Data Science Capstone – Healthcare

Report created by: **Sravanthi Baman**

DESCRIPTION

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). A predictor variable includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Sr.no	VARIABLES	DESCRIPTION
1.	PREGNANCIES	Number of times pregnant
2.	GLUCOSE	Plasma glucose concentration in an oral glucose tolerance test
3.	BLOODPRESSURE	Diastolic blood pressure (mm Hg)
4.	SKINTHICKNESS	Triceps skinfold thickness (mm)
5.	INSULIN	Two-hour serum insulin
6.	BMI	Body Mass Index
7.	DIABETESPEDIGREEFUNCTION	Diabetes pedigree function
8.	AGE	Age in years
9.	OUTCOME	Class variable (either 0 or 1). 268 of 768 values are 1, and the others

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
   - Blood Pressure
   - Skin Thickness

- Insulin
- BMI

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.
4. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
5. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
6. Perform correlation analysis. Visually explore it using a heat map.

Methodology for Task for Week -1

Note: All Python codes were written inside capstone. ipynb file. Attached this file set along with the report folder.

1. **Import all necessary libraries for this project. The Screenshot are attached below.**

```
# import the library
```

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

2. Load the dataset and read the csv file with pandas

```
n [2]: #load the dataset
       s1=pd.read_csv("health care diabetes.csv")
```

3. After reading csv file, we have to perform some following steps on the data set.

- **See the shape and columns in dataset**

```
In [3]: #check the shape of tthe dataset
        s1.shape
Out[3]: (768, 9)

In [4]: s1.columns
Out[4]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

- **Read first rows from the dataset.**

```
In [7]: #check the first 7 rows from the dataset
        s1.head(6)
```

Out[7]:

	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
5	5	116	74	0	0	25.6	0.201	30	0

2. Read the last five rows from the dataset

```
In [6]: #check for last 5 rows in dataset
        s1.tail(5)
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

3. Read the Info from Dataset

Info: it prints information about the Data Frame, which contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

Note: the info () method actually prints the info.

```
In [7]: s1.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
```

Describe: It returns a description of the data in the Data Frame. If the Data Frame contains numerical data, then the description shows the information for each column: count The number of not-empty values. mean: - The average (mean) value, std - The standard deviation.

```
In [8]: #describe shows five point summary
        s1.describe()
```

Out[8]:

	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

- **Check if any null value is present in dataset**
  Isnull & sum: It returns how many null values are present inside this dataset & its total count.

```
In [9]: s1.isna()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False
767	False	False	False	False	False	False	False	False	False

768 rows × 9 columns

```
In [10]: #check null
         s1.isnull().sum()
```

Out[10]:

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

- **Check Standard deviation and skew of each variable in dataset**

```
In [13]: print("Standard deviation of each variable are present")
         s1.apply(np.std)
```

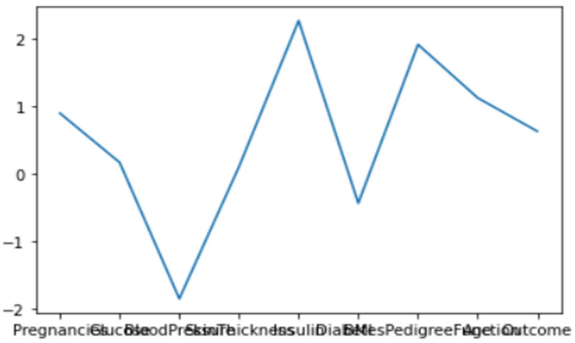
```
Standard deviation of each variable are present
```

Out[13]:

```
Pregnancies                 3.367384
Glucose                    31.951796
BloodPressure              19.343202
SkinThickness              15.941829
Insulin                   115.168949
BMI                         7.879026
DiabetesPedigreeFunction    0.331113
Age                        11.752573
Outcome                     0.476641
dtype: float64
```

Data Science Capstone -Healthcare

```
In [14]: s1.skew(axis = 0, skipna = True).plot()
Out[14]: <AxesSubplot:>
```



Treating Missing Values and Analysing Distribution of Data-set

Perform Descriptive Analysis

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

Case for Variable Glucose

Before treating missing value.

```
In [17]: # how many 0 value are present inside Glucose column.
         count =(s1['Glucose']==0).sum()
         count
Out[17]: 5
```

Handling the missing value.

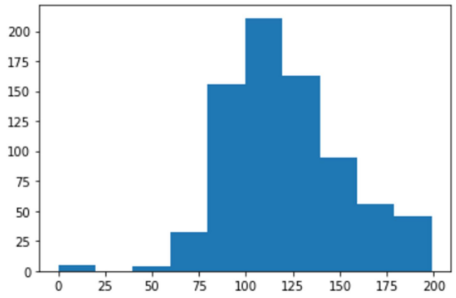
```
In [19]: s1['Glucose'].replace(to_replace = 0, value = s1['Glucose'].mean(), inplace=True)
```

```
In [20]: count =(s1['Glucose']==0).sum()
         count
Out[20]: 0
```

Before

```
In [18]: #Glucose before converting 0 to mean
         plt.hist(s1['Glucose'])
         print("Mean of Glucose level is :-", s1['Glucose'].mean())
         print("Datatype of Glucose Variable is:",s1['Glucose'].dtypes)
```

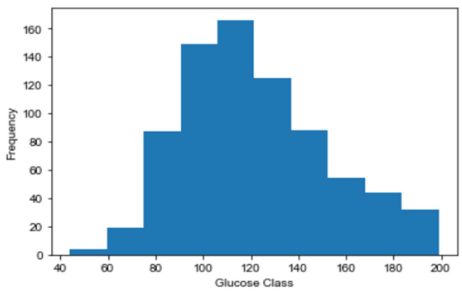
```
Mean of Glucose level is :- 120.89453125
Datatype of Glucose Variable is: int64
```



After

```
# #After converting 0 values to mean for the Glucose & plot its histogram
plt.figure(figsize=(6,4))
plt.xlabel('Glucose Class')
s1['Glucose'].plot.hist()
sns.set_style(style='darkgrid')
print("Mean of Glucose level is :-", s1['Glucose'].mean())
print("Datatype of Glucose Variable is:",s1['Glucose'].dtypes)
```

```
Mean of Glucose level is :- 121.68160502115886
Datatype of Glucose Variable is: float64
```



If, we compare the Histogram before & after the missing value treatment, it shows like this:

Case for Variable Blood Pressure

Before treating missing value.

```
In [22]: # how many 0 value are present inside BloodPressure column.
         BloodPressure_before_count =(s1['BloodPressure']==0).sum()
         BloodPressure_before_count

Out[22]: 35
```

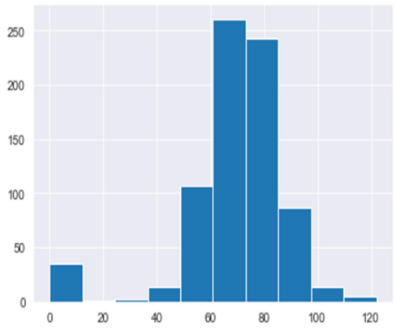
Handling the missing value.

```
In [25]: s1['BloodPressure'].replace(to_replace = 0, value = s1['BloodPressure'].mean(), inplace=True)

In [26]: BloodPressure_before_count =(s1['BloodPressure']==0).sum()
         BloodPressure_before_count

Out[26]: 0
```

If, we compare the Histogram before & after the missing value treatment, it shows like this:

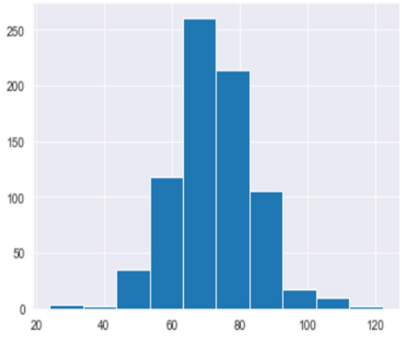
Before

```
In [24]: #BloodPressure before converting 0 to mean
         plt.hist(s1['BloodPressure'])
         print("Mean of BloodPressure level is :-", s1['BloodPressure'].mean())
         print("Datatype of BloodPressure Variable is:",s1['BloodPressure'].dtypes)

Mean of BloodPressure level is :- 69.10546875
Datatype of BloodPressure Variable is: int64
```



After

```
In [27]: #BloodPressure after converting 0 to mean
         plt.hist(s1['BloodPressure'])
         print("Mean of BloodPressure level is :-", s1['BloodPressure'].mean())
         print("Datatype of BloodPressure Variable is:",s1['BloodPressure'].dtypes)

Mean of BloodPressure level is :- 72.25480651855469
Datatype of BloodPressure Variable is: float64
```



Case for Variable BMI

Before treating missing value.

20 40 60 80 100 120

```
In [28]: # how many 0 value are present inside BMI column.
         count =(s1['BMI']==0).sum()
         count

Out[28]: 11
```

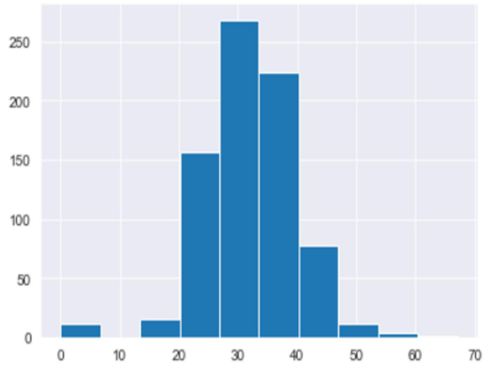
Handling the missing value.

```
In [29]: s1['BMI'].replace(to_replace = 0, value = s1['BMI'].mean(), inplace=True)

In [30]: count =(s1['BMI']==0).sum()
         count

Out[30]: 0
```

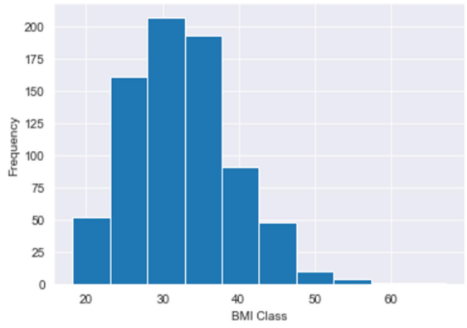
If, we compare the Histogram before & after the missing value treatment, it shows like this:

Before **After**

```
: #BMI Before converting 0 to mean
  plt.hist(s1['BMI'])
  print("Mean of BMI level is :-", s1['BMI'].mean())
  print("Datatype of BMI Variable is:",s1['BMI'].dtypes)

Mean of BMI level is :- 31.992578124999977
Datatype of BMI Variable is: float64
```




```
In [33]: ##After converting 0 values to mean for the BMI & plot its histogram.
         plt.figure(figsize=(6,4))
         plt.xlabel('BMI Class')
         s1['BMI'].plot.hist()
         sns.set_style(style='darkgrid')
         print("Mean of BMI is :-", s1['BMI'].mean())
         print("Datatype of BMI Variable is:",s1['BMI'].dtypes)

Mean of BMI is :- 32.45080515543617
Datatype of BMI Variable is: float64
```



Case for Variable Insulin

Before treating missing value.

```
In [34]: #How many 0 columns are present we have to check.
         count =(s1['Insulin']==0).sum()
         count

Out[34]: 374
```

Handling the missing value.

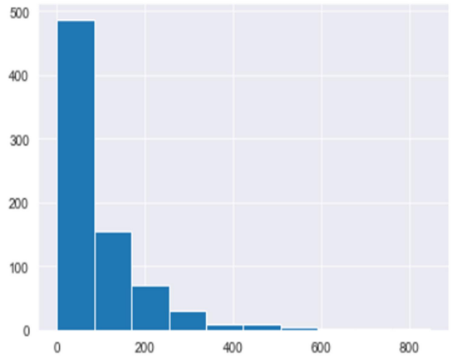
```
n [36]: s1["Insulin"].replace(to_replace=0, value=s1["Insulin"].mean(),inplace=True)

n [37]: Count=(s1["Insulin"]==0).sum()
        Count

ut[37]: 0
```

If, we compare the Histogram before & after the missing value treatment, it displays like this:

Before

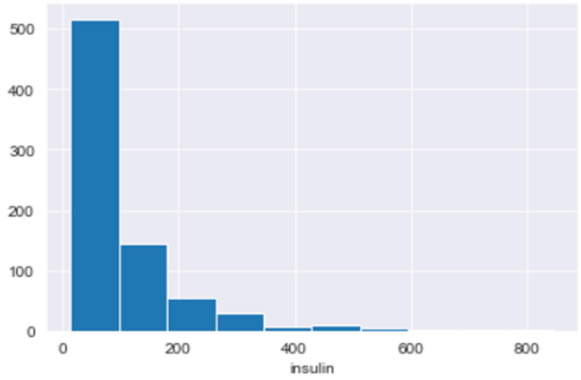
```
5]:
#Insulin before converting 0 to mean
plt.hist(s1['Insulin'])
print("Mean of Insulin level is :-", s1['Insulin'].mean())
print("Datatype of Insulin Variable is:",s1['Insulin'].dtypes)
```

```
Mean of Insulin level is :- 79.79947916666667
Datatype of Insulin Variable is: int64
```



After

```
In [38]: #after converting 0 to mean
plt.figure(figsize=(6,4))
plt.xlabel("insulin")
plt.hist(s1["Insulin"])
plt.show()
```



Case for Variable Skin thickness

Before treating missing value.

```
In [39]: #How many 0 columns are present we have to check.
SkinThickness_before_count =(s1['SkinThickness']==0).sum()
SkinThickness_before_count

Out[39]: 227
```

Handling the missing value.


```
In [41]: s1["SkinThickness"].replace(to_replace=0, value=s1["SkinThickness"].mean(),inplace=True)

In [42]: Count=(s1["SkinThickness"]==0).sum()
Count

Out[42]: 0
```

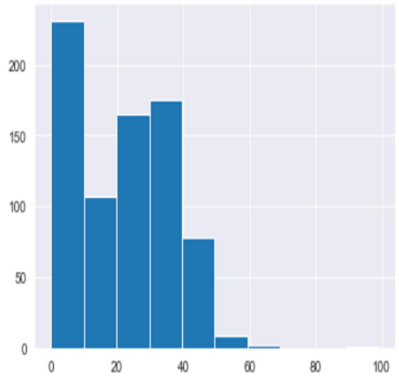
If, we compare the Histogram before & after the missing value treatment, it shows like this:

Data Science Capstone -Healthcare

Before

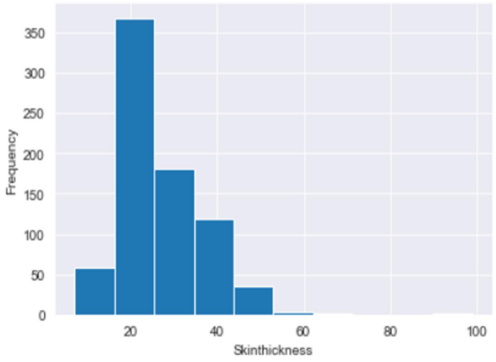
```
In [40]: #SkinThickness before converting 0 to mean
plt.hist(s1['SkinThickness'])
print("Mean of SkinThickness level is :-", s1['SkinThickness'].mean())
print("Datatype of SkinThickness Variable is:",s1['SkinThickness'].dtypes)
```

```
Mean of SkinThickness level is :- 20.536458333333332
Datatype of SkinThickness Variable is: int64
```



After

```
In [43]: plt.figure(figsize=(6,4))
plt.xlabel("Skinthickness")
s1["SkinThickness"].plot.hist()
plt.show()
print("Mean of Insulin level is :-", s1['Insulin'].mean())
print("Datatype of Insulin Variable is:",s1['Insulin'].dtypes)
```




```
Mean of Insulin level is :- 118.66016303168442
Datatype of Insulin Variable is: float64
```

Create a count (frequency) plot describing the data types & the count of variables

- **Value count for insulin**

```
In [48]: #value counts for Insulin
s1['Insulin'].value_counts().head(7)

Out[48]: 79.799479     374
105.000000     11
130.000000      9
140.000000      9
120.000000      8
94.000000       7
180.000000      7
Name: Insulin, dtype: int64
```

Value count for glucose

```
[45]: #value count  for  Glucose
s1['Glucose'].value_counts().head(7)

t[45]: 99.0     17
100.0    17
111.0    14
129.0    14
125.0    14
106.0    14
112.0    13
Name: Glucose, dtype: int64
```

Value count for BMI

```
In [47]: #value counts for bmi
s1['BMI'].value_counts().head(7)

Out[47]: 32.000000    13
31.600000    12
31.200000    12
31.992578    11
32.400000    10
33.300000    10
30.100000     9
Name: BMI, dtype: int64
```

Count table for value count of Skin thickness and Blood pressure

```
In [46]: #value counts for SkinThickness
         s1['SkinThickness'].value_counts().head(7)
```

```
Out[46]: 20.536458    227
         32.000000     31
         30.000000     27
         27.000000     23
         23.000000     22
         33.000000     20
         28.000000     20
         Name: SkinThickness, dtype: int64
```

```
In [9]: #value counts for bmi
        s1['BloodPressure'].value_counts().head(7)
```

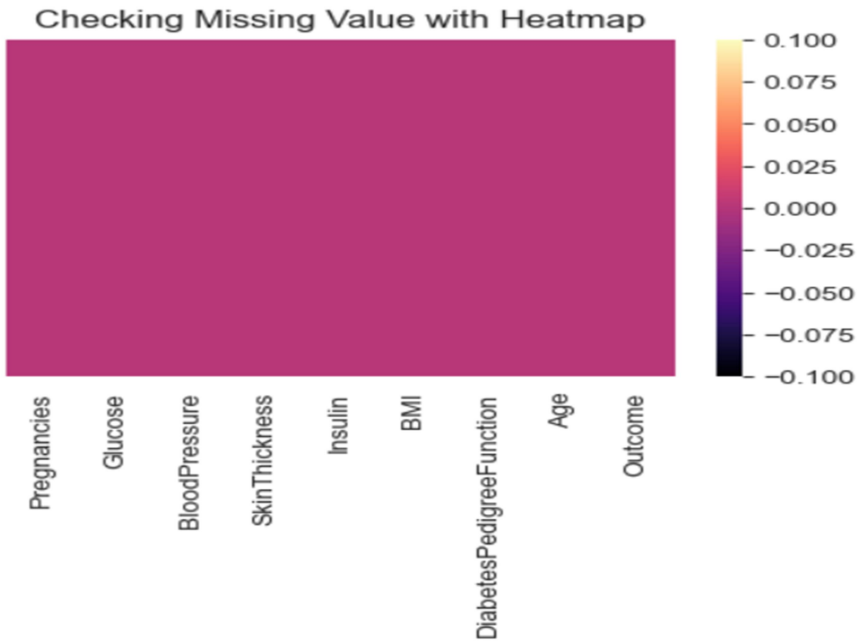
```
Out[9]: 70    57
        74    52
        78    45
        68    45
        72    44
        64    43
        80    40
        Name: BloodPressure, dtype: int64
```

Checking Missing Values with Heat-Map

Plotting the count of outcomes by their values

```
In [49]: #checking Missing value with heatmap
         plt.figure(figsize=(5,3),dpi=100)
         plt.title('Checking Missing Value with Heatmap')
         sns.heatmap(s1.isnull(),cmap='magma',yticklabels=False)
```

```
Out[49]: <AxesSubplot:title={'center':'Checking Missing Value with Heatmap'}>
```

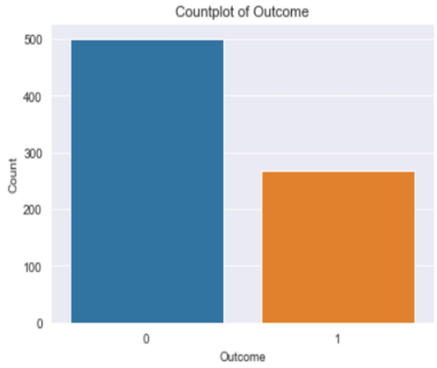


Describe the finding

In [50]:
```
#create the count plot
sns.set_style('darkgrid')
sns.countplot(s1['Outcome'])
plt.title("Countplot of Outcome")
plt.xlabel('Outcome')
plt.ylabel("Count")
print("Count of class is:\n",s1['Outcome'].value_counts())
```

```
d:\Users\admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

```
Count of class is:
 0    500
1    268
Name: Outcome, dtype: int64
```



In [51]:
```
print("Count of class is:\n",s1['Outcome'].value_counts())
```

```
Count of class is:
 0    500
1    268
```

Since classes in Outcome are a little skewed, we will generate new samples using SMOTE (Synthetic Minority Oversampling Technique) for class '1', which is under represented in our data. We will use SMOTE out of many other techniques available because it generates new samples by interpolation and doesn't duplicate the data.

In [50]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.datasets import make_classification
```

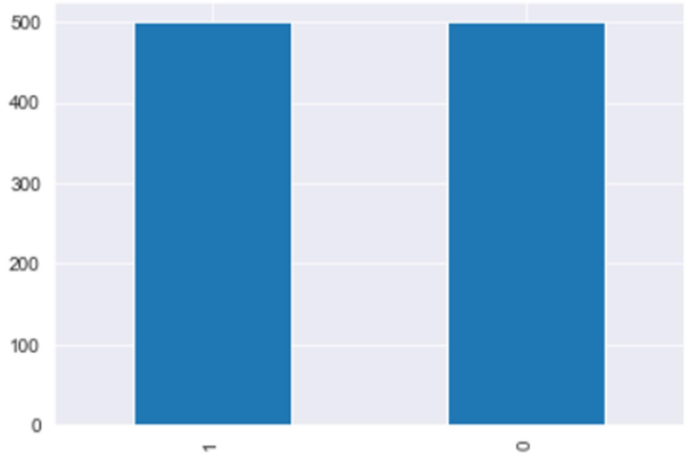
In [51]:
```
s1_X = s1.drop('Outcome', axis=1)
s1_y = s1['Outcome']
print(s1_X.shape, s1_y.shape)
```

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

In [52]:
```
s1_X_resampled, s1_y_resampled = SMOTE(random_state=108).fit_resample(s1_X, s1_y)
print(s1_X_resampled.shape, s1_y_resampled.shape)
#print('Original dataset shape %s' % Counter(y))
```

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

```
In [53]: s1_y_resampled.value_counts().plot(kind='bar')
         s1_y_resampled.value_counts()

Out[53]: 1    500
         0    500
         Name: Outcome, dtype: int64
```



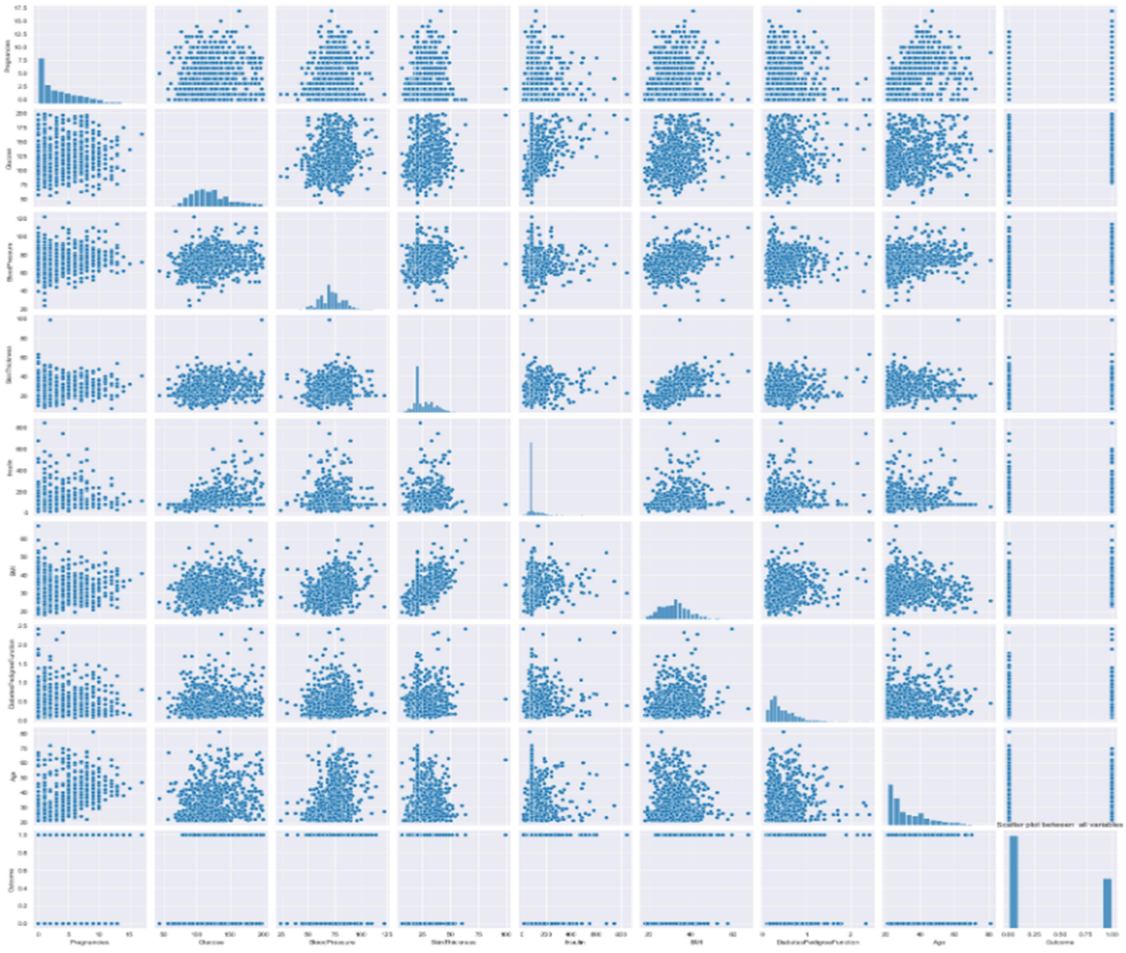
Scatter plot

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

Scatter Plot

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

```
In [56]: sns.pairplot(s1)
         plt.title("Scatter plot between  all variables")

Out[56]: Text(0.5, 1.0, 'Scatter plot between  all variables')
```

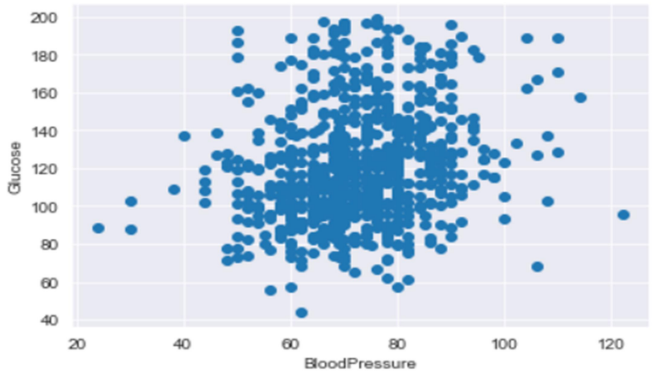


Describe the observation

We can see from scatter plot that there is no strong multicolinearity among features, but between skin thickness and BMI, Pregnancies and age it looks like there is small chance of positive correlation. Also create scatter plot to identify the co relation between variables observation.

Scatter plot with Glucose and Blood-Pressure

```
In [55]: plt.scatter(s1['BloodPressure'], s1['Glucose'])
         plt.xlabel("BloodPressure") #x label
         plt.ylabel("Glucose") #y label
         plt.show()
```

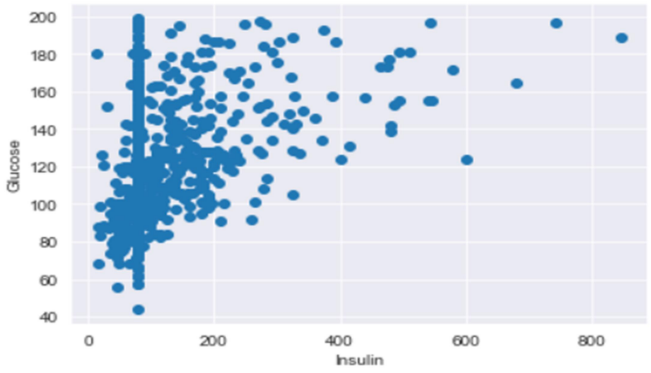


So, from this observation we can see that the positive (+) co-relation is present between Blood pressure & Glucose.

Scatter plot with insulin and glucose

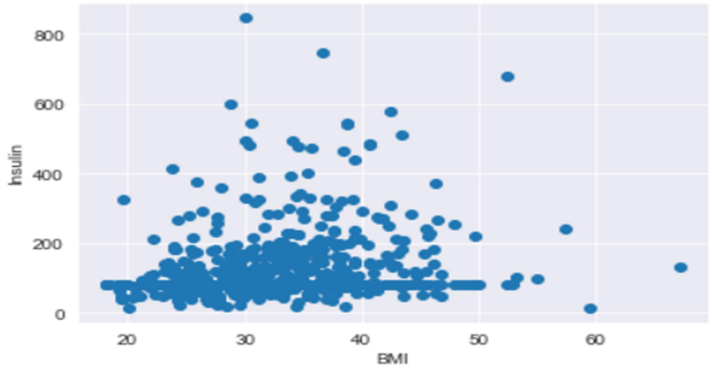
```
In [ ]: #scatter plot betwwen glucose and insulin
```

```
In [53]: plt.scatter(s1["Insulin"], s1["Glucose"] )
         plt.xlabel("Insulin") #x label
         plt.ylabel("Glucose") #y label
         plt.show()
```



Scatter plot with BMI and Insulin

```
In [54]: #creating a scatter plot
         plt.scatter(s1['BMI'], s1['Insulin'])
         plt.xlabel("BMI") #x label
         plt.ylabel("Insulin") #y label
         plt.show()
```



So, from this observation we can see that the positive (+) correlation is present between BMI & Insulin.

```
#**Correlation Analysis**
```

```
In [56]: s1.corr()
```

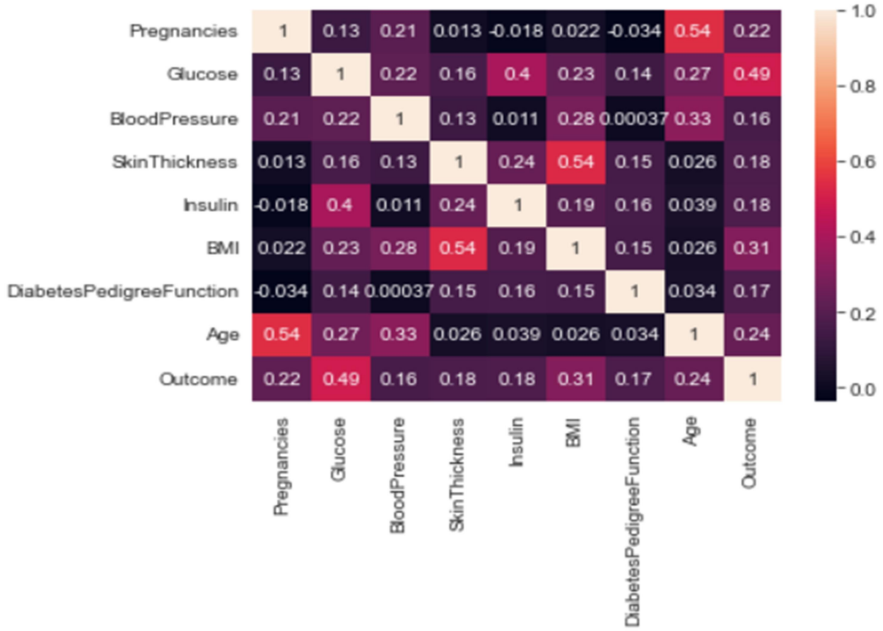
Out[56]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.127964	0.208984	0.013376	-0.018082	0.021546	-0.033523	0.544341	0.221898
Glucose	0.127964	1.000000	0.219666	0.160766	0.396597	0.231478	0.137106	0.266600	0.492908
BloodPressure	0.208984	0.219666	1.000000	0.134155	0.010926	0.281231	0.000371	0.326740	0.162986
SkinThickness	0.013376	0.160766	0.134155	1.000000	0.240361	0.535703	0.154961	0.026423	0.175026
Insulin	-0.018082	0.396597	0.010926	0.240361	1.000000	0.189856	0.157806	0.038652	0.179185
BMI	0.021546	0.231478	0.281231	0.535703	0.189856	1.000000	0.153508	0.025748	0.312254
DiabetesPedigreeFunction	-0.033523	0.137106	0.000371	0.154961	0.157806	0.153508	1.000000	0.033561	0.173844
Age	0.544341	0.266600	0.326740	0.026423	0.038652	0.025748	0.033561	1.000000	0.238356
Outcome	0.221898	0.492908	0.162986	0.175026	0.179185	0.312254	0.173844	0.238356	1.000000

Perform Co-relation Analysis with Heat Map

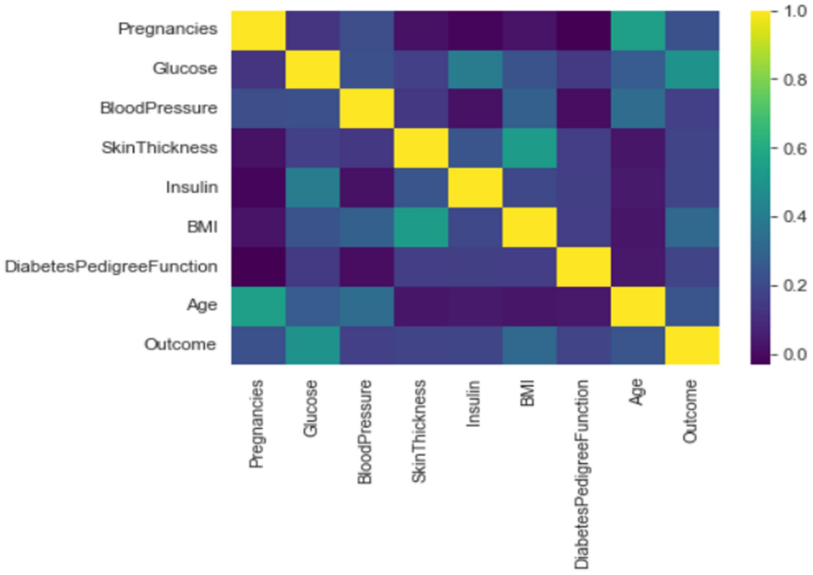
```
In [57]: sns.heatmap(s1.corr(),annot=True)
```

Out[57]: <AxesSubplot:>



```
In [58]:
plt.figure(dpi=80)
sns.heatmap(s1.corr(),cmap='viridis')
```

```
Out[58]: <AxesSubplot:>
```



Project Task: Week 2

Data Modelling

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.
3. Compare various models with the results from KNN algorithm.
4. Create a classification report by analysing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of this parameter you have used.

Step -1:

```
Data Preprocessing
```

```
In [59]:
#loc is label-based, which means that you have to specify rows and
#columns based on their row and column labels.
X=s1.iloc[:,:-1].values
y=s1.iloc[:,-1].values
```

Step-2:

(Splitting the data-set into training and testing part) and (To identify how many rows and columns are present in X_train, X_test, y train and y_test.

now we have to import train test split from sklearn. model_ selection. now split our dataset into 80% training and 20% testing & random state is treated as hyper parameter which control the shuffling process. With random _state=0, we get the same train and test sets across different executions.

Now Starting Different Model Building Process (KNN)

In [60]:
```
from sklearn.model_selection import train_test_split
```

In [61]:
```
#now we have import train test split from sklearn.model_selection
#now split our dataset into 80% training and 20% testing &
#random state is treated as hyperparameter which control the
#shuffing process.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614,)
(154,)
```

KNN

70]:
```
#Now time to import KNeighborsClassifier and Predict
from sklearn.neighbors import KNeighborsClassifier
knn_model_value = KNeighborsClassifier(n_neighbors=45)
knn_model_value.fit(X_train,y_train)
knn_pred_norm=knn_model_value.predict(X_test)
```

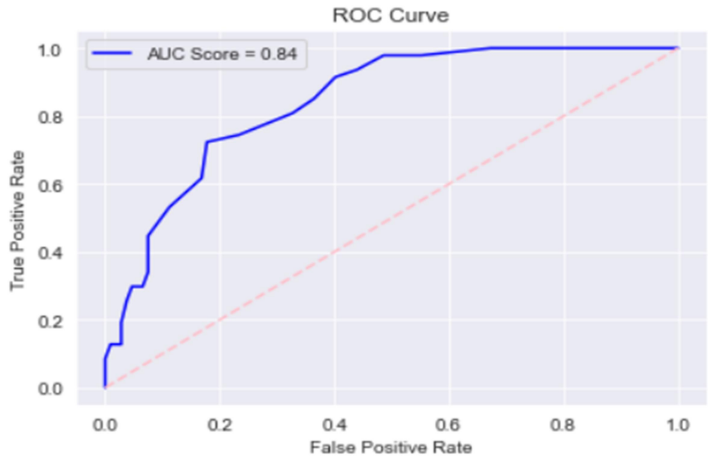
K-NN algorithm stores all the available data and classifies a new data point based on the similarity.

In [66]:
```
print("Model Validation ==>\n")
print("Accuracy Score of KNN Model with Normalization::")
print(metrics.accuracy_score(y_test,knn_pred_norm))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,knn_pred_norm),'\n')
print("\n","ROC Curve")
knn_prob_norm=knn_model_value.predict_proba(X_test)
knn_prob_norm1=knn_prob_norm[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,knn_prob_norm1)
roc_auc_knn=metrics.auc(fpr,tpr)
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='pink')
plt.legend()
```

```
Model Validation ==>

Accuracy Score of KNN Model with Normalization::
0.8051948051948052

 Classification Report::
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       107
           1       0.72      0.60      0.65        47

    accuracy                           0.81       154
   macro avg       0.78      0.75      0.76       154
weighted avg       0.80      0.81      0.80       154
```

```
yword argument and the fmt string "r--" (-> color='r'). In
  plt.plot(fpr,fpr,'r--',color='pink')
```

Out[66]: <matplotlib.legend.Legend at 0x220008778e0>



SO, from KNN standard scaling we got the Model Accuracy is 0.81 and from the classification report we got its precision, recall, f1-scoare and support value. With Addition its ROC Curve AUC score is 84%. That is good.

Now Starting Different Model Building Process (LOGSTIC REGRESSION)

Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring.

```
: #first we have to import metrics
  from sklearn import metrics
```

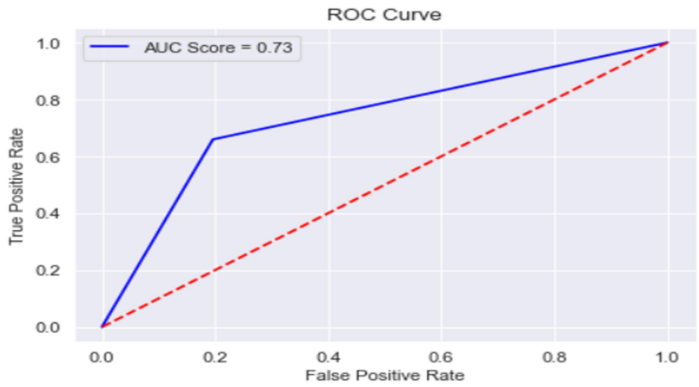
```
: #import Logistic Regression and tried to building a model.
  from sklearn.linear_model import LogisticRegression
  from sklearn.metrics import confusion_matrix
  lr_model_classifier = LogisticRegression(C=0.01)
  lr_model_classifier.fit(X_train,y_train)
  lr_pred=lr_model_classifier.predict(X_test)
```

```
]: print("Model Validation ==>\n")
   print("Accuracy Score of Logistic Regression Model::")
   print(metrics.accuracy_score(y_test,dtc_pred))
   print("\n","Classification Report::")
   print(metrics.classification_report(y_test,dtc_pred),'\n')
   print("\n","Confusion Matrix::")
   cm = confusion_matrix(y_test,dtc_pred,labels=None)
   print(cm)
   print("\n","ROC Curve")
   lr_prob=dtc_model_classifier.predict_proba(X_test)
   lr_prob1=lr_prob[:,1]
   fpr,tpr,thresh=metrics.roc_curve(y_test,lr_prob1)
   roc_auc_lr=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_lr)
   plt.plot(fpr,fpr,'r--',color='red')
   plt.legend()
```

From the above code we got output for Model Accuracy, classification report, confusion Matrix & ROC curve

```
Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.7597402597402597

 Classification Report::
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       107
           1       0.60      0.66      0.63        47

    accuracy                           0.76       154
   macro avg       0.72      0.73      0.72       154
weighted avg       0.77      0.76      0.76       154


 Confusion Matrix::
[[86 21]
 [16 31]]
```

```
ROC Curve
C:\Users\admin\AppData\Local\Temp/ipykernel_13276/3614937086.py:19: Use
yword argument and the fmt string "r--" (-> color='r'). The keyword arg
  plt.plot(fpr,fpr,'r--',color='red')
<matplotlib.legend.Legend at 0x22000054460>
```



Observation: Accuracy, recall, f1-score of KNN is better than Logistic Regression.

Now Starting Different Model Building Process (Random Forest)

A Random forest is a meta-estimator that fits a number of decision tree classifiers on various subsamples of the dataset and uses averaging to improve predictive accuracy and control over fitting. Model Creation:

```
#Random Forest Classifier
```

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
rfm=RandomForestClassifier(n_estimators=1000,random_state=0)
rfm.fit(X_train,y_train)
rfp=rfm.predict(X_test)
```

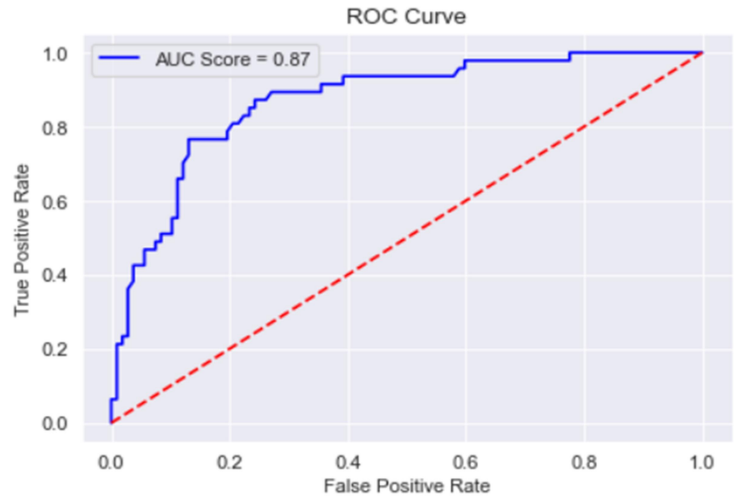
In [71]:
```
rf_model_classifier = RandomForestClassifier(n_estimators=1000,random_state=0)
rf_model_classifier.fit(X_train,y_train)
rf_pred_classifier=rf_model_classifier.predict(X_test)
```

In [72]:
```
print("Model Validation ==>\n")
print("Accuracy Score of Random Forest Classifier Model::")
print(metrics.accuracy_score(y_test,rf_pred_classifier))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,rf_pred_classifier),'\n')
print("\n","Confusion Matrix::")
cm = confusion_matrix(y_test,rf_pred_classifier,labels=None)
print(cm)
print("\n","ROC Curve")
rf_prob=rf_model_classifier.predict_proba(X_test)
rf_prob1=rf_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,rf_prob1)
roc_auc_rf=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_rf)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

From the above code we got output for Model Accuracy, classification report, confusion Matrix & ROC curve

```
Model Validation ==>

Accuracy Score of Random Forest Classifier Model::
0.8246753246753247

 Classification Report::
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       107
           1       0.71      0.72      0.72        47

    accuracy                           0.82       154
   macro avg       0.79      0.80      0.79       154
weighted avg       0.83      0.82      0.83       154


 Confusion Matrix::
[[93 14]
 [13 34]]
```

```
[72]: <matplotlib.legend.Legend at 0x22001ba99a0>
```



SO, from random Forest scaling we got the Model Accuracy is 0.82 and from the classification report we got its precision, recall, f1-scoare and support value. With Addition its ROC Curve AUC score is 87%. That is good.

Now Starting Different Model Building Process (Decision Tree Classifier)

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

```
## Decision Tress Classifier
```

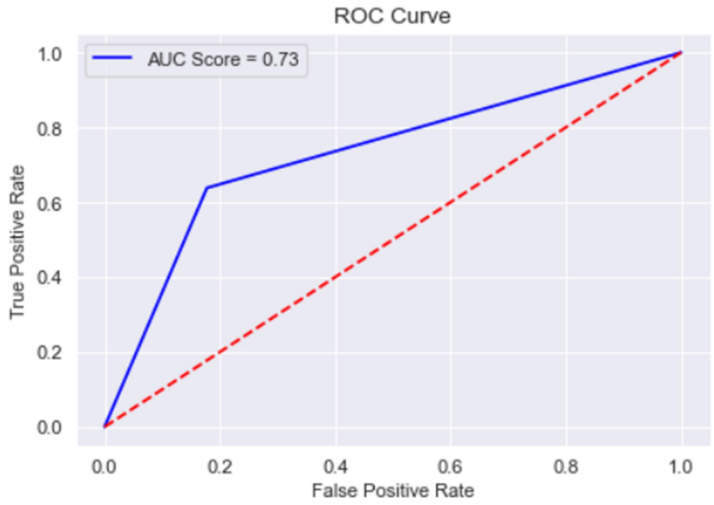
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
dtc_model_classifier = DecisionTreeClassifier()
dtc_model_classifier.fit(X_train,y_train)
dtc_pred=dtc_model_classifier.predict(X_test)
```

[123]:
```
print("Model Validation ==>\n")
print("Accuracy Score of Decision Tree Classfier Model::")
print(metrics.accuracy_score(y_test,dtc_pred))
print("\n","Classification Report::")
print(metrics.classification_report(y_test,dtc_pred),'\n')
print("\n","Confusion Matrix::")
cm = confusion_matrix(y_test,dtc_pred,labels=None)
print(cm)
print("\n","ROC Curve")
lr_prob=dtc_model_classifier.predict_proba(X_test)
lr_prob1=lr_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,lr_prob1)
roc_auc_lr=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_lr)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

From the above code we got output for Model Accuracy, classification report, confusion Matrix & ROC curve:

```
Model Validation ==>

Accuracy Score of Decesion Tree Classfier Model::
0.7662337662337663

 Classification Report::
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       107
           1       0.61      0.64      0.62        47

    accuracy                           0.77       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.77      0.77      0.77       154


 Confusion Matrix::
[[88 19]
 [17 30]]

 ROC Curve
```

23]: <matplotlib.legend.Legend at 0x245b7227100>



Observation: Accuracy, recall, f1-score of KNN is better than Decision Tree.

Now Starting Different Model Building Process (SVM)

SVM is also a Supervised Machine Learning Algorithm that uses a set of rules to make decisions, similarly to how humans make decisions. One way to think of a Machine Learning classification algorithm is that it is built to make decisions.

SVM MODEL

[124]:
```
#from sklearn import svm
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import classification_report,confusion_matrix
```

[125]:
```
# ALL Default hyperparameters
svc_model=SVC(probability=True)
svc_model.fit(X_train,y_train)
y_pred=svc_model.predict(X_test)
```

[127]:
```
#Percentage of observations used to build the SVC (Support Vectors)
svc_model.support_vectors_.shape[0]/s1.shape[0]*100
```

[127]:
```
48.95833333333333
```

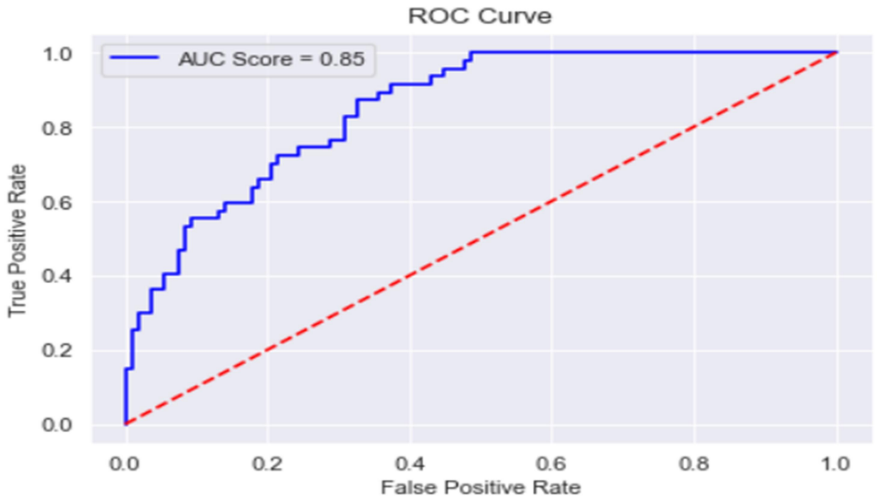
```
Model Evaluation
Now, get predictions from the model and print accuracy, and create a confusion matrix and a classification report.
```

]:
```
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')
print("\n","ROC Curve")
svm_prob=svc_model.predict_proba(X_test)
svm_proba1=svm_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,svm_proba1)
roc_auc_lr=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_lr)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

From the above code we got output for Model Accuracy, classification report, confusion Matrix & ROC curve:

```
Model Validation ==>

Accuracy-Score of SVM Model::
0.7922077922077922

 Classification Report::
              precision    recall  f1-score   support

           0       0.81      0.92      0.86       107
           1       0.73      0.51      0.60        47

    accuracy                           0.79       154
   macro avg       0.77      0.71      0.73       154
weighted avg       0.78      0.79      0.78       154


 ROC Curve
```

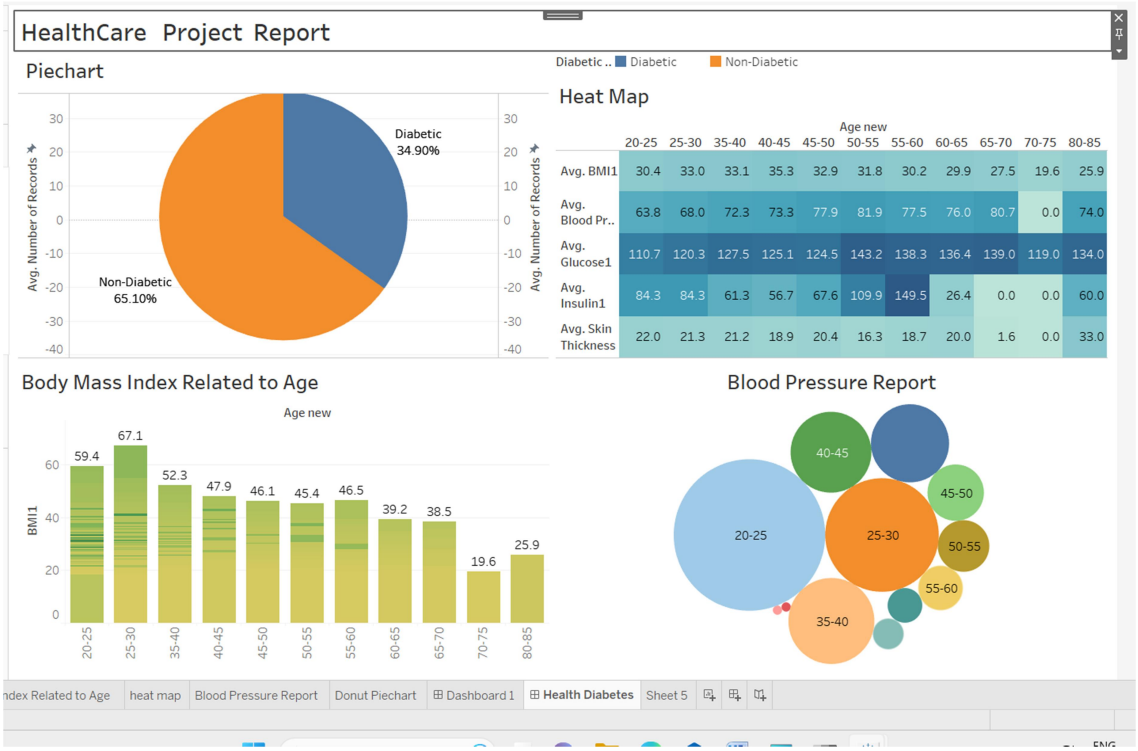
```
]: <matplotlib.legend.Legend at 0x2207e0bdcd0>
```



Observation:

We can see the SVM Classifier is the best among all. Accuracy of SVM is better than KNN. I also consider it to be the best. F1-score and recall (the fraction of relevant instances that were retrieved) is far better than other models. With Addition its ROC Curve AUC score is 85%. That is good.

Methodology for Data Reporting



WebLink:-

[health care diabetes | Tableau Public]

https://public.tableau.com/app/profile/baman.sravanthi/viz/healthcarediabetes_16891399317590/HealthDiabetes?publish=yes