Diabetes Prediction Using Machine Learning

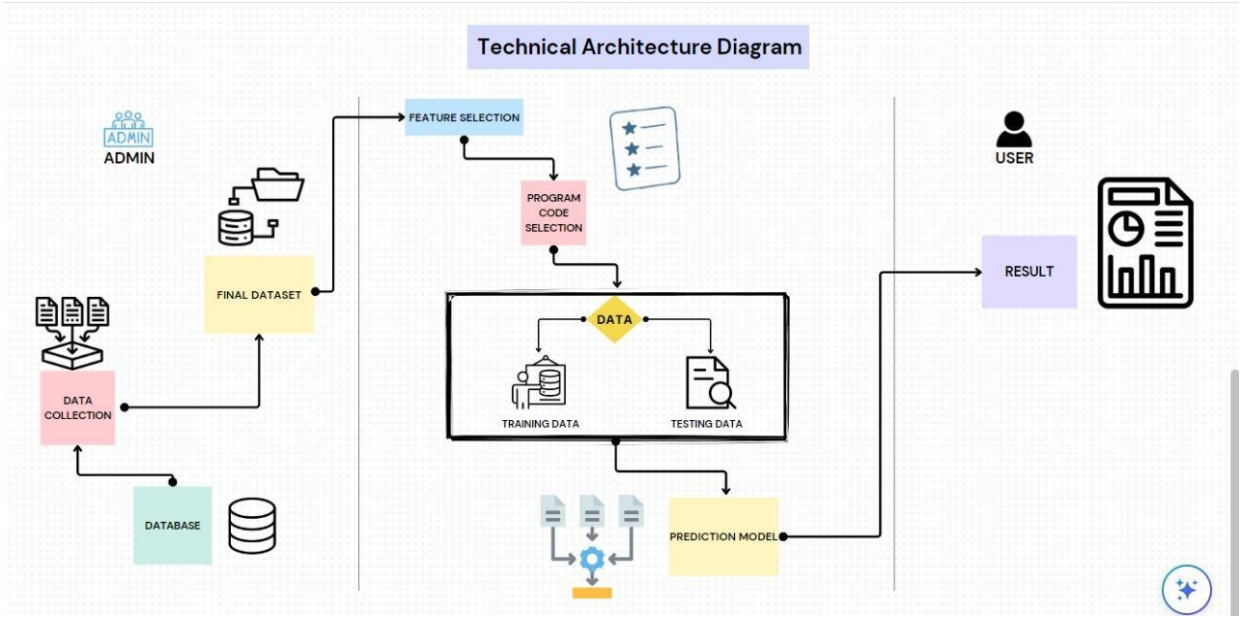
Diabetes Prediction Using Machine Learning

In this project, we aim to use machine learning algorithms to predict the onset of diabetes in individuals based on their health records and other relevant factors such as age, BMI, family history, and lifestyle habits. The dataset used in this project will include information on various clinical parameters such as blood pressure, BMI, Heart diseases and cholesterol levels.

Our goal is to develop a predictive model that can accurately identify individuals who are at high risk of developing diabetes, thereby allowing for early intervention and prevention of the disease. By using machine learning techniques to analyse large amounts of data, we can identify patterns and make accurate predictions that could potentially save lives.

Overall, this project has the potential to contribute to the field of healthcare by improving early detection and prevention of diabetes, ultimately leading to better health outcomes for individuals and communities.

Technical Architecture :



Project Flow:

- User figures out the symptoms and gives as the input to system.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

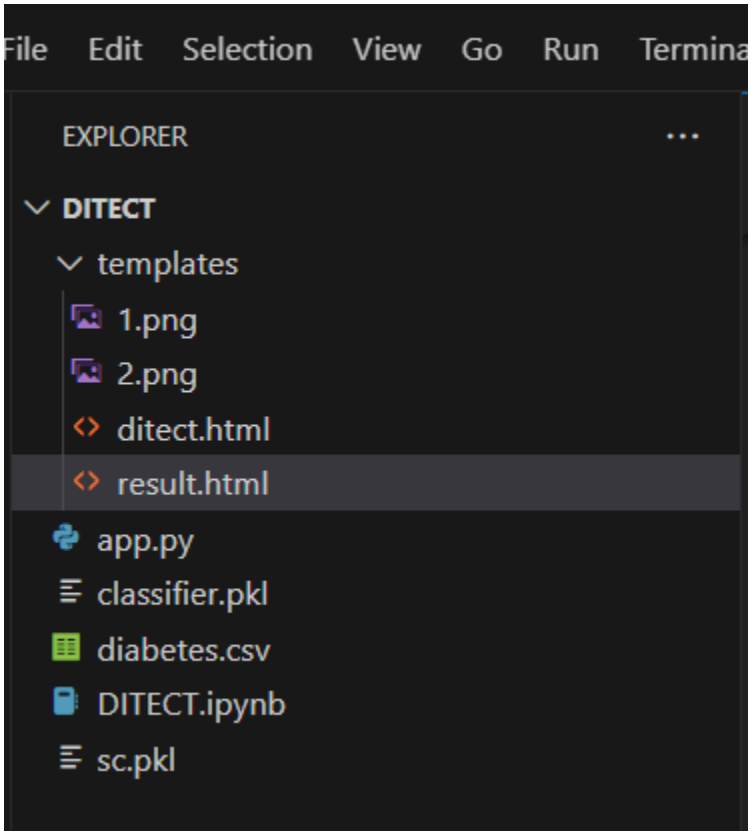
- Define Problem / Problem Understanding ○ Specify the Symptoms ○ Form requirements ○ Gives the information for analysis. ○ Social Impact
- Data Collection & Preparation ○ Collect the dataset ○ Data Preparation
- Exploratory Data Analysis ○ Descriptive statistical ○ Visual Analysis
- Model Building ○ Training the model in multiple algorithms ○ Testing the model
- Performance Testing ○ Testing model with multiple evaluation metrics
- Model Deployment ○ Save the best model ○ Integrate with Web Framework

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- ML Concepts
  Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- Decision tree:
https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
- SVM : https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
- Logistic Regression : https://www.javatpoint.com/logistic-regression-in-machine-learning
- Naive Bayes : https://www.javatpoint.com/machine-learning-naive-bayes-classifier
- Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-tounderstand-the-math-behind-xgboost/
- Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- NLP:-https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_python.htm
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:



Create the Project folder which contains files as shown below

- We are building a web application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Training folder contains a model training file.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Symptoms

Specifying symptoms in the prediction of diabetes using machine learning involves the identification and characterization of key indicators or features that may be associated with the development or presence of diabetes. This process is a crucial step in building predictive models that can assist in early diagnosis or risk assessment for individuals at risk of diabetes.Gather relevant data on individuals, including both those with and without diabetes. This dataset should ideally include a variety of features such as age, gender, family history, lifestyle factors, and most importantly, symptoms related to diabetes.Identify and select features that are likely to be indicative of diabetes. This could include symptoms such as increased thirst (polydipsia), frequent urination (polyuria), unexplained weight loss, fatigue, blurred vision, slow wound healing, and others.

Activity 2: Form requirements

The activity of form filling for predicting or assessing health-related conditions, such as diabetes, typically involves collecting relevant information from individuals. This information is crucial for building predictive models or conducting assessments that can help in understanding health risks or making informed decisions.

Form design : [‘,Design a form that includes fields for essential information related to the health condition of interest. For predicting diabetes, common fields might include:

- Age: The age of the individual.
- BMI (Body Mass Index): A measure of body fat based on height and weight.
- Cholesterol: Levels of cholesterol in the blood.
- Insulin: Levels of insulin, which is important for glucose metabolism.

User Input:

- Individuals fill out the form, providing accurate and honest information. The form may be presented in various formats, such as paper-based forms, online forms, or through dedicated applications.

Data Validation:

- Validate the entered data to ensure it falls within acceptable ranges or conforms to specific criteria. This step helps maintain data quality and accuracy.

Data Storage:

- Store the collected data securely. Depending on the scale and context, this could involve databases, electronic health records, or other secure data storage methods.

Privacy Considerations:

- Ensure that the process complies with privacy regulations and ethical standards. Health-related data is sensitive, and protecting individuals' privacy is of utmost importance. Implement appropriate security measures and obtain informed consent if necessary.

Feature Engineering:

- Once the data is collected, preprocess it for analysis. This may involve transforming or engineering features to make them suitable for input into machine learning algorithms. For example, converting categorical variables into a numerical format.

Analysis or Prediction:

- Use the collected data as input for a predictive model or analysis. For diabetes prediction, this could involve machine learning algorithms that consider age, BMI, cholesterol, insulin levels, and potentially other features.

Results Interpretation:

- Interpret the results of the analysis.

Activity 3: Giving the Information for analysis

Providing information for analysis in the context of diabetes prediction involves submitting relevant data to a system or healthcare professional for further examination. This process is crucial for leveraging data-driven approaches, such as machine learning, to identify patterns and make predictions about the likelihood of diabetes. Here's a step-by-step description of this activity:

Data Collection:

- Collect information based on a structured form, as mentioned earlier, including details like age, BMI, cholesterol levels, insulin levels, and other pertinent health indicators.

Data Entry:

- Enter the collected information into a digital system or a specific analysis platform. This may involve using software applications, electronic health record systems, or online forms.

Data Validation:

- Verify the accuracy and completeness of the entered data. Ensure that there are no errors or missing values that could compromise the analysis.

Submission of Data:

- Transmit the entered data to the analysis system. This could be a centralized database, a machine learning model, or a healthcare professional responsible for conducting the analysis.

Data Processing:

- If using machine learning or statistical analysis, the system processes the submitted data to identify patterns, correlations, and potential risk factors associated with diabetes. This step may involve complex algorithms that analyze the relationships between different variables.

Analysis and Prediction:

- Based on the processed data, the system generates predictions or assessments related to the likelihood of diabetes. This could include a risk score, probability estimate, or a binary classification (e.g., diabetic or non-diabetic).

Results Presentation:

- The analysis results are presented in a comprehensible format. This could be a detailed report, graphical representations, or a simple notification indicating the predicted risk level.

Interpretation:

- Healthcare professionals or individuals receiving the analysis interpret the results in the context of diabetes risk. They consider the identified factors and the associated risk level to make informed decisions or recommendations.

Feedback and Action:

- If necessary, healthcare professionals may provide feedback or recommendations based on the analysis results. This could involve lifestyle modifications, further diagnostic tests, or additional medical interventions.

Privacy and Security:

- Throughout the entire process, it is crucial to prioritize the privacy and security of the collected health data. Compliance with relevant data protection regulations and ethical considerations is essential.

By providing accurate and relevant information for analysis, individuals and healthcare providers can leverage the power of data to make more informed decisions regarding diabetes risk, prevention, and management.

Activity 4: Social or Business Impact

Integration of predictive analytics in healthcare can lead to innovative solutions for early diagnosis and personalized treatment plans. Businesses involved in healthcare technology and analytics can develop and offer advanced tools and platforms for diabetes risk assessment.Providing individuals with information about their diabetes risk empowers them to make informed decisions about their health. This knowledge can motivate lifestyle changes and promote healthier behaviors.Early prediction of diabetes allows for proactive measures, emphasizing prevention and lifestyle modifications. This can contribute to reducing the overall prevalence of diabetes and related complications.The data collected for predictive analysis can contribute to research on diabetes epidemiology and risk factors. This can lead to a better understanding of the disease and inform public health policies.

Milestone 2: Data Collection & Preparation

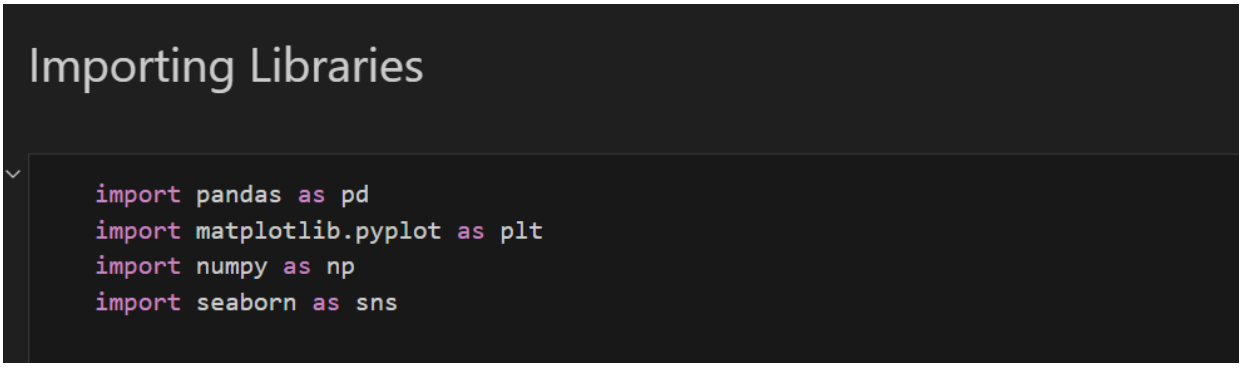
ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.
Link: Diabetes Health Indicators Dataset | Kaggle
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.
Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries



Here we used Python's pandas library is used for data manipulation, matplotlib aids in creating visualizations, numpy facilitates numerical computations, and seaborn simplifies statistical data visualization

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

We have named the dataset as ‘diabetes.csv’ and extracted the data based on this.

Data Collection

```
#Extracting data
dataset=pd.read_csv('diabetes.csv')
dataset.head()
```

[2]

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	2	138	62	35	0	33.6	0.127	47	1
1	0	84	82	31	125	38.2	0.233	23	0
2	0	145	0	0	0	44.2	0.630	31	1
3	0	135	68	42	250	42.3	0.365	24	1
4	1	139	62	41	480	40.7	0.536	21	0

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

- Let’s know the info and describe of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used

```
[3]
#Our dataset dimesnions
dataset.shape

(2000, 9)
```

```
[55]
dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               2000 non-null   int64
 1   Glucose                   2000 non-null   int64
 2   BloodPressure             2000 non-null   int64
 3   SkinThickness             2000 non-null   int64
 4   Insulin                   2000 non-null   int64
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64
 8   Outcome                   2000 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 140.8 KB
```

- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
[57]
null_values = dataset.isnull()

print(null_values)
```

```
...      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  \
0          False    False          False          False    False  False
1          False    False          False          False    False  False
2          False    False          False          False    False  False
3          False    False          False          False    False  False
4          False    False          False          False    False  False
...          ...      ...            ...            ...      ...    ...
1995       False    False          False          False    False  False
1996       False    False          False          False    False  False
1997       False    False          False          False    False  False
1998       False    False          False          False    False  False
1999       False    False          False          False    False  False

      DiabetesPedigreeFunction    Age  Outcome
0                        False  False    False
1                        False  False    False
2                        False  False    False
3                        False  False    False
4                        False  False    False
...                        ...    ...      ...
1995                     False  False    False
1996                     False  False    False
1997                     False  False    False
1998                     False  False    False
1999                     False  False    False

[2000 rows x 9 columns]
```

```
[58]
null_sum = dataset.isnull().sum()

print(null_sum)
```

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

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data, we must convert the categorical data to integer encoding or binary encoding.
To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using Label encoding with the help of list comprehension.

- In our project, categorical features are in many columns Label encoding is done

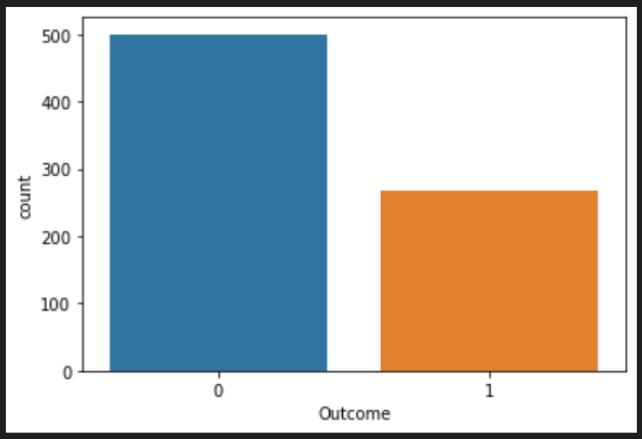
Activity 2.3: Handling Imbalance Data

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of some columns feature with some mathematical formula

- From the below diagram, we could visualize that some of the feature has outliers Boxplot from seaborn library is used here.

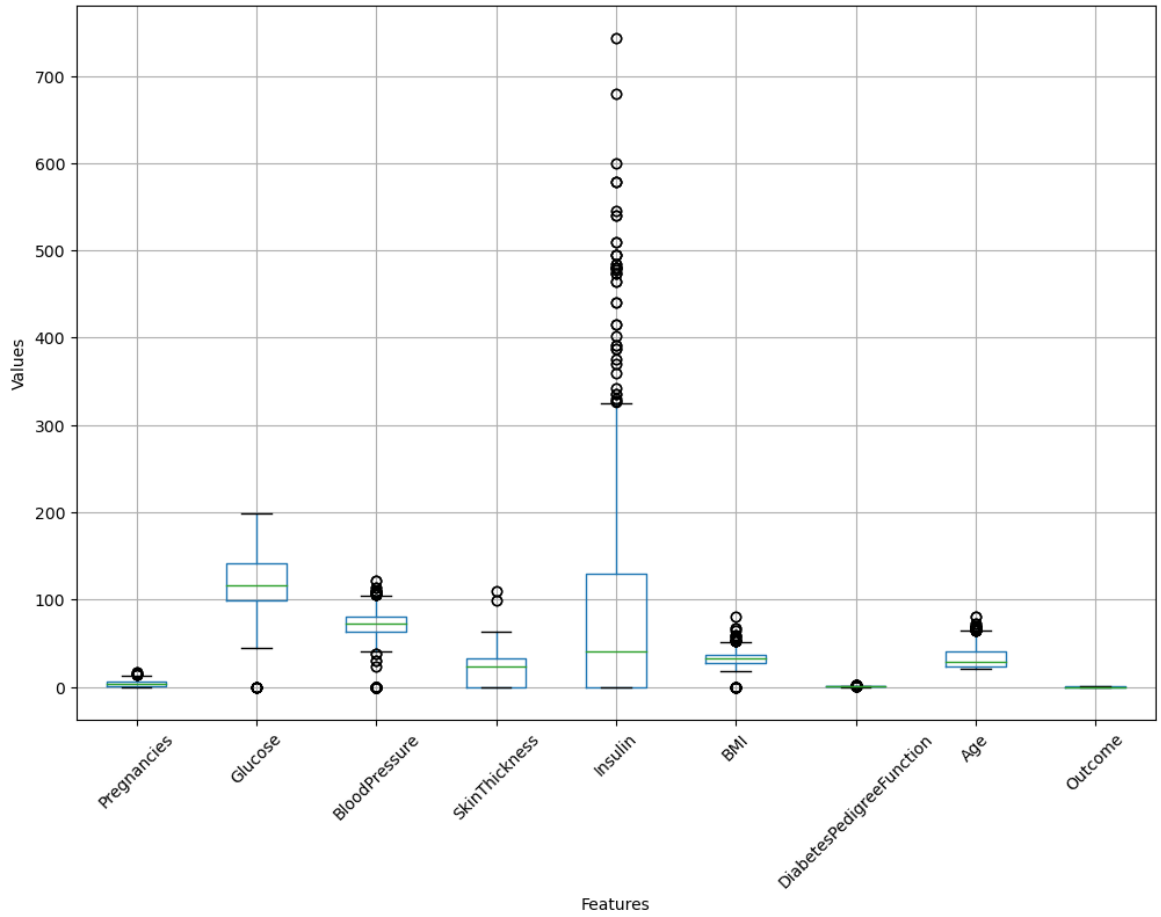
```
[5]
#Countng values of outcomes having 0 or 1, 0 means non diabetic and 1 means diabetic
sns.countplot(x='Outcome',data=dataset)
```

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




```
[60]
columns_to_plot = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
plt.figure(figsize=(12, 8))
dataset[columns_to_plot].boxplot()
box_color = 'skyblue'
plt.ylabel('Values')
plt.xlabel('Features')
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
```

Python


700
600
500
400
300
200
100
0
Values
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
Features

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
[56]  dataset.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	3.703500	121.182500	69.145500	20.935000	80.254000	32.193000	0.470930	33.090500	0.342000
std	3.306063	32.068636	19.188315	16.103243	111.180534	8.149901	0.323553	11.786423	0.474498
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.375000	0.244000	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	40.000000	32.300000	0.376000	29.000000	0.000000
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.800000	0.624000	40.000000	1.000000
max	17.000000	199.000000	122.000000	110.000000	744.000000	80.600000	2.420000	81.000000	1.000000

```
[61]  dataset['Outcome'].value_counts()
```

```
0    1316
1     684
Name: Outcome, dtype: int64
```

```
[62]  dataset.groupby('Outcome').mean()
```

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.168693	110.586626	68.094985	20.052432	70.563830	30.567477	0.434676	31.081307
1	4.732456	141.568713	71.166667	22.633041	98.897661	35.320468	0.540681	36.956140

```
[64]  np.std(dataset)
```

```
Pregnancies                 3.305236
Glucose                    32.060617
BloodPressure              19.183517
SkinThickness              16.099217
Insulin                   111.152735
BMI                         8.147863
DiabetesPedigreeFunction    0.323472
Age                        11.783476
Outcome                     0.474380
dtype: float64
```

```
[65]  np.min(dataset)
```

```
C:\Users\nsraj\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py:84: FutureWarning
  return reduction(axis=axis, out=out, **passkwargs)
```

```
Pregnancies                 0.000
Glucose                     0.000
BloodPressure               0.000
SkinThickness               0.000
Insulin                     0.000
BMI                         0.000
DiabetesPedigreeFunction    0.078
Age                        21.000
Outcome                     0.000
dtype: float64
```

```
[66]  np.max(dataset)

...   C:\Users\nsraj\anaconda3\Lib\site-packages\numpy\core\fromnum
        return reduction(axis=axis, out=out, **passkwargs)

...   Pregnancies                 17.00
      Glucose                    199.00
      BloodPressure              122.00
      SkinThickness              110.00
      Insulin                    744.00
      BMI                         80.60
      DiabetesPedigreeFunction     2.42
      Age                         81.00
      Outcome                      1.00
      dtype: float64

[67]  np.percentile(dataset, 75)

...   62.0
```

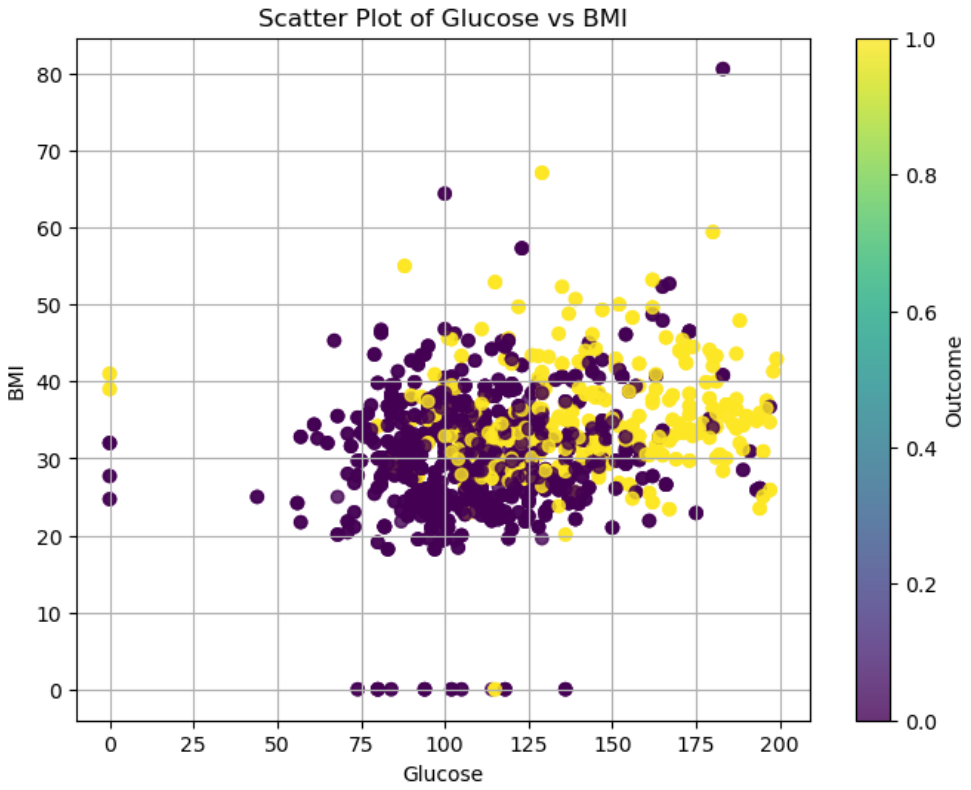
Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and count plot.
In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features.

```
[68]
plt.figure(figsize=(8, 6))
plt.scatter(dataset['Glucose'], dataset['BMI'], c=dataset['Outcome'], cmap='viridis', alpha=0.8)
plt.colorbar(label='Outcome')
plt.title('Scatter Plot of Glucose vs BMI')
plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.grid(True)
plt.show()
```

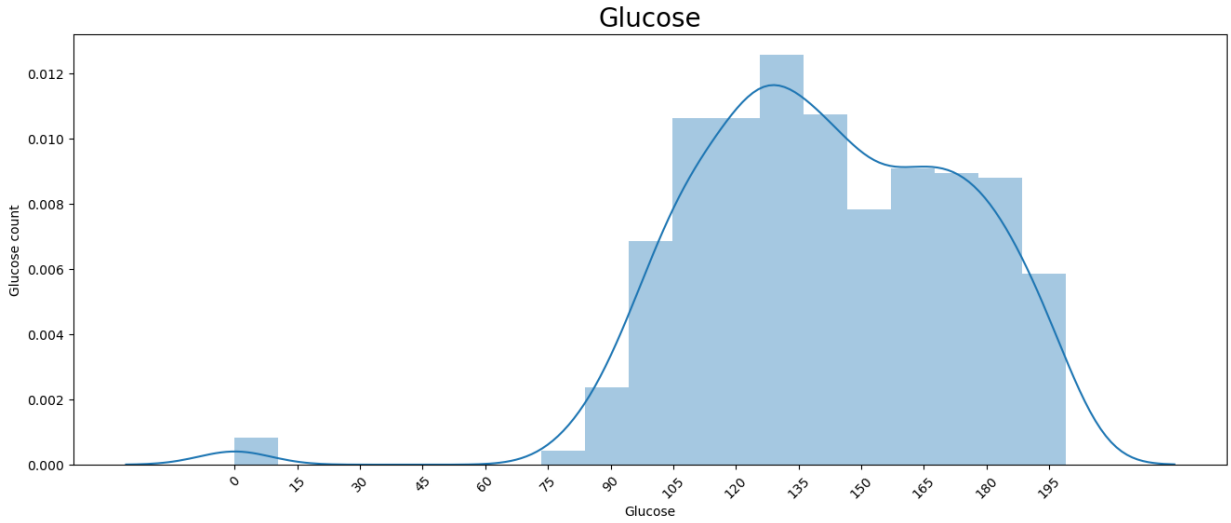



```
[45]
cols = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

# Creating a pairplot
sns.pairplot(dataset[cols], hue='Outcome', palette='viridis')
plt.suptitle('Pairplot of Diabetes Dataset')
plt.show()
```

Activity 2.2: Bivariate analysis

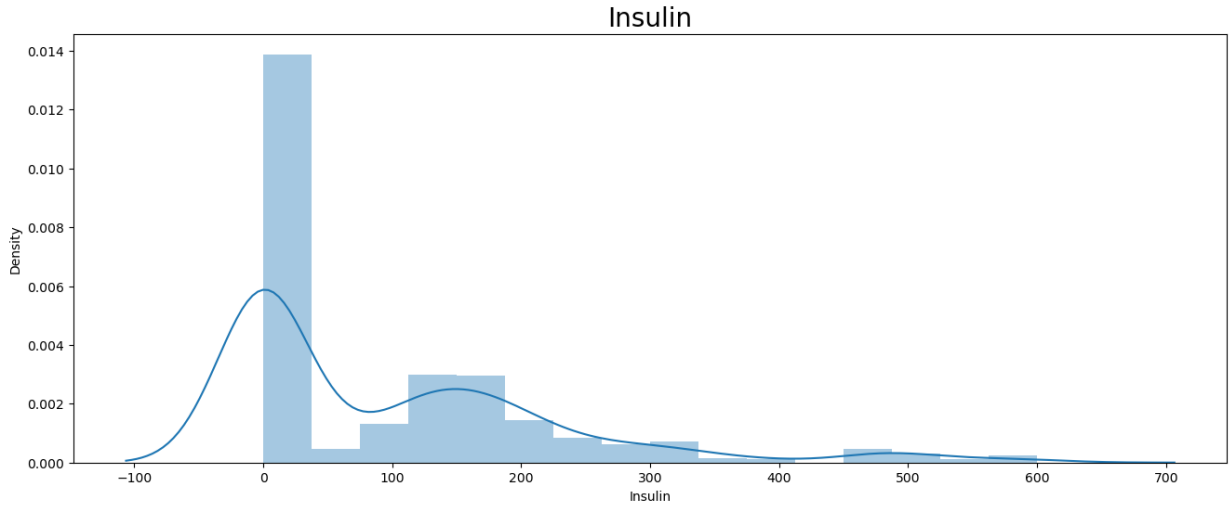
To find the relation between two features we use bivariate analysis. Here we are visualizing. Countplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
[15]
#glucose for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(dataset["Glucose"][dataset["Outcome"] == 1])
plt.xticks([i for i in range(0,201,15)],rotation = 45)
plt.ylabel("Glucose count")
plt.title("Glucose",fontsize = 20)
```

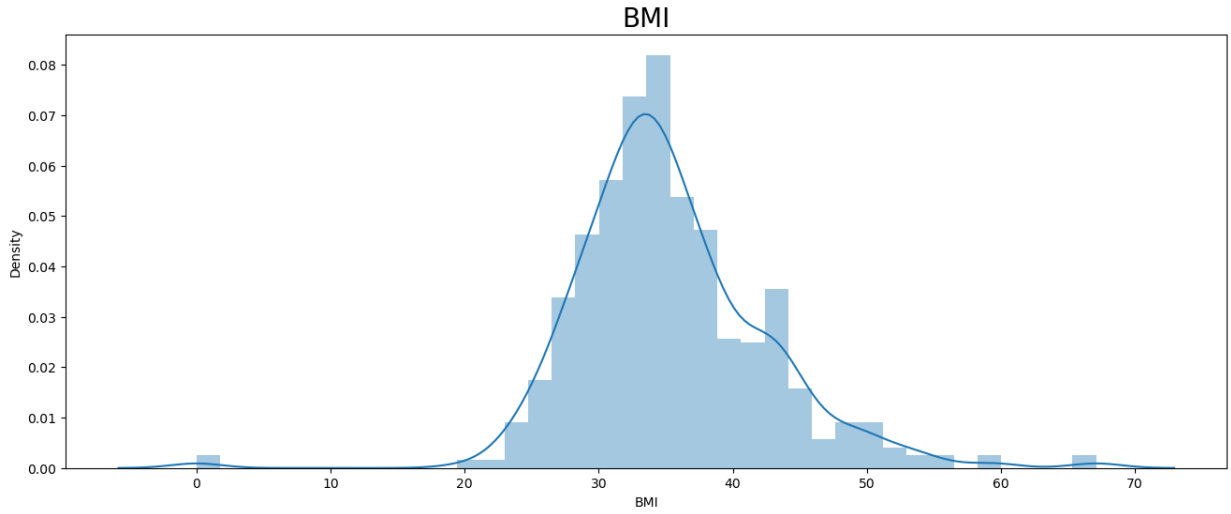



```
#insulin for diabetic

fig = plt.figure(figsize = (16,6))

sns.distplot(dataset["Insulin"][dataset["Outcome"]==1])
plt.xticks()
plt.title("Insulin",fontsize = 20)
```

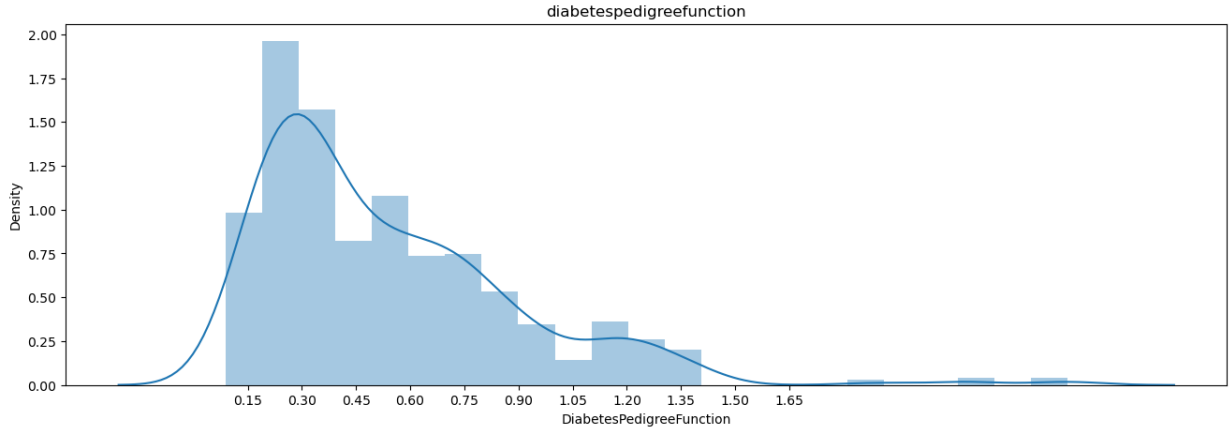
[16]



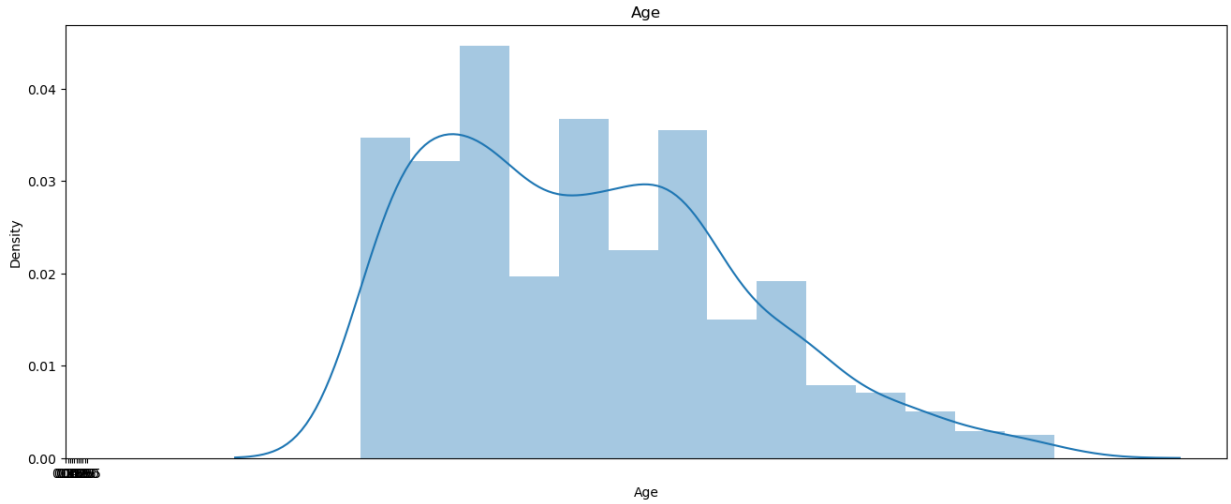
```
[17]
#BMI for diabetic
fig = plt.figure(figsize =(16,6))

sns.distplot(dataset["BMI"][dataset["Outcome"]==1])
plt.xticks()
plt.title("BMI",fontsize = 20)
```




```
[18]
#diabeticpedigreefunction for diabetic
fig = plt.figure(figsize = (16,5))
sns.distplot(dataset["DiabetesPedigreeFunction"][dataset["Outcome"] == 1])
plt.xticks([i*0.15 for i in range(1,12)])
plt.title("diabetespedigreefunction")
```



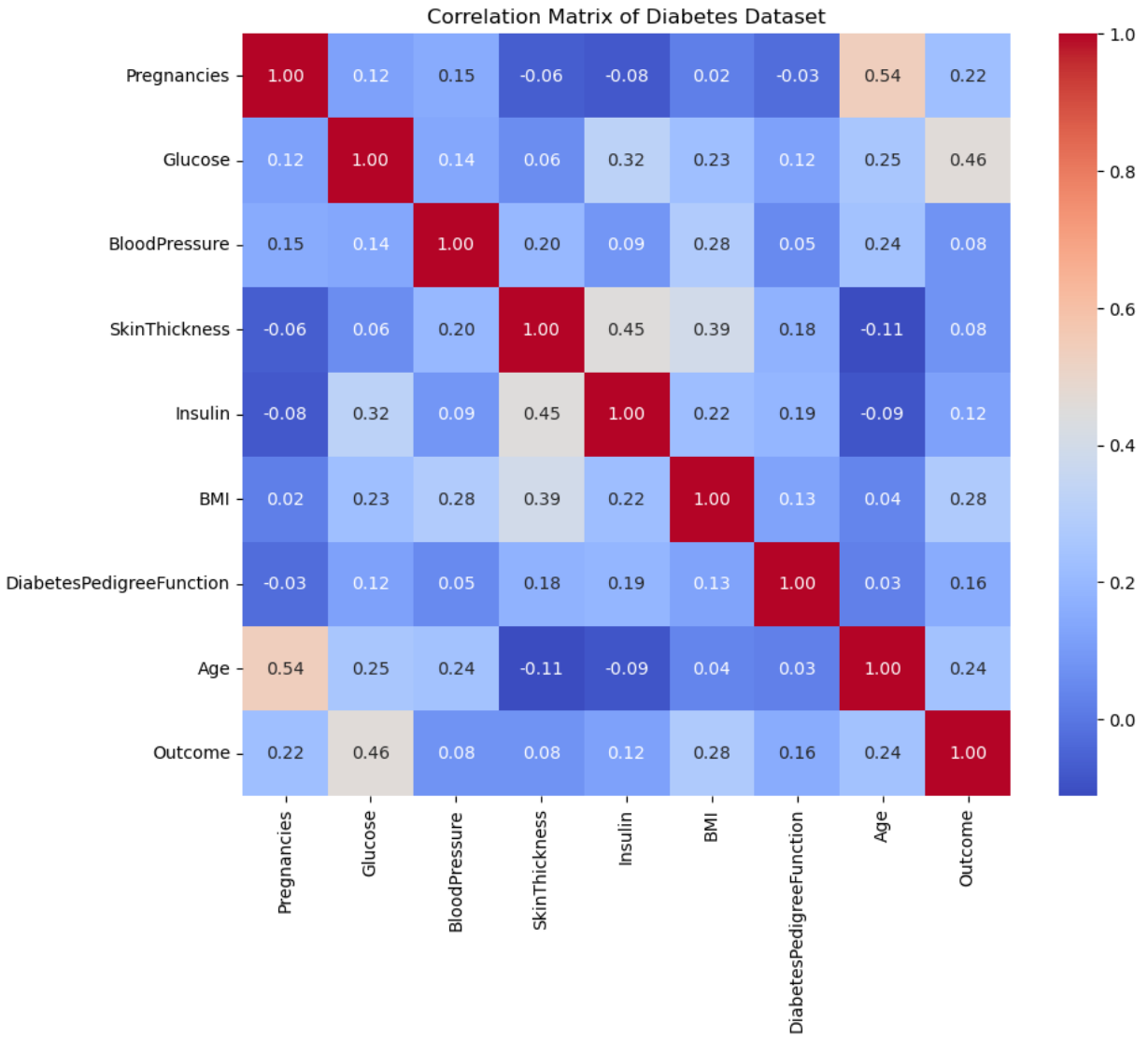
```
#Age for diabetic
fig = plt.figure(figsize = (16,6))

sns.distplot(dataset["Age"][dataset["Outcome"] == 1])
plt.xticks([i*0.15 for i in range(1,12)])
plt.title("Age")
```
[19]



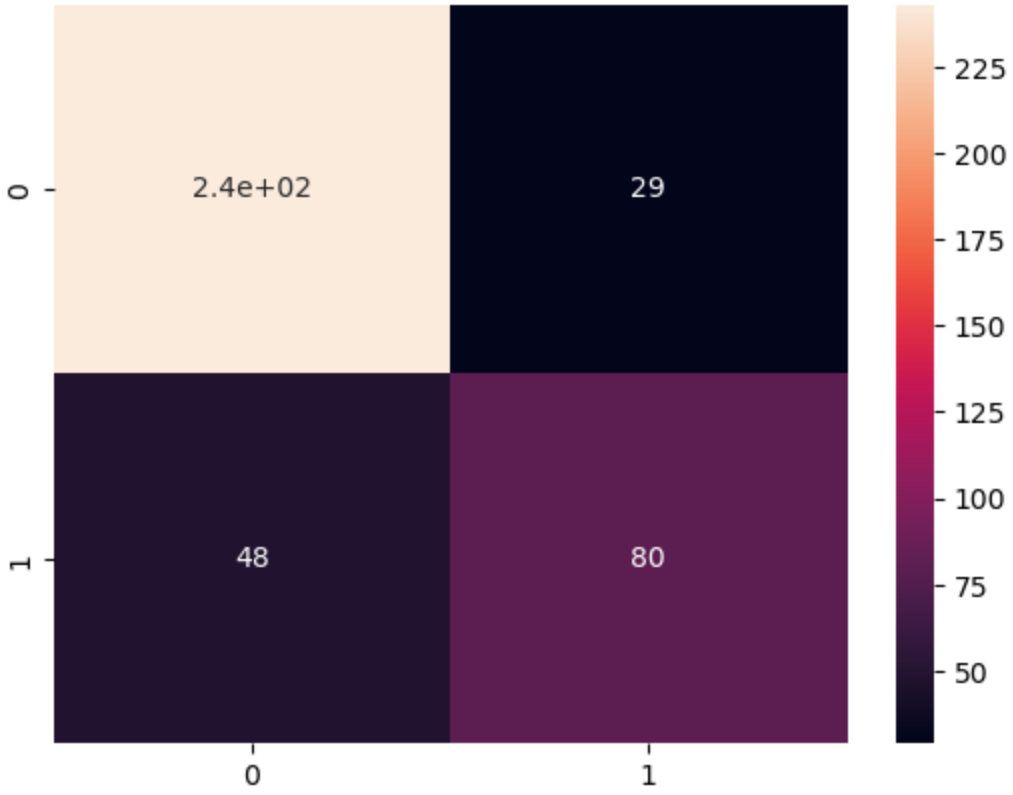
Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.

```
correlation_matrix = dataset.corr()

# Plotting the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Diabetes Dataset')
plt.show()
```
[23]




```
[29]
# Confusion matrix - To check how many are correct or wrong
from sklearn.metrics import confusion_matrix
knn_cm = confusion_matrix(y_test, knn_y_pred)
sns.heatmap(knn_cm, annot=True)
```

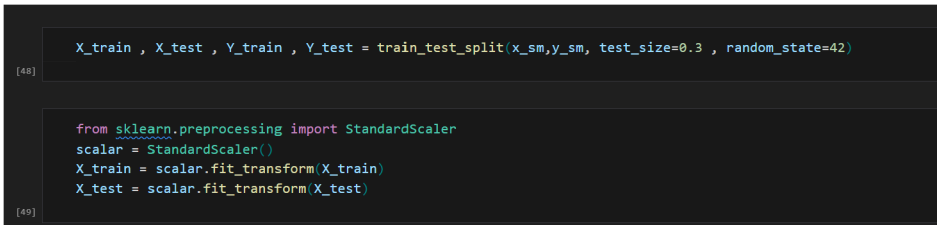


Applying PCA in a Machine Learning Pipeline for Diabetes Prediction:

Applying PCA (Principal Component Analysis) in a pipeline that includes hyperparameter tuning using GridSearchCV, data preprocessing using Standard Scaler, and applying a classifier can improve the performance of a machine learning model for cost prediction by reducing the dimensionality of the data, optimizing the hyperparameters of the classifier, and improving the accuracy of the predictions. StandardScaler scales the data, while PCA reduces dimensionality by identifying the most important features in the data. GridSearchCV helps to optimize the hyperparameters of the classifier, and finally, a suitable classifier is applied to the preprocessed and dimensionality-reduced data to evaluate the performance using appropriate metrics.

Splitting data into train and test:

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set
Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project, we are applying three classification algorithms. The best model is saved based on its performance.

Activity 1.1: Random Forest Regressor

A function named random forest regressor is created and train and test data are passed as the parameters. Inside the function, random forest regressor algorithm is initialized and training data is passed to the model with the fit() function. Test data is predicted with predict () function and saved in a new variable. For evaluating the model with R2_score.

```
rfc_train = rfc.predict(X_train)
from sklearn import metrics

print("Accuracy_Score =", format(metrics.accuracy_score(y_train, rfc_train)))
```

```
Accuracy_Score = 1.0
```

```
[33] from sklearn import metrics

predictions = rfc.predict(X_test)
print("Accuracy_Score =", format(metrics.accuracy_score(y_test, predictions)))
```

```
Accuracy_Score = 0.7716535433070866
```

Activity 1.2: Decision Tree Regressor

A function named decision Tree regressor is created and train and test data are passed as the parameters. Inside the function, decision Tree regressor algorithm is initialized and training data is passed to the model with fit() function. Test data is predicted with predict () function and saved in a new variable. For evaluating the model, For evaluating the model with R2_score

```
[34] from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
from sklearn import metrics

predictions = dtree.predict(X_test)
print("Accuracy Score =", format(metrics.accuracy_score(y_test,predictions)))
```

```
Accuracy Score = 0.6968503937007874
```

```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test,predictions))
```

```
[[126  36]
 [ 41  51]]
              precision    recall  f1-score   support

           0       0.75      0.78      0.77       162
           1       0.59      0.55      0.57        92

    accuracy                           0.70       254
   macro avg       0.67      0.67      0.67       254
weighted avg       0.69      0.70      0.69       254
```

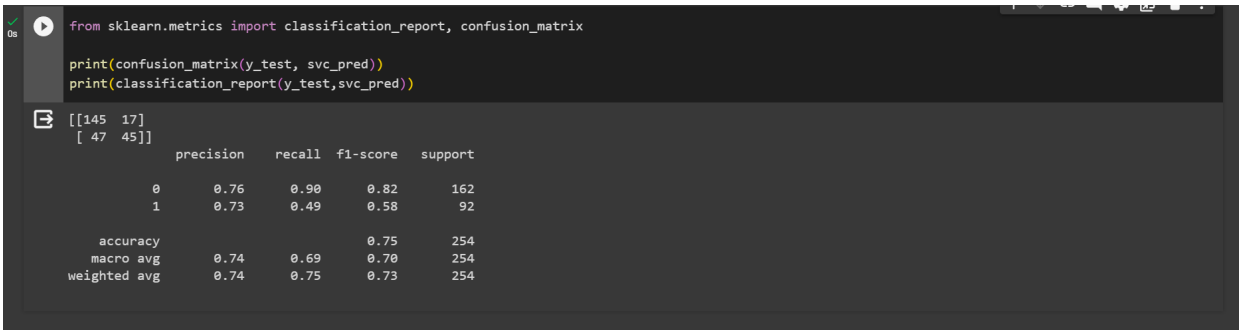
Activity 1.3: Logistic Regressor

To evaluate the performance of a logistic regression model, we need to test it on a separate dataset that it has not seen during training. This separate dataset is called the test data. We typically split the available data into two parts, the training data and the test data. The model is trained on the training data, and then tested on the test data to see how well it generalizes to new, unseen data.

```
[43] from sklearn import metrics

     xgb_pred = xgb_model.predict(X_test)
     print("Accuracy Score =", format(metrics.accuracy_score(y_test, xgb_pred)))

     Accuracy Score = 0.7283464566929134
```

Activity 2: Testing the model

Here we have tested with Logistic regression and SVM algorithms. With the help of predict () function.



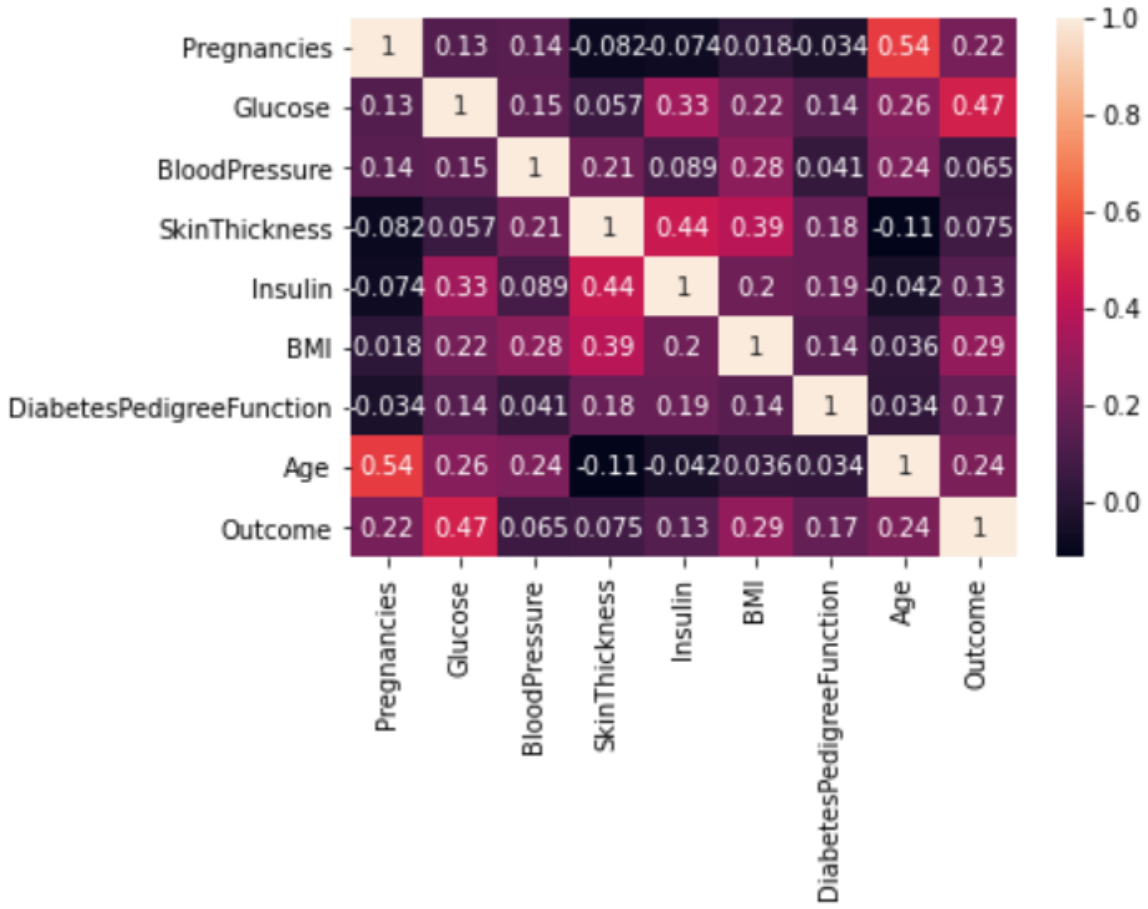
Milestone 5: Performance Testing

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the

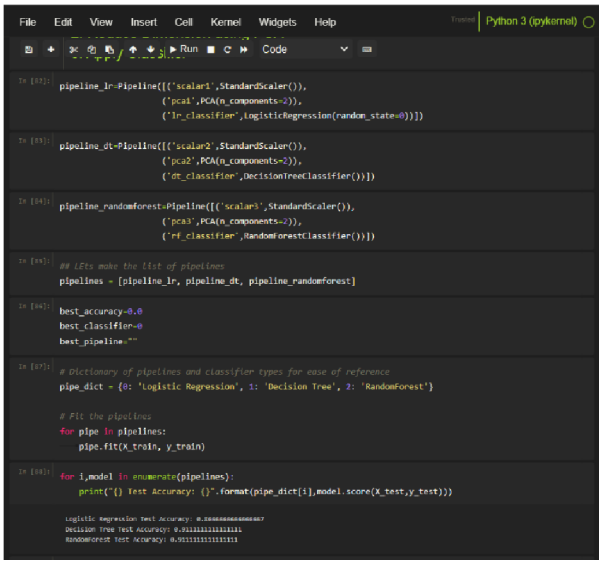
model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

```
plt.figure(figsize=(12,10))
# seaborn has an easy method to showcase heatmap
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')
```

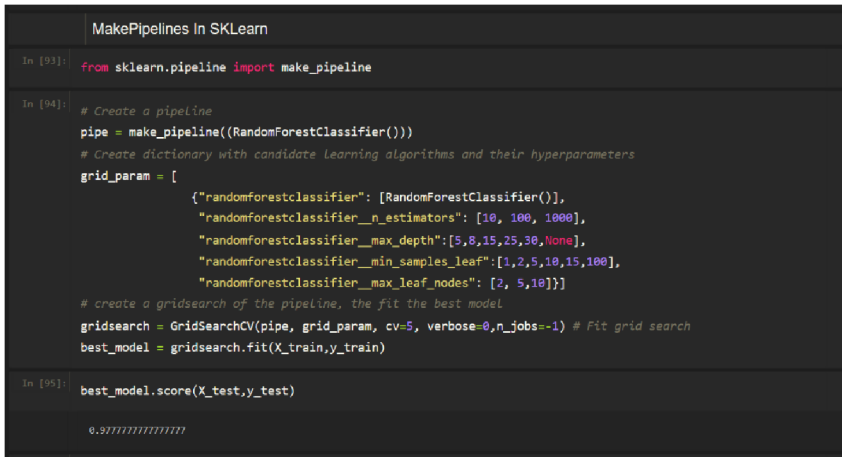


Activity 1.1: Compare the model

For comparing the below two models, with their R_2 score on training and testing data. the results of models are displayed as output. From the below three models random forest regressor is performing well



Activity 2: Comparing model accuracy before & after applying hyperparameter tuning
After seeing, the results of models are displayed as output. From the three models the random forest regressor model is performing well & Hyperparameter tuning For this model (it is not required)



Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project create two HTML files namely
index.html
prediction.html
result.html

and save them in the templates folder. Refer this
https://drive.google.com/drive/folders/13qqfrzmruVDk7J3mDedSuZ009qlokm7e?usp=sharing
for templates, static and python file

Activity 2.2: Build Python code:

Import the libraries in python file

```python
import numpy as np
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)
sc = pickle.load(open('sc.pkl', 'rb'))
model = pickle.load(open('classifier.pkl', 'rb'))

```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

Render HTML page:

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

```python
app = Flask(__name__)
sc = pickle.load(open('sc.pkl', 'rb'))
model = pickle.load(open('classifier.pkl', 'rb'))
```

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```python
@app.route('/')
def home():
    return render_template('ditect.html')

```

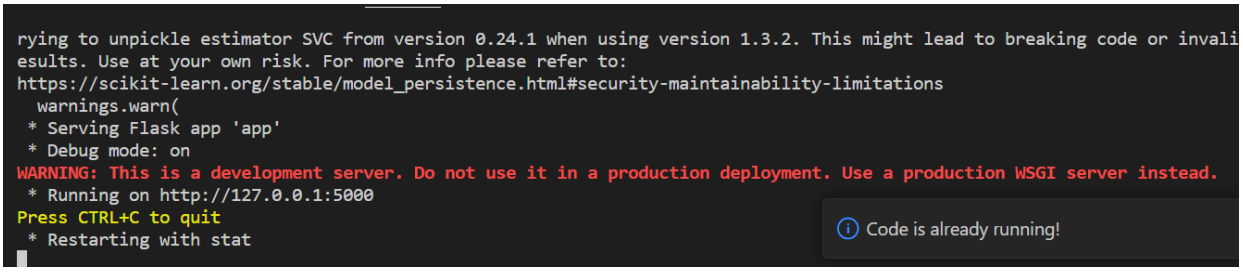
Here we are routing our app to prediction () function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model Predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
@app.route('/')
def home():
    return render_template('ditect.html')

@app.route('/predict',methods=['POST'])
def predict():

    float_features = [float(x) for x in request.form.values()]
    final_features = [np.array(float_features)]
    pred = model.predict( sc.transform(final_features) )
    return render_template('resultpage.html', prediction = pred)

if __name__ == "__main__":
    app.run(debug=True)
```

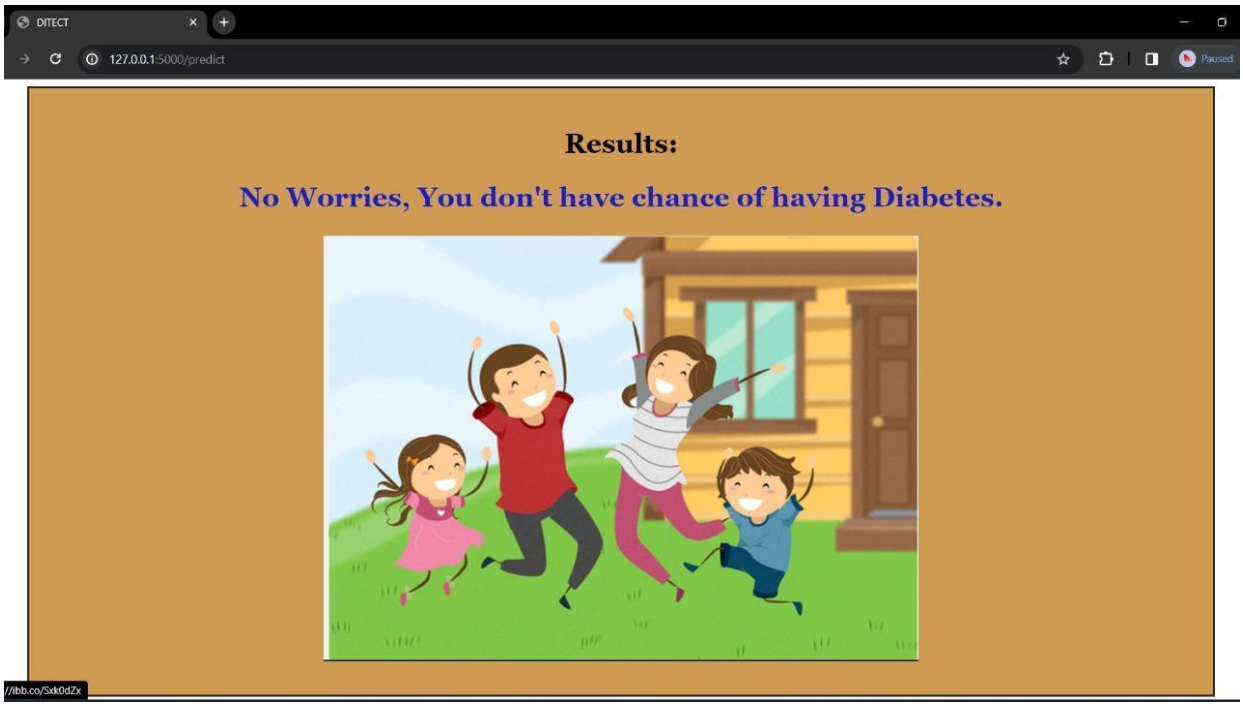
Activity 2.3: Run the web application

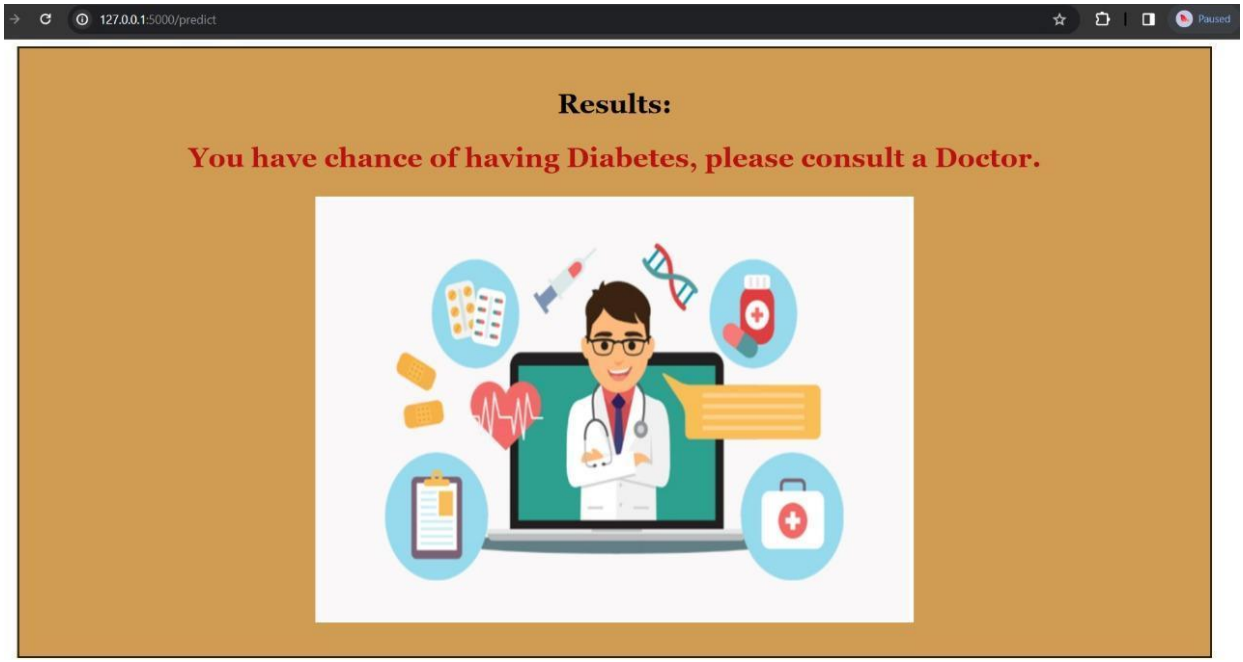
- Open anaconda prompt from the start menu ● Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



Now, Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result


Diabetes Prediction
Glucose Level
Insulin
BMI
Diabetes PF
Age
Predict


DITECT
127.0.0.1:5000/predict
Paused
Results:
No Worries, You don't have chance of having Diabetes.


127.0.0.1:5000/predict
Paused
Results:
You have chance of having Diabetes, please consult a Doctor.