import libaries

```
# Import libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# Visit https://ipython.readthedocs.io/en/stable/interactive/plotting.html for information on %matplotlib

import seaborn as sns

# Hide warnings if you are presenting your project to an audience to make your code look cleaner
import warnings
warnings.filterwarnings("ignore")
# Visit https://docs.python.org/3/library/warnings.html for information on warning control# Import library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# Visit https://ipython.readthedocs.io/en/stable/interactive/plotting.html for information on %matplotlib

import seaborn as sns

# Hide warnings if you are presenting your project to an audience to make your code look cleaner
import warnings
warnings.filterwarnings("ignore")
# Visit https://docs.python.org/3/library/warnings.html for information on warning control


data=pd.read_csv('/content/health care diabetes.csv')
```

Saved successfully!

WEEK 1

```
data.info(memory_usage='deep')
```

```
<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
```

Outcome is binary so classification or logistic regression can be used

```
data.head()
```

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

```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	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
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
			122.000000	99.000000	846.000000	67.100000

Saved successfully!

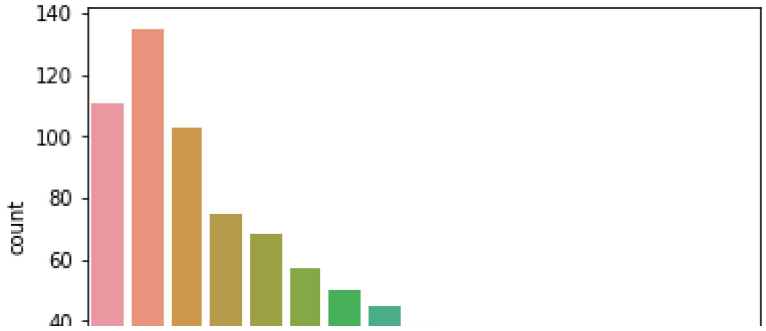
Need Standardization

no missing value

**Descriptive Analysis and visulaziation **

```
sns.countplot(data.Pregnancies)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f573e149410>
```

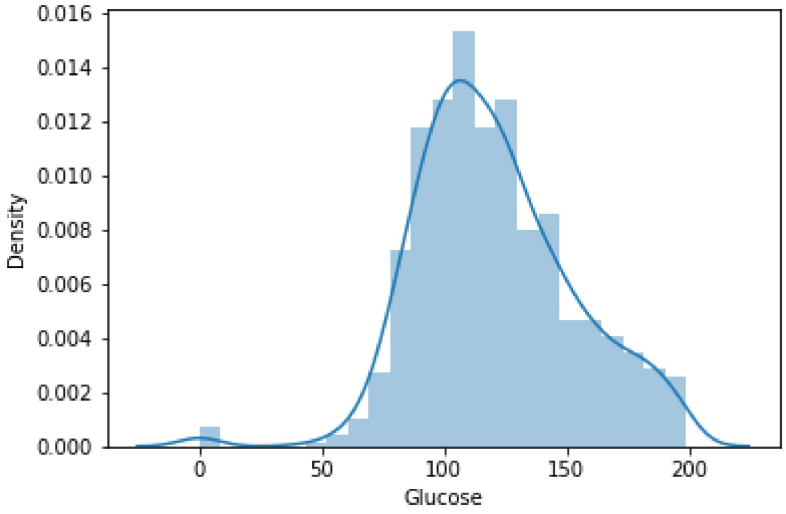


This can be treated as catagorical

Pregancies from 0 to 17

```
sns.distplot(data.Glucose)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f573dff6f50>
```



Clearly anything less than 50 is missing value USe median

Saved successfully!

rmal normal < 100 preddeb 100-125 deb >125

```
sns.distplot(data.BloodPressure)
```

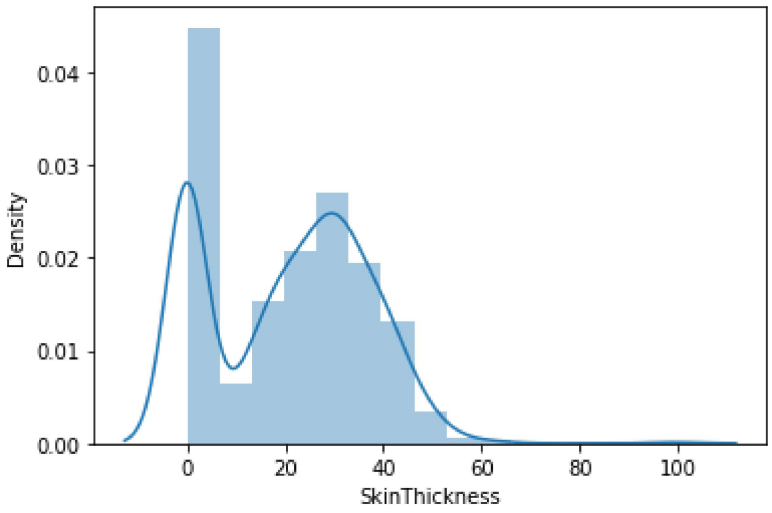
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f573db45250>
```


```
# This is formatted as code
```

Blood pressure <40 is missing value

```
sns.distplot(data.SkinThickness)
```

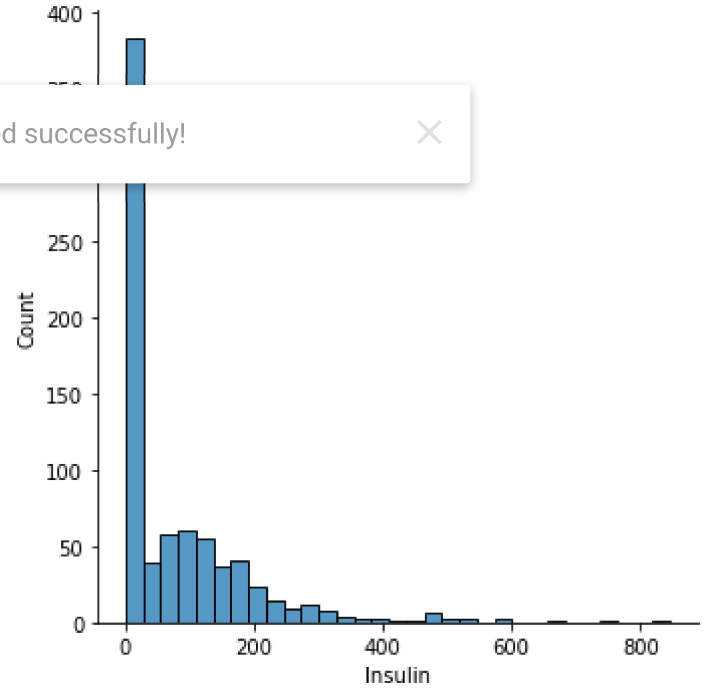
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f573b7ca590>
```



Skin Thickness zero not possible replace with mode

```
sns.displot(data.Insulin)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f573b78d290>
```

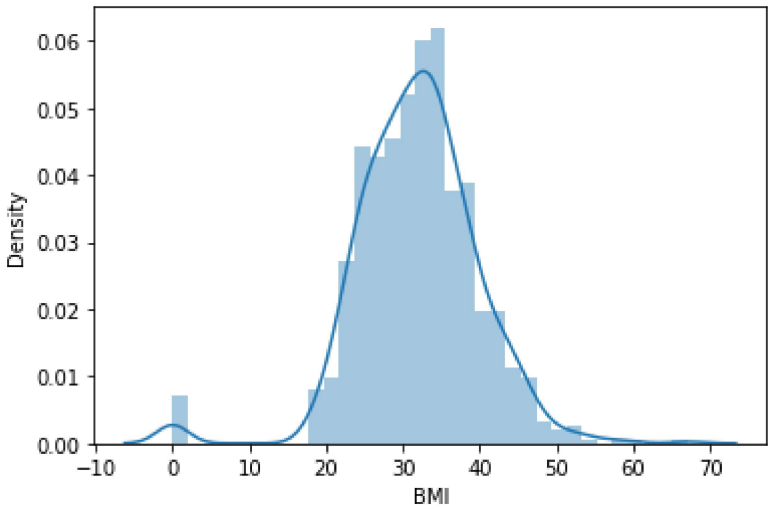


Saved successfully!

Insulin lvl will be replaced by median to don' disturb distribution

```
sns.distplot(data.BMI)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f573e149450>
```



Bmi with median

Double-click (or enter) to edit

Treating Missing Value

```
#!pip install feature-engine
```

```
#data.columns
```

```
feauture_col=['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI']
treated_data=data[feauture_col]
```

Saved successfully!

```
from feature_engine.imputation import MeanMedianImputer
num_imputer=MeanMedianImputer()
num_imputer.fit_transform(treated_data)
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
0	148	72	35	0	33.6
1	85	66	29	0	26.6
2	183	64	0	0	23.3
3	89	66	23	94	28.1

```
treated_data.describe()
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	120.894531	69.105469	20.536458	79.799479	31.992578
std	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	99.000000	62.000000	0.000000	0.000000	27.300000
50%	117.000000	72.000000	23.000000	30.500000	32.000000
75%	140.250000	80.000000	32.000000	127.250000	36.600000
max	199.000000	122.000000	99.000000	846.000000	67.100000

```
df=data[['Pregnancies','DiabetesPedigreeFunction', 'Age', 'Outcome']]
```

```
new_data=pd.concat([df,treated_data],axis=1)
```

```
new_data.head()
```

	Pregnancies	DiabetesPedigreeFunction	Age	Outcome	Glucose	BloodPressure	Skir
0	6	0.627	50	1	148	72	
1		0.351	31	0	85	66	
2	8	0.672	32	1	183	64	
3	1	0.167	21	0	89	66	
4	0	2.288	33	1	137	40	

Saved successfully!

```
new_data.info()
```

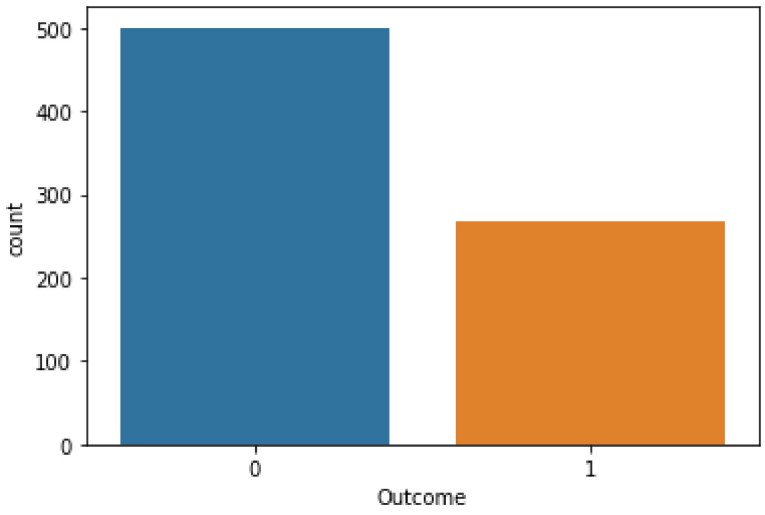
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Pregnancies              768 non-null    int64  
 1   DiabetesPedigreeFunction  768 non-null    float64
 2   Age                      768 non-null    int64  
 3   Outcome                  768 non-null    int64  
```

```
 4   Glucose                   768 non-null    int64
 5   BloodPressure             768 non-null    int64
 6   SkinThickness             768 non-null    int64
 7   Insulin                   768 non-null    int64
 8   BMI                       768 non-null    float64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Week 2

```
#balance
sns.countplot(new_data.Outcome)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5732199b10>
```



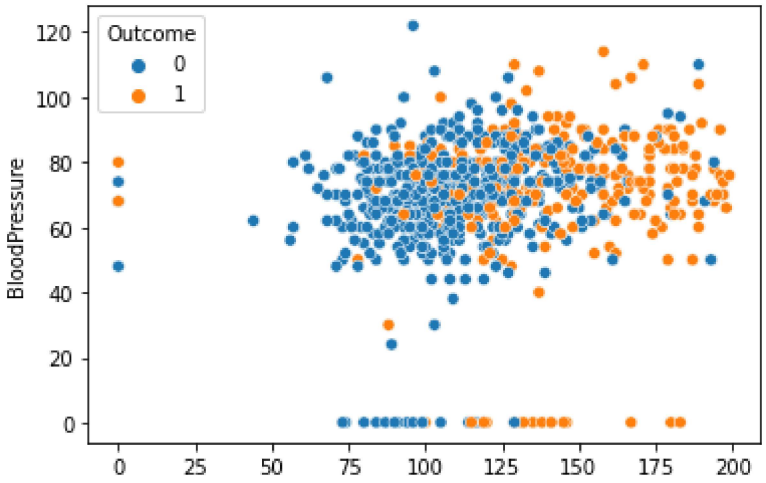

```
0    500
1    268
Name: Outcome, dtype: int64
```

Dataset is in ratio 2:1 and can be treted as balanced

```
#Scatter Plot
```

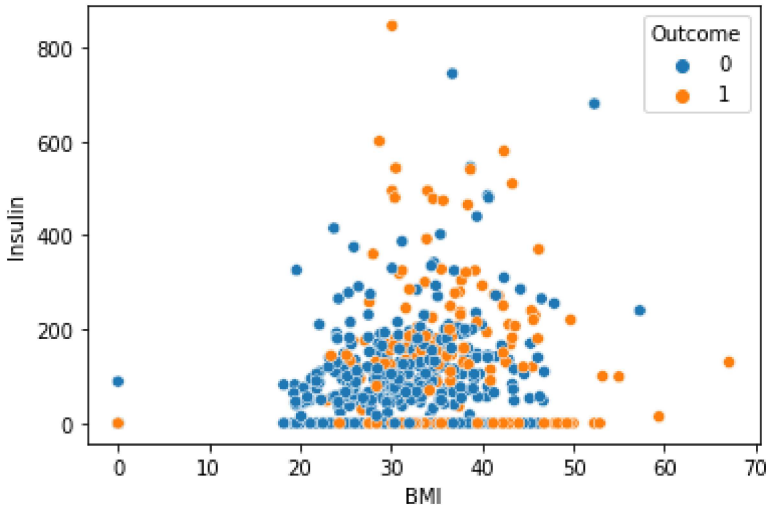
```
sns.scatterplot(x=new_data.Glucose, y=new_data.BloodPressure, hue=data.Outcome)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57320f5490>
```




```
sns.scatterplot(x=new_data.BMI, y=new_data.Insulin, hue=new_data.Outcome)
```

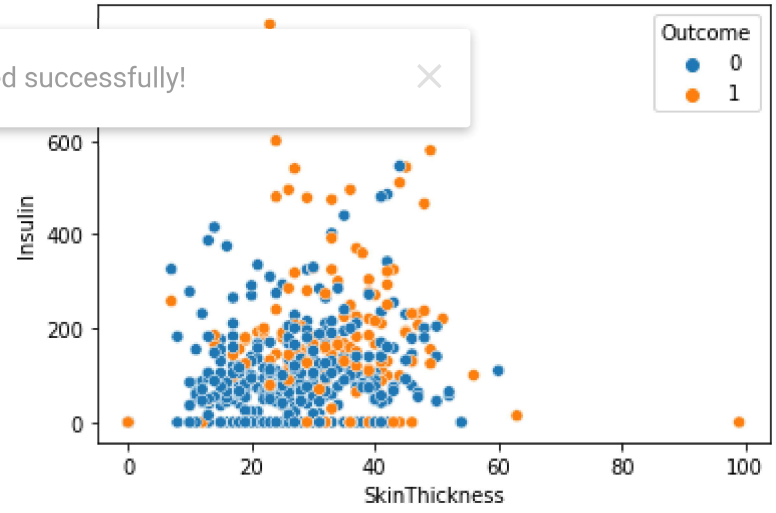
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f573208b3d0>
```




```
sns.scatterplot(x=new_data.SkinThickness, y=new_data.Insulin, hue=new_data.Outcome)
```

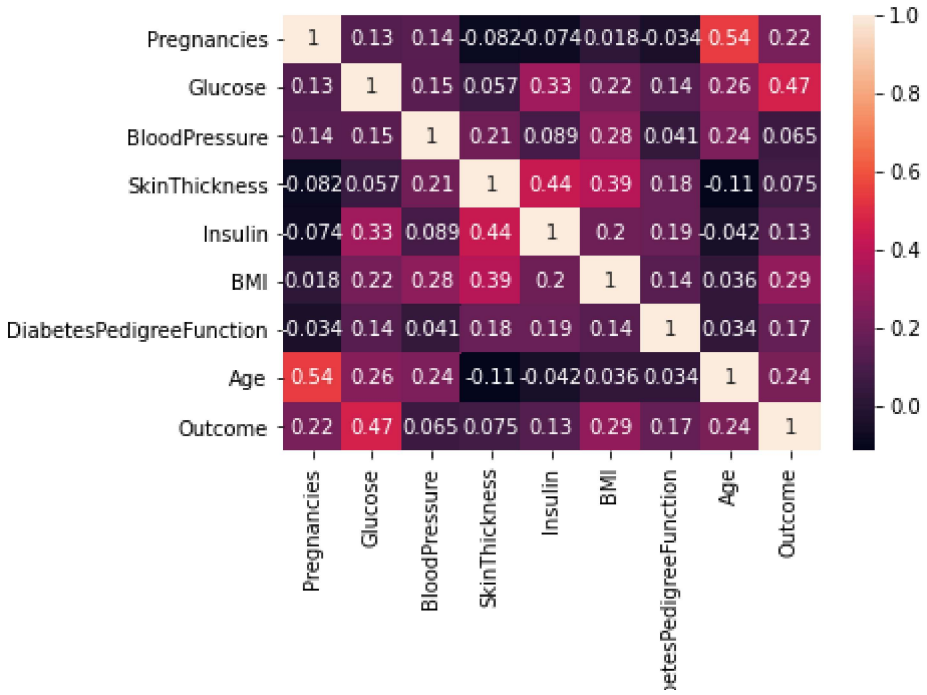
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5732013490>
```

Saved successfully!




```
sns.heatmap(data.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5731f79590>
```




```
#After alot of thinking I have chosen to to use smart correlation from feature engine to remove correlate
```

Outcome is good coorelated with Glucose

Age and preg are highly correlated

Insulin and skin_thickness is correlated

Bmi and skin_thickness is good_correlated

Week 3

```
new_data.columns
```

Saved successfully!

```
sPedigreeFunction', 'Age', 'Outcome', 'Glucose',
       'BloodPressure', 'SkinThickness', 'Insulin', 'BMI'],
      dtype='object')
```

```
feauture_col=['Pregnancies', 'DiabetesPedigreeFunction', 'Age', 'Glucose',
       'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
```

Before going further let's split data to validate our result

```
from sklearn.model_selection import train_test_split

X = new_data[feauture_col]
y = new_data['Outcome']
```

```
# For the larger the dataset, the smaller the test
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.10, random_state=42)
```

Stanadrdization

```
from sklearn.preprocessing import RobustScaler
from feature_engine.wrappers import SklearnTransformerWrapper
scaler = SklearnTransformerWrapper(transformer = RobustScaler())

scaler.fit_transform(X_train)
scaler.transform(X_val)
```

	Pregnancies	DiabetesPedigreeFunction	Age	Glucose	BloodPressure	SkinTh
668	0.6	0.131579	0.8750	-0.463415	-0.777778	
324	-0.2	-0.610526	-0.5000	-0.121951	0.166667	
624	-0.2	-0.584211	-0.5000	-0.219512	-0.444444	·
690	1.0	1.252632	0.3125	-0.243902	0.444444	·
473	0.8	-0.447368	1.3125	0.463415	1.000000	·
...	...	...	...	...	...	
512	1.2	-0.473684	1.8125	-0.634146	-0.222222	·
109	-0.6	-0.350000	-0.3125	-0.536585	0.722222	
587	0.6	-0.344737	0.0000	-0.341463	-0.333333	·
362	0.4	-0.197368	2.2500	-0.341463	2.000000	
734	-0.2	0.473684	1.5000	-0.292683	0.166667	·

77 rows × 8 columns

Saved successfully!

ncrease precicision of calculating **minority as thats more imp** I have choosen smote for this

```
from imblearn.under_sampling import RandomUnderSampler
undersample = RandomUnderSampler(sampling_strategy='majority')
X_train_under, y_train_under = undersample.fit_resample(X_train, y_train)
```

Week 4

Data modeling

```
#importing comparision
from sklearn.metrics import classification_report
```

```
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression()
model_lr.fit(X_train_under,y_train_under)
```

```
LogisticRegression()
```

```
print(model_lr.score(X_train,y_train))
print(model_lr.score(X_val,y_val))
```

```
0.7655571635311144
0.6753246753246753
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y,model_lr.predict(X))
cm
```

```
array([[382, 118],
       [ 69, 199]])
```

**Our main objective was to predict more ways a person can get diabetic and we could predict 198 out of 268 **

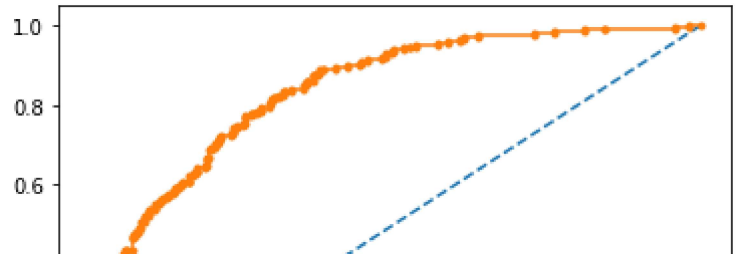
```
#Preparing ROC Curve (Receiver Operating Characteristics Curve)
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict probabilities
```

Saved successfully!

```
come only
# calculate AUC
auc = roc_auc_score(y, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y, probs)
# plot
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
```

```
AUC: 0.836
[<matplotlib.lines.Line2D at 0x7f5714e39450>]
```




```
y_pred=model_lr.predict(X_val)
```

```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.62      0.71        50
           1       0.53      0.78      0.63        27

    accuracy                           0.68        77
   macro avg       0.68      0.70      0.67        77
weighted avg       0.73      0.68      0.68        77
```

```
#Applying Decission Tree Classifier
from sklearn.tree import DecisionTreeClassifier
model_dt=DecisionTreeClassifier()
model_dt.fit(X_train_under,y_train_under)
```

```
DecisionTreeClassifier()
```

```
y_pred=model_dt.predict(X_val)
```

```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.76      0.78        50
           1       0.63      0.63      0.61        27

    accuracy                           0.71        77
   macro avg       0.69      0.69      0.69        77
weighted avg       0.72      0.71      0.72        77
```

Saved successfully!

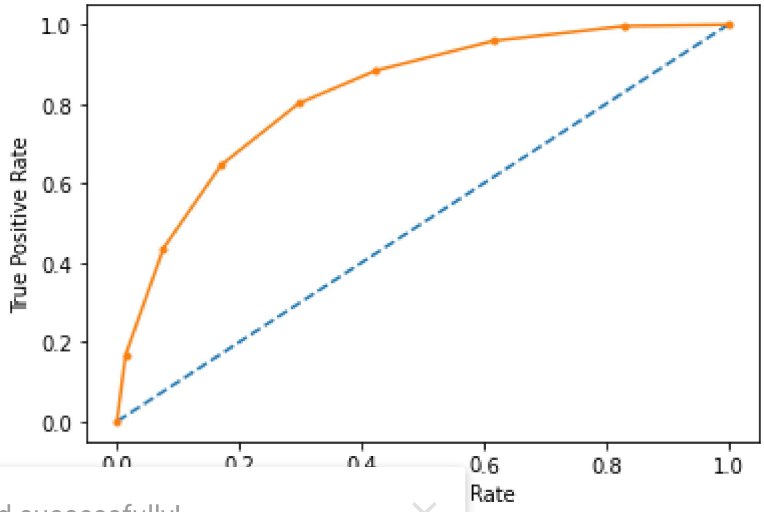
```
#Applying K-NN
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=7,
                              metric='minkowski',
                              p = 2)
model2.fit(X_train_under,y_train_under)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

```
#Preparing ROC Curve (Receiver Operating Characteristics Curve)
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# predict probabilities
probs = model2.predict_proba(X)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y, probs)
print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr,fpr,thresholds))
# plot
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
AUC: 0.826
True Positive Rate - [0.         0.16791045 0.43656716 0.64552239 0.80223881 0.88432
 0.95895522 0.99626866 1.        ], False Positive Rate - [0.    0.014 0.076 0.17  0
 0.28571429 0.14285714 0.        ]
Text(0, 0.5, 'True Positive Rate')
```



Saved successfully!

```
train_data=pd.concat([X_train_under,y_train_under],axis=1)
```

```
train_data.head()
```

	Pregnancies	DiabetesPedigreeFunction	Age	Glucose	BloodPressure	SkinThickness
0	2	1.101	24	128	64	42
2	2	0.246	28	112	86	41

To compare multiple classification algos I am using pycaret

```
#!pip install pycaret
```

```
from pycaret.classification import *
```

```
clf = setup(data, target = "Outcome",
            silent = True, session_id = 786)
```

Saved successfully!

1	Target	Outcome
2	Target Type	Binary
3	Label Encoded	0: 0, 1: 1
4	Original Data	(768, 9)
5	Missing Values	False
6	Numeric Features	7
7	Categorical Features	1
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(537, 24)
12	Transformed Test Set	(231, 24)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	74b1
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	mputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None
33	PCA	False
34	PCA Method	None

Saved successfully!

```
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7579	0.8181	0.5655	0.6978	0.6222	0.4478	0.4548	0.507
lda	Linear Discriminant Analysis	0.7543	0.8148	0.5550	0.6948	0.6147	0.4383	0.4458	0.020
rf	Random Forest Classifier	0.7525	0.8175	0.5700	0.6800	0.6172	0.4376	0.4429	0.543
ridge	Ridge Classifier	0.7506	0.0000	0.5392	0.6948	0.6051	0.4275	0.4362	0.022
gbc	Gradient Boosting Classifier	0.7432	0.8207	0.5918	0.6707	0.6225	0.4298	0.4369	0.127
lightgbm	Light Gradient Boosting Machine	0.7394	0.8253	0.6074	0.6506	0.6245	0.4259	0.4295	0.100
et	Extra Trees Classifier	0.7377	0.7941	0.5187	0.6870	0.5845	0.3986	0.4119	0.463
ada	Ada Boost Classifier	0.7338	0.7888	0.5863	0.6406	0.6087	0.4083	0.4120	0.112

Best model is:LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=1000, multi_class='auto', n_jobs=None, penalty='l2', random_state=786, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False)

56 Interaction Threshold None

```
final_model=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                               l1_ratio=None, max_iter=1000,
                               jobs=None, penalty='l2',
                               er='lbfgs', tol=0.0001, verbose=0,
                               warm_start=False)
```

Saved successfully!

```
final_model.fit(X_train_under,y_train_under)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=786, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
cm = confusion_matrix(y,model_lr.predict(X))
cm
```

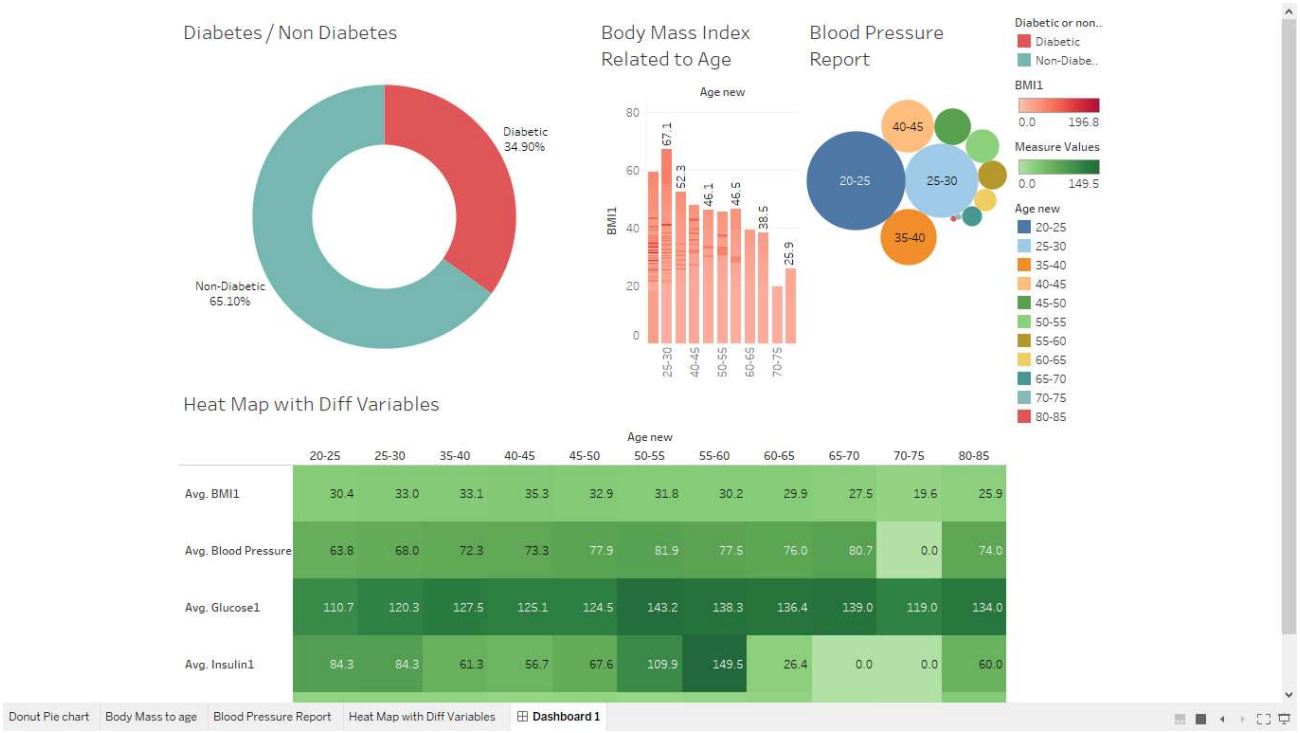
```
array([[382, 118],
       [ 69, 199]])
```

Data reporting

We could predict 199 diabetic patients

```
path=
```

```
from IPython.display import Image
Image("/content/Tableau dashbord.PNG")
```



Saved successfully!