A
MINI PROJECT REPORT ON

"Body Mass Index (BMI) Prediction"

Prepared By

Miss. Shinde Rutuja N.(121)

Miss. Takale Shraddha P.(124)

Miss. Take Vaishnavi A.(125)

(B.E. Computer)

SAVITRIBAIPHULE PUNE UNIVERSITY

In the academic year 2021-22

Department of Computer Engineering

Sanjivani Rural Education Society's

Sanjivani College of Engineering

Kopargaon - 423 603.

ACKNOWLEDGEMENT

The entire session of mini project completion phase so far was a great experience providing us with great insight and innovation into learning various data mining concepts and achievement of it. As is rightly said, for the successful completion of any work, people are the most important asset our mini project would not be materialized without the cooperation of many of the people involved. We would like to thank our family for their unconditional love and support. And being there for us, despite all the troubles and problems that grow in life.

We express our sincere gratitude to **Prof. D. B. Kshirsagar,** Head of Department (Computer) SRESCOE for his unending support and encouragement during the years We have studied under his tutelage.

We are greatly indebted to project guide **Dr. T.Bhaskar ,** Assistant Professor (Computer), for providing valuable guidance at every stage of this project work.

Our sincere thanks go to all the teachers and staff for their help and understanding.

Miss. Shinde Rutuja(121)

Miss. Takale Shraddha(124)

Miss. Take Vaishnavi(125)

(B.E Computer B)

Contents

ABSTRACT: -

Three characteristics of different individuals like gender, height and weight are stored in a dataset and used for training. Based on this training, the body mass index (BMI) of individual is predicted using data mining concepts. Before testing the dataset, it is pre-processed using different data mining concepts like handling missing values, data discretization, normalization etc. This preprocessed data can then be used to classify/predict user BMI based on past classifications. The system analyses user gender, height and weight. System then predicts new user BMI status based on data stored by classification of previous user data.

INTRODUCTION: -

- The Body Mass Index (BMI) of individuals are predicted using data mining concept.
- Three characteristics of different individuals like gender, height and weight are stored in a dataset and used for training.
- Before training data is preprocessed and then that data is used for prediction based on past classification.
- The prediction will show whether the person is Extremely Weak, Weak, Normal, Overweight, Obesity, Extreme Obesity.

For most adults, an ideal BMI is in the 18.5 to 24.9 range.

For children and young people aged 2 to 18, the BMI calculation takes into account age and gender as well as height and weight.

If your BMI is:

- below 18.5 — you're in the underweight range
- between 18.5 and 24.9 — you're in the healthy weight range
- between 25 and 29.9 — you're in the overweight range
- between 30 and 39.9 — you're in the obese range

SCOPE: -

- The data and the information are gained from the learning system can be used as to determine the BMI of an individual based on their gender, height and weight.
- This system will help individual to know about their body and to care of their health.

OBJECTIVES: -

- Understand the Dataset & cleanup.
- Build classification models to predict various categories of BMI.
- Also fine-tune the hyperparameters & compare the evaluation metrics of various classification algorithms.

We aim to solve the problem statement by creating a plan of action, here are some of the necessary steps:

1. Data Exploration
2. Exploratory Data Analysis (EDA)
3. Data Pre-processing
4. Data Manipulation
5. Feature Selection/Extraction
6. Predictive Modelling
7. Project Outcomes & Conclusion

METHODOLOGY: -

1. Random Forest Algorithm

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

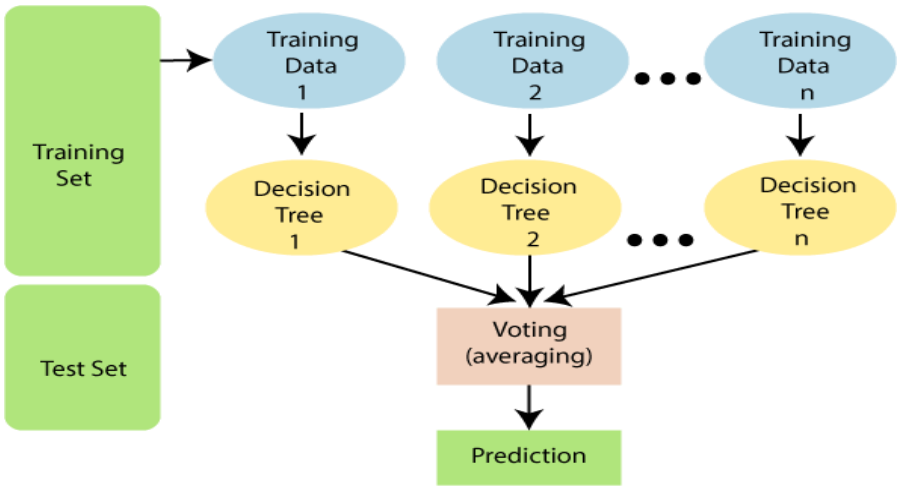


Fig .1: Working of the Random Forest Algorithm

2. Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

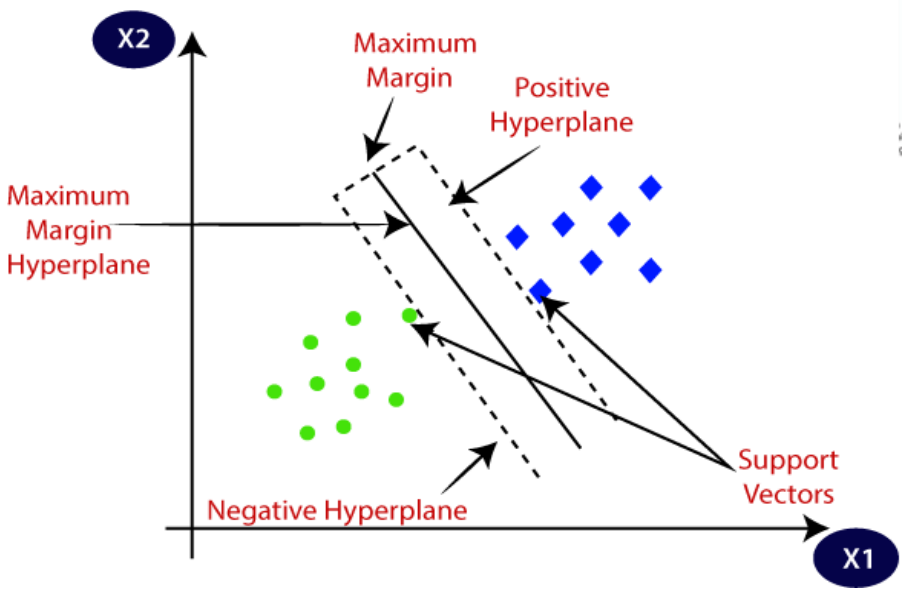


Fig.2: SVM hyperplane/ decision boundary

3. K-Nearest Neighbor (KNN) Algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

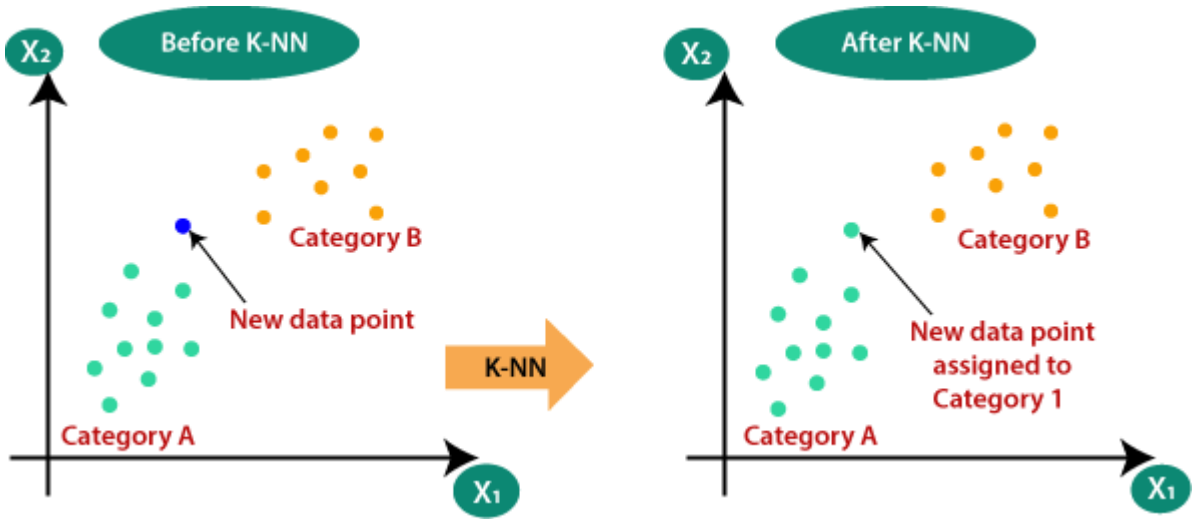


Fig.3: Working of K-NN

SYSTEM REQUIREMENT: -

Software Components:

- Operating System: Windows 10.
- Jupyter Notebook
- Matplotlib

Hardware Components:

- Processer: i3
- RAM: 1GB
- Hard Disk: 5 GB
- Monitor

IMPLEMENTATION: -

1. Data Exploration.

#Importing the basic librarires

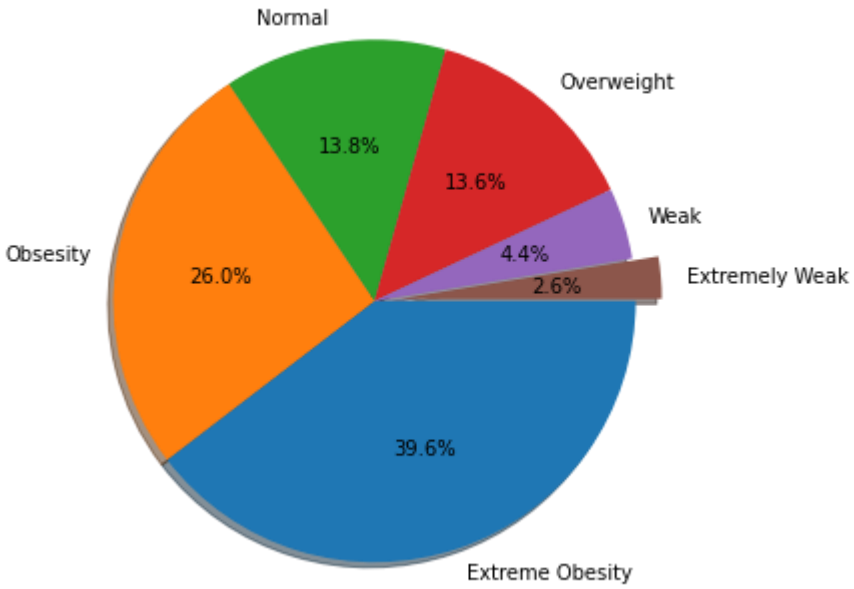
```
import os
import math
import scipy
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import tree
from scipy.stats import randint
from scipy.stats import loguniform
from IPython.display import display

from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.svm import SVC

from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier


from scikitplot.metrics import plot_roc_curve as auc_roc
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, \
f1_score, roc_auc_score, roc_curve, precision_score, recall_score

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', 50)
#Importing the dataset
```

```
df = pd.read_csv('../input/bmidataset/bmi.csv')
#df.drop([''],axis=1, inplace=True)

target = 'Index'
labels = ['Extremely Weak','Weak','Normal','Overweight','Obsesity','Extreme Obesity']
features = [i for i in df.columns.values if i not in [target]]

original_df = df.copy(deep=True)
display(df.head())

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {}
samples.'.format(df.shape[1], df.shape[0]))

#Checking number of unique rows in each feature

nu = df[features].nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df[features].shape[1]):
    if nu.values[i]<=7:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical
features.'.format(len(nf),len(cf)))
#Checking the stats of all the columns
 display(df.describe())
```

2. Exploratory Data Analysis (EDA)

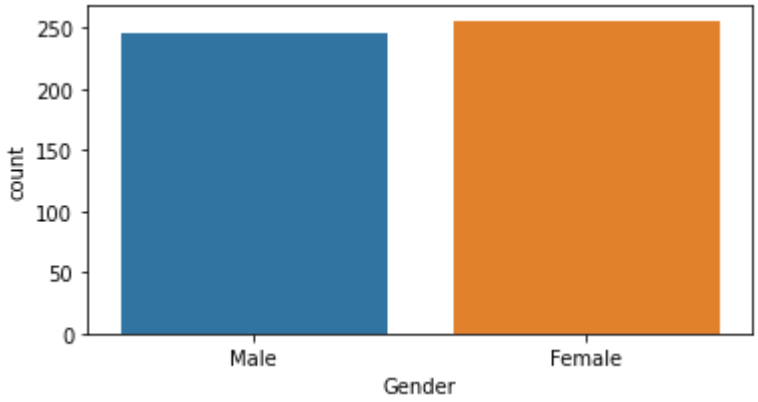
#Let us first analyze the distribution of the target variable

```
MAP={}
for e, i in enumerate(sorted(df[target].unique())):
    MAP[i]=labels[e]
#MAP={0:'Not-Survived',1:'Survived'}
df1 = df.copy()
df1[target]=df1[target].map(MAP)
explode=np.zeros(len(labels))
explode[-1]=0.1
print('\033[1mTarget Variable Distribution'.center(55))
plt.pie(df1[target].value_counts(), labels=df1[target].value_counts().index, counterclock=False,
shadow=True,
        explode=explode, autopct='%1.1f%%', radius=1, startangle=0)
plt.show()
```



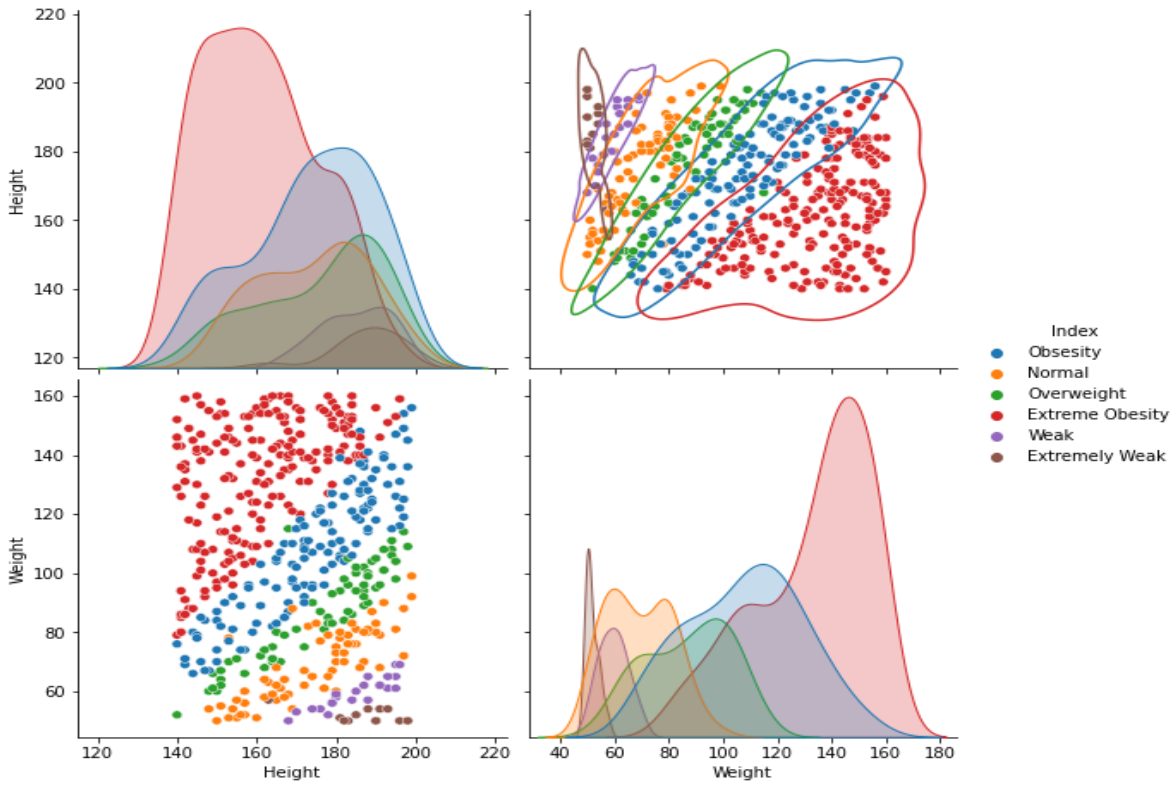
#Visualising the categorical features

```
print('\033[1mVisualising Categorical Features:'.center(100))

n=3
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=6:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    else:
        plt.subplot(3,1,i-1)
        sns.countplot(df[cf[i]])
plt.tight_layout()
plt.show()
```



#Understanding the relationship between all the features

```
g=sns.pairplot(df1, hue=target, size=4)
g.map_upper(sns.kdeplot, levels=1, color=".2")
plt.show()
```



3. Data Preprocessing.

```
#Removal of any Duplicate rows (if any)

counter = 0
r,c = original_df.shape

df1 = df.copy()
df1.drop_duplicates(inplace=True)
df1.reset_index(drop=True,inplace=True)

if df1.shape==(r,c):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped ---> {r-df1.shape[0]}')
#Check for empty elements

nvc = pd.DataFrame(df1.isnull().sum().sort_values(), columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null Values']/df1.shape[0],3)*100
print(nvc)

#Converting categorical Columns to Numeric

ecc = nvc[nvc['Percentage']!=0].index.values
dcc = [i for i in df.columns if i not in ecc]
```

```
df3 = df1[dcc]
fcc = [i for i in cf if i not in ecc]

#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    if df3[i].nunique()==2:
        if oh==True: print("\033[1m\nOne-Hot Encoding on features:\033[0m")
        print(i);oh=False
        df3[i]=pd.get_dummies(df3[i], drop_first=True, prefix=str(i))
    if (df3[i].nunique()>2 and df3[i].nunique()<17):
        if dm==True: print("\n\033[1mDummy Encoding on features:\033[0m")
        print(i);dm=False
        df3 = pd.concat([df3.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df3[i], drop_first=True,
prefix=str(i)))],axis=1)
df3.shape
#Final Dataset size after performing Preprocessing

df = df5.copy()
plt.title('Final Dataset Samples')
plt.pie([df.shape[0], original_df.shape[0]-df4.shape[0], df5.shape[0]-df4.shape[0]], radius = 1,
shadow=True,
        labels=['Retained','Dropped','Augmented'], counterclock=False, autopct='%1.1f%%',
pctdistance=0.9, explode=[0,0,0])
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78, shadow=True,
colors=['powderblue'])
plt.show()

print('\n\033[1mInference:\033[0mThe final dataset after cleanup has {} samples & {}
columns.'.format(df.shape[0], df.shape[1]))
```

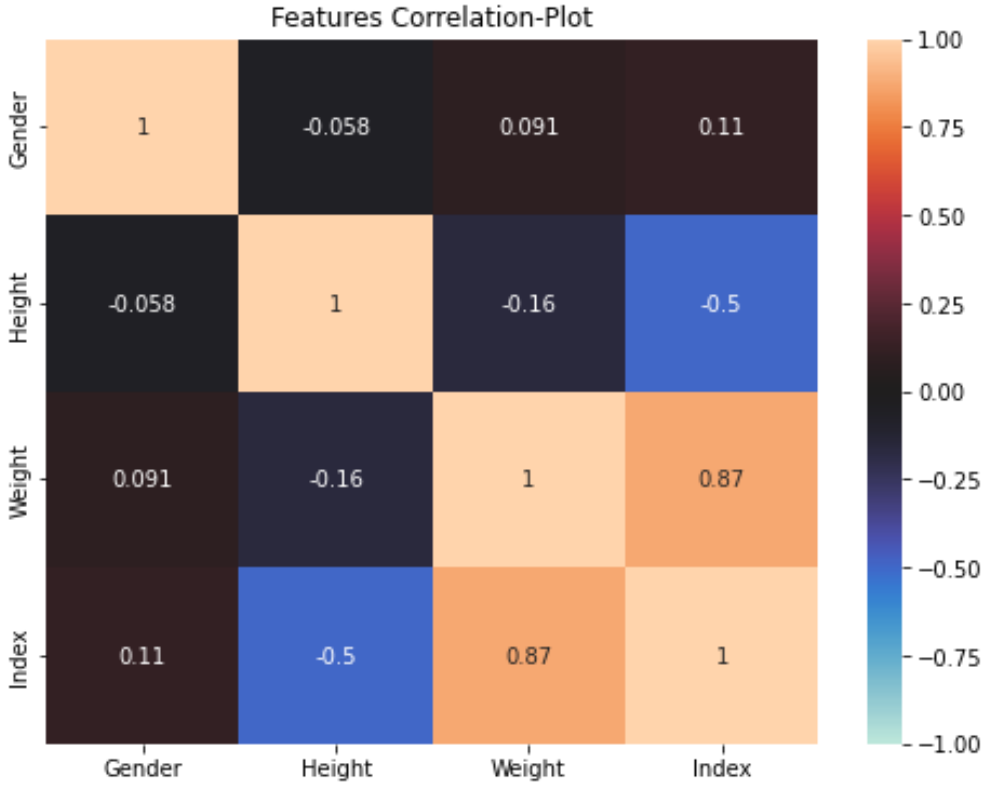
4. Data Manipulation.

```
#Splitting the data intro training & testing sets

df = df5.copy()

X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test_size=0.2,
random_state=0)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  --->
',Train_X.shape,Train_Y.shape,'\nTesting set   ---> ', Test_X.shape,", Test_Y.shape)
#Feature Scaling (Standardization)

std = StandardScaler()
```

```
print('\033[1mStandardardization on Training set'.center(100))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(100))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

5. Feature Selection/Extraction.

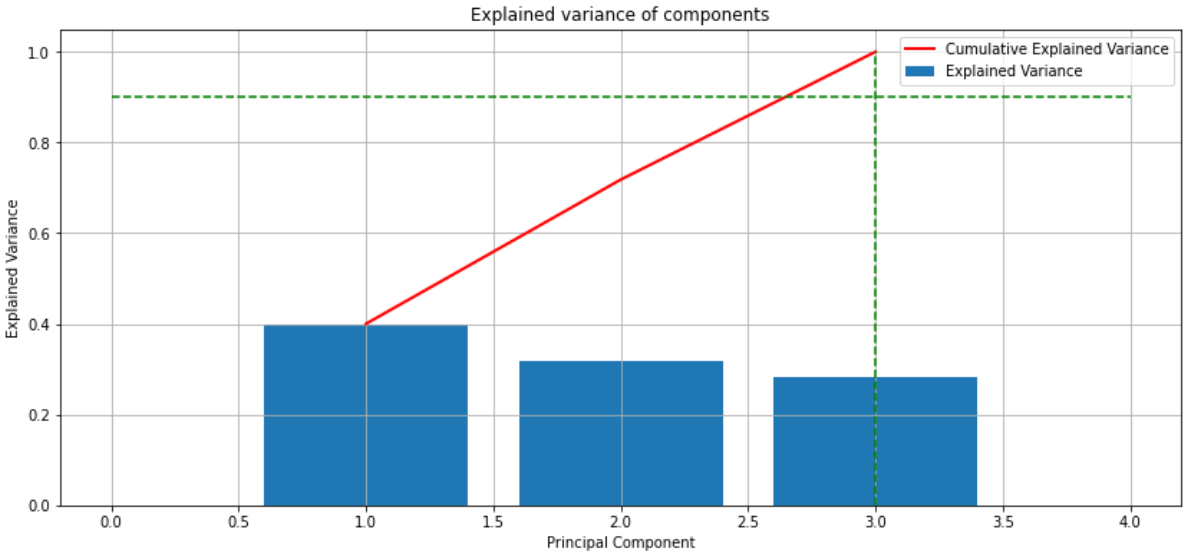
```
#Checking the correlation

features = df.columns
plt.figure(figsize=[8,6])
plt.title('Features Correlation-Plot')
sns.heatmap(df[features].corr(), vmin=-1, vmax=1, center=0, annot=True) #,
plt.show()
```



Correlation plt between the variables convey lot of information about the relationship between them. There seems to be strong multicollinearity in the dataset.
Let us check PCA techniques if we can improve the model's performance by performing Feature Selection/Extraction steps to take care of these multi-collinearity.

- Decomposition Method - Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA

pca = PCA().fit(Train_X_std)

fig, ax = plt.subplots(figsize=(14,6))
x_values = range(1, pca.n_components_+1)
ax.bar(x_values, pca.explained_variance_ratio_, lw=2, label='Explained Variance')
ax.plot(x_values, np.cumsum(pca.explained_variance_ratio_), lw=2, label='Cumulative Explained
Variance', color='red')
plt.plot([0,pca.n_components_+1],[0.90,0.90],'g--')
plt.plot([3,3],[0,1], 'g--')
ax.set_title('Explained variance of components')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Explained Variance')
plt.grid()
plt.legend()
plt.show()
```



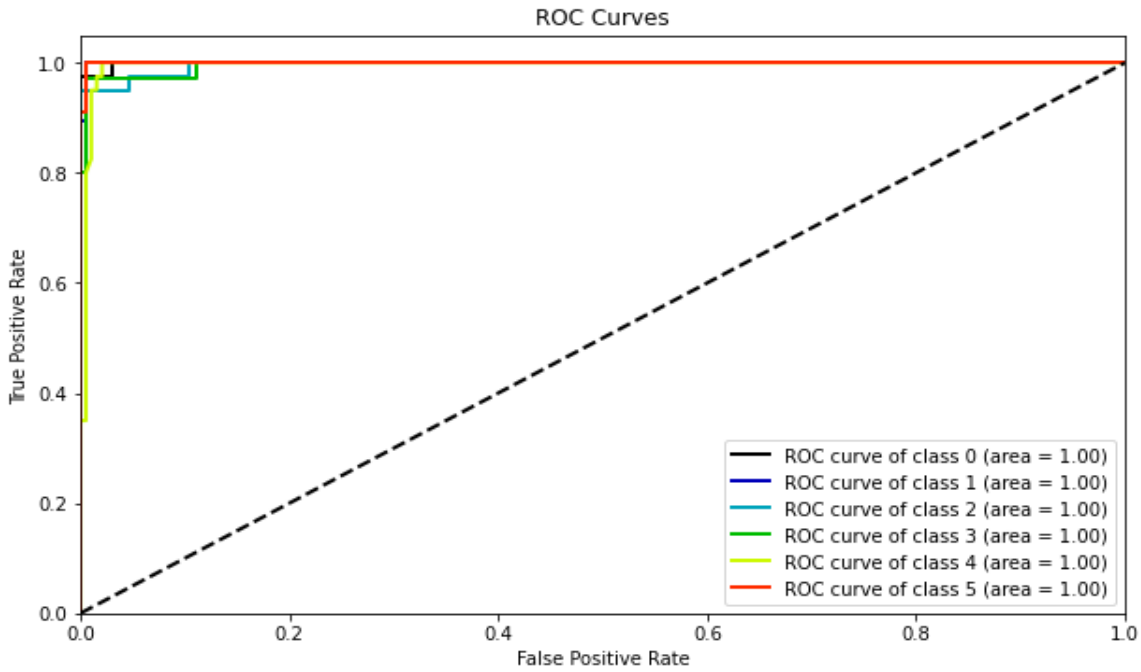
6. Predictive Modeling.

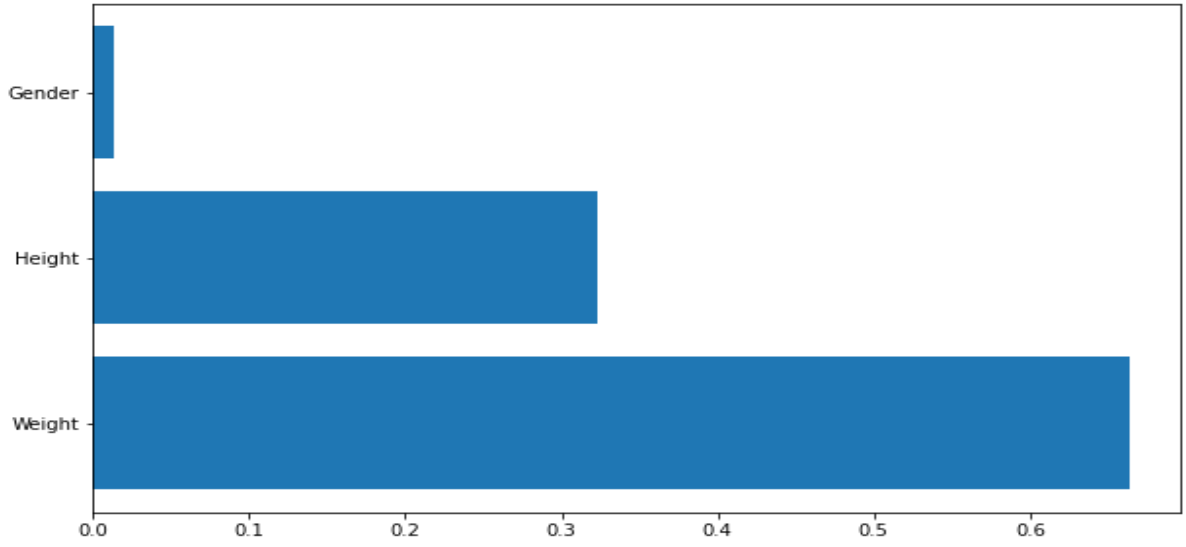
```
#Let us define functions to summarise the Prediction's scores .

#Classification Summary Function
def Classification_Summary(pred,pred_prob,i):
    Evaluation_Results.iloc[i]['Accuracy']=round(accuracy_score(Test_Y, pred),3)*100
    Evaluation_Results.iloc[i]['Precision']=round(precision_score(Test_Y, pred,
average='weighted'),3)*100 #
    Evaluation_Results.iloc[i]['Recall']=round(recall_score(Test_Y, pred, average='weighted'),3)*100
#
    Evaluation_Results.iloc[i]['F1-score']=round(f1_score(Test_Y, pred, average='weighted'),3)*100 #
    Evaluation_Results.iloc[i]['AUC-ROC score']=round(roc_auc_score(Test_Y, pred_prob,
multi_class='ovr'),3)*100 #[:, 1]
```

```
    print('{}{}\033[1m Evaluating {} \033[0m{}{}\n'.format('<'*3,'-'*35,Evaluation_Results.index[i],
'-'*35,'>'*3))
    print('Accuracy = {}%'.format(round(accuracy_score(Test_Y, pred),3)*100))
    print('F1 Score = {}%'.format(round(f1_score(Test_Y, pred, average='weighted'),3)*100)) #
    print('\n \033[1mConfusiton Matrix:\033[0m\n',confusion_matrix(Test_Y, pred))
    print('\n\033[1mClassification Report:\033[0m\n',classification_report(Test_Y, pred))

    auc_roc(Test_Y, pred_prob, curves=['each_class'])
    plt.show()

#Visualising Function
def AUC_ROC_plot(Test_Y, pred):
    ref = [0 for _ in range(len(Test_Y))]
    ref_auc = roc_auc_score(Test_Y, ref)
    lr_auc = roc_auc_score(Test_Y, pred)

    ns_fpr, ns_tpr, _ = roc_curve(Test_Y, ref)
    lr_fpr, lr_tpr, _ = roc_curve(Test_Y, pred)

    plt.plot(ns_fpr, ns_tpr, linestyle='--')
    plt.plot(lr_fpr, lr_tpr, marker='.', label='AUC = {}'.format(round(roc_auc_score(Test_Y,
pred)*100,2)))
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

1. Random Forest Classfier:¶

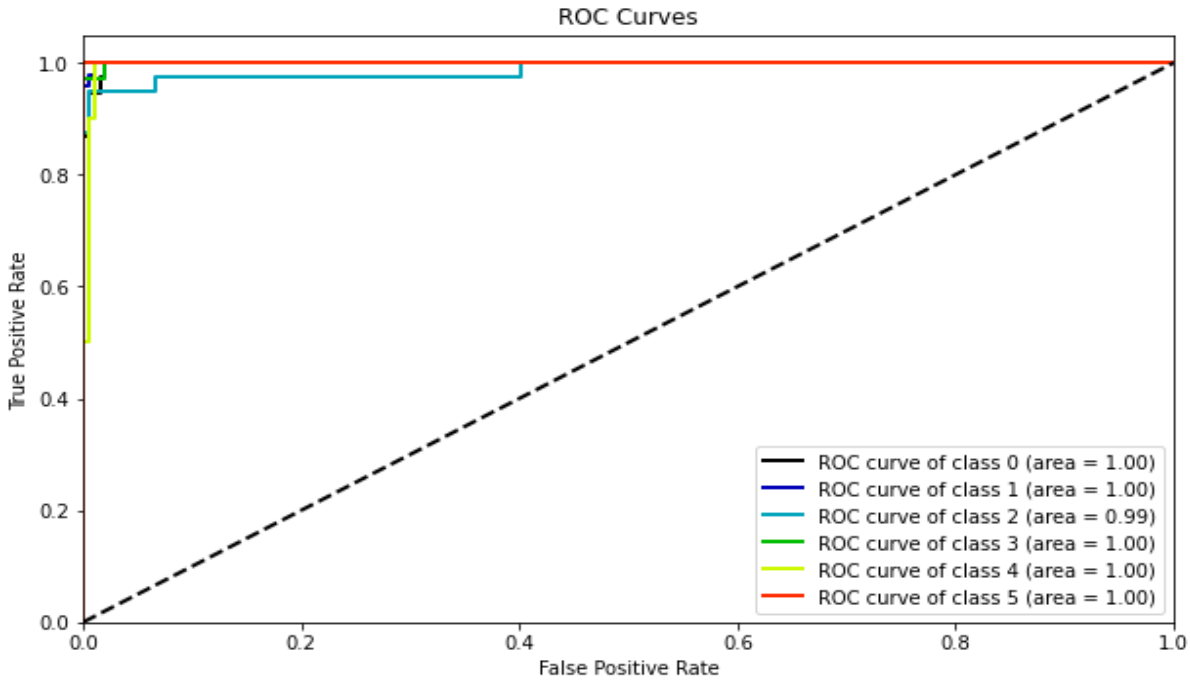
```
<<<------------------------------------ Evaluating Random Forest Classifier (RF) ------------
------------------------>>>

Accuracy = 97.0%
F1 Score = 97.0%

 Confusiton Matrix:
 [[37  1  0  0  0  0]
 [ 0 48  0  0  0  0]
 [ 0  0 37  2  1  0]
 [ 0  0  0 34  1  0]
 [ 0  0  0  0 39  1]
 [ 0  0  0  0  1 32]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        38
           1       0.98      1.00      0.99        48
           2       1.00      0.93      0.96        40
           3       0.94      0.97      0.96        35
           4       0.93      0.97      0.95        40
           5       0.97      0.97      0.97        33

    accuracy                           0.97       234
   macro avg       0.97      0.97      0.97       234
weighted avg       0.97      0.97      0.97       234
```





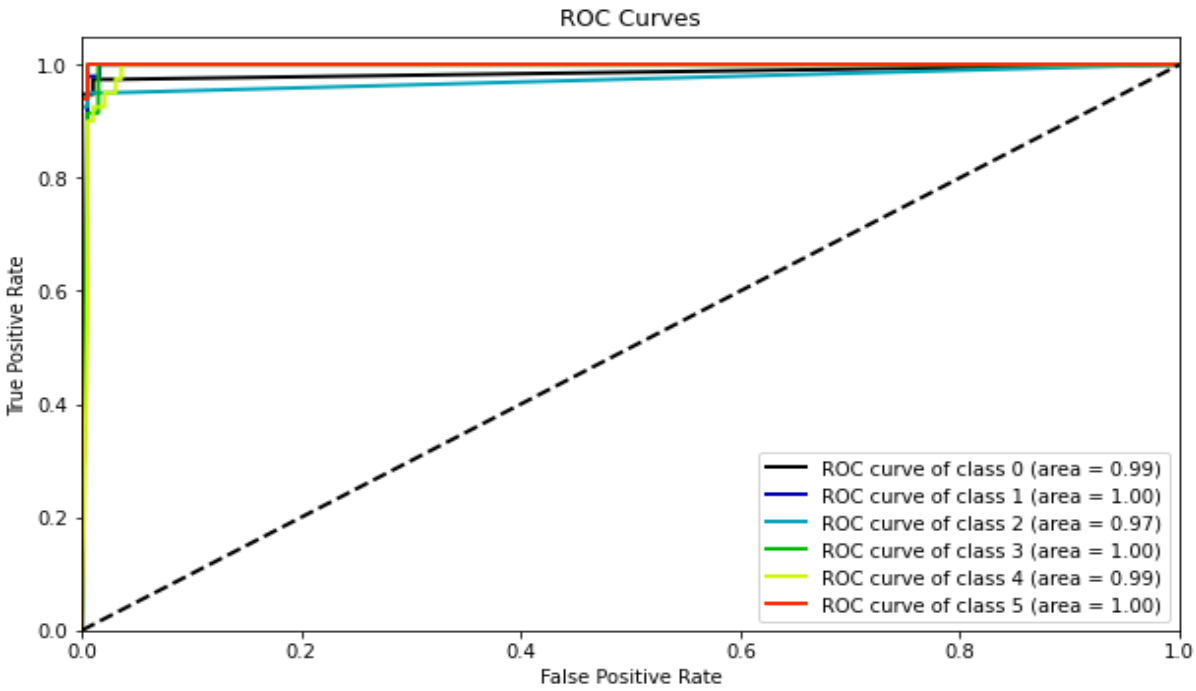
2. Support Vector Machine Classfier:¶

```
<<<-------------------------------------- Evaluating Support Vector Machine (SVM) -------------
------------------------>>>

Accuracy = 97.0%
F1 Score = 97.0%

 Confusiton Matrix:
 [[35  2  1  0  0  0]
 [ 1 47  0  0  0  0]
 [ 1  0 38  0  1  0]
 [ 0  0  0 34  1  0]
 [ 0  0  0  0 40  0]
 [ 0  0  0  0  0 33]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.92      0.93        38
           1       0.96      0.98      0.97        48
           2       0.97      0.95      0.96        40
           3       1.00      0.97      0.99        35
           4       0.95      1.00      0.98        40
           5       1.00      1.00      1.00        33

    accuracy                           0.97       234
   macro avg       0.97      0.97      0.97       234
weighted avg       0.97      0.97      0.97       234
```

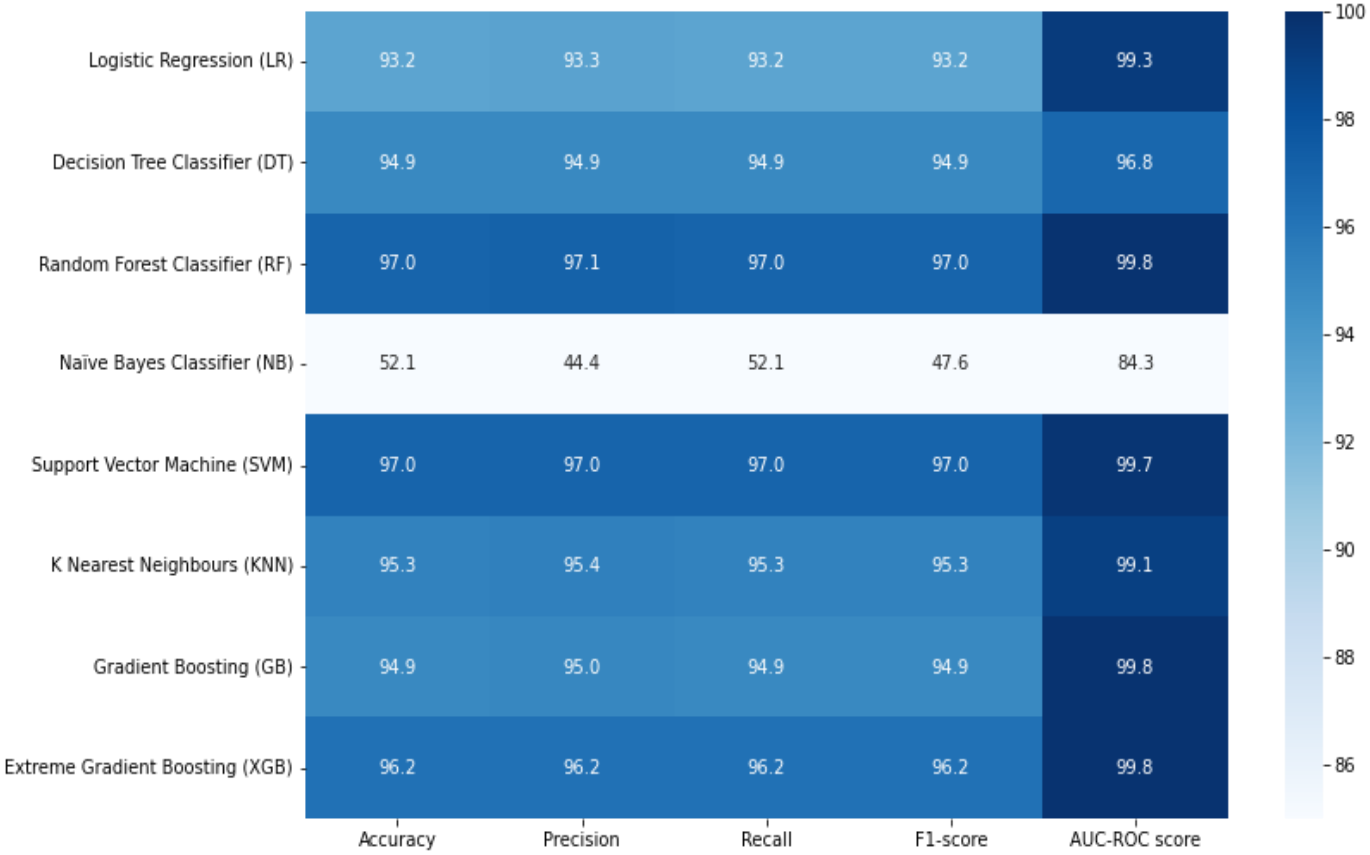


3. K-Nearest Neighbours Classifier:

```
<<<------------------------------------ Evaluating K Nearest Neighbours (KNN) ---------------
---------------------->>>

Accuracy = 95.3%
F1 Score = 95.3%

 Confusiton Matrix:
 [[36  2  0  0  0  0]
 [ 1 47  0  0  0  0]
 [ 1  0 37  1  1  0]
 [ 0  0  0 34  1  0]
 [ 0  0  0  2 37  1]
 [ 0  0  0  0  1 32]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        38
           1       0.96      0.98      0.97        48
           2       1.00      0.93      0.96        40
           3       0.92      0.97      0.94        35
           4       0.93      0.93      0.93        40
           5       0.97      0.97      0.97        33

    accuracy                           0.95       234
   macro avg       0.95      0.95      0.95       234
weighted avg       0.95      0.95      0.95       234
```



- Testing multiple algorithms with fine-tuning hyperparameters gave us some understanding on the model performance for various algorithms on this specific dataset.



CONCLUSION:

In the Project we evaluated several popular machine learning algorithms to accurately predict adolescents at risk of becoming overweight or obese at the teenage stage. This data can be helpful to classify persons using automated BMI prediction and classification. This prediction can help people to look after their health and to take necessary step towards their health.

REFERENCE:

- https://www.javatpoint.com/machine-learning-random-forest-algorithm
- https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
- https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

Sr No.	Roll No	Student Name(s)	Sign
1	121	Miss. Shinde Rutuja	
2	124	Miss. Takale Shraddha	
3	125	Miss. Take Vaishnavi	

Prof. **T.Bhaskar**
[Guide]