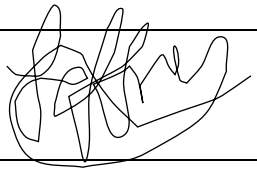
C3879C - Capstone Project

AY2021 Term 3

Coursework Final Submission

Personal Details	
Name	**WONG QI YUAN, JEFFREY**
Student No.	**20053371**

Compliance Statement	
Plagiarism I declare that this report is my original work. I understand that if I am suspected of plagiarism, my enrolment in the programme may be terminated.	✓
Retention of Backup Copy I declare that I have a back-up electronic copy of this report for immediate submission.	✓
Signature	
IMPORTANT: Non-compliance to these clauses will result in unconditional rejection of your submission	

School of Infocomm (SOI)

Republic Polytechnic

DEVELOPMENT OF HEALTH WEB APP WITH INTEGRATION OF ENSEMBLE MACHINE LEARNING FOR EARLY DIAGNOSIS IN OBESITY-RELATED CHRONIC DISEASES

Date of Submission: 01-10-2021

Submitted By:

20053371 WONG QI YUAN, JEFFREY

School of Infocomm (SOI)

Republic Polytechnic

Table of Contents

1.0 Abstract

With the advancement of technologies in the 21st century, artificial intelligence and machine learning have been widely used in the healthcare fields by making use of the predictive models to predict the possible risks of developing obesity-related chronic diseases such as obesity, diabetes, and hypertension at the early stages. Thus, the intervention not only improves the overall healthcare quality but also improving the individual's overall health status by minimizing the detrimental effects caused by obesity-related chronic diseases. This capstone project developed some predictive ensemble machine learning algorithms such as random forest, gradient boosting, and/ or extra trees to provide an early prediction for obesity, diabetes, and hypertension based on individual risk factors data.

Prior to algorithms training, an isolation forest algorithm was used to detect and remove the potential outlier data, followed by synthetic minority oversampling technique Tomek link (SMOTE-Tomek) to balance the training data distribution such that minimized the biased and unreliable outcomes on the machine learning model performance, and the training of the predictive algorithms with the best hyperparameters to predict the respective diseases. Three datasets were utilized to implement the proposed predictive models and the attributes were chosen based on the relevance to the disease contexts. The outcome of the performance evaluation based on the testing sets showed that the proposed predictive models with hyperparameters for each respective disease were achievable with high accuracy of at least 90%, at least 95% for ROC-AUC, and less than 5% of false-positive rate and false-negative rate.

This project also developed a simple and user-friendly web application using Streamlit to simulate the practical application of the proposed predictive models. The developed web application collates all the risk factors from the user inputs and then sends them to the proposed prediction models to predict the respective diseases based on the class probabilities. The prediction outcome, as well as the constructive recommendation, are then displayed on the web application. Thus, prompt action can be taken swiftly to minimize the effects and prevent individual risks once undesirable health conditions such as obesity, diabetes, and hypertension are detected at the early stages.

2.0 Acknowledgement

The success and final outcome of this capstone project required a lot of guidance, assistance, and past-experience from many people, especially the lecturers from Specialist Diploma in Artificial Intelligence, who previously sharing their knowledge and skills in AI and Machine Learning. I am extremely privileged to have got this all along with the completion of this project and I would not forget to thank them.

3.0 Background

3.0.1 Problem Statement

In a recent report, the World Health Organization (WHO) mentioned that obesity-related chronic diseases have contributed to approximately about 41 million premature deaths annually, which is about 71% of all death worldwide. If the situation is unmitigated, we are expected that the overall number of obesity-related chronic diseases related-deaths to be increased up to 52 million yearly by the end of 2030. The most common obesity-related chronic diseases are the obesity, diabetes, and the hypertension.

Furthermore, as the world currently facing the crisis of the COVID-19 pandemic, hence it is expected that the number of people developing the obesity-related chronic diseases to rise as most people are working from home in order to curb the spread of the COVID-19 in the local community. Furthermore, based on a recent news article, about 1/3 of the Singaporeans have gained weight during the COVID-19 pandemic and many fingers point their fingers at how their lifestyles have changed to become more sedentary since the COVID-19 pandemic.

Thus, it is an alarming concern in the road of the covid-19 pandemic as the majority of the people are unaware of their overall health status and some of the people do not have a little bit of time to go for a simple health screening.

3.0.2 Overview of the Project Context

In this capstone project, the implementation of the isolation forest algorithm, the SMOTE-TOMEK, and the various classifier algorithms such as random forest, gradient boosting, or extra trees will be integrated to predict the respective possible risk of developing obesity-related chronic diseases such as obesity, diabetes, and hypertension.

Furthermore, three datasets were downloaded from various sources to be used to trained and evaluate the performance of the proposed classifier models for respective obesity-related chronic diseases.

In addition, a web application via Streamlit will be implemented to simulate the real-life practical application with the integration of the proposed classifier models such that offering individual's an effective and appropriate way to monitor their current health status.

How does this health web application benefit the user?

This health web application is significant as it would provide an individual to know their likelihood of being developing the obesity-related chronic-diseases such as diabetes, hypertension, and obesity, and hence, reduce their time to visit the office clinic in person. Furthermore, being cost-effective, the users are able to know the diagnosis right on the spot and also provide the users the time to prevent and manage the chronic-diseases by making them aware of their present condition.

3.0.3 Basic Concepts of Obesity-related Chronic Diseases

Obesity can be defined as a weight that is higher than what is considered healthy for a given weight. BMI is the main tool to screen for obesity. Obesity is a complex health issue resulting from a combination of causes and individuals' factors such as genetics and behaviour. Behaviours can include sedentary lifestyles, physical activity, dietary patterns, and other exposures. Obesity is serious because it is often associated with poor mental health and reduced quality of life. Furthermore, obesity is also associated with the leading causes of death, including diabetes, and high blood pressure.

Diabetes is a metabolic disorder that changes the blood sugar levels in the bloodstream. When there is insufficient insulin or cells stop responding to insulin, too much blood sugar will remain in your bloodstream. Over time, this usually leads to consequences of health issues such as vision loss, kidney failure, etc.

Hypertension (High Blood Pressure) is a blood pressure that is usually higher than the global health recommended blood pressure (< 140/100 mm Hg). However, having blood pressure that is constantly above the recommended range may result in the diagnosis of high blood pressure or hypertension. Hypertension usually develops over time and it could happen due to sedentary lifestyles or certain existing health conditions such as diabetes, or obesity.

4.0 Methodology and Design

Figure 4.0 illustrates a schematic architecture design of the ML predictive models for all the respective diseases (obesity, diabetes, and hypertension)

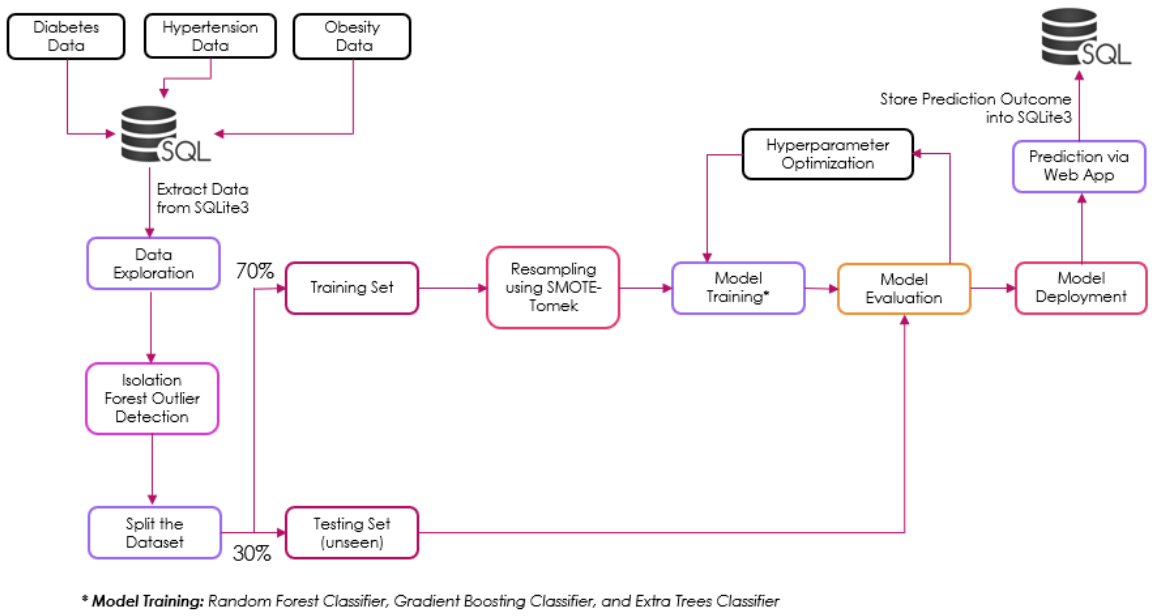


Figure 4.0: A schematic architecture design of the ML predictive models for all respective diseases

4.1 Datasets and SQL Database

4.1.1 Diabetes

The diabetes dataset was downloaded from **Mendeley Data** and the data were collected by the Iraqi society as were acquired from the laboratory of Medical City Hospital and the Specialization Center for Endocrinology and Diabetes AL Kindy Teaching Hospital. The **original** data consists of the patient ID, gender, age, urea, creatinine ratio (Cr), blood glucose level (HbA1c), cholesterol (chol), triglycerides (TG), high density lipid (HDL), low density lipoprotein (LDL), very-low density lipoprotein (VLDL), body-mass index (BMI), and the class outcomes (N: non-diabetic, P: pre-diabetic, or Y: diabetic).

Figure 4.1.1a shows the diabetes dataset (original) that was stored into SQLite3 database via Python.

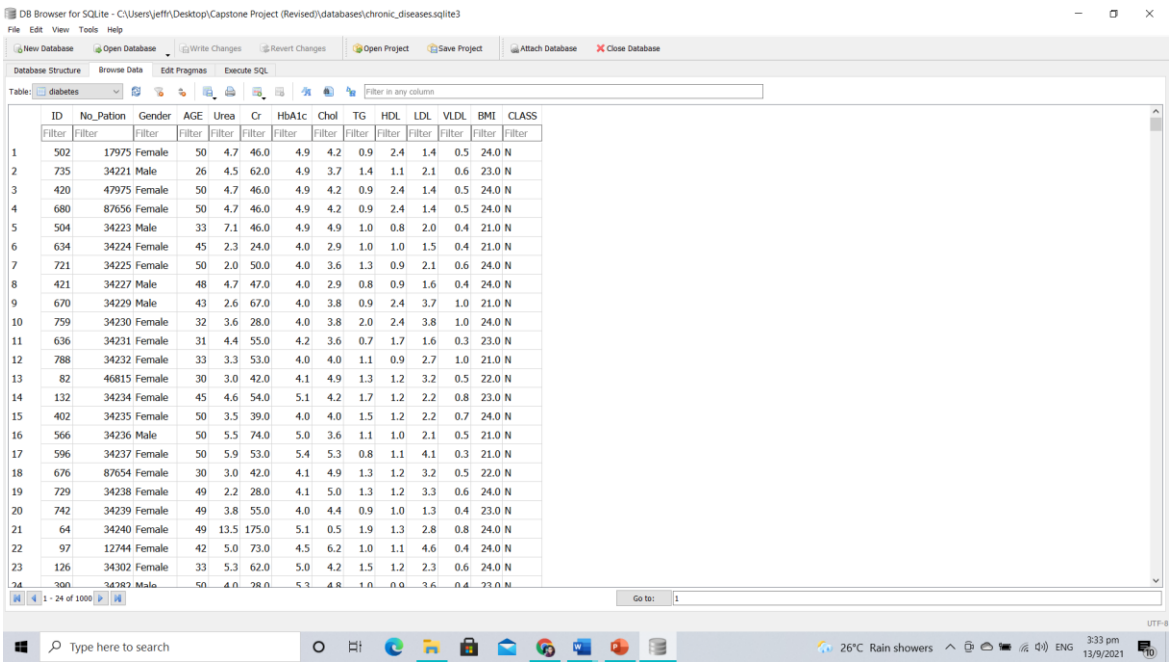


Figure 4.1.1a: Diabetes Dataset (Original)

Table 4.1.1a shows the description of the attributes for the diabetes dataset (original)

Attribute	**Description**
ID	Patient identification number
No_Pation	N/A
Gender	Male or Female
AGE	Age (in years)
Urea	Measure the amount of urea in a blood sample
Cr	Measure the level of creatinine in a blood sample
HbA1C	Measure the amount of fasting blood sugar in a blood sample
Chol	Measure the amount of cholesterol and triglycerides in a blood sample
TG	Measure the amount of thyroglobulin in a blood sample
HDL	Measure the amount of cholesterol found inside high-density lipoprotein in a blood sample
LDL	Measure the amount of cholesterol found inside low-density lipoprotein in a blood sample

VLDL	Measures the amount of cholesterol found inside very-low-density lipoprotein in a blood sample
BMI	Measures the amount of body fat based on height and weight
CLASS	N (Class 0): No Diabetes, P (Class 1): Pre-Diabetes, Y (Class 2): Diabetes

Table 4.1.1a: Description of the Attributes for Diabetes Dataset (Original)

As such, the relevant attributes to be used for training phases were **Gender, HbA1c, BMI, and the Class Outcomes** as the rest of the attributes were irrelevant to the diabetes context.

Figure 4.1.1b illustrate a sample python code to extract the relevant column data from the SQL database for diabetes.

```
23  def load_datasets():
24      conn = sqlite3.connect(r"C:\Users\jeffr\Desktop\Capstone Project (Revised)\databases\chronic_diseases.sqlite3")
25      diabetes_df = pd.read_sql("SELECT Gender, HbA1C, BMI, CLASS FROM diabetes", conn)
26
27      # create a set of copy of diabetes dataset
28      copy_diabetes_df = diabetes_df.copy()
29
30      conn.close()
31      return copy_diabetes_df
```

Figure 4.1.1b: A sample code to extract the relevant column data from SQL database

Figure 4.1.1c illustrate the output of the relevant column data extracted from the SQL database via Python code.

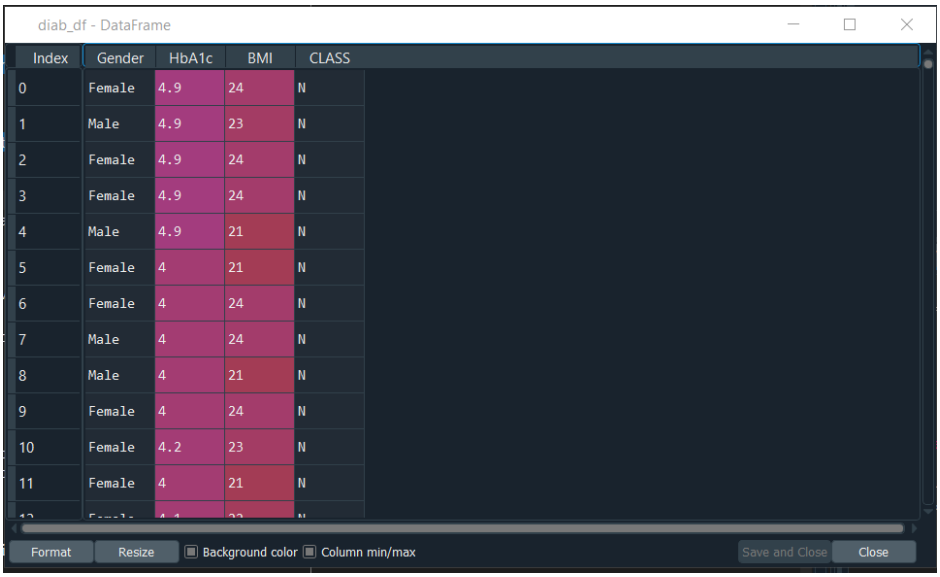


Figure 4.1.1c: An output of the relevant column data extracted from the SQL database via Python Code

4.1.2 Hypertension (or High Blood Pressure)

The hypertension (or high blood pressure) dataset was downloaded from **figshare** and the **original** dataset consists of the age, obese (Yes or No), body mass index (BMI), waist (WC), hip circumference (HC), waist-hip ratio (WHR), diastolic blood pressure (DBP), systolic blood pressure (SDB), and the class outcomes (Hypertension or No Hypertension).

Figure 4.1.2a shows the hypertension dataset (original) that was stored into SQLite3 database via Python.

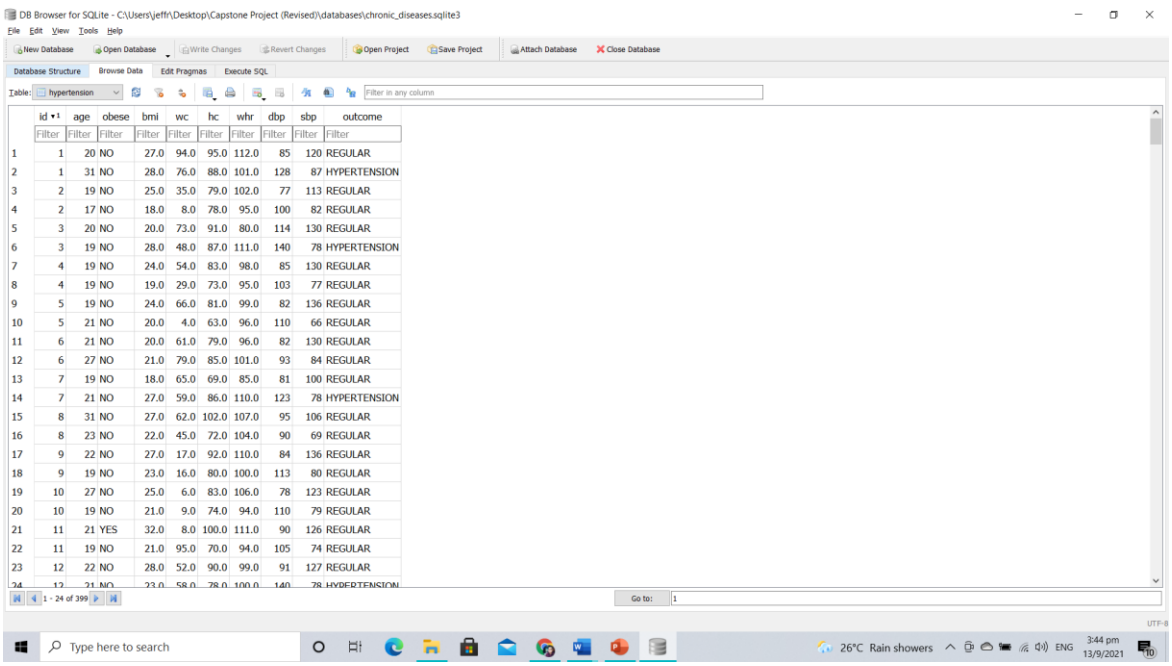


Figure 4.1.2a: Hypertension Dataset (Original)

Table 4.1.2a shows the description of the attributes for the hypertension dataset (original)

Attribute	Description
id	Patient identification numbers
age	Age (in years)
obese	Obesity Status
bmi	Measure the amount of body fat based on height and weight
wc	Measure the waist circumference (in cm)
hc	Measure the hip circumference (in cm)
whr	Ratio of the waist circumference (in cm) and the hip circumference (in cm)

dbp	Diastolic blood pressure is the bottom number found in the blood pressure measuring device. It measures the force your heart exerts on the wall of your arteries each time in between beats.
sbp	Systolic blood pressure is the top number found in the measuring blood pressure measuring device. It measures the force your heart exerts on the wall of your arteries each time it beats.
outcome	Class 0: Regular, Class 1: Hypertension

Table 4.1.2a: Description of the Attributes for Hypertension Dataset (Original)

As such, the relevant attributes to be used for training phases were **BMI, DBP, SBP, and the Class Outcomes** as the remaining attributes were irrelevant to the hypertension context.

Figure 4.1.2b illustrate a sample python code to extract the relevant column data from the SQL database for hypertension.

```
def load_datasets():
    conn = sqlite3.connect(r"C:\Users\jeffr\Desktop\Capstone Project (Revised)\databases\chronic_diseases.sqlite3")
    hypertension_df = pd.read_sql("SELECT bmi, dbp, sbp, outcome FROM hypertension", conn)

    # create a set of copy of diabetes dataset
    copy_hypertension_df = hypertension_df.copy()

    conn.close()
    return copy_hypertension_df
```

Figure 4.1.2b: A sample code to extract the relevant column data from SQL database

Figure 4.1.2c illustrate the output of the relevant column data extracted from the SQL database via Python code for hypertension.

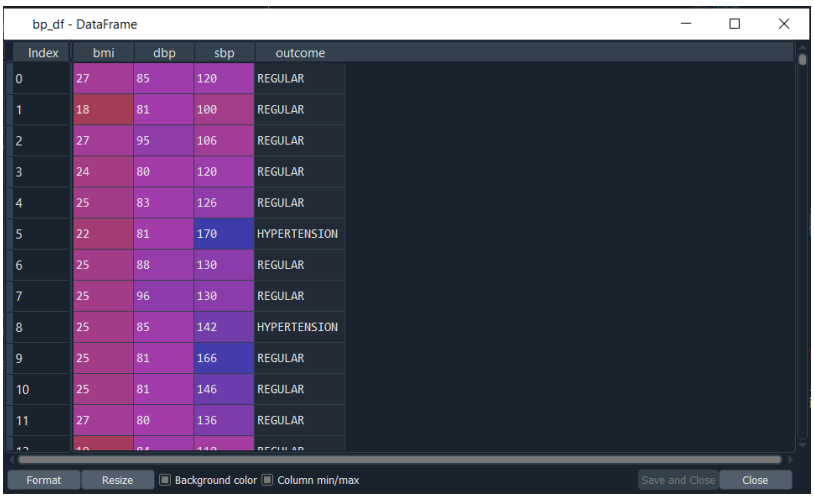


Figure 4.1.2c: An output of the relevant column data extracted from the SQL database via Python Code

4.1.3 Obesity

The obesity (or high blood pressure) dataset was downloaded from **neuraldesigner** and the **original** dataset consists of the attributes of the gender, age, height, weight, family history with overweight, caloric food, vegetables, number meals, food between meals, smoke, water, calories, activity, technology, alcohol, transportation, and the class outcomes of the obesity level.

Figure 4.1.3a shows the obesity dataset (original) that was stored into SQLite3 database via Python.

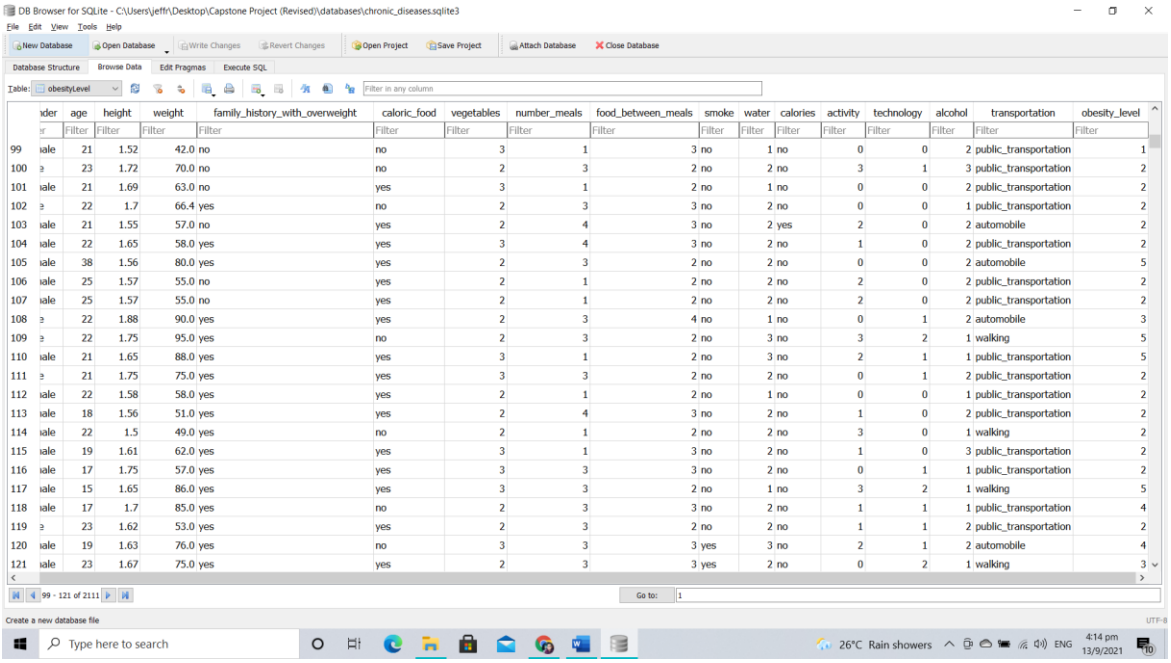


Figure 4.1.3a: Obesity Dataset (Original)

Table 4.1.3a shows the description of the attributes for the obesity dataset (original)

Attribute	Description
gender	Male or Female
age	Age (in years)
height	Measure the height of an individual (in m)
weight	Measure the weight of an individual (in kg)
family_history_ow	To check whether if there is a family history with overweight. (0 = No and 1= Yes)

caloric_food	Does an individual consumed high caloric food? (0 = No and 1 = Yes)
vegetables	How frequent consumption of vegetables? (1 = Not always, 2 = Frequently, 3 = Often)
number_meals	What is the number of meals per day? (1, 2, 3, or 4)
food_between_meals	How often do you consume foods between meals? (1 = Not always, 2 = Sometimes, 3 = Frequently, 4 = Always)
smoke	Do you smoke? (0 = No, 1 = Yes)
water	How often do you drink water daily? (1 = Not always, 2 = Sometimes, 3 = Frequently)
calories	Do you often monitor your calories? (0 = No, 1 = Yes)
activity	How often do you exercise? (0 = Not always, 1 = Sometimes, 2 = Frequently, 3 = Always)
technology	How often do you use your electronic devices? (0 = Sometimes, 1 = Frequently, 2 = Always)
alcohol	Do you drink alcohol? (1 = Not always, 2 = Sometimes, 3 = Frequently, 4 = Always)
transportation	Types of transportation used: Automobile, Motorbike, Bike, Public Transportation, Walking
obesity_level	1 = Insufficient Weight, 2 = Normal Weight, 3 = Overweight Level I, 4 = Overweight Level II, 5 = Obesity Type I, 6 = Obesity Type II, 7 = Obesity Type III

Table 4.1.3a: Description of the Attributes for Obesity Dataset (Original)

As such, all the relevant attributes were used for training phases except for transportation.

Figure 4.1.3b illustrate a sample python code to extract the relevant column data from the SQL database for obesity.

```
def load_datasets():
    conn = sqlite3.connect(r"C:\Users\jeffr\Desktop\Capstone Project (Revised)\databases\chronic_diseases.sqlite3")
    obesity_df = pd.read_sql("SELECT gender, age, height, weight, family_history_with_overweight, \
                             caloric_food, vegetables, number_meals, food_between_meals, smoke, water, \
                             calories, activity, technology, alcohol, obesity_level FROM obesityLevel", conn)

    # create a set of copy of diabetes dataset
    copy_obesity_df = obesity_df.copy()

    # rename the target column
    copy_obesity_df.rename(columns = {'obesity_level': 'outcome', 'family_history_with_overweight': 'family_history_ow'}, inplace = True)

    # manipulate some of the target values
    copy_obesity_df['outcome'].replace([(3, 4), (5, 6, 7)], (3, 4), inplace = True)

    conn.close()
    return copy_obesity_df
```

Figure 4.1.3b: A sample code to extract the relevant column data from SQL database

Figure 4.1.3c illustrate the output of the relevant column data extracted from the SQL database via Python code for obesity.

obs_df - DataFrame

Index	gender	age	height	weight	ily_history	aloric_foo	egetable:	mber_mea	between_	smoke	water	calories	activity	echnolog	alcohol	outcome
0	Female	21	1.62	64	yes	no	2	3	2	no	2	no	0	1	1	2
1	Female	21	1.52	56	yes	no	3	3	2	yes	3	yes	3	0	2	2
2	Male	23	1.8	77	yes	no	2	3	2	no	2	no	2	1	3	2
3	Male	27	1.8	87	no	no	3	3	2	no	2	no	2	0	3	3
4	Male	22	1.78	89.8	no	no	2	1	2	no	2	no	0	0	2	3
5	Male	29	1.62	53	no	yes	2	3	2	no	2	no	0	0	2	2
6	Female	23	1.5	55	yes	yes	3	3	2	no	2	no	1	0	2	2
7	Male	22	1.64	53	no	no	2	3	2	no	2	no	3	0	2	2
8	Male	24	1.78	64	yes	yes	3	3	2	no	2	no	1	1	3	2
9	Male	22	1.72	68	yes	yes	2	3	2	no	2	no	1	1	1	2
10	Male	26	1.85	105	yes	yes	3	3	3	no	3	no	2	2	2	4
11	Female	21	1.72	80	yes	yes	2	3	3	no	2	yes	2	1	2	3

Format Resize Background color Column min/max Save and Close Close

Figure 4.1.3c: An output of the relevant column data extracted from the SQL database via Python Code

4.2 Data Exploration on the Selected Attributes

4.2.1 Diabetes

4.2.1.1 Histogram

A **histogram** of each selected attributes in the **Diabetes DataFrame** was created by calling the **hist () function** to get an idea of the variable distribution. Figure 4.2.1.1 shows the histogram subplot and there are only **two numeric input** and **two class labels** in the diabetes DataFrame.

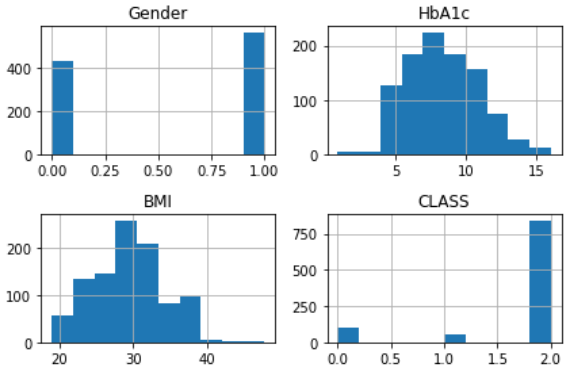


Figure 4.2.1.1: A histogram subplot of the diabetes

Based on the above histogram subplot, it would be expected to scale the distributions to the same range to be useful for modelling algorithms at the later stage, and perhaps the use of scaling techniques at the modelling stage.

4.2.1.2 Heatmap and Pair-plot

Figure 4.2.1.2a shows the **heatmap** of the selected attributes in the Diabetes DataFrame and the objective is to visualize the correlation of each pair of attributes and the outcome variable.

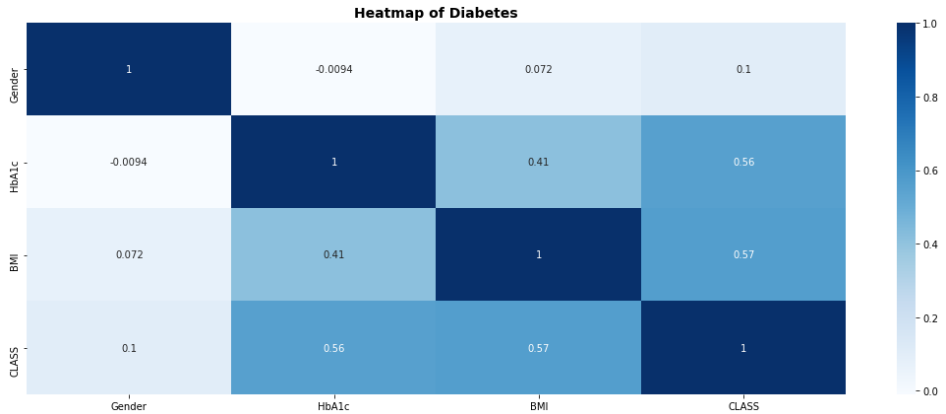


Figure 4.2.1.2a: A heatmap of the selected attributes in diabetes DataFrame

Based on the heatmap shown above, the darker colour indicates more correlation between variables. Even though, both HbA1c and BMI shows some moderate and positive correlation with the class outcome variables, but one must note that this is a classification context based. Therefore, there will be no significant impact on the overall performance during the algorithms analysis. In addition, there is also some weak positive correlation between the HbA1c and BMI variables, but the impact on the overall algorithm's performance is generally very small.

Figure 4.2.1.2b shows the pair-plot of the selected attributes in the Diabetes DataFrame and has the common objective as the Heatmap to visualize the correlation between variables.

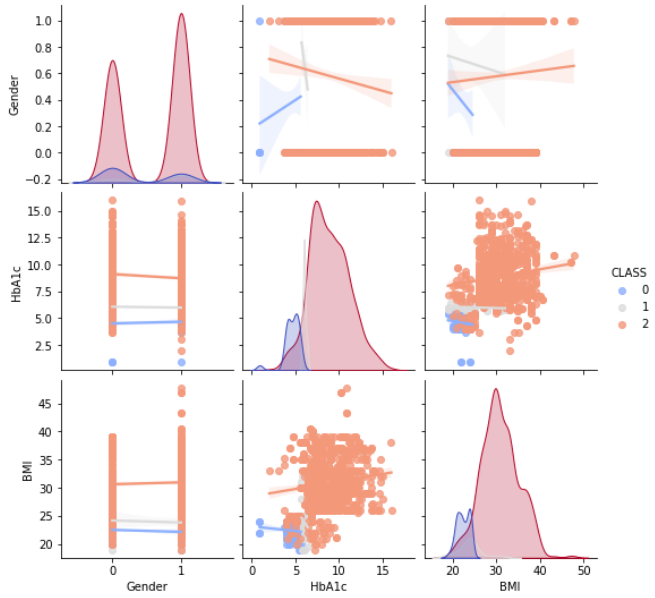


Figure 4.2.1.2b: A pair-plot of the selected attributes in the diabetes DataFrame

From the pair-plot, it can be observed that there is some weak correlation between the HbA1c and BMI attribute which can be proven from the above heatmap.

4.2.2 Hypertension

4.2.2.1 Histogram

Similar to the diabetes, a **histogram** of each selected attributes in the **Hypertension DataFrame** was created by calling the **hist () function** to get an idea of the variable distribution. Figure 4.2.2.1 shows the histogram subplot and there are only **three numeric input** and **one class label** in the hypertension DataFrame.

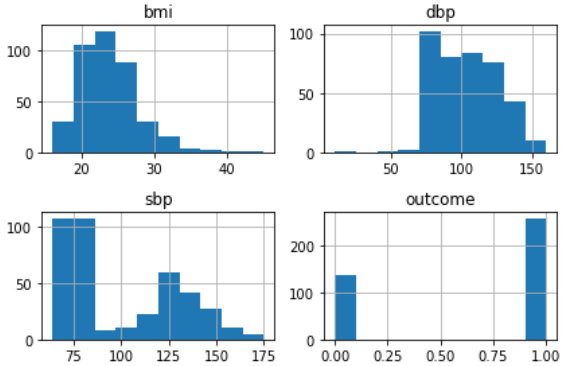


Figure 4.2.2.1: A histogram subplot of the Hypertension

Based on the above histogram subplot, one could be observed the BMI attribute is moderately skewed to the right, whereas the DBP attribute is also moderately skewed to the left. Furthermore, the SBP attribute consists of the bimodal distribution. As such, it would be expected to scale the distributions to the same range to be useful for algorithms analysis, and perhaps some power transforms involved at the modelling stage.

4.2.2.2 Heatmap and Pair-plot

Similar to the diabetes, Figure 4.2.2.2a shows the **heatmap** of the selected attributes in the Hypertension DataFrame and the objective is to visualize the correlation of each pair of attributes and the outcome variable.

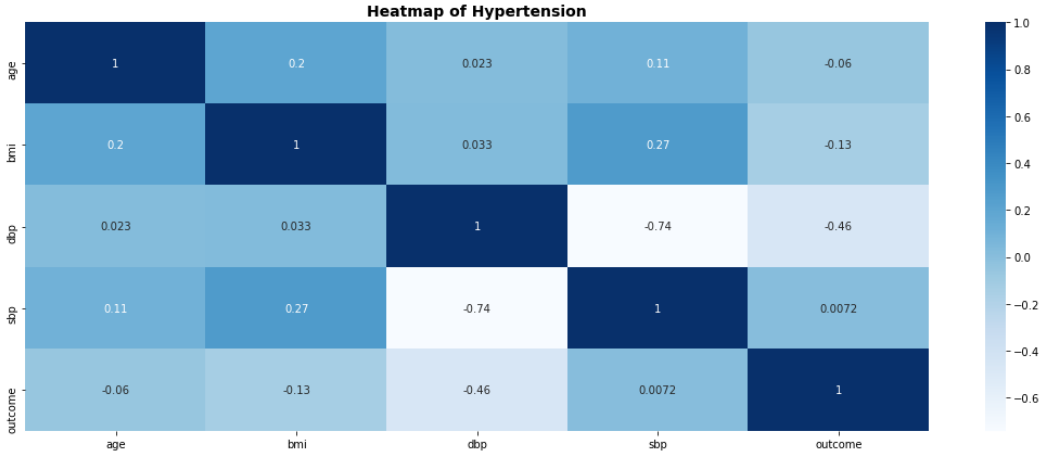


Figure 4.2.2.2a: A heatmap of the selected attributes in hypertension DataFrame

Based on the heatmap shown above, the darker colour indicates more correlation between variables. As such, no significant correlation was found between variables.

Figure 4.2.2.2b shows the pair-plot of the selected attributes in the Hypertension DataFrame and has the common objective as the Heatmap to visualize the correlation between variables.

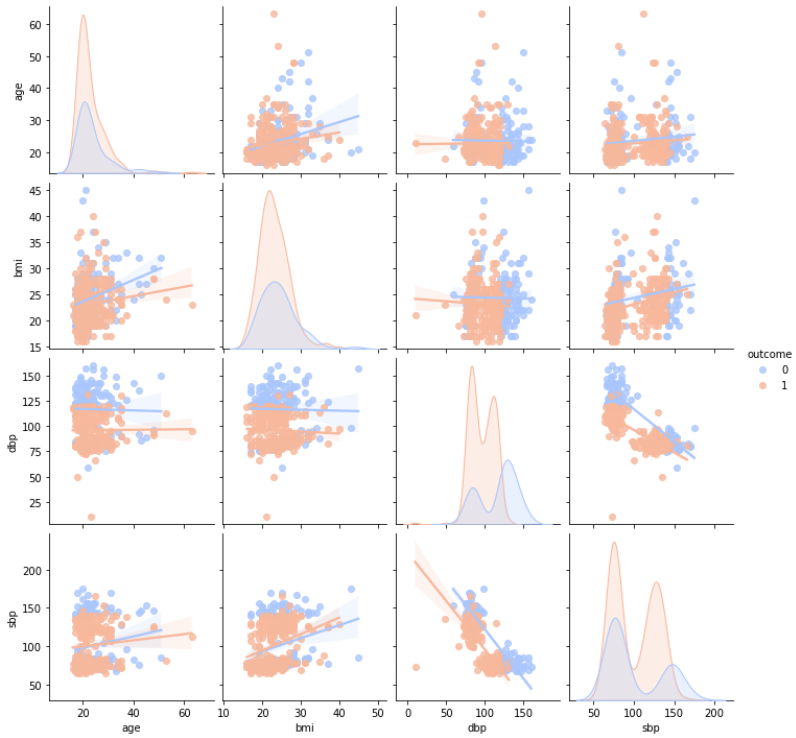


Figure 4.2.2.2b: A pair-plot of the selected attributes in the hypertension DataFrame

From the pair-plot, one can be observed that there is no significant correlation between the variables which can be proven from the above heatmap.

4.2.3 Obesity

4.2.3.1 Histogram

Similar to both diabetes and hypertension, a **histogram** of each selected attributes in the **Obesity DataFrame** was created by calling the **hist () function** to get an idea of the variable distribution. Figure 4.2.3.1 shows the histogram subplot and there are only **3 numeric input**, **13 categorical variables** and **1 class label** in the obesity DataFrame.

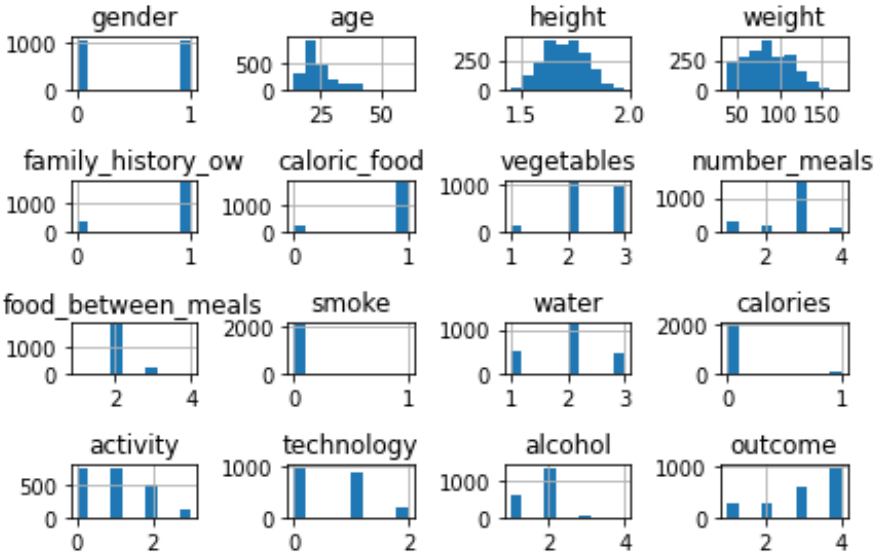


Figure 4.2.3.1: A histogram subplot of the Obesity

Based on the above histogram subplot, one could be observed both the age and weight attributes are moderately skewed to the right, whereas the height attribute represents the Gaussian distribution. As such, it would be expected to scale the distributions to the same range to be useful for algorithms analysis, and perhaps some power transforms involved at the modelling stage.

4.2.3.2 Heatmap and Pair-plot

Similar to the both diabetes and hypertension, Figure 4.2.3.2a shows the **heatmap** of the selected attributes in the Obesity DataFrame and the objective is to visualize the correlation of each pair of attributes and the outcome variable.

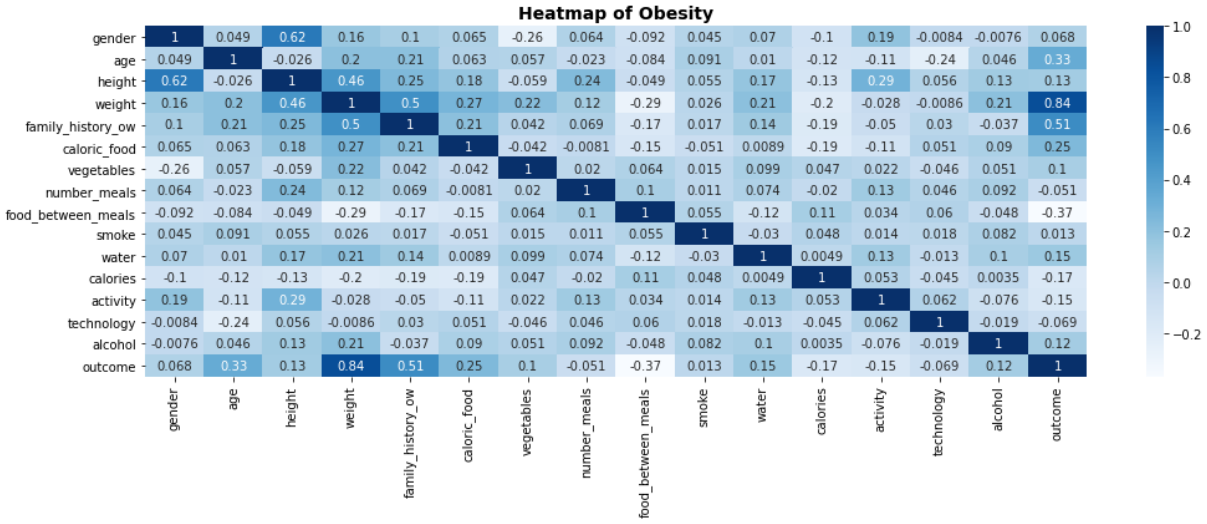


Figure 4.2.3.2a: A heatmap of the selected attributes in obesity DataFrame

Based on the heatmap shown above, the darker colour indicates more correlation between variables. Even though, the weight attribute_shows moderately strong positive correlation between with the class outcomes, but one must note that this is part of the classification context-based. As such, there will be no significant impact on the overall performance of the algorithms analysis. Furthermore, there are some weak and positive correlation between the height, weight, and family history with overweight attributes. Generally, the impact on the algorithm's performance is very small.

Figure 4.2.3.2b shows the pair-plot of the selected attributes in the Obesity DataFrame and has the common objective as the Heatmap to visualize the correlation between variables.

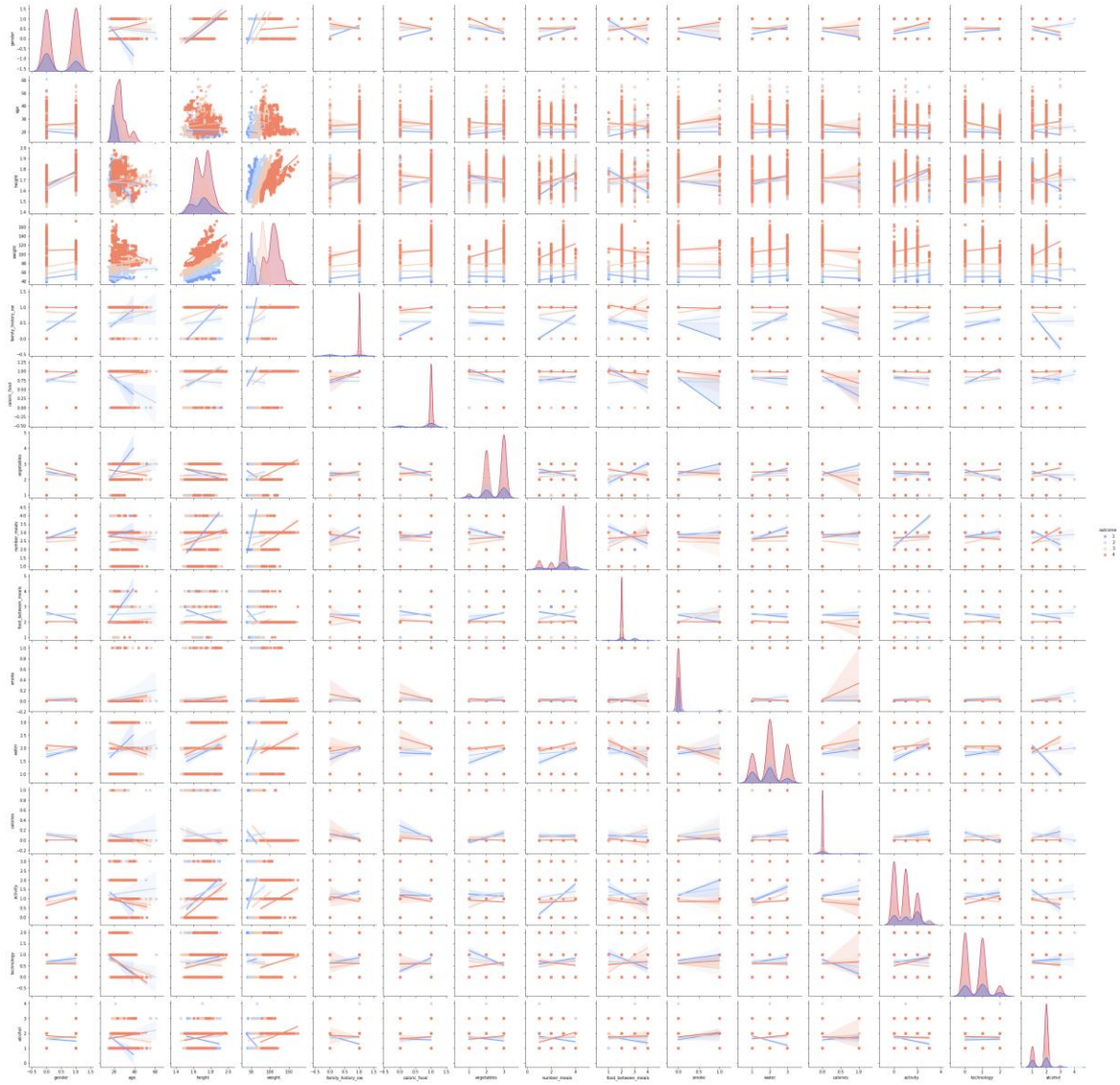


Figure 4.2.3.2b: A pair-plot of the selected attributes in the obesity DataFrame

From the pair-plot, it can observe that there is a weak and positive correlation between the height, weight, and family history with overweight variables which can be proven from the above heatmap.

4.3 Data Cleansing

4.3.1 Investigate Missing Data

Prior to the algorithms analysis, it is imperative to investigate for missing data during the pre-processing of the dataset as not all machine learning algorithms support missing values and might have a chance of leading to inaccurate and bias outcomes.

As such, the Seaborn heatmap was created for all three datasets to visualize the missing row data.

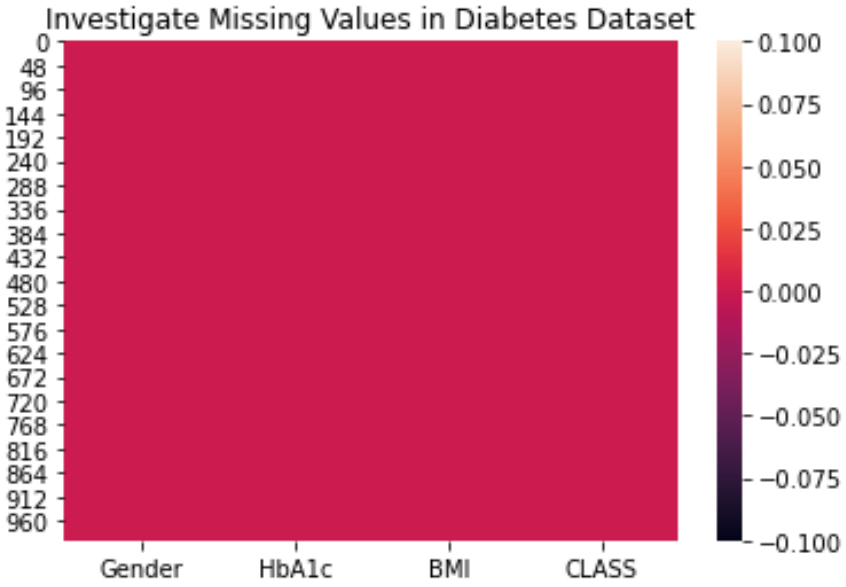


Figure 4.3.1a: Visualize the Missing Row Data in Diabetes Dataset

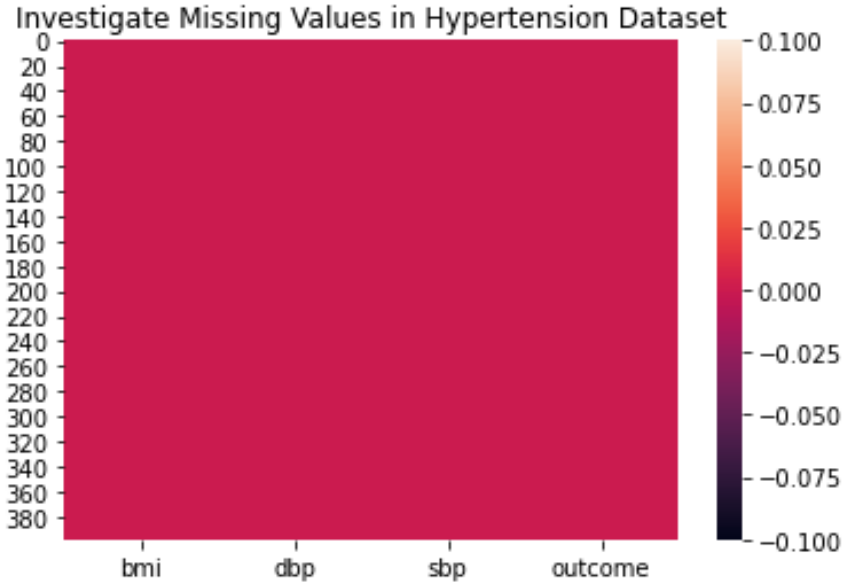


Figure 4.3.1b: Visualize the Missing Row Data in Hypertension Dataset

Figure 4.3.1c: Visualize the Missing Row Data in Obesity Dataset

From the above observations, there were no missing row data found in all three datasets.

4.3.2 Removal of Potential Outlier Detection using Isolation Forest

Isolation Forest (iForest) is the anomaly detection algorithm that identifies anomalies using isolation. As such, the iForest was used to identified the potential outlier data in the obesity, diabetes, and the hypertension datasets.

Basically, the isolation forest works by creating an ensemble of isolation forest trees for each dataset where outliers were as instances with short average length in the trees. The tree is then recursively created by dividing the dataset until all instances are isolated or specific tree height is achieved.

In this project, the isolation forest was implemented using Scikit-Learn Python library version 0.24.2.

After the potential outliers detected by the isolation forest, the outlier data from each dataset were eliminated and the remaining data were used for further modelling algorithms.

Figure 4.3.2a illustrates a sample code snippet for diabetes.

```
##################### Removing Outliers using Isolation Forest ##############################
# summarize the number of the dataset before removing outliers
X_features = diab_df2.drop("CLASS", axis = 1)
y_target = diab_df2['CLASS']

# define and identify outliers in the dataset
iso = IsolationForest(contamination = 0.06, random_state = 1)
iso_pred = iso.fit_predict(X_features)

# to obtain the anomaly score
iso_scores = iso.decision_function(X_features)

# display the anomaly score
sns.distplot(iso_scores, kde = False, color = 'green', bins = 50)
plt.title("Distribution of the Anomaly Scores using Isolation Forest for Diabetes", fontweight = 'bold', fontsize = 14)
plt.xlabel("Average Path Length", fontweight = 'bold', fontsize = 12)
plt.ylabel("Count", fontweight = 'bold', fontsize = 12)
plt.show()

# identify the outliers using the selected threshold value
outliers_index = np.where(iso_scores < -0.01)[0]

# filter out the outliers from the original dataset based on the outliers index
x_features_iso = X_features[~X_features.index.isin(outliers_index)]
y_target_iso = y_target[~y_target.index.isin(outliers_index)]

# convert the x_features_iso and y_target_iso into dataframe
X_features = pd.DataFrame(data = x_features_iso, columns = X_features.columns)
y_target = pd.Series(data = y_target_iso, name = 'CLASS')
```

Figure 4.3.2a: A sample code for outlier's detection using iForest for Diabetes

Figure 4.3.2b illustrates the distribution of the anomaly scores for diabetes.

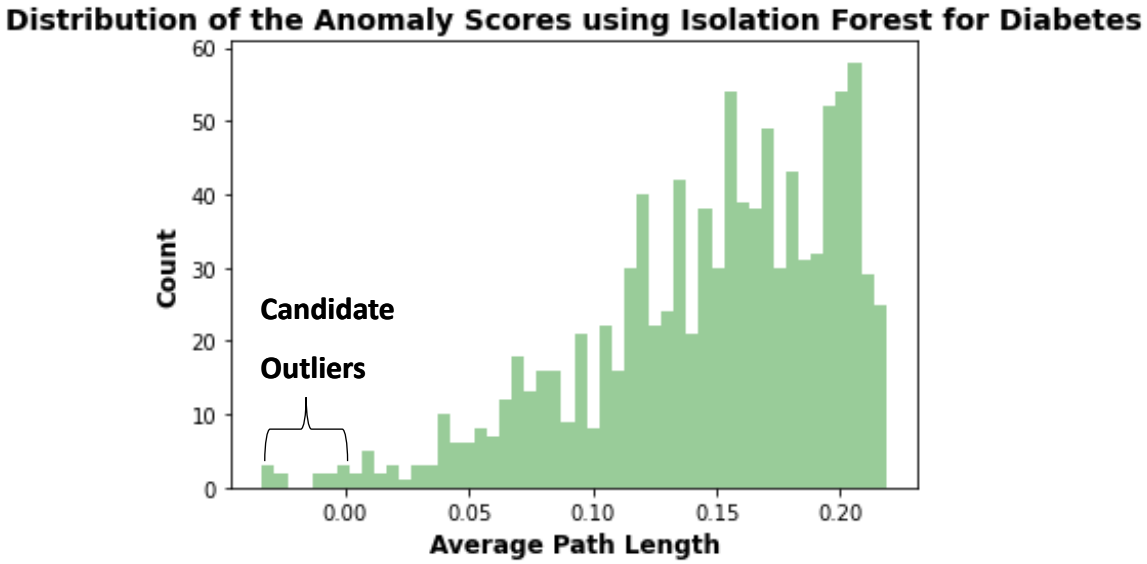


Figure 4.3.2b: The distribution of the anomaly scores using iForest for Diabetes

Figure 4.3.2c illustrates a sample code snippet for hypertension.

```
193  ##################### Removing Outliers using Isolation Forest ###################################
194  # summarize the number of the dataset before removing outliers
195  X_features = bp_df.drop("outcome", axis = 1)
196  y_target = bp_df['outcome']
197
198  iso = IsolationForest(contamination = 0.01, random_state = 1)
199  iso_predict = iso.fit_predict(X_features)
200
201  # to obtain the anomaly score
202  iso_scores = iso.decision_function(X_features)
203
204  # display the anomaly score
205  sns.distplot(iso_scores, kde = False, bins = 50)
206  plt.title("Distribution of the Anomaly Scores using Isolation Forest for Hypertension", fontweight = 'bold', fontsize = 14)
207  plt.xlabel("Average Path Length", fontweight = 'bold', fontsize = 12)
208  plt.ylabel("Count", fontweight = 'bold', fontsize = 12)
209  plt.show()
210
211  # identify the outliers using the selected threshold value
212  outliers_index = np.where(iso_scores < -0.01)[0]
213
214  # filter out the outliers from the original dataset based on the outliers index
215  x_features_iso = X_features[~X_features.index.isin(outliers_index)]
216  y_target_iso = y_target[~y_target.index.isin(outliers_index)]
217
218  # convert the x_features_iso and y_target_iso into dataframe
219  X_features = pd.DataFrame(data = x_features_iso, columns = X_features.columns)
220  y_target = pd.Series(data = y_target_iso, name = 'outcome')
```

Figure 4.3.2c: A sample code for outlier's detection using iForest for Hypertension

Figure 4.3.2d illustrates the distribution of the anomaly scores for hypertension.

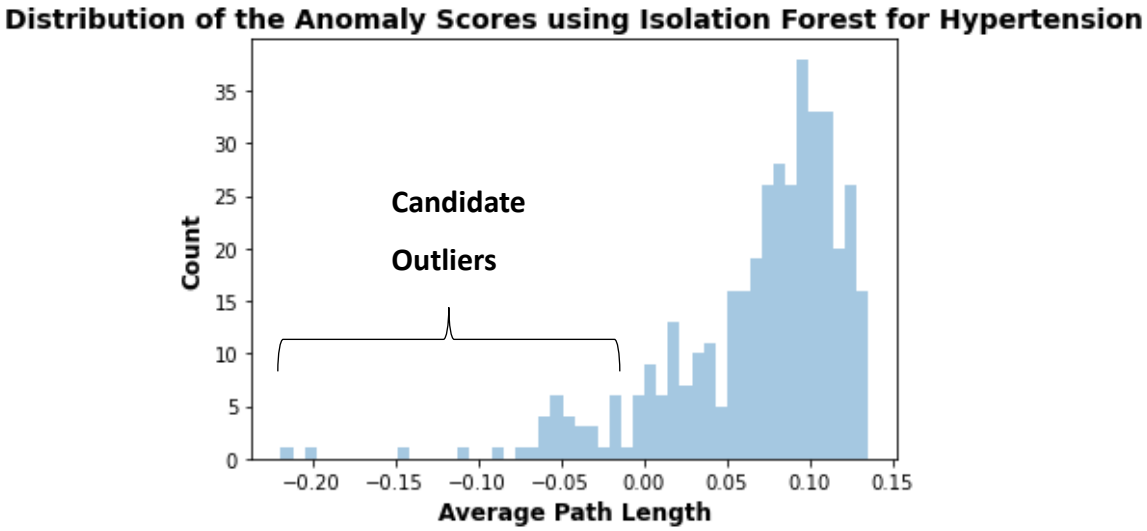


Figure 4.3.2d: The distribution of the anomaly scores using iForest for Hypertension

Figure 4.3.2e illustrates a sample code snippet for obesity.

```
#################### Removing Outliers using Isolation Forest ############################
# summarize the number of the dataset before removing outliers
X_features = obs_df.drop("outcome", axis = 1)
y_target = obs_df['outcome']

# define and identify outliers in the dataset
iso = IsolationForest(contamination = 0.1, random_state = 1)
iso_pred = iso.fit_predict(X_features)

# to obtain the anomaly score
iso_scores = iso.decision_function(X_features)

# display the anomaly score
sns.distplot(iso_scores, kde= False, color = 'crimson', bins = 100)
plt.title("Distribution of the Anomaly Scores using Isolation Forest for Obesity", fontweight = 'bold', fontsize = 14)
plt.xlabel("Average Path Length", fontweight = 'bold', fontsize = 12)
plt.ylabel("Count", fontweight = 'bold', fontsize = 12)
plt.show()

# identify the outliers using the selected threshold value
outliers_index = np.where(iso_scores < -0.01)[0]

# filter out the outliers from the original dataset based on the outliers index
x_features_iso = X_features[~X_features.index.isin(outliers_index)]
y_target_iso = y_target[~y_target.index.isin(outliers_index)]

# convert the x_features_iso and y_target_iso into dataframe
X_features = pd.DataFrame(data = x_features_iso, columns = X_features.columns)
y_target = pd.Series(data = y_target_iso, name = 'outcome')

after_rOutlier = pd.concat([X_features, y_target], axis = 1)
boxplot(after_rOutlier)
```

Figure 4.3.2e: A sample code for outlier's detection using iForest for Obesity

Figure 4.3.2f illustrates the distribution of the anomaly scores for obesity.

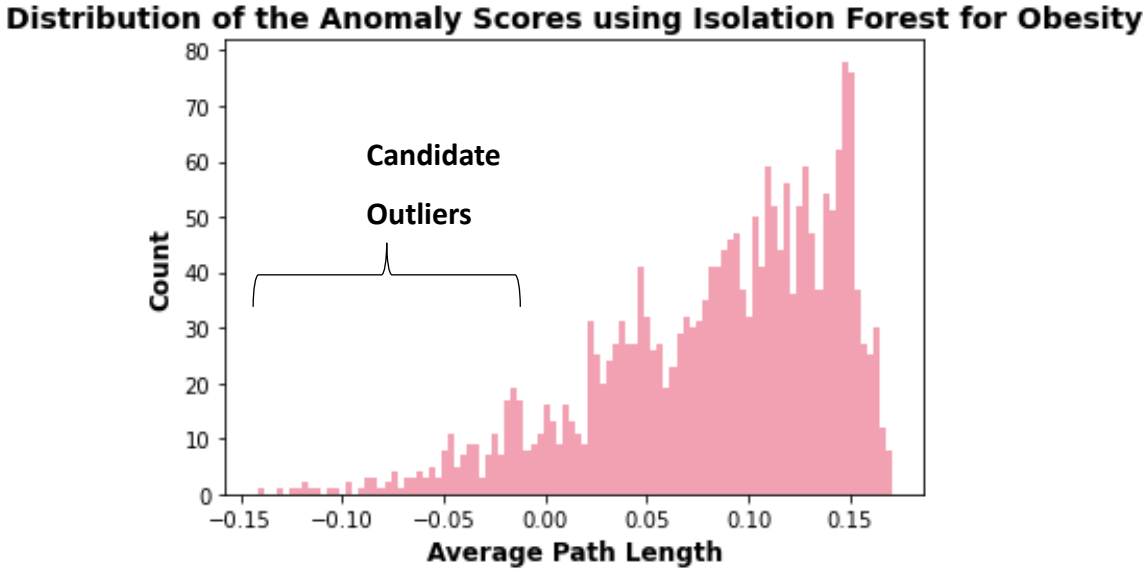


Figure 4.3.2f: The distribution of the anomaly scores using iForest for Obesity

Basically, the outliers with the lower anomaly scores are likely to be outliers. As such, the outlier threshold on the score will be set by the user preference to differentiate between the outlier and inlier. In this case, the outlier threshold will be set as below -0.01 for all data aspects.

4.4 Resampling Techniques for Imbalanced Training Sets

Prior to the resampling and algorithms analysis, all the respective pre-processed datasets were then divided into 70% training set and 30% testing set. Figure 4.4a illustrates a sample code snippet of splitting the pre-processed datasets and was applied to all the respective datasets.

```
228  X_train, X_test, y_train, y_test = train_test_split(X_features,
229                                                      y_target,
230                                                      test_size = 0.30,
231                                                      random_state = 42,
232                                                      stratify = y_target)
```

Figure 4.4a: A sample code of dividing the pre-processed data into 70% training set and 30% testing set

After splitting the datasets, it is also imperative to balance the datasets prior to the training phase. This is because if the imbalanced training data is not treated promptly prior to the training phase, this will degrade the performance of the classifier models. As such, most of the predictions will correspond to the majority class and treat the minority class features as noise in the data and resulting high bias in the models.

Therefore, resampling is one of the most commonly approach to achieve the aim and deal with the imbalanced training datasets.

In this case, the SMOTE-TOMEK resampling technique was chosen as a resampling technique on the imbalanced training datasets because the aims of this technique are to clean the overlapping data points for each of the classes distributed in the sample space. If oversampling is done by SMOTE alone, the class clusters may invade each other space and resulting in the classifier model being overfitting. Likewise, TOMEK links are the opposite paired samples that are closet neighbours to each other. Therefore, the majority of the class observations from these links are removed. Tomek links are applied to oversampled minority class samples performed by SMOTE. All in all, instead of removing the observations only from the majority class, only remove both class observations from TOMEK links.

One must note that only SMOTE-TOMEK technique was applied on the training set as testing set was not part during the training phase.

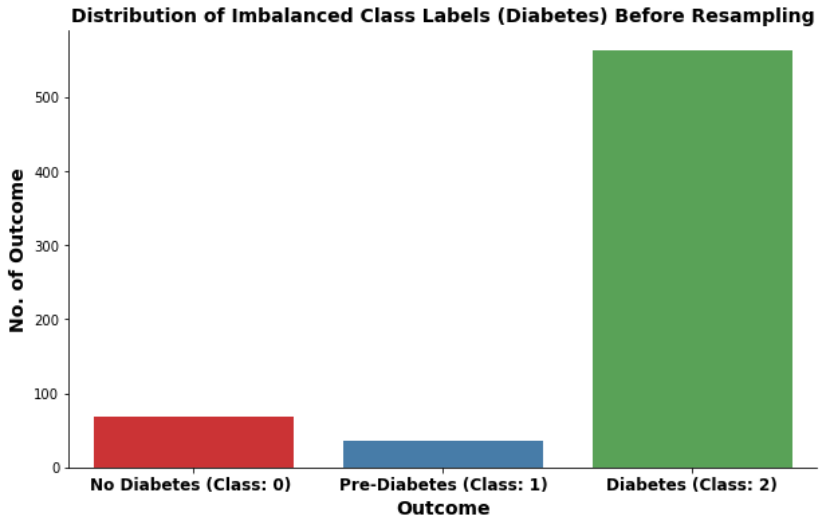
SMOTE-TOMEK technique was implemented using the Imbalanced-learn python library (V0.8.0). Figure 4.4b illustrates a sample code snippet to balance the training imbalanced datasets for all aspects.

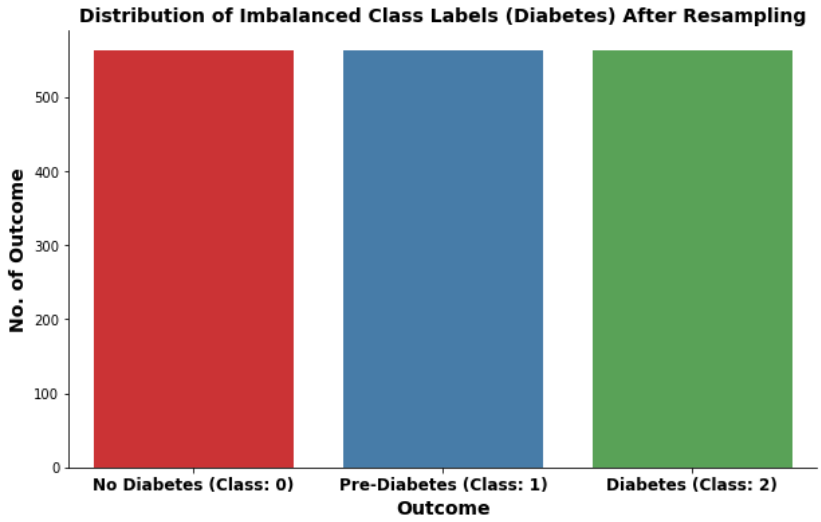
```
75  def balanced_datasets(X, y):
76      oversample_SMT = SMOTETomek(random_state = 1)
77      x_smt, y_smt = oversample_SMT.fit_resample(X, y)
78      return x_smt, y_smt
```

```
269 ##### call the function to balanced the datasets
270 X_smt, Y_smt = balanced_datasets(X_train, y_train)
```

Figure 4.4b: A sample code to balance the training imbalanced datasets using SMOTE-TOMEK technique

Figure 4.4c illustrates the initial minority class distributions **before** and **after** resampling for diabetes training datasets.



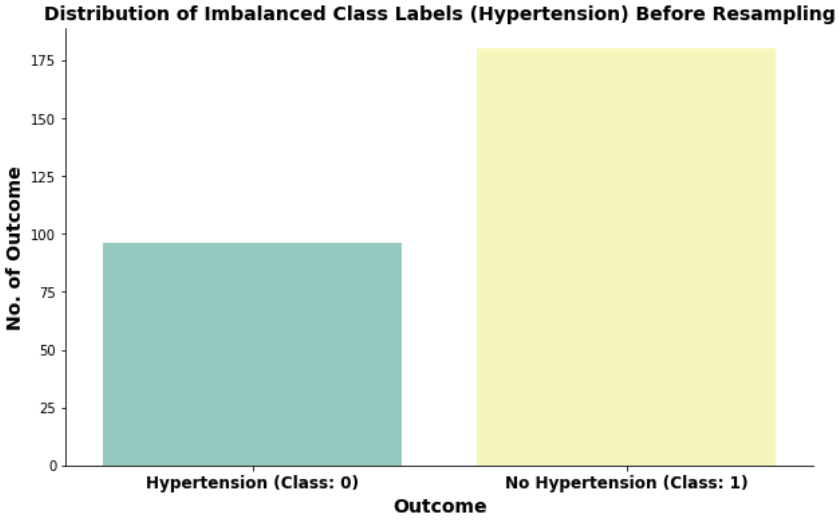


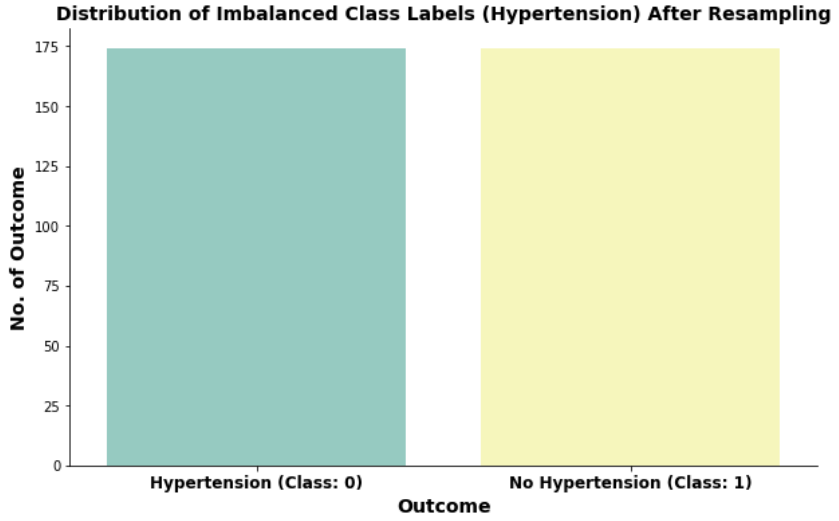
```
Before Resampling for Diabetes using SMOTE-Tomek:
0     68
1     36
2    563
Name: CLASS, dtype: int64


After Resampling for Diabetes using SMOTE-Tomek:
0    563
1    563
2    563
Name: CLASS, dtype: int64
```

Figure 4.4c: Before and After resampling for diabetes training datasets

Figure 4.4d illustrates the initial minority class distributions **before** and **after** resampling for hypertension training datasets.



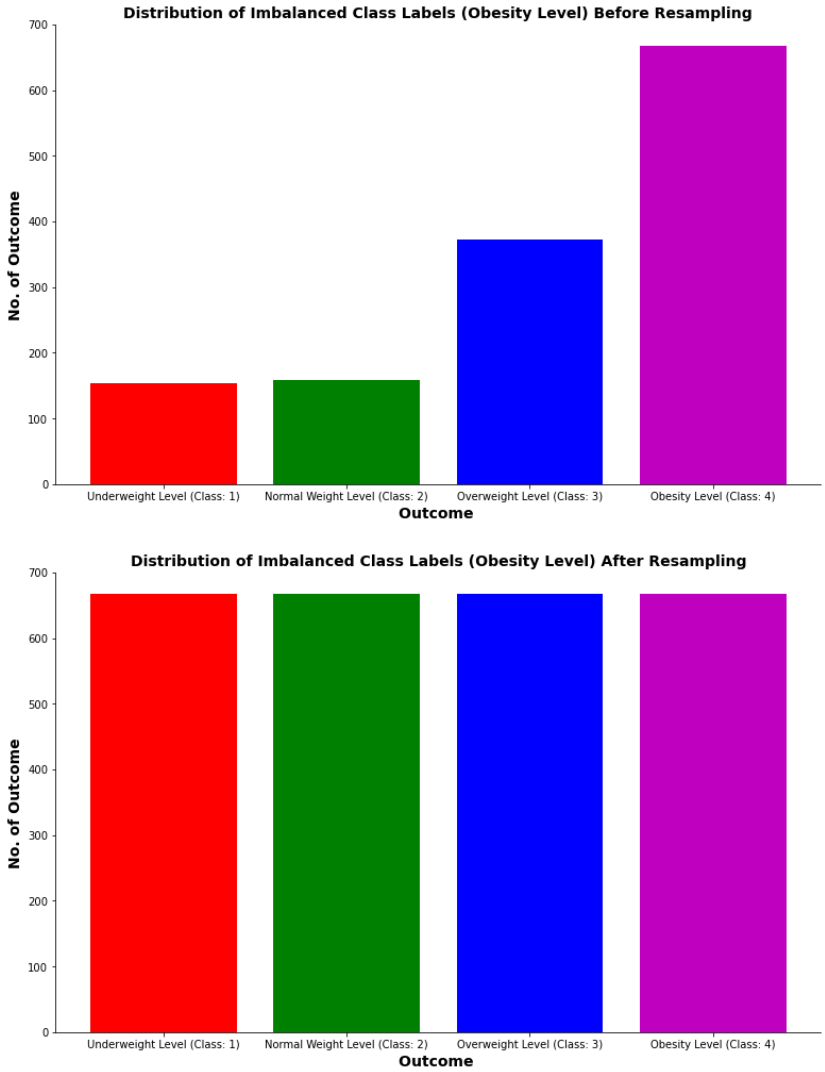


```
Before Resampling for Hypertension using SMOTE-Tomek:
0     96
1    180
Name: outcome, dtype: int64


After Resampling for Hypertension using SMOTE-Tomek:
0    174
1    174
Name: outcome, dtype: int64
```

Figure 4.4d: Before and After resampling for hypertension training datasets

Figure 4.4e illustrates the initial minority class distributions **before** and **after** resampling for obesity training datasets.



```
Before Resampling for Obesity Level using SMOTE-Tomek:
Class = 1, n = 158 (11.695%)
Class = 2, n = 153 (11.325%)
Class = 3, n = 373 (27.609%)
Class = 4, n = 667 (49.371%)

After Resampling for Obesity Level using SMOTE-Tomek:
Class = 1, n = 667 (25.000%)
Class = 2, n = 667 (25.000%)
Class = 3, n = 667 (25.000%)
Class = 4, n = 667 (25.000%)
```

Figure 4.4e: Before and After resampling for obesity training datasets

4.5 Baseline Model Selection and Evaluation via Shuffle-Split CV

Prior to the model building, the potential outlier data for the respective datasets (i.e., diabetes, hypertension, and obesity) were eliminated by the Isolation Forest algorithm as outlier detection, and the SMOTE-TOMEK was implemented to achieve the balance training datasets. Followed by, the classifier machine learning algorithms such as random forest, gradient boosting, and extra trees were implemented using Scikit-learn V0.24.2 and were used to learn from the respective training datasets.

Furthermore, the **baseline** of the classifier machine learning algorithms were built with the **default parameters** to investigate how well does each algorithm performed with the respective medical disease training datasets. Figure 4.5a illustrates a sample code of the baseline models.

```
311  # initialize the ensemble classifier models with their default parameters and add them into a model list
312  classifier_models = [('Random Forest', RandomForestClassifier(random_state = 42)),
313                       ('Gradient Boosting', GradientBoostingClassifier(random_state = 42)),
314                       ('Extra Trees', ExtraTreesClassifier(random_state = 42))]
```

Figure 4.5a: A sample code of the baseline models using the default parameters

Prior to the shuffle-split cross-validation, a pipeline object was built by providing it with a list of steps. Each step represents a tuple containing a string of name and an instance of an estimator.

For the hypertension training set, the first step was created with a ‘scaler’, which is an instance of MinMaxScaler (), followed by a ‘power’, is an instance of PowerTransformer (), and the final step was created with a ‘model’, is an instance of the respective classifier algorithms as mentioned above. In other words, each pipeline defines one (or more) data preparation techniques such as MinMaxScaler or StandardScaler, PowerTransformer, and ends with a defined model that takes the transformed training data as input.

The main objective of using the Power Transformer () was to change the distribution to be more alike of the Gaussian. Basically, the Power Transformer () performed the Yeo-Johnson transformation and automatically determined the best parameters that applied based on the dataset, e.g., how to best make each variable more Gaussian. Furthermore, the transformer will also standardize the dataset as part of the transform. However, the power transform may

make use of the log () function, which does not really work for zero values. As such, a MinMaxScaler () function was included to scaled the dataset prior to the power transformer.

Figure 4.5b illustrates a sample code snippet of the model pipeline for **hypertension** training set.

```
330     # define pipeline
331     pipe = Pipeline(steps = [('scaler', MinMaxScaler()),('power', PowerTransformer()), ('model', models)])
```

Figure 4.5b: A sample code of the model pipeline for hypertension training set

The model pipeline for obesity and diabetes can be created a similar process and achieve the same objective as hypertension, such that the first step was created with a 'scaler' of StandardScaler (), followed by created with a 'model', of the respective classifier algorithms. Figure 4.5c illustrates a sample code snippet of the model pipeline for both **obesity** and **diabetes** training set.

```
348     # build pipeline
349     pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', models)])
```

Figure 4.5c: A sample code of the model pipeline for both obesity and diabetes training set

After the baseline model pipelines were implemented, the **Shuffle-Split Cross Validation** was implemented to assess the performance and general errors of the entire classification algorithms. This cross-validation technique has been widely used and is one of the preferred validation techniques in machine learning as compared to the conventional split-sample approach (i.e., train-test split).

Shuffle-Split Cross-Validation allows for control over the number of iterations independently of the training and test sizes and also allows for using the only part of the data in each iteration, by providing the train size and test size setting that don't add up to one. Therefore, shuffle-split cross-validation provides a more reliable result for classification tasks. For this case, the train size and test size has been set as 80% and 20% respectively. One must note that the test used in the cross-validation was not originated from the testing set. In other words, the test used in the cross-validation was actually the % split from the training set itself. Figure 4.5d illustrate a set of the sample code to evaluate each

of the classification machine learning algorithms via shuffle-split cross validation with model pipeline.

A) For Diabetes:

```
82  def cross_validation_eval(model, X_smt, Y_smt):
83
84      shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
85      cv_result = cross_validate(model,
86                                 X_smt,
87                                 Y_smt,
88                                 cv = shuffle_split,
89                                 n_jobs = 1,
90                                 scoring = ('accuracy', 'precision_macro', 'recall_macro', 'f1_macro'),
91                                 return_train_score = True)
92
93
94      # compute the ROC scores
95      myscore = make_scorer(roc_auc_score, multi_class='ovo',needs_proba=True)
96      roc_scores = cross_validate(model,
97                                  X_smt,
98                                  Y_smt,
99                                  cv = shuffle_split,
100                                 n_jobs = 1,
101                                 scoring = myscore,
102                                 return_train_score = True)
103
104     return cv_result, roc_scores
```

```
342 for name, models in classifier_models:
343
344     # build pipeline
345     pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', models)])
346
347     # call the function to perform cross-validation evaluation
348     cvScores, rocScores = cross_validation_eval(pipe, X_smt, Y_smt)
```

B) For Hypertension:

```
81  def cross_validation_eval(pipe, X_smt, Y_smt):
82
83      shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
84      cv_result = cross_validate(pipe,
85                                 X_smt,
86                                 Y_smt,
87                                 cv = shuffle_split,
88                                 n_jobs = 1,
89                                 scoring = ('accuracy', 'precision_macro', 'recall_macro', 'f1_macro', 'roc_auc'),
90                                 return_train_score = True,
91                                 return_estimator = True)
92      return cv_result
```

```
328 for name, models in classifier_models:
329
330     # define pipeline
331     pipe = Pipeline(steps = [('scaler', MinMaxScaler()),('power', PowerTransformer()), ('model', models)])
332
333
334     # call the function to perform cross validation evaluation
335     cvScores = cross_validation_eval(pipe, X_smt, Y_smt)
```

C) For Obesity

```
def cross_validation_eval(model, X_smt, Y_smt):

    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)

    # compute the accuracy, precision, recall and f1-scores
    cv_result = cross_validate(model,
                               X_smt,
                               Y_smt,
                               cv = shuffle_split,
                               n_jobs = 1,
                               scoring = ('accuracy', 'precision_macro', 'recall_macro', 'f1_macro'),
                               return_train_score = True)

    # compute the ROC scores
    myscore = make_scorer(roc_auc_score, multi_class='ovo',needs_proba=True)
    roc_scores = cross_validate(model,
                                X_smt,
                                Y_smt,
                                cv = shuffle_split,
                                n_jobs = 1,
                                scoring = myscore,
                                return_train_score = True)
    return cv_result, roc_scores
```

```
for name, models in classifier_models:

    # build pipeline
    pipe = Pipeline(steps = [('scaler', StandardScaler()), ('model', models)])

    # call function to compute cross validation evaluation
    cvScores, rocScores = cross_validation_eval(pipe, X_smt, Y_smt)
```

Figure 4.5d: A sample code of Shuffle-Split Cross Validation with Model Pipeline

The following show the outcomes of the performance evaluation for the respective **classification machine learning algorithms (baseline)** for the respective medical diseases.

A) For Diabetes

Cross-Validation Scores for Diabetes:

	Random Forest	Gradient Boosting	Extra Trees
Train Accuracy	0.928201	0.983124	0.725093
Validate Accuracy	0.918935	0.971598	0.701183
Train Precision	0.929950	0.983575	0.748202
Validate Precision	0.920668	0.972481	0.729633
Train Recall	0.928091	0.982962	0.725264
Validate Recall	0.919798	0.972397	0.700724
Train F1	0.927649	0.982974	0.729448
Validate F1	0.919090	0.971457	0.704465
Train ROC	0.968148	0.997788	0.909537
Validate ROC	0.960816	0.993452	0.896641

Figure 4.5e: Shuffle-Split Cross-Validation Scores for Diabetes

Classification Report for Random Forest

	precision	recall	f1-score	support
No Diabetes	0.96	1.00	0.98	563
Pre-Diabetes	0.96	0.86	0.90	563
Diabetes	0.87	0.92	0.89	563
accuracy			0.92	1689
macro avg	0.93	0.92	0.92	1689
weighted avg	0.93	0.92	0.92	1689

Figure 4.5f: Classification Report for Random Forest (Diabetes)

Classification Report for Gradient Boosting

	precision	recall	f1-score	support
No Diabetes	0.96	1.00	0.98	563
Pre-Diabetes	0.98	0.99	0.99	563
Diabetes	0.99	0.94	0.97	563
accuracy			0.98	1689
macro avg	0.98	0.98	0.98	1689
weighted avg	0.98	0.98	0.98	1689

Figure 4.5g: Classification Report for Gradient Boosting (Diabetes)

Classification Report for Extra Trees

	precision	recall	f1-score	support
No Diabetes	0.59	0.78	0.67	563
Pre-Diabetes	0.66	0.55	0.60	563
Diabetes	0.99	0.86	0.92	563
accuracy			0.73	1689
macro avg	0.75	0.73	0.73	1689
weighted avg	0.75	0.73	0.73	1689

Figure 4.5h: Classification Report for Extra Trees (Diabetes)

B) For Hypertension

Cross-Validation Scores for Hypertension:

	Random Forest	Gradient Boosting	Extra Trees
Train Accuracy	1.000000	1.000000	1.000000
Validate Accuracy	0.971429	0.974286	0.954286
Train Precision	1.000000	1.000000	1.000000
Validate Precision	0.971179	0.974958	0.955446
Train Recall	1.000000	1.000000	1.000000
Validate Recall	0.971604	0.974040	0.952646
Train F1	1.000000	1.000000	1.000000
Validate F1	0.970803	0.973696	0.952981
Train ROC	1.000000	1.000000	1.000000
Validate ROC	0.983561	0.979181	0.984008

Figure 4.5i: Shuffle-Split Cross-Validation Scores for Hypertension

Classification Report for Random Forest (Hypertension)

	precision	recall	f1-score	support
No Hypertension	0.95	0.98	0.96	174
Hypertension	0.98	0.95	0.96	174
accuracy			0.96	348
macro avg	0.96	0.96	0.96	348
weighted avg	0.96	0.96	0.96	348

Figure 4.5j: Classification Report for Random Forest (Hypertension)

Classification Report for Gradient Boosting (Hypertension)

	precision	recall	f1-score	support
No Hypertension	0.95	0.97	0.96	174
Hypertension	0.96	0.95	0.96	174
accuracy			0.96	348
macro avg	0.96	0.96	0.96	348
weighted avg	0.96	0.96	0.96	348

Figure 4.5k: Classification Report for Gradient Boosting (Hypertension)

Classification Report for Extra Trees (Hypertension)

	precision	recall	f1-score	support
No Hypertension	0.94	0.97	0.96	174
Hypertension	0.97	0.94	0.96	174
accuracy			0.96	348
macro avg	0.96	0.96	0.96	348
weighted avg	0.96	0.96	0.96	348

Figure 4.5l: Classification Report for Extra Trees (Hypertension)

C) For Obesity

Cross-Validation Scores for Obesity:

	Random Forest	Gradient Boosting	Extra Trees
Train Accuracy	1.000000	1.000000	1.000000
Validate Accuracy	0.989888	0.991011	0.976779
Train Precision	1.000000	1.000000	1.000000
Validate Precision	0.989712	0.990865	0.976325
Train Recall	1.000000	1.000000	1.000000
Validate Recall	0.989671	0.991119	0.976314
Train F1	1.000000	1.000000	1.000000
Validate F1	0.989662	0.990931	0.976187
Train ROC	1.000000	1.000000	1.000000
Validate ROC	0.999657	0.999787	0.998888

Figure 4.5m: Shuffle-Split Cross-Validation Scores for Obesity

Classification Report for Random Forest

	precision	recall	f1-score	support
Underweight	0.99	1.00	0.99	667
Normal Weight	0.97	0.98	0.98	667
Overweight	0.97	0.98	0.98	667
Obesity	1.00	0.99	0.99	667
accuracy			0.99	2668
macro avg	0.99	0.99	0.99	2668
weighted avg	0.99	0.99	0.99	2668

Figure 4.5n: Classification Report for Random Forest (Obesity)

Classification Report for Gradient Boosting

	precision	recall	f1-score	support
Underweight	0.99	0.99	0.99	667
Normal Weight	0.99	0.98	0.98	667
Overweight	0.98	0.99	0.99	667
Obesity	1.00	0.99	1.00	667
accuracy			0.99	2668
macro avg	0.99	0.99	0.99	2668
weighted avg	0.99	0.99	0.99	2668

Figure 4.5o: Classification Report for Gradient Boosting (Obesity)

Classification Report for Extra Trees

	precision	recall	f1-score	support
Underweight	0.99	1.00	0.99	667
Normal Weight	0.96	0.96	0.96	667
Overweight	0.96	0.96	0.96	667
Obesity	1.00	0.98	0.99	667
accuracy			0.97	2668
macro avg	0.97	0.97	0.97	2668
weighted avg	0.97	0.97	0.97	2668

Figure 4.5p: Classification Report for Extra Trees (Obesity)

Based on the above observation, a **random forest classifier** was selected as the baseline model for both **obesity and hypertension** because of its model characteristics such that it is less biased and exhibits moderate variance in nature. Furthermore, the training and validation scores for all performance metrics showing not much difference with the rest of the classifiers, and the ROC scores also revealed that the random forest classifier is capable of distinguishing between classes. On the other hand, the **gradient boosting classifier** was selected as the baseline model for **diabetes** as it shows the only best performance metrics as compared to the rest of the other classifiers.

The above-selected classifier baseline models will proceed with the hyperparameter optimization process, which will be discussed in the next section, it is to achieve the best hyperparameters that maximize the predictive performance.

4.6 Hyperparameters Optimization for Proposed Baseline Model

Similar as previous section, a pipeline object was built by providing it with a list of steps. Each steps represents a tuple containing a string of name and an instance of an estimator.

For example, in hypertension, the first step was created with a ‘scaler’, which is an instance of MinMaxScaler (), followed by a ‘power’, is an instance of PowerTransformer (), and the final step was created with a ‘rfc, is an instance of the random forest classifier. As mentioned earlier, each pipeline defines one (or more) data preparation techniques such as MinMaxScaler or StandardScaler, PowerTransformer, and ends with a defined model that takes the transformed training data as input. Figure 4.6a illustrates a sample code snippet of the model pipeline for hypertension.

```
#build pipeline
pipe = Pipeline([('scaler', MinMaxScaler()), ('power', PowerTransformer()),("rfc", RandomForestClassifier(random_state = 42,
                                                                                max_leaf_nodes = None,
                                                                                max_features = "sqrt"))])
```

Figure 4.6a: A sample code of the model pipeline for hypertension

For both obesity and diabetes, it can be created a similar process and achieve the same objective as hypertension, such that the first step was created with a ‘scaler’ of StandardScaler (), followed by created with a ‘model’, of the respective classifier algorithms. Figure 4.6b and Figure 4.6c illustrates a sample code snippet of the model pipeline for both **obesity** and **diabetes** training set.

```
# #build pipeline
pipe = Pipeline([('scaler', StandardScaler()),("gbc", GradientBoostingClassifier(random_state = 42,
                                                max_leaf_nodes = None))])
```

Figure 4.6b: A sample code of the model pipeline for diabetes

```
# build pipeline
pipe = Pipeline([('scaler', StandardScaler()),("rfc", RandomForestClassifier(random_state = 42,
                                                max_leaf_nodes = None,
                                                max_features = 'sqrt'))])
```

Figure 4.6c: A sample code of the model pipeline for obesity

The syntax to define a parameter grid for a pipeline is to specify for each parameter the step name, followed by __(a double underscore), followed by the parameter name. To search over the n_estimator parameter, for example, in the random forest classifier, the

"rfc__n_estimators" was used as key in the parameter grid dictionary, and similar for the rest of the other parameters. Figure 4.6d, Figure 4.6e, and Figure 4.6f illustrate the sample code of construction of the parameter grid for the respective medical disease.

```
401  # set parameter grid
402  param_grid = {'gbc__n_estimators': [50, 100, 150, 200, 250],
403                'gbc__max_depth': np.arange(1,5),
404                'gbc__learning_rate': [0.001, 0.01, 0.1]}
```

Figure 4.6d: A sample code of the parameter grid for Gradient Boosting Classifier (Diabetes)

```
391  # set parameter grid
392  param_grid = {'rfc__n_estimators': [50, 100, 150, 200, 250, 300, 350],
393                'rfc__criterion': ['gini', 'entropy'],
394                'rfc__max_depth': np.arange(5, 10),
395                'rfc__min_samples_leaf': np.arange(2, 7)}
```

Figure 4.6e: A sample code of the parameter grid for Random Forest Classifier (Hypertension)

```
401  # set parameter grid
402  param_grid = {'rfc__n_estimators': [80, 100, 150, 200, 250, 300],
403                'rfc__max_depth': np.arange(2, 7),
404                'rfc__criterion': ['gini', 'entropy'],
405                'rfc__min_samples_split': np.arange(2, 5)}
```

Figure 4.6f: A sample code of the parameter grid for Random Forest Classifier (Obesity)

In the hyperparameter optimization process, a GridSearchCV function was implemented to search the best or optimal hyperparameters based on the best scores. For each split in the cross-validation, the MinMaxScaler () or the StandardScaler () is refit only with the training splits and no information was leaked from the testing split into the parameter search. As mentioned before, the testing split was not based on the testing set. It is the defined amount of the % split from the training set. As such, all the respective test sets were not seen during the training phase as well as the hyperparameter optimization process. Furthermore, to get a finer control over the cross-validation, the CV parameter in the GridSearchCV process uses Shuffle-Split and the amount of split were defined as 80% training set and 20% testing (or validation) set.

The main important parameters of **gradient boosted tree models** to be optimized are the number of trees, **n_estimators**, and the **learning_rate**, which controls the degree to which each tree is allowed to correct the mistakes of the previous trees. These two parameters are highly interconnected such that a lower learning_rate means that more trees are required to build a model in more complexity. Similar to the random forests, a higher n_estimators are always better. But however, increasing n_estimators in gradient boosting will leads to a model complexity, which may lead to overfitting. In addition, another important parameter is max_depth, it is to minimizes the complexity of each tree. Usually, max_depth is set to a very low split in gradient boosted models, often no deeper than five splits.

Figure 4.6g illustrates the sample code of the entire hyperparameter optimization process for Gradient Boosting Classifier for diabetes.

```
def hyperparameters_tuning(pipe, param_grid, X, y):

    # split the dataset
    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
    # perform hyperparameter tuning using gridsearchcv
    gs_res = GridSearchCV(pipe, param_grid = param_grid,
                          cv = shuffle_split, scoring = 'accuracy', verbose = 3)
    # fitted the model for grid search
    gs_res.fit(X, y)
    return gs_res
```

```
##################### Hyperparameter Optimization on Gradient Boosting ########################
# #build pipeline
pipe = Pipeline([('scaler', StandardScaler()),("gbc", GradientBoostingClassifier(random_state = 42,
                                            max_leaf_nodes = None))])

# set parameter grid
param_grid = {'gbc__n_estimators': [50, 100, 150, 200, 250],
              'gbc__max_depth': np.arange(1,5),
              'gbc__learning_rate': [0.001, 0.01, 0.1]}

# call the function to perform hyperparameter tuning
gridResult = hyperparameters_tuning(pipe, param_grid, X_smt, Y_smt)

# display the best parameters after tuning
print("\nHyperparameter Optimization Process for Gradient Boosting Classifer (Diabetes): ")
print("================================================================================")
print("The best score is {}".format(gridResult.best_score_))
print("The best params is {}".format(gridResult.best_params_))
print("The best estimator is {}".format(gridResult.best_estimator_))

# store the best params into Series
best_params_df = pd.Series(gridResult.best_params_)
print(best_params_df)
```

Figure 4.6g: A sample code of the hyperparameter optimization process for Gradient Boosting Classifier (Diabetes)

The important parameters to be optimized in **random forest tree models** are **n_estimators, criterion, max_depth, and min_samples_leaf. For n_estimators**, the larger is always better, and averaging more trees will yield a more robust ensemble by minimizing overfitting. However, there are some drawbacks, more trees need more memory and more time to train. In general, a **smaller max_features** reduce overfitting and it is a good rule of thumb to use the default values for max_features: max_features=sqrt(n_features) for classification problems. Furthermore, adding min_samples_leaf might sometimes also improve performance.

Figure 4.6h and Figure 4.6i illustrates the sample code of the entire hyperparameter optimization process for Random Forest Classifier for hypertension and obesity respectively.

```
def hyperparameters_tuning(pipe, param_grid, X, y):

    # split the dataset
    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
    # perform hyperparameter tuning using gridsearchcv
    gs_res = GridSearchCV(pipe, param_grid = param_grid,
                          cv = shuffle_split, scoring = 'accuracy', verbose = 3)
    # fitted the model for grid search
    gs_res.fit(X, y)
    return gs_res
```

```
##################### Hyperparameter Optimization on Random Forest Classifier ########################
#build pipeline
pipe = Pipeline([('scaler', MinMaxScaler()), ('power', PowerTransformer()),("rfc", RandomForestClassifier(random_state = 42,
                                                                                max_leaf_nodes = None,
                                                                                max_features = "sqrt"))])


# set parameter grid
param_grid = {'rfc__n_estimators': [50, 100, 150, 200, 250, 300, 350],
              'rfc__criterion': ['gini', 'entropy'],
              'rfc__max_depth': np.arange(5, 10),
              'rfc__min_samples_leaf': np.arange(2, 7)}


# call the function to perform hyperparameter tuning
gridResult = hyperparameters_tuning(pipe, param_grid, X_smt, Y_smt)

# display the best parameters after tuning
print("\nHyperparameter Optimization Process for Random Forest Classifier (Hypertension): ")
print("================================================================================")
print("The best score is {}".format(gridResult.best_score_))
print("The best params is {}".format(gridResult.best_params_))
print("The best estimator is {}".format(gridResult.best_estimator_))

# store the best params into Series
best_params_df = pd.Series(gridResult.best_params_)
print(best_params_df)
```

Figure 4.6h: A sample code of the hyperparameter optimization process for Random Forest Classifier (Hypertension)

```python
def hyperparameters_tuning(pipe, param_grid, X, y):
    # split the dataset
    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
    # perform hyperparameter tuning using gridsearchcv
    gs_res = GridSearchCV(pipe, param_grid = param_grid,
                          cv = shuffle_split, scoring = 'accuracy', verbose = 3)
    # fitted the model for grid search
    gs_res.fit(X, y)
    return gs_res
```

```python
################ Hyperparameter Optimization on Random Forest ########################
# build pipeline
pipe = Pipeline([('scaler', StandardScaler()),("rfc", RandomForestClassifier(random_state = 42,
                                                max_leaf_nodes = None,
                                                max_features = 'sqrt'))])

# set parameter grid
param_grid = {'rfc__n_estimators': [80, 100, 150, 200, 250, 300],
              'rfc__max_depth': np.arange(2, 7),
              'rfc__criterion': ['gini', 'entropy'],
              'rfc__min_samples_split': np.arange(2, 5)}

# call the function to perform hyperparameter tuning
gridResult = hyperparameters_tuning(pipe, param_grid, X_smt, Y_smt)

# display the best parameters after tuning
print("\nHyperparameter Optimization Process for Random Forest Classifer (Obesity): ")
print("================================================================================")
print("The best score is {}".format(gridResult.best_score_))
print("The best params is {}".format(gridResult.best_params_))
print("The best estimator is {}".format(gridResult.best_estimator_))

# store the best params into Series
best_params_df = pd.Series(gridResult.best_params_)
print(best_params_df)
```

Figure 4.6i: A sample code of the hyperparameter optimization process for Random Forest Classifier (Obesity)

As mentioned earlier, the hyperparameter optimization process derived the hyperparameters with the best scores. Figure 4.6j, Figure 4.6k, and Figure 4.6l illustrate the outcome of the hyperparameter optimization process for all the respective classifier models.

```
Hyperparameter Optimization Process for Gradient Boosting Classifer (Diabetes):
================================================================================================
The best score is 0.9863905325443787
The best params is {'gbc__learning_rate': 0.1, 'gbc__max_depth': 3, 'gbc__n_estimators': 100}
The best estimator is Pipeline(steps=[('scaler', StandardScaler()),
                ('gbc', GradientBoostingClassifier(random_state=42))])
```

Figure 4.6j: The outcome of the hyperparameter optimization process for Gradient Boosting Classifier (Diabetes)

```
Hyperparameter Optimization Process for Random Forest Classifier (Hypertension):
================================================================================
The best score is 0.9742857142857144
The best params is {'rfc__criterion': 'gini', 'rfc__max_depth': 6, 'rfc__min_samples_leaf': 2, 'rfc__n_estimators': 150}
The best estimator is Pipeline(steps=[('scaler', MinMaxScaler()), ('power', PowerTransformer()),
                ('rfc',
                 RandomForestClassifier(max_depth=6, max_features='sqrt',
                                        min_samples_leaf=2, n_estimators=150,
                                        random_state=42))])
```

Figure 4.6k: The outcome of the hyperparameter optimization process for Random Forest Classifier (Hypertension)

```
Hyperparameter Optimization Process for Random Forest Classifer (Obesity):
================================================================================
The best score is 0.9573033707865168
The best params is {'rfc__criterion': 'entropy', 'rfc__max_depth': 6, 'rfc__min_samples_split': 2, 'rfc__n_estimators':
300}
The best estimator is Pipeline(steps=[('scaler', StandardScaler()),
                ('rfc',
                 RandomForestClassifier(criterion='entropy', max_depth=6,
                                        max_features='sqrt', n_estimators=300,
                                        random_state=42))])
```

Figure 4.6l: The outcome of the hyperparameter optimization process for Random Forest Classifier (Obesity)

4.7 Retrained Model with Best Hyperparameter and Evaluation

The selected baseline classifier models (from the previous section) for each respective medical disease were re-trained with the best hyperparameters achieved during the hyperparameter optimization process.

Similar to the baseline model building and evaluation, the performance of the re-trained model with the best hyperparameters were evaluated via the shuffle-split cross-validation technique.

Figure 4.7a, Figure 4.7b, and Figure 4.8c illustrate the sample codes for the re-trained classifier models as well as the evaluation of the performance metrics.

```python
def cross_validation_eval(model, X_smt, Y_smt):

    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
    cv_result = cross_validate(model,
                               X_smt,
                               Y_smt,
                               cv = shuffle_split,
                               n_jobs = 1,
                               scoring = ('accuracy', 'precision_macro', 'recall_macro', 'f1_macro'),
                               return_train_score = True)


    # compute the ROC scores
    myscore = make_scorer(roc_auc_score, multi_class='ovo',needs_proba=True)
    roc_scores = cross_validate(model,
                                X_smt,
                                Y_smt,
                                cv = shuffle_split,
                                n_jobs = 1,
                                scoring = myscore,
                                return_train_score = True)

    return cv_result, roc_scores
```

```python
###################### Rebuild Model (Gradient Boosting) using Best Parameters #################

# From Hyperparameter Tuning:
# learning rate = 0.001, max_depth = 2,  n_estimators = 50

gbc_model = GradientBoostingClassifier(random_state = 42,
                                       max_leaf_nodes = None,
                                       n_estimators = int(best_params_df.loc['gbc__n_estimators']),
                                       max_depth = int(best_params_df.loc['gbc__max_depth']),
                                       learning_rate = best_params_df.loc['gbc__learning_rate'])


###################### Evaluate the Rebuild Model - Gradient Boosting ########################
pipe_gbc = Pipeline([('scaler', StandardScaler()),  ('gbc', gbc_model)])
gbc_cv_eval, gbc_rocScores = cross_validation_eval(pipe_gbc, X_smt, Y_smt)

gbc_cv_results = pd.DataFrame({
                               'Train Accuracy': gbc_cv_eval['train_accuracy'].mean(),
                               'Validate Accuracy': gbc_cv_eval['test_accuracy'].mean(),
                               'Train Precision': gbc_cv_eval['train_precision_macro'].mean(),
                               'Validate Precision': gbc_cv_eval['test_precision_macro'].mean(),
                               'Train Recall': gbc_cv_eval['train_recall_macro'].mean(),
                               'Validate Recall': gbc_cv_eval['test_recall_macro'].mean(),
                               'Train F1': gbc_cv_eval['train_f1_macro'].mean(),
                               'Validate F1': gbc_cv_eval['test_f1_macro'].mean(),
                               'Train ROC': gbc_rocScores['train_score'].mean(),
                               'Validate ROC': gbc_rocScores['test_score'].mean()}, index = ['Gradient Boosting']).T


# display the cross-validation scores before and after hyperparameter optimization
print("\nCross-Validation Scores (Before Hyperparameter Optimization) for Gradient Boosting Classifier (Diabetes): ")
print(baseline_models_CV_eval['Gradient Boosting'])
print("\nCross-Validation Scores (After Hyperparameter Optimization) for Gradient Boosting Classifier (Diabetes): ")
print(gbc_cv_results)
```

Figure 4.7a: A sample code of the re-trained and re-evaluated of the Gradient Boosting Classifier with the Best Hyperparameters (Diabetes)

```python
def cross_validation_eval(pipe, X_smt, Y_smt):

    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
    cv_result = cross_validate(pipe,
                               X_smt,
                               Y_smt,
                               cv = shuffle_split,
                               n_jobs = 1,
                               scoring = ('accuracy', 'precision_macro', 'recall_macro', 'f1_macro', 'roc_auc'),
                               return_train_score = True,
                               return_estimator = True)
    return cv_result
```

```python
###################### Rebuild Model (Random Forest) using Best Parameters #################

# From Hyperparameter Tuning:
# criterion = gini, max_depth = 5,  n_estimators = 50, min_samples_leaf = 2

rfc_model = RandomForestClassifier(random_state = 42,
                                   max_leaf_nodes = None,
                                   max_features = "sqrt",
                                   n_estimators = int(best_params_df.loc['rfc__n_estimators']),
                                   criterion = best_params_df.loc['rfc__criterion'],
                                   max_depth = int(best_params_df.loc['rfc__max_depth']),
                                   min_samples_leaf = int(best_params_df.loc['rfc__min_samples_leaf']))


###################### Evaluate the Rebuild Model - Random Forest ########################
pipe_rfc = Pipeline([('scaler', MinMaxScaler()),('power', PowerTransformer()), ('rfc', rfc_model)])
rfc_cv_eval = pd.DataFrame(cross_validation_eval(pipe_rfc, X_smt, Y_smt))

rfc_cv_results = pd.DataFrame({
                               'Train Accuracy': rfc_cv_eval['train_accuracy'].mean(),
                               'Validate Accuracy': rfc_cv_eval['test_accuracy'].mean(),
                               'Train Precision': rfc_cv_eval['train_precision_macro'].mean(),
                               'Validate Precision': rfc_cv_eval['test_precision_macro'].mean(),
                               'Train Recall': rfc_cv_eval['train_recall_macro'].mean(),
                               'Validate Recall': rfc_cv_eval['test_recall_macro'].mean(),
                               'Train F1': rfc_cv_eval['train_f1_macro'].mean(),
                               'Validate F1': rfc_cv_eval['test_f1_macro'].mean(),
                               'Train ROC': rfc_cv_eval['train_roc_auc'].mean(),
                               'Validate ROC': rfc_cv_eval['test_roc_auc'].mean()}, index = ['Random Forest']).T



# display the cross-validation scores before and after hyperparameter optimization
print("\nCross-Validation Scores (Before Hyperparameter Optimization) for Random Forest Classifier (Hypertension): ")
print(baseline_models_CV_eval['Extra Trees'])
print("\nCross-Validation Scores (After Hyperparameter Optimization) for Random Forest Classifier (HYpertension): ")
print(rfc_cv_results)
```

Figure 4.7b: A sample code of the re-trained and re-evaluated of the Random Forest Classifier with the Best Hyperparameters (Hypertension)

```python
def cross_validation_eval(model, X_smt, Y_smt):

    shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)

    # compute the accuracy, precision, recall and f1-scores
    cv_result = cross_validate(model,
                               X_smt,
                               Y_smt,
                               cv = shuffle_split,
                               n_jobs = 1,
                               scoring = ('accuracy', 'precision_macro', 'recall_macro', 'f1_macro'),
                               return_train_score = True)

    # compute the ROC scores
    myscore = make_scorer(roc_auc_score, multi_class='ovo',needs_proba=True)
    roc_scores = cross_validate(model,
                                X_smt,
                                Y_smt,
                                cv = shuffle_split,
                                n_jobs = 1,
                                scoring = myscore,
                                return_train_score = True)
    return cv_result, roc_scores
```

```
######################## Rebuild Model (Random Forest) using Best Parameters ##################

# From Hyperparameter Tuning:
# criterion = entropy, max_depth = 6,  n_estimators = 80, min_samples_split = 3

rfc_model = RandomForestClassifier(random_state = 42,
                                   max_features = 'sqrt',
                                   max_leaf_nodes = None,
                                   n_estimators = int(best_params_df.loc['rfc__n_estimators']),
                                   max_depth = int(best_params_df.loc['rfc__max_depth']),
                                   criterion = best_params_df.loc['rfc__criterion'],
                                   min_samples_split = int(best_params_df.loc['rfc__min_samples_split']))


######################## Evaluate the Rebuild Model - Random Forest ##########################
pipe_rfc = Pipeline([('scaler', StandardScaler()), ('rfc', rfc_model)])
rfc_cv_eval, rfc_rocScores = cross_validation_eval(pipe_rfc, X_smt, Y_smt)


rfc_cv_results = pd.DataFrame({
                                'Train Accuracy': rfc_cv_eval['train_accuracy'].mean(),
                                'Validate Accuracy': rfc_cv_eval['test_accuracy'].mean(),
                                'Train Precision': rfc_cv_eval['train_precision_macro'].mean(),
                                'Validate Precision': rfc_cv_eval['test_precision_macro'].mean(),
                                'Train Recall': rfc_cv_eval['train_recall_macro'].mean(),
                                'Validate Recall': rfc_cv_eval['test_recall_macro'].mean(),
                                'Train F1': rfc_cv_eval['train_f1_macro'].mean(),
                                'Validate F1': rfc_cv_eval['test_f1_macro'].mean(),
                                'Train ROC': rfc_rocScores['train_score'].mean(),
                                'Validate ROC': rfc_rocScores['test_score'].mean()}, index = ['Random Forest']).T


# display the cross-validation scores before and after hyperparameter optimization
print("\nCross-Validation Scores (Before Hyperparameter Optimization) for Random Forest Classifier (Obesity): ")
print(baseline_models_CV_eval['Random Forest'])
print("\nCross-Validation Scores (After Hyperparameter Optimization) for Random Forest Classifier (Obesity): ")
print(rfc_cv_results)
```

Figure 4.7c: A sample code of the re-trained and re-evaluated of the Random Forest Classifier with the Best Hyperparameters (Obesity)

Figure 4.7d, Figure 4.7e, Figure 4.7f and Figure 4.7g illustrate the outcomes of the **performance metrics before** and **after**, the **confusion matrix**, the **ROC-AUC** as well as the **classification report** of the Gradient Boosting Classifier with the best hyperparameters for diabetes.

```
Cross-Validation Scores (Before Hyperparameter Optimization) for Gradient Boosting Classifier (Diabetes):
Train Accuracy        0.983124
Validate Accuracy     0.971598
Train Precision       0.983575
Validate Precision    0.972481
Train Recall          0.982962
Validate Recall       0.972397
Train F1              0.982974
Validate F1           0.971457
Train ROC             0.997788
Validate ROC          0.993452
```

```
Cross-Validation Scores (After Hyperparameter Optimization) for Gradient Boosting Classifier (Diabetes):
                   Gradient Boosting
Train Accuracy              0.994671
Validate Accuracy           0.986391
Train Precision             0.994773
Validate Precision          0.986391
Train Recall                0.994599
Validate Recall             0.986762
Train F1                    0.994655
Validate F1                 0.986402
Train ROC                   0.999918
Validate ROC                0.997943
```

Figure 4.7d: The outcome of the performance metrics **before** and **after** for Gradient Boosting Classifier with best hyperparameters (Diabetes)

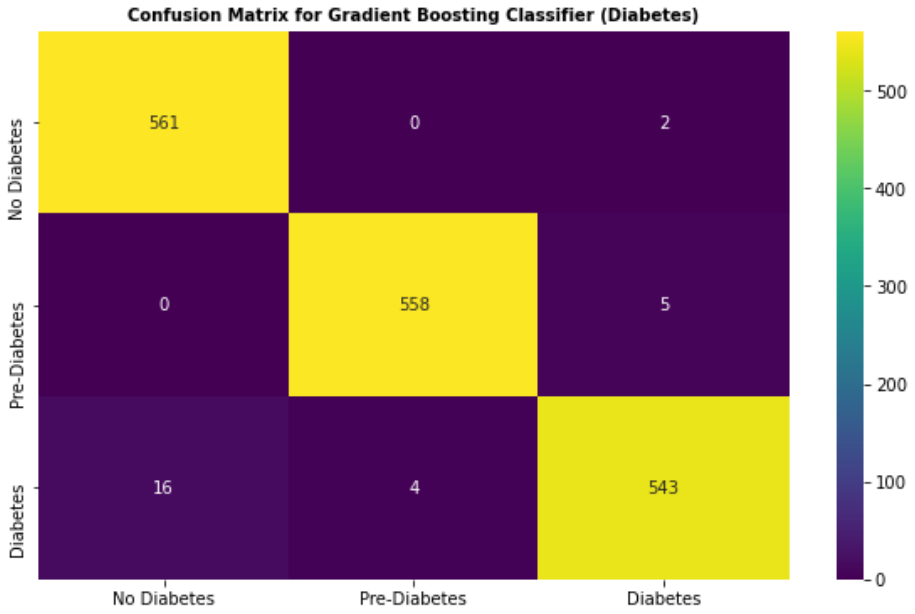


Figure 4.7e: The outcome of the confusion matrix for Gradient Boosting Classifier with best hyperparameters (Diabetes)

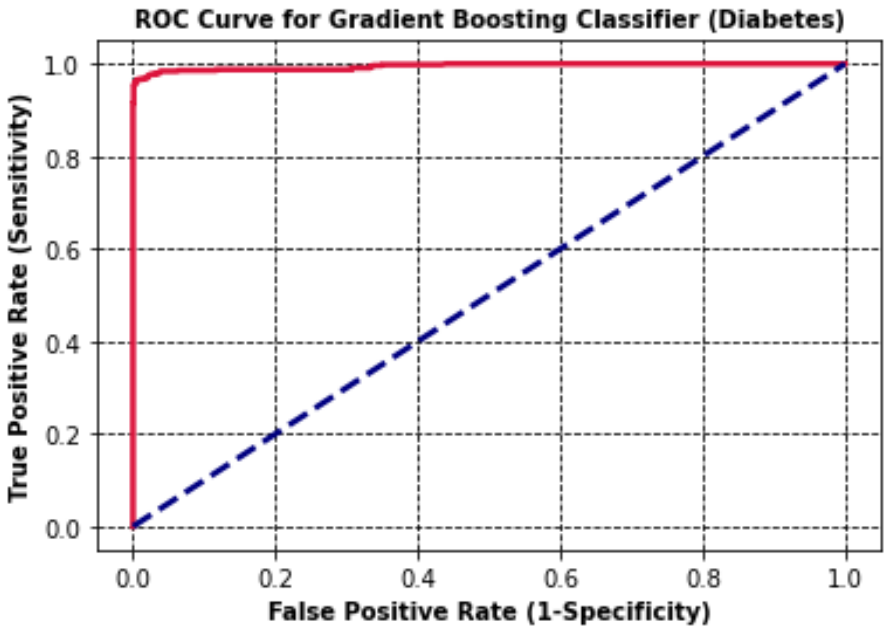


Figure 4.7f: The outcome of the ROC-AUC for Gradient Boosting Classifier with best hyperparameters (Diabetes)

Classification Report for Gradient Boosting with Hyperparameter Tuning (Diabetes)

	precision	recall	f1-score	support
No Diabetes	0.97	1.00	0.98	563
Pre-Diabetes	0.99	0.99	0.99	563
Diabetes	0.99	0.96	0.98	563
accuracy			0.98	1689
macro avg	0.98	0.98	0.98	1689
weighted avg	0.98	0.98	0.98	1689

Figure 4.7g: The outcome of the classification report for Gradient Boosting Classifier with best hyperparameters (Diabetes)

Based on the illustration shown in figure 4.7d, the performance metrics for the gradient boosting classifier have improved after the best hyperparameters have been applied onto the gradient boosting classifier with the diabetes training set. Moreover, both training and validation ROC with 99% shows that the gradient boosting classifier model has the excellent capability to differentiate between the classes (no diabetes, pre-diabetes, or diabetes), which can be proven by the ROC-AUC shown in figure 4.7f. Furthermore, the gradient boosting classifier with hyperparameters has a very high accuracy rate of 99%, but it is also imperative to look at the other parameters to evaluate the performance of this classifier model. For example, high precision rate relates to the low false-positive rate, this is absolutely aggregable with the confusion matrix shown in figure 4.7e, which has about less

than 5% of having false-positive rate. As such, this gradient boosting classifier with hyperparameters has a very high precision rate of nearly 99% based on the training and validation set. In addition, this gradient boosting classifier with hyperparameters also has an excellent recall rate of 99% based on the validation and training set, which is good for this hyperparameter model as it is above 50% and the high recall rate also relates to the low false-negative rate, which can be further proven by the confusion matrix as shown in figure 4.7e.

Based on the confusion matrix shown in figure 4.7e, both **false positive** and **false negative** are the major concern in the biomedical field. False-positive represented that these patients were predicted with diabetes, but in fact, these patients were having no diabetes. Conversely, false-negative represented that these patients were predicted with no diabetes, but in actual fact, these patients were having diabetes. As such, both false-positive and false-negative are the most important factors that we would like to minimize. One must also note that the diabetes training set consists of multi-class. As such, the computation for TP, FP, TN, and FN are slightly different from the binary classification, and below shows the computation for the TP, FP, TN, and FN for confusion matrix with multi-class.

For Class 0: No Diabetes

TP = 561

FP = 0 + 16 = 16

TN = 558 + 5 + 4 + 543 = 1110

FN = 0 + 2 = 2

For Class 1: Pre-Diabetes

TP = 558

FP = 0 + 4 = 4

TN = 561 + 2 + 16 + 543 = 1122

FN = 0 + 5 = 5

For Class 2: Diabetes

TP = 543

FP = 5 + 2 = 7

TN = 561 + 558 + 0 + 0 = 1119

FN = 16 + 4 = 20

Thus, both false-positive rate and false-negative rate for all classes are not too much of a concern for this case as it is likely to have less than 5% chances. In other words, the gradient boosting classifier with hyperparameters for diabetes works considerably well.

Figure 4.7h, Figure 4.7i, Figure 4.7j and Figure 4.7k illustrate the outcomes of the **performance metrics before** and **after**, the **confusion matrix**, the **ROC-AUC** as well as the **classification report** of the Random Forest Classifier with the best hyperparameters for hypertension.

```
Cross-Validation Scores (Before Hyperparameter Optimization) for Random Forest Classifier (Hypertension):
Train Accuracy          1.000000
Validate Accuracy       0.954286
Train Precision         1.000000
Validate Precision      0.955446
Train Recall            1.000000
Validate Recall         0.952646
Train F1                1.000000
Validate F1             0.952981
Train ROC               1.000000
Validate ROC            0.984008
```

```
Cross-Validation Scores (After Hyperparameter Optimization) for Random Forest Classifier (HYpertension):
                    Random Forest
Train Accuracy           0.976259
Validate Accuracy        0.974286
Train Precision          0.976673
Validate Precision       0.974363
Train Recall             0.976329
Validate Recall          0.973351
Train F1                 0.976230
Validate F1              0.973652
Train ROC                0.998363
Validate ROC             0.983529
```

Figure 4.7h: The outcome of the performance metrics **before** and **after** for Random Forest Classifier with best hyperparameters (Hypertension)

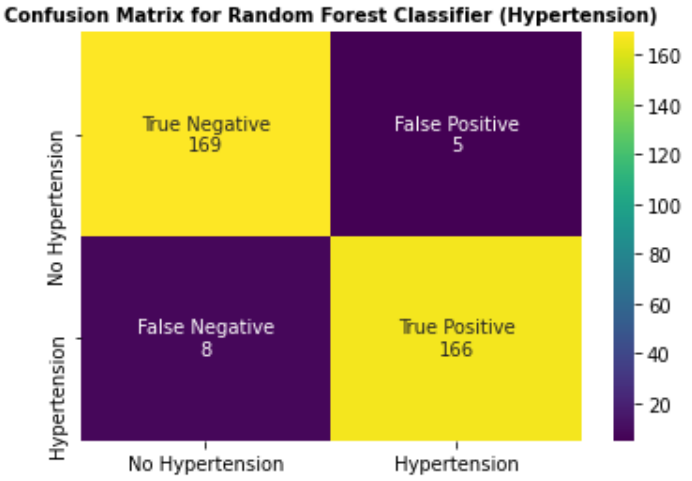


Figure 4.7i: The outcome of the confusion matrix for Random Forest Classifier with best hyperparameters (Hypertension)

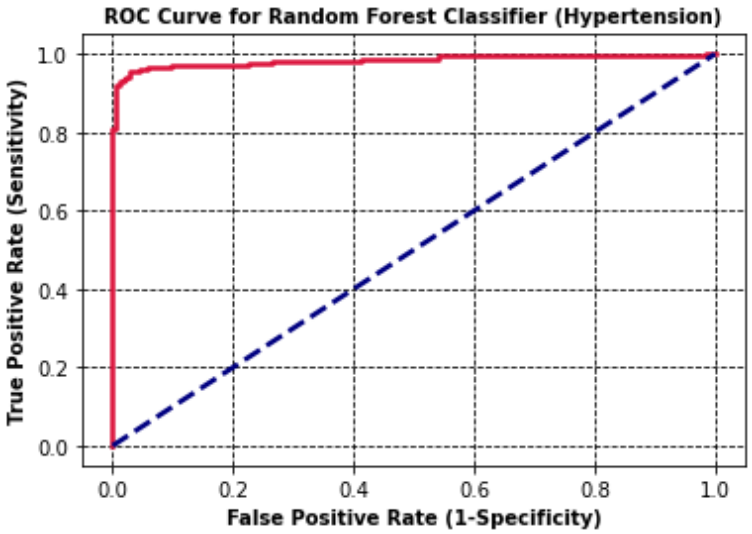


Figure 4.7j: The outcome of the ROC-AUC for Random Forest Classifier with best hyperparameters (Hypertension)

Classification Report for Random Forest with Hyperparameter Tuning (Hypertension)

	precision	recall	f1-score	support
No Hypertension	0.95	0.97	0.96	174
Hypertension	0.97	0.95	0.96	174
accuracy			0.96	348
macro avg	0.96	0.96	0.96	348
weighted avg	0.96	0.96	0.96	348

Figure 4.7k: The outcome of the classification report for Random Forest Classifier with best hyperparameters (Hypertension)

Based on the illustration shown in figure 4.7h, prior to the hyperparameters optimization process, the random forest classifier accuracy score for the training set is slightly higher than the validation set. This phenomenon is known as **overfitting** and the cause of the overfitting could be mainly due to the **use of default parameters**. As such, after applying the best hyperparameters into the random forest classifier, the overfitting issues have been resolved. Furthermore, the rest of the performance metrics were also improved significantly. Moreover, both training and validation ROC with 98% shows that the random forest classifier model has also with the excellent capability to differentiate between the classes (no hypertension or hypertension), which can be proven by the ROC-AUC shown in figure 4.7j. As mentioned earlier, it is also imperative to look at the other parameters to evaluate the performance of this classifier model. For example, high precision rate relates to the low

false-positive rate, this is absolutely aggregable with the confusion matrix shown in figure 4.7e, which has about less than 5% of having false-positive rate. As such, this random forest classifier with hyperparameters has a very high precision rate of nearly 97% based on the training and validation set. In addition, this random forest classifier with hyperparameters also performed exceptionally well with a recall rate of 97%, which is good for this hyperparameter model as it is above 50%, and high recall rate can also relate to the low false-negative rate, which can be proven by the confusion matrix as shown in Figure 4.7i.

Based on the confusion matrix shown in figure 4.7i, both **false-positive** and **false-negative** are the major concern in the biomedical field. False-positive represented that these patients were predicted with hypertension, but in fact, these patients were having no hypertension. Conversely, false-negative represented that these patients were predicted with no hypertension, but in actual fact, these patients were having hypertension. As such, both false positive and false negative are the most potent factors that we would like to minimize. For this case, both false positive and false negative for all classes are extremely low and most likely to have less than 5% chances. This means that the random forest classifier with hyperparameters for hypertension works generally well.

Figure 4.7l, Figure 4.7m, Figure 4.7n and Figure 4.7o illustrate the outcomes of the **performance metrics before** and **after**, the **confusion matrix**, the **ROC-AUC** as well as the **classification report** of the Random Forest Classifier with the best hyperparameters for obesity.

```
Cross-Validation Scores (Before Hyperparameter Optimization) for Random Forest Classifier (Obesity):
Train Accuracy        1.000000
Validate Accuracy     0.989888
Train Precision       1.000000
Validate Precision    0.989712
Train Recall          1.000000
Validate Recall       0.989671
Train F1              1.000000
Validate F1           0.989662
Train ROC             1.000000
Validate ROC          0.999657
```

```
Cross-Validation Scores (After Hyperparameter Optimization) for Random Forest Classifier (Obesity):
                   Random Forest
Train Accuracy          0.968885
Validate Accuracy       0.957303
Train Precision         0.969301
Validate Precision      0.956819
Train Recall            0.969021
Validate Recall         0.956741
Train F1                0.969021
Validate F1             0.956539
Train ROC               0.996811
Validate ROC            0.995296
```

Figure 4.7l: The outcome of the performance metrics **before** and **after** for Random Forest Classifier with best hyperparameters (Obesity)

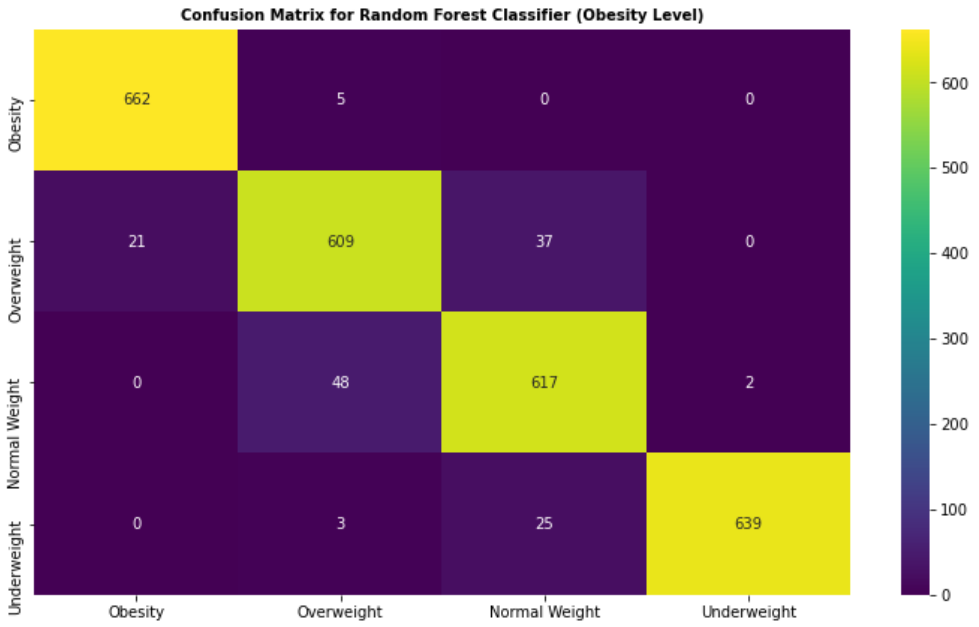


Figure 4.7m: The outcome of the confusion matrix for Random Forest Classifier with best hyperparameters (Obesity)

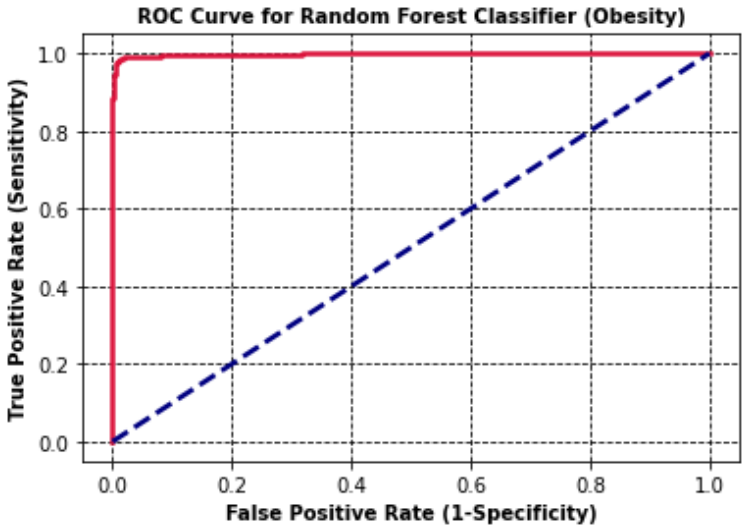


Figure 4.7n: The outcome of the confusion matrix for Random Forest Classifier with best hyperparameters (Obesity)

Classification Report for Random Forest with Hyperparameter Tuning (Obesity)

	precision	recall	f1-score	support
Underweight	0.97	0.99	0.98	667
Normal Weight	0.92	0.91	0.91	667
Overweight	0.91	0.93	0.92	667
Obesity	1.00	0.96	0.98	667
accuracy			0.95	2668
macro avg	0.95	0.95	0.95	2668
weighted avg	0.95	0.95	0.95	2668

Figure 4.7o: The outcome of the classification report for Random Forest Classifier with best hyperparameters (Obesity)

Based on the illustration shown in figure 4.7l, prior to the hyperparameters optimization process, the random forest classifier accuracy score for the obesity training set is slightly higher than the obesity validation set. This phenomenon is known as **overfitting** and the cause of the overfitting could be mainly due to the **use of default parameters**. As such, after applying with the best hyperparameters into the random forest classifier, the overfitting issues has been minimized. Furthermore, the rest of the performance metrics were also improved significantly. Moreover, both training and validation ROC with 99% shows that the random forest classifier model has also with the excellent capability to differentiate between the classes (underweight, normal weight, overweight, or obesity), which can be

proven by the ROC-AUC shown in figure 4.7n. As mentioned earlier, it is also imperative to look at the other parameters to evaluate the performance of this classifier model. For example, high precision relates to the low false positive rate, this is absolutely aggregable with the confusion matrix shown in figure 4.7m, which is about less than 5% of having false positive rate. As such, this random forest classifier with hyperparameters has a very high precision rate of nearly 96% based on the training and validation set. In addition, this random forest classifier with hyperparameters also performed exceptionally recall rate of 96% based on the validation and training set, which is good for this hyperparameter model as it is above 50% and high recall rate also relates to low false negative rate, which can be proven by the confusion matrix shown in Figure 4.7m.

Based on the confusion matrix shown in figure 4.7i, both **false positive** and **false negative** are the major concern in the biomedical field. False positive represented that these patients were predicted with obesity, but in fact, these patients were having no obesity. Conversely, false negative represented that these patients were predicted with no obesity, but in actual fact, these patients were having obesity. As such, both false positive and false negative are the most crucial factors that we would like to minimize. Similar as the diabetes, the obesity training set consists of multi-class. As such, the computation for TP, FP, TN, and FN are slightly different from the binary classification, and below shows the computation for the TP, FP, TN, and FN for confusion matrix with multi-class.

For Class 1: Underweight

TP = 662

FP = 21 + 0 + 0 = 21

TN = 609 + 37 + 0 + 48 + 617 + 2 + 3 + 25 + 639 = 1980

FN = 5 + 0 + 0 = 5

For Class 2: Normal Weight

TP = 609

FP = 48 + 3 + 5 = 56

TN = 662 + 0 + 0 + 0 + 0 + 617 + 2 + 25 + 639 = 1945

FN = 21 + 37 + 0 = 58

For Class 3: Overweight

TP = 617

FP = 0 + 37 + 25 = 62

TN = 662 + 5 + 21 + 609 + 0 + 3 + 0 + 0 + 639 = 1939

FN = 48 + 2 = 50

For Class 4: Obesity

TP = 639

FP = 0 + 0 + 0 = 2

TN = 662 + 5 + 0 + 21 + 609 + 37 + 0 + 48 + 617 = 1999

FN = 3 + 25 + 0 = 28

Generally, both false positive and false negative for all classes are not so much of the concern as they are likely to have less than 5% chances. In other words, this classifier model with hyperparameters for obesity works generally well.

4.8 Evaluation of the Hyperparameter Classifier Models using Testing Sets

Basically, a test set is a data set that is independent of the training set, but it follows the same probability distribution as the training data set. If a model fits into training set also fits into the test data as well. A test set is therefore a set of examples used to only assess the overall performance of the fully implemented classifiers. To achieve this, the final classifiers models were used to predict the classifications of examples in the test set. These predictions are then compared with the examples of true classifications in the test set to assess the model performances.

Figure 4.8a, Figure 4.8b, Figure 4.8c, and Figure 4.8d illustrate the outcome of the performance of the gradient boosting classifier with hyperparameter based on **diabetes testing set.**

```
Performance Metrics of the Gradient Boosting with Hyperparameter Tuning (Diabetes) using Testing Set:
=============================================================================================================
Accuracy Score:  0.9685314685314685
Precision Score:  0.8902255639097745
Recall Score:  0.9875518672199171
F1 Score:  0.9338360767139836
ROC Score:  0.996394632040826
```

Figure 4.8a: Performance metrics of Gradient Boosting Classifier with hyperparameters (Diabetes)

```
Classification Report for Gradient Boosting with Hyperparameter Tuning (Diabetes) using Testing Set:
              precision    recall  f1-score   support

 No Diabetes       0.83      1.00      0.91        29
Pre-Diabetes       0.84      1.00      0.91        16
    Diabetes       1.00      0.96      0.98       241

    accuracy                           0.97       286
   macro avg       0.89      0.99      0.93       286
weighted avg       0.97      0.97      0.97       286
```

Figure 4.8b: Classification report of Gradient Boosting Classifier with hyperparameters (Diabetes)

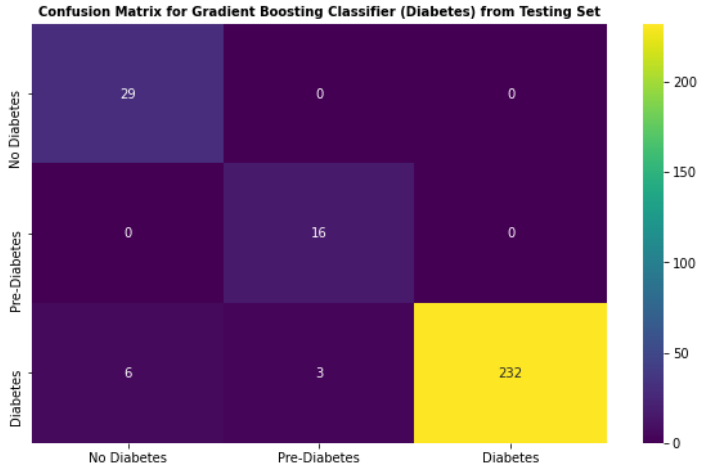


Figure 4.8c: Confusion matrix of Gradient Boosting Classifier with hyperparameters (Diabetes)

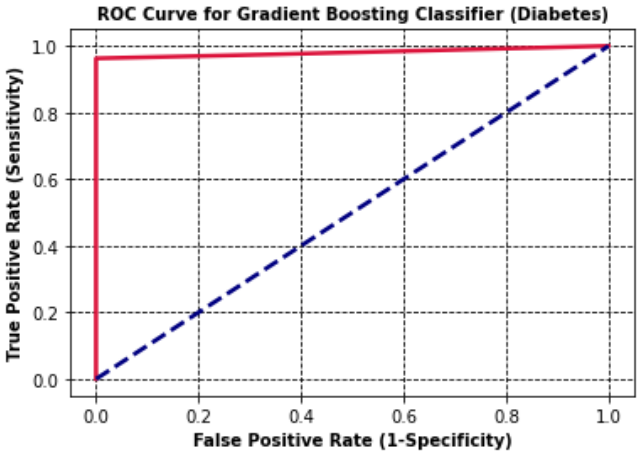


Figure 4.8d: ROC-AUC of Gradient Boosting Classifier with hyperparameters (Diabetes)

Based on the performance metrics and the classification report shown in Figure 4.8a and Figure 4.8b respectively, the gradient boosting classifier with hyperparameters has a very high accuracy rate of 97% as well as an excellent ROC-AUC score of about 99%, which means that this gradient boosting classifier with hyperparameters has the strong capability to distinguishes between classes (i.e., no diabetes, pre-diabetes, or diabetes) and this can also be proven by the ROC-AUC shown in Figure 4.8d. Apart from the accuracy and ROC scores, it is also imperative to investigate other performance metrics as well. The precision score, recall score and f1 score shown in Figure 4.8a are actually based on macro-averaging. This means that the macro-averaging is performed by first computing the respective precision, recall, and f1-score of each class, and then taking the average of all the respective precision, recall and f1-scores. The reason of using macro-average is to ensure all classes

are contributed equally regardless of how often they appear in the testing dataset. Based on the performance metrics as well as the classification report, it can see that the gradient boosting classifier with hyperparameters has done generally well with respect to the f1-score, precision and recall macro-averaging.

One can see that this is a multi-classification problem for diabetes. The computation of the TP, FP, TN, and FN in confusion matrix for multi-class is slightly indifferent from the binary classification. As such, the following below illustrate the calculation of the TP, FP, TN, and FN for multi-class.

For Class 0: No Diabetes

TP = 29

FP = 0 + 6 = 6

TN = 16 + 0 + 3 + 232 = 251

FN = 0 + 0 = 0

For Class 1: Pre-Diabetes

TP = 16

FP = 0 + 3 = 3

TN = 29 + 0 + 6 + 232 = 267

FN = 0 + 0 = 0

For Class 2: Diabetes

TP = 232

FP = 0 + 0 = 0

TN = 29 + 0 + 0 + 16 = 45

FN = 6 + 3 = 9

Based on the confusion matrix computation shown above, both **false positive (FP) rate** and **false-negative (FN) rate** for **all classes are generally low**, which is about less than 5%. Therefore, the low false positive rate also leading to a high precision rate and also a low false negative rate also leading to a high recall rate, which can be proven in the performance metrics shown in Figure 4.8a. Overall, this proposed classifier model is generally acceptable.

Figure 4.8e, Figure 4.8f, Figure 4.8g, and Figure 4.8h illustrate the outcome of the performance of the random forest classifier with hyperparameter based on **hypertension testing set.**

```
Performance Metrics of the Random Forest with Hyperparameter Tuning (Hypertension) using Testing Set:
=======================================================================================================
Accuracy Score:  0.9747899159663865
Precision Score:  0.9748417721518987
Recall Score:  0.9691994996873046
F1 Score:  0.971927341354093
ROC Score:  0.9691994996873046
```

Figure 4.8e: Performance metrics of Random Forest Classifier with hyperparameters (Hypertension)

```
Classification Report for Random Forest with Hyperparameter Tuning (Hypertension) using Testing Set:
                precision    recall  f1-score   support

No Hypertension       0.97      0.95      0.96        41
   Hypertension       0.97      0.99      0.98        78

       accuracy                           0.97       119
      macro avg       0.97      0.97      0.97       119
   weighted avg       0.97      0.97      0.97       119
```

Figure 4.8f: Classification report of Random Forest Classifier with hyperparameters (Hypertension)

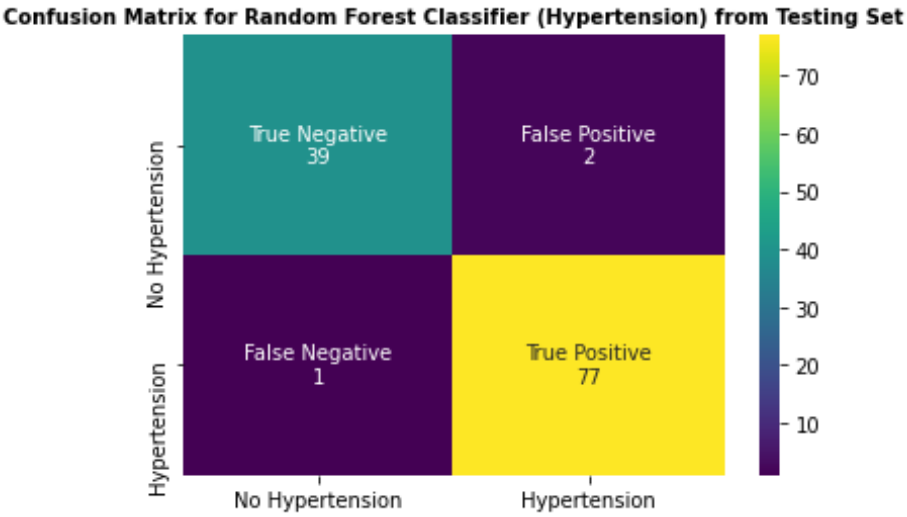


Figure 4.8g: Confusion matrix of Random Forest Classifier with hyperparameters (Hypertension)

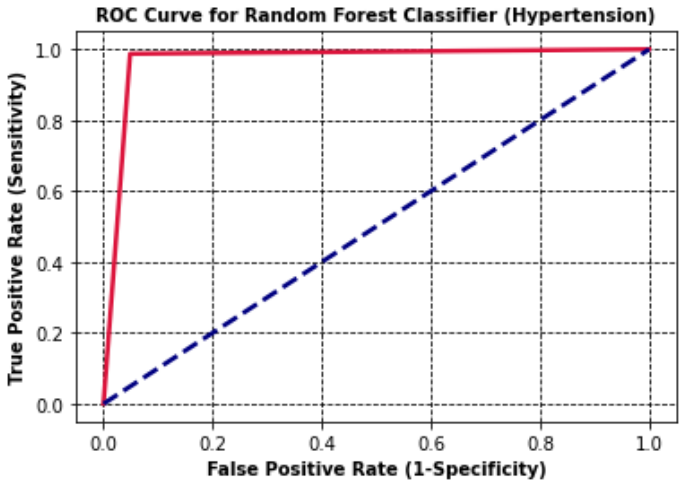


Figure 4.8h: ROC-AUC of Random Forest Classifier with hyperparameters (Hypertension)

Based on the performance metrics and the classification report shown in Figure 4.8e and Figure 4.8f respectively, the random forest classifier with hyperparameters has a very high accuracy rate of 97% as well as an excellent ROC-AUC score of about 97%, which means that this random forest classifier with hyperparameters has the strong capability to distinguishes between classes (i.e., no hypertension or hypertension) and this can also be proven by the ROC-AUC shown in Figure 4.8h. As mentioned earlier, it is also imperative to investigate other performance metrics as well. The precision score, recall score and f1 score shown in Figure 4.8e are based on macro-averaging. Based on the performance metrics as well as the classification report, it can see that the random forest classifier with hyperparameters has done exceptionally well with respect to the scores of the f1-score, precision and recall macro-averaging.

Based on the confusion matrix shown in Figure 4.8g, both **false positive (FP) rate** and **false-negative (FN) rate** for **all classes are generally low**, which is about less than 5%. Therefore, the low false-positive rate leading to a high precision rate and also a low false negative-rate leading to a high recall rate which can be proven in the performance metrics shown in Figure 4.8e, and the proposed classifier model for hypertension is generally acceptable.

Figure 4.8i, Figure 4.8j, Figure 4.8k, and Figure 4.8l illustrate the outcome of the performance of the random forest classifier with hyperparameter based on **obesity testing set.**

```
Performance Metrics of the Random Forest with Hyperparameter Tuning (Obesity) using Testing Set:
=================================================================================================
Accuracy Score:  0.9396551724137931
Precision Score:  0.9171024981257231
Recall Score:  0.9337673968188673
F1 Score:  0.9249661599872964
ROC Score:  0.9875320006419606
```

Figure 4.8i: Performance metrics of Random Forest Classifier with hyperparameters (Obesity)

```
Classification Report for Random Forest with Hyperparameter Tuning (Obesity) using Testing Set:
              precision    recall  f1-score   support

  Underweight       0.96      0.97      0.96        68
Normal Weight       0.82      0.89      0.86        66
   Overweight       0.91      0.91      0.91       160
      Obesity       0.99      0.96      0.97       286

     accuracy                           0.94       580
    macro avg       0.92      0.93      0.92       580
 weighted avg       0.94      0.94      0.94       580
```

Figure 4.8j: Classification report of Random Forest Classifier with hyperparameters (Obesity)

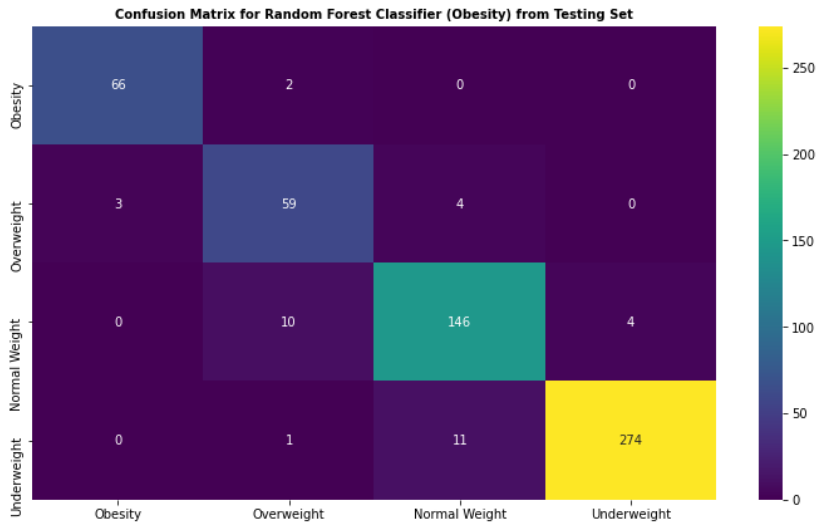


Figure 4.8k: Confusion matrix of Random Forest Classifier with hyperparameters (Obesity)

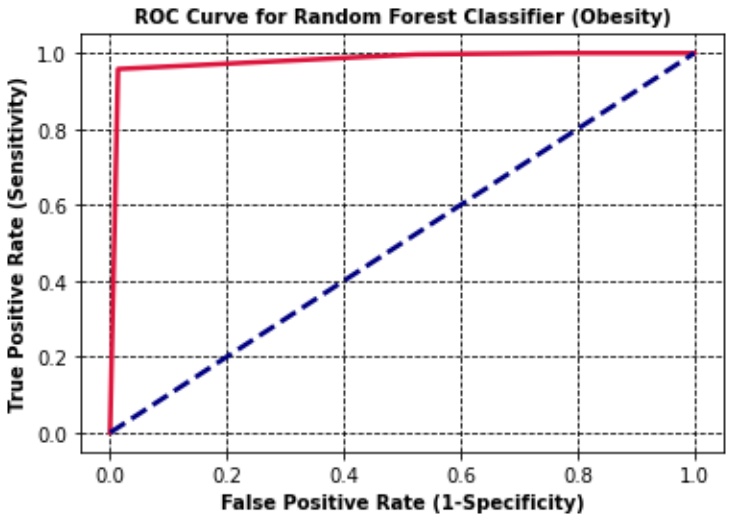


Figure 4.8l: ROC-AUC of Random Forest Classifier with hyperparameters (Obesity)

Based on the performance metrics and the classification report shown in Figure 4.8i and Figure 4.8j respectively, the random forest classifier with hyperparameters has a very high accuracy rate of 94% as well as an excellent ROC-AUC score of about 98%, which means that this random forest classifier with hyperparameters has the strong capability to distinguishes between classes (i.e., underweight, normal weight, overweight, or obesity) and this can also be proven by the ROC-AUC shown in Figure 4.8l. Furthermore, the precision score, recall score and f1 score shown in Figure 4.8i are actually based on macro-averaging. Based on the performance metrics as well as the classification report, this random forest classifier model has done generally well with respect to the scores of the f1-score, precision and recall macro-averaging.

Similar as diabetes, this is a multi-classification problem for obesity. As such, the following below illustrate the computation of the TP, FP, TN, and FN for multi-class.

For Class 1: Underweight

TP = 66

FP = 0 + 0 + 3 = 3

TN = 59 + 4 + 0 + 10 + 146 + 4 + 1 + 11 + 274 = 509

FN = 2 + 0 + 0 = 2

For Class 2: Normal Weight

TP = 59

FP = 10 + 1 + 2 = 13

TN = 66 + 0 + 0 + 0 + 0 + 146 + 4 + 11 + 274 = 501

FN = 3 + 4 + 0 = 7

For Class 3: Overweight

TP = 146

FP = 11 + 4 + 0 = 15

TN = 66 + 2 + 3 + 59 + 0 + 1 + 274 + 0 + 0 = 405

FN = 0 + 10 + 4 = 14

For Class 4: Obesity

TP = 274

FP = 0 + 0 + 4 = 4

TN = 66 + 2 + 0 + 3 + 59 + 4 + 0 + 10 + 146 = 290

FN = 0 + 1 + 11 = 12

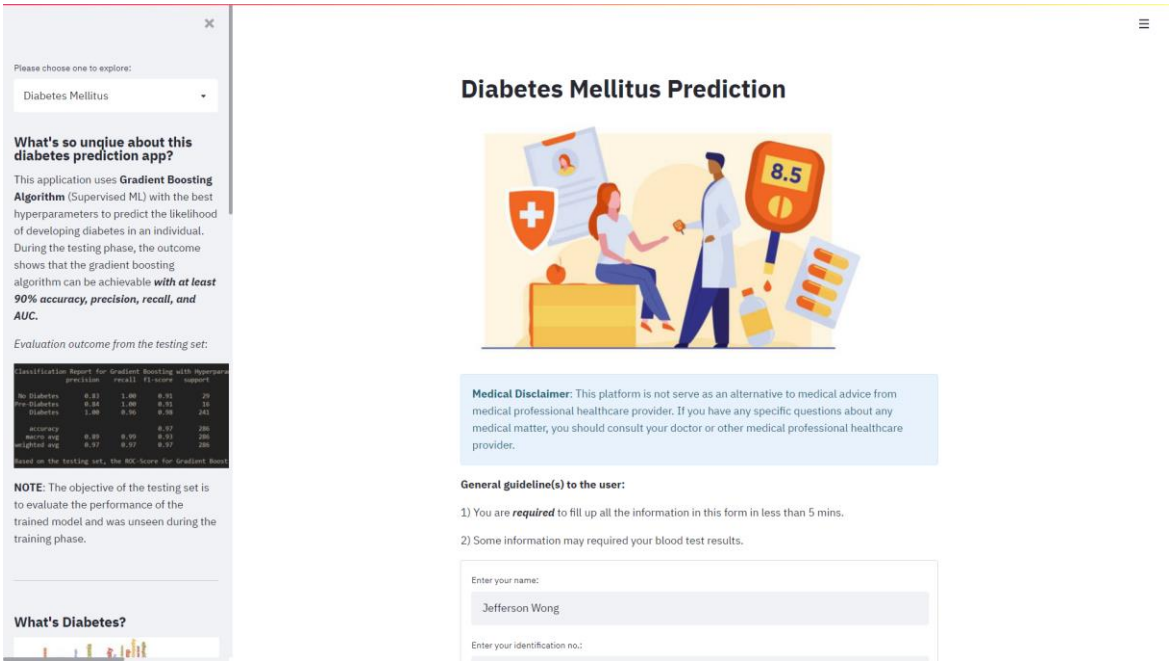
Based on the confusion matrix computation shown above, both **false positive (FP) rate** and **false-negative (FN) rate** for **all classes are generally low**, which is about less than 5%. As such, the low false positive rate also leading to a high precision rate and also a low false negative rate also leading to a high recall rate, which can be proven in the performance metrics shown in Figure 4.8i. Thus, the proposed classifier model with hyperparameter for obesity is generally acceptable.

4.9 Development of Health Web Application using Streamlit

All the respective predictive algorithms that have been developed in Python using Spyder IDE were saved in the Pickle files and the web application that has been built for this project was using Streamlit. Thereafter, the proposed respective predictive algorithms have been integrated with the Streamlit app as its backbone. The development of this web app and disease predictive model could help users effectively minimize their hypertension, diabetes as well as obesity while all diseases are in their early stages. The outcome also included some recommendations such as try to achieve a healthy weight by balancing your caloric input or physical activity over the next 6 to 12 months and also consult your doctor as soon as possible, etc.

Figure 4.9a, Figure 4.9b, and Figure 4.9c illustrate the respective prediction outcome interface as well as recommendations for the **diabetes** via the Streamlit app.

For Class 0: No Diabetes



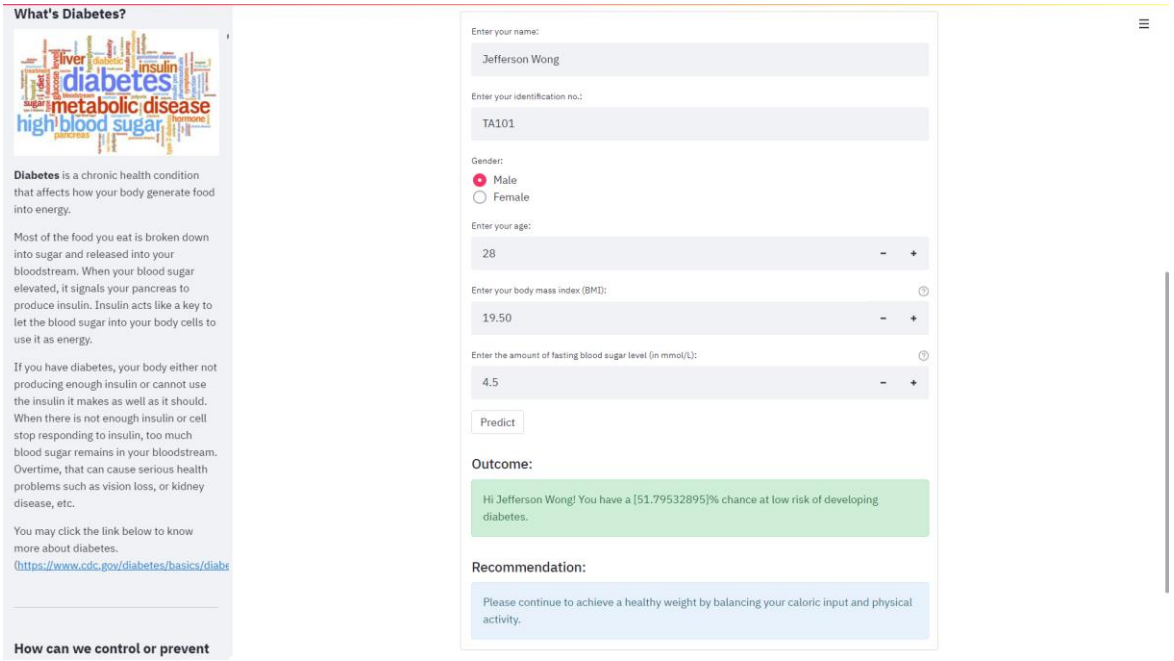
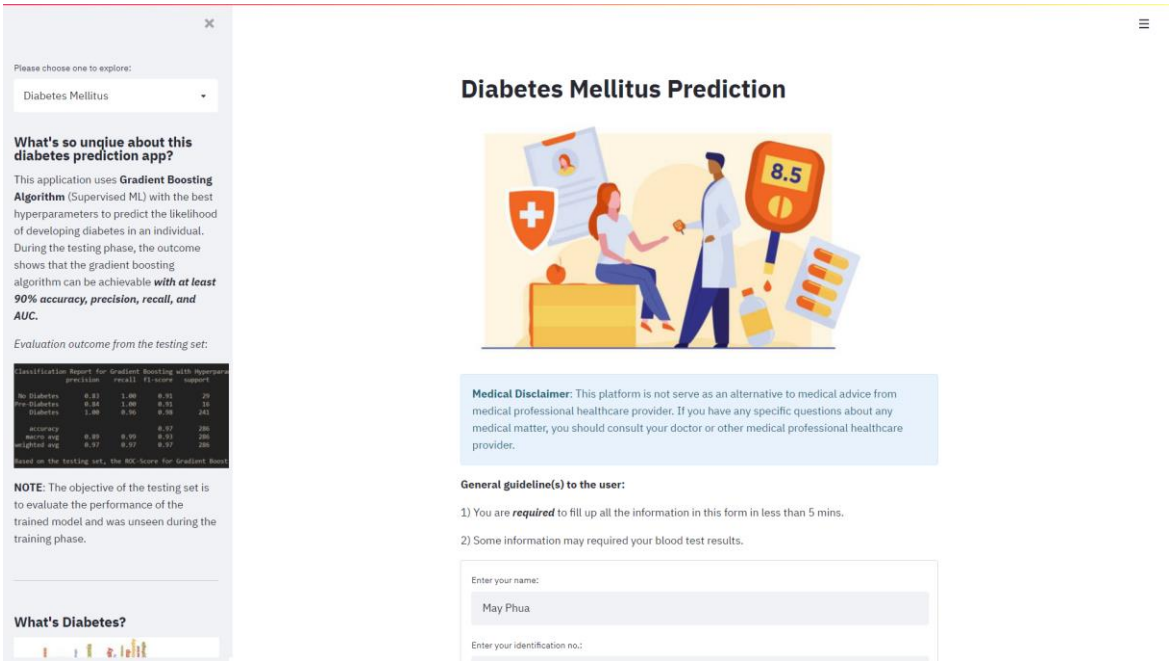


Figure 4.9a: A sample interface of the prediction outcome and recommendation in Streamlit app for **no diabetes**

For Class 1: Pre-Diabetes



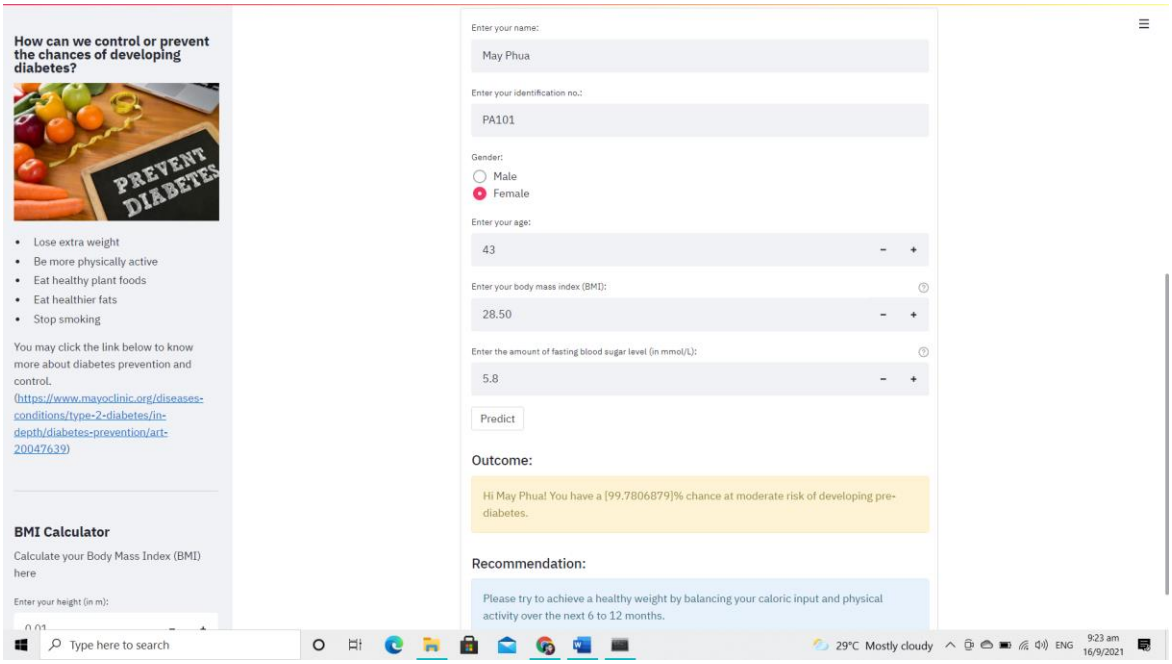
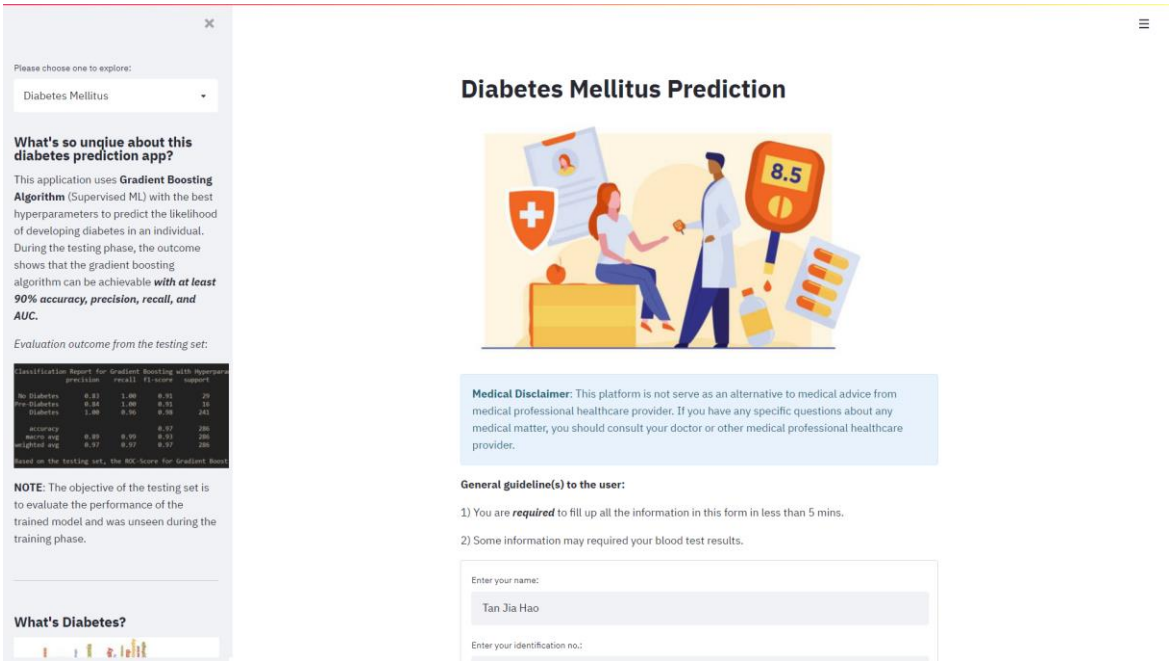


Figure 4.9b: A sample interface of the prediction outcome and recommendation in Streamlit app for **pre-diabetes**

For Class 2: Diabetes



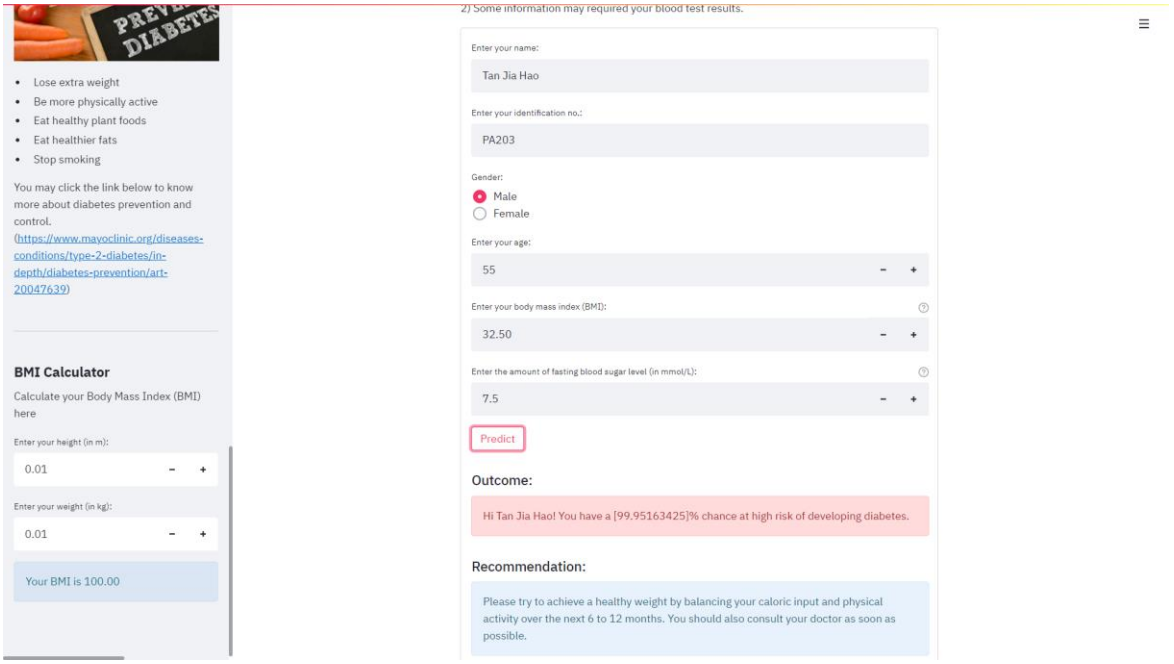


Figure 4.9c: A sample interface of the prediction outcome and recommendation in Streamlit app **for diabetes**

NOTE: Gender, BMI, and Fasting Blood Sugar Level are the attributes for the predictive process.

Figure 4.9d, and Figure 4.9e illustrate the respective prediction outcome interface as well as recommendations for the **hypertension** via the Streamlit app.

For Class 0: No hypertension

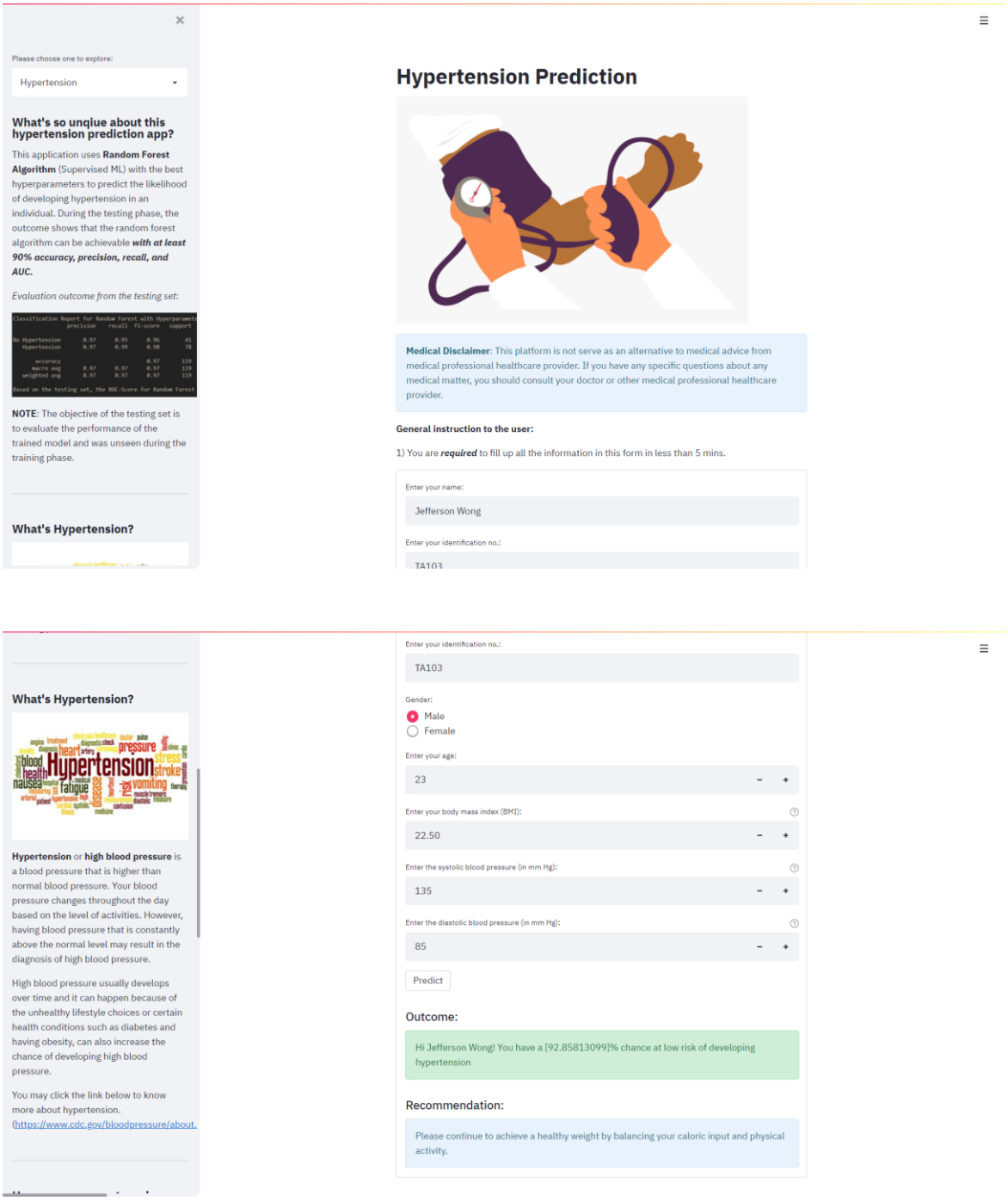


Figure 4.9d: A sample interface of the prediction outcome and recommendation in Streamlit app for **no hypertension**

For Class 1: Hypertension

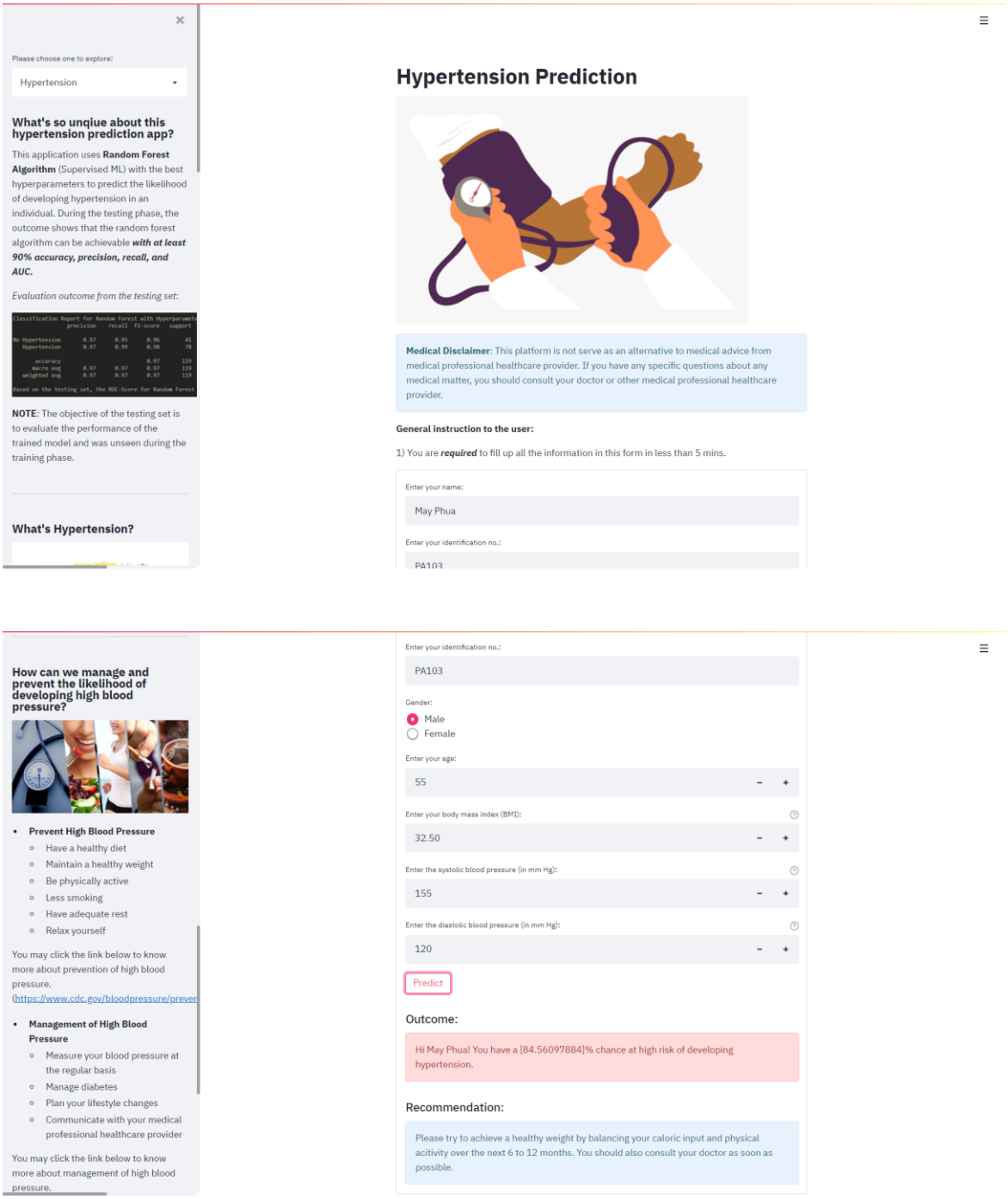
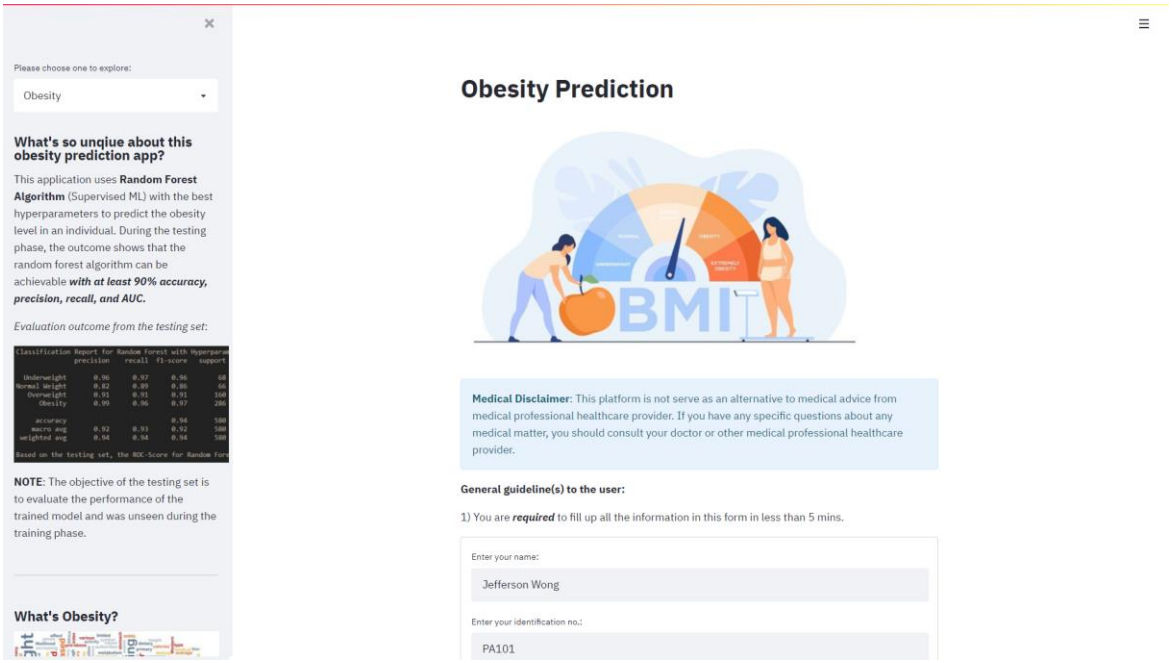


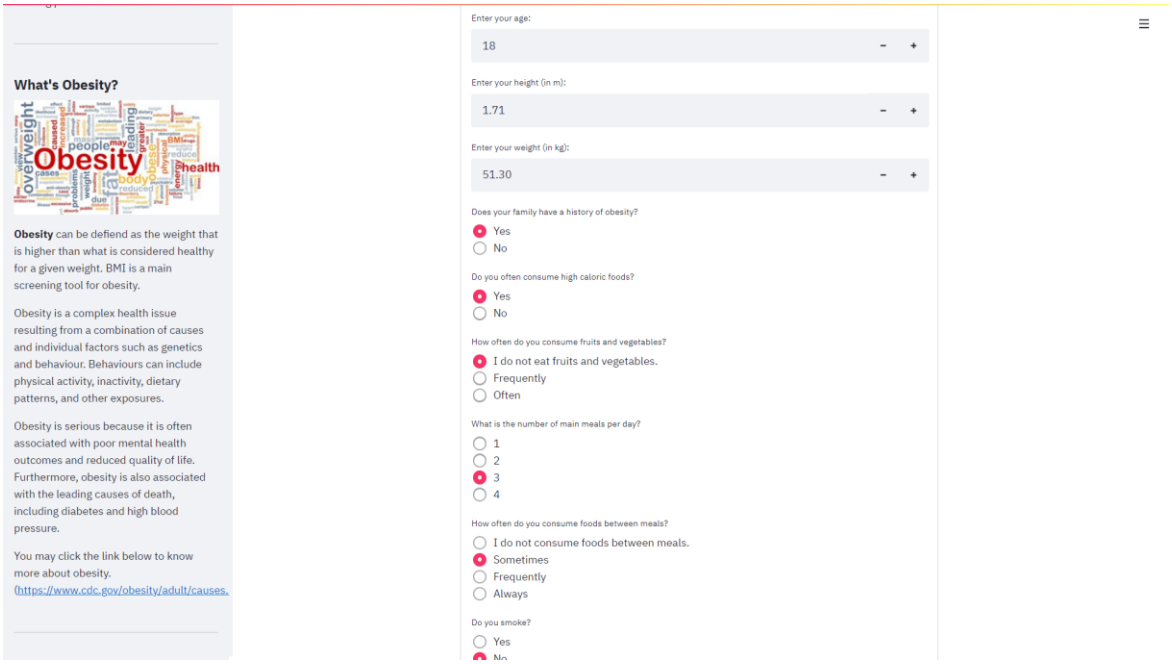
Figure 4.9e: A sample interface of the prediction outcome and recommendation in Streamlit app **for hypertension**

NOTE: BMI, Systolic Blood Pressure (SBP), and Diastolic Blood Pressure (DBP) are the attributes for the predictive process.

Figure 4.9f, Figure 4.9g, Figure 4.9h, and Figure 4.9i illustrate the respective prediction outcome interface as well as recommendations for the **obesity classification** via the Streamlit app.

For Class 1: At low-risk for obesity-related diseases (underweight)





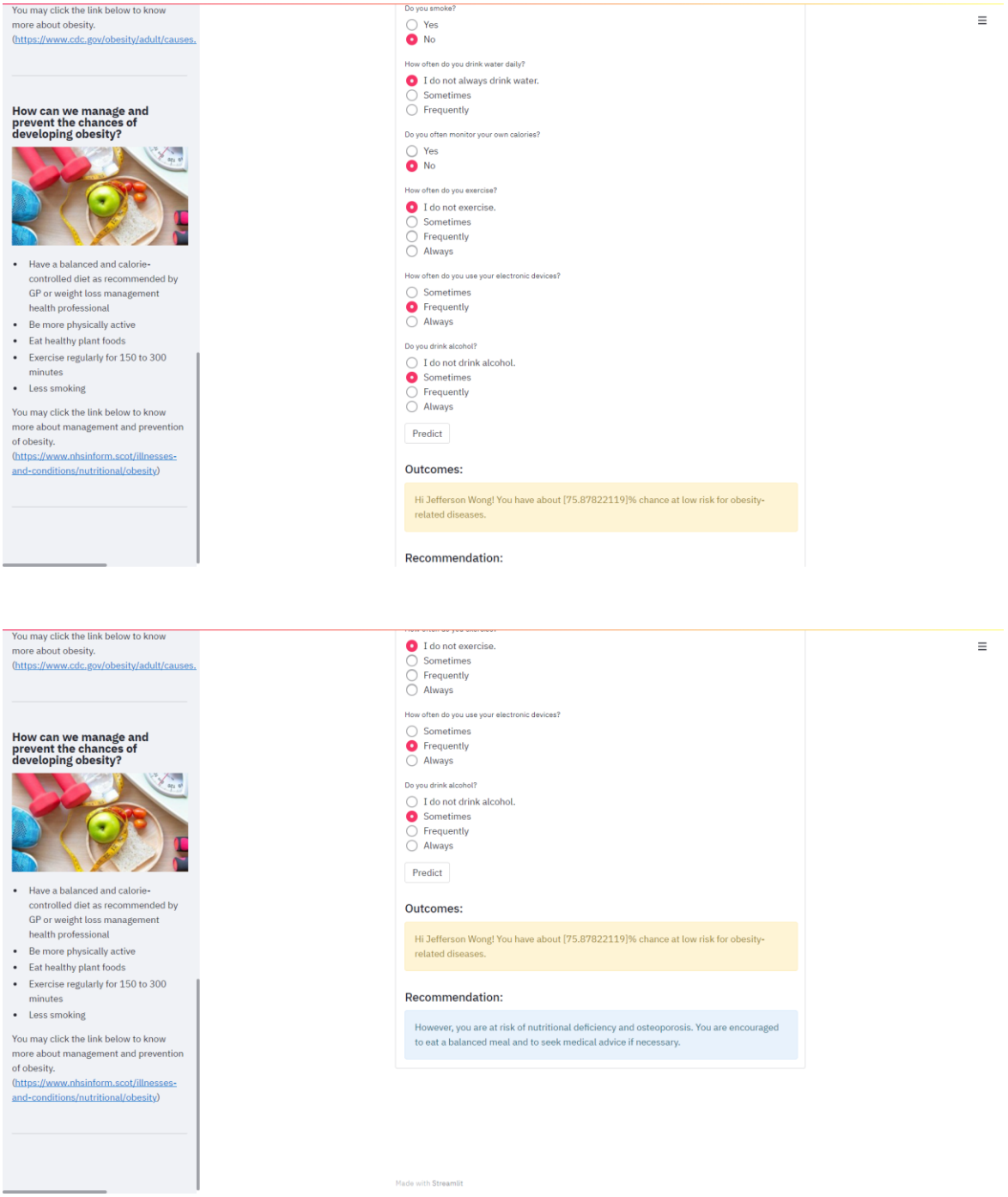
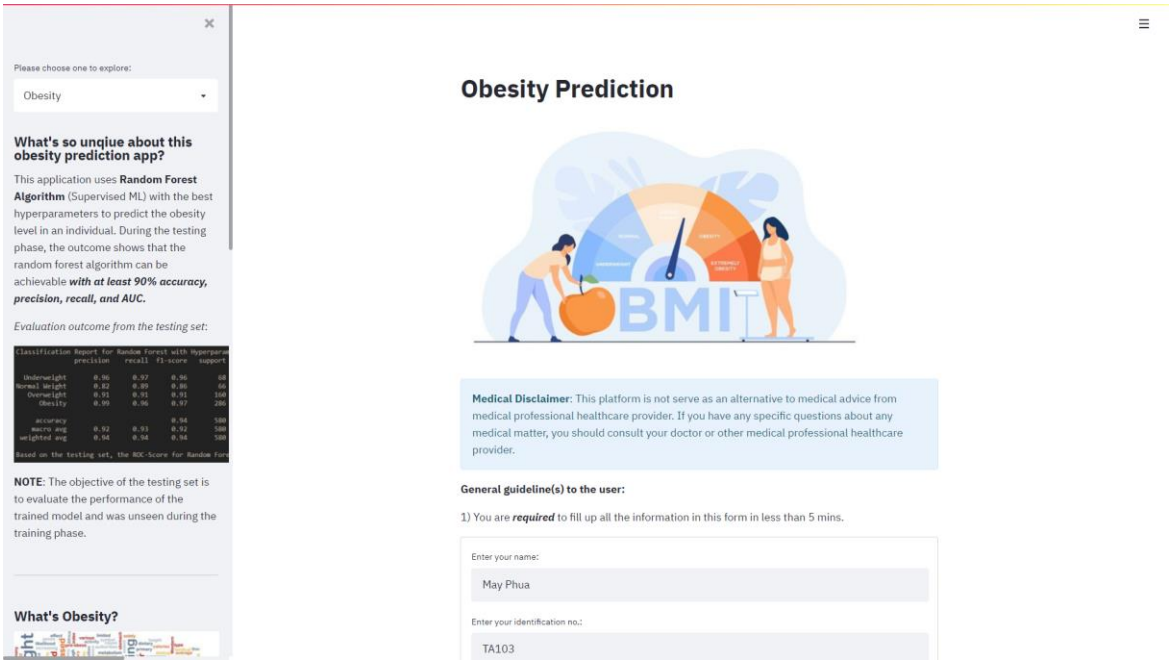
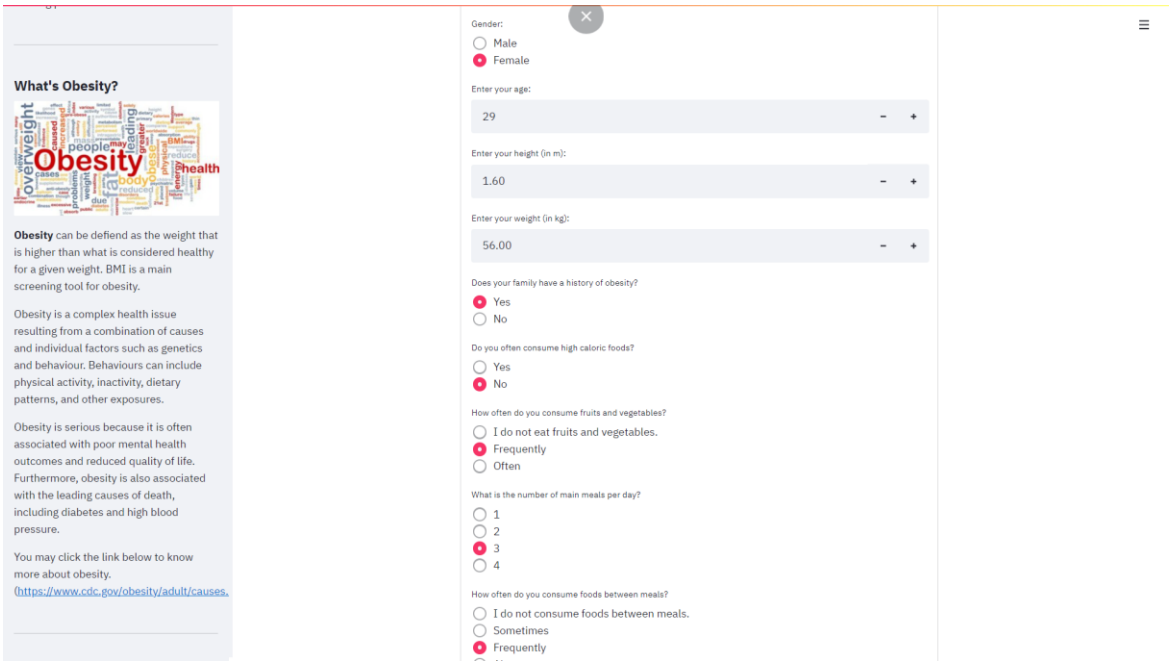


Figure 4.9f: A sample interface of the prediction outcome and recommendation in Streamlit app for **at low-risk of having obesity-related diseases (underweight)**

For Class 2: At low-risk for obesity-related diseases (normal weight)





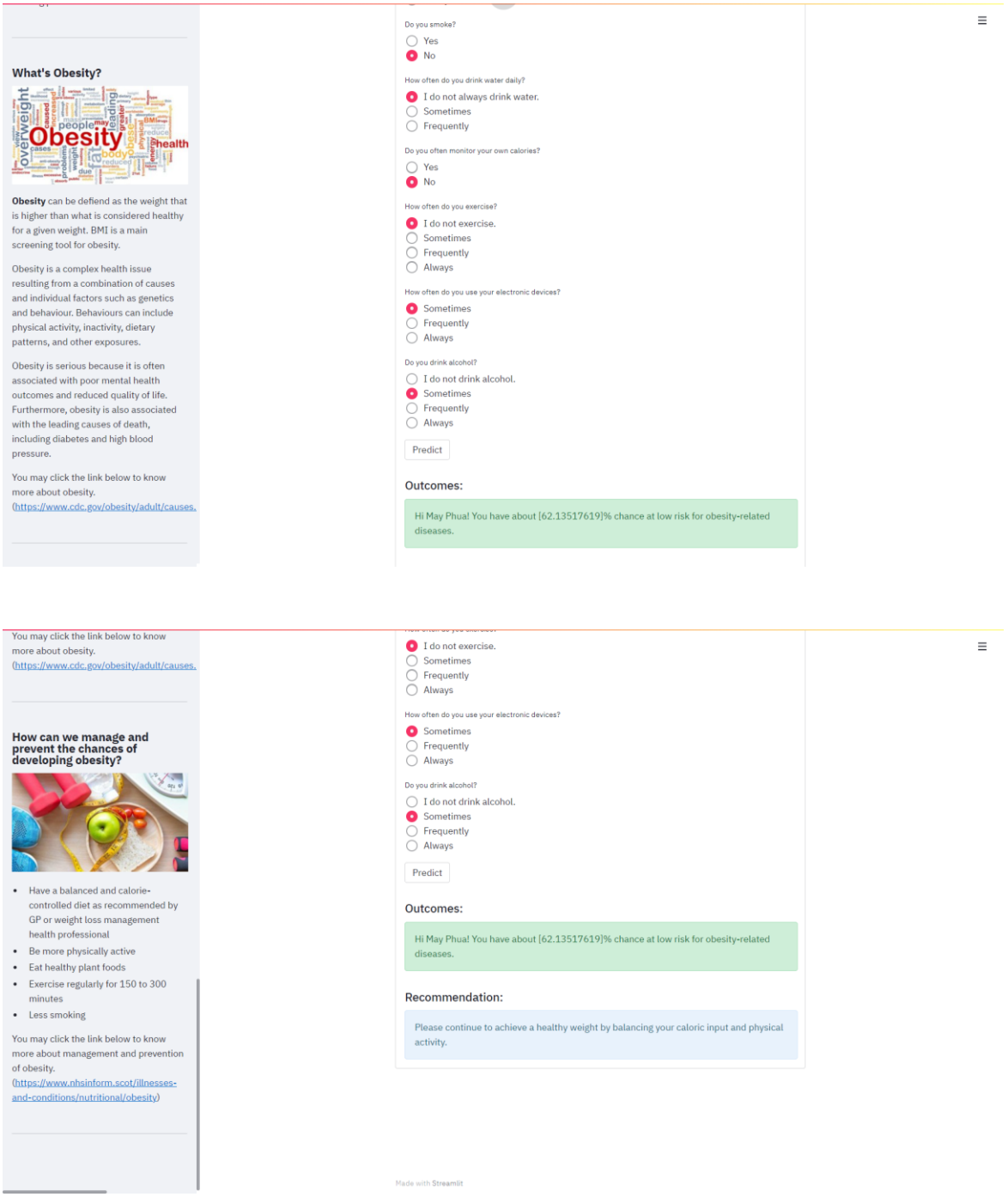
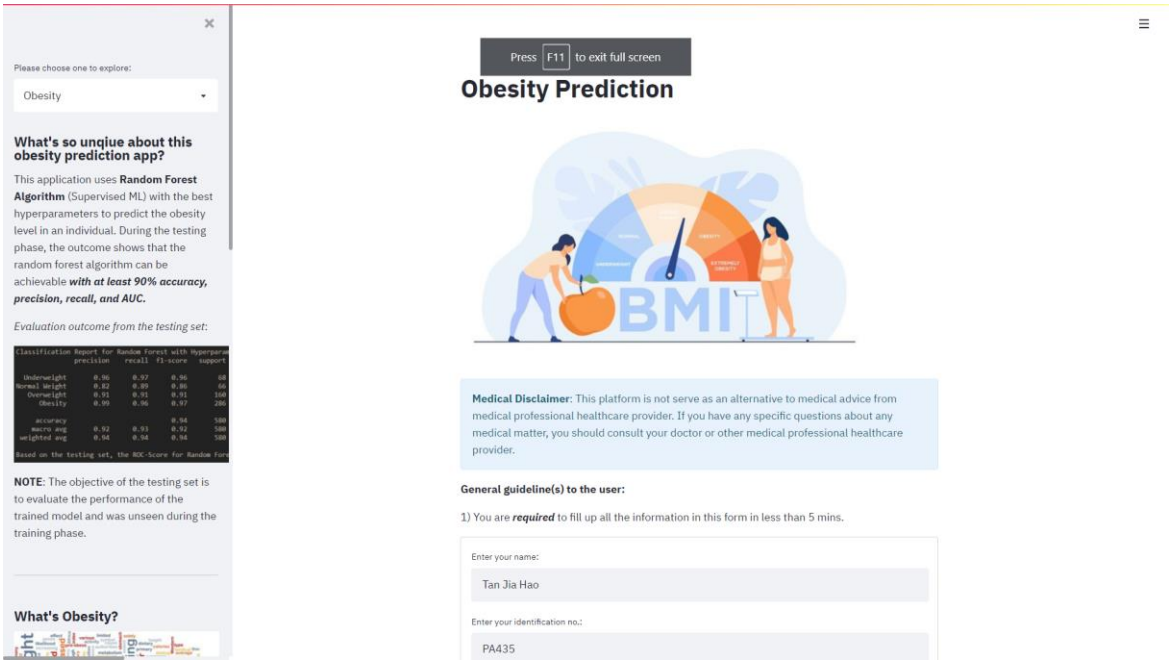
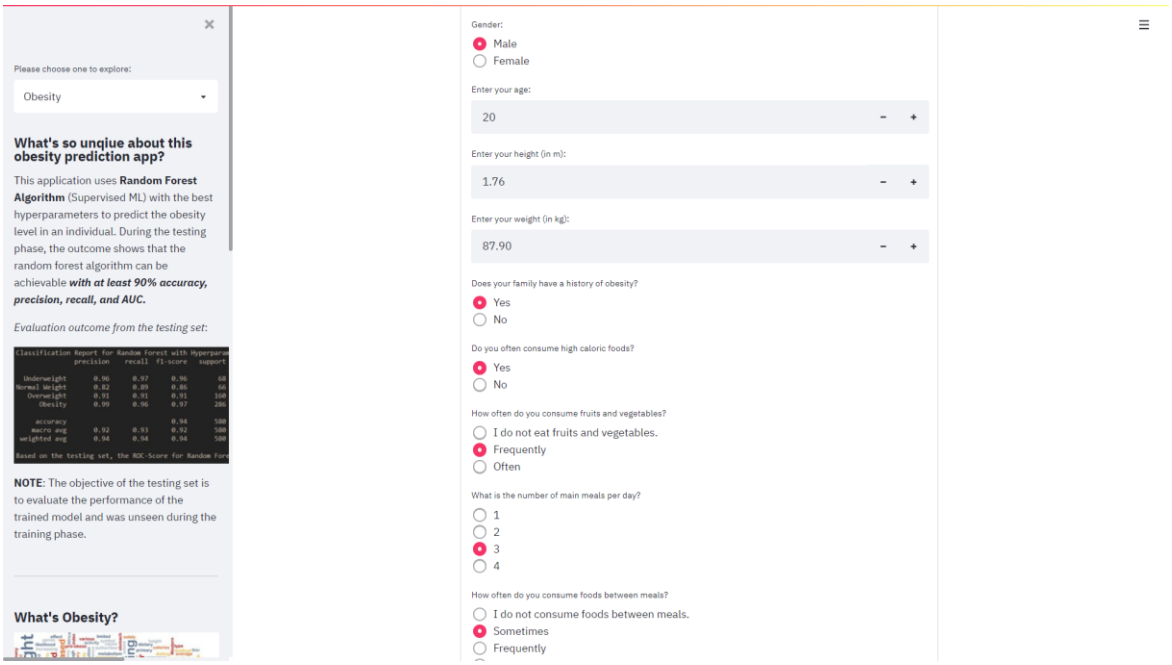


Figure 4.9g: A sample interface of the prediction outcome and recommendation in Streamlit app for **at low-risk of having obesity-related diseases (normal weight)**

For Class 3: At moderate-risk for obesity-related diseases (overweight)





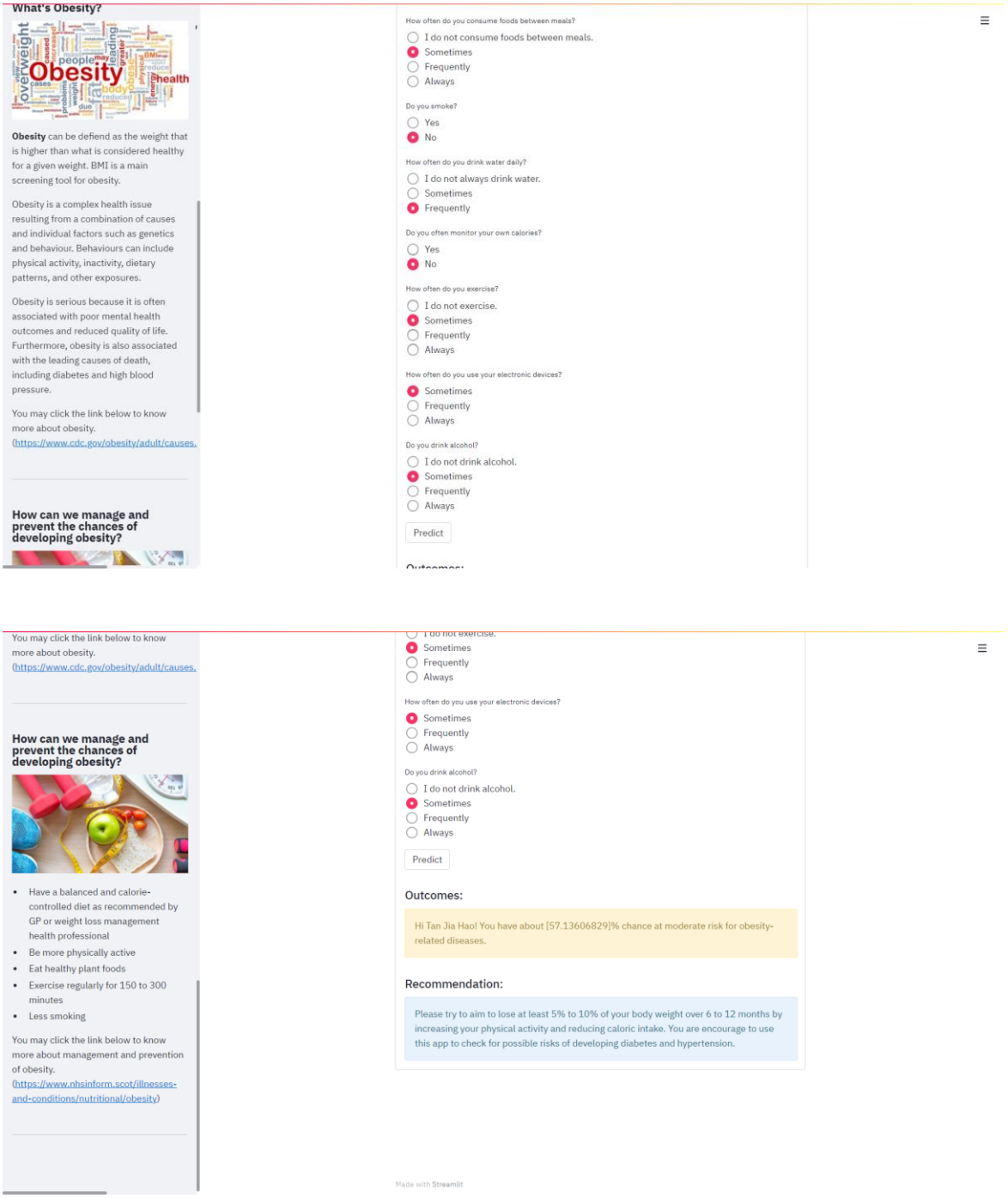
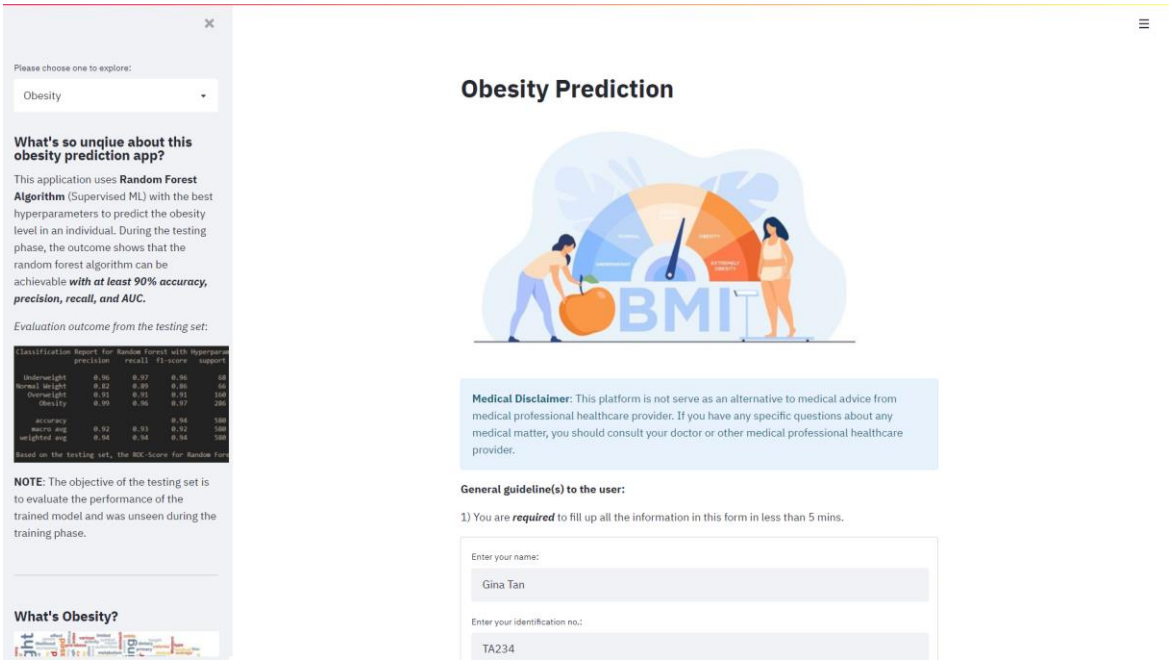
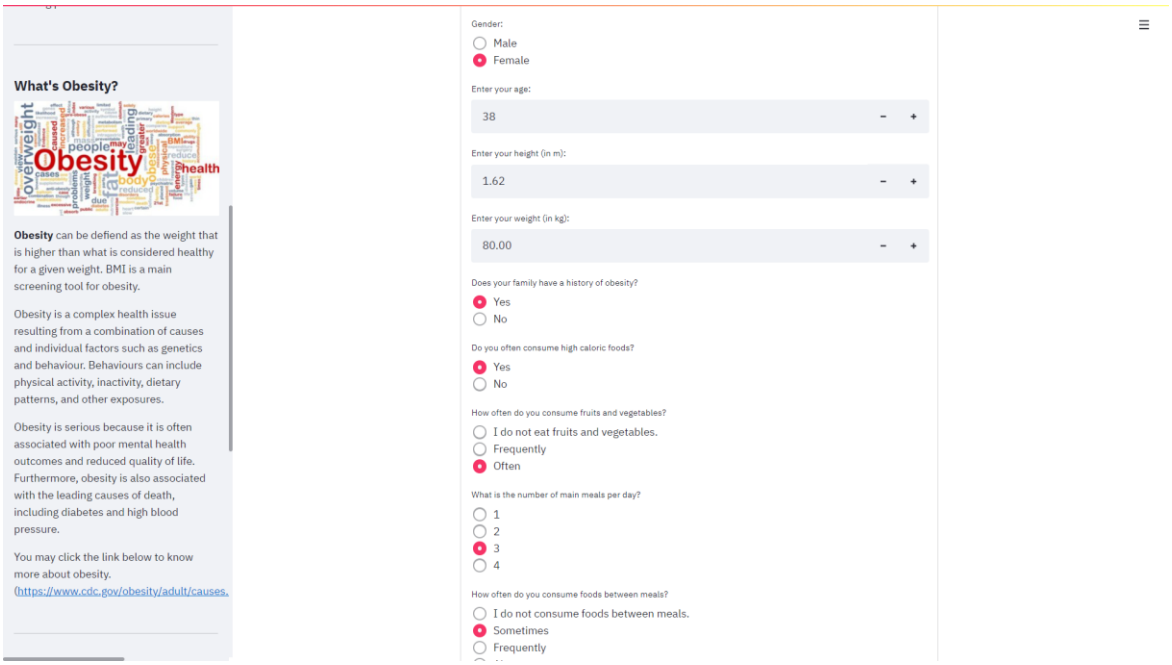


Figure 4.9h: A sample interface of the prediction outcome and recommendation in Streamlit app for **at moderate-risk of having obesity-related diseases (overweight)**

For Class 4: At high-risk for obesity-related diseases (obesity)





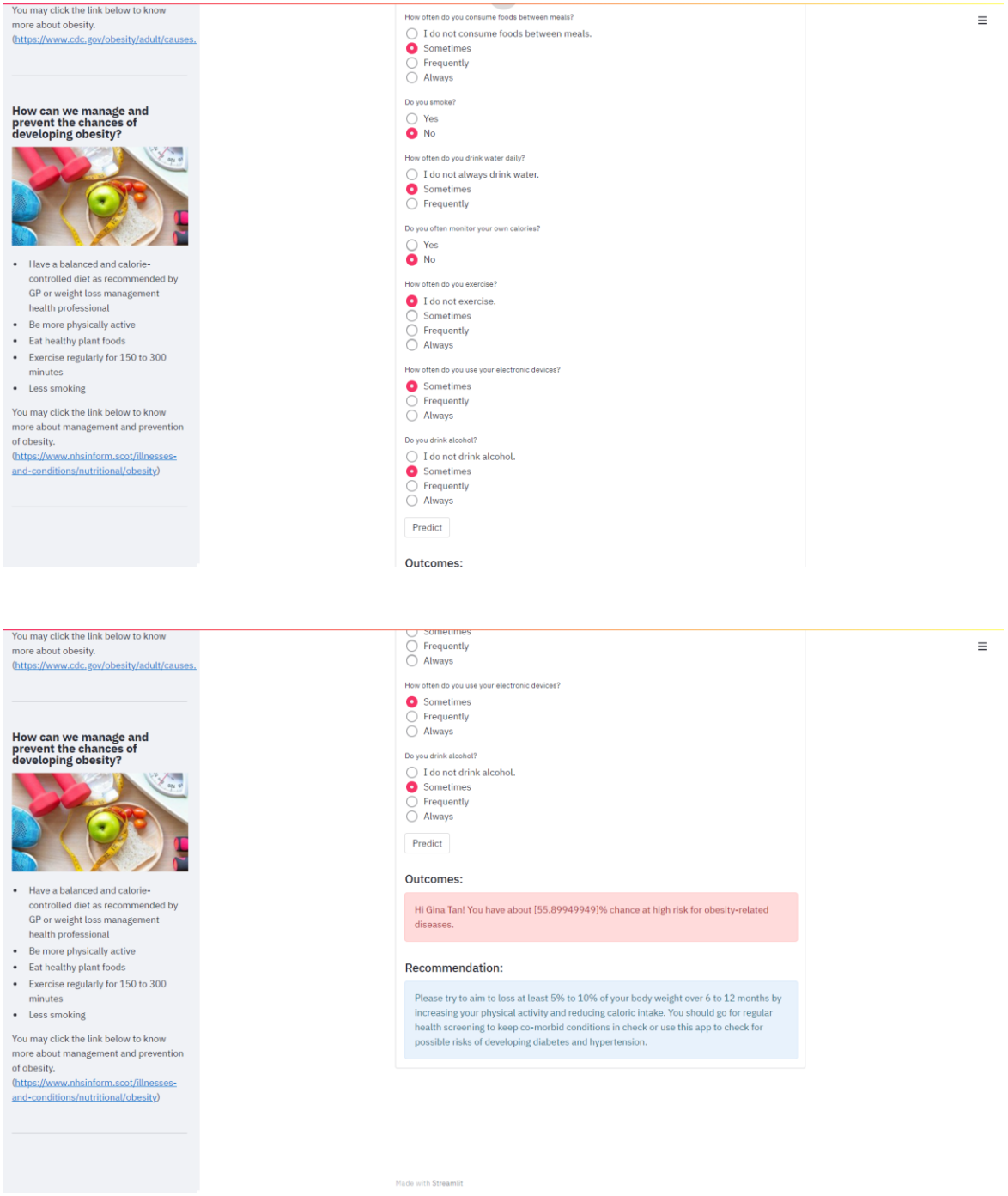


Figure 4.9i: A sample interface of the prediction outcome and recommendation in Streamlit app for **at high-risk of having obesity-related diseases (obesity)**

5.0 Conclusion

The development of the respective predictive models for respective obesity-related chronic diseases involved in the integration of the isolation forest to detect the potential outliers present in the original datasets, the SMOTE-TOMEK to balanced the training sets to minimize the bias between classes, and the investigation of the baseline models and the hyperparameters optimization process for the proposed machine learning models. The performance evaluation for the respective predictive models for respective obesity-related chronic diseases was done using the unseen testing sets. It was observed that all of the proposed predictive models for respective chronic diseases such as diabetes, hypertension, and obesity were found to have at least 90% for accuracy, at least 95% for ROC-AUC, and less than 5% of false-positive rate and false-negative rate based on the confusion matrix computation.

To simulate a real-life practical application, it has also developed a simple and friendly web application via Streamlit with the integration of the proposed predictive models with optimal hyperparameters for the respective obesity-related chronic diseases such as obesity, diabetes, and hypertension to diagnosed the user at their convenience. The web application comprises of respective obesity-related chronic diseases platform whereby the respective platform collates all the user risk factors input and then transmitted to the respective predictive model with optimal hyperparameters to performed the prediction. Hence, the class outcome with the highest probability represents the final outcome diagnosis of the respective obesity-related chronic diseases. The final outcome of the respective diagnosis appeared on the user's web app inclusive of the general health recommendation, which would allow them to take prompt and appropriate action to minimize and/ or prevent the risks further.

6.0 Recommendation

Future studies or expansion will include a more adequate large amount of data, more relevant attributes, and features, the implementation of the architecture design of the neural network to predict obesity-related chronic diseases, and enhancement of the web application based on a user perspective.

Firstly, the collection of a huge amount of data not only enhances the knowledge base of the predictive model but also improves the accurateness of the system to a more reliable extent.

Secondly, apart from the medical factors like BMI, blood sugar level, blood pressure level, etc., it will also put into consideration of other high-risk factors that have been found in both diabetes and hypertension in recent research studies such as the duration of exercise, dietary habits, smoking habits, and drinking of alcohol habits for the predictive models. Furthermore, different age groups could also be further investigated.

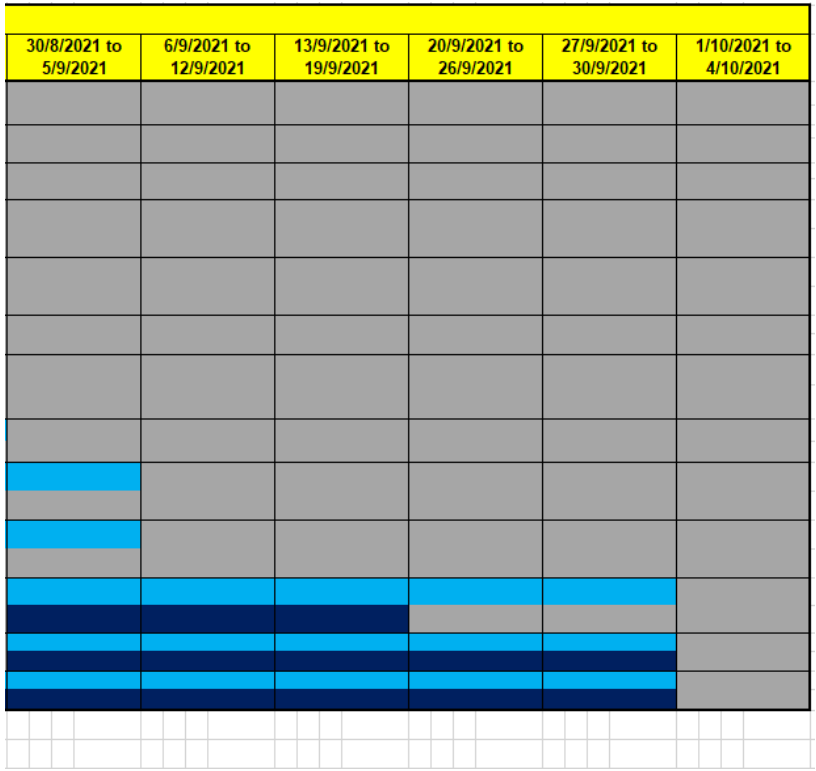
Thirdly, it will also be implementing and architecting the prediction systems using some kinds of neural networks such as ANN, etc., to predict the respective obesity-related chronic diseases as the neural networks are generally considerably better than the machine learning classifiers.

Lastly, an extensive evaluation of the usability and user experience of the web application should also be carried out in the near future. Furthermore, some additional feature(s) will be can also be included in the existing web application such as the transmission of the user's personal info with disease predictive outcomes to their respective professional healthcare providers. This will benefit to those users with unexpected prediction outcomes or possibly under threat of both diabetes and hypertension.

7.0 Project Implementation Scheduled

s/n	Task(s)	Scheduled Date								
		27/6/2021 to 4/7/2021	5/7/2021 to 11/7/2021	12/7/2021 to 18/7/2021	19/7/2021 to 25/7/2021	26/7/2021 to 1/8/2021	2/8/2021 to 8/8/2021	9/8/2021 to 15/8/2021	16/8/2021 to 22/8/2021	23/8/2021 to 29/8/2021
1	Collect of 1-dimensional datasets from various sources									
2	Store the 1-dimensional datasets into SQLite3 via Python									
3	Data Exploration on selected features for all respective obesity-related chronic diseases									
4	Data Cleansing (Investigate Missing Data) for all Datasets (Obesity, Diabetes, and Hypertension)									
5	Data Cleansing (Removal of Potential Outlier Detection using Isolation Forest) for all Datasets (Obesity, Diabetes, and Hypertension)									
6	Resampling techniques for imbalanced training datasets using SMOTE-Tomek									
7	Data Splitting, Model Selection (Baseline) and Evaluation via Shuffle-Split Cross-Validation for all respective Balanced Training Sets (Obesity, Diabetes, and Hypertension)									
8	Hyperparameters Optimization for Proposed Baseline Classifier Models									
9	Re-trained and Re-evaluation of the Proposed Classifier Models with the Best Hyperparameters via Shuffle-Split Cross-Validation									
10	Performance Evaluation of the Proposed Classifier Models with Hyperparameters using Testing Sets									
11	Development of Web App using Streamlit for Respective Obesity-related Chronic Diseases (Obesity, Diabetes, and Hypertension) with Integration of Predictive Modelling									
12	White-Box Testing on Web App									
13	Capstone Report Writing and Poster									

Planned Actual


30/8/2021 to 5/9/2021
6/9/2021 to 12/9/2021
13/9/2021 to 19/9/2021
20/9/2021 to 26/9/2021
27/9/2021 to 30/9/2021
1/10/2021 to 4/10/2021

8.0 References

[1] Rashid, A. (2020, Jul 18). Diabetes Dataset (Mendeley Data, version 1). Retrieved from https://data.mendeley.com/datasets/wj9rwkp9c2/1

[2] Machine Learning Examples (Neural Designer). Retrieved from https://www.neuraldesigner.com/learning/examples/Obesity-level

[3] Golino, H. (2013, Nov 11). Men's dataset from the "Predicting increased blood pressure using Machine Learning" paper. Retrieved from https://figshare.com/articles/dataset/Men_s_dataset_from_the_Predicting_increased_blood_pressure_using_Machine_Learning_paper/845665/1

[4] Golino, H. (2013, Nov 11). Women's dataset from the "Predicting increased blood pressure using Machine Learning" paper. Retrieved from https://figshare.com/articles/dataset/Women_s_dataset_from_the_Predicting_increased_blood_pressure_using_Machine_Learning_paper/845664/1

[5] Fitriyano, N.L. Syafdrudin, M. Alfian, G. Rhee, J. (2019, Oct 17). Development of Disease Prediction Model Based on Ensemble Learning Approach for Diabetes and Hypertension. Retrieved from https://www.researchgate.net/publication/336223791_Development_of_Disease_Prediction_Model_Based_on_Ensemble_Learning_Approach_for_Diabetes_and_Hypertension

[6] Streamlit Documentation. Retrieved from https://docs.streamlit.io/en/stable/getting_started.html

[7] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[8] Dhiraj, K. (2019). Anomaly Detection Using Isolation Forest in Python. Retrieved from https://blog.paperspace.com/anomaly-detection-isolation-forest/

[9] Brownlee, J. (2020, Jan 22). How to Combine Oversampling and Undersampling for Imbalanced Classification. Retrieved from https://machinelearningmastery.com/combine-oversampling-and-undersampling-for-imbalanced-classification/

[10] Satpathy, S. (2020, Oct 6). Overcoming Class Imbalance using SMOTE Techniques. Retrieved from https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

[11] API Reference. Retrieved from https://imbalanced-learn.org/dev/references/generated/imblearn.combine.SMOTETomek.html

[12] Scikit Learn on Gradient Boosting Classifier. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[13] Scikit Learn on Random Forest Classifier. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[14] Scikit Learn on Extra Trees Classifier. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

[15] Khan, N.S. Muaz, M.H. Kabir, A. Islam, M.N. (December 2017). Diabetes Predicting mHealth Application Using Machine Learning. Retrieved from https://www.researchgate.net/publication/327817574_Diabetes_Predicting_mHealth_Application_Using_Machine_Learning

[16] Qing, A. (2021, Jan 23). Nearly one-third of Singaporeans gained weight during COVID-19 pandemic: Survey. Retrieved from https://www.straitstimes.com/singapore/nearly-one-third-of-singaporeans-gain-weight-during-covid-19-pandemic-survey

[17] Division of Nutrition, Physical Activity, and Obesity, National Center for Chronic Diseases Prevention and Health Promotion. (2021, Mar 22). Adult Obesity Causes & Consequences. Retrieved from https://www.cdc.gov/obesity/adult/causes.html

[18] NHS Inform. (2021, May 17). Obesity. Retrieved from https://www.nhsinform.scot/illnesses-and-conditions/nutritional/obesity

[19] Centers for Disease Control and Prevention. (2020, Jun 11). What is Diabetes? Retrieved from https://www.cdc.gov/diabetes/basics/diabetes.html

[20] Mayo Clinic Staff. (2021, Jun 25). Diabetes prevention: 5 tips for taking control. Retrieved from https://www.mayoclinic.org/diseases-conditions/type-2-diabetes/in-depth/diabetes-prevention/art-20047639

[21] National Center for Chronic Diseases Prevention and Health Promotion, Division for Heart Disease and Stroke Prevention. (2021, May 18). High Blood Pressure Symptoms and Causes. Retrieved from https://www.cdc.gov/bloodpressure/about.htm

[22] National Center for Chronic Diseases Prevention and Health Promotion, Division for Heart Disease and Stroke Prevention. (2020, Feb 24). Prevent High Blood Pressure. Retrieved from https://www.cdc.gov/bloodpressure/prevent.htm

[23] National Center for Chronic Diseases Prevention and Health Promotion, Division for Heart Disease and Stroke Prevention. (2020, Feb 24). Prevent High Blood Pressure. Retrieved from https://www.cdc.gov/bloodpressure/manage.htm

[24] Andreas C.Muller, Sarah Guido. Introduction to Machine Learning with Python Books (A Guide for Data Scientists).