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

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:
```
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()
```

In [3]:
```
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())
```

In [4]:
```
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()
```

In [5]:
```
diabetes_data = pd.read_csv('health care diabetes.csv')
```

In [6]:
```
diabetes_data.head()
```

Out[6]:

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

In [7]:
```
diabetes_data.groupby('Outcome').size()
```

Out[7]:
```
Outcome
0    500
1    268
dtype: int64
```

In [8]:
```
diabetes_data.isnull().sum()
```

Out[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [9]:
```
diabetes_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
```

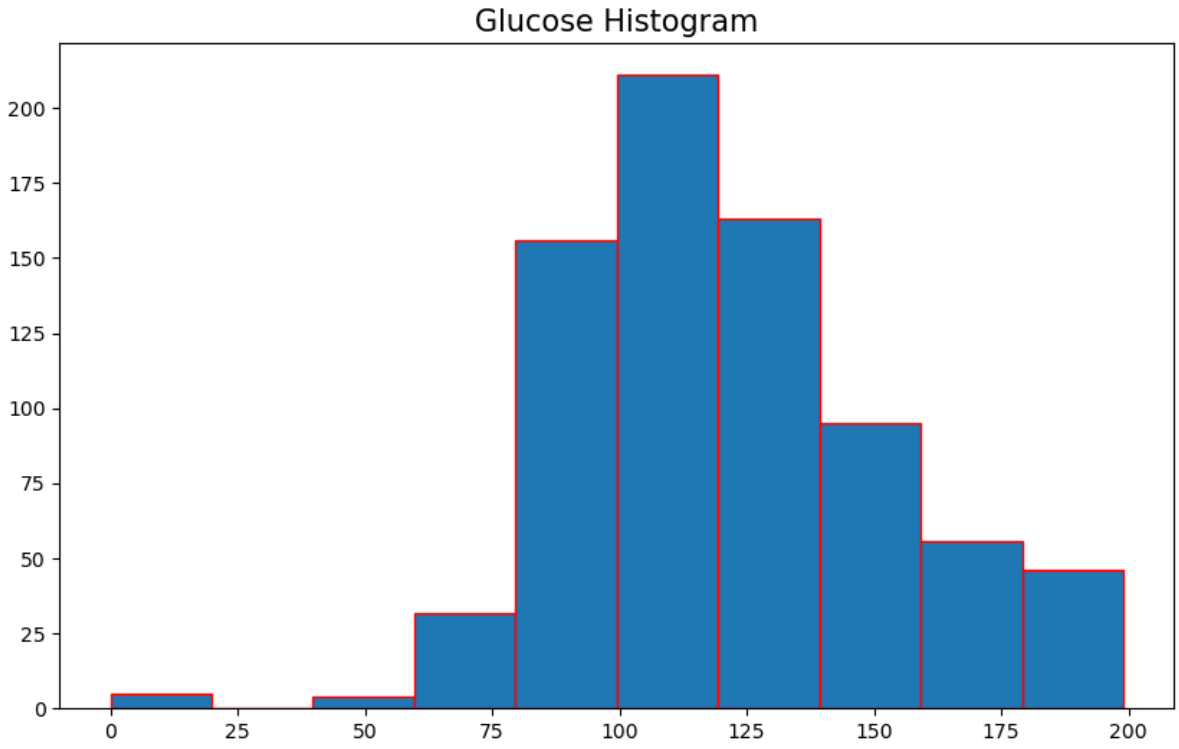
In [10]:
```
diabetes_data['Glucose'].value_counts().head(10)
```

Out[10]:
```
Glucose
99     17
100    17
111    14
129    14
125    14
106    14
112    13
108    13
95     13
105    13
Name: count, dtype: int64
```

In [11]:
```
diabetes_data['Glucose']
```

Out[11]:
```
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
```

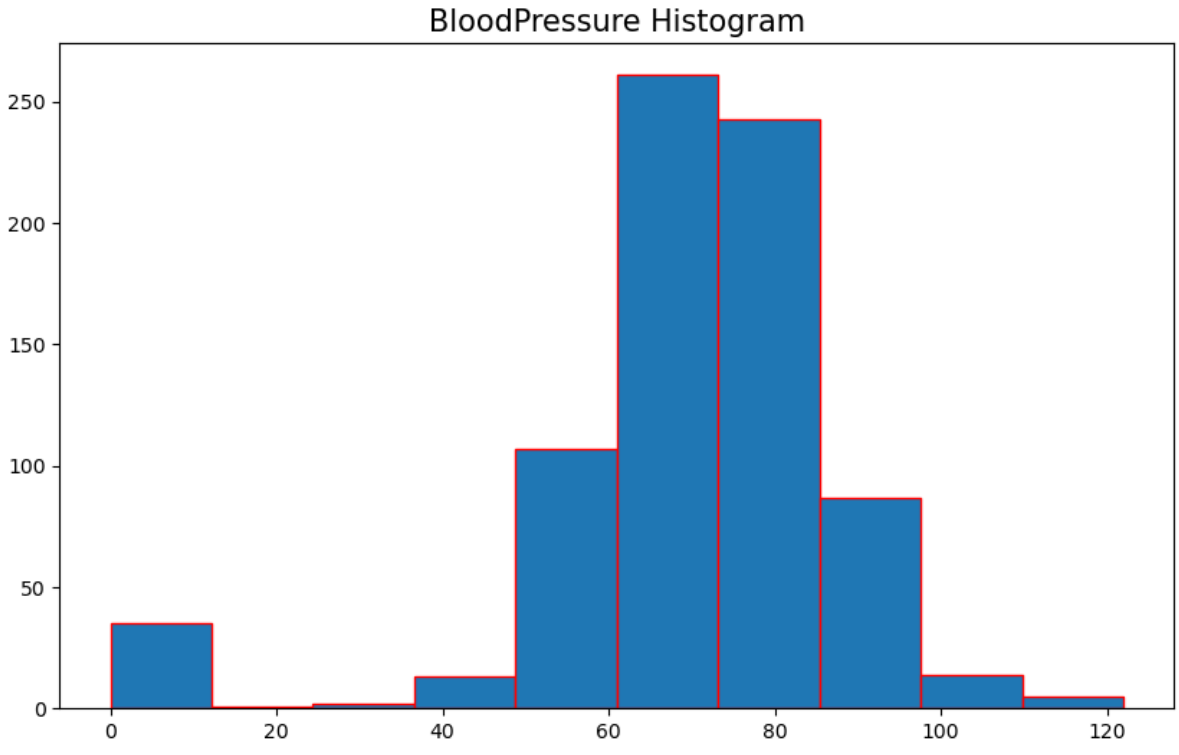
In [12]:
```
plot_histogram(diabetes_data['Glucose'],'Glucose Histogram')
```



In [13]:
```
diabetes_data['BloodPressure'].value_counts().head(7)
```

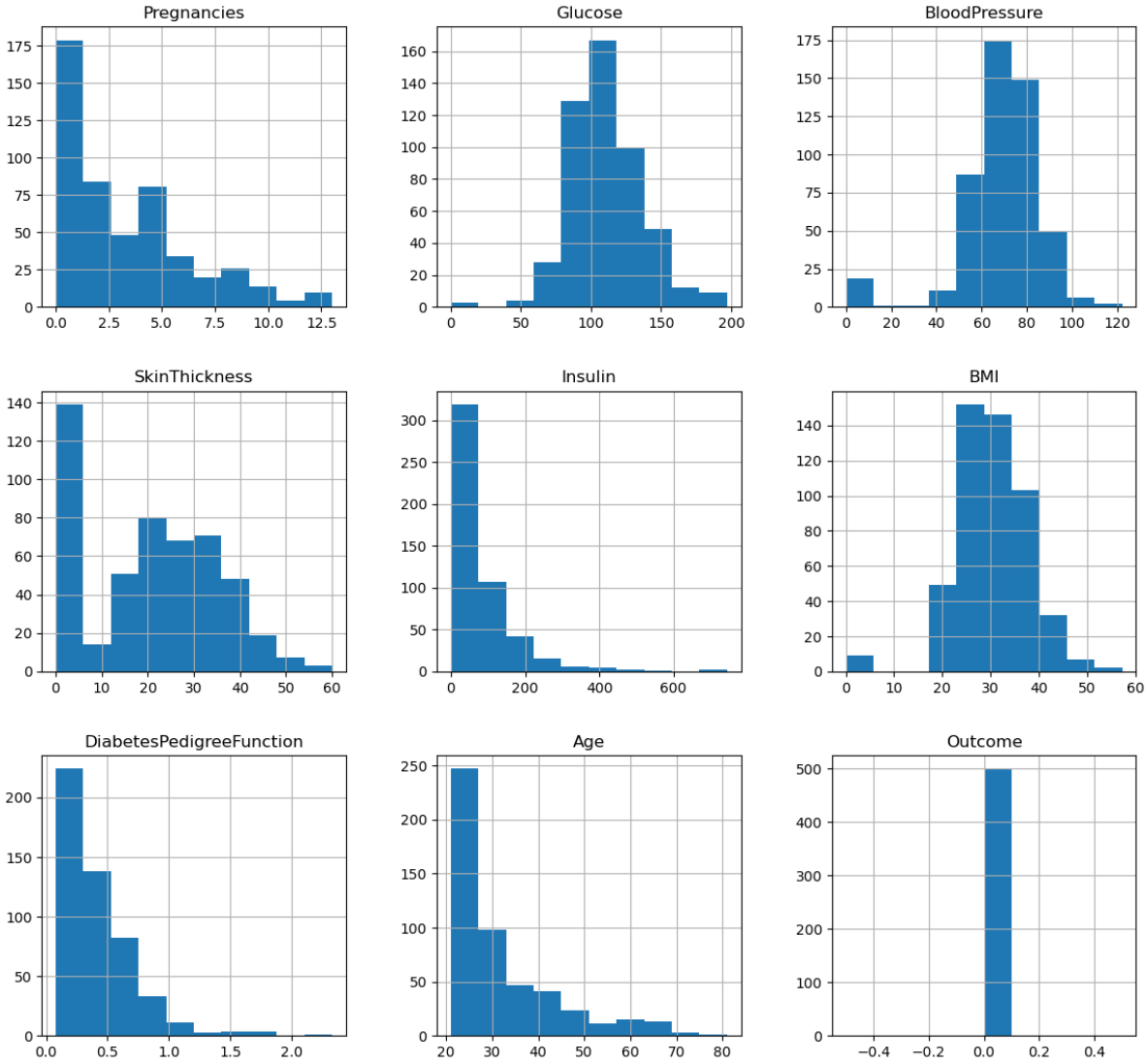
Out[13]:
```
BloodPressure
70    57
74    52
78    45
68    45
72    44
64    43
80    40
Name: count, dtype: int64
```

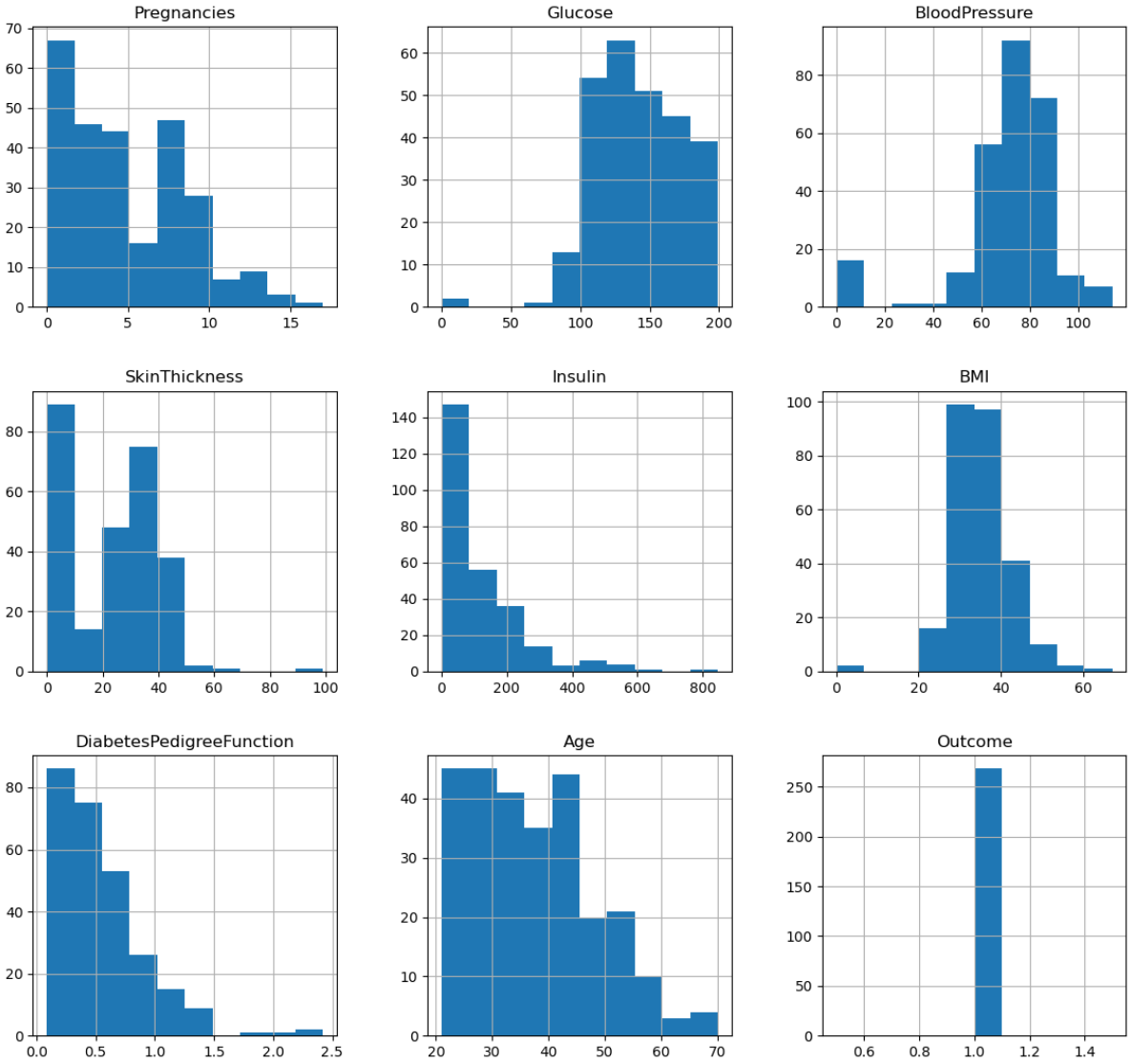
In [14]:
```
plot_histogram(diabetes_data['BloodPressure'],'BloodPressure Histogram')
```



In [15]:
```
diabetes_data.groupby('Outcome').hist(figsize=(14, 13))
```

Out[15]:
```
Outcome
0    [[Axes(0.125,0.666111;0.215278x0.213889), Axes...
1    [[Axes(0.125,0.666111;0.215278x0.213889), Axes...
dtype: object
```





In [16]:
```
get_zeros_outcome_count(diabetes_data,'BloodPressure')
```

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

In [17]:
```
get_zeros_outcome_count(diabetes_data,'Glucose')
```

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

In [19]:
```
get_zeros_outcome_count(diabetes_data,'SkinThickness')
```

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

In [20]:
```
get_zeros_outcome_count(diabetes_data,'BMI')
```

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

In [21]:
```
get_zeros_outcome_count(diabetes_data,'Insulin')
```

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

In [22]:
```
diabetes_data_mod = diabetes_data[(diabetes_data.BloodPressure != 0) & (diabetes_da
print(diabetes_data_mod.shape)
```

```
(724, 9)
```

In [23]:
```
diabetes_data_mod.describe().transpose()
```

Out[23]:

	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

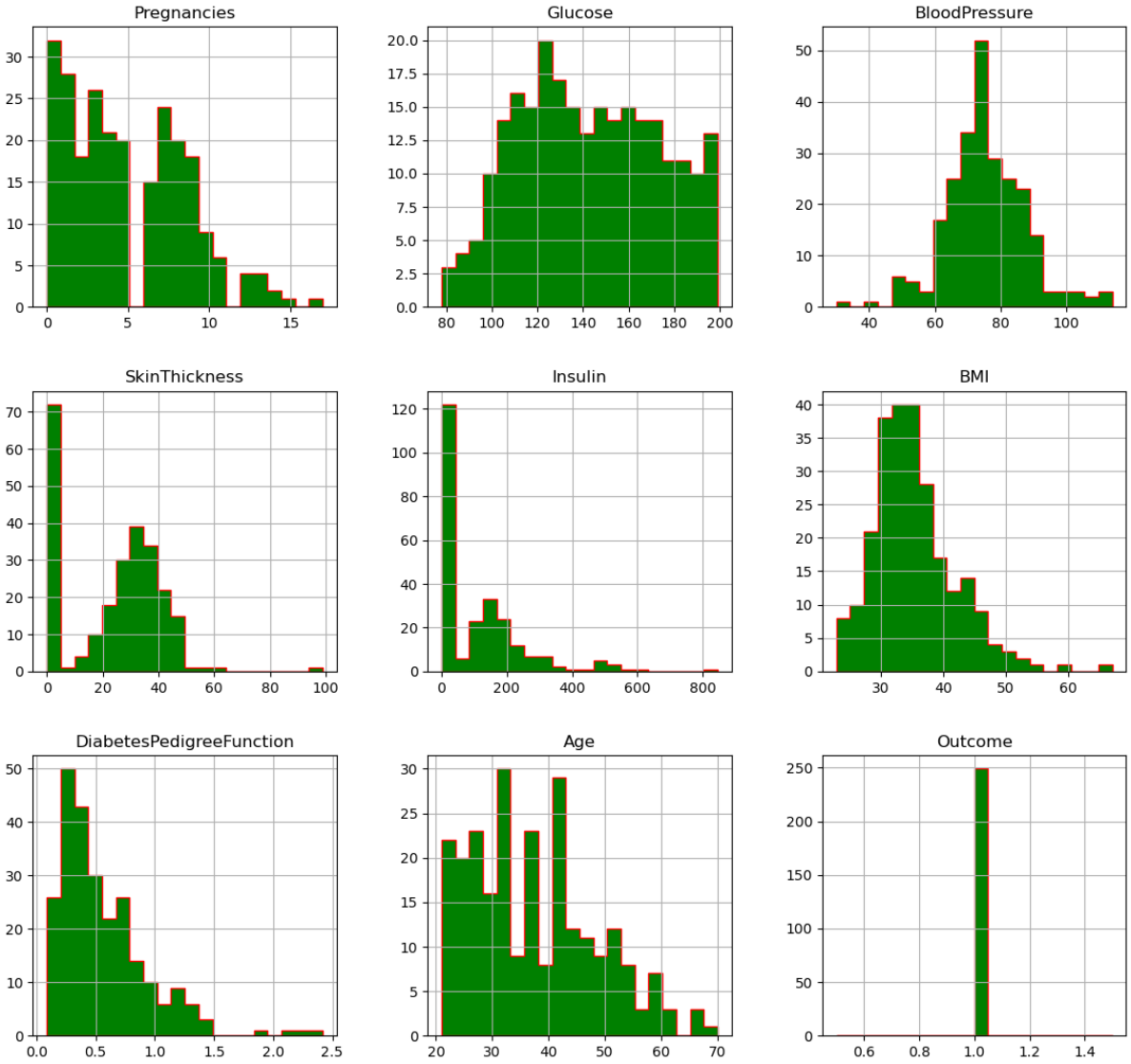
In [24]:
```
Positive = diabetes_data_mod[diabetes_data_mod['Outcome']==1]
Positive.head(5)
```

Out[24]:

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

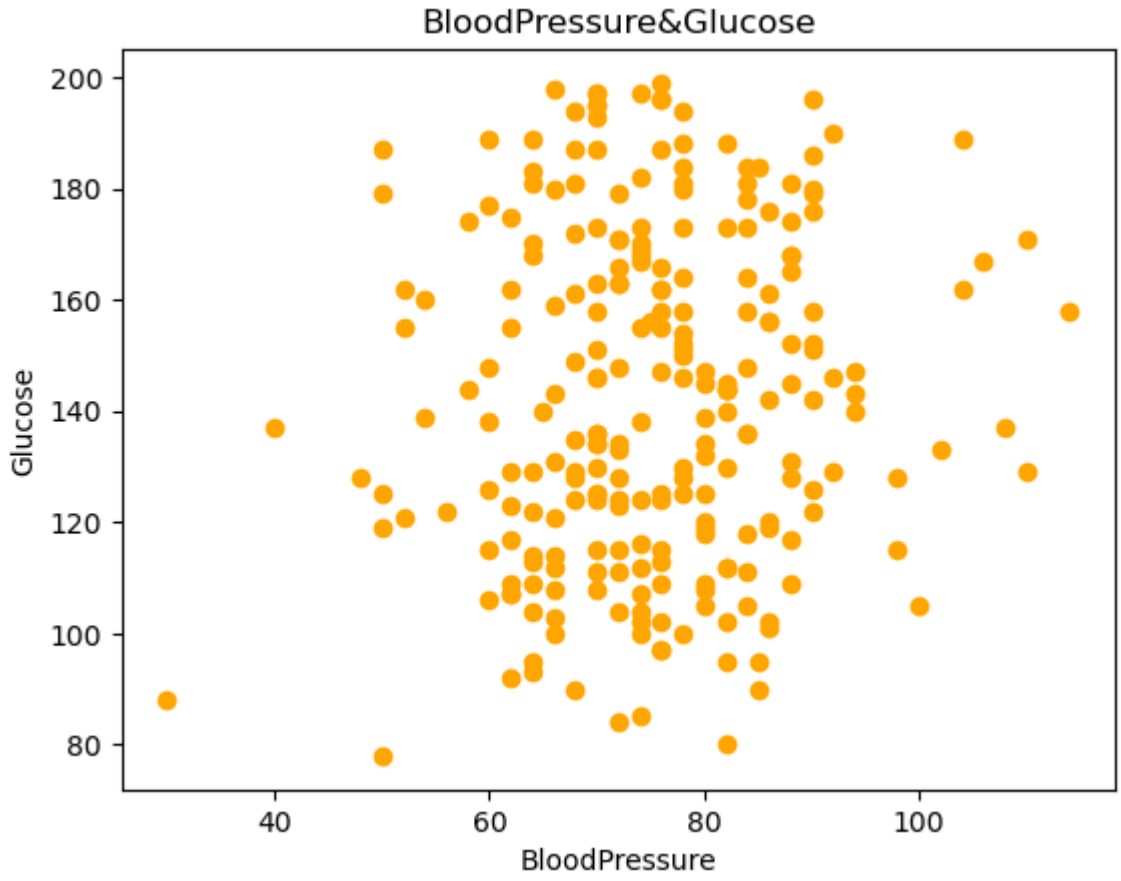
In [25]:
```
Positive.groupby('Outcome').hist(figsize=(14, 13),histtype='stepfilled',bins=20,col
```

Out[25]:
```
Outcome
1    [[Axes(0.125,0.666111;0.215278x0.213889), Axes...
dtype: object
```



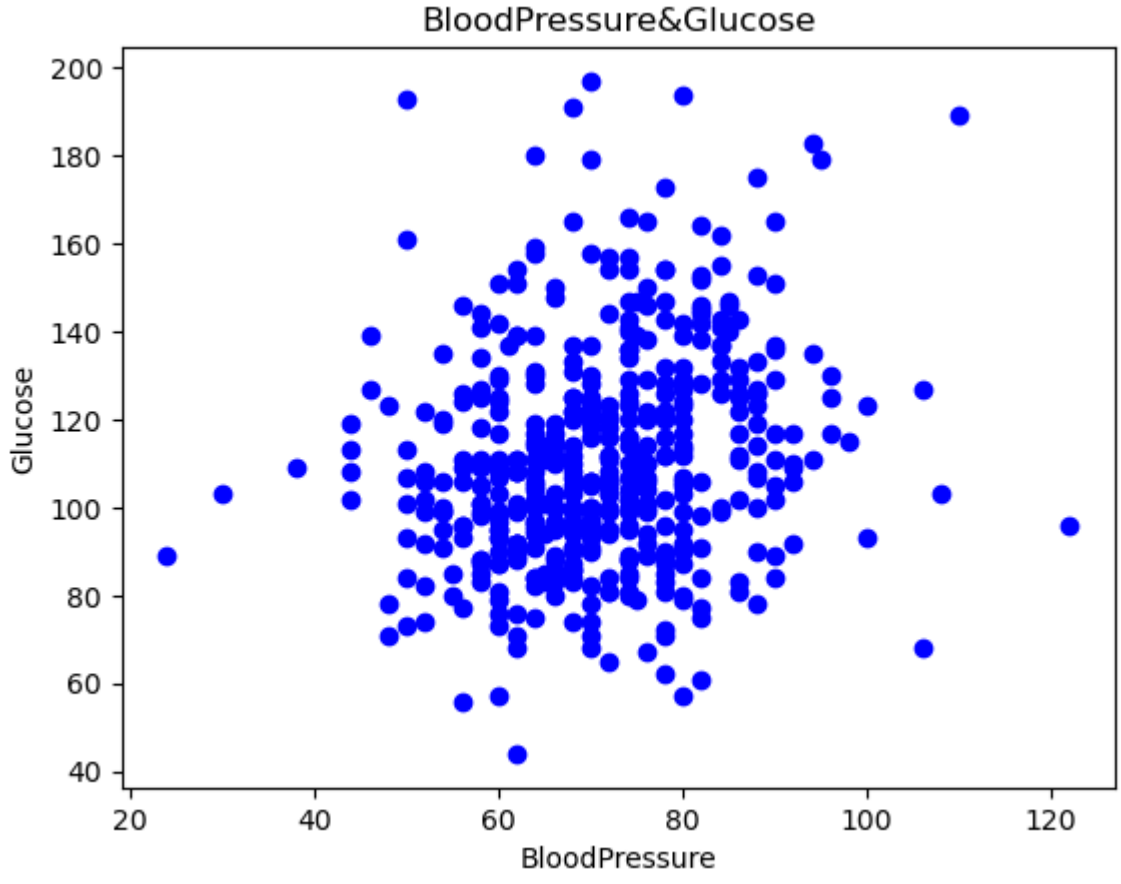
In [26]:
```
BloodPressure = Positive['BloodPressure']
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
Insulin = Positive['Insulin']
BMI = Positive['BMI']
```

In [27]:
```
create_scatter_plot(Positive['BloodPressure'],Positive['Glucose'],'BloodPressure','
```



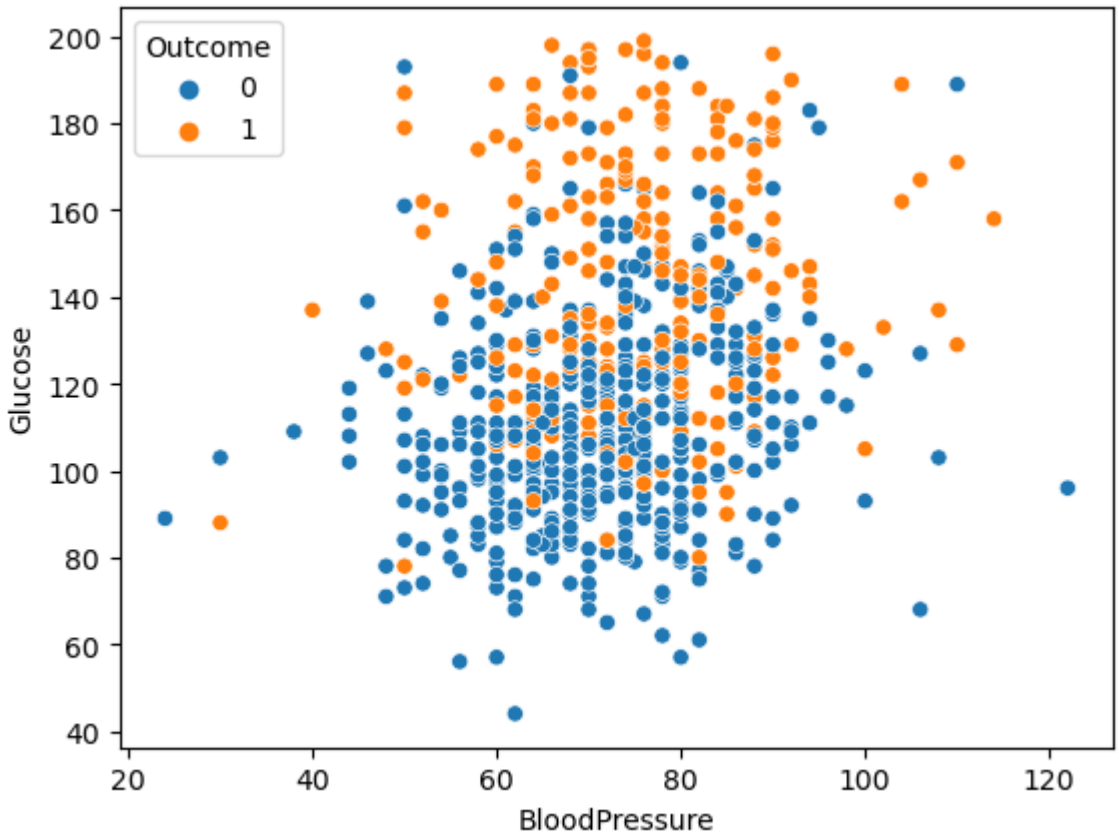
In [28]:
```
Negative = diabetes_data_mod[diabetes_data_mod['Outcome']==0]
```

In [29]:
```
create_scatter_plot(Negative['BloodPressure'],Negative['Glucose'],'BloodPressure','
```

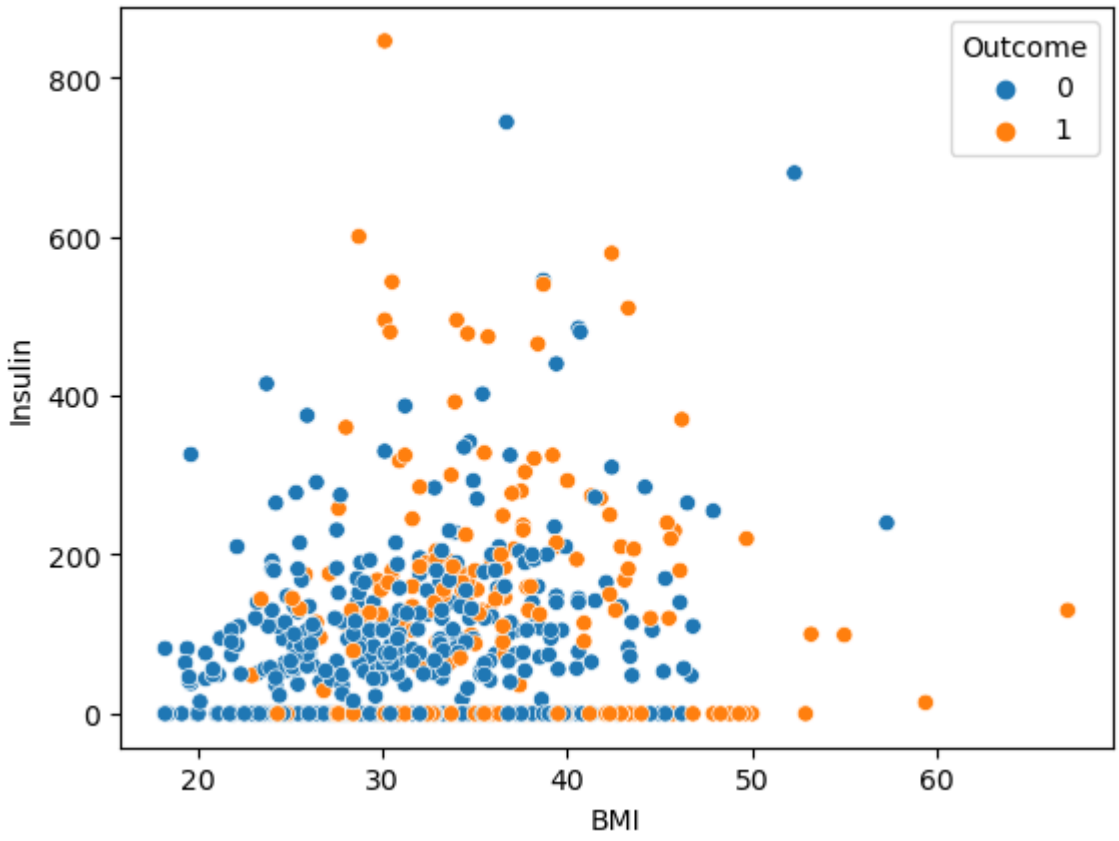


In [30]:
```
g =sns.scatterplot(x= "BloodPressure" ,y= "Glucose",
                   hue="Outcome",
```

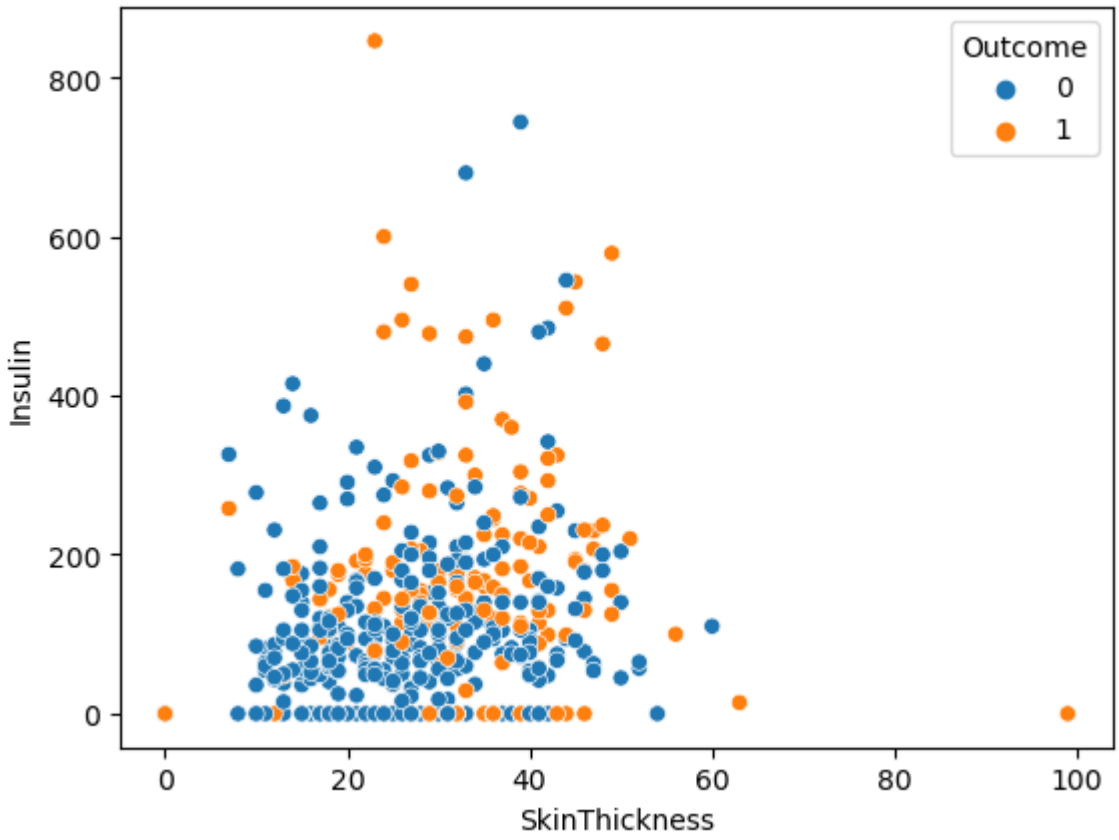
```
                data=diabetes_data_mod);
```



In [31]:
```
B =sns.scatterplot(x= "BMI" ,y= "Insulin",
                hue="Outcome",
                data=diabetes_data_mod);
```



In [32]:
```
S =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",
                hue="Outcome",
                data=diabetes_data_mod);
```



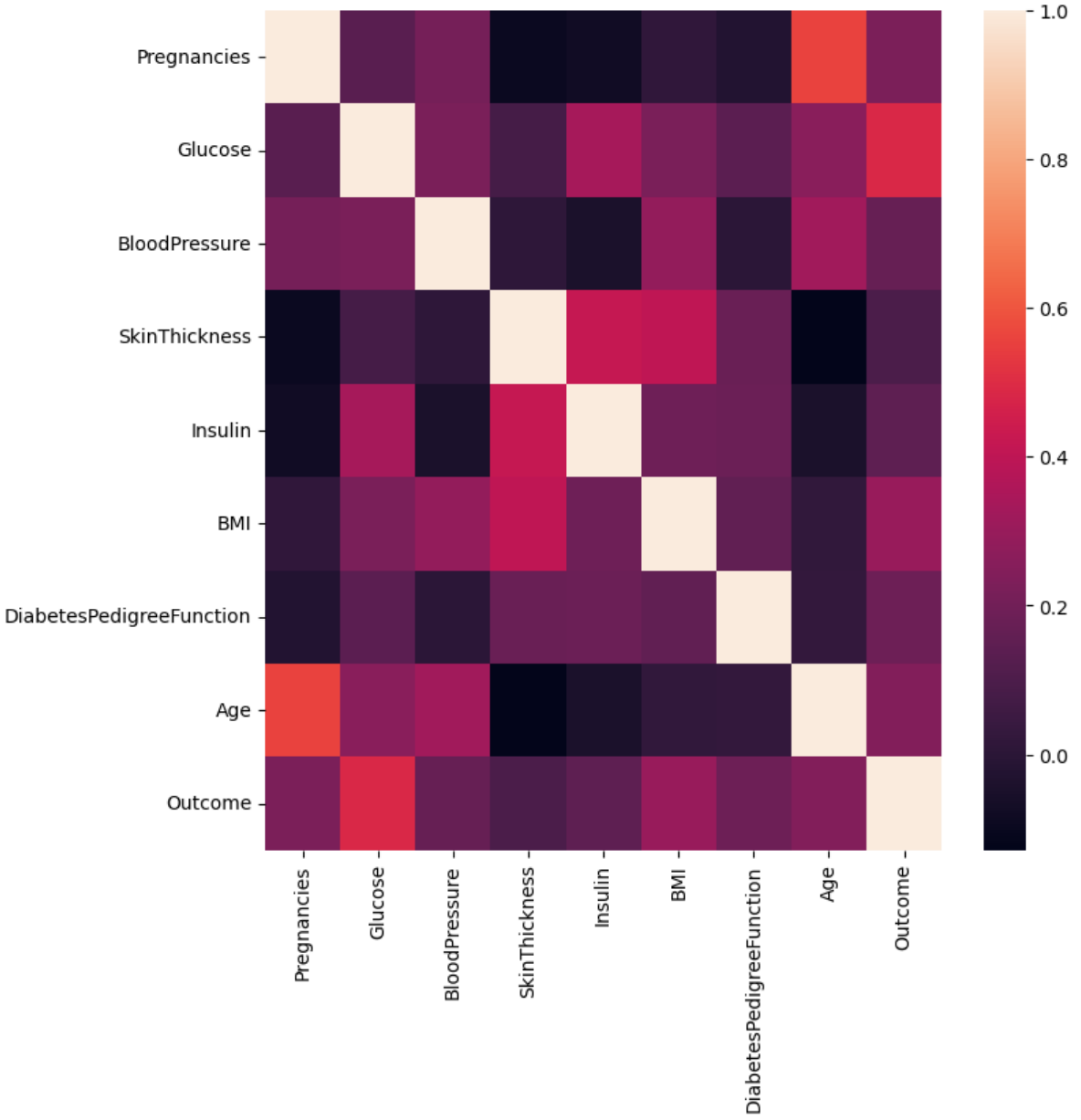
In [33]:
```
diabetes_data_mod.corr()
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BM
Pregnancies	1.000000	0.134915	0.209668	-0.095683	-0.080059	0.012342
Glucose	0.134915	1.000000	0.223331	0.074381	0.337896	0.223276
BloodPressure	0.209668	0.223331	1.000000	0.011777	-0.046856	0.287403
SkinThickness	-0.095683	0.074381	0.011777	1.000000	0.420874	0.401528
Insulin	-0.080059	0.337896	-0.046856	0.420874	1.000000	0.191831
BMI	0.012342	0.223276	0.287403	0.401528	0.191831	1.000000
DiabetesPedigreeFunction	-0.025996	0.136630	-0.000075	0.176253	0.182656	0.154858
Age	0.557066	0.263560	0.324897	-0.128908	-0.049412	0.020835
Outcome	0.224417	0.488384	0.166703	0.092030	0.145488	0.299375

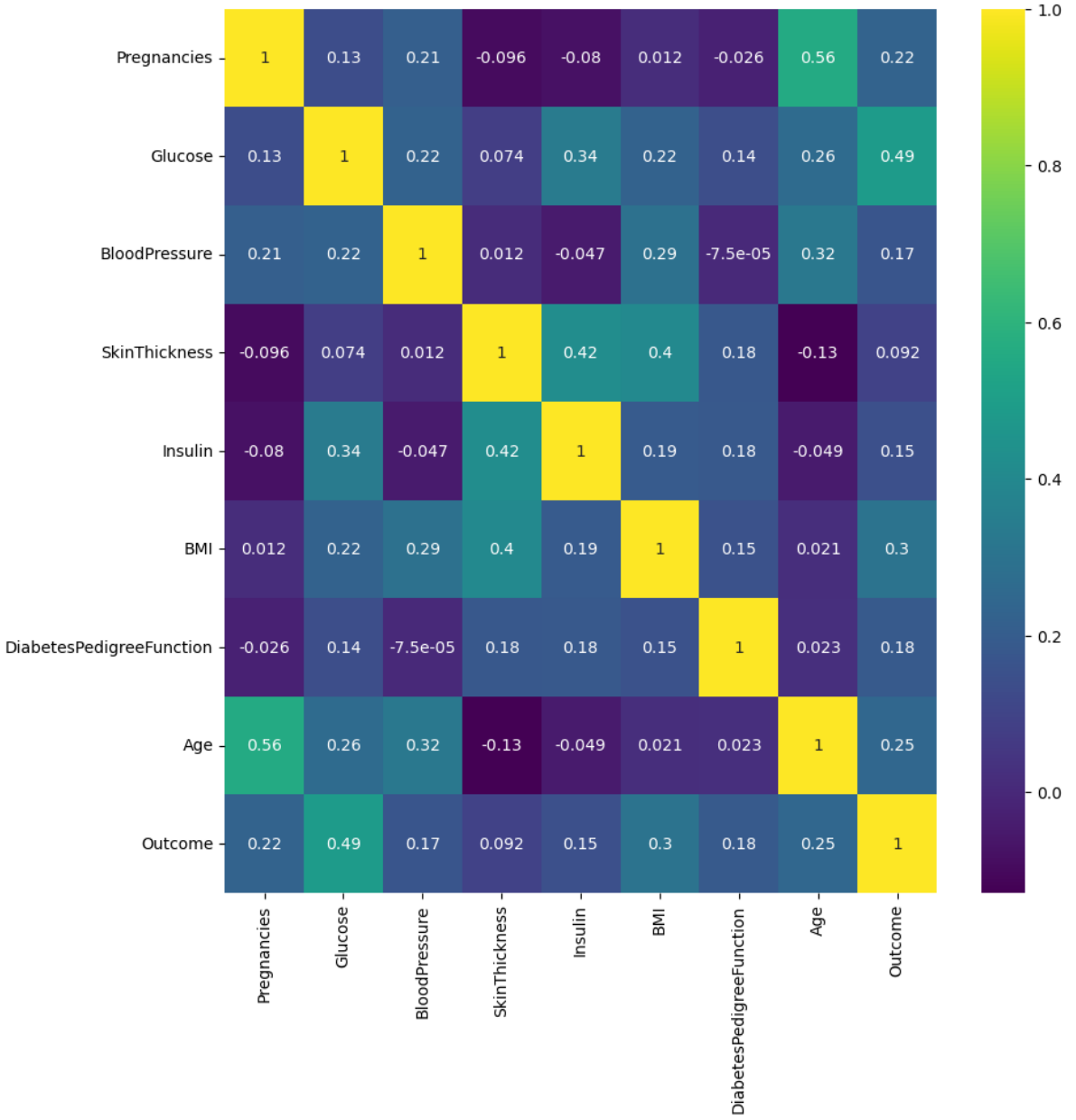
In [34]:
```
plt.subplots(figsize=(8,8))
sns.heatmap(diabetes_data_mod.corr())
```

Out[34]:
```
<Axes: >
```



In [35]:
```
plt.subplots(figsize=(10,10))
sns.heatmap(diabetes_data_mod.corr(),annot=True,cmap='viridis')
```

Out[35]:
```
<Axes: >
```



In [36]:
```
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insul
X = diabetes_data_mod[feature_names]
y = diabetes_data_mod.Outcome
```

In [37]:
```
X.head()
```

Out[37]:

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

In [38]:
```
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 =
```

In [39]:
```
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)
```

In [40]:
```
model_LR = LogisticRegression(solver='liblinear')
model_LR.fit(X_train,y_train)
```

Out[40]:
```
LogisticRegression(solver='liblinear')
```

In [41]:
```
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
```

In [42]:
```
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()))
```

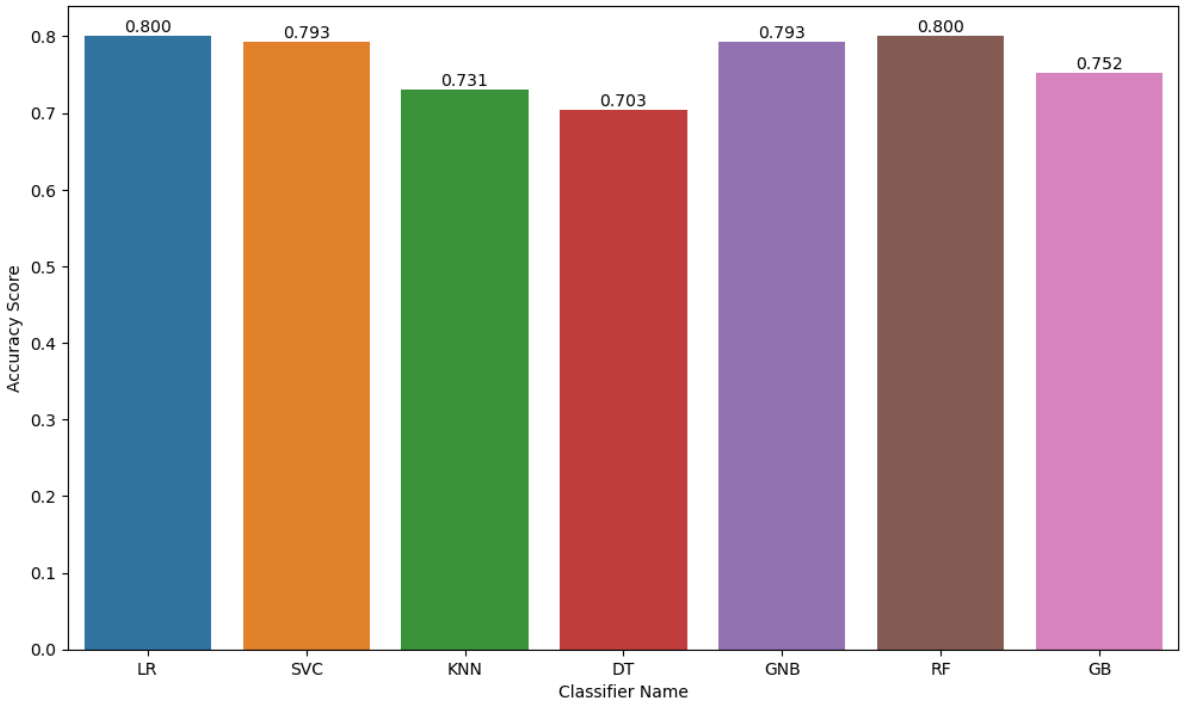
In [43]:
```
for name, model in models:
    model.fit(X_train, y_train)
    modelScores.append(model.score(X_train,y_train))
    y_pred = model.predict(X_test)
    accuracyScores.append(accuracy_score(y_test, y_pred))
    names.append(name)

tr_split_data = pd.DataFrame({'Name': names, 'Score': modelScores,'Accuracy Score':
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	0.703448
4	GNB	0.751295	0.793103
5	RF	1.000000	0.800000
6	GB	0.929188	0.751724

In [44]:
```
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)

plt.show()
```



In [53]:
```
cm = confusion_matrix(y,model_LR.predict(X))
cm
```

Out[53]:
```
array([[427,  48],
       [114, 135]], dtype=int64)
```

In [54]:
```
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
```

In [52]:
```
cm = confusion_matrix(y,model_LR.predict(X))
cm
```

Out[52]:
```
array([[427,  48],
       [114, 135]], dtype=int64)
```

In [55]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

In [56]:
```
#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)
```

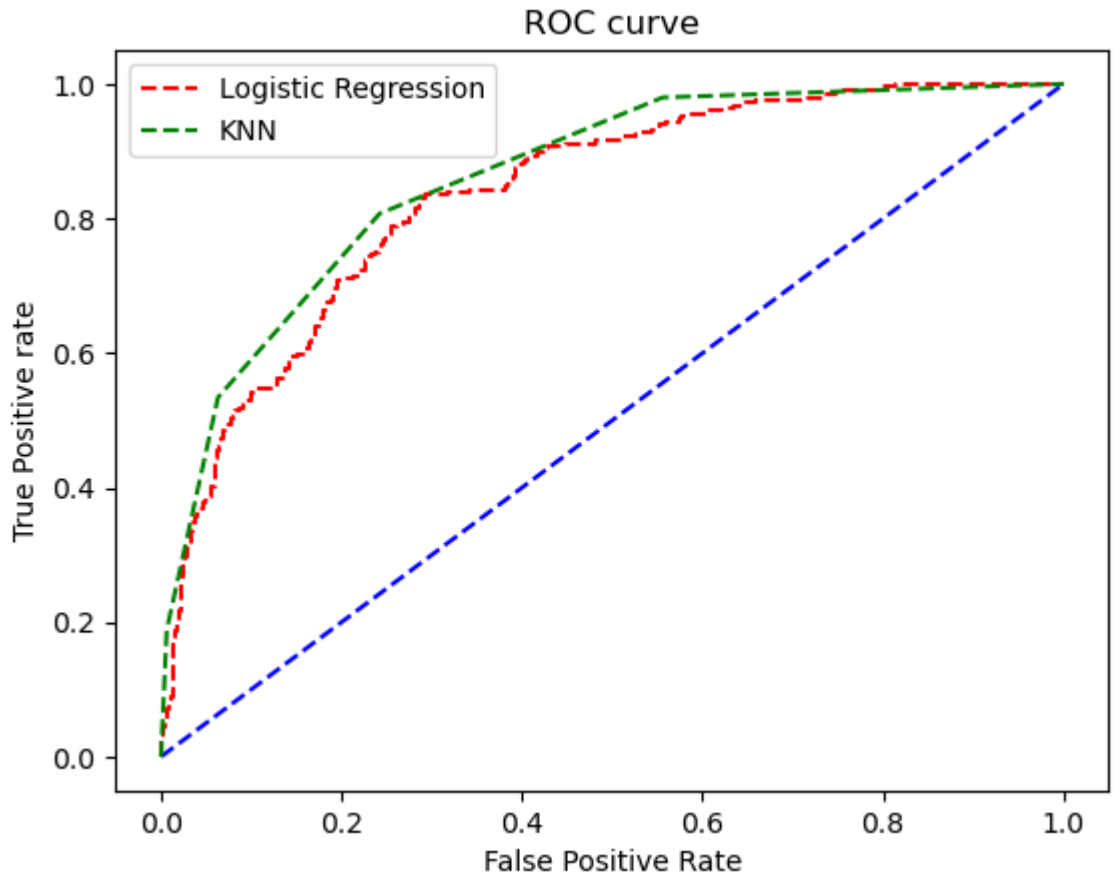
```
model_KNN.fit(X_train, y_train)
probs_KNN = model_KNN.predict_proba(X)

# Sklearn has a very potent method roc_curve() which computes the ROC for your clas
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: calcula
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: 0.83722 AUC KNN: 0.86121
```

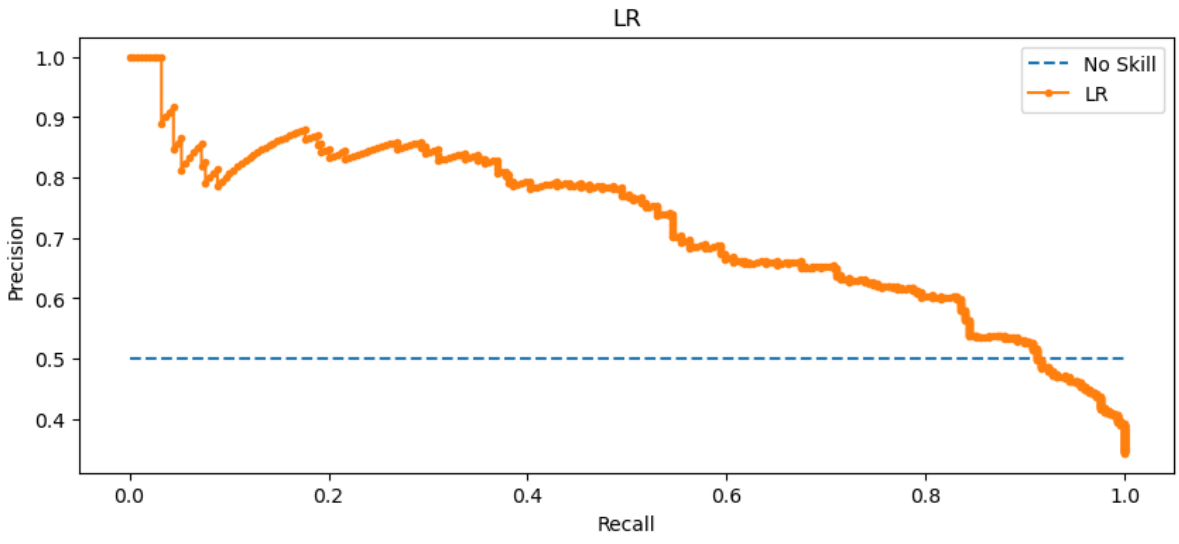
In [57]:
```
def generate_graph(recall, precision,name):
    # plot no skill
    # plot the precision-recall curve for the model
```

```
    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()
```

In [58]:
```
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 {} -
    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 Curv
    print("The above precision-recall curve plot is showing the precision/recall fc
```

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

```
<Figure size 640x480 with 0 Axes>
```

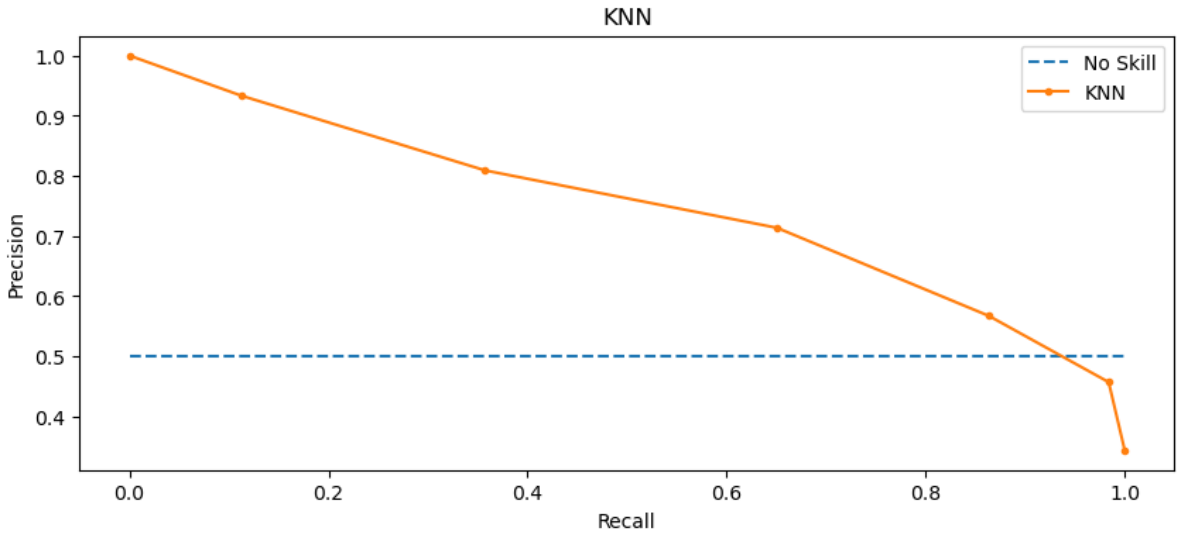


```
LR calculated value : F1 Score =0.625, Area Under the Curve=0.721, Average Precisi
on=0.723

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

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

<Figure size 640x480 with 0 Axes>
```

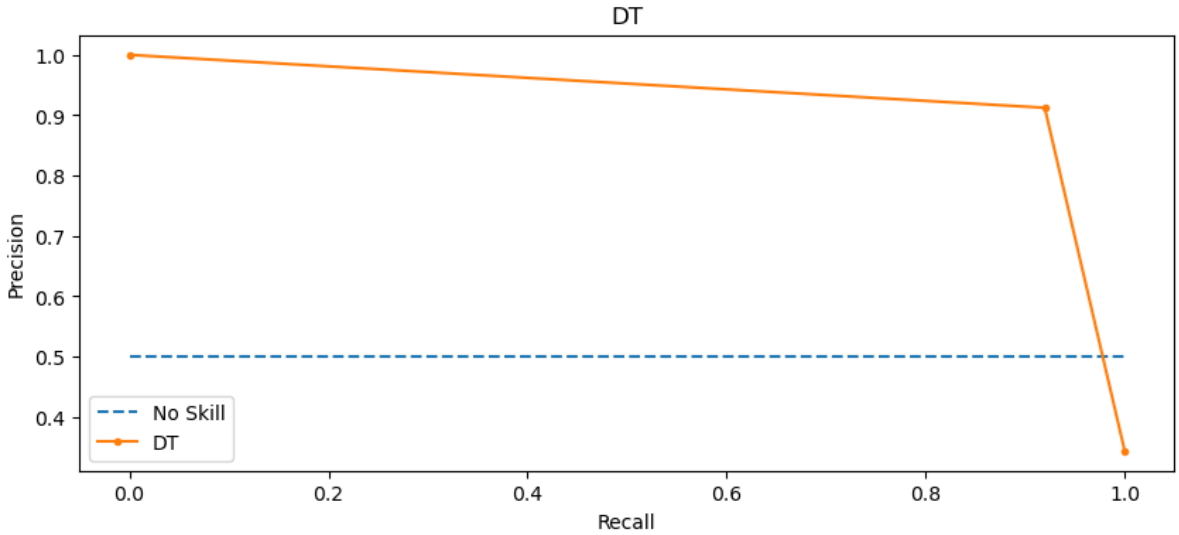



```
KNN calculated value : F1 Score =0.681, Area Under the Curve=0.750, Average Precis
ion=0.694

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

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

<Figure size 640x480 with 0 Axes>
```

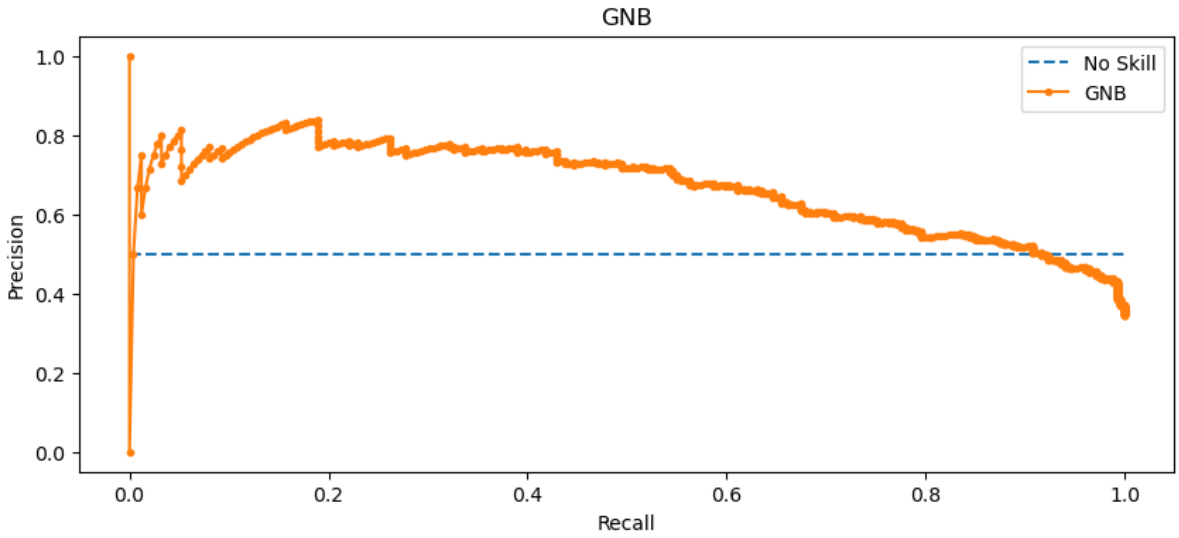



```
DT calculated value : F1 Score =0.916, Area Under the Curve=0.930, Average Precisi
on=0.867

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

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

<Figure size 640x480 with 0 Axes>
```

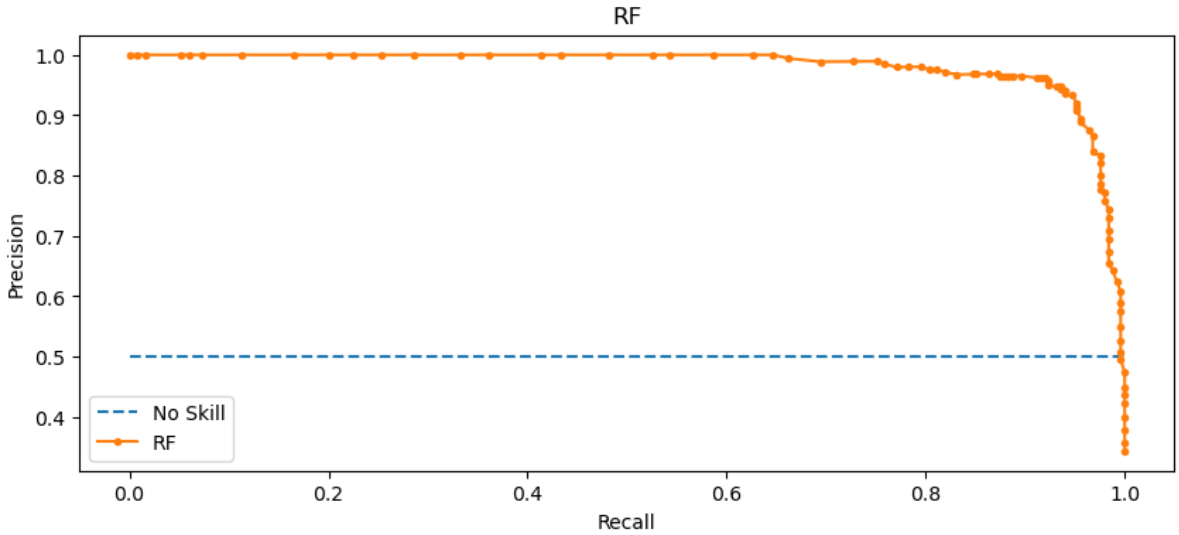


GNB calculated value : F1 Score =0.637, Area Under the Curve=0.671, Average Precis
ion=0.674

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

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

<Figure size 640x480 with 0 Axes>

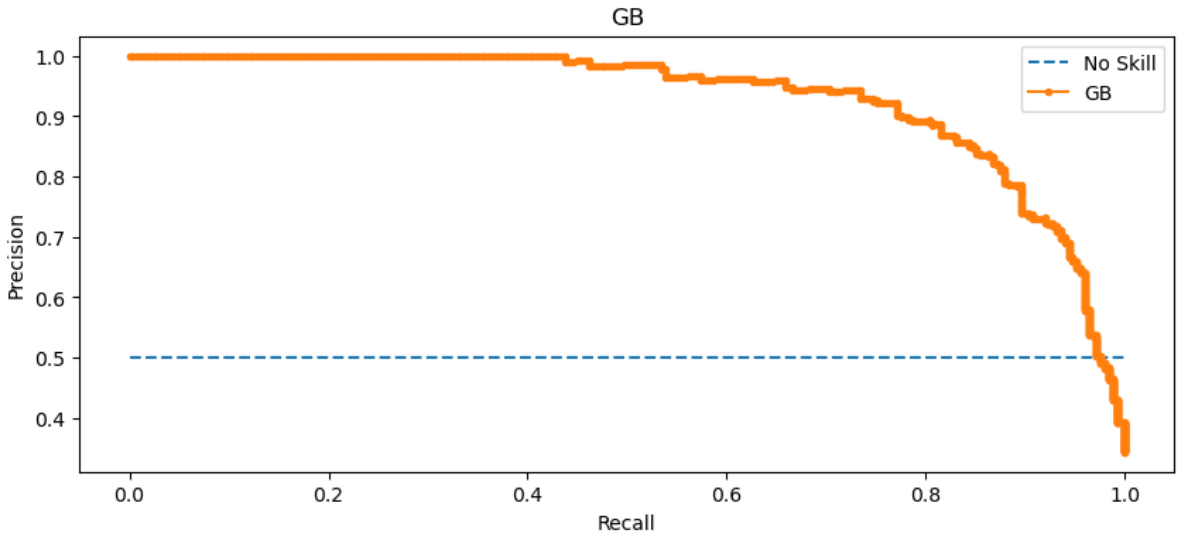


RF calculated value : F1 Score =0.938, Area Under the Curve=0.981, Average Precisi
on=0.980

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

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

<Figure size 640x480 with 0 Axes>




```
GB calculated value : F1 Score =0.834, Area Under the Curve=0.929, Average Precisi
on=0.929
```

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

In [62]:

In [ ]: