Title:

Diabetes Prediction Using Machine Learning

Name: Bommaraju Saketh Ram

Team ID : Team-591879

Title: Diabetes Prediction Using Machine Learning

Diabetes prediction using machine learning is a groundbreaking application that leverages advanced algorithms to analyze and interpret complex data, aiding in the early identification of individuals at risk of developing diabetes. This innovative approach combines the power of data science with medical insights to create predictive models capable of assessing an individual's susceptibility to diabetes with high accuracy.

Machine learning algorithms, such as logistic regression, decision trees, and support vector machines, are employed to analyze diverse sets of data, including demographic information, lifestyle factors, genetic predispositions, and historical health records. These algorithms learn from patterns within the data, enabling them to make predictions based on new, unseen information.

The predictive models take into account various risk factors associated with diabetes, such as age, body mass index (BMI), family history, physical activity, and dietary habits. By inputting relevant information into the system, the model generates personalized risk scores, allowing healthcare professionals to identify individuals who may be at an elevated risk of developing diabetes in the future.

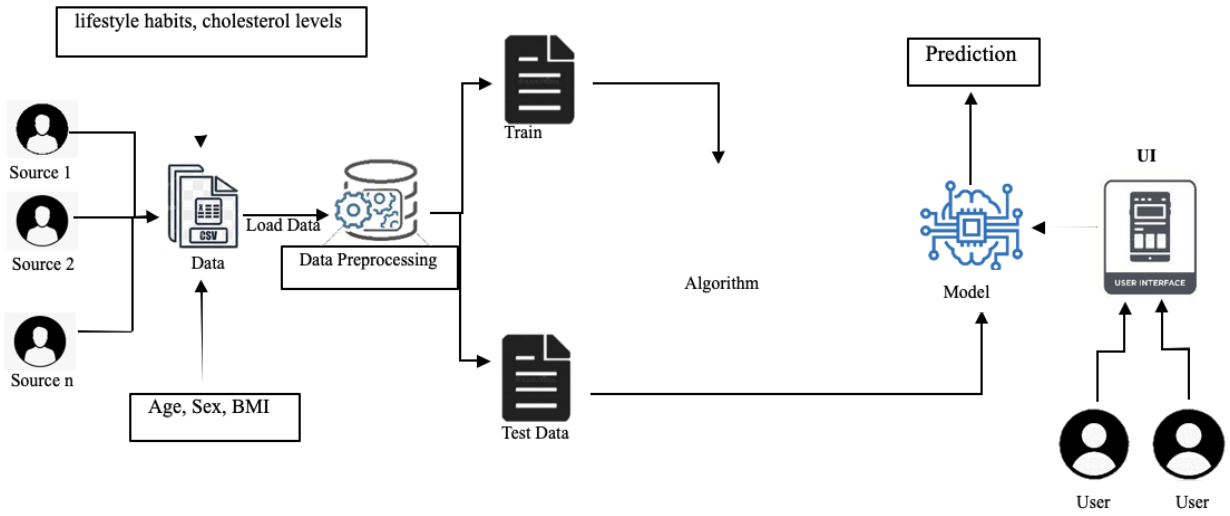
This proactive approach to diabetes prediction holds significant promise in preventive healthcare. Early identification of at-risk individuals enables timely interventions, such as

lifestyle modifications, targeted education, and personalized healthcare plans. This not only helps in preventing the onset of diabetes but also in reducing the overall burden on healthcare systems by focusing on proactive and preventive measures.

Moreover, the integration of machine learning in diabetes prediction facilitates continuous improvement and refinement of models over time. As more data becomes available and the models learn from real-world outcomes, their predictive accuracy and effectiveness are enhanced, making them valuable tools in the ongoing battle against diabetes.

In summary, diabetes prediction using machine learning represents a cutting-edge solution that harnesses the power of data-driven insights to identify and address diabetes risk factors early on. By blending technology with medical expertise, this approach holds the potential to revolutionize healthcare by enabling proactive, personalized, and effective interventions in the fight against diabetes.

Data flow Diagram:



Project flow:

Building a Diabetes Prediction App using Machine Learning involves several key steps. Below is a simplified project flow to guide you through the process:

1. Problem Definition and Data Collection:
   - Clearly define the problem you want to address: predicting the risk of diabetes.
   - Gather relevant data, including demographic information, lifestyle factors, genetic predispositions, and health records.

2. Data Preprocessing:
   - Handle missing data and outliers.
   - Normalize or standardize numerical features.
   - Encode categorical variables.
   - Split the dataset into training and testing sets.

3. Exploratory Data Analysis (EDA):
    - Perform exploratory analysis to understand the distribution of features and relationships within the data.
   - Visualize key statistics and patterns.

4. Feature Selection:
   - Identify and select the most relevant features for building the predictive model.
   - Consider techniques like correlation analysis and feature importance from machine learning models.

5. Model Selection:
   - Choose appropriate machine learning algorithms for classification, such as logistic regression, decision trees, or support vector machines.
   - Train multiple models and evaluate their performance on the training set.

6. Model Training:
   - Train the selected model on the training dataset.
   - Optimize hyperparameters to improve model performance.

7. Model Evaluation
   - Evaluate the model using the testing dataset to ensure its generalization to new, unseen data.
   - Use metrics like accuracy, precision, recall, and F1-score to assess performance.

8. Deployment:
   - Develop a user-friendly interface for the app.
   - Integrate the trained model into the app.
   - Ensure the app can handle input data and return predictions in real-time.

9. User Interface (UI) Design:
   - Design an intuitive and user-friendly interface for users to input their information.
   - Provide clear instructions and feedback on the prediction results.

10. Testing:
    - Conduct thorough testing of the app to identify and fix any bugs or issues.
    - Perform end-to-end testing to validate the functionality and accuracy of the prediction model within the app.

11. Deployment and Maintenance:
   - Deploy the app to a platform of choice.
   - Implement regular updates and maintenance to keep the app running smoothly.
   - Monitor the app's performance and address any issues that may arise.

12. Education and User Support:
   - Provide educational resources within the app to help users understand their risk factors and potential preventive measures.
   - Establish a support system for users who may have questions or concerns.

Prior Knowledge:

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

- ML Concepts

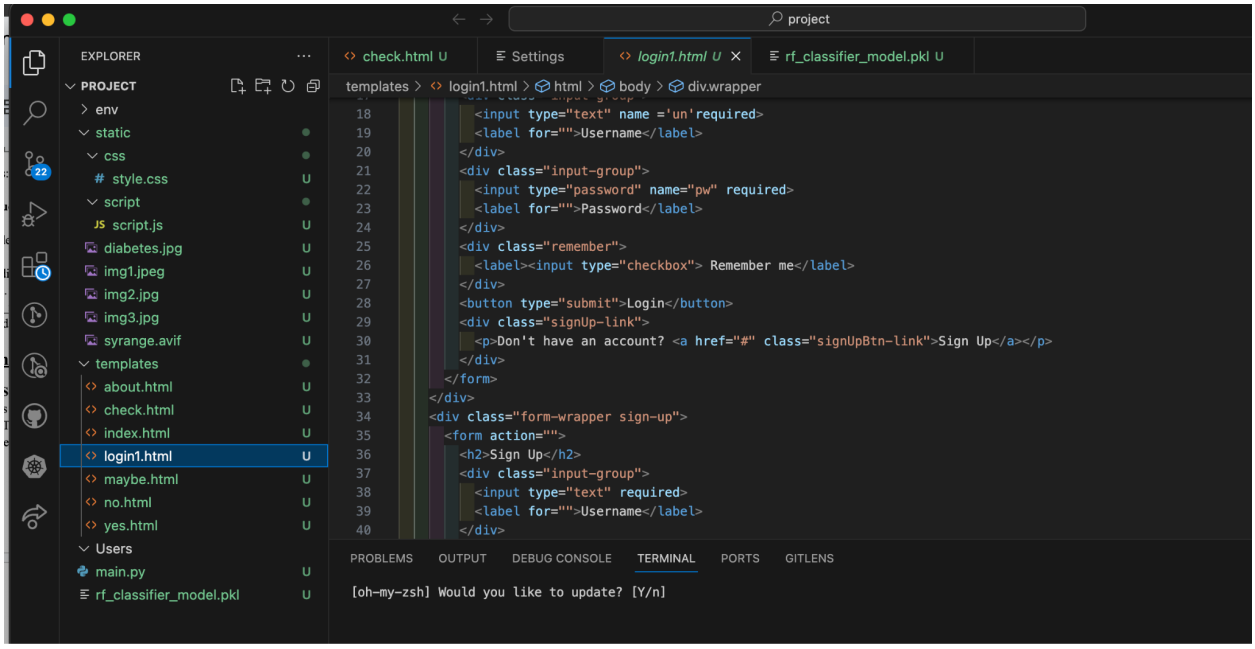
  Supervised learning: https://www.javatpoint.com/supervised-machine-learning Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning ● Decision tree: https://www.javatpoint.com/machine-learning-decision-tree- classificationalgorithm

- 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_processin g_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 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.

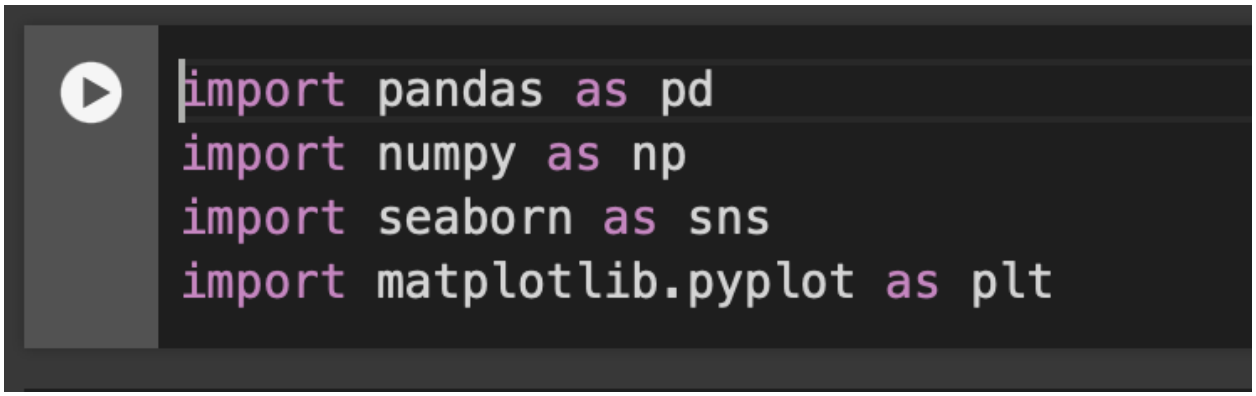
Link: Diabetes Health Indicators Dataset | Kaggle

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

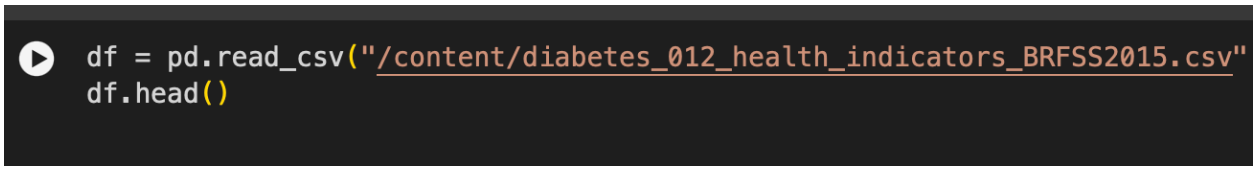
Import the necessary libraries as shown in the image.



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

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.

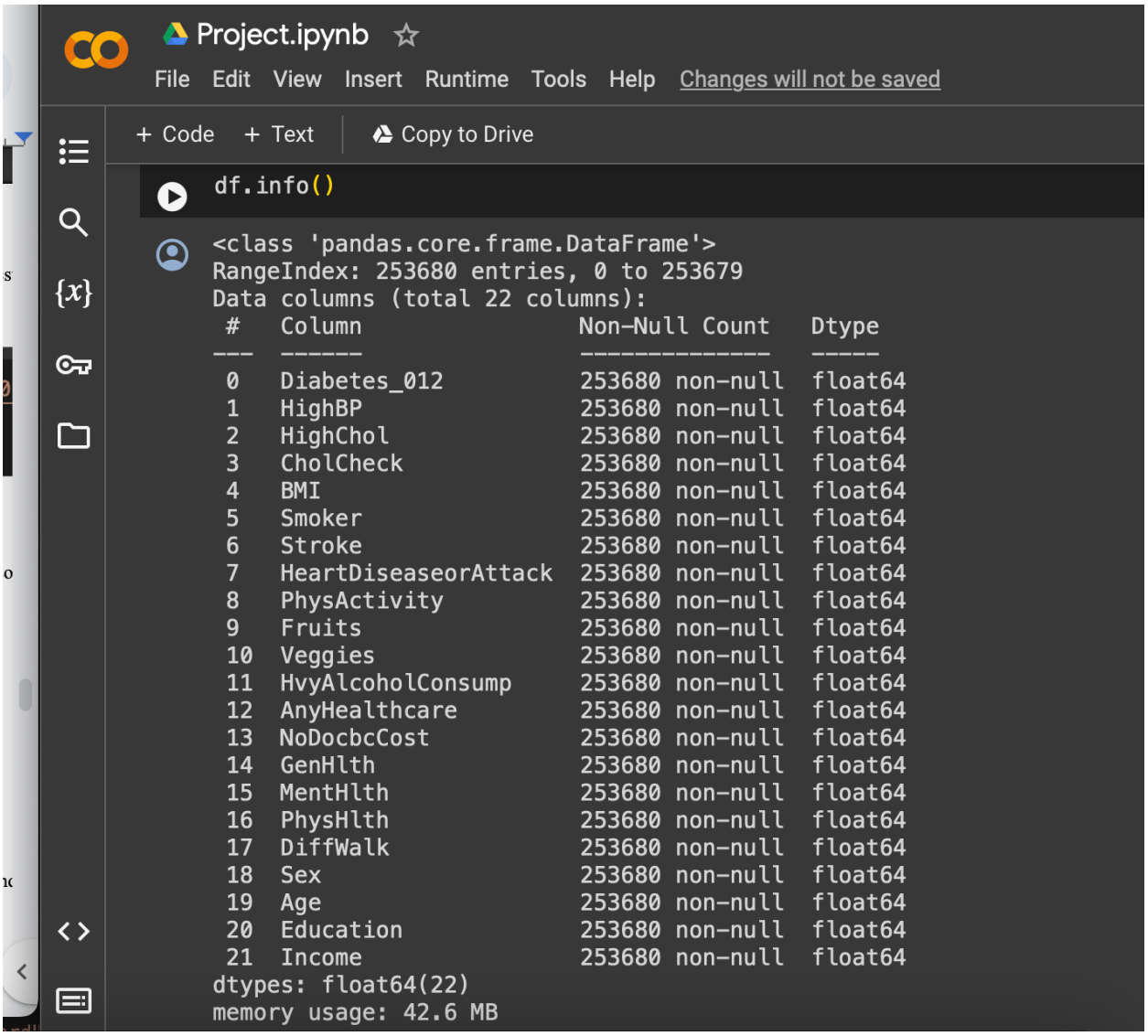
This activity includes the following steps.

- 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



- 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.

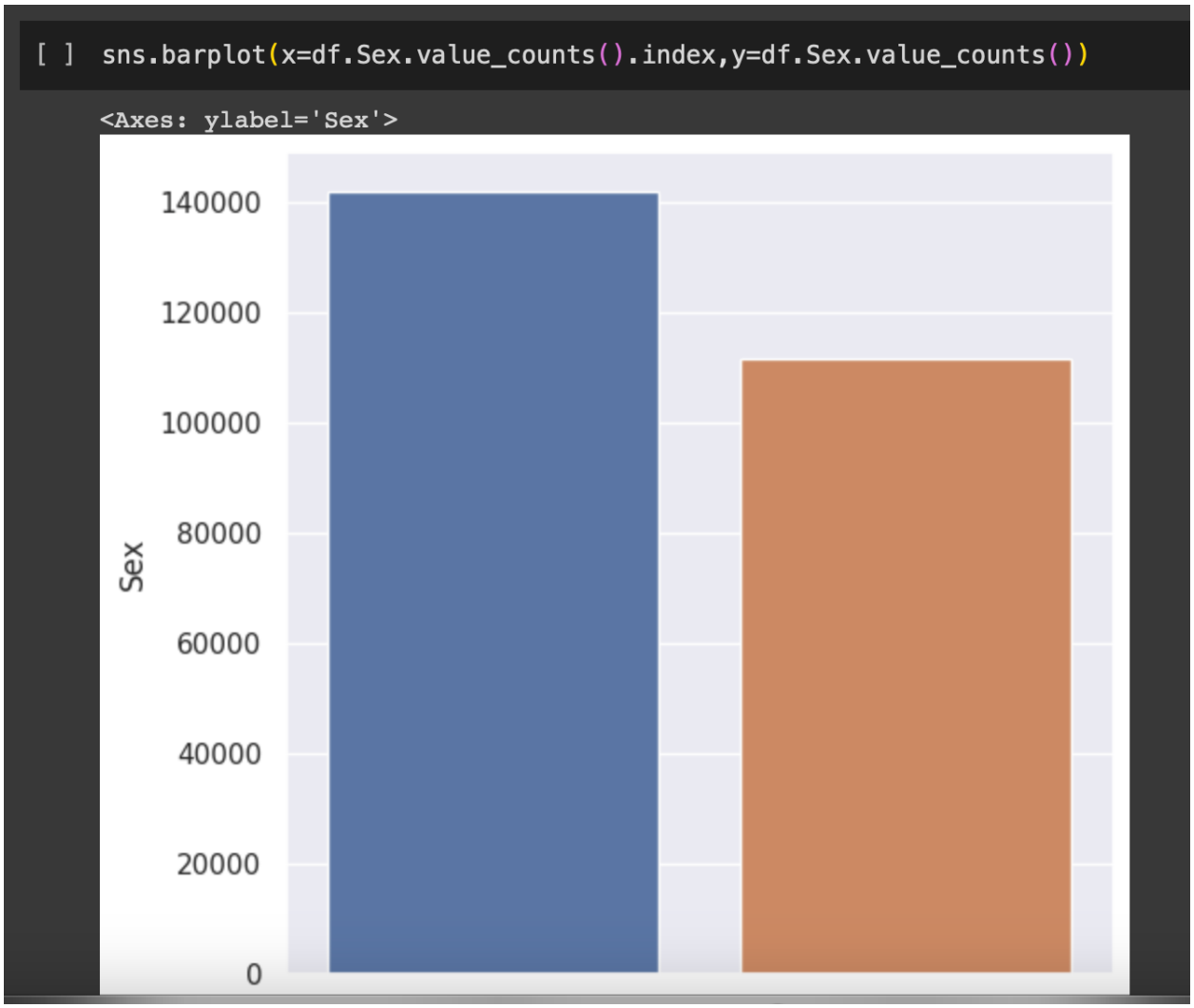
To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using Label encoding with the help of list comprehension.

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

Activity 2.3: Handling Imbalance Data

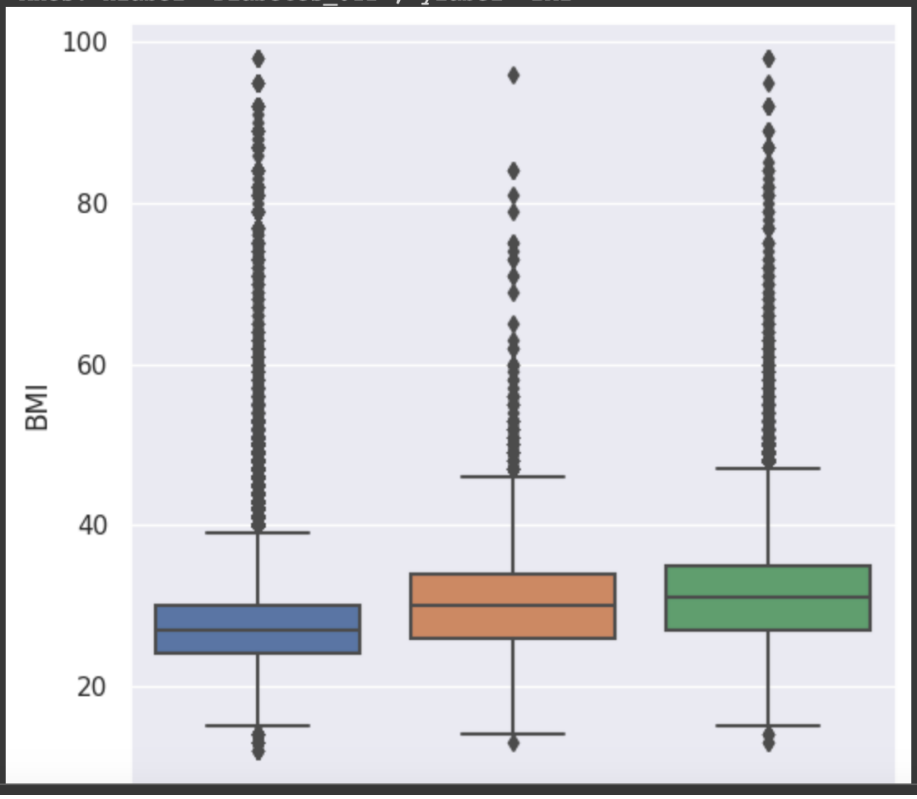
With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of some columns feature with some mathematical formula

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



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

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



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

.

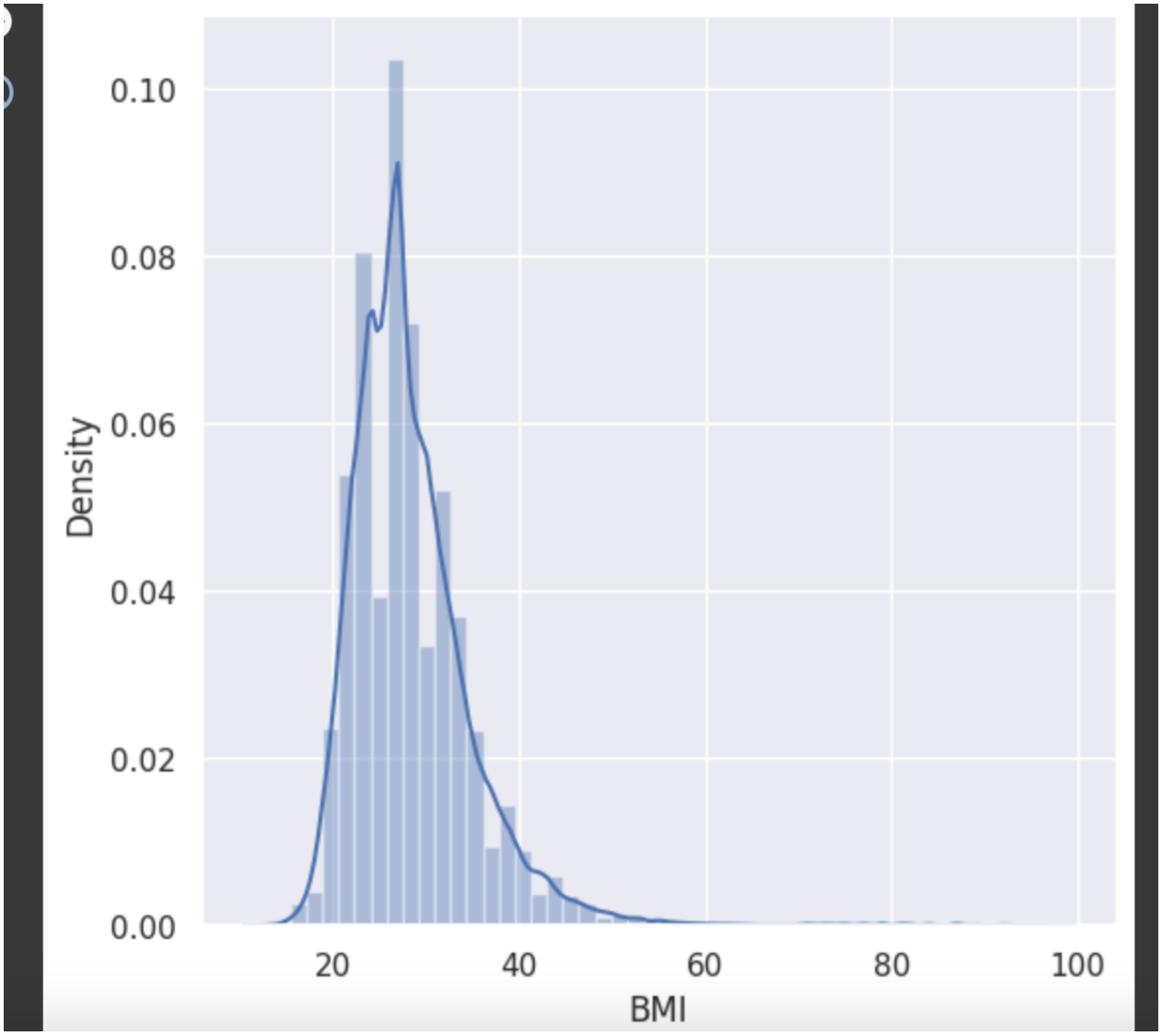
```
[ ] df.describe()
```

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAtta
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.0000
mean	0.296921	0.429001	0.424121	0.962670	28.382364	0.443169	0.040571	0.0941
std	0.698160	0.494934	0.494210	0.189571	6.608694	0.496761	0.197294	0.2920
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	0.0000
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	0.0000
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	0.0000
max	2.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.0000

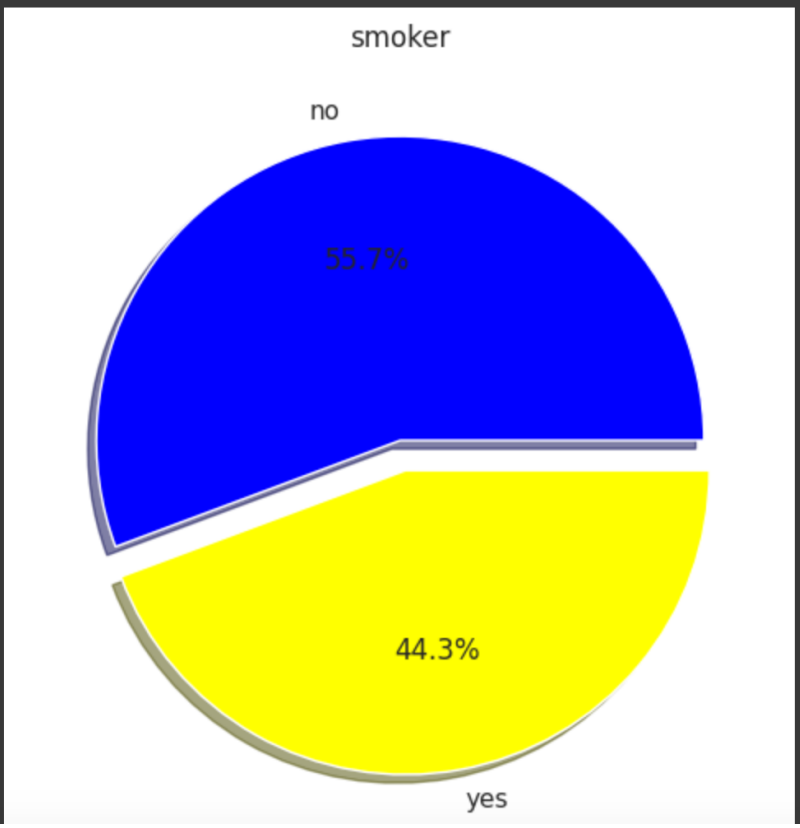
8 rows × 22 columns

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.



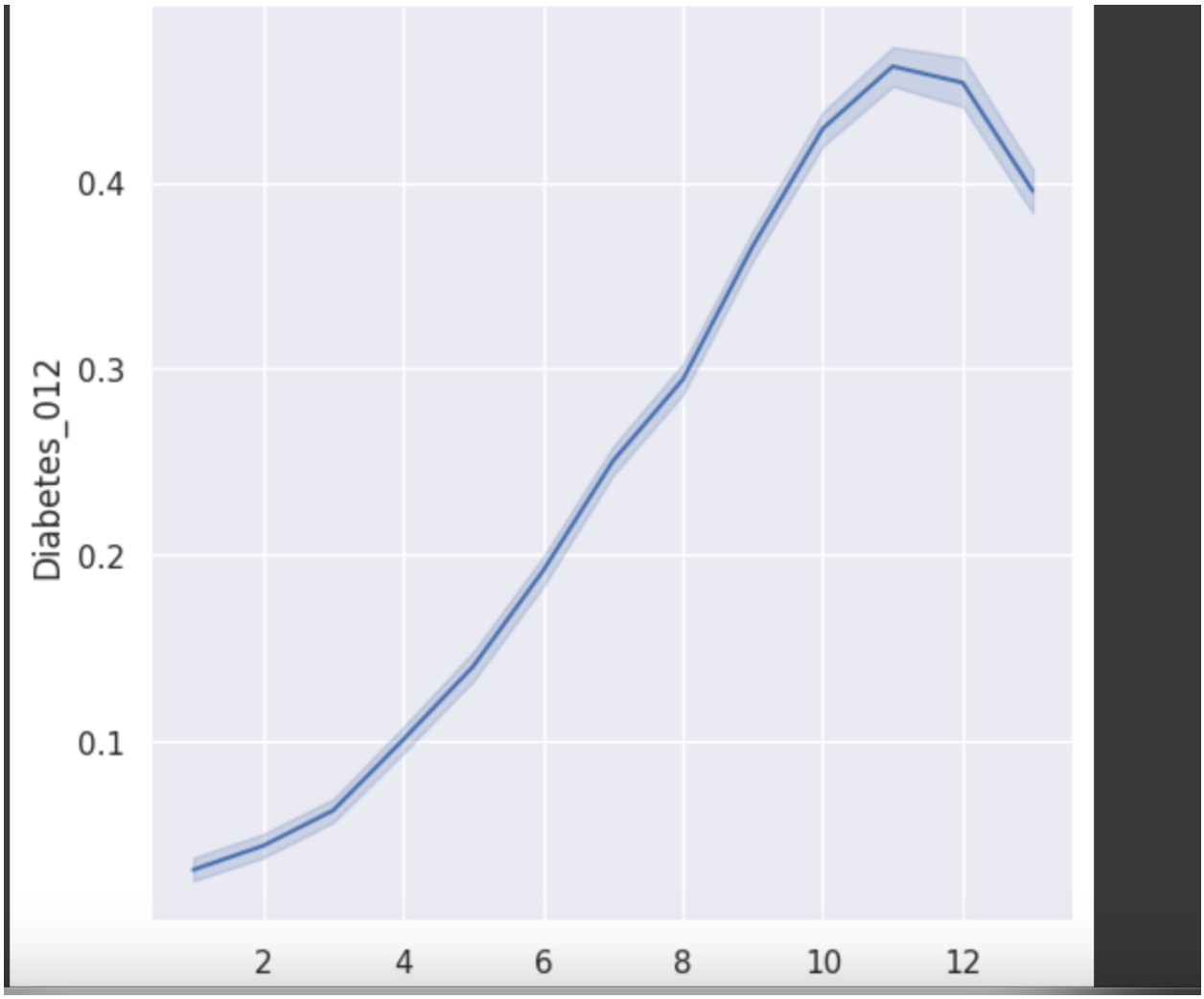
```
plt.pie(df.Smoker.value_counts(),[0,0.1],labels=["no","yes"],autopct='%1.1f%%', shadow =
plt.title("smoker")
plt.show()
```



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.

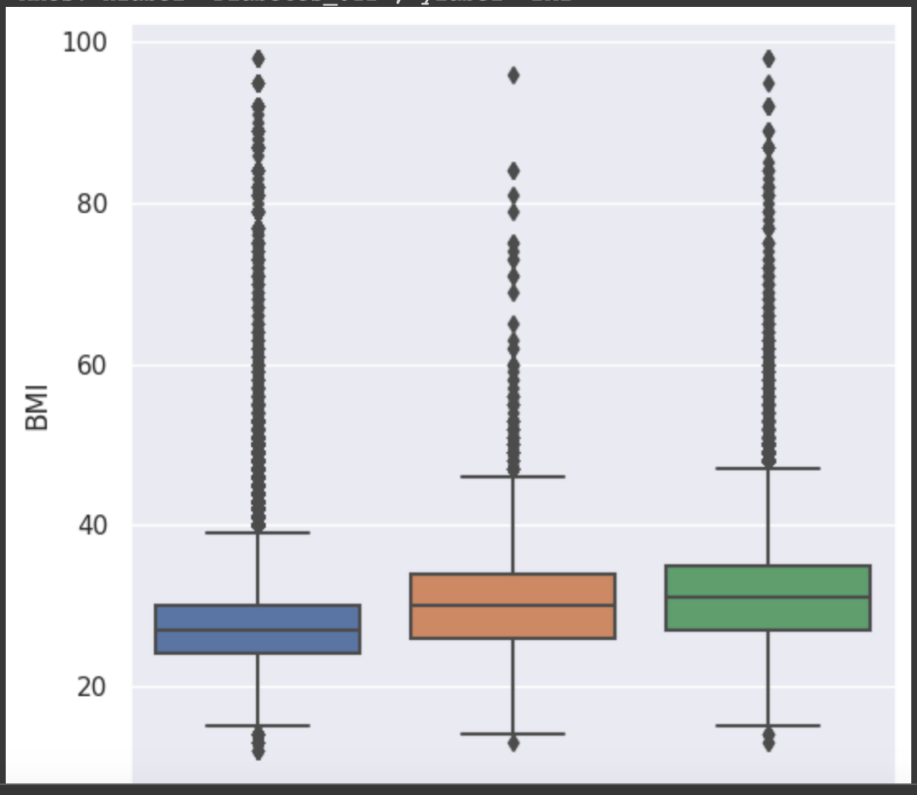


Activity 2.2: Bivariate analysis

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.

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)

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

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.

```python
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: 0.86840011185443996

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):
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
```

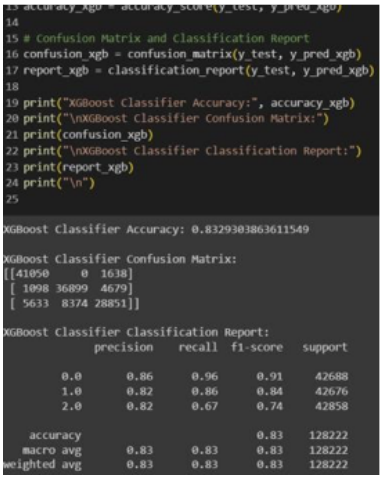
Activity 2: Testing the model

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

Milestone 5: Performance Testing

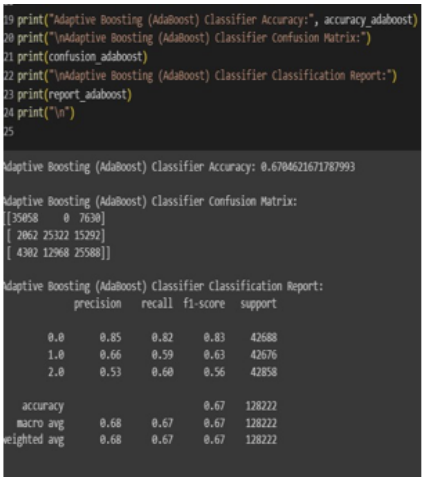
Activity 1: Testing model with multiple evaluation metrics

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.



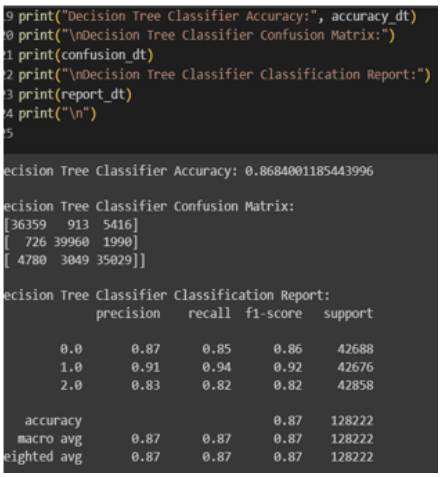
```
14
15 # Confusion Matrix and Classification Report
16 confusion_xgb = confusion_matrix(y_test, y_pred_xgb)
17 report_xgb = classification_report(y_test, y_pred_xgb)
18
19 print("XGBoost Classifier Accuracy:", accuracy_xgb)
20 print("\nXGBoost Classifier Confusion Matrix:")
21 print(confusion_xgb)
22 print("\nXGBoost Classifier Classification Report:")
23 print(report_xgb)
24 print("\n")
25
```

```
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
```



```
19 print("Adaptive Boosting (AdaBoost) Classifier Accuracy:", accuracy_adaboost)
20 print("\nAdaptive Boosting (AdaBoost) Classifier Confusion Matrix:")
21 print(confusion_adaboost)
22 print("\nAdaptive Boosting (AdaBoost) Classifier Classification Report:")
23 print(report_adaboost)
24 print("\n")
25
```

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

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

daptive 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
eighted avg       0.68      0.67      0.67    128222
```



```
9 print("Decision Tree Classifier Accuracy:", accuracy_dt)
0 print("\nDecision Tree Classifier Confusion Matrix:")
1 print(confusion_dt)
2 print("\nDecision Tree Classifier Classification Report:")
3 print(report_dt)
4 print("\n")
5
```

```
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, 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, 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, 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
```

```
1 r_y_predict_train = rf_classifier.predict(x_train)
2
3 print('Testing Accuracy = ', accuracy_score(y_test,y_pred_rf))
4 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.

```
[ ]  1 import pickle
     2
     3
     4 # Save the model
     5 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
from flask import Flask, render_template, request
import pickle
import numpy as np
import pandas as pd
from numpy.typing import NDArray

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



@app.route("/", methods=['GET', 'POST'])
def startone():
    if request.method == 'POST':
        un = request.form['un']
        pw = request.form['pw']
        if un == 'admin' and pw == 'admin':
            return render_template('index.html')
    return render_template('login1.html')

@app.route("/index.html")
def start():
    return render_template('index.html')
@app.route("/about.html")
def about():
    return render_template('about.html')

@app.route("/check.html")
def check():
    return render_template('check.html')
```

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.

```
app=Flask(__name__)
model = pickle.load(open('rf_classifier_model.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:

```
@app.route("/", methods=['GET', 'POST'])
def startone():
    if request.method == 'POST':
        un = request.form['un']
        pw = request.form['pw']
        if un == 'admin' and pw == 'admin':
            return render_template('index.html'
    return render_template('login1.html')
```

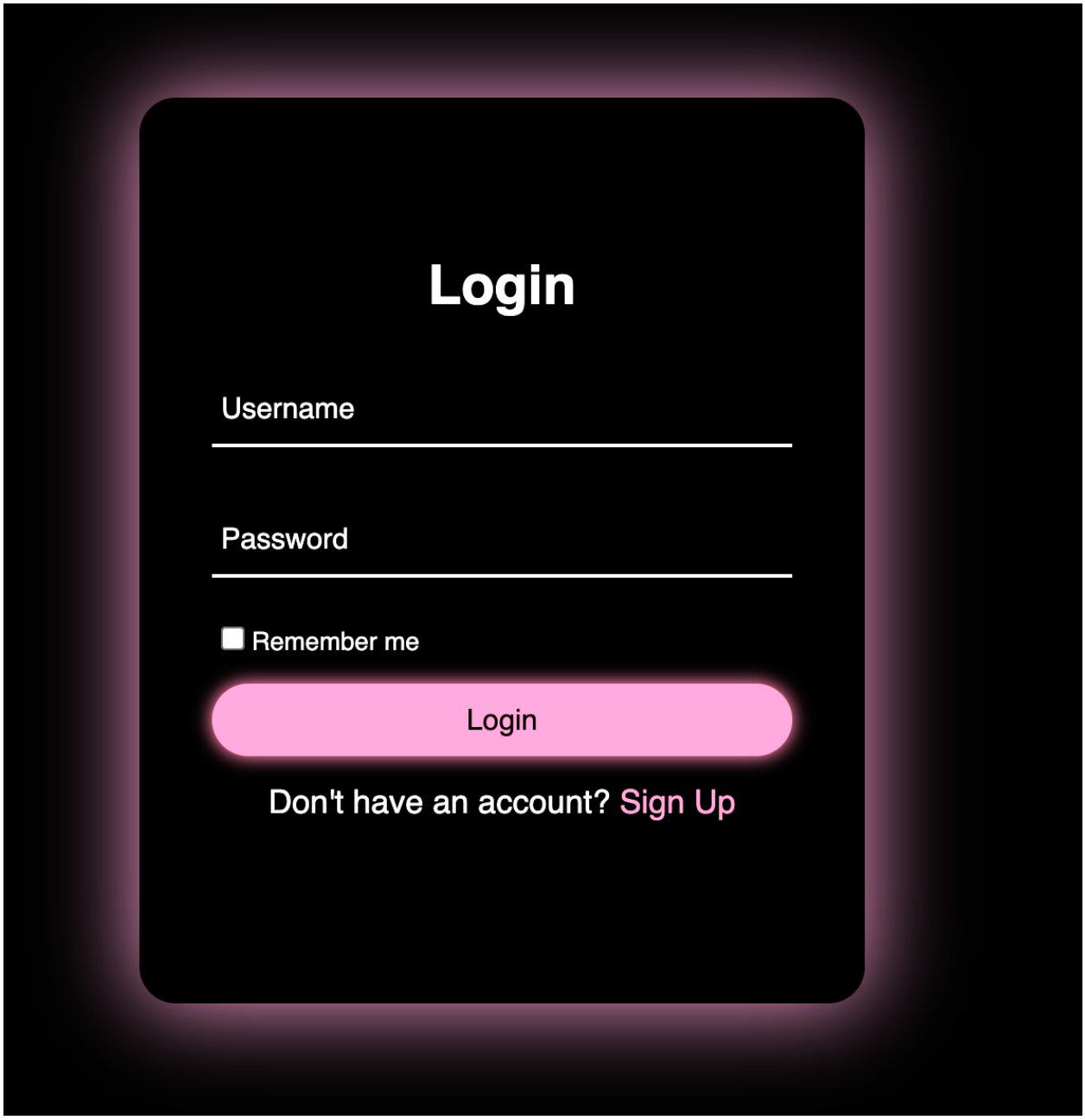
Here we are routing our app to prediction () function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model Predict() function. This function returns the prediction. And this prediction value will be rendered to the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

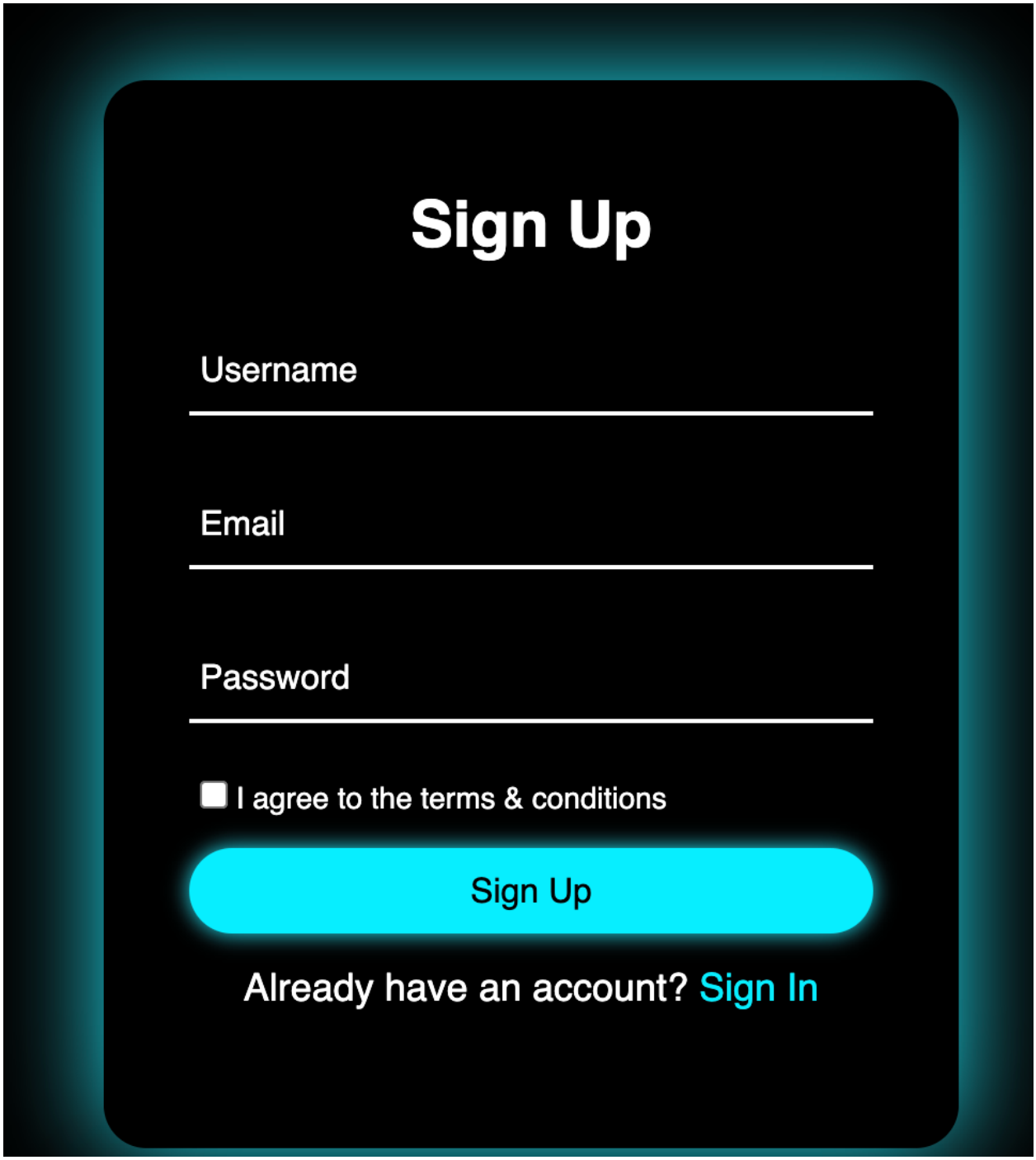
Main Function:

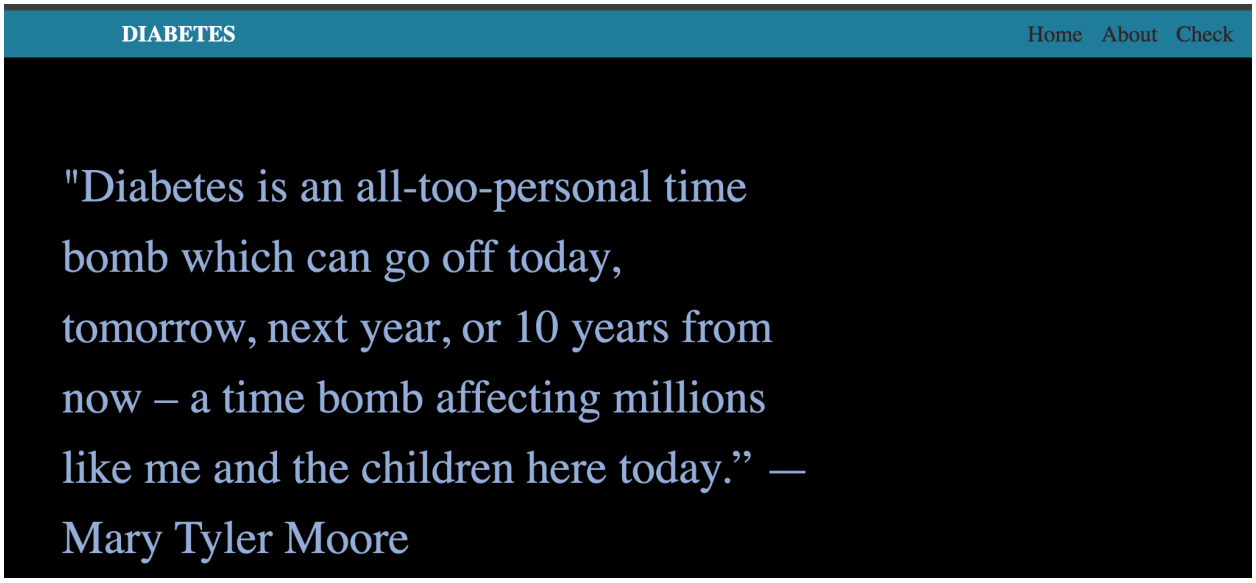
```
@app.route('/predict',methods=['POST'])
def home():
    fn=request.form['fn']
    a=request.form['a']
    b=request.form['b']
    c=request.form['c']
    d=request.form['d']
    e=request.form['e']
    f=request.form['f']
    g=request.form['g']
    h=request.form['h']
    i=request.form['i']
    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)
```

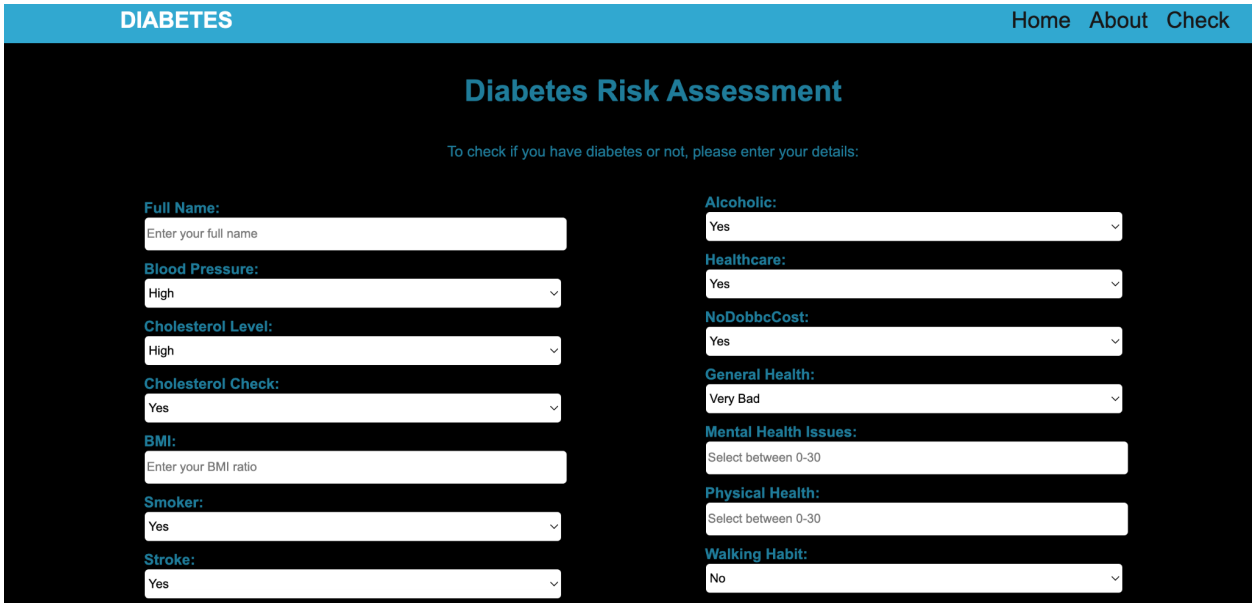
Activity 2.3: Run the web application

- 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.


Login
Username
Password
Remember me
Login
Don't have an account? Sign Up


Sign Up
Username
Email
Password
I agree to the terms & conditions
Sign Up
Already have an account? Sign In


DIABETES
Home About Check
"Diabetes is an all-too-personal time bomb which can go off today, tomorrow, next year, or 10 years from now – a time bomb affecting millions like me and the children here today." — Mary Tyler Moore


DIABETES
Home About Check
Diabetes Risk Assessment
To check if you have diabetes or not, please enter your details:
Full Name:
Enter your full name
Blood Pressure:
High
Cholesterol Level:
High
Cholesterol Check:
Yes
BMI:
Enter your BMI ratio
Smoker:
Yes
Stroke:
Yes
Alcoholic:
Yes
Healthcare:
Yes
NoDobbcCost:
Yes
General Health:
Very Bad
Mental Health Issues:
Select between 0-30
Physical Health:
Select between 0-30
Walking Habit:
No