Healthcare PGP

December 29, 2022

1 Healthcare PGP

1.0.1 Description

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.

1.0.2 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

Variables	Description
Outcome	Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

1.1 Project Task: Week 1

1.1.1 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

2. Visually explore these variables using histograms. Treat the missing values accordingly.
3. 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.

[126]:
```
import numpy as np
import pandas as pd


import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

%matplotlib inline
```

[127]:
```
#This function will draw histogram by passing data column name and tilte name
def plot_histogram(data_val,title_name):
    plt.figure(figsize=[10,6])
    plt.hist(data_val,edgecolor="red")
    #plt.grid(axis='y', alpha=0.75)
    plt.title(title_name,fontsize=15)
    plt.show()
```

[128]:
```
#function to get total count of zeros and outcome details together
def get_zeros_outcome_count(data,column_name):
    count = data[data[column_name] == 0].shape[0]
    print("Total No of zeros found in " + column_name + " : " + str(count))
```

```
    print(data[data[column_name] == 0].groupby('Outcome')['Age'].count())
```

[129]:
```
#function to create scatter plot
def create_scatter_plot(first_value,second_value,x_label,y_label,colour):
    plt.scatter(first_value,second_value, color=[colour])
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    title_name = x_label + '&' + y_label
    plt.title(title_name)
    plt.show()
```

[130]:
```
data = pd.read_csv('./health care diabetes.csv')
```

[131]:
```
data.head()
```

[131]:

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

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

[132]:
```
data.isnull().any()
```

[132]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

[133]:
```
data.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
```

[134]:
```
#Get count of outcome column
data.groupby('Outcome').size()
```

[134]:
```
Outcome
0    500
1    268
dtype: int64
```

[135]:
```
Positive = data[data['Outcome']==1]
Positive.head(5)
```

[135]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1

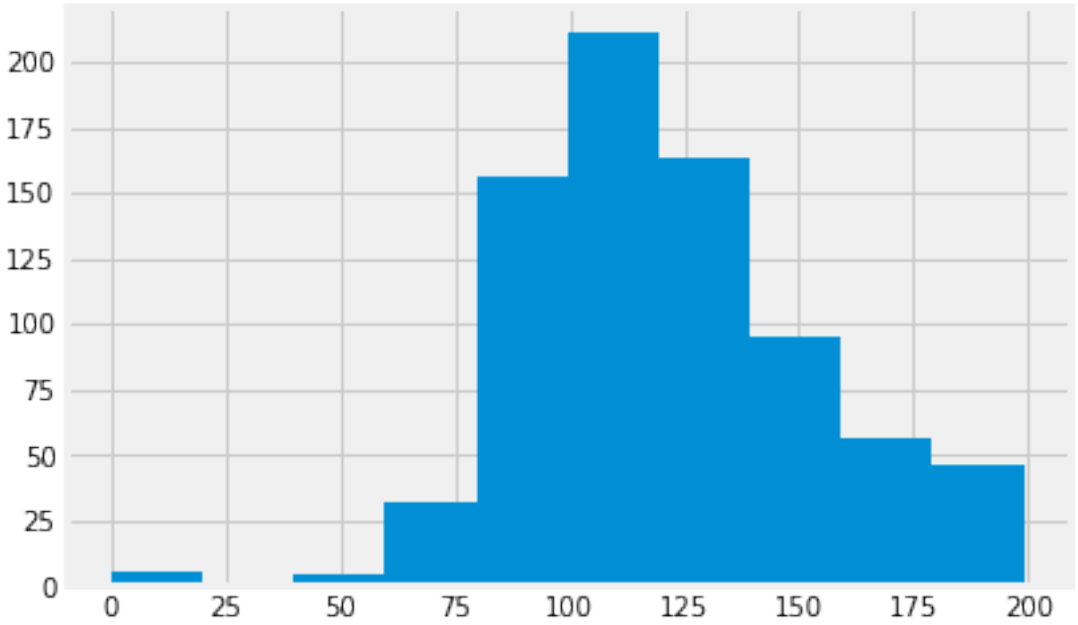
[136]:
```
data['Glucose'].value_counts().head(7)
```

[136]:
```
100    17
99     17
129    14
125    14
111    14
106    14
```

```
95     13
Name: Glucose, dtype: int64
```

[137]:
```
plt.hist(data['Glucose'])
```

[137]:
```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <BarContainer object of 10 artists>)
```



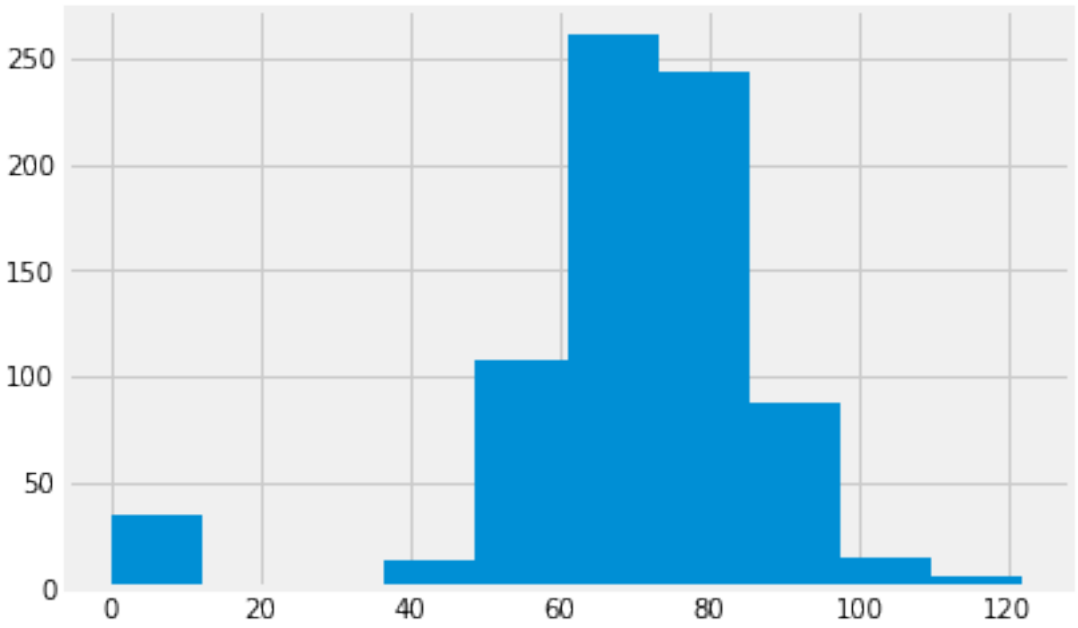
[138]:
```
data['BloodPressure'].value_counts().head(7)
```

[138]:
```
70    57
74    52
68    45
78    45
72    44
64    43
80    40
Name: BloodPressure, dtype: int64
```

[139]:
```
plt.hist(data['BloodPressure'])
```

[139]:
```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
```

```
 <BarContainer object of 10 artists>)
```

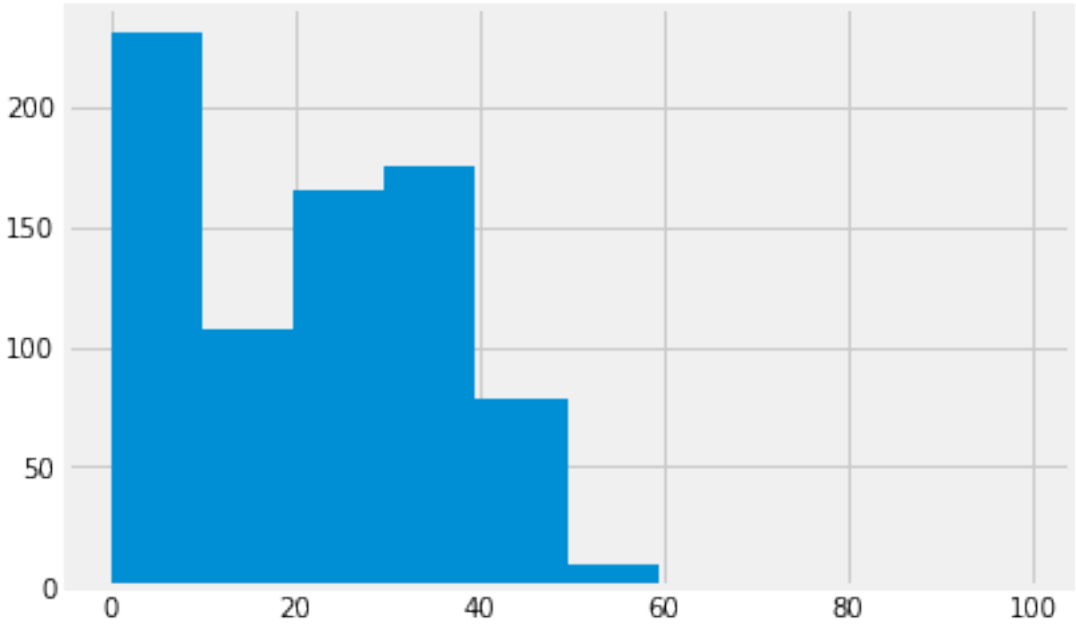


[140]:
```
data['SkinThickness'].value_counts().head(7)
```

[140]:
```
0     227
32     31
30     27
27     23
23     22
33     20
18     20
Name: SkinThickness, dtype: int64
```

[141]:
```
plt.hist(data['SkinThickness'])
```

[141]:
```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```

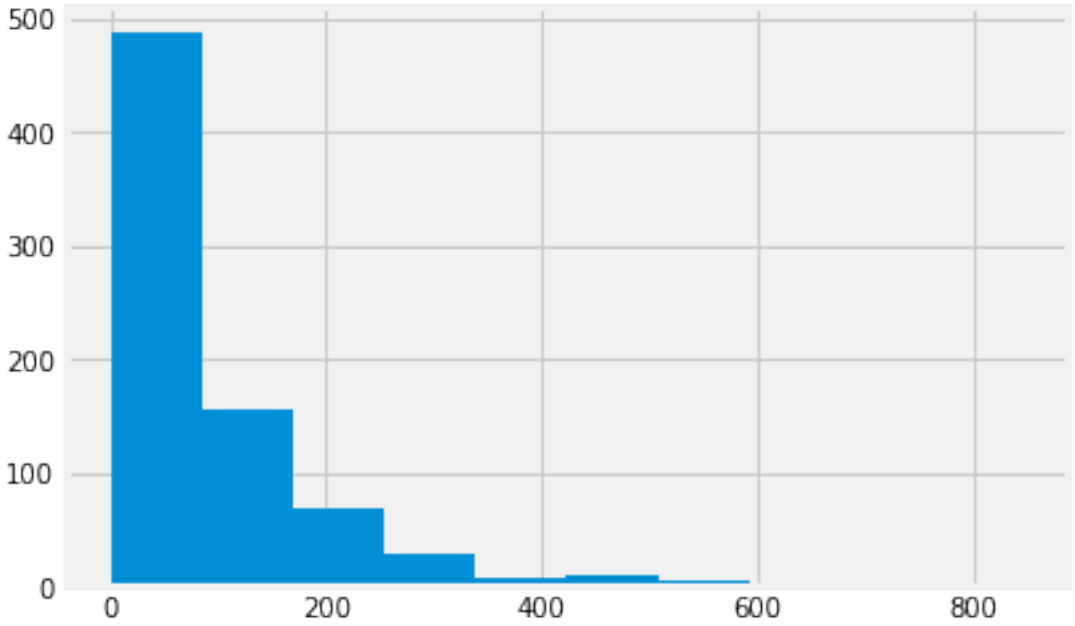


[142]:
```
data['Insulin'].value_counts().head(7)
```

[142]:
```
0      374
105     11
140      9
130      9
120      8
100      7
94       7
Name: Insulin, dtype: int64
```

[143]:
```
plt.hist(data['Insulin'])
```

[143]:
```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```

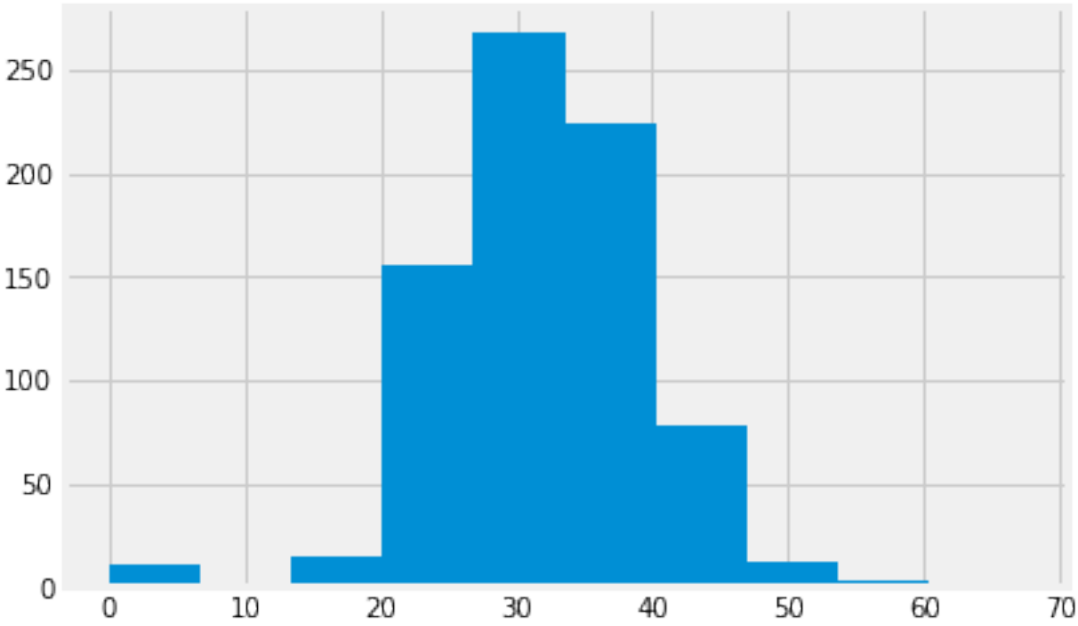


[144]:
```
data['BMI'].value_counts().head(7)
```

[144]:
```
32.0    13
31.6    12
31.2    12
0.0     11
33.3    10
32.4    10
32.8     9
Name: BMI, dtype: int64
```

[145]:
```
plt.hist(data['BMI'])
```

[145]:
```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <BarContainer object of 10 artists>)
```



[146]:
```
data.describe().transpose()
```

[146]:

	count	mean	std	min	25%
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000
Glucose	768.0	120.894531	31.972618	0.000	99.00000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000
Insulin	768.0	79.799479	115.244002	0.000	0.00000
BMI	768.0	31.992578	7.884160	0.000	27.30000
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375
Age	768.0	33.240885	11.760232	21.000	24.00000
Outcome	768.0	0.348958	0.476951	0.000	0.00000

	50%	75%	max
Pregnancies	3.0000	6.00000	17.00
Glucose	117.0000	140.25000	199.00
BloodPressure	72.0000	80.00000	122.00
SkinThickness	23.0000	32.00000	99.00
Insulin	30.5000	127.25000	846.00
BMI	32.0000	36.60000	67.10
DiabetesPedigreeFunction	0.3725	0.62625	2.42
Age	29.0000	41.00000	81.00
Outcome	0.0000	1.00000	1.00

1.2 Now lets create a count (frequency) plot describing the data types and the count of variables.

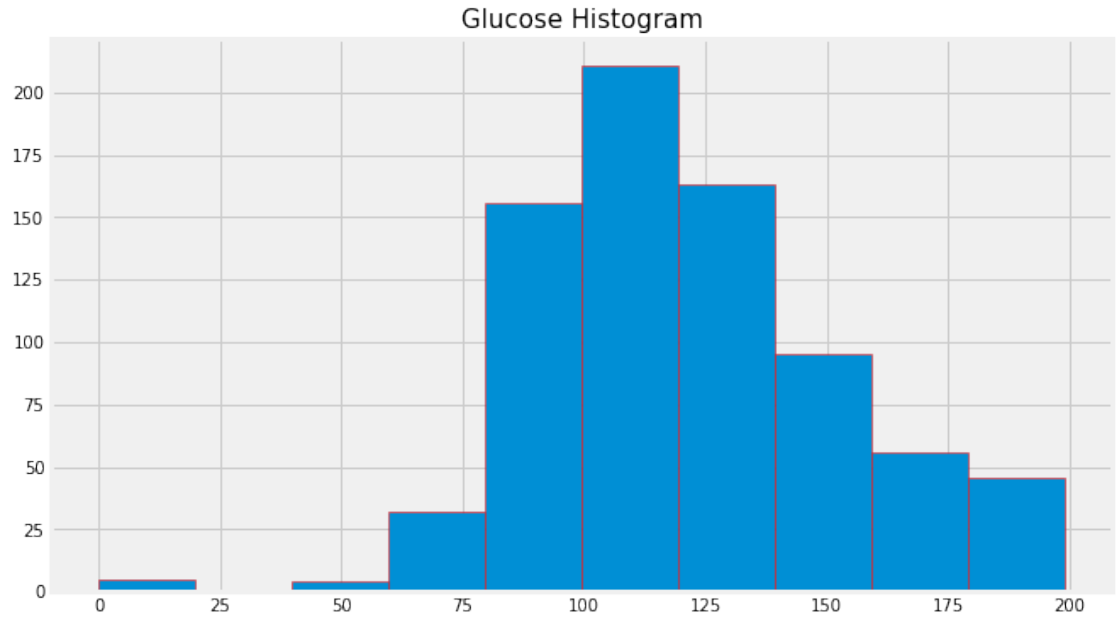
[147]:
```
data['Glucose'].value_counts().head(10)
```

[147]:
```
100    17
99     17
129    14
125    14
111    14
106    14
95     13
108    13
105    13
102    13
Name: Glucose, dtype: int64
```

[148]:
```
data['Glucose']
```

[148]:
```
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose, Length: 768, dtype: int64
```

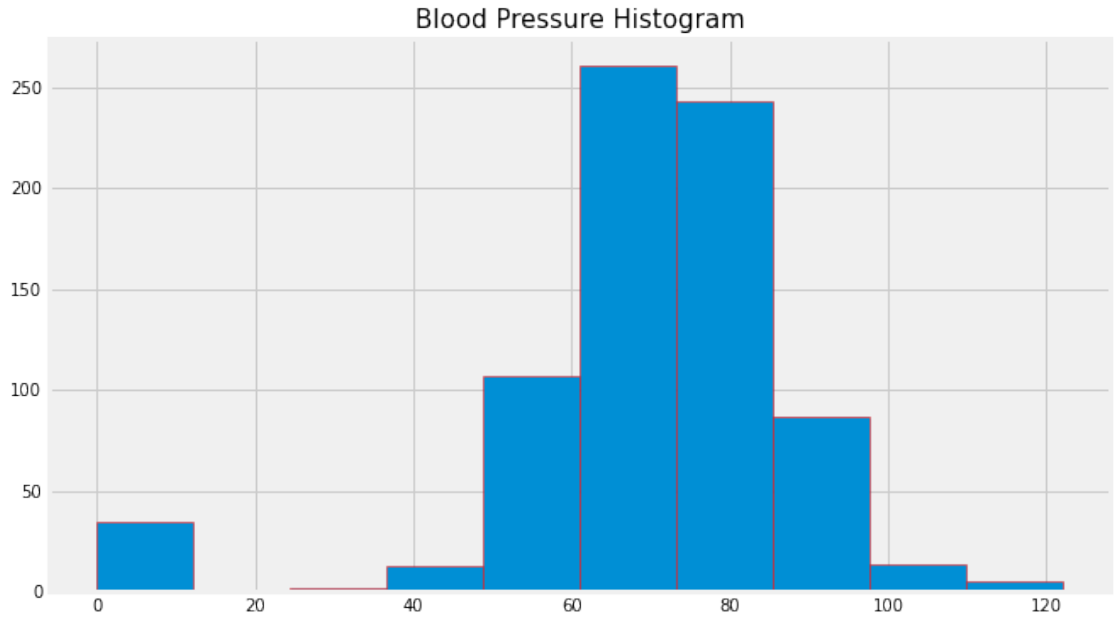
[149]:
```
#Drawing histogram for glucose
plot_histogram(data['Glucose'],'Glucose Histogram')
```



[150]:
```
#Now will check for another column bloodpressure
data['BloodPressure'].value_counts().head(7)
```

[150]:
```
70    57
74    52
68    45
78    45
72    44
64    43
80    40
Name: BloodPressure, dtype: int64
```

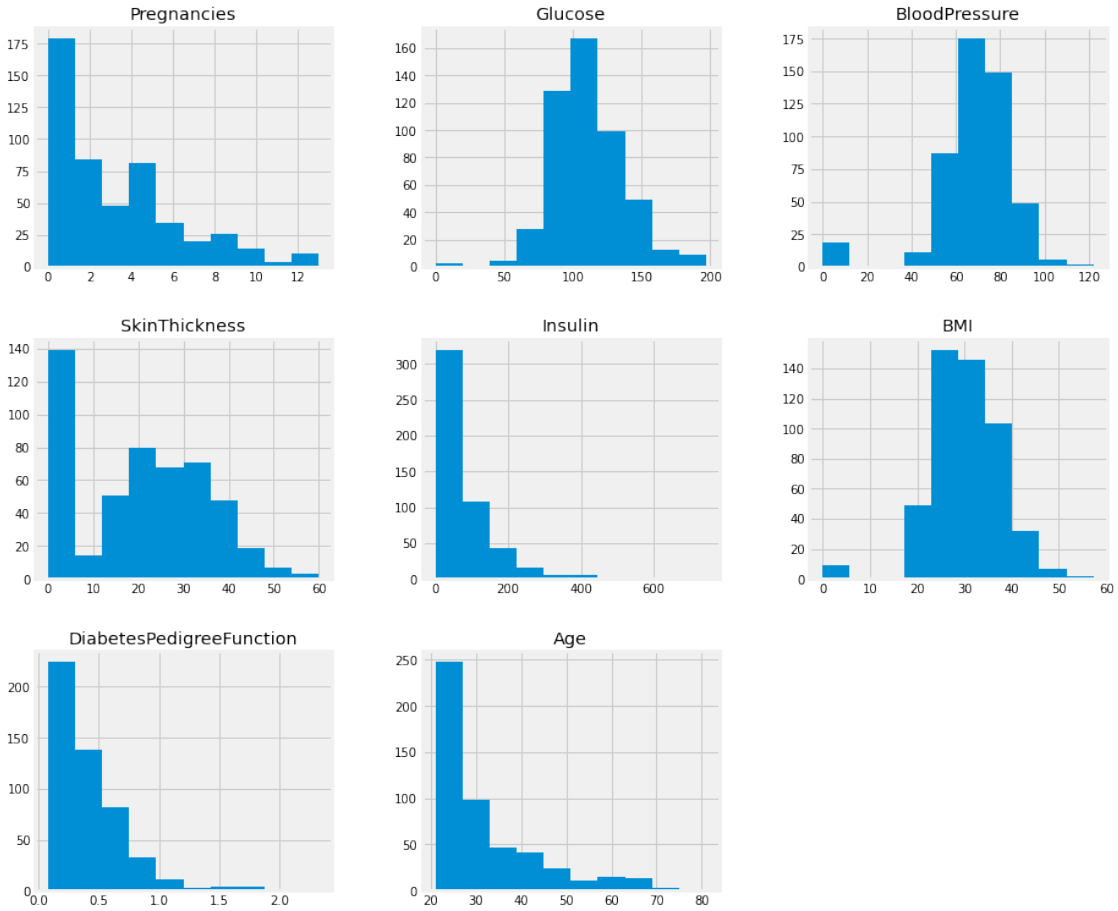
[151]:
```
#Drawing Bloodpressure histogram
plot_histogram(data['BloodPressure'],'Blood Pressure Histogram')
```

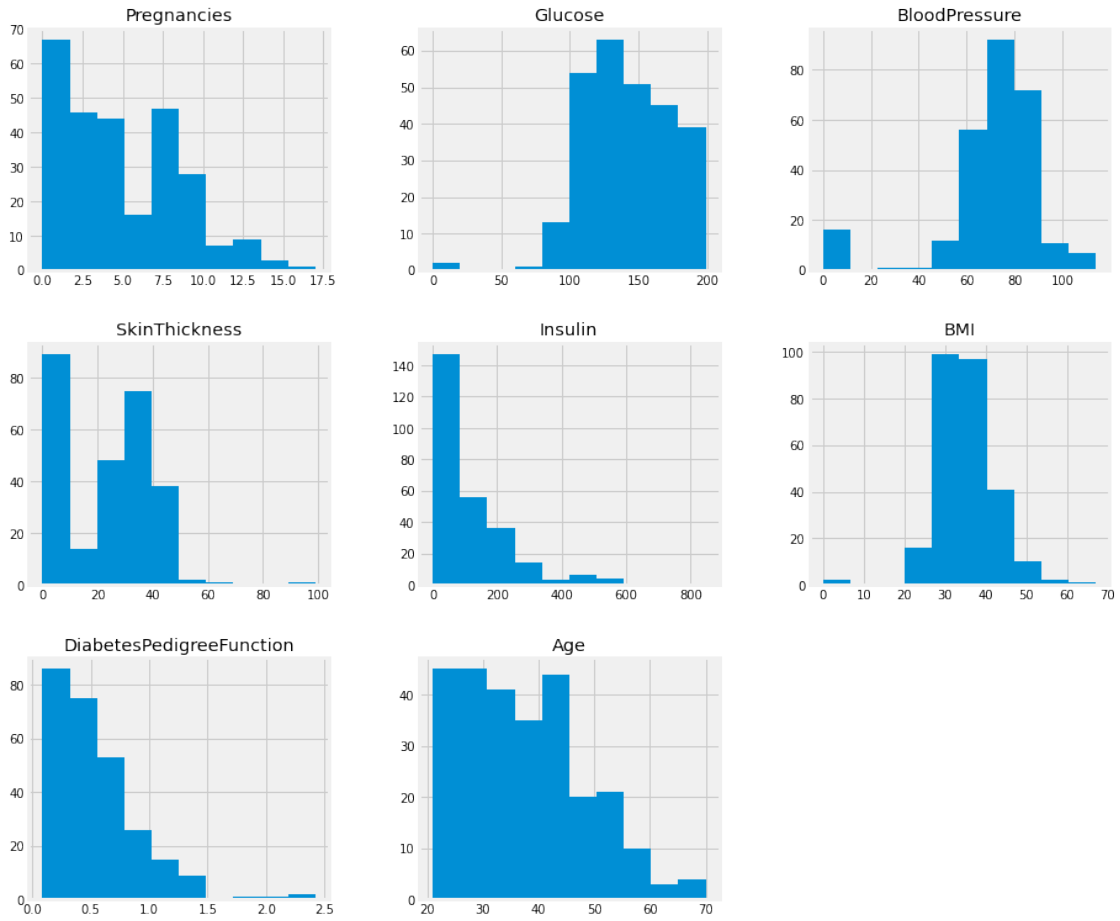


Instead of creating historam one by one. With the help of group by and Outcome we can create all column hisotram

[152]:
```
data.groupby('Outcome').hist(figsize=(14, 13))
```

[152]:
```
Outcome
0    [[AxesSubplot(0.08,0.670278;0.241667x0.209722)…
1    [[AxesSubplot(0.08,0.670278;0.241667x0.209722)…
dtype: object
```


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age



After analyzing the histogram we can identify that there are some outliers in some columns.

For Example:-

```
<li>BloodPressure - A living person cannot have a diastolic blood pressure of zero.</li>
<li>Plasma glucose levels - Zero is invalid number as fasting glucose level would never be as
<li>Skin Fold Thickness - For normal people, skin fold thickness can't be less than 10 mm bette
<li>BMI: Should not be 0 or close to zero unless the person is really underweight which could
<li>Insulin: In a rare situation a person can have zero insulin but by observing</li>
```

[153]:
```
#Checking count of zeros in blood pressure
get_zeros_outcome_count(data,'BloodPressure')
```

```
Total No of zeros found in BloodPressure : 35
Outcome
0    19
1    16
Name: Age, dtype: int64
```

[154]:
```
#Checking count of zeros in Glucose
get_zeros_outcome_count(data,'Glucose')
```

```
Total No of zeros found in Glucose : 5
Outcome
0    3
1    2
Name: Age, dtype: int64
```

[155]:
```
#Checking count of zeros in SkinThickness
get_zeros_outcome_count(data,'SkinThickness')
```

```
Total No of zeros found in SkinThickness : 227
Outcome
0    139
1     88
Name: Age, dtype: int64
```

[156]:
```
#Checking count of zeros in BMI
get_zeros_outcome_count(data,'BMI')
```

```
Total No of zeros found in BMI : 11
Outcome
0    9
1    2
Name: Age, dtype: int64
```

[157]:
```
#Checking count of zeros in BMI
get_zeros_outcome_count(data,'Insulin')
```

```
Total No of zeros found in Insulin : 374
Outcome
0    236
1    138
Name: Age, dtype: int64
```

After analysing above data we found lots of 0 in Insulin and SkinThickness and removing them or putting mean value will not good dataset. However, we can remove ”BloodPressure”, ”BMI” and ”Glucose” zeros row

[158]:
```
diabetes_data_mod = data[(data.BloodPressure != 0) & (data.BMI != 0) & (data.
↪Glucose != 0)]
print(diabetes_data_mod.shape)
```

```
(724, 9)
```

[159]:
```
#Now we will check the stats of data after removing BloodPressure, BMI and␣
↪Glucose 0 rows
```

```
diabetes_data_mod.describe().transpose()
```

[159]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	724.0	3.866022	3.362803	0.000	1.000	3.000	6.0000	17.00
Glucose	724.0	121.882597	30.750030	44.000	99.750	117.000	142.0000	199.00
BloodPressure	724.0	72.400552	12.379870	24.000	64.000	72.000	80.0000	122.00
SkinThickness	724.0	21.443370	15.732756	0.000	0.000	24.000	33.0000	99.00
Insulin	724.0	84.494475	117.016513	0.000	0.000	48.000	130.5000	846.00
BMI	724.0	32.467127	6.888941	18.200	27.500	32.400	36.6000	67.10
DiabetesPedigreeFunction	724.0	0.474765	0.332315	0.078	0.245	0.379	0.6275	2.42
Age	724.0	33.350829	11.765393	21.000	24.000	29.000	41.0000	81.00
Outcome	724.0	0.343923	0.475344	0.000	0.000	0.000	1.0000	1.00

1.2.1 Data Exploration:

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

[160]:
```
#Lets create positive variable and store all 1 value Outcome data
Positive = diabetes_data_mod[diabetes_data_mod['Outcome']==1]
Positive.head(5)
```

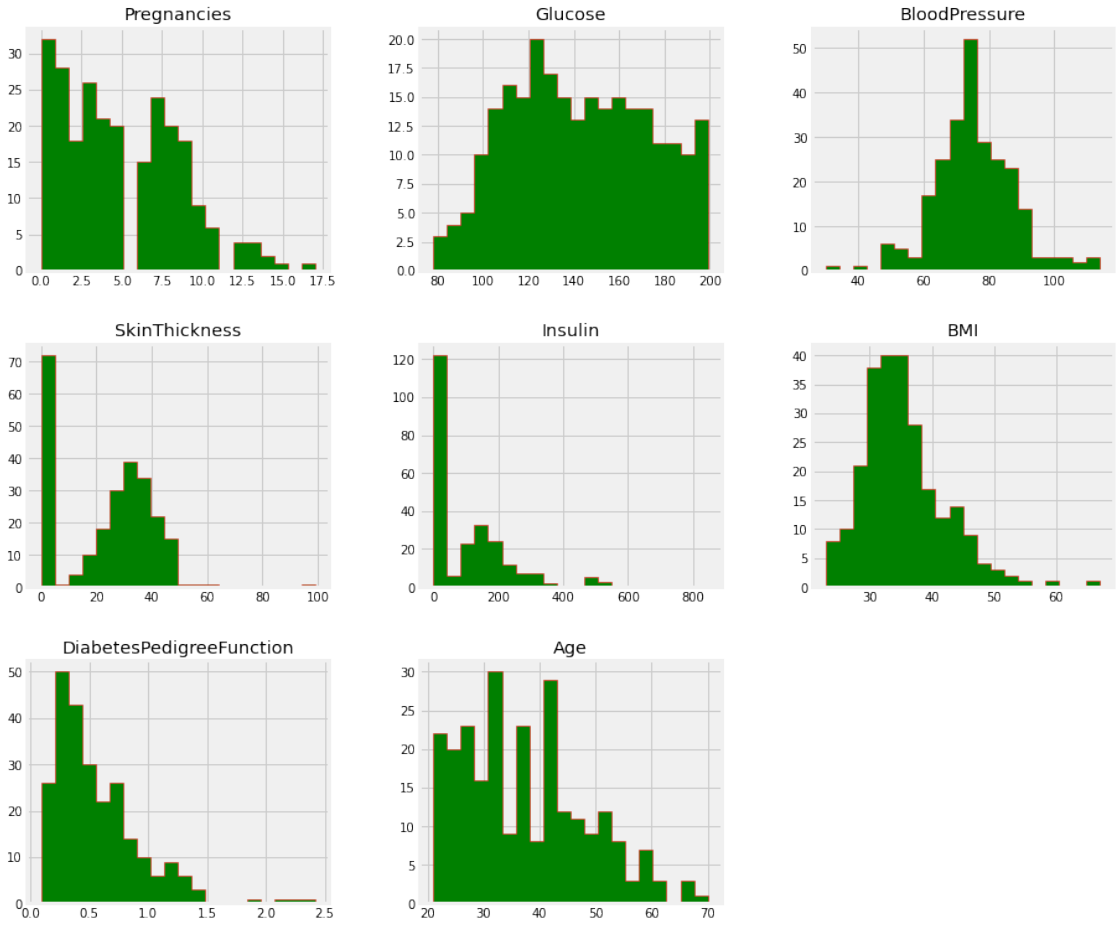
[160]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3			
4	0	137	40	35	168	43.1			
6	3	78	50	32	88	31.0			
8	2	197	70	45	543	30.5			

```
2                       0.672   32          1
4                       2.288   33          1
6                       0.248   26          1
8                       0.158   53          1
```

[161]:
```
Positive.groupby('Outcome').hist(figsize=(14,13),histtype='stepfilled',bins=20,color="green",edgecolor="red")
```

[161]:
```
Outcome
1    [[AxesSubplot(0.08,0.670278;0.241667x0.209722)…
dtype: object
```



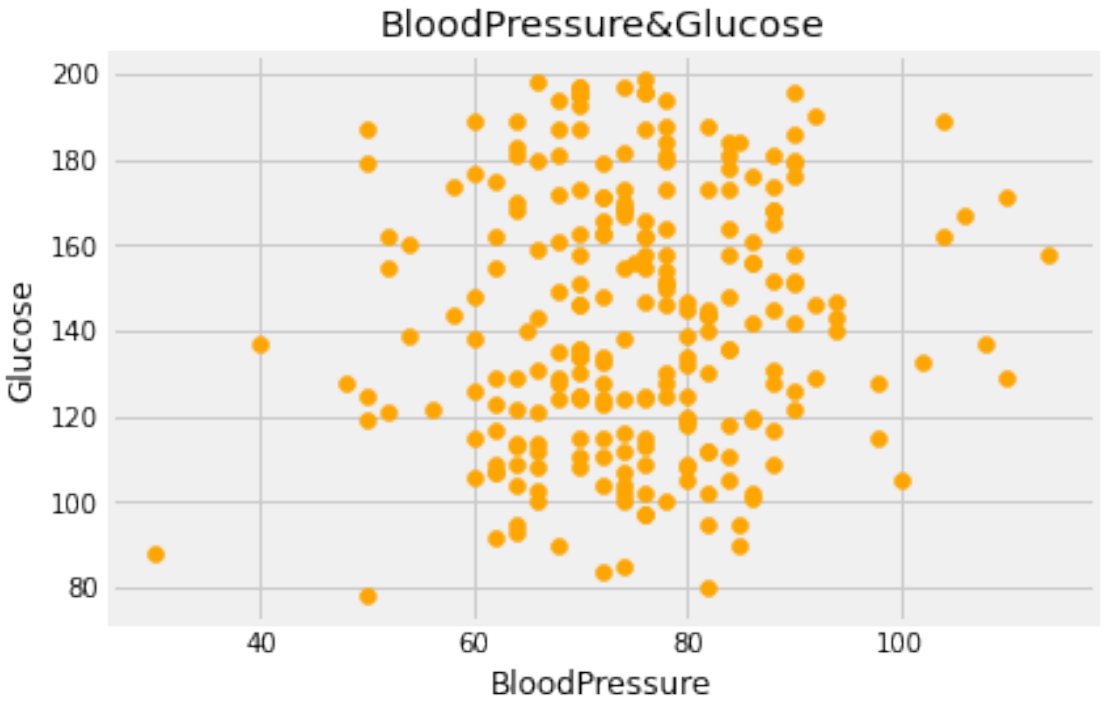
From positive outcome histogram we can see the outlier in SkinThickness, BMI & Insulin.

Now creating scatter plot for positive outcome

[162]:
```
BloodPressure = Positive['BloodPressure']
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
```

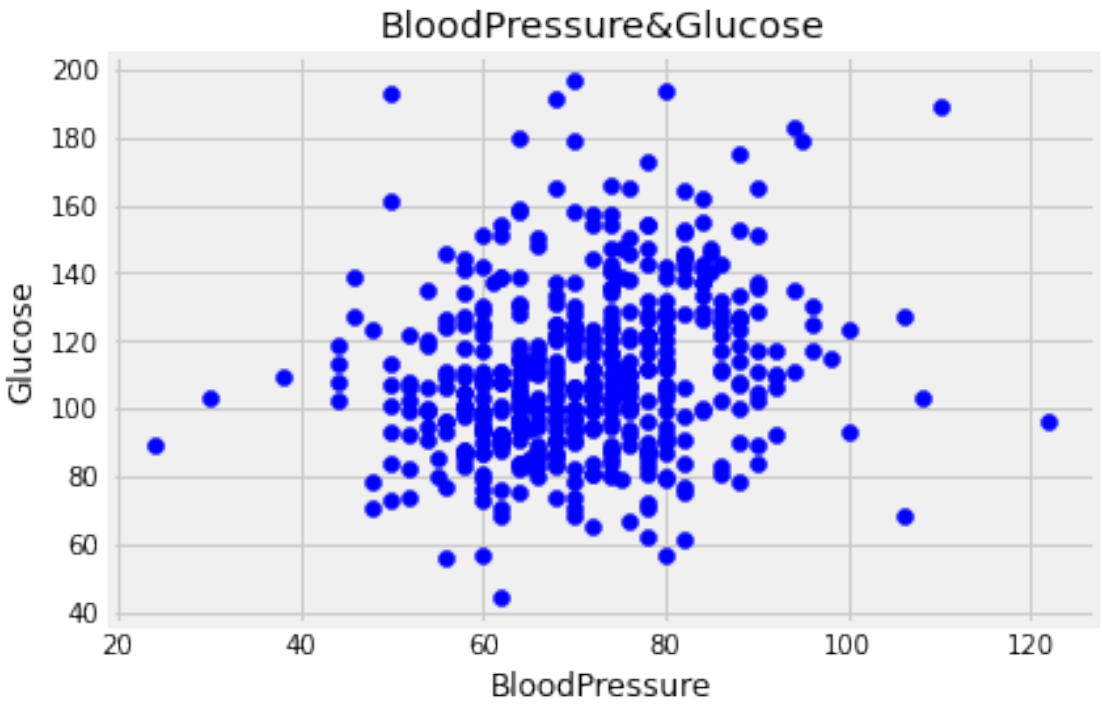
```
Insulin = Positive['Insulin']
BMI = Positive['BMI']
```

[163]:
```
create_scatter_plot(Positive['BloodPressure'],Positive['Glucose'],'BloodPressure','Glucose','
```



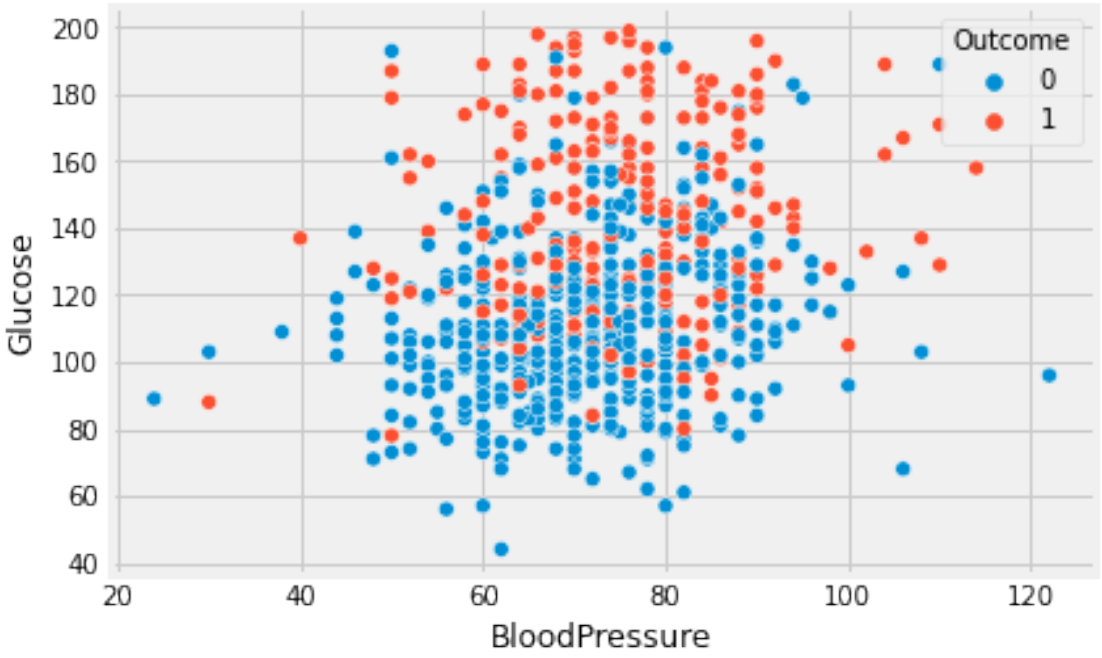
[164]:
```
#Creating scatter plot for negative outcome
Negative = diabetes_data_mod[diabetes_data_mod['Outcome']==0]
```

[165]:
```
create_scatter_plot(Negative['BloodPressure'],Negative['Glucose'],'BloodPressure','Glucose','b
```



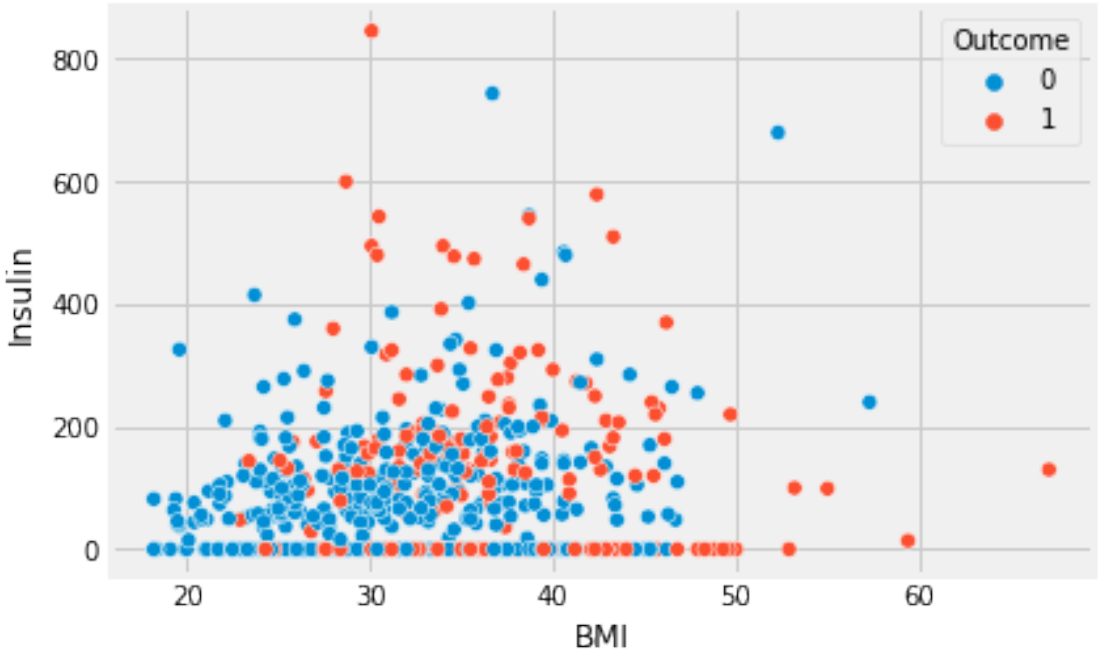
We don't need to create negative scatter plot, but I am creating it to verify the values and points which we will get for both outcome value using sns scatterplot.

[166]:
```
g =sns.scatterplot(x= "BloodPressure" ,y= "Glucose",
                   hue="Outcome",
                   data=diabetes_data_mod);
```

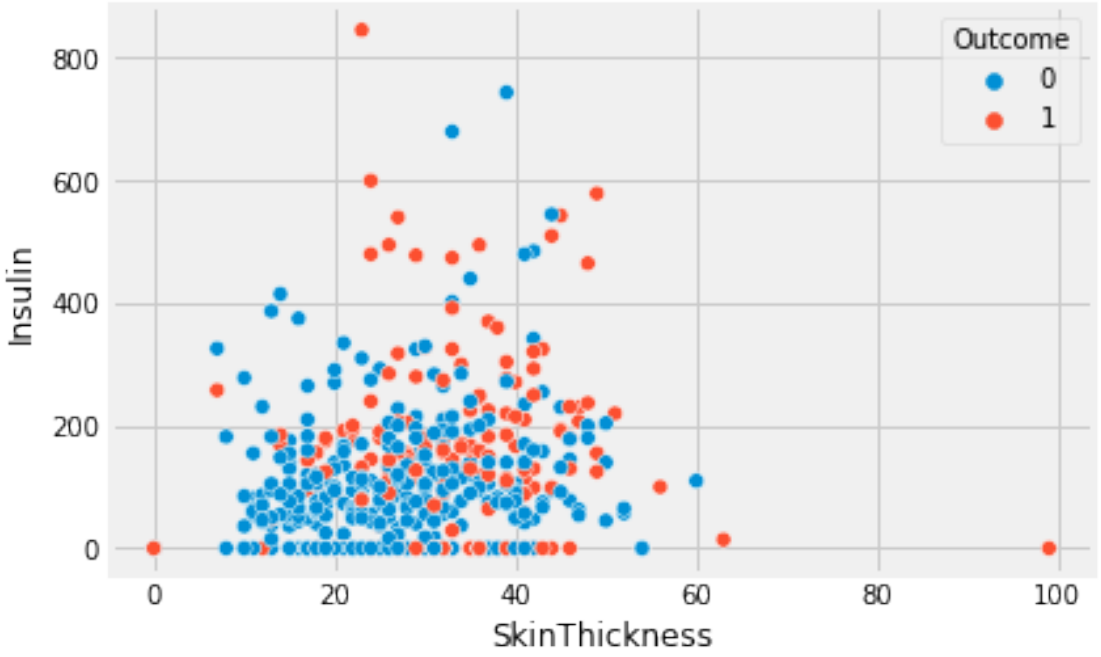


As you can compare postive & negative scatter plot with sns scatter plot all the value is matching, so now I will create common scatter plot for both outcome.

[167]:
```
B =sns.scatterplot(x= "BMI" ,y= "Insulin",
                   hue="Outcome",
                   data=diabetes_data_mod);
```



[168]:
```
S =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",
                   hue="Outcome",
                   data=diabetes_data_mod);
```



[169]:
```
### correlation matrix
diabetes_data_mod.corr()
```

[169]:

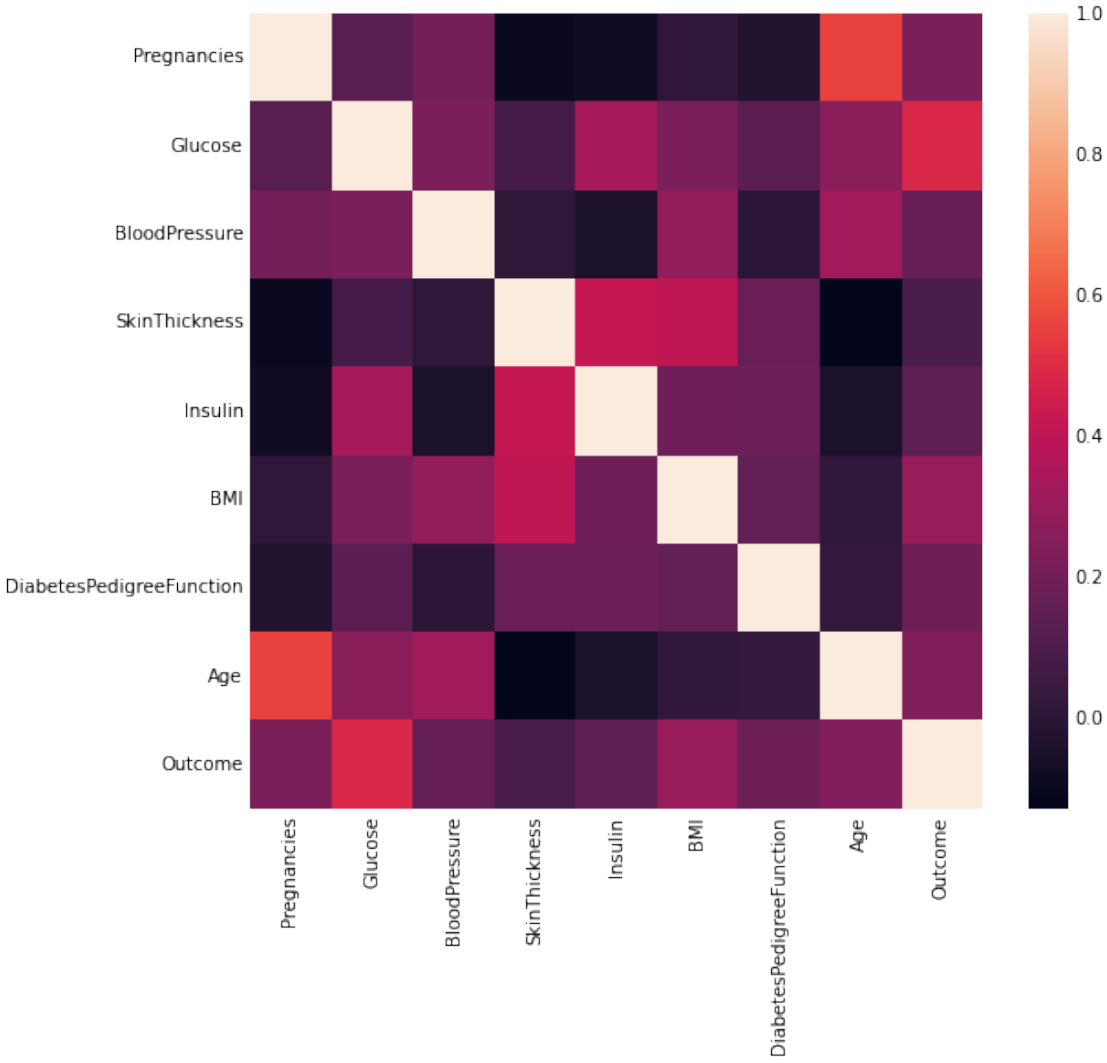
	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.134915	0.209668	-0.095683
Glucose	0.134915	1.000000	0.223331	0.074381
BloodPressure	0.209668	0.223331	1.000000	0.011777
SkinThickness	-0.095683	0.074381	0.011777	1.000000
Insulin	-0.080059	0.337896	-0.046856	0.420874
BMI	0.012342	0.223276	0.287403	0.401528
DiabetesPedigreeFunction	-0.025996	0.136630	-0.000075	0.176253
Age	0.557066	0.263560	0.324897	-0.128908
Outcome	0.224417	0.488384	0.166703	0.092030

	Insulin	BMI	DiabetesPedigreeFunction
Pregnancies	-0.080059	0.012342	-0.025996
Glucose	0.337896	0.223276	0.136630
BloodPressure	-0.046856	0.287403	-0.000075
SkinThickness	0.420874	0.401528	0.176253

```
Insulin                    1.000000  0.191831                 0.182656
BMI                        0.191831  1.000000                 0.154858
DiabetesPedigreeFunction   0.182656  0.154858                 1.000000
Age                       -0.049412  0.020835                 0.023098
Outcome                    0.145488  0.299375                 0.184947

                                Age   Outcome
Pregnancies                0.557066  0.224417
Glucose                    0.263560  0.488384
BloodPressure              0.324897  0.166703
SkinThickness             -0.128908  0.092030
Insulin                   -0.049412  0.145488
BMI                        0.020835  0.299375
DiabetesPedigreeFunction   0.023098  0.184947
Age                        1.000000  0.245741
Outcome                    0.245741  1.000000
```

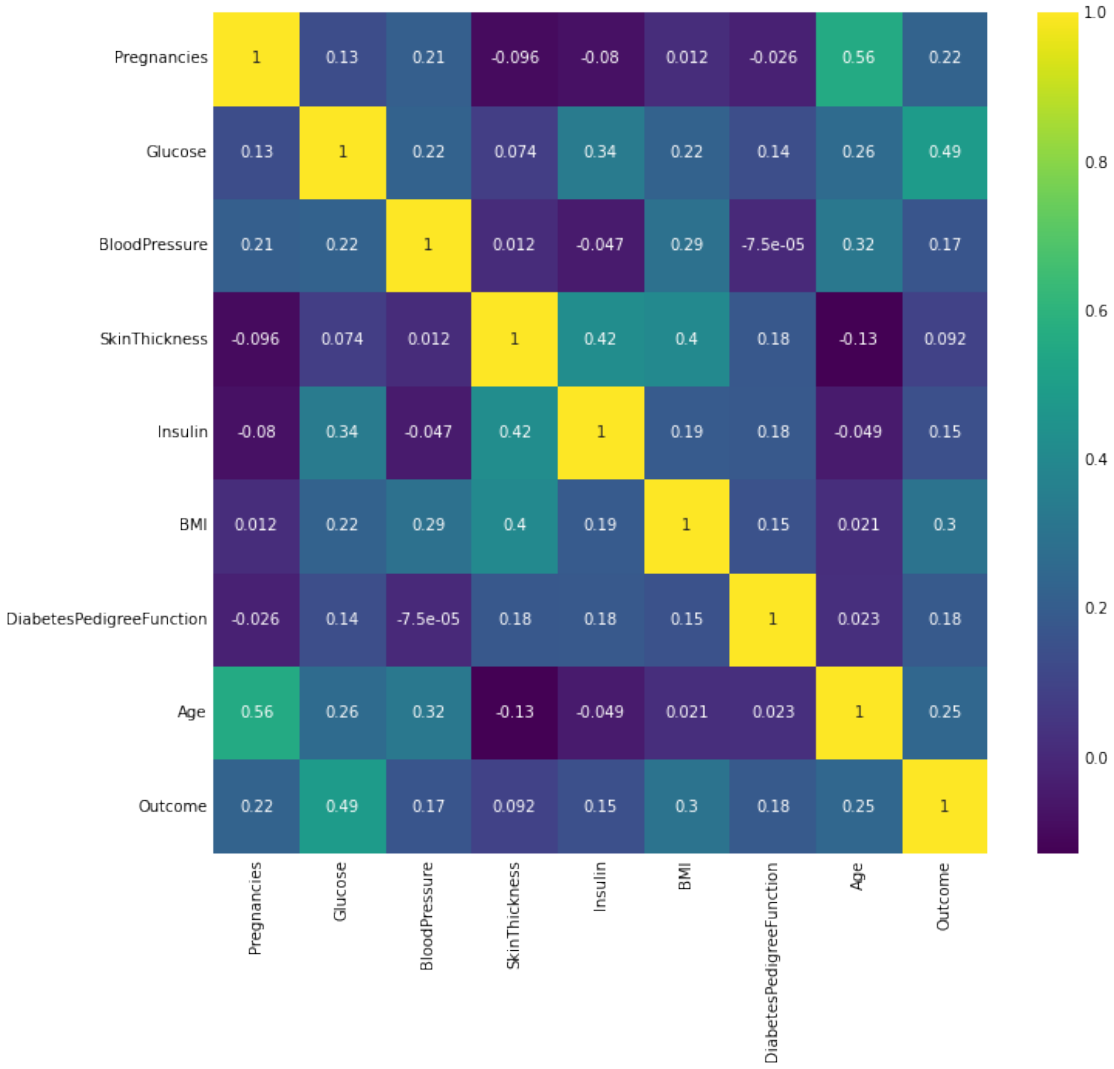
[170]:
```
### create correlation heat map
plt.subplots(figsize=(8,8))
sns.heatmap(diabetes_data_mod.corr())
```

[170]: <AxesSubplot:>



[171]:
```
### gives correlation value
plt.subplots(figsize=(10,10))
sns.heatmap(diabetes_data_mod.corr(),annot=True,cmap='viridis')
```

[171]: <AxesSubplot:>



1.3 Project Task: Week 2

1.3.1 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.
3. Compare various models with the results from KNN algorithm.
4. 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.

Logistic Regreation and model building

[172]:
```
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
X = diabetes_data_mod[feature_names]
y = diabetes_data_mod.Outcome
```

[173]:
```
X.head()
```

[173]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
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

[174]:
```
#Train test split
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 =10)
```

Create Model

[175]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

[176]:
```
#LR Model
model_LR = LogisticRegression(solver='liblinear')
model_LR.fit(X_train,y_train)
```

[176]:
```
LogisticRegression(solver='liblinear')
```

[177]:
```
#now check LR model score and accuracy score

print("LogisticRegression Score :{}".format(model_LR.score(X_train,y_train)))
y_pred = model_LR.predict(X_test)
scores = (accuracy_score(y_test, y_pred))
print("LogisticRegression Accuracy Score :{}".format(scores))
```

```
LogisticRegression Score :0.770293609671848
LogisticRegression Accuracy Score :0.8
```

[178]:
```
accuracyScores = []
modelScores = []
models = []
names = []
#Store algorithm into array to get score and accuracy
models.append(('LR', LogisticRegression(solver='liblinear')))
models.append(('SVC', SVC()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
```

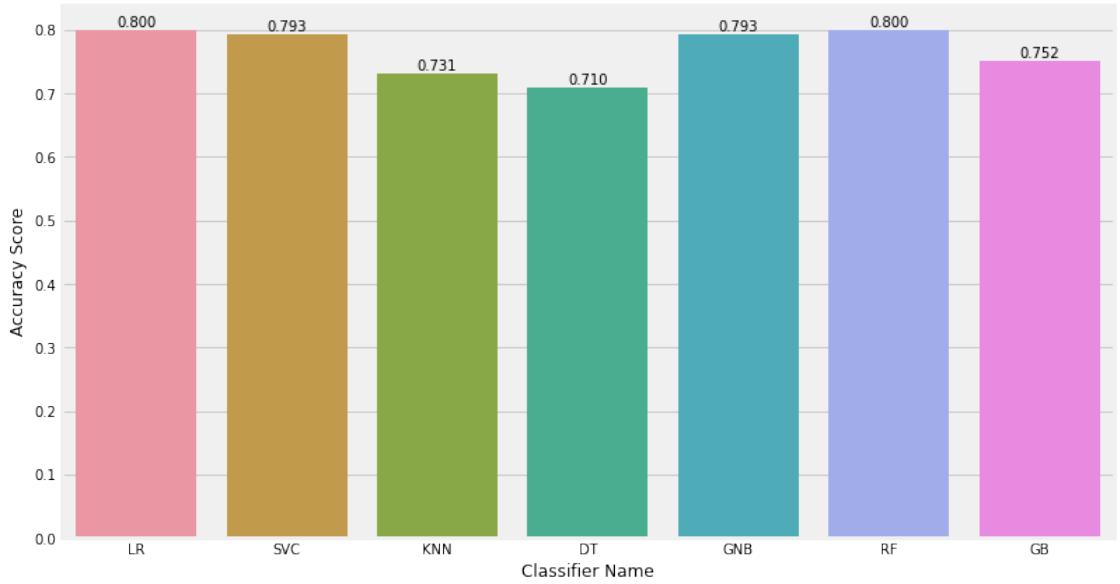
[179]:
```
#We fit each model in a loop and calculate the accuracy of the respective model␣
 ↪using the "accuracy_score"
for name, model in models:
    model.fit(X_train, y_train)
    modelScores.append(model.score(X_train,y_train))
    y_pred = model.predict(X_test)
    accuracyScores.append(accuracy_score(y_test, y_pred))
    names.append(name)

tr_split_data = pd.DataFrame({'Name': names, 'Score': modelScores,'Accuracy␣
 ↪Score': accuracyScores})
print(tr_split_data)
```

```
  Name     Score  Accuracy Score
0   LR  0.770294        0.800000
1  SVC  0.768566        0.793103
2  KNN  0.804836        0.731034
3   DT  1.000000        0.710345
```

```
4  GNB  0.751295        0.793103
5   RF  1.000000        0.800000
6   GB  0.929188        0.751724
```

1.3.2 Lets draw graph to understand more.

[180]:
```
plt.subplots(figsize=(12,7))
axis = sns.barplot(x = 'Name', y = 'Accuracy Score', data = tr_split_data)
axis.set(xlabel='Classifier Name', ylabel='Accuracy Score')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005, '{:1.3f}'.
 ↪format(height), ha="center")

plt.show()
```



Now lets perform K-Fold Cross Validation with Scikit Learn

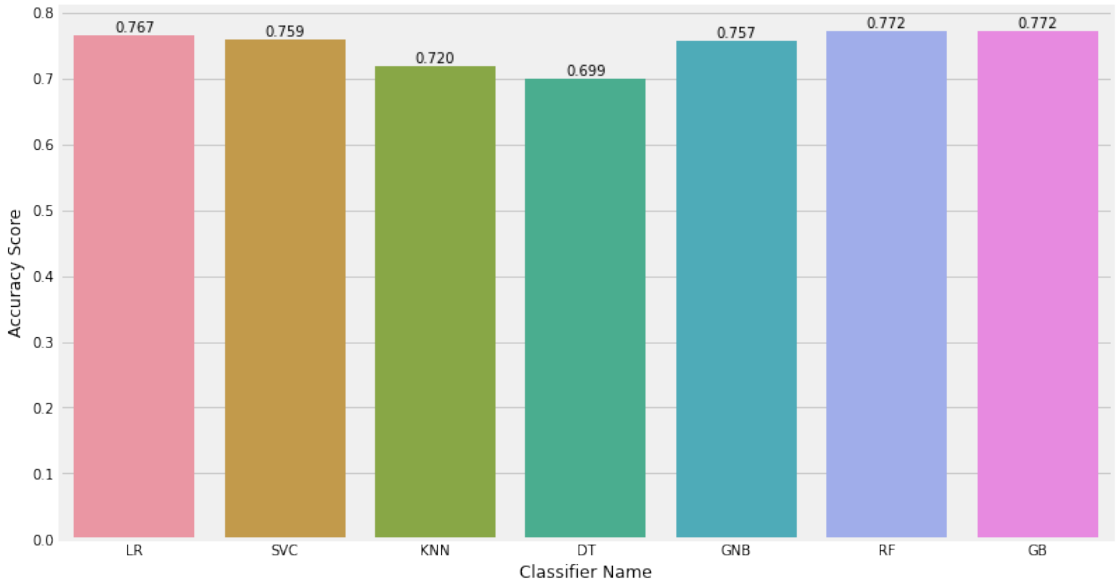
We will move forward with K-Fold cross validation as it is more accurate and use the data efficiently. We will train the models using 10 fold cross validation and calculate the mean accuracy of the models. "k_fold_cross_val_score" provides its own training and accuracy calculation interface.

[181]:
```
names = []
scores = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=None, shuffle=False)
    score = cross_val_score(model, X, y, cv=kfold, scoring='accuracy').mean()
```

```
    names.append(name)
    scores.append(score)
k_fold_cross_val_score = pd.DataFrame({'Name': names, 'Score': scores})
print(k_fold_cross_val_score)
```

```
  Name     Score
0   LR  0.766781
1  SVC  0.758581
2  KNN  0.719787
3   DT  0.699049
4  GNB  0.757021
5   RF  0.772355
6   GB  0.772298
```

[182]:
```
plt.subplots(figsize=(12,7))
axis = sns.barplot(x = 'Name', y = 'Score', data = k_fold_cross_val_score)
axis.set(xlabel='Classifier Name', ylabel='Accuracy Score')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005, '{:1.3f}'.
 ↪format(height), ha="center")

plt.show()
```



Now lets check confussion matric

[183]:
```
#y is label value & X is feature value
cm = confusion_matrix(y,model_LR.predict(X))
```

```
cm
```

[183]:
```
array([[427,  48],
       [114, 135]])
```

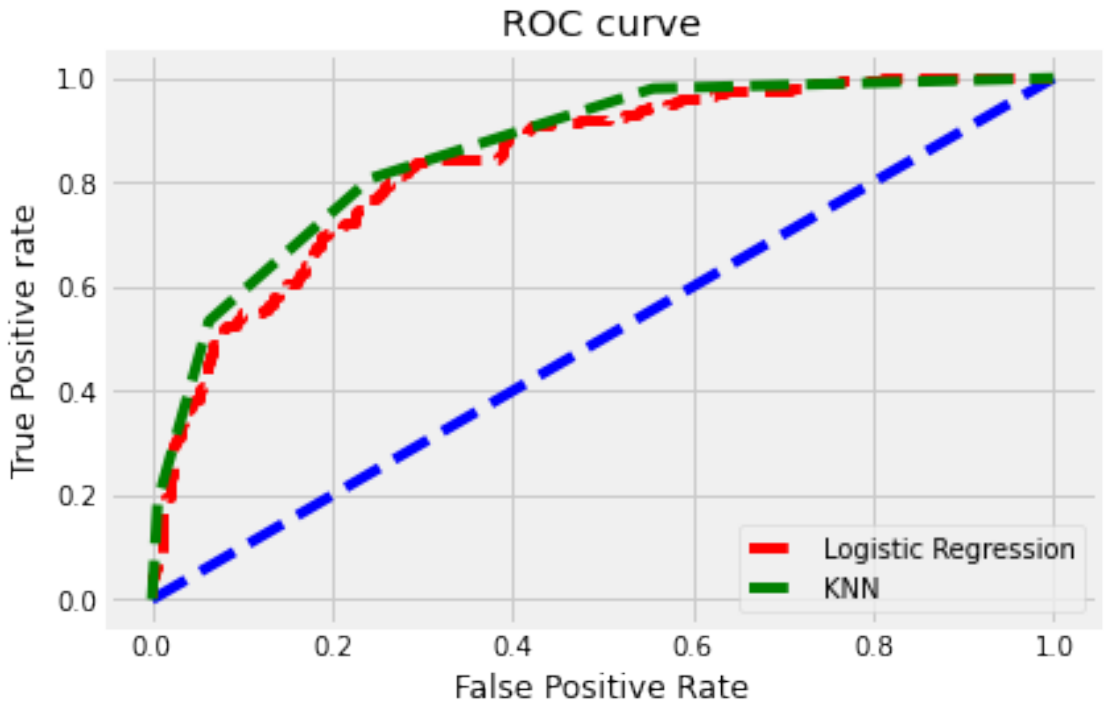
[184]:
```
print(classification_report(y,model_LR.predict(X)))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       475
           1       0.74      0.54      0.62       249

    accuracy                           0.78       724
   macro avg       0.76      0.72      0.73       724
weighted avg       0.77      0.78      0.77       724
```

[185]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

[186]:
```
#Preparing ROC Curve (Receiver Operating Characteristics Curve) - LR, KNN
# predict probabilities for LR
probs_LR = model_LR.predict_proba(X)
# predict probabilities for KNN - where models[2] is KNN
model_KNN = KNeighborsClassifier(n_neighbors=4)
model_KNN.fit(X_train, y_train)
probs_KNN = model_KNN.predict_proba(X)

# Sklearn has a very potent method roc_curve() which computes the ROC for your␣
 ↪classifier in a matter of seconds! It returns the FPR, TPR, and threshold␣
 ↪values: calculate roc curve
fpr, tpr, thresholds = roc_curve(y, probs_LR[:, 1],pos_label=1)
fpr1, tpr1, thresholds1 = roc_curve(y, probs_KNN[:, 1],pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y))]
p_fpr, p_tpr, _ = roc_curve(y, random_probs, pos_label=1)

# plot no skill
plt.plot(p_fpr, p_tpr, linestyle='--',color='blue')
plt.plot(fpr, tpr, linestyle='--',color='red', label='Logistic Regression')
plt.plot(fpr1, tpr1, linestyle='--',color='green', label='KNN')

# plot the roc curve for the model
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
```

```
# y label
plt.ylabel('True Positive rate')
#plt.plot(fpr, tpr, marker='.')
plt.legend(loc='best')
plt.show();
# keep probabilities for the positive outcome only
#The AUC score can be computed using the roc_auc_score() method of sklearn:␣
 ↪calculate AUC
auc_LR = roc_auc_score(y, probs_LR[:, 1])
auc_KNN = roc_auc_score(y, probs_KNN[:, 1])
print('AUC LR: %.5f' % auc_LR, 'AUC KNN: %.5f' % auc_KNN)
```



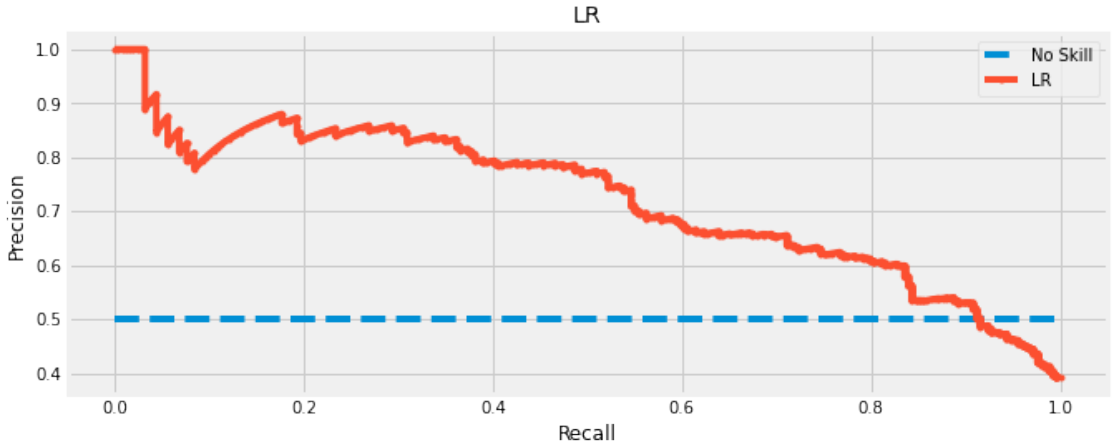

```
AUC LR: 0.83806 AUC KNN: 0.86121
```

[187]:
```
def generate_graph(recall, precision,name):
    # plot no skill
    # plot the precision-recall curve for the model
    plt.figure()
    plt.subplots(figsize=(10,4))
    plt.plot([0, 1], [0.5, 0.5], linestyle='--',label='No Skill')
    plt.plot(recall, precision, marker='.',label=name)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(name)
```

```
    plt.legend(loc='best')
    plt.show()
```

[188]:
```
#Store algorithm into array to get score and accuracy
p_r_Models = []
p_r_Models.append(('LR', LogisticRegression(solver='liblinear')))
p_r_Models.append(('KNN', KNeighborsClassifier()))
p_r_Models.append(('DT', DecisionTreeClassifier()))
p_r_Models.append(('GNB', GaussianNB()))
p_r_Models.append(('RF', RandomForestClassifier()))
p_r_Models.append(('GB', GradientBoostingClassifier()))
#Precision Recall Curve for All classifier
for name, model in p_r_Models:
    from sklearn.metrics import precision_recall_curve
    from sklearn.metrics import f1_score
    from sklearn.metrics import auc
    from sklearn.metrics import average_precision_score
    print("\n==================-------------------- Precision Recall Curve for␣
 ↪{} --------------------==================\n".format(name))
    model.fit(X_train, y_train)
    # predict probabilities
    probs = model.predict_proba(X)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # predict class values
    yhat = model.predict(X)
    # calculate precision-recall curve
    precision, recall, thresholds = precision_recall_curve(y, probs)
    # calculate F1 score, # calculate precision-recall AUC
    f1, auc = f1_score(y, yhat), auc(recall, precision)
    # calculate average precision score
    ap = average_precision_score(y, probs)
    generate_graph(recall, precision,name)
    print(str(name) + " calculated value : " + 'F1 Score =%.3f, Area Under the␣
 ↪Curve=%.3f, Average Precision=%.3f\n' % (f1, auc, ap))
    print("The above precision-recall curve plot is showing the precision/
 ↪recall for each threshold for a {} model (orange) compared to a no skill␣
 ↪model (blue).".format(name))
```

```
==================-------------------- Precision Recall Curve for LR
--------------------==================


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

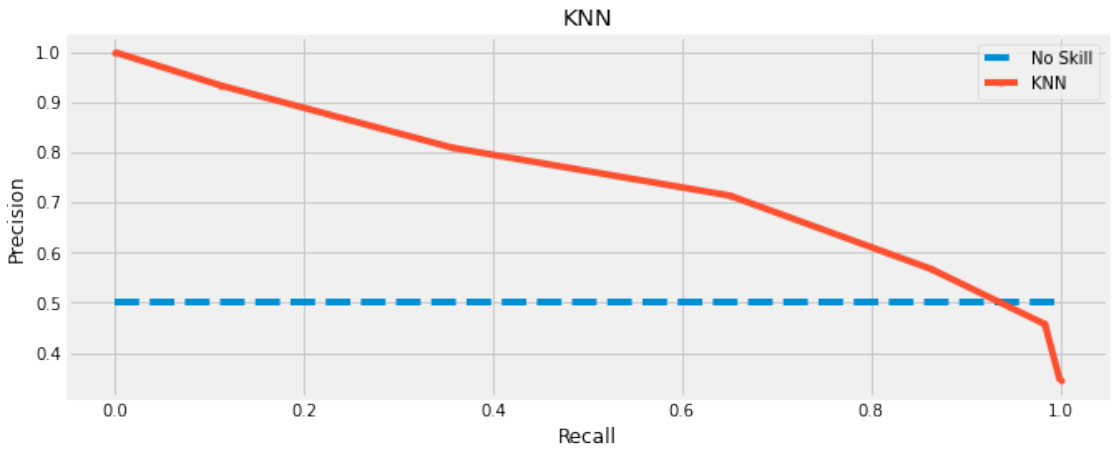


LR calculated value : F1 Score =0.625, Area Under the Curve=0.722, Average Precision=0.723

The above precision-recall curve plot is showing the precision/recall for each threshold for a LR model (orange) compared to a no skill model (blue).

==========---------- Precision Recall Curve for KNN ----------==========

<Figure size 432x288 with 0 Axes>



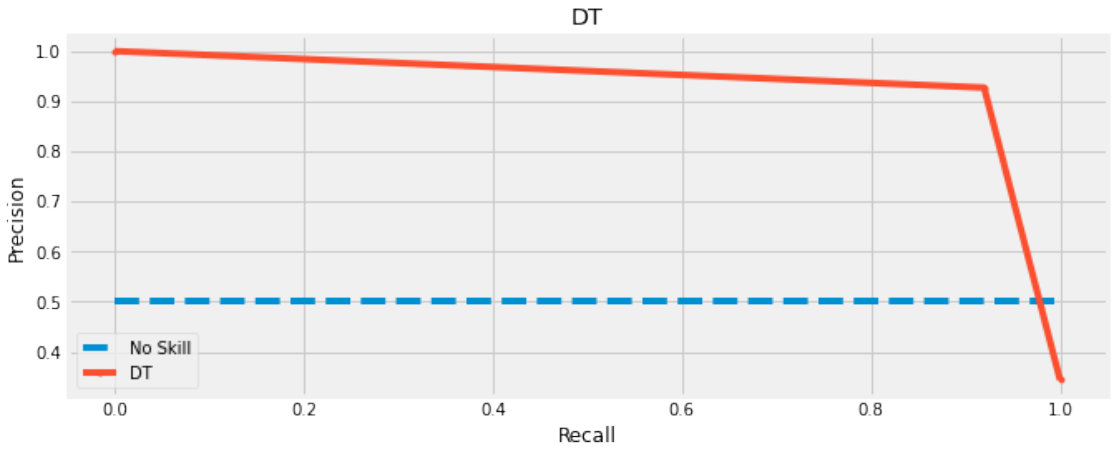
KNN calculated value : F1 Score =0.681, Area Under the Curve=0.750, Average Precision=0.694

The above precision-recall curve plot is showing the precision/recall for each

threshold for a KNN model (orange) compared to a no skill model (blue).

==========---------- Precision Recall Curve for DT
----------==========

<Figure size 432x288 with 0 Axes>

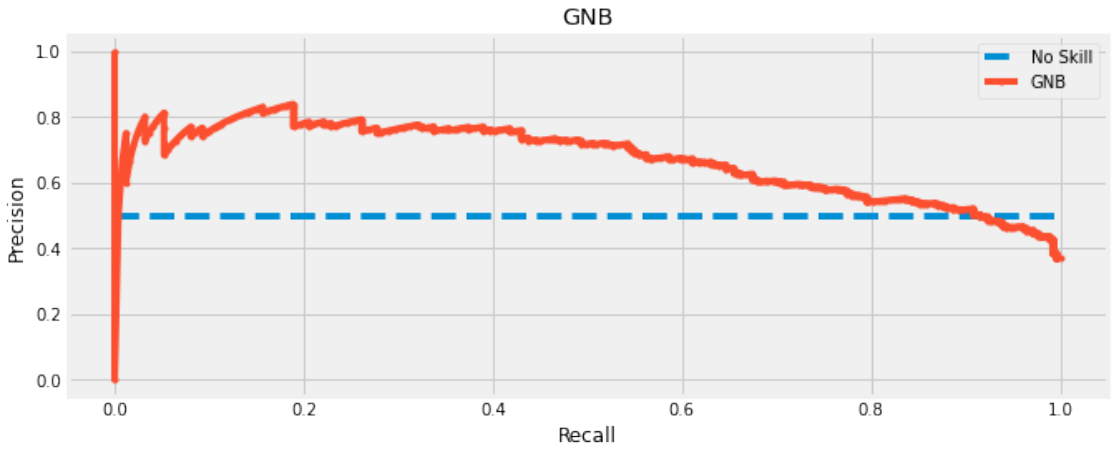


DT calculated value : F1 Score =0.923, Area Under the Curve=0.937, Average Precision=0.880

The above precision-recall curve plot is showing the precision/recall for each threshold for a DT model (orange) compared to a no skill model (blue).

==========---------- Precision Recall Curve for GNB
----------==========

<Figure size 432x288 with 0 Axes>

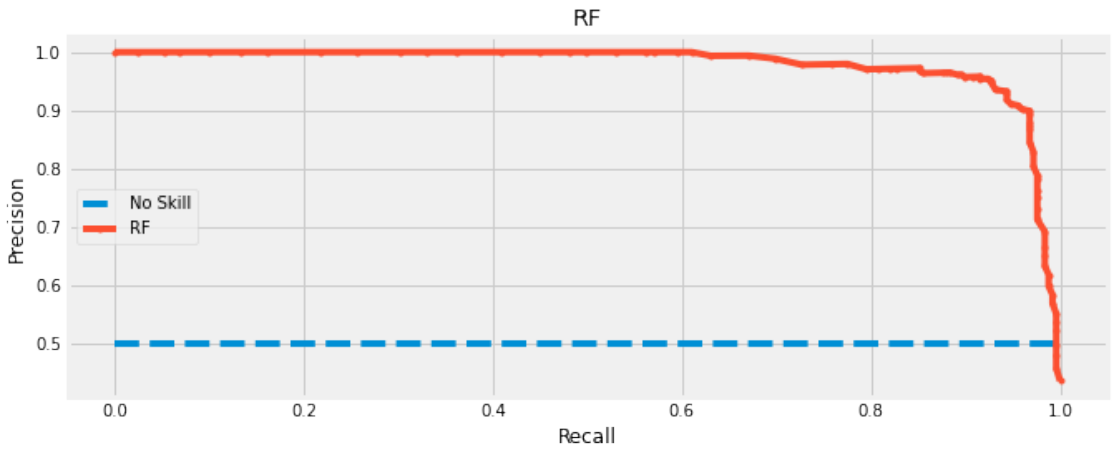


GNB calculated value : F1 Score =0.637, Area Under the Curve=0.671, Average Precision=0.674

The above precision-recall curve plot is showing the precision/recall for each threshold for a GNB model (orange) compared to a no skill model (blue).

==========---------- Precision Recall Curve for RF ----------==========

<Figure size 432x288 with 0 Axes>



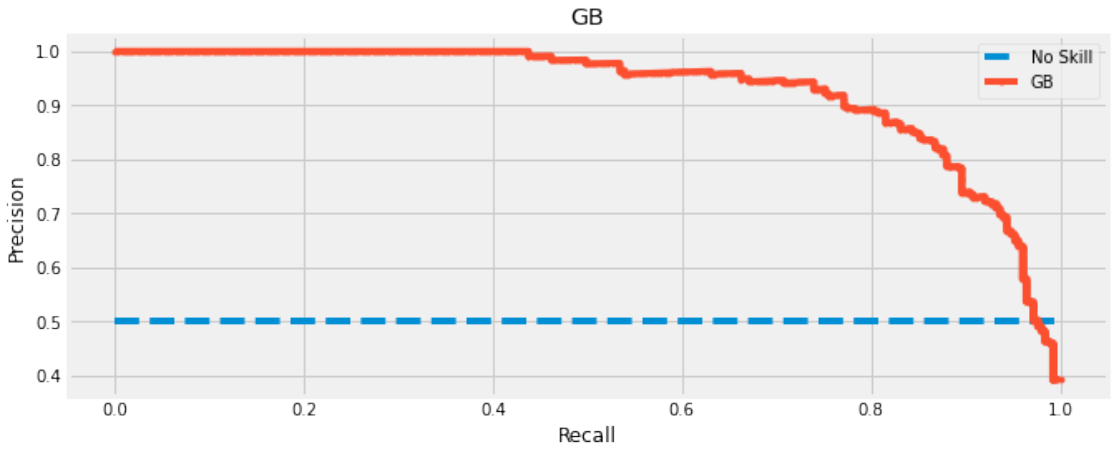
RF calculated value : F1 Score =0.928, Area Under the Curve=0.979, Average Precision=0.978

The above precision-recall curve plot is showing the precision/recall for each

threshold for a RF model (orange) compared to a no skill model (blue).

==========---------- Precision Recall Curve for GB
----------==========

<Figure size 432x288 with 0 Axes>



GB calculated value : F1 Score =0.832, Area Under the Curve=0.929, Average Precision=0.929

The above precision-recall curve plot is showing the precision/recall for each threshold for a GB model (orange) compared to a no skill model (blue).

1.3.3 Data Reporting:

5. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

   - Pie chart to describe the diabetic or non-diabetic population

   - Scatter charts between relevant variables to analyze the relationships

   - Histogram or frequency charts to analyze the distribution of the data

   - Heatmap of correlation analysis among the relevant variables

   - Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables f

Tableau Dashboard Link - LINK

https://public.tableau.com/views/HealthcarePGP_16722699126210/HealthcarePGPDashboard?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link

[ ]: