Module Code & Module Title

CU6051NI Artificial Intelligence

75% Individual Coursework

Submission: Final Submission

Academic Semester: Autumn Semester 2025

Credit: 15 credit semester long module

Student Name: Unison Raj Tuladhar

London Met ID: 23050414

College ID: NP01CP4A230141

Assignment Due Date: 21/01/2026

Assignment Submission Date: 21/01/2026

Submitted To: Er. Roshan Shrestha

GitHub Link	*https://github.com/UnisonTuladhar/AI_Project.git*

Table of Contents

Table of Figures

1. Introduction

1.1 Introduction to Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a technology that enables computers and machines to perform cognitive functions we associate with human minds, such as perceiving, reasoning, learning, interacting and problem solving. We all have interacted with AI technology even if you don't realize it. Voice assistants like Siri, Alexa or some customer chat bots are some examples of AI (McKinsey, 2024). AI encompasses the fields of computer and data science focused on building machines with human intelligence to perform various tasks. Instead of relying instructions from an individual, AI system can learn from the data which lets them handle the hard-complex problems by their own and improve their accuracy overtime (Tech, 2025).

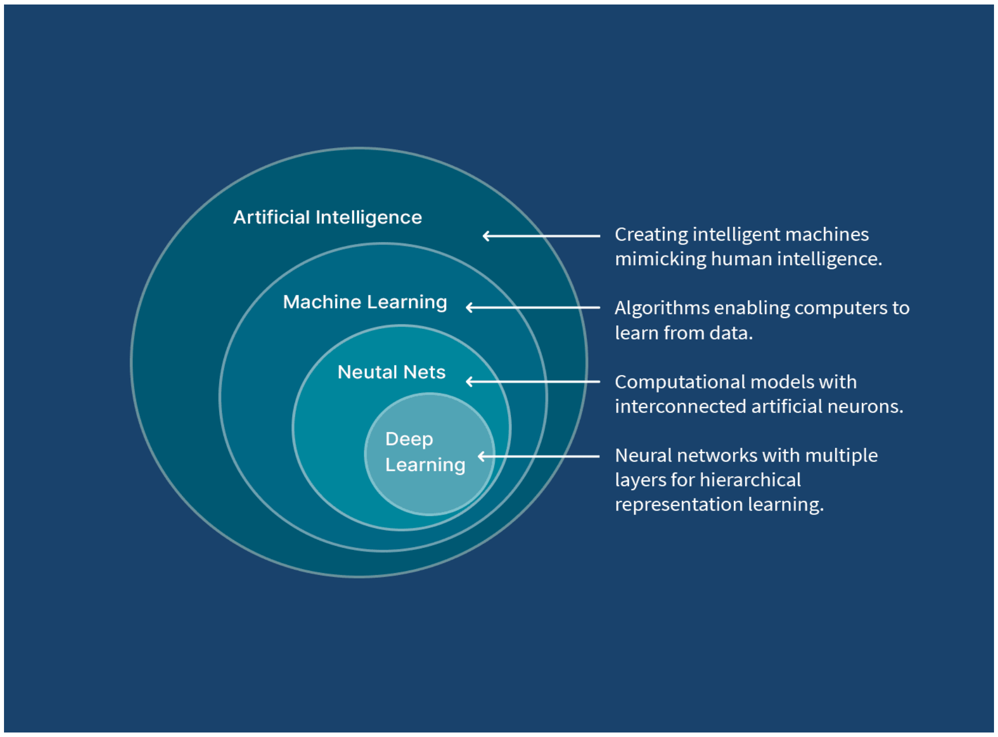

Artificial Intelligence
Creating intelligent machines mimicking human intelligence.
Machine Learning
Algorithms enabling computers to learn from data.
Neutal Nets
Computational models with interconnected artificial neurons.
Deep Learning
Neural networks with multiple layers for hierarchical representation learning.

Figure 1: Artificial Intelligence and Machine Learning.

Machine Learning (ML) is a form of AI that enables machines to learn data patterns automatically and improve their performance over time without being explicitly programmed. It does this by optimizing model parameters through calculations. The learning algorithm then continuously updates the parameter values allowing the model to make predictions or decisions (ISO, 2022). Machine Learning is categorized into three types:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Supervised Learning (classification) has been chosen for this project. This model uses machine learning classification to analyse the healthcare data from the diabetes dataset. The diabetes dataset contains data such as age, gender, BMI, Cholesterol, Triglycerides, HDL, LDL, Creatinine and Blood Urea Nitrogen. These data will be pre-processed and converted into suitable numerical format that can be learned from the ML algorithms. For this model three best classification algorithms were selected (Logistic Regression, Random Forest Classifier and SVM).

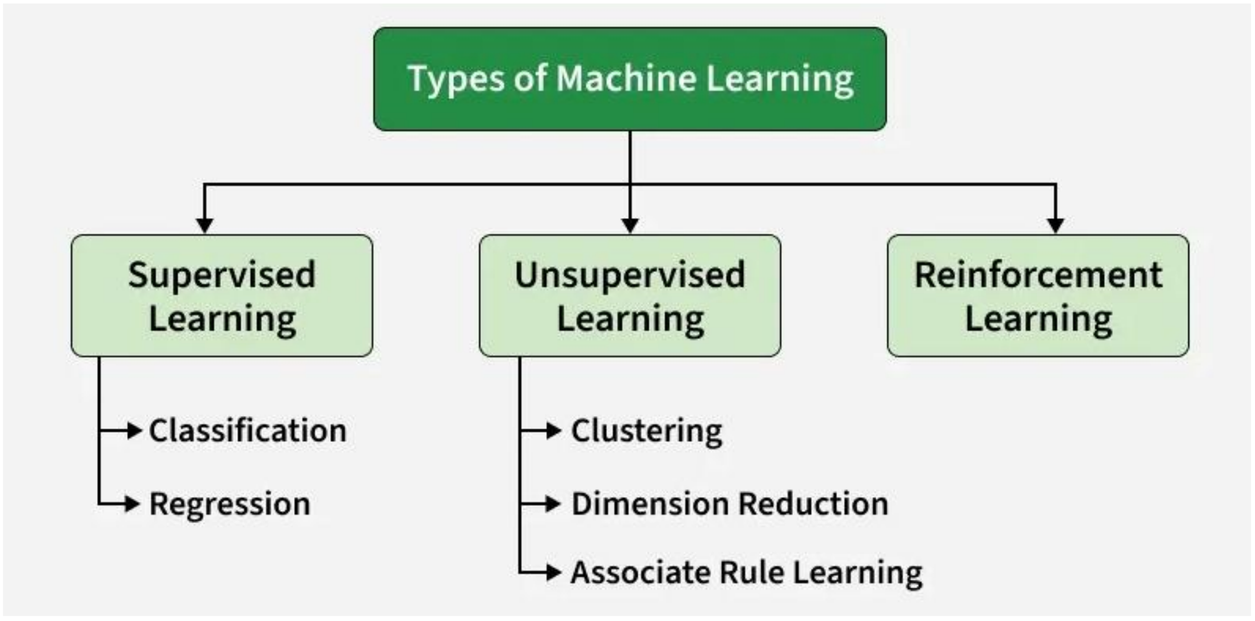


Figure 2: Types of Machine Learning (ML).

1.2 Problem Domain

The chosen problem domain for this project lies under healthcare sector. The topic of my project is Diabetes Prediction using supervised learning classification algorithm. Diabetes is a chronic disease that occurs when the pancreas does not produce enough insulin or the body cannot effectively use the insulin it produces. Diabetes is a disease that has affected millions of people worldwide and if not treated well it leads to serious damage to the body overtime. Around 14% of the adults aged 18 and older were having diabetes and more than 59% of adults 30 or older are living with diabetes according to the research done in 2022. In 2021 diabetes was the cause of death of around 1.6 million and 47% of them occurred before the age of 70 (WHO, 2024).

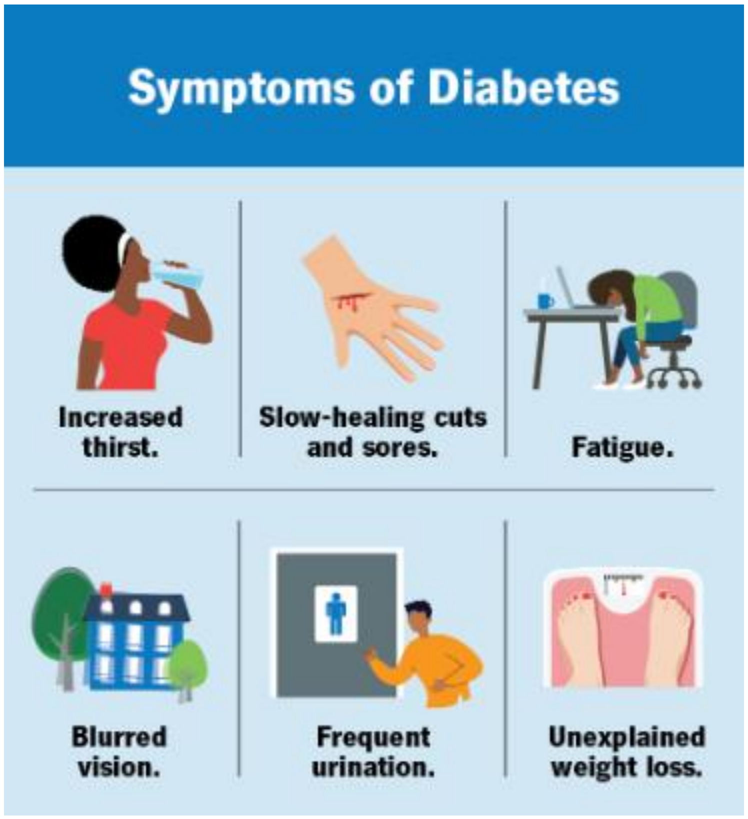


Figure 3: Symptoms of Diabetes.

2. Background

2.1 Research on Diabetes Prediction

Diabetes is a common disease that affected around 537 million adults around the world and this number might go up to 643 million by 2030 and 783 million in 2040 people of all ages. There are many types of diabetes but Type 2 diabetes are the most common ones, Type 1 diabetes is mostly diagnosed to children and teenagers and Gestational diabetes are the once developed during pregnancy (Clinic, 2023). Many individuals tend to delay their medical tests due to lack of time, laziness, lack of awareness even the symptoms are visible. As a result, diabetes reaches at later stages and the later stages conditions becomes more severe.

Since many individuals don't even realize they have diabetes until later stages which could be harmful so early prediction of diabetes using medical data can help individuals to know about the disease earlier. The main goal of this project is to build a classification model using algorithms like logistic regression, random forest classifier, support vector machine (SVM) to predict whether an individual is likely to have diabetes based on the persons attributes such as age, gender, body mass index, cholesterol, creatine and so on.

2.2 Existing Research Work

Several studies have been done and explored in the medical field related to diabetes as it is one of the most common disease. Here below are the three previous researches done in the medical field related to diabetes.

- **Research 1 (Diabetes Prediction Using Machine Learning Classification)**

This research is based on the supervised learning using classification algorithms to predict diabetes using medical data. Algorithms like Logistic regression, KNN, SVM, Decision Tree algorithms are using in this research. As the research the prevalence of diabetes for all the age group was estimated to be 171 million (2.8%) in 2000 A.D and increases up to 366 million (4.4%) in 2030 A.D. So, the fact that these many individuals are affected from the disease but still many of them don’t even realize diabetes in them until later stages which could be harmful. The final result displayed the SVM and the Decision Tree algorithm performed better than the other algorithms with an accuracy rate of 80% which is considered to be a great amount. The study demonstrates that ML based classification can be a great help for the healthcare domain in early diabetes prediction (B.V., 2017).

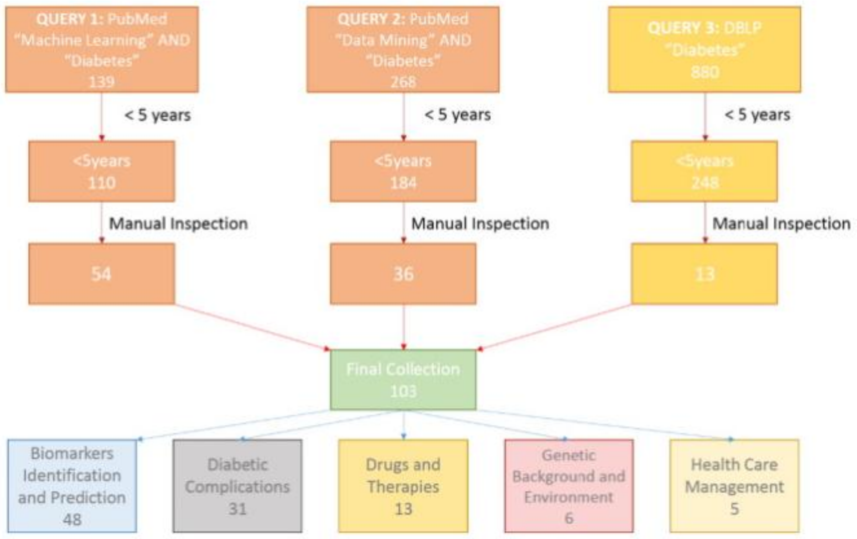
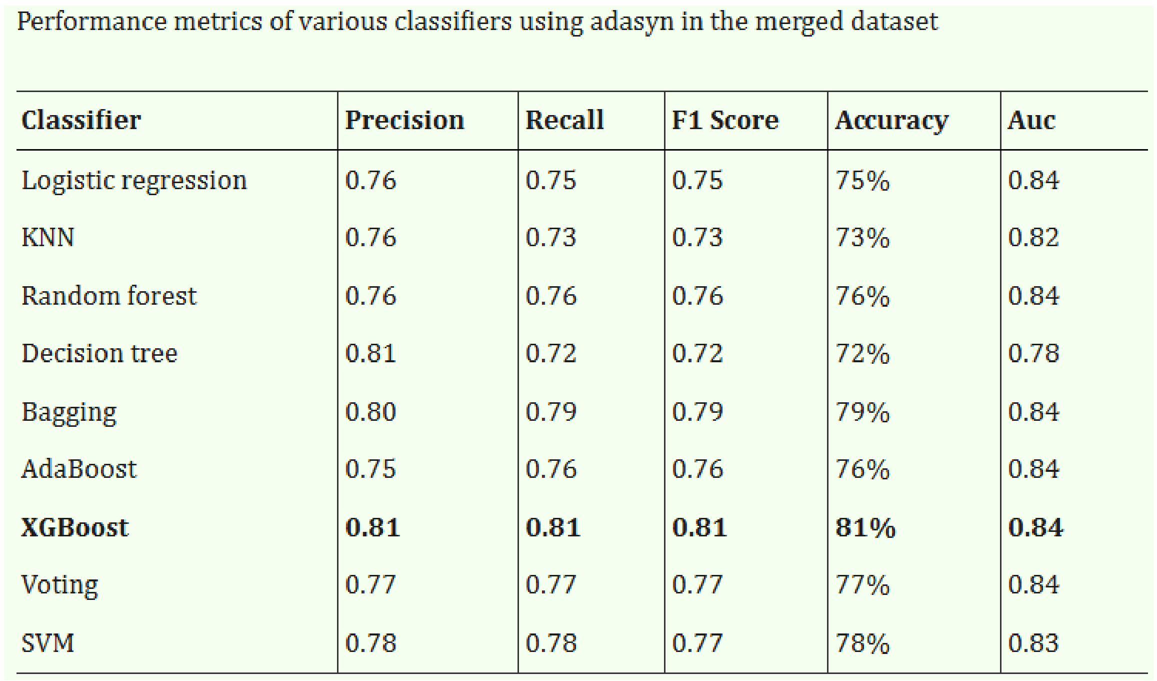

QUERY 1: PubMed "Machine Learning" AND "Diabetes" 139
QUERY 2: PubMed "Data Mining" AND "Diabetes" 268
QUERY 3: DBLP "Diabetes" 880
< 5 years
< 5 years
< 5 years
<5years 110
<5years 184
<5years 248
Manual Inspection
Manual Inspection
Manual Inspection
54
36
13
Final Collection 103
Biomarkers Identification and Prediction 48
Diabetic Complications 31
Drugs and Therapies 13
Genetic Background and Environment 6
Health Care Management 5

Figure 4: Workflow diagram of Research 1.

- **Research 2 (National Library of Medicine Diabetes Prediction)**

This research is based on the supervised learning using classification algorithms to predict diabetes using medical data. According to the IDF (International Diabetes Federation) statistics in the research about 537 million people had diabetes all around the world in 2021. The most suffered country according to the statistics were Bangladesh with around 7.10 million people affected with this disease. The open source Pima Indian and a private data set of female Bangladeshi patients' data were used for this research. This research used various algorithms with accuracy around 60% to 70% but the highest accuracy was obtained by the XG boost classifier with an accuracy rate of 81% and an AUC and F1 score of 0.84 and 0.81 respectively. The study concludes that ML classification models can assist significantly the healthcare professionals by providing a fast and reliable data. This research also highlights the challenges faced during the research like the data imbalance and the model interpretability (Khan, 2022).


Performance metrics of various classifiers using adasyn in the merged dataset

Classifier	Precision	Recall	F1 Score	Accuracy	Auc
Logistic regression	0.76	0.75	0.75	75%	0.84
KNN	0.76	0.73	0.73	73%	0.82
Random forest	0.76	0.76	0.76	76%	0.84
Decision tree	0.81	0.72	0.72	72%	0.78
Bagging	0.80	0.79	0.79	79%	0.84
AdaBoost	0.75	0.76	0.76	76%	0.84
XGBoost	**0.81**	**0.81**	**0.81**	**81%**	**0.84**
Voting	0.77	0.77	0.77	77%	0.84
SVM	0.78	0.78	0.77	78%	0.83

Figure 5: Performance Metrics of Research 2.

- **Research 3 (Machine Learning Based Approaches for Diabetes Prediction)**

According to the MDPI research as per reports 8.5% of the world population in 2014 as suffering from diabetes where type-2 were for 95% of them. The diabetes patients were increasing day by day. till the date 2019 1.5 million deaths were caused by diabetes. Although diabetes is affected to both men and women but according to the research women has a higher chance of developing diabetes. The Pima Indian Diabetes dataset is used in this research. Many algorithms were used in this project like Decision Tree with an accuracy rate of 72% similarly Naive Bayes with an accuracy rate of 77% but the highest accuracy rate among all the algorithms was Logistic Regression with an accuracy rate of 80%. At last in the research it says early detection is crucial for preventing or delaying the development of serious diabetes cases. This research collected the dataset and using the help of the dataset created heatmap analytical tools, ML based prediction model which would be helpful for the health sectors to predict early diabetes (Ahmed, 2024).

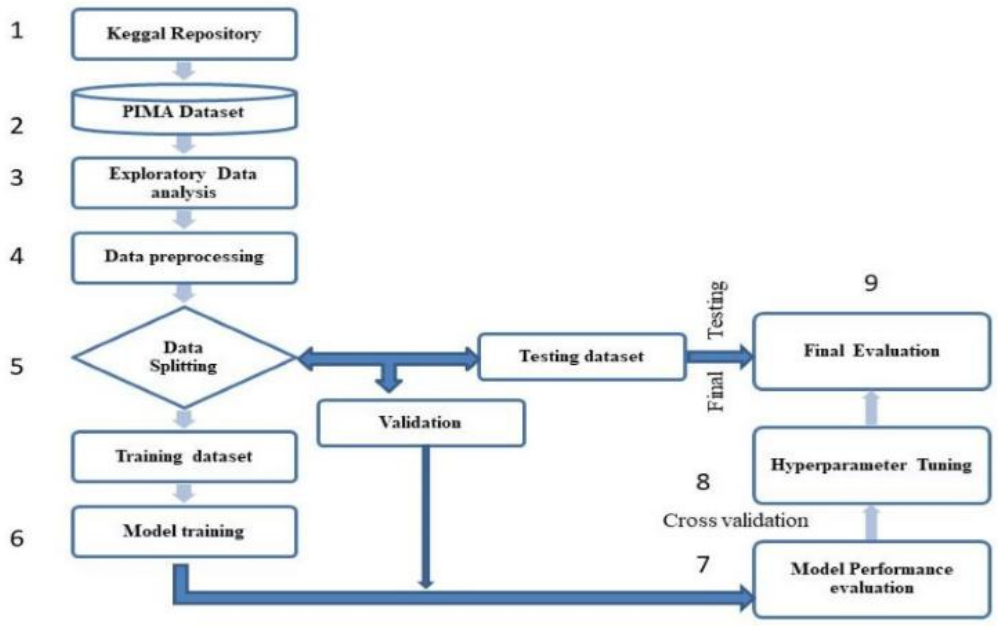


Figure 6: Flowchart of the Methodology used for the ML Model Development.

2.3 Framework and Tools Used

- Python
- Pandas, NumPy, sklearn
- Matplotlib/ seaborn
- Jupiter Notebook
- GitHub

2.4 Advantages, Drawbacks and Issues

Advantages:

- Diabetes ML classification model can help detect early diabetes which might help preventing or delaying the development of serious diabetes cases.
- Algorithms selected in this project like Logistic Regression, Random Forest Classifier, Support Vector Machine (SVM) provide reliable performance in predicting diabetes.
- The selected dataset contains relevant clean medical features making the predictions accurate and applicable to the real-world health care scenarios.

Drawbacks:

- The performance of the ML learning models depends on the data quality and proper pre-processing.
- Complex models may lack some interpretability making this a bit difficult for healthcare professionals.
- Training advanced models may need better resources.

Issues:

- The database models mostly contain more non-diabetic patient cases rather than diabetic patient cases which can be bias model prediction if not trained properly.
- Models trained on a specific dataset may not perform the same or with the same accuracy on different regions.
- Without proper validation the model may perform with better accuracy on the testing but different on unseen data.

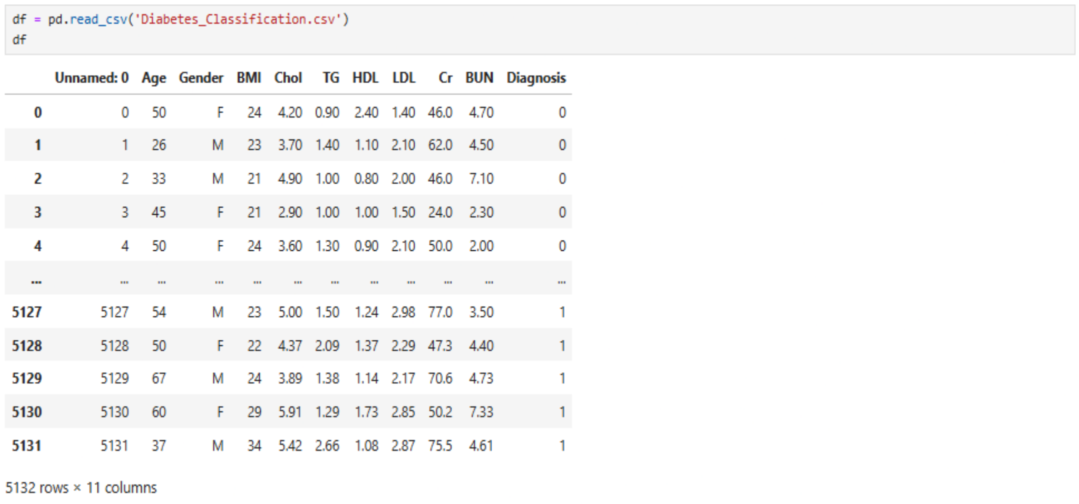
2.5 Dataset Information and Background

The dataset used in this project is a Diabetes Classification Dataset. The dataset was obtained from Opendatabay.

https://www.opendatabay.com/data/healthcare/4ea2d478-76c7-42eb-9c80-0d69050d2e40

The dataset contains 5132 rows and 11 columns with the following attributes.

- Age
- Gender
- Body Mass Index (BMI)
- Cholesterol (Chol)
- Triglycerides (TG)
- High Density Lipoprotein (HDL)
- Low Density Lipoprotein (LDL)
- Creatinine (Cr)
- Blood Urea Nitrogen (BUN)
- Diagnosis



```
df = pd.read_csv('Diabetes_Classification.csv')
df
```

	Unnamed: 0	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN	Diagnosis
0	0	50	F	24	4.20	0.90	2.40	1.40	46.0	4.70	0
1	1	26	M	23	3.70	1.40	1.10	2.10	62.0	4.50	0
2	2	33	M	21	4.90	1.00	0.80	2.00	46.0	7.10	0
3	3	45	F	21	2.90	1.00	1.00	1.50	24.0	2.30	0
4	4	50	F	24	3.60	1.30	0.90	2.10	50.0	2.00	0
...	...	...	...	...	...	...	...	...	...	...	...
5127	5127	54	M	23	5.00	1.50	1.24	2.98	77.0	3.50	1
5128	5128	50	F	22	4.37	2.09	1.37	2.29	47.3	4.40	1
5129	5129	67	M	24	3.89	1.38	1.14	2.17	70.6	4.73	1
5130	5130	60	F	29	5.91	1.29	1.73	2.85	50.2	7.33	1
5131	5131	37	M	34	5.42	2.66	1.08	2.87	75.5	4.61	1

5132 rows × 11 columns

Figure 7: Diabetes Dataset.

The dataset contains both input values X and target values y which is Diagnosis.

3. Solution

3.1 Proposed Solution

The project uses a ML based supervised learning module using classification approach to predict whether a person is diabetic or non-diabetic based on their medical data provided. The prediction process begins with the loading the dataset and pre-processing the data such as handling the categorical variables like gender. The pre-processed data is then divided into training and test sets. For this model three best classification algorithms were selected (Logistic Regression, Random Forest Classifier and SVM). These three algorithms are trained and are evaluated using performance metrics like accuracy, precision to measure their effectiveness in the diabetes prediction. The proposed solution helps in identifying early diabetes cases which might help prevent the individuals from serious diabetic cases.

3.2 Algorithms Used

- **Random Forest Classifier**

  Random Forest Classifier is a supervised learning algorithm that builds or combines multiple decision trees to produce an accurate single result. Random forest classifier can handle both numerical and categorical data. This algorithm can handle both numerical and categorical data and is capable of capturing complex relationships between functions (Kavlakoglu, 2025). This algorithm is suitable for this project because medical data mostly contains non-linear interactions between variables such as BMI, age and the ensemble of this algorithm reduces overfitting.

- **Logistic Regression**

Logistic regression is a supervised machine learning algorithm widely used for binary classification problems. Logistic regression is a type of classification algorithm that predicts discrete binary outcomes (Lee, 202). Logistic regression is suited for this project because it is simple, efficient and also allows easy understanding of how medical data contributes to the diabetes prediction.

- **Support Vector Machine (SVM)**

Support Vector Machine (SVM) is a supervised learning algorithm that classifies data by finding an optimal line or hyperplane that separates different classes with maximum margin. SVM is effective in high dimensional spaces and can also model both linear and non-linear decision boundaries (Kavlakogl, 2025). SVM is perfect for this project because it performs best with medium sized datasets and can handle complex decisions. SVM is robust to overfitting making it effective for medical projects and diabetes prediction.

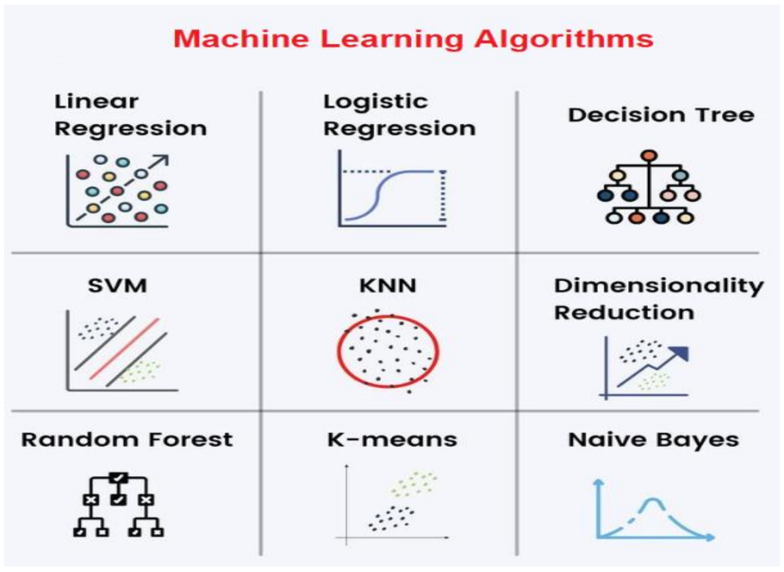


Figure 8: Machine Learning Algorithms.

3.3 Pseudocode

Pseudocode is a detailed description of what a program or algorithm should do. It is written in a formal and readable natural syntax and formatting state so that programmers can easily understand the development process. It basically serves as a blueprint for translating codes logic into an actual programming language (Sheldon, 2023).

3.3.1 Pseudocode for Overall System

```
START
    IMPORT required libraries
    LOAD diabetes dataset
    DISPLAY dataset information and check data type
    IF missing values exists
        THEN handle missing values
    END IF

    SELECT input features
    SELECT target variable
    GENERATE heatmap, bar graph and scatter plot

    INITIALIZE Logistic Regression model
    TRAIN Logistic Regression model
    PREDICT test results
    CALCULATE accuracy score

    INITIALIZE SVM model
    TRAIN SVM model
    PREDICT test results
    CALCULATE accuracy score

    INITIALIZE Random forest model
    TRAIN Random forest model
    PREDICT test results
    CALCULATE accuracy score

    COMPARE accuracy scores of all the models

    SELECT model with highest accuracy
    DISPLAY best performing algorithm

    IF new patient data is provided
        APPLY same pre-processing steps
        PREDICT diabetes status
        DISPLAY result
    END IF
END
```

3.3.2 Pseudocode for Logistic Regression

```
START
    IMPORT required libraries
    LOAD diabetes dataset
    SELECT input features
    SELECT target variable
    PREPROCESS dataset

    SPLIT X and y into training and testing sets
    INITIALIZE Logistic Regression model
    TRAIN model using training data
    CALCULATE predicted values for test data

    FOR EACH prediction
    APPLY sigmoid decision boundary
    CLASSIFY as Diabetic or Non-Diabetic
    END FOR
    EVALUATE model performance
    RETURN Logistic Regression accuracy
END
```

3.3.3 Pseudocode for Random Forest Classifier

```
START
    IMPORT required libraries
    LOAD diabetes dataset
    SELECT input features
    SELECT target variable
    PREPROCESS dataset

    SPLIT X and y into training and testing sets
    INITIALIZE Random Forest with N decision trees
    TRAIN model using training data
    FOR EACH tree in forest
    DO
    SELECT random sample of data
    BUILD decision tree
    END FOR

    CALCULATE predicted values for test data
    EVALUATE model performance
    RETURN random forest accuracy
END
```

3.3.4 Pseudocode for Support Vector Machine (SVM)

```
START
    IMPORT required libraries
    LOAD diabetes dataset

    SELECT input features
    SELECT target variable

    PREPROCESS dataset
    SPLIT X and y into training and testing sets
    INITIALIZE SVM classifier with kernel function
    MAP input data
    FIND optional hyperplane that maximizes margin
    SEPARATE diabetic and non-diabetic classes

    PREDICT class labels for test data
    EVALUATE model performance
    RETURN SVM accuracy
END
```

3.4 Flowchart

3.4.1 Flowchart Diagram of Overall System

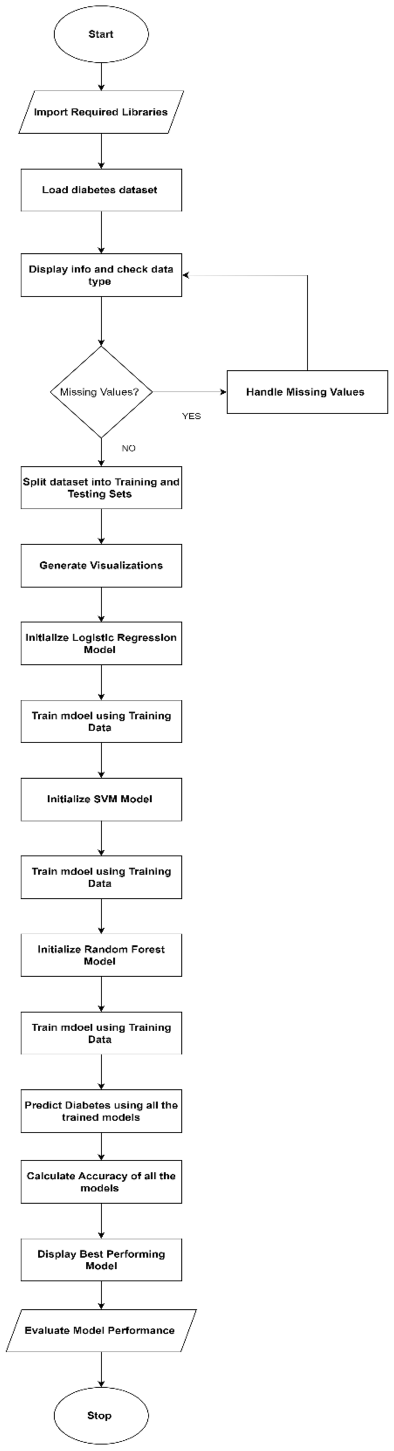

Start
Import Required Libraries
Load diabetes dataset
Display info and check data type
Missing Values?
YES
Handle Missing Values
NO
Split dataset into Training and Testing Sets
Generate Visualizations
Initialize Logistic Regression Model
Train mdoel using Training Data
Initialize SVM Model
Train mdoel using Training Data
Initialize Random Forest Model
Train mdoel using Training Data
Predict Diabetes using all the trained models
Calculate Accuracy of all the models
Display Best Performing Model
Evaluate Model Performance
Stop

Figure 9: Flowchart Diagram of Overall System.

3.4.1 Flowchart Diagram of Logistic Regression

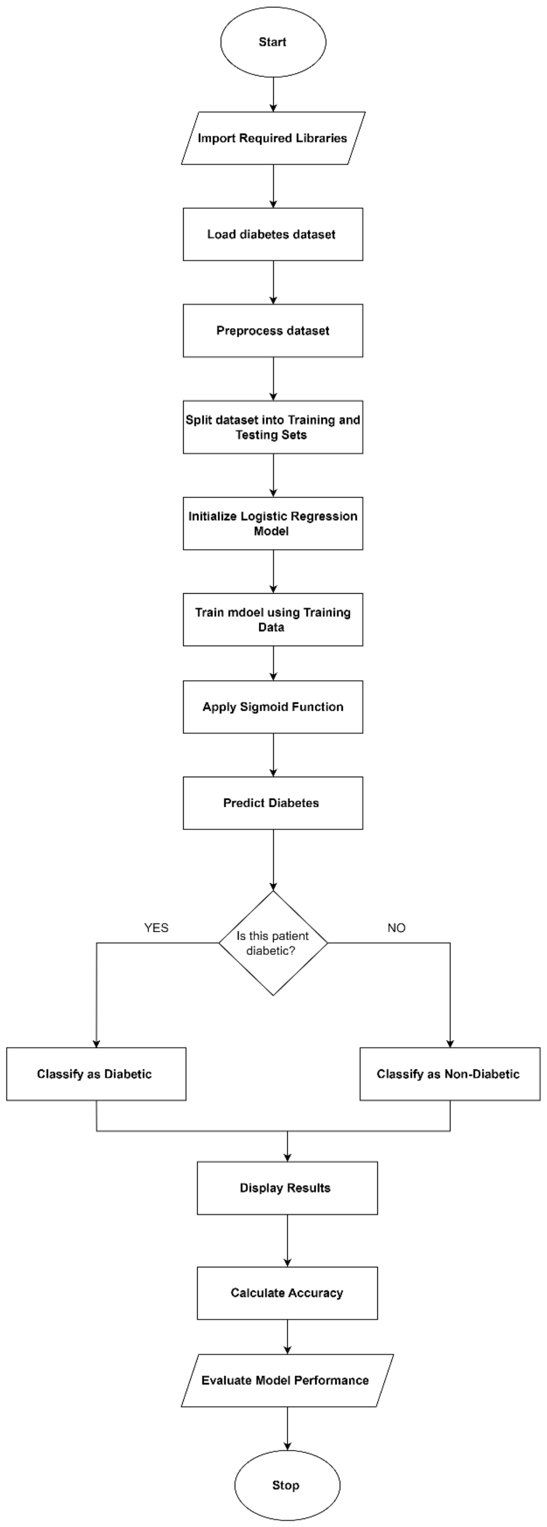

Start
Import Required Libraries
Load diabetes dataset
Preprocess dataset
Split dataset into Training and Testing Sets
Initialize Logistic Regression Model
Train mdoel using Training Data
Apply Sigmoid Function
Predict Diabetes
YES
Is this patient diabetic?
NO
Classify as Diabetic
Classify as Non-Diabetic
Display Results
Calculate Accuracy
Evaluate Model Performance
Stop

Figure 10: Flowchart Diagram of Logistic Regression.

3.4.2 Flowchart Diagram of Random Forest Classifier

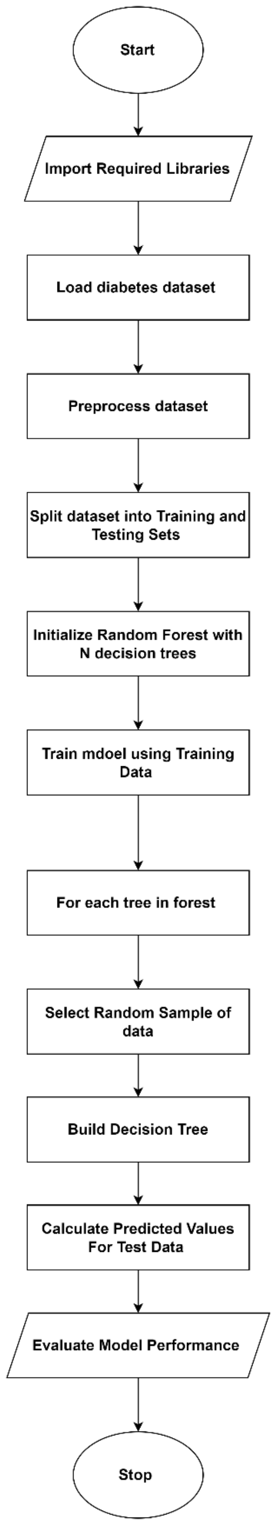

Start
Import Required Libraries
Load diabetes dataset
Preprocess dataset
Split dataset into Training and Testing Sets
Initialize Random Forest with N decision trees
Train mdoel using Training Data
For each tree in forest
Select Random Sample of data
Build Decision Tree
Calculate Predicted Values For Test Data
Evaluate Model Performance
Stop

Figure 11: Flowchart Diagram of Random Forest Classifier

3.4.3 Flowchart Diagram of Support Vector Machine (SVM)

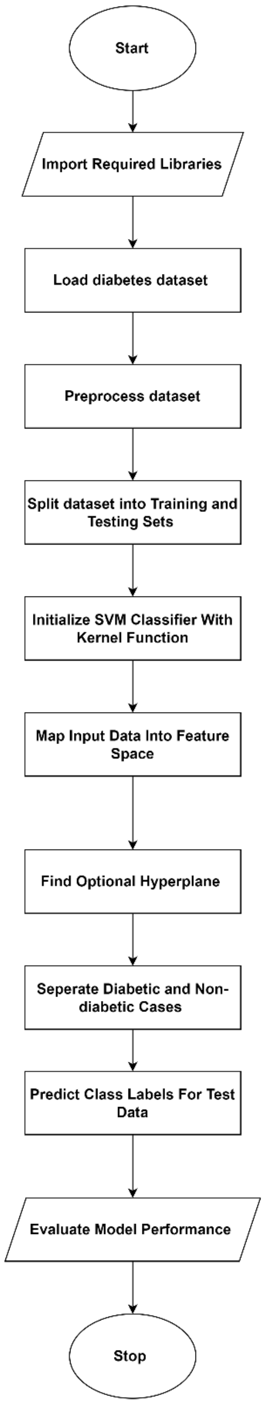

Start
Import Required Libraries
Load diabetes dataset
Preprocess dataset
Split dataset into Training and Testing Sets
Initialize SVM Classifier With Kernel Function
Map Input Data Into Feature Space
Find Optional Hyperplane
Seperate Diabetic and Non-diabetic Cases
Predict Class Labels For Test Data
Evaluate Model Performance
Stop

Figure 12: Flowchart Diagram of Support Vector Machine.

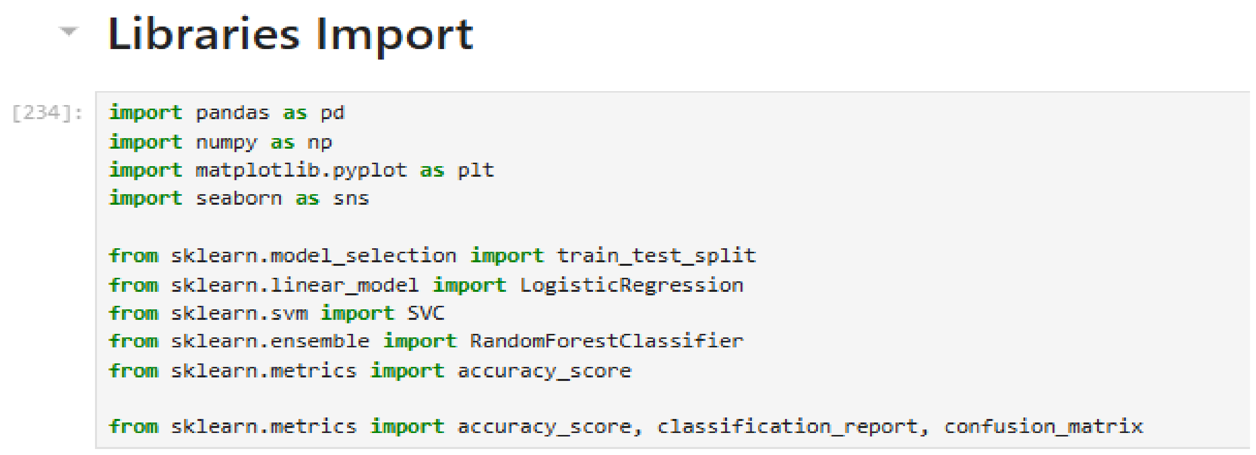
3.5 Development Process

The development of this project was carried out using the python programming language for proper data analytics and machine learning. This project was implemented on the Jupiter notebook platform.

The development process began with the data cleaning, fixing nan values and pre-prepossessing the data. Data visualization was done with the help of the heat map, bar graph and scatter plot to understand the data properly. After the data preparation supervised learning models were implemented such as Logistic regression, SVM and Random Forest Classifier. The models were evaluated using the performance tests such as recall, F1-score, precision and accuracy to know about the prediction effectiveness.

The following key libraries were used for the development process such as pandas for the data loading and pre-processing of the diabetes dataset, NumPy for the numerical computations, Scikit-learn used for implementing machine learning algorithms for train test split, matplotlib and seaborn for the data visualization.

3.5.1 Libraries Import



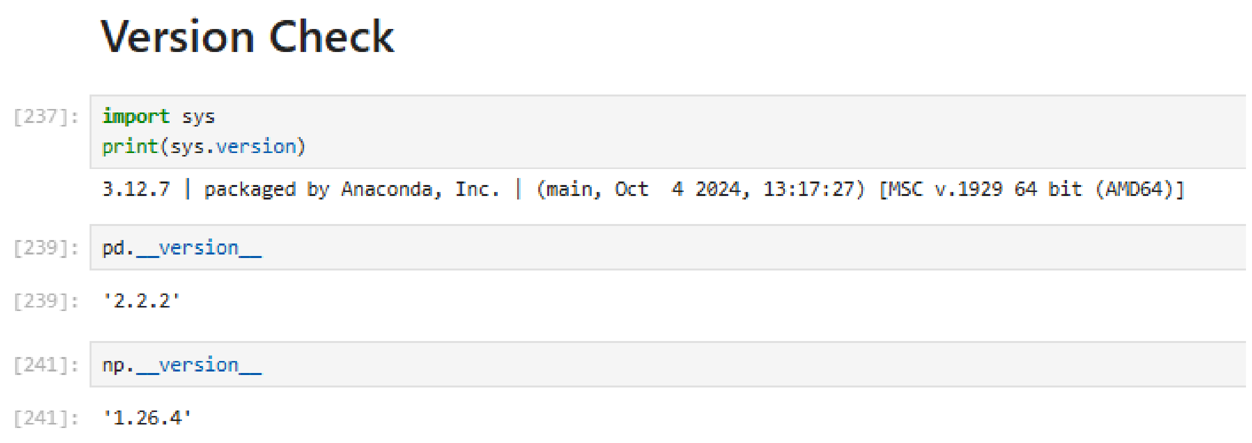
Libraries Import

```
[234]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Figure 13: Libraries Import.

All the required libraries are being imported in this phase at the beginning of the development process. Libraries such as Pandas and NumPy are being used for data handling and numerical operations. Matplotlib and Seaborn are used for data visualization process.

3.5.2 Version Check



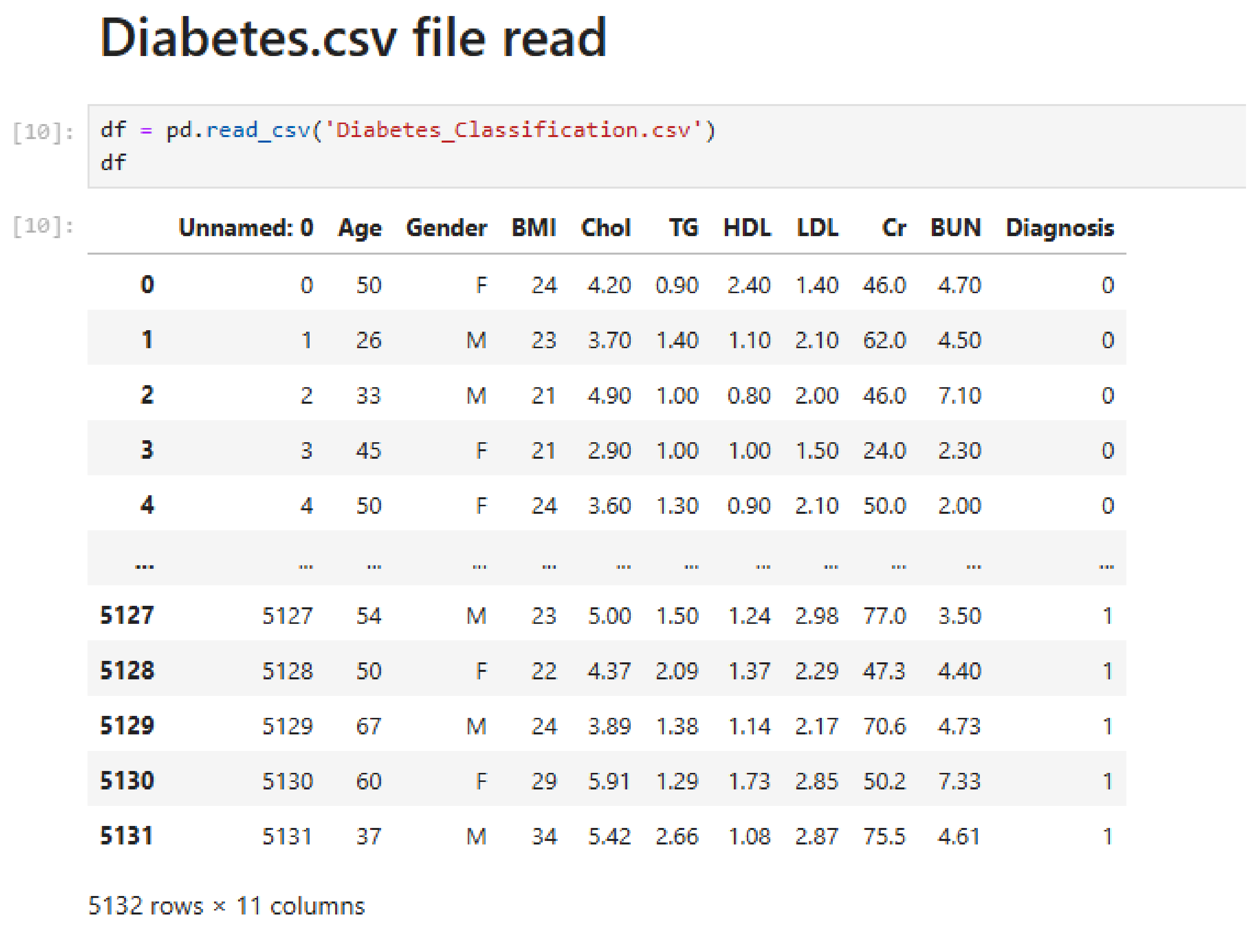
Version Check

```
[237]: import sys
       print(sys.version)

3.12.7 | packaged by Anaconda, Inc. | (main, Oct  4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]

[239]: pd.__version__

[239]: '2.2.2'

[241]: np.__version__

[241]: '1.26.4'
```

Figure 14: Version Check.

This image displays the version information of the python key libraries.

3.5.3 CSV File Read



Diabetes.csv file read

```
[10]: df = pd.read_csv('Diabetes_Classification.csv')
      df
```

[10]:

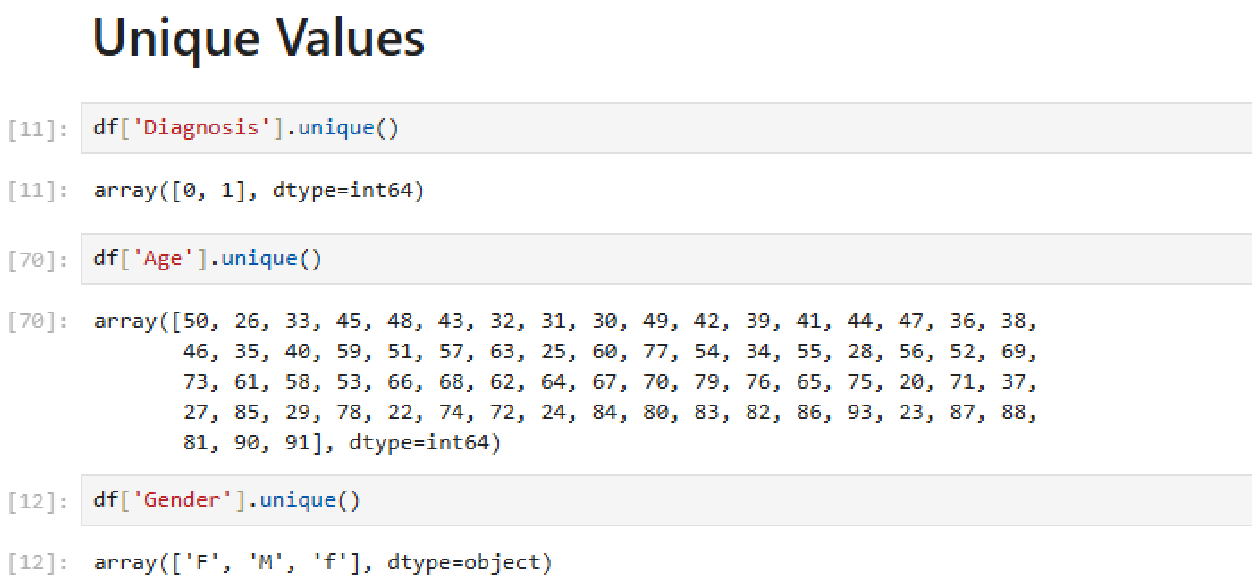
	Unnamed: 0	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN	Diagnosis
0	0	50	F	24	4.20	0.90	2.40	1.40	46.0	4.70	0
1	1	26	M	23	3.70	1.40	1.10	2.10	62.0	4.50	0
2	2	33	M	21	4.90	1.00	0.80	2.00	46.0	7.10	0
3	3	45	F	21	2.90	1.00	1.00	1.50	24.0	2.30	0
4	4	50	F	24	3.60	1.30	0.90	2.10	50.0	2.00	0
...	...	...	...	...	...	...	...	...	...	...	...
5127	5127	54	M	23	5.00	1.50	1.24	2.98	77.0	3.50	1
5128	5128	50	F	22	4.37	2.09	1.37	2.29	47.3	4.40	1
5129	5129	67	M	24	3.89	1.38	1.14	2.17	70.6	4.73	1
5130	5130	60	F	29	5.91	1.29	1.73	2.85	50.2	7.33	1
5131	5131	37	M	34	5.42	2.66	1.08	2.87	75.5	4.61	1

5132 rows × 11 columns

Figure 15: Diabeties.csv File Read.

This image displays the "Diabetes_Classification.csv" file being read by the Jupiter notebook environment.

3.5.4 Unique Values



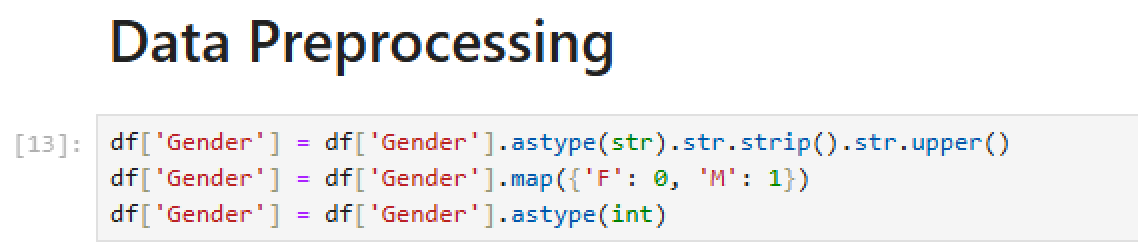
Unique Values

```
[11]: df['Diagnosis'].unique()

[11]: array([0, 1], dtype=int64)

[70]: df['Age'].unique()

[70]: array([50, 26, 33, 45, 48, 43, 32, 31, 30, 49, 42, 39, 41, 44, 47, 36, 38,
             46, 35, 40, 59, 51, 57, 63, 25, 60, 77, 54, 34, 55, 28, 56, 52, 69,
             73, 61, 58, 53, 66, 68, 62, 64, 67, 70, 79, 76, 65, 75, 20, 71, 37,
             27, 85, 29, 78, 22, 74, 72, 24, 84, 80, 83, 82, 86, 93, 23, 87, 88,
             81, 90, 91], dtype=int64)

[12]: df['Gender'].unique()

[12]: array(['F', 'M', 'f'], dtype=object)
```

Figure 16: Unique Values.

This image displays the unique values of the Diagnosis column and the unique values of the Gender attribute in the dataset.

3.5.5 Data Pre-processing



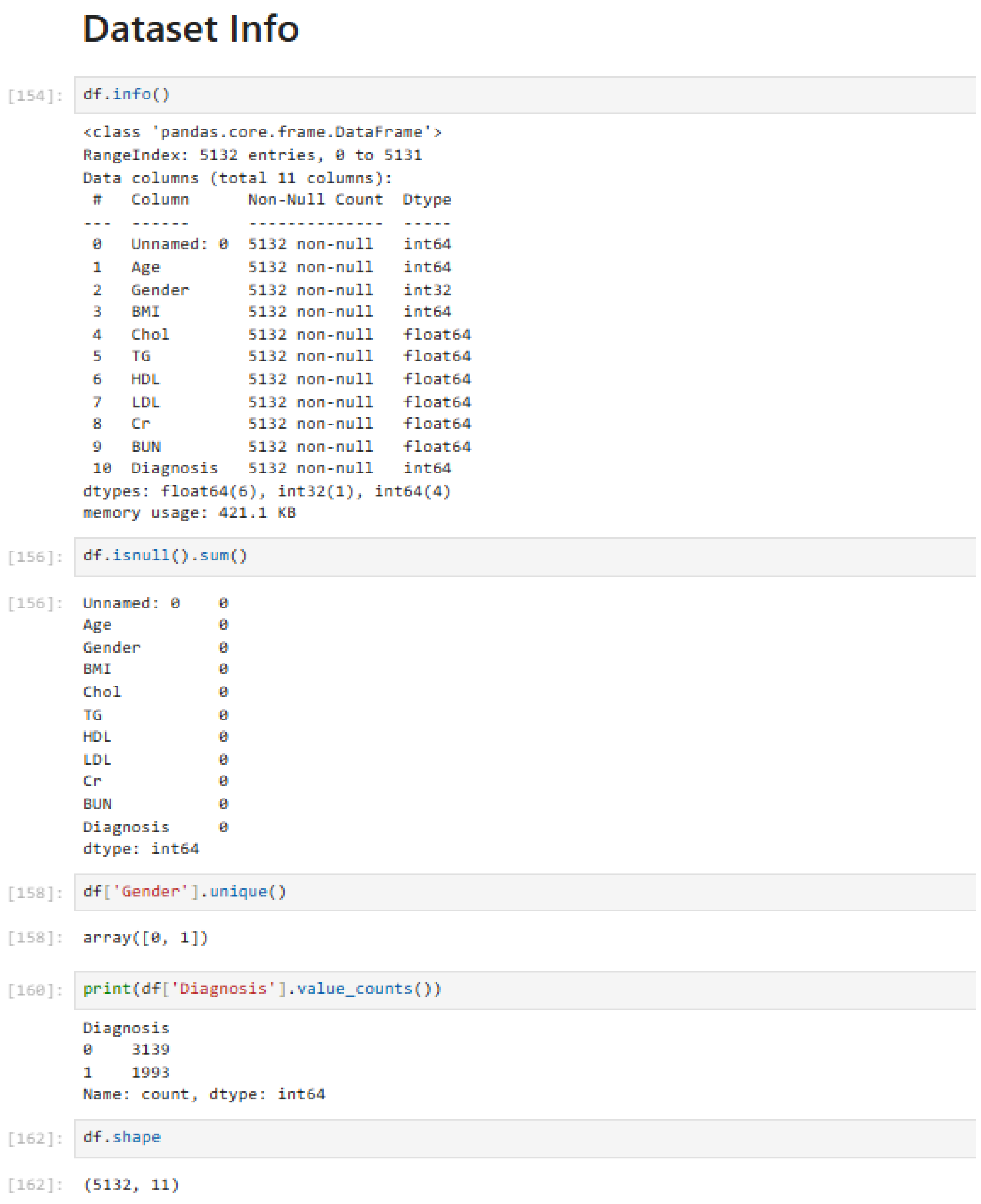
Data Preprocessing

```
[13]: df['Gender'] = df['Gender'].astype(str).str.strip().str.upper()
      df['Gender'] = df['Gender'].map({'F': 0, 'M': 1})
      df['Gender'] = df['Gender'].astype(int)
```

Figure 17: Data Preprocessing.

This image displays the pre-processing stage where the dataset is prepared for the model training converting the values into upper case.

3.5.6 Dataset Information



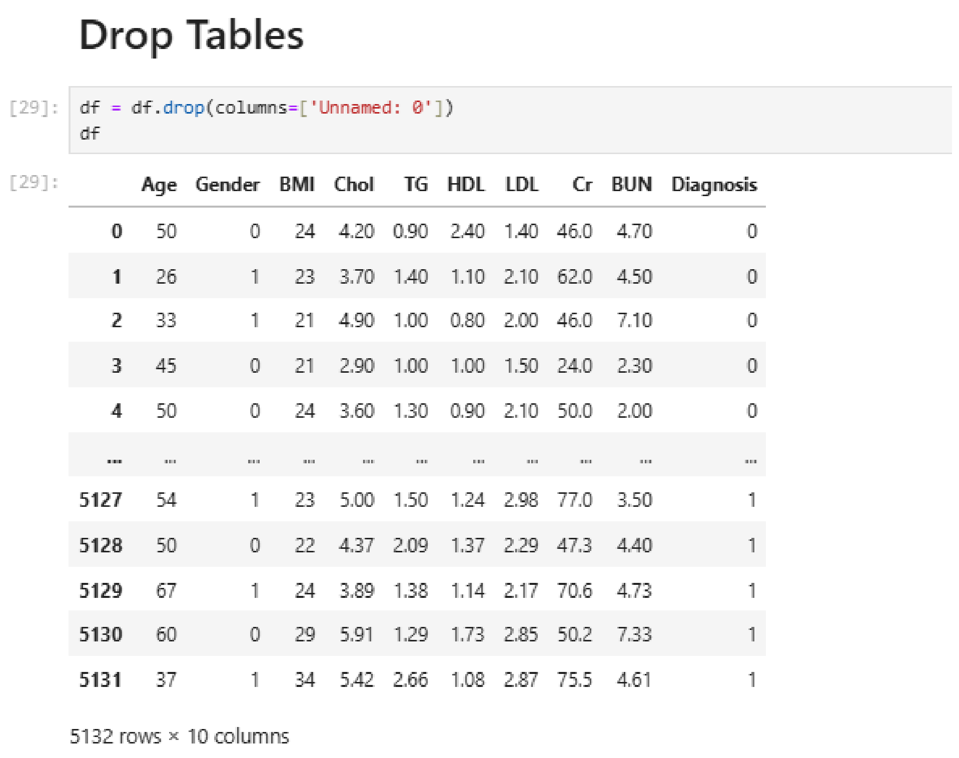
Dataset Info

```
[154]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5132 entries, 0 to 5131
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  5132 non-null   int64
 1   Age         5132 non-null   int64
 2   Gender      5132 non-null   int32
 3   BMI         5132 non-null   int64
 4   Chol        5132 non-null   float64
 5   TG          5132 non-null   float64
 6   HDL         5132 non-null   float64
 7   LDL         5132 non-null   float64
 8   Cr          5132 non-null   float64
 9   BUN         5132 non-null   float64
 10  Diagnosis   5132 non-null   int64
dtypes: float64(6), int32(1), int64(4)
memory usage: 421.1 KB

[156]: df.isnull().sum()

[156]: Unnamed: 0    0
       Age           0
       Gender        0
       BMI           0
       Chol          0
       TG            0
       HDL           0
       LDL           0
       Cr            0
       BUN           0
       Diagnosis     0
       dtype: int64

[158]: df['Gender'].unique()

[158]: array([0, 1])

[160]: print(df['Diagnosis'].value_counts())

Diagnosis
0    3139
1    1993
Name: count, dtype: int64

[162]: df.shape

[162]: (5132, 11)
```

Figure 18: Dataset Info.

This image displays the information about the diabetes dataset.

3.5.7 Drop Table



Drop Tables

```
[29]: df = df.drop(columns=['Unnamed: 0'])
      df
```

[29]:

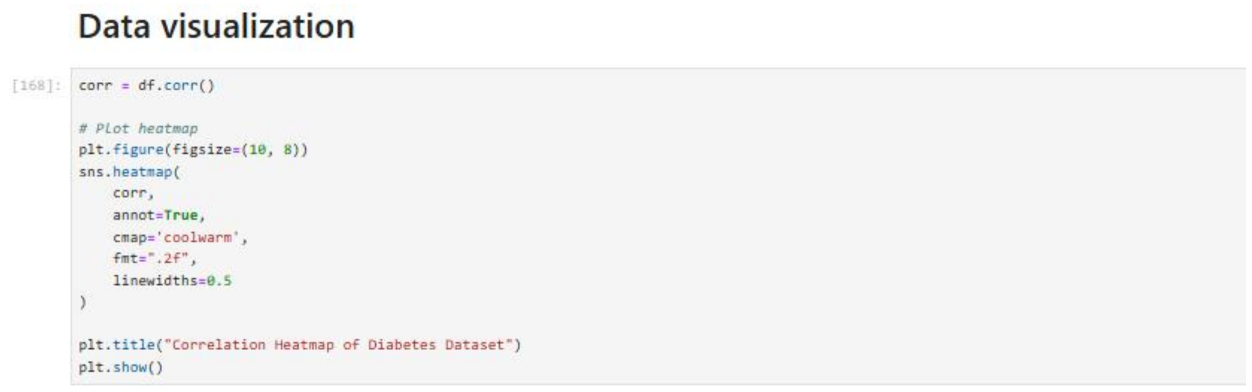
	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN	Diagnosis
0	50	0	24	4.20	0.90	2.40	1.40	46.0	4.70	0
1	26	1	23	3.70	1.40	1.10	2.10	62.0	4.50	0
2	33	1	21	4.90	1.00	0.80	2.00	46.0	7.10	0
3	45	0	21	2.90	1.00	1.00	1.50	24.0	2.30	0
4	50	0	24	3.60	1.30	0.90	2.10	50.0	2.00	0
...	...	...	...	...	...	...	...	...	...	...
5127	54	1	23	5.00	1.50	1.24	2.98	77.0	3.50	1
5128	50	0	22	4.37	2.09	1.37	2.29	47.3	4.40	1
5129	67	1	24	3.89	1.38	1.14	2.17	70.6	4.73	1
5130	60	0	29	5.91	1.29	1.73	2.85	50.2	7.33	1
5131	37	1	34	5.42	2.66	1.08	2.87	75.5	4.61	1

5132 rows × 10 columns

Figure 19: Drop Tables.

This image displays the Unnamed: 0 column being dropped.

3.5.8 Data Visualization



Data visualization

[168]:
```python
corr = df.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(
    corr,
    annot=True,
    cmap='coolwarm',
    fmt=".2f",
    linewidths=0.5
)

plt.title("Correlation Heatmap of Diabetes Dataset")
plt.show()
```

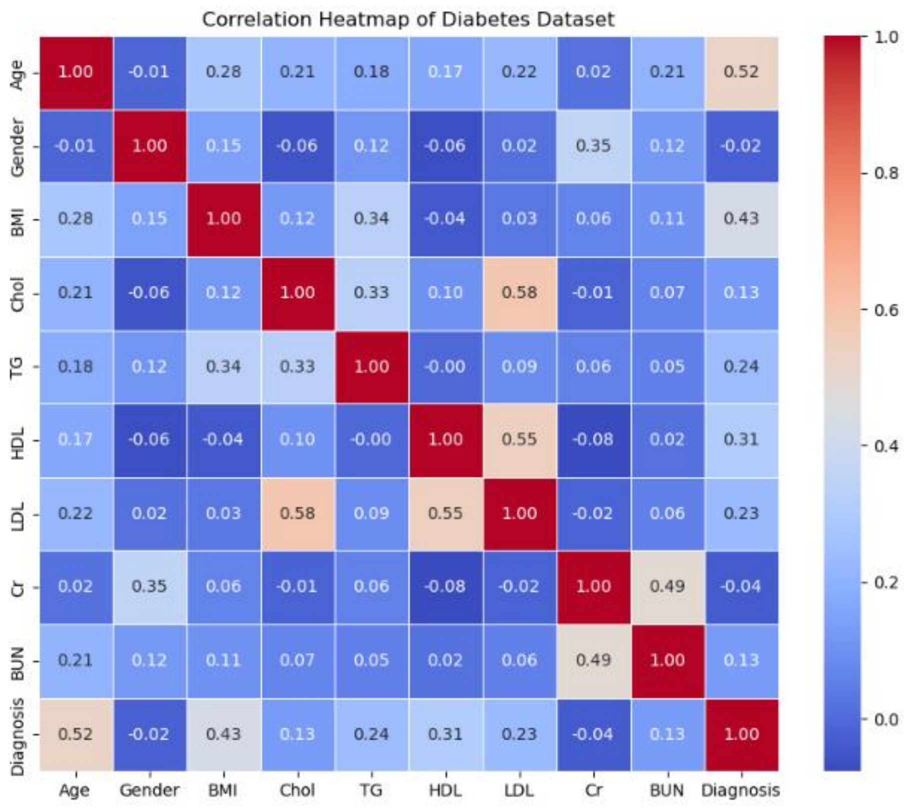
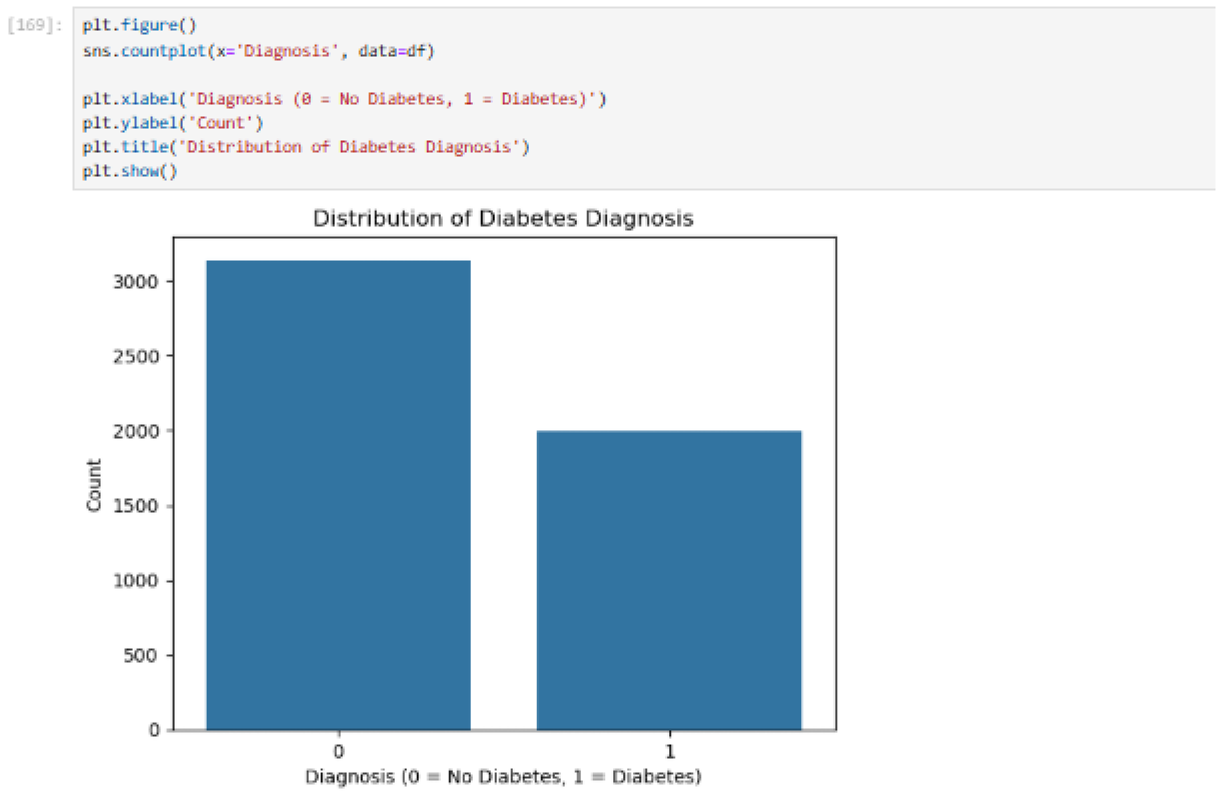
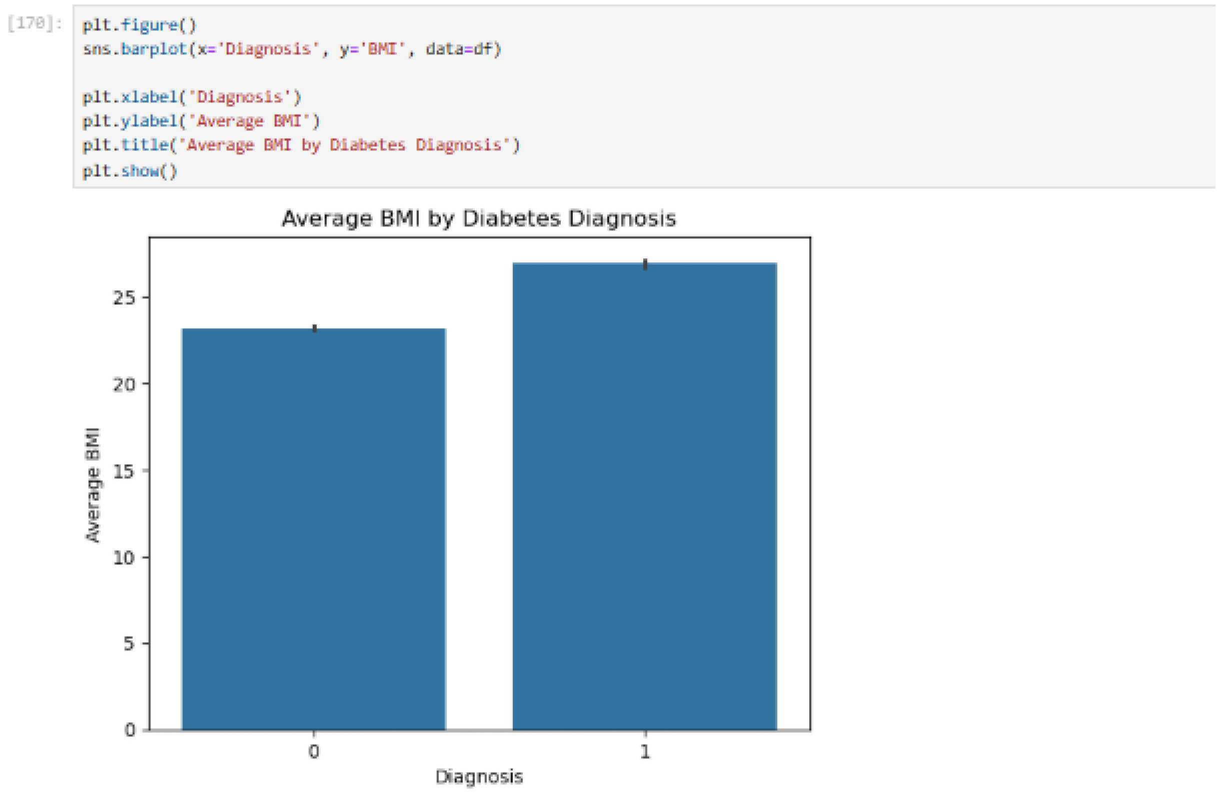


Figure 20: Data Visualization (Heatmap).



```
[169]: plt.figure()
       sns.countplot(x='Diagnosis', data=df)

       plt.xlabel('Diagnosis (0 = No Diabetes, 1 = Diabetes)')
       plt.ylabel('Count')
       plt.title('Distribution of Diabetes Diagnosis')
       plt.show()
```

Distribution of Diabetes Diagnosis
Count
3000
2500
2000
1500
1000
500
0
0
1
Diagnosis (0 = No Diabetes, 1 = Diabetes)



```
[170]: plt.figure()
       sns.barplot(x='Diagnosis', y='BMI', data=df)

       plt.xlabel('Diagnosis')
       plt.ylabel('Average BMI')
       plt.title('Average BMI by Diabetes Diagnosis')
       plt.show()
```

Figure 21: Data Visualization (Bar graph).

```
[171]: plt.figure()

       for label in df['Diagnosis'].unique():
           subset = df[df['Diagnosis'] == label]
           plt.scatter(subset['Age'], subset['BMI'], label=f'Diagnosis {label}')

       plt.xlabel('Age')
       plt.ylabel('BMI')
       plt.title('Age vs BMI by Diabetes Diagnosis')
       plt.legend()
       plt.show()
```

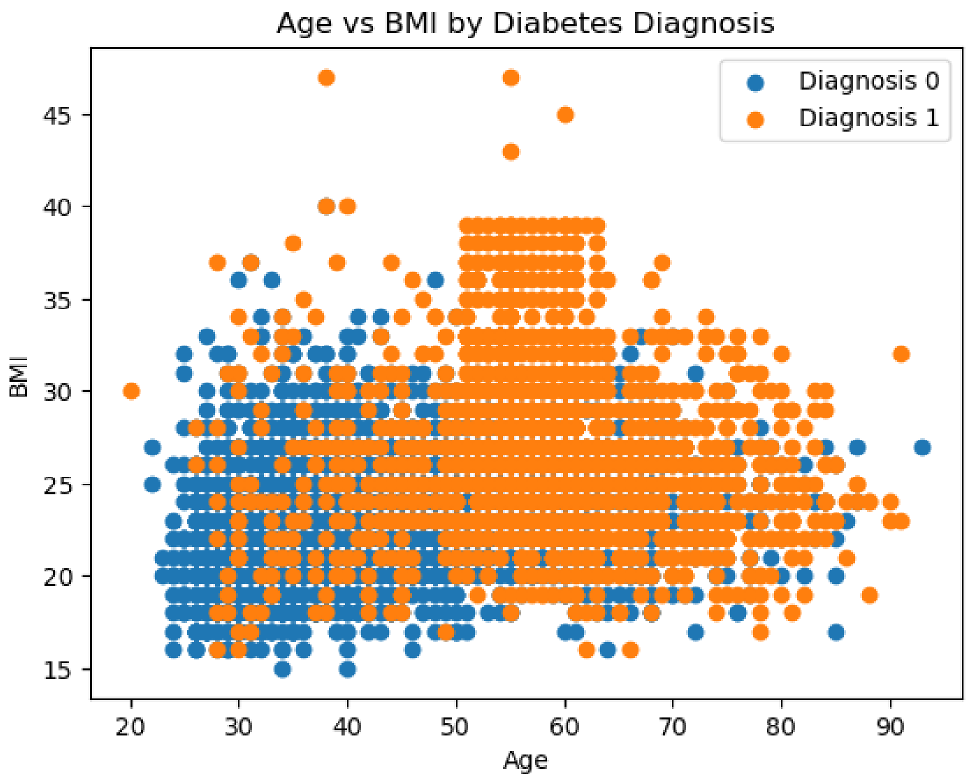
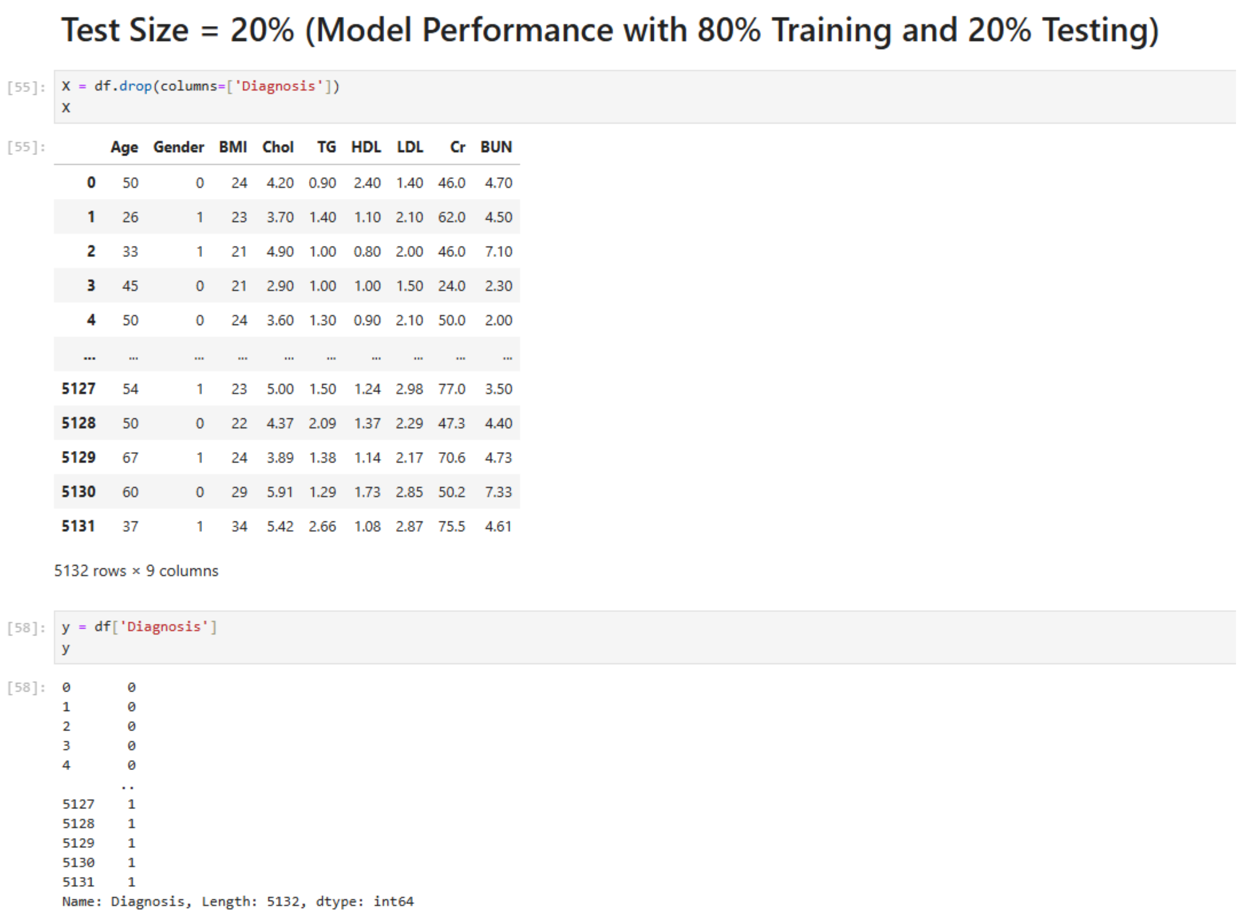


Figure 22: Data Visualization (Scatterplot).

3.5.9 X and y train for Model Performance with 80% Training and 20% Testing



Test Size = 20% (Model Performance with 80% Training and 20% Testing)

```
[55]: X = df.drop(columns=['Diagnosis'])
      X
```

[55]:

	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN
0	50	0	24	4.20	0.90	2.40	1.40	46.0	4.70
1	26	1	23	3.70	1.40	1.10	2.10	62.0	4.50
2	33	1	21	4.90	1.00	0.80	2.00	46.0	7.10
3	45	0	21	2.90	1.00	1.00	1.50	24.0	2.30
4	50	0	24	3.60	1.30	0.90	2.10	50.0	2.00
...	...	...	...	...	...	...	...	...	...
5127	54	1	23	5.00	1.50	1.24	2.98	77.0	3.50
5128	50	0	22	4.37	2.09	1.37	2.29	47.3	4.40
5129	67	1	24	3.89	1.38	1.14	2.17	70.6	4.73
5130	60	0	29	5.91	1.29	1.73	2.85	50.2	7.33
5131	37	1	34	5.42	2.66	1.08	2.87	75.5	4.61

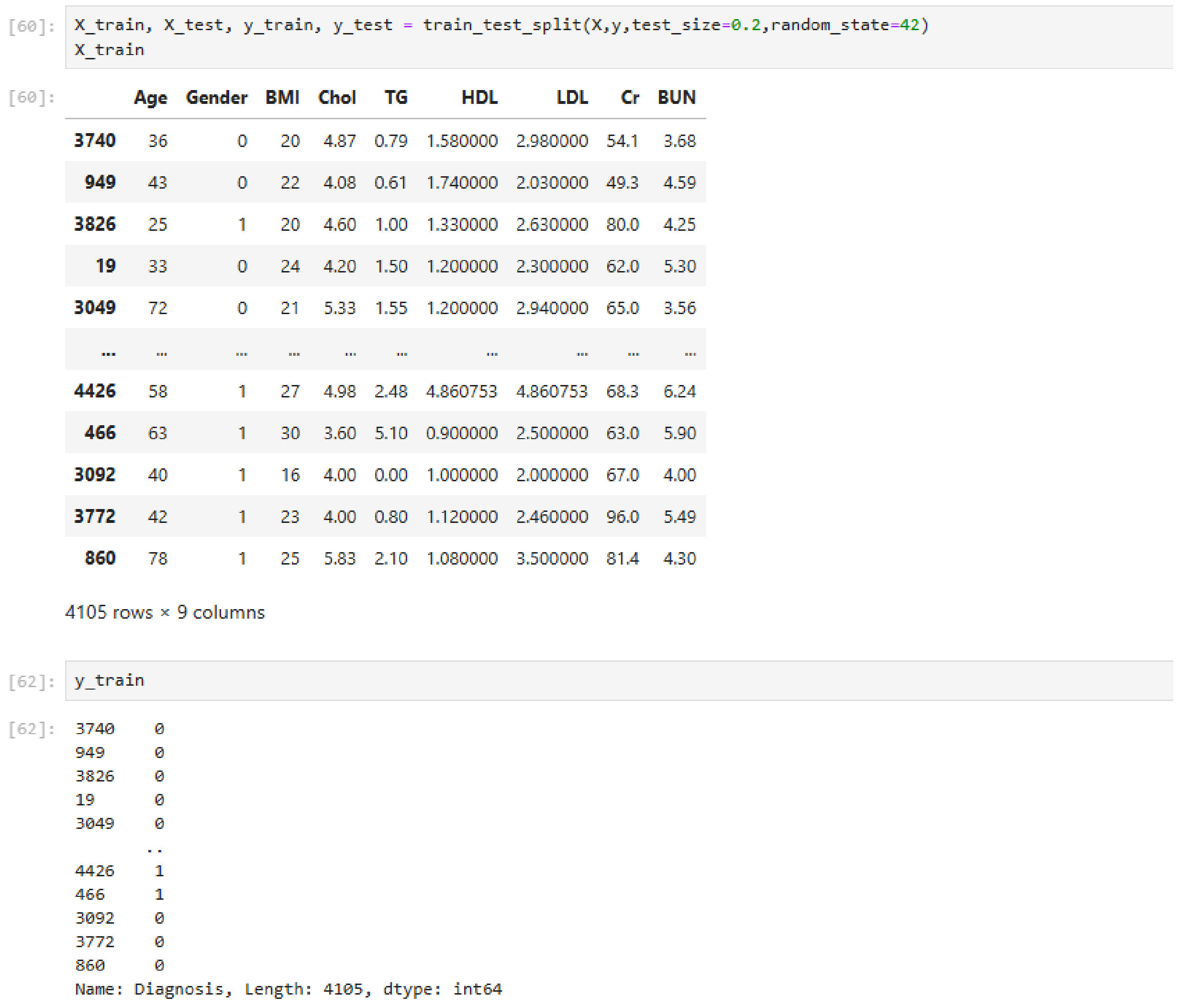
5132 rows × 9 columns

```
[58]: y = df['Diagnosis']
      y
```

```
[58]: 0       0
      1       0
      2       0
      3       0
      4       0
             ..
      5127    1
      5128    1
      5129    1
      5130    1
      5131    1
      Name: Diagnosis, Length: 5132, dtype: int64
```

Figure 18: X and y Train 1 for 20%.

This image displays the separation of the X and y variables. All the medical attributes such as age, BMI, cholesterol, TG, HDL, LDL, Cr, BUN is the know variables X and the Diagnosis is the y variable.



```
[60]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
      X_train
```

[60]:

	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN
3740	36	0	20	4.87	0.79	1.580000	2.980000	54.1	3.68
949	43	0	22	4.08	0.61	1.740000	2.030000	49.3	4.59
3826	25	1	20	4.60	1.00	1.330000	2.630000	80.0	4.25
19	33	0	24	4.20	1.50	1.200000	2.300000	62.0	5.30
3049	72	0	21	5.33	1.55	1.200000	2.940000	65.0	3.56
...	...	...	...	...	...	...	...	...	...
4426	58	1	27	4.98	2.48	4.860753	4.860753	68.3	6.24
466	63	1	30	3.60	5.10	0.900000	2.500000	63.0	5.90
3092	40	1	16	4.00	0.00	1.000000	2.000000	67.0	4.00
3772	42	1	23	4.00	0.80	1.120000	2.460000	96.0	5.49
860	78	1	25	5.83	2.10	1.080000	3.500000	81.4	4.30

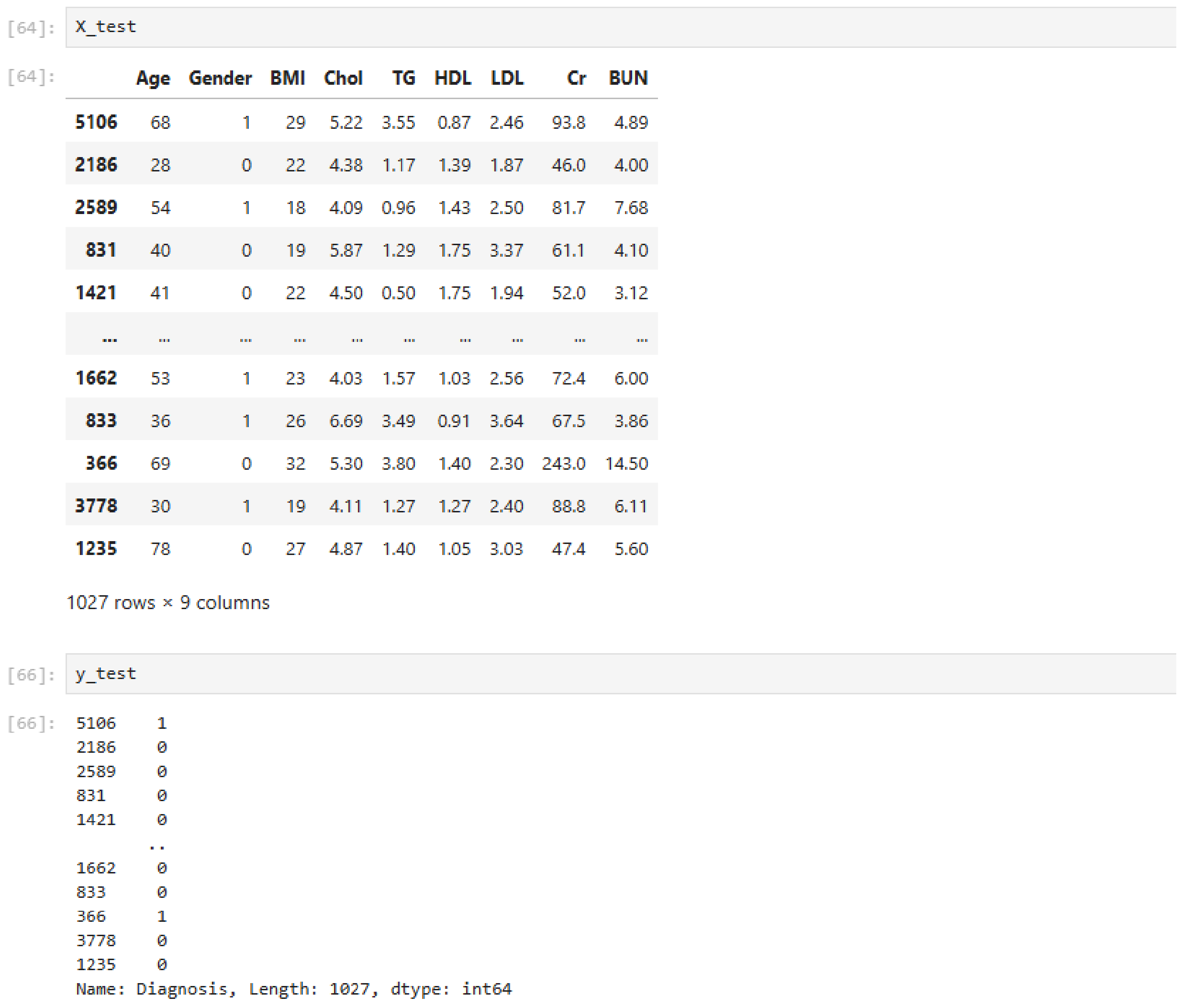
4105 rows × 9 columns

```
[62]: y_train
```

```
[62]: 3740    0
      949     0
      3826    0
      19      0
      3049    0
             ..
      4426    1
      466     1
      3092    0
      3772    0
      860     0
      Name: Diagnosis, Length: 4105, dtype: int64
```

Figure 19: X and y Train 2 for 20%.

This image displays the result of splitting the diabetes dataset into training (80%) and testing (20%) sets using the train_test_split ().



```
[64]: X_test
```

[64]:

	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN
5106	68	1	29	5.22	3.55	0.87	2.46	93.8	4.89
2186	28	0	22	4.38	1.17	1.39	1.87	46.0	4.00
2589	54	1	18	4.09	0.96	1.43	2.50	81.7	7.68
831	40	0	19	5.87	1.29	1.75	3.37	61.1	4.10
1421	41	0	22	4.50	0.50	1.75	1.94	52.0	3.12
...	...	...	...	...	...	...	...	...	...
1662	53	1	23	4.03	1.57	1.03	2.56	72.4	6.00
833	36	1	26	6.69	3.49	0.91	3.64	67.5	3.86
366	69	0	32	5.30	3.80	1.40	2.30	243.0	14.50
3778	30	1	19	4.11	1.27	1.27	2.40	88.8	6.11
1235	78	0	27	4.87	1.40	1.05	3.03	47.4	5.60

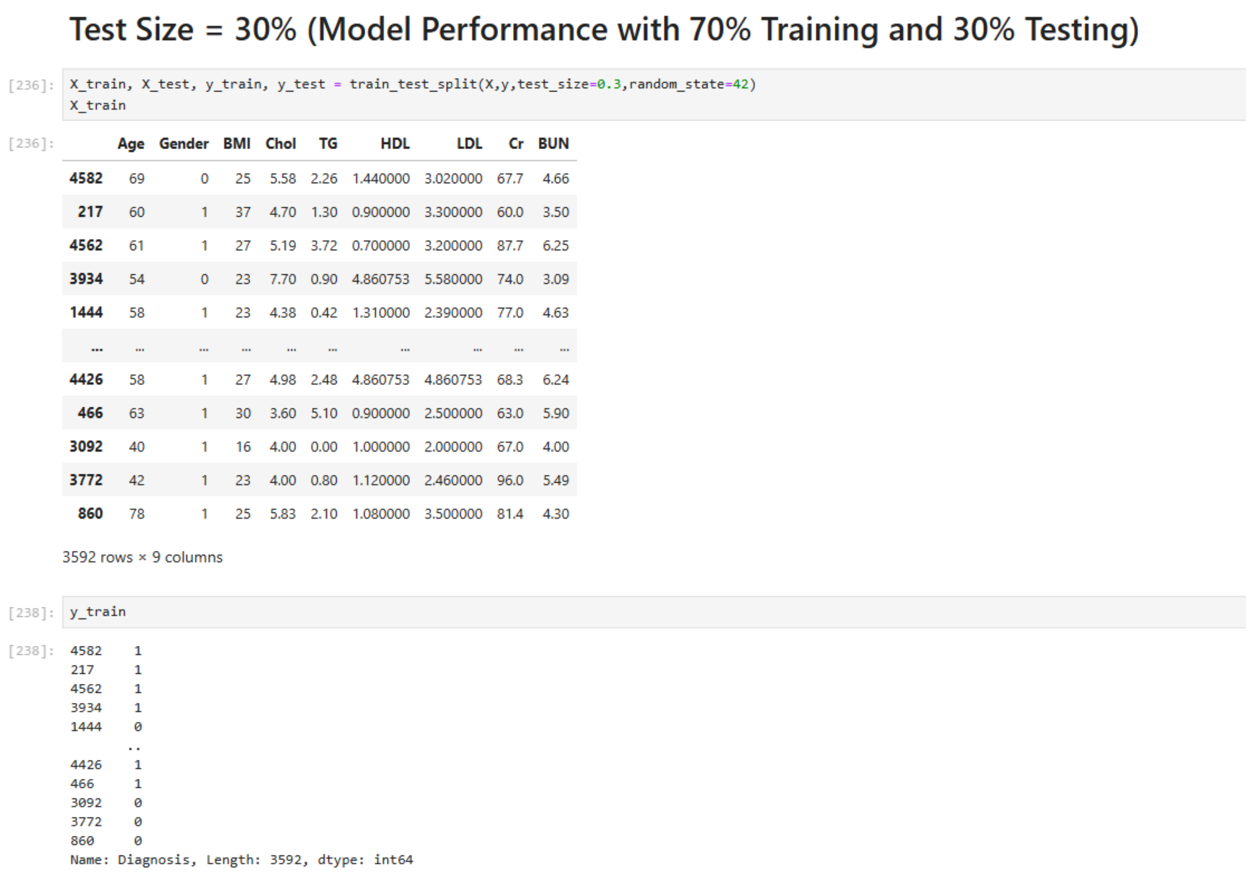
1027 rows × 9 columns

```
[66]: y_test
```

```
[66]: 5106    1
      2186    0
      2589    0
      831     0
      1421    0
             ..
      1662    0
      833     0
      366     1
      3778    0
      1235    0
      Name: Diagnosis, Length: 1027, dtype: int64
```

Figure 20: X and y Train 3 for 30%.

3.5.10 X and y train for Model Performance with 70% Training and 30% Testing



Test Size = 30% (Model Performance with 70% Training and 30% Testing)

```
[236]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
       X_train
```

[236]:

	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN
4582	69	0	25	5.58	2.26	1.440000	3.020000	67.7	4.66
217	60	1	37	4.70	1.30	0.900000	3.300000	60.0	3.50
4562	61	1	27	5.19	3.72	0.700000	3.200000	87.7	6.25
3934	54	0	23	7.70	0.90	4.860753	5.580000	74.0	3.09
1444	58	1	23	4.38	0.42	1.310000	2.390000	77.0	4.63
...	...	...	...	...	...	...	...	...	...
4426	58	1	27	4.98	2.48	4.860753	4.860753	68.3	6.24
466	63	1	30	3.60	5.10	0.900000	2.500000	63.0	5.90
3092	40	1	16	4.00	0.00	1.000000	2.000000	67.0	4.00
3772	42	1	23	4.00	0.80	1.120000	2.460000	96.0	5.49
860	78	1	25	5.83	2.10	1.080000	3.500000	81.4	4.30

3592 rows × 9 columns

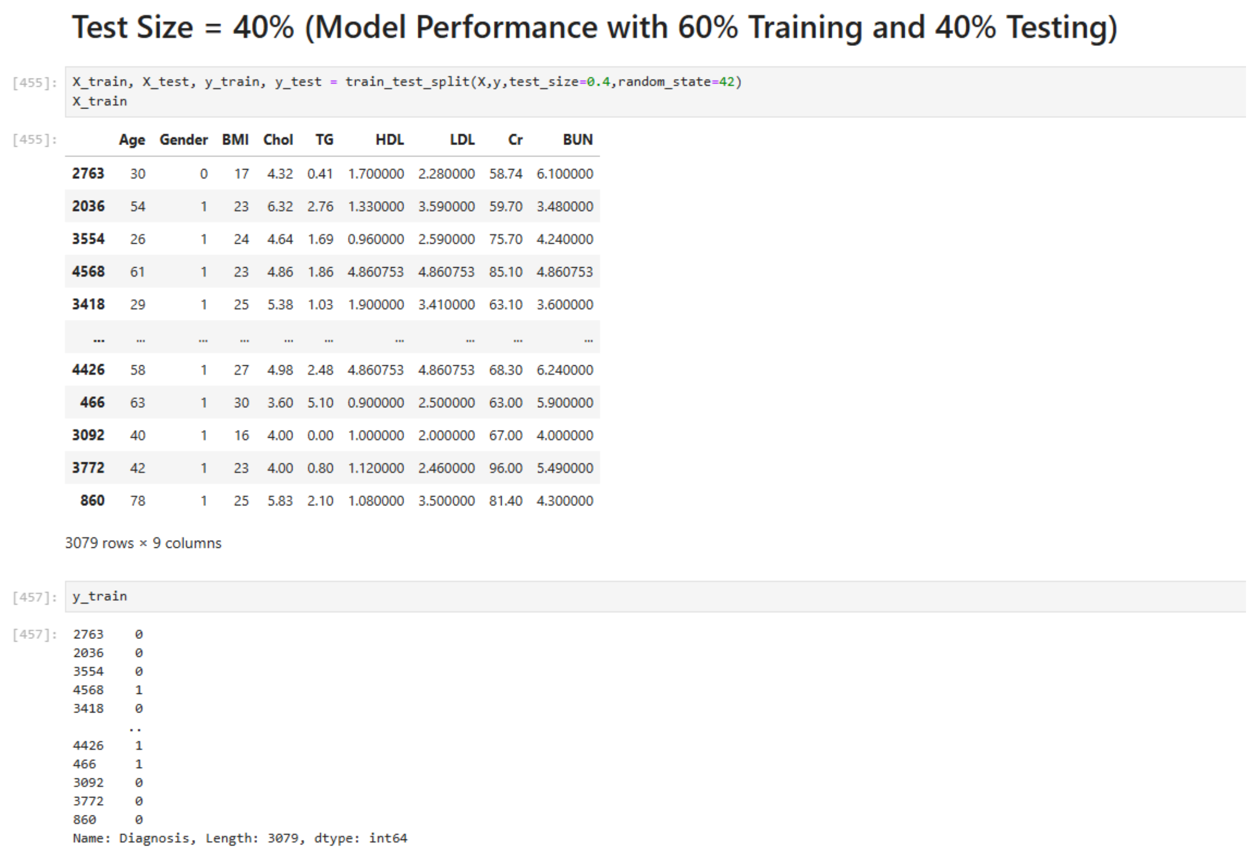
```
[238]: y_train
```

```
[238]: 4582    1
       217     1
       4562    1
       3934    1
       1444    0
              ..
       4426    1
       466     1
       3092    0
       3772    0
       860     0
       Name: Diagnosis, Length: 3592, dtype: int64
```

Figure 21: X and y Train for 30%.

This image displays the result of splitting the diabetes dataset into training (70%) and testing (30%) sets using the train_test_split ().

3.5.11 X and y train for Model Performance with 60% Training and 40% Testing



Test Size = 40% (Model Performance with 60% Training and 40% Testing)

```
[455]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.4,random_state=42)
       X_train
```

[455]:

	Age	Gender	BMI	Chol	TG	HDL	LDL	Cr	BUN
2763	30	0	17	4.32	0.41	1.700000	2.280000	58.74	6.100000
2036	54	1	23	6.32	2.76	1.330000	3.590000	59.70	3.480000
3554	26	1	24	4.64	1.69	0.960000	2.590000	75.70	4.240000
4568	61	1	23	4.86	1.86	4.860753	4.860753	85.10	4.860753
3418	29	1	25	5.38	1.03	1.900000	3.410000	63.10	3.600000
...	...	...	...	...	...	...	...	...	...
4426	58	1	27	4.98	2.48	4.860753	4.860753	68.30	6.240000
466	63	1	30	3.60	5.10	0.900000	2.500000	63.00	5.900000
3092	40	1	16	4.00	0.00	1.000000	2.000000	67.00	4.000000
3772	42	1	23	4.00	0.80	1.120000	2.460000	96.00	5.490000
860	78	1	25	5.83	2.10	1.080000	3.500000	81.40	4.300000

3079 rows × 9 columns

```
[457]: y_train
```

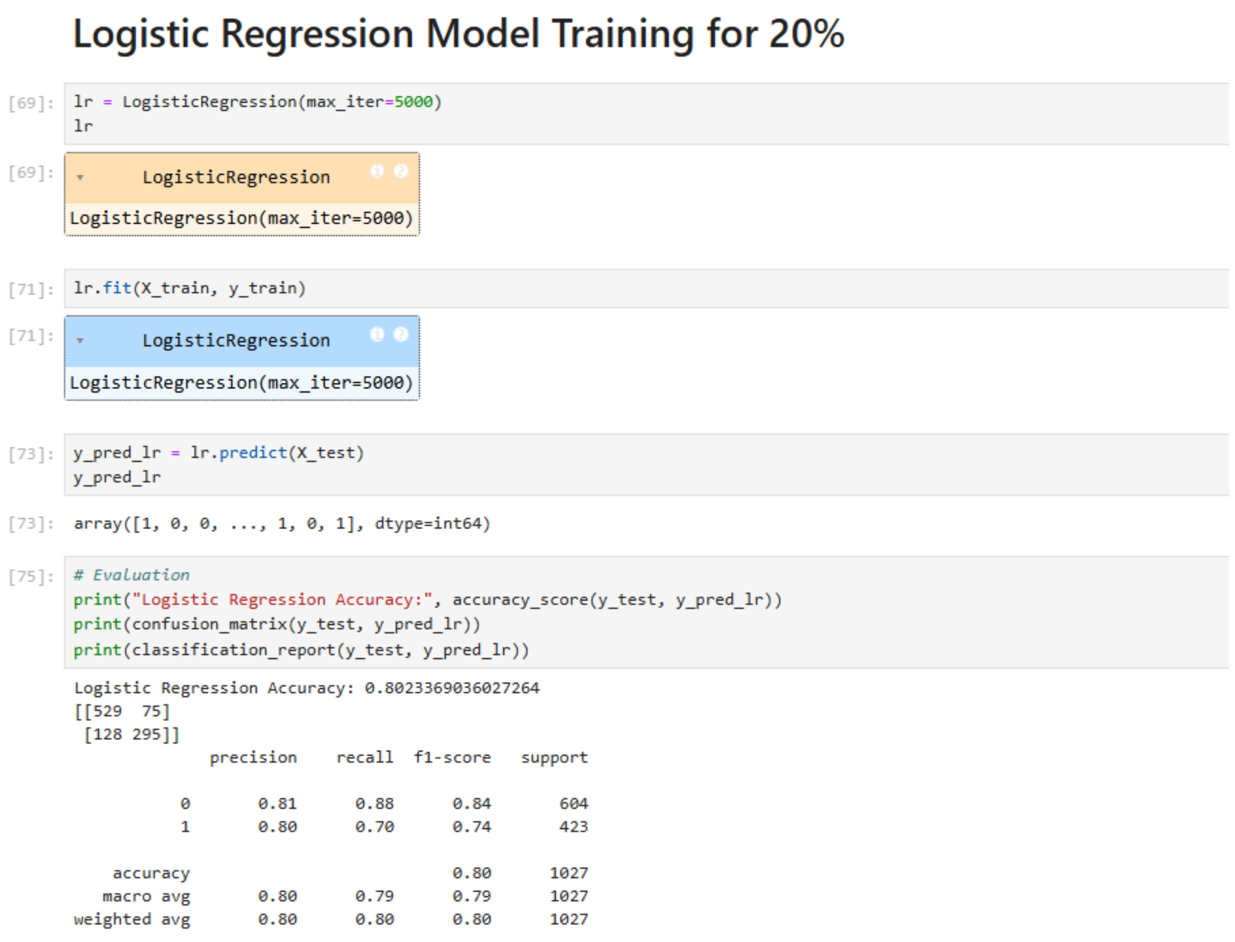
```
[457]: 2763    0
       2036    0
       3554    0
       4568    1
       3418    0
              ..
       4426    1
       466     1
       3092    0
       3772    0
       860     0
       Name: Diagnosis, Length: 3079, dtype: int64
```

Figure 22: X and y Train for 40%.

This image displays the result of splitting the diabetes dataset into training (60%) and testing (40%) sets using the train_test_split ().

3.6 Achieved Results

3.6.1 Logistic Regression Model Training and Accuracy for 20%



Logistic Regression Model Training for 20%

```
[69]: lr = LogisticRegression(max_iter=5000)
      lr
```

```
[69]: LogisticRegression
      LogisticRegression(max_iter=5000)
```

```
[71]: lr.fit(X_train, y_train)
```

```
[71]: LogisticRegression
      LogisticRegression(max_iter=5000)
```

```
[73]: y_pred_lr = lr.predict(X_test)
      y_pred_lr
```

```
[73]: array([1, 0, 0, ..., 1, 0, 1], dtype=int64)
```

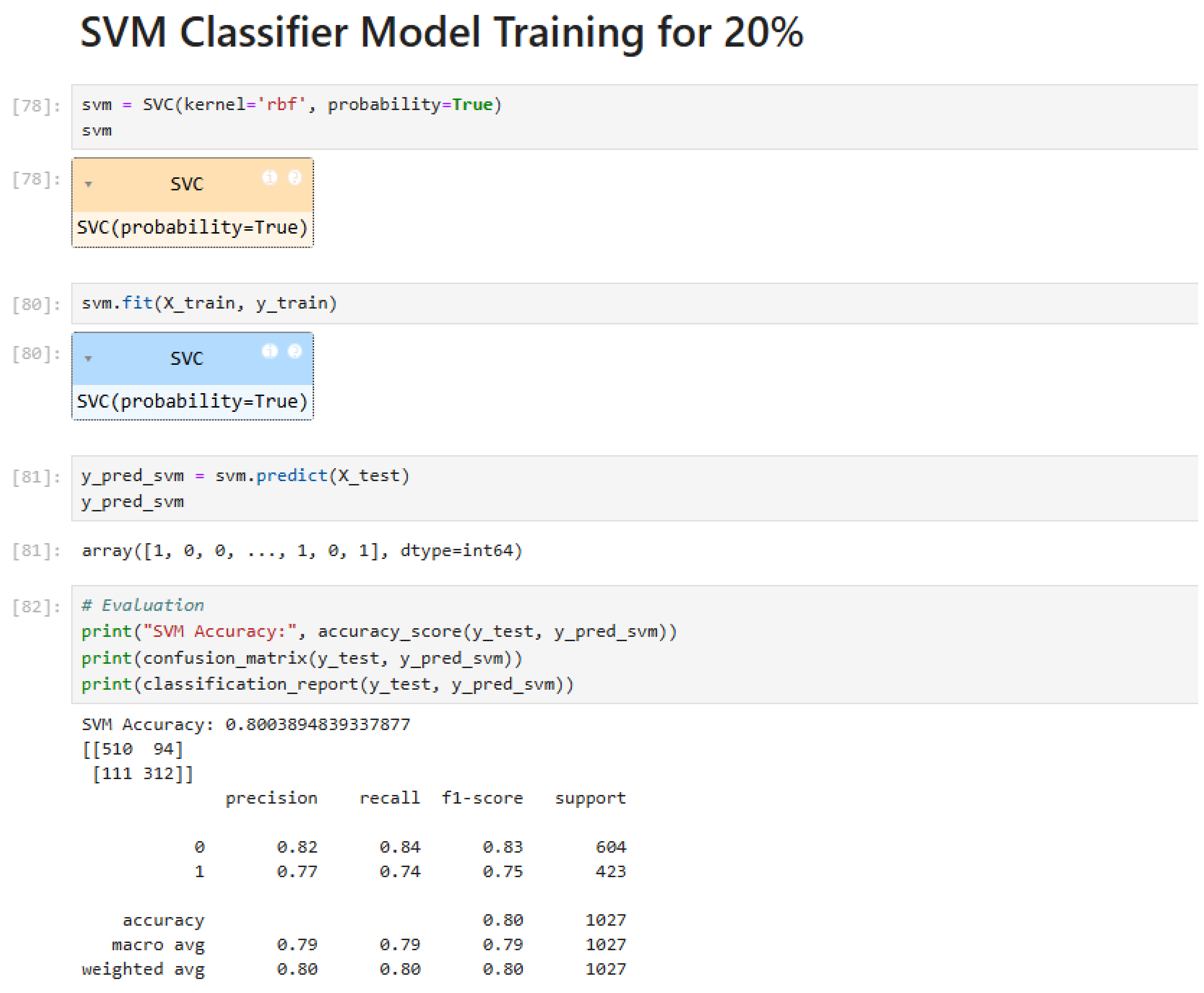
```
[75]: # Evaluation
      print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
      print(confusion_matrix(y_test, y_pred_lr))
      print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Accuracy: 0.8023369036027264
[[529  75]
 [128 295]]
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       604
           1       0.80      0.70      0.74       423

    accuracy                           0.80      1027
   macro avg       0.80      0.79      0.79      1027
weighted avg       0.80      0.80      0.80      1027
```

Figure 23: Logistic Regression Model Training for 20%.

The training process is done for the logistic regression for 80:20 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.2 SVM Classifier Model Training and Accuracy for 20%



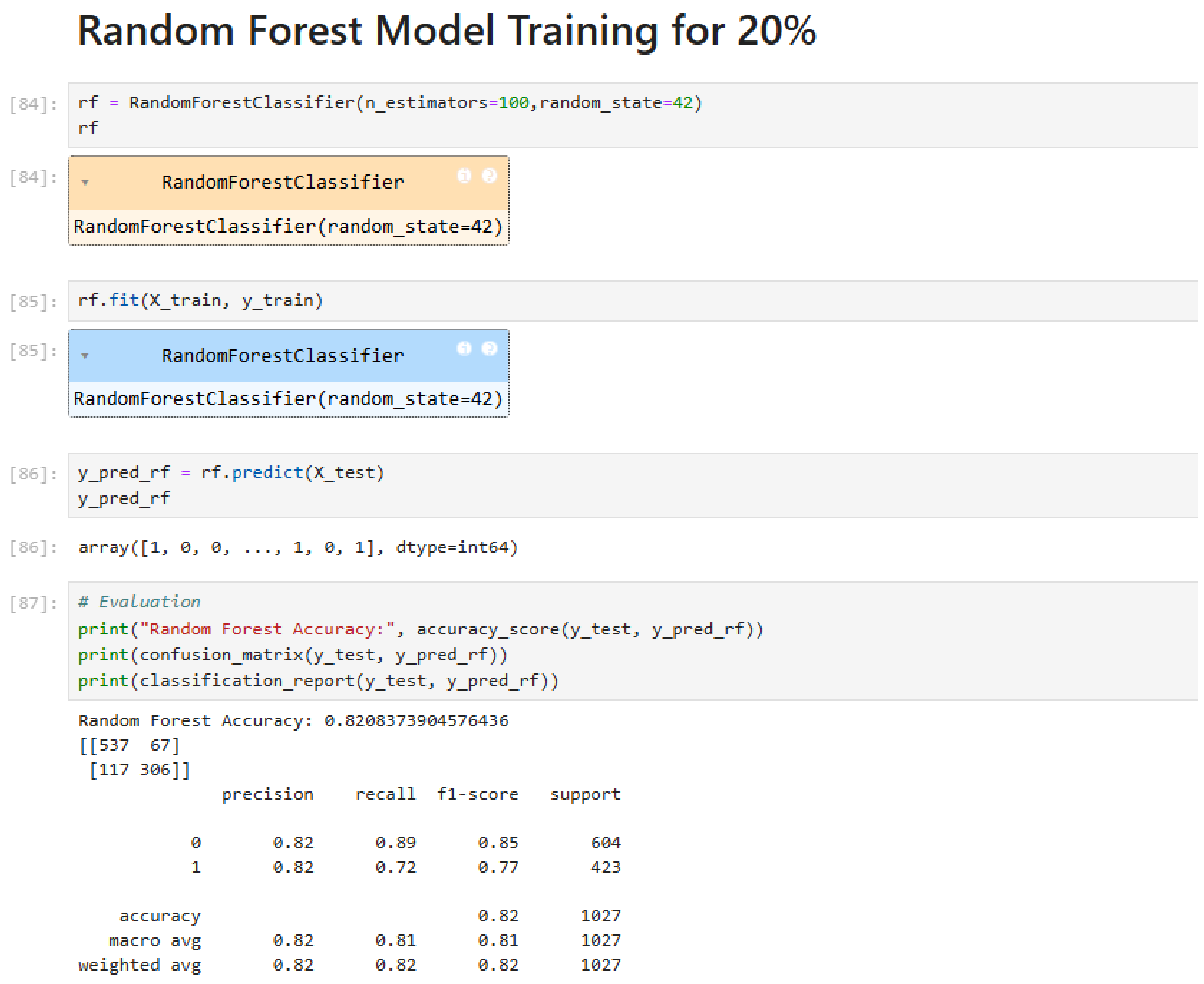
SVM Classifier Model Training for 20%

```
[78]: svm = SVC(kernel='rbf', probability=True)
      svm

[78]: SVC
      SVC(probability=True)

[80]: svm.fit(X_train, y_train)

[80]: SVC
      SVC(probability=True)

[81]: y_pred_svm = svm.predict(X_test)
      y_pred_svm

[81]: array([1, 0, 0, ..., 1, 0, 1], dtype=int64)

[82]: # Evaluation
      print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
      print(confusion_matrix(y_test, y_pred_svm))
      print(classification_report(y_test, y_pred_svm))

      SVM Accuracy: 0.8003894839337877
      [[510  94]
       [111 312]]
                    precision    recall  f1-score   support

                 0       0.82      0.84      0.83       604
                 1       0.77      0.74      0.75       423

          accuracy                           0.80      1027
         macro avg       0.79      0.79      0.79      1027
      weighted avg       0.80      0.80      0.80      1027
```

Figure 24: SVM Classifier Model Training for 20%.

The training process is done for the SVM classifier for 80:20 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.3 Random Forest Model Training and Accuracy for 20%



Random Forest Model Training for 20%

```
[84]: rf = RandomForestClassifier(n_estimators=100,random_state=42)
      rf
```

```
[84]: RandomForestClassifier
      RandomForestClassifier(random_state=42)
```

```
[85]: rf.fit(X_train, y_train)
```

```
[85]: RandomForestClassifier
      RandomForestClassifier(random_state=42)
```

```
[86]: y_pred_rf = rf.predict(X_test)
      y_pred_rf
```

```
[86]: array([1, 0, 0, ..., 1, 0, 1], dtype=int64)
```

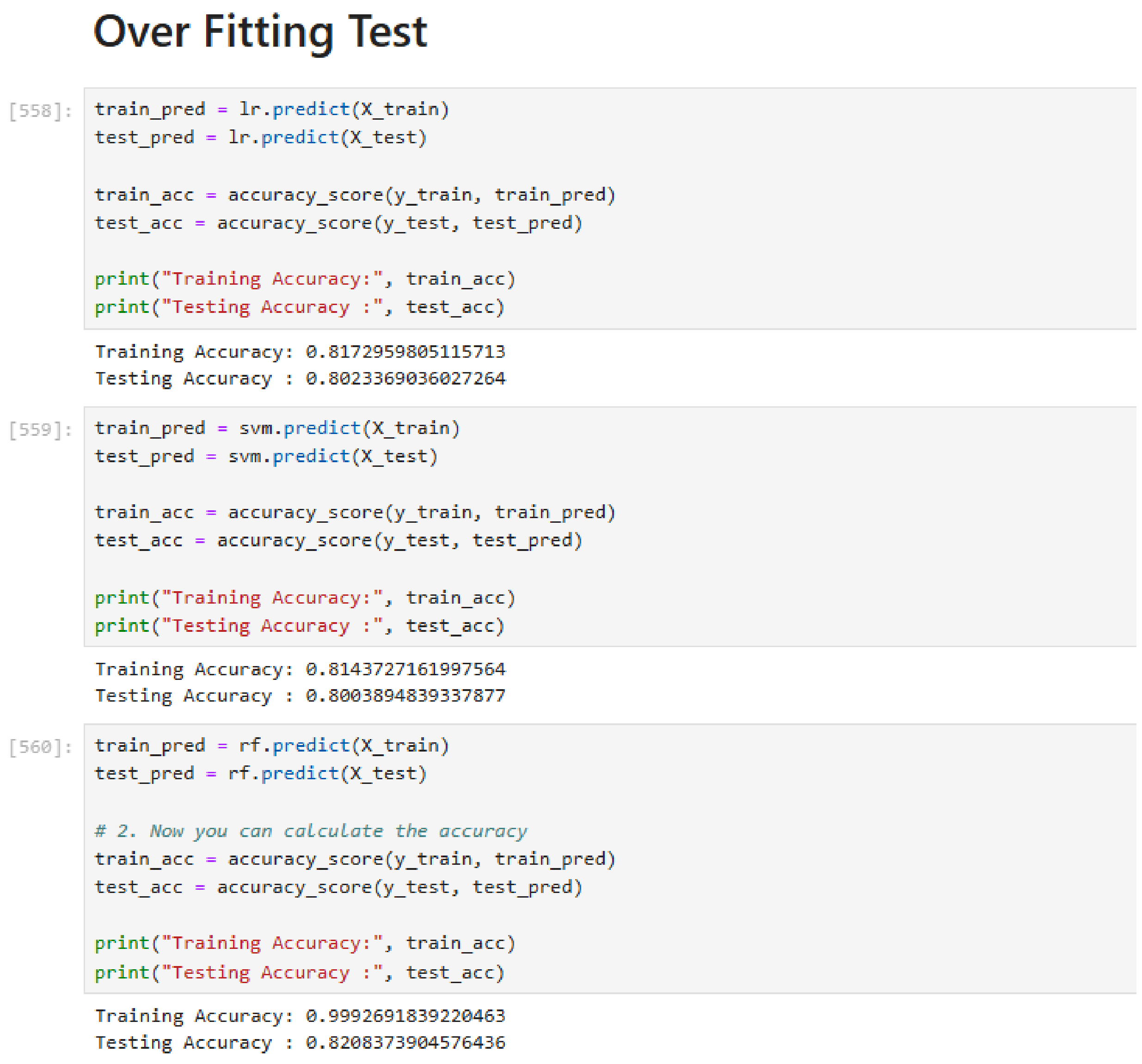
```
[87]: # Evaluation
      print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
      print(confusion_matrix(y_test, y_pred_rf))
      print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8208373904576436
[[537  67]
 [117 306]]
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       604
           1       0.82      0.72      0.77       423

    accuracy                           0.82      1027
   macro avg       0.82      0.81      0.81      1027
weighted avg       0.82      0.82      0.82      1027
```

Figure 25: Random Forest Model Training for 20%.

The training process is done for the SVM classifier for 80:20 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.4 Over Fitting Test



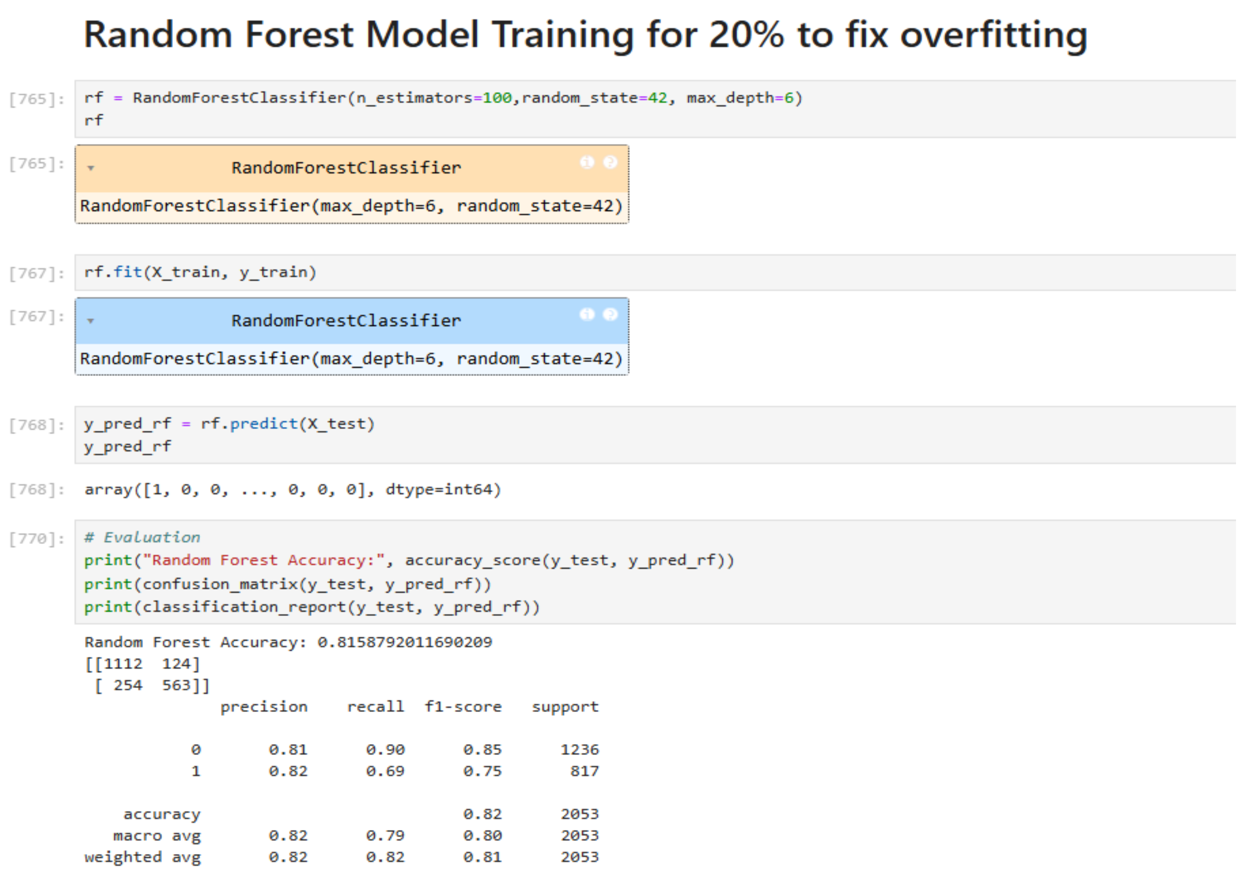
Over Fitting Test

```
[558]: train_pred = lr.predict(X_train)
       test_pred = lr.predict(X_test)

       train_acc = accuracy_score(y_train, train_pred)
       test_acc = accuracy_score(y_test, test_pred)

       print("Training Accuracy:", train_acc)
       print("Testing Accuracy :", test_acc)

       Training Accuracy: 0.8172959805115713
       Testing Accuracy : 0.8023369036027264

[559]: train_pred = svm.predict(X_train)
       test_pred = svm.predict(X_test)

       train_acc = accuracy_score(y_train, train_pred)
       test_acc = accuracy_score(y_test, test_pred)

       print("Training Accuracy:", train_acc)
       print("Testing Accuracy :", test_acc)

       Training Accuracy: 0.8143727161997564
       Testing Accuracy : 0.8003894839337877

[560]: train_pred = rf.predict(X_train)
       test_pred = rf.predict(X_test)

       # 2. Now you can calculate the accuracy
       train_acc = accuracy_score(y_train, train_pred)
       test_acc = accuracy_score(y_test, test_pred)

       print("Training Accuracy:", train_acc)
       print("Testing Accuracy :", test_acc)

       Training Accuracy: 0.9992691839220463
       Testing Accuracy : 0.8208373904576436
```

Figure 26: Overfitting Test for all the Used Algorithms.

This image displays if the used algorithms are overfit or not by comparing the training accuracy and the testing accuracy. From the above image the logistic regression and the SVM does not suffer from the overfitting and displays stable performance however, the random forest model displays very high training accuracy this shows that the random forest algorithm has suffered from the overfitting.

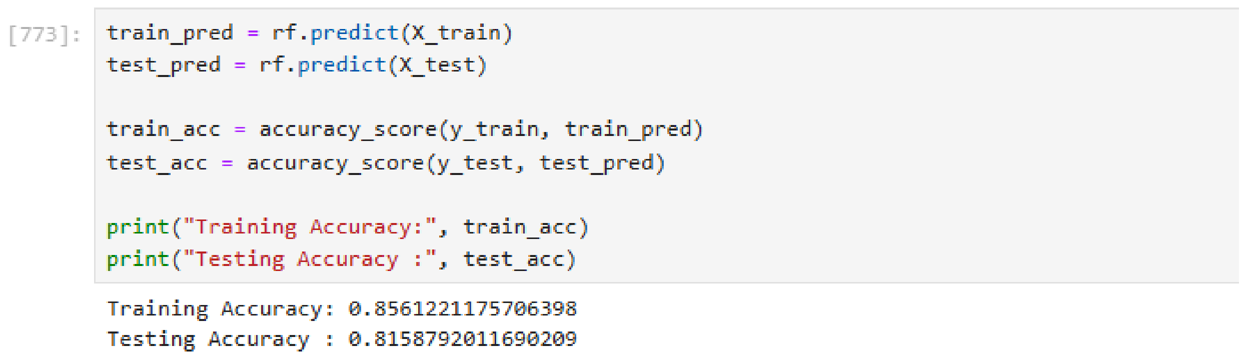
3.6.5 Random Forest Model Training and Accuracy for 20%



Random Forest Model Training for 20% to fix overfitting

```
[765]: rf = RandomForestClassifier(n_estimators=100,random_state=42, max_depth=6)
       rf

[765]: RandomForestClassifier
       RandomForestClassifier(max_depth=6, random_state=42)

[767]: rf.fit(X_train, y_train)

[767]: RandomForestClassifier
       RandomForestClassifier(max_depth=6, random_state=42)

[768]: y_pred_rf = rf.predict(X_test)
       y_pred_rf

[768]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)

[770]: # Evaluation
       print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
       print(confusion_matrix(y_test, y_pred_rf))
       print(classification_report(y_test, y_pred_rf))

       Random Forest Accuracy: 0.8158792011690209
       [[1112  124]
        [ 254  563]]
                     precision    recall  f1-score   support

                  0       0.81      0.90      0.85      1236
                  1       0.82      0.69      0.75       817

           accuracy                           0.82      2053
          macro avg       0.82      0.79      0.80      2053
       weighted avg       0.82      0.82      0.81      2053
```

Figure 27: Random Forest Model Training for 20% to Fix Overfitting.

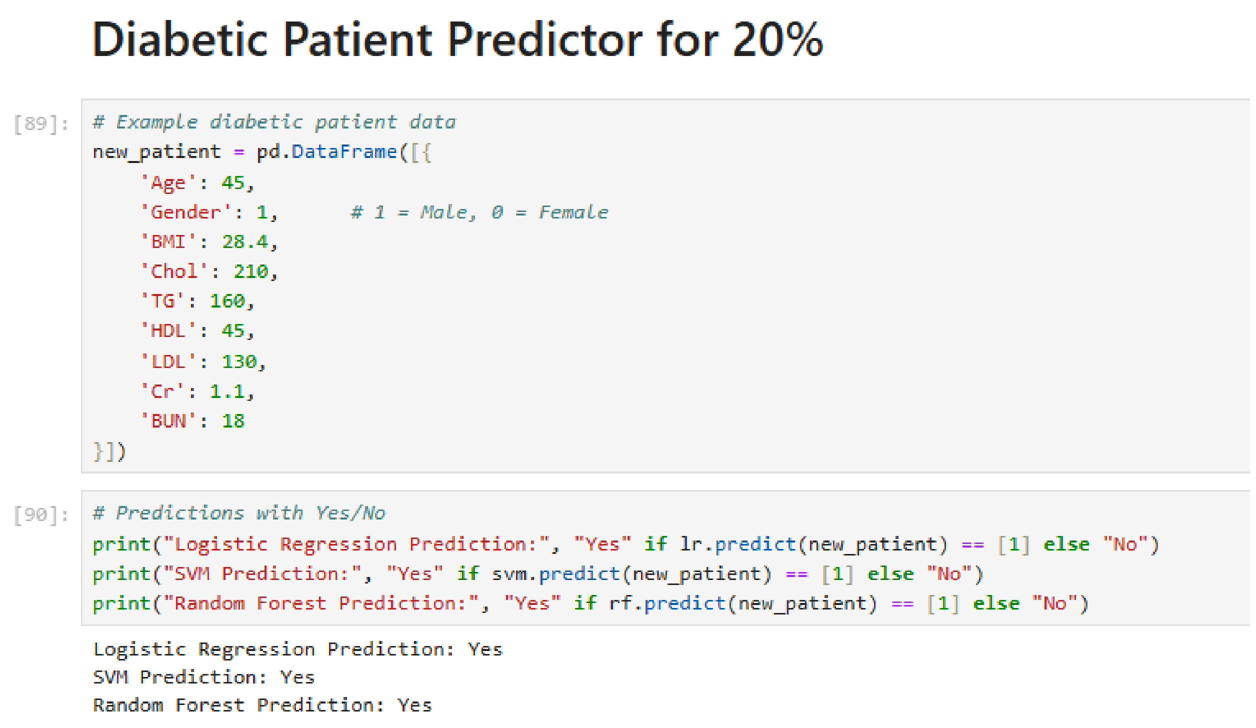


```
[773]: train_pred = rf.predict(X_train)
       test_pred = rf.predict(X_test)

       train_acc = accuracy_score(y_train, train_pred)
       test_acc = accuracy_score(y_test, test_pred)

       print("Training Accuracy:", train_acc)
       print("Testing Accuracy :", test_acc)

       Training Accuracy: 0.8561221175706398
       Testing Accuracy : 0.8158792011690209
```

Figure 28: Random Forest Overfitting Test.

After applying the max_ depth regularization the random forest model achieved balanced performance and no longer suffers from the overfitting.

3.6.6 Diabetic Patient Predictor for 20%



Diabetic Patient Predictor for 20%

```
[89]: # Example diabetic patient data
new_patient = pd.DataFrame([{
    'Age': 45,
    'Gender': 1,      # 1 = Male, 0 = Female
    'BMI': 28.4,
    'Chol': 210,
    'TG': 160,
    'HDL': 45,
    'LDL': 130,
    'Cr': 1.1,
    'BUN': 18
}])
```

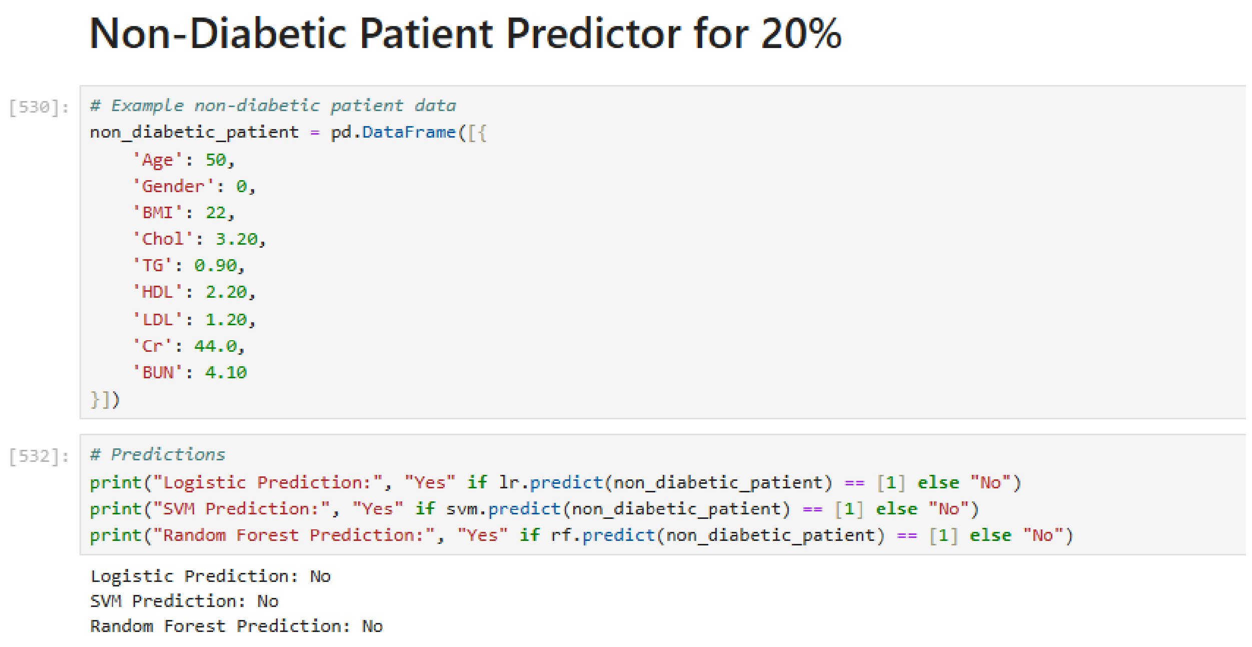
```
[90]: # Predictions with Yes/No
print("Logistic Regression Prediction:", "Yes" if lr.predict(new_patient) == [1] else "No")
print("SVM Prediction:", "Yes" if svm.predict(new_patient) == [1] else "No")
print("Random Forest Prediction:", "Yes" if rf.predict(new_patient) == [1] else "No")
```

```
Logistic Regression Prediction: Yes
SVM Prediction: Yes
Random Forest Prediction: Yes
```

Figure 29: Diabetic Patient Predictor for 20%.

A sample of diabetic dataset is created using random patient medical report. The data is then passed through all the trained models (Logistic, SVM, Random Forest). Each model evaluates the input features and predicted data is then displayed.

3.6.7 Non-Diabetic Patient Predictor for 20%



Non-Diabetic Patient Predictor for 20%

```
[530]: # Example non-diabetic patient data
       non_diabetic_patient = pd.DataFrame([{
           'Age': 50,
           'Gender': 0,
           'BMI': 22,
           'Chol': 3.20,
           'TG': 0.90,
           'HDL': 2.20,
           'LDL': 1.20,
           'Cr': 44.0,
           'BUN': 4.10
       }])
```

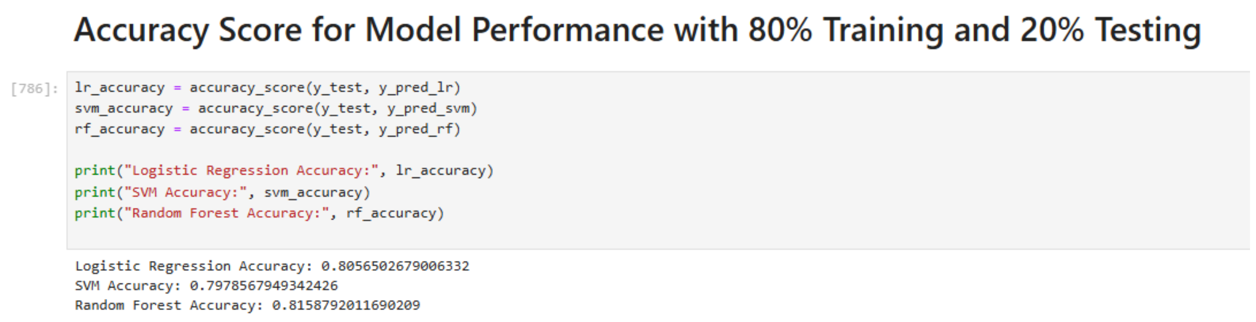
```
[532]: # Predictions
       print("Logistic Prediction:", "Yes" if lr.predict(non_diabetic_patient) == [1] else "No")
       print("SVM Prediction:", "Yes" if svm.predict(non_diabetic_patient) == [1] else "No")
       print("Random Forest Prediction:", "Yes" if rf.predict(non_diabetic_patient) == [1] else "No")
```

```
Logistic Prediction: No
SVM Prediction: No
Random Forest Prediction: No
```

Figure 30: Non-diabetic Patient Predictor for 30%.

A sample of non-diabetic dataset is created using random patient medical report. The data is then passed through all the trained models (Logistic, SVM, Random Forest). Each model evaluates the input features and predicted data is then displayed.

3.6.8 Accuracy Score for 20%



Accuracy Score for Model Performance with 80% Training and 20% Testing

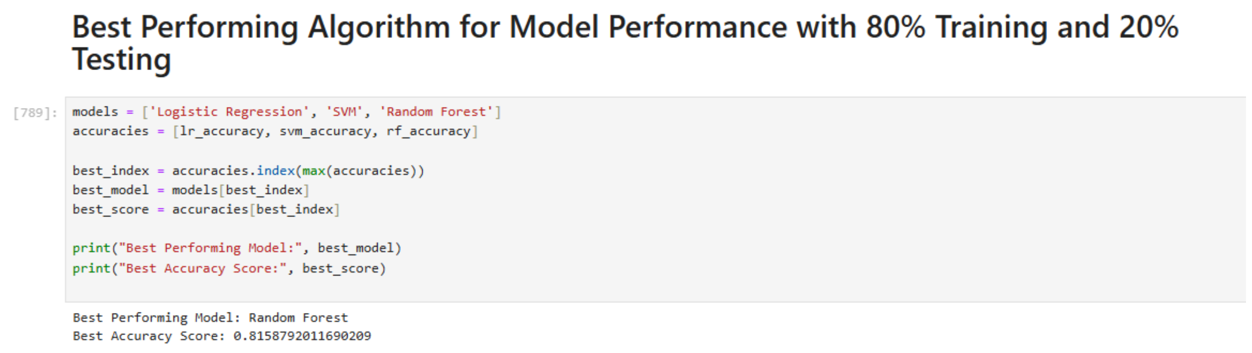
```
[786]: lr_accuracy = accuracy_score(y_test, y_pred_lr)
svm_accuracy = accuracy_score(y_test, y_pred_svm)
rf_accuracy = accuracy_score(y_test, y_pred_rf)

print("Logistic Regression Accuracy:", lr_accuracy)
print("SVM Accuracy:", svm_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Logistic Regression Accuracy: 0.8056502679006332
SVM Accuracy: 0.7978567949342426
Random Forest Accuracy: 0.8158792011690209
```

Figure 31: Accuracy Score Results for 20%.

The accuracy score displays the accuracy of all the algorithms for 80:20 and represents the percentage of the accuracy made by all the used algorithms which helps in identifying the best algorithm for the prediction.

3.6.9 Best Performing Algorithm for 20%



Best Performing Algorithm for Model Performance with 80% Training and 20% Testing

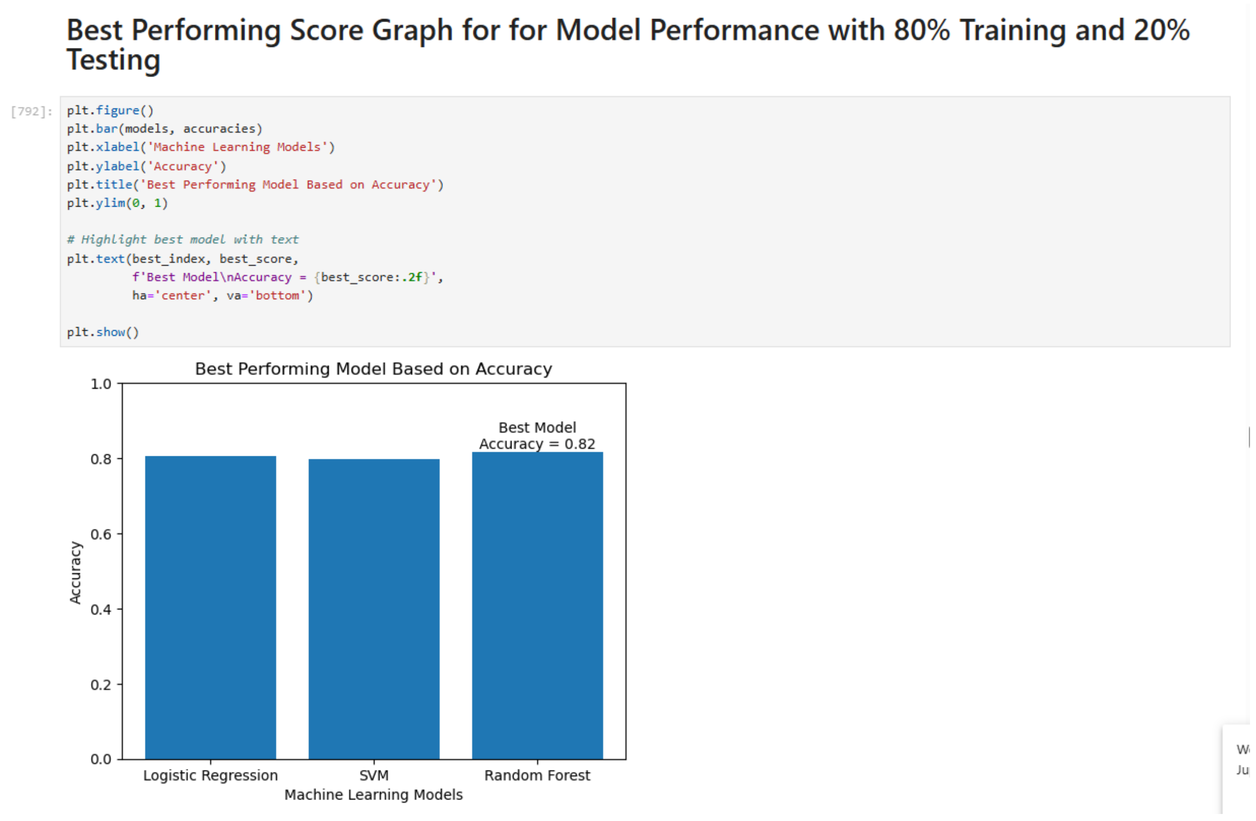
```
[789]: models = ['Logistic Regression', 'SVM', 'Random Forest']
accuracies = [lr_accuracy, svm_accuracy, rf_accuracy]

best_index = accuracies.index(max(accuracies))
best_model = models[best_index]
best_score = accuracies[best_index]

print("Best Performing Model:", best_model)
print("Best Accuracy Score:", best_score)
```

```
Best Performing Model: Random Forest
Best Accuracy Score: 0.8158792011690209
```

Figure 32: Best Performing Algorithm for 20%.

This image displays the best algorithm for this diabetes prediction model using the accuracy score.

3.6.10 Best Performing Algorithm Bar Graph for 20%



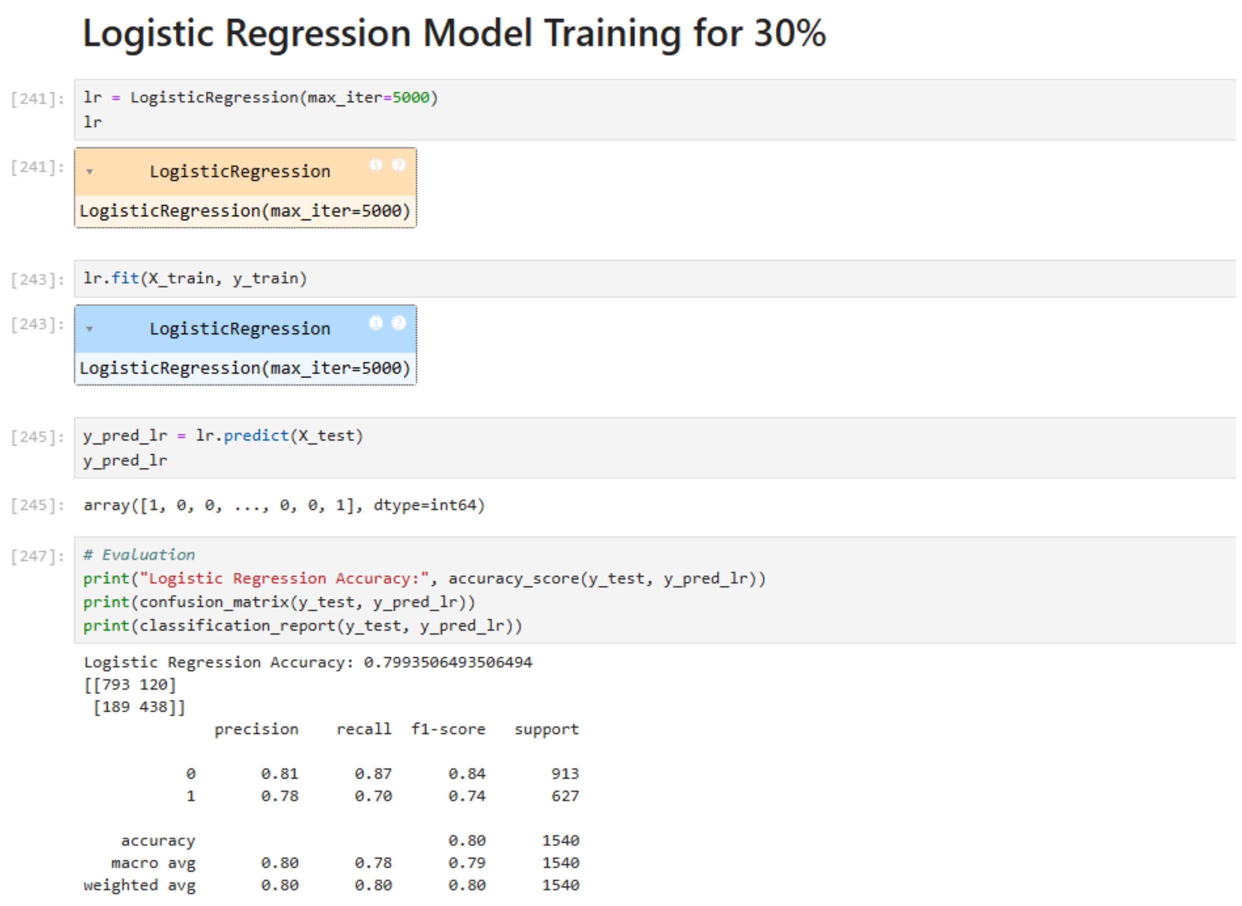
Best Performing Score Graph for for Model Performance with 80% Training and 20% Testing

```
[792]: plt.figure()
plt.bar(models, accuracies)
plt.xlabel('Machine Learning Models')
plt.ylabel('Accuracy')
plt.title('Best Performing Model Based on Accuracy')
plt.ylim(0, 1)

# Highlight best model with text
plt.text(best_index, best_score,
         f'Best Model\nAccuracy = {best_score:.2f}',
         ha='center', va='bottom')

plt.show()
```

Figure 33: Best Performing Algorithm Bar Graph for 20%.

The image displays the bar graph of the best performing algorithm.

3.6.11 Logistic Regression Model Training and Accuracy for 30%



Logistic Regression Model Training for 30%

```
[241]: lr = LogisticRegression(max_iter=5000)
       lr
```

```
[241]: LogisticRegression
       LogisticRegression(max_iter=5000)
```

```
[243]: lr.fit(X_train, y_train)
```

```
[243]: LogisticRegression
       LogisticRegression(max_iter=5000)
```

```
[245]: y_pred_lr = lr.predict(X_test)
       y_pred_lr
```

```
[245]: array([1, 0, 0, ..., 0, 0, 1], dtype=int64)
```

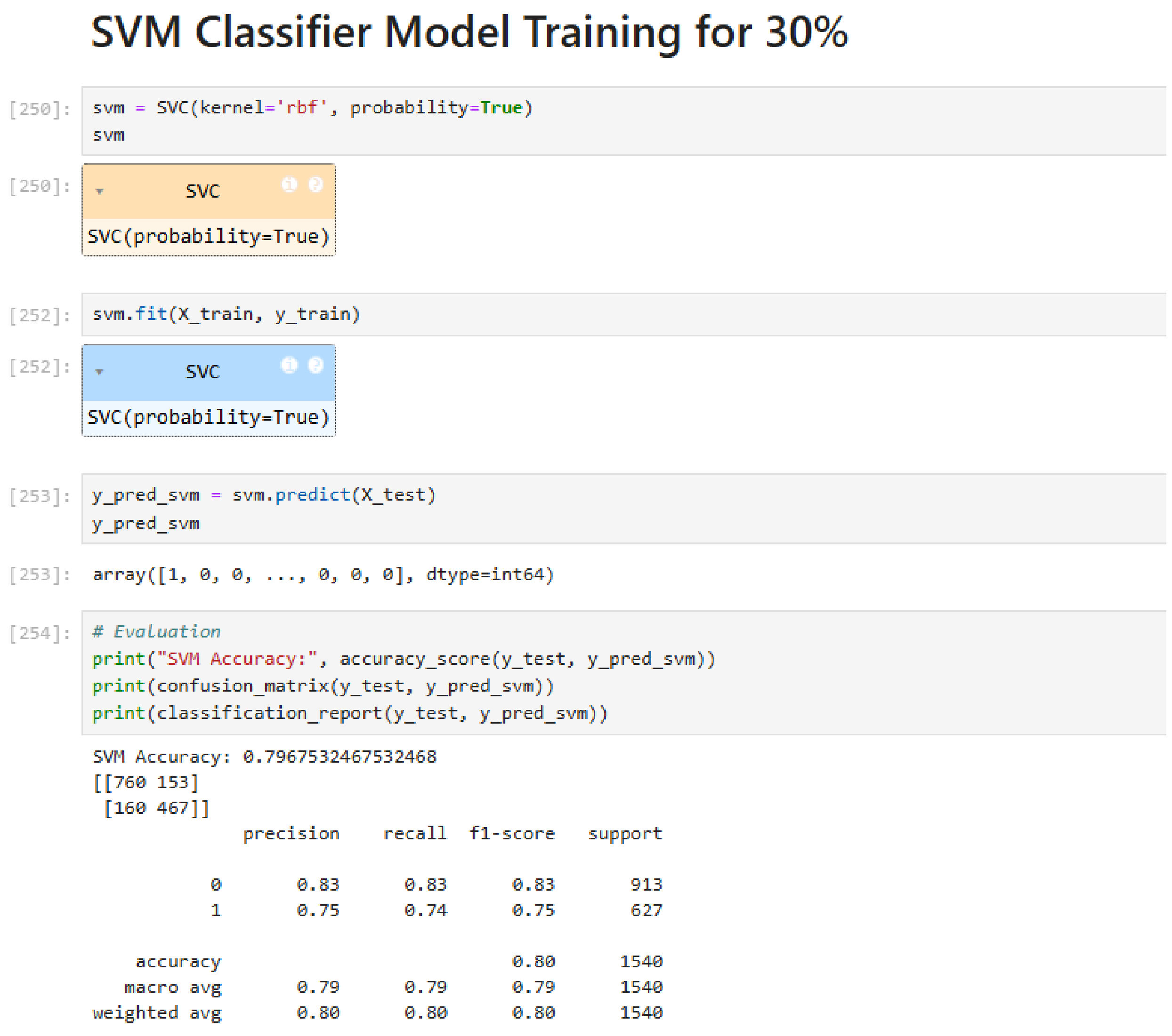
```
[247]: # Evaluation
       print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
       print(confusion_matrix(y_test, y_pred_lr))
       print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Accuracy: 0.7993506493506494
[[793 120]
 [189 438]]
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       913
           1       0.78      0.70      0.74       627

    accuracy                           0.80      1540
   macro avg       0.80      0.78      0.79      1540
weighted avg       0.80      0.80      0.80      1540
```

Figure 34: Logistic Regression Model Training for 30%.

The training process is done for the logistic regression for 70:30 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.12 SVM Classifier Model Training and Accuracy for 30%



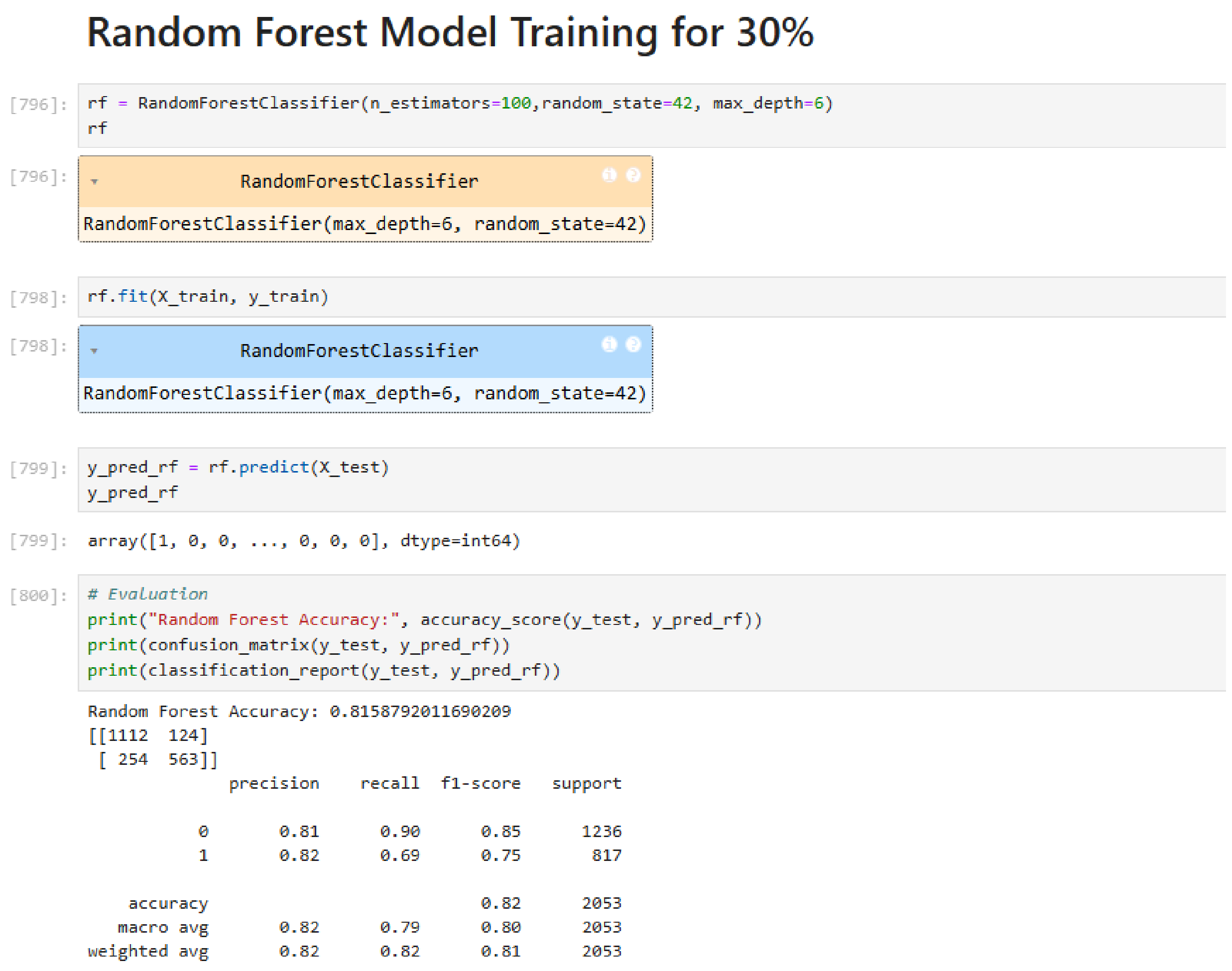
SVM Classifier Model Training for 30%

```
[250]: svm = SVC(kernel='rbf', probability=True)
       svm

[250]: SVC
       SVC(probability=True)

[252]: svm.fit(X_train, y_train)

[252]: SVC
       SVC(probability=True)

[253]: y_pred_svm = svm.predict(X_test)
       y_pred_svm

[253]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)

[254]: # Evaluation
       print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
       print(confusion_matrix(y_test, y_pred_svm))
       print(classification_report(y_test, y_pred_svm))

       SVM Accuracy: 0.7967532467532468
       [[760 153]
        [160 467]]
                     precision    recall  f1-score   support

                  0       0.83      0.83      0.83       913
                  1       0.75      0.74      0.75       627

           accuracy                           0.80      1540
          macro avg       0.79      0.79      0.79      1540
       weighted avg       0.80      0.80      0.80      1540
```

Figure 35: SVM Classifier Model Training for 30%.

The training process is done for the logistic regression for 70:30 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.13 Random Forest Model Training and Accuracy for 30%



Random Forest Model Training for 30%

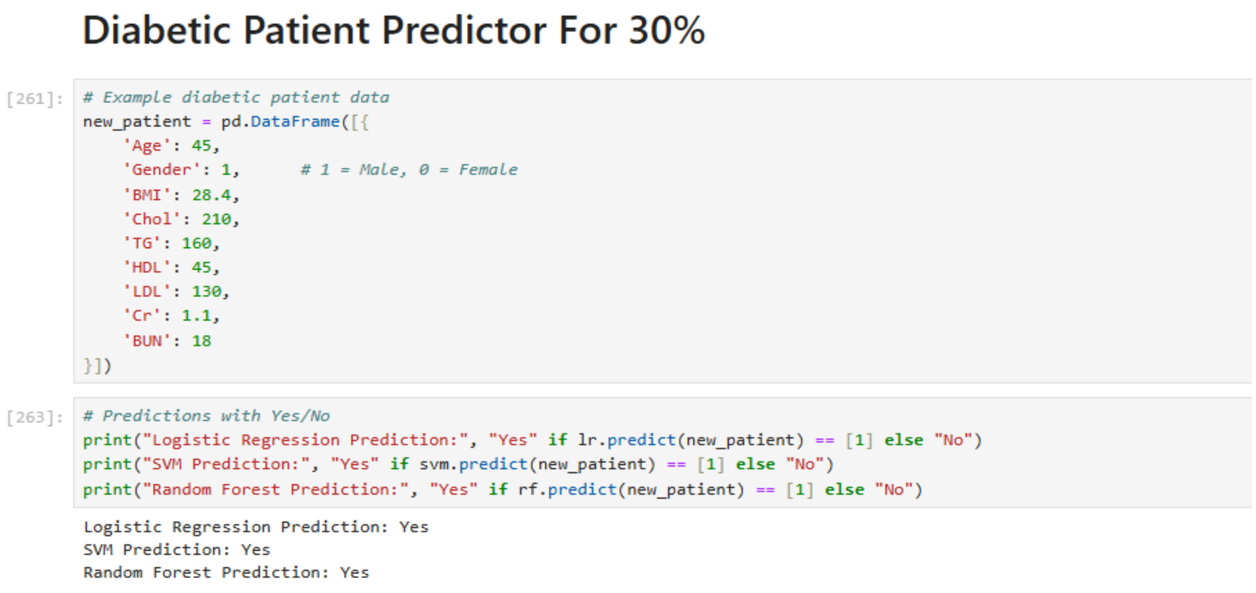
```
[796]: rf = RandomForestClassifier(n_estimators=100,random_state=42, max_depth=6)
       rf

[796]: RandomForestClassifier
       RandomForestClassifier(max_depth=6, random_state=42)

[798]: rf.fit(X_train, y_train)

[798]: RandomForestClassifier
       RandomForestClassifier(max_depth=6, random_state=42)

[799]: y_pred_rf = rf.predict(X_test)
       y_pred_rf

[799]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)

[800]: # Evaluation
       print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
       print(confusion_matrix(y_test, y_pred_rf))
       print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8158792011690209
[[1112  124]
 [ 254  563]]
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      1236
           1       0.82      0.69      0.75       817

    accuracy                           0.82      2053
   macro avg       0.82      0.79      0.80      2053
weighted avg       0.82      0.82      0.81      2053
```

Figure 36: Random Forest Model Training for 30%.

The training process is done for the logistic regression for 70:30 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.14 Diabetic Patient Predictor for 30%



Diabetic Patient Predictor For 30%

```
[261]: # Example diabetic patient data
new_patient = pd.DataFrame([{
    'Age': 45,
    'Gender': 1,       # 1 = Male, 0 = Female
    'BMI': 28.4,
    'Chol': 210,
    'TG': 160,
    'HDL': 45,
    'LDL': 130,
    'Cr': 1.1,
    'BUN': 18
}])
```

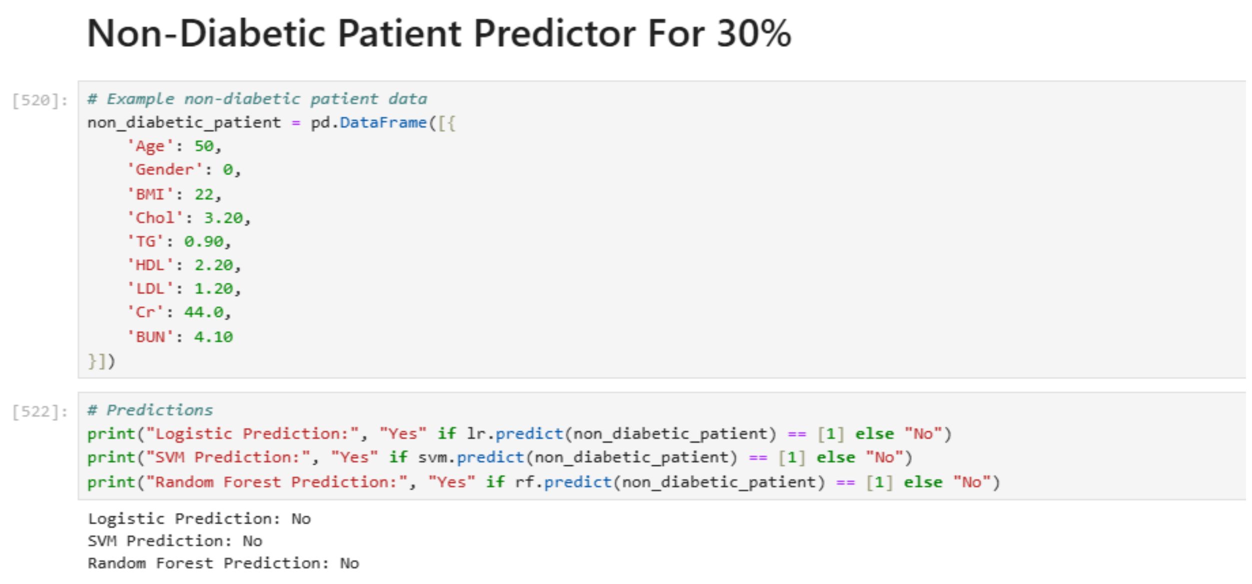
```
[263]: # Predictions with Yes/No
print("Logistic Regression Prediction:", "Yes" if lr.predict(new_patient) == [1] else "No")
print("SVM Prediction:", "Yes" if svm.predict(new_patient) == [1] else "No")
print("Random Forest Prediction:", "Yes" if rf.predict(new_patient) == [1] else "No")
```

```
Logistic Regression Prediction: Yes
SVM Prediction: Yes
Random Forest Prediction: Yes
```

Figure 37:Diabetic Patient Predictor for 30%.

A sample of diabetic dataset is created using random patient medical report. The data is then passed through all the trained models (Logistic, SVM, Random Forest). Each model evaluates the input features and predicted data is then displayed.

3.6.15 Non-diabetic Patient Predictor for 30%



Non-Diabetic Patient Predictor For 30%

```
[520]: # Example non-diabetic patient data
non_diabetic_patient = pd.DataFrame([{
    'Age': 50,
    'Gender': 0,
    'BMI': 22,
    'Chol': 3.20,
    'TG': 0.90,
    'HDL': 2.20,
    'LDL': 1.20,
    'Cr': 44.0,
    'BUN': 4.10
}])
```

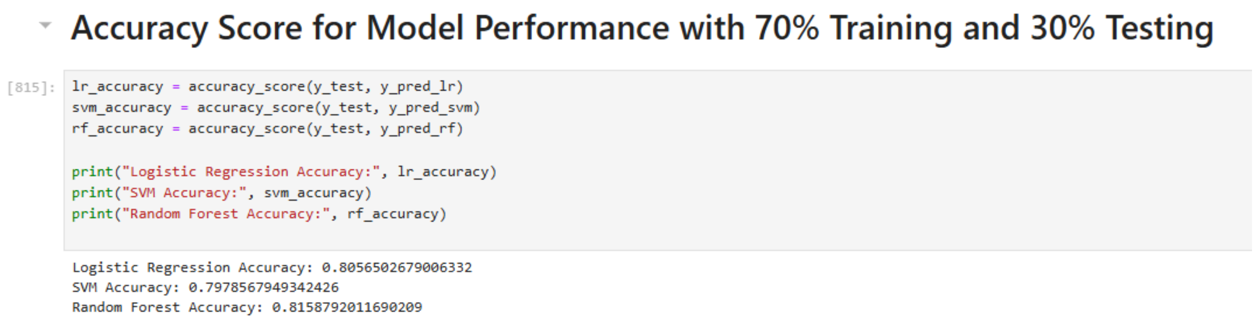
```
[522]: # Predictions
print("Logistic Prediction:", "Yes" if lr.predict(non_diabetic_patient) == [1] else "No")
print("SVM Prediction:", "Yes" if svm.predict(non_diabetic_patient) == [1] else "No")
print("Random Forest Prediction:", "Yes" if rf.predict(non_diabetic_patient) == [1] else "No")
```

```
Logistic Prediction: No
SVM Prediction: No
Random Forest Prediction: No
```

Figure 38:Non-diabetic Patient Predictor for 30%.

A sample of non-diabetic dataset is created using random patient medical report. The data is then passed through all the trained models (Logistic, SVM, Random Forest). Each model evaluates the input features and predicted data is then displayed.

3.6.16 Accuracy Score Model for 30%


Accuracy Score for Model Performance with 70% Training and 30% Testing

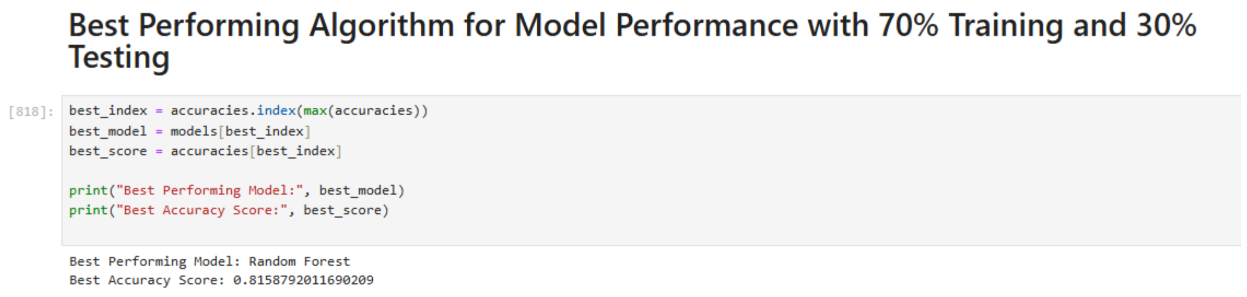
```
[815]: lr_accuracy = accuracy_score(y_test, y_pred_lr)
svm_accuracy = accuracy_score(y_test, y_pred_svm)
rf_accuracy = accuracy_score(y_test, y_pred_rf)

print("Logistic Regression Accuracy:", lr_accuracy)
print("SVM Accuracy:", svm_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Logistic Regression Accuracy: 0.8056502679006332
SVM Accuracy: 0.7978567949342426
Random Forest Accuracy: 0.8158792011690209
```

Figure 39: Accuracy Score Results for 30%.

The accuracy score displays the accuracy of all the algorithms for 70:30 and represents the percentage of the accuracy made by all the used algorithms which helps in identifying the best algorithm for the prediction.

3.6.17 Best Performing Algorithm for 30%


Best Performing Algorithm for Model Performance with 70% Training and 30% Testing

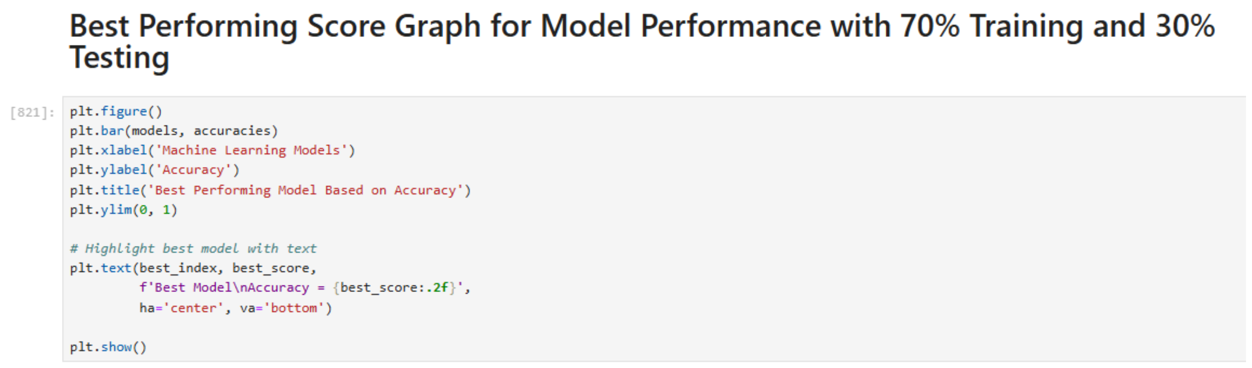
```
[818]: best_index = accuracies.index(max(accuracies))
best_model = models[best_index]
best_score = accuracies[best_index]

print("Best Performing Model:", best_model)
print("Best Accuracy Score:", best_score)
```

```
Best Performing Model: Random Forest
Best Accuracy Score: 0.8158792011690209
```

Figure 40: Best Performing Algorithm for 30%.

This image displays the best algorithm for this diabetes prediction model using the accuracy score.

3.6.18 Best Performing Score Graph for 30%



Best Performing Score Graph for Model Performance with 70% Training and 30% Testing

```
[821]: plt.figure()
plt.bar(models, accuracies)
plt.xlabel('Machine Learning Models')
plt.ylabel('Accuracy')
plt.title('Best Performing Model Based on Accuracy')
plt.ylim(0, 1)

# Highlight best model with text
plt.text(best_index, best_score,
         f'Best Model\nAccuracy = {best_score:.2f}',
         ha='center', va='bottom')

plt.show()
```

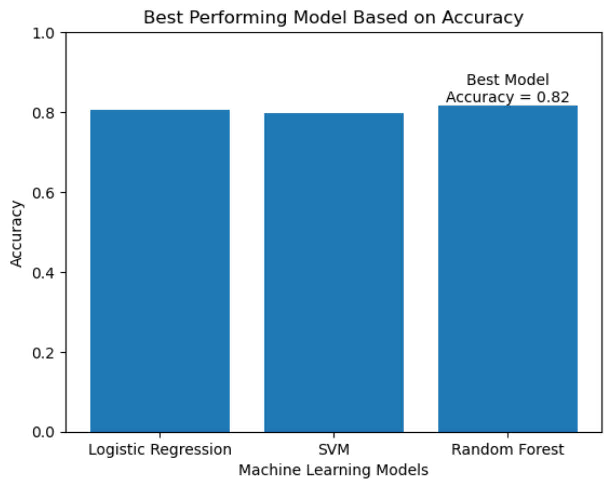
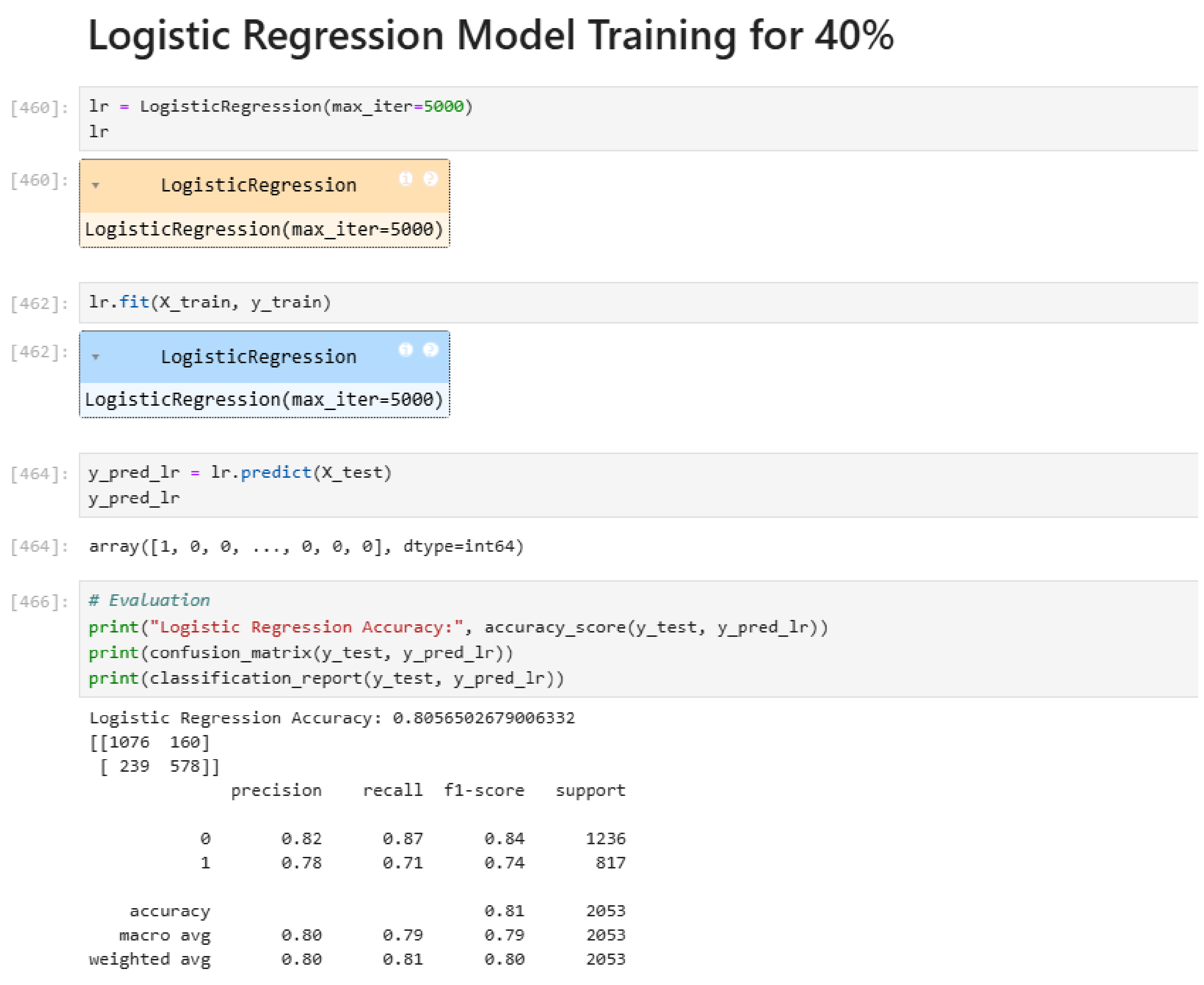


Figure 41: Best Performing Score Graph for 30%.

The image displays the bar graph of the best performing algorithm.

3.6.19 Logistic Regression Model Training and Accuracy for 40%



Logistic Regression Model Training for 40%

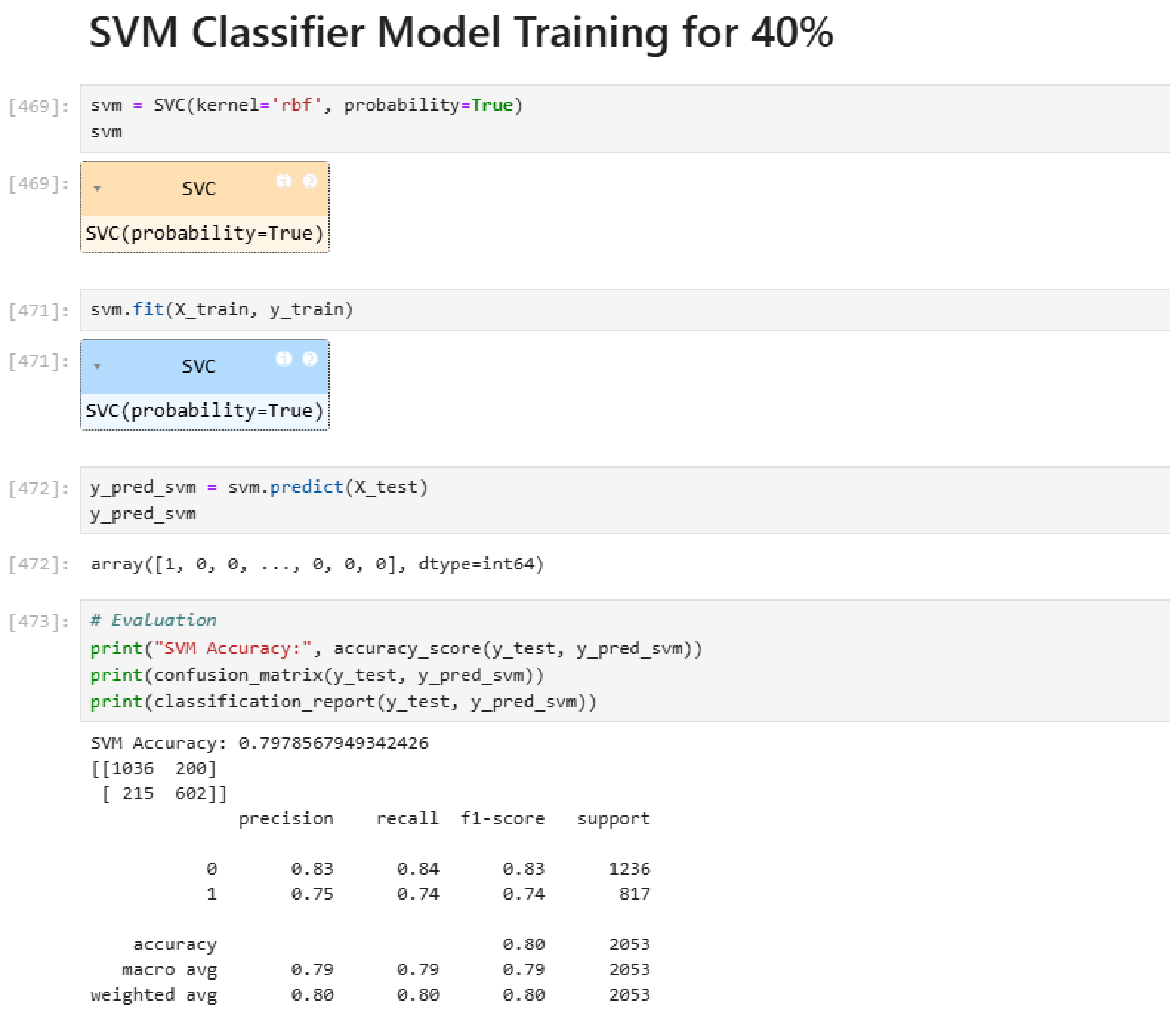
```
[460]: lr = LogisticRegression(max_iter=5000)
       lr
[460]: LogisticRegression
       LogisticRegression(max_iter=5000)

[462]: lr.fit(X_train, y_train)
[462]: LogisticRegression
       LogisticRegression(max_iter=5000)

[464]: y_pred_lr = lr.predict(X_test)
       y_pred_lr
[464]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)

[466]: # Evaluation
       print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
       print(confusion_matrix(y_test, y_pred_lr))
       print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Accuracy: 0.8056502679006332
[[1076  160]
 [ 239  578]]
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      1236
           1       0.78      0.71      0.74       817

    accuracy                           0.81      2053
   macro avg       0.80      0.79      0.79      2053
weighted avg       0.80      0.81      0.80      2053
```

Figure 42: Logistic Regression Model Training for 40%.

The training process is done for the logistic regression for 60:40 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.20 SVM Classifier Model and Accuracy for 40%



SVM Classifier Model Training for 40%

```
[469]: svm = SVC(kernel='rbf', probability=True)
       svm
```

```
[469]: SVC
       SVC(probability=True)
```

```
[471]: svm.fit(X_train, y_train)
```

```
[471]: SVC
       SVC(probability=True)
```

```
[472]: y_pred_svm = svm.predict(X_test)
       y_pred_svm
```

```
[472]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

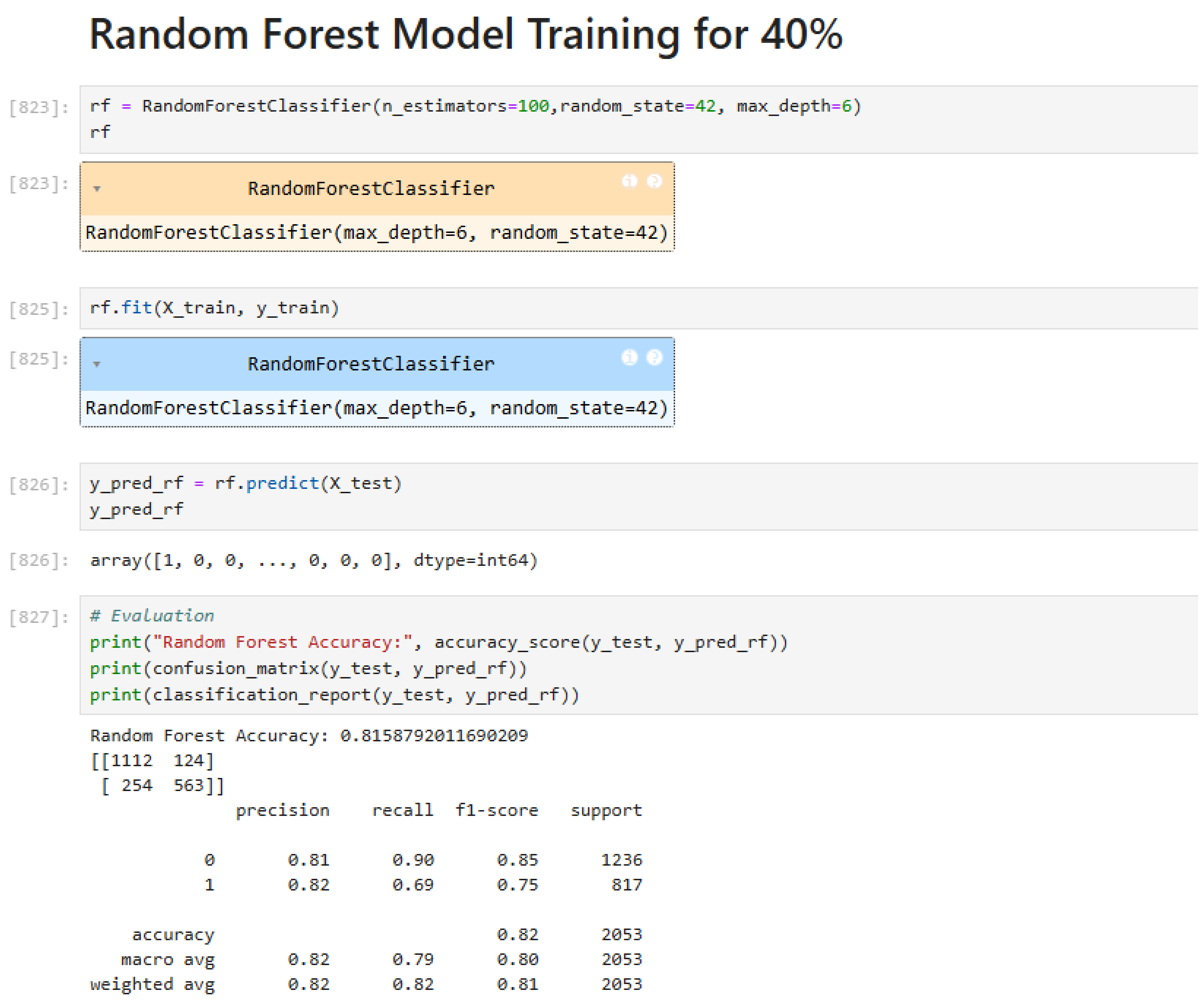
```
[473]: # Evaluation
       print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
       print(confusion_matrix(y_test, y_pred_svm))
       print(classification_report(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.7978567949342426
[[1036  200]
 [ 215  602]]
              precision    recall  f1-score   support

           0       0.83      0.84      0.83      1236
           1       0.75      0.74      0.74       817

    accuracy                           0.80      2053
   macro avg       0.79      0.79      0.79      2053
weighted avg       0.80      0.80      0.80      2053
```

Figure 43: SVM Classifier Model Training for 40%.

The training process is done for the logistic regression for 60:40 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.21 Random Forest Model Training and Accuracy for 40%



Random Forest Model Training for 40%

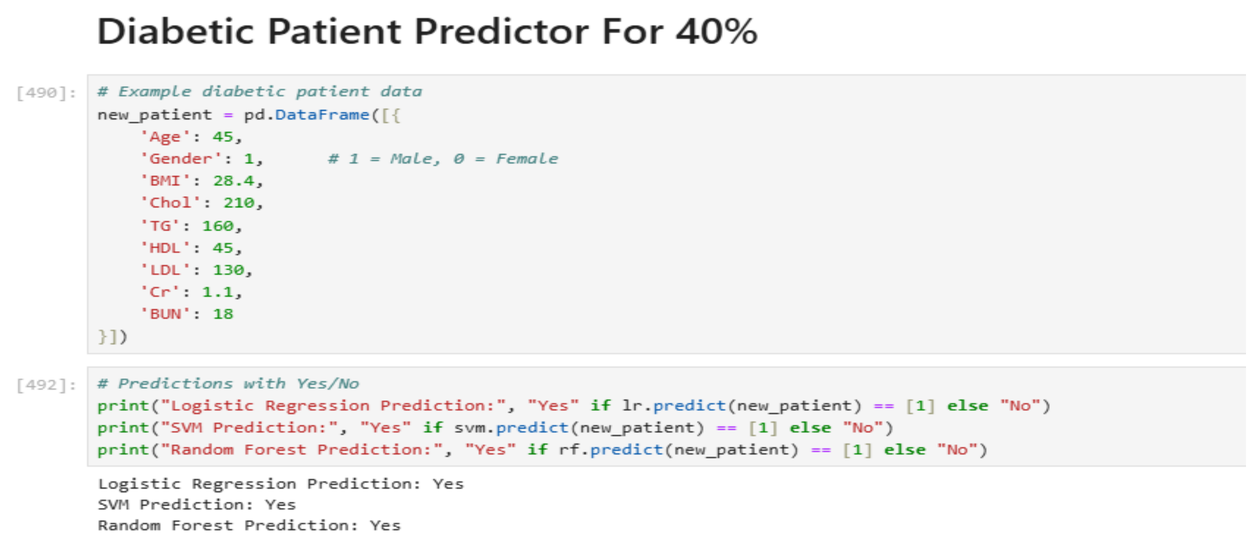
```
[823]: rf = RandomForestClassifier(n_estimators=100,random_state=42, max_depth=6)
       rf

[823]: RandomForestClassifier
       RandomForestClassifier(max_depth=6, random_state=42)

[825]: rf.fit(X_train, y_train)

[825]: RandomForestClassifier
       RandomForestClassifier(max_depth=6, random_state=42)

[826]: y_pred_rf = rf.predict(X_test)
       y_pred_rf

[826]: array([1, 0, 0, ..., 0, 0, 0], dtype=int64)

[827]: # Evaluation
       print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
       print(confusion_matrix(y_test, y_pred_rf))
       print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8158792011690209
[[1112  124]
 [ 254  563]]
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      1236
           1       0.82      0.69      0.75       817

    accuracy                           0.82      2053
   macro avg       0.82      0.79      0.80      2053
weighted avg       0.82      0.82      0.81      2053
```

Figure 44: Random Forest Model Training for 40%.

The training process is done for the logistic regression for 60:40 where the machine learns the patterns from the training data to predict the diabetic and non-diabetic patients. The evaluation includes the classification report including the precision, recall, f1-score and support.

3.6.22 Diabetic Patient Predictor for 40%



Diabetic Patient Predictor For 40%

```
[490]: # Example diabetic patient data
new_patient = pd.DataFrame([{
    'Age': 45,
    'Gender': 1,       # 1 = Male, 0 = Female
    'BMI': 28.4,
    'Chol': 210,
    'TG': 160,
    'HDL': 45,
    'LDL': 130,
    'Cr': 1.1,
    'BUN': 18
}])
```

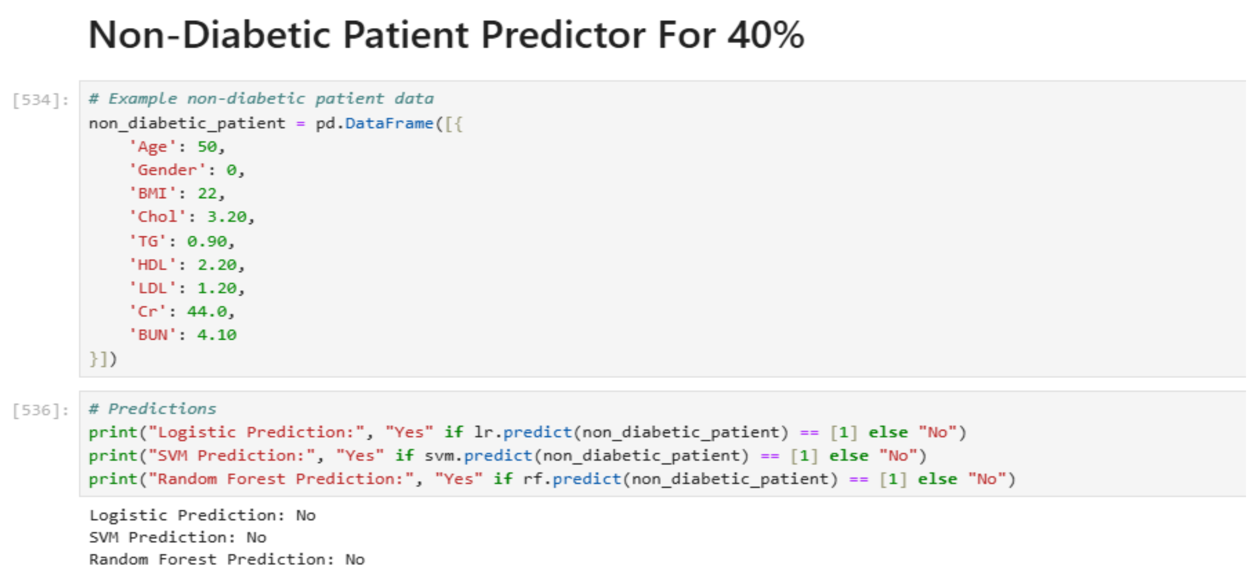
```
[492]: # Predictions with Yes/No
print("Logistic Regression Prediction:", "Yes" if lr.predict(new_patient) == [1] else "No")
print("SVM Prediction:", "Yes" if svm.predict(new_patient) == [1] else "No")
print("Random Forest Prediction:", "Yes" if rf.predict(new_patient) == [1] else "No")
```

```
Logistic Regression Prediction: Yes
SVM Prediction: Yes
Random Forest Prediction: Yes
```

Figure 45: Diabetic Patient Predictor for 40%.

A sample of diabetic dataset is created using random patient medical report. The data is then passed through all the trained models (Logistic, SVM, Random Forest). Each model evaluates the input features and predicted data is then displayed.

3.6.23 Non-diabetic Patient Predictor for 40%



Non-Diabetic Patient Predictor For 40%

```
[534]: # Example non-diabetic patient data
non_diabetic_patient = pd.DataFrame([{
    'Age': 50,
    'Gender': 0,
    'BMI': 22,
    'Chol': 3.20,
    'TG': 0.90,
    'HDL': 2.20,
    'LDL': 1.20,
    'Cr': 44.0,
    'BUN': 4.10
}])
```

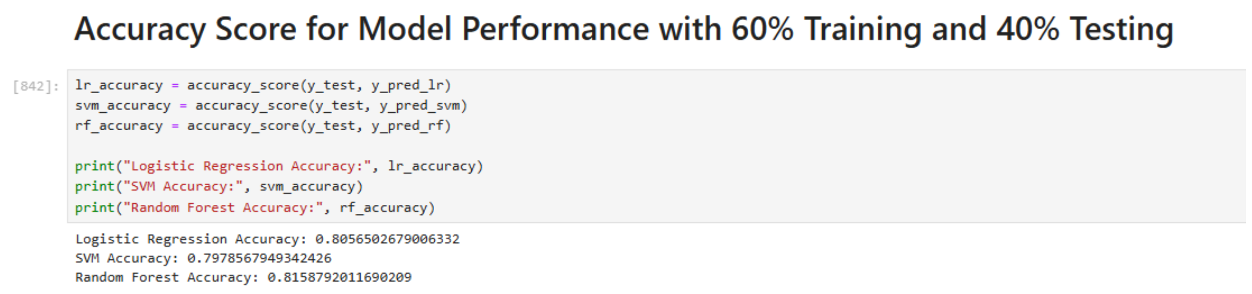
```
[536]: # Predictions
print("Logistic Prediction:", "Yes" if lr.predict(non_diabetic_patient) == [1] else "No")
print("SVM Prediction:", "Yes" if svm.predict(non_diabetic_patient) == [1] else "No")
print("Random Forest Prediction:", "Yes" if rf.predict(non_diabetic_patient) == [1] else "No")
```

```
Logistic Prediction: No
SVM Prediction: No
Random Forest Prediction: No
```

Figure 46: Non-diabetic Patient Predictor for 40%.

A sample of non-diabetic dataset is created using random patient medical report. The data is then passed through all the trained models (Logistic, SVM, Random Forest). Each model evaluates the input features and predicted data is then displayed.

3.6.24 Accuracy Score Results for 40%



Accuracy Score for Model Performance with 60% Training and 40% Testing

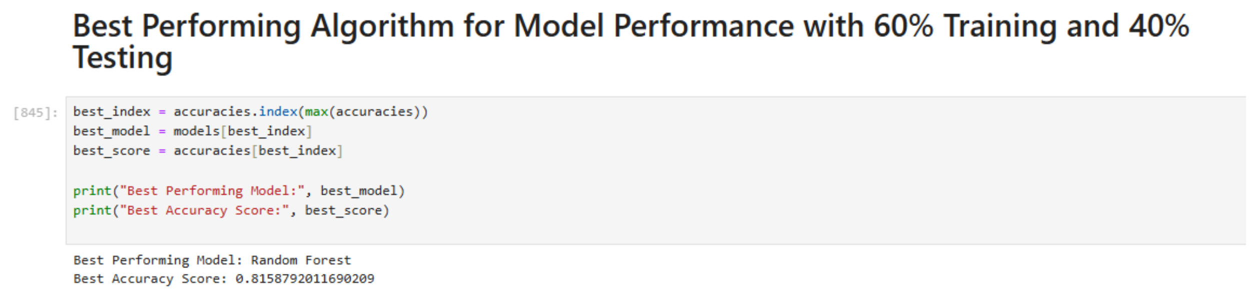
```
[842]: lr_accuracy = accuracy_score(y_test, y_pred_lr)
svm_accuracy = accuracy_score(y_test, y_pred_svm)
rf_accuracy = accuracy_score(y_test, y_pred_rf)

print("Logistic Regression Accuracy:", lr_accuracy)
print("SVM Accuracy:", svm_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Logistic Regression Accuracy: 0.8056502679006332
SVM Accuracy: 0.7978567949342426
Random Forest Accuracy: 0.8158792011690209
```

Figure 47: Accuracy Score Results for 40%.

The accuracy score displays the accuracy of all the algorithms for 60:40 and represents the percentage of the accuracy made by all the used algorithms which helps in identifying the best algorithm for the prediction.

3.6.25 Best Performing Algorithm for 40%



Best Performing Algorithm for Model Performance with 60% Training and 40% Testing

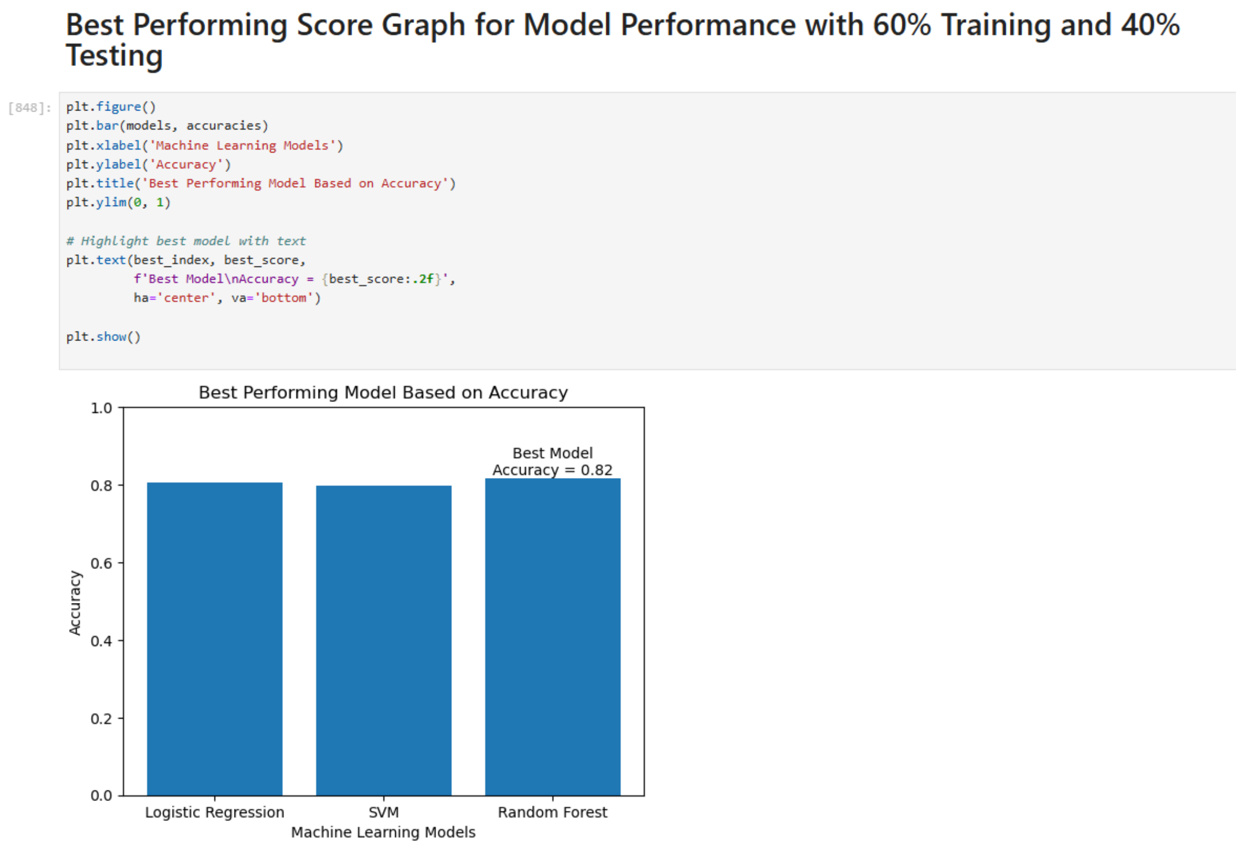
```
[845]: best_index = accuracies.index(max(accuracies))
best_model = models[best_index]
best_score = accuracies[best_index]

print("Best Performing Model:", best_model)
print("Best Accuracy Score:", best_score)
```

```
Best Performing Model: Random Forest
Best Accuracy Score: 0.8158792011690209
```

Figure 48: Best Performing Algorithm for 40%.

This image displays the best algorithm for this diabetes prediction model using the accuracy score.

3.6.26 Best Performing Score Graph for 40%



Best Performing Score Graph for Model Performance with 60% Training and 40% Testing

```
[848]: plt.figure()
plt.bar(models, accuracies)
plt.xlabel('Machine Learning Models')
plt.ylabel('Accuracy')
plt.title('Best Performing Model Based on Accuracy')
plt.ylim(0, 1)

# Highlight best model with text
plt.text(best_index, best_score,
         f'Best Model\nAccuracy = {best_score:.2f}',
         ha='center', va='bottom')

plt.show()
```

Figure 49: Best Performing Score Graph for 40%.

The image displays the bar graph of the best performing algorithm.

4. Conclusion

This diabetes predictor project is focused on developing a predicting system using supervised ML classification. This project focused on analysing structured medical diabetes dataset containing all the needed attributes to build a diabetes prediction model. All the needed data pre-processing was applied to prepare the dataset for the model training. This project was done using three best classification algorithms. Logistic Regression, Random Forest and SVM were implemented to predict whether an individual is diabetic or non-diabetic.

This project is completed applying the necessary data preparation tasks and model training tasks. These included all the data cleaning process, handling missing values and removing the irrelevant columns for suitable machine learning. Data visualization process is done to display the data information such as heatmap, bar graph and scatter plot. These steps help in effective model training. The models were then trained and tested using multiple train test split ratios if 80:20, 70:30, 60:40 to evaluate the model performance. Three supervised learning classification model algorithms (logistic regression, random forest classifier, support vector machine) are being used to train the dataset. The predicted data is then evaluated using performance metrics to calculate the accuracy, precision, recall, f1-score and support. At last, the best performing algorithm is displayed to visualize using bar graph identifying the best performing algorithm. After the successful model training the system was tested using both diabetic and non-diabetic random medical report. The predicted data is then displayed for model evaluation.

Among these three models the Random Forest Classifier achieved the highest accuracy rate across different data train split ratio. Although the random forest algorithm indicated overfitting at the start, the issue was then successfully solved using max_depth regularization. In conclusion, the Random Forest Classifier emerged as the best performing algorithm for this diabetes prediction project.

5. References

Ahmed, A. (2024) *MDPI* [Online]. Available from: https://www.mdpi.com/2227-9032/13/1/37#healthcare-13-00037-t006 [Accessed 15 December 2025].

B.V., E. (2017) *ScienceDirect* [Online]. Available from: https://www.sciencedirect.com/science/article/pii/S2001037016300733 [Accessed 15 December 2025].

Clinic, C. (2023) *Cleveland Clinic* [Online]. Available from: https://my.clevelandclinic.org/health/diseases/7104-diabetes [Accessed 17 December 2025].

ISO. (2022) *ISO* [Online]. Available from: https://www.iso.org/artificial-intelligence/machine-learning [Accessed 13 December 2025].

Kavlakogl, E. (2025) *IBM* [Online]. Available from: https://www.ibm.com/think/topics/support-vector-machine#684929714 [Accessed 16 December 2025].

Kavlakoglu, E. (2025) *IBM* [Online]. Available from: https://www.ibm.com/think/topics/random-forest#684929713 [Accessed 16 December 2025].

Khan, R. (2022) *National Library of Medicine* [Online]. Available from: https://pmc.ncbi.nlm.nih.gov/articles/PMC10107388/#htl212039-bib-0003 [Accessed 15 December 2025].

Lee, F. (202) *IBM* [Online]. Available from: https://www.ibm.com/think/topics/logistic-regression [Accessed 16 December 2025].

McKinsey. (2024) *McKinsey & Company* [Online]. Available from: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-ai [Accessed 13 December 2025].

Sheldon, R. (2023) *TechTarget* [Online]. Available from: https://www.techtarget.com/whatis/definition/pseudocode [Accessed 16 December 2025].

Tech, M. (2025) *Michigan Tech* [Online]. Available from: https://www.mtu.edu/computing/ai/ [Accessed 13 December 2025].

WHO. (2024) *World Health Organization (WHO)* [Online]. Available from: https://www.who.int/news-room/fact-sheets/detail/diabetes [Accessed 13 December 2025].