Simplilearn Post Graduate Program - Data Science

In Partnership With Purdue University

Capstone Project Report - Healthcare PGP (Diabetes)

Organization: **Simplilearn - Purdue University**
Batch: **PGP DS Mar 2022 COHORT 2**
Course: **PC DS - Data Science Capstone**
Project: **Healthcare PGP (Diabetes)**
Programming Language: **Python**
Submitted by: **Lavkush Singh**

Problem Statement

The project aims at building 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

- **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 are 0

Analysis Tasks to be performed

Task I: Data Exploration

```
- 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
    * SkinThickness
    * Insulin
    * BMI
- Visually explore these variables using histograms. Treat the
missing values accordingly.
- 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.
- Check the balance of the data by plotting the count of outcomes by
their value. Describe your findings and plan future course of action.
- Create scatter charts between the pair of variables to understand
the relationships. Describe your findings.
- Perform correlation analysis. Visually explore it using a heat map.
```

Task II: Data Modeling

```
- Devise strategies for model building. It is important to decide the
right validation framework. Express your thought process.
- Apply an appropriate classification algorithm to build a model.
- Compare various models with the results from KNN algorithm.
- 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.
```

Task III: Tableau Report

```
- 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.
```

Task I: Data Exploration

In [1]:
```
# Importing required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import dabl
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, (
from plot_metric.functions import BinaryClassification
```

```
C:\Users\Lenovo\.conda\envs\machine_learning\lib\site-packages\sklearn\experimental\e
nable_hist_gradient_boosting.py:16: UserWarning: Since version 1.0, it is not needed
to import enable_hist_gradient_boosting anymore. HistGradientBoostingClassifier and H
istGradientBoostingRegressor are now stable and can be normally imported from sklear
n.ensemble.
  warnings.warn(
C:\Users\Lenovo\.conda\envs\machine_learning\lib\site-packages\xgboost\compat.py:36:
FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas in a f
uture version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [2]:
```
# settings to display all columns

pd.set_option("display.max_columns", None)
pd.options.display.max_rows = None
```

In [3]:
```
# reading the data

diabetes_data = pd.read_csv('Datasets/health care diabetes.csv')
```

In [4]:
```
diabetes_data.head() # viewing first few observations of train dataset
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

Task I (a): Perform descriptive analysis. (Preliminary Data Inspection and Data Cleaning)

In [5]:
```
diabetes_data.shape # checking rows and cols of the train dataset
```
Out[5]:
```
(768, 9)
```

In [6]:
```
diabetes_data.info()  # understanding column wise datatype and null values
```
```
<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 [7]:
```
diabetes_data.describe() # descriptive statistics of numerical columns
```
Out[7]:

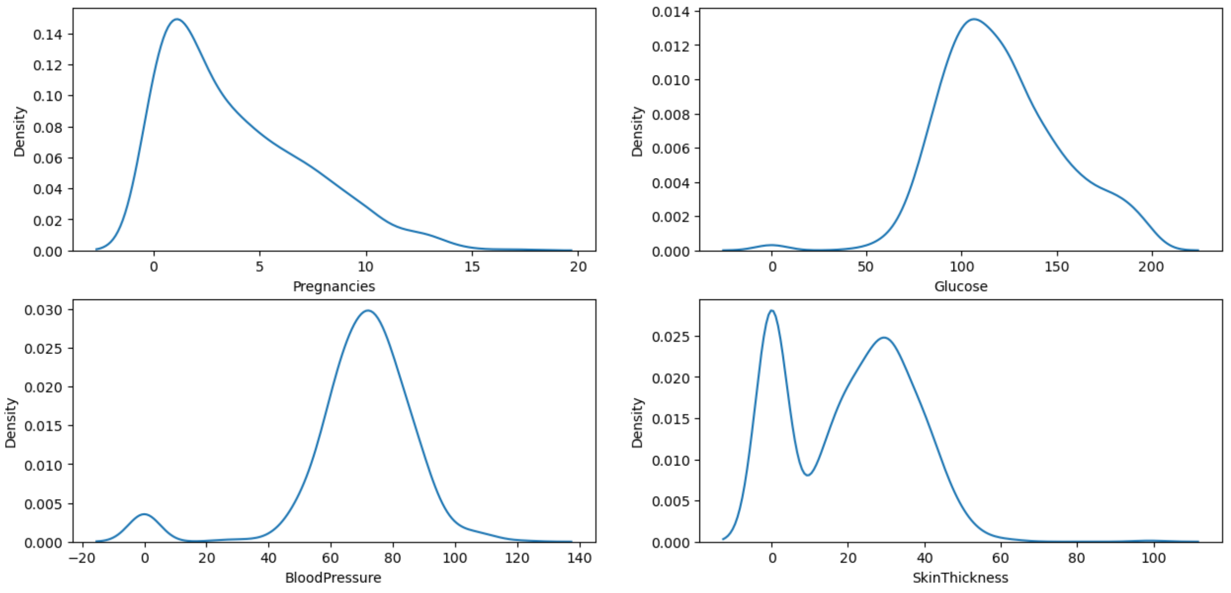
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	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	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Visualizing the distribution of columns (independent variables) via KDE Plot

In [8]:
```
# converting columns into array to traverse through it and plotting graph as sub-plots

arr_cols = np.array(diabetes_data.columns[:4]).reshape(2,2)
arr_cols
```
Out[8]:
```
array([['Pregnancies', 'Glucose'],
       ['BloodPressure', 'SkinThickness']], dtype=object)
```

In [9]:
```
# plotting KDE plot to view how the data is distributed
```

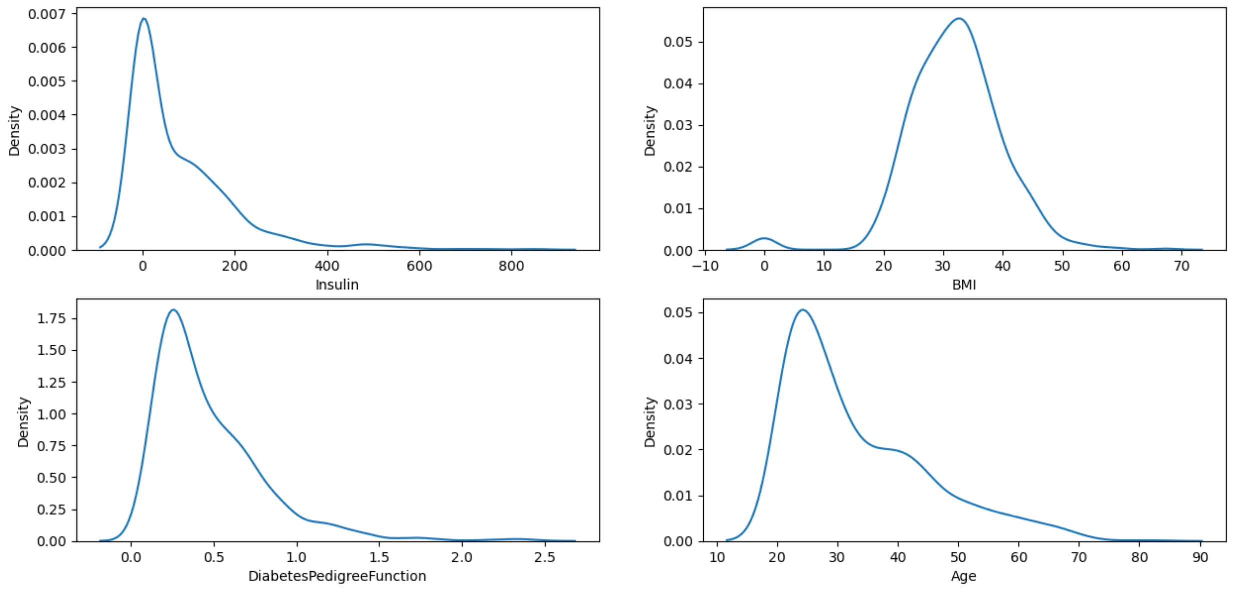
```python
fig, axes = plt.subplots(2, 2, figsize=(15, 7))
for i in range(2):
    for j in range(2):
        sns.kdeplot(ax=axes[i, j], data = diabetes_data, x = arr_cols[i,j])
```



In [10]:
```python
# converting columns into array to traverse through it and plotting graph as sub-plots

arr_cols = np.array(diabetes_data.columns[4:-1]).reshape(2,2)
arr_cols
```

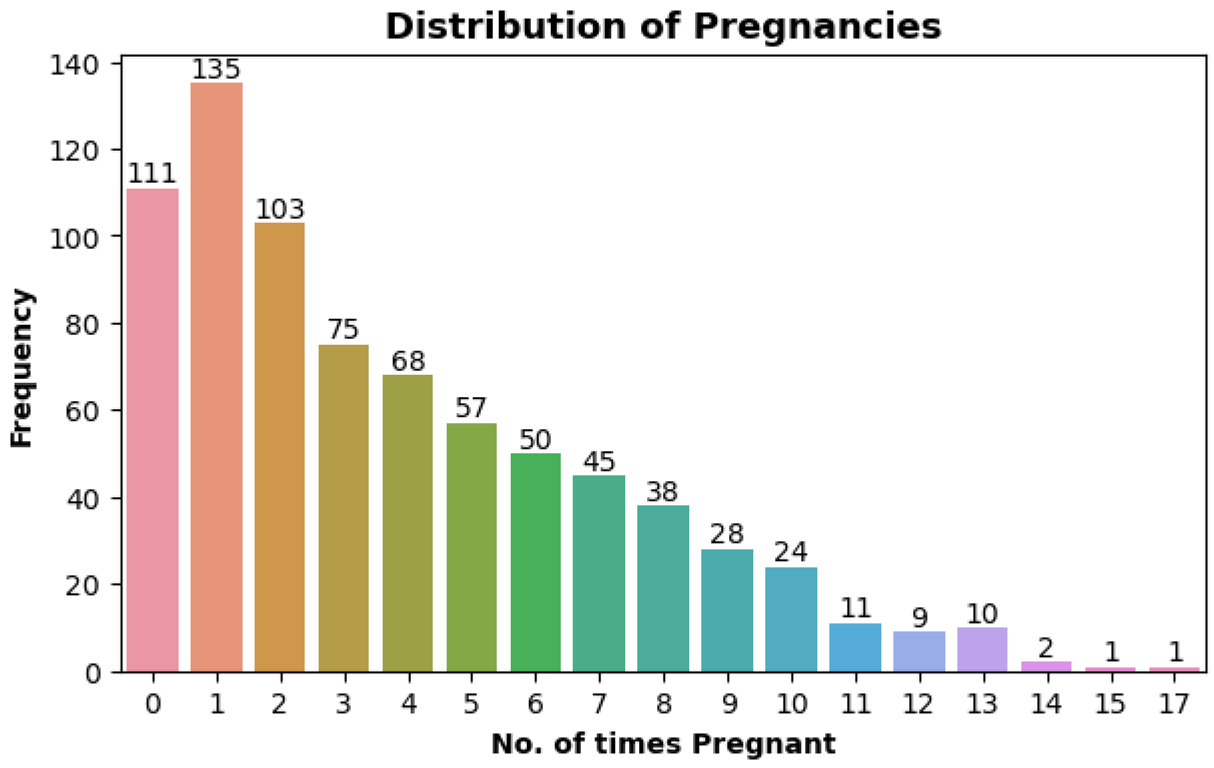
Out[10]:
```
array([['Insulin', 'BMI'],
       ['DiabetesPedigreeFunction', 'Age']], dtype=object)
```

In [11]:
```python
# plotting KDE plot to view how the data is distributed

fig, axes = plt.subplots(2, 2, figsize=(15, 7))
for i in range(2):
    for j in range(2):
        sns.kdeplot(ax=axes[i, j], data = diabetes_data, x = arr_cols[i,j])
```

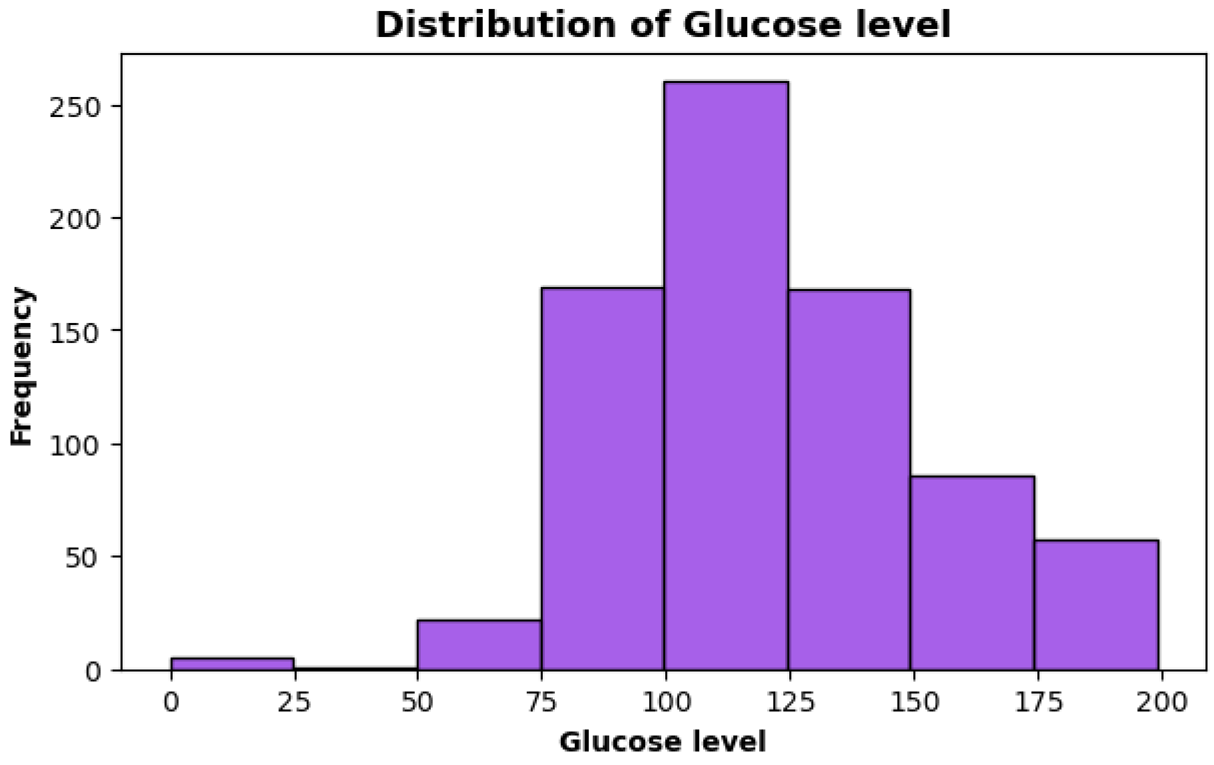


Task I (b): Visual exploration of individual variables using histograms

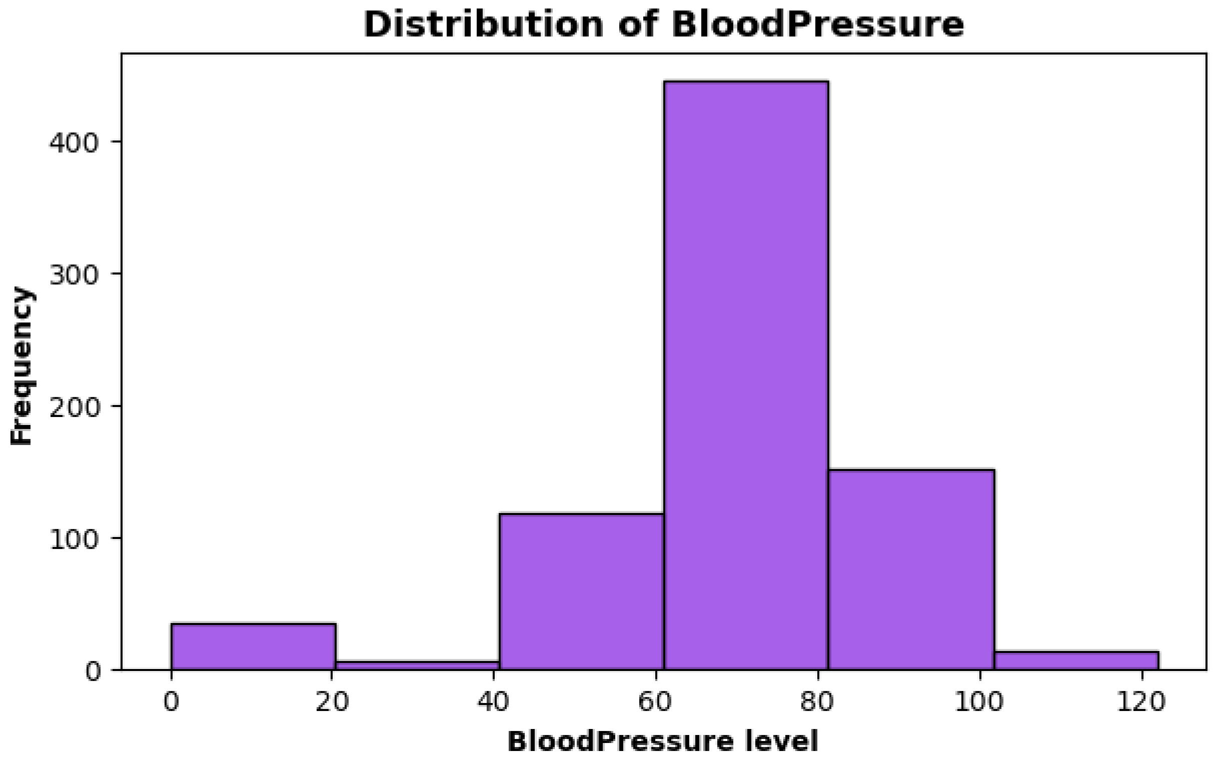
In [12]:
```
plt.figure(figsize=(7,4))
ax = sns.countplot( x = 'Pregnancies' , data = diabetes_data)
ax.bar_label(ax.containers[0])
plt.title('Distribution of Pregnancies', fontsize = 13, fontweight="bold")
plt.xlabel('No. of times Pregnant', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



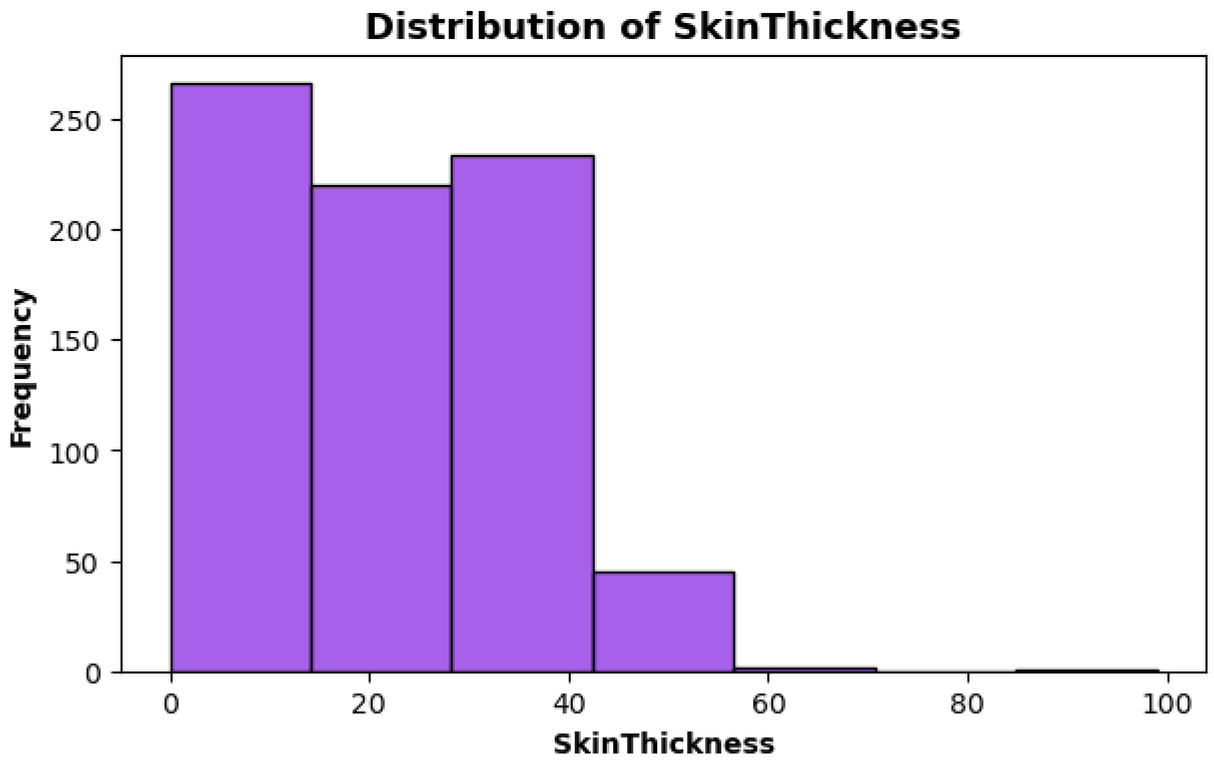
In [13]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'Glucose' , data = diabetes_data, bins = 8, color= 'blueviolet')
plt.title('Distribution of Glucose level', fontsize = 13, fontweight="bold")
plt.xlabel('Glucose level', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



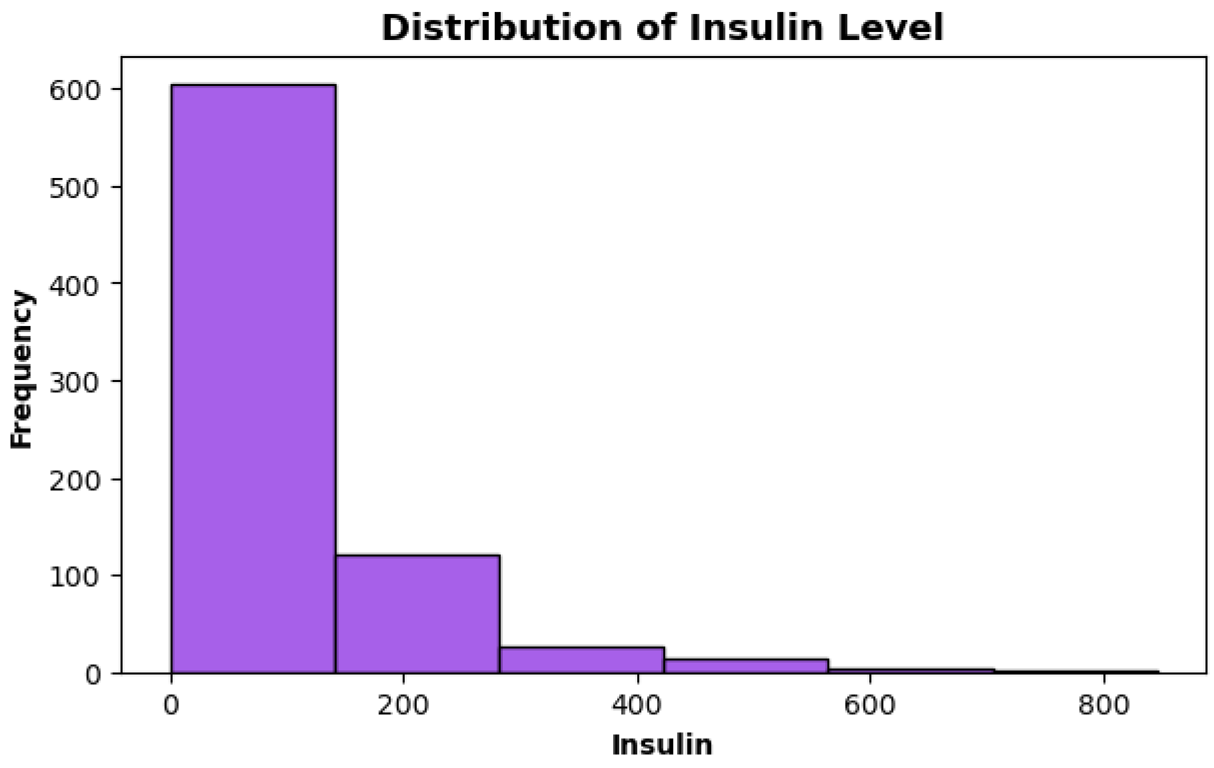
In [14]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'BloodPressure' , data = diabetes_data, bins = 6, color= 'bluevioleت
plt.title('Distribution of BloodPressure', fontsize = 13, fontweight="bold")
plt.xlabel('BloodPressure level', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



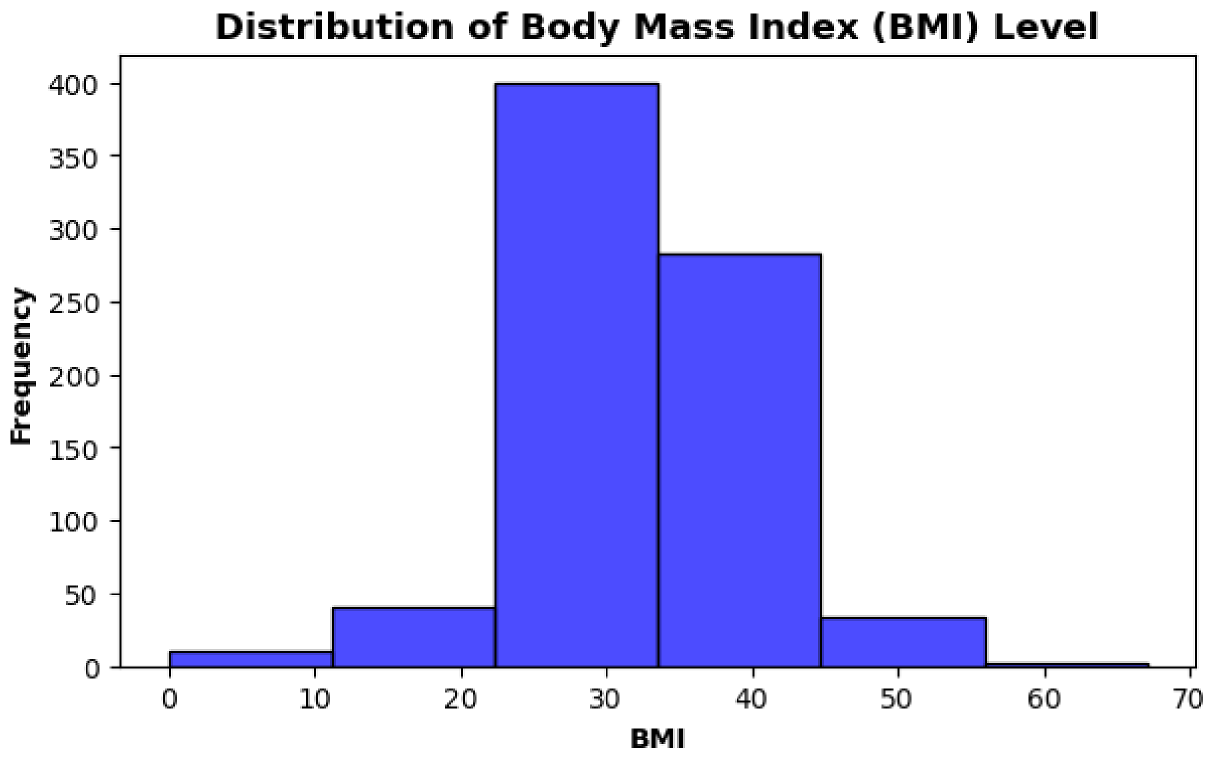
In [15]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'SkinThickness' , data = diabetes_data, bins = 7, color= 'bluevioleت
plt.title('Distribution of SkinThickness', fontsize = 13, fontweight="bold")
plt.xlabel('SkinThickness', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



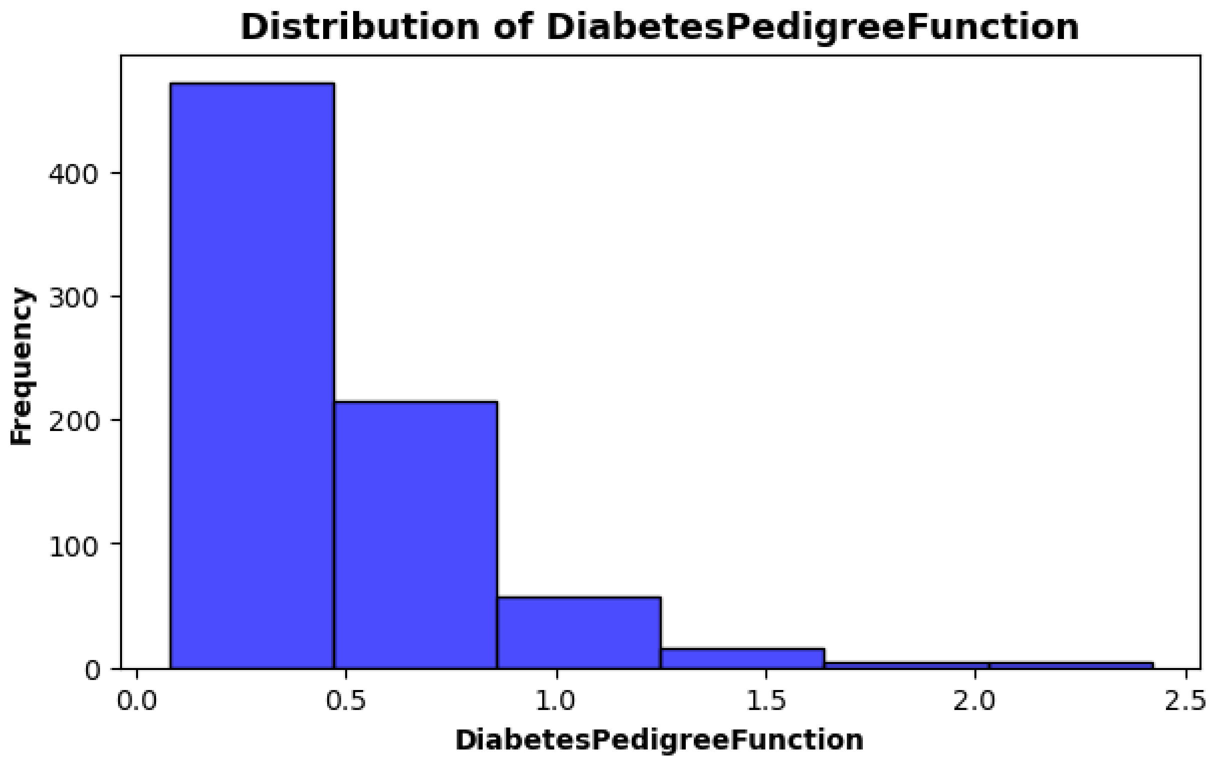
In [16]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'Insulin' , data = diabetes_data, bins = 6, color= 'blueviolet')
plt.title('Distribution of Insulin Level', fontsize = 13, fontweight="bold")
plt.xlabel('Insulin', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



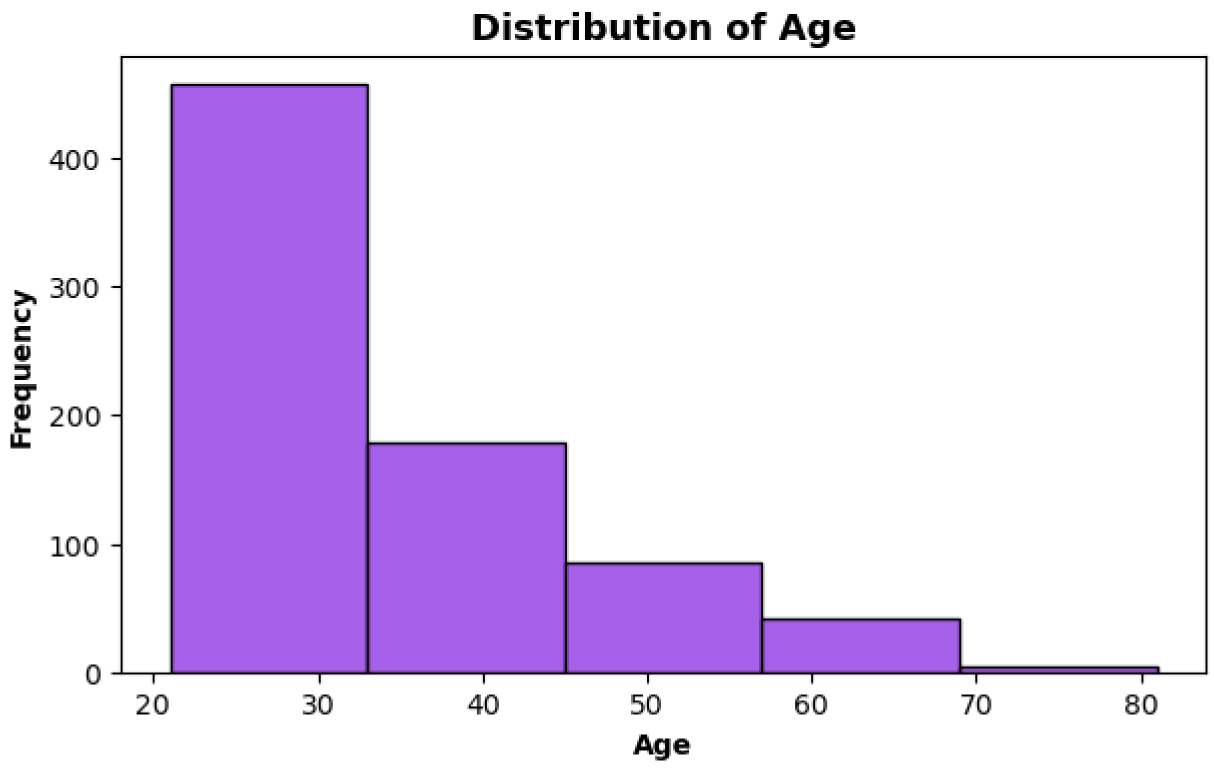
In [17]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'BMI' , data = diabetes_data, bins = 6, color= 'blue', alpha = 0.7)
plt.title('Distribution of Body Mass Index (BMI) Level', fontsize = 13, fontweight="bo
plt.xlabel('BMI', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



In [18]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'DiabetesPedigreeFunction' , data = diabetes_data, bins = 6, color=
plt.title('Distribution of DiabetesPedigreeFunction', fontsize = 13, fontweight="bold'
plt.xlabel('DiabetesPedigreeFunction', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



In [19]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = 'Age' , data = diabetes_data, bins = 5, color= 'blueviolet')
plt.title('Distribution of Age', fontsize = 13, fontweight="bold")
plt.xlabel('Age', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



In [20]:
```
people_with_0_pregnancies = diabetes_data[diabetes_data['Pregnancies'] == 0].shape[0]
people_with_0_pregnancies
```

Out[20]:
```
111
```

In [21]:
```
people_with_atleast_1_pregnancies = diabetes_data.shape[0] - people_with_0_pregnancies
people_with_atleast_1_pregnancies
```

Out[21]:
```
657
```

In [22]:
```
percent_people_with_0_pregnancies = round((people_with_0_pregnancies/diabetes_data.sha
percent_people_with_atleast_1_pregnancies = round((people_with_atleast_1_pregnancies/d
```

In [23]:
```
print(f"Percent of People in dataset with 0 pregnancies: {percent_people_with_0_pregna
print(f"Percent of People in dataset with atlest 1 pregnancies: {percent_people_with_a
```

```
Percent of People in dataset with 0 pregnancies: 14.45%
Percent of People in dataset with atlest 1 pregnancies: 85.55%
```

Analysis Summary:

- From the descriptive statistics, the columns ‘Glucose’, ‘Blood Pressure’, ‘SkinThickness’, ‘Insulin’, ‘DiabetesPedigreeFunction’ has outliers as the mean and median values are not close.
- ‘Glucose’, ‘BloodPressure’, ‘BMI’ columns has approximately normal distribution, while other features have skewed distributions
- There are total 768 observations for 9 variables.
- Of the total 768 observations, 14.45% (111 observations) comprises of the persons who have been pregnant. This data may consists of males, however this cannot be verified since

the data given does not have ‘Gender’ column to determine. But the 85.55% (657 observations) of the data is for women, because they have been pregnant for at least once.

- Among Pregnancies, Maximum Number of people has been pregnant for at least once. Second highest number of people are those who have never been pregnant, this might also include the number of males in the consideration; however 'Gender' of the dataset is not given.

Task I (c): Inspection of missing values and treatment

In [24]:
```
missing_val_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

# as per the problem description, 'missing_val_cols' columns shouldn't have 0 as the v

(diabetes_data[missing_val_cols] == 0).sum() # Total number of Zeros (missing values)
```

Out[24]:
```
Glucose            5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI               11
dtype: int64
```

In [25]:
```
# getting percent of missing values of the columns

percent_of_missing_vals = ((diabetes_data[missing_val_cols] == 0).sum()/diabetes_data.
percent_of_missing_vals
```

Out[25]:
```
Glucose           0.65
BloodPressure     4.56
SkinThickness    29.56
Insulin          48.70
BMI               1.43
dtype: float64
```

In [26]:
```
# Getting the dataframe after removing all the missing values (basically removing 0's

diabetes_data_without_missing_vals = diabetes_data[missing_val_cols].loc[(diabetes_dat
diabetes_data_without_missing_vals.head()
```

Out[26]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
3	89	66	23	94	28.1
4	137	40	35	168	43.1
6	78	50	32	88	31.0
8	197	70	45	543	30.5
13	189	60	23	846	30.1

In [27]:
```
diabetes_data_without_missing_vals.shape  # checking the shape of dataframe
```

Out[27]:
```
(392, 5)
```

In [28]:
```
diabetes_data.columns
```

Out[28]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [29]:
```
# We have data for without missing values, but only for 'missing_val_cols'
# In this step, I have combined the remaining columns from original dataframe using me

diabetes_data_without_missing_vals = pd.merge(diabetes_data_without_missing_vals,
                                              diabetes_data[['Pregnancies', 'DiabetesP
                                              left_index = True, right_index=True)
diabetes_data_without_missing_vals.head()
```

Out[29]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pregnancies	DiabetesPedigreeFunction	Age
3	89	66	23	94	28.1	1	0.167	21
4	137	40	35	168	43.1	0	2.288	33
6	78	50	32	88	31.0	3	0.248	26
8	197	70	45	543	30.5	2	0.158	53
13	189	60	23	846	30.1	1	0.398	59

In [30]:
```
diabetes_data_without_missing_vals.columns
```

Out[30]:
```
Index(['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
       'Pregnancies', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [31]:
```
# checking the percent of correlation of the columns which have missing values with th

corr_of_missing_vals_cols_with_target = diabetes_data_without_missing_vals[missing_val
corr_of_missing_vals_cols_with_target = corr_of_missing_vals_cols_with_target['Outcome
corr_of_missing_vals_cols_with_target
```

Out[31]:
```
Glucose           52.0
BloodPressure     19.0
SkinThickness     26.0
Insulin           30.0
BMI               27.0
Outcome          100.0
Name: Outcome, dtype: float64
```

In [32]:
```
# converting the missing values percent and correlation percent values to dataframe, c

missing_and_corr_df = pd.DataFrame(percent_of_missing_vals).join(corr_of_missing_vals_
missing_and_corr_df.columns = ['column', 'cols_missing_vals_percent', 'cols_missing_va
missing_and_corr_df
```

Out[32]:

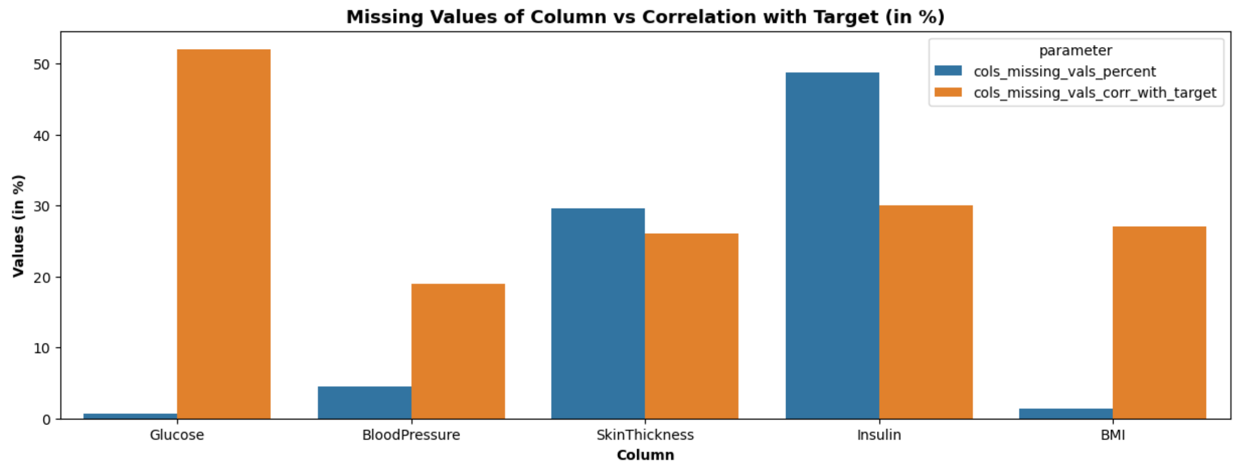
	column	cols_missing_vals_percent	cols_missing_vals_corr_with_target
0	Glucose	0.65	52.0
1	BloodPressure	4.56	19.0
2	SkinThickness	29.56	26.0
3	Insulin	48.70	30.0
4	BMI	1.43	27.0

In [33]:
```
# transforming the dataframe to get it ready for side-by-side bar chart (for comparisi

df = pd.melt(missing_and_corr_df, id_vars = "column").rename(columns={"variable": "par
df
```

Out[33]:

	column	parameter	value
0	Glucose	cols_missing_vals_percent	0.65
1	BloodPressure	cols_missing_vals_percent	4.56
2	SkinThickness	cols_missing_vals_percent	29.56
3	Insulin	cols_missing_vals_percent	48.70
4	BMI	cols_missing_vals_percent	1.43
5	Glucose	cols_missing_vals_corr_with_target	52.00
6	BloodPressure	cols_missing_vals_corr_with_target	19.00
7	SkinThickness	cols_missing_vals_corr_with_target	26.00
8	Insulin	cols_missing_vals_corr_with_target	30.00
9	BMI	cols_missing_vals_corr_with_target	27.00

In [34]:
```
# This graph explains about how much missing values the column has, and how much corre

plt.figure(figsize=(15,5))
sns.barplot(x = 'column', y='value', hue = 'parameter',data=df)
plt.title('Missing Values of Column vs Correlation with Target (in %)', fontsize = 13,
plt.xlabel('Column', fontweight="bold")
plt.ylabel('Values (in %)', fontweight="bold")
plt.show()
```



In [35]:
```
# getting the mean and median values of columns having missing values, from the origir

diabetes_data[missing_val_cols].describe().loc[['mean', '50%']]
```

Out[35]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
mean	120.894531	69.105469	20.536458	79.799479	31.992578
50%	117.000000	72.000000	23.000000	30.500000	32.000000

In [36]:
```
# getting the mean and median values of columns having missing values, from the datase

diabetes_data_without_missing_vals[missing_val_cols].describe().loc[['mean', '50%']]
```

Out[36]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
mean	122.627551	70.663265	29.145408	156.056122	33.086224
50%	119.000000	70.000000	29.000000	125.500000	33.200000

In [37]:
```
# creating the copy of original dataset. The new dataset 'diabetes_data_missing_impute

diabetes_data_missing_imputed = diabetes_data.copy(deep = True)
```

In [38]:
```
# because the columns ['Glucose', 'Insulin'] has outlier (diff b/w mean and median is

for col in ['Glucose', 'Insulin']:
    diabetes_data_missing_imputed[col] = diabetes_data_missing_imputed[col].apply(
                                                        lambda x: np.median(diabetes_c
```

In [39]:
```
# because the columns ['BloodPressure', 'SkinThickness', 'BMI'] has no outliers (mean
# mean imputation is done

for col in ['BloodPressure', 'SkinThickness', 'BMI']:
    diabetes_data_missing_imputed[col] = diabetes_data_missing_imputed[col].apply(
                                                        lambda x: np.mean(diabetes_dat
```

In [40]:
```
(diabetes_data_missing_imputed[missing_val_cols] == 0).sum() # Validating if the missi
```

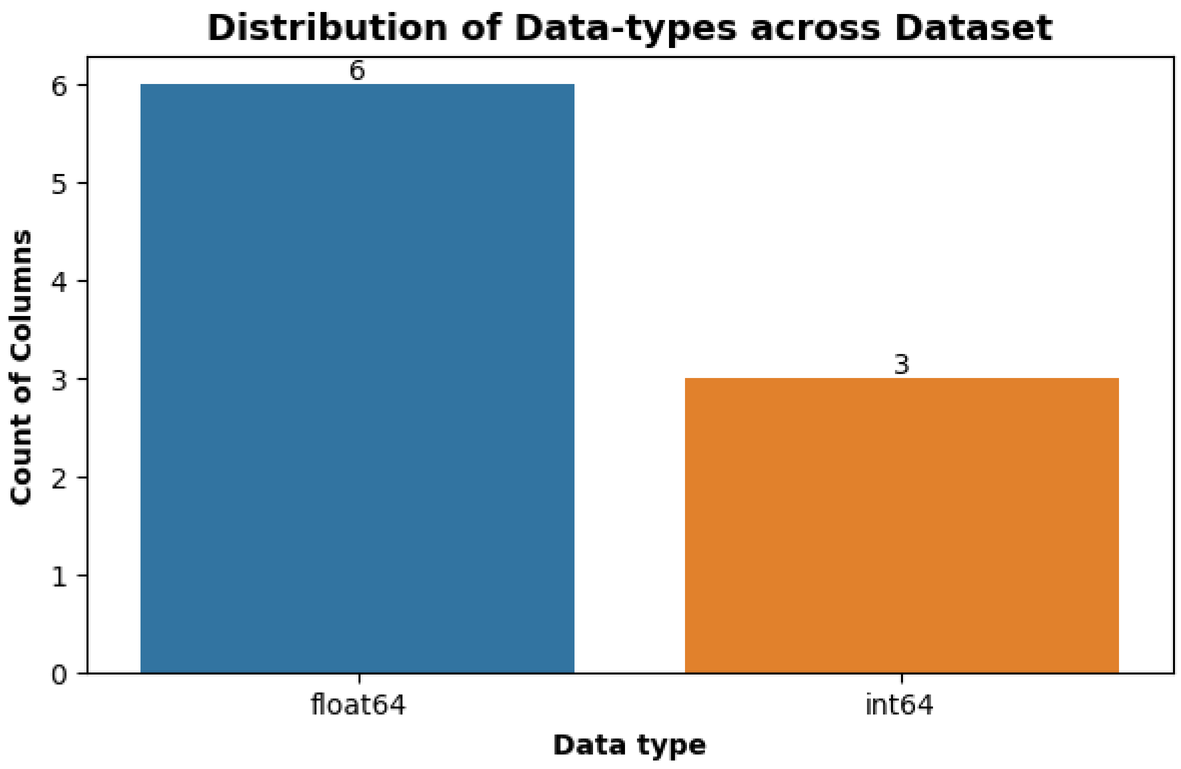
Out[40]:
```
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
dtype: int64
```

Task I (d): Count Plot of Variable Datatypes

In [41]:
```
# geting the number of variables (columns) per data-type

dtypes_var = diabetes_data_missing_imputed.dtypes.value_counts()
dtypes_var
```

Out[41]:
```
float64    6
int64      3
dtype: int64
```

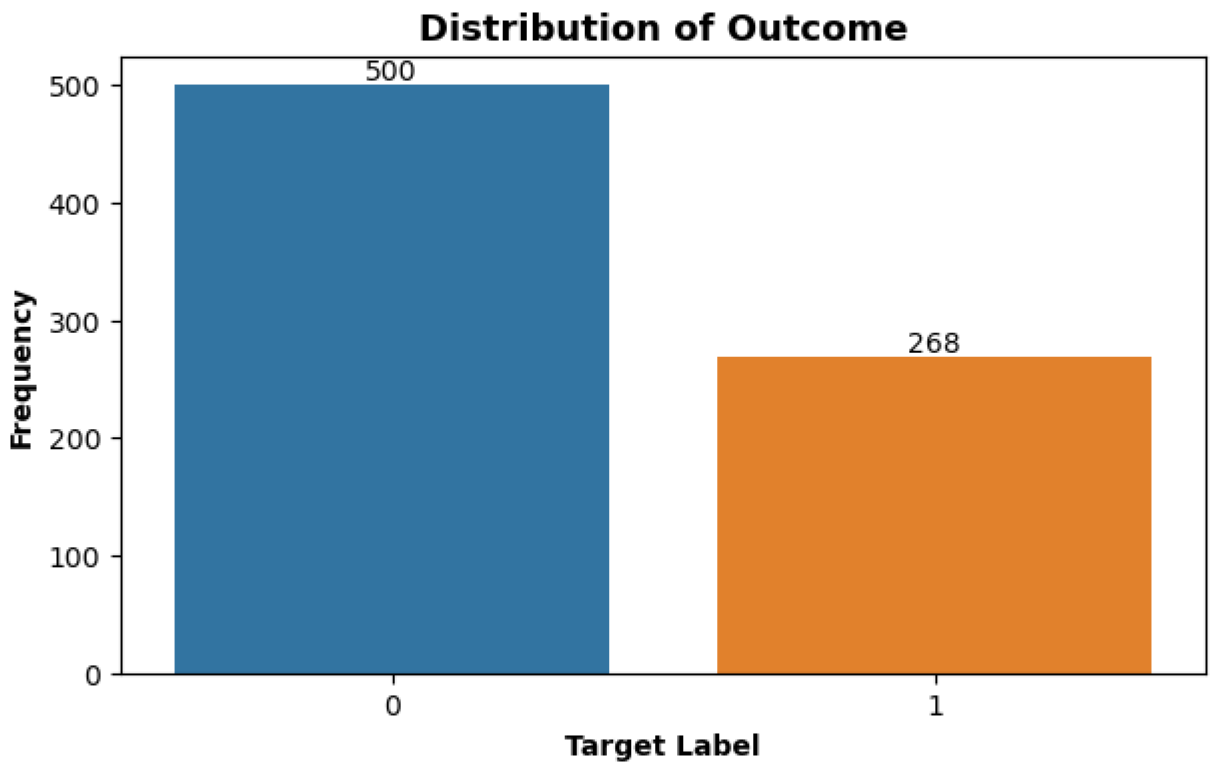
In [42]:
```
# Distribution of Data-types across Dataset

plt.figure(figsize=(7,4))
ax = sns.barplot( x =  dtypes_var.index, y = dtypes_var.values)
ax.bar_label(ax.containers[0])
plt.title('Distribution of Data-types across Dataset', fontsize = 13, fontweight="bol
plt.xlabel('Data type', fontweight="bold")
plt.ylabel('Count of Columns', fontweight="bold")
plt.show()
```



Task I (e): Count Plot of Outcome Variable

In [43]:
```
# Distribution of the Target label count across the dataset

plt.figure(figsize=(7,4))
ax = sns.countplot( x = 'Outcome' , data = diabetes_data_missing_imputed)
```

```
ax.bar_label(ax.containers[0])
plt.title('Distribution of Outcome', fontsize = 13, fontweight="bold")
plt.xlabel('Target Label', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



In [44]:
```
# percent of target label in the dataset

(diabetes_data_missing_imputed['Outcome'].value_counts()/diabetes_data_missing_imputed
```

Out[44]:
```
0    65.0
1    35.0
Name: Outcome, dtype: float64
```

Analysis Summary:

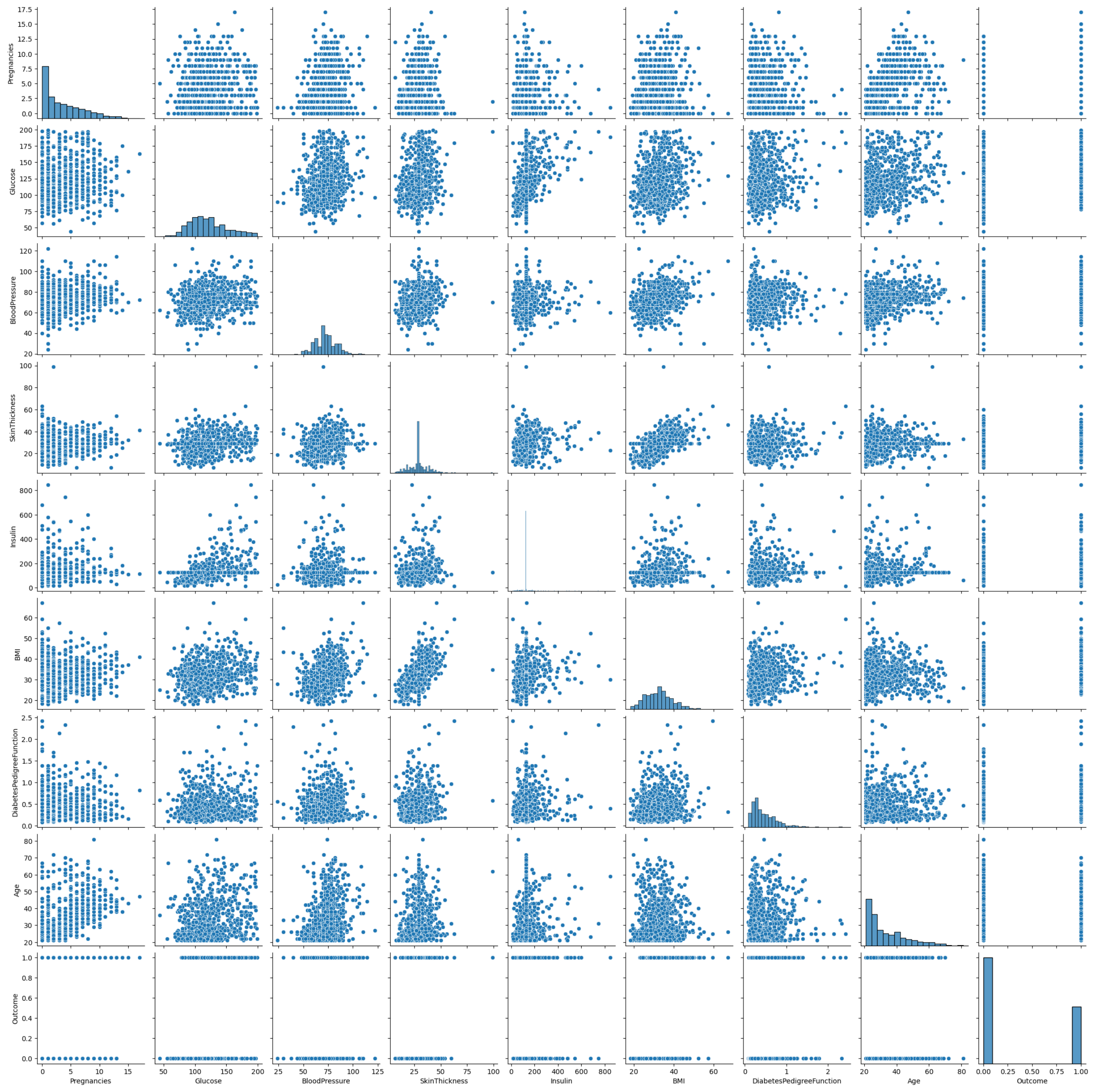
- It was observed that there are 9 columns (or features, independent variables) present in the dataset. Of these, 6 columns are of float type (meaning the data recorded was numerical with decimals) and 3 columns was of integer type (meaning the data recorded was numerical without any decimal places, aka whole numbers)
- As per the problem statement, inofmration is provided that '0' value (observation) of the columns ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI'] is absurd, and that the '0' datapoint of these columns can be treated as ‘missing values’
- Columns ‘SkinThickness’,’ Insulin’ has maximum of the missing values, of 29.56% and 48.70% respectively.
- Columns ‘Glucose’, ‘BloodPressure’, ‘BMI’ has relatively negligible/lower missing values count, of about 0.65%, 4.56% and 1.43% respectively.
- It was observed that among the columns having missing values, the columns (after removing missing values) Glucose is 52% correlated with the target. The colums which has

maximum of the missing values, ‘SkinThickness’,’ Insulin’ have 26% and 30% correlation with the target variable.

- Missing values of the columns ['Glucose', 'Insulin'] were imputed with median and ['BloodPressure', 'SkinThickness', 'BMI'] was imputed with mean, keeping in mind the distribution and outliers presence.
- It was observed that the target variable is binary type, it has values 0 and 1, which implies Diabetic and Non-Diabetic respectively.
- Target values are not balanced as ‘0’ (500 count) value comprises of 65% (268 count) and ‘1’ comprises of 35% of the column values.

Task I (f): Scatter Charts between the pair of variables to understand the relationships.

In [45]:
```
# pairwise scatter plot of the dataset columns

sns.pairplot(diabetes_data_missing_imputed);
```



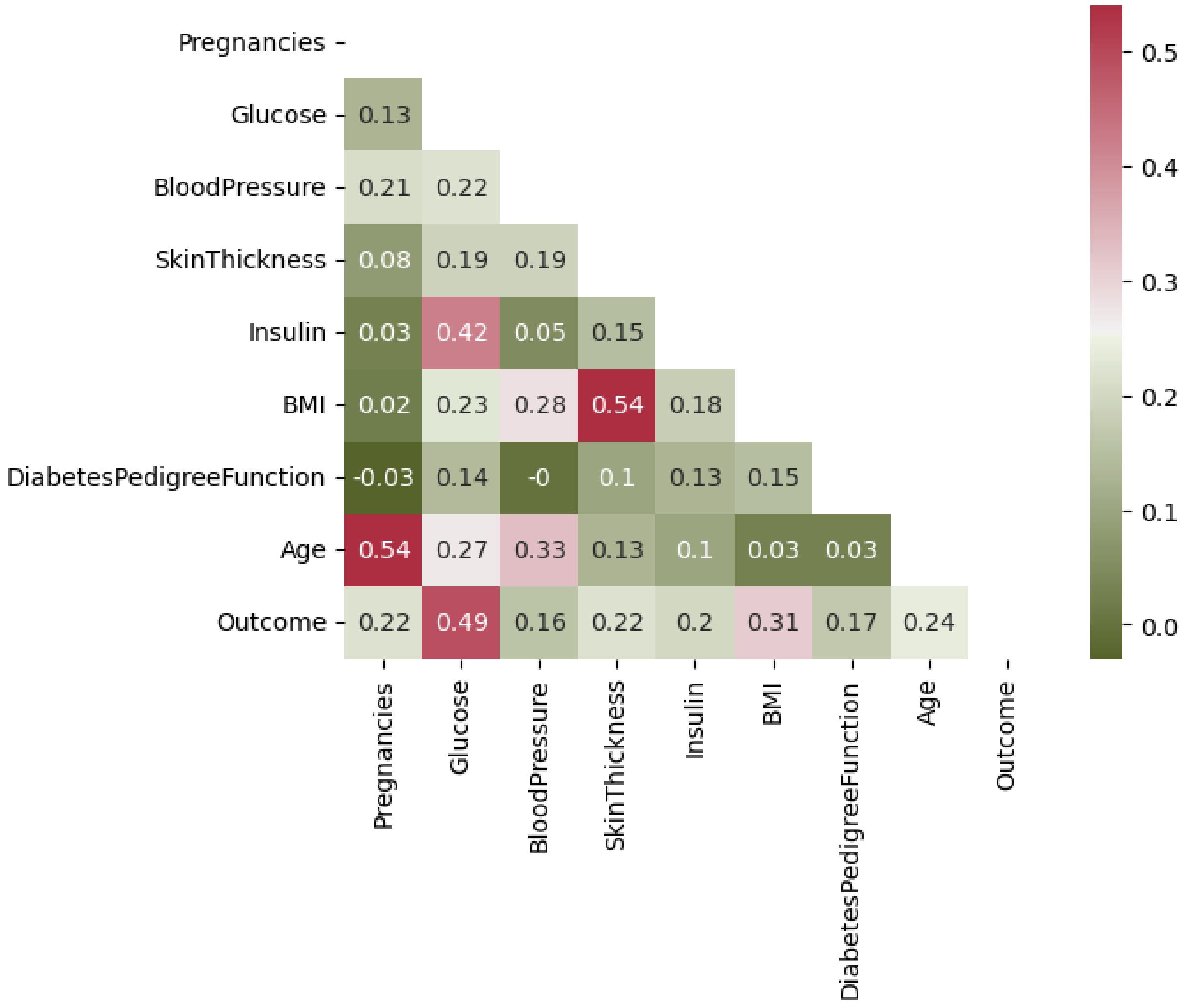
Task I (g): Correlation analysis using Heatmap

In [46]:
```
# getting correlation of the data columns with each other

corr_data = diabetes_data_missing_imputed.corr().round(2)
corr_data
```

Out[46]:

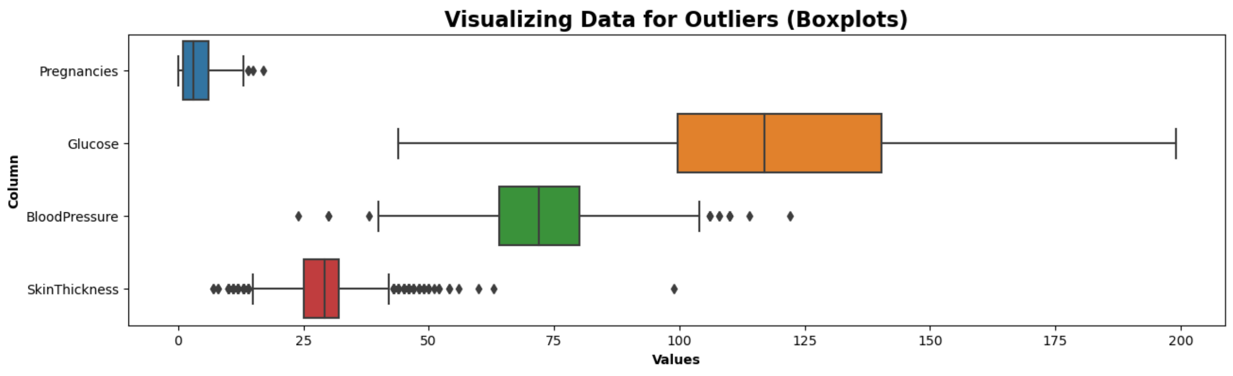
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
Pregnancies	1.00	0.13	0.21	0.08	0.03	0.02	
Glucose	0.13	1.00	0.22	0.19	0.42	0.23	
BloodPressure	0.21	0.22	1.00	0.19	0.05	0.28	
SkinThickness	0.08	0.19	0.19	1.00	0.15	0.54	
Insulin	0.03	0.42	0.05	0.15	1.00	0.18	
BMI	0.02	0.23	0.28	0.54	0.18	1.00	
DiabetesPedigreeFunction	-0.03	0.14	-0.00	0.10	0.13	0.15	
Age	0.54	0.27	0.33	0.13	0.10	0.03	
Outcome	0.22	0.49	0.16	0.22	0.20	0.31	

In [47]:
```
# Getting the Upper Triangle of the co-relation matrix
matrix = np.triu(np.ones_like(corr_data))

# Create a custom divergin palette
cmap = sns.diverging_palette(100, 7, s=75, l=40,
                            n=5, center="light", as_cmap=True)

# using the upper triangle matrix as mask
sns.heatmap(corr_data, annot=True, mask=matrix, cmap=cmap);
```



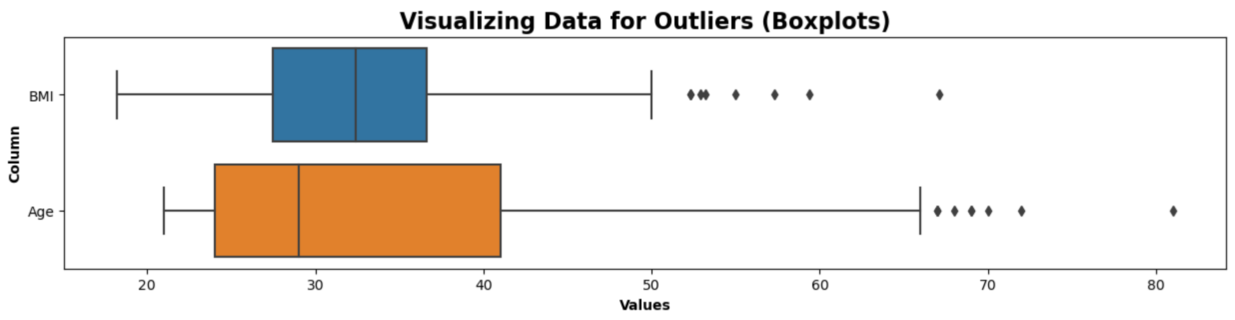
Task I (h): Outlier Detection and treatment

In [48]:
```
plt.figure(figsize=(15,4))

sns.boxplot(data = diabetes_data_missing_imputed[diabetes_data_missing_imputed.columns
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold"
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```

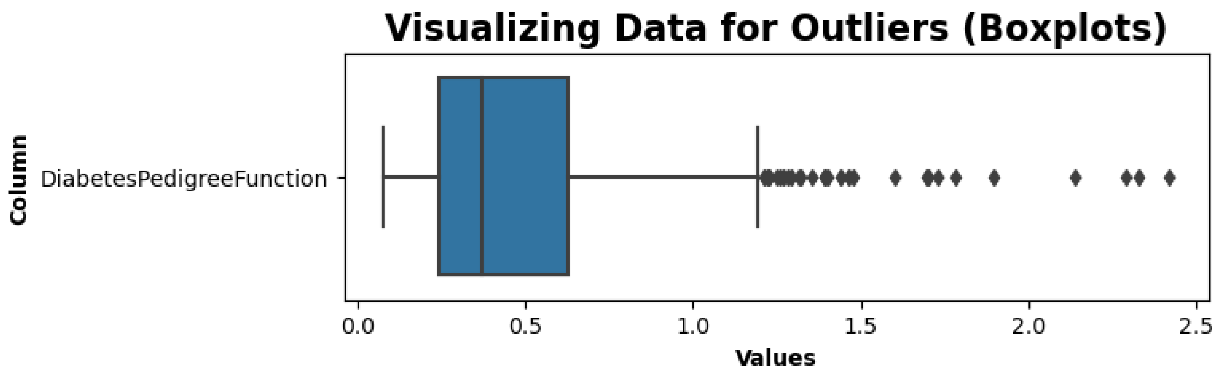


In [49]:
```
plt.figure(figsize=(15,3))

sns.boxplot(data = diabetes_data_missing_imputed[['BMI', 'Age']], orient='h');
```

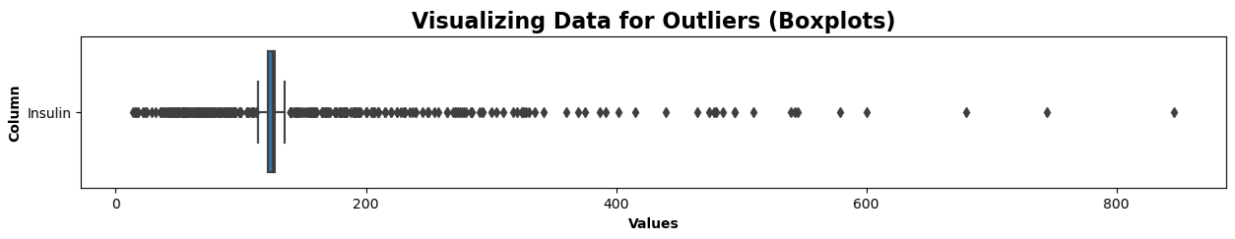
```
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold'
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



In [50]:
```
plt.figure(figsize=(7,2))

sns.boxplot(data = diabetes_data_missing_imputed[['DiabetesPedigreeFunction']], orient
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold'
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



In [51]:
```
plt.figure(figsize=(15,2))

sns.boxplot(data = diabetes_data_missing_imputed[['Insulin']], orient='h');
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold'
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



In [52]:
```
# this function takes the column and the dataframe and returns the total number of out
# based on IQR (i.e. based on 25th and 75th percentile values)

def find_outliers_stats(col, df):

    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)
    iqr = percentile75 - percentile25

    upper_limit = percentile75 + (1.5 * iqr)
    lower_limit = percentile25 - (1.5 * iqr)

    outliers_count = len(df[df[col] > upper_limit]) + len(df[df[col] < lower_limit])
    outliers_percent = round((outliers_count/df.shape[0])*100,2)

    return outliers_count, outliers_percent
```

In [53]:
```
diabetes_data_without_missing_vals.columns
```

Out[53]:
```
Index(['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
       'Pregnancies', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [54]:
```
# this function iterates over the columns of the passed df, and internally calls 'finc
# outlier count and percent. The reason that I have seperate function then 'find_outli
# it gives more freedom. If I want outlier count and percent of single column, 'find_c
# If I want it for the list of columns or for all the columns of passed dataframe, 'ou

def outlier_stats(df):

    outlier_stats = dict()

    for c in df.columns[:-1]:
        vals = find_outliers_stats(c, df)
        outlier_stats[c] = vals

    df_outlier_stats = pd.DataFrame(outlier_stats, index=['no_of_outliers', 'no_of_out
    return df_outlier_stats
```

In [55]:
```
# checking the outliers percent of the data for which missing values were "removed"

outlier_stats_of_data_with_no_missing_vals = outlier_stats(diabetes_data_without_missi
outlier_stats_of_data_with_no_missing_vals
```

Out[55]:

	no_of_outliers	no_of_outliers_percent
Glucose	0.0	0.00
BloodPressure	7.0	1.79
SkinThickness	1.0	0.26
Insulin	25.0	6.38
BMI	6.0	1.53
Pregnancies	11.0	2.81
DiabetesPedigreeFunction	12.0	3.06
Age	13.0	3.32

In [56]:
```
# checking the outliers percent of the data for which missing values were "imputed"

outlier_stats_after_missing_imputation = outlier_stats(diabetes_data_missing_imputed)
outlier_stats_after_missing_imputation
```

Out[56]:

	no_of_outliers	no_of_outliers_percent
Pregnancies	4.0	0.52
Glucose	0.0	0.00
BloodPressure	14.0	1.82
SkinThickness	87.0	11.33
Insulin	346.0	45.05
BMI	8.0	1.04
DiabetesPedigreeFunction	29.0	3.78
Age	9.0	1.17

In [57]:
```
# re-imputing the missing values of the 'Insulin' column with mean values of 'Insulin

col = 'Insulin'

diabetes_data_missing_imputed[col] = diabetes_data[col].apply(
                                                    lambda x: np.mean(diabetes_dat
```

In [58]:
```
# checking outliers stats after imputation

outlier_stats_after_missing_imputation = outlier_stats(diabetes_data_missing_imputed)
outlier_stats_after_missing_imputation
```

Out[58]:

	no_of_outliers	no_of_outliers_percent
Pregnancies	4.0	0.52
Glucose	0.0	0.00
BloodPressure	14.0	1.82
SkinThickness	87.0	11.33
Insulin	159.0	20.70
BMI	8.0	1.04
DiabetesPedigreeFunction	29.0	3.78
Age	9.0	1.17

In [59]:
```
# checking mean and standard deviation of 'Insulin' column from the data where missing

np.mean(diabetes_data_without_missing_vals['Insulin']), np.std(diabetes_data_without_m
```

Out[59]:
```
(156.05612244897958, 118.69000917870957)
```

In [60]:
```
# checking the value of mean + standard deviation of 'Insulin' column from the data wh

np.mean(diabetes_data_without_missing_vals['Insulin']) + np.std(diabetes_data_without_
```

Out[60]:
```
274.7461316276891
```

In [61]:
```
# checking the value of mean - standard deviation of 'Insulin' column from the data wh

np.mean(diabetes_data_without_missing_vals['Insulin']) - np.std(diabetes_data_without_
```
Out[61]:
```
37.366113270270006
```

In [62]:
```
# re-imputation of Insulin column with impute_value (mean of 'Insulin' - standard devi

col = 'Insulin'

impute_value = np.mean(diabetes_data_without_missing_vals[col]) - np.std(diabetes_data

diabetes_data_missing_imputed[col] = diabetes_data[col].apply(
                                                        lambda x: impute_value if x ==
```

In [63]:
```
# checking outliers stats after imputation

outlier_stats_after_missing_imputation = outlier_stats(diabetes_data_missing_imputed)
outlier_stats_after_missing_imputation
```

Out[63]:

	no_of_outliers	no_of_outliers_percent
Pregnancies	4.0	0.52
Glucose	0.0	0.00
BloodPressure	14.0	1.82
SkinThickness	87.0	11.33
Insulin	54.0	7.03
BMI	8.0	1.04
DiabetesPedigreeFunction	29.0	3.78
Age	9.0	1.17

In [64]:
```
# re-imputation of Insulin column with impute_value (mean of 'Insulin' - standard devi

col = 'Insulin'

impute_value = np.mean(diabetes_data_without_missing_vals[col]) + np.std(diabetes_data

diabetes_data_missing_imputed[col] = diabetes_data[col].apply(
                                                        lambda x: impute_value if x ==
```

In [65]:
```
# checking outliers stats after imputation

outlier_stats_after_missing_imputation = outlier_stats(diabetes_data_missing_imputed)
outlier_stats_after_missing_imputation
```

Out[65]:

	no_of_outliers	no_of_outliers_percent
Pregnancies	4.0	0.52
Glucose	0.0	0.00
BloodPressure	14.0	1.82
SkinThickness	87.0	11.33
Insulin	9.0	1.17
BMI	8.0	1.04
DiabetesPedigreeFunction	29.0	3.78
Age	9.0	1.17

In [66]:

```
# fuction which accepts the column and dataframe and returns a dataframe with removed

def outliar_removal(col, df):

    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)

#     print(f'For column {col}:\n')
#     print(f'25th Percentile: {percentile25}')
#     print(f'75th Percentile: {percentile75}')

    iqr = percentile75 - percentile25

#     print(f'IQR: {iqr}\n')

    upper_limit = percentile75 + (1.5 * iqr)
    lower_limit = percentile25 - (1.5 * iqr)

    rows_before = df.shape[0]

    df = df[df[col] < upper_limit]
    df = df[df[col] > lower_limit]

    print(f'Column in consideration: {col}')
    print(f'Current number of rows: {rows_before}')
    print(f'Rows removed: {rows_before - df.shape[0]}')
    print(f'Rows removed (in %): { round(((rows_before - df.shape[0])/rows_before)*10

    return df
```

In [67]:

```
# removing the outliers of the columns one by one using loop via 'outliar_removal' fu

diabetes_data_missing_imputed_outlier_removed = diabetes_data_missing_imputed.copy(de

for col in diabetes_data_missing_imputed_outlier_removed.columns[:-1]:
    diabetes_data_missing_imputed_outlier_removed = outliar_removal(col, diabetes_data
Column in consideration: Pregnancies
Current number of rows: 768
Rows removed: 4
Rows removed (in %): 0.52


Column in consideration: Glucose
Current number of rows: 764
Rows removed: 0
Rows removed (in %): 0.0


Column in consideration: BloodPressure
Current number of rows: 764
Rows removed: 17
Rows removed (in %): 2.23


Column in consideration: SkinThickness
Current number of rows: 747
Rows removed: 85
Rows removed (in %): 11.38


Column in consideration: Insulin
Current number of rows: 662
Rows removed: 7
Rows removed (in %): 1.06


Column in consideration: BMI
Current number of rows: 655
Rows removed: 6
Rows removed (in %): 0.92


Column in consideration: DiabetesPedigreeFunction
Current number of rows: 649
Rows removed: 27
Rows removed (in %): 4.16


Column in consideration: Age
Current number of rows: 622
Rows removed: 9
Rows removed (in %): 1.45
```

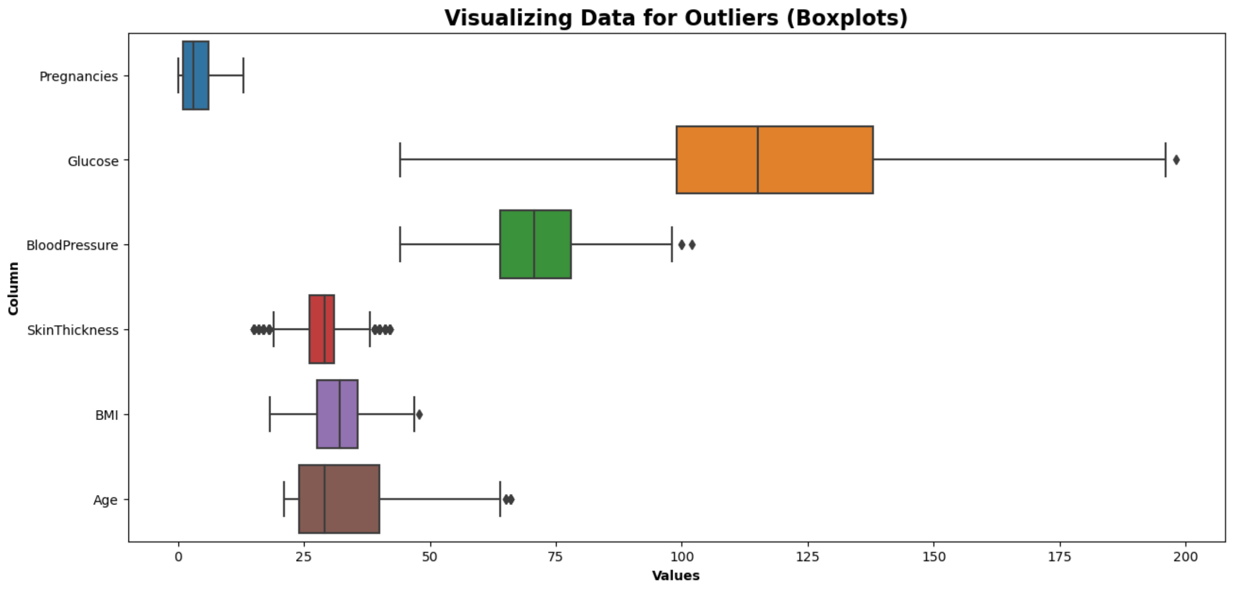
In [68]:
```
diabetes_data_missing_imputed_outlier_removed.columns
```

Out[68]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

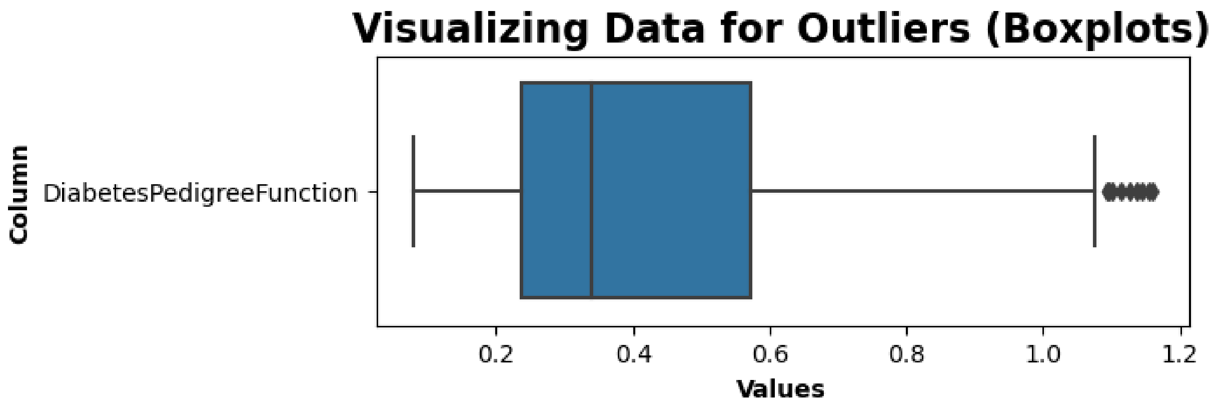
Visualizing columns through boxplots after removal of outliers

In [69]:
```
plt.figure(figsize=(15,7))

cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Age']
```

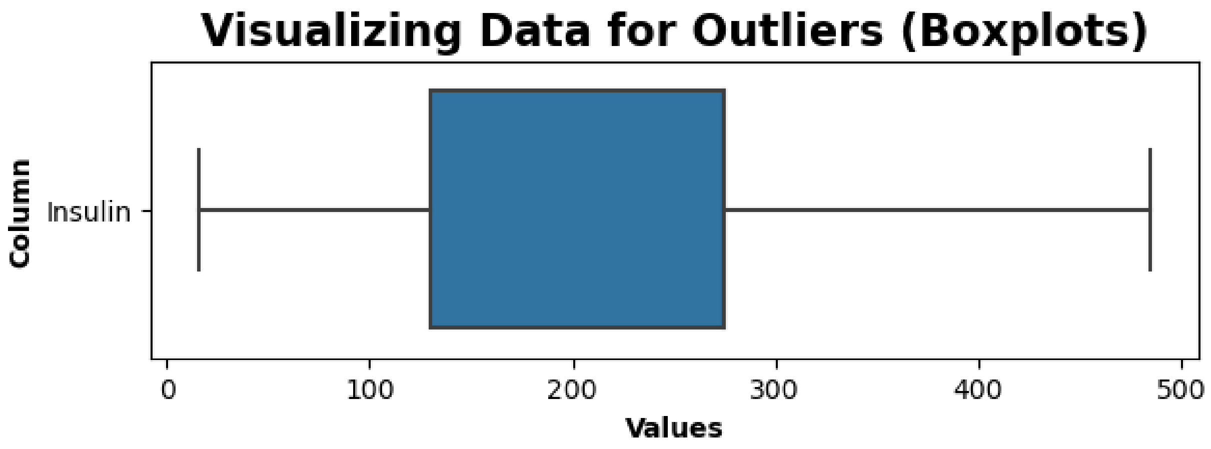
```
sns.boxplot(data = diabetes_data_missing_imputed_outlier_removed[cols], orient='h');
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold'
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



In [70]:
```
plt.figure(figsize=(6,2))

sns.boxplot(data = diabetes_data_missing_imputed_outlier_removed[['DiabetesPedigreeFur
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold'
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



In [71]:
```
plt.figure(figsize=(7,2))

sns.boxplot(data = diabetes_data_missing_imputed_outlier_removed[['Insulin']], orient=
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweight="bold'
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



**Analysis Summary*:

- From the pairwise scatter plot, following are the relationship among columns identified:

```
* Glucose with Insulin
* BMI with BloodPressure
* Pregnancies with Age
* BMI with SkinThickness
```

- The above relationship was further verified with Correlation Matrix as heatmap and was found that the following columns were related:

```
* Glucose with Insulin
* BMI with Skinthickness
* Pregnancies with Age
```

- However, because the values were less than 0.6, the above correlations were disregarded,
- It was found the except Glucose column, all the other independent variables had outliers present.
- Also, during reinspection of outliers (via values), it was observed that Insulin column had invariably large number of outliers present.
- So this Insulin column was reimputed with mean and standard deviation values. The appropriate value was settled at mean + standard deviation, as this gave minimum number of outliers.
- Therefore, the outliers were removed by keeping only the values within 25th and 75th percentile ± 1.5 times IQR values of the respective column

Task II: Data Modeling

Task II (a): Splitting of Data

In [72]:
```
X = diabetes_data_missing_imputed_outlier_removed.drop(['Outcome'], axis = 1)
y = diabetes_data_missing_imputed_outlier_removed['Outcome']
```

In [73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state
```

In [74]:
```
X_train.shape, X_test.shape
```

```
Out[74]: ((521, 8), (92, 8))
```

```
In [75]: y_train.shape, y_test.shape
```

```
Out[75]: ((521,), (92,))
```

Task II (b): Scaling of the Data

```
In [76]: X_train[X_train < 0].any().sum(), X_test[X_test < 0].any().sum()
```

```
Out[76]: (0, 0)
```

```
In [77]: cols_X = X_train.columns
```

```
In [78]: scaler = MinMaxScaler()

         X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=cols_X)
         X_test = pd.DataFrame(scaler.transform(X_test), columns=cols_X)
```

```
In [79]: scaler.get_feature_names_out()
```

```
Out[79]: array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
                'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'], dtype=object)
```

```
In [80]: scaler.get_params()
```

```
Out[80]: {'clip': False, 'copy': True, 'feature_range': (0, 1)}
```

```
In [81]: X_train.head()
```

Out[81]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	0.230769	0.857143	0.750000	0.444444	0.298507	0.508418	0.995375
1	0.000000	0.448052	0.642857	0.037037	0.551698	0.430976	0.736355
2	0.615385	0.136364	0.500000	0.296296	0.551698	0.464646	0.482886
3	0.076923	0.532468	0.285714	0.523904	0.551698	0.400673	0.250694
4	0.076923	0.175325	0.071429	0.111111	0.127932	0.074074	0.226642

```
In [82]: X_test.head()
```

Out[82]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	0.461538	0.675325	0.500000	0.740741	0.551698	0.518519	0.507863
1	0.461538	0.435065	0.357143	0.888889	0.551698	0.538721	0.168363
2	0.076923	0.733766	0.500000	0.222222	0.324094	0.249158	0.041628
3	0.692308	0.441558	0.678571	0.629630	0.339019	0.538721	0.168363
4	0.076923	0.474026	0.285714	0.296296	0.191898	0.525253	0.358927

Task II (c): Finding Principle Components to reduce dataset Dimention (PCA)

In [83]:
```
X_train_covariance_matrix = np.cov(X_train.T)
```

In [84]:
```
X_train_covariance_matrix.shape
```

Out[84]:
```
(8, 8)
```

In [85]:
```
eig_vals, eig_vecs = np.linalg.eig(X_train_covariance_matrix)
```

In [86]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)
```

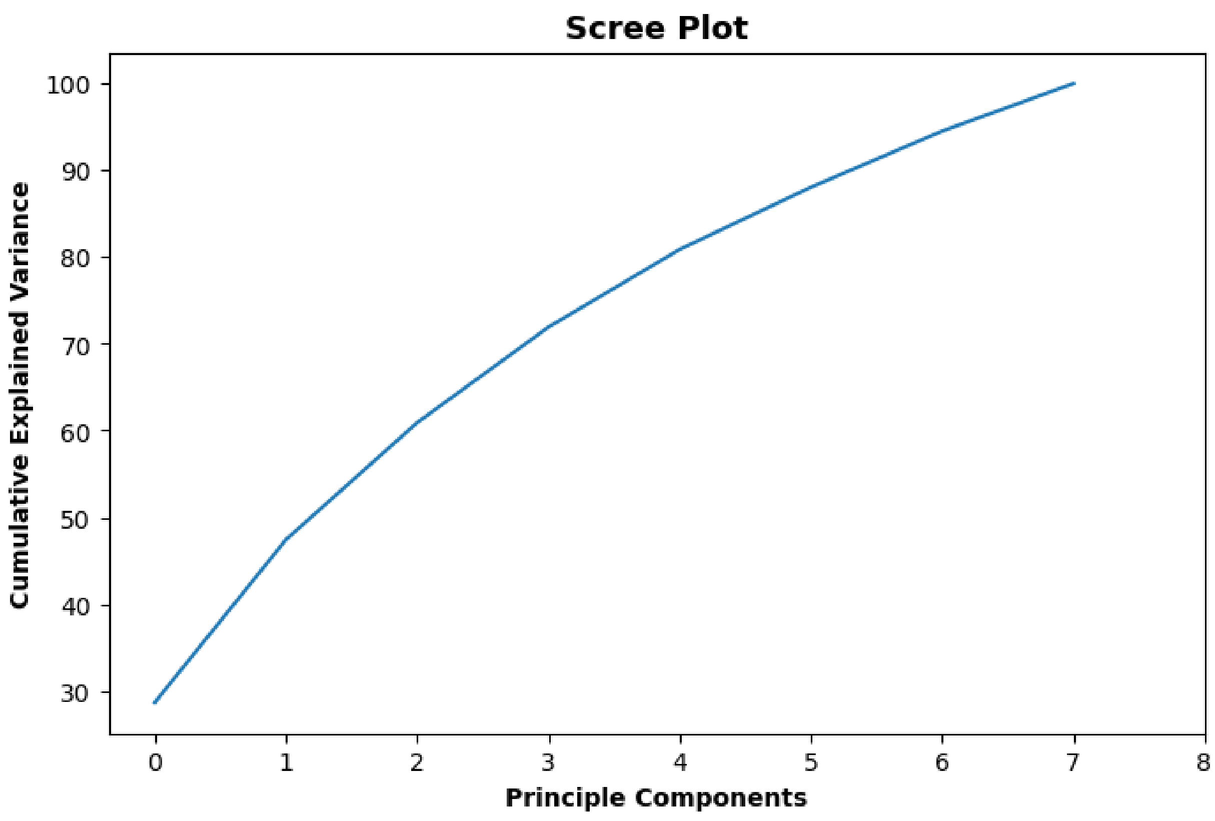
In [87]:
```
# Visually confirm that the list is correctly sorted by decreasing eigenvalues

print('Eigenvalues in descending order: \n')
for i in eig_pairs:   # first 5 values
    print(i[0])
```

```
Eigenvalues in descending order:

0.1071646092785274
0.07019661864203482
0.05038791668180087
0.04115853810931163
0.033448586737969944
0.026781469126898123
0.024184039973273588
0.02050363671641903
```

In [88]:
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]  # Variance capturec
cum_var_exp = np.cumsum(var_exp)

# Since the cumulative variance is in complex form,
cum_var_exp = np.round(np.real(cum_var_exp),2)

print("Cumulative variance: \n\n", cum_var_exp)
```

```
Cumulative variance:

 [ 28.67  47.44  60.92  71.93  80.88  88.05  94.52 100.  ]
```

In [89]:
```
plt.figure(figsize=(8,5))
sns.lineplot(x = range(X_train.shape[1]), y = cum_var_exp);
plt.title('Scree Plot', fontsize = 13, fontweight="bold")
plt.xlabel('Principle Components', fontweight="bold")
plt.ylabel('Cumulative Explained Variance', fontweight="bold")

plt.xticks(range(0, 9, 1))
#plt.yticks(range(0, 101, 10))

plt.show()
```



In [90]:
```
components = np.where(cum_var_exp>94)[0][0]  # First index of Principle component whic
components
```

Out[90]:
```
6
```

In [91]:
```
# defining PCA with components to capture 98% variance
pca = PCA(n_components=components+1)
```

In [92]:
```
X_train_pc = pca.fit_transform(X_train)
```

In [93]:
```
X_test_pc = pca.transform(X_test)
```

In [94]:
```
X_train.shape, X_train_pc.shape
```

Out[94]:
```
((521, 8), (521, 7))
```

In [95]:
```
X_test.shape, X_test_pc.shape
```

Out[95]:
```
((92, 8), (92, 7))
```

Analysis Summary:

- Dataset was splitted in training and testing set, in 85% and 15% ratio respectively.
- Splitted data was scaled using MinMax Scaler, because not all columns follow normal distribution
- Principle Component Analysis (PCA) was also performed in the dataset. It was observed that about 94% of the explained variance was captured by 7 columns, whereas the dataset has 8 columns (independent features)
- Therefore, in this particular dataset, the principle components aren't useful because there is no significant reduction of features observed.

Task II (d): Examining the appropriate (baseline) model using DABL

- Since this is a Supervised Classification problem, we will use classification Machine Learning Algorithms.
- As the Outcome (Target) variable is not balanced, it would be appropriate to use tree models or ensemble models.
- However, it is to good to test this dataset with DABL package, which can give us a rough estimate on how the dataset will perform on different models. From this baseline, ideas can be followed up to select the best model further.

In [96]:
```
ref_model = dabl.SimpleClassifier(random_state=0).fit(diabetes_data_missing_imputed_ou
ref_model
```

```
Running DummyClassifier()
accuracy: 0.672 average_precision: 0.328 roc_auc: 0.500 recall_macro: 0.500 f1_macro:
0.402
=== new best DummyClassifier() (using recall_macro):
accuracy: 0.672 average_precision: 0.328 roc_auc: 0.500 recall_macro: 0.500 f1_macro:
0.402

Running GaussianNB()
accuracy: 0.715 average_precision: 0.540 roc_auc: 0.743 recall_macro: 0.650 f1_macro:
0.655
=== new best GaussianNB() (using recall_macro):
accuracy: 0.715 average_precision: 0.540 roc_auc: 0.743 recall_macro: 0.650 f1_macro:
0.655

Running MultinomialNB()
accuracy: 0.669 average_precision: 0.491 roc_auc: 0.676 recall_macro: 0.561 f1_macro:
0.550
Running DecisionTreeClassifier(class_weight='balanced', max_depth=1)
accuracy: 0.632 average_precision: 0.438 roc_auc: 0.661 recall_macro: 0.661 f1_macro:
0.620
=== new best DecisionTreeClassifier(class_weight='balanced', max_depth=1) (using reca
ll_macro):
accuracy: 0.632 average_precision: 0.438 roc_auc: 0.661 recall_macro: 0.661 f1_macro:
0.620

Running DecisionTreeClassifier(class_weight='balanced', max_depth=5)
accuracy: 0.685 average_precision: 0.558 roc_auc: 0.729 recall_macro: 0.693 f1_macro:
0.670
=== new best DecisionTreeClassifier(class_weight='balanced', max_depth=5) (using reca
ll_macro):
accuracy: 0.685 average_precision: 0.558 roc_auc: 0.729 recall_macro: 0.693 f1_macro:
0.670

Running DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.01)
accuracy: 0.710 average_precision: 0.577 roc_auc: 0.744 recall_macro: 0.709 f1_macro:
0.689
=== new best DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.
01) (using recall_macro):
accuracy: 0.710 average_precision: 0.577 roc_auc: 0.744 recall_macro: 0.709 f1_macro:
0.689

Running LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000)
accuracy: 0.736 average_precision: 0.706 roc_auc: 0.828 recall_macro: 0.726 f1_macro:
0.713
=== new best LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000) (using
recall_macro):
accuracy: 0.736 average_precision: 0.706 roc_auc: 0.828 recall_macro: 0.726 f1_macro:
0.713

Running LogisticRegression(class_weight='balanced', max_iter=1000)
accuracy: 0.727 average_precision: 0.707 roc_auc: 0.822 recall_macro: 0.716 f1_macro:
0.704

Best model:
LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000)
Best Scores:
accuracy: 0.736 average_precision: 0.706 roc_auc: 0.828 recall_macro: 0.726 f1_macro:
0.713
```

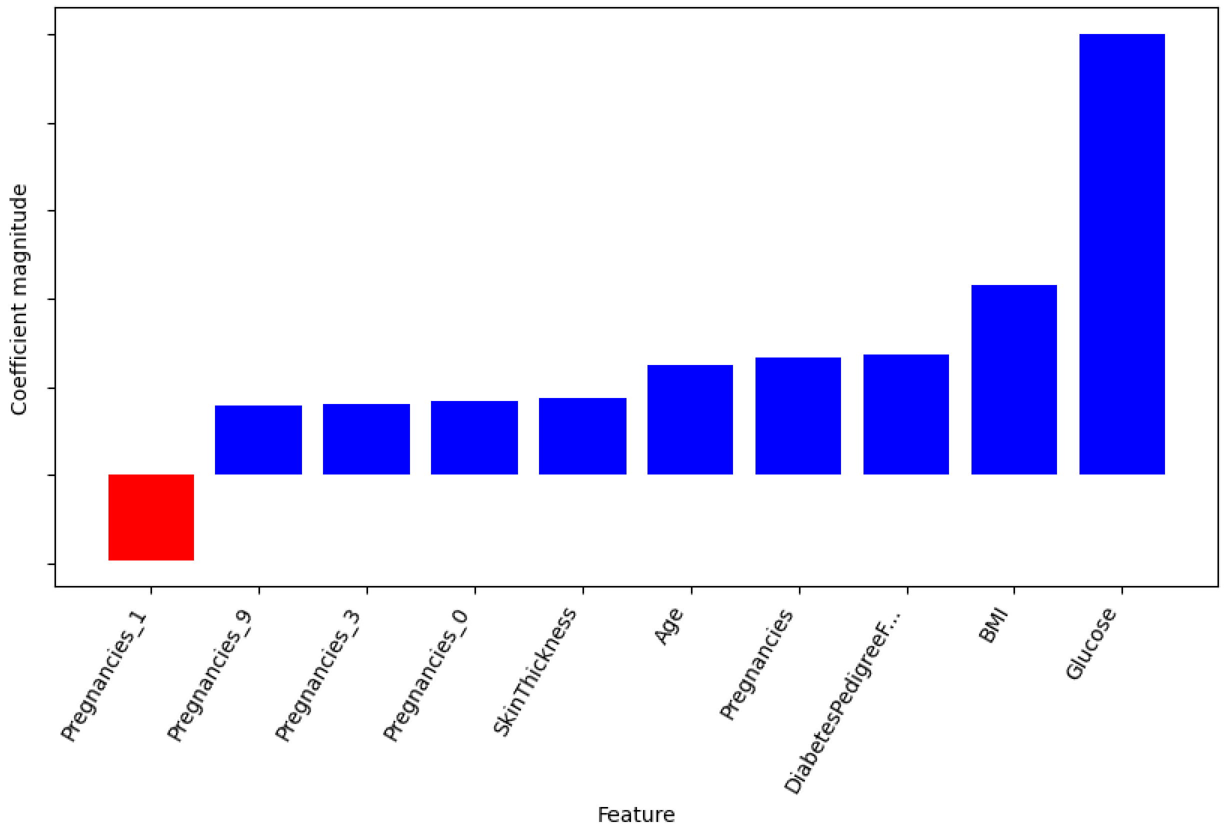
```
Out[96]:  ▾ SimpleClassifier
          SimpleClassifier(random_state=0)
```

```
In [97]:  ref_model.feature_names_
```

```
Out[97]:  Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
                 'BMI', 'DiabetesPedigreeFunction', 'Age'],
                dtype='object')
```

```
In [98]:  dabl.explain(ref_model)
```

```
C:\Users\Lenovo\.conda\envs\machine_learning\lib\site-packages\sklearn\utils\deprecat
ion.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out ins
tead.
  warnings.warn(msg, category=FutureWarning)
C:\Users\Lenovo\.conda\envs\machine_learning\lib\site-packages\dabl\plot\utils.py:37
8: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels(
```



```
In [99]:  # checking the reference model for the data where missing values were removed instead

          ref_model_no_missing_df = dabl.SimpleClassifier(random_state=0).fit(diabetes_data_witl
          ref_model_no_missing_df
```

```
Running DummyClassifier()
accuracy: 0.668 average_precision: 0.332 roc_auc: 0.500 recall_macro: 0.500 f1_macro:
0.401
=== new best DummyClassifier() (using recall_macro):
accuracy: 0.668 average_precision: 0.332 roc_auc: 0.500 recall_macro: 0.500 f1_macro:
0.401

Running GaussianNB()
accuracy: 0.633 average_precision: 0.489 roc_auc: 0.671 recall_macro: 0.575 f1_macro:
0.543
=== new best GaussianNB() (using recall_macro):
accuracy: 0.633 average_precision: 0.489 roc_auc: 0.671 recall_macro: 0.575 f1_macro:
0.543

Running MultinomialNB()
accuracy: 0.714 average_precision: 0.542 roc_auc: 0.676 recall_macro: 0.627 f1_macro:
0.633
=== new best MultinomialNB() (using recall_macro):
accuracy: 0.714 average_precision: 0.542 roc_auc: 0.676 recall_macro: 0.627 f1_macro:
0.633

Running DecisionTreeClassifier(class_weight='balanced', max_depth=1)
accuracy: 0.763 average_precision: 0.545 roc_auc: 0.753 recall_macro: 0.753 f1_macro:
0.741
=== new best DecisionTreeClassifier(class_weight='balanced', max_depth=1) (using reca
ll_macro):
accuracy: 0.763 average_precision: 0.545 roc_auc: 0.753 recall_macro: 0.753 f1_macro:
0.741

Running DecisionTreeClassifier(class_weight='balanced', max_depth=5)
accuracy: 0.750 average_precision: 0.572 roc_auc: 0.757 recall_macro: 0.751 f1_macro:
0.732
Running DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.01)
accuracy: 0.752 average_precision: 0.591 roc_auc: 0.777 recall_macro: 0.763 f1_macro:
0.738
=== new best DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.
01) (using recall_macro):
accuracy: 0.752 average_precision: 0.591 roc_auc: 0.777 recall_macro: 0.763 f1_macro:
0.738

Running LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000)
accuracy: 0.760 average_precision: 0.736 roc_auc: 0.848 recall_macro: 0.749 f1_macro:
0.739
Running LogisticRegression(class_weight='balanced', max_iter=1000)
accuracy: 0.768 average_precision: 0.707 roc_auc: 0.825 recall_macro: 0.749 f1_macro:
0.742

Best model:
DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.01)
Best Scores:
accuracy: 0.752 average_precision: 0.591 roc_auc: 0.777 recall_macro: 0.763 f1_macro:
0.738
```

```
Out[99]: ▾ SimpleClassifier
SimpleClassifier(random_state=0)
```

Task II (e): Running appropriate (multiple) Classification Models on the Data to determine the best model

In [100…

```
# custom function which accepts the ML model and the spittled data, and returns the tr
# this fuction is quick way to getting the accuracies, just by creating the model and

def ML_model_classifier(model, X_train, X_test, y_train, y_test, verbose=0):

    clf.fit(X_train, y_train)

    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)

    train_acc = round((accuracy_score(y_train, y_pred_train)*100),2)
    test_acc = round((accuracy_score(y_test, y_pred_test)*100),2)

    if verbose:
        print(f'Model: {model}')
        print(f'Training Accuracy: {train_acc}%')
        print(f'Test Accuracy: {test_acc}%')

    return model, train_acc, test_acc
```

In [101…

```
# Running the data with LogisticRegression

clf = LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000)
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000)
Training Accuracy: 72.74%
Test Accuracy: 80.43%
```

In [102…

```
# Running the data with DecisionTreeClassifier

clf = DecisionTreeClassifier(class_weight='balanced')
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: DecisionTreeClassifier(class_weight='balanced')
Training Accuracy: 100.0%
Test Accuracy: 71.74%
```

In [103…

```
# Running the data with RandomForestClassifier

clf = RandomForestClassifier(n_estimators=200, class_weight='balanced')
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: RandomForestClassifier(class_weight='balanced', n_estimators=200)
Training Accuracy: 100.0%
Test Accuracy: 81.52%
```

In [104…

```
# Running the data with SVC

clf = SVC(kernel = 'linear',gamma = 'scale', shrinking = False)
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: SVC(kernel='linear', shrinking=False)
Training Accuracy: 77.74%
Test Accuracy: 77.17%
```

In [105…
```
# Running the data with XGBoost Classifier

xgb_classifier = xgb.XGBClassifier()
model, train_acc, test_acc = ML_model_classifier(xgb_classifier, X_train, X_test, y_tr
```

```
Model: XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None,
              enable_categorical=False, gamma=None, gpu_id=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, reg_alpha=None,
              reg_lambda=None, scale_pos_weight=None, subsample=None,
              tree_method=None, validate_parameters=None, verbosity=None)
Training Accuracy: 77.74%
Test Accuracy: 77.17%
```

In [106…
```
# Running the data with KNeighborsClassifier

clf = KNeighborsClassifier()
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: KNeighborsClassifier()
Training Accuracy: 83.49%
Test Accuracy: 77.17%
```

Analysis Summary:

- Running the dataset with DABL package gave a LogisticRegression as a baseline model to start with. However, DecisionTreeClassifier was nominated as the best model for dataset with principle components.
- LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, SVC, XGBClassifier and KNeighborsClassifier were implemented one by one and training & testing accuracies were compared.
- It was found that DecisionTreeClassifier, RandomForestClassifier were overfitting (Training Accuracy:100%, Testing accuracy at about 70% to 80%)
- LogisticRegression model underfits. (Training Accuracy: 72.74%, Test Accuracy: 80.43%)
- KNeighborsClassifier gave best accuracy on training data with relatively lower on testing data (Training Accuracy: 83.49%, Test Accuracy: 77.17%)
- SVC and XGBClassifier are the best classification models for this problem, and gave highest accuracies of 77% in both training and testing data with no overfitting.

Task II (f): KNN Model Analysis on training and testing data accuracy

In [107…
```
# Getting the training and test accuracies in a list for different values of K (of KNN

train_acc_list = []
test_acc_list = []

for n in range(5, 101,5):
    clf = KNeighborsClassifier(n_neighbors=n)
    model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
```

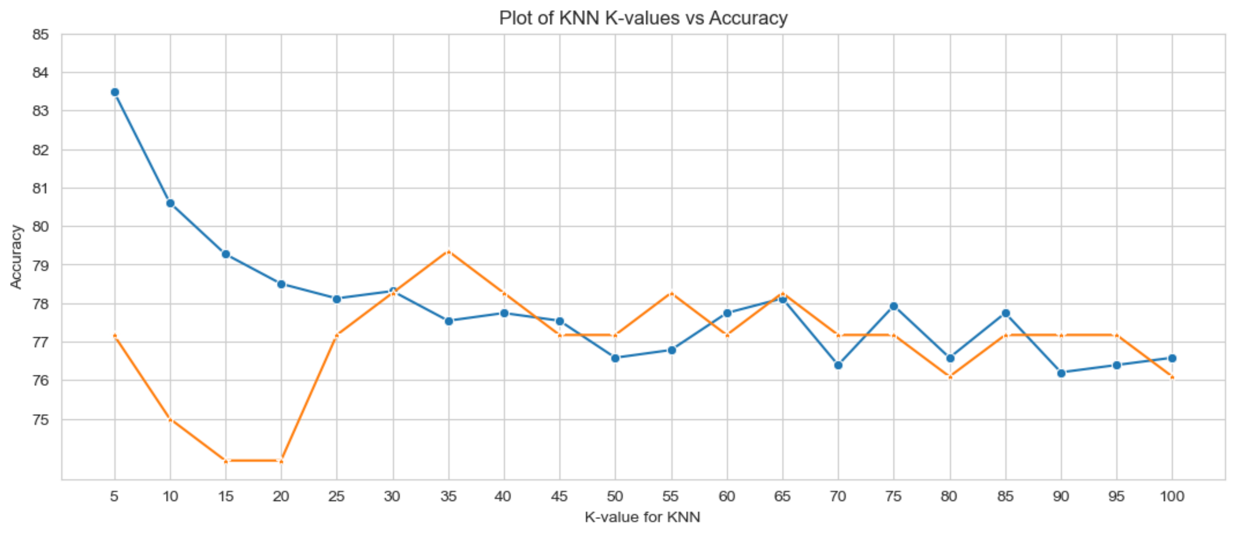
In [108…

```
print(train_acc_list[:10])
print(test_acc_list[:10])
```

```
[83.49, 80.61, 79.27, 78.5, 78.12, 78.31, 77.54, 77.74, 77.54, 76.58]
[77.17, 75.0, 73.91, 73.91, 77.17, 78.26, 79.35, 78.26, 77.17, 77.17]
```

In [109…

```
sns.set_style("whitegrid")

plt.figure(figsize=(13,5))
sns.lineplot(y = train_acc_list, x = range(5, 101,5), marker='o')
sns.lineplot(y = test_acc_list, x = range(5, 101,5), marker='*')
plt.xticks(range(5, 101,5))
plt.yticks(range(75,86,1))
plt.title("Plot of KNN K-values vs Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("K-value for KNN")
plt.show()
```



In [110…

```
knn_accuracy_df  = pd.DataFrame((train_acc_list, test_acc_list),
                                 index = ['knn_train_accuracy', 'knn_test_accuracy'],
                                 columns = range(5, 101,5))

knn_accuracy_df
```

Out[110]:

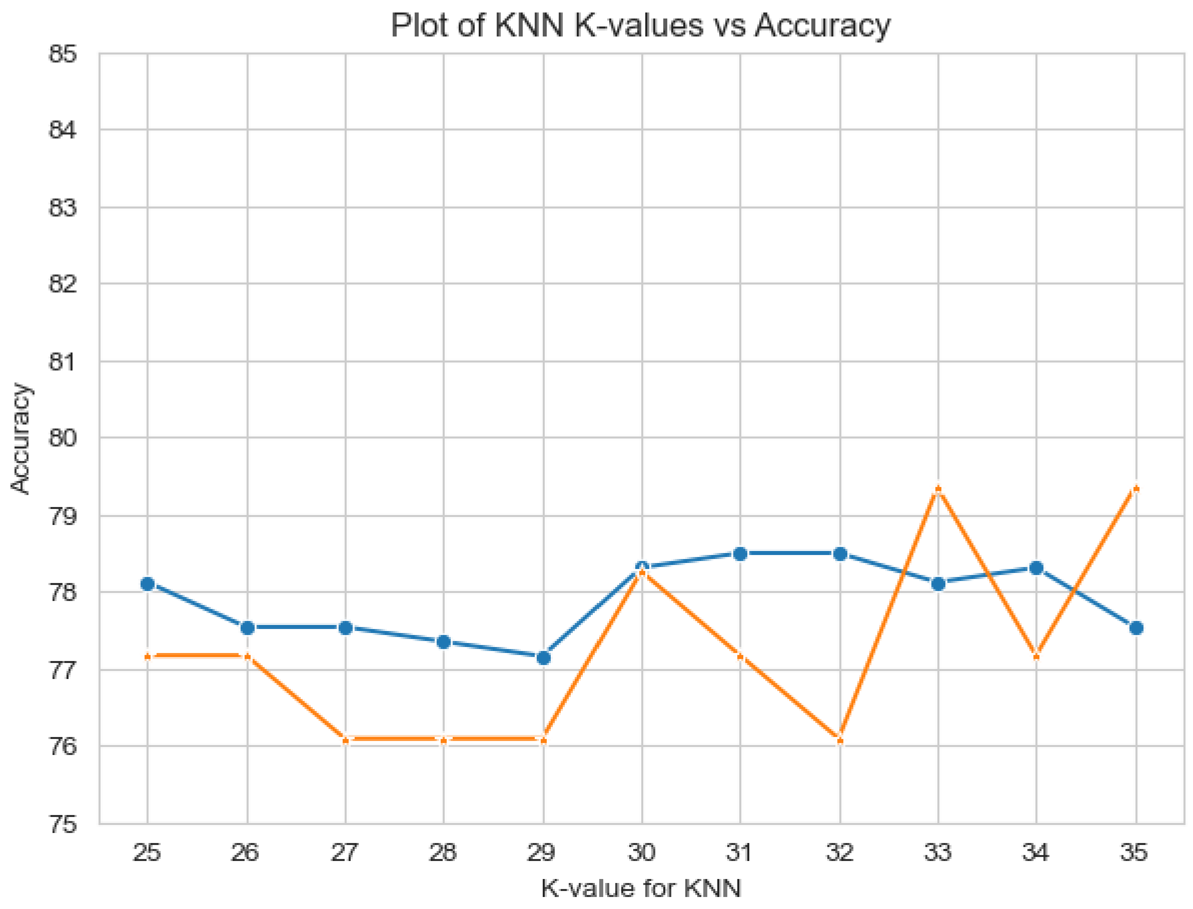
	5	10	15	20	25	30	35	40	45	50	55	60
knn_train_accuracy	83.49	80.61	79.27	78.50	78.12	78.31	77.54	77.74	77.54	76.58	76.78	77.74
knn_test_accuracy	77.17	75.00	73.91	73.91	77.17	78.26	79.35	78.26	77.17	77.17	78.26	77.17

In [111…

```
# Getting the training and test accuracies in a list for different values of K (of KNN

train_acc_list = []
test_acc_list = []

for n in range(25, 36,1):
    clf = KNeighborsClassifier(n_neighbors=n)
    model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
```

In [112…

```
sns.set_style("whitegrid")

plt.figure(figsize=(7,5))
sns.lineplot(y = train_acc_list, x = range(25, 36,1), marker='o')
sns.lineplot(y = test_acc_list, x = range(25, 36,1), marker='*')
plt.xticks(range(25, 36,1))
plt.yticks(range(75,86,1))
plt.title("Plot of KNN K-values vs Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("K-value for KNN")
plt.show()
```



In [113…

```
# re-running the classifier with k=30 (as in graph above, 30 neighbours gives optimum

clf = KNeighborsClassifier(n_neighbors=30)
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: KNeighborsClassifier(n_neighbors=30)
Training Accuracy: 78.31%
Test Accuracy: 78.26%
```

Task II (g): KNN Model Analysis on training and testing data accuracy (Principle Components)

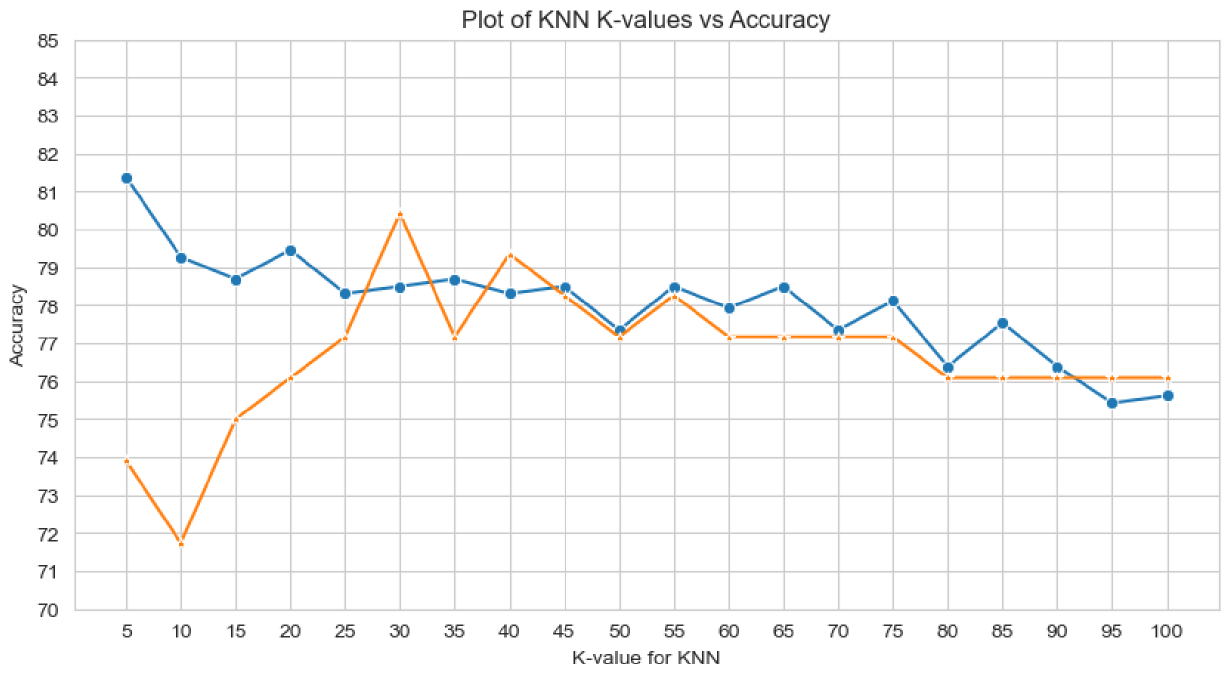
In [114…

```
train_acc_list_pc = []
test_acc_list_pc = []
```

```
for n in range(5, 101, 5):
    clf = KNeighborsClassifier(n_neighbors=n)
    model, train_acc, test_acc = ML_model_classifier(clf, X_train_pc, X_test_pc, y_tra
    train_acc_list_pc.append(train_acc)
    test_acc_list_pc.append(test_acc)
```

In [115…

```
plt.figure(figsize=(10,5))
sns.lineplot(y = train_acc_list_pc, x = range(5, 101, 5), marker='o')
sns.lineplot(y = test_acc_list_pc, x = range(5, 101, 5), marker='*')
plt.xticks(range(5, 101, 5))
plt.yticks(range(70,86,1))
plt.title("Plot of KNN K-values vs Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("K-value for KNN")
plt.show()
```



Analysis Summary:

- Since KNN model gave highest accuracy in training data, this was optimized for best K value using loops and graphs. (even though the highest accuracy on training data is given by random forest and decision tress, however, those model are overfitting)
- It was found that when k=30, the model gave its best accuracy on training and testing data of about 78% without overfitting.
- KNN model was also checked for its best values using 7 principle components, however, the training and testing accuracies of about 78% converged at k = 45. Moreover, there was no significant dimensionality reduction achieved with PCA, therefore it is disregarded in further metric calculations.

Task II (h): Hyperparameter Tuning of Random Forest using GridSearchCV

In [116…

```
rf = RandomForestClassifier(n_jobs=-1)

params = {
    'max_depth': [10, 20, 25, 30, 35, 40, 50, 100],
    'min_samples_leaf': [2,5,7,9,15, 25, 50],
    'n_estimators': [150, 200, 250, 300, 350, 400, 500, 700]
}
```

In [117…

```
# Instantiate the grid search model

grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 10,
                           n_jobs=-1, verbose=2, scoring="accuracy", return_train_scor
```

In [118…

```
grid_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 448 candidates, totalling 4480 fits
```

Out[118]:

```
▸            GridSearchCV
▸ estimator: RandomForestClassifier
        ▸ RandomForestClassifier
```

In [119…

```
grid_search.cv_results_.keys()
```

Out[119]:

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'par
am_max_depth', 'param_min_samples_leaf', 'param_n_estimators', 'params', 'split0_test
_score', 'split1_test_score', 'split2_test_score', 'split3_test_score', 'split4_test_
score', 'split5_test_score', 'split6_test_score', 'split7_test_score', 'split8_test_s
core', 'split9_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score',
'split0_train_score', 'split1_train_score', 'split2_train_score', 'split3_train_scor
e', 'split4_train_score', 'split5_train_score', 'split6_train_score', 'split7_train_s
core', 'split8_train_score', 'split9_train_score', 'mean_train_score', 'std_train_sco
re'])
```

In [120…

```
grid_search.cv_results_['mean_train_score'][:10]
```

Out[120]:

```
array([0.95947962, 0.96012019, 0.96118537, 0.96097215, 0.96118583,
       0.96139859, 0.96140042, 0.95990742, 0.902324  , 0.89933711])
```

In [121…

```
grid_cv_scores = pd.DataFrame([grid_search.cv_results_['mean_train_score'], grid_searc
grid_cv_scores.columns = ['mean_train_score', 'mean_test_score']
grid_cv_scores.head(7)
```

Out[121]:

	mean_train_score	mean_test_score
0	0.959480	0.765965
1	0.960120	0.767816
2	0.961185	0.765893
3	0.960972	0.767816
4	0.961186	0.765893
5	0.961399	0.765856
6	0.961400	0.767816

In [122…

```
grid_search.best_score_
```

Out[122]:

```
0.7851596516690856
```

In [123…

```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'max_depth': 100, 'min_samples_leaf': 15, 'n_estimators': 700}
```

In [124…

```
rf_best = grid_search.best_estimator_
rf_best
```

Out[124]:

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=100, min_samples_leaf=15, n_estimators=700,
                       n_jobs=-1)
```

Task II (i): Using best model obtained from Gridsearch for predictions

In [125…

```
rf_best.fit(X_train, y_train)

y_pred_train = rf_best.predict(X_train)
y_pred_test = rf_best.predict(X_test)
```

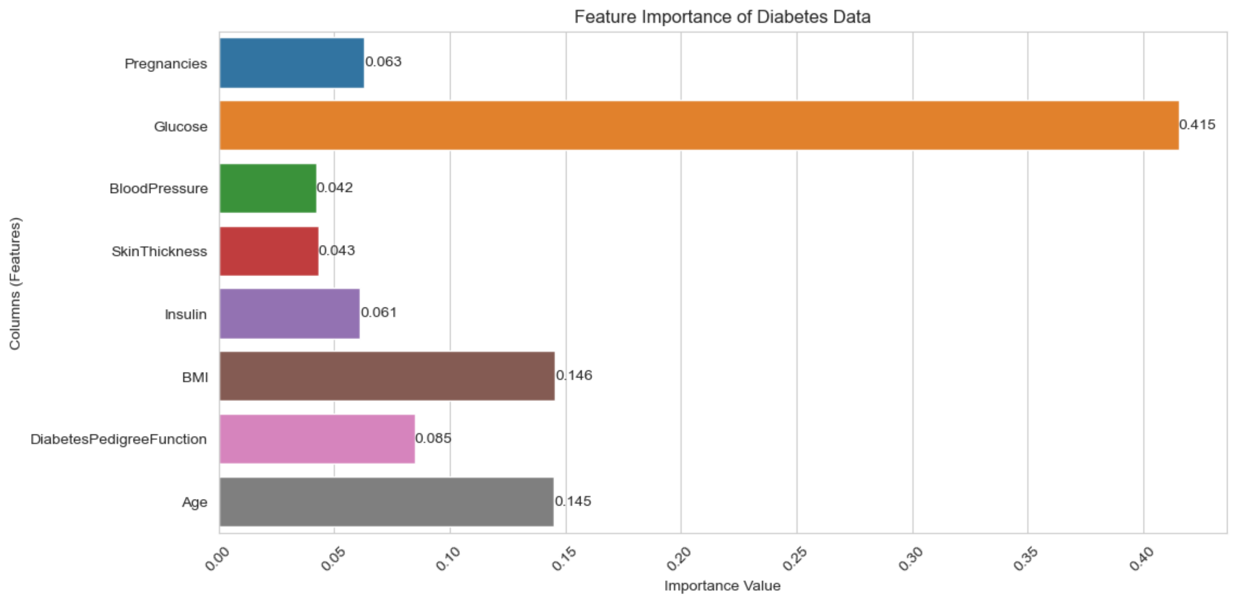
In [126…

```
print(f'Training Accuracy: {round((accuracy_score(y_train, y_pred_train)*100),2)}%')
print(f'Test Accuracy: {round((accuracy_score(y_test, y_pred_test)*100),2)}%')
```

```
Training Accuracy: 81.77%
Test Accuracy: 73.91%
```

In [127…

```
# Visualizing feature importance of diabetes data using random forest

plt.figure(figsize=(12,6))
ax = sns.barplot(y = diabetes_data_missing_imputed_outlier_removed.columns[:-1], x = r
ax.bar_label(ax.containers[0], fmt='%.3f')
plt.title("Feature Importance of Diabetes Data")
plt.xlabel("Importance Value")
plt.ylabel("Columns (Features)")
plt.xticks(rotation=45)
plt.show()
```



Analysis Summary:

- GridSearchCV was used to find optimum parameters for RandomForestClassifier
- GridSearchCV returned the tuned RandomForestClassifier model which has Training Training Accuracy: 81.19% Test Accuracy: 73.91%
- This model even though has best Training accuracy but still it is overfitting.
- Feature importance was extracted from Random Forest and it was identified that 'Glucose' is the most important feature influencing the target, followed by 'Age' and 'BMI'.

Task II (j): Final Model (KNN) Analysis of Performance

In [128…
```
# Best model obtained was KNN with neighbors = 30

clf = KNeighborsClassifier(n_neighbors=30)
model, train_acc, test_acc = ML_model_classifier(clf, X_train, X_test, y_train, y_test
```

```
Model: KNeighborsClassifier(n_neighbors=30)
Training Accuracy: 78.31%
Test Accuracy: 78.26%
```

In [129…
```
# Getting training and test predictions

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

In [130…
```
y_pred_test[:10]
```

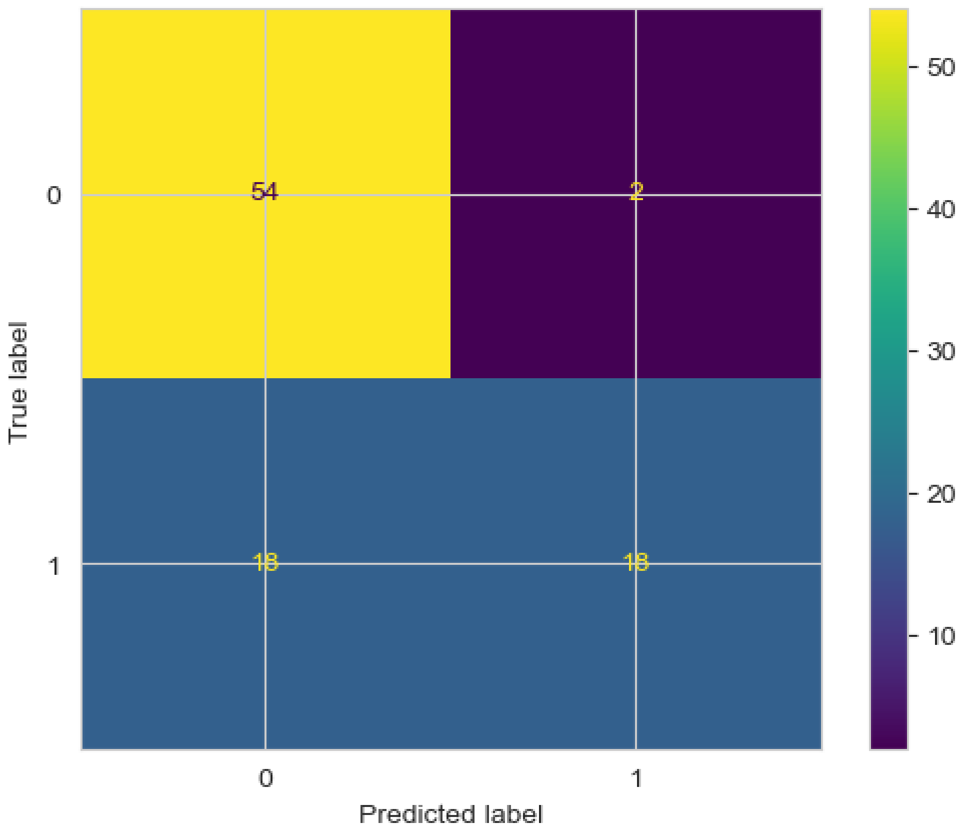
Out[130]:
```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

In [131…
```
print(confusion_matrix(y_test, y_pred_test))
```

```
[[54  2]
 [18 18]]
```

In [132…
```
# Visualizing confusion matrix in a better way

cm = confusion_matrix(y_test, y_pred_test, labels=model.classes_)
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
disp.plot()
plt.show()
```



In [133…

```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.96      0.84        56
           1       0.90      0.50      0.64        36

    accuracy                           0.78        92
   macro avg       0.82      0.73      0.74        92
weighted avg       0.81      0.78      0.77        92
```

In [134…

```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test, labels=model.classes_).ravel()
specificity = tn / (tn+fp)
sensitivity = tp / (tp+fn)
```

In [135…

```
print(f"True Negative: {tn}")
print(f"False Positive: {fp}")
print(f"False Negative: {fn}")
print(f"True Positive: {tp}")
```

```
True Negative: 54
False Positive: 2
False Negative: 18
True Positive: 18
```

In [136…

```
print(f"Sensitivity/Recall: {round(sensitivity,2)}")
print(f"Specificity: {round(specificity,2)}")
```

```
Sensitivity/Recall: 0.5
Specificity: 0.96
```

In [137…

```
# Getting predicted probabilites values, for plotting of ROC curve

knn_probs = model.predict_proba(X_test)[:, 1]
```

In [138…

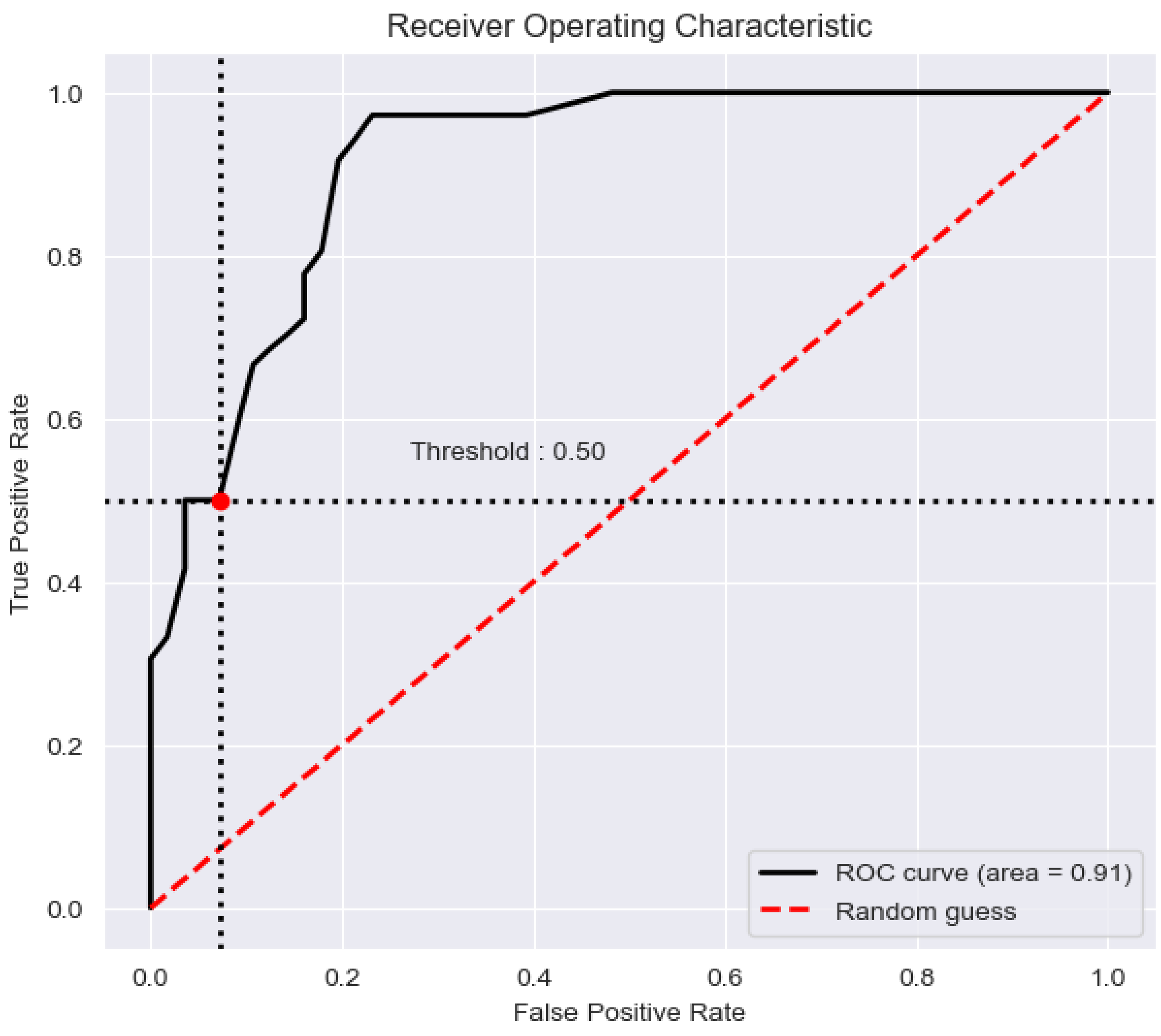
```
knn_probs[:10]
```

Out[138]:

```
array([0.6       , 0.23333333, 0.23333333, 0.33333333, 0.1       ,
       0.43333333, 0.33333333, 0.66666667, 0.1       , 0.23333333])
```

In [139…

```
# plotting ROC Curve

# Visualisation with plot_metric
bc = BinaryClassification(y_test, knn_probs, labels=["Class 1", "Class 2"])

# Figures
plt.figure(figsize=(7,6))
bc.plot_roc_curve()
plt.show()
```



Analysis Summary:

- Overall, the final best model is KNN with K=30.

  - Training Accuracy: 78.31%
  - Test Accuracy: 78.26%
- Following are the confusion metrics obtained on test data:
  - True Negative: 54
  - False Positive: 2
  - False Negative: 18
  - True Positive: 18
  - Sensitivity/Recall: 0.5
  - Specificity: 0.96
  - ROC Curve Area: 0.91

Task II (k): Checking the Accuracy of KNN with Cross Validation.

In [140…

```
%%time

# Not defining model again because we already have it in 'model' variable

k_folds = KFold(n_splits = 10)

scores = cross_val_score(model, X, y, cv = k_folds)
```

```
CPU times: total: 234 ms
Wall time: 226 ms
```

In [141…

```
print("Cross Validation Scores: \n", scores)
print(f"\nAverage CV Score: {round(scores.mean()*100,2)}%")
print("\nNumber of CV Scores used in Average: ", len(scores))
```

```
Cross Validation Scores: 
 [0.69354839 0.80645161 0.72580645 0.60655738 0.70491803 0.73770492
 0.80327869 0.80327869 0.75409836 0.78688525]

Average CV Score: 74.23%

Number of CV Scores used in Average:  10
```

Task II (l): Predicition of test datapoints and comparison with actual datapoints

In [142…

```
# since predicted test values were in numpy array type, converting it to series type

y_pred_test = pd.Series(y_pred_test, index=y_test.index)
```

In [143…

```
type(y_test), type(y_pred_test)
```

Out[143]:

```
(pandas.core.series.Series, pandas.core.series.Series)
```

In [144…

```
# preparing a dataframe to get th actual values and predicted values side by side

actual_pred_comparison = pd.DataFrame([y_test, y_pred_test], index=['actual_outcomes',
actual_pred_comparison.head(6)
```

Out[144]:

	actual_outcomes	predicted_outcomes
0	1	1
121	0	0
325	0	0
214	1	0
651	0	0
345	0	0

In [145…

```
# exporting the actual vs predicted values comparison dataframe as csv file (for repor

actual_pred_comparison.to_csv('actual_pred_comparison.csv', index = False)
```

Task III: Tableau Report

Task III (a): Exporting treated data for Tableau Analysis

In [146…

```
diabetes_data_missing_imputed_outlier_removed.to_csv("diabetes_data_for_tableau_report
```

Task III (b): Link of Tableau Dashboard

Tableau Report Link - https://public.tableau.com/app/profile/lavkush.singh4748/viz/PCDS-DataScienceCapstoneTableauReport/ProportionofDiabeticPopulation

End of the Project