AARSH MEHTANI

PROJECT-3

Obesity-Prediction Data-Science

PROBLEM DESCRIPTION:

The problem statement given to us is as follows : " Applying ML Classification algorithms on the data set and getting inferences from the data using appropriate ML algorithms. " We collected the dataset Estimation of obesity levels based on eating habits and physical condition Data Set from the [UCL Machine Learning Repository]Official publication of the research that provides data can be accessed from here.

(https://archive.ics.uci.edu/ml/index.php).

AIM:

We will predict levels of obesity using various types of machine learning classification models to our dataset to make relevant predictions on which factors influence obesity.

DATA SET INFORMATION:

This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru, and Colombia, based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labeled with the class variable NObesity (Obesity Level), that allows classification of the data using the values of Insufficient **Weight, Normal Weight, Overweight Level l, Overweight Level II, Obesity Type l, Obesity Type II and Obesity Type III.**

77% of the data was generated synthetically using the Weka tool and the SMOTE filter, 23% of the data was collected directly from users through a web platform.

Data Set Characteristics	**Multivariate**
Attribute Characteristics	Integer
Associated Tasks	Classification, Regression, Clustering
Number of Instances	21 1 1
Number of Attributes	17
Missing Values?	N/A
Area	Life
Date Donated	2019-08-27
Number of Web Hits	105220

More information regarding the dataset like - csv files , attribute information and processing done on the dataset is given in the subsequent pages.

IMPORTS

We have imported the following libraries to perform EDA, preprocessing, classification, and regression.

```
[56]: #import packages
      import csv
      import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns
      from scipy import stats
```

First, we will read the csv file ObesityDataSet_raw_and_data_sinthetic.csv with the help of panda libraries.

```
[57]: #read data
      df = pd.read_csv("ObesityDataSet_raw_and_data_sinthetic.csv")
```

```
[58]: df.head()
```

[58]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation	N
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	N
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation	N
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking	Ove
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation	Over

Applying the head function on dataframe which contains the csv file, we get the first 5 records.

DATA CLEANING

We will rename the last column which tells the type of obesity of the person from NObeyesdad to result.

```
[59]: df = df.rename(columns={'NObeyesdad': 'result'})
```

We will rename the last column which tells the type of obesity of the person from NObeyesdad to result

```
[60]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   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  result                          2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

```
[61]: df.shape
```

```
[61]: (2111, 17)
```

```
[62]: df.isnull().sum()
      # no data cleaning required
```

```
[62]: Gender                          0
      Age                             0
      Height                          0
      Weight                          0
      family_history_with_overweight  0
      FAVC                            0
      FCVC                            0
      NCP                             0
      CAEC                            0
      SMOKE                           0
      CH2O                            0
      SCC                             0
      FAF                             0
      TUE                             0
      CALC                            0
      MTRANS                          0
      result                          0
      dtype: int64
```

- Using the shape function we obtain the following inference:
    - Number of Rows which is the number of records - 2111.
    - Number of Columns which is the number of attributes – 17.
- Using the info function, we got to know that the data loads as text and float objects for most of the objects. However, we know that some are float, categorical and ordinal.

- Using the isnull().sum() function we infer that all the records are unique and contain no null values [e] Height and Weight are included however they have a direct correlation to each other and our target variable.

After the analysis of the csv file, we concluded that no data cleaning is required as no null values or missing values are present.

EXPLORATORY DATA ANALYSIS

- Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods.

- We will analyse the attributes of the dataset and summarize the variables, visualizing it and study the relationship between two variables and plot the data using matplotlib library and seaborn library.

These are the plots we are going to see below for the various attributes of the dataset:

- **Histogram is for quantitative data** — the number of bars depends on the user or to the software. It is an approximate representation of numerical data distribution.
- **Bar Chart is for categorical data** — the number of bars depends on the number of categories,
- **Scatter Plot** - Scatter Plot is yet another way of representation of attributes of a dataset. It is basically a type of plot or mathematical diagram which uses cartesian coordinates for the display of values. It uses two variables for a set of data , one on the X-axis and one on the Y-axis. In scatter plot , data is represented in form of points on the cartesian plane , with one variable determining the position on the horizontal axis and the other variable determining the position on the vertical axis.

```
[63]: def count_values(dataset, cat_variable, order = None):
          """
          Function: Counts values in each category and displays them on a plot.

          Parameters: Dataset, category feature, and order of appearance (order is optional).
          """
          ax = sns.countplot(x = cat_variable, data = dataset, palette = sns.cubehelix_palette(), order = order)
          for p in ax.patches:
              ax.annotate(f"\n{p.get_height()}", (p.get_x()+0.2, p.get_height()),
                          ha = "center", va = "top", color = "white", size = 10)

          plt.title(f"Number of items in each {cat_variable} category")
          plt.show()
```

count_values function takes the parameter dataset i.e. the dataframe which contains the csv file, an attribute of the csv file and the order we want to see in the plot.

- For example- Gender has two values male and female so [male, female] can be our order as well as [female, male].
- The function plots a bar graph i.e. it is used for categorical data.
- The function uses seaborn and matplotlib libraries.

```
[64]: def plot_distribution(dataset, feature):
          """
          Function: Computes and displays distribution of features with continuous values; plots their mean and median.

          Parameters: Dataset and feature with continuous values.
          """
          plt.hist(dataset[feature], bins = "fd")

          plt.axvline(dataset[feature].mean(), color = "red", label = "mean")
          plt.axvline(dataset[feature].median(), color = "orange", label = "median")

          plt.xlabel(f"{feature}")
          plt.ylabel("Count")
          plt.legend()
          plt.title(f"Distribution of values in {feature}")
          plt.show()
```

plot_distibution function takes the dataframe of csv file and attribute of the dataset. It does not require any order as it uses continuous values.

- The function plots a histogram and displays the mean and median of the attribute.
- The function uses matplotlib library.

[65]:
```
def cross_plot(dataset, lead_category, sup_category, order = None, hue_order= None):
    """
    Function: Plots interaction between two categorical variables.

    Parameters: Dataset, lead category, suplemental category, and order of appearance (order is optional).
    """

    ax = sns.countplot(x = lead_category, hue = sup_category, data = dataset, order = order, palette="Blues_r", hue_order= hue_order)
    for p in ax.patches:
        ax.annotate(f"\n{p.get_height()}", (p.get_x()+0.2, p.get_height()),
                    ha = "center", va = "top", color = "white", size = 10)

    plt.title(f"Number of items in each {lead_category} with respect to {sup_category}")

    plt.show()
```

cross_plot uses 5 parameters. First being the dataframe of the csv file, second the leading attribute which will be plotted on X-axis, third the supplemental attribute which will be plotted with respect to the result and then the orders of lead attribute and supplemental attribute respectively.

- It is a plot between two categorical variables.
- Tt plots a bar graph where each value of X-axis has n no. of bars where n is equal to the order of supplemental attribute.
- The function uses seaborn library.

[66]:
```
def cmp_plot(dataset, lead_category, sup_category, order = None):
    """
    Function: Plots interaction between numerical and categorical variables.

    Parameters: Dataset, lead category, suplemental category, and order of appearance (order is optional).
    """

    ax = sns.histplot(data= dataset, x=sup_category, hue=lead_category, palette= "rocket", hue_order= order)

    plt.show()
```

cmp_plot has the dataset, lead attribute, supplemental attribute, and order of lead attribute as parameters.

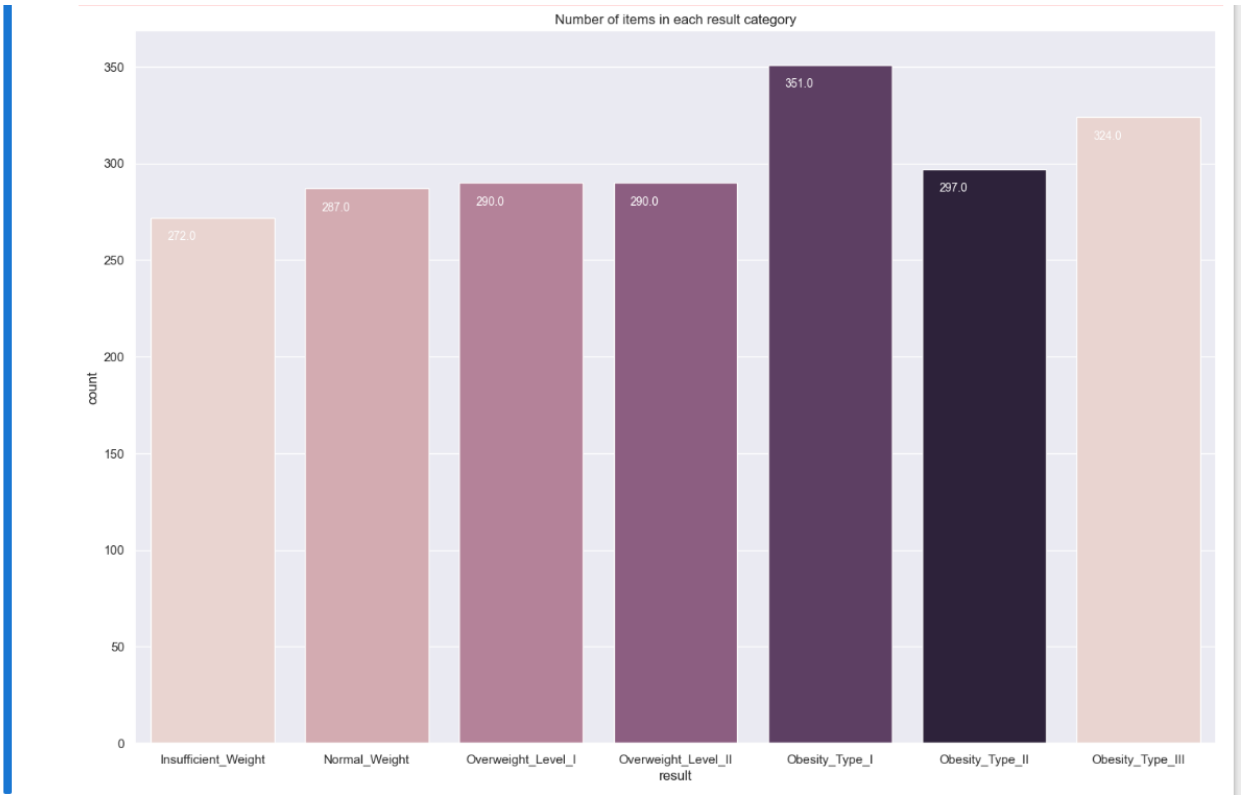
- It plots between numerical and categorical variables.
- It plots a histogram.
- The function uses seaborn library.

OBESITY LEVELS

- It is the different values we can see in the result column also known as obesity levels hence this order will be used in cross_plot and cmp_plot function for the lead attribute.

Result

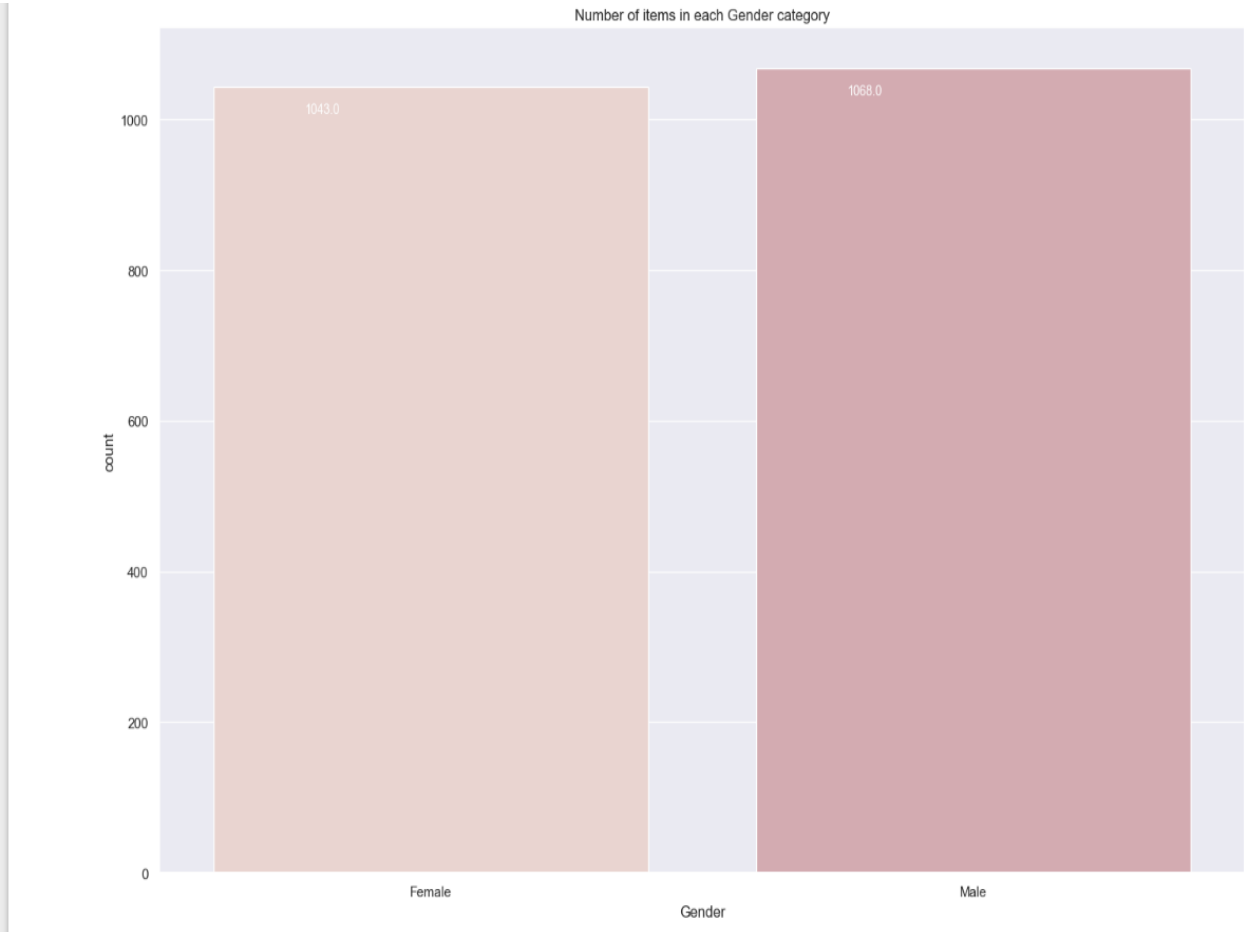
```
[69]: count_values(df, "result", weight_order)
```



- 351 instances belong to “obsesity_type_1”, more than 250 instances belong to “insufficient_weight" class.
- The plot shows the dataset is balanced; only "Obese Type I [II] class slightly outnumber the other categories.

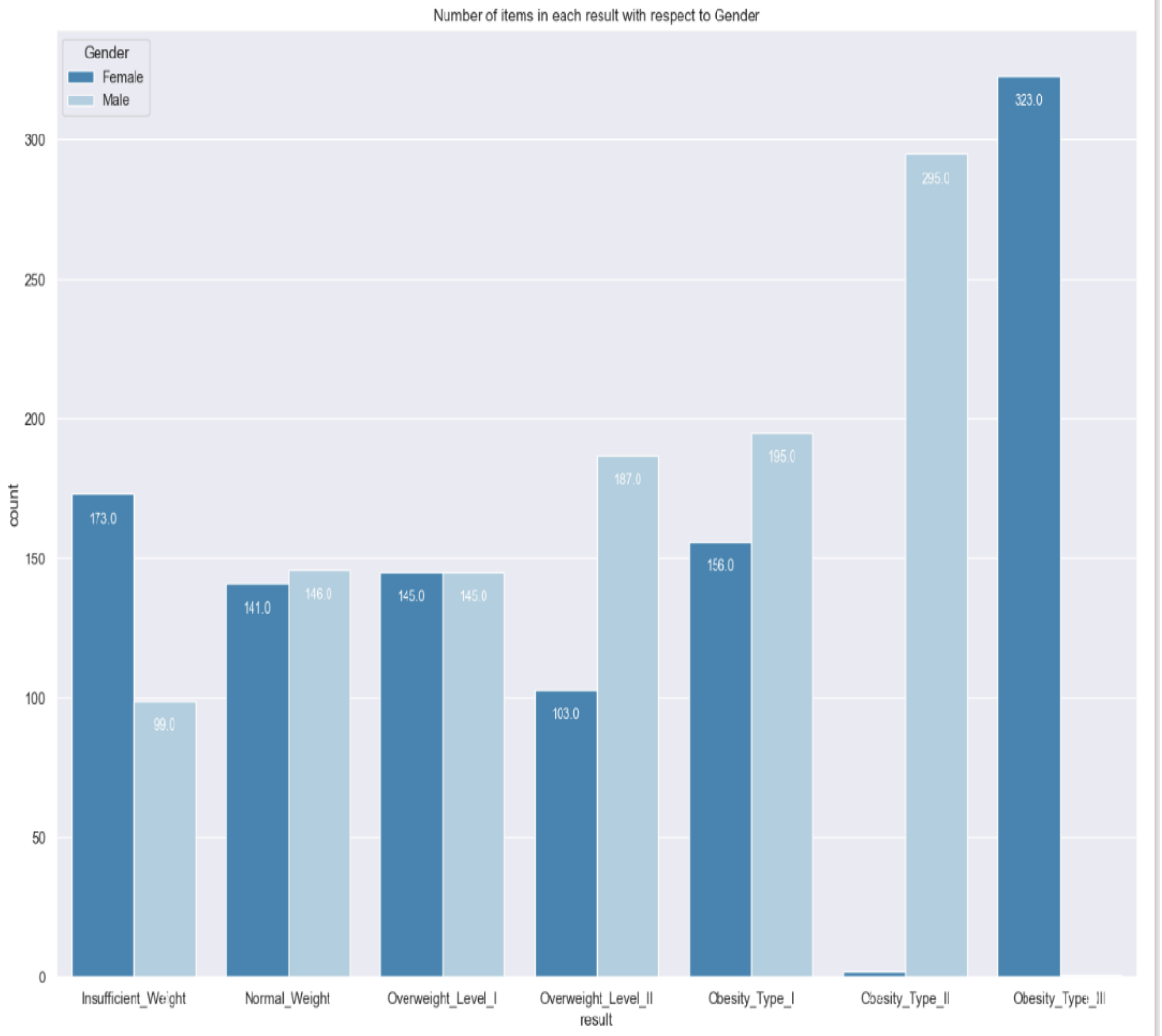
Gender

```
[70]: count_values(df, "Gender")
```



- There are almost an equal number of females and males in the dataset.
- Data is available for slightly more men than women, but this does not make it imbalanced.

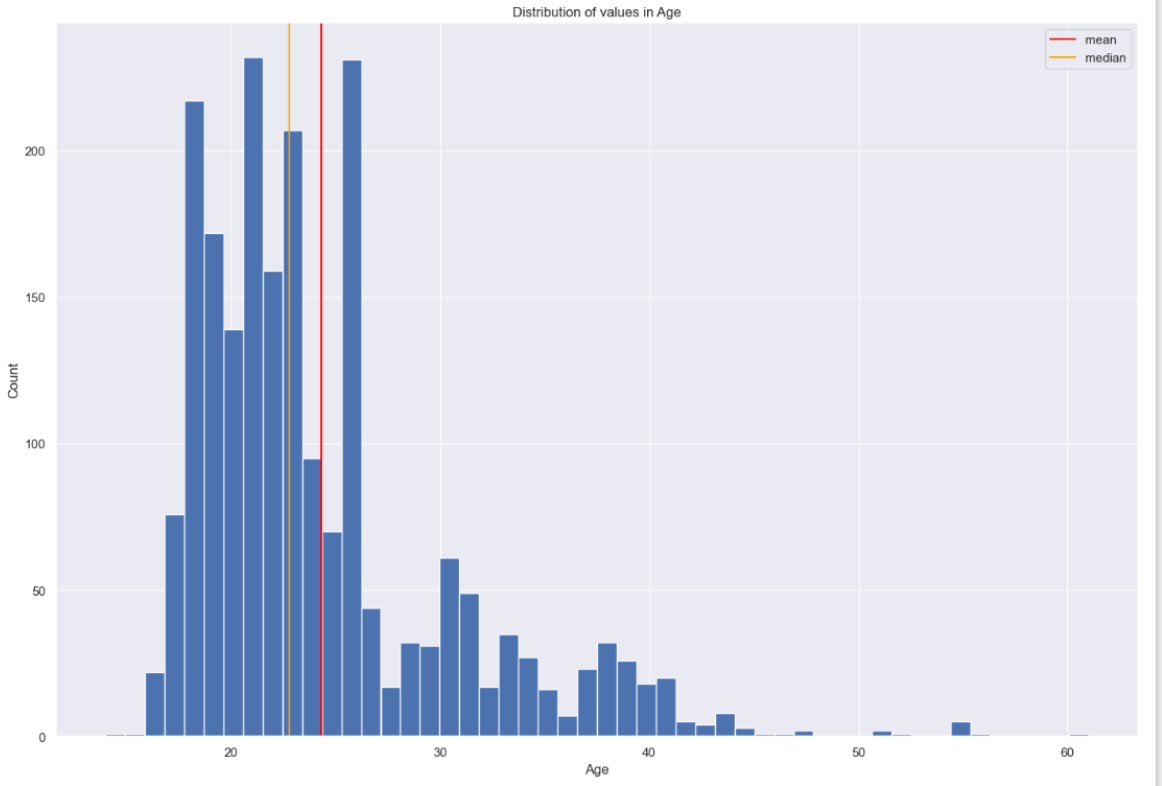
```
[71]: cross_plot(df, "result", "Gender", weight_order)
```



- Women are more obese men than women, save in the last, extreme obesity category.
- Females do not have 'obesity_type_2". males do not have"obesity_type_3".

Age

```
[72]: plot_distribution(df, "Age")
```



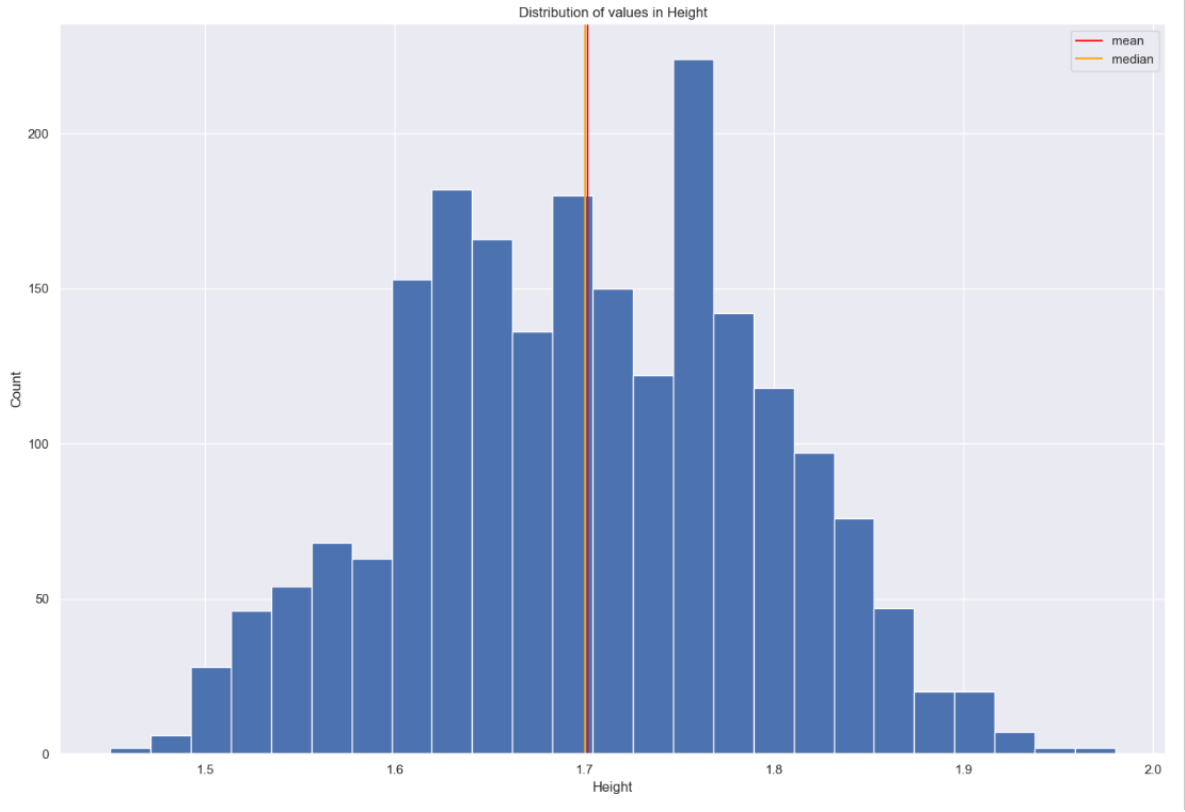

```
[73]: df["Age"].describe()
```

```
[73]: count    2111.000000
      mean       24.312600
      std         6.345968
      min        14.000000
      25%        19.947192
      50%        22.777890
      75%        26.000000
      max        61.000000
      Name: Age, dtype: float64
```

- We can see that mostly young people joined the survey.
- Top 3 ages: 21, 18, 19

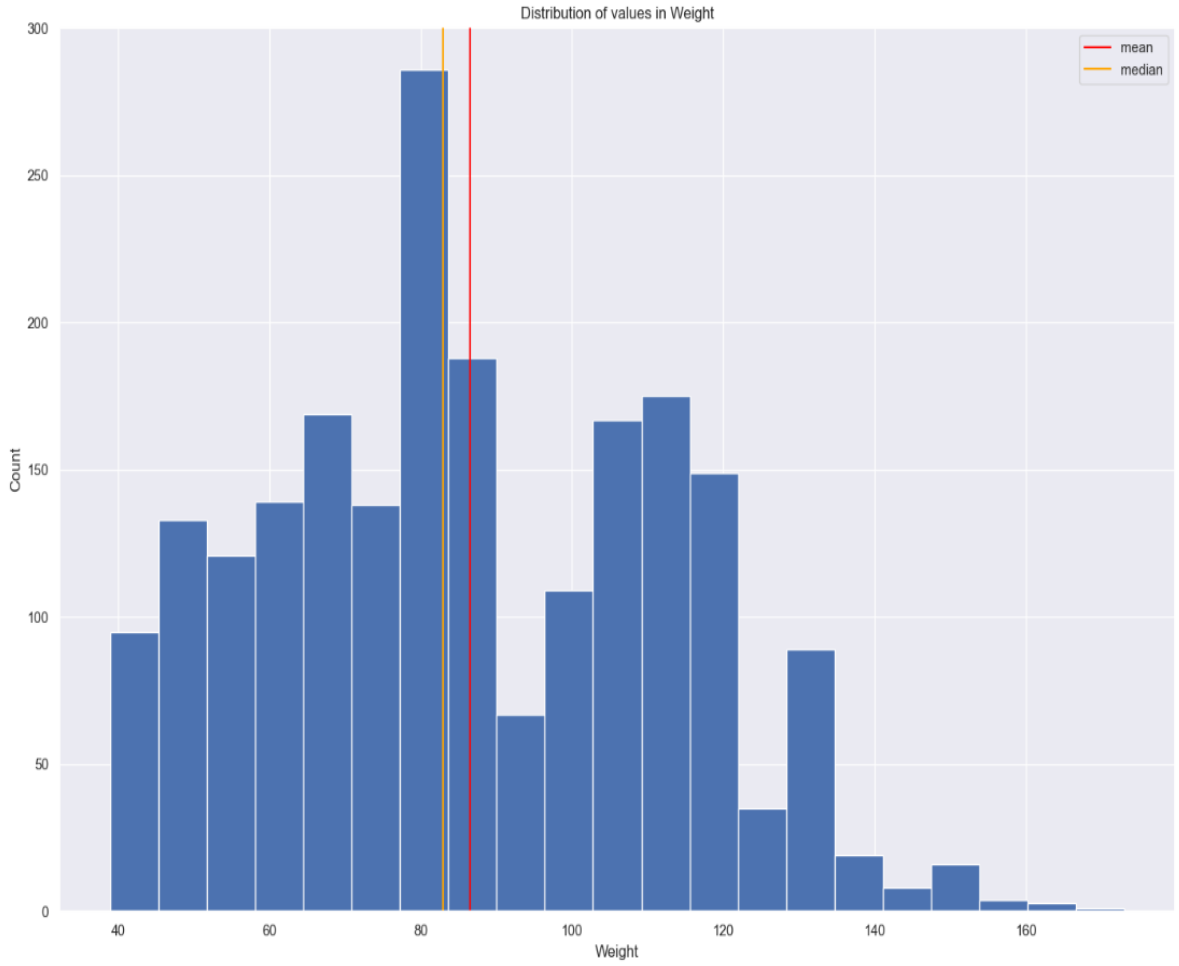
Height and Weight

```
[74]: plot_distribution(df, "Height")
```



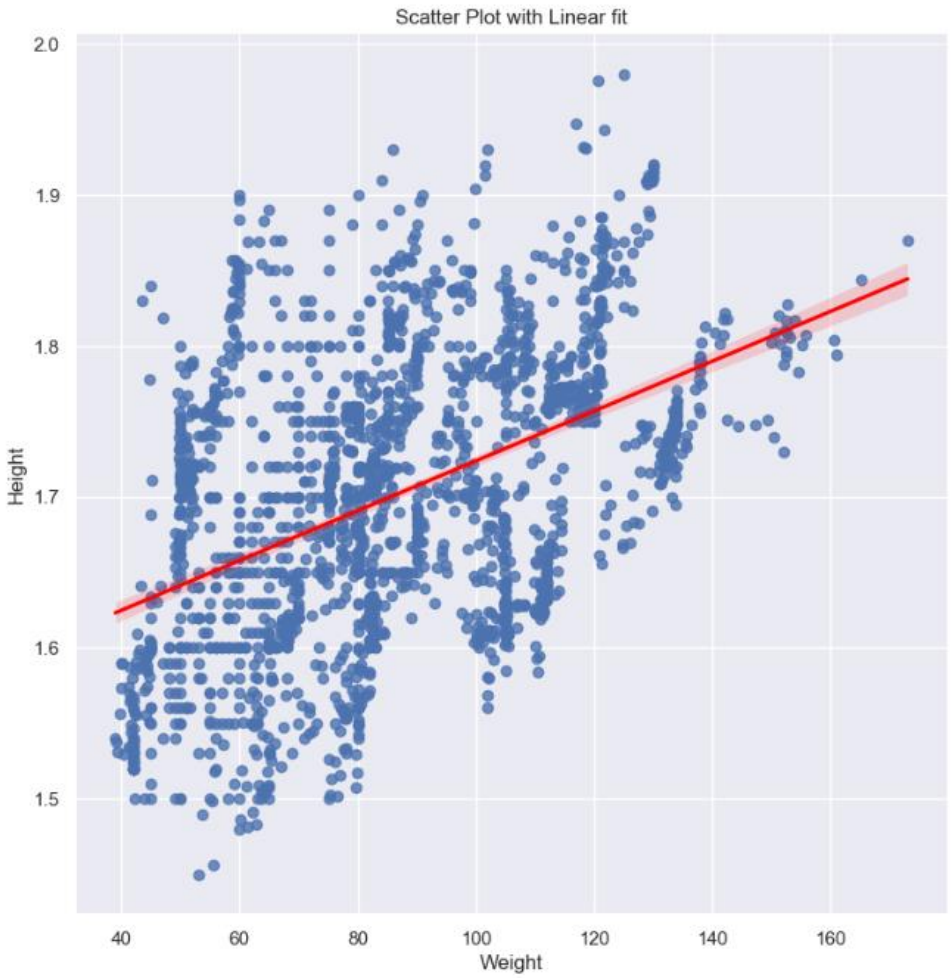
- Most people are 1.60 m - 1.85 m tall. Both mean and median values are around 1.70.

```
[75]: plot_distribution(df, "Weight")
```



- The mean and the median are shifted to the left because of the larger number of people weighing 80 kg.

[76]:
```
# Obesity is calculated by BMI
# Body Mass Index (BMI) = Weight/(Height)^2
# Correlation Between Height & Weight


sns.lmplot(x='Weight', y='Height', data=df, height= 8, line_kws={'color': 'red'})
plt.title("Scatter Plot with Linear fit");
```

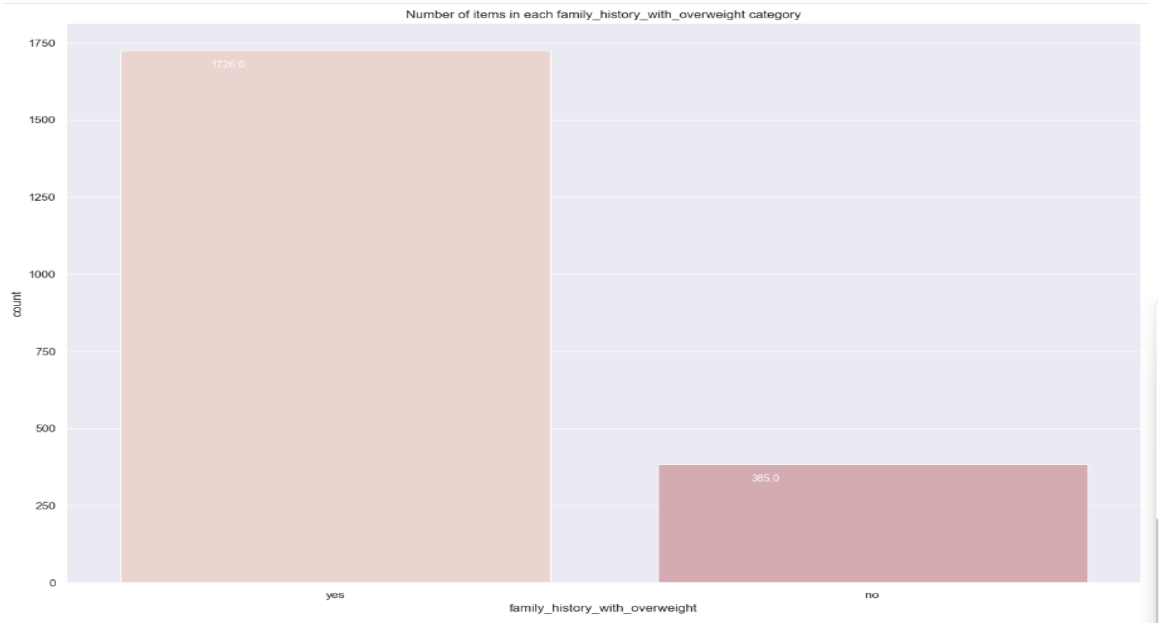


- The red line shows that there is a positive correlation between them, which means an increase in one variable leads to an increase in the other. In other words, taller people are more likely to weigh more.

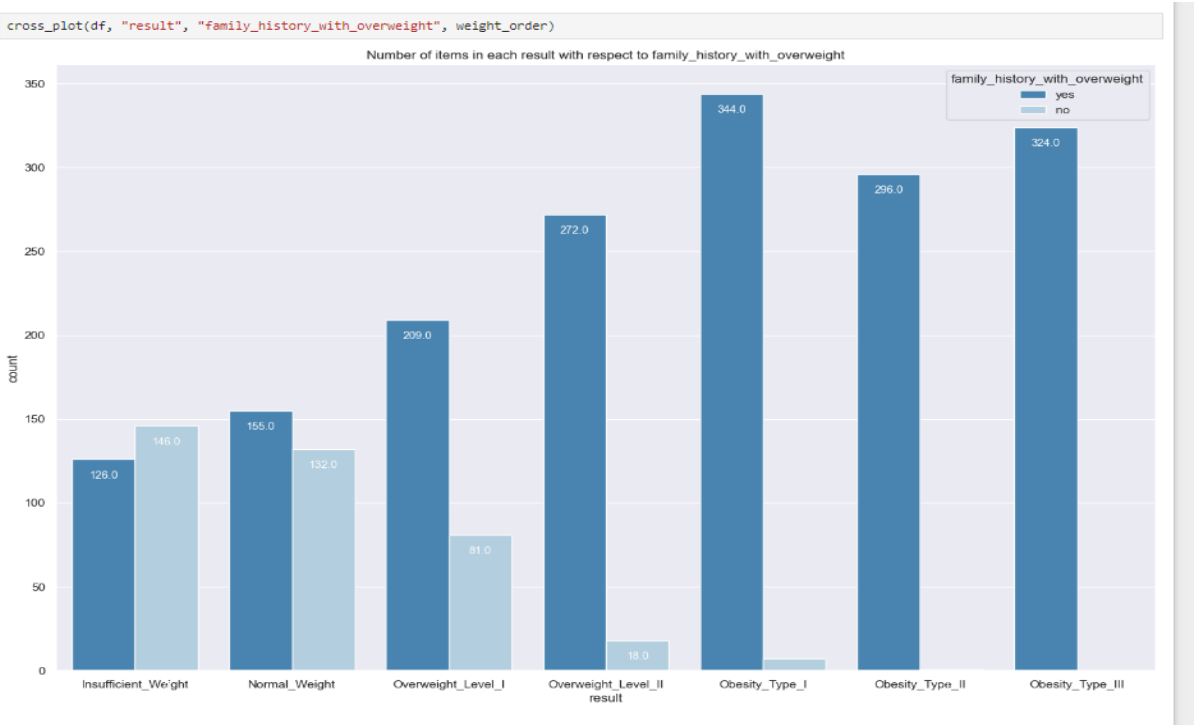
Family History

All those categorized as overweight or obese had family members suffering from weight problems.

```
[77]: count_values(df, "family_history_with_overweight")
```

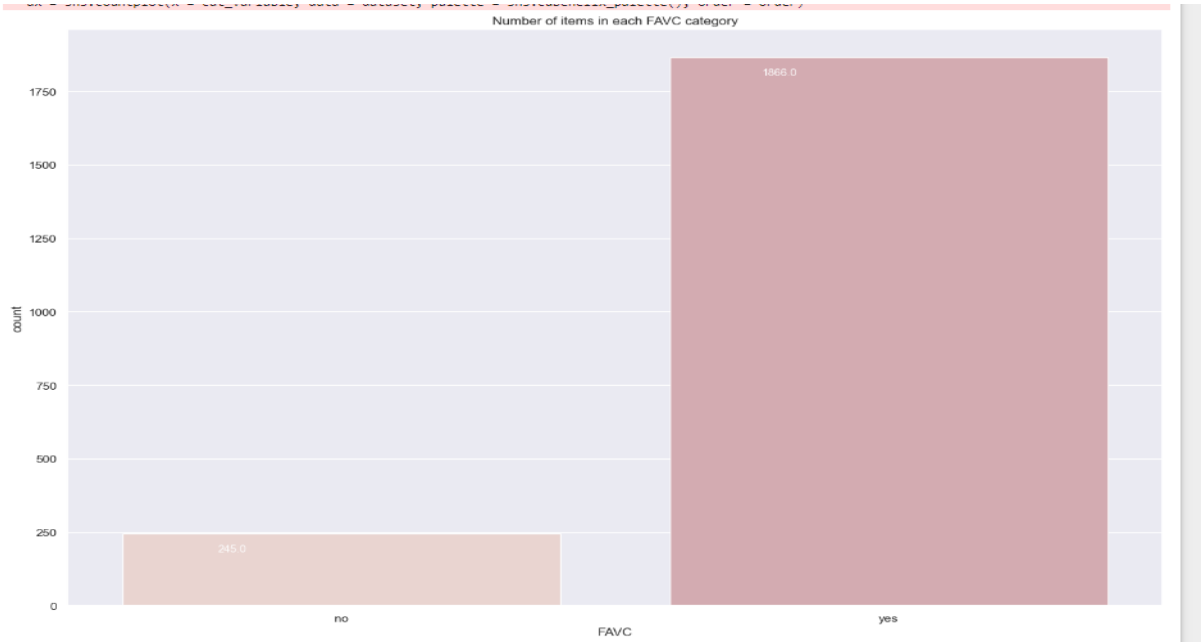



```
[78]: cross_plot(df, "result", "family_history_with_overweight", weight_order)
```

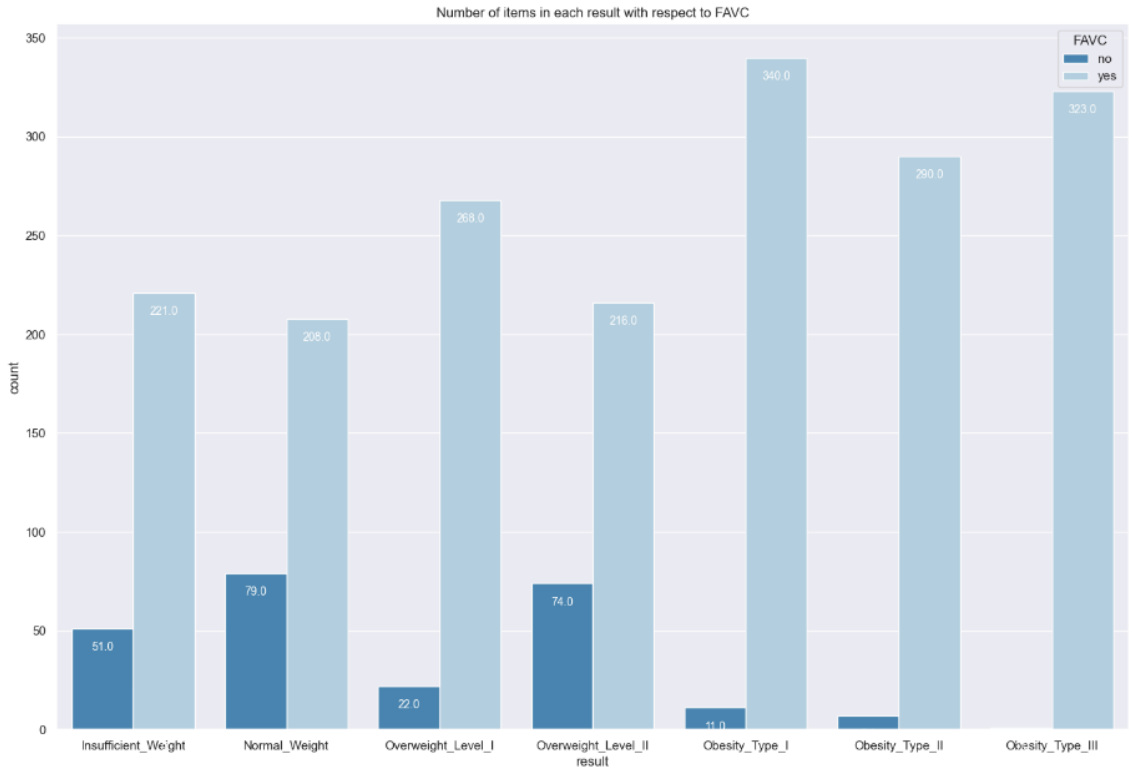


High Caloric Food Consumption

```
[79]: count_values(df, "FAVC")
```



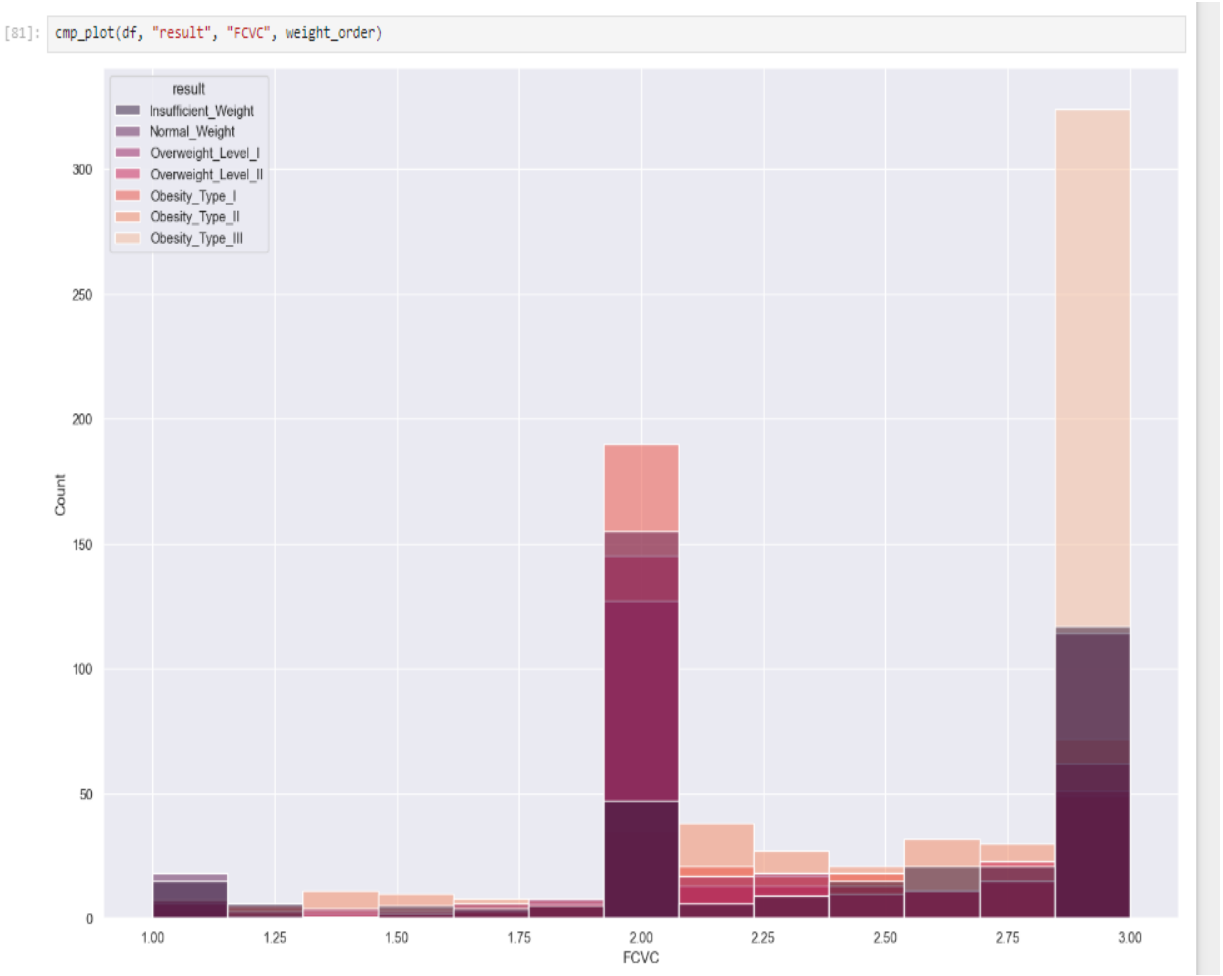

```
[80]: cross_plot(df, "result", "FAVC", weight_order)
```



Consumption of Vegetables

- "FCVC " column denotes if people consume vegetables.

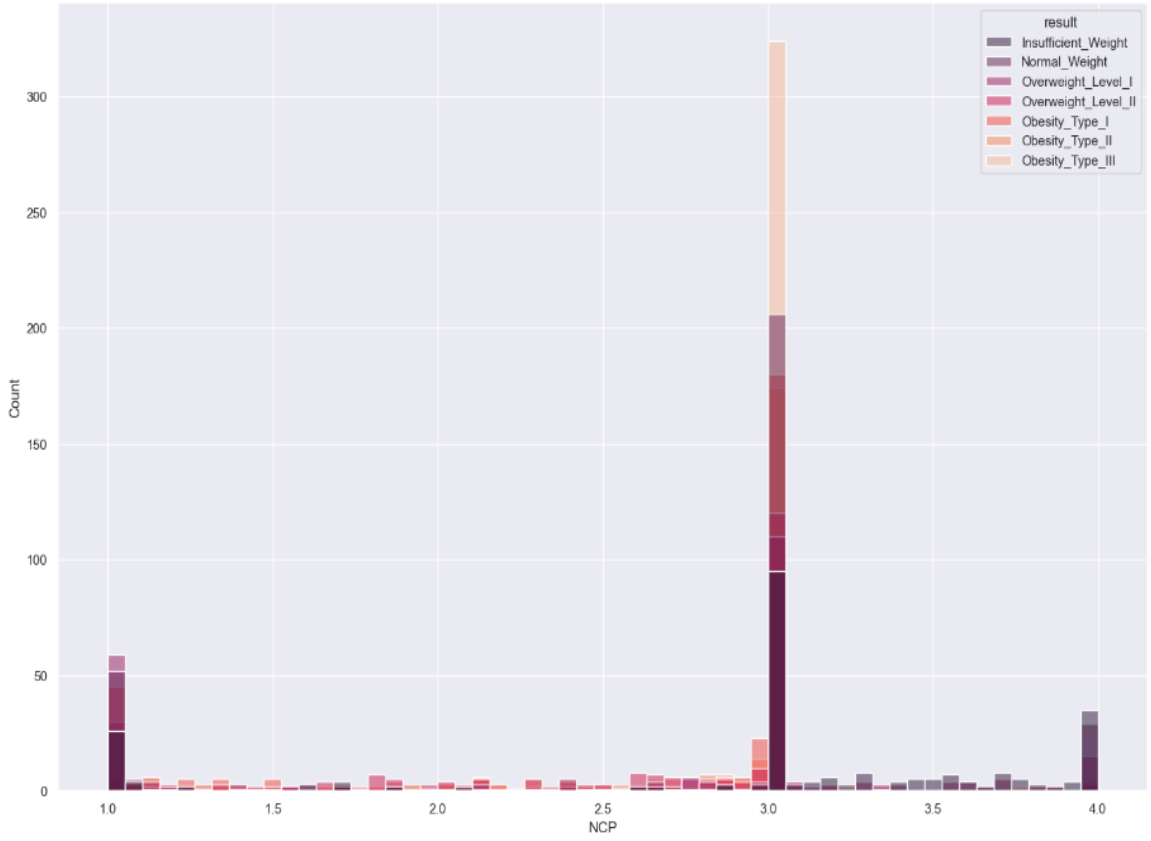
- It could be assumed that "3" means "Always", "2" - "Sometimes", and " 1 " - 'Never", but it is not clear what the values in-between mean.



Meals per Day

- From the above plot we can infer that most obese type people have meals more than 3 times.

```
[82]: cmp_plot(df, "result", "NCP", weight_order)
```

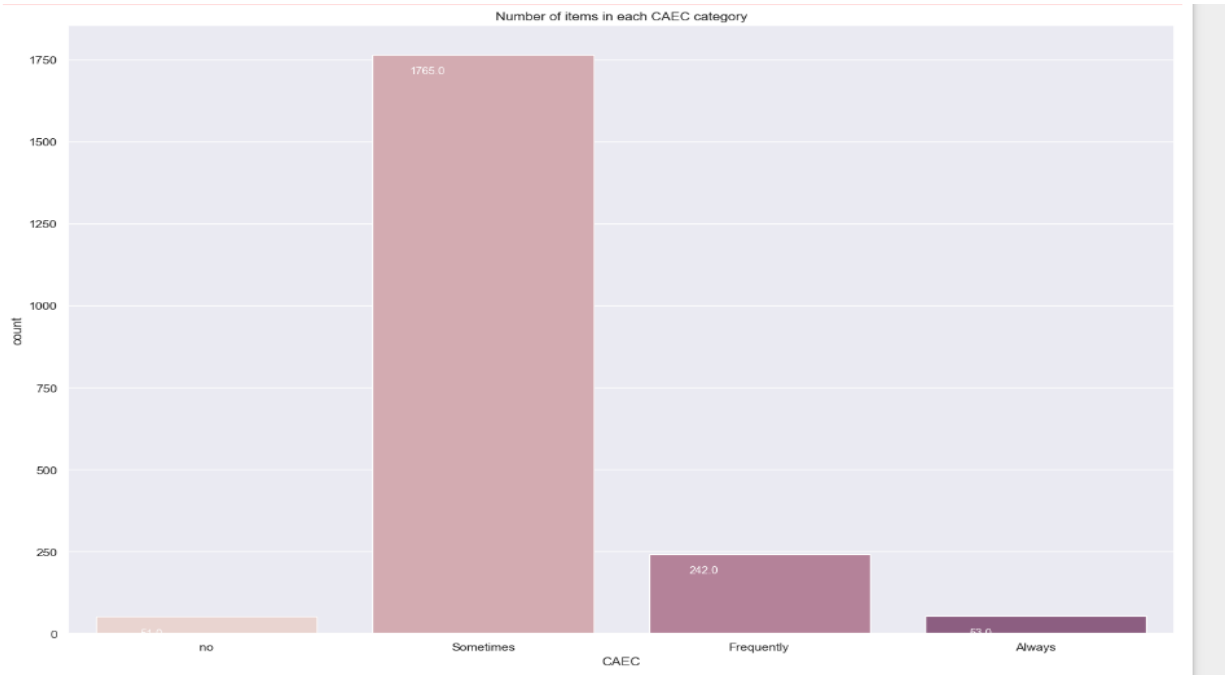



```
[83]: df["NCP"].describe()
```

```
[83]: count    2111.000000
      mean        2.685628
      std         0.778039
      min         1.000000
      25%         2.658738
      50%         3.000000
      75%         3.000000
      max         4.000000
      Name: NCP, dtype: float64
```

Food between Meals

```
[84]: count_values(df, "CAEC", ["no", "Sometimes", "Frequently", "Always"])
```

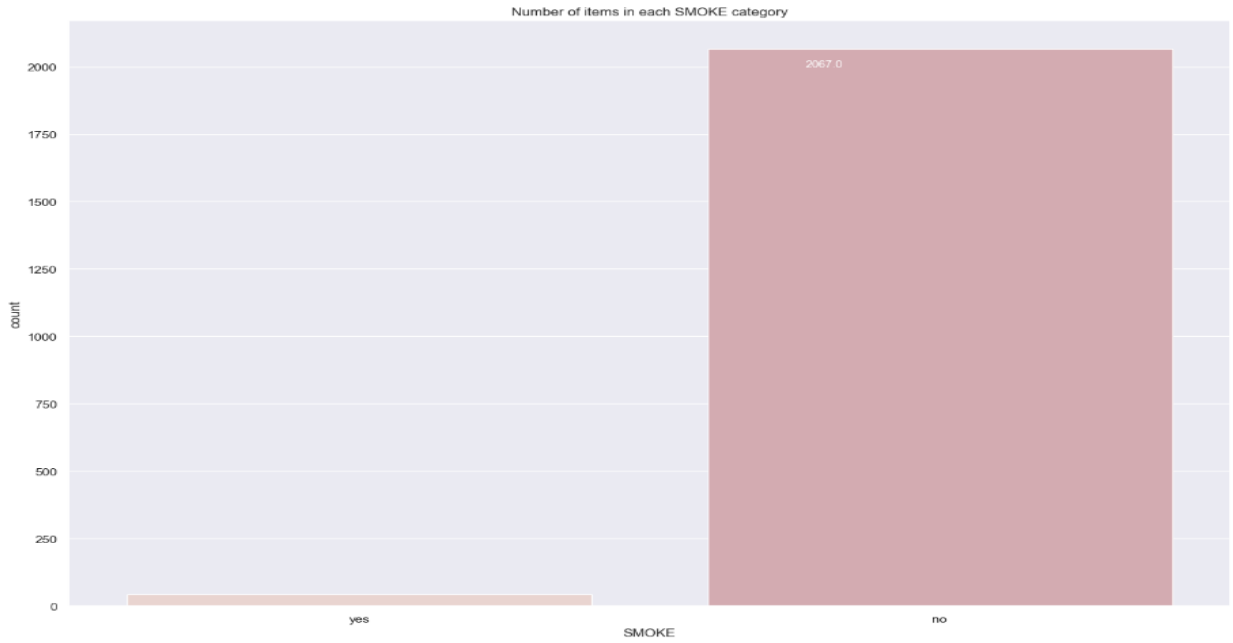


- Most of those who have food between meals sometimes are found to be in obesity type 13 range and overweight levels.

```
[85]: cross_plot(df, "result", "CAEC", weight_order, hue_order= ["no", "Sometimes", "Frequently", "Always"])
```

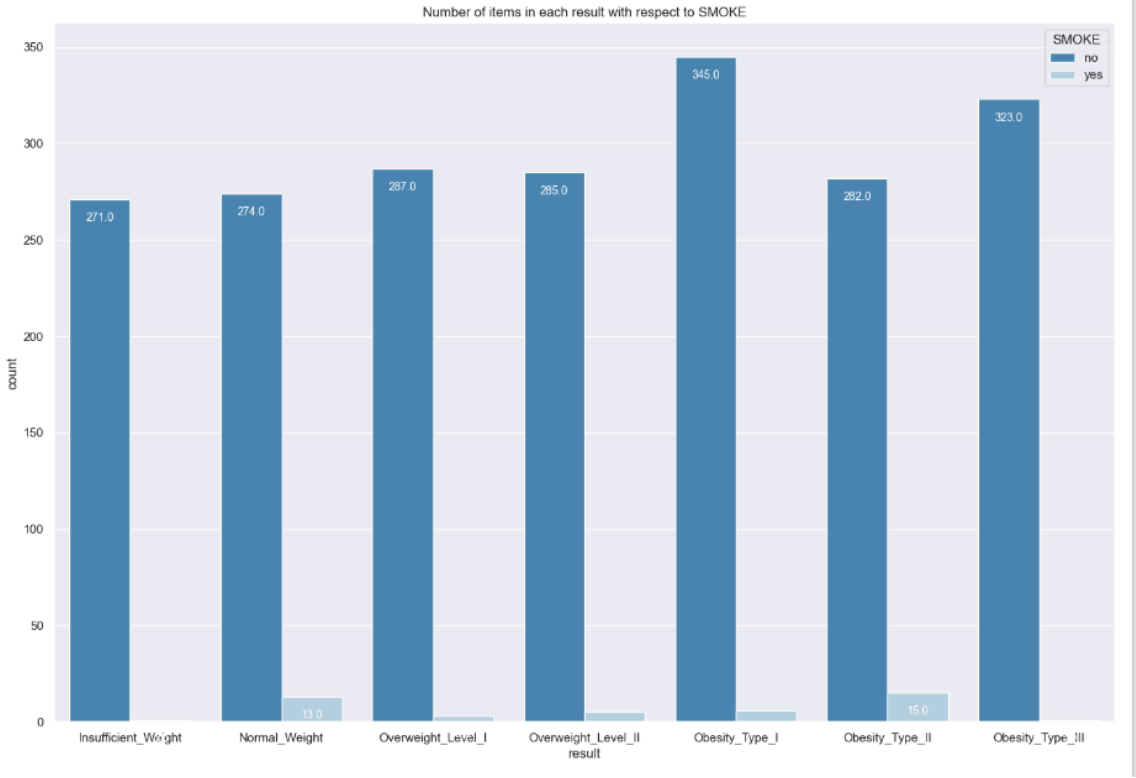
Smoke

```
[86]: count_values(df,"SMOKE",["yes","no"])
```



- Most of the people in the survey are non-smokers which makes it difficult to find inference about smokers.

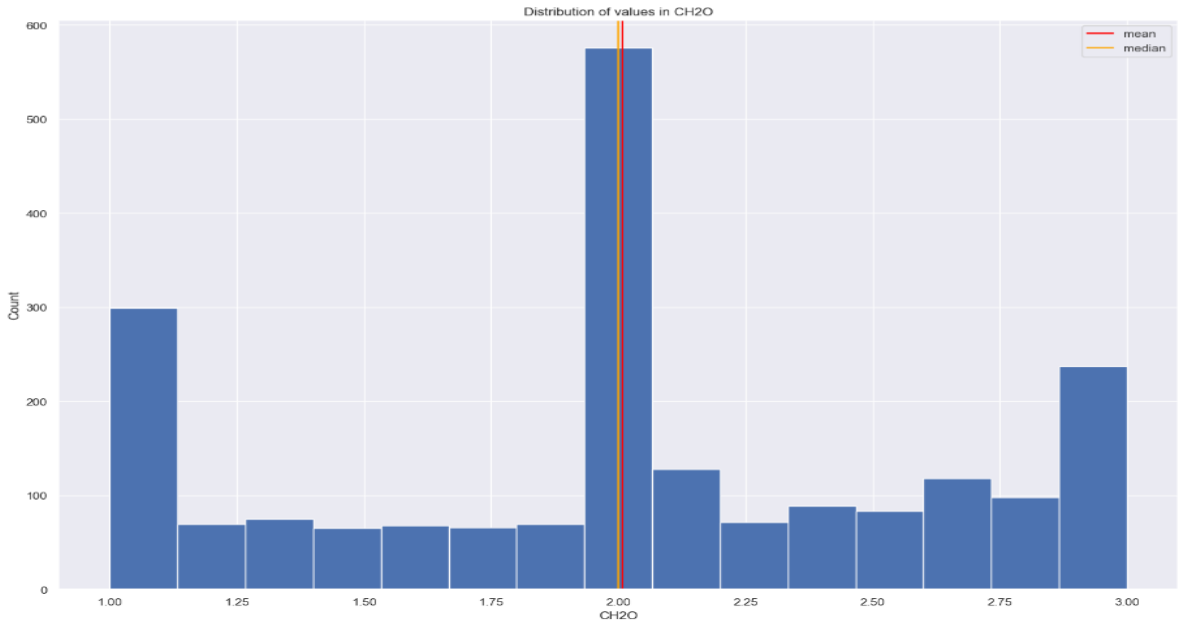
```
[87]: cross_plot(df, "result", "SMOKE", weight_order)
```



- Analyzing the plot, we understand that non-smoking does not have any effect on obesity or insufficient weight as all the attributes are ranging from 270-345 hence not much deviation

Water Intake

```
[88]: plot_distribution(df,"CH2O")
```

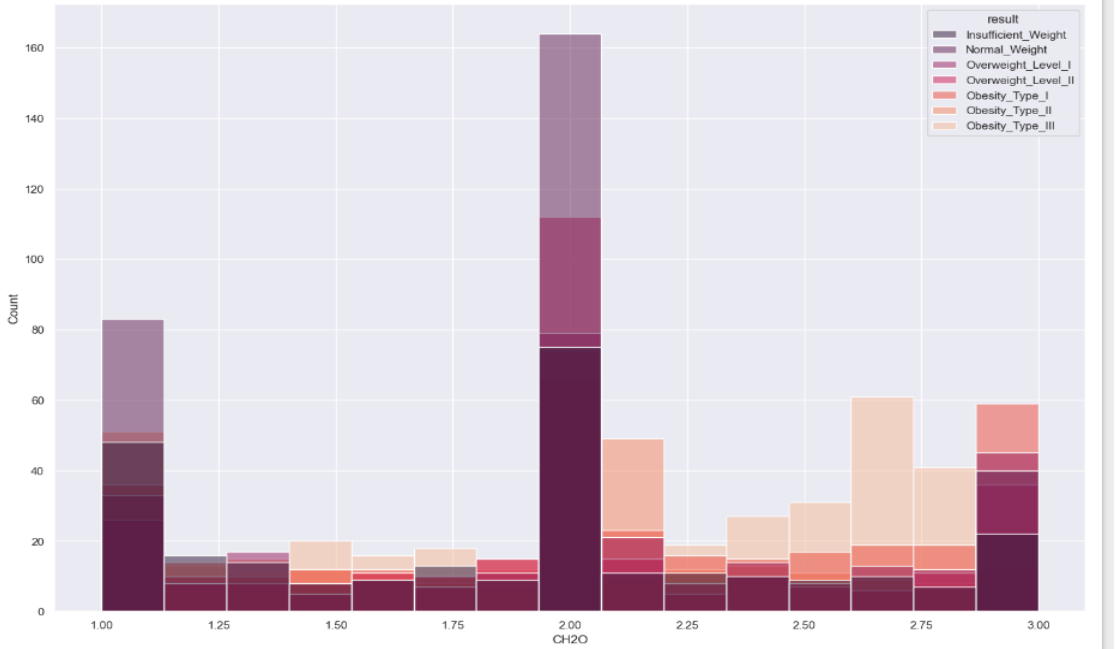



```
[89]: df["CH2O"].describe()
```

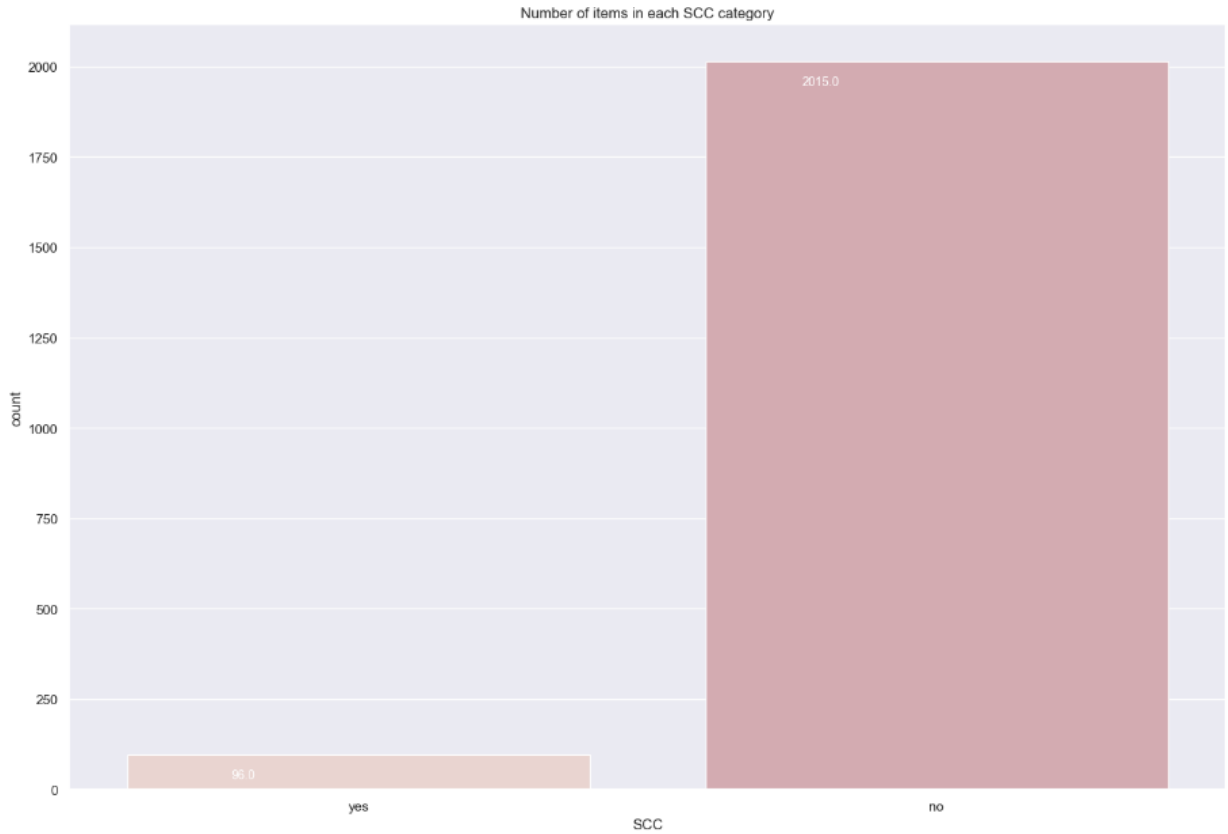
```
[89]: count    2111.000000
      mean        2.008011
      std         0.612953
      min         1.000000
      25%         1.584812
      50%         2.000000
      75%         2.477420
      max         3.000000
      Name: CH2O, dtype: float64
```

- According to the distribution and statistics we can infer that most of the people in the survey drink 2 liters of water and there is no one who drinks less than a liter.

```
[90]: cmp_plot(df, "result", "CH2O", weight_order)
```

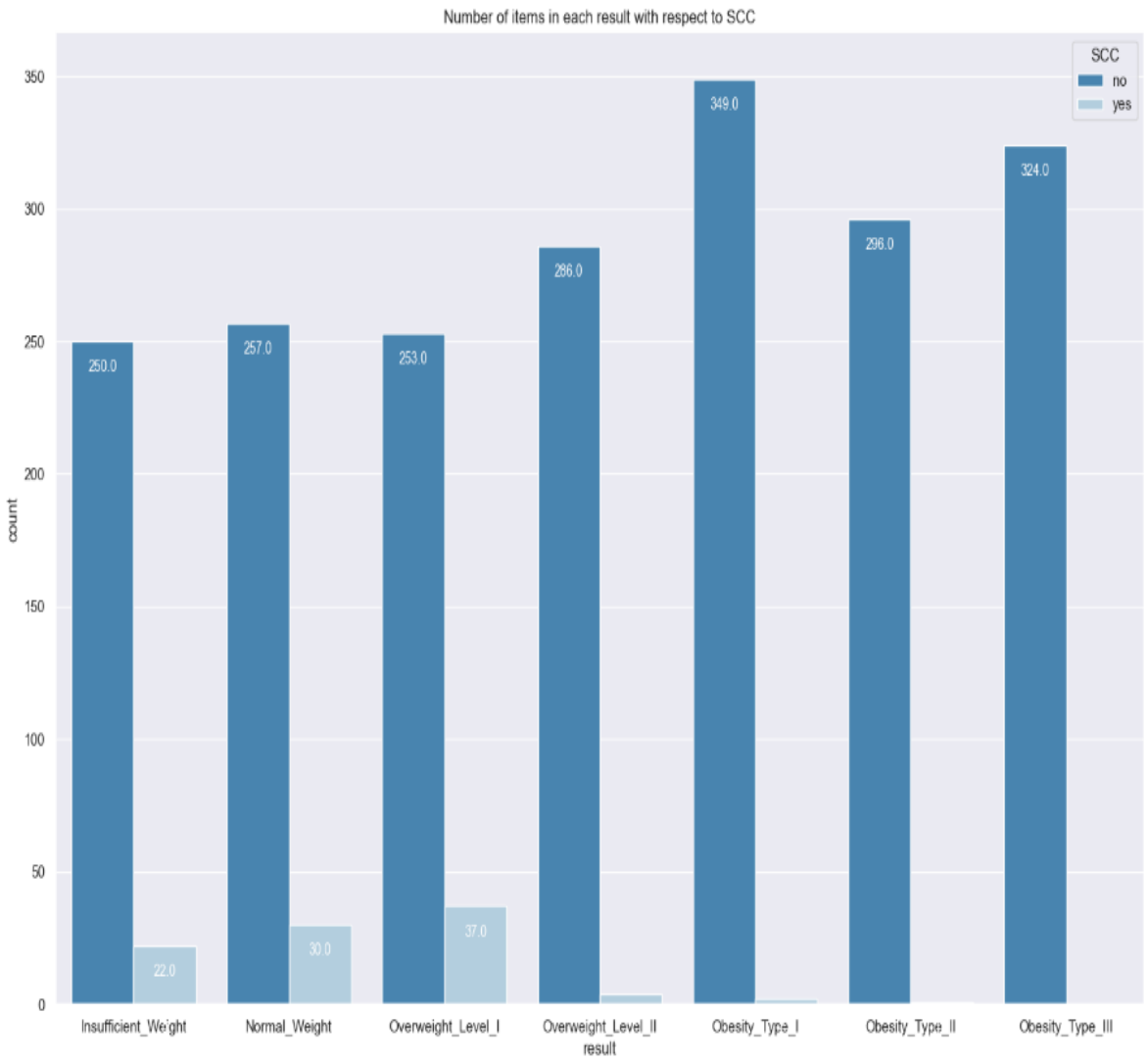


Record Calories Intake



- Most of the people who took part in the survey did not record their daily calories intake.

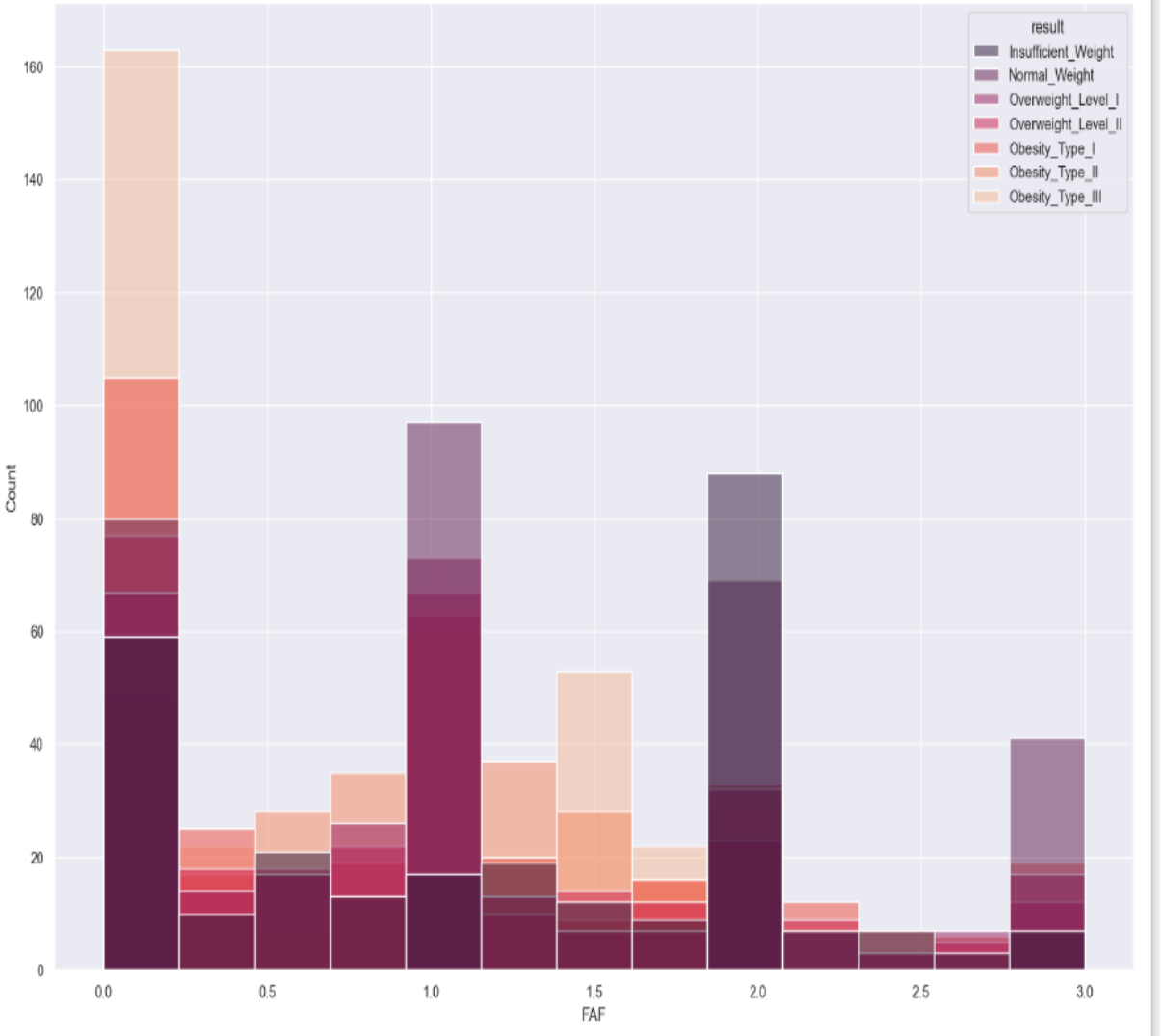
```
[92]: cross_plot(df, "result","SCC", weight_order,)
```



- The majority of the people who recorded their daily calories intake were either insufficient weight or normal weight or overweight but the majority of those who did not measure their calories intake belong to overweight and obesity categories.

Physical Activity

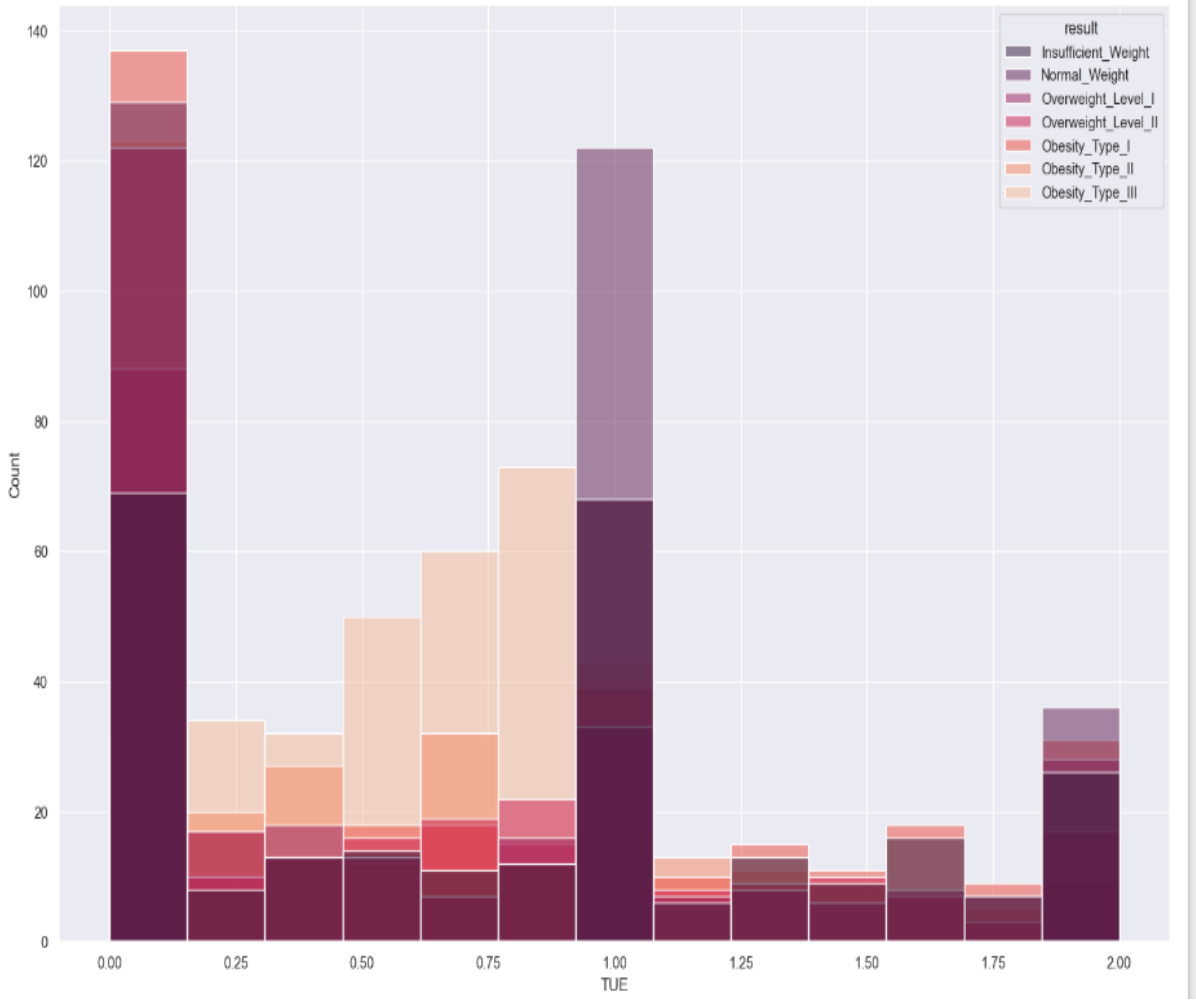
```
[93]: cmp_plot(df, "result", "FAF", weight_order)
```



- People who do not do physical activity more than 2 days a week suffer from Obesity type-3, type-2 and type-I.

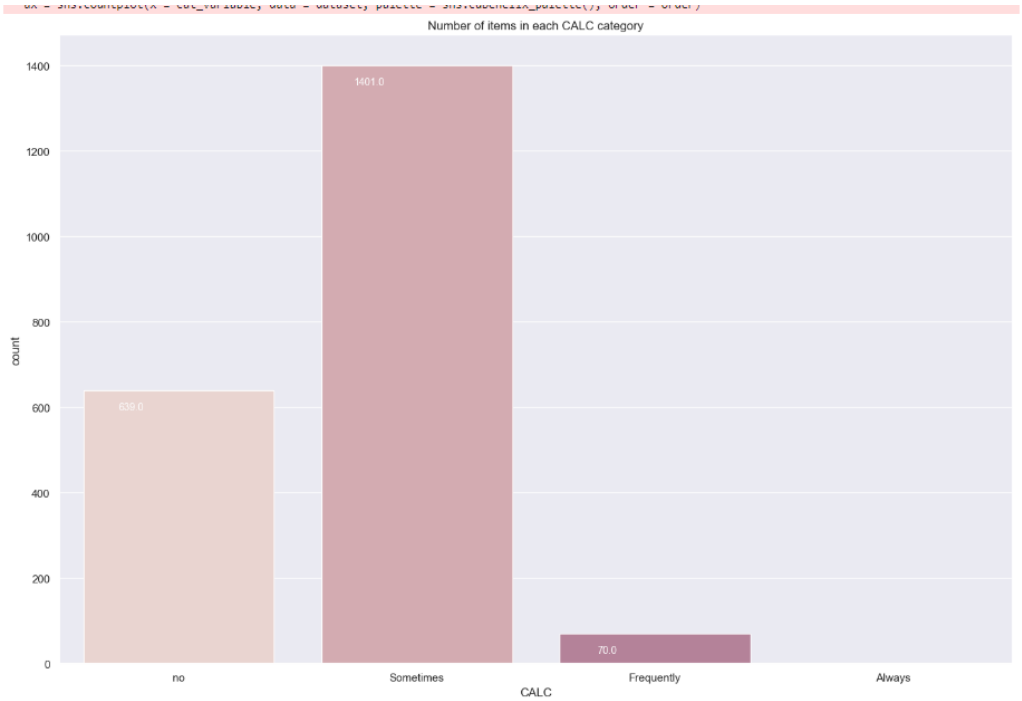
Use of Technological Devices

```
[94]: cmp_plot(df, "result", "TUE", weight_order)
```

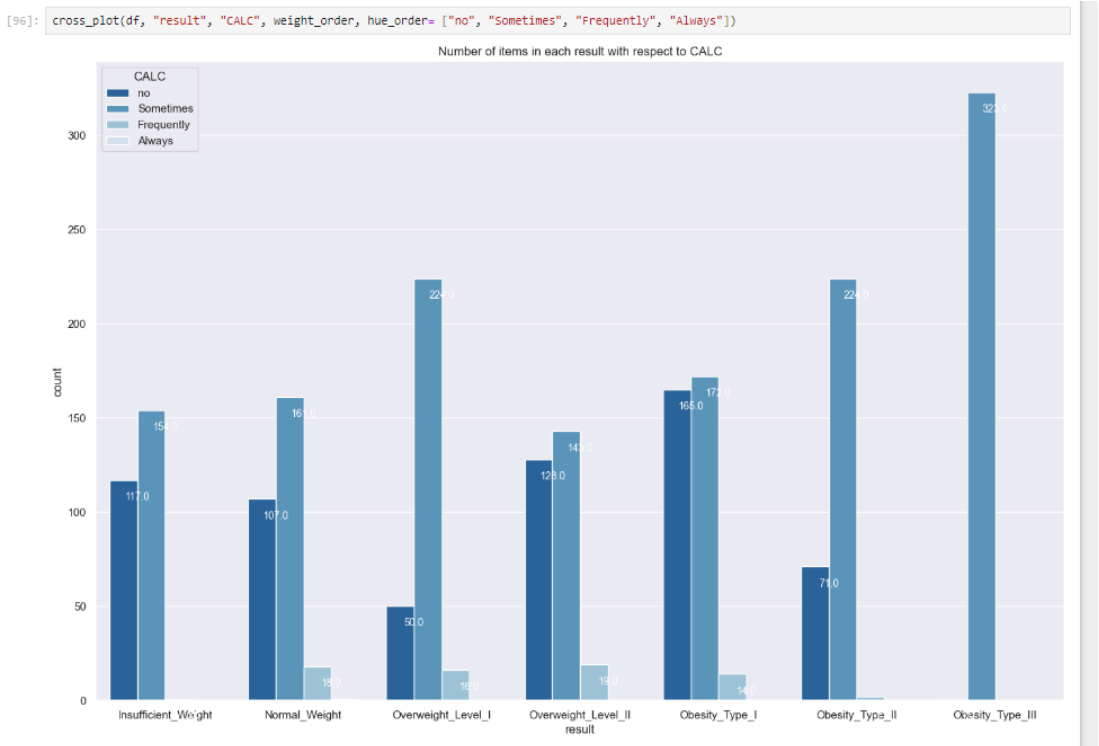


- The plot shows that most of the people suffering from obesity type 1 to 3 do not use technological devices for more than hour.

Alcohol Consumption

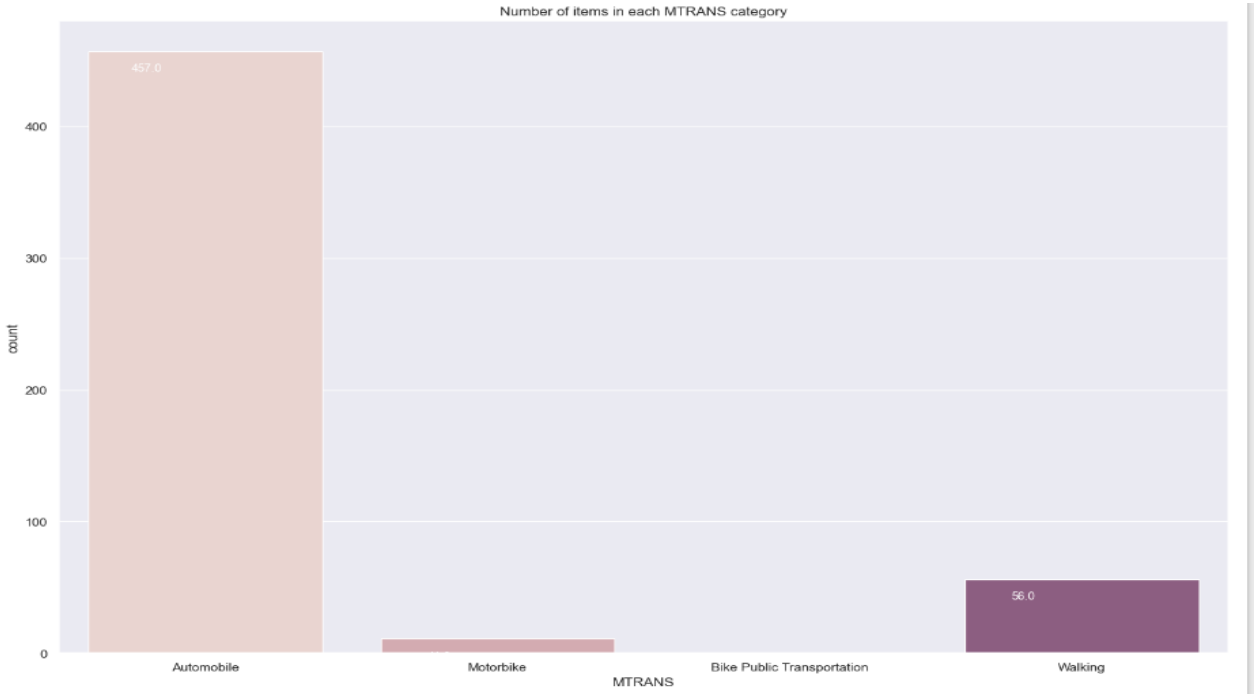


- Most of the people who took part in the survey either did not consume alcohol or consumed sometimes.



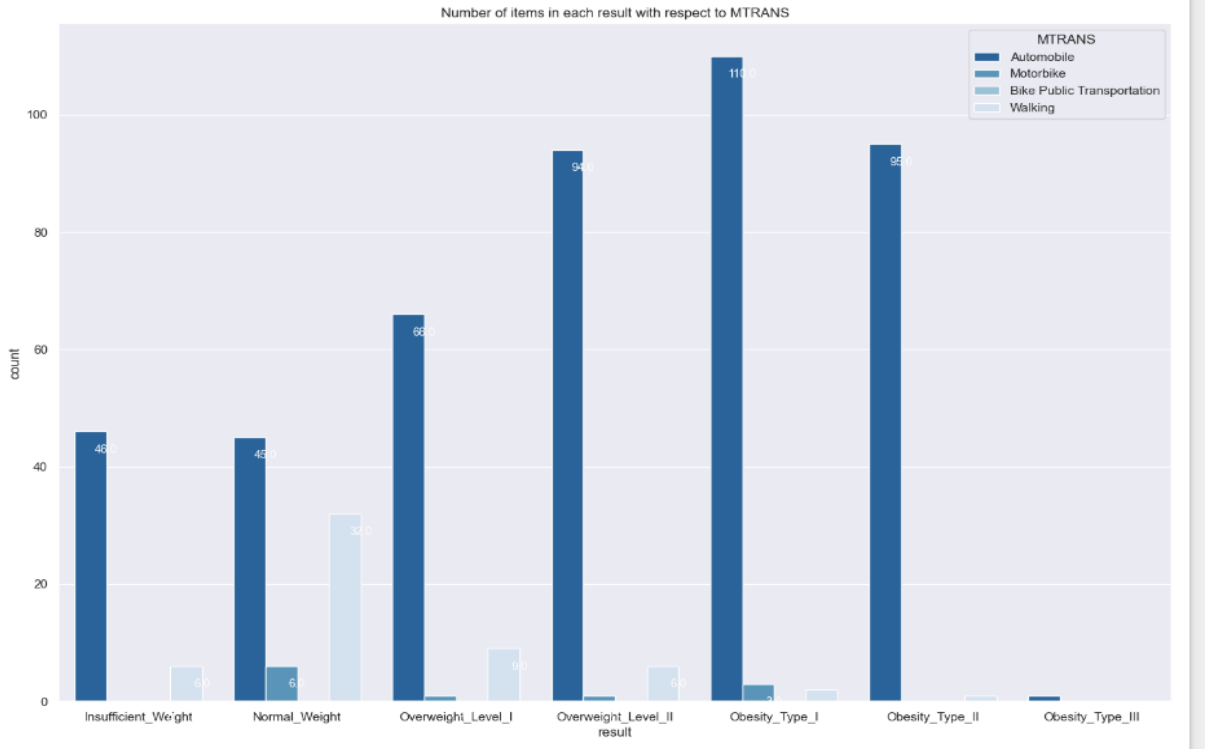
- Most of the people who took part in the survey used automobiles.

Transportation



- The majority of the people who prefer walking belong to the insufficient to overweight category i.e. people who walk to commute from one place to another are not suffering from obesity.

```
[98]: cross_plot(df, "result", "MTRANS", weight_order,["Automobile","Motorbike","Bike Public Transportation","Walking"] )
```



Correlation

Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statement about cause and effect. We can determine that there are two types of correlation .

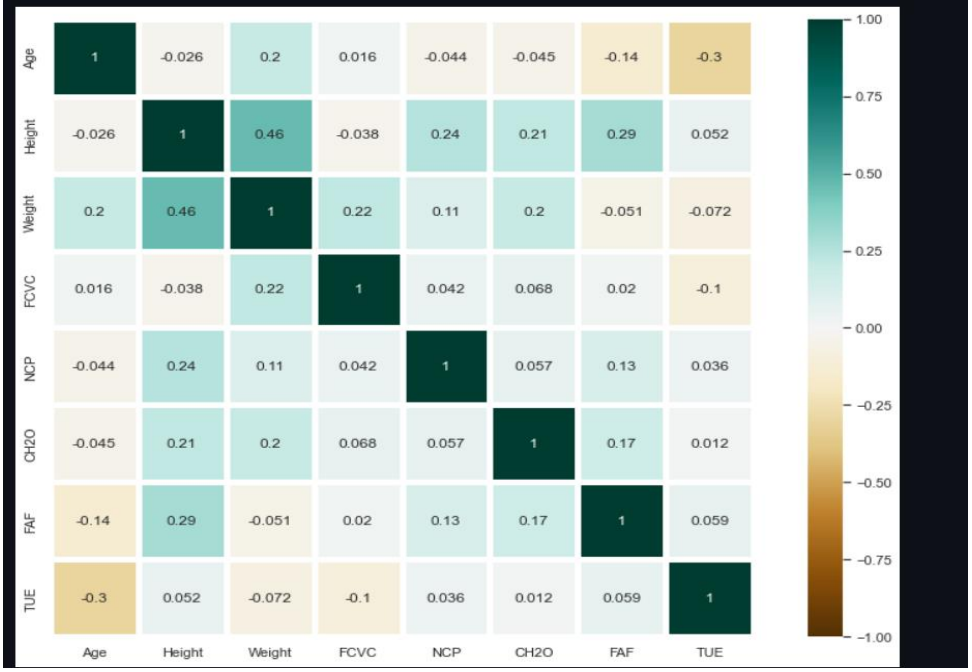
1. **Positive correlation** : In positive correlation , if one variable increase then the other variable increases and same if one variable decrease then the other variable decreases. The slope of the fitting line in such a case would be positive.
2. **Negative correlation** : In negative correlation , the inverse relationship is there between the correlated variables , that is if one variable increases/decreases then the other variable decreases/increases.

Here we are using the heatmap function of the seaborn library to obtain the correlation between attributes of the dataset.

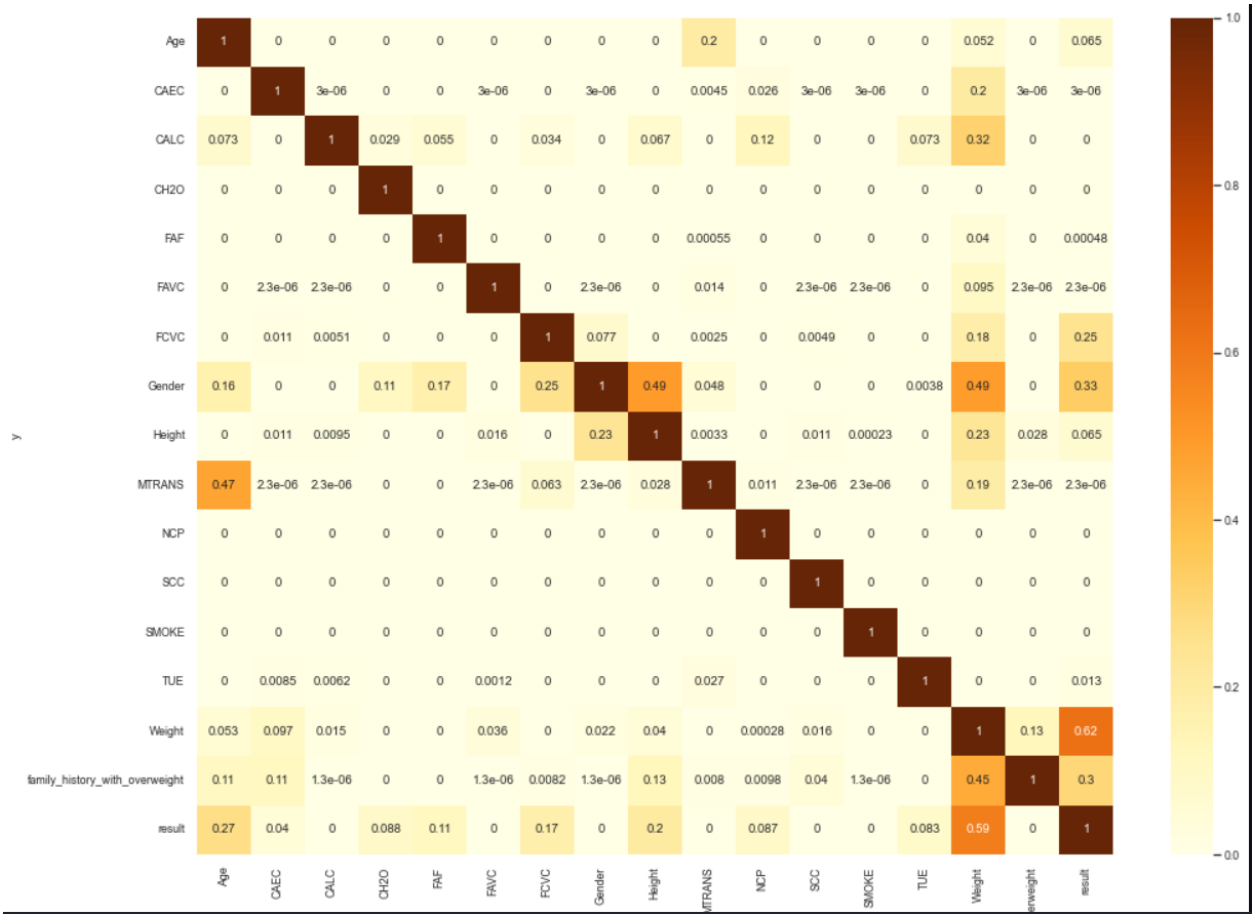
The PPS is an asymmetric, data-type-agnostic score that can detect linear or non-linear relationships between two columns. The score ranges from 0 (no predictive power) to 1 (perfect predictive power). It can be used as an alternative to the correlation (matrix).

```
plt.figure(figsize = (15, 10))
sns.heatmap(df.corr(),annot=True,square=True,center=0,vmin=-1,vmax=1,
            cmap='BrBG',linewidths=5)
```

```
<AxesSubplot: >
```



```
import ppscore as pps
import warnings
warnings.filterwarnings("ignore")


matrix_df = pps.matrix(df).pivot(columns='x', index='y',  values='ppscore')

sns.set(font_scale=0.8)
plt.figure(figsize=(16,12))
sns.heatmap(matrix_df, annot=True, cmap = sns.color_palette("YlOrBr", as_cmap=True))
plt.show()
```



Inferences of the Dataset

- The dataset contains integer and float values, it also contains text objects.
- The data did not require any cleaning as there were no null or missing values.
- The dataset had all types of people in the survey from insufficient weight to obesity type 3 and every type had almost an equal no. of people except obesity type 1 which had more people with respect to other categories.

- There was almost equal no. of people of both genders and women are more likely to have insufficient weight whereas men suffer from obesity.
- People with family history of obesity were either overweight or obese.
- Most of the people in the survey were non-smokers, hence little inference about smokers was made.
- Most of the people do not record their daily calories intake.
- The correlation matrix which is only applicable on numerical data showed that height and weight are highly correlated.
- We can calculate PPS for both numerical and categorical variables and it not only shows linear relation but also quadratic or logarithmic.
- The PPS matrix shows that following attributes are highly dependent on each other:-
    - weight and result.
    - weight and family_history_with_overweight.
    - age and mode of transport
    - gender and height
    - gender and weight
    - gender and result

DATA PREPROCESSING

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert raw data into a clean data set.

Need of Data Preprocessing:-

- For achieving better results from the applied model in Machine Learning projects the format of the data must be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore, to execute random forest algorithm null values must be managed from the original raw data set, as we already checked earlier our dataset does not have any null value.
- Data is limited so models will need to perform well with limited amount of data.
- The model will need to handle dummy variables well and not be dependent on ratio data.
- The ordinal variables may or may not be helpful to the model.
- The data is wide but not very deep, so reducing the factors may be necessary.

Making New Data-Frame

```
[101]: df_prep = df.copy()
ord_feature_list= ["Gender","family_history_with_overweight",
                                        "FAVC","CAEC","SMOKE","SCC","CALC","MTRANS", "result"]
```

ENCODE LABELS

- Encode categorical features as an integer array.
- The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are converted to ordinal integers. This results in a single column of integers (0 to n_categories - 1) per feature.

```
[102]: #encoding
from sklearn.preprocessing import OrdinalEncoder
df_ord=df_prep[ord_feature_list]

enc = OrdinalEncoder()
enc.fit(df_ord)
df_ord_arr=enc.transform(df_ord)

encoded_cat_df=pd.DataFrame(df_ord_arr, columns=ord_feature_list)
encoded_cat_df
```

[102]:

	Gender	family_history_with_overweight	FAVC	CAEC	SMOKE	SCC	CALC	MTRANS	result
0	0.0	1.0	0.0	2.0	0.0	0.0	3.0	3.0	1.0
1	0.0	1.0	0.0	2.0	1.0	1.0	2.0	3.0	1.0
2	1.0	1.0	0.0	2.0	0.0	0.0	1.0	3.0	1.0
3	1.0	0.0	0.0	2.0	0.0	0.0	1.0	4.0	5.0
4	1.0	0.0	0.0	2.0	0.0	0.0	2.0	3.0	6.0
...	...	...	...	...	...	...	...	...	...
2106	0.0	1.0	1.0	2.0	0.0	0.0	2.0	3.0	4.0
2107	0.0	1.0	1.0	2.0	0.0	0.0	2.0	3.0	4.0
2108	0.0	1.0	1.0	2.0	0.0	0.0	2.0	3.0	4.0
2109	0.0	1.0	1.0	2.0	0.0	0.0	2.0	3.0	4.0
2110	0.0	1.0	1.0	2.0	0.0	0.0	2.0	3.0	4.0

2111 rows × 9 columns

CHANGES

```
[103]: train_df_final = pd.concat( [df[df.columns.difference(ord_feature_list)] , encoded_cat_df ], axis= 1)
```

- Gender column - Female is represented by 0 and male is represented by 1.
- family_history_with_overweight - yes is represented by 1 and no is represented by 0.
- Other categorical variables are changed in a similar fashion.

TRAIN - TEST SPLIT

```
[104]: # split df to X and Y
from sklearn.model_selection import train_test_split

y = train_df_final.loc[:, 'result'].values
X = train_df_final.drop('result', axis=1)

# split data into 80-20 for training set / test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, stratify = y, random_state=1)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

- Split arrays or matrices into random train and test subsets.
- Why we are doing this : We are splitting the dataset into train and test so that we can pass them through different ML algorithms which we will proceed for in the further sections. The training dataset is given to the algorithm and based on the training dataset , prediction of the test dataset takes place.

We have 1688 rows out of 21 1 1 rows as training dataset and 423 rows as test dataset.

NORMALIZE DATA

```
[105]: #normalization(make all values bet. 0-1)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_normalized_arr=scaler.transform(X_train)
X_train_normalized_df=pd.DataFrame(X_train_normalized_arr, columns=X_train.columns.to_list())

X_test_normalized_arr=scaler.transform(X_test)
X_test_normalized_df=pd.DataFrame(X_test_normalized_arr, columns=X_test.columns.to_list())
```

- Data normalization is the systematic process of grouping similar values into one common value, bringing greater context and accuracy to your marketing database. Basically, data normalization formats your data to look and read the same across all records in a database. Here all the values will be between 0 and 1.
- **Why are we doing this**- Normalization is a good technique to use when you do not know the distribution of your data or when you know the distribution is not Gaussian (a bell curve). Normalization is useful when your data has varied scales and the algorithm you are using does not make assumptions about the distribution of your data, such as k-nearest neighbors and artificial neural networks.

```
[106]: X_train_normalized_df
```

[106]:

	Age	CH2O	FAF	FCVC	Height	NCP	TUE	Weight	Gender	family_history_with_overweight	FAVC	CAEC	SMOKE	SCC	CALC	M
0	0.245550	0.206025	0.047985	1.000000	0.412658	0.666667	0.376538	0.539261	0.0	1.0	1.0	0.666667	0.0	0.0	0.5	
1	0.191489	0.500000	0.333333	1.000000	0.377358	0.000000	1.000000	0.164021	0.0	0.0	0.0	0.666667	0.0	0.0	0.5	
2	0.533806	0.502597	0.108438	0.851228	0.566038	0.666667	0.209528	0.656324	1.0	1.0	1.0	0.666667	0.0	0.0	0.5	
3	0.159611	0.675604	0.082277	1.000000	0.679002	0.666667	0.456680	0.929958	0.0	1.0	1.0	0.666667	0.0	0.0	0.5	
4	0.190119	0.501785	0.419850	1.000000	0.532655	0.666667	0.208558	0.129869	0.0	1.0	1.0	0.333333	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1683	0.234043	0.500000	0.666667	0.500000	0.226415	0.000000	0.000000	0.131217	0.0	0.0	1.0	0.666667	0.0	0.0	0.5	
1684	0.207358	0.500000	0.543971	0.268752	0.578815	0.666667	0.315711	0.642361	1.0	1.0	1.0	0.666667	0.0	0.0	0.5	
1685	0.094117	0.693195	0.514877	1.000000	0.538606	0.666667	0.450033	0.720039	0.0	1.0	1.0	0.666667	0.0	0.0	0.5	
1686	0.170139	0.328041	0.142590	1.000000	0.534458	0.921842	0.277963	0.097514	0.0	0.0	0.0	0.333333	0.0	0.0	0.5	
1687	0.148936	1.000000	0.333333	0.500000	0.320755	0.000000	0.000000	0.254233	1.0	0.0	1.0	1.000000	0.0	0.0	0.5	

1688 rows × 16 columns

```
[107]: X_test_normalized_df
```

[107]:

	Age	CH2O	FAF	FCVC	Height	NCP	TUE	Weight	Gender	family_history_with_overweight	FAVC	CAEC	SMOKE	SCC	CALC	MT
0	0.187546	0.500000	0.030781	0.500000	0.502432	0.666667	0.733248	0.300884	1.0	1.0	1.0	0.666667	0.0	0.0	0.5	
1	0.122464	0.256018	0.101241	0.814396	0.174204	0.587628	0.001615	0.203847	0.0	0.0	1.0	0.666667	0.0	1.0	0.5	
2	0.253601	0.011664	0.016977	1.000000	0.304645	0.666667	0.500000	0.517432	0.0	1.0	1.0	0.666667	0.0	0.0	0.5	
3	0.136989	0.693995	0.283572	1.000000	0.671326	0.666667	0.328246	0.936148	0.0	1.0	1.0	0.666667	0.0	0.0	0.5	
4	0.425532	1.000000	0.666667	0.500000	0.452830	0.666667	0.000000	0.369048	1.0	1.0	0.0	0.666667	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
418	0.085106	0.500000	1.000000	0.500000	0.660377	0.666667	0.000000	0.270635	1.0	1.0	1.0	0.666667	0.0	0.0	0.5	
419	0.172086	0.548375	0.181521	0.674373	0.666287	0.666667	0.338440	0.552174	1.0	1.0	1.0	0.666667	0.0	0.0	0.5	
420	0.124806	0.791082	0.666667	0.560093	0.585528	1.000000	0.755305	0.134257	1.0	1.0	1.0	0.666667	0.0	0.0	1.0	
421	0.085106	0.500000	0.666667	0.000000	0.339623	0.666667	1.000000	0.196826	0.0	1.0	1.0	0.666667	0.0	0.0	0.5	
422	0.127660	0.500000	0.333333	0.000000	0.603774	0.000000	0.500000	0.254233	1.0	1.0	1.0	0.666667	0.0	0.0	0.5	

423 rows × 16 columns

Both the training dataset and test dataset have now valued of all attributes between 0 and 1.

We are importing necessary libraries from sklearn which will help us in various classifications which we are going to perform ahead.

CROSS-VALIDATION (k-FOLD)

```
[110]: # cross-validation with 10 splits
       cv = StratifiedShuffleSplit(n_splits=10, random_state = 42, test_size=0.2)
```

[111]:
```
def display_test_scores(test, pred, prob):
    print("\n\n   TEST SCORES\n")
    print("=================================================================\n")

    #print accuracy
    accuracy = accuracy_score(test, pred)
    print("ACCURACY: {:.4f}\n".format(accuracy))
    print("\n")
    print("-----------------------------------------------------\n")


    #print confusion matrix
    print ("CONFUSION MATRIX:\n")
    conf_mat = confusion_matrix(test, pred)
    cm_df = pd.DataFrame(conf_mat)

    #draw cm

    sns.heatmap(cm_df, annot=True)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label',)
    plt.show()
    print("\n")
    print("\n")
    print("-----------------------------------------------------\n")

    #print FP, FN
    print("FALSE POSITIVES:\n")
    fp = conf_mat[1][0]
    pos_labels = conf_mat[1][0]+conf_mat[1][1]
    print("{} out of {} positive labels ({:.4f}%)\n".format(fp, pos_labels,fp/pos_labels))
    print("\n")
    print("-----------------------------------------------------\n")

    print("FALSE NEGATIVES:\n")
    fn = conf_mat[0][1]
    neg_labels = conf_mat[0][1]+conf_mat[0][0]
    print("{} out of {} negative labels ({:.4f}%)\n".format(fn, neg_labels, fn/neg_labels))
    print("\n")
    print("-----------------------------------------------------\n")

    #roc
    roc = roc_auc_score(test, prob, multi_class="ovr", average="macro")
    print("ROC-AUC score:({:.4f}%)\n".format(roc))
    print("\n")
```

```
    skplt.metrics.plot_roc(pred, prob)
    plt.show()

    print("-----------------------------------------------------\n")


    #print classification report
    print("PRECISION, RECALL, F1 scores:\n\n")
    print("{}".format(classification_report(test, pred)))

    false_indexes = np.where(test != pred)
    return false_indexes
```

- Randomly partition the data into k mutually exclusive subsets, each approximately equal size
- At i-th iteration, use Di as test set and others as training set.

MACHINE LEARING

CLASSIFICATION

[112]:
```
X_train= X_train_normalized_df
X_test= X_test_normalized_df
```

[113]:
```
# decesion tree
# Naive bayes
# SVM
# KNN
# Logistic Regression
# Random Forest
```

---

1. DECISION TREE

Decision Tree is also the most used approach that is used for supervised learning.

One of the benefits of Decision Tree is that it can solve both regression and classification tasks.

- **Root node**:- It is the starting/initial node of the tree. It represents the entire sample, and it further gets split into child nodes(interior nodes)
- **Interior nodes**:-the interior nodes represent the features of the dataset, and its branches represent the decision rules that need to be taken.
- **Leaf Nodes**:- It is used to represent the outcome of the regression classification model.

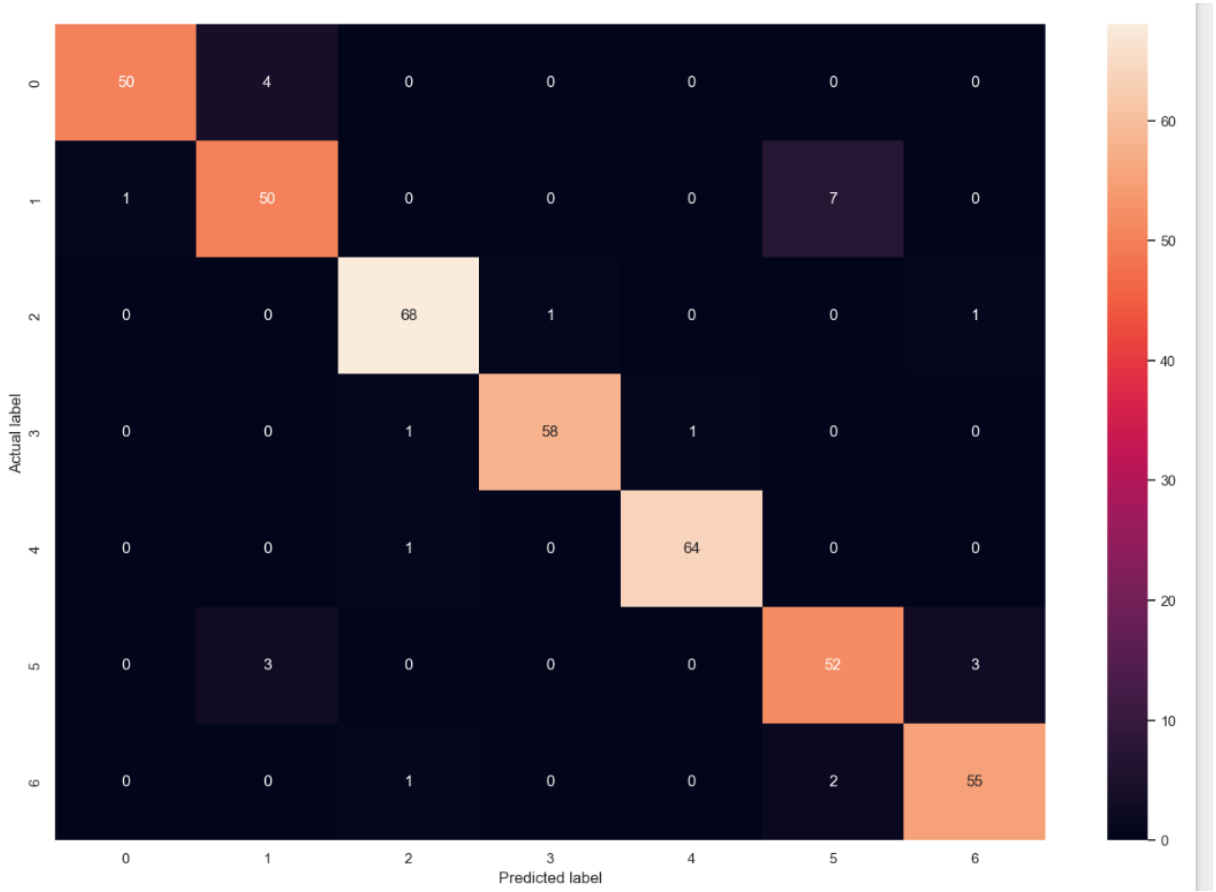
[114]:
```
#parameters
param_grid = {'criterion':['gini', 'entropy'],
              "class_weight": [None, "balanced"],
              }

#Decision Tree
grid_1 = GridSearchCV(DecisionTreeClassifier(random_state=0), param_grid, verbose=False, n_jobs= -1, cv= cv)
grid_1.fit(X_train, y_train)

#print best scores
print("The best parameters are %s with a score of %0.4f"
      % (grid_1.best_params_, grid_1.best_score_))

#prediction results
y_pred = grid_1.predict(X_test)
y_prob = grid_1.predict_proba(X_test)
display_test_scores(y_test, y_pred, y_prob)
```

The best parameters are { ' class _weight' : ' balanced' , ' criterion' 'entropy' } with a score of 0.9470

TEST SCORES

ACCURACY : 0.9385

CONFUSION MATRIX:



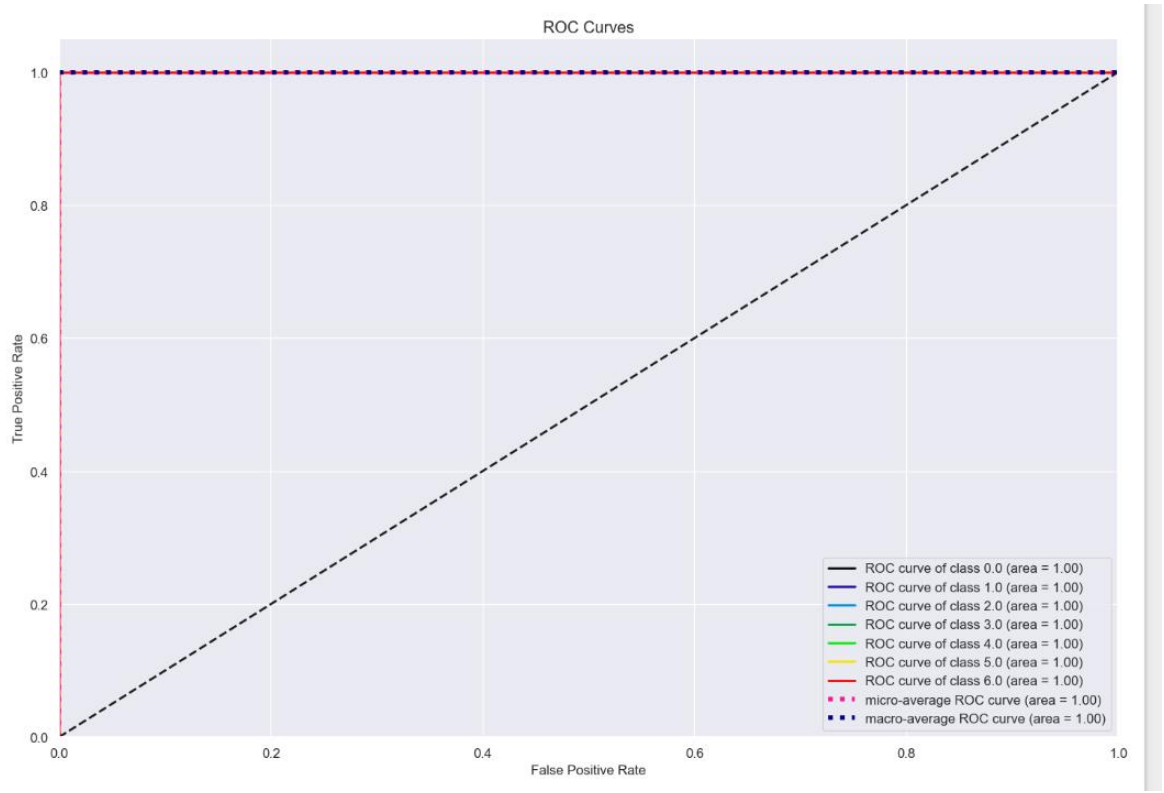
FALSE POSITIVES :

1 out of 51 positive labels (0.0196%)

FALSE NEGATIVES :

4 out of 54 negative labels (0.0741%)

ROC-AUC score : (0.9631%)



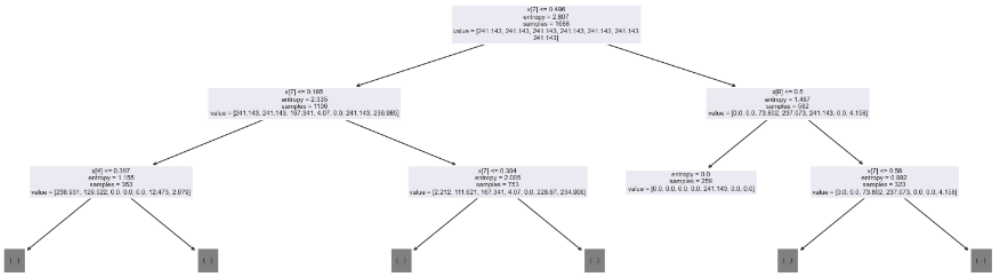
PRECISION, RECALL, F1 scores:

	precision	recall	f1-score	support
0.0	0.98	0.93	0.95	54
1.0	0.88	0.86	0.87	58
2.0	0.96	0.97	0.96	70
3.0	0.98	0.97	0.97	60
4.0	0.98	0.98	0.98	65
5.0	0.85	0.90	0.87	58
6.0	0.93	0.95	0.94	58
accuracy			0.94	423
macro avg	0.94	0.94	0.94	423
weighted avg	0.94	0.94	0.94	423

Inference

- In decision tree we use entropy as a parameter to infer the best result here the entropy or information gain is the best suited value for calculating the prediction with accuracy.
- The attribute with the highest information gain will produce the best split as it's doing the best job at classifying the training data according to its target classification the attribute x[7] value as shown in the graph that is Weight is that attribute.

```
[115]: plt.figure(figsize = (22, 6))
       plot_tree(grid_1.best_estimator_, max_depth= 2)
       plt.show()
```



2. NAIVE-BAYES

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

- **Bayes Theorem**--Bayes Theorem finds the probability of an event occurring given the probability of another event that has already occurred.
- **Gaussian Naive Bayes**-In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution.
- A Gaussian distribution is also called **Normal distribution**. When plotted, it gives a bell-shaped curve which is symmetric about the mean of the feature values

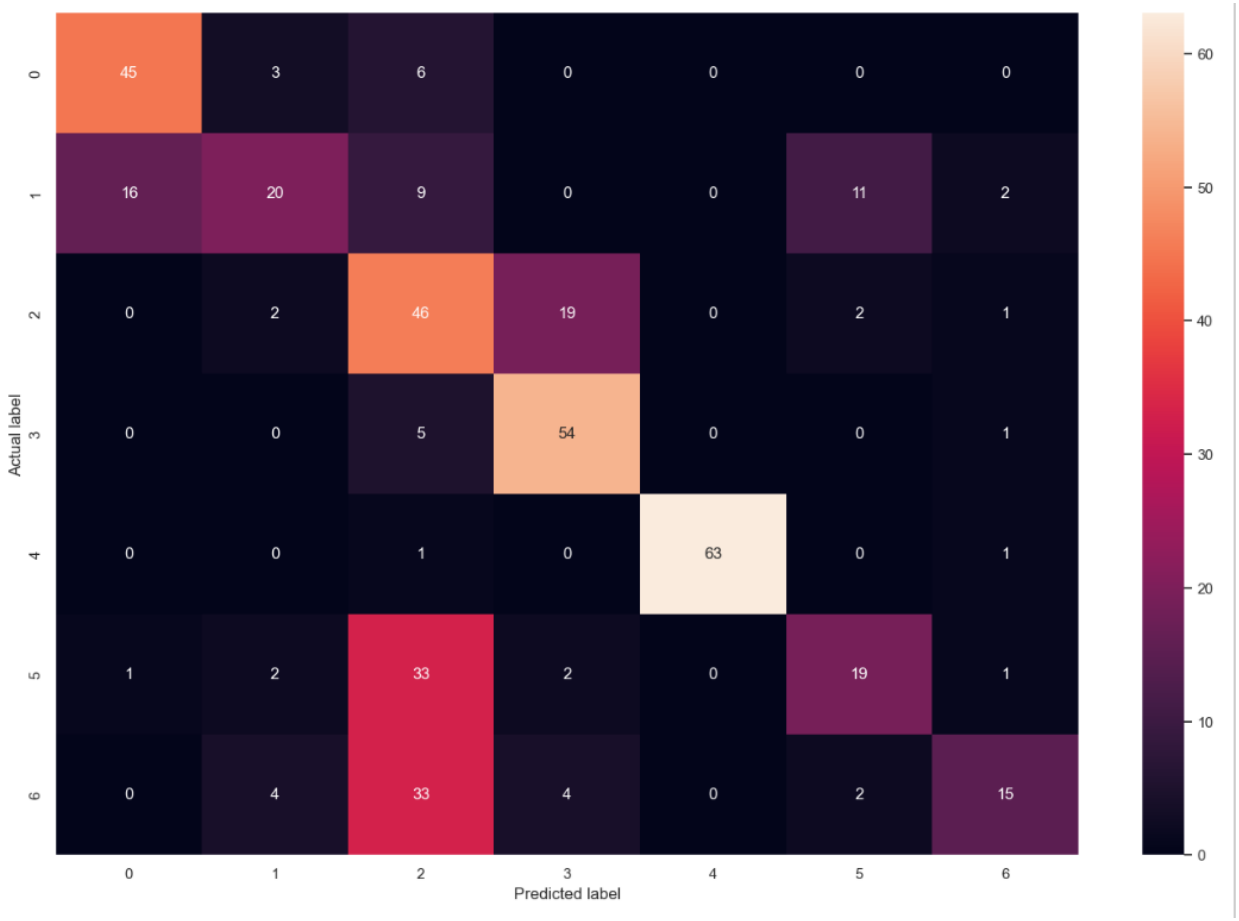
[116]:
```
# Naive-Bayes with different approaches
nb_list = [ GaussianNB(), MultinomialNB(), ComplementNB()]

for nb in nb_list:
    print("\n\n-----------", str(nb), "--------------")

    # parameters
    parameters = {}

    # grid search for parameters
    grid_2 = GridSearchCV(estimator=nb, param_grid=parameters, cv=cv, n_jobs=-1)
    grid_2.fit(X_train, y_train)

    # print best scores
    print("The best parameters are %s with a score of %0.4f\n"
          % (grid_2.best_params_, grid_2.best_score_))

    # prediction results
    y_pred = grid_2.predict(X_test)
    y_prob = grid_2.predict_proba(X_test)

    # print accuracy metrics
    display_test_scores(y_test, y_pred, y_prob)
```

The best parameters are { } with a score of 0.5754

TEST SCORES

ACCURACY : 0.6194

CONFUSION MATRIX:



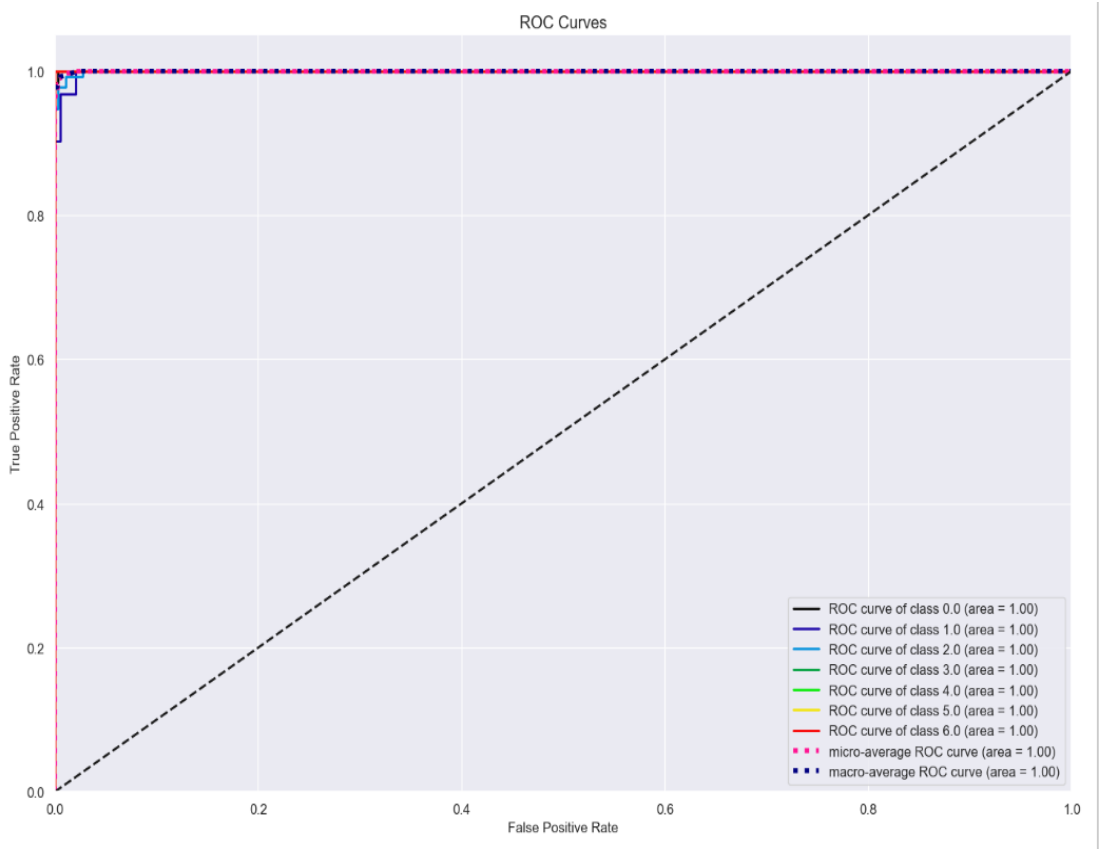
FALSE POSITIVES:

16 out of 36 positive labels (0.4444%)

FALSE NEGATIVES:

3 out of 48 negative labels (0.0625%)

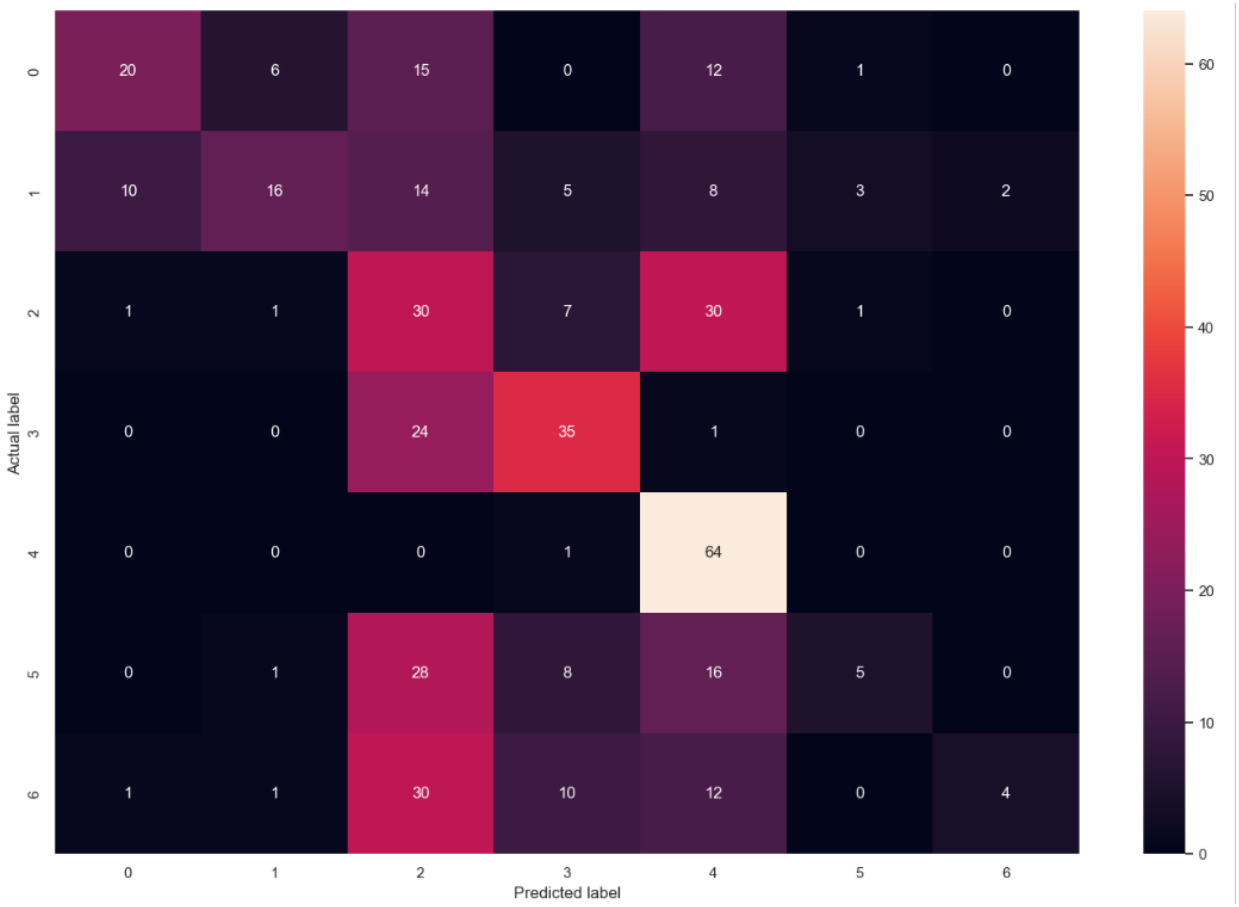
ROC-AUC score: (0.9236%)



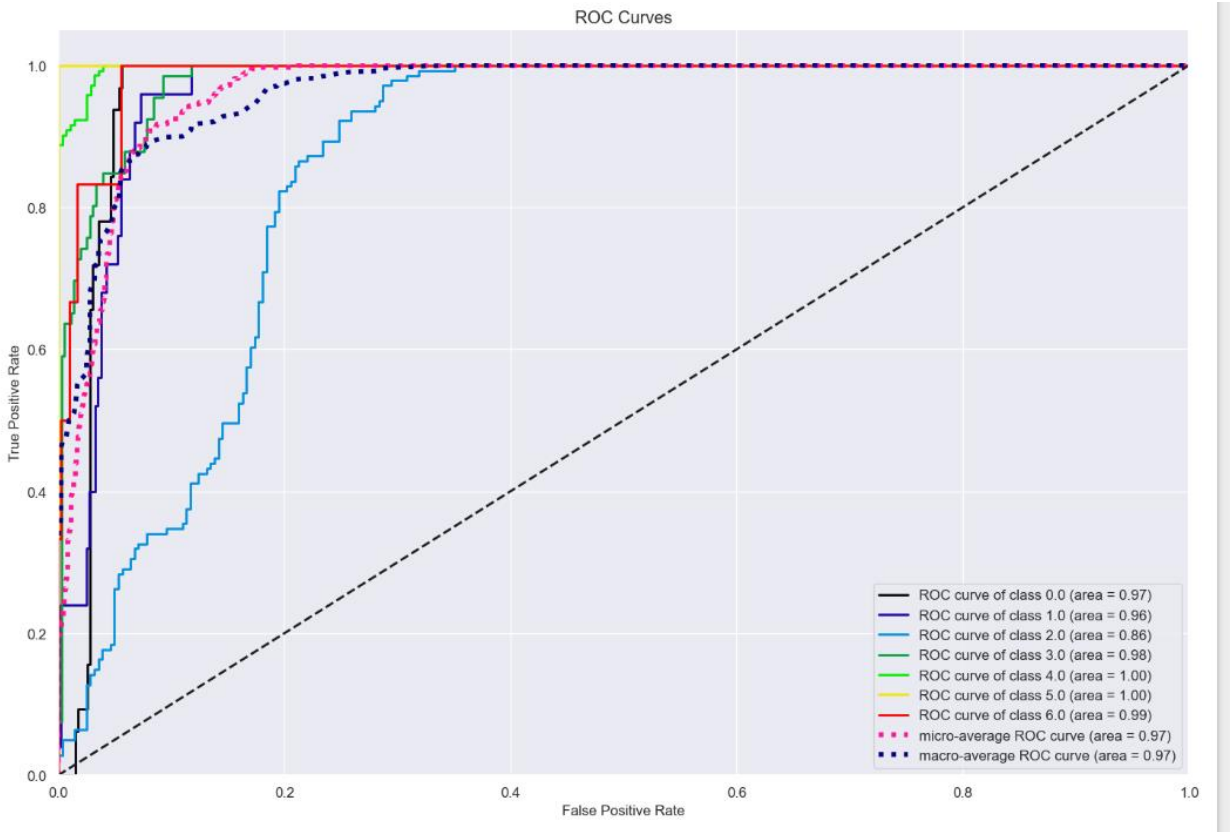
PRECISION, RECALL, F1 scores:

	precision	recall	f1-score	support
0.0	0.73	0.83	0.78	54
1.0	0.65	0.34	0.45	58
2.0	0.35	0.66	0.45	70
3.0	0.68	0.90	0.78	60
4.0	1.00	0.97	0.98	65
5.0	0.56	0.33	0.41	58
6.0	0.71	0.26	0.38	58
accuracy			0.62	423
macro avg	0.67	0.61	0.60	423
weighted avg	0.66	0.62	0.61	423

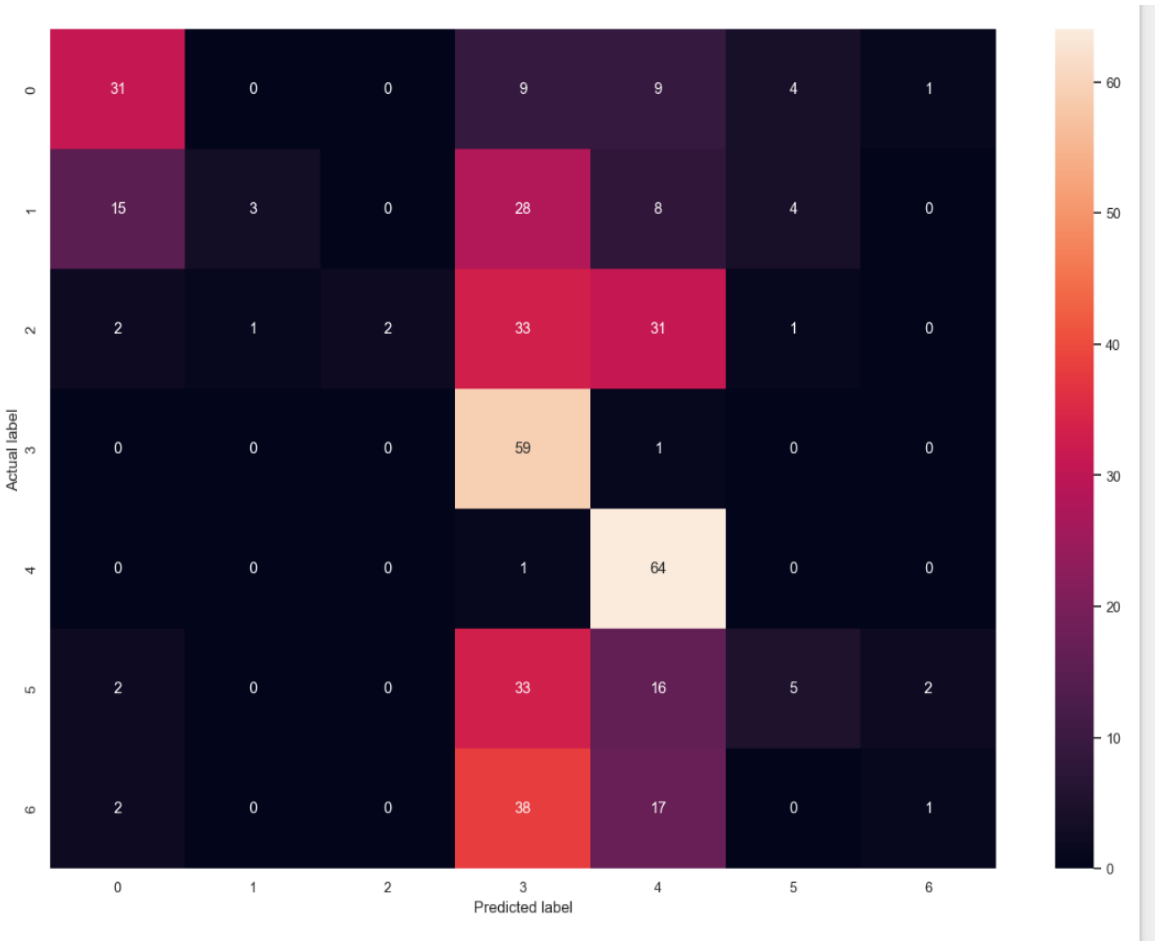
```
---------- MultinomialNB() --------------
The best parameters are {} with a score of 0.4071


   TEST SCORES

=========================================================================

ACCURACY: 0.4113


---------------------------------------------------

CONFUSION MATRIX:
```



```
-----------------------------------------------------

FALSE POSITIVES:

10 out of 26 positive labels (0.3846%)


-----------------------------------------------------

FALSE NEGATIVES:

6 out of 26 negative labels (0.2308%)


-----------------------------------------------------

ROC-AUC score:(0.8253%)
```



```
-------------------------------------------------

PRECISION, RECALL, F1 scores:

              precision    recall  f1-score   support

         0.0       0.62      0.37      0.47        54
         1.0       0.64      0.28      0.39        58
         2.0       0.21      0.43      0.28        70
         3.0       0.53      0.58      0.56        60
         4.0       0.45      0.98      0.62        65
         5.0       0.50      0.09      0.15        58
         6.0       0.67      0.07      0.12        58

    accuracy                           0.41       423
   macro avg       0.52      0.40      0.37       423
weighted avg       0.51      0.41      0.37       423


---------- ComplementNB() --------------
The best parameters are {} with a score of 0.3899


   TEST SCORES

=========================================================================

ACCURACY: 0.3901


-------------------------------------------------

CONFUSION MATRIX:
```



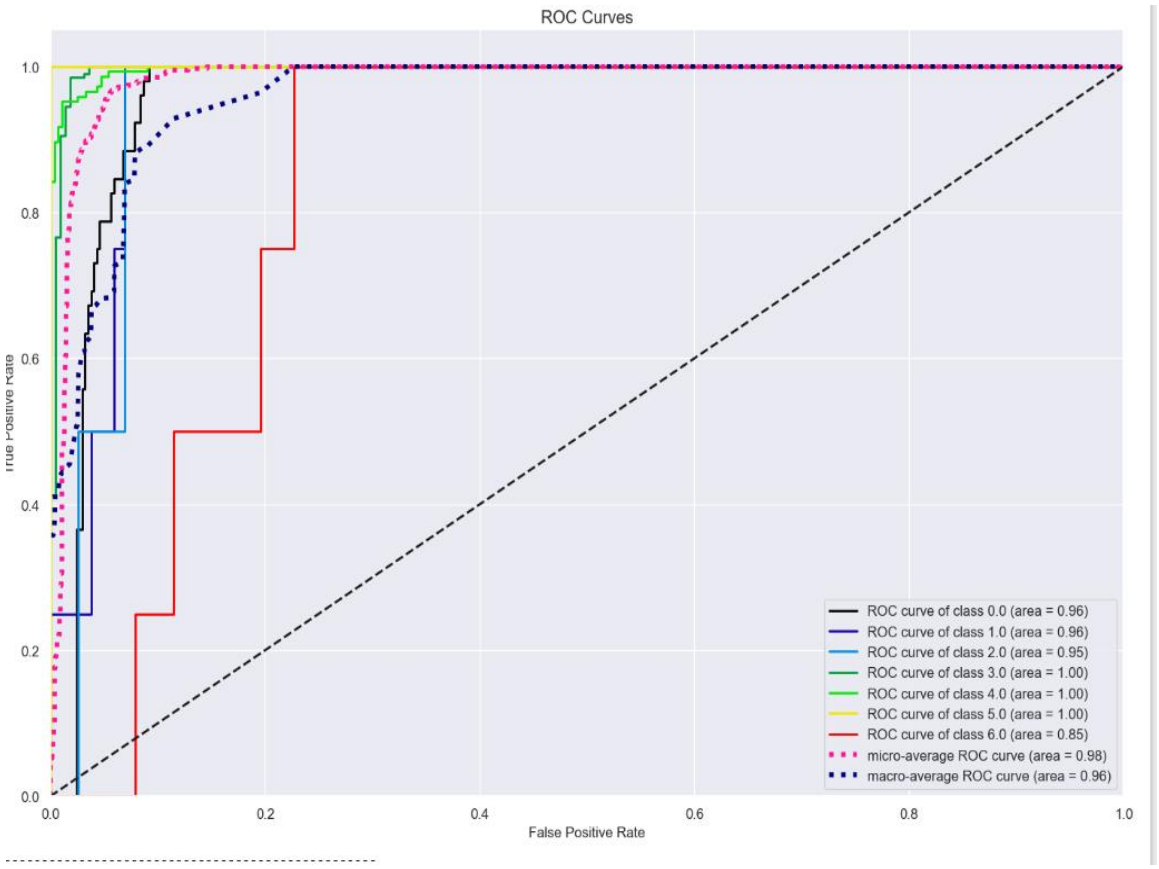
FALSE POSITIVES:

15 out of 18 positive labels (0.8333%)

FALSE NEGATIVES:

0 out of 31 negative labels (0.0000%)

ROC-AUC score:(0.8287%)



PRECISION, RECALL, F1 scores:

	precision	recall	f1-score	support
0.0	0.60	0.57	0.58	54
1.0	0.75	0.05	0.10	58
2.0	1.00	0.03	0.06	70
3.0	0.29	0.98	0.45	60
4.0	0.44	0.98	0.61	65
5.0	0.36	0.09	0.14	58
6.0	0.25	0.02	0.03	58
accuracy			0.39	423
macro avg	0.53	0.39	0.28	423
weighted avg	0.54	0.39	0.28	423

INFERENCE

Using the dataset we have calculated Gaussian, Multinomial are complemented and it can be clearly seen that the best accuracy was given by gaussian naive bayes.

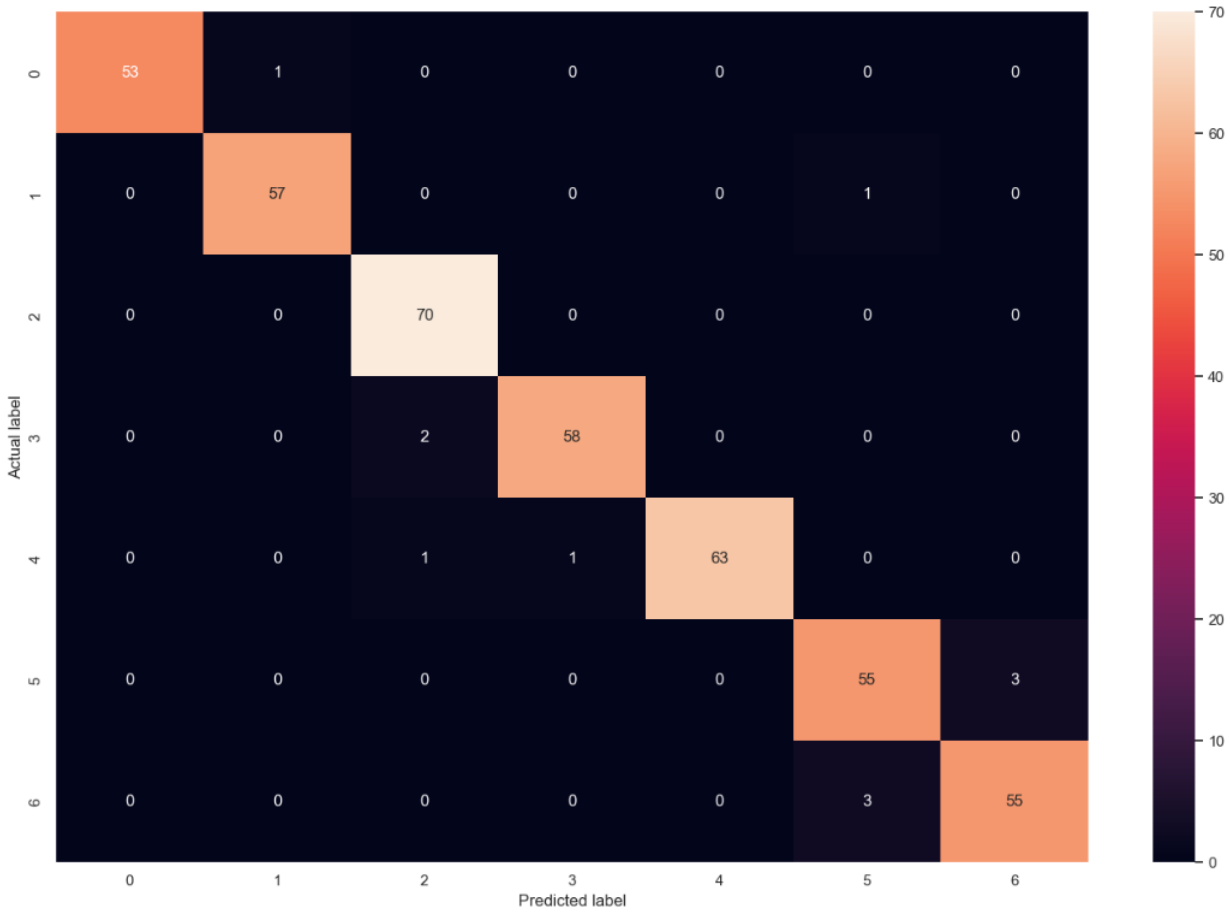
Biggest disadvantage is that the requirement of predictors to be independent, but here the contrary example is Weight. This is the reason for low score.

3. Support Vector Machine

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three. The best hyperplane is the one that represents the largest separation or margin between the two classes. So, we choose the hyperplane whose distance from it to the nearest data point on each side is maximized.

[117]:
```
# SVM classifier
svm = SVC(tol=1e-5, random_state=0, probability=True)

# parameters
parameters = {
                'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
                'C': [0.01, 0.1, 1, 10, 100],
            }

# grid search for parameters
grid_3 = GridSearchCV(estimator=svm, param_grid=parameters, cv=cv, n_jobs=-1)
grid_3.fit(X_train, y_train)

# print best scores
print("The best parameters are %s with a score of %0.4f\n"
      % (grid_3.best_params_, grid_3.best_score_))

# prediction results
y_pred = grid_3.predict(X_test)
y_prob = grid_3.predict_proba(X_test)

# print accuracy metrics
display_test_scores(y_test, y_pred, y_prob)
```

```
   TEST SCORES

==========================================================================

ACCURACY: 0.9716


------------------------------------------------------

CONFUSION MATRIX:
```



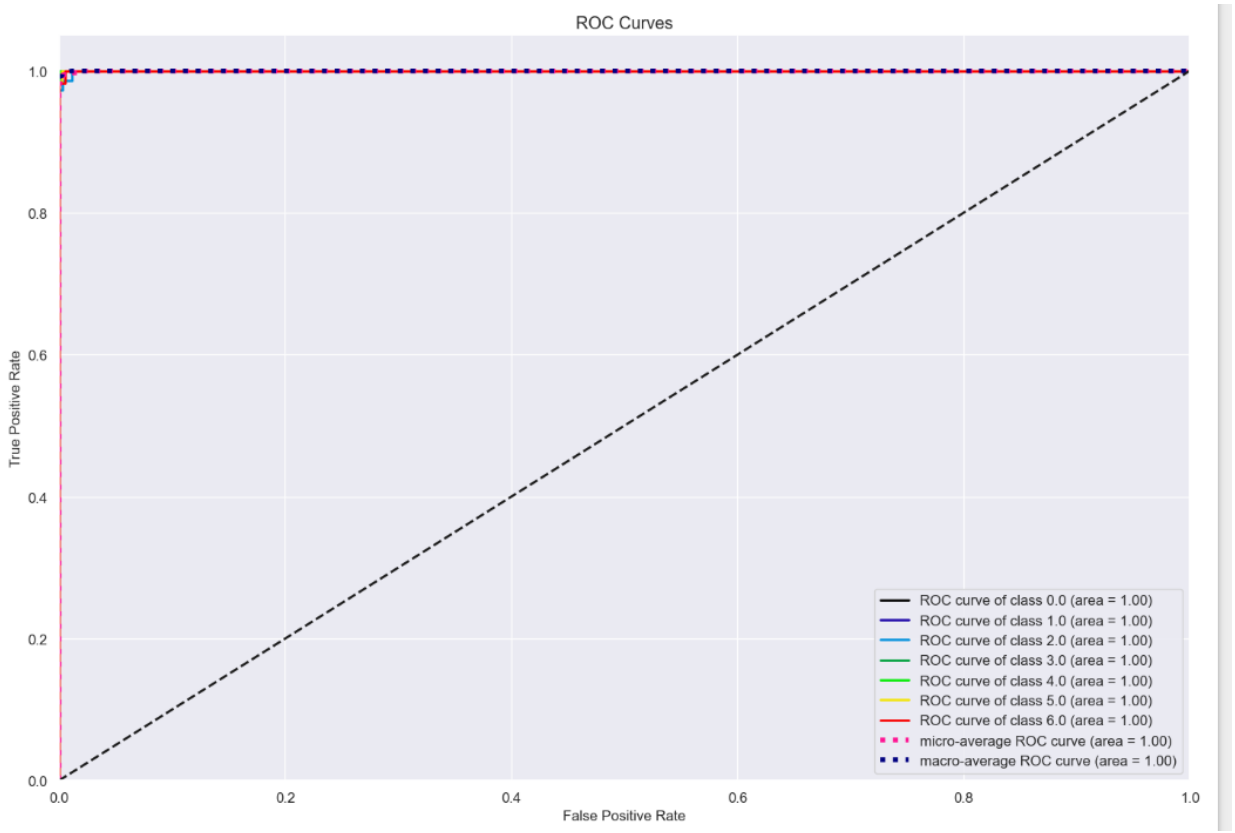
FALSE POSITIVES:

0 out of 57 positive labels (0.0000%)

FALSE NEGATIVES:

1 out of 54 negative labels (0.0185%)

ROC-AUC score:(0.9981%)



PRECISION, RECALL, F1 scores:

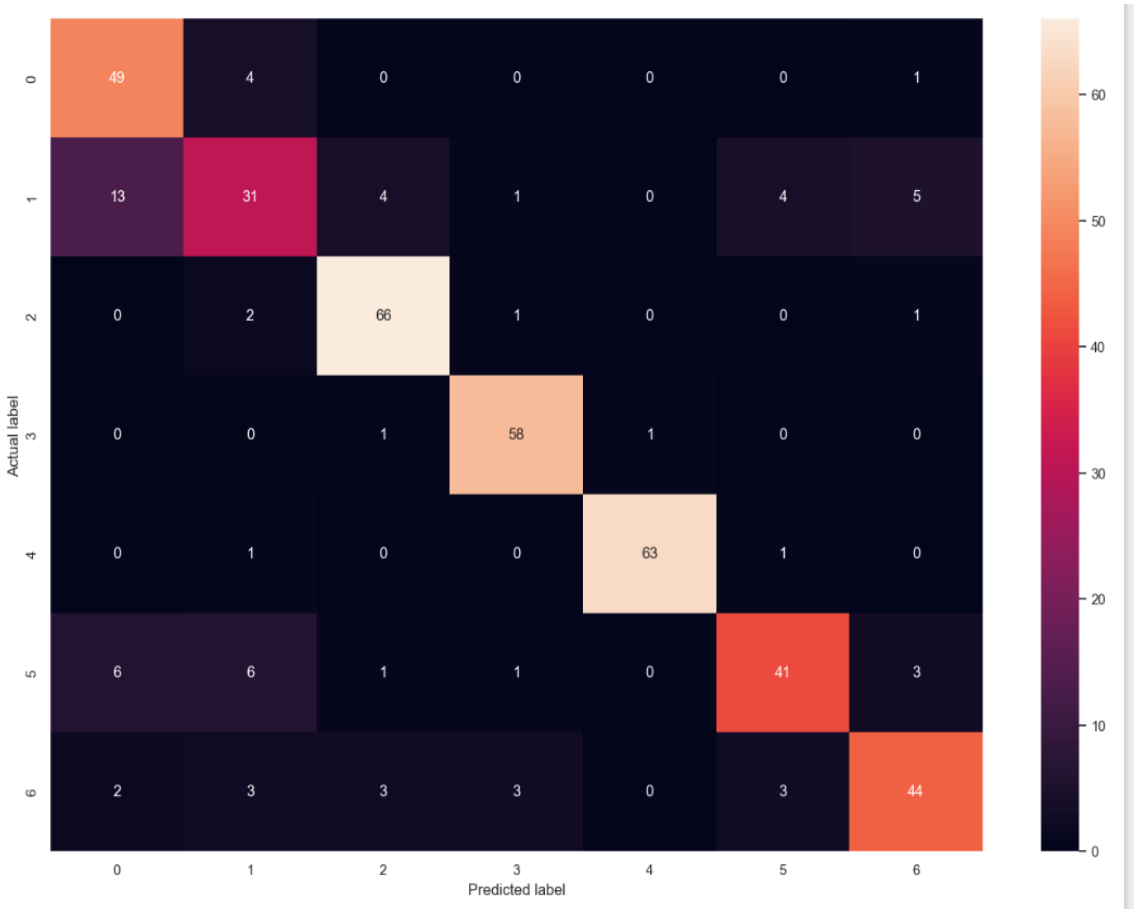
	precision	recall	f1-score	support
0.0	1.00	0.98	0.99	54
1.0	0.98	0.98	0.98	58
2.0	0.96	1.00	0.98	70
3.0	0.98	0.97	0.97	60
4.0	1.00	0.97	0.98	65
5.0	0.93	0.95	0.94	58
6.0	0.95	0.95	0.95	58
accuracy			0.97	423
macro avg	0.97	0.97	0.97	423
weighted avg	0.97	0.97	0.97	423

4. K-NEAREST NEAIGHBOURS

The k-nearest neighbors' classifier (KNN) is a non-parametric supervised machine learning algorithm. It's distance-based: it classifies objects based on their proximate neighbors' classes. The k-nearest neighbors' algorithm is also non-linear. In contrast to simpler models like linear regression, it will work well with data in which the relationship between the independent variable (x) and the dependent variable (y) is not a straight line.

[118]:
```
knn = KNeighborsClassifier()
# parameters
parameters = {
                "algorithm": ["auto", "ball_tree", "kd_tree", "brute"],
                "n_neighbors": list(range(0, 25)),
                "metric": ['euclidean', 'manhattan', 'chebyshev'],
                "p": [1,2],
    }

# grid search for parameters
grid_4 = GridSearchCV(estimator=knn, param_grid=parameters, cv=cv, n_jobs=-1)
grid_4.fit(X_train, y_train)

# print best scores
print("The best parameters are %s with a score of %0.4f\n"
      % (grid_4.best_params_, grid_4.best_score_))

# prediction results
y_pred = grid_4.predict(X_test)
y_prob = grid_4.predict_proba(X_test)

# print accuracy metrics
display_test_scores(y_test, y_pred, y_prob)
```

```
TEST SCORES

==================================================================

ACCURACY: 0.8322



--------------------------------------------------

CONFUSION MATRIX:
```



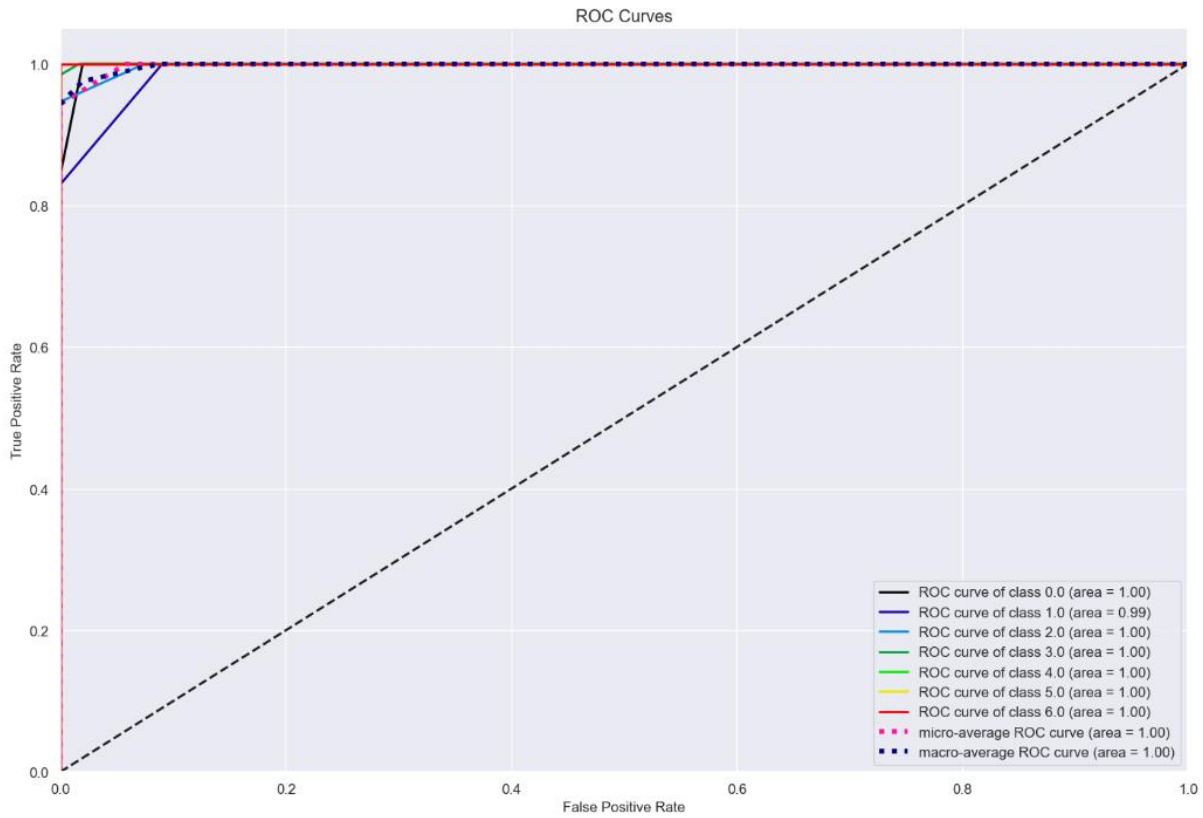
FALSE POSITIVES:

13 out of 44 positive labels (0.2955%)

FALSE NEGATIVES:

4 out of 53 negative labels (0.0755%)

ROC-AUC score:(0.9478%)



PRECISION, RECALL, F1 scores:

	precision	recall	f1-score	support
0.0	0.70	0.91	0.79	54
1.0	0.66	0.53	0.59	58
2.0	0.88	0.94	0.91	70
3.0	0.91	0.97	0.94	60
4.0	0.98	0.97	0.98	65
5.0	0.84	0.71	0.77	58
6.0	0.81	0.76	0.79	58
accuracy			0.83	423
macro avg	0.83	0.83	0.82	423
weighted avg	0.83	0.83	0.83	423

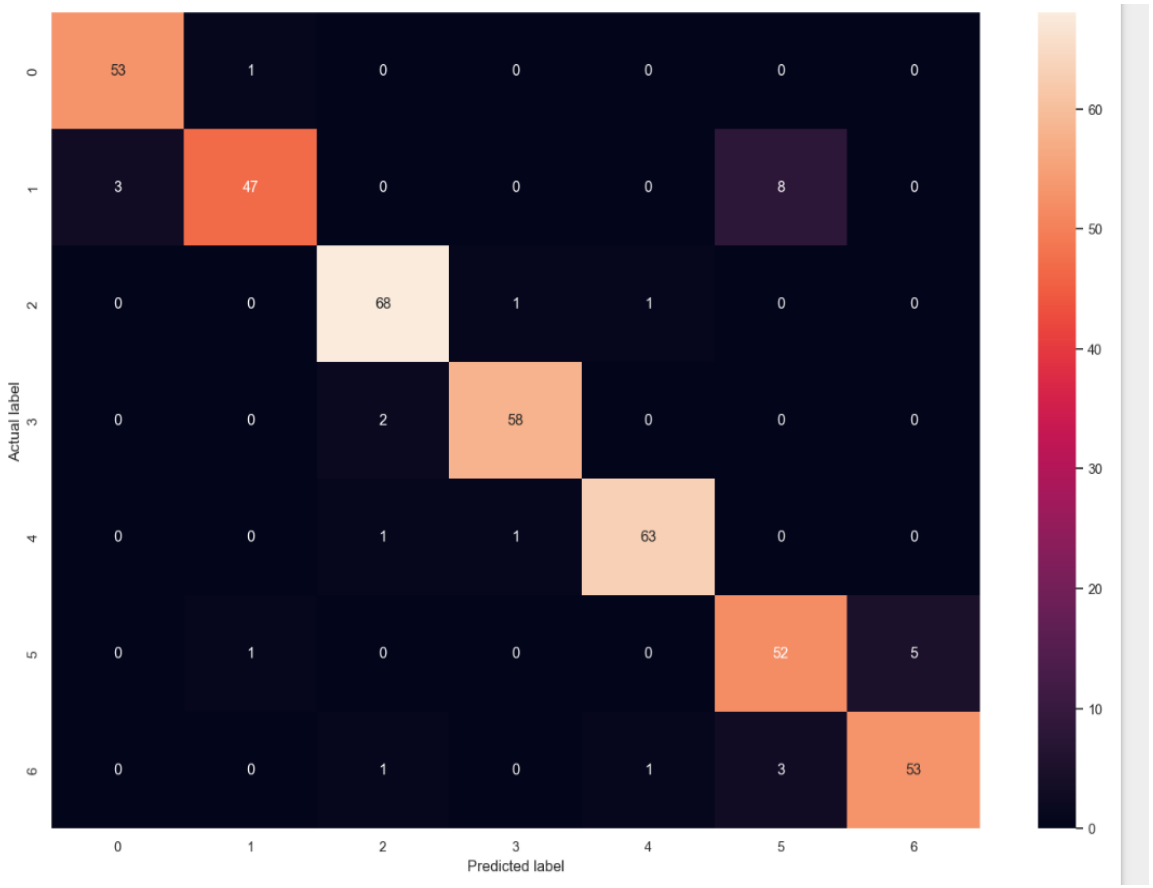
```
]: (array([  0,  10,  20,  23,  25,  32,  33,  40,  49,  55,  56,  57,  63,
         65,  69,  75,  82,  84,  89,  91,  98, 103, 107, 109, 113, 122,
        123, 125, 126, 128, 152, 173, 183, 184, 186, 189, 193, 209, 214,
        223, 232, 244, 251, 264, 270, 279, 286, 292, 295, 304, 306, 309,
        315, 320, 328, 330, 335, 351, 354, 364, 366, 370, 372, 383, 384,
        393, 399, 409, 418, 421, 422], dtype=int64),)
```

5. LOGISTIC REGRESSION

Logistic regression is used to obtain odds ratio in the presence of more than one explanatory variable. The procedure is quite like multiple linear regression, with the exception that the response variable is binomial. The result is the impact of each variable on the odds ratio of the observed event of interest. The main advantage is to avoid confounding effects by analyzing the association of all variables together.

[119]:
```
logit = LogisticRegression(solver='lbfgs', random_state=0)
# parameters
parameters = {
                "penalty":['l1', 'l2'],
                "C": [0.01, 0.1, 1, 10, 100],
                "max_iter": [250],
                "penalty":["l1","l2"]
             }

# grid search for parameters
grid_5 = GridSearchCV(estimator=logit, param_grid=parameters, cv=cv, n_jobs=-1)
grid_5.fit(X_train, y_train)

# print best scores
print("The best parameters are %s with a score of %0.4f\n"
      % (grid_5.best_params_, grid_5.best_score_))

# prediction results
y_pred = grid_5.predict(X_test)
y_prob = grid_5.predict_proba(X_test)

# print accuracy metrics
display_test_scores(y_test, y_pred, y_prob)
```

```
   TEST SCORES

======================================================================

ACCURACY: 0.9314


------------------------------------------------------

CONFUSION MATRIX:
```



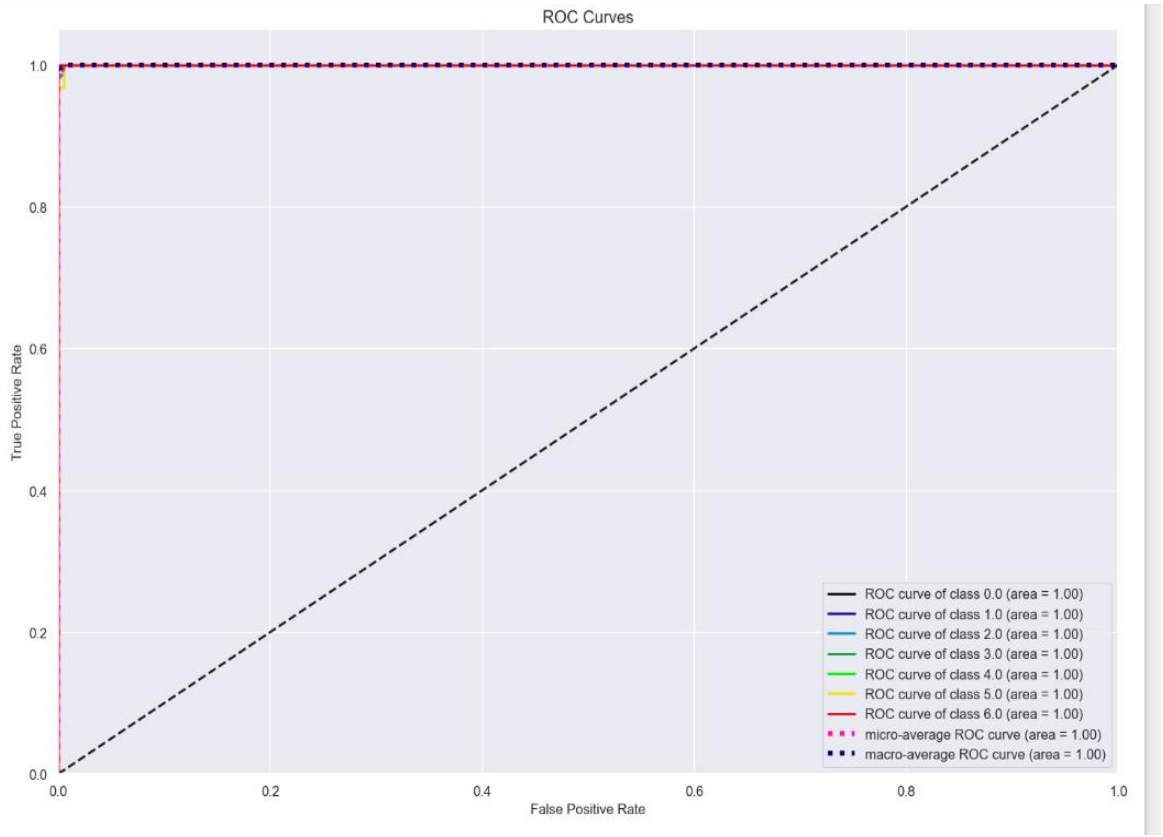
FALSE POSITIVES:

3 out of 50 positive labels (0.0600%)

FALSE NEGATIVES:

1 out of 54 negative labels (0.0185%)

ROC-AUC score:(0.9929%)



PRECISION, RECALL, F1 scores:

	precision	recall	f1-score	support
0.0	0.95	0.98	0.96	54
1.0	0.96	0.81	0.88	58
2.0	0.94	0.97	0.96	70
3.0	0.97	0.97	0.97	60
4.0	0.97	0.97	0.97	65
5.0	0.83	0.90	0.86	58
6.0	0.91	0.91	0.91	58
accuracy			0.93	423
macro avg	0.93	0.93	0.93	423
weighted avg	0.93	0.93	0.93	423

INFERENCE

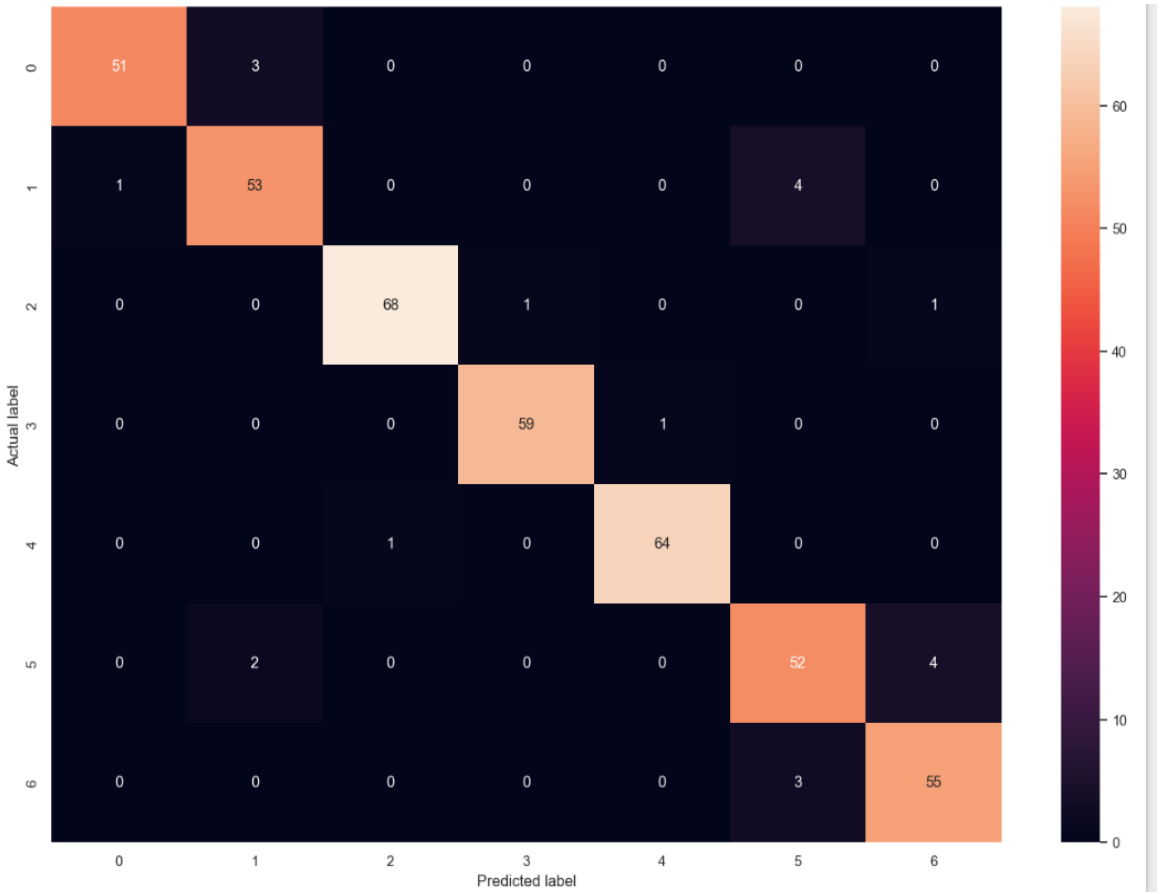
- From this model we can infer that it can be used for multiclass classifications also therefore this model was the one with the closest prediction, and Logistic Regression can derive confidence level (about its prediction), whereas KNN can only output the labels.

6. RANDOM FOREST

The random forest is a classification algorithm consisting of many decision trees. It uses bagging and features randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

[120]:
```
rf = RandomForestClassifier(random_state=0)

# parameters
parameters = {
                "bootstrap": [True,False],
                "max_features": [None, "sqrt", "log2"],
                "class_weight": [None, "balanced", "balanced_subsample"],
                "max_samples": [None, 0.3, 0.5, 0.7, 0.9],
                "n_estimators": [10, 100, 200],
                'criterion':['gini','entropy'],

}

# grid search for parameters
grid_6 = GridSearchCV(estimator=rf, param_grid=parameters, cv=cv, n_jobs=-1)
grid_6.fit(X_train, y_train)

# print best scores
print("The best parameters are %s with a score of %0.4f"
      % (grid_6.best_params_, grid_6.best_score_))

# prediction results
y_pred = grid_6.predict(X_test)
y_prob = grid_6.predict_proba(X_test)

# print accuracy metrics
display_test_scores(y_test, y_pred, y_prob)
```

```
The best parameters are {'bootstrap': True, 'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'max_features': None, 'max_samples': None, 'n_estimators': 200} with a score of 0.9624


   TEST SCORES

================================================================

ACCURACY: 0.9504


----------------------------------------------------

CONFUSION MATRIX:
```



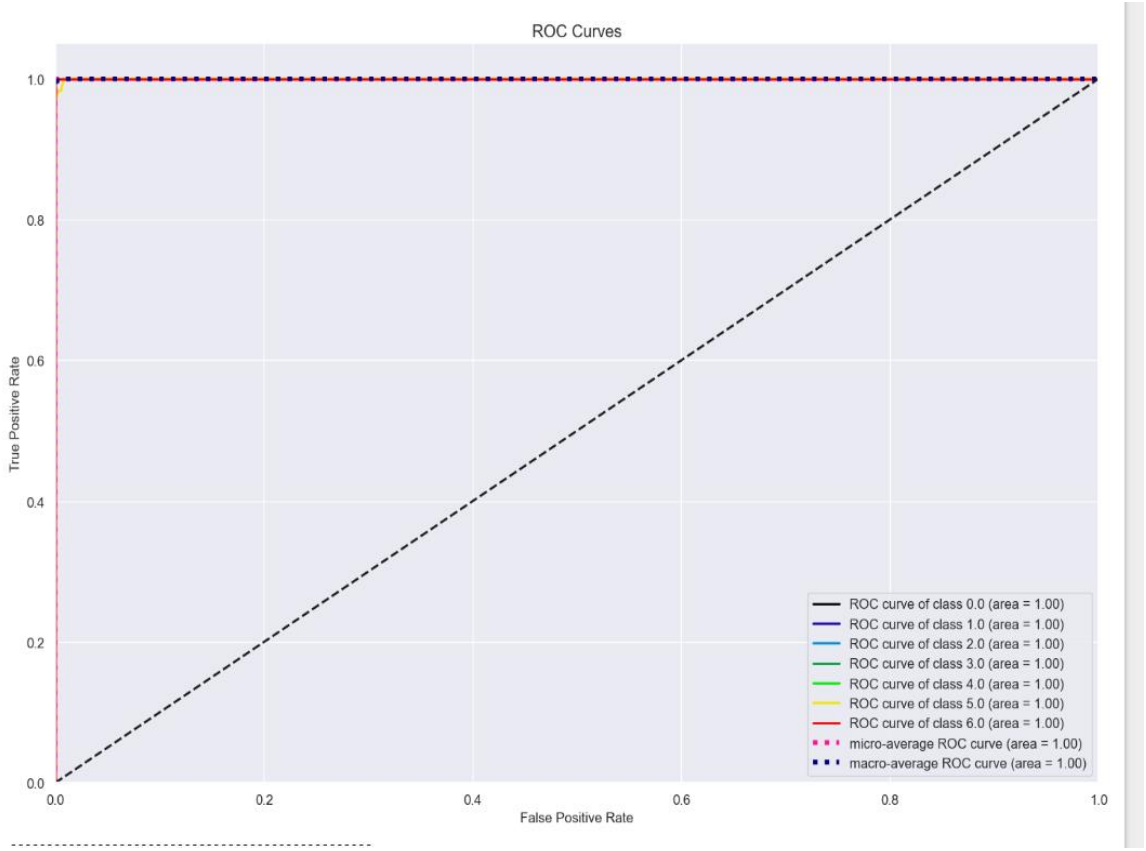
FALSE POSITIVES:

1 out of 54 positive labels (0.0185%)

FALSE NEGATIVES:

3 out of 54 negative labels (0.0556%)

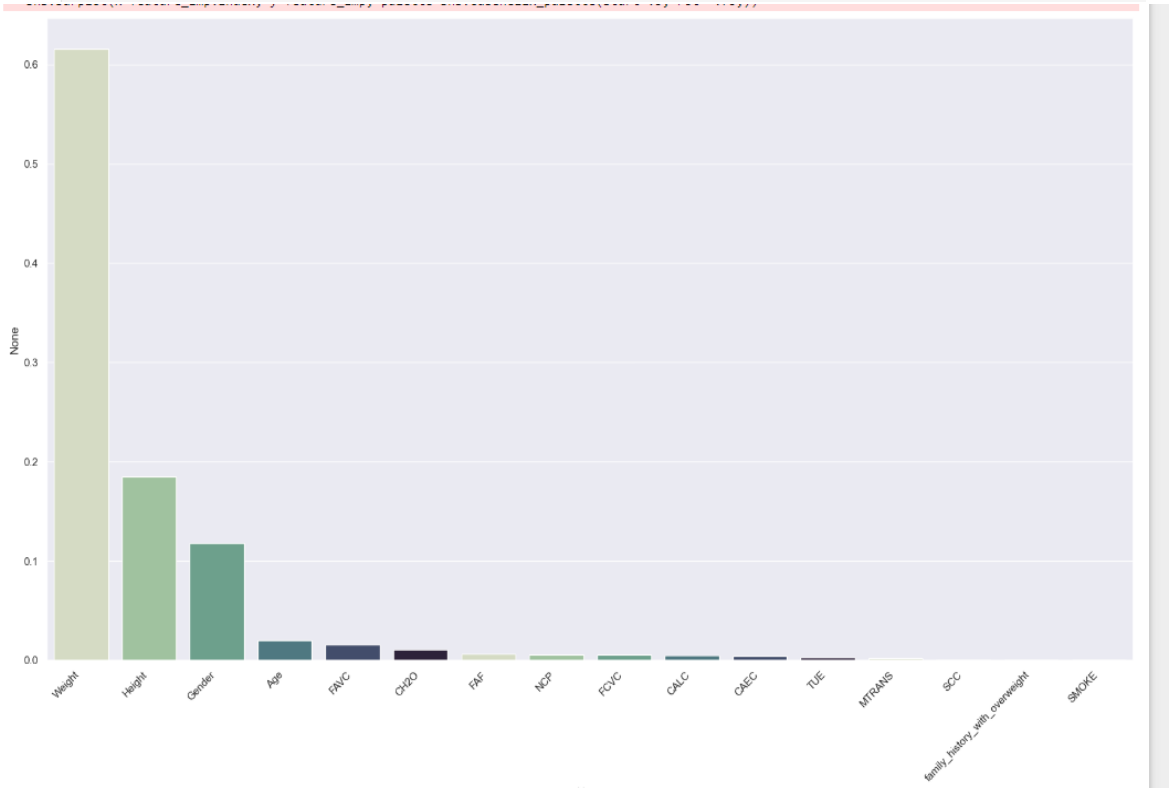
ROC-AUC score:(0.9961%)



PRECISION, RECALL, F1 scores:

	precision	recall	f1-score	support
0.0	0.98	0.94	0.96	54
1.0	0.91	0.91	0.91	58
2.0	0.99	0.97	0.98	70
3.0	0.98	0.98	0.98	60
4.0	0.98	0.98	0.98	65
5.0	0.88	0.90	0.89	58
6.0	0.92	0.95	0.93	58
accuracy			0.95	423
macro avg	0.95	0.95	0.95	423
weighted avg	0.95	0.95	0.95	423

```
[121]: feature_imp = pd.Series(grid_6.best_estimator_.feature_importances_, index=X_train.columns).sort_values(ascending=False)

       # Creating a bar plot
       sns.barplot(x=feature_imp.index, y=feature_imp, palette=sns.cubehelix_palette(start=.5, rot=-.75,))

       # Add labels to your graph
       plt.xticks(rotation=45, horizontalalignment='right')

       plt.tight_layout()
       plt.show()
```



Inference

• From this model we can infer that our prediction is most dependent on weight followed by height and gender.

```
[122]: import pandas as pd

data = {'Accuracy': [0.9385,0.6194,0.9716,0.8322,0.9314,0.9504],
        'ROC_AUC': [0.9631, 0.9236, 0.9981, 0.9478, 0.9931,0.9961],
        'Precision':[0.94,0.66,0.97,0.83,0.93,0.95],
        'Recall':[0.94,0.62,0.97,0.83,0.93,0.95],
        'F1-Score':[0.94,0.61,0.97,0.83,0.93,0.95]};



df = pd.DataFrame(data, index=['Decision Tree', 'Naive Bayes(Gaussian)', 'SVM', 'k-Nearest Neighbour', 'Logitic Regression','Random Forest classifier'])

print(df)
```

	Accuracy	ROC_AUC	Precision	Recall	F1-Score
Decision Tree	0.9385	0.9631	0.94	0.94	0.94
Naive Bayes(Gaussian)	0.6194	0.9236	0.66	0.62	0.61
SVM	0.9716	0.9981	0.97	0.97	0.97
k-Nearest Neighbour	0.8322	0.9478	0.83	0.83	0.83
Logitic Regression	0.9314	0.9931	0.93	0.93	0.93
Random Forest classifier	0.9504	0.9961	0.95	0.95	0.95

CONCLUSION

Our result is most dependent on weight i.e. it is the most significant attribute, which we inferred from Random Forest Classifier.

Out of all the classifications, Support Vector Machine is the most accurate whereas Naive Bayes is the least accurate. We have classified the data with best accuracy of 97.17%