Fundamentals of Data Mining - IT3051

Group Project – G21

Statement Of Work Document

IT Number	Student Name
IT21068850	Yusri M.A.M
IT21014840	Rathnayake R.M.K.D.B
IT21107078	Rupasingha W.P.S
IT21086984	Senadheera S.A.T.P

Date of Submit

18/10/23

Table of Contents

Background

Predict diabetes in patients based on their medical history and demographic information.

For the following assignment, our group will be addressing a critical issue in the field of healthcare and medical research, specifically focused on diabetes prediction. Diabetes is a prevalent and chronic medical condition that affects millions of individuals worldwide. Early detection and accurate prediction of diabetes are vital for timely intervention and improved patient outcomes.

The Diabetes Prediction dataset is a valuable resource comprising medical and demographic data from patients, along with their diabetes status (positive or negative). This dataset includes a range of essential features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. These factors play a crucial role in assessing an individual's risk of developing diabetes.

Our team has set out on a goal to create an effective machine learning model for diabetes prediction as a result of the expanding worldwide burden of diabetes and the requirement for more accurate diagnostic tools. In order for healthcare workers, researchers, and patients to properly utilize this model, we plan to build a web application that incorporates it.

The primary objectives of our solution are as follows:

- Early Diabetes Detection: Our machine learning model will analyze the patient data to predict the likelihood of an individual developing diabetes. This predictive capability can aid healthcare professionals in identifying high-risk patients and initiating preventive measures.

- Personalized Treatment: The model's predictions will assist in the development of personalized treatment plans for patients. Healthcare providers can tailor interventions based on an individual's risk profile and medical history, thereby improving treatment effectiveness.

- Research Insights: Researchers can utilize our dataset and machine learning model to explore the relationships between various medical and demographic factors and the likelihood of developing diabetes. This could lead to valuable insights into the disease's etiology and potential avenues for prevention and treatment.

- Patient Empowerment: Patients can also benefit from our solution by gaining a better understanding of their risk factors. They can proactively make lifestyle changes or seek medical advice if their risk of diabetes is elevated.

Scope of work

The main objective of this project is to provide a data mining solution for the above-mentioned real-world problem. The problem is related to predicting diabetes in patients based on their medical history and demographic information. Therefore, regression is the reliable method which is used as the data mining function to build models after applying specific algorithms. The optimal model owes the higher accuracy which is going to deploy as a web application.

As the final output, the optimal model will be developed and deployed as a single convenient web application using web development tools. The user possesses the ability to provide the required features and based on the user's requirements the model will predict the occurrence of diabetes. According to the timeline, the project will process and the time duration for every task of the project plan is mentioned below. After the deployment of the web application, a final report will be submitted at the end.

Choice of Technology

Python

Python is a high-level, versatile, and interpreted programming language known for its simplicity and readability. Python is a great choice for both beginning and advanced programmers because of its emphasis on code readability and clear syntax.

Streamlit

An open-source Python framework called Streamlit makes it easier to build interactive web applications for data science and machine learning. Developers and data scientists may easily convert data scripts into shareable web apps with Streamlit. It offers a simple and intuitive approach for anyone to create web applications using Python code.

Methodology

Jupyter Notebook was used to create the regression models required for the process of evaluation. The project began with the selection of a specific dataset via online, later supervised by the lecturer. After the selection of the dataset, it was imported, and the data was preprocessed. Later the Exploratory Data Analysis was done. Eventually the model was built. As this dataset was based on a regression model, we used 4 models to evaluate and get a more accurate result.

- Support vector machine
- Decision Tree
- Random forest
- Logistic Regression

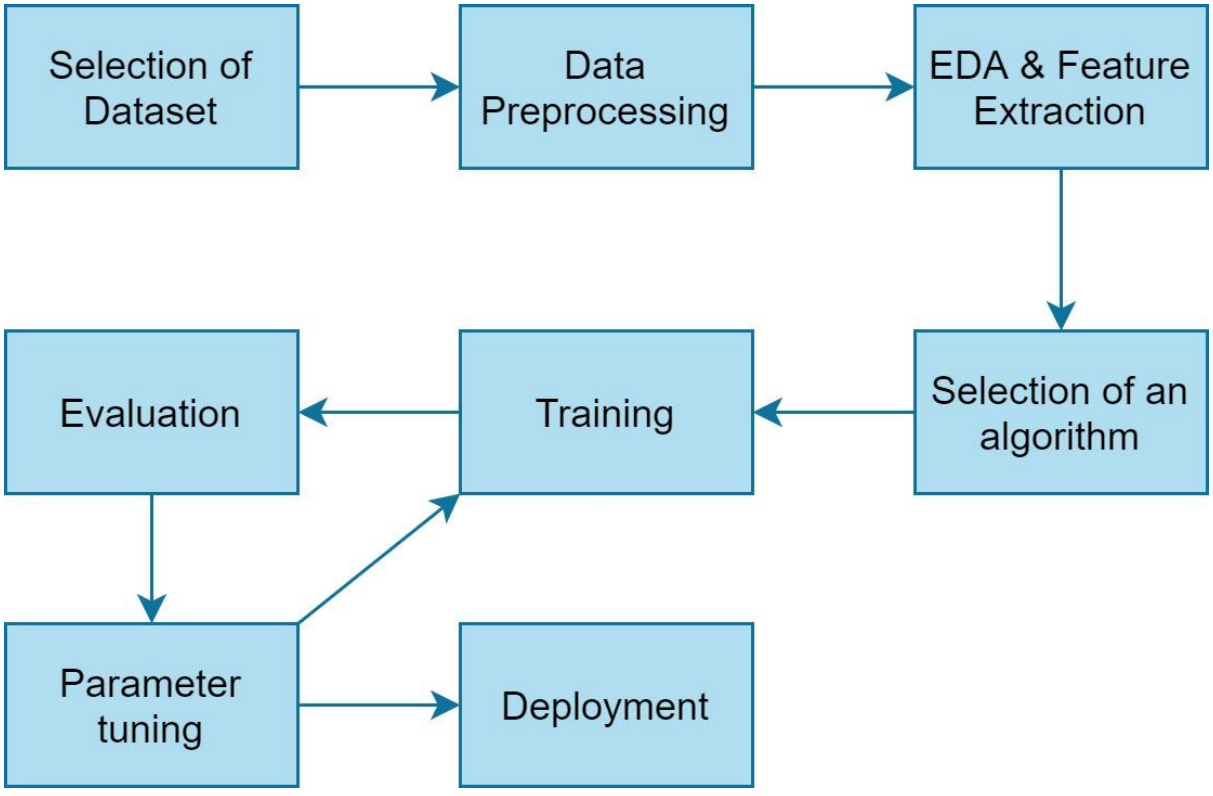

Figure 1: Process

Dataset Introduction

Diabetes Prediction Dataset - https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset

Patients' medical and demographic information, as well as their diabetes status (positive or negative), are included in the Diabetes prediction dataset. The data includes features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. The dataset consists of 9 columns and 100000 rows.

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
5	Female	20.0	0	0	never	27.32	6.6	85	0
6	Female	44.0	0	0	never	19.31	6.5	200	1
7	Female	79.0	0	0	No Info	23.86	5.7	85	0
8	Male	42.0	0	0	never	33.64	4.8	145	0
9	Female	32.0	0	0	never	27.32	5.0	100	0

Figure 2:Dataset values

check count of rows with 'No info' in 'smoking_history' column

```
count_no_info = (dataset['smoking_history'] == 'No Info').sum()

print(f"Count of rows with 'No info' in 'Smoking history' column: {count_no_info}")
```

```
Count of rows with 'No info' in 'Smoking history' column: 32887
```

check for any missing or null value

```
dataset.isnull().values.any()
```

```
False
```

Figure 3:Null values and No info values

Data Preprocessing and Transformation

Data Cleaning

As a part of data preparation, data preprocessing refers to any type of processing done on raw data to get it ready for another data processing technique. It has long been regarded as a crucial first stage in the data mining process. This is the process of making data usable and accurate by fixing or removing any errors present on the dataset.

1. Removing irrelevant data

   Removal of redundant data and irrelevant observations. During data gathering, repetitive observations occur regularly, and irrelevant observations are ones that don't directly relate to the issue being addressed. Redundant data significantly reduces productivity and increases the risk of falsified outcomes.

```
### Check duplicate values
```

```
duplicate = dataset.duplicated()
```

```
print(duplicate)
```

```
0        False
1        False
2        False
3        False
4        False
         ...
99995     True
99996    False
99997    False
99998    False
99999    False
Length: 100000, dtype: bool
```

```
### Remove duplicate data
```

```
dataset = dataset.drop_duplicates()
dataset
```

Figure 4:Checking and removing duplicate values.

2. Structural error fixation

   Typos in feature names, incorrect class labels, the same attribute with a different name, or uneven capitalization are all examples of structural faults.

3. Handling missing data

   Raw datasets normally consist of missing data due to several factors. There are many ways to work with missing data. Here in the figure below we have used removing of rows containing missing data.

drop 'No Info' rows

```
dataset.drop(dataset[dataset['smoking_history'] == 'No Info'].index, inplace=True)
```

Figure 5:Dropping 'No info' rows.

4. Outliers

   During the model's build, outliers are very important. These will directly affect the accuracy of the model. They may result from mistakes made during data entry or gathering. They may bias the model's findings and produce incorrect predictions.

Feature Engineering

- The amount of diabetes patients available on the data set is taken with count variables and displayed as a bar chart and pie chart to get a better understanding of ratios. These visualizations show that there are binary variants of 1 and 0 (1 for patients with diabetes and 0 for patients with no diabetes).

get a count of the number of diabetes patients

```
dataset['diabetes'].value_counts()
```

```
0    56222
1     7037
Name: diabetes, dtype: int64
```

Figure 6:Getting counts.

```
sns.countplot(x='diabetes', data=dataset)

plt.xlabel('diabetes')
plt.ylabel('Count')
plt.title('diabetes in patients')

plt.show()
```

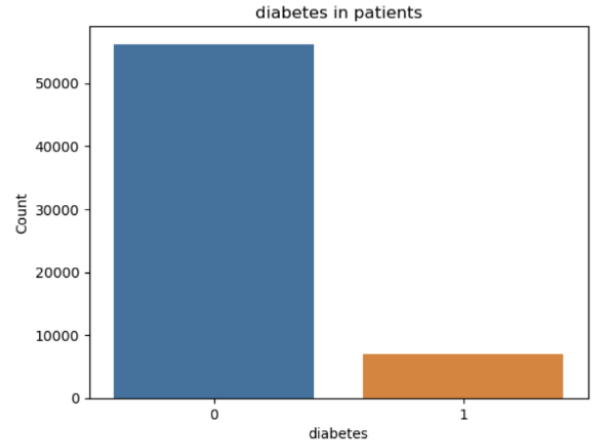

Figure 7:Bar chart for diabetes patients

```
dataset.diabetes.value_counts().plot(kind = 'pie', autopct = '%1.1f%%',explode = [0, 0.1], shadow = True)
plt.title('Distribution of Diabetes')
plt.show()
```

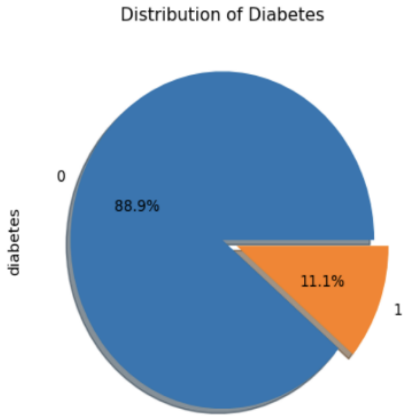

Figure 8:Pie chart for diabetes patients

- The following shows the distribution of gender and diabetes. Only 2 genders are found on the dataset and both these categories are affected by diabetes.

Distribution of Diabetes status within Gender

```
sns.countplot(x="gender", data=dataset, hue="diabetes")

plt.title("Distribution of Diabetes status within Gender")
plt.xlabel("Gender")
plt.ylabel("Count")

# Show the plot
plt.show()
```

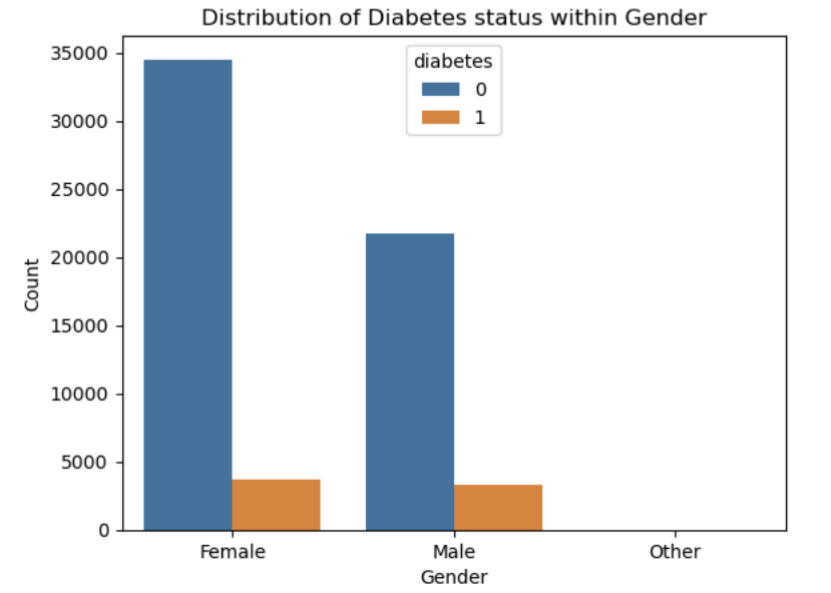

Figure 9: Diabetes status with gender

- The following graphic shows a bar chart of the distribution of diabetes during a heart disease.

Distribution of Diabetes status within Heart Disease

```
sns.countplot(x="heart_disease", hue = 'diabetes',data=dataset)

plt.title("Distribution of Diabetes status within Heart Disease")
plt.xlabel("Heart Disease")
plt.ylabel("Count")

plt.show()
```

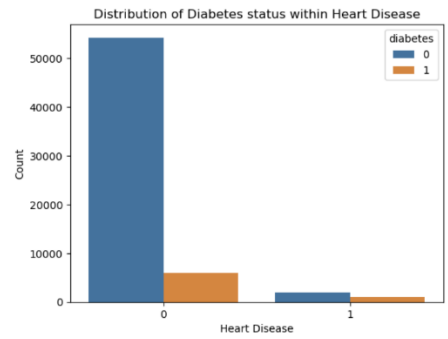

Figure 10: Diabetes status with heart disease

- The following bar chart shows the distribution of diabetes considered with the habit of smoking. There are 5 categories present in the smoking feature (never, current, former, ever, and not current). These categories are helpful to get a clear understanding on smoking and diabetes relationship.

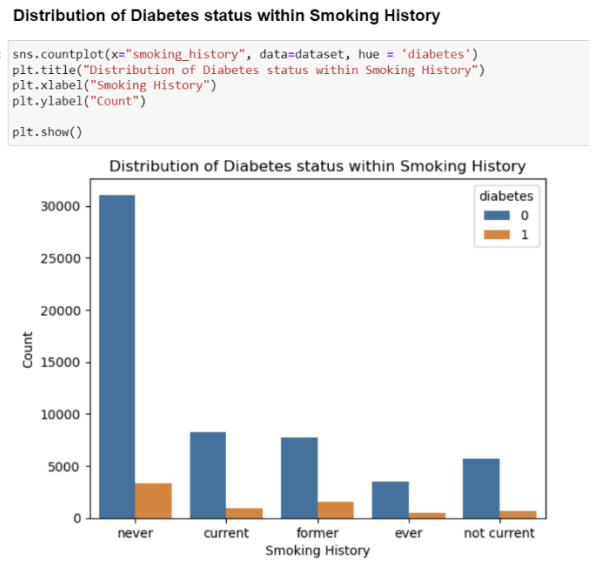

Figure 11:Diabetes status with smoking history

- This set of graphs gives a visualization of patients with diabetes. The columns with no diabetes have been dropped on each of these graphs.

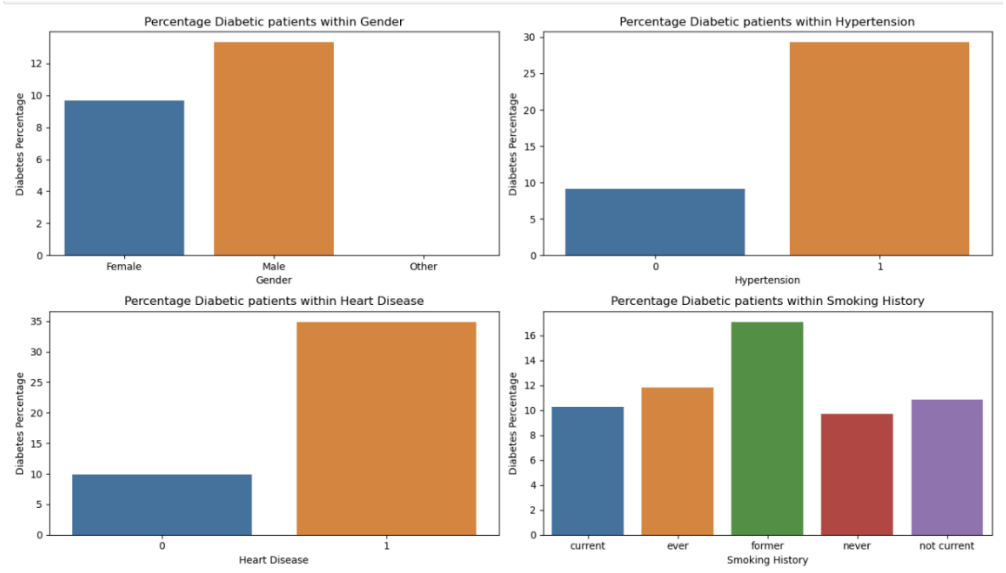

Figure 12:Diabetes patient visualizations

- Distribution of numerical features are clearly visible in these graphs. As you can see in the age graph there is a high number of patients belonging to the 80s and similar patterns can be seen on the other graphs as well.

Distribution of numeric features

```
: # Create subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# List of numeric feature columns
numeric_features = ['age', 'bmi', 'HbA1c_level', 'blood_glucose_level']

# Loop through numeric features and plot them
for i, feature in enumerate(numeric_features):
    row = i // 2
    col = i % 2

    sns.histplot(dataset[feature], ax=axes[row, col])
    axes[row, col].set_title(f'{feature}')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```

Figure 13: Numeric features

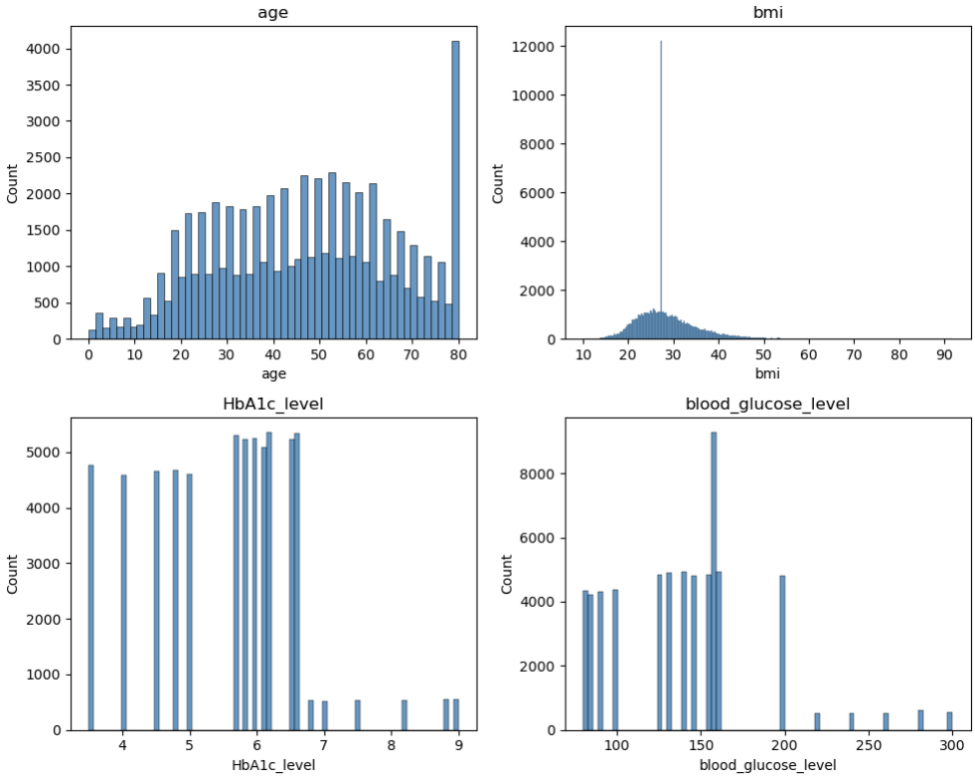

Figure 14: Numeric feature graphs

- All the columns containing unique values and their counts are displayed via this code.

print all of the columns and their unique values

```
for column in dataset.columns:
    if dataset[column].dtype == 'object':
        print(str(column) + ' : ' + str(dataset[column].unique()))
        print(dataset[column].value_counts())
        print('----------------------------------------------------------------------------------------')
```

```
gender : ['Female' 'Male' 'Other']
Female    38192
Male      25055
Other        12
Name: gender, dtype: int64
----------------------------------------------------------------------------------------
smoking_history : ['never' 'current' 'former' 'ever' 'not current']
never          34398
former          9299
current         9197
not current     6367
ever            3998
Name: smoking_history, dtype: int64
----------------------------------------------------------------------------------------
```

Figure 15: Unique values

- The correlation between all the categories is displayed in the below heat map.

visualize the correlation

```
plt.figure(figsize= (5,5))
sns.heatmap(dataset.corr(), annot=True, fmt= '.0%')
```

```
<AxesSubplot:>
```

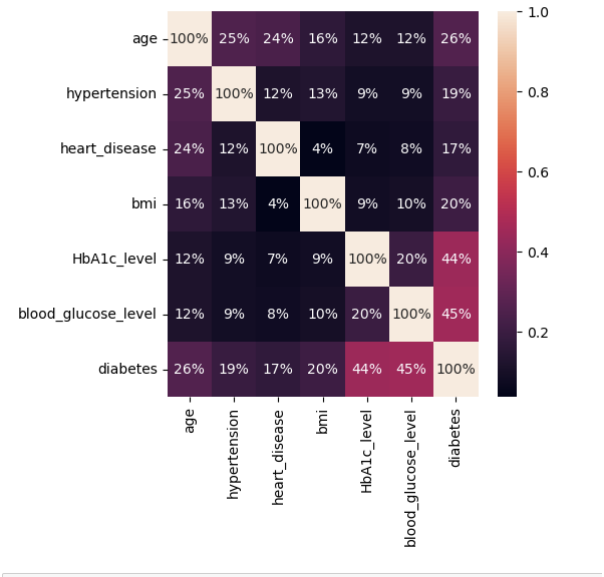

Figure 16: Correlation visualization

- Data after being preprocessed and feature engineering process.

```
# Define a mapping for Gender
gender_mapping = {'Male': 0, 'Female': 1}

# Use the map function to convert Gender column
dataset['gender'] = dataset['gender'].map(gender_mapping)

# Define a mapping for smoking history
smoking_history_mapping = {'never': 0, 'former': 1 , 'current':2 , 'not current' : 3 , "ever" : 4}

# Use the map function to convert Gender column
dataset['smoking_history'] = dataset['smoking_history'].map(smoking_history_mapping)
```

```
C:\Users\dulara\AppData\Local\Temp\ipykernel_2648\1829781541.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['gender'] = dataset['gender'].map(gender_mapping)
C:\Users\dulara\AppData\Local\Temp\ipykernel_2648\1829781541.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['smoking_history'] = dataset['smoking_history'].map(smoking_history_mapping)
```

```
dataset.head()
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	1	80.0	0	1	0	25.19	6.6	140	0
1	0	28.0	0	0	0	27.32	5.7	158	0
2	1	36.0	0	0	2	23.45	5.0	155	0
3	0	76.0	1	1	2	20.14	4.8	155	0
4	1	20.0	0	0	0	27.32	6.6	85	0

Figure 17: Preprocessed data

Model Implementation

Our main goal is to predict whether a certain patient claims to have diabetes or not. Since our data set consists of categorical data, we have come up with these models mentioned below. To enable machines to learn the relationships within the provided data and create predictions based on patterns or rules discovered from the dataset, we employed the following set of machine learning algorithms. Each algorithm has its own presumptions, benefits, and drawbacks.

Building of the Model

Testing and training sets of data are separated through this piece of code. 20% of the dataset is used as the test dataset, while 80% is used as the train dataset.

split data into training and testing

```
from sklearn.model_selection import train_test_split

X = dataset.drop('diabetes',axis='columns')
y = dataset['diabetes']


X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=5)
```

```
X_train.shape
```

```
(50597, 8)
```

```
y_test.value_counts()
```

```
0    11236
1     1414
Name: diabetes, dtype: int64
```

Figure 18: Data split

Random forest algorithm

The random forest algorithm is a technique for classification and regression applications that uses ensemble learning. A large number of decision trees are built during training, and after that, the most well-liked class or mean prediction of each tree is predicted.

- ✓ Assumptions
  - Features should be independent.

- Randomly sampled data from the population.

✓ Pros

- Robust to overfitting.
- High dimensional data is well handled.
- Accurate when compared to many other machine learning models.

✓ Cons

- Computationally expensive when it comes to large datasets.
- Understanding difficulties when it comes to prediction.

Model : Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model_1 = RandomForestClassifier(n_estimators=50) # trees
model_1.fit(X_train , y_train)
```

```
RandomForestClassifier(n_estimators=50)
```

```
model_1.score(X_test , y_test)
```

```
0.9586561264822134
```

```
y_predicted_1 = model_1.predict(X_test)
```

Figure 19:Random Forest code

Support vector machine algorithm

Support vector machine (SVM) can be applied to both classification and regression applications. It operates by locating a hyperplane in the data that most clearly divides the various classes.

✓ Assumptions

- Linearly separable data
- Data with higher accuracy

✓ Pros

- High dimensional data is well handled.

- Accurate when compared to many other machine learning models.

✓ Cons

- Computationally expensive when it comes to large datasets.
- Sensitive to outliers
- Understanding difficulties when it comes to prediction.

Model : SVM Classifier

```
from sklearn.svm import SVC
model_4 = SVC()
```

```
model_4.fit(X_train , y_train)
```

```
SVC()
```

```
model_4.score(X_test , y_test)
```

```
0.9310671936758893
```

```
# confusion metrix
y_predicted_4 = model_4.predict(X_test)
cm4 = confusion_matrix(y_test , y_predicted_4)
print(cm4)
```

```
[[11236     0]
 [  872   542]]
```

Figure 20: SVM code

Decision tree algorithm

All data that contains numerical and categorical attributes can be used with decision tree models. The non-linear interactions between the features and the goal variable are well-captured by decision trees. Decision trees are fairly similar to how people think, making it easy to grasp the facts.

A decision tree is a tree where each node corresponds to a feature and each branch to a conclusion.

✓ Assumptions

- Data independence
- Identical distribution

✓ Pros

- Denormalized data can be used.
- Requires less data preparation time.
- Missing values does not severely affect the prediction.

✓ Cons

- Computationally expensive when it comes to large datasets.
- Small changes, large impact
- Not a good model for regression tasks

Model : Decision Tree

```
from sklearn import tree

model_3 = tree.DecisionTreeClassifier()
```

```
model_3.fit(X_train , y_train)
```

```
DecisionTreeClassifier()
```

```
model_3.score(X_test , y_test)
```

```
0.9354150197628458
```

```
# confusion metrix
y_predicted_3 = model_3.predict(X_test)
cm3 = confusion_matrix(y_test , y_predicted_3)
print(cm3)
```

```
[[10813   423]
 [  394  1020]]
```

Figure 21: Decision tree code

Logistic regression model

To determine the likelihood that an event will succeed or fail, the classification algorithm logistic regression is utilized. This used mainly with binary data.

✓ Assumptions

- Data should be linearly separable.

✓ Pros

- Easy implementation
- Performs well with low dimensional data.

✓ Cons

- Overfit with high dimensional data
- Complex relationships cannot be detected easily.

Model: Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model_2 = LogisticRegression()
```

```
model_2.fit(X_train , y_train)
```

Figure 22: Logistic Regression model

User Interface

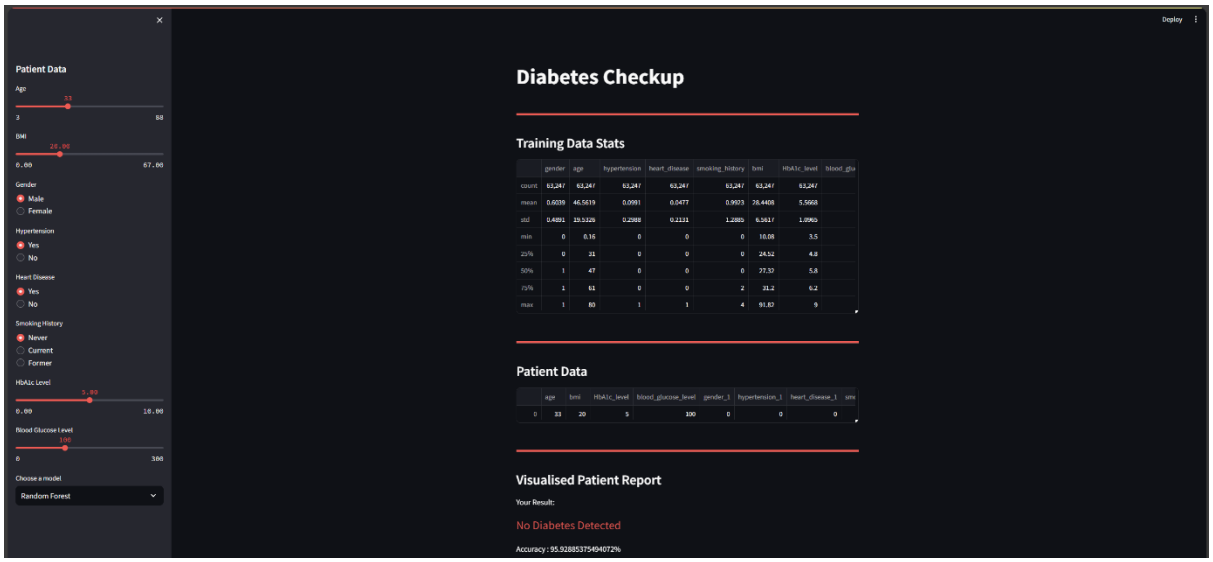

Figure 23: Interface view 1

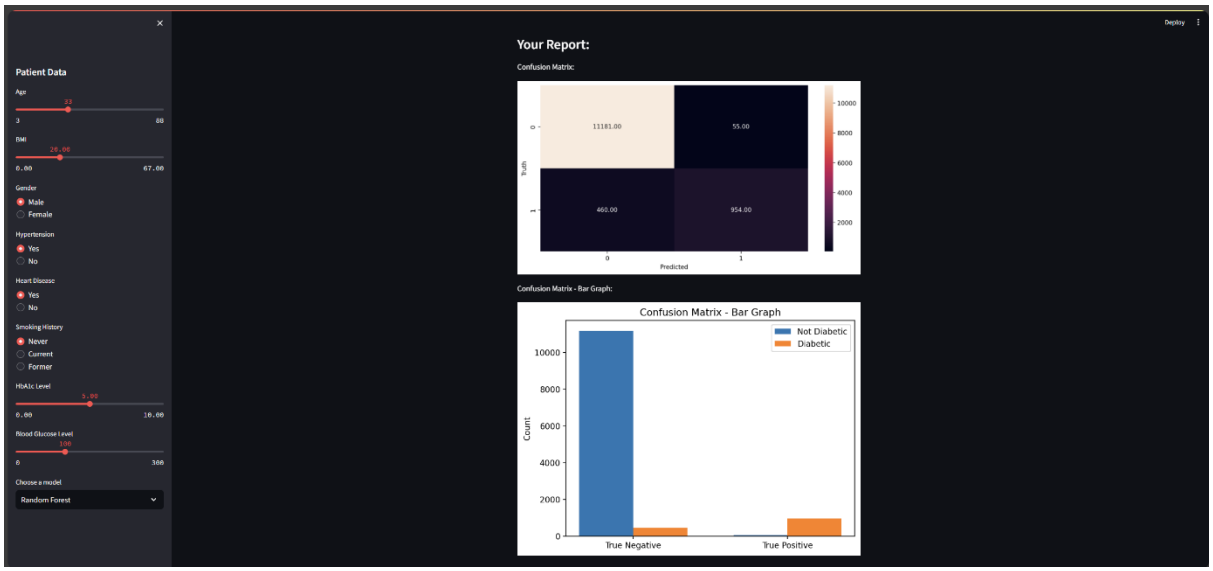

Figure 24: Interface view 2

Deliverables

The main objective of this web application is to predict the status of diabetes in a patient considering certain health aspects. The user will have to input data related to the day today habitual activities in order to get a prediction. This will help the health community and the society in many dimensions.

References

- https://www.geeksforgeeks.org/basic-concept-classification-data-mining/
- https://datatrained.com/post/classification-algorithms-in-data-mining/
- https://www.upgrad.com/blog/classification-in-data-mining/
- https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm