_pred_grid,y_test))
```

```
Grid Search best parameter for SVC are :  {'C': 100.0, 'gamma': 0.00316227
76601683794, 'kernel': 'rbf'}

SVC predicted accuracy score is :  0.7272727272727273
```

Light GBM Classifier

In [106]:

```
lgbm = LGBMClassifier(boosting_type='goss',   # Gradient-based One-Side Sampling
                      n_jobs=-1,
                      objective='binary',
                      random_state=9,
                      importance_type='split'
                      )
```

In [107]:

```
lgbm.fit(x_train, y_train)
y_pred_lgbm = lgbm.predict(x_test)
print('lgbm predicted accuracy score is : ', accuracy_score (y_pred_lgbm,y_test))
```

```
lgbm predicted accuracy score is :  0.7597402597402597
```

Light GBM Classifier gives the better accuracy when compared to other models.

Statistics on Data Availability

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [108]:

```
temp = df['Outcome'].value_counts().to_frame()
```

In [109]:

```
temp
```

Out[109]:

	Outcome
0	500
1	268

In [110]:

```
#temp.to_csv(intermediate_path+"Data availability_stats.csv")
```

In [111]:

```
#df.to_csv(intermediate_path + "Premodel_Processed_Data.csv", index = False)
```

In [112]:

```
COLUMN_NAMES = ["Process", "Model Name", "F1 Scores","Range of F1 Scores","Std Deviatio
n of F1 Scores"]
df_model_selection = pd.DataFrame(columns=COLUMN_NAMES)
```

In [113]:

```
df_model_selection
```

Out[113]:

Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores

In [114]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
from sklearn import metrics

def stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y):
    global df_model_selection
    skf = StratifiedKFold(n_splits=5, random_state=None)
    weighted_f1_score = []
    for train_index, val_index in skf.split(X,y):
        X_train, X_test = X.iloc[train_index], X.iloc[val_index]
        y_train, y_test = y.iloc[train_index], y.iloc[val_index]
        model_obj.fit(X_train, y_train)##### HERE ###
        test_ds_predicted = model_obj.predict( X_test ) ##### HERE ####
        #print( metrics.classification_report( y_test, test_ds_predicted ) )
        weighted_f1_score.append(round(f1_score(y_test, test_ds_predicted , average='we
ighted'),2))

    sd_weighted_f1_score = np.std(weighted_f1_score, ddof=1)
    range_of_f1_scores = "{}-{}".format(min(weighted_f1_score),max(weighted_f1_score))
    df_model_selection = pd.concat([df_model_selection,pd.DataFrame([[process,model_nam
e,sorted(weighted_f1_score),range_of_f1_scores,sd_weighted_f1_score]], columns =COLUMN_
NAMES) ])
```

In [115]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   BMI_binned                768 non-null    int64  
 1   Aged_bin                  768 non-null    int64  
 2   Pregnancies               768 non-null    int64  
 3   Glucose                   768 non-null    float64
 4   BloodPressure             768 non-null    float64
 5   SkinThickness             768 non-null    float64
 6   Insulin                   768 non-null    float64
 7   DiabetesPedigreeFunction  768 non-null    float64
 8   Outcome                   768 non-null    int64  
dtypes: float64(5), int64(4)
memory usage: 54.1 KB
```

In [116]:

```
train_ds_features = df.drop(["Outcome"],axis='columns', inplace=False)
train_ds_result = pd.DataFrame(df["Outcome"])
#train_ds_features
#train_ds_result
```

In [117]:

```
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(train_ds_features,train_ds_result,t
est_size = 0.2,random_state = 55)
```

In [118]:

```
print(train_X)
```

```
     BMI_binned  Aged_bin  Pregnancies   Glucose  BloodPressure  \
129           1         2            0 -0.296296          0.750   
758           2         0            1 -0.271605          0.250   
460           0         1            9  0.074074          0.000   
282           2         1            7  0.395062          1.000   
15            1         1            7 -0.419753         -0.125   
..          ...       ...          ...       ...            ...   
37            2         1            9 -0.370370          0.250   
541           2         0            3  0.271605          0.000   
295           2         0            6  0.839506         -0.625   
666           2         2            4  0.691358          0.625   
461           0         0            1 -1.135802         -0.625   

     SkinThickness   Insulin  DiabetesPedigreeFunction  
129      -0.181818  0.000000                     0.741  
758      -0.181818  0.000000                     0.197  
460      -0.090909  0.305931                     0.733  
282      -0.727273  1.166818                     0.262  
15       -0.181818  0.000000                     0.484  
..             ...       ...                       ...  
37        1.272727  0.000000                     0.665  
541       0.181818  1.338983                     0.549  
295       0.727273  0.950400                     0.692  
666      -0.454545  0.000000                     0.235  
461      -0.181818  0.000000                     0.416  

[614 rows x 8 columns]
```

In [119]:

```
print(train_y)
```

```
     Outcome
129        1
758        0
460        0
282        0
15         1
..       ...
37         1
541        1
295        0
666        1
461        0

[614 rows x 1 columns]
```

Naive Bayes Model for Classification

In [120]:

```
from sklearn.naive_bayes import BernoulliNB
nb_clf = BernoulliNB()
nb_clf.fit( train_X, train_y )
```

Out[120]:

```
BernoulliNB()
```

In [121]:

```
test_ds_predicted = nb_clf.predict(test_X)
```

In [122]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.76      0.73      0.75        93
           1       0.62      0.66      0.63        61

    accuracy                           0.70       154
   macro avg       0.69      0.69      0.69       154
weighted avg       0.71      0.70      0.70       154
```

In [123]:

```
train_ds_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   BMI_binned                768 non-null    int64  
 1   Aged_bin                  768 non-null    int64  
 2   Pregnancies               768 non-null    int64  
 3   Glucose                   768 non-null    float64
 4   BloodPressure             768 non-null    float64
 5   SkinThickness             768 non-null    float64
 6   Insulin                   768 non-null    float64
 7   DiabetesPedigreeFunction  768 non-null    float64
dtypes: float64(5), int64(3)
memory usage: 48.1 KB
```

In [124]:

```
train_ds_result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Outcome  768 non-null    int64
dtypes: int64(1)
memory usage: 6.1 KB
```

In [125]:

```
model_obj = nb_clf
model_name = "Binomial Naive Bayes Classifier"
process = "B-NBC"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)


df_model_selection
```

Out[125]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373

In [ ]:

```

```

Logistic Regression

In [126]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(train_X, train_y)
test_ds_predicted = logreg.predict( test_X )
```

In [127]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.76      0.82      0.79        93
           1       0.69      0.61      0.64        61

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

In [128]:

```
model_obj = logreg
model_name = "Logistic Regression"
process = "L R"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[128]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332

In [ ]:

```
```

Decision Tree

In [129]:

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(criterion='entropy')

decision_tree.fit(train_X, train_y)
test_ds_predicted = decision_tree.predict( test_X )
```

In [130]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.78      0.80      0.79        93
           1       0.68      0.66      0.67        61

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

In [131]:

```
model_obj = decision_tree
model_name = "Decission Tree"
process = "DT"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[131]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495

In [ ]:

```

```

Random Forest

In [132]:

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=10)
```

In [133]:

```
random_forest.fit(train_X, train_y)
test_ds_predicted = random_forest.predict( test_X)
```

In [134]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.75      0.88      0.81        93
           1       0.75      0.54      0.63        61

    accuracy                           0.75       154
   macro avg       0.75      0.71      0.72       154
weighted avg       0.75      0.75      0.74       154
```

In [135]:

```
model_obj = random_forest
model_name = "Random Forest"
process = "R F"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[135]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148

In [ ]:

```

```

XG Boost

In [136]:

```
from xgboost import XGBClassifier
xgboost = XGBClassifier()
```

In [137]:

```
xgboost.fit(train_X, train_y)
test_ds_predicted = xgboost.predict( test_X )
```

```
[12:34:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

In [138]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.76      0.83      0.79        93
           1       0.70      0.61      0.65        61

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

In [139]:

```
model_obj = xgboost
model_name = "XG Boost"
process = "XG B"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

```
[12:34:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:34:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:34:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:34:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:34:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluat
ion metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

Out[139]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014

In [ ]:

SGD Classifier

In [140]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier

sgd = OneVsRestClassifier(SGDClassifier())
```

In [141]:

```
sgd.fit(train_X, train_y)
test_ds_predicted = sgd.predict( test_X )
```

In [142]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.77      0.84      0.80        93
           1       0.72      0.62      0.67        61

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

In [143]:

```
model_obj = sgd
model_name = "Stochastic Gradient Descent"
process = "SGD"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[143]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014
0	SGD	Stochastic Gradient Descent	[0.57, 0.64, 0.7, 0.74, 0.74]	0.57-0.74	0.072938

In [ ]:

```
```

Gaussian Process Classifier

In [144]:

```
from sklearn.gaussian_process import GaussianProcessClassifier
gausian_process = GaussianProcessClassifier()
```

In [145]:

```
gausian_process.fit(train_X, train_y)
test_ds_predicted = gausian_process.predict( test_X )
```

In [146]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.74      0.87      0.80        93
           1       0.73      0.54      0.62        61

    accuracy                           0.74       154
   macro avg       0.74      0.71      0.71       154
weighted avg       0.74      0.74      0.73       154
```

In [147]:

```
model_obj = gausian_process
model_name = "Gausian Process"
process = "G P"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[147]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014
0	SGD	Stochastic Gradient Descent	[0.57, 0.64, 0.7, 0.74, 0.74]	0.57-0.74	0.072938
0	G P	Gausian Process	[0.7, 0.73, 0.74, 0.75, 0.77]	0.7-0.77	0.025884

In [ ]:

```

```

KNN Classifier

In [148]:

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

In [149]:

```
knn.fit(train_X, train_y)
test_ds_predicted = knn.predict( test_X )
```

In [150]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.71      0.85      0.77        93
           1       0.67      0.48      0.56        61

    accuracy                           0.70       154
   macro avg       0.69      0.66      0.67       154
weighted avg       0.70      0.70      0.69       154
```

In [151]:

```
model_obj = knn
model_name = "K Nearst Neighbour"
process = "KNN C"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[151]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014
0	SGD	Stochastic Gradient Descent	[0.57, 0.64, 0.7, 0.74, 0.74]	0.57-0.74	0.072938
0	G P	Gausian Process	[0.7, 0.73, 0.74, 0.75, 0.77]	0.7-0.77	0.025884
0	KNN C	K Nearst Neighbour	[0.69, 0.71, 0.72, 0.74, 0.75]	0.69-0.75	0.023875

In [ ]:

```

```

Linear Discriminant Analysis

In [152]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
```

In [153]:

```
lda.fit(train_X, train_y)
test_ds_predicted = lda.predict( test_X )
```

In [154]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.75      0.83      0.79        93
           1       0.69      0.59      0.64        61

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

In [155]:

```
model_obj = lda
model_name = "Linear Discriminant Analysis"
process = "LDA"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[155]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014
0	SGD	Stochastic Gradient Descent	[0.57, 0.64, 0.7, 0.74, 0.74]	0.57-0.74	0.072938
0	G P	Gausian Process	[0.7, 0.73, 0.74, 0.75, 0.77]	0.7-0.77	0.025884
0	KNN C	K Nearst Neighbour	[0.69, 0.71, 0.72, 0.74, 0.75]	0.69-0.75	0.023875
0	LDA	Linear Discriminant Analysis	[0.7, 0.77, 0.77, 0.77, 0.8]	0.7-0.8	0.037014

In [ ]:

```

```

Support Vector Machine

In [156]:

```
from sklearn.svm import SVC
svm = SVC()
```

In [157]:

```
svm.fit(train_X, train_y)
test_ds_predicted = svm.predict( test_X )
```

In [158]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81        93
           1       0.73      0.61      0.66        61

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

In [159]:

```
model_obj = svm
model_name = "Support Vector Machine"
process = "SVM"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[159]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014
0	SGD	Stochastic Gradient Descent	[0.57, 0.64, 0.7, 0.74, 0.74]	0.57-0.74	0.072938
0	G P	Gausian Process	[0.7, 0.73, 0.74, 0.75, 0.77]	0.7-0.77	0.025884
0	KNN C	K Nearst Neighbour	[0.69, 0.71, 0.72, 0.74, 0.75]	0.69-0.75	0.023875
0	LDA	Linear Discriminant Analysis	[0.7, 0.77, 0.77, 0.77, 0.8]	0.7-0.8	0.037014
0	SVM	Support Vector Machine	[0.69, 0.73, 0.74, 0.78, 0.78]	0.69-0.78	0.037815

In [ ]:

```

```

In [160]:

```
#df_model_selection.to_csv("../data/intermediate/Model_statistics.csv",index = False)
```

In [161]:

```
# Binomial Naive Bayes Classifier looks better than the other models with F1 score od
 0.014832
```

In [162]:

```
lgbm = LGBMClassifier(boosting_type='goss',   # Gradient-based One-Side Sampling
                      n_jobs=-1,
                      objective='binary',
                      random_state=9,
                      importance_type='split'
                      )
```

In [163]:

```
lgbm.fit(train_X, train_y)
test_ds_predicted = lgbm.predict(test_X)
print('lgbm predicted accuracy score is : ', accuracy_score (y_pred_lgbm,y_test))
```

```
lgbm predicted accuracy score is :  0.7597402597402597
```

In [164]:

```
from sklearn import metrics
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.78      0.82      0.80        93
           1       0.70      0.66      0.68        61

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

In [165]:

```
model_obj = lgbm
model_name = "LGBMClassifier"
process = "LGBMC"
n_splits = 5
X = train_ds_features
y = train_ds_result
stratified_K_fold_validation(model_obj, model_name, process, n_splits, X, y)
df_model_selection
```

Out[165]:

	Process	Model Name	F1 Scores	Range of F1 Scores	Std Deviation of F1 Scores
0	B-NBC	Binomial Naive Bayes Classifier	[0.67, 0.67, 0.74, 0.74, 0.75]	0.67-0.75	0.040373
0	L R	Logistic Regression	[0.71, 0.75, 0.77, 0.77, 0.81]	0.71-0.81	0.036332
0	DT	Decission Tree	[0.65, 0.69, 0.69, 0.7, 0.72]	0.65-0.72	0.025495
0	R F	Random Forest	[0.69, 0.72, 0.73, 0.75, 0.79]	0.69-0.79	0.037148
0	XG B	XG Boost	[0.69, 0.71, 0.72, 0.76, 0.78]	0.69-0.78	0.037014
0	SGD	Stochastic Gradient Descent	[0.57, 0.64, 0.7, 0.74, 0.74]	0.57-0.74	0.072938
0	G P	Gausian Process	[0.7, 0.73, 0.74, 0.75, 0.77]	0.7-0.77	0.025884
0	KNN C	K Nearst Neighbour	[0.69, 0.71, 0.72, 0.74, 0.75]	0.69-0.75	0.023875
0	LDA	Linear Discriminant Analysis	[0.7, 0.77, 0.77, 0.77, 0.8]	0.7-0.8	0.037014
0	SVM	Support Vector Machine	[0.69, 0.73, 0.74, 0.78, 0.78]	0.69-0.78	0.037815
0	LGBMC	LGBMClassifier	[0.69, 0.72, 0.73, 0.74, 0.79]	0.69-0.79	0.036469

In [166]:

```
df_model_selection.to_csv("../data/intermediate/Model_statistics.csv",index = False)
```

In [ ]:

```
```