Project Title: “Estimation of Obesity Levels Based On Eating Habits and Physical Condition”

Teja Akula

importing libraries

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import pandas as pd
warnings.filterwarnings('ignore')
```

Importing data

```
df=pd.read_csv("obesity.csv")
df.head(10)
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	
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_
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_
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_
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	
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_
5	Male	29.0	1.62	53.0	no	yes	2.0	3.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	
6	Female	23.0	1.50	55.0	yes	yes	3.0	3.0	Sometimes	no	2.0	no	1.0	0.0	Sometimes	
7	Male	22.0	1.64	53.0	no	no	2.0	3.0	Sometimes	no	2.0	no	3.0	0.0	Sometimes	Public_
8	Male	24.0	1.78	64.0	yes	yes	3.0	3.0	Sometimes	no	2.0	no	1.0	1.0	Frequently	Public_
9	Male	22.0	1.72	68.0	yes	yes	2.0	3.0	Sometimes	no	2.0	no	1.0	1.0	no	Public

```
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  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

```
# all columns in df
df.columns
```

```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
```

```
       'CALC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

```
len(df.columns)
```

```
17
```

All columns and their significance

- Gender: Gender of an individual
- Age: The age of the individuals, in years.
- Height: Indicates the height of the individuals
- Weight: Denotes the weight of the individuals.
- Family_history_with_overweight: Indicates whether there is a family history of overweight among the individuals
- FAVC: Stands for "Frequent consumption of high calorie food,.
- FCVC: Stands for "Frequent consumption of vegetables,"
- NCP: Represents the number of main meals per day that individuals typically consume.
- CAEC: Indicates the consumption of food between meals, categorized into different levels
- Smoke: Indicates whether the individuals smoke or not.
- CH2O: Consumption of water daily
- SCC: Indicating whether the individuals monitor the calories in their daily food intake.
- FAF: Representing the frequency of physical activity.
- TUE: Representing the time spent sitting per day.
- CALC: Indicating the consumption of alcohol.
- MTRANS: Representing the mode of transportation.
- NObeyesdad: Indicating the weight status of the individuals.

```
df.shape
```

```
(2111, 17)
```

```
df.describe()
```

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000
mean	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	1.010298	0.657866
std	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	0.850592	0.608927
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000	0.000000
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	0.124505	0.000000
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	1.000000	0.625350
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	1.666678	1.000000
max	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000	2.000000

The **Body Mass Index (BMI)** column

we created a BMI column using weight and height columns.

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)}^2}$$

- BMI is important because it serves as a simple and widely-used measure to assess weight status and associated health risks, helping identify individuals at risk of obesity-related diseases, guiding treatment decisions, monitoring population health trends, and informing public health interventions.

```
df['BMI'] = df['Weight']/(df['Height']**2)
```

```
# Gender    Age Height  Weight  family_history_with_overweight  FAVC    FCVC    NCP CAEC    SMOKE   CH2O    SCC FAF TUE CALC    MTRANS  NObe
df = df[['Gender', 'Age', 'Height', 'Weight','BMI','family_history_with_overweight','FAVC','FCVC','NCP','CAEC','SMOKE','CH2O','SCC', 'FAF','
```

```
df.head()
```

	Gender	Age	Height	Weight	BMI	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	C
0	Female	21.0	1.62	64.0	24.386526	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	
1	Female	21.0	1.52	56.0	24.238227	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometir
2	Male	23.0	1.80	77.0	23.765432	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Freque
3	Male	27.0	1.80	87.0	26.851852	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Freque
4	Male	22.0	1.78	89.8	28.342381	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometir

Checking for Null values

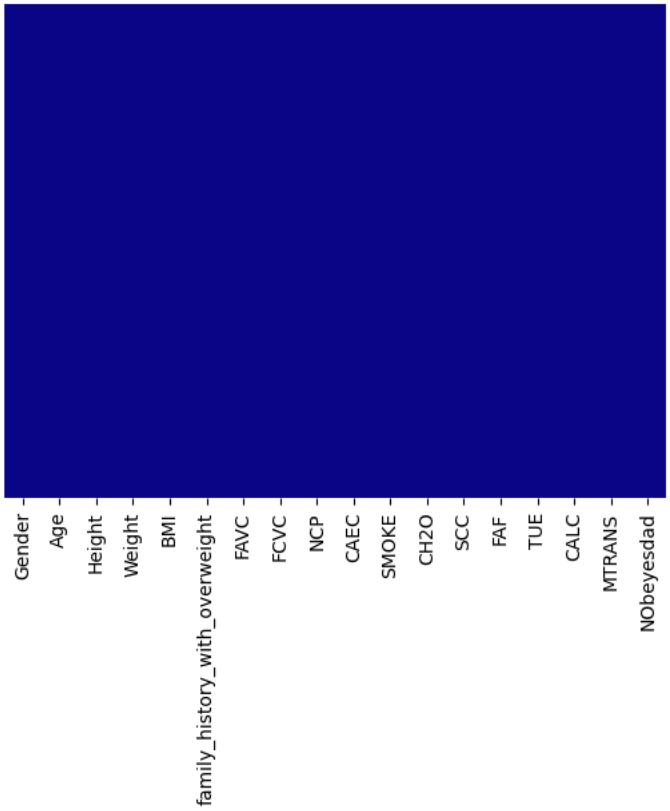
```
df.isna().sum()
```

```
Gender                          0
Age                             0
Height                          0
Weight                          0
BMI                             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
NObeyesdad                      0
dtype: int64
```

Checking heatmap for missing values

```
sns.heatmap(df.isnull(), cbar=False, yticklabels=False, cmap='plasma')
```

```
<Axes: >
```



No null values observed

Checking for duplicated values

```
df.duplicated().sum()
```

```
24
```

Found 24 duplicated rows are present

```
dup_data = df[df.duplicated()]
print(dup_data.shape)
```

```
(24, 18)
```

```
df = df.drop_duplicates(keep='last')
df.shape
```

```
(2087, 18)
```

Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight', 'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS', 'NObeyesdad'], dtype='object')

Visualize distribution of relationship between features: family_history_with_overweight, Gender

[ ] ↳ *4 cells hidden*

Visualize distribution of feature: SMOKE

[ ] ↳ *1 cell hidden*

Visualize distribution of feature: CALC

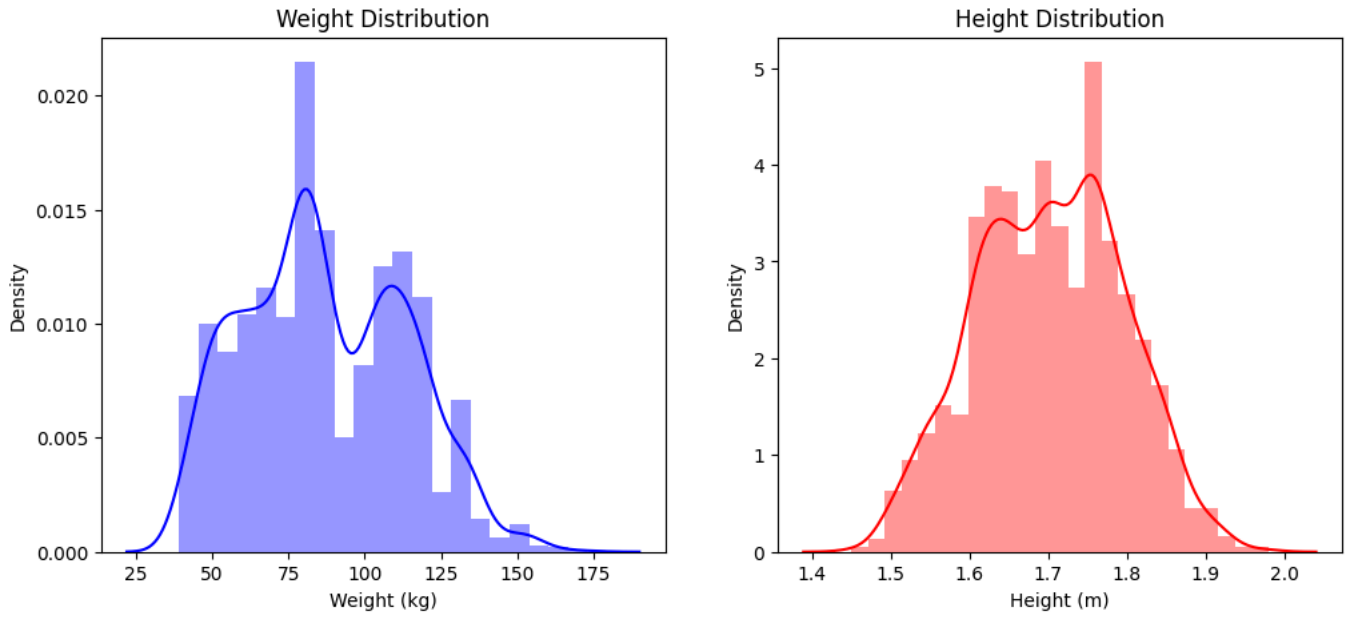
[ ] ↳ *2 cells hidden*

Visualize distribution of feature: MTRANS

[ ] ↳ *2 cells hidden*

Plotting Weight & Height Distribution levels over Desity

```
plt.figure(figsize=(12,5))
#Weight distributon suplot
plt.subplot(1, 2, 1)
sns.distplot(df["Weight"], color="b").set_title('Weight Distribution')
plt.xlabel("Weight (kg)")

#Height distribution subplot
plt.subplot(1, 2, 2)
sns.distplot(df["Height"], color="r").set_title('Height Distribution')
plt.xlabel("Height (m)")

plt.show()
```



Understanding the distribution of classes/categories within the target variable

[ ] ↳ *1 cell hidden*

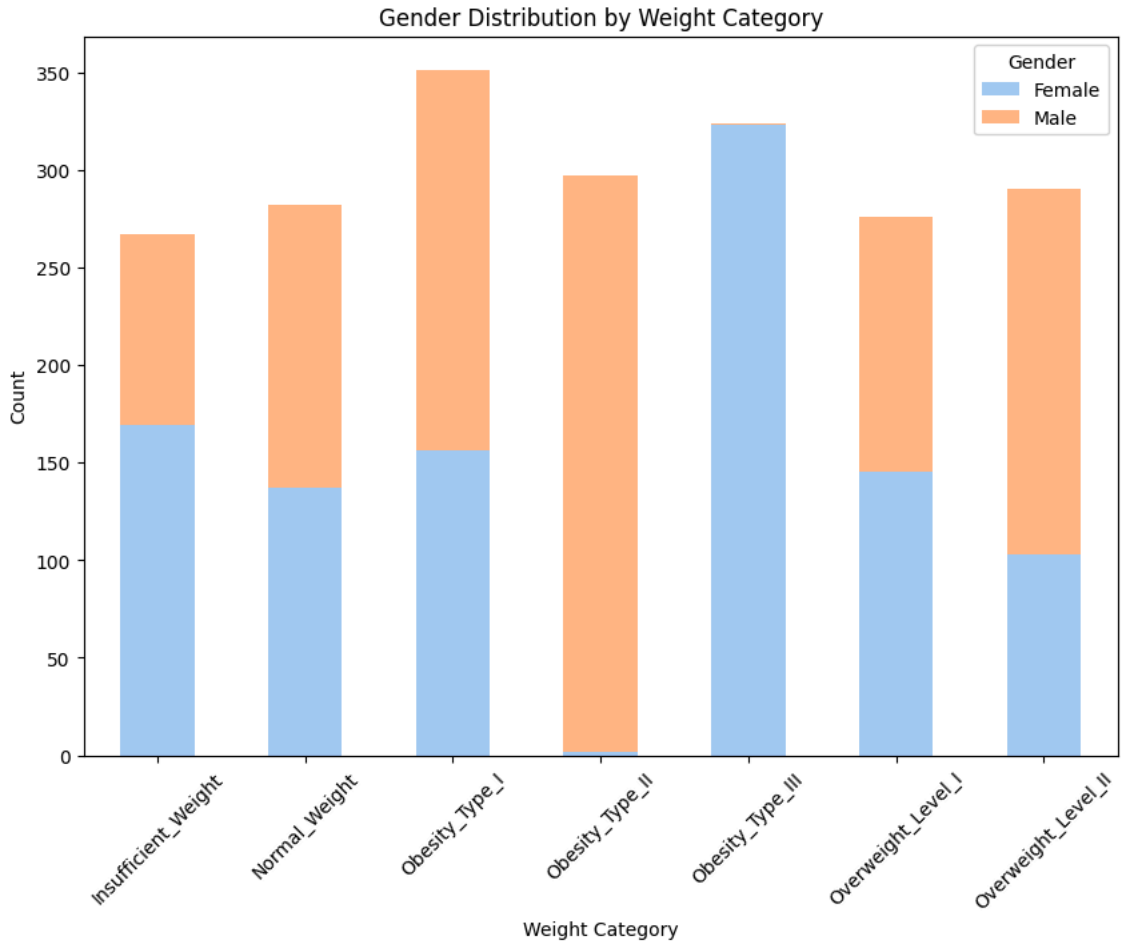
Distribution of classes/categories in the 'NObeyesdad' column

[ ] ↳ *1 cell hidden*

Plotting Gender Distribution levels by Weight

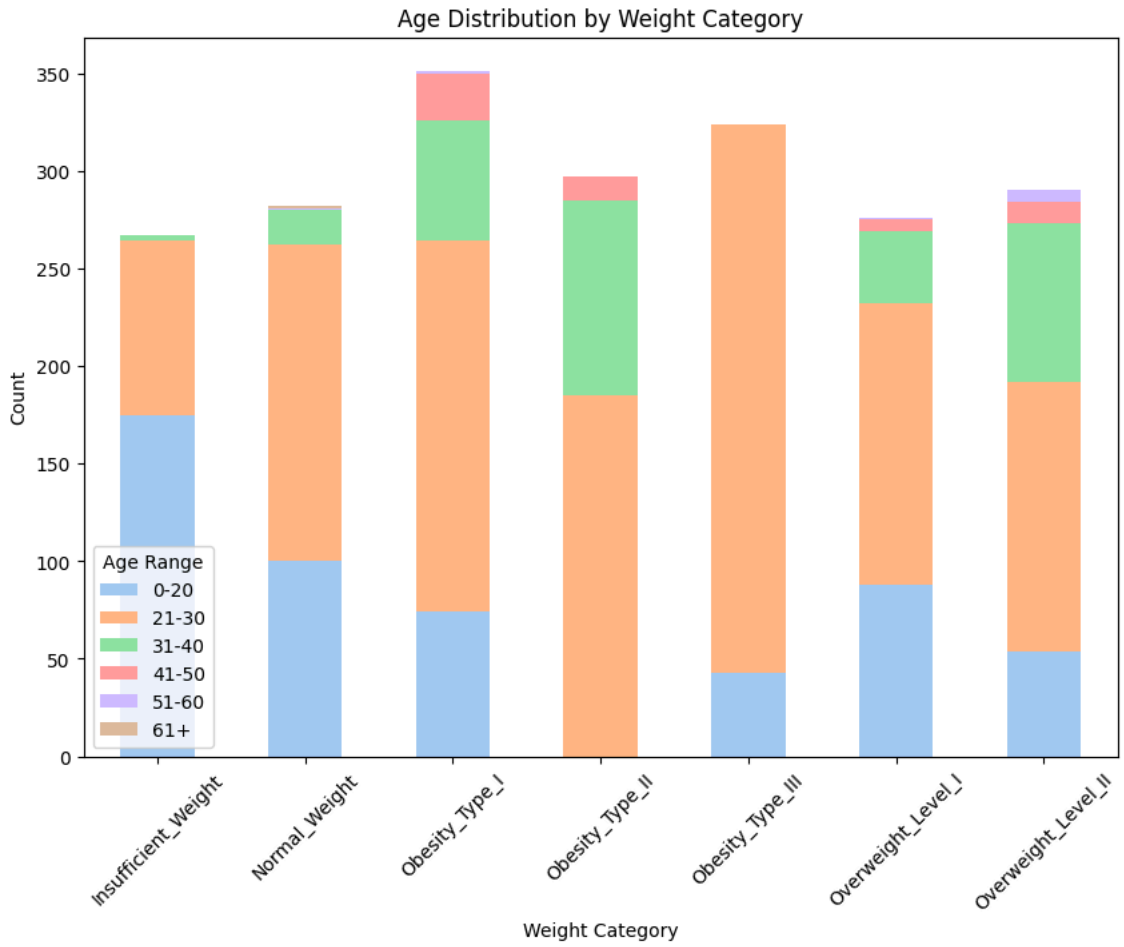
```
import matplotlib.pyplot as plt
import seaborn as sns

# Grouping data by weight category and gender
gender_counts = df.groupby(['NObeyesdad', 'Gender']).size().unstack()

# Plotting grouped bar chart for gender distribution
gender_counts.plot(kind='bar', stacked=True, figsize=(10, 7))
plt.title('Gender Distribution by Weight Category')
```

```
plt.xlabel('Weight Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Gender')

plt.show()
```



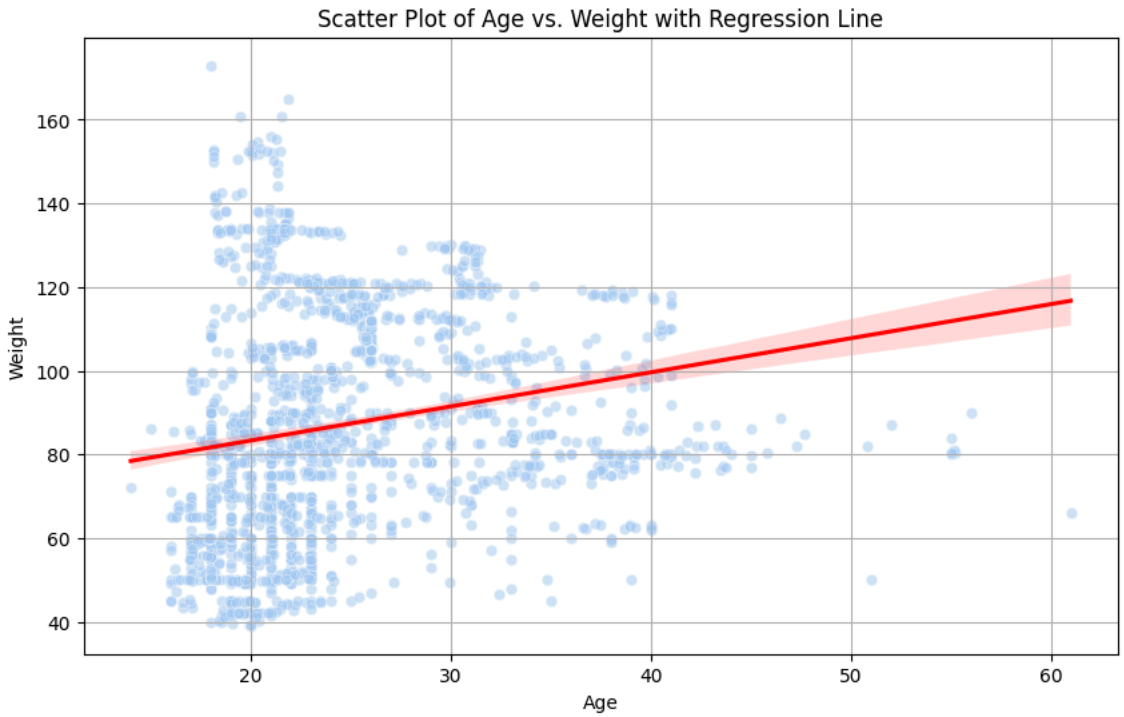
Plotting Age Distribution levels by Weight

```
import matplotlib.pyplot as plt
import seaborn as sns
#Testing encoding on a copy of the df to not mess with original, and to maintain original column names
dfc = df.copy()

# Define age ranges
age_bins = [0, 20, 30, 40, 50, 60, 100]  # Define your own age bins as needed
age_labels = ['0-20', '21-30', '31-40', '41-50', '51-60', '61+']

# Create a new column in the DataFrame with age ranges
dfc['Age_range'] = pd.cut(dfc['Age'], bins=age_bins, labels=age_labels, right=False)

# Grouping data by weight category and age range
age_counts = dfc.groupby(['NObeyesdad', 'Age_range']).size().unstack()

# Plotting grouped bar chart for age distribution
age_counts.plot(kind='bar', stacked=True, figsize=(10, 7))
plt.title('Age Distribution by Weight Category')
plt.xlabel('Weight Category')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Age Range')

plt.show()
```



Scatter plot to visualize the relationship between Age and Weight, with a regression line added to show the trend

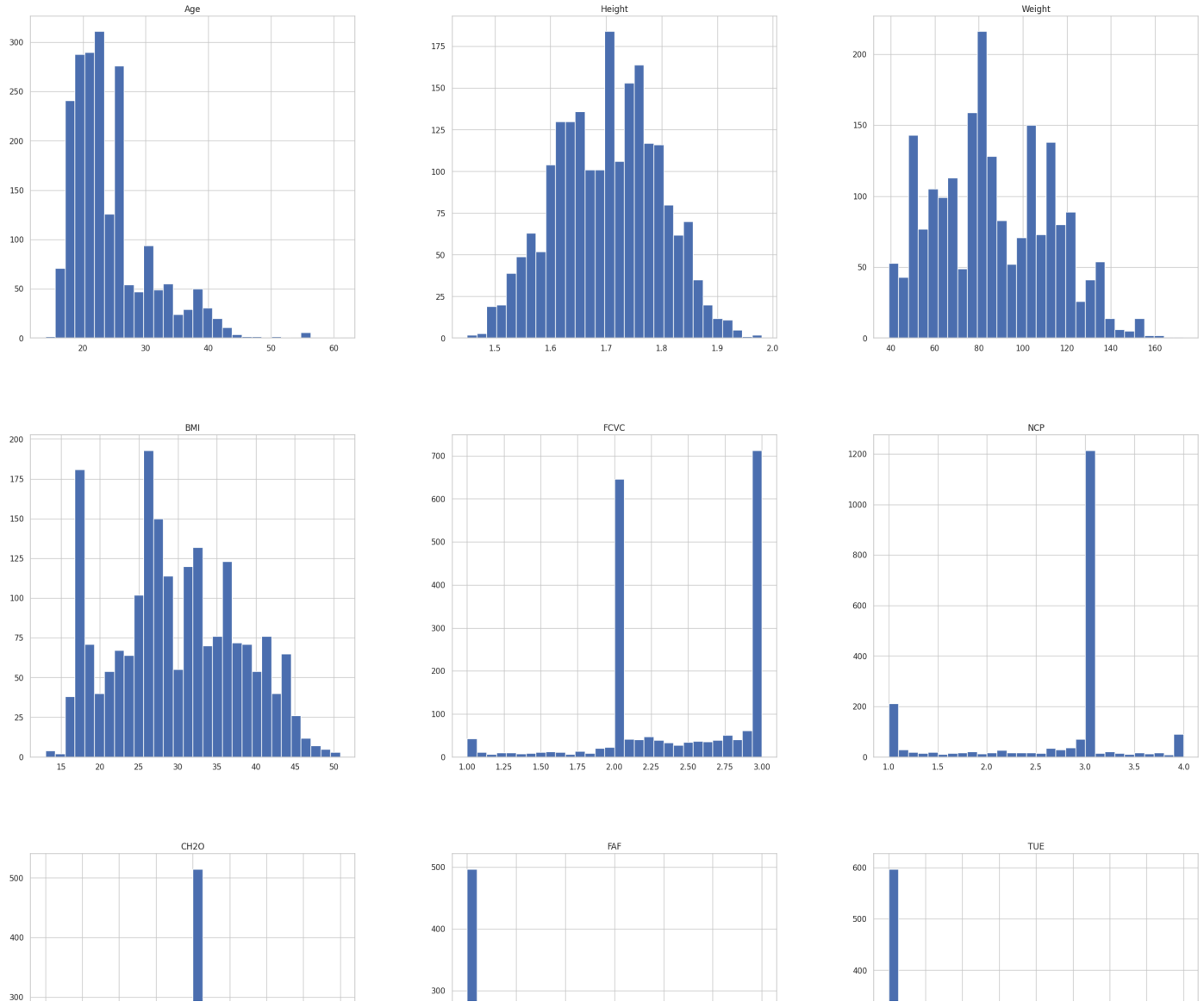
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x="Age", y="Weight", data=df, alpha=0.5)  # Scatter plot
sns.regplot(x="Age", y="Weight", data=df, scatter=False, color='red')  # Regression line
plt.title('Scatter Plot of Age vs. Weight with Regression Line')
plt.xlabel('Age')
plt.ylabel('Weight')
plt.grid(True)
plt.show()
```



Histograms for each feature

```
sns.set(style="whitegrid")
df.hist(figsize=(30, 30), bins=30)
plt.suptitle('Feature Distributions', fontsize=20)
plt.show()
```

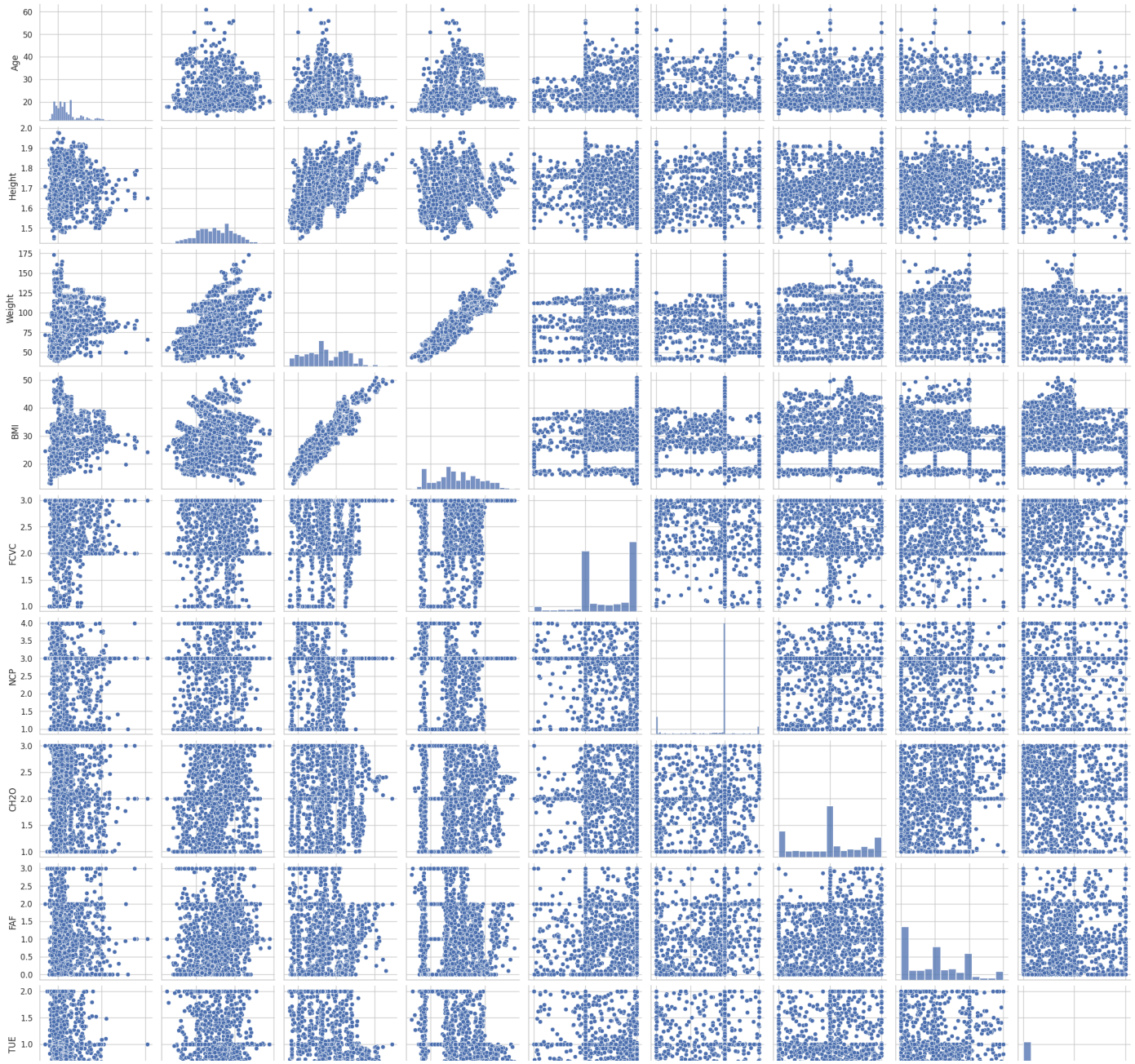
Feature Distributions



Pairwise plots to visualize relationships between variables

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7e612fcb7610>
```



Observations

- we observed there are no missing values in the dataset.
- The shape is 2111 rows and 17 columns, including some categorical and some numerical. The initial averages of the columns are different than eachother with a wide range, the lowest is around 0.65 and the high is around 86.
- The genders are fairly balanced as well as the target variable. This is due to SMOTE, which helps balance the dataset.
- According to a research paper provided by TA, SMOTE provides benefits for the model to predict the lower category especially for weak classifiers, "like SVM and MLP." Regardless, it is a valid dataset to use.
- For the feature distributions only 1 or 2 look similar to normal distribution, like height.

Preprocessing

Outlier detection

```
list1 = ["Age", "Height", "Weight", "FCVC", "NCP", "TUE", 'FAF', 'CH2O','BMI']
for col in list1:
    print(col + " - min: " + str(df[col].min()) + ", max: " + str(df[col].max()))
```

```
Age - min: 14.0, max: 61.0
Height - min: 1.45, max: 1.98
Weight - min: 39.0, max: 173.0
FCVC - min: 1.0, max: 3.0
NCP - min: 1.0, max: 4.0
TUE - min: 0.0, max: 2.0
FAF - min: 0.0, max: 3.0
CH2O - min: 1.0, max: 3.0
BMI - min: 12.998684889724604, max: 50.81175280566433
```

We considered all columns. to check min and max range.

- Age
- FCVC (consumption of veggies)
- NCP (number of main meals)
- TUE (technology usage per day in hours)
- FAF (physical activity per day in hours)
- CH20 (Water consumption per day per liters)

We calculated the min and max values for above columns we didn't find any unusal cases.

- For columns Height,Weight columns, it is possible that there are extreme or unusual cases, such as height or weight, which it does not make sense to change them with the mean or delete the row.
- The reason why we are not doing outlier handling is because we want our model to be trained with data that has variability and isn't used to only one range of data. Furthermore for this dataset specifically,

```
df.columns
```

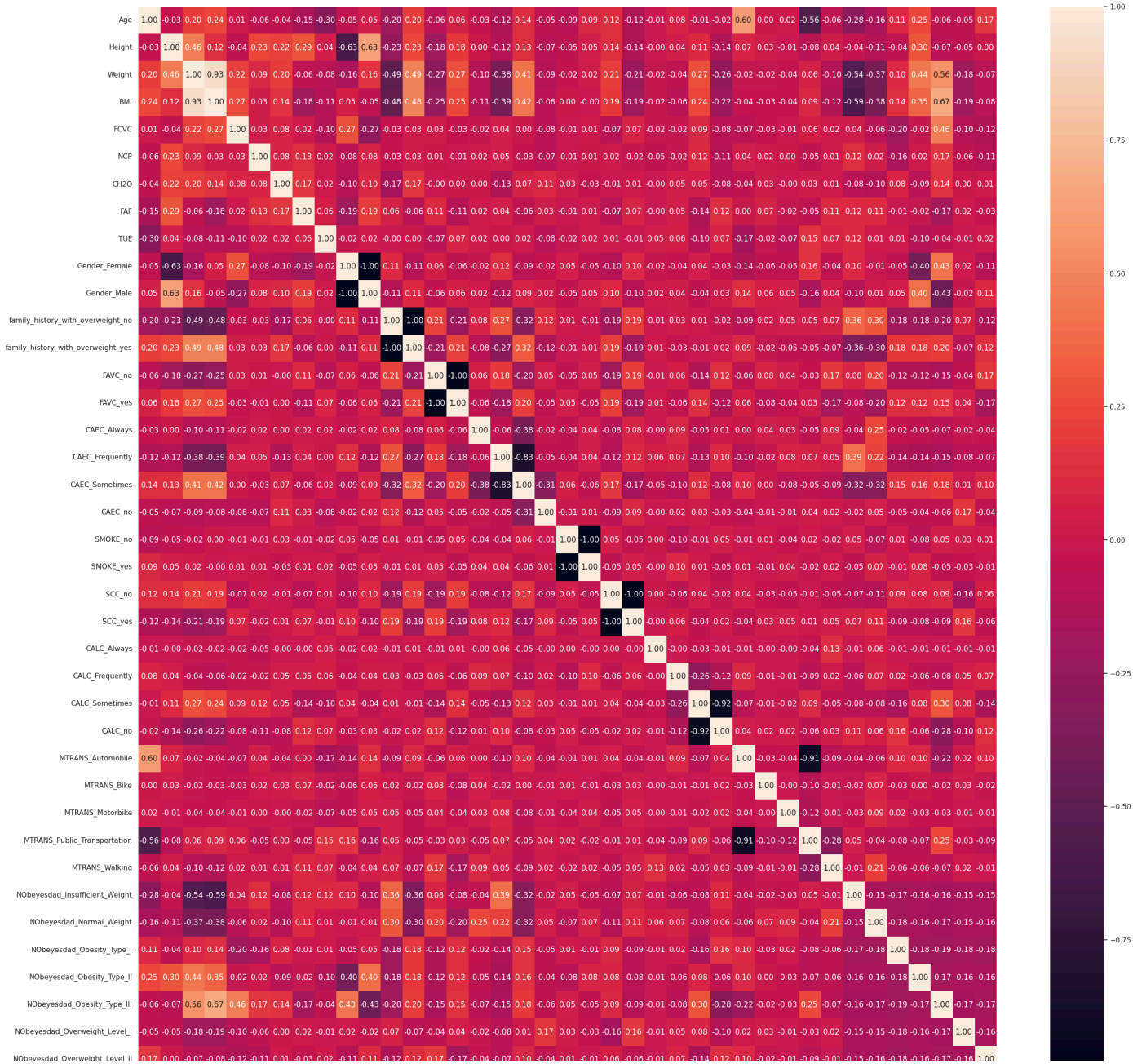
```
Index(['Gender', 'Age', 'Height', 'Weight', 'BMI',
       'family_history_with_overweight', 'FAVC', 'FCVC', 'NCP', 'CAEC',
       'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

```
# Identifying non-numeric columns
non_numeric_columns = df.select_dtypes(include=['object']).columns

# One-hot encoding non-numeric columns
numeric_df = pd.get_dummies(df, columns=non_numeric_columns)

# Computing the correlation matrix for the numeric DataFrame
corr = numeric_df.corr()

# Plotting the correlation heatmap using the correlation_matrix

corr
```

	Age	Height	Weight	BMI	FCVC	NCP	CH2O	FAF	TUE	Gender_Fe
Age	1.000000	-0.031748	0.198160	0.240769	0.013572	-0.055823	-0.044058	-0.148202	-0.302927	-0.05
Height	-0.031748	1.000000	0.457468	0.124466	-0.040363	0.227806	0.220487	0.293584	0.041808	-0.62
Weight	0.198160	0.457468	1.000000	0.934494	0.216574	0.092149	0.203823	-0.056490	-0.079351	-0.16
BMI	0.240769	0.124466	0.934494	1.000000	0.265082	0.027936	0.144110	-0.182932	-0.105036	0.05
FCVC	0.013572	-0.040363	0.216574	0.265082	1.000000	0.034885	0.081332	0.022003	-0.104128	0.27
NCP	-0.055823	0.227806	0.092149	0.027936	0.034885	1.000000	0.075335	0.127816	0.015693	-0.07
CH2O	-0.044058	0.220487	0.203823	0.144110	0.081332	0.075335	1.000000	0.165310	0.020704	-0.09
FAF	-0.148202	0.293584	-0.056490	-0.182932	0.022003	0.127816	0.165310	1.000000	0.058716	-0.18
TUE	-0.302927	0.041808	-0.079351	-0.105036	-0.104128	0.015693	0.020704	0.058716	1.000000	-0.02
Gender_Female	-0.050641	-0.626748	-0.163176	0.054737	0.271575	-0.077863	-0.095129	-0.189471	-0.022356	1.00
Gender_Male	0.050641	0.626748	0.163176	-0.054737	-0.271575	0.077863	0.095129	0.189471	0.022356	-1.00
family_history_with_overweight_no	-0.200379	-0.232258	-0.492969	-0.483648	-0.033199	-0.028411	-0.168627	0.062937	-0.002314	0.11
family_history_with_overweight_yes	0.200379	0.232258	0.492969	0.483648	0.033199	0.028411	0.168627	-0.062937	0.002314	-0.11
FAVC_no	-0.063895	-0.180694	-0.274655	-0.247368	0.025419	0.006398	-0.002993	0.111184	-0.071505	0.06
FAVC_yes	0.063895	0.180694	0.274655	0.247368	-0.025419	-0.006398	0.002993	-0.111184	0.071505	-0.06
CAEC_Always	-0.032390	0.002886	-0.097207	-0.109475	-0.019015	0.023258	0.003748	0.022562	0.019332	-0.02
CAEC_Frequently	-0.120945	-0.115986	-0.378262	-0.393218	0.040306	0.054341	-0.125923	0.040035	0.002043	0.11
CAEC_Sometimes	0.136576	0.125208	0.406347	0.419729	0.003200	-0.033534	0.069554	-0.055138	0.017753	-0.08
CAEC_no	-0.046874	-0.069557	-0.094259	-0.080559	-0.082861	-0.065879	0.106376	0.028718	-0.076790	-0.02
SMOKE_no	-0.091261	-0.054326	-0.024369	0.002022	-0.013716	-0.005009	0.031642	-0.010811	-0.016491	0.04
SMOKE_yes	0.091261	0.054326	0.024369	-0.002022	0.013716	0.005009	-0.031642	0.010811	0.016491	-0.04
SCC_no	0.117959	0.137078	0.205409	0.186875	-0.071179	0.020461	-0.009325	-0.073768	0.012875	-0.10
SCC_yes	-0.117959	-0.137078	-0.205409	-0.186875	0.071179	-0.020461	0.009325	0.073768	-0.012875	0.10
CALC_Always	-0.011530	-0.000628	-0.018278	-0.019852	-0.017261	-0.048725	-0.000171	-0.000329	0.048145	-0.02
CALC_Frequently	0.081574	0.043854	-0.043966	-0.064658	-0.019581	-0.023367	0.051117	0.054329	0.059820	-0.03
CALC_Sometimes	-0.010701	0.114983	0.266804	0.240513	0.086239	0.115166	0.054465	-0.136111	-0.097873	0.04
CALC_no	-0.020356	-0.135351	-0.256268	-0.221067	-0.080193	-0.106960	-0.075988	0.118718	0.074948	-0.02
MTRANS_Automobile	0.604473	0.072450	-0.018723	-0.037421	-0.065109	0.043570	-0.042212	0.000814	-0.169907	-0.13
MTRANS_Bike	0.003291	0.028580	-0.022475	-0.033238	-0.030233	0.022677	0.026802	0.067117	-0.022371	-0.05
MTRANS_Motorbike	0.019867	-0.011323	-0.038274	-0.036321	-0.007874	0.002485	-0.000568	-0.016604	-0.068496	-0.04
MTRANS_Public_Transportation	-0.555532	-0.084407	0.064241	0.091741	0.061370	-0.048486	0.031353	-0.048621	0.149448	0.16

```
plt.figure(figsize=(30,30))
sns.heatmap(corr, annot = True, fmt = '.2f')
```

<Axes: >



SPLIT AND TRAIN DATA

Importing for this section

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, StratifiedKFold, KFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, recall_score, f1_score, roc_auc_score, precision_score
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline

import xgboost as xgb
```

Label encoding

```
# Find which columns are catagorical
list_of_categorical_columns = []

for col, dtype in df.dtypes.items(): #going through each category and checking if it is categotical
    if dtype == 'object':
        list_of_categorical_columns.append(col)

print(list_of_categorical_columns)
```

```
['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS', 'NObeyesdad']
```

```
#Label encoding

le = LabelEncoder()

for col in list_of_categorical_columns: # going through these categorical features and changing them into numerical
    df[col] = le.fit_transform(df[col])

df.head()
```

	Gender	Age	Height	Weight	BMI	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS
0	0	21.0	1.62	64.0	24.386526	1	0	2.0	3.0	2	0	2.0	0	0.0	1.0	3	3
1	0	21.0	1.52	56.0	24.238227	1	0	3.0	3.0	2	1	3.0	1	3.0	0.0	2	3
2	1	23.0	1.80	77.0	23.765432	1	0	2.0	3.0	2	0	2.0	0	2.0	1.0	1	3
3	1	27.0	1.80	87.0	26.851852	0	0	3.0	3.0	2	0	2.0	0	2.0	0.0	1	4
4	1	22.0	1.78	89.8	28.342381	0	0	2.0	1.0	2	0	2.0	0	0.0	0.0	2	3

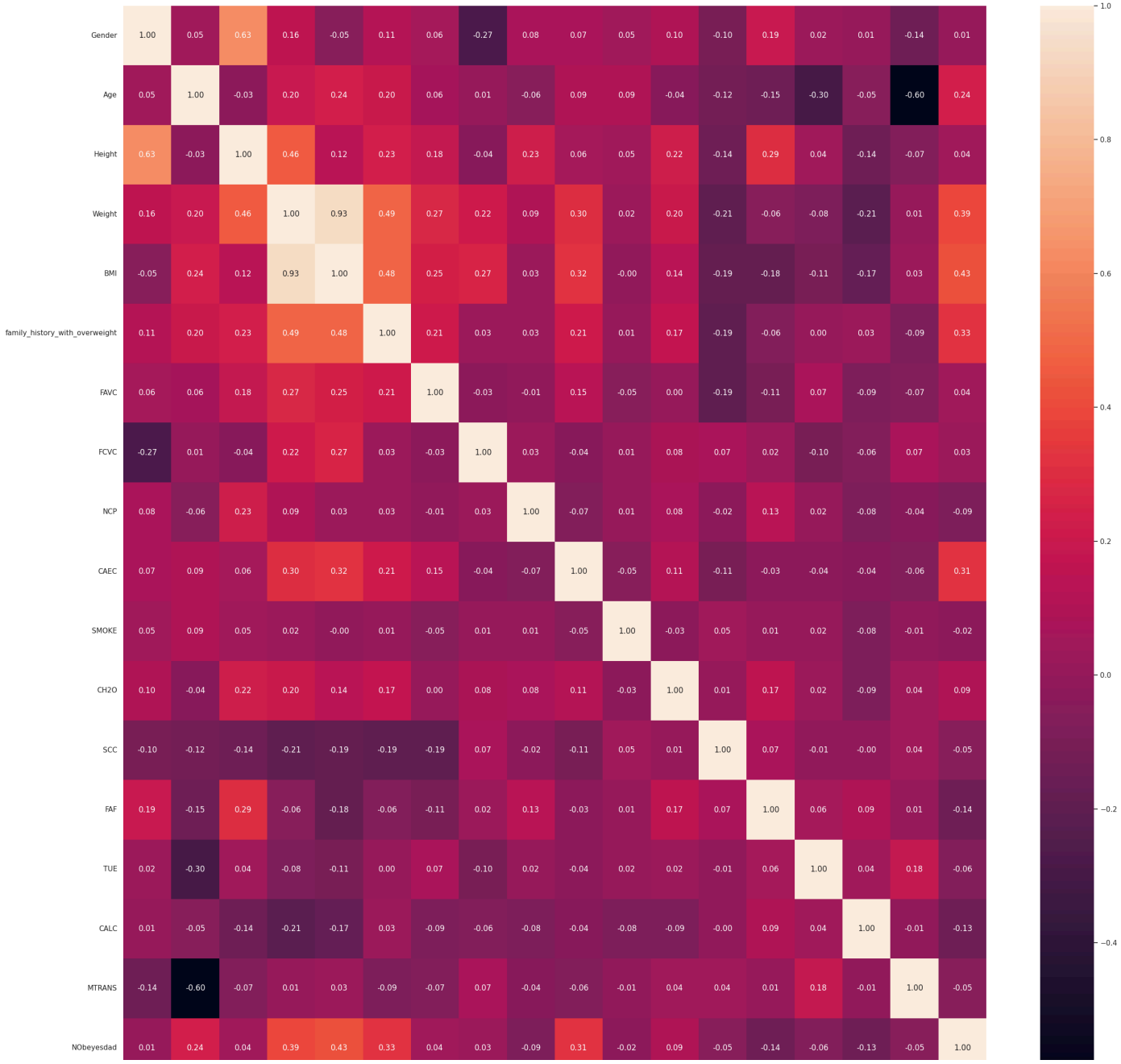
- We are applying label encoding to categorize variables numerically while maintaining consistency in our preprocessing approach.
- This allows us to construct a correlation matrix that captures relationships between all variables, helping us gain insights into how different factors interrelate.

```
df.corr()
```

	Gender	Age	Height	Weight	BMI	family_history_with_overweight	FAVC	FCVC	
Gender	1.000000	0.050641	0.626748	0.163176	-0.054737	0.113492	0.061220	-0.271575	0.0
Age	0.050641	1.000000	-0.031748	0.198160	0.240769	0.200379	0.063895	0.013572	-0.0
Height	0.626748	-0.031748	1.000000	0.457468	0.124466	0.232258	0.180694	-0.040363	0.2
Weight	0.163176	0.198160	0.457468	1.000000	0.934494	0.492969	0.274655	0.216574	0.0
BMI	-0.054737	0.240769	0.124466	0.934494	1.000000	0.483648	0.247368	0.265082	0.0
family_history_with_overweight	0.113492	0.200379	0.232258	0.492969	0.483648	1.000000	0.214329	0.033199	0.0
FAVC	0.061220	0.063895	0.180694	0.274655	0.247368	0.214329	1.000000	-0.025419	-0.0
FCVC	-0.271575	0.013572	-0.040363	0.216574	0.265082	0.033199	-0.025419	1.000000	0.0
NCP	0.077863	-0.055823	0.227806	0.092149	0.027936	0.028411	-0.006398	0.034885	1.0
CAEC	0.074564	0.092097	0.058001	0.300271	0.322919	0.207738	0.147921	-0.038565	-0.0
SMOKE	0.045501	0.091261	0.054326	0.024369	-0.002022	0.014885	-0.050713	0.013716	0.0
CH2O	0.095129	-0.044058	0.220487	0.203823	0.144110	0.168627	0.002993	0.081332	0.0
SCC	-0.102435	-0.117959	-0.137078	-0.205409	-0.186875	-0.193947	-0.191277	0.071179	-0.0
FAF	0.189471	-0.148202	0.293584	-0.056490	-0.182932	-0.062937	-0.111184	0.022003	0.1
TUE	0.022356	-0.302927	0.041808	-0.079351	-0.105036	0.002314	0.071505	-0.104128	0.0
CALC	0.010574	-0.045565	-0.135756	-0.211351	-0.172662	0.028403	-0.087661	-0.063132	-0.0
MTRANS	-0.139044	-0.601476	-0.068258	0.009836	0.026061	-0.092730	-0.069422	0.069012	-0.0
NObeyesdad	0.014699	0.238308	0.038700	0.388802	0.429668	0.330391	0.041023	0.025728	-0.0

```
plt.figure(figsize=(30,30))
sns.heatmap(df.corr(), annot = True, fmt = '.2f')
```

```
<Axes: >
```



- by observing the heatmap we can say that the target variable is most correlated with BMI, Weight, Family_history_with_overweight, and CAEC (Consumption of food between meals) and age, in that order.
- These are the features that have the most influence over changes in the target variable.

LASSO Feature selection

```
x = (df.drop('NObeyesdad', axis=1))
y = df['NObeyesdad']
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size = .2, shuffle = True)
```

```
features = df.columns[:-1]
```

```
features
```

```
Index(['Gender', 'Age', 'Height', 'Weight', 'BMI',
       'family_history_with_overweight', 'FAVC', 'FCVC', 'NCP', 'CAEC',
       'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS'],
      dtype='object')
```

```
# pipeline will scale and do lasso at once internally, this will not change the df

pipeline = Pipeline([
                    ('scaler',StandardScaler()),
                    ('model',Lasso())
])

search = GridSearchCV(pipeline,
                      {'model__alpha':np.arange(0.1,10,0.1)},
                      cv = 5, scoring="neg_mean_squared_error",verbose=3
                      )

search.fit(xtrain,ytrain)

search.best_params_
```

```
Fitting 5 folds for each of 99 candidates, totalling 495 fits
[CV 1/5] END .................model__alpha=0.1;, score=-2.762 total time=   0.0s
[CV 2/5] END .................model__alpha=0.1;, score=-2.745 total time=   0.0s
[CV 3/5] END .................model__alpha=0.1;, score=-3.084 total time=   0.0s
[CV 4/5] END .................model__alpha=0.1;, score=-2.860 total time=   0.0s
[CV 5/5] END .................model__alpha=0.1;, score=-2.703 total time=   0.0s
[CV 1/5] END .................model__alpha=0.2;, score=-2.856 total time=   0.0s
[CV 2/5] END .................model__alpha=0.2;, score=-2.894 total time=   0.0s
[CV 3/5] END .................model__alpha=0.2;, score=-3.150 total time=   0.0s
[CV 4/5] END .................model__alpha=0.2;, score=-2.998 total time=   0.0s
[CV 5/5] END .................model__alpha=0.2;, score=-2.875 total time=   0.0s
[CV 1/5] END .model__alpha=0.30000000000000004;, score=-2.948 total time=   0.0s
[CV 2/5] END .model__alpha=0.30000000000000004;, score=-3.032 total time=   0.0s
[CV 3/5] END .model__alpha=0.30000000000000004;, score=-3.225 total time=   0.0s
[CV 4/5] END .model__alpha=0.30000000000000004;, score=-3.109 total time=   0.0s
[CV 5/5] END .model__alpha=0.30000000000000004;, score=-3.027 total time=   0.0s
[CV 1/5] END .................model__alpha=0.4;, score=-3.034 total time=   0.0s
[CV 2/5] END .................model__alpha=0.4;, score=-3.175 total time=   0.0s
[CV 3/5] END .................model__alpha=0.4;, score=-3.346 total time=   0.0s
[CV 4/5] END .................model__alpha=0.4;, score=-3.256 total time=   0.0s
[CV 5/5] END .................model__alpha=0.4;, score=-3.177 total time=   0.0s
[CV 1/5] END .................model__alpha=0.5;, score=-3.145 total time=   0.0s
[CV 2/5] END .................model__alpha=0.5;, score=-3.309 total time=   0.0s
[CV 3/5] END .................model__alpha=0.5;, score=-3.446 total time=   0.0s
[CV 4/5] END .................model__alpha=0.5;, score=-3.392 total time=   0.0s
[CV 5/5] END .................model__alpha=0.5;, score=-3.331 total time=   0.0s
[CV 1/5] END .................model__alpha=0.6;, score=-3.237 total time=   0.0s
[CV 2/5] END .................model__alpha=0.6;, score=-3.433 total time=   0.0s
[CV 3/5] END .................model__alpha=0.6;, score=-3.526 total time=   0.0s
[CV 4/5] END .................model__alpha=0.6;, score=-3.515 total time=   0.0s
[CV 5/5] END .................model__alpha=0.6;, score=-3.465 total time=   0.0s
[CV 1/5] END ..model__alpha=0.7000000000000001;, score=-3.347 total time=   0.0s
[CV 2/5] END ..model__alpha=0.7000000000000001;, score=-3.578 total time=   0.0s
[CV 3/5] END ..model__alpha=0.7000000000000001;, score=-3.625 total time=   0.0s
[CV 4/5] END ..model__alpha=0.7000000000000001;, score=-3.660 total time=   0.0s
[CV 5/5] END ..model__alpha=0.7000000000000001;, score=-3.621 total time=   0.0s
[CV 1/5] END .................model__alpha=0.8;, score=-3.474 total time=   0.0s
[CV 2/5] END .................model__alpha=0.8;, score=-3.743 total time=   0.0s
[CV 3/5] END .................model__alpha=0.8;, score=-3.741 total time=   0.0s
[CV 4/5] END .................model__alpha=0.8;, score=-3.826 total time=   0.0s
[CV 5/5] END .................model__alpha=0.8;, score=-3.797 total time=   0.0s
[CV 1/5] END .................model__alpha=0.9;, score=-3.591 total time=   0.0s
[CV 2/5] END .................model__alpha=0.9;, score=-3.837 total time=   0.0s
[CV 3/5] END .................model__alpha=0.9;, score=-3.867 total time=   0.0s
[CV 4/5] END .................model__alpha=0.9;, score=-3.925 total time=   0.0s
[CV 5/5] END .................model__alpha=0.9;, score=-3.877 total time=   0.0s
[CV 1/5] END .................model__alpha=1.0;, score=-3.591 total time=   0.0s
[CV 2/5] END .................model__alpha=1.0;, score=-3.837 total time=   0.0s
[CV 3/5] END .................model__alpha=1.0;, score=-3.867 total time=   0.0s
[CV 4/5] END .................model__alpha=1.0;, score=-3.925 total time=   0.0s
[CV 5/5] END .................model__alpha=1.0;, score=-3.877 total time=   0.0s
[CV 1/5] END .................model__alpha=1.1;, score=-3.591 total time=   0.0s
[CV 2/5] END .................model__alpha=1.1;, score=-3.837 total time=   0.0s
[CV 3/5] END .................model__alpha=1.1;, score=-3.867 total time=   0.0s
[CV 4/5] END .................model__alpha=1.1;, score=-3.925 total time=   0.0s
[CV 5/5] END .................model__alpha=1.1;, score=-3.877 total time=   0.0s
[CV 1/5] END ..model__alpha=1.2000000000000002;, score=-3.591 total time=   0.0s
[CV 2/5] END ..model__alpha=1.2000000000000002;, score=-3.837 total time=   0.0s
```

- Calculate coefficients and importances

```
coefficients = search.best_estimator_.named_steps['model'].coef_
importance = np.abs(coefficients)
print('These are the importances: ', importance)
```

```
These are the importances:  [0.         0.16048272 0.         0.         0.53787958 0.21654765
 0.09237026 0.         0.05246924 0.27481512 0.         0.
 0.         0.03421652 0.         0.06199364 0.        ]
```

Features with importance

```
Imp_features = np.array(features)[importance > 0]
Imp_features
```

```
array(['Age', 'BMI', 'family_history_with_overweight', 'FAVC', 'NCP',
       'CAEC', 'FAF', 'CALC'], dtype=object)
```

Features without importance

```
np.array(features)[importance == 0]
```

```
array(['Gender', 'Height', 'Weight', 'FCVC', 'SMOKE', 'CH2O', 'SCC',
       'TUE', 'MTRANS'], dtype=object)
```

```
x = df[Imp_features]
x.head()
```

	Age	BMI	family_history_with_overweight	FAVC	NCP	CAEC	FAF	CALC
0	21.0	24.386526	1	0	3.0	2	0.0	3
1	21.0	24.238227	1	0	3.0	2	3.0	2
2	23.0	23.765432	1	0	3.0	2	2.0	1
3	27.0	26.851852	0	0	3.0	2	2.0	1
4	22.0	28.342381	0	0	1.0	2	0.0	2

```
y.head()
```

```
0    1
1    1
2    1
3    5
4    6
Name: NObeyesdad, dtype: int64
```

Standard Scaling

```
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size = .2, shuffle = True) #redoing this to get the x changes to apply

# ********
scaler = StandardScaler()
xtrain_scaled = scaler.fit_transform(xtrain)
xtest_scaled = scaler.transform(xtest)
```

Models before hyperparameter tuning

Logistic Regression

check below comment in cell remove that line if not needed

```
# Instantiate one instance of the LogisticRegression class
logistic_reg_model = LogisticRegression()

# Fit the model to the training data
logistic_reg_model.fit(xtrain_scaled,ytrain)

# Compute predict labels on x
logistic_reg_ypred = logistic_reg_model.predict(xtest_scaled)

# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
logistic_reg_accuracy = accuracy_score(ytest, logistic_reg_ypred)
logistic_reg_precision = precision_score(ytest, logistic_reg_ypred, average="macro")
logistic_reg_recall = recall_score(ytest, logistic_reg_ypred, average="macro")
logistic_reg_f1 = f1_score(ytest, logistic_reg_ypred, average="macro")

print('Logistic Regression before Hyperparameter Tuning')
print("Accuracy:", logistic_reg_accuracy)
print("Precision:", logistic_reg_precision)
print("Recall:", logistic_reg_recall)
print("F1-score:", logistic_reg_f1)
```

```
Logistic Regression before Hyperparameter Tuning
Accuracy: 0.8732057416267942
Precision: 0.8694513962156927
Recall: 0.8716374269005848
F1-score: 0.868248103123533
```

Naive Bayes

```
# Instantiate one instance of the GaussianNB class
naive_bayes_model = GaussianNB()

# Fit the model to the training data
naive_bayes_model.fit(xtrain_scaled, ytrain)

# Compute predict labels on x
naive_bayes_ypred = naive_bayes_model.predict(xtest_scaled)

# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
nb_accuracy = accuracy_score(ytest, naive_bayes_ypred)
nb_precision = precision_score(ytest, naive_bayes_ypred, average="macro")
nb_recall = recall_score(ytest, naive_bayes_ypred, average="macro")
nb_f1 = f1_score(ytest, naive_bayes_ypred, average="macro")

print('Naive Bayes')
print("Accuracy:", nb_accuracy)
print("Precision:", nb_precision)
print("Recall:", nb_recall)
print("F1-score:", nb_f1)
```

```
Naive Bayes
Accuracy: 0.8086124401913876
Precision: 0.8718832976488405
Recall: 0.7914786967418547
F1-score: 0.8006326820240518
```

Decision Trees

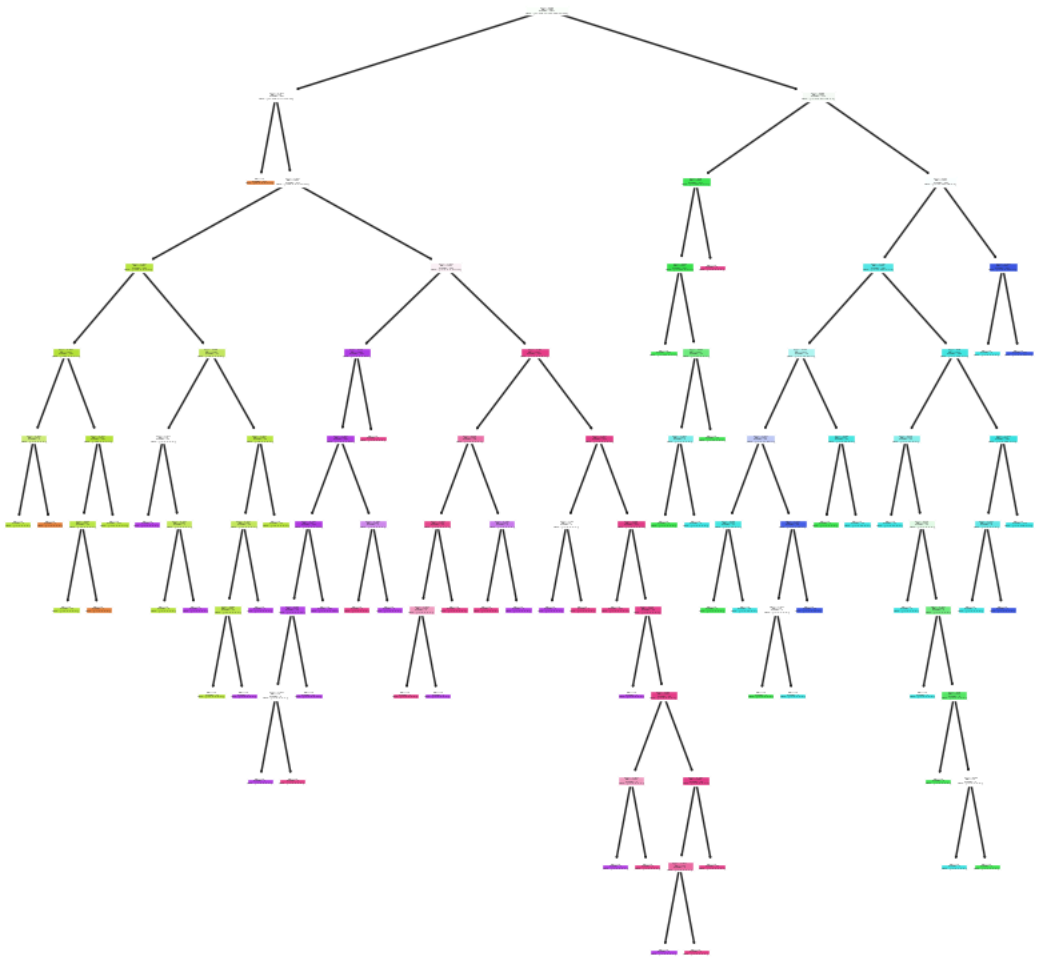
```
# Instantiate one instance of the DecisionTreeClassifier class
decision_trees_model = DecisionTreeClassifier()

# Fit the model to the training data
decision_trees_model.fit(xtrain_scaled, ytrain)

# Compute predict labels on x
decision_trees_ypred = decision_trees_model.predict(xtest_scaled)

# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
```

```
dt_accuracy = accuracy_score(ytest, decision_trees_ypred)
dt_precision = precision_score(ytest, decision_trees_ypred, average="macro")
dt_recall = recall_score(ytest, decision_trees_ypred, average="macro")
dt_f1 = f1_score(ytest, decision_trees_ypred, average="macro")

print('Decision Tree')
print("Accuracy:", dt_accuracy)
print("Precision:", dt_precision)
print("Recall:", dt_recall)
print("F1-score:", dt_f1)
```

```
Decision Tree
Accuracy: 0.9665071770334929
Precision: 0.9665057778400097
Recall: 0.9652408799777221
F1-score: 0.965269164917766
```

Plot decision tree

```
plt.figure(figsize=(10,10))
tree.plot_tree(decision_trees_model, filled=True)
plt.show()
```



KNN

```
# Instantiate one instance of the KNeighborsClassifier class with k=7
knn_model = KNeighborsClassifier(n_neighbors=7)

# Fit the model to the training data
knn_model.fit(xtrain_scaled, ytrain)
```

```
# Compute predict labels on x
knn_ypred = knn_model.predict(xtest_scaled)

# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
knn_accuracy = accuracy_score(ytest, knn_ypred)
knn_precision = precision_score(ytest, knn_ypred, average="macro")
knn_recall = recall_score(ytest, knn_ypred, average="macro")
knn_f1 = f1_score(ytest, knn_ypred, average="macro")

print('K-Nearest Neighbors')
print("Accuracy:", knn_accuracy)
print("Precision:", knn_precision)
print("Recall:", knn_recall)
print("F1-score:", knn_f1)
```

```
K-Nearest Neighbors
Accuracy: 0.8277511961722488
Precision: 0.817376735020035
Recall: 0.8210275689223057
F1-score: 0.8182169147363381
```

SVM

```
# Instantiate one instance of the SVC class
svm_model = SVC()

# Fit the model to the training data
svm_model.fit(xtrain_scaled,ytrain)

# Compute predict labels on x
svm_ypred = svm_model.predict(xtest_scaled)

# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
svm_accuracy = accuracy_score(ytest, svm_ypred)
svm_precision = precision_score(ytest, svm_ypred, average="macro")
svm_recall = recall_score(ytest, svm_ypred, average="macro")
svm_f1 = f1_score(ytest, svm_ypred, average="macro")

print('Support Vector Machine before Hyperparameter Tuning')
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
print("Recall:", svm_recall)
print("F1-score:", svm_f1)
```

```
Support Vector Machine before Hyperparameter Tuning
Accuracy: 0.9043062200956937
Precision: 0.8964480168095582
Recall: 0.9002005012531329
F1-score: 0.8979244263988143
```

XGBoost

```
# Instantiate one instance of the XGBClassifier class
xgb_model = xgb.XGBClassifier()

# Fit the model to the training data
xgb_model.fit(xtrain_scaled, ytrain)

# Compute predict labels on x
xgb_ypred = xgb_model.predict(xtest_scaled)

# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
xgboost_accuracy = accuracy_score(ytest, xgb_ypred)
xgboost_precision = precision_score(ytest, xgb_ypred, average="macro")
xgboost_recall = recall_score(ytest, xgb_ypred, average="macro")
xgboost_f1 = f1_score(ytest, xgb_ypred, average="macro")

print('XGBoost Classifier')
print("Accuracy:", xgboost_accuracy)
print("Precision:", xgboost_precision)
print("Recall:", xgboost_recall)
print("F1-score:", xgboost_f1)
```

```
XGBoost Classifier
Accuracy: 0.9784688995215312
Precision: 0.9776681164268985
Recall: 0.9768532442216653
F1-score: 0.9771369976666577
```

Decision Trees model performed the best, followed by SVM

Models after hyperparameter tuning

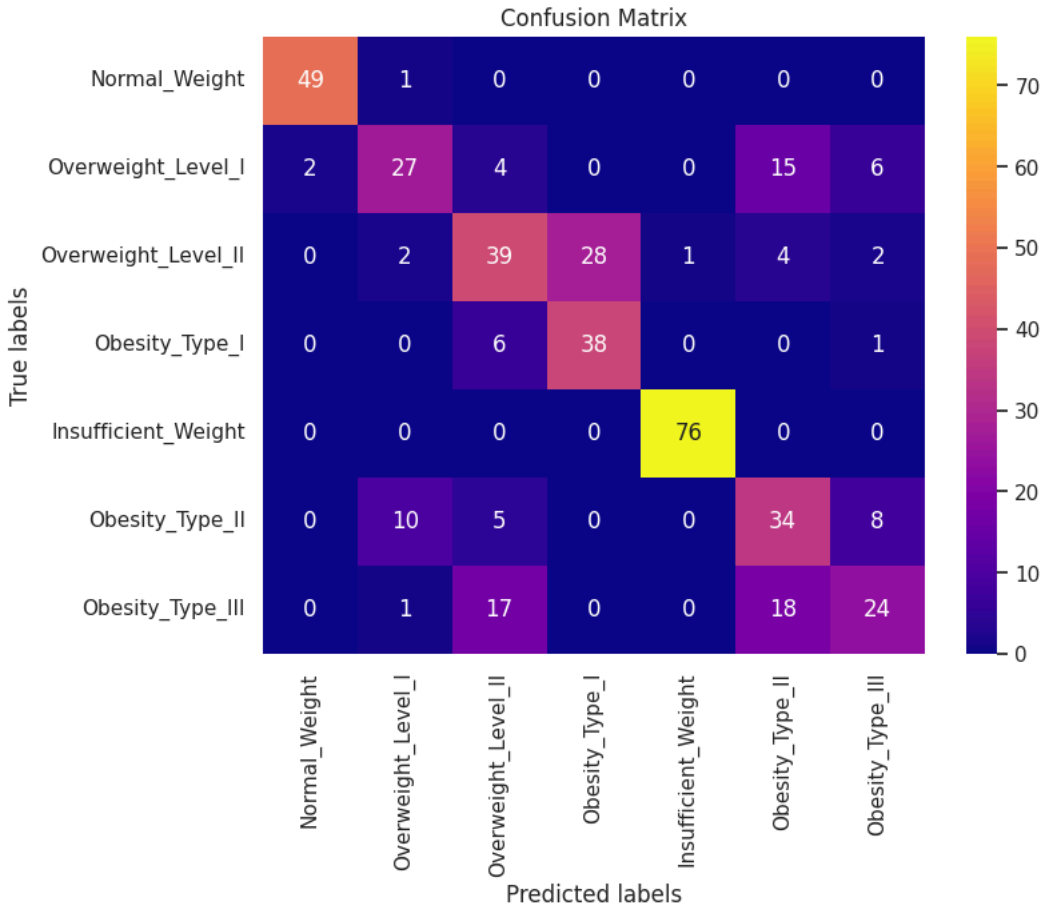
Logistic Regression

```
# Param grid for GridSearchCV
param_grid_logistic_reg = {
    'estimator__C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter
    'estimator__penalty': ['l1', 'l2']
}

# Instantiate one instance of the LogisticRegression class
logistic_reg_model = LogisticRegression()
logistic_reg_model_multi_label = OneVsRestClassifier(logistic_reg_model)

# Create a GridSearchCV object and fit it to the training data
logistic_reg_grid_search = GridSearchCV(logistic_reg_model_multi_label,
                                        param_grid_logistic_reg,
                                        cv = 5,
                                        scoring = 'accuracy')

# Fit the model to the training data
logistic_reg_grid_search.fit(xtrain_scaled, ytrain)

# Print the best hyperparameters found by GridSearchCV
print("Best Hyperparameters:", logistic_reg_grid_search.best_params_)

# Get the best model
logistic_reg_tunned_best_model = logistic_reg_grid_search.best_estimator_

# Evaluate the best model on the test set
logistic_reg_tunned_ytest_prediction = logistic_reg_tunned_best_model.predict(xtest_scaled)
```

```
Best Hyperparameters: {'estimator__C': 100, 'estimator__penalty': 'l2'}
```

Visualizing the confusion matrix with a heatmap to assess model performance

```
log_tunned_cm = confusion_matrix(ytest, logistic_reg_tunned_ytest_prediction)

plt.figure(figsize=(8, 6))
sns.heatmap(log_tunned_cm, annot=True, cmap='plasma', fmt='g',
            xticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'],
            yticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
logistic_reg_tunned_accuracy = accuracy_score(ytest, logistic_reg_tunned_ytest_prediction)
logistic_reg_tunned_precision = precision_score(ytest, logistic_reg_tunned_ytest_prediction, average = 'macro')  # Macro-averaged precision
logistic_reg_tunned_recall = recall_score(ytest, logistic_reg_tunned_ytest_prediction, average = 'macro')  # Macro-averaged recall
logistic_reg_tunned_f1 = f1_score(ytest, logistic_reg_tunned_ytest_prediction, average = 'macro')  # Macro-averaged F1-score

print('Logistic Regression After Hyperparameters [macro]')
print("Accuracy:", logistic_reg_tunned_accuracy)
print("Precision:", logistic_reg_tunned_precision)
print("Recall:", logistic_reg_tunned_recall)
print("F1-score:", logistic_reg_tunned_f1)
```

```
Logistic Regression After Hyperparameters [macro]
Accuracy: 0.6866028708133971
Precision: 0.68508947565785
Recall: 0.6905847953216374
F1-score: 0.6791395127081786
```

Naive Bayes

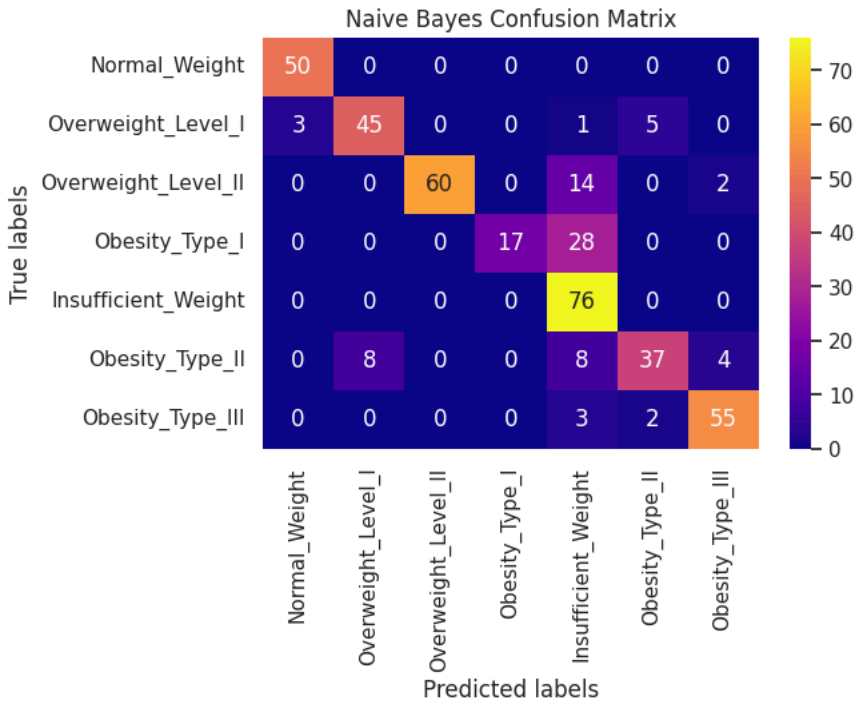
```
# Param grid for GridSearchCV
param_grid_naive_bayes = {
```

```
    'priors': [None, [0.25, 0.75], [0.5, 0.5]],
    'var_smoothing': [1e-9, 1e-8, 1e-7]
}

# Instantiate one instance of the GaussianNB class
naive_bayes_model = GaussianNB()

# Create a GridSearchCV object and fit it to the training data
naive_bayes_grid_search = GridSearchCV(
    estimator=naive_bayes_model,
    param_grid=param_grid_naive_bayes,
    scoring='accuracy',
    cv=5)

# Fit the model to the training data
naive_bayes_grid_search.fit(X=xtrain_scaled, y=ytrain)

# Print the best hyperparameters found by GridSearchCV
print('Best parameters', naive_bayes_grid_search.best_params_)

# Get the best model
naive_bayes_best_model = naive_bayes_grid_search.best_estimator_

# Evaluate the best model on the test set
naive_bayes_tunned_ytest_prediction = naive_bayes_best_model.predict(xtest_scaled)
```

```
Best parameters {'priors': None, 'var_smoothing': 1e-07}
```

Visualizing the confusion matrix with a heatmap to assess model performance

```
naive_bayes_tunned_cm = confusion_matrix(ytest, naive_bayes_tunned_ytest_prediction)

plt.figure(figsize=(6, 4))
sns.heatmap(naive_bayes_tunned_cm, annot=True, cmap='plasma', fmt='g',
            xticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'],
            yticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Naive Bayes Confusion Matrix')
plt.show()
```



```
# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
naive_bayes_accuracy = accuracy_score(ytest, naive_bayes_tunned_ytest_prediction)
naive_bayes_precision = precision_score(ytest, naive_bayes_tunned_ytest_prediction, average="macro")
naive_bayes_recall = recall_score(ytest, naive_bayes_tunned_ytest_prediction, average="macro")
naive_bayes_f1 = f1_score(ytest, naive_bayes_tunned_ytest_prediction, average="macro")

print('Naive Bayes Classifier after hyperparameter')
print("Accuracy:", naive_bayes_accuracy)
print("Precision:", naive_bayes_precision)
print("Recall:", naive_bayes_recall)
print("F1-score:", naive_bayes_f1)
```

```
Naive Bayes Classifier after hyperparameter
Accuracy: 0.8133971291866029
Precision: 0.8742309499964929
Recall: 0.7951963241436927
F1-score: 0.8031947962532969
```

Both Naive Bayes accuracy values are pretty similar which means the model did not overfit

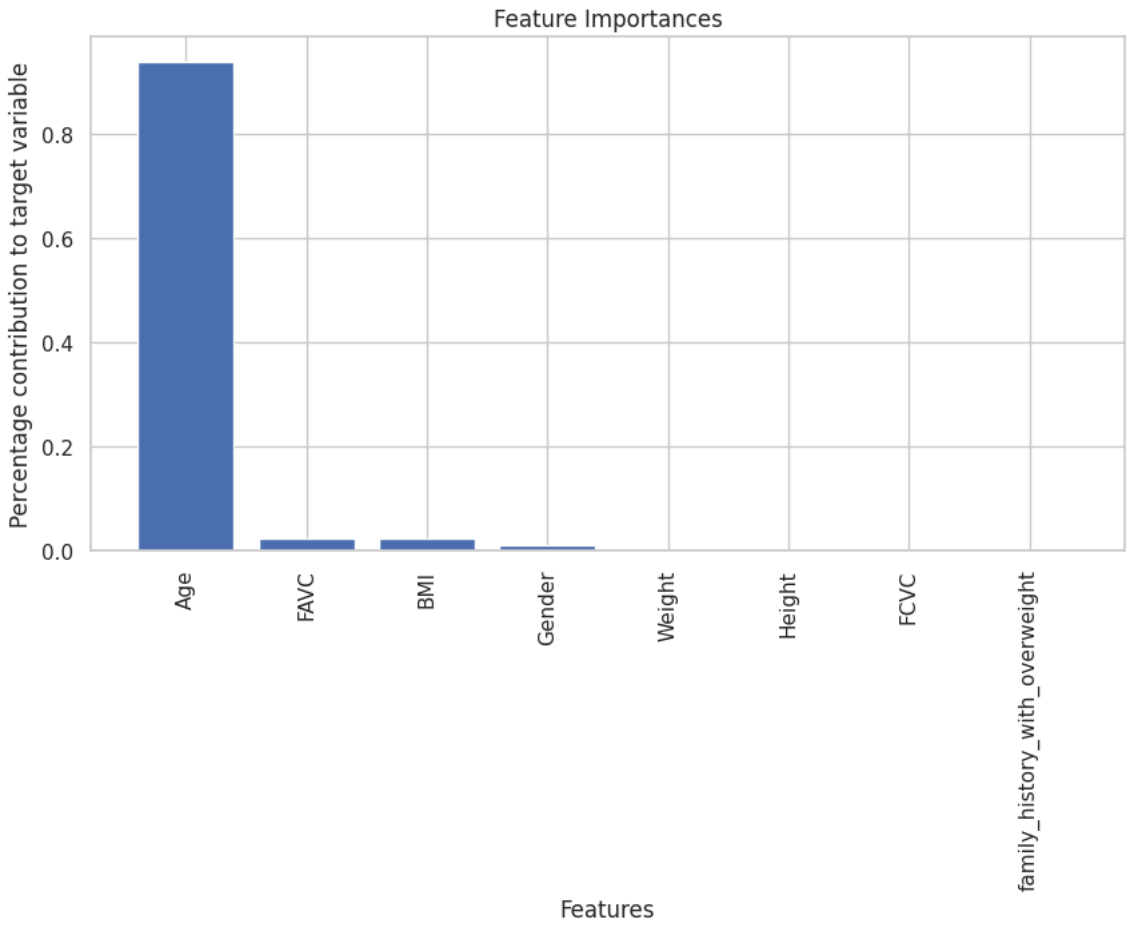
Decision Trees

```
# Param grid for GridSearchCV
param_grid_decision_trees = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'splitter': ['best','random'],
    'max_depth':['None', 5, 10,15],
    'min_samples_split':[2,4,6,8],
    'min_samples_leaf':[1,2,3]
}

# Instantiate one instance of the DecisionTreeClassifier class
decision_trees_model = DecisionTreeClassifier()

# Create a GridSearchCV object and fit it to the training data
decision_trees_grid_search = GridSearchCV(DecisionTreeClassifier(),
                                          param_grid = param_grid_decision_trees,
                                          scoring = 'accuracy',
                                          cv = 5)

# Fit the model to the training data
decision_trees_grid_search.fit(X=xtrain_scaled,y=ytrain)

# Print the best hyperparameters found by GridSearchCV
print("best parameters : ", decision_trees_grid_search.best_params_)

# Get the best model
decision_trees_best_model = decision_trees_grid_search.best_estimator_

# Evaluate the best model on the test set
decision_trees_tunned_ytest_prediction = decision_trees_best_model.predict(xtest_scaled)
```

```
best parameters :  {'criterion': 'log_loss', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

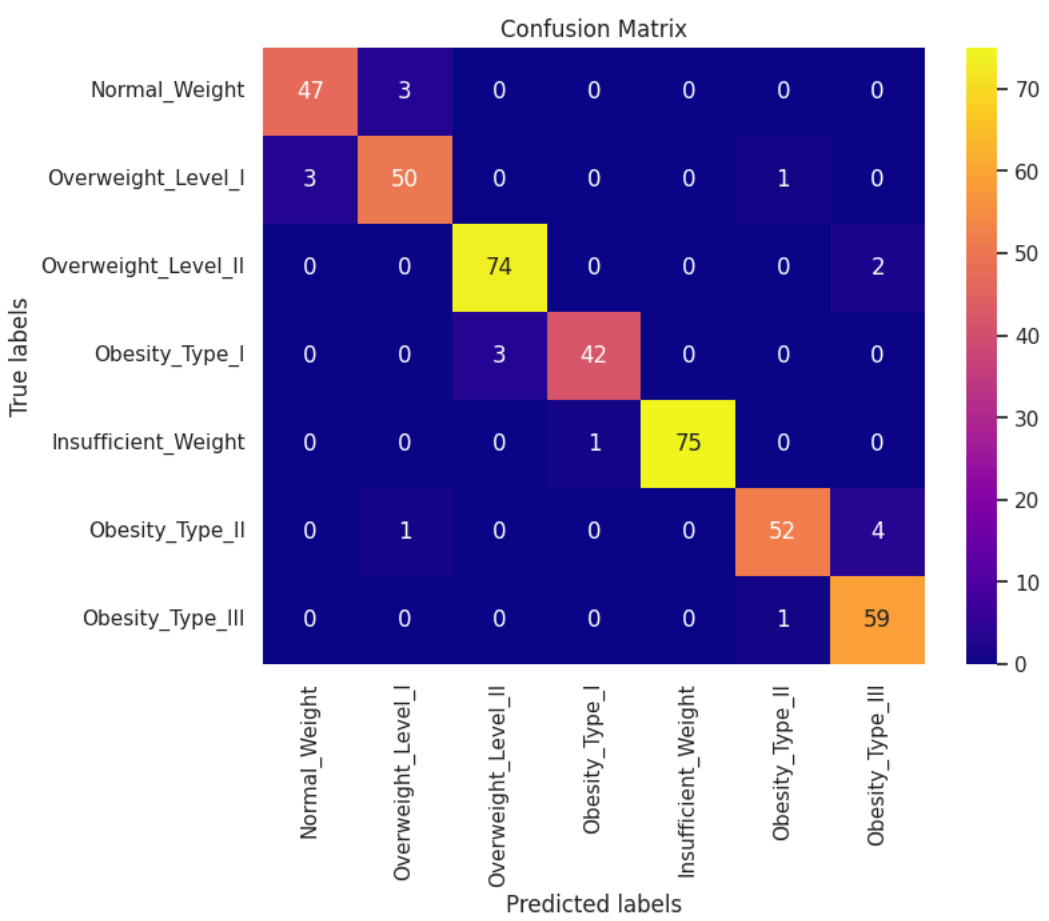
feature importances for decision tree best model

```
importances = decision_trees_best_model.feature_importances_
features = df.columns[:-1]
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10, 5))
plt.title("Feature Importances")
plt.bar(range(len(importances)), importances[indices])
plt.xticks(range(len(importances)), features[indices], rotation=90)
plt.xlabel('Features')
plt.ylabel('Percentage contribution to target variable')
plt.show()

# NOTE: 1) Only a few models inherently have feature importances, one is decision trees
#       2) The other models we have chosen does not show this, so we are only providing here
```



Visualizing the confusion matrix with a heatmap to assess model performance

```
decision_trees_tunned_cm = confusion_matrix(ytest, decision_trees_tunned_ytest_prediction)

plt.figure(figsize=(8, 6))
sns.heatmap(decision_trees_tunned_cm, annot=True, cmap='plasma', fmt='g',
            xticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'],
            yticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
dt_accuracy_hp = accuracy_score(ytest, decision_trees_tunned_ytest_prediction)
dt_precision_hp = precision_score(ytest, decision_trees_tunned_ytest_prediction, average="macro")
dt_recall_hp = recall_score(ytest, decision_trees_tunned_ytest_prediction, average="macro")
dt_f1_hp = f1_score(ytest, decision_trees_tunned_ytest_prediction, average="macro")

print('Decision Trees Classifier after hyperparameter')
print("Accuracy:", dt_accuracy_hp)
print("Precision:", dt_precision_hp)
print("Recall:", dt_recall_hp)
print("F1-score:", dt_f1_hp)
```

```
Decision Trees Classifier after hyperparameter
Accuracy: 0.9545454545454546
Precision: 0.95348062052381
Recall: 0.950771372876636
F1-score: 0.9517294374699906
```

KNN

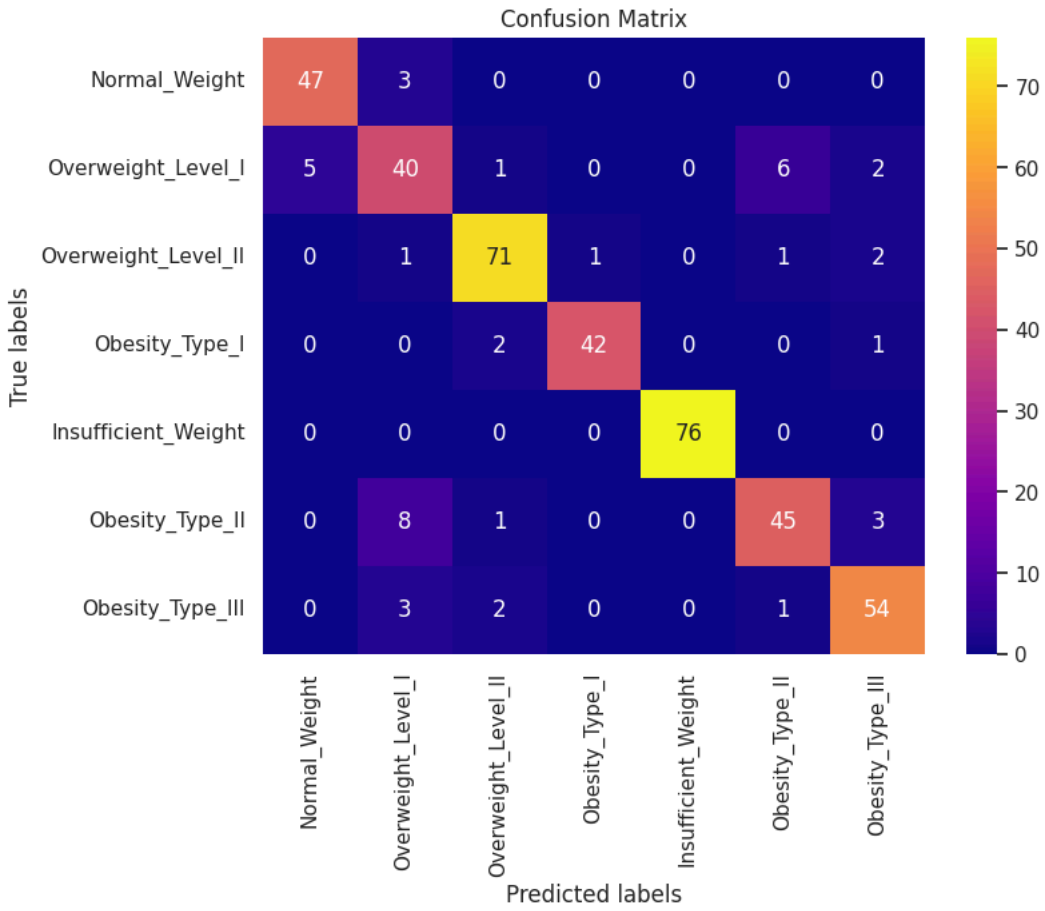
```
# Param grid for GridSearchCV
param_grid_knn = {
    "n_neighbors": range(1, 51),
    "weights": ['uniform', 'distance'],
    "p": [1, 2]
}

# Instantiate one instance of the KNeighborsClassifier class
knn_model = KNeighborsClassifier()

# Create a GridSearchCV object and fit it to the training data
knn_grid_search = GridSearchCV(estimator = knn_model,
                               param_grid = param_grid_knn,
                               scoring = 'accuracy',
                               cv = 5)
```

```
# Fit the model to the training data
knn_grid_search.fit(xtrain_scaled, ytrain)

# Print the best hyperparameters found by GridSearchCV
print("Best Parameters:", knn_grid_search.best_params_)

# Get the best model
knn_tunned_best_model = knn_grid_search.best_estimator_

# Evaluate the best model on the test set
knn_tunned_ypred = knn_tunned_best_model.predict(xtest_scaled)
```

```
Best Parameters: {'n_neighbors': 4, 'p': 1, 'weights': 'distance'}
```

Visualizing the confusion matrix with a heatmap to assess model performance

```
knn_tunned_cm = confusion_matrix(ytest, knn_tunned_ypred)

plt.figure(figsize=(8, 6))
sns.heatmap(knn_tunned_cm, annot=True, cmap='plasma', fmt='g',
            xticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'],
            yticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
knn_tunned_accuracy = accuracy_score(ytest, knn_tunned_ypred)
knn_tunned_precision = precision_score(ytest, knn_tunned_ypred, average='macro')
knn_tunned_recall = recall_score(ytest, knn_tunned_ypred, average='macro')
knn_tunned_f1 = f1_score(ytest, knn_tunned_ypred, average='macro')
```

```
print('K-Nearest Neighbors after Hyperparameter Tuning [macro]')
print("Accuracy:", knn_tunned_accuracy)
print("Precision:", knn_tunned_precision)
print("Recall:", knn_tunned_recall)
print("F1-score:", knn_tunned_f1)
```

```
K-Nearest Neighbors after Hyperparameter Tuning [macro]
Accuracy: 0.8971291866028708
Precision: 0.8928521907074833
Recall: 0.8911083263714844
F1-score: 0.8916559044442052
```

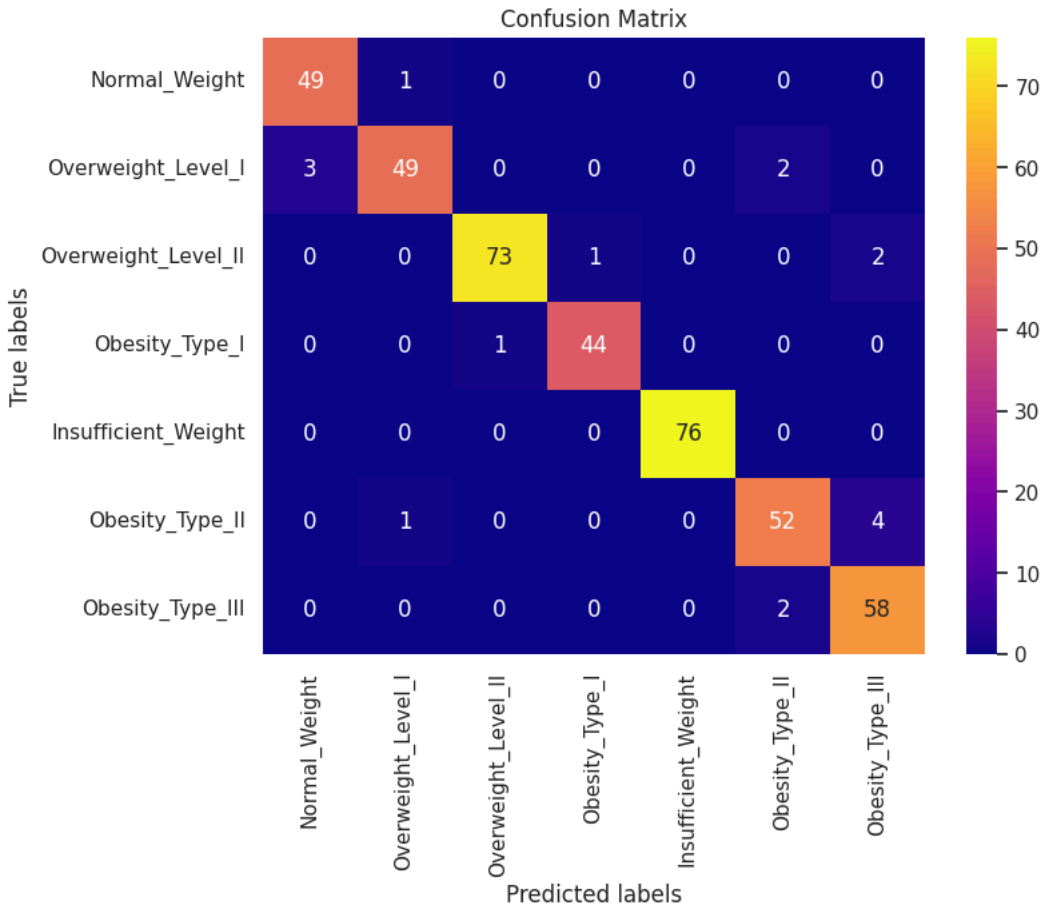
SVM

```
# Param grid for GridSearchCV
param_grid_svm = {
    'C': [0.1, 1, 10, 100],            # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Kernel type
    'gamma': [0.1, 0.01, 0.001, 0.0001],  # Kernel coefficient
    'degree': [2, 3, 4],                # Degree of the polynomial kernel
}

# Instantiate one instance of the SVC class
svm_model = SVC()

# Create a GridSearchCV object and fit it to the training data
svm_grid_search = GridSearchCV(estimator = svm_model,
                               param_grid = param_grid_svm,
                               cv = 5,
                               scoring = 'accuracy')

# Fit the model to the training data
svm_grid_search.fit(xtrain_scaled, ytrain)

# Print the best hyperparameters found by GridSearchCV
print("Best Parameters:", svm_grid_search.best_params_)

# Get the best model
svm_tunned_best_model = svm_grid_search.best_estimator_

# Evaluate the best model on the test set
svm_tunned_ytest_prediction = svm_tunned_best_model.predict(xtest_scaled)
```

```
Best Parameters: {'C': 10, 'degree': 2, 'gamma': 0.1, 'kernel': 'linear'}
```

Visualizing the confusion matrix with a heatmap to assess model performance

```
svm_tunned_cm = confusion_matrix(ytest, svm_tunned_ytest_prediction)

plt.figure(figsize=(8, 6))
sns.heatmap(svm_tunned_cm, annot=True, cmap='plasma', fmt='g',
            xticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'],
            yticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
svm_tunned_accuracy = accuracy_score(ytest, svm_tunned_ytest_prediction)
svm_tunned_precision = precision_score(ytest, svm_tunned_ytest_prediction, average = 'macro')  # macro-averaged precision
svm_tunned_recall = recall_score(ytest, svm_tunned_ytest_prediction, average = 'macro')  # macro-averaged recall
svm_tunned_f1 = f1_score(ytest, svm_tunned_ytest_prediction, average = 'macro')  # macro-averaged F1-score

print('Support Vector Machine After Hyperparameters [macro]')
print("Accuracy:", svm_tunned_accuracy)
print("Precision:", svm_tunned_precision)
print("Recall:", svm_tunned_recall)
print("F1-score:", svm_tunned_f1)
```

```
Support Vector Machine After Hyperparameters [macro]
Accuracy: 0.9593301435406698
Precision: 0.9574539569812679
Recall: 0.9578084099136731
F1-score: 0.9572952302055088
```

XGBoost

```
# Param grid for GridSearchCV
param_grid_xgb = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01],
    'n_estimators': [100, 200],
    'gamma': [0, 0.1, 0.2]

}

# Instantiate one instance of the XGBClassifier class
xgb_model = xgb.XGBClassifier()

# Create a GridSearchCV object and fit it to the training data
xgb_grid_search = GridSearchCV(xgb_model,
                               param_grid = param_grid_xgb,
                               cv = 5,
```

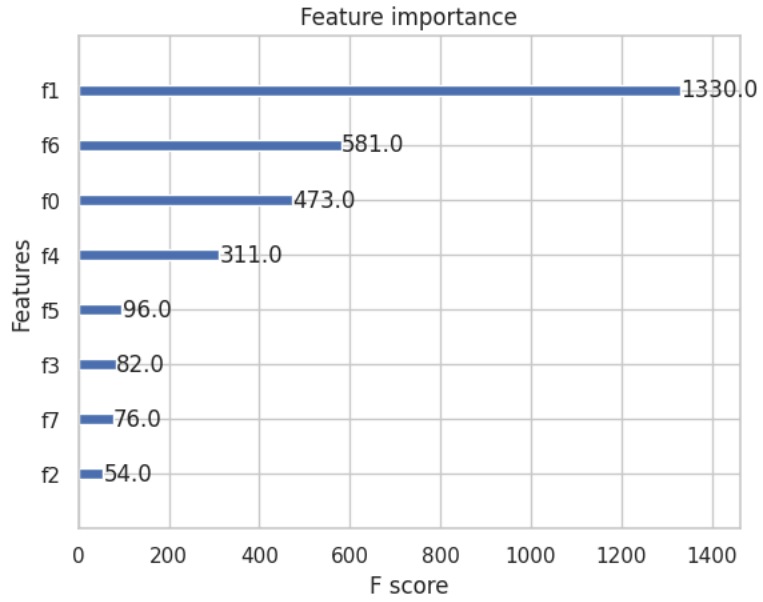
```
                                scoring = 'accuracy')

# Fit the model to the training data
xgb_grid_search.fit(xtrain_scaled, ytrain)

# Print the best hyperparameters found by GridSearchCV
print("Best Parameters:", xgb_grid_search.best_params_)

# Get the best model
xgb_tunned_best_model = xgb_grid_search.best_estimator_

# Evaluate the best model on the test set
xgb_tunned_ypred = xgb_tunned_best_model.predict(xtest_scaled)
```

```
Best Parameters: {'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
```

feature importances for XGBoost best model

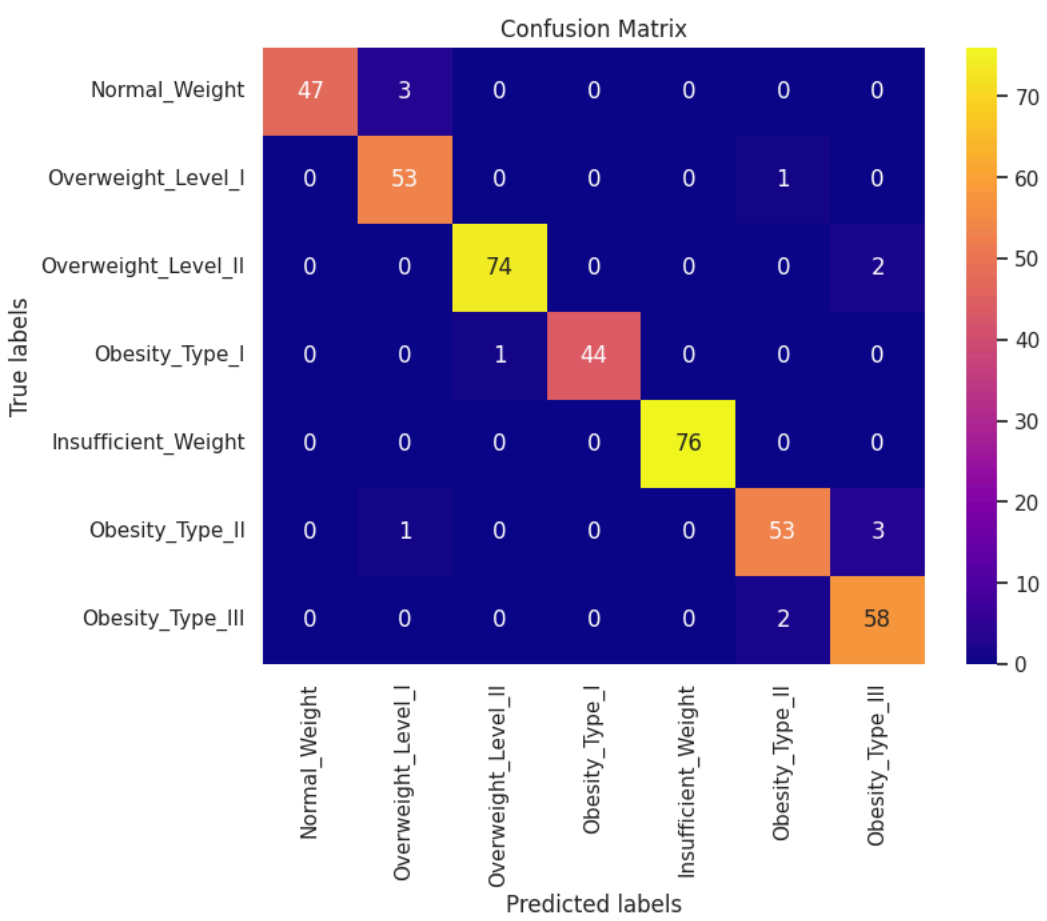
```
plt.figure(figsize = (12,10))
xgb.plot_importance(xgb_tunned_best_model)
plt.show()
```

```
<Figure size 1200x1000 with 0 Axes>
```



Visualizing the confusion matrix with a heatmap to assess model performance

```
xgb_tunned_cm = confusion_matrix(ytest, xgb_tunned_ypred)

plt.figure(figsize=(8, 6))
sns.heatmap(xgb_tunned_cm, annot=True, cmap='plasma', fmt='g',
            xticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'],
            yticklabels=['Normal_Weight', 'Overweight_Level_I', 'Overweight_Level_II',
       'Obesity_Type_I', 'Insufficient_Weight', 'Obesity_Type_II',
       'Obesity_Type_III'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Calculating and printing evaluation metrics (Accuracy, Precision, Recall, and F1-score)
xgboost_tunned_accuracy = accuracy_score(ytest, xgb_tunned_ypred)
xgboost_tunned_precision = precision_score(ytest, xgb_tunned_ypred, average = "macro")
xgboost_tunned_recall = recall_score(ytest, xgb_tunned_ypred, average = "macro")
xgboost_tunned_f1 = f1_score(ytest, xgb_tunned_ypred, average = "macro")

print('XGBoost Classifier')
print("Accuracy:", xgboost_tunned_accuracy)
print("Precision:", xgboost_tunned_precision)
print("Recall:", xgboost_tunned_precision)
print("F1-score:", xgboost_tunned_f1)
```

```
XGBoost Classifier
Accuracy: 0.9688995215311005
Precision: 0.969079245733381
Recall: 0.969079245733381
F1-score: 0.967723734773971
```

Final performance metrics. Before vs After hyperparameters

```
# Before hyperparameters
before_hyperparameters = {
    'Model': ['Naive Bayes', 'Decision Tree', 'KNN', 'Logistic Regression', 'SVM','XGBOOST'],
    'Accuracy': [nb_accuracy, dt_accuracy, knn_accuracy, logistic_reg_accuracy, svm_accuracy,xgboost_accuracy],
    'Precision': [nb_precision, dt_precision, knn_precision, logistic_reg_precision, svm_precision,xgboost_precision ],
    'Recall': [nb_recall, dt_recall, knn_recall, logistic_reg_recall, svm_recall,xgboost_recall],
    'F1': [nb_f1, dt_f1, knn_f1, logistic_reg_f1, svm_f1,xgboost_f1]
}

# After hyperparameters
after_hyperparameters = {
    'Model': ['Naive Bayes (Tuned)', 'Decision Tree (Tuned)', 'KNN (Tuned)', 'Logistic Regression (Tuned)', 'SVM (Tuned)','XGBOOST (Tuned)']
    'Accuracy': [naive_bayes_accuracy, dt_accuracy_hp, knn_tunned_accuracy, logistic_reg_tunned_accuracy, svm_tunned_accuracy,xgboost_tunned
    'Precision': [naive_bayes_precision, dt_precision_hp, knn_tunned_precision, logistic_reg_tunned_precision, svm_tunned_precision,xgboost_
    'Recall': [naive_bayes_recall, dt_recall_hp, knn_tunned_recall, logistic_reg_tunned_recall, svm_tunned_recall,xgboost_tunned_recall ],
    'F1': [naive_bayes_f1, dt_f1_hp, knn_tunned_f1, logistic_reg_tunned_f1, svm_tunned_f1,xgboost_tunned_f1]
```

```
}

# Create pandas DataFrames
before_df = pd.DataFrame(before_hyperparameters)
after_df = pd.DataFrame(after_hyperparameters)


# Display the DataFrames
print("Before Hyperparameters:")
print(before_df)
```

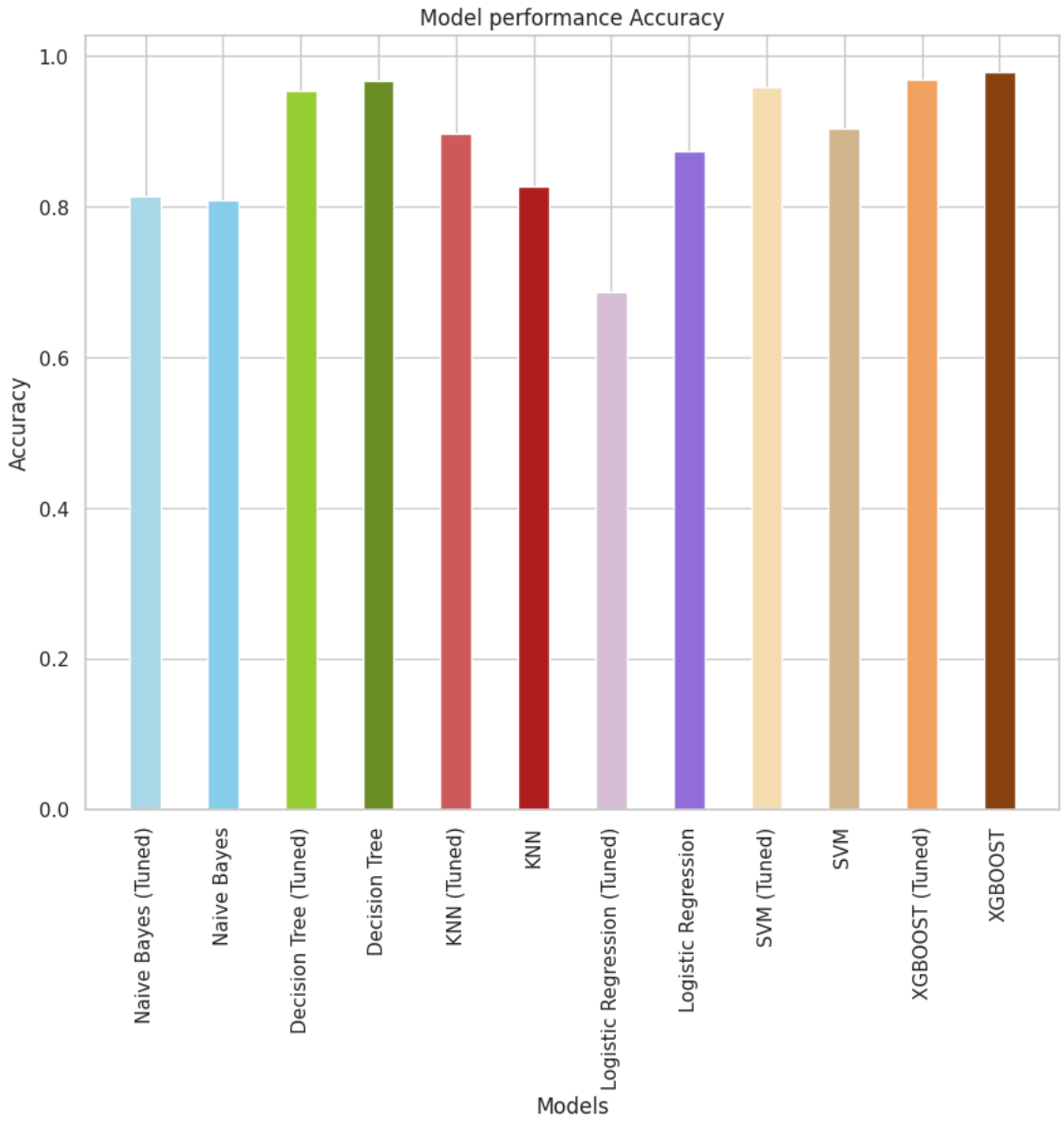
```
Before Hyperparameters:
                   Model  Accuracy  Precision    Recall        F1
0          Naive Bayes  0.808612   0.871883  0.791479  0.800633
1        Decision Tree  0.966507   0.966506  0.965241  0.965269
2                  KNN  0.827751   0.817377  0.821028  0.818217
3  Logistic Regression  0.873206   0.869451  0.871637  0.868248
4                  SVM  0.904306   0.896448  0.900201  0.897924
5              XGBOOST  0.978469   0.977668  0.976853  0.977137
```

```
print('After Hyperparameters:')
print(after_df)
```

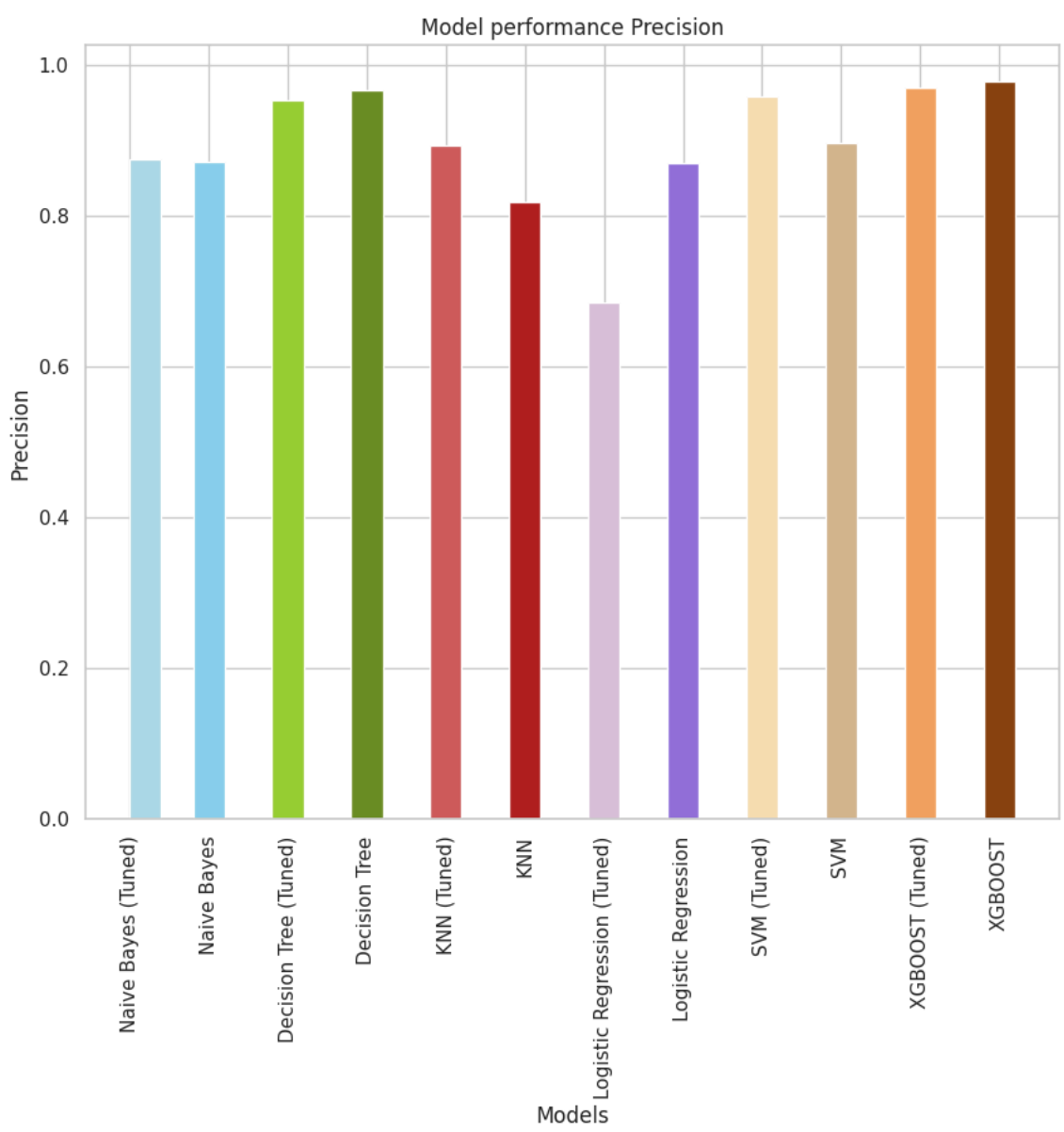
```
After Hyperparameters:
                           Model  Accuracy  Precision    Recall        F1
0          Naive Bayes (Tuned)  0.813397   0.874231  0.795196  0.803195
1        Decision Tree (Tuned)  0.954545   0.953481  0.950771  0.951729
2                  KNN (Tuned)  0.897129   0.892852  0.891108  0.891656
3  Logistic Regression (Tuned)  0.686603   0.685089  0.690585  0.679140
4                  SVM (Tuned)  0.959330   0.957454  0.957808  0.957295
5              XGBOOST (Tuned)  0.968900   0.969079  0.967062  0.967724
```

Graph to visually compare - the lighter colors represent the tuned and the darker colors are the not-tuned hyperparameter models

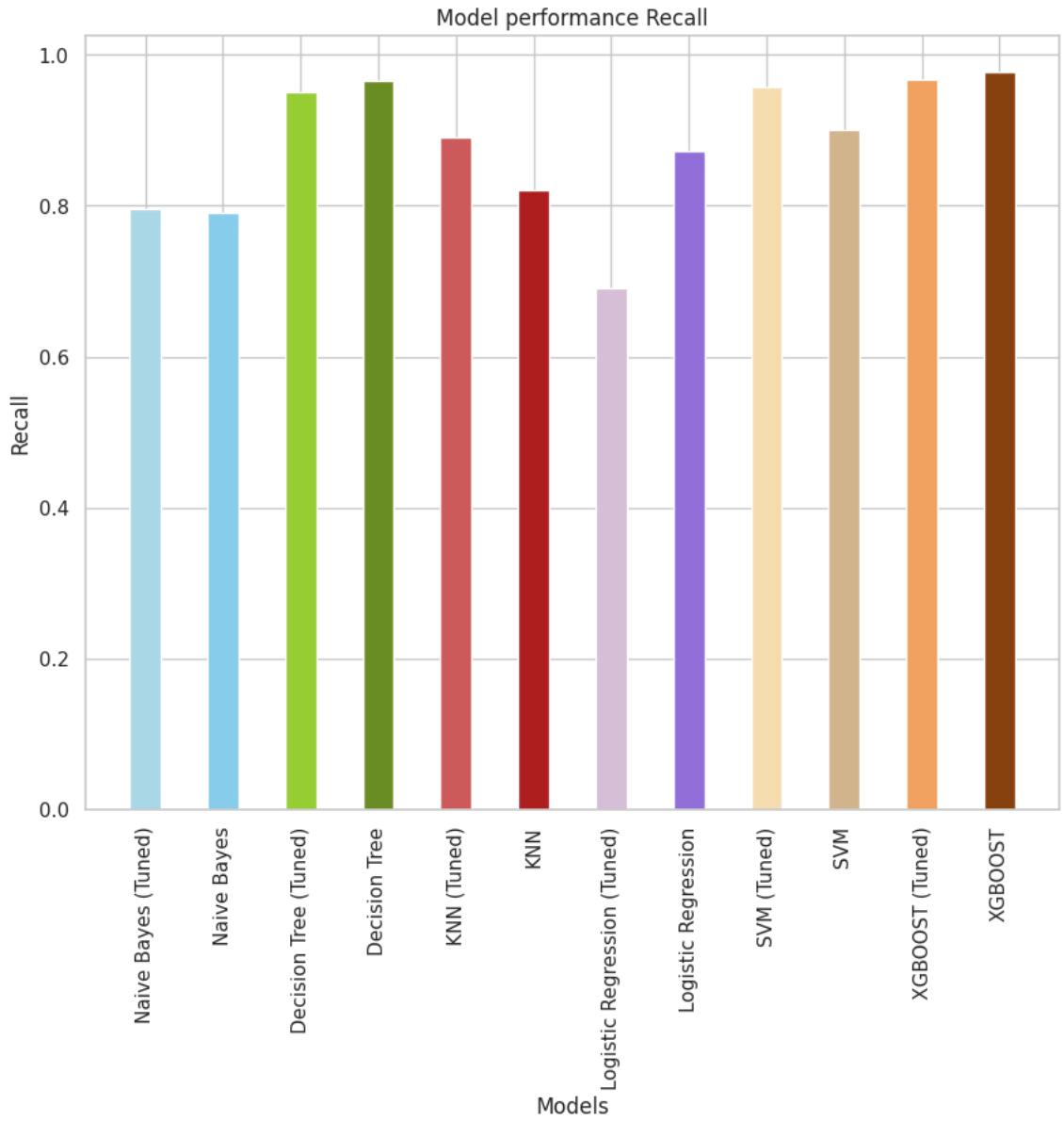
```
fig = plt.figure(figsize = (10, 8))
plt.bar('Naive Bayes (Tuned)',naive_bayes_accuracy, width = 0.4, color='lightblue')
plt.bar('Naive Bayes',nb_accuracy, width = 0.4, color='skyblue')

plt.bar('Decision Tree (Tuned)', dt_accuracy_hp, width = 0.4, color='yellowgreen')
plt.bar('Decision Tree', dt_accuracy, width = 0.4, color='olivedrab')

plt.bar('KNN (Tuned)', knn_tunned_accuracy, width = 0.4, color='indianred')
plt.bar('KNN', knn_accuracy, width = 0.4, color='firebrick')

plt.bar('Logistic Regression (Tuned)', logistic_reg_tunned_accuracy, width = 0.4, color='thistle')
plt.bar('Logistic Regression', logistic_reg_accuracy, width = 0.4, color='mediumpurple')

plt.bar(' SVM (Tuned)', svm_tunned_accuracy, width = 0.4, color = 'wheat')
plt.bar(' SVM', svm_accuracy, width = 0.4, color='tan')

plt.bar('XGBOOST (Tuned)',xgboost_tunned_accuracy, width = 0.4, color='sandybrown')
plt.bar('XGBOOST',xgboost_accuracy, width = 0.4, color='saddlebrown')

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model performance Accuracy')

plt.xticks(rotation=90)

plt.show()
```



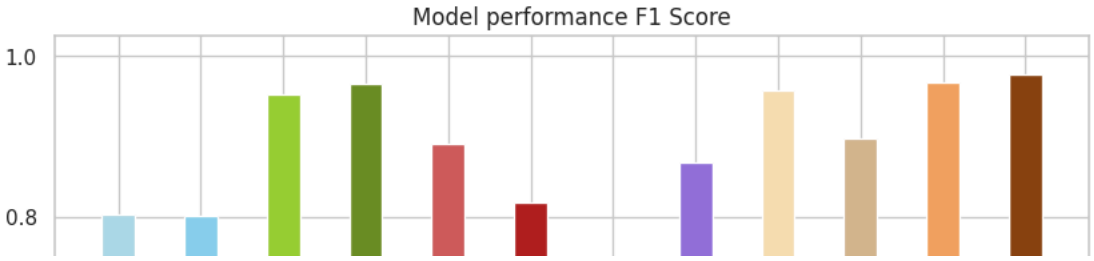

```
fig = plt.figure(figsize = (10, 8))
plt.bar('Naive Bayes (Tuned)',naive_bayes_precision, width = 0.4, color='lightblue',align='edge')
plt.bar('Naive Bayes',nb_precision, width = 0.4, color='skyblue')

plt.bar('Decision Tree (Tuned)', dt_precision_hp, width = 0.4, color='yellowgreen')
plt.bar('Decision Tree', dt_precision, width = 0.4, color='olivedrab')

plt.bar('KNN (Tuned)', knn_tunned_precision, width = 0.4, color='indianred')
plt.bar('KNN', knn_precision, width = 0.4, color='firebrick')

plt.bar('Logistic Regression (Tuned)', logistic_reg_tunned_precision , width = 0.4, color='thistle')
plt.bar('Logistic Regression', logistic_reg_precision, width = 0.4, color='mediumpurple')

plt.bar(' SVM (Tuned)', svm_tunned_precision, width = 0.4, color = 'wheat')
plt.bar(' SVM', svm_precision, width = 0.4, color='tan')

plt.bar('XGBOOST (Tuned)',xgboost_tunned_precision, width = 0.4, color='sandybrown')
plt.bar('XGBOOST',xgboost_precision, width = 0.4, color='saddlebrown')

plt.xlabel('Models')
plt.ylabel('Precision')
plt.title('Model performance Precision')

plt.xticks(rotation=90)

plt.show()
```




```
fig = plt.figure(figsize=(10, 8))
plt.bar('Naive Bayes (Tuned)', naive_bayes_recall, width=0.4, color='lightblue')
plt.bar('Naive Bayes', nb_recall, width=0.4, color='skyblue')

plt.bar('Decision Tree (Tuned)', dt_recall_hp, width=0.4, color='yellowgreen')
plt.bar('Decision Tree', dt_recall, width=0.4, color='olivedrab')

plt.bar('KNN (Tuned)', knn_tunned_recall, width=0.4, color='indianred')
plt.bar('KNN', knn_recall, width=0.4, color='firebrick')

plt.bar('Logistic Regression (Tuned)', logistic_reg_tunned_recall, width=0.4, color='thistle')
plt.bar('Logistic Regression', logistic_reg_recall, width=0.4, color='mediumpurple')

plt.bar(' SVM (Tuned)', svm_tunned_recall, width=0.4, color='wheat')
plt.bar(' SVM', svm_recall, width=0.4, color='tan')

plt.bar('XGBOOST (Tuned)',xgboost_tunned_recall, width = 0.4, color='sandybrown')
plt.bar('XGBOOST',xgboost_recall, width = 0.4, color='saddlebrown')

plt.xlabel('Models')
plt.ylabel('Recall')
plt.title('Model performance Recall')

plt.xticks(rotation=90)

plt.show()
```




```
fig = plt.figure(figsize=(10, 8))
plt.bar('Naive Bayes (Tuned)', naive_bayes_f1, width=0.4, color='lightblue')
plt.bar('Naive Bayes', nb_f1, width=0.4, color='skyblue')

plt.bar('Decision Tree (Tuned)', dt_f1_hp, width=0.4, color='yellowgreen')
plt.bar('Decision Tree', dt_f1, width=0.4, color='olivedrab')

plt.bar('KNN (Tuned)', knn_tunned_f1, width=0.4, color='indianred')
plt.bar('KNN', knn_f1, width=0.4, color='firebrick')

plt.bar('Logistic Regression (Tuned)', logistic_reg_tunned_f1, width=0.4, color='thistle')
plt.bar('Logistic Regression', logistic_reg_f1, width=0.4, color='mediumpurple')

plt.bar(' SVM (Tuned)', svm_tunned_f1, width=0.4, color='wheat')
plt.bar(' SVM', svm_f1, width=0.4, color='tan')

plt.bar('XGBOOST (Tuned)',xgboost_tunned_f1, width = 0.4, color='sandybrown')
plt.bar('XGBOOST',xgboost_f1, width = 0.4, color='saddlebrown')

plt.xlabel('Models')
plt.ylabel('F1 Score')
plt.title('Model performance F1 Score')

plt.xticks(rotation=90)

plt.show()
```


Model performance F1 Score
1.0
0.8