



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

CAPSTONE PROJECT (C3879C)

NAME: WONG QI YUAN, JEFFREY

**COURSE: SPECIALIST DIPLOMA IN
APPLIED ARTIFICIAL INTELLIGENCE
(INTAKE 5)**

WHAT PROBLEM ARE WE CURRENTLY FACING NOW AND THE FUTURE?

The World Health Organization (WHO) mentioned in a recent report 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 unresolved, 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 hypertension.

Furthermore, as the world currently facing the crisis of the COVID-19 pandemic, hence it would be 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 their communities. Also, based on the 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 sedentary since the COVID-19 pandemic.

Thus, this is an alarming concern in the road of the battle of the COVID-19 pandemic as the majority of the people are unaware of their overall obesity status and some of them do not have time to for a simple health screening.

OVERVIEW OF THE PROJECT CONTEXT



- Prior to training the algorithms, three datasets (i.e., diabetes, hypertension, and obesity) were downloaded from various sources.
- Followed by, the implementation of the **Isolation Forest** to detect and removed the potential outliers, and then performed data splitting in the ratio of 70:30 such that 70% training data and 30% testing data.
- As the classes are imbalanced for all the training sets, the **SMOTE-TOMEK technique** was used to balance the training data distribution to minimized the biased and unreliable outcomes on the machine learning model performance.
- Implementation of the **classifiers algorithms (random forest, gradient boosting , and extra trees)** with hyperparameters optimization to predict the respective possible risk of developing obesity-related chronic diseases such as obesity, diabetes, and hypertension.
- To simulate the real-life practical application, a **web application was implemented using Streamlit** with the integration of the respective proposed classifier models to predict the likelihood of developing of the respective obesity-related chronic diseases.

BENEFITS OF USING THE HEALTH APP

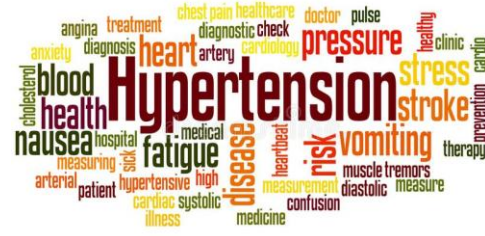
- It would provide an individual to know their **likelihood (probability)** of being developing the obesity-related chronic-diseases such as diabetes, hypertension, and obesity, and hence, **reduce their time** to visit the office clinic in person.
- Being **cost-effective**, the users are **able to 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.
- **Who are the target users?**
 - Anyone who are self-conscious about own health status.
 - Anyone who seldom or have no intention to visit the clinics or hospitals on regular basis for health screening.



BASIC CONCEPTS OF OBESITY-RELATED CHRONIC DISEASES



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



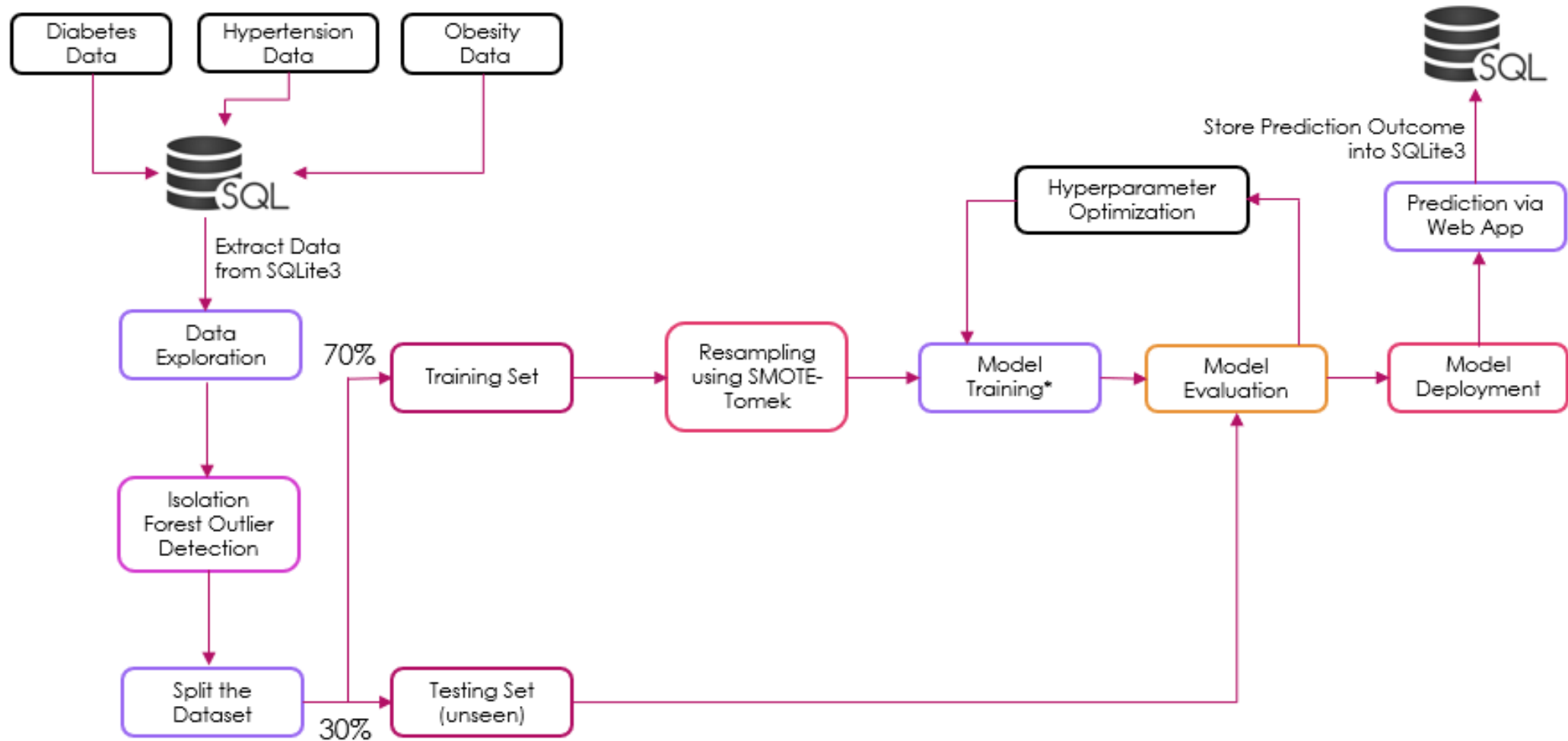
Hypertension or High Blood Pressure is a blood pressure that is usually higher than the global health recommended blood pressure ($< 140/120$ mmHg). However, high blood pressure that is consistently above the recommended range may result in kidney failure, stroke, heart diseases, etc.



Obesity can be defined as a weight that is higher than what is considered healthy for a given weight. Obesity is a serious diseases as it is often related with poor mental health and reduced quality of life, and also leading causes of death, diabetes, and high blood pressure.



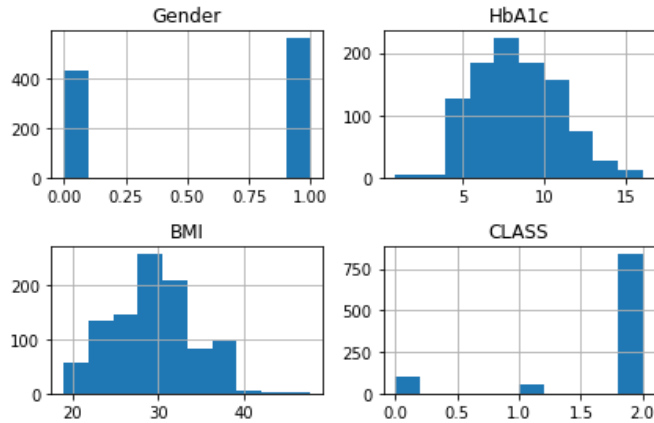
MACHINE LEARNING WORKFLOW



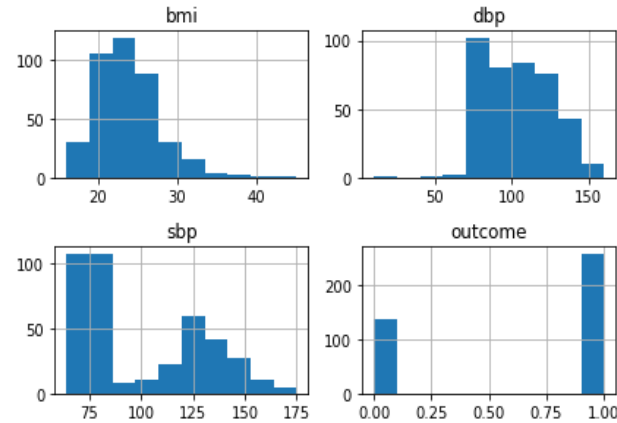
* **Model Training:** Random Forest Classifier, Gradient Boosting Classifier, and Extra Trees Classifier

1) DATA EXPLORATION (HISTOGRAM)

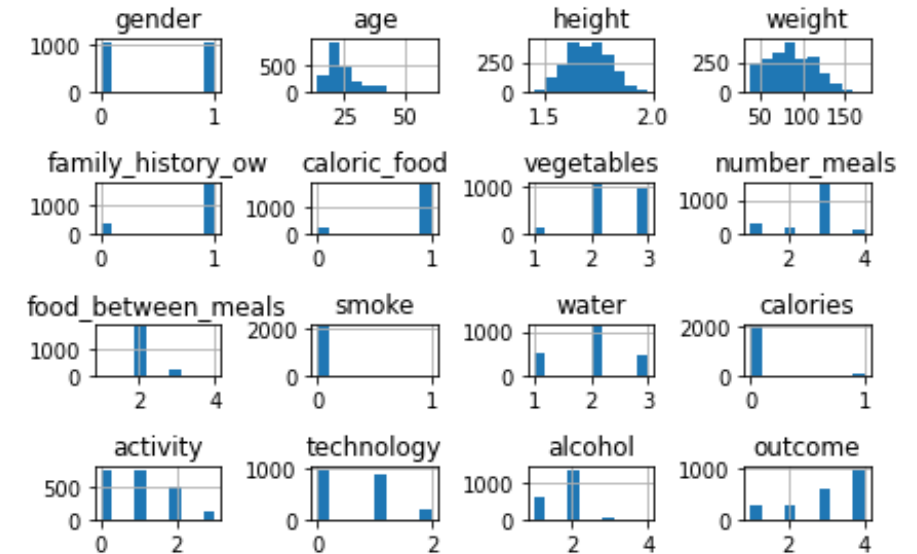
Diabetes



Hypertension

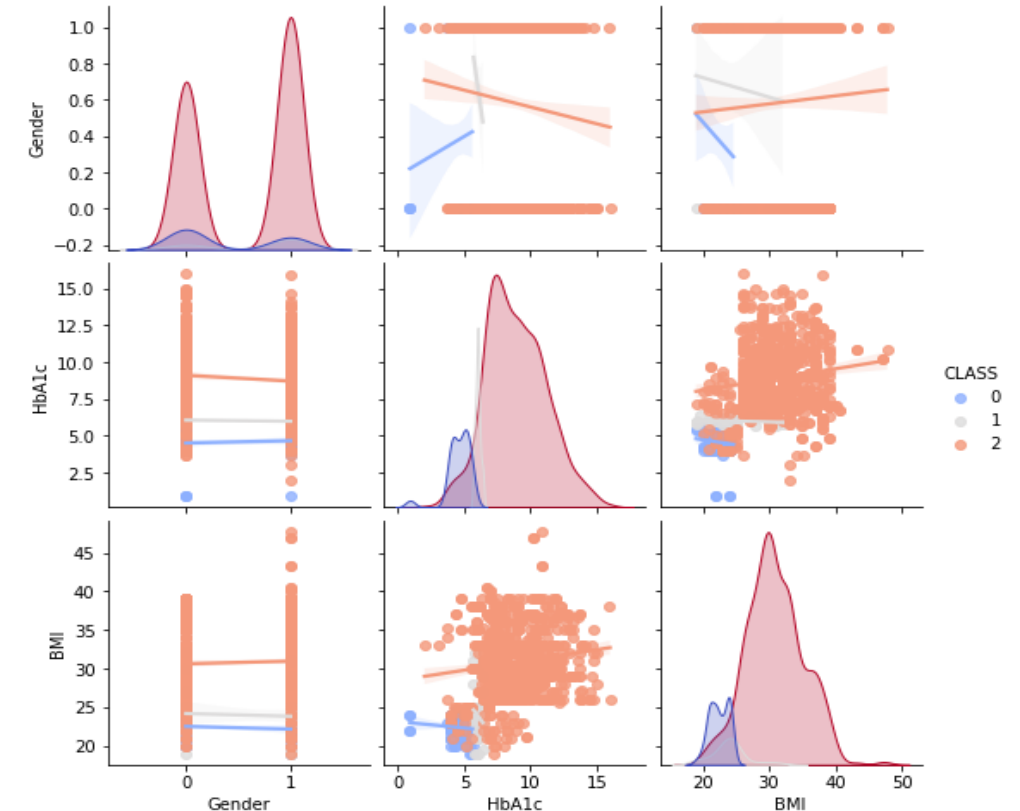
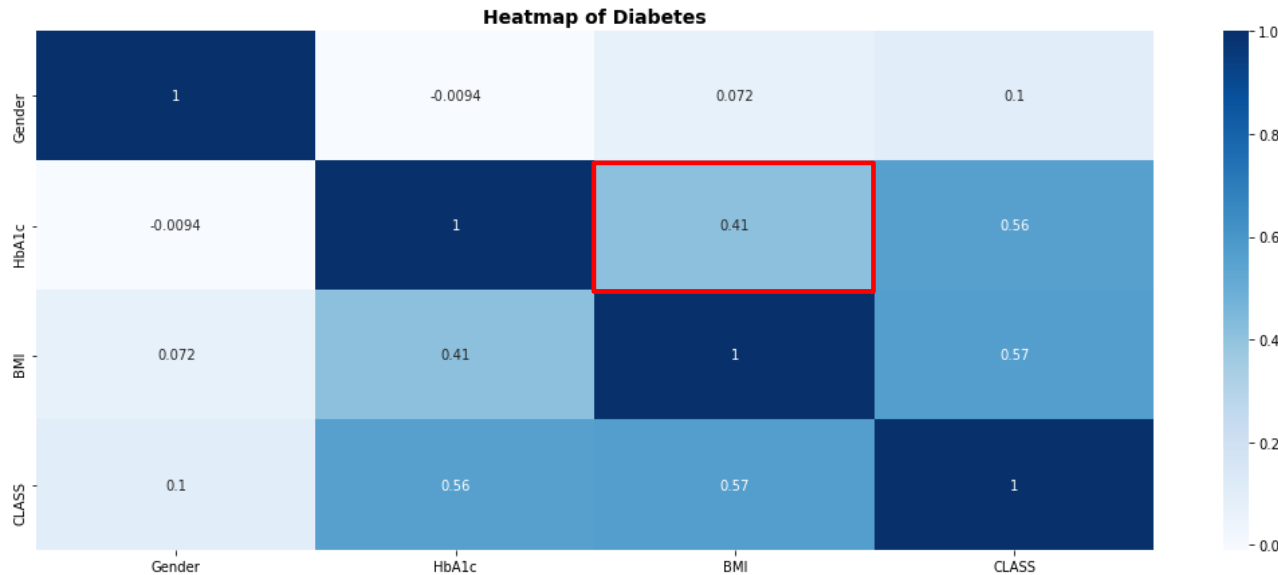


Obesity



- Based on all the above histogram subplots shown above, 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 such as power transform, min max scaler, and/ or standard scaler at the modelling stage.

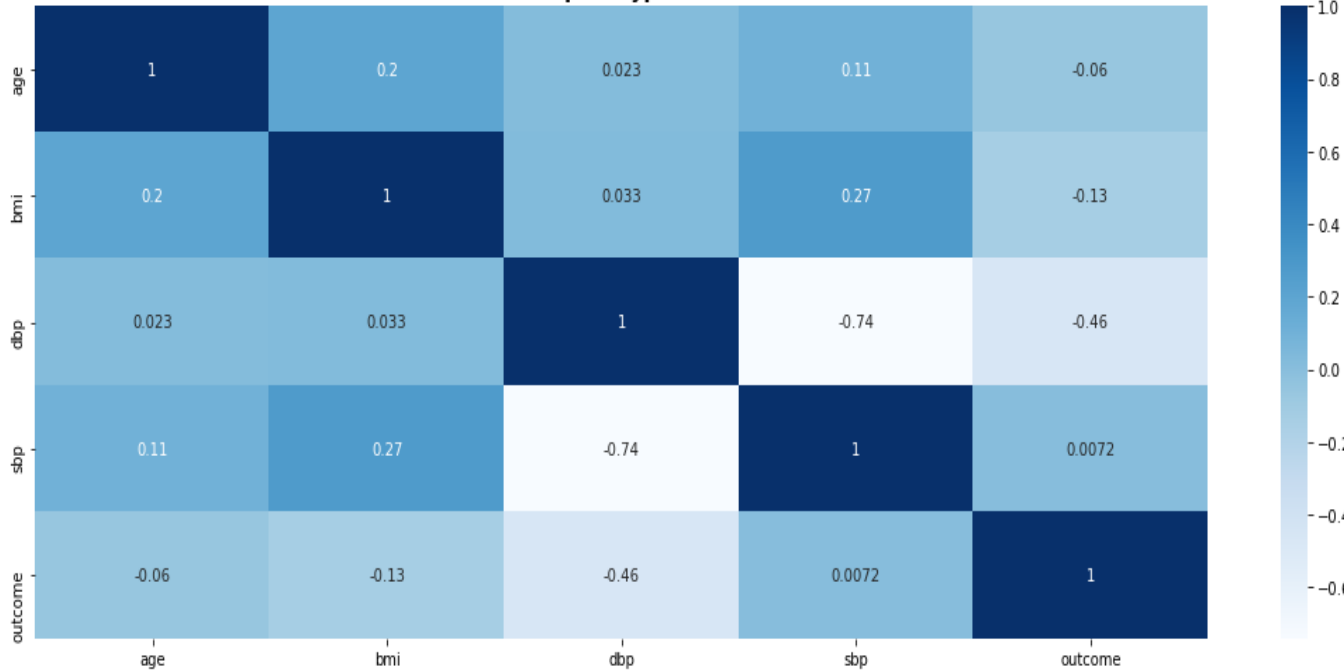
1) DATA EXPLORATION (HEATMAP AND PAIR-PLOT) - DIABETES



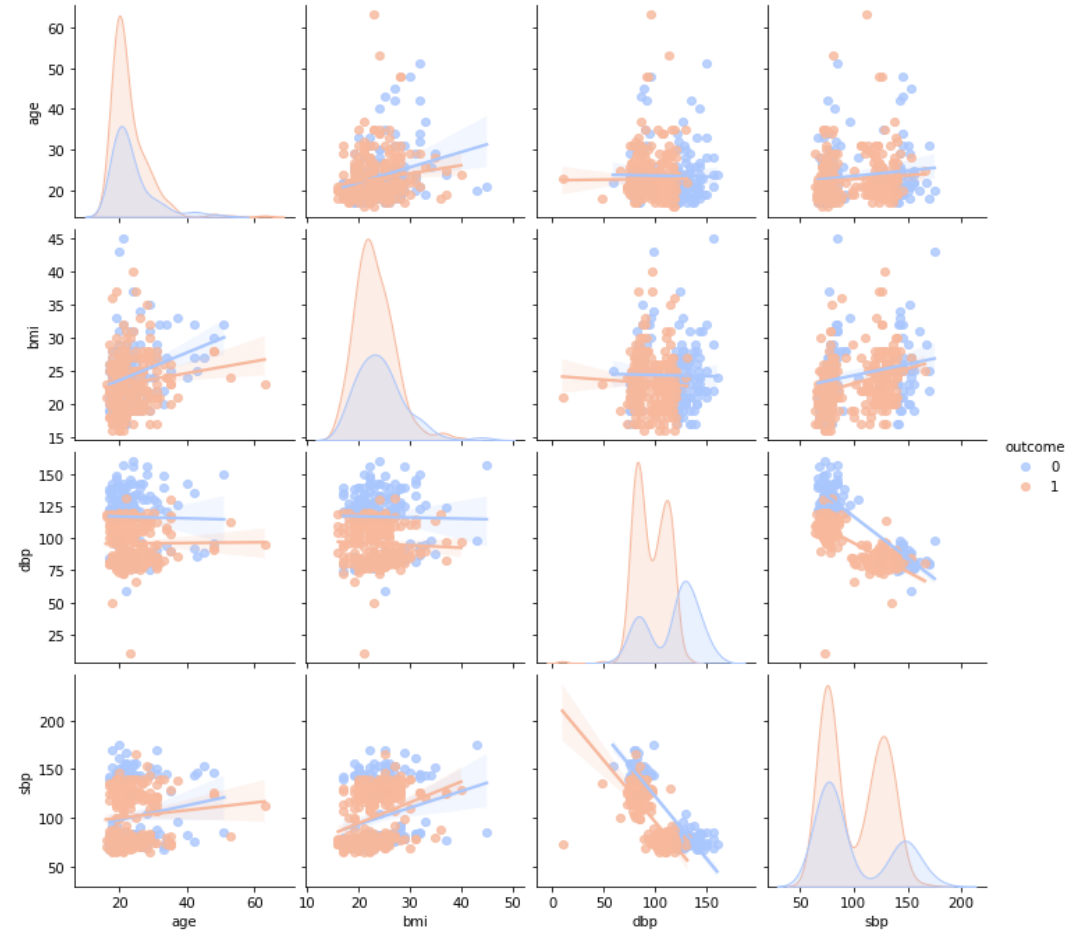
- Based on the heatmap and pair-plot, both show that there is a **weak and positive correlation between HbA1c and BMI attribute**, but the impact on the overall algorithm's performance is generally small.

1) DATA EXPLORATION (HEATMAP AND PAIR-PLOT) - HYPERTENSION

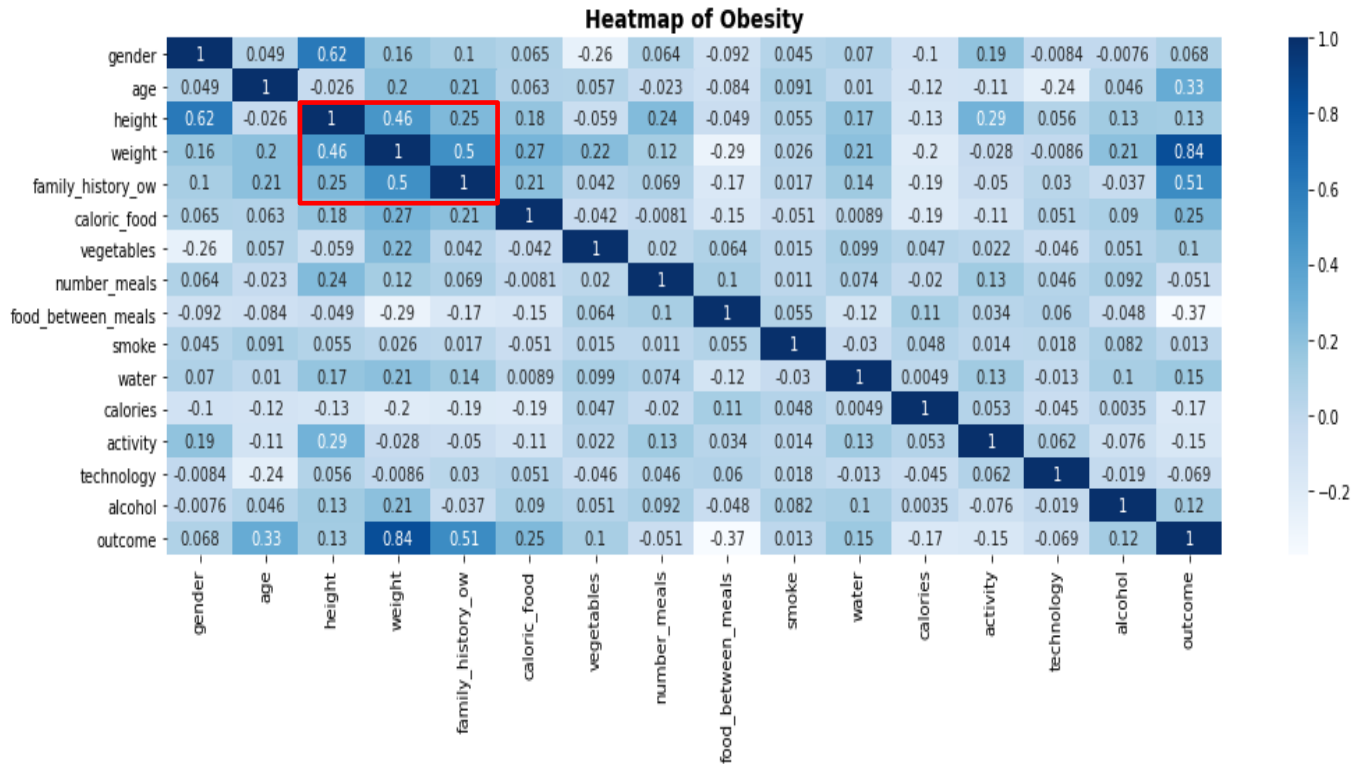
Heatmap of Hypertension



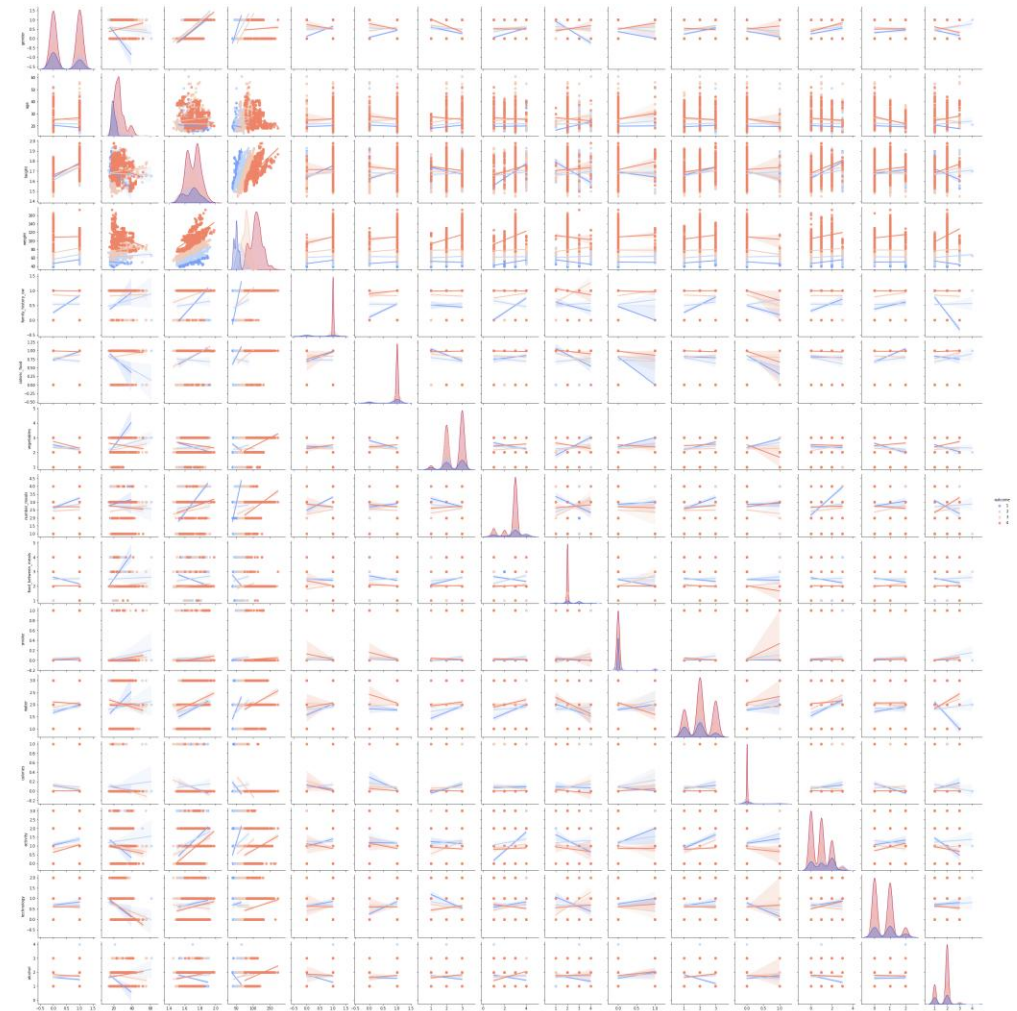
- Based on the heatmap and pair-plot, both show that there is a **no significant correlation between variables.**



1) DATA EXPLORATION (HEATMAP AND PAIR-PLOT - OBESITY)

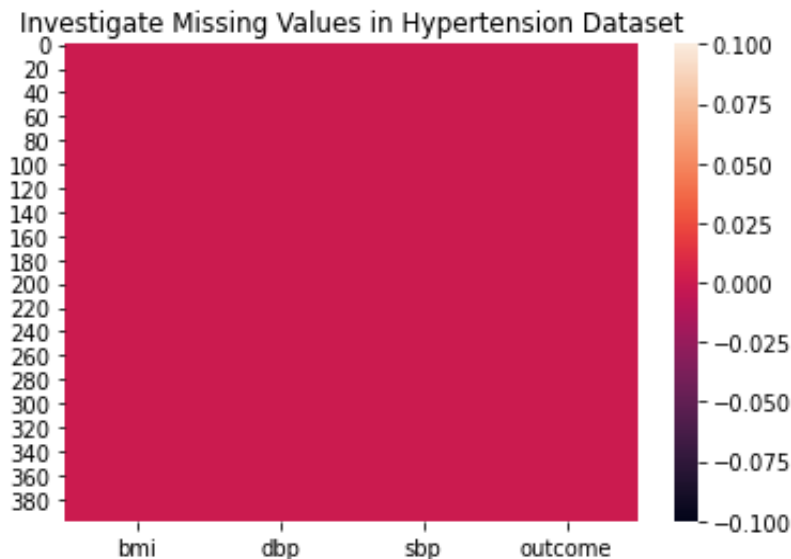
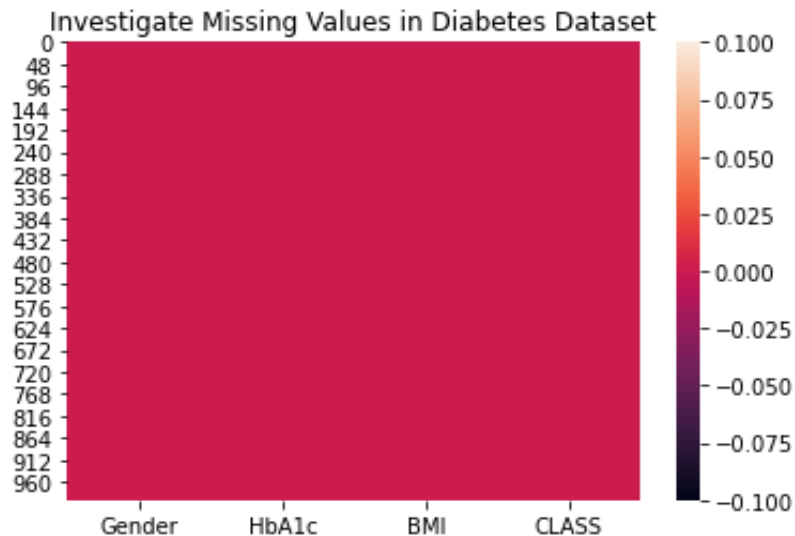


- Based on the heatmap and pair-plot, both show that there is a **weak and positive correlation between height, weight, and family history with overweight attributes**, but the impact on the overall algorithm's performance is also generally very small.



2) DATA CLEANSING – INVESTIGATION OF MISSING DATA

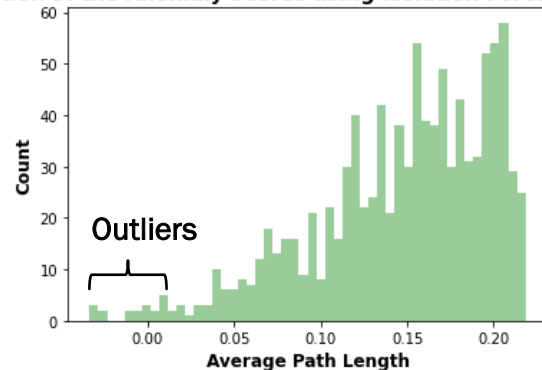
- Prior to the algorithms training, it is imperative to investigate for missing data during the pre-processing steps as not all machine learning algorithms support missing values and might have a chance of leading to inaccurate and bias outcome.
- Based on the heatmaps shown below, there were no missing row data found in all three datasets.



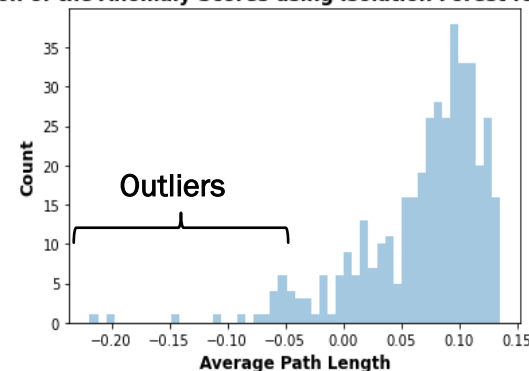
3) DATA CLEANSING – REMOVAL OF POTENTIAL OUTLIERS USING ISOLATION FOREST

- Isolation Forest is the anomaly detection algorithm that identifies anomalies using Isolation. As such, the isolation forest was used to identify the potential outliers data in the respective datasets.
- Basically, the iForest 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.

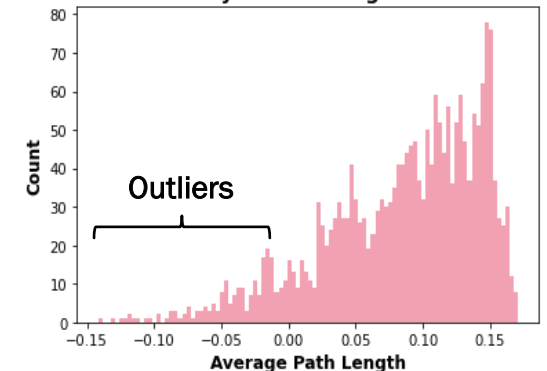
Distribution of the Anomaly Scores using Isolation Forest for Diabetes



Distribution of the Anomaly Scores using Isolation Forest for Hypertension



Distribution of the Anomaly Scores using Isolation Forest for Obesity



- Based on the anomaly scores distribution shown above, the outliers with the lower anomaly scores are likely to be outliers. As such, the outlier threshold on the score can be set by the user preference to differentiate between the outlier and inlier. In this case, the outlier threshold was set as below -0.01 for all data aspects.
- Thereafter, the outlier data from each dataset was removed, and the remaining data were used for further modelling algorithms.

4) DATA SPLITTING

- All the respective pre-processed datasets were divided into 70% training set and 30% testing set.

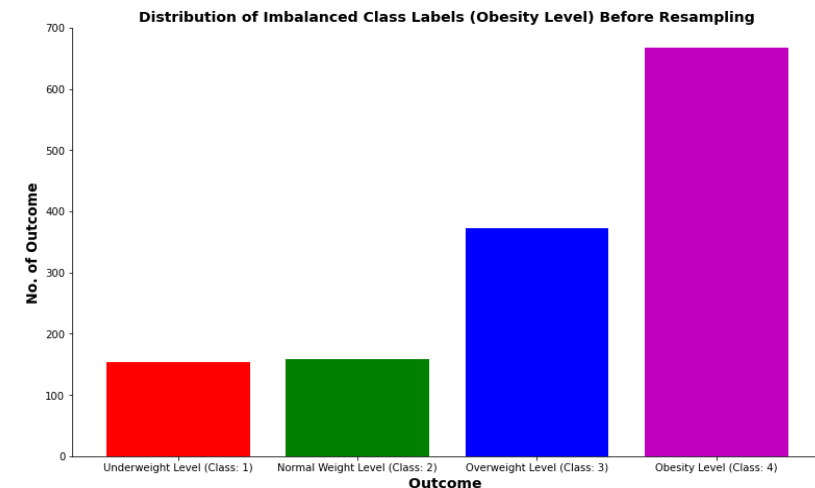
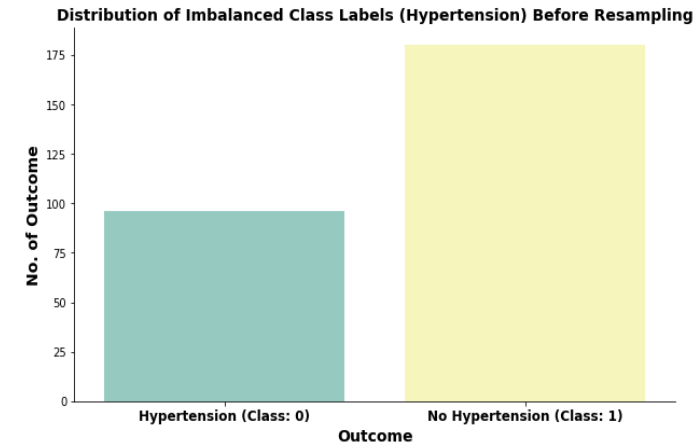
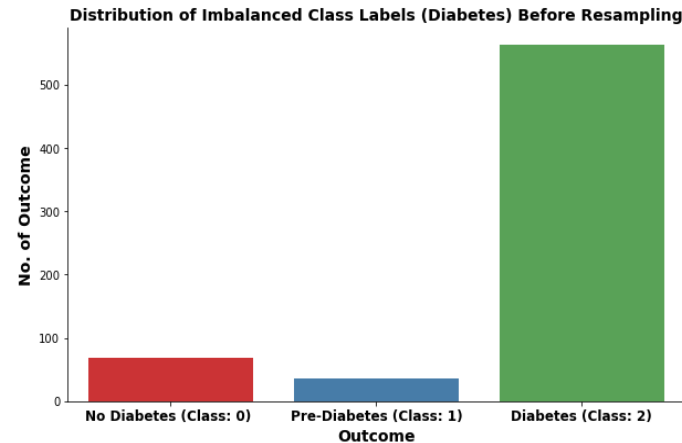


- A sample code snippet of dividing the datasets.

```
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)  
233
```


5) RESAMPLING TECHNIQUE (SMOTE-TOMEK)

- After splitting the datasets, it was also observed that **all of the classes present in the respective training sets were imbalanced**. Hence, it is also imperative to balance the datasets prior to the modelling pipeline.
- This is because if the imbalanced training data is not treated promptly prior to the modelling pipeline, 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 in 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.



5) RESAMPLING TECHNIQUE (SMOTE-TOMEK)

- Even though SMOTE is one of the famous oversampling techniques and is very effective in handling class imbalance. The idea is to **combine the SMOTE with Tomek to increase the effectiveness of handling the imbalanced class in training set.**
- Basically, **SMOTE** is an oversampling technique that creates synthetic minority class data points to balance the training set. It works using a k-nearest neighbor algorithm to create synthetic data points.
- As for **Tomek Links**, it is an under-sampling approach that identifies all the pairs of data points that are nearest to each other but belong to different class, and these pairs are termed as Tomek Links. If these Tomek link points are present on the boundary of separation of the two classes. So **removing the majority class of Tomek links points increases the class separation, and reduces the number of majority class samples along the boundary of the majority cluster.**



5) RESAMPLING TECHNIQUE (SMOTE-TOMEK)

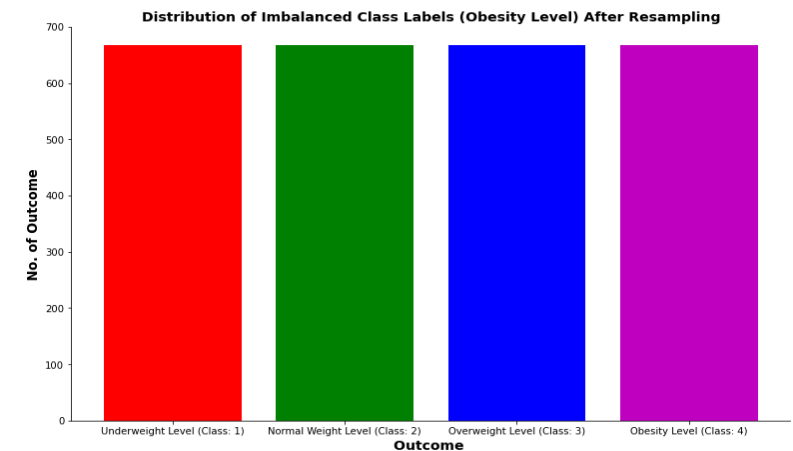
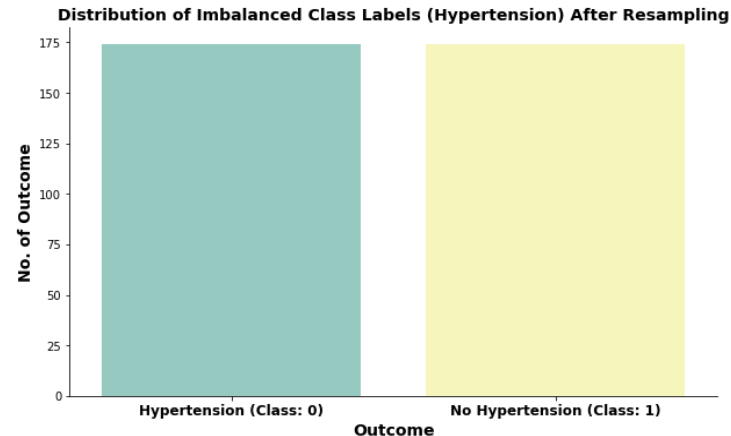
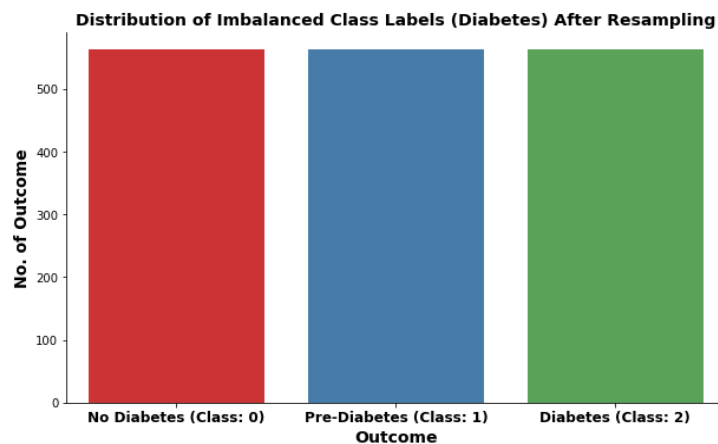
- Below shows a sample code snippet of the implementation of SMOTE and Tomek Links using the Imblearn package.

A sample code snippet of resampling technique (SMOTE-TOMEK)

```
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)
```

- Below are the outcomes of the balanced classes in the training sets using SMOTE-Tomek Technique.



6) BASELINE MODEL SELECTION & EVALUATION

- 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 training datasets.
- A sample code snippet of the baseline using the default parameters.

```
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))]
```

- A **model pipeline** was also built by providing the list of steps. Each step represents a tuple containing a string of name and an instance of an estimator.
- A sample code snippet of the model pipeline of **hypertension**.

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

- A sample code snippet of the model pipeline of **obesity and diabetes**.

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

6) BASELINE MODEL SELECTION & EVALUATION

- 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.
- **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 (validation) 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.**

A sample code snippet of Shuffle-Split Cross-Validation for Baseline Models

```
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)
```


6) BASELINE MODEL SELECTION & EVALUATION

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

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.

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

The **random forest classifier** was selected for **hypertension** because of its model characteristics such that it is **less biased and exhibits moderate variance in nature**.

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

Similar to hypertension, the **random forest classifier** was selected for obesity because of its model characteristics such that it is **less biased and exhibits moderate variance in nature**.



7) HYPERPARAMETER OPTIMIZATION

- The objective of hyperparameter optimization is to find the **most optimal hyperparameters** in machine learning classifiers.
- Before the hyperparameter optimization process, a **model pipeline object** was first built by providing it with a list of steps and each step represents a tuple containing a string of name and an instance of an estimator.
- Below shows the sample code snippets for the respective obesity-related chronic diseases with the chosen baseline models.

A sample code snippet of model pipeline (Diabetes)

```
397 # #build pipeline
398 pipe = Pipeline([('scaler', StandardScaler()), ('gbc', GradientBoostingClassifier(random_state = 42,
399                                         max_leaf_nodes = None))])
400
```

A sample code snippet of model pipeline (Hypertension)

```
386 #build pipeline
387 pipe = Pipeline([('scaler', MinMaxScaler()), ('power', PowerTransformer()), ('rfc', RandomForestClassifier(random_state = 42,
388                                                         max_leaf_nodes = None,
389                                                         max_features = "sqrt"))])
390
```

A sample code snippet of model pipeline (Obesity)

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

7) HYPERPARAMETER OPTIMIZATION

- Next, it is the **construction of the parameter grid** such that it is to specify for each parameter the step, 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_estimator`” was used as key in the parameter grid dictionary, and similar for the rest of the other parameters.
- Below shows the sample code snippets for the construction of the parameter grid for each respective chosen baseline model.

A sample code snippet of model pipeline (Diabetes)

```
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]}
```

A sample code snippet of model pipeline (Hypertension)

```
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)}
```

A sample code snippet of model pipeline (Obesity)

```
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)}
```

For **Gradient Boosting Classifier**, the most important parameters to be optimized are the number of trees **n_estimators** and **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 as random forest, a higher `n_estimators` are always better. But however, increasing `n_estimators` in gradient boosting will lead to a model complexity, which may lead to overfitting. Furthermore, another important parameter is **max_depth**, it is to minimize 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.

For **Random Forest Classifier**, the most important parameters to be optimized 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.

7) HYPERPARAMETER OPTIMIZATION

- Finally, 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 Min-Max-Scaler or Standard Scaler is refit only with the training splits and no information was leaked from the testing split into the parameter search.
- As mentioned in the earlier slides, the testing split was not based on the testing set. It is the defined amount of the % split from the training set.
- Furthermore, to get a finer control over the cross-validation, the CV parameter in the GridSearchCV function uses Shuffle-Split and the amount of split were defined as 80% training and 20% testing (validation) sets.
- On the right, it shows a sample code snippet of the hyperparameter optimization.

A sample code snippet of Grid Search CV (Hyperparameter Optimization)

```
121 def hyperparameters_tuning(pipe, param_grid, X, y):
122
123     # split the dataset
124     shuffle_split = ShuffleSplit(test_size = 0.20, train_size = 0.80, n_splits = 5, random_state = 0)
125     # perform hyperparameter tuning using gridsearchcv
126     gs_res = GridSearchCV(pipe, param_grid = param_grid,
127                           cv = shuffle_split, scoring = 'accuracy', verbose = 3)
128     # fitted the model for grid search
129     gs_res.fit(X, y)
130     return gs_res

```

```
396 ##### Hyperparameter Optimization on Gradient Boosting #####
397 # #build pipeline
398 pipe = Pipeline([('scaler', StandardScaler()),('gbc', GradientBoostingClassifier(random_state = 42,
399                                         max_leaf_nodes = None))])
400
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]}
405
406 # call the function to perform hyperparameter tuning
407 gridResult = hyperparameters_tuning(pipe, param_grid, X_smt, Y_smt)
408
409 # display the best parameters after tuning
410 print("\nHyperparameter Optimization Process for Gradient Boosting Classifier (Diabetes): ")
411 print("=====")
412 print("The best score is {}".format(gridResult.best_score_))
413 print("The best params is {}".format(gridResult.best_params_))
414 print("The best estimator is {}".format(gridResult.best_estimator_))
415
416 # store the best params into Series
417 best_params_df = pd.Series(gridResult.best_params_)
418 print(best_params_df)

```

7) HYPERPARAMETER OPTIMIZATION

- Below show the **outcome of the hyperparameter optimization** for the respective classifier model for respective obesity-related chronic diseases.

Outcome of the Hyperparameter Optimization for Gradient Boosting Classifier (Diabetes)

```
Hyperparameter Optimization Process for Gradient Boosting Classifier (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))])
```

Outcome of the Hyperparameter Optimization for Random Forest Classifier (Hypertension)

```
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))])
```

Outcome of the Hyperparameter Optimization for Random Forest Classifier (Obesity)

```
Hyperparameter Optimization Process for Random Forest Classifier (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))])
```


8) RETRAINED MODEL WITH BEST HYPERPARAMETER AND EVALUATION

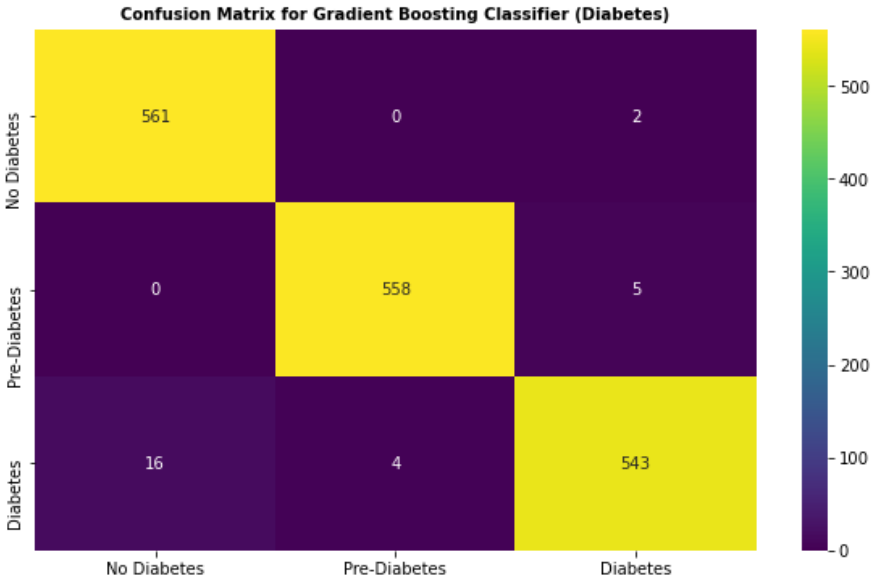
■ Diabetes (Gradient Boosting Classifier)

Before Hyperparameter Optimization

```
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
```

After Hyperparameter Optimization

```
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
```



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

With the best hyperparameters, the Gradient Boosting Classifier for Diabetes produces a very high accuracy rate of 99%. **High precision rates** also relates to the **low-false positive rate**, which is aggregable with the confusion matrix calculation shown on the right, which is about less than 5% chances for all classes. Also, **high recall rates** also relates to the **low-false negative rate**, which can be also proven from the confusion matrix, which is also about less than 5% chances for all classes. **ROC-AUC** with 99% means that the Gradient Boosting Classifier has the excellent capability to differentiate between classes (i.e., No Diabetes, Pre-Diabetes, and Diabetes).

8) RETRAINED MODEL WITH BEST HYPERPARAMETER AND EVALUATION

■ Hypertension (Random Forest Classifier)

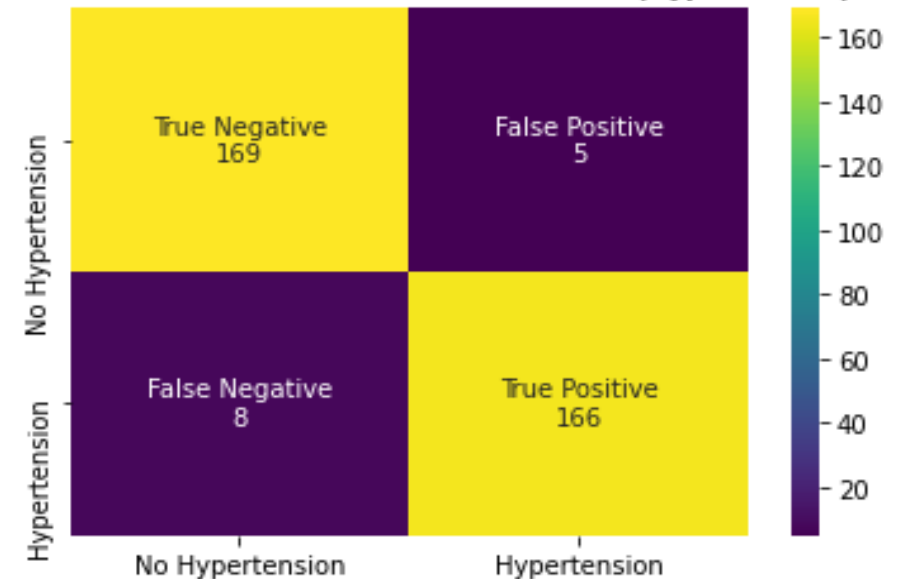
Before Hyperparameter Optimization

```
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
```

After Hyperparameter Optimization

```
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
```

Confusion Matrix for Random Forest Classifier (Hypertension)



Prior to the hyperparameter optimization, it was observed that the performance metrics for the training sets is slightly higher than the validation sets. This phenomenon is known as **overfitting**.

With the best hyperparameters, the overfitting issues has been **minimized** and the Random Forest Classifier has yielded with an accurate rate of 97%.

High precision rates also relates to the low-false positive rate, which is about less than 2% chances for all classes. Also, **high recall rates** also relates to the low-false negative rate, which can be also proven from the confusion matrix, which is also about less than 2% chances for all classes.

ROC-AUC with 98% means that the Random Forest Classifier has the excellent capability to differentiate between classes (i.e., No Hypertension and Hypertension).

8) RETRAINED MODEL WITH BEST HYPERPARAMETER AND EVALUATION

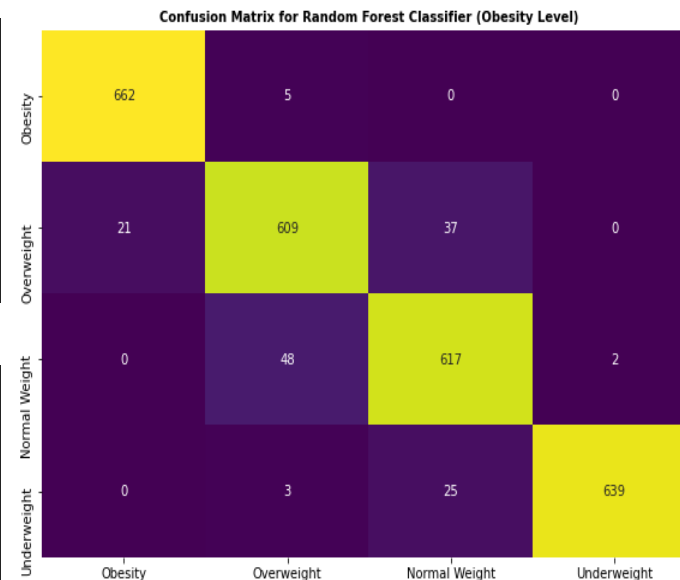
■ Obesity (Random Forest Classifier)

Before Hyperparameter Optimization

```
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
```

After Hyperparameter Optimization

```
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
```



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

Prior to the hyperparameter optimization, it was observed that the performance metrics for the training sets is slightly higher than the validation sets. This phenomenon is known as **overfitting**.

With the best hyperparameters, the overfitting issues has been **minimized** and the Random Forest Classifier has yielded with an accurate rate of 97% and ROC-AUC of 99% which has an excellent capability to distinguishes between classes.

High precision rates also relates to the low-false positive rate, which is about less than 2% chances for all classes. Also, **high recall rates** also relates to the low-false negative rate, which can be also proven from the confusion matrix, which is also about less than 2% chances for all classes.

9) EVALUATION OF THE HYPERPARAMETER TRAINED 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 with the best hyperparameters**.
- 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.



9) EVALUATION ON THE OPTIMIZED HYPERPARAMETER MODEL USING TESTING SET

■ Diabetes (Gradient Boosting Classifier)

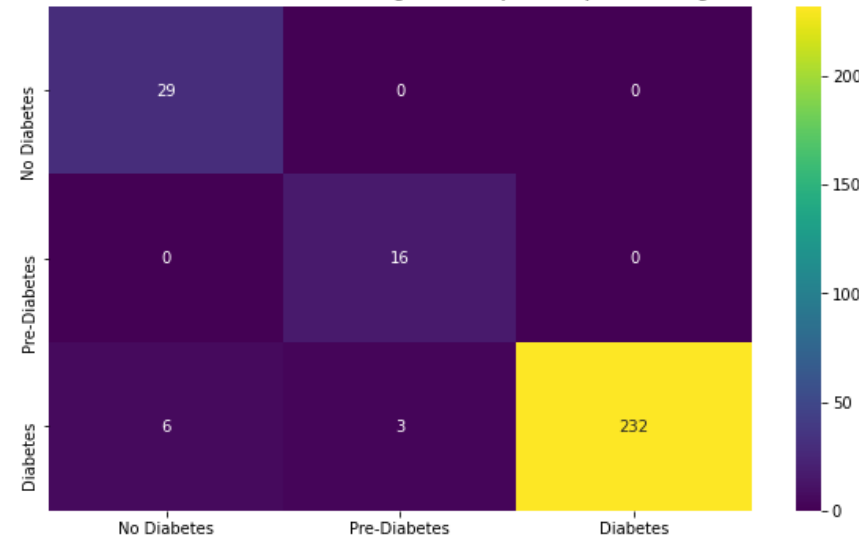
Performance Metrics of the Gradient Boosting with Hyperparameter Tuning (Diabetes) using Testing Set:

```
=====
Accuracy Score: 0.9685314685314685
Precision Score: 0.8902255639097745
Recall Score: 0.9875518672199171
F1 Score: 0.9338360767139836
ROC Score: 0.996394632040826
```

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

Confusion Matrix for Gradient Boosting Classifier (Diabetes) from Testing Set



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

High ROC score: This model has an excellent capability to differentiate between classes (i.e., No Diabetes, Pre-Diabetes, Diabetes).

High Precision Rate: This means that the chances of misdiagnosed of diabetes (false-positive) is generally very low, which can be proven by the confusion matrix calculation.

High Recall Rate: This means that the chances of misdiagnosed of diabetes (false-negative) is also generally very low, which can also be proven by the confusion matrix calculation.

High Accuracy: This model has 97% of predicting the classes correctly with reference to the unseen testing set.

Overall, the use of Gradient Boosting Classifier to predict the likelihood of diabetes is generally acceptable.

9) EVALUATION ON THE OPTIMIZED HYPERPARAMETER MODEL USING TESTING SET

■ Hypertension (Random Forest Classifier)

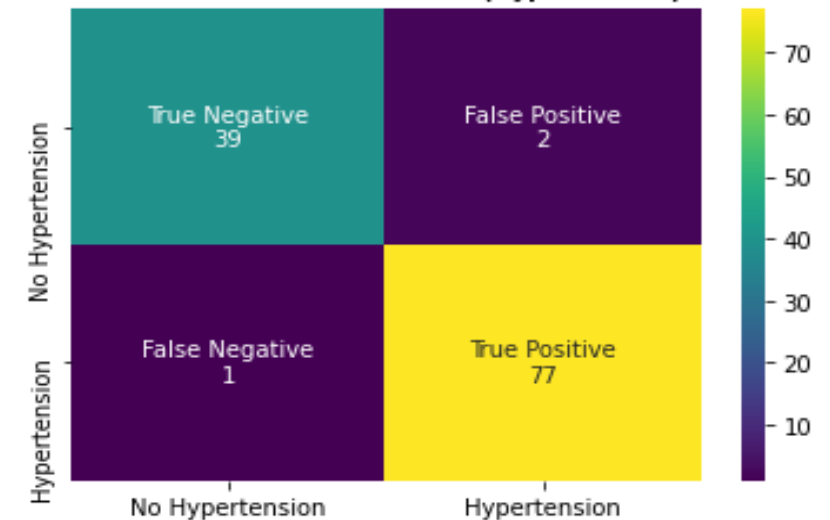
Performance Metrics of the Random Forest with Hyperparameter Tuning (Hypertension) using Testing Set:

```
=====
Accuracy Score: 0.9747899159663865
Precision Score: 0.9748417721518987
Recall Score: 0.9691994996873046
F1 Score: 0.971927341354093
ROC Score: 0.9691994996873046
```

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

Confusion Matrix for Random Forest Classifier (Hypertension) from Testing Set



High ROC score: This model has an excellent capability to differentiate between classes (i.e., Hypertension, No Hypertension).

High Precision Rate: This means that the chances of misdiagnosed of hypertension (false-positive) is generally very low, which can be proven by the confusion matrix.

High Recall Rate: This means that the chances of misdiagnosed of hypertension (false-negative) is also generally very low, which can also be proven by the confusion matrix.

High Accuracy: This model has 97% of predicting the classes correctly with reference to the unseen testing set.

Overall, the use of Random Forest Classifier to predict the likelihood of hypertension is generally acceptable.

9) EVALUATION ON THE OPTIMIZED HYPERPARAMETER MODEL USING TESTING SET

■ Obesity (Random Forest Classifier)

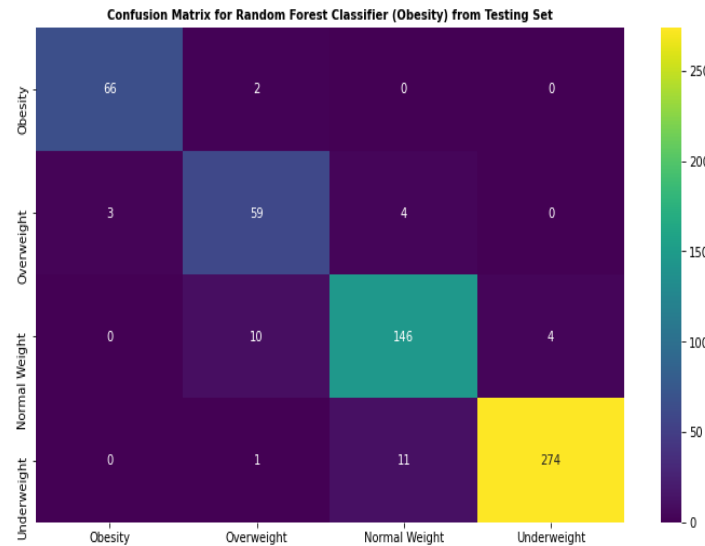
Performance Metrics of the Random Forest with Hyperparameter Tuning (Obesity) using Testing Set:

=====

Accuracy Score: 0.9396551724137931
Precision Score: 0.9171024981257231
Recall Score: 0.9337673968188673
F1 Score: 0.9249661599872964
ROC Score: 0.987532006419606

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



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

High ROC score: This model has an excellent capability to differentiate between classes (i.e., Underweight, Normal Weight, Overweight, Obesity).

High Precision Rate: This means that the chances of misdiagnosed of obesity level (false-positive) is generally very low, which can be proven by the confusion matrix.

High Recall Rate: This means that the chances of misdiagnosed of obesity-level (false-negative) is also generally very low, which can also be proven by the confusion matrix.

High Accuracy: This model has 94% of predicting the classes correctly with reference to the unseen testing set.

Overall, the use of Random Forest Classifier to predict the likelihood of obesity level is generally acceptable.

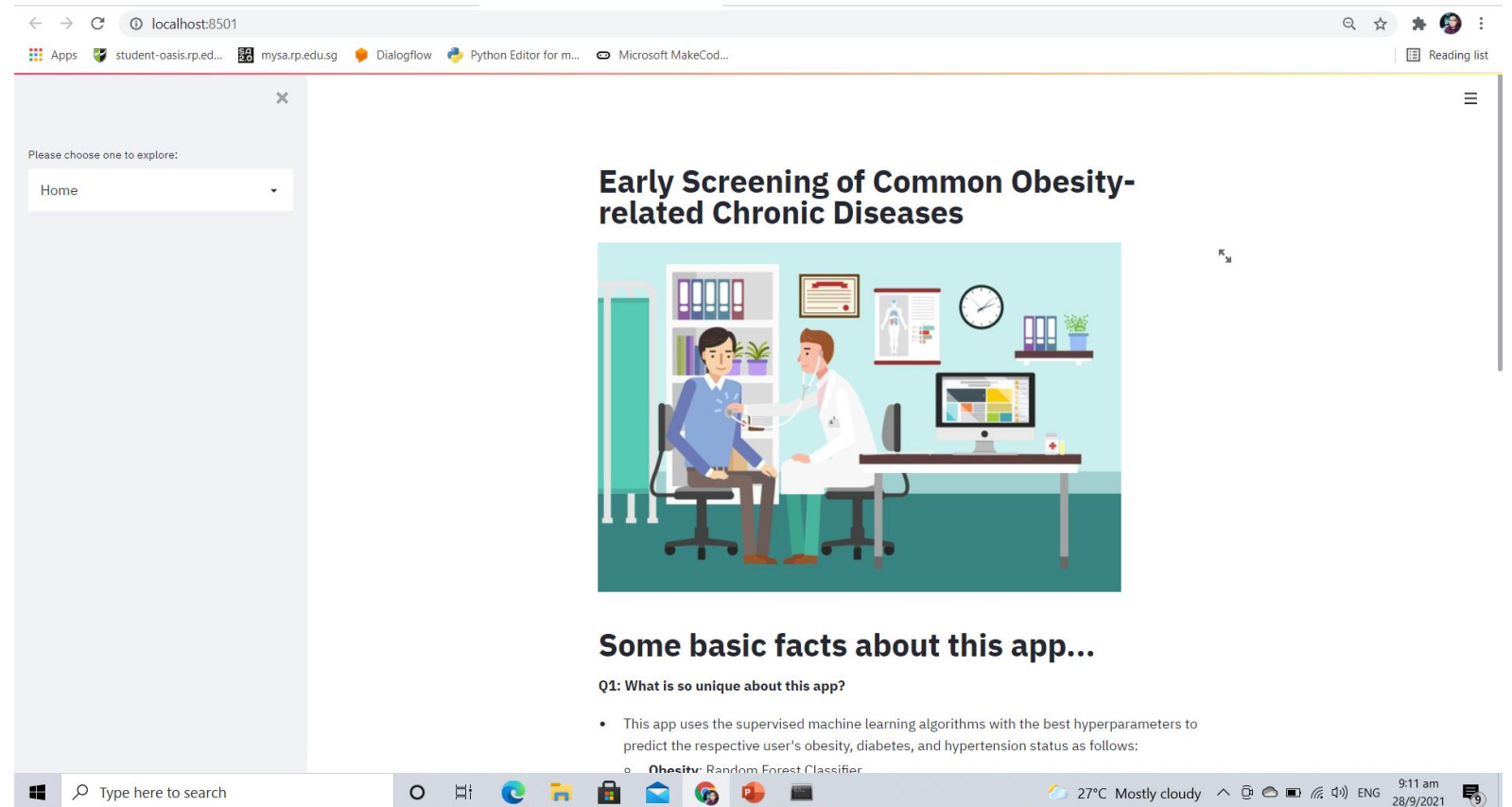
10) DEVELOPMENT OF HEALTH WEB APPLICATION



- All the proposed ensemble machine learning algorithms that have been optimized and trained in Python using Spyder IDE were saved in the **Pickle files**.
- **Streamlit** was used as the development and deployment of the health web application.
- All the proposed machine learning algorithms were integrated with the Streamlit App as its backbone to predict the respective obesity-related chronic diseases.



LIVE DEMO




localhost:8501

Please choose one to explore:

Home

Early Screening of Common Obesity-related Chronic Diseases



Some basic facts about this app...

Q1: What is so unique about this app?

- This app uses the supervised machine learning algorithms with the best hyperparameters to predict the respective user's obesity, diabetes, and hypertension status as follows:
 - Obesity: Random Forest Classifier

27°C Mostly cloudy 9:11 am 28/9/2021