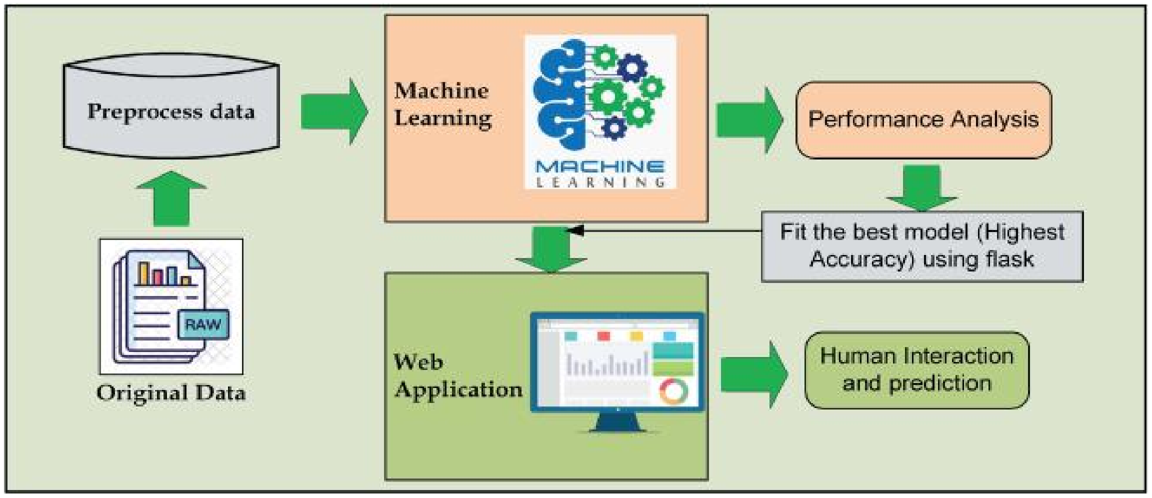
Project Manual

Date	08 October 2023
Team ID	Team-592607
Project	Diabetes Prediction Using Machine Learning
Marks	4 Marks

Team Leader : L.Mohith
Team Member: AS. Vikram Aditya
Team Member: K. Rami Reddy
Team Member: PVS. Uday Kiran

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 analyze 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.



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyzes the input the prediction is showcased on the UI

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

- Define Problem / Problem Understanding
    - Specify the business problem
    - Business requirements
    - Literature Survey
    - Social or Business Impact.

- Data Collection & Preparation
    - Collect the dataset
    - Data Preparation

- Exploratory Data Analysis
    - Descriptive statistical
    - Visual Analysis

- Model Building
    - Training the model in multiple algorithms
    - Testing the model

- Performance Testing
    - Testing model with multiple evaluation metrics

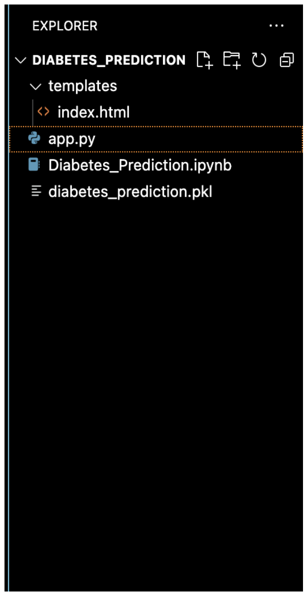
- Model Deployment
    - Save the best model
    - Integrate with Web Framework

Prior Knowledge:

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

- Random Forest
- Logistic Regression
- XGBClassifier

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.
● diabetes_prediction.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:

- Gauri D. Kalyankar, Shivananda R. Poojara and Nagaraj V. Dharwadkar,” Predictive Analysis of Diabetic Patient Data Using Machine Learning and Hadoop”, International Conference On I-SMAC, 978-1-5090-3243-3, 2017.

- Ayush Anand and Divya Shakti,” Prediction of Diabetes Based on Personal Lifestyle Indicators”, 1st International Conference on Next Generation Computing Technologies, 978-1-4673-6809-4, September 2015.

- B. Nithya and Dr. V. Ilango,” Predictive Analytics in Health Care Using Machine Learning Tools and Techniques”, International Conference on Intelligent Computing and Control Systems, 978-1-5386-2745-7, 2017.

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

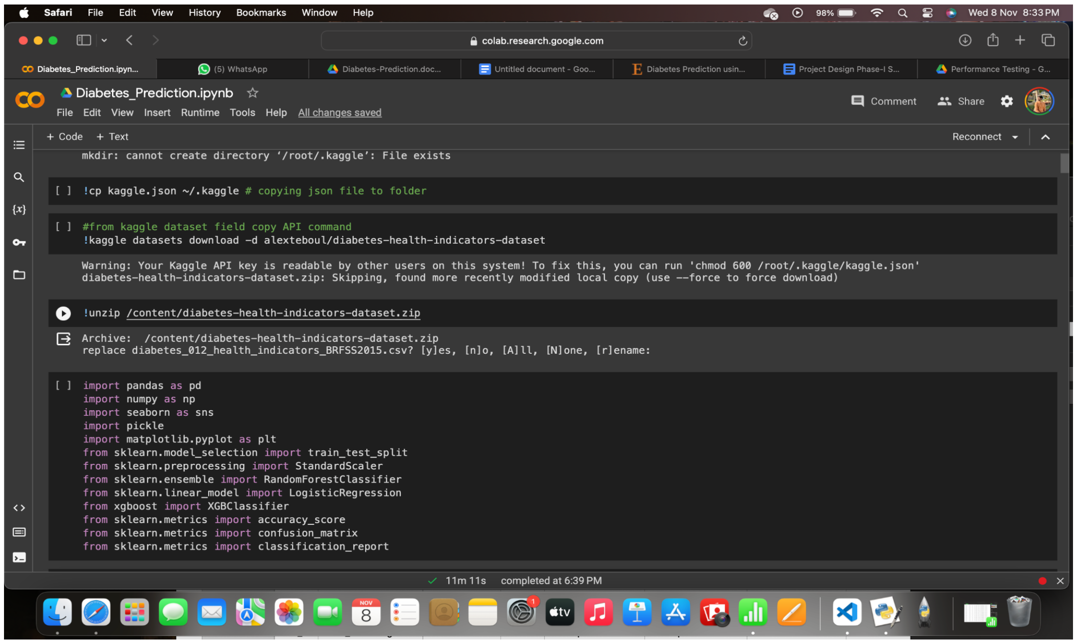
ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

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.

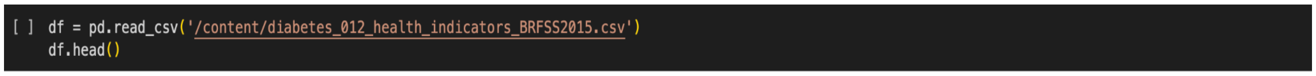

Diabetes_Prediction.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Reconnect
Comment
Share

```
mkdir: cannot create directory '/root/.kaggle': File exists

[ ] !cp kaggle.json ~/.kaggle # copying json file to folder

[ ] #from kaggle dataset field copy API command
    !kaggle datasets download -d alexteboul/diabetes-health-indicators-dataset

    Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
    diabetes-health-indicators-dataset.zip: Skipping, found more recently modified local copy (use --force to force download)

    !unzip /content/diabetes-health-indicators-dataset.zip

    Archive:  /content/diabetes-health-indicators-dataset.zip
    replace diabetes_012_health_indicators_BRFSS2015.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename:

[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
    import pickle
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.linear_model import LogisticRegression
    from xgboost import XGBClassifier
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import classification_report
```

```
✓ 11m 11s   completed at 6:39 PM
```

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()
```

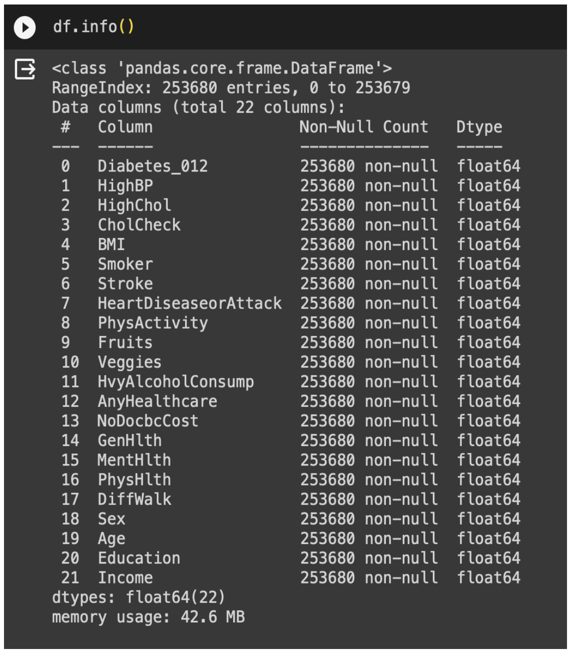
Activity 2: Data Preparation

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

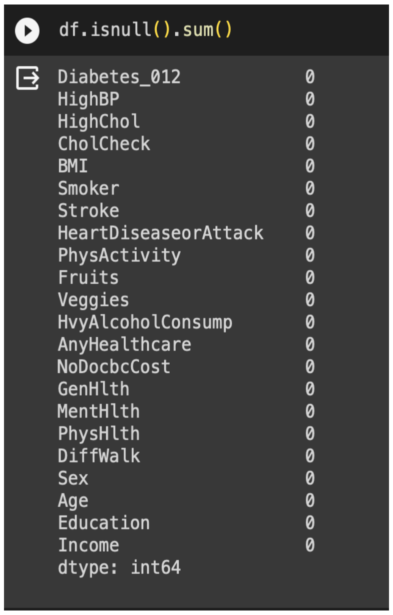
● Let’s know the info and describe our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used



```
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().sum()
```

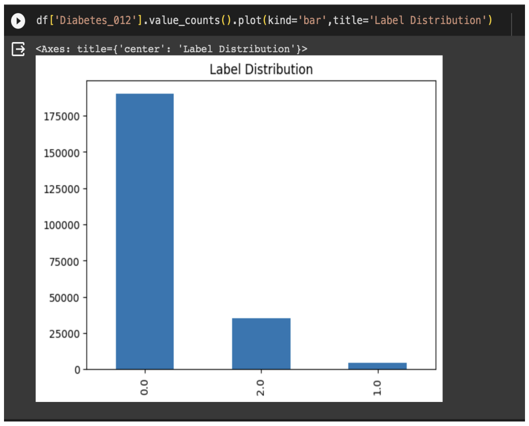
```
Diabetes_012            0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data, we must convert the categorical data to integer encoding or binary encoding.
To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using Label encoding with the help of list comprehension.
• In our project, there are no categorical features therefore no encoding has to be done.

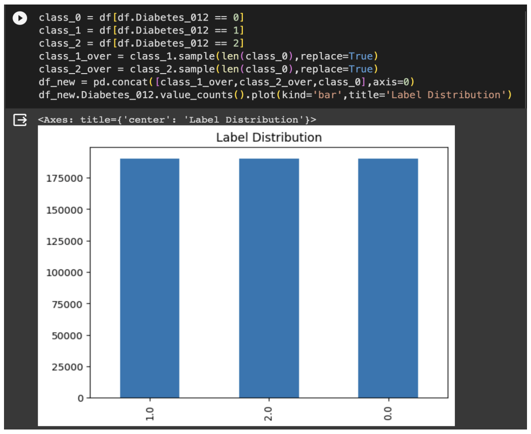
Activity 2.3: Handling Imbalance Data

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 features have outliers Boxplot from seaborn library is used here.



```
df['Diabetes_012'].value_counts().plot(kind='bar',title='Label Distribution')
```

```
<Axes: title={'center': 'Label Distribution'}>
```



```
class_0 = df[df.Diabetes_012 == 0]
class_1 = df[df.Diabetes_012 == 1]
class_2 = df[df.Diabetes_012 == 2]
class_1_over = class_1.sample(len(class_0),replace=True)
class_2_over = class_2.sample(len(class_0),replace=True)
df_new = pd.concat([class_1_over,class_2_over,class_0],axis=0)
df_new.Diabetes_012.value_counts().plot(kind='bar',title='Label Distribution')
```

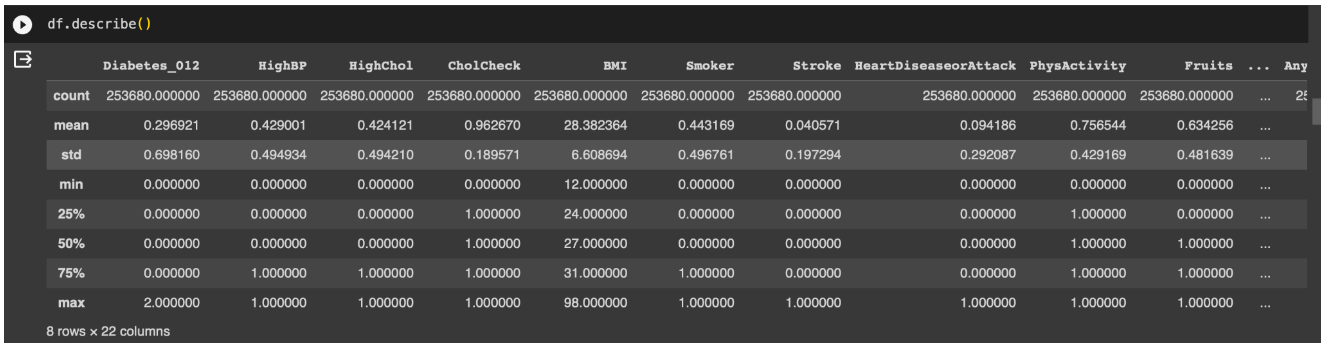
```
<Axes: title={'center': 'Label Distribution'}>
```

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

.



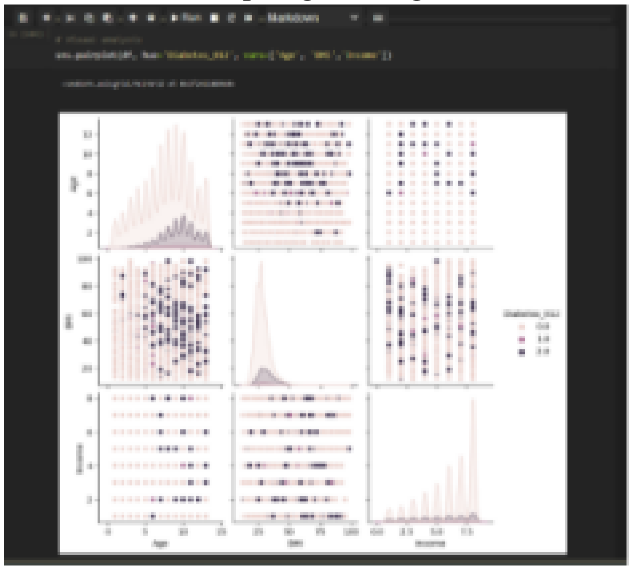
```
df.describe()
```

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

8 rows × 22 columns

Activity 2: Visual analysis

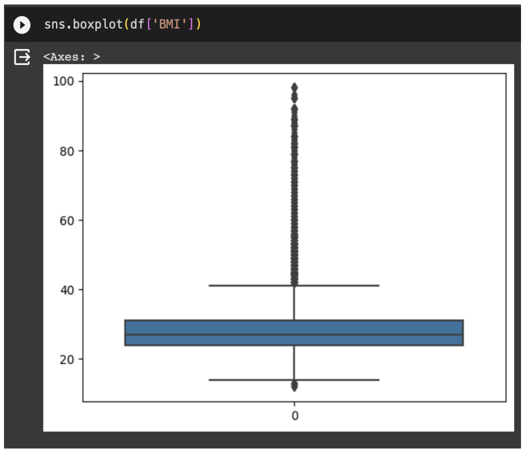
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.



```
# Visual analysis
sns.pairplot(df, hue='Diabetes_012', vars=['Age', 'BMI', 'Income'])
```

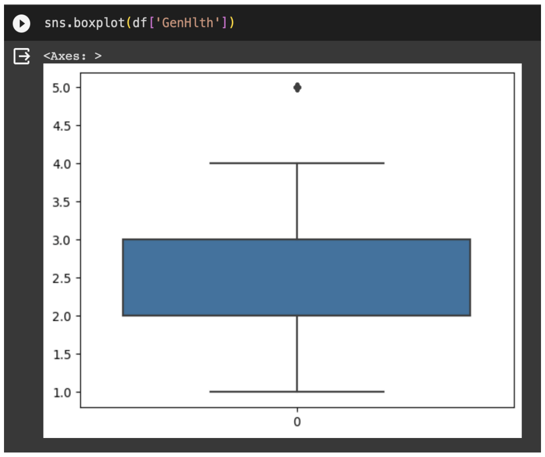
Activity 2.1: Univariate analysis

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.boxplot(df['BMI'])
```

```
<Axes: >
```



```
sns.boxplot(df['GenHlth'])
```

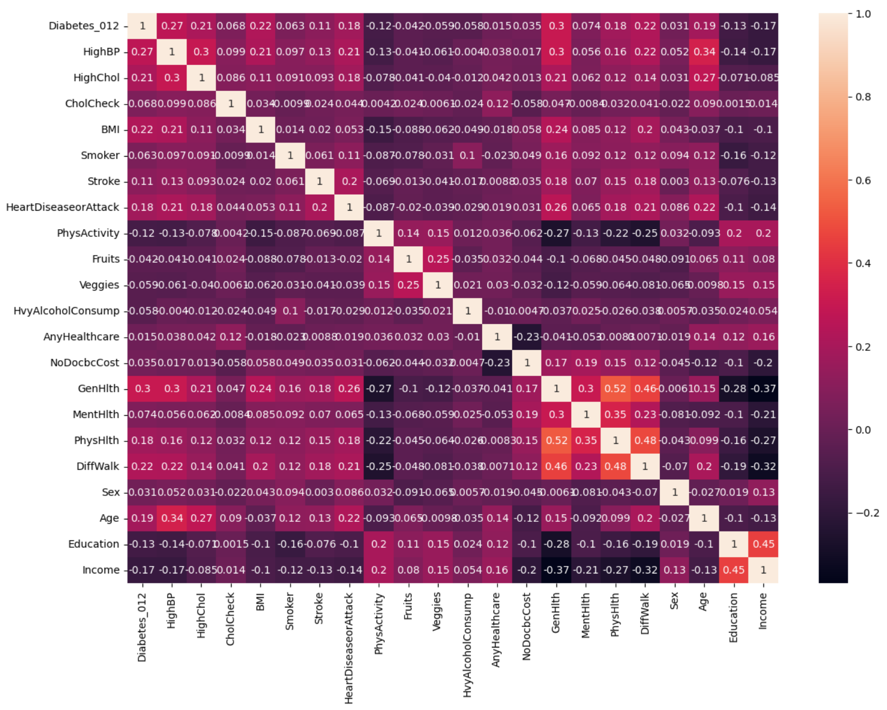
```
<Axes: >
```

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.

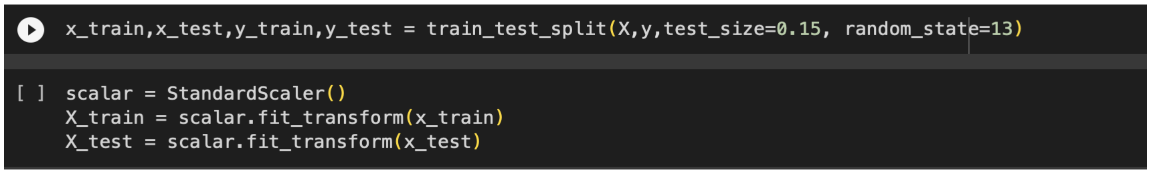
Activity 2.3: Multivariate analysis

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



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 my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.15, random_state=13)
```

```
[ ] scalar = StandardScaler()
    X_train = scalar.fit_transform(x_train)
    X_test = scalar.fit_transform(x_test)
```

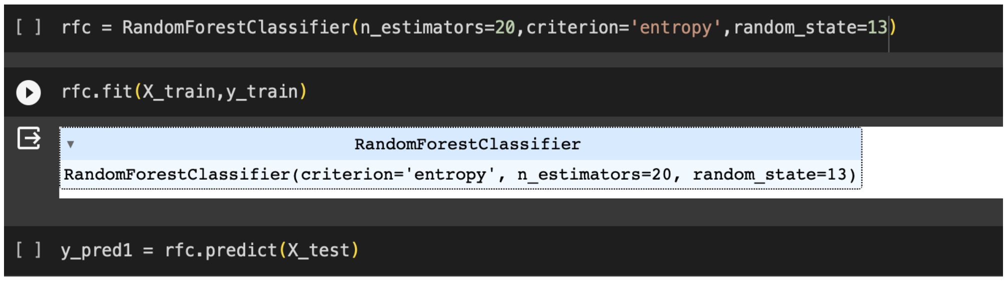
Milestone 4: Model Building

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

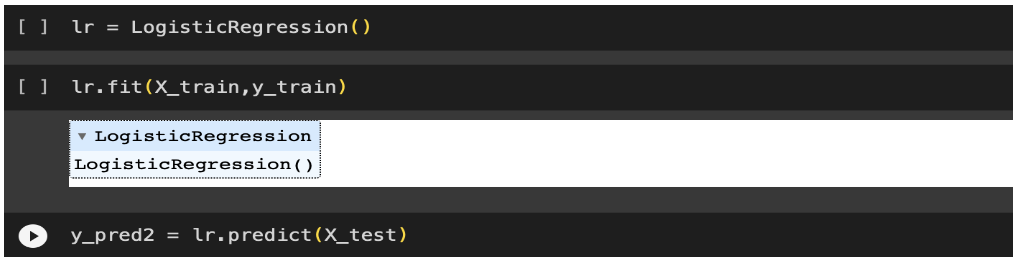
A function named random forest regressor is created and train and test data are passed as the parameters. Inside the function, a random forest regressor algorithm is initialized and training data is passed to the model with the fit() function. Test data is predicted with predict () function and saved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



```
[ ] rfc = RandomForestClassifier(n_estimators=20,criterion='entropy',random_state=13)

rfc.fit(X_train,y_train)

RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=20, random_state=13)

[ ] y_pred1 = rfc.predict(X_test)
```

Activity 1.2: Logistic Regression

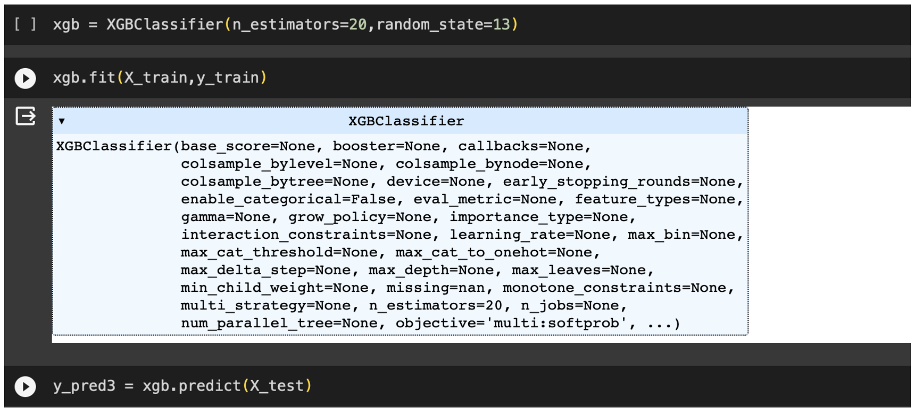
To evaluate the performance of a logistic regression model, we need to test it on a separate dataset that it has not seen during training. This separate dataset is called the test data. We typically split the available data into two parts, the training data and the test data. The model is trained on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



```
[ ] lr = LogisticRegression()

[ ] lr.fit(X_train,y_train)

LogisticRegression
LogisticRegression()

y_pred2 = lr.predict(X_test)
```

Activity 1.3: XGBClassifier

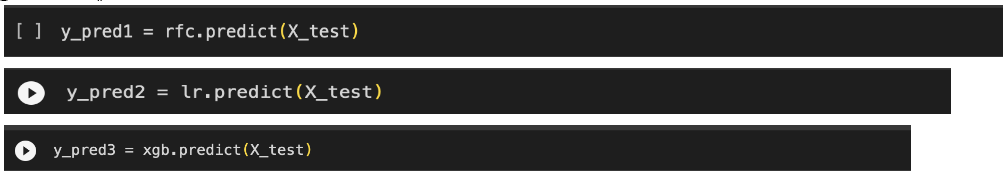
XGBoost (eXtreme Gradient Boosting) is a powerful and widely-used gradient boosting algorithm that is used to solve many different types of machine learning problems. It is an implementation of gradient boosting that is specifically designed to be efficient and scalable, making it a popular choice for working with large datasets.



```
[ ] xgb = XGBClassifier(n_estimators=20,random_state=13)

xgb.fit(X_train,y_train)

XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=20, n_jobs=None,
              num_parallel_tree=None, objective='multi:softprob', ...)

y_pred3 = xgb.predict(X_test)
```

Activity 2: Testing the model

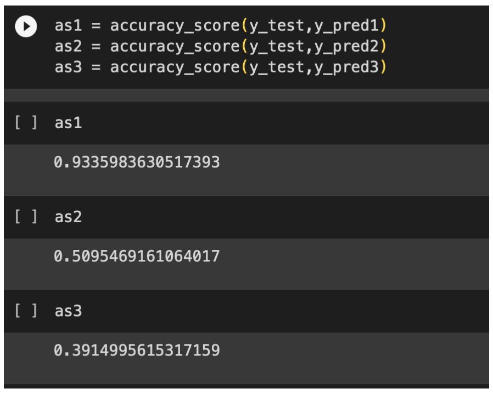
Here we have tested with Logistic regression and Random Forest algorithms. With the help of predict() function.



```
[ ] y_pred1 = rfc.predict(X_test)

y_pred2 = lr.predict(X_test)

y_pred3 = xgb.predict(X_test)
```

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.

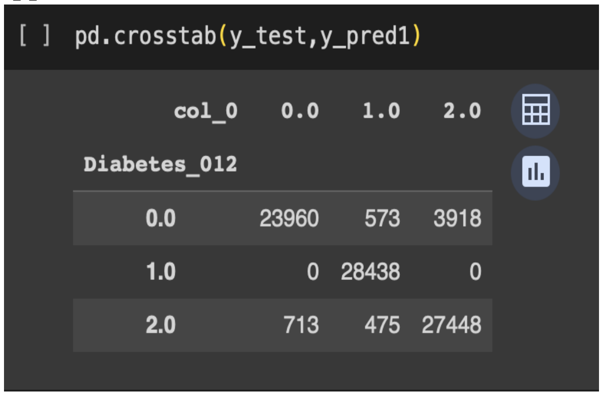


```
as1 = accuracy_score(y_test,y_pred1)
as2 = accuracy_score(y_test,y_pred2)
as3 = accuracy_score(y_test,y_pred3)
```

```
as1
0.9335983630517393
```

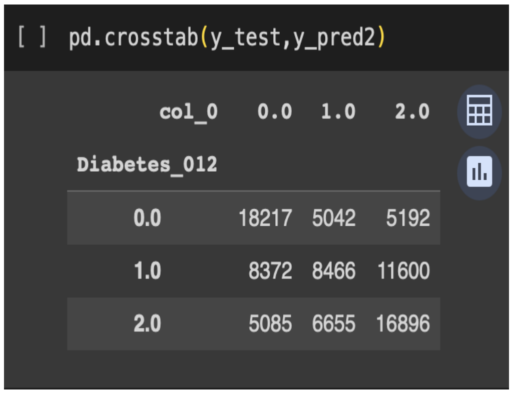
```
as2
0.5095469161064017
```

```
as3
0.3914995615317159
```



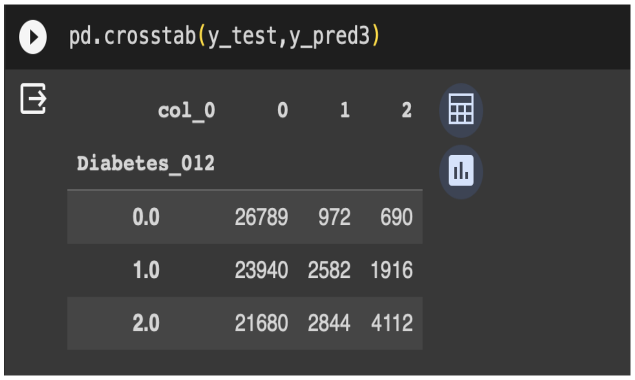
```
pd.crosstab(y_test,y_pred1)
```

col_0 / Diabetes_012	0.0	1.0	2.0
0.0	23960	573	3918
1.0	0	28438	0
2.0	713	475	27448



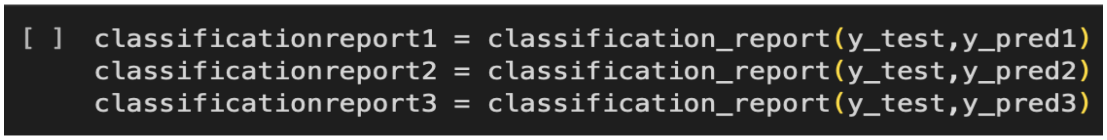
```
pd.crosstab(y_test,y_pred2)
```

col_0 / Diabetes_012	0.0	1.0	2.0
0.0	18217	5042	5192
1.0	8372	8466	11600
2.0	5085	6655	16896

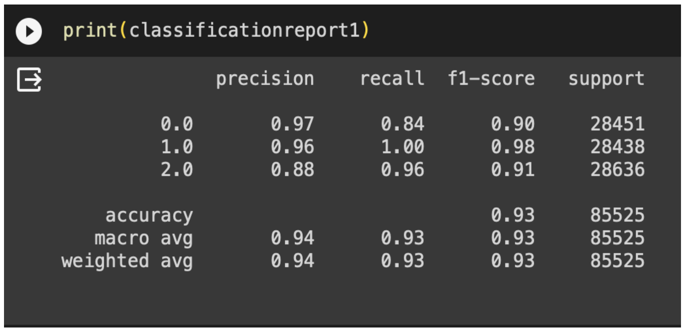


```
pd.crosstab(y_test,y_pred3)
```

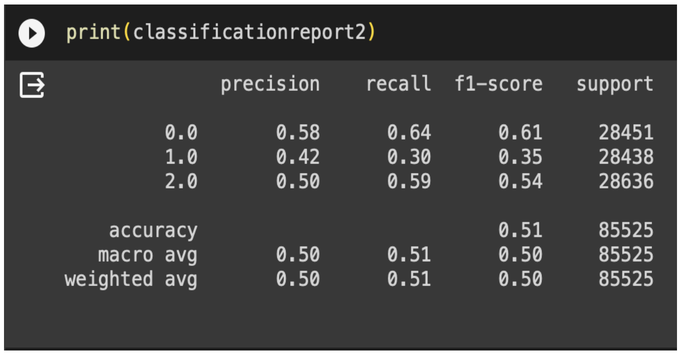
col_0 / Diabetes_012	0	1	2
0.0	26789	972	690
1.0	23940	2582	1916
2.0	21680	2844	4112

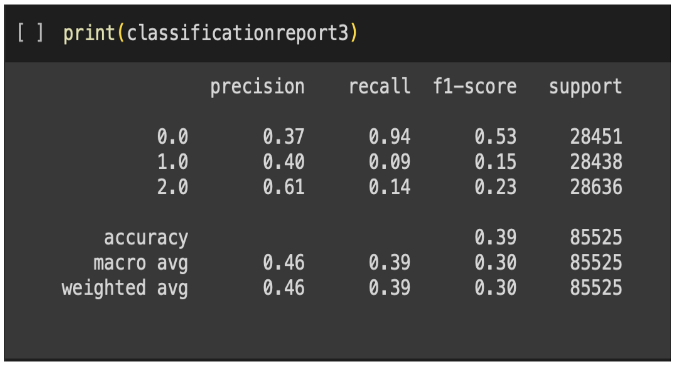


```
classificationreport1 = classification_report(y_test,y_pred1)
classificationreport2 = classification_report(y_test,y_pred2)
classificationreport3 = classification_report(y_test,y_pred3)
```



```
print(classificationreport1)

              precision    recall  f1-score   support

         0.0       0.97      0.84      0.90     28451
         1.0       0.96      1.00      0.98     28438
         2.0       0.88      0.96      0.91     28636

    accuracy                           0.93     85525
   macro avg       0.94      0.93      0.93     85525
weighted avg       0.94      0.93      0.93     85525
```


```
print(classificationreport2)

              precision    recall  f1-score   support

         0.0       0.58      0.64      0.61     28451
         1.0       0.42      0.30      0.35     28438
         2.0       0.50      0.59      0.54     28636

    accuracy                           0.51     85525
   macro avg       0.50      0.51      0.50     85525
weighted avg       0.50      0.51      0.50     85525
```

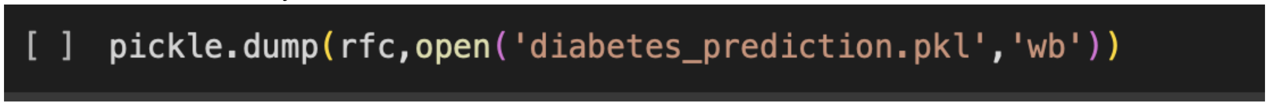

```
[ ] print(classificationreport3)

              precision    recall  f1-score   support

         0.0       0.37      0.94      0.53     28451
         1.0       0.40      0.09      0.15     28438
         2.0       0.61      0.14      0.23     28636

    accuracy                           0.39     85525
   macro avg       0.46      0.39      0.30     85525
weighted avg       0.46      0.39      0.30     85525
```

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

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.


```
[ ] pickle.dump(rfc,open('diabetes_prediction.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

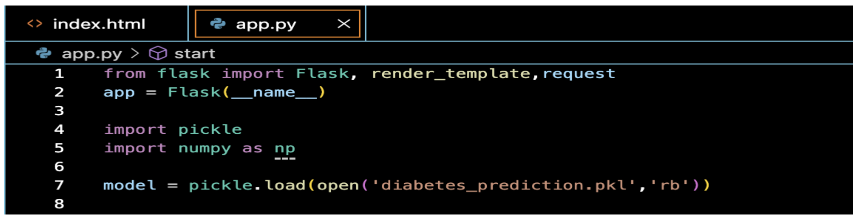
Activity 2.1: Building Html Pages:

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

and save them in the templates folder. Refer this [ENTER THE LINK]() for templates, static and python file

Activity 2.2: Build Python code:

Import the libraries in python file



index.html   app.py

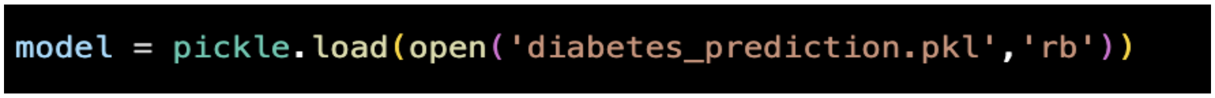
app.py > start

```python
from flask import Flask, render_template,request
app = Flask(__name__)

import pickle
import numpy as np

model = pickle.load(open('diabetes_prediction.pkl','rb'))

```

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

Render HTML page:

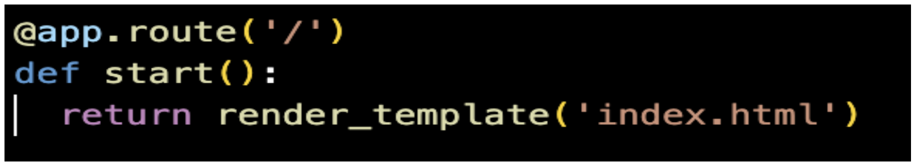
Here we will be using a declared constructor to route to the HTML page which we have created earlier.



```python
model = pickle.load(open('diabetes_prediction.pkl','rb'))
```

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

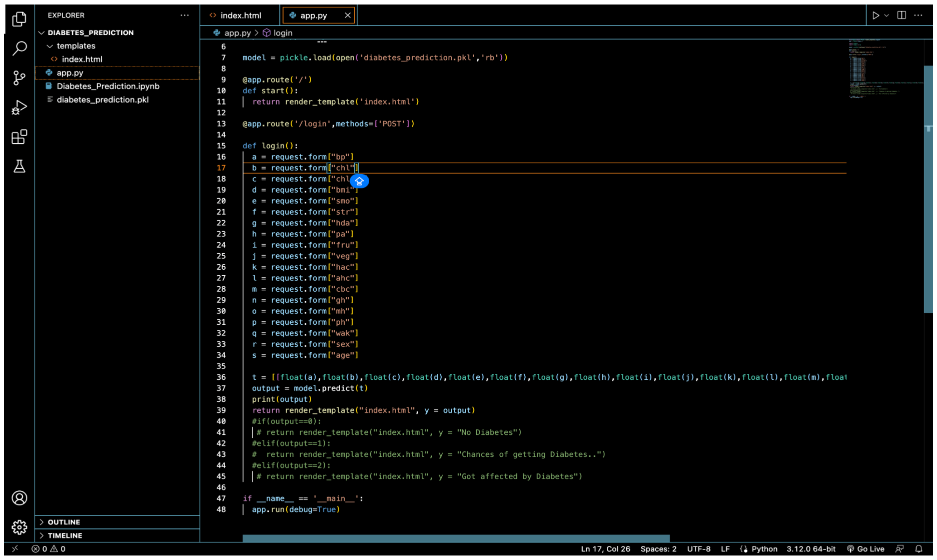
Retrieves the value from UI:



```python
@app.route('/')
def start():
    return render_template('index.html')
```

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

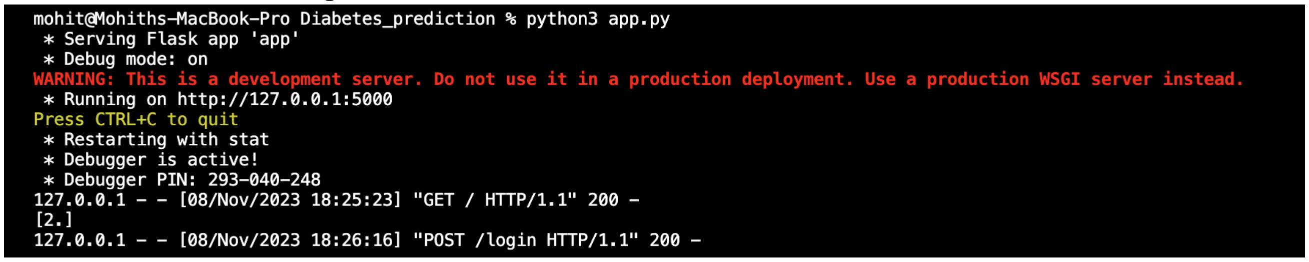
Main Function:



```python

model = pickle.load(open('diabetes_prediction.pkl','rb'))

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

@app.route('/login',methods=['POST'])

def login():
  a = request.form["bp"]
  b = request.form["chl"]
  c = request.form["chl
  d = request.form["bmi"]
  e = request.form["smo"]
  f = request.form["str"]
  g = request.form["hda"]
  h = request.form["pa"]
  i = request.form["fru"]
  j = request.form["veg"]
  k = request.form["hac"]
  l = request.form["ahc"]
  m = request.form["cbc"]
  n = request.form["gh"]
  o = request.form["mh"]
  p = request.form["ph"]
  q = request.form["wak"]
  r = request.form["sex"]
  s = request.form["age"]

  t = [[float(a),float(b),float(c),float(d),float(e),float(f),float(g),float(h),float(i),float(j),float(k),float(l),float(m),floa
  output = model.predict(t)
  print(output)
  return render_template("index.html", y = output)
  #if(output==0):
   # return render_template("index.html", y = "No Diabetes")
  #elif(output==1):
  #  return render_template("index.html", y = "Chances of getting Diabetes..")
  #elif(output==2):
   # return render_template("index.html", y = "Got affected by Diabetes")

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

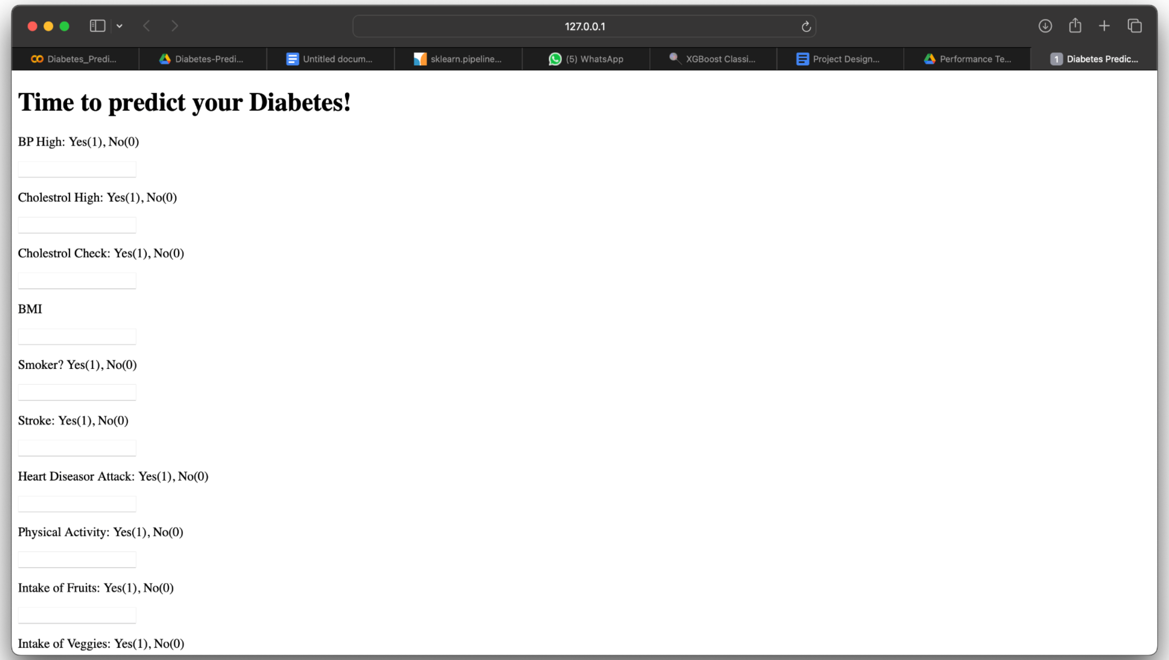
Activity 2.3: Run the web application

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



```
mohit@Mohiths-MacBook-Pro Diabetes_prediction % python3 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
 * Debugger is active!
 * Debugger PIN: 293-040-248
127.0.0.1 - - [08/Nov/2023 18:25:23] "GET / HTTP/1.1" 200 -
[2.]
127.0.0.1 - - [08/Nov/2023 18:26:16] "POST /login HTTP/1.1" 200 -
```

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



Time to predict your Diabetes!

BP High: Yes(1), No(0)

Cholestrol High: Yes(1), No(0)

Cholestrol Check: Yes(1), No(0)

BMI

Smoker? Yes(1), No(0)

Stroke: Yes(1), No(0)

Heart Diseasor Attack: Yes(1), No(0)

Physical Activity: Yes(1), No(0)

Intake of Fruits: Yes(1), No(0)

Intake of Veggies: Yes(1), No(0)