Project Title: HEALTH CARE

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

* The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

* Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description

- Pregnancies Number of times pregnant
- Glucose Plasma glucose concentration in an oral glucose tolerance test
- BloodPressure Diastolic blood pressure (mm Hg)
- SkinThickness Triceps skinfold thickness (mm)
- Insulin Two hour serum insulin
- BMI Body Mass Index
- DiabetesPedigreeFunction Diabetes pedigree function
- Age Age in years
- Outcome Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Project Task: Week 1 DESCRIPTIVE ANALYSIS

Data Exploration:

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

1. Visually explore these variables using histograms. Treat the missing values accordingly.
2. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

Understanding the Data(Overview):

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases(NIDDK).

- The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.
- Several constraints were placed on the selection of these instances from a larger database.
- Build a model to accurately predict whether the patients in the dataset have diabetes or not.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

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

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	O
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

Insight: From the above data, it is noted that there are eight independent variables (Pregnancies, Glucose, Blood Pressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age) and one dependent variable (Outcome).

- Here, we need to predict outcome column. 1 indicate person is diabetes and 0 denote person is non-diabetes.

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [5]:
```
df.Outcome.value_counts(normalize=True)
```

Out[5]:
```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

In [76]:
```
for i in ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI']:
    print(i)
    print (df[i].value_counts().head())
```

```
Glucose
99.0     17
100.0    17
111.0    14
129.0    14
125.0    14
Name: Glucose, dtype: int64
BloodPressure
70.0    57
74.0    52
78.0    45
68.0    45
72.0    44
Name: BloodPressure, dtype: int64
SkinThickness
20.536458    227
32.000000     31
30.000000     27
27.000000     23
23.000000     22
Name: SkinThickness, dtype: int64
Insulin
79.799479     374
105.000000     11
130.000000      9
140.000000      9
120.000000      8
Name: Insulin, dtype: int64
BMI
32.000000    13
31.600000    12
31.200000    12
31.992578    11
32.400000    10
Name: BMI, dtype: int64
```

In [7]:
```
df[df[['Pregnancies','Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']]==0].count()
```

```
Pregnancies                 111
```

Out[7]:
```
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                   500
dtype: int64
```

In [8]:
```
df.dtypes
```

Out[8]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [9]:
```
df.shape
```

Out[9]:
```
(768, 9)
```

In [10]:
```
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
```

Insight: The above data information shows the number of rows, number of columns, data types information, Memory usage, number of null values in each column.

In [11]:
```
df.corr()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	

In [12]:
```
df.describe()
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
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	

Insight: The data consists of 768 observations of 9 variables.

** Independent variables are Pregnencies , Glucose, BloodPressure, Insulin, BMI and DiabetesPedigree Function.

** Age is Outcome Variable.

** Average Age of Patients are 33.24 with minimum being 21 and maximum 81.

** Avg. value of independent variables are Preg = 3.845052,Glucose = 120.894531, BP = 69.105469, ST=20.536458, Insulin = 79.799479, BMI = 31.992578 DPF = 0.471876 .

In [13]:
```
# Let's try to observe the variation in variables:

print("Standard Deviation of each variables are: ")
df.apply(np.std)
```

Out[13]:
```
Standard Deviation of each variables are:
Pregnancies                 3.367384
Glucose                    31.951796
```

```
BloodPressure               19.343202
SkinThickness               15.941829
Insulin                    115.168949
BMI                          7.879026
DiabetesPedigreeFunction     0.331113
Age                         11.752573
Outcome                      0.476641
dtype: float64
```

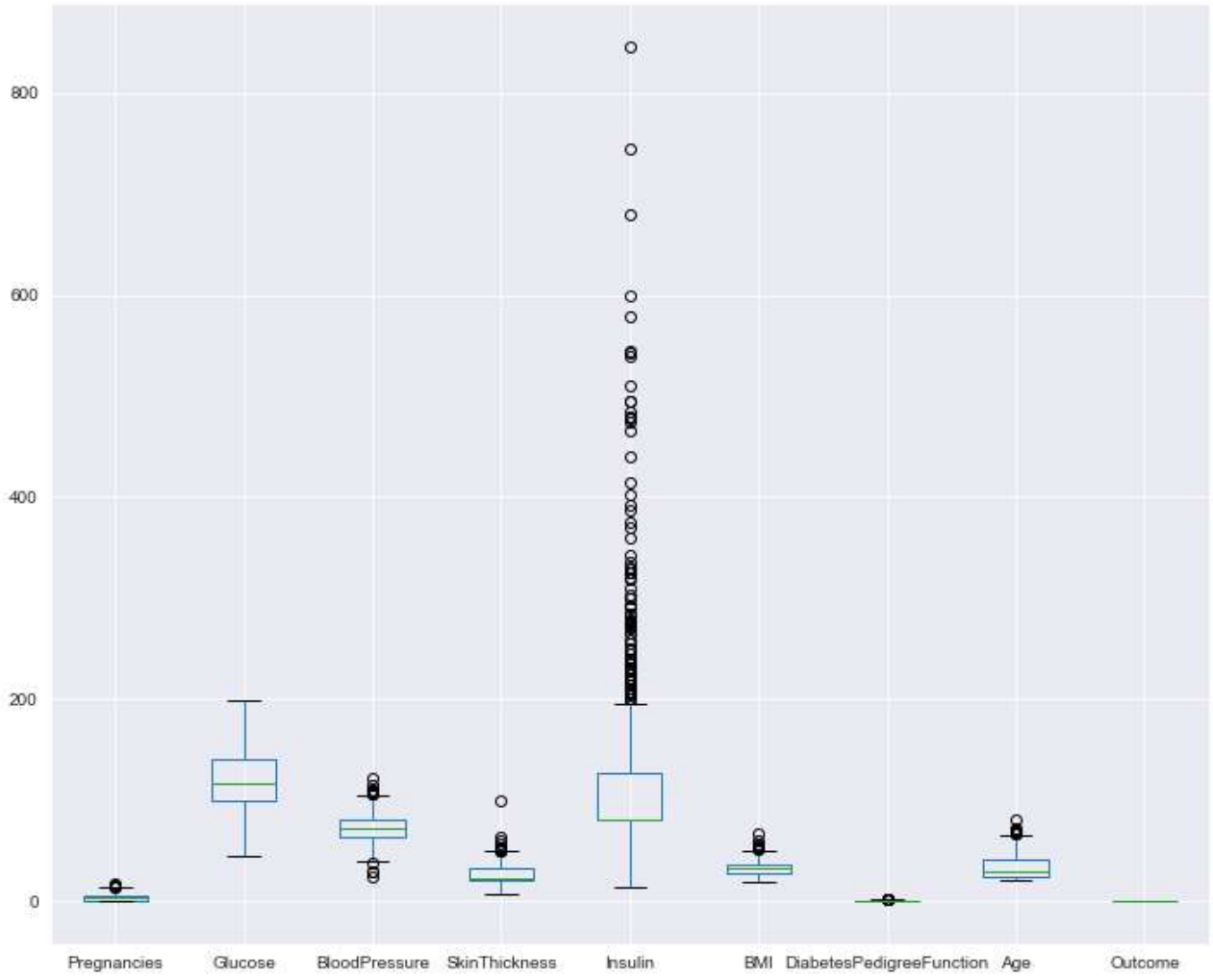
In [14]:
```
# Visually exploring the variables using histograms by treating the missing values acco

df.isnull().any()
```

Out[14]:
```
Pregnancies               False
Glucose                   False
BloodPressure             False
SkinThickness             False
Insulin                   False
BMI                       False
DiabetesPedigreeFunction  False
Age                       False
Outcome                   False
dtype: bool
```

Insight: The above finding shows that there are no null values (missing values) in the dataset. But, it does not make sense. It seems very likely that zero values encode missing data.We replace 0 by NaN values to count the missing values.

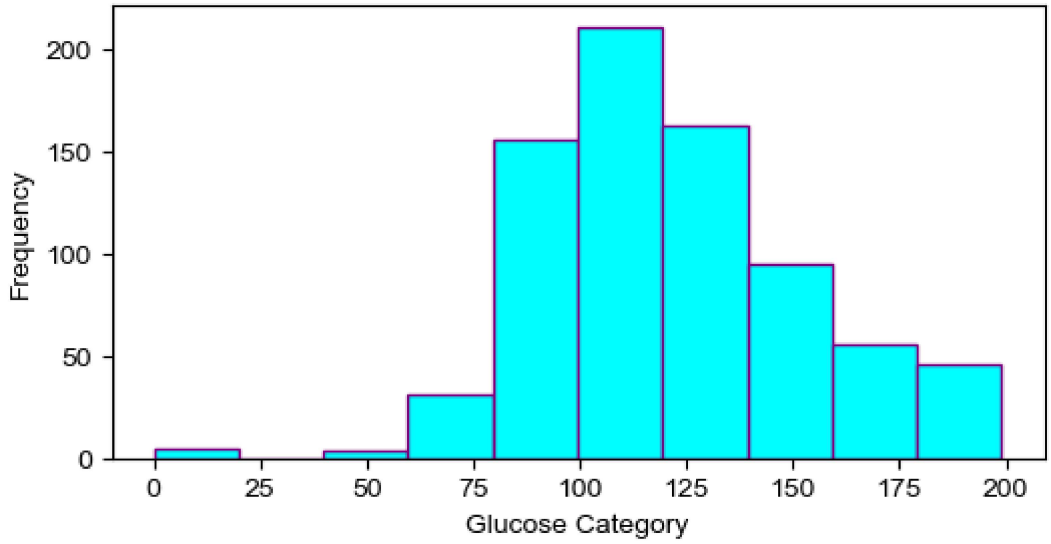
In [83]:
```
plt.figure(figsize=(12,10))
df.boxplot()
```

Out[83]:
```
<AxesSubplot:>
```



In [15]:
```
plt.figure(figsize=(6,3),dpi=100)
plt.xlabel('Glucose Category')
df['Glucose'].plot.hist(df, color = "cyan", edgecolor="purple")
sns.set_style(style='darkgrid')
print("Mean of Glucose level is :", df['Glucose'].mean())
print("Datatype of Glucose Variable is:",df['Glucose'].dtypes)
```

```
Mean of Glucose level is : 120.89453125
Datatype of Glucose Variable is: int64
```



In [16]:
```
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
```

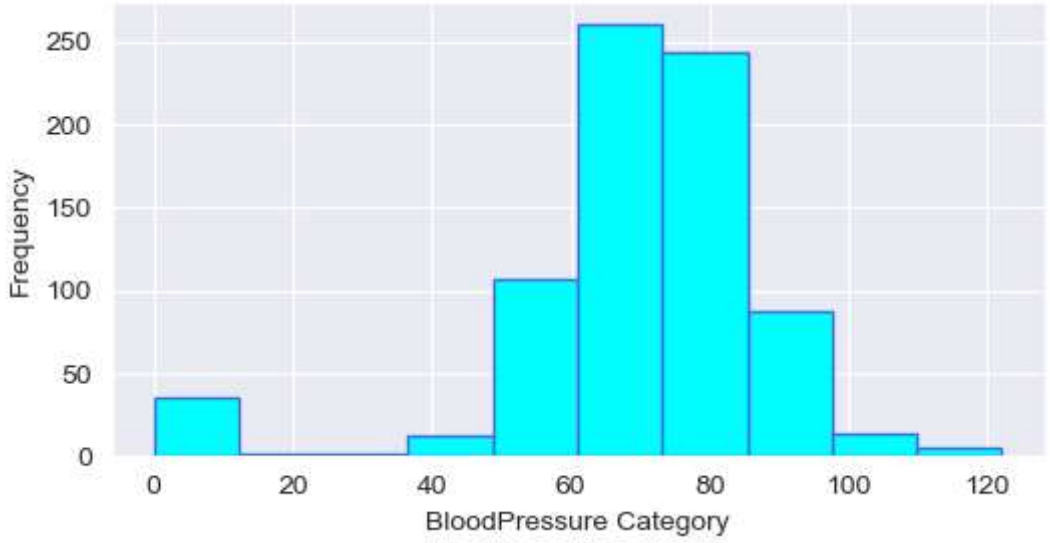
Insight: Assuming the missing values are 0 by mean of Glucose Level. From the above hist, it is observed that the highest glucose level lies between 100 and 200.

In [17]:
```
# BloodPressure Level:

df['BloodPressure'].value_counts().head(5)
```

Out[17]:
```
70    57
74    52
78    45
68    45
72    44
Name: BloodPressure, dtype: int64
```

In [18]:
```
plt.figure(figsize=(6,3),dpi=100)
plt.xlabel('BloodPressure Category')
df['BloodPressure'].plot.hist(df, color = "cyan", edgecolor="royalblue")
sns.set_style(style='darkgrid')
print("Mean of BloodPressure level is :", df['BloodPressure'].mean())
print("Datatype of BloodPressure Variable is:",df['BloodPressure'].dtypes)
```

```
Mean of BloodPressure level is : 69.10546875
Datatype of BloodPressure Variable is: int64
```



In [19]:
```
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
```

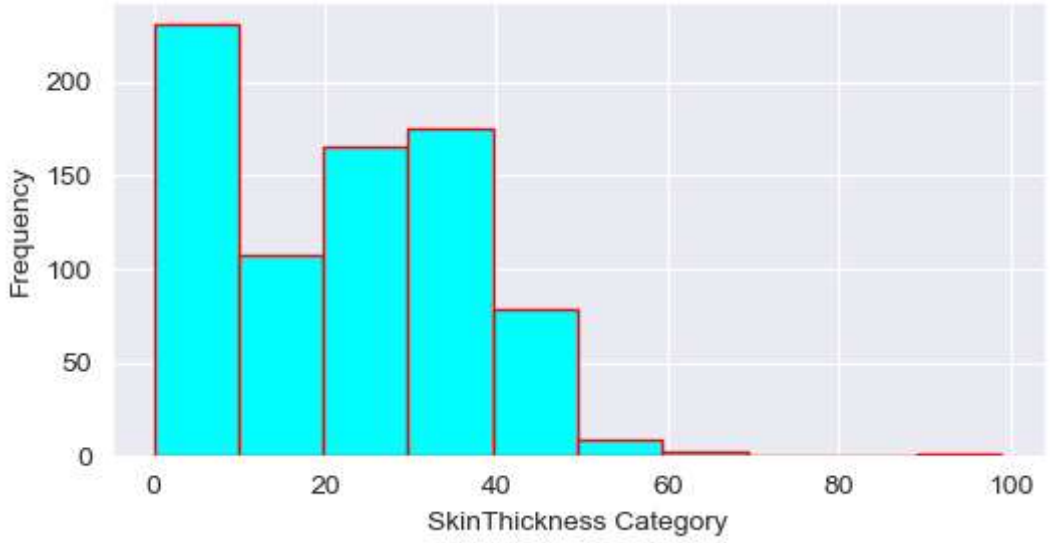
Insight: Assuming the missing values are 0 by the mean of BloodPressure Level. From the above hist, it is observed that the highest BloodPressure level lies between 70 and 80.

In [20]:
```
# SkinThickness Level:
df['SkinThickness'].value_counts().head(5)
```

Out[20]:
```
0     227
32     31
30     27
27     23
23     22
Name: SkinThickness, dtype: int64
```

In [21]:
```
plt.figure(figsize=(6,3),dpi=100)
plt.xlabel('SkinThickness Category')
df['SkinThickness'].plot.hist(df, color = "cyan", edgecolor="red")
sns.set_style(style='darkgrid')
print("Mean of SkinThickness is :", df['SkinThickness'].mean())
print("Datatype of SkinThickness Variable is:",df['SkinThickness'].dtypes)
```

```
Mean of SkinThickness is : 20.536458333333332
Datatype of SkinThickness Variable is: int64
```



In [22]:
```
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
```

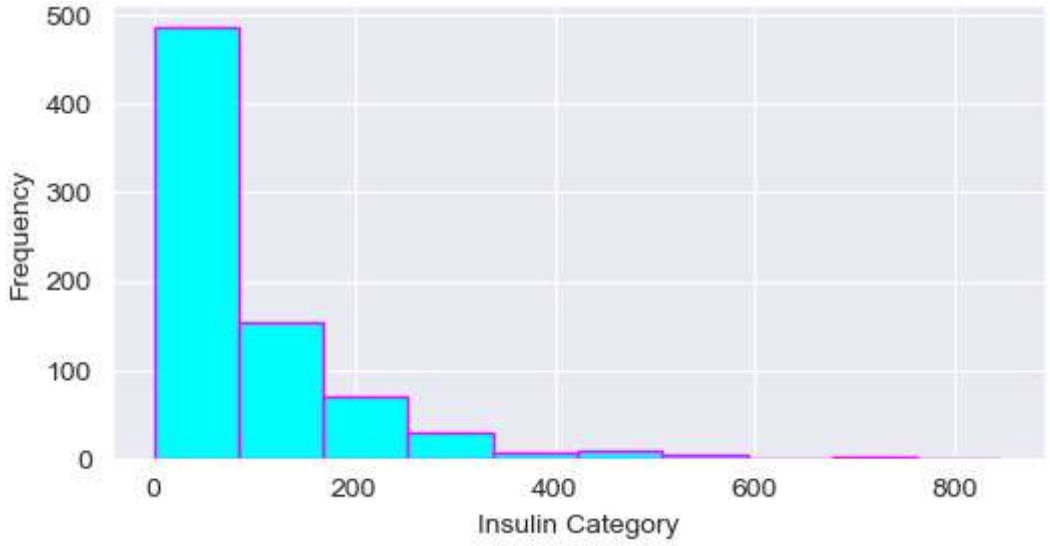
Insight: Assuming the missing values are 0 by the mean of SkinThickness Level. From the above hist, it is observed that the highest SkinThickness level lies between 20 and 30.

In [23]:
```
# Insulin Level
df['Insulin'].value_counts().head(5)
```

Out[23]:
```
0      374
105     11
130      9
140      9
120      8
Name: Insulin, dtype: int64
```

In [24]:
```
plt.figure(figsize=(6,3),dpi=100)
plt.xlabel('Insulin Category')
df['Insulin'].plot.hist(df, color = "cyan", edgecolor="magenta")
sns.set_style(style='darkgrid')
print("Mean of Insulin is :", df['Insulin'].mean())
print("Datatype of Insulin Variable is:",df['Insulin'].dtypes)
```

```
Mean of Insulin is : 79.79947916666667
Datatype of Insulin Variable is: int64
```



In [25]:
```
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
```

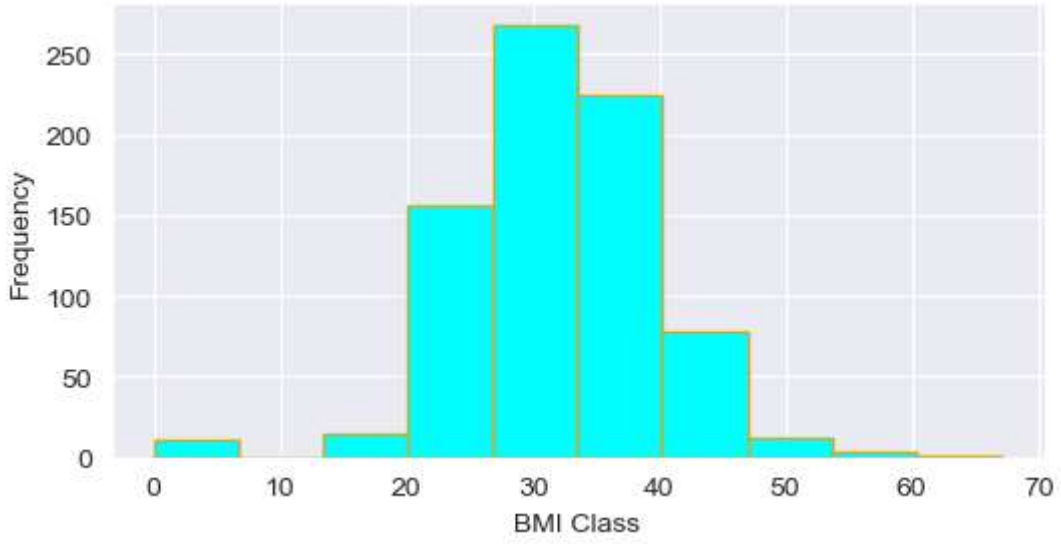
Insight: Assuming the missing values are 0 by the mean of Insulin Level. From the above hist, it is observed that the highest Insulin level lies between 0 and 10.

In [26]:
```
# BMI Level
df['BMI'].value_counts().head(5)
```

Out[26]:
```
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
Name: BMI, dtype: int64
```

In [27]:
```
plt.figure(figsize=(6,3),dpi=100)
plt.xlabel('BMI Class')
df['BMI'].plot.hist(df, color = "cyan", edgecolor="goldenrod")
sns.set_style(style='darkgrid')
print("Mean of BMI is :", df['BMI'].mean())
print("Datatype of BMI Variable is:",df['BMI'].dtypes)
```

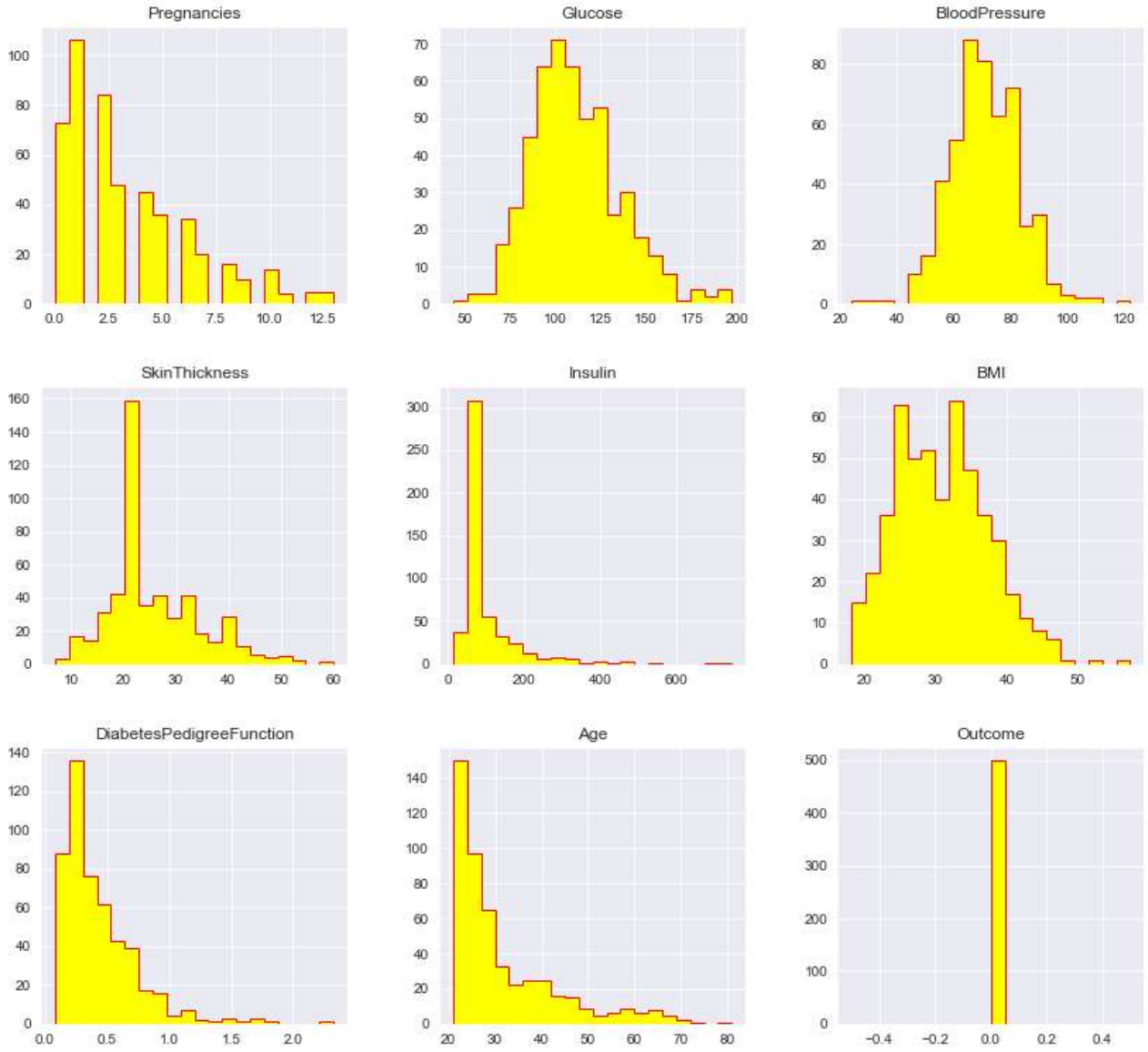
```
Mean of BMI is : 31.992578124999977
Datatype of BMI Variable is: float64
```

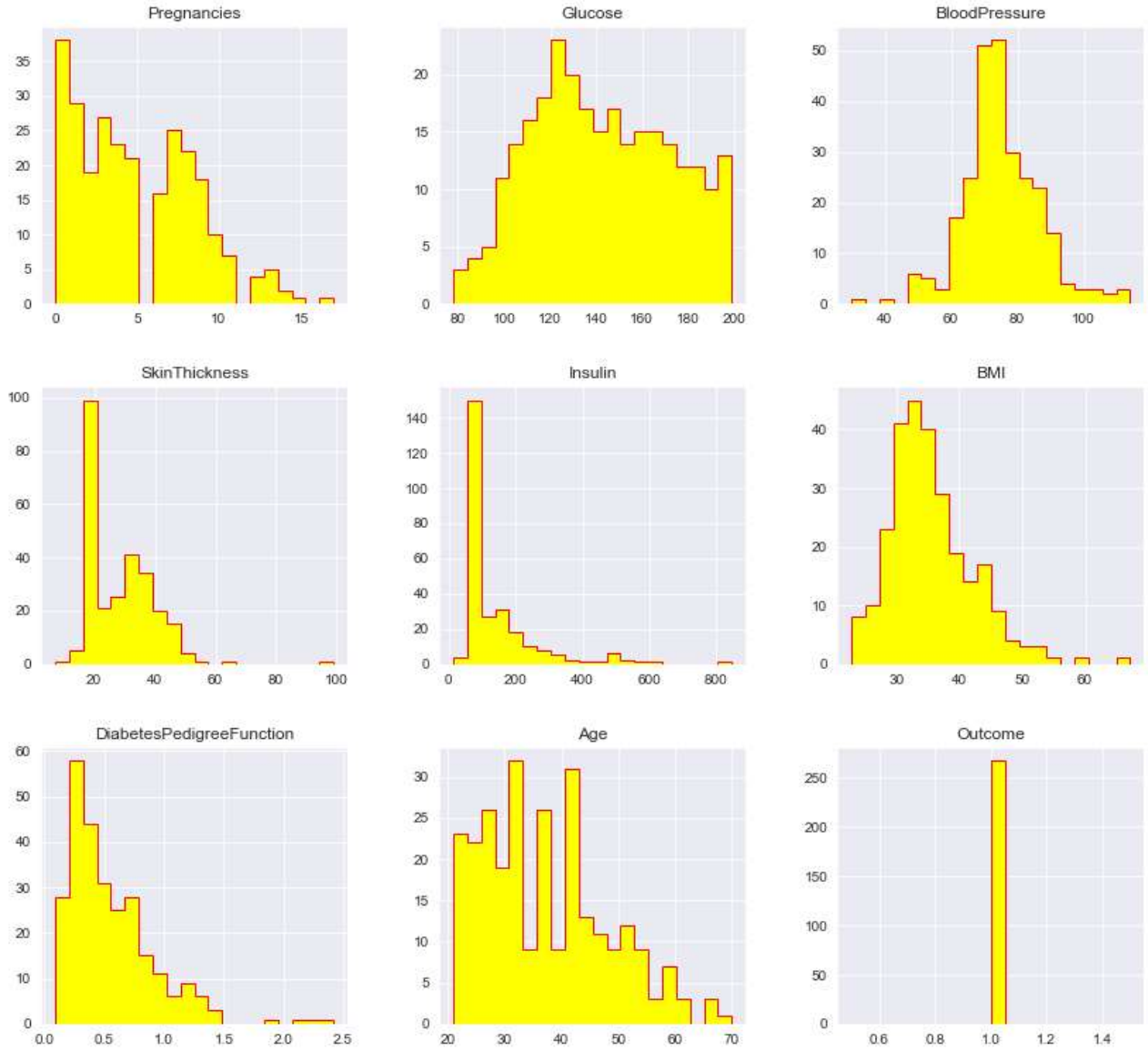


In [28]:
```
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

In [29]:
```
# Let's check with each category at once...in a column hist...

df.groupby('Outcome').hist(figsize=(14, 13), histtype='stepfilled',bins=20,color="yello
```

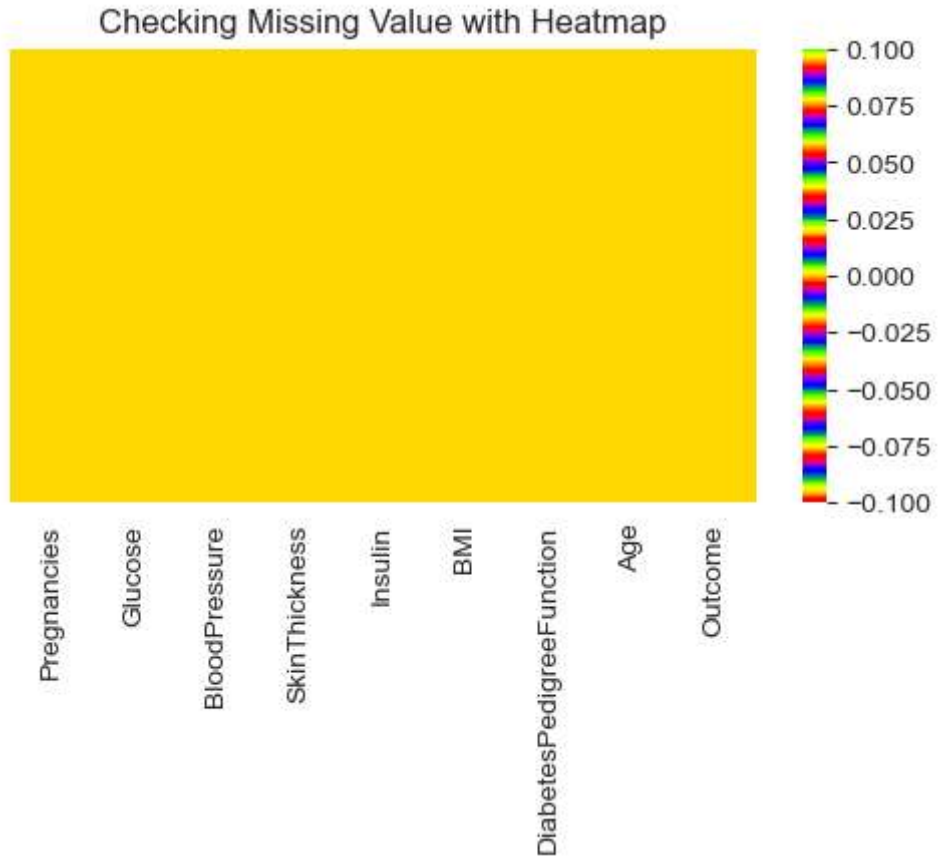
Out[29]:
```
Outcome
0    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
1    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
dtype: object
```


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome



In [30]:
```
# Checking Missing Values by using heatmap

plt.figure(figsize=(6,3),dpi=100)
plt.title('Checking Missing Value with Heatmap')
sns.heatmap(df.isnull(),cmap='prism',yticklabels=False)
```

Out[30]:
```
<AxesSubplot:title={'center':'Checking Missing Value with Heatmap'}>
```



In [31]:
```
df.head()
```

Out[31]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	5(
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

In [32]:
```
df.tail()
```

Out[32]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
763	10	101.0	76.0	48.000000	180.000000	32.9	0.171	
764	2	122.0	70.0	27.000000	79.799479	36.8	0.340	
765	5	121.0	72.0	23.000000	112.000000	26.2	0.245	
766	1	126.0	60.0	20.536458	79.799479	30.1	0.349	
767	1	93.0	70.0	31.000000	79.799479	30.4	0.315	

In [33]:
```
# Replacing 0 to NaN

Newdf=df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]=df[['Glucose','Bl
Newdf.head()
```

Out[33]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
0	148.0	72.0	35.000000	79.799479	33.6
1	85.0	66.0	29.000000	79.799479	26.6
2	183.0	64.0	20.536458	79.799479	23.3
3	89.0	66.0	23.000000	94.000000	28.1
4	137.0	40.0	35.000000	168.000000	43.1

In [34]:
```
Newdf.isnull().sum()[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
```

Out[34]:
```
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
dtype: int64
```

In [35]:
```
# Replacing NaN to mean value to explore the dataset

df['Glucose'].fillna(df['Glucose'].median(),inplace=True)
df['BloodPressure'].fillna(df['BloodPressure'].median(),inplace=True)
df['SkinThickness'].fillna(df['SkinThickness'].median(),inplace=True)
df['Insulin'].fillna(df['Insulin'].median(),inplace=True)
df['BMI'].fillna(df['BMI'].median(),inplace=True)
df.head()
```

Out[35]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

In [36]:
```
df.groupby('Outcome').size()
```

Out[36]:
```
Outcome
0    500
1    268
dtype: int64
```

Project Task: Week 2 EDA

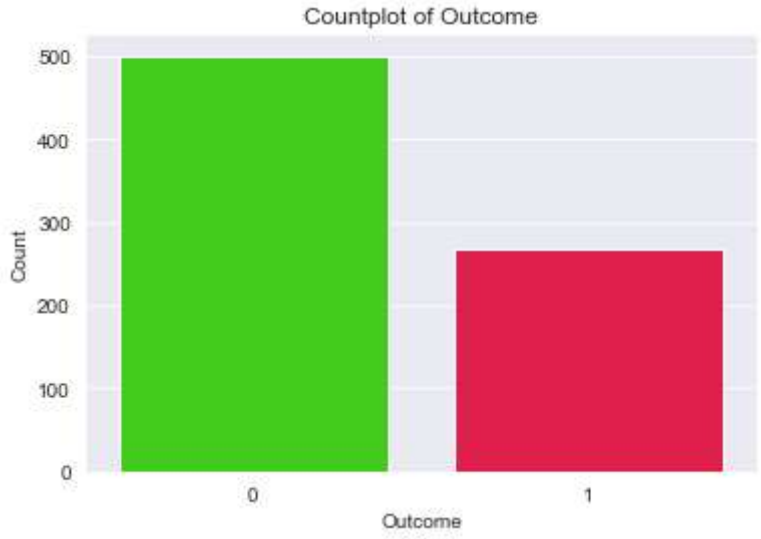
Data Exploration:

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

Countplot

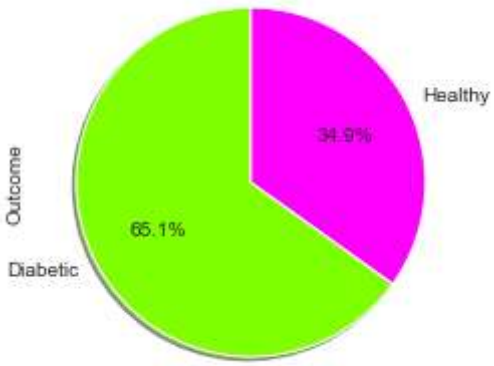
In [37]:
```
sns.set_style('darkgrid')
sns.countplot(df['Outcome'], palette = "prism")
plt.title("Countplot of Outcome")
plt.xlabel('Outcome')
plt.ylabel("Count")
print("Count of class is:\n",df['Outcome'].value_counts())
```

```
Count of class is:
 0    500
1    268
Name: Outcome, dtype: int64
```



In [38]:
```
#Find the % of diabetic and Healthy person

labels = 'Diabetic', 'Healthy'

df.Outcome.value_counts().plot.pie(labels=labels, autopct='%1.1f%%',shadow=True, starta
```

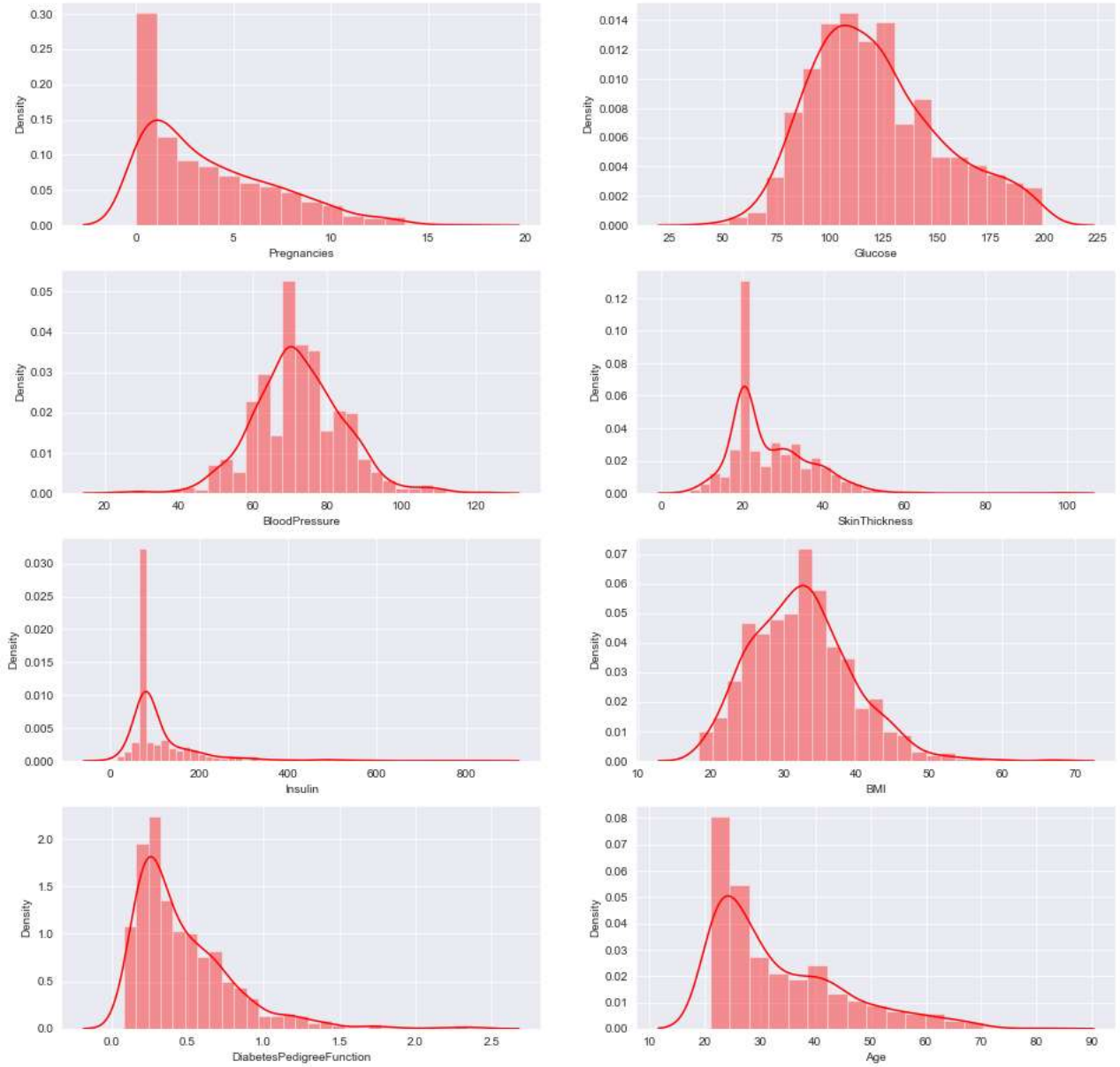
Out[38]:
```
<AxesSubplot:ylabel='Outcome'>
```



Insight: The above pieplot shows that 65.1% people are diabetic and 34.9% people are Healthy.

In [39]:
```
# Distplot

fig, ax2 = plt.subplots(4, 2, figsize=(16, 16))
sns.distplot(df['Pregnancies'],ax=ax2[0][0],color="red")
sns.distplot(df['Glucose'],ax=ax2[0][1], color="red")
sns.distplot(df['BloodPressure'],ax=ax2[1][0], color="red")
sns.distplot(df['SkinThickness'],ax=ax2[1][1], color="red")
sns.distplot(df['Insulin'],ax=ax2[2][0], color="red")
sns.distplot(df['BMI'],ax=ax2[2][1], color="red")
sns.distplot(df['DiabetesPedigreeFunction'],ax=ax2[3][0], color="red")
sns.distplot(df['Age'],ax=ax2[3][1], color="red")
```

Out[39]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```



Insight: The above Dist Plots show that Glucose, Blood Pressure, BMI are normally distributed whereas Pregnancies, Insulin, Age, DiabetesPedigreeFunction are rightly skewed.

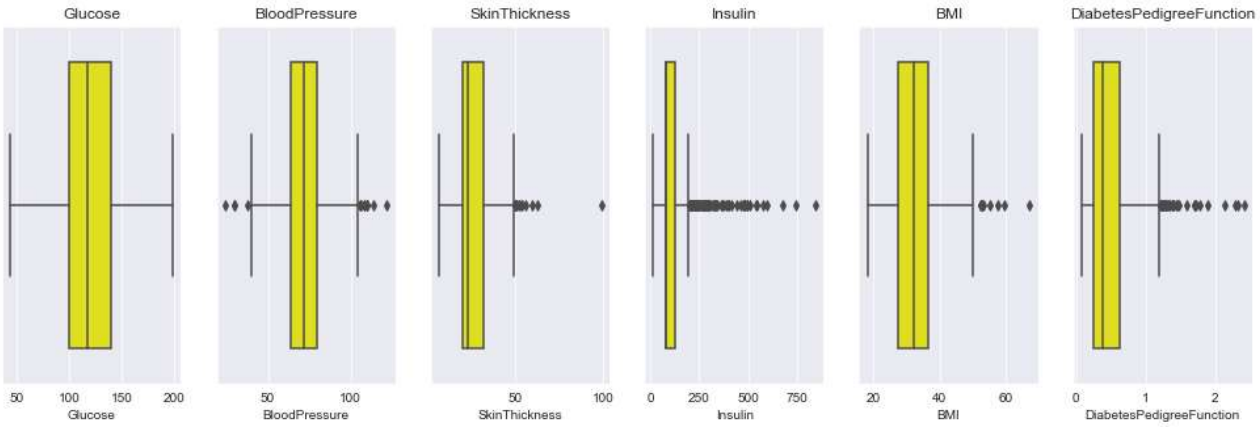
In [40]:
```
df.describe().transpose()
```

Out[40]:

	count	mean	std	min	25%	50%	75%	ma
Pregnancies	768.0	3.845052	3.369578	0.000	1.000000	3.000000	6.00000	17.0
Glucose	768.0	121.681605	30.436016	44.000	99.750000	117.000000	140.25000	199.0
BloodPressure	768.0	72.254807	12.115932	24.000	64.000000	72.000000	80.00000	122.0
SkinThickness	768.0	26.606479	9.631241	7.000	20.536458	23.000000	32.00000	99.0
Insulin	768.0	118.660163	93.080358	14.000	79.799479	79.799479	127.25000	846.0
BMI	768.0	32.450805	6.875374	18.200	27.500000	32.000000	36.60000	67.1
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.243750	0.372500	0.62625	2.4
Age	768.0	33.240885	11.760232	21.000	24.000000	29.000000	41.00000	81.0

	count	mean	std	min	25%	50%	75%	ma
Outcome	768.0	0.348958	0.476951	0.000	0.000000	0.000000	1.00000	1.0

In [41]:
```
# Checking for the Outliers

fig=plt.figure(figsize=(20,5))

for i in np.arange(1,7):
    data3=plt.subplot(1,7,i,title=df.columns[i])
    sns.boxplot(df[df.columns[i]], color="yellow")
```



Insight:The plot shows outliers present in the dataset.

- BloodPressure, SkinThickness, Insulin, BMI & DiabetesPedigreeFunction have outliers. It is clearly visible in the boxplots.

Scatter Plot

Pairplot - multiple relation of Scatterplot for positive outcome

In [42]:
```
sns.pairplot(df,hue='Outcome')
```

Out[42]:
```
<seaborn.axisgrid.PairGrid at 0x1a452e1b910>
```

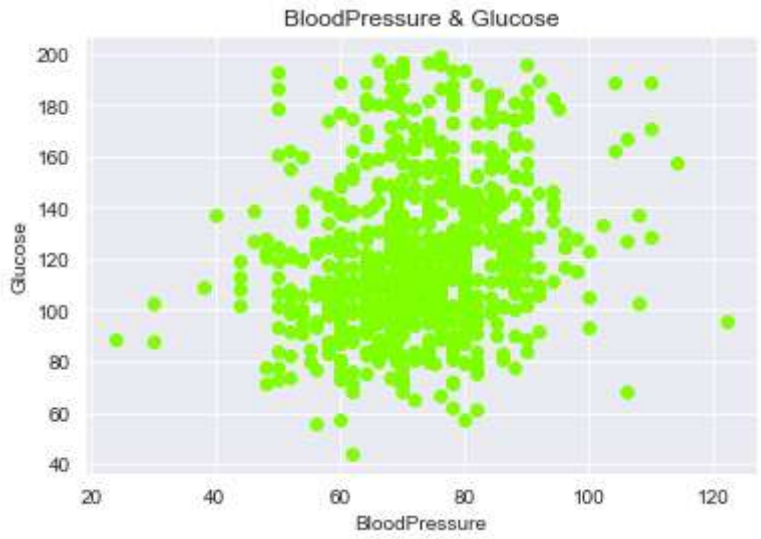
Insight:The plot shows that there is some relationship between parameters. Outcome is added as hue. We see that blue and orange dots are overlap. Also,

- Pregnancies and age have a kind of a linear line.
- BloodPressure and age have little relation. Most of the aged people have BloodPressure.
- Insulin and Glucose have some relation. We can see that there is no strong multicolinearity among features, but between skin thickness and BMI, Pregnancies and age it looks like there is small chance of positive correlation.

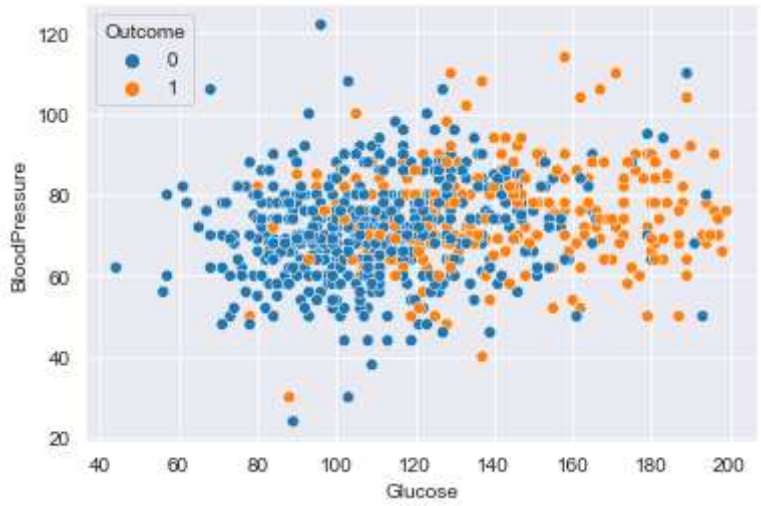
In [84]:
```
BloodPressure = df['BloodPressure']
Glucose = df['Glucose']
SkinThickness = df['SkinThickness']
Insulin = df['Insulin']
BMI = df['BMI']
```

In [88]:
```
plt.scatter(BloodPressure, Glucose, color=['chartreuse'])
plt.xlabel('BloodPressure')
plt.ylabel('Glucose')
```

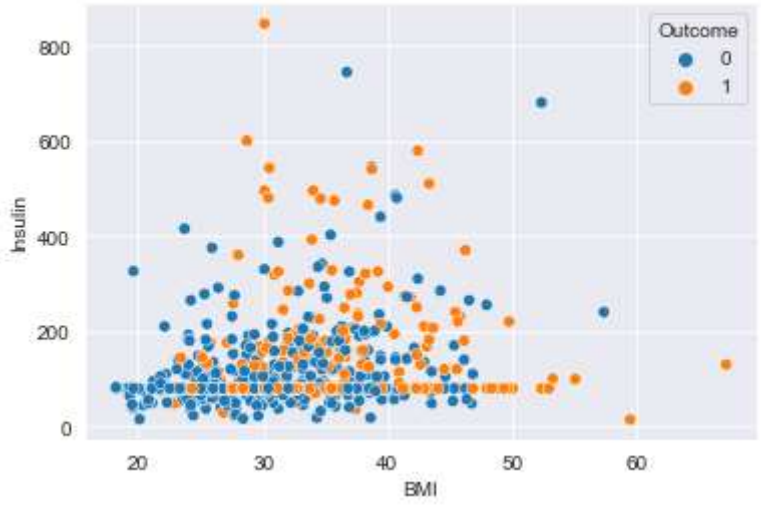
```
plt.title('BloodPressure & Glucose')
plt.show()
```



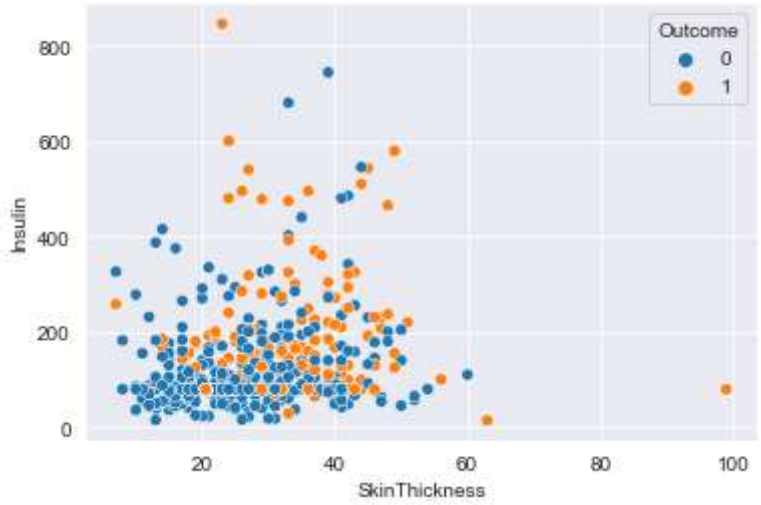
In [91]:
```
glu =sns.scatterplot(x= "Glucose" ,y= "BloodPressure", hue="Outcome", data=df);
```



In [92]:
```
Blp =sns.scatterplot(x= "BMI" ,y= "Insulin",hue="Outcome", data=df);
```



In [93]:
```
Skt =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",hue="Outcome", data=df);
```



Correlation Analysis:

In [43]:
```
df.corr()
```

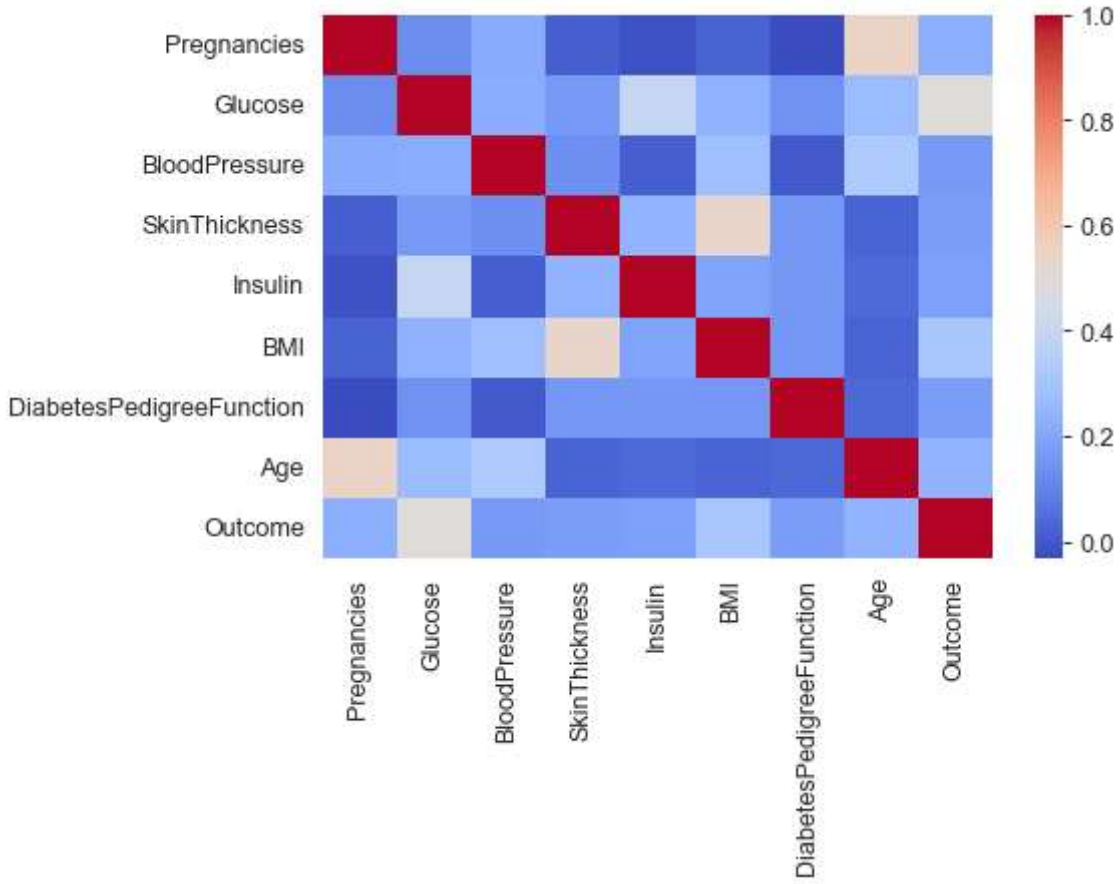
Out[43]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Pregnancies	1.000000	0.127964	0.208984	0.013376	-0.018082	0.021546	
Glucose	0.127964	1.000000	0.219666	0.160766	0.396597	0.231478	
BloodPressure	0.208984	0.219666	1.000000	0.134155	0.010926	0.281231	
SkinThickness	0.013376	0.160766	0.134155	1.000000	0.240361	0.535703	
Insulin	-0.018082	0.396597	0.010926	0.240361	1.000000	0.189856	
BMI	0.021546	0.231478	0.281231	0.535703	0.189856	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.154961	0.157806	0.153508	
Age	0.544341	0.266600	0.326740	0.026423	0.038652	0.025748	
Outcome	0.221898	0.492908	0.162986	0.175026	0.179185	0.312254	

Insight: We can perfectly notice that Glucose (0.492908) and BMI (0.312254) has good impact on the outcome. There is a strong positive correlation between BMI and Skinthickness or Pregnancies and age.

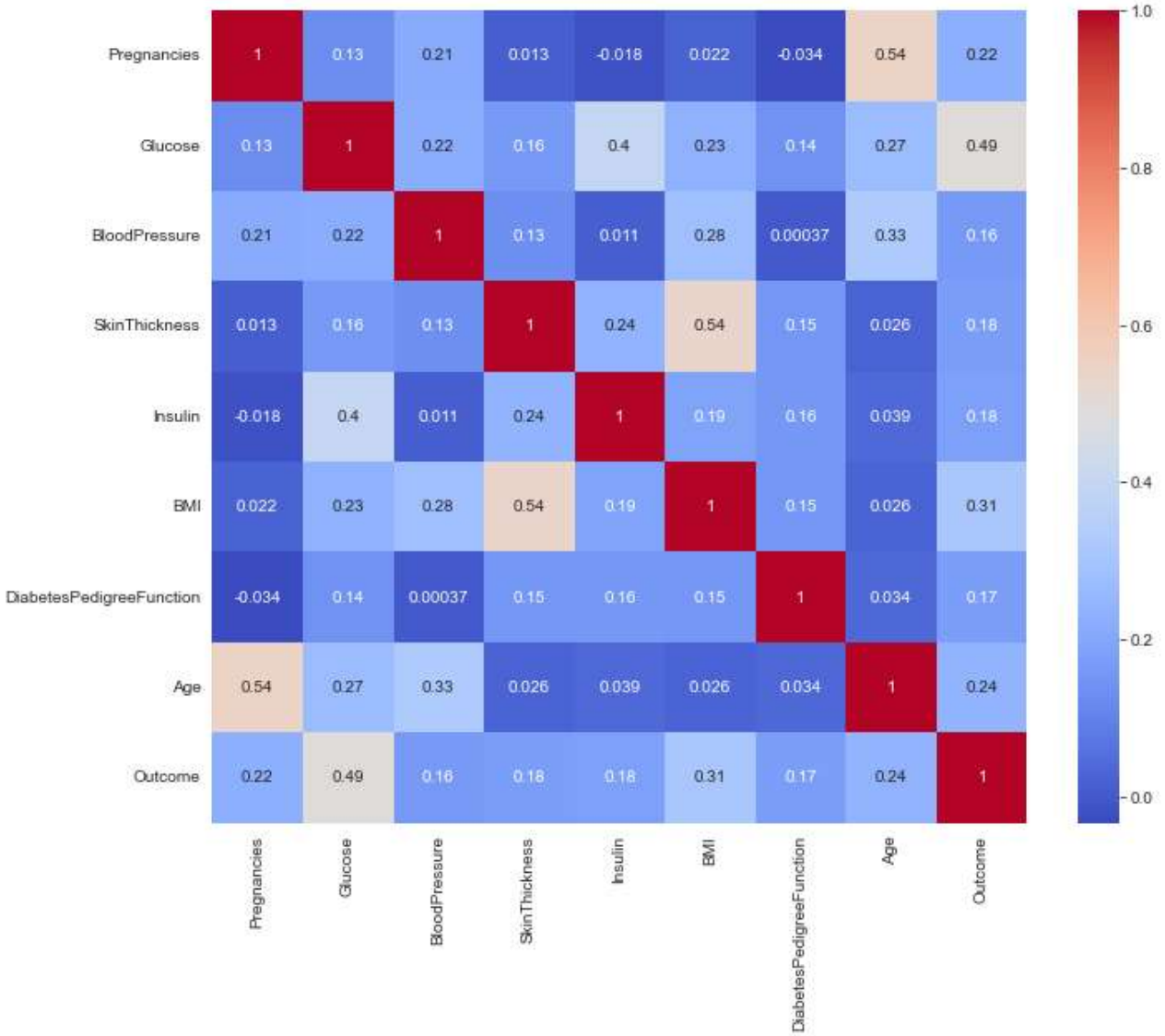
In [44]:
```
plt.figure(dpi=90)
sns.heatmap(df.corr(),cmap='coolwarm')
```

Out[44]:
```
<AxesSubplot:>
```



In [45]:
```
plt.subplots(figsize=(11,9))
sns.heatmap(df.corr(),annot=True,cmap='coolwarm')  ### gives correlation value
```

Out[45]:
```
<AxesSubplot:>
```



Insight:

The above correlation plot shows the relation between the parameters.

- Glucose,Age,BMI and Pregnancies are the most correlated parameters with the Outcome.
- Insulin and DiabetesPedigreeFunction have little correlation with the outcome.
- BloodPressure and SkinThickness have tiny correlation with the outcome.
- There is a little correlation between Age and Pregnancies,Insulin and Skin Thickness, BMI and Skin Thickness,Insulin and Glucose

Project Task: Week 3 & Week 4

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

In [100…
```
x=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

In [47]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=0)
```

In [48]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614,)
(154,)
```

In [49]:
```
from sklearn.preprocessing import StandardScaler
```

In [50]:
```
Scale=StandardScaler()
x_train_std=Scale.fit_transform(x_train)
x_test_std=Scale.transform(x_test)
```

In [51]:
```
norm=lambda a:(a-min(a))/(max(a)-min(a))
```

In [52]:
```
df_norm=df.iloc[:,:-1]
```

In [53]:
```
df_normalized=df_norm.apply(norm)
```

In [54]:
```
x_train_norm,x_test_norm,y_train_norm,y_test_norm=train_test_split(df_normalized.values
```

In [55]:
```
print(x_train_norm.shape)
print(x_test_norm.shape)
print(y_train_norm.shape)
print(y_test_norm.shape)
```

```
(614, 8)
(154, 8)
```

```
(614,)
(154,)
```

Note: Since the variables are linearly dependent on the target and the data is mostly in numeric form, it is compatible for LOGISTIC REGRESSION as these learning algorithms works well on linear data.

Let's verify the accuracy rate with KNN(standardization as well as normilization), Random Forest, SVC Model besides Logistic Regression

KNN with Standard Scaling

In [56]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
knn_model = KNeighborsClassifier(n_neighbors=25) # In general practice, choosing the va
knn_model.fit(x_train_std,y_train)
knn_pred=knn_model.predict(x_test_std)
```

In [57]:
```
features = df.iloc[:,[0,1,2,3,4,5,6,7]].values
label = df.iloc[:,8].values
```

In [58]:
```
print(confusion_matrix(y_test, knn_pred))
```

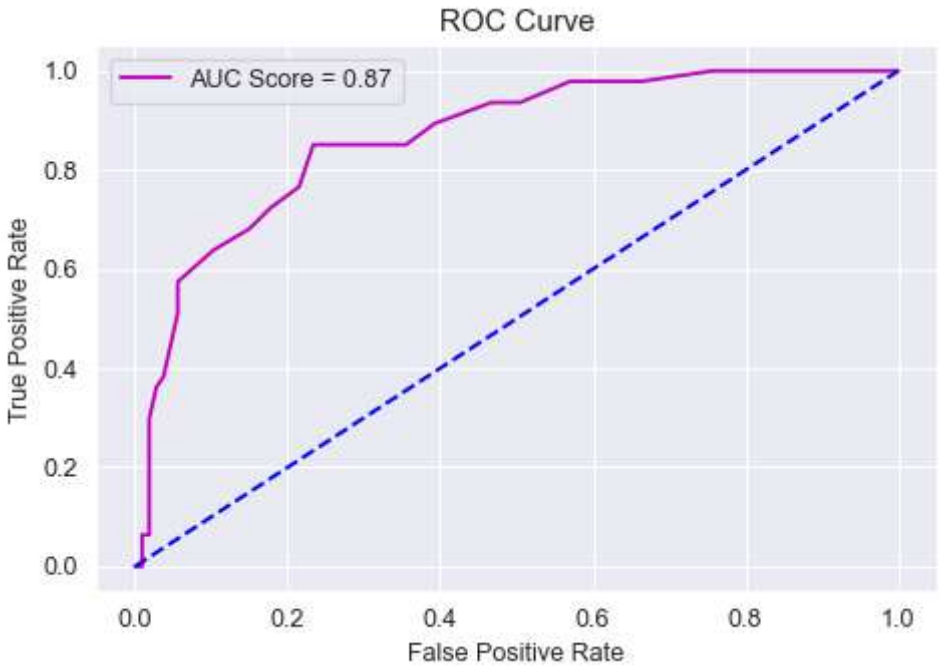
```
[[96 11]
 [17 30]]
```

In [59]:
```
import sklearn.metrics as metrics
print("Model Validation ==>\n")
print("Accuracy Score of KNN Model::")
print(metrics.accuracy_score(y_test,knn_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,knn_pred),'\n')
print("\n","ROC Curve")
knn_prob=knn_model.predict_proba(x_test_std)
knn_prob1=knn_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,knn_prob1)
roc_auc_knn=metrics.auc(fpr,tpr)
plt.figure(dpi=90)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'m',label='AUC Score = %0.2f'%roc_auc_knn)
plt.plot(fpr,fpr,'r--',color='blue')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of KNN Model::
0.8181818181818182

 Classification Report::
              precision    recall  f1-score   support
```

```
                0       0.85      0.90      0.87       107
                1       0.73      0.64      0.68        47

         accuracy                           0.82       154
        macro avg       0.79      0.77      0.78       154
     weighted avg       0.81      0.82      0.81       154
```

Out[59]:
```
 ROC Curve
<matplotlib.legend.Legend at 0x1a4576fa820>
```



KNN with Normalization

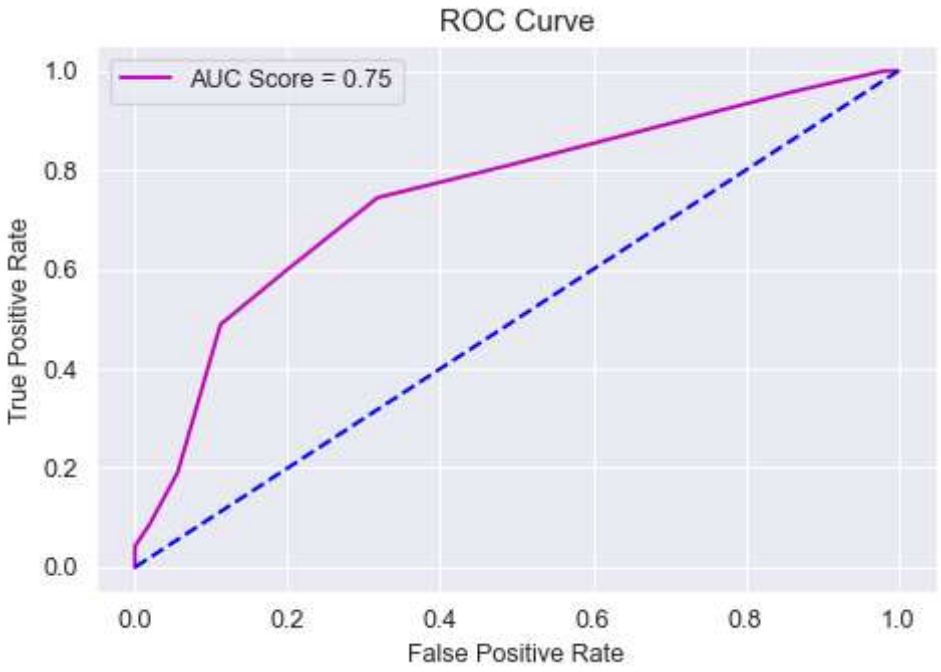
In [60]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
knn_model_norm = KNeighborsClassifier(n_neighbors=25)
knn_model_norm.fit(x_train_norm,y_train_norm)
knn_pred_norm=knn_model_norm.predict(x_test_norm)
```

In [61]:
```
print("Model Validation ==>\n")
print("Accuracy Score of KNN Model with Normalization::")
print(metrics.accuracy_score(y_test_norm,knn_pred_norm))
print("\n","Classification Report::")
print(metrics.classification_report(y_test_norm,knn_pred_norm),'\n')
print("\n","ROC Curve")
knn_prob_norm=knn_model.predict_proba(x_test_norm)
knn_prob_norm1=knn_prob_norm[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test_norm,knn_prob_norm1)
roc_auc_knn=metrics.auc(fpr,tpr)
plt.figure(dpi=90)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'m',label='AUC Score = %0.2f'%roc_auc_knn)
```

```
plt.plot(fpr,fpr,'r--',color='blue')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of KNN Model with Normalization::
0.8311688311688312

 Classification Report::
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       107
           1       0.74      0.68      0.71        47

    accuracy                           0.83       154
   macro avg       0.80      0.79      0.80       154
weighted avg       0.83      0.83      0.83       154


 ROC Curve
```

Out[61]:
```
<matplotlib.legend.Legend at 0x1a458289bb0>
```



In [62]:
```
print(confusion_matrix(y_test, knn_pred_norm))
```

```
[[96 11]
 [15 32]]
```

Inference: From the above verification, it is clearly shown that KNN with Standard Scaling is more effective and better than KNN with Normalization.

Random Factor(Ensemble Learning)

In [63]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=1000,random_state=0)
```

```
rf_model.fit(x_train_std,y_train)
rf_pred=rf_model.predict(x_test_std)
```

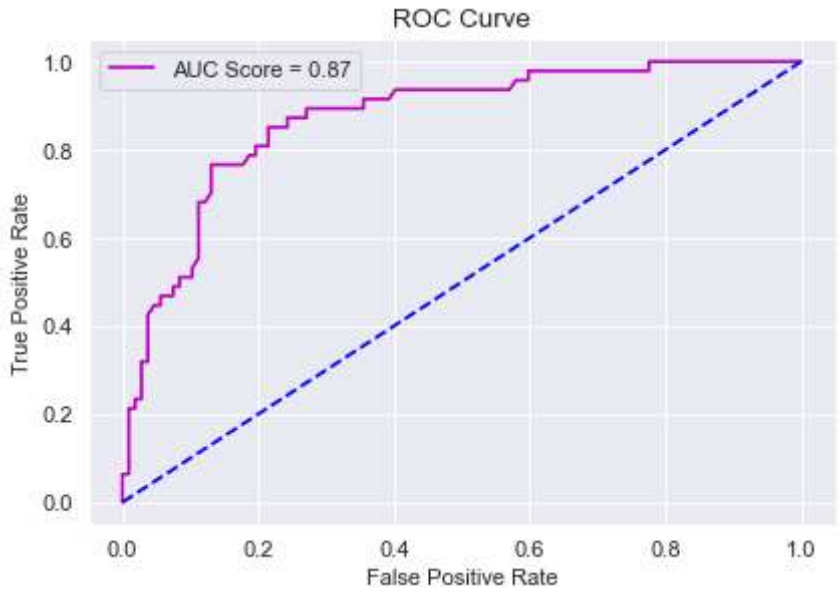
In [64]:

```
print("Model Validation ==>\n")
print("Accuracy Score of Logistic Regression Model::")
print(metrics.accuracy_score(y_test,rf_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,rf_pred),'\n')
print("\n","ROC Curve")
rf_prob=rf_model.predict_proba(x_test_std)
rf_prob1=rf_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,rf_prob1)
roc_auc_rf=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.plot(fpr,tpr,'m',label='AUC Score = %0.2f'%roc_auc_rf)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,fpr,'r--',color='blue')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.8246753246753247

 Classification Report::
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       107
           1       0.71      0.72      0.72        47

    accuracy                           0.82       154
   macro avg       0.79      0.80      0.79       154
weighted avg       0.83      0.82      0.83       154

```

Out[64]:

```
 ROC Curve
<matplotlib.legend.Legend at 0x1a45952e310>
```



Inference: Random Forest: Accuracy: 0.8246753246753247 and AUC Score: 0.87

SVC Model

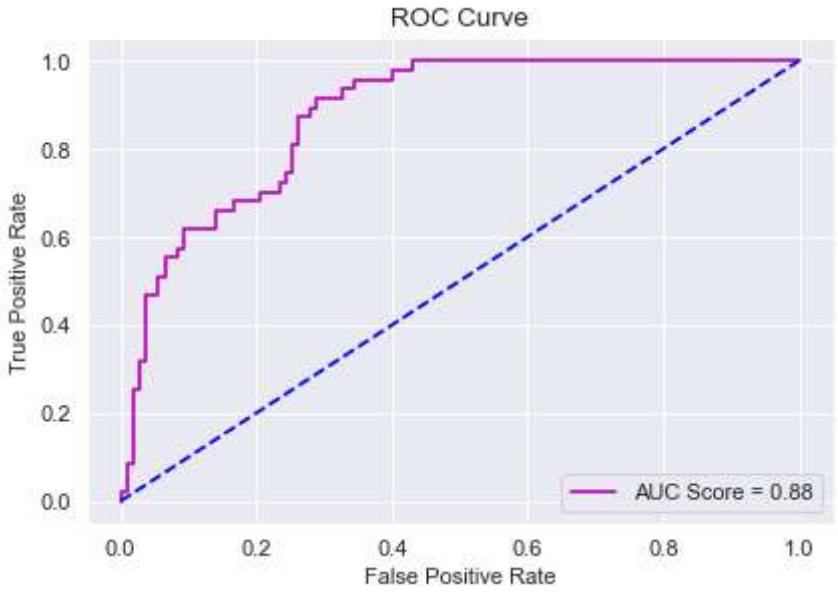
In [65]:
```
from sklearn.svm import SVC
svc_model_linear = SVC(kernel='linear',random_state=0,probability=True,C=0.01)
svc_model_linear.fit(x_train_std,y_train)
svc_pred=svc_model_linear.predict(x_test_std)
```

In [66]:
```
print("Model Validation ==>\n")
print("Accuracy Score of SVC Model with Linear Kernel::")
print(metrics.accuracy_score(y_test,svc_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred),'\n')
print("\n","ROC Curve")
svc_prob_linear=svc_model_linear.predict_proba(x_test_std)
svc_prob_linear1=svc_prob_linear[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svc_prob_linear1)
roc_auc_svc=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'m',label='AUC Score = %0.2f'%roc_auc_svc)
plt.plot(fpr,fpr,'r--',color='blue')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of SVC Model with Linear Kernel::
0.8116883116883117

 Classification Report::
              precision    recall  f1-score   support

           0       0.83      0.92      0.87       107
           1       0.75      0.57      0.65        47
```

```
              precision    recall  f1-score   support

    accuracy                         0.81       154
   macro avg       0.79      0.75      0.76       154
weighted avg       0.81      0.81      0.80       154
```

Out[66]:
```
 ROC Curve
<matplotlib.legend.Legend at 0x1a45955e8b0>
```



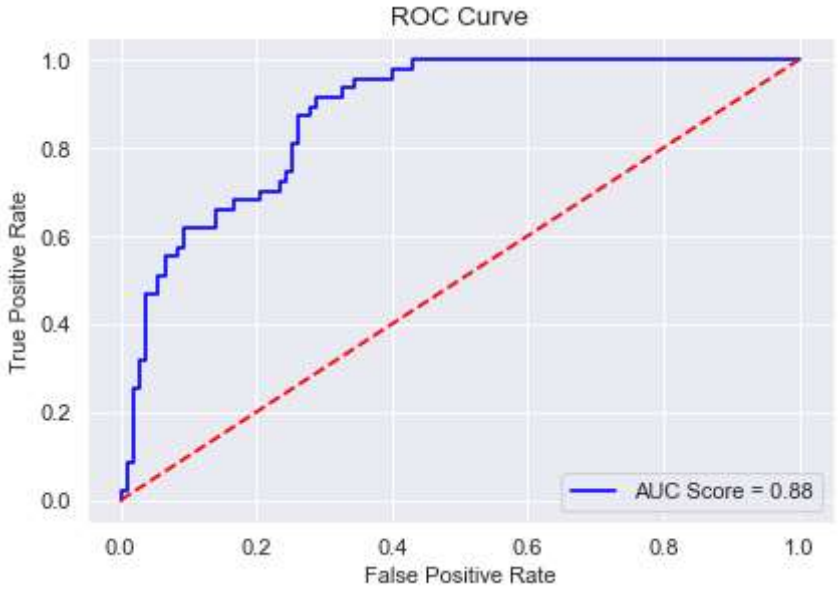
In [67]:
```
from sklearn.svm import SVC
svc_model_rbf = SVC(kernel='rbf',random_state=0,probability=True,C=1)
svc_model_rbf.fit(x_train_std,y_train)
svc_pred_rbf=svc_model_rbf.predict(x_test_std)
```

In [68]:
```
print("Model Validation ==>\n")
print("Accuracy Score of SVC Model with RBF Kernel::")
print(metrics.accuracy_score(y_test,svc_pred_rbf))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,svc_pred_rbf),'\n')
print("\n","ROC Curve")
svc_prob_rbf=svc_model_linear.predict_proba(x_test_std)
svc_prob_rbf1=svc_prob_rbf[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svc_prob_rbf1)
roc_auc_svc=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_svc)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of SVC Model with RBF Kernel::
0.7727272727272727
```

```
Classification Report::
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       107
           1       0.66      0.53      0.59        47

    accuracy                           0.77       154
   macro avg       0.73      0.71      0.72       154
weighted avg       0.76      0.77      0.77       154
```

Out[68]:
```
 ROC Curve
<matplotlib.legend.Legend at 0x1a459559e20>
```



Inference: Here, SVC with Linear Kernel is better than RBF Kernel and hence noted that the variables are linearly dependent on outcome.

- When Comparing with KNN, both Models are working well however SVC Linear with C=0.01 is better in terms of AUC Score.

Logistic Regression

In [69]:
```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.1)
lr_model.fit(x_train_std,y_train)
lr_pred=lr_model.predict(x_test_std)
```

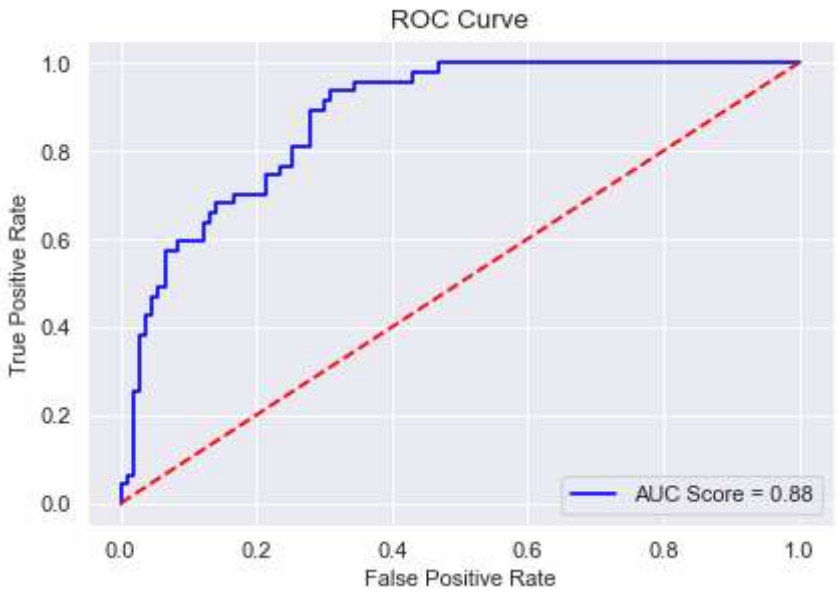
In [70]:
```
print("Model Validation ==>\n")
print("Accuracy Score of Logistic Regression Model::")
print(metrics.accuracy_score(y_test,lr_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,lr_pred),'\n')
print("\n","ROC Curve")
lr_prob=lr_model.predict_proba(x_test_std)
lr_prob1=lr_prob[:,1]
```

```
fpr,tpr,thresh=metrics.roc_curve(y_test,lr_prob1)
roc_auc_lr=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_lr)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.8181818181818182

 Classification Report::
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.60      0.67        47

    accuracy                           0.82       154
   macro avg       0.80      0.76      0.77       154
weighted avg       0.81      0.82      0.81       154
```

Out[70]:

```
 ROC Curve
<matplotlib.legend.Legend at 0x1a45a5f1dc0>
```



Inference: Well, the accuracy of KNN is better than Logistic Regression whereas the AUC score of Logistic regression is way better.

Let's interpret the results...

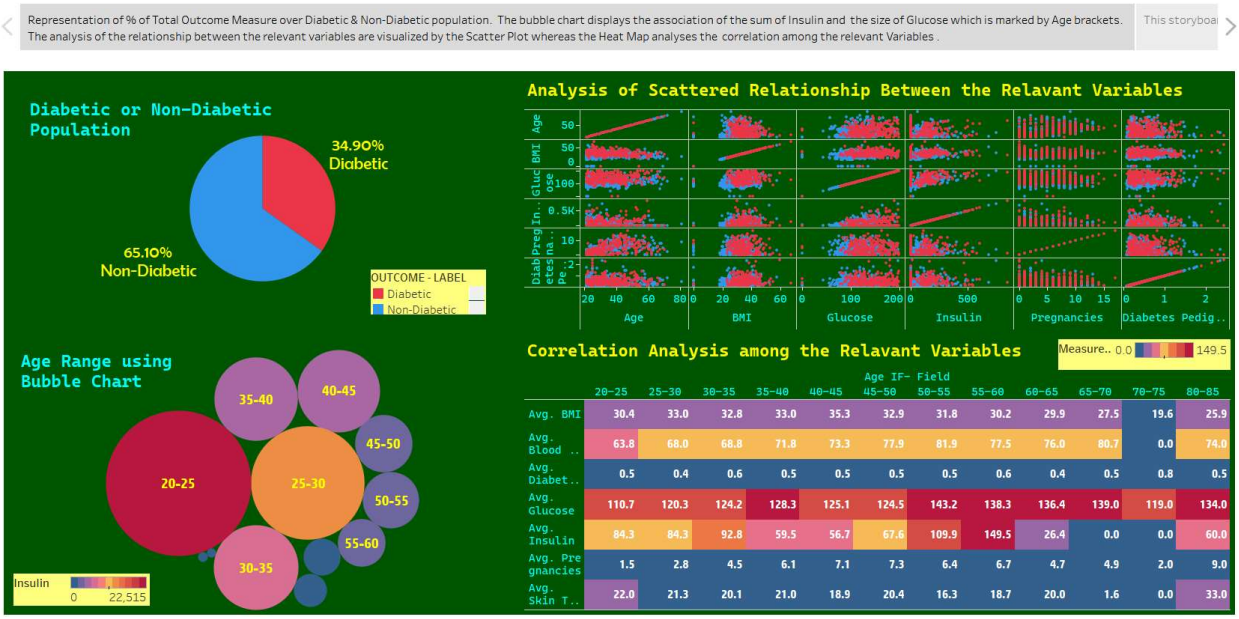
```
    Model                ---       Accuracy Score        ---        AUC
    ---        f1 Score
```

- KNN(Standard Scaling) --- 0.8181818181818182 --- 0.87 --- 0.87
- KNN(Normalization) --- 0.8311688311688312 --- 0.75 --- 0.88
- Random Forest --- 0.8246753246753247 --- 0.87 --- 0.87
- SVC Model --- 0.8116883116883117 --- 0.88 --- 0.87(linear_Kernel), 0.84(Rbf_kernel)
- Logistic Regression --- 0.8181818181818182 --- 0.88 --- 0.88

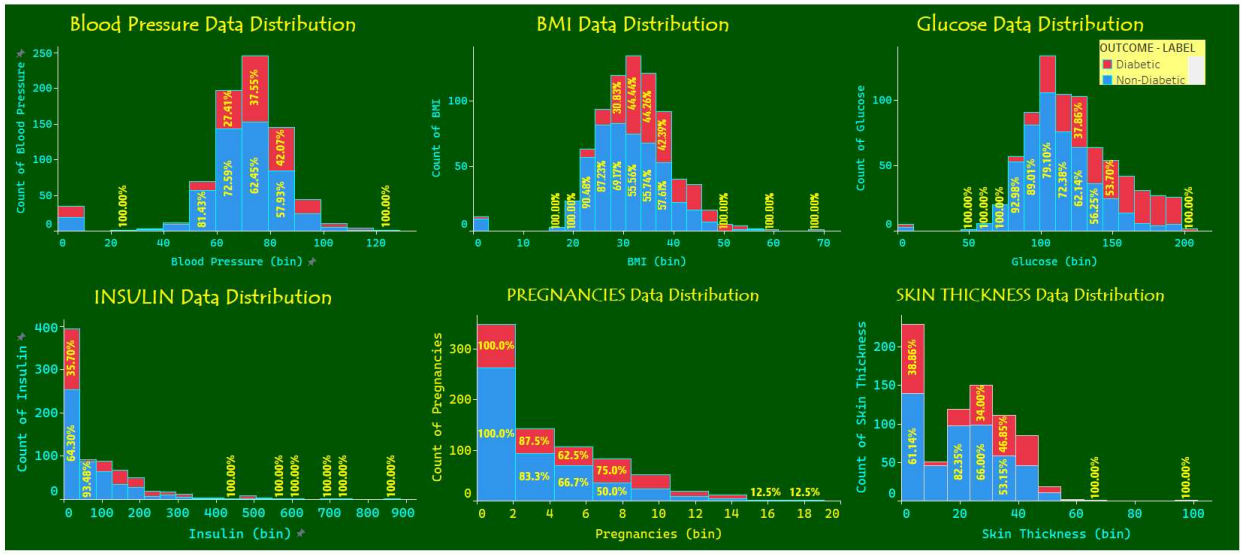
Summary:

It is evident from the verified models that Random Forest Classifier is the best among all, however the auc score is lesser by 1 than others hence it is considered to be the best as the balance of classes between Precision and Recall is far better than other Models. Noted to be considered the loss in AUC by 1.

Data Reporting



Age brackets. This storyboard of Histogram (Data Distribution) depicts and summarizes the trend of % of Total Count of Blood Pressure, BMI, Glucose, Insulin, Pregnancies & Skin Thickness



In [ ]: