

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



GRADUATION RESEARCH 1

**Classifying obesity levels with Machine Learning models
based on several features.**

Student:

Nguyen Tuan Hiep

hiep.nt205151@sis.hust.edu.vn

Under the assistance of:

Supervisor: Nguyen Hong Phuong

Semester 20221

Hanoi, 2/2023

TABLE OF CONTENTS

CHAPTER 1. Introduction	1
1.1 Problem Statement.....	1
1.2 Organization of Report.....	2
CHAPTER 2. Data exploration.....	3
2.1 Importing the data.....	3
2.2 Data visualization	4
CHAPTER 3. Data preprocessing.....	9
3.1 Overview	9
3.2 Data cleaning	9
3.3 Data scaling	10
3.4 Data encoding	11
3.5 Split the dataset into training set and testing set	12
CHAPTER 4. Apply Machine learning models on the dataset	15
4.1 Support Vector Machine Classification	15
4.2 K Nearest Neighbors Classification	18
4.3 Decision Tree Classification	19
4.4 Random Forest Classification.....	20

4.5 Neural Network Classification	20
CHAPTER 5. Testing the models on new dataset	23
5.1 Overview	23
5.2 Creating new dataset	23
CHAPTER 6. CONCLUSIONS	26

CHAPTER 1. Introduction

1.1 Problem Statement

Obesity is a complex and growing problem that affects a large percentage of the global population. According to the World Health Organization, obesity is defined as excessive fat accumulation that presents a risk to an individual's health. It is a major risk factor for a range of chronic diseases, including type 2 diabetes, cardiovascular disease, and certain types of cancer. The causes of obesity are multifaceted, including a combination of genetic, environmental, and lifestyle factors. As such, identifying and analyzing these factors is crucial to developing effective prevention and treatment strategies.

In recent years, machine learning has emerged as a powerful tool for predicting and classifying health conditions based on a range of features. In this report, I will explore the use of machine learning models to classify obesity levels based on several features. Specifically, I will examine attributes related to eating habits, such as frequent consumption of high caloric food, consumption of vegetables, number of main meals, consumption of food between meals, consumption of water daily, and consumption of alcohol. Additionally, I will consider attributes related to physical condition, including calories consumption monitoring, physical activity frequency, time using technology devices, and transportation used.

To carry out this analysis, I will use a publicly available dataset from Kaggle.com, which contains 2111 rows (observations) and 17 columns (features and labels). The labels for this classification task include insufficient weight, normal weight, overweight level I, overweight level II, obesity type I, obesity type II, and obesity type III.

In this project, I will use several supervised multi-class classification machine learning models, including K Nearest Neighbors, Decision Tree, Random Forest, Support Vector Machine and Neural Networks Classification to predict the obesity level of individuals based on their features. Through this analysis, I hope to gain insights into the factors that contribute to obesity and develop models that can help identify individuals at risk for the condition.

1.2 Organization of Report

The report will be organized into several sections:

The introduction will provide an overview of obesity, the features used in the analysis, the labels used to classify obesity levels, and the dataset used for analysis.

The second section will focus on data exploration and visualization using the *pandas*, *seaborn*, and *matplotlib* libraries.

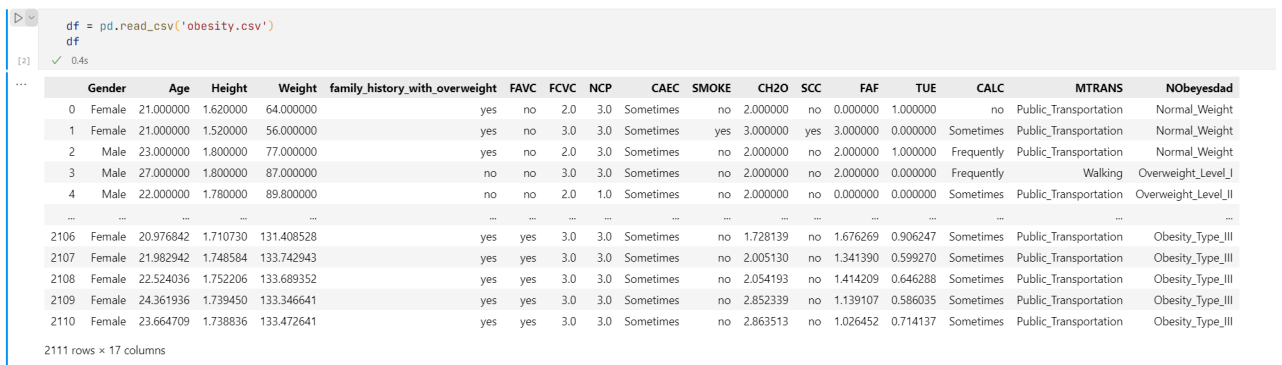
The third section will cover data preprocessing techniques such as data cleaning, handling missing values, and feature scaling to normalize the data.

In the fourth section, we will apply several supervised multiclass classification machine learning models to predict obesity levels based on the features. The models used will include K nearest neighbors, decision tree, random forest, support vector machine and neural networks. Model evaluation will be performed using *accuracy*, *precision*, *recall*, and *f1 – score* metrics, as well as *cross – validation* to test model performance. The results and discussion section will summarize model performance and highlight important features for predicting obesity levels. We will also compare the performance of different machine learning models.

CHAPTER 2. Data exploration

2.1 Importing the data

The dataset used in this analysis is stored in a *.csv* file format. To load and manipulate the data, I use the *pandas* library in *Python*. *Pandas* is a powerful library for data manipulation and analysis, providing functions for loading data from various file formats, such as *.csv* files, into a *DataFrame*, a two-dimensional table-like data structure. Using the *read_csv()* function in *pandas*, I import the data and store it in a *DataFrame*, which can be easily manipulated and analyzed using *pandas*' built-in functions. With the data in a *DataFrame*, I will explore and preprocess the data, as well as train and evaluate machine learning models to predict obesity levels.



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

[2] ✓ 0.4s

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObesyesdad
0	Female	21.000000	1.620000	64.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	0.000000	1.000000	no	Public_Transportation	Normal_Weight
1	Female	21.000000	1.520000	56.000000	yes	no	3.0	3.0	Sometimes	yes	3.000000	yes	3.000000	0.000000	Sometimes	Public_Transportation	Normal_Weight
2	Male	23.000000	1.800000	77.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	2.000000	1.000000	Frequently	Public_Transportation	Normal_Weight
3	Male	27.000000	1.800000	87.000000	no	no	3.0	3.0	Sometimes	no	2.000000	no	2.000000	0.000000	Frequently	Walking	Overweight_Level_I
4	Male	22.000000	1.780000	89.800000	no	no	2.0	1.0	Sometimes	no	2.000000	no	0.000000	0.000000	Sometimes	Public_Transportation	Overweight_Level_II
...
2106	Female	20.976842	1.710730	131.408528	yes	yes	3.0	3.0	Sometimes	no	1.728139	no	1.676269	0.906247	Sometimes	Public_Transportation	Obesity_Type_III
2107	Female	21.982942	1.748584	133.742943	yes	yes	3.0	3.0	Sometimes	no	2.005130	no	1.341390	0.599270	Sometimes	Public_Transportation	Obesity_Type_III
2108	Female	22.524036	1.752206	133.689352	yes	yes	3.0	3.0	Sometimes	no	2.054193	no	1.414209	0.646288	Sometimes	Public_Transportation	Obesity_Type_III
2109	Female	24.361936	1.739450	133.346641	yes	yes	3.0	3.0	Sometimes	no	2.852339	no	1.139107	0.586035	Sometimes	Public_Transportation	Obesity_Type_III
2110	Female	23.664709	1.738836	133.472641	yes	yes	3.0	3.0	Sometimes	no	2.863513	no	1.026452	0.714137	Sometimes	Public_Transportation	Obesity_Type_III

2111 rows x 17 columns

Figure 2.1: Pandas Dataframe

My machine learning models will be trained on several features and labels. Each feature will have their own abbreviation for short.

1. Attributes related with eating habits:

- Frequent consumption of high caloric food (*FAVC*)
- Frequency of consumption of vegetables (*FCVC*)
- Number of main meals (*NCP*)
- Consumption of food between meals (*CAEC*)

(e) Consumption of water daily (*CH20*)

(f) Consumption of alcohol (*CALC*)

2. Attributes related with the physical condition:

(a) Calories consumption monitoring (*SCC*)

(b) Physical activity frequency (*FAF*)

(c) Time using technology devices (*TUE*)

(d) Transportation used (*MTRANS*)

3. Label

(a) Weight level (*NObeyesdad*)

2.2 Data visualization

Data visualization is an essential part of exploratory data analysis, and it allows us to gain insights and detect patterns in the data. In this analysis, I use various visualization techniques to better understand the dataset. One important visualization is the correlation plot, which helps us understand the relationships between different features. For instance, we can use a scatter plot to visualize the correlation between Height and Weight. I also use histogram plots to visualize the distribution of age in the dataset. Additionally, I can use a bar plot to visualize the different types of weight categories. Another important visualization is the correlation matrix plot, which shows the correlation coefficients between all pairs of features. By visualizing the data in different ways, we can better understand the relationships between different features and identify potential correlations or patterns in the data.

- First of all, the count plot of different weight categories is plotted as below. It can be seen from the chart that the experiment are evenly carried out on all types of weights. Therefore, our machine learning models will not be affected by the factor of unbalanced distribution.

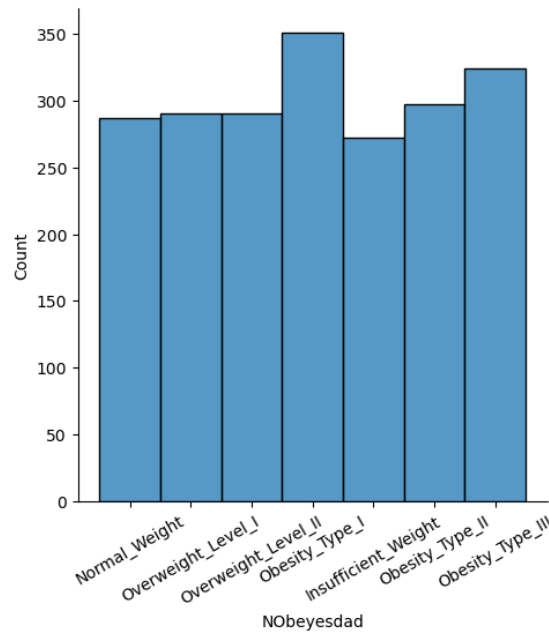


Figure 2.2: The number of people with their weight level

- The weight distribution, height distribution and weight distribution with respect to height are plotted below. The red line indicates that on average, the higher a person is, the more weight he has.

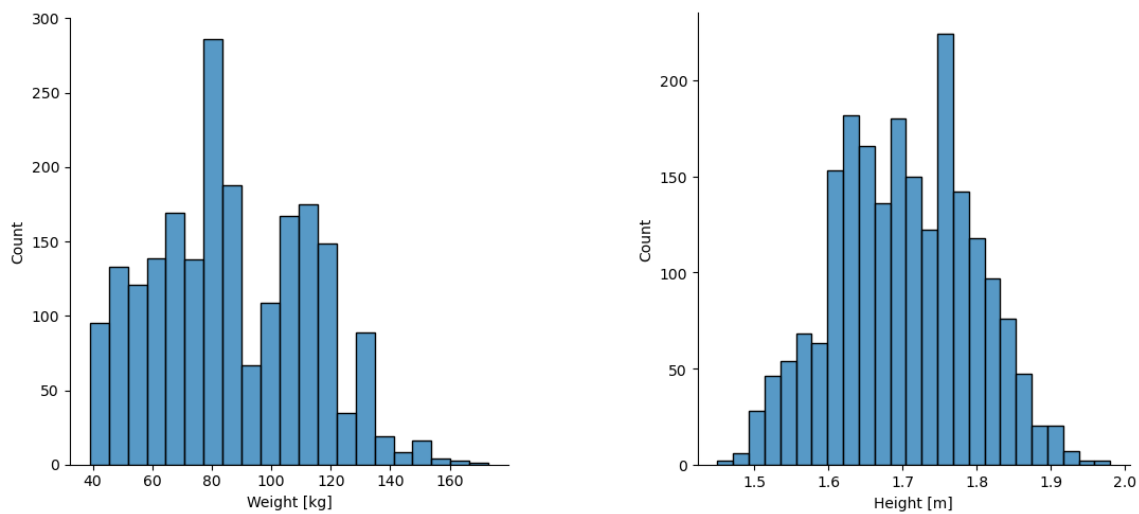


Figure 2.3: (a) Weight distribution, (b) Height distribution

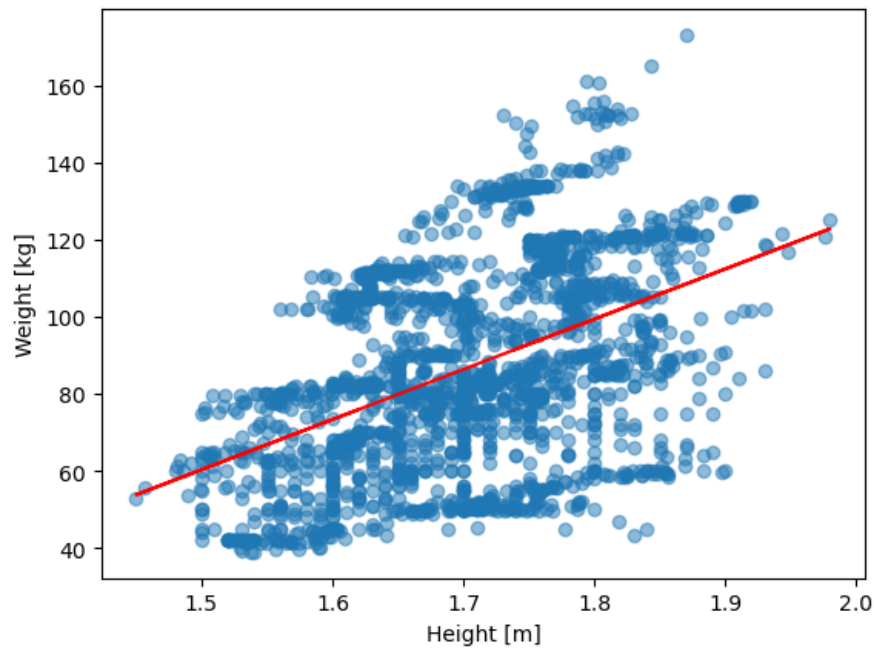


Figure 2.4: Correlation between Height and Weight

- The weight level between gender are illustrated as below. From the figure, it can be seen that the patients suffered from obesity type II are males while almost all patients who are diagnosed as obesity type III are females.

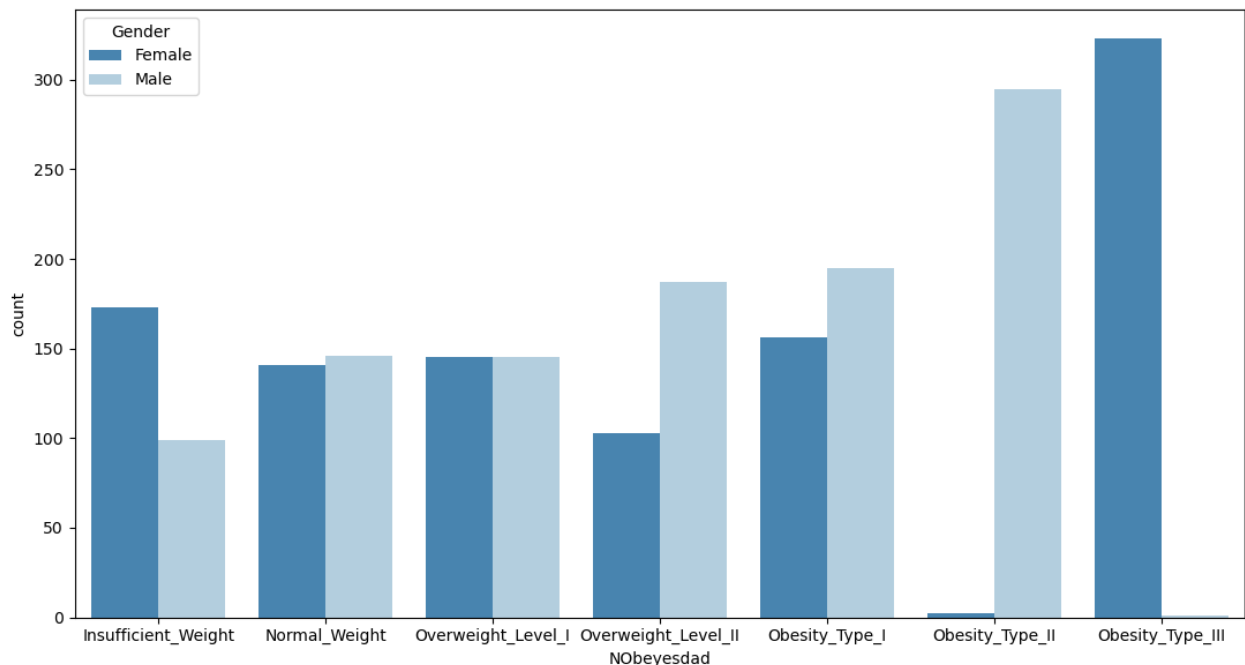


Figure 2.5: Weight distribution with respect to gender

- The weight levels among those having family history with overweight are also considered. It is apparent that people with family history with overweight have a higher

tendency to suffer from overweight and obesity.

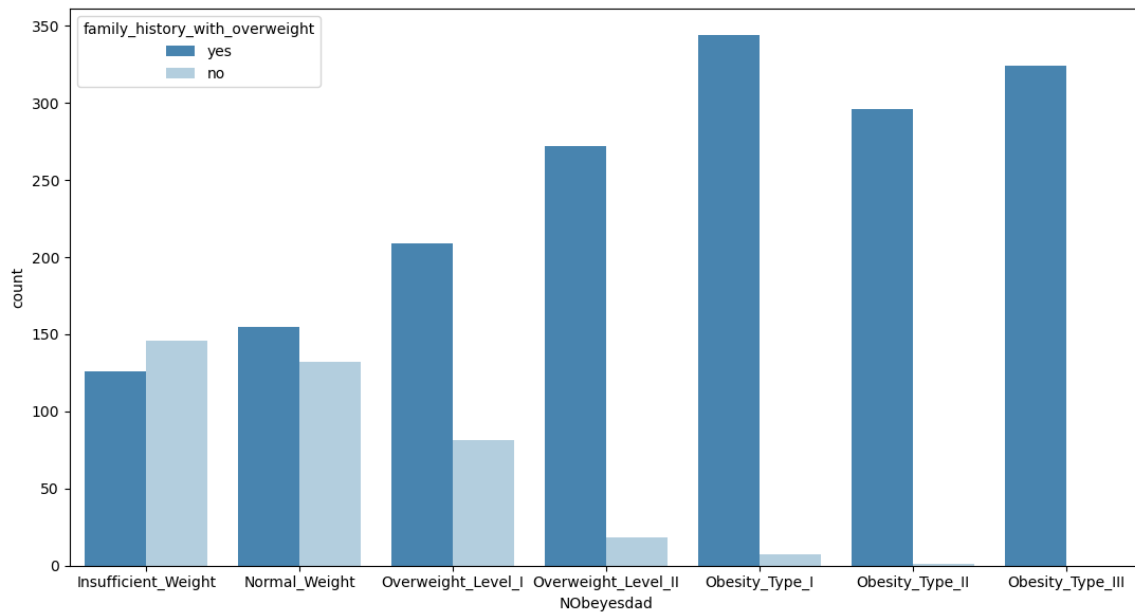


Figure 2.6: Weight distribution with respect to family history with overweight

Besides, I also plot the average weights for each weight category:

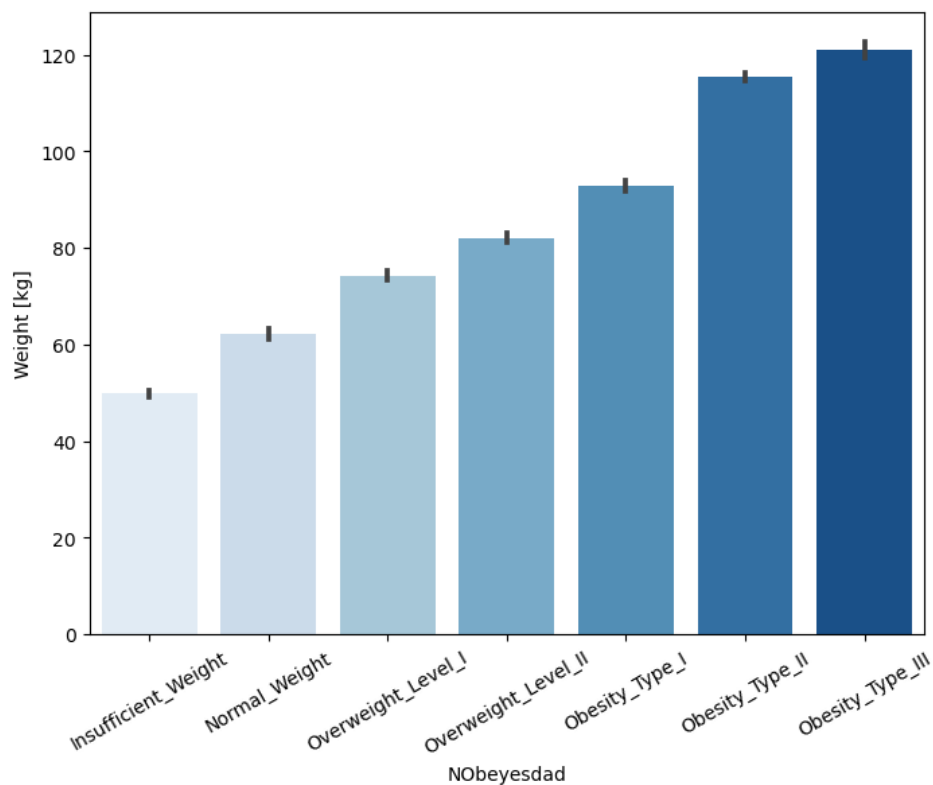


Figure 2.7: The average weight for each weight category

I also plot the correlation between the weight and the physical activity frequency

among candidates. It is likelihood that people with low physical activity frequency tend to have overweight and obesity.

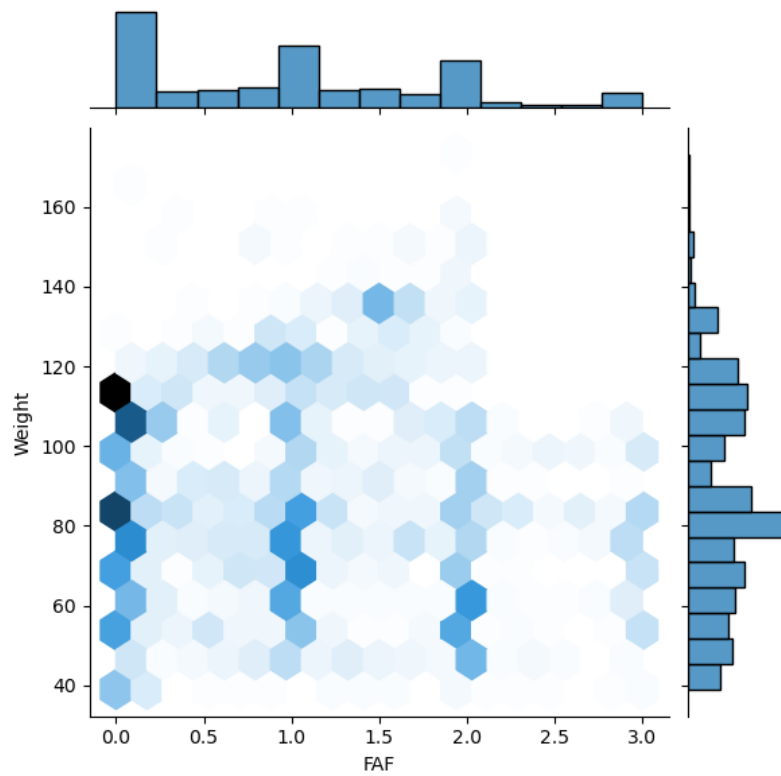


Figure 2.8: Weight with respect to physical activity frequency

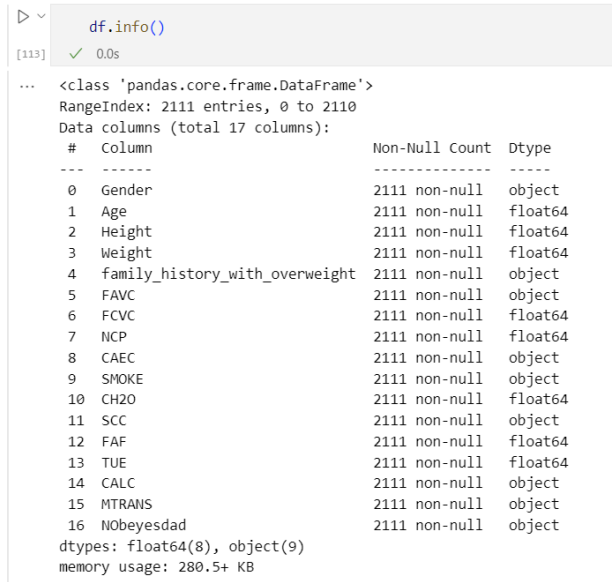
CHAPTER 3. Data preprocessing

3.1 Overview

Data preprocessing is an important step in data analysis and machine learning, and it involves cleaning, transforming, and preparing the data for analysis. In this analysis, I will preprocess the data to ensure that it is ready for machine learning modeling. First, I will clean the data by handling missing values and removing duplicates, which can negatively affect model performance. Next, I will transform the data by scaling the features, which helps improve model accuracy and stability. I can use techniques such as normalization or standardization to scale the data and finally I will encode the data from categorical values to numeric values.

3.2 Data cleaning

Data cleaning and handling missing values are essential steps in any machine learning project. The quality of the input data can significantly affect the accuracy and reliability of the machine learning models. Data cleaning involves identifying and correcting errors, inconsistencies, and outliers in the data, which may arise due to human errors or technical issues during data collection. Some common techniques for data cleaning include removing duplicate records, handling missing values, correcting data types and formats, and removing irrelevant or redundant features. Handling missing values is an important aspect of data cleaning, as missing data can cause bias and reduce the accuracy of the models. There are several techniques for handling missing values, such as removing the records with missing values, imputing the missing values using statistical methods, or using machine learning algorithms that can handle missing values. The choice of the method depends on the nature and amount of missing data, as well as the specific requirements of the problem. Data cleaning and handling missing values are iterative and ongoing processes that require close attention and monitoring throughout the project to ensure the quality and reliability of the results.



```

df.info()
[113] ✓ 0.0s

... <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   float64
2   Height                               2111 non-null   float64
3   Weight                               2111 non-null   float64
4   family_history_with_overweight       2111 non-null   object
5   FAVC                                 2111 non-null   object
6   FCVC                                 2111 non-null   float64
7   NCP                                  2111 non-null   float64
8   CAEC                                 2111 non-null   object
9   SMOKE                                2111 non-null   object
10  CH2O                                  2111 non-null   float64
11  SCC                                  2111 non-null   object
12  FAF                                  2111 non-null   float64
13  TUE                                  2111 non-null   float64
14  CALC                                  2111 non-null   object
15  MTRANS                               2111 non-null   object
16  Nobeyesdad                           2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB

```

Figure 3.1: Cheking the null values and missing values

It can be seen the dataset contains no record with NULL or NaN values, therefore I will come to the step of data scaling.

3.3 Data scaling

As we know, most of the machine learning models are trained from the dataset by the time the learning model maps the data points from input and output. However, the distribution of the data points can be different for every feature of the data. Larger differences between the data points of input variables may lead to uncertainty in the results of the model. Data scaling is a crucial step in data preprocessing and is used to normalize the range of values of features in the dataset. The process of scaling ensures that all features are on the same scale, which can improve model performance and stability. Machine learning models often perform better when the features have a similar scale, and scaling can also help to prevent features with larger magnitudes from dominating the model's learning process, which can make the model produce poor results or can perform poorly during learning.

Common scaling techniques include normalization and standardization. Normalization scales the values of each feature to a range between 0 and 1, while standardization scales the data so that it has a mean of 0 and a standard deviation of 1. The choice of scaling technique depends on the nature of the dataset and the machine learning algorithm being used.

In the project, *sklearn StandardScaler* is preferred as it is more powerful in classification problem and this scaling technique works well for datasets that have a normal distribution.

3.4 Data encoding

One hot encoding is a technique used in data preprocessing to transform categorical variables into numerical features that can be used in machine learning models. In some cases, machine learning models cannot handle categorical variables directly, and therefore we need to convert them into numerical features. One hot encoding creates a new binary feature for each category in the original variable, and the value of each binary feature is either 0 or 1, depending on whether the original observation belongs to that category or not. For instance, if we have a categorical variable "Color" with three categories - red, blue, and green, one hot encoding would create three new features: "IsRed", "IsBlue", and "IsGreen". The value of each feature would be either 0 or 1, depending on whether the original observation is red, blue, or green. One hot encoding is a powerful technique because it preserves the categorical information in the data while enabling machine learning models to use this information for accurate predictions. However, it is important to note that one hot encoding can increase the number of features in the dataset, which can negatively impact model performance and training time, especially for large datasets. Therefore, it is important to use one hot encoding judiciously and to experiment with different techniques to find the optimal approach for the given problem.

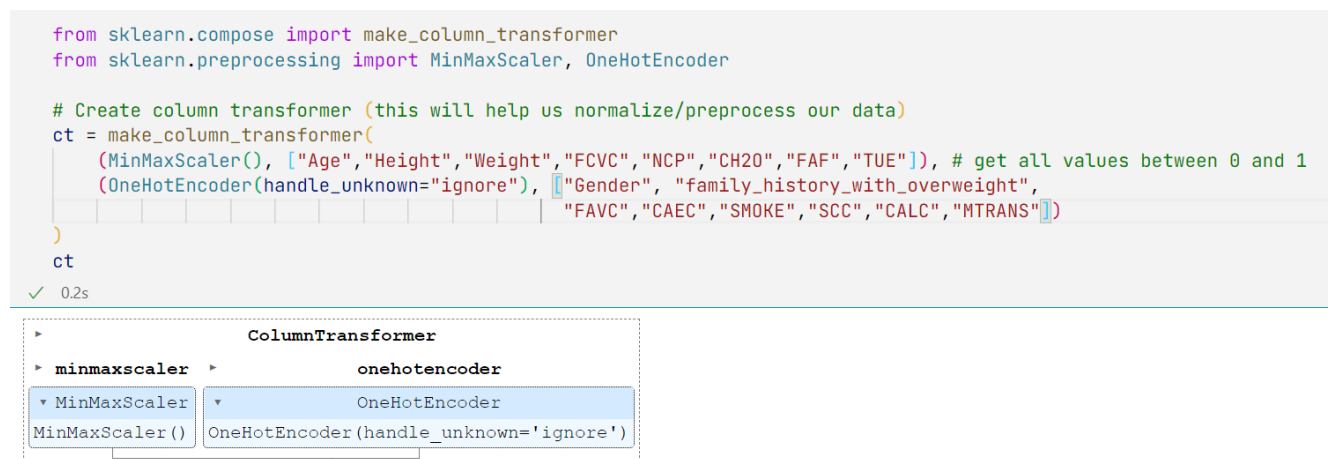


Figure 3.2: Scale and encode the data

In above code, the *ColumnTransformer* object is used to apply different transformations to the numeric and categorical features in the dataset. The numeric features

variables ("Age", "Height", "Weight", "FCVC", "NCP", "CH2O", "FAF", "TUE") and categorical features variables ("Gender", "family_history_with_overweight", "FAVC", "CAEC", "SMOKE", "SCC", "CALC", "MTRANS") are used to define which columns should be transformed. The numeric transformer and categorical transformer variables are used to define the transformations to be applied to each type of column. In this case, we are using *StandardScaler* to standardize the numeric features and *OneHotEncoder* to one hot encode the categorical features. Finally, we apply the transformations to the training and test data using the *fit_transform* and *transform* methods, respectively.

3.5 Split the dataset into training set and testing set

Train-test split is a common technique used in machine learning to evaluate the performance of a model. The goal of train-test split is to divide the available data into two separate sets: one for training the model and another for testing the model's performance. The training set is used to fit the model and learn the underlying patterns in the data, while the testing set is used to evaluate the model's performance on new, unseen data. The idea is to simulate the real-world scenario where the model is trained on historical data and then used to make predictions on new data.

In Python's *scikit-learn* library, the *train_test_split* function can be used to split the data into training and testing sets. The function randomly shuffles the data and splits it into two parts based on a specified percentage, usually 80/20 or 70/30. It is important to ensure that the data is split in a way that preserves the original distribution of the target variable to ensure that the model is tested on data that is representative of the real-world scenario. Train-test split is a simple yet effective technique to evaluate the performance of a model, but it has some limitations, such as the risk of *overfitting* the model to the training data or *underfitting* the model to the testing data. Therefore, it is important to use other techniques, such as *cross-validation*, to assess the robustness and generalization of the model.

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

✓ 0.3s

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.3s

```
((1477, 16), (634, 16), (1477,), (634,))
```

Figure 3.3: Split the dataset into training set and testing set

We fit the transformer to the training data (X_{train}) using the `fit_transform()` method, which applies the preprocessing steps defined by the `ColumnTransformer` to the training data and returns the preprocessed data in the form of a *numpy* array. The resulting preprocessed data (X_{train}) can be used to train a machine learning model.

```
# Fit the column transformer to the data
X_train = ct.fit_transform(X_train)
X_test = ct.transform(X_test)
```

✓ 0.2s

Figure 3.4: `fit_transform()` on training data and `transform()` on the test data

Note:

- Data standardization is the process of rescaling the attributes so that they have mean as 0 and variance as 1.
- The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.
- In `sklearn.preprocessing.StandardScaler()`, centering and scaling happens independently on each feature. The formula which performs standardization:

$$X_{scale} = \frac{X - \mu}{\sigma} \quad (3.1)$$

On the one hand, `fit_transform()` is used on the training data so that we can scale the training data and also learn the scaling parameters of that data. Here, the model built by us will learn the mean and variance of the features of the training set. These learned parameters are then used to scale our test data. The fit method is calculating the mean and variance of each of the features present in our data. The transform method is transforming all the features using the respective mean and variance. After that, we want scaling to be applied to our test data too and at the same time do not want to be biased with our model. We want our test data to be a completely new and a surprise set for our model. The transform method helps us in this case.

On the other hand, using the `transform()` method we can use the same mean and variance as it is calculated from our training data to transform our test data. Thus, the parameters learned by our model using the training data will help us to transform our test data. If we will use the fit method on our test data too, we will compute a new mean and variance that is a new scale for each feature and will let our model learn about our test

data too. Thus, what we want to keep as a surprise is no longer unknown to our model and we will not get a good estimate of how our model is performing on the test (unseen) data which is the ultimate goal of building a model using machine learning algorithm.

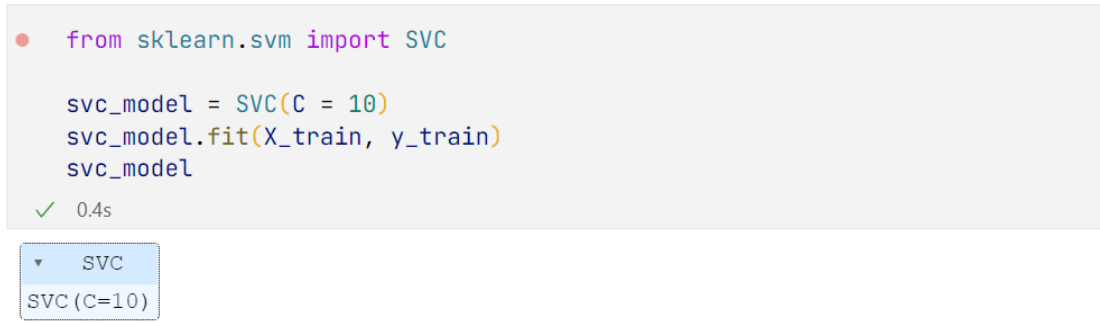
This is the standard procedure to scale our data while building a machine learning model so that our model is not biased towards a particular feature of the dataset and at the same time prevents our model to learn the features/values/trends of our test data.

CHAPTER 4. Apply Machine learning models on the dataset

4.1 Support Vector Machine Classification

Support Vector Machine (*SVM*) is a popular machine learning algorithm used for classification and regression analysis. *SVM* is a supervised learning algorithm that can be used for both linear and nonlinear classification problems. In *SVM*, the algorithm tries to find the optimal hyperplane that separates the different classes in the feature space with maximum margin, thereby minimizing the generalization error. The optimal hyperplane is found by maximizing the margin between the classes, which is the distance between the hyperplane and the closest data points of each class. *SVM* is a powerful algorithm that can handle high-dimensional feature spaces and is less prone to *overfitting* compared to other algorithms like decision trees. *SVM* can also handle nonlinear classification problems by mapping the original data to a higher-dimensional space using kernel functions. *SVM* is a powerful algorithm for multi-class classification and is widely used in various applications such as image recognition, text classification, and bioinformatics.

In this code snippet, we are importing the Support Vector Classifier (*SVC*) from the Scikit-learn library and instantiating it with a regularization parameter (*C*) set to 10. The fit method is then called on the instantiated *svc_model* object with the training data (*X_train* and *y_train*) as input parameters. This trains the model on the training data by learning the underlying patterns in the data and estimating the optimal hyperplane that separates the different classes in the feature space. The trained model can then be used to make predictions on new, unseen data using the predict method. The performance of the model can be evaluated using various metrics such as *accuracy*, *precision*, *recall*, and *f1 - score*, among others.



```

from sklearn.svm import SVC

svc_model = SVC(C = 10)
svc_model.fit(X_train, y_train)
svc_model

```

✓ 0.4s

▼ SVC

SVC (C=10)

Figure 4.1: Support Vector Machine model

In multi-class classification, True/False Positives and Negatives are defined based on the correct or incorrect classification of each class separately. For a given class, a True Positive (TP) is the number of samples that belong to that class and are correctly classified as such, while a False Negative (FN) is the number of samples that belong to that class but are incorrectly classified as another class. A False Positive (FP) is the number of samples that do not belong to that class but are incorrectly classified as belonging to that class, while a True Negative (TN) is the number of samples that do not belong to that class and are correctly classified as such.

Therefore, in multi-class classification, we can have TP , TN , FP , and FN for each class separately. The total number of TP , TN , FP , and FN across all classes can be used to calculate various evaluation metrics such as *accuracy*, *precision*, *recall*, *f1-score*, etc.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1-score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Figure 4.2: Different metrics to evaluate the model

Accuracy score: 0.8785488958990536

	precision	recall	f1-score	support
0	0.90	0.93	0.91	86
1	0.74	0.73	0.74	93
2	0.78	0.74	0.76	88
3	0.79	0.81	0.80	79
4	0.94	0.94	0.94	102
5	0.97	0.99	0.98	88
6	1.00	0.99	0.99	98
accuracy			0.88	634
macro avg	0.87	0.88	0.87	634
weighted avg	0.88	0.88	0.88	634

Figure 4.3: Performance of Support Vector Machine model

A confusion matrix is a table that is used to evaluate the performance of a classification model. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class in a multi-class classification problem. The rows of the matrix represent the true classes, while the columns represent the predicted classes. The diagonal elements of the matrix represent the correctly classified samples, while the off-diagonal elements represent the misclassified samples. The confusion matrix can be used to calculate various evaluation metrics such as *accuracy*, *precision*, *recall*, *f1 – score*, etc. It is a useful tool to understand how well a classification model is performing and to identify which classes are being misclassified more often than others.

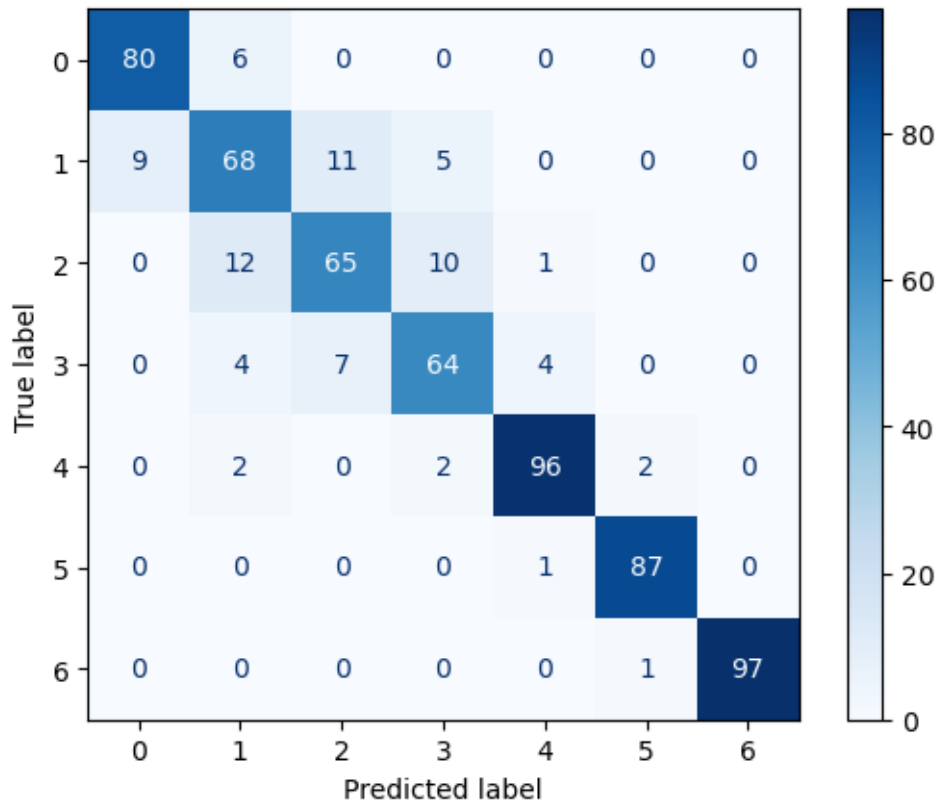


Figure 4.4: Confusion matrix of Support Vector Machine classification

4.2 K Nearest Neighbors Classification

K nearest neighbor (KNN) is a non-parametric classification algorithm that is used to classify a sample based on the majority class of its k -nearest neighbors in the feature space. The value of k is a hyperparameter that needs to be specified before training the model. To predict the class label of a new sample, KNN algorithm computes the distance between the new sample and all the training samples, and selects the k training samples with the shortest distance. The class label of the new sample is then assigned to the class that appears most frequently among its k nearest neighbors. KNN is a simple and intuitive algorithm that does not require any assumption about the distribution of the data. However, it can be computationally expensive to compute the distance between the new sample and all the training samples, especially in high-dimensional feature spaces.

Accuracy score: 0.7902208201892744

	precision	recall	f1-score	support
0	0.73	0.92	0.81	86
1	0.68	0.48	0.57	93
2	0.71	0.65	0.68	88
3	0.75	0.77	0.76	79
4	0.78	0.77	0.78	102
5	0.86	0.94	0.90	88
6	0.96	0.99	0.97	98
accuracy			0.79	634
macro avg	0.78	0.79	0.78	634
weighted avg	0.79	0.79	0.78	634

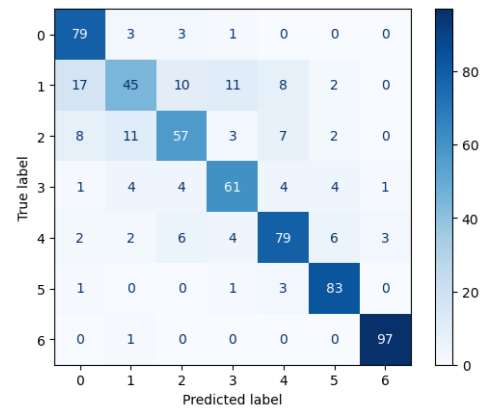


Figure 4.5: (a) KNN score, (b) KNN Confusion matrix

4.3 Decision Tree Classification

A decision tree is a supervised learning algorithm that is used for classification and regression tasks. It works by recursively partitioning the feature space into smaller regions that are pure in terms of the target variable. At each node of the tree, the algorithm selects the feature that maximizes the information gain or reduces the impurity of the target variable the most. The tree is built until all the training samples belong to the same class or when a stopping criterion is met. To predict the class label of a new sample, the decision tree algorithm traverses the tree from the root to a leaf node based on the feature values of the new sample. The class label of the new sample is then assigned to the majority class of the training samples that belong to the same leaf node. Decision trees are easy to interpret and visualize, and they can handle both categorical and numerical features. However, they are prone to *overfitting* when the tree is too deep, and they can be sensitive to small changes in the training data. The performance of this machine learning model is evaluated as below:

Accuracy score: 0.9148264984227129

	precision	recall	f1-score	support
0	0.73	0.92	0.81	86
1	0.68	0.48	0.57	93
2	0.71	0.65	0.68	88
3	0.75	0.77	0.76	79
4	0.78	0.77	0.78	102
5	0.86	0.94	0.90	88
6	0.96	0.99	0.97	98
accuracy			0.79	634
macro avg	0.78	0.79	0.78	634
weighted avg	0.79	0.79	0.78	634

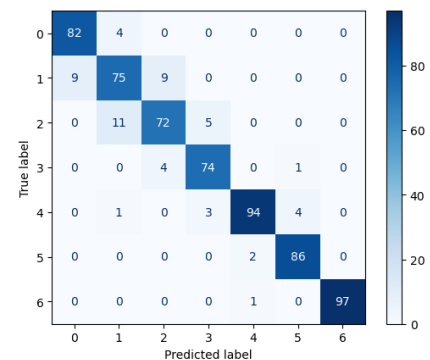


Figure 4.6: (a) Decision Tree score, (b) Decision Tree Confusion matrix

4.4 Random Forest Classification

Random forest is an ensemble learning method that combines multiple decision trees to improve the performance and reduce the *overfitting* of individual trees. The algorithm randomly selects a subset of features and a subset of training samples with replacement, and trains a decision tree on each subset. To predict the class label of a new sample, each tree in the forest independently predicts the class label based on the feature values of the new sample, and the class label with the highest frequency is assigned to the new sample. Random forest can handle both categorical and numerical features, and it can automatically select the important features by measuring their relative importance in the forest. Random forest is a powerful algorithm that can achieve high accuracy and robustness against outliers and noisy data, and it can be used for both classification and regression tasks. However, it can be computationally expensive and memory-intensive for large datasets and complex models. Finally, the model's performance is evaluated using the accuracy metric.

Accuracy score: 0.917981072555205

	precision	recall	f1-score	support
0	0.73	0.92	0.81	86
1	0.68	0.48	0.57	93
2	0.71	0.65	0.68	88
3	0.75	0.77	0.76	79
4	0.78	0.77	0.78	102
5	0.86	0.94	0.90	88
6	0.96	0.99	0.97	98
accuracy			0.79	634
macro avg	0.78	0.79	0.78	634
weighted avg	0.79	0.79	0.78	634

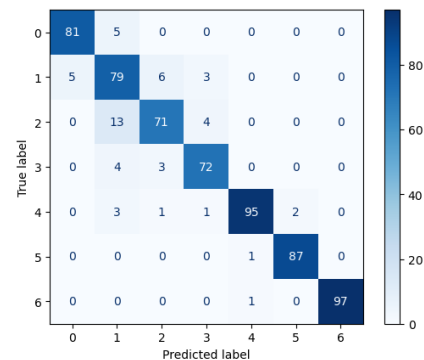


Figure 4.7: (a) Random Forest score, (b) Random Forest Confusion matrix

4.5 Neural Network Classification

Neural networks are a popular machine learning approach that are widely used for multi-class classification problems. A neural network is composed of interconnected nodes, or neurons, that receive inputs from other neurons, process them using activation functions, and produce outputs that are propagated to other neurons. Neural networks can be used to learn complex nonlinear relationships between the input features and the target labels, and they can automatically extract relevant features from the data through a process known as feature learning. The most common neural network architecture for multi-class classification is the feedforward Neural Network, which consists of an input layer, one or more hidden layers, and an output layer. The output layer typically uses a

softmax activation function to produce probabilities of the target classes, and the predicted class is the one with the highest probability. Neural networks can be trained using various optimization algorithms, such as stochastic gradient descent, and regularization techniques, such as dropout and weight decay, to prevent *overfitting* and improve generalization. However, neural networks are computationally intensive and require large amounts of data and careful tuning of the hyperparameters. Neural networks can be useful for tasks that involve complex and high-dimensional data, such as image recognition, speech recognition, and natural language processing.

```
tf.random.set_seed(987)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(15, activation = 'relu'),
    tf.keras.layers.Dense(10, activation = 'relu'),
    tf.keras.layers.Dense(10, activation = 'relu'),
    # tf.keras.layers.Dense(10, activation = 'relu'),
    # tf.keras.layers.Dense(10, activation = 'relu'),
    tf.keras.layers.Dense(len(class_names), activation = 'softmax'),
])

model.compile( loss= tf.keras.losses.SparseCategoricalCrossentropy(),
               optimizer = tf.keras.optimizers.Adam(learning_rate=0.003),
               metrics=['accuracy']
            )
```

Figure 4.8: Neural network models

This is a neural network model created using the *TensorFlow Keras API*. It is a sequential model, meaning that the layers are stacked on top of each other in a linear fashion. The first three layers are dense layers, meaning that each neuron is connected to every neuron in the previous layer. The activation function used for all three dense layers is *ReLU* (rectified linear unit), which helps to introduce non-linearity into the model and improve its ability to capture complex relationships in the data. The last layer is also a dense layer with the number of neurons equal to the number of classes in the dataset, and uses the *softmax* activation function to output probabilities for each class. The loss function used for training the model is *SparseCategoricalCrossentropy*, which is commonly used for multi-class classification problems. The *Adam* optimizer with a *learningrate* of 0.003 is used to optimize the model's parameters during training.

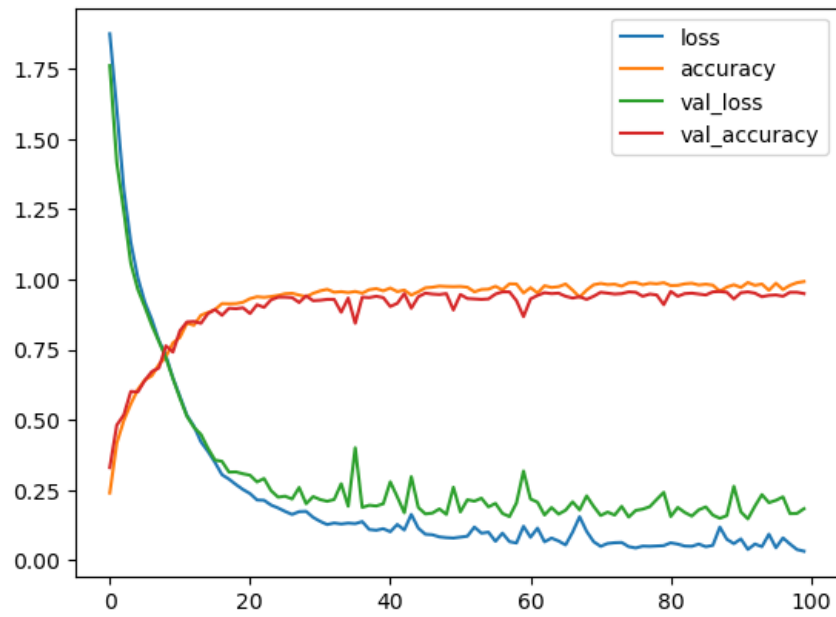


Figure 4.9: Neural network models

After defining and compiling a neural network model, the next step is to fit the model to the training data using the `fit()` method. The `fit()` method updates the weights and biases of the neural network based on the input data and the specified loss function and optimization algorithm. The `fit()` method also provides several optional parameters, such as the number of epochs, batch size, and validation data, which can be used to fine-tune the training process and improve the performance of the model. The history object returned by the `fit()` method contains a record of the training and validation metrics for each epoch, such as the loss and accuracy, which can be visualized using various data visualization techniques. In the given code snippet, the history object is used to create a pandas `DataFrame` and plot the training and validation loss and accuracy curves over the 100 epochs. This can help in monitoring the training process and identifying any issues, such as *overfitting* or *underfitting*, that may arise during the model training. Finally, the model's performance is evaluated using the accuracy metric.

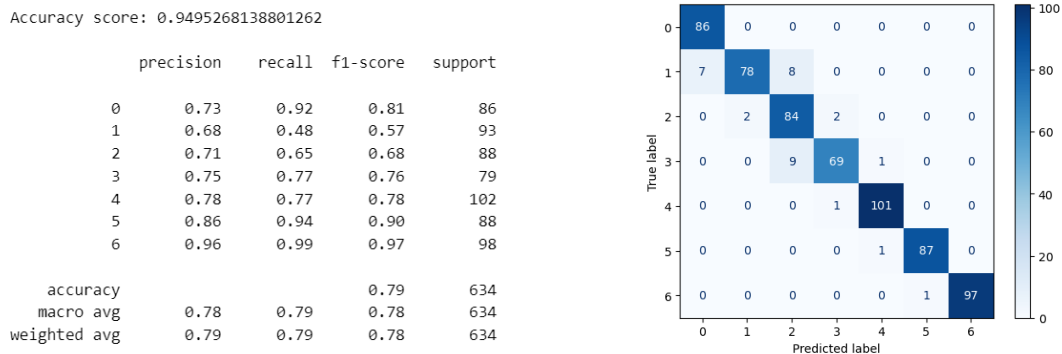


Figure 4.10: (a) Neural Network score, (b) Neural Network Confusion matrix

CHAPTER 5. Testing the models on new dataset

5.1 Overview

In this study, we have explored the performance of various machine learning algorithms for classifying obesity levels based on several features related to eating habits and physical condition. The dataset used for this study contained 2111 observations with 17 features and labels. We used supervised multiclass classification models, including KNN, decision tree, random forest, SVM, and neural networks, to predict the obesity levels of the subjects. After preprocessing the data, we split the data into training and testing sets, and then trained and evaluated the models using various metrics, such as accuracy and confusion matrix. The results showed that all models performed well in classifying the obesity levels, with the highest accuracy achieved by the neural network model at 0.95, outperforming the other models. The SVM model achieved an accuracy of 0.878, while the KNN model achieved 0.79. The decision tree and random forest models achieved 0.914 and 0.917 accuracy, respectively. These results indicate that machine learning models can be effective in predicting obesity levels based on various features, and the choice of the model depends on the specific requirements and objectives of the application.

5.2 Creating new dataset

To test the performance of the machine learning models trained on the original dataset, a new dataset can be created with random values for the features. In this case, a two-row dataset named *test_data* was created with random values for each feature. The *ran_test* variable stores the transformed version of this dataset using the column transformer object *ct*, which was previously fitted on the training data. The purpose of transforming the new data is to apply the same preprocessing steps as in the training data, such as one-hot encoding and scaling, to ensure that the new data is consistent with the training data. The transformed *ran_test* dataset can then be used to predict the obesity level using the previously trained machine learning models.

**Figure 5.1:** Creating new testing dataset with 2 records

Test on Decision Tree

```
for i in dt_model.predict(ran_test):
    print(class_names[i])
```

✓ 0.0s

Normal_Weight
Insufficient_Weight

Test on KNN

```
for i in knn_model.predict(ran_test):
    print(class_names[i])
```

✓ 0.0s

Obesity_Type_I
Overweight_Level_I

Test on Random Forest

```
for i in rf_model.predict(ran_test):
    print(class_names[i])
```

✓ 0.0s

Normal_Weight
Insufficient_Weight

Test on Decision Tree

```
for i in dt_model.predict(ran_test):
    print(class_names[i])
```

✓ 0.0s

Normal_Weight
Insufficient_Weight

Figure 5.2: Result of each Machine Learning model

Test on Neural Network Classification

```
for i in neural_nets_classification(model, ran_test):
    print(class_names[i])
```

✓ 0.1s

1/1 [=====] - 0s 31ms/step
Normal_Weight
Insufficient_Weight

Figure 5.3: Result of Neural network Classification

As can be seen, when the new test data was passed to the KNN classifier, the model performed poorly compared to the other machine learning models. This could be because KNN is a lazy algorithm that relies on nearest neighbors to classify new data points, and may not generalize well to new data that is significantly different from the training data. On the other hand, the other models - SVM, Decision Tree, Random Forest and Neural Network Classification performed well on the new test data. This suggests that these models were able to capture the underlying patterns in the data and make accurate predictions for unseen data. Therefore, it is important to carefully select

the appropriate machine learning algorithm based on the characteristics of the data and the desired outcome.

CHAPTER 6. CONCLUSIONS

In this report, we explored the use of machine learning models to classify obesity levels based on various features. We used a dataset from Kaggle.com with 2111 rows and 17 columns. The dataset contains features related to eating habits and physical conditions, and the labels are categorized into seven levels of obesity. We used supervised multi-class classification machine learning models such as K Nearest Neighbors, Decision Tree, Random Forest, Support Vector Machine, and Neural Networks to predict obesity levels. We performed data cleaning, visualization, preprocessing, and scaling before training the models. After evaluating the models using accuracy score, we found that SVM, Decision Tree, Random Forest, and Neural Network Classifier have a good performance, while KNN performs poorly. Finally, we created a new test dataset to assess the models' performance, and we found that the models (SVM, Decision Tree, Random Forest and Neural Network) performed well on the new test data. Overall, this study highlights the potential of machine learning models in classifying obesity levels, and neural networks appear to be a promising approach for accurate prediction.