EXPOSYS DATA LABS INTERNSHIP PROJECT

Name : Dhyey Joshi
Email : dhyey194@gmail.com
College : Charotar University of Science and Technology (CHARUSAT)
Department : Computer Science & Engineering
Domain : Data Science

Problem Statement:
Diabetes is a type of chronic disease which is more common among the people of all age groups. Predicting this disease at an early stage can help a person to take the necessary precautions and change his/her lifestyle accordingly to either prevent the occurrence of this disease or control the disease(For people who already have the disease).
Task:

1. **Prepare the data-set using several methods to train the model.**
2. **Build a model which can give high accuracy of predicting the disease.**

Importing Libraries

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

Data Collection

In [3]:

```
#Loading data
df = pd.read_csv('./diabetes.csv')
df.head(5)
```

Out[3]:

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

Exploratory Data Analysis

In [4]:

```
df.describe() # This function is very helpfull in extracting the insights of dataset i.e mean,median,mod,std etc..
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	
count	768.000000	768.000000	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	0.471876	33.240885	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	

In [8]:

```
#Total number of rows & columns in dataset
print("Total number of rows: ", df.shape[0])
print("Total number of columns: ", df.shape[1])
print("Total Dimensions : ", df.shape)
```

```
Total number of rows:  768
Total number of columns:  9
Total Dimensions :  (768, 9)
```

In [10]:

```
# Checking for null values & data-types of cols.
df.info()
df.columns
```

```
<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
```

Out[10]:

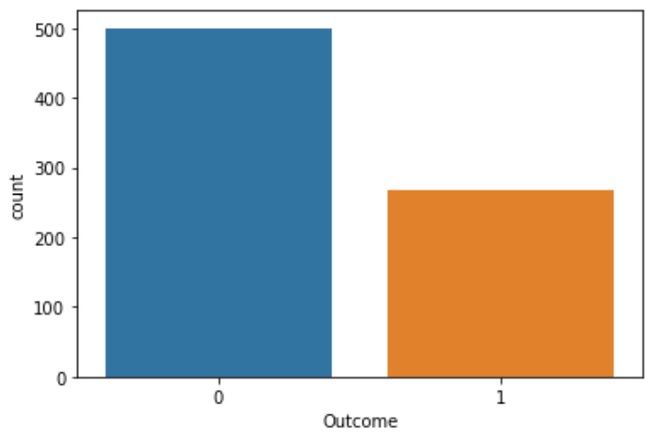
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [13]:

```
#Countng values of outcomes having 0 or 1, 0 means non diabetic and 1 means diabetic
sns.countplot(x='Outcome',data=df)
```

Out[13]:

```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



In [18]:

```
df.Outcome.unique()
```

Out[18]:

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

In [16]:

```
df['Outcome'].value_counts()
```

Out[16]:

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

In [17]:

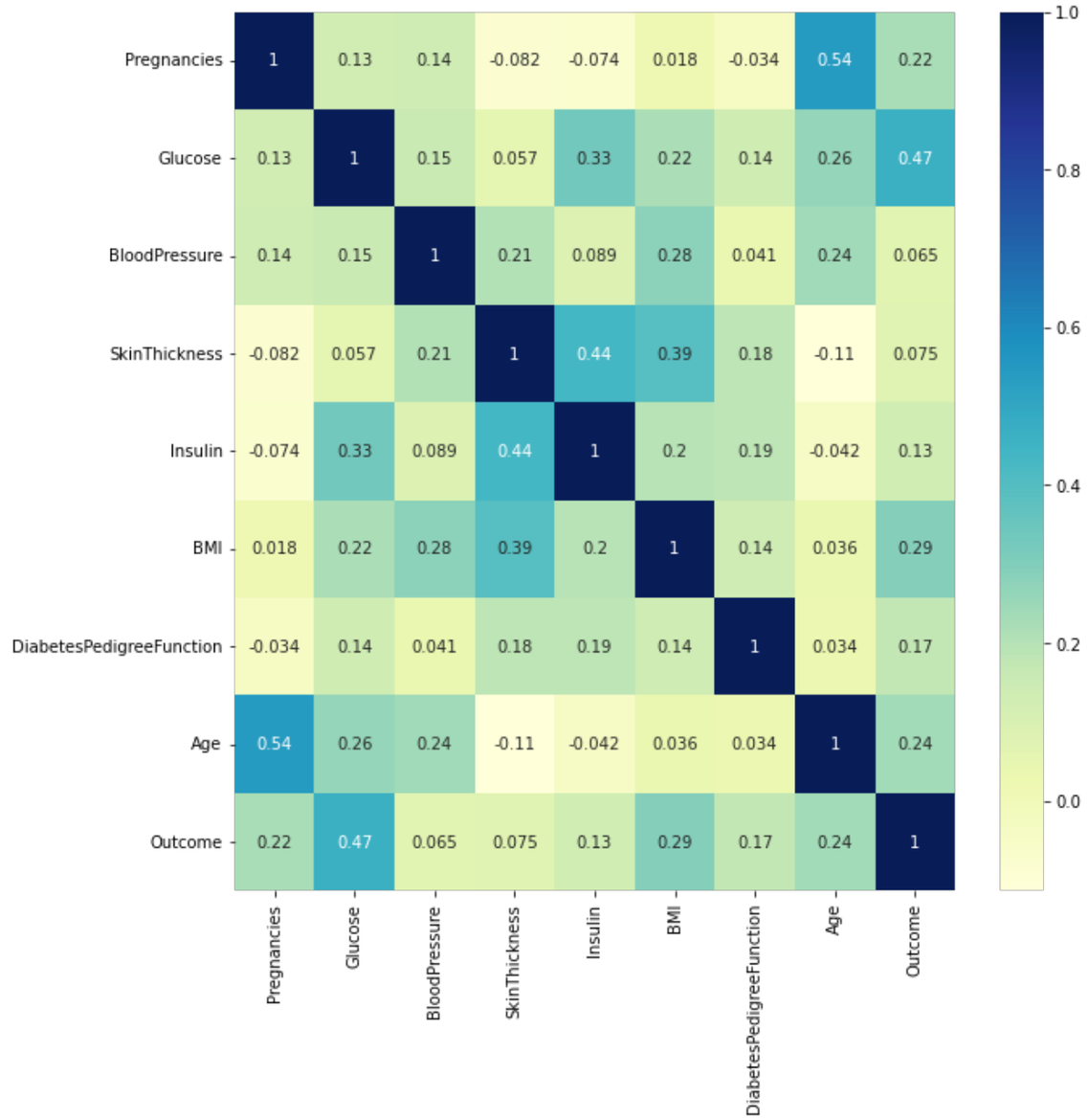
```
df.groupby('Outcome').mean()
```

Out[17]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

In [20]:

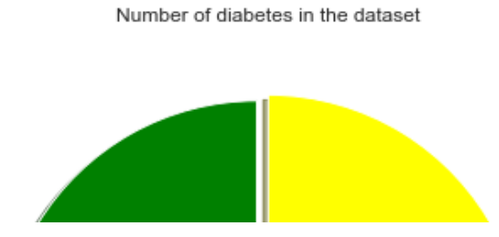
```
#Correlation matrix to show correlation between two variables, 0.x means x% similar
plt.figure(figsize=(10,10))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True,cmap="YlGnBu")
plt.show()
```

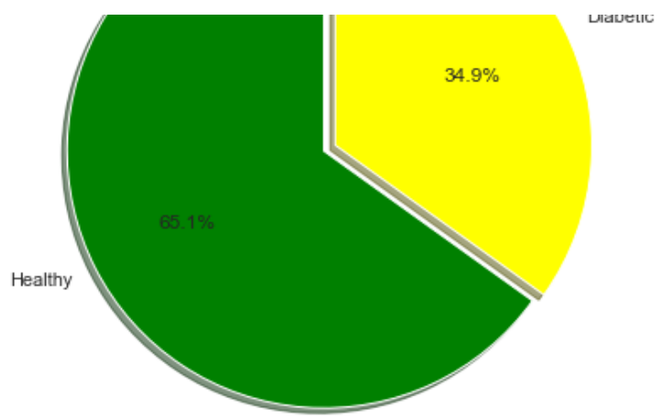


DATASET

In [23]:

```
sns.set(style="whitegrid")
labels = ['Healthy', 'Diabetic']
sizes = df['Outcome'].value_counts(sort = True)

colors = ["green","yellow"]
explode = (0.05,0)

_
plt.figure(figsize=(7,7))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90,)

plt.title('Number of diabetes in the dataset')
plt.show()
```





Healthy vs Diabetic by Pragnancy

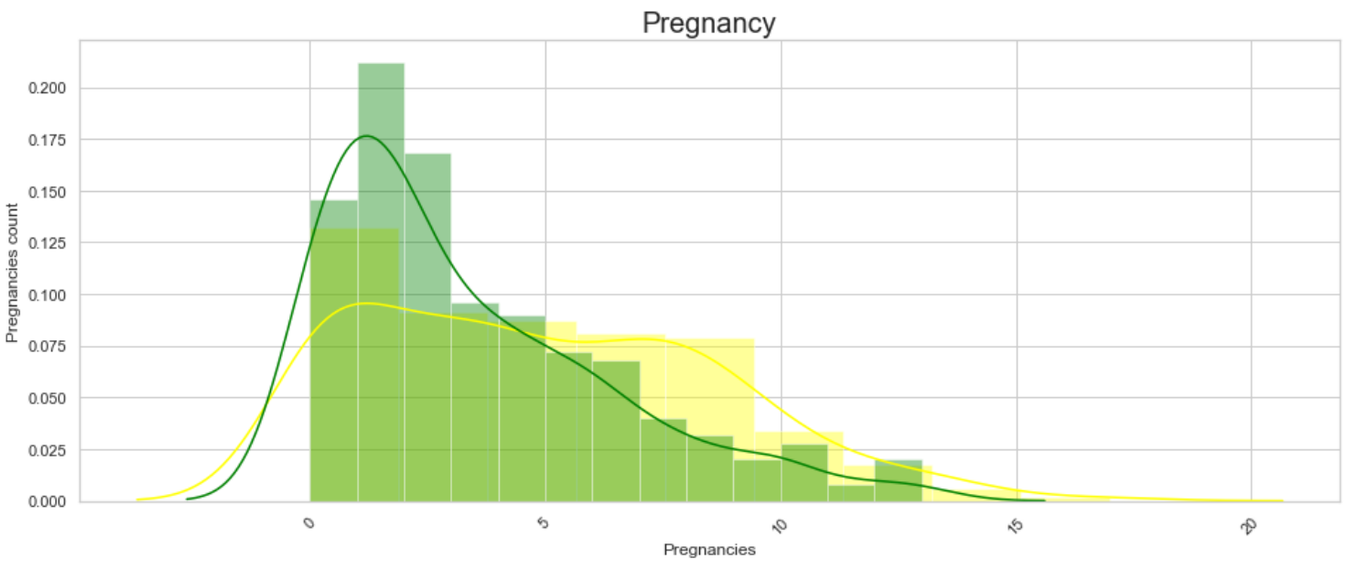
In [30]:

```
#pregnancy for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["Pregnancies"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["Pregnancies"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("Pregnancies count")
plt.title("Pregnancy",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[30]:

```
Text(0.5, 1.0, 'Pregnancy')
```



Healthy vs Diabetic by Glucose

In [32]:

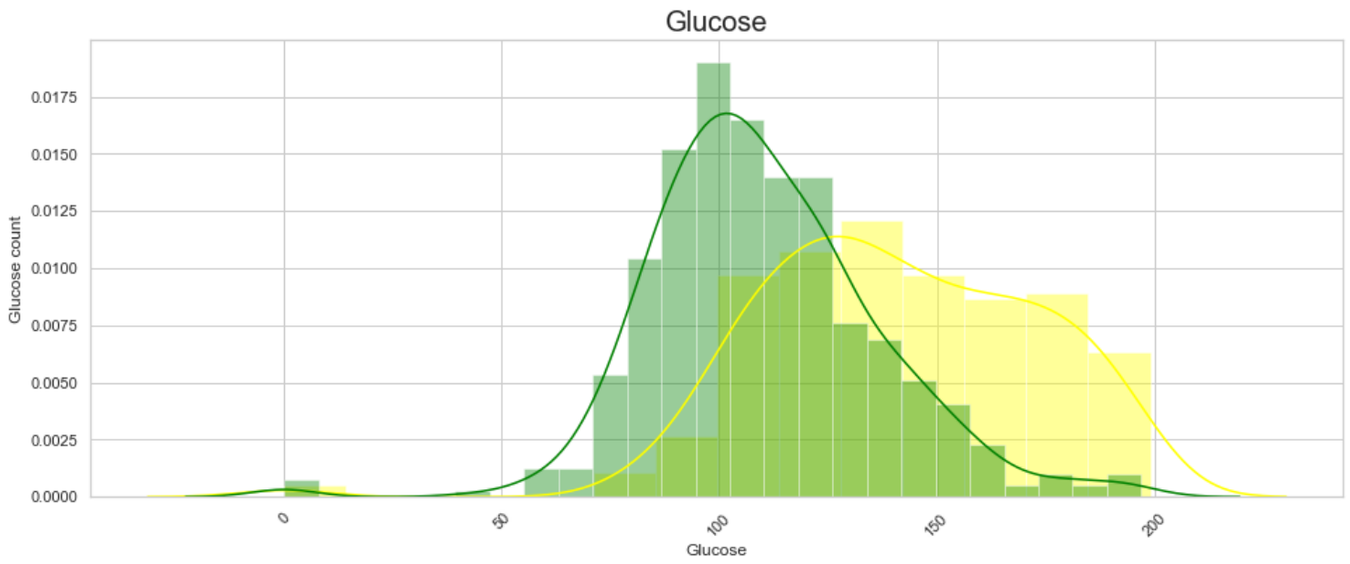
```
#glucose for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["Glucose"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["Glucose"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("Glucose count")
plt.title("Glucose",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please
```

```
arning: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[32]:

```
Text(0.5, 1.0, 'Glucose')
```



Healthy vs Diabetic by Blood Pressure

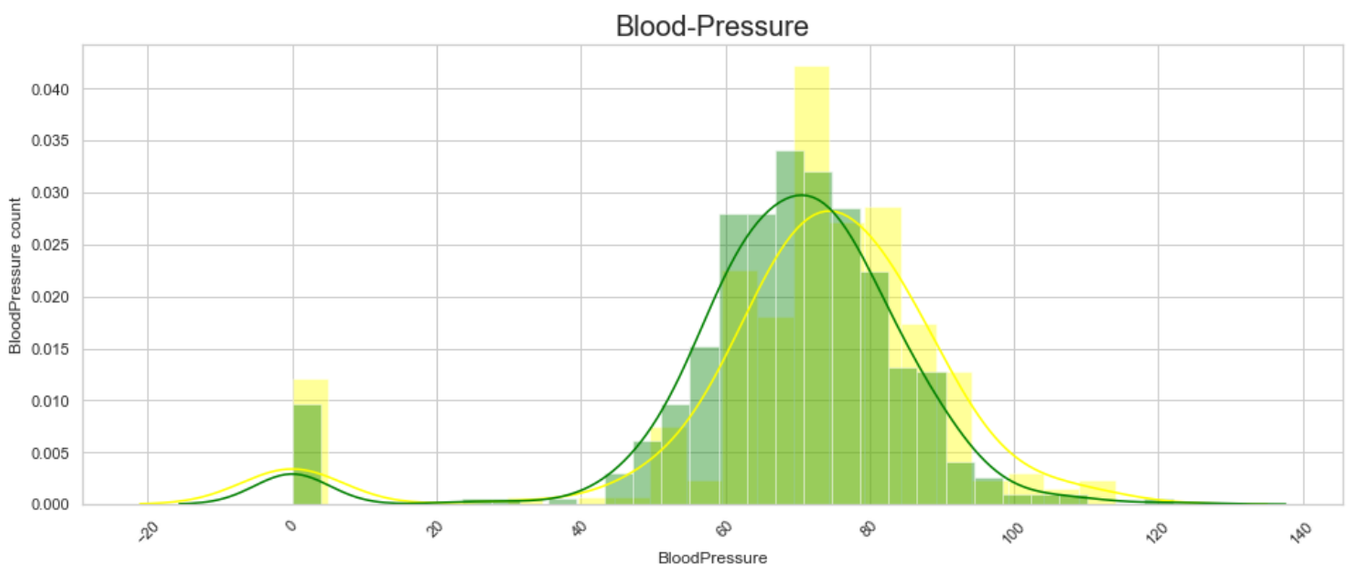
In [33]:

```
#glucose for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["BloodPressure"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["BloodPressure"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("BloodPressure count")
plt.title("Blood-Pressure",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[33]:

```
Text(0.5, 1.0, 'Blood-Pressure')
```



Healthy vs Diabetic by Skin Thickness

In [36]:

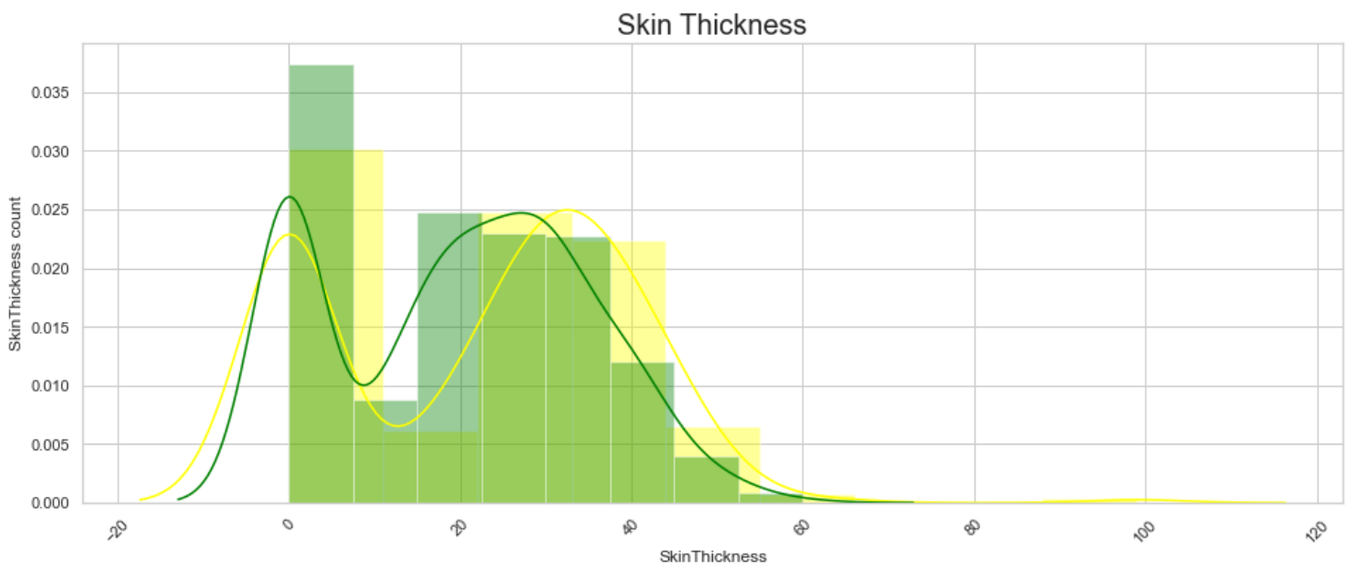
```
#SkinThickness for diabetic
fig = plt.figure(figsize =(16,6))
```

```
sns.distplot(df["SkinThickness"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["SkinThickness"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("SkinThickness count")
plt.title("Skin Thickness",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[36]:

```
Text(0.5, 1.0, 'Skin Thickness')
```



Healthy vs Diabetic by Insulin

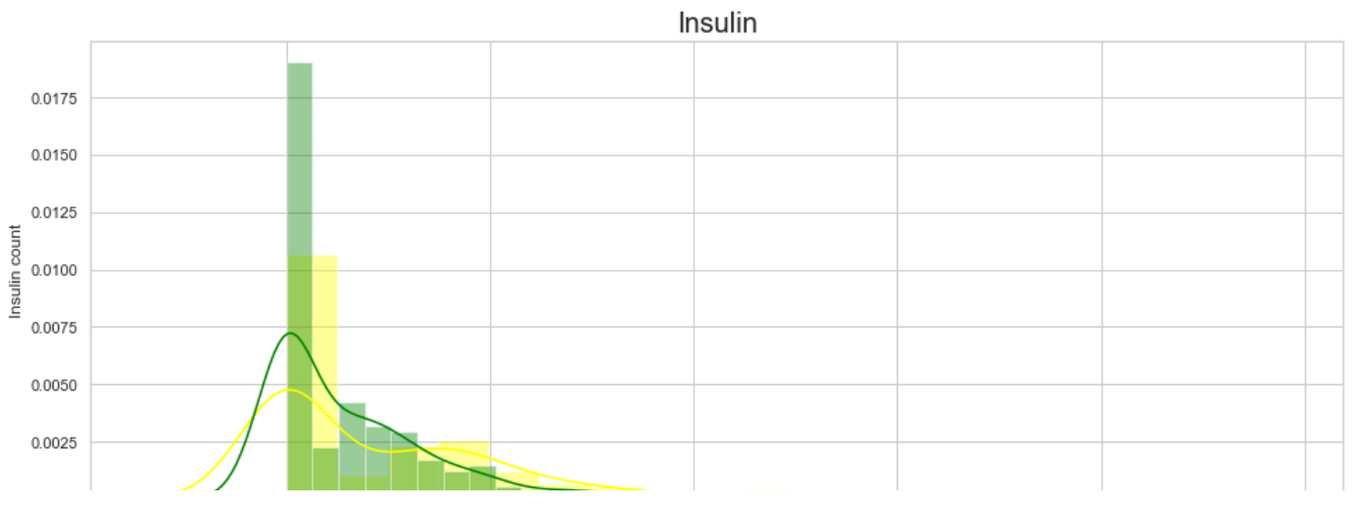
In [37]:

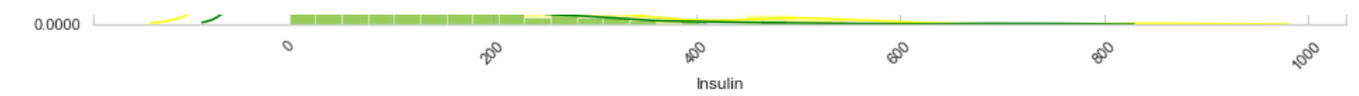
```
#Insulin for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["Insulin"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["Insulin"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("Insulin count")
plt.title("Insulin",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[37]:

```
Text(0.5, 1.0, 'Insulin')
```





Healthy vs Diabetic by BMI

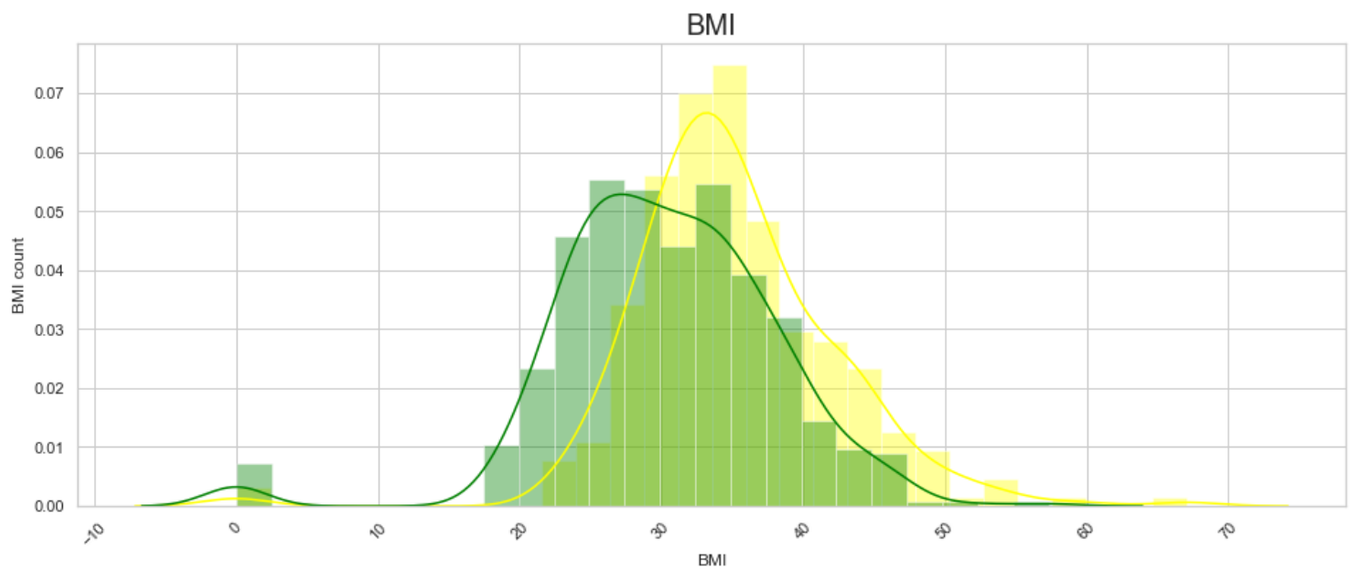
In [38]:

```
#Insulin for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["BMI"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["BMI"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("BMI count")
plt.title("BMI",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[38]:

```
Text(0.5, 1.0, 'BMI')
```



Healthy vs Diabetic by DiabetesPedigreeFunction

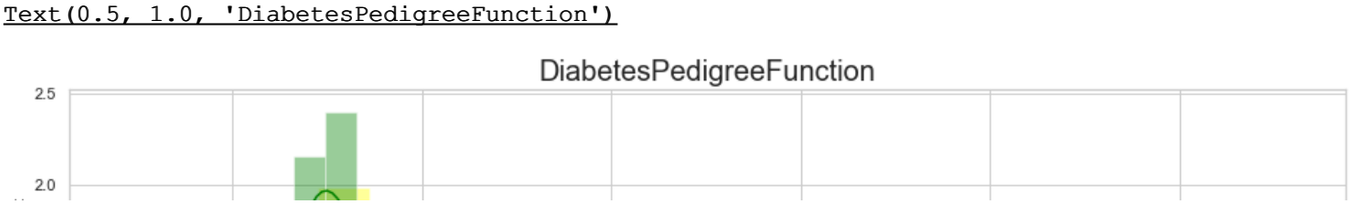
In [39]:

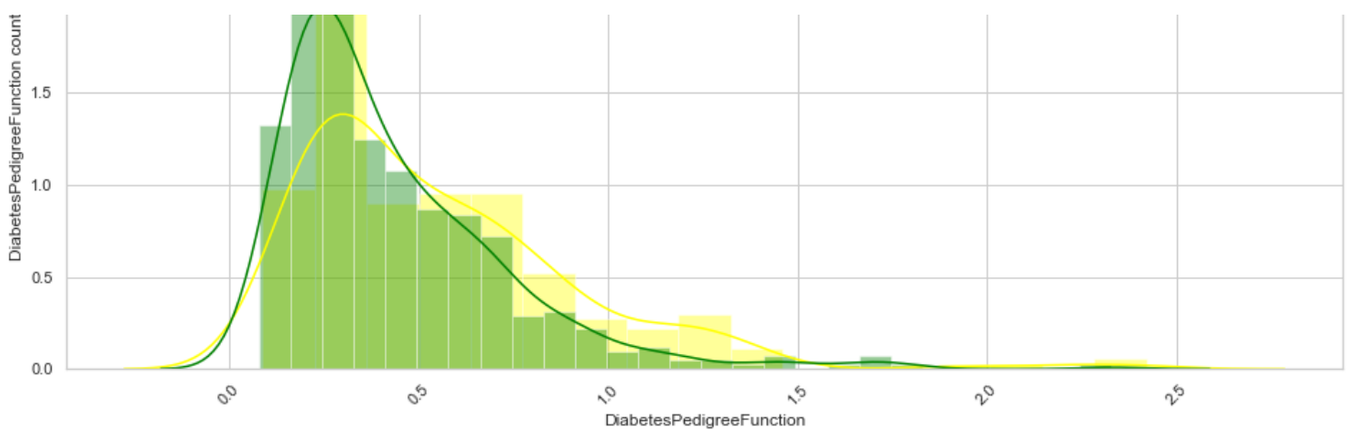
```
#DiabetesPedigreeFunction for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["DiabetesPedigreeFunction"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["DiabetesPedigreeFunction"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("DiabetesPedigreeFunction count")
plt.title("DiabetesPedigreeFunction",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[39]:

```
Text(0.5, 1.0, 'DiabetesPedigreeFunction')
```





Healthy vs Diabetic by Age

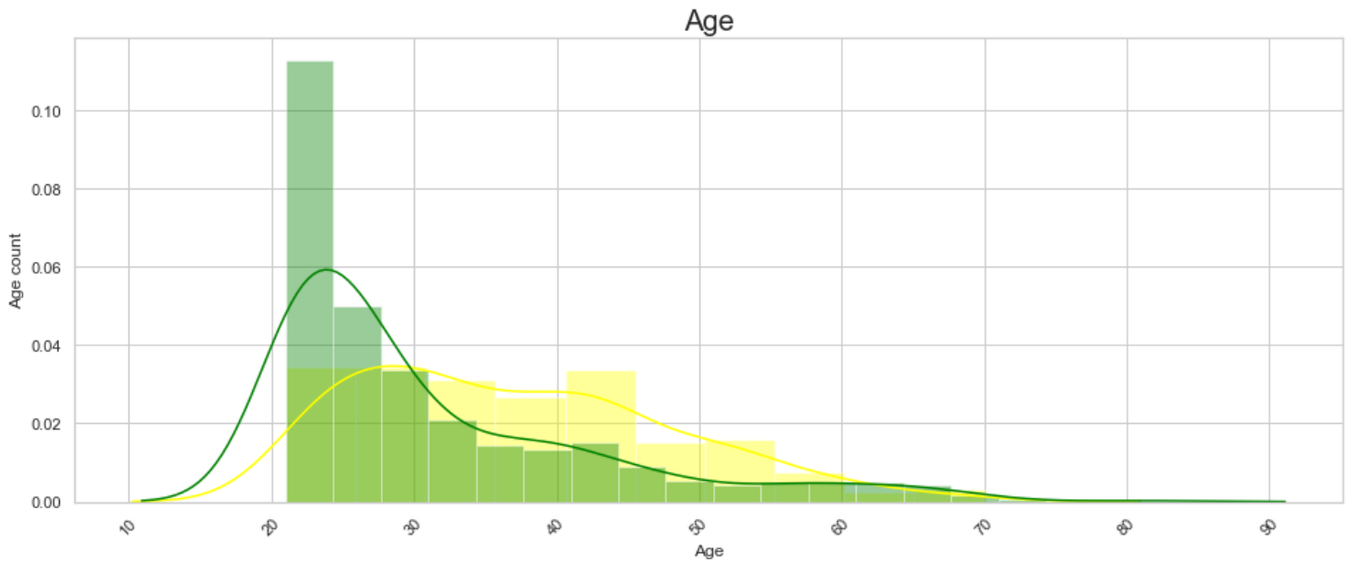
In [40]:

```
#Age for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(df["Age"][df["Outcome"] == 1], color='yellow')
sns.distplot(df["Age"][df["Outcome"] == 0], color='green')
plt.xticks(rotation = 45)
plt.ylabel("Age count")
plt.title("Age",fontsize = 20)
```

```
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Lenovo\anaconda3\envs\myDL\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[40]:

```
Text(0.5, 1.0, 'Age')
```



Data Cleaning

In [45]:

```
df.isna().sum()
```

Out[45]:

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

Data Preprocessing

Data Preprocessing

In [372]:

```
#Removing unnessary columns
x = df.drop(["Pregnancies","BloodPressure","SkinThickness","Outcome"],axis = 1)
y = df.iloc[:,-1]
```

In [377]:

```
x
```

Out[377]:

	Glucose	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148	0	33.6	0.627	50
1	85	0	26.6	0.351	31
2	183	0	23.3	0.672	32
3	89	94	28.1	0.167	21
4	137	168	43.1	2.288	33
...	...	...	...	...	...
763	101	180	32.9	0.171	63
764	122	0	36.8	0.340	27
765	121	112	26.2	0.245	30
766	126	0	30.1	0.349	47
767	93	0	30.4	0.315	23

768 rows × 5 columns

In [378]:

```
y
```

Out[378]:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [373]:

```
# Splitting the dataset into training and test set.
#test_size 0.3 means for testing data 30% and training data 70%

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)
```

In [374]:

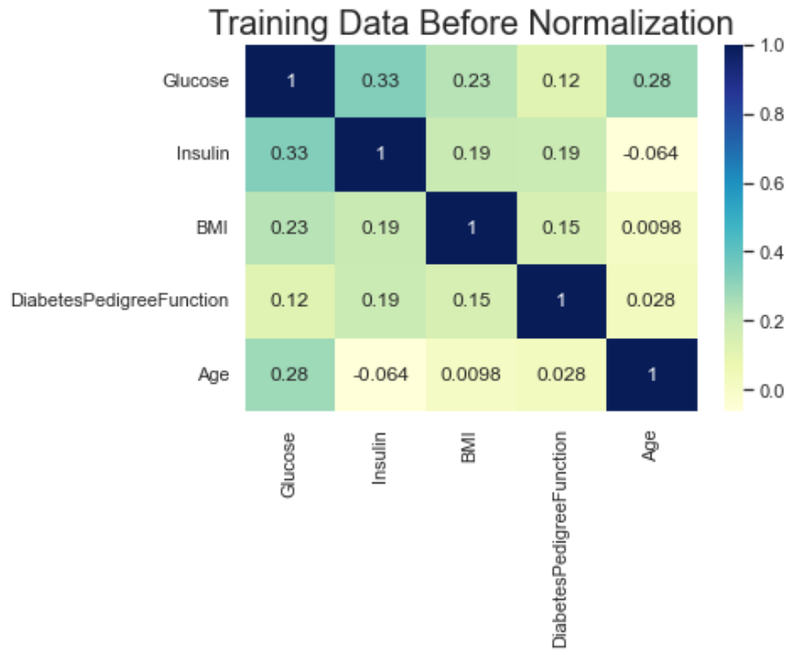
```
print("Number of dimensions of dataset: ",df.shape)
print("Training Data ---- ")
print("Number of dimensions x_train dataset: ", x_train.shape)
print("Number of dimensions y_train dataset: ", y_train.shape)
print("Testing Data ---- ")
print("Number of dimensions x_test dataset: ", x_test.shape)
print("Number of dimensions y_test dataset: ", y_test.shape)
```

```
Number of dimensions of dataset:  (768, 9)
Training Data ----
Number of dimensions x_train dataset:  (614, 5)
Number of dimensions y_train dataset:  (614,)
Testing Data ----
Number of dimensions x_test dataset:  (154, 5)
Number of dimensions y_test dataset:  (154,)
```

In [375]:

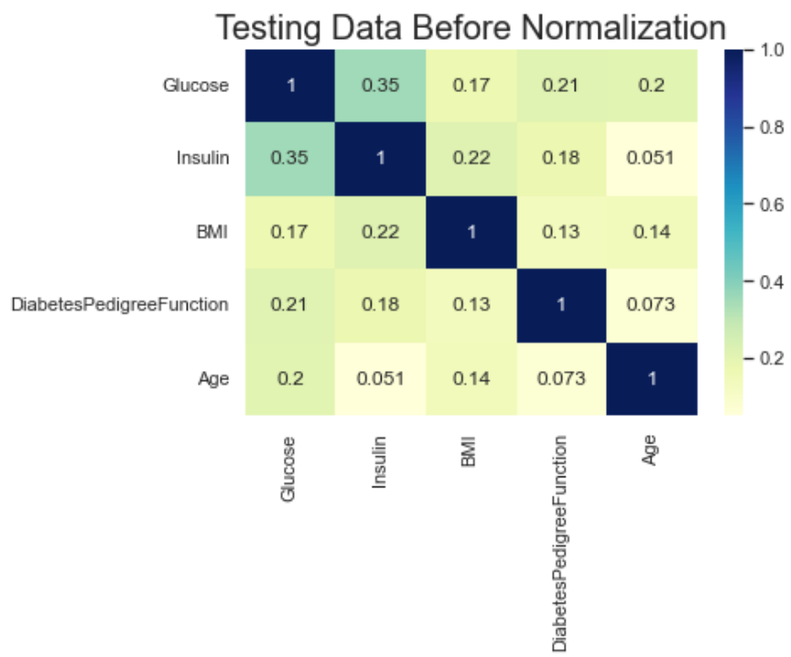
```
#Correlation matrix to show correlation between two variables, 0.x means x% similar
corr_matrix = x_train.corr()
sns.heatmap(corr_matrix, annot=True,cmap="YlGnBu")
```

```
plt.title("Training Data Before Normalization" ,fontsize=20)
plt.show()
```



In [376]:

```
#Correlation matrix to show correlation between two variables, 0.x means x% similar
corr_matrix = x_test.corr()
sns.heatmap(corr_matrix, annot=True,cmap="YlGnBu")
plt.title("Testing Data Before Normalization", fontsize=20)
plt.show()
```



In [341]:

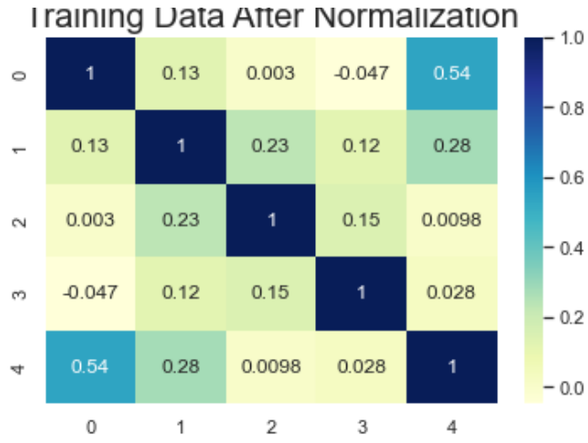
```
#Feature Scaling - To standardize the independent features present in the data in a fixed ra
nge.
#If feature scaling is not done, then a machine learning algorithm tends to weigh greater va
lues, higher and consider smaller
#values as the lower values, regardless of the unit of the values.

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [308]:

```
x_train = pd.DataFrame(x_train)
x_test = pd.DataFrame(x_test)
```

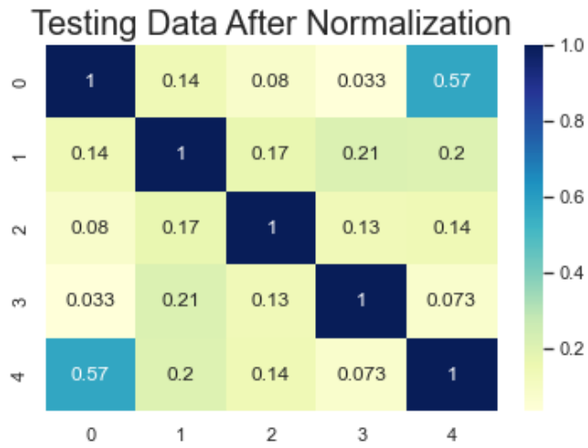
In [309]:

```
#Correlation matrix to show correlation between two variables, 0.x means x% similar
corr_matrix = x_train.corr()
sns.heatmap(corr_matrix, annot=True,cmap="YlGnBu")
plt.title("Training Data After Normalization" ,fontsize=20)
plt.show()
```



In [310]:

```
#Correlation matrix to show correlation between two variables, 0.x means x% similar
corr_matrix = x_test.corr()
sns.heatmap(corr_matrix, annot=True,cmap="YlGnBu")
plt.title("Testing Data After Normalization" ,fontsize=20)
plt.show()
```



Building Predictive Models Using Supervised ML Algorithm & Ensemble Learning Algorithm

Training Models

1. Logistic Regression

In [342]:

```
#Training the logistic regression model
from sklearn.linear_model import LogisticRegression
Lr_classifier = LogisticRegression(random_state=0)
Lr_classifier.fit(x_train, y_train)
```

Out[342]:

```
LogisticRegression(random_state=0)
```

2. Naive Bayes

In [343]:

```
#Training the Naive Bayes Model
from sklearn.naive_bayes import GaussianNB , BernoulliNB ,MultinomialNB # GaussianNB : For Continuous distribution of data , BernoulliNB: For the dataset having only two classes i.e 0 & 1 , MultinomialNB: For the dataset having count or we can say how many times word repeated ..mostly used for NLP .

Gaussian_NB = GaussianNB()

Gaussian_NB.fit(x_train,y_train)
```

Out[343]:

```
GaussianNB()
```

In [344]:

```
#Training the Naive Bayes Model
from sklearn.naive_bayes import GaussianNB , BernoulliNB ,MultinomialNB # GaussianNB : For
```

```
Continuous distribution of data , BernoulliNB: For the dataset having only two classes i.e 0 & 1 , MultinomialNB: For the dataset having count or we can say how many times word repeated ..mostly used for NLP .

Bernoulli_NB = BernoulliNB()

Bernoulli_NB.fit(x_train,y_train)
```

Out[344]:

```
BernoulliNB()
```

3. K Nearset Neighbor

In [345]:

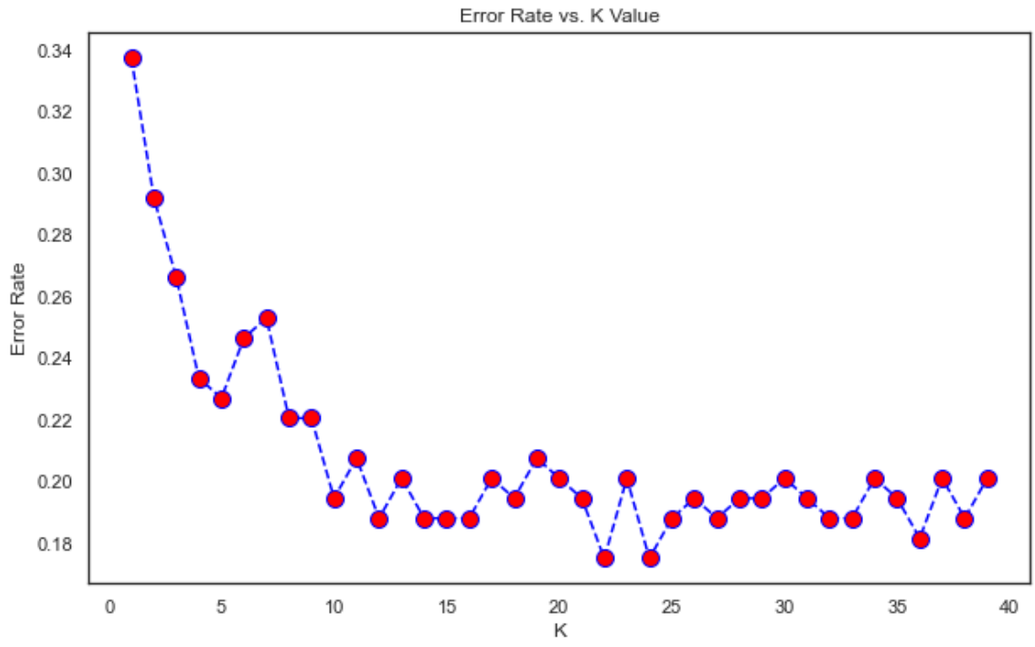
```
# Training the K - Nearest Neighbour Model
from sklearn.neighbors import KNeighborsClassifier
import warnings

warnings.filterwarnings('ignore')
# Choosing correct value of K
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(x_train,y_train)

    pred_i = knn.predict(x_test)

#     pred_i=pred_i.reshape(231,1)

    error_rate.append(np.mean(pred_i != y_test))
```

In [346]:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[346]:

```
Text(0, 0.5, 'Error Rate')
```



In [348]:

```
# NOW WITH K=30
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=24, metric = 'minkowski')

knn.fit(x_train,y_train)
```

Out[348]:

KNeighborsClassifier(n_neighbors=24)

4. Support Vector Machine

In [349]:

```
# Training the SVM Model
from sklearn.svm import SVC
Svc_classifier = SVC()
Svc_classifier.fit(x_train,y_train)
```

Out[349]:

SVC()

Ensemble Learning -

5. Random Forest Classifier

In [350]:

```
# Training the Random Forest  Model
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

Rf_classifier = RandomForestClassifier(n_estimators=50,random_state=0)
Rf_classifier.fit(x_train,y_train)
```

Out[350]:

RandomForestClassifier(n_estimators=50, random_state=0)

Evaluation of Models

In [351]:

```
print("Training Score:", Lr_classifier.score(x_train, y_train))
print("Testing Score:", Lr_classifier.score(x_test, y_test))
```

Training Score: 0.7671009771986971
Testing Score: 0.8051948051948052

In [352]:

```
print("Training Score:", Gaussian_NB.score(x_train, y_train))
print("Testing Score:", Gaussian_NB.score(x_test, y_test))
```

Training Score: 0.7638436482084691
Testing Score: 0.7792207792207793

In [353]:

```
print("Training Score:", Bernoulli_NB.score(x_train, y_train))
print("Testing Score:", Bernoulli_NB.score(x_test, y_test))
```

Training Score: 0.737785016286645
Testing Score: 0.7532467532467533

In [354]:

```
print("Training Score:", knn.score(x_train, y_train))
print("Testing Score:", knn.score(x_test, y_test))
```

Training Score: 0.7703583061889251
Testing Score: 0.8246753246753247

In [355]:

```
print("Training Score:", Svc_classifier.score(x_train, y_train))
print("Testing Score:", Svc_classifier.score(x_test, y_test))
```

Training Score: 0.8013029315960912
Testing Score: 0.7987012987012987

In [356]:

```
print("Training Score:", Rf_classifier.score(x_train, y_train))
print("Testing Score:", Rf_classifier.score(x_test, y_test))
```

Training Score: 1.0
Testing Score: 0.7532467532467533

6. AdaBoost

In [215]:

```
# Here We have taken base estimator as - Logistic Regression Model
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier

AdaB_classifier = AdaBoostClassifier(base_estimator=Lr_classifier,n_estimators=200,learning_
rate=1,random_state=42,algorithm='SAMME.R')
AdaB_classifier.fit(x_train,y_train)
```

Out[215]:

```
AdaBoostClassifier(base_estimator=LogisticRegression(random_state=0),
                   learning_rate=1, n_estimators=200, random_state=42)
```

In [216]:

```
print("Training Score:", AdaB_classifier.score(x_train, y_train))
print("Testing Score:", AdaB_classifier.score(x_test, y_test))
```

```
Training Score: 0.7654723127035831
Testing Score: 0.7987012987012987
```

7. Gradient Boost

In [221]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
LR = {'learning_rate':[0.15,0.1,0.10,0.05], 'n_estimators' : [100,150,200,250]}

tunning = GridSearchCV(estimator=GradientBoostingClassifier(),param_grid = LR , scoring= 'a
ccuracy')
tunning.fit(x_train,y_train)
tunning.best_params_ , tunning.best_score_
```

Out[221]:

```
({'learning_rate': 0.15, 'n_estimators': 150}, 0.757350393176063)
```

In [222]:

```
print("Training Score:", tunning.score(x_train, y_train))
print("Testing Score:", tunning.score(x_test, y_test))
```

```
Training Score: 0.9706840390879479
Testing Score: 0.8311688311688312
```

In [223]:

```
GB_pred = tunning.predict(x_test)
print("Actual : " , y_test[0:5])
print("Predicted : ", GB_pred[0:5])
```

```
Actual :  661    1
122    0
113    0
14     1
529    0
Name: Outcome, dtype: int64
Predicted :  [1 0 0 1 0]
```

8. Extreme Gradient Boost

In [226]:

```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.5.1-py3-none-win_amd64.whl (106.6 MB)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\envs\mydl\lib\site-package
s (from xgboost) (1.21.4)
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\envs\mydl\lib\site-package
s (from xgboost) (1.7.3)
Installing collected packages: xgboost
Successfully installed xgboost-1.5.1
```

In [227]:

```
## Hyper Parameter Optimization
```

```
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
____
}
## Hyperparameter optimization using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
import xgboost

def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(ts
ec, 2)))
________
xgbc = xgboost.XGBClassifier()

random_search = RandomizedSearchCV( xgbc , param_distributions = params , n_iter=5 , scorin
g='roc_auc' , n_jobs=-1 , cv=5 , verbose = 3)

from datetime import datetime
# Here we go
start_time = timer(None) # timing starts from this point for "start_time" variable
random_search.fit(x_train,y_train)
timer(start_time) # timing ends here for "start_time" variable
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[17:36:06] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner
.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective '
binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd
like to restore the old behavior.

 Time taken: 0 hours 0 minutes and 27.92 seconds.
```

In [228]:

```
random_search.best_estimator_
```

Out[228]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4,
              enable_categorical=False, gamma=0.1, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=3,
              min_child_weight=5, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [232]:

```
xgbc = xgboost.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4,
              enable_categorical=False, gamma=0.1, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=3,
              min_child_weight=5, monotone_constraints='()',
              n_estimators=100, n_jobs=8, num_parallel_tree=1, predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
xgbc.fit(x_train,y_train)
xgbc_pred = xgbc.predict(x_test)
```

```
[17:40:19] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner
.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective '
binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd
like to restore the old behavior.
```

In [233]:

```
print("Training Score:", xgbc.score(x_train, y_train))
print("Testing Score:", xgbc.score(x_test, y_test))
```

```
Training Score: 0.8876221498371335
Testing Score: 0.7792207792207793
```

In [234]:

```
XGB_pred = xgbc.predict(x_test)
print("Actual : " , y_test[0:5])
print("Predicted : ", XGB_pred[0:5])
```

```
Actual :  661    1
122    0
113    0
14     1
529    0
Name: Outcome, dtype: int64
Predicted :  [1 0 0 1 0]
```

9. Extra Tree Classifier

In [294]:

```
# Splitting the dataset into training and test set.
#test_size 0.3 means for testing data 30% and training data 70%
#Removing unnessary columns
x = df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']]
y = df.iloc[:,-1]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

from sklearn.ensemble import ExtraTreesClassifier

Etc = ExtraTreesClassifier(n_estimators=100,random_state=0)
Etc.fit(x_train,y_train)
```

Out[294]:

```
ExtraTreesClassifier(random_state=0)
```

In [295]:

```
print("Training Score:", Etc.score(x_train, y_train))
print("Testing Score:", Etc.score(x_test, y_test))
```

```
Training Score: 1.0
Testing Score: 0.8051948051948052
```

In [297]:

```
Etc.feature_names_in_
```

Out[297]:

```
array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
       'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'], dtype=object)
```

In [298]:

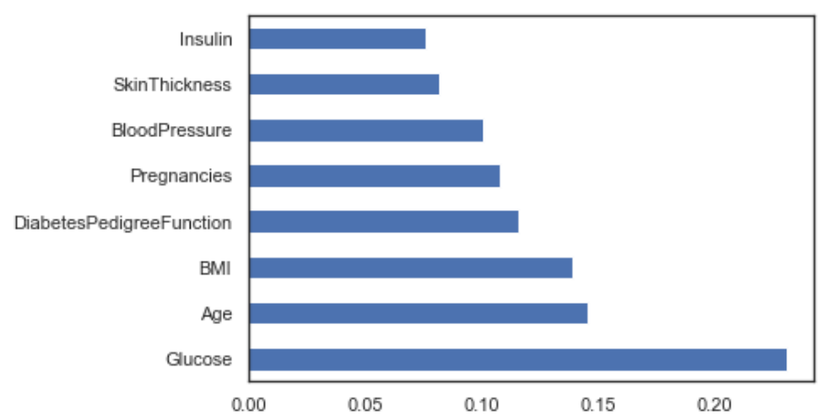
```
Etc.feature_importances_
```

Out[298]:

```
array([0.10821986, 0.23101101, 0.10104879, 0.08205671, 0.07654321,
       0.13886959, 0.1162792 , 0.14597163])
```

In [301]:

```
feature_imp = pd.Series(Etc.feature_importances_,index=x.columns)
feature_imp.nlargest(10).plot(kind='barh')
plt.show()
```



10. LightGBM

In [238]:

```
!pip install lightgbm
```

```
Collecting lightgbm
  Downloading lightgbm-3.3.1-py3-none-win_amd64.whl (1.0 MB)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\users\lenovo\anaconda3\envs\mydl\lib\site-packages (from lightgbm) (1.0.1)
Requirement already satisfied: wheel in c:\users\lenovo\anaconda3\envs\mydl\lib\site-packages (from lightgbm) (0.37.0)
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\envs\mydl\lib\site-packages (from lightgbm) (1.7.3)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\envs\mydl\lib\site-packages (from lightgbm) (1.21.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenovo\anaconda3\envs\mydl\lib\site-packages (from scikit-learn!=0.22.0->lightgbm) (3.0.0)
Requirement already satisfied: joblib>=0.11 in c:\users\lenovo\anaconda3\envs\mydl\lib\site-packages (from scikit-learn!=0.22.0->lightgbm) (1.1.0)
Installing collected packages: lightgbm
Successfully installed lightgbm-3.3.1
```

In [241]:

```
from lightgbm import LGBMClassifier

Lgbm = LGBMClassifier(n_estimators=100,random_state=0)
Lgbm.fit(x_train,y_train)
```

Out[241]:

```
LGBMClassifier(random_state=0)
```

In [242]:

```
print("Training Score:", Lgbm.score(x_train, y_train))
print("Testing Score:", Lgbm.score(x_test, y_test))
```

```
Training Score: 1.0
Testing Score: 0.7857142857142857
```

After extensive data analysis and I tried different classification models to see how it performs on the dataset. I got pretty good results with accuracy, roc, precision and recall scores.
Thus , We will take the best model out of it that is Gradient Boosting Model and K Nearset Neighbor Model.

K Nearest Neighbor Model

In [244]:

```
knn_pred = knn.predict(x_test)

print("Actual Outcomes : " , y_test[0:5])
print(" ")
print("Predicted Outcomes : ", GB_pred[0:5])
```

```
Actual Outcomes :  661    1
122    0
113    0
14     1
529    0
Name: Outcome, dtype: int64

Predicted Outcomes :  [1 0 0 1 0]
```

In [245]:

```
misclassified = np.where(y_test!=knn_pred)
misclassified
```

Out[245]:

```
(array([ 10,  21,  27,  30,  36,  39,  47,  48,  49,  50,  58,  59,  63,
         73,  77,  86,  94,  96, 105, 111, 117, 127, 135, 137, 138, 141,
        149], dtype=int64),)
```

In [261]:

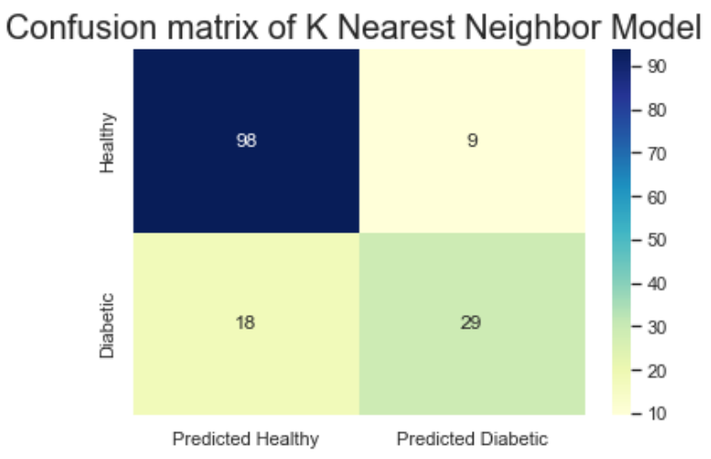
```
print("Total Misclassified Samples: ",len(misclassified[0]))
print(" ")
print("Actual first row outcome :",y_test[:1])
```

```
print("Predicted first row outcome :",knn_pred[:1])
```

```
Total Misclassified Samples:  27

Actual first row outcome : 661    1
Name: Outcome, dtype: int64
Predicted first row outcome : [1]
```

In [273]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
print('\n')
corr_matrix = confusion_matrix(y_test,knn_pred)
sns.heatmap(corr_matrix ,cmap="YlGnBu",annot=True,robust=True,linecolor='black',yticklabels
= ['Healthy', 'Diabetic'], xticklabels = ['Predicted Healthy', 'Predicted Diabetic'])
plt.title("Confusion matrix of K Nearest Neighbor Model " ,fontsize=20)
plt.show()
print('\n')
target_names = ['Healthy Patient','Diabetic Patient']
print(classification_report(y_test,knn_pred,target_names=target_names))
```




```
                   precision    recall  f1-score   support

 Healthy Patient       0.84      0.92      0.88       107
Diabetic Patient       0.76      0.62      0.68        47

        accuracy                           0.82       154
       macro avg       0.80      0.77      0.78       154
    weighted avg       0.82      0.82      0.82       154
```

In [275]:

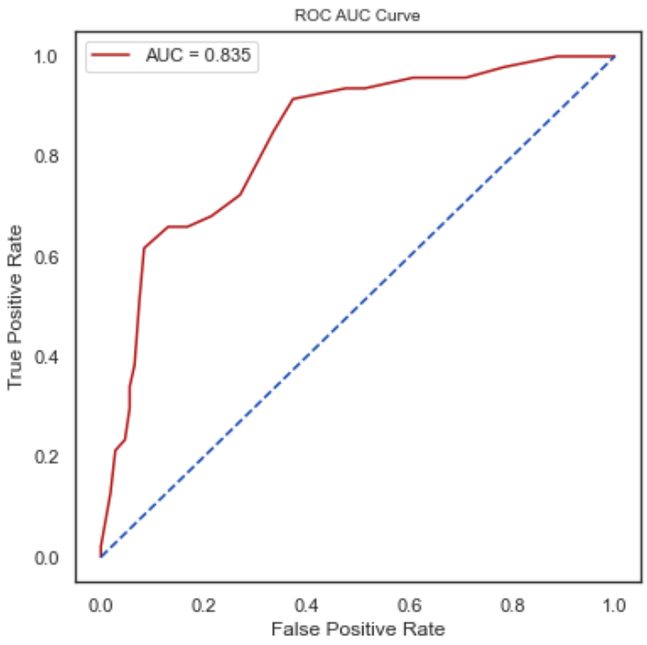
```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, ConfusionMatrix
Display, precision_score, recall_score, f1_score, classification_report, roc_curve, plot_ro
c_curve, auc, precision_recall_curve, plot_precision_recall_curve, average_precision_score
from sklearn.model_selection import cross_val_score
print(f'ROC AUC score: {roc_auc_score(y_test, knn_pred)}')
print('Accuracy Score: ',accuracy_score(y_test, knn_pred))
```

```
ROC AUC score: 0.7664545635315172
Accuracy Score:  0.8246753246753247
```

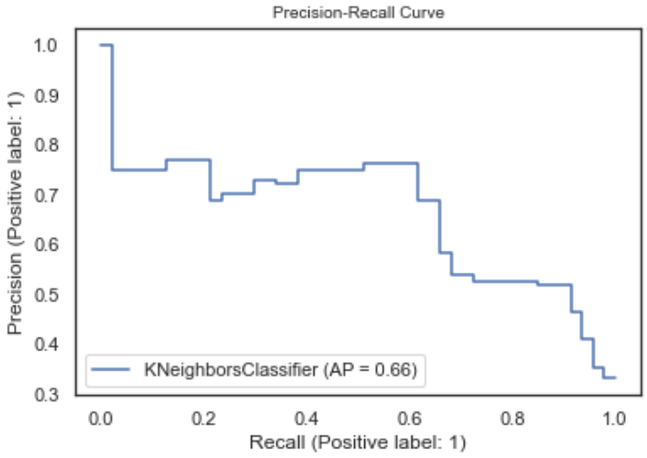
In [276]:

```
knn_prob = knn.predict_proba(x_test)[:,1]

# Roc AUC Curve
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, knn_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)

sns.set_theme(style = 'white')
plt.figure(figsize = (6, 6))
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f' %
roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC AUC Curve',fontsize=10)
plt.legend()
plt.show()

#Precision Recall Curve
plt.figure(figsize = (6, 6))
```

```
average_precision = average_precision_score(y_test, knn_prob)
disp = plot_precision_recall_curve(knn, x_test, y_test)
plt.title('Precision-Recall Curve',fontsize=10)
plt.show()
```




```
<Figure size 432x432 with 0 Axes>
```

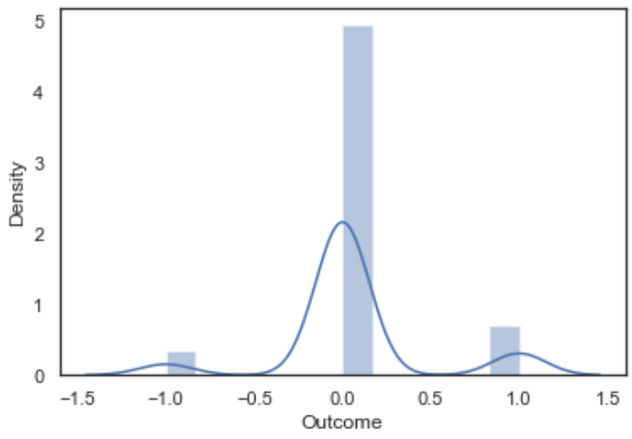


In [363]:

```
sns.distplot(y_test-knn_pred)
```

Out[363]:

```
<AxesSubplot:xlabel='Outcome', ylabel='Density'>
```



Gradient Boosting Model

In [278]:

```
GB_pred = tunning.predict(x_test)

print("Actual Outcomes : " , y_test[0:5])
print(" ")
print("Predicted Outcomes : ", GB_pred[0:5])
```

```
Actual Outcomes :  661    1
122    0
113    0
14     1
529    0
Name: Outcome, dtype: int64

_
Predicted Outcomes :  [1 0 0 1 0]
```

In [279]:

```
misclassified = np.where(y_test!=GB_pred)
misclassified
```

Out[279]:

```
(array([  9,  15,  21,  27,  39,  47,  48,  50,  53,  58,  59,  61,  68,
         73,  75,  86,  94,  96, 105, 111, 117, 129, 133, 137, 141, 149],
       dtype=int64),)
```

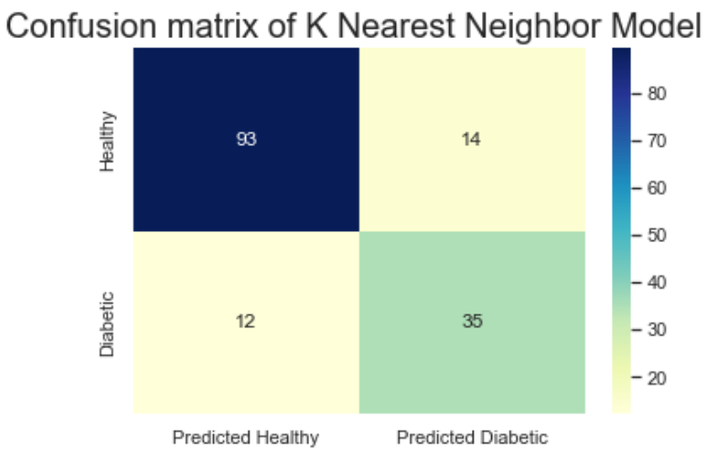
In [284]:

```
print("Total Misclassified Samples: ",len(misclassified[0]))
print(" ")
print("Actual first row outcome :",y_test[:3])
print("Predicted first row outcome :",knn_pred[:3])
```

```
Total Misclassified Samples:  26

_
Actual first row outcome : 661    1
122    0
113    0
Name: Outcome, dtype: int64
Predicted first row outcome : [1 0 0]
```

In [285]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
print('\n')
corr_matrix = confusion_matrix(y_test,GB_pred)
sns.heatmap(corr_matrix ,cmap="YlGnBu",annot=True,robust=True,linecolor='black',yticklabels
= ['Healthy', 'Diabetic'], xticklabels = ['Predicted Healthy', 'Predicted Diabetic'])
plt.title("Confusion matrix of K Nearest Neighbor Model " ,fontsize=20)
plt.show()
print('\n')
target_names = ['Healthy Patient','Diabetic Patient']
print(classification_report(y_test,GB_pred,target_names=target_names))
```




```
                   precision    recall  f1-score   support

 Healthy Patient       0.89      0.87      0.88       107
Diabetic Patient       0.71      0.74      0.73        47

        accuracy                           0.83       154
       macro avg       0.80      0.81      0.80       154
    weighted avg       0.83      0.83      0.83       154
```

In [286]:

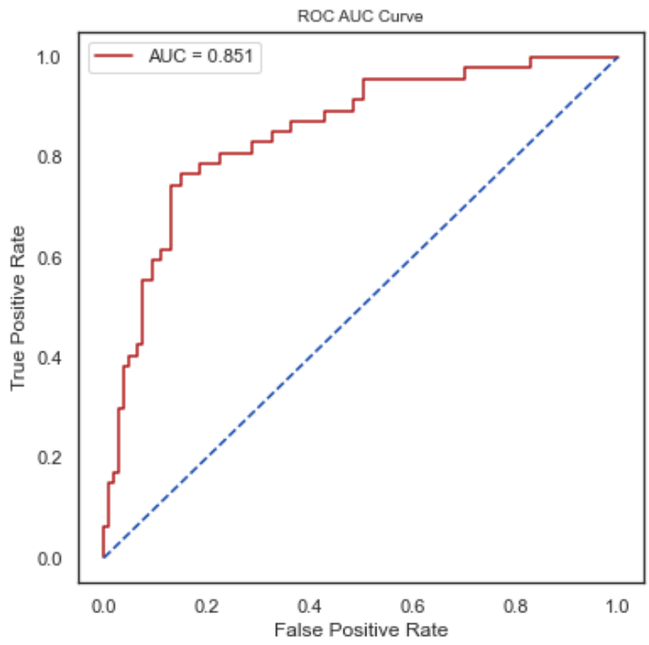
```
print(f'ROC AUC score: {roc_auc_score(y_test, GB_pred)}')
print('Accuracy Score: ',accuracy_score(y_test,GB_pred))
```

```
ROC AUC score: 0.8069198647842514
Accuracy Score:  0.8311688311688312
```

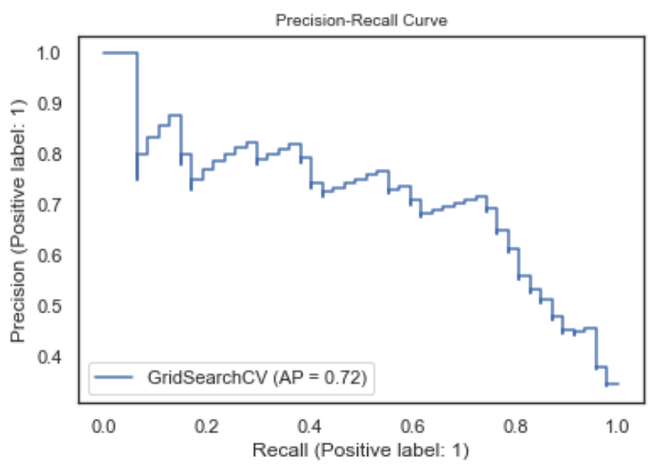
In [287]:

In [287]:

```
GB_prob = tunning.predict_proba(x_test)[:,1]

# Roc AUC Curve
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, GB_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)

sns.set_theme(style = 'white')
plt.figure(figsize = (6, 6))
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f' %
roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC AUC Curve',fontsize=10)
plt.legend()
plt.show()

#Precision Recall Curve
plt.figure(figsize = (6, 6))
average_precision = average_precision_score(y_test,GB_prob)
disp = plot_precision_recall_curve(tunning, x_test, y_test)
plt.title('Precision-Recall Curve',fontsize=10)
plt.show()
```



<Figure size 432x432 with 0 Axes>

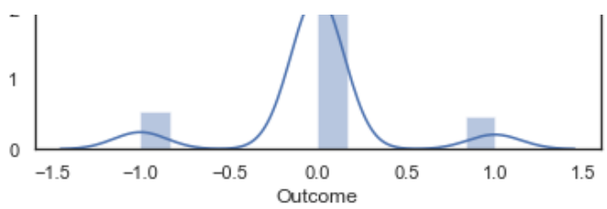


In [367]:

```
sns.distplot(y_test-GB_pred)
```

Out[367]:

<AxesSubplot:xlabel='Outcome', ylabel='Density'>



Saving the classifier

In [369]:

```
import pickle
pickle.dump(tunning, open('GB_classifier.pkl', 'wb'))
```

In [370]:

```
pickle.dump(knn, open('knn_classifier.pkl', 'wb'))
```

In [371]:

```
pickle.dump(sc, open('scaler.pkl', 'wb'))
```

In [ ]:

```

```