औद्योगिक प्रशिक्षण के लिए राष्ट्रीय संस्थान

National Institute for Industrial Training

One Premier Organization with Non Profit Status | Registered Under Govt. of WB
Empanelled Under Planning Commission Govt. of India
Inspired By: National Task Force on IT & SD Government of India

National Institute for Industrial Training- One Premier Organization with Non Profit Status Registered Under Govt.of West Bengal,Empanelled Under Planning Commission Govt.of India , Empanelled Under Central Social Welfare Board Govt. of India , Registered with National Career Services , Registered with National Employment Services.

Data Science with Python

Subject: Diabetes Prediction

Submitted By: Subham Kundu

Submitted To: Soumotanu Mazumdar Sir

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to all those who provided me the possibility to complete this project. I give special gratitude to my project guider, Soumotanu Mazumdar Sir, whose contribution in stimulating ideas and encouragement helped me coordinate my project. Furthermore, I would also like to acknowledge with much appreciation the crucial role of the National Institute for Industrial Training, which gave the permission to use all required equipment and the necessary materials to complete the course “Data Science with Python”. I have put tremendous effort in this project. However, it would not have been possible without the kind support and help of above-mentioned individual and organization. I would like to extend my sincere thanks to all of them.

Name: Subham Kundu

College: Techno India University

Stream: B. Tech CSE

Email: subhamkundu486@gmail.com

LinkedIn: https://www.linkedin.com/in/subham-kundu-10654994/

GitHub Profile: https://github.com/subhamrex

Project: Diabetes Prediction

Project Files: https://github.com/subhamrex/NIIT-DataSc-Project

INTRODUCTION

Python comes with a huge number of inbuilt libraries. Many of the libraries are for Data Science and Machine Learning. Some of the libraries are **scikit-learn** (for data mining, data analysis and machine learning), **NumPy** (for adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays)**, pandas** (for data manipulation and analysis), **seaborn** (data visualization library based on matplotlib) etc. The list keeps going and never ends.

What makes Python favourite for everyone is its powerful and easy implementation. For other languages, students and researchers need to get to know the language before getting into Data Science with that language. This is not the case with python. Even a programmer with very basic knowledge can easily handle python. Apart from that, the time someone spends on writing and debugging code in python is way less when compared to C, C++ or Java. This is exactly what the students of Data Science want. They don't want to spend time on debugging the code for syntax errors, they want to spend more time on their algorithms and heuristics related to Data science. Not just the libraries but their tutorials, handling of interfaces are easily available online. People build their own libraries and upload them on GitHub or elsewhere to be used by others. Apart from this statistic is also required for this field. Now-a-days Packages have made this easier.

Hardware and software requirements

Software Requirements:

Operating System: Windows/Linux

Programming Language: Python 3.8

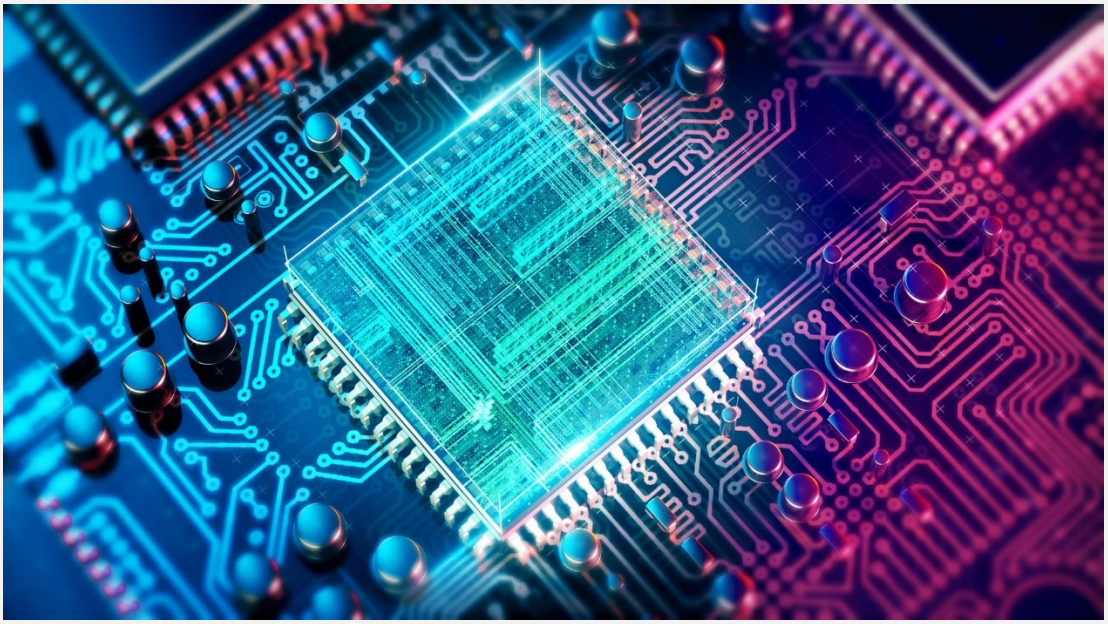
Software (IDE): PyCharm or VSCode

Hardware requirements:

Speed: 233MHz and above

Hard disk: 10GB

RAM: 512 MB

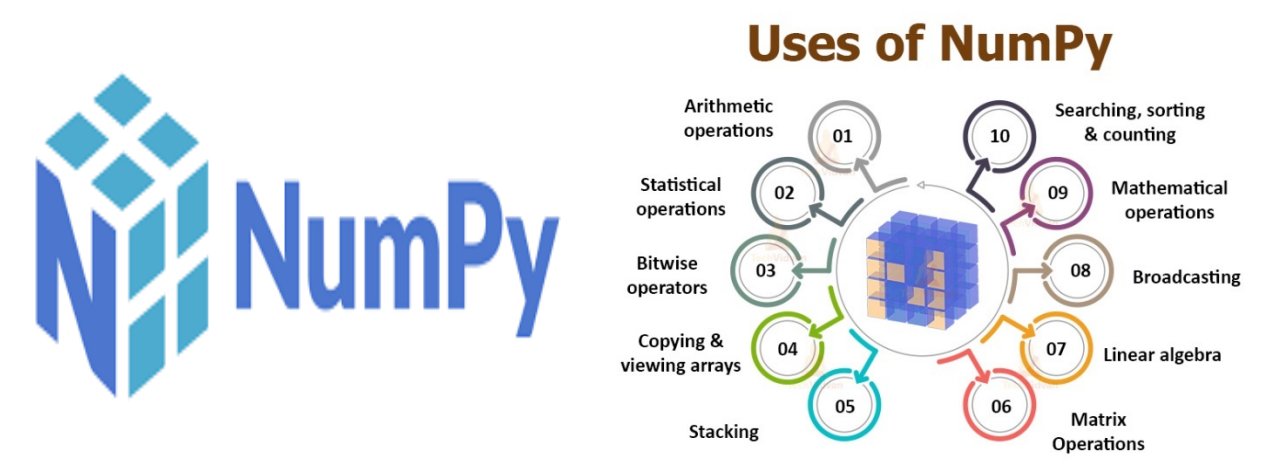


Objective & Procedure

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

Python is a powerful open-source programming language, which means that it's free to use while having all the properties that a programming language should have. Python has some 72,000 libraries in the Python Package Index that aid in scientific calculations and machine learning applications. In this project I have used Python as a programming language.

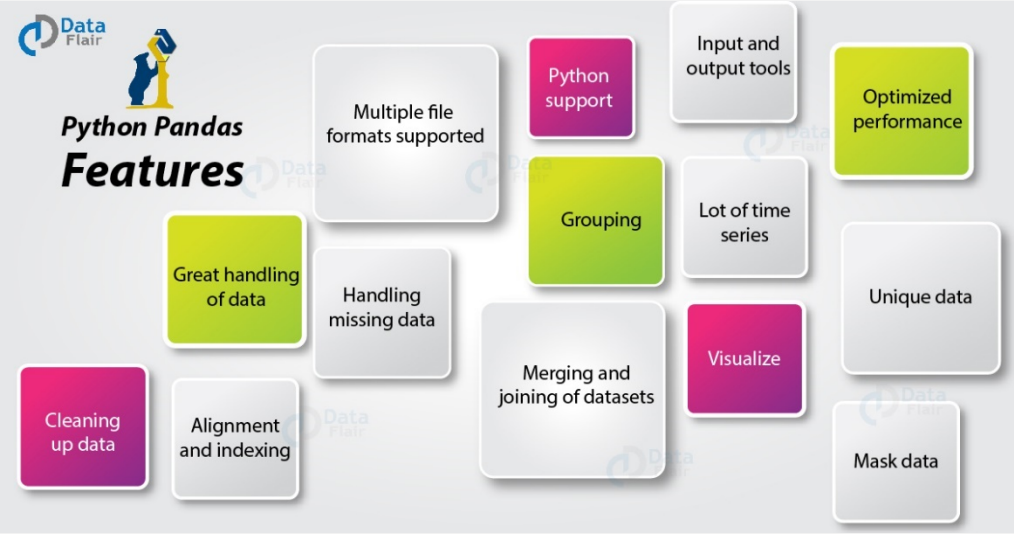
Firstly, I have used **NumPy** which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.



Here I have used it for perform mathematical and logical operations.

Secondly, I have used **pandas** which is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

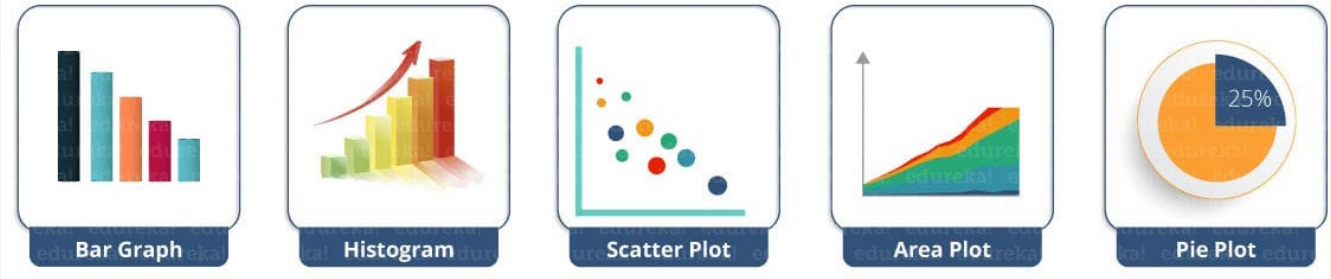
pandas



Here I have used it for reading the csv file and doing data manipulation.

Thirdly, I have used **matplotlib** which is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

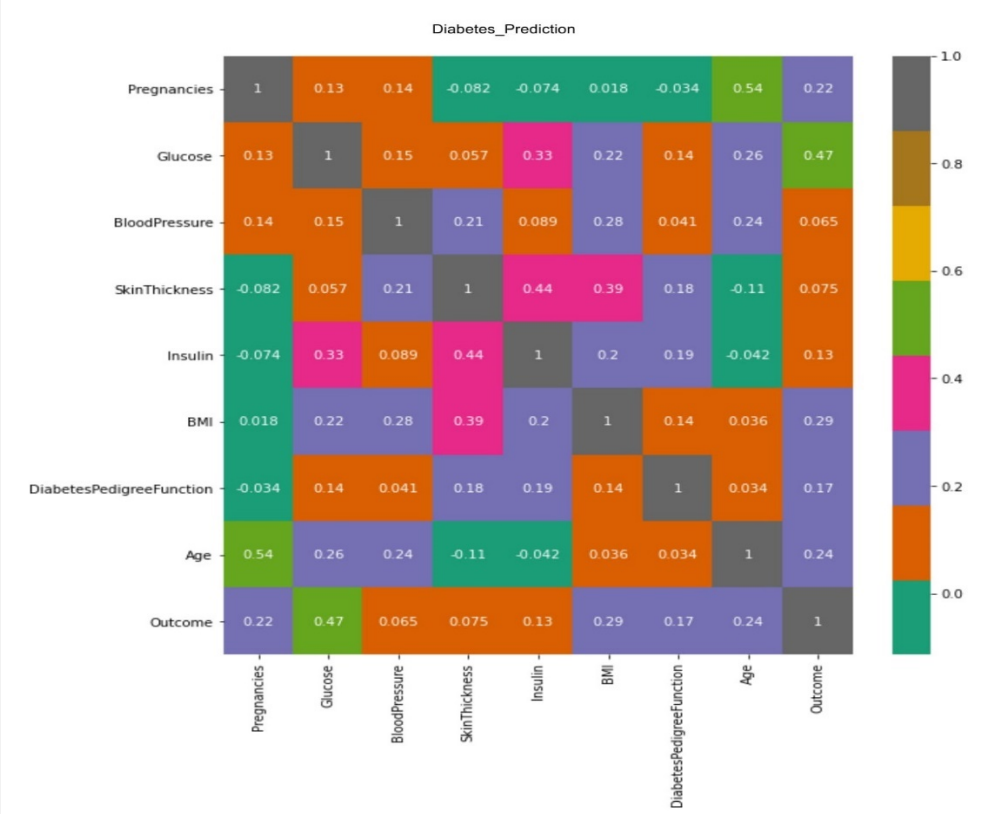
matplotlib



Here I have used it to plot my required data for visualization.

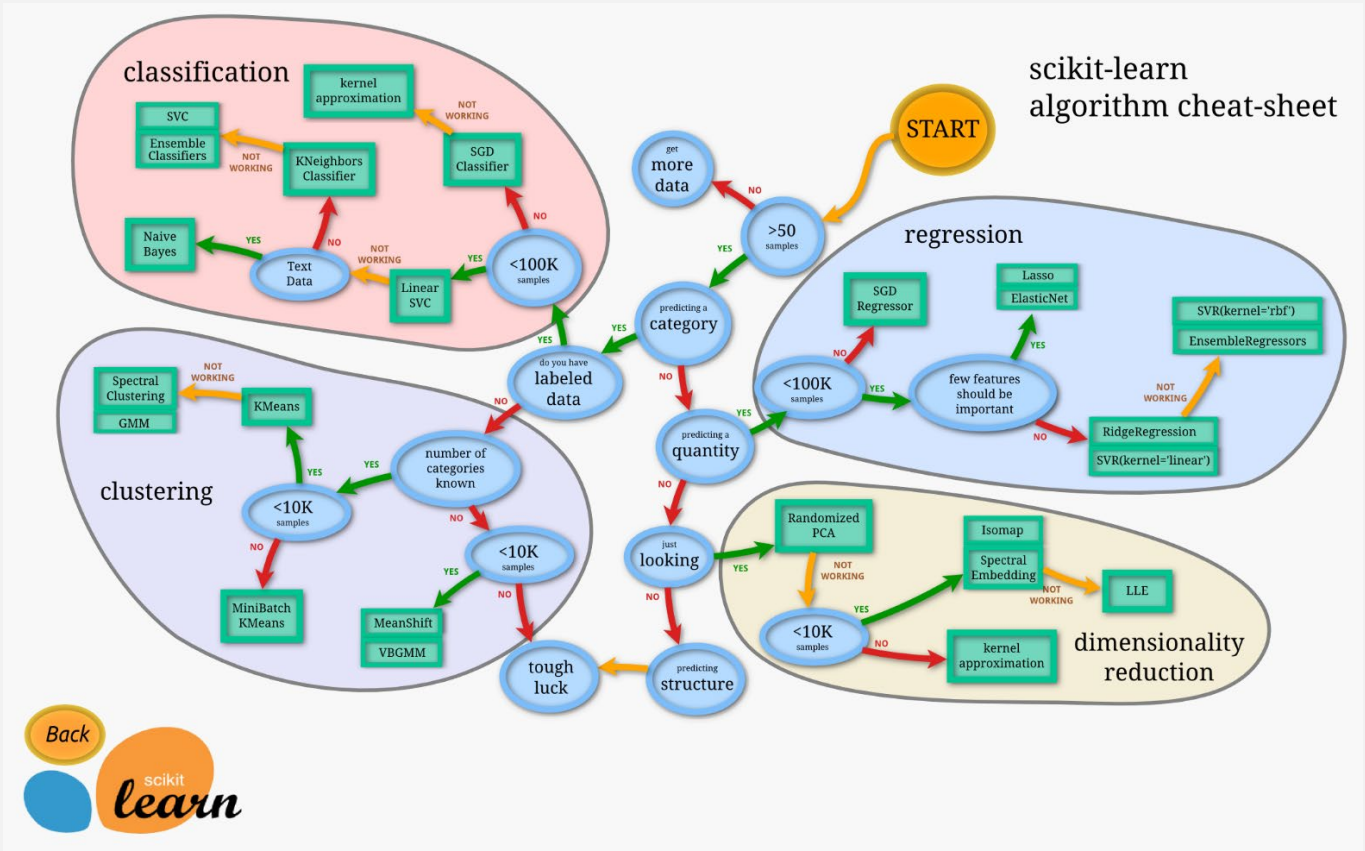
Fourthly, I have used **seaborn** which a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

seaborn



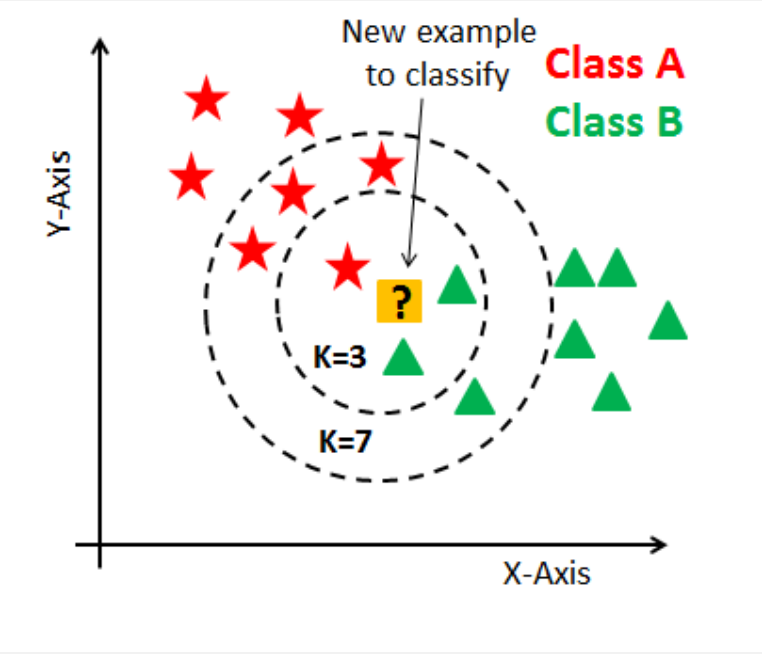
Here I have used it for better visualization of my data.

Lastly, I have used **scikit-learn** which is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.



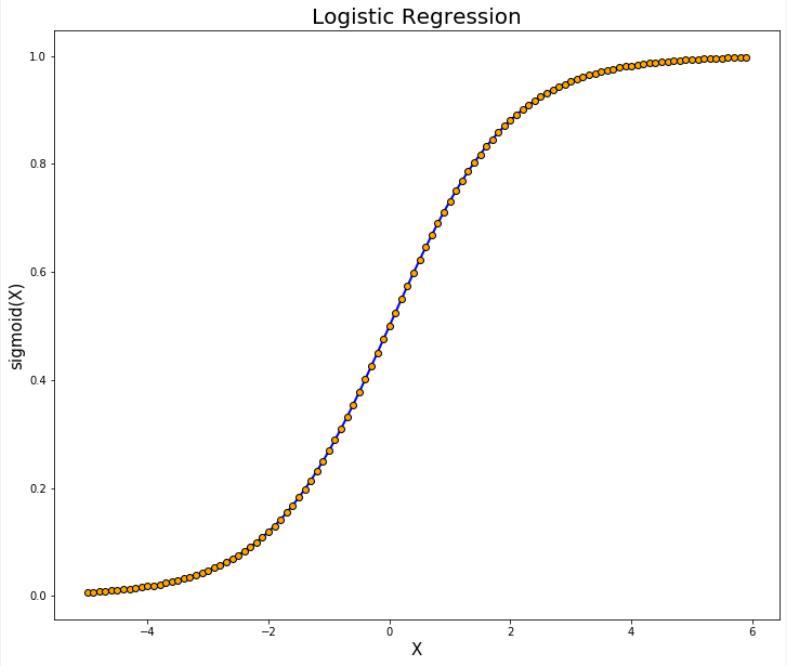
Here I have used it for KNN, Logistic regression and Random forest algorithm. This algorithm will give us the desire output with corresponding accuracy.

KNN: The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows**.**



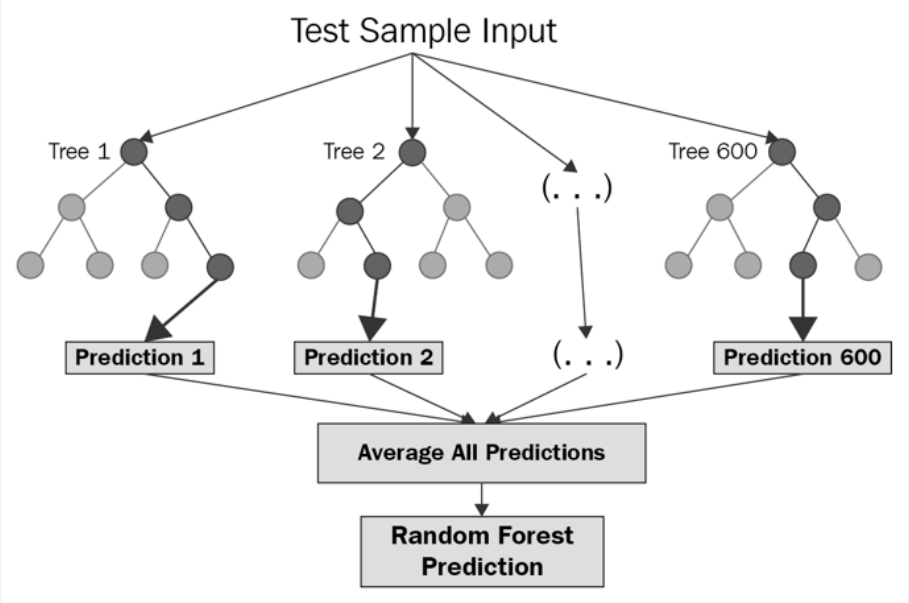
In this project, I have used KNN from sk-learn library.

Logistic regression: Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).



In this project, I have used Logistic regression from sk-learn library.

Random forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.



In this project, I have used Random forest from sk-learn library.

Code Snapshots:

Import libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.linear_model import LinearRegression
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing,metrics,svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from pprint import pprint
from sklearn.metrics import f1_score,log_loss,accuracy_score,recall_score,cohen_kappa_s
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import itertools
```

Read CSV file

In [3]:
```
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[3]:

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

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.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

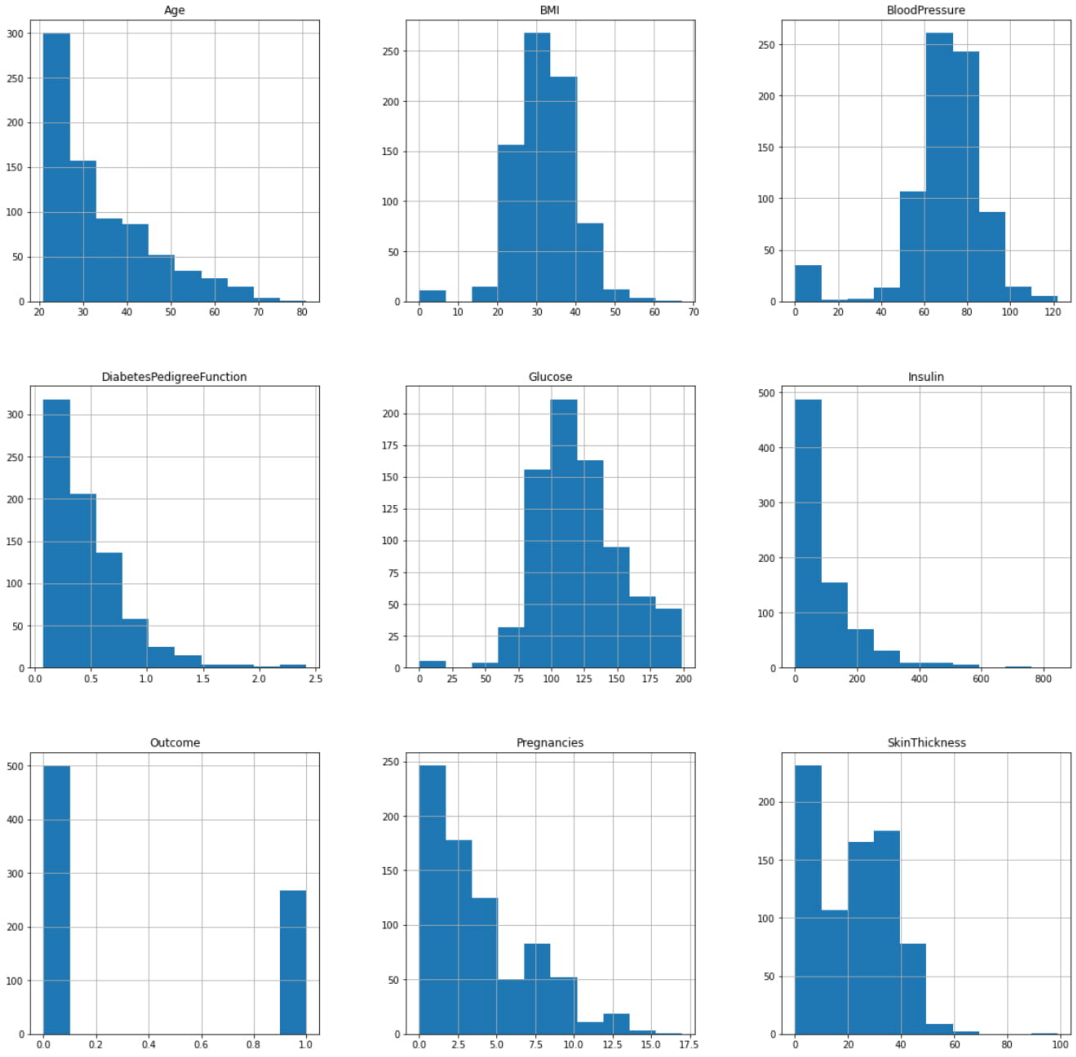
In [6]:
```
print(df.isnull().sum())
```

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

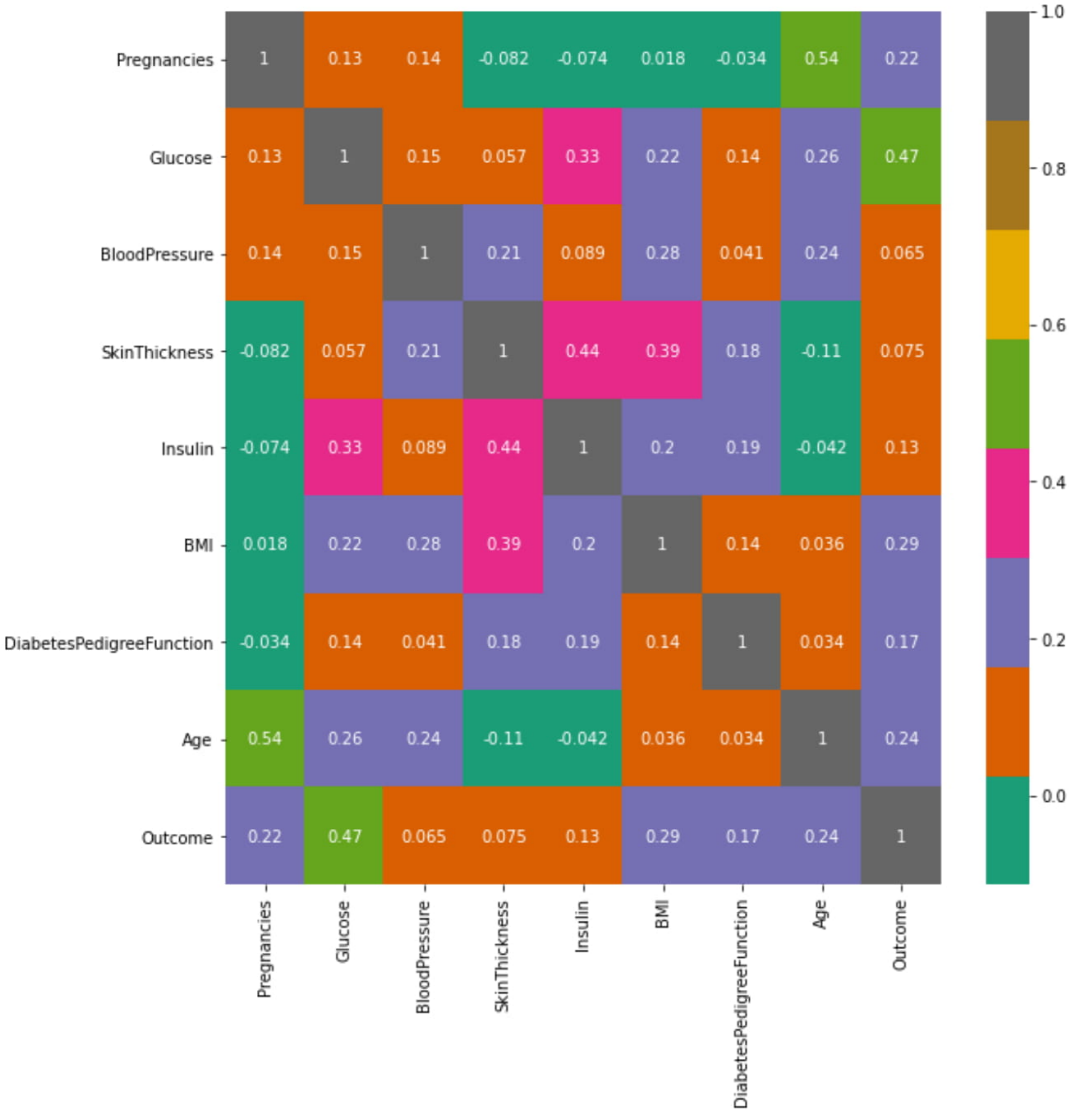
In [7]:
```
df.corr()['Outcome'].sort_values()
```

Out[7]:
```
BloodPressure               0.065068
SkinThickness               0.074752
Insulin                     0.130548
DiabetesPedigreeFunction    0.173844
Pregnancies                 0.221898
Age                         0.238356
BMI                         0.292695
Glucose                     0.466581
Outcome                     1.000000
Name: Outcome, dtype: float64
```

In [8]:
```
his = df.hist(figsize = (20,20))
```



In [9]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), cmap="Dark2", annot= True,)
plt.show()
```



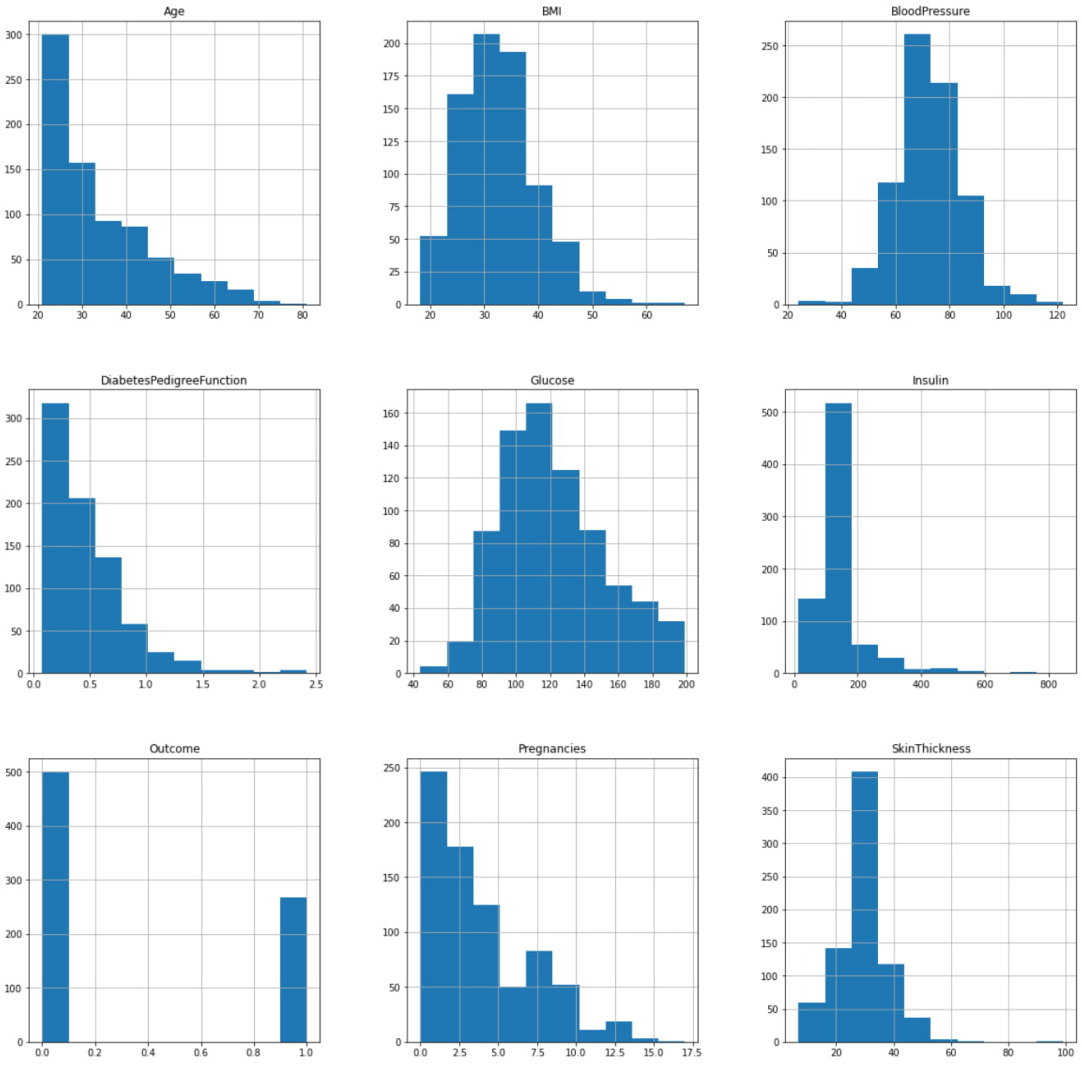
In [10]:
```
df1 = df.copy(deep = True)
df1[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df1[['Glucose','Bloo

## showing the count of Nans
print(df1.isnull().sum())
```

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

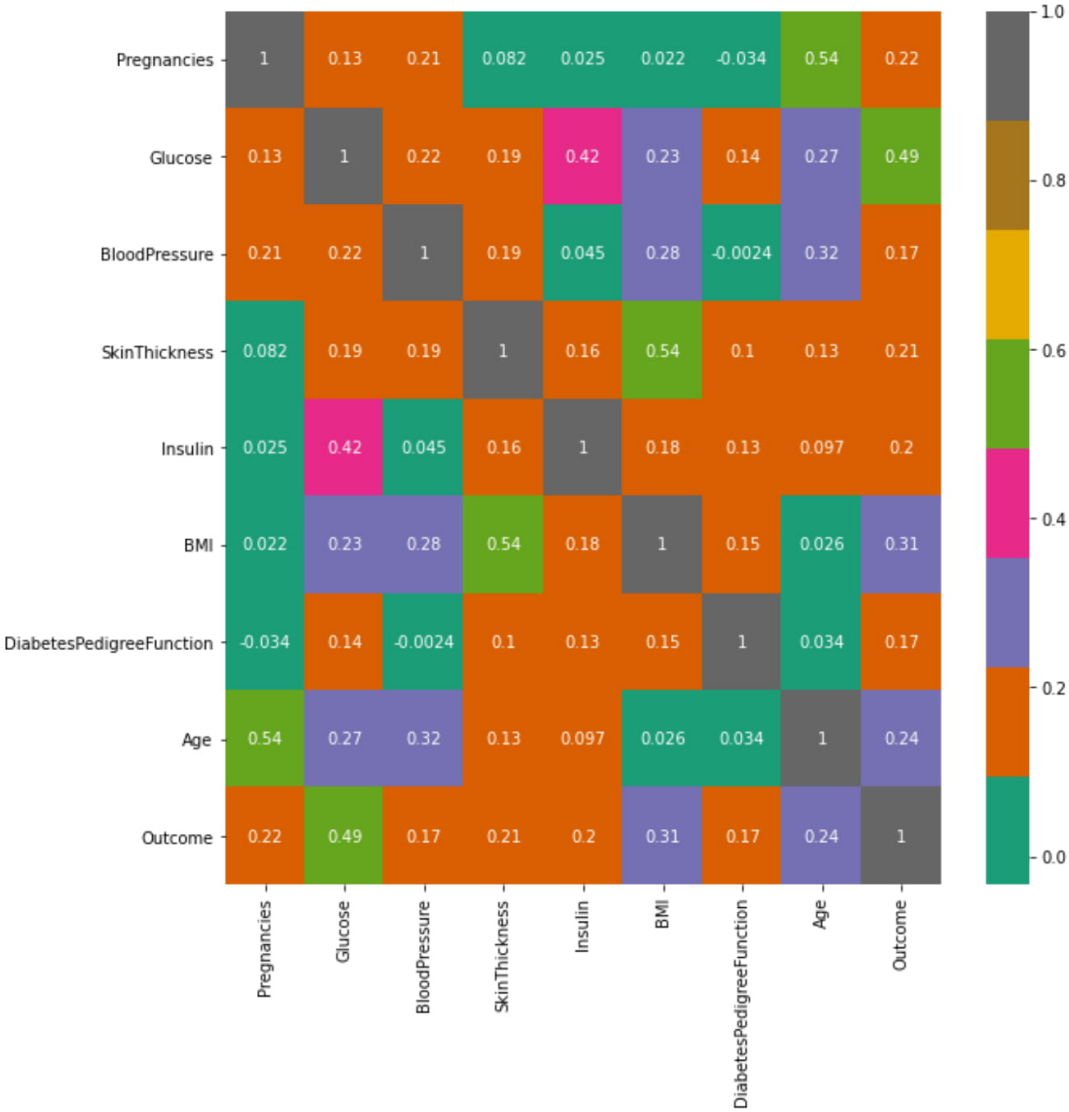
In [11]:
```
df1['Glucose'].fillna(df1['Glucose'].median(), inplace = True)
df1['BloodPressure'].fillna(df1['BloodPressure'].median(), inplace = True)
```

```
df1['SkinThickness'].fillna(df1['SkinThickness'].median(), inplace = True)
df1['Insulin'].fillna(df1['Insulin'].median(), inplace = True)
df1['BMI'].fillna(df1['BMI'].median(), inplace = True)
```

In [12]:
```
his1 = df1.hist(figsize = (20,20))
```



In [13]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df1.corr(), cmap="Dark2", annot= True,)
plt.show()
```



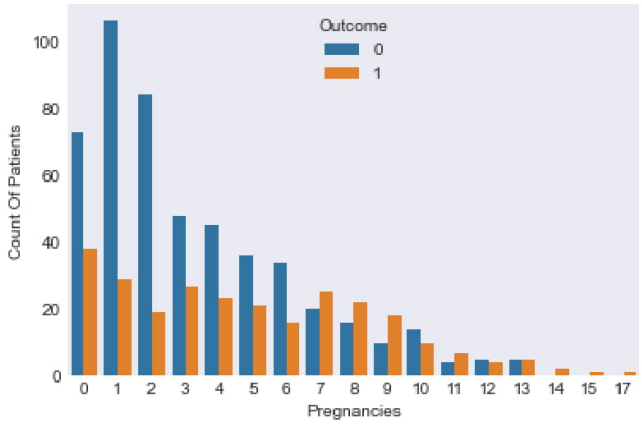
In [14]:
```
df1.corr()['Outcome'].sort_values()
```

Out[14]:
```
BloodPressure               0.165723
DiabetesPedigreeFunction    0.173844
Insulin                     0.203790
SkinThickness               0.214873
Pregnancies                 0.221898
Age                         0.238356
BMI                         0.312038
Glucose                     0.492782
Outcome                     1.000000
Name: Outcome, dtype: float64
```

In [15]:
```
plt.style.use('seaborn-dark')
sns.pairplot(df1,hue='Outcome', palette='husl', diag_kind="hist");
plt.tight_layout()
```

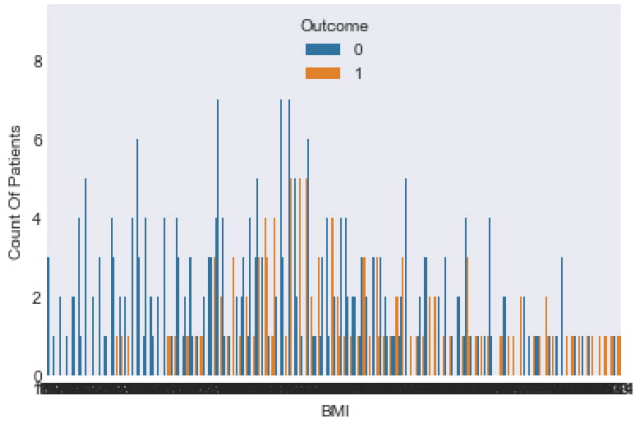
In [16]:
```
sns.countplot(x = 'Pregnancies',hue = 'Outcome',data = df1)

plt.xlabel("Pregnancies")
plt.ylabel("Count Of Patients")
```

Out[16]:
```
Text(0, 0.5, 'Count Of Patients')
```



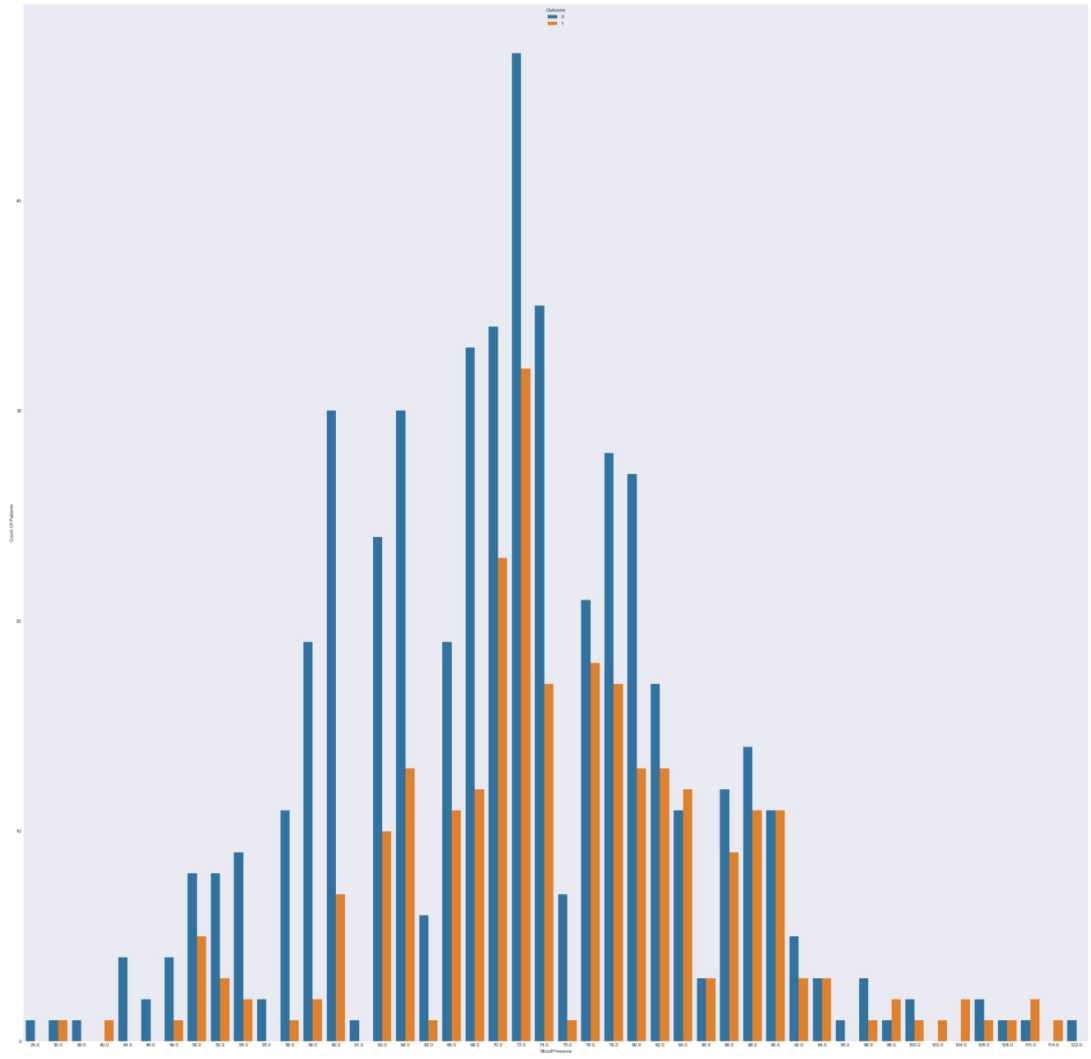
In [17]:
```
sns.countplot(x = 'BMI',hue = 'Outcome',data = df1)

plt.xlabel("BMI")
plt.ylabel("Count Of Patients")
```

Out[17]:
```
Text(0, 0.5, 'Count Of Patients')
```



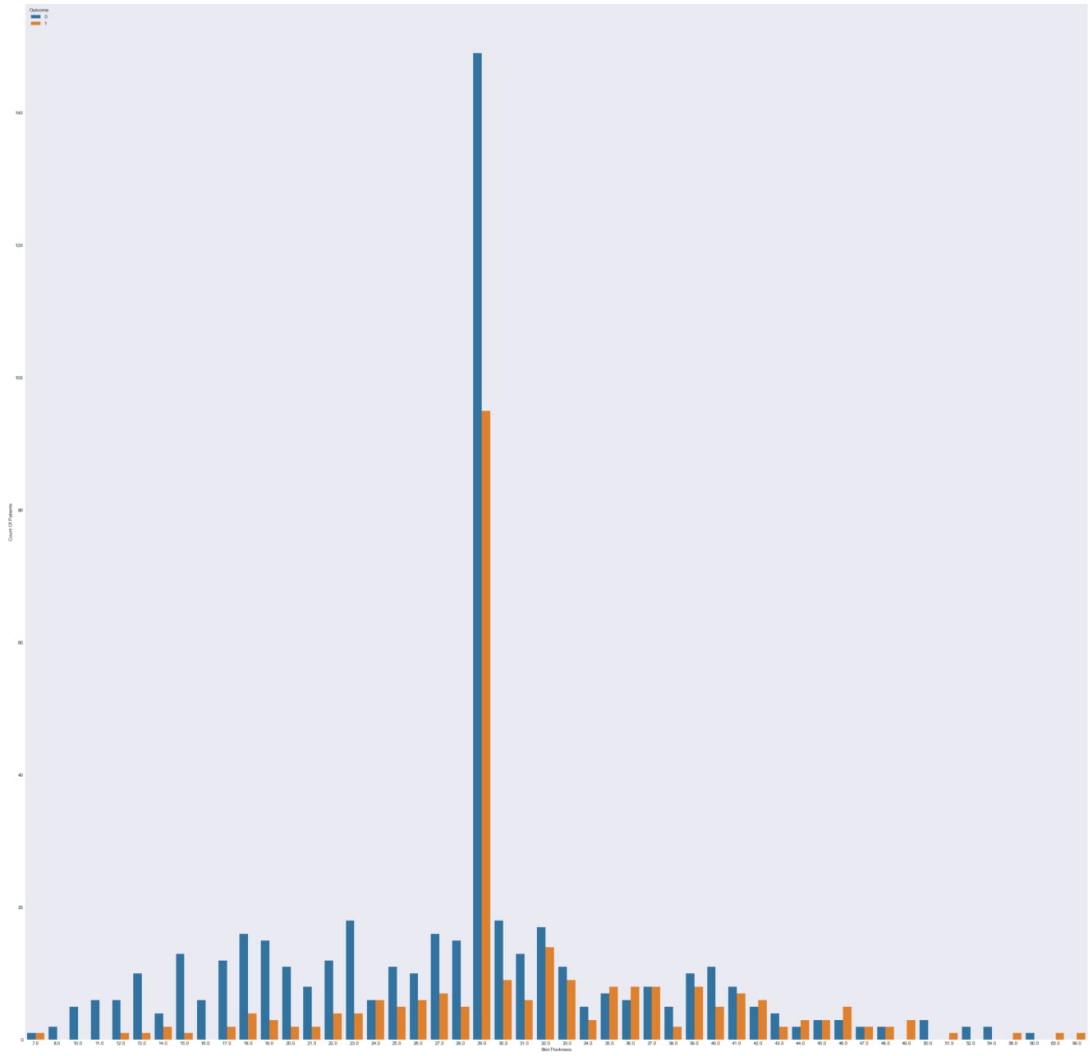
In [18]:
```
plt.figure(figsize=(40,40))
sns.countplot(x = 'BloodPressure',hue = 'Outcome',data = df1)

plt.xlabel("BloodPressure")
plt.ylabel("Count Of Patients")
```

Out[18]:
```
Text(0, 0.5, 'Count Of Patients')
```



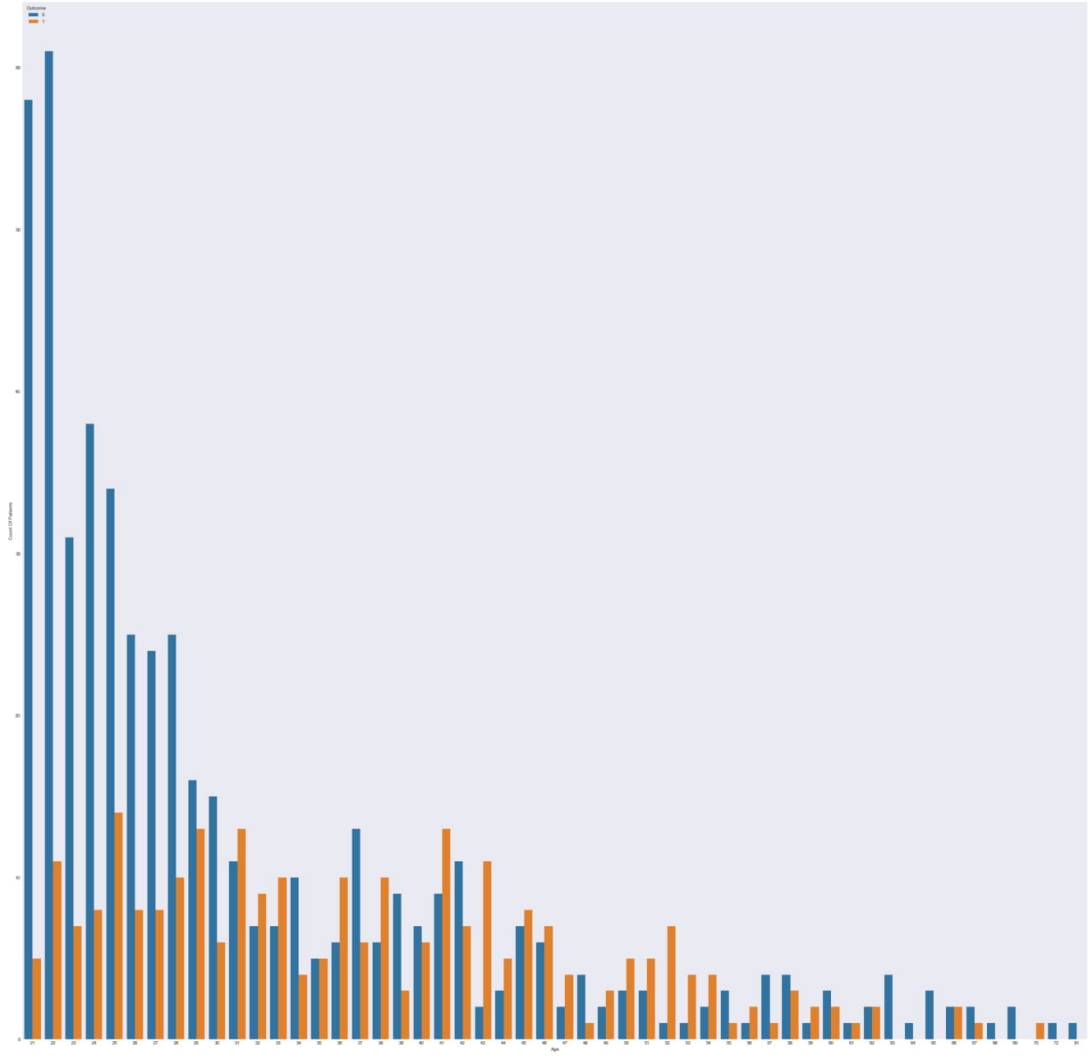
In [19]:
```
plt.figure(figsize=(40,40))
sns.countplot(x = 'SkinThickness',hue = 'Outcome',data = df1)

plt.xlabel("SkinThickness")
plt.ylabel("Count Of Patients")
```

Out[19]:
```
Text(0, 0.5, 'Count Of Patients')
```



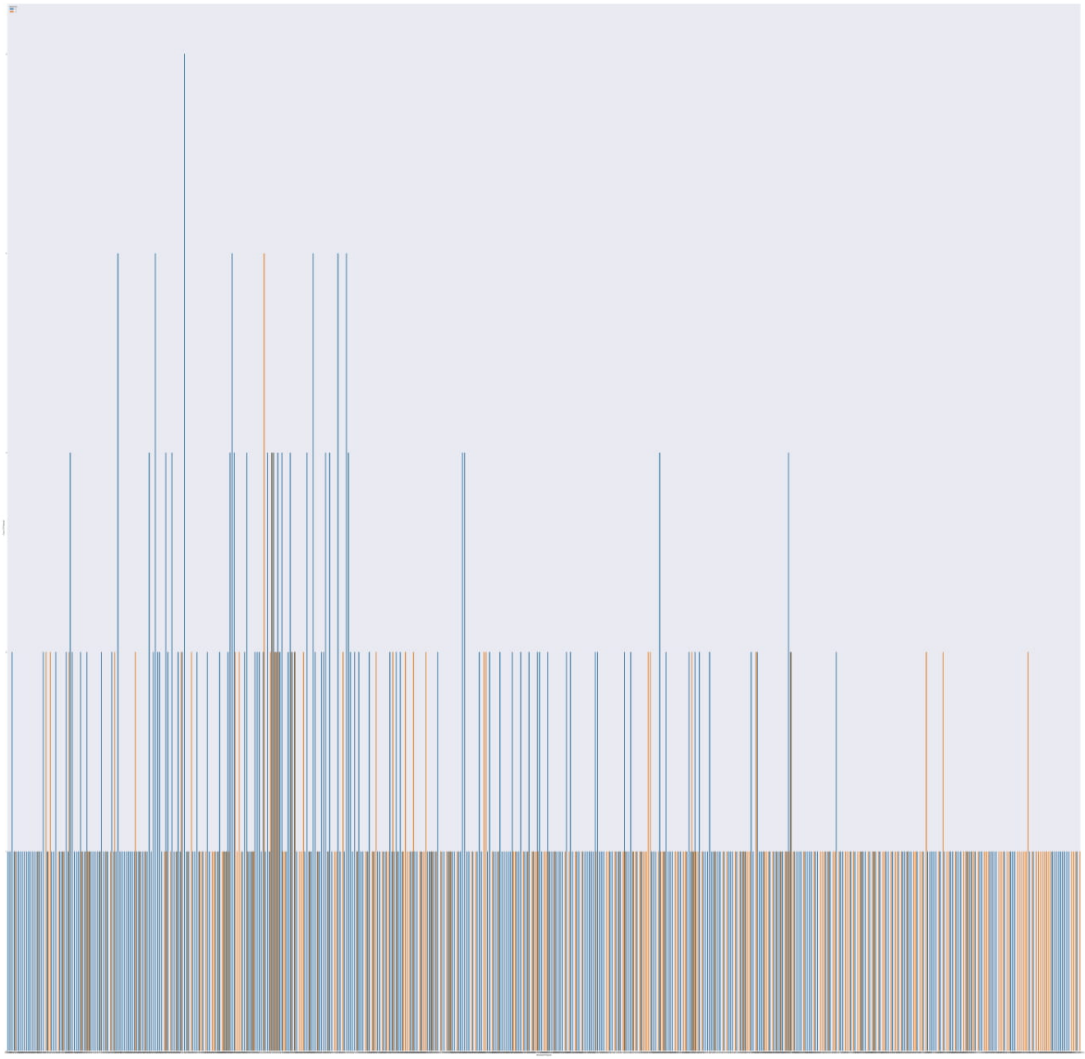
In [20]:
```
plt.figure(figsize=(40,40))
sns.countplot(x = 'Age',hue = 'Outcome',data = df1)

plt.xlabel("Age")
plt.ylabel("Count Of Patients")
```

Out[20]:
```
Text(0, 0.5, 'Count Of Patients')
```



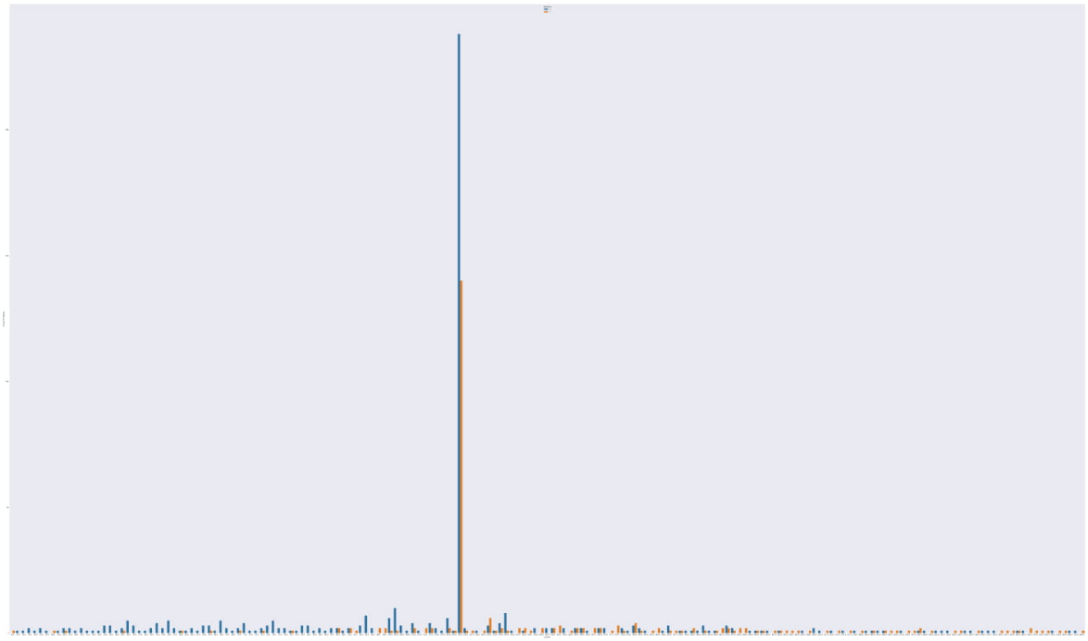
In [21]:
```
plt.figure(figsize=(100,100))
sns.countplot(x = 'DiabetesPedigreeFunction',hue = 'Outcome',data = df1)

plt.xlabel("DiabetesPedigree")
plt.ylabel("Count Of Patients")
```

Out[21]:
```
Text(0, 0.5, 'Count Of Patients')
```



In [22]:
```
plt.figure(figsize=(100,60))
sns.countplot(x = 'Insulin',hue = 'Outcome',data = df1)

plt.xlabel("Insulin")
plt.ylabel("Count Of Patients")
```

Out[22]:
```
Text(0, 0.5, 'Count Of Patients')
```



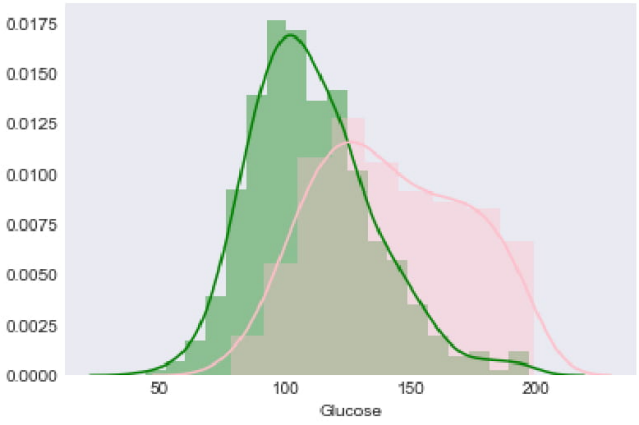
In [23]:
```
df1['Outcome'].value_counts().to_frame()
```

Out[23]:

	Outcome
0	500
1	268

In [24]:
```
sns.distplot(df1[df1['Outcome'] == 0]["Glucose"], color='green') # Healthy - green
sns.distplot(df1[df1['Outcome'] == 1]["Glucose"], color='pink') # Diabetic - Red
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1942ed500d0>
```



In [25]:
```
x = df1.iloc[:, :-1].values
y = df1.iloc[:, -1].values
```

In [26]:
```
x
```

Out[26]:
```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

In [27]:
```
y
```

Out[27]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

In [28]:
```
# Split dataset into training set and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state =
# Scaling to bring values to the same range
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)


print ('Train set:', x_train.shape,  y_train.shape)
print ('Test set:', x_test.shape,  y_test.shape)
```

```
Train set: (537, 8) (537,)
Test set: (231, 8) (231,)
```

KNN Model

In [29]:
```
Ks = 10
```

```
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
for n in range(1,Ks):
    neigh = KNeighborsClassifier(n_neighbors = n).fit(x_train,y_train)
    yhat = neigh.predict(x_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)
    std_acc[n-1] = np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

print(mean_acc)
print("Confusion matrix:\n",confusion_matrix(y_test,yhat))
```

```
[0.68831169 0.70562771 0.72294372 0.75324675 0.77056277 0.77922078
 0.76190476 0.77489177 0.77489177]
Confusion matrix:
 [[125  27]
 [ 25  54]]
```

In [30]:
```
plt.plot(range(1,Ks),mean_acc,'m')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```

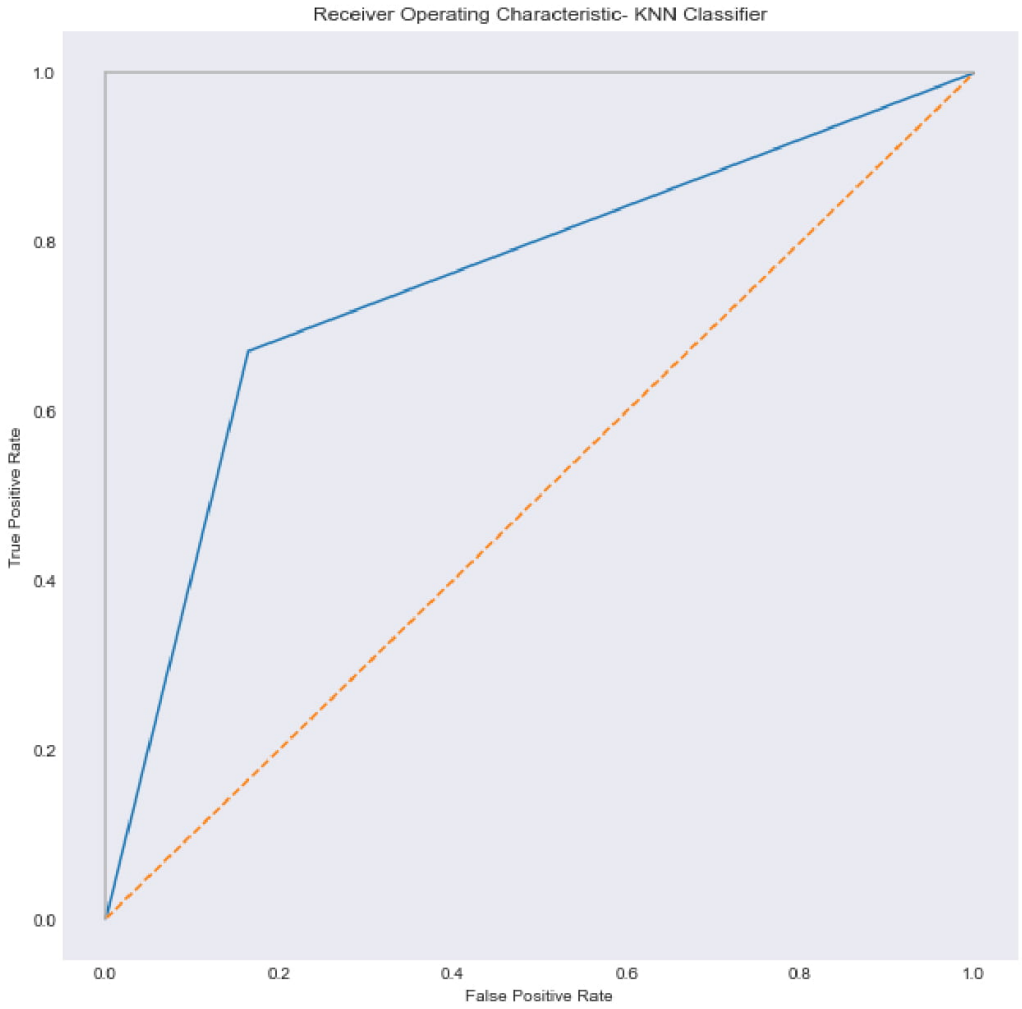
In [31]:
```
# build model with best accuracy, K=6
knn_model = KNeighborsClassifier(n_neighbors=6).fit(x_train, y_train)
yhat = knn_model.predict(x_test)
mean = metrics.accuracy_score(y_test, yhat)
mean
```

Out[31]:
```
0.7792207792207793
```

In [32]:
```
# Plot Receiving Operating Characteristic Curve
    # Create true and false positive rates
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat)
print('roc_auc_score: ', roc_auc_score(y_test, yhat))
# Plot ROC curves
plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic- KNN Classifier')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
```

```
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
roc_auc_score:  0.7532061958694204
```



LogisticRegression

In [33]:
```
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
regularisations = [1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001]
solver_mean_acc = {}
solver_std_acc = {}
solver_best_reg = {}
for solver in solvers:
    best_mean = 0
    best_std = 0
    best_reg = 0
    for reg in regularisations:
        lr = LogisticRegression(C=reg, solver=solver).fit(x_train, y_train)
        yhat = lr.predict(x_test)
        mean = metrics.accuracy_score(y_test, yhat)
        std = np.std(yhat==y_test)/np.sqrt(yhat.shape[0])
        if mean > best_mean:
```

```
            best_mean = mean
            best_std = std
            best_reg = reg
    solver_mean_acc[solver] = best_mean
    solver_std_acc[solver] = best_std
    solver_best_reg[solver] = best_reg

print(solver_mean_acc)
print("Confusion matrix:\n ",confusion_matrix(y_test,yhat))
```

```
{'newton-cg': 0.8181818181818182, 'lbfgs': 0.8181818181818182, 'liblinear': 0.8051948051
948052, 'sag': 0.8181818181818182, 'saga': 0.8181818181818182}
Confusion matrix:
  [[151   1]
 [ 79   0]]
```

In [34]:
```
solver_best_reg
```

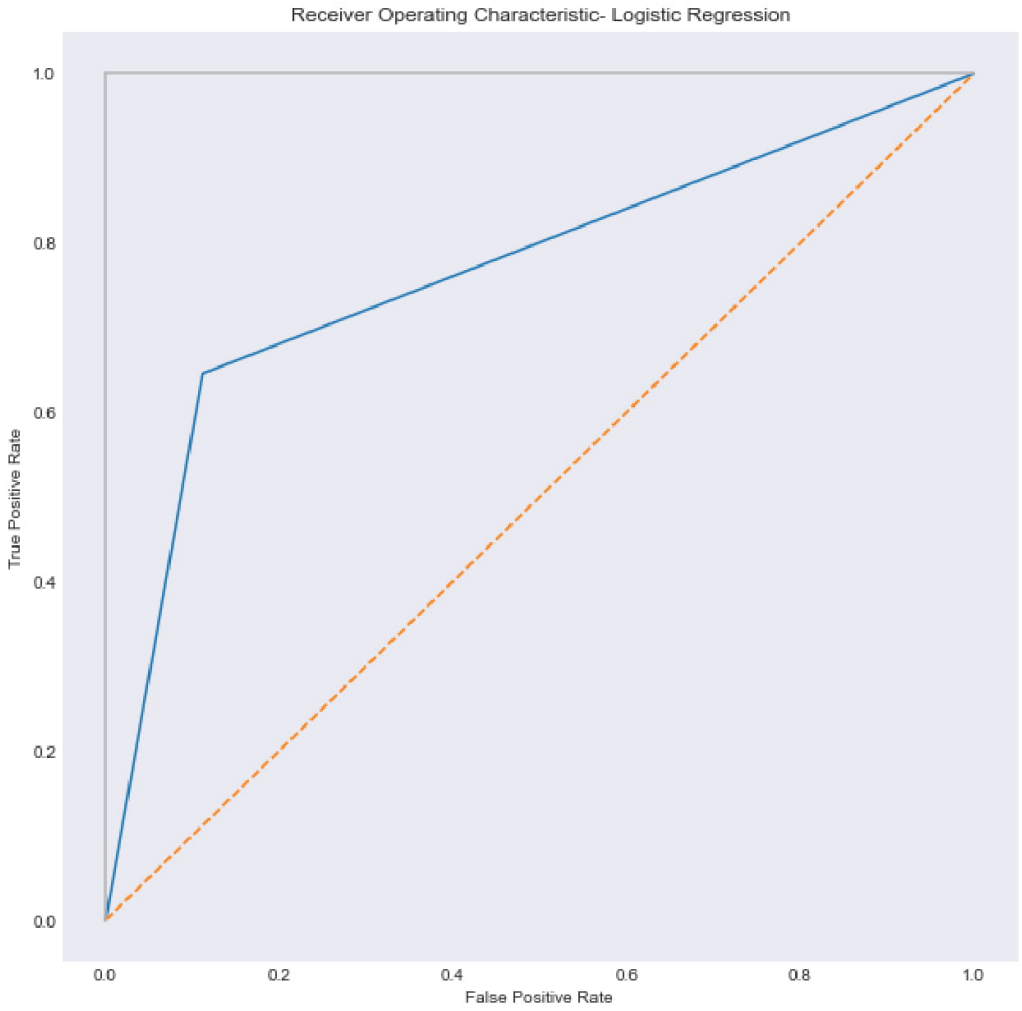
Out[34]:
```
{'newton-cg': 0.01, 'lbfgs': 0.01, 'liblinear': 1, 'sag': 0.01, 'saga': 0.01}
```

In [35]:
```
lr_model = LogisticRegression(C=1, solver='liblinear', max_iter=200).fit(x_train, y_tra
yhat = lr_model.predict(x_test)
mean = metrics.accuracy_score(y_test, yhat)
mean
```

Out[35]:
```
0.8051948051948052
```

In [36]:
```
# Plot Receiving Operating Characteristic Curve
    # Create true and false positive rates
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat)
print('roc_auc_score: ', roc_auc_score(y_test, yhat))
# Plot ROC curves
plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic- Logistic Regression')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
roc_auc_score:  0.7668637574950034
```



RandomForestRegressor

In [37]:
```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 100, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
```

```
                'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
```

In [38]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_ite
# Fit the random search model
rf_random.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:   10.2s finished
```

Out[38]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [10, 20, 30, 40, 50, 60,
                                                         70, 80, 90, 100]},
                   random_state=42, verbose=2)
```

In [39]:
```
rf_random.best_params_
```

Out[39]:
```
{'n_estimators': 90,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 60,
 'bootstrap': True}
```

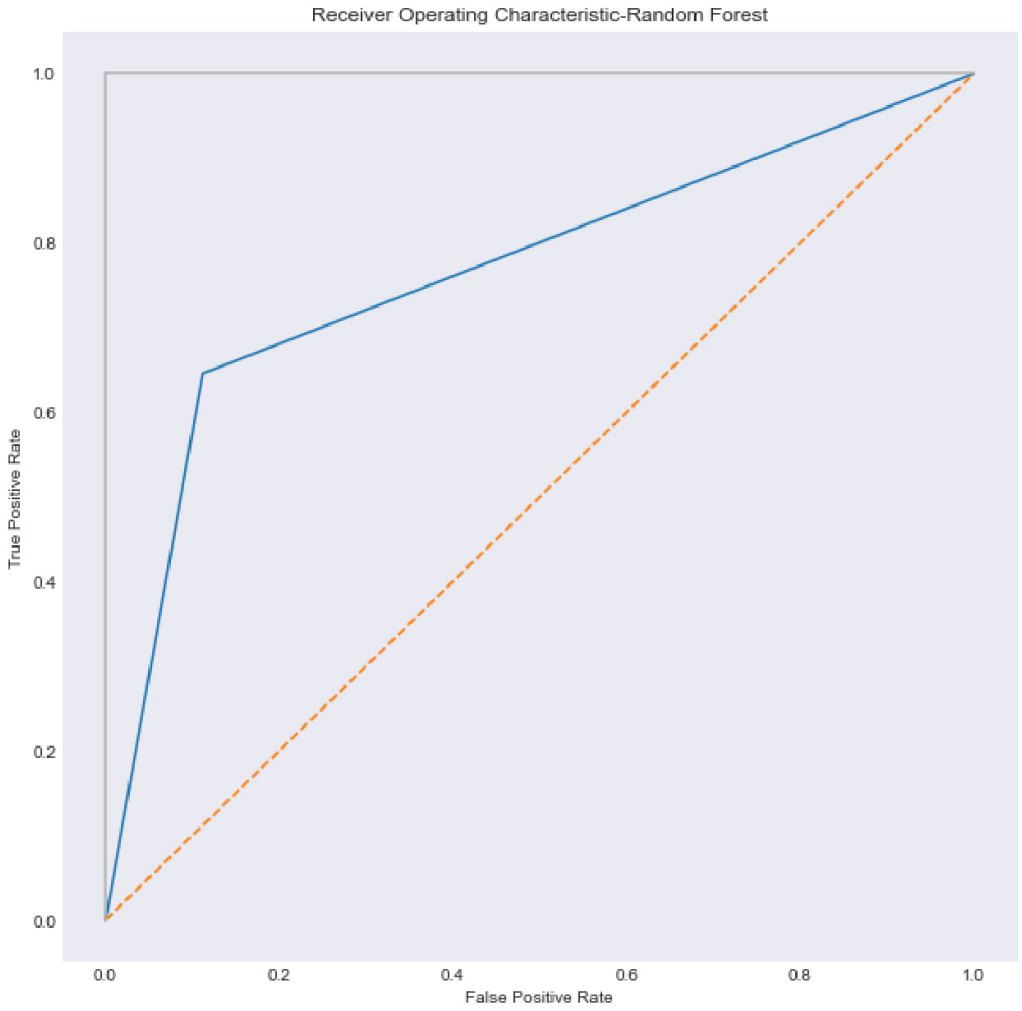
In [40]:
```
def evaluate(model, x_test, y_test):
    yhat = model.predict(x_test)
   # accuracy: (tp + tn) / (p + n)
    accuracy = accuracy_score(y_test, yhat.round())
    print('Accuracy: %f' % accuracy)
    # precision tp / (tp + fp)
    precision = precision_score(y_test, yhat.round())
    print('Precision: %f' % precision)
    # recall: tp / (tp + fn)
    recall = recall_score(y_test, yhat.round())
    print('Recall: %f' % recall)
    # f1: 2 tp / (2 tp + fp + fn)
    f1 = f1_score(y_test, yhat.round(),'weighted')
    print('F1 score: %f' % f1)
    # Jaccard Index
```

```
    jaccard=jaccard_score(y_test, yhat.round(),'weighted')
    print('Jaccard: %f' % jaccard)
    # kappa
    kappa = cohen_kappa_score(y_test, yhat.round())
    print('Cohens kappa: %f' % kappa)
    # ROC AUC
    auc = roc_auc_score(y_test, yhat.round())
    print('ROC AUC: %f' % auc)
    # confusion matrix
    matrix = confusion_matrix(y_test, yhat.round())
    print(matrix)

    return accuracy,precision,recall,f1,jaccard,kappa,auc,matrix
```

In [41]:

```
# Plot Receiving Operating Characteristic Curve
    # Create true and false positive rates
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, yhat)
print('roc_auc_score: ', roc_auc_score(y_test, yhat))
# Plot ROC curves
plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic-Random Forest')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
roc_auc_score:  0.7668637574950034
```



In [44]:
```
df2 = pd.DataFrame(index=['KNN','Logistic Regression','Random Forest', ],
                   columns=['Accuracy','Precision','Recall','Kappa', 'F1-score', 'ROC','

# KNN
yhat = knn_model.predict(x_test)
# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, yhat.round())
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, yhat.round())
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, yhat.round())
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, yhat.round(),'weighted')
print('F1 score: %f' % f1)
# kappa
kappa = cohen_kappa_score(y_test, yhat.round())
print('Cohens kappa: %f' % kappa)
# ROC AUC
auc = roc_auc_score(y_test, yhat.round())
```

```
print('ROC AUC: %f' % auc)
# confusion matrix
matrix = confusion_matrix(y_test, yhat.round())
print(matrix)


df2.loc['KNN'] = [accuracy, precision, recall, kappa, f1, auc, matrix, np.nan]
print("-------------")
# Logistic Regression


yhat = lr_model.predict(x_test)
# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, yhat.round())
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, yhat.round())
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, yhat.round())
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, yhat.round(),'weighted')
print('F1 score: %f' % f1)
# kappa
kappa = cohen_kappa_score(y_test, yhat.round())
print('Cohens kappa: %f' % kappa)
# ROC AUC
auc = roc_auc_score(y_test, yhat.round())
print('ROC AUC: %f' % auc)
# confusion matrix
matrix = confusion_matrix(y_test, yhat.round())
print(matrix)
yhat_prob = lr_model.predict_proba(x_test)
ll = log_loss(y_test, yhat_prob)
print(ll)
df2.loc['Logistic Regression'] = [accuracy, precision, recall, kappa, f1, auc, matrix,l

print("-------------")
# Random Forest
yhat = rf_random.predict(x_test)
# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, yhat.round())
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, yhat.round())
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, yhat.round())
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, yhat.round(),'weighted')
print('F1 score: %f' % f1)
# kappa
kappa = cohen_kappa_score(y_test, yhat.round())
print('Cohens kappa: %f' % kappa)
# ROC AUC
auc = roc_auc_score(y_test, yhat.round())
print('ROC AUC: %f' % auc)
# confusion matrix
matrix = confusion_matrix(y_test, yhat.round())
```

```
print(matrix)
df2.loc['Random Forest'] = [accuracy, precision, recall, kappa,f1, auc, matrix,np.nan]
```

```
Accuracy: 0.779221
Precision: 0.679487
Recall: 0.670886
F1 score: 0.675159
Cohens kappa: 0.507956
ROC AUC: 0.753206
[[127  25]
 [ 26  53]]
-------------
Accuracy: 0.805195
Precision: 0.750000
Recall: 0.645570
F1 score: 0.693878
Cohens kappa: 0.552191
ROC AUC: 0.766864
[[135  17]
 [ 28  51]]
0.43626351838797006
-------------
Accuracy: 0.792208
Precision: 0.686747
Recall: 0.721519
F1 score: 0.703704
Cohens kappa: 0.543854
ROC AUC: 0.775233
[[126  26]
 [ 22  57]]
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:68: FutureWarnin
g: Pass labels=weighted as keyword args. From version 0.25 passing these as positional a
rguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:68: FutureWarnin
g: Pass labels=weighted as keyword args. From version 0.25 passing these as positional a
rguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:68: FutureWarnin
g: Pass labels=weighted as keyword args. From version 0.25 passing these as positional a
rguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

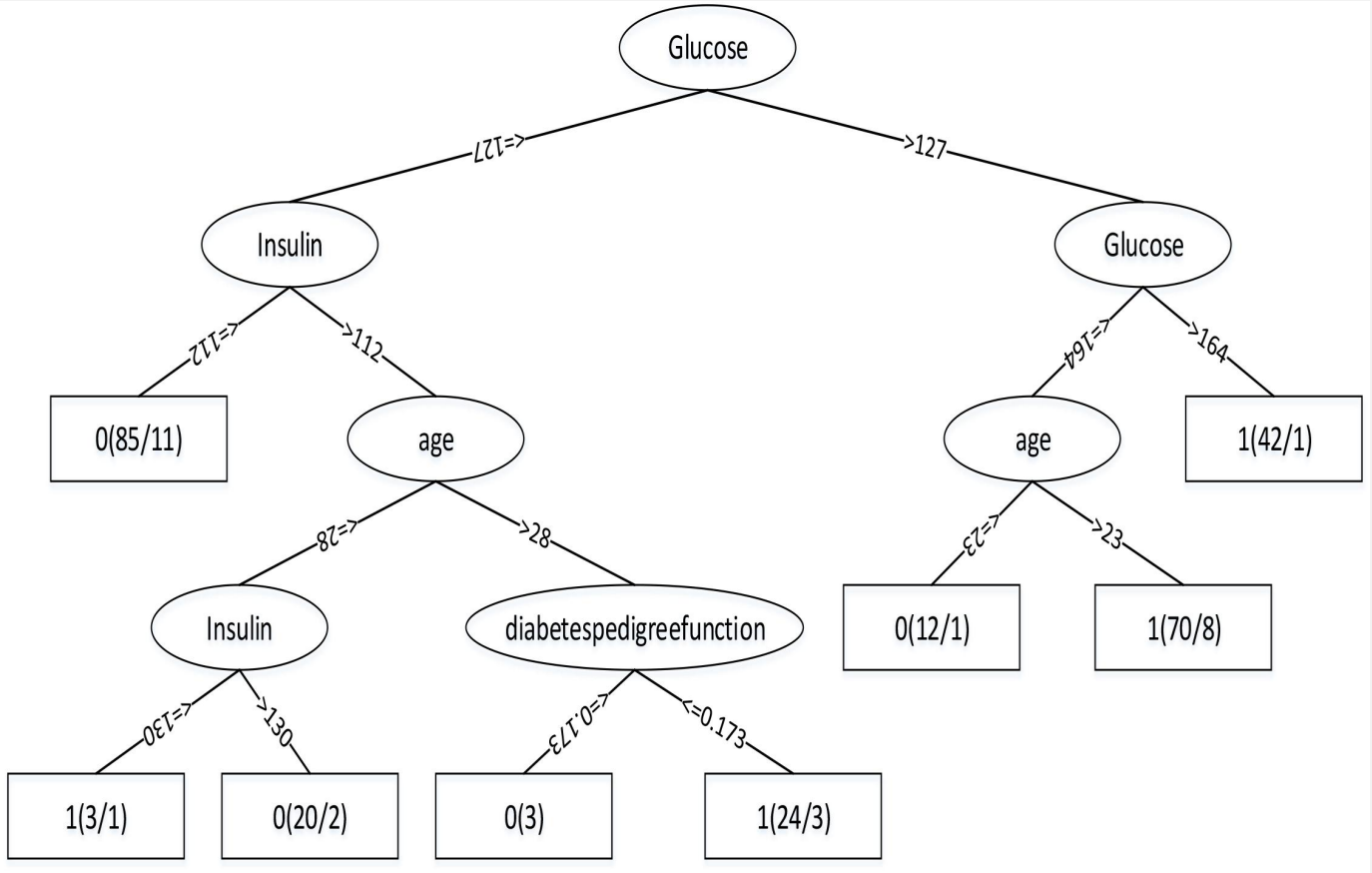
In [45]:

```
df2
```

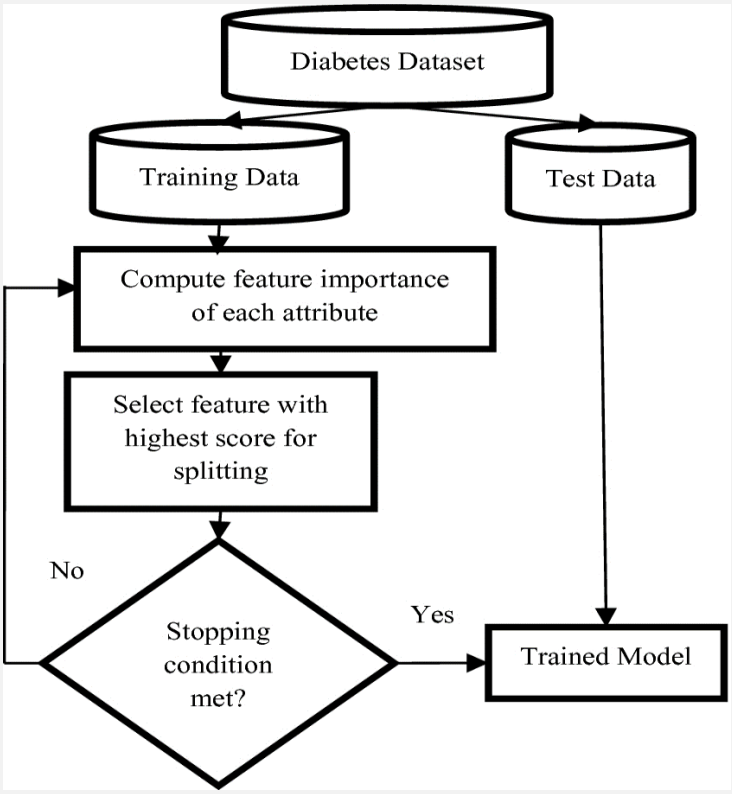
Out[45]:

	Accuracy	Precision	Recall	Kappa	F1-score	ROC	Confusion Matrix	LogLoss
KNN	0.779221	0.679487	0.670886	0.507956	0.675159	0.753206	[[127, 25], [26, 53]]	NaN
Logistic Regression	0.805195	0.75	0.64557	0.552191	0.693878	0.766864	[[135, 17], [28, 51]]	0.436264
Random Forest	0.792208	0.686747	0.721519	0.543854	0.703704	0.775233	[[126, 26], [22, 57]]	NaN

In [ ]:


Glucose
<=127
>127
Insulin
Glucose
<=112
>112
<=164
>164
0(85/11)
age
age
1(42/1)
<=28
>28
<=23
>23
Insulin
diabetespedigreefunction
0(12/1)
1(70/8)
<=130
>130
<=0.173
<=0.173
1(3/1)
0(20/2)
0(3)
1(24/3)

Flow chart


Diabetes Dataset
Training Data
Test Data
Compute feature importance of each attribute
Select feature with highest score for splitting
No
Stopping condition met?
Yes
Trained Model

Future Scope & Advantages

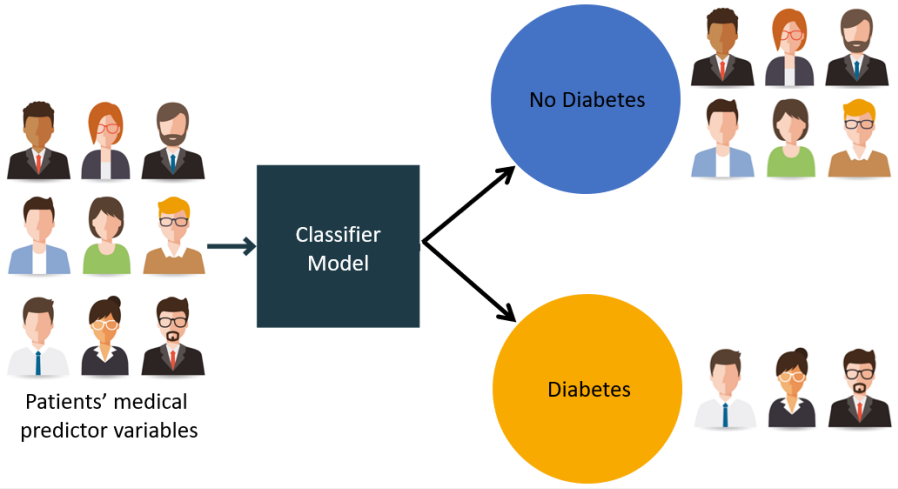
Python has become a formidable language in the data science, artificial intelligence, and machine learning spheres. This is largely due to the language's flexibility and community, but it's also a direct result of the production of many ultra-powerful, high-quality packages and modules. Further considerations should include the situations where data science tasks (analytics, machine learning, artificial intelligence) are carried out either on a local desktop or laptop machine by a data scientist (for example), or where these tasks are performed on servers (usually in the cloud).

This project definitely will open a future path in data science. Working in the Data science industry means dealing with a bunch of data that you need to process in the most convenient and effective way. The low entry barrier allows more data scientists to quickly pick up Python and start using it for AI development without wasting too much effort on learning the language. Python for machine learning development can run on any platform including Windows, MacOS, Linux, Unix, and twenty-one others.

Uses of Diabetes prediction:

Specially we can use this model to predict diabetes of a person before it happens.

Mainly, it would be beneficial for medical industry. After knowing this he or she can take decision calmly. It can also detect it after happened.



Conclusion

One of the important real-world medical problems is the detection of diabetes at its early stage. In this study, systematic efforts are made in designing a system which results in the prediction of disease like diabetes. During this work, three machine learning classification algorithms are studied and evaluated on various measures. Experiments are performed on Pima Indians Diabetes Database. Experimental results determine the adequacy of the designed system with an achieved accuracy of 80.51 % using the Logistic regression algorithm. In future, the designed system with the used machine learning classification algorithms can be used to predict or diagnose other diseases. The work can be extended and improved for the automation of diabetes analysis including some other machine learning algorithms.

The ability of our model to predict patients with Diabetes using some commonly used lab results is high with satisfactory sensitivity. These models can be built into an online computer program to help physicians in predicting patients with future occurrence of diabetes and providing necessary preventive interventions. The model is developed and validated on the population which is more specific and powerful to apply on patients than existing models developed from other populations. Fasting blood glucose, body mass index, high-density lipoprotein, and triglycerides were the most important predictors in these models.

Bibliography

The contents have been gathered from the following:

1.Google Search(https://www.google.com/)

2. Self-performed

3.Youtube Tutorials (https://www.youtube.com/)