TITLE: OBESITY CLASSIFICATION USING MACHINE LEARNING ON LIFESTYLE AND HEALTH PARAMETERS

Contributor 1

Fariha Ferdous [22201725]

Computer Science and Engineering

Contributor 2

Amir Sakib Saad [22299222]

Computer Science and Engineering

Course Code: CSE422

Semester: Spring 25

Submitted to

Mahjabin Chowdhury [MCW] & MD Mazed Hossain [CMZH]

TABLE OF CONTENTS

Introduction

Overweight and obesity are defined as abnormal or excessive fat accumulation that presents a risk to health. There are several factors which cause obesity. Some are genetic and some are havital. This project aims to make a deep analysis on the factors and parameters for which obesity is caused for and stand an algorithmic analysis with visual representation of the data via some machine learning model and make a prediction of if the person is under which obesity category. The core motivation behind this project is to make awareness as well as track an individuals' body condition especially those who are concerned about their lifestyle.

Dataset Description

Number of Features in the Dataset

The number of features simply demonstrate how many columns or how many parameters the dataset has including the output column. For this dataset, there are in total 16 columns with an output column (total 17) before any kind of data preprocessing. However, we can simply use *.shape* to see the dimension of the dataset. The output contains 2 numerical integer values within a tuple. The first index value is the representation of column numbers in the dataset.

	Age	Gender	Height	Weight	CALC	FAVC	FCVC	NCP	SCC	SMOKE	CH2O	family_history_with_overweight	FAF	TUE	CAEC	MTRANS	NObeyesdad
0	21.0	Female	1.62	64.0	no	no	2.0	3.0	no	no	2.0	yes	0.0	1.0	Sometimes	Public_Transportation	Normal_Weight
1	21.0	Female	1.52	56.0	Sometimes	no	3.0	3.0	yes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight
2	23.0	Male	1.80	77.0	Frequently	no	2.0	3.0	no	no	2.0	yes	2.0	1.0	Sometimes	Public_Transportation	Normal_Weight
3	27.0	Male	1.80	87.0	Frequently	no	3.0	3.0	no	no	2.0	no	2.0	0.0	Sometimes	Walking	Overweight_Level_I
4	22.0	Male	1.78	89.8	Sometimes	no	2.0	1.0	no	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation	Overweight_Level_II
5	29.0	Male	1.62	53.0	Sometimes	yes	2.0	3.0	no	no	2.0	no	0.0	0.0	Sometimes	Automobile	Normal_Weight
6	23.0	Female	1.50	55.0	Sometimes	yes	3.0	3.0	no	no	2.0	yes	1.0	0.0	Sometimes	Motorbike	Normal_Weight
7	22.0	Male	1.64	53.0	Sometimes	no	2.0	3.0	no	no	2.0	no	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight
8	24.0	Male	1.78	64.0	Frequently	yes	3.0	3.0	no	no	2.0	yes	1.0	1.0	Sometimes	Public_Transportation	Normal_Weight
9	22.0	Male	1.72	68.0	no	yes	2.0	3.0	no	no	2.0	yes	1.0	1.0	Sometimes	Public_Transportation	Normal_Weight

Type of Machine Learning Problem

Based on our dataset, the output column contains seven different types of obesity which highlighted emphatically that the dataset is a categorical problem. The model needs to determine if the person is under normal weight, overweight or categorized under any types of obesity level based on the rest 15 parameters. As we will use mathematical equations to build our model, we will convert this column into numerical data later via encoding.

```
1  dataset["NObeyesdad"].unique()
```

```
array(['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'], dtype=object)
```

Total Number of Data Points

Data points basically represent the total number of rows a dataset set has before applying any kind of data preprocessing operations.

There are several methods to see the data points of a dataset among them two popular methods have been highlighted here. We can use *.info()* and *.shape* on the dataset to see how many entries or data points are there. However, our dataset contains 2111 data points at the nascent stage (Before applying any data cleaning queries).

```
[65]  1  dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
```

```
1  dataset.shape
```

```
(2111, 17)
```

Types of Features

The dataset contains both Quantitative (float64) and Categorical (object) features. Among the categorical features there are:

Fetching Object Type Columns

```
[69]  1  dataset.select_dtypes(include="object").columns
```

```
Index(['Gender', 'CALC', 'FAVC', 'SCC', 'SMOKE',
       'family_history_with_overweight', 'CAEC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

Fetching Numeric Type Columns

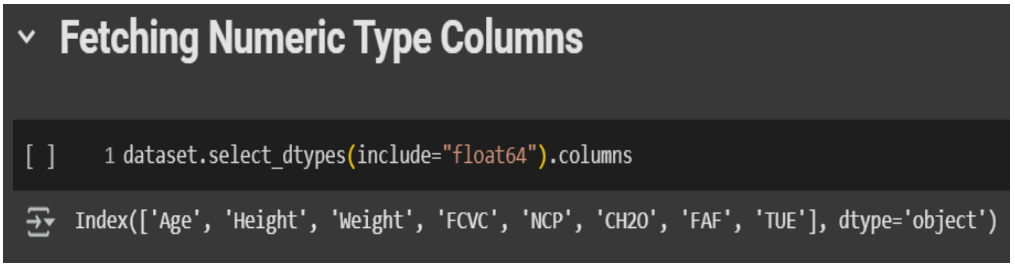
```
[70]  1  dataset.select_dtypes(include="float64").columns
```

```
Index(['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE'], dtype='object')
```

Gender	CALC	FAVC
SCC	SMOKE	family_history_with_overweight
CAEC	MTRANS	NObeyesdad

These types of features' values can be classified into different categories and thus can be called categorical features.

On the other hand, quantitative features include numeric values such as:



Age	Height	Weight	FCVC
NCP	CH2O	FAF	TUE

.select_dtypes() function can be used to figure out whether a feature in the dataset falls under a quantitative or a categorical one. We need to pass the parameter "object" in the function to fetch the categorical features and "float64" to fetch the quantitative features.

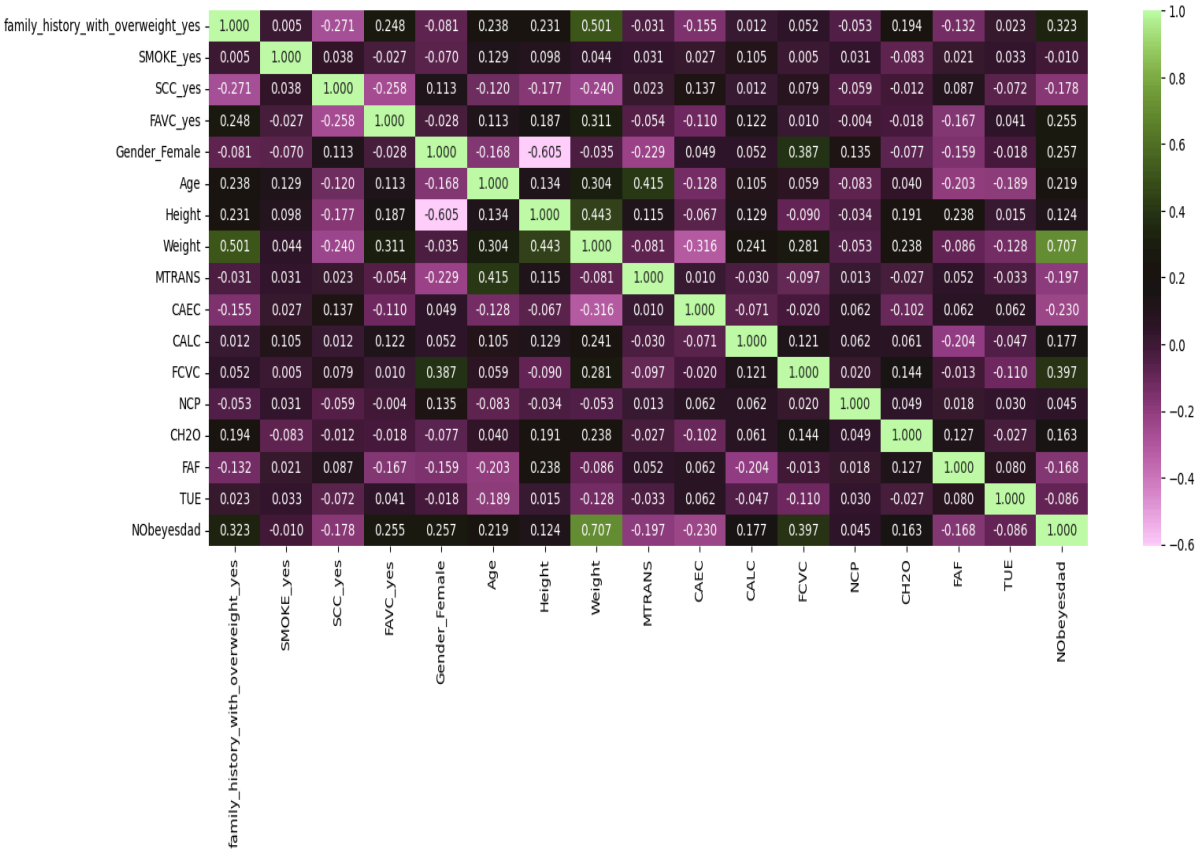
Correlation of all the features

Matrix representation of Correlation

```
1 dataset.corr()
```

	family_history_with_overweight_yes	SMOKE_yes	SCC_yes	FAVC_yes	Gender_Female	Age	Height	Weight	MTRANS	CAEC	CALC	FCVC	NCP	CH2O	FAF	TUE	NObeyesdad
family_history_with_overweight_yes	1.000000	0.005181	-0.271021	0.247972	-0.081159	0.238124	0.231034	0.500941	-0.031123	-0.155489	0.012293	0.052176	-0.052668	0.194172	-0.131693	0.023351	0.322903
SMOKE_yes	0.005181	1.000000	0.037741	-0.027160	-0.069593	0.128869	0.098275	0.043892	0.031021	0.026728	0.105148	0.005205	0.030630	-0.083140	0.020563	0.033384	-0.009974
SCC_yes	-0.271021	0.037741	1.000000	-0.258196	0.113083	-0.119784	-0.177147	-0.240160	0.023361	0.137182	0.012312	0.079212	-0.058990	-0.011704	0.086803	-0.071668	-0.177801
FAVC_yes	0.247972	-0.027160	-0.258196	1.000000	-0.027963	0.113107	0.186798	0.310725	-0.053550	-0.109900	0.121726	0.009690	-0.004039	-0.017881	-0.166947	0.041103	0.255338
Gender_Female	-0.081159	-0.069593	0.113083	-0.027963	1.000000	-0.167642	-0.605239	-0.035221	-0.228962	0.049481	0.052095	0.386731	0.135314	-0.076597	-0.158738	-0.018330	0.257151
Age	0.238124	0.128869	-0.119784	0.113107	-0.167642	1.000000	0.133900	0.303602	0.415356	-0.127955	0.105028	0.058826	-0.083457	0.039740	-0.202543	-0.188620	0.218582
Height	0.231034	0.098275	-0.177147	0.186798	-0.605239	0.133900	1.000000	0.442738	0.114843	-0.067416	0.129163	-0.090355	-0.033544	0.191211	0.237524	0.015441	0.124349
Weight	0.500941	0.043892	-0.240160	0.310725	-0.035221	0.303602	0.442738	1.000000	-0.081096	-0.316118	0.241068	0.280890	-0.052635	0.238085	-0.086270	-0.127825	0.707288
MTRANS	-0.031123	0.031021	0.023361	-0.053550	-0.228962	0.415356	0.114843	-0.081096	1.000000	0.010034	-0.029693	-0.096565	0.013167	-0.027488	0.052358	-0.032633	-0.196874
CAEC	-0.155489	0.026728	0.137182	-0.109900	0.049481	-0.127955	-0.067416	-0.316118	0.010034	1.000000	-0.071376	-0.020406	0.062367	-0.101556	0.062157	0.061558	-0.229673
CALC	0.012293	0.105148	0.012312	0.121726	0.052095	0.105028	0.129163	0.241068	-0.029693	-0.071376	1.000000	0.121388	0.062470	0.061333	-0.204146	-0.046547	0.176646
FCVC	0.052176	0.005205	0.079212	0.009690	0.386731	0.058826	-0.090355	0.280890	-0.096565	-0.020406	0.121388	1.000000	0.019638	0.144310	-0.013160	-0.110148	0.397369
NCP	-0.052668	0.030630	-0.058990	-0.004039	0.135314	-0.083457	-0.033544	-0.052635	0.013167	0.062367	0.062470	0.019638	1.000000	0.049327	0.018179	0.029654	0.045437
CH2O	0.194172	-0.083140	-0.011704	-0.017881	-0.076597	0.039740	0.191211	0.238085	-0.027488	-0.101556	0.061333	0.144310	0.049327	1.000000	0.126502	-0.027273	0.163200
FAF	-0.131693	0.020563	0.086803	-0.166947	-0.158738	-0.202543	0.237524	-0.086270	0.052358	0.062157	-0.204146	-0.013160	0.018179	0.126502	1.000000	0.080350	-0.168417
TUE	0.023351	0.033384	-0.071668	0.041103	-0.018330	-0.188620	0.015441	-0.127825	-0.032633	0.061558	-0.046547	-0.110148	0.029654	-0.027273	0.080350	1.000000	-0.085525
NObeyesdad	0.322903	-0.009974	-0.177801	0.255338	0.257151	0.218582	0.124349	0.707288	-0.196874	-0.229673	0.176646	0.397369	0.045437	0.163200	-0.168417	-0.085525	1.000000

Heatmap representation of Correlation



Insights From the Correlation Test

Before explaining the heatmap first we need to understand what the color code denotes.

Here 0.0 is the neutral point means neither the correlation is high nor the correlation is low. From the right side (-0.2 to -0.6) denotes the correlation between any two columns that are low-lower-lowest. On the other hand, the left side (0.2 to 1.0) denotes the correlation between any two columns that are high-higher-highest.

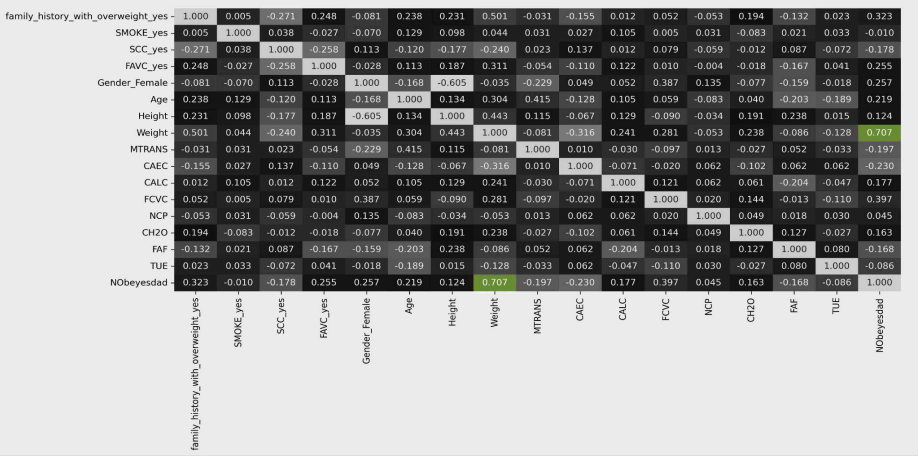
This color pattern creates a nice texture and color representation of the data distribution. The heatmap always has self-correlations points for the homogenous columns' linear relationship.

To understand the correlation heatmap lets first assume that the entire map is a 2D adjacency matrix. Now we need to check the point of intersection for every vertical and horizontal line simultaneously. (e.g: there exists any column X on the horizontal axis that has the highest correlation value (1.000) with the same column on the vertical axis.

Now the one question might be arised that, what are the other values represented for? To answer this question let's pick 2 pairs of columns and break down their correlations. (NObeyesdad is the output feature for this dataset)

Pair 1: The correlation between NObeyesdad & Weight

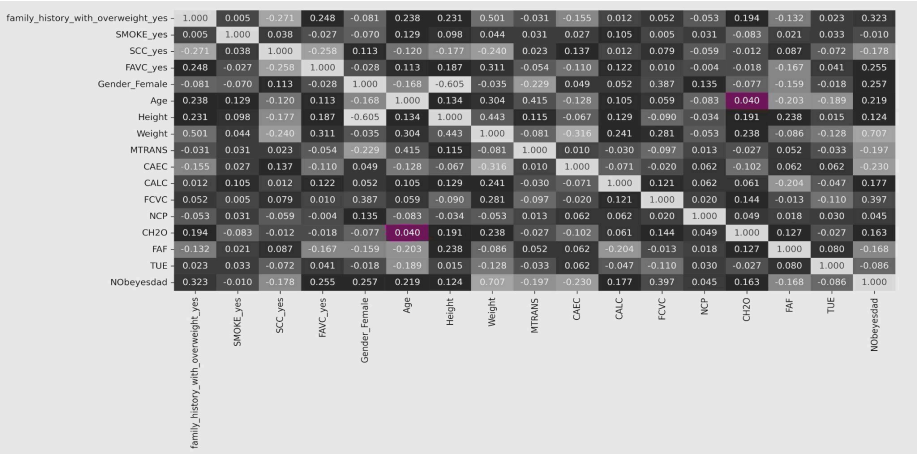
The correlation between the NObeyesdad and Weight is 0.707. The value has been highlighted in the given figure for a good visual and conspicuous understanding.



From here we can make an assumption and come to a final conclusion that whether a person is suffering from obesity or not is highly correlated with his/her body weight.

Pair 2: The correlation between CH2O & Age

The correlation between the NObeyesdad and Weight is 0.040. The value has been highlighted in the given figure for a good visual and conspicuous understanding.



From here we can make an assumption and come to a final conclusion that whether a person is consuming water or not is not as much correlated with his/her age as the correlation of the aforementioned comparison.

Correlation (r)	Interpretation
+1.0	Perfect positive correlation
0.7 - 0.9	Strong positive correlation
0.4 - 0.6	Moderate positive correlation
0.1 - 0.3	Weak positive correlation
0.0	No correlation
-0.1 - -0.3	Weak negative correlation
-0.4 - -0.6	Moderate negative correlation
-0.7 - -0.9	Strong negative correlation
-1.0	Perfect negative correlation

It is necessary to drop the highly correlated columns from the dataset because highly correlation of two or many columns can create noise, overfitting issues as well as reduced interpretability. So it is best practice to set a threshold limit and drop the columns before sending them into the training phase.

Distribution of Unique Classes in the Output Feature

This code targets the dataset and fetches the column named “Nobeyesdad”. After that it is grouped by the unique class and simultaneously counts the number of appearances of a particular column class in the dataset and then enumerates the result in the terminal.

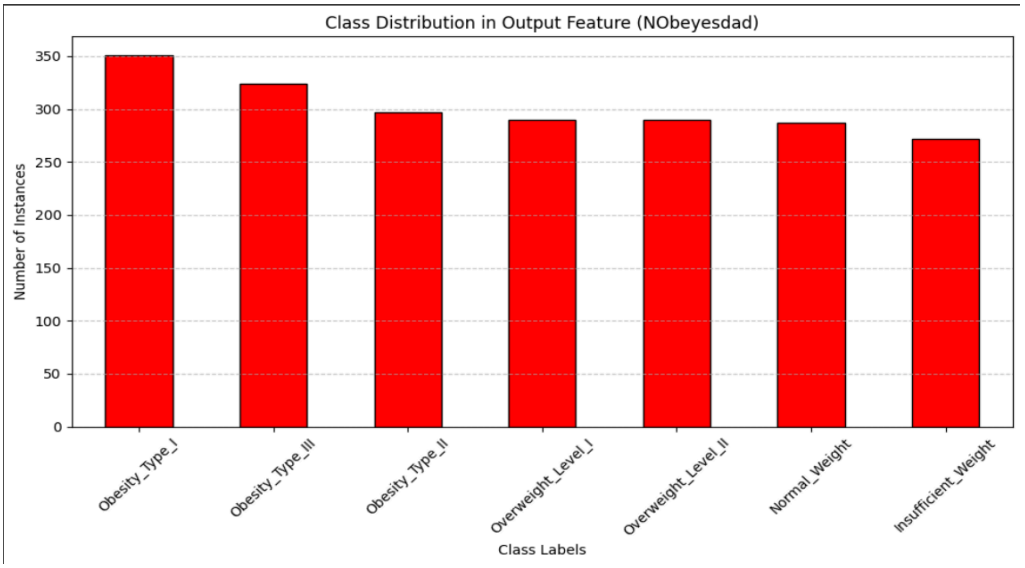
Data Distribution of Output Column

```
1  class_counts = dataset['NObeyesdad'].value_counts()
2  print(class_counts)
```

```
NObeyesdad
Obesity_Type_I         351
Obesity_Type_III       324
Obesity_Type_II        297
Overweight_Level_I     290
Overweight_Level_II    290
Normal_Weight          287
Insufficient_Weight    272
Name: count, dtype: int64
```

Here we can see that the appearance of the data is not equal. Hence it can be said that all unique classes don't have an equal number of instances for this dataset.

Bar Chart Representation of Class Distribution



Exploratory Data Analysis (EDA)

It’s the process of examining and understanding a dataset before modeling or drawing conclusions. EDA involves summarizing the main characteristics of the data using visual methods (like histogram, box plots, scatter plots) and statistical techniques (like mean, median, standard deviation, correlation etc.). EDA helps to understand the structure, patterns, and relationships of the data. It can show which features might be useful, which ones are redundant and variables interact with each

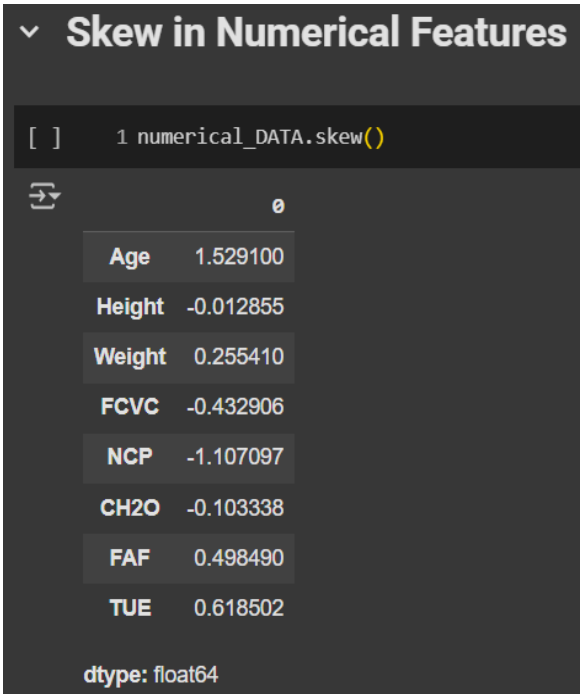
other. Better understanding of the data often leads to better preprocessing, which leads to more accurate models. Here we will see the deepest exploration of the data distributions of the dataset by plotting the data into the two dimensional graph and then try to compare between or among miscellaneous data. We will analyze the data distribution, skewness and the density of the data along with finding the outliers using boxplot.

Skewness in numerical features:

The skewness of the numerical features in the dataset reveals important patterns in participant behaviour and characteristics. Skewness measures the asymmetry of the distribution of a dataset.

Skewness = 0 → Perfectly symmetrical (e.g., normal distribution)
Skewness < 0 → Left-skewed (Long tail on the left)
Skewness > 0 → Right-skewed (Long tail on the right)

It helps to understand how balanced the data is and whether transformations are needed.



Skewness Value Interpretation:

Value	Interpretation
0	Symmetrical (normal-ish)
0 to ±0.5	Fairly symmetrical
±0.5 to ±1	Moderately skewed
Greater than ±1	Highly skewed

Skewness interpretation of the numeric features of the dataset:
Age(1.529) : Strongly Right-skewed. Most data is clustered on the left (younger ages), with fewer older participants stretching the tail on the right.

Height(-0.013) : Almost symmetrical - no transformation needed. Values are centered around the mean with balanced spread on both sides.

Weight(0.255) : Slightly Right-skewed. Most people have moderate weight; a few are heavier, stretching the right tail.

FCVC(-0.433) : Slightly left-skewed. Majority report higher vegetable consumption, fewer report low consumption.

NCP(-1.107) : Strong left-skew. Most people eat 3-4 meals, with fewer participants reporting lower values.

CH2O(-0.103) : Almost symmetrical. Water intake is fairly balanced; no transformation needed.

FAF(0.498) : Slightly right-skewed. Most people are less physically active; some engage in high activity levels.

TUE(0.619) : Moderately right-skewed. Most people spend little time on technology, but some spend a lot; stretching the right tail.

Density graph representation:

Density graph represents the distribution of a continuous variable. It shows where the data values are concentrated over the range of the variable.

According to the density graph:

Age : Majority of the participants are between 15 and 20 years old.

Height : Most values are concentrated between 1.6 m and 1.75 m.

Weight : Bulk of the data lies between 50 kg and 120 kg.

FCVC (*Frequency of vegetable consumption*) : Most people fall between 2 and 3. (indicating frequent consumption)

NCP (*Number of main meals consumed per day*) : Most people eat 3 meals a day.

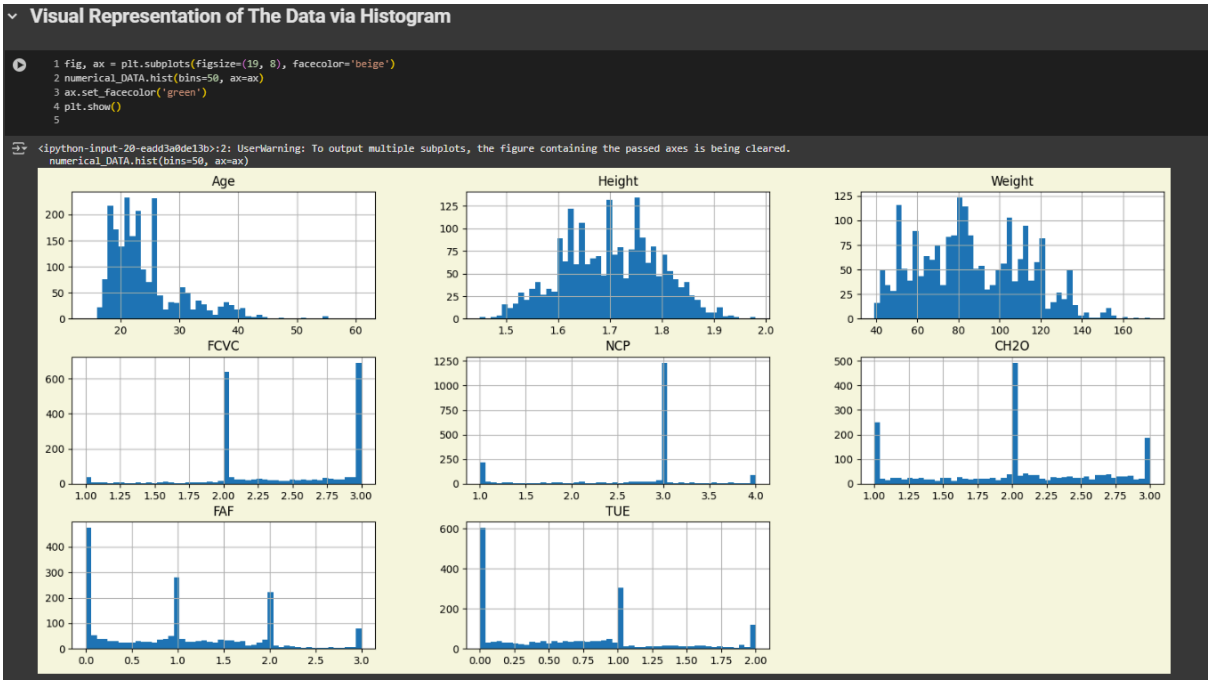
CH2O (*Daily water intake*) : The data is distributed between 1 to 3 liters per day with the majority of them concentrated in 2 liters per day.

FAF (*Physical activity frequency*) : Most values lie between 0 and 2 hours per day.

TUE (Time using technology) : The data lies between 0 to 2 hours per day. But the majority of the people spend less time with technology.

Histogram representation :

A histogram is a graphical representation that shows the frequency distribution of a continuous (or sometimes discrete) variable. It helps visualize how often different ranges of values occur.



According to the histogram:

Age : Most values fall between 15 and 25.

Height : Data is fairly evenly distributed between 1.6 m and 1.75 m, with a normal-like shape and no extreme skew.

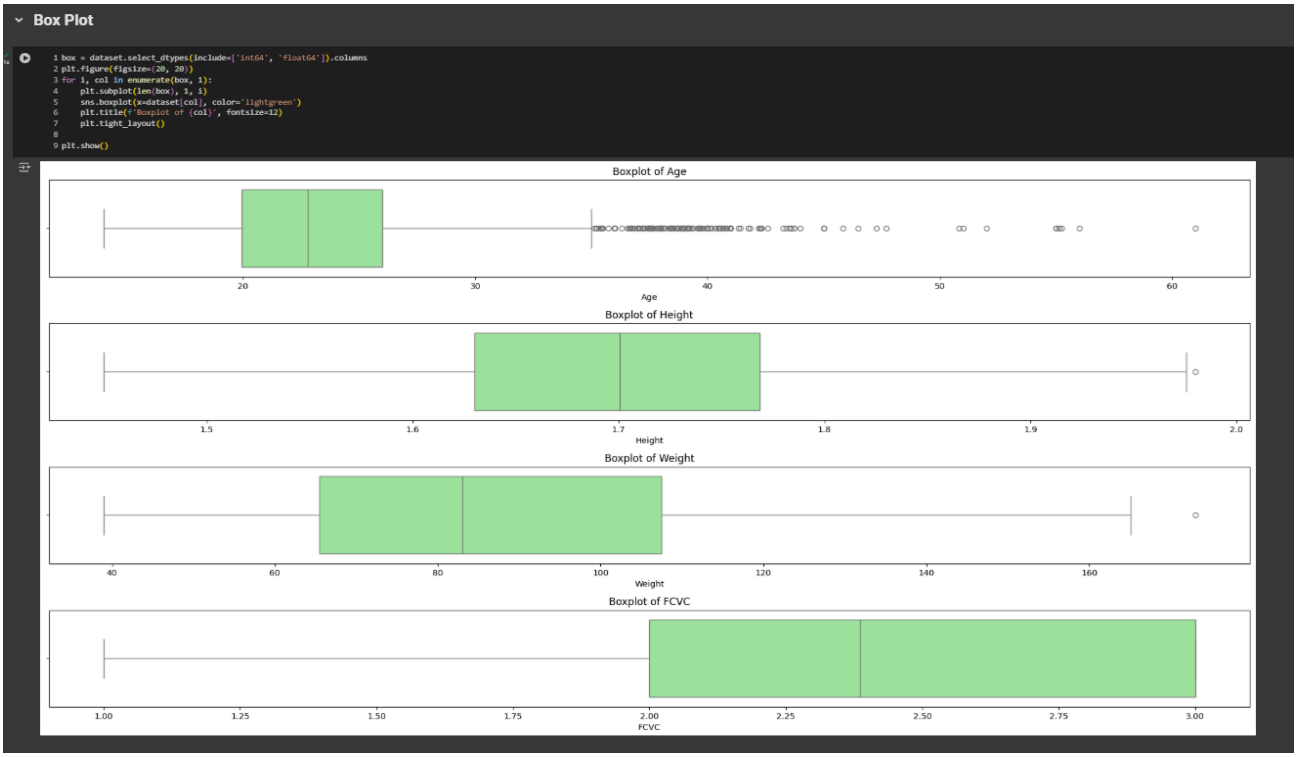
Weight : Most values are between 50 kg and 120 kg. There is a visible right tail, indicating some participants with higher weights.

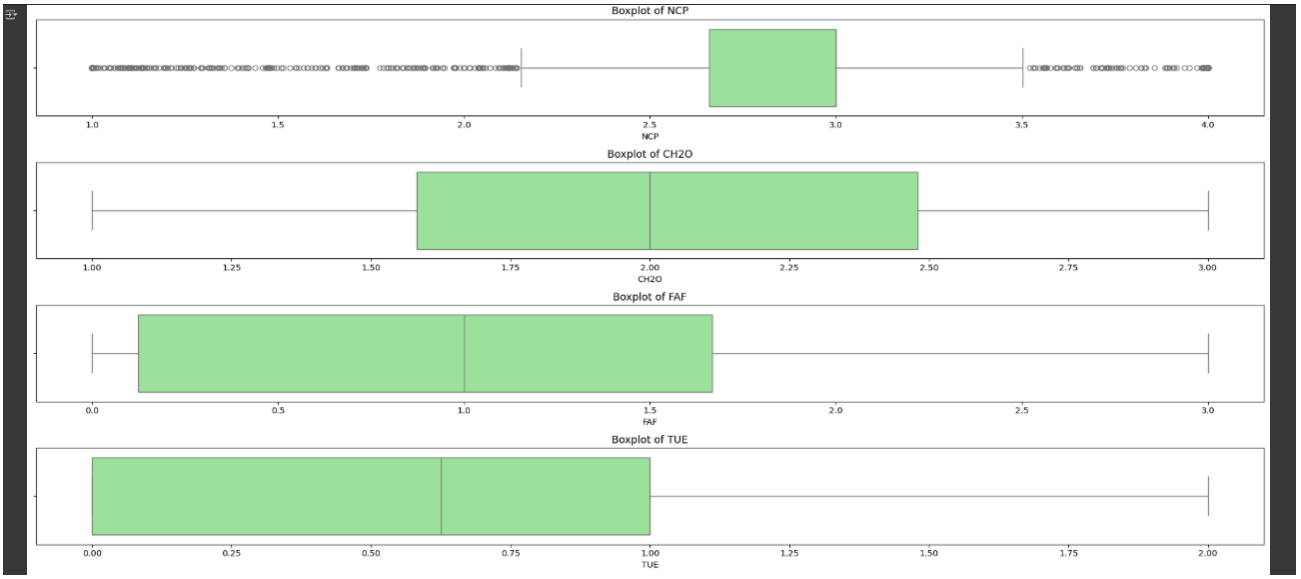
FCVC (Frequency of vegetable consumption) : The majority of responses are clustered near 2 and 3, showing high vegetable intake for most.
NCP (Number of main meals consumed per day) : Most people reported eating 3 meals per day. Some values are clustered near 1 and 4.
CH2O (Daily water intake) : Peaks around 1 to 3 liters per day, showing most participants drink a moderate amount of water.
FAF (Physical activity frequency) : Most participants reported 0 to 2 hours per day, But the tendency of the participants to engage in physical activities seems less from the histogram.
TUE (Time using technology) : The data lies between 0 to 2 hours per day. But the majority of the people spend less time with technology, with a few very high usage outliers.

Box plot representation:

A box plot is a statistical graph that shows the spread and summary of a dataset, highlighting the central value, variability and potential outliers. The box plot can reveal the distribution of the data, including whether it's symmetric or skewed.

- Box (the rectangle) : The box itself represents the middle 50% of the data, with the lower edge at Q1 and the upper edge at Q3. The line inside the box marks the median (Q2).
- The whiskers : Lines extending from the box to the minimum and maximum values, excluding outliers. They represent the spread of the data outside the interquartile range (IQR).
- Outliers : Individual data points plotted outside the whiskers, indicating values that fall outside the typical range.





According to the box plots:

Age : Majority of values are clustered between 15 and 25 years There are a few outliers beyond 35, indicating a young participant pool with some older individuals.
Height : Most values are concentrated between 1.6 m and 1.75 m. The median height is around 1.7 m. The distribution is symmetrical with no noticeable outliers, suggesting consistent height data.
Weight : Shows a wider range, mainly between 65 to 110 kg. There are a few upper outliers indicating a few participants with significantly higher weights.
FCVC (Frequency of vegetable consumption) : Most responses are centered between 2 and 3, with no outliers. Indicates high and consistent vegetable intake.
NCP (Number of main meals consumed per day) : Most people eat 3 meals a day, But a significant amount of people also consume 1 and 4 meals per day. Showcasing both upper and lower outliers.
CH2O (Daily water intake) : Most participants drink between 1 and 3 liters per day.
FAF (Physical activity frequency) : Concentrated near 0 to 2 hours per day. But it ranges upto 3 hours per day.
TUE (Time using technology) : The data points are clustered within 0 to 1 hour per day. But a significant amount of reported upto 2 hours per day technology usage.

Statistical summary of the numerical features:

```
numerical_DATA = dataset.select_dtypes(include='number')
numerical_DATA.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	2111.0	24.312600	6.345968	14.00	19.947192	22.777890	26.000000	61.00
Height	2111.0	1.701677	0.093305	1.45	1.630000	1.700499	1.768464	1.98
Weight	2111.0	86.586058	26.191172	39.00	65.473343	83.000000	107.430682	173.00
FCVC	2111.0	2.419043	0.533927	1.00	2.000000	2.385502	3.000000	3.00
NCP	2111.0	2.685628	0.778039	1.00	2.658738	3.000000	3.000000	4.00
CH2O	2090.0	2.009034	0.613164	1.00	1.582010	2.000000	2.479917	3.00
FAF	2111.0	1.010298	0.850592	0.00	0.124505	1.000000	1.666678	3.00
TUE	2111.0	0.657866	0.608927	0.00	0.000000	0.625350	1.000000	2.00

The numerical features show moderate variability across the dataset. Age has a mean of 24.3 with a standard deviation of 6.35, indicating a generally young population but with participants spanning from 14 to 61 years. Height is fairly consistent with low variation (std ~0.10), suggesting most people fall within a typical height range (1.7 m). Weight shows high variability (std ~26.19), reflecting a diverse body mass range, with a median of 83 kg and values extending up to 173 kg. Features like FCVC (vegetable intake), CH2O (water intake), and NCP (meals per day) are clustered around their means with relatively small standard deviations, showing consistent dietary habits. FAF (physical activity) and TUE (technology usage), however, display wider spreads—especially FAF, which ranges from 0 to 3, Although the standard deviation is less than 1, the range is wide (from no activity to daily activity), indicating diverse behaviour across individuals. Overall, while some lifestyle behaviors are consistent, others like weight, activity, and tech use show greater diversity.

Statistical summary of the categorical features:

```
categorical_DATA = dataset.select_dtypes(include= 'object')
categorical_DATA.describe().T
```

	count	unique	top	freq
Gender	2100	2	Male	1065
CALC	2111	4	Sometimes	1401
FAVC	2111	2	yes	1866
SCC	2111	2	no	2015
SMOKE	2111	2	no	2067
family_history_with_overweight	2090	2	yes	1706
CAEC	2111	4	Sometimes	1765
MTRANS	2111	5	Public_Transportation	1580
NObeyesdad	2111	7	Obesity_Type_I	351

Most categorical features have complete or near-complete data, with counts close to 2111 entries. Each feature generally has low cardinality (mostly 2 to 5 unique values), indicating clearly defined categories. For example, Gender has 2 unique values with "Male" being the most frequent (1065 out of 2100). FAVC (high-calorie food consumption) is dominated by "yes" responses (1866), while SCC (calorie consciousness) shows a large majority answering "no" (2015), suggesting low dietary awareness. CALC (alcohol intake) and CAEC (snacking) both have 4 unique categories, with "Sometimes" being the most common response in both cases. SMOKE and family_history_with_overweight are both strongly skewed toward "no" and "yes" respectively, with counts of 2067 and 1706. MTRANS shows that public transportation is the most used mode (1580 out of 2111), and the target variable NObeyesdad includes 7 classes, with "Obesity_Type_I" being the most frequent (351). This suggests that while some lifestyle behaviors are diverse, many responses cluster around dominant habits.

Variance of each numerical features:

Weight shows the highest variance (685.98), indicating a wide spread of body weights across participants. Age also has a relatively high variance (40.27), reflecting the inclusion of both younger and older individuals. On the other hand, height has very low variance (0.009), suggesting that participants are similar in height. Lifestyle features like FCVC (vegetable intake), NCP (meals per day), CH2O (water intake), FAF (physical activity), and TUE (technology use) show moderate variance, indicating some variation in daily habits but within a consistent range. Overall, while physical measures like weight and age vary significantly, most lifestyle-related numeric features are more tightly clustered.

Varience of The Data

```
1 numerical_DATA.var()
```

	0
Age	40.271313
Height	0.008706
Weight	685.977477
FCVC	0.285078
NCP	0.605344
CH2O	0.375970
FAF	0.723507
TUE	0.370792

Unique values in each column:

Most features in the dataset have low cardinality, which is typical for categorical and behavioral data. For example, features like Gender, FAVC, SMOKE, SCC, family_history_with_overweight each have only 2 unique values, indicating binary responses (e.g., yes/no). CALC and CAEC have around 4 unique values, reflecting frequency-based categorical responses like "no", "sometimes", or "frequently." The MTRANS (transportation) feature has **5** unique values, showing slightly more variety in commuting habits.
Among numeric features, Age, Weight, and Height have high unique counts, showing continuous or near-continuous distributions. Lifestyle scores such as FCVC, NCP, FAF, TUE, and

Unique Values of Each Column

```
1 dataset.nunique()
```

	0
Age	1402
Gender	2
Height	1574
Weight	1525
CALC	4
FAVC	2
FCVC	810
NCP	635
SCC	2
SMOKE	2
CH2O	1260
family_history_with_overweight	2
FAF	1190
TUE	1129
CAEC	4
MTRANS	5
NObeyesdad	7

CH2O have moderate uniqueness, with values often reported on a scale or range, showing behavioral variation but within limits.

Finally, the target column NObeyesdad has 7 unique values, representing distinct obesity classifications.

Data Pre-processing

NULL Values Problem

In our dataset there are a vast number of columns that contain NULL value. A NULL value does not hold any kind of information which creates a prediction mismatch while training the application of a machine learning model. A NULL value can be represented as a miscellaneous form of data. Sometimes it might be represented as NAN which is actually an empty data, sometimes as ERROR, NOT GIVEN etc. For this dataset the NULL values are represented as NAN for both categorical and numerical data.

	Age	Gender	Height	Weight	CALC	FAVC	FCVC	NCP	SCC	SMOKE	CH2O	family_history_with_overweight	FAF	TUE	CAEC	MTRANS	NObeyesdad
48	20.000000	NaN	1.660000	60.000000	Sometimes	no	3.000000	3.000000	no	no	2.000000	yes	1.000000	0.000000	Sometimes	Walking	Normal_Weight
70	23.000000	Female	1.650000	80.000000	no	yes	2.000000	3.000000	no	no	2.000000	NaN	0.000000	2.000000	Always	Public_Transportation	Overweight_Level_II
71	22.000000	Female	1.670000	50.000000	Sometimes	no	3.000000	3.000000	yes	no	NaN	yes	2.000000	1.000000	no	Public_Transportation	Insufficient_Weight
125	18.000000	NaN	1.750000	80.000000	Frequently	yes	2.000000	3.000000	no	no	2.000000	yes	0.000000	0.000000	Always	Public_Transportation	Overweight_Level_I
143	34.000000	Female	1.680000	75.000000	Sometimes	yes	3.000000	1.000000	no	no	NaN	no	0.000000	0.000000	Sometimes	Automobile	Overweight_Level_I
157	21.000000	Male	1.670000	60.000000	Sometimes	yes	2.000000	3.000000	no	no	NaN	yes	2.000000	1.000000	Frequently	Public_Transportation	Normal_Weight
196	22.000000	Male	1.750000	74.000000	Sometimes	no	2.000000	3.000000	no	no	2.000000	NaN	1.000000	2.000000	Sometimes	Bike	Normal_Weight
205	23.000000	Female	1.600000	78.000000	Frequently	yes	2.000000	1.000000	no	yes	NaN	yes	1.000000	0.000000	Sometimes	Public_Transportation	Obesity_Type_I
263	23.000000	NaN	1.740000	53.000000	no	yes	2.000000	3.000000	no	no	1.000000	no	3.000000	2.000000	Frequently	Public_Transportation	Insufficient_Weight
274	25.000000	Female	1.610000	61.000000	Sometimes	no	2.000000	3.000000	no	no	NaN	no	2.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
281	18.000000	Male	1.700000	55.000000	Sometimes	yes	2.000000	3.000000	no	no	NaN	yes	0.000000	0.000000	Frequently	Public_Transportation	Normal_Weight
284	20.000000	Male	1.770000	70.000000	Sometimes	yes	1.000000	1.000000	no	no	NaN	yes	1.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
314	26.000000	Female	1.650000	63.000000	no	yes	3.000000	3.000000	no	no	NaN	no	1.000000	0.000000	Sometimes	Walking	Normal_Weight
324	30.000000	Female	1.650000	71.000000	Sometimes	yes	2.000000	3.000000	no	no	1.000000	NaN	0.000000	0.000000	Sometimes	Public_Transportation	Overweight_Level_I
330	17.000000	Female	1.800000	50.000000	Sometimes	yes	3.000000	4.000000	no	no	NaN	no	2.000000	1.000000	Frequently	Public_Transportation	Insufficient_Weight
420	18.000000	Male	1.850000	60.000000	Sometimes	yes	3.000000	4.000000	yes	no	2.000000	NaN	2.000000	0.000000	Sometimes	Automobile	Insufficient_Weight
463	17.000000	Male	1.800000	68.000000	Sometimes	no	2.000000	3.000000	no	no	NaN	yes	2.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight

```
1  NULL = dataset.isnull().sum().sum()
2  print(f"Total NULL Values: {NULL}")
```

```
Total NULL Values: 53
```

Total NULL Values of Each Columns

```
1  dataset.isnull().sum()
```

	0
Age	0
Gender	11
Height	0
Weight	0
CALC	0
FAVC	0
FCVC	0
NCP	0
SCC	0
SMOKE	0
CH2O	21
family_history_with_overweight	21
FAF	0
TUE	0
CAEC	0
MTRANS	0
NObeyesdad	0

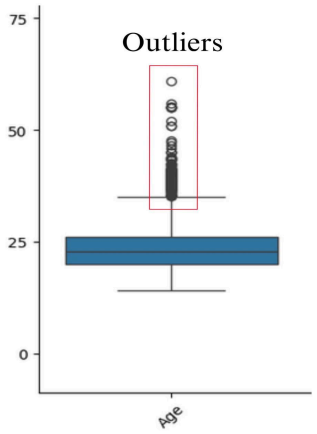
WE can easily detect the NULL value using *.isnull()*. It will aggregate the number of repetition of the NULL value for a particular column and show them as a dataframe as output. Additionally, to see the total number of NULL values in the dataset, we can use the *.sum()* function over *.isnull()* and it will again aggregate the whole columns' NULL value.

Applying any machine learning algorithm with NULL value may

reduce the prediction accuracy of the algorithm as the NULL value does not represent any kind of information for the dataset.

Outliers Problem

Assume a real life scenario where a google form has been distributed all over the country in order to see the obesity percentages. Around 100 people responded to the queries given that 8-10 people came out to be Sumo Wrestlers (An individual whose weight is higher than that of another person). These 8-10 people are the outliers.



Outliers are some data that are not represented as the regular data distribution and these outliers can create a prediction error while training any dataset. We can easily fetch the outliers from a dataset by using boxplot. The given dataset has a huge amount of outliers on the 'Age' column. The small round circles are the data that are over range. To see the outliers python has a built-in library named Seaborn by which we can fetch the outliers.

Detect Outliers

[274]

```
numeric_df = dataset.select_dtypes(include='number')
melted_df = numeric_df.melt(var_name='Variable', value_name='Value')

plt.figure(figsize=(20, 10))
sns.boxplot(x='Variable', y='Value', data=melted_df)
plt.xticks(rotation=45)
plt.title("Raw Outlier Data")
plt.tight_layout()
plt.show()
print(f"Total Rows: {dataset.shape[0]}")
print(f'Total Columns: {dataset.shape[-1]}')
```

Here, we need to select the data type and then variable and value as parameters in order to melt them. Feed the horizontal axis for variable and vertical axis for value representation.

Categorical Values Problem

This is the most common problem for the classification problem. We know that a machine learning algorithm is based on the derivative or complex mathematical calculation and predictions and as machine learning is based on some mathematical calculation to predict the output and mathematical formula only works for numerical data. Hence, the columns that contain the object data type can not participate or feed into any machine learning algorithm.

	Age	Gender	Height	Weight	CALC	FAVC	FCVC	NCP	SCC	SMOKE	CH2O	family_history_with_overweight	FAF	TUE	CAEC	MTRANS	NObeyesdad
48	20.000000	NaN	1.660000	60.000000	Sometimes	no	3.000000	3.000000	no	no	2.000000	yes	1.000000	0.000000	Sometimes	Walking	Normal_Weight
70	23.000000	Female	1.650000	80.000000	no	yes	2.000000	3.000000	no	no	2.000000	NaN	0.000000	2.000000	Always	Public_Transportation	Overweight_Level_II
71	22.000000	Female	1.670000	50.000000	Sometimes	no	3.000000	3.000000	yes	no	NaN	yes	2.000000	1.000000	no	Public_Transportation	Insufficient_Weight
125	18.000000	NaN	1.750000	80.000000	Frequently	yes	2.000000	3.000000	no	no	2.000000	yes	0.000000	0.000000	Always	Public_Transportation	Overweight_Level_I
143	34.000000	Female	1.680000	75.000000	Sometimes	yes	3.000000	1.000000	no	no	NaN	no	0.000000	0.000000	Sometimes	Automobile	Overweight_Level_I
157	21.000000	Male	1.670000	60.000000	Sometimes	yes	2.000000	3.000000	no	no	NaN	yes	2.000000	1.000000	Frequently	Public_Transportation	Normal_Weight
196	22.000000	Male	1.750000	74.000000	Sometimes	no	2.000000	3.000000	no	no	2.000000	NaN	1.000000	2.000000	Sometimes	Bike	Normal_Weight
205	23.000000	Female	1.600000	78.000000	Frequently	yes	2.000000	1.000000	no	yes	NaN	yes	1.000000	0.000000	Sometimes	Public_Transportation	Obesity_Type_I
263	23.000000	NaN	1.740000	53.000000	no	yes	2.000000	3.000000	no	no	1.000000	no	3.000000	2.000000	Frequently	Public_Transportation	Insufficient_Weight
274	25.000000	Female	1.610000	61.000000	Sometimes	no	2.000000	3.000000	no	no	NaN	no	2.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
281	18.000000	Male	1.700000	55.000000	Sometimes	yes	2.000000	3.000000	no	no	NaN	yes	0.000000	0.000000	Frequently	Public_Transportation	Normal_Weight
284	20.000000	Male	1.770000	70.000000	Sometimes	yes	1.000000	1.000000	no	no	NaN	yes	1.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight
314	26.000000	Female	1.650000	63.000000	no	yes	3.000000	3.000000	no	no	NaN	no	1.000000	0.000000	Sometimes	Walking	Normal_Weight
324	30.000000	Female	1.650000	71.000000	Sometimes	yes	2.000000	3.000000	no	no	1.000000	NaN	0.000000	0.000000	Sometimes	Public_Transportation	Overweight_Level_I
330	17.000000	Female	1.800000	50.000000	Sometimes	yes	3.000000	4.000000	no	no	NaN	no	2.000000	1.000000	Frequently	Public_Transportation	Insufficient_Weight
420	18.000000	Male	1.850000	60.000000	Sometimes	yes	3.000000	4.000000	yes	no	2.000000	NaN	2.000000	0.000000	Sometimes	Automobile	Insufficient_Weight
463	17.000000	Male	1.800000	68.000000	Sometimes	no	2.000000	3.000000	no	no	NaN	yes	2.000000	1.000000	Sometimes	Public_Transportation	Normal_Weight

Feature Scaling Mismatch

At the very nascent stage the data is not always scaled for both classification and regression based dataset. This is happening because of the various types of factors and parameters. These parameters hold some data that can be represented within 1 digit, 2 digits, sometimes fractional.

	Age	Gender	Height
48	20.000000	NaN	1.660000
70	23.000000	Female	1.650000
71	22.000000	Female	1.670000
125	18.000000	NaN	1.750000
143	34.000000	Female	1.680000
157	21.000000	Male	1.670000
196	22.000000	Male	1.750000
205	23.000000	Female	1.600000
263	23.000000	NaN	1.740000
274	25.000000	Female	1.610000
281	18.000000	Male	1.700000
284	20.000000	Male	1.770000
314	26.000000	Female	1.650000
324	30.000000	Female	1.650000
330	17.000000	Female	1.800000
420	18.000000	Male	1.850000
463	17.000000	Male	1.800000

Here, for Age = 20, Height = 1.66. There is a huge gap between two values. Similarly, for Age = 34, Height is 1.68 and so on. Again, for Weight = 60, FCVC = 3.0.

Algorithms like *Decision Tree* have the ability to overcome the scaling issue but apart from the *Decision Tree*

Height	Weight	CALC	FAVC	FCVC	NCP
1.660000	60.000000	Sometimes	no	3.000000	3.000000
1.650000	80.000000	no	yes	2.000000	3.000000
1.670000	50.000000	Sometimes	no	3.000000	3.000000
1.750000	80.000000	Frequently	yes	2.000000	3.000000
1.680000	75.000000	Sometimes	yes	3.000000	1.000000
1.670000	60.000000	Sometimes	yes	2.000000	3.000000
1.750000	74.000000	Sometimes	no	2.000000	3.000000
1.600000	78.000000	Frequently	yes	2.000000	1.000000
1.740000	53.000000	no	yes	2.000000	3.000000
1.610000	61.000000	Sometimes	no	2.000000	3.000000
1.700000	55.000000	Sometimes	yes	2.000000	3.000000
1.770000	70.000000	Sometimes	yes	1.000000	1.000000
1.650000	63.000000	no	yes	3.000000	3.000000
1.650000	71.000000	Sometimes	yes	2.000000	3.000000
1.800000	50.000000	Sometimes	yes	3.000000	4.000000
1.850000	60.000000	Sometimes	yes	3.000000	4.000000
1.800000	68.000000	Sometimes	no	2.000000	3.000000

algorithms like linear regression and logistic regression wont show the correct accuracy for the mismatch of the data scale. This is a very common problem in a real life dataset.

Duplicate Rows Problem

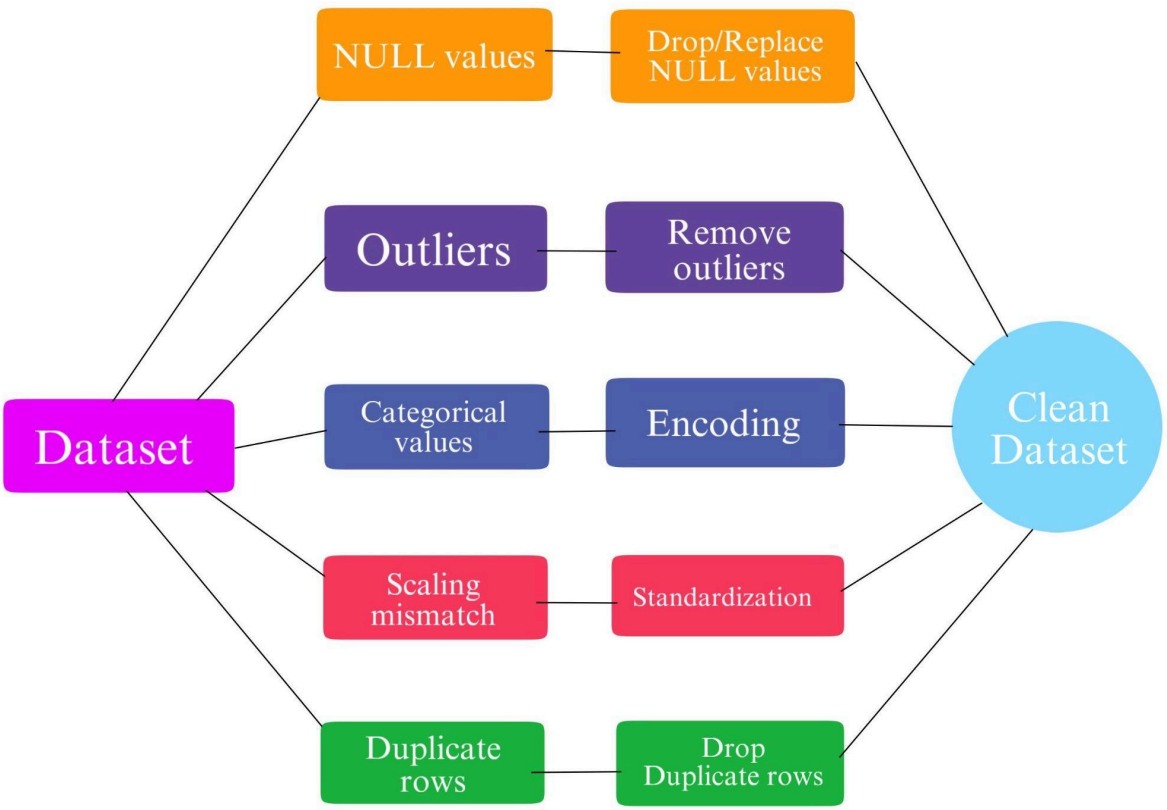
Duplicate value in the dataset can cause extra runtime to analyze the dataset. Moreover, Duplicates can skew statistics, aggregations, and machine learning models. The duplicated data can simultaneously negatively affect the computations if the number of duplicated rows are huge as well as the memory storage.

```
[ ]  1 dataset[dataset.duplicated(keep=False)]
```

	family_history_with_overweight_yes	SMOKE_yes	SCC_yes	FAVC_yes	Gender_Female	Age	Height	Weight	MTRANS	CAEC	CALC	FCVC	NCP	CH2O	FAF	TUE	NObeyesdad
208	1.0	0.0	0.0	1.0	1.0	22.0	1.69	65.0	0.0	1.0	1.0	2.0	3.0	2.0	1.0	1.0	0.0
209	1.0	0.0	0.0	1.0	1.0	22.0	1.69	65.0	0.0	1.0	1.0	2.0	3.0	2.0	1.0	1.0	0.0
282	1.0	0.0	0.0	1.0	1.0	18.0	1.62	55.0	0.0	2.0	0.0	2.0	3.0	1.0	1.0	1.0	0.0
443	1.0	0.0	0.0	1.0	0.0	18.0	1.72	53.0	0.0	1.0	1.0	2.0	3.0	2.0	0.0	2.0	4.0
460	1.0	0.0	0.0	1.0	1.0	18.0	1.62	55.0	0.0	2.0	0.0	2.0	3.0	1.0	1.0	1.0	0.0
466	1.0	0.0	0.0	1.0	0.0	22.0	1.74	75.0	2.0	2.0	0.0	3.0	3.0	1.0	1.0	0.0	0.0
467	1.0	0.0	0.0	1.0	0.0	22.0	1.74	75.0	2.0	2.0	0.0	3.0	3.0	1.0	1.0	0.0	0.0
496	1.0	0.0	0.0	1.0	0.0	18.0	1.72	53.0	0.0	1.0	1.0	2.0	3.0	2.0	0.0	2.0	4.0

To see the duplicated or repeated rows we can use .duplicated() with the parameter keep=False. keep=False ensures that the first and the last occurrence of the duplicated data is included in the output result.

These are some issues an individual might face while cleaning the data. Now there are some techniques to solve the aforementioned problems. We can remove outliers, drop duplicates and replace NULL values with the mean or highest frequency of a column data



NULL Values Handling

We can handle the NULL value by two types. First we need to see the total number of NULL values for each column. If a single column contains a vast number of NULL values, in that case we can easily drop the column without having to apply any kind of algorithmic operations. But if the column has a few NULL values. In that case we don't need to drop the column. It might be risky to drop because the column may or may not that highly affect the output. Instead of dropping the column deo the dataset we can fill the NULL data with some data. For numerical columns we can replace the NULL value with the mean of that column and for categorical columns, we can fill the NULL value with the highest frequency data of that column. To replace the NULL value from the dataset we use a built-in Class form python known as "SimpleImputer" for numeric data type. Again, for categorical features, there can be more than one same frequency count for different values. In that case we will use only the first appearance of the frequency list.

Replace The NULL Values With The Highest Frequency Data For Object Datatype

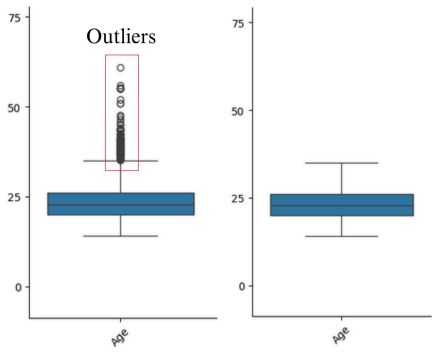
```
[ ]  1 for i in dataset.select_dtypes(include="object").columns:
     2     dataset[i].fillna(dataset[i].mode()[0], inplace=True)
```

Replace The NULL Values With The Mean Value For Numeric Datatype

```
[ ]  1 simpleimputer = SimpleImputer(strategy="mean")
     2 dataset[numerical_column] = simpleimputer.fit_transform(dataset[numerical_column])
     3 dataset.to_csv("dataset.csv", index=False)
```

Outliers Remove

To remove the outliers from the dataset, we use a method called "Interquartile Range" or shortly IQR. For this, we split the whole data into two ratios and assigned them into Q1 and Q3 variables. Q1 holds 25% of the entire data and Q3 holds




```
outlierColumn = []
for i in dataset.select_dtypes(include="number").columns:
    outlierColumn.append(i)
for i in outlierColumn:
    q1 = dataset[i].quantile(0.25)
    q3 = dataset[i].quantile(0.75)
    IQR = q3-q1
    minRange = q1-(1.5*IQR)
    maxRange = q3+(1.5*IQR)
    dataset = dataset[dataset[i]<=maxRange]
    dataset = dataset[dataset[i]>=minRange]
numeric_df = dataset.select_dtypes(include='number')
melted_df = numeric_df.melt(var_name='Variable', value_name='Value')
plt.figure(figsize=(20, 10))
sns.boxplot(x='Variable', y='Value', data=melted_df)
plt.xticks(rotation=45)
plt.title("After Removing Outlier")
plt.tight_layout()
plt.show()
print(f"Total Rows: {dataset.shape[0]}")
print(f'Total Columns: {dataset.shape[-1]}')
```

75% of the entire data. The subtraction of Q1 and Q3 is called the Interquartile Range. Now to remove outliers we use (Q1-1.5 x IQR) for minimum range data and also use (Q3+1.5 x IQR) for maximum range of the data. And then we simply drop those values from our main dataset.

Encoding

There are three types of encoding. Based on the problem structure three one of them is used.

Encoding Types	When to Used
OneHot Encoding	At most 2 types of categorical data
Label Encoding	More than 2 types of categorical data
Ordinal Encoding	More than 2 types of categorical data but encoding has an order of sequences

For this project we used OneHot Encoding and Ordinal Encoding. This encoding can convert any categorical data into numerical data that can be further used in any machine learning algorithm to predict the outcome. We use OnHot Encoding on:

Gender	FAVC	SCC	SMOKE
family_history_with_overweight			

Applying OneHot Encoding

```
encoder = OneHotEncoder(sparse_output=False)

Gender_encoded = encoder.fit_transform(dataset[['Gender']])
Gender_encoded_df = pd.DataFrame(Gender_encoded, columns=encoder.get_feature_names_out(['Gender']))
Gender_encoded_df.index = dataset.index
dataset = dataset.drop('Gender', axis=1)
dataset = pd.concat([Gender_encoded_df, dataset], axis=1)
dataset = dataset.drop('Gender_Male', axis=1)
```

We use Ordinal Encoding on:

CALC	MTRANS	CAEC	NObeyesdad

CALC, MTRANS, NObeyesdad has more than two unique data and hence the OneHot encoding can not be applied here. For this we have to initially pass a list of sequences of

data and run ordinal encoding, the backend algorithms convert them into numeric values according to their index position.

Applying Ordinal Encoding

```
calcDATA = [['no', 'Sometimes', 'Frequently', 'Always']]
ordinalencoder = OrdinalEncoder(categories=calcDATA)
result = ordinalencoder.fit_transform(dataset[["CALC"]])
dataset = dataset.drop('CALC', axis=1)
dataset.insert(loc=8, column="CALC", value=result)
```

Standardization

In real life the data will not always be in equal range. For example in our dataset the Age column has higher value than Height & weight column. So, when we plot the data or feed the dataset into the machine learning model the model gets biased for the large value and the small values are automatically ignored. This might create a low accuracy issue in the output. To solve this issue we need to use feature scaling to the dataset in order to make all values in an equal range.

Standardization

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns = X_test.columns)
```

We can only use feature scaling in the X_train. Because if we use standardization into the y_train and y_test, it creates a data leakage. To prevent data leakage, we use this method. Applying feature scaling into the whole dataset can increase the accuracy of the model but simultaneously it predicts a wrong accuracy in most of the cases.

Drop Duplicated Rows

Duplicated rows are actually those rows which have similar types of data with no variation. This type of row must be dropped from the dataset in order to improve run time complexity as well as space complexity. In our dataset there are in total four pairs of duplicated columns(in total 8 columns) which need to be removed from the dataset using *.drop_duplicates()*.

```
dataset[dataset.duplicated(keep=False)]
```

	family_history_with_overweight_yes	SMOKE_yes	SCC_yes	FAVC_yes	Gender_Female	Age	Height	Weight	MTRANS	CAEC	CALC	FCVC	NCP	CH2O	FAF	TUE	NObeyesdad
208	1.0	0.0	0.0	1.0	1.0	22.0	1.69	65.0	0.0	1.0	1.0	2.0	3.0	2.0	1.0	1.0	0.0
209	1.0	0.0	0.0	1.0	1.0	22.0	1.69	65.0	0.0	1.0	1.0	2.0	3.0	2.0	1.0	1.0	0.0
282	1.0	0.0	0.0	1.0	1.0	18.0	1.62	55.0	0.0	2.0	0.0	2.0	3.0	1.0	1.0	1.0	0.0
443	1.0	0.0	0.0	1.0	0.0	18.0	1.72	53.0	0.0	1.0	1.0	2.0	3.0	2.0	0.0	2.0	4.0
460	1.0	0.0	0.0	1.0	1.0	18.0	1.62	55.0	0.0	2.0	0.0	2.0	3.0	1.0	1.0	1.0	0.0
466	1.0	0.0	0.0	1.0	0.0	22.0	1.74	75.0	2.0	2.0	0.0	3.0	3.0	1.0	1.0	0.0	0.0
467	1.0	0.0	0.0	1.0	0.0	22.0	1.74	75.0	2.0	2.0	0.0	3.0	3.0	1.0	1.0	0.0	0.0
496	1.0	0.0	0.0	1.0	0.0	18.0	1.72	53.0	0.0	1.0	1.0	2.0	3.0	2.0	0.0	2.0	4.0

```
dataset = dataset.drop_duplicates()
dataset.to_csv('dataset.csv', index=False)
print(f"Total Rows: {dataset.shape[0]}")
print(f'Total Columns: {dataset.shape[-1]}')
```

Here, index=False demonstrates that in the new dataset we don't want to include the index into the csv data frame.

Dataset Splitting

Train Test Split and Ratio

Before sending the dataset into any machine learning model, first we need to split the dataset into two halves. For this dataset we split the dataset into 70% and 30%. For the training phase we use 70% of the entire dataset and for the testing phase we use 30% of the entire dataset.

Dataset in machine learning is generally divided into two main parts:

Features (X) – input data, based on which the model will make decisions.

Labels (y) – output/target data, i.e. the desired results that the model learns.

These x and y are then split again into two parts:

Train (training set): The data used for learning.

Test (testing set): data used to verify after learning is complete.

Train Test Split with Train set (70%) | Test set (30%)

```
input = dataset.drop("NObeyesdad", axis=1)
output = dataset["NObeyesdad"]
X_train, X_test, y_train, y_test  = train_test_split(input, output, test_size=0.3, random_state=42)
```

Data Shape

```
X_train.shape, X_test.shape
```

```
((975, 16), (419, 16))
```

We divided the dataset into two segments. For input we drop the "*NObeyesdad*" column from the dataset and for the output we drop all the columns except

the "NO*beyesdad*" column. We set the test size 0.3 means 30% of the data will be used for test cases. For randomness of the splitting data we use random_state as a parameter. The size of the number (high or low) does not affect randomness quality or model performance.

Goal and Achievement Behind the Data Splitting	
To avoid bias	If we train and test the model with the same data, the model can easily memorize, not learn.
To check generalization	X_test and y_test are used to know how the model performs on new unknown data
For model tuning	trained with train and tuned and performance measured with tests.

Model Training & Testing

Random Forest Classifier:

Decision Trees are an important tool in the world of machine learning. However, a single decision tree often suffers from the problem of overfitting. The solution to this is Random Forest — a powerful ensemble technique that works in combination with many decision trees. Random Forest is used for both classification and regression and is capable of handling large and complex datasets with high efficiency.

Random Forest is an example of Ensemble Learning. It uses many decision trees together to make the results reliable and stable. "A forest is stronger than a single tree." It basically works based on a technique called Bagging.

A Random Forest basically follows the following steps:

Bootstrapping

Random sampling with replacement is done repeatedly from the dataset. A Decision Tree is trained from each sample dataset.

Random Feature Selection

While building each tree, some features are randomly selected in each split, not the entire feature set. This helps in bringing diversification among the trees.

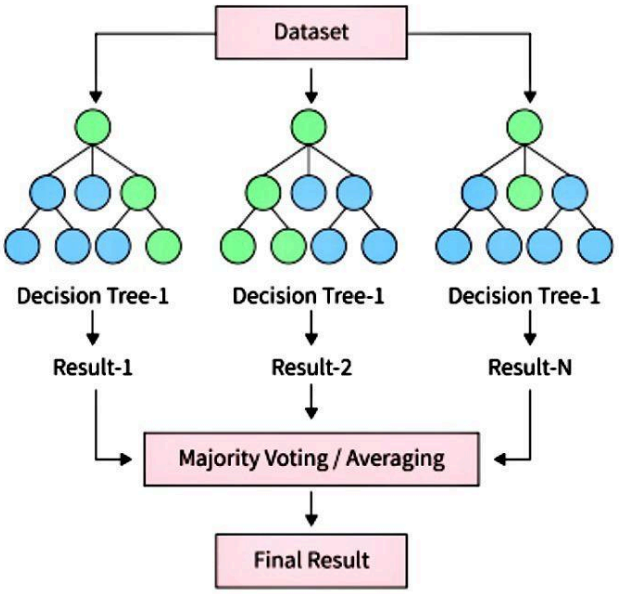
Training Multiple Trees

Multiple Decision Trees are built from different Random Subsets.

Aggregation (Voting/Averaging)

Each Decision Tree predicts a class, and the final decision is made by majority voting.

Each Tree predicts a value, and the average of them is the final prediction. A Decision Tree usually memorizes the training data very well (Overfit). Random Forest eliminates the problem by creating different trees with different features and data. Due to the ensemble method, it gives good results in most problems. If there are complex relationships in the data, Random Forest can capture them very well. Random Forest can determine how important each feature is.



Prediction Calculation:

For classification problem $\varphi = arg\ MAX \sum_{i=1}^{n} \{(T1(x), T2(x), ..., Tn(x)\}$

$\varphi = Predicted\ Class\ Label$

Ti(x) = Prediction form the ith decision tree

Arg MAX = most frequency output

For regression problem $\varphi = \frac{1}{n} \sum_{i=1}^{n} Ti(x)$

$\varphi = Final\ Predicted\ value$ & n = number of trees

Random Forest accuracy: 91.41%
Precision Score (macro): 89.87
Recall Score (macro): 89.43
F1 Score (macro): 89.49
Correct Predictions: 383
Wrong Predictions: 36

Neural Network

In today's era of Artificial Intelligence and Machine Learning, "Neural Network" is a technology inspired by the human brain. It is a method that learns from data, makes predictions, and makes decisions. Neural Networks are the basis of Deep Learning, and are currently being widely used in self-driving cars, facial recognition, language translation, healthcare, and even artificial chatbots.

A neural network (Artificial Neural Network or ANN) is a type of computational model that works like the functioning of human neurons. It is composed of multiple layers of nodes (or neurons), where each node calculates something on input data, and outputs it.

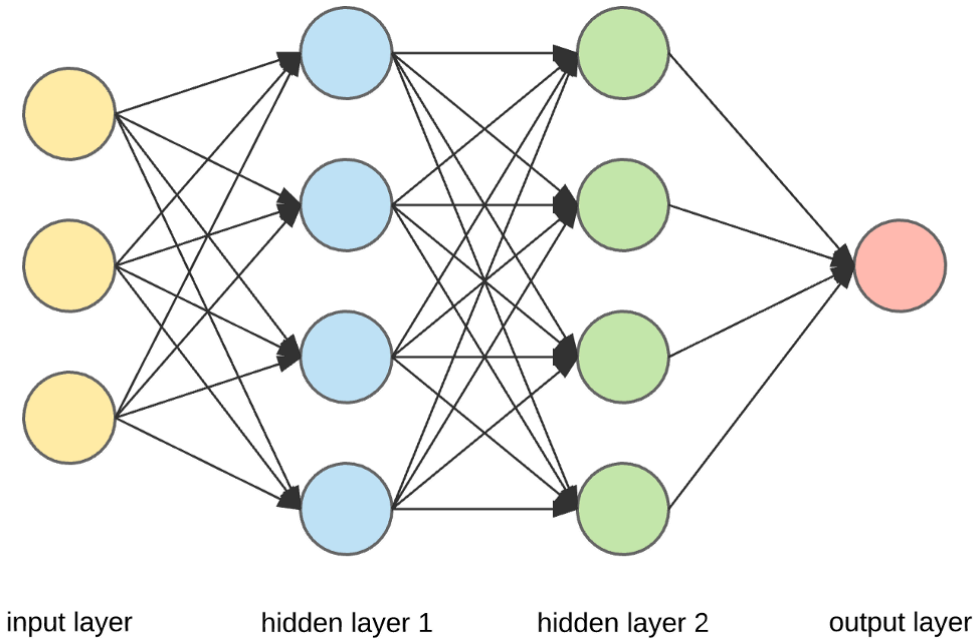
Three layers of structure:

Input Layer: Data enters here.

Hidden Layer(s): Information is processed here.

Output Layer: The final result is obtained here.

Each neuron is connected to another neuron, and each connection has a weight.



Working Architecture:

Step 1: Take Input Data

The input layer takes various features/data.

Step 2: Apply Weight and Bias

Each connection has a weight that determines how much it will affect the input. A bias is added to each neuron.

Step 3: Activation Function

The output of the neuron is extracted through a function. For example: Sigmoid, ReLU (Rectified Linear Unit), tanh

Step 4: Forward Propagation

Information flows from input to output. At each level, the output is obtained by calculating the input * weight + bias and using the activation function.

Step 5: Loss Function

If the prediction is wrong, the amount of error is measured (e.g. Mean Squared Error).

Step 6: Backpropagation

If the model is wrong, it finds out where the error occurred and updates the weights.

Step 7: Gradient Descent

Weights are used for updating, so that the error gradually decreases.

Backpropagation:

An algorithm that updates weights layer by layer based on errors in the output.

Neural Network Accuracy: 86.40%
Precision Score: 87.41%
Recall Score: 86.40%
F1 Score: 86.49%
Correct Predictions: 362
Wrong Predictions: 57

Decision tree

In the field of supervised machine learning, a Decision Tree is one of the most interpretable and widely used algorithms. It mimics human decision-making logic through a tree-like structure of conditions, making it ideal for classification and regression tasks. Each internal node represents a decision based on a feature, branches represent the outcomes of these decisions, and the leaves represent the final output or class label.

Structure of a Decision Tree:

Root Node: Starts the decision process with the most informative feature.
Internal Nodes: Feature-based decision splits that divide data further.
Leaf Nodes: Final predictions or class outcomes.
Branches: Represent the decision rules from root to leaves.

Working Architecture:

Step 1: Data Splitting

The algorithm begins at the root node and splits the dataset using feature values.

Step 2: Impurity Measure

Entropy is used to measure the randomness or impurity of the dataset at a node. A node with low entropy has more pure classes, while a node with high entropy contains a mixture of classes.

$$\text{Entropy}(S) = \sum_{i=1}^{n} -P_i \log_2 P_i$$

Here, 'P' represents the probability of class i in subset S

Step 3: Selecting the best feature

Information Gain tells us how important a given attribute of the feature vectors is. The algorithm chooses the feature that gives the highest information gain, ensuring the most informative split.

$$\text{Information Gain}(S,A) = \text{Entropy}(S) - \sum[P(S|A).\text{Entropy}(S|A)]$$

Step 4: Recursive Splitting

The process continues recursively, building deeper levels of the tree until stopping conditions are met (e.g., max depth or pure leaves).

Step 5: Prediction

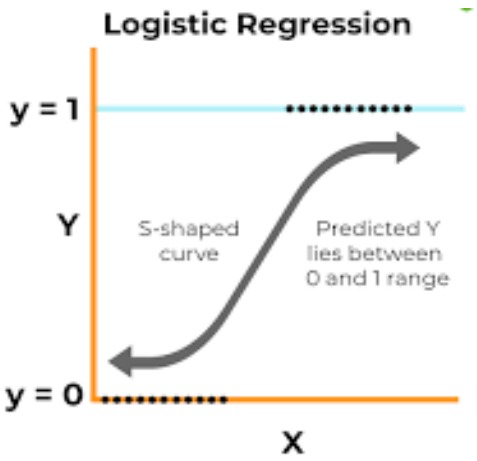
For a new input, the tree is traversed based on the conditions until a leaf node is reached for the prediction.

Decision Trees are based on threshold splits rather than distance calculations, so they are not affected by feature scaling. The model treats all features independently, based on their ability to reduce entropy, not their numerical scale.

Decision Tree Accuracy Score: 85.20%
Precision Score: 85.37%
Recall Score: 85.2%
F1 Score: 85.16
Correct Predictions: 357
Wrong Predictions: 62

Logistic Regression

Logistic Regression is a foundational machine learning algorithm used primarily for classification tasks. It is a probabilistic classification model that predicts the likelihood of class membership. It is widely applied in areas like disease prediction, fraud detection, credit scoring, and marketing analysis due to its simplicity and interpretability.



Working architecture:

Step 1: Input Features

The model starts by taking in multiple input features (like age, height, meal frequency, etc.) for each individual.

Step 2: Weight the inputs

Each input feature is multiplied by a certain weight, which reflects how important that feature is in predicting the outcome. A small extra value (called bias) is also added.

Step 3: Combine the Inputs

All the weighted inputs are added together to form a single combined value. This value represents the model's raw prediction before converting it into a decision.

Step 4: Pass Through a Sigmoid Function

This combined value is passed through a special function that converts it into a probability between 0 and 1 — representing how likely it is that the input belongs to a certain class.

Step 5: Make a Prediction

Based on the probability, the model makes a final prediction — for example, whether a person is obese, overweight, or normal. It assigns the class with the highest probability.

Step 6: Compare with Actual Outcome

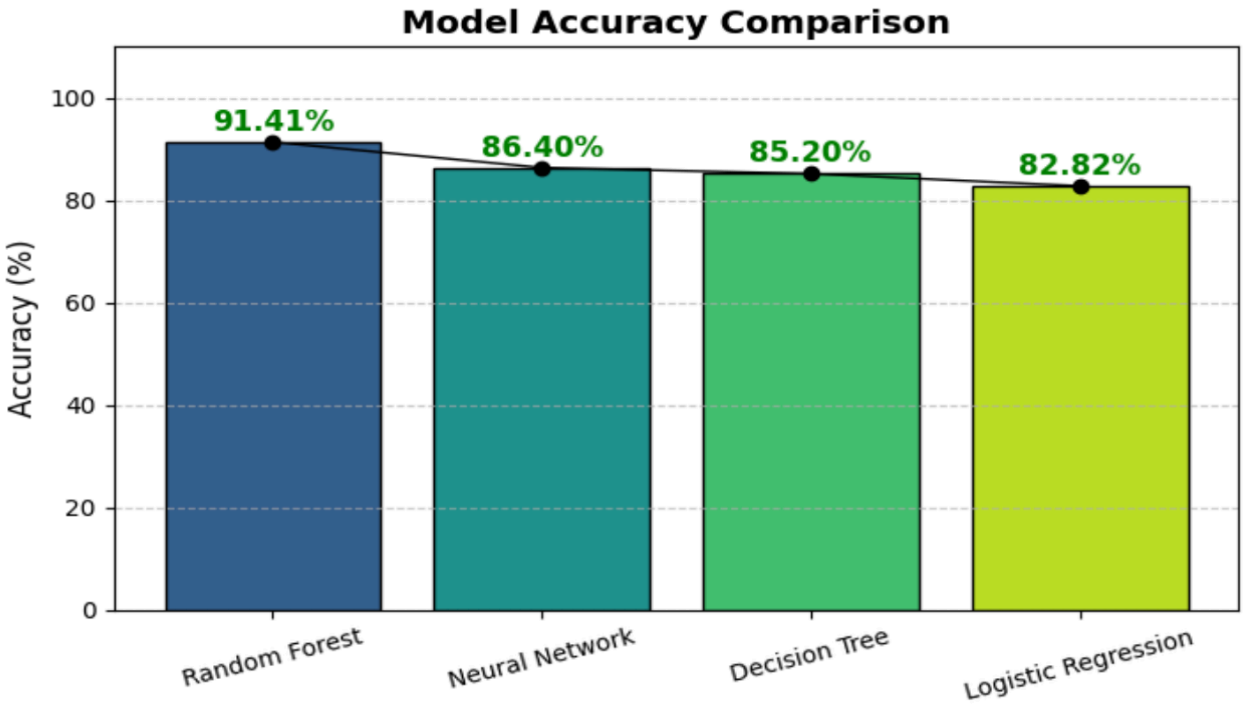
The model checks how accurate the prediction is by comparing it to the real result. If it's wrong, it calculates how far off it was.

Step 7: Learn and Improve

Using the difference between its prediction and the real answer, the model adjusts its weights so it performs better next time. This process repeats many times to improve accuracy.

Logistic Regression Accuracy score: 82.82%
Precision score: 80.22%
Recall score: 79.67%
F1 score: 79.24%
Correct predictions: 347
Wrong predictions: 72

Model Selection & Comparison Analysis



Bar Chart Showcasing Prediction Accuracy

This graph is a Model Accuracy Comparison bar chart that shows the performance or accuracy rate (%) of different machine learning models. A total of four models are compared here. The performance of each model is expressed in terms of Accuracy (%), that is, the percentage of accuracy a model is working or predicting. This graph shows how much a model is working or predicting correctly.

Random Forest Classifier
91.41% of accuracy

This model showed the highest accuracy on the graph, meaning it performed better than all other models.

Random Forest is an ensemble learning method, where many decision trees work together to reach a final decision. It reduces overfitting and generally provides high accuracy.

Widely used in classification and regression problems.

Neural Network

86.40% of accuracy

It is in second place. Neural networks are generally very good at understanding complex data relationships, especially when the data contains many variables.

Neural networks work like neurons in the human brain and process data step by step to make decisions.

Sometimes neural networks can give good results, but they take more time and energy to train.

Decision Tree

85.20% of accuracy

In third place is the Decision Tree. It is a straightforward model that makes decisions through questions and answers.

Works in a tree-structure, where each node represents a question and reaches a leaf node to reach a final decision.

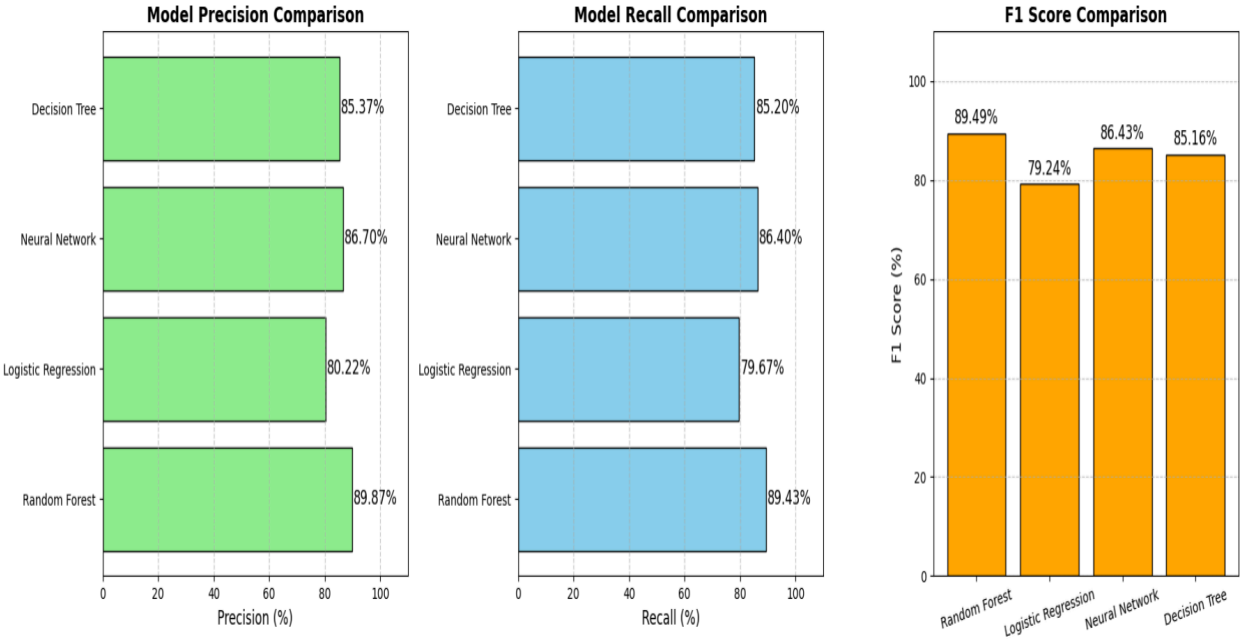
It is easy to understand and explain, but tends to overfit when used alone.

Logistic Regression

82.82% of accuracy

In fourth place is Logistic Regression. It is usually used in binary classification, such as yes/no, true/false.
It is a statistical model, where decisions are made based on probability.
Logistic Regression is relatively simple, fast and easy to interpret.

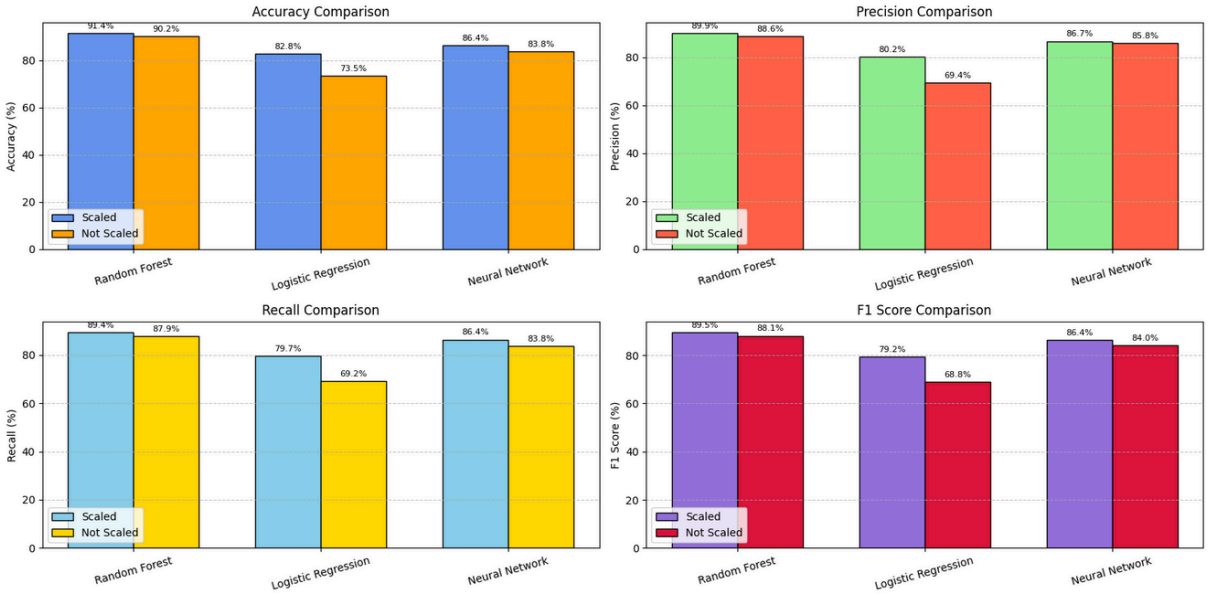
Precision, Recall & F1 Score Comparison



Precision: How many of the results that the model identified as “positive” were actually positive.

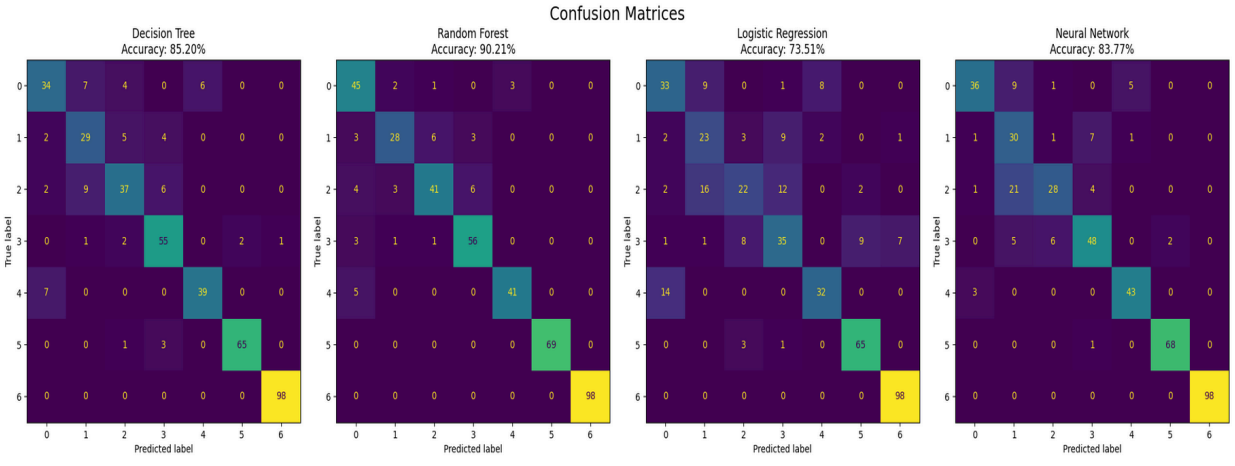
Recall: How many of the true positives the model correctly identified.

F1 Score: The average of Precision and Recall, which balances the two values. It uses the harmonic mean.



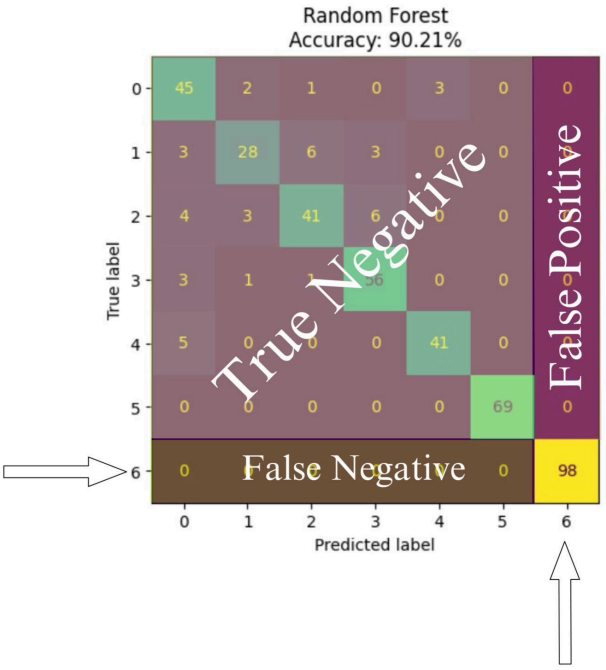
	Precision	Recall	F1 Score
Decision Tree	85.37%	85.20%	89.49%
Neural Network	86.70%	86.40%	79.24%
Logistic Regression	80.22%	79.67%	86.43%
Random Forest	89.87%	89.43%	85.16%

Confusion Matrix



For example: In case of Random Forest Classifier
feature 6:

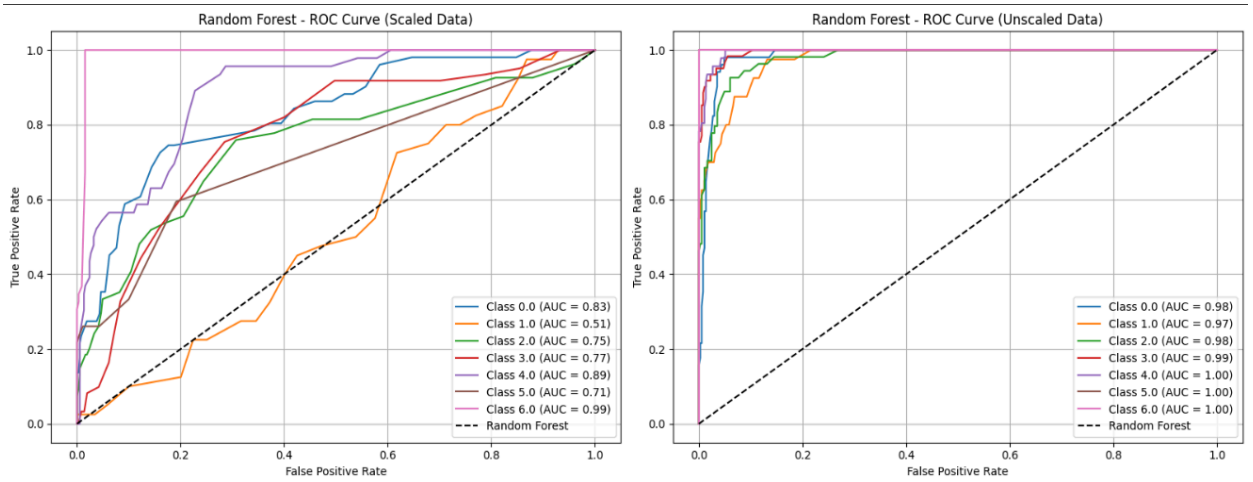
True Positive: 98
False Negative: 0
False Positive: 0
True Negative: 319
So, Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
Accuracy = $\frac{98+319}{98+319+0+0}$ = 100%
So, for feature 6, the accuracy is 100%

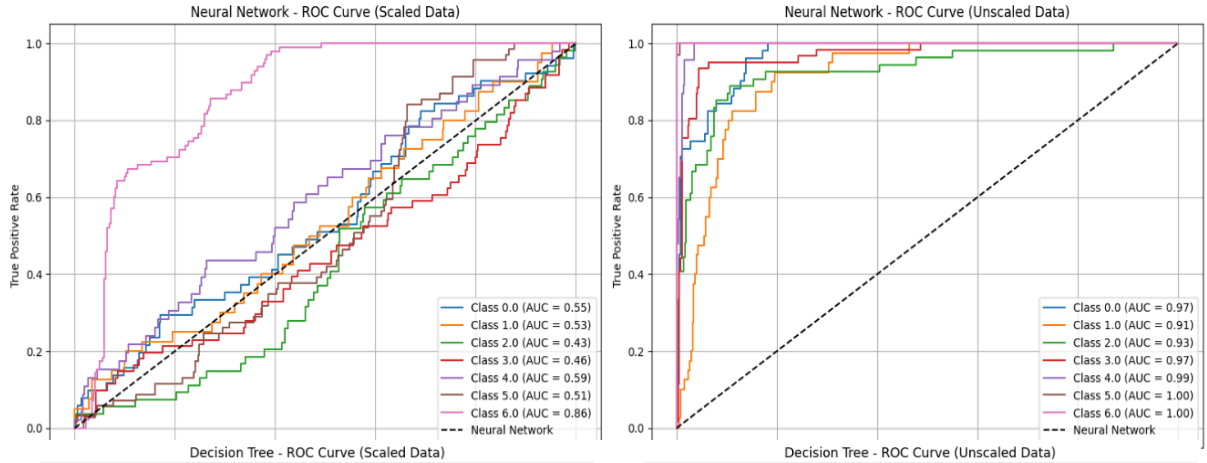


AUC Score, ROC Curve

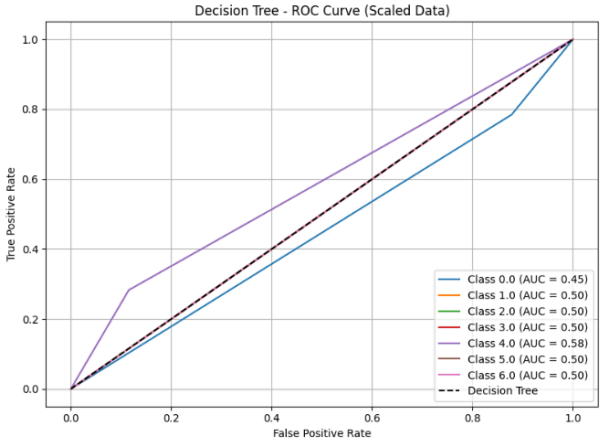
AUC (Area Under the Curve), tells us how much the model is capable of distinguishing between classes.
AUC = 1 : Perfect classifier
AUC = 0.5 : No discrimination (random guessing)

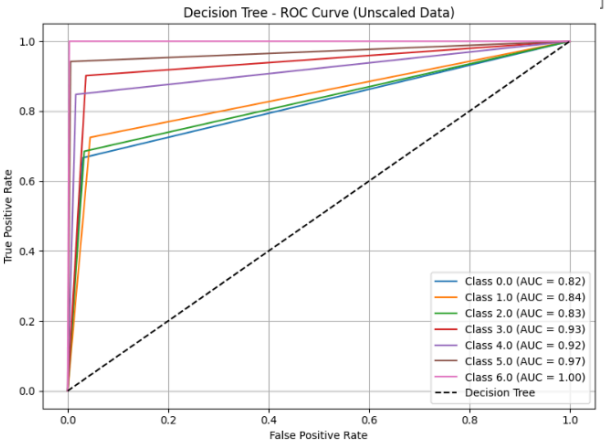
& **ROC Curve** is a plot of True Positive Rate (TPR) vs False Positive Rate (FPR) at various threshold settings.
TPR (Recall) = $\frac{TP}{TP+FN}$
FPR = $\frac{FP}{FP+TN}$

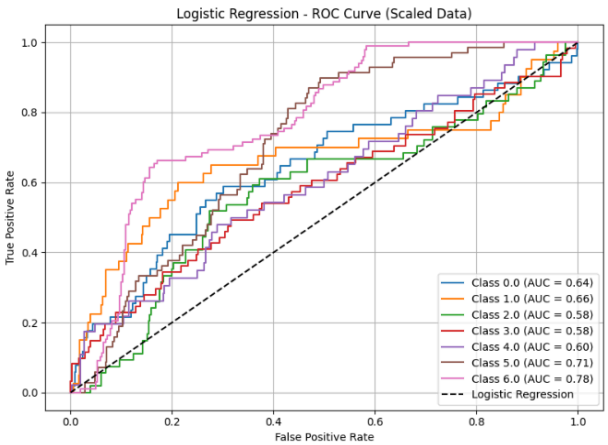


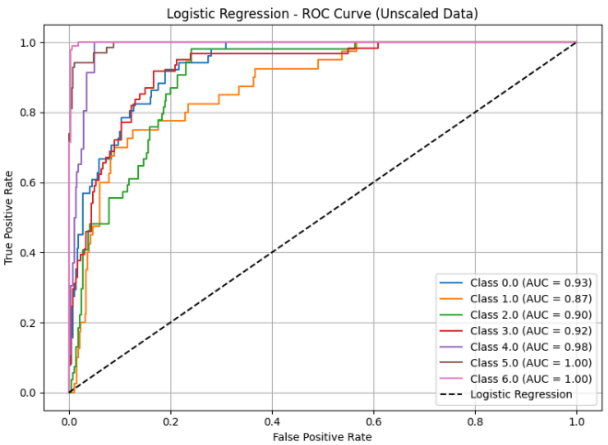

Neural Network - ROC Curve (Scaled Data)
True Positive Rate
Class 0.0 (AUC = 0.55)
Class 1.0 (AUC = 0.53)
Class 2.0 (AUC = 0.43)
Class 3.0 (AUC = 0.46)
Class 4.0 (AUC = 0.59)
Class 5.0 (AUC = 0.51)
Class 6.0 (AUC = 0.86)
Neural Network

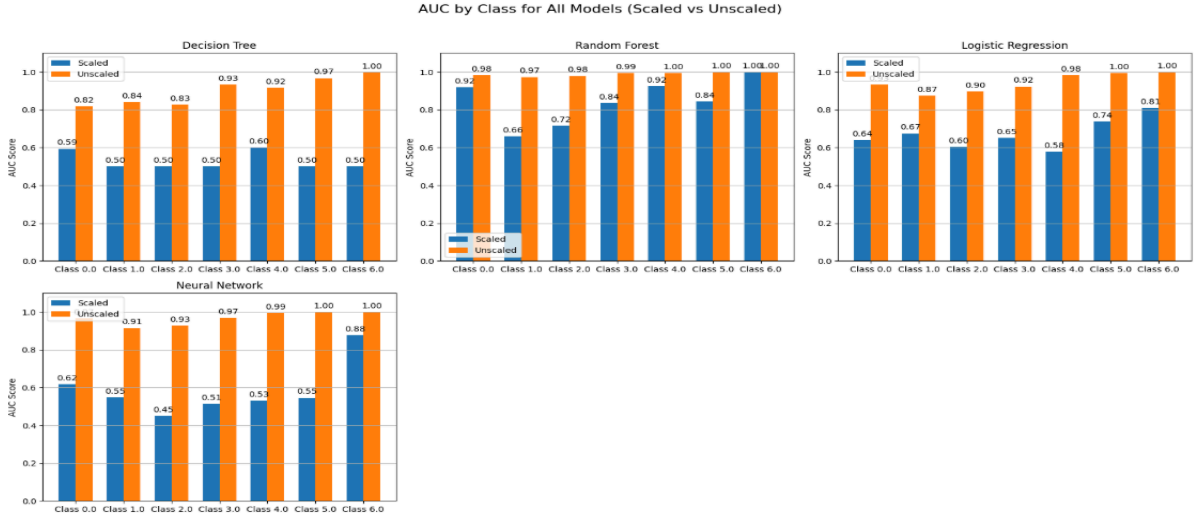
Neural Network - ROC Curve (Unscaled Data)
True Positive Rate
Class 0.0 (AUC = 0.97)
Class 1.0 (AUC = 0.91)
Class 2.0 (AUC = 0.93)
Class 3.0 (AUC = 0.97)
Class 4.0 (AUC = 0.99)
Class 5.0 (AUC = 1.00)
Class 6.0 (AUC = 1.00)
Neural Network


Decision Tree - ROC Curve (Scaled Data)
True Positive Rate
False Positive Rate
Class 0.0 (AUC = 0.45)
Class 1.0 (AUC = 0.50)
Class 2.0 (AUC = 0.50)
Class 3.0 (AUC = 0.50)
Class 4.0 (AUC = 0.58)
Class 5.0 (AUC = 0.50)
Class 6.0 (AUC = 0.50)
Decision Tree


Decision Tree - ROC Curve (Unscaled Data)
True Positive Rate
False Positive Rate
Class 0.0 (AUC = 0.82)
Class 1.0 (AUC = 0.84)
Class 2.0 (AUC = 0.83)
Class 3.0 (AUC = 0.93)
Class 4.0 (AUC = 0.92)
Class 5.0 (AUC = 0.97)
Class 6.0 (AUC = 1.00)
Decision Tree


Logistic Regression - ROC Curve (Scaled Data)
True Positive Rate
False Positive Rate
Class 0.0 (AUC = 0.64)
Class 1.0 (AUC = 0.66)
Class 2.0 (AUC = 0.58)
Class 3.0 (AUC = 0.58)
Class 4.0 (AUC = 0.60)
Class 5.0 (AUC = 0.71)
Class 6.0 (AUC = 0.78)
Logistic Regression


Logistic Regression - ROC Curve (Unscaled Data)
True Positive Rate
False Positive Rate
Class 0.0 (AUC = 0.93)
Class 1.0 (AUC = 0.87)
Class 2.0 (AUC = 0.90)
Class 3.0 (AUC = 0.92)
Class 4.0 (AUC = 0.98)
Class 5.0 (AUC = 1.00)
Class 6.0 (AUC = 1.00)
Logistic Regression


AUC by Class for All Models (Scaled vs Unscaled)
Decision Tree
Random Forest
Logistic Regression
Neural Network
Scaled
Unscaled
AUC Score
Class 0.0
Class 1.0
Class 2.0
Class 3.0
Class 4.0
Class 5.0
Class 6.0

Conclusion

For random forest classification we have seen that the accuracy is 91.41% approximately along with the precision rate of 89.87%. The recall rate is 89.49% with an F1 score of 89.49%. This model can successfully predict 383 predictions and 36 wrong predictions. Analyzing the aforementioned data we can make a conclusion that, This model performed the best among the four. In addition to Accuracy, Precision, Recall, and F1-score are each between 89%–91%, indicating that the model is not only able to classify correctly, but also performs well in almost all classes. However, random forest is an advanced version of decision tree which is the accumulation of miscellaneous decision trees. This technique helps the model to prevent bias and risk of overfitting.

Furthermore, neural network performance is good, but slightly lower than that of random forest classification models. This model has an accuracy of 86.40% with precision rate of 87.41%, the recall and F1 score is 86.40% and 86.49% respectively. This model successfully predicted 362 predictions with 57 wrong predictions which make the model stand at 2nd position of our dataset. The Neural Network model is based on deep learning. It is capable of learning many complex patterns, but its training process is relatively more data-dependent. If there is not enough data or the pre-processing is not done properly, the results may be reduced.

Moreover, in terms of decision tree, the accuracy percentage and all the other parameters are not as good as the random forest and neural network. But has a decent accuracy (85.20%). The other factors such as precision, recall, F1 score has 85.37%, 85.20% and 85.16% respectively. This model is able to predict 357 correct predictions along with 62 wrong predictions. Decision Tree is a basic and straightforward model. Although the accuracy is good, it is not as generalized as Random Forest or Neural Network. As a result, it can be prone to overfitting.

Finally, logistic regression is the worst model for this dataset. Its performance is too low to predict the actual result. The accuracy, precision, recall and F1 score for thai model is consequently 82.82%, 80.22%, 79.67%, 79.24%. Out of 419 predictions, this model can successfully predict 347 predictions and 72 wrong predictions. Logistic Regression is a linear model, which is simple and interpretable, but unable to capture complex or non-linear relationships. Due to this, its performance is relatively low. If there is a non-linear relationship between the features and the target variable in the data, Logistic Regression will not be able to capture it. In addition, if there are many categorical or high-dimensional features, the effectiveness of this model decreases. And this limitation makes this model stand at the last priority model for our dataset.

From the very beginning to the end, we both have faced miscellaneous problems while data pre-processing, feature scaling, accuracy enhancement and model train are various threshold values.

First we removed one portion of the categorical data as outliers after encoding. After all, we have fixed that problem later. While doing feature scaling, I was applying the standard scaler on the categorical data which has been encoded. But that was a wrong move and decreased my accuracy. However, we have solved the issue.