Problem Statement :-

Build a model to accurately predict whether the patients in the dataset have diabetes or not?

Data 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 so on.

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age (years)

Outcome: Class variable (0 or 1) 268 of 768 are 1, the others are 0

In [170…
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Importing the data

In [171…
```
data=pd.read_csv('/content/health care diabetes.csv')
```

In [172…
```
data.shape
```

Out[172…
```
(768, 9)
```

In [173…
```
data.columns
```

Out[173…
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [174…
```
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
```

In [175…
```
data['Outcome'].unique()
```

Out[175…
```
array([1, 0])
```

In [176…
```
# Last column of dataset i..e, Outcome is categorical because it has only two
# unique items that are 0(not have diabetes) and 1(have diabetes) so I am changing int64 to category
```

In [177…
```
data['Outcome']=data['Outcome'].astype('category')
```

In [178…
```
data.info() # now Outcome is changes to category
```

```
<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    category
dtypes: category(1), float64(2), int64(6)
memory usage: 49.0 KB
```

Checking null values in dataset

In [179…
```
data.isnull().sum()
```

Out[179…
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [180…
```
# By this we clearly known that there are no null values in our dataset
# But in some columns we have Zero values means missing values
# Why because see in Glucose,BloodPressure,SkinThickness,Insulin,BMI zero means
# As per domain knowledge Zero not makes sense so these Zero's are missing values
```

In [181…
```
lst = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
```

In [182…
```
for i in lst:
  print(i, (data[i]==0).sum())
```

```
Glucose 5
BloodPressure 35
SkinThickness 227
Insulin 374
BMI 11
```

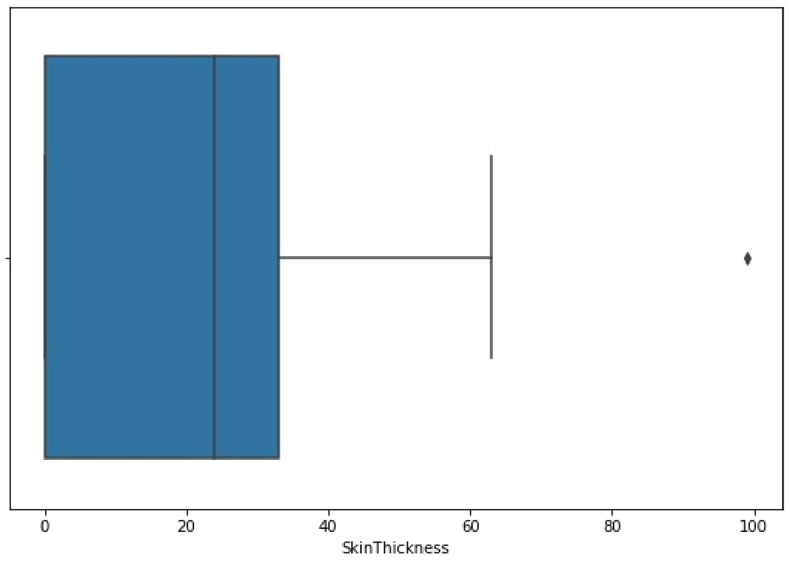
In [183…
```
# By this we can see that missing values in Glucose,BloodPressure and BMI are negligible
# But in SkinThickness and Insulin we have more missing values
# I am not removing these missing values because our dataset is small dataset if we remove these
# data points data becomes not sufficient so I am filing Zero's with Mean or Median
```

In [184…
```
# And Glucose=5, BloodPressure=35 and BMI=11 these have small missing values
# so I am removing these rows with missing values
```

In [185…
```
data2 = data[(data["Glucose"] >0) & (data["BloodPressure"] >0) & (data['BMI']>0)]
```

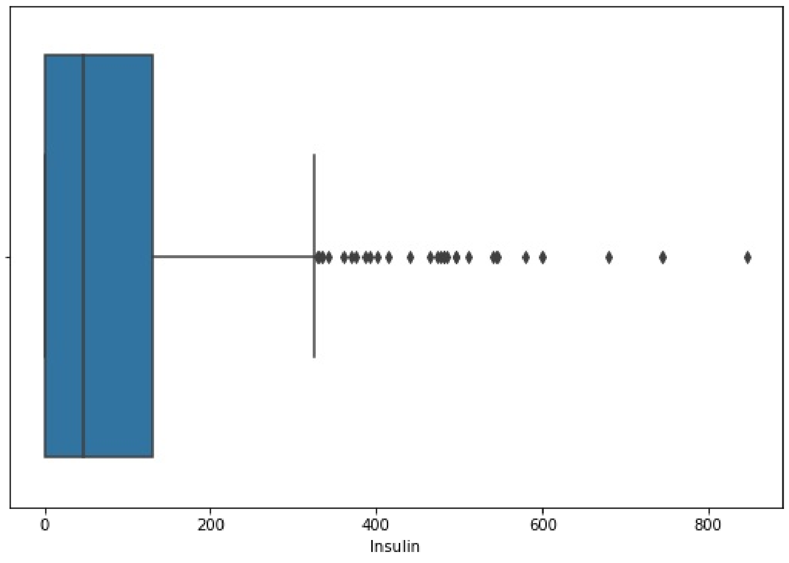
In [186…
```
# First see outliers in these (SkinThickness and Insulin) two columns and decide which is best to fill
```

In [187…
```
plt.figure(figsize=(9,6))
sns.boxplot(data2['SkinThickness'])
plt.show()
```



It has less outliers so fill Zero's with mean

```
In [188… plt.figure(figsize=(9,6))
sns.boxplot(data2['Insulin'])
plt.show()
```



It has more outliers so fill Zero's with Median Because mean is affected by outliers and Median is not affected by outliers

```
In [189… data2['SkinThickness'].mean(), data2['Insulin'].median()
```

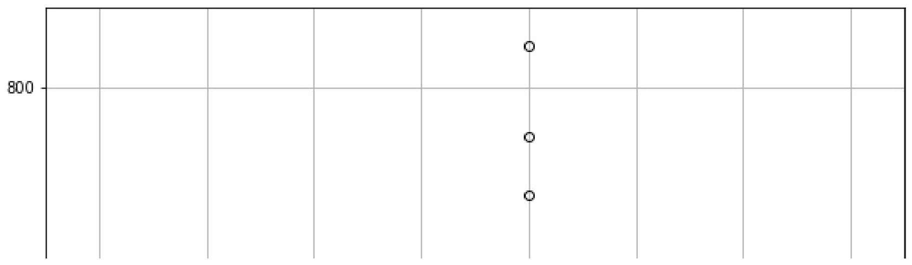
```
Out[189… (21.443370165745858, 48.0)
```

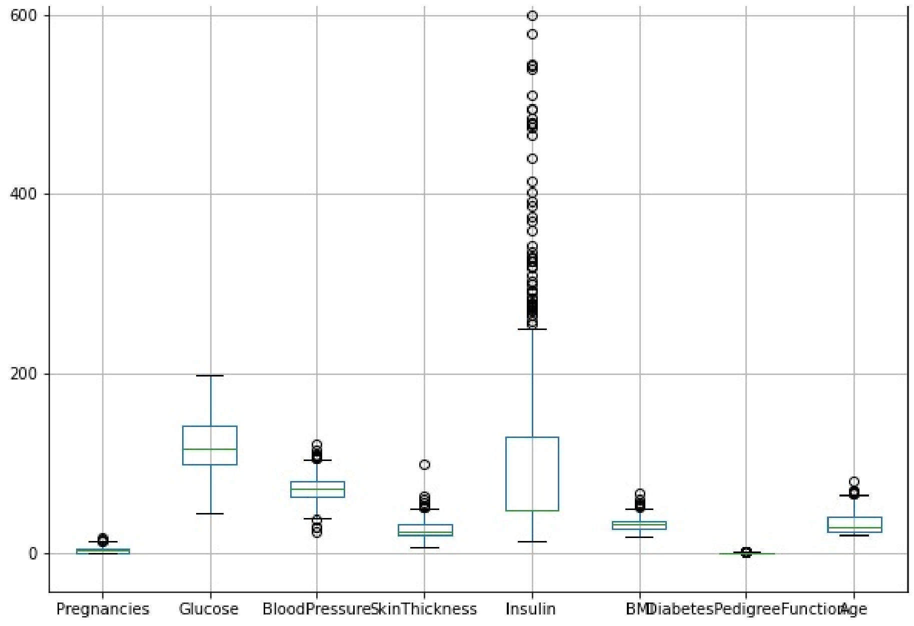
```
In [190… data2['SkinThickness']=data2['SkinThickness'].replace(0,21.443370165745858)
data2['Insulin']=data2['Insulin'].replace(0,48.0)
```

Detecting and Treating of outliers in datase

```
In [191… data2.boxplot(figsize=(10, 10))
```

```
Out[191… <matplotlib.axes._subplots.AxesSubplot at 0x7fe47234ab50>
```

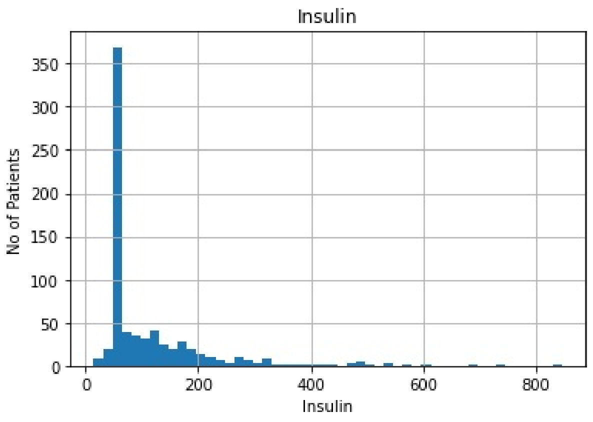




In [192…
```
# By above boxplot Insulin have more outliers so we have to treat these outliers
# Removing outliers is not an option to our dataset because our dataset is small in size
```

In [193…
```
figure=data2['Insulin'].hist(bins=50)
figure.set_title('Insulin')
figure.set_xlabel('Insulin')
figure.set_ylabel('No of Patients')
```

Out[193…
```
Text(0, 0.5, 'No of Patients')
```



In [194…
```
# Here the Data is not Normally distributed and is following the Right Skewed distribution
# that means we can use the Interquartile Range to measure the boundaries for outliers
# IQR = Q3 - Q1 (quartile 3 - quartile 1)
```

In [195…
```
IQR= data2['Insulin'].quantile(0.75) - data2['Insulin'].quantile(0.25)
IQR
```

Out[195…
```
82.5
```

In [196…
```
## Calculating the boundaries
lower_bridge= data2['Insulin'].quantile(0.25)-(IQR*1.5)
upper_bridge= data2['Insulin'].quantile(0.75)+(IQR*1.5)
print(lower_bridge), print(upper_bridge)
```

```
-75.75
254.25
```

Out[196…
```
(None, None)
```

In [197…
```
data2['Insulin'].describe()
```

Out[197…
```
count    724.000000
mean     106.505525
std      102.669035
min       14.000000
25%       48.000000
50%       48.000000
75%      130.500000
max      846.000000
Name: Insulin, dtype: float64
```

In [198…
```
# Here the maximum value of outliers is very high compare to upper boundary that indicates
# we need to calculate the extreme outliers boundaries
```

In [199…
```
## Calculating the extreme boundaries
lower_bridge= data2['Insulin'].quantile(0.25)-(IQR*3)
upper_bridge= data2['Insulin'].quantile(0.75)+(IQR*3)
print(lower_bridge), print(upper_bridge)
```

```
-199.5
378.0
```

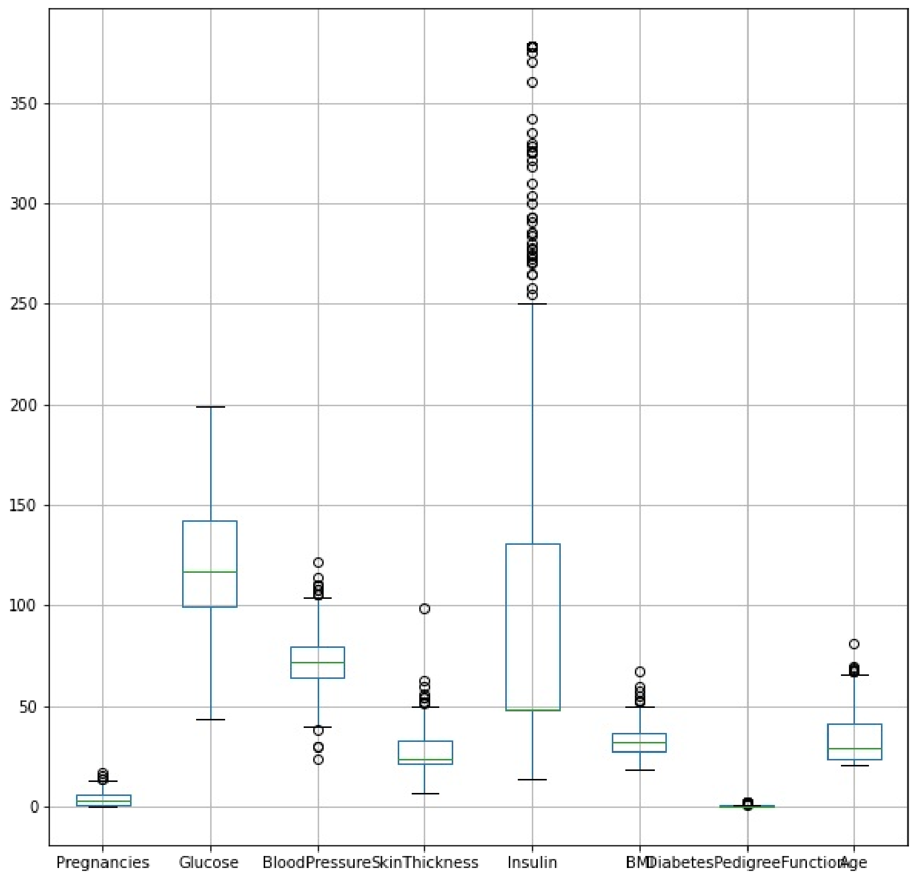
Out[199…
```
(None, None)
```

In [200…
```
# Replacing the outliers
# Since the lower boundaries of Insulin column is negative value
# we do not need to consider the lower boundary because as per the domain knowledge
# there won't be any negative values exists for Insulin Column
# Insulin upper bridge is 378
```

In [201…
```
data2.loc[data2['Insulin']>=378,'Insulin']=378
```

In [202…
```
data2.boxplot(figsize=(10, 10))
```

Out[202…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe472124710>
```



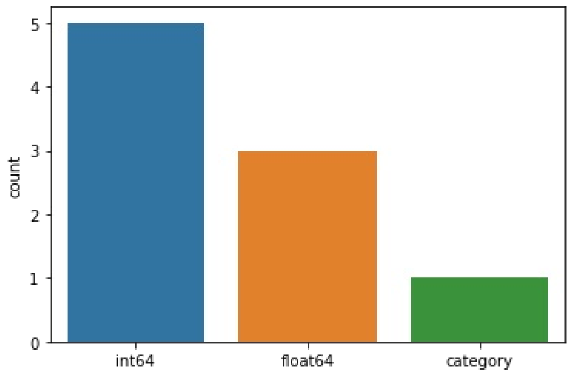
In [203…
```
# By comparing above boxplot outliers are only reduced not completely gone
```

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

In [204…
```
data2.dtypes.value_counts()
```

Out[204…
```
int64       5
float64     3
category    1
dtype: int64
```

In [205…
```
sns.countplot(data2.dtypes.map(str))
plt.show()
```

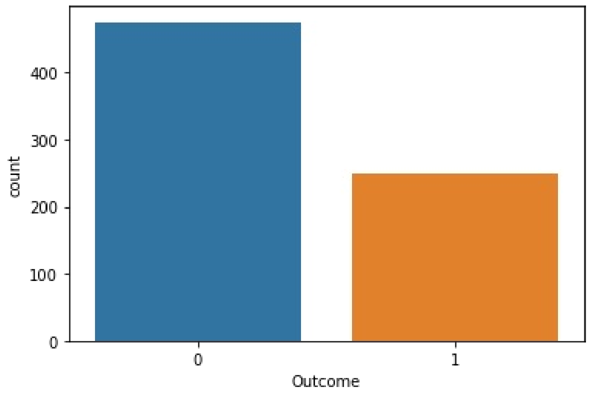


Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of actions.

In [206…
```
data2['Outcome'].value_counts()
```

Out[206…
```
0    475
1    249
Name: Outcome, dtype: int64
```

In [207…
```
sns.countplot(data2['Outcome'].map(str))
plt.show()
```



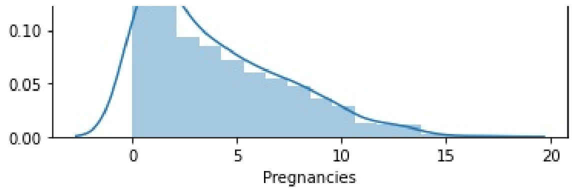
In [208…
```
# By above we have clear evidence that our dataset has class imbalance issue
# There are different methods to solve class imbalace issue
# 1.Over Sampling Technique
# 2.Under sampling Technique
# 3.Cross validation Technique
# But see in both under and over samplung we are adding and deleting data points.
# High Accuracy will come but we cannot assure that that accuracy is biased or not
# But with cross validation in this techinque we train our model using the subset of the data-set
# so we can assure that accuracy got by this technique is unbiaed.
```

Checking variables normal or not if not normal making to normal

In [209…
```
sns.distplot(data2['Pregnancies'])
```

Out[209…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe471f03f50>
```

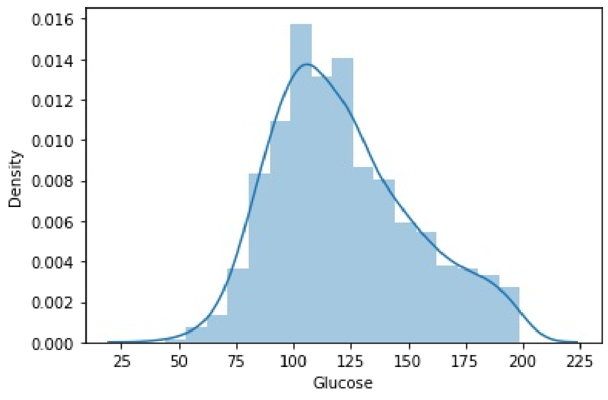
Density
0.15
0.20
0.25
0.30



It is not normal it is right skewed

In [210…
```
sns.distplot(data2['Glucose'])
```

Out[210…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe471ee2dd0>
```



it is visually normal but slight skewed so we can declare it is skewed or not by summary statistics

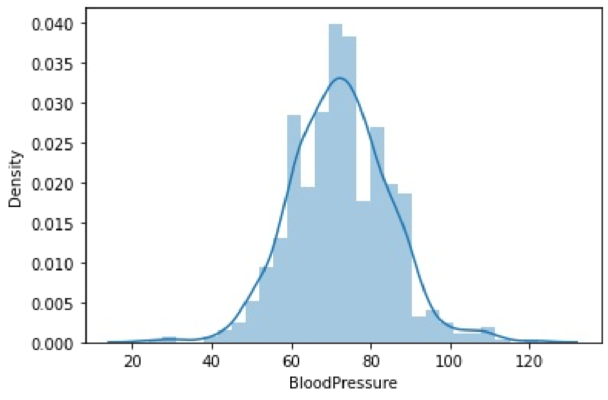
In [211…
```
data2['Glucose'].describe()
```

Out[211…
```
count    724.000000
mean     121.882597
std       30.750030
min       44.000000
25%       99.750000
50%      117.000000
75%      142.000000
max      199.000000
Name: Glucose, dtype: float64
```

Here there is 4 difference between mean and median so it is not normal

In [212…
```
sns.distplot(data2['BloodPressure'])
```

Out[212…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe471dfadd0>
```



In [213…
```
data2['BloodPressure'].describe()
```

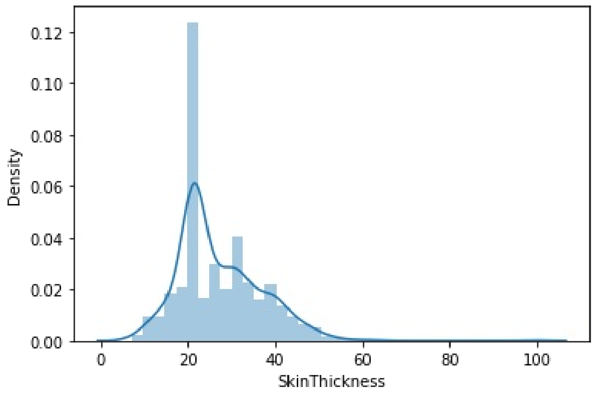
Out[213…
```
count    724.000000
mean      72.400552
std       12.379870
```

```
min        24.000000
25%        64.000000
50%        72.000000
75%        80.000000
max       122.000000
Name: BloodPressure, dtype: float64
```

It is normal

In [214…
```
sns.distplot(data2['SkinThickness'])
```

Out[214…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe471d08e10>
```



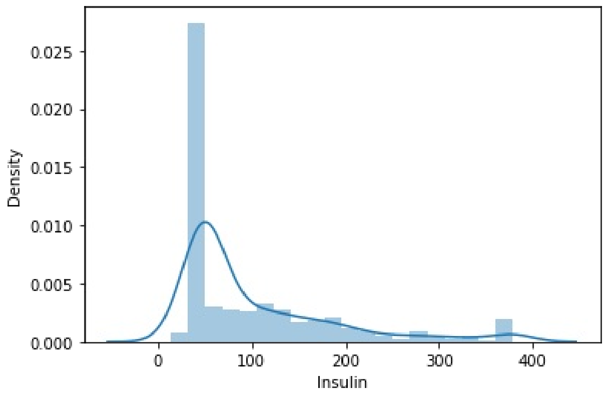
In [215…
```
data2['SkinThickness'].describe()
```

Out[215…
```
count     724.000000
mean       27.130010
std         9.645083
min         7.000000
25%        21.443370
50%        24.000000
75%        33.000000
max        99.000000
Name: SkinThickness, dtype: float64
```

Not normal right skewed

In [216…
```
sns.distplot(data2['Insulin'])
```

Out[216…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe471c43090>
```



In [217…
```
data2['Insulin'].describe()
```

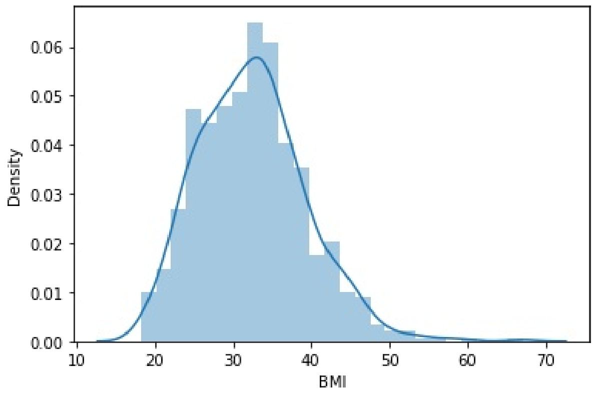
Out[217…
```
count     724.000000
mean      102.142265
std        84.536572
min        14.000000
25%        48.000000
50%        48.000000
75%       130.500000
```

```
max      378.000000
Name: Insulin, dtype: float64
```

Not normal and right skewed

```
In [218…  sns.distplot(data2['BMI'])
```

```
Out[218…  <matplotlib.axes._subplots.AxesSubplot at 0x7fe471b75190>
```



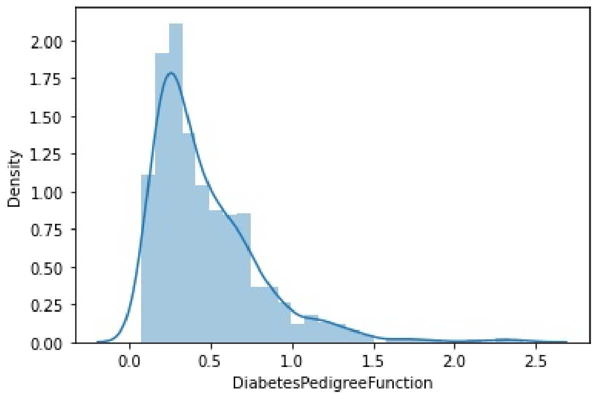
```
In [219…  data2['BMI'].describe()
```

```
Out[219…  count    724.000000
          mean      32.467127
          std        6.888941
          min       18.200000
          25%       27.500000
          50%       32.400000
          75%       36.600000
          max       67.100000
          Name: BMI, dtype: float64
```

It is normal distribution

```
In [220…  sns.distplot(data2['DiabetesPedigreeFunction'])
```

```
Out[220…  <matplotlib.axes._subplots.AxesSubplot at 0x7fe471b10350>
```



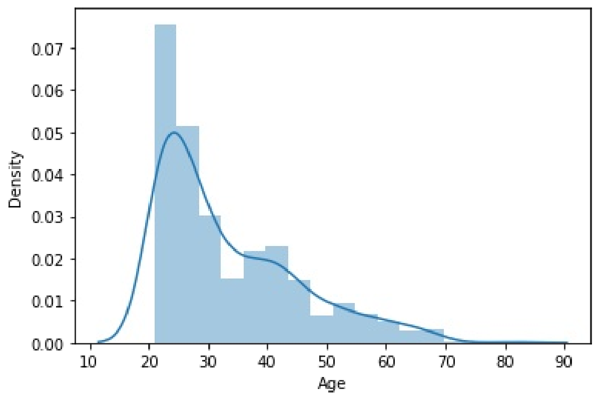
```
In [221…  data2['DiabetesPedigreeFunction'].describe()
```

```
Out[221…  count    724.000000
          mean       0.474765
          std        0.332315
          min        0.078000
          25%        0.245000
          50%        0.379000
          75%        0.627500
          max        2.420000
          Name: DiabetesPedigreeFunction, dtype: float64
```

It is normal difference between mean and median is negligble

```
In [222… sns.distplot(data2['Age'])
```

```
Out[222… <matplotlib.axes._subplots.AxesSubplot at 0x7fe471a38990>
```



```
In [223… data2['Age'].describe()
```

```
Out[223… count    724.000000
mean      33.350829
std       11.765393
min       21.000000
25%       24.000000
50%       29.000000
75%       41.000000
max       81.000000
Name: Age, dtype: float64
```

It is not normal

So here Pregnancies,Glucose,SkinThickness,Insulin,Age are not normal we have to make it to normal

```
In [224… lst1 = ['Pregnancies','Glucose','SkinThickness','Insulin','Age']
```

```
In [225… for i in lst1:
           data2[i] = np.log1p(data2[i])
```

```
In [226… data2.describe()
```

Out[226…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000	724.000000
mean	1.319311	4.780277	72.400552	3.280856	4.387084	32.467127	0.474765	3.484730
std	0.762929	0.249781	12.379870	0.336677	0.662871	6.888941	0.332315	0.313971
min	0.000000	3.806662	24.000000	2.079442	2.708050	18.200000	0.078000	3.091042
25%	0.693147	4.612633	64.000000	3.110995	3.891820	27.500000	0.245000	3.218876
50%	1.386294	4.770685	72.000000	3.218876	3.891820	32.400000	0.379000	3.401197
75%	1.945910	4.962845	80.000000	3.526361	4.878985	36.600000	0.627500	3.737670
max	2.890372	5.298317	122.000000	4.605170	5.937536	67.100000	2.420000	4.406719

Checking correlation between variables

First Doing Scaling to all dataset

```
In [227… from sklearn.preprocessing import MinMaxScaler,StandardScaler
         m=MinMaxScaler()
```

Here x is independent variable and y is dependent variable (Outcome)

```
In [228… x=data2.drop(['Outcome'],axis=1)
         y=data2['Outcome']
```

In [229…

```python
data_sc=m.fit_transform(x)
```

In [230…

```python
data_sc_df=pd.DataFrame(data_sc,columns=x.columns,index=x.index)
data_sc_df.head(5)
```

Out[230…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.673239	0.802655	0.489796	0.595502	0.366551	0.314928	0.234415	0.639050
1	0.239812	0.434206	0.428571	0.523317	0.366551	0.171779	0.116567	0.284791
2	0.760188	0.944101	0.408163	0.408418	0.366551	0.104294	0.253629	0.308180
3	0.239812	0.464683	0.428571	0.434968	0.571554	0.202454	0.038002	0.000000
4	0.000000	0.751240	0.163265	0.595502	0.749918	0.509202	0.943638	0.330870

Finding correlation between variables

In [231…

```python
data_sc_df.corr()
```

Out[231…

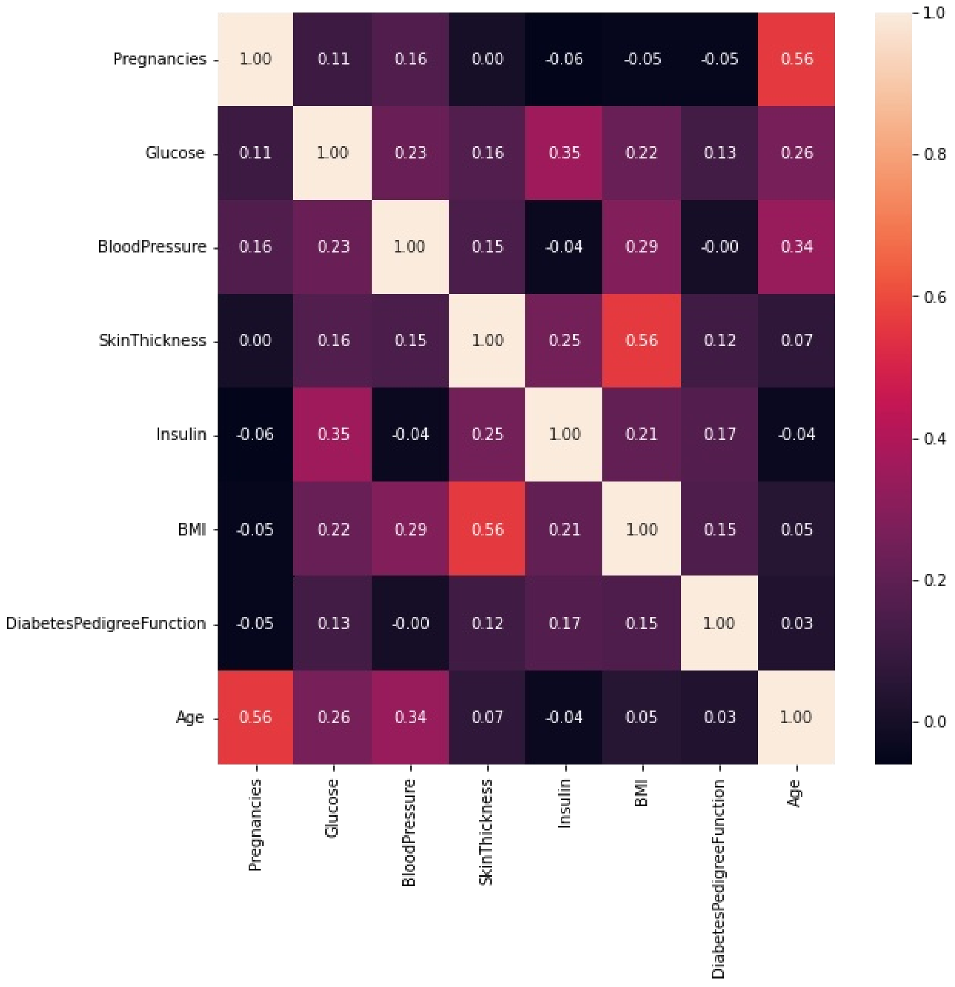
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.000000	0.110236	0.159556	0.000653	-0.061777	-0.046226	-0.046698	0.560507
Glucose	0.110236	1.000000	0.226538	0.164075	0.354076	0.220961	0.125562	0.257609
BloodPressure	0.159556	0.226538	1.000000	0.147366	-0.035981	0.287403	-0.000075	0.340852
SkinThickness	0.000653	0.164075	0.147366	1.000000	0.249853	0.561603	0.119397	0.069261
Insulin	-0.061777	0.354076	-0.035981	0.249853	1.000000	0.205610	0.167646	-0.035048
BMI	-0.046226	0.220961	0.287403	0.561603	0.205610	1.000000	0.154858	0.049484
DiabetesPedigreeFunction	-0.046698	0.125562	-0.000075	0.119397	0.167646	0.154858	1.000000	0.032301
Age	0.560507	0.257609	0.340852	0.069261	-0.035048	0.049484	0.032301	1.000000

In [232…

```python
plt.figure(figsize = (9,9))
sns.heatmap(data_sc_df.corr(),annot=True,fmt='.2f')
```

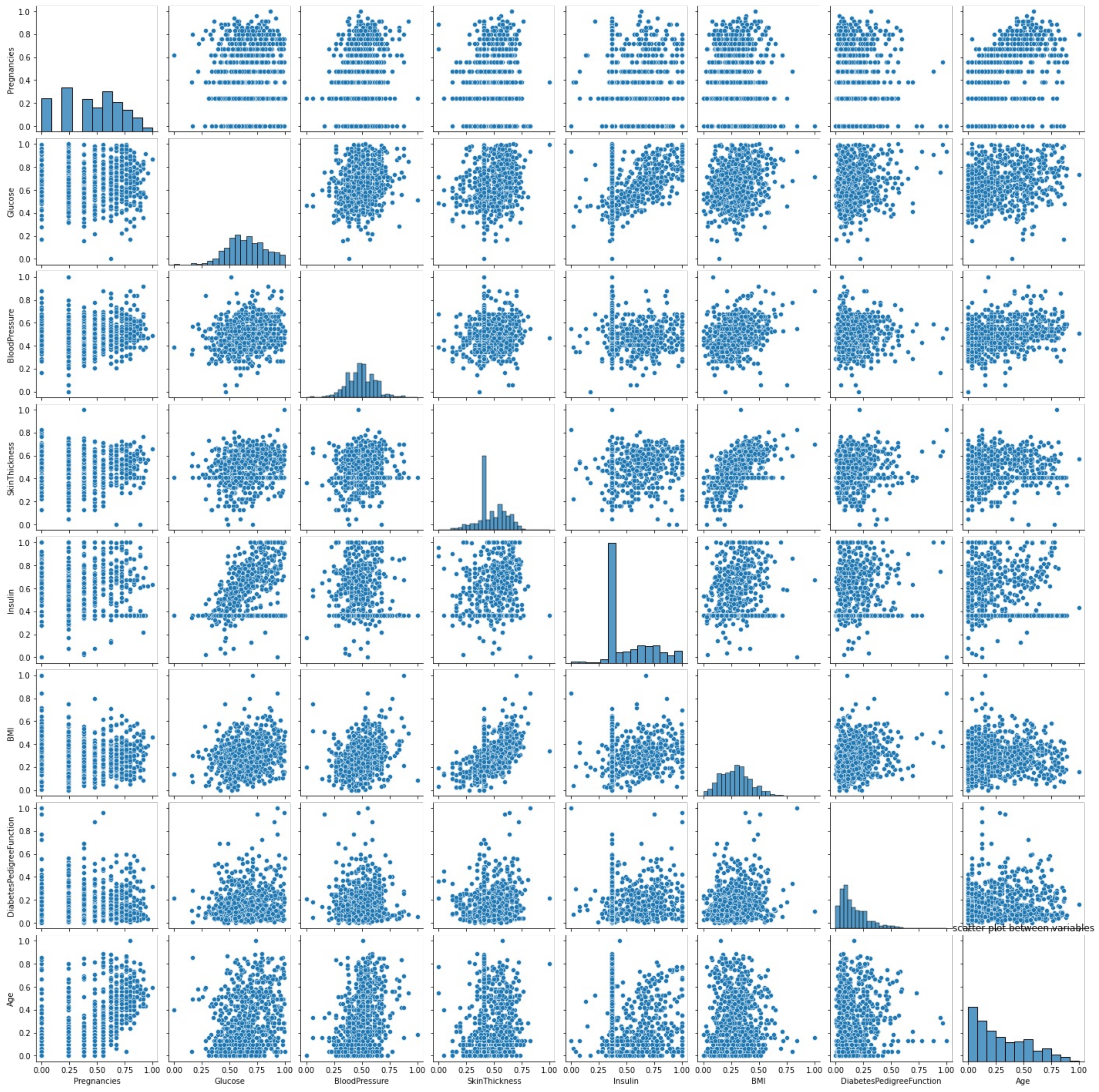
Out[232…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe4719c6f10>
```



In [233…
```
sns.pairplot(data_sc_df)
plt.title('scatter plot between variables')
```

Out[233…
```
Text(0.5, 1.0, 'scatter plot between variables')
```



In [234…
```
# Clearly muli-collinearity exists in variables in our data set
# 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 positive correlation.
# In heat map we got cleared there is multi-collinearity
```

In [235…
```
# For Logistic Regression multi-collinearity should not be in data
# But here there is clearly multi-collinearity exists between variables
# Multi-collinearity means presence of relation between independent variables
# Multi-collinearity is a problem we should avoid
# There are two types o avoid multi-collinearity
# 1.) Remove independent variables which are corelated
# 2.) Use dimensionality reduction technique to merge variables
```

In [236…
```
# Variance Inflation Factor (VIF)
# equal to the ratio of the overall model variance to the variance of a model
# that includes only that single independent variable.
# VIF=1/1-R square (R-square is correlation coefficient of that variable)
```

In [237…
```
# VIF ~ 1: Negligible
# 1<VIF<5 : Moderate
# VIF>5 : Extreme
# if a variable with extreme VIF is should be removed
```

In [238…
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [239…
```
# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = data_sc_df.columns
```

In [240…
```
# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(data_sc_df.values, i)
for i in range(len(data_sc_df.columns))]
```

In [241…
```
print(vif_data)
```

```
                    feature        VIF
0               Pregnancies   5.512882
1                   Glucose  18.220714
2             BloodPressure  15.132647
3             SkinThickness  17.090657
4                   Insulin   9.077077
5                       BMI   8.266541
6  DiabetesPedigreeFunction   2.534800
7                       Age   4.104236
```

In [242…
```
# Here five variables have high VIF value so we have to remove these variables
# That leads heavy data loss in our scenario so by using PCA echnique.
# we are going to merge there all variables.
```

PCA Technique

In [243…
```
# using PCA technique
from sklearn.decomposition import PCA
```

In [244…
```
pca=PCA(n_components=6)
var_x=pca.fit_transform(data_sc_df)
```

In [245…
```
new_data=pd.DataFrame(var_x,columns=['pca1','pca2','pca3','pca4','pca5','pca6'])
```

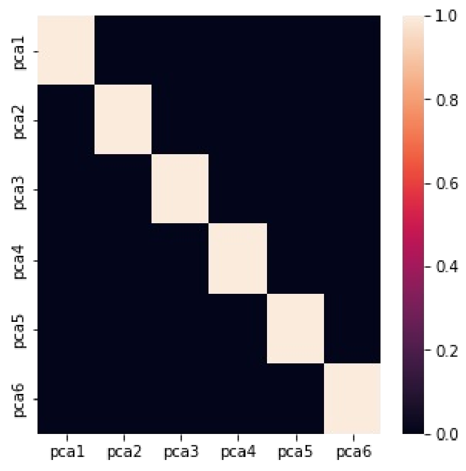
In [246…
```
b=pca.explained_variance_ratio_
b*100
```

Out[246…
```
array([37.81312702, 20.79637505, 11.53703291,  8.99408893,  7.07498587,
        6.66814181])
```

Here 88% data captured by these pca variables

In [247…
```
plt.figure(figsize = (5,5))
sns.heatmap(new_data.corr())
```

Out[247…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe46fdc9a90>
```



so finally no multicollinearity so we can proceed with logistic regression

Logistic Regression Model Building

Logistic Regression Model Building

In [248…
```
# so now
x_var=new_data
y_var=y
```

In [249…
```
# splitting our data
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
```

In [250…
```
x_train,x_test,y_train,y_test=train_test_split(x_var,y_var,test_size=0.2,random_state=50)
```

In [251…
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
```

Out[251…
```
LogisticRegression()
```

In [252…
```
pred=lr.predict(x_test)
```

In [253…
```
accuracy_score(y_test,pred)
```

Out[253…
```
0.7793103448275862
```

In [254…
```
accuracy_score(y_train,lr.predict(x_train))
```

Out[254…
```
0.7651122625215889
```

In [255…
```
from prettytable import PrettyTable
x=PrettyTable(['Accuracy','score'])
x.add_row(['y_test and y_test predicted',0.7793103448275862])
x.add_row(['y_train and y_train predicted',0.7651122625215889])
print(x)
```

```
+-------------------------------+--------------------+
|            Accuracy           |       score        |
+-------------------------------+--------------------+
|  y_test and y_test predicted  | 0.7793103448275862 |
| y_train and y_train predicted | 0.7651122625215889 |
+-------------------------------+--------------------+
```

see clearly our model is not over fitted so see for other reports

In [256…
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.94      0.84        89
           1       0.85      0.52      0.64        56

    accuracy                           0.78       145
   macro avg       0.80      0.73      0.74       145
weighted avg       0.79      0.78      0.76       145
```

1.Accuracy is over all 78% percent it is good because our data predicts 78% correctly 2.But here our outcome is having diabetes (1) and not having diabetes (0) means both 1 and 0 are important here so we have to consider macro average because it gives equal importance to both 1 and 0 so our macro average is 74%

AUC and ROC curve

AUC Curve

The Area Under the Curve (AUC) Is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative class

ROC curve

ROC (Reciever Operating Characteristic Curve) In ROC curve we plot pairs of the true positive (Sensitivity) rate vs the false positive rate (Specificity). for every possible decision threshold of a logistic regression model. By ROC curve we can find optimal Threshold value for our model.

In [257…
```
# plot roc and auc curve
from sklearn.metrics import roc_auc_score,roc_curve
```

Here first find predicted probabilities of test data. Then created dataframe of actual and predicted probabilities of data

In [258…
```
y_prob=lr.predict_proba(x_test)
df_pred_prob=pd.DataFrame({'Actual':y_test,'Predicted_prob':y_prob[:,1]})
```

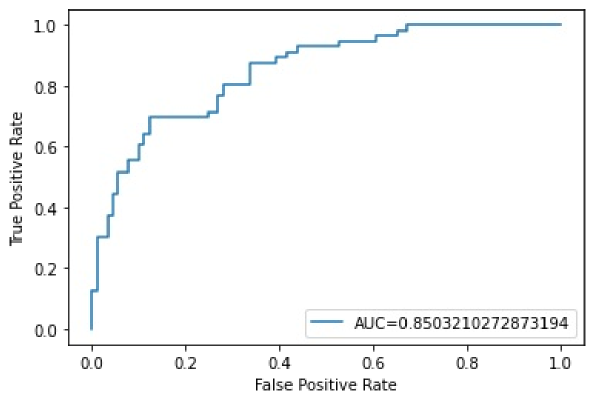
In [259…
```
df_pred_prob
```

Out[259…

	Actual	Predicted_prob
19	1	0.274795
750	1	0.396869
213	1	0.318996
424	1	0.713071
501	0	0.144898
...	...	...
647	1	0.405455
108	0	0.088859
730	1	0.315705
761	1	0.726563
45	1	0.704235

145 rows × 2 columns

In [260…
```
y_prob[:,1][0:10]
fpr,tpr,thresholds=roc_curve(y_test,y_prob[:,1])
auc_curve=roc_auc_score(y_test,y_prob[:,1])
plt.plot(fpr,tpr,label="AUC="+str(auc_curve))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Here our AUC is 85% means our model distinguished 85% correctly

In [261…
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
confusion_matrix(y_test,pred)
```

Out[261…
```
array([[84,  5],
       [27, 29]])
```

BY this confusion matrix we can say that actually 5 patients have no diabetes but our model says have diabetes. Actually 27 patients have diabetes but our predicted model says not have diabetes this is false negative here our false negative=27 we have to reduce our False Negative. we have to find threshold value to reduce our error(false negative) by ROC curve we can know that

Put threshold at 0.4 means probabilities more than 0.4 are 1 (have diabetes) rest are 0 (not have diabetes)

Put threshold at 0.4 means probabilities more than 0.4 are 1 (have diabetes) rest are 0 (not have diabetes)

In [262…
```
df_pred_prob['prediction_0.4']=np.where(df_pred_prob['Predicted_prob']>0.4,1,0)
confusion_matrix(df_pred_prob['Actual'],df_pred_prob['prediction_0.4'])
```

Out[262…
```
array([[78, 11],
       [18, 38]])
```

Here our false negative is reduced I think it's fair so I am fixing threshold value at 0.4 . By domain knowledge if we want min (least) false negative we can reduce threshold value below 0.4

put threshold at 0.2

In [263…
```
df_pred_prob['prediction_0.2']=np.where(df_pred_prob['Predicted_prob']>0.2,1,0)
confusion_matrix(df_pred_prob['Actual'],df_pred_prob['prediction_0.2'])
```

Out[263…
```
array([[46, 43],
       [ 4, 52]])
```

Here see false negative(FN) is 4 means FN is reduced but see False positive(FP) is increased

Other Classification Models Except Logistic Model

Decision Trees

For Tree models preprocessing steps are not necessary but we will use above data

In [264…
```
# Here I am using Scaling Data not PCA data
x_dtree=data_sc_df
y_dtree=y
```

In [265…
```
x_dtree_train,x_dtree_test,y_dtree_train,y_dtree_test=train_test_split(x_dtree,y_dtree,test_size=0.2,random_state
```

In [266…
```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_dtree_train,y_dtree_train)
dpred=dt.predict(x_dtree_test)
```

In [267…
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [268…
```
accuracy_score(y_dtree_test,dpred)
```

Out[268…
```
0.6689655172413793
```

In [269…
```
accuracy_score(y_dtree_train,dt.predict(x_dtree_train))
```

Out[269…
```
1.0
```

Now use PCA independent variables and check accuracy

In [270…
```
x_train_p,x_test_p,y_train_p,y_test_p=train_test_split(x_var,y_var,test_size=0.2,random_state=50)
```

In [271…
```
from sklearn.tree import DecisionTreeClassifier
pdt=DecisionTreeClassifier()
```

In [272…
```
pdt.fit(x_train_p,y_train_p)
```

Out[272…
```
DecisionTreeClassifier()
```

In [273…
```
pdpred=pdt.predict(x_test_p)
```

In [274…
```
accuracy_score(y_test_p,pdpred)
```

Out[274…
```
0.7034482758620689
```

In [275…
```
accuracy_score(y_train_p,pdt.predict(x_train_p))
```
Out[275…
```
1.0
```

There is small difference in accuracy between scaling data and PCA data (more accuracy in PCA data). So we can use PCA independent variables for further models because our PCA data is multicollinearity free and it reduces overfitting. Here our model is clearly over fitted. It is I think due to class imbalance issue. First we build all classification models and finally we see every model with cross validation technique and finalize our model based on accuracy got by cross validation technique.

Random Forest

In [276…
```
x_rforest=x_var
y_rforest=y_var
```
In [277…
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
```
In [278…
```
x_rforest_train,x_rforest_test,y_rforest_train,y_rforest_test=train_test_split(x_rforest,y_rforest,test_size=0.2,
```
In [279…
```
rf.fit(x_rforest_train,y_rforest_train)
```
Out[279…
```
RandomForestClassifier()
```
In [280…
```
rf_pred=rf.predict(x_rforest_test)
```
In [281…
```
accuracy_score(y_rforest_test,rf_pred)
```
Out[281…
```
0.7517241379310344
```
In [282…
```
accuracy_score(y_rforest_train,rf.predict(x_rforest_train))
```
Out[282…
```
1.0
```
In [283…
```
print(classification_report(y_rforest_test,rf_pred))
```
```
              precision    recall  f1-score   support

           0       0.74      0.91      0.82        89
           1       0.78      0.50      0.61        56

    accuracy                           0.75       145
   macro avg       0.76      0.71      0.71       145
weighted avg       0.76      0.75      0.74       145
```

Support Vector Machine Model

In [284…
```
x_svm=x_var
y_svm=y_var
```
In [285…
```
x_svm_train,x_svm_test,y_svm_train,y_svm_test=train_test_split(x_svm,y_svm,test_size=0.2,random_state=50)
```
In [286…
```
from sklearn.svm import SVC
```
In [287…
```
svm=SVC()
svm.fit(x_svm_train,y_svm_train)
```
Out[287…
```
SVC()
```
In [288…
```
svm_pred=svm.predict(x_svm_test)
```
In [289…
```
accuracy_score(y_svm_test,svm_pred)
```

Out[289… 0.7241379310344828

In [290…
```
accuracy_score(y_svm_train,svm.predict(x_svm_train))
```

Out[290… 0.7962003454231433

In [291…
```
print(classification_report(y_svm_test,svm_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.91      0.80        89
           1       0.75      0.43      0.55        56

    accuracy                           0.72       145
   macro avg       0.73      0.67      0.67       145
weighted avg       0.73      0.72      0.70       145
```

Naive Bayes Classification Algorithm

In [292…
```
x_train_nb,x_test_nb,y_train_nb,y_test_nb=train_test_split(data_sc_df,y_var,test_size=0.2,random_state=50)
```

In [293…
```
from sklearn.naive_bayes import MultinomialNB
nb=MultinomialNB()
```

In [294…
```
nb.fit(x_train_nb,y_train_nb)
```

Out[294… MultinomialNB()

In [295…
```
pred_nb=nb.predict(x_test_nb)
```

In [296…
```
accuracy_score(y_test_nb,pred_nb)
```

Out[296… 0.6137931034482759

In [297…
```
accuracy_score(y_train_nb,nb.predict(x_train_nb))
```

Out[297… 0.6666666666666666

In [298…
```
print(classification_report(y_test_nb,pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.61      1.00      0.76        89
           1       0.00      0.00      0.00        56

    accuracy                           0.61       145
   macro avg       0.31      0.50      0.38       145
weighted avg       0.38      0.61      0.47       145
```

K-Nearest Neighbors Algorithm

I am using PCA independent variables

In [299…
```
x_var.head(2)
```

Out[299…

	pca1	pca2	pca3	pca4	pca5	pca6
0	0.407391	0.001247	0.171459	-0.012504	0.031812	0.080258
1	-0.205940	0.217559	0.105583	0.029990	0.076450	-0.148843

In [300…
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=5)
km.fit(x_var)
```

Out[300…
```
KMeans(n_clusters=5)
```

In [301…
```
km.labels_
```

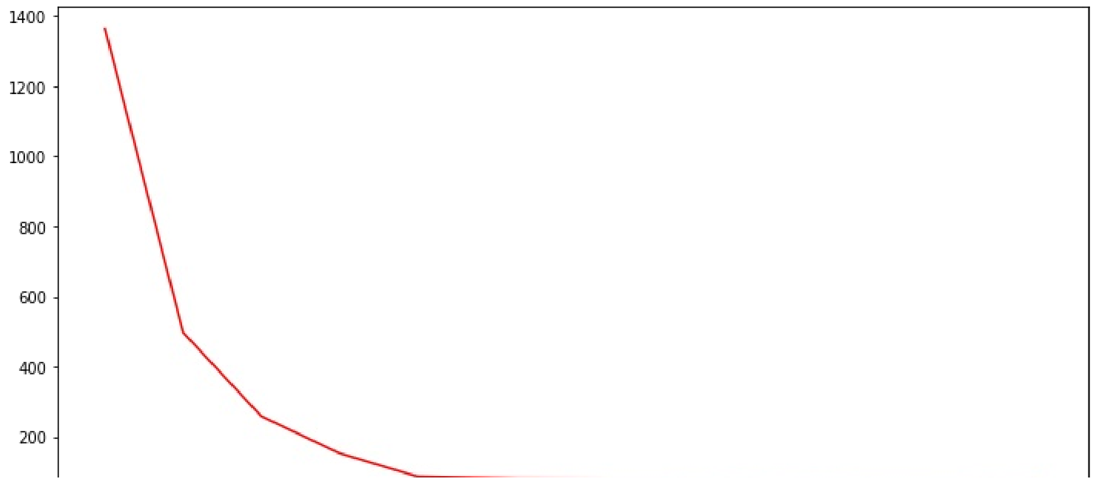
Out[301…
```
array([1, 2, 1, 2, 4, 2, 2, 3, 1, 1, 1, 3, 3, 4, 1, 0, 0, 4, 1, 1, 1, 3,
       3, 1, 2, 1, 1, 1, 4, 2, 2, 1, 3, 1, 1, 2, 3, 2, 1, 1, 3, 1, 0, 2,
       2, 1, 2, 2, 2, 3, 3, 2, 3, 0, 0, 0, 1, 2, 4, 1, 1, 0, 1, 2, 2, 2,
       4, 1, 4, 2, 1, 2, 2, 2, 1, 0, 1, 4, 1, 2, 3, 2, 2, 3, 1, 1, 2, 3,
       2, 2, 2, 4, 0, 2, 0, 2, 2, 4, 0, 3, 2, 0, 4, 3, 2, 2, 3, 1, 1, 2,
       2, 2, 0, 2, 2, 1, 0, 4, 4, 4, 4, 0, 3, 1, 4, 1, 2, 4, 0, 0, 0, 4,
       1, 1, 2, 1, 4, 1, 4, 1, 2, 4, 1, 3, 4, 1, 1, 2, 4, 2, 3, 1, 1, 4,
       2, 0, 3, 2, 1, 2, 2, 1, 3, 0, 2, 3, 1, 0, 1, 1, 1, 0, 2, 1, 1, 3,
       0, 3, 4, 2, 1, 1, 1, 3, 2, 2, 4, 4, 0, 0, 0, 2, 3, 2, 3, 1, 2, 1,
       2, 0, 1, 0, 3, 3, 4, 3, 2, 1, 4, 1, 3, 2, 2, 0, 2, 3, 0, 2, 3, 2,
       2, 2, 2, 3, 0, 1, 0, 2, 2, 2, 3, 4, 1, 1, 4, 3, 2, 1, 2, 2, 0, 3,
       0, 2, 2, 4, 3, 3, 2, 1, 1, 1, 0, 0, 1, 2, 1, 0, 1, 0, 2, 0, 1, 4,
       0, 3, 3, 1, 1, 3, 3, 4, 2, 1, 0, 0, 4, 4, 0, 3, 4, 4, 3, 1, 4, 1,
       1, 1, 4, 3, 0, 4, 4, 1, 0, 2, 2, 1, 2, 2, 2, 4, 1, 3, 2, 0, 1, 2,
       4, 4, 1, 4, 1, 1, 2, 1, 2, 4, 1, 3, 1, 2, 2, 1, 1, 3, 2, 2, 2, 1,
       1, 2, 2, 1, 4, 1, 4, 3, 1, 1, 1, 3, 2, 1, 0, 2, 4, 4, 0, 2, 4, 3,
       0, 0, 1, 0, 0, 0, 2, 2, 2, 2, 1, 1, 3, 2, 4, 1, 4, 1, 1, 4, 3, 0,
       2, 2, 2, 1, 1, 1, 1, 4, 1, 0, 1, 4, 1, 4, 4, 2, 4, 4, 2, 1, 2, 2,
       4, 2, 0, 2, 3, 3, 4, 0, 4, 2, 2, 2, 2, 1, 2, 2, 1, 0, 2, 2, 1, 2,
       0, 2, 0, 0, 0, 2, 2, 4, 2, 1, 1, 2, 3, 1, 1, 2, 1, 1, 1, 0, 2, 0,
       3, 0, 0, 0, 1, 2, 0, 4, 1, 1, 1, 3, 0, 2, 0, 4, 4, 4, 2, 1, 2, 1,
       1, 3, 1, 2, 2, 3, 3, 2, 2, 1, 1, 1, 0, 4, 2, 1, 1, 4, 1, 2, 2, 4,
       3, 1, 1, 3, 2, 4, 1, 2, 2, 2, 2, 4, 0, 2, 0, 0, 2, 0, 0, 4, 4, 3,
       4, 1, 2, 0, 3, 3, 4, 1, 1, 2, 2, 1, 2, 0, 3, 0, 1, 1, 1, 1, 4, 0,
       2, 0, 2, 0, 1, 3, 4, 1, 2, 2, 2, 4, 0, 3, 2, 1, 1, 0, 2, 1, 1, 3,
       2, 1, 2, 3, 1, 4, 1, 2, 3, 4, 0, 2, 0, 2, 2, 0, 3, 0, 4, 2, 4, 2,
       2, 3, 3, 2, 3, 2, 1, 2, 1, 4, 2, 1, 0, 2, 2, 0, 0, 1, 2, 1, 0, 2,
       4, 1, 1, 1, 2, 1, 2, 0, 2, 1, 4, 4, 4, 4, 3, 0, 2, 4, 2, 2, 4, 4,
       2, 4, 1, 2, 1, 0, 3, 3, 1, 4, 1, 1, 3, 3, 3, 2, 1, 4, 1, 1, 1, 0,
       2, 4, 2, 0, 0, 2, 4, 2, 2, 4, 4, 1, 1, 4, 3, 2, 3, 4, 4, 2, 4, 1,
       1, 2, 2, 4, 1, 4, 4, 1, 1, 4, 2, 3, 4, 1, 4, 1, 1, 4, 4, 1, 1, 1,
       4, 0, 2, 2, 1, 2, 4, 4, 1, 2, 0, 1, 4, 0, 3, 2, 2, 1, 3, 1, 0, 0,
       3, 1, 2, 0, 2, 4, 1, 4, 1, 0, 0, 1, 2, 1, 1, 3, 2, 2, 1, 2],
      dtype=int32)
```

In [302…
```
d=x_var
d["labels"] =km.labels_
```

In [303…
```
## elbow technique to decide the number of clusters
import matplotlib.pyplot as plt
import seaborn as sns
```

In [304…
```
wcss=[]
for i in range(1,14):
  km = KMeans(n_clusters=i, init="k-means++")
  km.fit(x_var)
  wcss.append(km.inertia_)
plt.figure(figsize=(12,6))
sns.lineplot(range(1,14), wcss, color="red")
```

Out[304…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe4721c29d0>
```



In [305…
```
km.inertia_
```
Out[305…
```
62.401864862006605
```

See by inertia at 5 clusters we got low inertia that is 62% so our data distinguished into 5 clusters

Model Selection

Using Cross Validation Technique to finalize model

1.I am using cross validation technique other than over and under sampling.

2.Because In over and under sampling there is a chance that accuracy is biased because we are adding data points in over sampling and removing some data points in under samplaing.

3.But in cross validation technique this algorithm takes random sample subsets in dataset as test data and goes through further analysis so in this process there is less chance of overfitting and we can get unbiased accuracy.

4.And in this technique it calculates accuracies of all subsets and takes its average so means it considers all values in data even data is in imbalance.

Logistic Regression Cross Validation

Cross validation technique for logistic regression where we used PCA variables as independent variables.

In [306…
```
from sklearn.model_selection import cross_validate
```

I am taking 5 subsets

In [307…
```
res_lr=cross_validate(lr,x_var,y_var,cv=5,return_train_score=True)
```

Test Score

In [308…
```
test_lr=np.average(res_lr['test_score'])
test_lr
```
Out[308…
```
0.7638314176245211
```

In [309…
```
train_lr=np.average(res_lr['train_score'])
train_lr
```
Out[309…
```
0.7710612828300876
```

Decision Tree Cross Validation

I am taking 5 subsets

In [310…
```
res_dt=cross_validate(pdt,x_var,y_var,cv=5,return_train_score=True)
```

Test Score

In [311…
```
test_dt=np.average(res_dt['test_score'])
test_dt
```
Out[311…
```
0.6615229885057472
```

In [312…
```
train_dt=np.average(res_dt['train_score'])
train_dt
```

```
Out[312… 1.0
```

Random Forest Cross Validation

I am taking 4 subsets

```
In [313… res_rf=cross_validate(rf,x_rforest,y_rforest,cv=4,return_train_score=True)
```

Test Score

```
In [314… test_rf=np.average(res_rf['test_score'])
         test_rf
```

```
Out[314… 0.7527624309392265
```

```
In [315… train_rf=np.average(res_rf['train_score'])
         train_rf
```

```
Out[315… 1.0
```

Support Vector Machine Model Cross Validation

I am taking 5 subsets

```
In [316… res_svm=cross_validate(svm,x_svm,y_svm,cv=5,return_train_score=True)
```

Test Score

```
In [317… test_svm=np.average(res_svm['test_score'])
         test_svm
```

```
Out[317… 0.7389463601532567
```

```
In [318… train_svm=np.average(res_svm['train_score'])
         train_svm
```

```
Out[318… 0.7569060806384373
```

Naive Bayes Classification Algorithm Model Cross Validation

I am taking 5 subsets

```
In [319… res_nb=cross_validate(nb,data_sc_df,y_var,cv=5,return_train_score=True)
```

Test Score

```
In [320… test_nb=np.average(res_nb['test_score'])
         test_nb
```

```
Out[320… 0.6560823754789272
```

Train Score

```
In [321… train_nb=np.average(res_nb['train_score'])
         train_nb
```

```
Out[321… 0.6560776606515395
```

Comparing Models Based On Cross Validation Technique

In [322…
```
from prettytable import PrettyTable
x=PrettyTable(['Model','Train_score','Test_score'])
x.add_row(['Logistic Regression',0.7665713775236733,0.7569061302681993])
x.add_row(['Decision Tree',1.0,0.6587164750957853])
x.add_row(['Random Forest',1.0,0.7596685082872928])
x.add_row(['Support Vector Machine',0.7272098147817283,0.7126819923371647])
x.add_row(['Naive Bayes Classification Algorithm',0.6560776606515395,0.6560823754789272])
print(x)
```

```
+--------------------------------------+--------------------+--------------------+
|                Model                 |    Train_score     |     Test_score     |
+--------------------------------------+--------------------+--------------------+
|         Logistic Regression          | 0.7665713775236733 | 0.7569061302681993 |
|            Decision Tree             |        1.0         | 0.6587164750957853 |
|            Random Forest             |        1.0         | 0.7596685082872928 |
|        Support Vector Machine        | 0.7272098147817283 | 0.7126819923371647 |
| Naive Bayes Classification Algorithm | 0.6560776606515395 | 0.6560823754789272 |
+--------------------------------------+--------------------+--------------------+
```

1.There is only 1% difference in accuracy of train and test data in Logistic Regression.

2.Among all models logistic regression got more accuracy and overfitting also less so we fix logistic regression as our final model.

Finding Best Parameters For Logistic Regression By Hyperparametric Tuning Technique

In [323…
```
# Gridsearchcv
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

In [324…
```
pca=PCA(n_components=6)
x_vari=pca.fit_transform(data_sc_df)
```

In [328…
```
new_data1=pd.DataFrame(var_x,columns=['pca1','pca2','pca3','pca4','pca5','pca6'])
```

In [332…
```
x_traing,x_testg,y_traing,y_testg=train_test_split(new_data1,y_var,test_size=0.2,random_state=50)
```

In [341…
```
grid = {'C' : np.logspace(-3,3,7), 'penalty'  : ['l1','l2']}
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg,grid,cv = 10)
logreg_cv.fit(x_traing,y_traing)
print("Tuned hyperparameters : (best parameters)", logreg_cv.best_params_)
print("accuracy : ",logreg_cv.best_score_)
```

```
Tuned hyperparameters : (best parameters) {'C': 10.0, 'penalty': 'l2'}
accuracy :  0.7686025408348457
```

By this also we proved that logistic regression is best because we got 76% accuracy in our model also we got 0.7569 ~ 76 so we predicted correctly and above parameters are best parameters

Final Model With Function

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
from sklearn.linear_model import LogisticRegression
```

Using Logistic Model

In [352…
```
def predict_class_logistic(x_train,x_test,y_train,y_test):
  """this function predicts the final o/p class label"""
  """ put scaled data to form PCA after all necessary preprocessing steps and insert in model"""
  model = LogisticRegression(penalty = 'l2',C = 10.0)
  model.fit(x_train,y_train)
  pred = model.predict(x_test)
  return roc_auc_score(y_test,pred).round(2)
```

In [353…
```
predict_class_logistic(x_traing,x_testg,y_traing,y_testg)
```

Out[353…
```
0.74
```

By this we defined our model function

Loading [MathJax]/extensions/Safe.js