INSTITUTE OF TECHNOLOGY OF CAMBODIA

Department of applied mathematics and statistics

Project report of Introduction to Data Science

Group AMS

Topic: Using Machine Learning to Predict Type 2 Diabetes: A Data-Driven Approach to Early Diagnosis

Lecture: Mr. PHAUK Sokkhey (Cours)

Mr. PEN Chentra (TP)

Name of Student	ID of Student	Score
NGEN Tina	e20221516	…….
NOB Sreynich	e20220741	…….
NEANG Vanna	e20221650	…….
OENG Kimthai	e20220182	…….
NOV Panhavath	e20221643	…….
SOPHAT Vitou	e20220813	…….

Academic Year 2024-2025

Content

Abstract

This project aims to build a practical system that can help identify individuals at risk of Type 2 Diabetes at an early stage, using patterns and trends found in everyday health data like age, weight, family history, and blood sugar levels. Moreover, we applied machine learning to predict type 2 diabetes based on 15 attributes and used data from the open source on the internet from health data for 5437 patients, which was freely available online. As a result, by using machine learning, we found that there are 344 (228 female and 116 male), or 6.3%, who were diagnosed with diabetes. Additionally, this study is significant for the health system by providing accurate predictions that are easy to understand and can be used by healthcare professionals by offering clear insights that can support better decision-making.

1. INTRODUCTION

1.1. Problem Statement

Type 2 Diabetes is a long-term health condition that affects millions of people worldwide and can cause serious health problems if it is not found and treated early. Unfortunately, even with improvements in healthcare, many people are not diagnosed until the condition has already caused significant damage. This makes early detection very important. Currently, diagnosing diabetes often involves invasive tests, complex lab procedures, and expert analysis, which can slow down the process. However, with the growing amount of health data available, we now have an opportunity to use technology, like machine learning, to predict whether someone is at risk of developing Type 2 Diabetes based on their age, or gender, pulse rate, blood pressure (systolic and diastolic, glucose level, BMI and family history of diabetes and related conditions like hypertension and cardiovascular disease.

1.2. Objective

The goal of this project is to develop a practical system that can help identify individuals at risk of Type 2 Diabetes at an early stage, using patterns and trends found in everyday health data like age, weight, family history, and blood sugar levels. The project involves several steps, starting with gathering and cleaning the data to ensure its accurate and useful. The next step is to explore the data deeply, looking for important connections and features that play a key role in predicting diabetes risk. Using this knowledge, we will design and test a range of predictive tools based on machine learning, comparing their performance to find the most reliable and accurate model. The final system should not only provide accurate predictions but also be easy to understand and use by healthcare professionals, offering clear insights that can support better decision-making. This approach aims to simplify the process of spotting potential cases, making it faster and more efficient to help patients get the care they need sooner, reducing the long-term impact of the disease.

2. DATA COLLECTION

2.1. Data Collection

Data Collection refers to the systematic process of gathering, measuring, and analysing information to answer research questions, solve problems, or evaluate outcomes. It is a fundamental step in research, business, science, and decision-making processes.

For the project of "**Classification Diabetes**", data collection needs to have important information about the disease such as:

- Patient information (Demographics: age, gender, …)
- Medical history (past illnesses, allergies, and reactions,...)
- Clinical measurements (Blood glucose levels, Body measurements, insulin levels.)
- Medications (current medications, side effects of medication)
- Symptoms (Fatigue, excessive thirst, frequent urination, blurred vision, slow healing wounds)
- Health behaviours (smoking, alcohol, dietary habits)
- Diagnosis information (lab result, type 1 or type 2 diabetes,...)

2.2. Data Cleaning

Data cleaning is the process of identifying, correcting, or removing errors, inconsistencies, inaccuracies, and discrepancies in a dataset. It is a critical step in the data preprocessing pipeline and must be completed before performing any analysis or modeling.

The primary goal of data cleaning is to ensure the dataset is accurate, consistent, and reliable, making it suitable for analysis. This process involves addressing various issues commonly found in real-world data, such as missing values, duplicate entries, outliers, and formatting errors.

2.3. Data Extraction

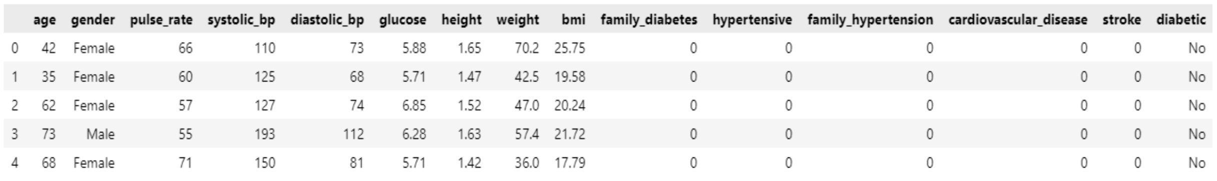
The dataset was collected from open source on the internet. It contains comprehensive health data for 5437 patients, including 14 independent attributes such as demographics, clinical parameters, and medical history.

The dataset originally had 5437 rows and 15 columns (features + target). The 12 features are equipped with meaning as follows:

NO.	ATTRIBUTES	DESCRIPTIONS
1	age	The individual's age in years.
2	gender	The individual's gender (male, female, or other).
3	pulse_rate	Heartbeats per minute.
4	systolic_bp	Blood pressure during heart contraction.
5	diastolic_bp	Blood pressure during heart rest (mmHg).
6	glucose	Blood sugar level (mg/dL).
7	height	Height measurement (cm).
8	weight	Body mass (kg).
9	bmi	Body Mass Index: Weight (kg) divided by height (m^2).
10	family_diabetes	Family history of diabetes.
11	hypertensive	Status of having high blood pressure.
12	family_hypertension	Family history of high blood pressure.
13	cardiovascular_disease	Disorders affecting the heart and blood vessels.
14	stroke	Condition from interrupted blood flow to the brain.
15	diabetic	Status of having diabetes.

Figure 1: Attributes of Datasets

The value of the first 5 rows of datasets



	age	gender	pulse_rate	systolic_bp	diastolic_bp	glucose	height	weight	bmi	family_diabetes	hypertensive	family_hypertension	cardiovascular_disease	stroke	diabetic
0	42	Female	66	110	73	5.88	1.65	70.2	25.75	0	0	0	0	0	No
1	35	Female	60	125	68	5.71	1.47	42.5	19.58	0	0	0	0	0	No
2	62	Female	57	127	74	6.85	1.52	47.0	20.24	0	0	0	0	0	No
3	73	Male	55	193	112	6.28	1.63	57.4	21.72	0	0	0	0	0	No
4	68	Female	71	150	81	5.71	1.42	36.0	17.79	0	0	0	0	0	No

Figure 2: First 5 rows of the data

3. EXPLORATORY DATA ANALYSIS

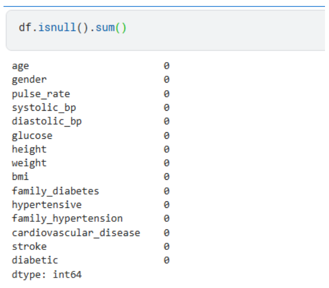
Exploratory Data Analysis (EDA) is a crucial initial step in any data science project. It involves a thorough examination and visualization of data to uncover key characteristics, patterns, and potential insights. By summarizing data, identifying trends, and detecting anomalies. Common techniques include creating histograms, scatter plots, and calculating summary statistics. The insights gained from EDA guide researchers in leading to more informed and accurate conclusions.

3.1. Data Cleaning

Data cleaning, also known as data cleansing, is the process of identifying and correcting or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. It is a crucial step in the data science process because dirty data can lead to inaccurate results and misleading conclusions.

3.1.1. Handling Missing Value

This can involve imputing missing values using various methods, such as mean imputation or regression imputation, or removing rows or columns with missing values.



```
df.isnull().sum()
```

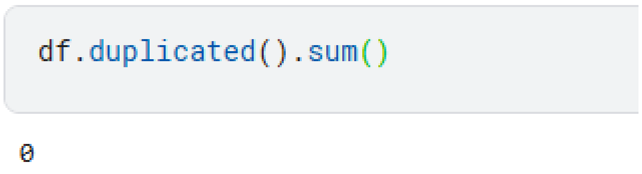
```
age                      0
gender                   0
pulse_rate               0
systolic_bp              0
diastolic_bp             0
glucose                  0
height                   0
weight                   0
bmi                      0
family_diabetes          0
hypertensive             0
family_hypertension      0
cardiovascular_disease   0
stroke                   0
diabetic                 0
dtype: int64
```

Figure 3: Missing Data

As shown in the result, we have no missing data.

3.1.2. Removing duplicates

Duplicate data can skew results and should be removed, so checking duplicated data is also mandatory.



```
df.duplicated().sum()
```

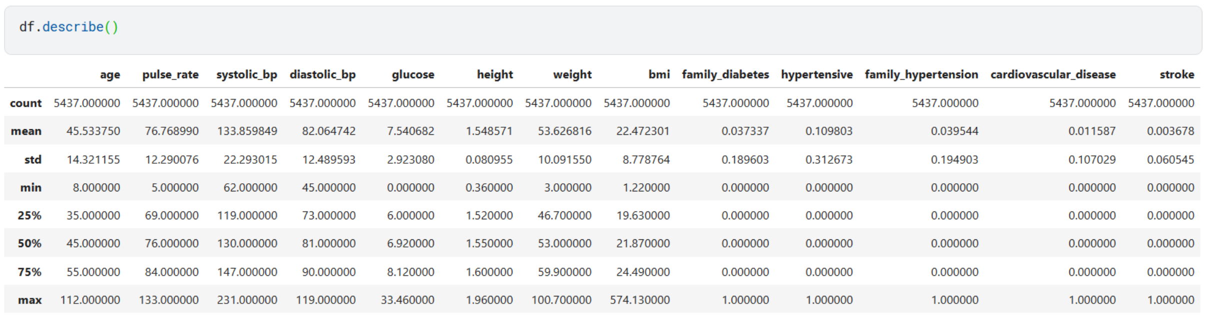
```
0
```

Figure 4: Number of Duplicate Data

Also, there is no duplicated data.

3.2. Summary Statistics

Summary statistics are numerical values that provide a concise summary of a dataset. They help to understand the main characteristics of the data, such as its central tendency, dispersion, and shape.



```
df.describe()
```

	age	pulse_rate	systolic_bp	diastolic_bp	glucose	height	weight	bmi	family_diabetes	hypertensive	family_hypertension	cardiovascular_disease	stroke
count	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000	5437.000000
mean	45.533750	76.768990	133.859849	82.064742	7.540682	1.548571	53.626816	22.472301	0.037337	0.109803	0.039544	0.011587	0.003678
std	14.321155	12.290076	22.293015	12.489593	2.923080	0.080955	10.091550	8.778764	0.189603	0.312673	0.194903	0.107029	0.060545
min	8.000000	5.000000	62.000000	45.000000	0.000000	0.360000	3.000000	1.220000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	35.000000	69.000000	119.000000	73.000000	6.000000	1.520000	46.700000	19.630000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	45.000000	76.000000	130.000000	81.000000	6.920000	1.550000	53.000000	21.870000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	55.000000	84.000000	147.000000	90.000000	8.120000	1.600000	59.900000	24.490000	0.000000	0.000000	0.000000	0.000000	0.000000
max	112.000000	133.000000	231.000000	119.000000	33.460000	1.960000	100.700000	574.130000	1.000000	1.000000	1.000000	1.000000	1.000000

Figure 5: Summary Statistics of Datasets

3.3. Data Visualization

Data visualization transforms raw data into meaningful visual representations, enabling us to see patterns, trends, and insights that might otherwise remain hidden. By utilizing charts, graphs, and maps, data visualization simplifies complex information, making it more accessible and understandable to a wider audience. This visual approach facilitates faster comprehension, improved decision-making, and more effective communication of key findings.

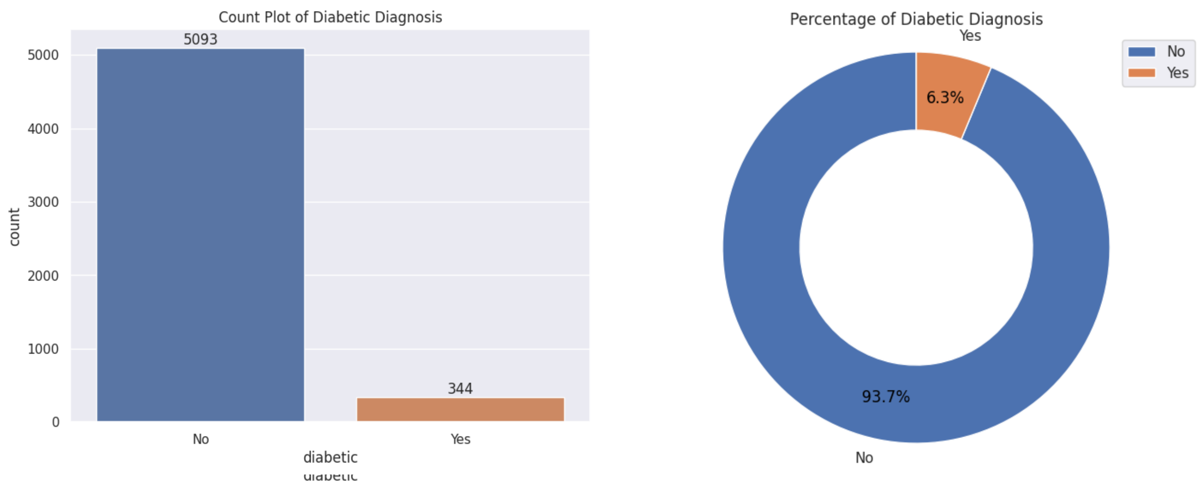


Figure 6: Count Plot and Percentage of Diabetic Diagnosis

Based on data, the above graph shows that the number of those who are NOT Diabetic outnumbers that of who are NOT, relative to all the records.

Diabetic Diagnosis by Genders

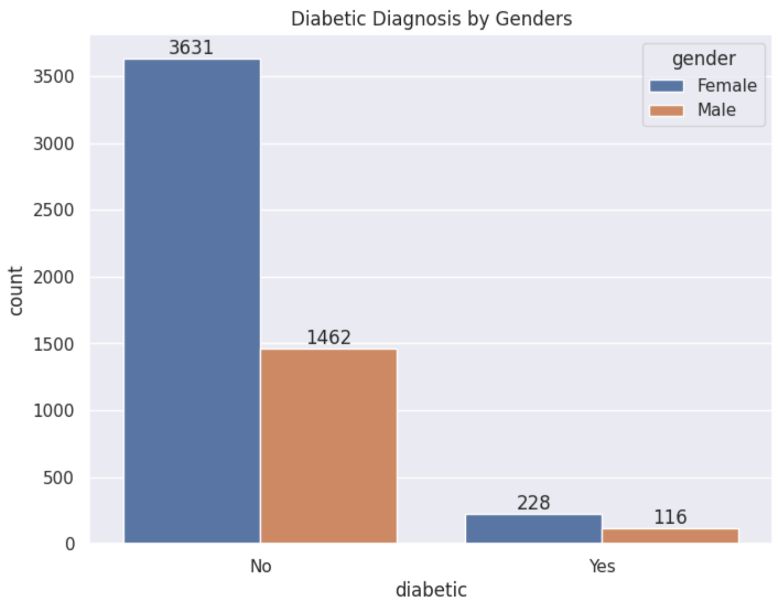


Figure 7: Number of Diagnosis based on Genders

Histogram of Numerical Data

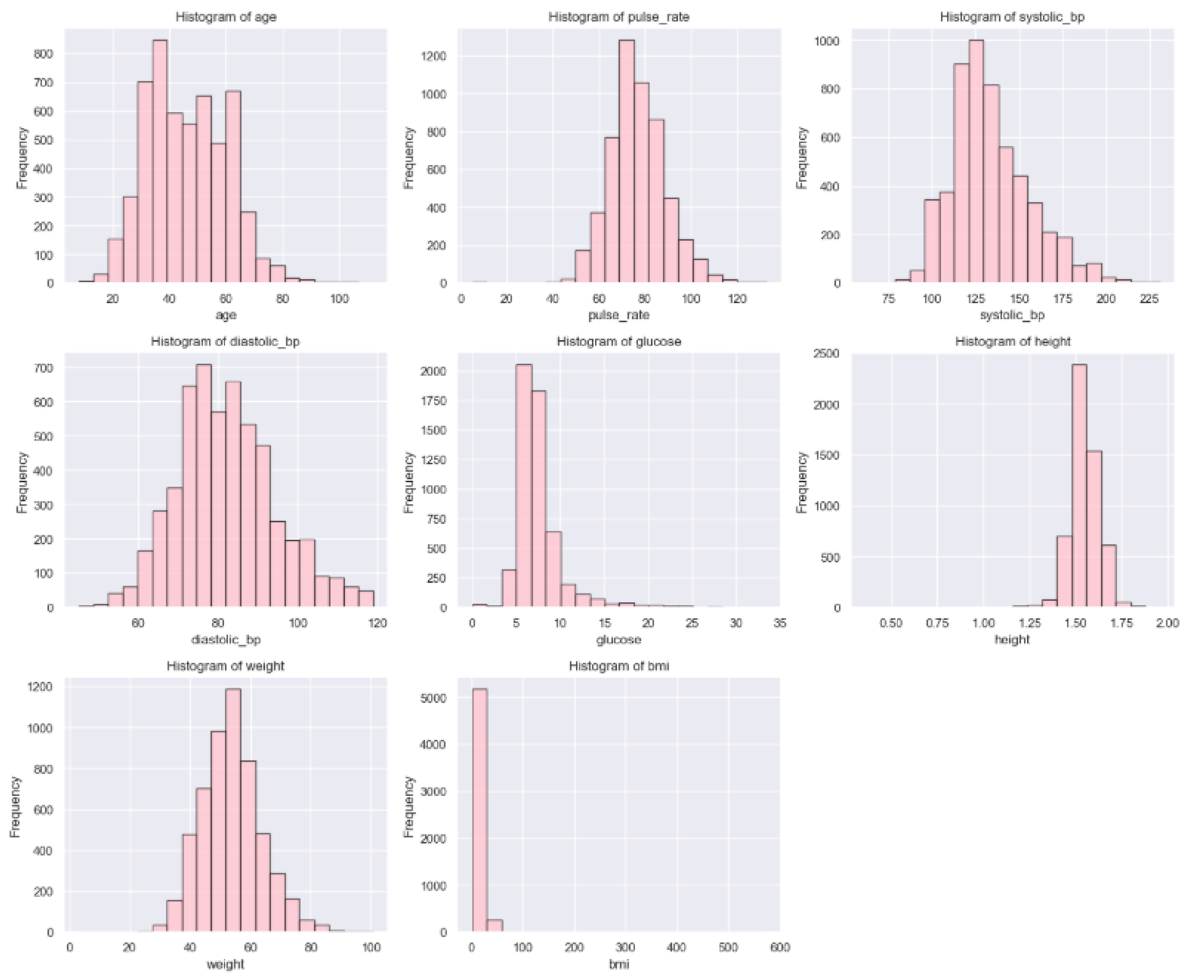


Figure 8: Histogram of Numerical Data

As shown on the histograms that are graphical representation of the distribution of numerical data, almost all the numerical features tend to be Normally Distributed (Normal Distribution); however, the *distribution of bmi* (Body Mass Index) is quite *right-skewed.* To be a bit clearer, let’s plot it out:

The histogram of BMI

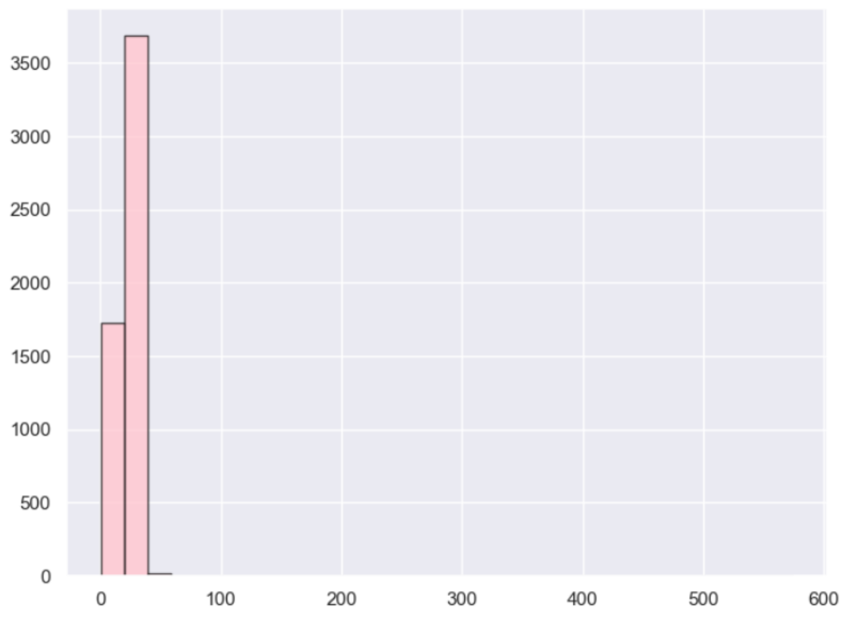


Figure 9: Distribution of BMI

3.3.1. Data Transformation

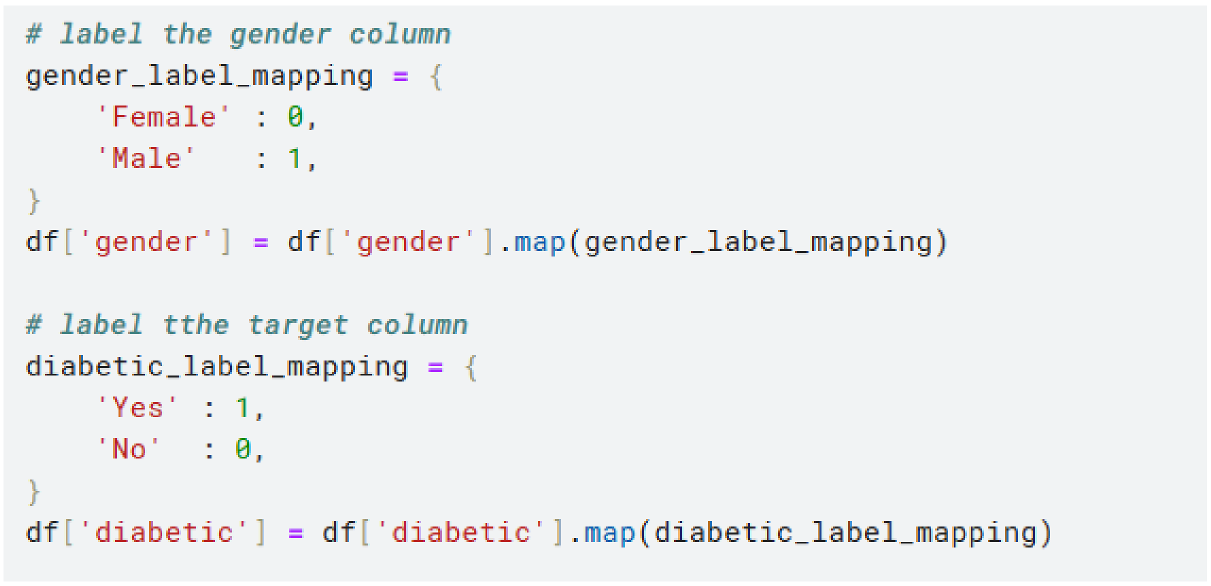
Manual Label Encoding

Because machines know only numbers, we thus need to represent all string data with some sorts of arbitrary numbers, *label encoding* is a technique used in machine learning and data preprocessing to convert categorical variables into numerical variables. It assigns a unique numerical value to each category in the dataset.

In our case, the Tags field in the dataset description is likely a categorical variable that has not been label encoded. This means that each tag (e.g., "gander", "diabetic") will be assigned a unique numerical value.

Gender: Male: 1, Female: 0

Diabetic: No: 0, Yes: 1



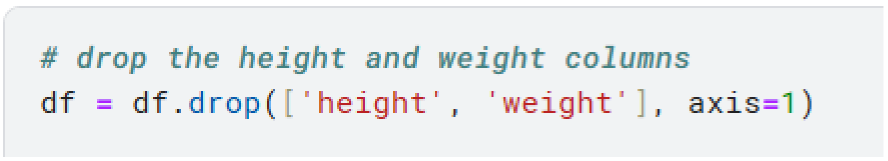
```python
# label the gender column
gender_label_mapping = {
    'Female' : 0,
    'Male'   : 1,
}
df['gender'] = df['gender'].map(gender_label_mapping)

# label tthe target column
diabetic_label_mapping = {
    'Yes' : 1,
    'No'  : 0,
}
df['diabetic'] = df['diabetic'].map(diabetic_label_mapping)
```

Figure 10: Labeling Categorical Data

3.3.2. Domain Knowledge Application

Drop Redundant Columns

Body mass index (BMI) is a value derived from the mass (weight) and height of a person. Thus, technically, BMI alone can describe the two variables. By that, dropping height and weight is recommended for redundant feature reduction.



```python
# drop the height and weight columns
df = df.drop(['height', 'weight'], axis=1)
```

Figure 11: Dropping Redundant Columns

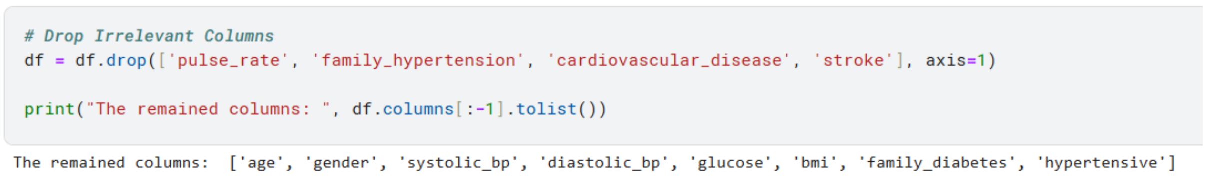
Drop Irrelevant Columns

We also decided to exclude other features from the classification of Type 2 Diabetes:

+ **Pulse Rate (`pulse_rate`):**
    - **Reason for Dropping:** Pulse rate is more indicative of cardiovascular or fitness-related health and is not strongly linked to the pathophysiology or risk of Type 2 Diabetes. While it might be indirectly affected by diabetes through complications like autonomic neuropathy, it is not a primary predictive feature for diabetes.

- **References:** Carnethon, M. R., et al. (2003). "Resting heart rate in middle age and diabetes development in older age." Diabetes Care (Link). The Study suggests that *heart rate changes are a consequence of diabetes complications rather than a predictor of its onset*.

+ **Family History of Hypertension (`family_hypertension`):**
    - **Reason for Dropping:** Hypertension is a risk factor for diabetes, but family history of hypertension does not directly predict Type 2 Diabetes. Instead, hypertensive status (already included) provides stronger and more direct predictive value.
    - **References:** Dewi Nurhanifah, et.al (2023). "Family History and Hypertension with Incidence of Diabetes Mellitus Type II." Jurnal eduhealth, Volume 14. (Link) *significantly highlights that while there is a connection between family history of hypertension and Type 2 Diabetes, it emphasizes the importance of other factors, such as diet and lifestyle, in the development of diabetes*.
+ **Cardiovascular Disease (`cardiovascular_disease`):**
    - **Reason for Dropping:** Cardiovascular disease (CVD) is more of an outcome or complication of Type 2 Diabetes rather than a predictor. Including it could lead to data leakage if using a dataset where CVD status arises from diabetes.
    - **References:** Low Wang, C. C., et al. (2016). "Cardiovascular disease in diabetes." Diabetes Care. (Link) *discusses the bidirectional relationship between diabetes and CVD but highlights that CVD is typically a consequence, not a predictor*.
+ **Stroke (`stroke`):**
    - **Reason for Dropping:** Similar to cardiovascular disease, stroke is a complication often resulting from poorly managed diabetes and associated conditions (e.g., hypertension). It is not a primary predictor of diabetes onset.
    - **References:** Emerging Risk Factors Collaboration (2010). "Diabetes mellitus, fasting glucose, and risk of cause-specific death." New England Journal of Medicine. (Link) *suggests stroke as a complication associated with diabetes rather than an indicator of its development*.



```
# Drop Irrelevant Columns
df = df.drop(['pulse_rate', 'family_hypertension', 'cardiovascular_disease', 'stroke'], axis=1)

print("The remained columns: ", df.columns[:-1].tolist())
```

```
The remained columns:  ['age', 'gender', 'systolic_bp', 'diastolic_bp', 'glucose', 'bmi', 'family_diabetes', 'hypertensive']
```

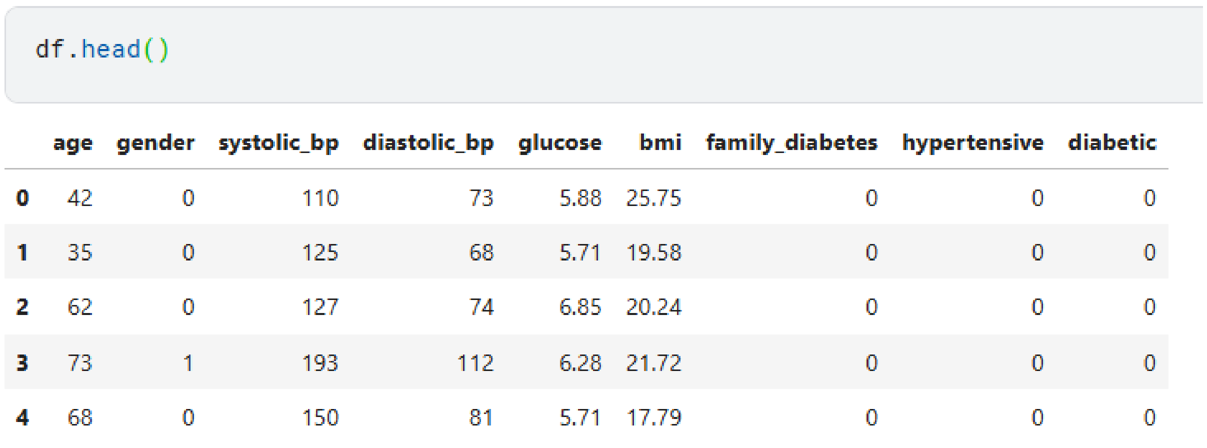
Figure 12: Drop Irrelevant Columns

3.3.3. Clearing Outlier

Clearing outliers refers to the process of identifying and removing or adjusting data points that are significantly different from the rest of the dataset. Outliers can distort statistical analyses and machine learning models, leading to inaccurate results.

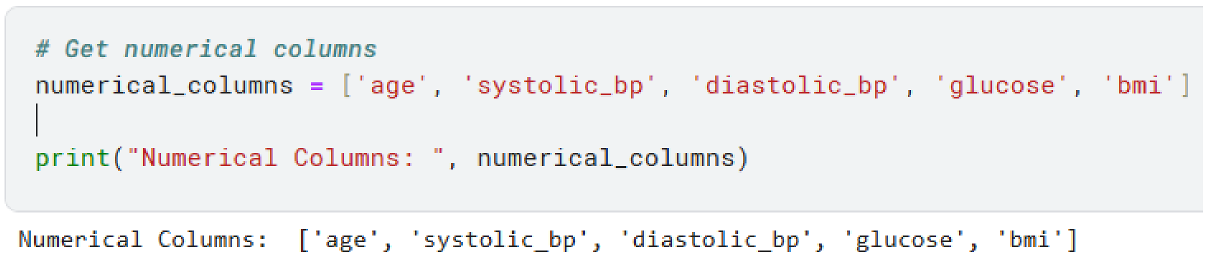
Checking Outlier via Boxplots

Clearing outliers is normally performed on *numerical data*; thus, we first need to specify the numerical columns and inspect the box plots of the data to see the outliers in the following step.



```
df.head()
```

	age	gender	systolic_bp	diastolic_bp	glucose	bmi	family_diabetes	hypertensive	diabetic
0	42	0	110	73	5.88	25.75	0	0	0
1	35	0	125	68	5.71	19.58	0	0	0
2	62	0	127	74	6.85	20.24	0	0	0
3	73	1	193	112	6.28	21.72	0	0	0
4	68	0	150	81	5.71	17.79	0	0	0



```
# Get numerical columns
numerical_columns = ['age', 'systolic_bp', 'diastolic_bp', 'glucose', 'bmi']

print("Numerical Columns: ", numerical_columns)
```

```
Numerical Columns:  ['age', 'systolic_bp', 'diastolic_bp', 'glucose', 'bmi']
```

Figure 13: Getting Numerical Columns

In our dataset, there are only 5 numerical columns, such as `age`, `systolic_bp`, `diastolic_bp`, `glucose`, and `bmi`.

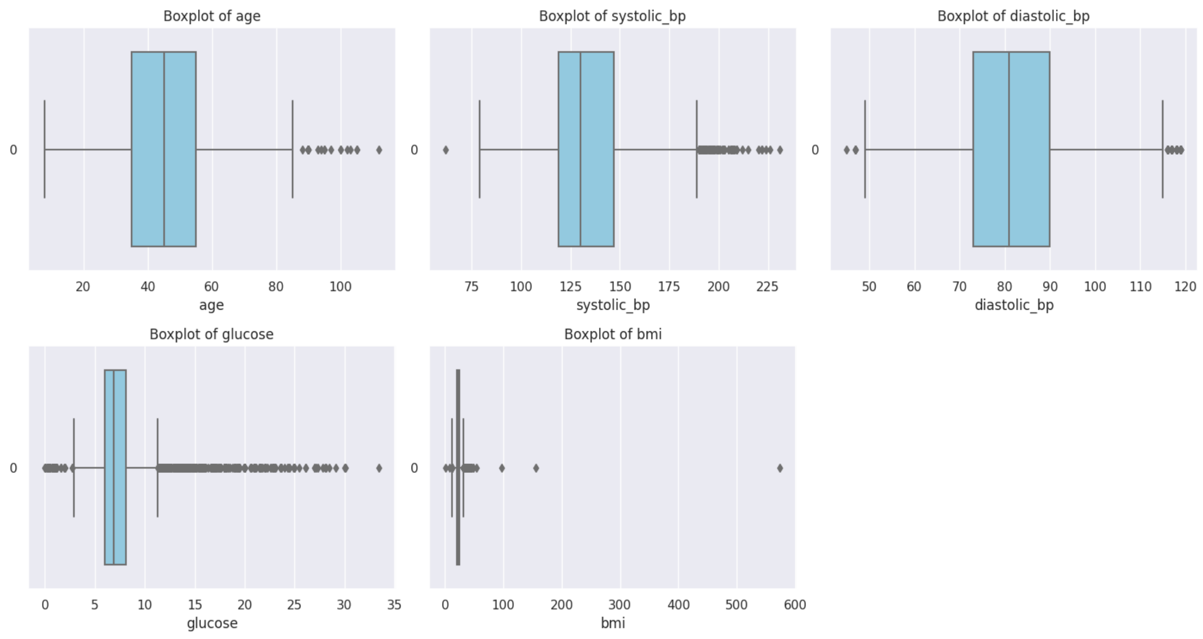


Figure 14: Combined Box Plots

Clearing outliers using quartiles in a box plots:

1. Calculate Q1 (25th percentile) and Q3 (75th percentile).
2. Compute IQR (Interquartile Range): IQR = Q3 − Q1
3. Set bounds for outliers:
    - Lower Bound: Q1 − 1.5 × IQR
    - Upper Bound: Q3 + 1.5 × IQR
4. Identify and remove/adjust data points outside these bounds.
5. Outliers are points below the lower bound or above the upper bound.

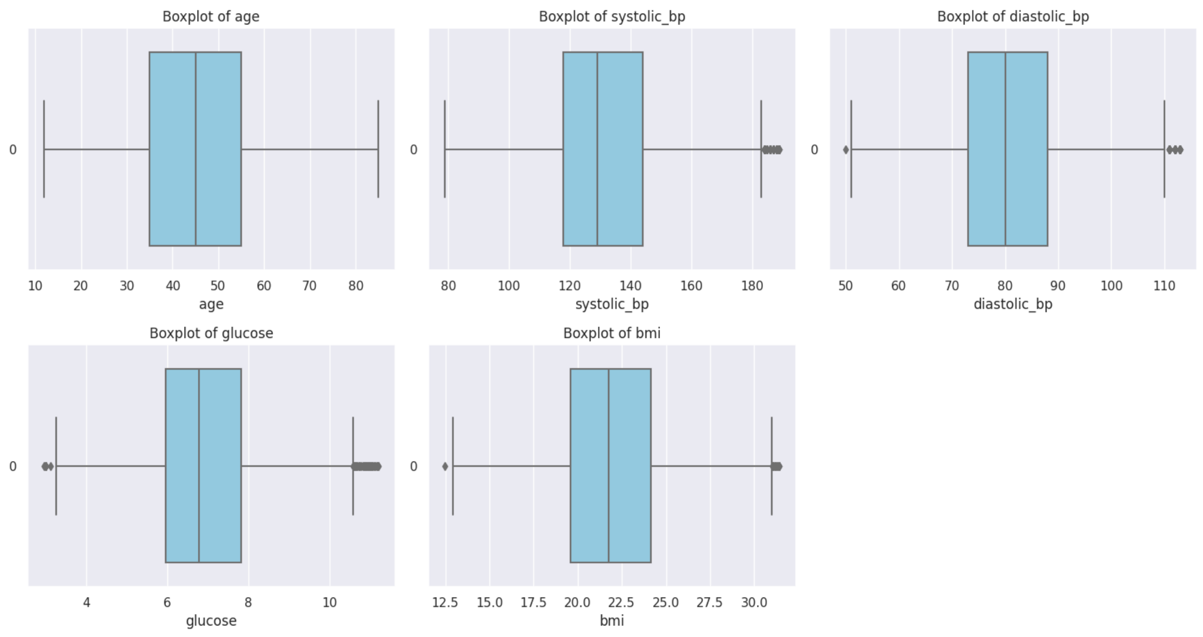


Figure 15: Plots Box plots of Cleaned Data

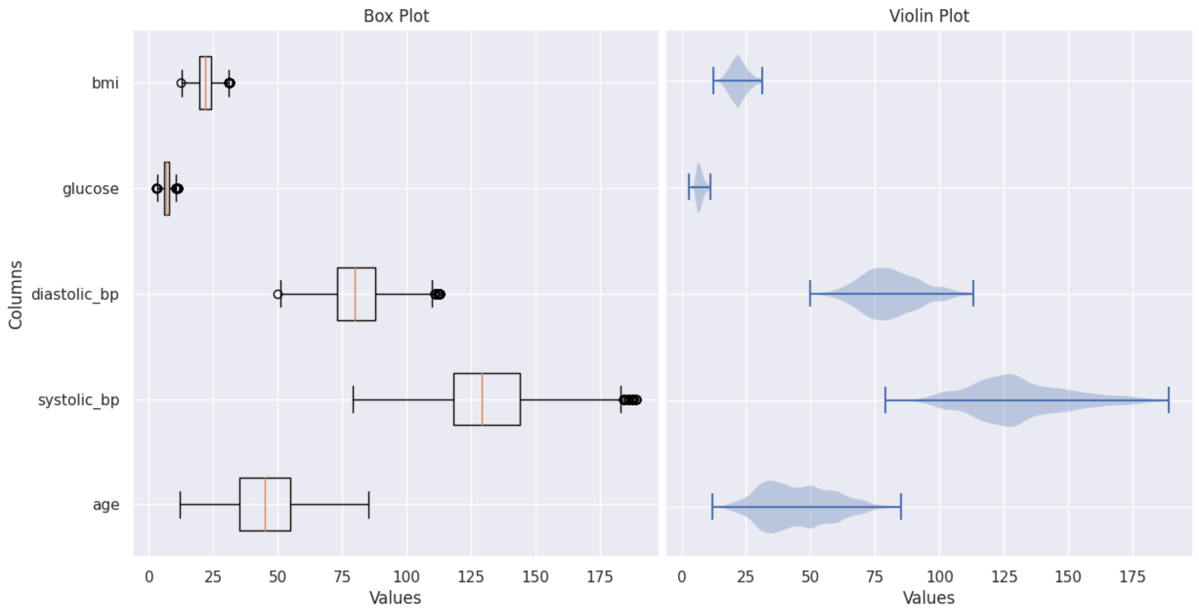


Figure 16: Box plots and Violin plots

After cleaning the outliers, some columns resample normal distribution.

Correlation Matrix

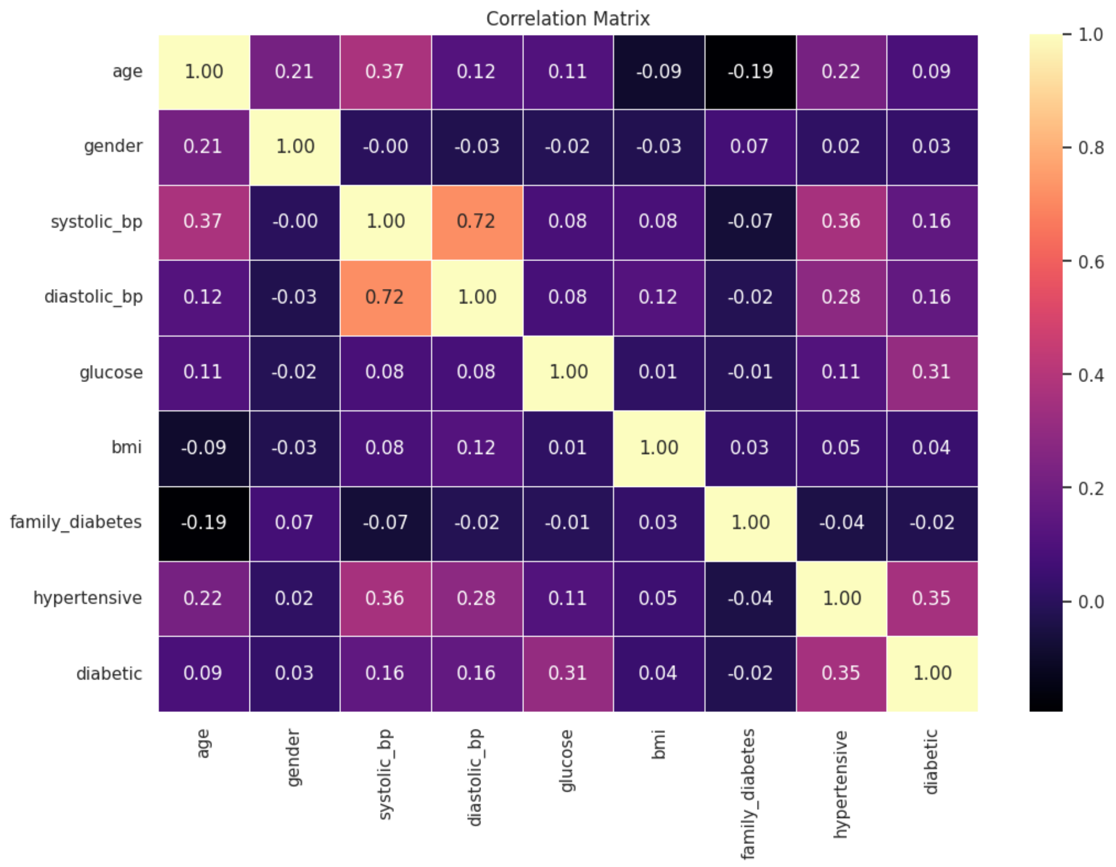


Figure 17: Correlation Matrix between Features and Target

Pair plot

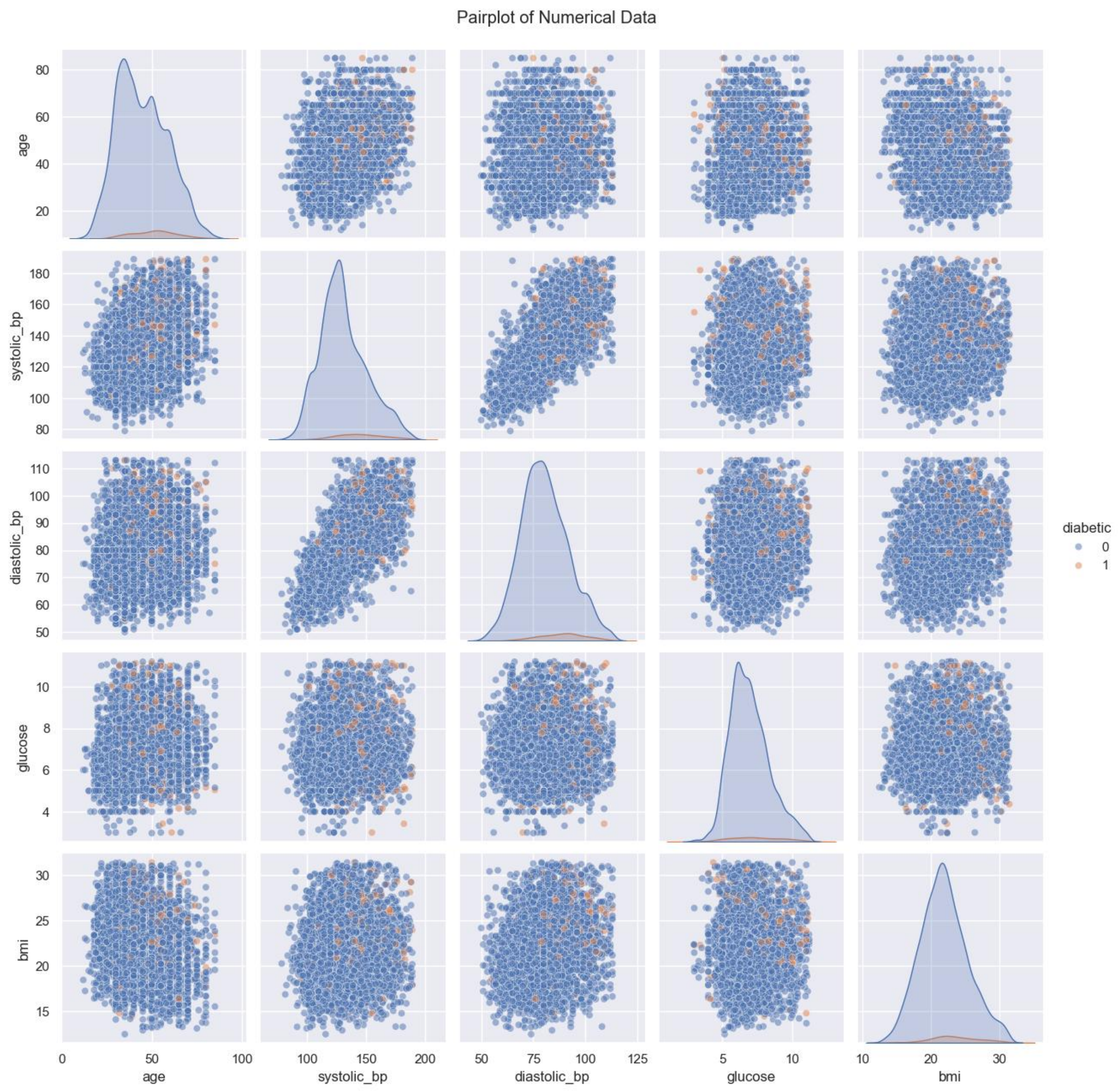


Figure 18: Pair plots

4. MODEL BUILDING

For a classification model, the goal is to predict discrete labels or categories based on input features. The process begins with defining the problem, such as identifying spam emails or diagnosing diseases, and collecting a labelled dataset. The data is then pre-processed to handle missing values, normalize features, and encode categorical variables. Feature engineering is performed to extract or create relevant features that improve the model's ability to distinguish between classes. An appropriate algorithm, such as logistic regression, decision trees, random forests, or support vector machines (SVM) is selected and trained on the training dataset. Hyperparameters are tuned using techniques like cross-validation to optimize performance. The model is evaluated on a validation or test set using metrics like accuracy, precision, recall, and F1-score to ensure it generalizes well to unseen data. Once validated, the model is deployed for real-world use, and its performance is monitored over time to detect and address issues like data drift or degradation in accuracy. This structured approach ensures the creation of a robust and reliable classification model tailored to the specific task at hand.

4.1. Machine Learning Models

As per our project, it is about training models to classify if someone has Type 2 Diabetes or not, so we choose classification models. Classification models are a category of machine learning algorithms designed to predict discrete labels or categories for input data. These models are widely used in tasks such as spam detection, image recognition, medical diagnosis, and sentiment analysis. Below is a comprehensive list of common classification models, grouped by their underlying principles:

+ **Linear Models:**
    - **Logistic Regression:** A linear model for binary or multiclass classification that uses a logistic function to estimate probabilities.
    - **Linear Discriminant Analysis (LDA):** A method that finds a linear combination of features to separate classes.
    - **Support Vector Machines (SVM):** A model that finds the optimal hyperplane to separate classes, often using kernel functions for non-linear data.
+ **Tree-Based Models:**
    - **Decision Trees:** A model that splits data into branches based on feature values to make predictions.
    - **Random Forest:** An ensemble of decision trees that combines their outputs to improve accuracy and reduce overfitting.
    - **Gradient Boosting Machines (GBM):** A sequential ensemble technique that builds trees to correct errors from previous trees.
    - **XGBoost:** An optimized implementation of gradient boosting with additional regularization and efficiency improvements.
    - **LightGBM:** A gradient-boosting framework designed for speed and scalability, particularly with large datasets.

- **CatBoost:** A gradient-boosting algorithm optimized for categorical data and handling missing values

+ **<u>Probabilistic Models:</u>**
    - **Naive Bayes:** A probabilistic model based on Bayes' theorem, assuming independence between features.
    - **Gaussian Naive Bayes:** A variant of Naive Bayes that assumes features follow a Gaussian distribution.

4.2. Training and Evaluation

We have so far come across class imbalance which is a general classification problem. *Class imbalance* refers to a situation in machine learning where the number of observations in one class (the majority class) is significantly higher than the number of observations in another class (the minority class). This disparity can lead to challenges in training predictive models, as the algorithms may become biased towards the majority class, resulting in poor performance for the minority class, which is often of greater interest.

In practical terms, when a dataset is imbalanced, a model trained on it might achieve high overall accuracy simply by predicting the majority class most of the time, while failing to accurately predict instances from the minority class.

4.2.1. Class Resampling

Resampling refers to the process of changing the sampling rate of a discrete signal, either by increasing (up-sampling or oversampling) or decreasing (down-sampling or undersampling) the number of samples. It is commonly used in digital signal processing, audio processing, and data analysis to adjust the resolution or size of a dataset.

In our case, we will employ oversampling for our minority class, which was many times outnumbered by the majority class.

4.2.2. Data Scaling

Data scaling is a crucial preprocessing step in machine learning that helps ensure that features contribute equally to model training. Different scaling methods can be employed depending on the characteristics of the data and the specific requirements of the algorithms being used.

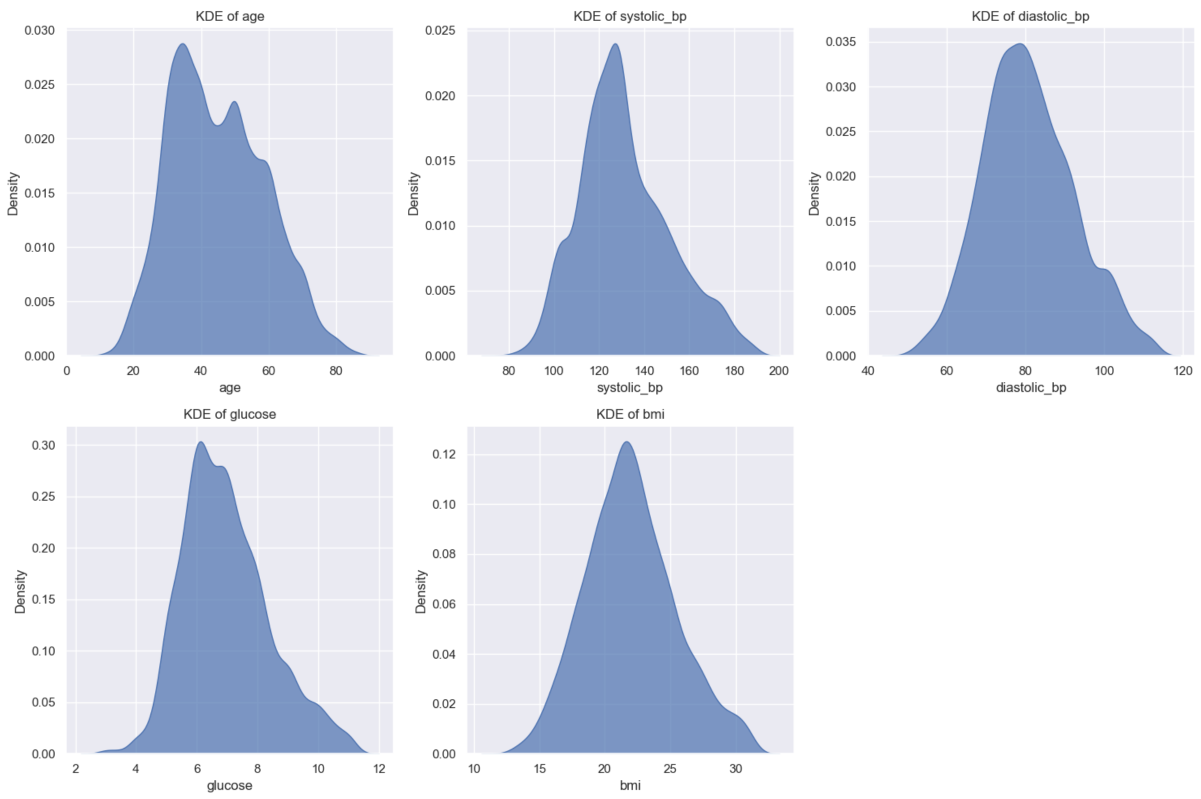
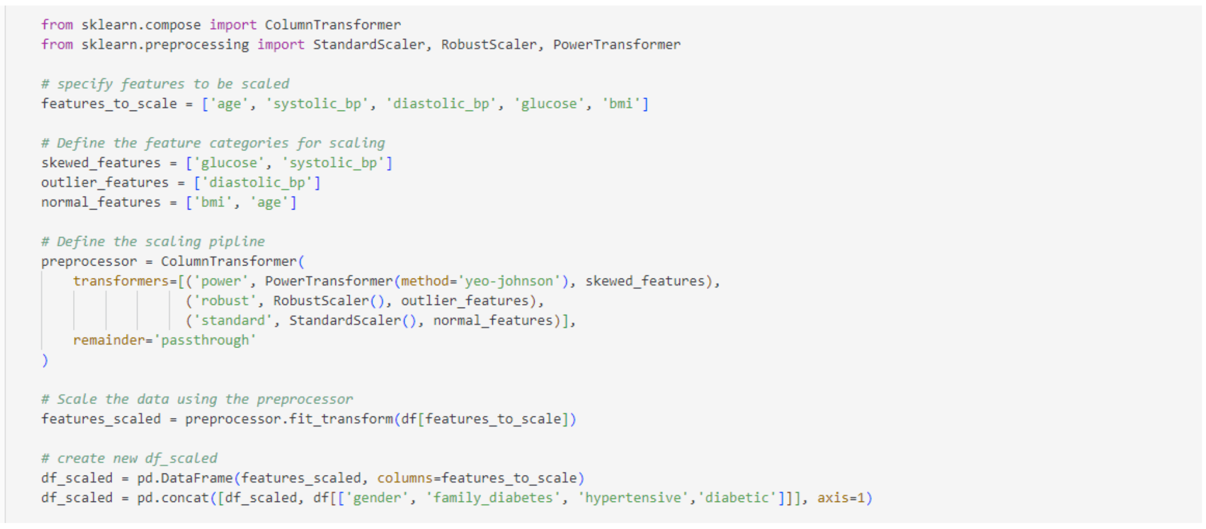


Figure 19: Distribution of Cleaned Data

In favour of our data trends, we are going to scale the numerical data with a combination of scalers.

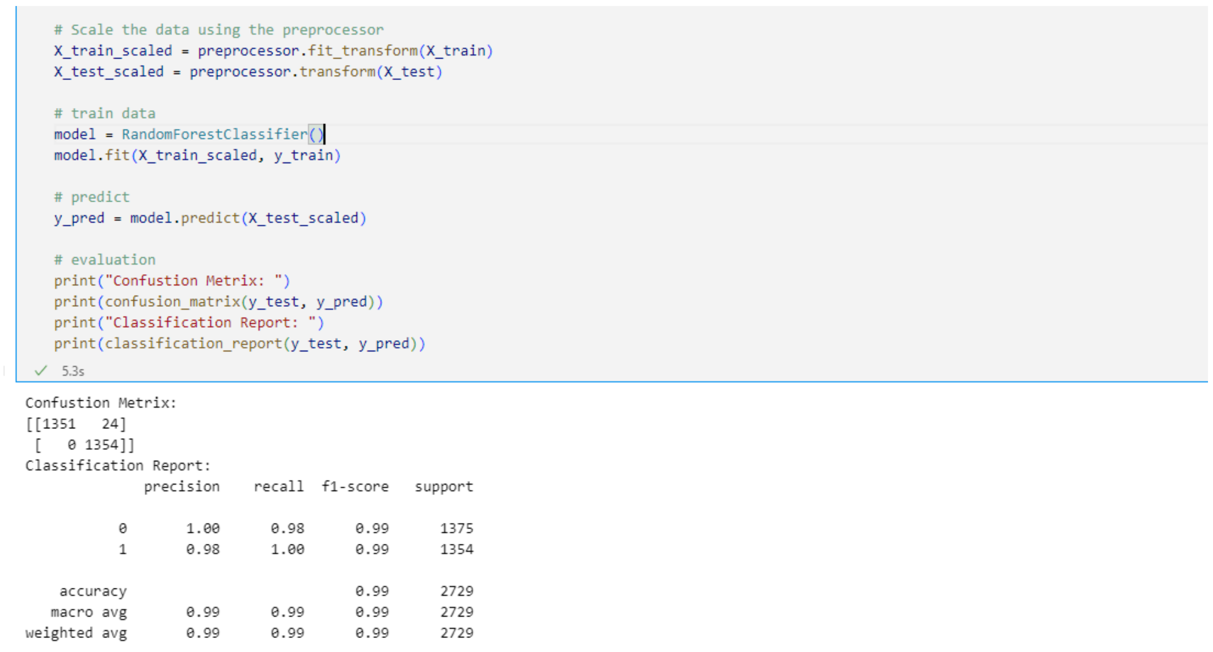


```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, RobustScaler, PowerTransformer

# specify features to be scaled
features_to_scale = ['age', 'systolic_bp', 'diastolic_bp', 'glucose', 'bmi']

# Define the feature categories for scaling
skewed_features = ['glucose', 'systolic_bp']
outlier_features = ['diastolic_bp']
normal_features = ['bmi', 'age']

# Define the scaling pipline
preprocessor = ColumnTransformer(
    transformers=[('power', PowerTransformer(method='yeo-johnson'), skewed_features),
                  ('robust', RobustScaler(), outlier_features),
                  ('standard', StandardScaler(), normal_features)],
    remainder='passthrough'
)

# Scale the data using the preprocessor
features_scaled = preprocessor.fit_transform(df[features_to_scale])

# create new df_scaled
df_scaled = pd.DataFrame(features_scaled, columns=features_to_scale)
df_scaled = pd.concat([df_scaled, df[['gender', 'family_diabetes', 'hypertensive','diabetic']]], axis=1)
```

Figure 20:Combined Scalers

4.2.3. Linear Models

Logistic Regression



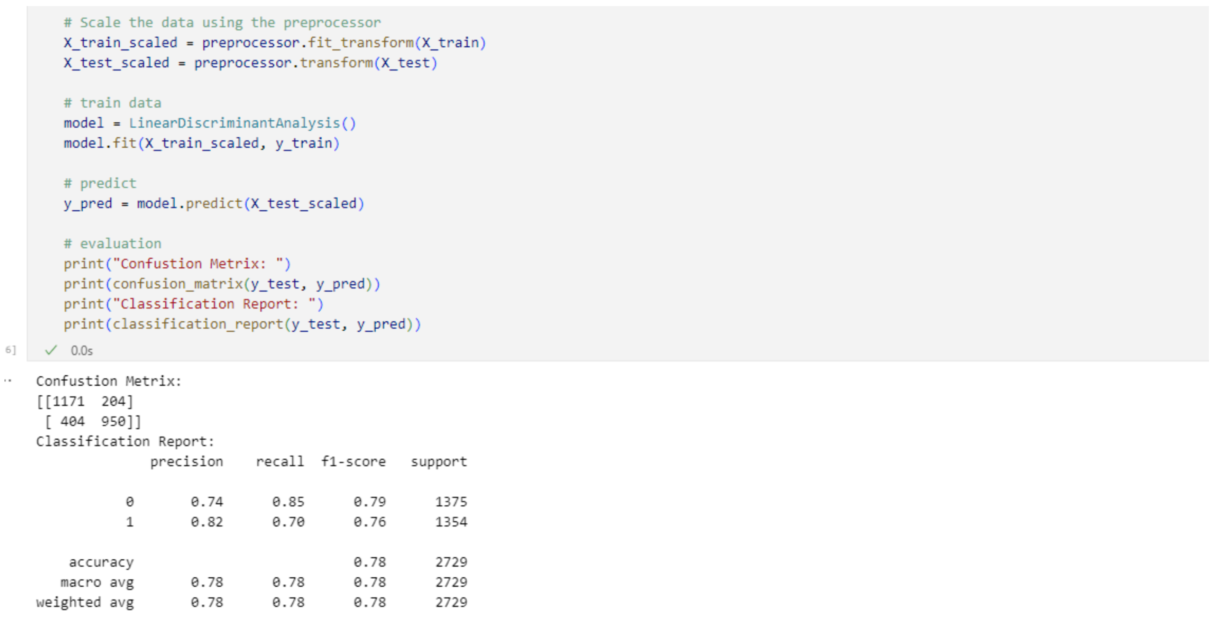
```
# Scale the data using the preprocessor
X_train_scaled = preprocessor.fit_transform(X_train)
X_test_scaled = preprocessor.transform(X_test)

# train data
model = RandomForestClassifier()
model.fit(X_train_scaled, y_train)

# predict
y_pred = model.predict(X_test_scaled)

# evaluation
print("Confustion Metrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

✓ 5.3s

```
Confustion Metrix:
[[1351   24]
 [   0 1354]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1375
           1       0.98      1.00      0.99      1354

    accuracy                           0.99      2729
   macro avg       0.99      0.99      0.99      2729
weighted avg       0.99      0.99      0.99      2729
```

Figure 21: Logistic Regression Model

LDA



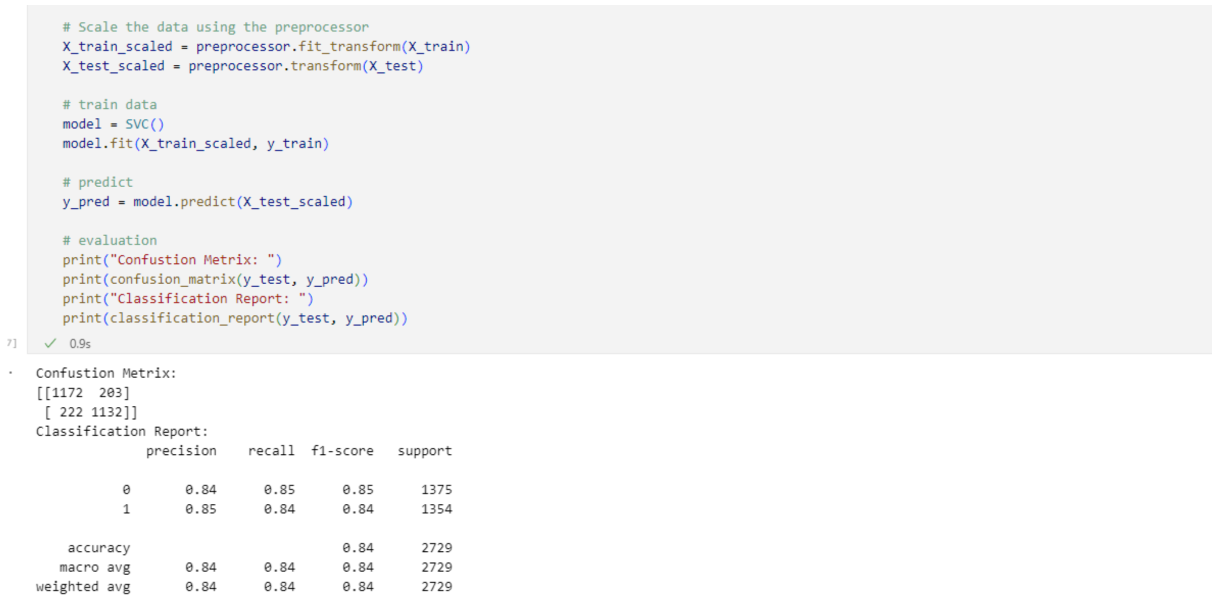
```
# Scale the data using the preprocessor
X_train_scaled = preprocessor.fit_transform(X_train)
X_test_scaled = preprocessor.transform(X_test)

# train data
model = LinearDiscriminantAnalysis()
model.fit(X_train_scaled, y_train)

# predict
y_pred = model.predict(X_test_scaled)

# evaluation
print("Confustion Metrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
Confustion Metrix:
[[1171  204]
 [ 404  950]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.85      0.79      1375
           1       0.82      0.70      0.76      1354

    accuracy                           0.78      2729
   macro avg       0.78      0.78      0.78      2729
weighted avg       0.78      0.78      0.78      2729
```

Figure 22: Linear Discriminant Analysis Model

SVM



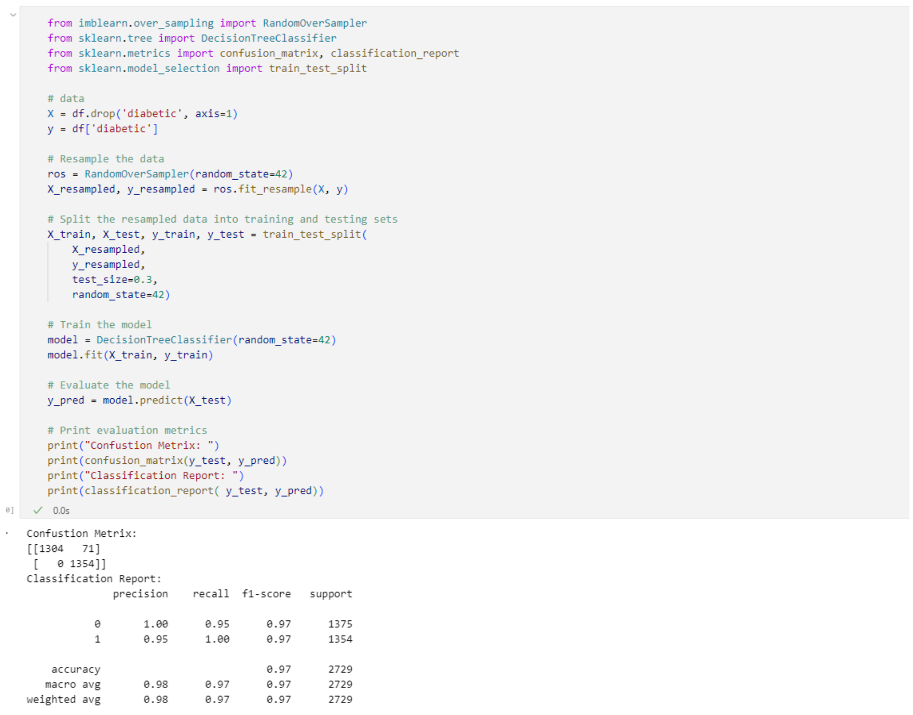
```
# Scale the data using the preprocessor
X_train_scaled = preprocessor.fit_transform(X_train)
X_test_scaled = preprocessor.transform(X_test)

# train data
model = SVC()
model.fit(X_train_scaled, y_train)

# predict
y_pred = model.predict(X_test_scaled)

# evaluation
print("Confustion Metrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

```
Confustion Metrix:
[[1172  203]
 [ 222 1132]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      1375
           1       0.85      0.84      0.84      1354

    accuracy                           0.84      2729
   macro avg       0.84      0.84      0.84      2729
weighted avg       0.84      0.84      0.84      2729
```

Figure 23: Support Vector Machine Model

4.2.4. Tree-Based Models

Decision Trees

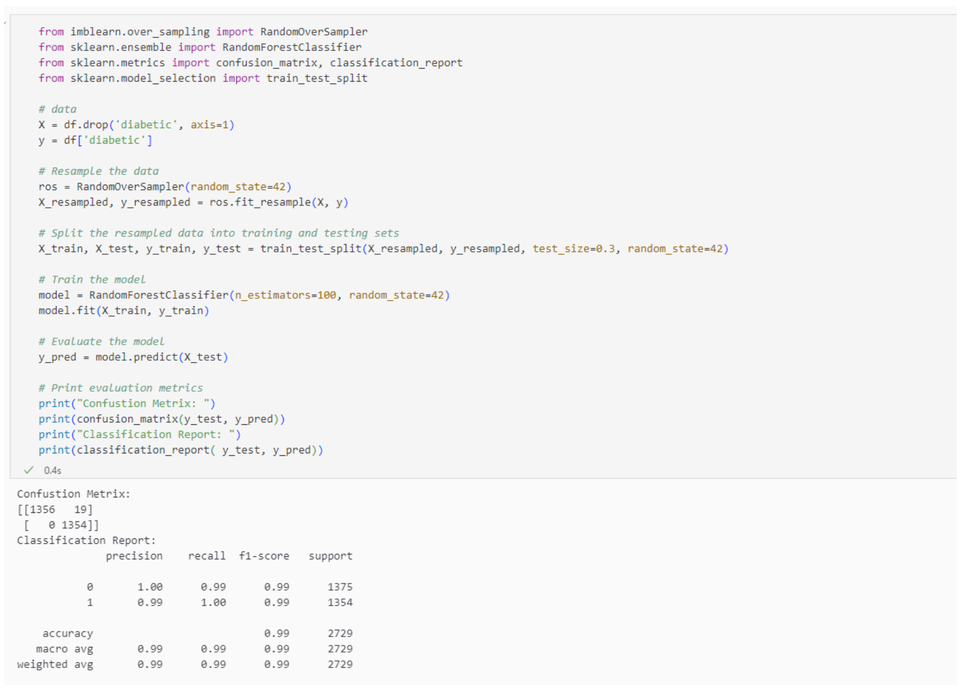


```
from imblearn.over_sampling import RandomOverSampler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Resample the data
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_resampled,
    y_resampled,
    test_size=0.3,
    random_state=42)

# Train the model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confustion Metrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report( y_test, y_pred))
```

```
Confustion Metrix:
[[1304   71]
 [   0 1354]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.97      1375
           1       0.95      1.00      0.97      1354

    accuracy                           0.97      2729
   macro avg       0.98      0.97      0.97      2729
weighted avg       0.98      0.97      0.97      2729
```

Figure 24: Decision Tree Model

Random Forest

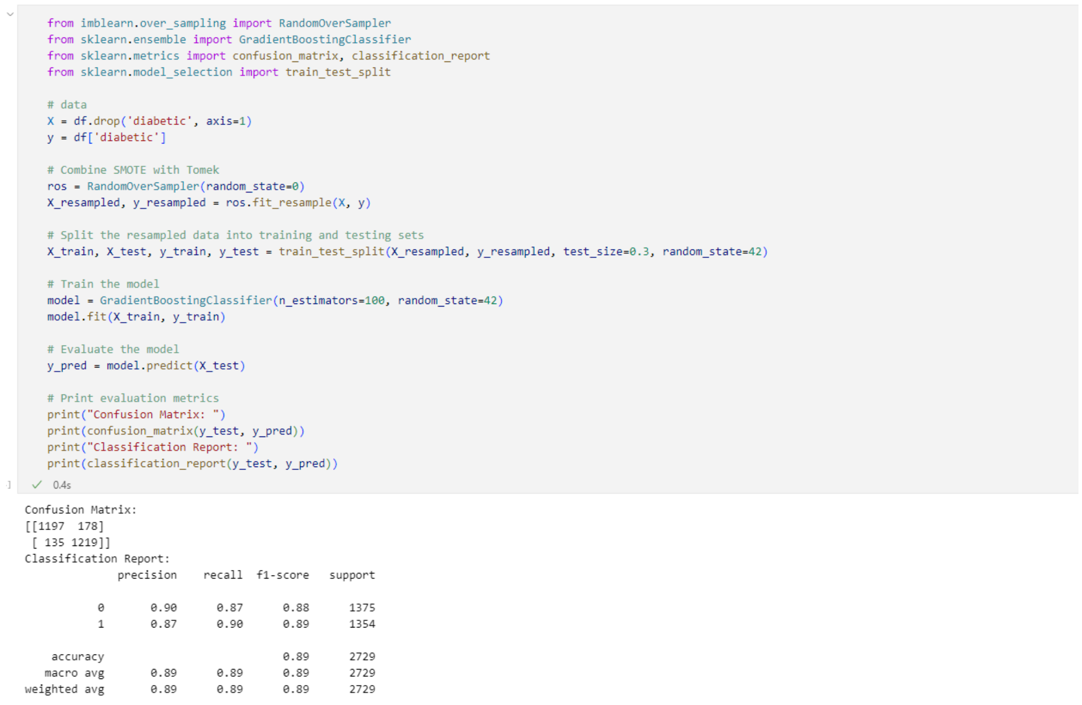


```
from imblearn.over_sampling import RandomOverSampler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Resample the data
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confustion Metrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report( y_test, y_pred))
```

```
Confustion Metrix:
[[1356   19]
 [   0 1354]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1375
           1       0.99      1.00      0.99      1354

    accuracy                           0.99      2729
   macro avg       0.99      0.99      0.99      2729
weighted avg       0.99      0.99      0.99      2729
```

Figure 25: Random Forest Model

Gradient Boosting Machines (GBM)

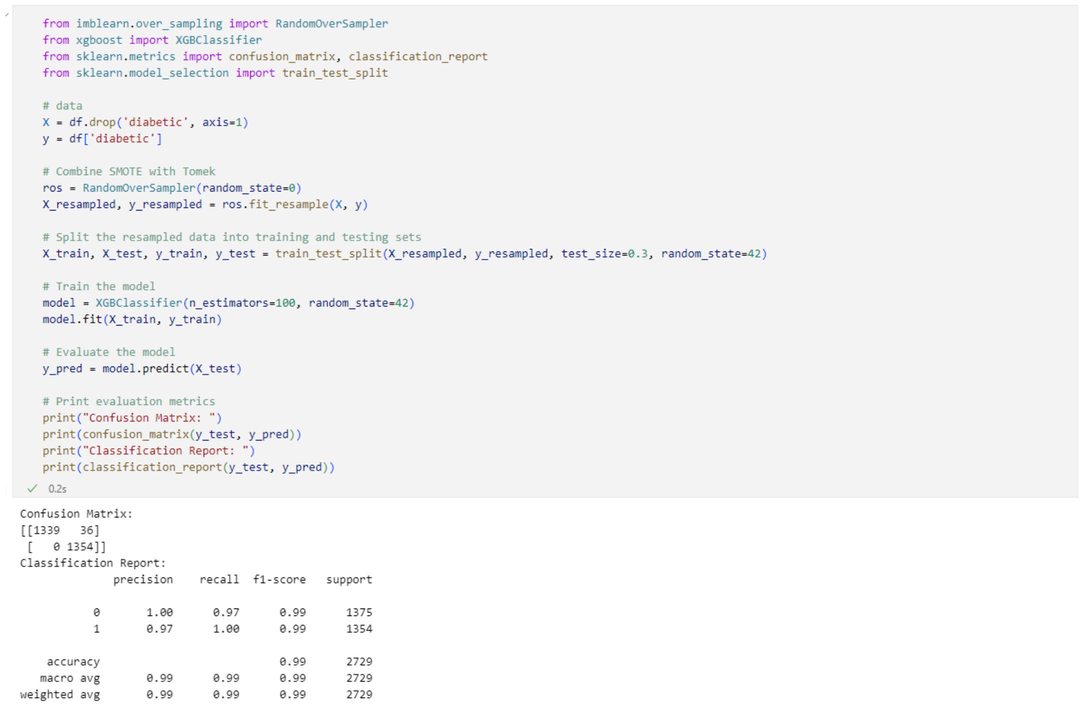


```python
from imblearn.over_sampling import RandomOverSampler
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Combine SMOTE with Tomek
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
model = GradientBoostingClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[1197  178]
 [ 135 1219]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.87      0.88      1375
           1       0.87      0.90      0.89      1354

    accuracy                           0.89      2729
   macro avg       0.89      0.89      0.89      2729
weighted avg       0.89      0.89      0.89      2729
```

Figure 26: Gradient Boosting Machines Model

XGBoost

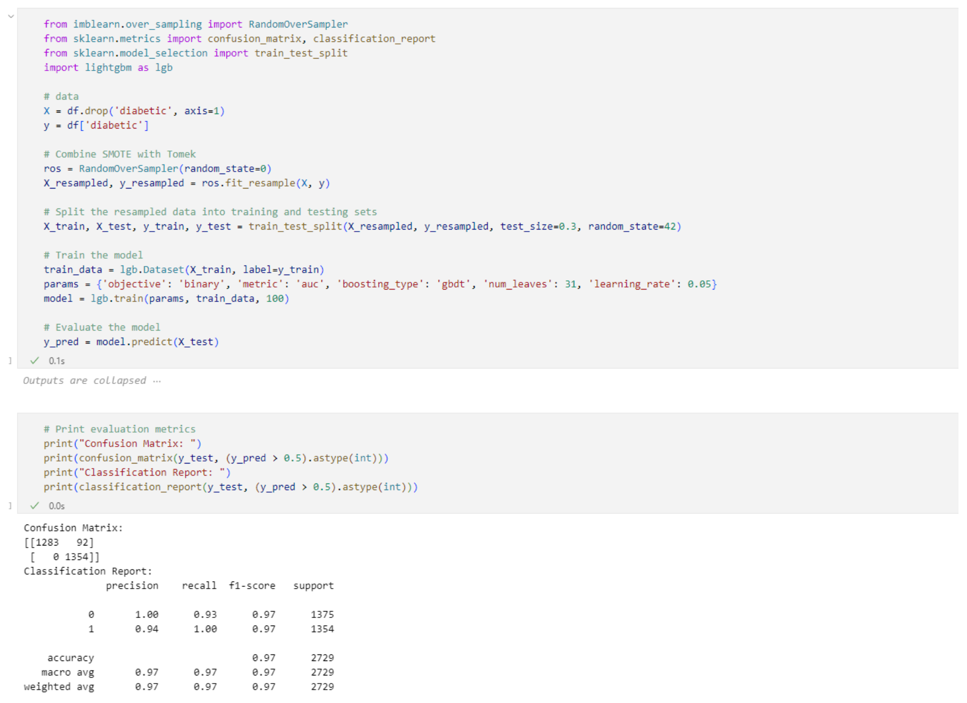


```python
from imblearn.over_sampling import RandomOverSampler
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Combine SMOTE with Tomek
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
model = XGBClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[1339   36]
 [   0 1354]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99      1375
           1       0.97      1.00      0.99      1354

    accuracy                           0.99      2729
   macro avg       0.99      0.99      0.99      2729
weighted avg       0.99      0.99      0.99      2729
```

Figure 27: XGBooost Model

LightGBM



```
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import lightgbm as lgb

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Combine SMOTE with Tomek
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
train_data = lgb.Dataset(X_train, label=y_train)
params = {'objective': 'binary', 'metric': 'auc', 'boosting_type': 'gbdt', 'num_leaves': 31, 'learning_rate': 0.05}
model = lgb.train(params, train_data, 100)

# Evaluate the model
y_pred = model.predict(X_test)
```

```
✓ 0.1s
Outputs are collapsed ...
```

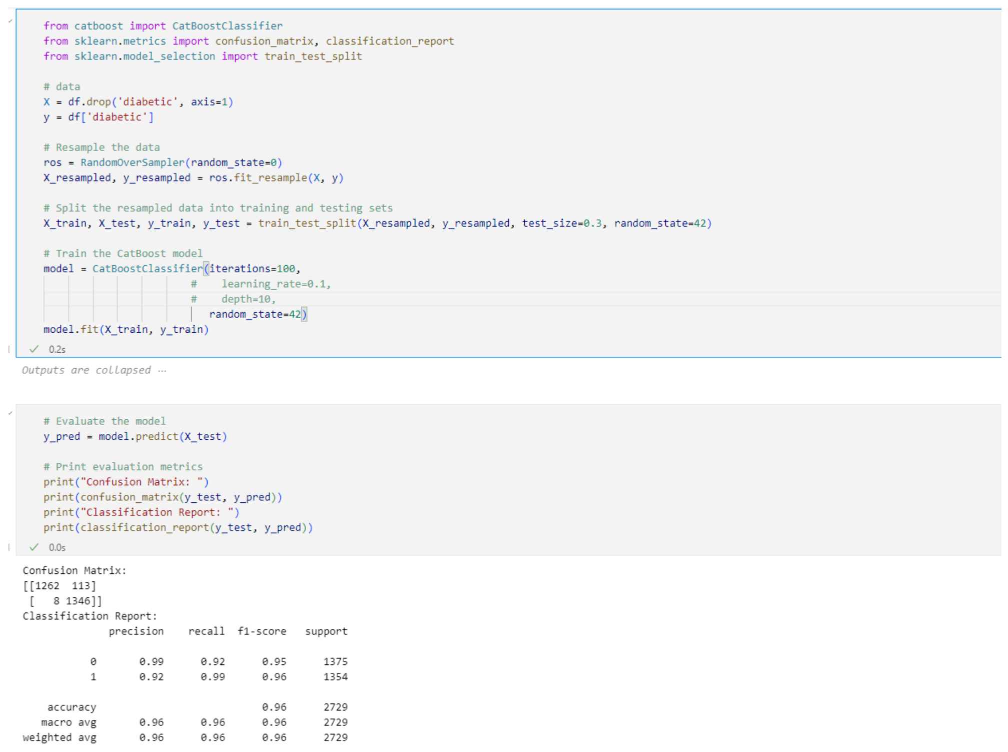
```
# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, (y_pred > 0.5).astype(int)))
print("Classification Report: ")
print(classification_report(y_test, (y_pred > 0.5).astype(int)))
```

```
✓ 0.0s
```

```
Confusion Matrix:
[[1283   92]
 [   0 1354]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.93      0.97      1375
           1       0.94      1.00      0.97      1354

    accuracy                           0.97      2729
   macro avg       0.97      0.97      0.97      2729
weighted avg       0.97      0.97      0.97      2729
```

Figure 28: LightGBM Model

CatBoost



```
from catboost import CatBoostClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Resample the data
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the CatBoost model
model = CatBoostClassifier(iterations=100,
                        #    learning_rate=0.1,
                        #    depth=10,
                           random_state=42)
model.fit(X_train, y_train)
```

```
✓ 0.2s
Outputs are collapsed ...
```

```
# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

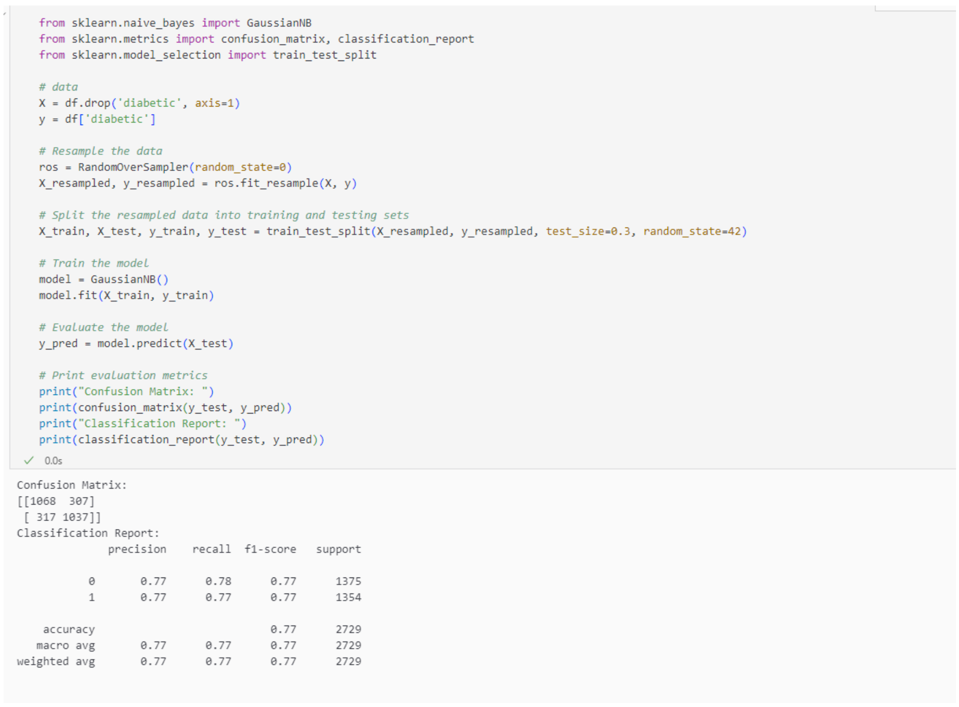
```
✓ 0.0s
```

```
Confusion Matrix:
[[1262  113]
 [   8 1346]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.92      0.95      1375
           1       0.92      0.99      0.96      1354

    accuracy                           0.96      2729
   macro avg       0.96      0.96      0.96      2729
weighted avg       0.96      0.96      0.96      2729
```

Figure 29: CatBoost Model

4.2.5. Probabilistic Models

Gaussian Naive Bayes



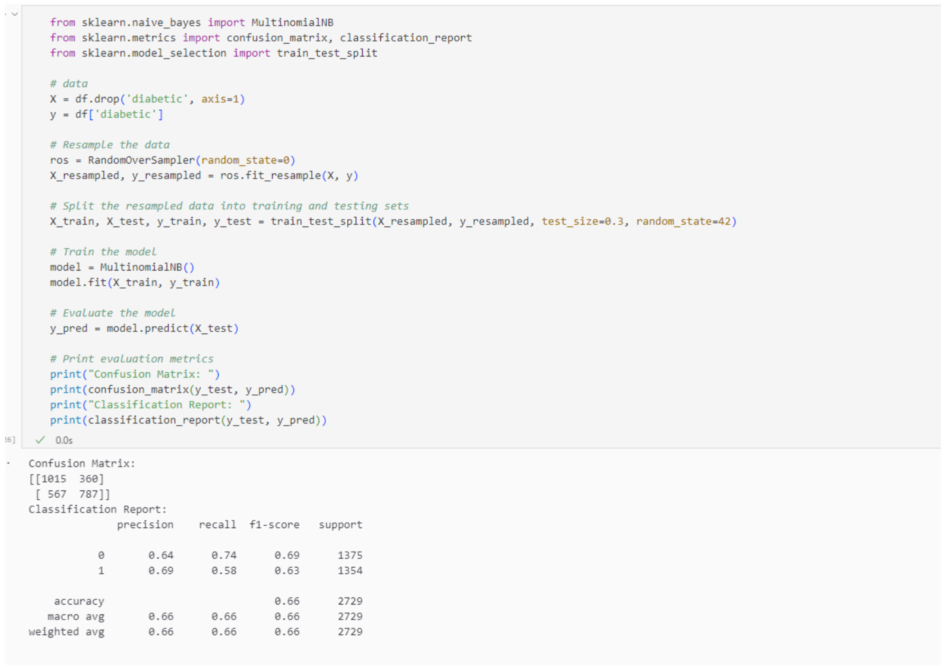
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Resample the data
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
model = GaussianNB()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
Confusion Matrix:
[[1068  307]
 [ 317 1037]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.78      0.77      1375
           1       0.77      0.77      0.77      1354

    accuracy                           0.77      2729
   macro avg       0.77      0.77      0.77      2729
weighted avg       0.77      0.77      0.77      2729
```

Figure 30: Gaussian Naive Bayes Model

Multinomial Naive Bayes



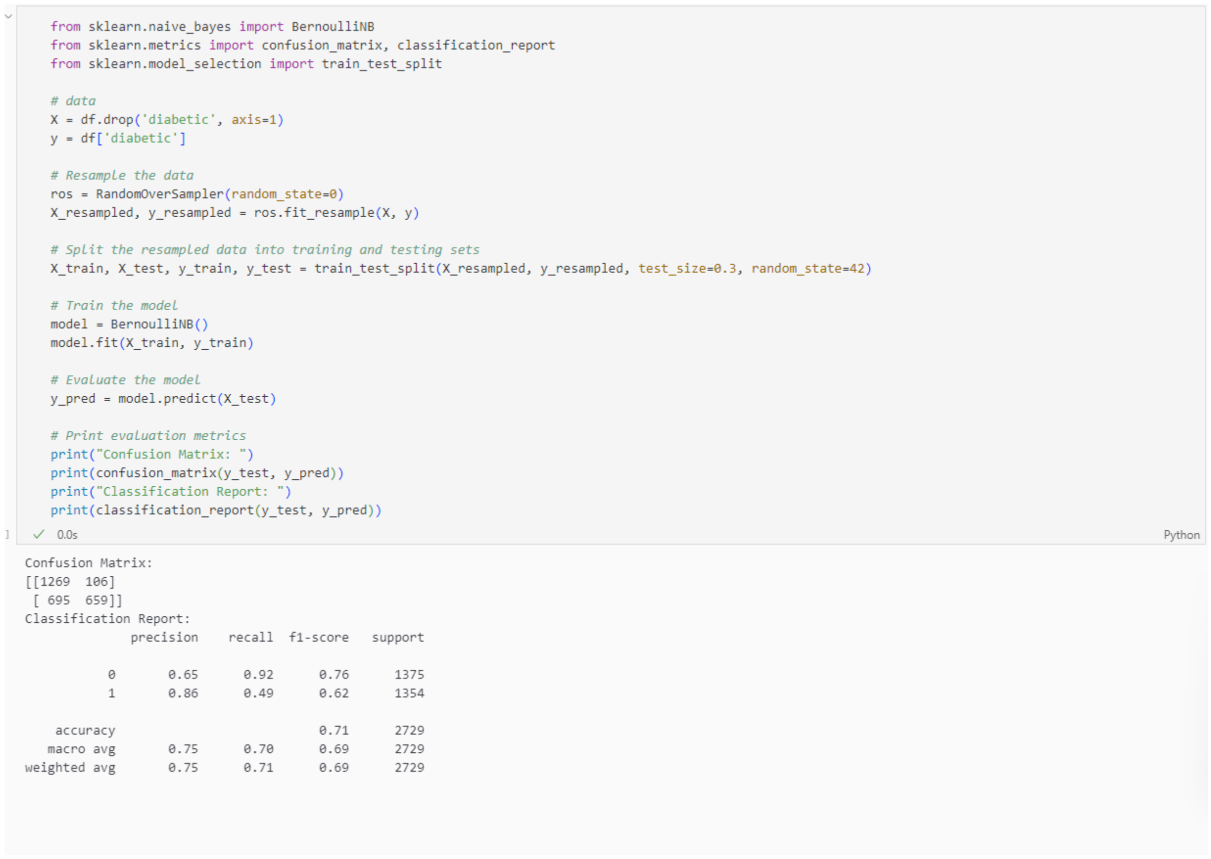
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Resample the data
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
model = MultinomialNB()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
Confusion Matrix:
[[1015  360]
 [ 567  787]]
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.74      0.69      1375
           1       0.69      0.58      0.63      1354

    accuracy                           0.66      2729
   macro avg       0.66      0.66      0.66      2729
weighted avg       0.66      0.66      0.66      2729
```

Figure 31: Multinomial Naive Bayes Model

Bernoulli Naive Bayes



```python
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# data
X = df.drop('diabetic', axis=1)
y = df['diabetic']

# Resample the data
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Train the model
model = BernoulliNB()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
print("Classification Report: ")
print(classification_report(y_test, y_pred))
```

✓ 0.0s Python

```
Confusion Matrix:
[[1269  106]
 [ 695  659]]
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.92      0.76      1375
           1       0.86      0.49      0.62      1354

    accuracy                           0.71      2729
   macro avg       0.75      0.70      0.69      2729
weighted avg       0.75      0.71      0.69      2729
```

Figure 32: Bernoulli Naive Bayes Model

4.3. Model Selection

For this medical diagnosis task, model selection was guided by the need to minimize false negatives (missed diagnoses). To select the best-performing model, we evaluated models using a Confusion Matrix and Classification Report. The *Random Forest* model achieved the highest accuracy of 99% and an F1-score of 0.99, outperforming the other models. The Confusion Matrix revealed that Random Forest had the lowest number of false positives and false negatives, indicating robust performance across all classes.

5. DISCUSSION

5.1. Interpretation of Results

The model's high recall (100% for positive cases) ensures that most patients with the condition are correctly identified, reducing the risk of missed diagnoses. While the precision of 99% for positive cases means that a few healthy individuals may be flagged for further testing, this trade-off is acceptable in a medical context where the cost of missing a diagnosis is significantly higher than the cost of additional testing.

5.2. Challenges

One key challenge is the quality of data, as missing or incorrect information can impact the accuracy of predictions. Additionally, protecting sensitive health data with encryption and access controls is essential. Another issue is bias in the data, which can lead to unfair predictions, especially for underrepresented groups. To fix this, careful selection of data is needed. Also, Type 2 Diabetes is influenced by many factors, making it hard to predict accurately. On top of that, machine learning models may not work well for different populations, and integrating these systems into current healthcare practices can be difficult.

5.3. Future Work

To tackle these challenges, future work should include using more data, like wearables and genetic information, which will help make predictions more accurate. In addition, exploring better machine learning methods and creating easier-to-understand models will make the system more reliable for healthcare professionals. Moreover, real-time monitoring could allow for constant risk tracking and quicker intervention. Furthermore, testing the model on diverse populations will ensure fairness, while connecting it to Electronic Health Records (EHR) will make it easier to use in practice. Finally, long-term studies will refine the predictions, and working closely with healthcare experts will ensure the system remains effective and practical:

6. CONCLUSION

This project shows that machine learning is essential in predicting early diagnosis of type 2 diabetes, which makes it easy to identify people at risk without an internal test for the patient directly. Moreover, this model uses health information such as age, blood pressure, glucose levels, and BMI to give accurate results. While there were challenges, including the quality of the data, our next projects will improve it by adding more data from devices and genetic information. Additionally, if we test the model with different groups of people and connect it to the healthcare systems, it will be more reliable and accurate. In short, this project highlights that machine learning can make a big difference in finding diabetes early and improving the healthcare system effectively.

Reference

Carnethon, M. R., Yan, L., Greenland, P., Garside, D. B., Dyer, A. R., Metzger, B., & Daviglus, M. L. (2008). Resting heart rate in middle age and diabetes development in older age. *Diabetes Care*, *31*(2), 335–339. https://doi.org/10.2337/dc07-0874

Kannel, W. B., McGee, D. L., & National Institutes of Health, National Heart, Lung, and Blood Institute, Heart Disease Epidemiology Study, Framingham, Massachusetts. (1979). Diabetes and cardiovascular risk Factors: The Framingham Study. In *Circulation* (Vol. 59, Issue No. 1).http://ahajournals.org

Seshasai, S. R. K., Kaptoge, S., Thompson, A., Di Angelantonio, E., Gao, P., Sarwar, N., Whincup, P. H., Mukamal, K. J., Gillum, R. F., Holme, I., Njolstad, I., Fletcher, A., Nilsson, P., Lewington, S., Collins, R., Gudnason, V., Thompson, S. G., Sattar, N., Selvin, E., . . . Danesh, J. (2011). Diabetes mellitus, fasting glucose, and risk of Cause-Specific death. *New England Journal of Medicine*, *364*(9), 829–841. https://doi.org/10.1056/nejmoa1008862

Family History and Hypertension with Incidence of Diabetes Mellitus Type II. (2023). In *Jurnal Eduhealth* (Vol. 14, Issue 03, pp. 1125–1126) [Journal-article]. http://ejournal.seaninstitute.or.id/index.php/healt