Project 2 - Healthcare - Data Science Capstone

We will be using Jupyter Notebook to execute this project and the detailed steps are given below with the code explanation

Description :

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 [124]:
```
# import required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [125]:
```
# Set working directory

import io
%cd "E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Healthcare - Diabetes"
```

```
E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Healthcare - Diabetes
```

Project Task: Week 1

Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose • BloodPressure • SkinThickness • Insulin • BMI

In [126]:
```
# import the dataset

healthcare = pd.read_csv('health care diabetes.csv')

healthcare.head()  # to show first 5 records
```

Out[126]:

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

In [127]:
```
healthcare.shape # to get number of rows and columns
```

Out[127]:
```
(768, 9)
```

In [128]:
```
healthcare.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 [129]:
```
healthcare.dtypes
```

Out[129]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [130]:
```
healthcare.describe()
```

Out[130]:

	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
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

In [131]:
```
# Lets check for missing values in the dataset

healthcare.isnull().sum()
```

Out[131]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Now according to our initial analysis the dataset has 768 rows and 9 columns. All the variables are of the type int64 except for BMI & DiabetesPedigreeFunction which are float64.

Here we can see that there are no missing values in the dataset. However, as mentioned in the problem statement for below columns a value of zero does not make sense and thus indicates missing value:

• Glucose • BloodPressure • SkinThickness • Insulin • BMI.

So we will first replace these 0 values by np.nan to treat them as missing and then we will use MICE to handle these missing data

In [132]:
```
# Replacing 0 values in the below given columns by np.nan

healthcare['Glucose'] = healthcare['Glucose'].replace({0: np.nan})
healthcare['BloodPressure'] = healthcare['BloodPressure'].replace({0: np.nan})
healthcare['SkinThickness'] = healthcare['SkinThickness'].replace({0: np.nan})
healthcare['BMI'] = healthcare['BMI'].replace({0: np.nan})
healthcare['Insulin'] = healthcare['Insulin'].replace({0: np.nan})
```

In [133]:
```
# Lets check for missing values in the dataset

healthcare.isnull().sum()
```

Out[133]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

So now we can see that there are some missing values in the data and we will handle these using MICE

In [134]:
```
from statsmodels.imputation import mice

healthcare = mice.MICEData(healthcare).data
```

In [135]:
```
 # Lets check for missing values in the dataset

healthcare.isnull().sum()
```

Out[135]:
```
Pregnancies               0
Glucose                   0
BloodPressure             0
SkinThickness             0
Insulin                   0
BMI                       0
DiabetesPedigreeFunction  0
Age                       0
Outcome                   0
dtype: int64
```

Here we can see that there are no missing values in the dataset.

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

We have already handled the missing data. However we will check if there are any columns which has 0 value.

In [136]:
```
# Lets explore the Glucose Column

healthcare['Glucose'].value_counts(ascending=False)
```

Out[136]:
```
100.0    17
99.0     17
122.0    17
111.0    14
106.0    14
         ..
182.0     1
65.0      1
178.0     1
67.0      1
56.0      1
Name: Glucose, Length: 135, dtype: int64
```

Above analysis does not give much details if there are any rows with Glucose = 0. So we will use below code to find if there are any rows that has Glucose = 0

In [137]:
```
healthcare[healthcare['Glucose']==0]
```

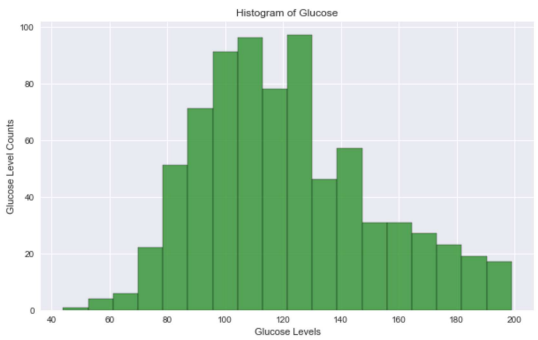
Out[137]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Here we can see that there no rows with Glucose = 0.

Now lets create a histogram to visualize the Glucose column

In [138]:
```
# Histogram of Glucose
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'Glucose', stat='count', color='forestgreen')
plt.xlabel('Glucose Levels')
plt.ylabel('Glucose Level Counts')
plt.title('Histogram of Glucose')
plt.show()
```



In [139]:
```
# Lets explore BloodPressure column

healthcare['BloodPressure'].value_counts(ascending=False)
```

Out[139]:
```
72.0     79
70.0     57
74.0     52
78.0     45
68.0     45
64.0     43
80.0     40
76.0     39
60.0     37
62.0     34
82.0     30
66.0     30
88.0     25
84.0     23
90.0     22
58.0     21
86.0     21
50.0     13
56.0     12
54.0     11
52.0     11
92.0      8
75.0      8
65.0      7
85.0      6
94.0      6
48.0      5
96.0      4
44.0      4
100.0     3
106.0     3
98.0      3
110.0     3
55.0      2
46.0      2
104.0     2
30.0      2
108.0     2
24.0      1
95.0      1
114.0     1
102.0     1
122.0     1
40.0      1
61.0      1
38.0      1
Name: BloodPressure, dtype: int64
```

In [140]:
```
healthcare[healthcare['BloodPressure']==0]
```

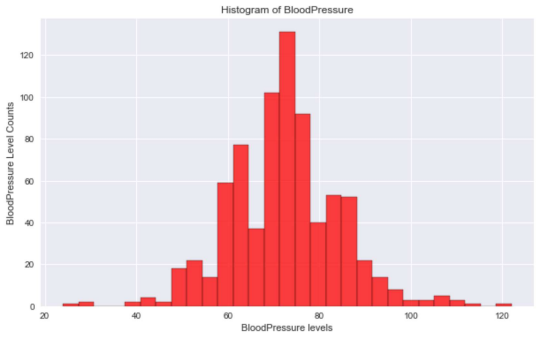
Out[140]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Here we can see that there no rows with BloodPressure = 0.

Now lets create a histogram to visualize the BloodPressure column

In [141]:
```
# Histogram of BloodPressure
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'BloodPressure', stat='count', color='red')
plt.xlabel('BloodPressure levels')
plt.ylabel('BloodPressure Level Counts')
plt.title('Histogram of BloodPressure')
plt.show()
```



In [142]:
```
# Lets explore SkinThickness column

healthcare['SkinThickness'].value_counts(ascending=False)
```

Out[142]:
```
29.0    244
32.0     31
30.0     27
27.0     23
23.0     22
18.0     20
28.0     20
33.0     20
31.0     19
39.0     18
19.0     18
22.0     16
37.0     16
25.0     16
40.0     16
26.0     16
41.0     15
35.0     15
17.0     14
36.0     14
15.0     14
20.0     13
24.0     12
13.0     11
42.0     11
21.0     10
46.0      8
34.0      8
12.0      7
38.0      7
16.0      6
45.0      6
14.0      6
11.0      6
43.0      6
10.0      5
44.0      5
47.0      4
48.0      4
49.0      3
50.0      3
54.0      2
7.0       2
52.0      2
8.0       2
51.0      1
99.0      1
60.0      1
56.0      1
63.0      1
Name: SkinThickness, dtype: int64
```

In [143]:
```
healthcare[healthcare['SkinThickness']==0]
```

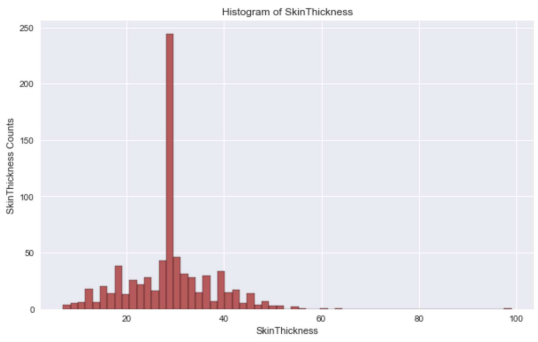
Out[143]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Here we can see that there no rows with SkinThickness = 0.

Now lets create a histogram to visualize the SkinThickness column

In [144]:
```
# Histogram of SkinThickness
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'SkinThickness', stat='count', color='brown')
plt.xlabel('SkinThickness')
plt.ylabel('SkinThickness Counts')
plt.title('Histogram of SkinThickness')
plt.show()
```



In [145]:
```
# Lets explore Insulin column

healthcare['Insulin'].value_counts(ascending=False)
```

Out[145]:
```
156.0    377
105.0     11
140.0      9
130.0      9
120.0      8
          ...
278.0      1
184.0      1
300.0      1
91.0       1
485.0      1
Name: Insulin, Length: 185, dtype: int64
```

In [146]:
```
healthcare[healthcare['Insulin']==0]
```

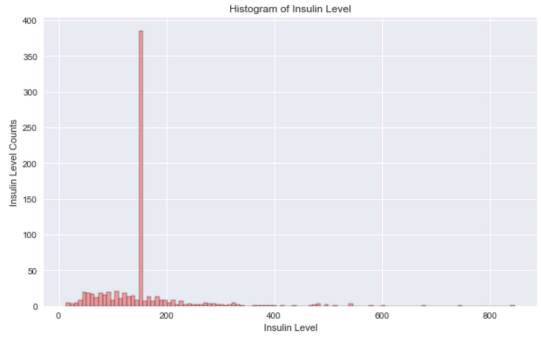
Out[146]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Here we can see that there no rows with Insulin = 0.

Now lets create a histogram to visualize the Insulin column

In [147]:
```
# Histogram of Insulin Level
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'Insulin', stat='count', color='lightcoral')
plt.xlabel('Insulin Level')
plt.ylabel('Insulin Level Counts')
plt.title('Histogram of Insulin Level')
plt.show()
```



In [148]:
```
# Lets explore BMI column

healthcare['BMI'].value_counts(ascending=False)
```

Out[148]:
```
32.5    17
32.0    13
31.6    12
31.2    12
32.4    10
        ..
19.3     1
49.3     1
19.4     1
20.0     1
40.1     1
Name: BMI, Length: 247, dtype: int64
```

In [149]:
```
healthcare[healthcare['BMI']==0]
```

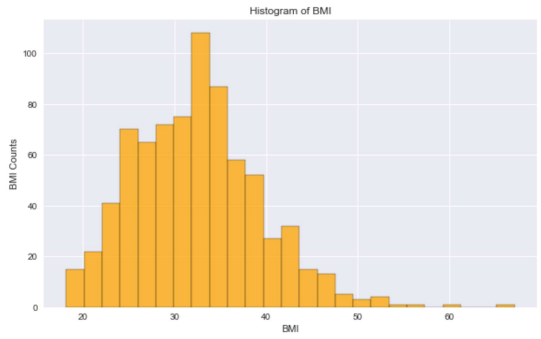
Out[149]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

Here we can see that there no rows with BMI = 0.

Now lets create a histogram to visualize the BMI column

In [150]:
```
# Histogram of BMI
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'BMI', stat='count', color='orange')
plt.xlabel('BMI')
plt.ylabel('BMI Counts')
plt.title('Histogram of BMI')
plt.show()
```



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.

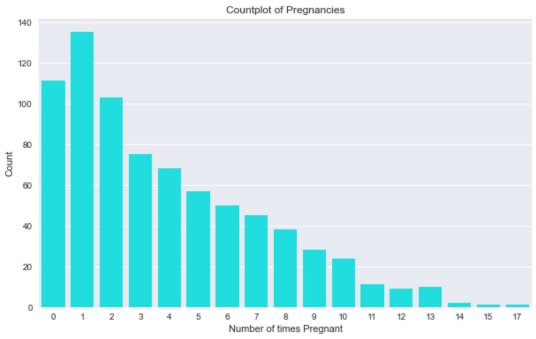
Here we will explore the other variables using a countplot

In [151]:
```
# Lets explore Pregnancies column

healthcare['Pregnancies'].value_counts(ascending=False)
```

Out[151]:
```
1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
Name: Pregnancies, dtype: int64
```

Now lets create a countplot to visualize the Pregnancies column

In [152]:
```
# Countplot of Pregnancies
plt.figure(figsize=(10,6))
sns.countplot(data=healthcare, x = 'Pregnancies', orient='v', color='cyan')
plt.xlabel('Number of times Pregnant')
plt.ylabel('Count')
plt.title('Countplot of Pregnancies')
plt.show()
```

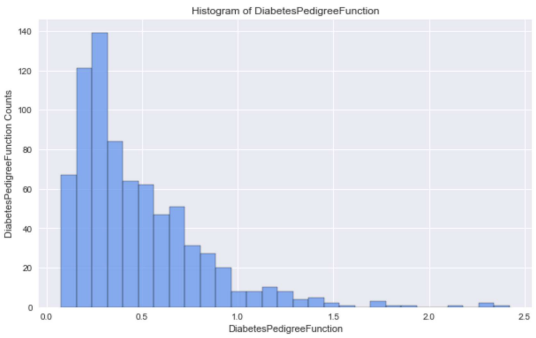


In [153]:
```
# Lets explore DiabetesPedigreeFunction column

healthcare['DiabetesPedigreeFunction'].value_counts(ascending=False)
```

Out[153]:
```
0.258    6
0.254    6
0.268    5
0.261    5
0.207    5
        ..
0.145    1
0.241    1
1.292    1
0.627    1
0.804    1
Name: DiabetesPedigreeFunction, Length: 517, dtype: int64
```

Now lets create a histogram to visualize the DiabetesPedigreeFunction column

In [154]:
```
# Histogram of DiabetesPedigreeFunction
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'DiabetesPedigreeFunction', stat='count', color='cornflowerblue')
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('DiabetesPedigreeFunction Counts')
plt.title('Histogram of DiabetesPedigreeFunction')
plt.show()
```

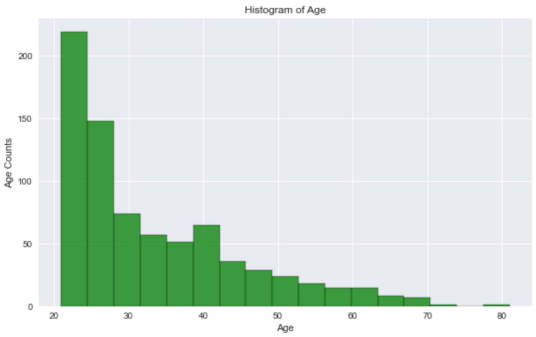


In [155]:
```
# Lets explore Age column

healthcare['Age'].value_counts(ascending=False)
```

Out[155]:
```
22    72
21    63
25    48
24    46
23    38
28    35
26    33
27    32
29    29
31    24
41    22
30    21
37    19
42    18
33    17
38    16
36    16
32    16
45    15
34    14
40    13
43    13
46    13
39    12
35    10
52     8
44     8
50     8
51     8
58     7
54     6
47     6
53     5
60     5
49     5
57     5
48     5
66     4
62     4
63     4
55     4
59     3
56     3
65     3
67     3
61     2
69     2
64     1
68     1
70     1
72     1
81     1
Name: Age, dtype: int64
```

Now lets create a histogram to visualize the Age column

In [156]:
```
# Histogram of Age
plt.figure(figsize=(10,6))
sns.histplot(data=healthcare, x = 'Age', stat='count', color='green')
plt.xlabel('Age')
plt.ylabel('Age Counts')
plt.title('Histogram of Age')
plt.show()
```

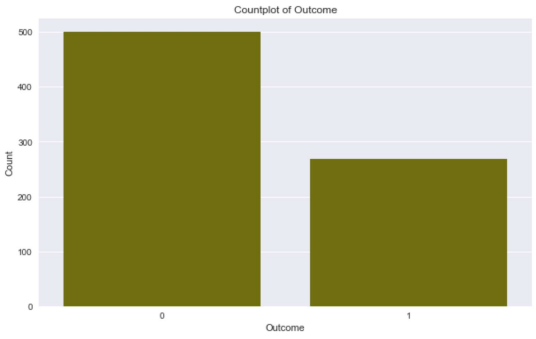


In [157]:
```
# Lets explore Outcome column

healthcare['Outcome'].value_counts()
```

Out[157]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

Now lets create a countplot to visualize the Outcome column

In [158]:
```
# Countplot of Outcome
plt.figure(figsize=(10,6))
sns.countplot(data=healthcare, x = 'Outcome', orient='v', color='olive')
plt.xlabel('Outcome')
plt.ylabel('Count')
plt.title('Countplot of Outcome')
plt.show()
```



From analysis of Week 1 Task we can say that the data is imbalanced as there are 500 counts for outcome 0 and 268 counts for outcome 1 which will somehow affect our ML models performance. Also only the Glucose column is somehow

close to a normal distribution copmpared to other columns.

Project Task: Week 2

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.

Here we have to separate the data according to whether the outcome is 0 or 1. Then we will visualize the variables accordingly

In [159]:
```
# Lets split the data between outcome = 0 & 1
positive_data = healthcare[healthcare['Outcome']==1]
negative_data = healthcare[healthcare['Outcome']==0]
```

In [160]:
```
# Lets check the shape of these two datasets
print(positive_data.shape)
print(negative_data.shape)
```

```
(268, 9)
(500, 9)
```

So we can see that there are 268 records with diabetes outcome as positive and 500 records with diabetes outcome as negative

In [161]:
```
positive_data.head() # first 5 records in the data
```

Out[161]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	156.0	33.6	0.627	50	1
2	8	183.0	64.0	29.0	156.0	23.3	0.672	32	1
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	1
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1

In [162]:
```
negative_data.head() # first 5 records in the data
```

Out[162]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	1	85.0	66.0	29.0	156.0	26.6	0.351	31	0
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
5	5	116.0	74.0	29.0	156.0	25.6	0.201	30	0
7	10	115.0	72.0	29.0	156.0	35.3	0.134	29	0
10	4	110.0	92.0	29.0	156.0	37.6	0.191	30	0

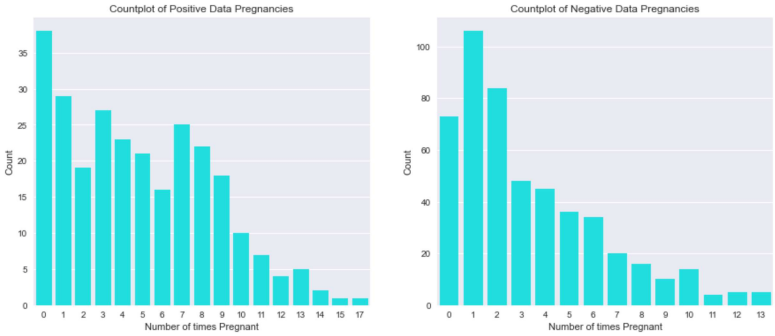
Now lets explore each variable according to positive and negative data and visualize them using histogram or countplot

In [163]:
```
# Lets explore Pregnancies column

print('positive_data pregnancies value counts are :')
print(positive_data['Pregnancies'].value_counts(ascending=False))
print('negative_data pregnancies value counts are :')
print(negative_data['Pregnancies'].value_counts(ascending=False))
```

```
positive_data pregnancies value counts are :
0     38
1     29
3     27
7     25
4     23
8     22
5     21
2     19
9     18
6     16
10    10
11     7
13     5
12     4
14     2
15     1
17     1
Name: Pregnancies, dtype: int64
negative_data pregnancies value counts are :
1     106
2      84
0      73
3      48
4      45
5      36
6      34
7      20
8      16
10     14
9      10
12      5
13      5
11      4
Name: Pregnancies, dtype: int64
```

Now we will visualize these data using countplot

In [164]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.countplot(data=positive_data, x = 'Pregnancies', orient='v', color='cyan')
plt.xlabel('Number of times Pregnant')
plt.ylabel('Count')
plt.title('Countplot of Positive Data Pregnancies')

# for negative data
plt.subplot(1,2,2)
sns.countplot(data=negative_data, x = 'Pregnancies', orient='v', color='cyan')
plt.xlabel('Number of times Pregnant')
plt.ylabel('Count')
plt.title('Countplot of Negative Data Pregnancies')

# show plots
plt.show()
```

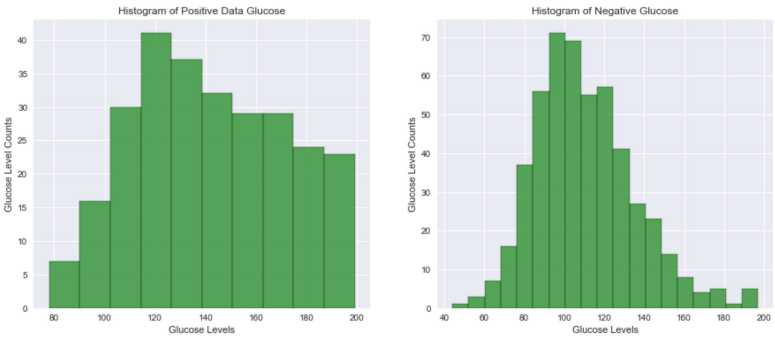


In [165]:
```
# Lets explore Glucose column

print('positive_data Glucose value counts are :')
print(positive_data['Glucose'].value_counts(ascending=False))
print('negative_data Glucose value counts are :')
print(negative_data['Glucose'].value_counts(ascending=False))
```

```
positive_data Glucose value counts are :
125.0    7
128.0    6
115.0    6
158.0    6
129.0    6
        ..
92.0     1
103.0    1
80.0     1
150.0    1
101.0    1
Name: Glucose, Length: 103, dtype: int64
negative_data Glucose value counts are :
99.0     17
100.0    13
106.0    13
122.0    12
111.0    11
         ..
173.0     1
161.0     1
175.0     1
193.0     1
56.0      1
Name: Glucose, Length: 110, dtype: int64
```

Now we will visualize these data using histogram

In [166]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'Glucose', stat='count', color='forestgreen')
plt.xlabel('Glucose Levels')
plt.ylabel('Glucose Level Counts')
plt.title('Histogram of Positive Data Glucose')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'Glucose', stat='count', color='forestgreen')
plt.xlabel('Glucose Levels')
plt.ylabel('Glucose Level Counts')
plt.title('Histogram of Negative Glucose')

# show plots
plt.show()
```

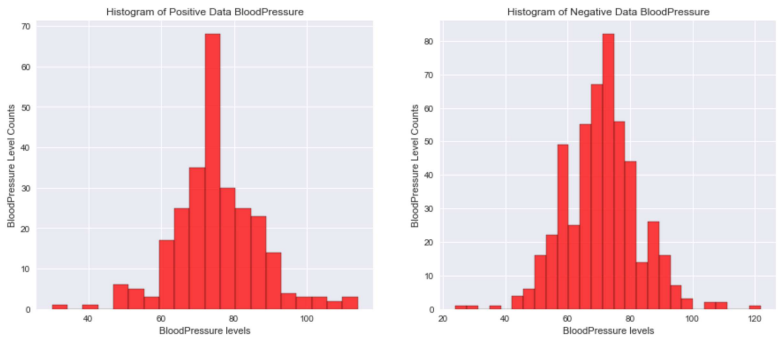


In [167]:
```
# Lets explore BloodPressure column

print('positive_data BloodPressure value counts are :')
print(positive_data['BloodPressure'].value_counts(ascending=False))
print('negative_data BloodPressure value counts are :')
print(negative_data['BloodPressure'].value_counts(ascending=False))
```

```
positive_data BloodPressure value counts are :
72.0     32
70.0     23
76.0     18
78.0     17
74.0     17
82.0     13
64.0     13
80.0     13
84.0     12
68.0     12
88.0     11
66.0     11
90.0     11
62.0     10
86.0      9
60.0      7
50.0      5
94.0      3
92.0      3
52.0      3
85.0      3
54.0      2
104.0     2
110.0     2
98.0      2
58.0      2
102.0     1
114.0     1
48.0      1
56.0      1
108.0     1
96.0      1
40.0      1
65.0      1
100.0     1
106.0     1
75.0      1
30.0      1
Name: BloodPressure, dtype: int64
negative_data BloodPressure value counts are :
72.0     47
74.0     35
70.0     34
68.0     33
64.0     30
60.0     30
78.0     28
80.0     27
62.0     24
76.0     21
58.0     19
66.0     19
82.0     17
88.0     14
86.0     12
84.0     11
90.0     11
56.0     11
54.0      9
52.0      8
50.0      8
75.0      7
65.0      6
92.0      5
44.0      4
48.0      4
85.0      3
94.0      3
96.0      3
106.0     2
46.0      2
100.0     2
55.0      2
61.0      1
110.0     1
24.0      1
95.0      1
122.0     1
30.0      1
98.0      1
108.0     1
38.0      1
Name: BloodPressure, dtype: int64
```

Now we will visualize these data using histogram

In [168]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'BloodPressure', stat='count', color='red')
plt.xlabel('BloodPressure levels')
plt.ylabel('BloodPressure Level Counts')
plt.title('Histogram of Positive Data BloodPressure')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'BloodPressure', stat='count', color='red')
plt.xlabel('BloodPressure levels')
plt.ylabel('BloodPressure Level Counts')
plt.title('Histogram of Negative Data BloodPressure')

# show plots
plt.show()
```

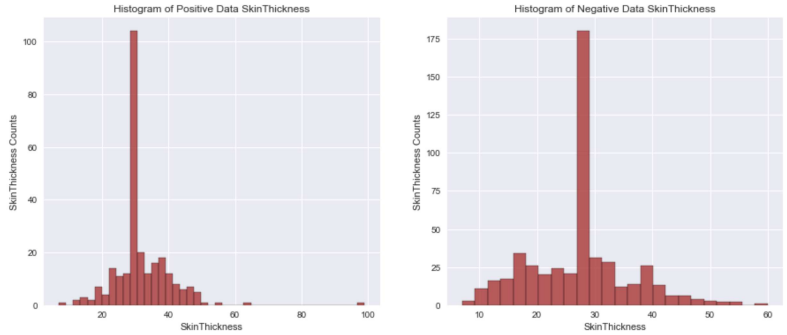


In [169]:
```
# Lets explore SkinThickness column

print('positive_data SkinThickness value counts are :')
print(positive_data['SkinThickness'].value_counts(ascending=False))
print('negative_data SkinThickness value counts are :')
print(negative_data['SkinThickness'].value_counts(ascending=False))
```

```
positive_data SkinThickness value counts are :
29.0    95
32.0    14
30.0     9
33.0     9
39.0     8
35.0     8
36.0     8
37.0     8
41.0     7
27.0     7
24.0     6
26.0     6
31.0     6
42.0     6
25.0     5
28.0     5
40.0     5
46.0     5
18.0     4
22.0     4
23.0     4
19.0     3
49.0     3
44.0     3
34.0     3
45.0     3
14.0     2
48.0     2
17.0     2
21.0     2
20.0     2
43.0     2
47.0     2
38.0     2
51.0     1
13.0     1
12.0     1
99.0     1
7.0      1
15.0     1
56.0     1
63.0     1
Name: SkinThickness, dtype: int64
negative_data SkinThickness value counts are :
29.0    149
23.0     18
30.0     18
32.0     17
18.0     16
27.0     16
19.0     15
28.0     15
31.0     13
15.0     13
17.0     12
22.0     12
33.0     11
40.0     11
25.0     11
20.0     11
13.0     10
26.0     10
39.0     10
21.0      8
37.0      8
41.0      8
35.0      7
16.0      6
12.0      6
11.0      6
24.0      6
36.0      6
42.0      5
34.0      5
38.0      5
10.0      5
43.0      4
14.0      4
45.0      3
46.0      3
50.0      3
52.0      2
8.0       2
47.0      2
44.0      2
48.0      2
54.0      2
7.0       1
60.0      1
Name: SkinThickness, dtype: int64
```

Now we will visualize these data using histogram

In [170]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'SkinThickness', stat='count', color='brown')
plt.xlabel('SkinThickness')
plt.ylabel('SkinThickness Counts')
plt.title('Histogram of Positive Data SkinThickness')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'SkinThickness', stat='count', color='brown')
plt.xlabel('SkinThickness')
plt.ylabel('SkinThickness Counts')
plt.title('Histogram of Negative Data SkinThickness')

# show plots
plt.show()
```

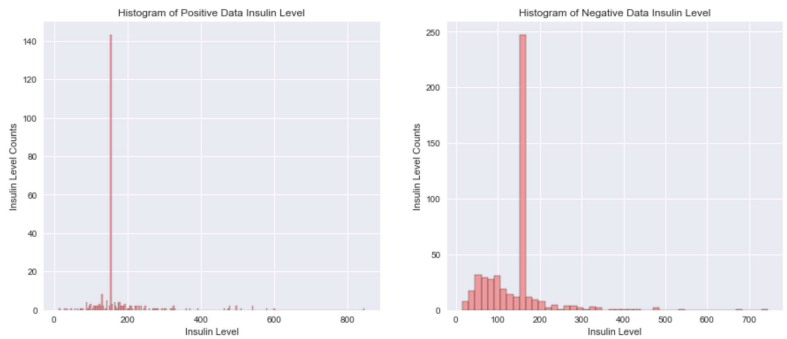


In [171]:
```
# Lets explore Insulin column

print('positive_data Insulin value counts are :')
print(positive_data['Insulin'].value_counts(ascending=False))
print('negative_data Insulin value counts are :')
print(negative_data['Insulin'].value_counts(ascending=False))
```

```
positive_data Insulin value counts are :
156.0    141
130.0      6
180.0      4
175.0      3
165.0      2
        ...
70.0       1
465.0      1
274.0      1
105.0      1
176.0      1
Name: Insulin, Length: 92, dtype: int64
negative_data Insulin value counts are :
156.0    236
105.0     10
140.0      8
94.0       7
100.0      6
        ...
270.0      1
272.0      1
43.0       1
188.0      1
61.0       1
Name: Insulin, Length: 137, dtype: int64
```

Now we will visualize these data using histogram

In [172]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'Insulin', stat='count', color='lightcoral')
plt.xlabel('Insulin Level')
plt.ylabel('Insulin Level Counts')
plt.title('Histogram of Positive Data Insulin Level')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'Insulin', stat='count', color='lightcoral')
plt.xlabel('Insulin Level')
plt.ylabel('Insulin Level Counts')
plt.title('Histogram of Negative Data Insulin Level')

# show plots
plt.show()
```

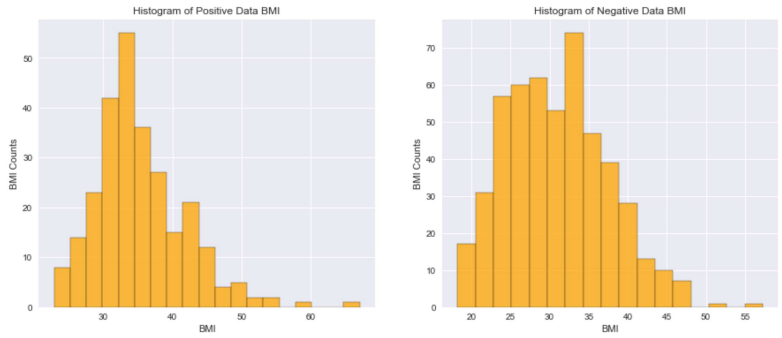


In [173]:
```
# Lets explore BMI column

print('positive_data BMI value counts are :')
print(positive_data['BMI'].value_counts(ascending=False))
print('negative_data BMI value counts are :')
print(negative_data['BMI'].value_counts(ascending=False))
```

```
positive_data BMI value counts are :
32.9    8
31.6    7
33.3    6
31.2    5
30.5    5
       ..
40.0    1
28.3    1
22.9    1
37.2    1
47.9    1
Name: BMI, Length: 147, dtype: int64
negative_data BMI value counts are :
32.5    13
32.0     8
33.2     7
30.8     7
27.8     7
        ..
22.7     1
26.9     1
22.9     1
24.9     1
19.9     1
Name: BMI, Length: 209, dtype: int64
```

Now we will visualize these data using histogram

In [174]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'BMI', stat='count', color='orange')
plt.xlabel('BMI')
plt.ylabel('BMI Counts')
plt.title('Histogram of Positive Data BMI')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'BMI', stat='count', color='orange')
plt.xlabel('BMI')
plt.ylabel('BMI Counts')
plt.title('Histogram of Negative Data BMI')

# show plots
plt.show()
```

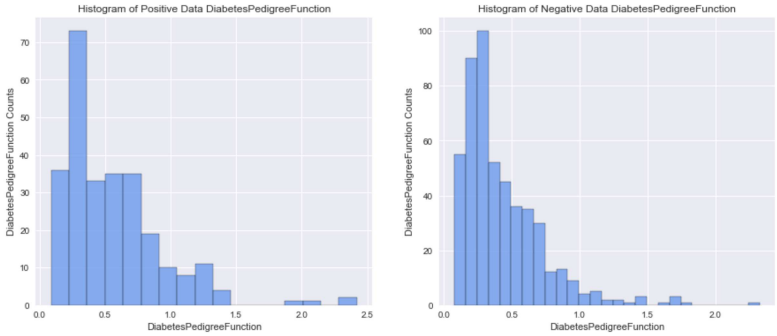


In [175]:
```
# Lets explore DiabetesPedigreeFunction column

print('positive_data DiabetesPedigreeFunction value counts are :')
print(positive_data['DiabetesPedigreeFunction'].value_counts(ascending=False))
print('negative_data DiabetesPedigreeFunction value counts are :')
print(negative_data['DiabetesPedigreeFunction'].value_counts(ascending=False))
```

```
positive_data DiabetesPedigreeFunction value counts are :
0.254    4
0.258    3
0.328    2
0.302    2
0.542    2
        ..
0.422    1
1.258    1
0.293    1
0.711    1
0.358    1
Name: DiabetesPedigreeFunction, Length: 231, dtype: int64
negative_data DiabetesPedigreeFunction value counts are :
0.207    5
0.284    4
0.190    4
0.237    4
0.167    4
        ..
0.182    1
0.158    1
0.640    1
0.293    1
0.365    1
Name: DiabetesPedigreeFunction, Length: 372, dtype: int64
```

Now we will visualize these data using histogram

In [176]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'DiabetesPedigreeFunction', stat='count', color='cornflowerblue')
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('DiabetesPedigreeFunction Counts')
plt.title('Histogram of Positive Data DiabetesPedigreeFunction')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'DiabetesPedigreeFunction', stat='count', color='cornflowerblue')
plt.xlabel('DiabetesPedigreeFunction')
plt.ylabel('DiabetesPedigreeFunction Counts')
plt.title('Histogram of Negative Data DiabetesPedigreeFunction')

# show plots
plt.show()
```

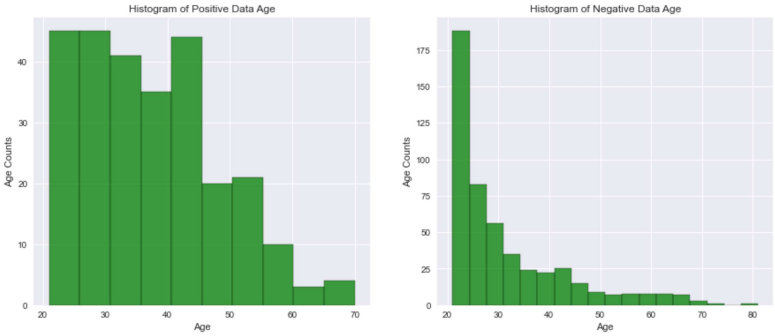


In [177]:
```
# Lets explore Age column

print('positive_data Age value counts are :')
print(positive_data['Age'].value_counts(ascending=False))
print('negative_data Age value counts are :')
print(negative_data['Age'].value_counts(ascending=False))
```

```
positive_data Age value counts are :
25    14
29    13
41    13
31    13
43    11
22    11
38    10
36    10
33    10
28    10
32     9
45     8
27     8
26     8
24     8
52     7
42     7
23     7
46     7
30     6
37     6
40     6
51     5
50     5
21     5
44     5
35     5
34     4
53     4
54     4
47     4
49     3
39     3
58     3
59     2
66     2
62     2
60     2
56     2
57     1
61     1
55     1
48     1
67     1
70     1
Name: Age, dtype: int64
negative_data Age value counts are :
22    61
21    58
24    38
25    34
23    31
26    25
28    25
27    24
29    16
30    15
37    13
42    11
31    11
34    10
41     9
39     9
33     7
45     7
32     7
40     7
46     6
38     6
36     6
35     5
48     4
63     4
58     4
57     4
60     3
50     3
51     3
55     3
65     3
44     3
69     2
67     2
66     2
62     2
49     2
47     2
43     2
54     2
59     1
61     1
64     1
56     1
53     1
68     1
52     1
72     1
81     1
Name: Age, dtype: int64
```

Now we will visualize these data using histogram

In [178]:
```
# Set figure size
plt.figure(figsize=(15,6))

# for positive data
plt.subplot(1,2,1)
sns.histplot(data=positive_data, x = 'Age', stat='count', color='green')
plt.xlabel('Age')
plt.ylabel('Age Counts')
plt.title('Histogram of Positive Data Age')

# for negative data
plt.subplot(1,2,2)
sns.histplot(data=negative_data, x = 'Age', stat='count', color='green')
plt.xlabel('Age')
plt.ylabel('Age Counts')
plt.title('Histogram of Negative Data Age')

# show plots
plt.show()
```



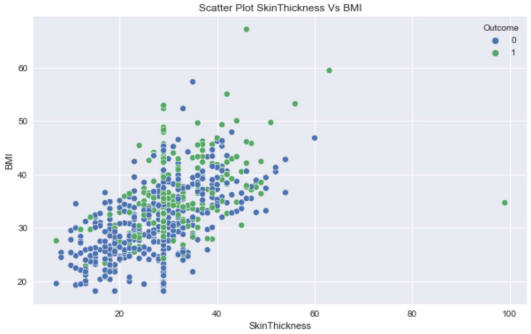
From above analysis we can clearly see the effect of unbalanced data individually on various columns and as we already stated the negative outcome i.e. 0 has more records than the positive outcome 1, so we can conclude following points considering the data:

1. Count of a people who has no pregnancy records show a higher number of diabetes patients as compared to a person with atleast 1 pegnancy or more.
2. Glucose levels are nearly same in both diabetic and non-diabetic person and same applies to BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction.
3. Age category between 20 to 45 has shown higher chance of diabetes.

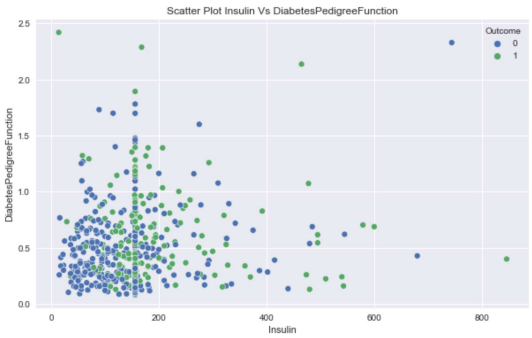
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

In [179]:
```
# Lets create a scatter plot between Glucose & BloodPressure
plt.figure(figsize=(10,6))
sns.scatterplot(x='Glucose', y='BloodPressure', hue='Outcome', data=healthcare)
plt.xlabel('Glucose')
plt.ylabel('BloodPressure')
plt.title('Scatter Plot Glucose Vs BloodPressue')
plt.show()
```

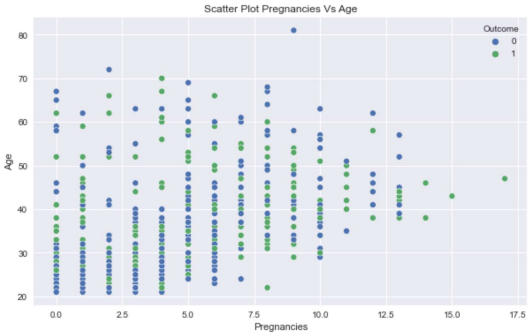
In [180]:
```
# Lets create a scatter plot between SkinThickness & BMI
plt.figure(figsize=(10,6))
sns.scatterplot(x='SkinThickness', y='BMI', hue='Outcome', data=healthcare)
plt.xlabel('SkinThickness')
plt.ylabel('BMI')
plt.title('Scatter Plot SkinThickness Vs BMI')
plt.show()
```



In [181]:
```
# Lets create a scatter plot between Insulin & DiabetesPedigreeFunction
plt.figure(figsize=(10,6))
sns.scatterplot(x='Insulin', y='DiabetesPedigreeFunction', hue='Outcome', data=healthcare)
plt.xlabel('Insulin')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Scatter Plot Insulin Vs DiabetesPedigreeFunction')
plt.show()
```



In [182]:
```
# Lets create a scatter plot between Pregnancies & Age
plt.figure(figsize=(10,6))
sns.scatterplot(x='Pregnancies', y='Age', hue='Outcome', data=healthcare)
plt.xlabel('Pregnancies')
plt.ylabel('Age')
plt.title('Scatter Plot Pregnancies Vs Age')
plt.show()
```



From above analysis we can see that there is somewhat fair positive linear realtionship between SkinThickness and BMI only.

3. Perform correlation analysis. Visually explore it using a heat map.

In [183]:
```
# Correlation Matrix

healthcare.corr()
```

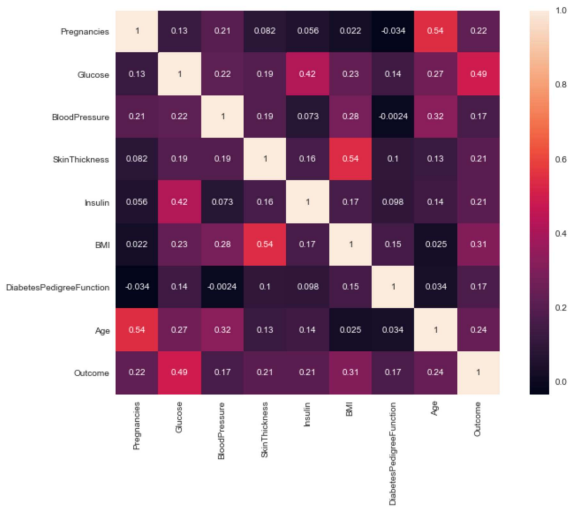
Out[183]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.127891	0.208615	0.081770	0.056479	0.021567	-0.033523	0.544341	0.221898
Glucose	0.127891	1.000000	0.218530	0.192690	0.420052	0.230894	0.137041	0.266507	0.492935
BloodPressure	0.208615	0.218530	1.000000	0.191892	0.072906	0.281335	-0.002378	0.324915	0.165723
SkinThickness	0.081770	0.192690	0.191892	1.000000	0.158134	0.543150	0.102188	0.126107	0.214873
Insulin	0.056479	0.420052	0.072906	0.158134	1.000000	0.166336	0.098191	0.137296	0.214519
BMI	0.021567	0.230894	0.281335	0.543150	0.166336	1.000000	0.153390	0.025498	0.311893
DiabetesPedigreeFunction	-0.033523	0.137041	-0.002378	0.102188	0.098191	0.153390	1.000000	0.033561	0.173844
Age	0.544341	0.266507	0.324915	0.126107	0.137296	0.025498	0.033561	1.000000	0.238356
Outcome	0.221898	0.492935	0.165723	0.214873	0.214519	0.311893	0.173844	0.238356	1.000000

Lets create a heatmap to visualize this data

In [184]:
```
plt.figure(figsize=(12,8))
sns.heatmap(data=healthcare.corr(), annot=True, cbar='viridis', square=True)
```

Out[184]: <AxesSubplot:>



From above analysis we can say that there is no multi-collinearity exits between the independent variable and our target variable Outcome has fairly linear realation ship with Glucose column.

Project Task: Week 3

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

Here we will use Holdout Validation Approach - Train test Split

First we will separate the data into Independent (X) and Dependent (y) variables.

In [185]:
```
# Independent (X) and Dependent (y) variables.

X = healthcare.drop('Outcome', axis=1)
y = healthcare['Outcome']

print(X.shape)
print(y.shape)
```

```
(768, 8)
(768,)
```

In [186]:
```
# Lets split the data into train and test dataset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

In [187]:
```
# We will scale the data using StandardScalar so that we can improve our ML model performance
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Logistic Regression Model

In [188]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000, penalty='l1', solver='liblinear')
```

In [189]:
```
# fit the train data
lr.fit(X_train, y_train)
```

Out[189]: LogisticRegression(max_iter=1000, penalty='l1', solver='liblinear')

In [190]:
```
lr.score(X_train, y_train)
```

Out[190]: 0.7821229050279329

In [191]:
```
lr_predict = lr.predict(X_test)
lr_prob = lr.predict_proba(X_test)
```

In [192]:
```
# Lets check the model performance using metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print('Accuracy Score : ', accuracy_score(y_test, lr_predict))
print('Confusion Matrix : ')
print(confusion_matrix(y_test, lr_predict))
print('Classification Report : ')
print(classification_report(y_test, lr_predict))
```

```
Accuracy Score :  0.7445887445887446
Confusion Matrix :
[[125  26]
 [ 33  47]]
Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       151
           1       0.64      0.59      0.61        80

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

Decision Tree Classifier

In [193]:
```
from sklearn.tree import DecisionTreeClassifier
DTC = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=3)
```

In [194]:
```
# fit the train data
DTC.fit(X_train, y_train)
```

Out[194]:
```
DecisionTreeClassifier(max_depth=3)
```

In [195]:
```
DTC.score(X_train, y_train)
```

Out[195]:
```
0.7653631284916201
```

In [196]:
```
DTC_predict = DTC.predict(X_test)
DTC_prob = DTC.predict_proba(X_test)
```

In [197]:
```
# Lets check the model performance using metrics

print('Accuracy Score : ', accuracy_score(y_test, DTC_predict))
print('Confusion Matrix : ')
print(confusion_matrix(y_test, DTC_predict))
print('Classification Report : ')
print(classification_report(y_test, DTC_predict))
```

```
Accuracy Score :  0.7186147186147186
Confusion Matrix :
[[141  10]
 [ 55  25]]
Classification Report :
              precision    recall  f1-score   support

           0       0.72      0.93      0.81       151
           1       0.71      0.31      0.43        80

    accuracy                           0.72       231
   macro avg       0.72      0.62      0.62       231
weighted avg       0.72      0.72      0.68       231
```

Random Forest Classifier

In [198]:
```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
```

First we will find out the best parameter for Random Forest Classifier. the steps are given below:

In [199]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

In [200]:
```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_spli
t': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

In [201]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
RF = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
RF_random = RandomizedSearchCV(estimator = RF, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
                               random_state=42, n_jobs = -1)
# Fit the random search model
RF_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[201]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

In [202]:
```
RF_random.best_params_
```

Out[202]:
```
{'n_estimators': 400,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'sqrt',
 'max_depth': 80,
 'bootstrap': False}
```

In [243]:
```
# Lets use this parametrs for our model

RF = RandomForestClassifier(n_estimators=400, min_samples_split=10, min_samples_leaf=4, max_features='sqrt',
                            max_depth=80, bootstrap=False)
```

In [244]:
```
# fit the train data
RF.fit(X_train, y_train)
```

Out[244]:
```
RandomForestClassifier(bootstrap=False, max_depth=80, max_features='sqrt',
                       min_samples_leaf=4, min_samples_split=10,
                       n_estimators=400)
```

In [245]:
```
RF.score(X_train, y_train)
```

Out[245]:
```
0.9739292364990689
```

In [246]:
```
RF_predict = RF.predict(X_test)
RF_prob = RF.predict_proba(X_test)
```

In [247]:
```
# Lets check the model performance using metrics

print('Accuracy Score : ', accuracy_score(y_test, RF_predict))
print('Confusion Matrix : ')
print(confusion_matrix(y_test, RF_predict))
print('Classification Report : ')
print(classification_report(y_test, RF_predict))
```

```
Accuracy Score :  0.7445887445887446
Confusion Matrix :
[[119  32]
 [ 27  53]]
Classification Report :
              precision    recall  f1-score   support

           0       0.82      0.79      0.80       151
           1       0.62      0.66      0.64        80

    accuracy                           0.74       231
   macro avg       0.72      0.73      0.72       231
weighted avg       0.75      0.74      0.75       231
```

Support Vector Classifier

In [208]:
```
from sklearn.svm import SVC
svc = SVC(probability=True)
```

In [209]:
```
# fit the train data
svc.fit(X_train, y_train)
```

Out[209]:
```
SVC(probability=True)
```

In [210]:
```
svc.score(X_train, y_train)
```

Out[210]:
```
0.8361266294227188
```

In [211]:
```
svc_predict = svc.predict(X_test)
svc_prob = svc.predict_proba(X_test)
```

In [212]:
```
# Lets check the model performance using metrics

print('Accuracy Score : ', accuracy_score(y_test, svc_predict))
print('Confusion Matrix : ')
print(confusion_matrix(y_test, svc_predict))
print('Classification Report : ')
print(classification_report(y_test, svc_predict))
```

```
Accuracy Score :  0.7445887445887446
Confusion Matrix :
[[127  24]
 [ 35  45]]
Classification Report :
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       151
           1       0.65      0.56      0.60        80

    accuracy                           0.74       231
   macro avg       0.72      0.70      0.71       231
weighted avg       0.74      0.74      0.74       231
```

Naive Bayes Classifier

In [213]:
```
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
```

In [214]:
```
# fit the train data
NB.fit(X_train, y_train)
```

Out[214]:
```
GaussianNB()
```

In [215]:
```
NB.score(X_train, y_train)
```

Out[215]:
```
0.7523277467411545
```

In [216]:
```
NB_predict = NB.predict(X_test)
NB_prob = NB.predict_proba(X_test)
```

In [217]:
```
# Lets check the model performance using metrics

print('Accuracy Score : ', accuracy_score(y_test, NB_predict))
print('Confusion Matrix : ')
print(confusion_matrix(y_test, NB_predict))
print('Classification Report : ')
print(classification_report(y_test, NB_predict))
```

```
Accuracy Score :  0.7402597402597403
Confusion Matrix :
[[119  32]
 [ 28  52]]
Classification Report :
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       151
           1       0.62      0.65      0.63        80

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

K-Nearest Neighbor Classifier

In [218]:
```
# First we will find an optimum value k for n_neighbours

from sklearn.neighbors import KNeighborsClassifier
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[218]: Text(0, 0.5, 'Error Rate')

From above we can see that the error is minimum for k = 25. Lets use this and build KNN model

In [219]:
```
KNN = KNeighborsClassifier(n_neighbors = 25, metric='minkowski', p=2)
```

In [220]:
```
# fit the train data
KNN.fit(X_train, y_train)
```

Out[220]: KNeighborsClassifier(n_neighbors=25)

In [221]:
```
KNN.score(X_train, y_train)
```

Out[221]: 0.7988826815642458

In [222]:
```
KNN_predict = KNN.predict(X_test)
KNN_prob = KNN.predict_proba(X_test)
```

In [223]:
```
# Lets check the model performance using metrics

print('Accuracy Score : ', accuracy_score(y_test, KNN_predict))
print('Confusion Matrix : ')
print(confusion_matrix(y_test, KNN_predict))
print('Classification Report : ')
print(classification_report(y_test, KNN_predict))
```

```
Accuracy Score :  0.7445887445887446
Confusion Matrix :
[[125  26]
 [ 33  47]]
Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       151
           1       0.64      0.59      0.61        80

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

Project Task: Week 4

Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

Now we will calculate auc score and will also plot the roc curve to understand which model performed better. AUC-ROC curve helps us visualize how well our machine learning classifier is performing. We will use tpr - true positive rate and fpr - false positive rate as parameters.

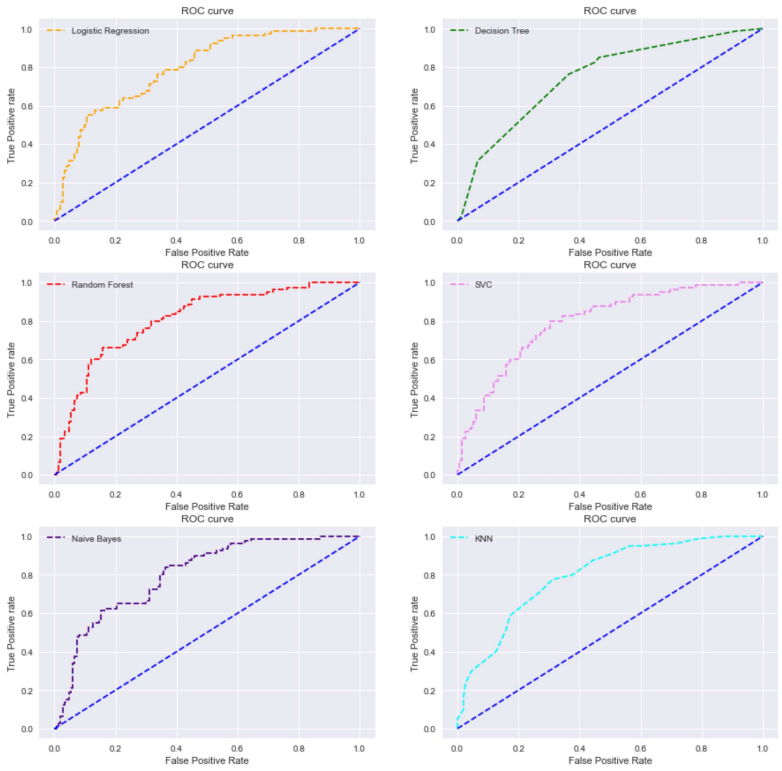
In [224]:
```
from sklearn.metrics import roc_auc_score

# Logistic Regression
print('Logistic Regression AUC Score is : ')
print(roc_auc_score(y_test, lr_prob[:,1]))

# Decision Tree Classifier
print('Decision Tree Classifier AUC Score is : ')
print(roc_auc_score(y_test, DTC_prob[:,1]))

# Random Forest Classifier
print('Random Forest Classifier AUC Score is : ')
print(roc_auc_score(y_test, RF_prob[:,1]))

# Support Vector Classifier
print('Support Vector Classifier AUC Score is : ')
print(roc_auc_score(y_test, svc_prob[:,1]))

# Naive Bayes Classifier
print('Naive Bayes Classifier AUC Score is : ')
print(roc_auc_score(y_test, NB_prob[:,1]))

# K-Nearest Neighbor Classifier
print('K-Nearest Neighbor Classifier AUC Score is : ')
print(roc_auc_score(y_test, KNN_prob[:,1]))
```

```
Logistic Regression AUC Score is :
0.7955298013245033
Decision Tree Classifier AUC Score is :
0.7480132450331125
Random Forest Classifier AUC Score is :
0.8101821192052979
Support Vector Classifier AUC Score is :
0.7971854304635762
Naive Bayes Classifier AUC Score is :
0.8013245033112583
K-Nearest Neighbor Classifier AUC Score is :
0.7963162251655628
```

In [225]:
```
# ROC Curve
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, lr_prob[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, DTC_prob[:,1], pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(y_test, RF_prob[:,1], pos_label=1)
fpr4, tpr4, thresh4 = roc_curve(y_test, svc_prob[:,1], pos_label=1)
fpr5, tpr5, thresh5 = roc_curve(y_test, NB_prob[:,1], pos_label=1)
fpr6, tpr6, thresh6 = roc_curve(y_test, KNN_prob[:,1], pos_label=1)

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

In [226]:
```
# We can also plot the ROC curves for the models using matplotlib:

plt.figure(figsize=(15,15))
plt.style.use('seaborn')

# Logistic Regression
plt.subplot(3,2,1)
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')

# Decision Tree
plt.subplot(3,2,2)
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='Decision Tree')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')

# Random Forest
plt.subplot(3,2,3)
plt.plot(fpr3, tpr3, linestyle='--',color='red', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')

# Support Vector
plt.subplot(3,2,4)
plt.plot(fpr4, tpr4, linestyle='--',color='violet', label='SVC')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')

# Naive Bayes
plt.subplot(3,2,5)
plt.plot(fpr5, tpr5, linestyle='--',color='indigo', label='Naive Bayes')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')

# KNN
plt.subplot(3,2,6)
plt.plot(fpr6, tpr6, linestyle='--',color='cyan', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
```

Out[226]:
```
<matplotlib.legend.Legend at 0x12a1ec07130>
```



From above analysis of various ML algorithms we can conclude following points:

1. Logistic Regression : this model has a score of 0.78, accuracy score of 0.74 and auc score of 0.79 which is fairly good.
2. Decision Tree Classifier : this model has a score of 0.76, accuracy score of 0.71 and auc score is 0.74 which is fairly good.
3. Random Forest Classifier : this model has a score of 0.97, accuracy score of 0.74 which clearly indicates this model is over-fitting and auc score is 0.81. Although auc score is good but model is over-fitting so we need to reject this.
4. Support Vector Classifier : this model has a score of 0.83, accuracy score of 0.74 which also indicates that model performed good on training data but poor on the test data and auc score is 0.79 which is fairly good but still not good for our data.
5. Naive Bayes Classifier : this model has a score of 0.75, accuracy score of 0.74 and auc score of 0.80 which is fairly good.
6. K Nearest Neighbour Classifier : this model has a score of 0.79, accuracy score of 0.74 and auc score of 0.79 which is also fairly good.

Also after looking at the classification report and confusion matrix of these various ML algorithms that we used we conclude that Logistic Regression and KNN algorithm are the best model in this case.

Data Reporting:

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

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

b. Scatter charts between relevant variables to analyze the relationships

c. Histogram or frequency charts to analyze the distribution of the data

d. Heatmap of correlation analysis among the relevant variables

e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

Above task will be performed in Tableau Public Software and we will just upload the Dashboard image here to get a view of the above tasks.

In [227]:
```
# First we will get our cleaned data and download it as csv file and we will use it for our tableau visualization
# healthcare.to_csv('Healthcare_clean.csv')
```

Open Tableau Desktop Public on your system and import the above csv file using the text option. Now click on new workseet and start creating visualizaions.

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

Here first we will create a calculated field named as 'DiabetesTF' using below formula:

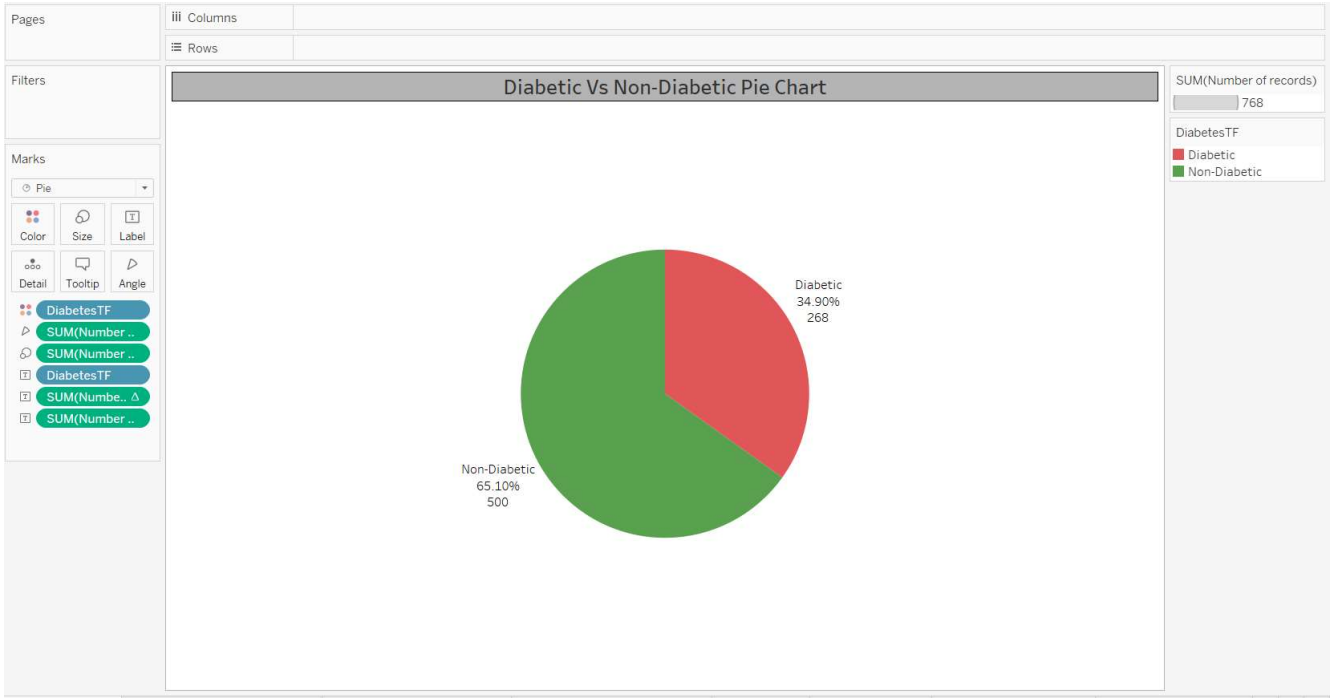
```
IF [Outcome] =1 THEN "Diabetic"
ELSEIF [Outcome]=0 THEN "Non-Diabetic"
END
```

Then create a calculated field named as 'Number of records' and in formula just write 1.

Then select these two fields click on show me and select pie chart. Now add 'DiabetesTF' to label and color and 'Number of records' to label and in measure select 'SUM' and in table calculation select 'Percent of Total'. Now click on color and change the colors to 'Green' for Non-Diabetic and 'Red' for Diabetic. Rename the sheet and title as 'Diabetic Vs Non-Diabetic Pie Chart' and use appropriate shading and other required thing. Also select view as 'Entire View'

The result will look like below:

In [228]:
```
from IPython.display import Image, display
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Diabetic Vs Non-Diabetic Pie Chart.png'))
```



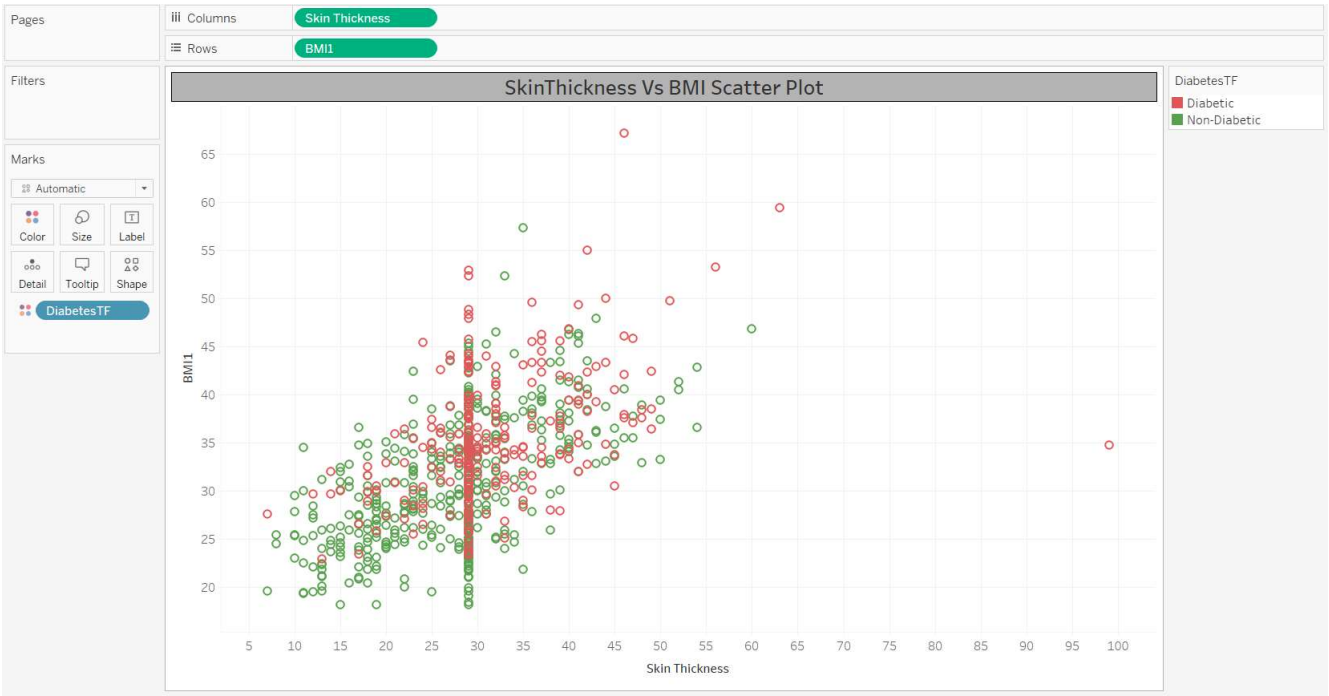
b. Scatter charts between relevant variables to analyze the relationships

Select 'Skin Thickness' and 'BMI1' and in show me select 'Scatter Plot'. Add 'DiabetesTF' to 'Colors'. Rename the sheet and title as 'SkinThickness Vs BMI Scatter Plot' with appropriate shading and borders etc. Select Entire View.

Repeat the same above steps for 'Glucose' Vs 'Insulin' and 'Age' Vs 'Blood Pressure'. Rename those sheets with appropriate shading and borders etc.

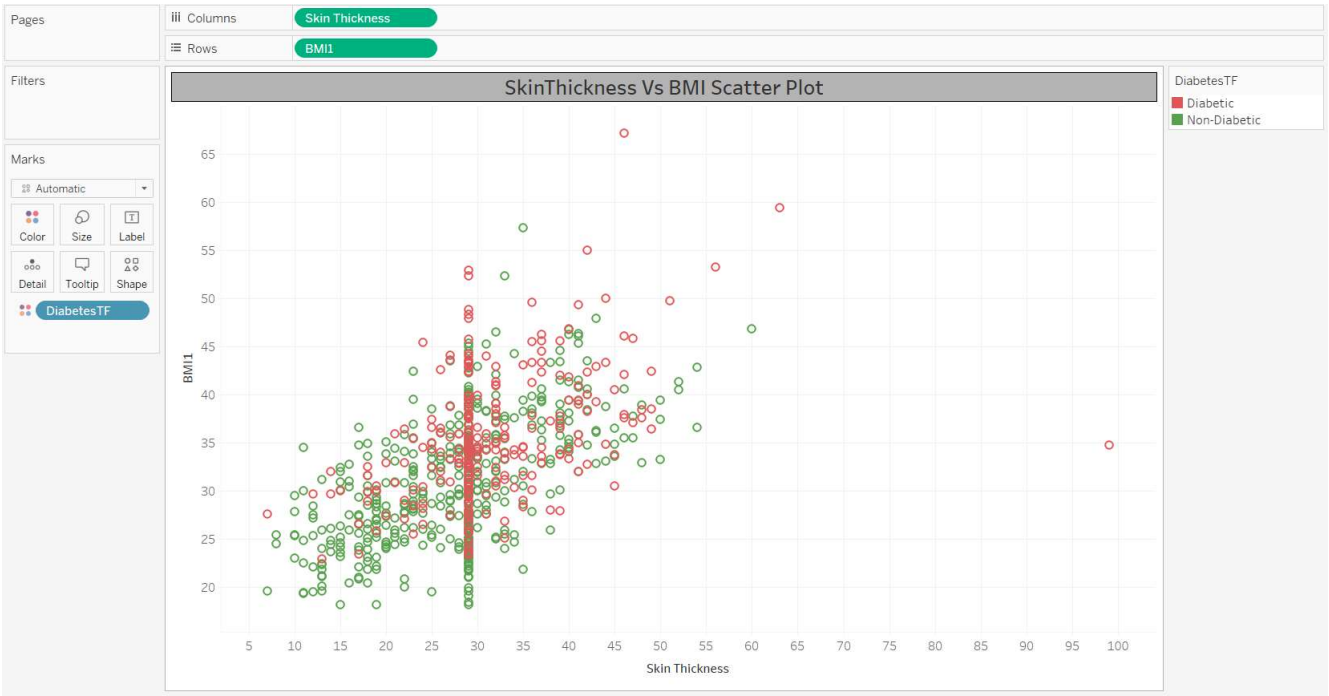
The result will look like below:

In [229]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\SkinThickness Vs BMI Scatter Plot.png'))
```



In [230]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Glucose Vs Insulin Scatter Plot.png'))
```

In [231]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\SkinThickness Vs BMI Scatter Plot.png'))
```



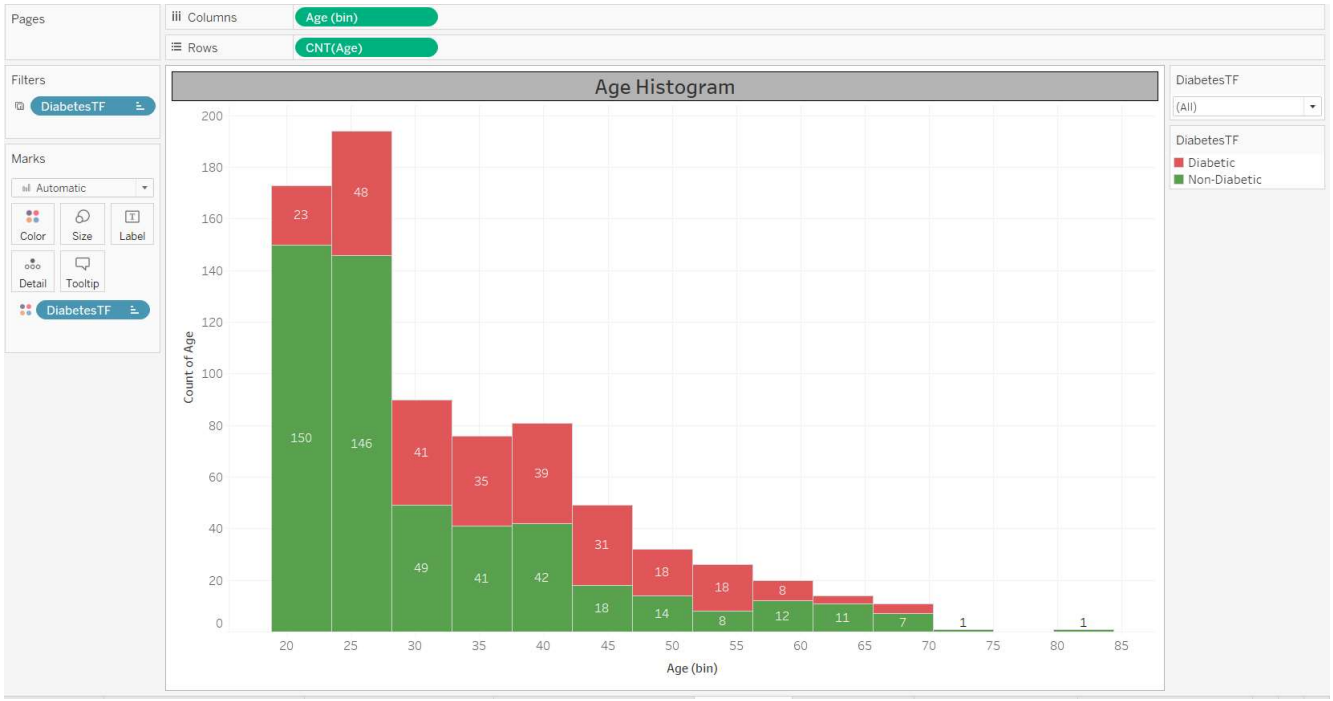
c. Histogram or frequency charts to analyze the distribution of the data

Here select 'Age' and click on Show Me and select Hostogram. Add 'DiabetesTF' to 'Colors' and also to 'Filter' and select all click on apply>>ok. Rename the sheet with appropriate name and borders etc. Select Entire View.

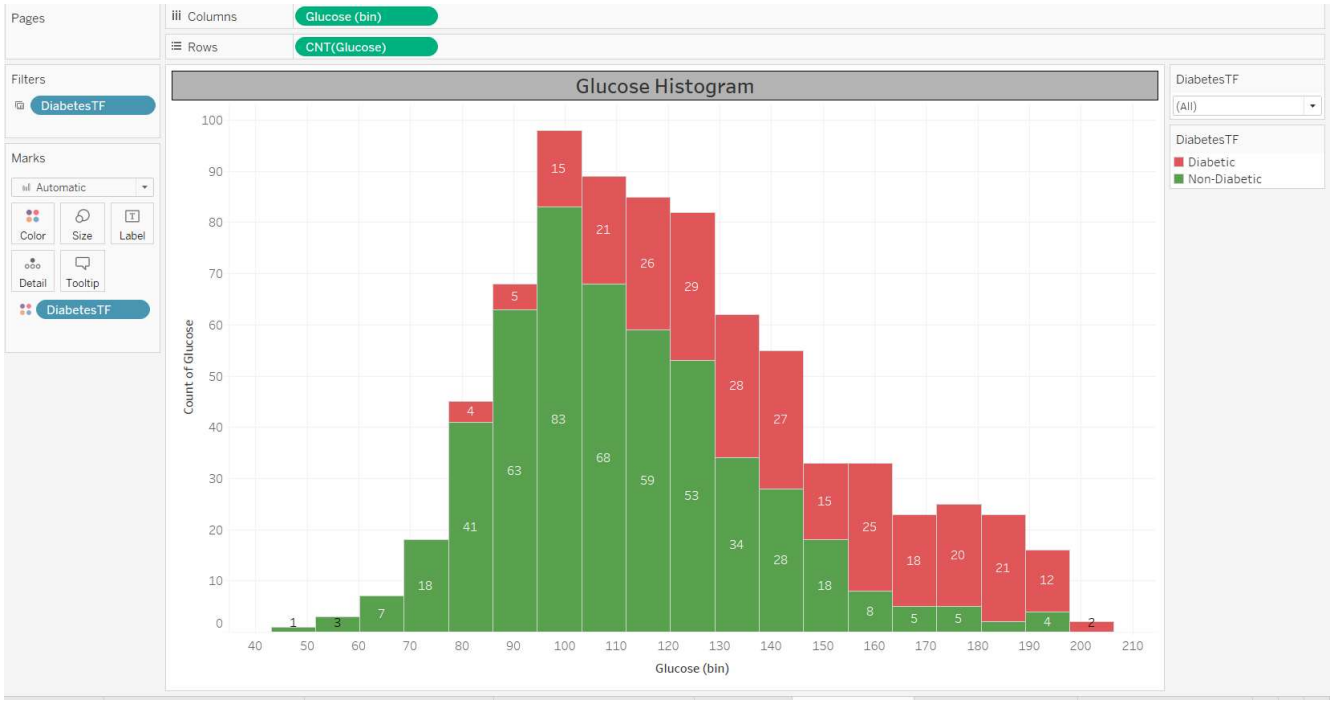
Repeat above step for 'Glucose' and 'Blood Pressure'. Rename those sheets with appropriate shading and borders etc.

The result will look like below:

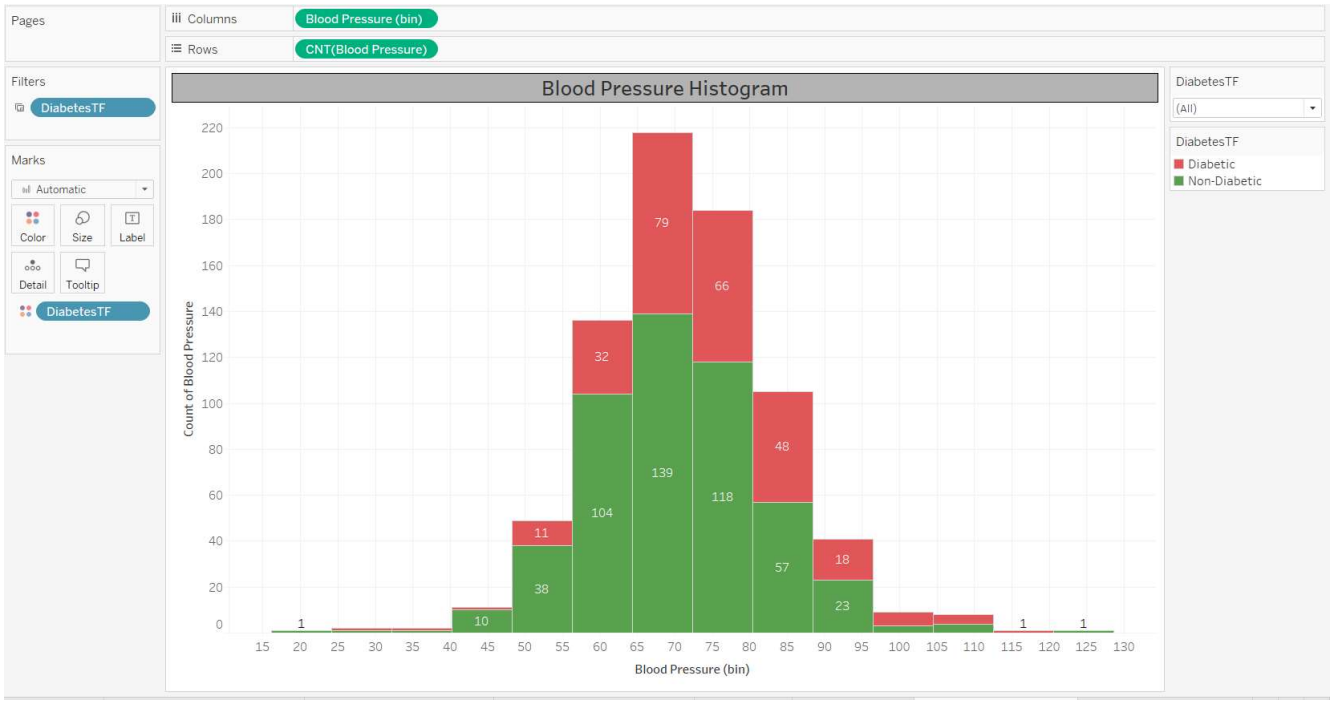
In [232]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Age Histogram.png'))
```



In [233]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Glucose Histogram.png'))
```



In [234]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Blood Pressure Histogram.png'))
```



d. Heatmap of correlation analysis among the relevant variables

Here first create a calculated field using formula:

```
IF
[Age]>=20 and [Age]<=25
THEN "20-25"
ELSEIF
[Age]>=25 and [Age]<=30
THEN "25-30"
ELSEIF
[Age]>=35 and [Age]<=40
THEN "35-40"
ELSEIF
[Age]>=40 and [Age]<=45
THEN "40-45"
ELSEIF
[Age]>=45 and [Age]<=50
THEN "45-50"
ELSEIF
[Age]>=50 and [Age]<=55
THEN "50-55"
ELSEIF
[Age]>=55 and [Age]<=60
THEN "55-60"
ELSEIF
[Age]>=60 and [Age]<=65
THEN "60-65"
ELSEIF
[Age]>=65 and [Age]<=70
THEN "65-70"
ELSEIF
[Age]>=70 and [Age]<=75
THEN "70-75"
ELSEIF
[Age]>=75 and [Age]<=80
THEN "75-80"
ELSEIF
[Age]>=80 and [Age]<=85
THEN "80-85"

END
```

Now add 'AgeBins' to columns and 'Measure Names' to rows 'Measure Values' to 'Details' and 'Colors' and in 'Measure Values' keep only 'BMI1', 'Blood Pressure', 'Glucose', 'Insulin', 'Pregnancies', 'Skin thickness' and select 'Measure' as 'AVG'. Rename the sheet and title with appropriate name, shading and border etc. Select Entire View.

The result will look like below:

In [235]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Heatmap of relevant variables.png'))
```

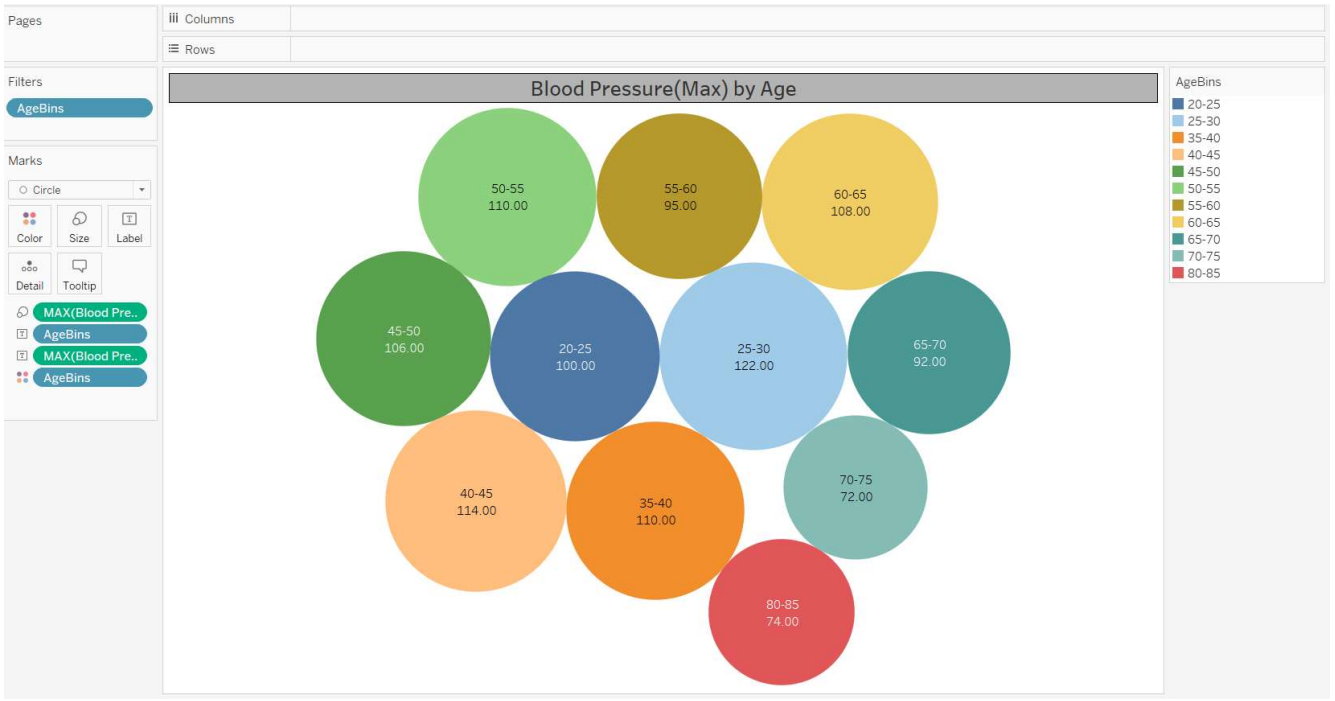
e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

We have already created 'AgeBins'. So now select 'Blood Pressure' and 'AgeBins' and click on Bubble chart in 'Show Me'. Add 'Blood Pressure' to label and also select 'Measure' as 'Maximimum' for size and label. Rename the sheet and title appropriately and select shading and border etc. Select Entire View.

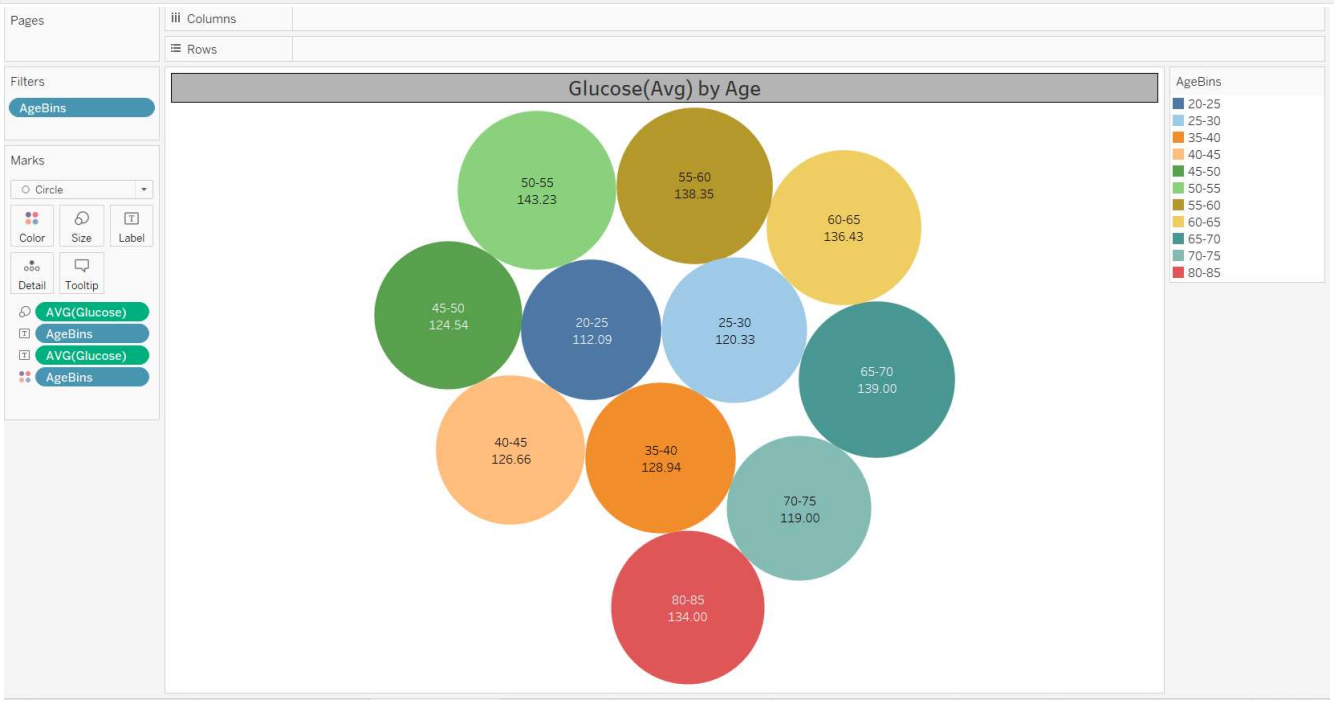
Repeat above steps for 'Glucose', 'BMI1' and 'Pregnancies' with only one difference select 'Measure' as 'AVG' in these. Rename the sheet and title appropriately and select shading and border etc.

The result will look like below.

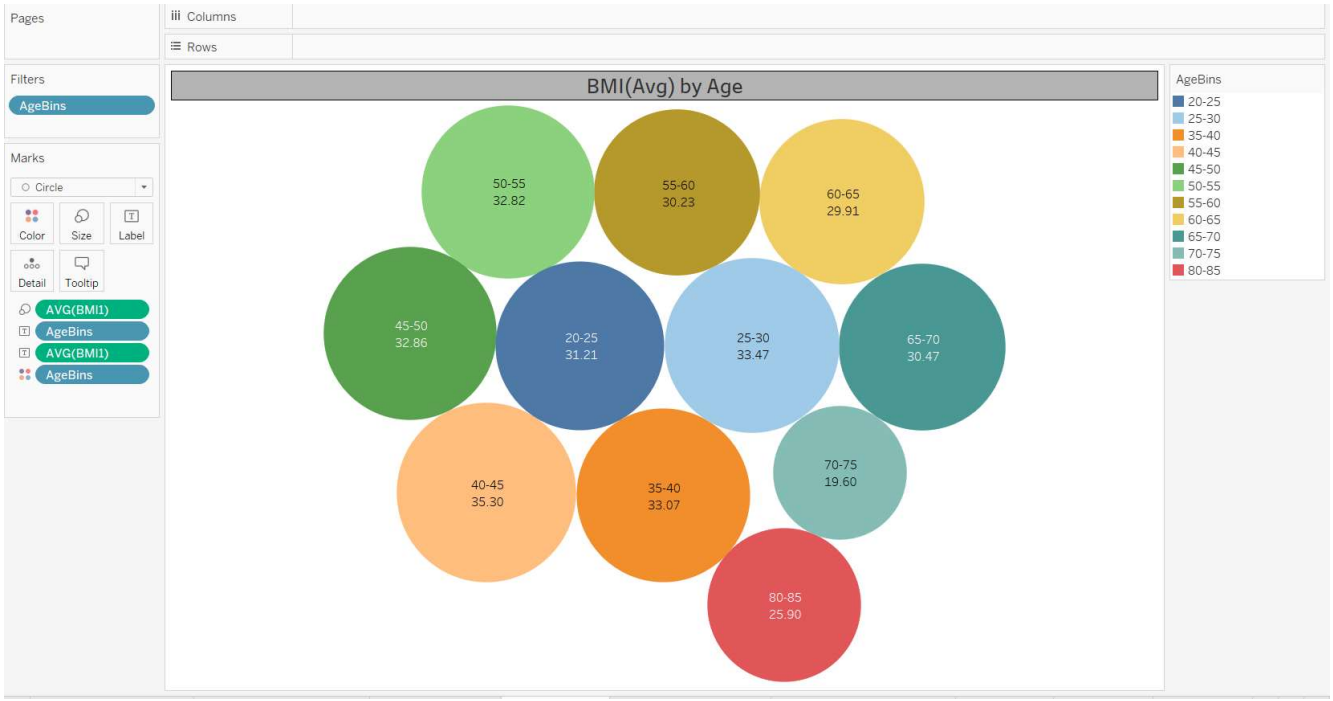
In [236]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Blood Pressure(Max) by Age.png'))
```



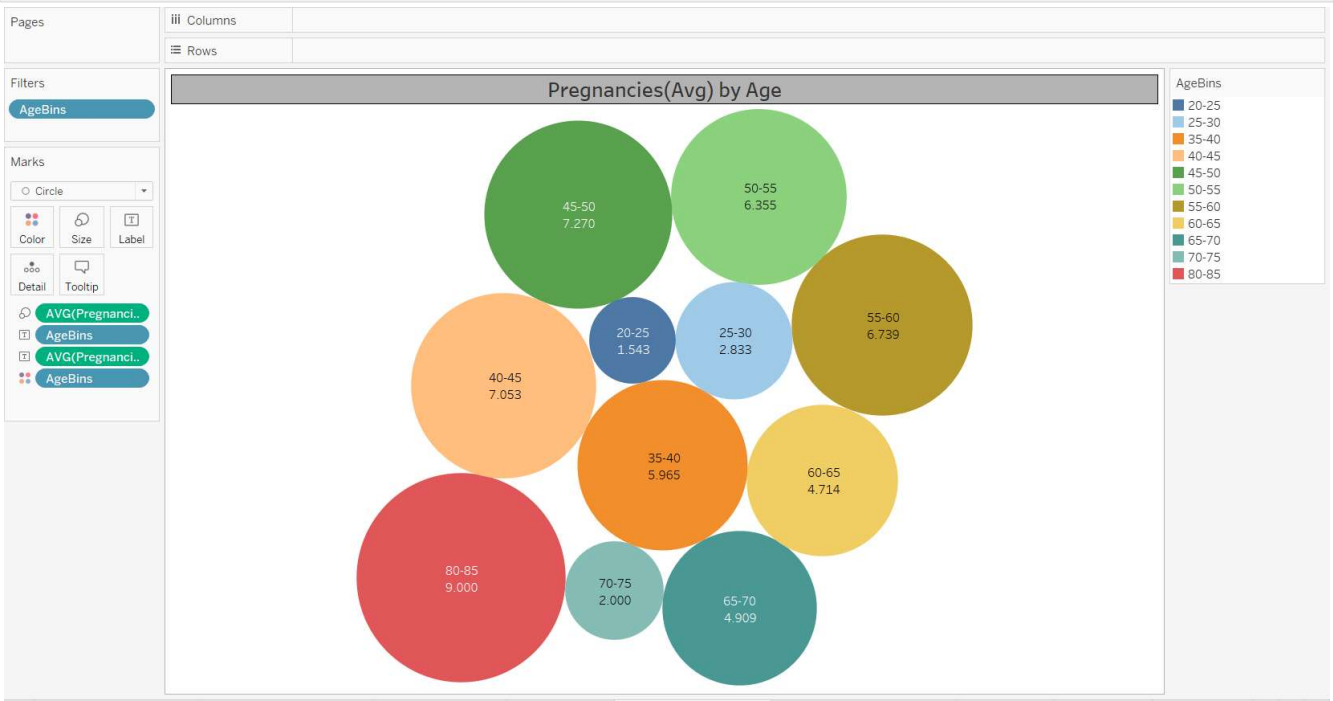
In [237]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Glucose(Avg) by Age.png'))
```



In [238]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\BMI(Avg) by Age.png'))
```



In [239]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Pregnancies(Avg) by Age.png'))
```



Creating the Dashboard

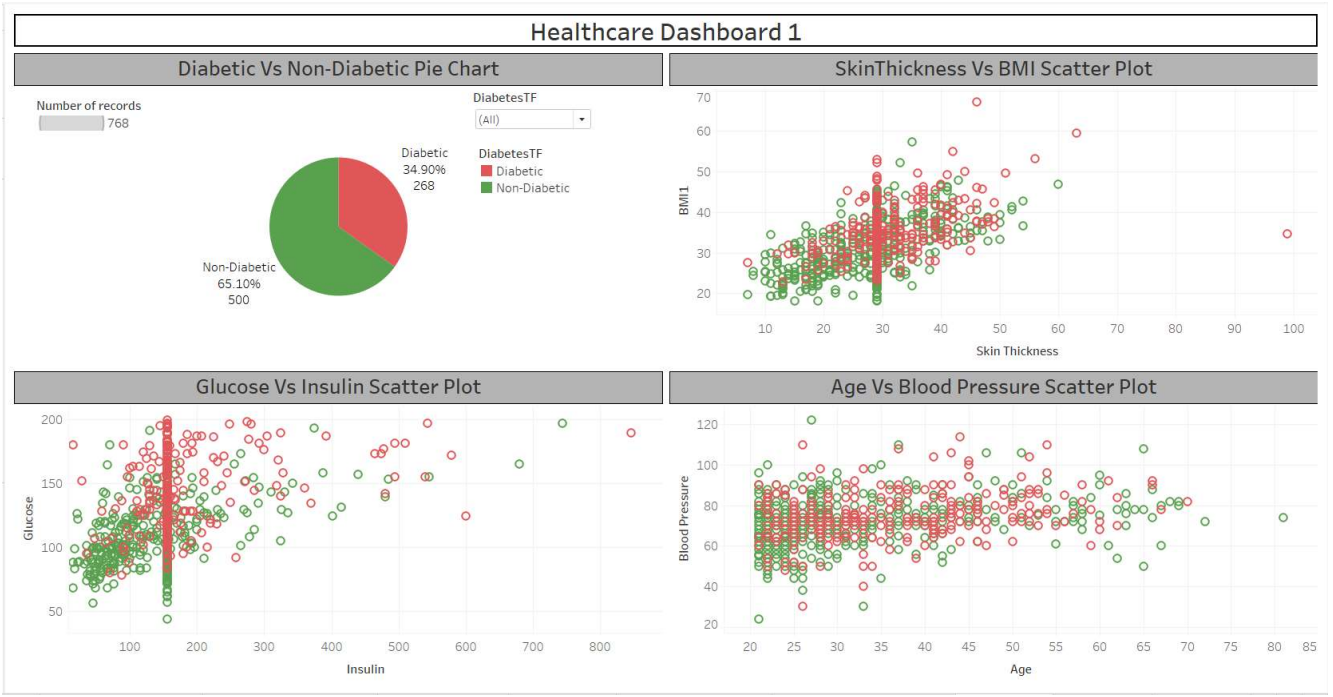
Select New Dashboard. Select 'Size' >> 'Automatic'. Drag 'Horizontal Container' to canvas and select 'Distribute contents evenly'. Also add another 'Horizontal Container' to the bottom of previous. Now in first Dashboard add pie chart and scatter plots. Rename the Dashboard Title as 'Healthcare Dashboard 1'.

Now select New Dashboard and repeat the same procedure as above but add the histograms and heatmap to it. Rename the Dashboard Title as 'Healthcare Dashboard 2'.

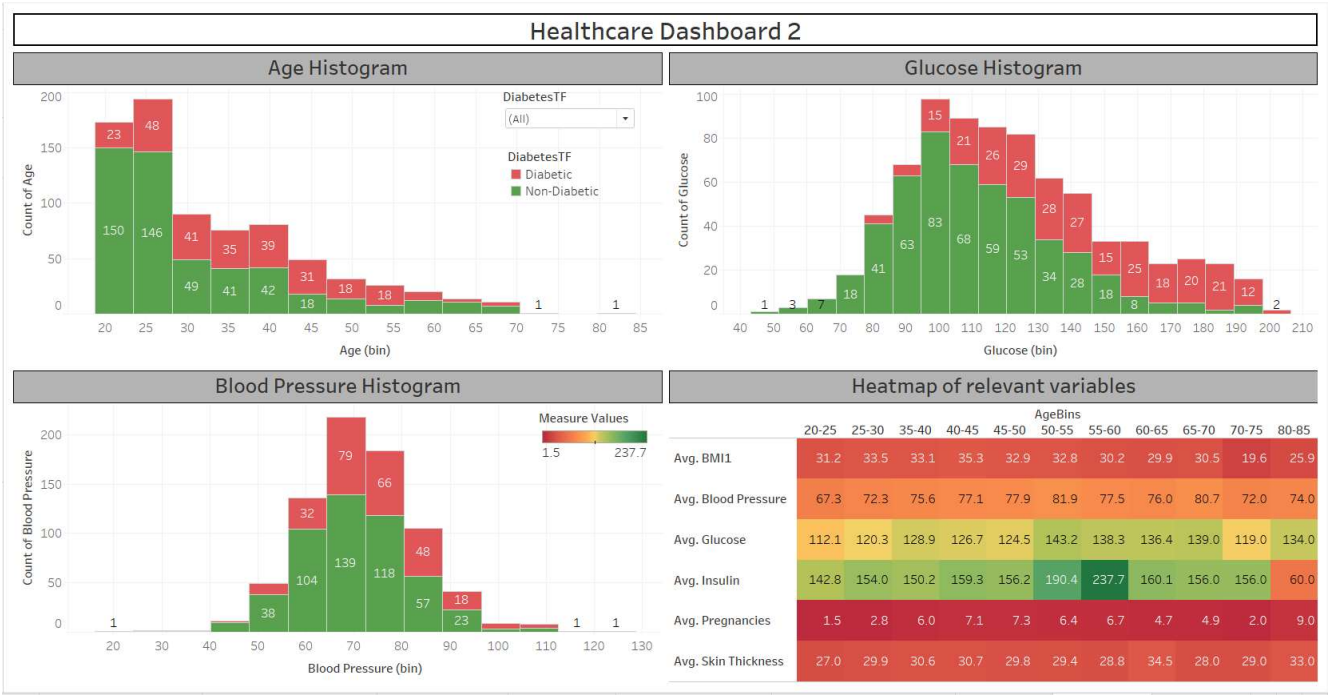
Now select New Dashboard and repeat the same procedure as above but add the Bubble Charts to it. Rename the Dashboard Title as 'Healthcare Dashboard 3'.

The result will look like below:

In [240]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Dashboard1.png'))
```



In [241]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Dashboard2.png'))
```



In [242]:
```
display(Image(filename='E:\Simplilearn\Electives\Data Science Capstone\Projects\Healthcare\Final Submission\Tableau\Dashboard3.png'))
```