



Diabetes Prediction Using Machine

Learning

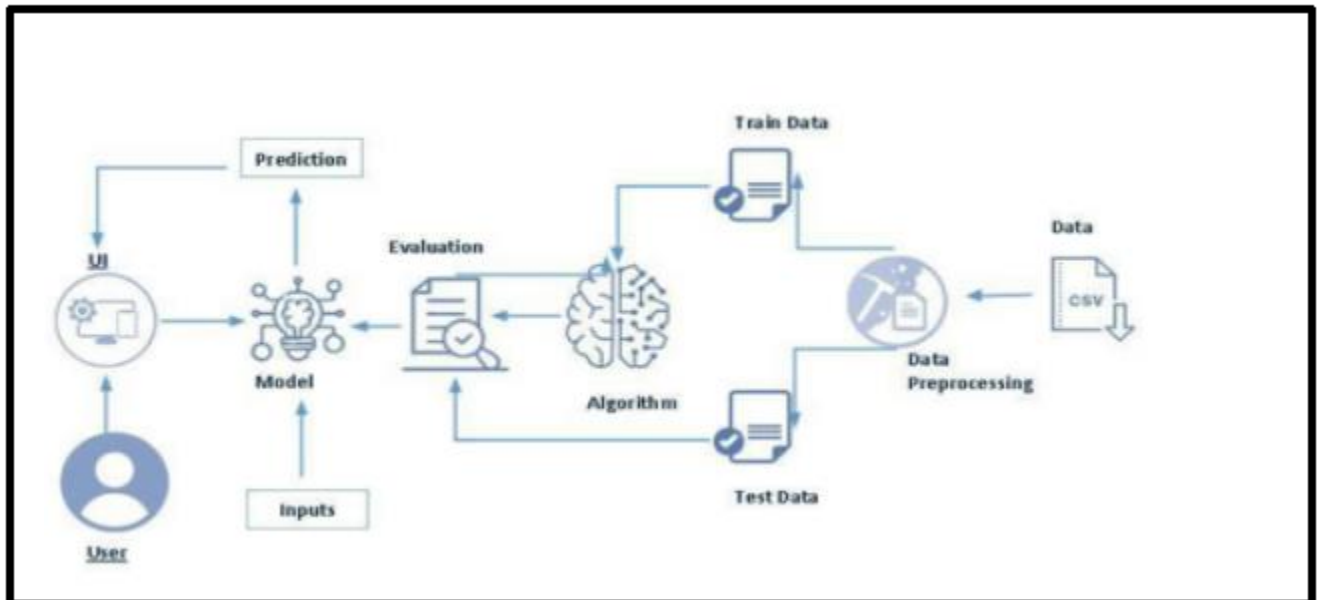
Diabetes Prediction Using Machine Learning

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

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

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

Technical Architecture:



Project Flow:

- User 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 ○ 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.

- 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>

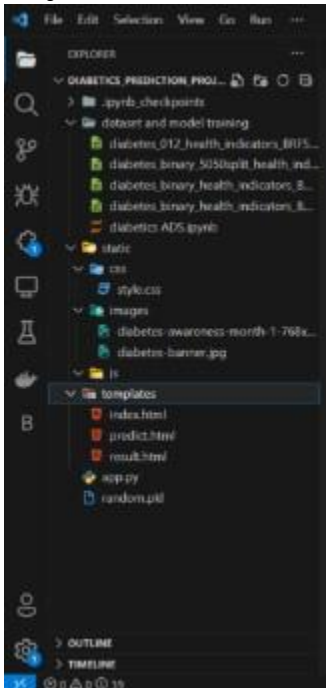
- KNN: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
- Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
- Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>

•

NLP:-https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_python.htm

- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:



Create the Project folder which contains files as shown below

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Random.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
from pandas_profiling import ProfileReport
import math

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.model_selection import train_test_split
from sklearn.over_sampling import SMOTE

from sklearn.metrics import confusion_matrix, plot_roc_curve, classification_report
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error, acc
from sklearn.metrics import plot_confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline

import warnings

```

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.

IMPORTING DATASET AND PREPROCESSING THE DATA

```
df=pd.read_csv("diabetes_binary_health_indicators_BRFSS2015.csv")
df.head()
```

[140]

Python

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost
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
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
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
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
4	0.0	1.0	1.0	1.0	24.0	1.0	0.0	0.0	1.0	1.0	...	1.0	0.0

5 rows x 22 columns

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

```

In [7]: df.info()

Out[7]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20000 entries, 0 to 19999
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  Dtype1
---  --
 0   Diabetes_binary       20000 non-null  float64
 1   HighBP                20000 non-null  float64
 2   HighChol              20000 non-null  float64
 3   CholCheck            20000 non-null  float64
 4   BMI                  20000 non-null  float64
 5   Smoker               20000 non-null  float64
 6   Stroke               20000 non-null  float64
 7   HeartDiseaseorAttack  20000 non-null  float64
 8   PhysActivity         20000 non-null  float64
 9   Fruits               20000 non-null  float64
10   Veggies              20000 non-null  float64
11   HvyAlcoholConsump    20000 non-null  float64
12   AnyHealthcare        20000 non-null  float64
13   NoDocbcCost          20000 non-null  float64
14   GenHlth              20000 non-null  float64
15   MentHlth             20000 non-null  float64
16   PhysHlth             20000 non-null  float64
17   DiffWalk             20000 non-null  float64
18   Sex                  20000 non-null  float64
19   Age                  20000 non-null  float64
20   Education            20000 non-null  float64
21   Income               20000 non-null  float64

dtypes: float64(22)
memory usage: 41.0 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()

[7]
... Diabetes_binary      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                  1
    Age                  1
    Education            1
    Income               1

```



```
for i in bin_col:
    df[i]=df[i].fillna(df[i].mode()[0])
for i in val_col:
    df[i]=df[i].fillna(df[i].mean())

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

Diabetes_binary	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

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.

```
#CHECKING FOR OUTLIERS AND REMOVING IF ANY
def rem_out(df,col,threshold=1.5):
    cleaned_df=df.copy()
    for column in col:
        q1=df[column].quantile(0.25)
        q3=df[column].quantile(0.75)
        iqr=q3-q1
        upper_limit=q3+threshold*iqr
        lower_limit=q1-threshold*iqr
        cleaned_df=cleaned_df[(cleaned_df[column] >= lower_limit) & (cleaned_df[column] <= upper_limit)]
    return cleaned_df
```

Milestone 3: Exploratory Data Analysis

Python

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

```
18 [7]: #checking statistical analysis of dataset
df.describe()
```

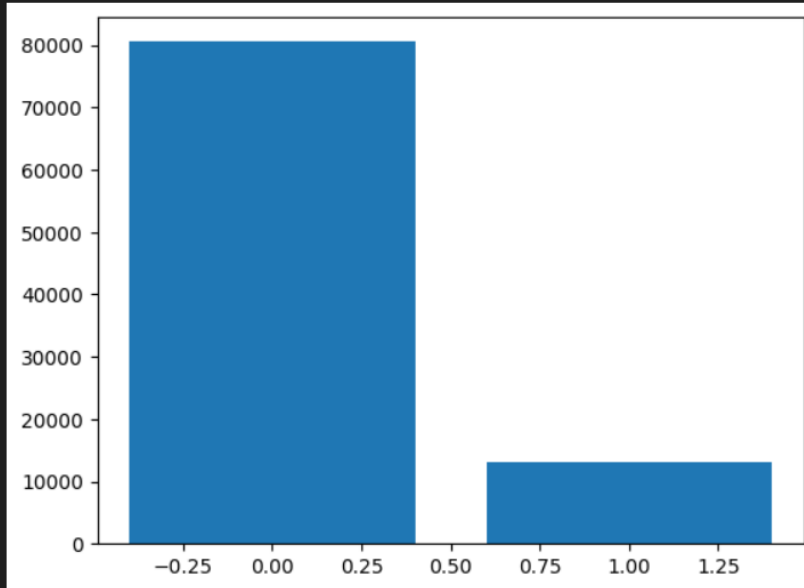
	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	k	PhysActivity	
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	0	93579.000000	93579
mean	0.296921	0.429001	0.424121	0.962670	26.362364	0.443169	0.040571	0.094186	2	0.761977	C
std	0.698160	0.494834	0.494210	0.189571	6.638694	0.496761	0.197294	0.292087	1	0.425876	C
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0	0.000000	C
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	0.000000	0	1.000000	C
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	0.000000	0	1.000000	C
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	0.000000	0	1.000000	1
max	2.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.000000	0	1.000000	1

8 rows x 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.

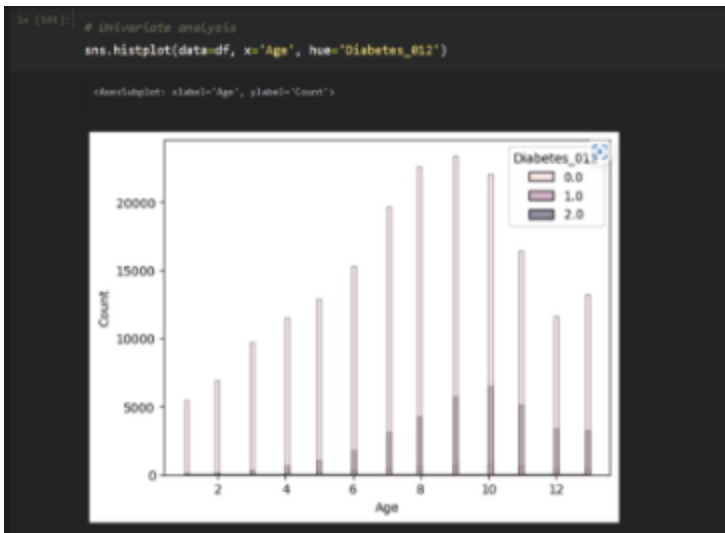
```
a=df['Diabetes_binary'].value_counts().index  
plt.bar(a,df['Diabetes_binary'].value_counts().index)  
plt.show()
```



Activity 2.1: Univariate analysis

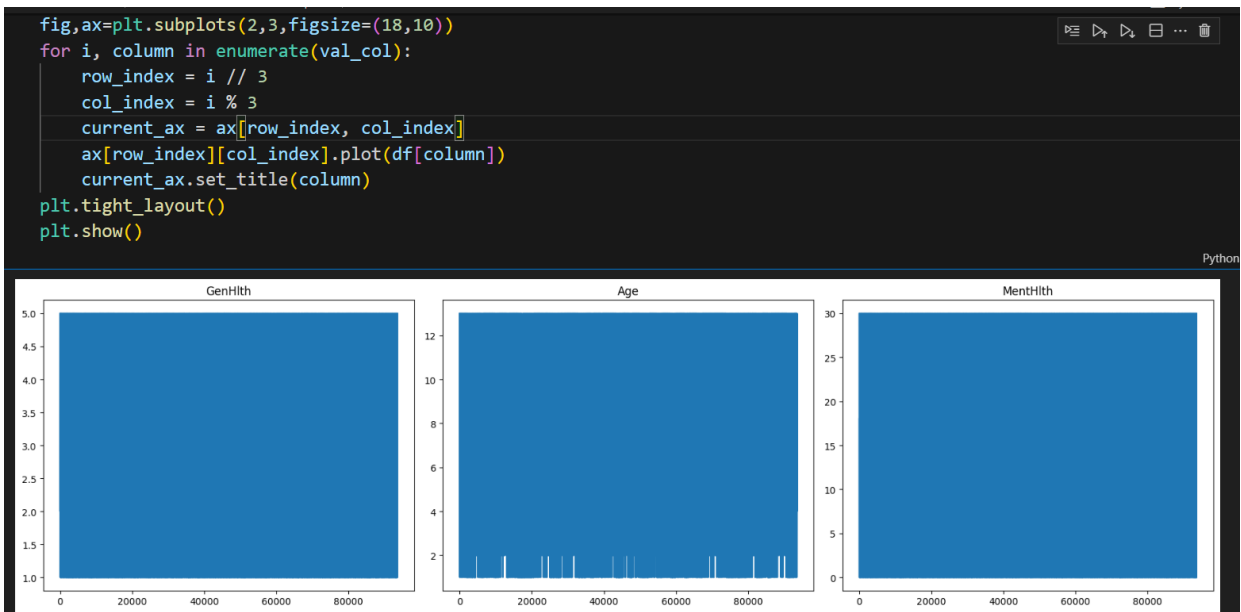
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and count plot.

In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features.



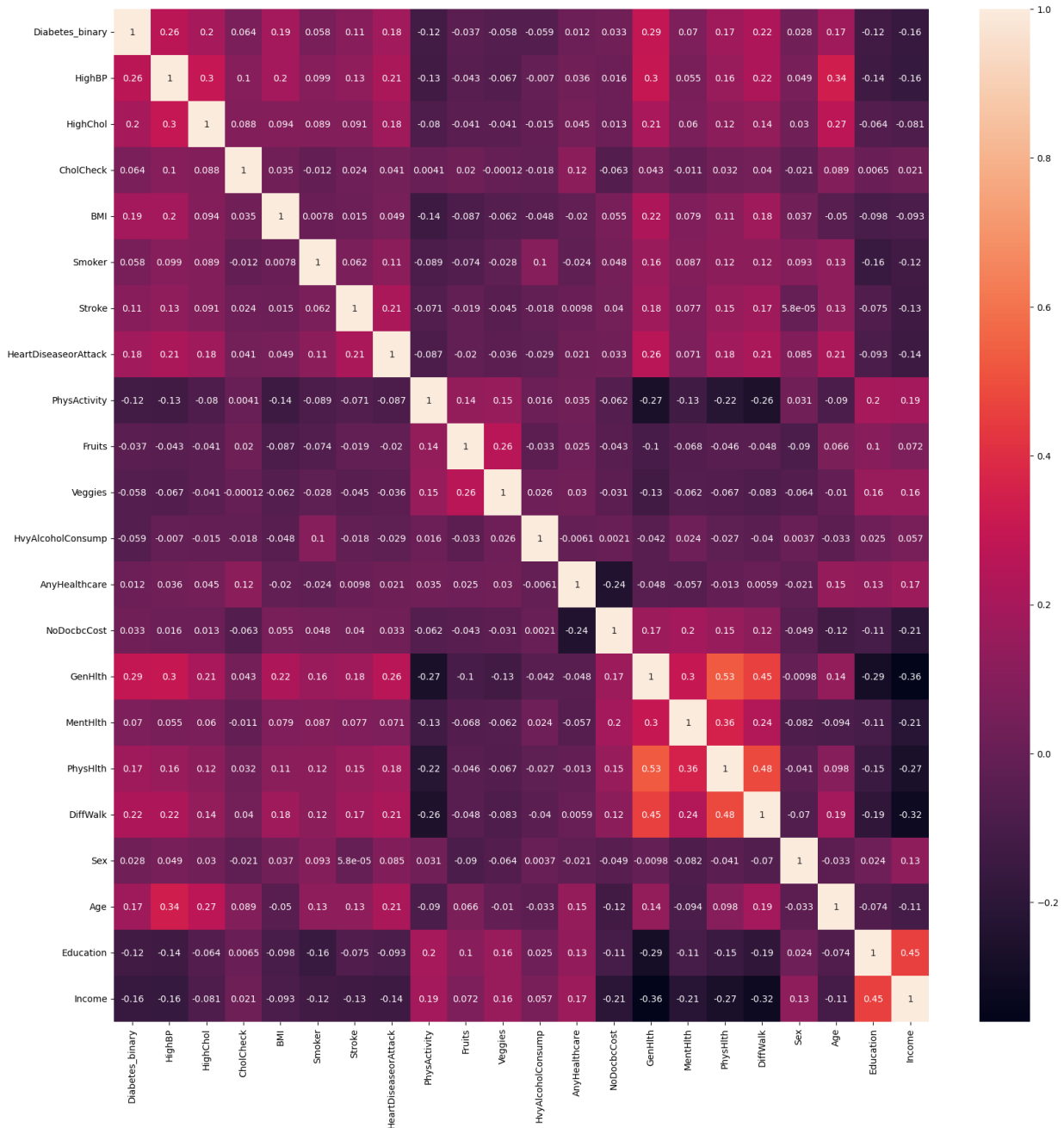
Activity 2.2: Bivariate analysis

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.



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.

```
from sklearn.preprocessing import MinMaxScaler
scaler =MinMaxScaler()
X= pd.DataFrame(scaler.fit_transform(X),columns =X.columns)
X.head()
```

	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	...	AnyHealthcare	NoDocbcCost	Genl
0	1.0	1.0	1.0	0.325581	1.0	0.0	0.0	0.0	0.0	1.0	...	1.0	0.0	
1	0.0	0.0	0.0	0.151163	1.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	
2	1.0	1.0	1.0	0.186047	0.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	
3	1.0	0.0	1.0	0.174419	0.0	0.0	0.0	1.0	1.0	1.0	...	1.0	0.0	
4	1.0	1.0	1.0	0.139535	0.0	0.0	0.0	1.0	1.0	1.0	...	1.0	0.0	

5 rows × 21 columns

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.

```
In [62]: rf = RandomForestClassifier(max_depth=12, n_estimators=10, random_state=42)
# fitting the model on the train data
rf.fit(X_train, Y_train)

RandomForestClassifier(max_depth=12, n_estimators=10, random_state=42)

In [66]: # make predictions on test set
y_pred=rf.predict(X_test)

print('Training set score: {:.4f}'.format(rf.score(X_train, Y_train)))
print('Test set score: {:.4f}'.format(rf.score(X_test, Y_test)))

Training set score: 0.7648
Test set score: 0.7264

In [67]: #check MSE & RMSE
mse = mean_squared_error(Y_test, y_pred)
print('Mean Squared Error : '+ str(mse))
rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error : '+ str(rmse))

Mean Squared Error : 0.7847375518925189
Root Mean Squared Error : 0.8853539581646523
```

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

```
from sklearn.tree import DecisionTreeClassifier
model_2 = DecisionTreeClassifier(max_depth=6, splitter='best', criterion='entropy')
model_2.fit(x_train, y_train)

DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=6)

train_pred_2=model_2.predict(x_train)
y_pred_2=model_2.predict(x_test)

print("train accuracy",accuracy_score(y_train,train_pred_2))
print("test accuracy",accuracy_score(y_test,y_pred_2))

train accuracy 0.8096807067162085
test accuracy 0.8108640392058076
```

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.

```
[187] from sklearn.linear_model import LogisticRegression
      model_1 = LogisticRegression()
      model_1.fit(x_train,y_train)

...
▼ LogisticRegression
LogisticRegression()

▷
[188] from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve
      y_pred_1 = model_1.predict(x_test)
      train_pred_1=model_1.predict(x_train)

[189] print("train accuracy",accuracy_score(y_train,train_pred_1))
      print("test accuracy",accuracy_score(y_test,y_pred_1))

... train accuracy 0.7542466522010454
      test accuracy 0.7566468958252227
```

Activity 2: Testing the model

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

```
In [70]: from sklearn.svm import SVC

# define the model
clf = SVC(kernel='rbf', C=1.0)

# train the model
clf.fit(X_train, Y_train)

y_pred=clf.predict(X_test)

print('Training set score: {:.4f}'.format(clf.score(X_train, Y_train)))

print('Test set score: {:.4f}'.format(clf.score(X_test, Y_test)))

Training set score: 0.7392
Test set score: 0.7298

In [71]: #check MSE & RMSE
mse = mean_squared_error(Y_test, y_pred)
print('Mean Squared Error : '+ str(mse))
rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error : '+ str(rmse))

Mean Squared Error : 0.7756179505639549
Root Mean Squared Error : 0.8806917454841705
```


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.

```
confusion_matrix(y_test,y_pred_1)

array([[31854, 11919],
       [ 9334, 34227]])
```

Activity 1.1: Compare the model

For comparing the below two models, with their R₂ 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

```
File Edit View Insert Cell Kernel Widgets Help Python 3 (pykernel)

In [102]: pipeline_lr=Pipeline([('scalar',StandardScaler()),
                                ('pca',PCA(n_components=3)),
                                ('lr_classifier',LogisticRegression(random_state=0))])

In [103]: pipeline_dt=Pipeline([('scalar',StandardScaler()),
                                ('pca',PCA(n_components=3)),
                                ('dt_classifier',DecisionTreeClassifier())])

In [104]: pipeline_randomforest=Pipeline([('scalar',StandardScaler()),
                                           ('pca',PCA(n_components=3)),
                                           ('rf_classifier',RandomForestClassifier())])

In [105]: # Let's make the list of pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest]

In [106]: best_accuracy=0.0
best_classifier=""
best_pipeline=""

In [107]: # Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {'lr': 'Logistic Regression', 1: 'Decision Tree', 2: 'Randomforest'}

# Fit the pipelines
for pipe in pipelines:
    pipe.fit(X_train, y_train)

In [108]: for i,model in enumerate(pipelines):
    print(' {} Test Accuracy: {}'.format(pipe_dict[i],model.score(X_test,y_test)))

Logistic Regression Test Accuracy: 0.9999999999999999
Decision Tree Test Accuracy: 0.9111111111111111
Randomforest Test Accuracy: 0.9111111111111111
```

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)

```
MakePipelines in SKLearn

In [93]: from sklearn.pipeline import make_pipeline

In [94]: # Create a pipeline
pipe = make_pipeline(RandomForestClassifier())
# Create dictionary with candidate Learning algorithms and their hyperparameters
grid_param = [
    {"randomforestclassifier": [RandomForestClassifier()],
     "randomforestclassifier__n_estimators": [10, 100, 1000],
     "randomforestclassifier__max_depth": [5, 8, 15, 25, 30, None],
     "randomforestclassifier__min_samples_leaf": [1, 2, 5, 10, 15, 100],
     "randomforestclassifier__max_leaf_nodes": [2, 5, 10]}]
# create a gridsearch of the pipeline, the fit the best model
gridsearch = GridSearchCV(pipe, grid_param, cv=5, verbose=0, n_jobs=-1) # Fit grid search
best_model = gridsearch.fit(X_train, y_train)

In [95]: best_model.score(X_test, y_test)

0.9777777777777777
```

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
pickle.dump(model_2, 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:

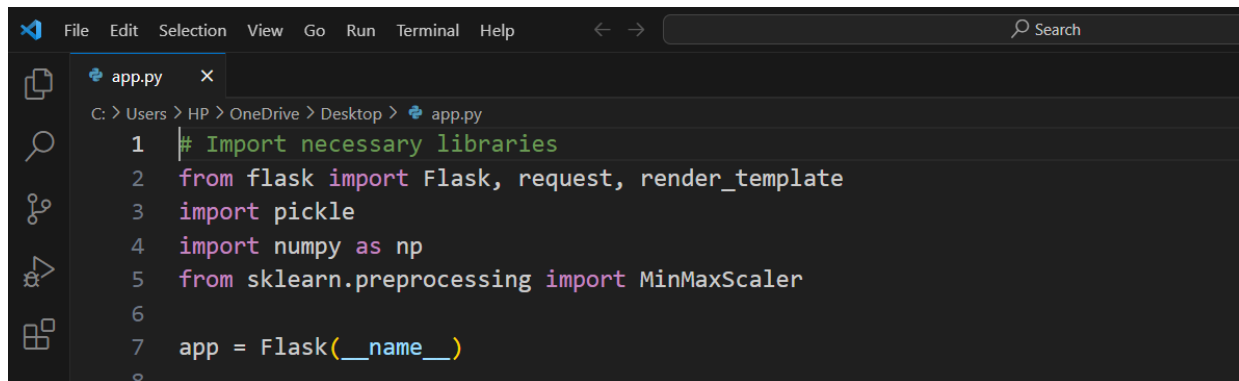
For this project create two HTML files namely

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

A screenshot of a code editor window with a dark theme. The title bar shows 'app.py' and a close button. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The breadcrumb path is 'C: > Users > HP > OneDrive > Desktop > app.py'. The code is as follows:

```
1  # Import necessary libraries
2  from flask import Flask, request, render_template
3  import pickle
4  import numpy as np
5  from sklearn.preprocessing import MinMaxScaler
6
7  app = Flask(__name__)
8
```

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__)

# Load the pre-trained model and scaler
with open('Diabetes_prediction.pkl', 'rb') as model_file:
    model = pickle.load(model_file)
```

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:

```
# Define a route for the home page
@app.route('/')
def home():
    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:

```

def predict():
    input_data = []

    # Get user input for the specified columns
    for column in ['Diabetes_binary', 'HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker',
                  'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
                  'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
                  'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education',
                  'Income']:
        input_data.append(float(request.form[column]))

    # Scale the input data using the MinMaxScaler
    input_data = np.array(input_data).reshape(1, -1)
    scaled_input = scaler.transform(input_data)

    # Make a prediction using the pre-trained model
    prediction = model.predict(scaled_input)

    return f'Predicted Diabetes Probability: {prediction[0]}'

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.

```

PS D:\ads\Diabetics_Prediction_Project> C:/Users/siddh/AppData/Local/Programs/Python/Python39/python.exe d:/ads/Diabetics_Prediction_Project/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: 400-570-382

```

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

Diabetes is a very common disease affecting individuals worldwide. Diabetes increases the risk of long-term complications including heart disease, and kidney failure among others.....

To check You have Diabetes Or No

[Check](#)



Name	BMI
Select Blood Pressure value: No High BP	Select Attitude: Not a Smoker
Select Age Range: No high cholesterol	Select Alcohol Limit: No Heart Attack
Age: 18-24	Select list (select one): No Alcohol Consumption
Select About Physical Activity: No physical activity in past 30 days	Select Health: Poor Health
Select about walking: Not having difficulty while walking or climbing stairs	Education: Never Attended School
Select Consulted: Not consulted Doctor in 12 months	Income: Less Than \$10,000
Select Gender: Female	Selecting about heart Diseases: No coronary heart disease (CHD) or myocardial infarction
Your mental health, for past 30 days, like stress, emotions etc	Your Physical health, for past 30 days, like illness and injury etc
Submit	

Name: test123

Sex: Female

Having Diabetes or Not: Not having Diabetes