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.

Dataset Description

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

Variables Description

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

Importing Necessary Libraries

In [190…

```
#Classic, Data Manipulation

import pandas as pd
import numpy as np

#Plots
import matplotlib.pyplot as plt
import seaborn as sns


#Data processing, metrics and modeling

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score, p
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# to display Image files
from PIL import Image as PILImage
```

```
#ignore warning messages
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('C:/Users/vipul/Downloads/Project_2/Project 2/Healthcare - Diabetes
```

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

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	5(
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	3:

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

Out[4]: (768, 9)

In [5]:
```
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
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	

Unique

In [6]:
```
for  i in df.columns:
    print(i,df[i].unique(),'\n')
```

```
Pregnancies [ 6  1  8  0  5  3 10  2  4  7  9 11 13 15 17 12 14]

Glucose [148  85 183  89 137 116  78 115 197 125 110 168 139 189 166 100 118 107
 103 126  99 196 119 143 147  97 145 117 109 158  88  92 122 138 102  90
 111 180 133 106 171 159 146  71 105 101 176 150  73 187  84  44 141 114
  95 129  79   0  62 131 112 113  74  83 136  80 123  81 134 142 144  93
 163 151  96 155  76 160 124 162 132 120 173 170 128 108 154  57 156 153
 188 152 104  87  75 179 130 194 181 135 184 140 177 164  91 165  86 193
 191 161 167  77 182 157 178  61  98 127  82  72 172  94 175 195  68 186
 198 121  67 174 199  56 169 149  65 190]
```

```
BloodPressure [ 72  66  64  40  74  50   0  70  96  92  80  60  84  30  88  90  94
76
  82  75  58  78  68 110  56  62  85  86  48  44  65 108  55 122  54  52
  98 104  95  46 102 100  61  24  38 106 114]

SkinThickness [35 29  0 23 32 45 19 47 38 30 41 33 26 15 36 11 31 37 42 25 18 24 39
27
 21 34 10 60 13 20 22 28 54 40 51 56 14 17 50 44 12 46 16  7 52 43 48  8
 49 63 99]

Insulin [  0  94 168  88 543 846 175 230  83  96 235 146 115 140 110 245  54 192
 207  70 240  82  36  23 300 342 304 142 128  38 100  90 270  71 125 176
  48  64 228  76 220  40 152  18 135 495  37  51  99 145 225  49  50  92
 325  63 284 119 204 155 485  53 114 105 285 156  78 130  55  58 160 210
 318  44 190 280  87 271 129 120 478  56  32 744 370  45 194 680 402 258
 375 150  67  57 116 278 122 545  75  74 182 360 215 184  42 132 148 180
 205  85 231  29  68  52 255 171  73 108  43 167 249 293  66 465  89 158
  84  72  59  81 196 415 275 165 579 310  61 474 170 277  60  14  95 237
 191 328 250 480 265 193  79  86 326 188 106  65 166 274  77 126 330 600
 185  25  41 272 321 144  15 183  91  46 440 159 540 200 335 387  22 291
 392 178 127 510  16 112]

BMI [33.6 26.6 23.3 28.1 43.1 25.6 31.  35.3 30.5  0.  37.6 38.  27.1 30.1
 25.8 30.  45.8 29.6 43.3 34.6 39.3 35.4 39.8 29.  36.6 31.1 39.4 23.2
 22.2 34.1 36.  31.6 24.8 19.9 27.6 24.  33.2 32.9 38.2 37.1 34.  40.2
 22.7 45.4 27.4 42.  29.7 28.  39.1 19.4 24.2 24.4 33.7 34.7 23.  37.7
 46.8 40.5 41.5 25.  25.4 32.8 32.5 42.7 19.6 28.9 28.6 43.4 35.1 32.
 24.7 32.6 43.2 22.4 29.3 24.6 48.8 32.4 38.5 26.5 19.1 46.7 23.8 33.9
 20.4 28.7 49.7 39.  26.1 22.5 39.6 29.5 34.3 37.4 33.3 31.2 28.2 53.2
 34.2 26.8 55.  42.9 34.5 27.9 38.3 21.1 33.8 30.8 36.9 39.5 27.3 21.9
 40.6 47.9 50.  25.2 40.9 37.2 44.2 29.9 31.9 28.4 43.5 32.7 67.1 45.
 34.9 27.7 35.9 22.6 33.1 30.4 52.3 24.3 22.9 34.8 30.9 40.1 23.9 37.5
 35.5 42.8 42.6 41.8 35.8 37.8 28.8 23.6 35.7 36.7 45.2 44.  46.2 35.
 43.6 44.1 18.4 29.2 25.9 32.1 36.3 40.  25.1 27.5 45.6 27.8 24.9 25.3
 37.9 27.  26.  38.7 20.8 36.1 30.7 32.3 52.9 21.  39.7 25.5 26.2 19.3
 38.1 23.5 45.5 23.1 39.9 36.8 21.8 41.  42.2 34.4 27.2 36.5 29.8 39.2
 38.4 36.2 48.3 20.  22.3 45.7 23.7 22.1 42.1 42.4 18.2 26.4 45.3 37.
 24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
 37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
 38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]

DiabetesPedigreeFunction [0.627 0.351 0.672 0.167 2.288 0.201 0.248 0.134 0.158 0.23
2 0.191 0.537
 1.441 0.398 0.587 0.484 0.551 0.254 0.183 0.529 0.704 0.388 0.451 0.263
 0.205 0.257 0.487 0.245 0.337 0.546 0.851 0.267 0.188 0.512 0.966 0.42
 0.665 0.503 1.39  0.271 0.696 0.235 0.721 0.294 1.893 0.564 0.586 0.344
 0.305 0.491 0.526 0.342 0.467 0.718 0.962 1.781 0.173 0.304 0.27  0.699
 0.258 0.203 0.855 0.845 0.334 0.189 0.867 0.411 0.583 0.231 0.396 0.14
 0.391 0.37  0.307 0.102 0.767 0.237 0.227 0.698 0.178 0.324 0.153 0.165
 0.443 0.261 0.277 0.761 0.255 0.13  0.323 0.356 0.325 1.222 0.179 0.262
 0.283 0.93  0.801 0.207 0.287 0.336 0.247 0.199 0.543 0.192 0.588 0.539
 0.22  0.654 0.223 0.759 0.26  0.404 0.186 0.278 0.496 0.452 0.403 0.741
 0.361 1.114 0.457 0.647 0.088 0.597 0.532 0.703 0.159 0.268 0.286 0.318
 0.272 0.572 0.096 1.4   0.218 0.085 0.399 0.432 1.189 0.687 0.137 0.637
 0.833 0.229 0.817 0.204 0.368 0.743 0.722 0.256 0.709 0.471 0.495 0.18
 0.542 0.773 0.678 0.719 0.382 0.319 0.19  0.956 0.084 0.725 0.299 0.244
 0.745 0.615 1.321 0.64  0.142 0.374 0.383 0.578 0.136 0.395 0.187 0.905
 0.15  0.874 0.236 0.787 0.407 0.605 0.151 0.289 0.355 0.29  0.375 0.164
 0.431 0.742 0.514 0.464 1.224 1.072 0.805 0.209 0.666 0.101 0.198 0.652
 2.329 0.089 0.645 0.238 0.394 0.293 0.479 0.686 0.831 0.582 0.446 0.402
 1.318 0.329 1.213 0.427 0.282 0.143 0.38  0.284 0.249 0.926 0.557 0.092
 0.655 1.353 0.612 0.2   0.226 0.997 0.933 1.101 0.078 0.24  1.136 0.128
 0.422 0.251 0.677 0.296 0.454 0.744 0.881 0.28  0.259 0.619 0.808 0.34
 0.434 0.757 0.613 0.692 0.52  0.412 0.84  0.839 0.156 0.215 0.326 1.391
 0.875 0.313 0.433 0.626 1.127 0.315 0.345 0.129 0.527 0.197 0.731 0.148
 0.123 0.127 0.122 1.476 0.166 0.932 0.343 0.893 0.331 0.472 0.673 0.389
 0.485 0.349 0.279 0.346 0.252 0.243 0.58  0.559 0.302 0.569 0.378 0.385
 0.499 0.306 0.234 2.137 1.731 0.545 0.225 0.816 0.528 0.509 1.021 0.821
```

```
 0.947 1.268 0.221 0.66  0.239 0.949 0.444 0.463 0.803 1.6   0.944 0.196
 0.241 0.161 0.135 0.376 1.191 0.702 0.674 1.076 0.534 1.095 0.554 0.624
 0.219 0.507 0.561 0.421 0.516 0.264 0.328 0.233 0.108 1.138 0.147 0.727
 0.435 0.497 0.23  0.955 2.42  0.658 0.33  0.51  0.285 0.415 0.381 0.832
 0.498 0.212 0.364 1.001 0.46  0.733 0.416 0.705 1.022 0.269 0.6   0.571
 0.607 0.17  0.21  0.126 0.711 0.466 0.162 0.419 0.63  0.365 0.536 1.159
 0.629 0.292 0.145 1.144 0.174 0.547 0.163 0.738 0.314 0.968 0.409 0.297
 0.525 0.154 0.771 0.107 0.493 0.717 0.917 0.501 1.251 0.735 0.804 0.661
 0.549 0.825 0.423 1.034 0.16  0.341 0.68  0.591 0.3   0.121 0.502 0.401
 0.601 0.748 0.338 0.43  0.892 0.813 0.693 0.575 0.371 0.206 0.417 1.154
 0.925 0.175 1.699 0.682 0.194 0.4   0.1   1.258 0.482 0.138 0.593 0.878
 0.157 1.282 0.141 0.246 1.698 1.461 0.347 0.362 0.393 0.144 0.732 0.115
 0.465 0.649 0.871 0.149 0.695 0.303 0.61  0.73  0.447 0.455 0.133 0.155
 1.162 1.292 0.182 1.394 0.217 0.631 0.88  0.614 0.332 0.366 0.181 0.828
 0.335 0.856 0.886 0.439 0.253 0.598 0.904 0.483 0.565 0.118 0.177 0.176
 0.295 0.441 0.352 0.826 0.97  0.595 0.317 0.265 0.646 0.426 0.56  0.515
 0.453 0.785 0.734 1.174 0.488 0.358 1.096 0.408 1.182 0.222 1.057 0.766
 0.171]

Age [50 31 32 21 33 30 26 29 53 54 34 57 59 51 27 41 43 22 38 60 28 45 35 46
 56 37 48 40 25 24 58 42 44 39 36 23 61 69 62 55 65 47 52 66 49 63 67 72
 81 64 70 68]

Outcome [1 0]
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

What all features have zero values

In [8]:
```
for i in df.columns[1:-1]:
    l = len(df[df[i]==0])
    if l>=1:
        print(i,'---- has total {} Zero values'.format(l))
    else:
        print(i,'---- has no zero values and is good to go')
```

```
Glucose ---- has total 5 Zero values
BloodPressure ---- has total 35 Zero values
SkinThickness ---- has total 227 Zero values
Insulin ---- has total 374 Zero values
BMI ---- has total 11 Zero values
DiabetesPedigreeFunction ---- has no zero values and is good to go
Age ---- has no zero values and is good to go
```

A person can not have zero values for Glucose, Bloodpressure, SkinThickness, Insulin, BMI and Diabetes Pedigress Function. All these zero values don't make any sense hence these are nothing but the missing values. So we'll treat them with missing values imputation techniques

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

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

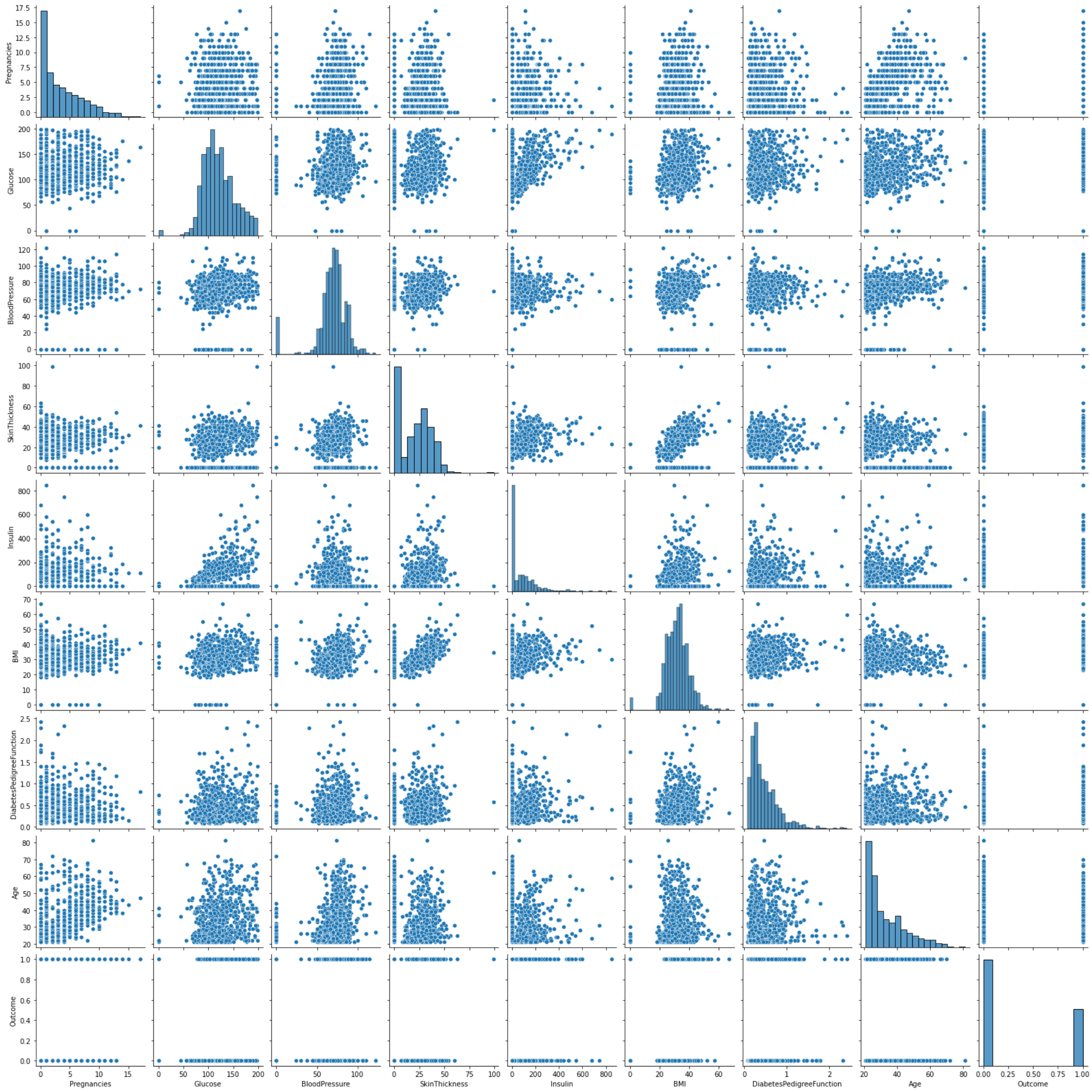
Out[10]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
```

```
Outcome                      0
dtype: int64
```

Let's see the distribution of data points in order to fill the null values.

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

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x249dc9ccd60>
```



Filling Missing Values

In [12]:
```
df_copy.isnull().sum()
```

Out[12]:
```
Pregnancies                  0
Glucose                      5
BloodPressure               35
SkinThickness              227
Insulin                    374
BMI                         11
DiabetesPedigreeFunction     0
Age                          0
Outcome                      0
dtype: int64
```

In [13]:
```
#Function to find median
```

```
def median_imp(var):
    med_df = df_copy[df_copy[var].notnull()]
    med_df = med_df[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_ind
    return med_df
```

In [14]:
```
median_imp('Glucose')
```

Out[14]:

	Outcome	Glucose
0	0	107.0
1	1	140.0

In [15]:
```
df_copy.loc[(df_copy['Outcome']==0) & df_copy['Glucose'].isnull(),'Glucose'] = 107
df_copy.loc[(df_copy['Outcome']==1) & df_copy['Glucose'].isnull(),'Glucose'] = 140
```

In [16]:
```
median_imp('BloodPressure')
```

Out[16]:

	Outcome	BloodPressure
0	0	70.0
1	1	74.5

In [17]:
```
df_copy.loc[(df_copy['Outcome']==0) & df_copy['BloodPressure'].isnull(),'BloodPressu
df_copy.loc[(df_copy['Outcome']==1) & df_copy['BloodPressure'].isnull(),'BloodPressu
```

In [18]:
```
median_imp('SkinThickness')
```

Out[18]:

	Outcome	SkinThickness
0	0	27.0
1	1	32.0

In [19]:
```
df_copy.loc[(df_copy['Outcome']==0) & df_copy['SkinThickness'].isnull(),'SkinThickne
df_copy.loc[(df_copy['Outcome']==1) & df_copy['SkinThickness'].isnull(),'SkinThickne
```

In [20]:
```
median_imp('Insulin')
```

Out[20]:

	Outcome	Insulin
0	0	102.5
1	1	169.5

In [21]:
```
df_copy.loc[(df_copy['Outcome']==0) & df_copy['Insulin'].isnull(),'Insulin'] = 102.5
df_copy.loc[(df_copy['Outcome']==1) & df_copy['Insulin'].isnull(),'Insulin'] = 169.5
```

In [22]:
```
median_imp('BMI')
```

Out[22]:

	Outcome	BMI
0	0	30.1
1	1	34.3

In [23]:
```
df_copy.loc[(df_copy['Outcome']==0) & df_copy['BMI'].isnull(),'BMI'] = 30.1
df_copy.loc[(df_copy['Outcome']==1) & df_copy['BMI'].isnull(),'BMI'] = 34.3
```

In [24]:
```
df_copy.isnull().sum()
```

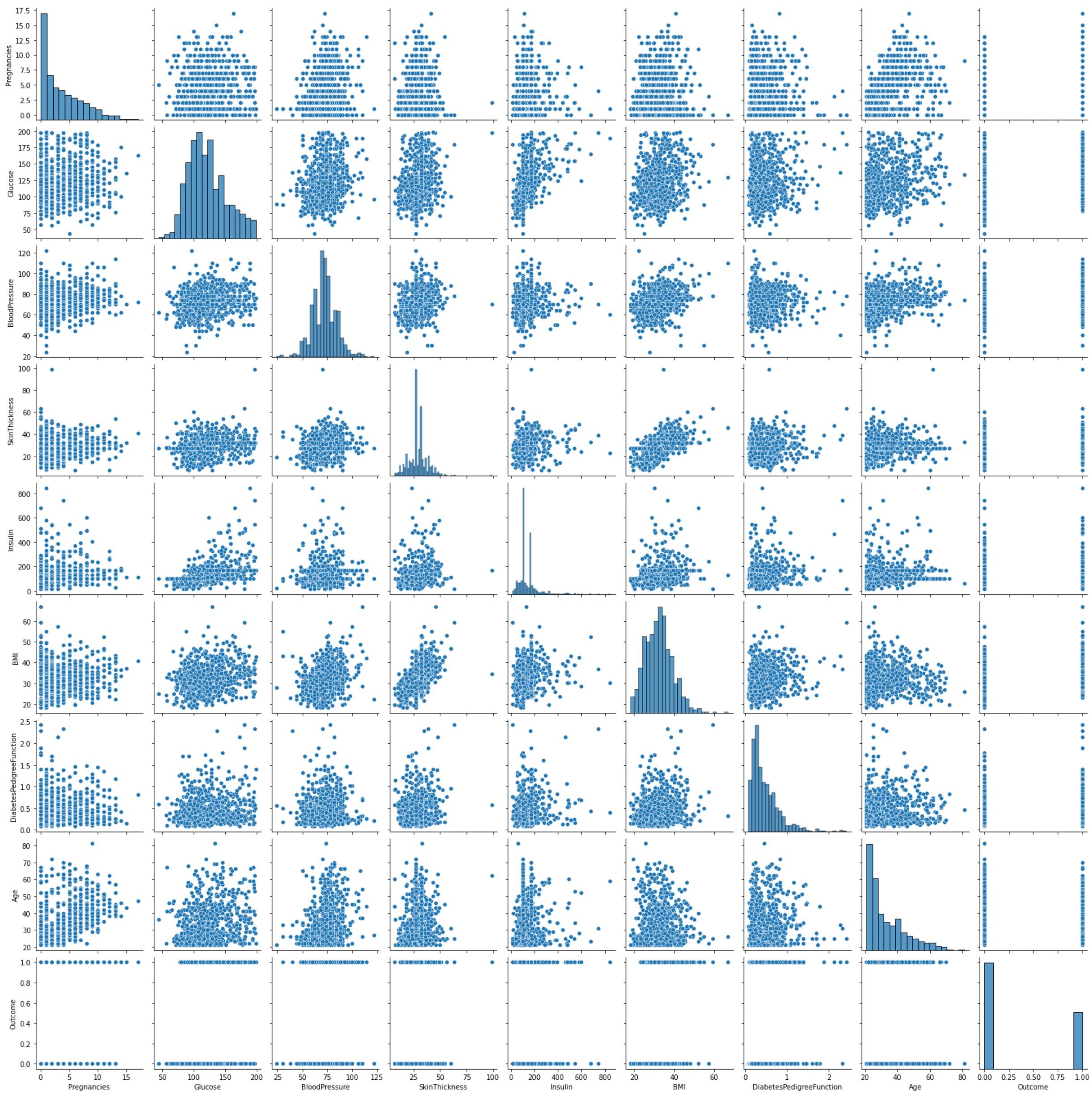
Out[24]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Now our dataset is free from any null values so we can proceed further

Pair Plot after handeling missing values

In [25]:
```
sns.pairplot(df_copy)
```

Out[25]:
```
<seaborn.axisgrid.PairGrid at 0x249dc9aeeb0>
```



Count of types of columns in dataset

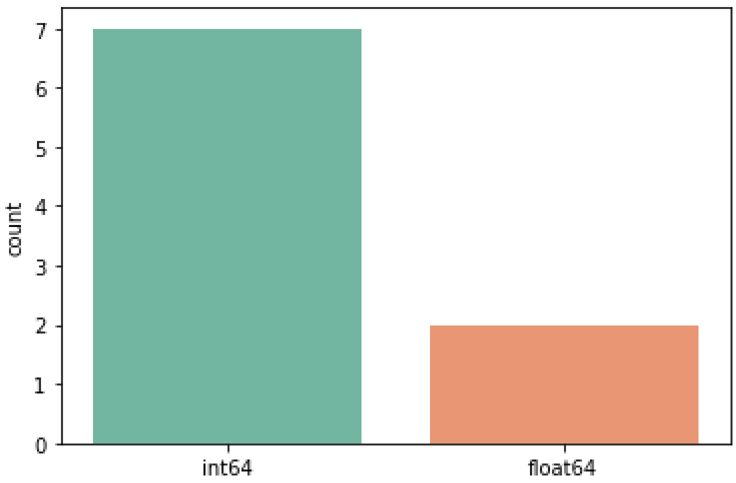
In [27]:
```
int_dtype = df.select_dtypes(include=['int64']).columns
float_dtype = df.select_dtypes(include=['float64']).columns
obj_dtype = df.select_dtypes(include=['object']).columns
```

In [28]:
```
print('No of integer columns in dataframe is :',len(int_dtype))
print('No of flaot columns in dataframe is   :',len(float_dtype))
print('No of object columns in dataframe is  :',len(obj_dtype))
```

```
No of integer columns in dataframe is : 7
No of flaot columns in dataframe is   : 2
No of object columns in dataframe is  : 0
```

In [29]:
```
sns.countplot(x = df.dtypes.map(str),palette='Set2')
```

Out[29]: <AxesSubplot:ylabel='count'>



Count of diabetic and healty people in dataset

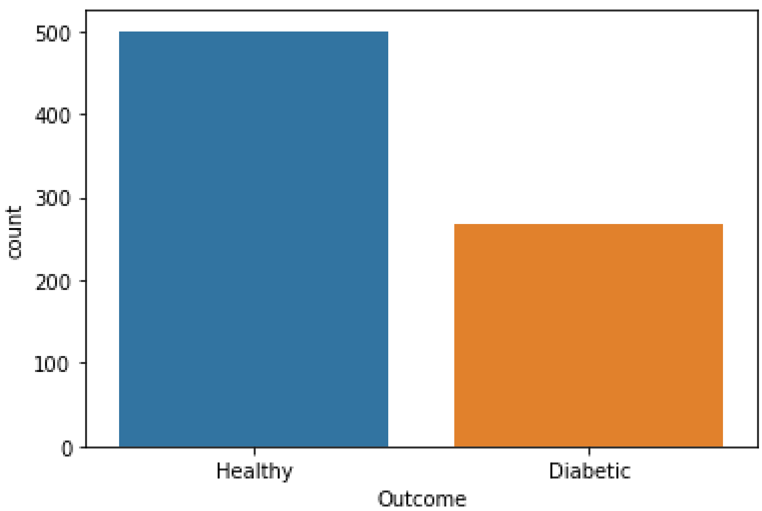
In [30]:
```
df.Outcome.value_counts()
```

Out[30]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [31]:
```
diab_count = df.Outcome.astype('category').cat.rename_categories(['Healthy','Diabeti
```

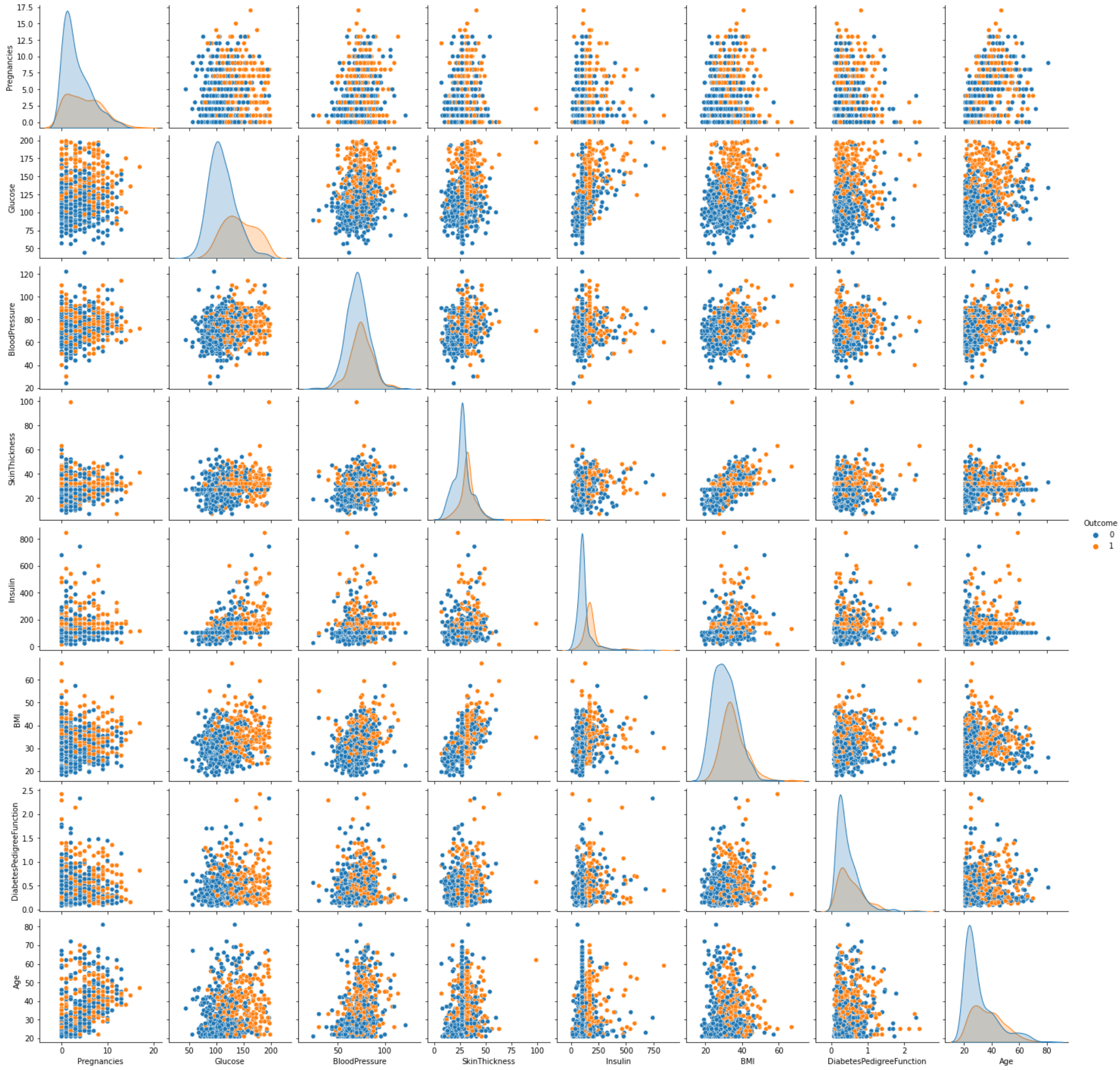
In [32]:
```
sns.countplot(x= diab_count)
```

Out[32]: <AxesSubplot:xlabel='Outcome', ylabel='count'>



In [33]:
```
sns.pairplot(df_copy,hue = 'Outcome')
```

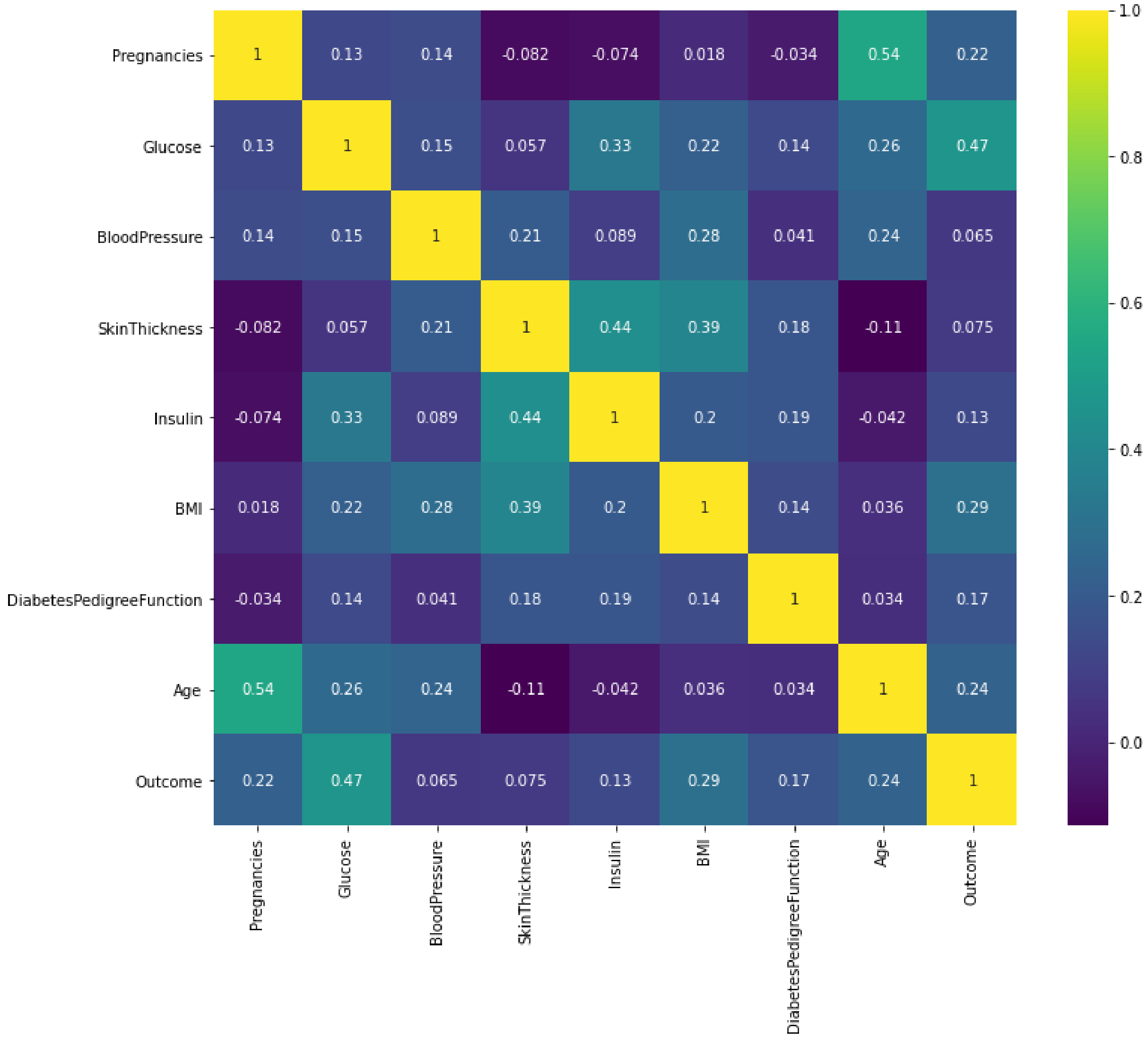
Out[33]: <seaborn.axisgrid.PairGrid at 0x249e5ecd070>



Heatmap of Original Dataset

In [34]:
```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),annot=True,cmap = 'viridis')
```

Out[34]: <AxesSubplot:>

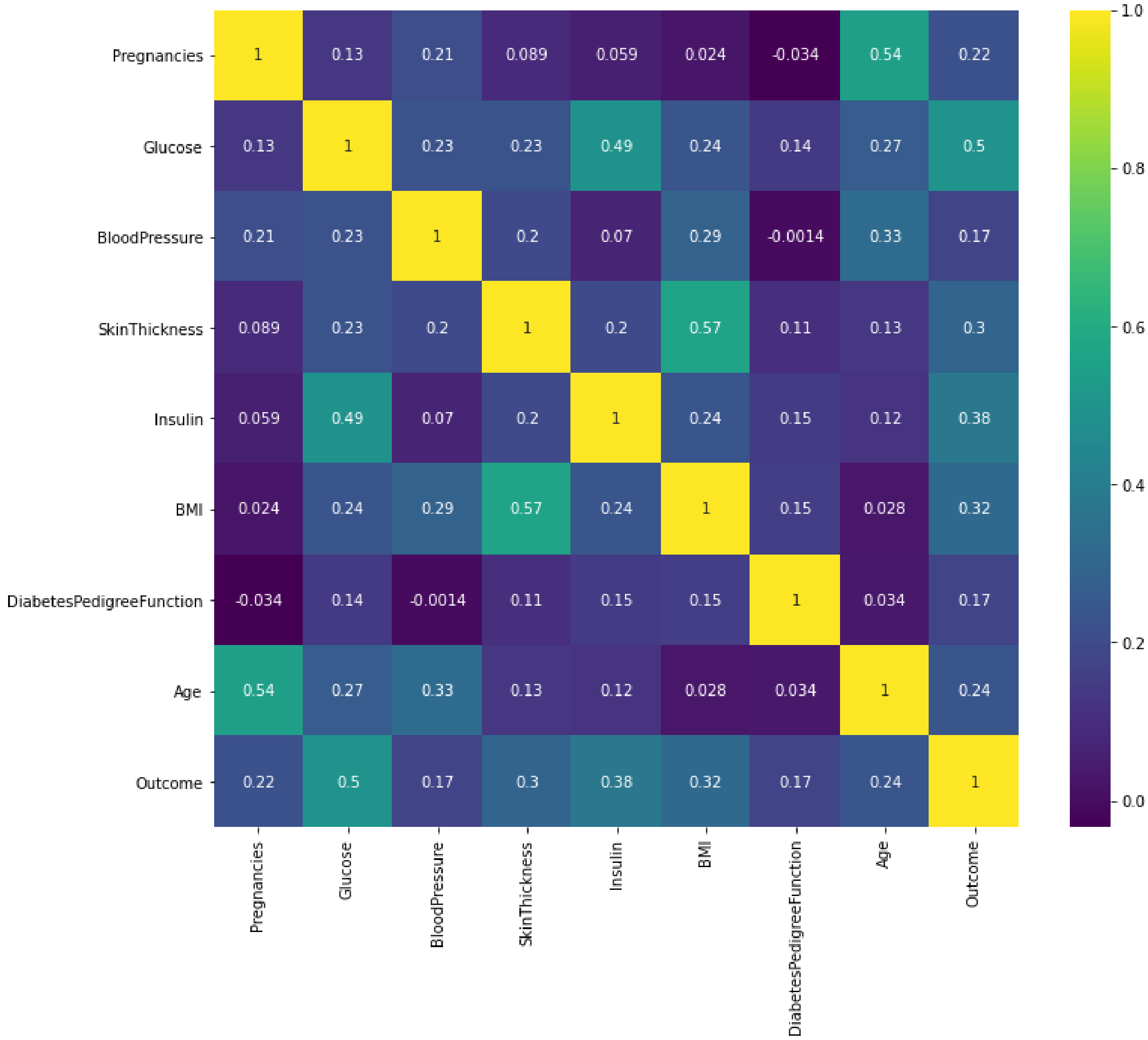


Heatmap of Clean Data

In [35]:

```
plt.figure(figsize=(12,10))
sns.heatmap(df_copy.corr(),annot=True,cmap = 'viridis')
```

Out[35]: <AxesSubplot:>



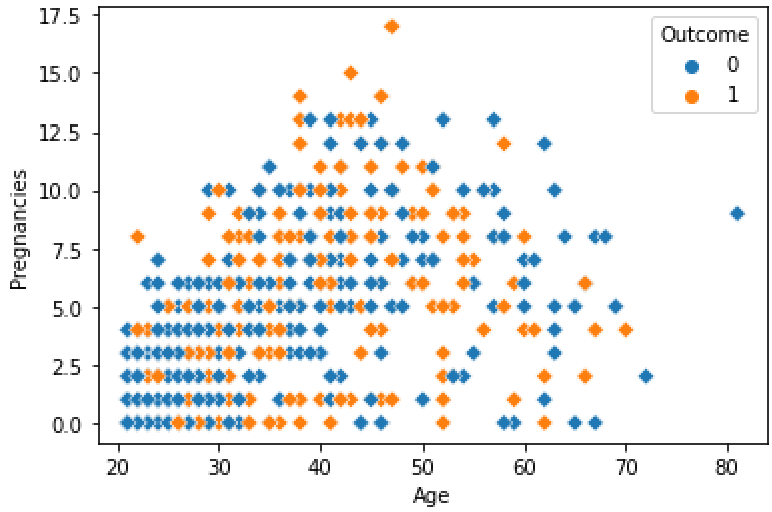
From the above heatmap we see a bit of correlation between some columns i.e.

- Age and Pregnancies = 0.54
- Glucose and insulin = 0.49
- SkinThickness and BMI = 0.57

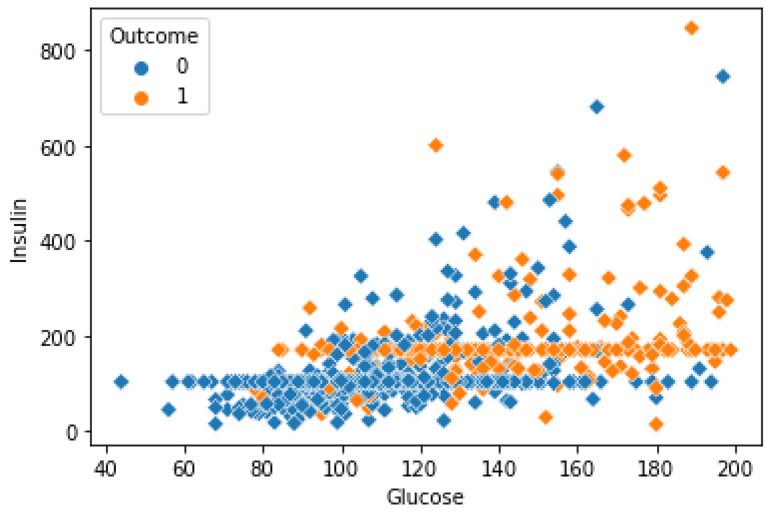
Let's create some scatter plots for above mentioned column pairs to understand the relationship among the top correlation values:

In [92]:
```
def sctr_plot(var1,var2):
    sns.scatterplot(x = var1,y = var2, data = df_copy,hue = 'Outcome',marker = 'D')
```

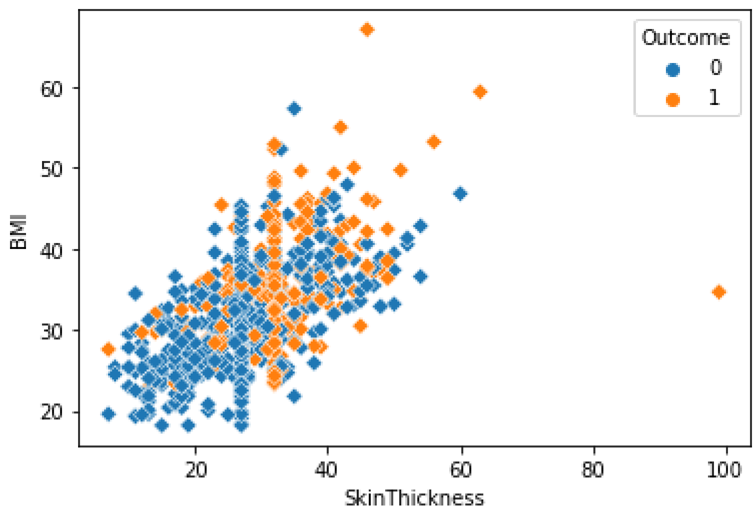
In [94]:
```
sctr_plot('Age','Pregnancies')
```



In [95]:
```
sctr_plot('Glucose','Insulin')
```



In [96]:
```
sctr_plot('SkinThickness','BMI')
```



Data Split for training and testing

In [36]:
```
X = df_copy.drop('Outcome',axis=1)
y=df_copy.Outcome
```

In [37]:
```
X.shape,y.shape
```

Out[37]: ((768, 8), (768,))

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
```

In [39]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[39]:
```
((537, 8), (231, 8), (537,), (231,))
```

Standardization

To bring the whole data at a same scale we'll perform satnadardization.

In [40]:
```
sc = StandardScaler()
```

In [41]:
```
X_train_scaled = sc.fit_transform(X_train)
```

In [42]:
```
X_test_scaled = sc.transform(X_test)
```

1. LogisticRegression Model

In [43]:
```
Log_model = LogisticRegression(max_iter=10000)
```

In [44]:
```
Log_model.fit(X_train_scaled,y_train)
```

Out[44]:
```
LogisticRegression(max_iter=10000)
```

In [45]:
```
log_pred = Log_model.predict(X_test_scaled)
```

In [46]:
```
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 [47]:
```
print(confusion_matrix(y_test,log_pred))
print('\n','Accuracy - ',accuracy_score(y_test,log_pred))
```

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

 Accuracy -  0.7489177489177489
```

2. RandomForest Classifier

In [48]:
```
rfc = RandomForestClassifier()
```

In [49]:
```
n_estimators = [75,100,125,150,200]
```

In [50]:
```
max_features = [4,5,6,7,8]
```

In [51]:
```
bootstrap=[True,False]
```

In [52]:
```
oob_score=[True,False]
```

In [53]:
```
param_grid = {'n_estimators':n_estimators,
              'max_features':max_features,
              'bootstrap':bootstrap,
              'oob_score':oob_score}
```

In [54]:
```
grid = GridSearchCV(rfc,param_grid)
```

In [55]:
```
grid.fit(X_train_scaled,y_train)
```

Out[55]:
```
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 [56]:
```
rfc_pred = grid.predict(X_test_scaled)
```

In [57]:
```
print(classification_report(y_test,rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.93      0.92       150
           1       0.87      0.80      0.83        81

    accuracy                           0.89       231
   macro avg       0.88      0.87      0.87       231
weighted avg       0.89      0.89      0.89       231
```

In [58]:
```
print(confusion_matrix(y_test,log_pred))
print('\n','Accuracy - ',accuracy_score(y_test,rfc_pred))
```

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

 Accuracy -  0.8874458874458875
```

In [59]:
```
grid.best_params_
```

Out[59]:
```
{'bootstrap': True, 'max_features': 5, 'n_estimators': 125, 'oob_score': True}
```

3. Support Vector Machine

In [60]:
```
svc = SVC()
```

In [61]:
```
param_grid = {'C':[0.01,0.1,1,10],'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
```

In [62]:
```
grid = GridSearchCV(svc,param_grid)
```

In [63]:
```
grid.fit(X_train_scaled,y_train)
```

Out[63]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.01, 0.1, 1, 10],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']})
```

In [64]:
```
svc_pred = grid.predict(X_test_scaled)
```

In [65]:
```
print(classification_report(y_test,svc_pred))
```

```
              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 [66]:
```
print(confusion_matrix(y_test,log_pred))
print('\n','Accuracy - ',accuracy_score(y_test,svc_pred))
```

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

 Accuracy -  0.8138528138528138
```

In [67]:
```
grid.best_params_
```

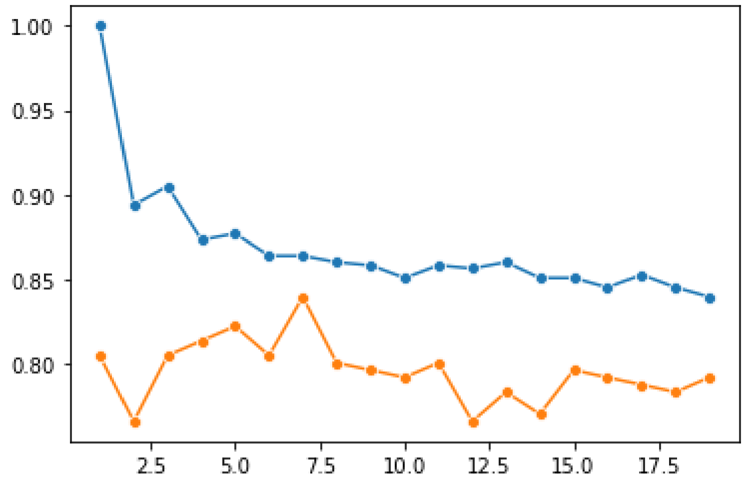
Out[67]:
```
{'C': 1, 'kernel': 'rbf'}
```

4. K Nearest Neighbour

In [68]:
```
train_score = []
test_score = []
for i in range (1,20):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train_scaled,y_train)
#     knn_pred = knn.predict(X_test)
    train_score.append(knn.score(X_train_scaled,y_train))
    test_score.append(knn.score(X_test_scaled,y_test))

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

Out[68]: <AxesSubplot:>



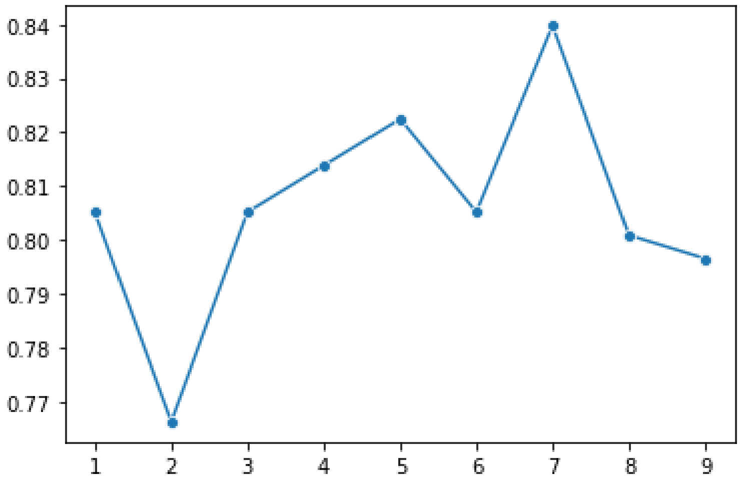
In [69]:
```
acc_score = []
for i in range (1,10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train_scaled,y_train)
    knn_pred = knn.predict(X_test_scaled)

    acc_score.append(accuracy_score(y_test,knn_pred))
print(max(acc_score))
```

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

```
0.8398268398268398
```

Out[69]: `<AxesSubplot:>`



From above results this could be concluded that n=7 gives the best results so we'll take n_neighbors = 7 for final model

In [70]:
```
final_knn_model = KNeighborsClassifier(n_neighbors=7)
```

In [71]:
```
final_knn_model.fit(X_train_scaled,y_train)
```

Out[71]: `KNeighborsClassifier(n_neighbors=7)`

In [72]:
```
knn_pred = final_knn_model.predict(X_test_scaled)
```

In [73]:
```
print(accuracy_score(y_test,knn_pred))
```

```
0.8398268398268398
```

In [74]:
```
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 [75]:
```
print(confusion_matrix(y_test,knn_pred))
```

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

In [ ]:

5. Decision Tree

In [76]:
```
dt = DecisionTreeClassifier(random_state=42)
```

In [77]:
```
param_grid = {'criterion' : ["gini", "entropy"],
              'min_samples_split' : [2,3,4,5],
```

```
                'max_features' : [4,5,6,7,8]
               }
```

In [78]:
```
grid_dt = GridSearchCV(dt,param_grid)
```

In [79]:
```
grid_dt.fit(X_train_scaled,y_train)
```

Out[79]:
```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_features': [4, 5, 6, 7, 8],
                         'min_samples_split': [2, 3, 4, 5]})
```

In [80]:
```
grid_dt.best_params_
```

Out[80]:
```
{'criterion': 'entropy', 'max_features': 6, 'min_samples_split': 3}
```

In [81]:
```
dt_pred = grid_dt.predict(X_test_scaled)
```

In [82]:
```
print(accuracy_score(y_test,dt_pred))
```

```
0.8614718614718615
```

In [83]:
```
print(classification_report(y_test,dt_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.92      0.90       150
           1       0.84      0.75      0.79        81

    accuracy                           0.86       231
   macro avg       0.85      0.84      0.84       231
weighted avg       0.86      0.86      0.86       231
```

In [104…
```
dt_cm = confusion_matrix(y_test,dt_pred)
```

In [105…
```
print(confusion_matrix(y_test,dt_pred))
```

```
[[138  12]
 [ 20  61]]
```

In [85]:
```
grid_dt.best_params_
```

Out[85]:
```
{'criterion': 'entropy', 'max_features': 6, 'min_samples_split': 3}
```

In [111…
```

```

In a Nutshell

Accuracy , Sensitivity and Specificity

In [124…
```
models = [
{
    'label': 'Logistic Regression',
    'model': LogisticRegression(),
},
{
    '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_featu
},
    {
    'label' : 'Random Forest Classifier',
    'model' : RandomForestClassifier(bootstrap= True, max_features= 6, n_estimators=
},
]
```

In [136…

```
accu = []
model_name= []
sensitivity = []
specificity = []
for m in models:
    model1 = m['model']
    model1.fit(X_train_scaled, y_train) # train the model
    pred = model1.predict(X_test_scaled) # predict the test data
    cm = confusion_matrix(y_test,pred)

    accu.append(accuracy_score(y_test,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_accuracy= pd.DataFrame(data=accu,index = model_name,columns=['Accuracy Sc
models_accuracy
```

Out[136…

	Accuracy Score
Logistic Regression	0.748918
KNeighbors Classifier	0.839827
Support Vector Classifier	0.813853
Decision Tress	0.861472
Random Forest Classifier	0.883117

In [182…

```
accu = []
model_name= []
sensitivity = []
specificity = []
for m in models:
    model1 = m['model']
    model1.fit(X_train_scaled, y_train) # train the model
    pred = model1.predict(X_test_scaled) # predict the test data
    cm = confusion_matrix(y_test,pred)


    accu.append(accuracy_score(y_test,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 = ['A
                                    columns=[model_name]).T
models_accu_sen_sp
```

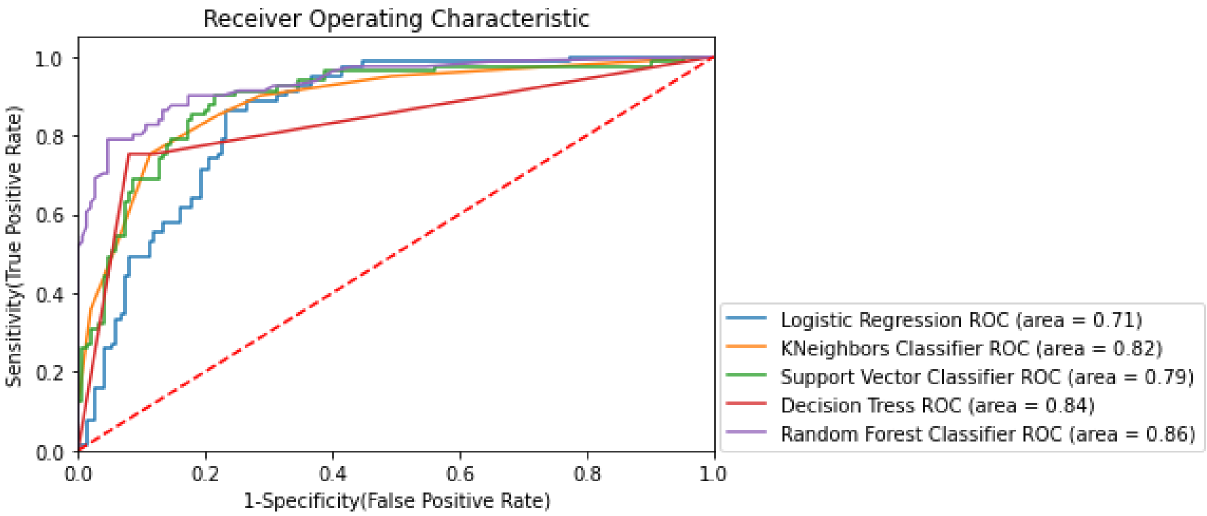
Out[182…

	Accuracy	Sensitivity	Specificity
Logistic Regression	0.748918	0.840000	0.580247
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
Random Forest Classifier	0.874459	0.920000	0.790123

Combined ROC Curve for all the models

In [88]:

```
# 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
```



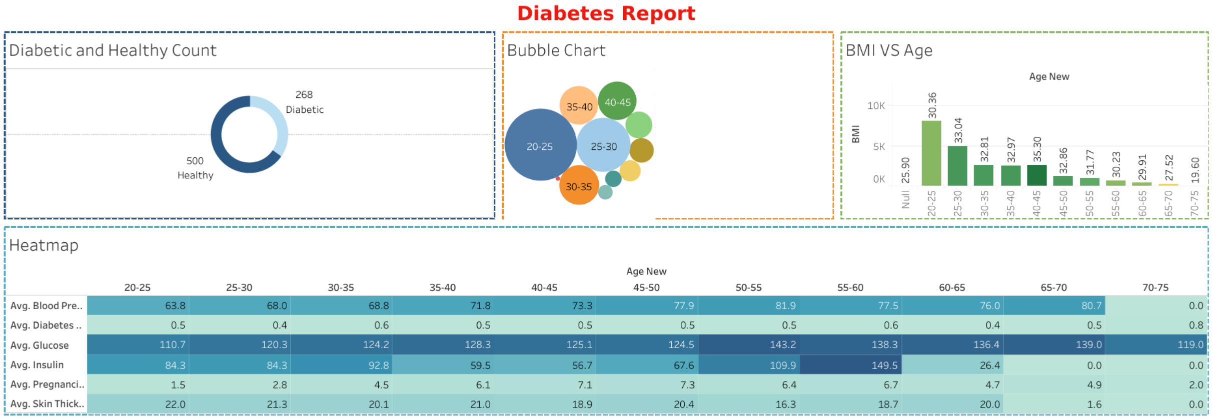
So the above curve gives the max area under the curve for Random Forest Classifier, hence this could be the best algorithm for the problem.

Tableau Visualisation

In [199…

```
image1 = PILImage.open('C:/Users/vipul/ML project 1/Tableau Dashboard/Dashboard 1.pn
image1
```

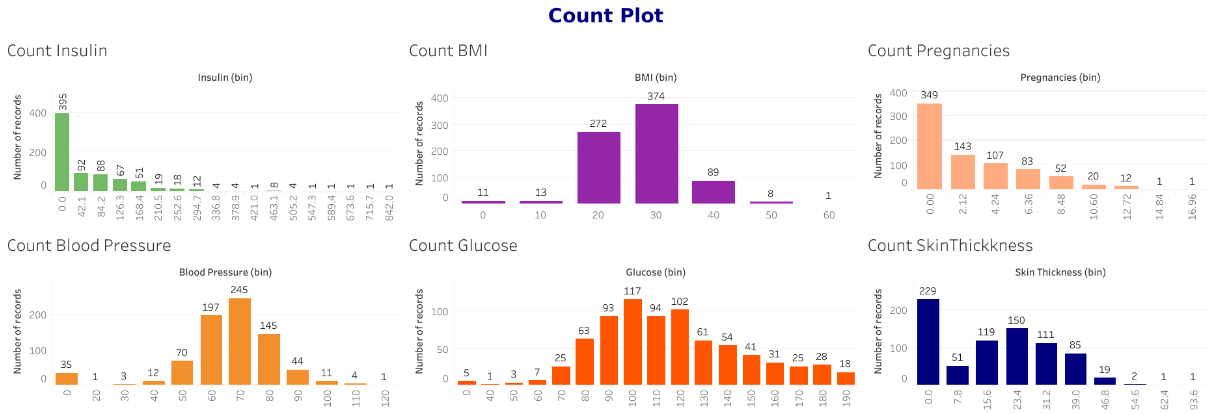
Out[199…



In [195…

```
image2 = PILImage.open('C:/Users/vipul/ML project 1/Tableau Dashboard/Dashboard 4.pn
image2
```

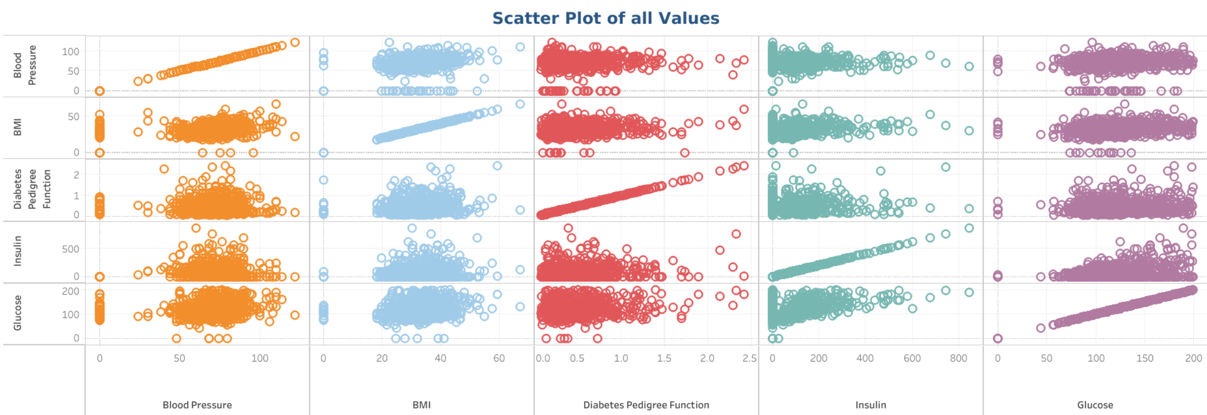
Out[195…



In [196…

```
image3 = PILImage.open('C:/Users/vipul/ML project 1/Tableau Dashboard/Scatter Plot a
image3
```

Out[196…

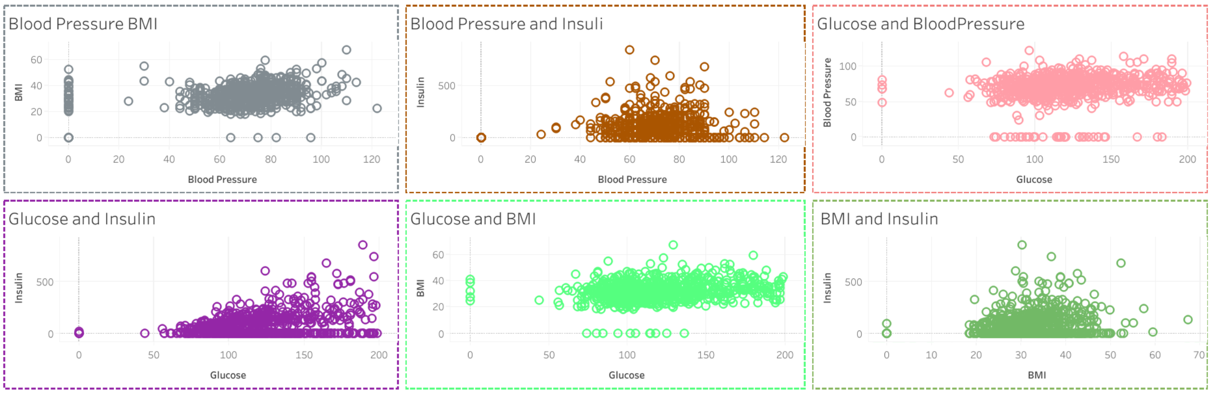


In [197…

```
image4 = PILImage.open('C:/Users/vipul/ML project 1/Tableau Dashboard/Scatter Plot.p
image4
```

Out[197…

Scatter Plot



In [ ]: