4222-SURYA GROUP OF INSTITUTIONS

VIKRAVANDI-605652

NAAN MUDHALVAN PROJECT

SUBJECT CODE: SB3001

COURSE NAME: EXPERIENCE BASED PRACTICAL LEARNING

AI BASED DIABETES PREDICTION SYSTEM

PRESENTED BY

NAME: K.KAVIRAJAN

REG.NO:422221106011

DEPT:ECE 3RD YEAR

AI-based-diabetes-prediction-system:

INTRODUCTION:

Welcome to the future of healthcare with our AI-based Diabetes Prediction System. Harnessing the power of artificial intelligence, our innovative solution is designed to revolutionize the way we approach diabetes detection and prevention. By analyzing vast amounts of data and employing advanced machine learning algorithms, our system provides accurate predictions, enabling early intervention and personalized care. Stay ahead of diabetes with our cutting-edge technology, leading the way towards a healthier tomorrow.

AI MODELS USED:

- Logistic Regression
- Decision tress
- Random forest

STEPS TAKEN:

DATA COLLECTION:

	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

When encountered with a data set, first we should analyze and "**get to know**" the data set.This step is necessary to familiarize with the data, to gain some understanding of the potential

features and to see if data cleaning is needed. First, we will import the necessary libraries and import our data set. We can observe the mentioned columns in the data set.

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as snsdiabetes = pd.read_csv('datasets/diabetes.csv')
diabetes.columns
```

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

We can examine the data set using the pandas' **head()** method**.**

```
diabetes.head()
```

```
print("Diabetes data set dimensions : {}".format(diabetes.shape))
```

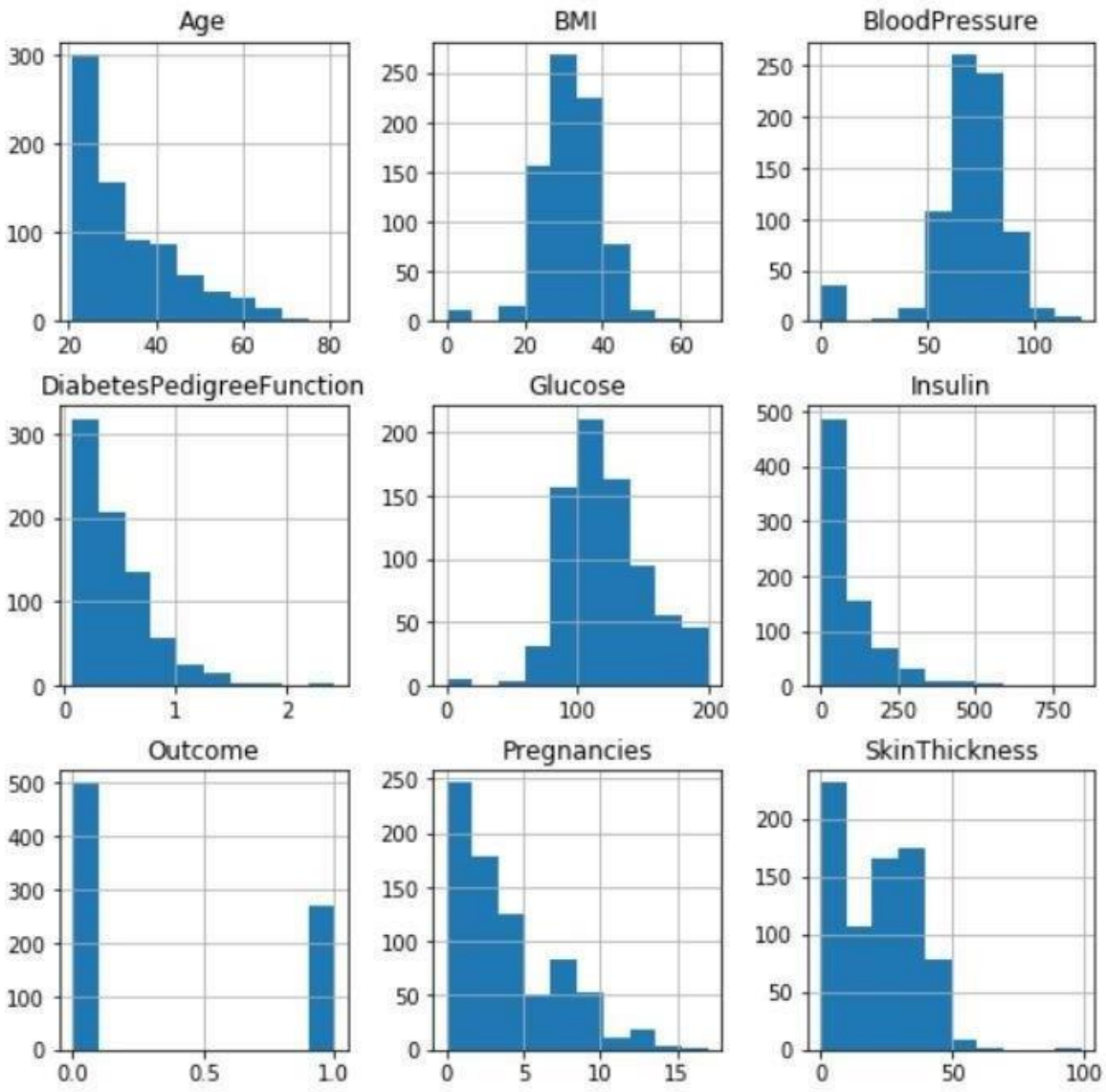
Diabetes data set dimensions : (768, 9) .

We can observe that the data set contain 768 rows and 9 columns. '*Outcome*' is the column which we are going to predict, which says if the patient is diabetic or not. 1 means the person is diabetic and 0 means a person is not. We can identify that out of the 768 persons, 500 are labeled as 0 (non-diabetic) and 268 as 1 (diabetic)**.**

```
diabetes.groupby('Outcome').size()
```

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

Visualization of data is an imperative aspect of data science. It helps to understand data and also to explain the data to another person. Python has several interesting visualization libraries such as Matplotlib, Seaborn, etc.visualization which is built on top of matplotlib, to find the data distribution of the features.



diabetes.groupby('Outcome').hist(figsize=(9, 9))

DATA PREPROCESSING:

The next phase of the machine learning work flow is data cleaning. Considered to be one of the crucial steps of the workflow, because it can make or break the model. There is a saying in machine learning **"Better data beats fancier algorithms"**, which suggests better data gives you better resulting models.

There are several factors to consider in the data cleaning process

1. Duplicate or irrelevant observations

2. Bad labeling of data, same category occurring multiple times

3. Missing or null data points.

4. Unexpected outliers.

```
diabetes.isnull().sum()
```

```
diabetes.isna().sum()
```

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

Blood pressure: By observing the data we can see that there are 0 values for blood pressure. And it is evident that the readings of the data set seem wrong because a living person cannot have a diastolic blood pressure of zero. By observing the data we can see 35 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.BloodPressure == 0].shape[0])Total :
35print(diabetes[diabetes.BloodPressure == 0].groupby('Outcome')['Age'].count())Outcome
0   19
1   16
Name: Age, dtype: int64
```

Plasma glucose levels: Even after fasting glucose levels would not be as low as zero. Therefore zero is an invalid reading. By observing the data we can see 5 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.Glucose == 0].shape[0])Total : 5print(diabetes[diabetes.Glucose
== 0].groupby('Outcome')['Age'].count())Total : 5
Outcome
0   3
1   2
Name: Age, dtype: int64
```

Skin Fold Thickness: For normal people, skin fold thickness can't be less than 10 mm better yet zero. Total count where value is 0: 227.

```
print("Total : ", diabetes[diabetes.SkinThickness == 0].shape[0])Total :
227print(diabetes[diabetes.SkinThickness == 0].groupby('Outcome')['Age'].count())Outcome
0   139
```

1 88
Name: Age, dtype: int64
BMI: Should not be 0 or close to zero unless the person is really underweight which could be life-threatening.
print("Total : ", diabetes[diabetes.BMI == 0].shape[0])Total : 11print(diabetes[diabetes.BMI == 0].groupby('Outcome')['Age'].count())Outcome
0 9
1 2
Name: Age, dtype: int64
Insulin: In a rare situation a person can have zero insulin but by observing the data, we can find that there is a total of 374 counts.
print("Total : ", diabetes[diabetes.Insulin == 0].shape[0])Total : 374print(diabetes[diabetes.Insulin == 0].groupby('Outcome')['Age'].count())Outcome
0 236
1 138
Name: Age, dtype: int64

Here are several ways to handle invalid data values :

1. Ignore/remove these cases: This is not actually possible in most cases because that would mean losing valuable information. And in this case “skin thickness” and “insulin” columns mean to have a lot of invalid points. But it might work for “BMI”, “glucose ”and “blood pressure” data points.

2. Put average/mean values: This might work for some data sets, but in our case putting a mean value to the blood pressure column would send a wrong signal to the model.

3. Avoid using features: It is possible to not use the features with a lot of invalid values for the model. This may work for “skin thickness” but it's hard to predict that.

4. We will remove the rows which the “BloodPressure”, “BMI” and “Glucose” are zero.
5. diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) & (diabetes.Glucose != 0)]print(diabetes_mod.shape)(724, 9)

MODEL SELECTION:

Feature Selection is the process of transforming the gathered data into features that better represent the problem that we are trying to solve to the model, to improve its performance and accuracy.Feature selection creates more input features from the existing features and also combines several features to produce more intuitive features to feed to the model.
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']X = diabetes_mod[feature_names]
y = diabetes_ mod.

MODEL SELECTION:

Model selection or algorithm selection phase is the most exciting and the heart of machine learning. It is the phase where we select the model which performs best for the data set at hand .First, we will be calculating the **"Classification Accuracy (Testing Accuracy)"** of a given set of classification models with their default parameters to determine which model performs better with the diabetes data set. We will import the necessary libraries for the notebook. We import 7 classifiers namely **K-Nearest Neighbours , Support Vector Classifier, Logistic Regression, Gaussian Naive Bayes, Random Forest, and Gradient Boost** to be contenders for the best classifier.

```
. neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from  sklearn.tree  import  DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn models = []models.append(('KNN', KNeighborsClassifier()))
models.append(('SVC', SVC()))
models.append(('LR', LogisticRegression()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
```

EVALUATION:

It is a general practice to avoid training and testing on the same data. The reasons are that the goal of the model is to predict **out-of-sample data**, and the model could be overly complex leading to **overfitting**. To avoid the aforementioned problems, there are two precautions.

1. Train/Test Split
2. K-Fold Cross-Validation

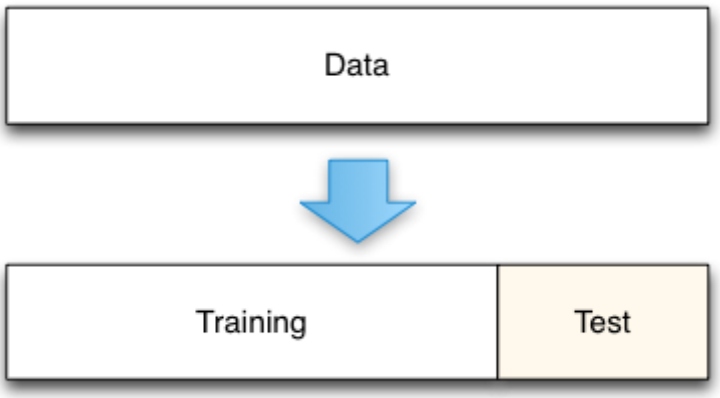
We will import *"train_test_split"* for train/test split **and *"cross_val_score" for*** *k-fold cross-validation.* ***"accuracy_score"*** **is to evaluate the accuracy of the model in the train/test split method.**

TRAIN/TEST SPLIT:

This method split the data set into two portions: a **training set** and a **testing set**. The **training set** is used to train the model. And the **testing set** is used to test the model, and evaluate the accuracy.

Pros : *But, train/test split is still useful because of its* ***flexibility and speed***

Cons : *Provides a* ***high-variance estimate*** *of out-of-sa*mple `accuracy`




```
X_train, X_test, y_train, y_test = train_test_split(X, y,
stratify = diabetes_mod.Outcome, random_state=0)
```

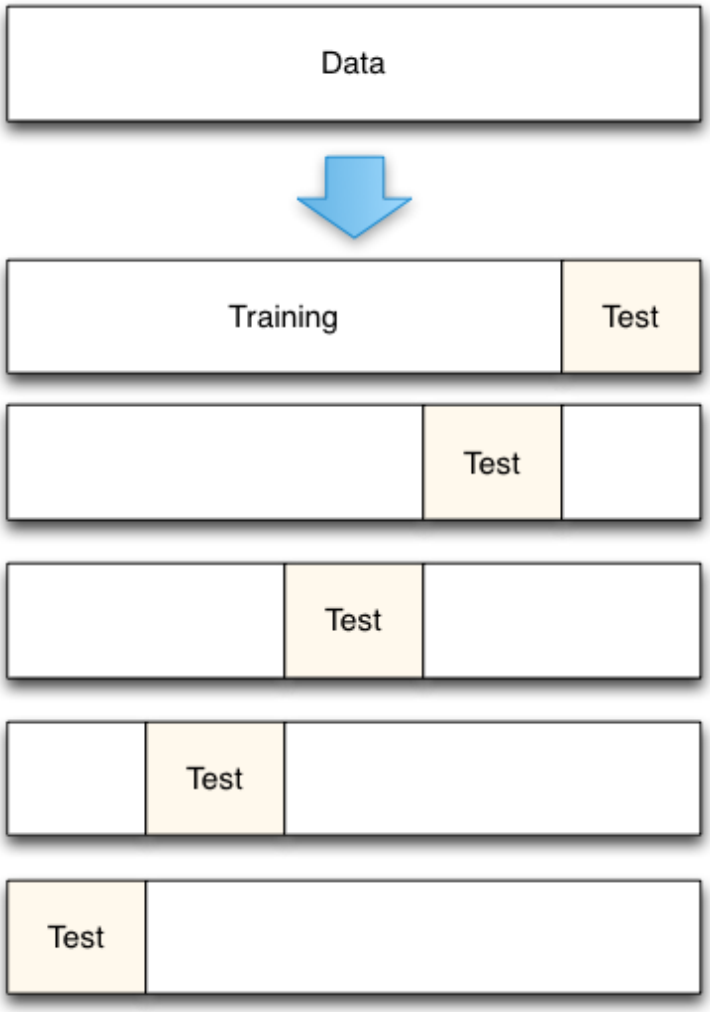
Then we fit each model in a loop and calculate the accuracy of the respective model using the *"accuracy_score"*.

```
names = []
scores = []for name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
    names.append(name)tr_split = pd.DataFrame({'Name': names,
'Score': scores})
print(tr_split)
```

	Name	Score
0	KNN	0.711521
1	SVC	0.656075
2	LR	0.776440
3	DT	0.681327
4	GNB	0.755681
5	RF	0.739165
6	GB	0.765442

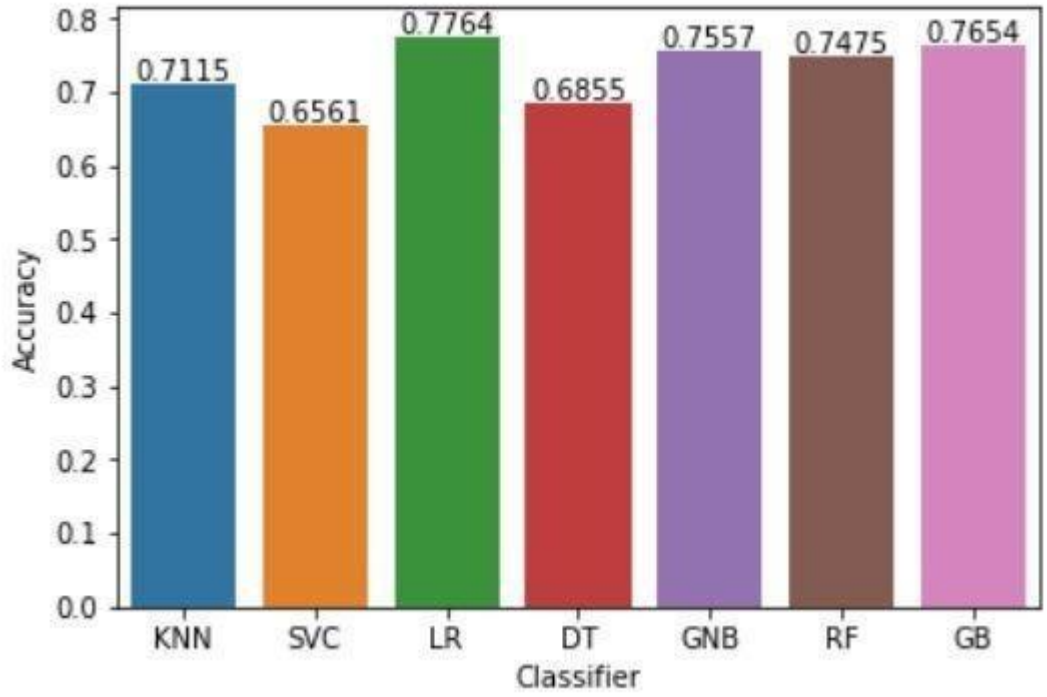
K-FOLD CROSS VALIDATION:

```
names = []
scores = []for name, model in models:

    kfold = KFold(n_splits=10, random_state=10)
    score = cross_val_score(model, X, y, cv=kfold,
scoring='accuracy').mean()

    names.append(name)
    scores.append(score)kf_cross_val = pd.DataFrame({'Name':
names, 'Score': scores})
print(kf cross val)
```



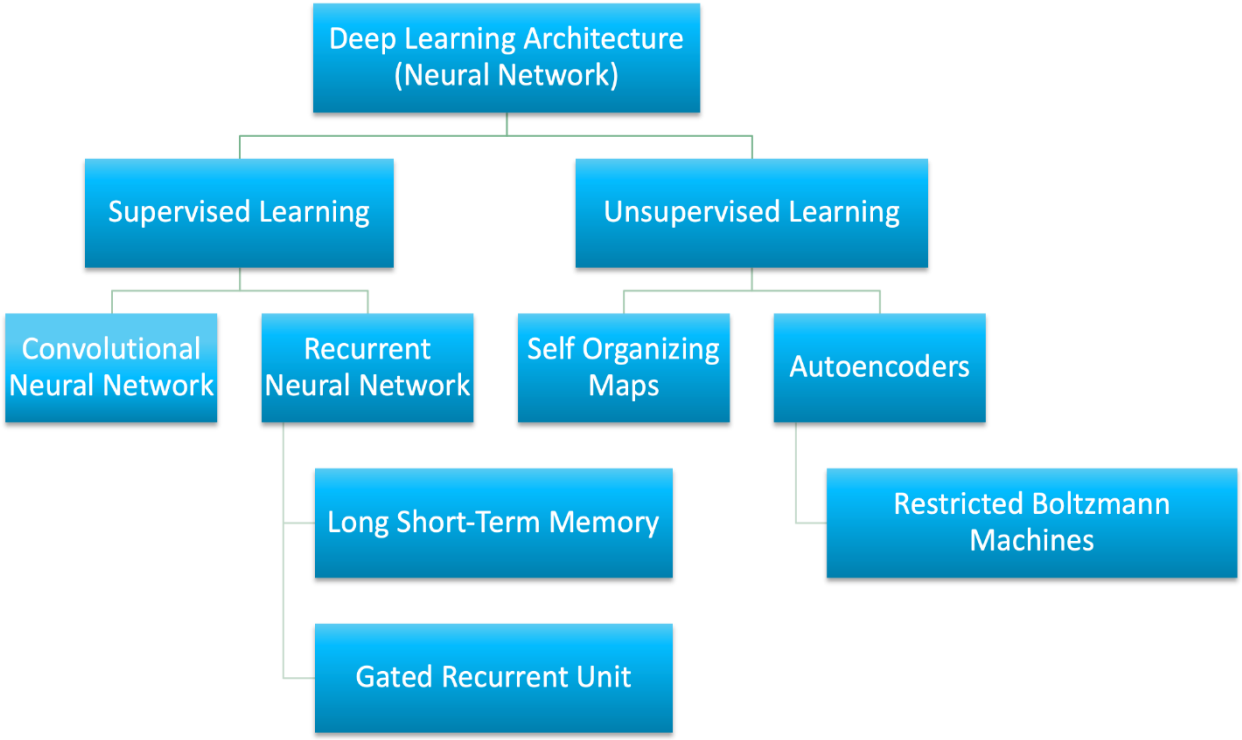
	Name	Score
0	KNN	0.711521
1	SVC	0.656075
2	LR	0.776440
3	DT	0.685494
4	GNB	0.755681
5	RF	0.747519
6	GB	0.765442

```
axis = sns.barplot(x = 'Name', y = 'Score', data = kf_cross_val)
axis.set(xlabel='Classifier', ylabel='Accuracy')for p in
axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005,
'{:1.4f}'.format(height), ha="center")

plt.show()
```



We can see the Logistic Regression, Gaussian Naive Bayes, Random Forest and Gradient Boosting have performed better than the rest. From the base level we can observe that the Logistic Regression performs better than the other algorithms.

DEEP LEARNING ARCHITECTURE:



Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing dataset

```
dataset = pd.read_csv('../input/diabetes-data-set/diabetes.csv')
```

Viewing the dataset, its dimensions, features and statistical Summary

```
dataset.head()
```

	Pregnancies	Glucose	BP	Skin thickness	Insulin	BMI	Diabetes pedigree function	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

```
dataset.shape
```

```
(768, 9)
```

```
dataset.info()
```

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

```
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigree
     Function                  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
```

```
dataset.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

Detecting null values

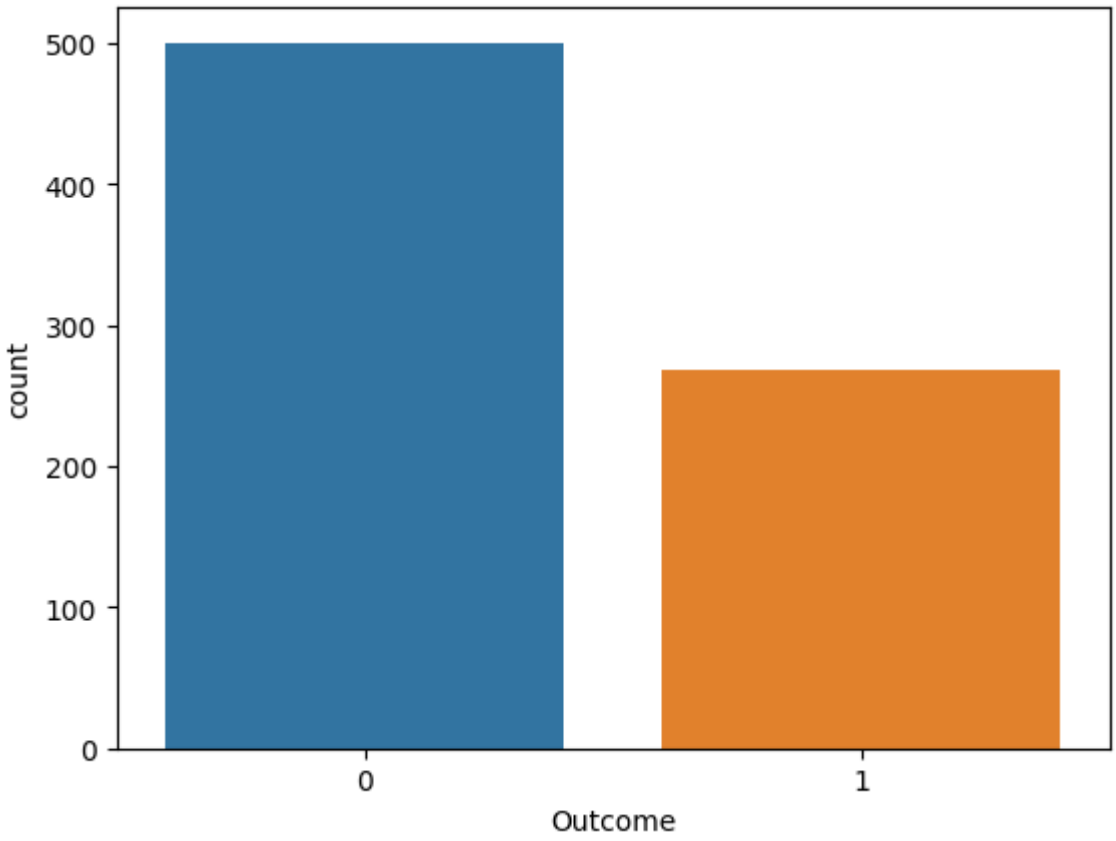
```
dataset.isnull().sum()
```

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

Data Visualization

```
sns.countplot(x = 'Outcome',data = dataset)
```

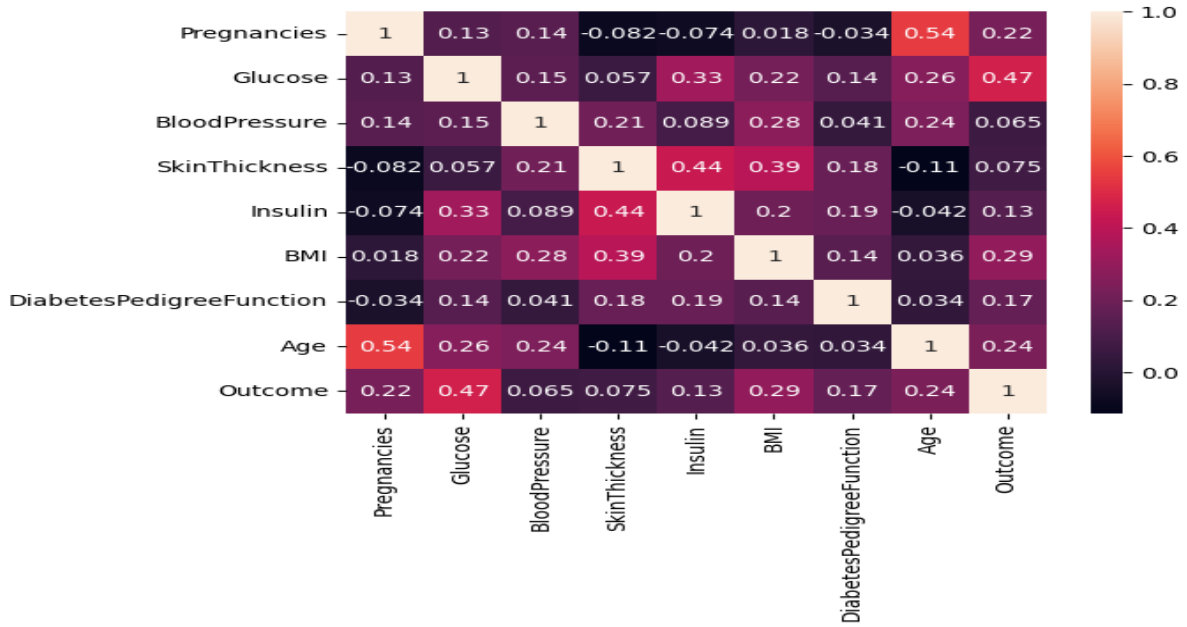
```
<Axes: xlabel='Outcome', ylabel='count'>
```




```
# Pairplot
sns.pairplot(data = dataset, hue = 'Outcome')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layo
ut has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```


```
# Heatmap
sns.heatmap(dataset.corr(), annot = True)
plt.show()
```



Processing the Data

```
# Replacing zero values with NaN
dataset_new = dataset
dataset_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = dataset_new[[
"Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].replace(0, np.NaN)
```

linkcode

```
# Count of NaN
dataset_new.isnull().sum()
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigree
Function                    0
Age                         0
Outcome                     0
dtype: int64
```

```
# Replacing NaN with mean values
dataset_new["Glucose"].fillna(dataset_new["Glucose"].mean(), inplace = True)
dataset_new["BloodPressure"].fillna(dataset_new["BloodPressure"].mean(), inplace = True)
dataset_new["SkinThickness"].fillna(dataset_new["SkinThickness"].mean(), inplace = True)
dataset_new["Insulin"].fillna(dataset_new["Insulin"].mean(), inplace = True)
dataset_new["BMI"].fillna(dataset_new["BMI"].mean(), inplace = True)
```

```
dataset_new.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
```

```
DiabetesPedigree
Function                 0
Age                      0
Outcome                  0
dtype: int64
```

Logistic Regression

```
y = dataset_new['Outcome']
X = dataset_new.drop('Outcome', axis=1)
```

```
# Splitting X and Y
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.20, random_state = 42, strat
ify = dataset_new['Outcome'] )
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
y_predict = model.predict(X_test)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
y_predict = model.predict(X_test)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
y_predict
```

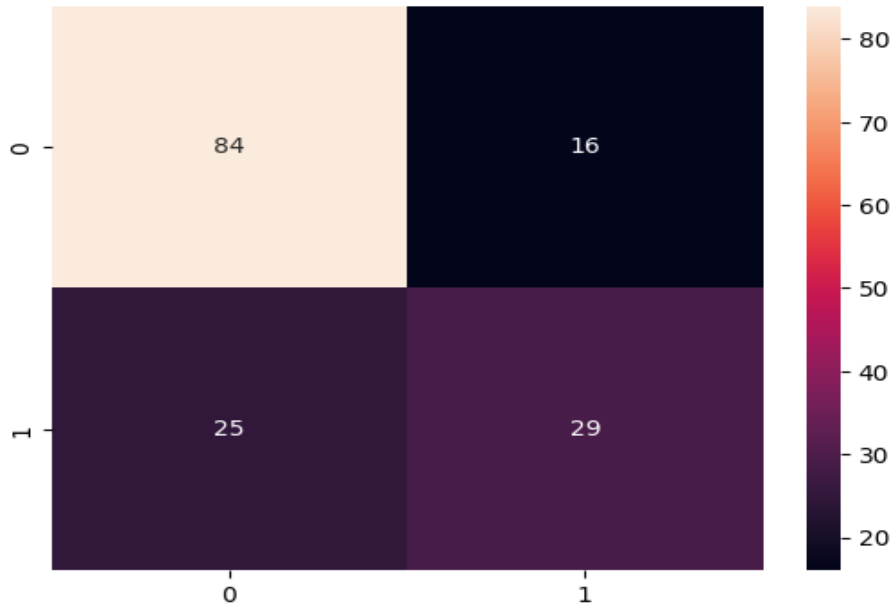
```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    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

```
# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, y_predict)
cm
```

```
array([[84, 16],
       [25, 29]])
```

```
# Heatmap of Confusion matrix
sns.heatmap(pd.DataFrame(cm), annot=True)
```

```
<Axes: >
```




```
from sklearn.metrics import accuracy_score
accuracy =accuracy_score(Y_test, y_predict)
accuracy
```

```
0.7337662337662337
```

```
y_predict = model.predict([[1,148,72,35,79.799,33.6,0.627,50]])
print(y_predict)
if y_predict==1:
    print("Diabetic")
else:
    print("Non Diabetic")
```

```
[1]
Diabetic
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does not have val
id feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

Import Required Libraries:

```
# Ignore warning messages to prevent them from being displayed during code execution
import warnings
warnings.filterwarnings('ignore')
```

```
import numpy as np    # Importing the NumPy library for linear algebra operations
import pandas as pd   # Importing the Pandas library for data processing and CSV file handling
```

```
import os
```

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import seaborn as sns                    # Importing the Seaborn library for statistical data visualization
import matplotlib.pyplot as plt          # Importing the Matplotlib library for creating plots and visualizations
import plotly.express as px              # Importing the Plotly Express library for interactive visualizations
```

Load and Prepare Data:

```
df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```

UnderStanding the Variables:

```
df.head(10)
```

	pregnancies	glucose	BP	Skin thickness	insulin	BMI	Diabetes pedigree function	age	outcome
1	10	148	96	35	0	0	33.6	26	1
2	3	85	0	29	0	00	26.6	30	0
3	9	183	40	0	35	0	23.3	33	1
4	5	89	65	23	23	94	28.1	21	0
5	0	137	64	35	0	168	43.1	32	1
6	7	116	66	0	29	0	25.6	31	0
7	8	78	73	32	35	88	31.0	50	1

```
df.tail(10)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
758	1	106	76	0	0	37.5	0.197	26
759	6	190	92	0	0	35.5	0.278	66
760	2	88	58	26	16	28.4	0.766	22
761	9	170	74	31	0	44.0	0.403	43

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
762	9	89	62	0	0	22.5	0.142	33
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

```
df.sample(5)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
355	9	165	88	0	0	30.4	0.302	49
187	1	128	98	41	58	32.0	1.321	33
235	4	171	72	0	0	43.6	0.479	26

```
df.describe()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

<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

```
df.size
```

6912

df.shape

(768, 9)

Data Cleaning:

df.shape

(768, 9)

```
df=df.drop_duplicates()
df.shape
```

(768, 9)

Check null Values

df.isnull().sum()

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

There is no Missing Values present in the Data

df.columns

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

Check the number of Zero Values in Dataset:

print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])

No. of Zero Values in Glucose 5

print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])

No. of Zero Values in Glucose 5

print("No. of Zero Values in Blood Pressure ", df[df['BloodPressure']==0].shape[0])

No. of Zero Values in Blood Pressure 35

print("No. of Zero Values in SkinThickness ", df[df['SkinThickness']==0].shape[0])

No. of Zero Values in SkinThickness 227

print("No. of Zero Values in Insulin ", df[df['Insulin']==0].shape[0])

No. of Zero Values in Insulin 374

```
print("No. of Zero Values in BMI ", df[df['BMI']==0].shape[0])
```

```
No. of Zero Values in BMI  11
```

```
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
```

Replace zeroes with mean of that Columns:

print('No of zero Values in Glucose ', df[df['Glucose']==0].shape[0])

No of zero Values in Glucose 0
linkcode

```
df['BloodPressure']=df['BloodPressure'].replace(0, df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0, df['SkinThickness'].mean())
df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0, df['BMI'].mean())
```

Validate the Zero Values:

```
df.describe()
```

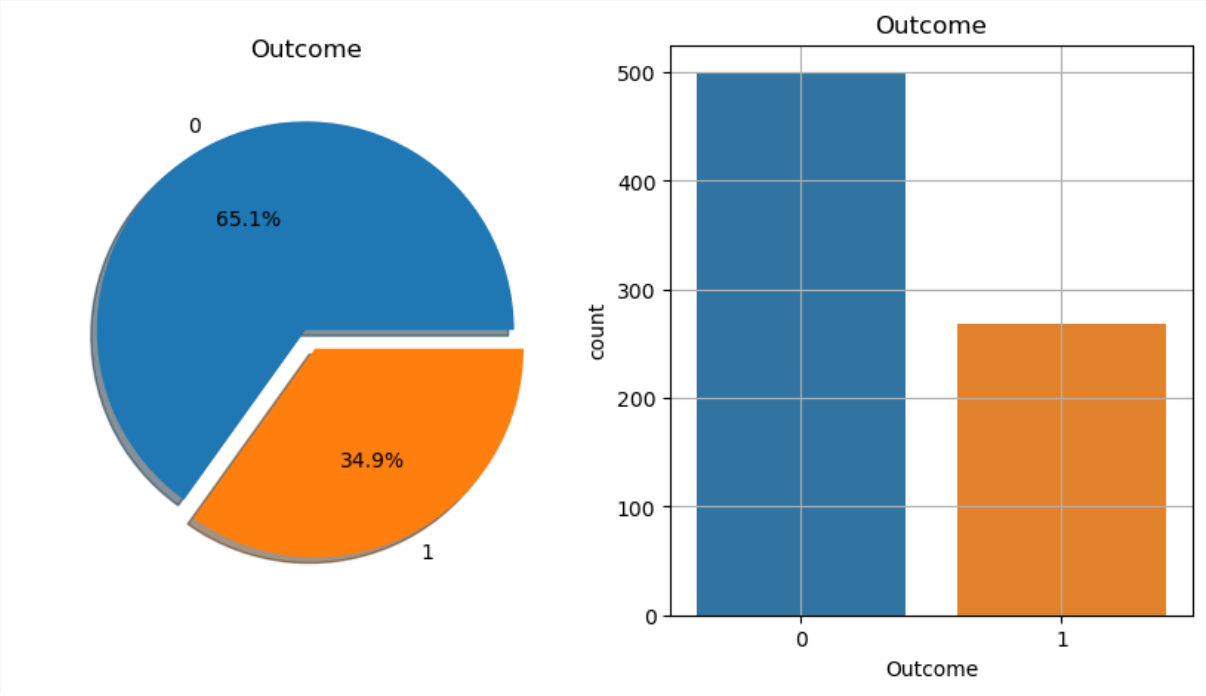
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Data Visualization:

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Assuming 'df' is your DataFrame containing the dataset
# If you haven't imported your dataset yet, import it here

# Create subplots
f, ax = plt.subplots(1, 2, figsize=(10, 5))

# Pie chart for Outcome distribution
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0], shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel(' ')

# Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of 'Outcome'
ax[1].set_title('Outcome')

# Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of 'Outcome'
ax[1].set_title('Outcome')

# Display class distribution
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)
# Adding grid and showing plots
plt.grid()
plt.show()
```

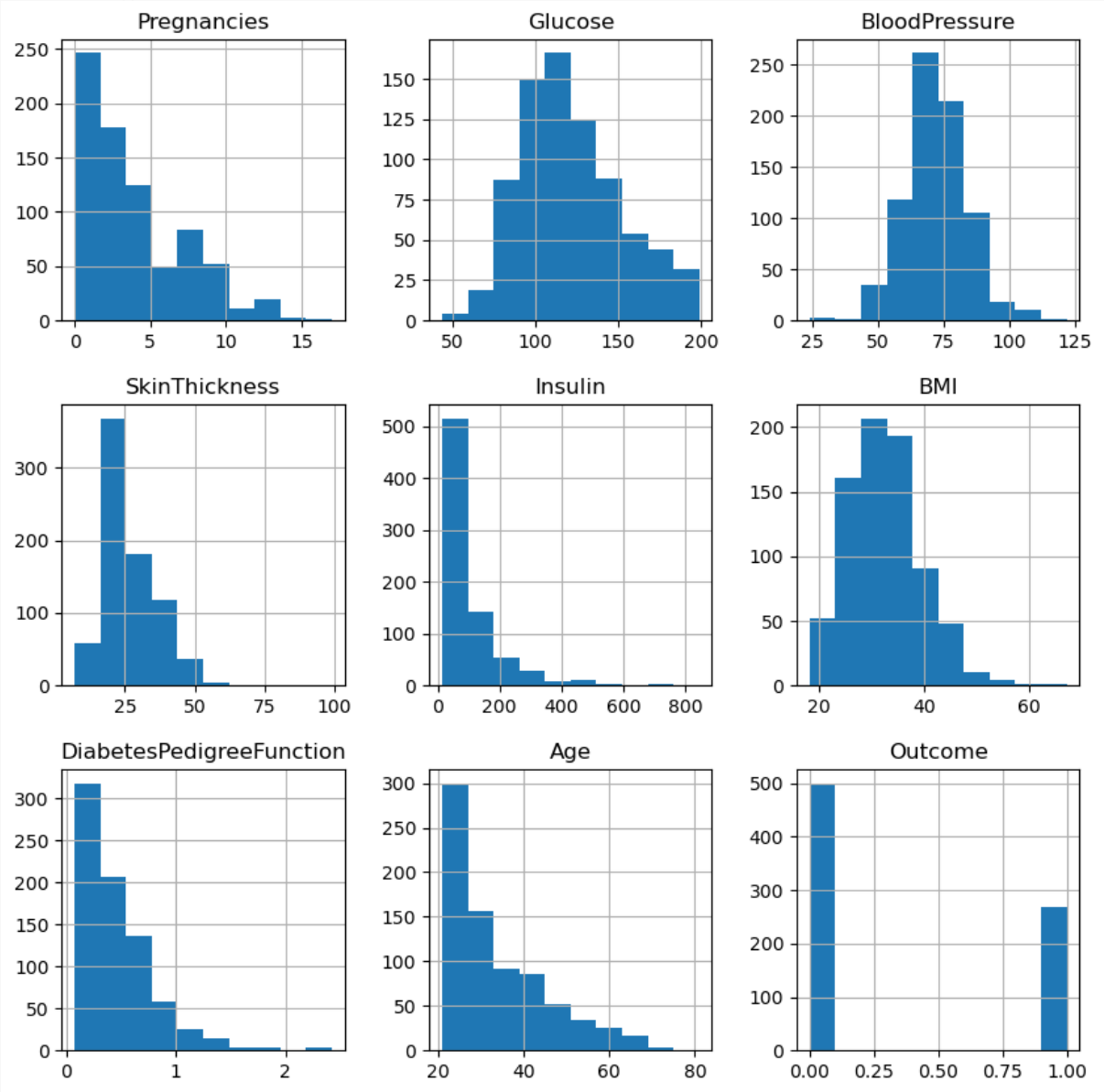
Negative (0): 500
Positive (1): 268



- 1 Represent --> Diabetes Positive
- 0 Represent --> Daibetes Negative

Histograms:

```
df.hist(bins=10, figsize=(10, 10))
plt.show()
```



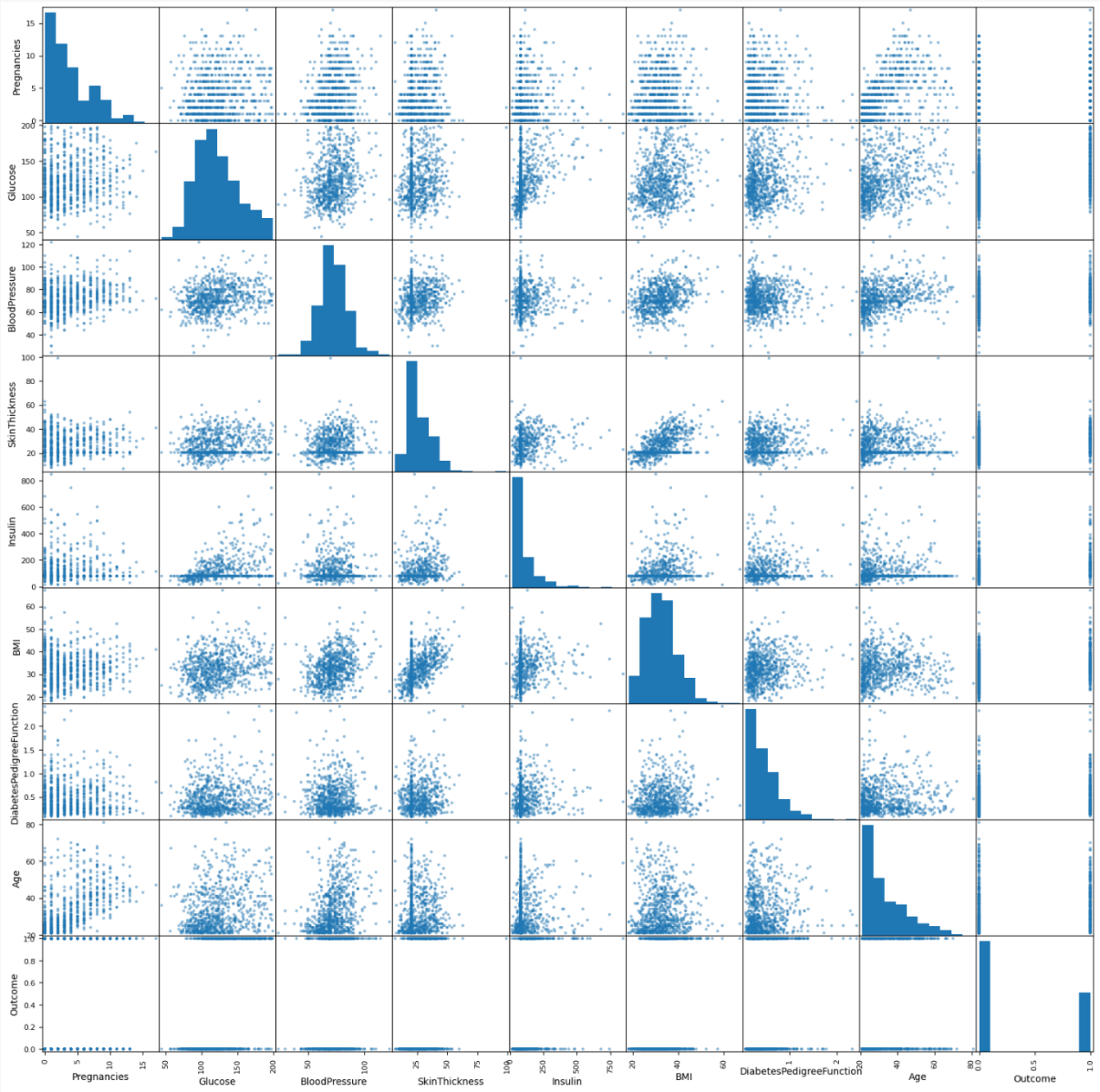
Scatter Plot:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize =(20, 20))
```

```
array([[<Axes: xlabel='Pregnancies', ylabel='Pregnancies'>,
        <Axes: xlabel='Glucose', ylabel='Pregnancies'>,
        <Axes: xlabel='BloodPressure', ylabel='Pregnancies'>,
        <Axes: xlabel='SkinThickness', ylabel='Pregnancies'>,
        <Axes: xlabel='Insulin', ylabel='Pregnancies'>,
        <Axes: xlabel='BMI', ylabel='Pregnancies'>,
        <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Pregnancies'>,
        <Axes: xlabel='Age', ylabel='Pregnancies'>,
        <Axes: xlabel='Outcome', ylabel='Pregnancies'>],
       [<Axes: xlabel='Pregnancies', ylabel='Glucose'>,
        <Axes: xlabel='Glucose', ylabel='Glucose'>,
        <Axes: xlabel='BloodPressure', ylabel='Glucose'>,
        <Axes: xlabel='SkinThickness', ylabel='Glucose'>,
        <Axes: xlabel='Insulin', ylabel='Glucose'>,
```

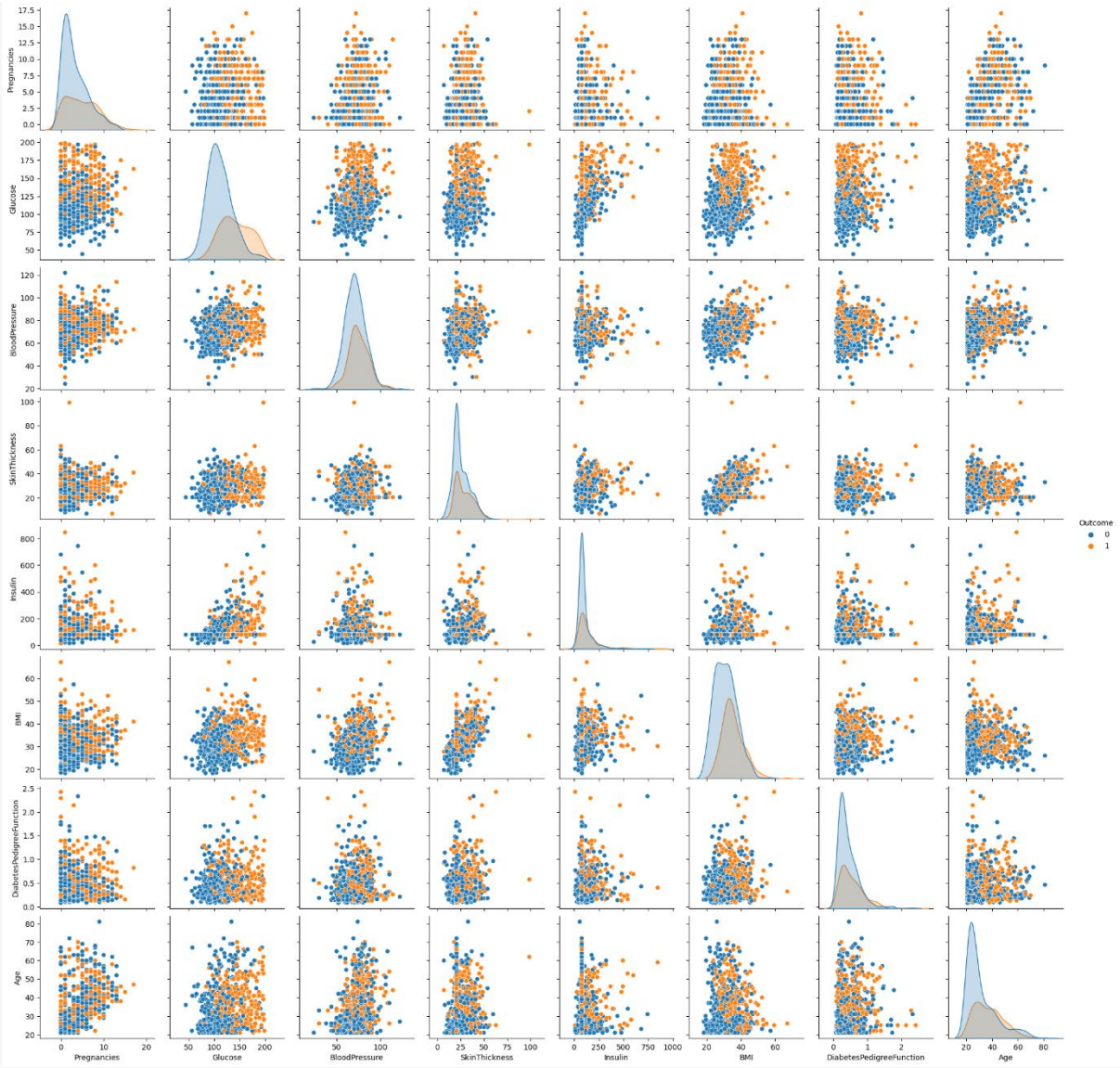
<Axes: xlabel='BMI', ylabel='Glucose'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Glucose'>,
<Axes: xlabel='Age', ylabel='Glucose'>,
<Axes: xlabel='Outcome', ylabel='Glucose'>],
[<Axes: xlabel='Pregnancies', ylabel='BloodPressure'>,
<Axes: xlabel='Glucose', ylabel='BloodPressure'>,
<Axes: xlabel='BloodPressure', ylabel='BloodPressure'>,
<Axes: xlabel='SkinThickness', ylabel='BloodPressure'>,
<Axes: xlabel='Insulin', ylabel='BloodPressure'>,
<Axes: xlabel='BMI', ylabel='BloodPressure'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='BloodPressure'>,
<Axes: xlabel='Age', ylabel='BloodPressure'>,
<Axes: xlabel='Outcome', ylabel='BloodPressure'>],
[<Axes: xlabel='Pregnancies', ylabel='SkinThickness'>,
<Axes: xlabel='Glucose', ylabel='SkinThickness'>,
<Axes: xlabel='BloodPressure', ylabel='SkinThickness'>,
<Axes: xlabel='SkinThickness', ylabel='SkinThickness'>,
<Axes: xlabel='Insulin', ylabel='SkinThickness'>,
<Axes: xlabel='BMI', ylabel='SkinThickness'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='SkinThickness'>,
<Axes: xlabel='Age', ylabel='SkinThickness'>,
<Axes: xlabel='Outcome', ylabel='SkinThickness'>],
[<Axes: xlabel='Pregnancies', ylabel='Insulin'>,
<Axes: xlabel='Glucose', ylabel='Insulin'>,
<Axes: xlabel='BloodPressure', ylabel='Insulin'>,
<Axes: xlabel='SkinThickness', ylabel='Insulin'>,
<Axes: xlabel='Insulin', ylabel='Insulin'>,
<Axes: xlabel='BMI', ylabel='Insulin'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Insulin'>,
<Axes: xlabel='Age', ylabel='Insulin'>,
<Axes: xlabel='Outcome', ylabel='Insulin'>],
[<Axes: xlabel='Pregnancies', ylabel='BMI'>,
<Axes: xlabel='Glucose', ylabel='BMI'>,
<Axes: xlabel='BloodPressure', ylabel='BMI'>,
<Axes: xlabel='SkinThickness', ylabel='BMI'>,
<Axes: xlabel='Insulin', ylabel='BMI'>,
<Axes: xlabel='BMI', ylabel='BMI'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='BMI'>,
<Axes: xlabel='Age', ylabel='BMI'>,
<Axes: xlabel='Outcome', ylabel='BMI'>],
[<Axes: xlabel='Pregnancies', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Glucose', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='BloodPressure', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='SkinThickness', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Insulin', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='BMI', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Age', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Outcome', ylabel='DiabetesPedigreeFunction'>],
[<Axes: xlabel='Pregnancies', ylabel='Age'>,
<Axes: xlabel='Glucose', ylabel='Age'>,
<Axes: xlabel='BloodPressure', ylabel='Age'>,
<Axes: xlabel='SkinThickness', ylabel='Age'>,
<Axes: xlabel='Insulin', ylabel='Age'>,
<Axes: xlabel='BMI', ylabel='Age'>,

```
        <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Age'>,
        <Axes: xlabel='Age', ylabel='Age'>,
<Axes: xlabel='Outcome', ylabel='Age'>],
       [<Axes: xlabel='Pregnancies', ylabel='Outcome'>,
        <Axes: xlabel='Glucose', ylabel='Outcome'>,
        <Axes: xlabel='BloodPressure', ylabel='Outcome'>,
        <Axes: xlabel='SkinThickness', ylabel='Outcome'>,
        <Axes: xlabel='Insulin', ylabel='Outcome'>,
        <Axes: xlabel='BMI', ylabel='Outcome'>,
        <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Outcome'>,
        <Axes: xlabel='Age', ylabel='Outcome'>,
        <Axes: xlabel='Outcome', ylabel='Outcome'>]], dtype=object)
```



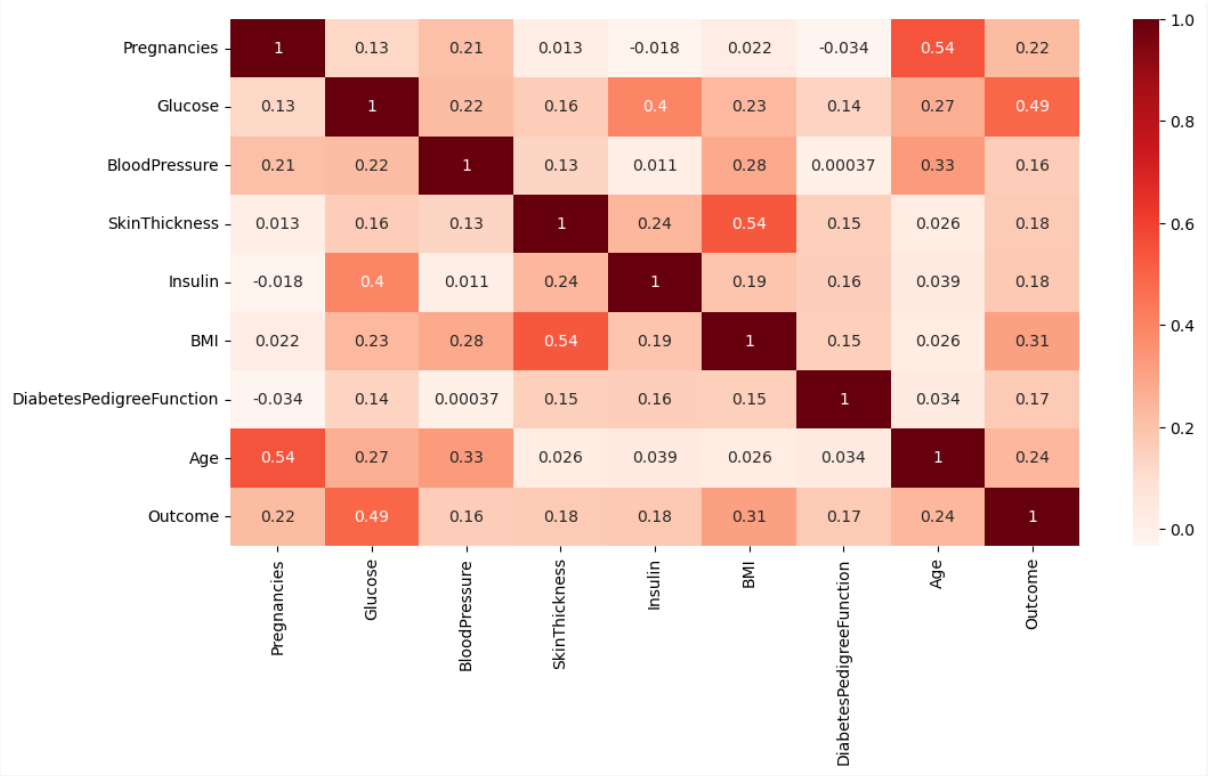
Pair plot:

```
sns.pairplot(data=df, hue='Outcome')
plt.show()
```




```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
```

 # Creating a heatmap of the correlation matrix for the columns in the DataFrame data




```
mean = df['Outcome'].mean()
# Calculating the mean value of the 'Outcome' column in the DataFrame data
mean
# Displaying the calculated mean value
```

0.3489583333333333

Split the DataFrame into X and y:

```
target_name='Outcome'

y=df[target_name]

X= df.drop(target_name, axis=1)

X.head()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288

```
y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Future Scalling**:**

```
# Standard Scaler:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2, random_state=7)
```

```
X_train.shape, y_train.shape
```

```
((154, 8), (154,))
```

Making prediction:

```
X_test.shape
```

```
(154, 8)
```

```
lr_pred=lr.predict(X_test)
```

```
lr_pred.shape
```

```
(154,)
```

linkcode

Decision Tree:

```
dt_pred=dt.predict(X_test)
```

linkcode

```
dt_pred.shape
```

```
# For Logistic Regression:
from sklearn.metrics import accuracy_score
print("Train Accuracy of Logistic Regression: ", lr.score(X_train, y_train)*100)
print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test, y_test)*100)
print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, lr_pred)*100)
```

```
Train Accuracy of Logistic Regression:  77.36156351791531
```

Accuracy (Test) Score of Logistic Regression: 77.27272727272727
Accuracy Score of Logistic Regression: 77.27272727272727

linkcode
 # For Decesion Tree:
print("Train Accuracy of Decesion Tree: ", dt.score(X_train, y_train)*100)
print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test, y_test)*100)
print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test, dt_pred)*100)

Train Accuracy of Decesion Tree: 100.0
Accuracy (Test) Score of Decesion Tree: 80.51948051948052
Accuracy Score of Decesion Tree: 80.51948051948052

from sklearn.metrics import precision_score
print("Precision Score is: ", precision_score(y_test, lr_pred)*100)
print("Micro Average Precision Score is: ", precision_score(y_test, lr_pred, average='micro')*100)
print("Macro Average Precision Score is: ", precision_score(y_test, lr_pred, average='macro')*100)
print("Weighted Average Precision Score is: ", precision_score(y_test, lr_pred, average='weighted')*100)
print("precision Score on Non Weighted score is: ", precision_score(y_test, lr_pred, average=None)*100)
Precision Score is: 75.0
Micro Average Precision Score is: 77.27272727272727
Macro Average Precision Score is: 76.5909090909091
Weighted Average Precision Score is: 77.00413223140497
precision Score on Non Weighted score is: [78.18181818 75.

print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))

Classification Report of Logistic Regression:
```
              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

True Positive Rate(TPR)

Recall = True Positive/True Positive + False Negative
Recall = TP/TP+FN

In [65]:

```
recall_score = TP/ float(TP+FN)*100
print('recall_score', recall_score)
```

recall_score 57.89473684210527

In [66]:

```
TP, FN
```

Out[66]:

```
(33, 24)
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_Score: ', recall_score(y_test, lr_pred)*100)
```

```
Recall or Sensitivity_Score:  57.89473684210527
```

In [69]:

```
linkcode
print("recall Score is: ", recall_score(y_test, lr_pred)*100)
print("Micro Average recall Score is: ", recall_score(y_test, lr_pred, average='micro')*100)
print("Macro Average recall Score is: ", recall_score(y_test, lr_pred, average='macro')*100)
print("Weighted Average recall Score is: ", recall_score(y_test, lr_pred, average='weighted')*10
0)
```

```
recall Score is:  57.89473684210527
Micro Average recall Score is:  77.27272727272727
Macro Average recall Score is:  73.27726532826912
Weighted Average recall Score is:  77.27272727272727
recall Score on Non Weighted score is:  [88.65979381 57.89473684]
```

```
print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digit
s=4))
```

```
Classification Report of Logistic Regression:
              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

ROC Curve& ROC AUC

```
# Area under Curve:
auc= roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of logistic Regression is ", auc)
```

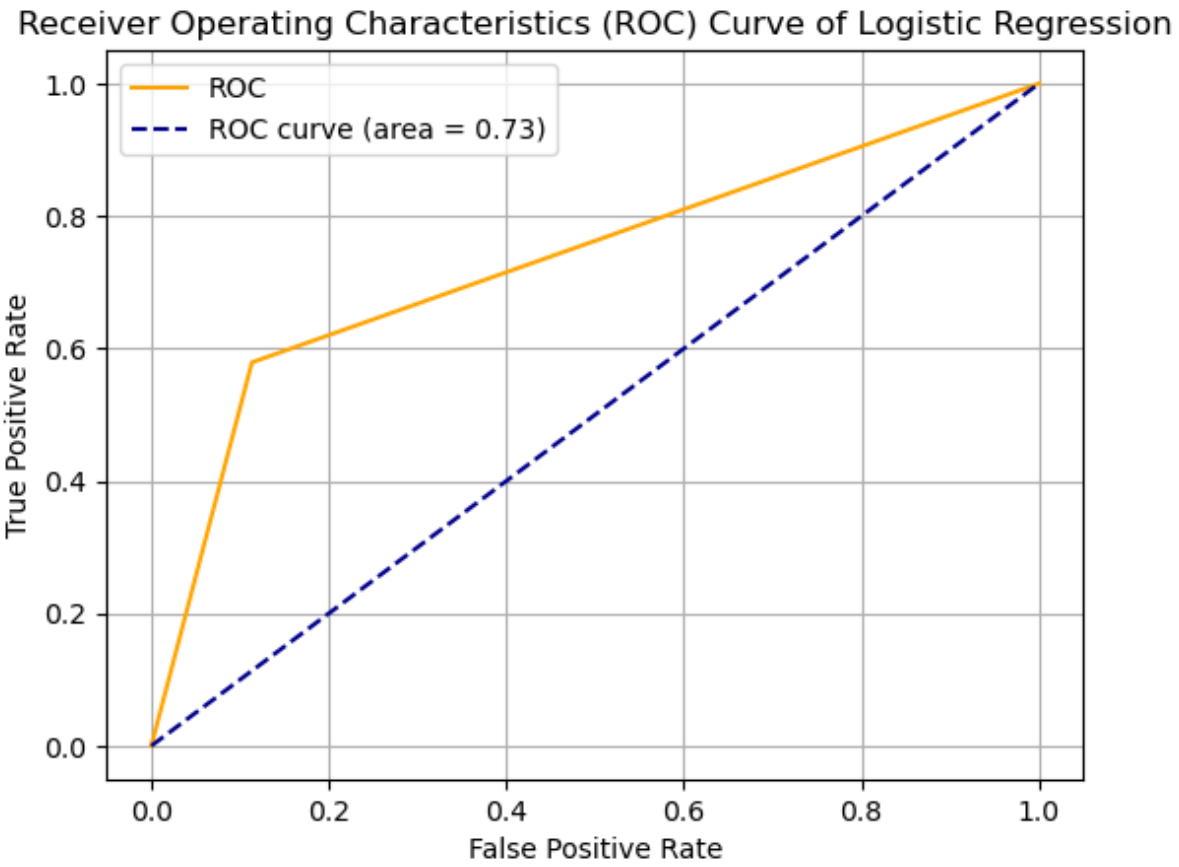
```
ROC AUC SCORE of logistic Regression is  0.7327726532826913
```

In [79]:

```
linkcode
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve (area = %0.2f)' % auc(fpr
, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic
Regression")
plt.legend()
```

```
plt.grid()
plt.show()
```



IMPORT LIBRARIES:

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

Scikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run

seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library that is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that support Computer Vision, Natural Language Processing(NLP), and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs.

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc..,

 # Import Dependencies

```
%matplotlib inline

# Start Python Imports
import math, time, datetime

import random as rd

# Data Manipulation
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import missingno as msno
import seaborn as sns
plt.style.use('seaborn-whitegrid')

# Preprocessing
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize

# Machine learning
import catboost
from sklearn.model_selection import train_test_split
from sklearn import model_selection, tree, preprocessing, metrics, linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from catboost import CatBoostClassifier, Pool, cv

# Let's be rebels and ignore warnings for now
import warnings
warnings.filterwarnings('ignore')
```

```
/tmp/ipykernel_20/160898924.py:16: MatplotlibDeprecationWarning: The seaborn s
tyles shipped by Matplotlib are deprecated since 3.6, as they no longer corres
pond to the styles shipped by seaborn. However, they will remain available as
'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn-whitegrid')
```

HELPING FUNCTIONS:

Activation Function: Activation functions introduce non-linearity into the neural networks, allowing them to learn complex patterns. Common activation functions include Sigmoid, Tanh, ReLU (Rectified Linear Unit), and Softmax.

Loss Functions (Cost Functions): Loss functions measure the difference between the predicted values and the actual values (labels) in the training data. They are used to train machine learning models by minimizing the error. Common loss functions include Mean Squared Error (MSE) for regression problems and Cross-Entropy Loss for classification problems.

Optimization Algorithms: Optimization algorithms are used to minimize the loss function during the training of machine learning models. Gradient Descent, Stochastic Gradient Descent (SGD), and variants like Adam and RMSprop are commonly used optimization algorithms.

Regularization Techniques: Regularization methods are used to prevent overfitting, which occurs when a model performs well on the training data but poorly on unseen data. L1 and L2 regularization add penalty terms to the loss function, encouraging the model to use simpler (smoother) solutions. Dropout is another technique where randomly selected neurons are ignored during training to prevent overfitting.

Data Preprocessing: Data preprocessing techniques include data cleaning, feature scaling, and feature engineering. Data is often normalized or standardized to ensure that all features have the same scale. Categorical variables are encoded into numerical values through techniques like one-hot encoding.

Cross-Validation: Cross-validation is a technique used to assess the performance and generalizability of a machine learning model. It involves dividing the dataset into multiple subsets (folds) and training the model on different subsets while evaluating it on the remaining data.

Evaluation Metrics: Evaluation metrics are used to measure the performance of machine learning models. Common evaluation metrics include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) for classification problems. For regression problems, metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared are used.

Feature Selection: Feature selection techniques help in selecting the most relevant features for the model, improving its performance and reducing overfitting. Common methods include Recursive Feature Elimination (RFE) and feature importance from tree-based models.

Ensemble Methods: Ensemble methods combine predictions from multiple machine learning models to improve overall performance. Bagging, Boosting, and Stacking are popular ensemble techniques.

Dimensionality Reduction: Dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are used to reduce the number of features while preserving the important information, making it easier to visualize and analyze the data.

These helping functions and techniques are fundamental to the practice of machine learning and are applied across various types of machine learning algorithms and models.

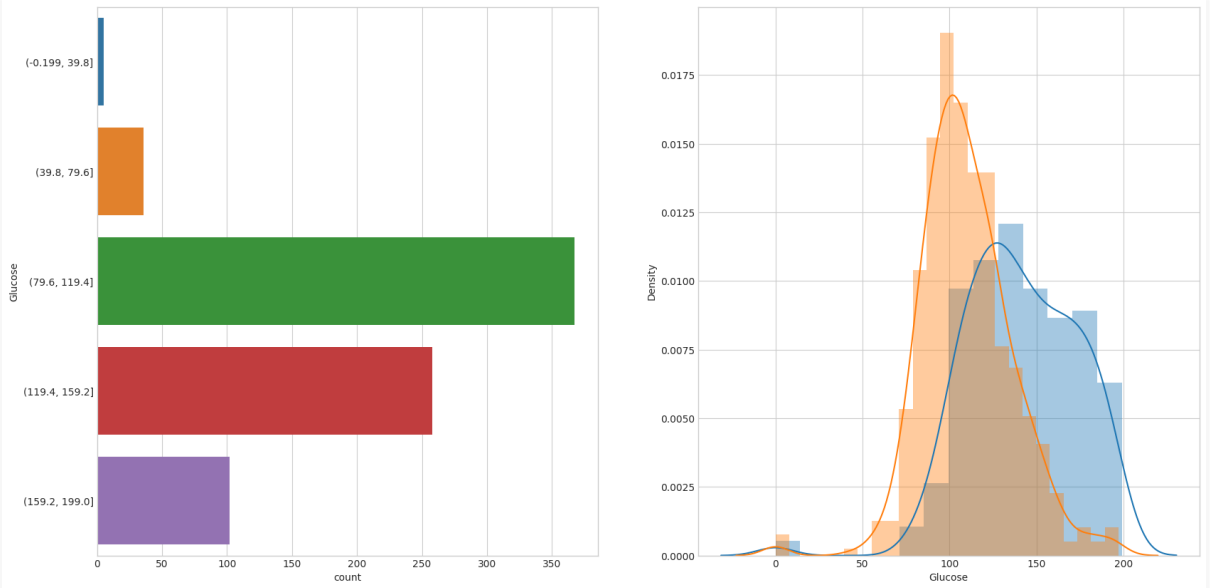
```
def systematic_sample(df, size):
    length = len(df)
    interval = length // size
    rd.seed(None)
    first = rd.randint(0, interval)
    indexes = np.arange(first, length, step = interval)
    return df.iloc[indexes]
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
def fill_na(df):
    for col in df.columns:
        if df[col].isnull().any():
            if df[col].dtypes in ["float", "int"]:
                df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].mode()[0], inplace=True)
def plot_count_dist(data, bin_df, label_column, target_column, figsize=(20, 5)
, use_bin_df=False):
    """
    Function to plot counts and distributions of a label variable and
    target variable side by side.
    ::param_data:: = target dataframe
    ::param_bin_df:: = binned dataframe for countplot
    ::param_label_column:: = binary labelled column
```

```
    ::param_target_column:: = column you want to view counts and distributions
    ::param_figsize:: = size of figure (width, height)
    ::param_use_bin_df:: = whether or not to use the bin_df, default False
    """
    if use_bin_df:
        fig = plt.figure(figsize=figsize)
        plt.subplot(1, 2, 1)
        sns.countplot(y=target_column, data=bin_df);
        plt.subplot(1, 2, 2)
        sns.distplot(data.loc[data[label_column] == 1][target_column],
                     kde_kws={"label": "Survived"});
        sns.distplot(data.loc[data[label_column] == 0][target_column],
                     kde_kws={"label": "Did not survive"});
    else:
        fig = plt.figure(figsize=figsize)
        plt.subplot(1, 2, 1)
        sns.countplot(y=target_column, data=data);
        plt.subplot(1, 2, 2)
        sns.distplot(data.loc[data[label_column] == 1][target_column],
                     kde_kws={"label": "Survived"});
        sns.distplot(data.loc[data[label_column] == 0][target_column],
                     kde_kws={"label": "Did not survive"});
# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, cv):

    # One Pass
    model = algo.fit(X_train, y_train)
    acc = round(model.score(X_train, y_train) * 100, 2)

    # Cross Validation
    train_pred = model_selection.cross_val_predict(algo,
                                           X_train,
                                           y_train,
                                           cv=cv,
                                           n_jobs = -1)
    # Cross-validation accuracy metric
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)

    return train_pred, acc, acc_cv
```

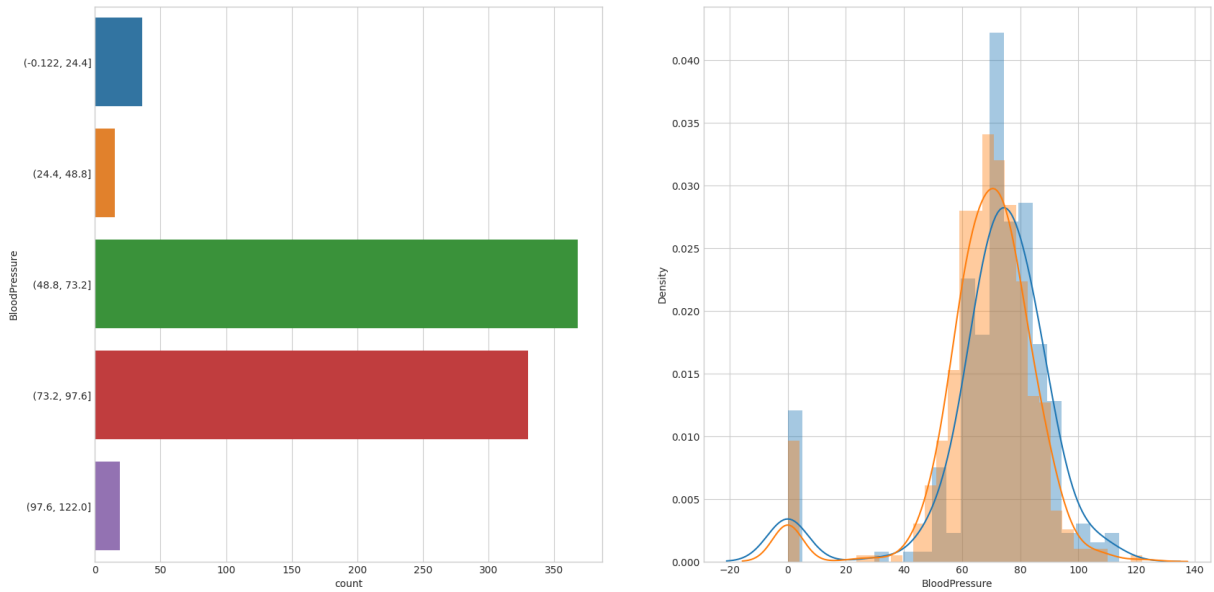
linkcode

```
# Feature Importance
def feature_importance(model, data):
    """
    Function to show which features are most important in the model.
    ::param_model:: Which model to use?
    ::param_data:: What data to use?
    """
    fea_imp = pd.DataFrame({'imp': model.feature_importances_, 'col': data.columns})
    fea_imp = fea_imp.sort_values(['imp', 'col'], ascending=[True, False]).iloc[-30:]
    _ = fea_imp.plot(kind='barh', x='col', y='imp', figsize=(20, 10))
```

```
    return fea_imp
    #plt.savefig('catboost_feature_importance.png')
```

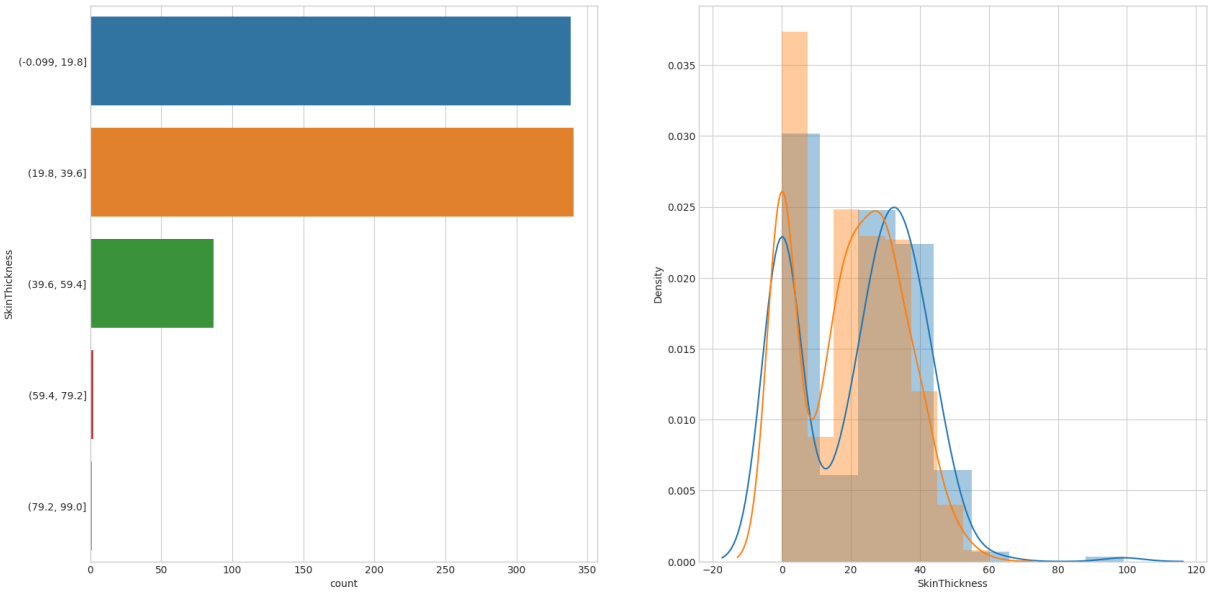



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
                bin_df=df_dis,
                label_column='Outcome',
                target_column='BloodPressure',
                figsize=(20,10),
                use_bin_df=True)
```

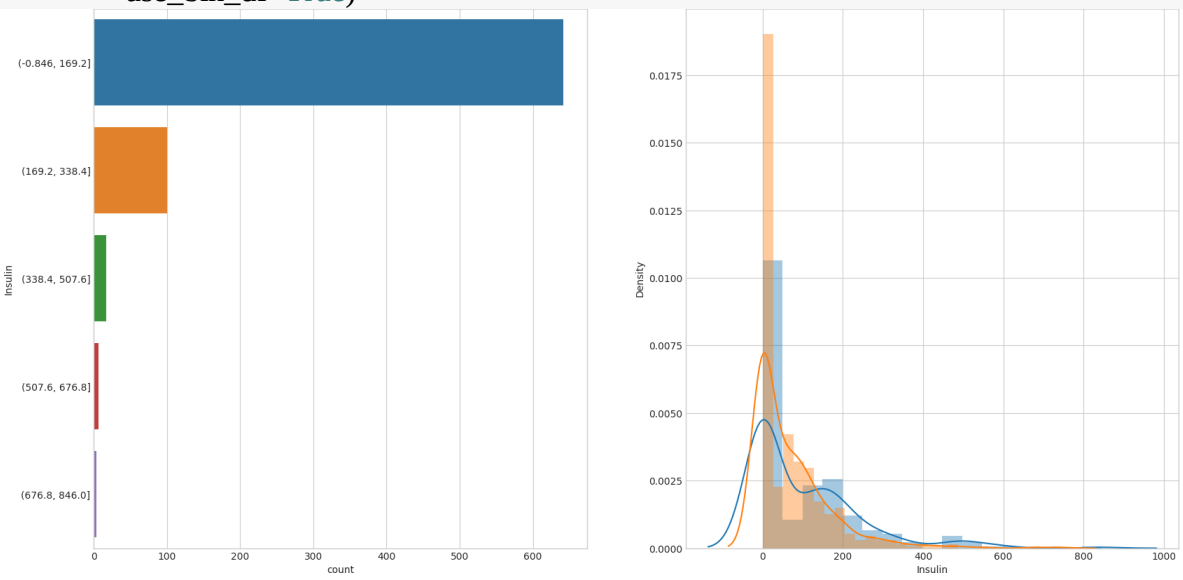



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
```

```
        bin_df=df_dis,
        label_column='Outcome',
        target_column='SkinThickness',

                figsize=(20,10),
                use_bin_df=True)
```

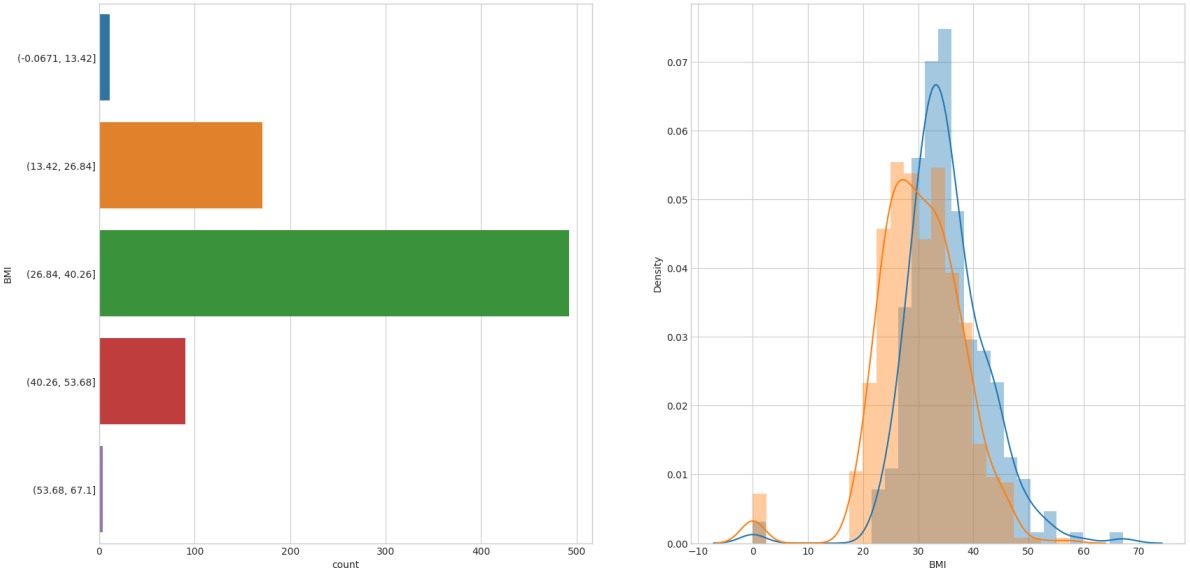



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
        bin_df=df_dis,
        label_column='Outcome',
        target_column='Insulin',
        figsize=(20,10),
        use_bin_df=True)
```

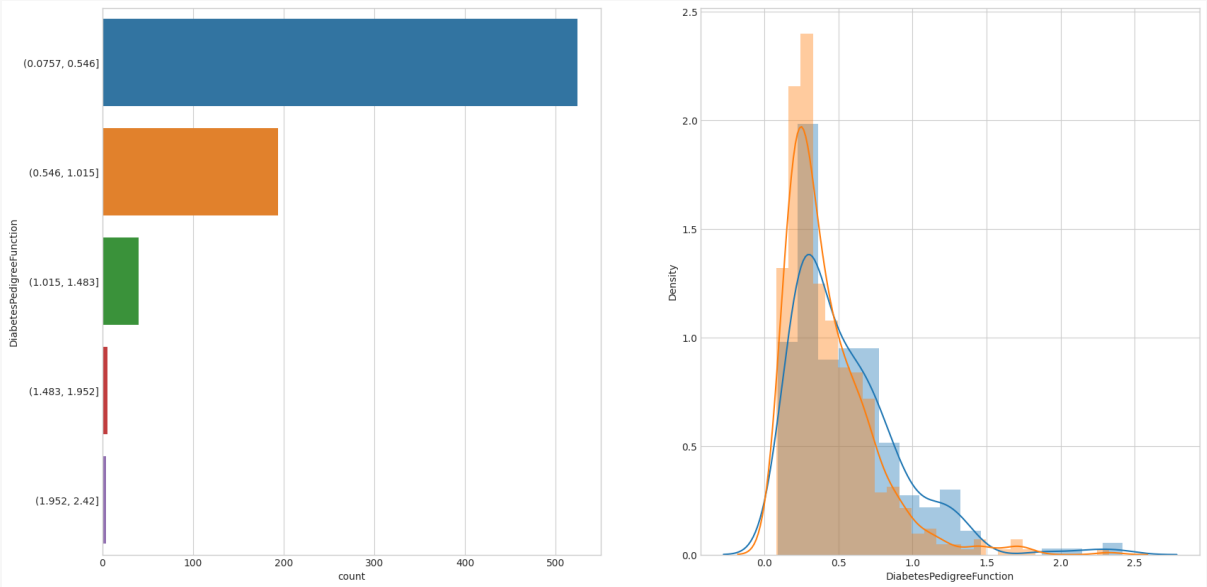



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
        bin_df=df_dis,
        label_column='Outcome',
```

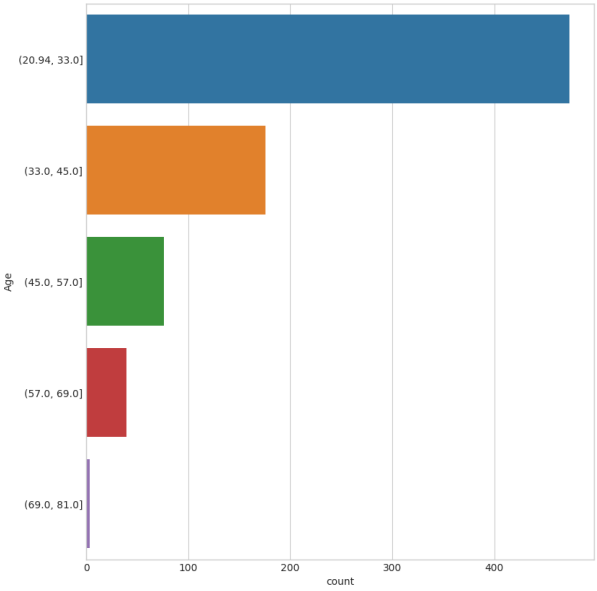
```
            target_column='BMI',
            figsize=(20,10),
            use_bin_df=True)
```

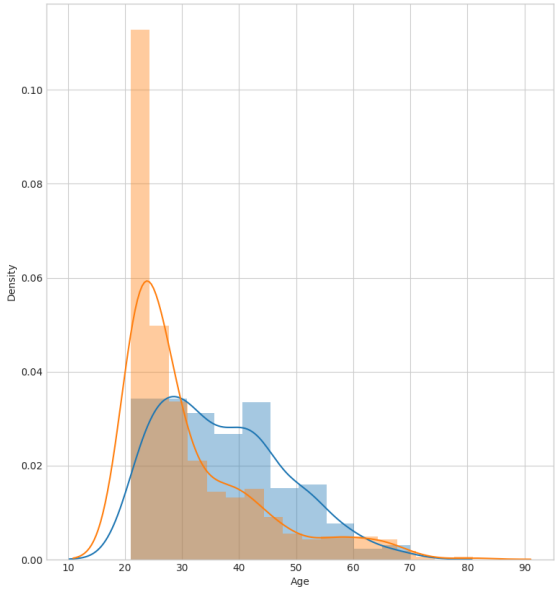



```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
            bin_df=df_dis,
            label_column='Outcome',
            target_column='DiabetesPedigreeFunction',
            figsize=(20,10),
            use_bin_df=True)
```




```
# Visualise the Fare bin counts as well as the Fare distribution versus Survived.
plot_count_dist(data=df,
            bin_df=df_dis,
            label_column='Outcome',
            target_column='Age',
            figsize=(20,10),
            use_bin_df=True)
```





431	0	False	False	False	True	False	False	False	False	False	...	False	True	False	False	False	False	True	False	False	False
584	1	False	False	False	False	False	False	False	False	True	...	False	True	False	False	False	False	False	True	False	False
737	0	False	False	False	False	False	False	False	False	True	...	False	True	False	False	False	False	True	False	False	False

```
df_con_enc = pd.get_dummies(df_con, columns=one_hot_cols)
df_con_enc.head()
```

MACHINE LEARNING:

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, and to uncover key insights in data mining projects.These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase. They will be required to help identify the most relevant business questions and the data to answer them.Machine learning algorithms are typically

created using frameworks that accelerate solution development, such as TensorFlow and PyTorch.

1. **A Decision Process**: In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data.

2. **An Error Function**: An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.

3. **A Model Optimization Process**: If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met

COMMON MACHINE LEARNING ALGORITHMS:

- **Neural networks:** Neural networks simulate the way the human brain works, with a huge number of linked processing nodes. Neural networks are good at recognizing patterns and play an important role in applications including natural language translation, image recognition, speech recognition, and image creation.

- **Linear regression:** This algorithm is used to predict numerical values, based on a linear relationship between different values. For example, the technique could be used to predict house prices based on historical data for the area.

- **Logistic regression:** This supervised learning algorithm makes predictions for categorical response variables, such as"yes/no" answers to questions. It can be used for applications such as classifying spam and quality control on a production line.

- **Clustering:** Using unsupervised learning, clustering algorithms can identify patterns in data so that it can be grouped. Computers can help data scientists by identifying differences between data items that humans have overlooked.

- **Decision trees:** Decision trees can be used for both predicting numerical values (regression) and classifying data into categories. Decision trees use a branching sequence of linked decisions that can be represented with a tree diagram. One of the advantages of decision trees is that they are easy to validate and audit, unlike the black box of the neural network.

- **Random forests:** In a random forest, the machine learning algorithm predicts a value or category by combining the results from a number of decision trees.

```
# Seclect the dataframe we want to use first for predictions
selected_df = df_con_enc
selected_df.head()
```

```
# Split the dataframe into data and labels
X_train = selected_df.drop('Outcome', axis=1) # data
y_train = selected_df.Outcome # labels

# Shape of the data (without labels)
X_train.shape


(768, 1254)
linkcode
X_train.head()

# Shape of the labels
y_train.shape

(768,)
# Logistic Regression
start_time = time.time()
train_pred_log, acc_log, acc_cv_log = fit_ml_algo(LogisticRegression(),
                                                  X_train,
                                                  y_train,
                                                      10)
log_time = (time.time() - start_time)
print("Accuracy: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))

Accuracy: 96.09
Accuracy CV 10-Fold: 67.45
Running Time: 0:00:02.451250
# k-Nearest Neighbours
start_time = time.time()
train_pred_knn, acc_knn, acc_cv_knn = fit_ml_algo(KNeighborsClassifier(),
                                          X_train,
                                          y_train,
                                          10)
knn_time = (time.time() - start_time)
print("Accuracy: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time: %s" % datetime.timedelta(seconds=knn_time))

Accuracy: 75.26
Accuracy CV 10-Fold: 66.8
Running Time: 0:00:00.358875


# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, acc_gaussian, acc_cv_gaussian = fit_ml_algo(GaussianNB(),
                                                     X_train,
                                                     y_train,
                                                        10)
gaussian_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
```

```
print("Running Time: %s" % datetime.timedelta(seconds=gaussian_time))
```

```
Accuracy: 94.79
Accuracy CV 10-Fold: 59.9
Running Time: 0:00:00.219412
```

```
# Linear SVC
start_time = time.time()
train_pred_svc, acc_linear_svc, acc_cv_linear_svc = fit_ml_algo(LinearSVC(),
                                                    X_train,
                                                    y_train,
                                                    10)
linear_svc_time = (time.time() - start_time)
print("Accuracy: %s" % acc_linear_svc)
print("Accuracy CV 10-Fold: %s" % acc_cv_linear_svc)
print("Running Time: %s" % datetime.timedelta(seconds=linear_svc_time))
```

```
Accuracy: 100.0
Accuracy CV 10-Fold: 65.23
Running Time: 0:00:00.268296
```

```
# Stochastic Gradient Descent
start_time = time.time()
train_pred_sgd, acc_sgd, acc_cv_sgd = fit_ml_algo(SGDClassifier(),
                                      X_train,
                                      y_train,
                                      10)
sgd_time = (time.time() - start_time)
print("Accuracy: %s" % acc_sgd)
print("Accuracy CV 10-Fold: %s" % acc_cv_sgd)
print("Running Time: %s" % datetime.timedelta(seconds=sgd_time))
```

```
Accuracy: 100.0
Accuracy CV 10-Fold: 63.93
Running Time: 0:00:00.437200
```

```
# Decision Tree Classifier
start_time = time.time()
train_pred_dt, acc_dt, acc_cv_dt = fit_ml_algo(DecisionTreeClassifier(),
                                                    X_train,
                                                    y_train,
                                                   )
dt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))
```

```
Accuracy: 100.0
Accuracy CV 10-Fold: 61.85
Running Time: 0:00:00.596504
```

```
# Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, acc_gbt, acc_cv_gbt = fit_ml_algo(GradientBoostingClassifier(),
                                                    X_train,
                                                    y_train,
                                                    10)
```

```
gbt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gbt)
print("Accuracy CV 10-Fold: %s" % acc_cv_gbt)
print("Running Time: %s" % datetime.timedelta(seconds=gbt_time))
```

Accuracy: 81.9
Accuracy CV 10-Fold: 63.54
Running Time: 0:00:06.448709

CATBOOST ALGORITHM:

CatBoost is a supervised machine learning method that is used by the Train Using AutoML tool and uses decision trees for classification and regression. As its name suggests, CatBoost has two main features, it works with categorical data (the Cat) and it uses gradient boosting (the Boost). Gradient boosting is a process in which many decision trees are constructed iteratively. Each subsequent tree improves the result of the previous tree, leading to better results. CatBoost improves on the original gradient boost method for a faster implementation.

CatBoost overcomes a limitation of other decision tree-based methods in which, typically, the data must be pre-processed to convert categorical string variables to numerical values, one-hot-encodings, and so on. This method can directly consume a combination of categorical and non-categorical explanatory variables without preprocessing. It preprocesses as part of the algorithm. CatBoost uses a method called ordered encoding to encode categorical features. Ordered encoding considers the target statistics from all the rows prior to a data point to calculate a value to replace the categorical feature.Another unique characteristic of CatBoost is that it uses symmetric trees. This means that at every depth level, all the decision nodes use the same split condition.

CatBoost can also be faster than other methods such as XGBoost. It retains certain features—such as cross-validation, regularization, and missing value support—from the prior algorithms. This method performs well with both small data and large data.

```
systematic_sample(X_train, 5)
```

Pregnancies_0	Pregnancies_1	Pregnancies_2	Pregnancies_3	Pregnancies_4	Pregnancies_5	Pregnancies_6	Pregnancies_7	Pregnancies_8	Pregnancies_9	...	Age_63	Age_64	Age_65	Age_66	Age_67	Age_68	Age_69	Age_70	Age_72	Age_81	
42	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False	False

Pregnancies_0	Pregnancies_1	Pregnancies_2	Pregnancies_3	Pregnancies_4	Pregnancies_5	Pregnancies_6	Pregnancies_7	Pregnancies_8	Pregnancies_9	...	Age_63	Age_64	Age_65	Age_66	Age_67	Age_68	Age_69	Age_70	Age_72	Age_81	
195	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
348	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
501	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
654	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

```
systematic_sample(y_train, 5)
```

```
302    0
455    1
608    0
761    1
Name: Outcome, dtype: int64
```

```
# Define the categorical features for the CatBoost model
cat_features = np.where(X_train.dtypes != np.float)[0]
cat_features
```

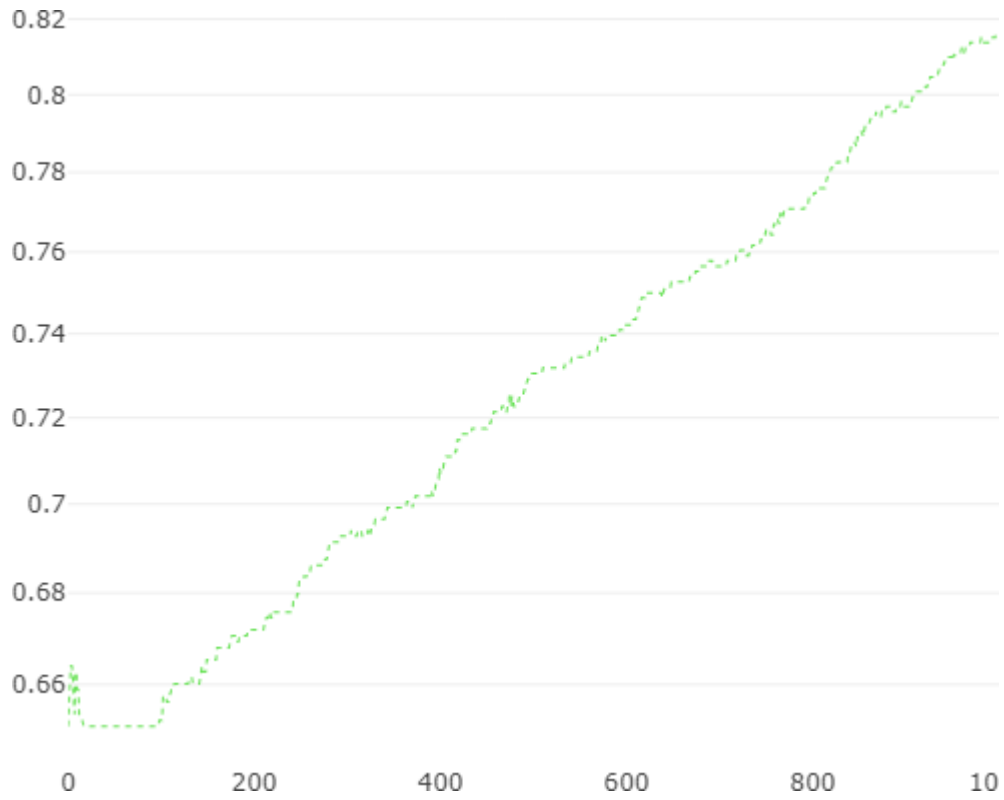
```
array([   0,    1,    2, ..., 1251, 1252, 1253])
```

```
# Use the CatBoost Pool() function to pool together the training data and categori
cal feature labels
train_pool = Pool(X_train,
                  y_train,
                  cat_features)
```

linkcode

```
# CatBoost model definition
```

```
catboost_model = CatBoostClassifier(iterations=1000,
                                    custom_loss=['Accuracy'],
                                    loss_function='Logloss')

# Fit CatBoost model
catboost_model.fit(train_pool,
                   plot=True)

# CatBoost accuracy
acc_catboost = round(catboost_model.score(X_train, y_train) * 100, 2)
```




```
469:    learn: 0.5395230        total: 2.34s    remaining: 2.64s
470:    learn: 0.5394029        total: 2.34s    remaining: 2.63s
471:    learn: 0.5392298        total: 2.35s    remaining: 2.63s
472:    learn: 0.5390315        total: 2.36s    remaining: 2.63s
473:    learn: 0.5388890        total: 2.37s    remaining: 2.63s
474:    learn: 0.5387269        total: 2.37s    remaining: 2.62s
475:    learn: 0.5384723        total: 2.38s    remaining: 2.62s
476:    learn: 0.5383353        total: 2.38s    remaining: 2.62s
477:    learn: 0.5382355        total: 2.39s    remaining: 2.61s
478:    learn: 0.5380677        total: 2.4s     remaining: 2.61s
479:    learn: 0.5378452        total: 2.4s     remaining: 2.61s
480:    learn: 0.5376842        total: 2.41s    remaining: 2.6s
481:    learn: 0.5375600        total: 2.42s    remaining: 2.6s
482:    learn: 0.5374369        total: 2.42s    remaining: 2.6s
483:    learn: 0.5372150        total: 2.43s    remaining: 2.59s
484:    learn: 0.5371141        total: 2.44s    remaining: 2.59s
485:    learn: 0.5369491        total: 2.45s    remaining: 2.59s
```

```
486:    learn: 0.5368111        total: 2.46s    remaining: 2.59s
487:    learn: 0.5367199        total: 2.46s    remaining: 2.59s
488:    learn: 0.5365787        total: 2.47s    remaining: 2.58s
489:    learn: 0.5364203        total: 2.48s    remaining: 2.58s
490:    learn: 0.5362019        total: 2.48s    remaining: 2.57s
491:    learn: 0.5361098        total: 2.49s    remaining: 2.57s
492:    learn: 0.5358471        total: 2.49s    remaining: 2.56s
493:    learn: 0.5357295        total: 2.5s     remaining: 2.56s
494:    learn: 0.5355541        total: 2.5s     remaining: 2.55s
495:    learn: 0.5353691        total: 2.51s    remaining: 2.55s
496:    learn: 0.5352261        total: 2.51s    remaining: 2.54s
497:    learn: 0.5351270        total: 2.52s    remaining: 2.54s
498:    learn: 0.5350399        total: 2.52s    remaining: 2.53s
499:    learn: 0.5349238        total: 2.53s    remaining: 2.53s
500:    learn: 0.5348211        total: 2.53s    remaining: 2.52s
501:    learn: 0.5347660        total: 2.54s    remaining: 2.52s
502:    learn: 0.5346753        total: 2.54s    remaining: 2.51s
503:    learn: 0.5344722        total: 2.55s    remaining: 2.51s
504:    learn: 0.5341997        total: 2.56s    remaining: 2.5s
505:    learn: 0.5340434        total: 2.56s    remaining: 2.5s
```

```
print("---CatBoost Metrics---")
print("Accuracy: {}".format(acc_catboost))
print("Accuracy cross-validation 10-Fold: {}".format(acc_cv_catboost))
print("Running Time: {}".format(datetime.timedelta(seconds=catboost_time)))
```

```
---CatBoost Metrics---
Accuracy: 81.64
Accuracy cross-validation 10-Fold: 66.54
Running Time: 0:00:59.138368
```

linkcode

```
models = pd.DataFrame({
   'Model': ['KNN', 'Logistic Regression', 'Naive Bayes',
            'Stochastic Gradient Decent', 'Linear SVC',
            'Decision Tree', 'Gradient Boosting Trees',
            'CatBoost'],
   'Score':  [
      acc_knn,
      acc_log,
      acc_gaussian,
      acc_sgd,
      acc_linear_svc,
      acc_dt,
      acc_gbt,

      acc_catboost
   ]})
print("---Reuglar Accuracy Scores---")
models.sort_values(by='Score', ascending=False)
```

```
---Reuglar Accuracy Scores---
```

Model	Score	
3	Stochastic Gradient Decent	100.00
4	Linear SVC	100.00
5	Decision Tree	100.00
1	Logistic Regression	96.09
2	Naive Bayes	94.79
6	Gradient Boosting Trees	81.90
7	CatBoost	81.64
0	KNN	75.26

```
cv_models = pd.DataFrame({
   'Model': ['KNN', 'Logistic Regression', 'Naive Bayes',
            'Stochastic Gradient Decent', 'Linear SVC',
            'Decision Tree', 'Gradient Boosting Trees',
            'CatBoost'],
   'Score':  [
      acc_cv_knn,
      acc_cv_log,
      acc_cv_gaussian,
      acc_cv_sgd,
      acc_cv_linear_svc,
      acc_cv_dt,
      acc_cv_gbt,
      acc_cv_catboost
   ]})
print('---Cross-validation Accuracy Scores---')
```

```
cv_models.sort_values(by='Score', ascending=False)
```

---Cross-validation Accuracy Scores---

	Model	Score
1	Logistic Regression	67.45
0	KNN	66.80
7	CatBoost	66.54
4	Linear SVC	65.23
3	Stochastic Gradient Decent	63.93
6	Gradient Boosting Trees	63.54
5	Decision Tree	61.85
2	Naive Bayes	59.90

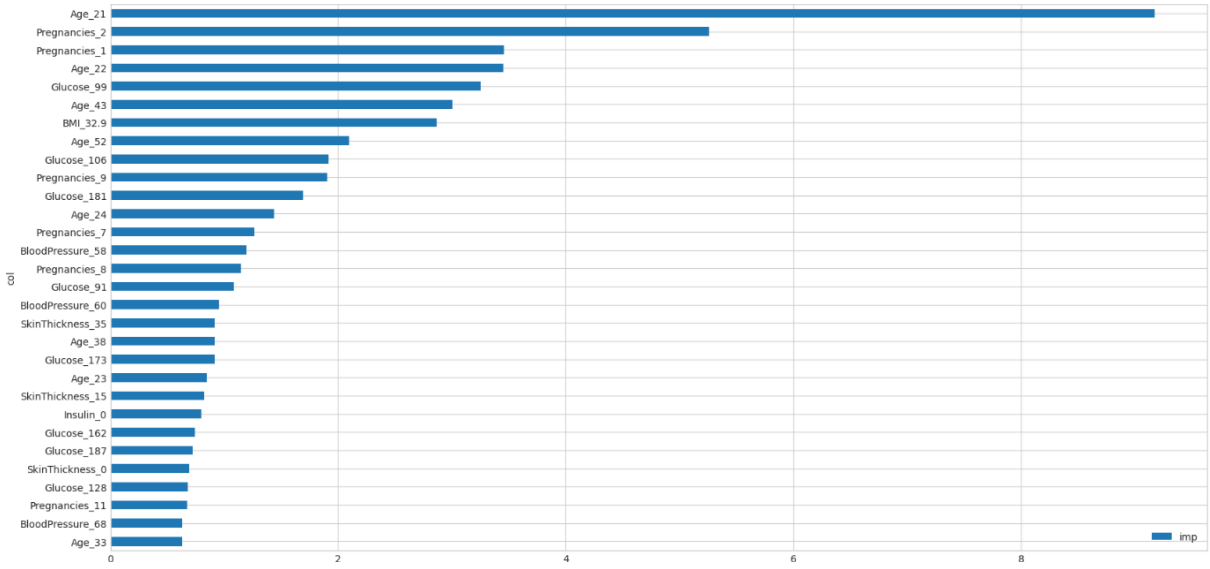
```
# Plot the feature importance scores
feature_importance(catboost_model, X_train)
```

	imp	col
1214	0.628039	Age_33
173	0.628329	BloodPressure_68

imp	col	
11	0.674076	Pregnancies_11
83	0.677186	Glucose_128
200	0.690573	SkinThickness_0
141	0.722829	Glucose_187
117	0.741188	Glucose_162
251	0.797263	Insulin_0
208	0.821834	SkinThickness_15
1204	0.847008	Age_23
128	0.914077	Glucose_173
1219	0.917231	Age_38
228	0.917400	SkinThickness_35
167	0.954864	BloodPressure_60
46	1.084594	Glucose_91

imp	col	
8	1.143536	Pregnancies_8
166	1.197212	BloodPressure_58
7	1.261164	Pregnancies_7
1205	1.437076	Age_24
136	1.694708	Glucose_181
9	1.900946	Pregnancies_9
61	1.916399	Glucose_106
1233	2.096556	Age_52
555	2.866333	BMI_32.9
1224	3.000875	Age_43
54	3.251553	Glucose_99
1203	3.452897	Age_22
1	3.460366	Pregnancies_1

imp	col	
2	5.262568	Pregnancies_2
1202	9.173822	Age_21



FUTURE WORK:

In the future,AI-based diabetes prediction systems are likely to become more advanced and accurate. They may incorporate deep learning algorithms, analyzing vast datasets to identify subtle patterns and early indicators of diabetes. These systems could provide real-time monitoring, personalized recommendations for lifestyle changes, and predictive alters to both individuals and health care providers. Integration with wearable devices and continuous glucose monitoring tools might offer seamless data collection, enabling precise predictions and timely interventions. Additionally ethical considerations, data privacy and user friendly interfaces will play pivotal roles in shaping the future landscapeof AI-based diabetes prediction system.

CONCLUSION:

We have gone through many more different kind of model and we analyse that the top most efficient algorithm among them till now is the random forest one, that is producing best output and result from all the tried model.