Data Science Project Report

Obesity Dataset

Munaima Muzamil

023-19-0096

BSCS-VI B

Submitted to

Dr.Ghulam Murtaza

Data Set Information:

This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labeled with the class variable NObesity (Obesity Level), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III. 77% of the data was generated synthetically using the Weka tool and the SMOTE filter, 23% of the data was collected directly from users through a web platform.

Many of these attributes have acronyms, so I briefly described all of them below:

- **Gender**: 1= female, 2 = male
- **Age**: numeric
- **Height**: numeric, in meters
- **Weight**: numeric, in kilograms
- **family_history** (family history of obesity): 1 = yes, 2 = no
- **FAVC** 1= yes, 2= no
- **FCVC** (frequency of consumption of vegetables: 1 = never, 2 = sometimes, 3 = always
- **NCP**(number of main meals): 1, 2, 3 or 4 meals a day
- **CAEC**(consumption of food between meals): 1=no, 2=sometimes, 3=frequently, 4=always
- **Smoke**: 1= yes, 2= no
- **CH2O**(consumption of water): 1 = less than a liter, 2 = 1–2 liters, 3 = more than 2 liters
- **SCC** (calorie consumption): 1= yes, 2 = no
- **FAF** (physical activity frequency per week): 0 = none, 1 = 1 to 2 days, 2= 2 to 4 days, 3 = 4 to 5 days
- **TUE** (time using technology devices a day): 0 = 0–2 hours, 1 = 3–5 hours, 2 = more than 5 hours
- **CALC** (consumption of alcohol): 1= never, 2 = sometimes, 3 = frequently, 4 = always

- **MTRANS**(Transportation): 1 = automobile, 2 = motorbike, 3 = bike, 4 = public transportation, 5= walking
- **NObesity (target variable)**: 2 = not obese, 4 = obese

Data Preparation

Import libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report,recall_score,precision_score,accuracy_score)
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from scipy.stats import loguniform
from sklearn.model_selection import GridSearchCV
from yellowbrick.features import FeatureImportances
from sklearn import metrics
from yellowbrick.classifier import ClassificationReport
import pandas as pd
import numpy as np
from numpy import mean, std
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import model_selection
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, make_scorer
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, RepeatedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import SMOTE
```

Importing Data and Checking shape of data

```
data=pd.read_csv("mm.csv")


data.shape
```

```
(2111, 17)
```

The read_csv function reads the dataset and the .shape function accurately returned (2111,17).

Have a look at the top 10 rows of dataset

```
data.head(10)
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation	
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation	
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking	Ove
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation	Over
5	Male	29.0	1.62	53.0	no	yes	2.0	3.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Automobile	
6	Female	23.0	1.50	55.0	yes	yes	3.0	3.0	Sometimes	no	2.0	no	1.0	0.0	Sometimes	Motorbike	
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_Transportation	
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_Transportation	
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_Transportation	

Information about dataset

```
from sklearn.utils import shuffle

cols = data.columns.values
data.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
```

Checking null values

```
[82] pd.isnull(data).sum()

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

This shows that there are no null values.

Converting categorical data into numeric data

```
[83] from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     Y=data['NObeyesdad']

     data['NObeyesdad'] = le.fit_transform(Y)


     data['Gender'] = le.fit_transform(data['Gender'])
     data['family_history_with_overweight'] = le.fit_transform(data['family_history_with_overweight'])
     data['CAEC'] = le.fit_transform(data['CAEC'])
     data['SMOKE'] = le.fit_transform(data['SMOKE'])
     data['SCC'] = le.fit_transform(data['SCC'])
     data['CALC'] = le.fit_transform(data['CALC'])
     data['MTRANS'] = le.fit_transform(data['MTRANS'])
     data['FAVC'] = le.fit_transform(data['FAVC'])


     X=data.drop(["NObeyesdad"],axis=1)
     Y=data['NObeyesdad']

     data.head(10)
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
0	0	21.0	1.62	64.0	1	0	2.0	3.0	2	0	2.0	0	0.0	1.0	3	3	1
1	0	21.0	1.52	56.0	1	0	3.0	3.0	2	1	3.0	1	3.0	0.0	2	3	1
2	1	23.0	1.80	77.0	1	0	2.0	3.0	2	0	2.0	0	2.0	1.0	1	3	1
3	1	27.0	1.80	87.0	0	0	3.0	3.0	2	0	2.0	0	2.0	0.0	1	4	5
4	1	22.0	1.78	89.8	0	0	2.0	1.0	2	0	2.0	0	0.0	0.0	2	3	6
5	1	29.0	1.62	53.0	0	1	2.0	3.0	2	0	2.0	0	0.0	0.0	2	0	1

Label Encoder converts the categorical data into numeric.

Splitting test and training data

The dataset is partitioned into training (70%) and testing (30%) sets, and the respective shapes are printed to make sure the data was split correctly before the models are built.

```
#Divide into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

These numbers indicate that the training set has 1477 data points, while the testing set has 634 data points.

Models

For modeling I have used KNN classifier and Logistic regression model

```
# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score
val = accuracy_score(y_test, y_pred)
print('accuracy score is: '+str(val))

# Logistic regression
LR = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X_train, y_train)
LR.predict(X_test)
round(LR.score(X_test,y_test), 4)
```

```
accuracy score is: 0.8785488958990536
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.6467
```

Accuracy score with KNN is 87% and with Logistic regression it is 64% .

Data Visualization, Analysis, Correction and Modeling

Let's use Sweet viz for visualizing the data to get the data insights. The code is given:

```
!pip install sweetviz
import sweetviz as sv

my_report = sv.analyze(data)
my_report.show_html() # Default arguments will generate to "SWEETVIZ_REPORT.html"
```

It generates a full fledged report with many insights of data readily available

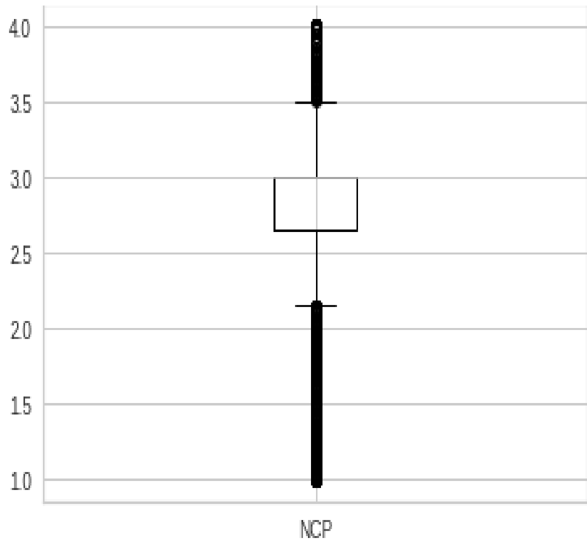
Data duplication

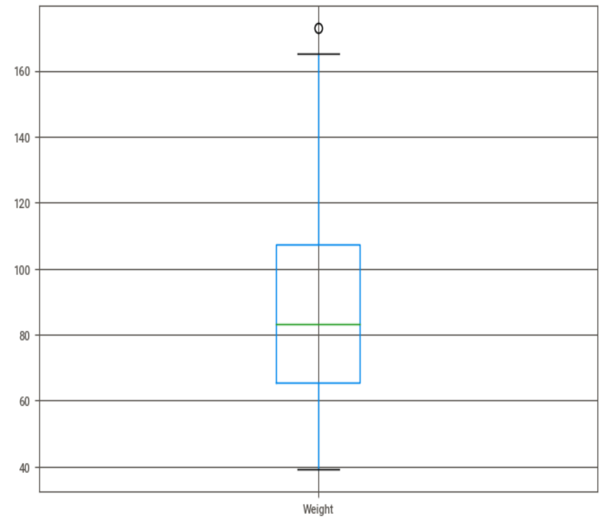
According to Sweet viz report there are no duplicate rows in this dataset.

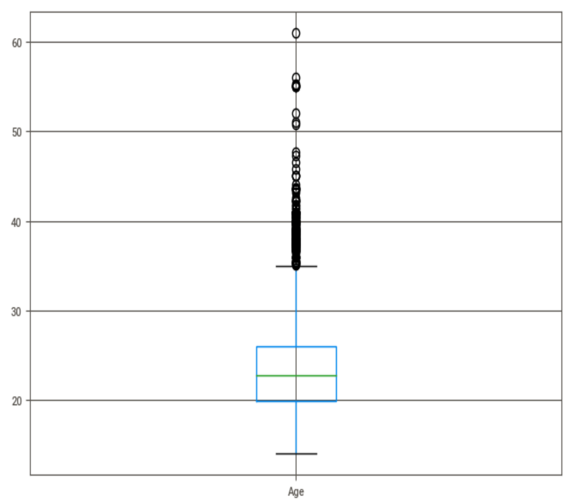
```
[48] # issues:
     # checking duplicate rows
     bool_series = data.duplicated()
     bool_series

0       False
1       False
2       False
3       False
4       False
        ...
2106    False
2107    False
2108    False
2109    False
2110    False
Length: 2111, dtype: bool
```

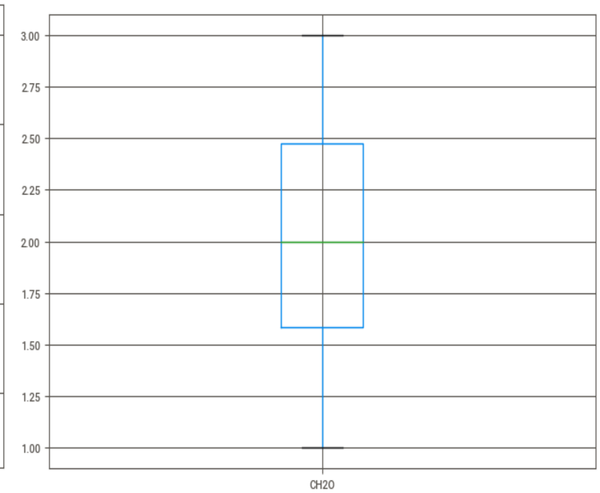
BOXPLOTS

```
# import the required library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
for i in data.columns:
  data[[i]].boxplot()
  plt.show()
```









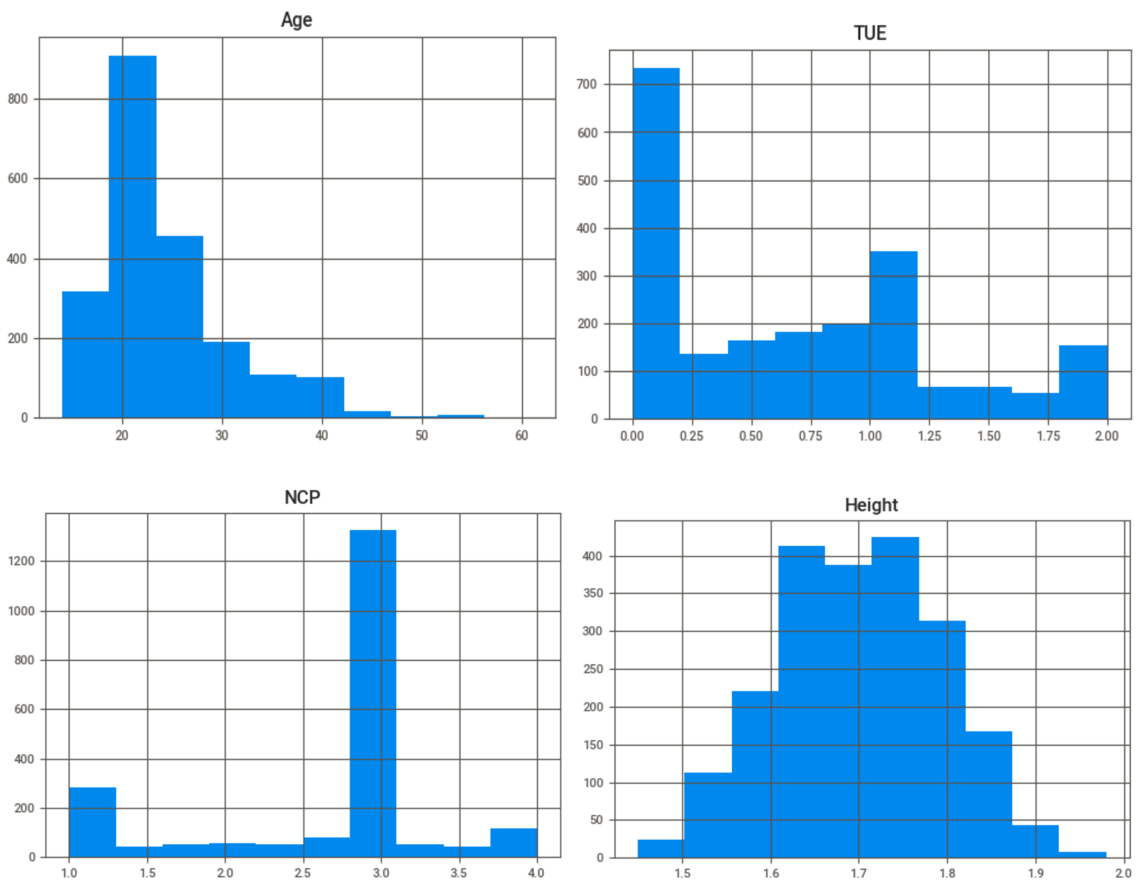
Here are the only few boxplots, I choose these because these boxplots show all the 5 measures. These boxplot shows that there are outliers in multiple features of dataset.

Data Distribution:

Data distribution can be analyzed be using histograms of multiple features.

```
# Histogram
for i in data.columns:
    data[[i]].hist()
    plt.show()
```

Output:

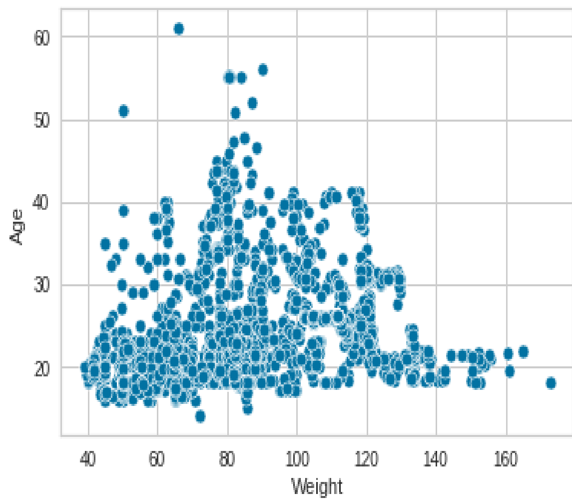


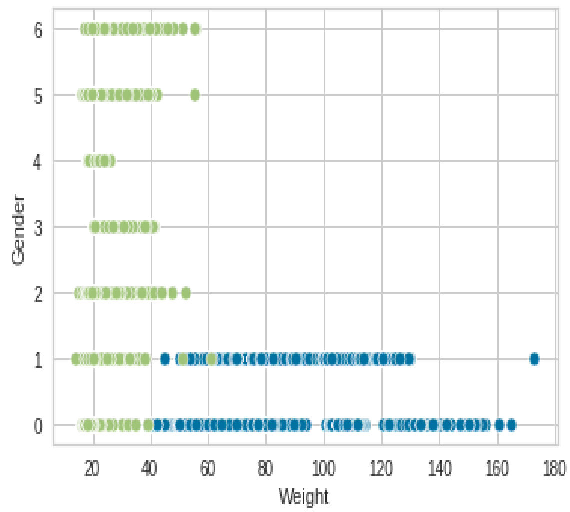
Histograms tells us about how data is distributed. Here are the few histograms of some features which shows that either data is normally distributed or not .

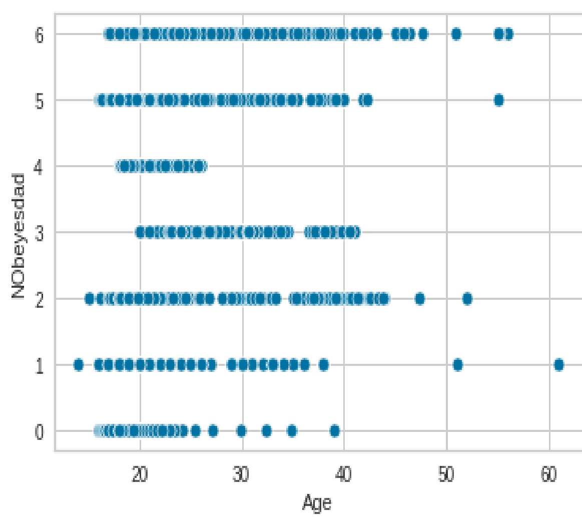
Scatter plot for Correlation

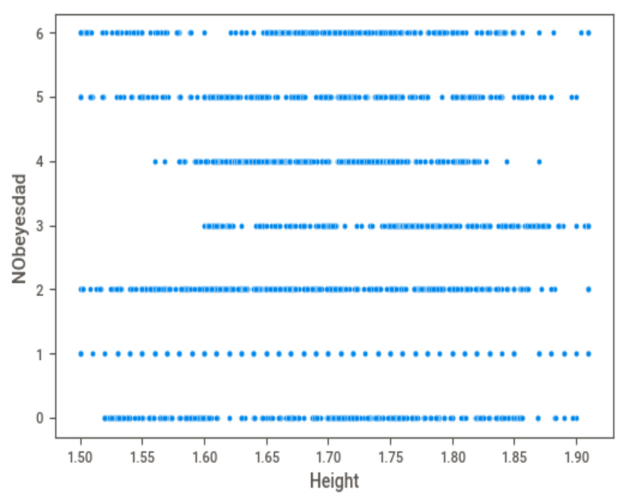
```
# Scatter plot for corelation
import seaborn as sns
sns.scatterplot(x="Weight", y="Age", data=data);
```

```
sns.scatterplot(x="Weight", y="Gender", data=data);
```









```
# Removing highly correlated features having correlation > 0.90
cor_matrix = data.corr().abs()
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(np.bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.90)]
data = data.drop(to_drop, axis=1)
data.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   int64  
 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   int64  
 5   FAVC                            2111 non-null   int64  
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   int64  
 9   SMOKE                           2111 non-null   int64  
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   int64  
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   int64  
 15  MTRANS                          2111 non-null   int64  
 16  NObeyesdad                      2111 non-null   int64  
dtypes: float64(8), int64(9)
```

After removing the highly correlated features ,the number of columns remain same, means these scatter plots shows that there are no features which are highly correlated.

Preprocessing

Normalization:

The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range.

```python
from sklearn import preprocessing

# x_array = np.array(data['Weight'])
# normalized_arr = preprocessing.normalize([x_array])
# print(normalized_arr)
for i in data.columns:
  x_arr=np.array(data[i])
  normalized_arr = preprocessing.normalize([x_arr])
  print(normalized_arr)
```

Here is the output:

```
[[0.         0.         0.0305995  ... 0.         0.         0.        ]]
[[0.01819024 0.01819024 0.01992264 ... 0.01951036 0.02110236 0.02049842]]
[[0.02068913 0.01941202 0.02298792 ... 0.02237754 0.02221463 0.02220679]]
[[0.01539873 0.01347389 0.0185266  ... 0.03216635 0.03208389 0.03211421]]
[[0.02407019 0.02407019 0.02407019 ... 0.02407019 0.02407019 0.02407019]]
[[0.         0.         0.         ... 0.02314964 0.02314964 0.02314964]]
[[0.01757187 0.02635781 0.01757187 ... 0.02635781 0.02635781 0.02635781]]
[[0.0233528 0.0233528 0.0233528 ... 0.0233528 0.0233528 0.0233528]]
[[0.02270237 0.02270237 0.02270237 ... 0.02270237 0.02270237 0.02270237]]
[[0.         0.15075567 0.         ... 0.         0.         0.        ]]
[[0.02073398 0.03110097 0.02073398 ... 0.0212958  0.02957017 0.02968601]]
[[0.         0.10206207 0.         ... 0.         0.         0.        ]]
[[0.         0.04944479 0.0329632  ... 0.02330842 0.0187743  0.01691757]]
[[0.02428224 0.         0.02428224 ... 0.01569332 0.01423024 0.01734085]]
[[0.02806682 0.01871121 0.00935561 ... 0.01871121 0.01871121 0.01871121]]
[[0.02435967 0.02435967 0.02435967 ... 0.02435967 0.02435967 0.02435967]]
[[0.00605916 0.00605916 0.00605916 ... 0.02423664 0.02423664 0.02423664]]
```

Imbalance:

Now let's see the bar graph for Obesity classes. We can easily highlight that there is a class imbalance in the data set. Overweight_level2 has high proportion as compare to other classes such as normal weight.

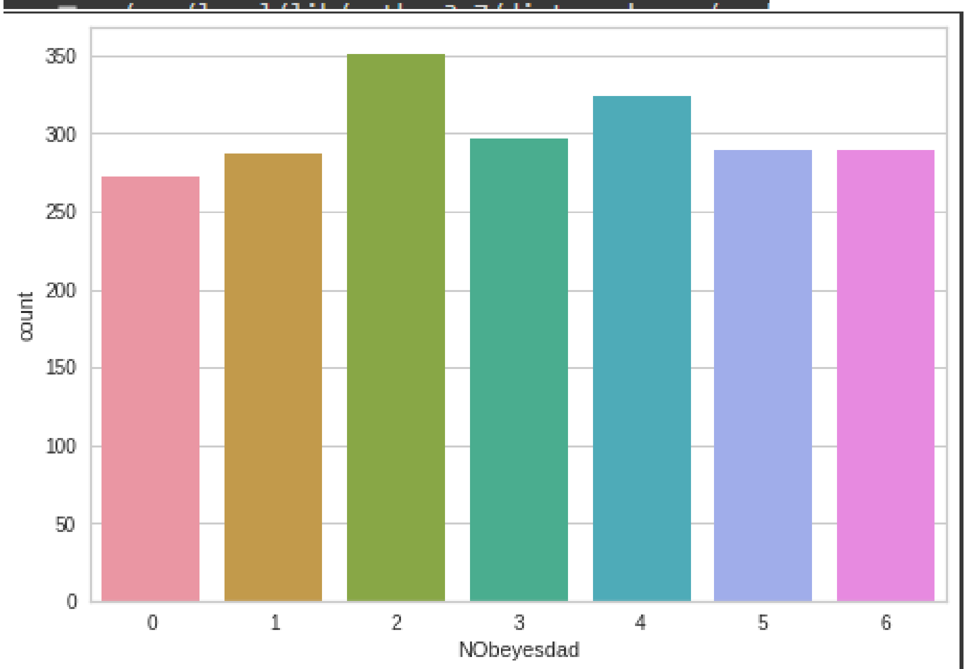
1st way:

```
# Dataimbalance
NObeyesdad_all = list(data.shape)[0]
NObeyesdad_cat = list(data['NObeyesdad'].value_counts())

print("\n \t The data has {} NObeyesdad, {} normal and {} overweight_level1 and {} overweight_level2 and {} obesity_level1 and {} obesity_level2 and {}
```

```
ta has 2111 NObeyesdad, 351 normal and 324 overweight_level1 and 297 overweight_level2 and 290 obesity_level1 and 290 obesity_level2 and 287 insufficient.
```

2nd way:

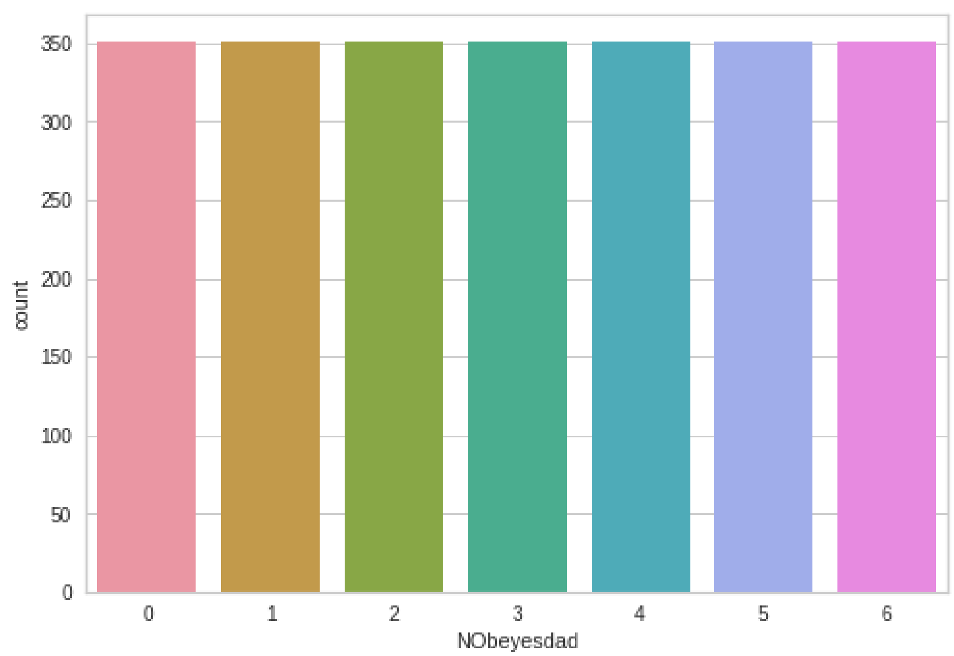
```
X=data.drop(["NObeyesdad"],axis=1)
Y=data['NObeyesdad']
sns.countplot(Y,label="count")
```



Applying SMOTE(oversampling) for Balancing the data

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X=data.drop(["NObeyesdad"],axis=1)
Y=data['NObeyesdad']


X, Y = oversample.fit_resample(X,Y)
```

```
[54] # Balanced
     sns.countplot(Y,label="count")
     print(X.shape)
```



After applying SMOTE(Synthetic Minority Over-sampling Technique) the classes are balanced now.

Checking Outliers

```
num_cols = data.select_dtypes(['int64','float64']).columns
for column in num_cols:
    q1 = data[column].quantile(0.25)    # First Quartile
    q3 = data[column].quantile(0.75)    # Third Quartile
    IQR = q3 - q1                            # Inter Quartile Range

    llimit = q1 - 1.5*IQR                       # Lower Limit
    ulimit = q3 + 1.5*IQR                        # Upper Limit

    outliers = data[(data[column] < llimit) | (data[column] > ulimit)]
    print('Number of outliers in "' + column + '" : ' + str(len(outliers)))
    print(llimit)
    print(ulimit)
    print(IQR)
```

Output:

```
2.537334499999993
170.3666905
41.957339000000005
Number of outliers in "family_history_with_overweight" : 385
1.0
1.0
0.0
Number of outliers in "FAVC" : 245
1.0
1.0
0.0
Number of outliers in "FCVC" : 0
0.5
4.5
1.0
Number of outliers in "NCP" : 579
2.146845
3.5118929999999997
0.34126199999999995
Number of outliers in "CAEC" : 346
2.0
2.0
0.0
Number of outliers in "SMOKE" : 44
0.0
0.0
0.0
Number of outliers in "CH2O" : 0
0.24590124999999996
3.81633125
0.8926075
Number of outliers in "SCC" : 96
0.0
0.0
0.0
Number of outliers in "FAF" : 0
-2.18875375
```

Removing outliers:

```
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] < fence_high)]
    return df_out
for c in data.columns[:-1]:
    df = remove_outlier(data, c)
len(df)
```

I have created method to remove the outliers from data.

Training model again to check accuracy

```
# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score
val = accuracy_score(y_test, y_pred)
print('accuracy score is: '+str(val))

# Logistic regression
LR = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X_trai
LR.predict(X_test)
round(LR.score(X_test,y_test), 4)
```

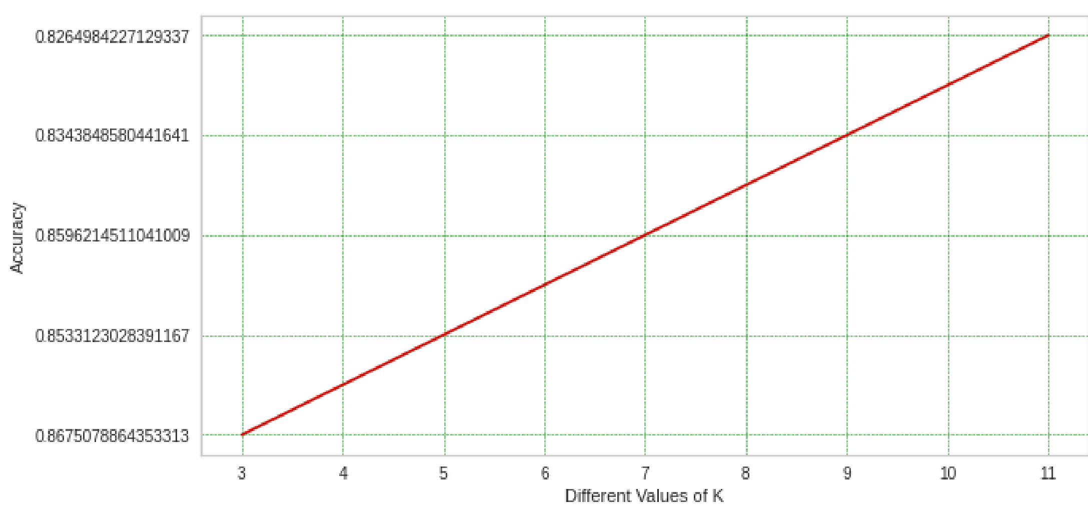
```
accuracy score is: 0.861198738170347
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.6956
```

The accuracy decreased from 87% to 86% according to KNN Classifier. And With Logistic regression the accuracy increases from 64% to 69%.

Results Presentation(Performance Metrics)

Loss Graph

```
neighbour = [3,5,7,9,11]
Accuracy = []
accuracyRate = []
i=30
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30)
for j in neighbour:
    from sklearn.neighbors import KNeighborsClassifier
    classifier = KNeighborsClassifier(n_neighbors=j)
    classifier.fit(X_train, Y_train)
    Y_pred = classifier.predict(X_test)
    from sklearn.metrics import accuracy_score
  #Printing
# print('accuracy score is: '+str(accuracy_score(Y_test, Y_pred)))
    Accuracy.append(str(accuracy_score(Y_test, Y_pred)))
    accuracyRate.append([0.3,j,str(accuracy_score(Y_test, Y_pred))])

#Graph For K values
plt.figure(figsize=(10,5))
plt.xlabel('Different Values of K')
plt.ylabel('Accuracy')
plt.plot(neighbour,Accuracy, color = 'r', label = "Accuracy at different K-values, when test-size is "+str(i)
                )

    # plt.legend(bbox_to_anchor=(1, 1)

         #bbox_transform=plt.gcf().transFigure)
plt.grid(axis='both', color = 'green', linestyle = '--' , linewidth = 0.5)
plt.show()
Accuracy = []
```



Confusion Matrix

```
# Confussion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split, RepeatedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier

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

model = KNeighborsClassifier(3)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Predicted	0	1	2	3	4	5	6	All
True								
0	79	3	0	0	0	0	0	82
1	9	57	0	0	0	15	5	86
2	0	0	99	1	0	0	6	106
3	0	0	1	84	1	0	3	89
4	0	0	0	0	97	0	0	97
5	1	4	0	0	0	77	5	87
6	0	1	6	2	0	1	77	87
All	89	65	106	87	98	93	96	634

Confusion matrix defines the performance of a classification algorithm. With this I have visualized and summarized the performance of a classification algorithm.

Classification Report:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.92        82
           1       0.88      0.66      0.75        86
           2       0.93      0.93      0.93       106
           3       0.97      0.94      0.95        89
           4       0.99      1.00      0.99        97
           5       0.83      0.89      0.86        87
           6       0.80      0.89      0.84        87

    accuracy                           0.90       634
   macro avg       0.90      0.90      0.89       634
weighted avg       0.90      0.90      0.90       634
```

Sensitivity and precision for all six

AUCROC:

```
y_pred_prob = model.predict_proba(X_test)
roc_auc_score(y_test,y_pred_prob,multi_class='ovr')
```

The AUC score of my dataset is 0.98