Development of Health Web App with Integration of Ensemble Machine Learning for Early Diagnosis in Obesity

Wong Qi Yuan, Jeffrey (20053371)

Specialist Diploma in Applied Artificial Intelligence (Part-Time)

C3379C – Capstone Project 2021

Project Supervisor: Dr. Wee Kah Huat

Co-Supervisor: Dr. Ho Chee Wai

Problem Statement

The Word Health Organization (WHO) reports that obesity has reached epidemic proportions globally, with at least 2.8 million people dying each year as a result of being overweight or obese. Without immediate intervention, deaths may reach 52 million annually by 2030. Furthermore, COVID-19 accelerates this, with increased sedentary lifestyles and weight gain. As such, many people are unaware of their health status.

Source: https://www.who.int/news-room/facts-in-pictures/detail/6-facts-on-obesity
Source: https://www.straitstimes.com/singapore/health/people-in-singapore-less-healthy-and-covid-19-may-worsen-situation-national

Solution

A cost-effective and user-friendly web application, developed using Streamlit, incorporates with optimized ensemble machine learning model to predict the likelihood of having obesity, providing individuals as an effective means to monitor their health.

Project Objectives

1. Explore data processing techniques to handle missing values, outliers, non-gaussian like distribution, etc, and ensure the integrity of the dataset.
2. Explore the possibility of creating new features that may enhance the model's performance.
3. Explore and evaluate the impact of combine data sampling techniques such as SMOTE-Tomek, etc, for imbalanced class and determine the most effective approach.
4. Evaluate and compare any **three** ensemble machine learning algorithms such as gradient boosting, decision trees, random forest, etc, and choose **one** machine learning algorithms with the most suitable model for early diagnosis of obesity.
5. Fine-tune the hyperparameters of the chosen model to maximize predictive accuracy and efficiency.
6. Implement an appropriate cross-validation technique for data splitting and to assess the generalizability of the model and validate its performance during model selection and fine-tuning process.
7. Design and use appropriate evaluation metrics such as precision, f1-score, etc, to assess the performance of the model, with a specific focus on achieving high sensitivity in early diagnosis of obesity.
8. Develop an intuitive and user-friendly interface for general user to interact with the machine learning application.
9. Explain the rationale behind for all the findings with aid of visual media such as poster, PowerPoint presentation slides, etc.
10. Document all codes with appropriate comments.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 9, 2019, accepted September 27, 2019, date of publication October 2, 2019, date of current version October 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2945129

Development of Disease Prediction Model Based on Ensemble Learning Approach for Diabetes and Hypertension

NORMA LATIF FITRIYANI[1], MUHAMMAD SYAFRUDIN[1], GANJAR ALFIAN[2], AND JONGTAE RHEE[1]

[1]Department of Industrial and Systems Engineering, Dongguk University, Seoul 04620, South Korea
[2]Nano Information Technology Academy, Dongguk University, Seoul 04626, South Korea

Corresponding authors: Muhammad Syafrudin (udin@dongguk.edu) and Jongtae Rhee (jtrhee@dongguk.edu)

This work was supported by the Ministry of Trade, Industry and Energy (MOTIE) and the Korea Institute for Advancement of Technology (KIAT) through the International Cooperative Research and Development Program under Grant N0002301.

ABSTRACT Early diseases prediction plays an important role for improving healthcare quality and can help individuals avoid dangerous health situations before it is too late. This paper proposes a disease prediction model (DPM) to provide an early prediction for type 2 diabetes and hypertension based on individual's risk factors data. The proposed DPM consists of isolation forest (iForest) based outlier detection method to remove outlier data, synthetic minority oversampling technique tomek link (SMOTETomek) to balance data distribution, and ensemble approach to predict the diseases. Four datasets were utilized to build the model

Software, Libraries & Frameworks

python™

pandas

NumPy

scikit learn

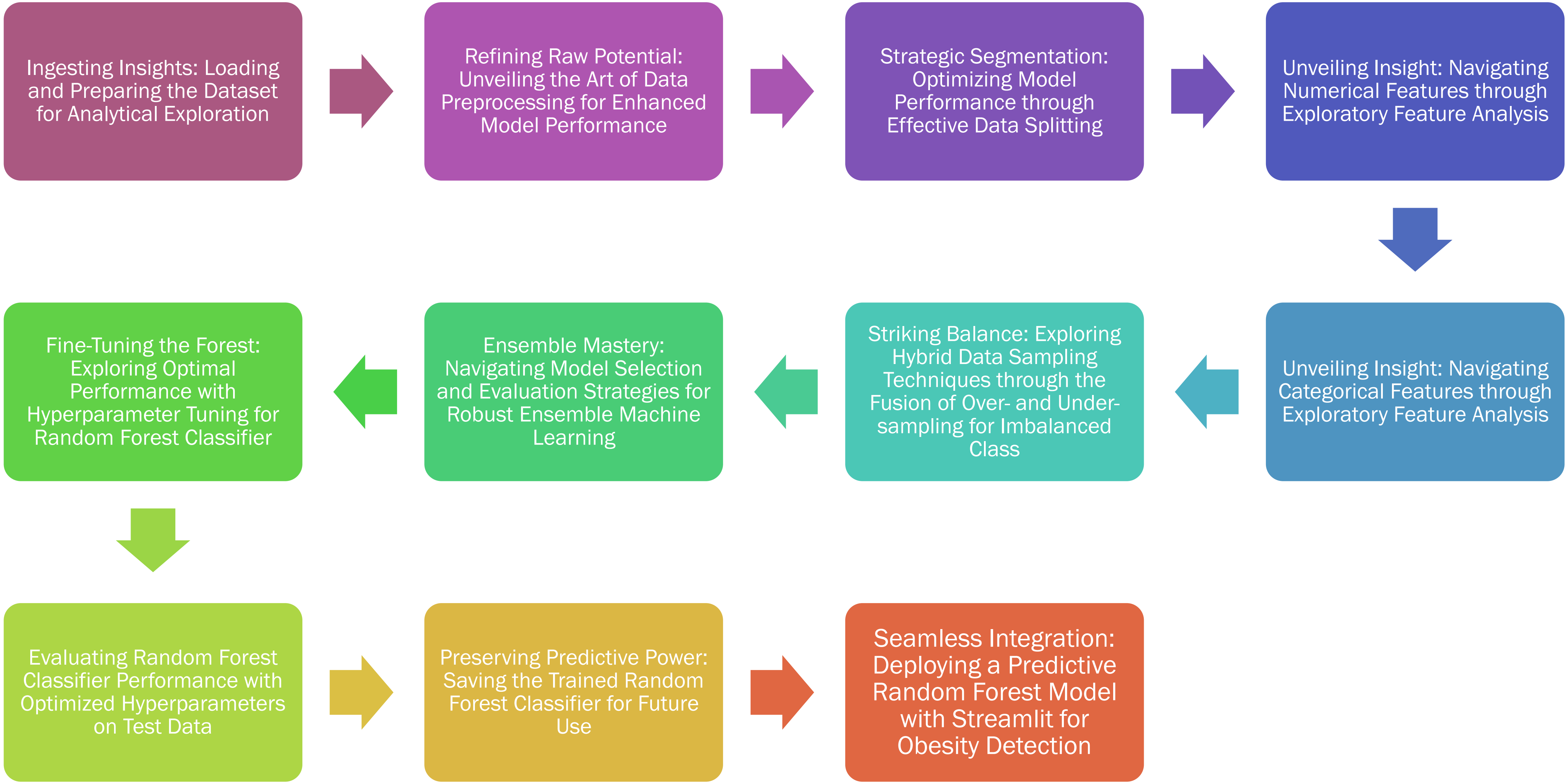
matplotlib

seaborn

SciPy

Streamlit

Project Roadmap



Ingesting Insights: Loading and Preparing the Dataset for Analytical Exploration

Data Loading

```
1 # read the obesity datafile from the Google drive
2 obesity_df = pd.read_csv('/content/drive/MyDrive/rp_capstone_project/obesity.csv')
```

	gender	age	height	weight	family_history_with_overweight	caloric_food	vegetables	number_meals	food_between_meals	smoke	water	ca
0	Female	21	1.62	64.0	yes	no	2	3	2	no	2	
1	Female	21	1.52	56.0	yes	no	3	3	2	yes	3	
2	Male	23	1.80	77.0	yes	no	2	3	2	no	2	
3	Male	27	1.80	87.0	no	no	3	3	2	no	2	
4	Male	22	1.78	89.8	no	no	2	1	2	no	2	

calories	activity	technology	alcohol	transportation	obesity_level
no	0	1	1	public_transportation	2
yes	3	0	2	public_transportation	2
no	2	1	3	public_transportation	2
no	2	0	3	walking	3
no	0	0	2	public_transportation	4

Data Information

```
1 # display the data information
2 obesity_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   gender                          2111 non-null   object 
 1   age                             2111 non-null   int64  
 2   height                          2111 non-null   float64
 3   weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object 
 5   caloric_food                    2111 non-null   object 
 6   vegetables                      2111 non-null   int64  
 7   number_meals                    2111 non-null   int64  
 8   food_between_meals              2111 non-null   int64  
 9   smoke                           2111 non-null   object 
 10  water                           2111 non-null   int64  
 11  calories                        2111 non-null   object 
 12  activity                        2111 non-null   int64  
 13  technology                      2111 non-null   int64  
 14  alcohol                         2111 non-null   int64  
 15  transportation                  2111 non-null   object 
 16  obesity_level                   2111 non-null   int64  
dtypes: float64(2), int64(9), object(6)
memory usage: 280.5+ KB
```

Data Description

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 consume 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

Refining Raw Potential: Unveiling the Art of Data Preprocessing for Enhanced Model Performance

Rename Column in DataFrame

```
# rename the target column
obesity_df.rename({'obesity_level': 'obesity_class'}, axis = 1, inplace = True)
```

New Column in DataFrame

```
# create new columns
obesity_df['bmi'] = obesity_df['weight'] / obesity_df['height']**2
obesity_df.head(5)
```

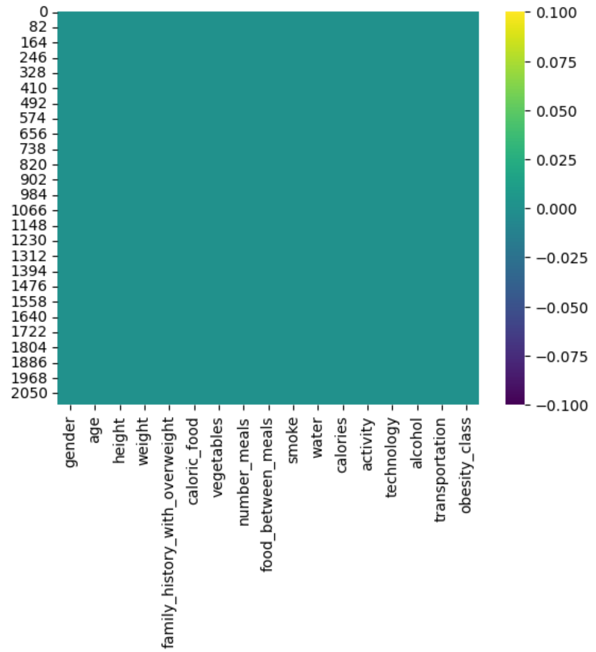
	gender	age	height	weight	family_history_with_overweight	caloric_food	vegetables	number_meals	food_between_meals	smoke	water	calories	activity	technology	alcohol	transportation	obesity_class	bmi
0	Female	21	1.62	64.0	yes	no	2	3	2	no	2	no	0	1	1	public_transportation	2	24.386526
1	Female	21	1.52	56.0	yes	no	3	3	2	yes	3	yes	3	0	2	public_transportation	2	24.238227
2	Male	23	1.80	77.0	yes	no	2	3	2	no	2	no	2	1	3	public_transportation	2	23.765432
3	Male	27	1.80	87.0	no	no	3	3	2	no	2	no	2	0	3	walking	3	26.851852
4	Male	22	1.78	89.8	no	no	2	1	2	no	2	no	0	0	2	public_transportation	4	28.342381

Missing Values

```python
# check for missing values
obesity_df.isnull().sum()
```

```python
# use heatmap to visualize the presence of missing values
sns.heatmap(obesity_df.isnull(), annot = False, cmap = 'viridis')
plt.show()
```

```
gender                            0
age                               0
height                            0
weight                            0
family_history_with_overweight    0
caloric_food                      0
vegetables                        0
number_meals                      0
food_between_meals                0
smoke                             0
water                             0
calories                          0
activity                          0
technology                        0
alcohol                           0
transportation                    0
obesity_class                     0
dtype: int64
```

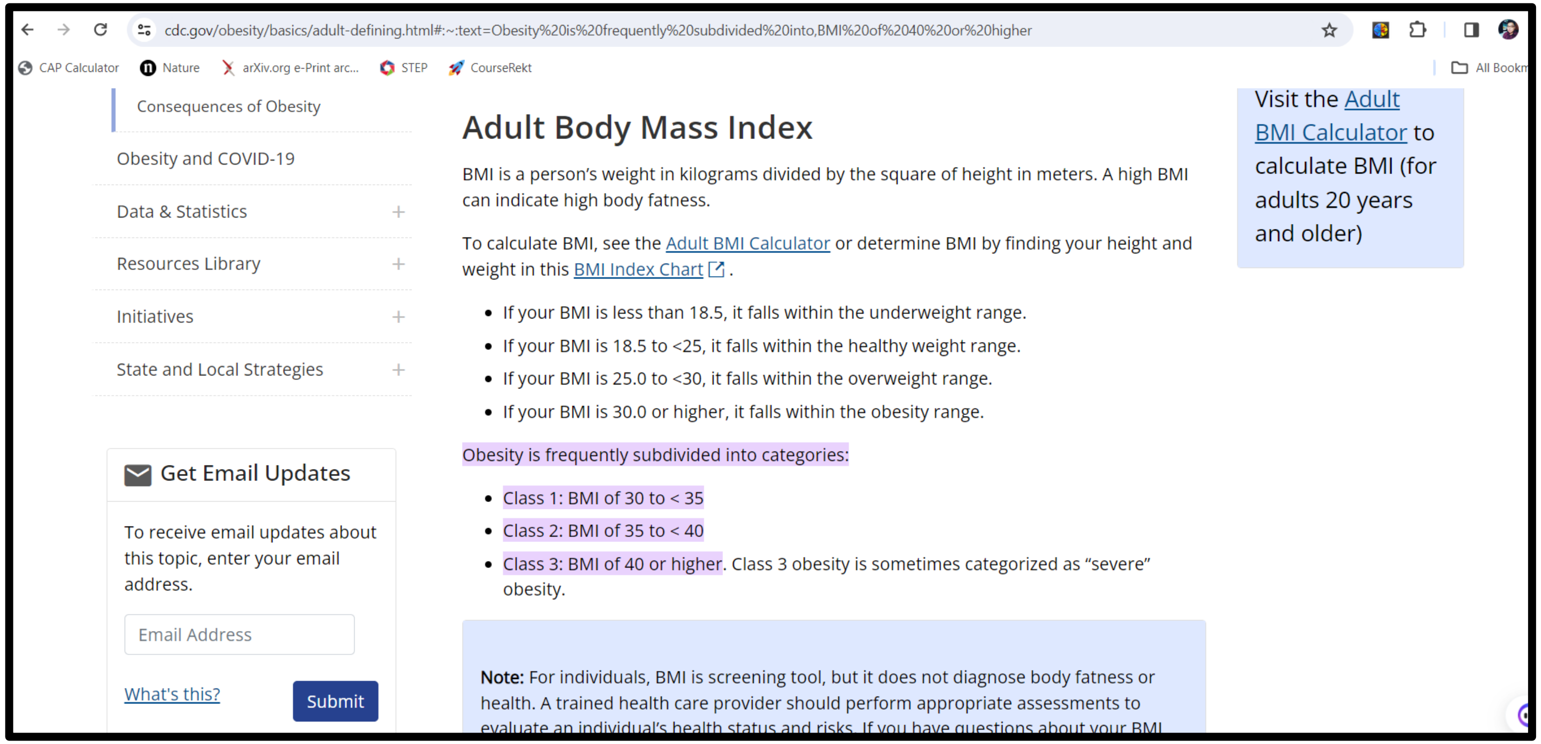


Integer-to-Categorical Label Mapping for Specific Categorical Features

```python
# convert the num values to cat values for the respective categorical columns
# categorical columns: vegetables, food_between_meals, water, activity, technology, alcohol, obesity_class

obesity_df['vegetables'] = obesity_df['vegetables'].map({1: 'not always', 2: 'frequently', 3: 'often'})
obesity_df['food_between_meals'] = obesity_df['food_between_meals'].map({1: 'not always', 2: 'sometimes', 3: 'frequently', 4: 'always'})
obesity_df['water'] = obesity_df['water'].map({1: 'not always', 2: 'sometimes', 3: 'frequently'})
obesity_df['activity'] = obesity_df['activity'].map({0: 'not always', 1: 'sometimes', 2: 'frequently', 3: 'always'})
obesity_df['technology'] = obesity_df['technology'].map({0: 'sometimes', 1: 'frequently', 2: 'often'})
obesity_df['alcohol'] = obesity_df['alcohol'].map({1: 'not always', 2: 'sometimes', 3: 'frequently', 4: 'always'})
obesity_df['obesity_class'] = obesity_df['obesity_class'].map({1: 'underweight', 2: 'normal_weight', 3: 'overweight', 4: 'overweight', 5: 'obesity_type_I', 6: 'obesity_type_II',
```

	gender	age	height	weight	family_history_with_overweight	caloric_food	vegetables	number_meals	food_between_meals	smoke	water	calories	activity	technology	alcohol	transportation	obesity_class	bmi
0	Female	21	1.62	64.0	yes	no	frequently	3	sometimes	no	sometimes	no	not always	frequently	not always	public_transportation	normal_weight	24.386526
1	Female	21	1.52	56.0	yes	no	often	3	sometimes	yes	frequently	yes	always	sometimes	sometimes	public_transportation	normal_weight	24.238227
2	Male	23	1.80	77.0	yes	no	frequently	3	sometimes	no	sometimes	no	frequently	frequently	frequently	public_transportation	normal_weight	23.765432
3	Male	27	1.80	87.0	no	no	often	3	sometimes	no	sometimes	no	frequently	sometimes	frequently	walking	overweight	26.851852
4	Male	22	1.78	89.8	no	no	frequently	1	sometimes	no	sometimes	no	not always	sometimes	sometimes	public_transportation	overweight	28.342381

CDC Body Mass Index and Category Requirements


cdc.gov/obesity/basics/adult-defining.html#:~:text=Obesity%20is%20frequently%20subdivided%20into,BMI%20of%2040%20or%20higher

CAP Calculator | Nature | arXiv.org e-Print arc... | STEP | CourseRekt | All Bookm

Consequences of Obesity

Obesity and COVID-19

Data & Statistics +

Resources Library +

Initiatives +

State and Local Strategies +

Get Email Updates

To receive email updates about this topic, enter your email address.

Email Address

What's this? Submit

Adult Body Mass Index

BMI is a person's weight in kilograms divided by the square of height in meters. A high BMI can indicate high body fatness.

To calculate BMI, see the Adult BMI Calculator or determine BMI by finding your height and weight in this BMI Index Chart.

- If your BMI is less than 18.5, it falls within the underweight range.
- If your BMI is 18.5 to <25, it falls within the healthy weight range.
- If your BMI is 25.0 to <30, it falls within the overweight range.
- If your BMI is 30.0 or higher, it falls within the obesity range.

Obesity is frequently subdivided into categories:

- Class 1: BMI of 30 to < 35
- Class 2: BMI of 35 to < 40
- Class 3: BMI of 40 or higher. Class 3 obesity is sometimes categorized as "severe" obesity.

Note: For individuals, BMI is screening tool, but it does not diagnose body fatness or health. A trained health care provider should perform appropriate assessments to evaluate an individual's health status and risks. If you have questions about your BMI

Visit the Adult BMI Calculator to calculate BMI (for adults 20 years and older)

obesity_class	bmi
underweight	18.565866
underweight	19.082206
underweight	18.545503
underweight	19.036573

An of BMI category that does not comply with the CDC BMI category

Filter Rows with Multiple Conditions

```
# create a function to verify the violation of the underweight range
def check_underweight_range(obesity_df):
  # create a boolean mask to filter rows based on multiple conditions that to verify the violation of the underweight range
  mask_obs_underweight = ((obesity_df['obesity_class'] == 'underweight') & (obesity_df['bmi'] > 18.5))
  res_obs_underweight = obesity_df[mask_obs_underweight]
  return res_obs_underweight
```

```
# create a function to verify the violation of the normal weight range
def check_normal_weight_range(obesity_df):
  # create a boolean mask to filter rows based on multiple conditions to verify the violation of the normal weight range
  mask_obs_normalweight = ((obesity_df['obesity_class'] == 'normal_weight') & (obesity_df['bmi'] >= 25.0)) | ((obesity_df['obesity_
  res_obs_normalweight = obesity_df[mask_obs_normalweight]
  return res_obs_normalweight
```

```
# create a function to verify the violation of the overweight range
def check_overweight_range(obesity_df):
  # create a boolean mask to filter rows based on multiple conditions that violated the overweight range
  mask_obs_overweight = ((obesity_df['obesity_class'] == 'normal_weight') & (obesity_df['bmi'] >= 30.0)) | ((obesity_df['obesity_cl
  res_obs_overweight = obesity_df[mask_obs_overweight]
  return res_obs_overweight
```

```
# create a function to verify the violation of the obesity type i range
def check_obesity_type_I_range(obesity_df):
  # create a boolean mask to filter rows based on multiple conditions that violated the obesity class I range
  mask_obs_type_i = ((obesity_df['obesity_class'] == 'obesity_type_I') & (obesity_df['bmi'] >= 35)) | ((obesity_df['obesity_class']
  res_obs_type_i = obesity_df[mask_obs_type_i]
  return res_obs_type_i
```

```
# create a function to verify the violation of the obesity type ii range
def check_obesity_type_II_range(obesity_df):
  # create a boolean mask to filter rows based on multiple conditions that vioplated the obesity class II range
  mask_obs_type_ii = ((obesity_df['obesity_class'] == 'obesity_type_II') & (obesity_df['bmi'] >= 40)) | ((obesity_df['obesity_class'
  res_obs_type_ii = obesity_df[mask_obs_type_ii]
  return res_obs_type_ii
```

Filter Rows with Multiple Conditions (con't)

```
# create a function to verify the violation of the obesity type ii range
def check_obesity_type_II_range(obesity_df):
    # create a boolean mask to filter rows based on multiple conditions that vioplated the obesity class II range
    mask_obs_type_ii = ((obesity_df['obesity_class'] == 'obesity_type_II') & (obesity_df['bmi'] >= 40)) | ((obesity_df['obesity_class
    res_obs_type_ii = obesity_df[mask_obs_type_ii]
    return res_obs_type_ii
```

```
# create a function to verify the violation of the obesity type iii range
def check_obesity_type_III_range(obesity_df):
    # create a boolean mask to filter rows based on multiple conditions that violated the obesity class III range
    mask_obs_type_iii = ((obesity_df['obesity_class'] == 'obesity_type_III') & (obesity_df['bmi'] < 40))
    res_obs_type_iii = obesity_df[mask_obs_type_iii]
    return res_obs_type_iii
```

Update Obesity Class based on BMI values

```
def corrected_obesity_class(bmi):
    if bmi < 18.5:
        corrected_class = 'underweight'
    elif bmi >= 18.5 and bmi < 25.0:
        corrected_class = 'normal_weight'
    elif bmi >= 25.0 and bmi < 30.0:
        corrected_class = 'overweight'
    elif bmi >= 30.0 and bmi < 35.0:
        corrected_class = 'obesity_type_I'
    elif bmi >= 35.0 and bmi < 40.0:
        corrected_class = 'obesity_type_II'
    elif bmi >= 40:
        corrected_class = 'obesity_type_III'
    return corrected_class
```

```
# update the obesity class column by applying the corrected obesity class based on BMI values
obesity_df['obesity_class'] = obesity_df['bmi'].apply(lambda x: corrected_obesity_class(x))
```

“

Strategic Segmentation: Optimizing Model Performance through Effective Data Splitting

Stratified K-Fold for Data Splitting

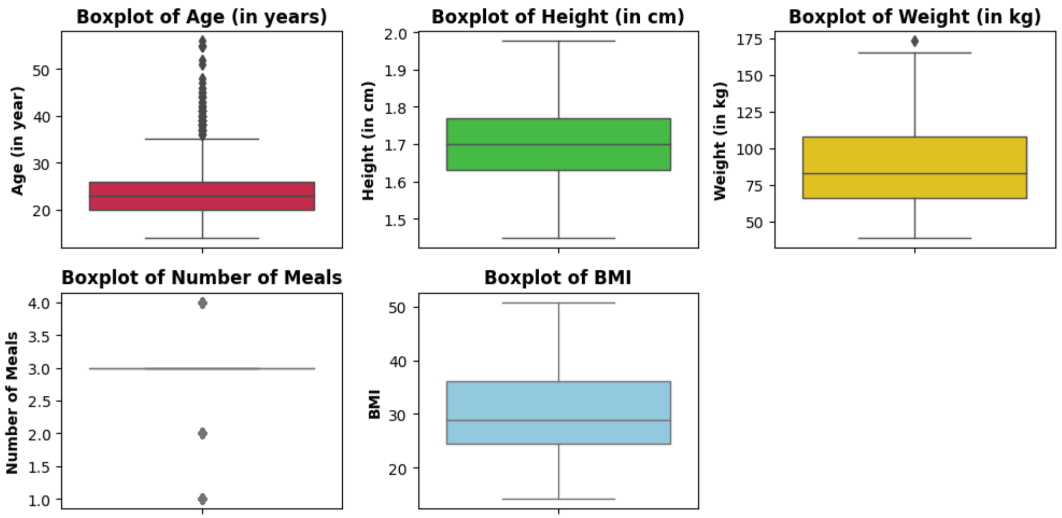
```
# splitting the dataset into features (X) and target variable (y)
X = obesity_df.drop('obesity_class', axis = 1)
y = obesity_df['obesity_class']

# creating an instance of stratifiedkfold
stratified_k_fold = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)

# iterating through the folds generated by stratifiedkfold
for train_index, test_index in stratified_k_fold.split(X, y):
  # splitting the data into training and testing sets based on the current fold indices
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]
```

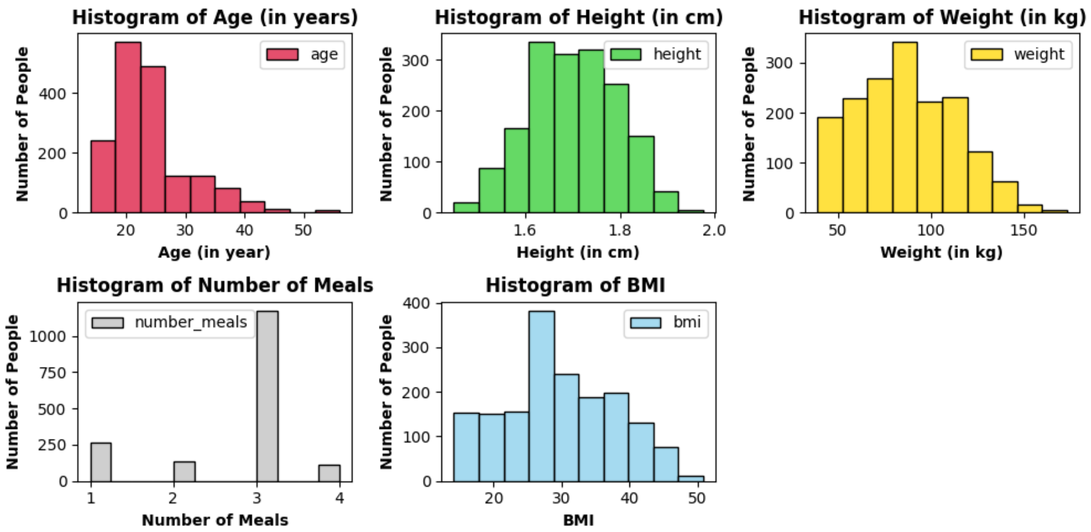
- Using Stratified K-Fold to split the data over a simple train-test-split because the number of **row data present in this dataset is limited** and **has imbalanced class distribution** (i.e. some classes have significantly fewer samples than others).
- Simple random splitting may lead to a training set that does not adequately represent the minority class, so Stratified K-Fold helps ensure that each fold has a representative distribution of classes.

Unveiling Insight: Navigating Numerical Features through Exploratory Feature Analysis

Detecting Outliers, Gaussian Distribution, and the Art of Normalizing and Scaling

Detect Presence of Outliers and Gaussian Distribution

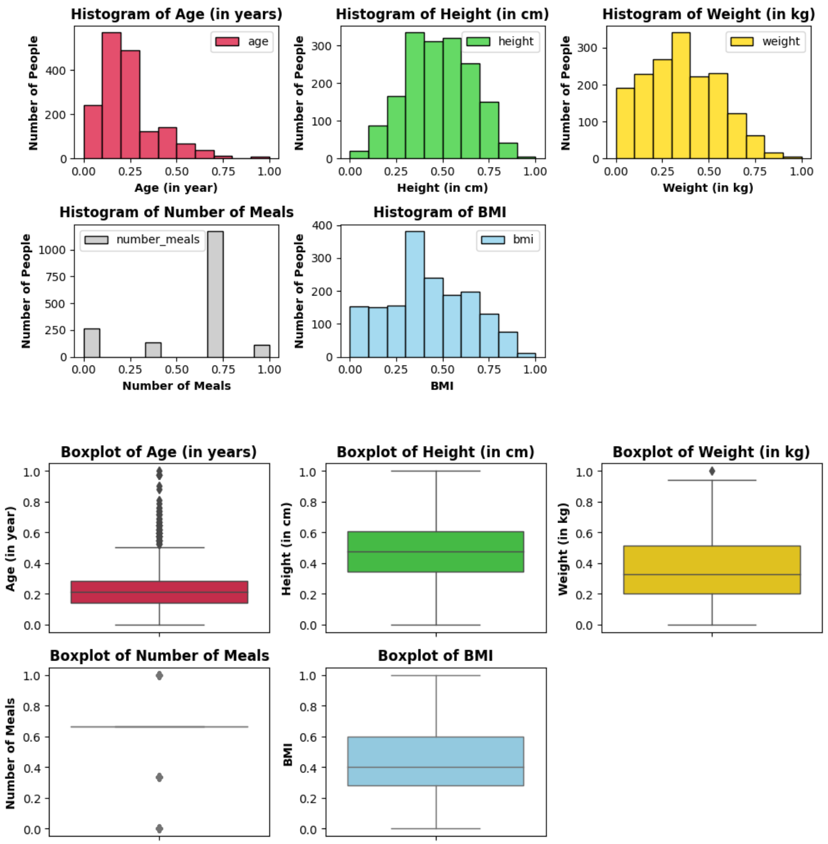




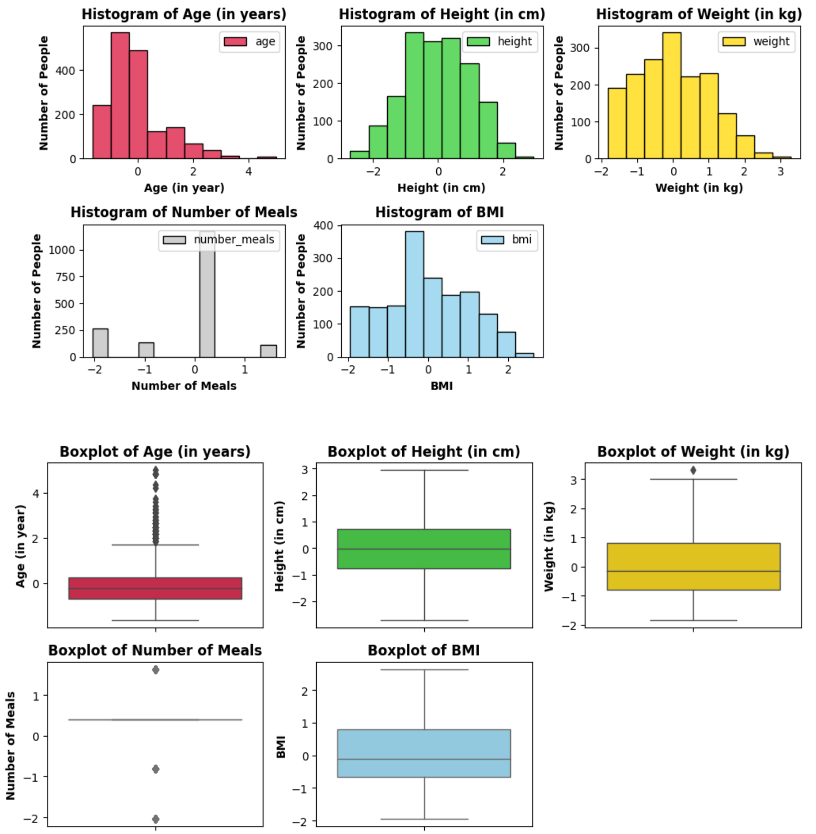
How do we remove outliers and achieve more-like Gaussian Distribution?

How do we remove outliers and achieve more-like Gaussian Distribution?

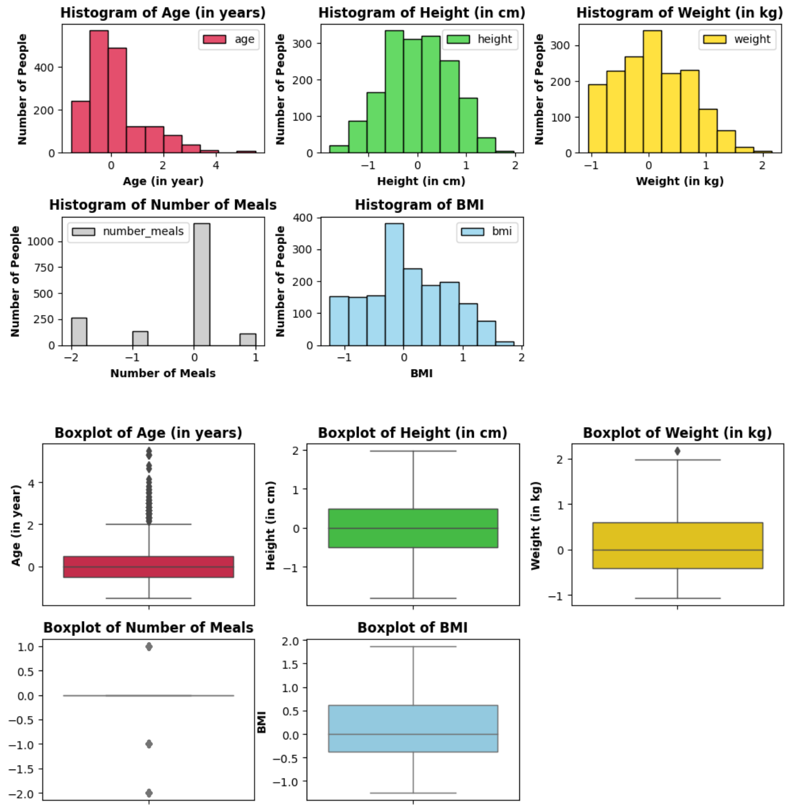
Min-Max Scaler



Standard Scaler



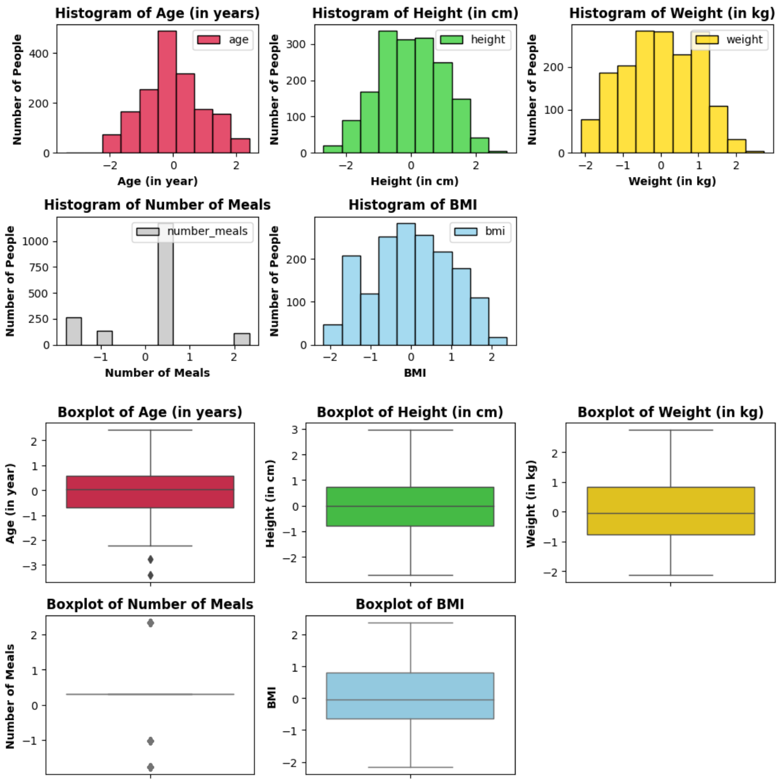
Robust Scaler



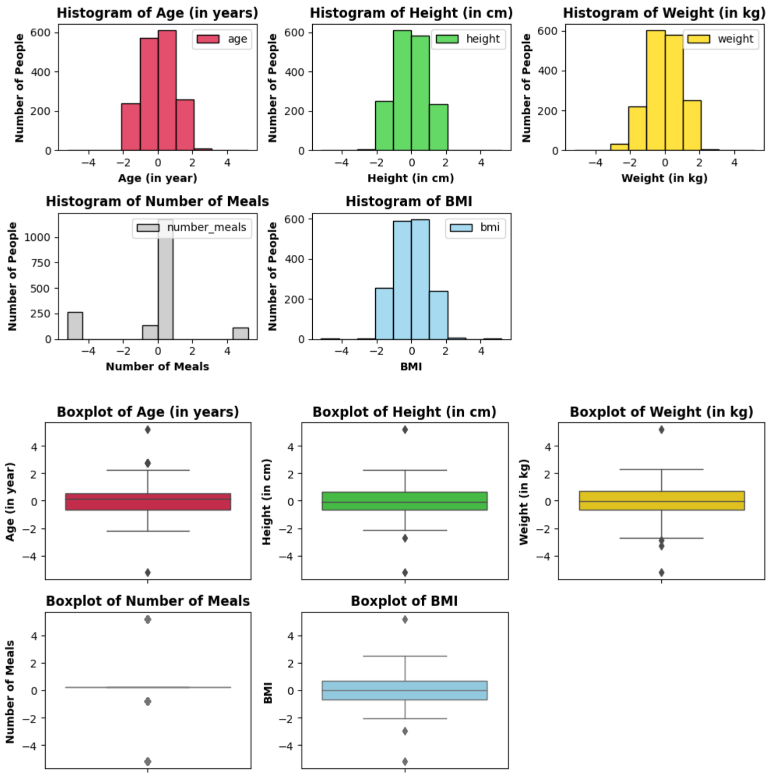
StandardScaler and **Min-Max Scaler** simply standardize or normalize the data, respectively, without addressing distribution shape, while **Robust Scaler** is designed to handle outliers but might not be as effective in transforming the data to a Gaussian distribution.

How do we remove outliers and achieve more-like Gaussian Distribution?

PowerTransformer



Quantile Transformer



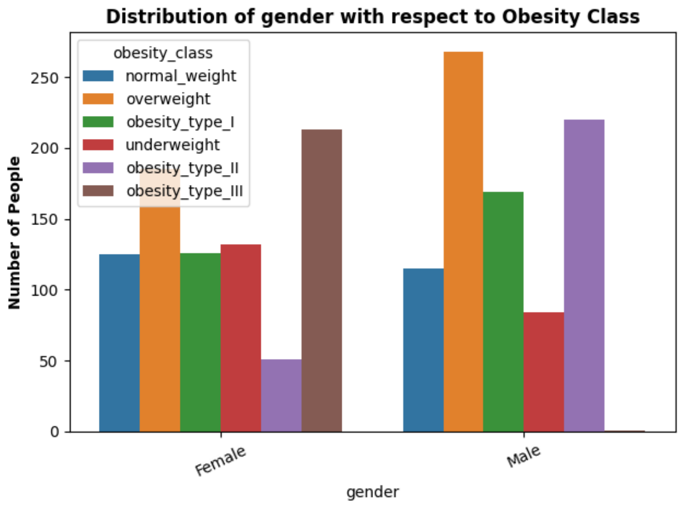
- **Power Transformer** uses the Yeo-Johnson transformation that can handle both positive and negative values. By estimating the optimal transformation parameters through maximum likelihood estimation, the Yeo-Johnson transformation applies a family of power transformation to the data. This flexibility allows it to adapt to different patterns of skewness and variance within the dataset. The transformation stabilizes variance by mitigating the impact of extreme values and asymmetry, making the distribution more symmetry.
- **QuantileTransformer** helps in achieving Gaussian-like properties, but it does not necessarily remove the outliers entirely. Outliers may still exist in the transformed data if they were present in the original distribution tails.

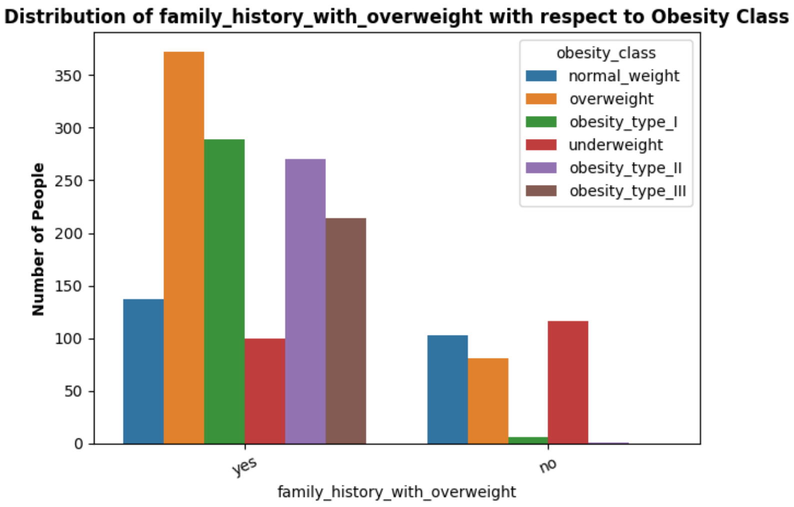
"

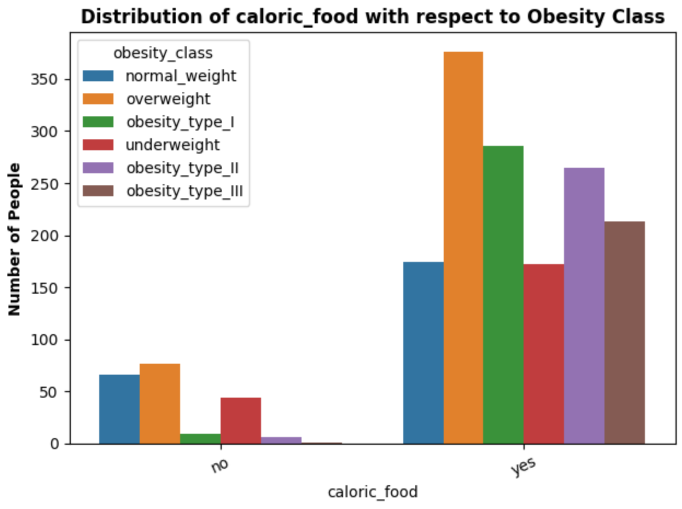
Unveiling Insight: Navigating Categorical Features through Exploratory Feature Analysis

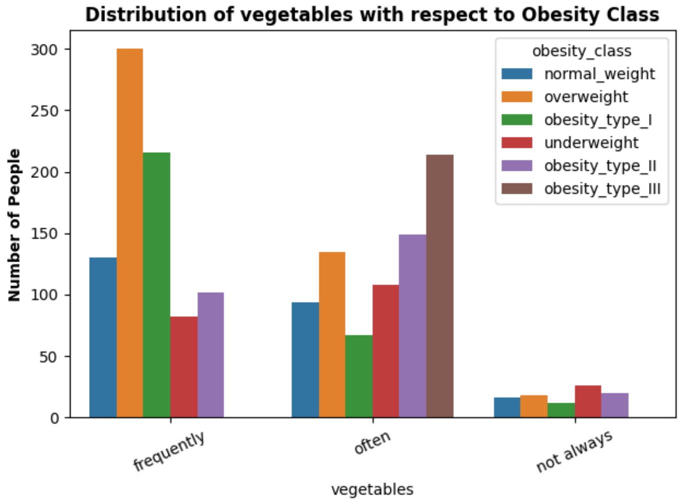
Empowering Transformation with Label Encoding

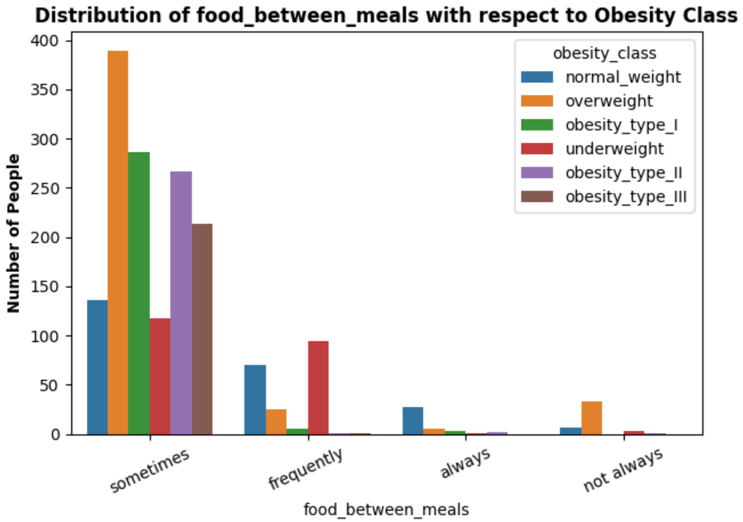
Visualizing Distribution of Various Categorical Features with respect to Obesity Class

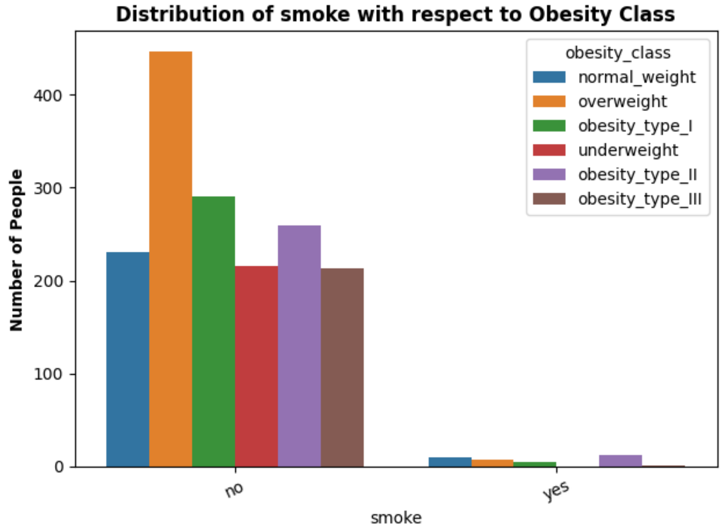




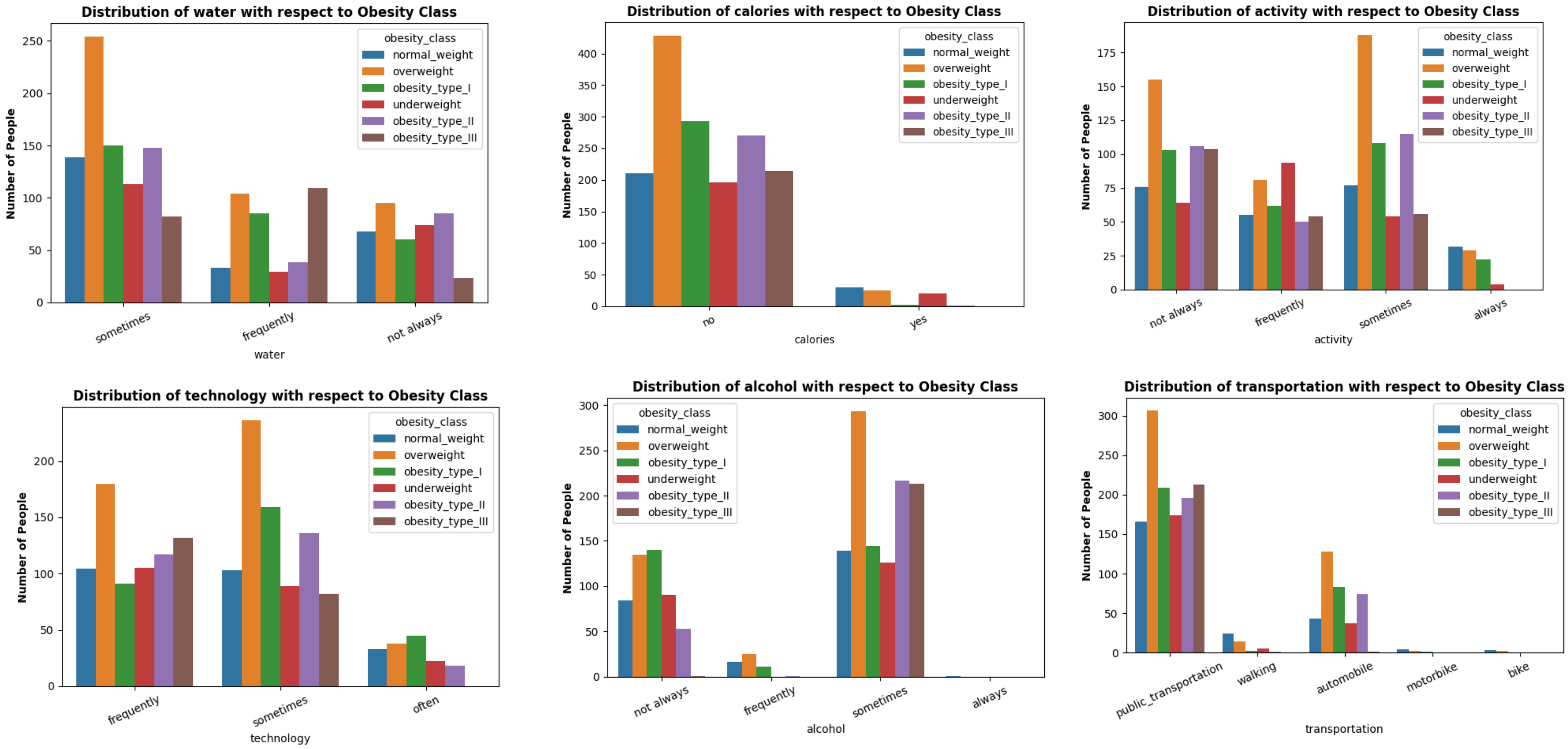








Visualizing Distribution of Various Categorical Features with respect to Obesity Class



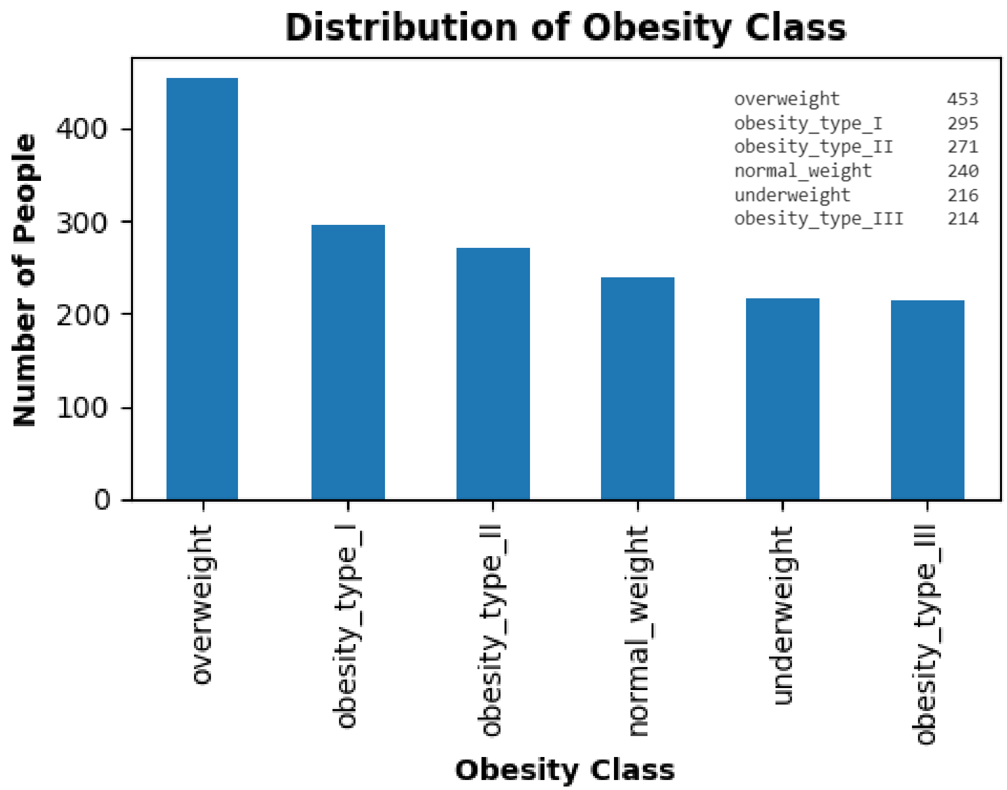
Label Encoder

```python
# initializing a LabelEncoder to encode categorical features
le = LabelEncoder()
# iterating through each categorical column in the training and testing sets
for name in cat_col_names:
    # encoding the categorical feature in both training and testing sets if it is not the target variable
    if name != 'obesity_class':
        X_train[name] = le.fit_transform(X_train[name])
        X_test[name] = le.transform(X_test[name])
    else: # encoding the categorical feature in both training and testing sets if it is target variable
        y_train = le.fit_transform(y_train)
        y_test = le.transform(y_test)

```

	gender	age	height	weight	family_history_with_overweight	caloric_food	vegetables	number_meals	food_between_meals	smoke	water	calories	activity	technology	alcohol	transportation	bmi
0	0	-0.418997	-0.880606	-0.847174	1	0	0	0.304430	3	0	2	0	2	0	2	3	-0.633026
2	1	0.044030	1.049359	-0.295850	1	0	0	0.304430	3	0	2	0	1	0	1	3	-0.717085
3	1	0.725906	1.049359	0.094331	0	0	2	0.304430	3	0	2	0	1	2	1	4	-0.307815
4	1	-0.174673	0.834137	0.199123	0	0	0	-1.773055	3	0	2	0	2	2	3	3	-0.117228
5	1	0.982251	-0.880606	-1.363716	0	1	0	0.304430	3	0	2	0	2	2	3	0	-1.219205

- This **LabelEncoder** is a simple and straightforward method as it **maps each category to an integer** and the transformation is reversible.

Visualizing Distribution of Obesity Class



What would happen if imbalanced classes were trained in the ML models?

- **Biased:** The model may tend to be biased towards the majority class since it is more prevalent in the training data. As a result, the model may struggle to accurately predict instances of the minority class.

- **Poor Generalization:** The model may not be generalised well to unseen data, especially for the minority class. It may perform well on majority class instances but poorly on minority class instances, leading to suboptimal overall performance.

- **Misleading Evaluation Metrics**: A model could achieve high accuracy by simply predicting the majority class, even if it fails to identify minority class instances.

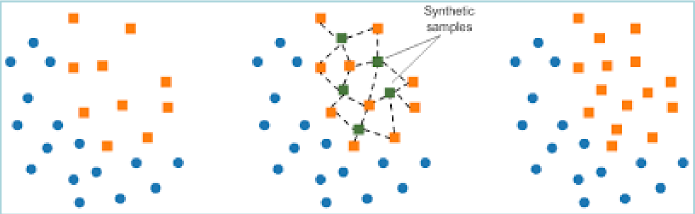
How do we achieve a balanced for each class?

Striking Balance: Exploring Hybrid Data Sampling Techniques through the Fusion of Over- and Under- sampling for Imbalanced Class

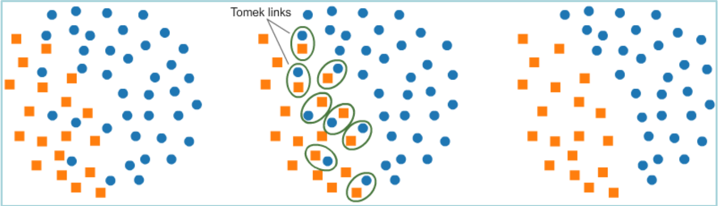
SMOTE-Tomek

```
# initializing smote-tomek for resampling to address imbalanced class labels
smt = SMOTETomek(random_state = 42)
# applying smote-tomek to the training data to balance the class distribution
X_train_smt, y_train_smt = smt.fit_resample(X_train, y_train)
```

- **SMOTE (Synthetic Minority Over-Sampling Technique)** and **Tomek links** are two techniques used to address the imbalanced class distribution.
- **SMOTE** helps to overcome the class imbalance by generating synthetic examples for the minority class, increasing it representation. The algorithm works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and draw a new sample at a point along that line.
- **Tomek** links aim to remove overlapping instances that might cause misclassification, in which identify and remove instances that form Tomek pairs. A Tomek pair consists of two instances of different classes that are nearest to each other.

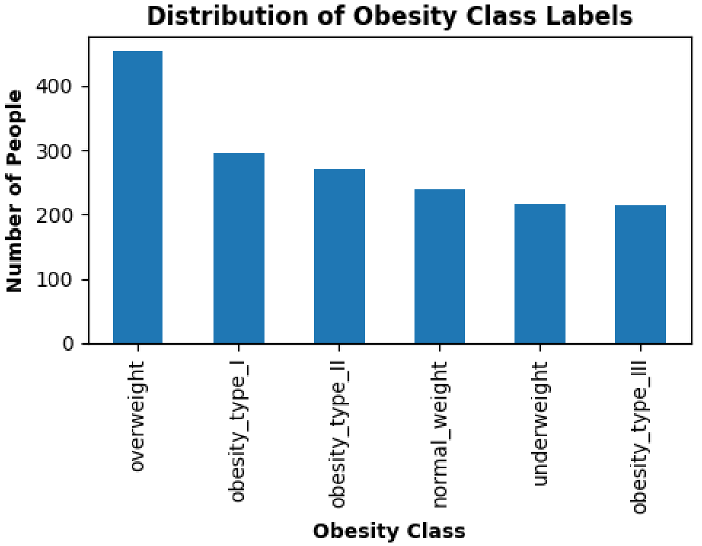


SMOTE

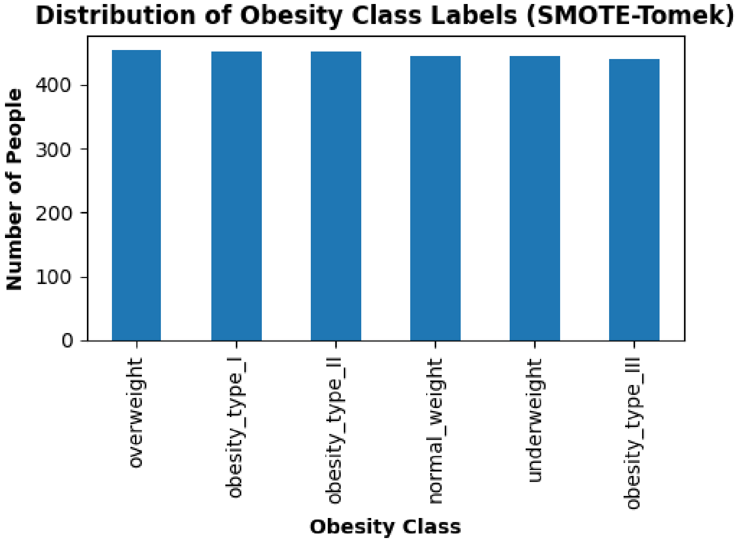


Tomek

Before applying SMOTE-Tomek



After applying SMOTE-Tomek



SMOTE-ENN

```
# initializing smote-enn for resampling to address imbalanced class labels
smtenn = SMOTEENN(random_state = 42)
# applying smote-enn to the training data to balance the class distribution
X_train_smtenn, y_train_smtenn = smtenn.fit_resample(X_train, y_train)
```

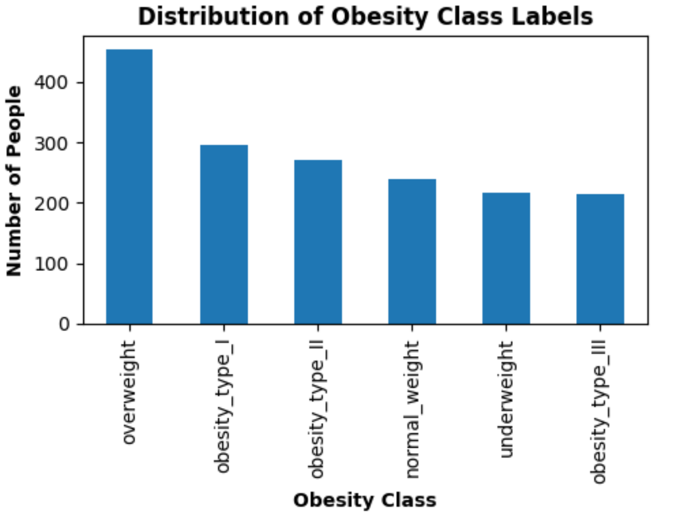
- **SMOTE** focuses on oversampling the minority class by generating synthetic instances.

- **ENN (Edited Nearest Neighbors)** aims to clean the dataset by removing potentially mislabel instances using a nearest neighbour's approach, by which identifies and removes instances that might be causing misclassification. If the majority class of the observation KNN and the observation class is different, then the observation and its K-nearest neighbor are deleted from the dataset.

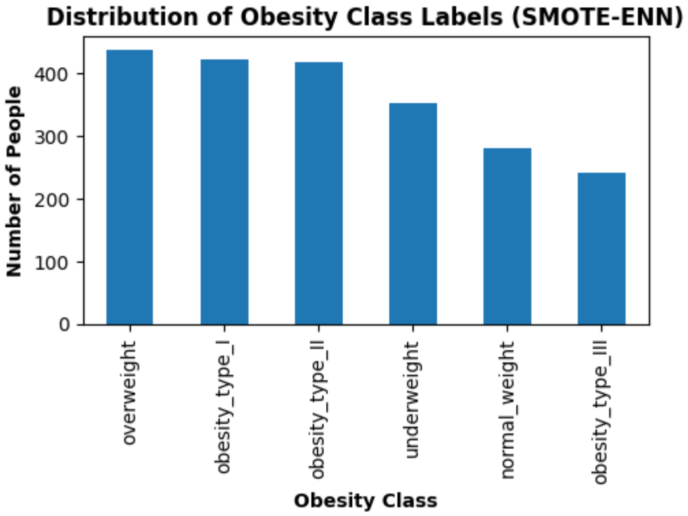
- **SMOTE-ENN** combines the strengths of both methods to enhance the model's performance on imbalanced datasets.

- **SMOTE-ENN** may not always result in a fully balanced distribution because ENN might not remove enough instances to achieve perfect balance, especially if there are complex noise or overlaps in the dataset.

Before applying SMOTE-ENN



After applying SMOTE-ENN



“

Ensemble Mastery: Navigating Model Selection and Evaluation Strategies for Robust Ensemble Machine Learning

Ensemble Machine Learning Algorithms

- Random Forest Classifier
- Gradient Boosting Classifier
- Extra Trees Classifier

Cross-Validation Score (Train + Validation)

```python
# perform cross-validation for the respective model using StratifiedKFold
def cross_validation_score(model, X_train_data, y_train_data):
  cv = StratifiedKFold(n_splits=5, shuffle = True, random_state = 42)
  # define a scorer for evaluation during the cross-validation
  myScorer = make_scorer(balanced_accuracy_score)
  # calculate the balanced accuracy scores for each fold and store them in the form of array list
  scores = cross_val_score(model, X_train_data, y_train_data, cv=cv, n_jobs = 1, scoring = myScorer)
  # calculate the average balanced accuracy scores across all folds
  avgScores = np.mean(scores)
  return avgScores
```

- **Cross-validation score** able to provide a more robust estimate of a model performance by averaging the performance metrics across multiple folds, reducing the impact of variability introduced by a single train-test split.
- This helps to obtain a more reliable assessment of a model's generalization ability and helps in selecting a model that performs well across different subsets of the data.
- **Stratified K-Fold** ensures that each fold maintains the same class distribution as the original training datasets, addressing imbalances in class representation.
- **"balanced_accuracy_score"** was used in this context as metrics because of robustness to class imbalance by considering the performance across all classes, providing a balanced view of the model effectiveness.

Cross-Validation Score (Train + Validation)

```
# create a baseline random forest classifier
rfc = RandomForestClassifier(random_state = 42)

# call the function to calculate cross validation scores for the random forest classifier usir
rfc_scores_smt = cross_validation_score(rfc, X_train_smt, y_train_smt)
rfc_scores_smtenn = cross_validation_score(rfc, X_train_smtenn, y_train_smtenn)

# display the cross-validation scores for the random forest classifier
print("Random Forest Classifier CV-Scores with SMOTE-Tomek: {:.4f}".format(rfc_scores_smt))
print("Random Forest Classifier CV-Scores with SMOTE-ENN: {:.4f}".format(rfc_scores_smtenn))
```

```
Random Forest Classifier CV-Scores with SMOTE-Tomek: 0.9963
Random Forest Classifier CV-Scores with SMOTE-ENN: 0.9982
```

```
# create a baseline gradient boosting classifier
gbc = GradientBoostingClassifier(random_state = 42)

# call the function to calculate cross validation scores for the gradient boosting classifier usi
gbc_scores_smt = cross_validation_score(gbc, X_train_smt, y_train_smt)
gbc_scores_smtenn = cross_validation_score(gbc, X_train_smtenn, y_train_smtenn)

# display the cross-validation scores for the gradient boosting classifier
print("Gradient Boosting Classifier CV-Scores with SMOTE-Tomek: {:.4f}".format(gbc_scores_smt))
print("Gradient Boosting Classifier CV-Scores with SMOTE-ENN: {:.4f}".format(gbc_scores_smtenn))
```

```
Gradient Boosting Classifier CV-Scores with SMOTE-Tomek: 0.9992
Gradient Boosting Classifier CV-Scores with SMOTE-ENN: 0.9996
```

```
# create a baseline extra trees classifier
etc = ExtraTreesClassifier(random_state = 42)

# call the function to calculate cross validation scores for the extra trees classifier usi
etc_scores_smt = cross_validation_score(etc, X_train_smt, y_train_smt)
etc_scores_smtenn = cross_validation_score(etc, X_train_smtenn, y_train_smtenn)

# display the cross-validation scores for the extra trees classifier
print("Extra Trees Classifier CV-Scores with SMOTE-Tomek: {:.4f}".format(etc_scores_smt))
print("Extra Trees Classifier CV-Scores with SMOTE-ENN: {:.4f}".format(etc_scores_smtenn))
```

```
Extra Trees Classifier CV-Scores with SMOTE-Tomek: 0.9820
Extra Trees Classifier CV-Scores with SMOTE-ENN: 0.9970
```

Why do we choose default parameters for CV?

Default parameters often represent a baseline configuration that may provide a reasonable performance and can serve as a benchmark against which we can compare the performance of models with customized parameters.

Cross Validation Predict (Train + Test)

```python
# perform cross-validation for the respective model using StratifiedKFold
def cross_validation_predict(model, X_test_data, y_test_data):
  cv = StratifiedKFold(n_splits=5, shuffle = True, random_state = 42)
  # generate cross-validation prediction for the test data using the specified model
  y_pred = cross_val_predict(model, X_test_data, y_test_data, cv=cv, n_jobs = 1)
  return y_pred
```

- **Cross-validation predict** allows for predictions on each fold, providing insights of how well the model generalizes to unseen data and helping to identify potential issues like overfitting and underfitting.

Cross Validation Predict (Train + Test)

Classification Report for Random Forest Classifier (SMOTE-Tomek)

	precision	recall	f1-score	support
normal_weight	0.99	1.00	0.99	445
underweight	0.99	1.00	0.99	445
obesity_type_II	1.00	1.00	1.00	451
overweight	1.00	1.00	1.00	453
obesity_type_III	1.00	0.99	0.99	441
obesity_type_I	1.00	1.00	1.00	451
accuracy			1.00	2686
macro avg	1.00	1.00	1.00	2686
weighted avg	1.00	1.00	1.00	2686

Classification Report for Random Forest Classifier (SMOTE-ENN)

	precision	recall	f1-score	support
normal_weight	1.00	1.00	1.00	282
underweight	1.00	1.00	1.00	353
obesity_type_II	1.00	1.00	1.00	418
overweight	1.00	1.00	1.00	437
obesity_type_III	1.00	0.99	1.00	241
obesity_type_I	1.00	1.00	1.00	422
accuracy			1.00	2153
macro avg	1.00	1.00	1.00	2153
weighted avg	1.00	1.00	1.00	2153

Classification Report for Gradient Boosting Classifier (SMOTE-Tomek)

	precision	recall	f1-score	support
normal_weight	1.00	1.00	1.00	445
underweight	1.00	1.00	1.00	445
obesity_type_II	1.00	1.00	1.00	451
overweight	1.00	1.00	1.00	453
obesity_type_III	1.00	1.00	1.00	441
obesity_type_I	1.00	1.00	1.00	451
accuracy			1.00	2686
macro avg	1.00	1.00	1.00	2686
weighted avg	1.00	1.00	1.00	2686

Classification Report for Gradient Boosting Classifier (SMOTE-ENN)

	precision	recall	f1-score	support
normal_weight	1.00	1.00	1.00	282
underweight	1.00	1.00	1.00	353
obesity_type_II	1.00	1.00	1.00	418
overweight	1.00	1.00	1.00	437
obesity_type_III	1.00	1.00	1.00	241
obesity_type_I	1.00	1.00	1.00	422
accuracy			1.00	2153
macro avg	1.00	1.00	1.00	2153
weighted avg	1.00	1.00	1.00	2153

Classification Report for Extra Trees Classifier (SMOTE-Tomek)

	precision	recall	f1-score	support
normal_weight	0.95	0.97	0.96	445
underweight	0.99	0.99	0.99	445
obesity_type_II	0.99	0.99	0.99	451
overweight	1.00	1.00	1.00	453
obesity_type_III	0.97	0.96	0.96	441
obesity_type_I	0.99	0.99	0.99	451
accuracy			0.98	2686
macro avg	0.98	0.98	0.98	2686
weighted avg	0.98	0.98	0.98	2686

Classification Report for Extra Trees Classifier (SMOTE-ENN)

	precision	recall	f1-score	support
normal_weight	0.99	1.00	0.99	282
underweight	1.00	0.99	1.00	353
obesity_type_II	1.00	1.00	1.00	418
overweight	1.00	1.00	1.00	437
obesity_type_III	1.00	1.00	1.00	241
obesity_type_I	1.00	1.00	1.00	422
accuracy			1.00	2153
macro avg	1.00	1.00	1.00	2153
weighted avg	1.00	1.00	1.00	2153

- **Random Forest Classifier** is less prone to overfitting as it builds multiple decision trees independently and combines their prediction through averaging or voting, which helps mitigate overfitting.
- On the other hand, **Gradient Boosting** builds tree sequentially, with each tree correcting the errors of its predecessors, making it more susceptible to overfitting.

Fine-Tuning the Forest: Exploring Optimal Performance with Hyperparameter Tuning for Random Forest Classifier

Grid-Search CV on Random Forest Classifier with SMOTE-Tomek Training Sets

```
# define a parameter grid for hyperparameter tuning with GridSearchCV
param_grid = {'n_estimators': np.arange(50, 150, 10),
              'max_depth': [2, 3, 4],
              'criterion': ['gini', 'entropy'],
              'min_samples_leaf': np.arange(2, 5),
              'min_samples_split': np.arange(1, 4)}
```

```
# create a stratified k-fold for cross-validation object
cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)
# initialize a GridSearchCV object for hyperparameter tuning of a Random Forest classifier
rfc_smt_gs = GridSearchCV(estimator = RandomForestClassifier(random_state = 42),
                          param_grid = param_grid,
                          cv = cv,
                          scoring = "accuracy",
                          verbose = 3,
                          return_train_score = True,
                          n_jobs = 1)
# fit the GridSearchCV object to the training data with SMOTE-Tomek resampling
rfc_smt_gs.fit(X_train_smt, y_train_smt)
```

GridSearchCV automates the process of search through a predefined hyperparameter grid, testing different combinations of hyperparameters, and evaluate the model performance using CV. It performs an exhaustive search over the specified hyperparameter space, considering all possible combinations, helps to identify the set of hyperparameters that produces the best model performance.

- **"n_estimators": number of trees in the RFC; more trees can increase model robustness (but may increase computation time)**
- **"max_depth": maximum depth of each decision tree; a deeper tree can capture more complex pattern (but may lead to overfitting)**
- **"criterion": to measure the quality of a split**
- **"min_samples_leaf": minimum number of samples required to be at leaf node which can be useful to prevent the overfitting**
- **"min_samples_split": minimum number of samples required to split an internal node which can be useful to prevent overfitting**

Similar process for SMOTE-ENN training set

Grid-Search CV on Random Forest Classifier with SMOTE-Tomek Training Sets

```
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=1, n_estimators=140;, score=(train=nan, test=nan) total time=   0.0s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=50;, score=(train=0.959, test=0.961) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=50;, score=(train=0.958, test=0.952) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=50;, score=(train=0.959, test=0.954) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=50;, score=(train=0.961, test=0.950) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=50;, score=(train=0.955, test=0.939) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=60;, score=(train=0.965, test=0.972) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=60;, score=(train=0.964, test=0.955) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=60;, score=(train=0.970, test=0.970) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=60;, score=(train=0.974, test=0.965) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=60;, score=(train=0.964, test=0.954) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=70;, score=(train=0.971, test=0.972) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=70;, score=(train=0.973, test=0.972) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=70;, score=(train=0.974, test=0.981) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=70;, score=(train=0.981, test=0.974) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=70;, score=(train=0.973, test=0.959) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=80;, score=(train=0.976, test=0.980) total time=   0.3s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=80;, score=(train=0.968, test=0.961) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=80;, score=(train=0.969, test=0.974) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=80;, score=(train=0.975, test=0.974) total time=   0.2s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=80;, score=(train=0.972, test=0.955) total time=   0.3s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=90;, score=(train=0.974, test=0.974) total time=   0.3s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=90;, score=(train=0.952, test=0.931) total time=   0.3s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=90;, score=(train=0.964, test=0.970) total time=   0.3s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=90;, score=(train=0.971, test=0.970) total time=   0.3s
END criterion=entropy, max_depth=3, min_samples_leaf=2, min_samples_split=2, n_estimators=90;, score=(train=0.961, test=0.946) total time=   0.3s
```

Similar process for SMOTE-ENN training set

Grid-Search CV on Random Forest Classifier with SMOTE-Tomek Training Sets

```
Best score for Random Forest Classifier (SMOTETomek): 0.9918
Best parameters for Random Forest Classifier (SMOTETomek):
{'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 120}
```

Grid-Search CV on Random Forest Classifier with SMOTE-ENN Training Sets

```
Best score for Random Forest Classifier (SMOTE-ENN + Feature Selected): 0.9907
Best parameters for Random Forest Classifier (SMOTE-ENN + Feature Selected):
{'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 70}
```

Evaluating Random Forest Classifier Performance with Optimized Hyperparameters on Test Data

Random Forest Classifier (trained with SMOTE-Tomek + Hyperparameter Optimization) on Test Set (Unseen)

Classification Report for Random Forest Classifier
(SMOTE-Tomek + Optimized Hyperparameters)

	precision	recall	f1-score	support
normal_weight	0.98	0.98	0.98	60
underweight	1.00	1.00	1.00	73
obesity_type_II	1.00	1.00	1.00	67
overweight	1.00	1.00	1.00	54
obesity_type_III	0.99	0.99	0.99	113
obesity_type_I	1.00	1.00	1.00	55
accuracy			1.00	422
macro avg	1.00	1.00	1.00	422
weighted avg	1.00	1.00	1.00	422

Random Forest Classifier (trained with SMOTE-ENN + Hyperparameter Optimization) on Test Set (Unseen)

Classification Report for Random Forest Classifier
(SMOTE-ENN + Optimized Hyperparameters)

	precision	recall	f1-score	support
normal_weight	0.92	0.93	0.93	60
underweight	1.00	0.99	0.99	73
obesity_type_II	0.99	1.00	0.99	67
overweight	1.00	1.00	1.00	54
obesity_type_III	0.99	0.96	0.97	113
obesity_type_I	0.95	1.00	0.97	55
accuracy			0.98	422
macro avg	0.97	0.98	0.98	422
weighted avg	0.98	0.98	0.98	422

Preserving Predictive Power: Saving the Trained Random Forest Classifier for Future Use

Save the model

```
import pickle
# save the trained random forest classifier with optimized hyperparameters
pickle.dump(rfc_smt_gs, open('/content/drive/MyDrive/rp_capstone_project/rfc_smt_fs_gs.pkl', 'wb'))
```

“

Seamless Integration: Deploying a Predictive Random Forest Model with Streamlit for Obesity Detection

Press F11 to exit full screen

Obesity Prediction

Medical Disclaimer: This platform is not serve as an alternative to medical advice from medical professional healthcare provider. If you have any specific questions about any medical matter, you should consult your doctor or other medical professional healthcare provider.

General guideline(s) to the user:

1) You are ***required*** to fill up all the information in this form in less than 5 mins.

Enter your name:

Tan Jia Hao

Enter your identification no.:

PA435

Gender:

- (•) Male
- ( ) Female

Enter your age:

20 − +

Enter your height (in m):

1.76 − +

Enter your weight (in kg):

87.90 − +

Does your family have a history of obesity?

- (•) Yes
- ( ) No

Do you often consume high caloric foods?

- (•) Yes
- ( ) No

How often do you consume fruits and vegetables?

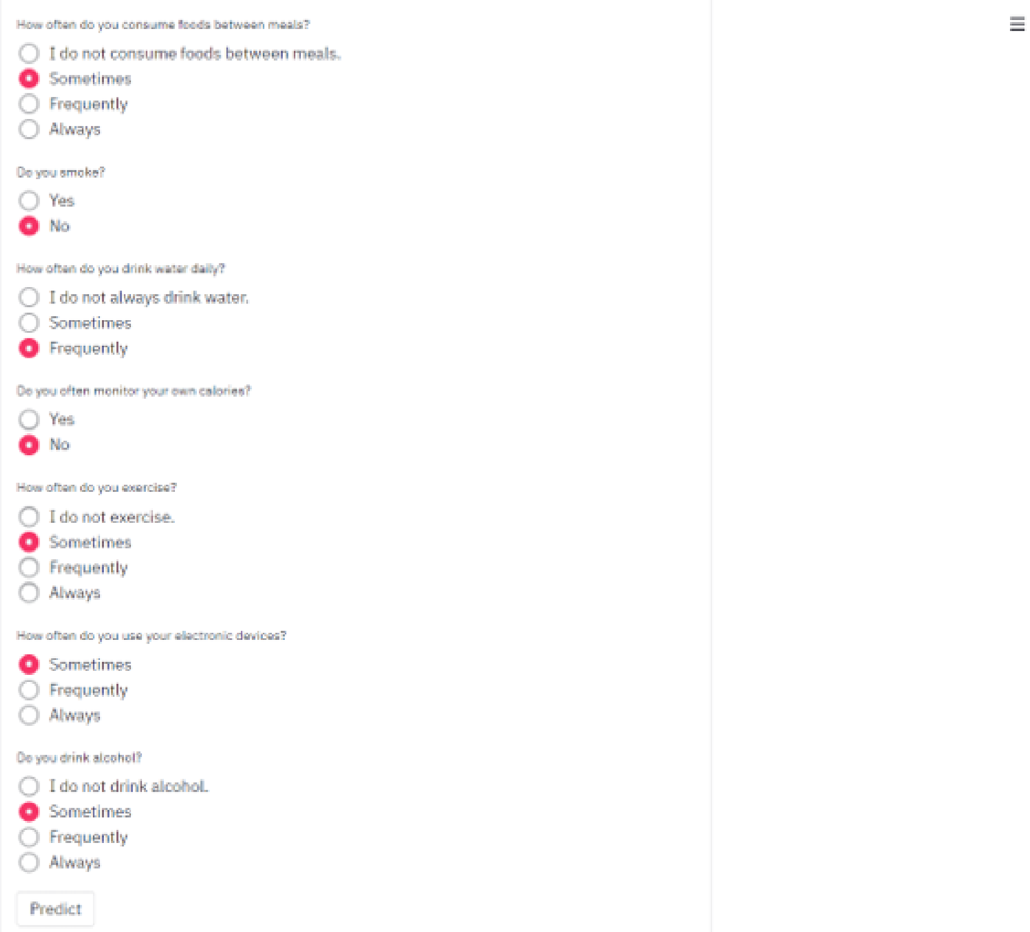
- ( ) I do not eat fruits and vegetables.
- (•) Frequently
- ( ) Often

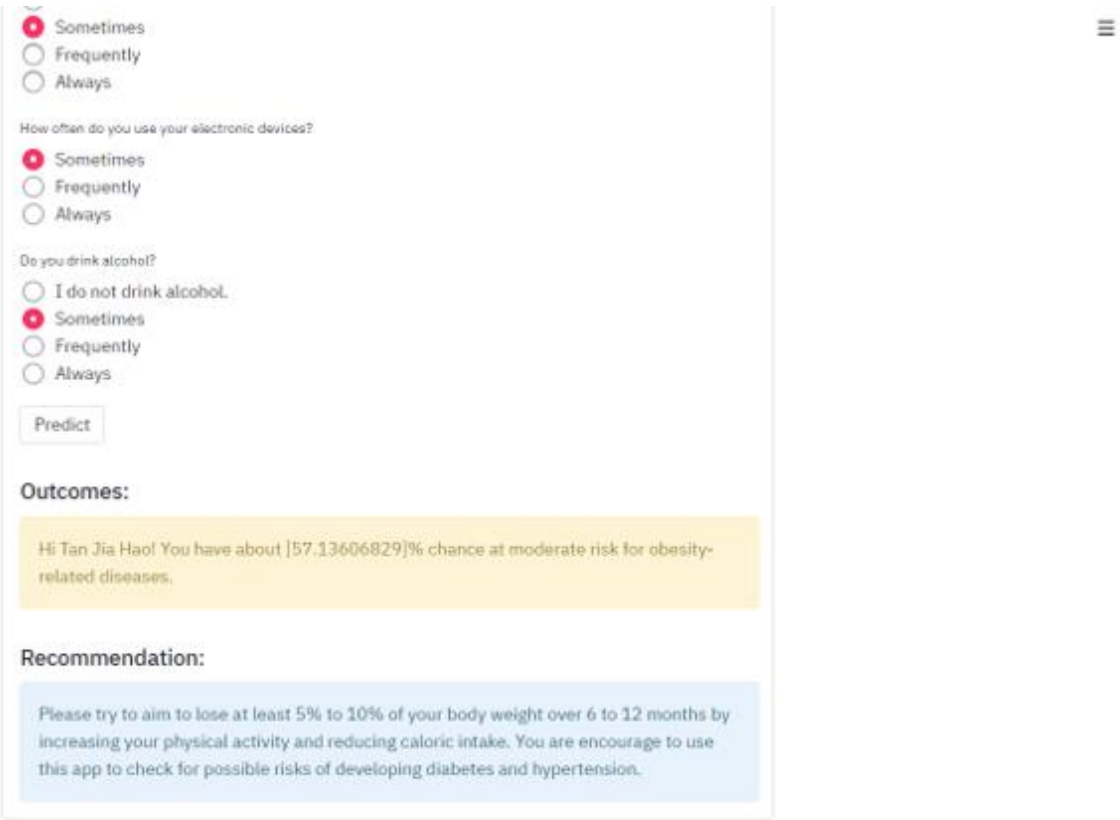
What is the number of main meals per day?

- ( ) 1
- ( ) 2
- (•) 3
- ( ) 4

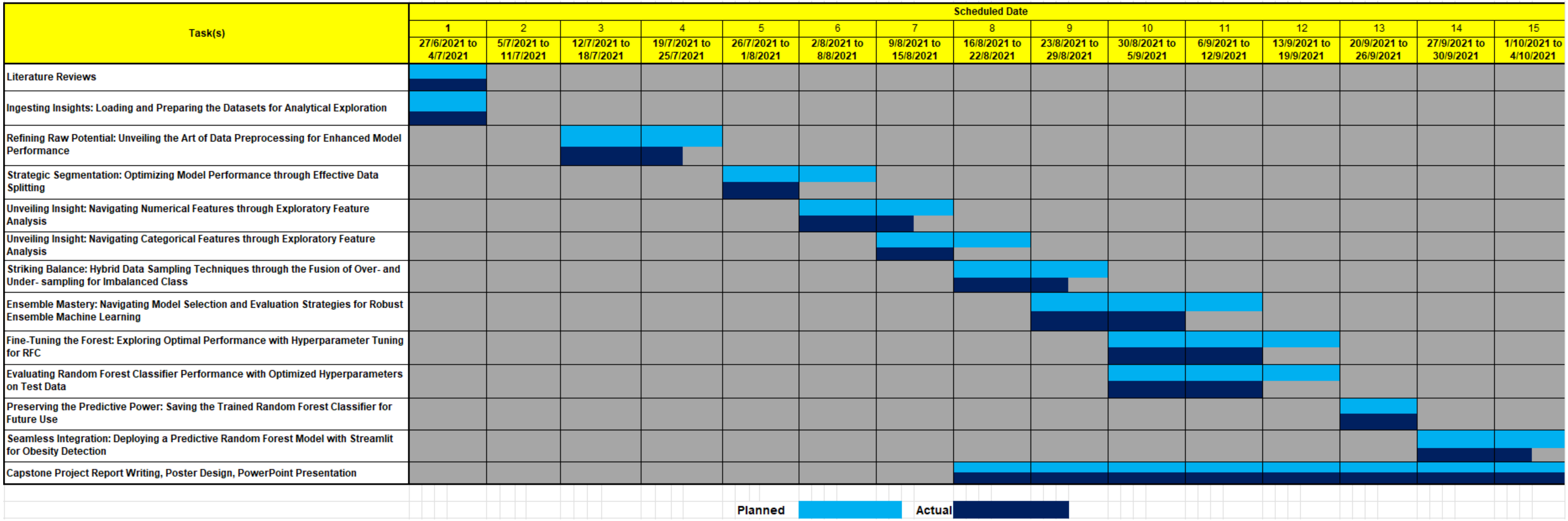
How often do you consume foods between meals?

- ( ) I do not consume foods between meals.
- (•) Sometimes
- ( ) Frequently


How often do you consume foods between meals?
I do not consume foods between meals.
Sometimes
Frequently
Always
Do you smoke?
Yes
No
How often do you drink water daily?
I do not always drink water.
Sometimes
Frequently
Do you often monitor your own calories?
Yes
No
How often do you exercise?
I do not exercise.
Sometimes
Frequently
Always
How often do you use your electronic devices?
Sometimes
Frequently
Always
Do you drink alcohol?
I do not drink alcohol.
Sometimes
Frequently
Always
Predict


Sometimes
Frequently
Always
How often do you use your electronic devices?
Sometimes
Frequently
Always
Do you drink alcohol?
I do not drink alcohol.
Sometimes
Frequently
Always
Predict
Outcomes:
Hi Tan Jia Hao! You have about [57.13606829]% chance at moderate risk for obesity-related diseases.
Recommendation:
Please try to aim to lose at least 5% to 10% of your body weight over 6 to 12 months by increasing your physical activity and reducing caloric intake. You are encourage to use this app to check for possible risks of developing diabetes and hypertension.

Project Timeline



References