16MDS55 MACHINE LEARNING LABORATORY

CONTINUOUS ASSESSMENT TEST II – NOVEMBER 2020

Estimation Of Obesity Levels Based On Eating Habits and Physical condition

DONE BY:
Abhishek R - 1832001
Sahana S R - 1832043

Abstract:

This project presents 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 the output labeled with the class variable NObesity (Obesity Level), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III. This data can be used to generate intelligent computational tools to identify the obesity level of an individual and to build models that monitor obesity levels.

Data Description:

This data for the estimation of obesity levels in people from the countries of Mexico, Peru and Colombia, with ages between 14 and 61 and diverse eating habits and physical condition. The data was collected using a web platform with a survey where anonymous users answered each question, then the information was processed.

The attributes related with eating habits are:

- Frequent consumption of high caloric food (FAVC)
- Frequency of consumption of vegetables (FCVC)
- Number of main meals (NCP)
- Consumption of food between meals (CAEC)
- Consumption of water daily (CH20)
- Consumption of alcohol (CALC).

The attributes related with the physical condition are:

- Calories consumption monitoring (SCC)
- Physical activity frequency (FAF)
- Time using technology devices (TUE)
- Transportation used (MTRANS)

Other variables obtained were:

- Gender
- Age
- Height
- Weight

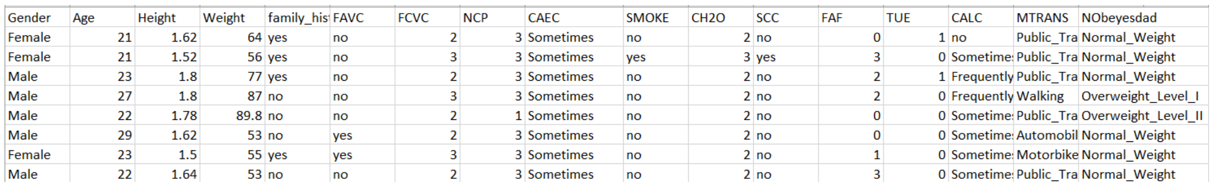
Finally, all data was labeled and the class variable NObesity was created with the values of BMI as follows:

- Underweight Less than 18.5

- Normal 18.5 to 24.9
- Overweight 25.0 to 29.9
- Obesity I 30.0 to 34.9
- Obesity II 35.0 to 39.9
- Obesity III Higher than 40

The data contains numerical data and continuous data, so it can be used for analysis based on algorithms of classification, prediction, segmentation and association.

Data Source and Sample:



Gender	Age	Height	Weight	family_his	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
Female	21	1.62	64	yes	no	2	3	Sometimes	no	2	no	0	1	no	Public_Tra	Normal_Weight
Female	21	1.52	56	yes	no	3	3	Sometimes	yes	3	yes	3	0	Sometime:	Public_Tra	Normal_Weight
Male	23	1.8	77	yes	no	2	3	Sometimes	no	2	no	2	1	Frequently	Public_Tra	Normal_Weight
Male	27	1.8	87	no	no	3	3	Sometimes	no	2	no	2	0	Frequently	Walking	Overweight_Level_I
Male	22	1.78	89.8	no	no	2	1	Sometimes	no	2	no	0	0	Sometime:	Public_Tra	Overweight_Level_II
Male	29	1.62	53	no	yes	2	3	Sometimes	no	2	no	0	0	Sometime:	Automobil	Normal_Weight
Female	23	1.5	55	yes	yes	3	3	Sometimes	no	2	no	1	0	Sometime:	Motorbike	Normal_Weight
Male	22	1.64	53	no	no	2	3	Sometimes	no	2	no	3	0	Sometime:	Public_Tra	Normal_Weight

Source:
https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition+#

Concepts used:

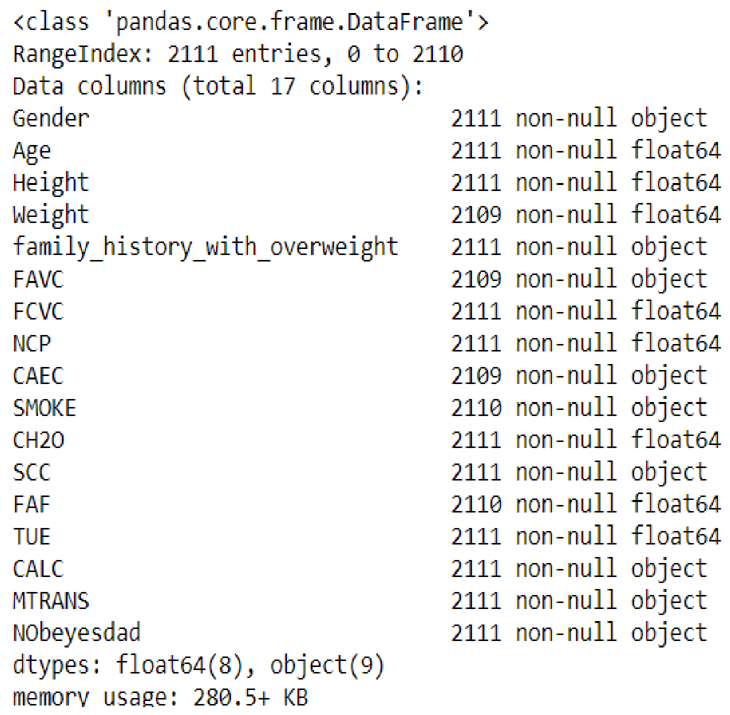
1. Exploratory data analysis
2. Recursive Feature Elimination
3. Bias Variance Trade - Off
4. Hyper parameter tuning and Best model selection
5. Support Vector Classifier
6. Logistic regression
7. K Nearest Neighbours
8. Decision Trees
9. Random Forest
10. Feature Importance Analysis
11. 10 fold Cross Validation

Project Building:

Importing necessary libraries and data:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
df=pd.read_csv("ObesityDataSet_raw_and_data_sinthetic.csv")
df.head()
df.info()
```

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
Gender                            2111 non-null object
Age                               2111 non-null float64
Height                            2111 non-null float64
Weight                            2109 non-null float64
family_history_with_overweight    2111 non-null object
FAVC                              2109 non-null object
FCVC                              2111 non-null float64
NCP                               2111 non-null float64
CAEC                              2109 non-null object
SMOKE                             2110 non-null object
CH2O                              2111 non-null float64
SCC                               2111 non-null object
FAF                               2110 non-null float64
TUE                               2111 non-null float64
CALC                              2111 non-null object
MTRANS                            2111 non-null object
NObeyesdad                        2111 non-null object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

Inference:

From the above output it can be seen that there are a few values missing in some columns. These missing values are filled in appropriately in the Data preprocessing step

Data Preprocessing:

In this section , all the missing values are filled appropriately. If the missing values are numerical, they are filled with mean values of the corresponding column and if the missing values are categorical then they are filled with mode of the corresponding column

```
df.isnull().any()
X = df.drop("NObeyesdad",axis=1)
y = df["NObeyesdad"]
catagorical =list( X.select_dtypes(include='object').columns )
numerical = list( X.select_dtypes(exclude='object').columns )
for cat in catagorical:
    df[cat].fillna(df[cat].mode()[0],inplace=True)
for num in numerical:
    df[num].fillna(df[num].mean(),inplace=True)
df.isnull().any()
df_new=df.copy(deep=True)
```

Output:

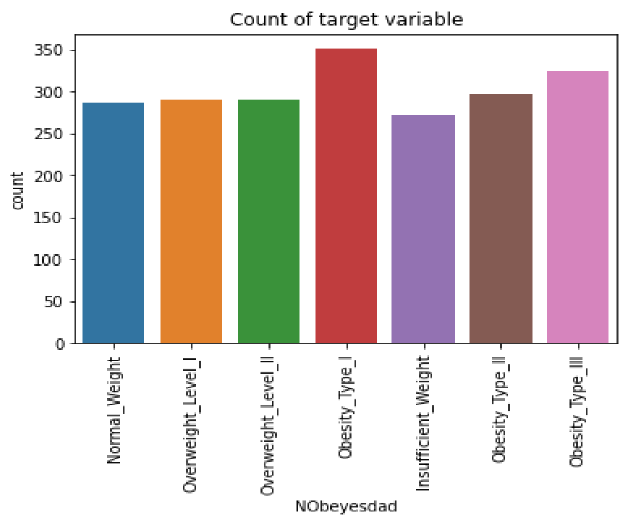
```
Gender                            False
Age                               False
Height                            False
Weight                            False
family_history_with_overweight    False
FAVC                              False
FCVC                              False
NCP                               False
CAEC                              False
SMOKE                             False
CH2O                              False
SCC                               False
FAF                               False
TUE                               False
CALC                              False
MTRANS                            False
NObeyesdad                        False
dtype: bool
```

Inference:

The data is now clean and it does not have any missing values

Exploratory Data Analysis:

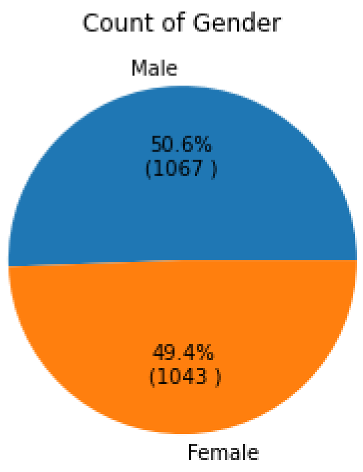
```
sns.countplot(df["NObeyesdad"])
plt.xticks(rotation="vertical")
plt.title("Count of target variable")
```



Inference:

From this graph it can be seen that all the categories of the output were given equal weightage while collecting data and there is no imbalance in the data

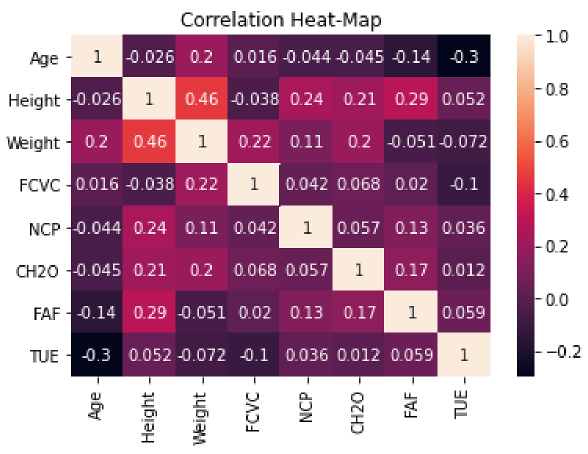
```
f_count=df['Gender'].value_counts()
def func(pct, allvalues):
    absolute = int(pct / 100.*np.sum(allvalues))
    return "{:.1f}%\n({:d} )".format(pct, absolute)
plt.pie(f_count,labels=f_count.index,autopct = lambda pct: func(pct,
f_count))
plt.title("Count of Gender")
```



Inference:

From this graph it can be seen that equal weightage has been given to both males and females during the data collection. Hence the data is unbiased in terms of gender

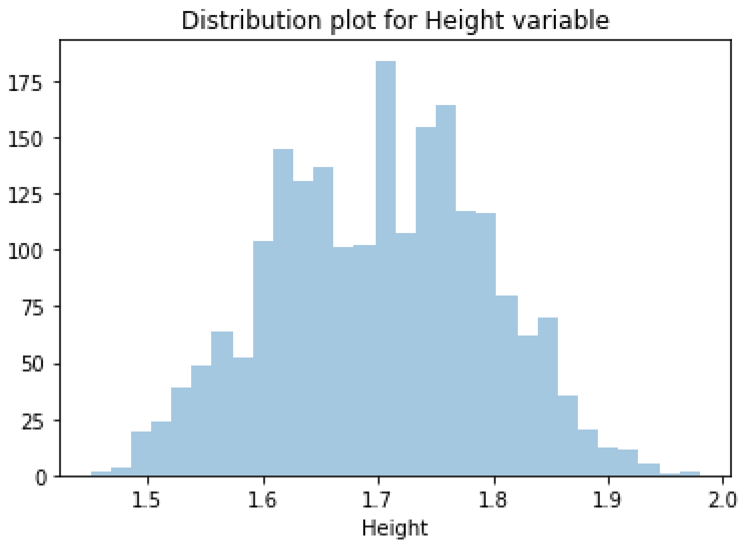
```
sns.heatmap(df.corr(),annot=True)
plt.title("Correlation Heat-Map")
```



Inference:

This graph shows the correlation between all the numeric variables. As we can see, there is no multicollinearity in the data since there are no inter correlation between the features

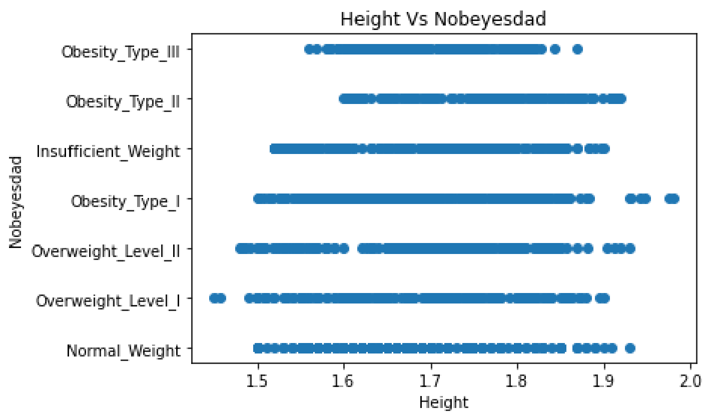
```
sns.distplot(df['Height'],kde=False,bins=30)
plt.title("Distribution plot for Height variable")
```



Inference:

This graph shows the distribution of height in the data. The height ranges from 1.5 to 1.9 meters on average and there are more number of people with the height between 1.7 to 1.75

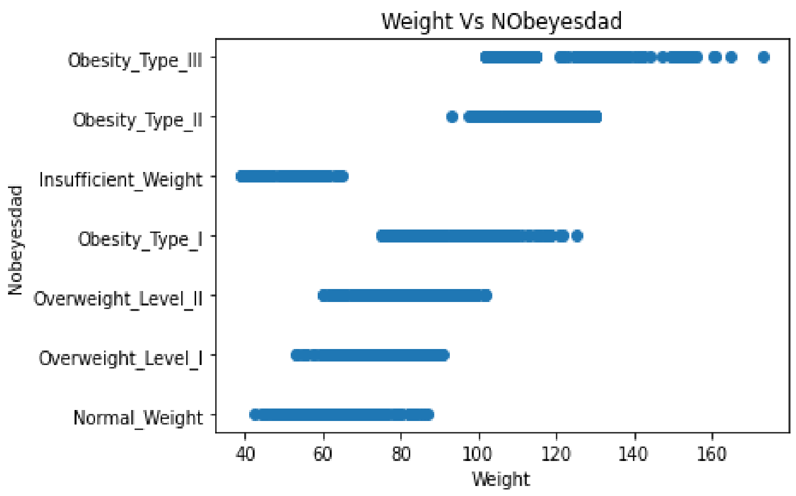
```
plt.scatter(x='Height',y='NObeyesdad',data=df)
plt.xlabel('Height')
plt.ylabel('Nobeyesdad')
plt.title("Height Vs Nobeyesdad")
```



Inference:

This graph says that the distribution of height is even to all the categories of obesity

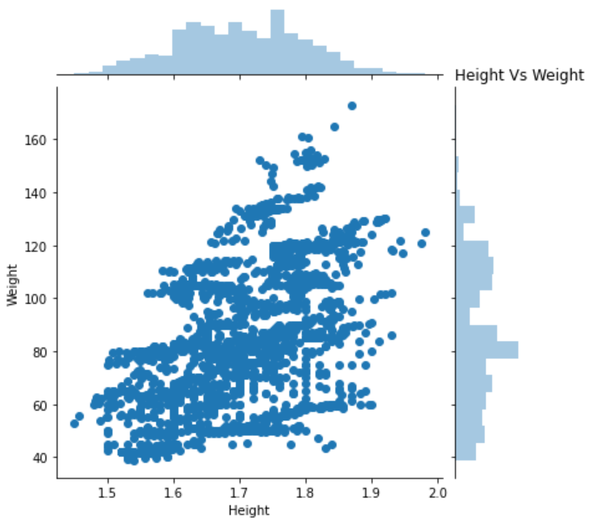
```
plt.scatter(x='Weight',y='NObeyesdad',data=df)
plt.ylabel('Nobeyesdad')
plt.xlabel('Weight')
plt.title("Weight Vs NObeyesdad")
```



Inference:

The distribution of weight is uneven over the categories of obesity which is obvious because even small changes in the weight will affect the obesity. It also shows the range of weight that lie in each category

```
sns.jointplot(x='Height',y='Weight',data=df,kind='scatter')
plt.title("Height Vs Weight",loc='left')
```

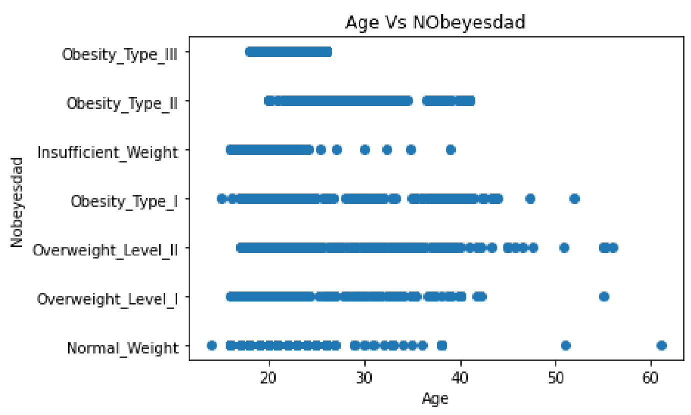


I

nference:

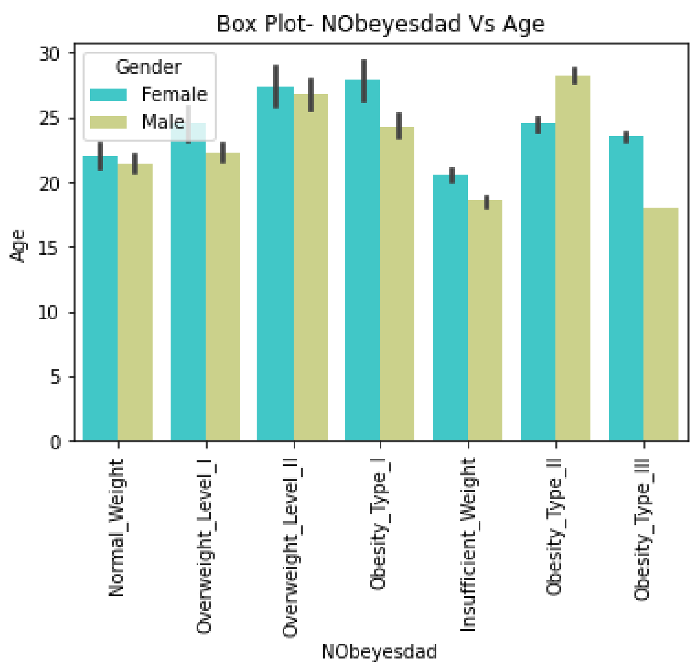
This graph shows the scatter of height vs weight. There is a dense scatter where the height is between 1.5 to 1.7 and weight is between 60 to 80

```
plt.scatter(x='Age',y='NObeyesdad',data=df)
plt.xlabel('Age')
plt.ylabel('Nobeyesdad')
plt.title("Age Vs NObeyesdad")
```



Inference:
This shows the distribution of age over the obesity type. From this we can also see the age group of the people lying in each category of obesity. We can also see some outliers for age of people who fall under Normal weight category

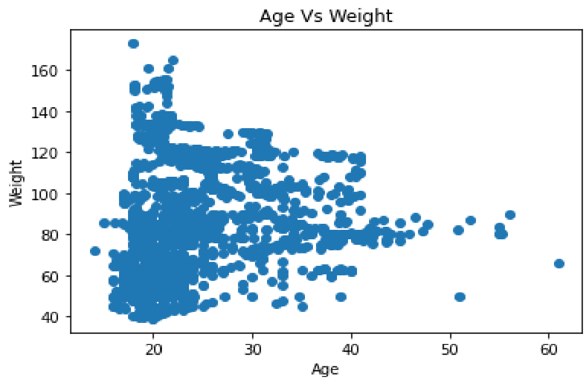
```
sns.barplot(df['NObeyesdad'],df['Age'],hue=df['Gender'],palett
e='rainbow')
plt.xticks(rotation='vertical')
plt.title(" NObeyesdad Vs Age")
```



Inference:
From this plot we can see that the average age of people under Overweight_Level_II and Obsity_Type_I is greater than people in other

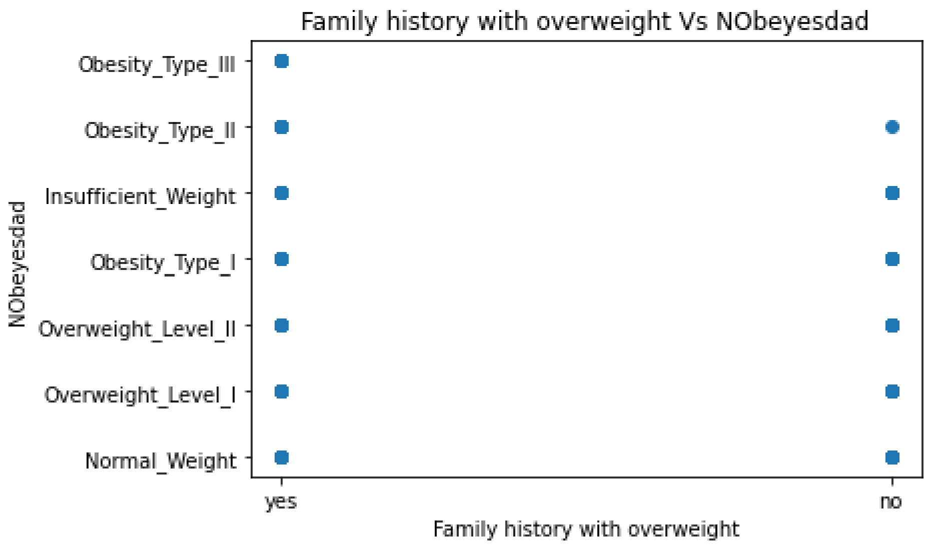
categories of obesity. Hence we can say that Overweight_Level_II and Obesity_Type_I is not more likely to be caused in people with age below 20

```
plt.scatter(x='Age',y='Weight',data=df)
plt.xlabel('Age')
plt.ylabel('Weight')
plt.title('Age Vs Weight')
```



Inference:
From this plot we can see that the density is maximum where age is between 15 To 25 and weight is 40 to 80

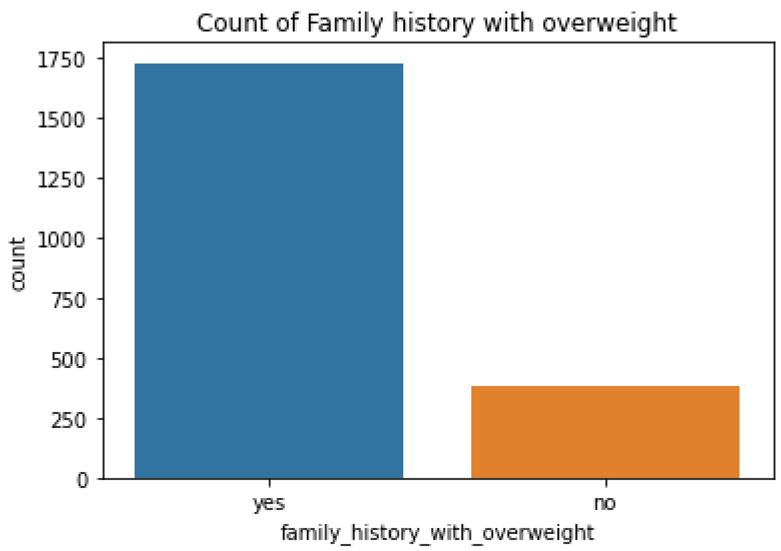
```
plt.scatter(x='family_history_with_overweight',y='NObeyesdad',
data=df)
plt.xlabel('Family history with overweight')
plt.ylabel('NObeyesdad')
plt.title('Family history with overweight Vs NObeyesdad')
```



Inference:
This shows the distribution of family history with overweight over the obesity column. Here we can see that there are no people with Obesity_Type_III and no

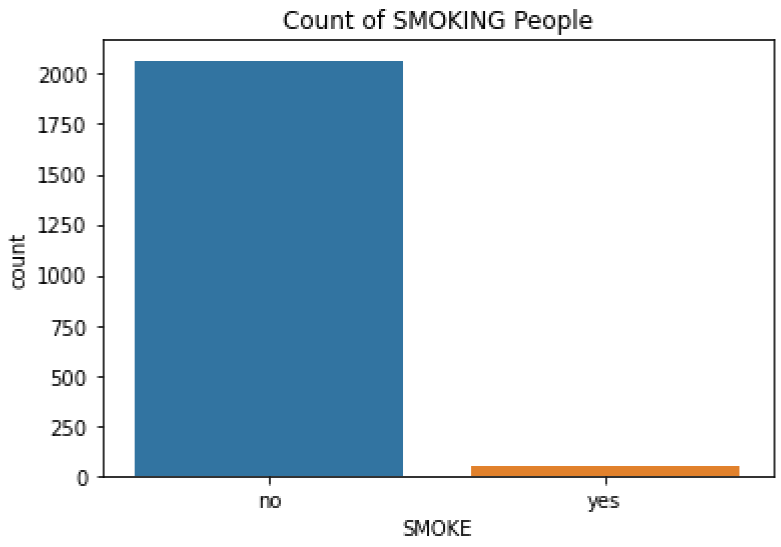
family history with overweight. Hence it can be said that Obesity_type_III is more of a genetic disorder

```
sns.countplot(df['family_history_with_overweight'])
plt.title('Count of Family history with overweight')
```



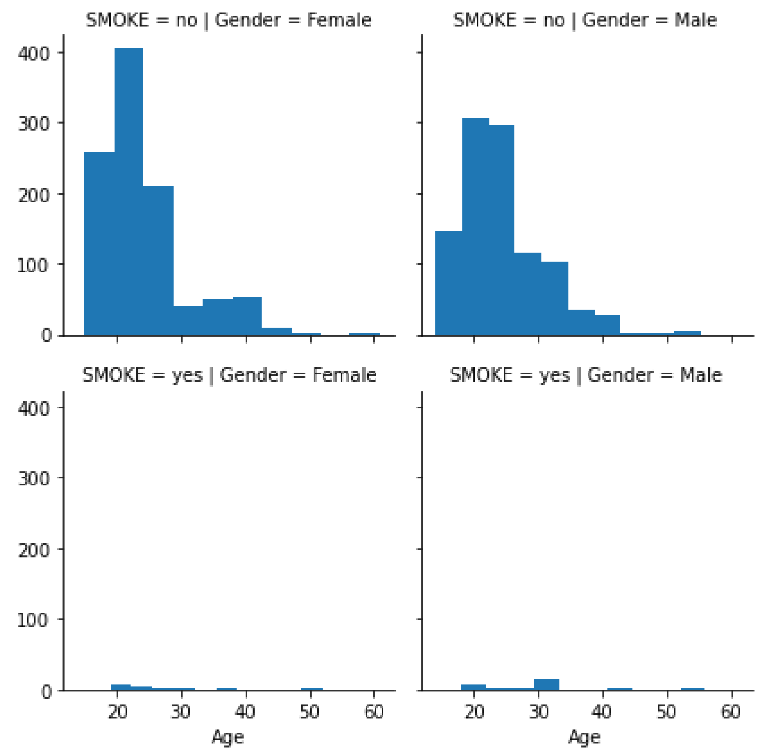
From this graph we can see that there is a bias in sampling of the data in terms of family history because we can see only a few samples of data for "no" category and we can see more number of data for yes category

```
sns.countplot(df['SMOKE'])
plt.title("Count of SMOKING People")
```



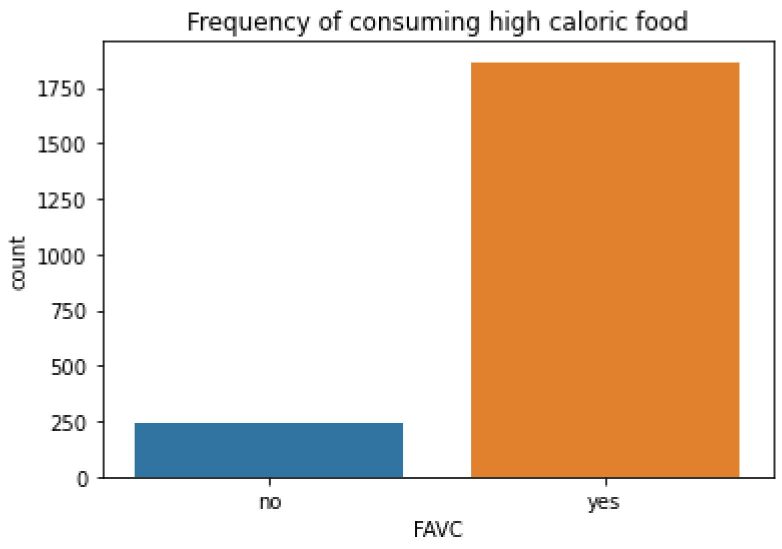
From this graph also, we can see a bias in the data collection since we don't have enough data where SMOKE = "yes".

```
fg = sns.FacetGrid(df, col="Gender",  row="SMOKE")
fg = fg.map(plt.hist,"Age")
```



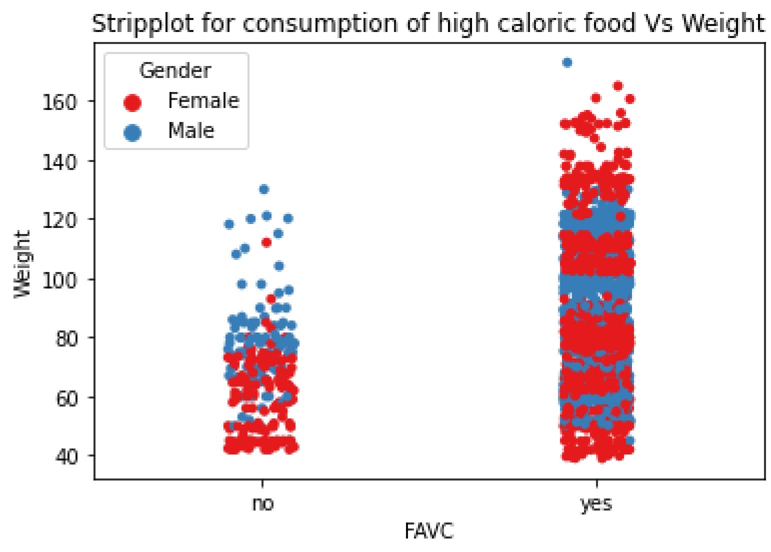
Inference:
From this graph we can see that of people who dont smoke, female are more and of people who smoke, men are more

```
sns.countplot(df['FAVC'])
plt.title('Frequency of consuming high caloric food')
```



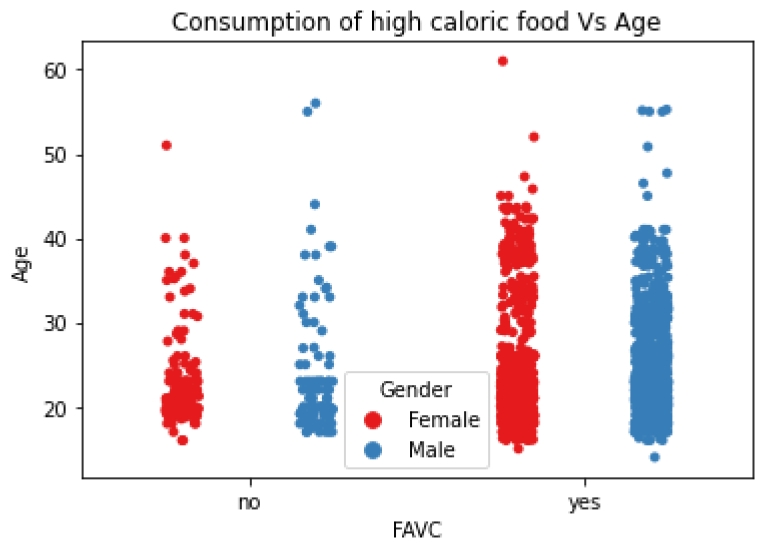
Inference:
The graph shows that people who consume high caloric food are more compared to those who do not consume.

```
sns.stripplot(x="FAVC", y="Weight",
data=df,jitter=True,hue='Gender',palette='Set1')
plt.title('Stripplot for consumption of high caloric food Vs
Weight')
```



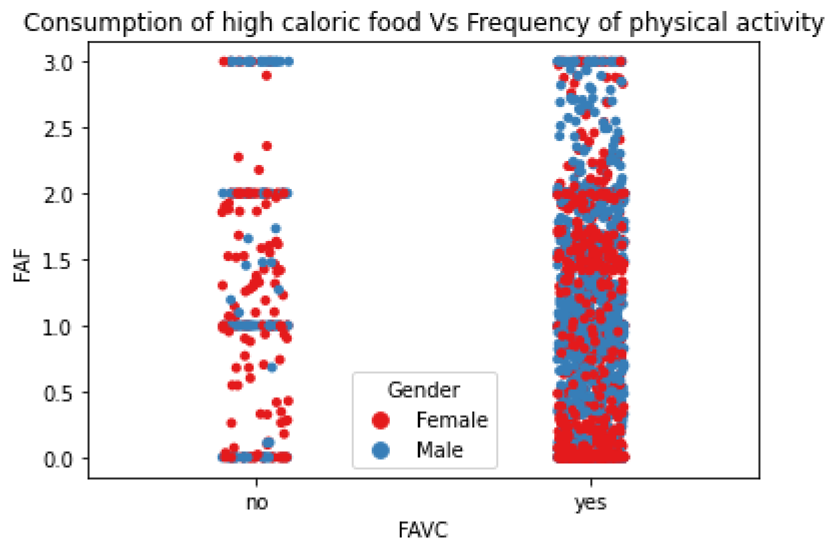
Inference:
This graph shows that the number of people who fall between the weights of 80 to 120 are more likely to have high caloric foods. And there are more males who gain weight without consuming high calorie foods than that of females

```
sns.stripplot(x="FAVC", y="Age",
data=df,jitter=True,hue='Gender',split=True,palette='Set1')
plt.title('Consumption of high caloric food Vs Age')
```



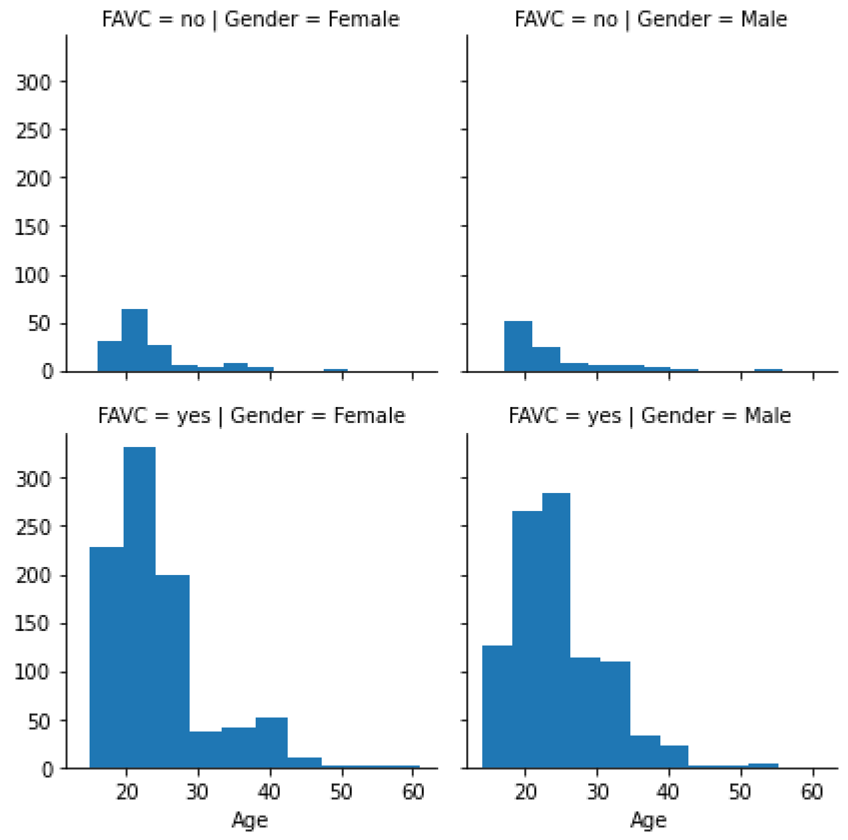
Inference:
This graph shows both male and female consume high caloric food during their age of 20-30

```
sns.stripplot(x="FAVC", y="FAF",
data=df,jitter=True,hue='Gender',palette='Set1')
plt.title('Consumption of high caloric food Vs Frequency of
physical activity')
```



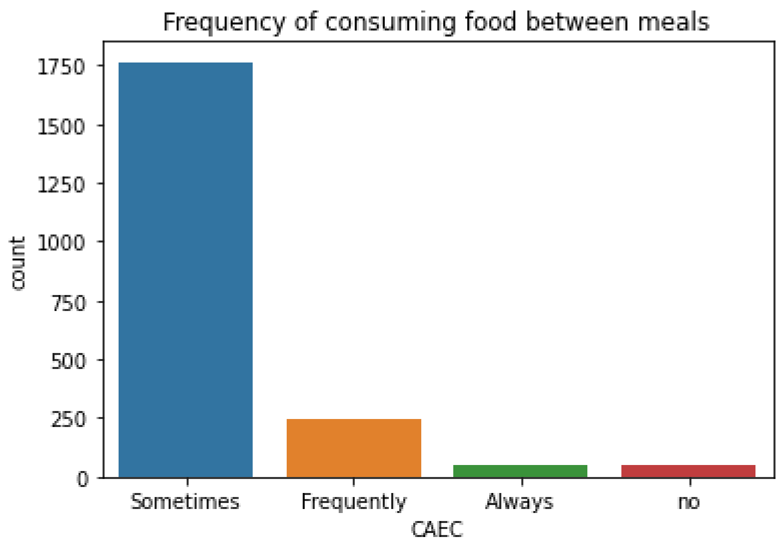
Inference:
This graph shows that the people who consume high caloric food perform physical activity for at least 1-2 hrs daily.

```
fg = sns.FacetGrid(df, col="Gender",  row="FAVC")
fg = fg.map(plt.hist,"Age")
```



Inference:
This graph shows that females who fall under the age of 20-30 tend to consume more high caloric food. Male who fall under the age of 20-25 tend to consume more high caloric food.

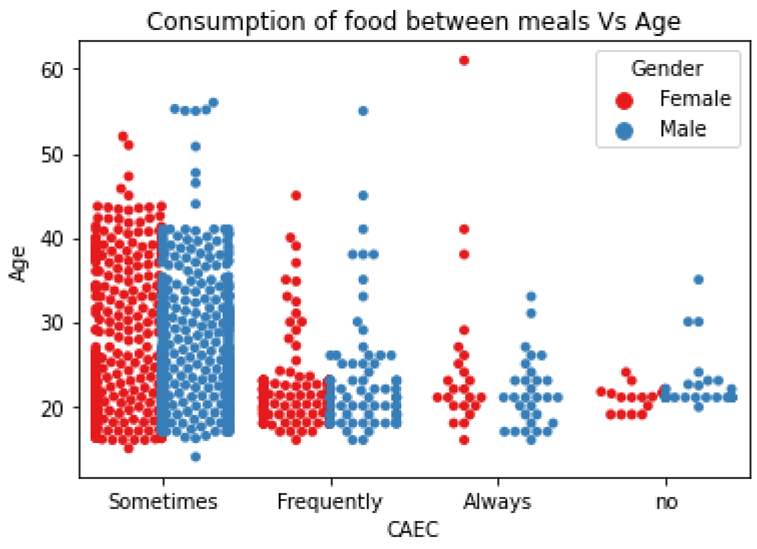
```
sns.countplot(df['CAEC'])
plt.title('Frequency of consuming food between meals')
```



Inference:

This graph shows that people who consume food between meals sometimes are more compared to other categories.

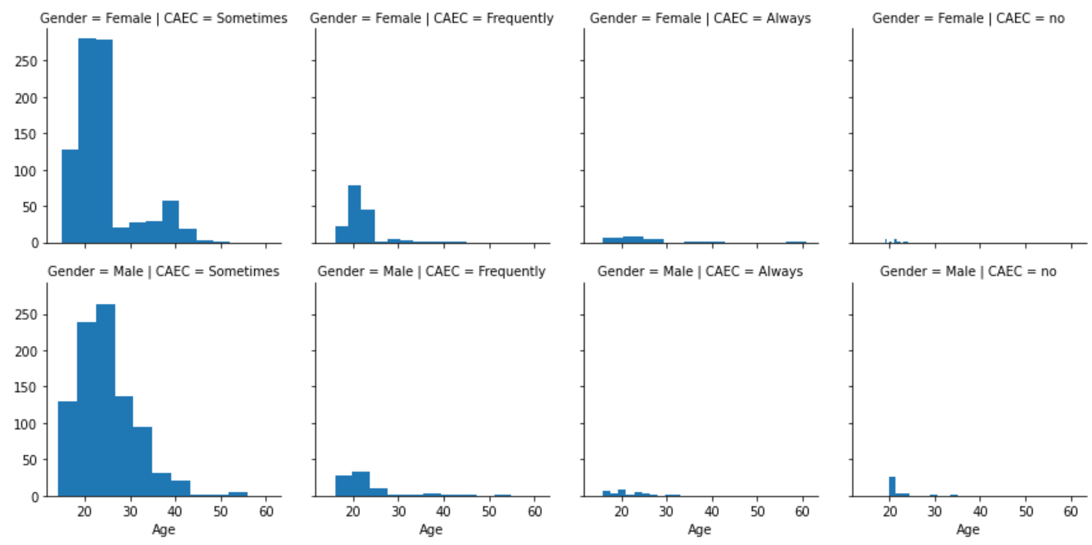
```
sns.swarmplot(x="CAEC", y="Age",
data=df,hue='Gender',split=True,palette='Set1')
plt.title('Consumption of food between meals Vs Age')
```



Inference:

The swarm plot shows that people who consume food between meals sometimes are most likely to be in the age of 20-40.

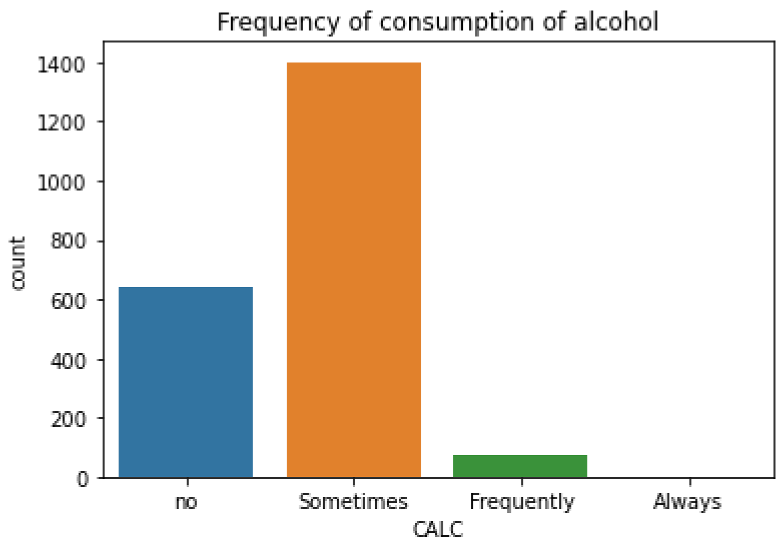
```
fg = sns.FacetGrid(df, col="CAEC",  row="Gender")
fg = fg.map(plt.hist,"Age")
```



Inference:

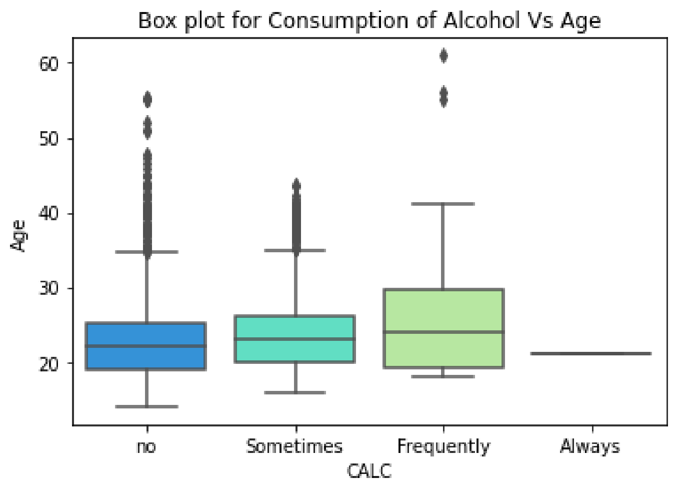
This graph shows that females who fall under the age of 20-25 are most likely to consume food between meals sometimes.Male who fall under the age of 20-30 are most likely to consume food between meals sometimes.

```
sns.countplot(df['CALC'])
plt.title('Frequency of consumption of alcohol')
```



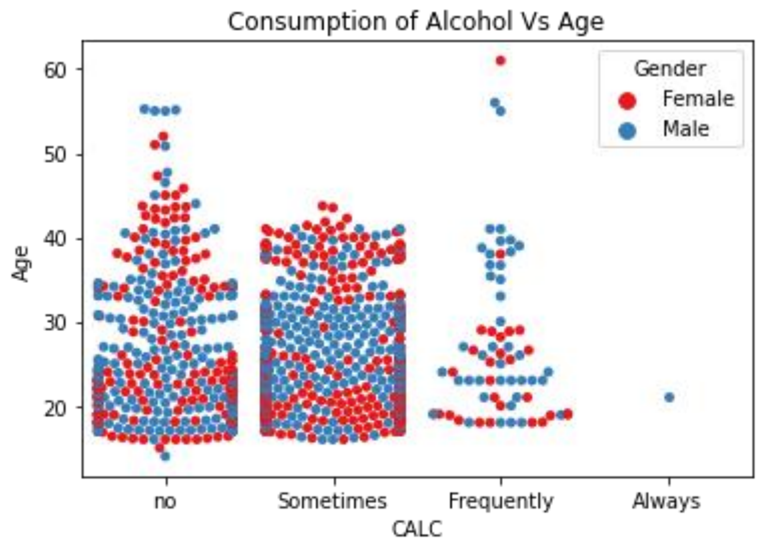
Inference:
This graph shows that consumption of alcohol sometimes is more while compared to other categories.

```
sns.boxplot(x="CALC", y="Age", data=df,palette='rainbow')
plt.title('Box plot for Consumption of Alcohol Vs Age')
```



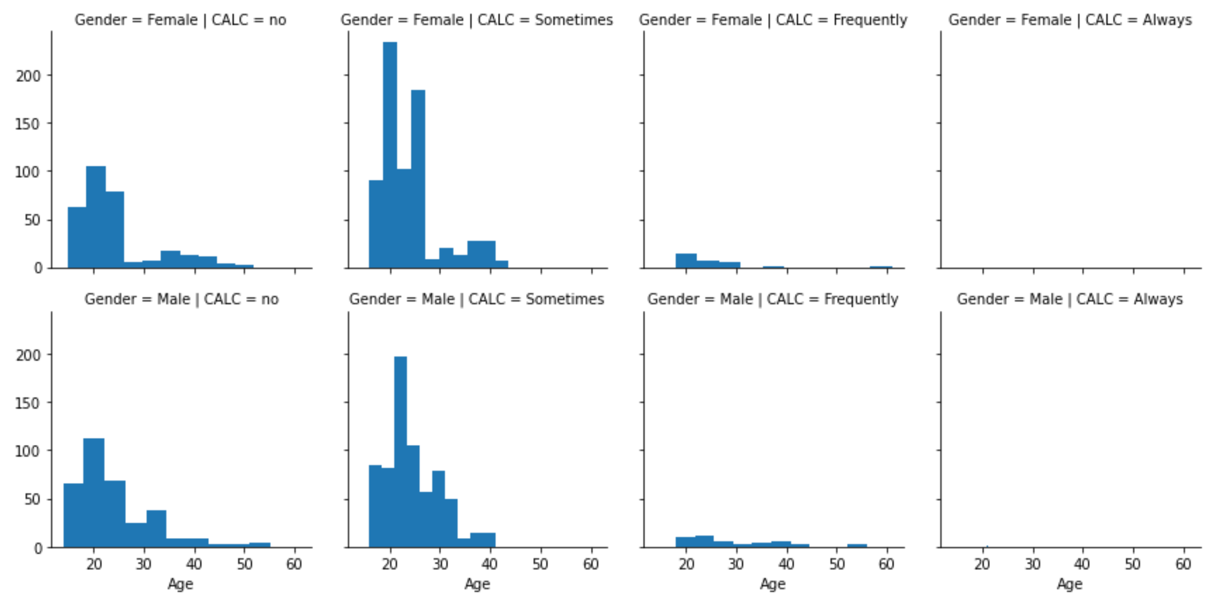
Inference:
This box plot shows that people who consume alcohol frequently fall under the age of 20-30.

```
sns.swarmplot(x="CALC", y="Age",
data=df,hue='Gender',palette='Set1')
plt.title('Consumption of Alcohol Vs Age')
```



Inference:
This graph shows that there are a few male in the age of 20 who consume alcohol always and few people who are more than 50 years old consume alcohol frequently.

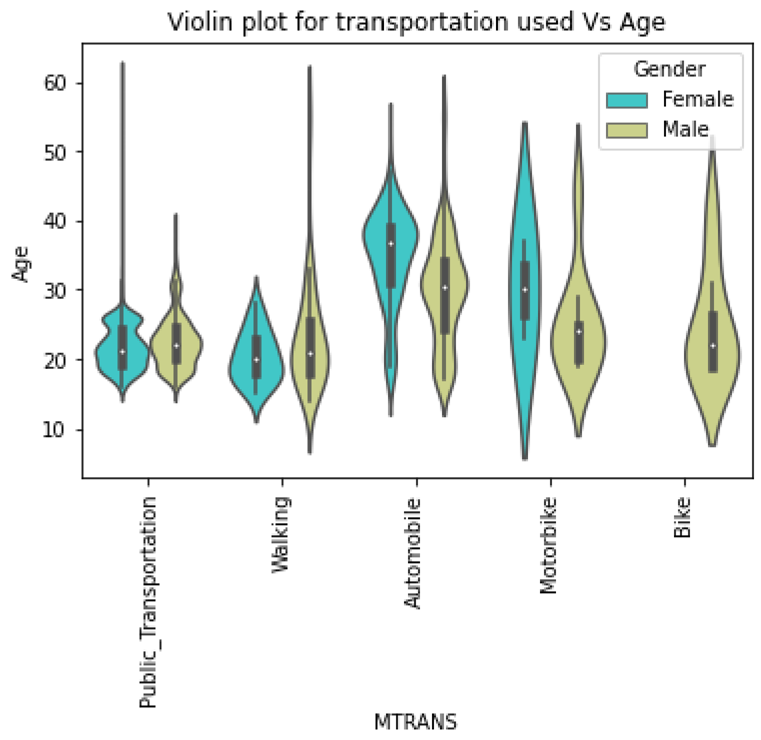
```
fg = sns.FacetGrid(df, col="CALC",  row="Gender")
fg = fg.map(plt.hist,"Age")
```



Inference:

This graph shows that females in the age of 20 consume alcohol sometimes.And males in the age of 25 consume alcohol sometimes.

```
sns.violinplot(x="MTRANS",
y="Age",hue='Gender',data=df,palette='rainbow')
plt.xticks(rotation='vertical')
plt.title('Violin plot for transportation used Vs Age')
```



Inference:

This violin plot shows the distribution of Male and Female using different kinds of modes of transportation. Here we can see that there are no females who ride bikes. And we can see that people across all age groups choose walking as their mode of transportation.

Label Encoding and One Hot encoding:

In this section we encode the categories of the output variables from 0 to 6 and we also obtain the One hot encoded dummy variables for all the categorical input variables.

```
from sklearn.preprocessing import LabelEncoder
le =LabelEncoder()
df_new["NObeyesdad"]=le.fit_transform(df_new["NObeyesdad"])
print("Encoding of output columnn")
print(pd.DataFrame(pd.Series(le.inverse_transform([0,1,2,3,4,5
,6])),columns=["Class Names indexed with class labels"]))
for cat in catagorical:

d=pd.get_dummies(df_new[cat],prefix=str(cat),drop_first=True)
    df_new.drop(cat,axis="columns",inplace=True)
    df_new=pd.concat([df_new,d],axis="columns")
```

Output:

```
Encoding of output columnn
  Class Names indexed with class labels
0                      Insufficient_Weight
1                            Normal_Weight
2                           Obesity_Type_I
3                          Obesity_Type_II
4                         Obesity_Type_III
5                       Overweight_Level_I
6                      Overweight_Level_II
```

Inference:

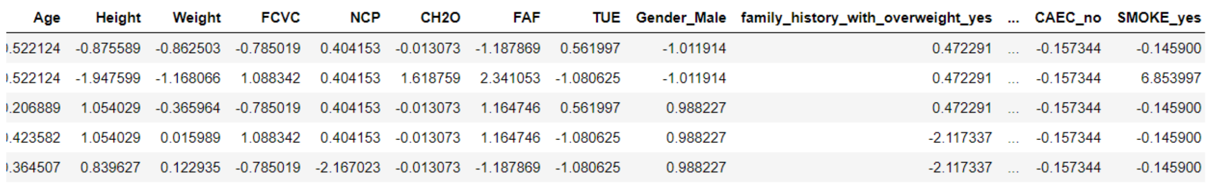
The above output tells us the index to which each category of the output column is encoded. For example “Insufficient_Weight” is encoded to 0 , “Normal_Weight” is encoded to 1 and so on

Scaling and X-y split of the data:

Here the data is scaled using Standard Scalar where the features are standardised by removing the mean and scaling to unit variance.Standardization of a dataset is a common requirement for many machine learning estimators . They might behave badly if the individual features do not more or less look like standard normally distributed data

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
df_scaled=pd.DataFrame(sc.fit_transform(df_new.drop("NObeyesda
d",axis=1)),columns=df_new.drop("NObeyesdad",axis=1).columns)
df_scaled.head()
X = df_scaled
y = df_new["NObeyesdad"]
X.head()
y.head()
```

Output:



Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	Gender_Male	family_history_with_overweight_yes	...	CAEC_no	SMOKE_yes
.522124	-0.875589	-0.862503	-0.785019	0.404153	-0.013073	-1.187869	0.561997	-1.011914	0.472291	...	-0.157344	-0.145900
.522124	-1.947599	-1.168066	1.088342	0.404153	1.618759	2.341053	-1.080625	-1.011914	0.472291	...	-0.157344	6.853997
.206889	1.054029	-0.365964	-0.785019	0.404153	-0.013073	1.164746	0.561997	0.988227	0.472291	...	-0.157344	-0.145900
.423582	1.054029	0.015989	1.088342	0.404153	-0.013073	1.164746	-1.080625	0.988227	-2.117337	...	-0.157344	-0.145900
.364507	0.839627	0.122935	-0.785019	-2.167023	-0.013073	-1.187869	-1.080625	0.988227	-2.117337	...	-0.157344	-0.145900

```
0     1
1     1
2     1
3     5
4     6
Name: NObeyesdad, dtype: int32
```

Inference:

A sample of the X and y data after performing scaling and one hot encoding can be seen above.

PCA Test:

Kaiser-Meyer-Olkin (KMO) Test of Sphericity:

Kaiser-Meyer-Olkin (KMO) Test is a measure of how suited your data is for Factor Analysis. The test measures sampling adequacy for each variable in the model and for the complete model. The statistic is a measure of the proportion of variance among variables that might be common variance.

- 0.00 to 0.49 unacceptable.
- 0.50 to 0.59 miserable.
- 0.60 to 0.69 mediocre.
- 0.70 to 0.79 middling.
- 0.80 to 0.89 meritorious.
- 0.90 to 1.00 marvelous.

```
# Adequacy Test
# Kaiser-Meyer-Olkin (KMO) Test
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all,kmo_model=calculate_kmo(df_scaled)
print(kmo_model)
```

Output:

```
0.45688235721397424
```

Inference:

Since our output is only 0.45, it is not a suggested to perform PCA on our data

Feature Selection using Recursive Elimination:

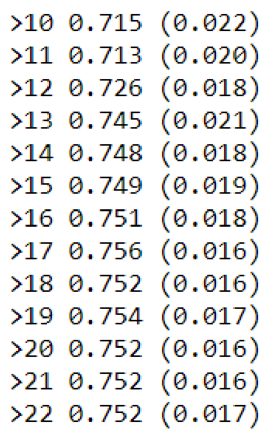
Since we can not perform PCA, we can try to remove some variables that are least important using Recursive feature Elimination.

Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. Features are ranked by the model and by recursively eliminating a small number of features per loop, RFE attempts to eliminate dependencies and collinearity that may exist in the model.Feature ranking with recursive feature elimination and cross-validated selection is performed to obtain the best number of features

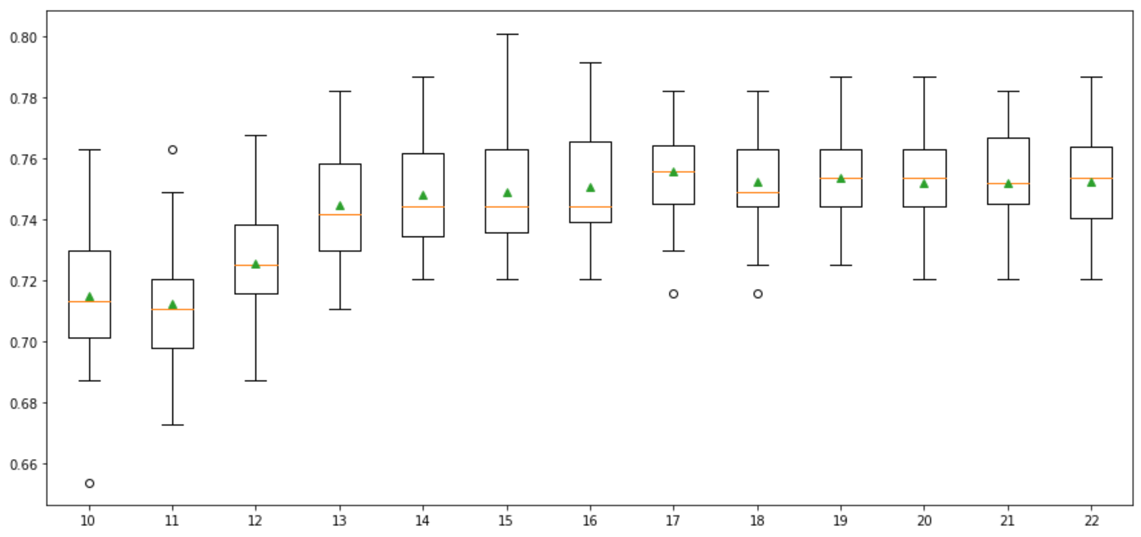
```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.pipeline import Pipeline
from matplotlib import pyplot
# get a list of models to evaluate
def get_models():
    models = dict()
    for i in range(10, 23):
        rfe =
RFE(estimator=LogisticRegression(solver='liblinear',multi_clas
s="auto"), n_features_to_select=i)
        model =
LogisticRegression(solver='liblinear',multi_class="auto")
        models[str(i)] =
Pipeline(steps=[('s',rfe),('m',model)])
    return models
# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy',
cv=cv, n_jobs=-1, error_score='raise')
    return scores
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores),
std(scores)))
```

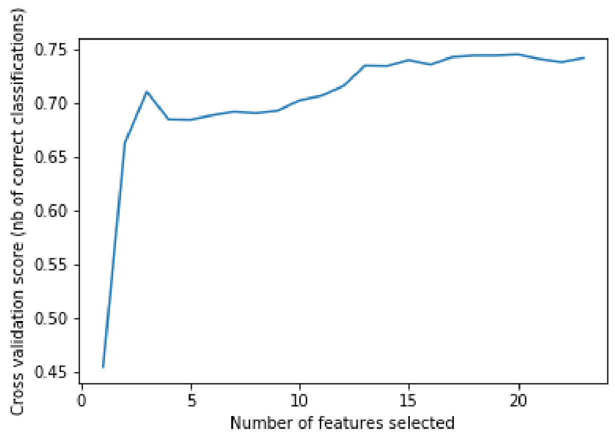
```
# plot model performance for comparison
plt.figure(figsize=(15,7))
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
model =
LogisticRegression(solver='liblinear',multi_class="auto")
# The "accuracy" scoring is proportional to the number of
correct
# classifications
rfecv = RFECV(estimator=model, step=1, cv=StratifiedKFold(10),
              scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct
classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1),
rfecv.grid_scores_)
plt.show()
print("Num Features: %d" % rfecv.n_features_)
print("Selected Features:\n %s " %
X.loc[:,rfecv.support_].columns)
print("Feature Ranking:\n %s" % rfecv.ranking_)
X_opt=X.loc[:,rfecv.support_]
```

Output:



```
>10 0.715 (0.022)
>11 0.713 (0.020)
>12 0.726 (0.018)
>13 0.745 (0.021)
>14 0.748 (0.018)
>15 0.749 (0.019)
>16 0.751 (0.018)
>17 0.756 (0.016)
>18 0.752 (0.016)
>19 0.754 (0.017)
>20 0.752 (0.016)
>21 0.752 (0.016)
>22 0.752 (0.017)
```





```
Num Features: 20
Selected Features:
 Index(['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'Gender_Male',
       'family_history_with_overweight_yes', 'FAVC_yes', 'CAEC_Frequently',
       'CAEC_Sometimes', 'CAEC_no', 'SCC_yes', 'CALC_Frequently',
       'CALC_Sometimes', 'CALC_no', 'MTRANS_Bike',
       'MTRANS_Public_Transportation', 'MTRANS_Walking'],
      dtype='object')
Feature Ranking:
 [1 1 1 1 1 1 1 3 1 1 1 1 1 1 2 1 1 1 1 1 4 1 1]
```

Inference:

- The first output shows the accuracy when we have number of features from 10 to 22. When the number of features is 20 , there is no increase in the accuracy even after having 21 or 22 variables. Hence it can be said that 20 is the optimum number of features.
- The second output explains the same in a graphical manner. We can confirm that 20 is the optimum number of variables.
- The third graph explains the spike in the accuracy as we keep adding features. As we can see, the graph does not tend to spike much after the feature size 20. Hence we can be sure that the optimal number of features is 20.
- The fourth output explains what are those 20 features that the algorithm has considered and it also explains the feature ranking of those features.

Random Train Test Split:

Here we perform train test split at random for our data set inorder to find the best model and its best parameters. Once the best model and its best parameters are found, we find the best train test split for that model using Bias-Variance Tradeoff

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_opt, y,
test_size=0.20)
```

Finding the best model with best parameters using GridSearchCV:

Here we are finding the best model that could fit our multiclass - classification dataset along with its best parameters using Grid Search CV. We try to fit the following models with its parameters and finally a best model is chosen along with its best parameters

1. Support Vector Classifier
2. Random Forest Classifier
3. Logistic regression
4. Decision Tree Classifier
5. K Nearest Neighbours Classifier

Support Vector Classifier:

For support vector classification, the model is trained on both linear and Radial bias kernels. The C values are also given as 10,20 and 30. Finally the best fit for SVC is obtained by tuning the above parameters.

C : Regularization parameter	1,10,20
Kernel : Specifies the kernel type to be used in the algorithm	rbf,linear

Random Forest Classifier:

The Random forest classifier is trained on the following possible combination of parameters

n_estimators : number of trees to be considered	5,10,20,50
Criterion: error criterion	gini,entropy
Min_samples_split: number of mini sample splits	2,3,4

Logistic regression:

Logistic regression model is trained for the following c values and the optimum model is obtained

C : Regularization parameter.	1,5,10,20

Decision Tree Classifier:

Decision tree classifier is trained for the following criterion options and the best one that fits the model is obtained

Criterion: error criterion	gini,entropy

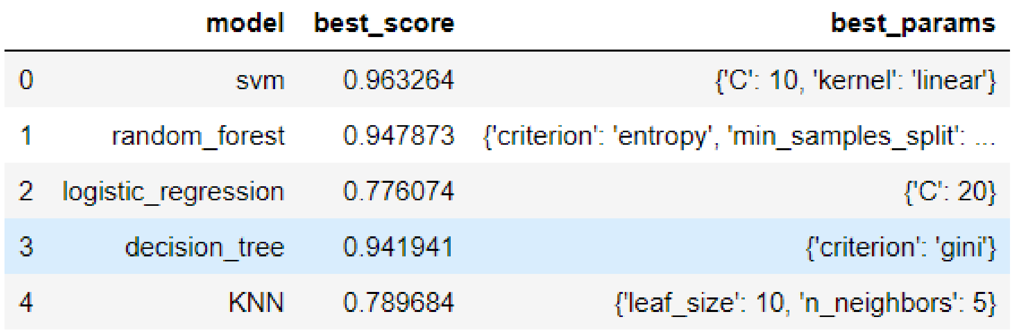
K Nearest Neighbours Classifier:

KNN model is trained for the following parameters and the best one is obtained after tuning

N_neighbors: Number of neighbors to use	5 to 20
Leaf_size: Leaf size passed to BallTree or KDTree	10,20,30,40,50,60

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [5,10,20,50],
            "criterion":['gini','entropy'],
            'min_samples_split':[2,3,4],

        }
    },
    'logistic_regression' : {
        'model':
LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10,20]
        }
    },

    'decision_tree': {
        'model': DecisionTreeClassifier(),
        'params': {
            'criterion': ['gini','entropy'],

        }
    },

    'KNN': {
        'model': KNeighborsClassifier(),
        'params': {
            'n_neighbors': [x for x in range(5,20)],
            'leaf_size':[10,20,30,40,50,60],
        }
    }
}
from sklearn.model_selection import GridSearchCV
import pandas as pd
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=10,
return_train_score=False,verbose=10,n_jobs=1)
    clf.fit(X_train, y_train)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

model_results =
pd.DataFrame(scores,columns=['model','best_score','best_params
'])
```

Output:



	model	best_score	best_params
0	svm	0.963264	{'C': 10, 'kernel': 'linear'}
1	random_forest	0.947873	{'criterion': 'entropy', 'min_samples_split': ...
2	logistic_regression	0.776074	{'C': 20}
3	decision_tree	0.941941	{'criterion': 'gini'}
4	KNN	0.789684	{'leaf_size': 10, 'n_neighbors': 5}

Inference:
From the above output we can see that svm is the one that is best for our dataset compared to other models. Hence we continue with the Support Vector Classifier. We have also obtained our best parameters as c=10 and kernel=linear.

Finding the best train-test split using Bias Variance Tradeoff:

After finding the best model to our data, we find the best split of training and test data by using Bias variance trade off.

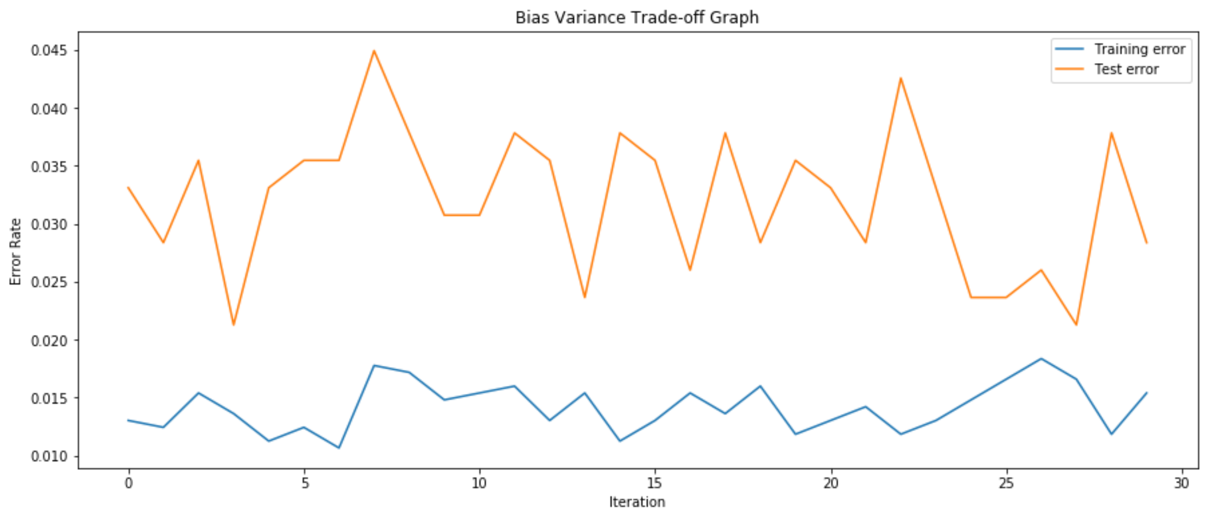
Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us the spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but have high error rates on test data.

```
from sklearn.model_selection import train_test_split
train_score=[]
test_score=[]
splits = []
svc_bvt=svm.SVC(C=20,kernel="linear",gamma="auto")
for i in range(0,30):
    X_train_bvt, X_test_bvt, y_train_bvt, y_test_bvt = 
train_test_split(X_opt, y, test_size=0.20,shuffle=True)
    svc_bvt.fit(X_train_bvt,y_train_bvt)
    splits.append({
        "X_train":X_train_bvt,
        "X_test":X_test_bvt,
        "y_train":y_train_bvt,
        "y_test":y_test_bvt})
    train_score.append(1 - 
svc_bvt.score(X_train_bvt,y_train_bvt))
```

```
    test_score.append(1 -
svc_bvt.score(X_test_bvt,y_test_bvt))
plt.figure(figsize=(15,6))
plt.plot(range(0,30),train_score,label="Training error")
plt.plot(range(0,30),test_score,label="Test error")
plt.xlabel("Iteration")
plt.ylabel("Error Rate")
plt.legend(loc="best")
plt.title("Bias Variance Trade-off Graph")
diff=abs(np.array(train_score)-np.array(test_score))
best_split_index = np.argmin(diff)
best_split=splits[best_split_index]
X_train = best_split["X_train"]
X_test = best_split["X_test"]
y_train = best_split["y_train"]
y_test = best_split["y_test"]
```

Output:



Inference:

From the above graph we can see that the difference between the bias and variance is minimum for the 27th split. Hence that split is considered as the best split and we train the SVC model using that split.

Fitting the best model with best parameters and best data split:

```
svc = svm.SVC(C=20,kernel="linear",gamma="auto")
svc.fit(X_train,y_train)
y_pred = svc.predict(X_test)
y_score = svc.decision_function(X_test)
```

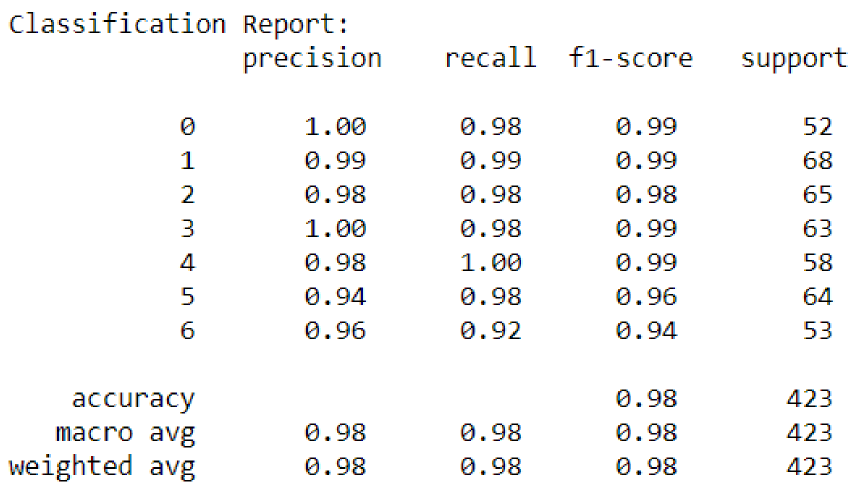
Output:

```
SVC(C=20, gamma='auto', kernel='linear')
```

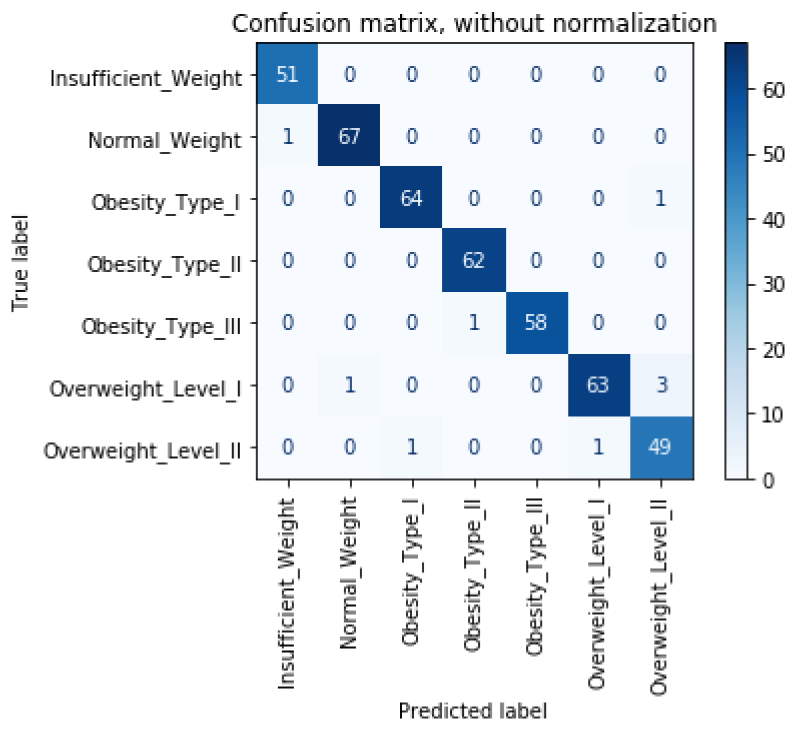
Model Evaluation:

```
from sklearn.metrics import
confusion_matrix,classification_report
print("Classification Report:\n" ,
classification_report(y_pred,y_test))
# Plot non-normalized confusion matrix
class_names = le.inverse_transform([0,1,2,3,4,5,6])
from sklearn.metrics import plot_confusion_matrix
titles_options = [("Confusion matrix, without normalization",
None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(svc, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)
    plt.xticks(rotation="vertical")

    print(title)
    print(pd.DataFrame(disp.confusion_matrix))
plt.show()
```

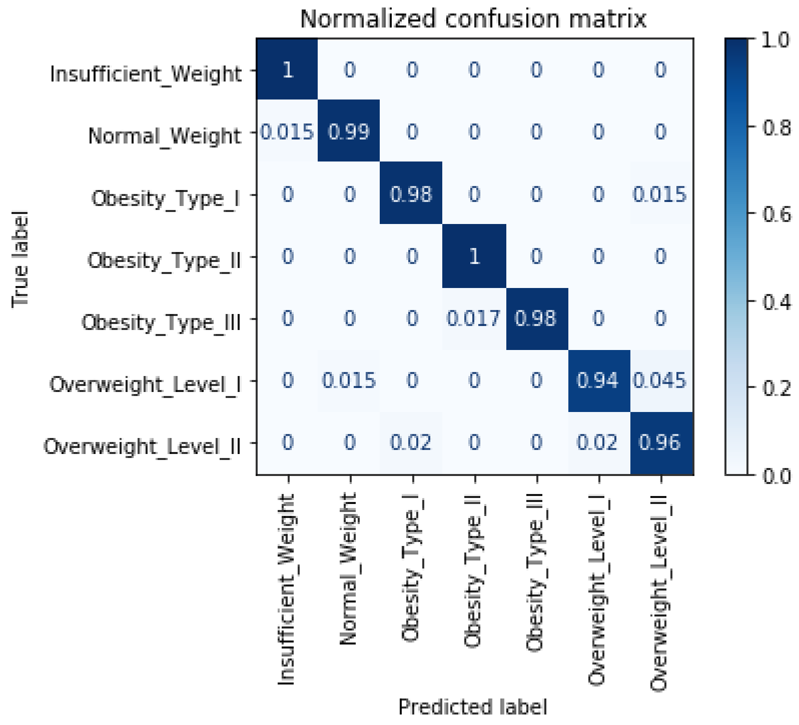
Output:



```
 Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.98      0.99        52
           1       0.99      0.99      0.99        68
           2       0.98      0.98      0.98        65
           3       1.00      0.98      0.99        63
           4       0.98      1.00      0.99        58
           5       0.94      0.98      0.96        64
           6       0.96      0.92      0.94        53

    accuracy                           0.98       423
   macro avg       0.98      0.98      0.98       423
weighted avg       0.98      0.98      0.98       423
```



True label \ Predicted label	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_II	Obesity_Type_III	Overweight_Level_I	Overweight_Level_II
Insufficient_Weight	51	0	0	0	0	0	0
Normal_Weight	1	67	0	0	0	0	0
Obesity_Type_I	0	0	64	0	0	0	1
Obesity_Type_II	0	0	0	62	0	0	0
Obesity_Type_III	0	0	0	1	58	0	0
Overweight_Level_I	0	1	0	0	0	63	3
Overweight_Level_II	0	0	1	0	0	1	49



True label \ Predicted label	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_II	Obesity_Type_III	Overweight_Level_I	Overweight_Level_II
Insufficient_Weight	1	0	0	0	0	0	0
Normal_Weight	0.015	0.99	0	0	0	0	0
Obesity_Type_I	0	0	0.98	0	0	0	0.015
Obesity_Type_II	0	0	0	1	0	0	0
Obesity_Type_III	0	0	0	0.017	0.98	0	0
Overweight_Level_I	0	0.015	0	0	0	0.94	0.045
Overweight_Level_II	0	0	0.02	0	0	0.02	0.96

Inference:

The first output shows the classification report of our model. Precision is the ratio of correctly predicted observations to the total predicted observations. The average precision of our model is 98% Recall is the ratio of correctly predicted observations to the all observations in actual class. The average recall of our dataset is also 98%. F1 score is the harmonic mean of precision and recall. F1 score of our model is also 98% which is a great fit

The second output shows the confusion matrix of our model. As we can see we have only 7 mis-classified data in our model.

The third output is a normalized version of the confusion matrix.

Feature Importance evaluation:

Feature importance evaluation tells us how much each feature is contributing towards predicting the output class.

```
from sklearn.inspection import permutation_importance
r = permutation_importance(svc, X_train,
y_train,n_repeats=30,random_state=0)
print("FEATURE IMPORTANCES:")
for i in r.importances_mean.argsort()[::-1]:
    if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
            print(f"{X_train.columns[i]:<35}:"
                 f"{r.importances_mean[i]*100:.2f}"
                    f" +/- {r.importances_std[i]:.3f}")
```

Output:

```
FEATURE IMPORTANCES:
Weight                             :77.78 +/- 0.007
Height                             :38.11 +/- 0.007
Gender_Male                        :13.94 +/- 0.006
Age                                :3.37 +/- 0.004
NCP                                :3.00 +/- 0.004
CAEC_Sometimes                     :2.41 +/- 0.002
MTRANS_Public_Transportation       :1.99 +/- 0.003
CAEC_Frequently                    :1.44 +/- 0.002
family_history_with_overweight_yes :1.29 +/- 0.003
SCC_yes                            :0.95 +/- 0.002
FCVC                               :0.66 +/- 0.002
FAF                                :0.63 +/- 0.002
CAEC_no                            :0.63 +/- 0.002
FAVC_yes                           :0.56 +/- 0.001
CALC_Frequently                    :0.36 +/- 0.001
CALC_no                            :0.27 +/- 0.001
MTRANS_Walking                     :0.19 +/- 0.001
CALC_Sometimes                     :0.16 +/- 0.001
```

Inference:

From the above output we can say that weight is the one that contributes the most towards predicting the output which is followed by height.

10 Fold Cross Validation:

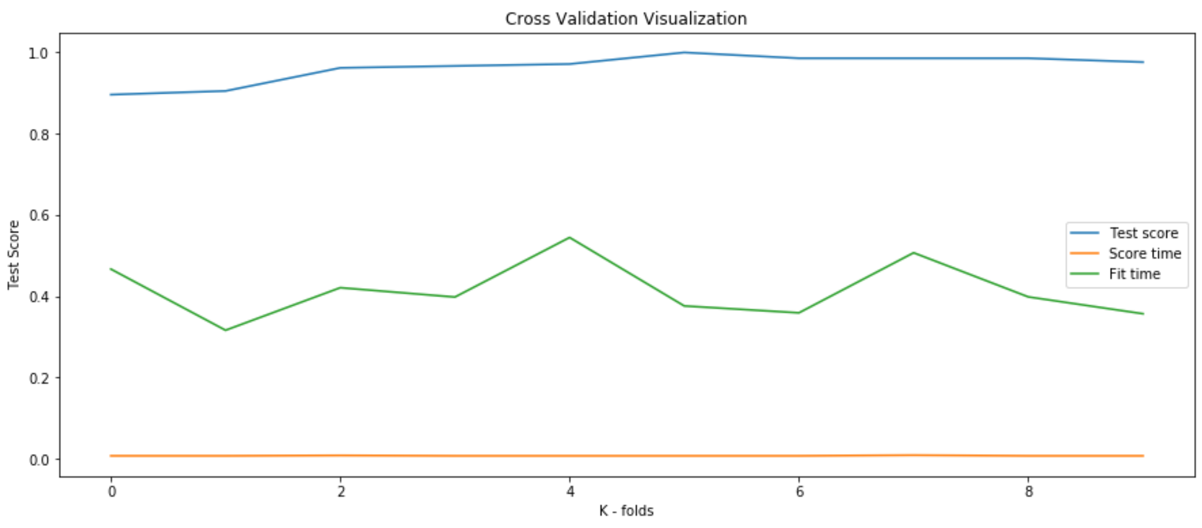
Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

```
from sklearn.model_selection import cross_validate
cv_results = cross_validate(svc, X_opt, y, cv=10)
print(pd.DataFrame(cv_results))
print("The average Cross validation score is
{}".format(np.mean(cv_results["test_score"])))
plt.figure(figsize=(15,6))
plt.plot(range(0,10),cv_results["test_score"],label="Test
score")
plt.plot(range(0,10),cv_results["score_time"],label="Score
time")
plt.plot(range(0,10),cv_results["fit_time"],label="Fit time")
plt.xlabel("K - folds")
plt.ylabel("Test Score")
plt.title("Cross Validation Visualization")
plt.legend(loc="best")
```

Output:

```
   fit_time  score_time  test_score
0  0.466751    0.007026    0.896226
1  0.316154    0.006982    0.905213
2  0.420874    0.008018    0.962085
3  0.397895    0.006982    0.966825
4  0.544543    0.006982    0.971564
5  0.375994    0.006981    1.000000
6  0.359081    0.006986    0.985782
7  0.506931    0.008653    0.985782
8  0.398286    0.006980    0.985782
9  0.356817    0.006991    0.976303
```



Inference:

From the above output we can see that our model performs well on all the 10 folds of cross validation. Hence we can be sure that our model is fit to be deployed.
The graph also explains that there is no major variation in the accuracy of the model over different folds of data. Hence with this we can also confirm that our model is fit to be deployed and used

Conclusion:

Hence we have now successfully built a model that can be used to estimate the type of Obesity Levels Based On Eating Habits and Physical condition. This can be used by people to know in which category of obesity they lie based on answering a few survey questions.