Data Capstone-Healthcare

July 10, 2023

1 Capstone Project: 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.

2 Solution:

2.1 Week 1:

2.1.1 Data Exploration:

(1) Read Data and Perform descriptive analysis:

[1]:
```
# Importing required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
from warnings import filterwarnings
filterwarnings('ignore')
%matplotlib inline
```

[2]:
```
#reading the dataset
df=pd.read_csv('diabetes.csv')
```

```
df.head()
```

```
[2]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
     0            6      148             72             35        0  33.6
     1            1       85             66             29        0  26.6
     2            8      183             64              0        0  23.3
     3            1       89             66             23       94  28.1
     4            0      137             40             35      168  43.1

        DiabetesPedigreeFunction  Age  Outcome
     0                     0.627   50        1
     1                     0.351   31        0
     2                     0.672   32        1
     3                     0.167   21        0
     4                     2.288   33        1
```

```
[ ]:
```

```
[3]: #shape of the dataset
     df.shape
```

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

```
[4]: # checking for missing values
     df.isnull().sum()
```

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

```
[5]: # checking dublicate rows in dataset
     df.duplicated().sum()
```

```
[5]: 0
```

```
[6]: # getting information about dataset
     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
```

[7]:
```
# getting statical information about our dataset
df.describe().transpose()
```

[7]:
```
                          count        mean         std     min       25%  \
Pregnancies               768.0    3.845052    3.369578   0.000   1.00000
Glucose                   768.0  120.894531   31.972618   0.000  99.00000
BloodPressure             768.0   69.105469   19.355807   0.000  62.00000
SkinThickness             768.0   20.536458   15.952218   0.000   0.00000
Insulin                   768.0   79.799479  115.244002   0.000   0.00000
BMI                       768.0   31.992578    7.884160   0.000  27.30000
DiabetesPedigreeFunction  768.0    0.471876    0.331329   0.078   0.24375
Age                       768.0   33.240885   11.760232  21.000  24.00000
Outcome                   768.0    0.348958    0.476951   0.000   0.00000

                               50%        75%     max
Pregnancies                 3.0000    6.00000   17.00
Glucose                   117.0000  140.25000  199.00
BloodPressure              72.0000   80.00000  122.00
SkinThickness              23.0000   32.00000   99.00
Insulin                    30.5000  127.25000  846.00
BMI                        32.0000   36.60000   67.10
DiabetesPedigreeFunction    0.3725    0.62625    2.42
Age                        29.0000   41.00000   81.00
Outcome                     0.0000    1.00000    1.00
```

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

[8]:
```
df[df['Glucose']==0]
```

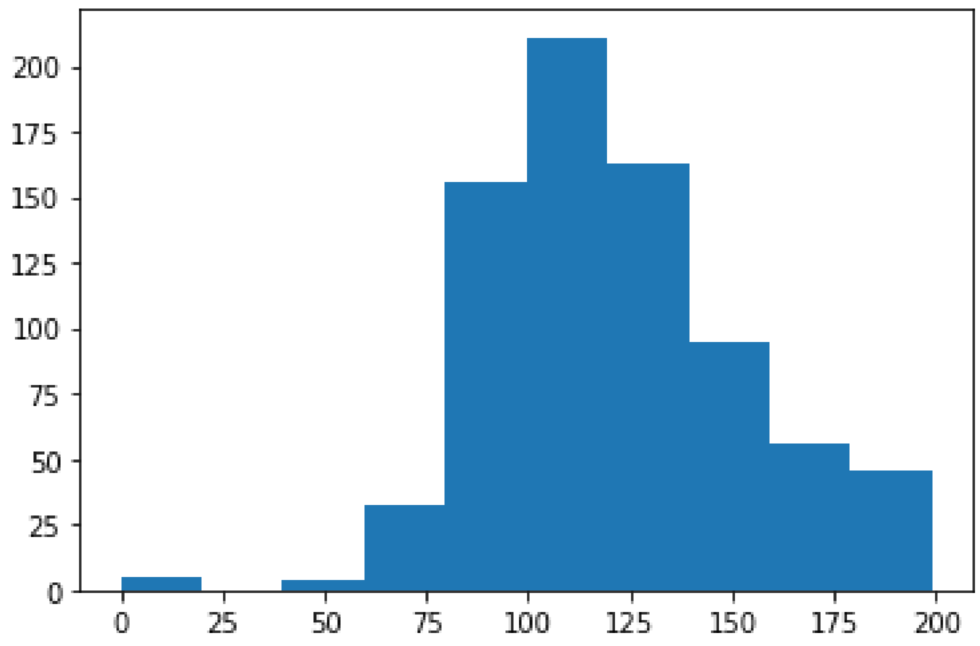
[8]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
75             1        0             48             20        0  24.7
182            1        0             74             20       23  27.7
```

```
342            1        0              68             35         0  32.0
349            5        0              80             32         0  41.0
502            6        0              68             41         0  39.0

     DiabetesPedigreeFunction  Age  Outcome
75                      0.140   22        0
182                     0.299   21        0
342                     0.389   22        0
349                     0.346   37        1
502                     0.727   41        1
```

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

```
[9]: plt.hist(df['Glucose'])
```

```
[9]: (array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
      array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
             179.1, 199. ]),
      <BarContainer object of 10 artists>)
```



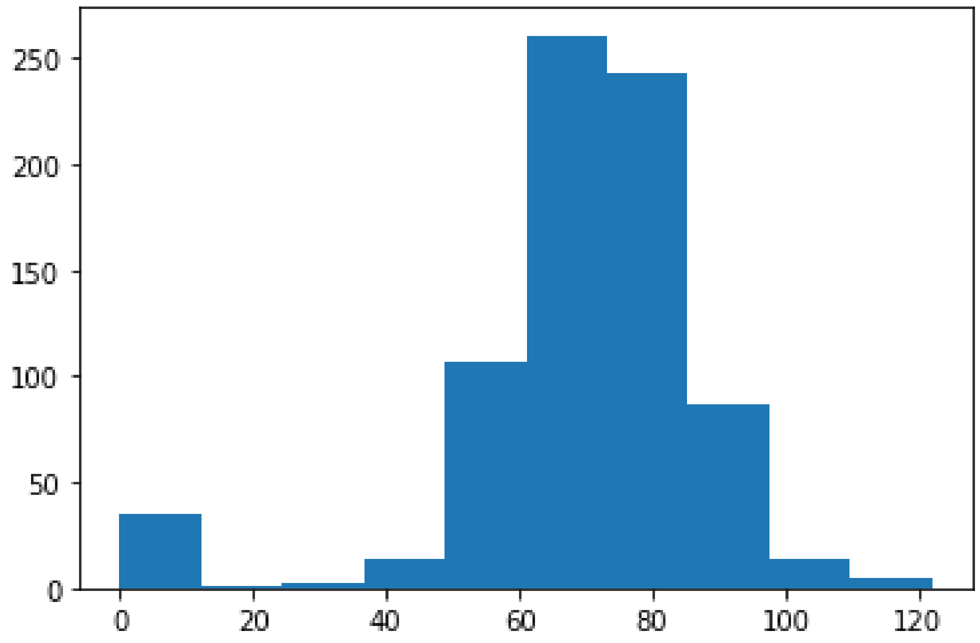
```
[10]: df[df['BloodPressure']==0].head()
```

```
[10]:     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
      7             10      115              0              0        0  35.3
      15             7      100              0              0        0  30.0
      49             7      105              0              0        0   0.0
```

```
60             2      84              0              0        0   0.0
78             0     131              0              0        0  43.2

    DiabetesPedigreeFunction  Age  Outcome
7                      0.134   29        0
15                     0.484   32        1
49                     0.305   24        0
60                     0.304   21        0
78                     0.270   26        1
```

[11]:
```
plt.hist(df['BloodPressure'])
```

[11]:
```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
 <BarContainer object of 10 artists>)
```



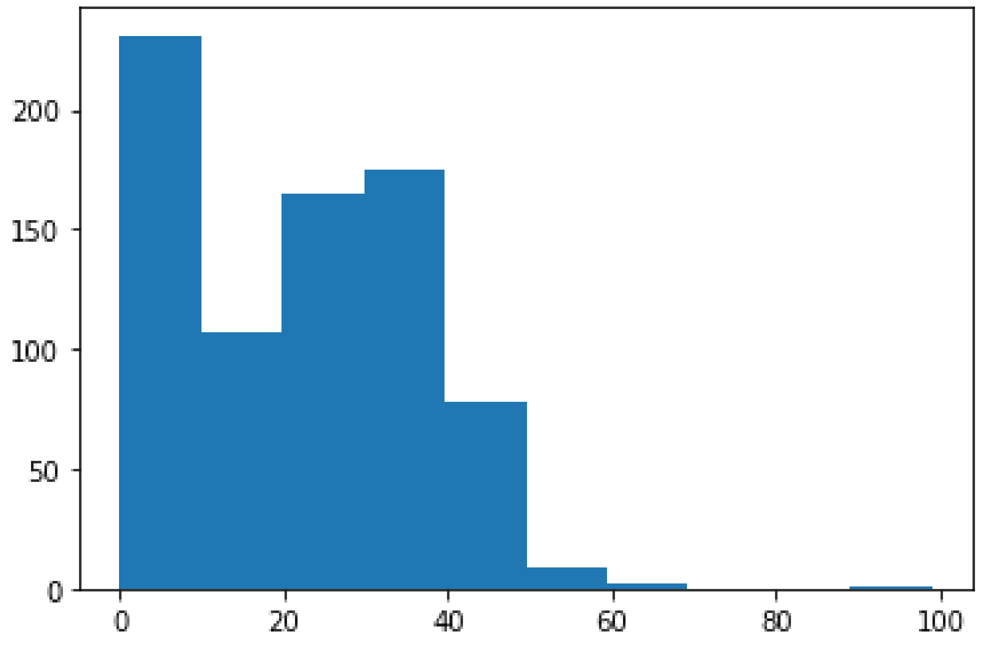
[12]:
```
df[df['SkinThickness']==0].head()
```

[12]:
```
    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
2             8      183             64              0        0  23.3
5             5      116             74              0        0  25.6
7            10      115              0              0        0  35.3
9             8      125             96              0        0   0.0
10            4      110             92              0        0  37.6
```

```
   DiabetesPedigreeFunction  Age  Outcome
2                     0.672   32        1
5                     0.201   30        0
7                     0.134   29        0
9                     0.232   54        1
10                    0.191   30        0
```

[13]:
```
plt.hist(df['SkinThickness'])
```

[13]:
```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```



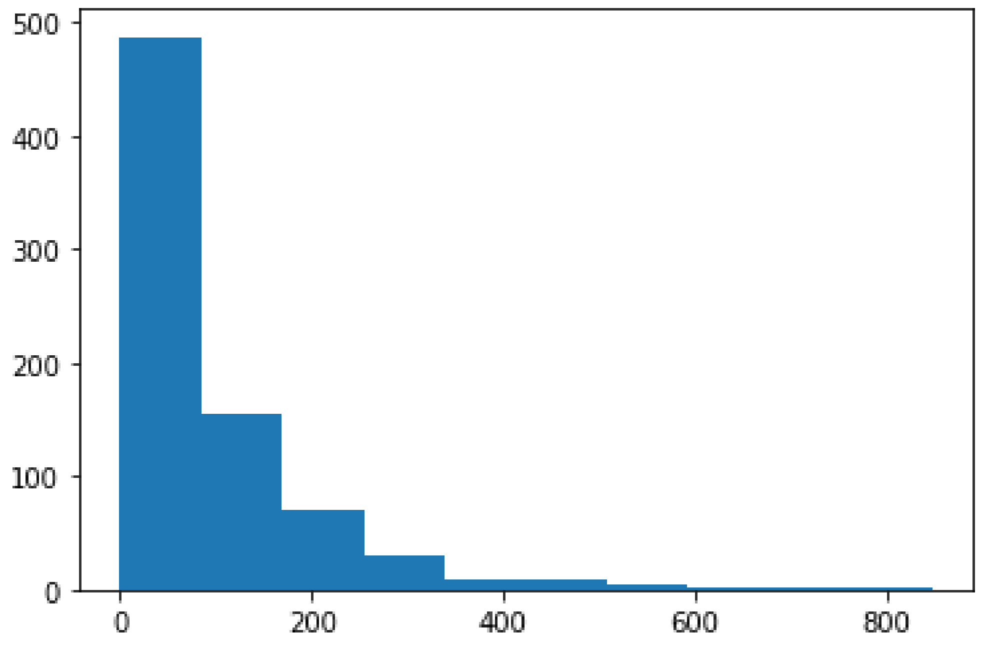
[14]:
```
df[df['Insulin']==0].head()
```

[14]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
5            5      116             74              0        0  25.6
7           10      115              0              0        0  35.3

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

```
5                       0.201   30         0
7                       0.134   29         0
```

[15]:
```
plt.hist(df['Insulin'])
```

[15]:
```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```

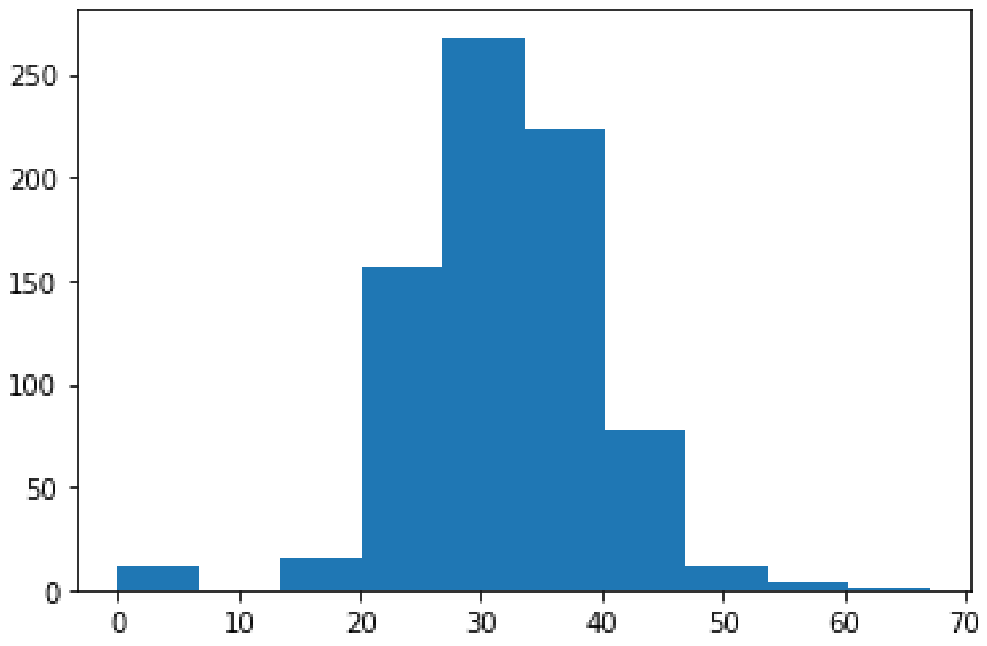


[16]:
```
df[df['BMI']==0].head()
```

[16]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
9              8      125             96              0        0  0.0
49             7      105              0              0        0  0.0
60             2       84              0              0        0  0.0
81             2       74              0              0        0  0.0
145            0      102             75             23        0  0.0

     DiabetesPedigreeFunction  Age  Outcome
9                       0.232   54        1
49                      0.305   24        0
60                      0.304   21        0
81                      0.102   22        0
145                     0.572   21        0
```

```
[17]: plt.hist(df['BMI'])
```

```
[17]: (array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
       array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
              60.39, 67.1 ]),
       <BarContainer object of 10 artists>)
```



inference: we clearly observe that column like BP, Glucose etc have 0 value. It isn't medically possible for some data record to have 0 value such as Blood Pressure or Glucose levels. Hence we replace them with the median value of that particular column.

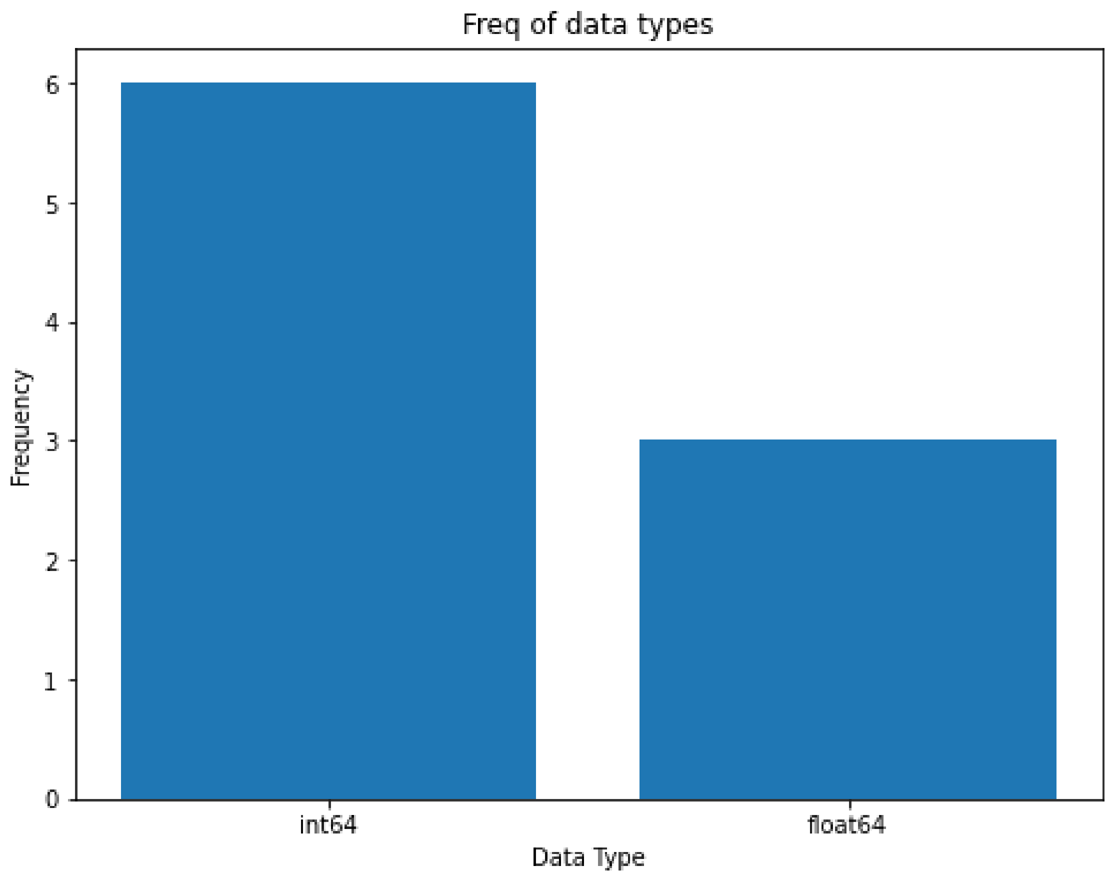
```
[18]: #replacing 0 value with median

      df['Glucose']=df['Glucose'].replace(0,df['Glucose'].median())

      df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].median())

      df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].median())

      df['Insulin']=df['Insulin'].replace(0,df['Insulin'].median())

      df['BMI']=df['BMI'].replace(0,df['BMI'].median())
```

(3)Create a count (frequency) plot describing the data types and the count of variables.

```
[19]: vc=df.dtypes.value_counts()
```

[20]:
```
plt.figure(figsize=(8,6))
plt.bar(vc.index.astype(str),vc.values)
plt.xlabel('Data Type')
plt.ylabel('Frequency')
plt.title('Freq of data types')
```

[20]:
```
Text(0.5, 1.0, 'Freq of data types')
```



2.2 Week 2:

2.2.1 Data Exploration:

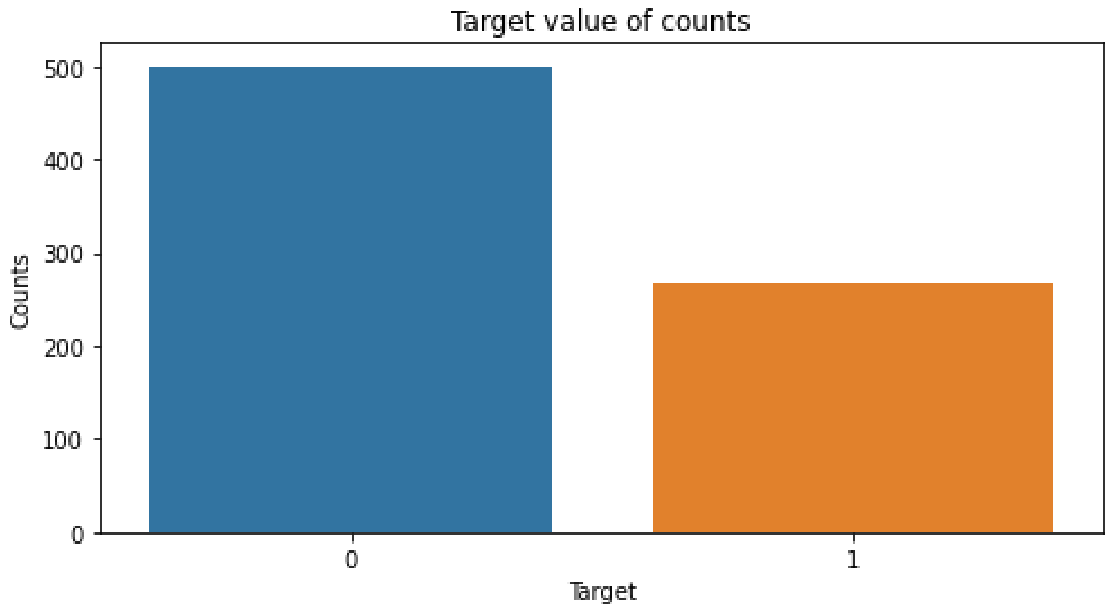
(1) Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action:

[21]:
```
target_count=df['Outcome'].value_counts()
target_count
```

[21]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[ ]:
```

```

[22]:
```
# plot a counter plot to better understand our target feature
plt.figure(figsize=(8,4))
sns.countplot(x = 'Outcome',data = df)
plt.xlabel('Target')
plt.ylabel('Counts')
plt.title('Target value of counts')
plt.show()
```

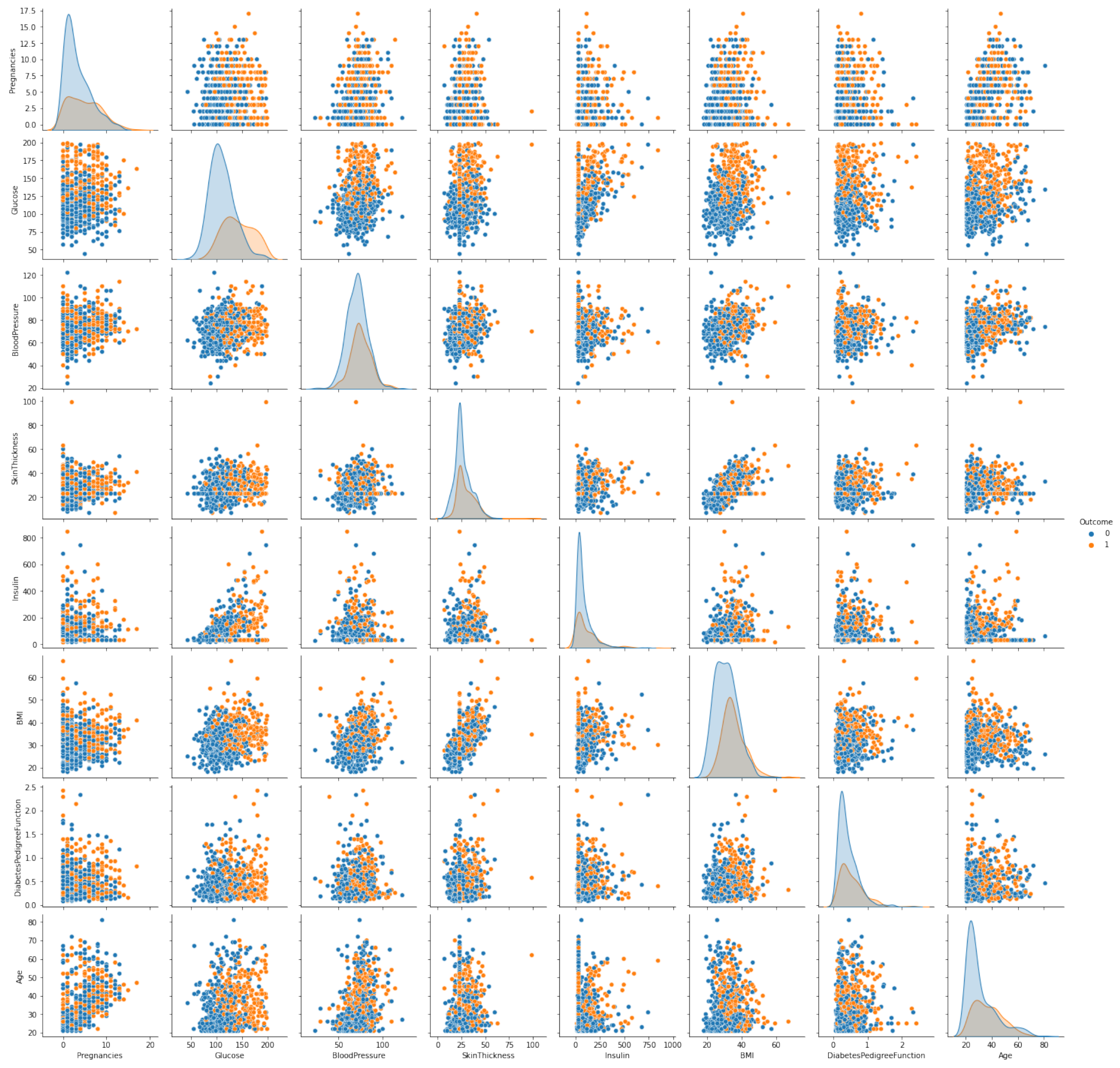


Inference: Our target varaible have only two class. 0 and 1. We can see that we have imbalacne dataset. We have to make balance by using sampling technique to avoid overfitting.

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

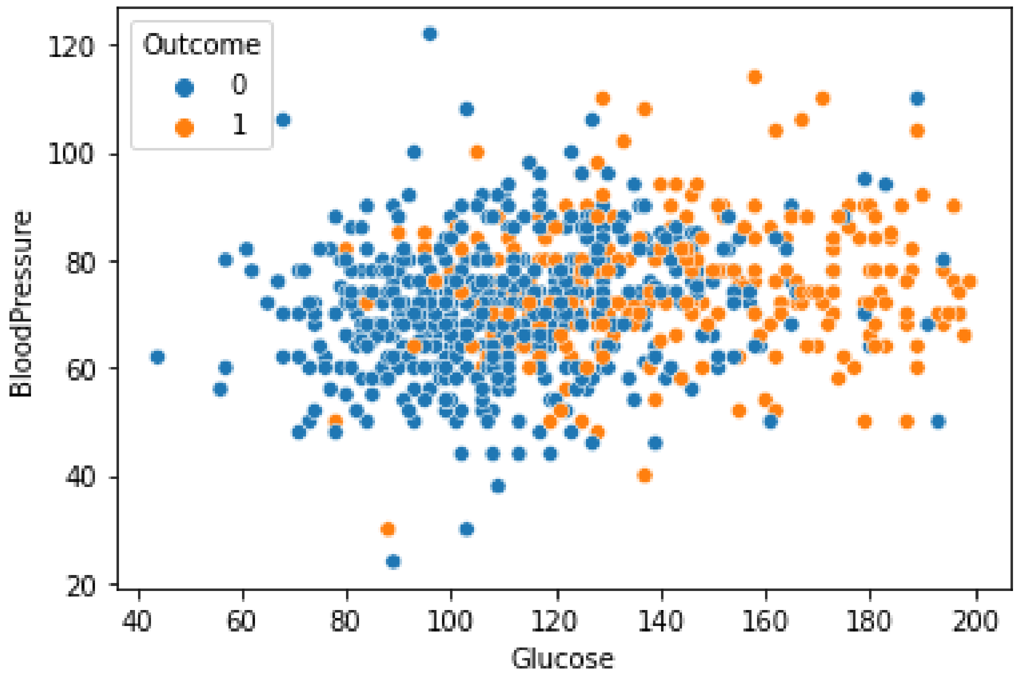
[23]:
```
#creating pair plot
import seaborn as sns
sns.pairplot(df,hue='Outcome')
```

[23]:
```
<seaborn.axisgrid.PairGrid at 0x7fe5b3de1e10>
```



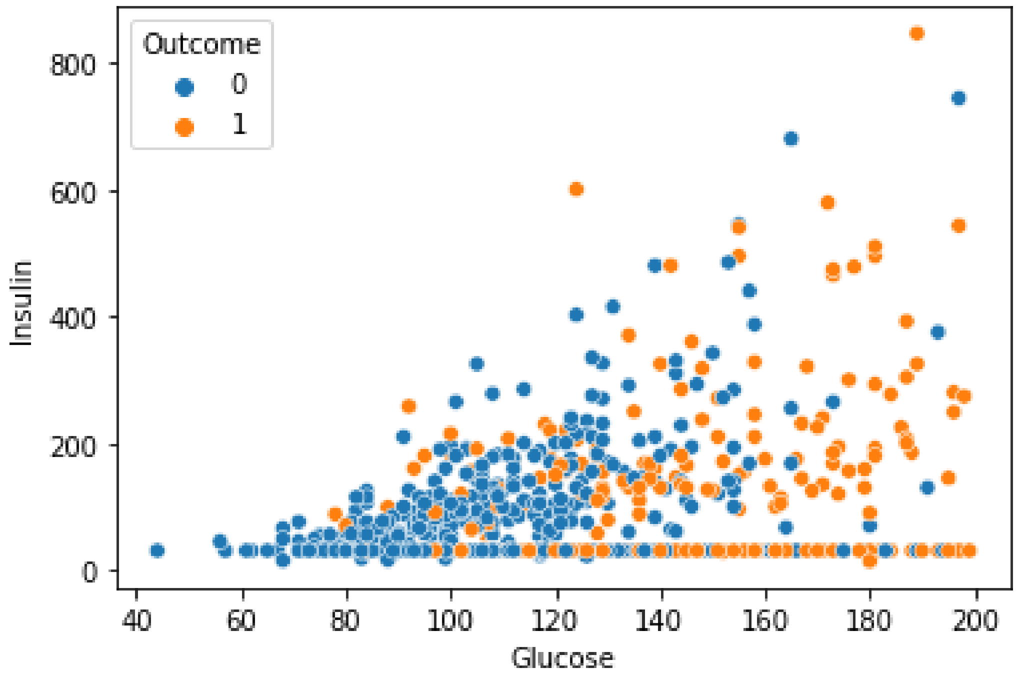
```
[24]: #let's find the relationship between independent and dependent numerical␣
       ↪features
      sns.scatterplot(x='Glucose',y='BloodPressure',data=df,hue='Outcome')
```

```
[24]: <AxesSubplot:xlabel='Glucose', ylabel='BloodPressure'>
```



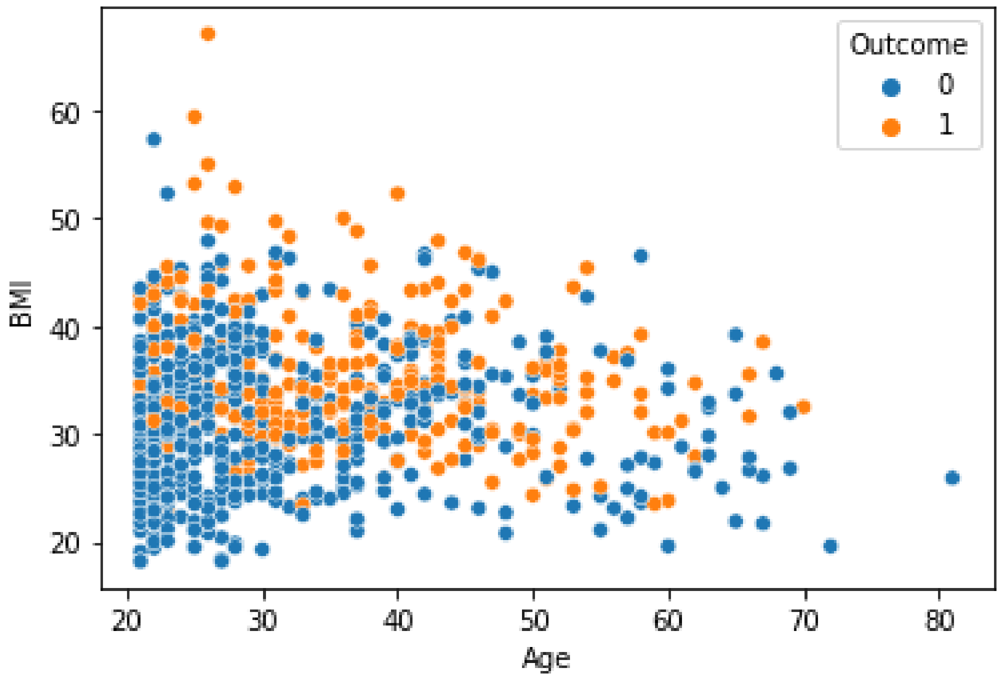
[25]:
```
sns.scatterplot(y='Insulin',x='Glucose',data=df,hue='Outcome')
```

[25]:
```
<AxesSubplot:xlabel='Glucose', ylabel='Insulin'>
```



```
[26]: sns.scatterplot(y='BMI',x='Age',data=df,hue='Outcome')
```

```
[26]: <AxesSubplot:xlabel='Age', ylabel='BMI'>
```



We have some observations from above scatter plot of pairs of features: * **Glucose** alone is impressively good to distinguish between the **Outcome** classes. * **Age** is also able to distinguish between classes to some extent

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

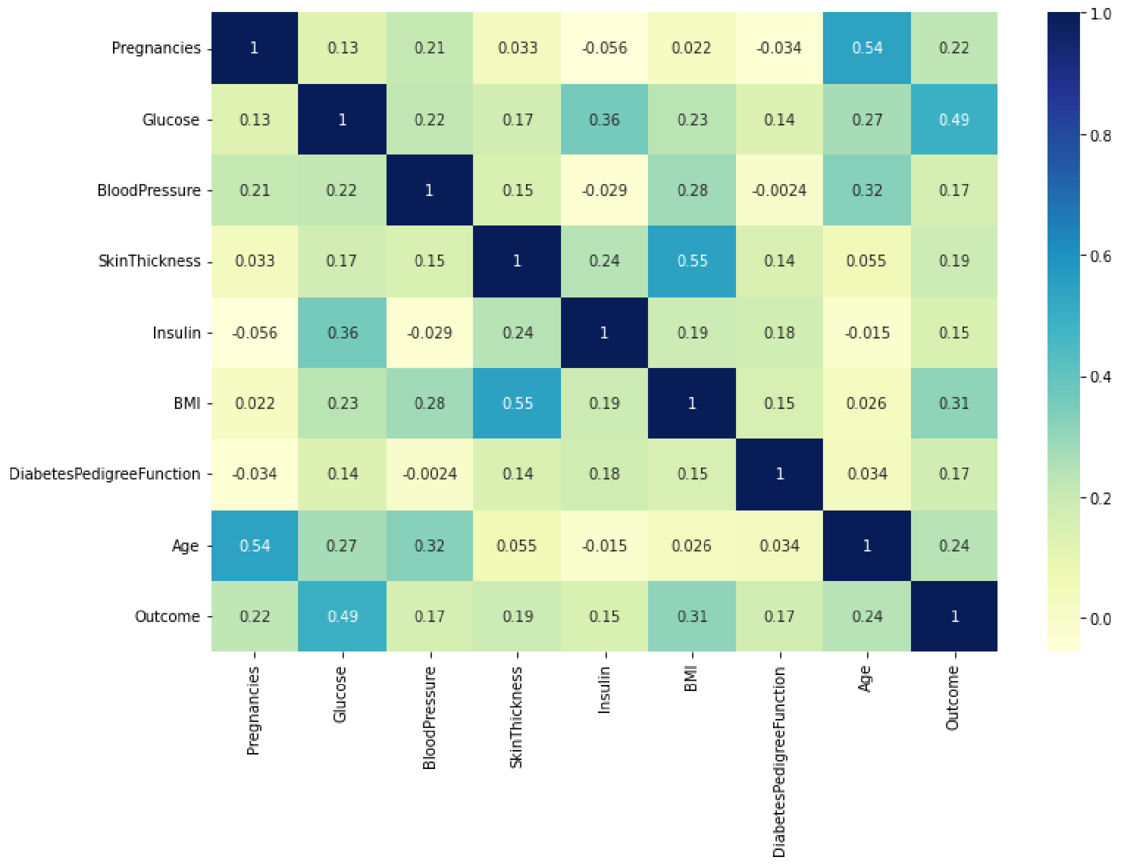
```
[27]: # Finding the correlation between each features
      df.corr()
```

```
[27]:                           Pregnancies   Glucose  BloodPressure  SkinThickness  \
      Pregnancies                  1.000000  0.128213       0.208615       0.032568
      Glucose                      0.128213  1.000000       0.218937       0.172143
      BloodPressure                0.208615  0.218937       1.000000       0.147809
      SkinThickness                0.032568  0.172143       0.147809       1.000000
      Insulin                     -0.055697  0.357573      -0.028721       0.238188
      BMI                          0.021546  0.231400       0.281132       0.546951
      DiabetesPedigreeFunction    -0.033523  0.137327      -0.002378       0.142977
      Age                          0.544341  0.266909       0.324915       0.054514
      Outcome                      0.221898  0.492782       0.165723       0.189065
```

```
                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.055697  0.021546                 -0.033523
Glucose                   0.357573  0.231400                  0.137327
BloodPressure            -0.028721  0.281132                 -0.002378
SkinThickness             0.238188  0.546951                  0.142977
Insulin                   1.000000  0.189022                  0.178029
BMI                       0.189022  1.000000                  0.153506
DiabetesPedigreeFunction  0.178029  0.153506                  1.000000
Age                      -0.015413  0.025744                  0.033561
Outcome                   0.148457  0.312249                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266909  0.492782
BloodPressure             0.324915  0.165723
SkinThickness             0.054514  0.189065
Insulin                  -0.015413  0.148457
BMI                       0.025744  0.312249
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[28]:
```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')
```

[28]:
```
<AxesSubplot:>
```



It appears from correlation matrix and heatmap that there exists significant correlation between some pairs such as - * Age-Pregnancies * BMI-SkinThickness

2.3 Week 3:

2.3.1 Data Modeling:

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

Answer: Since this is a classification problem, we will be building popular classification models for our training data and then compare performance of each model on test data to accurately predict target variable (Outcome):

```
1) Logistic Regression
2) Decision Tree
3) RandomForest Classifier
4) K-Nearest Neighbour (KNN)
```

[29]:
```
#spliting ind and dep var
X=df.drop(columns=['Outcome'])
y=df.Outcome
```

[30]:
```
! pip install imblearn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: imblearn in ./.local/lib/python3.7/site-packages
(0.0)
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.7/site-packages (from imblearn) (0.8.0)
Requirement already satisfied: scikit-learn>=0.24 in
/usr/local/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/site-
packages (from imbalanced-learn->imblearn) (0.14.1)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/site-
packages (from imbalanced-learn->imblearn) (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/site-
packages (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/site-packages (from scikit-learn>=0.24->imbalanced-
learn->imblearn) (2.2.0)
WARNING: You are using pip version 22.0.3; however, version 23.1.2 is
available.
You should consider upgrading via the '/usr/local/bin/python3 -m pip install
--upgrade pip' command.
```

[31]:
```
from imblearn.over_sampling import SMOTE
smk = SMOTE()
x_train_smote,y_train_smote=smk.fit_resample(X,y)
```

[32]:
```
from collections import Counter
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_train_smote)))
```

```
Original dataset shape Counter({0: 500, 1: 268})
Resampled dataset shape Counter({1: 500, 0: 500})
```

[33]:
```
print(x_train_smote.shape)
print(y_train_smote.shape)
```

```
(1000, 8)
(1000,)
```

[34]:
```
# Split the data set into training and testing
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =␣
↪train_test_split(x_train_smote,y_train_smote, test_size=0.2, random_state=3)
```

[35]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train_std = scaler.fit_transform(x_train)
x_test_std=scaler.transform(x_test)
```

[36]:
```
x_train_std.shape
```

[36]:
```
(800, 8)
```

[37]:
```
x_test_std.shape
```

[37]:
```
(200, 8)
```

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

[38]:
```
# Building Model For Logistic Regression
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
```

[39]:
```
# Traning Logistic Regression Model
# Logistic regression works well on scalled
lr_model.fit(x_train_std, y_train)
y_predict = lr_model.predict(x_test_std)
```

[40]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score, classification_report,roc_curve
```

[41]:
```
# Getting all accuracy socres for Logistic Regression
accuracy = accuracy_score(y_test,y_predict)
recall = recall_score(y_test,y_predict)
f1 = f1_score(y_test,y_predict)

print("Accuracy: ", accuracy)
print("Recall: ", recall)
print("F1: ", f1)
```

```
Accuracy:  0.755
Recall:  0.7289719626168224
F1:  0.7609756097560975
```

[42]:
```
# Building Model For Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
# scalling doest affect decision tree
dt_model.fit(x_train_std, y_train)
y_pred = dt_model.predict(x_test_std)
```

[43]:
```
# getting all types of accuracy for decision tree
dt_accuracy = accuracy_score(y_test,y_pred)
dt_recall = recall_score(y_test,y_pred)
dt_f1 = f1_score(y_test,y_pred)

print("Accuracy: ", dt_accuracy)
print("Recall: ", dt_recall)
print("F1: ", dt_f1)
```

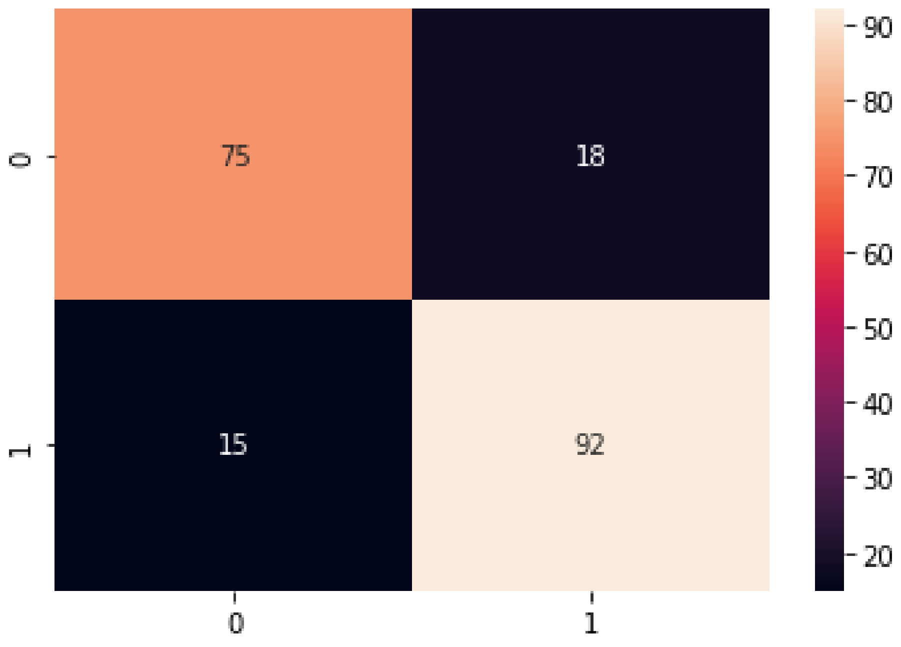
```
Accuracy:  0.755
Recall:  0.8130841121495327
F1:  0.7802690582959642
```

[44]:
```
# Building model for Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100,max_depth=5,random_state=42)
rf_model.fit(x_train_std, y_train)
y_pred1 = rf_model.predict(x_test_std)
```

[45]:
```
# getting all types of evaulation scores
rf_accuracy = accuracy_score(y_test,y_pred1)
rf_recall = recall_score(y_test,y_pred1)
rf_f1 = f1_score(y_test,y_pred1)

print("Accuracy: ", rf_accuracy)
print("Recall: ", rf_recall)
print("F1: ", rf_f1)
```

```
Accuracy:  0.835
Recall:  0.8598130841121495
F1:  0.847926267281106
```

[46]:
```
# plotting confusion metrix for random forest
rf_cm = confusion_matrix(y_test, y_pred1)
sns.heatmap(rf_cm, annot=True)
plt.show()
```



[47]:
```python
# Printing overall report for random forest
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.83      0.81      0.82        93
           1       0.84      0.86      0.85       107

    accuracy                           0.83       200
   macro avg       0.83      0.83      0.83       200
weighted avg       0.83      0.83      0.83       200
```

[48]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=20).fit(x_train_std,y_train)
y_pred2 = knn.predict(x_test_std)
```

[49]:
```python
# getting all types of evaulation scores
knn_accuracy = accuracy_score(y_test,y_pred2)
knn_recall = recall_score(y_test,y_pred2)
knn_f1 = f1_score(y_test,y_pred2)

print("Accuracy: ", knn_accuracy)
print("Recall: ", knn_recall)
print("F1: ", knn_f1)
```

```
Accuracy:  0.775
Recall:  0.8130841121495327
F1:  0.7945205479452055
```

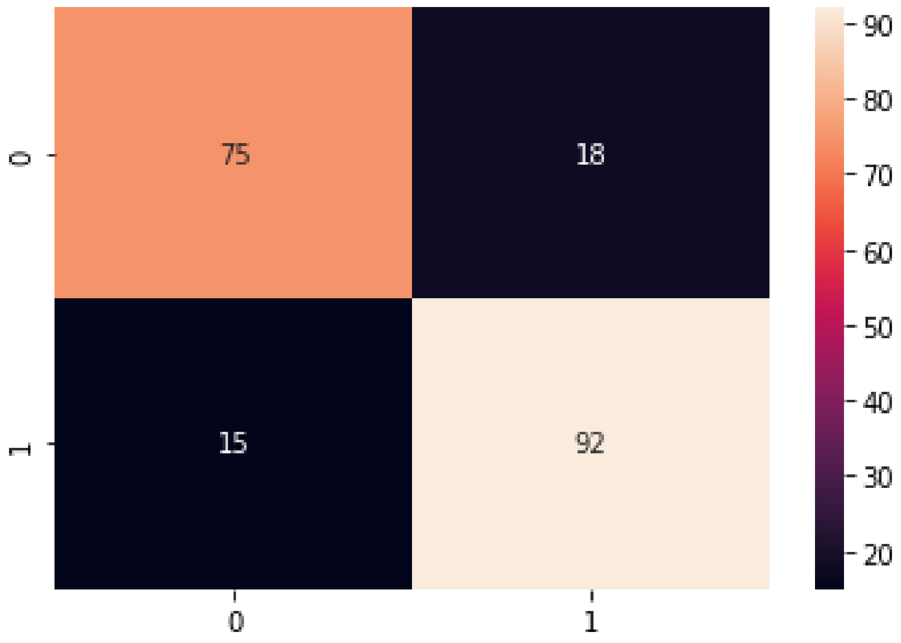
Among all models, RandomForest has given best accuracy,recall and f1_score. Therefore we will build final model using RandomForest.

2.4 Week 4:

2.4.1 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:

[50]:
```
# plotting confusion metrix for random forest
rf_cm = confusion_matrix(y_test, y_pred1)
sns.heatmap(rf_cm, annot=True)
plt.show()
```



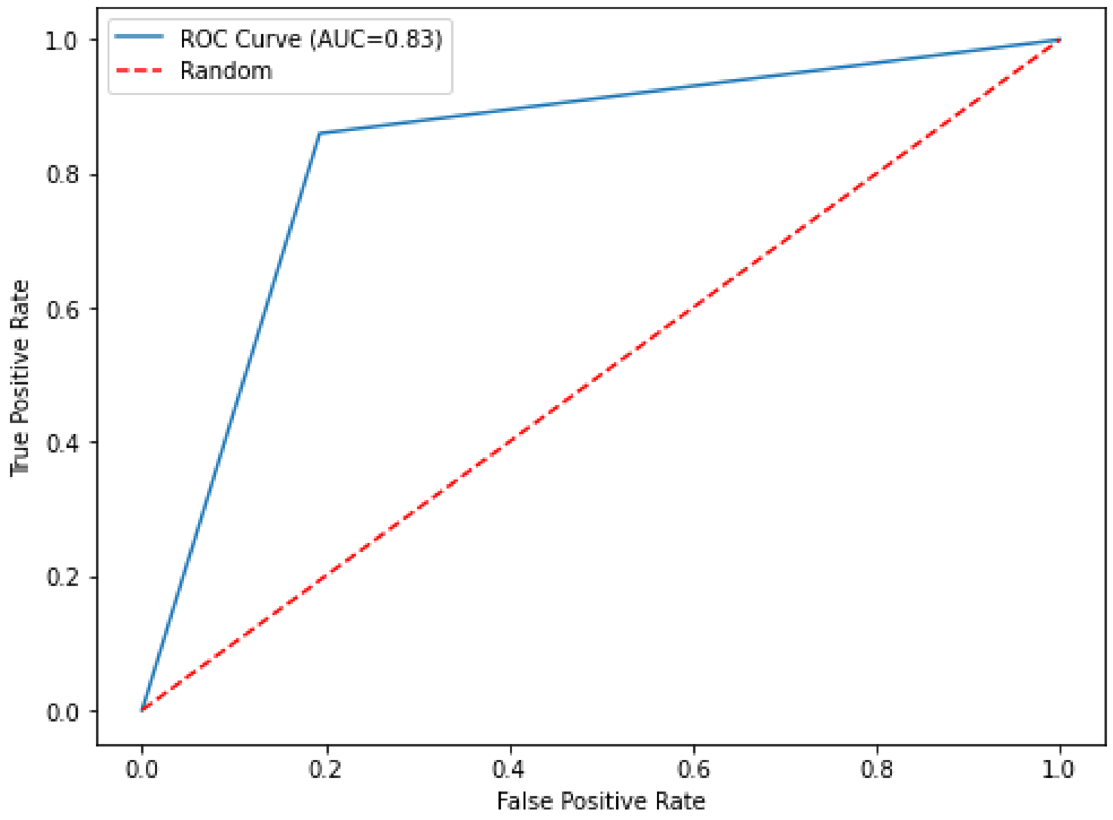
[51]:
```
# Printing overall report for random forest
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.83      0.81      0.82        93
           1       0.84      0.86      0.85       107

    accuracy                           0.83       200
   macro avg       0.83      0.83      0.83       200
weighted avg       0.83      0.83      0.83       200
```

[52]:
```
fpr,tpr,thersholds=roc_curve(y_test,y_pred1)
```

[53]:
```
auc=roc_auc_score(y_test,y_pred1)
```

[54]:
```
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr,label='ROC Curve (AUC={:.2f})'.format(auc))
plt.plot([0,1],[0,1],linestyle='--',color='r',label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In this scenario, we encountered a data imbalance issue, where the distribution of classes in the dataset was not uniform. To address this problem, we applied the Synthetic Minority Over-sampling

Technique (SMOTE) to balance the data. The objective was to improve the performance of a classification model by ensuring equal representation of the minority class.

After applying SMOTE, we evaluated the model's performance using several metrics. The accuracy of the model was measured to be 83%, indicating that 83% of the predictions made by the model were correct. This metric provides an overall assessment of the model's performance.

Additionally, we assessed the model's ability to correctly identify positive instances of the minority class. The recall, also known as sensitivity or true positive rate, was measured to be 85%. This means that out of all the actual positive instances, the model successfully identified 85% of them. A higher recall score indicates a better ability to detect positive cases.

Furthermore, we examined the model's balance between precision and recall using the F1 score. The F1 score, which is the harmonic mean of precision and recall, was calculated to be 84%. This score provides a balanced assessment of the model's performance in terms of both false positives and false negatives. A higher F1 score indicates a better trade-off between precision and recall.

Finally, we assessed the model's ability to rank instances and discriminate between positive and negative classes using the Receiver Operating Characteristic (ROC) curve. The area under the ROC curve (ROC AUC score) was calculated to be 0.83. This metric measures the model's ability to distinguish between positive and negative instances. A higher ROC AUC score indicates better discrimination ability.

In conclusion, the application of SMOTE to address the data imbalance issue improved the performance of the classification model. The model achieved an accuracy of 83%, recall of 85%, F1 score of 84%, and an ROC AUC score of 0.83. These results indicate that the model has demonstrated promising performance in handling the imbalanced data and is effective in identifying positive instances of the minority class. However, it is important to note that further evaluation and validation on additional datasets may be necessary to ascertain the generalizability of these results.

2.4.2 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
```

2.5 PLEASE REFER TABLEAU PUBLIC LINK FOR DASHBOARD AND VISUALIZATION.

[ ]: