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

In [2]:
```python
data_df=pd.read_csv(r'C:\Users\Mohd Shadab\Downloads\Projects\CAPSTONE PROJECTS\Project 2\Healthcare - Diabetes\health care diabetes.csv')
data_df
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [3]:
```python
data_df.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```python
data_df.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 [5]:
```python
data_df.describe()
```

Out[5]:

	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 [6]:
```python
data_df
```

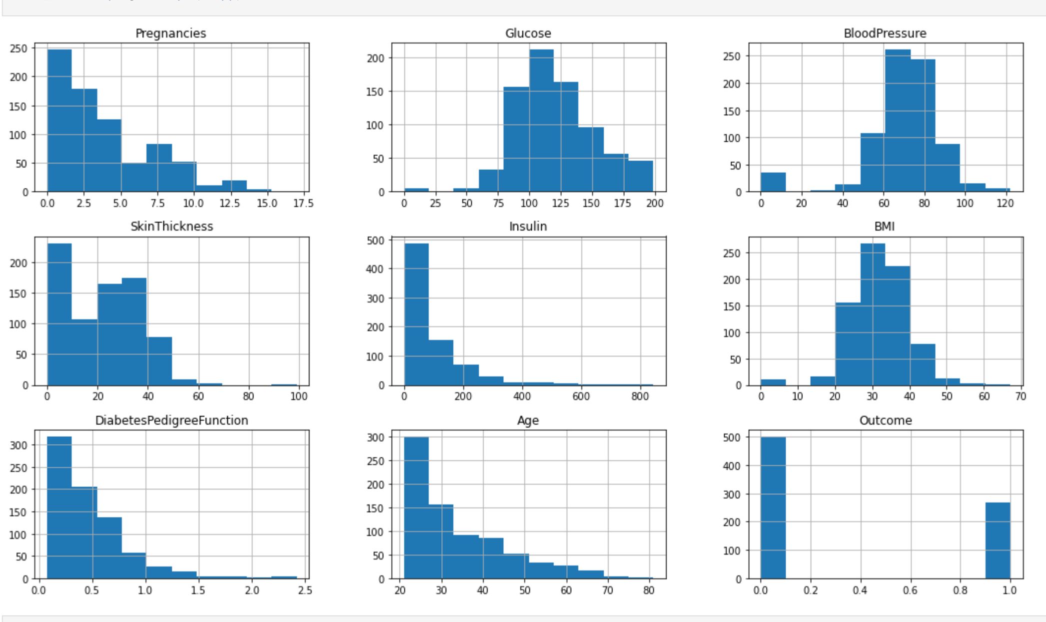
Out[6]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [7]:
```python
#Statistical data above shows the presence of nulls in the columns Glucose,BloodPressure,SkinThickness,Insulin,BMI
#Since the value 0 doesn't make any sense.
```

In [9]:
```python
data_df.hist(figsize=(18,10));
```



In [10]:
```python
#Histogram visualisation confirms the presence of 0 values.The rows with 0 in such columns shall be treated as null
```

In [11]:
```python
data_df['Glucose'].replace(0,np.nan,inplace=True)
data_df['BloodPressure'].replace(0,np.nan,inplace=True)
data_df['SkinThickness'].replace(0,np.nan,inplace=True)
data_df['Insulin'].replace(0,np.nan,inplace=True)
data_df['BMI'].replace(0,np.nan,inplace=True)
```

In [12]:
```python
data_df.isnull().sum() #Total null values
```

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

In [13]:
```python
data_df.isnull().sum()*100/len(data_df) # Percentage of null values
```

Out[13]:
```
Pregnancies                  0.000000
Glucose                      0.651042
BloodPressure                4.557292
SkinThickness               29.557292
Insulin                     48.697917
BMI                          1.432292
DiabetesPedigreeFunction     0.000000
Age                          0.000000
Outcome                      0.000000
dtype: float64
```

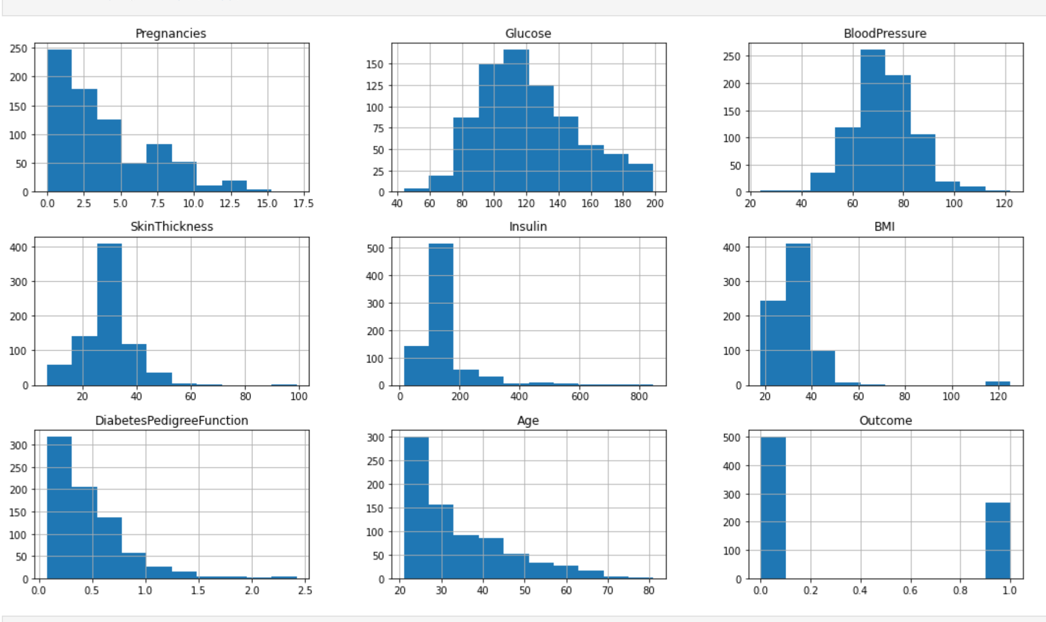
In [14]:
```python
#Since the data is skewed, we will use median values to treat null values
```

In [15]:
```python
data_df['Glucose'].fillna(data_df['Glucose'].median(),inplace=True)
data_df['BloodPressure'].fillna(data_df['BloodPressure'].median(),inplace=True)
data_df['SkinThickness'].fillna(data_df['SkinThickness'].median(),inplace=True)
data_df['Insulin'].fillna(data_df['Insulin'].median(),inplace=True)
data_df['BMI'].fillna(data_df['Insulin'].median(),inplace=True)
```

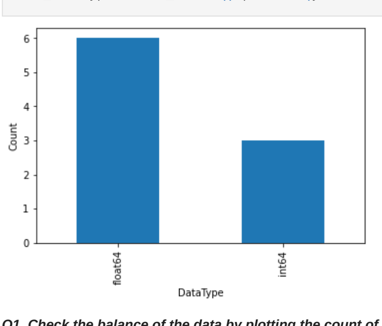
In [16]:
```python
data_df.isnull().sum() #Hence Null values have been treated
```

Out[16]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [17]:
```python
data_df.hist(figsize=(18,10));
```



In [18]:
```python
data_df.dtypes.value_counts().plot.bar(ylabel='Count',xlabel='DataType');
```



Q1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

In [19]:
```python
data_df['Outcome'].value_counts()*100/len(data_df)
```

Out[19]:
```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

In [20]:
```python
#The above percentage shows that the data is not imbalanced.
```

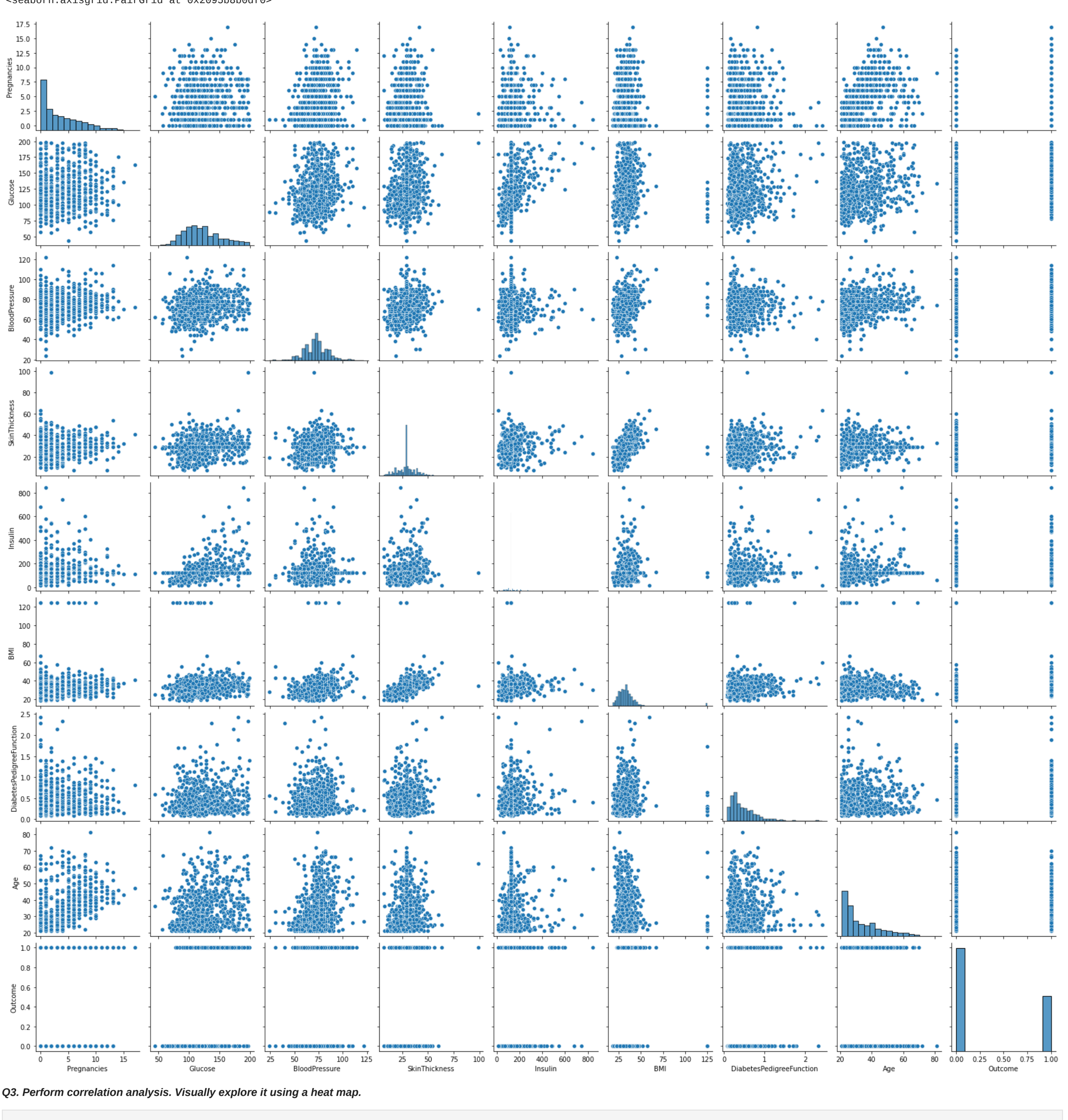
In [21]:
```python
data_df.columns
```

Out[21]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Q2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

In [22]:
```python
sns.pairplot(data_df)
```

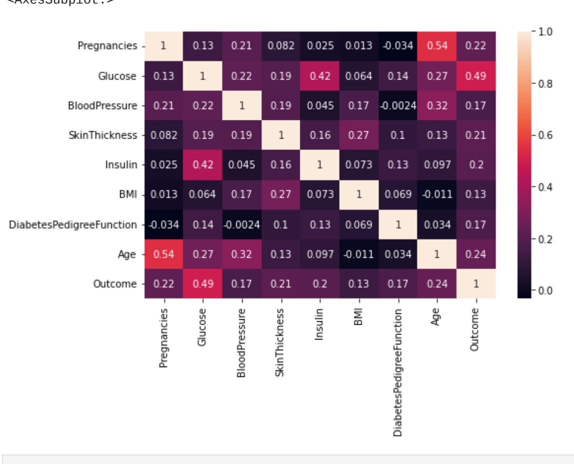
Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x2095b8b0df0>
```



Q3. Perform correlation analysis. Visually explore it using a heat map.

In [23]:
```python
plt.figure(figsize=(8,5))
data=data_df.corr()
sns.heatmap(data=data ,annot=True)
```

Out[23]:
```
<AxesSubplot:>
```



In [24]:
```python
#Glucose is best correlated with Outcome
#Age and Pregnancies shows medium correlation
```

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.

In [25]:
```python
x=data_df.drop(['Outcome'],axis=1)
y=data_df['Outcome']
```

In [26]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

In [27]:
```python
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(576, 8)
(192, 8)
(576,)
(192,)
```

In [28]:
```python
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [29]:
```python
x_train
```

Out[29]:
```
array([[ 1.50755225, -1.09847803, -0.91178438, ..., -0.88985272,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.13223385, -1.24920025, ..., -0.05014643,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.0318405 ,  0.60658704, ..., -0.12012195,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.9318842 , -0.65872248, ..., -0.71880144,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23175309,  0.10046323, ..., -0.29894829,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46750392,  0.77529498, ...,  7.07958205,
         0.51627505,  2.953134  ]])
```

KNN

In [30]:
```python
from sklearn.neighbors import KNeighborsClassifier
KNN_classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
KNN_classifier.fit(x_train, y_train)
```

Out[30]:
```
KNeighborsClassifier()
```

In [31]:
```python
#Predicting the test set result
y_pred= KNN_classifier.predict(x_test)
y_pred
```

Out[31]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0], dtype=int64)
```

In [32]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm= confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy =",accuracy_score(y_test,y_pred))
```

```
[[112  18]
 [ 27  35]]
Accuracy = 0.765625
```

In [33]:
```python
#Tuning Hyperparameter
for i in range(1,7):
    for j in range(1,5):
            KNN_classifier= KNeighborsClassifier(n_neighbors=i, metric='minkowski', p=j )
            KNN_classifier.fit(x_train, y_train)
            y_pred= KNN_classifier.predict(x_test)
            print("Accuracy =",accuracy_score(y_test,y_pred),"n_neighbors =",i,"  p =",j)
```

```
Accuracy = 0.6979166666666666 n_neighbors = 1   p = 1
Accuracy = 0.6822916666666666 n_neighbors = 1   p = 2
Accuracy = 0.671875 n_neighbors = 1   p = 3
Accuracy = 0.6875 n_neighbors = 1   p = 4
Accuracy = 0.703125 n_neighbors = 2   p = 1
Accuracy = 0.71875 n_neighbors = 2   p = 2
Accuracy = 0.71875 n_neighbors = 2   p = 3
Accuracy = 0.7135416666666666 n_neighbors = 2   p = 4
Accuracy = 0.7291666666666666 n_neighbors = 3   p = 1
Accuracy = 0.7447916666666666 n_neighbors = 3   p = 2
Accuracy = 0.7291666666666666 n_neighbors = 3   p = 3
Accuracy = 0.734375 n_neighbors = 3   p = 4
Accuracy = 0.7552083333333334 n_neighbors = 4   p = 1
Accuracy = 0.7604166666666666 n_neighbors = 4   p = 2
Accuracy = 0.7604166666666666 n_neighbors = 4   p = 3
Accuracy = 0.7552083333333334 n_neighbors = 4   p = 4
Accuracy = 0.7604166666666666 n_neighbors = 5   p = 1
Accuracy = 0.765625 n_neighbors = 5   p = 2
Accuracy = 0.765625 n_neighbors = 5   p = 3
Accuracy = 0.75 n_neighbors = 5   p = 4
Accuracy = 0.78125 n_neighbors = 6   p = 1
Accuracy = 0.7864583333333334 n_neighbors = 6   p = 2
Accuracy = 0.7708333333333334 n_neighbors = 6   p = 3
Accuracy = 0.7604166666666666 n_neighbors = 6   p = 4
```

In [36]:
```python
#best accuracy we get out of KNN algorithm is when n_neighbors=6 and p=2
KNN_classifier= KNeighborsClassifier(n_neighbors=6, metric='minkowski', p=2 )
KNN_classifier.fit(x_train, y_train)
y_pred= KNN_classifier.predict(x_test)
KNN_Accuracy =accuracy_score(y_test,y_pred)
KNN_Accuracy
```

Out[36]:
```
0.7864583333333334
```

Logistic regression model

In [37]:
```python
from sklearn.linear_model import LogisticRegression
LR_classifier= LogisticRegression(random_state=0)
LR_classifier.fit(x_train, y_train)
```

Out[37]:
```
LogisticRegression(random_state=0)
```

In [38]:
```python
#Predicting the test set result
y_pred= LR_classifier.predict(x_test)
y_pred
```

Out[38]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [39]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm= confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy =",accuracy_score(y_test,y_pred))
```

```
[[115  15]
 [ 25  37]]
Accuracy = 0.7916666666666666
```

In [40]:
```python
LR_Accuracy=accuracy_score(y_test,y_pred)
```

Using Decision tree

In [41]:
```python
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier
DT_classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
DT_classifier.fit(x_train, y_train)
```

Out[41]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [42]:
```python
#Predicting the test set result
y_pred= DT_classifier.predict(x_test)
y_pred
```

Out[42]:
```
array([1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0], dtype=int64)
```

In [43]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm= confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy =",accuracy_score(y_test,y_pred))
```

```
[[100  30]
 [ 25  37]]
Accuracy = 0.7135416666666666
```

In [44]:
```python
#Tuning Hyperparameter
#using gini as criteriom and testing accuracy
from sklearn.tree import DecisionTreeClassifier
DT_classifier= DecisionTreeClassifier(criterion='gini', random_state=0)
DT_classifier.fit(x_train, y_train)
y_pred= DT_classifier.predict(x_test)
y_pred
from sklearn.metrics import confusion_matrix, accuracy_score
cm= confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy =",accuracy_score(y_test,y_pred))

DT_Accuracy=accuracy_score(y_test,y_pred)
```

```
[[108  22]
 [ 24  38]]
Accuracy = 0.7604166666666666
```

In [45]:
```python
#Tuning hyperparameter increases the accuracy by around 5%
```

Random Forest

In [46]:
```python
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
for i in range(5,15):
    RF_classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
    RF_classifier.fit(x_train, y_train)
    y_pred= RF_classifier.predict(x_test)
    cm= confusion_matrix(y_test, y_pred)
    print("Accuracy =",accuracy_score(y_test,y_pred),"n_estimators =",i)
```

```
Accuracy = 0.7604166666666666 n_estimators = 5
Accuracy = 0.7552083333333334 n_estimators = 6
Accuracy = 0.7395833333333334 n_estimators = 7
Accuracy = 0.7604166666666666 n_estimators = 8
Accuracy = 0.78125 n_estimators = 9
Accuracy = 0.75 n_estimators = 10
Accuracy = 0.7135416666666666 n_estimators = 11
Accuracy = 0.765625 n_estimators = 12
Accuracy = 0.734375 n_estimators = 13
Accuracy = 0.7708333333333334 n_estimators = 14
```

In [47]:
```python
#best Accuracy is obtained using the hyperparameter values as n_estimators=11
RF_classifier= RandomForestClassifier(n_estimators= 8, criterion="entropy")
RF_classifier.fit(x_train, y_train)
y_pred= RF_classifier.predict(x_test)
cm= confusion_matrix(y_test, y_pred)
RF_Accuracy =accuracy_score(y_test,y_pred)
```

Gradient Boosting

In [48]:
```python
from sklearn.ensemble import GradientBoostingClassifier
GB_classifier=GradientBoostingClassifier()
GB_classifier.fit(x_train,y_train)
y_pred=GB_classifier.predict(x_test)

GB_Accuracy=accuracy_score(y_test,y_pred)
GB_Accuracy
```

Out[48]:
```
0.7760416666666666
```

In [49]:
```python
pd.DataFrame([KNN_Accuracy,LR_Accuracy,DT_Accuracy,RF_Accuracy,GB_Accuracy],index=['KNN','LR','DT','RF','GB'],columns=['Accuracy'])
```

Out[49]:

	Accuracy
KNN	0.786458
LR	0.791667
DT	0.760417
RF	0.776042
GB	0.776042

In [50]:
```python
#Highest accuracy is given by Logistic Regression
```

Using cross validation to avoid the data leakage effect on model and checking the accuracy

In [51]:
```python
#KNN
```

In [52]:
```python
#Testing on train dataset without exposing the test dataset
from sklearn.model_selection import cross_val_score
scores=cross_val_score(KNN_classifier,x_train,y_train,scoring='accuracy',cv=10)
```

In [53]:
```python
scores.mean()
```

Out[53]:
```
0.7045674531155475
```

In [54]:
```python
#fitting the model
KNN_classifier.fit(x_train,y_train)
```

Out[54]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [55]:
```python
y_final_pred=KNN_classifier.predict(x_test)
accuracy_score(y_test,y_final_pred)
```

Out[55]:
```
0.7864583333333334
```

In [56]:
```python
#Logistic Regression
```

In [57]:
```python
from sklearn.model_selection import cross_val_score
scores=cross_val_score(LR_classifier,x_train,y_train,scoring='accuracy',cv=10)
scores.mean()
```

Out[57]:
```
0.7447368421052631
```

In [58]:
```python
LR_classifier.fit(x_train,y_train)
y_final_pred=LR_classifier.predict(x_test)
accuracy_score(y_test,y_final_pred)
```

Out[58]:
```
0.7916666666666666
```

In [59]:
```python
#Decision Tree
```

In [60]:
```python
from sklearn.model_selection import cross_val_score
scores=cross_val_score(DT_classifier,x_train,y_train,scoring='accuracy',cv=10)
scores.mean()
```

Out[60]:
```
0.6771324863883847
```

In [61]:
```python
DT_classifier.fit(x_train,y_train)
y_final_pred=DT_classifier.predict(x_test)
accuracy_score(y_test,y_final_pred)
```

Out[61]:
```
0.7604166666666666
```

In [62]:
```python
#Random Forest
```

In [63]:
```python
from sklearn.model_selection import cross_val_score
scores=cross_val_score(RF_classifier,x_train,y_train,scoring='accuracy',cv=10)
scores.mean()
```

Out[63]:
```
0.7290683605565638
```

In [64]:
```python
RF_classifier.fit(x_train,y_train)
y_final_pred=RF_classifier.predict(x_test)
accuracy_score(y_test,y_final_pred)
```

Out[64]:
```
0.7760416666666666
```

In [65]:
```python
#Gradient Boosting
```

In [66]:
```python
from sklearn.model_selection import cross_val_score
scores=cross_val_score(GB_classifier,x_train,y_train,scoring='accuracy',cv=10)
scores.mean()
```

Out[66]:
```
0.7394736842105264
```

In [67]:
```python
GB_classifier.fit(x_train,y_train)
y_final_pred=GB_classifier.predict(x_test)
accuracy_score(y_test,y_final_pred)
```

Out[67]:
```
0.7864583333333334
```

In [68]:
```python
#After cross validation, Logistic regression still gives the highest accuracy.
```

Prediction

In [70]:
```python
def diabetes_prediction (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
       BMI, DiabetesPedigreeFunction, Age):
        pr=int(Pregnancies)
        gluc=float(Glucose)
        bp=float(BloodPressure)
        st=float(SkinThickness)
        ins=float(Insulin)
        bmi=float(BMI)
        dpf=float(DiabetesPedigreeFunction)
        age=int(Age)

        x=[[pr,gluc,bp,st,ins,bmi,dpf,age]]
        x=st_x.transform(x)

        return LR_classifier.predict(x)
```

In [71]:
```python
#Providing random values to the parameters to check our model
```

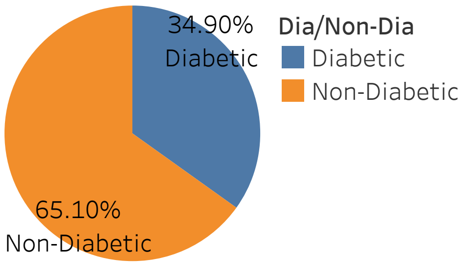
In [72]:
```python
#Prediction 1
pred=diabetes_prediction(2,80,73,16,75,31.1,0.555,25)[0]
if pred:
    print("You have diabetes.")
else:
    print("You don't have diabetes")
```

```
You don't have diabetes
```

In [73]:
```python
#Prediction 2
pred=diabetes_prediction(1,140,92,30,145,35,0.555,70)[0]
if pred:
    print("You have diabetes.")
else:
    print("You don't have diabetes")
```

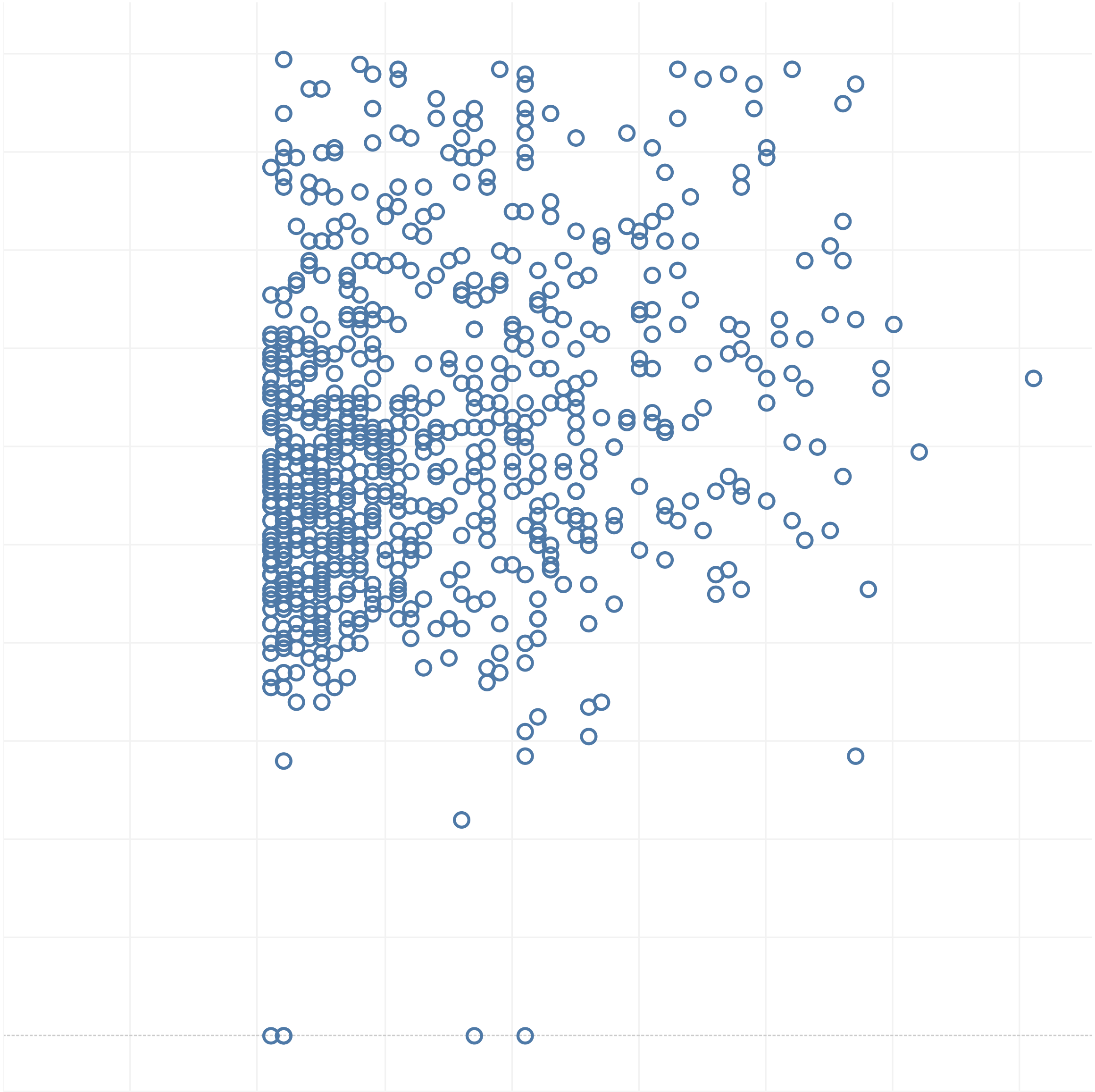
```
You have diabetes.
```

Diabetic vs Non-Diabet-ic



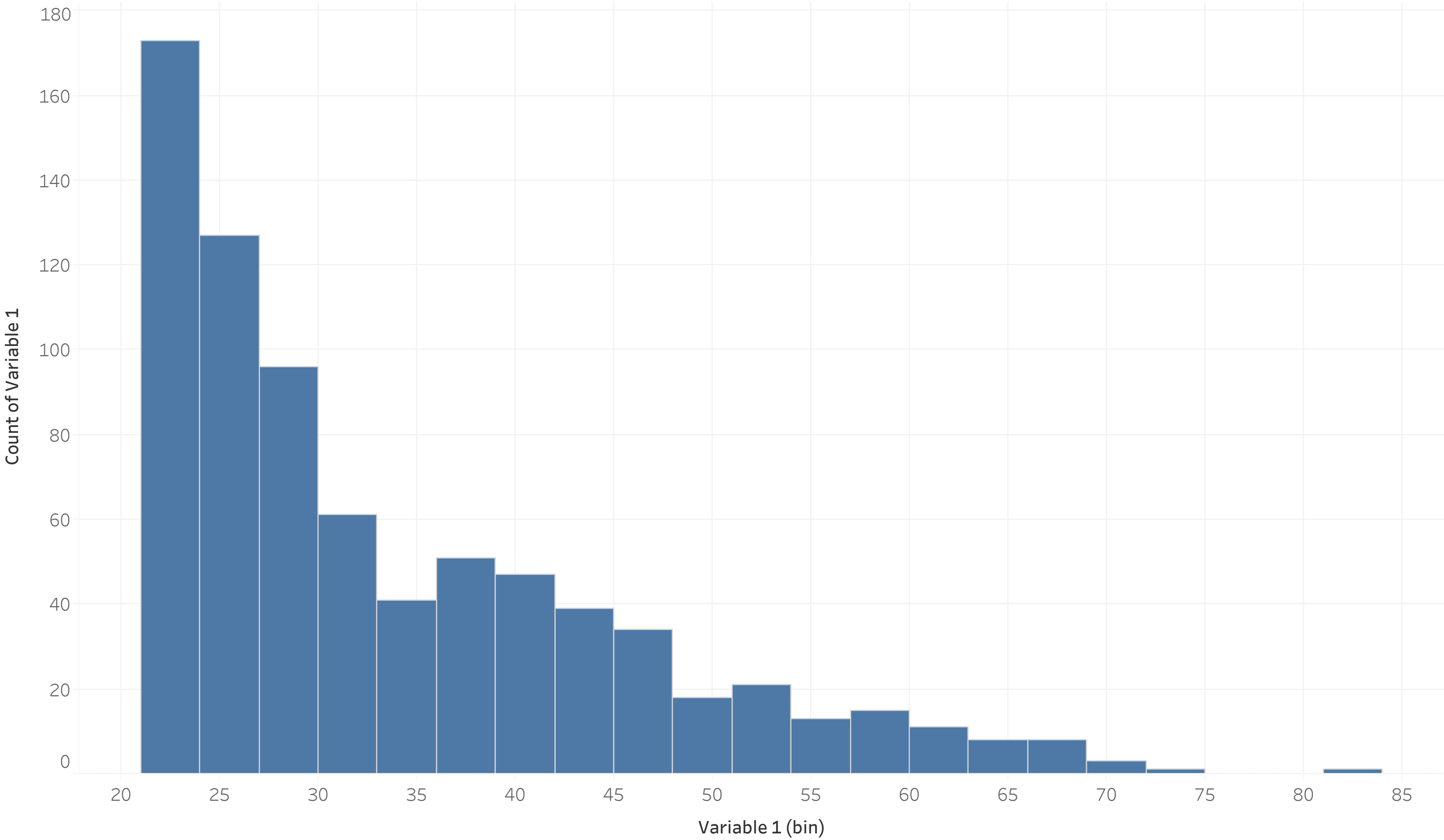
% of Total Count of Outcome and Dia/Non-Dia. Color shows details about Dia/Non-Dia. The marks are labeled by % of Total Count of Outcome and Dia/Non-Dia.

Age vs Glucose -Scatter Plots



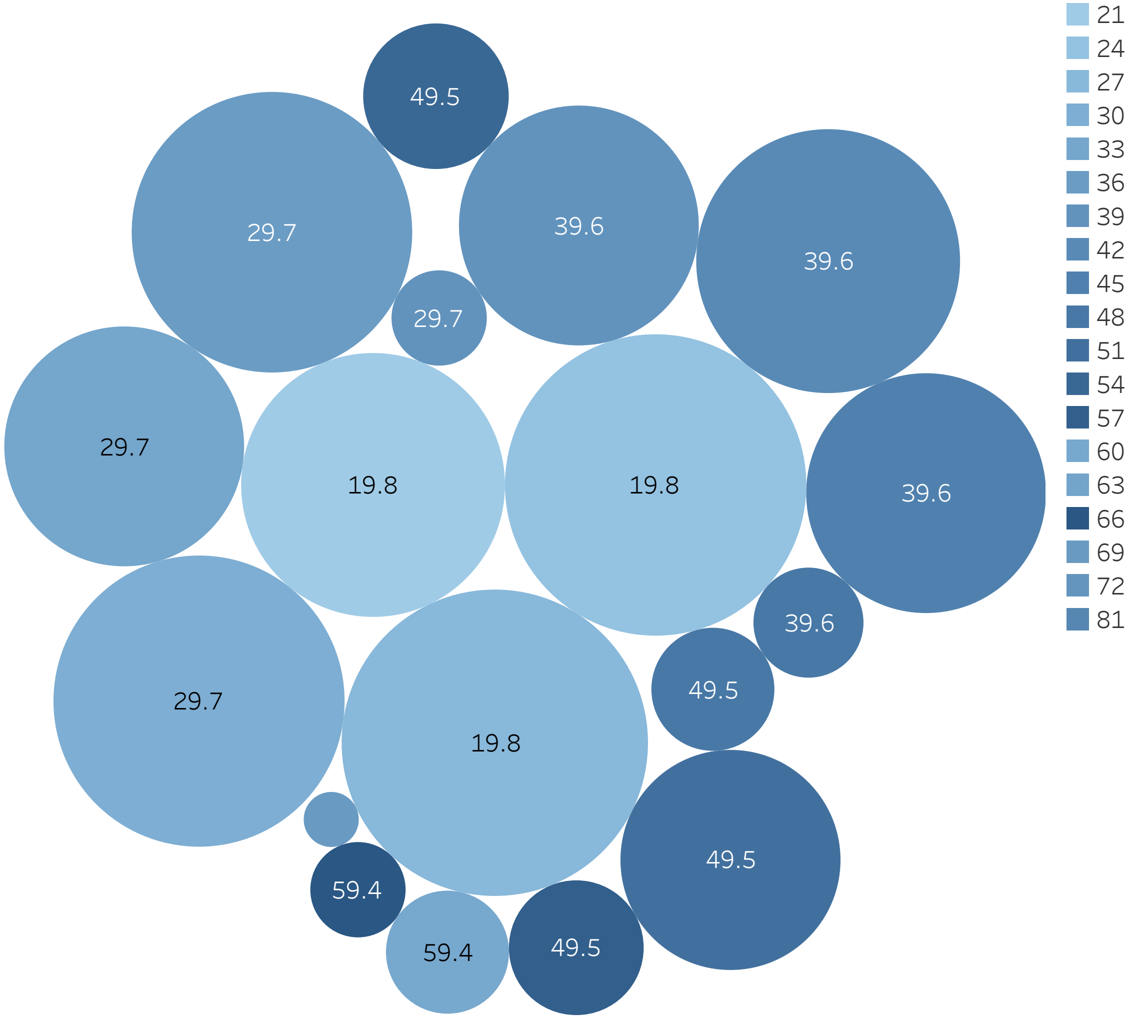
Variable 1 vs. Variable 2.

Age

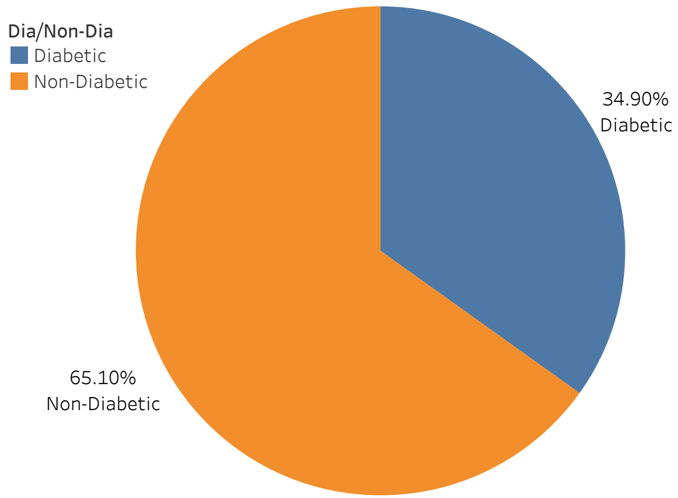


The trend of count of Variable 1 for Variable 1 (bin).

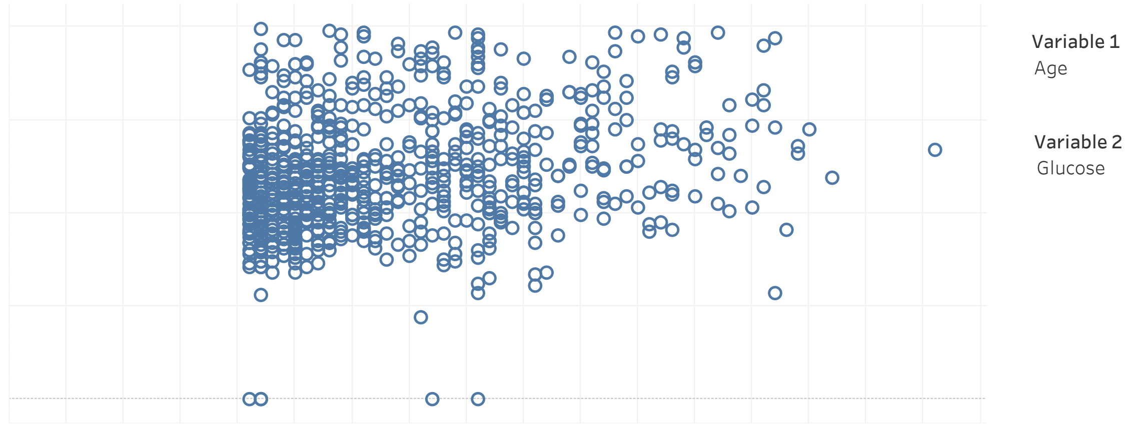
Age vs Outcome - Bubble Charts



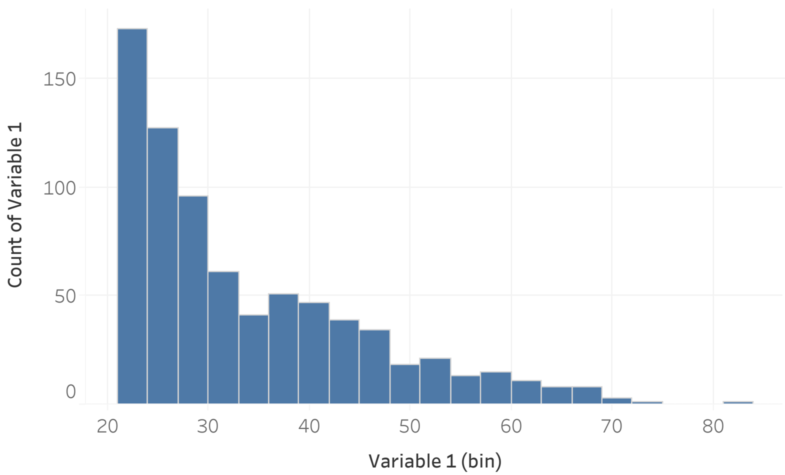
Variable 1 (bin) 2. Color shows details about Variable 1 (bin). Size shows sum of Outcome.
The marks are labeled by Variable 1 (bin) 2.



Age vs Glucose -Scatter Plots



Age



Age vs Outcome - Bubble Charts