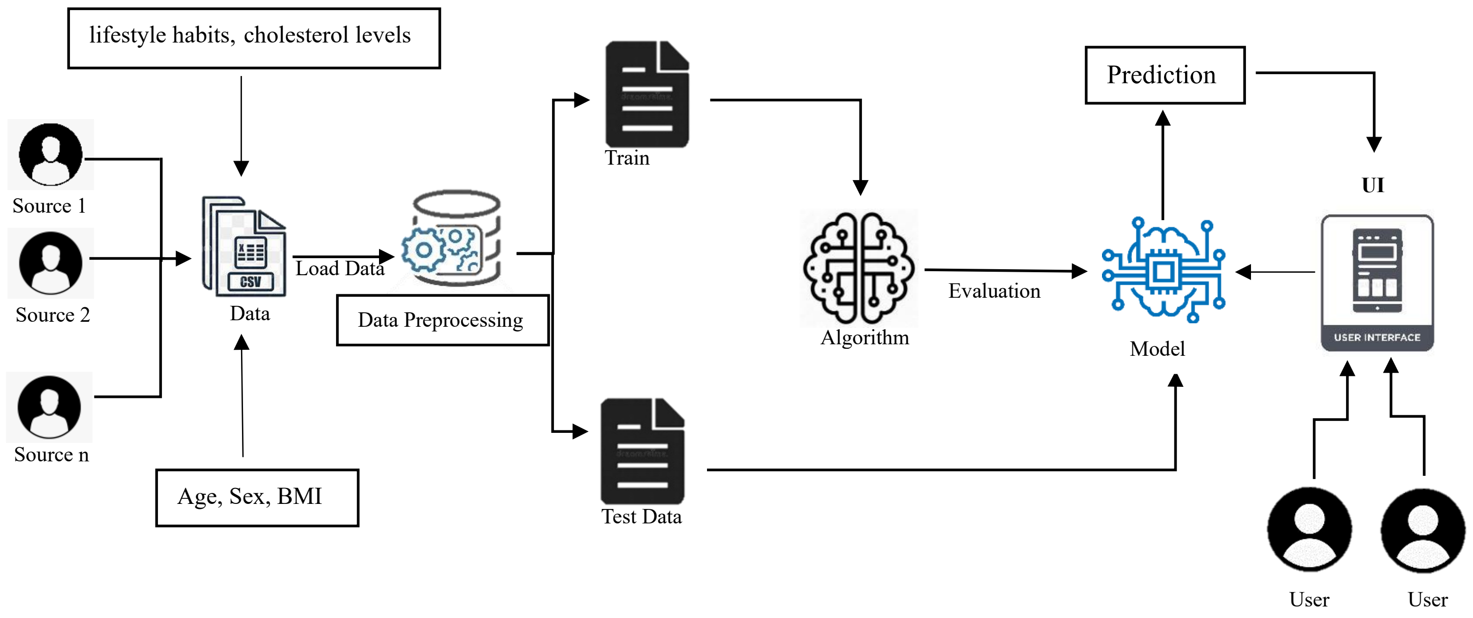
Diabetes Prediction Using Machine Learning

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.

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



Project Flow:

● User interacts with the UI to enter the input.

● 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

 o Specify the business problem and business requirements

 o Literature Survey

 o Social or Business Impact.

● Data Collection & Preparation

 o Collect the dataset and preparing the data

● Exploratory Data Analysis

 o Descriptive statistical and Visual Analysis

 o Visual Analysis

● Model Building

 o Training the model in multiple algorithms

 o Testing the model

● Performance Testing

 o Testing model with multiple evaluation metrics

● Model Deployment

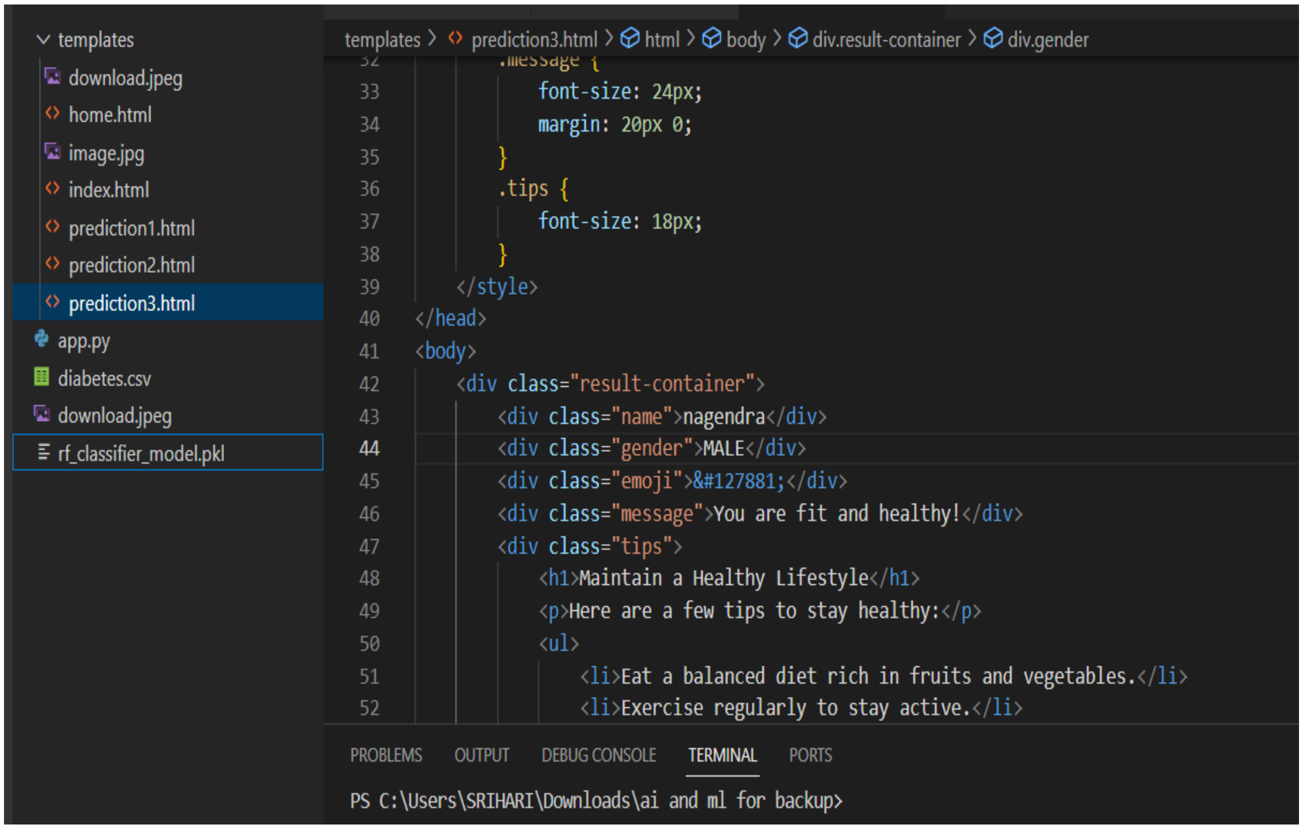
 o Save the best model and Integrate with Web Framework

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
- 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:



```
            font-size: 24px;
            margin: 20px 0;
        }
        .tips {
            font-size: 18px;
        }
    </style>
</head>
<body>
    <div class="result-container">
        <div class="name">nagendra</div>
        <div class="gender">MALE</div>
        <div class="emoji">&#127881;</div>
        <div class="message">You are fit and healthy!</div>
        <div class="tips">
            <h1>Maintain a Healthy Lifestyle</h1>
            <p>Here are a few tips to stay healthy:</p>
            <ul>
                <li>Eat a balanced diet rich in fruits and vegetables.</li>
                <li>Exercise regularly to stay active.</li>
```

```
PS C:\Users\SRIHARI\Downloads\ai and ml for backup>
```

Create the Project folder which contains files as shown below

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- rf_classifier_model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains a model training file.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

The business problem addressed in this project is the early detection and prediction of diabetes using machine learning algorithms. The goal is to develop a predictive model that can accurately identify individuals at high risk of developing diabetes based on their health records and other relevant factors. Early detection and management of diabetes can improve healthcare outcomes, reduce costs, and benefit healthcare providers and insurance companies. Therefore, developing an accurate and reliable predictive model for diabetes detection can have a significant impact on healthcare outcomes and costs.

Activity 2: Business requirements

Business requirements are the specific needs and expectations of the business stakeholders regarding the desired outcome of the project. In the case of the diabetes prediction project, the following are the key business requirements:

- Accurate prediction: The predictive model should be accurate in identifying individuals who are at high risk of developing diabetes based on their health records and other relevant factors.
- Efficiency: The model should be efficient and fast in analyzing large amounts of data to provide timely predictions.
- Scalability: The model should be scalable to handle large datasets and accommodate future growth in data volume.
- Flexibility: The model should be flexible and adaptable to accommodate changes in data sources or input parameters.
- User-friendliness: The model should be user-friendly, easy to use, and understand by healthcare providers and insurance companies.
- Integration: The model should be easily integrated with existing healthcare systems and processes.
- Security: The model should be secure and protect patient data privacy.
- Compliance: The model should comply with relevant healthcare regulations and standards.

Activity 3: Literature Survey

A literature survey is an essential step in any diabetes prediction using machine learning:

- "Machine Learning for Diabetes Prediction: A Review" by E. Şahin et al. This paper provides a comprehensive review of the latest research on machine learning for diabetes prediction. The authors discuss the challenges, approaches, and evaluation metrics used in various studies.

- "Predicting Type 2 Diabetes Mellitus Using Machine Learning Techniques" by S. Chakraborty et al. This paper proposes a machine learning approach for predicting the risk of developing type 2 diabetes mellitus based on demographic and clinical data. The authors compare various algorithms and feature selection techniques and evaluate their performance.

- "Deep Learning for Diabetes Prediction: A Review" by Y. Zhao et al. This paper provides a review of the latest research on deep learning for diabetes prediction, with a focus on the use of convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

- "Diabetes Prediction Using Machine Learning Techniques: A Comparative Study" by S. B. Gawali and R. K. Kamat. This paper compares the performance of various machine learning algorithms, including logistic regression, decision trees, and neural networks, for diabetes prediction using clinical and demographic data.
- "Machine Learning for Early Detection of Diabetic Retinopathy" by A. Gulshan et al. This paper proposes a deep learning approach for the early detection of diabetic retinopathy based on retinal images. The authors use a CNN to classify images into different stages of the disease and achieve high accuracy.

Activity 4: Social or Business Impact

Accurate diabetes prediction using machine learning can have a significant social and business impact. It can help identify individuals at high risk of developing diabetes, leading to earlier intervention and prevention efforts. From a business perspective, accurate prediction can help healthcare providers and insurers manage healthcare costs and resources better. Additionally, it can lead to more personalized healthcare, improving patient outcomes and adherence to treatment plans. In conclusion, accurate diabetes prediction using machine learning can improve patient outcomes, reduce healthcare costs, and lead to more personalized healthcare.

Milestone 2: Data Collection & Preparation

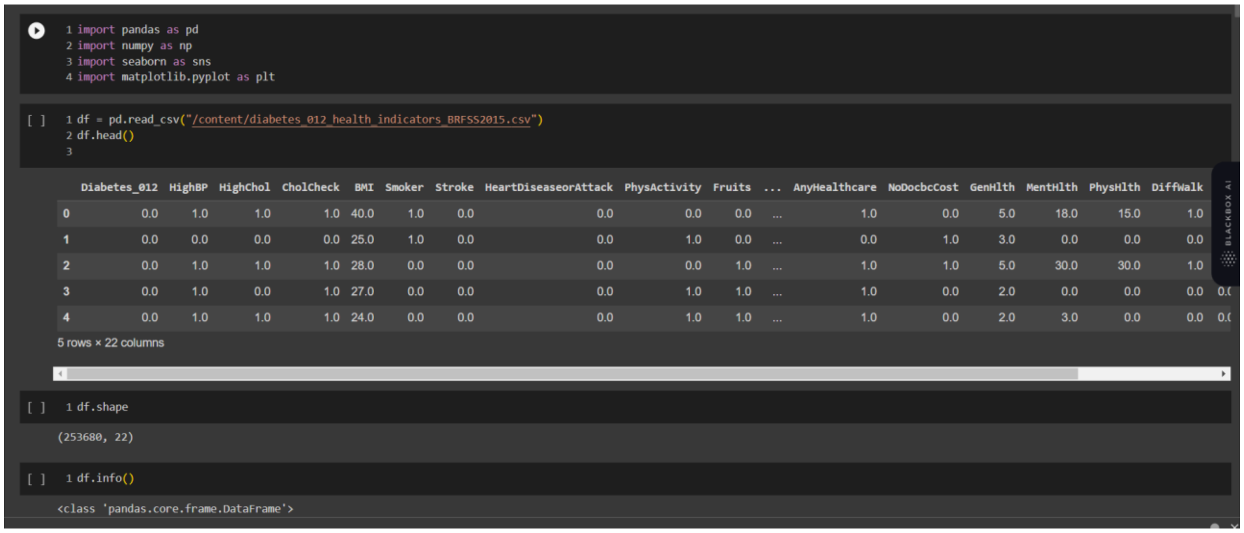
ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

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 as shown in the image.



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

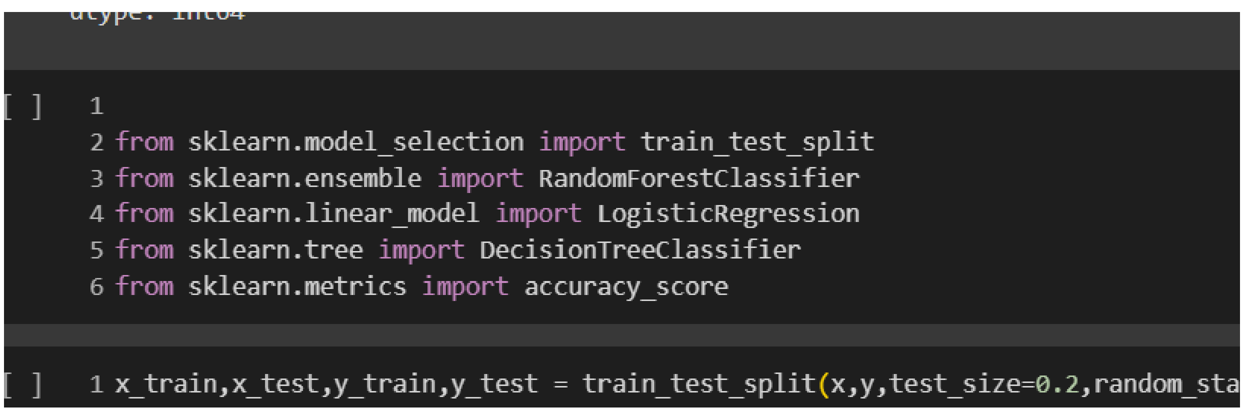
```
[ ] df = pd.read_csv("/content/diabetes_012_health_indicators_BRFSS2015.csv")
    df.head()
```

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	5.0	18.0	15.0	1.0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0	1.0	3.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0	1.0	5.0	30.0	30.0	1.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0	0.0	0.0	0.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0	3.0	0.0	0.0

5 rows × 22 columns

```
[ ] df.shape

(253680, 22)
```

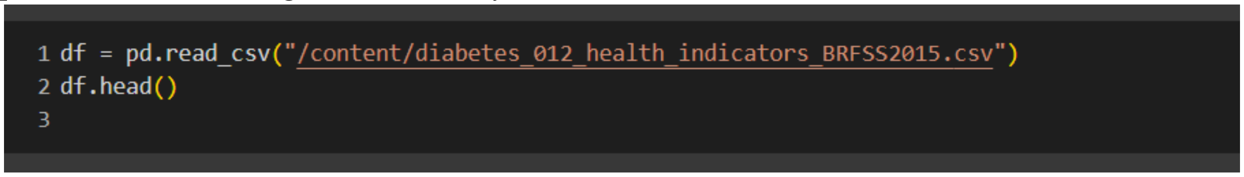
```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
```



```
    dtype: int64

[ ]
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import accuracy_score

[ ] x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_sta
```

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.



```
df = pd.read_csv("/content/diabetes_012_health_indicators_BRFSS2015.csv")
df.head()
```

Activity 2: Data Preparation

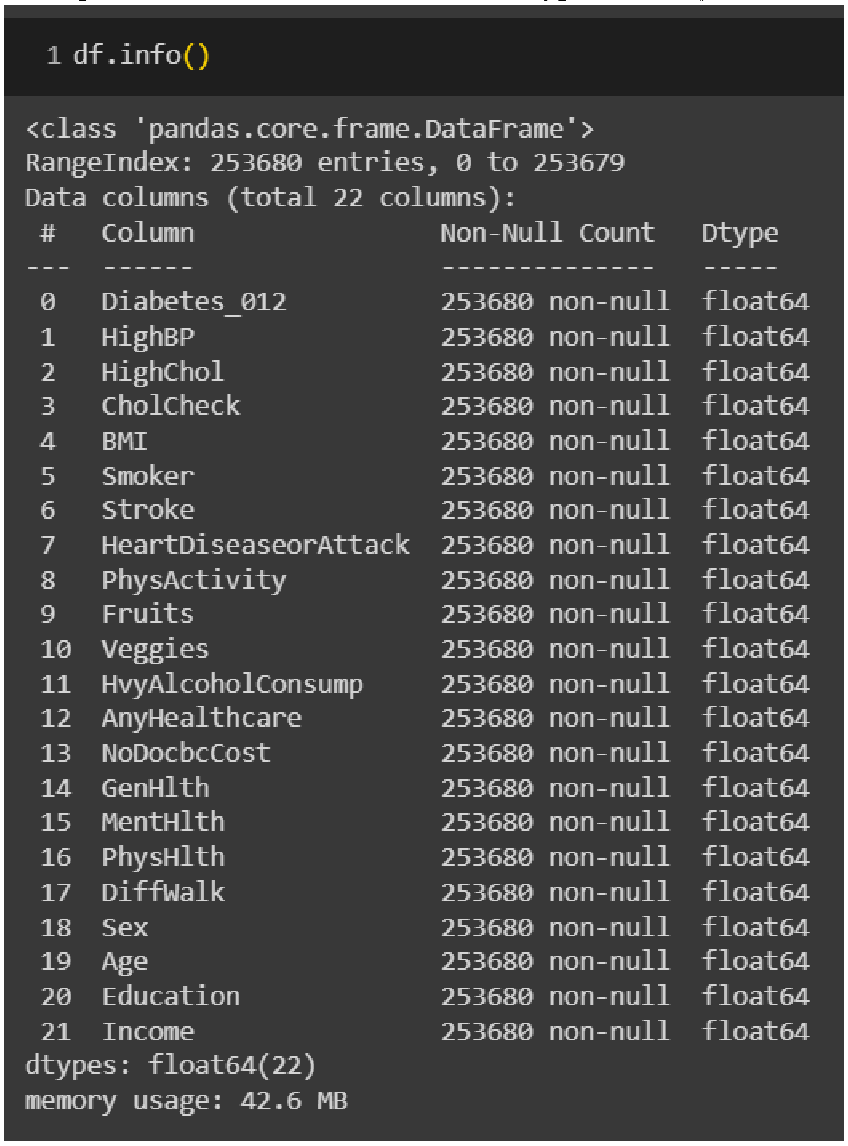
As we have understood how the data is, let's pre-process the collected data.
The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following 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

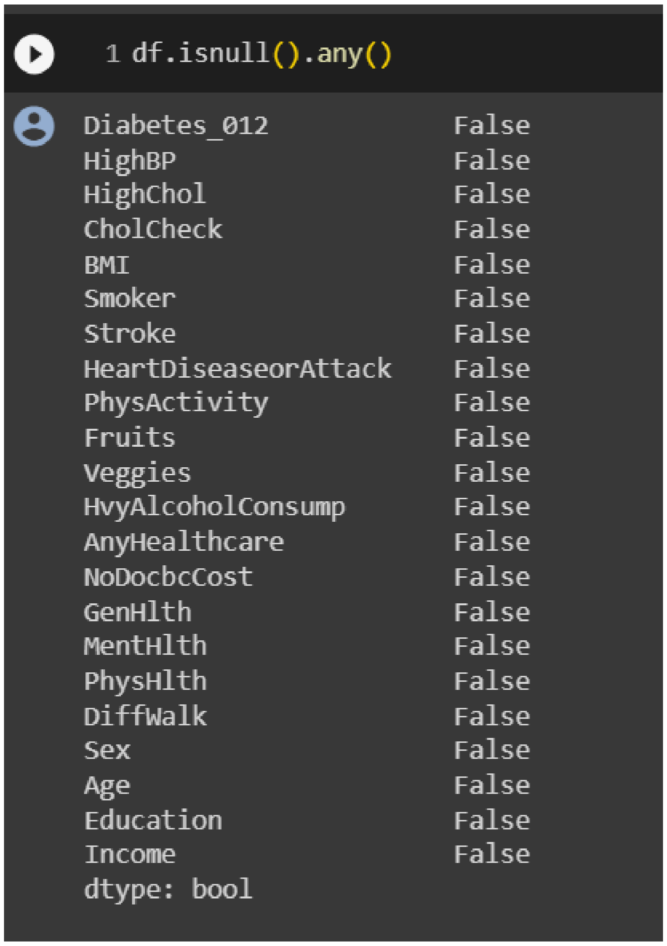
● Let's know the info and describe of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_012          253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

● 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.



```
df.isnull().any()
```

```
Diabetes_012            False
HighBP                  False
HighChol                False
CholCheck               False
BMI                     False
Smoker                  False
Stroke                  False
HeartDiseaseorAttack    False
PhysActivity            False
Fruits                  False
Veggies                 False
HvyAlcoholConsump       False
AnyHealthcare           False
NoDocbcCost             False
GenHlth                 False
MentHlth                False
PhysHlth                False
DiffWalk                False
Sex                     False
Age                     False
Education               False
Income                  False
dtype: bool
```

Activity 2.2: Handling Categorical Values

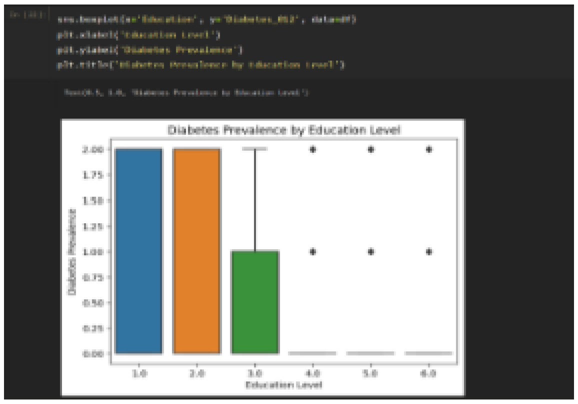
As we can see our dataset has categorical data, we must convert the categorical data to integer encoding or binary encoding.
To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using Label encoding with the help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

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

Activity 2.3: Handling Imbalance Data

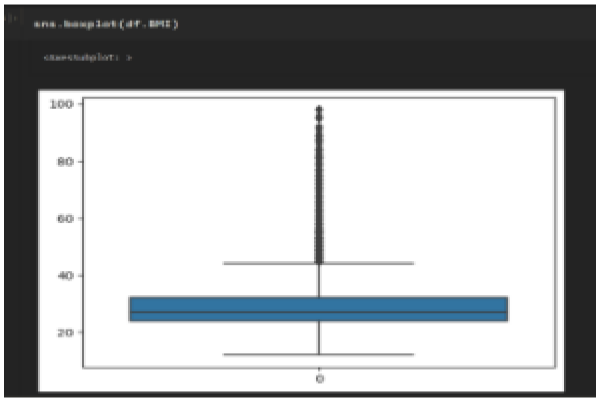
With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of some columns feature with some mathematical formula

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



```
sns.boxplot(x='Education', y='Diabetes_012', data=df)
plt.xlabel('Education Level')
plt.ylabel('Diabetes Prevalence')
plt.title('Diabetes Prevalence by Education Level')
```

```
Text(0.5, 1.0, 'Diabetes Prevalence by Education Level')
```



```
sns.boxplot(df.BMI)
```

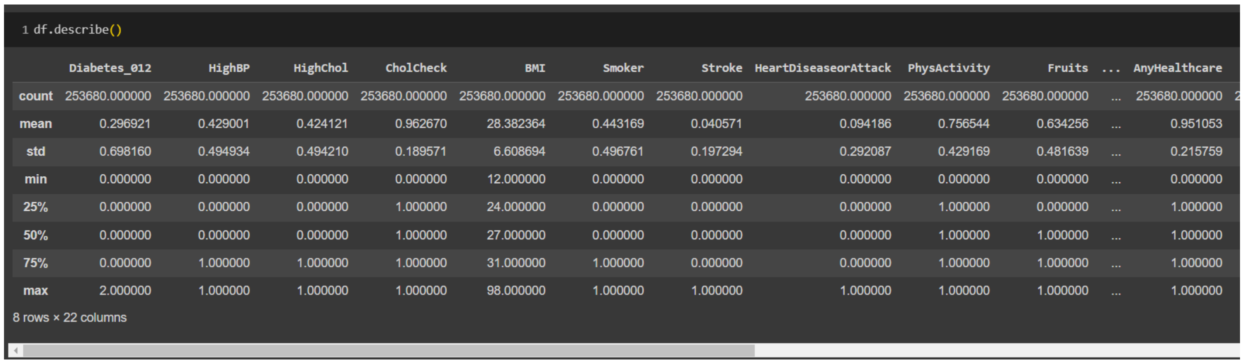
```
<AxesSubplot: >
```

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

.



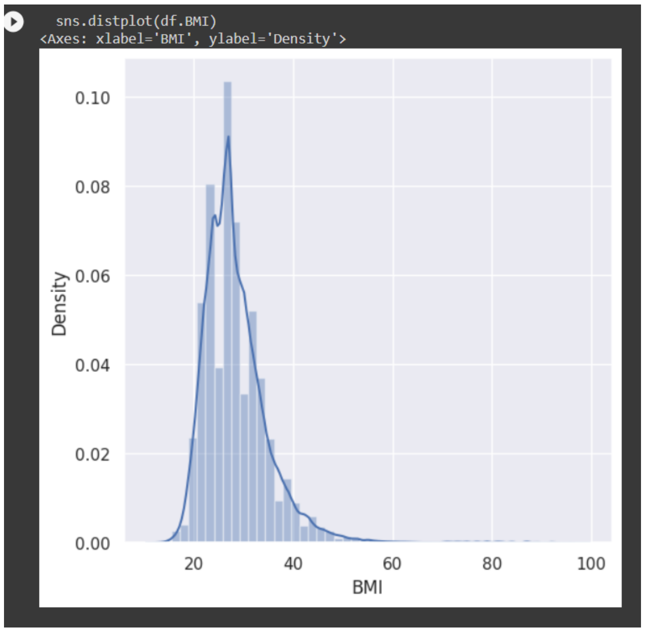
```
df.describe()
```

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	...	253680.000000
mean	0.296921	0.429001	0.424121	0.962670	28.382364	0.443169	0.040571	0.094186	0.756544	0.634256	...	0.951053
std	0.698160	0.494934	0.494210	0.189571	6.608694	0.496761	0.197294	0.292087	0.429169	0.481639	...	0.215759
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	0.000000	1.000000	0.000000	...	1.000000
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	0.000000	1.000000	1.000000	...	1.000000
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	0.000000	1.000000	1.000000	...	1.000000
max	2.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000

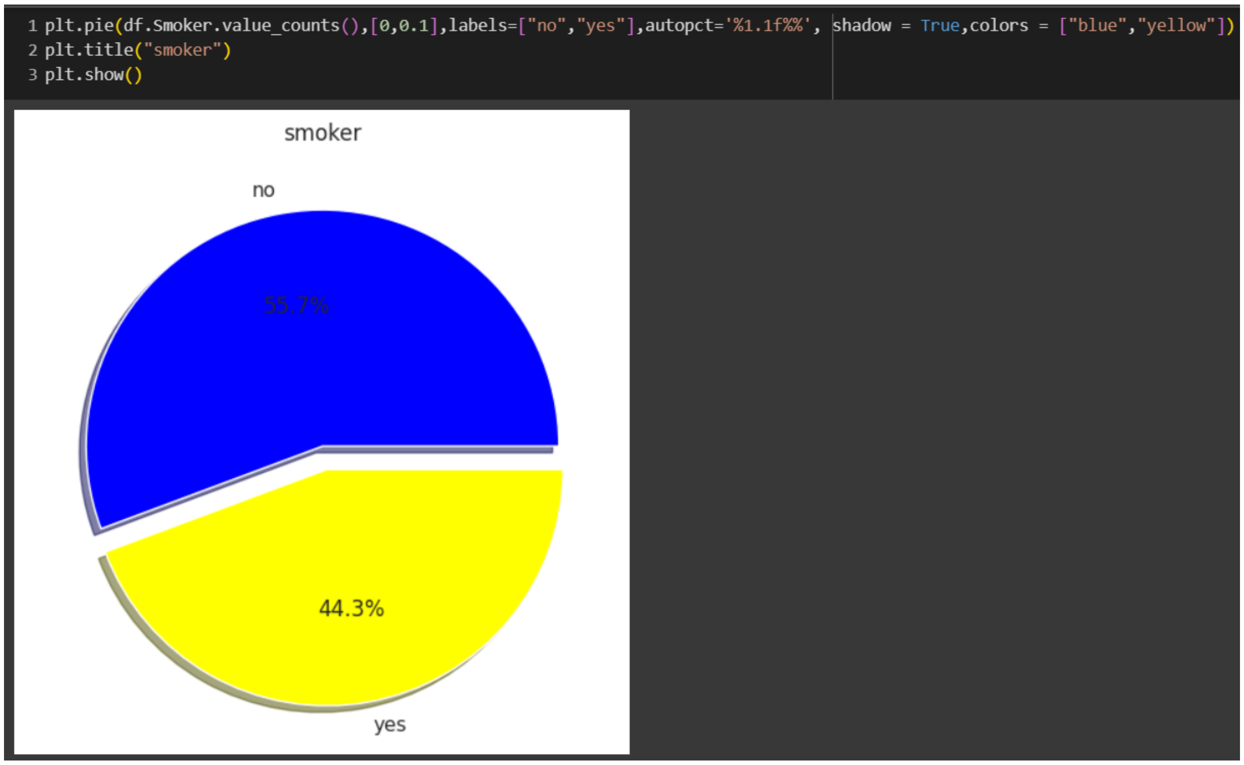
8 rows × 22 columns

Activity 2: Visual analysis

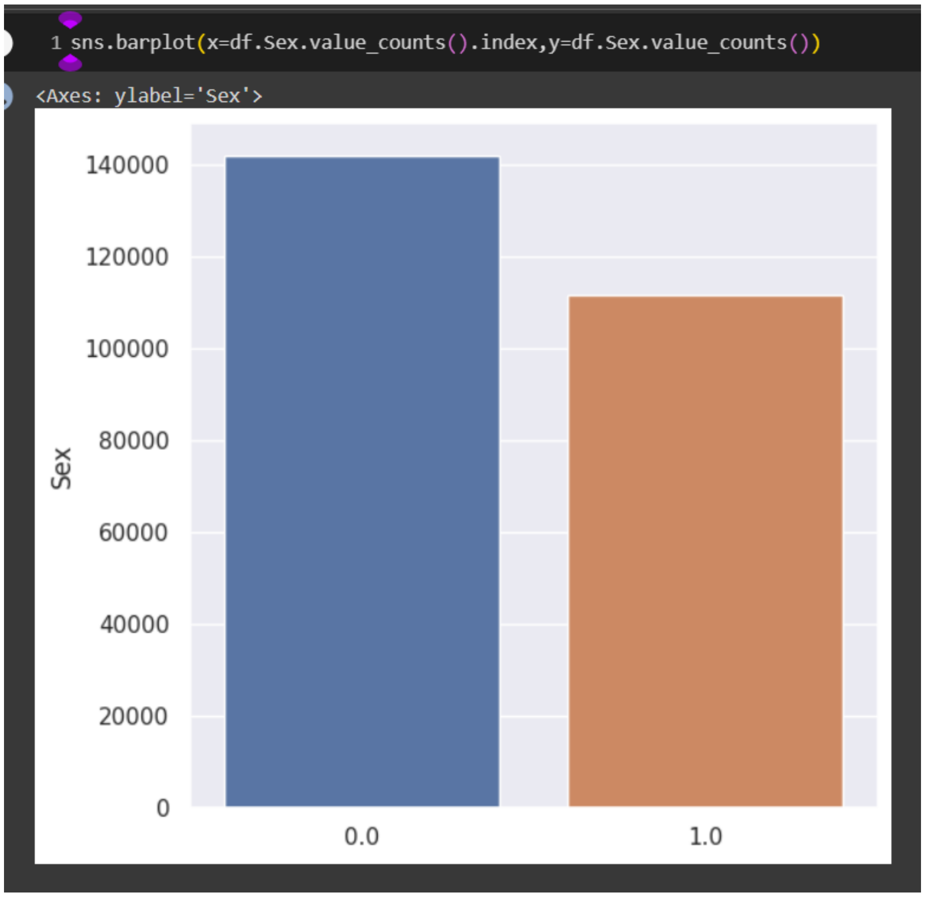
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.



```
sns.distplot(df.BMI)
<Axes: xlabel='BMI', ylabel='Density'>
```



```
plt.pie(df.Smoker.value_counts(),[0,0.1],labels=["no","yes"],autopct='%1.1f%%', shadow = True,colors = ["blue","yellow"])
plt.title("smoker")
plt.show()
```

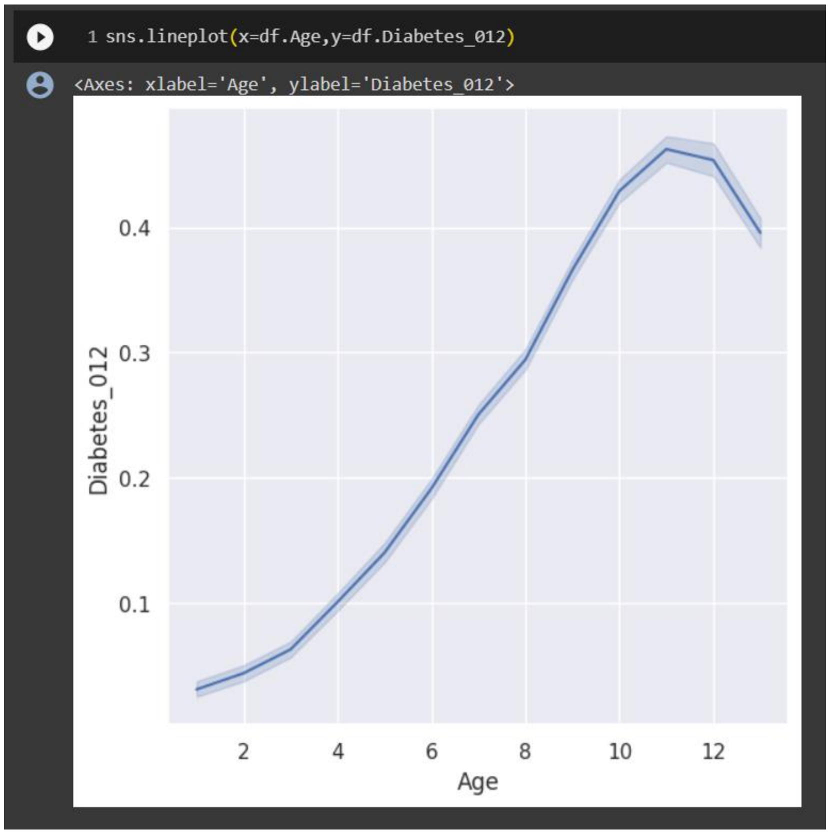


```
sns.barplot(x=df.Sex.value_counts().index,y=df.Sex.value_counts())
```

```
<Axes: ylabel='Sex'>
```

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.

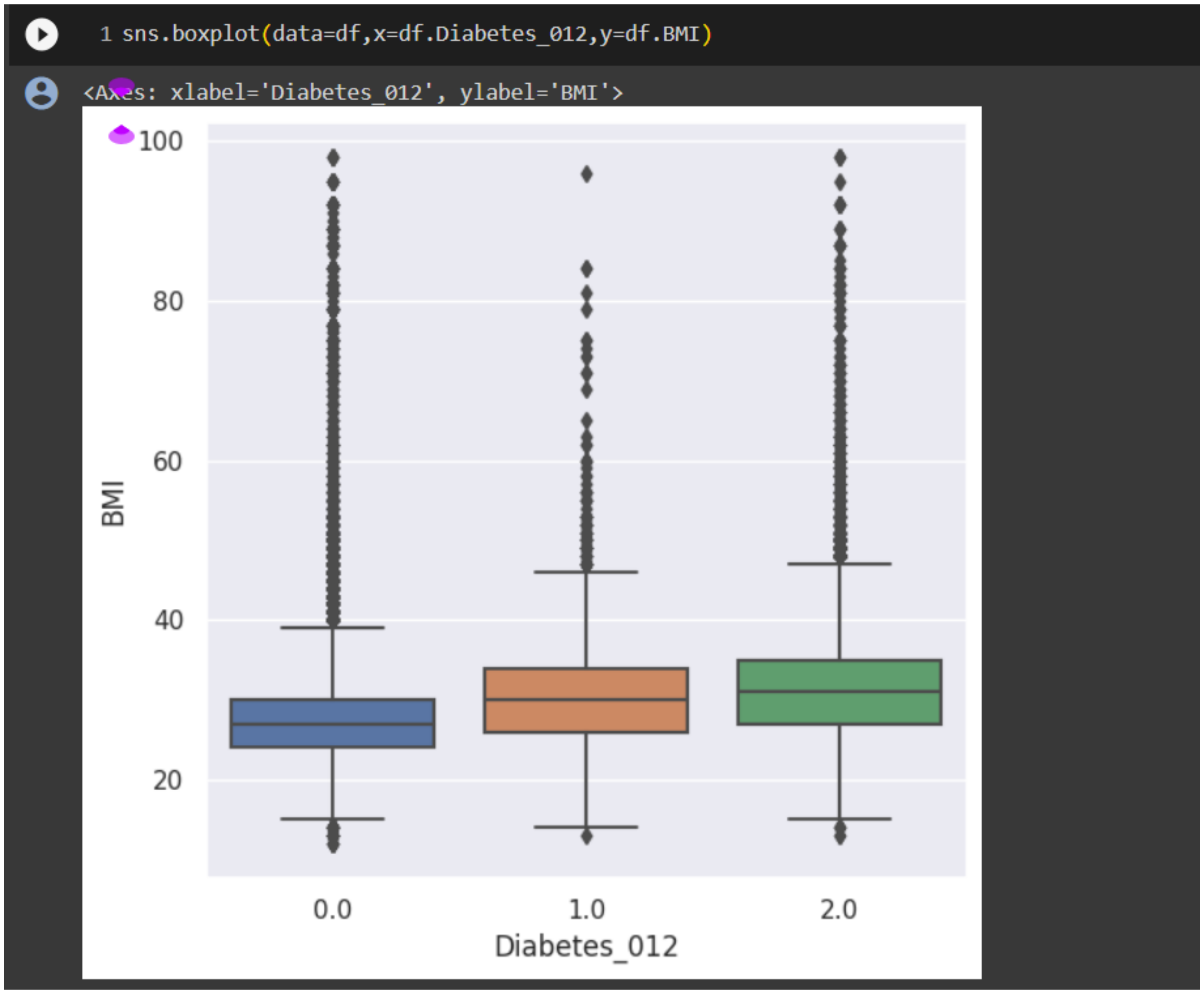


```
sns.lineplot(x=df.Age,y=df.Diabetes_012)
```

```
<Axes: xlabel='Age', ylabel='Diabetes_012'>
```

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.

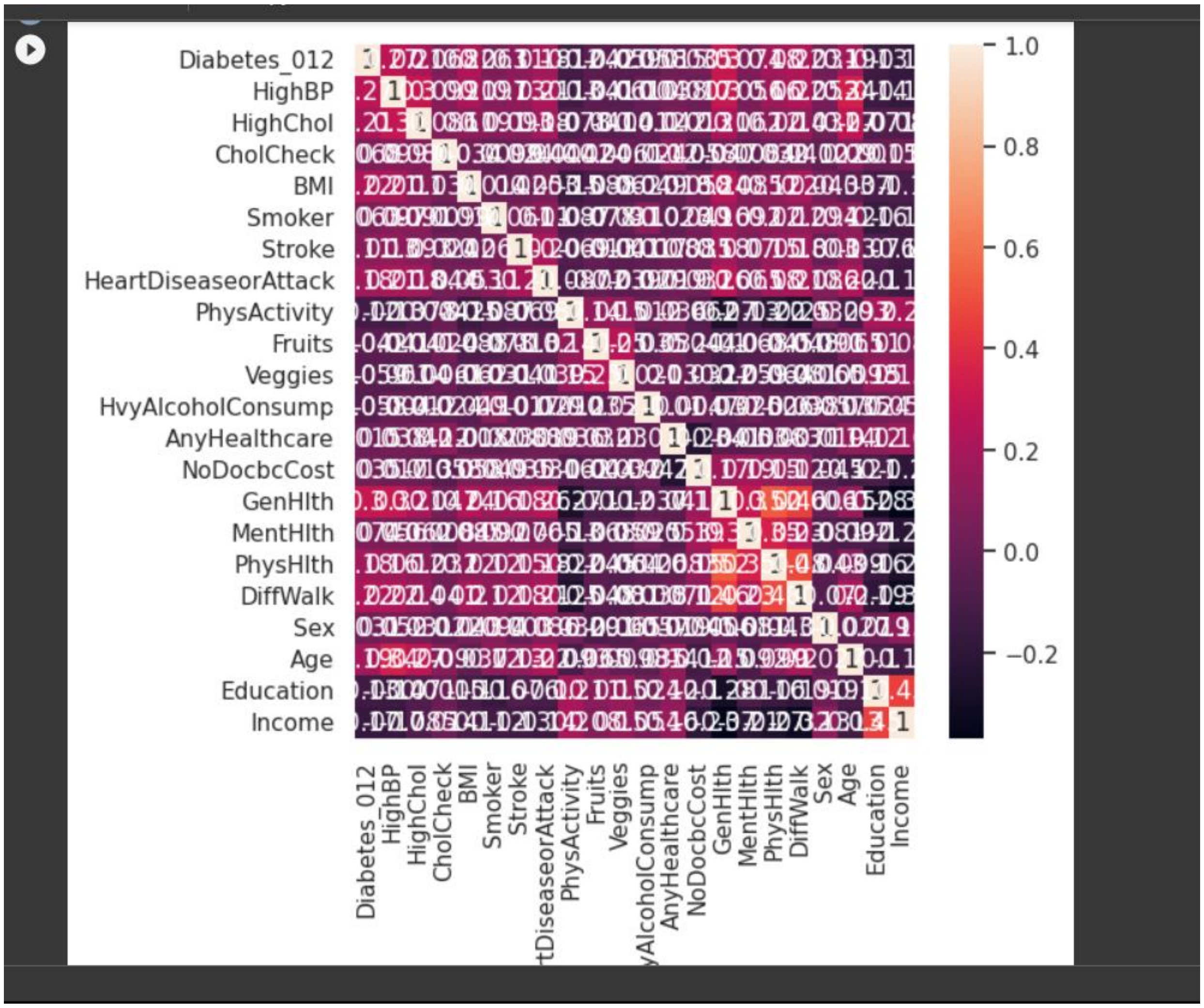


```
sns.boxplot(data=df,x=df.Diabetes_012,y=df.BMI)
```

```
<Axes: xlabel='Diabetes_012', ylabel='BMI'>
```

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.

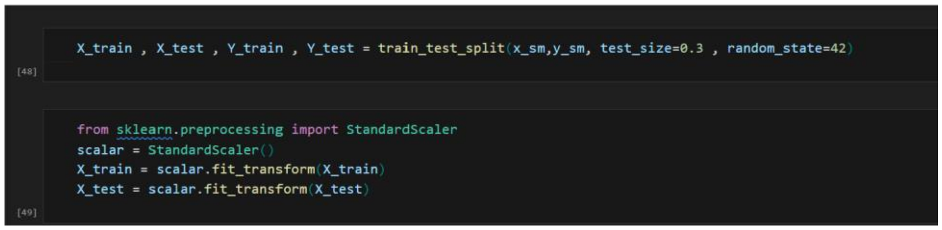

Diabetes_012
HighBP
HighChol
CholCheck
BMI
Smoker
Stroke
HeartDiseaseorAttack
PhysActivity
Fruits
Veggies
HvyAlcoholConsump
AnyHealthcare
NoDocbcCost
GenHlth
MentHlth
PhysHlth
DiffWalk
Sex
Age
Education
Income
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

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.



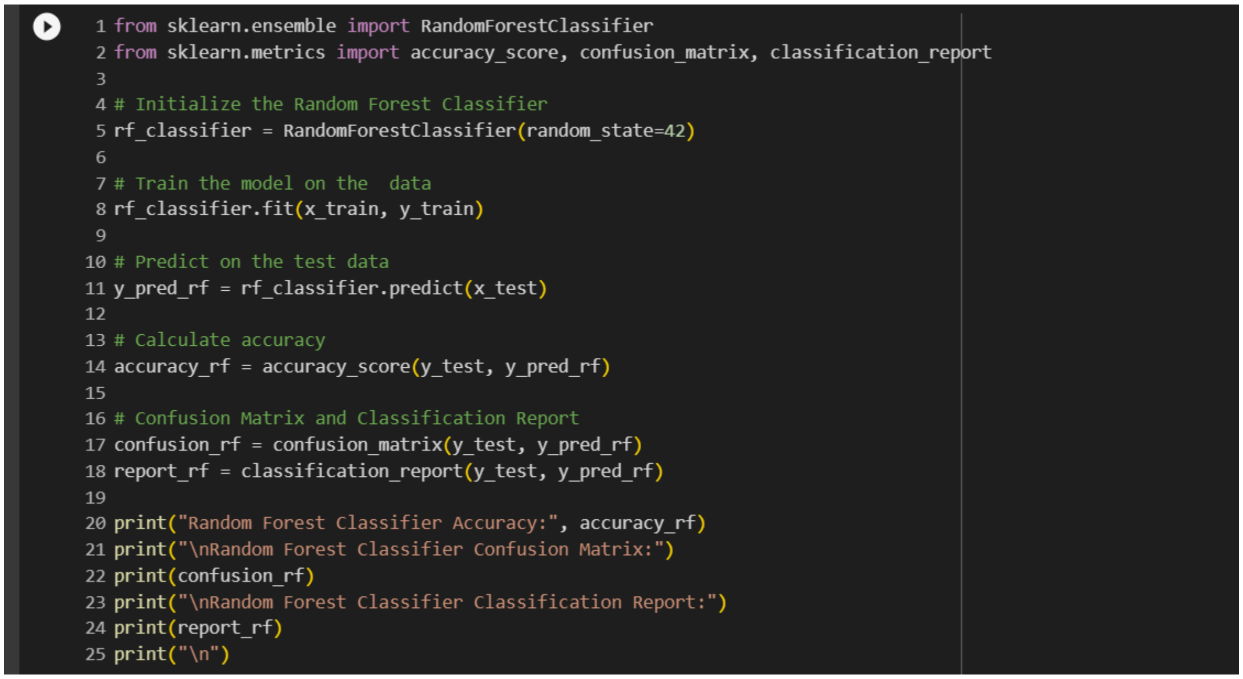
```
[48]
X_train , X_test , Y_train , Y_test = train_test_split(x_sm,y_sm, test_size=0.3 , random_state=42)
```

```
[49]
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
X_train = scalar.fit_transform(X_train)
X_test = scalar.fit_transform(X_test)
```

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.

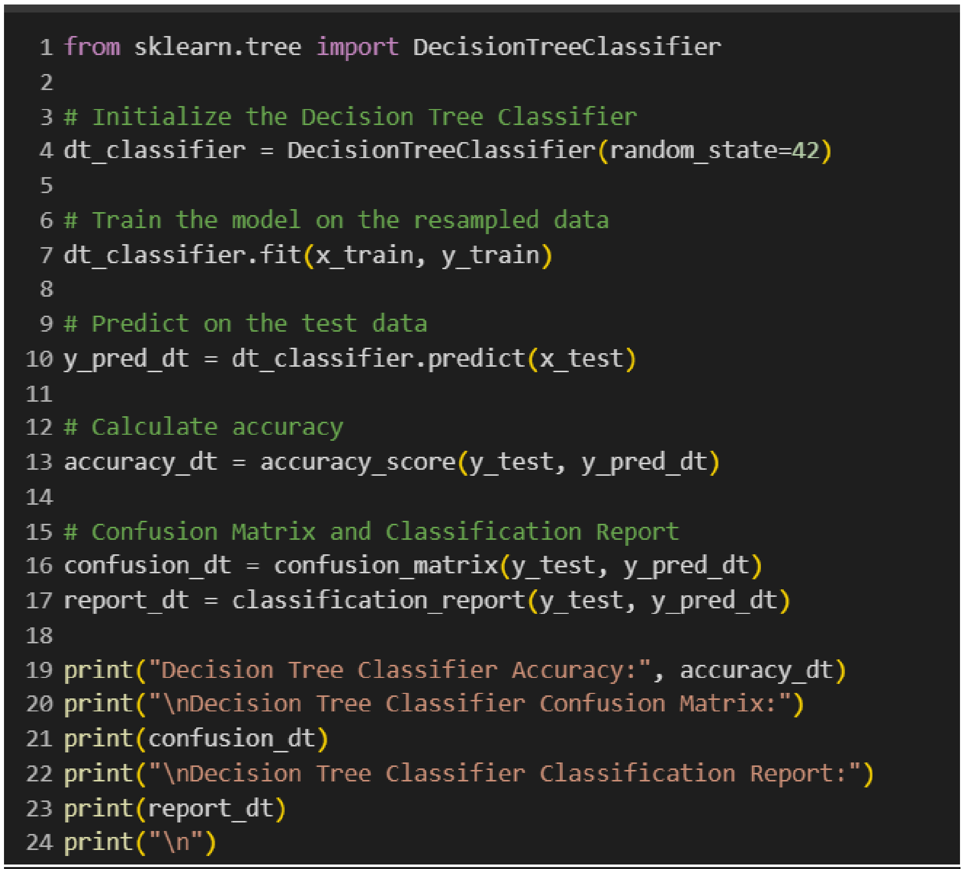
Activity 1.1: Random Forest Regressor
A function named random forest regressor is created and train and test data are passed as the parameters. Inside the function, random forest regressor algorithm is initialized and training data is passed to the model with the fit() function. Test data is predicted with predict () function and saved in a new variable. For evaluating the model with R2_score.

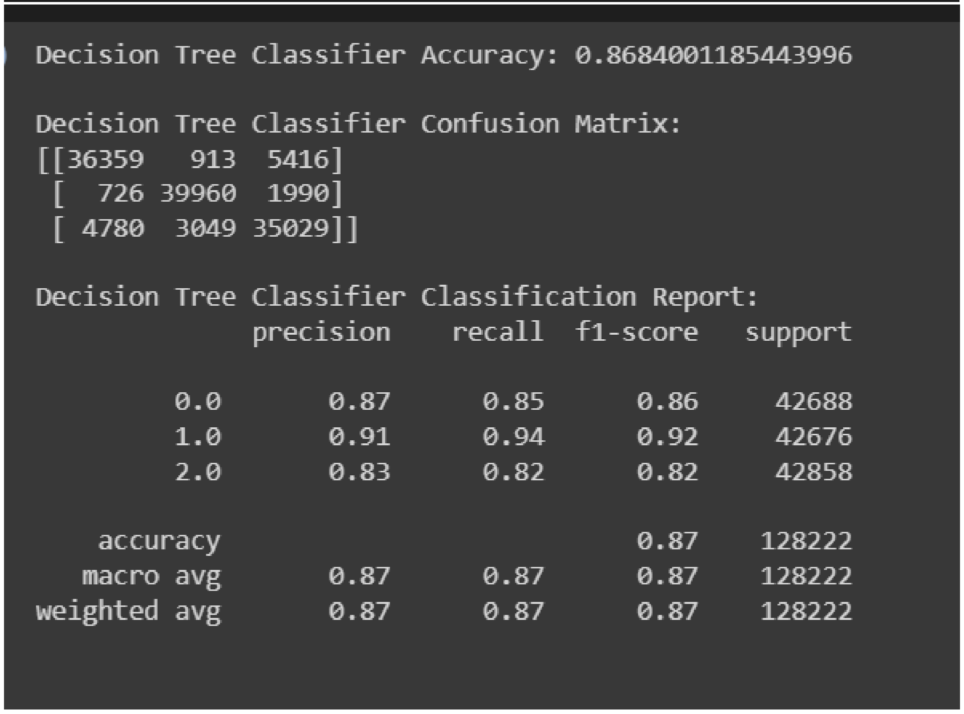


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model on the  data
rf_classifier.fit(x_train, y_train)

# Predict on the test data
y_pred_rf = rf_classifier.predict(x_test)

# Calculate accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Confusion Matrix and Classification Report
confusion_rf = confusion_matrix(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Classifier Accuracy:", accuracy_rf)
print("\nRandom Forest Classifier Confusion Matrix:")
print(confusion_rf)
print("\nRandom Forest Classifier Classification Report:")
print(report_rf)
print("\n")
```

Activity 1.2: Decision Tree Regressor

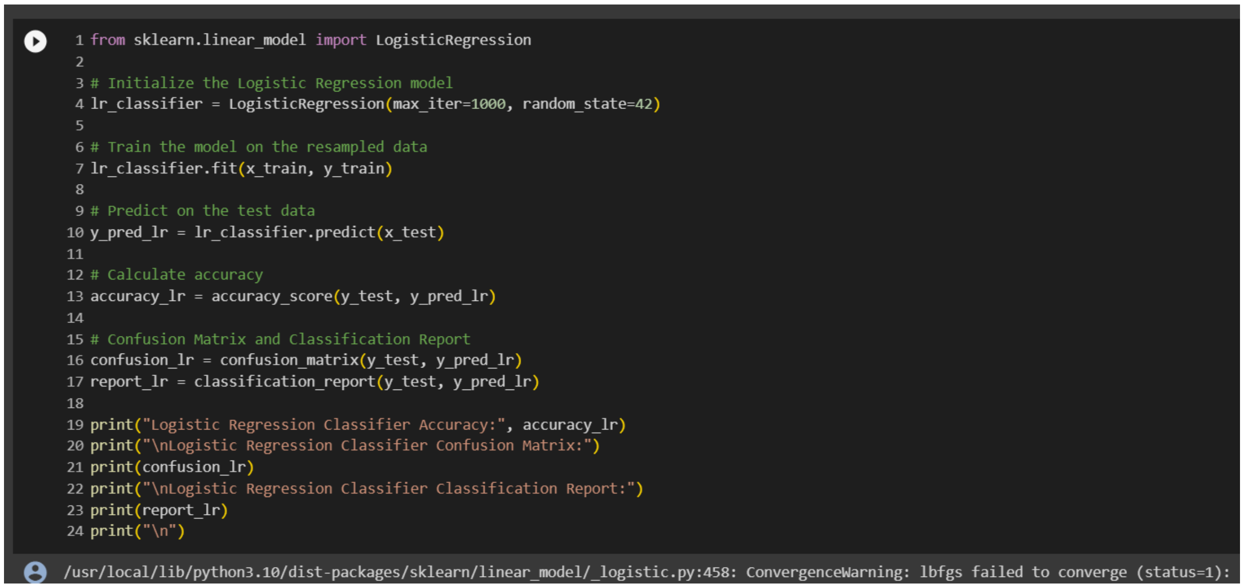
A function named decision Tree regressor is created and train and test data are passed as the parameters. Inside the function, decision Tree regressor algorithm is initialized and training data is passed to the model with fit() function. Test data is predicted with predict () function and saved in a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


```python
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the model on the resampled data
dt_classifier.fit(x_train, y_train)

# Predict on the test data
y_pred_dt = dt_classifier.predict(x_test)

# Calculate accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# Confusion Matrix and Classification Report
confusion_dt = confusion_matrix(y_test, y_pred_dt)
report_dt = classification_report(y_test, y_pred_dt)

print("Decision Tree Classifier Accuracy:", accuracy_dt)
print("\nDecision Tree Classifier Confusion Matrix:")
print(confusion_dt)
print("\nDecision Tree Classifier Classification Report:")
print(report_dt)
print("\n")
```


```
Decision Tree Classifier Accuracy: 0.8684001185443996

Decision Tree Classifier Confusion Matrix:
[[36359   913  5416]
 [  726 39960  1990]
 [ 4780  3049 35029]]

Decision Tree Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.85      0.86     42688
         1.0       0.91      0.94      0.92     42676
         2.0       0.83      0.82      0.82     42858

    accuracy                           0.87    128222
   macro avg       0.87      0.87      0.87    128222
weighted avg       0.87      0.87      0.87    128222
```

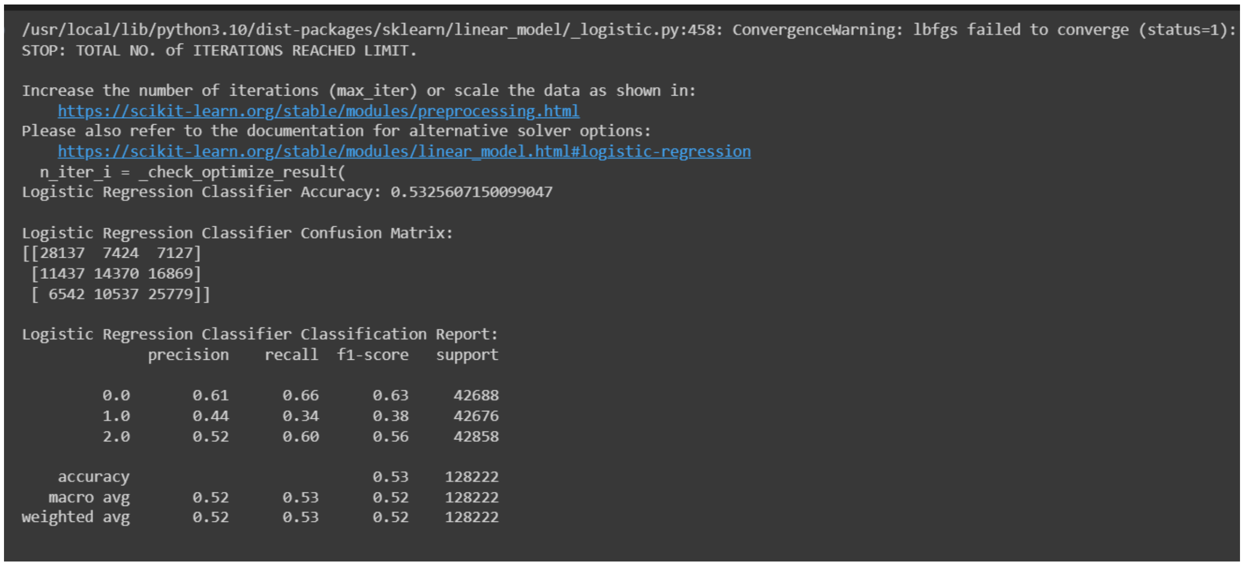
Activity 1.3: Logistic Regressor

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.



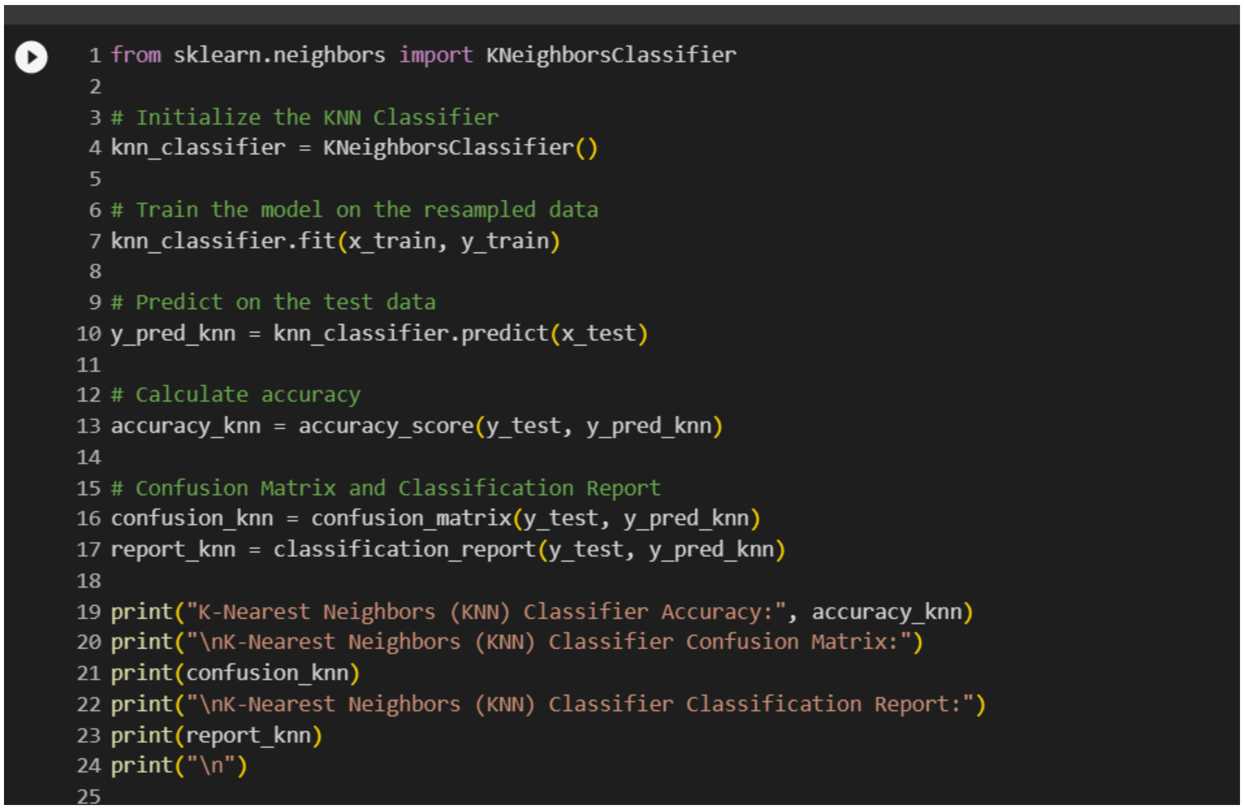
```python
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
lr_classifier = LogisticRegression(max_iter=1000, random_state=42)

# Train the model on the resampled data
lr_classifier.fit(x_train, y_train)

# Predict on the test data
y_pred_lr = lr_classifier.predict(x_test)

# Calculate accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# Confusion Matrix and Classification Report
confusion_lr = confusion_matrix(y_test, y_pred_lr)
report_lr = classification_report(y_test, y_pred_lr)

print("Logistic Regression Classifier Accuracy:", accuracy_lr)
print("\nLogistic Regression Classifier Confusion Matrix:")
print(confusion_lr)
print("\nLogistic Regression Classifier Classification Report:")
print(report_lr)
print("\n")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
```

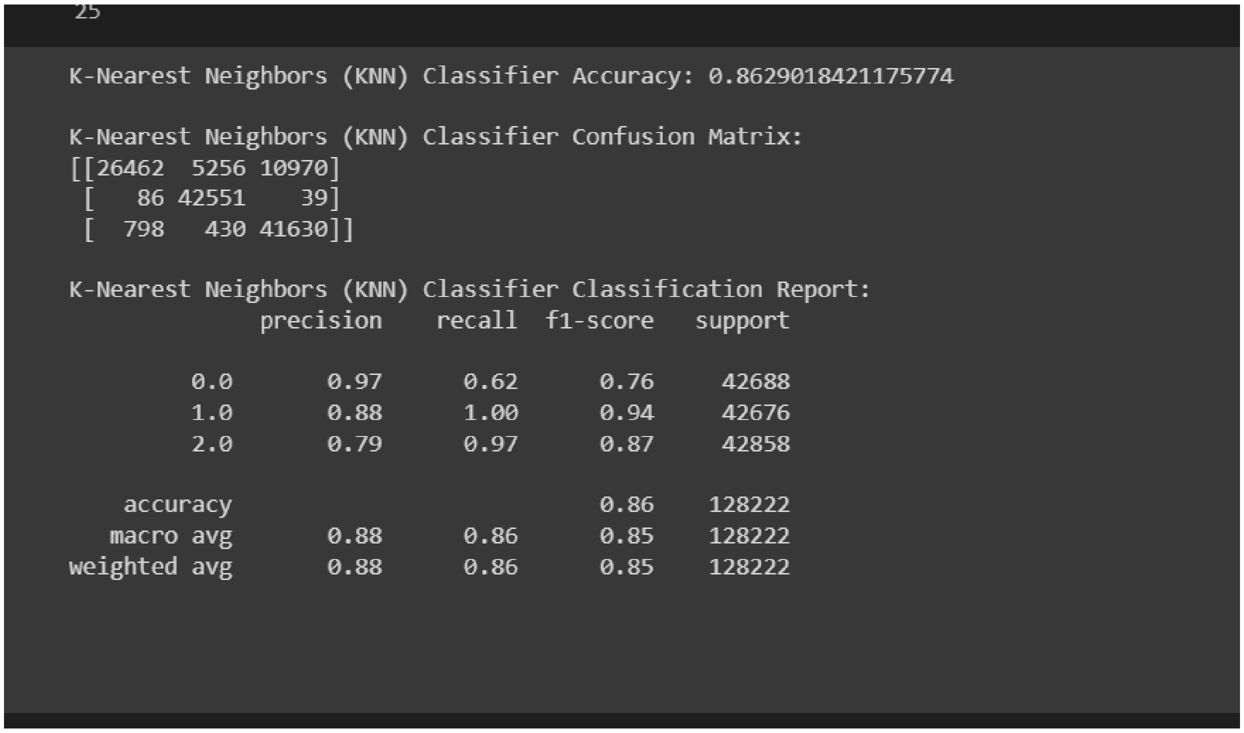


```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Classifier Accuracy: 0.5325607150099047

Logistic Regression Classifier Confusion Matrix:
[[28137  7424  7127]
 [11437 14370 16869]
 [ 6542 10537 25779]]

Logistic Regression Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.61      0.66      0.63     42688
         1.0       0.44      0.34      0.38     42676
         2.0       0.52      0.60      0.56     42858

    accuracy                           0.53    128222
   macro avg       0.52      0.53      0.52    128222
weighted avg       0.52      0.53      0.52    128222
```

Knn:



```python
from sklearn.neighbors import KNeighborsClassifier

# Initialize the KNN Classifier
knn_classifier = KNeighborsClassifier()

# Train the model on the resampled data
knn_classifier.fit(x_train, y_train)

# Predict on the test data
y_pred_knn = knn_classifier.predict(x_test)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# Confusion Matrix and Classification Report
confusion_knn = confusion_matrix(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)

print("K-Nearest Neighbors (KNN) Classifier Accuracy:", accuracy_knn)
print("\nK-Nearest Neighbors (KNN) Classifier Confusion Matrix:")
print(confusion_knn)
print("\nK-Nearest Neighbors (KNN) Classifier Classification Report:")
print(report_knn)
print("\n")

```



```
K-Nearest Neighbors (KNN) Classifier Accuracy: 0.8629018421175774

K-Nearest Neighbors (KNN) Classifier Confusion Matrix:
[[26462  5256 10970]
 [   86 42551    39]
 [  798   430 41630]]

K-Nearest Neighbors (KNN) Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.62      0.76     42688
         1.0       0.88      1.00      0.94     42676
         2.0       0.79      0.97      0.87     42858

    accuracy                           0.86    128222
   macro avg       0.88      0.86      0.85    128222
weighted avg       0.88      0.86      0.85    128222
```

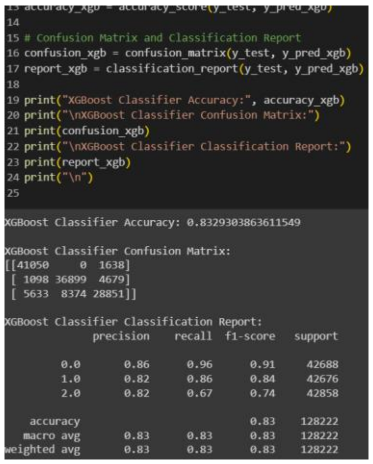
Activity 2: Testing the model

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

Milestone 5: Performance Testing

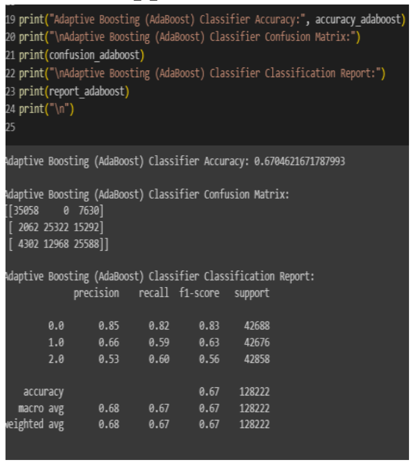
Activity 1: Testing model with multiple evaluation metrics

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.



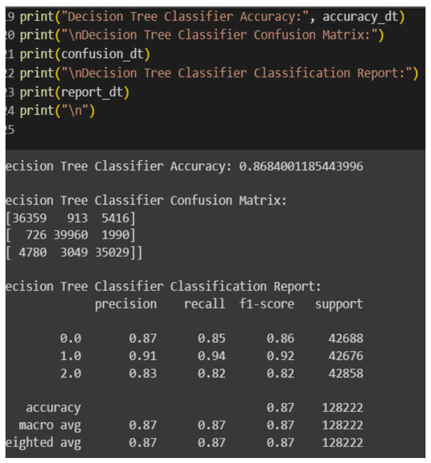
```
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)

# Confusion Matrix and Classification Report
confusion_xgb = confusion_matrix(y_test, y_pred_xgb)
report_xgb = classification_report(y_test, y_pred_xgb)

print("XGBoost Classifier Accuracy:", accuracy_xgb)
print("\nXGBoost Classifier Confusion Matrix:")
print(confusion_xgb)
print("\nXGBoost Classifier Classification Report:")
print(report_xgb)
print("\n")
```

```
XGBoost Classifier Accuracy: 0.8329303863611549

XGBoost Classifier Confusion Matrix:
[[41050     0  1638]
 [ 1098 36899  4679]
 [ 5633  8374 28851]]

XGBoost Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     42688
         1.0       0.82      0.86      0.84     42676
         2.0       0.82      0.67      0.74     42858

    accuracy                           0.83    128222
   macro avg       0.83      0.83      0.83    128222
weighted avg       0.83      0.83      0.83    128222
```



```
print("Adaptive Boosting (AdaBoost) Classifier Accuracy:", accuracy_adaboost)
print("\nAdaptive Boosting (AdaBoost) Classifier Confusion Matrix:")
print(confusion_adaboost)
print("\nAdaptive Boosting (AdaBoost) Classifier Classification Report:")
print(report_adaboost)
print("\n")
```

```
Adaptive Boosting (AdaBoost) Classifier Accuracy: 0.6704621671787993

Adaptive Boosting (AdaBoost) Classifier Confusion Matrix:
[[35058     0  7630]
 [ 2062 25322 15292]
 [ 4302 12968 25588]]

Adaptive Boosting (AdaBoost) Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.82      0.83     42688
         1.0       0.66      0.59      0.63     42676
         2.0       0.53      0.60      0.56     42858

    accuracy                           0.67    128222
   macro avg       0.68      0.67      0.67    128222
weighted avg       0.68      0.67      0.67    128222
```

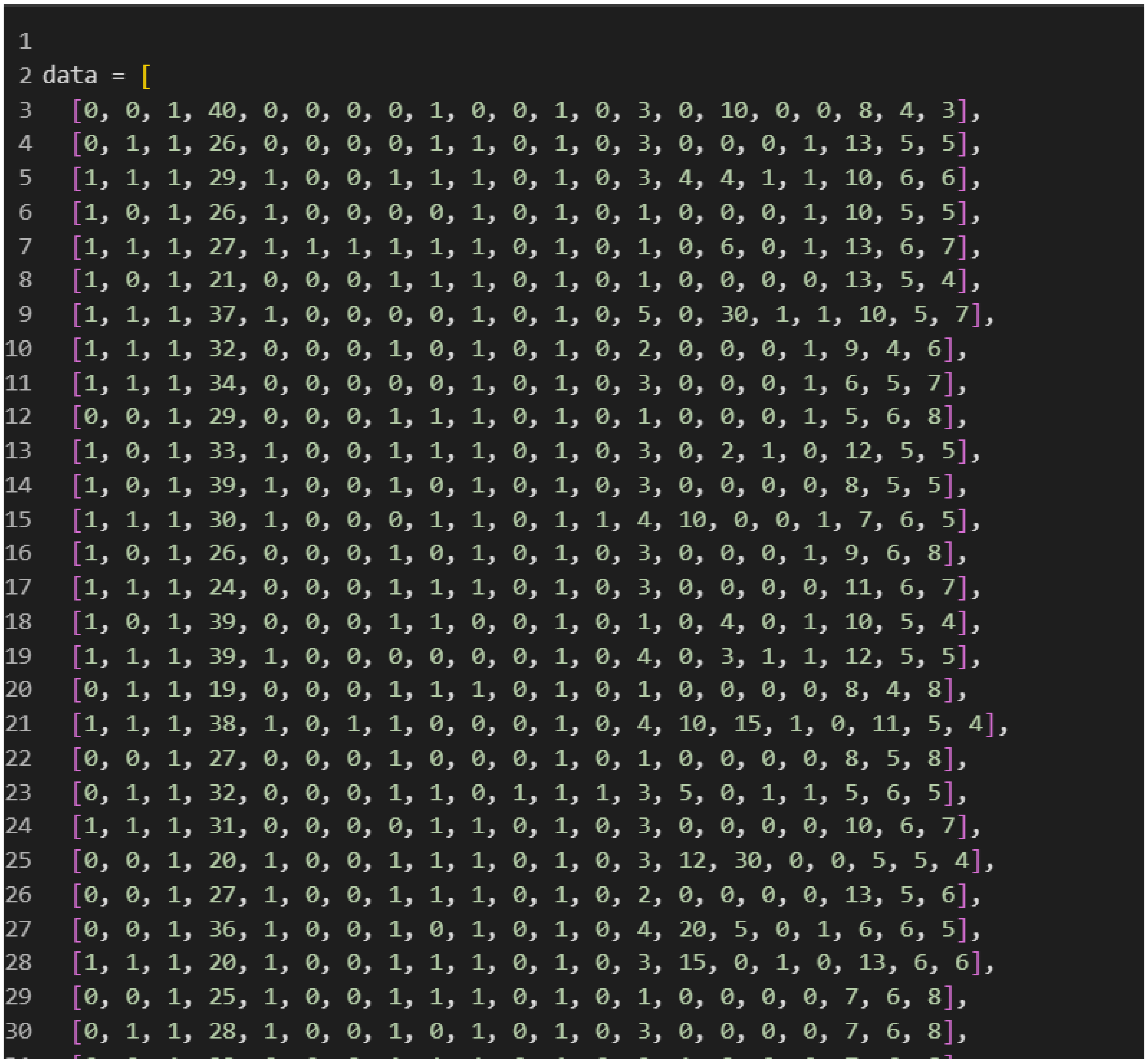


```
print("Decision Tree Classifier Accuracy:", accuracy_dt)
print("\nDecision Tree Classifier Confusion Matrix:")
print(confusion_dt)
print("\nDecision Tree Classifier Classification Report:")
print(report_dt)
print("\n")
```

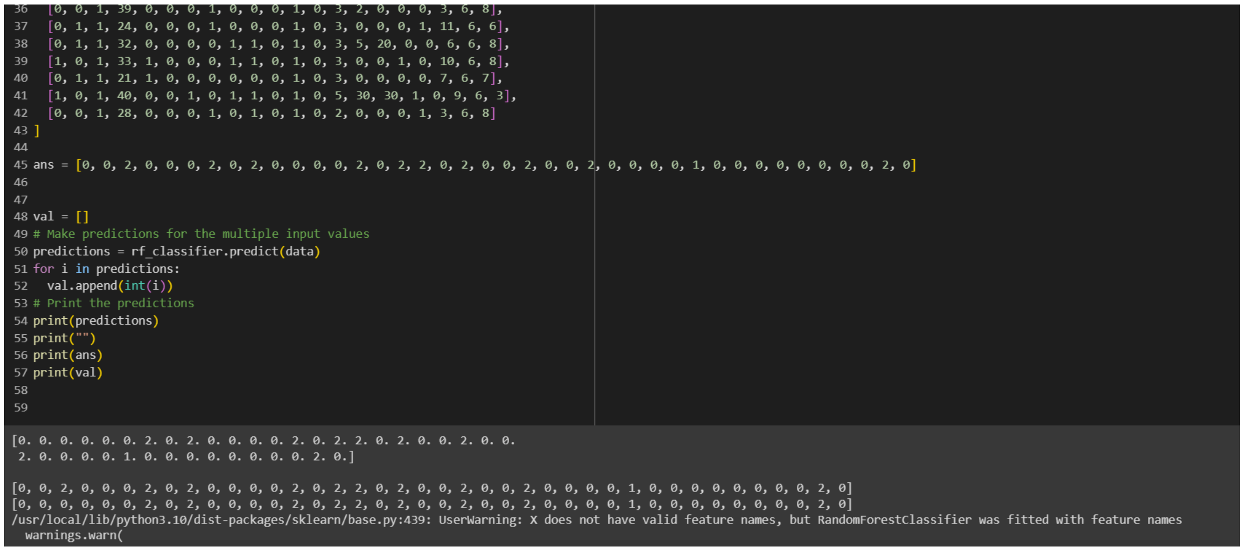
```
ecision Tree Classifier Accuracy: 0.8684001185443996

ecision Tree Classifier Confusion Matrix:
[36359   913  5416]
[  726 39960  1990]
[ 4780  3049 35029]]

ecision Tree Classifier Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.85      0.86     42688
         1.0       0.91      0.94      0.92     42676
         2.0       0.83      0.82      0.82     42858

    accuracy                           0.87    128222
   macro avg       0.87      0.87      0.87    128222
eighted avg       0.87      0.87      0.87    128222
```

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



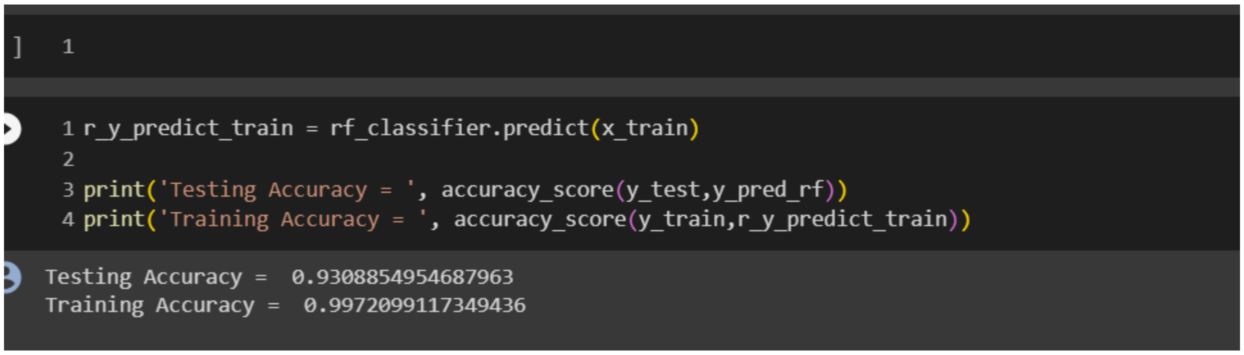
```
data = [
  [0, 0, 1, 40, 0, 0, 0, 0, 1, 0, 0, 1, 0, 3, 0, 10, 0, 0, 8, 4, 3],
  [0, 1, 1, 26, 0, 0, 0, 0, 1, 1, 0, 1, 0, 3, 0, 0, 0, 1, 13, 5, 5],
  [1, 1, 1, 29, 1, 0, 0, 1, 1, 1, 0, 1, 0, 3, 4, 4, 1, 1, 10, 6, 6],
  [1, 0, 1, 26, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 10, 5, 5],
  [1, 1, 1, 27, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 6, 0, 1, 13, 6, 7],
  [1, 0, 1, 21, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 13, 5, 4],
  [1, 1, 1, 37, 1, 0, 0, 0, 0, 1, 0, 1, 0, 5, 0, 30, 1, 1, 10, 5, 7],
  [1, 1, 1, 32, 0, 0, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 1, 9, 4, 6],
  [1, 1, 1, 34, 0, 0, 0, 0, 0, 1, 0, 1, 0, 3, 0, 0, 0, 1, 6, 5, 7],
  [0, 0, 1, 29, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 5, 6, 8],
  [1, 0, 1, 33, 1, 0, 0, 1, 1, 1, 0, 1, 0, 3, 0, 2, 1, 0, 12, 5, 5],
  [1, 0, 1, 39, 1, 0, 0, 1, 0, 1, 0, 1, 0, 3, 0, 0, 0, 0, 8, 5, 5],
  [1, 1, 1, 30, 1, 0, 0, 0, 1, 1, 0, 1, 1, 4, 10, 0, 0, 1, 7, 6, 5],
  [1, 0, 1, 26, 0, 0, 0, 1, 0, 1, 0, 1, 0, 3, 0, 0, 0, 1, 9, 6, 8],
  [1, 1, 1, 24, 0, 0, 0, 1, 1, 1, 0, 1, 0, 3, 0, 0, 0, 0, 11, 6, 7],
  [1, 0, 1, 39, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 4, 0, 1, 10, 5, 4],
  [1, 1, 1, 39, 1, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0, 3, 1, 1, 12, 5, 5],
  [0, 1, 1, 19, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 8, 4, 8],
  [1, 1, 1, 38, 1, 0, 1, 1, 0, 0, 0, 1, 0, 4, 10, 15, 1, 0, 11, 5, 4],
  [0, 0, 1, 27, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 8, 5, 8],
  [0, 1, 1, 32, 0, 0, 0, 1, 1, 0, 1, 1, 1, 3, 5, 0, 1, 1, 5, 6, 5],
  [1, 1, 1, 31, 0, 0, 0, 0, 1, 1, 0, 1, 0, 3, 0, 0, 0, 0, 10, 6, 7],
  [0, 0, 1, 20, 1, 0, 0, 1, 1, 1, 0, 1, 0, 3, 12, 30, 0, 0, 5, 5, 4],
  [0, 0, 1, 27, 1, 0, 0, 1, 1, 1, 0, 1, 0, 2, 0, 0, 0, 0, 13, 5, 6],
  [0, 0, 1, 36, 1, 0, 0, 1, 0, 1, 0, 1, 0, 4, 20, 5, 0, 1, 6, 6, 5],
  [1, 1, 1, 20, 1, 0, 0, 1, 1, 1, 0, 1, 0, 3, 15, 0, 1, 0, 13, 6, 6],
  [0, 0, 1, 25, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 7, 6, 8],
  [0, 1, 1, 28, 1, 0, 0, 1, 0, 1, 0, 1, 0, 3, 0, 0, 0, 0, 7, 6, 8],
```



```
  [0, 0, 1, 39, 0, 0, 0, 1, 0, 0, 0, 1, 0, 3, 2, 0, 0, 0, 3, 6, 8],
  [0, 1, 1, 24, 0, 0, 0, 1, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 11, 6, 6],
  [0, 1, 1, 32, 0, 0, 0, 0, 1, 1, 0, 1, 0, 3, 5, 20, 0, 0, 6, 6, 8],
  [1, 0, 1, 33, 1, 0, 0, 0, 1, 1, 0, 1, 0, 3, 0, 0, 1, 0, 10, 6, 8],
  [0, 1, 1, 21, 1, 0, 0, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 0, 7, 6, 7],
  [1, 0, 1, 40, 0, 0, 1, 0, 1, 1, 0, 1, 0, 5, 30, 30, 1, 0, 9, 6, 3],
  [0, 0, 1, 28, 0, 0, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 1, 3, 6, 8]
]

ans = [0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]


val = []
# Make predictions for the multiple input values
predictions = rf_classifier.predict(data)
for i in predictions:
  val.append(int(i))
# Print the predictions
print(predictions)
print("")
print(ans)
print(val)


```

```
[0. 0. 0. 0. 0. 0. 2. 0. 2. 0. 0. 0. 0. 2. 0. 2. 2. 0. 2. 0. 0. 2. 0. 0.
 2. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 2. 0.]

[0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

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)



```
]  1
```

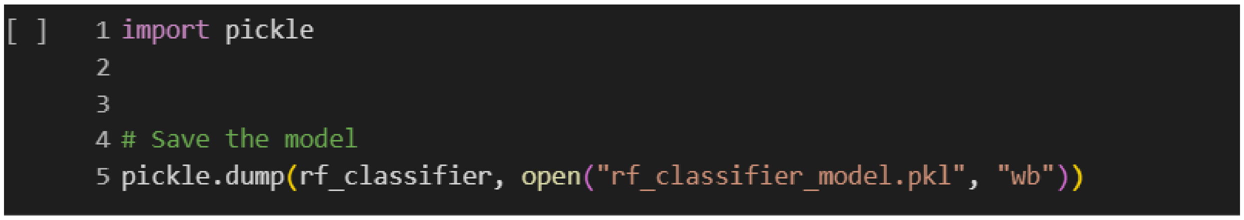
```
r_y_predict_train = rf_classifier.predict(x_train)

print('Testing Accuracy = ', accuracy_score(y_test,y_pred_rf))
print('Training Accuracy = ', accuracy_score(y_train,r_y_predict_train))
```

```
Testing Accuracy =  0.9308854954687963
Training Accuracy =  0.9972099117349436
```

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.



```
[ ]
import pickle


# Save the model
pickle.dump(rf_classifier, open("rf_classifier_model.pkl", "wb"))
```

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

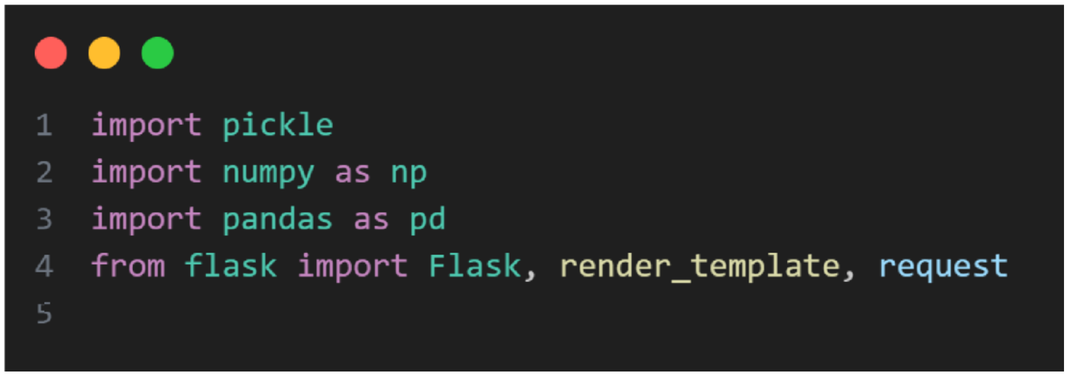
We create these html file

- home.html
- index.html
- diabetic.html
- prediction1.html
- prediction2.html
- prediction2.html
- prediction3.html

and save them in the folder named templates.

Activity 2.2: Build Python code:

Import the libraries in python file

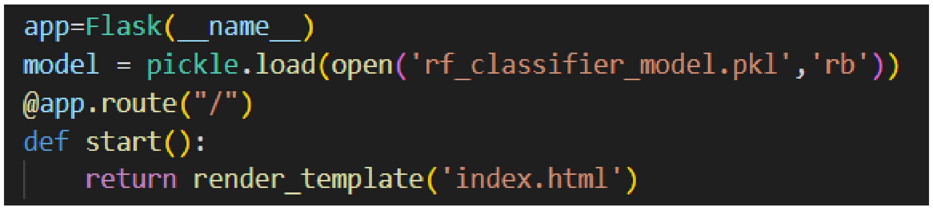


```python
import pickle
import numpy as np
import pandas as pd
from flask import Flask, render_template, request

```

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:

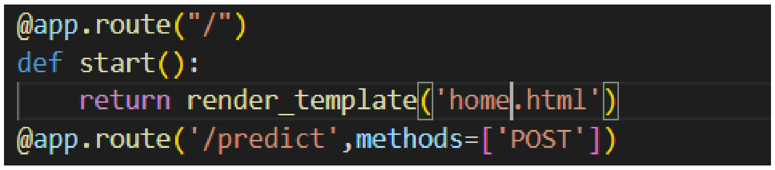
Here we will be using a declared constructor to route to the HTML page which we have created earlier.



```python
app=Flask(__name__)
model = pickle.load(open('rf_classifier_model.pkl','rb'))
@app.route("/")
def start():
    return render_template('index.html')
```

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.

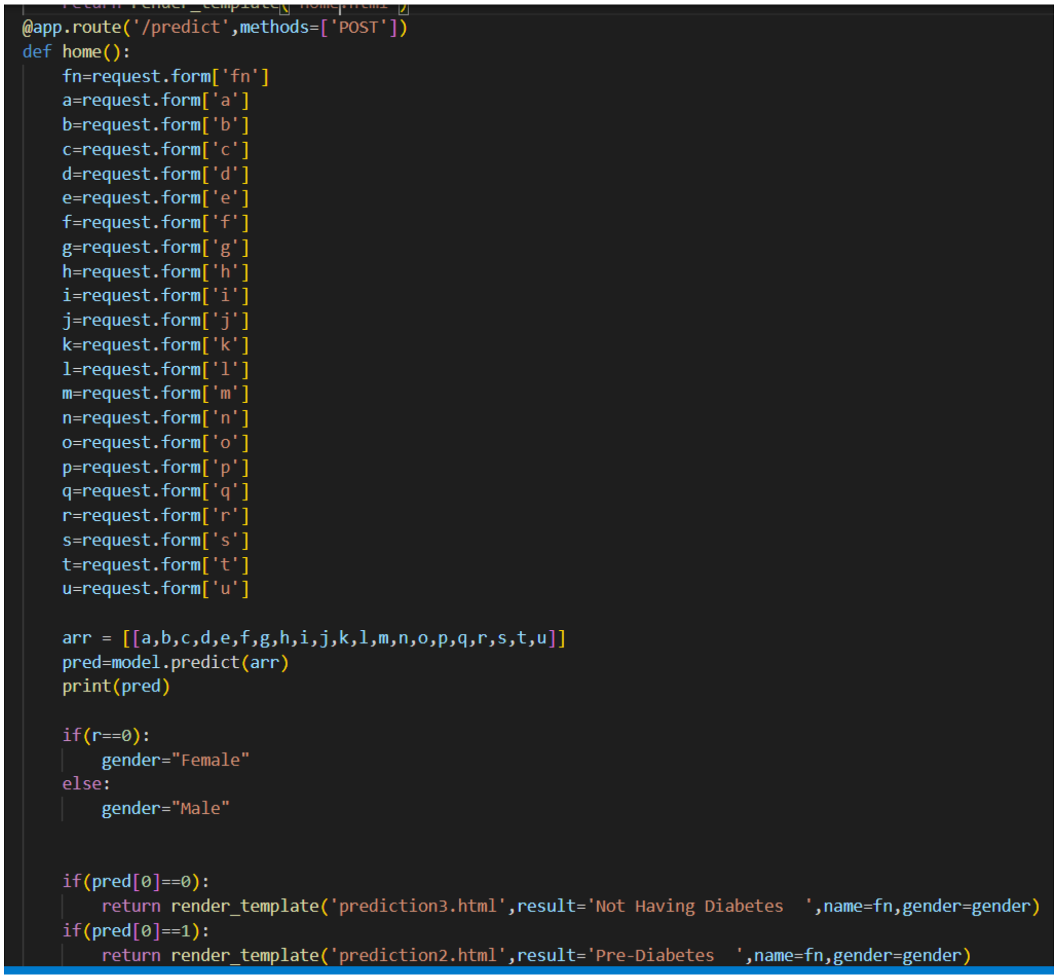
Retrieves the value from UI:



```python
@app.route("/")
def start():
    return render_template('home.html')
@app.route('/predict',methods=['POST'])
```

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 index.html page earlier.

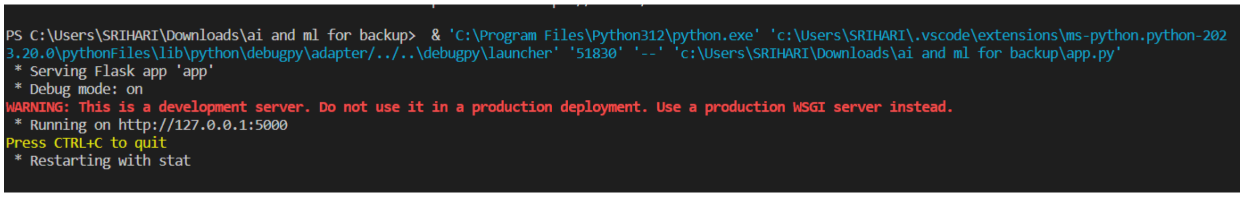
Main Function:



```python
@app.route('/predict',methods=['POST'])
def home():
    fn=request.form['fn']
    a=request.form['a']
    b=request.form['b']
    c=request.form['c']
    d=request.form['d']
    e=request.form['e']
    f=request.form['f']
    g=request.form['g']
    h=request.form['h']
    i=request.form['i']
    j=request.form['j']
    k=request.form['k']
    l=request.form['l']
    m=request.form['m']
    n=request.form['n']
    o=request.form['o']
    p=request.form['p']
    q=request.form['q']
    r=request.form['r']
    s=request.form['s']
    t=request.form['t']
    u=request.form['u']

    arr = [[a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u]]
    pred=model.predict(arr)
    print(pred)

    if(r==0):
        gender="Female"
    else:
        gender="Male"


    if(pred[0]==0):
        return render_template('prediction3.html',result='Not Having Diabetes  ',name=fn,gender=gender)
    if(pred[0]==1):
        return render_template('prediction2.html',result='Pre-Diabetes  ',name=fn,gender=gender)
```

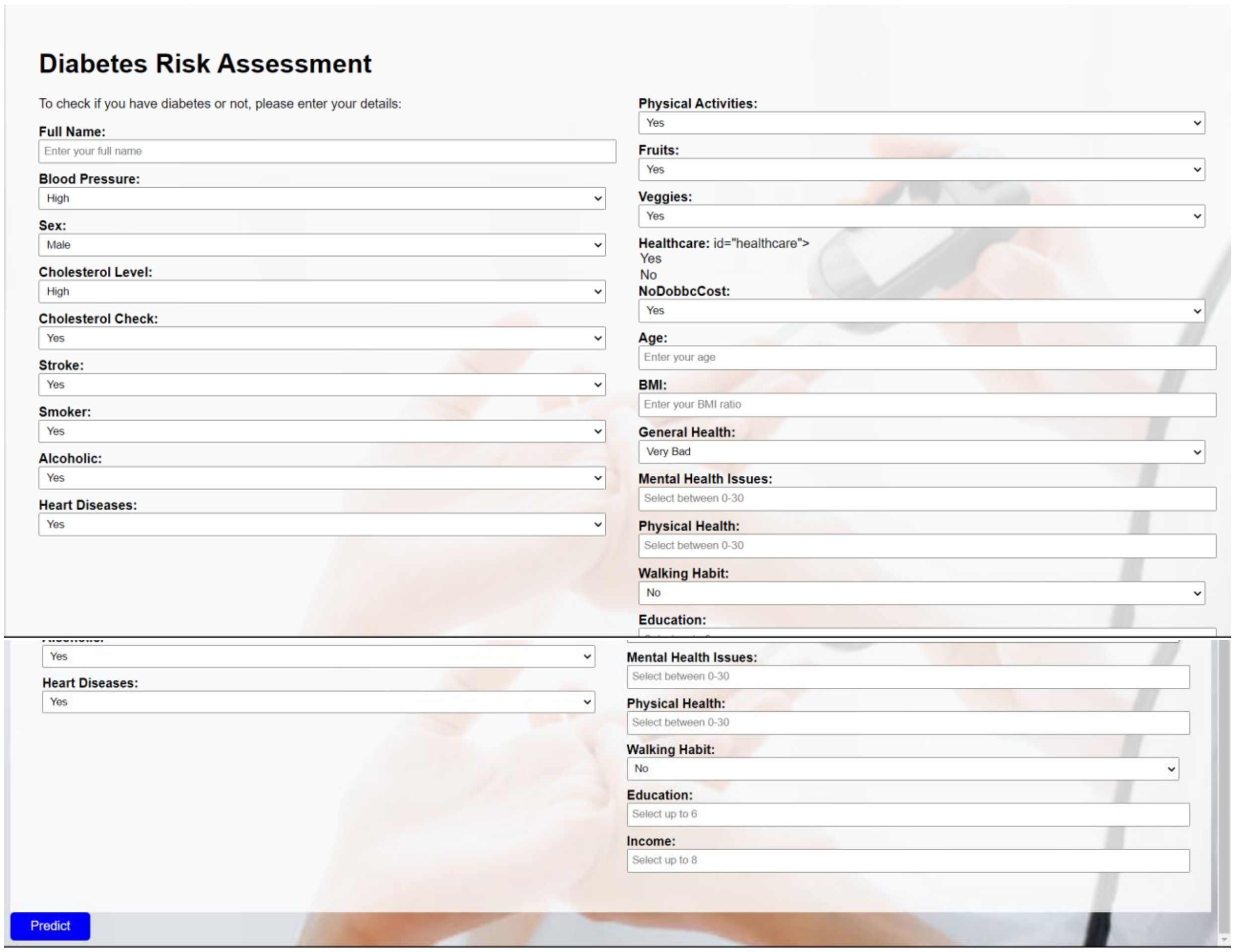
Activity 2.3: Run the web application

- 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.



```
PS C:\Users\SRIHARI\Downloads\ai and ml for backup>  & 'C:\Program Files\Python312\python.exe' 'c:\Users\SRIHARI\.vscode\extensions\ms-python.python-202
3.20.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '51830' '--' 'c:\Users\SRIHARI\Downloads\ai and ml for backup\app.py'
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
```

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



Diabetes Risk Assessment

To check if you have diabetes or not, please enter your details:

Full Name: Enter your full name

Blood Pressure: High

Sex: Male

Cholesterol Level: High

Cholesterol Check: Yes

Stroke: Yes

Smoker: Yes

Alcoholic: Yes

Heart Diseases: Yes

Physical Activities: Yes

Fruits: Yes

Veggies: Yes

Healthcare: id="healthcare">
Yes
No

NoDobbcCost: Yes

Age: Enter your age

BMI: Enter your BMI ratio

General Health: Very Bad

Mental Health Issues: Select between 0-30

Physical Health: Select between 0-30

Walking Habit: No

Education:

Yes

Heart Diseases: Yes

Mental Health Issues: Select between 0-30

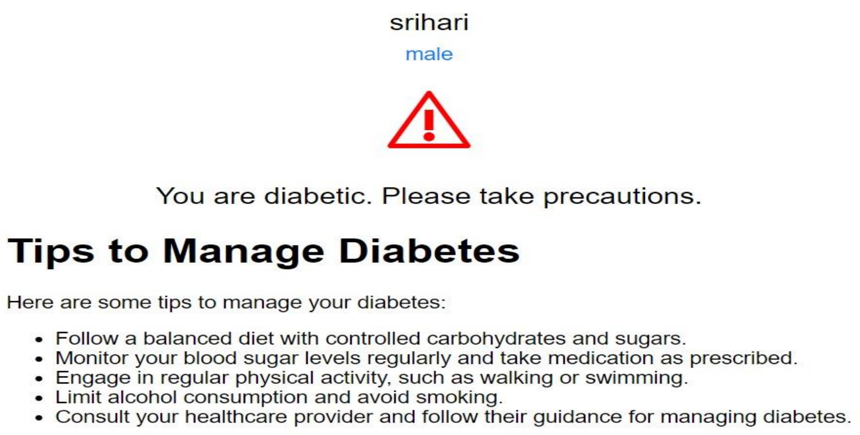
Physical Health: Select between 0-30

Walking Habit: No

Education: Select up to 6

Income: Select up to 8

Predict



srihari

male

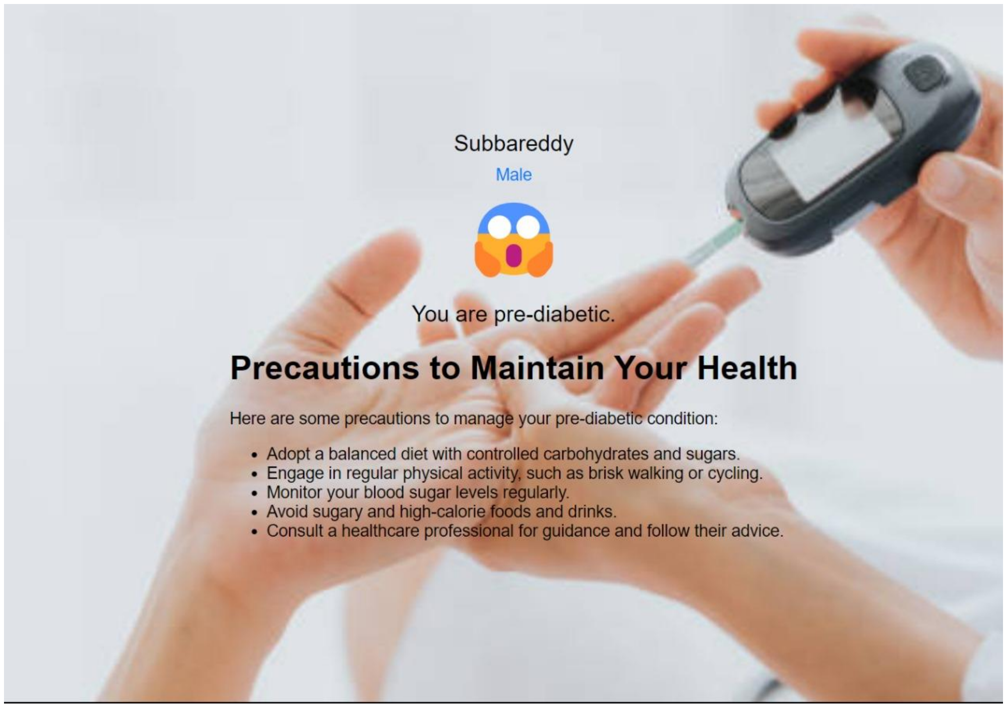
⚠

You are diabetic. Please take precautions.

Tips to Manage Diabetes

Here are some tips to manage your diabetes:

- Follow a balanced diet with controlled carbohydrates and sugars.
- Monitor your blood sugar levels regularly and take medication as prescribed.
- Engage in regular physical activity, such as walking or swimming.
- Limit alcohol consumption and avoid smoking.
- Consult your healthcare provider and follow their guidance for managing diabetes.



Subbareddy

Male

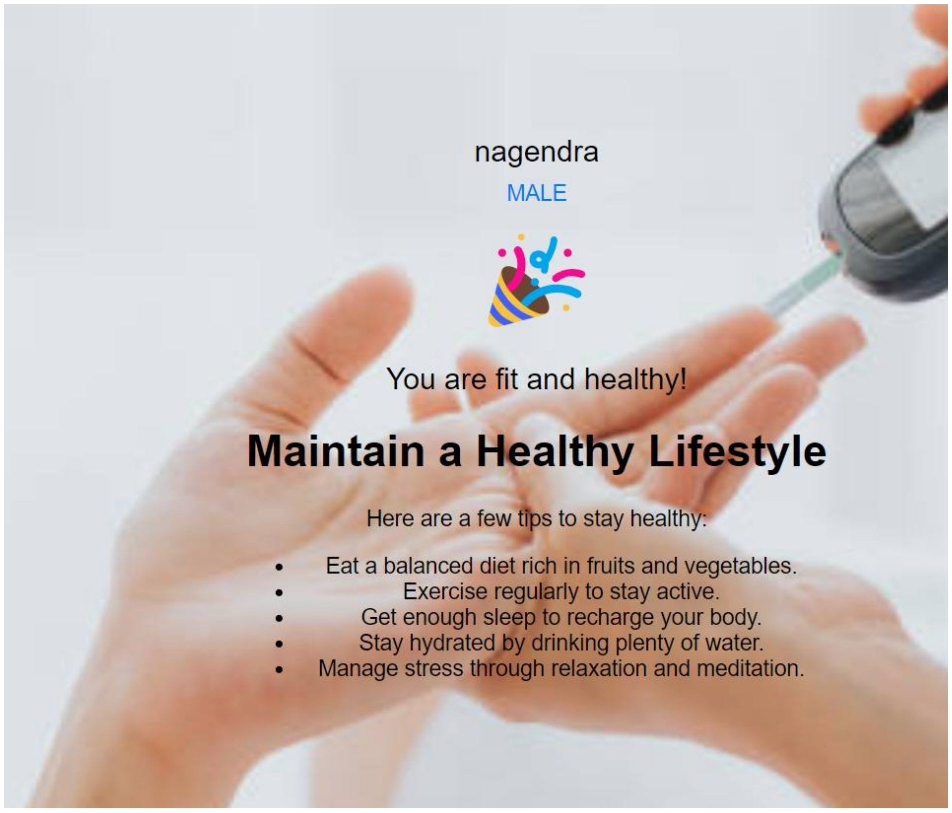
😱

You are pre-diabetic.

Precautions to Maintain Your Health

Here are some precautions to manage your pre-diabetic condition:

- Adopt a balanced diet with controlled carbohydrates and sugars.
- Engage in regular physical activity, such as brisk walking or cycling.
- Monitor your blood sugar levels regularly.
- Avoid sugary and high-calorie foods and drinks.
- Consult a healthcare professional for guidance and follow their advice.



nagendra

MALE

🎉

You are fit and healthy!

Maintain a Healthy Lifestyle

Here are a few tips to stay healthy:

- Eat a balanced diet rich in fruits and vegetables.
- Exercise regularly to stay active.
- Get enough sleep to recharge your body.
- Stay hydrated by drinking plenty of water.
- Manage stress through relaxation and meditation.