PROJECT 2 Problem statement Context:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Problem Statement:

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

Dataset Description:

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the

number of pregnancies the patient has had,

their BMI,

insulin level,

age, and

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

Approach:

Following pointers will be helpful to structure your findings.

Perform descriptive analysis. It is very important to understand the variables and corresponding values. We need to think through - Can minimum value of below listed columns be zero (0)? On these columns, a value of zero does not make sense and thus indicates missing value.

• Glucose

• BloodPressure

• SkinThickness

• Insulin

• BMI

How will you treat these values?

Visually explore these variable, you may need to look for the distribution of these variables using histograms. Treat the missing values accordingly.

We observe integer as well as float data-type of 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 actions.

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.

Devise strategies for model building. It is important to decide the right validation framework. Express your thought process. Would Cross validation be useful in this scenario?

Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN.

Create a classification report by analysing sensitivity, specificity, AUC(ROC curve) etc. Please try to be as descriptive as possible to explain what values of these parameter you settled for? any why?

Importing important libraries

In [1]:
```
# Importing important libaries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from pandas_profiling import ProfileReport
from dataprep.eda import create_diff_report,plot,plot_correlation,plot_missing,create_report
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
## Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.svm import SVC

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Import ROC curve function from metrics module
from sklearn.metrics import plot_roc_curve
## Model evaluators
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import learning_curve
from sklearn.metrics import explained_variance_score, make_scorer
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score
#Scalling
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
#Importing the data set
df=pd.read_csv(r"E:\data_sciene_course\simplilearn\Capstone project\my project\Project 2\Healthcare - Diabetes\health care diabetes.csv")
```

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
df.dtypes
```

Out[4]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [5]:
```
df.shape
```

Out[5]:
```
(768, 9)
```

In [6]:
```
#Separating the Dependent and Independent variable though loc
x=df.loc[:,"Pregnancies":"Age"]
y=df.loc[:,"Outcome":]
x.head(2),y.head(2)
```

Out[6]:
```
(   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
 0            6      148             72             35        0  33.6
 1            1       85             66             29        0  26.6

    DiabetesPedigreeFunction  Age
 0                     0.627   50
 1                     0.351   31  ,
    Outcome
 0        1
 1        0)
```

In [7]:
```
###Total data points in "x"
x.count().sum()
```

Out[7]:
```
6144
```

Task 3

We observe integer as well as float data-type of variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

In [8]:
```
#Total Interger data points at "x"
k=[]
for lebel, content in x.items():
    if pd.api.types.is_integer_dtype(content):
        k.append(x[lebel].count())
#Calculating the Nos of integer values
NosOfInter=pd.DataFrame(k).sum()
NosOfFloat=x.count().sum()-NosOfInter
# initialize list of lists
data = [['NosOfInter', pd.DataFrame(k).sum()[0]], ['NosOfFloat', (x.count().sum()-NosOfInter)[0]]]

# Create the pandas DataFrame
df_data=pd.DataFrame(data, columns = ['Data_Type', 'Count'])

print(NosOfInter[0])
print(NosOfFloat[0])
df_data
```

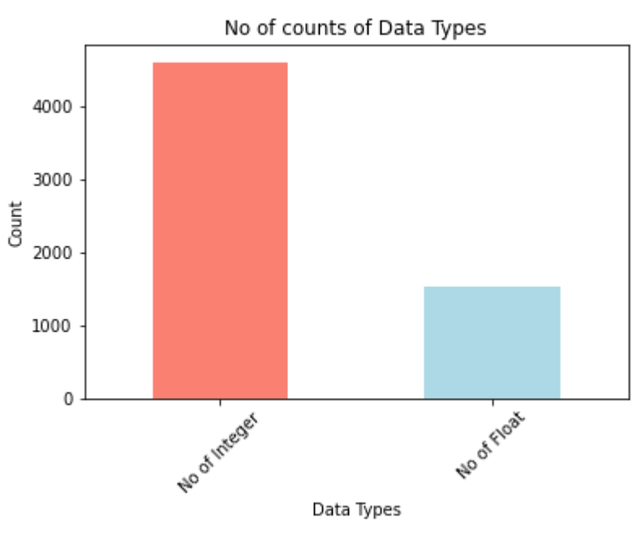
```
4608
1536
```

Out[8]:

	Data_Type	Count
0	NosOfInter	4608
1	NosOfFloat	1536

In [9]:
```
df_data.set_index("Data_Type",inplace=True)
```

In [10]:
```
figure, ax = plt.subplots(1,1)
df_data.Count.plot(kind="bar", color=["salmon", "lightblue"])
ax.set_xticklabels(("No of Integer","No of Float"))
plt.xlabel('Data Types')
plt.ylabel('Count')
plt.tick_params(axis="x", rotation=45)
plt.title("No of counts of Data Types");
```



Task 1

Perform descriptive analysis. It is very important to understand the variables and corresponding values. We need to think through - Can minimum value of below listed columns be zero (0)? On these columns, a value of zero does not make sense and thus indicates missing value. • Glucose

•BloodPressure

•SkinThickness

•Insulin

- BMI

How will you treat these values?

In [11]:
```
x.isnull().sum()
```

Out[11]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
dtype: int64
```

In [12]:
```
#Replacing 0 with NAN
x = x.drop("Pregnancies", axis=1).replace({'0':np.nan, 0:np.nan})
```

In [13]:
```
x["Pregnancies"]=df["Pregnancies"]
```

In [14]:
```
x.isnull().sum()
```

Out[14]:
```
Glucose                   5
BloodPressure            35
SkinThickness           227
Insulin                 374
BMI                      11
DiabetesPedigreeFunction  0
Age                       0
Pregnancies               0
dtype: int64
```

Task 2

Visually explore these variable, you may need to look for the distribution of these variables using histograms. Treat the missing values accordingly.

In [15]:
```
ProfileReport(x)
```

Overview

Dataset statistics

Number of variables	8
Number of observations	768
Missing cells	652
Missing cells (%)	10.6%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	48.1 KiB
Average record size in memory	64.2 B

Variable types

Numeric	8

Alerts

`Glucose` is highly correlated with `Insulin`	High correlation
`SkinThickness` is highly correlated with `BMI`	High correlation
`Insulin` is highly correlated with `Glucose`	High correlation
`BMI` is highly correlated with `SkinThickness`	High correlation
`Age` is highly correlated with `Pregnancies`	High correlation
`Pregnancies` is highly correlated with `Age`	High correlation
`Glucose` is highly correlated with `Insulin`	High correlation

Out[15]:

In [16]:
```
create_report(x)
```

Out[16]:

DataPrep Report | Overview | Variables ≡ | Interactions | Correlations | Missing Values

Overview

Dataset Statistics

Number of Variables	8
Number of Rows	768
Missing Cells	652
Missing Cells (%)	10.6%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	48.1 KB
Average Row Size in Memory	64.2 B
Variable Types	Numerical: 8

Dataset Insights

`BloodPressure` has 35 (4.56%) missing values	Missing
`SkinThickness` has 227 (29.56%) missing values	Missing
`Insulin` has 374 (48.7%) missing values	Missing
`BMI` has 11 (1.43%) missing values	Missing
`Age` is skewed	Skewed
`Pregnancies` is skewed	Skewed
`Pregnancies` has 111 (14.45%) zeros	Zeros

Variables

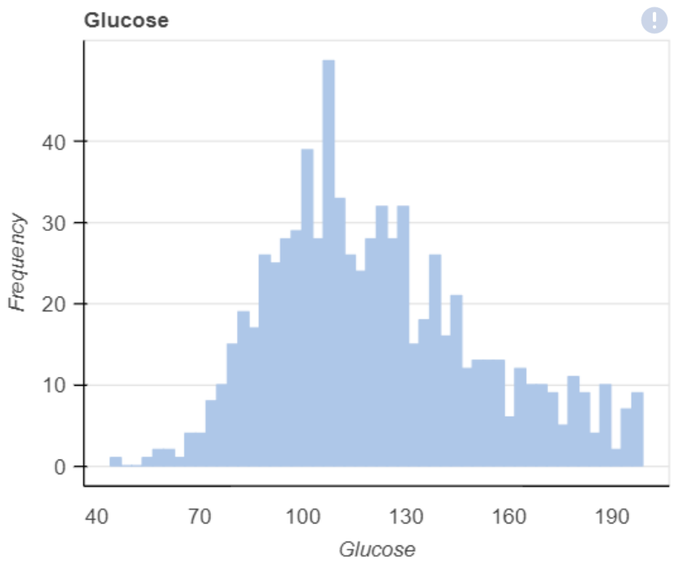
Sort by Feature order ☐ Reverse order

Glucose
numerical

Show Details

Approximate Distinct Count	135
Approximate Unique (%)	17.7%
Missing	5
Missing (%)	0.7%
Infinite	0
Infinite (%)	0.0%
Memory Size	11.9 KB

Mean	121.6868
Minimum	44
Maximum	199
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

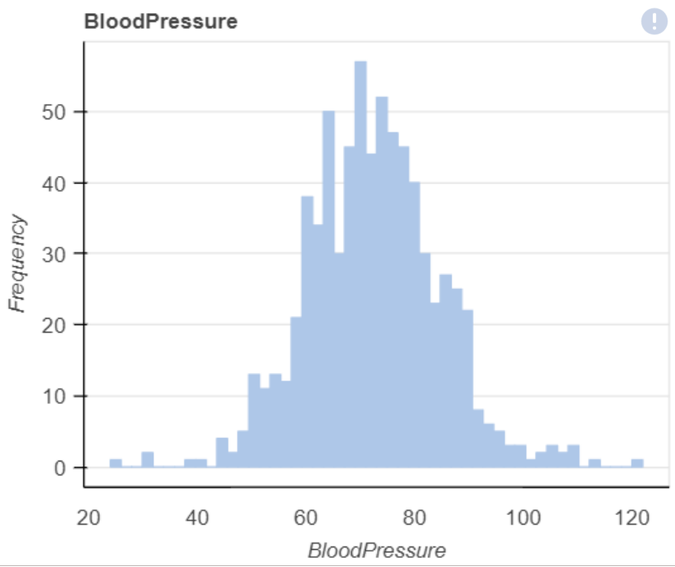


BloodPressure
numerical

Show Details

Approximate Distinct Count	46
Approximate Unique (%)	6.3%
Missing	35
Missing (%)	4.6%
Infinite	0
Infinite (%)	0.0%
Memory Size	11.5 KB

Mean	72.4052
Minimum	24
Maximum	122
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

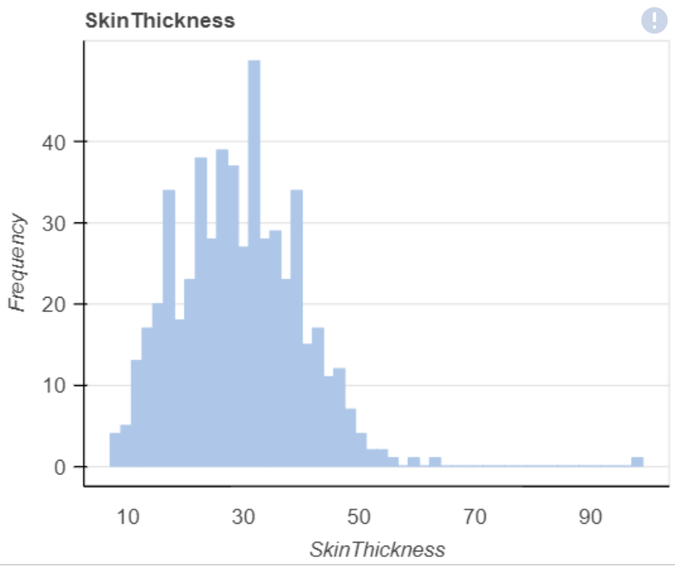


SkinThickness
numerical

Show Details

Approximate Distinct Count	50
Approximate Unique (%)	9.2%
Missing	227
Missing (%)	29.6%
Infinite	0
Infinite (%)	0.0%
Memory Size	8.5 KB

Mean	29.1534
Minimum	7
Maximum	99
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

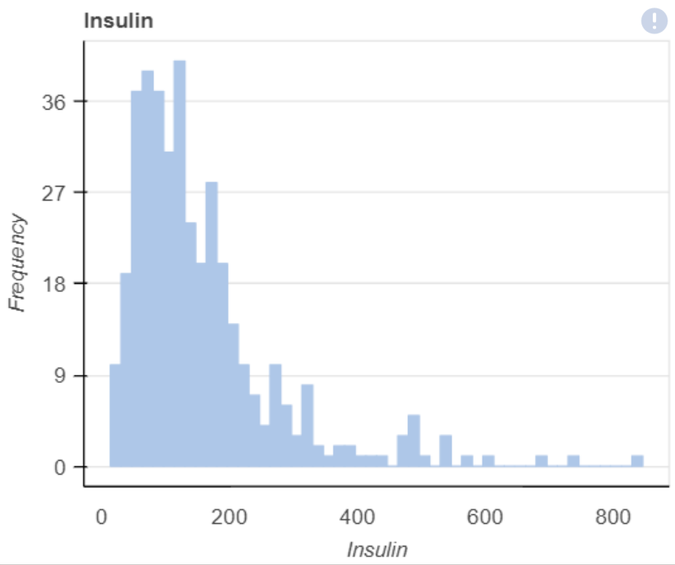


Insulin
numerical

Show Details

Approximate Distinct Count	185
Approximate Unique (%)	46.9%
Missing	374
Missing (%)	48.7%
Infinite	0
Infinite (%)	0.0%
Memory Size	6.2 KB

Mean	155.5482
Minimum	14
Maximum	846
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

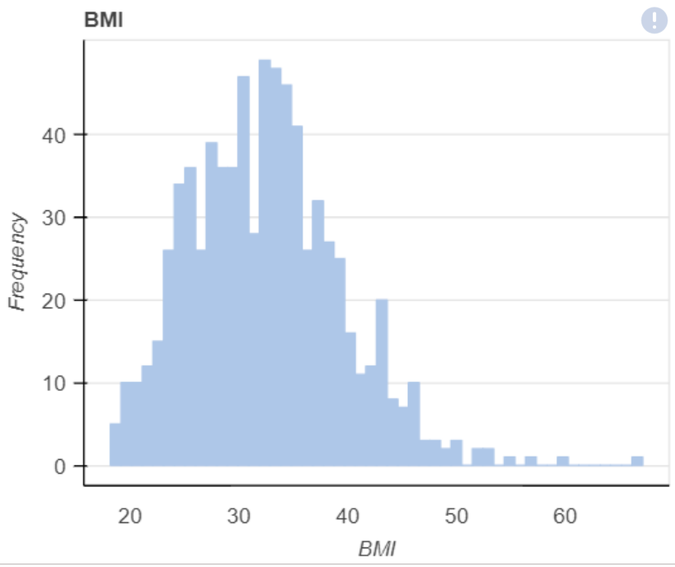


BMI
numerical

Show Details

Approximate Distinct Count	247
Approximate Unique (%)	32.6%
Missing	11
Missing (%)	1.4%
Infinite	0
Infinite (%)	0.0%
Memory Size	11.8 KB

Mean	32.4575
Minimum	18.2
Maximum	67.1
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

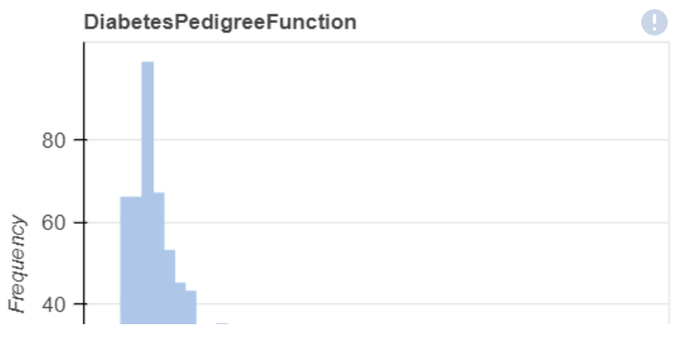


DiabetesPedigreeFun…
numerical

Show Details

Approximate Distinct Count	517
Approximate Unique (%)	67.3%
Missing	0
Missing (%)	0.0%
Infinite	0

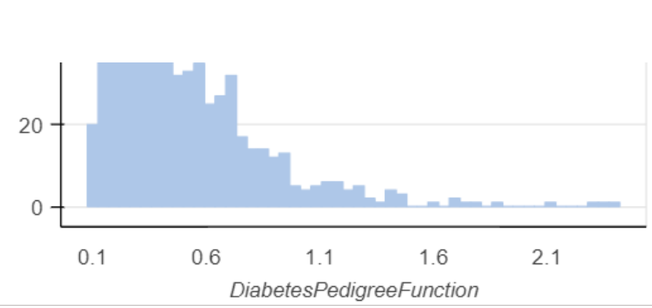
Memory Size	12.0 KB
Mean	0.4719
Minimum	0.078
Maximum	2.42
Zeros	0



Show Details

Infinite	0
Infinite (%)	0.0%
Negatives (%)	0.0%

Zeros (%)	0.0%
Negatives	0

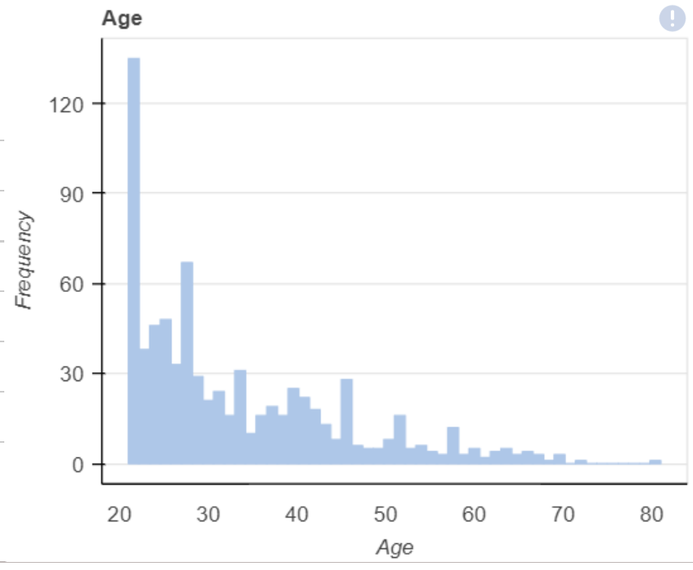


Age
numerical

Show Details

Approximate Distinct Count	52
Approximate Unique (%)	6.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

Mean	33.2409
Minimum	21
Maximum	81
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

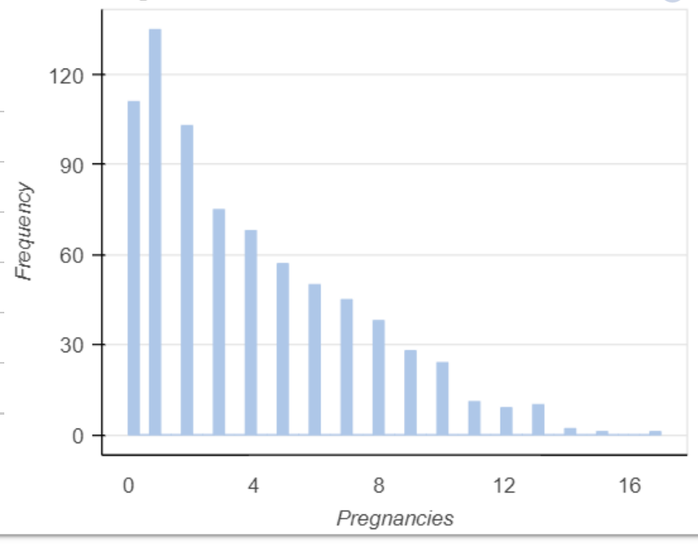


Pregnancies
numerical

Show Details

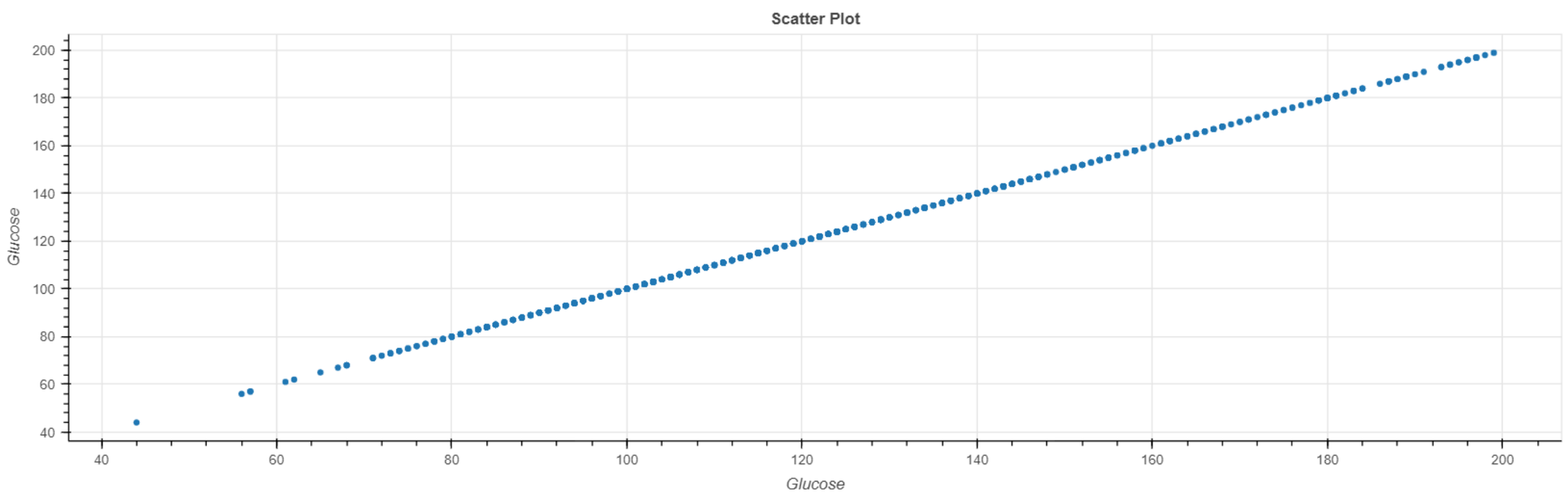
Approximate Distinct Count	17
Approximate Unique (%)	2.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

Mean	3.8451
Minimum	0
Maximum	17
Zeros	111
Zeros (%)	14.4%
Negatives	0
Negatives (%)	0.0%



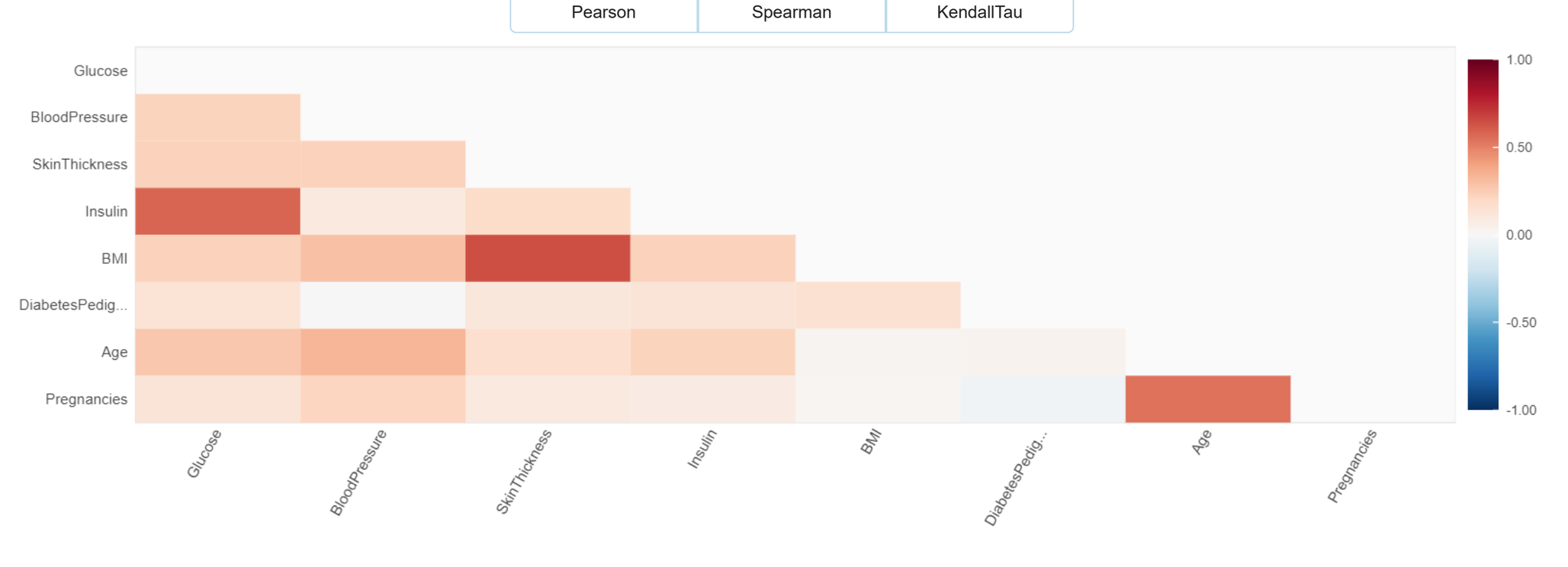
Interactions

X-Axis: Glucose ▾ Y-Axis: Glucose ▾

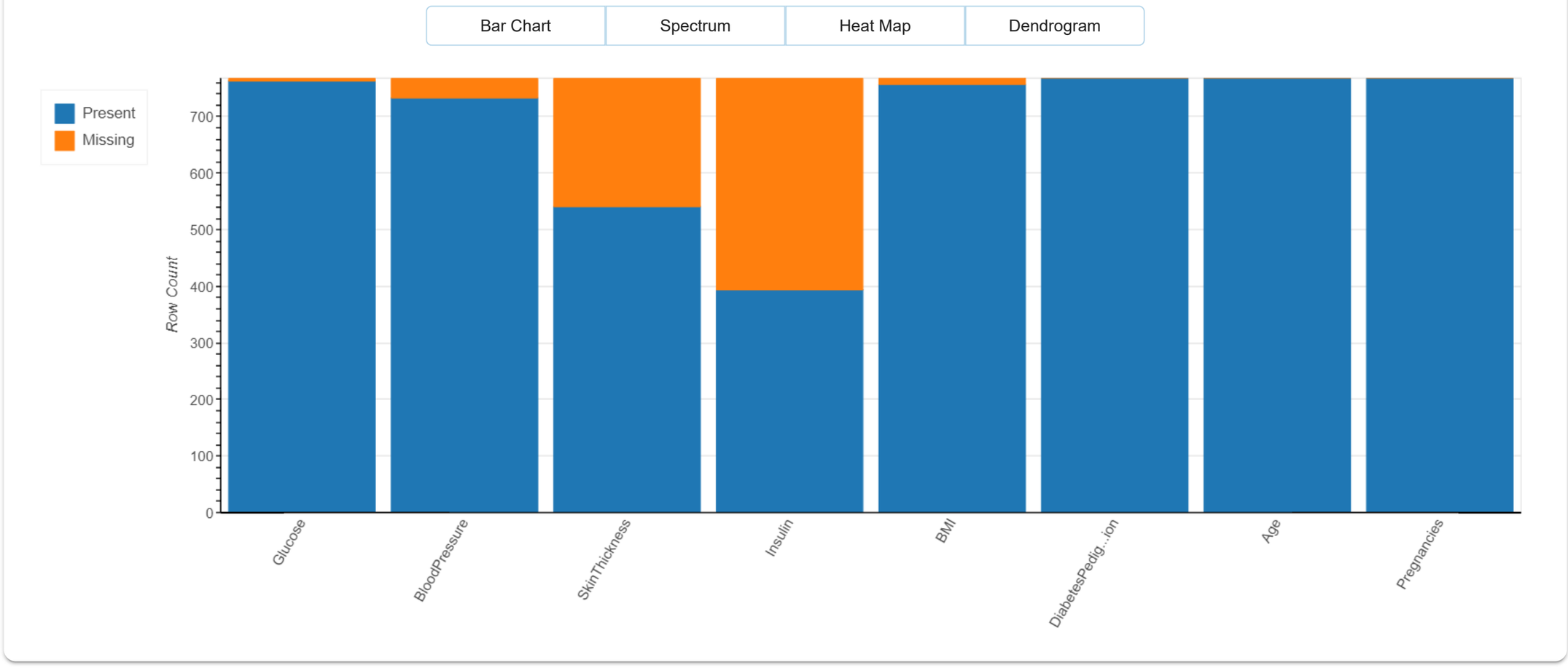


Correlations

Pearson | Spearman | KendallTau



Missing Values



Report generated with DataPrep

In [68]:
```
x1=x.copy()
#Percentage of null values
(x.isnull().sum()/len(x.index)*100)
```

Out[68]:
```
Glucose                    0.651042
BloodPressure              4.557292
SkinThickness             29.557292
Insulin                   48.697917
BMI                        1.432292
DiabetesPedigreeFunction   0.000000
Age                        0.000000
Pregnancies                0.000000
dtype: float64
```

In [69]:
```
#Droping "Insuline" as it has 48% Null values
x1.drop("Insulin", axis=1, inplace=True)
```

In [70]:
```
x1.head()
```

Out[70]:

	Glucose	BloodPressure	SkinThickness	BMI	DiabetesPedigreeFunction	Age	Pregnancies
0	148.0	72.0	35.0	33.6	0.627	50.0	6
1	85.0	66.0	29.0	26.6	0.351	31.0	1
2	183.0	64.0	NaN	23.3	0.672	32.0	8
3	89.0	66.0	23.0	28.1	0.167	21.0	1
4	137.0	40.0	35.0	43.1	2.288	33.0	0

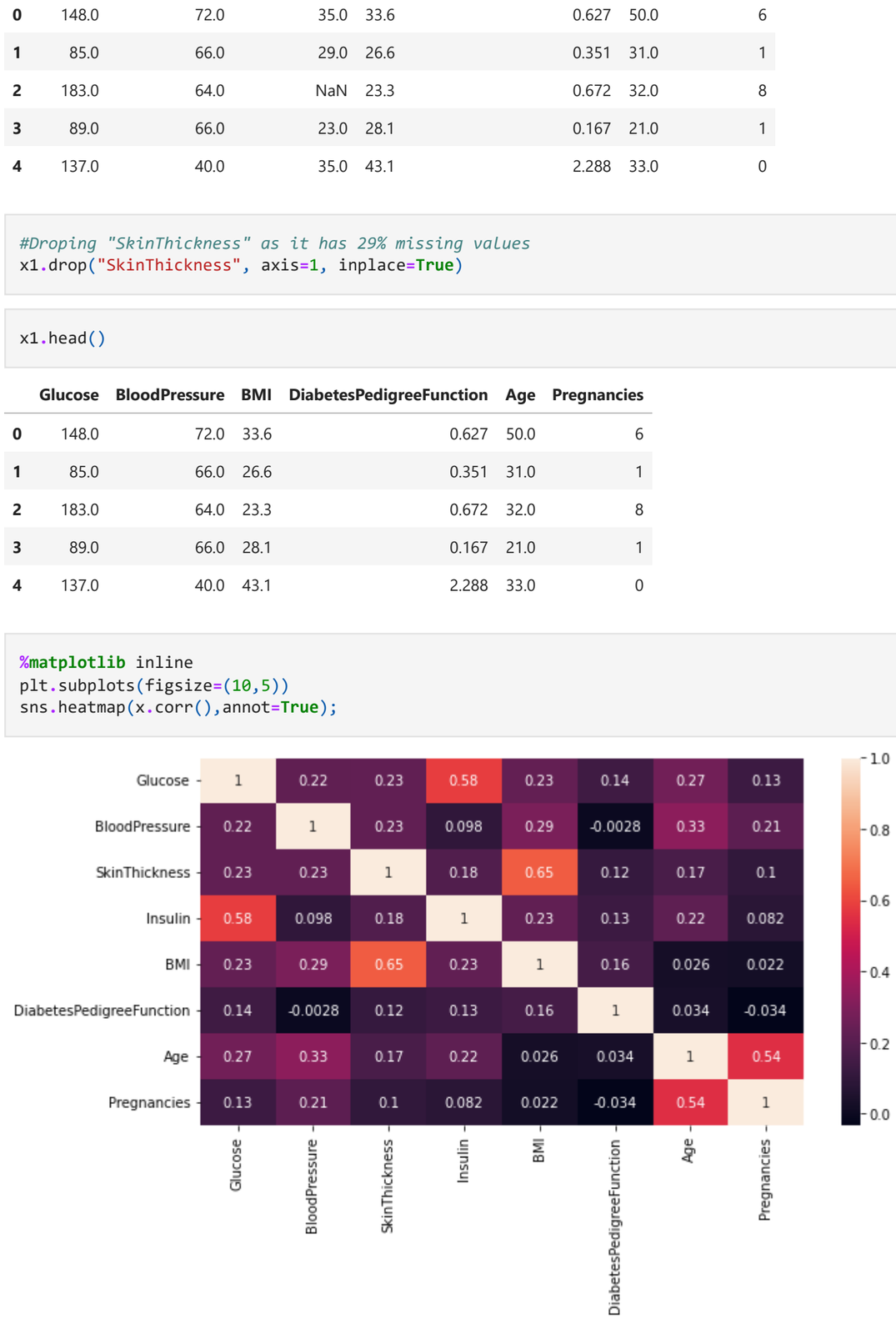
In [71]:
```
#Droping "SkinThickness" as it has 29% missing values
x1.drop("SkinThickness", axis=1, inplace=True)
```

In [72]:
```
x1.head()
```

Out[72]:

	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age	Pregnancies
0	148.0	72.0	33.6	0.627	50.0	6
1	85.0	66.0	26.6	0.351	31.0	1
2	183.0	64.0	23.3	0.672	32.0	8
3	89.0	66.0	28.1	0.167	21.0	1
4	137.0	40.0	43.1	2.288	33.0	0

In [73]:
```
%matplotlib inline
plt.subplots(figsize=(10,5))
sns.heatmap(x.corr(),annot=True);
```



In [74]:
```
(x1.isnull().sum()/len(x1.index)*100)
```

Out[74]:
```
Glucose                    0.651042
BloodPressure              4.557292
```

```
BMI                         1.432292
DiabetesPedigreeFunction    0.000000
Age                         0.000000
Pregnancies                 0.000000
dtype: float64
```

In [75]:
```
## Removing the null values with "median"
x1["Glucose"].fillna(np.median(df["Glucose"]), inplace=True)
x1["BloodPressure"].fillna(np.median(df["BloodPressure"]), inplace=True)
x1["BMI"].fillna(np.median(df["BMI"]), inplace=True)
```

Here we have replaced the null values with median as median is not affcted by outliers

In [76]:
```
x1.isnull().sum()
```

Out[76]:
```
Glucose                     0
BloodPressure               0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Pregnancies                 0
dtype: int64
```

Task 4

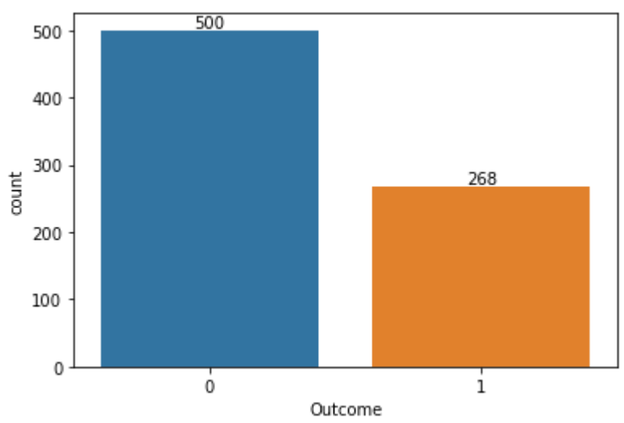
Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of actions.

In [77]:
```
y.head()
```

Out[77]:

	Outcome
0	1
1	0
2	1
3	0
4	1

In [78]:
```
figure,ax=plt.subplots(1,1)
ax=sns.countplot(x="Outcome", data=y)
ax.bar_label(ax.containers[0]);
```

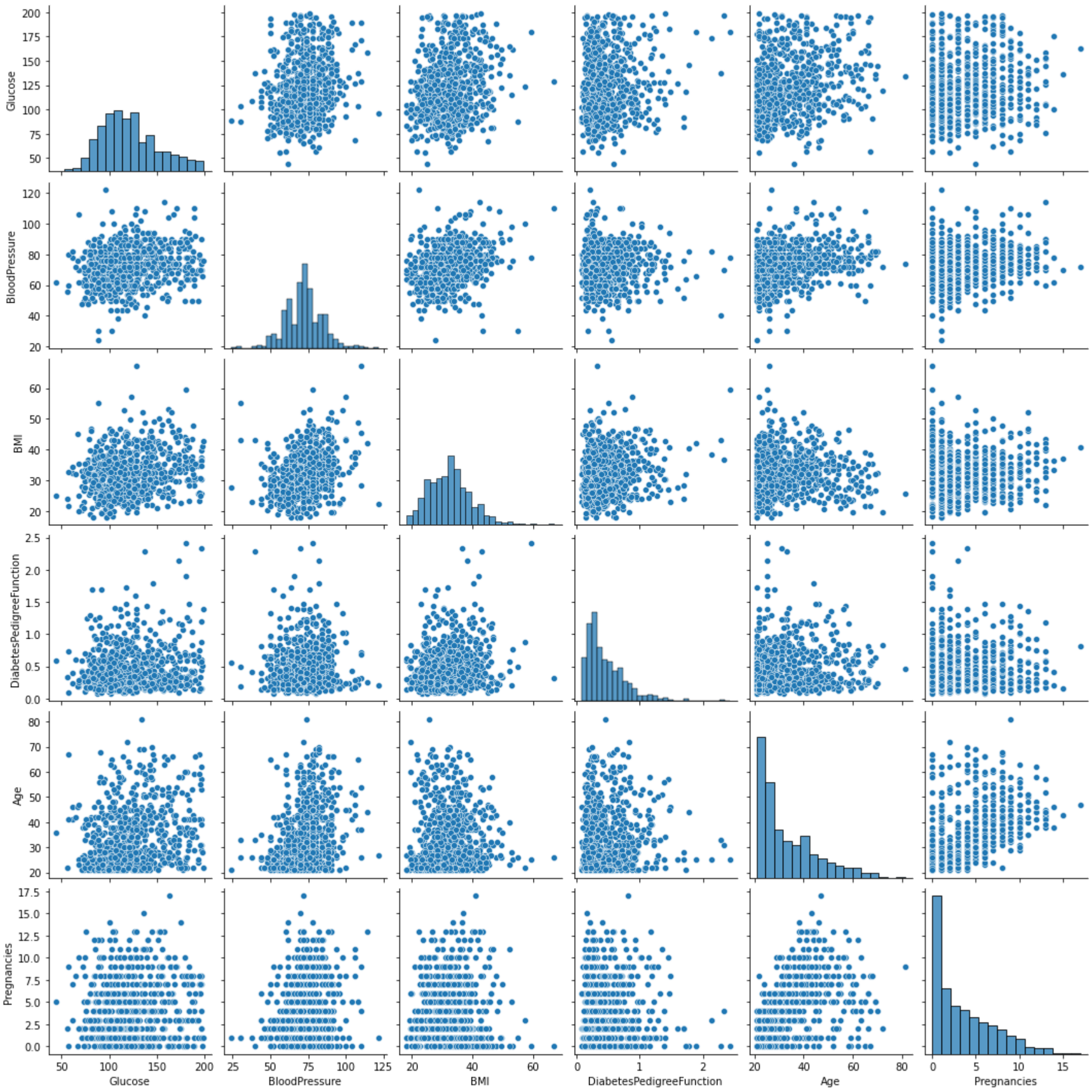


The outcome data is unbalanced. Hence we will Oversample the data using `SMOTE` technique to balance the data and evalueate the model result. If balancing data results imporovement of accuracy we will keep it otherwise we will discrad the Oversampled data.

Task 5.

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

In [79]:
```
sns.pairplot(data=x1);
```



In [80]:
```
##Again performing EDA to uderstand the data after a few manupulation performed
create_report(x1)
```

Out[80]:

DataPrep Report | Overview | Variables ≡ | Interactions | Correlations | Missing Values

Overview

Dataset Statistics

Number of Variables	6
Number of Rows	768
Missing Cells	0
Missing Cells (%)	0.0%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	36.1 KB
Average Row Size in Memory	48.2 B
Variable Types	Numerical: 6

Dataset Insights

`Age` is skewed	Skewed
`Pregnancies` is skewed	Skewed
`Pregnancies` has 111 (14.45%) zeros	Zeros

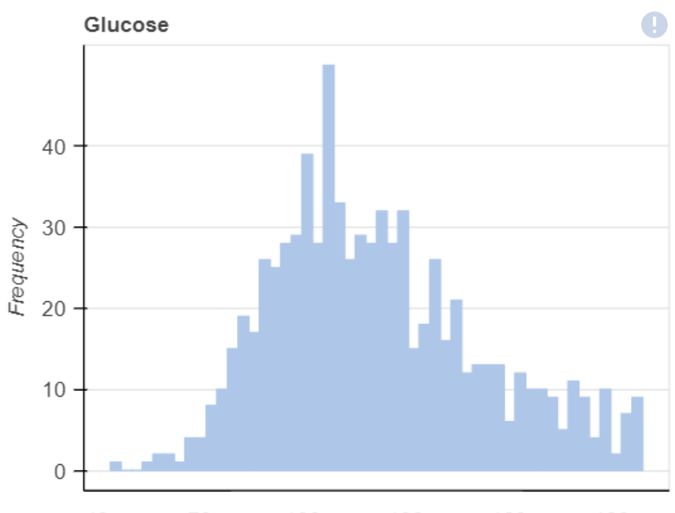
Variables

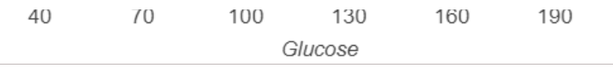
Sort by Feature order ☐ Reverse order

Glucose
numerical

Show Details

Approximate Distinct Count	135	Minimum	44
Approximate Unique (%)	17.6%	Maximum	199
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negatives	0
Infinite (%)	0.0%	Negatives (%)	0.0%
Memory Size	12.0 KB		
Mean	121.6562		



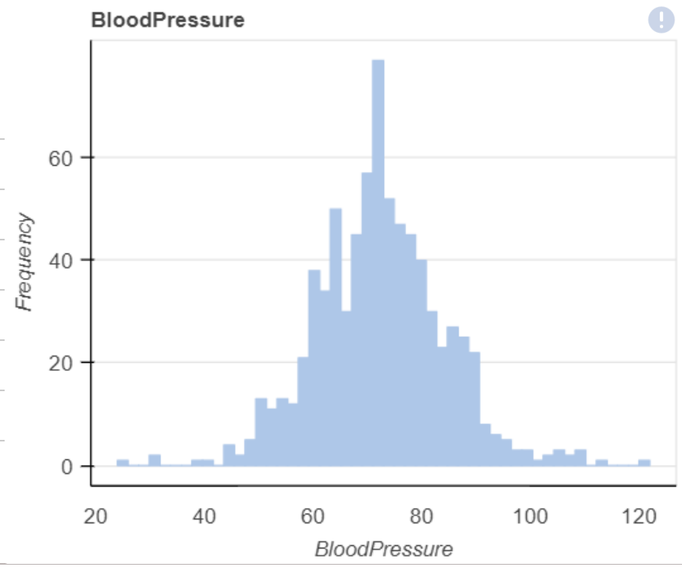


BloodPressure
numerical

Show Details

Approximate Distinct Count	46
Approximate Unique (%)	6.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

Mean	72.3867
Minimum	24
Maximum	122
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

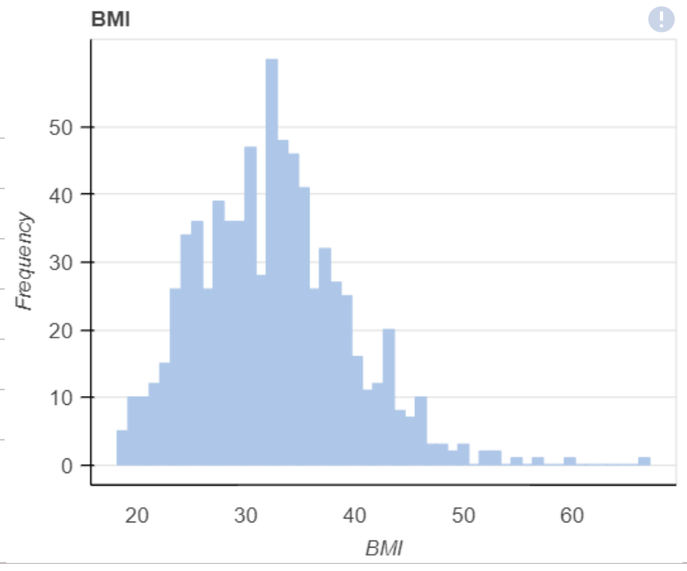


BMI
numerical

Show Details

Approximate Distinct Count	247
Approximate Unique (%)	32.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

Mean	32.4509
Minimum	18.2
Maximum	67.1
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

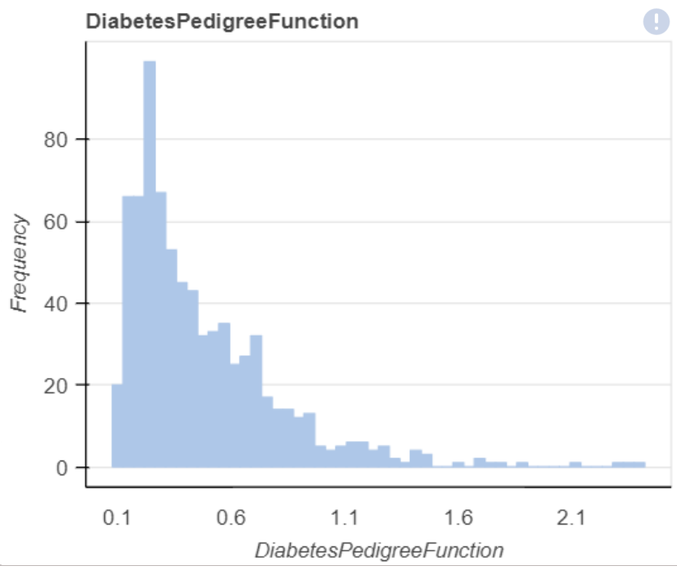


DiabetesPedigreeFun…
numerical

Show Details

Approximate Distinct Count	517
Approximate Unique (%)	67.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

Mean	0.4719
Minimum	0.078
Maximum	2.42
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

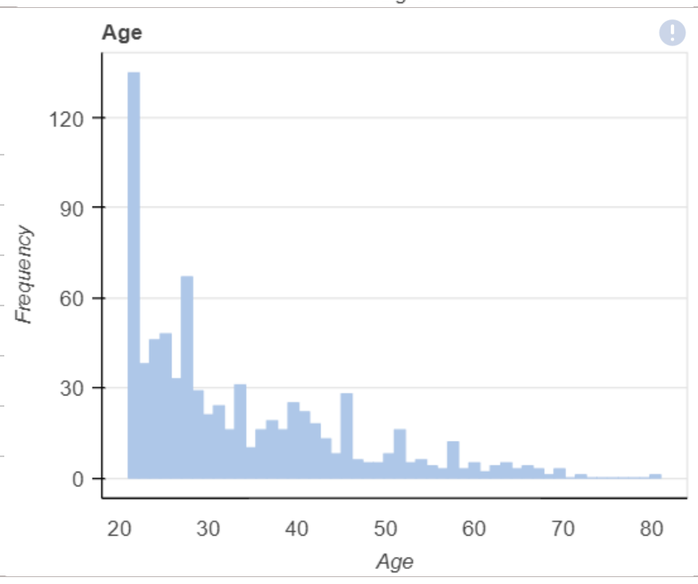


Age
numerical

Show Details

Approximate Distinct Count	52
Approximate Unique (%)	6.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

Mean	33.2409
Minimum	21
Maximum	81
Zeros	0
Zeros (%)	0.0%
Negatives	0
Negatives (%)	0.0%

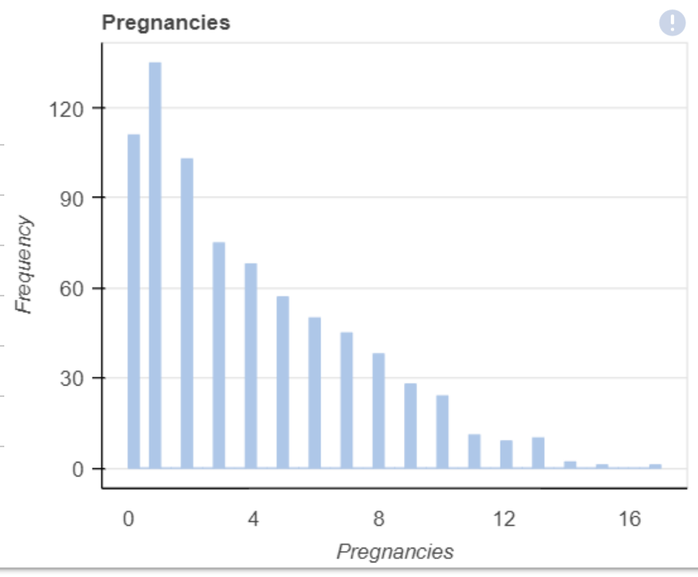


Pregnancies
numerical

Show Details

Approximate Distinct Count	17
Approximate Unique (%)	2.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Memory Size	12.0 KB

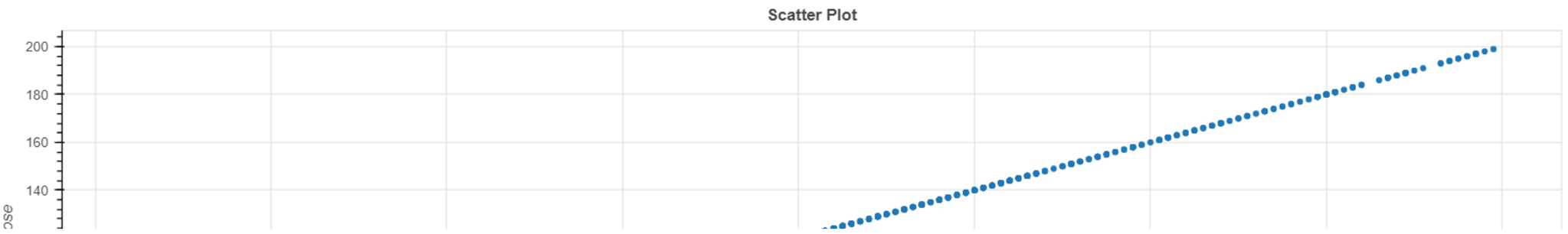
Mean	3.8451
Minimum	0
Maximum	17
Zeros	111
Zeros (%)	14.4%
Negatives	0
Negatives (%)	0.0%

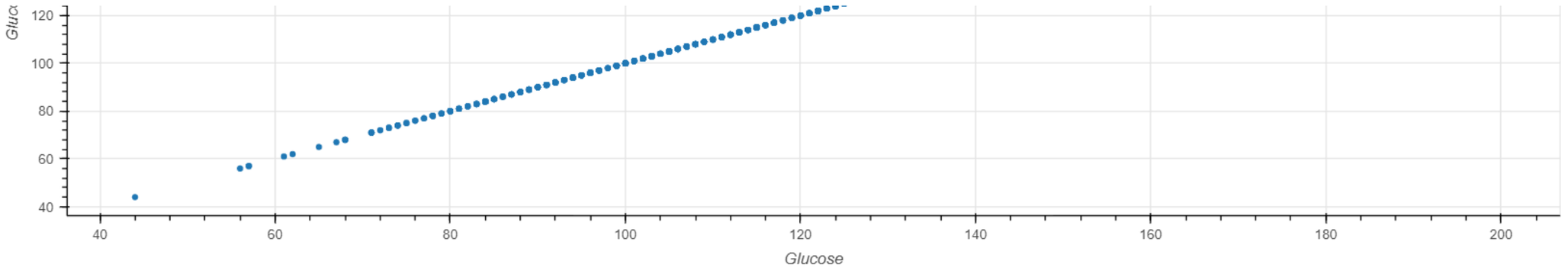


Interactions

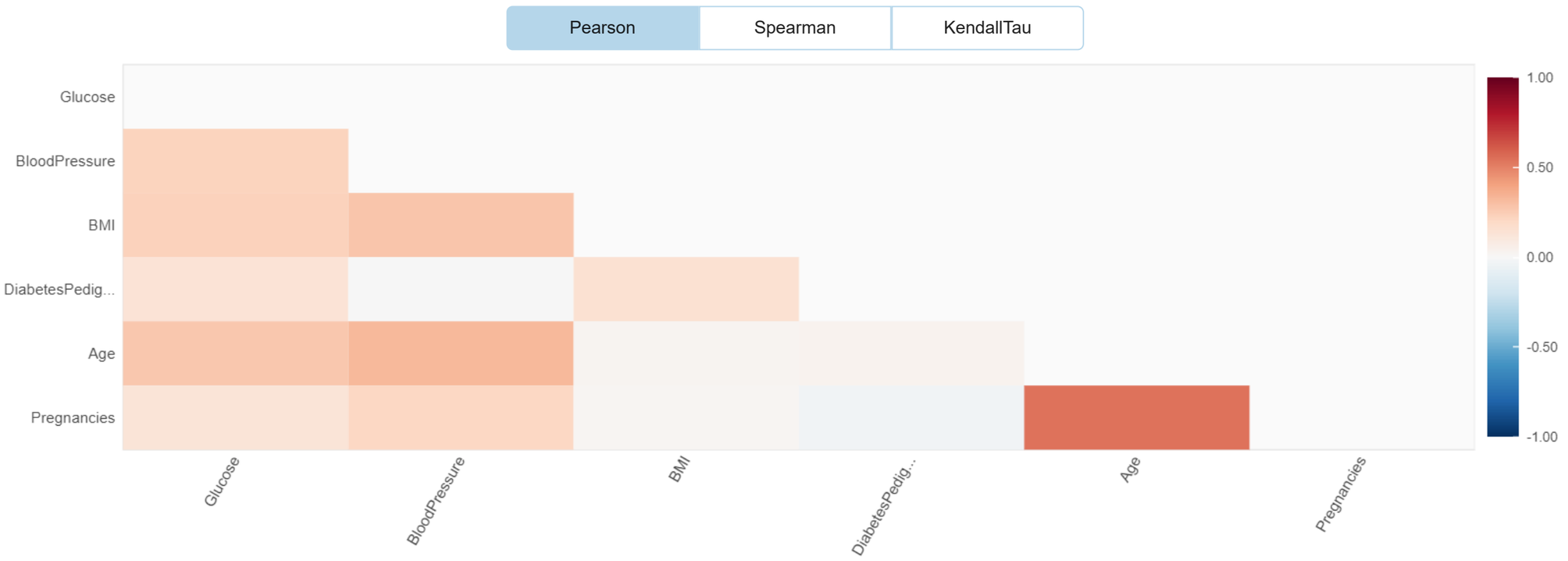
X-Axis
Glucose

Y-Axis
Glucose

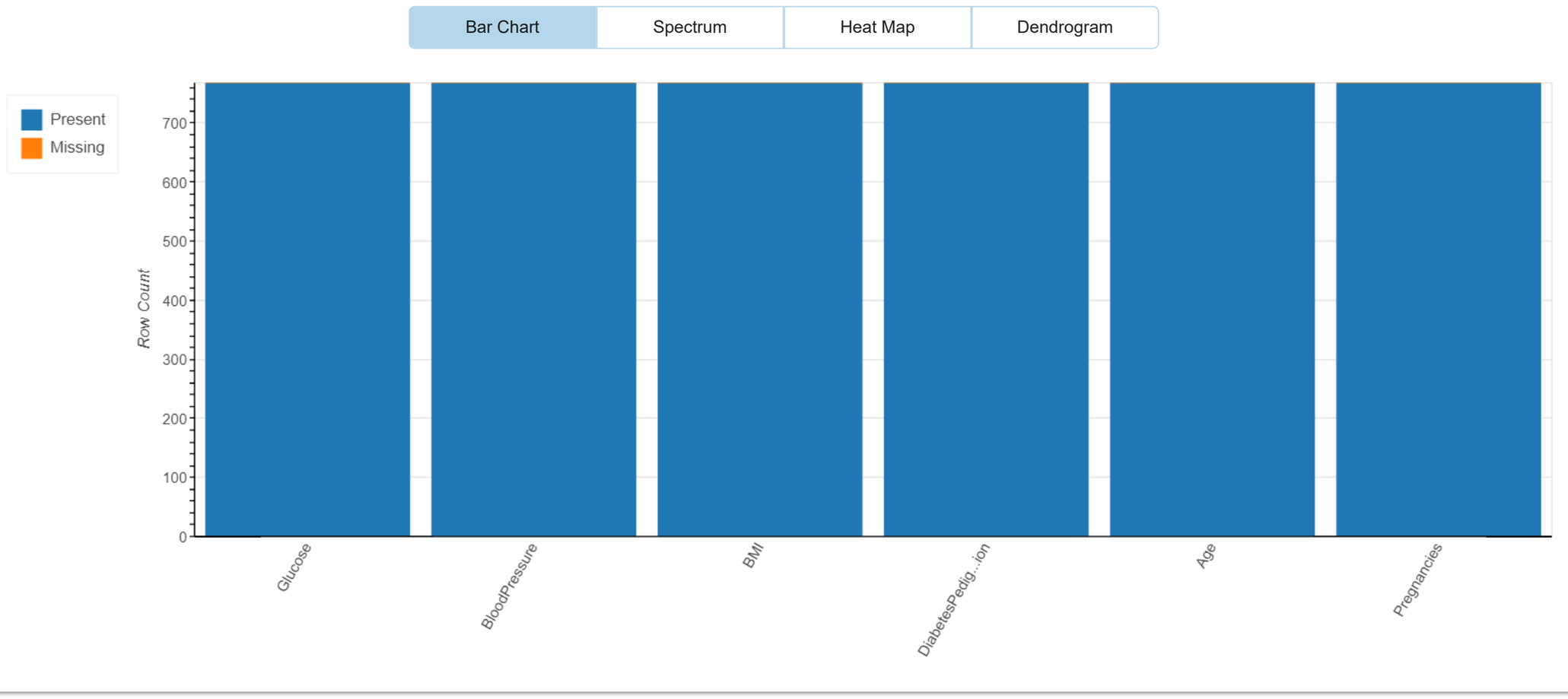




Correlations



Missing Values



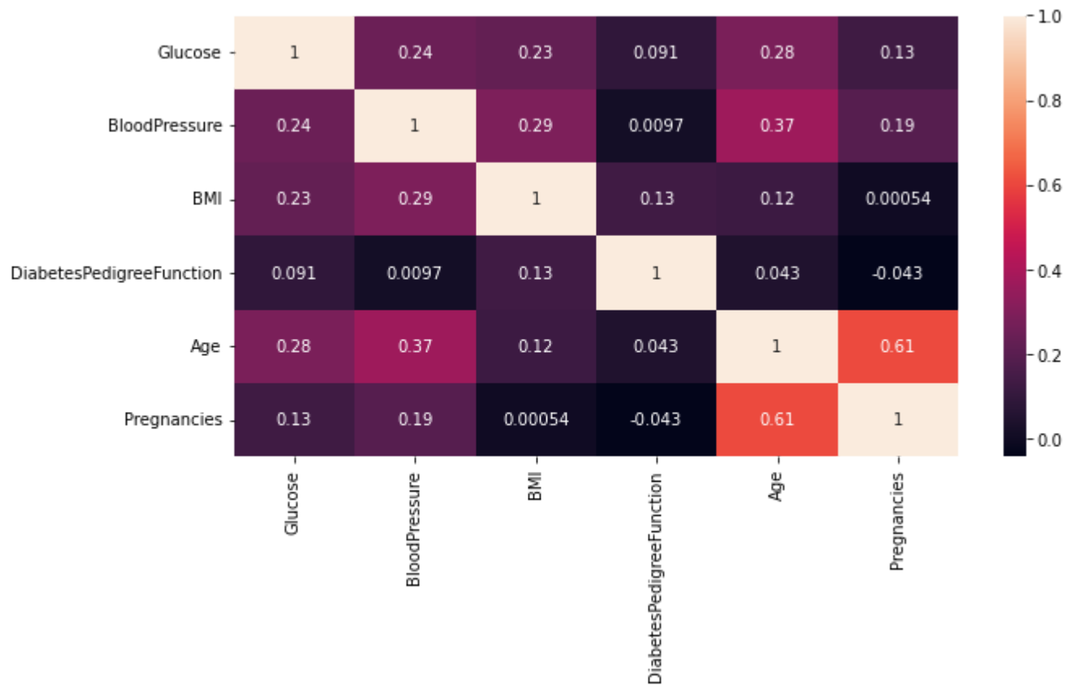
Report generated with DataPrep

Task 6

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

In [81]:

```
#Performing correlation analysis
%matplotlib inline
plt.subplots(figsize=(10,5))
sns.heatmap(x1.corr(method="spearman"),annot=True);
```



Pearson correlation is appropriate to linearly related variables, which each have a normal (Gaussian, “bell-shaped curve”, parametric) distribution, where most of the measures cluster around the mean (roughly 68% of the scores or measures will fall between the range mean plus or minus one standard deviation). Variables such Glucose, Blood Pressure etc. all form normal distributions except Age and Pregnancies. Spearman's rank correlation is a non-parametric method, it is based on rank, so when the data are not from normal distribution, Spearman's rank correlation may be more appropriate.

Spearman’s correlation is less sensitive to outliers, and is generally a more robust measure of association between continuous variables. It also measure monotonic correlation, and so may be more sensitive to non-linear relationships that Pearson correlation.

Spearman’s rank correlation can be used on non-linear related, non-normal distributions (non-parametric). Extremely outlying scores or measures are not given any undue meaning or influence. Since in this data set(x1) both Age and Pregnencies are skewed i.e don't have normal distribution we are considering `Pearson` correlation

Non-parametric tests are experiments that do not require the underlying population for assumptions. It does not rely on any data referring to any particular parametric group of probability distributions. Non-parametric methods are also called distribution-free tests since they do not have any underlying population.

From the above co-relation it is observed that Age and Pregnencies are highly correlated. But since data is skewed we are currently keep both the variables and try to normalise the data then decide the future course of action.

In [82]:
```
x1.agg(['skew', 'kurtosis']).transpose()
```

Out[82]:

	skew	kurtosis
Glucose	0.535587	-0.257847
BloodPressure	0.141885	1.098239
BMI	0.601059	0.921296
DiabetesPedigreeFunction	1.919911	5.594954
Age	1.129597	0.643159
Pregnancies	0.901674	0.159220

As rule of thumb, skewness can be interpreted like this:

Skewness Fairly Symmetrical-0.5 to 0.5

Moderate Skewed-0.5 to -1.0 and 0.5 to 1.0

Highly Skewed< -1.0 and > 1.0

In [83]:
```
#Making a copy to protect the data
df_trnsform=x1.copy()
```

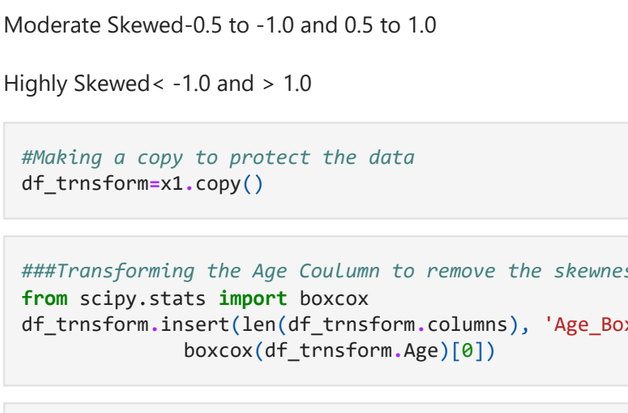
In [84]:
```
###Transforming the Age Coulumn to remove the skewness
from scipy.stats import boxcox
df_trnsform.insert(len(df_trnsform.columns), 'Age_Boxcox',
             boxcox(df_trnsform.Age)[0])
```

In [85]:
```
df_trnsform.head()
```

Out[85]:

	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age	Pregnancies	Age_Boxcox
0	148.0	72.0	33.6	0.627	50.0	6	0.901093
1	85.0	66.0	26.6	0.351	31.0	1	0.892411
2	183.0	64.0	23.3	0.672	32.0	8	0.893138
3	89.0	66.0	28.1	0.167	21.0	1	0.881083
4	137.0	40.0	43.1	2.288	33.0	0	0.893820

In [86]:
```
sns.kdeplot(data=df_trnsform.Age_Boxcox);
```



In [87]:
```
df_trnsform.Age_Boxcox.describe()
```

Out[87]:
```
count    768.000000
mean       0.891511
std        0.007036
min        0.881083
25%        0.885521
50%        0.890797
75%        0.898029
max        0.906274
Name: Age_Boxcox, dtype: float64
```

After Boxcox transformation it can be observed that Age coulumn as almost become Normalally distributed as Mean and 2nd Quertile (median)is almost equal

In [88]:
```
df_trnsform.insert(len(df_trnsform.columns), 'Pregnencies_Boxcox',
```

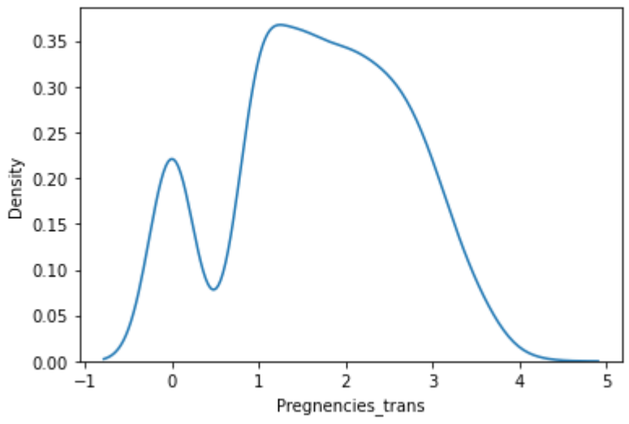
```
            boxcox(df_trnsform.Pregnancies)[0])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_19800/1847533117.py in <module>
      1 df_trnsform.insert(len(df_trnsform.columns), 'Pregnencies_Boxcox',
----> 2              boxcox(df_trnsform.Pregnancies)[0])

E:\notebooks\env\lib\site-packages\scipy\stats\morestats.py in boxcox(x, lmbda, alpha, optimizer)
   1062
   1063     if np.any(x <= 0):
-> 1064         raise ValueError("Data must be positive.")
   1065
   1066     if lmbda is not None:  # single transformation

ValueError: Data must be positive.
```

We can not use BoxCox transformation in Pregnencies as it contains 0 values also hence we have to look for alternative transformations

In [93]:
```
#Performing Square root method to remove the skewness
#Similarly transforming the "Pragnencies" column to remove the skewness using Square Root method
df_trnsform.insert(len(df_trnsform.columns), 'Pregnencies_trans',
             np.sqrt(df_trnsform.Pregnancies))
```

In [94]:
```
sns.kdeplot(data=df_trnsform.Pregnencies_trans);
```



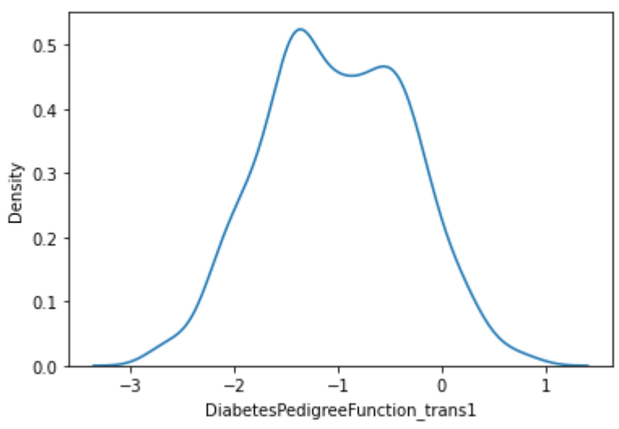
In [95]:
```
df_trnsform.Pregnencies_trans.describe(), np.median(df_trnsform.Pregnencies_trans)
```

Out[95]:
```
(count    768.000000
 mean       1.695477
 std        0.985735
 min        0.000000
 25%        1.000000
 50%        1.732051
 75%        2.449490
 max        4.123106
 Name: Pregnencies_trans, dtype: float64,
 1.7320508075688772)
```

We can oberve that mean and 2nd Quertile (median) is almost same hence almost the data is normalised

In [96]:
```
#Similarly transforming the "Pragnencies" column to remove the skewness using Square Root method
df_trnsform.insert(len(df_trnsform.columns), 'DiabetesPedigreeFunction_trans1',
             boxcox(df_trnsform.DiabetesPedigreeFunction)[0])
```

In [97]:
```
sns.kdeplot(data=df_trnsform.DiabetesPedigreeFunction_trans1);
```



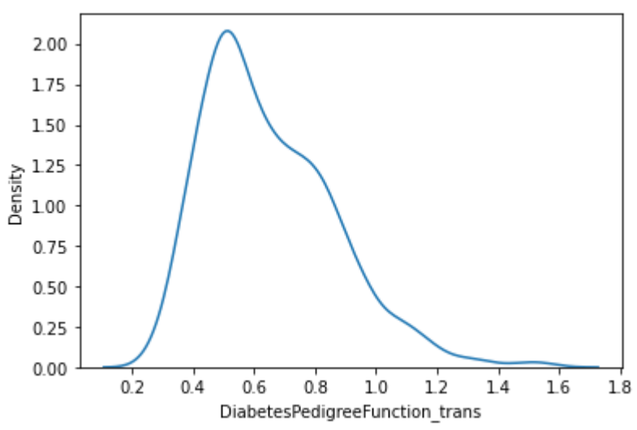
In [98]:
```
df_trnsform.DiabetesPedigreeFunction_trans1.describe(), np.median(df_trnsform.DiabetesPedigreeFunction_trans1)
```

Out[98]:
```
(count    768.000000
 mean      -1.010662
 std        0.690239
 min       -2.804441
 25%       -1.487026
 50%       -1.024049
 75%       -0.476104
 max        0.855822
 Name: DiabetesPedigreeFunction_trans1, dtype: float64,
 -1.024048576630844)
```

We can oberve that mean and 2nd Quertile (median) is almost same hence almost the data Diabetes Pedigree Function is normalised, but contains negative values which may not be in favour of the model prediction

In [89]:
```
#Similarly transforming the "Pragnencies" column to remove the skewness using Square Root method
df_trnsform.insert(len(df_trnsform.columns), 'DiabetesPedigreeFunction_trans',
             np.sqrt(df_trnsform.DiabetesPedigreeFunction))
```

In [45]:
```
sns.kdeplot(data=df_trnsform.DiabetesPedigreeFunction_trans);
```



In [90]:
```
df_trnsform.DiabetesPedigreeFunction_trans.describe(), np.median(df_trnsform.DiabetesPedigreeFunction)
```

Out[90]:
```
(count    768.000000
 mean       0.651989
 std        0.216445
 min        0.279285
 25%        0.493710
 50%        0.610327
 75%        0.791360
 max        1.555635
 Name: DiabetesPedigreeFunction_trans, dtype: float64,
 0.3725)
```

We can oberve that mean and 2nd Quertile (median) is almost same hence almost the data Diabetes Pedigree Function is normalised, Here the value is positive so we are going ahead with square root transformation

In [47]:
```
#df2=df_trnsform.copy()
```

In [99]:
```
df_trnsform.head()
```

Out[99]:

	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age	Pregnancies	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans	DiabetesPedigreeFunction_trans1
0	148.0	72.0	33.6	0.627	50.0	6	0.901093	0.791833	2.449490	-0.474866
1	85.0	66.0	26.6	0.351	31.0	1	0.892411	0.592453	1.000000	-1.088080
2	183.0	64.0	23.3	0.672	32.0	8	0.893138	0.819756	2.828427	-0.403329
3	89.0	66.0	28.1	0.167	21.0	1	0.881083	0.408656	1.000000	-1.912132
4	137.0	40.0	43.1	2.288	33.0	0	0.893820	1.512614	0.000000	0.803134

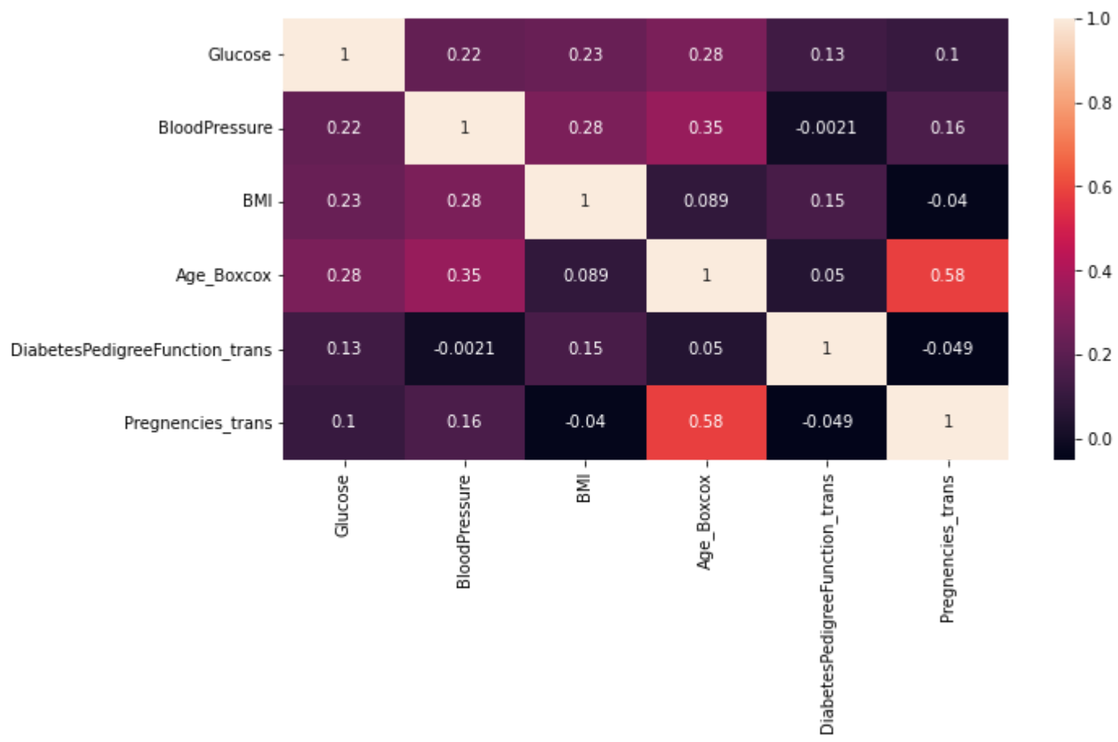
In [100…
```
df_trnsform.drop(["Age","Pregnancies","DiabetesPedigreeFunction","DiabetesPedigreeFunction_trans1"], axis=1, inplace=True)
```

In [101…
```
df_trnsform.head(1)
```

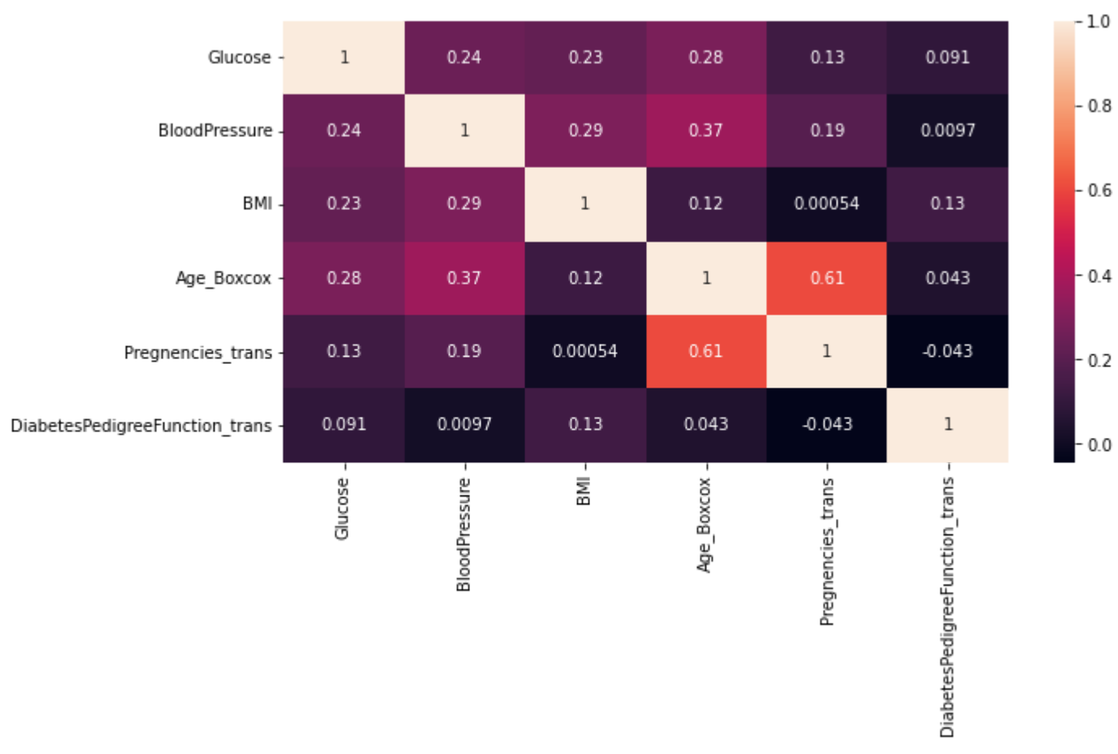
Out[101…

	Glucose	BloodPressure	BMI	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans
0	148.0	72.0	33.6	0.901093	0.791833	2.44949

In [103…
```
##Checking correlation after data manupulation
%matplotlib inline
plt.subplots(figsize=(10,5))
sns.heatmap(df_trnsform.corr(method="pearson"),annot=True);
```



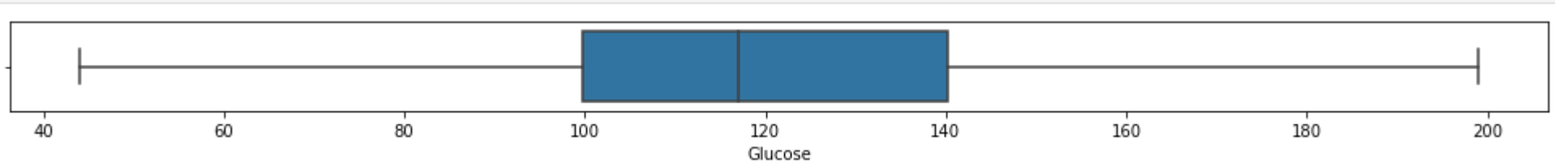
In [54]:
```
plt.subplots(figsize=(10,5))
sns.heatmap(df_trnsform.corr(method="spearman"),annot=True);
```

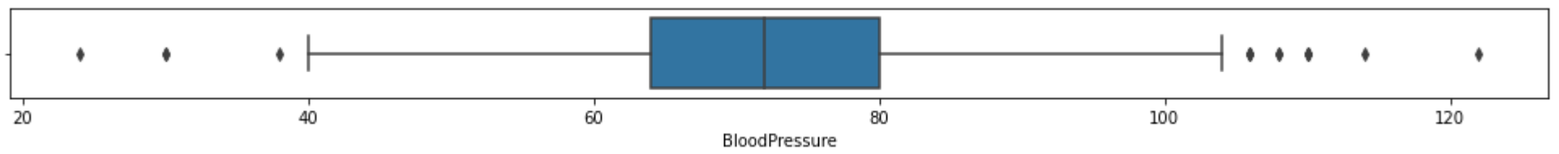


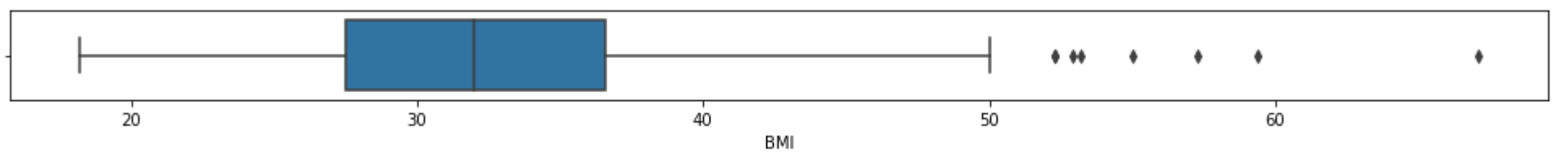
Now lets look at the outlier prospective

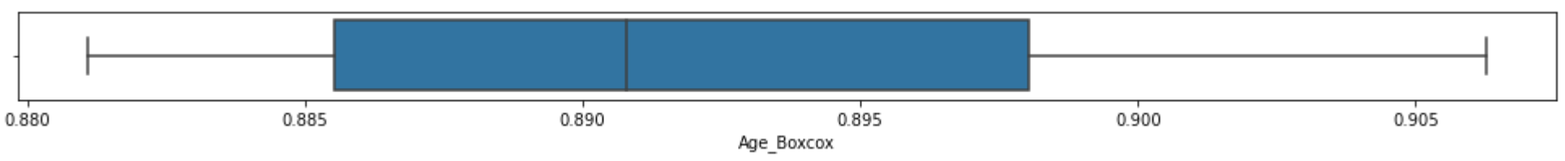
In [104…
```
df_imputed=df_trnsform.copy()
```

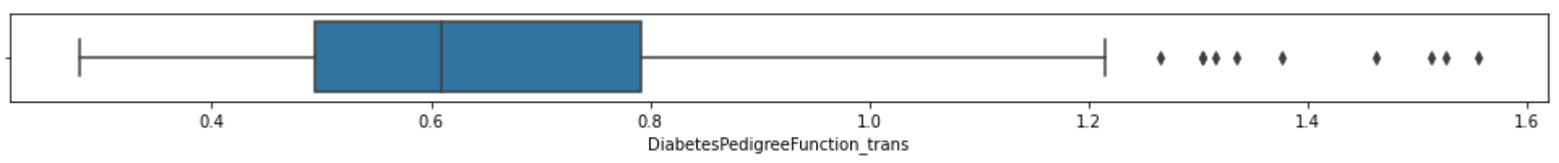
In [105…
```
for column in df_imputed:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=df_imputed, x=column)
```

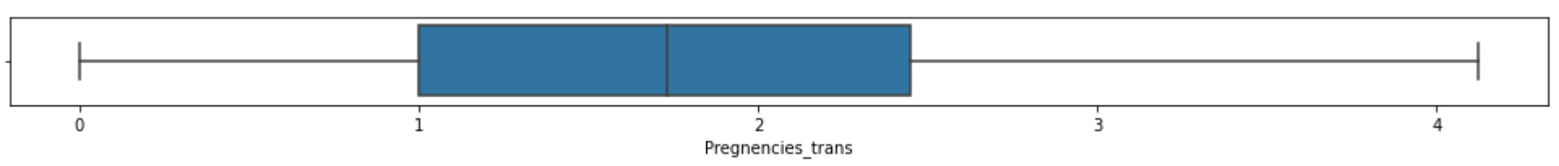












In [106…
```
df_imputed.shape
```

Out[106…
```
(768, 6)
```

Removing the outlier with Z score method

Z-score indicates how much a given value differs from the standard deviation. The Z-score, or standard score, is the number of standard deviations a given data point lies above or below mean. Standard deviation is essentially a reflection of the amount of variability within a given data set.

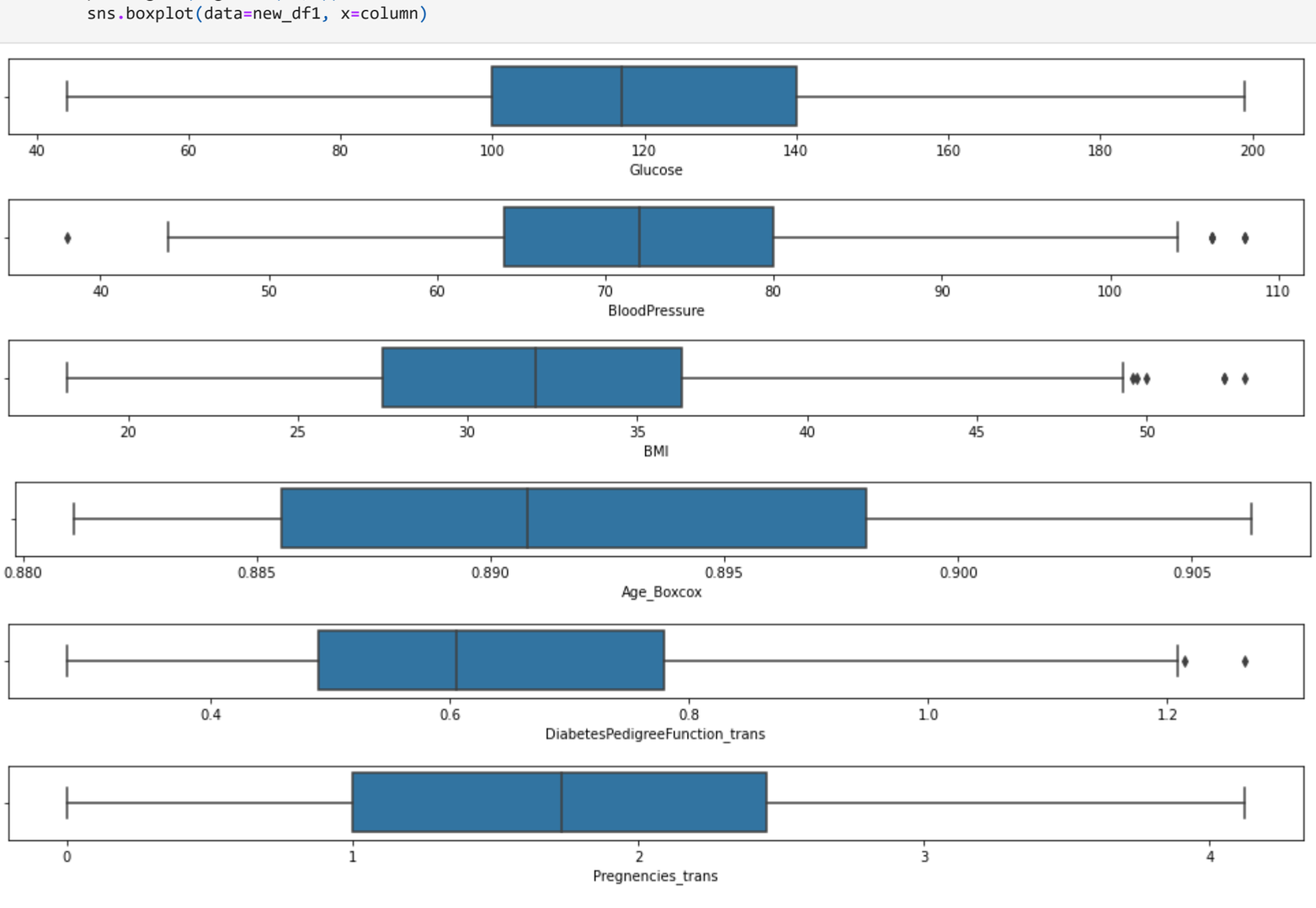
Z-score indicates how much a given value differs from the standard deviation. The Z-score, or standard score, is the number of standard deviations a given data point lies above or below mean. Standard deviation is essentially a reflection of the amount of variability within a given data set. When you have multiple samples and want to describe the standard deviation of those sample means (the standard error), you would use this z score formula: z = (x – μ) / (σ / √n) This z-score will tell you how many standard errors there are between the sample mean and the population mean.

In [107…
```
from scipy import stats
z_scores = stats.zscore(df_imputed)
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
new_df1 = df_imputed[filtered_entries]
```

In [108…
```
new_df1.shape
```

Out[108…
```
(749, 6)
```

In [109…
```
for column in new_df1:
        plt.figure(figsize=(17,1))
        sns.boxplot(data=new_df1, x=column)
```



If we are apllying the scalled data to the Ensable and Logistic models the accuracy is decreasing with Z_score 2.5, but increases with Z_Score 3. Hence we are keeping Z_Score as 3

Now we need to allign the shape of test data (new_df1) and train data (y)¶

In [110…
```
x.shape,df_imputed.shape,new_df1.shape,y.shape
```

Out[110…
```
((768, 8), (768, 6), (749, 6), (768, 1))
```

In [111…
```
#Extracting the rows with are diffrent before outlier removal and after outlier removal
Df_diff=pd.concat([df_imputed,new_df1]).drop_duplicates(keep=False)
```

In [112…
```
Df_diff.head()
```

Out[112…

	Glucose	BloodPressure	BMI	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans
4	137.0	40.0	43.1	0.893820	1.512614	0.0
18	103.0	30.0	43.3	0.893820	0.427785	1.0
43	171.0	110.0	45.4	0.902113	0.849117	3.0
45	180.0	66.0	42.0	0.886753	1.375863	0.0

	Glucose	BloodPressure	BMI	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans
58	146.0	82.0	40.5	0.899196	1.334541	0.0

In [113…
```
Df_diff.shape,Df_diff.index
```

Out[113…
```
((19, 6),
 Int64Index([  4,  18,  43,  45,  58, 106, 120, 125, 177, 228, 370, 371, 445,
             549, 593, 597, 621, 673, 691],
            dtype='int64'))
```

In [114…
```
y_train=y.copy()
```

In [115…
```
y_train.drop(Df_diff.index, axis=0, inplace=True)
```

In [116…
```
y_train.shape, new_df1.shape
```

Out[116…
```
((749, 1), (749, 6))
```

In [117…
```
y_train.head()
```

Out[117…

	Outcome
0	1
1	0
2	1
3	0
5	0

Task 7

Devise strategies for model building. It is important to decide the right validation framework. Express your thought process. Would Cross validation be useful in this scenario?

Model choices

Now we've got our data prepared, we can start to fit models. We'll be using the following and comparing their results.

Logistic Regression - `LogisticRegression()`

K-Nearest Neighbors - `KNeighboursClassifier()`

RandomForest - `RandomForestClassifier()`

Xgboost classifier- `XGBClassifier()`

catboost classifier- `CatBoostClassifier()`

But before going to model building we will Scale the data as we are using both distance based as well as tree based models. Tree based models does not require scalling necessarily but no harm in doing scalling. But distance based model need scalling on mendatory basis. So we are going ahead with scalled data

Task 7

It is important to decide the right validation framework. Express your thought process. Would Cross validation be useful in this scenario?

Building machine learning models is an important element of predictive modeling. However, without proper model validation, the confidence that the trained model will generalize well on the unseen data can never be high. Model validation helps in ensuring that the model performs well on new data, and helps in selecting the best model, the parameters, and the accuracy metrics.

Hold Out Validation K-fold Cross-Validation.

Stratified K-fold Cross-Validation

Leave One Out Cross-Validation.

Repeated Random Test-Train Splits

We will try with

i)K-fold Cross-Validation.

ii)Stratified K-fold Cross-Validation

link https://www.pluralsight.com/guides/validating-machine-learning-models-scikit-learn

To avoid overfitting https://medium.com/analytics-vidhya/overfitting-detection-prevention-848e6adeabf8

Cross-validation: We will consider Cross-Validation as method to prevent overfitting

In standard k-fold cross-validation, we split the data into k subsets of approximately equal size, known as folds. Then, we iteratively train the algorithm by keeping one fold for testing and other k-1 fold for training. Each time a different set or group of folds is for validation. hat we'll do to make them more solid is calculate them using cross-validation.

We'll take the best model along with the best hyperparameters and use cross_val_score() along with various scoring parameter values.

cross_val_score() works by taking an estimator (machine learning model) along with data and labels. It then evaluates the machine learning model on the data and labels using cross-validation and a defined scoring parameter.

We will handle Cross Validation while hyper parameter tuning section

In [118…
```
#Our data before Scalling
y_train.head()
```

Out[118…

	Outcome
0	1
1	0
2	1
3	0
5	0

In [119…
```
new_df1.head()
```

Out[119…

	Glucose	BloodPressure	BMI	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans
0	148.0	72.0	33.6	0.901093	0.791833	2.449490
1	85.0	66.0	26.6	0.892411	0.592453	1.000000
2	183.0	64.0	23.3	0.893138	0.819756	2.828427
3	89.0	66.0	28.1	0.881083	0.408656	1.000000

	Glucose	BloodPressure	BMI	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans
5	116.0	74.0	25.6	0.891632	0.448330	2.236068

In [120…
```
new_df1.describe()
```

Out[120…

	Glucose	BloodPressure	BMI	Age_Boxcox	DiabetesPedigreeFunction_trans	Pregnencies_trans
count	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000
mean	121.253672	72.305741	32.223231	0.891549	0.642388	1.710585
std	30.126778	11.285727	6.520959	0.007058	0.201198	0.979297
min	44.000000	38.000000	18.200000	0.881083	0.279285	0.000000
25%	100.000000	64.000000	27.500000	0.885521	0.489898	1.000000
50%	117.000000	72.000000	32.000000	0.890797	0.604979	1.732051
75%	140.000000	80.000000	36.300000	0.898029	0.779102	2.449490
max	199.000000	108.000000	52.900000	0.906274	1.264911	4.123106

We are going ahead with Min-Max scalling due to following reason

`It essentially shrinks the range such that the range is now between 0 and 1 (or -1 to 1 if there are negative values).

This scaler works better for cases in which the standard scaler might not work so well.

If the distribution is not Gaussian or the standard deviation is very small, the min-max scaler works better.

xi–min(x)/max(x)–min(x)` However, it is sensitive to outliers and we have already removed the outliers

Thumb rule

Use MinMaxScaler as your default

Use RobustScaler if you have outliers and can handle a larger range

Use StandardScaler if you need normalized features

Use Normalizer sparingly - it normalizes rows, not columns

link:https://www.kaggle.com/discdiver/guide-to-scaling-and-standardizing

In [121…
```
MinMaxscaler = MinMaxScaler()
```

In [122…
```
###Train test split
X_train,X_test,Y_train,Y_test=train_test_split(new_df1,y_train,random_state=2, test_size=.2)
```

In [123…
```
X_train_scalled=MinMaxscaler.fit_transform(X_train)
```

In [124…
```
X_train_scalled
```

Out[124…
```
array([[0.64935065, 0.59375   , 0.66570605, 0.34997412, 0.50710626,
        0.24253563],
       [0.30519481, 0.40625   , 0.42939481, 0.06431762, 0.39421384,
        0.48507125],
       [0.95454545, 0.375     , 0.36599424, 0.53102019, 0.27142552,
        0.42008403],
       ...,
       [0.88311688, 0.71875   , 0.52737752, 0.55489944, 0.2851712 ,
        0.        ],
       [0.36363636, 0.46875   , 0.61095101, 0.70433523, 0.5415179 ,
        0.68599434],
       [0.26623377, 0.53125   , 0.37463977, 0.68890773, 0.34371719,
        0.59408853]])
```

In [125…
```
X_test_scalled=MinMaxscaler.transform(X_test)
```

In [126…
```
#Scalling the unsplitted data as well
df_scalled=MinMaxscaler.fit_transform(new_df1)
```

Now we've got our data prepared, we can start to fit models. We'll be using the following and comparing their results.

1. Logistic Regression - `LogisticRegression()`
2. K-Nearest Neighbors - `KNeighboursClassifier()`
3. RandomForest - `RandomForestClassifier()`
4. Xgboost classifier- `XGBClassifier()`
5. catboost classifier- `CatBoostClassifier()`

In [129…
```
# Put models in a dictionary
model_distance= {"Logistic_Regression": LogisticRegression(),
                 "KNN": KNeighborsClassifier(),
                 "SVM":SVC()}
model_ensamble={"Random_Forest": RandomForestClassifier(),
          "Catboost": CatBoostClassifier(verbose=False),
           "Xgboost Classifier": XGBClassifier()
          }
models={"KNN":KNeighborsClassifier(),
        "SVM":SVC(),
       "Random_Forest":RandomForestClassifier(),
       "Logistic_Regression":LogisticRegression(),
       "Catboot":CatBoostClassifier(),
       "Xgboost_Classifier":XGBClassifier()}

# Create function to fit and score models
def fit_and_score(models, X_train, X_test, y_train, y_test):
    """
    Fits and evaluates given machine learning models.
    models : a dict of different Scikit-Learn machine learning models
    X_train : training data
    X_test : testing data
    y_train : labels assosciated with training data
    y_test : labels assosciated with test data
    """
    # Random seed for reproducible results
    np.random.seed(42)
    # Make a list to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[name] = model.score(X_test, y_test)

    return model_scores
```

In [ ]:
```
```

In [130…
```
## Fitting ensmbled model with unscalled data
model_scores_ensable = fit_and_score(models=model_ensamble,
                             X_train=X_train,
                             X_test=X_test,
                             y_train=Y_train,
                             y_test=Y_test)
model_scores_ensable
```

```
[07:35:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```
Out[130…
```
{'Random_Forest': 0.7666666666666667,
 'Catboost': 0.78,
 'Xgboost Classifier': 0.72}
```

In [133…
```
## Fitting models on scalled data
model_score = fit_and_score(models=model_distance,
                             X_train=X_train_scalled,
                             X_test=X_test_scalled,
                             y_train=Y_train,
                             y_test=Y_test)
model_score
```

Out[133…
```
{'Logistic_Regression': 0.8133333333333334,
 'KNN': 0.72,
 'SVM': 0.7933333333333333}
```

In [134…
```
#Fittiing scalled data on Ensambled data
model_scalled_score = fit_and_score(models=model_ensamble,
                             X_train=X_train_scalled,
                             X_test=X_test_scalled,
                             y_train=Y_train,
                             y_test=Y_test)
model_scalled_score
```

```
[07:37:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```
Out[134…
```
{'Random_Forest': 0.76, 'Catboost': 0.78, 'Xgboost Classifier': 0.72}
```

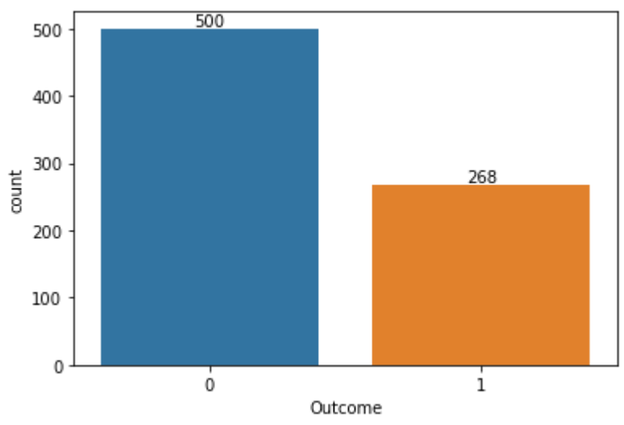
The result of Ensambled models are not effected by whether data is scalled or not. So will be doing further analysis with scalled data only

In [135…
```
X_train_scalled.shape,Y_train.shape
```

Out[135…
```
((599, 6), (599, 1))
```

If we are apllying the scalled data to the Ensable and Logistic models the accuracy is decreasing with Z_score 2.5, but increases with Z_Score 3.

In [136…
```
#### Considering the data imbalance
figure,ax=plt.subplots(1,1)
ax=sns.countplot(x="Outcome", data=y)
ax.bar_label(ax.containers[0]);
```



In [137…
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 42)
X_train_res, Y_train_res = sm.fit_resample(X_train_scalled, Y_train)
```

In [138…
```
X_train_res.shape,Y_train_res.shape
```

Out[138…
```
((772, 6), (772, 1))
```

In [139…
```
model_scores_ensable_res = fit_and_score(models=model_ensamble,
                             X_train=X_train_res,
                             X_test=X_test,
                             y_train=Y_train_res,
                             y_test=Y_test)
model_scores_ensable_res
```

```
[07:38:26] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```
Out[139…
```
{'Random_Forest': 0.35333333333333333,
 'Catboost': 0.30666666666666664,
 'Xgboost Classifier': 0.30666666666666664}
```

In [140…
```
model_KNN_score_res = fit_and_score(models=model_distance,
                             X_train=X_train_res,
                             X_test=X_test_scalled,
                             y_train=Y_train_res,
                             y_test=Y_test)
model_KNN_score_res
```

Out[140…
```
{'Logistic_Regression': 0.76,
 'KNN': 0.7133333333333334,
 'SVM': 0.7266666666666667}
```

In [141…
```
model_scalled_score = fit_and_score(models=model_ensamble,
                             X_train=X_train_scalled,
                             X_test=X_test_scalled,
                             y_train=Y_train,
                             y_test=Y_test)
model_scalled_score
```

```
[07:38:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```
Out[141…
```
{'Random_Forest': 0.76, 'Catboost': 0.78, 'Xgboost Classifier': 0.72}
```

Compared to the result before balancing the accuracy of the model is decreasing darstically. So will use imbalanced data. However we will try to balance the data using the "child_weight" parameter while building the model itself

Lets check overfitting perspective

In [143…
```
# Put models in a dictionary
```

```
model_distance= {"KNN": KNeighborsClassifier(),
                "SVM":SVC(),
                 "Logistic_Regression": LogisticRegression(class_weight={0:0.4,1:0.6})}
model_ensamble={"Random_Forest": RandomForestClassifier(),
          "Catboost": CatBoostClassifier(verbose=False),
          "Xgboost Classifier": XGBClassifier()
         }
models={"KNN":KNeighborsClassifier(),
        "SVM":SVC(),
       "Random_Forest":RandomForestClassifier(),
       "Logistic_Regression":LogisticRegression(),
       "Catboot":CatBoostClassifier(),
       "Xgboost_Classifier":XGBClassifier()}

# Create function to fit and score models
def Overfitting(models, X_train, X_test, y_train, y_test):
    """
    Fits and evaluates given machine learning models.
    models : a dict of different Scikit-Learn machine learning models
    X_train : training data
    X_test : testing data
    y_train : labels assosciated with training data
    y_test : labels assosciated with test data
    """
    # Random seed for reproducible results
    np.random.seed(42)
    # Make a list to keep model scores
    accuracy_train = {}
    accuracy_test={}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)

        # Evaluate the model and append its score to model_scores
        #y_pred[name]=model.predict(X_test)
        model.predict(X_test)
        #model.predict(X_train)
        #predictions_train[name] = [round(value) for value in  y_pred[name]]
        accuracy_train[name] = accuracy_score(Y_train, model.predict(X_train))
        accuracy_test[name]= accuracy_score(Y_test, model.predict(X_test))

    #return accuracy_train,accuracy_test
    Test= pd.DataFrame().append(accuracy_test, ignore_index=True)
    Train=pd.DataFrame().append(accuracy_train, ignore_index=True)
    #print(Test)
    Table=Test.append(Train)
    Table.index=["Test_Accuracy","Train_Accuracy"]
    return Table
```

In [144…

```
Overfitting(models=model_distance, X_train=X_train_scalled, X_test=X_test_scalled,
            y_train=Y_train, y_test=Y_test)
```

Out[144…

	KNN	SVM	Logistic_Regression
Test_Accuracy	0.720000	0.793333	0.780000
Train_Accuracy	0.844741	0.804674	0.766277

In [145…

```
Overfitting(models=model_ensamble, X_train=X_train_scalled, X_test=X_test_scalled,
            y_train=Y_train, y_test=Y_test)
```

```
[07:41:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[145…

	Random_Forest	Catboost	Xgboost Classifier
Test_Accuracy	0.76	0.780000	0.72
Train_Accuracy	1.00	0.951586	1.00

In [146…

```
Overfitting(models=model_ensamble, X_train=X_train, X_test=X_test,
            y_train=Y_train, y_test=Y_test)
```

```
[07:41:06] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[146…

	Random_Forest	Catboost	Xgboost Classifier
Test_Accuracy	0.766667	0.780000	0.72
Train_Accuracy	1.000000	0.951586	1.00

From the above it can be seen that Only "Logistic Regression", "SVM" and "KNN" wroks good. Out of which SVM is best as both Train and Test Accuracy is almost same. KNN miserably over fits

We will consider only SVM and Logistic Regression for future analysis and KNN as per requirement of the problem statement

Task 8 Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN.

Parth of the Task 8 already completed and part will be carry forwarded with Task 9

Task 9

Create a classification report by analysing sensitivity, specificity, AUC(ROC curve) etc. Please try to be as descriptive as possible to explain what values of these parameter you settled for? any why?

We will carry out both the task togather

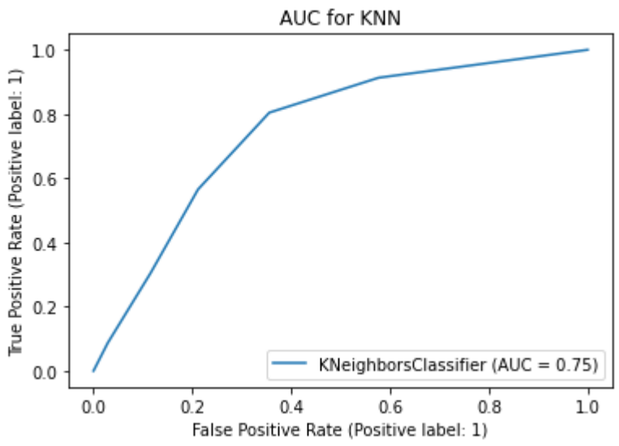
In [150…

```
### Defining a function to draw AUC curve as well as printing classification report
model_SVC = SVC()
model_KNN=KNeighborsClassifier()
model_Logistic=LogisticRegression()
def ROC_Curve(model,X_train,y_train,X_test,y_test):
    model.fit(X_train,Y_train)
    y_pred = model.predict(X_test)
    plot_roc_curve(model, X_test,y_test)
    print(classification_report(y_test,y_pred))
```

In [151…

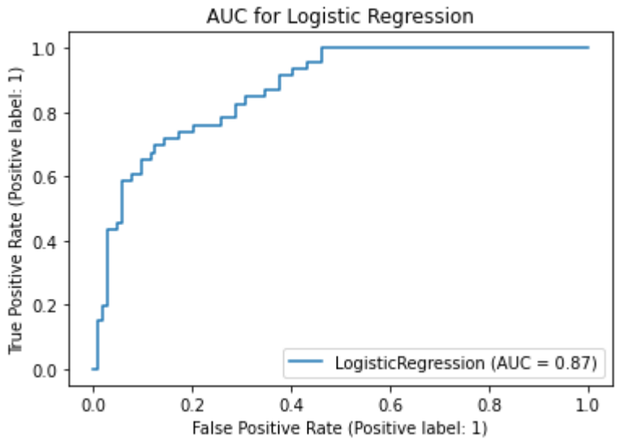
```
ROC_Curve(model=model_KNN,X_train=X_train_scalled, X_test=X_test_scalled,
            y_train=Y_train, y_test=Y_test)
plt.title("AUC for KNN");
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.80       104
           1       0.54      0.57      0.55        46

    accuracy                           0.72       150
   macro avg       0.67      0.68      0.67       150
weighted avg       0.72      0.72      0.72       150
```



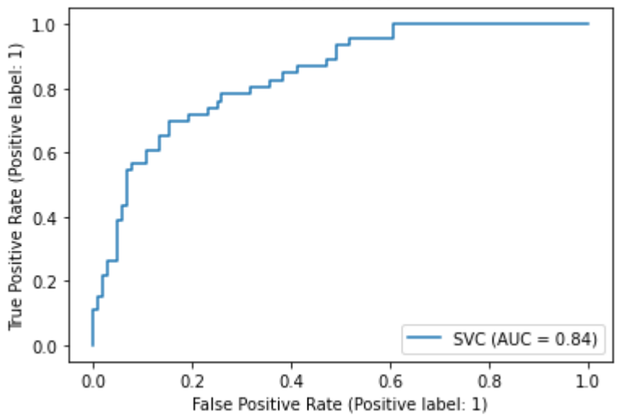
In [152…
```
ROC_Curve(model=model_Logistic,X_train=X_train_scalled, X_test=X_test_scalled,
          y_train=Y_train, y_test=Y_test)
plt.title("AUC for Logistic Regression");
```

	precision	recall	f1-score	support
0	0.85	0.88	0.87	104
1	0.71	0.65	0.68	46
accuracy			0.81	150
macro avg	0.78	0.77	0.77	150
weighted avg	0.81	0.81	0.81	150



In [153…
```
ROC_Curve(model=model_SVC,X_train=X_train_scalled, X_test=X_test_scalled,
          y_train=Y_train, y_test=Y_test)
plt.title("AUC for SVM");
```

	precision	recall	f1-score	support
0	0.85	0.86	0.85	104
1	0.67	0.65	0.66	46
accuracy			0.79	150
macro avg	0.76	0.75	0.76	150
weighted avg	0.79	0.79	0.79	150



In [154…
```
df_scalled=MinMaxscaler.fit_transform(new_df1)
df_scalled.shape,y_train.shape
```

Out[154… ((749, 6), (749, 1))

In [155…
```
X_train_scalled.shape,Y_train.shape
```

Out[155… ((599, 6), (599, 1))

Hyper parameter tuning for Logistic Regression

In [ ]:
```
```

In [150…
```
#Hyperparameter tuning
# define model/create instance
lr=LogisticRegression()
#tuning weight for minority class then weight for majority class will be 1-weight of minority class
#Setting the range for class weights
weights = np.linspace(0.0,0.99,500)
#specifying all hyperparameters with possible values

param= {'C': [0.1, 0.5, 1,10,15,20], 'penalty': ['l1', 'l2'],"class_weight":[{0:x ,1:1.0 -x} for x in weights],
       'solver':['newton-cg', 'lbfgs', 'liblinear']}
# create 5 folds
folds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)
#Gridsearch for hyperparam tuning
model= GridSearchCV(estimator= lr,param_grid=param,scoring="f1",cv=folds,return_train_score=True,verbose=True)
#train model to learn relationships between x and y
model.fit(X_train_scalled,Y_train)
```

Out[150…
```
Fitting 5 folds for each of 18000 candidates, totalling 90000 fits
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=42, shuffle=True),
             estimator=LogisticRegression(),
             param_grid={'C': [0.1, 0.5, 1, 10, 15, 20],
                         'class_weight': [{0: 0.0, 1: 1.0},
                                          {0: 0.0019839679358717435,
                                           1: 0.9980160320641283},
                                          {0: 0.003967935871743487,
                                           1: 0.9960320641282565},
                                          {0: 0.0059519038076152305,
                                           1: 0.9940480961923848},
                                          {0: 0.007935871743486974,
                                           1: 0.99206412825...
                                           1: 0.9523847695390781},
                                          {0: 0.04959919839679359,
```

```
                                    1: 0.9504008016032064},
                                   {0: 0.05158316633266533,
                                    1: 0.9484168336673346},
                                   {0: 0.05356713426853708,
                                    1: 0.9464328657314629},
                                   {0: 0.055551102204408814,
                                    1: 0.9444488977955912},
                                   {0: 0.05753507014028056,
                                    1: 0.9424649298597194}, ...],
                'penalty': ['l1', 'l2'],
                'solver': ['newton-cg', 'lbfgs', 'liblinear']},
       return_train_score=True, scoring='f1', verbose=True)
```

In [151…

```
# print best hyperparameters
print("Best score: ", model.best_score_)
print("Best hyperparameters: ", model.best_params_)
```

```
Best score:  0.6929245283018869
Best hyperparameters:  {'C': 0.5, 'class_weight': {0: 0.3134669338677355, 1: 0.6865330661322645}, 'penalty': 'l2', 'solver': 'newton-cg'}
```

In [156…

```
#Building Model again with best params
lr2=LogisticRegression(C=0.5, class_weight={0: 0.3134669338677355, 1: 0.6865330661322645}, penalty='l2', solver='newton-cg')
lr2.fit(X_train_scalled,Y_train)
```

Out[156…

```
LogisticRegression(C=0.5,
                   class_weight={0: 0.3134669338677355, 1: 0.6865330661322645},
                   solver='newton-cg')
```

Evaluating a classification model, beyond accuracy

Now we've got a tuned model, let's get some of the metrics we discussed before.

We want:

- ROC curve and AUC score - `plot_roc_curve()`
- Confusion matrix - `confusion_matrix()`
- Classification report - `classification_report()`
- Precision - `precision_score()`
- Recall - `recall_score()`
- F1-score - `f1_score()`

Scikit-Learn has these all built-in.

To access them, we'll have to use our model to make predictions on the test set. We can make predictions by calling `predict()` on a trained model and passing it the data you'd like to predict on.

We'll make predictions on the test data.

In [157…

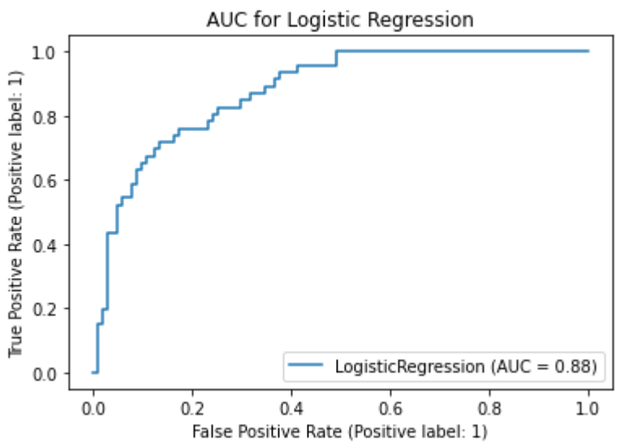
```
# predict probabilities on Test and take probability for class 1([:1])
y_pred_prob_test = lr2.predict_proba(X_test_scalled)[:, 1]

#predict labels on test dataset
y_pred_test = lr2.predict(X_test_scalled)

# create onfusion matrix
cm = confusion_matrix(Y_test, y_pred_test)

print("confusion Matrix is :\n",cm)
print("----------------------------------------------------------------------")

# ROC- AUC score
print("ROC-AUC score  test dataset:  ", roc_auc_score(Y_test,y_pred_prob_test))

#Precision score
print("precision score  test dataset:  ", precision_score(Y_test,y_pred_test))

#Recall Score
print("Recall score  test dataset:  ", recall_score(Y_test,y_pred_test))

#f1 score
print("f1 score  test dataset :  ", f1_score(Y_test,y_pred_test))
```

```
confusion Matrix is :
 [[73 31]
 [ 7 39]]
----------------------------------------------------------------------
ROC-AUC score  test dataset:   0.8779264214046822
precision score  test dataset:   0.5571428571428572
Recall score  test dataset:   0.8478260869565217
f1 score  test dataset :   0.6724137931034483
```

ROC Curve and AUC Scores

In [158…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(lr2, X_test_scalled, Y_test)
plt.title("AUC for Logistic Regression");
```



There is slight increase in AUC after hyperpameter tuning

Our model does far better than guessing which would be a line going from the bottom left corner to the top right corner, AUC = 0.5. But a perfect model would achieve an AUC score of 1.0, so there's still room for improvement.

Classification report

We can make a classification report using `classification_report()` and passing it the true labels as well as our models predicted labels.

A classification report will also give us information of the precision and recall of our model for each class.

In [157…

```
print(classification_report(Y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.70      0.79       104
           1       0.56      0.85      0.67        46

    accuracy                           0.75       150
```

```
   macro avg       0.73      0.77      0.73       150
weighted avg       0.80      0.75      0.76       150
```

- **Precision** - Indicates the proportion of positive identifications (model predicted class 1) which were actually correct. A model which produces no false positives has a precision of 1.0.
- **Recall** - Indicates the proportion of actual positives which were correctly classified. A model which produces no false negatives has a recall of 1.0.
- **F1 score** - A combination of precision and recall. A perfect model achieves an F1 score of 1.0.
- **Support** - The number of samples each metric was calculated on.
- **Accuracy** - The accuracy of the model in decimal form. Perfect accuracy is equal to 1.0.
- **Macro avg** - Short for macro average, the average precision, recall and F1 score between classes. Macro avg doesn't class imbalance into effort, so if you do have class imbalances, pay attention to this metric.
- **Weighted avg** - Short for weighted average, the weighted average precision, recall and F1 score between classes. Weighted means each metric is calculated with respect to how many samples there are in each class. This metric will favour the majority class (e.g. will give a high value when one class out performs another due to having more samples).

In our model we will try to increase **Recall** because **Recall=TP/(TP+FN)** and the problem we are dealing with is a medical problem where it is more hazardous to classify an person falsely as "Not patient" than to miss classify as "A patient". In other words we need to decrease **FN** than to **FP**.

In [ ]:

Cross-validation:

We will consider Cross-Validation as method to prevent overfitting

In standard k-fold cross-validation, we split the data into k subsets of approximately equal size, known as folds. Then, we iteratively train the algorithm by keeping one fold for testing and other k-1 fold for training. Each time a different set or group of folds is for validation. hat we'll do to make them more solid is calculate them using cross-validation.

We'll take the best model along with the best hyperparameters and use `cross_val_score()` along with various `scoring` parameter values.

`cross_val_score()` works by taking an estimator (machine learning model) along with data and labels. It then evaluates the machine learning model on the data and labels using cross-validation and a defined `scoring` parameter.

Let's remind ourselves of the best hyperparameters and then see them in action.

We are considering the Logistic Regression model (lr2) trained on the parameters decided from GridSearchCV

In [159…
```
df_scalled.shape,y_train.shape
```

Out[159…
```
((749, 6), (749, 1))
```

In [162…
```
# Instantiate best model with best hyperparameters (found with GridSearchCV)
clf =LogisticRegression(C=0.5, class_weight={0: 0.3134669338677355, 1: 0.6865330661322645}, penalty='l2', solver='newton-cg')
```

In [163…
```
# Cross-validated recall score
cv_recall = np.mean(cross_val_score(clf,
                                    df_scalled,
                                    y_train,
                                    cv=5, # 5-fold cross-validation
                                    scoring="recall")) # recall as scoring
cv_recall
```

Out[163…
```
0.8187782805429865
```

The final thing to check off the list of our model evaluation techniques is feature importance.

Feature importance

Feature importance is another way of asking, "which features contributing most to the outcomes of the model?"

Or for our problem, trying to predict heart disease using a patient's medical characterisitcs, which charateristics contribute most to a model predicting whether someone has heart disease or not?

Unlike some of the other functions we've seen, because how each model finds patterns in data is slightly different, how a model judges how important those patterns are is different as well. This means for each model, there's a slightly different way of finding which features were most important.

You can usually find an example via the Scikit-Learn documentation or via searching for something like "[MODEL TYPE] feature importance", such as, "random forest feature importance".

Since we're using `LogisticRegression`, we'll look at one way we can calculate feature importance for it.

To do so, we'll use the `coef_` attribute. Looking at the Scikit-Learn documentation for `LogisticRegression`, the `coef_` attribute is the coefficient of the features in the decision function.

We can access the `coef_` attribute after we've fit an instance of `LogisticRegression`.

In [164…
```
lr2.coef_
```

Out[164…
```
array([[2.60716713, 0.18172633, 1.49135988, 1.10048602, 0.93673485,
        0.4619436 ]])
```

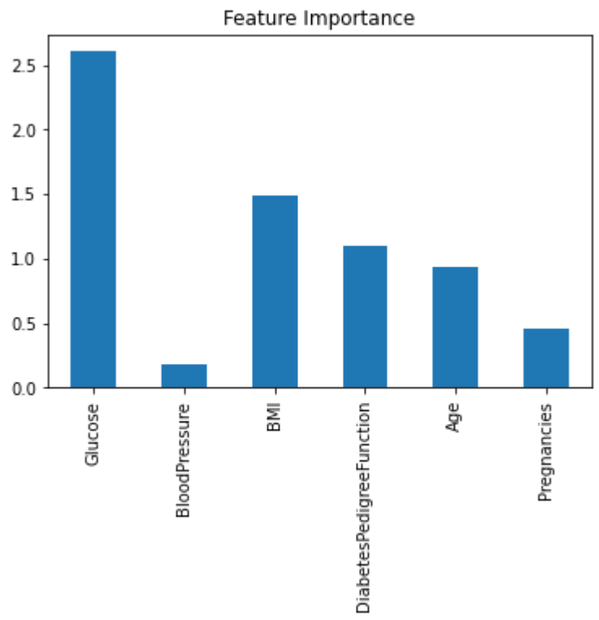
Looking at this it might not make much sense. But these values are how much each feature contributes to how a model makes a decision on whether patterns in a sample of patients health data leans more towards having heart disease or not.

Even knowing this, in it's current form, this `coef_` array still doesn't mean much. But it will if we combine it with the columns (features) of our dataframe.

In [165…
```
# Match features to columns
# Match features to columns
features_dict = dict(zip(x1.columns, list(lr2.coef_[0])))
features_dict
```

Out[165…
```
{'Glucose': 2.6071671320872154,
 'BloodPressure': 0.1817263269843448,
 'BMI': 1.4913598778692565,
 'DiabetesPedigreeFunction': 1.1004860240112164,
 'Age': 0.936734850176985,
 'Pregnancies': 0.4619436014654956}
```

In [166…
```
# Visualize feature importance
features_df = pd.DataFrame(features_dict, index=[0])
features_df.T.plot.bar(title="Feature Importance", legend=False);
```



It can be observed that in the model Blood Pressure is given less importance followed by Age. But the importance of none of the vairable is 0.

KNN Hyper parameter tuning

In [167…

```
###Hyperparameter for KNN
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2]
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
#Fit the model
best_model = clf.fit(X_train_scalled,Y_train)
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 2
Best n_neighbors: 7
```

In [168…

```
# print best hyperparameters
print("Best score: ", best_model.best_score_)
print("Best hyperparameters: ", best_model.best_params_)
```

```
Best score:  0.7845197740112994
Best hyperparameters:  {'leaf_size': 1, 'n_neighbors': 7, 'p': 2}
```

In [169…

```
knn_3=KNeighborsClassifier(leaf_size=1, n_neighbors=7, p=2)
knn_3.fit(X_train_scalled,Y_train)
```

Out[169…

```
KNeighborsClassifier(leaf_size=1, n_neighbors=7)
```

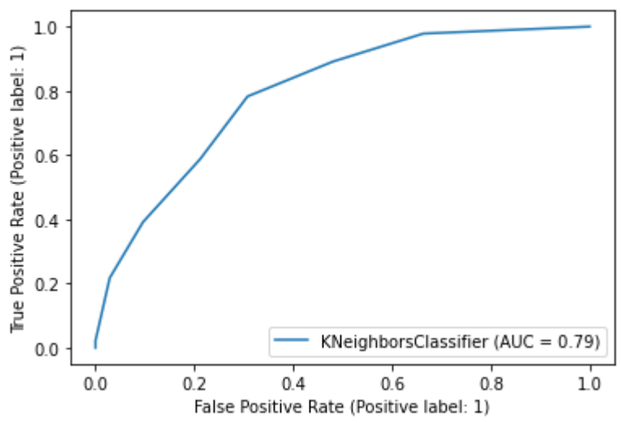
In [175…

```
# predict probabilities on Test and take probability for class 1([:1])
y_pred_prob_test_knn = knn_3.predict_proba(X_test_scalled)[:, 1]

#predict labels on test dataset
y_pred_test_knn = knn_3.predict(X_test_scalled)

# create onfusion matrix
cm_knn = confusion_matrix(Y_test, y_pred_test)

print("confusion Matrix is :\n",cm_knn)
print("-----------------------------------------------------------------------")

# ROC- AUC score
print("ROC-AUC score  test dataset:  ", roc_auc_score(Y_test,y_pred_prob_test_knn))

#Precision score
print("precision score  test dataset:  ", precision_score(Y_test,y_pred_test_knn))

#Recall Score
print("Recall score  test dataset:  ", recall_score(Y_test,y_pred_test_knn))

#f1 score
print("f1 score  test dataset :  ", f1_score(Y_test,y_pred_test_knn))
```

```
confusion Matrix is :
 [[73 31]
 [ 7 39]]
-----------------------------------------------------------------------
ROC-AUC score  test dataset:   0.7947324414715718
precision score  test dataset:   0.5510204081632653
Recall score  test dataset:   0.5869565217391305
f1 score  test dataset :   0.568421052631579
```

In [177…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(knn_3, X_test_scalled, Y_test);
```



Result of KNN is worse than Logistic Regression. Hence we are not going ahead with KNN.

Recall score for KNN is 0.5869565217391305

Recall score for Logistic Regression is 0.8478260869565217

Hyper parameter tuning for SVM

In [181…

```
model_SVC = SVC()
```

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid','Linear']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=True)
best_model_SVC=grid.fit(X_train_scalled,Y_train)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
```

In [182…

```
# print best hyperparameters
print("Best score: ", best_model_SVC.best_score_)
print("Best hyperparameters: ", best_model_SVC.best_params_)
```

```
Best score:  0.7696218487394958
Best hyperparameters:  {'C': 100, 'gamma': 0.1, 'kernel': 'sigmoid'}
```

In [186…

```
SVM_3=SVC(C=100, gamma= 0.1, kernel= 'sigmoid',probability=True)
SVM_3.fit(X_train_scalled,Y_train)
```

Out[186… `SVC(C=100, gamma=0.1, kernel='sigmoid', probability=True)`

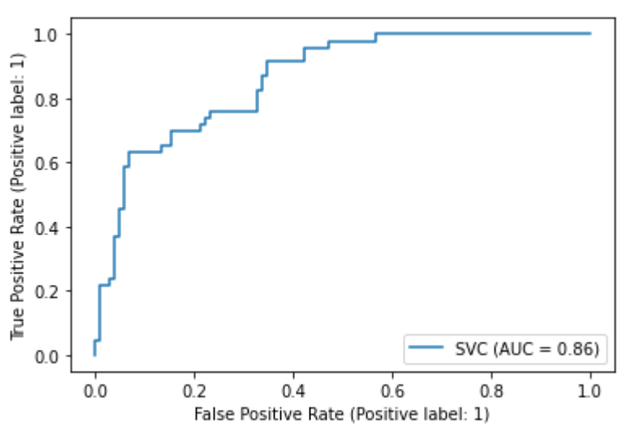
In [187…

```
# predict probabilities on Test and take probability for class 1([:1])
y_pred_prob_test = SVM_3.predict_proba(X_test_scalled)[:, 1]

#predict labels on test dataset
y_pred_test = SVM_3.predict(X_test_scalled)

# create onfusion matrix
cm = confusion_matrix(Y_test, y_pred_test)

print("confusion Matrix is :\n",cm)
print("-----------------------------------------------------------------------")

# ROC- AUC score
print("ROC-AUC score  test dataset:  ", roc_auc_score(Y_test,y_pred_prob_test))

#Precision score
print("precision score  test dataset:  ", precision_score(Y_test,y_pred_test))

#Recall Score
print("Recall score  test dataset:  ", recall_score(Y_test,y_pred_test))

#f1 score
print("f1 score  test dataset :  ", f1_score(Y_test,y_pred_test))
```

```
confusion Matrix is :
 [[90 14]
 [17 29]]
-----------------------------------------------------------------------
ROC-AUC score  test dataset:   0.8620401337792641
precision score  test dataset:   0.6744186046511628
Recall score  test dataset:   0.6304347826086957
f1 score  test dataset :   0.651685393258427
```

In [188…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(SVM_3, X_test_scalled, Y_test);
```



In [189…

```
# Instantiate best model with best hyperparameters (found with GridSearchCV)
clf_SVM =SVC(C=100, gamma= 0.1, kernel= 'sigmoid',probability=True)
```

In [190…

```
# Cross-validated recall score
cv_recall_SVM = np.mean(cross_val_score(clf_SVM,
                                        df_scalled,
                                        y_train,
                                        cv=5, # 5-fold cross-validation
                                        scoring="recall")) # recall as scoring
cv_recall_SVM
```

Out[190… `0.5791101055806939`

Out of SVM, KNN and Logistic Regression ultimately our analysis shows that the Logistic Regression is the best algorithim to handle the our problem.

Submitted by:Mintu Medhi

In [ ]: