Capstone Project-

Healthcare

Problem Statement-

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.

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

In [2]:
```
#import dataset
hc=pd.read_csv('health care diabetes.csv')
```

In [3]:
```
hc.head()
```

Out[3]:

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

In [4]:
```
hc.shape
```

Out[4]:
```
(768, 9)
```

In [5]:
```
hc.isnull().sum()
```

Out[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
```

```
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [6]:
```
hc.dtypes
```

Out[6]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

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.

In [7]:
```
hc.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [8]:
```
# finding zero values in all columns
for x in hc.columns[1:-1]:
    l=len(hc[hc[x]==0])
    if l>1:
        print(x,'  has {} zeros'.format(l))
    else:
        print(x,'  has no zero values')
```

```
Glucose   has 5 zeros
BloodPressure   has 35 zeros
SkinThickness   has 227 zeros
Insulin   has 374 zeros
BMI   has 11 zeros
DiabetesPedigreeFunction   has no zero values
Age   has no zero values
```

A person cannot have zero values for glucose,bmi,blood pressure,insulin,skin thickness,diabetes pedigree function.all these zero values doesn't make sense so these are missing values so we have to treat them accordingly.

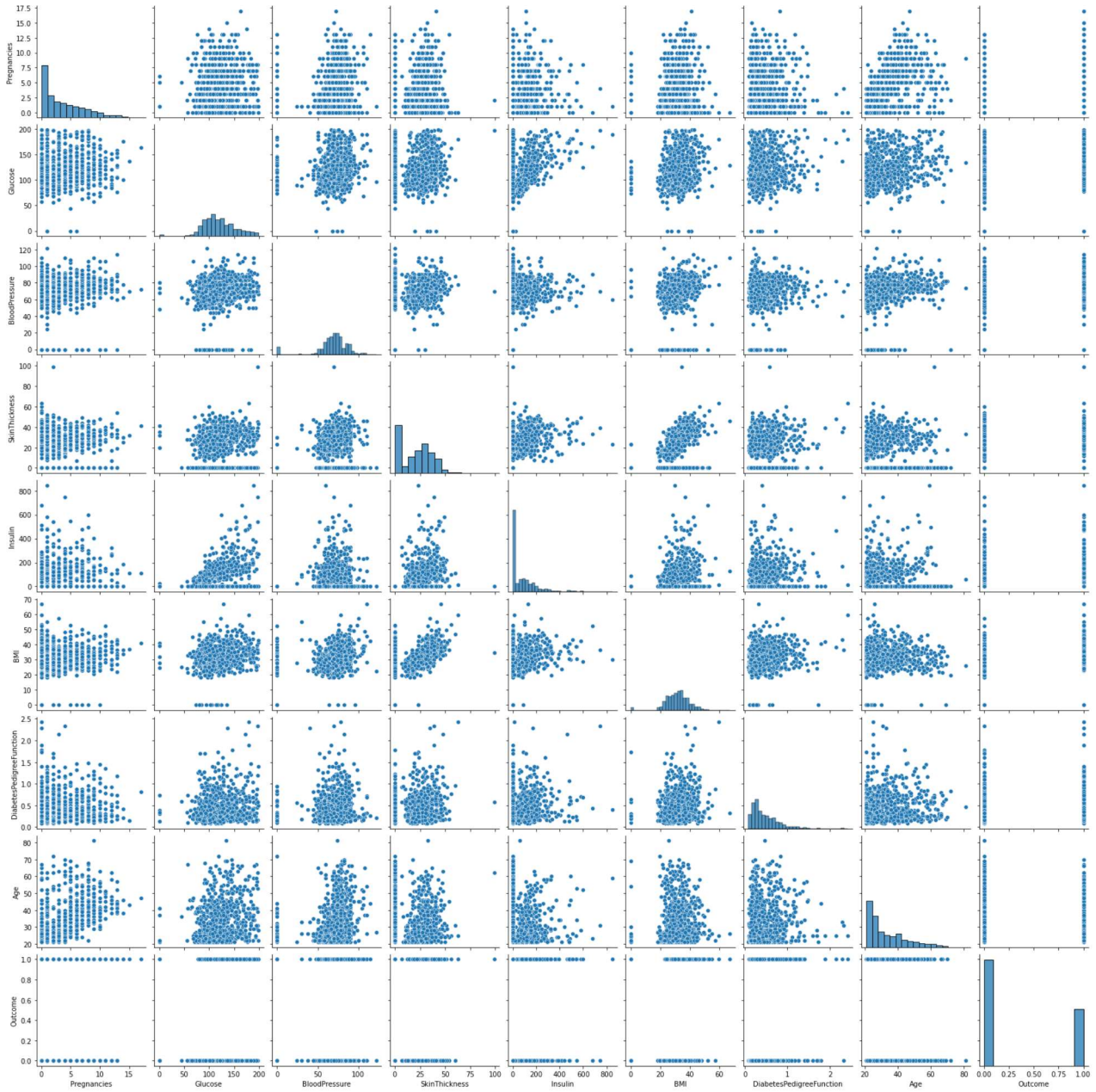
In [9]:
```
hc_copy=hc.copy(deep=True)
hc_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]=hc_copy[['Glucose'
```

In [10]:
```
hc_copy.isnull().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [11]:
```
sns.pairplot(hc)
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x1ccf5920a60>
```



In [12]:

```
diab=hc[hc['Outcome']==1]
diab.head()
```

Out[12]:

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

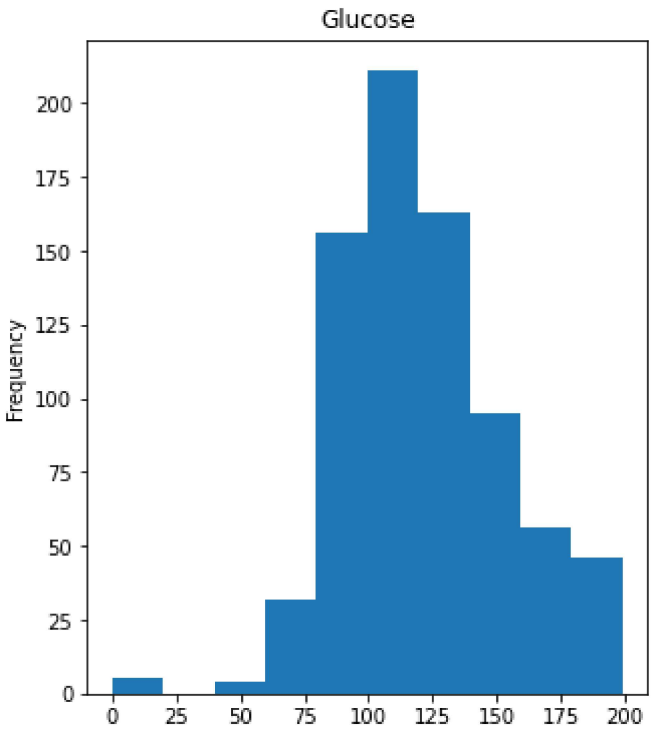
In [13]:

```
hc['Glucose'].value_counts().head(10)
```

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

In [14]:
```
hc['Glucose'].plot(kind='hist',figsize=(5,6),title='Glucose')
```

Out[14]:
```
<AxesSubplot:title={'center':'Glucose'}, ylabel='Frequency'>
```

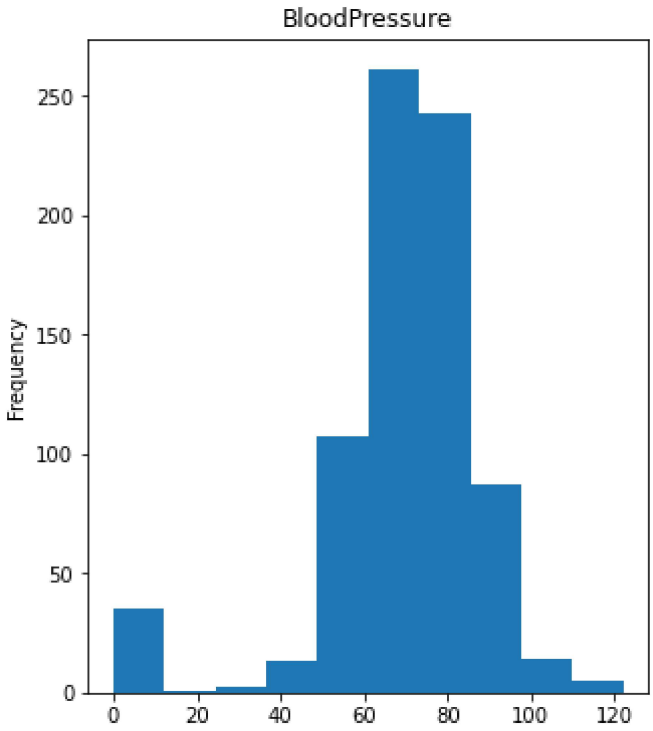


In [15]:
```
hc['BloodPressure'].value_counts().head()
```

Out[15]:
```
70    57
74    52
78    45
68    45
72    44
Name: BloodPressure, dtype: int64
```

In [16]:
```
hc['BloodPressure'].plot(kind='hist',figsize=(5,6),title='BloodPressure')
```

Out[16]:
```
<AxesSubplot:title={'center':'BloodPressure'}, ylabel='Frequency'>
```

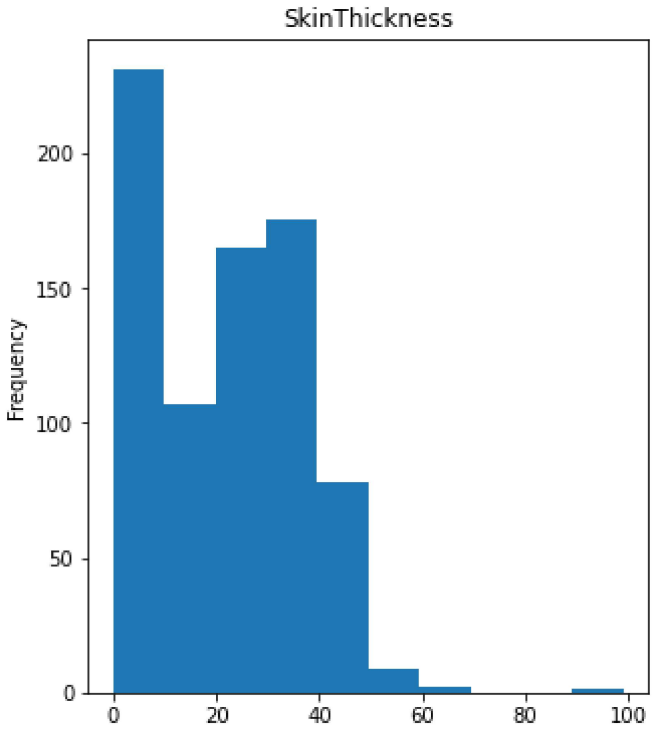


In [17]:
```
hc['SkinThickness'].value_counts().head(10)
```

Out[17]:
```
0     227
32     31
30     27
27     23
23     22
33     20
28     20
18     20
31     19
19     18
Name: SkinThickness, dtype: int64
```

In [18]:
```
hc['SkinThickness'].plot(kind='hist',figsize=(5,6),title='SkinThickness')
```

Out[18]:
```
<AxesSubplot:title={'center':'SkinThickness'}, ylabel='Frequency'>
```

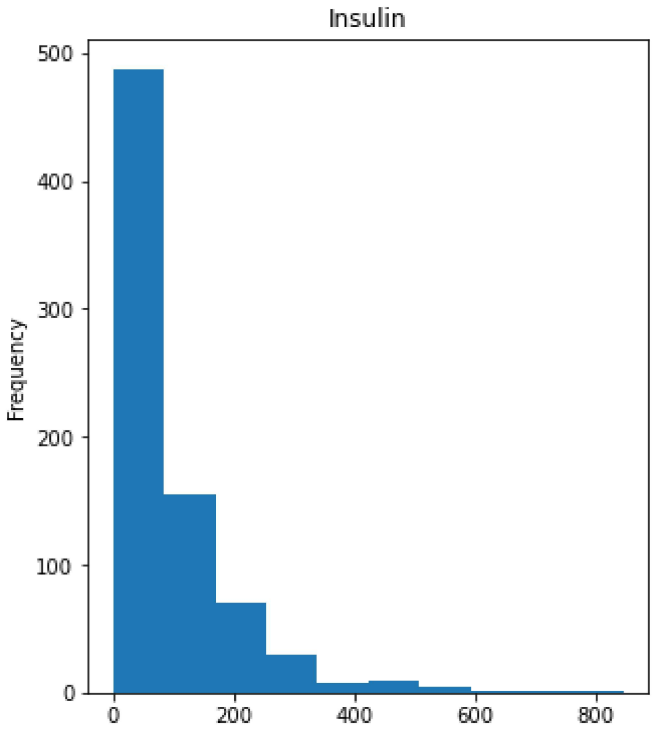


In [19]:
```
hc['Insulin'].value_counts().head()
```

Out[19]:
```
0      374
105     11
130      9
140      9
120      8
Name: Insulin, dtype: int64
```

In [20]:
```
hc['Insulin'].plot(kind='hist',figsize=(5,6),title='Insulin')
```

Out[20]:
```
<AxesSubplot:title={'center':'Insulin'}, ylabel='Frequency'>
```

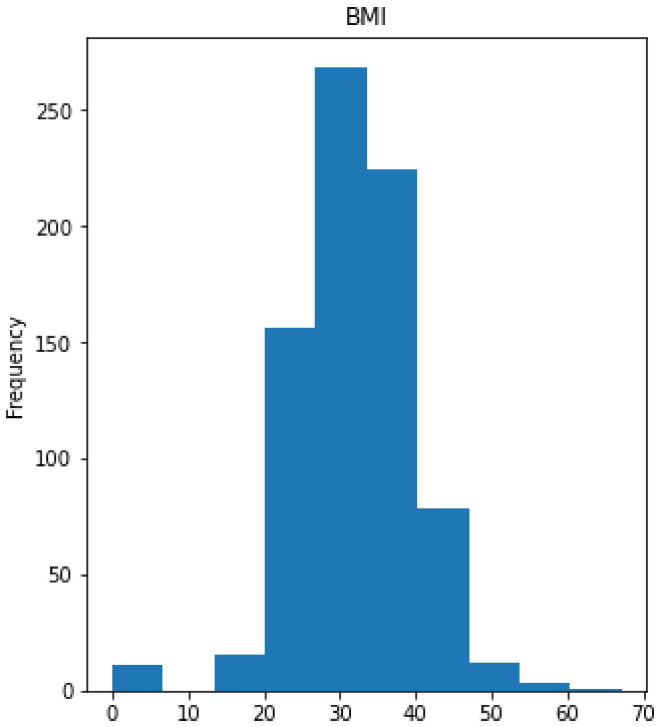


In [21]:
```
hc['BMI'].value_counts().head(10)
```

Out[21]:
```
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
33.3    10
30.1     9
32.8     9
32.9     9
30.8     9
Name: BMI, dtype: int64
```

In [22]:
```
hc['BMI'].plot(kind='hist',figsize=(5,6),title='BMI')
```

Out[22]:
```
<AxesSubplot:title={'center':'BMI'}, ylabel='Frequency'>
```



Filling missing values-

In [23]:
```
hc_copy.isnull().sum()
```

Out[23]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [24]:
```
#Replace missing values by Median
def median_val(var):
    med_hc=hc_copy[hc_copy[var].notnull()]
    med_hc=med_hc[[var,'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return med_hc
```

In [25]:
```
median_val('Glucose')
```

Out[25]:

	Outcome	Glucose
0	0	107.0
1	1	140.0

In [26]:
```
hc_copy.loc[(hc_copy['Outcome']==0)& hc_copy['Glucose'].isnull(),'Glucose']=107
hc_copy.loc[(hc_copy['Outcome']==1)& hc_copy['Glucose'].isnull(),'Glucose']=140
```

In [27]:
```
median_val('BloodPressure')
```

Out[27]:

	Outcome	BloodPressure
0	0	70.0
1	1	74.5

In [28]:
```
hc_copy.loc[(hc_copy['Outcome']==0)& hc_copy['BloodPressure'].isnull(),'BloodPressure']
hc_copy.loc[(hc_copy['Outcome']==1)& hc_copy['BloodPressure'].isnull(),'BloodPressure']
```

In [29]:
```
median_val('SkinThickness')
```

Out[29]:

	Outcome	SkinThickness
0	0	27.0
1	1	32.0

In [30]:
```
hc_copy.loc[(hc_copy['Outcome']==0)& hc_copy['SkinThickness'].isnull(),'SkinThickness']
hc_copy.loc[(hc_copy['Outcome']==1)& hc_copy['SkinThickness'].isnull(),'SkinThickness']
```

In [31]:
```
median_val('Insulin')
```

Out[31]:

	Outcome	Insulin
0	0	102.5
1	1	169.5

In [32]:
```
hc_copy.loc[(hc_copy['Outcome']==0)& hc_copy['Insulin'].isnull(),'Insulin']=102.5
hc_copy.loc[(hc_copy['Outcome']==1)& hc_copy['Insulin'].isnull(),'Insulin']=169.5
```

In [33]:
```
median_val('BMI')
```

Out[33]:

	Outcome	BMI
0	0	30.1
1	1	34.3

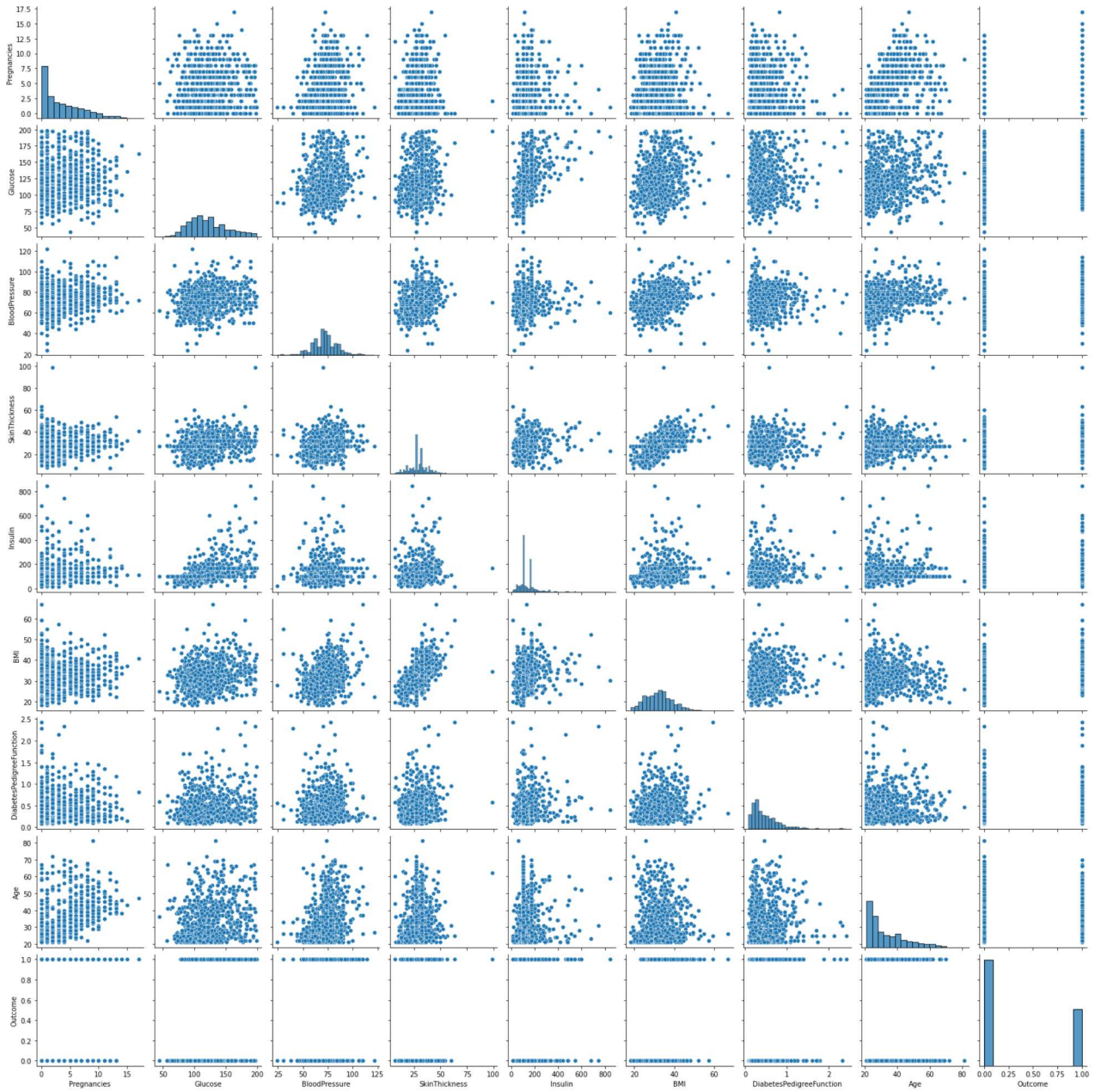
In [34]:
```
hc_copy.loc[(hc_copy['Outcome']==0)& hc_copy['BMI'].isnull(),'BMI']=30.1
hc_copy.loc[(hc_copy['Outcome']==1)& hc_copy['BMI'].isnull(),'BMI']=34.3
```

In [35]:
```
hc_copy.isnull().sum()
```

Out[35]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [36]:
```
#Pair plot after handling missing values
sns.pairplot(hc_copy)
```

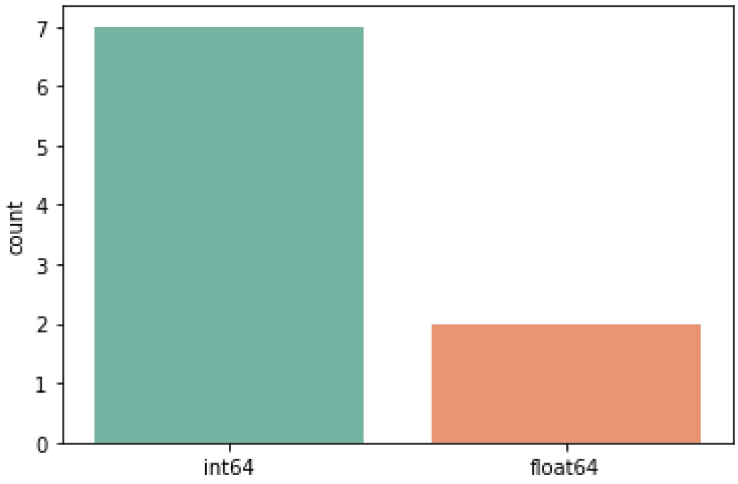
Out[36]:
```
<seaborn.axisgrid.PairGrid at 0x1ccfa787bb0>
```



In [37]:
```
#Create a count (frequency) plot describing the data types and the count of variables.
sns.countplot(hc.dtypes.map(str),palette='Set2')
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

Out[37]:
```
<AxesSubplot:ylabel='count'>
```



Data Exploration

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

In [38]:
```
hc.Outcome.value_counts()
```

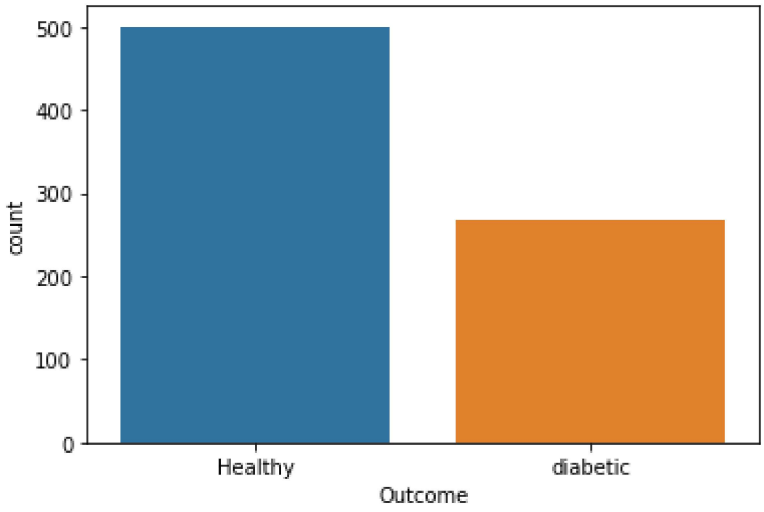
Out[38]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [39]:
```
diab_count=hc.Outcome.astype('category').cat.rename_categories(['Healthy','diabetic'])
```

In [40]:
```
sns.countplot(diab_count)
```

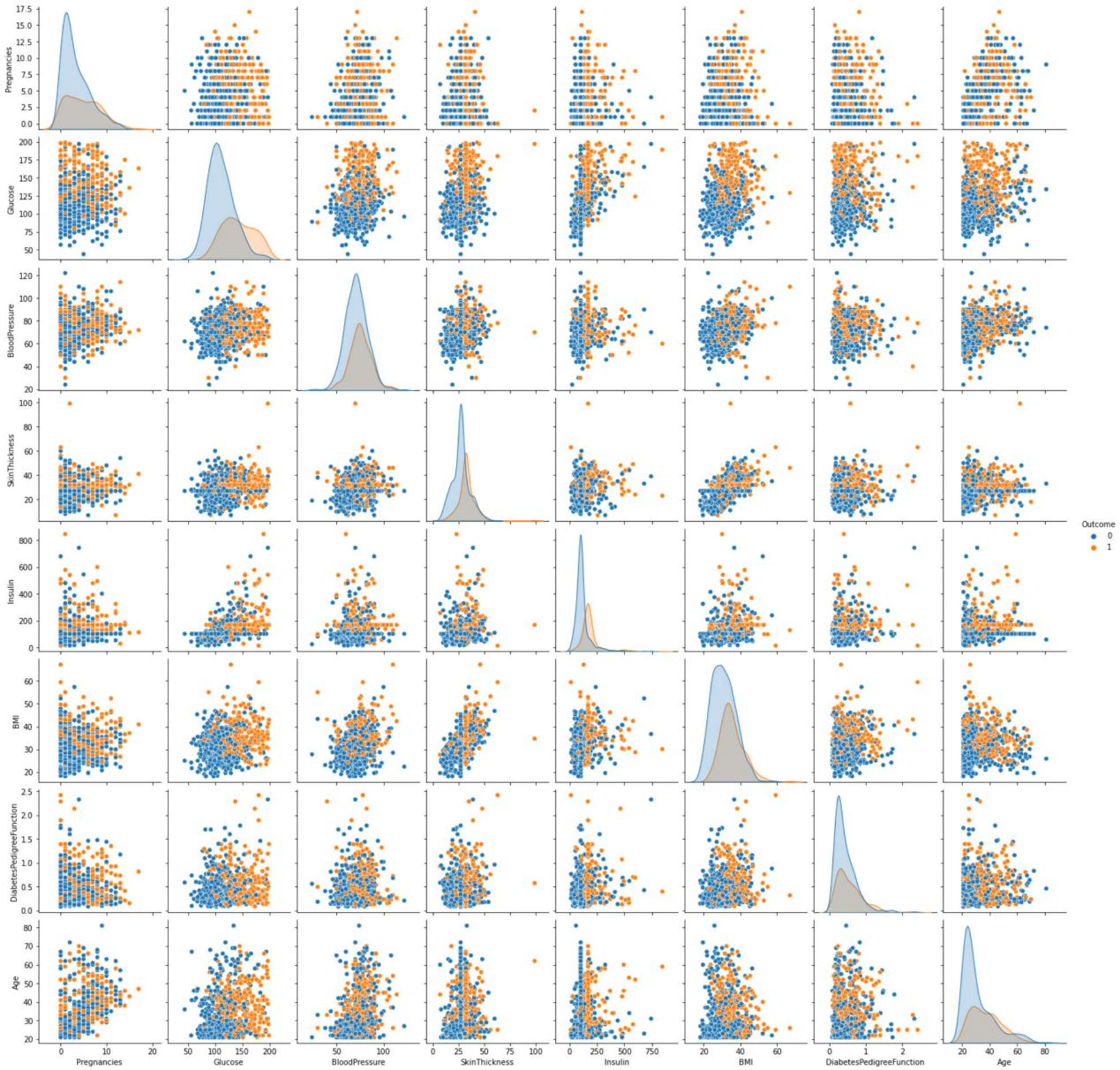
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

Out[40]:
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



In [41]:
```
sns.pairplot(hc_copy,hue='Outcome')
```

Out[41]:
```
<seaborn.axisgrid.PairGrid at 0x1cc81006a60>
```



Heatmap of original dataset

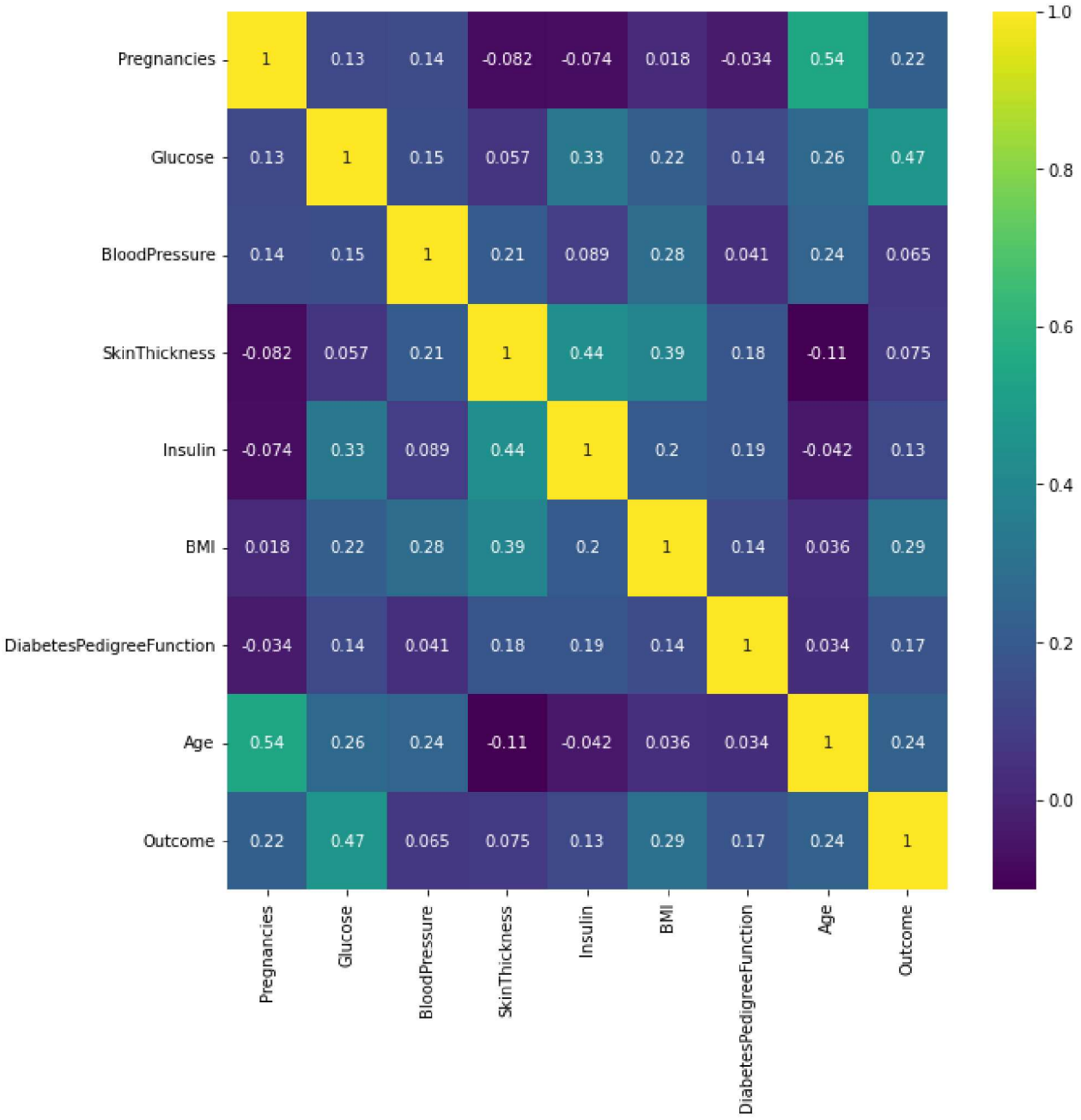
In [42]:
```
hc.corr()
```

Out[42]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	

In [43]:
```
plt.figure(figsize=(10,10))
sns.heatmap(hc.corr(),annot=True,cmap='viridis')
```

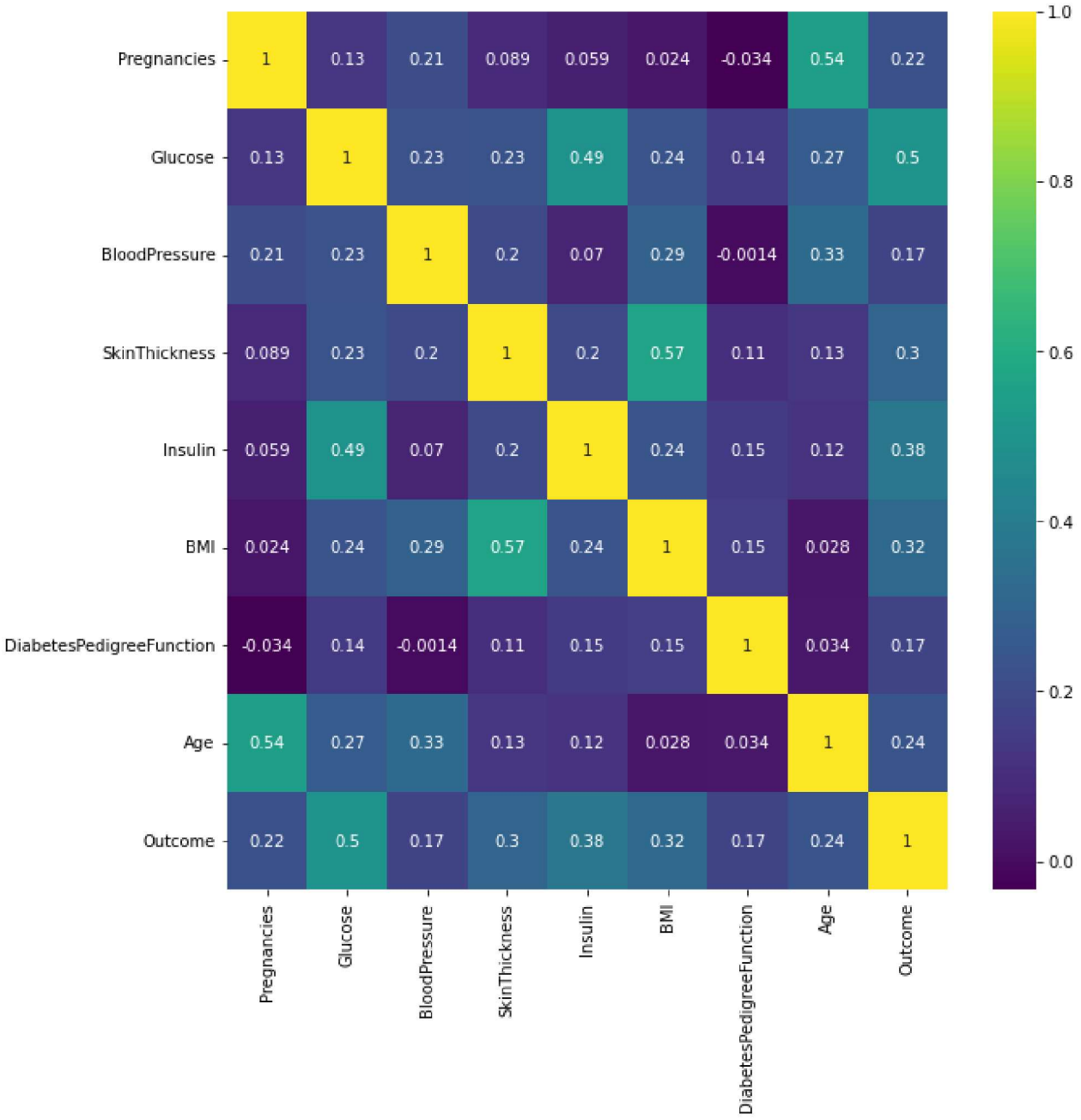
Out[43]:
```
<AxesSubplot:>
```



Heatmap of Clean data

In [44]:
```
plt.figure(figsize=(10,10))
sns.heatmap(hc_copy.corr(),annot=True,cmap='viridis')
```

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



From the above heatmap we see a bit of correlation between some columns-.

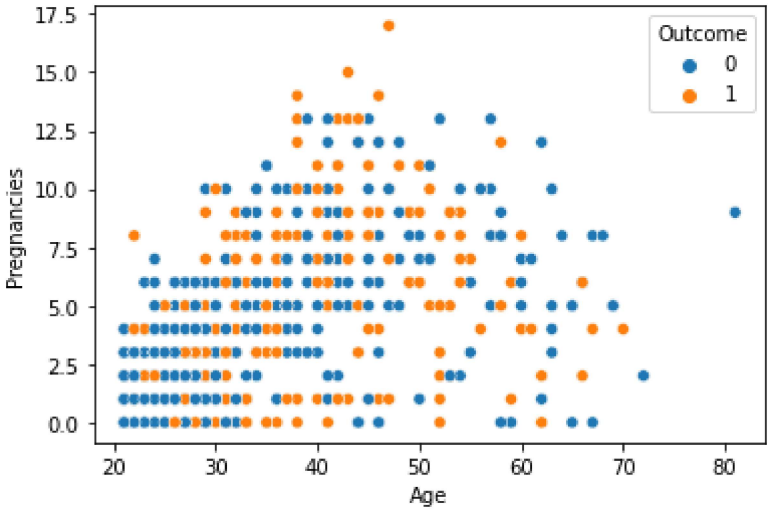
1.Age and Pregnancies = 0.54

2.Glucose and insulin = 0.49

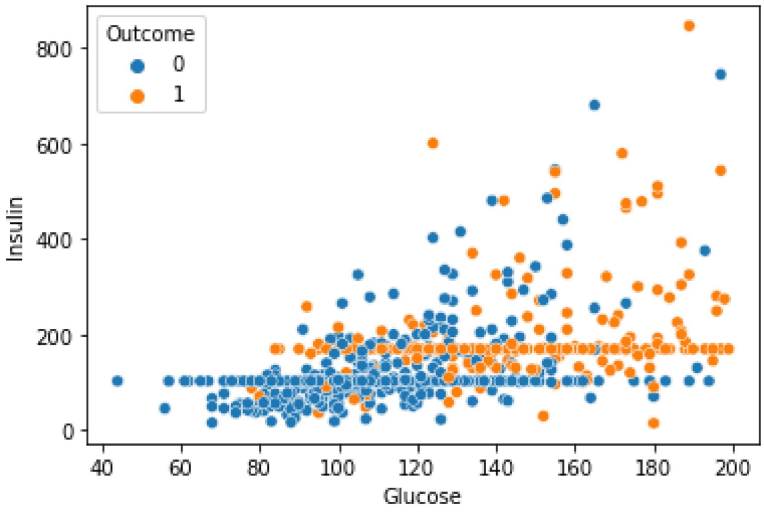
3.SkinThickness and BMI = 0.57

In [45]:
```
# let's visualise the relationship between above pairs
def pair(var1,var2):
    sns.scatterplot(x=var1,y=var2,data=hc_copy,hue='Outcome')
```

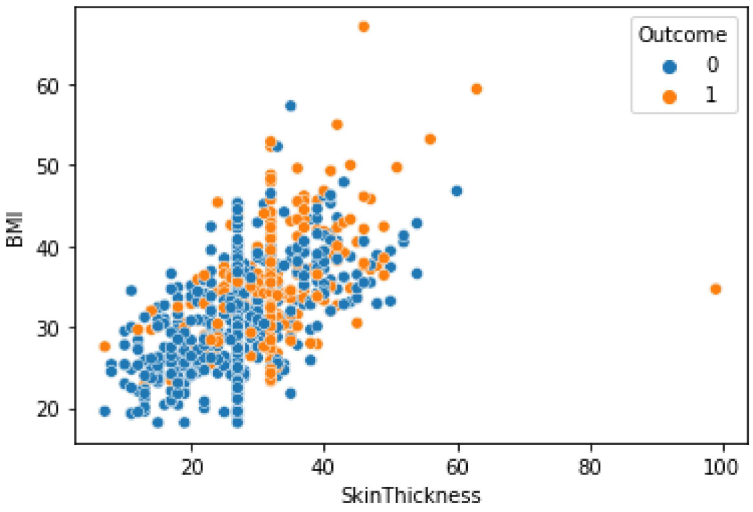
In [46]:
```
pair('Age','Pregnancies')
```



In [47]:
```
pair('Glucose','Insulin')
```



In [48]:
```
pair('SkinThickness','BMI')
```



Split the dataset

In [90]:
```
x=hc_copy.iloc[:,:-1]
y=hc_copy.iloc[:,-1]
```

In [91]:
```
x.shape,y.shape
```

Out[91]:
```
((768, 8), (768,))
```

In [92]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42,strati
```

In [93]:
```
x_train.shape,y_train.shape,y_test.shape,x_test.shape
```

Out[93]:
```
((537, 8), (537,), (231,), (231, 8))
```

In [94]:
```
#to bring whole data at a same scale then we will perform standardization
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_scaled=sc.fit_transform(x_train)
x_test_scaled=sc.transform(x_test)
```

Perform different models to check accuracy

1. LogisticRegression Model

In [95]:
```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
classifier.fit(x_train_scaled,y_train)
```

Out[95]:
```
LogisticRegression()
```

In [96]:
```
log_pred=classifier.predict(x_test_scaled)
```

In [97]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [98]:
```
print(classification_report(y_test,log_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81       150
           1       0.66      0.58      0.62        81

    accuracy                           0.75       231
   macro avg       0.72      0.71      0.72       231
weighted avg       0.74      0.75      0.74       231
```

In [99]:
```
print(confusion_matrix(y_test,log_pred))
```

```
[[126  24]
 [ 34  47]]
```

In [100…
```
print('\n','Accuracy-',accuracy_score(y_test,log_pred))
```

```
Accuracy- 0.7489177489177489
```

Random forest classifier

In [101…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc=RandomForestClassifier()
```

In [102…
```
n_estimators=[75,100,125,150,200]
max_features=[4,5,6,7,8]
bootstrap=[True,False]
oob_score=[True,False]
```

In [103…
```
param_grid={'n_estimators':n_estimators,
            'max_features':max_features,
             'bootstrap':bootstrap,
              'oob_score':oob_score}
```

In [104…
```
grid=GridSearchCV(rfc,param_grid)
```

In [108…
```
import warnings
warnings.filterwarnings('ignore')
```

In [109…
```
grid.fit(x_train_scaled,y_train)
```

Out[109…
```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'bootstrap': [True, False],
                         'max_features': [4, 5, 6, 7, 8],
                         'n_estimators': [75, 100, 125, 150, 200],
                         'oob_score': [True, False]})
```

In [110…
```
rfc_pred=grid.predict(x_test_scaled)
```

In [111…
```
print(classification_report(y_test,rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       150
           1       0.84      0.79      0.82        81

    accuracy                           0.87       231
   macro avg       0.87      0.86      0.86       231
```

```
weighted avg       0.87      0.87      0.87       231
```

In [112…
```
print(confusion_matrix(y_test,log_pred))
print('\n','Accuracy',accuracy_score(y_test,rfc_pred))
```

```
[[126  24]
 [ 34  47]]

 Accuracy 0.8744588744588745
```

In [114…
```
grid.best_params_
```

Out[114…
```
{'bootstrap': True, 'max_features': 4, 'n_estimators': 100, 'oob_score': True}
```

Support Vector Machine

In [115…
```
from sklearn.svm import SVC
```

In [116…
```
svc=SVC()
```

In [117…
```
param_grid = {'C':[0.01,0.1,1,10],'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
```

In [118…
```
grid=GridSearchCV(svc,param_grid)
```

In [119…
```
grid.fit(x_train_scaled,y_train)
```

Out[119…
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.01, 0.1, 1, 10],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']})
```

In [121…
```
svc_predict=grid.predict(x_test_scaled)
```

In [122…
```
print(classification_report(y_test,svc_predict))
```

```
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       150
           1       0.75      0.70      0.73        81

    accuracy                           0.81       231
   macro avg       0.80      0.79      0.79       231
weighted avg       0.81      0.81      0.81       231
```

In [123…
```
print(confusion_matrix(y_test,log_pred))
print('\n','Accuracy',accuracy_score(y_test,svc_predict))
```

```
[[126  24]
 [ 34  47]]

 Accuracy 0.8138528138528138
```

In [124…
```
grid.best_params_
```

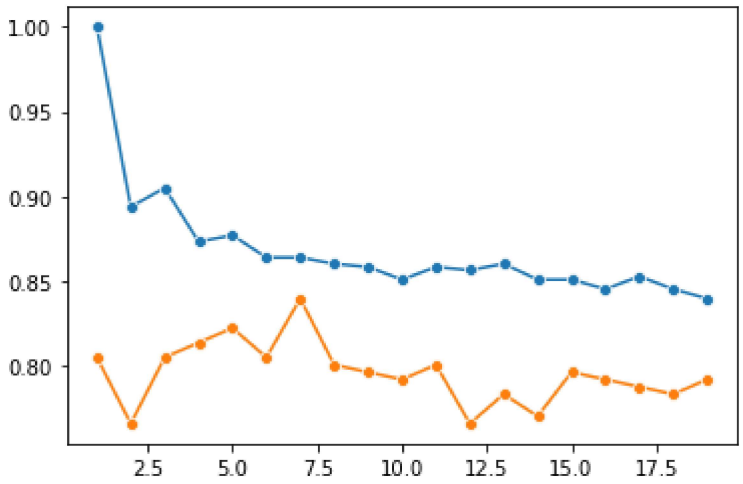
Out[124…
```
{'C': 1, 'kernel': 'rbf'}
```

4. K Nearest Neighbour

In [130…
```
from sklearn.neighbors import KNeighborsClassifier
train_score=[]
test_score=[]
for i in range(1,20):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train_scaled,y_train)
    train_score.append(knn.score(x_train_scaled,y_train))
    test_score.append(knn.score(x_test_scaled,y_test))
```

In [131…
```
sns.lineplot(x=range(1,20),y=train_score,marker='o')
sns.lineplot(x=range(1,20),y=test_score,marker='o')
```

Out[131…
```
<AxesSubplot:>
```



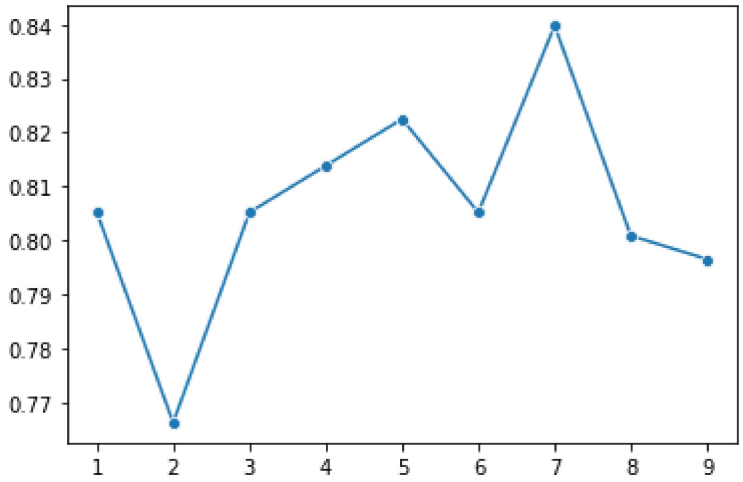
In [137…
```
acc_score=[]
for i in range(1,10):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train_scaled,y_train)
    knn_pred=knn.predict(x_test_scaled)
    acc_score.append(accuracy_score(y_test,knn_pred))
print(max(acc_score))
```

```
0.8398268398268398
```

In [138…
```
sns.lineplot(x=range(1,10),y=acc_score,marker='o')
```

```
<AxesSubplot:>
```

Out[138...



From the above results,n=7 gives the best results so we will take n_neighbors=7 for final model

In [141...

```
knn_model=KNeighborsClassifier(n_neighbors=7)
knn_model.fit(x_train_scaled,y_train)
```

Out[141...

```
KNeighborsClassifier(n_neighbors=7)
```

In [142...

```
knn_pred=knn_model.predict(x_test_scaled)
```

In [145...

```
print(classification_report(y_test,knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       150
           1       0.78      0.75      0.77        81

    accuracy                           0.84       231
   macro avg       0.83      0.82      0.82       231
weighted avg       0.84      0.84      0.84       231
```

In [147...

```
print(confusion_matrix(y_test,knn_pred))
print('\n','Accuracy',accuracy_score(y_test,knn_pred))
```

```
[[133  17]
 [ 20  61]]

 Accuracy 0.8398268398268398
```

5. Decision Tree

In [148...

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(random_state=50)
dtc.fit(x_train_scaled,y_train)
```

Out[148… DecisionTreeClassifier(random_state=50)

In [149…
```
dtc_pred=dtc.predict(x_test_scaled)
```

In [151…
```
print(classification_report(y_test,dtc_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.88      0.87       150
           1       0.77      0.75      0.76        81

    accuracy                           0.84       231
   macro avg       0.82      0.82      0.82       231
weighted avg       0.83      0.84      0.84       231
```

In [152…
```
print(confusion_matrix(y_test,dtc_pred))
```

```
[[132  18]
 [ 20  61]]
```

In [153…
```
print(accuracy_score(y_test,dtc_pred))
```

```
0.8354978354978355
```

Accuracy,Sensitivity,Specificity

In [156…
```
models=[
    {'label':'LogisticRegression',
      'model':LogisticRegression(),
    },
    {'label':'RandomForestClassifier',
      'model':RandomForestClassifier(),
    },
    {'label': 'KNeighbors Classifier',
    'model': KNeighborsClassifier(n_neighbors=7),
},
{
    'label' : 'Support Vector Classifier',
    'model' : SVC(C= 1, kernel='rbf',probability=True),
},
{
    'label' : 'Decision Tress',
    'model' : DecisionTreeClassifier(random_state=42,criterion= 'entropy', max_features
},

]
```

In [160…
```
accu=[]
model_name=[]
sensitivity=[]
specificity=[]
for m in models:
```

```
    mod_1=m['model']
    mod_1.fit(x_train_scaled,y_train)
    mod_pred=mod_1.predict(x_test_scaled)
    cm=confusion_matrix(y_test,mod_pred)

    accu.append(accuracy_score(y_test,mod_pred))
    model_name.append(m['label'])

    sensitivity.append(cm[0,0]/(cm[0,0]+cm[0,1]))
    specificity.append(cm[1,1]/(cm[1,0]+cm[1,1]))

    models_accu_sen_sp= pd.DataFrame(data=(accu,sensitivity,specificity),index = ['Accu
                                     columns=[model_name]).T
models_accu_sen_sp
```

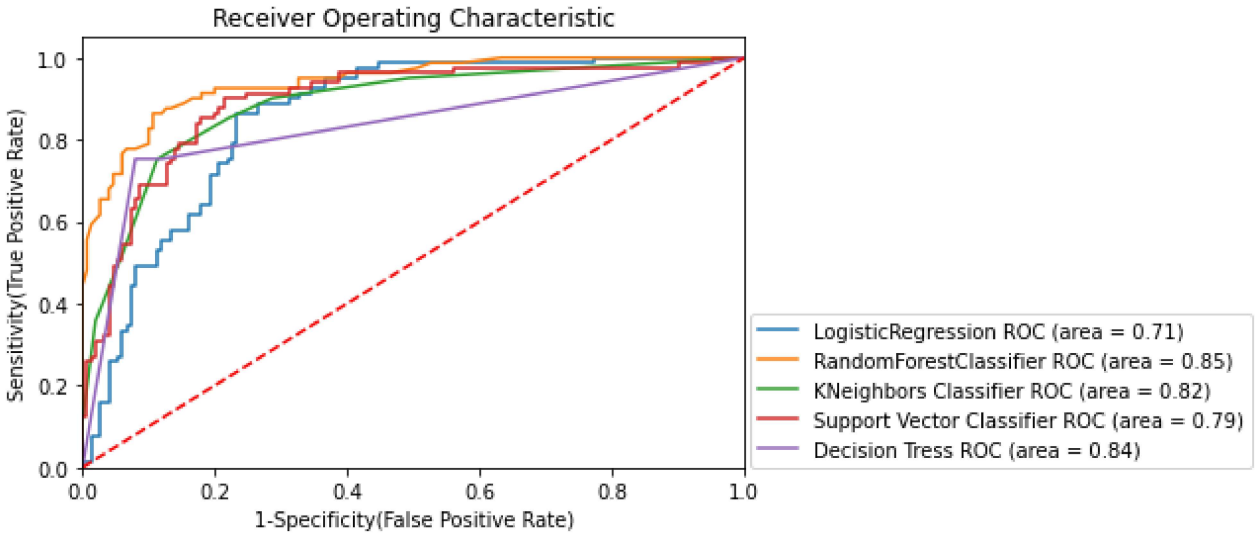
Out[160…

	Accuracy	Sensitivity	Specificity
LogisticRegression	0.748918	0.840000	0.580247
RandomForestClassifier	0.874459	0.926667	0.777778
KNeighbors Classifier	0.839827	0.886667	0.753086
Support Vector Classifier	0.813853	0.873333	0.703704
Decision Tress	0.861472	0.920000	0.753086

ROC curve

In [164…

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
# Below for loop iterates through your models list
for m in models:
    model = m['model'] # select the model
    model.fit(x_train_scaled, y_train) # train the model
    y_pred=model.predict(x_test_scaled) # predict the test data
# Compute False postive rate, and True positive rate
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test_scaled)[:,1])
# Calculate Area under the curve to display on the plot
    auc = roc_auc_score(y_test,model.predict(x_test_scaled))
# Now, plot the computed values
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))
# Custom settings for the plot
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc =(1.01,0))
plt.show()   # Display
```



Random forest classifier has the best curve area so this is the best model for this problem.

In [ ]: