Data Science Capstone: Healthcare

Problem Statement

- NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.
- The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.
- Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description

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

Variables	**Description**
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration in an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skinfold thickness (mm)
Insulin	Two hour serum insulin
BMI	Body Mass Index
DiabetesPedigreeFunction	Diabetes pedigree function
Age	Age in years

Outcome	Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

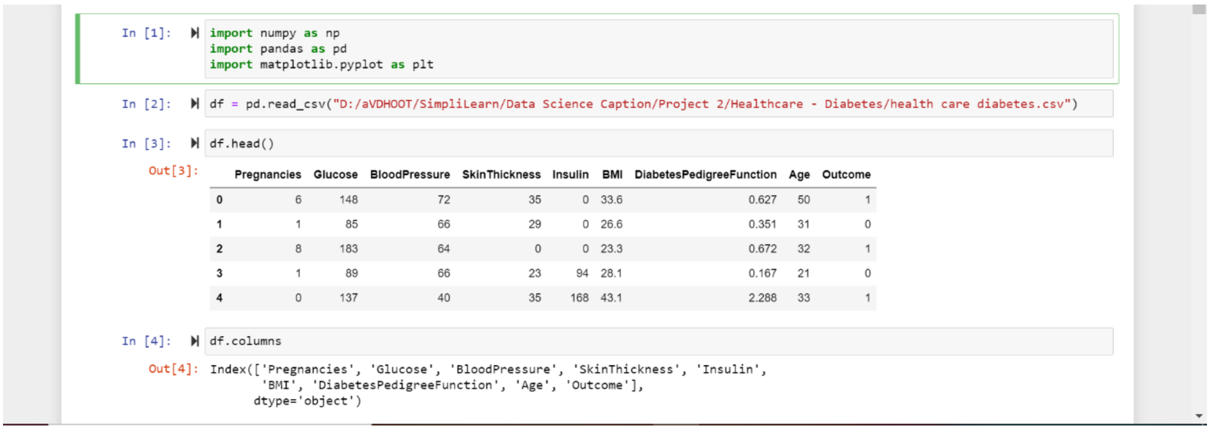
Project Task: Week 1

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

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv("D:/aVDHOOT/SimpliLearn/Data Science Caption/Project 2/Healthcare - Diabetes/health care diabetes.csv")

#Descriptive Analysis

df.head()

df.columns

df.isna().sum()/len(df)*100

df.describe()

df.info()
```



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

In [2]: df = pd.read_csv("D:/aVDHOOT/SimpliLearn/Data Science Caption/Project 2/Healthcare - Diabetes/health care diabetes.csv")

In [3]: df.head()

Out[3]:
```

	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 [4]: df.columns

Out[4]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

```
In [13]: df.isna().sum()/len(df)*100
Out[13]: Pregnancies                 0.0
         Glucose                     0.0
         BloodPressure               0.0
         SkinThickness               0.0
         Insulin                     0.0
         BMI                         0.0
         DiabetesPedigreeFunction    0.0
         Age                         0.0
         Outcome                     0.0
         dtype: float64

In [6]: df.shape
Out[6]: (768, 9)

In [7]: df.describe()
Out[7]:
```

	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

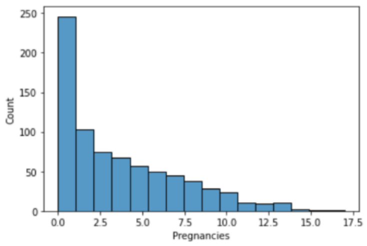
2. Visually explore these variables using histograms. Treat the missing values accordingly.

3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

import seaborn as sns

sns.histplot(df["Pregnancies"])

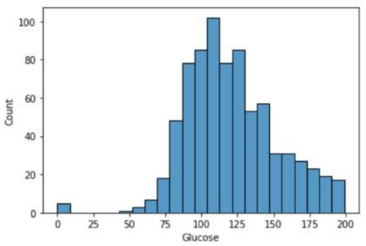
```
In [19]: import seaborn as sns

In [10]: sns.histplot(df["Pregnancies"])
Out[10]: <AxesSubplot:xlabel='Pregnancies', ylabel='Count'>
```



sns.histplot(df["Glucose"])

df["Glucose"].value_counts().head()

```
In [11]: sns.histplot(df["Glucose"])
Out[11]: <AxesSubplot:xlabel='Glucose', ylabel='Count'>
```

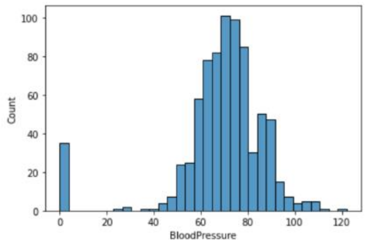



```
In [12]: df["Glucose"].value_counts().head()
Out[12]: 99     17
         100    17
         129    14
         125    14
         106    14
         Name: Glucose, dtype: int64
```

sns.histplot(df["BloodPressure"])

df["BloodPressure"].value_counts().head(7)

```
In [13]:  sns.histplot(df["BloodPressure"])
Out[13]: <AxesSubplot:xlabel='BloodPressure', ylabel='Count'>
```

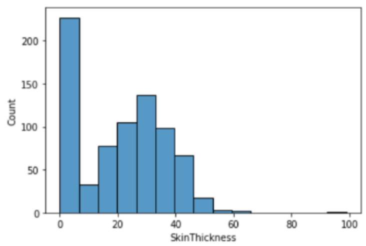



```
In [14]:  df["BloodPressure"].value_counts().head(7)
Out[14]: 70    57
         74    52
         78    45
         68    45
         72    44
```

sns.histplot(df["SkinThickness"])

df["SkinThickness"].value_counts().head(7)

```
In [15]:  sns.histplot(df["SkinThickness"])
Out[15]: <AxesSubplot:xlabel='SkinThickness', ylabel='Count'>
```

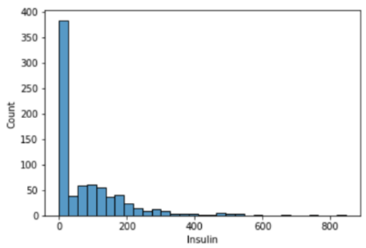



```
In [16]:  df["SkinThickness"].value_counts().head(7)
Out[16]: 0     227
         32     31
         30     27
         27     23
         23     22
```

sns.histplot(df["Insulin"])

df["Insulin"].value_counts().head(7)

```
In [17]:  sns.histplot(df["Insulin"])
Out[17]: <AxesSubplot:xlabel='Insulin', ylabel='Count'>
```

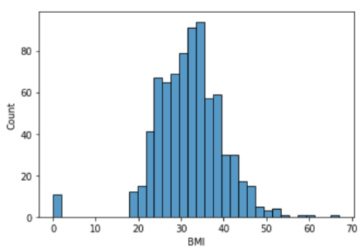



```
In [18]:  df["Insulin"].value_counts().head(7)
Out[18]: 0      374
         105     11
         140      9
         130      9
         120      8
```

sns.histplot(df["BMI"])

df["BMI"].value_counts().head(7)

```
In [19]:  sns.histplot(df["BMI"])
Out[19]: <AxesSubplot:xlabel='BMI', ylabel='Count'>
```



Project Task: Week 2

Data Exploration:

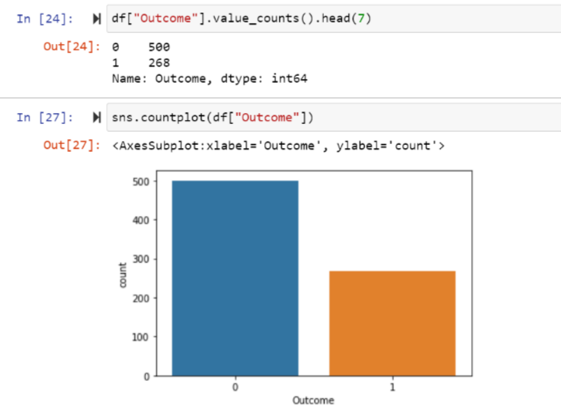
1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

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

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

df["Outcome"].value_counts().head(7)

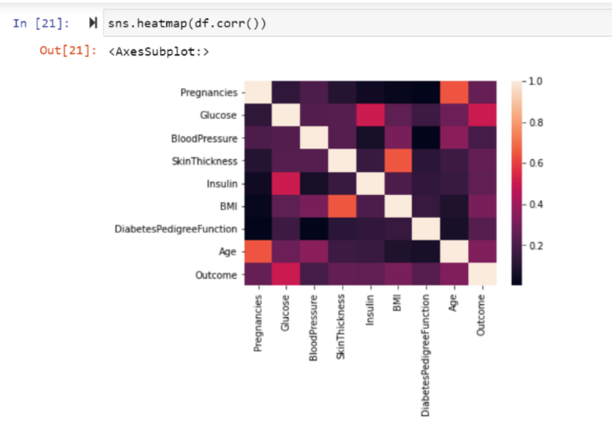
sns.countplot(df["Outcome"])



sns.pairplot(df)

df.corr()

sns.heatmap(df.corr())



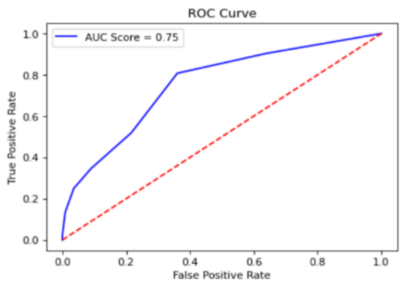
Project Task: Week 3
Data Modeling:

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.

```
# KNN

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=7)

knn_model.fit(X_train,y_train)

knn_pred=knn_model.predict(X_test)

print("Model Validation\n")

print("Accuracy Score of KNN Model::")

print(metrics.accuracy_score(y_test,knn_pred))

print("\n","Classification Report::")

print(metrics.classification_report(y_test,knn_pred),'\n')

print("\n","ROC Curve")

knn_prob=knn_model.predict_proba(X_test)

knn_prob1=knn_prob[:,1]

fpr,tpr,thresh=metrics.roc_curve(y_test,knn_prob1)

roc_auc_knn=metrics.auc(fpr,tpr)

plt.figure(dpi=80)

plt.title("ROC Curve")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_knn)

plt.plot(fpr,fpr,'r--',color='red')

plt.legend()
```

```
Model Validation

Accuracy Score of KNN Model::
0.7300613496932515

 Classification Report::
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       111
           1       0.64      0.35      0.45        52

    accuracy                           0.73       163
   macro avg       0.70      0.63      0.64       163
weighted avg       0.71      0.73      0.70       163
```



from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("Model Validation\n")

print("Accuracy Score of Logistic Model::")

print(metrics.accuracy_score(y_test,y_pred))

print("\n","Classification Report::")

print(metrics.classification_report(y_test,y_pred),'\n')

print("\n","ROC Curve")

y_prob=classifier.predict_proba(X_test)

y_prob1=y_prob[:,1]

fpr,tpr,thresh=metrics.roc_curve(y_test,y_prob1)

roc_auc=metrics.auc(fpr,tpr)

plt.figure(dpi=80)

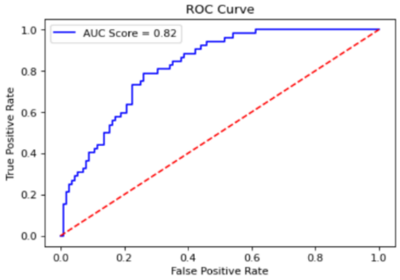
plt.title("ROC Curve")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

```
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc)

plt.plot(fpr,fpr,'r--',color='red')

plt.legend()
```

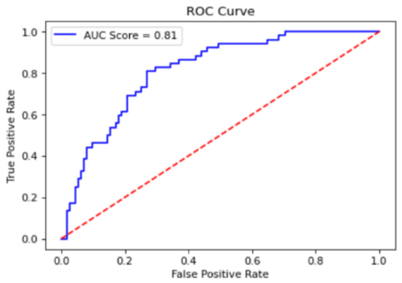
```
Model Validation

Accuracy Score of Logistic Model::
0.7361963190184049

 Classification Report::
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       111
           1       0.63      0.42      0.51        52

    accuracy                           0.74       163
   macro avg       0.70      0.65      0.66       163
weighted avg       0.72      0.74      0.72       163
```




```
#Naive Bayes

from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X_train, y_train)



# making predictions on the testing set

y_pred = gnb.predict(X_test)



# comparing actual response values (y_test) with predicted response values (y_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)

# Predicton on test with giniIndex

print("Model Validation\n")

print("Accuracy Score of Decision Tree Model::")

print(metrics.accuracy_score(y_test,y_pred))

print("\n","Classification Report::")
```

```
print(metrics.classification_report(y_test,y_pred),'\n')

print("\n","ROC Curve")

y_prob=gnb.predict_proba(X_test)

y_prob1=y_prob[:,1]

fpr,tpr,thresh=metrics.roc_curve(y_test,y_prob1)

roc_auc=metrics.auc(fpr,tpr)

plt.figure(dpi=80)

plt.title("ROC Curve")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc)

plt.plot(fpr,fpr,'r--',color='red')

plt.legend()
```

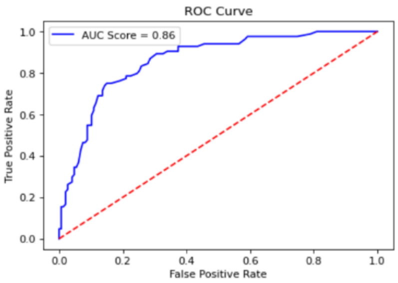
```
Model Validation

Accuracy Score of Decision Tree Model::
0.7484662576687117

 Classification Report::
              precision    recall  f1-score   support

           0       0.81      0.82      0.82       111
           1       0.61      0.60      0.60        52

    accuracy                           0.75       163
   macro avg       0.71      0.71      0.71       163
weighted avg       0.75      0.75      0.75       163
```




```
#random forest

from sklearn.ensemble import RandomForestClassifier

# creating a RF classifier

clf = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset

# fit function is used to train the model using the training sets as parameters
```

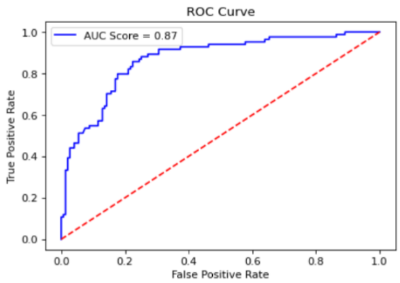
```
clf.fit(X_train, y_train)

# performing predictions on the test dataset

y_pred = clf.predict(X_test)

# metrics are used to find accuracy or error

from sklearn import metrics

# using metrics module for accuracy calculation

print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, y_pred))

print("Model Validation\n")

print("Accuracy Score of Decision Model::")

print(metrics.accuracy_score(y_test,y_pred))

print("\n","Classification Report::")

print(metrics.classification_report(y_test,y_pred),'\n')

print("\n","ROC Curve")

y_prob=clf.predict_proba(X_test)

y_prob1=y_prob[:,1]

fpr,tpr,thresh=metrics.roc_curve(y_test,y_prob1)

roc_auc_dt=metrics.auc(fpr,tpr)

plt.figure(dpi=80)

plt.title("ROC Curve")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_dt)

plt.plot(fpr,fpr,'r--',color='red')

plt.legend()
```

```
Model Validation

Accuracy Score of Decision Model::
0.7965367965367965

 Classification Report::
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       147
           1       0.77      0.63      0.69        84

    accuracy                           0.80       231
   macro avg       0.79      0.76      0.77       231
weighted avg       0.79      0.80      0.79       231
```




```
#SVM

from sklearn.svm import SVC

svm = SVC(probability=True)

svm.fit(X_train,y_train)

# Predicton on test with giniIndex

y_pred = svm.predict(X_test)

print("Model Validation\n")

print("Accuracy Score of SVM Model::")

print(metrics.accuracy_score(y_test,y_pred))

print("\n","Classification Report::")

print(metrics.classification_report(y_test,y_pred),'\n')

y_prob=svm.predict_proba(X_test)

y_prob1=y_prob[:,1]

fpr,tpr,thresh=metrics.roc_curve(y_test,y_prob1)

roc_auc_dt=metrics.auc(fpr,tpr)

plt.figure(dpi=80)

plt.title("ROC Curve")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_dt)

plt.plot(fpr,fpr,'r--',color='red')

plt.legend()
```

```
Model Validation

Accuracy Score of Decision Model::
0.7965367965367965

 Classification Report::
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       147
           1       0.77      0.63      0.69        84

    accuracy                           0.80       231
   macro avg       0.79      0.76      0.77       231
weighted avg       0.79      0.80      0.79       231
```

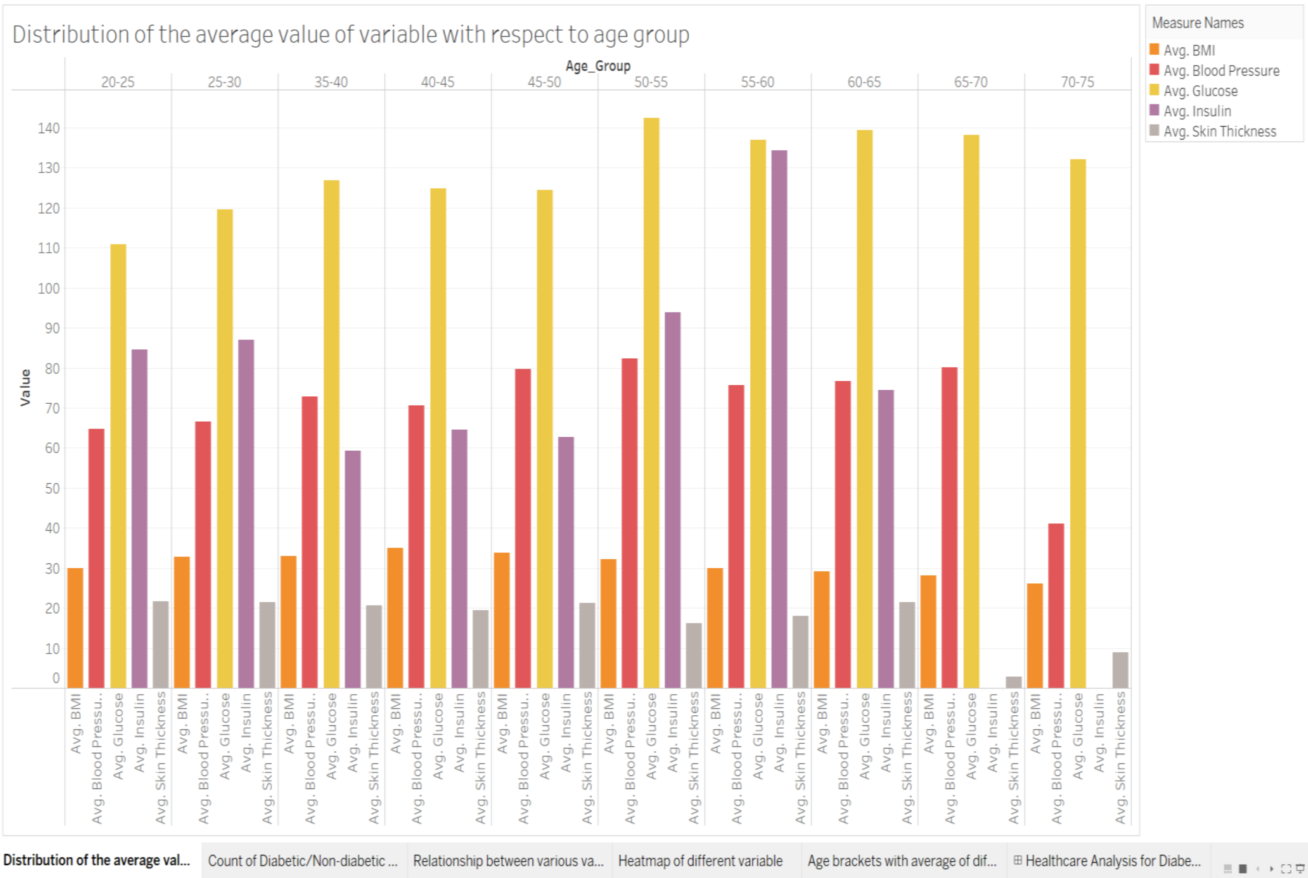


Project Task: Week 4

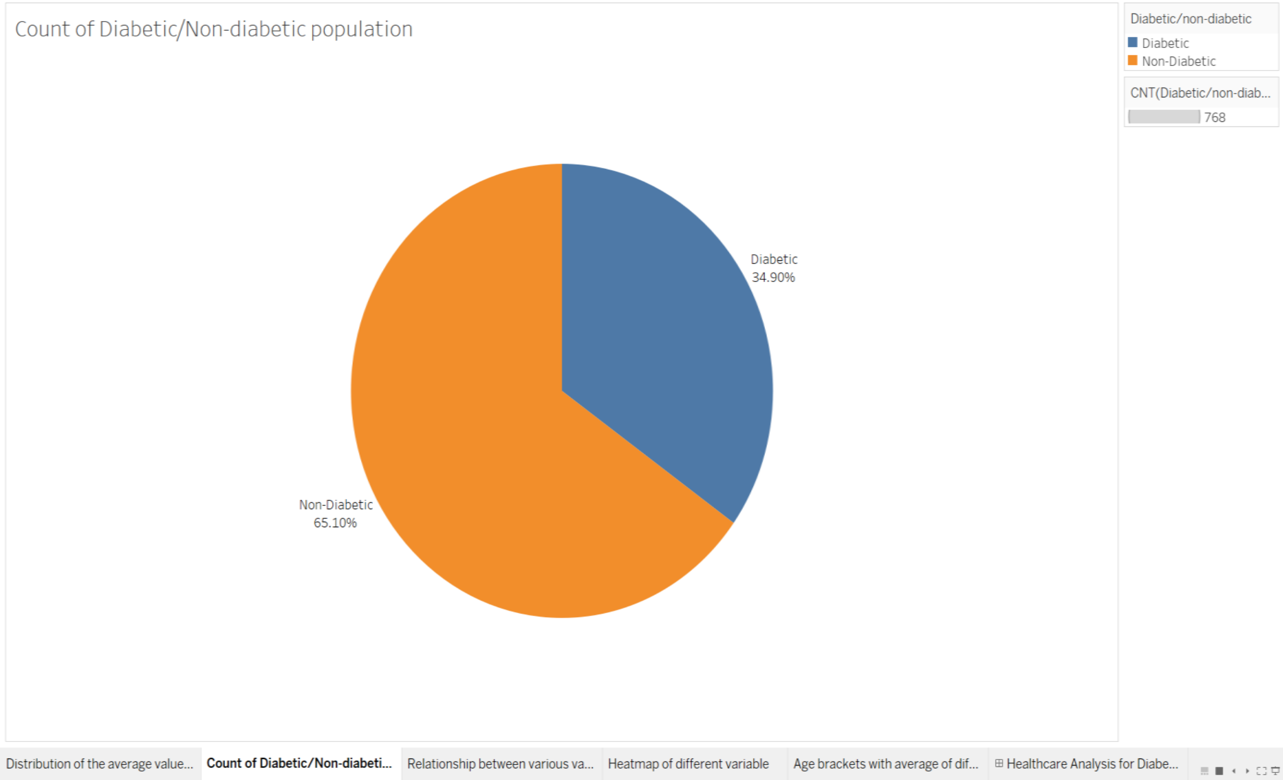
Data Reporting:

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

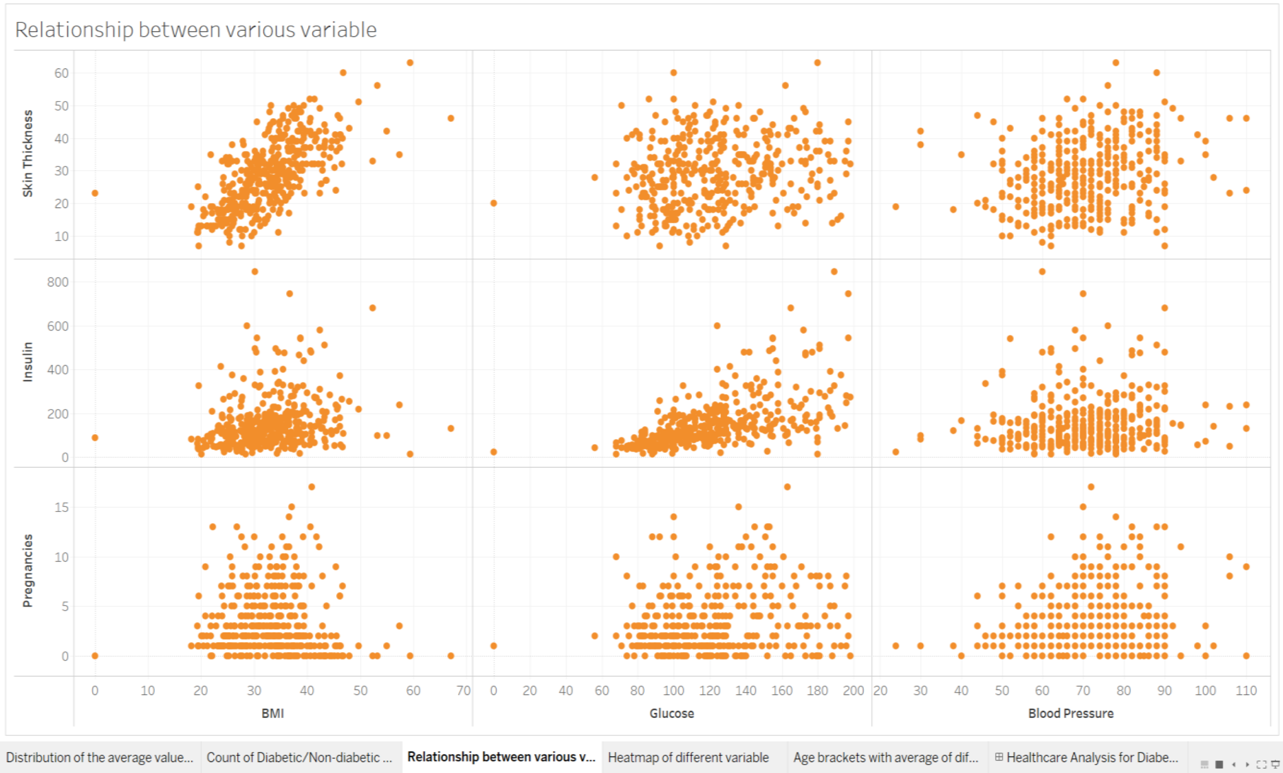
a. Histogram or frequency charts to analyze the distribution of the data



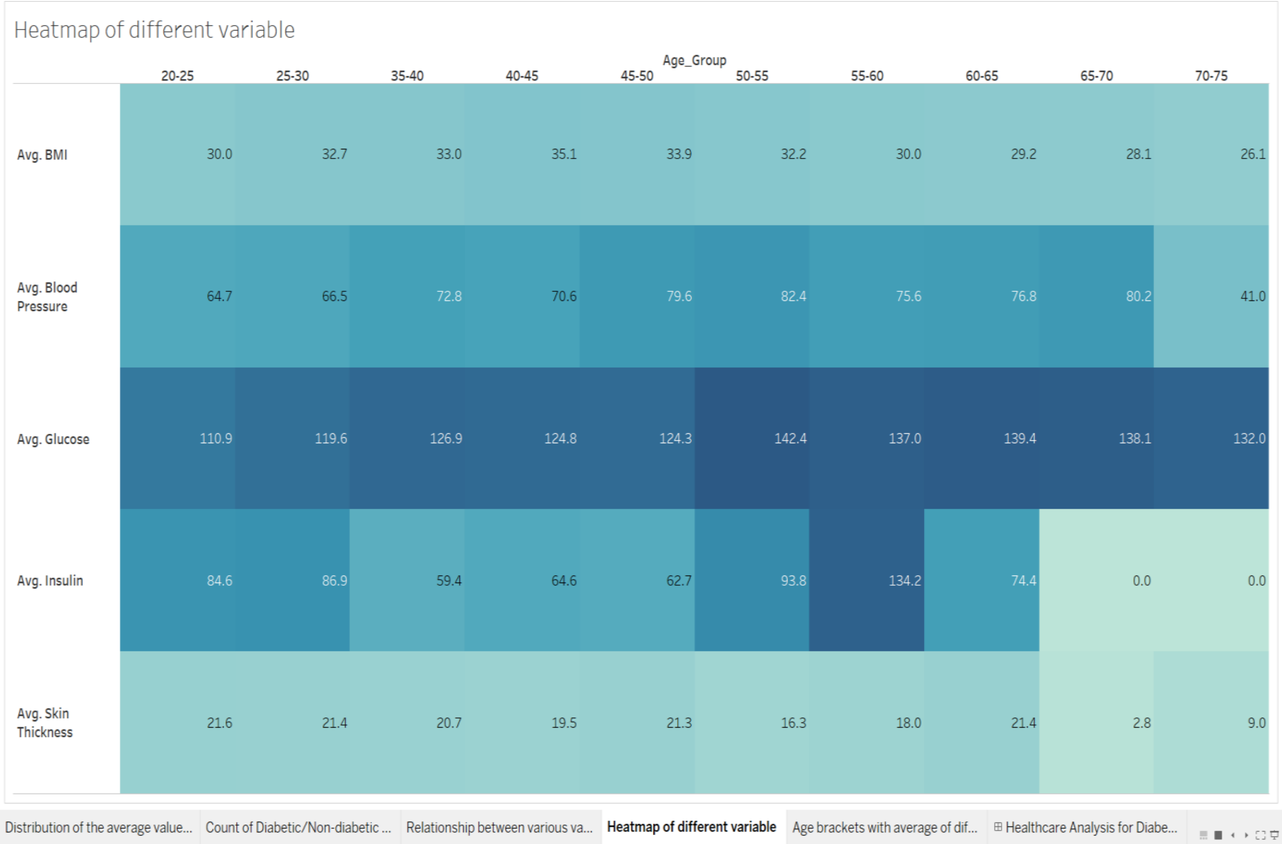
b. Pie chart to describe the diabetic or non-diabetic population



c. Scatter charts between relevant variables to analyze the relationships



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.

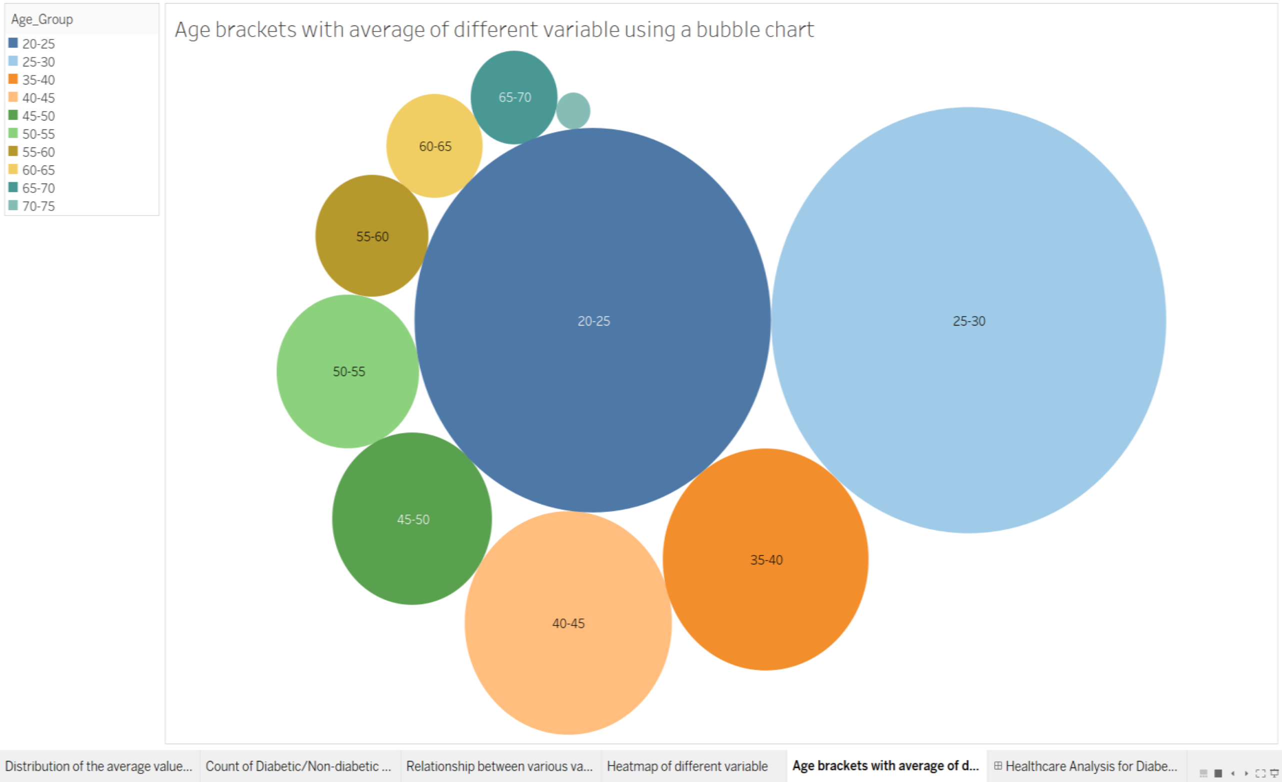


Tableau Dashboard: