Harsh Seksaria

2048011

1-MDS

Date : *24-01-2021*

Topic : *Exploratory Data Analysis*

The following dataset contains data of diabetic patients recording different measures of biological factors which help in predicting whether or not the patient is prone to be affected from diabetes.

The dataset has been downloaded from Kaggle.com

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

In [2]:
```
fl = pd.read_csv('diabetes.csv')
fl.sample(3)
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
509	8	120	78	0	0	25.0	0.409	64	0
739	1	102	74	0	0	39.5	0.293	42	1
651	1	117	60	23	106	33.8	0.466	27	0

In [3]:
```
fl.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	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	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

The above output generated descriptive statistics summarizing data disribution on numerical values.

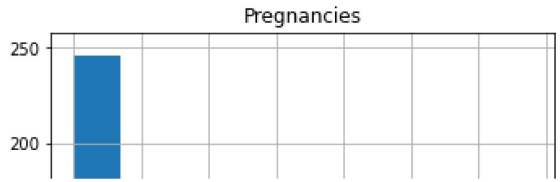
In [4]:
```
fl.info(verbose=True)
```

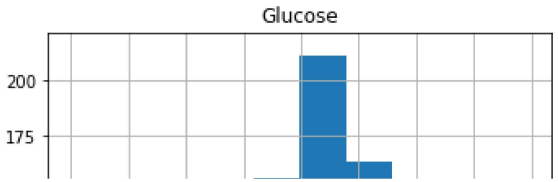
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   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
```

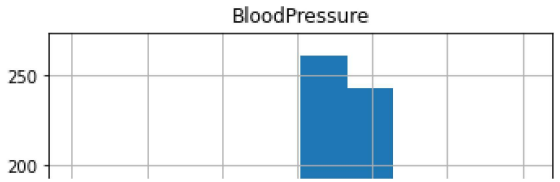
The above output gives us the information about the data types, non-null count, total columns, total records and memory usage.

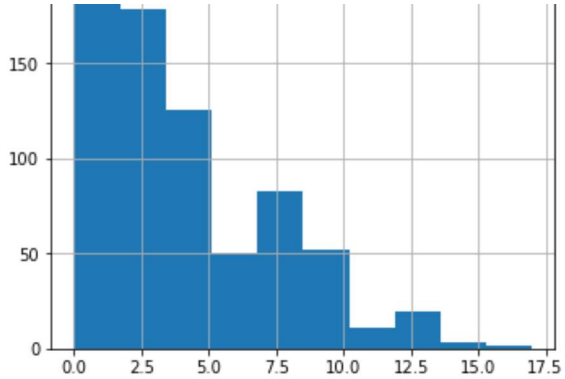
Data Distribution

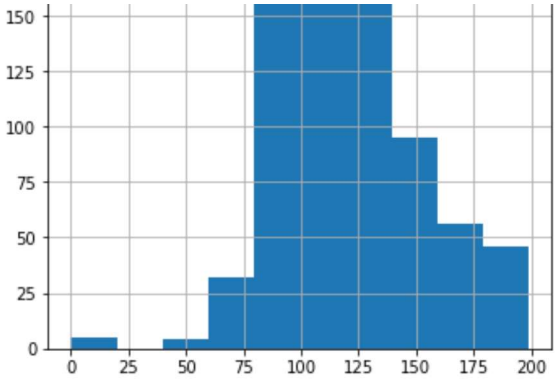
In [6]:
```
p = fl.hist(figsize=(20,20))
```

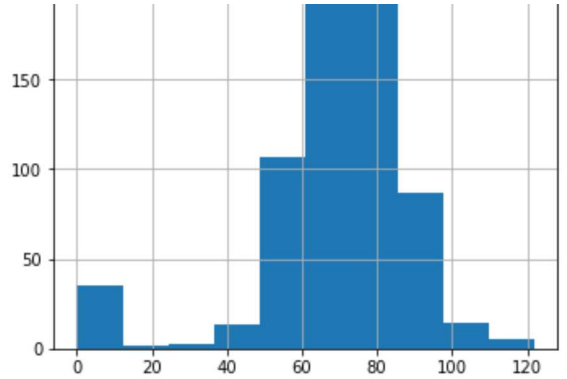


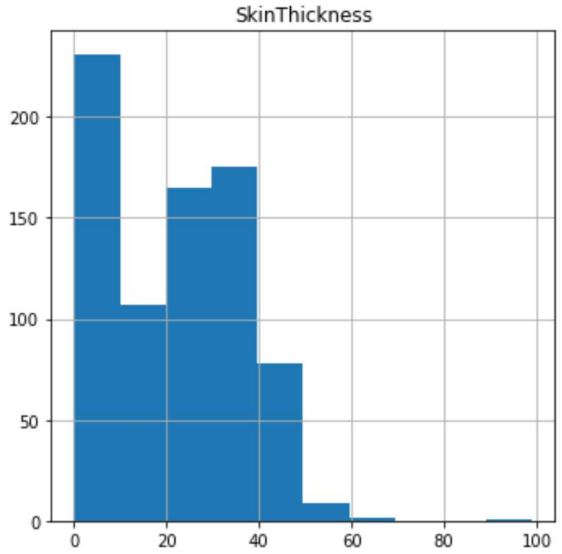


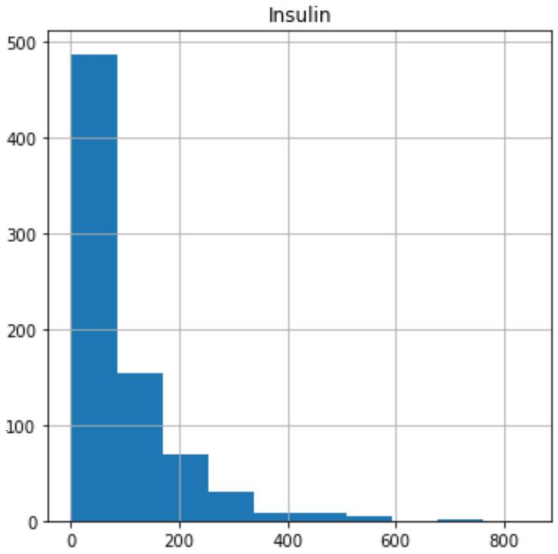


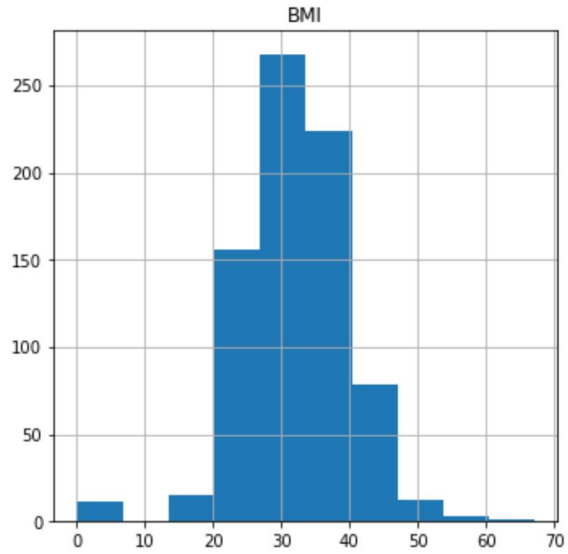

150
100
50
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

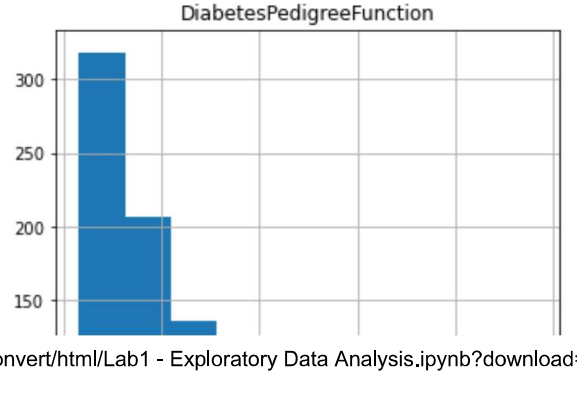

150
125
100
75
50
25
0
0
25
50
75
100
125
150
175
200

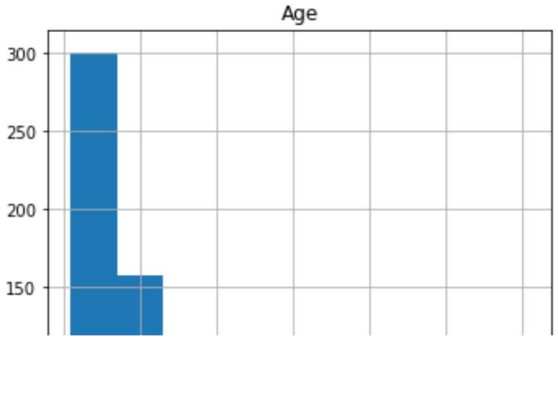

150
100
50
0
0
20
40
60
80
100
120

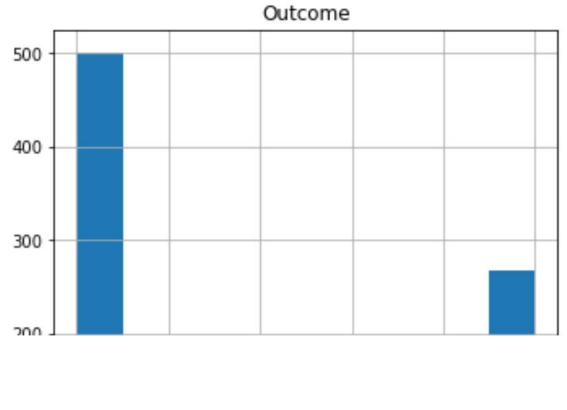

SkinThickness
200
150
100
50
0
0
20
40
60
80
100

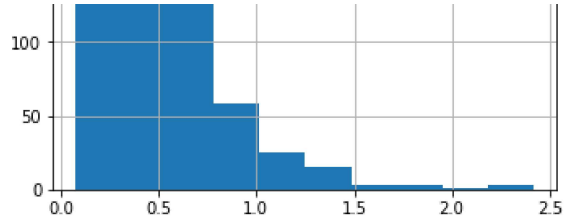

Insulin
500
400
300
200
100
0
0
200
400
600
800

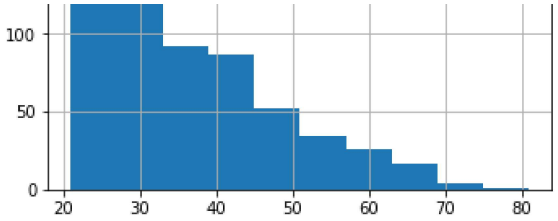

BMI
250
200
150
100
50
0
0
10
20
30
40
50
60
70

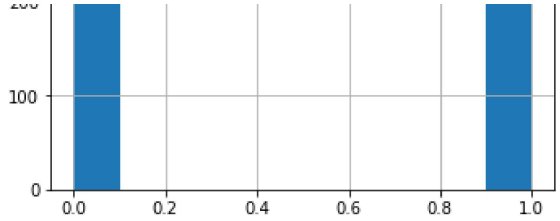

DiabetesPedigreeFunction
300
250
200
150


Age
300
250
200
150


Outcome
500
400
300







Missing Values Imputation

Wherever we have values as 0, it doesn't make any sense will cause problems in accurately understanding data. They are just like null. So, we first convert them to *null* and then do the missing value imputation.

In [7]:
```
fl[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']] = fl[['Glucose', 'BloodPres
fl.head(5)
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	NaN	33.6	0.627	50	1
1	1	85.0	66.0	29.0	NaN	26.6	0.351	31	0
2	8	183.0	64.0	NaN	NaN	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

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

Out[8]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We can clearly see that there were many 0 in our dataset, and now they have been converted to 0.

Now, we will fill these null values with appropriate methods to represent the best of data.

In [9]:
```
fl['Glucose'].fillna(fl['Glucose'].mean(), inplace=True)
fl['BloodPressure'].fillna(fl['BloodPressure'].mean(), inplace=True)
fl['SkinThickness'].fillna(fl['SkinThickness'].median(), inplace=True)
fl['Insulin'].fillna(fl['Insulin'].median(), inplace=True)
fl['BMI'].fillna(fl['BMI'].median(), inplace=True)
```

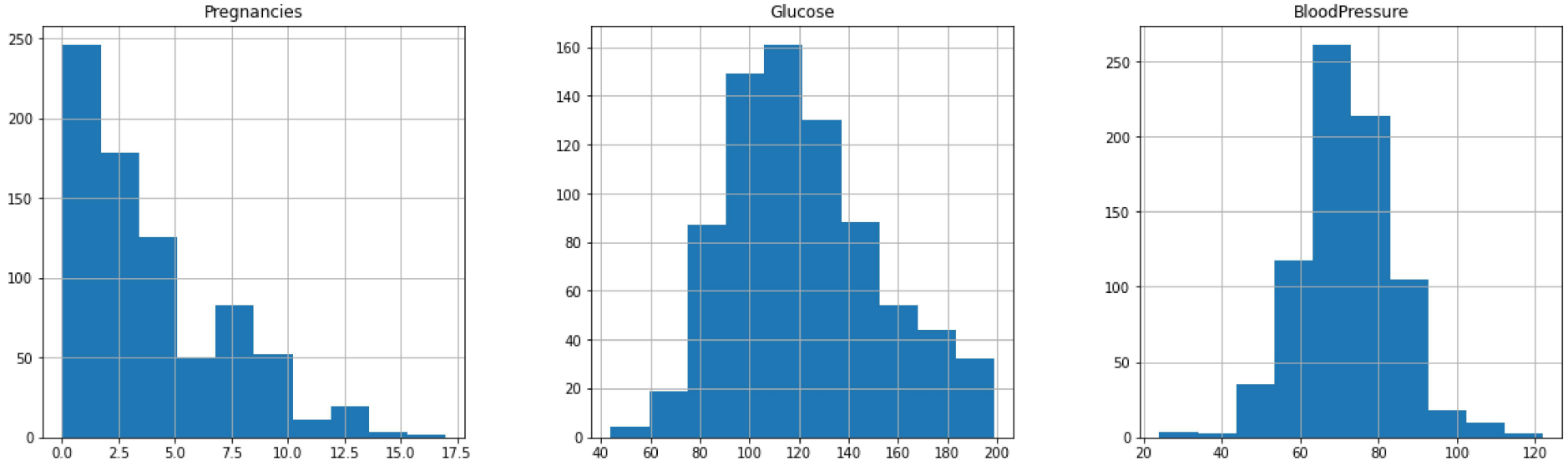
In [10]:
```
fl.isnull().sum()
```

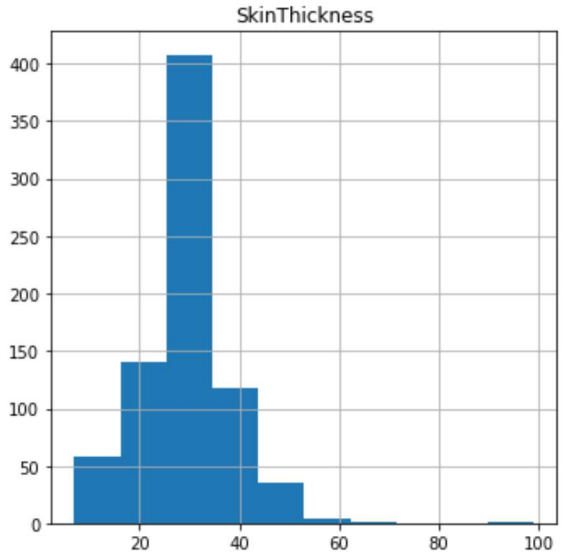
Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

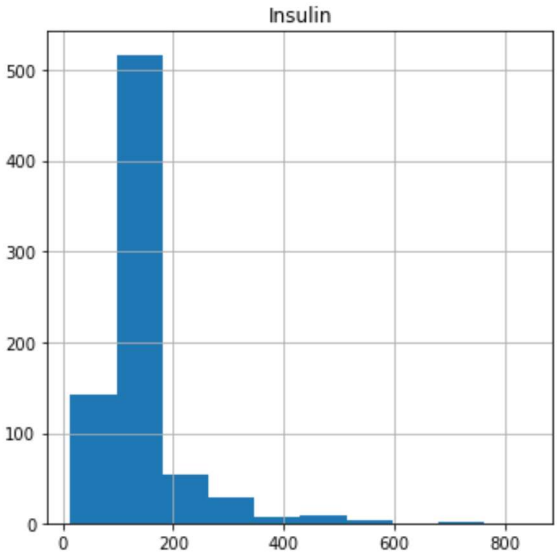
We first imputed null values by using mean/median whichever was suitable and then checked for null values to verify that there existed none.

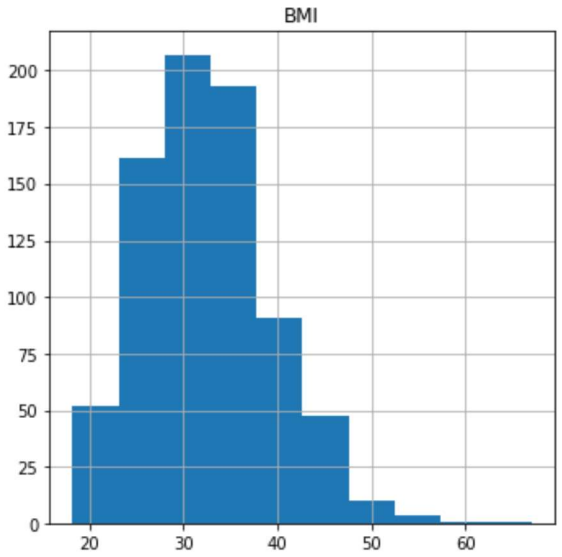
Let's see the histogram again.

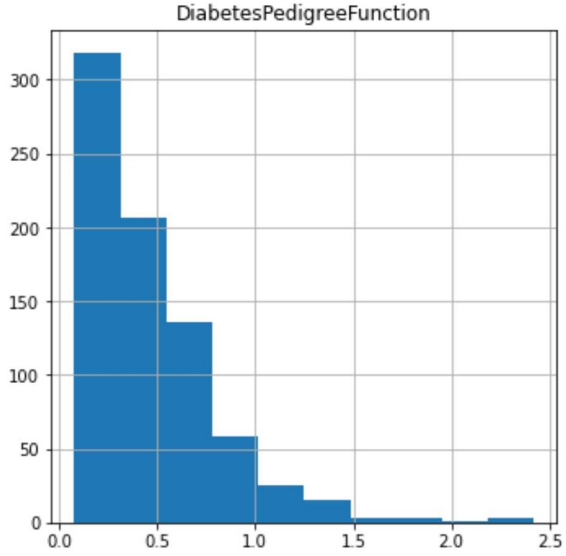
In [12]:
```
p = fl.hist(figsize=(20,20))
```

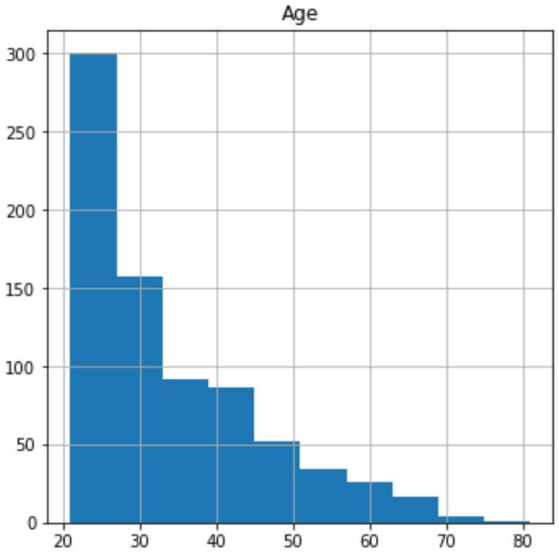


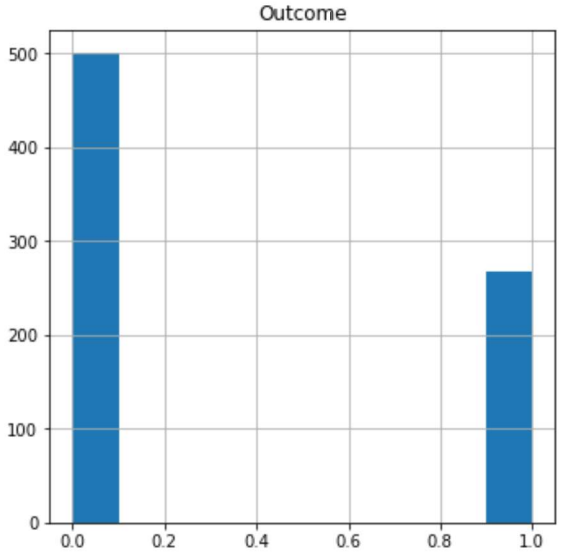








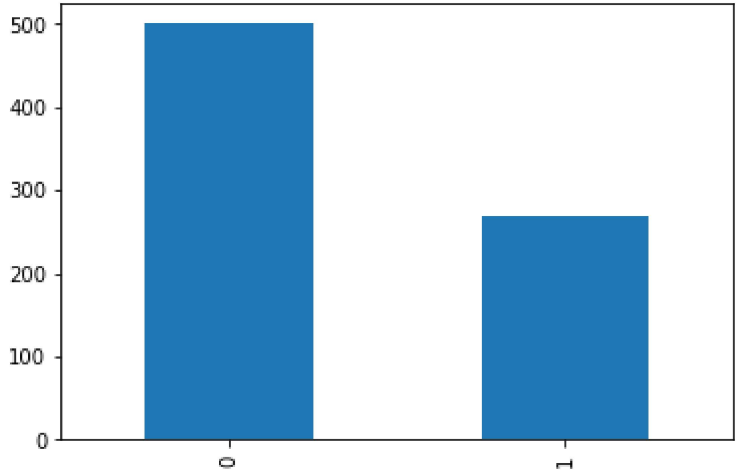




In [13]:
```
fl['Outcome'].value_counts().plot(kind='bar', y=['0', '1'])
var = fl['Outcome'].value_counts().to_dict()
```

```
print('Diabetic patients: ', str(var[1.0]))
print('Non-Diabetic patients: ', str(var[0.0]))
```

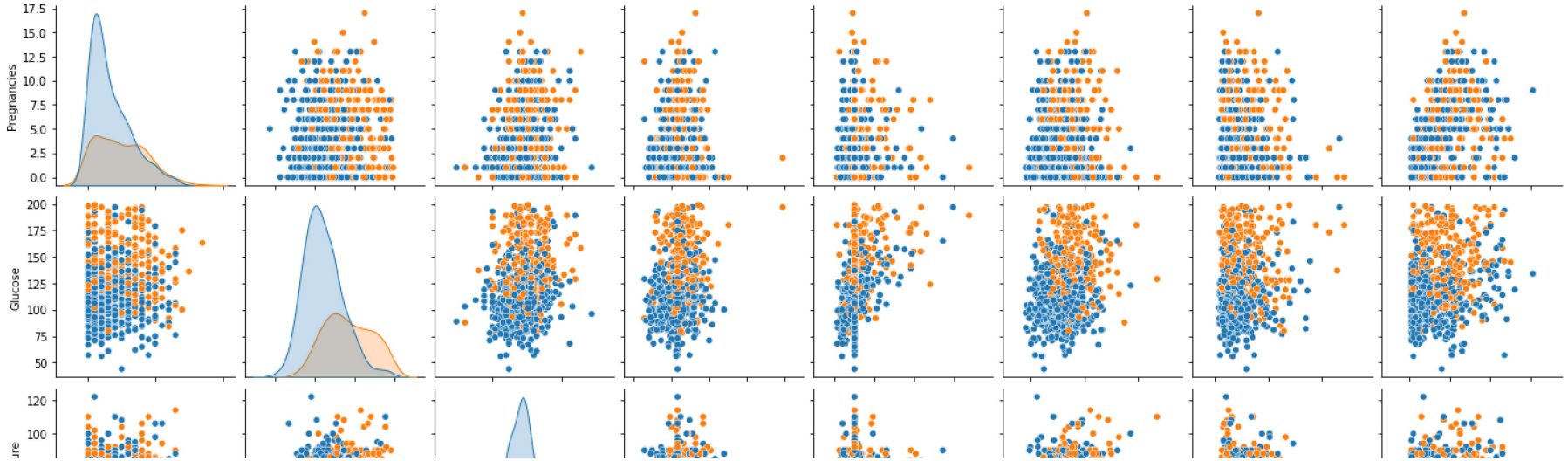
```
Diabetic patients:  268
Non-Diabetic patients:  500
```

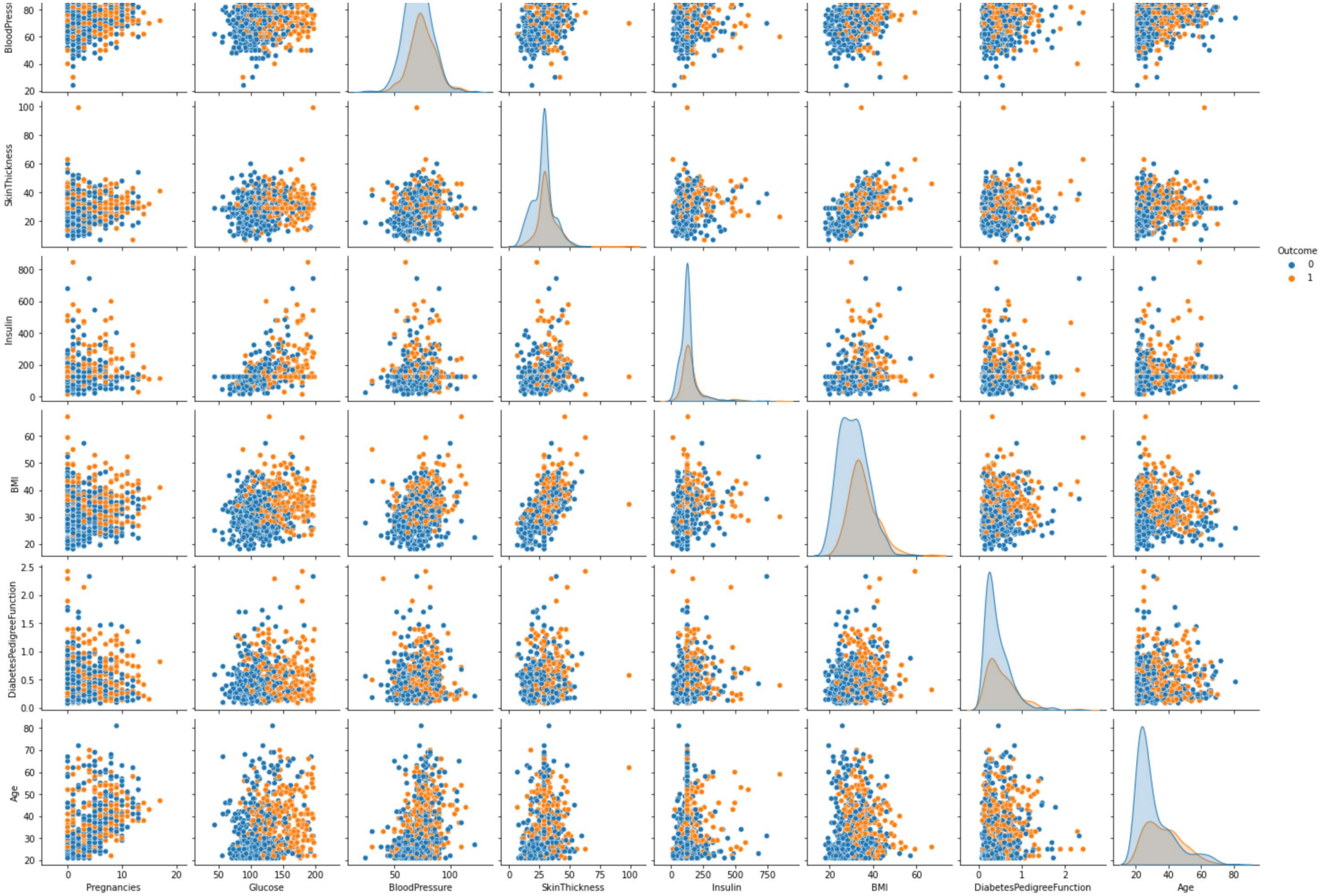


We can see that there are 500 patients who don't have diabetes but 268 do suffer from it.

Scatter Plot

In [14]:
```
p=sns.pairplot(fl, hue = 'Outcome')
```



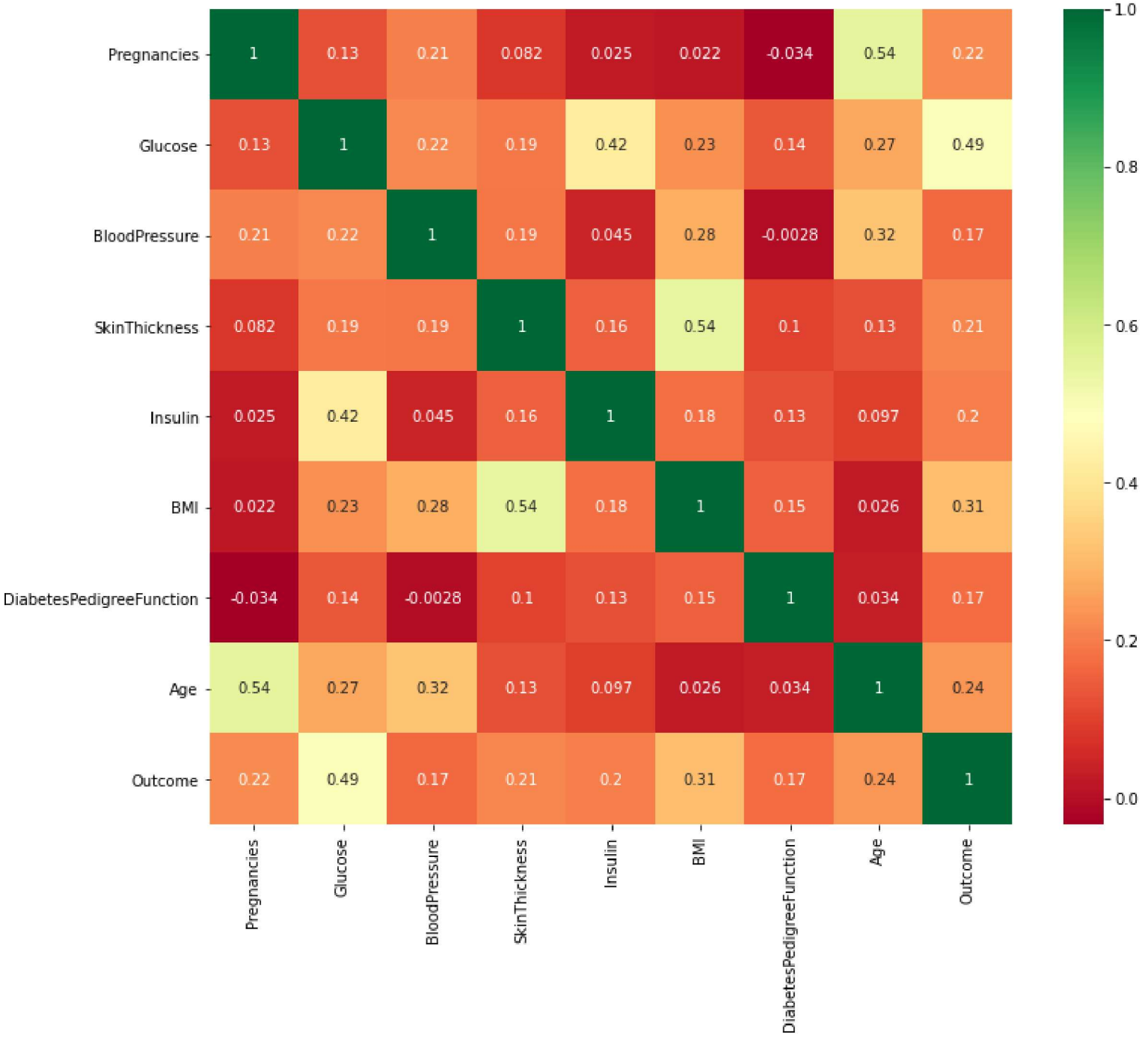


The above graphs show the relationship between two quantities - the measure of the strength of association between two variables.

Heat-map

A heat-map represents the information with the help of different colours, rather, different shades of a color.

In [15]:
```
plt.figure(figsize=(12,10))
p=sns.heatmap(fl.corr(), annot=True, cmap ='RdYlGn')
```

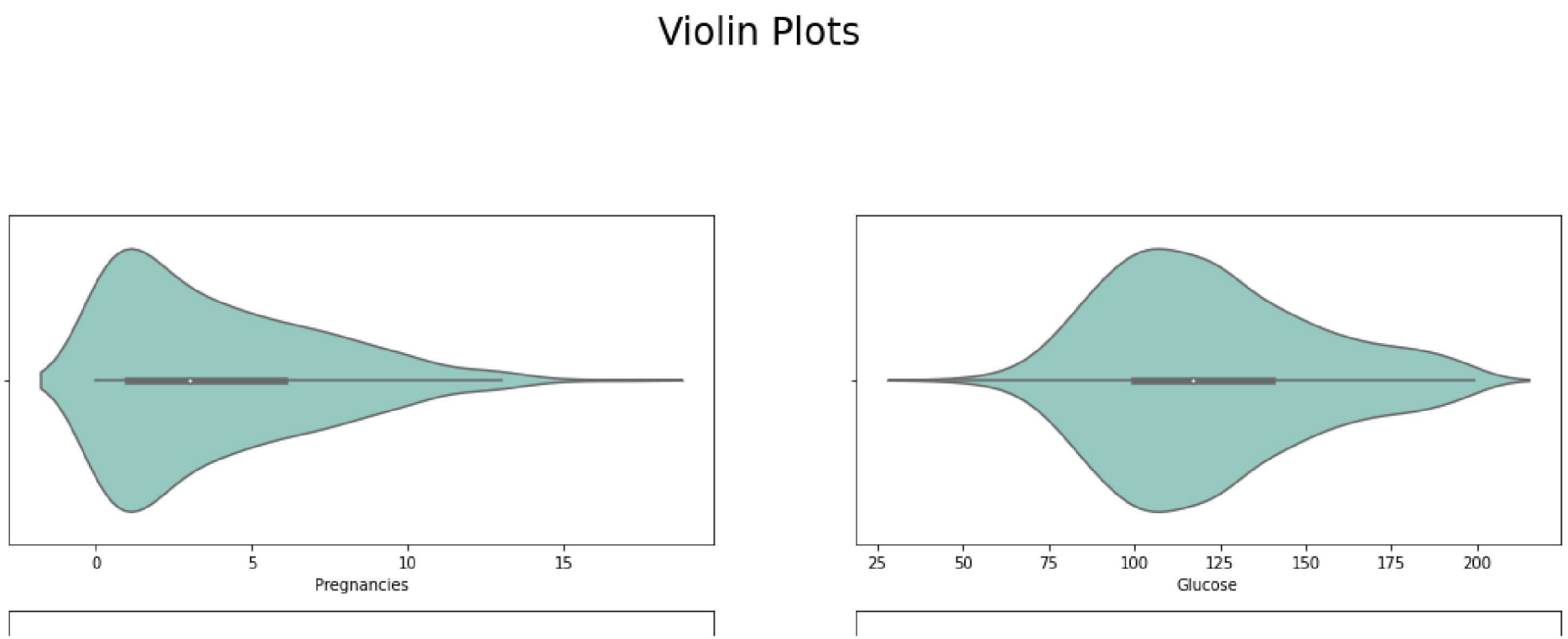

Pregnancies 1 0.13 0.21 0.082 0.025 0.022 -0.034 0.54 0.22
Glucose 0.13 1 0.22 0.19 0.42 0.23 0.14 0.27 0.49
BloodPressure 0.21 0.22 1 0.19 0.045 0.28 -0.0028 0.32 0.17
SkinThickness 0.082 0.19 0.19 1 0.16 0.54 0.1 0.13 0.21
Insulin 0.025 0.42 0.045 0.16 1 0.18 0.13 0.097 0.2
BMI 0.022 0.23 0.28 0.54 0.18 1 0.15 0.026 0.31
DiabetesPedigreeFunction -0.034 0.14 -0.0028 0.1 0.13 0.15 1 0.034 0.17
Age 0.54 0.27 0.32 0.13 0.097 0.026 0.034 1 0.24
Outcome 0.22 0.49 0.17 0.21 0.2 0.31 0.17 0.24 1
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

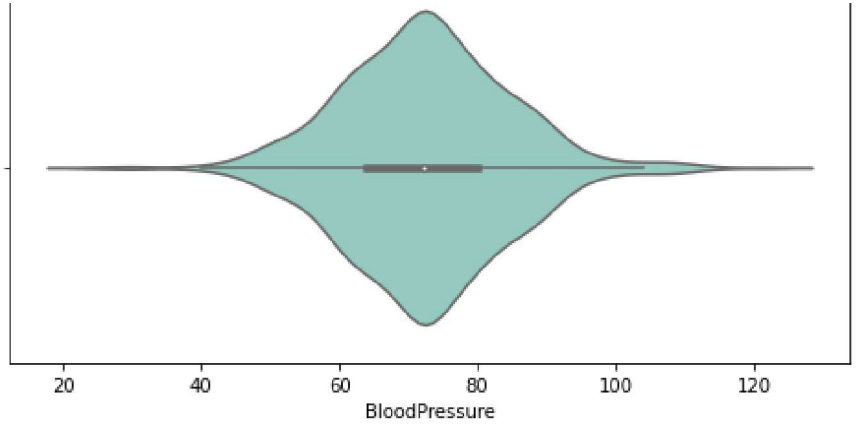
Violin Plot

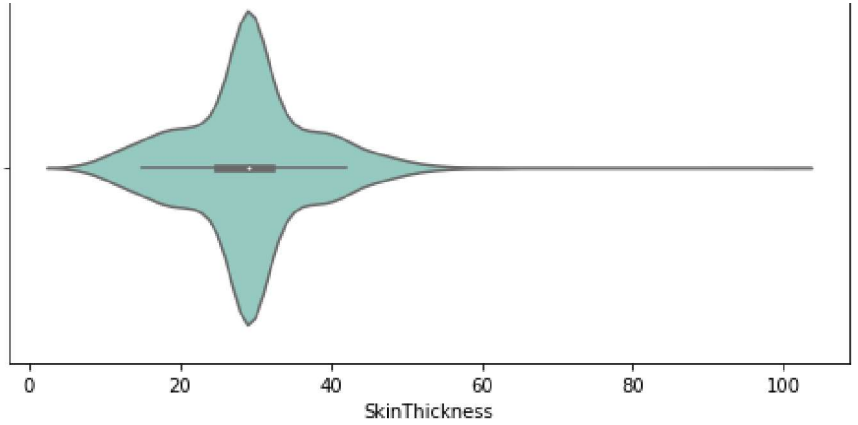
A violin plot is a method of plotting numeric data. It is similar to box plot with a rotated kernel density plot on each side. Violin plots are similar to box plots, except that they also show the probability density of the data at different values (in the simplest case this could be a histogram).

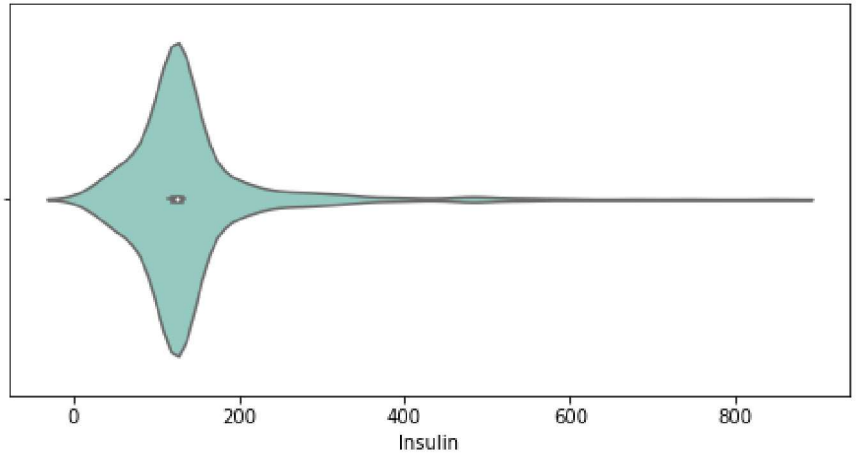
In [16]:
```
df=fl
```

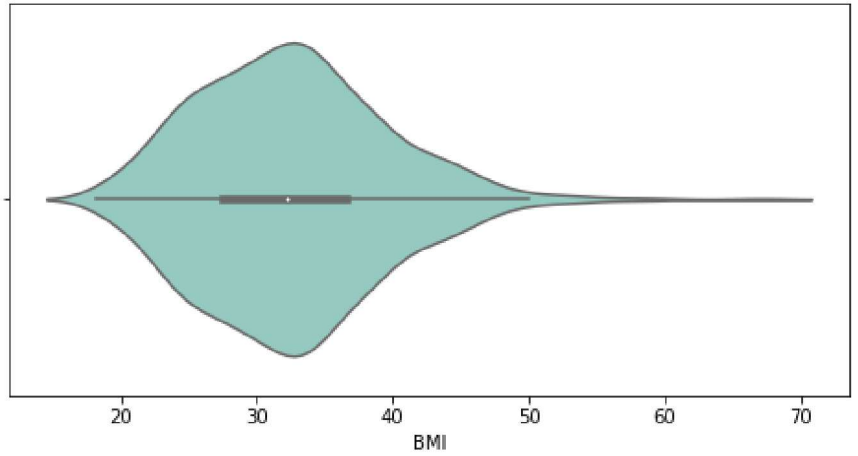
In [17]:
```
fig,ax = plt.subplots(nrows=4, ncols=2, figsize=(18,18))
plt.suptitle('Violin Plots',fontsize=24)
sns.violinplot(x="Pregnancies", data=df,ax=ax[0,0],palette='Set3')
sns.violinplot(x="Glucose", data=df,ax=ax[0,1],palette='Set3')
sns.violinplot (x ='BloodPressure', data=df, ax=ax[1,0], palette='Set3')
sns.violinplot(x='SkinThickness', data=df, ax=ax[1,1],palette='Set3')
sns.violinplot(x='Insulin', data=df, ax=ax[2,0], palette='Set3')
sns.violinplot(x='BMI', data=df, ax=ax[2,1],palette='Set3')
sns.violinplot(x='DiabetesPedigreeFunction', data=df, ax=ax[3,0],palette='Set3')
sns.violinplot(x='Age', data=df, ax=ax[3,1],palette='Set3')
plt.show()
```

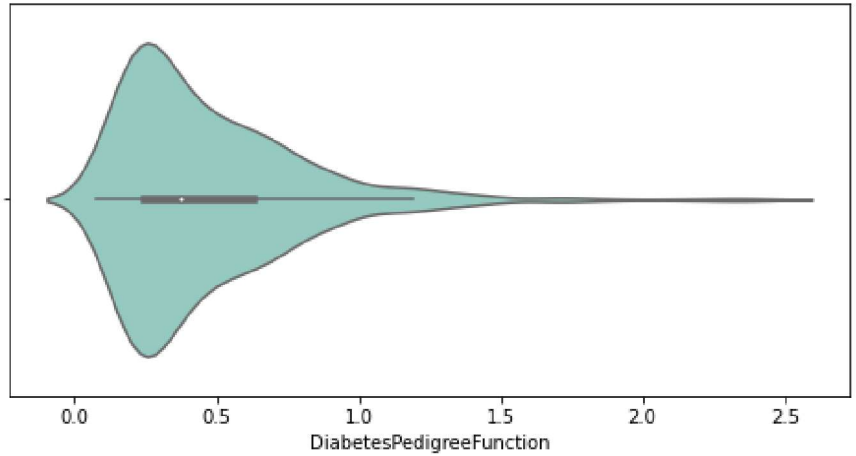


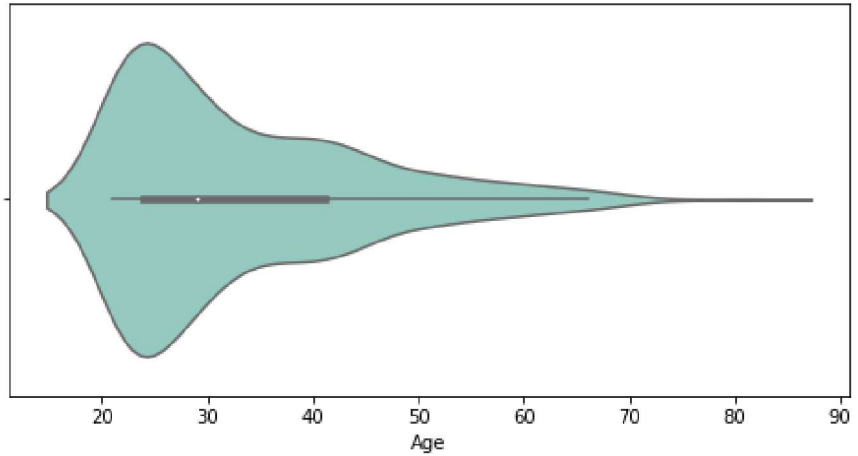

20
40
60
80
100
120
BloodPressure


0
20
40
60
80
100
SkinThickness


0
200
400
600
800
Insulin

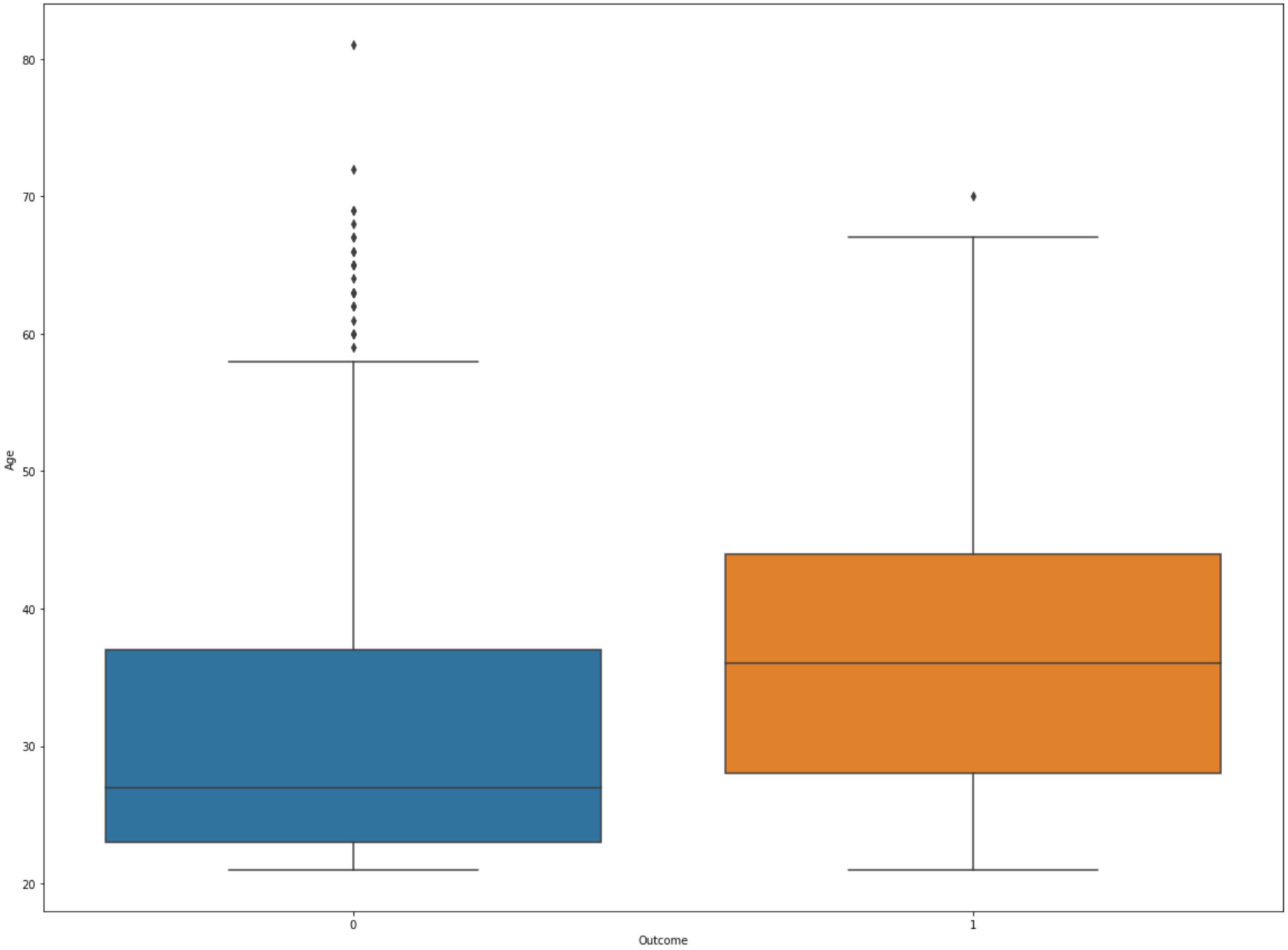

20
30
40
50
60
70
BMI


0.0
0.5
1.0
1.5
2.0
2.5
DiabetesPedigreeFunction


20
30
40
50
60
70
80
90
Age

In [18]:
```
plt.subplots(figsize=(20,15))
sns.boxplot(x='Outcome', y='Age', data=fl)
```

Out[18]:
```
<AxesSubplot:xlabel='Outcome', ylabel='Age'>
```


80
70
60
50
40
30
20
Age
0
1
Outcome

Conclusion

We saw the data with the help of different types of graphs which told us how the different variables behave under the influence of other variables.

Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.

The most widely used supervised learning algorithms are:

1. Support Vector Machines
2. Linear Regression
3. Logistic Regression
4. k-nearest neighbour
5. Decision trees
6. Naive Bayes
7. Linear Discriminant Analysis

The best suited algorithm for the model is the one that yeilds highest accuracy. Upon implementing and checking the algorithms, we can know which is most accurate and hence choose that one.

Now, for the dataset selected, supervised learning will be a suitable choice as it contains labels.