Obesity Prediction Using Machine Learning Methods

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology
In
Computer Science

By

Name: Tris Marie Joe
Reg. No.: 18BCE2299
Mobile No.: +971526110178
Mail Id.:trismarie.joe2018@vitstudent.ac.in

Name: Nidhi Mankala
Reg. No.: 18BCE2340
Mobile No.: 8374702343
Mail Id.: nidhi.mankala2018@vitstudent.ac.in

Under the guidance of

Prof. / Dr. Naveen Kumar N

SCOPE

VIT, Vellore.

June, 2022

DECLARATION

I hereby declare that the thesis entitled "Obesity Prediction Using Machine Learning Methods" submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science* to VIT is a record of bonafide work carried out by me under the supervision of Naveen Kumar N.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 03/06/2022

Signature of the Candidate

Tris Marie Joe
Nidhi Mankala

CERTIFICATE

This is to certify that the thesis entitled "Obesity Prediction Using Machine Learning Methods" submitted by **Tris Marie Joe & 18BCE2299, SCOPE & Nidhi Mankala & 18BCE2340**, VIT University, for the award of the degree of *Bachelor of Technology in Programme*, is a record of bonafide work carriedout by them under my supervision during the period, 01. 01. 2022 to 03.06.2022, as perthe VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 03/06/2022 **Signature of the Guide**

Internal Examiner **External Examiner**

Dr. VAIRAMUTHU S

SCOPE

ACKNOWLEDGEMENTS

We cordially thank our Prof. Naveen Kumar N for his precious guidance, the Dean of SCOPE, and SCHOOL OF CSE for their pleaded permission and opportunity given to us for completion of our project. Also, we would like to thank Professors for their ideas and execution of our plans successfully.

Student Name

Tris Marie Joe
Nidhi Mankala

Executive Summary

In the current times, obesity has become a worldwide disease that affects people all over the world regardless of gender or age. This work seeks to predict obesity using data from the 'Estimation of Obesity Levels based on Eating Habits and Physical Condition dataset' from the UC,IRVINE Machine Learning database. Certain ML methods that we feel will help predict obesity levels at a higher accuracy which we have implemented are Support Vector Machine with Radial Basis Kernel, Support Vector Machine with Linear Kernel, XGBoost (eXtreme Gradient Boosting), Adaboost with decision tree, Adaboost with Random Forest, Adaboost with SVM, Random Forest Classifier, Category Boosting, Light Gradient Boosting Method, Gradient Boosting and Adaboost with Extra Tree Classifier trained on the dataset. The machine learning approach with the highest prediction rate for the dataset was then evaluated using performance indicators such as accuracy, recall, precision rate, and f1number.

Keywords: Obesity, Machine learning, Prediction, Accuracy, Support Vector Machine, Neural Networks, XGBoost, Light Gradient Boosting Machine, Adaboost, CatBoost

TABLE OF CONTENTS

List of Figures

1. INTRODUCTION

1.1. **OBJECTIVE**

This work seeks to predict obesity using data from the 'Estimation of Obesity Levels based on Eating Habits and Physical Condition dataset' from the UC,IRVINE Machine Learning database. There are a total of 2111 instances of data collected, with 17 attributes. The data is classified using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II, and Obesity Type III in the class variable NObesity, which is the Obesity Level. The data is made up of 77% synthetic data created with the Weka tool and the SMOTE filter, and 24% data obtained directly from consumers via a web platform. The data is then utilized to determine and forecast an individual's level of obesity using machine learning approaches.

1.2. **MOTIVATION**

We propose to use data from the UC, IRVINE machine learning database in our research. It was developed to estimate obesity levels in persons aged 14 to 61 from Colombia ,Mexico, Peru, with a variety of eating routines and physical circumstances. The information was gathered through a web-based survey. The survey was available for 30 days online. After all of the data was acquired, it was preprocessed so that it could be used in various data mining methods. Because the classification categories were not equally presented, the dataset was unbalanced. The Weka tool and the SMOTE filter were used to generate synthetic data that was around 77 percent of the actual data once the balance problem was solved. The final distribution of data, which is the dataset, was formed after a few more filters were applied to different categories.

It is critical to test alternative ways in public sector, and we believe that applying Machine Learning methods will aid in improving forecasts and discovering good data structures within the current data, as well as assisting in the design of new policies.

Support Vector Machine(SVM) with Radial Basis Kernel,XGBoost (eXtreme Gradient Boosting), Adaboost with decision tree, Support Vector Machine(SVM) with Linear Kernel, Adaboost with Random Forest, Adaboost with SVM, Random Forest Classifier, Category Boosting, Light Gradient Boosting Method, Gradient Boosting, and Adaboost with Extra Tree Classifier are some of the machine learning methods that we have implemented.

The plan is to analyze ten different machine learning methods: Support Vector Machine(SVM) with Radial Basis Kernel,, XGBoost (eXtreme Gradient Boosting), Adaboost with decision tree, Support Vector Machine(SVM) with Linear Kernel, Adaboost with Random Forest, Adaboost with Gradient Boost, Random Forest Classifier, MLP Classifier, Gradient Boosting, and Adaboost with Extra Tree Classifier trained on the dataset. The machine learning approach with the highest prediction rate for the dataset is then calculated using performance defining entities such as accuracy, recall, precisionscore, f1 score,

1.3. **BACKGROUND**

In the current times, obesity has become a worldwide disease that affects people all over the world regardless of gender or age. The definition of obesity says it’s a complex disease which means that there is an excessive or unnatural amount of fat that is present in our body. For this type of condition there is not just one cause or symptoms through which you can predict obesity as it depends on many large factors such as age, gender, early childhood habits, family history, the amount of physical activity a person does, their way of life and many more. It is a chronic disorder characterized by an surplus of body fat that exceeds the required quantity, despite the fact that a certain amount is still required for energy storage, shock absorption, and other activities. The best criterion for determining obesity is the BMI (BMI). This is because the BMI is calculated using a person's height and weight. Because BMI measures body weight in relation to height, there is a strong link between BMI and total body fat in adults. Obesity is linked to a number of physical and mental health issues, including heart and circulatory illness, cardio metabolic risk, type 2 diabetes, depression, and hypertension. Data from the Global Burden of Disease study in 2017 shows that 4.7 million people have died prematurely due to obesity. This was four times more than the number of people who have died due to road accidents. Globally in 2017, due to obesity the number of deaths rose to 8% compared to 4.5% in the 90s.

2. PRODUCT DESCRIPTION AND GOALS

The perspective is to try different complex machine learning algorithms on the dataset and then use various performance metrics to find the most optimized machine learning algorithm to the dataset. The project's main feature is trying out various machine learning algorithms for prediction of obesity and the using different performance metrics to compare and contrast these machine learning algorithms to estimate the best model for this prediction model project.

2.1 GAPS IDENTIFIED BASED ON OUR LITERATURE SURVEY

Based on the papers that we have researched, we have found for this particular prediction based model, not many different algorithms have been researched about. Most of the algorithms are the simple and common algorithms such as K-means, decision trees etc. These algorithms also have not given good performance metrics. We have found different and complex algorithms which we ran with this dataset to find one machine learning algorithm that gives high performance metric making it the best algorithm for this type of prediction model. Following is the result of models implemented in the base paper.

Eduardo De-La-Hoz-Correa *et al.* / Journal of Computer Science 2019, 15 (1): 67.77
DOI: 10.3844/jcssp.2019.67.77

Table 3: Results of the implemented techniques

Method	Precision	Recall	TP. Rate	FP. Rate
J48	97,4%	97,8%	97,8%	0,2%
Naive Bayes	90,1%	91,1%	91,1%	6,0%
Simple Logistic	90,4%	91,6%	91,6%	4,1%

2.1.1. Table from Base paper showing other ML models used before for comparative analysis

2.2 GOALS

This study aims to predict obesity using data from the UC, IRVINE Machine Learning database for the obesity prediction dataset. There are a total of 2111 instances of data collected, with 17 properties. It is critical to test alternative ways in public health, and we believe that applying Machine Learning methods will aid in improving forecasts and discovering good data structures within the current data, as well as assisting in the design of new policies. The plan is to analyze ten different machine learning methods: Support Vector Machine(SVM) with Radial Basis Kernel, XGBoost (eXtreme Gradient Boosting), Adaboost with decision tree, Adaboost with Random Forest, Support Vector Machine(SVM) with Linear Kernel, Adaboost with Gradient Boost, Random Forest Classifier, MLP Classifier, Gradient Boosting, and Adaboost with Extra Tree Classifier trained on the dataset.

3. TECHNICAL SPECIFICATION

The plan is to do an analysis of various machine learning algorithms: Support Vector Machine(SVM) with Radial Basis Kernel, Adaboost with Gradient Boost, Support Vector Machine(SVM) with Linear Kernel, XGBoost (eXtreme Gradient Boosting), Adaboost with decision tree, Adaboost with Random Forest, Random Forest Classifier, MLP Classifier, Gradient Boosting and Adaboost with Extra Tree Classifier trained on the dataset and then assessed using performance measures like accuracy, recall, precision score, f1 score, and others to find the machine learning approach with the best prediction rate for the dataset.

As this is a prediction model, it can be used by developers as a prediction model for their front end project or application or by any users who wish to use our resultant data to help jumpstart other projects. We are also assuming that the attributes that are present in our dataset are the only ones that are needed to determine if the person is obese and thus we are depending on these attributes to train and test our dataset. For this model to work the domain requirements are Google Colab and Jupyter Notebook, and the user requirements are Windows 10 or above, Jupyter Notebook, Python interpretor, Google chrome(version 99 or above), 8GB RAM, 2GB VRAM.

This project product efficient as it is able to run a lot of different machine learning models in a short period of time, reliable as our dataset is preprocessed in such a way that it is possible to run different ML methods to find the best one for the dataset in our project, portable as it is easily be able to install from one computing source to another as all you require is Jypter Notebook to be able to run the code properly and usable as the project can be easily be downloaded and implemented for the user to access and work on the project.

This product will help future developers use this prediction model to create product that can help people check their health digitally, hence contributing to smoother economic transactions. It is fully digital and so has no harmful effects to the environment. As this product is digitized, it does not need any external resources. And since all the machine learning models can be deployed in cloud server, it can be stored for a long period of time. It also can be used to increase the knowledge of the people. It can help them know more about the different attributes that can result to obesity. This model also gives them a platform to analyze their condition without the help of doctors. As the dataset used for this product is collected using real surveys and hence no false information is being spread. The predictions are also based on the dataset which contains real time data. For this product, all the data is available online and we have used open source softwares and libraries.

4. DESIGN APPROACH AND DETAILS

4.1 DESIGN APPROACH

We start by importing our dataset, performing some methods for data cleaning to make our dataset ready. We then tried various different machine learning algorithms on this dataset and then compare the results of these algorithms with different performance metrics.

Data Cleaning

The first step of this is to check for null values. In order to check the missing values we use a function isnull(). We also use a sum function. Together these both will help identify the null values in a column and the total number of null values in each column. The next step is encoding categorical variables. A machine learning model's performance is influenced not only by the model andhyperparameters, but also by how various types of variables are processed and fed into the algorithm. Because most ML methods only accept numerical variables, preprocessing categorical data becomes crucial. In order for the model to understand and extract relevant information, we must convert the categorical variables to integers/numbers that are readable by machine. We then do multicollinearity which occurs when two or more independent variables are highly correlated with one another. We do this to find the collinearity between the attributes.

Support Vector Machine (Radial Basis Kernel and Linear Kernel)

Support Vector Machines (SVMs) are most commonly used to handle classification problems, which fall under the supervised machine learning area. The SVM method looks for a hyperplane that has the largest distance between these two classes. If classes are completely linearly distinguishable, a hardmargin can be used. A soft-margin is required otherwise. The radial basis(RBF) kernel function calculates their similarity, for any 2 given points z1 and z2. Because of its similarities to the K-Nearest Neighbor Algorithm(KNN), the RBF Kernel is famous. RBFK Support Vector Machines do not have the issue of space complexity as it only takes into consideration the support vectors during model building and not the rest of the existing data. It also includes all the perks of KNN. The RBFK Support Vector Machines includes 2 hyperparameters: 'C' for SVM and ‘γ’ for the RBF Kernel.

The Linear Kernel is used when the data is linearly separable/distinguishable, that is, when it can be split using only one single line. When a data set contains a significant number of attributes, it is most typically used. For training an SVM, a Linear Kernel is faster than any other Kernel. When using a Linear Kernel to train an SV Machine, just the C Regularisation parameter needs to be improved. On the other hand, when training with different kernels, it is important to optimize the parameter, indicating that running a grid search will be a long process.

XGBoost (eXtreme Gradient Boosting)

Gradient boosting is a machine learning technique for solving regressive and classifying predictive training problems. Models are transformed with a arbitrary differentiable loss function and a gradient descent optimization technique. Gradient boosting takes its name from the fact that, similar to a neural network(NN), the loss gradient is reduced when the training model is adjusted. Extreme Gradient Boosting(XGBoost) is a highly efficient open-source deployment of the gradient boosting algorithm. The crucial factors to use XGBoost are execution efficiency and model performance.

The mathematical formulae is as follows:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2$$

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

Adaboost (Decision Tree, Random Forest, Extra Tree Classifier, SVM)

AdaBoost(Adaptive Boosting), is an approach that is utilized as part of an Ensemble learning. Decision tree for one level is also known as tree with only one division and it is usually included in the Adaboost algorithm. These are represented by the name Decision Stumps. This method generates a model by giving all of the data points the same amount of weight. It therefore grants improperly classified points a greater weight. All points with higher weights are given more importance/priority in the next modelling or training process. It will prolong the training process until models exhibit a smaller error. AdaBoost is a relatively new boosting approach that has been widely used to increase the accuracy of any learning process. It's an ensemble strategy that combines a large number of weak hypotheses or classifiers with high error rates to build a low-error-rate hypothesis. This method is simple, quick, and easy to put into action. The AdaBoost and random forests algorithms are coupled to improve accuracy, stability, and reduce the over-fitting problem.

Initial weight

$$D_{t,i} = \frac{1}{n}$$

Select h (x) to make the smallest error rate Ej

$$E_j = \sum_{j=1}^{n} | h_j(x_j) - y_j |$$

Calculate weights

$$\beta_j = E_j / (1 - E_j); a = \log(1/\beta_j)$$

Adjust the weight

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t}, & if \quad h_t(x_i) = y_i \\ e^{\alpha_t}, & if \quad h_t(x_i) \neq y_i \end{cases},$$

Output function

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

Random Forest Classifier

The random forest classifier is a unique estimator that uses averaging and analysis to improve projected accuracy and which hereby reduces overfitting of the data by including several decision tree algorithms to different sub parts of the information. The random forest accumulates more randomness/uniqueness to the model as it expands the trees.Instead of searching for the most significant characteristic when dividing a node, it seeks for the best attribute from a sample group of attributes. Therefore, there is variability in a large amount, resulting in a efficient algorithm. Furthermore, the methodology for dividing a node in random forest only evaluates a random sample of the attributes.

Using the Gini index, we split out the dataset and the feature with the lowest Gini index is made the root node.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Furthermore, to calculate information gain, entropy is calculated.

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

*Information Gain= Entropy(S)- [(Weighted Avg) * Entropy (each feature)]*

Gradient Boosting

Another boosting procedures included in our project which is used to lower the algorithm's bias error is gradient boosting. Gradient boosting is an approach that is unique for its predictability and efficacy, particularly when working with large, complex datasets.

The main idea behind this methodology is to build models in a sequential order, with each model seeking to improve on the previous model's faults. It delivers a predictive model in the form of a assembly of trivial predictive models, the most frequent of which are decision trees. At the time when the decision tree classifier is a slow learner, the output is called gradient-boosted trees.

LightGBM (Light Gradient Boosting Machine)

LightGBM is a gradient boosting architechture based on decision trees that enhances model efficacy while using less memory. It uses two novel techniques: Exclusive Feature Bundling(EFB), Gradient-based 1sided Sampling which overcome the shortcomings of typical Gradient Boosting Decision Tree frameworks' histogram-based approach. LightGBM separates the tree leaf by leaf, on contradiction to other boosting techniques that build trees level by level. It is the leaf with the highest delta loss that is expanded. Because the leaf is fixed, the leaf inducing algorithm has a lesser loss than the levelwise technique. Leafwise tree building may increase the model's intricacies and result in overfitting in short datasets.

For set with n instances {x1, · · ·, xn}, xi is a vector,
the negative gradients of the loss function {g1, · · ·, gn}.
split the instances according to the estimated variance gain at vector Vj (d)

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n_r^j(d)}\right)$$

Category Boosting (CatBoost)

CatBoost is the first to introduce ordered boosting. In ordered boosting, the training examples are randomly permuted, and n distinct supporting models are maintained (the ith model is trained using only the first I samples in the permutation), and the residual/error is calculated using prior model residuals at each step. For practical purposes, CatBoost employs a variation. All of the models in this version share the same tree structure (sequence of splitting features). That is, CatBoost uses the same Dk as the data for identifying the structure or fitting the decision tree ht, and the entire dataset D as the data for evaluating whether ht is the decision tree that minimizes the anticipated loss. It use many permutations to generate a number of sets of residual values that can be used to determine h, acquire Ft-1, and ensure that none of the values of are utilized to compute the gradients' values. As a result, the variance in the gradient estimates (rate of change of the loss function) is reduced, and prediction shift is avoided.

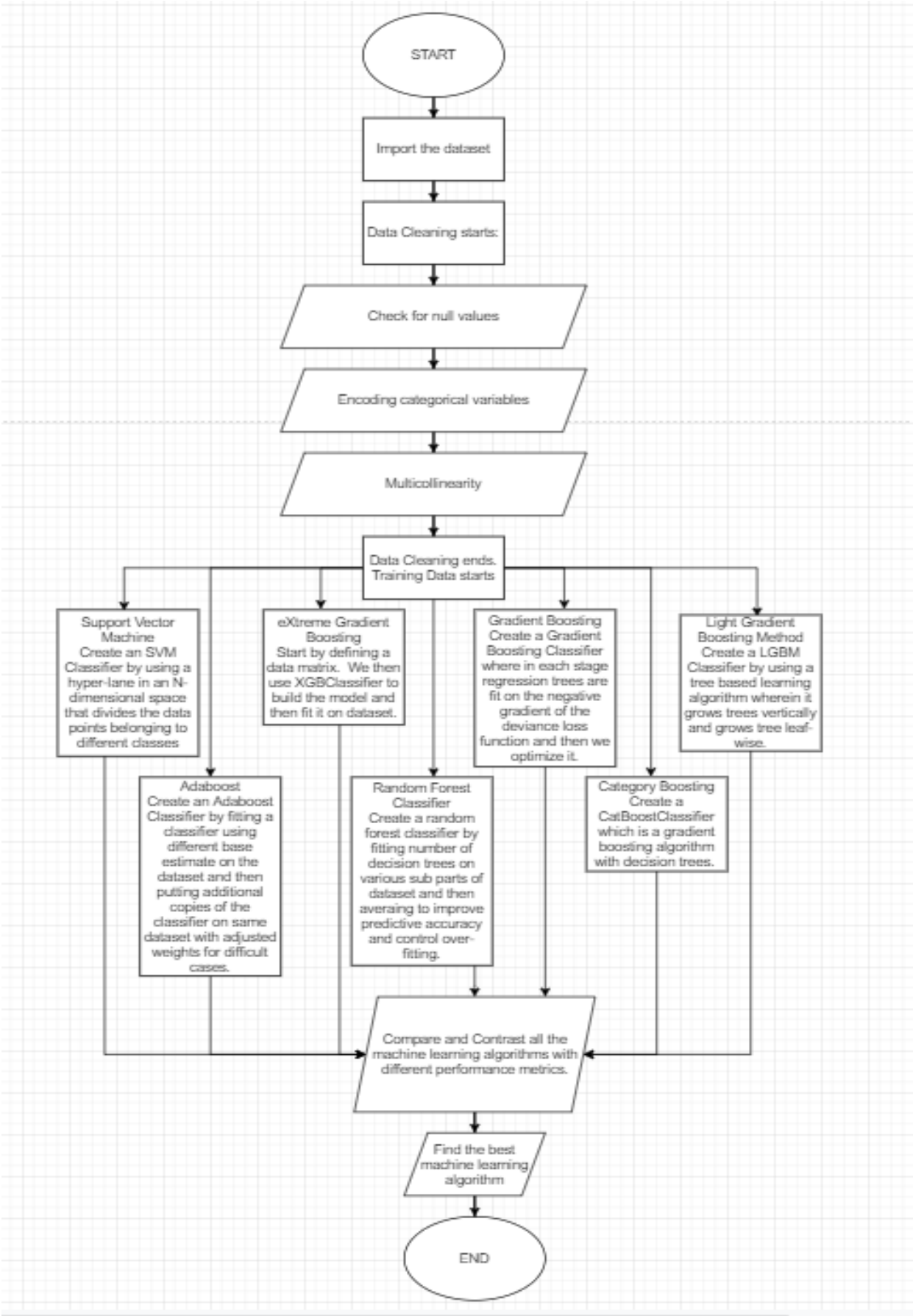
CountInClass: times the label value was equal to "1"
Prior: is the preliminary value of numerator.
TotalCount: is the total number of objects that have a categorical feature value matching the current one

$$avg\_target = \frac{countInClass + prior}{totalCount + 1}$$

Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be the permutation, then $x_{\sigma_p,k}$ is substituted with

$$\frac{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}]Y_{\sigma_j} + a \cdot P}{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}] + a},$$

Proposed System Model



4.1.1 Proposed System Model Flowchart

4.2 CODES AND STANDARDS

H/W Requirements:

Minimum 512MB Main Memory,
Processor: 64 bit Intel,
RAM: 4GB or more,
Monitor: EGA / SVGA (display), 800X600 24 bits True Color,
CPU speed: 2.6GHz,
Standard Keyboard: 106 Keys with Function Keys & Numeric Pad Separated,
Mouse: PS /2 Optical mouse and
Operating system: Windows XP SP-3 / 7/8.

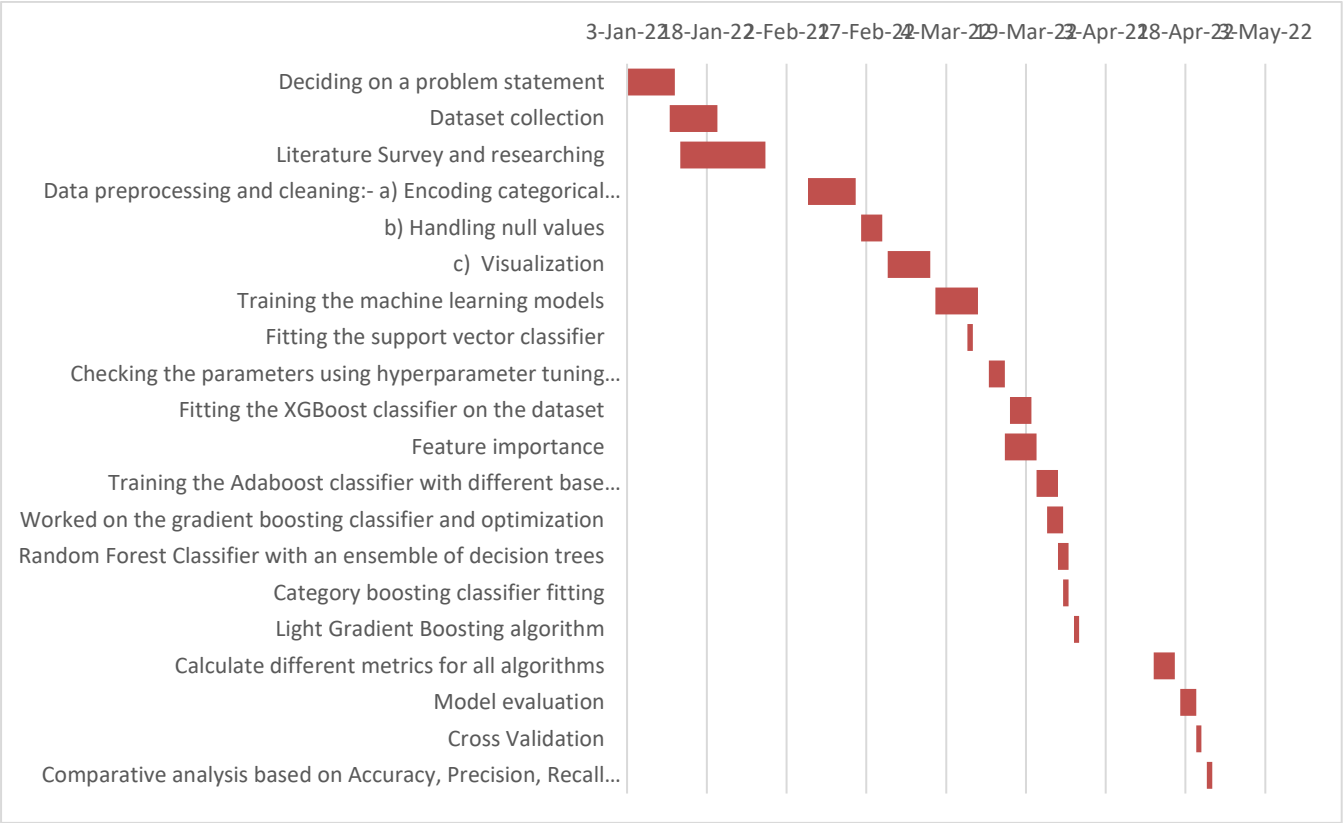
S/W Requirements:

Web Browser: Google Chrome, Internet Explorer 6.0/7.0, Android Studio and/or Mozilla Firefox,
Coding language: Python 3.8 and above,
IDE: Jupyter Notebook/ Google colab and
Microsoft office (dataset).

4.3 CONSTRAINTS, ALTERNATIVES AND TRADEOFFS

We are assuming that the attributes that are present in our dataset are the only ones that are needed to determine if the person is obese and thus we are depending on these attributes to train and test our dataset. For this model to work the domain requirements are Google Colab and Jupyter Notebook, and the user requirements are Windows 10 or above, Jupyter Notebook, Python interpretor, Google chrome(version 99 or above), 8GB RAM, 2GB VRAM.

5. SCHEDULE, TASKS AND MILESTONES



5.1. Gantt Chart depicting our schedule with our tasks and milestones for the project

6. PROJECT DEMONSTRATION

Obesity prediction using bagging and boosting machine learning models

In [1]:
```
import pandas as pd
import seaborn as sns
```

In [2]:
```
df= pd.read_csv("obesitydataset.csv")
df.head()
```

Out[2]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	M
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_Transp
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_Transp
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_Transp
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transp

6.1. Dataset Collection

ENCODING CATEGORICAL VARIABLES

In [3]:
```
import sklearn.preprocessing as preprocessing
le=preprocessing.LabelEncoder()
```

In [4]:
```
df['gender']=le.fit_transform(df['Gender'].astype(str))
```

In [5]:
```
df['famhistory']=le.fit_transform(df['family_history_with_overweight'].astype(str))
df['favc']=le.fit_transform(df['FAVC'].astype(str))
df['caec']=le.fit_transform(df['CAEC'].astype(str))
df['smoke']=le.fit_transform(df['SMOKE'].astype(str))
```

In [6]:
```
df['scc']=le.fit_transform(df['SCC'].astype(str))
df['calc']=le.fit_transform(df['CALC'].astype(str))
df['mtrans']=le.fit_transform(df['MTRANS'].astype(str))
df['nobeyes']=le.fit_transform(df['NObeyesdad'].astype(str))
```

In [7]:
```
df=df.drop(['Gender','family_history_with_overweight','FAVC','CAEC','SMOKE','SCC','CALC','MTRANS','NObeyesdad'],axis=1)
```

In [8]:
```
df
```

Out[8]:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	gender	famhistory	favc	caec	smoke	scc	calc	mtrans	nobeyes
0	21.000000	1.620000	64.000000	2.0	3.0	2.000000	0.000000	1.000000	0	1	0	2	0	0	3	3	1
1	21.000000	1.520000	56.000000	3.0	3.0	3.000000	3.000000	0.000000	0	1	0	2	1	1	2	3	1
2	23.000000	1.800000	77.000000	2.0	3.0	2.000000	2.000000	1.000000	1	1	0	2	0	0	1	3	1
3	27.000000	1.800000	87.000000	3.0	3.0	2.000000	2.000000	0.000000	1	0	0	2	0	0	1	4	5
4	22.000000	1.780000	89.800000	2.0	1.0	2.000000	0.000000	0.000000	1	0	0	2	0	0	2	3	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

6.2. Data Preprocessing: Encoding Categorical Variables

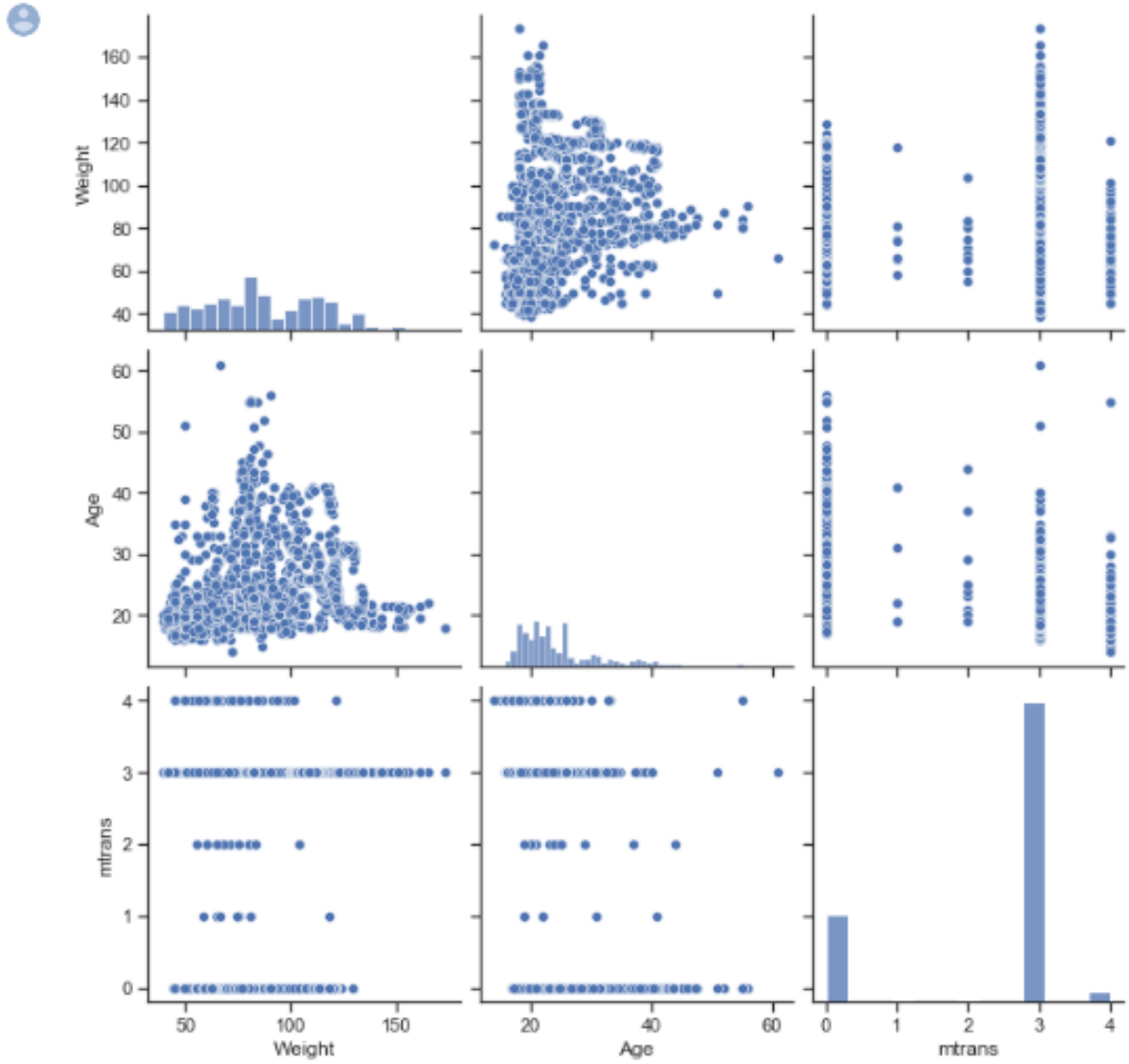
```
[ ] corr=df.corr()
```

```
corr.style.background_gradient(cmap='coolwarm')
```

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	gender	famhistory	favc	caec	smoke	scc	calc	mtrans	nobeyes
Age	1.000000	-0.025958	0.202560	0.016291	-0.043944	-0.045304	-0.144938	-0.296931	0.048394	0.205725	0.063902	0.083739	0.091987	-0.116283	-0.044487	-0.601945	0.236170
Height	-0.025958	1.000000	0.463136	-0.038121	0.243672	0.213376	0.294709	0.051912	0.618466	0.247684	0.178364	0.048818	0.055499	-0.133753	-0.129732	-0.073609	0.038986
Weight	0.202560	0.463136	1.000000	0.216125	0.107469	0.200575	-0.051436	-0.071561	0.161668	0.496820	0.272300	0.287493	0.025746	-0.201906	-0.206677	0.004610	0.387643
FCVC	0.016291	-0.038121	0.216125	1.000000	0.042216	0.068461	0.019939	-0.101135	-0.274505	0.040372	-0.027283	-0.054670	0.014320	0.071852	-0.060781	0.064743	0.018522
NCP	-0.043944	0.243672	0.107469	0.042216	1.000000	0.057088	0.129504	0.036326	0.067600	0.071370	-0.007000	-0.097801	0.007811	-0.015624	-0.071747	-0.053858	-0.092616
CH2O	-0.045304	0.213376	0.200575	0.068461	0.057088	1.000000	0.167236	0.011965	0.107930	0.147437	0.009719	0.144995	-0.031995	0.008036	-0.091386	0.044028	0.108868
FAF	-0.144938	0.294709	-0.051436	0.019939	0.129504	0.167236	1.000000	0.058562	0.189607	-0.056673	-0.107995	-0.030110	0.011216	0.074221	0.086799	0.006394	-0.129564
TUE	-0.296931	0.051912	-0.071561	-0.101135	0.036326	0.011965	0.058562	1.000000	0.017269	0.022943	0.068417	-0.048567	0.017613	-0.010928	0.045864	0.176945	-0.069448
gender	0.048394	0.618466	0.161668	-0.274505	0.067600	0.107930	0.189607	0.017269	1.000000	0.102512	0.064934	0.091543	0.044698	-0.102633	0.007616	-0.137537	0.024908
famhistory	0.205725	0.247684	0.496820	0.040372	0.071370	0.147437	-0.056673	0.022943	0.102512	1.000000	0.208036	0.169787	0.017385	-0.185422	0.036676	-0.101540	0.313667
favc	0.063902	0.178364	0.272300	-0.027283	-0.007000	0.009719	-0.107995	0.068417	0.064934	0.208036	1.000000	0.150068	-0.050660	-0.190658	-0.089520	-0.069800	0.044582
caec	0.083739	0.048818	0.287493	-0.054670	-0.097801	0.144995	-0.030110	-0.048567	0.091543	0.169787	0.150068	1.000000	-0.055282	-0.109179	-0.047540	-0.048535	0.327295
smoke	0.091987	0.055499	0.025746	0.014320	0.007811	-0.031995	0.011216	0.017613	0.044698	0.017385	-0.050660	-0.055282	1.000000	0.047731	-0.082471	-0.010702	-0.023256
scc	-0.116283	-0.133753	-0.201906	0.071852	-0.015624	0.008036	0.074221	-0.010928	-0.102633	-0.185422	-0.190658	-0.109179	0.047731	1.000000	-0.003463	0.043157	-0.050679
calc	-0.044487	-0.129732	-0.206677	-0.060781	-0.071747	-0.091386	0.086799	0.045864	0.007616	0.036676	-0.089520	-0.047540	-0.082471	-0.003463	1.000000	-0.012452	-0.134632
mtrans	-0.601945	-0.073609	0.004610	0.064743	-0.053858	0.044028	0.006394	0.176945	-0.137537	-0.101540	-0.069800	-0.048535	-0.010702	0.043157	-0.012452	1.000000	-0.046202
nobeyes	0.236170	0.038986	0.387643	0.018522	-0.092616	0.108868	-0.129564	-0.069448	0.024908	0.313667	0.044582	0.327295	-0.023256	-0.050679	-0.134632	-0.046202	1.000000

6.3. Data Preprocessing: Visualization – Correlation Table

```
sns.set(style="ticks", color_codes=True)
sns.pairplot(df,height=3,vars = ['Weight', 'Age','mtrans'])
plt.show()
```



6.4. Data Preprocessing: Visualization

SUPPORT VECTOR MACHINE

In [17]:
```
#support vector machine with radial basis function kernel
svc=SVC(C=100.0)
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
```

In [18]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.7730496453900709
```

In [19]:
```
precision_recall_fscore_support(y_test, y_pred, average='weighted')
```

Out[19]: (0.7677131618444506, 0.7730496453900709, 0.7685713590854568, None)

checking for overfitting and underfitting

In [20]:
```
svc.score(X_train, y_train)
```

Out[20]: 0.8139810426540285

In [21]:
```
svc.score(X_test, y_test)
```

Out[21]: 0.7730496453900709

In [22]:
```
#support vector machine with linear kernel
lsvc=SVC(kernel='linear',C=100.0)
lsvc.fit(X_train,y_train)
l_pred=lsvc.predict(X_test)
```

In [23]:
```
accuracy_score(y_test, l_pred)
```

Out[23]: 0.9692671394799054

6.5. Support Vector Machine Classifier

hyperparameter tuning using gridsearch

In [30]:
```
parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
               {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
               {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4] ,'gamma':[0.01,0.02,0.03,0.04,0.05]}
              ]
svc1=SVC()

```

In [31]:
```

grid_search = GridSearchCV(estimator = svc1,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)
```

In [32]:
```
grid_search.fit(X_train, y_train)
```

Out[32]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']}],
             scoring='accuracy')
```

In [36]:
```
print(grid_search.best_score_)
print(grid_search.best_params_)
print(grid_search.best_estimator_)
print(grid_search.score(X_test, y_test))
```

```
0.9537969905009394
{'C': 100, 'kernel': 'linear'}
SVC(C=100, kernel='linear')
0.9692671394799054
```

6.6. Using gridsearch to check parameters using hyperparameter tuning

eXtreme Gradient Boosting

In [37]:
```
# define data_dmatrix
data_dmatrix = xgb.DMatrix(data=X,label=y)
```

In [38]:
```
xgbc = XGBClassifier(learning_rate=0.5, n_estimators=600, objective='binary:logistic',
                     silent=True, nthread=1)
```

In [39]:
```
xgbc.fit(X_train, y_train)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_en
coder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[02:08:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.


[02:08:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explici
tly set eval_metric if you'd like to restore the old behavior.
```

Out[39]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.5, max_delta_step=0,
              max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=600, n_jobs=1, nthread=1,
              num_parallel_tree=1, objective='multi:softprob', predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None,
```

In [40]:
```
xgb_pred = xgbc.predict(X_test)
```

In [41]:
```
accuracy_score(y_test, xgb_pred)
```

Out[41]:
```
0.9763593380614657
```

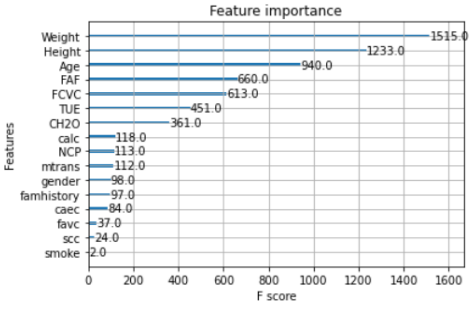
In [42]:
```
precision_recall_fscore_support(y_test, xgb_pred, average='macro')
```

Out[42]:
```
(0.9765892172936761, 0.975589322957744, 0.9757779053240883, None)
```

6.7. eXtreme Gradient Boosting

Feature Importance

In [43]:
```
xgb.plot_importance(xgbc)
plt.figure(figsize = (16, 12))
plt.show()
```




```
<Figure size 1152x864 with 0 Axes>
```

6.8. Feature Importance

ADAboost with different base estimators

Base estimator= decision tree

```
In [45]:
1 ada=AdaBoostClassifier(DecisionTreeClassifier(),learning_rate=0.1,n_estimators=200,algorithm= 'SAMME')

In [46]:
1 ada.fit(X_train,y_train)

Out[46]: AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(),
                            learning_rate=0.1, n_estimators=200)

In [47]:
1 pre=ada.predict(X_test)

In [48]:
1 accuracy_score(y_test, pre)

Out[48]: 0.9243498817966903

In [49]:
1 precision_recall_fscore_support(y_test, pre, average='macro')

Out[49]: (0.9249050660469891, 0.9224361147259071, 0.9227558966747847, None)
```

6.9. ADAboost with base estimator decision tree

base estimator= extra trees classifier

```
In [50]:
1 ada1=AdaBoostClassifier(ExtraTreesClassifier(),learning_rate=0.5,n_estimators=200,algorithm= 'SAMME.R')

In [51]:
1 ada1.fit(X_train,y_train)

Out[51]: AdaBoostClassifier(base_estimator=ExtraTreesClassifier(), learning_rate=0.5,
                            n_estimators=200)

In [52]:
1 pre1=ada1.predict(X_test)

In [53]:
1 accuracy_score(y_test, pre1)

Out[53]: 0.933806146572104

In [54]:
1 precision_recall_fscore_support(y_test, pre1, average='macro')

Out[54]: (0.9381833852422089, 0.9321595921753031, 0.932814352562044, None)
```

6.10. ADAboost with base estimator extra trees classifier

base estimator= support vector machine

```
In [55]:
1 ada2=AdaBoostClassifier(SVC(kernel='linear',C=100.0),learning_rate=0.4,n_estimators=100,algorithm= 'SAMME')
2 ada2.fit(X_train,y_train)
3 pre2=ada2.predict(X_test)
4 accuracy_score(y_test, pre2)

Out[55]: 0.8226950354609929

In [56]:
1 precision_recall_fscore_support(y_test, pre2, average='macro')

Out[56]: (0.8255784067082088, 0.8209615992805309, 0.8212744451019959, None)
```

6.11. ADAboost with base estimator support vector machine

base estimator= random forest

```
In [57]:
1 ada3=AdaBoostClassifier(RandomForestClassifier(),learning_rate=0.1,n_estimators=200,algorithm= 'SAMME.R')

In [58]:
1 ada3.fit(X_train,y_train)
2 pre3=ada3.predict(X_test)
3 accuracy_score(y_test, pre3)

Out[58]: 0.9527186761229315

In [59]:
1 precision_recall_fscore_support(y_test, pre3, average='macro')

Out[59]: (0.9553444566232291, 0.9512373736921307, 0.9514757258662894, None)
```

6.12. ADAboost with base estimator random forest

Hyperparameter tuning with adaboost

```
In [62]:
bdt_ada = GridSearchCV(ada,{'learning_rate':[0.1,0.4,0.5,0.8],
                            'n_estimators':[50,60,80,100,200,300,400],
                            'algorithm': ['SAMME', 'SAMME.R']})
```

```
In [63]:
bdt_ada.fit(X_train,y_train)
print(bdt_ada.best_score_)
print(bdt_ada.best_params_)
```

```
0.9242076800168559
{'algorithm': 'SAMME.R', 'learning_rate': 0.8, 'n_estimators': 100}
```

6.13. Hyperparameter tuning with adaboost

Gradient boosting

```
In [64]:
gb= GradientBoostingClassifier(n_estimators=200, learning_rate = 0.3, max_features=2, max_depth = 2, random_state =
```

```
In [65]:
gb.fit(X_train,y_train)
```

```
Out[65]: GradientBoostingClassifier(learning_rate=0.3, max_depth=2, max_features=2,
                                    n_estimators=200, random_state=0)
```

```
In [66]:
gb_pre=gb.predict(X_test)
```

```
In [67]:
accuracy_score(y_test, gb_pre)
```

```
Out[67]: 0.9598108747044918
```

```
In [68]:
precision_recall_fscore_support(y_test, gb_pre, average='macro')
```

```
Out[68]: (0.9588114956127379, 0.9583283958774924, 0.9583539706996208, None)
```

```
In [69]:
from sklearn import metrics
```

```
In [70]:
print(metrics.classification_report(y_test, gb_pre, digits=3))
```

```
              precision    recall  f1-score   support

           0      1.000     0.954     0.976        65
           1      0.900     0.947     0.923        57
           2      0.957     0.985     0.971        67
           3      1.000     1.000     1.000        53
           4      1.000     1.000     1.000        69
           5      0.911     0.895     0.903        57
           6      0.944     0.927     0.936        55

    accuracy                          0.960       423
   macro avg      0.959     0.958     0.958       423
weighted avg      0.960     0.960     0.960       423
```

6.14. Gradient Boosting and its optimization

Random forest

```
In [71]:
rfc=RandomForestClassifier()
rfc.fit(X_train,y_train)
rf_pred=rfc.predict(X_test)
accuracy_score(y_test,rf_pred)
```

```
Out[71]: 0.9527186761229315
```

```
In [72]:
precision_recall_fscore_support(y_test, rf_pred, average='macro')
```

```
Out[72]: (0.9593326986761973, 0.9513353888844855, 0.9520514626104063, None)
```

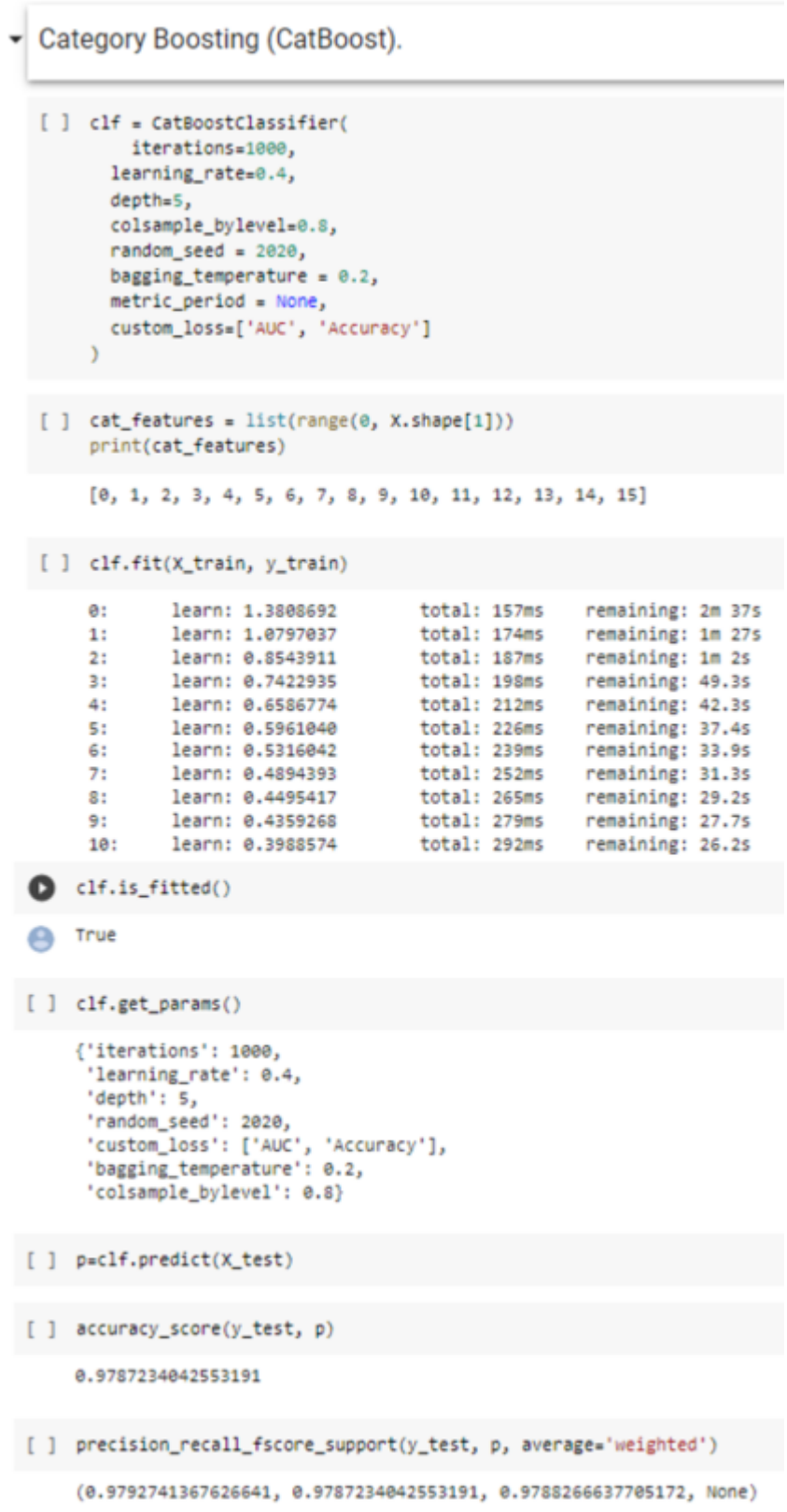
```
In [74]:
print(metrics.classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        65
           1       0.77      1.00      0.87        57
           2       0.99      0.99      0.99        67
           3       1.00      1.00      1.00        53
           4       1.00      1.00      1.00        69
           5       0.98      0.84      0.91        57
           6       0.98      0.91      0.94        55

    accuracy                           0.95       423
   macro avg       0.96      0.95      0.95       423
weighted avg       0.96      0.95      0.95       423
```

6.15. Random Forest



6.16. Category Boosting Classifier fitting

Light Gradient Boosting Method (LightGBM)

In [85]:
```
lgm= LGBMClassifier(objective='regression',num_leaves=31,
                                learning_rate=0.6, n_estimators=500,
                                max_bin = 100, bagging_fraction = 0.4)
```

In [87]:
```
lgm.fit(X_train, y_train)
```

```
[LightGBM] [Warning] bagging_fraction is set=0.4, subsample=1.0 will be ignored. Current value: bagging_fraction=0.4
```

Out[87]:
```
LGBMClassifier(bagging_fraction=0.4, learning_rate=0.6, max_bin=100,
               n_estimators=500, objective='regression')
```

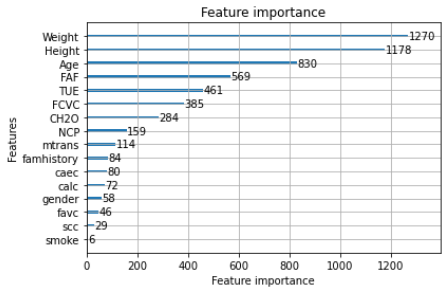
In [88]:
```
n=lgm.predict(X_test)
accuracy_score(y_test, n)
```

Out[88]:
```
0.9905437352245863
```

In [89]:
```
precision_recall_fscore_support(y_test, n, average='macro')
```

Out[89]:
```
(0.9904685037856951, 0.990900327742433, 0.9906096059113301, None)
```

In [90]:
```
ax = lgb.plot_importance(lgm, max_num_features=20)
plt.show()
```



6.17. Light Gradient Boosting Algorithm

7. RESULT & DISCUSSION

The goal of this paper is to determine obesity prediction and improve accuracy by using different machine learning models that aren't already available for our dataset to find one that has the highest accuracy rate and precision score, so it can be used to deploy a efficient machine learning algorithm and for future work.

ML MODELS	accuracy	precision	recall	f1
SVM with radial basis kernel	77.3	76.7	77.3	76.8
SVM with linear kernel	96.9	96.9	96.7	96.8
eXtreme Gradient Boosting	97.6	97.6	97.55	97.57
Adaboost with Decision Tree	93.1	92.9	92.8	92.8
Adaboost with Random Forest	96.2	96.4	96.1	96.1
Random Forest Classifier	95.5	95.6	95.3	95.3
Gradient Boosting	95.9	95.8	95.83	95.83
Adaboost with Extra Tree classifier	93.6	93.6	93.3	93.4
Adaboost with SVM	82.2	82.5	82.09	82.1
Category Boosting (CatBoost)	97.8	97.9	97.8	97.8
Light Gradient Boosting Method	99.05	99.04	99.09	99.06

7.1. Comparative analysis of ML Models with Performance Metrics

From our comparative analysis we see that the Light Gradient Boosting Method which gives an accuracy of 99.09% is the best ML algorithm for this prediction based model.

8. SUMMARY

Obesity has now evolved into a global condition that affects people of all ages and genders all over the world. Obesity is defined as a complex condition in which an excessive or abnormal amount of fat is present in our bodies. The best criterion for determining obesity is the BMI (BMI). This is because the BMI is evaluated using a person's height and weight of the body. Because BMI measures body weight in relation to height, there is a strong link between BMI and aggregate body fat in adults. Obesity is linked to a number of physical, emotional and mental health issues, including heart and circulatory illness, cardio metabolic risk, type 2 diabetes, depression, and hypertension. This work hopes to predict obesity in human being by utilizing data from the 'Estimation of Obesity Levels based on Eating Habits and Physical Condition dataset' from the UC,IRVINE Machine Learning database. It is critical to test alternative ways in public health, and we believe that applying Machine Learning methods will aid in improving forecasts and discovering good data structures within the current data, as well as assisting in the design of new policies. Support Vector Machine(SVM) with Radial Basis Kernel, XGBoost (eXtreme Gradient Boosting), Adaboost with decision tree, Adaboost with Random Forest, Adaboost with SVM, Random Forest Classifier, Category Boosting, Support Vector Machine(SVM) with Linear Kernel, Light Gradient Boosting Method, Gradient Boosting, and Adaboost with Extra Tree Classifier trained on the data are some of the machine learning methods we have implemented. The machine learning approach with the highest prediction rate for the dataset is then evaluated using performance measures including accuracy, recall, precision score, F1 score, and many more. **Based on our comparison, the Light Gradient Boosting Method, which has a 99.09% accuracy, is the most efficient ML algorithm for this predictive model.**

9. References

[1] & Sánchez Hernández, A. B. (2019). Obesity level estimation software based on decision trees.

[2] Cervantes, R. C., & Palacio, U. M. (2020). Estimation of obesity levels based on computational intelligence. *Informatics in Medicine Unlocked*, *21*, 100472.

[3] Abdullah, F. S., Manan, N. S. A., Ahmad, A., Wafa, S. W., Shahril, M. R., Zulaily, N., ... & Ahmed, A. (2016, August). Data mining techniques for classification of childhood obesity among year 6 school children. In *International Conference on Soft Computing and Data Mining* (pp. 465-474). Springer, Cham.

[4] Ferdowsy, F., Rahi, K. S. A., Jabiullah, M. I., & Habib, M. T. (2021). A machine learning approach for obesity risk prediction. *Current Research in Behavioral Sciences*, *2*, 100053.

[5] Gupta, M., Phan, T. L. T., Bunnell, T., & Beheshti, R. (2019). Obesity Prediction with EHR Data: A deep learning approach with interpretable elements. *arXiv preprint arXiv:1912.02655*.

[6] Dugan, T. M., Mukhopadhyay, S., Carroll, A., & Downs, S. (2015). Machine learning techniques for prediction of early childhood obesity. *Applied clinical informatics*, *6*(03), 506-520.

[7] Thamrin, S. A., Arsyad, D. S., Kuswanto, H., Lawi, A., & Nasir, S. (2021). Predicting Obesity in Adults Using Machine Learning Techniques: An Analysis of Indonesian Basic Health Research 2018. *Frontiers in nutrition*, *8*.

[8] Pang, X., Forrest, C. B., Le-Scherban, F., & Masino, A. J. (2021). Prediction of early childhood obesity with machine learning and electronic health record data. *International Journal of Medical Informatics*, *150*, 104454.

[9] Dunstan, J., Aguirre, M., Bastías, M., Nau, C., Glass, T. A., & Tobar, F. (2020). Predicting nationwide obesity from food sales using machine learning. Health informatics journal, 26(1), 652-663.

[10] Jindal, K., Baliyan, N., & Rana, P. S. (2018). Obesity prediction using ensemble machine learning approaches. In Recent Findings in Intelligent Computing Techniques (pp. 355-362). Springer, Singapore.

[11] Cheng, X., Lin, S. Y., Liu, J., Liu, S., Zhang, J., Nie, P., ... & Xue, H. (2021). Does Physical Activity Predict Obesity—A Machine Learning and Statistical Method-Based Analysis. International Journal of Environmental Research and Public Health, 18(8), 3966.

[12] https://www.mayoclinic.org/diseases-conditions/obesity/symptoms-causes/syc-20375742#:~:text=Obesity%20is%20a%20complex%20disease,blood%20pressure%20and%20certain%20cancers

[13] https://www.medicinenet.com/obesity_weight_loss/article.htm

[14] https://ourworldindata.org/obesity

[15] Palechor, F. M., & de la Hoz Manotas, A. (2019). Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico. Data in brief, 25, 104344.

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

10. APPENDIX A

CODE

Obesity prediction using bagging and boosting machine learning models

```
[ ] import pandas as pd
    import seaborn as sns
    df= pd.read_csv("obesitydataset.csv")
    df.head()
```

1)CHECK NULL VALUES

```
[ ] df.isnull().sum()
    df.dtypes
    sns.FacetGrid(df,size=5).map(sns.distplot,"Height").add_legend()
    sns.FacetGrid(df,size=5).map(sns.distplot,"Age").add_legend()
    sns.FacetGrid(df,size=5).map(sns.distplot,"Weight").add_legend()
```

ENCODING CATEGORICAL VARIABLES

```
[ ] import sklearn.preprocessing as preprocessing
    le=preprocessing.LabelEncoder()
    df['gender']=le.fit_transform(df['Gender'].astype(str))
    df['famhistory']=le.fit_transform(df['family_history_with_overweight'].astype(str))
    df['favc']=le.fit_transform(df['FAVC'].astype(str))
    df['caec']=le.fit_transform(df['CAEC'].astype(str))
    df['smoke']=le.fit_transform(df['SMOKE'].astype(str))
    df['scc']=le.fit_transform(df['SCC'].astype(str))
    df['calc']=le.fit_transform(df['CALC'].astype(str))
    df['mtrans']=le.fit_transform(df['MTRANS'].astype(str))
    df['nobeyes']=le.fit_transform(df['NObeyesdad'].astype(str))
    df=df.drop(['Gender','family_history_with_overweight','FAVC','CAEC','SMOKE','SCC','CALC','MTRANS','NObeyesdad'],axis=1)
    df
    df.to_csv('obesitydatasetpreprocessed.csv')
    corr=df.corr()
    corr.style.background_gradient(cmap='coolwarm')
    sns.set(style="ticks", color_codes=True)
    sns.pairplot(df,height=3,vars = ['Weight', 'Age','mtrans'])
    plt.show()
```

```
import pandas as pd
import numpy as np
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from datetime import datetime
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import StratifiedKFold, cross_val_score
from matplotlib import pyplot as plt
from sklearn.base import clone



from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
import xgboost as xgb
from xgboost import XGBClassifier,plot_importance
from sklearn.ensemble import (RandomForestClassifier,
                              AdaBoostClassifier,
                              GradientBoostingClassifier)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from lightgbm import LGBMRegressor
import lightgbm as lgb
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import precision_recall_fscore_support
y=df.nobeyes
X=df.drop(['nobeyes'],axis='columns')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
X_train.shape, X_test.shape
```

SUPPORT VECTOR MACHINE

```
[ ] #support vector machine with radial basis function kernel
    svc=SVC(C=100.0)
    svc.fit(X_train,y_train)
    y_pred=svc.predict(X_test)
    print(accuracy_score(y_test, y_pred))
    precision_recall_fscore_support(y_test, y_pred, average='weighted')
```

checking for overfitting and underfitting

```
[ ] svc.score(X_train, y_train)
    svc.score(X_test, y_test)
    #support vector machine with linear kernel
    lsvc=SVC(kernel='linear',C=100.0)
    lsvc.fit(X_train,y_train)
    l_pred=lsvc.predict(X_test)
    accuracy_score(y_test, l_pred)
    precision_recall_fscore_support(y_test, l_pred, average='macro')
    lsvc.score(X_train, y_train)
    lsvc.score(X_test, y_test)
```

hyperparameter tuning using gridsearch

```
[ ] parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
                   {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
                   {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4] ,'gamma':[0.01,0.02,0.03,0.04,0.05]}
                  ]
    svc1=SVC()
    grid_search = GridSearchCV(estimator = svc1,
                               param_grid = parameters,
                               scoring = 'accuracy',
                               cv = 5,
                               verbose=0)
    grid_search.fit(X_train, y_train)
    print(grid_search.best_score_)
    print(grid_search.best_params_)
    print(grid_search.best_estimator_)
    print(grid_search.score(X_test, y_test))
```

eXtreme Gradient Boosting

```
[ ] # define data_dmatrix
    data_dmatrix = xgb.DMatrix(data=X,label=y)
    xgbc = XGBClassifier(learning_rate=0.5, n_estimators=600, objective='binary:logistic',
                         silent=True, nthread=1)
    xgbc.fit(X_train, y_train)
    xgb_pred = xgbc.predict(X_test)
    accuracy_score(y_test, xgb_pred)
    precision_recall_fscore_support(y_test, xgb_pred, average='macro')
```

Feature Importance

```
[ ] xgb.plot_importance(xgbc)
    plt.figure(figsize = (16, 12))
    plt.show()
    list(xgbc.get_booster().get_fscore().items())
```

ADAboost with different base estimators

Base estimator= decision tree

```
[ ] ada=AdaBoostClassifier(DecisionTreeClassifier(),learning_rate=0.1,n_estimators=200,algorithm= 'SAMME')
    ada.fit(X_train,y_train)
    pre=ada.predict(X_test)
    accuracy_score(y_test, pre)
    precision_recall_fscore_support(y_test, pre, average='macro')
```

base estimator= extra trees classifier

```
[ ] ada1=AdaBoostClassifier(ExtraTreesClassifier(),learning_rate=0.5,n_estimators=200,algorithm= 'SAMME.R')
    ada1.fit(X_train,y_train)
    pre1=ada1.predict(X_test)
    accuracy_score(y_test, pre1)
    precision_recall_fscore_support(y_test, pre1, average='macro')
```

base estimator= support vector machine

```
[ ] ada2=AdaBoostClassifier(SVC(kernel='linear',C=100.0),learning_rate=0.4,n_estimators=100,algorithm= 'SAMME')
    ada2.fit(X_train,y_train)
    pre2=ada2.predict(X_test)
    accuracy_score(y_test, pre2)
    precision_recall_fscore_support(y_test, pre2, average='macro')
```

base estimator= random forest

```
[ ] ada3=AdaBoostClassifier(RandomForestClassifier(),learning_rate=0.1,n_estimators=200,algorithm= 'SAMME.R')
    ada3.fit(X_train,y_train)
    pre3=ada3.predict(X_test)
    accuracy_score(y_test, pre3)
    precision_recall_fscore_support(y_test, pre3, average='macro')
```

Hyperparameter tuning with adaboost

```
[ ] bdt_ada = GridSearchCV(ada,{'learning_rate':[0.1,0.4,0.5,0.8],
                                'n_estimators':[50,60,80,100,200,300,400],
                                'algorithm': ['SAMME', 'SAMME.R']})
    bdt_ada.fit(X_train,y_train)
    print(bdt_ada.best_score_)
    print(bdt_ada.best_params_)
```

Gradient boosting

```
[ ] gb= GradientBoostingClassifier(n_estimators=200, learning_rate = 0.3, max_features=2, max_depth = 2, random_state = 0)
    gb.fit(X_train,y_train)
    gb_pre=gb.predict(X_test)
    accuracy_score(y_test, gb_pre)
    precision_recall_fscore_support(y_test, gb_pre, average='macro')
    from sklearn import metrics
    print(metrics.classification_report(y_test, gb_pre, digits=3))
```

Random forest

```
rfc=RandomForestClassifier()
rfc.fit(X_train,y_train)
rf_pred=rfc.predict(X_test)
accuracy_score(y_test,rf_pred)
precision_recall_fscore_support(y_test, rf_pred, average='macro')
print(metrics.classification_report(y_test, rf_pred))
```

Category Boosting (CatBoost).

```
clf = CatBoostClassifier(
    iterations=1000,
  learning_rate=0.4,
  depth=5,
  colsample_bylevel=0.8,
  random_seed = 2020,
  bagging_temperature = 0.2,
  metric_period = None,
  custom_loss=['AUC', 'Accuracy']
)
cat_features = list(range(0, X.shape[1]))
print(cat_features)
clf.fit(X_train, y_train)
clf.is_fitted()
clf.get_params()
p=clf.predict(X_test)
accuracy_score(y_test, p)
precision_recall_fscore_support(y_test, p, average='weighted')
```

Light Gradient Boosting Method (LightGBM)

```
[ ] lgm= LGBMClassifier(objective='regression',num_leaves=31,
                                learning_rate=0.6, n_estimators=500,
                                max_bin = 100, bagging_fraction = 0.4)
    lgm.fit(X_train, y_train)
    n=lgm.predict(X_test)
    accuracy_score(y_test, n)
    precision_recall_fscore_support(y_test, n, average='macro')
    ax = lgb.plot_importance(lgm, max_num_features=20)
    plt.show()
```