Data-Science-Capstone-Healthcare

Report created by : Kaushik Prasad Dey

Topic : NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) – research for the most chronic, costly, and consequential diseases and predict whether a patient has diabetes, based on certain diagnostic measurements.

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). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

VARIABLES	DESCRIPTION

Data-Science-Capstone-Healthcare

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

Tools Used For this Project

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
   - 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.

Data-Science-Capstone-Healthcare

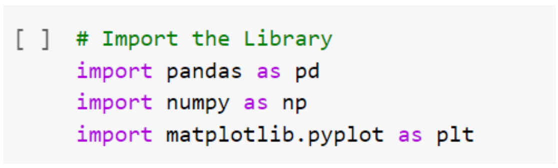
5. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

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

Methodology for Task for Week -1

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

1) **Import** all necessary library for this project. The Screenshot is given below.



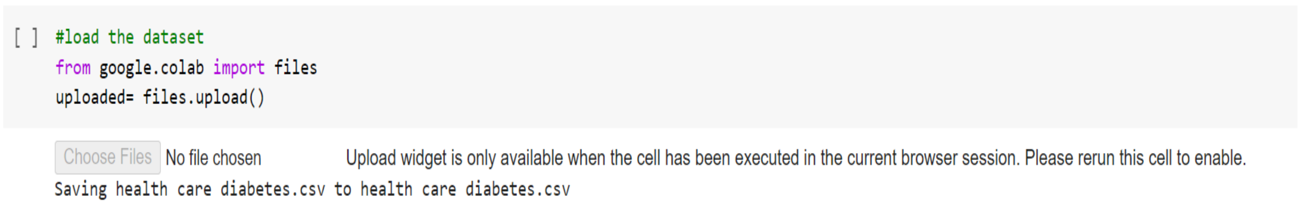
```
[ ] # Import the Library
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
```

2) Next, we have to read the required csv file. The screen-shot is given below.

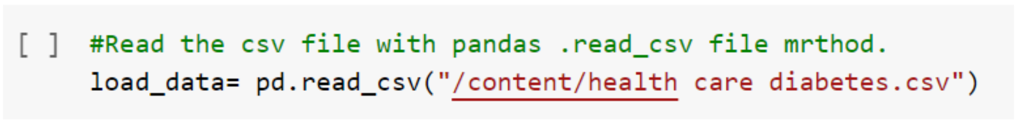
Note:

a) As using in google-colab note-book for this project, two extra code I have to add as a screen-shot.

b) Data set name is **"health care diabetes.csv"**.



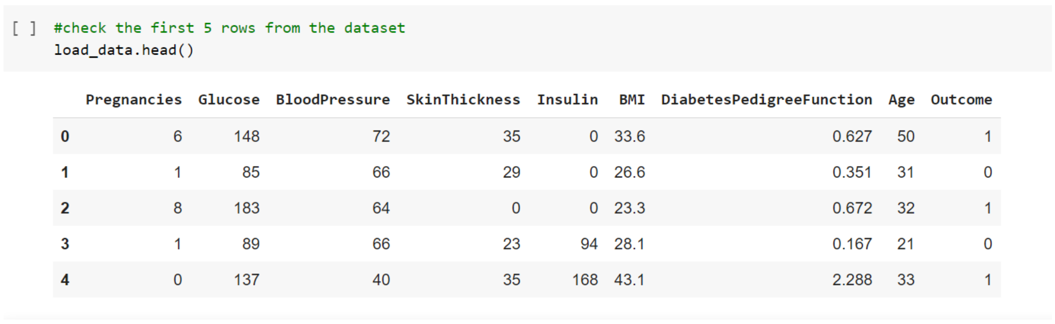
```
[ ] #load the dataset
    from google.colab import files
    uploaded= files.upload()

    Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving health care diabetes.csv to health care diabetes.csv
```



```
[ ] #Read the csv file with pandas .read_csv file mrthod.
    load_data= pd.read_csv("/content/health care diabetes.csv")
```

3) After reading csv file some following steps we have to perform for more details knowledge of this csv data set.

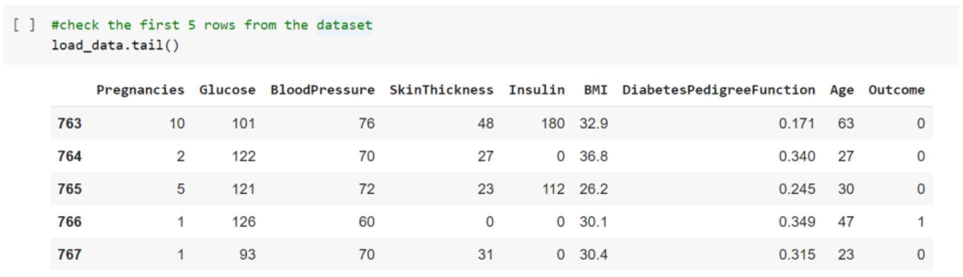
3.1 Read first and last five rows from the dataset.



```
[ ] #check the first 5 rows from the dataset
    load_data.head()
```

	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

Data-Science-Capstone-Healthcare

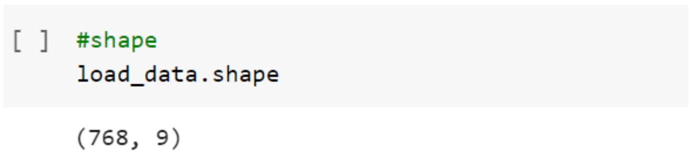


```
[ ] #check the first 5 rows from the dataset
    load_data.tail()
```

	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

3.2 Now, find out the shape, five point summary, statistical information & how many null values are present inside this data-set.

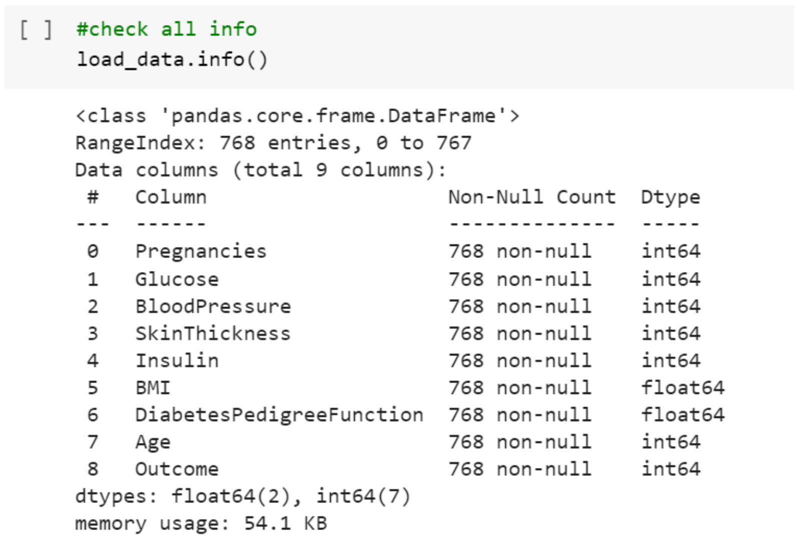
Shape: It helps to find out rows & columns from the data set.



```
[ ] #shape
    load_data.shape

    (768, 9)
```

So, we got to know that we have **768 rows and 9 columns** in that dataset.

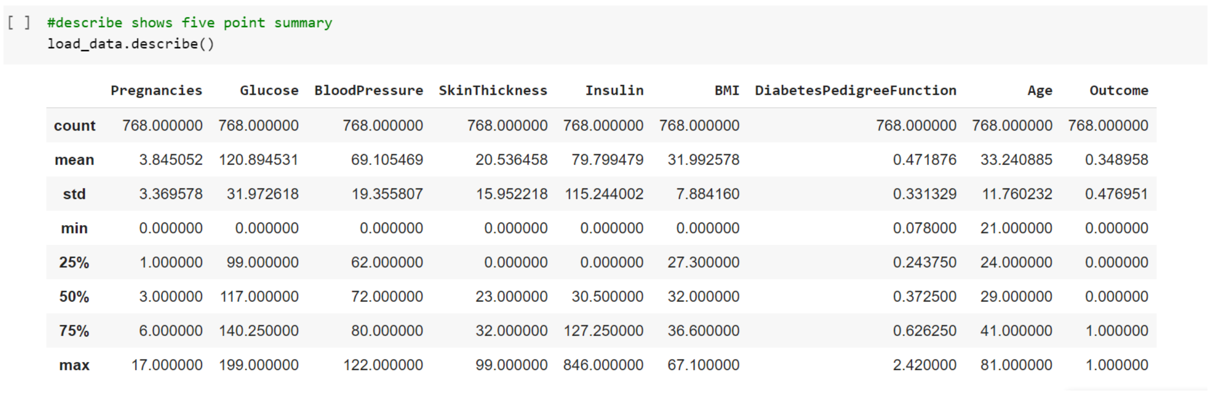
Info: It prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values). Note: the info() method actually prints the info.



```
[ ] #check all info
    load_data.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 description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column: count - The number of not-empty values. mean - The average (mean) value. std - The standard deviation.

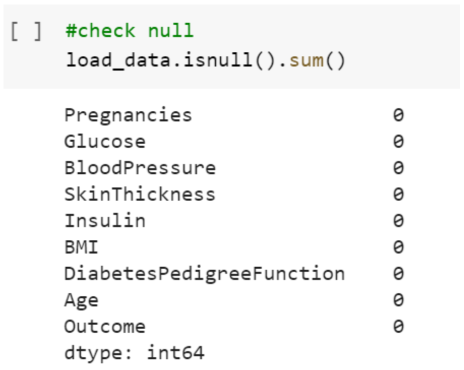
Data-Science-Capstone-Healthcare



```
[ ] #describe shows five point summary
    load_data.describe()
```

	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

Isnull & sum: It returns how many null values are present inside this dataset & its total count.



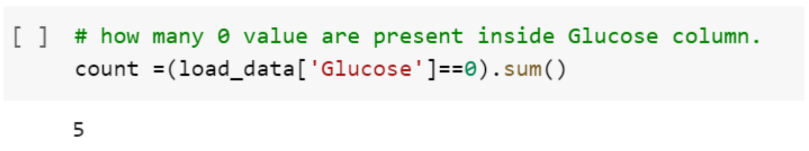
```
[ ] #check null
    load_data.isnull().sum()

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

Perform Descriptive Analysis

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

Case for Variable Glucose

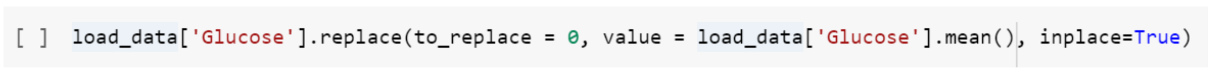
Before treating missing value(in my case 0) .



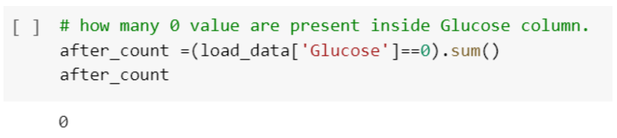
```
[ ] # how many 0 value are present inside Glucose column.
    count =(load_data['Glucose']==0).sum()

    5
```

Handle the missing value(in my case 0)

Here, we treated the 0 value as null value & have to replace with mean() for Gloucose column.



```
[ ] load_data['Glucose'].replace(to_replace = 0, value = load_data['Glucose'].mean(), inplace=True)
```



```
[ ] # how many 0 value are present inside Glucose column.
    after_count =(load_data['Glucose']==0).sum()
    after_count

    0
```

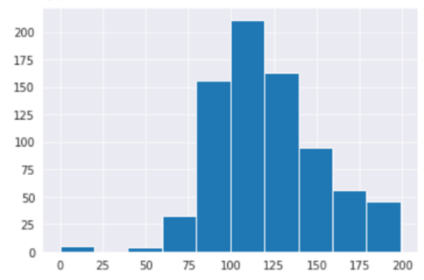
If, we compare the Histogram before & after the missing value treatment, it shows like that

Data-Science-Capstone-Healthcare

Before

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

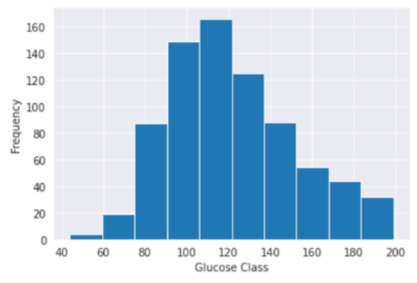
```
Mean of Glucose level is :- 120.89453125
Datatype of Glucose Variable is: int64
```



After

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

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



Case for Variable BloodPressure

Before treating missing value(in my case 0)

```
[ ] # how many 0 value are present inside Glucose column.
    BloodPressure_before_count =(load_data['BloodPressure']==0).sum()
    BloodPressure_before_count
```

```
35
```

Handle the missing value (in my case 0)

Here, we treated the 0 value as null value & have to replace with mean() for Gloucose column.BloodPressure:

```
[ ] load_data['BloodPressure'].replace(to_replace = 0, value = load_data['BloodPressure'].mean(), inplace=True)
```

```
[ ] #check BloodPressure after replace 0 value.
    BloodPressure_after_count =(load_data['BloodPressure']==0).sum()
    BloodPressure_after_count
```

```
0
```

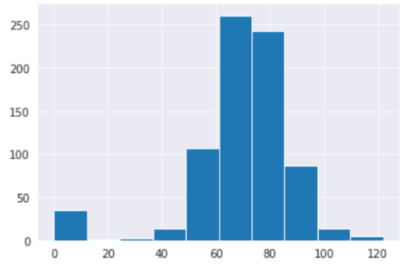
If, we compare the Histogram before & after the missing value treatment, it shows like that

Data-Science-Capstone-Healthcare

Before

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

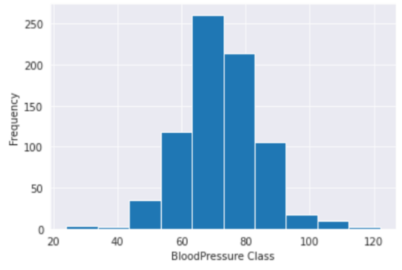
```
Mean of BloodPressure level is :- 69.10546875
Datatype of BloodPressure Variable is: int64
```



After

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

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



Case for Variable Skin-Thickness

Before treating missing value (in my case 0)

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

    227
```

Handle the missing value (in my case 0)

Here, we treated the 0 value as null value & have to replace with mean() for Gloucose column.SkinThickness.

```
[ ] load_data['SkinThickness'].replace(to_replace = 0, value = load_data['SkinThickness'].mean(), inplace=True)
```

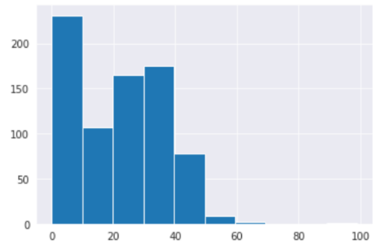
```
[ ] #How many 0 columns are present we have to check.
    SkinThickness_after_count =(load_data['SkinThickness']==0).sum()
    SkinThickness_after_count

    0
```

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

Before

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

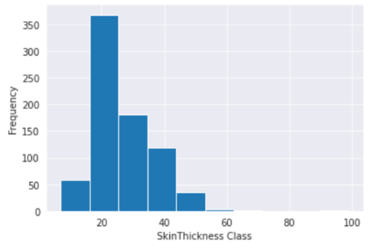
```
Mean of SkinThickness level is :- 20.536458333333332
Datatype of SkinThickness Variable is: int64
```



After

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

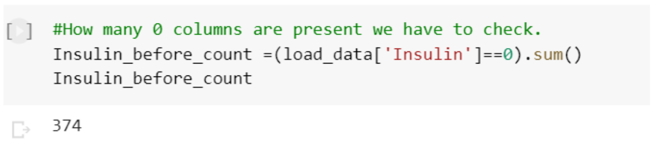
```
Mean of SkinThickness is :- 26.606479220920136
Datatype of SkinThickness Variable is: float64
```



Data-Science-Capstone-Healthcare

Case for Variable Insulin

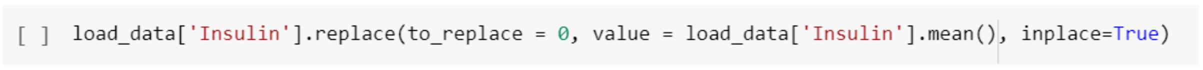
Before treating missing value (in my case 0)



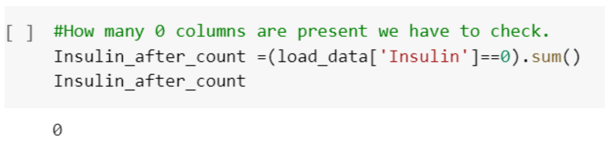
```
[ ] #How many 0 columns are present we have to check.
    Insulin_before_count =(load_data['Insulin']==0).sum()
    Insulin_before_count

    374
```

Handle the missing value (in my case 0)

Here, we treated the 0 value as null value & have to replace with mean() for Gloucose column.Insulin.



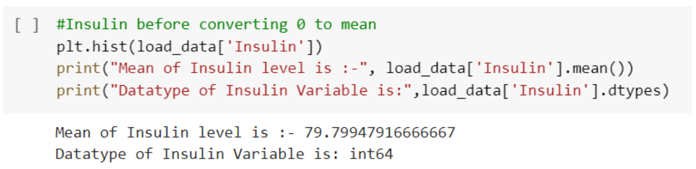
```
[ ]  load_data['Insulin'].replace(to_replace = 0, value = load_data['Insulin'].mean(), inplace=True)
```



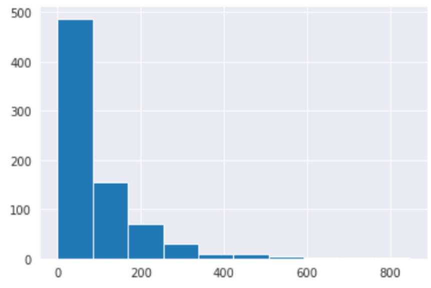
```
[ ] #How many 0 columns are present we have to check.
    Insulin_after_count =(load_data['Insulin']==0).sum()
    Insulin_after_count

    0
```

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

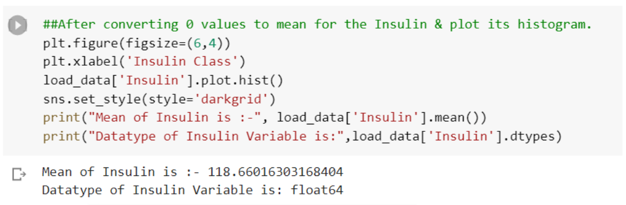
Before



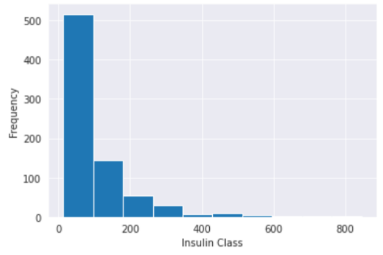
```
[ ] #Insulin before converting 0 to mean
    plt.hist(load_data['Insulin'])
    print("Mean of Insulin level is :-", load_data['Insulin'].mean())
    print("Datatype of Insulin Variable is:",load_data['Insulin'].dtypes)

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



After

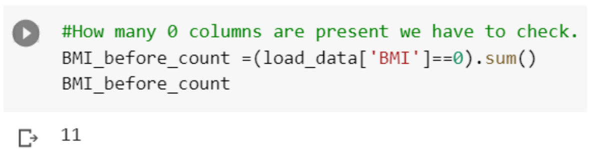


```
##After converting 0 values to mean for the Insulin & plot its histogram.
plt.figure(figsize=(6,4))
plt.xlabel('Insulin Class')
load_data['Insulin'].plot.hist()
sns.set_style(style='darkgrid')
print("Mean of Insulin is :-", load_data['Insulin'].mean())
print("Datatype of Insulin Variable is:",load_data['Insulin'].dtypes)

Mean of Insulin is :- 118.66016303168404
Datatype of Insulin Variable is: float64
```



Case for Variable BMI

Before treating missing value (in my case 0)

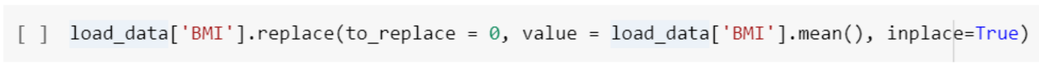


```
#How many 0 columns are present we have to check.
BMI_before_count =(load_data['BMI']==0).sum()
BMI_before_count

11
```

Data-Science-Capstone-Healthcare

Handle the missing value (in my case 0)

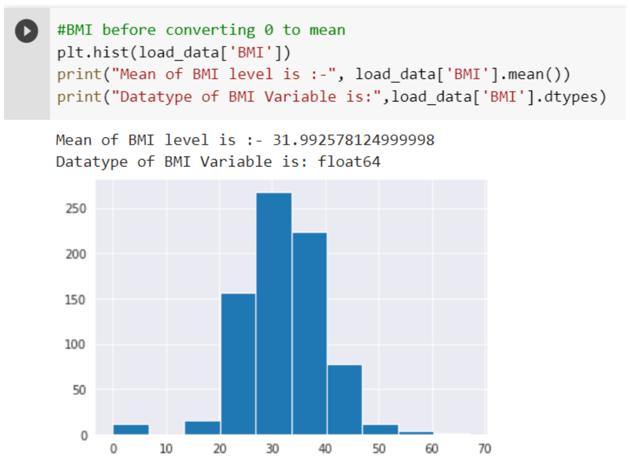
Here, we treated the 0 value as null value & have to replace with mean() for BMI column.



```
[ ]  load_data['BMI'].replace(to_replace = 0, value = load_data['BMI'].mean(), inplace=True)
```

If, we compare the Histogram before & after the missing value treatment, it shows like that.

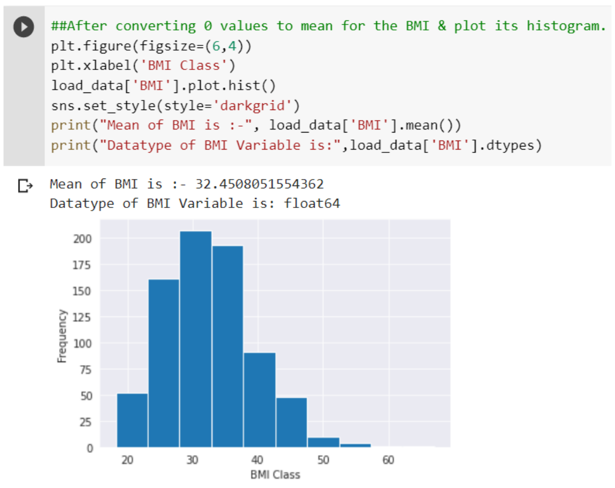
Before



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

```
Mean of BMI level is :- 31.992578124999998
Datatype of BMI Variable is: float64
```

After



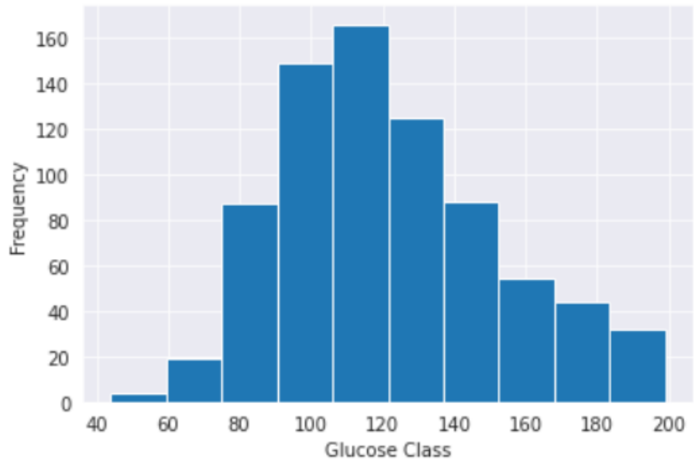
```
##After converting 0 values to mean for the BMI & plot its histogram.
plt.figure(figsize=(6,4))
plt.xlabel('BMI Class')
load_data['BMI'].plot.hist()
sns.set_style(style='darkgrid')
print("Mean of BMI is :-", load_data['BMI'].mean())
print("Datatype of BMI Variable is:",load_data['BMI'].dtypes)
```

```
Mean of BMI is :- 32.4508051554362
Datatype of BMI Variable is: float64
```

So, finally all histogram are looks like in below

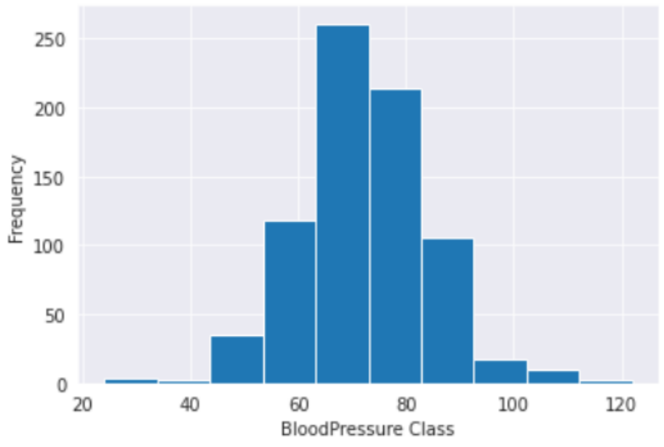
Histogram for Glucose

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



Histogram for Blood-Pressure

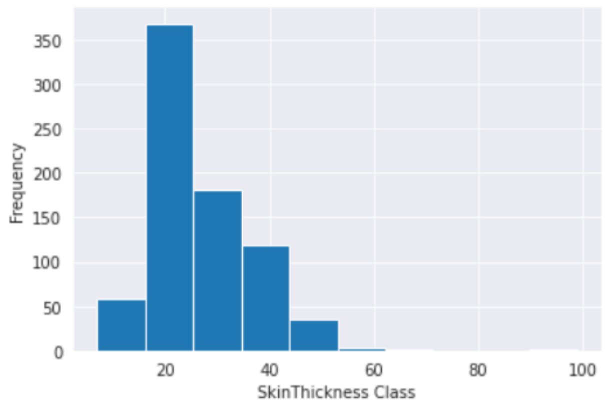
```
Mean of BloodPressure level is :- 72.25480651855469
Datatype of BloodPressure Variable is: float64
```



Data-Science-Capstone-Healthcare

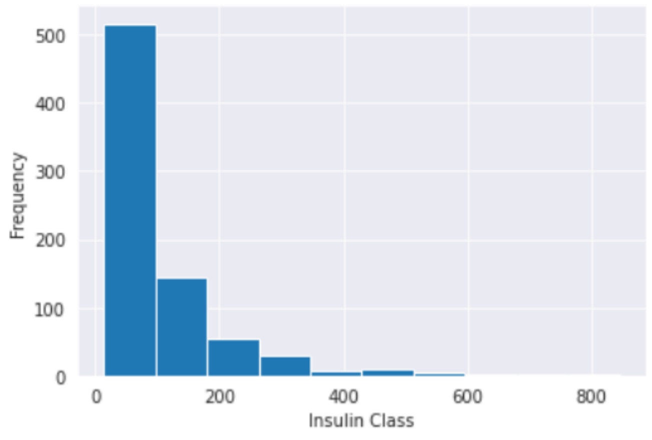
Histogram for Skin-ThickNess

```
Mean of SkinThickness is :- 26.606479220920136
Datatype of SkinThickness Variable is: float64
```



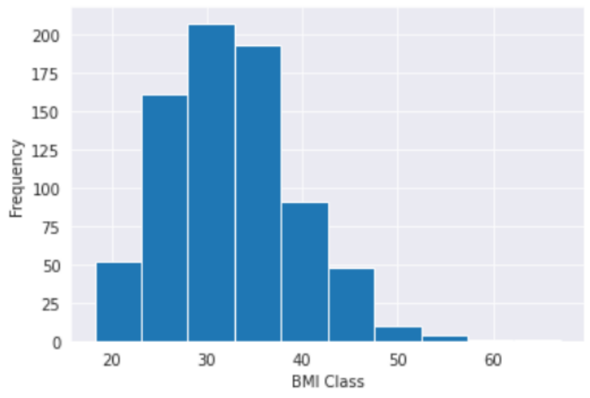
Histogram for Insulin

```
Mean of Insulin is :- 118.66016303168404
Datatype of Insulin Variable is: float64
```



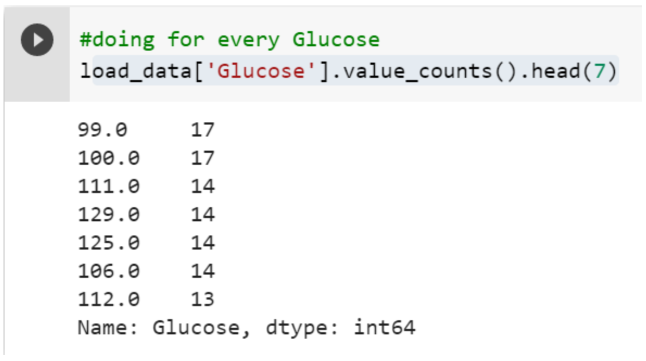
Histogram for BMI

```
Mean of BMI is :- 32.4508051554362
Datatype of BMI Variable is: float64
```



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

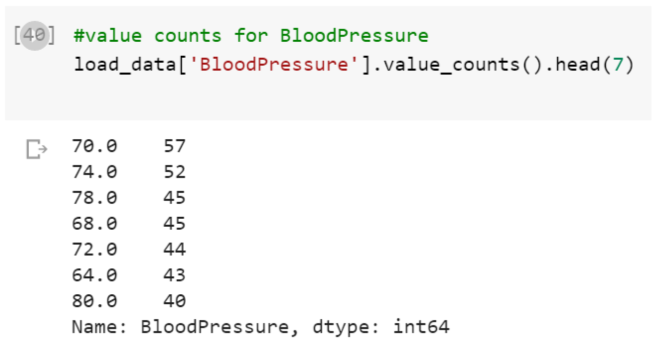
Count Table for Variable Glucose



```
#doing for every Glucose
load_data['Glucose'].value_counts().head(7)
```

```
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
```

Count Table for Variable Blood Pressure



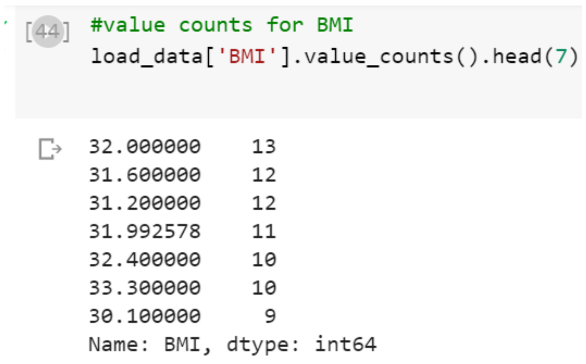
```
[40] #value counts for BloodPressure
load_data['BloodPressure'].value_counts().head(7)
```

```
70.0    57
74.0    52
78.0    45
68.0    45
72.0    44
64.0    43
80.0    40
Name: BloodPressure, dtype: int64
```

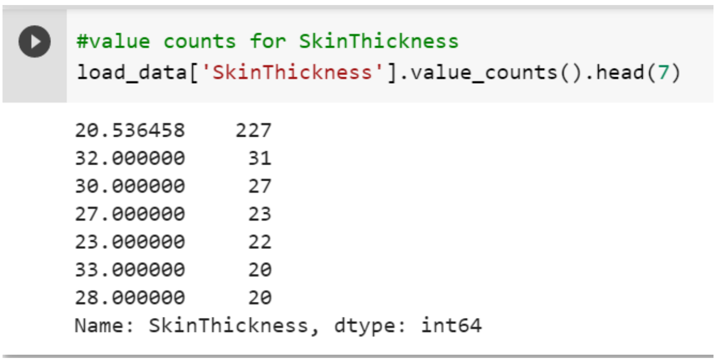
Count Table for Variable BMI

Count Table for Variable SkinThickness

Data-Science-Capstone-Healthcare

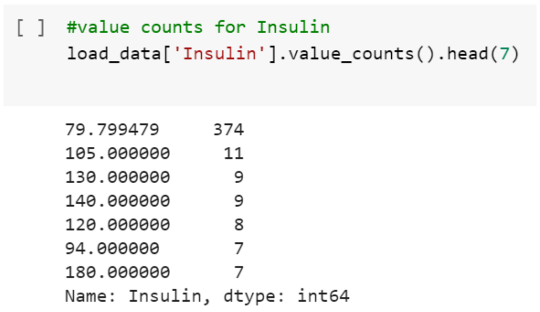


```
[44] #value counts for BMI
     load_data['BMI'].value_counts().head(7)

     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
```



```
#value counts for SkinThickness
load_data['SkinThickness'].value_counts().head(7)

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
```

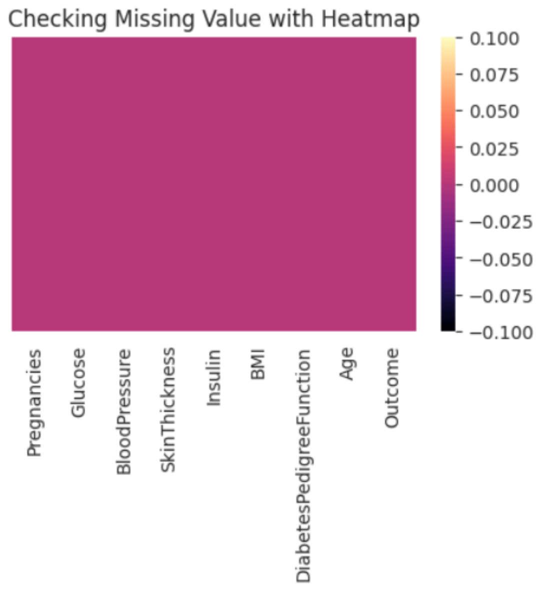
Count Table for Insulin



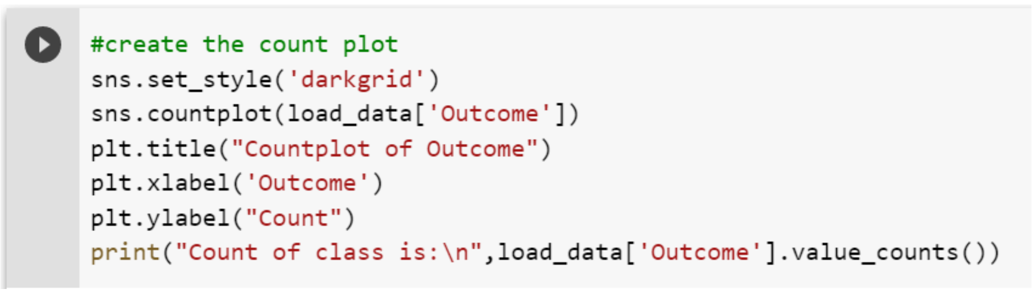
```
[ ] #value counts for Insulin
    load_data['Insulin'].value_counts().head(7)

    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
```

Checking Missing Values with Heat-Map

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

<matplotlib.axes._subplots.AxesSubplot at 0x7efde34749d0>
```

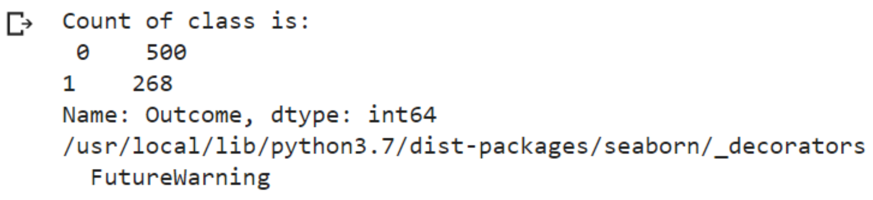


Plotting the count of outcomes by their values.

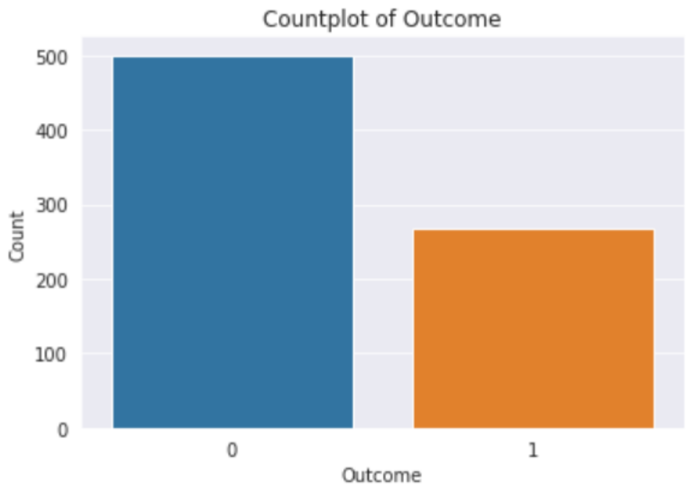


```
#create the count plot
sns.set_style('darkgrid')
sns.countplot(load_data['Outcome'])
plt.title("Countplot of Outcome")
plt.xlabel('Outcome')
plt.ylabel("Count")
print("Count of class is:\n",load_data['Outcome'].value_counts())
```

Data-Science-Capstone-Healthcare



```
Count of class is:
 0    500
1    268
Name: Outcome, dtype: int64
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators
  FutureWarning
```



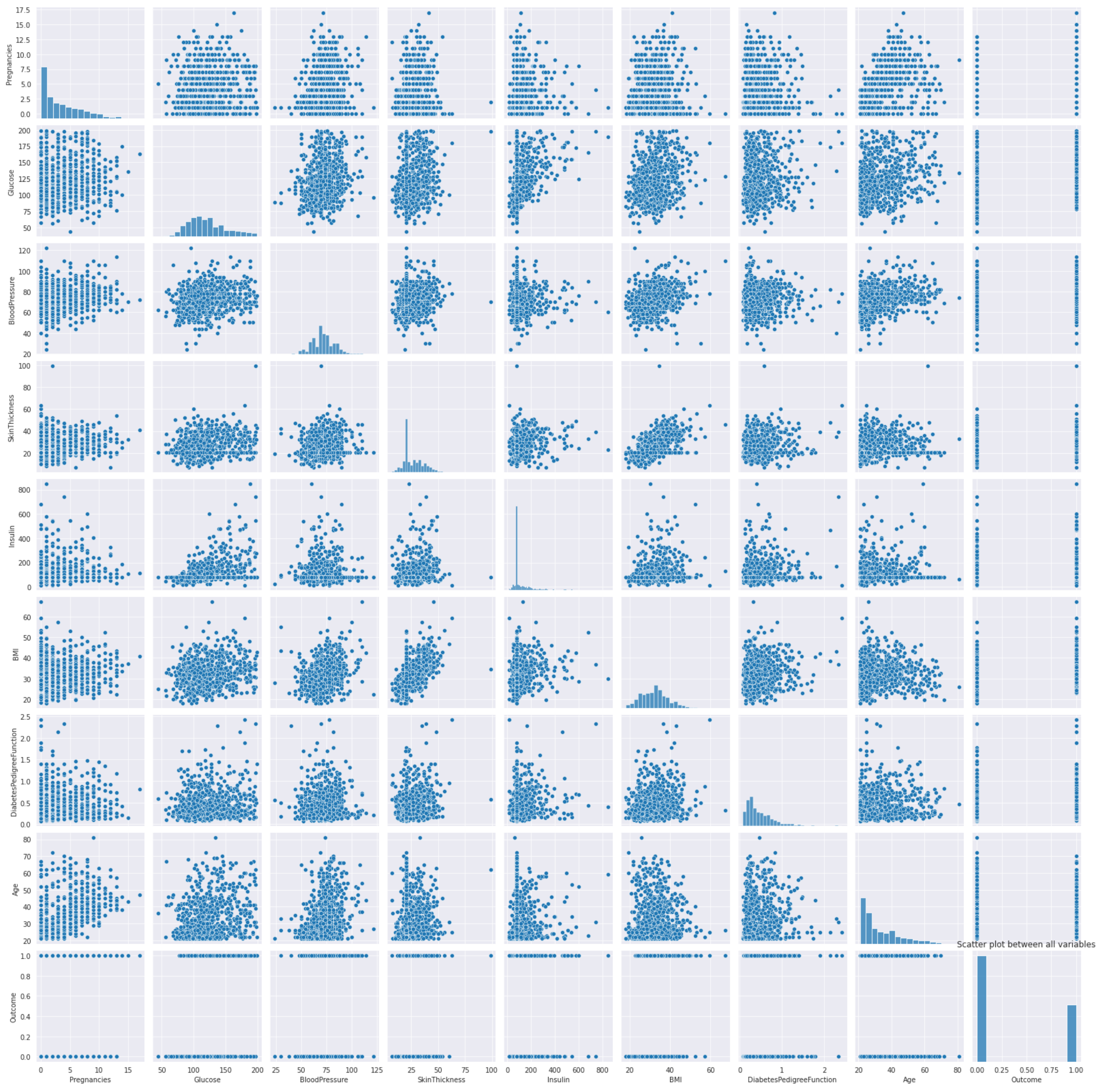
Describe the finding

We can see that both class is balanced so we need not to perform any sampling method to maintain the balance between both classes. Therefor directly using this data in training and testing purpose without performing any sampling method. Meanwhile during Model Validation , we also need not worry about ROC Curve because data is not imbalanced, but as this is a medical data so I will be using ROC curve to make sure TYPE 2 ERROR will not be there.

Scatter plot

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

Data-Science-Capstone-Healthcare



Describe the observation

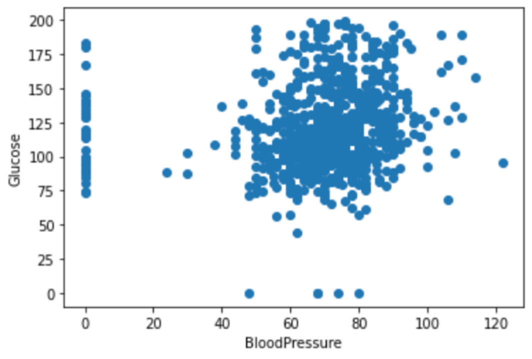
We can see from scatter plot that there is no strong multicolinearity among features, but between skin thickness and BMI, Pregnancies and age it looks like there is small chance of positive correlation.

Also create scatter plot to identify the co relation between variables observation.

Data-Science-Capstone-Healthcare

Scatter plot with Glucose and Blood-Pressure

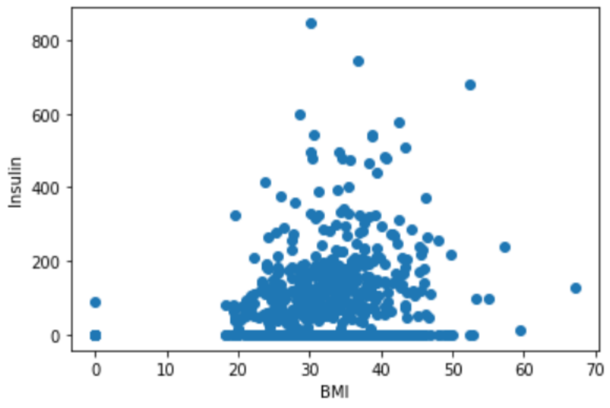
```
#creating a scatter plot
plt.scatter(load_data['BloodPressure'], load_data['Glucose'])
plt.xlabel("BloodPressure") #x label
plt.ylabel("Glucose") #y label
plt.show()
```



So, from this observation we can seen that the positive (+) co-relation is happened between BloodPressure & Glucose.

Scatter plot with BMI and Insulin

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



So, from this observation we can seen that the positive (+) co-relation is happened between BMI & Insulin.

Perform Co-relation Analysis with heat map

```
load_data.corr()
```

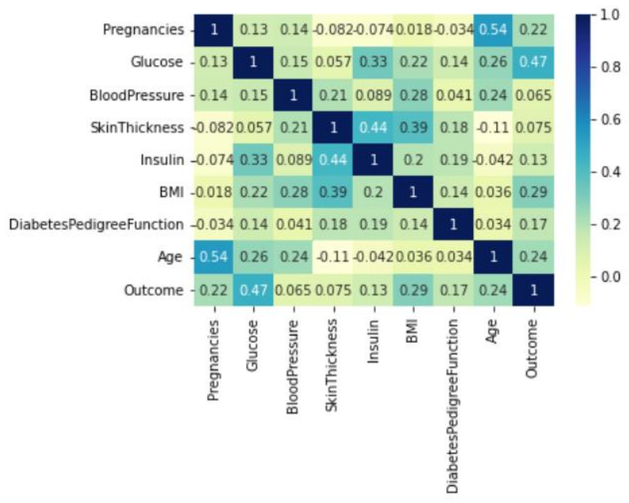
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

Data-Science-Capstone-Healthcare

Heat map with Theme(YlGnBu)

```
import seaborn as sb
sb.heatmap(load_data.corr(), cmap="YlGnBu", annot=True)
```

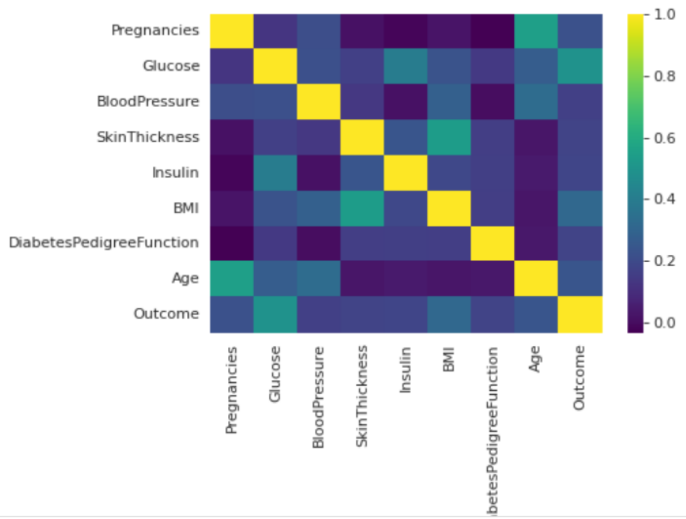
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28788976d0>
```



Heat map with Theme(virdis)

```
plt.figure(dpi=80)
sns.heatmap(load_data.corr(),cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffb91cf8c10>
```



Project Task: Week 2

Data Modeling

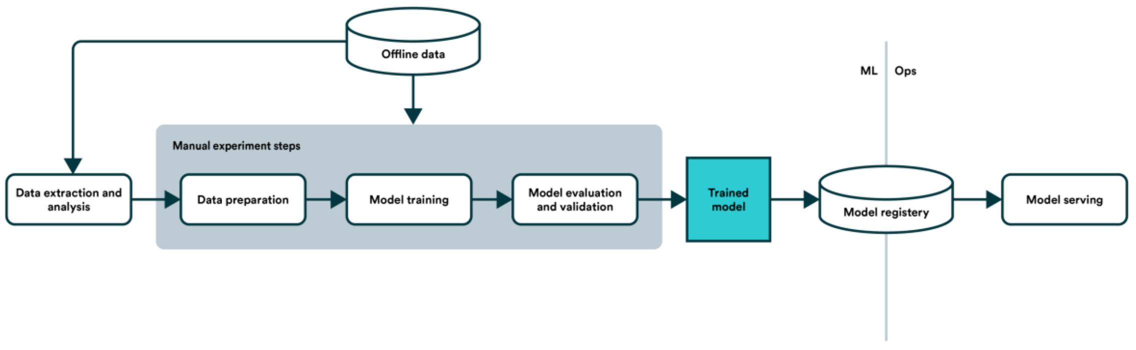
1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model.
3. Compare various models with the results from KNN algorithm.
4. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

Please be descriptive to explain what values of these parameter you have used.

Methodology for Task for Week -2

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

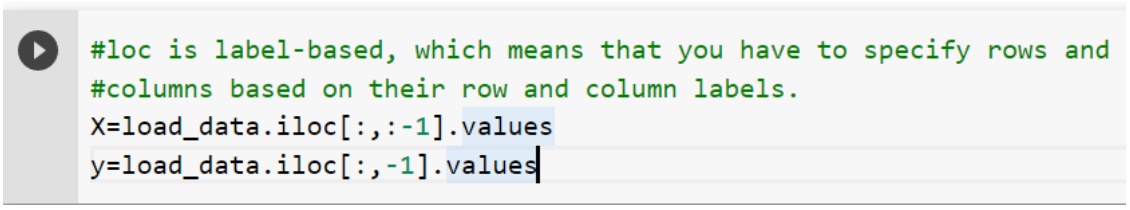
So, from the below data pipeline we have to reach that training model stage but currently we are



in data processing stage.

Data-Science-Capstone-Healthcare

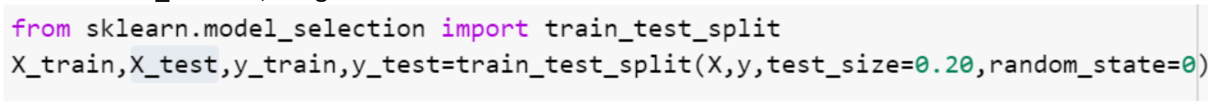
Step -1 :



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

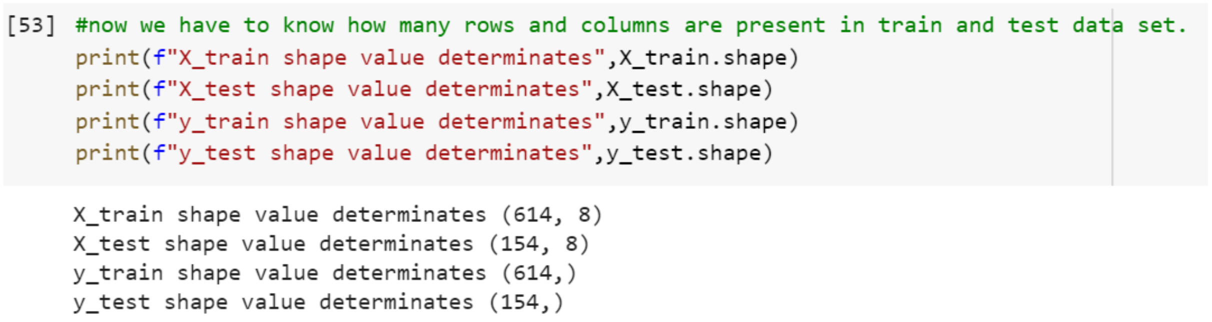
Step -2 (Splitting the data-set into training and testing part)

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.
With random_state=0 , we get the same train and test sets across different executions.



```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
```

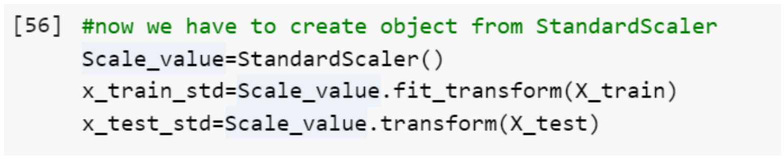
Step -3 (To identify how many rows and columns are present in X_train, X_test, y_train and y_test.



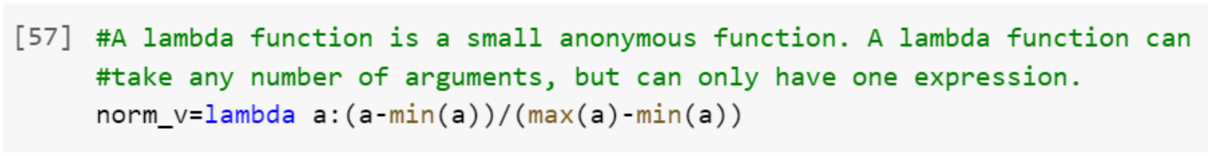
```
[53] #now we have to know how many rows and columns are present in train and test data set.
     print(f"X_train shape value determinates",X_train.shape)
     print(f"X_test shape value determinates",X_test.shape)
     print(f"y_train shape value determinates",y_train.shape)
     print(f"y_test shape value determinates",y_test.shape)

     X_train shape value determinates (614, 8)
     X_test shape value determinates (154, 8)
     y_train shape value determinates (614,)
     y_test shape value determinates (154,)
```

Step -4 : Now we have to import StandardScaler which

use to standardize the data values into a standard format

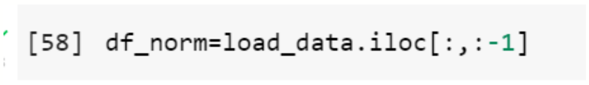


```
[56] #now we have to create object from StandardScaler
     Scale_value=StandardScaler()
     x_train_std=Scale_value.fit_transform(X_train)
     x_test_std=Scale_value.transform(X_test)
```

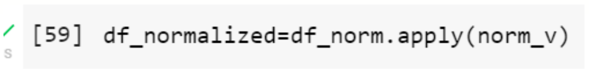


```
[57] #A lambda function is a small anonymous function. A lambda function can
     #take any number of arguments, but can only have one expression.
     norm_v=lambda a:(a-min(a))/(max(a)-min(a))
```

Step -5 : we again validated with the shape value of X_train_form, X_test_form, y_train_form & y_test_form.

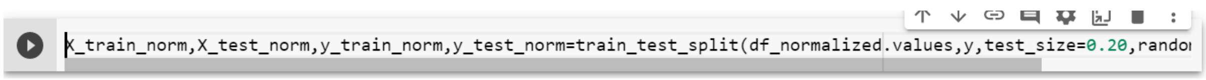


```
[58] df_norm=load_data.iloc[:,:-1]
```

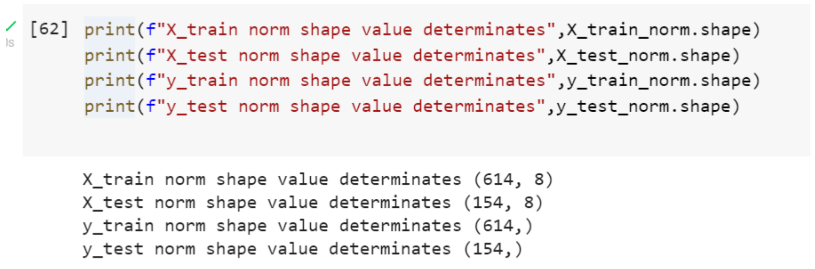


```
[59] df_normalized=df_norm.apply(norm_v)
```

Data-Science-Capstone-Healthcare



```
X_train_norm,X_test_norm,y_train_norm,y_test_norm=train_test_split(df_normalized.values,y,test_size=0.20,rando
```



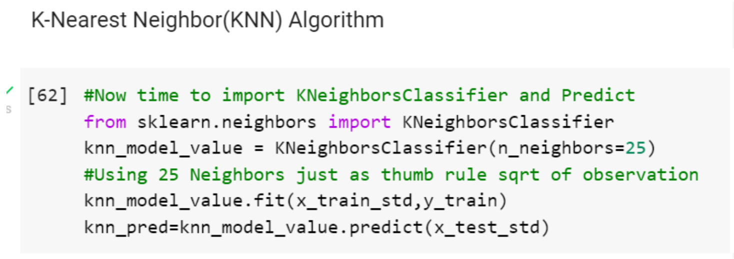
```
[62] print(f"X_train norm shape value determinates",X_train_norm.shape)
     print(f"X_test norm shape value determinates",X_test_norm.shape)
     print(f"y_train norm shape value determinates",y_train_norm.shape)
     print(f"y_test norm shape value determinates",y_test_norm.shape)
```

```
X_train norm shape value determinates (614, 8)
X_test norm shape value determinates (154, 8)
y_train norm shape value determinates (614,)
y_test norm shape value determinates (154,)
```

Now Starting Different Model Building Process (KNN)

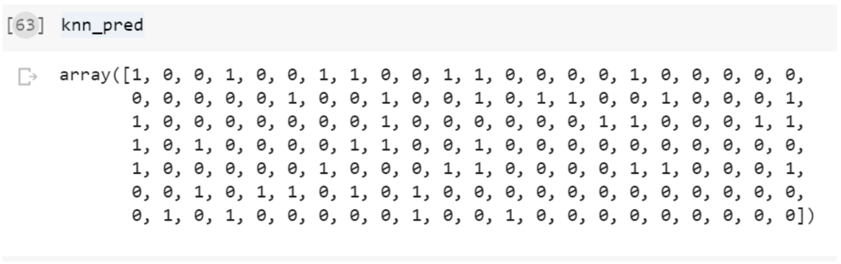
KNN (KNN with Standard Scaling)

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



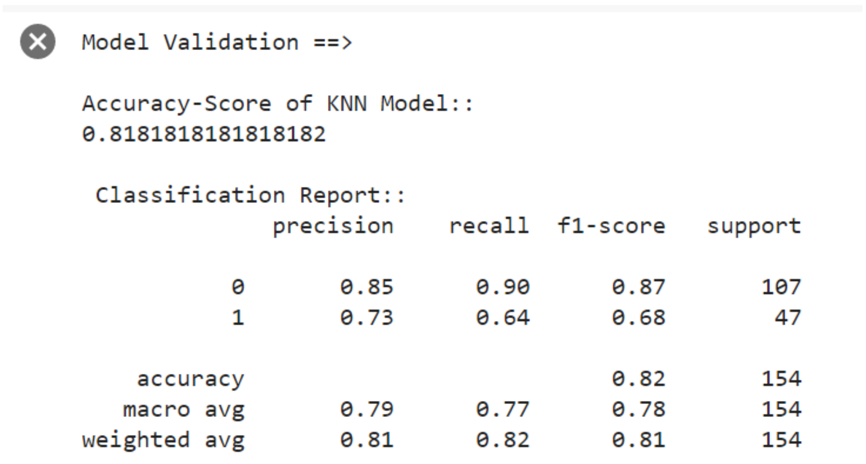
K-Nearest Neighbor(KNN) Algorithm

```
[62] #Now time to import KNeighborsClassifier and Predict
     from sklearn.neighbors import KNeighborsClassifier
     knn_model_value = KNeighborsClassifier(n_neighbors=25)
     #Using 25 Neighbors just as thumb rule sqrt of observation
     knn_model_value.fit(x_train_std,y_train)
     knn_pred=knn_model_value.predict(x_test_std)
```

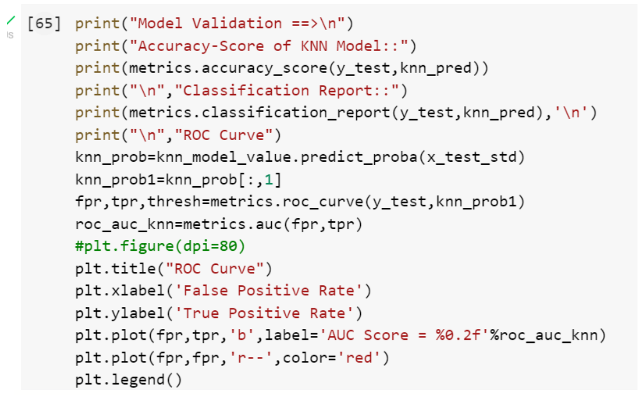


```
[63] knn_pred
```

```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 0, 1, 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, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```



```
Model Validation ==>

Accuracy-Score of KNN Model::
0.8181818181818182

 Classification Report::
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       107
           1       0.73      0.64      0.68        47

    accuracy                           0.82       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.81      0.82      0.81       154
```

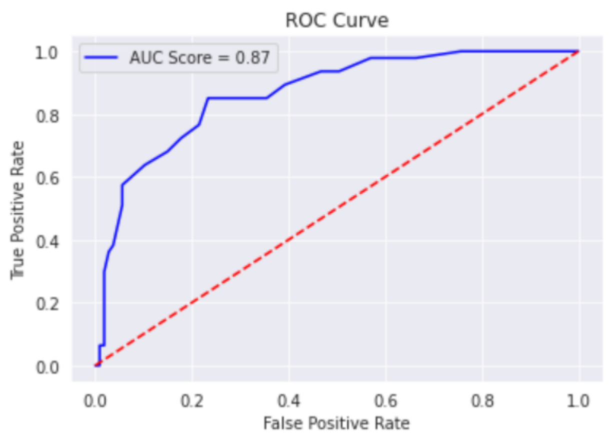


```
[65] print("Model Validation ==>\n")
     print("Accuracy-Score of KNN Model::")
     print(metrics.accuracy_score(y_test,knn_pred))
     print("\n","Classification Report::")
     print(metrics.classification_report(y_test,knn_pred),'\n')
     print("\n","ROC Curve")
     knn_prob=knn_model_value.predict_proba(x_test_std)
     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()
```

Data-Science-Capstone-Healthcare

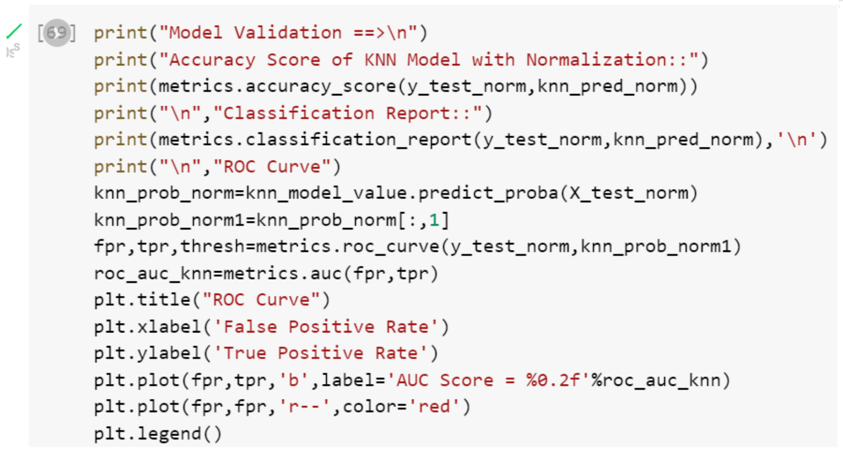
OBSERVATION

```
ROC Curve
<matplotlib.legend.Legend at 0x7f19bfb18a10>
```

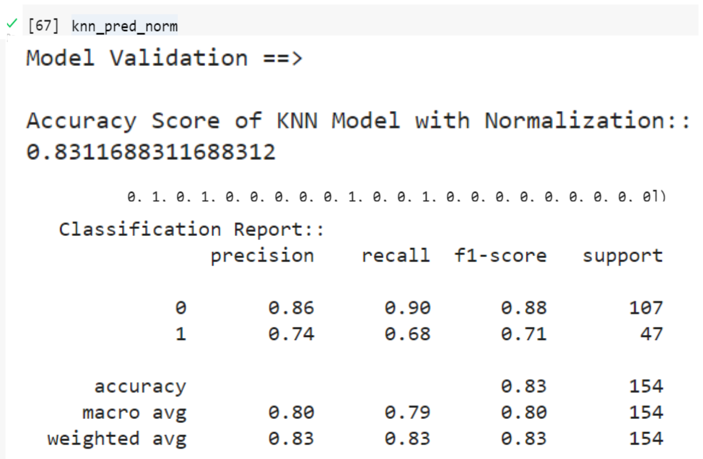


SO, from KNN standard scaling we can got the Model Accuracy is 0.81 and from the classification report we got its Precision, recall, f1-scoare and support value. With Addition its ROC Curve AUC score is 87%. That is good.

KNN-Normalization



```
[69] print("Model Validation ==>\n")
     print("Accuracy Score of KNN Model with Normalization::")
     print(metrics.accuracy_score(y_test_norm,knn_pred_norm))
     print("\n","Classification Report::")
     print(metrics.classification_report(y_test_norm,knn_pred_norm),'\n')
     print("\n","ROC Curve")
     knn_prob_norm=knn_model_value.predict_proba(X_test_norm)
     knn_prob_norm1=knn_prob_norm[:,1]
     fpr,tpr,thresh=metrics.roc_curve(y_test_norm,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='red')
     plt.legend()
```



```
[67] knn_pred_norm
Model Validation ==>

Accuracy Score of KNN Model with Normalization::
0.8311688311688312

         0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0])
  Classification Report::
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       107
           1       0.74      0.68      0.71        47

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

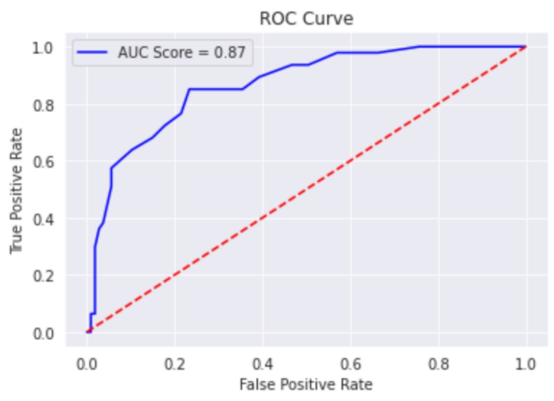
Data-Science-Capstone-Healthcare

OBSERVATION

we compare, the KNN-Standardization and Normalization from the below ROC curve.

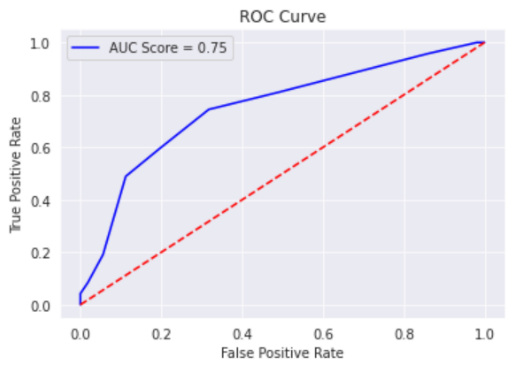
KNN- Standardization

```
ROC Curve
<matplotlib.legend.Legend at 0x7f19bfb18a10>
```



KNN- Normalization

```
ROC Curve
<matplotlib.legend.Legend at 0x7f19c2422c90>
```

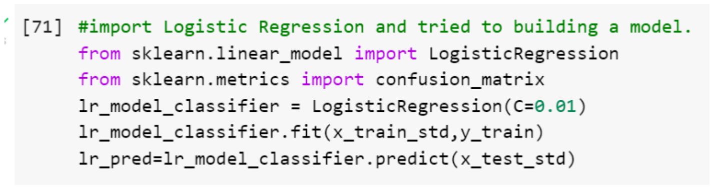


We can clearly see that KNN with Standardization is better than Normalization.

Now Starting Different Model Building Process (LOGSTIC REGRESSION)

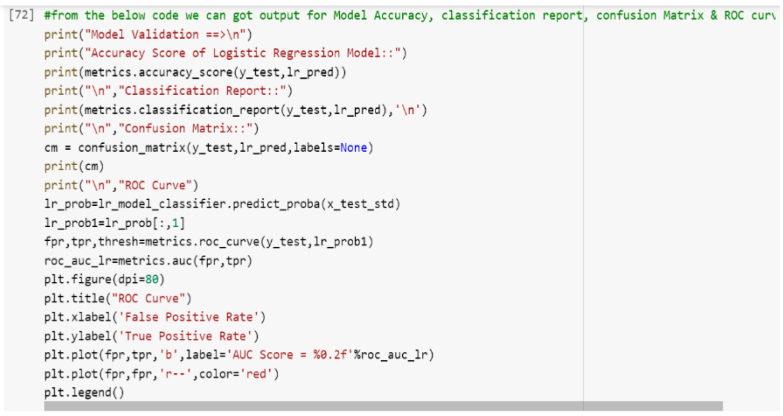
Logistic regression is an example of **supervised learning**. It is used to calculate or predict the probability of a binary (yes/no) event occurring.

Model Creation:



```
[71] #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_std,y_train)
     lr_pred=lr_model_classifier.predict(x_test_std)
```

From the below code we can got output for Model Accuracy, classification report, confusion Matrix & R OC curve:



```
[72] #from the below code we can got output for Model Accuracy, classification report, confusion Matrix & ROC curv
     print("Model Validation ==>\n")
     print("Accuracy Score of Logistic Regression Model::")
     print(metrics.accuracy_score(y_test,lr_pred))
     print("\n","Classification Report::")
     print(metrics.classification_report(y_test,lr_pred),'\n')
     print("\n","Confusion Matrix::")
     cm = confusion_matrix(y_test,lr_pred,labels=None)
     print(cm)
     print("\n","ROC Curve")
     lr_prob=lr_model_classifier.predict_proba(x_test_std)
     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()
```

Model Accuracy :

```
Model Validation ==>

Accuracy Score of Logistic Regression Model::
0.8116883116883117
```

Classification Report:

```
Classification Report::
              precision    recall  f1-score   support

           0       0.82      0.93      0.87       107
           1       0.78      0.53      0.63        47

    accuracy                           0.81       154
   macro avg       0.80      0.73      0.75       154
weighted avg       0.81      0.81      0.80       154
```

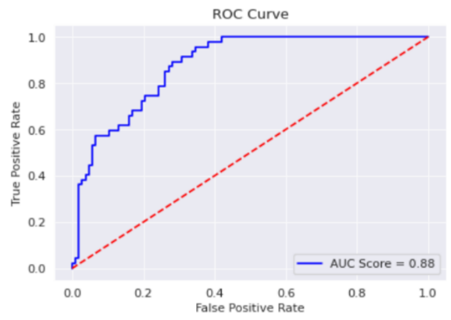
Data-Science-Capstone-Healthcare

Confusion Matrix:

```
 Confusion Matrix::
[[100   7]
 [ 22  25]]
```

ROC CURVE:

```
ROC Curve
<matplotlib.legend.Legend at 0x7fa81bc6b290>
```



Observation:

Accuracy of KNN is better than Logistic Regression,but auc score of Logistic regression is better.

Now Starting Different Model Building Process (Decision Tree Classifier)

Decision Tree 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.

Model Creation:

```
[84] #import Decesion Tree Classifier and tried to building a model.
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import confusion_matrix
     dtc_model_classifier = DecisionTreeClassifier()
     dtc_model_classifier.fit(x_train_std,y_train)
     dtc_pred=dtc_model_classifier.predict(x_test_std)
```

From the below code we can got output for Model Accuracy, classification report, confusion Matrix & R OC curve:

```
[85] print("Model Validation ==>\n")
     print("Accuracy Score of Decesion 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_std)
     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()
```

Model Accuracy :

```
Accuracy Score of Decesion Tree Classfier Model::
0.7402597402597403
```

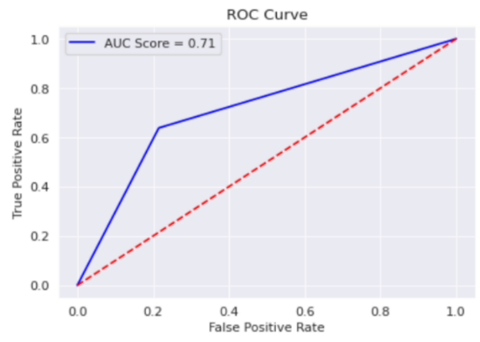
Classification Report :

```
 Classification Report::
              precision    recall  f1-score   support

           0       0.83      0.79      0.81       107
           1       0.57      0.64      0.60        47

    accuracy                           0.74       154
   macro avg       0.70      0.71      0.70       154
weighted avg       0.75      0.74      0.74       154
```

Data-Science-Capstone-Healthcare

ROC CURVE :

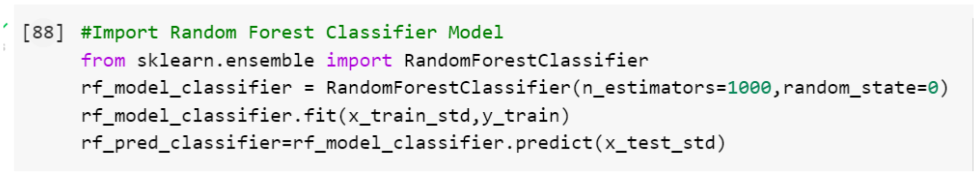
```
ROC Curve
<matplotlib.legend.Legend at 0x7fa81b7f54d0>
```



Now Starting Different Model Building Process (Random Forest Classifier)

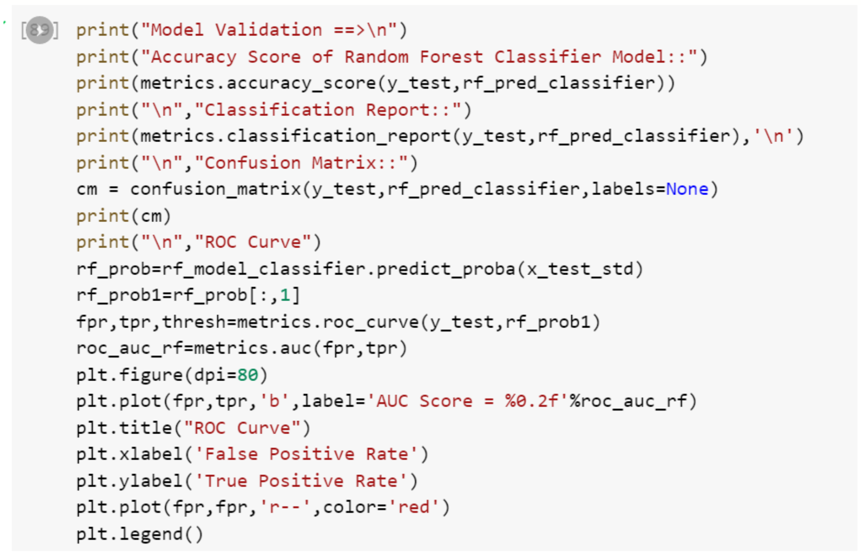
A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Model Creation:



```
[88] #Import Random Forest Classifier Model
     from sklearn.ensemble import RandomForestClassifier
     rf_model_classifier = RandomForestClassifier(n_estimators=1000,random_state=0)
     rf_model_classifier.fit(x_train_std,y_train)
     rf_pred_classifier=rf_model_classifier.predict(x_test_std)
```

From the below code we can got output for Model Accuracy, classification report, confusion Matrix & R OC curve:



```
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_std)
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()
```

Model Accuracy:

```
Accuracy Score of Random Forest Classifier Model::
0.8246753246753247
```

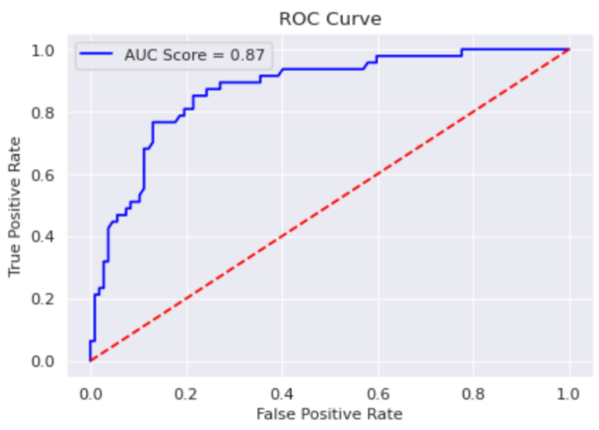
Data-Science-Capstone-Healthcare

Classification Report

```
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
```

ROC Curve

```
ROC Curve
<matplotlib.legend.Legend at 0x7fa81b457950>
```



Observation from All Model Decision

So we can see Random Forest Classifier is best among all, you also wondering auc score is lesser by 1 than others also am considering it to be best because balance of classes between Precision (is the fraction of relevant instances among the retrieved instances) **and Recall (** fraction of relevant instances that were retrieved) **is far better than other Models. So we can consider a loss in AUC by 1.**

Model Deployment

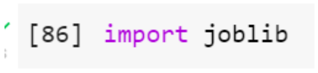
Deployment is the method by which you integrate a machine learning model into an existing production environment to make practical business decisions based on data. It is one of the last stages in the machine learning life cycle and can be one of the most cumbersome.

Introduction to JOB-LIB

Joblib is a set of tools to provide **lightweight pipelining in Python.** In particular: transparent disk-caching of functions and lazy re-evaluation (memoize pattern) easy simple parallel computing.

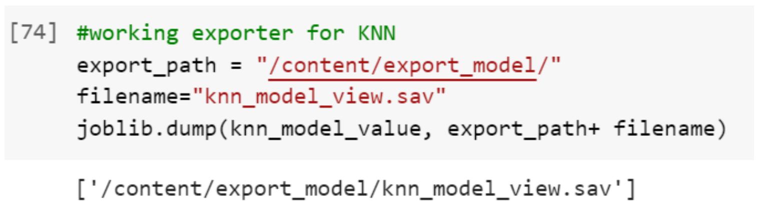
Data-Science-Capstone-Healthcare

IMPORT JOBLIB

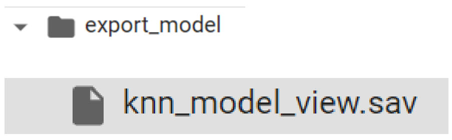


```
[86]  import joblib
```

KNN Model (Export):



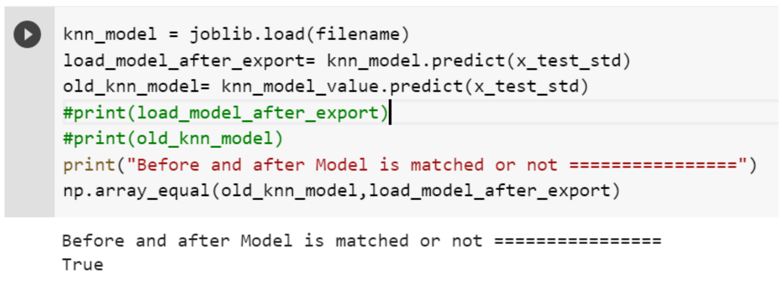
```
[74]  #working exporter for KNN
      export_path = "/content/export_model/"
      filename="knn_model_view.sav"
      joblib.dump(knn_model_value, export_path+ filename)

      ['/content/export_model/knn_model_view.sav']
```



export_model

knn_model_view.sav

Before and After Model is Matched or Not ?

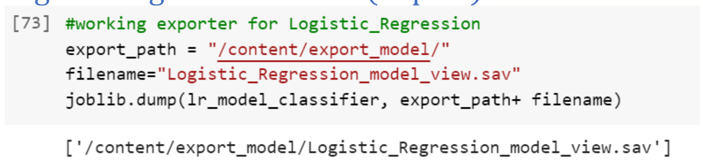


```
knn_model = joblib.load(filename)
load_model_after_export= knn_model.predict(x_test_std)
old_knn_model= knn_model_value.predict(x_test_std)
#print(load_model_after_export)
#print(old_knn_model)
print("Before and after Model is matched or not ================")
np.array_equal(old_knn_model,load_model_after_export)

Before and after Model is matched or not ================
True
```

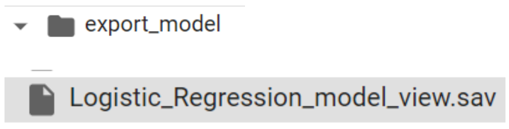
Observation

So, when we started model prediction and when we export the model and import both array value are same.

Logistic Regression Model (Export):



```
[73]  #working exporter for Logistic_Regression
      export_path = "/content/export_model/"
      filename="Logistic_Regression_model_view.sav"
      joblib.dump(lr_model_classifier, export_path+ filename)

      ['/content/export_model/Logistic_Regression_model_view.sav']
```

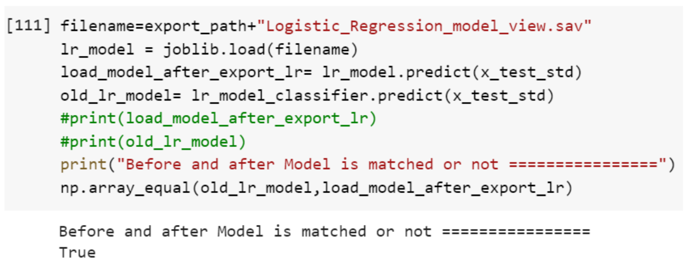


export_model

Logistic_Regression_model_view.sav

Data-Science-Capstone-Healthcare

Before and After Model is Matched or Not ?

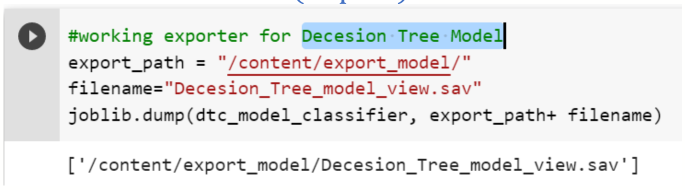


```
[111] filename=export_path+"Logistic_Regression_model_view.sav"
      lr_model = joblib.load(filename)
      load_model_after_export_lr= lr_model.predict(x_test_std)
      old_lr_model= lr_model_classifier.predict(x_test_std)
      #print(load_model_after_export_lr)
      #print(old_lr_model)
      print("Before and after Model is matched or not ================")
      np.array_equal(old_lr_model,load_model_after_export_lr)

      Before and after Model is matched or not ================
      True
```

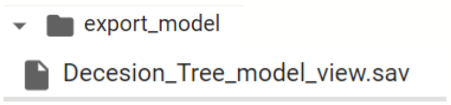
Observation

So, when we started model prediction and when we export the model and import both array value are same.

Decision Tree Model (Export):

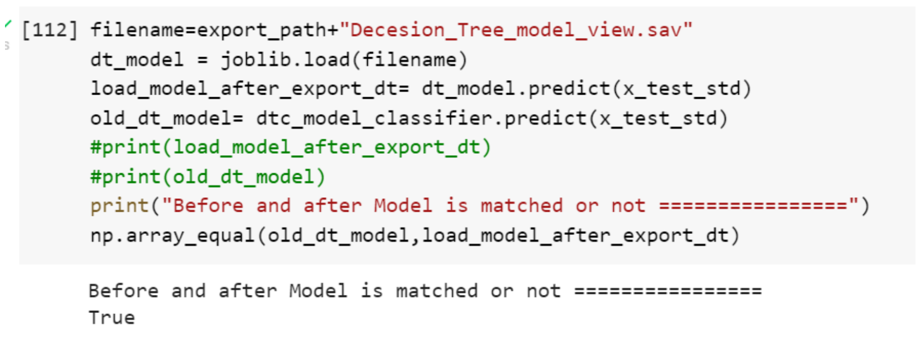


```
#working exporter for Decesion Tree Model
export_path = "/content/export_model/"
filename="Decesion_Tree_model_view.sav"
joblib.dump(dtc_model_classifier, export_path+ filename)

['/content/export_model/Decesion_Tree_model_view.sav']
```



```
export_model
  Decesion_Tree_model_view.sav
```

Before and After Model is Matched or Not ?

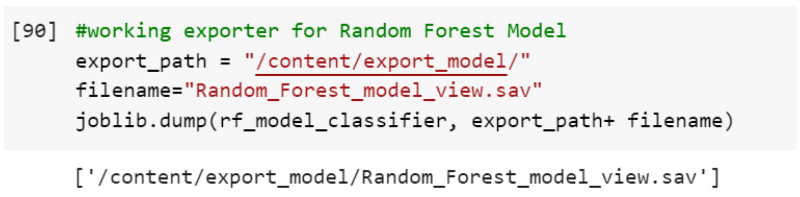


```
[112] filename=export_path+"Decesion_Tree_model_view.sav"
      dt_model = joblib.load(filename)
      load_model_after_export_dt= dt_model.predict(x_test_std)
      old_dt_model= dtc_model_classifier.predict(x_test_std)
      #print(load_model_after_export_dt)
      #print(old_dt_model)
      print("Before and after Model is matched or not ================")
      np.array_equal(old_dt_model,load_model_after_export_dt)

      Before and after Model is matched or not ================
      True
```

Observation

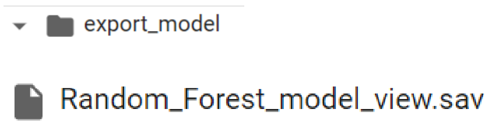
So, when we started model prediction and when we export the model and import both array value are same.

Random Forest Classifier (Export):



```
[90] #working exporter for Random Forest Model
     export_path = "/content/export_model/"
     filename="Random_Forest_model_view.sav"
     joblib.dump(rf_model_classifier, export_path+ filename)

     ['/content/export_model/Random_Forest_model_view.sav']
```

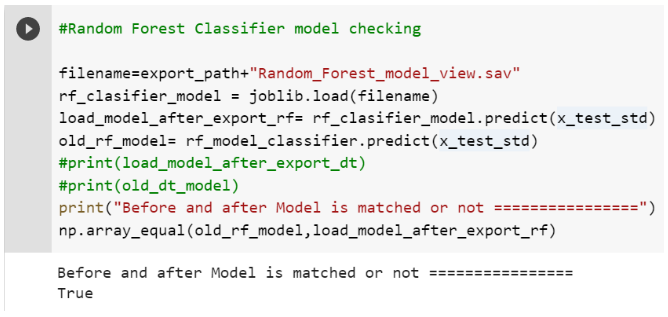
Data-Science-Capstone-Healthcare



export_model

Random_Forest_model_view.sav

Before and After Model is Matched or Not ?



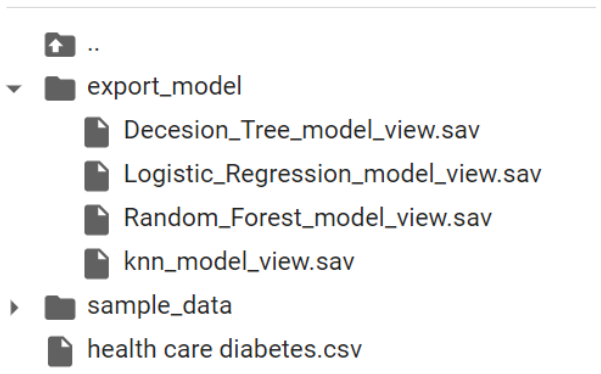
```
#Random Forest Classifier model checking

filename=export_path+"Random_Forest_model_view.sav"
rf_clasifier_model = joblib.load(filename)
load_model_after_export_rf= rf_clasifier_model.predict(x_test_std)
old_rf_model= rf_model_classifier.predict(x_test_std)
#print(load_model_after_export_dt)
#print(old_dt_model)
print("Before and after Model is matched or not ================")
np.array_equal(old_rf_model,load_model_after_export_rf)
```

```
Before and after Model is matched or not ================
True
```

Observation

So, when we started model prediction and when we export the model and import both array value are same.

Entire Project Structure



```
..
export_model
    Decesion_Tree_model_view.sav
    Logistic_Regression_model_view.sav
    Random_Forest_model_view.sav
    knn_model_view.sav
sample_data
health care diabetes.csv
```

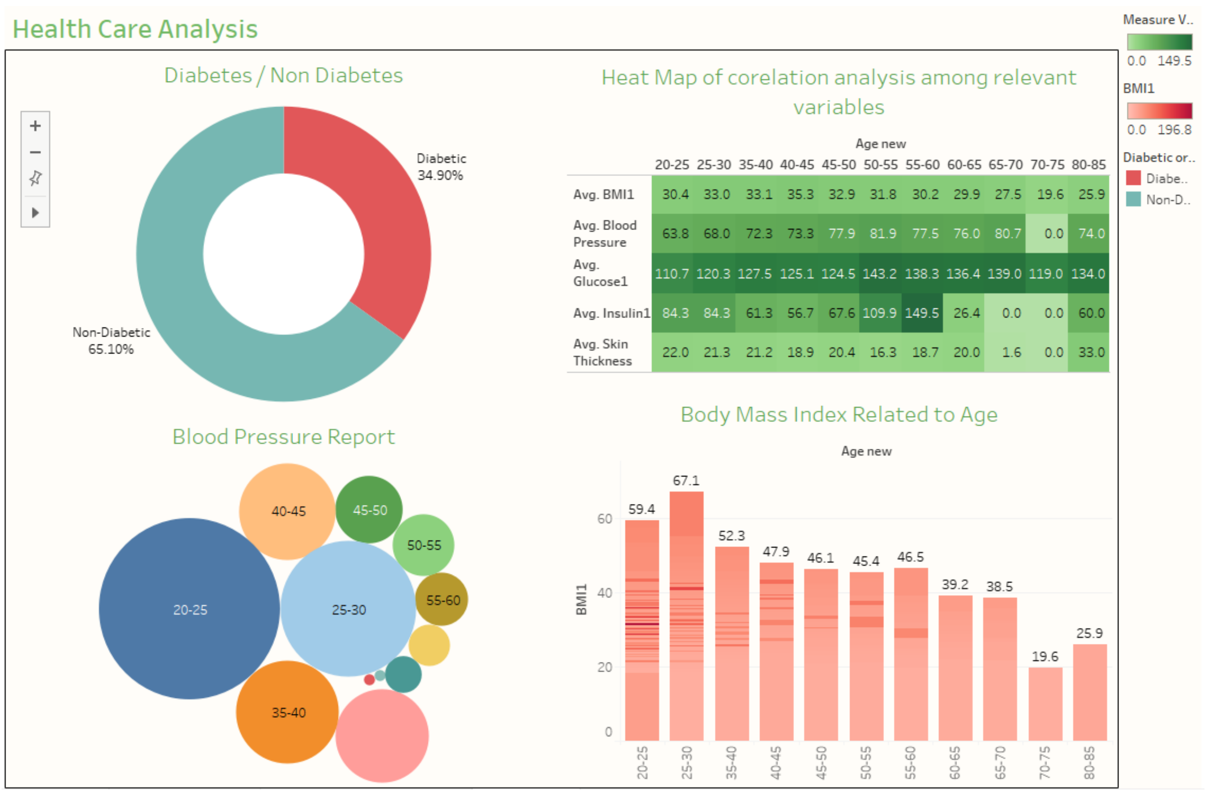
Data Reporting

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

- Pie chart to describe the diabetic or non-diabetic population
- Scatter charts between relevant variables to analyze the relationships
- Histogram or frequency charts to analyze the distribution of the data
- Heatmap of correlation analysis among the relevant variables
- Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

Data-Science-Capstone-Healthcare

Methodology for Data Reporting



Heat Map of corelation analysis among relevant variables

	Age new 20-25	25-30	35-40	40-45	45-50	50-55	55-60	60-65	65-70	70-75	80-85
Avg. BMI1	30.4	33.0	33.1	35.3	32.9	31.8	30.2	29.9	27.5	19.6	25.9
Avg. Blood Pressure	63.8	68.0	72.3	73.3	77.9	81.9	77.5	76.0	80.7	0.0	74.0
Avg. Glucose1	110.7	120.3	127.5	125.1	124.5	143.2	138.3	136.4	139.0	119.0	134.0
Avg. Insulin1	84.3	84.3	61.3	56.7	67.6	109.9	149.5	26.4	0.0	0.0	60.0
Avg. Skin Thickness	22.0	21.3	21.2	18.9	20.4	16.3	18.7	20.0	1.6	0.0	33.0

WebLink :
https://public.tableau.com/app/profile/kaushik.dey/viz/CapstoneProject_16577424000130/Dashboard1?publish=yes