
Project report on

# Building a predictive model which can predict the size of cloth for a customer

Submitted in partial fulfillment of the requirement for the award of

Internship completion letter

in

## Data Science, Machine Learning and AI using Python

*Submitted By*

**Kiran Suvas Patil**

B.E. in Electronics and Communication Engineering

KLS Gogte Institute of Technology, Belgaum

*Under the guidance of*

**Mr. Bandenawaz Bagwan**

# ACKNOWLEDGEMENT

The internship opportunity I had with Diginique TechLabs was a significant source for learning and professional development. Therefore, I consider myself a fortunate individual, as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me through this internship period.

I am highly indebted to Mr Bandenawaz Bagwan for his guidance and constant supervision, as well as for providing necessary information regarding the project that was extremely valuable for my study both theoretically and practically. It was only with his backing and support that I could complete the report. He provided me with all sorts of help and corrected me if ever seemed to make mistakes. I have no such words to express my gratitude.

Finally, I would like to thank my parents and all those who have helped me directly or indirectly for the successful completion of the project.

I perceive this opportunity as a significant milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work to attain my desired career objectives.

Sincerely,

Kiran Suvas Patil

## ABSTRACT:

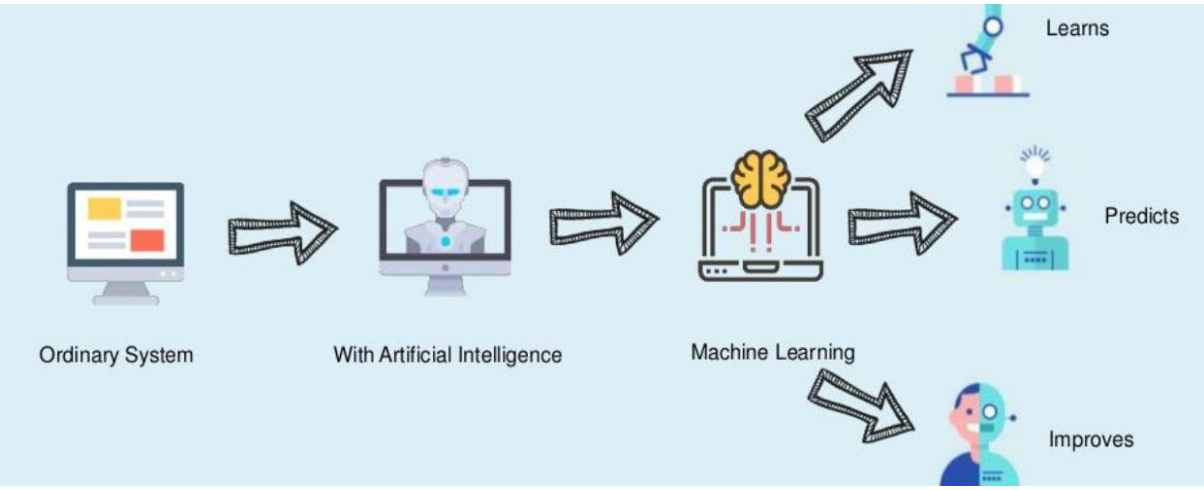
Online shopping provides a great convenience to customers. However, it also brings a few problems, one of which, especially in the fashion department, is the size fitting issue because of no immediate try-out option present. Without trying out, deciding the size of the product is one of the issues that affects customers' shopping experience, which also leads to high product return rates for businesses. According to a 2019 global study by the post-purchase vendor Narvar Inc., the size fitting issue is the most common reason for the product return, topping the list at a staggering rate of 46%, followed by the returns because of product damage which was at 15%. Therefore, it's obvious that products' size fitness prediction and size recommendation are thus essential for online shopping. In this paper, I studied the size prediction problem and proposed a method to improve the prediction accuracy. The size prediction model is carried out using several algorithms available to us using the scikit-learn machine learning library for the Python programming language. The models are built over a data set comprising four parameters, i.e. weight, height, age and size, with size being the output parameter and the remaining three being the output ones. The experiments on the real online clothing retailer dataset validate the effectiveness of the proposed method.

Online shopping is becoming popular at an astonishing pace, for its convenience of shopping anywhere we like without going to the hypostatic store. However, compared to offline shopping, which enables people to choose suitable clothes by trying them on, online shopping only allows customers to select clothes by browsing pictures, descriptions and comparing size charts. Apart from the lack of try-out options, different size standards across various brands have also become a hiccup for customers when choosing the right size. Generally, different brands of clothing may have different size catalogue mapping standards. For example, the M size T-shirt of brand A, may have the same length as the L size T-shirt of brand B. In addition, even clothing of the same brand has different size standards, for they may come from various product lines. These facts lead to a poor shopping experience when people often choose the wrong size.

# INTRODUCTION:

## Machine Learning –

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.




## How does Machine Learning Work?


With an exponential increase in data, there is a need for having a system that can handle this massive load of data. Machine Learning models like Deep Learning allow the vast majority of data to be handled with an accurate generation of predictions.

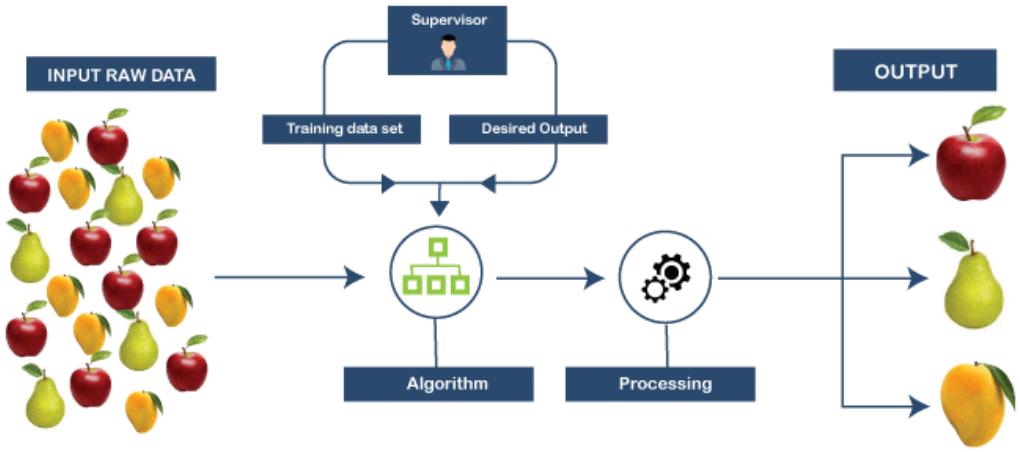
## Types of Machine Learning –

Machine Learning Algorithms can be classified into 3 types as follows –

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

## Supervised Learning –

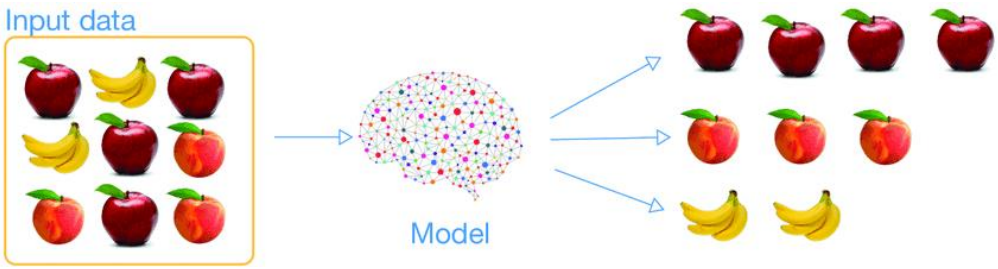
Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as *training data*, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a *supervisory signal*. In the mathematical model, each training example is represented by an array or vector, sometimes called a *feature vector*, and the training data is represented by a matrix.




Types of supervised learning algorithms include *active learning, classification* and *regression*. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range.

## Unsupervised Learning –

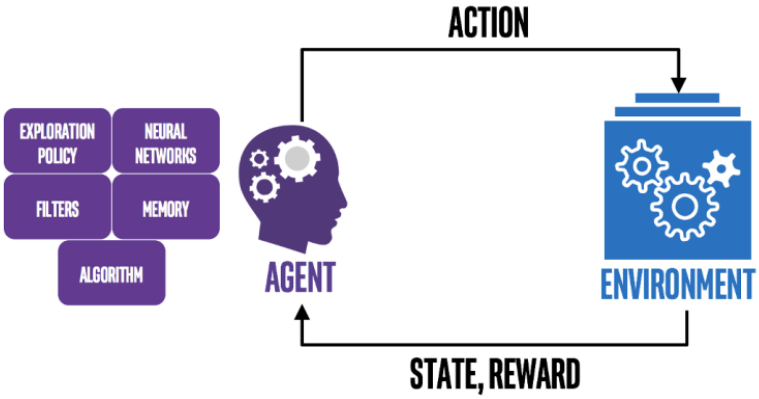
Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data.




Unsupervised Learning algorithms identify the data based on their densities, structures, similar segments, and other similar features. Unsupervised Learning Algorithms are based on *Hebbian Learning*. *Cluster* analysis is one of the most widely used techniques in supervised learning.

## Reinforcement Learning –

Reinforcement learning covers more area of Artificial Intelligence which allows machines to interact with their dynamic environment in order to reach their goals. With this, machines and software agents are able to evaluate the ideal behavior in a specific context. In the case of reinforcement learning, there is no answer key provided to the agent when they have to perform a particular task. When there is no training dataset, it learns from its own experience.




## Machine Learning Algorithms –

- ### Linear Regression:

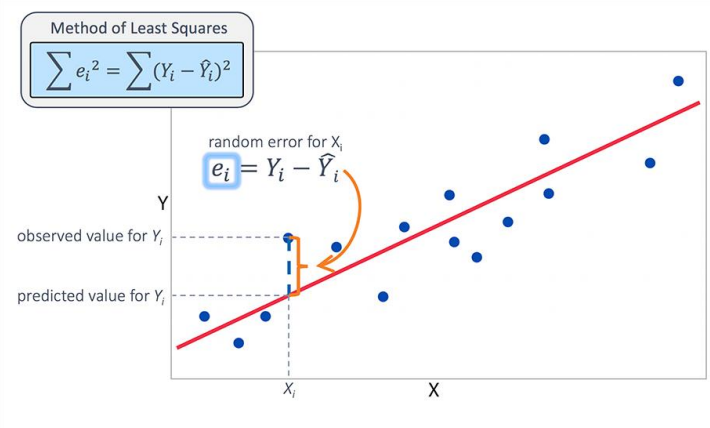
Regression analysis is a statistical technique for determining the relationship between two or more than two variables. There are two types of variables in regression analysis – *independent variable* and *dependent variable*. Independent variables are also known as predictor variables. These are the variables that do not change. On the other side, the variables whose values change are known as dependent variables. These variables depend on the independent variables. Dependent variables are also known as response variables.
The methodology for measuring the relationship between the two continuous variables is known as *Linear Regression*.

Independent Variable – "x"
Dependent Variable – "y"

The relationship between x and y is described as follows –

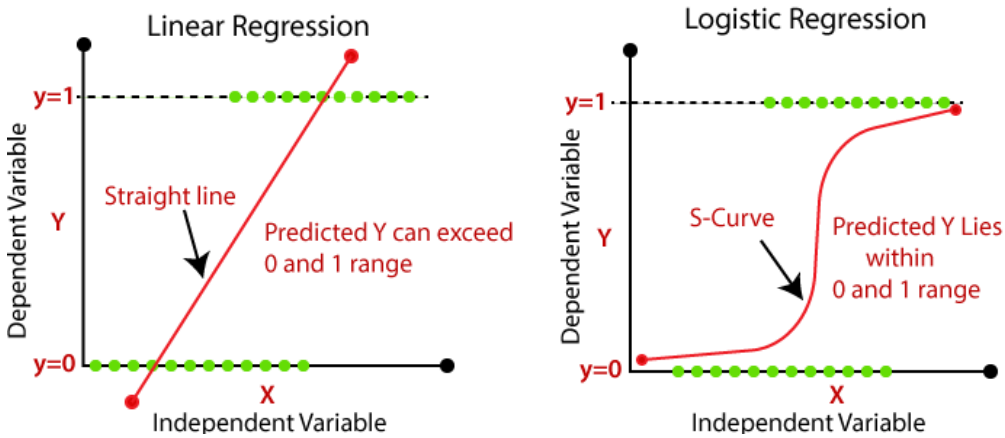
$y = mx + c$

Here, m is the slope and c is the intercept.



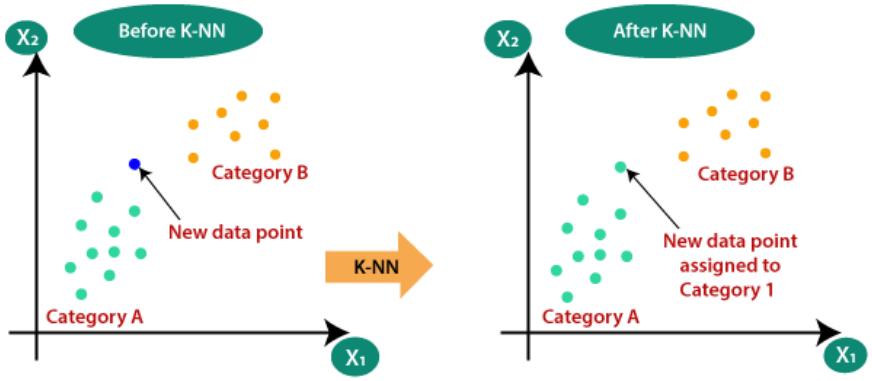
- Logistic Regression:

This is the most popular ML algorithm for binary classification of the data-points. With the help of logistic regression, we obtain a categorical classification that results in the output belonging to one of the two classes.
Logistic Regression has two components – *Hypothesis* and *Sigmoid Curve*. Based on this hypothesis, one can derive the resultant likelihood of the event. Data obtained from the hypothesis is then fit into the log function that forms the S-shaped curve called 'sigmoid'. Through this log function, one can determine the category to which the output data belongs to.




- K-Nearest Neighbor(KNN):

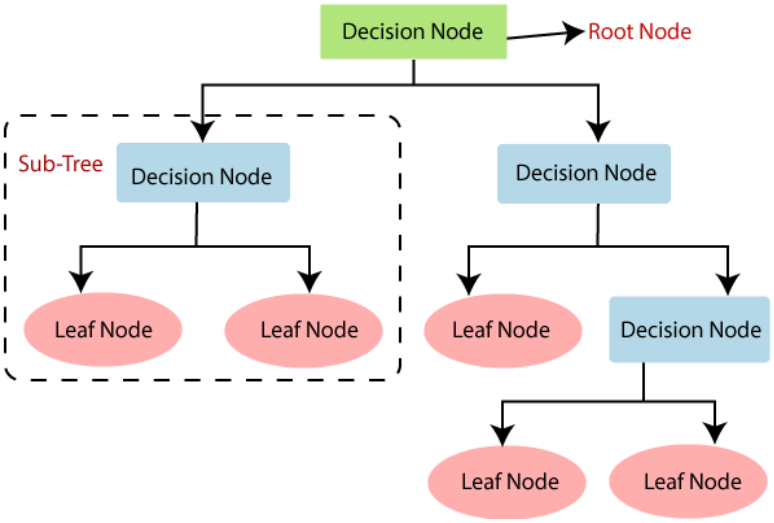
K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
K-NN algorithm stores 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. 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.

- Decision Tree :

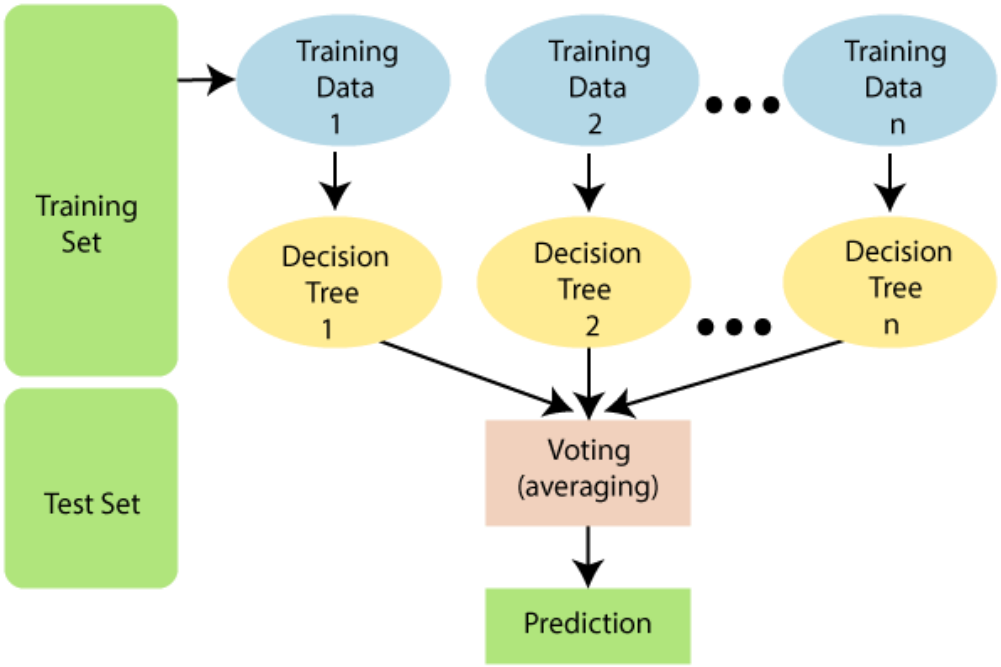
Decision tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a *tree-structured classifier*, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
In a Decision tree, there are two nodes, which are the *Decision Node* and *Leaf Node*. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.




In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

- Random Forest:

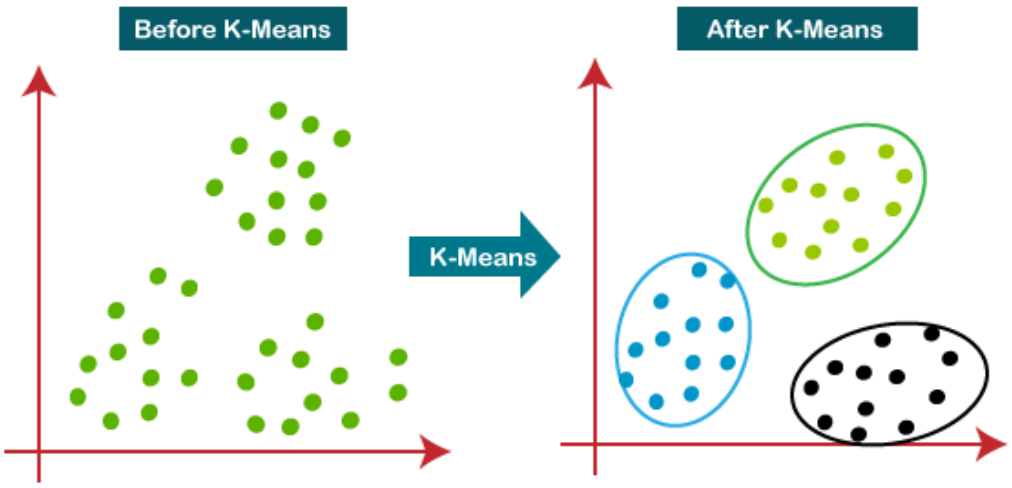
Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of *ensemble learning*, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "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.
The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

- K-Means Clustering –

K-Means is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters. The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.



## Scikit-learn –

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

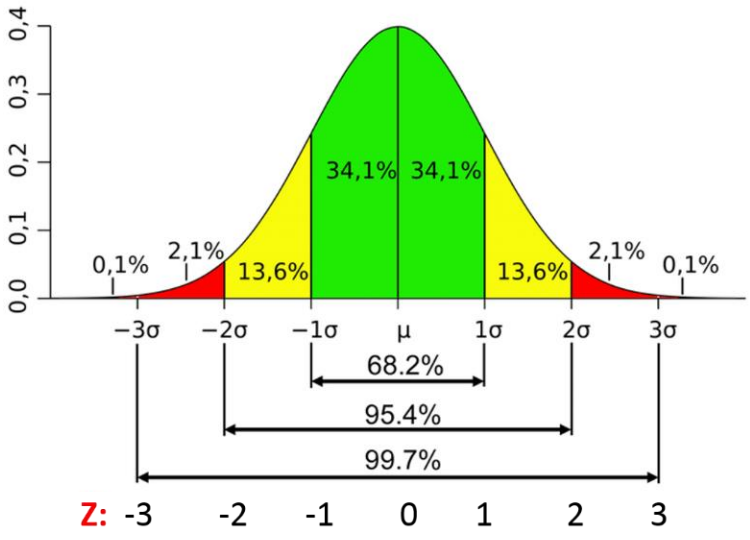
Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Scikit-learn integrates well with many other Python libraries, such as Matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.

## Outliers –

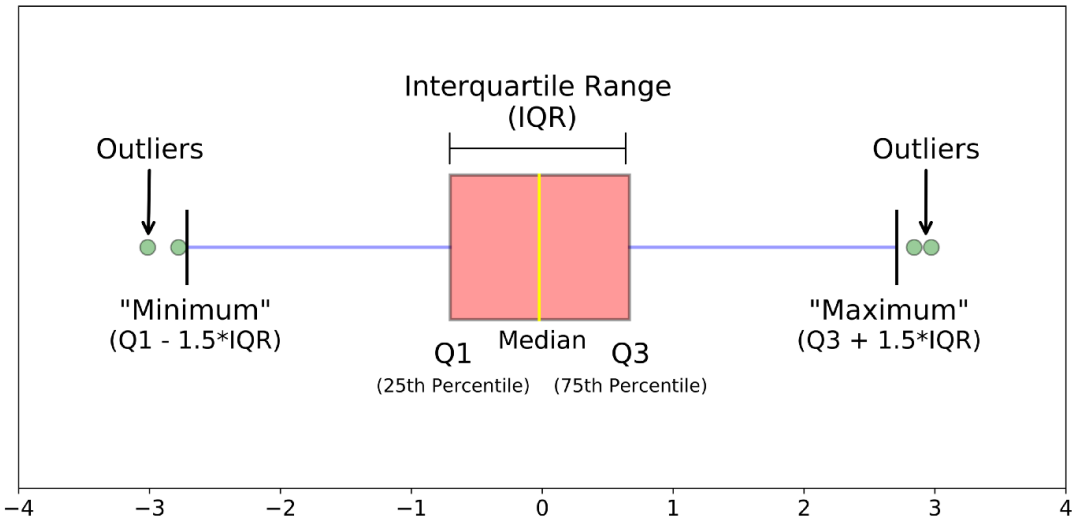
In statistics, an outlier is a data point that differs significantly from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. An outlier can cause serious problems in statistical analyses.

- How to detect outliers?
  For *Normal distributions*: Use empirical relations of Normal distribution.



The data points which fall below mean-3*(sigma) or above mean+3*(sigma) are outliers. Where mean and sigma are the average value and standard deviation of a particular column.

For *Skewed distributions*: Use Inter-Quartile Range (IQR) proximity rule.




The data points which fall below Q1 – 1.5 IQR or above Q3 + 1.5 IQR are outliers. Where Q1 and Q3 are the 25th and 75th percentile of the dataset respectively, and IQR represents the inter-quartile range and given by Q3 – Q1.

- Understanding Boxplots –

A boxplot is a standardized way of displaying the distribution of data based on a five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum"). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

Median (Q2/50th Percentile): the middle value of the dataset.

First quartile (Q1/25th Percentile): the middle number between the smallest number (not the "minimum") and the median of the dataset.

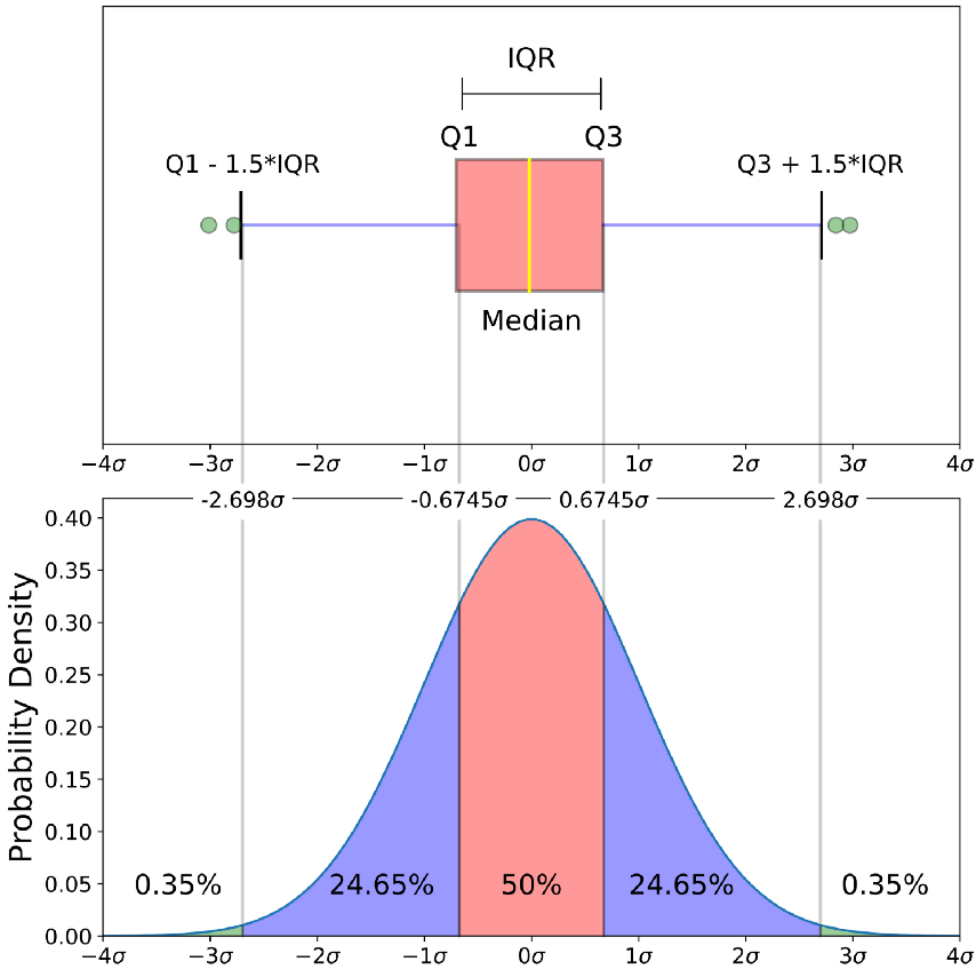
Third quartile (Q3/75th Percentile): the middle value between the median and the highest value (not the "maximum") of the dataset.

Interquartile range (IQR): 25th to the 75th percentile.

Whiskers (shown in blue) Outliers (shown as green circles)

Maximum: Q3 + 1.5*IQR Minimum: Q1 -1.5*IQR

Boxplot on a Normal Distribution




The image above is a comparison of a boxplot of a nearly normal distribution and the probability density function (pdf) for a normal distribution.
A PDF is used to specify the probability of the random variable falling within a particular range of values, as opposed to taking on any one value. This probability is given by the integral of this variable's PDF over that range — that is, it is given by the area under the density function but above the horizontal axis and between the lowest and greatest values of the range.

- Handling Outliers:

Using Z-score (standard score) –
Z-score is a statistical measure that tells you how far is a data point from the rest of the dataset. In a more technical term, Z-score tells how many standard deviations away a given observation is from the mean.
Z-score is a parametric measure and it takes two parameters — mean and standard deviation. Once you calculate these two parameters, finding the Z-score of a data point is easy.

$$Z\text{-}score = \frac{x - mean}{Standard\ Deviation}$$

Note that mean and standard deviation are calculated for the whole dataset, whereas x represents every single data point. That means, every data point will have its own z-score, whereas mean/standard deviation remains the same everywhere.
If the z score of a data point is more than 3, it indicates that the data point is quite different from the other data points. Such a data point can be an outlier.
In large production datasets, Z-score works best if data are normally distributed (aka. Gaussian distribution). Finally, Z-score is sensitive to extreme values, because the mean itself is sensitive to extreme values.

- Outliers good or bad?

Removing outliers generally helps us improve our model accuracy score significantly, but does this help during prediction? After removing outliers both the training and testing set doesn't contain them which results in a better accuracy score, but this might affect the result during prediction, if our input given falls to be an outlier itself.

Drop an outlier if:
  - You know that it's completely wrong
  - You have a lot of data in hand
  - You have an option to going back

Don't drop an outlier if:
  - Your results are critical
  - There are a lot of outliers

# METHODOLOGY:

## 1. Importing required libraries –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter("ignore")
import pickle
```

- NumPy, an acronym for Numerical Python, is a package to perform scientific computing in Python efficiently. It can be used as an efficient multi-dimensional container of generic data. This allows NumPy to integrate with a wide variety of databases seamlessly.
- Pandas is a Python library to deal with sequential and tabular data. It is built on top of the NumPy library and has two primary data structures i.e. Series (1-dimensional) and DataFrame (2-dimensional). It can handle both homogeneous and heterogeneous data.
- Matplotlib is a low level graph plotting library in python that serves as a visualization utility. It is single most used Python package for 2D-graphics along with limited support for 3D-graphics. Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias.
- Seaborn is a library in Python predominantly used for making statistical graphics. Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data.
- Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening.
  Pickling: It is a process where a Python object hierarchy is converted into a byte stream and dumps it into a file by using dump function. This character stream contains all the information necessary to reconstruct the object in another python script.
  Unpickling: It is the inverse of Pickling process where a byte stream is converted into an object hierarchy. While the process of retrieving original Python objects from the stored string representation is called unpickling. We use load function to do this.

## 2. Reading and understanding the data file –

```
data_set = pd.read_csv('cloth_size_test.csv')
data_set.shape
```

```
(119734, 4)
```

The basic process of loading data from a CSV file into a Pandas DataFrame is achieved using the “read_csv” function in Pandas.
NumPy arrays have an attribute called “shape” that returns a tuple with each index having the number of corresponding elements.
The output *(119734, 4)* shows that the data file has 4 columns each with 119734 entries.

```
data_set.head()
```

| | weight | age | height | size |
|---|---|---|---|---|
| 0 | 62 | 28.0 | 172.72 | XL |
| 1 | 59 | 36.0 | 167.64 | L |
| 2 | 61 | 34.0 | 165.10 | M |
| 3 | 65 | 27.0 | 175.26 | L |
| 4 | 62 | 45.0 | 172.72 | M |

"head()" returns the first 5 rows of the dataframe. To override the default, one may insert a value between the parentheses to change the number of rows returned.
The Dataset consists of 4 columns:
Weight (in Kgs)
Age
Height (in cm)
Size (XXS, S, M, L, XL, XXL, XXXL)
The output parameter is size and input parameters are Weight, Age, and Height.

## 3. Handling Missing Data –

```
data_set.isnull().sum()  #Checking for rows with null/NaN entries
```

```
weight      0
age       257
height    330
size        0
dtype: int64
```

```
data_set = data_set.dropna()   #Removing all such rows from our data_set
data_set.isnull().sum()
```

```
weight    0
age       0
height    0
size      0
dtype: int64
```

When we have empty values in our data, they'll often show up as NaN values. This missing data needs to be handled before we begin training on the data. There are two basic ways we can go about it. We can either remove rows with missing values or fill them in.

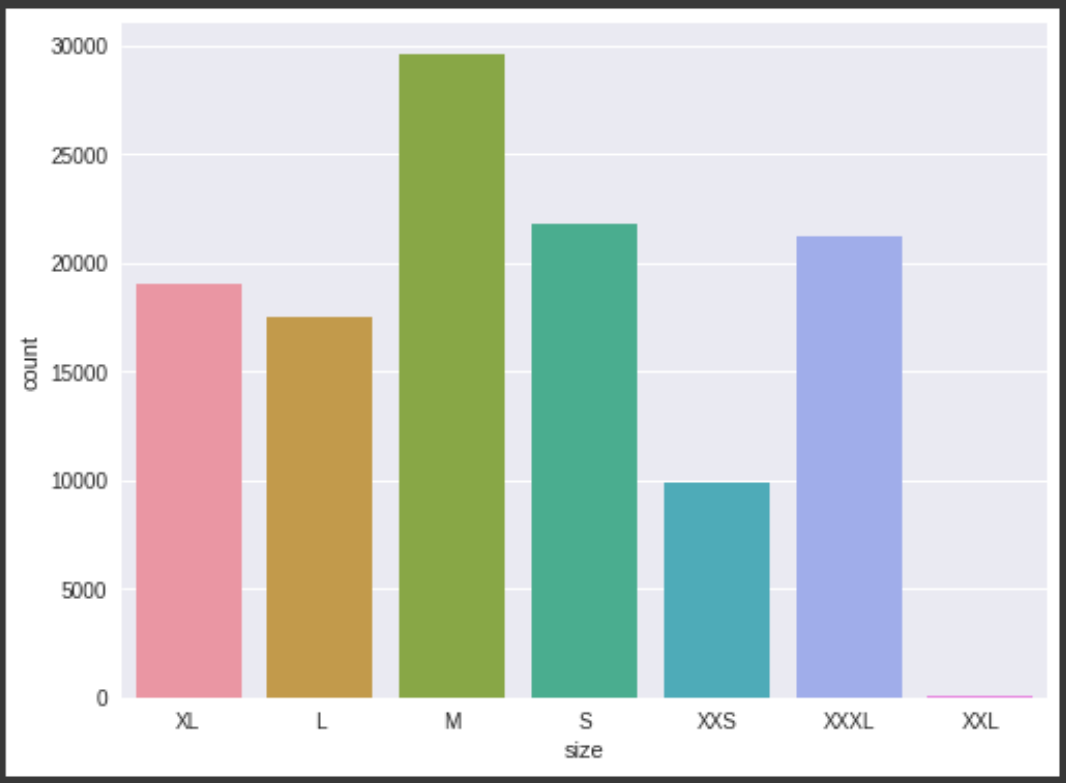
- Check if NaNs exist; we can use the isnull() method to find any possible NaN values that appear. We can use that with the .sum() method to get a count of all NaN values per column.
- Dealing with NaN values; there a few options that we have, depending on our dataset we can either remove or fill the missing values.
- Remove Missing Values; to achieve this we can use the built-in method, dropna(). This method automatically drops any row with NaN.
- Filling Missing Values; here we replace NaN values present by the average, median or most frequent value of the column.

## 4. Visualizing the data set –

```
data_set["size"].value_counts()
```

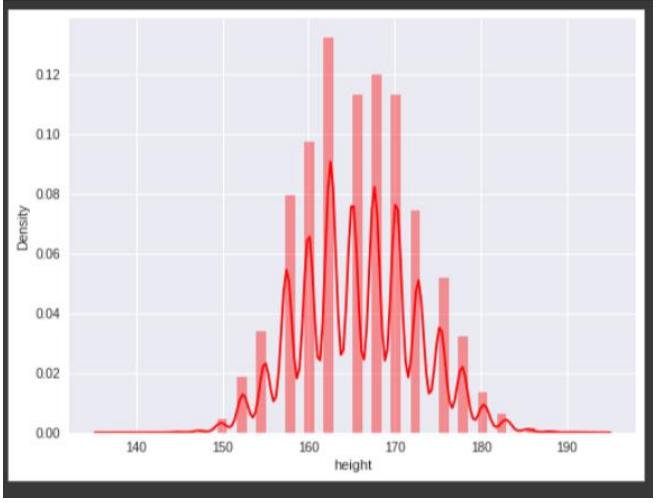
```
M       29575
S       21829
XXXL    21259
XL      19033
L       17481
XXS      9907
XXL        69
Name: size, dtype: int64
```

```
#visualize the number of entries for each size
fig, ax = plt.subplots(figsize=(8,6))
sns.countplot(x=data_set["size"]);
```



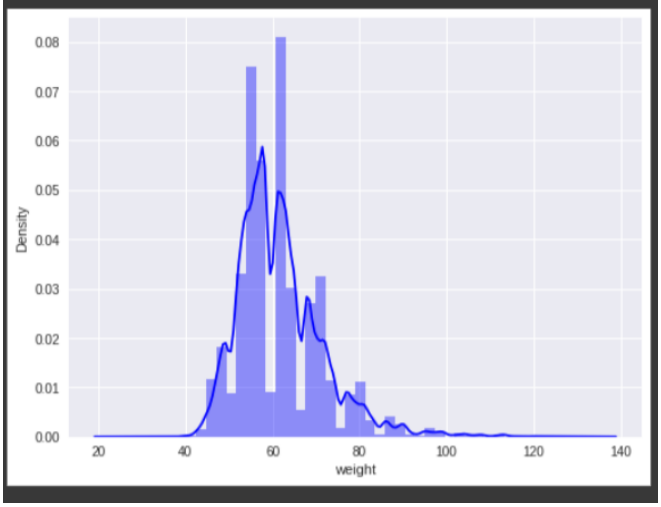

From the bar plot, we can infer that size M has the highest amount of entries whereas, XXL has the least.

```
fig, ax = plt.subplots(figsize=(8,6))
sns.distplot(data_set["height"], color="r");
```



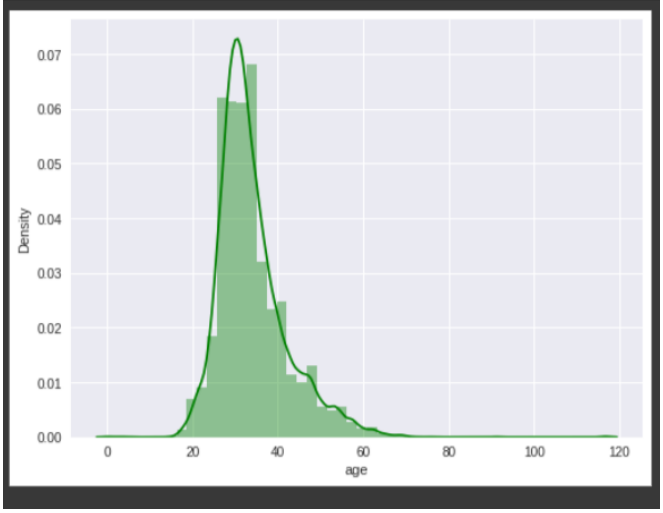

Majority of data present in the Height column is in the range of 160 – 170 cm.

```
fig, ax = plt.subplots(figsize=(8,6))
sns.distplot(data_set["weight"], color="b");
```



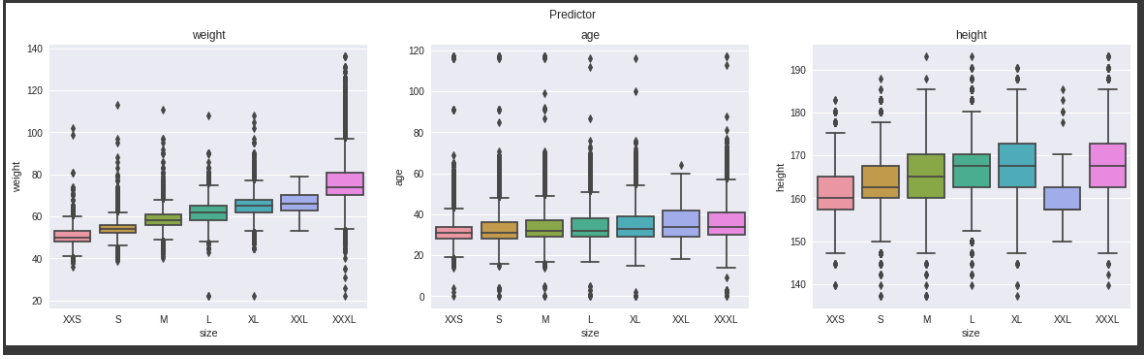

Majority of data present in the Weight column is in the range of 50 – 70 kgs.

```
fig, ax = plt.subplots(figsize=(8,6))
sns.distplot(data_set["age"], color="g");
```




Majority of data present in the Age column is in the range of 25 – 40 yrs.

```
fig, axes = plt.subplots(1,3,figsize=(20,5));
fig.suptitle('Predictor');
size_order = ['XXS','S','M','L','XL','XXL','XXXL']

# weight
sns.boxplot(x = 'size',y = 'weight', data = data_set, ax = axes[0], order=size_order);
axes[0].set_title('weight');

# age
sns.boxplot(x = 'size',y = 'age', data = data_set, ax = axes[1], order=size_order);
axes[1].set_title('age');

# height
sns.boxplot(x = 'size',y = 'height', data = data_set, ax = axes[2], order=size_order);
axes[2].set_title('height');
```




From the above boxplots we can observe that there is a good correlation between weight and size, since there is a gradual increase in size as the weight increases. Thus we can deduce that weight has the highest impact on size variation. Additionally we can also observe a lot of outliers which would hamper our training accuracy.

## 5. Setting up the data for training –

```
#Mapping the size of clothes
data_set['size'] = data_set['size'].map({'XXS': 1, 'S': 2, "M" : 3, "L" : 4, "XL" : 5, "XXL" : 6, "XXXL" : 7})
```

```
data_set.head()
```

| | weight | age | height | size |
|---|---|---|---|---|
| 0 | 62 | 28.0 | 172.72 | 5 |
| 1 | 59 | 36.0 | 167.64 | 4 |
| 2 | 61 | 34.0 | 165.10 | 3 |
| 3 | 65 | 27.0 | 175.26 | 4 |
| 4 | 62 | 45.0 | 172.72 | 3 |

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.
Since our size column contains words as labels, we are using map for label encoding here.

```
X = data_set.drop("size", axis=1)       #input parameters [weight, age, height]
y = data_set["size"]                    #output parameter [size]
```

```
print(X.shape)
print(y.shape)
```

```
(119153, 3)
(119153,)
```

```
X.head()
```

| | weight | age | height |
|---|---|---|---|
| 0 | 62 | 28.0 | 172.72 |
| 1 | 59 | 36.0 | 167.64 |
| 2 | 61 | 34.0 | 165.10 |
| 3 | 65 | 27.0 | 175.26 |
| 4 | 62 | 45.0 | 172.72 |

```
y.head()
```

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

Separating our data into two i.e. input and output

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

```
len(X_train), len(X_test)
```

```
(95322, 23831)
```

```
len(y_train), len(y_test)
```

```
(95322, 23831)
```

The train_test_split is a procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the machine learning model and is referred to as the training dataset. The second subset is used to evaluate the fit machine learning model and is known as the testing dataset.
We have set the test_size = 0.2, i.e. 20% of the complete data set will be used for testing and the remaining 80% of dataset is used for training our model.

## 6. Training / fitting –

- Linear regression:

```
from sklearn.metrics import accuracy_score,classification_report
from sklearn.linear_model import LinearRegression
Lreg = LinearRegression()
```

```
Lreg.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
Lreg.predict(X_test)
```

```
array([3.8923881 , 2.97285749, 3.15900999, ..., 5.26599531, 4.5102123 ,
       2.54025225])
```

```
LinearRegressionScore = Lreg.score(X_test,y_test)
print("Accuracy obtained by Linear Regression model:",LinearRegressionScore*100,"%")
```

```
Accuracy obtained by Linear Regression model: 64.60119935296535 %
```

- K Neighbors Classifier:

```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(42)
```

```
KNN.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=42, p=2,
                     weights='uniform')
```

```
KNN.predict(X_test)
```

```
array([3, 3, 3, ..., 7, 5, 2])
```

```
KNeighborsClassifierScore = KNN.score(X_test,y_test)
print("Accuracy obtained by K Neighbors Classifier model:",KNeighborsClassifierScore*100,"%")
```

```
Accuracy obtained by K Neighbors Classifier model: 51.57567873777852 %
```

- Decision Tree Classifier:

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
```

```
tree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
tree.predict(X_test)
```

```
array([3, 2, 3, ..., 7, 5, 2])
```

```
DecisionTreeClassifierScore = tree.score(X_test,y_test)
print("Accuracy obtained by Decision Tree Classifier model:",DecisionTreeClassifierScore*100,"%")
```

```
Accuracy obtained by Decision Tree Classifier model: 50.89169569048718 %
```

- Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

```
model.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

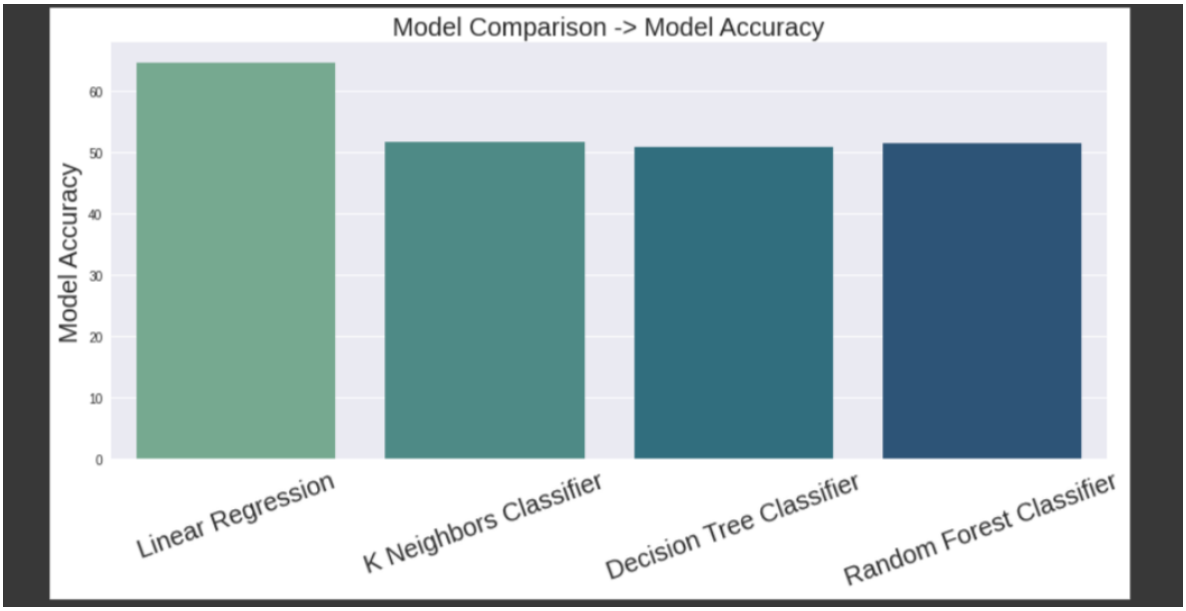
```
model.predict(X_test)
```

```
array([3, 2, 3, ..., 7, 5, 2])
```

```
RandomForestClassifierScore = model.score(X_test,y_test)
print("Accuracy obtained by Random Forest Classifier model:",RandomForestClassifierScore*100,"%")
```

```
Accuracy obtained by Random Forest Classifier model: 51.37006420289715 %
```

```
x = ["Linear Regression", "K Neighbors Classifier", "Decision Tree Classifier", "Random Forest Classifier" ]
y = [LinearRegressionScore*100, KNeighborsClassifierScore*100,  DecisionTreeClassifierScore*100, RandomForestClassifierScore*100]

fig, ax = plt.subplots(figsize=(14,6))
sns.barplot(x=x,y=y, palette="crest");
plt.ylabel("Model Accuracy",fontsize=20)
plt.xticks(rotation=20, fontsize=20)
plt.title("Model Comparison -> Model Accuracy",fontsize=20);
```




From the above bar-plot we can observe the comparison of model accuracy between the four models used for training, with linear regression having the highest accuracy of about 60%, whereas, the other three classification models have an accuracy of just about 50%. The low accuracy can be due to presence of large number of outliers present in our dataset, so let's try to remove these outliers and then test the classification models once again.

## 7. Removing Outliers (Z-score method) –

```
data_set_WO = []
sizes = []
for size_type in data_set['size'].unique():
    print('size type:',size_type)
    sizes.append(size_type)
    ndf = data_set[['age','height','weight']][data_set['size'] == size_type]
    zscore = ((ndf - ndf.mean())/ndf.std())
    data_set_WO.append(zscore)
```

```
size type: 5
size type: 4
size type: 3
size type: 2
size type: 1
size type: 7
size type: 6
```

```
for i in range(len(data_set_WO)):
    print(sizes[i])
    data_set_WO[i]['age'] = data_set_WO[i]['age'][(data_set_WO[i]['age']>-3) & (data_set_WO[i]['age']<3)]
    data_set_WO[i]['height'] = data_set_WO[i]['height'][(data_set_WO[i]['height']>-3) & (data_set_WO[i]['height']<3)]
    data_set_WO[i]['weight'] = data_set_WO[i]['weight'][(data_set_WO[i]['weight']>-3) & (data_set_WO[i]['weight']<3)]
```

```
5
4
3
2
1
7
6
```

```
for i in range(len(sizes)):
    data_set_WO[i]['size'] = sizes[i]
```

```
new_data_set = pd.concat(data_set_WO)
new_data_set.head()
```

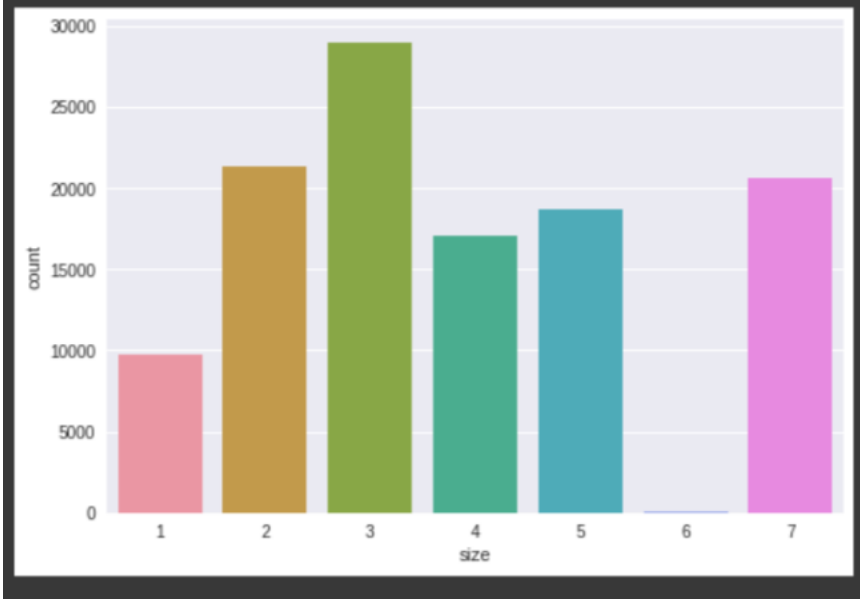
| | age | height | weight | size |
|---|---|---|---|---|
| **0** | -0.833765 | 0.767109 | -0.660874 | 5 |
| **24** | -0.713753 | 1.539084 | 1.556823 | 5 |
| **25** | -0.353718 | -1.162831 | -0.291257 | 5 |
| **28** | -0.473729 | -1.934807 | -1.400106 | 5 |
| **34** | 0.126330 | -0.776843 | 0.447975 | 5 |

```
new_data_set['age'][new_data_set['age']<-3]
new_data_set['height'][new_data_set['height']<-3]
new_data_set['weight'][new_data_set['weight']<-3]
```

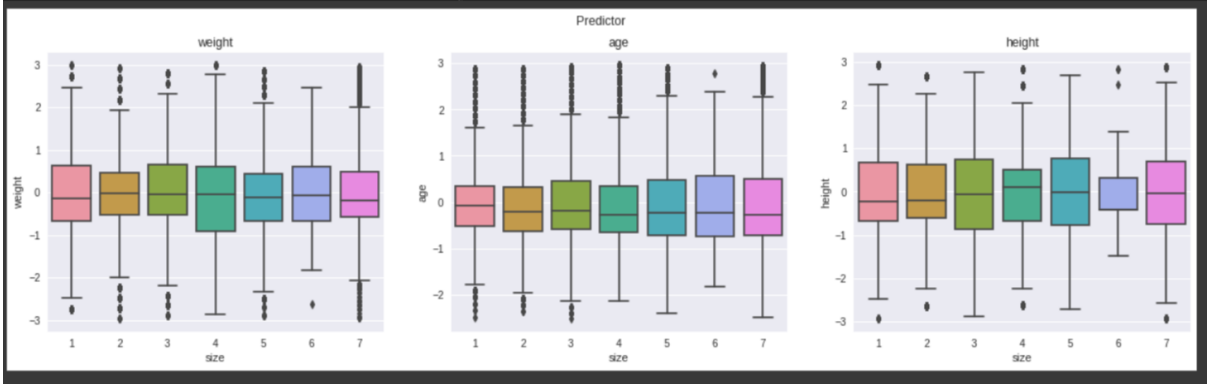
```
Series([], Name: weight, dtype: float64)
```

```
new_data_set = new_data_set.dropna()
```

```
#visualize the number of entries for each size after removing the outliers
plt.style.use('seaborn')
sns.countplot(x=new_data_set['size'])
plt.show()
```



```
fig, axes = plt.subplots(1,3,figsize=(20,5));
fig.suptitle('Predictor');

# weight
sns.boxplot(x = 'size',y = 'weight', data = new_data_set, ax = axes[0]);
axes[0].set_title('weight');

# age
sns.boxplot(x = 'size',y = 'age', data = new_data_set, ax = axes[1]);
axes[1].set_title('age');

# height
sns.boxplot(x = 'size',y = 'height', data = new_data_set, ax = axes[2]);
axes[2].set_title('height');
```




From the above boxplots we can observe that now the number of outliers have significantly reduced compared to before, mainly in weight and height parameters.

## 8. Training once again on new dataset after removing outliers –

```
new_X, new_y = new_data_set.drop('size', axis=1), new_data_set['size']
new_X.shape,new_y.shape
```

```
((116433, 3), (116433,))
```

```
new_x_train, new_x_test, new_y_train, new_y_test = train_test_split(new_X, new_y, test_size=0.2)
new_x_train.shape, new_x_test.shape
```

```
((93146, 3), (23287, 3))
```

```
new_y_train.shape, new_y_test.shape
```

```
((93146,), (23287,))
```

- K Neighbors Classifier 2.0

```
KNN_2 = KNeighborsClassifier(n_neighbors=7, metric='manhattan', weights='distance')
KNN_2.fit(new_x_train, new_y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='distance')
```

```
KNeighborsClassifierScore2 = KNN_2.score(new_x_test, new_y_test)
KNeighborsClassifierScore2
```

```
0.9425001073560355
```

```
print("Accuracy obtained by K Neighbors Classifier model:",KNeighborsClassifierScore2*100,"%")
```

```
Accuracy obtained by K Neighbors Classifier model: 94.25001073560355 %
```

- Decision Tree Classifier 2.0

```
tree2 = DecisionTreeClassifier()
tree2.fit(new_x_train,new_y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
DecisionTreeClassifierScore2 = tree2.score(new_x_test,new_y_test)
DecisionTreeClassifierScore2
```

```
0.9997852879288873
```

```
print("Accuracy obtained by Decision Tree Classifier model:",DecisionTreeClassifierScore2*100,"%")
```

```
Accuracy obtained by Decision Tree Classifier model: 99.97852879288874 %
```

- Random Forest Classifier 2.0

```
RF_2 = RandomForestClassifier()
RF_2.fit(new_x_train, new_y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

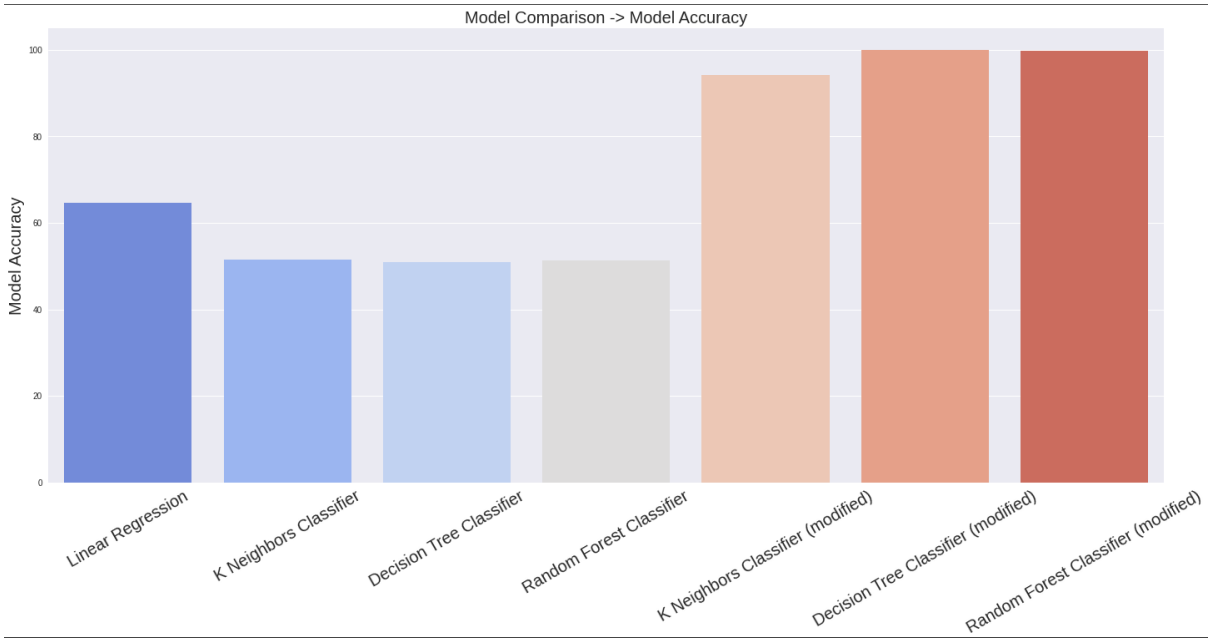
```
RandomForestClassifierScore2 = RF_2.score(new_x_test, new_y_test)
RandomForestClassifierScore2
```

```
0.9984540730879891
```

```
print("Accuracy obtained by Random Forest Classifier model:",RandomForestClassifierScore2*100,"%")
```

```
Accuracy obtained by Random Forest Classifier model: 99.84540730879891 %
```

```
x = ["Linear Regression", "K Neighbors Classifier",
     "Decision Tree Classifier", "Random Forest Classifier",
     "K Neighbors Classifier (modified)", "Decision Tree Classifier (modified)",
     "Random Forest Classifier (modified)"]
y = [LinearRegressionScore*100, KNeighborsClassifierScore*100,
     DecisionTreeClassifierScore*100, RandomForestClassifierScore*100,
     KNeighborsClassifierScore2*100, DecisionTreeClassifierScore2*100,
     RandomForestClassifierScore2*100]
fig, ax = plt.subplots(figsize=(24,10))
sns.barplot(x=x,y=y, palette="coolwarm");
plt.ylabel("Model Accuracy",fontsize=20)
plt.xticks(rotation=30,fontsize=20)
plt.title("Model Comparison -> Model Accuracy",fontsize=20);
```




Thus, we can infer from the above plot that after removing the outliers, the accuracy score of the models has increased significantly (almost twice).

One can find the complete .ipynb file here, and the dataset used in this can be found here.

## 9. Loading the trained data into a pickle file –

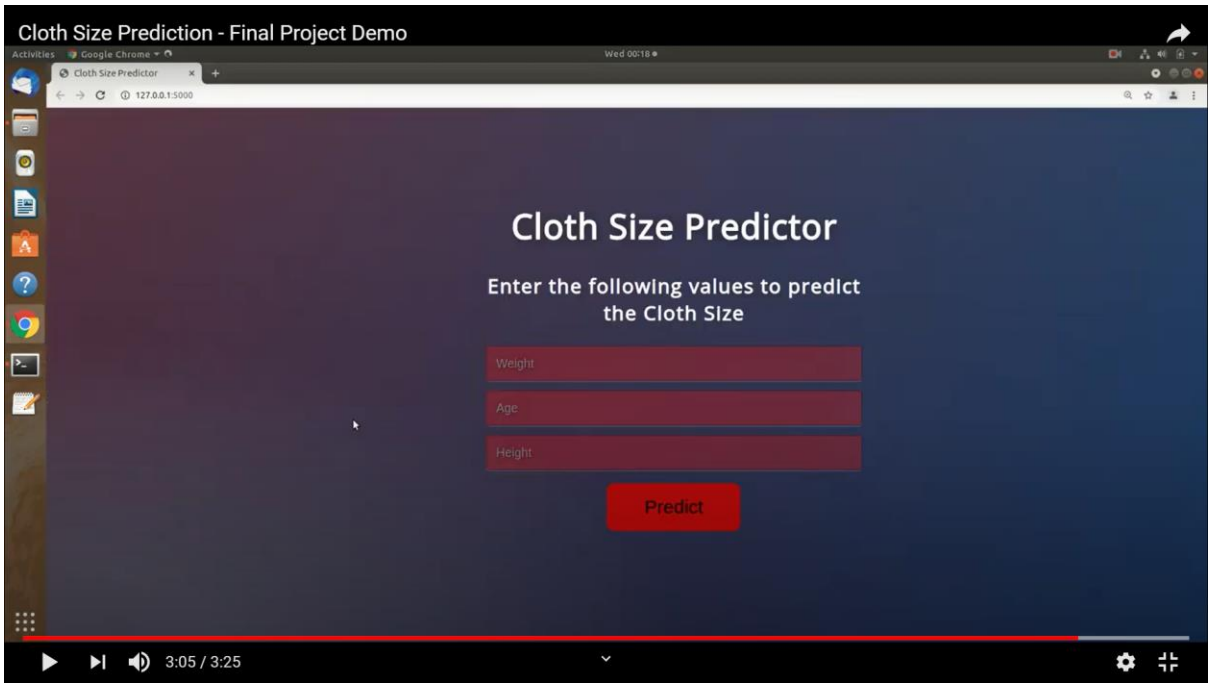
```
pickle.dump(Lreg, open('LinearReg.pkl','wb'))
pickle.dump(KNN, open('KNNclassifier.pkl','wb'))
pickle.dump(tree, open('DTclassifier.pkl','wb'))
pickle.dump(forest, open('RFclassifier.pkl','wb'))
```

Once the trained model is stored into a pickle file, we can use it any time in the future for prediction purpose.

## 10. Front End: Prediction Application

```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

#Initialize the flask App
app = Flask(__name__)

model = pickle.load(open('model.pkl', 'rb'))

#default page of our web-app
@app.route('/')
def home():
    return render_template('index.html')

#To use the predict button in our web-app
@app.route('/predict',methods=['POST'])
def predict():
        '''
        For rendering results on HTML GUI
        '''
        int_features = [float(x) for x in request.form.values()]
        final_features = [np.array(int_features)]
        prediction = model.predict(final_features)

        pred = int(prediction)
        sizemap = { 1:'XXS', 2:'S', 3:'M', 4:"L", 5:"XL", 6:"XXL", 7:"XXXL"}
        pred_size = sizemap.get(pred)

        return render_template('index.html', prediction_text='Size predicted is :{}'.format(pred_size))

if __name__ == "__main__":
    app.run(debug=True)
```

We are using flask for creating the front end, one can find all the necessary files used in this [github repository] and can go through the demo video present below.



## CONCLUSION:

This project deals with building a model for predicting the size of cloth for a customer, done with the help of the Scikit-learn machine learning library. I have used various machine learning algorithms present in the library to train and compare the models with each other. It is to be considered that the dataset used for this project contains loads of outliers, which have a massive impact on the model accuracy score.

With the outliers present, the regression model performed much better compared to the classification models getting an accuracy score of about 65% to that of 50%, respectively. After removing most of the outliers present, the classification models performed very well indeed, by achieving a significantly higher accuracy score by almost doubling it to 99%.

But the accuracy score doesn't reveal everything since, while testing our prediction model, the models trained with outliers, the linear regression one to be more specific, out-performed the models trained without outliers. But this seems to be odd considering their accuracy score. The reason for such behaviour is most probable that the input provided during prediction itself fell into a category of outlier. Thus, we can deduce that removing outliers is not always the best option for a machine learning application, especially when our training dataset comprises an ample amount of outliers. Therefore, it is recommended that don't get rid of the outliers until and unless we are sure that it is because of some miscalculation, since outliers can contain some unique information which might be useful for training out model.

## References:


- https://www.kaggle.com/tourist55/clothessizeprediction
- https://towardsdatascience.com/machine-learning-handling-missing-data-27b09ab146ba
- https://data-flair.training/blogs/machine-learning-tutorial/
- https://www.javatpoint.com/machine-learning
- https://www.analyticsvidhya.com/blog/2021/05/feature-engineering-how-to-detect-and-remove-outliers-with-python-code/
- https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51