About Diabetes

Diabetes is one of the deadliest diseases in the world. It is not only a disease but also creator of different kinds of diseases like heart attack, blindness etc. The normal identifying process is that patients need to visit a diagnostic center, consult their doctor, and sit tight for a day or more to get their report.

Diabetes, also known as diabetes mellitus, is a chronic medical condition characterized by high levels of glucose (sugar) in the blood. It occurs when the body either doesn't produce enough insulin (a hormone that regulates blood sugar) or cannot effectively use the insulin it produces.

There are several types of diabetes:

1. **Type 1 Diabetes:** This is an autoimmune disease where the immune system mistakenly attacks and destroys the insulin-producing cells in the pancreas. As a result, the body cannot produce insulin. Type 1 diabetes typically develops in childhood or adolescence, and people with this condition require insulin injections or an insulin pump to manage their blood sugar levels.
2. **Type 2 Diabetes:** This is the most common type of diabetes, accounting for the majority of cases. Type 2 diabetes occurs when the body becomes resistant to insulin or doesn't produce enough insulin to maintain normal blood sugar levels. It is often associated with lifestyle factors such as obesity, sedentary lifestyle, poor diet, and genetic predisposition. Type 2 diabetes can often be managed with lifestyle changes, oral medications, and sometimes insulin injections.

Cause of Diabetes varies depending on the genetic makeup, family history, ethnicity, health etc. Diabetes & pre-diabetes is diagnosed by blood test.

Introduction:

Diabetes stands as a highly perilous ailment globally, known to trigger various other conditions such as heart attacks and blindness. Typically, patients are required to visit a diagnostic center, consult with a doctor, and endure an extended waiting period to receive their test results.

The Diabetes Prediction System using Machine Learning (ML), Support Vector Machines (SVM), and Random Forest is a project designed to address the challenges associated with diagnosing diabetes. Diabetes is a life-threatening disease that can lead to various complications, including heart attacks and blindness. The conventional diagnostic process involves visiting a diagnostic center, consulting with a doctor, and waiting for an extended period to receive the test reports.

To overcome these limitations and improve the efficiency of diabetes diagnosis, this project utilizes the power of machine learning algorithms, specifically SVM and Random Forest. By analyzing diagnostic measurements, this system aims to accurately predict whether a patient has diabetes or not.

AIM:

The aim of this project is to develop a system capable of promptly detecting the presence of diabetes in a patient, leveraging diagnostic measurements as key indicators. To identify the probability of diabetes in patients using data mining techniques.

Problem Statement:

Diabetes is one of the deadliest diseases in the world. It is not only a disease but also creator of different kinds of diseases like heart attack, blindness etc. The normal identifying process is that patients need to visit a diagnostic center, consult their doctor, and sit tight for a day or more to get their reports. So, the objective of this project is to identify whether the patient has diabetes or not based on diagnostic measurements.

Advantage of this project

The rules derived will be helpful for doctors to identify patients suffering from diabetes. Further predicting the disease early leads to treating the patient before it becomes critical.

Algorithms Used:

As we have to classify the data into patients having diabetes or not, the best method which can be used is SVM and Randomforest, because in this, the dataset is divided into training and testing data. Further we can easily classify and predict the outcome using nodes and internodes.

Objective:

The primary objective of this project is to develop a diabetes prediction system that offers the following benefits:

Early Detection: By leveraging machine learning algorithms, the system can identify the presence of diabetes at an early stage, enabling timely intervention and treatment.

E?ciency: The system eliminates the need for patients to visit a diagnostic center and wait for extended periods to receive their test results. It provides a faster and more efficient alternative for diagnosing diabetes.

Accuracy: By utilizing SVM and Random Forest algorithms, the system aims to achieve high accuracy in diabetes prediction. This will help healthcare professionals make informed decisions regarding patient care and treatment plans.

Accessibility: The diabetes prediction system can be made easily accessible to healthcare facilities, allowing them to efficiently screen a large number of patients for diabetes. This accessibility can aid in managing the disease on a broader scale.

The diabetes prediction system using ML, SVM, and Random Forest has the potential to significantly improve the diagnosis and management of diabetes, ultimately leading to better patient outcomes and a reduction in the associated health risks.

Dataset:

The dataset used has been obtained from UCI Machine Learning Repository having 768 records of Female Patients exclusively.
https://www.kaggle.com/datasets/mathchi/diabetes-data-set/download?datasetVersionNumber=1

Dataset Description:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Pregnancies: Number of times pregnant

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

BloodPressure:

Diastolic blood

pressure (mm Hg)

SkinThickness:

Triceps skin fold thickness (mm)

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

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

DiabetesPedigreeFunction:

Diabetes pedigree function The DiabetesPedigreeFunction is often used as a feature or input in machine learning models to predict the probability of an individual having diabetes. It takes into account factors such as the age and number of diabetic relatives and assigns a numeric value that represents the probability of developing diabetes.

The exact calculation and formulation of the DiabetesPedigreeFunction may vary depending on the specific study or model being used. Different researchers and models may have their own approaches to calculating this function.

The DiabetesPedigreeFunction is often used as a feature or input in machine learning models to predict the probability of an individual having diabetes. It takes into account factors such as the age and number of diabetic relatives and assigns a numeric value that represents the probability of developing diabetes.

Age: Age (years)

Outcome: Class variable (0 or 1)

Number of Instances: 768

Number of Attributes: 8 plus class

For Each Attribute: (all numeric-valued)

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

Missing Attribute Values: Yes

Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

About Jupyter Notebook google colab

Jupyter Notebook is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It provides an interactive computing environment that supports various programming languages, including Python, R, Julia, and others. Jupyter Notebook is widely used in data science, research, education, and other fields for data analysis,
prototyping, and collaborative coding.

The name Jupyter is derived from the combination of three programming languages supported by the application: Julia, Python, and R. The
project originated from IPython, an interactive command-line shell for Python, but has since evolved into a multi-language interactive computing environment.

The key feature of Jupyter Notebook is its ability to create notebooks, which are documents that contain a mix of code, text, equations, and visualizations. These notebooks are organized into cells, where each cell can hold either code or markdown text. Users can write and execute code in the code cells, and the results are displayed inline, allowing for immediate feedback and iterative development.

Markdown cells, on the other hand, enable users to write formatted text using the Markdown markup language. This allows for the creation of rich, well-documented notebooks that can include explanations, instructions, mathematical equations, images, and hyperlinks.

Jupyter Notebook provides an interactive interface that runs in a web browser. It consists of a file browser, a code editor, and the notebook area where the cells are displayed. Users can create new notebooks, open existing ones, and organize them in directories using the file browser. The code editor provides syntax highlighting, auto-completion, and other features to enhance the coding experience.

One of the main advantages of Jupyter Notebook is its ability to mix code with narrative text and visualizations. This makes it a powerful tool
for data exploration, analysis, and storytelling. Researchers and data scientists can document their thought process, explain their methodology, and showcase their findings in a single document, combining code execution with explanatory text and visualizations.

Jupyter Notebook supports the installation of additional libraries and extensions, which further extend its capabilities. These extensions can provide additional functionalities, such as interactive widgets, code snippets, and support for different programming languages. The Jupyter community actively develops and maintains a wide range of extensions, making it a flexible and customizable environment.

Moreover, Jupyter Notebook promotes collaboration and reproducibility. Notebooks can be easily shared with others, allowing for collaborative editing and reviewing. They can be exported to different formats, such as HTML or PDF, making it easy to share results with individuals who do not have Jupyter Notebook installed. Additionally, Jupyter Notebook supports version control systems like Git, enabling efficient collaboration and tracking of changes over time.

In recent years, Jupyter Notebook has gained significant popularity and has become a standard tool in the data science community. Its
versatility, interactivity, and collaborative features make it an essential tool for data exploration, analysis, and communication. Jupyter Notebook empowers users to combine code, visualizations, and narrative text in a single document, facilitating reproducibility, knowledge sharing, and interdisciplinary collaboration.

Software Used

Python-Scikit Learn

IDE: jupytr

notebook with

google

Colaboratory

Cloud: google

Cloude

About Google Colabratory

Google Colaboratory, also known as Google Colab, is a cloud-based platform provided by Google for running Python code in a Jupyter Notebook environment. It offers free access to computing resources, including CPU, GPU, and high-memory options, making it popular among
researchers, data scientists, and developers.

Here are some key features and aspects of Google Colaboratory:

Notebook Environment: Google Colab provides an interactive environment for creating and running Jupyter notebooks. Notebooks allow you to combine executable code, visualizations, and explanatory text in a single document.

Cloud-based: Colab runs entirely in the cloud, which means you don't need to install any software on your local machine. It leverages Google's infrastructure to execute code, making it convenient and accessible from anywhere with an internet connection.

Free Resource Allocation: Colab offers free access to computing resources, including a CPU runtime and limited GPU support. These resources have some usage limitations, such as time limits and restrictions on GPU availability, but they are usually sufficient for many tasks.

Python Support: Colab supports Python programming language and allows you to write, execute, and debug Python code within the notebook interface. It also provides access to popular Python libraries and frameworks such as TensorFlow, PyTorch, NumPy, and Pandas.

Collaboration and Sharing: Colab allows you to share notebooks with others and collaborate in real-time. You can grant permission to specific individuals or even make your notebooks public, which facilitates collaboration and knowledge sharing.

Integration with Google Drive: Colab integrates seamlessly with Google Drive, allowing you to save and load notebooks directly from your Google Drive storage. This feature simplifies file management and makes it easy to access your work from different devices.

Hardware Acceleration: Colab provides access to GPUs, which can significantly speed up computations for machine learning and deep learning tasks. You can enable GPU acceleration within your notebook by selecting the appropriate runtime type.

Package Installation: Colab allows you to install additional Python packages using pip or conda. This flexibility lets you utilize a wide range of libraries and tools beyond the pre-installed ones.

Code Snippets and Examples: Colab offers code snippets and example notebooks to help you get started quickly. These resources cover various topics, including machine learning, data analysis, and visualization.

Limitations: While Google Colab provides a powerful environment for experimentation, it has certain limitations. The resources allocated for free users may be limited, and long-running processes or heavy workloads may face restrictions. Additionally, there is no guarantee of data persistence, and instances may be terminated after a period of inactivity.

Library used

```
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection
import train_test_splitfrom
sklearn import svm
from sklearn.metrics import
confusion_matrix,classification_report,roc_curve,accuracy_score,aucfrom
sklearn.ensemble import RandomForestClassifier
from
sklearn.metrics
import
accuracy_scorefrom
os.path import
join
from google.colab import drive
```

Here is a description of each library used in the code:

numpy (imported as np): NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

pandas (imported as pd): Pandas is a powerful library for data manipulation and

analysis. It offers data structures like data frames, which allow easy handling and manipulation of structured data. Pandas provides various functions to read, write, and manipulate data, making it an
essential tool for data preprocessing.

StandardScaler from sklearn.preprocessing: StandardScaler is a class from the sklearn.preprocessing module in scikit-learn. It is used for standardizing features by removing the mean and scaling to unit variance. It is commonly applied to normalize the input data before training a machine learning model.

train_test_split from sklearn.model_selection: train_test_split is a function from the sklearn.model_selection module. It is used to split the data into training and testing sets. This function randomly shuffles the data and splits it into two portions, allowing evaluation of the model's
performance on unseen data.

svm from sklearn: svm is a module within scikit-learn that provides classes for Support Vector Machines (SVM). SVM is a powerful machine learning algorithm used for classification and regression tasks. It aims to find the optimal hyperplane that separates data points into different classes.

confusion_matrix, classification_report, roc_curve, accuracy_score, auc from sklearn.metrics: These are functions from the sklearn.metrics module in scikit-learn. They are used for evaluating the performance of machine learning models.

confusion_matrix computes a confusion matrix, which is a table that summarizes the performance of a classification model by displaying the counts of true positives, true negatives, false positives, and false negatives. classification_report generates a text report showing various classification metrics such as precision, recall, F1-score, and support. roc_curve computes the Receiver Operating Characteristic (ROC) curve, which is used to evaluate the performance of a binary classification model across different probability thresholds. accuracy_score computes the accuracy of a classification model by comparing the predicted labels with the true labels. auc computes the Area Under the ROC Curve
(AUC), which is a metric used to evaluate the performance of a binary classification model. RandomForestClassifier from sklearn.ensemble: RandomForestClassifier is a class from the sklearn.ensemble module in scikit-learn. It implements the Random

Forest algorithm, which is an ensemble learning method that combines multiple decision trees to make predictions. Random Forest is commonly used for classification tasks.

os.path module: The os.path module provides functions for manipulating file paths. In this case, the join function from os.path is used to concatenate directory paths and filenames in a cross-platform compatible way.

google.colab module: The google.colab module provides functionality specific to running code in Google Colaboratory. In this case, the drive function is imported from this module, which allows accessing files and directories stored on Google Drive.

These libraries and modules are essential for data handling, preprocessing, model training, evaluation, and accessing external resources in the code.

Data Collection and Analysis

```
drive.mount('/content/drive/')

     Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/",
     force_remount=True).
```

```
# loading the diabetes dataset to a pandas DataFrame
df = pd.read_csv('/content/drive/My Drive/Diabetes/diabetes.csv')
diabetes_dataset = pd.read_csv('/content/drive/My Drive/Diabetes/diabetes.csv')
```

```
# printing the
first 5 rows of
the dataset
diabetes_dataset.h
ead()
```

0

1

	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

```
# number of rows and Columns in this datasetdiabetes_dataset.shape

     (768, 9)
```

Data Cleaning

Data Cleaning will take place as data has got lot of missing values. Handling missing values can be done either by replacing null values withmode or mean or replacing the null value with a random variable.

Data Cleaning Replacing 0 with NaN

```
col=['Glucose','BloodPressure','Insulin
','BMI','SkinThickness']for i in col:
    diabetes_dataset[i].replace(0, np.nan, inplace= True)
```

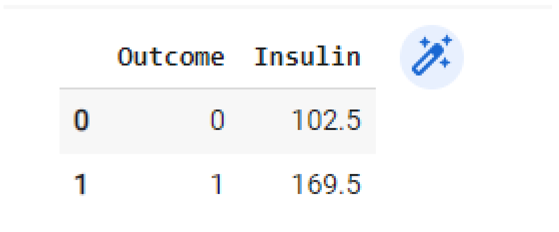
Data Analysis

Function to calculate Median according to the Output

Median = {(n + 1) ÷ 2}th value The median is the 3.5th value in the data set meaning that it lies between the third and fourth values. Thus, the median is calculated by averaging the two middle values of 25.2 and 25.6. Use the formula below to get the average value.

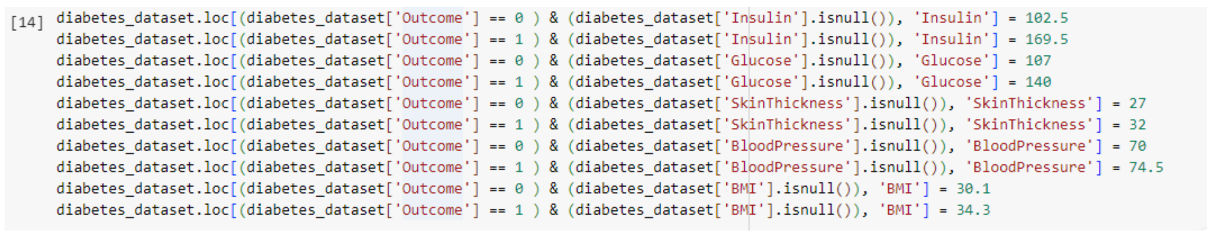
```
def median_target(var):
    temp = diabetes_dataset[diabetes_dataset[var].notnull()]
    temp = temp[[var,
    'Outcome']].groupby(['Outcome'])[[var]].median().reset_i
    ndex()return temp
```

```
#Here we can find the
median value according to
colmedian_target('Insulin')
```



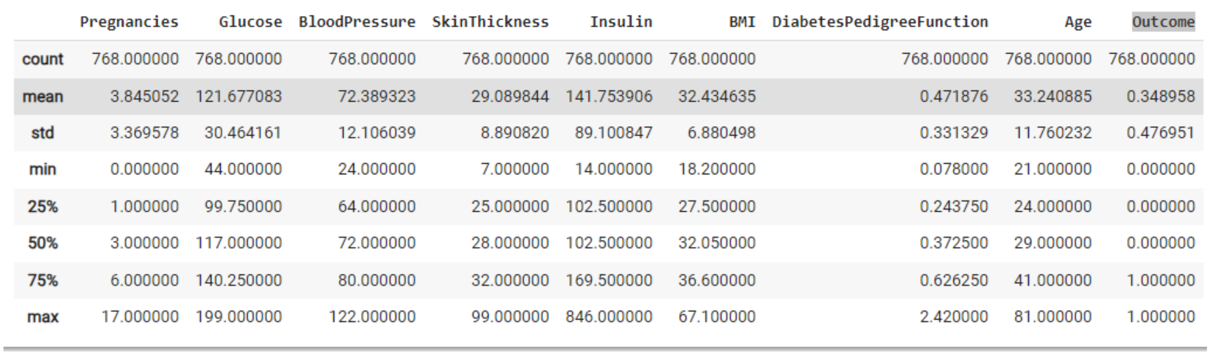
	Outcome	Insulin
0	0	102.5
1	1	169.5

Filling the NaN value with Median according to Output



```
[14] diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 0 ) & (diabetes_dataset['Insulin'].isnull()), 'Insulin'] = 102.5
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 1 ) & (diabetes_dataset['Insulin'].isnull()), 'Insulin'] = 169.5
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 0 ) & (diabetes_dataset['Glucose'].isnull()), 'Glucose'] = 107
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 1 ) & (diabetes_dataset['Glucose'].isnull()), 'Glucose'] = 140
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 0 ) & (diabetes_dataset['SkinThickness'].isnull()), 'SkinThickness'] = 27
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 1 ) & (diabetes_dataset['SkinThickness'].isnull()), 'SkinThickness'] = 32
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 0 ) & (diabetes_dataset['BloodPressure'].isnull()), 'BloodPressure'] = 70
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 1 ) & (diabetes_dataset['BloodPressure'].isnull()), 'BloodPressure'] = 74.5
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 0 ) & (diabetes_dataset['BMI'].isnull()), 'BMI'] = 30.1
     diabetes_dataset.loc[(diabetes_dataset['Outcome'] == 1 ) & (diabetes_dataset['BMI'].isnull()), 'BMI'] = 34.3
```

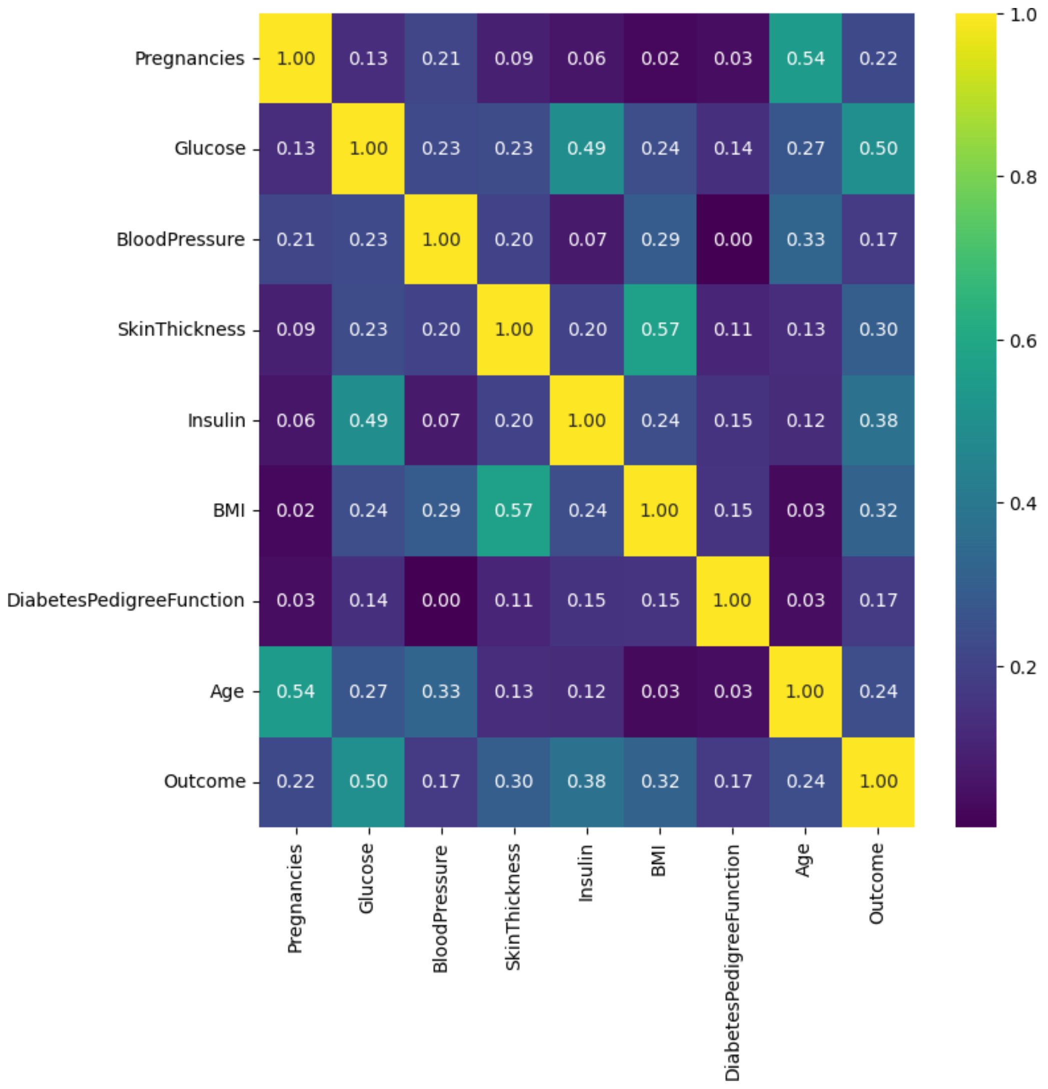
```
# getting the statistical measures of the data
diabetes_dataset.describe()
```



	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	768.000000
mean	3.845052	121.677083	72.389323	29.089844	141.753906	32.434635	0.471876	33.240885	0.348958
std	3.369578	30.464161	12.106039	8.890820	89.100847	6.880498	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	102.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	28.000000	102.500000	32.050000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	169.500000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

To generate headmap code

```
from matplotlib.pyplot import figure
plt.figure(figsize=(8,8))
sns.heatmap(np.abs(diabetes_dataset.corr()),annot=True,cmap="viridis",fmt="0.2f");
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.00	0.13	0.21	0.09	0.06	0.02	0.03	0.54	0.22
Glucose	0.13	1.00	0.23	0.23	0.49	0.24	0.14	0.27	0.50
BloodPressure	0.21	0.23	1.00	0.20	0.07	0.29	0.00	0.33	0.17
SkinThickness	0.09	0.23	0.20	1.00	0.20	0.57	0.11	0.13	0.30
Insulin	0.06	0.49	0.07	0.20	1.00	0.24	0.15	0.12	0.38
BMI	0.02	0.24	0.29	0.57	0.24	1.00	0.15	0.03	0.32
DiabetesPedigreeFunction	0.03	0.14	0.00	0.11	0.15	0.15	1.00	0.03	0.17
Age	0.54	0.27	0.33	0.13	0.12	0.03	0.03	1.00	0.24
Outcome	0.22	0.50	0.17	0.30	0.38	0.32	0.17	0.24	1.00

heatmap graph:

generates a square correlation heatmap, visually representing the absolute correlation values between variables in the diabetes_dataset DataFrame. The higher the absolute value of the correlation, the stronger the relationship between the variables. The annotations provide additional information about the correlation values.

```
#sns.pairplot(diabetes_dataset,hue='Outcome',diag_kind='hist')
```

Graph of all attributes pair

```
diabetes_dataset['Outcome'].value_counts()
#diabetes_dataset['Outcome'].value_counts() returns the count of unique values in
#the 'Outcome' column of the dataset named "diabetes_dataset". It provides a summary of the distribution of values
in
#the 'Outcome' column, which is often used to analyze the imbalance or prevalence of a specific outcome or target
variable.

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

matplotlib.pyplot (imported as plt): matplotlib.pyplot is a plotting library that provides a wide range of functions for creating various types of plots and visualizations. It is a powerful tool for creating line plots, scatter plots, bar plots, histograms, and more. The plt alias is commonly used for convenience.

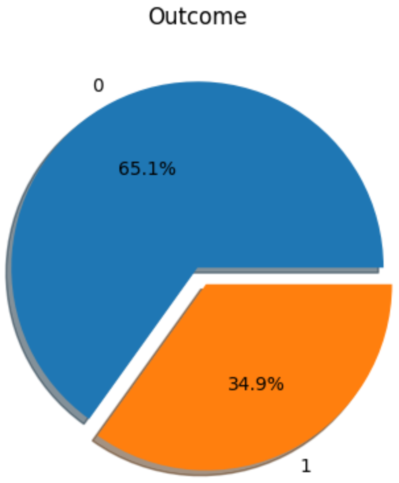
seaborn: seaborn is a Python data visualization library that builds on top of matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex plots such as heatmaps, box plots, violin plots, and joint distributions. It offers a wide range of built-in themes and color palettes that enhance the aesthetics of the visualizations.

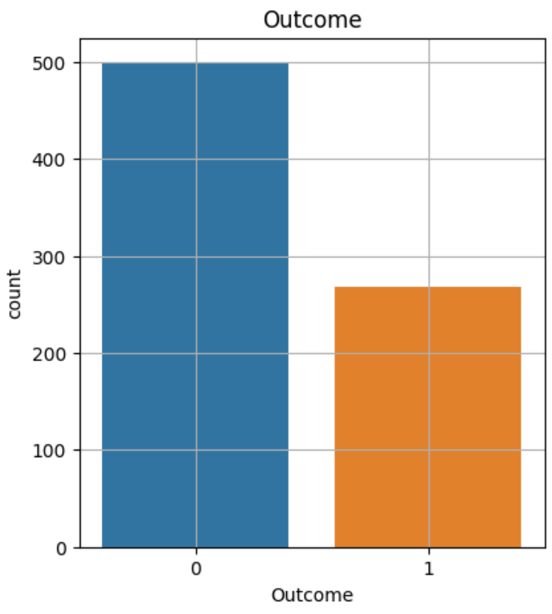
```
import matplotlib.pyplot as pltimport seaborn as sns

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

diabetes_dataset['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('')

sns.countplot(data=diabetes_dataset
, x='Outcome', ax=ax[1])
ax[1].set_title('Outcome')

N, P =
diabetes_dataset['Outcome
'].value_counts()
print('Neg (0):', N)
print('Positive (1):', P)
```

plt.grid() plt.show()

```
     Neg (0): 500
```

```
Positive (1): 268
```



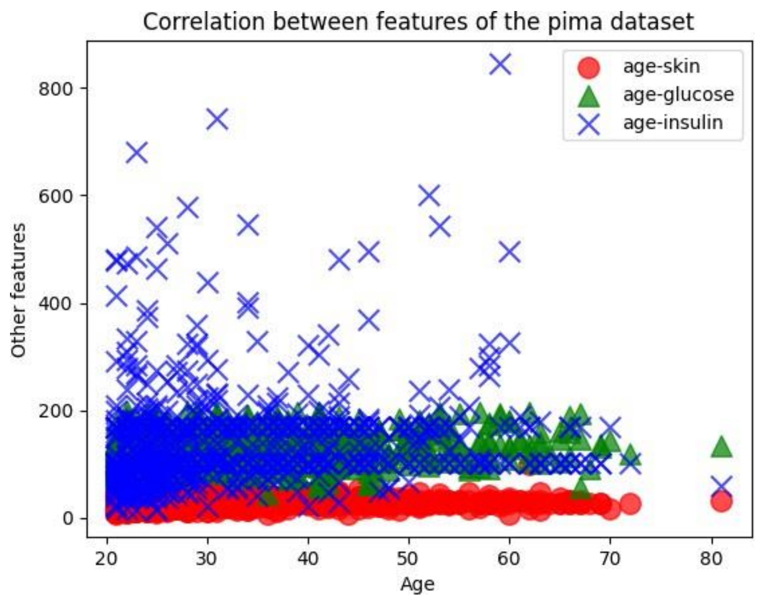


0 --> No-Diabetic

1 --> Yes-Diabetic

Above the graph show the analysis of outcome means class of result. ther are two class 0- No Diabetic and 1- Yes Diabetic

```
# generate scatter plots to find the correlation between different features

plt.figsize=(10, 5)
plt.scatter(diabetes_dataset['Age'], diabetes_dataset['SkinThickness'], marker = 'o', color= 'r', alpha= 0.7, s=
124, label= "age-skin")plt.scatter(diabetes_dataset['Age'], diabetes_dataset['Glucose'], marker = '^', color=
'g', alpha= 0.7, s= 124, label= "age-glucose")
plt.scatter(diabetes_dataset['Age'], diabetes_dataset['Insulin'], marker = 'x', color= 'b', alpha= 0.7, s=

124, label= "age-insulin")plt.title('Correlation between features of the pima dataset')

plt.xlabel('Age')
plt.ylabel('Other features') plt.legend(loc='upper right')
```



The above code generates a visually appealing and informative scatter plot that aids in understanding the correlations between age and different features in the Pima dataset.

```
#mean Value
diabetes_dataset.groupby('Outcome').mean()
```

```
         Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin        BMI  DiabetesPedigreeFunction        Age
Outcome
   0        3.298000  110.622000      70.844000      27.170000  117.172000  30.846000                  0.429734  31.190000
   1        4.865672  142.302239      75.272388      32.671642  187.615672  35.398507                  0.550500  37.067164
```

The code diabetes_dataset.groupby('Outcome').mean() groups the dataset named "diabetes_dataset" by the unique values in the 'Outcome' column and calculates the mean (average) of the other numeric columns for each group. This operation provides insights into the average values of the different features based on the outcome groups.

The output of diabetes_dataset.groupby('Outcome').mean() will be a table or dataframe that displays the mean values of each numeric column, grouped by the unique values in the 'Outcome' column.

```
# separating the data and labels
X =
diabetes_dataset.drop(columns
= 'Outcome', axis=1)Y =
diabetes_dataset['Outcome']
```

```
print(X)

         Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
    0              6    148.0           72.0           35.0    169.5  33.6
    1              1     85.0           66.0           29.0    102.5  26.6
    2              8    183.0           64.0           32.0    169.5  23.3
    3              1     89.0           66.0           23.0     94.0  28.1
    4              0    137.0           40.0           35.0    168.0  43.1
    ..           ...      ...            ...            ...      ...   ...
    763           10    101.0           76.0           48.0    180.0  32.9
    764            2    122.0           70.0           27.0    102.5  36.8
    765            5    121.0           72.0           23.0    112.0  26.2
    766            1    126.0           60.0           32.0    169.5  30.1
    767            1     93.0           70.0           31.0    102.5  30.4

         DiabetesPedigreeFunction  Age
    0                       0.627   50
    1                       0.351   31
    2                       0.672   32
    3                       0.167   21
    4                       2.288   33
    ..                        ...  ...
    763                     0.171   63
    764                     0.340   27
    765                     0.245   30
    766                     0.349   47
    767                     0.315   23

    [768 rows x 8 columns]
```

```
print(Y)

    0      1
    1      0
    2      1
    3      0
    4      1
          ..
    763    0
    764    0
    765    0
    766    1
    767    0
    Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators

```
#initilize the model
scaler = StandardScaler()
```

```
#Compute the mean and std to be
used for later scaling.
scaler.fit(X)
```

The code scaler.fit(X) is used to fit a data scaler to a dataset X. Here, scaler refers to an instance of a scaler class, such as StandardScaler fromscikit-learn.

Data scaling is a common preprocessing step in machine learning that aims to normalize the features of a dataset. Scaling ensures that all

features have similar scales and ranges, which can improve the performance of certain machine learning algorithms and prevent features with larger scales from dominating the learning process.

To apply data scaling, the fit() method is used. It analyzes the data in X and computes the necessary parameters for scaling, such as the mean and standard deviation for StandardScaler. These parameters are calculated based on the values in X.

After calling scaler.fit(X), the scaler object is fitted to the data, and it can be used to transform other datasets or perform scaling operations on new data using the learned parameters. The fit() method is typically followed by the transform() or fit_transform() method to apply the scaling to the dataset.

```
#Perform standardization
by centering and scaling.
standardized_data =
scaler.transform(X)
```

```
print(standardized_data)

     [[ 0.63994726  0.86462486 -0.03218035 ...  0.16948251  0.46849198
        1.4259954 ]
      [-0.84488505 -1.20472661 -0.52812374 ... -0.84854874 -0.36506078
       -0.19067191]
      [ 1.23388019  2.01426457 -0.69343821 ... -1.32847775  0.60439732
       -0.10558415]
      ...
      [ 0.3429808  -0.02224005 -0.03218035 ... -0.90672195 -0.68519336
       -0.27575966]
      [-0.84488505  0.14199419 -1.02406713 ... -0.33953311 -0.37110101
        1.17073215]
      [-0.84488505 -0.94195182 -0.19749482 ... -0.2959032  -0.47378505
       -0.87137393]]


X = standardized_data
Y = diabetes_dataset['Outcome']
```

print(X) print(Y)

```
     [[ 0.63994726  0.86462486 -0.03218035 ...  0.16948251  0.46849198
        1.4259954 ]
      [-0.84488505 -1.20472661 -0.52812374 ... -0.84854874 -0.36506078
       -0.19067191]
      [ 1.23388019  2.01426457 -0.69343821 ... -1.32847775  0.60439732
       -0.10558415]
      ...
      [ 0.3429808  -0.02224005 -0.03218035 ... -0.90672195 -0.68519336
       -0.27575966]
      [-0.84488505  0.14199419 -1.02406713 ... -0.33953311 -0.37110101
        1.17073215]
      [-0.84488505 -0.94195182 -0.19749482 ... -0.2959032  -0.47378505
       -0.87137393]]
     0      1
     1      0
     2      1
     3      0
     4      1
           ..
     763    0
     764    0
     765    0
     766    1
     767    0
     Name: Outcome, Length: 768, dtype: int64
```

Train Test Split

The train-test split is a common technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing a dataset into two subsets: a training set and a testing (or validation) set. The training set is used to train the model, while the testing set is used to assess its performance and generalization ability.

The purpose of the train-test split is to simulate how well the trained model would perform on new, unseen data. By evaluating the model on the testing set, which was not used during training, we can estimate its performance on real-world data and detect any potential issues, such as overfitting or underfitting.

The typical ratio for the train-test split is 70-30 or 80-20, where the training set accounts for the majority of the data (70% or 80%), and the

testing set represents the remaining portion (30% or 20%). However, these ratios can vary

depending on the size of the dataset and the specific problem being addressed. In some cases, a three-way split involving a training set, validation set, and testing set might be employed,
particularly when hyperparameter tuning or model selection is involved.

The splitting process should be random to ensure that the resulting subsets are representative of the overall dataset. This helps to avoid any bias or skewed representation in either the training or testing sets. Randomness can be achieved through shuffling the dataset before splitting or by using randomized sampling techniques.

It's important to note that the testing set should only be used once for the final evaluation of the model. Iterative testing on the same testing set can lead to overfitting to the test set, which compromises the model's generalization ability. To address this, additional techniques like cross- validation can be employed to further assess model performance.

In Python, various machine learning libraries, such as scikit-learn, provide functions to split datasets into training and testing sets. For example, scikit-learn's train_test_split() function can be used to split data easily by specifying the desired ratio or the number of samples for each subset.

In summary, the train-test split is a fundamental step in machine learning model evaluation. It allows for estimating the model's performance on unseen data and helps in detecting issues like overfitting. By dividing the dataset into training and testing subsets, we can train the model on one portion and evaluate its performance on the other, enabling us to make informed decisions about the model's suitability and generalization ability.

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y, random_state=2)

print(X.shape,
    X_train.shape,
    X_test.shape)
    (768, 8) (614,
    8) (154, 8)
```

Training the Model

1. **Algorithem SVM Themory**

SVM, or Support Vector Machine, is a supervised machine learning algorithm commonly used for classification and regression tasks. It is particularly effective in cases where the data has clear separation boundaries.

The basic idea behind SVM is to find a hyperplane that best separates the data points of different classes. A hyperplane is a decision boundary that divides the feature space into separate regions, each representing a different class. The goal is to maximize the margin, which is the distance between the hyperplane and the nearest data points of each class. By maximizing the margin, SVM aims to achieve better generalization and robustness against noise. The SVM algorithm works as follows:

1. Data Preparation: Firstly, the input data is prepared by selecting relevant features and ensuring they are appropriately scaled or normalized.
2. Training Data: The algorithm takes a labeled training dataset with input features and corresponding target labels.
3. Feature Representation: Each data point is represented as a feature vector in an n-dimensional feature space, where n represents the number of features.
4. Hyperplane Selection: SVM searches for an optimal hyperplane that separates the data points of different classes. This hyperplane is selected to maximize the

margin, i.e., the distance between the hyperplane and the nearest data points of each class.

5. Margin Optimization: SVM aims to find the hyperplane that achieves the maximum margin while still correctly classifying the training data. It solves an optimization problem to find the optimal hyperplane parameters.
6. Kernel Trick: In cases where the data is not linearly separable, SVM can utilize the kernel trick. The kernel function transforms the data points into a higher-dimensional space, where they become separable by a hyperplane. Common kernel functions include linear,
polynomial, Gaussian (RBF), and sigmoid.
7. Classification: Once the optimal hyperplane is determined, SVM can classify new, unseen data points by assigning them to one of the classes based on which side of the hyperplane they fall.

SVM has several advantages, including its ability to handle high-dimensional data, its effectiveness in handling datasets with clear separation boundaries, and its versatility due to the kernel trick. It is also less affected by overfitting compared to other algorithms. However, SVM can be sensitive to the choice of hyperparameters and may not scale well with large datasets.

To implement SVM, various libraries and frameworks, such as scikit-learn in Python, provide SVM implementations with easy-to-use APIs. These libraries offer flexibility in selecting different kernel functions and parameter tuning to optimize performance.

In summary, SVM is a powerful algorithm for classification and regression tasks. It finds an optimal hyperplane to separate data points of different classes by maximizing the margin. It can handle both linearly separable and non-linearly separable data through the use of the kernel trick. SVM is widely used in various domains, including image recognition, text classification, and bioinformatics, due to its effectiveness and versatility.

```
#classifier =
svm.SVC(kernel
='linear')
classifier=
svm.SVC(kernel
='rbf')

#training the support
vector Machine
```

```
Classifier
classifier.fit(X_train
, Y_train)
```

Model Evaluation

Accuracy Score

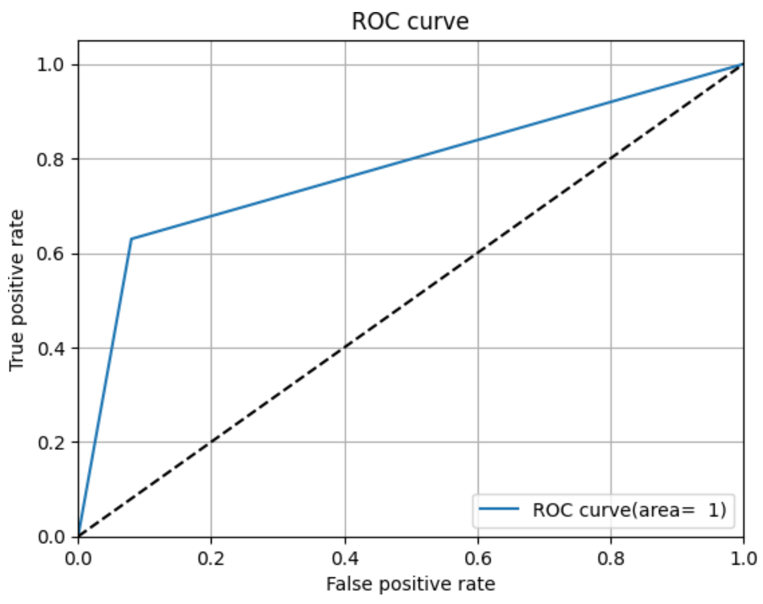
```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
print('Accuracy score of the training data : ', training_data_accuracy)

     Accuracy score of the training data :  0.8908794788273615
```

```
# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

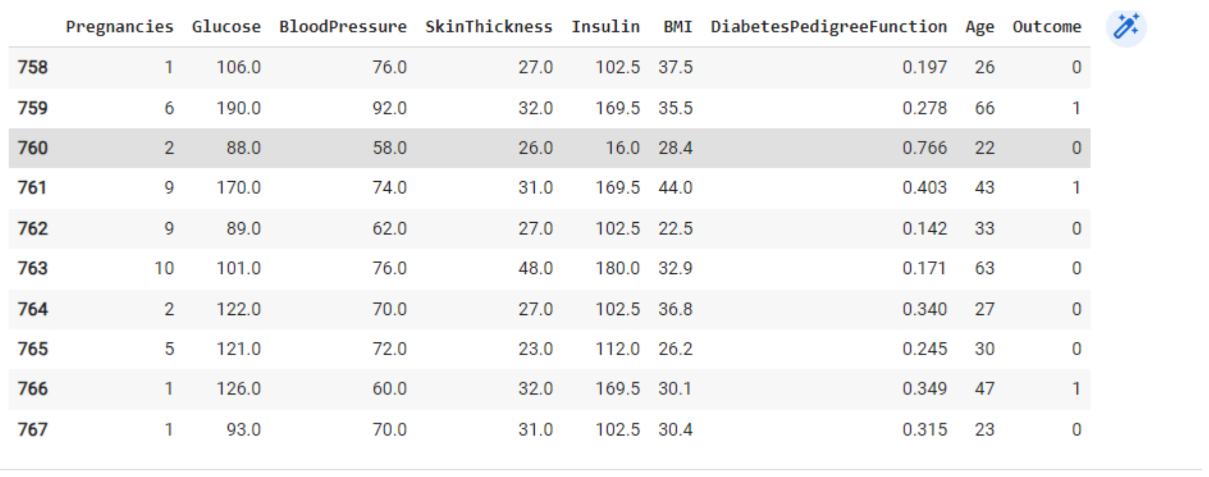
```
print('Accuracy score of the test data

     : ', test_data_accuracy)Accuracy

     score of the test data :

     0.8181818181818182
```

```
fpr,tpr,_=roc_curve(Y_test,X_test_prediction)
#calculate AUC UC (Area Under The Curve) ROC (Receiver
Operating Characteristics)roc_auc=auc(fpr,tpr)
print('AUC: %0.2f' % roc_auc)
#plot of ROC
curve for a
specified class
plt.figure()
plt.plot(fpr,tpr,label='ROC
curve(area= %2.f)' %roc_auc)
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False positive rate') plt.ylabel('True positive rate') plt.title('ROC curve')
plt.legend(loc='lower right') plt.grid()
plt.show()

     AUC: 0.77
```



Making a Predictive System

```
#hree we use data
show for head() and tail()
diabetes_dataset.tail(10)
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
758	1	106.0	76.0	27.0	102.5	37.5	0.197	26	0
759	6	190.0	92.0	32.0	169.5	35.5	0.278	66	1
760	2	88.0	58.0	26.0	16.0	28.4	0.766	22	0
761	9	170.0	74.0	31.0	169.5	44.0	0.403	43	1
762	9	89.0	62.0	27.0	102.5	22.5	0.142	33	0
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63	0
764	2	122.0	70.0	27.0	102.5	36.8	0.340	27	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0
766	1	126.0	60.0	32.0	169.5	30.1	0.349	47	1
767	1	93.0	70.0	31.0	102.5	30.4	0.315	23	0

2. Algorithm Random Forest

Random Forest is an ensemble learning algorithm that combines the power of multiple decision trees to make predictions. It is commonly used for both classification and regression tasks and is known for its accuracy and robustness.

The Random Forest algorithm works as follows:

1. Data Preparation: Firstly, the input data is prepared by selecting relevant features and ensuring they are appropriately scaled or normalized.
2. Training Data: The algorithm takes a labeled training dataset with input features and corresponding target labels.
3. Random Subsampling: Random Forest builds multiple decision trees by randomly selecting subsets of the training data. Each tree is

   trained on a different subset of the original dataset, which is called bootstrap aggregating or "bagging." This random subsampling helps introduce diversity in the forest.
4. Feature Randomness: In addition to subsampling the data, Random Forest also introduces randomness in feature selection. Instead of considering all features at each split, it randomly selects a subset of features for each tree. This further increases the diversity and
   reduces correlation among the trees.
5. Decision Tree Construction: Each decision tree in the Random Forest is constructed using a process called recursive partitioning. At each node, the tree algorithm selects the best feature and split point to maximize information gain or

Gini impurity. The process continues
recursively until a stopping condition is met, such as reaching a maximum tree depth or the minimum number of samples required for a

leaf node.

6. Ensemble Prediction: Once all the decision trees are constructed, the Random Forest combines their predictions to make the final

   prediction. For classification tasks, the most common prediction aggregation method is majority voting, where the class that receives the most votes from the trees is chosen. For regression tasks, the predictions from individual trees are averaged to obtain the final prediction.

Random Forest has several advantages, including its ability to handle high-dimensional data, capture complex relationships between features, and handle missing values. It is also less prone to overfitting compared to individual decision trees due to the randomness introduced during the ensemble construction.

Random Forest is an ensemble learning algorithm that combines multiple decision trees to make accurate predictions. It introduces randomness through random subsampling of the training data and feature selection. Random Forest is widely used in various domains, including classification, regression, and feature selection tasks, due to its robustness, accuracy, and ability to handle complex data.

```
#X_train, X_test, Y_train, Y_test
print(X.shape,

     X_train.shape,

     X_test.shape)

     (768, 8) (614,

     8) (154, 8)
```

```
RFclassifier=
RandomForestC
lassifier()
RFclassifier.
fit(X_train,Y
_train)
```

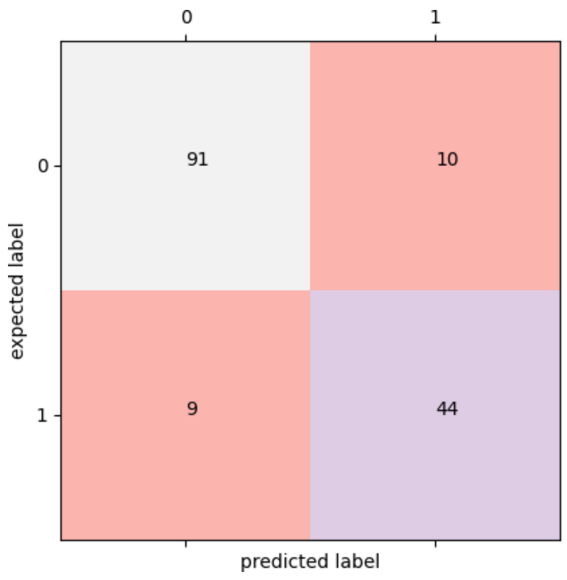
```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
Y_pred=RFclassifier.predict(X_test)
```

confusion_matrix1=confusion_matrix(Y_test,Y_pred)confusion_matrix1

output
array([[91, 9],
[10, 44]])

```
plt.show()
```



```
test_data_accuracy2=accuracy_score(Y_test,Y_pred)
print('Accuracy score of the test data :
        ', test_data_accuracy2)

   Accuracy score of the test data :
          0.8766233766233766
```

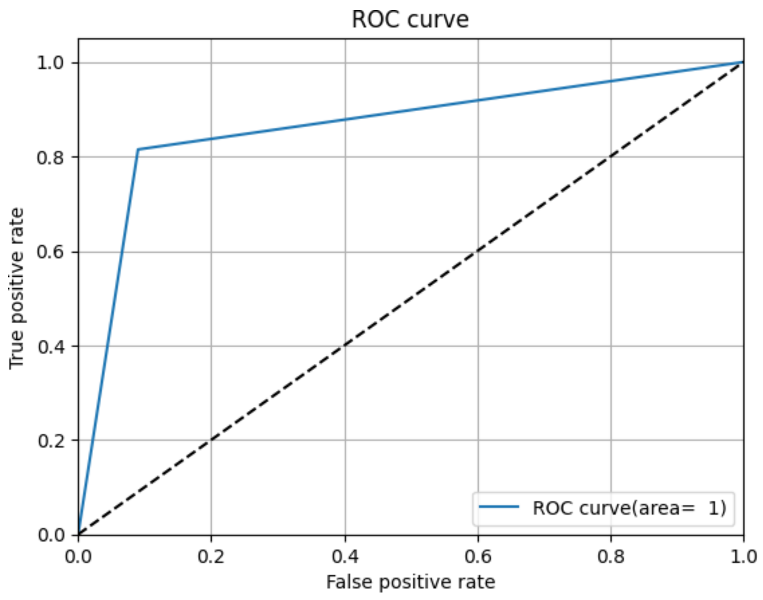
```
print(classification_report(Y_test,Y_pred))
```

```
                 precision    recall  f1-score   support

              0       0.90      0.91      0.91       100
              1       0.83      0.81      0.82        54

       Accuracy                           0.88       154
      macro avg       0.87      0.86      0.86       154
   weighted avg       0.88      0.88      0.88       154
```

```
fpr,tpr,_=roc_curve(Y_test,Y_pred)
#calculate AUC UC (Area Under The Curve) ROC (Receiver
Operating Characteristics)roc_auc=auc(fpr,tpr)
print('AUC: %0.2f' % roc_auc)
#plot of ROC
curve for a
specified class
```

```
plt.figure()
plt.plot(fpr,tpr,label='ROC
curve(area= %2.f)' %roc_auc)
plt.plot([0,1],[0,1],'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False positive rate')
```

```
     AUC: 0.86
```



Double-click (or enter) to edit

```
prediction =
RFclassifier.predic
t(std_data)if
(prediction[0] ==
0):
  print('The
person is not
diabetic')
else:
  print('The person is diabetic')
```

Result: The person is diabetic

CONCLUSION

The SVM and Random forest Model achieved 81 and 88% accuracy.

During this process we figured out few attributes that played an important role. Out of eight attributes Glucose, BMI, Pregnancies, Age and Insulin were the important ones. As per our results and data the other factors like Diabetes Pedigree Function, Skin Thickness and Blood Pressure had negligible effect in determining diabetes.

The rules derived will be helpful for doctors to identify patients suffering from diabetes. Further predicting the disease early leads to treating the patient before it becomes critical