
# Rizvi College of Engineering

Department of **Computer** Engineering

Mini Project Synopsis Report

on

## Big Mart Sales Prediction

Submitted in partial fulfillment of the

requirementsof the Mini-Project 2A Year of

Bachelors of Engineering

By

**Utsav G. Kuntalwad (201P049)**

**Srushti S. Sawant (201P037)**

**Prerna S. Shakwar (201P041)**

**Mayur R. Kyatham (201P013)**

Guide:

Prof. **Vishal Patil**


University of Mumbai(2021 – 2022)

# *Certificate*

This is to certify that the project synopsis entitled "**Big Mart Sales Prediction**" has been submitted by **Mayur R. Kyatham, Srushti S. Sawant, Utsav G. Kuntalwad, Prerna S. Shakwar** under the guidance of **Prof. Vishal Patil** in partial fulfillment of the requirement for the award of the Degree of Bachelor of Engineering in **Computer** Engineering from University of Mumbai.

Certified By

| | |
|---|---|
| **Prof. Vishal Patil** | **Prof. Shiburaj Pappu** |
| Project Guide | Head of Department |
| Prof._____ | Prof._____ |
| Internal Examiner | External Examiner |

Dr. Varsha Shah

Principal

Department of **Computer** Engineering

**Rizvi College of Engineering,**

Off Carter Road, Bandra(W), Mumbai-400050

# Index

# Abstract

***" Nowadays shopping malls and Big Marts keep the track of their sales data of each and every individual item for predicting future demand of the customer and update the inventory management as well. These data stores basically contain a large number of customer data and individual item attributes in a data warehouse. Further, anomalies and frequent patterns are detected by mining the data store from the data warehouse. The resultant data can be used for predicting future sales volume with the help of different machine learning techniques for the retailers like Big Mart. In this paper, we propose a predictive model using Random Forest Regressor technique for predicting the sales of a company like Big Mart and found that the model produces better performance as compared to existing models. A comparative analysis of the model with others in terms performance metrics is also explained in details "***

# Introduction

Lets see how we can build this Machine learning system that can predict big mart sales prediction .So first of all lets try to understand this problem statement better lets say that there is a grocery store company or a supermarket company that has several outlets or several stores around the world and they want us to predict the sales which they can expect okay so you may ask what is the application of predicting this sales what can happen by predicting the sales so there are actually a lot of things that are helpful which we can derive from this predictions so if we can predict the sales revenue we can tell the company that what are the challenges they may face what are the brands and what are the products that sells the most and other such kind of things so this helps the sales team to understand which product to sell and which product to promote and such kind of thins and they can also make several marketing plans lets say that a particular store is selling the most and we may find some insights from it as of why this product is selling the most and this helps the company to make better marketing decisions so these are some advantages of this particular project


So lets understand one more thing so this project is a regression based ML project so machine learning is generally classified into supervised learning and unsupervised learning so in supervised learning we train our machine learning model with labeled dataset whereas in unsupervised learning we train the model with unlabeled datasets so you may ask what is mean by this labels and

what is meant by unlabeled dataset so in this project the labels are the target so to be more precise we should say that these are targets so in this case


the targets are the sales amount so sales cost or price they are getting so that is the target if we train our machine learning model with those target as well we call it a supervised learning in case we don't use those labels or targets we call it as unsupervised learning and this supervised learning can be classified into two types classification and regression okay and classification is about predictiing a class or discrete values so we are just predicting some categories so these categorues can be male or female , true or false and such kind of thins whereas regression is about predicting a particular value or continuous value so this can be like predicting a person says and such kind of things where this is not category but some numerical values so this is called the regression and this project is called as regression because we are predicting the sales so now lets understand the work flow first we got the data set for this project we are going to use bug mart sales data from kaggle when it comes to machine learning we will be disussing what are the different features and variables this data set contains later we will be preparing the data replacing the null values remving some useless variables to make our work easy then we will be using explonatory data analysis (EDA) with the help of Dtale, Pandas Profiling and Klib then we will be cleaning the data with the help of Klib and then do some pre processing task like label encoding splitting our data in the form of train and test csv files then we will be doing standarization after all the pre processing task we will be finally building our model we will test different ML models like linear regression , random forest regressor and XG boost regressor then to improve our accuracy we will be using hyper parameter tuning and after collecting the data and analysing it we will deploy our model with the help of python and flask this how we have worked in this project

# Literature Survey (1)

## Sales Prediction Model for Big Mart

**Nikita Malik, Karan Singh**

**Abstract**: Machine Learning is a category of algorithms that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build modelsand employ algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available. These models can be applied in different areas and trained to match the expectations of management so that accurate steps can be taken to achieve the organization's target.In this paper, the case of Big Mart, a one-stop-shoppingcenter, has been discussed to predict the sales of different types of items and for understanding the effects of different factors on the items' sales.Taking various aspects of a dataset collected for Big Mart, and the methodology followed for building a predictive model, results with high levels of accuracy are generated, and these observations can be employed to take decisions to improve sales.

**Keywords**: Machine Learning, Sales Prediction, Big Mart, Random Forest, Linear Regression

### Introduction:

In today's modern world, huge shopping centers such as big malls and marts are recording data related to sales of items or products with their various dependent or independent factors as an important step to be helpful in prediction of future demands and inventory management. The dataset built with various dependent and independent variables is a composite form of item attributes, data gathered by means of customer, and also data related to inventory management in a data warehouse. The data is thereafter refined in order to get accurate predictions and gather new as well as interesting results that shed a new light on our knowledge with respect to the task's data. This can then further be used for forecasting future sales by means of employing machine learning algorithms such as the random forests and simple or multiple linear regression model.

### Cause:

In this paper, basics of machine learning and the associated data processing and modeling algorithms have been described, followed by their application for the task of sales prediction in Big Mart shopping centers at different locations. On implementation, the prediction results show the correlation among different attributes considered and how a particular location of medium size recorded the highest sales, suggesting that other shopping locations should follow similar patterns for improved sales.

### Future scope:

Multiple instances parameters and various factors can be used to make this sales prediction more innovative and successful. Accuracy, which plays a key role in prediction-based systems, can be significantly increased as the number of parameters used are increased. Also, a look into how the sub-models work can lead to increase in productivity of system. The project can be further collaborated in a web-based application or in any device supported with an in-built intelligence by virtue of Internet of Things (IoT), to be more feasible for use. Various stakeholders concerned with sales information can also provide more inputs to help in hypothesis generation and more instances can be taken into consideration such that more precise results that are closer to real world situations are generated. When combined with effective data mining methods and properties, the traditional means could be seen to make a higher and positive effect on the overall development of corporation's tasks on the whole. One of the main highlights is more expressive regression outputs, which are more understandable bounded with some of accuracy. Moreover, the flexibility of the proposed approach can be increased with variants at a very appropriate stage of regression modelbuilding. There is a further need of experiments for proper measurements of both accuracy and resource efficiency to assess and optimize correctly.

# Literature Survey (2)

## Big Mart Sales Using Machine Learning with Data analysis

**AYESHA SYED, ASHA JYOTHI KALLURI, VENKATESWARA REDDY POCHA, VENKATA ARUN KUMAR DASARI, B.RAMASUBBAIAH**

**Abstract**: The sales forecast is based on BigMart sales for various outlets to adjust the business model to expected outcomes.The resulting data can then be used to prediction potential sales volumes for retailers such as BigMart through various machine learning methods. The estimate of the system proposed shouldtake account of price tag, outlet and outlet location. A number of networks use the various machine- learning algorithms, such as linear regression and decision tree algorithms, and an XGBoost regressor, which offers an efficient prevision of BigMart sales based on gradient. At last, hyperparameter tuning is used to help you to choose relevant hyperparameters that make the algorithmshine and produce the highest accuracy

**Keywords:** Machine Learning Algorithms, Prediction, Reliability,Sales forecasting ,Prediction model, Regression

### Introduction:

Every item is tracked for its shopping centers and BigMarts in order to anticipate a future demand of the customer and also improve the management of its inventory. BigMart is an immense network of shops virtually all over the world. Trends in BigMart are very relevant and data scientists evaluate those trends per product and store in order to create potential centres.Using the machine to forecast the transactions of BigMart helps data scientists to test the various patterns by store and product to achieve the correct results. Many companies rely heavily on the knowledge base and need market patterns to be forecasted. To address the issue of deals expectation of things dependent on client's future requests in various BigMarts across different areas diverse Machine Learning algorithms like Linear Regression, Random Forest, Decision Tree, Ridge Regression, XGBoost are utilized for gauging of deals volume. Deals foresee the outcome as deals rely upon the sort of store, populace around the store, a city wherein the store is located.

### Cause:

Experts also shown that a smart sales forecasting program is required to manage vast volumes of data for business organizations. Business assessments are based on the speed and precision of the methods used to analyze the results. The Machine Learning Methods presented in this research paper should provide an effective method for data shaping and decision-making. New approaches that can better identify consumer needs and formulate marketing plans will be implemented.

### Future scope:

To forecast BigMart's revenue, simple to advanced machine learning algorithms have been implemented, such as Linear Regression, Ridge Regression, Decision Tree, Random Forest, XGBoost. It has been observed that increased efficiency is observed with XGBoost algorithms with lower RMSE rating. As a result , additional Hyperparameter Tuning was conducted on XGBoost with Bayesian Optimization technique due to its quick and fairly simple computation, which culminated in the acquisition of the lowest RMSE value

# Tools's Used

## Jupyter Notebook:

Jupyter Notebook (formerly IPython Notebook) is a web-based interactive computational environment for creating notebook documents. Jupyter Notebook is built using several open-source libraries, including IPython, ZeroMQ, Tornado, jQuery, Bootstrap, and MathJax. A Jupyter Notebook document is a browser-based REPL containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media. Underneath the interface, a notebook is a JSON document, following a versioned schema, usually ending with the ".ipynb" extension.

Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s. Jupyter interest overtook the popularity of the Mathematica notebook interface in early 2018.

JupyterLab is a newer user interface for Project Jupyter, offering a flexible user interface and more features than the classic notebook UI. The first stable release was announced on February 20, 2018. In 2015, a joint $6 million grant from The Leona M. and Harry B. Helmsley Charitable Trust, The Gordon and Betty Moore Foundation, and The Alfred P. Sloan Foundation funded work that led to expanded capabilities of the core Jupyter tools, as well as to the creation of JupyterLab.

JupyterHub is a multi-user server for Jupyter Notebooks. It is designed to support many users by spawning, managing, and proxying many singular Jupyter Notebook servers

## Python :

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0.[] Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

## PyCharm:

PyCharm is a dedicated Python integrated development environment (IDE) providing a wide range of essential tools for python developers, tightly integrated to create a convenient environment for productive python, web and data science development

## Flask:

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other

components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools

## Pandas :

Data manipulation and analysis it offers data structure and operations for manipulating numerical tables and time series

## Numpy :

Adding support for large, multidimensional arrays and matrices large collection of high level maths functions to operate

## Seaborn :

Uses matplotlib underneats to plot graphs visualize random distributions

## Matplotlib:

Creating static, animated and interactive visualizations

## Klib :

Importing , cleaning , analyzing and pre processing data

## Scikit-learn :

data analysis library

## joblib :

Provide light weight pipelining in python parallel computing

## Pandas-profiling :

Allows us to become familiar with our data by exploring it from multiple angles through statistics data visualizations and data summaries

## XG boost:

Provides parallel tree boosting and is the leading machine learning library for regression classification and ranking problems

# Methodology

# 1. Collection of data

Kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

Kaggle got its start in 2010 by offering machine learning competitions and now also offers a public data platform, a cloud-based workbench for data science, and Artificial Intelligence education. Its key personnel were Anthony Goldbloom and Jeremy Howard. Nicholas Gruen was the founding chair succeeded by Max Levchin. Equity was raised in 2011 valuing the company at $25.2 million. On 8 March 2017, Google announced that they were acquiring Kaggle

The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and predict the sales of each product at a particular outlet.

Using this model, BigMart will try to understand the properties of products and outlets which play a key role in increasing sales.

Please note that the data may have missing values as some stores might not report all the data due to technical glitches. Hence, it will be required to treat them accordingly.

We have train (8523) and test (5681) data set, train data set has both input and output variable(s). We need to predict the sales for test data set.

- Item_Identifier: Unique product ID
- Item_Weight: Weight of product
- Item_Fat_Content: Whether the product is low fat or not
- Item_Visibility: The % of total display area of all products in a store allocated to the particular product
- Item_Type: The category to which the product belongs
- Item_MRP: Maximum Retail Price (list price) of the product
- Outlet_Identifier: Unique store ID
- Outlet_Establishment_Year: The year in which store was established
- Outlet_Size: The size of the store in terms of ground area covered
- Outlet_Location_Type: The type of city in which the store is located
- Outlet_Type: Whether the outlet is just a grocery store or some sort of supermarket
- Item_Outlet_Sales: Sales of the product in the particulat store. This is the outcome variable to be predicted.

# 2) Preparation of data

## Checking the number of null values in the train and test dataset

```
In [7]: df_train.isnull().sum()  #seeing the number of null values in the dataset
```

```
Out[7]: Item_Identifier                0
        Item_Weight                 1463
        Item_Fat_Content               0
        Item_Visibility                0
        Item_Type                      0
        Item_MRP                       0
        Outlet_Identifier              0
        Outlet_Establishment_Year      0
        Outlet_Size                 2410
        Outlet_Location_Type           0
        Outlet_Type                    0
        Item_Outlet_Sales              0
        dtype: int64
```

```
In [8]: df_test.isnull().sum()
```

```
Out[8]: Item_Identifier                0
        Item_Weight                  976
        Item_Fat_Content               0
        Item_Visibility                0
        Item_Type                      0
        Item_MRP                       0
        Outlet_Identifier              0
        Outlet_Establishment_Year      0
        Outlet_Size                 1606
        Outlet_Location_Type           0
        Outlet_Type                    0
        dtype: int64
```

## Checking the type of targets (objects, category, numerical etc)

```
In [9]: df_train.info()   #seeing the detailed info of the dataset and its types of target variables
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8523 non-null   object
 1   Item_Weight                7060 non-null   float64
 2   Item_Fat_Content           8523 non-null   object
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object
 7   Outlet_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                6113 non-null   object
 9   Outlet_Location_Type       8523 non-null   object
 10  Outlet_Type                8523 non-null   object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

## Using describe() to see the statistics of the target(mean median mode etc)

The describe() method returns description of the data in the DataFrame.

If the DataFrame contains numerical data, the description contains these information for each column:

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.
75% - The 75% percentile*.
max - the maximum value.

```
In [10]: df_train.describe()  # to generate descriptive statistics that summarize the central tendency, dispersion and
                              # shape of a dataset's distribution, excluding NaN values.
```

Out[10]:

| | Item_Weight | Item_Visibility | Item_MRP | Outlet_Establishment_Year | Item_Outlet_Sales |
|---|---|---|---|---|---|
| count | 7060.000000 | 8523.000000 | 8523.000000 | 8523.000000 | 8523.000000 |
| mean | 12.857645 | 0.066132 | 140.992782 | 1997.831867 | 2181.288914 |
| std | 4.643456 | 0.051598 | 62.275067 | 8.371760 | 1706.499616 |
| min | 4.555000 | 0.000000 | 31.290000 | 1985.000000 | 33.290000 |
| 25% | 8.773750 | 0.026989 | 93.826500 | 1987.000000 | 834.247400 |
| 50% | 12.600000 | 0.053931 | 143.012800 | 1999.000000 | 1794.331000 |
| 75% | 16.850000 | 0.094585 | 185.643700 | 2004.000000 | 3101.296400 |
| max | 21.350000 | 0.328391 | 266.888400 | 2009.000000 | 13086.964800 |

## Replacing numerical columns null values with mean imputation(item weight)

mean imputation in which the missing values are replaced with the mean value of the entire feature column. In the case of fields like salary, the data may be skewed as shown in the previous section. In such cases, it may not be a good idea to use mean imputation for replacing the missing values. Note that imputing missing data with mean values can only be done with **numerical data**.

```
In [11]: df_train['Item_Weight'].describe()  #seeing all the central tendenies of the dataset
```

```
Out[11]: count    7060.000000
         mean       12.857645
         std         4.643456
         min         4.555000
         25%         8.773750
         50%        12.600000
         75%        16.850000
         max        21.350000
         Name: Item_Weight, dtype: float64
```

```
In [12]: df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)  #replacing null values with mean values
         df_test['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
```

```
In [13]: df_train.isnull().sum()  #no null values in item weight
```

```
Out[13]: Item_Identifier               0
         Item_Weight                   0
         Item_Fat_Content              0
```

# Replacing categorical columns null values with mode imputation (outlet size)->special mode()[0]

mode imputation in which the missing values are replaced with the mode value or most frequent value of the entire feature column. When the data is skewed, it is good to consider using mode values for replacing the missing values. For data points such as the salary field, you may consider using mode for replacing the values. Note that imputing missing data with mode values can be done with numerical and categorical data.
Here is the python code sample where the mode of salary column is replaced in place of missing values in the column:

```
1 df['salary'] = df['salary'].fillna(df['salary'].mode()[0])
```

Here is how the data frame would look like (df.head())after replacing missing values of the salary column with the mode value.

```
In [18]: df_train['Outlet_Size'].fillna(df_train['Outlet_Size'].mode()[0],inplace=True)
         df_test['Outlet_Size'].fillna(df_test['Outlet_Size'].mode()[0],inplace=True)
```

**pandas treats the mode as something special since they can be unimodal , bimodal or multimodal distributions they**

**had to make sure that 1 value could be returned "Always return series even if only one value is returned"**

```
In [19]: df_train.isnull().sum()  #no null value :)
```

```
Out[19]: Item_Identifier              0
         Item_Weight                  0
         Item_Fat_Content             0
         Item_Visibility              0
         Item_Type                    0
         Item_MRP                     0
         Outlet_Identifier            0
         Outlet_Establishment_Year    0
         Outlet_Size                  0
         Outlet_Location_Type         0
         Outlet_Type                  0
         Item_Outlet_Sales            0
         dtype: int64
```

# Removing waste targets (item identifier and outlet identifier)

```
In [21]: df_train.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
         df_test.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

```
In [22]: df_train
```

Out[22]:

| | Item_Weight | Item_Fat_Content | Item_Visibility | Item_Type | Item_MRP | Outlet_Establishment_Year | Outlet_Size | Outlet_Location_Type | Outlet_Type | Item_Out |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9.300 | Low Fat | 0.016047 | Dairy | 249.8092 | 1999 | Medium | Tier 1 | Supermarket Type1 | 3 |
| 1 | 5.920 | Regular | 0.019278 | Soft Drinks | 48.2692 | 2009 | Medium | Tier 3 | Supermarket Type2 | |
| 2 | 17.500 | Low Fat | 0.016760 | Meat | 141.6180 | 1999 | Medium | Tier 1 | Supermarket Type1 | 2 |
| 3 | 19.200 | Regular | 0.000000 | Fruits and Vegetables | 182.0950 | 1998 | Medium | Tier 3 | Grocery Store | |
| 4 | 8.930 | Low Fat | 0.000000 | Household | 53.8614 | 1987 | High | Tier 3 | Supermarket Type1 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 8518 | 6.865 | Low Fat | 0.056783 | Snack Foods | 214.5218 | 1987 | High | Tier 3 | Supermarket Type1 | 2 |
| 8519 | 8.380 | Regular | 0.046982 | Baking Goods | 108.1570 | 2002 | Medium | Tier 2 | Supermarket Type1 | |
| 8520 | 10.600 | Low Fat | 0.035186 | Health and Hygiene | 85.1224 | 2004 | Small | Tier 2 | Supermarket Type1 | |
| 8521 | 7.210 | Regular | 0.145221 | Snack Foods | 103.1332 | 2009 | Medium | Tier 3 | Supermarket Type2 | 1 |

# 3) Exploratory Data Analysis (EDA)

## EDA

Exploratory Data Analysis, EDA for short, is simply a 'first look at the data'. It forms a critical part of the machine learning workflow and it is at this stage we start to understand the data we are working with and what it contains. In essence, it allows us to make sense of the the data before applying advanced analytics and machine learning.

During EDA we can begin to identify patterns within the data, understand relationships between features, identify possible outliers that may exist within the dataset and identify if we have missing values. Once we have gained an understanding about the data and we can then check whether further processing is required or if data cleaning is necessary.

When working with Python or if you are working through a Python training course, you will typically carry out EDA on your data using pandas and matplotlib. Pandas has a number of functions including df.describe() and df.info() which help summarise the statistics of the dataset, and matplotlib has a number of plots such as barplots, scatter plots and histograms to allow us to visualise our data.

When working with machine learning or data science training datasets the above methods may be satisfactory as much of the data has already been cleaned and engineered to make it easier to work with. In real world datasets, data is often dirty and requires cleaning. This can be a time consuming task to check using the methods above. This is where auto EDA can come to the rescue and help us speed up this part of the workflow without compromising on quality.

## Pandas Profiling

Pandas Profiling is a Python library that allows you to generate a very detailed report on our pandas dataframe without much input from the user. It

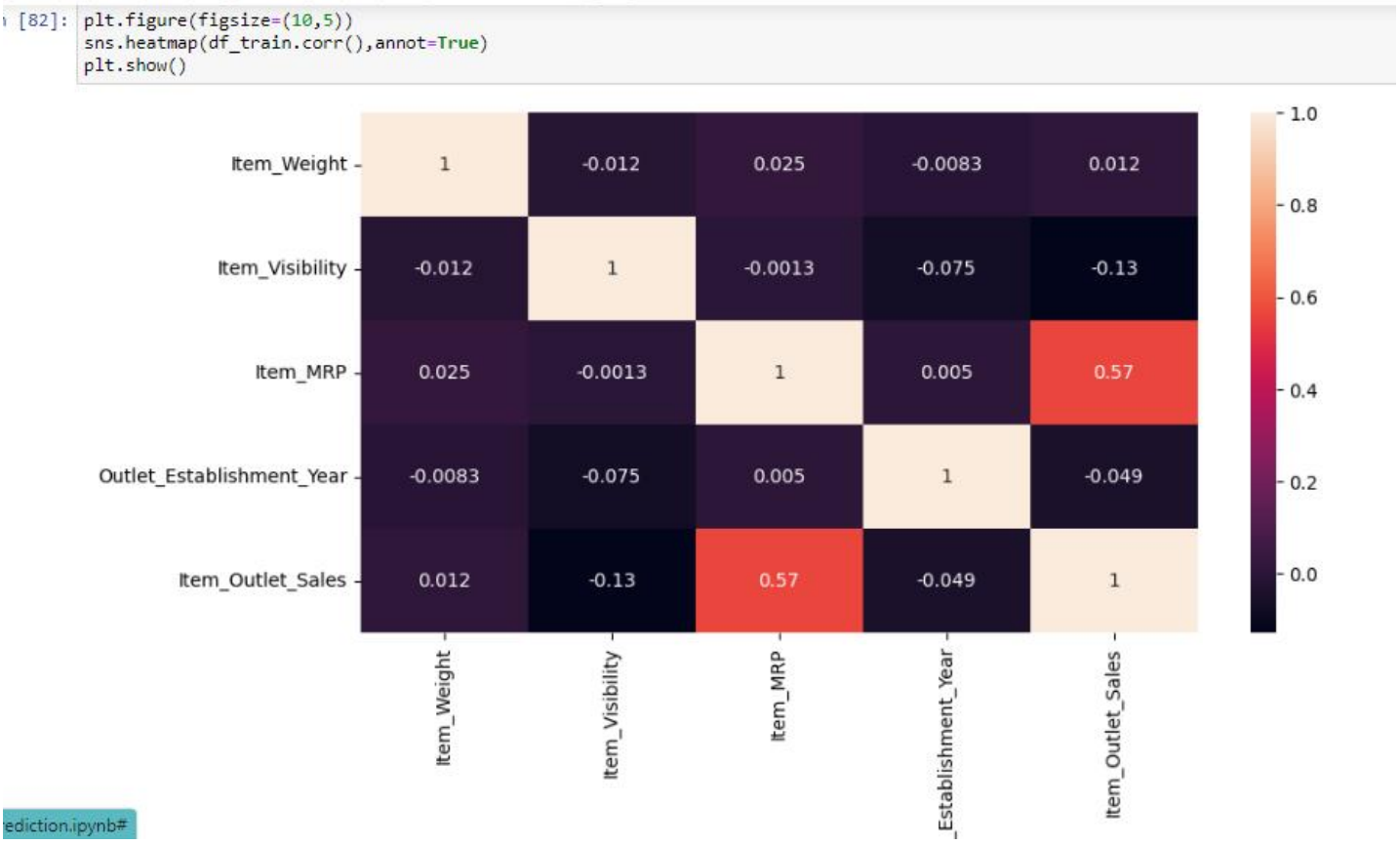
According to PyPi Stats, the library has over 1,000,000 downloads each month, which proves its a very popular library within data science.


### Correlations

The correlations section allows us to understand the degree at which two variables are correlated with one another. Within the pandas_profile report, we can view different methods of correlation:

- Spearman's ρ
- Pearson's r
- Kendall's τ
- Phik (φk)

If you are unsure what each method is, you can click on the button "Toggle Correlation Descriptions" and it will provide details of the meaning of each method.



# Klib

Klib is a Python library that provides amazing functionality for exploring your data in just a few lines of code. If you find that data exploration takes a lot of time, you can use this library as it gives you all the functions that will help you to explore, clean and prepare your data. If you've never used the Klib library in Python, this article is for you. In this article, I will introduce you to a tutorial on the Klib library in Python.

# klib.describe - functions for visualizing datasets
klib.cat_plot(df_train) # returns a visualization of the number and frequency of categorical features

klib.corr_mat(df_train) # returns a color-encoded correlation matrix
In the correlation matrix above, the red coloured values represent a negative correlation and the black coloured values represent a positive correlation. You can also visualize the correlation plot using this library as shown below:

Out[88]:

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type | outlet_ |
|---|---|---|---|---|---|---|---|---|---|
| item_weight | 1.00 | -0.02 | -0.01 | 0.03 | 0.02 | -0.01 | -0.01 | 0.00 | - |
| item_fat_content | -0.02 | 1.00 | 0.03 | -0.12 | -0.00 | -0.00 | -0.01 | -0.00 | - |
| item_visibility | -0.01 | 0.03 | 1.00 | -0.04 | -0.00 | -0.07 | 0.07 | -0.03 | - |
| item_type | 0.03 | -0.12 | -0.04 | 1.00 | 0.03 | 0.00 | -0.00 | 0.00 | |
| item_mrp | 0.02 | -0.00 | -0.00 | 0.03 | 1.00 | 0.01 | 0.01 | 0.00 | - |
| outlet_establishment_year | -0.01 | -0.00 | -0.07 | 0.00 | 0.01 | 1.00 | 0.19 | -0.09 | - |
| outlet_size | -0.01 | -0.01 | 0.07 | -0.00 | 0.01 | 0.19 | 1.00 | -0.61 | - |
| outlet_location_type | 0.00 | -0.00 | -0.03 | 0.00 | 0.00 | -0.09 | -0.61 | 1.00 | |
| outlet_type | -0.00 | -0.00 | -0.17 | 0.00 | -0.00 | -0.12 | -0.20 | 0.47 | |
| item_outlet_sales | 0.01 | 0.01 | -0.13 | 0.02 | 0.57 | -0.05 | -0.09 | 0.09 | |

klib.corr_plot(df_train) # returns a color-encoded heatmap, ideal for correlations
Understanding the distribution of each column is also very important to understand what kind of data you are working with. So here is how you can visualize the distribution of each column in the data:

klib.dist_plot(df_train) # returns a distribution plot for every numeric feature

klib.missingval_plot(df_train) # returns a figure containing information about missing values

# 4) Data cleaning using KLIB library

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset. But it is crucial to establish a template for your data cleaning process so you know you are doing it the right way every time.

With this insight, we can go ahead and start cleaning the data. With klib this is as simple as calling *klib.data_cleaning()*, which performs the following operations:

- cleaning the column names:
  This unifies the column names by formatting them, splitting, among others, CamelCase into camel_case, removing special characters as well as leading and trailing white-spaces and formatting all column names to *lowercase_and_underscore_separated*. This also checks for and fixes duplicate column names, which you sometimes get when reading data from a file.
- dropping empty and virtually empty columns:
  You can use the parameters *drop_threshold_cols* and *drop_threshold_rows* to adjust the dropping to your needs. The default is to drop columns and rows with more than 90% of the values missing.
- removes single valued columns:
  As the name states, this removes columns in which each cell contains the same value. This comes in handy when columns such as “year” are included while you’re just looking at a single year. Other examples are “download_date” or indicator variables which are identical for all entries.
- drops duplicate rows:
  This is a straightforward drop of entirely duplicate rows. If you are dealing with data where duplicates add value, consider setting *drop_duplicates=False.*
- Lastly, and often times most importantly, especially for memory reduction and therefore for speeding up the subsequent steps in your workflow, *klib.data_cleaning()* also optimizes the datatypes

## klib.data_cleaning(df_train) # performs datacleaning (drop duplicates & empty rows/cols, adjust dtypes,...)

In [37]:
```
# klib.clean - functions for cleaning datasets
klib.data_cleaning(df_train) # performs datacleaning (drop duplicates & empty rows/cols, adjust dtypes,...)
```

```
Shape of cleaned data: (8523, 10) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.46 MB (-70.77%)
```

Out[37]:

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type | outlet_type | item_outlet_ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9.300000 | Low Fat | 0.016047 | Dairy | 249.809204 | 1999 | Medium | Tier 1 | Supermarket Type1 | 3735.1 |
| 1 | 5.920000 | Regular | 0.019278 | Soft Drinks | 48.269199 | 2009 | Medium | Tier 3 | Supermarket Type2 | 443.4 |
| 2 | 17.500000 | Low Fat | 0.016760 | Meat | 141.617996 | 1999 | Medium | Tier 1 | Supermarket Type1 | 2097.2 |
| 3 | 19.200001 | Regular | 0.000000 | Fruits and Vegetables | 182.095001 | 1998 | Medium | Tier 3 | Grocery Store | 732.3 |
| 4 | 8.930000 | Low Fat | 0.000000 | Household | 53.861401 | 1987 | High | Tier 3 | Supermarket Type1 | 994.7 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 8518 | 6.865000 | Low Fat | 0.056783 | Snack Foods | 214.521805 | 1987 | High | Tier 3 | Supermarket Type1 | 2778.3 |
| 8519 | 8.380000 | Regular | 0.046982 | Baking Goods | 108.156998 | 2002 | Medium | Tier 2 | Supermarket Type1 | 549.2 |
| 8520 | 10.600000 | Low Fat | 0.035186 | Health and Hygiene | 85.122398 | 2004 | Small | Tier 2 | Supermarket Type1 | 1193.1 |
| 8521 | 7.210000 | Regular | 0.145221 | Snack Foods | 103.133202 | 2009 | Medium | Tier 3 | Supermarket Type2 | 1845.5 |
| 8522 | 14.800000 | Low Fat | 0.044878 | Soft Drinks | 75.467003 | 1997 | Small | Tier 1 | Supermarket Type1 | 765.6 |

8523 rows × 10 columns

## klib.clean_column_names(df_train) # cleans and standardizes column names, also called inside data_cleaning()

In [38]:
```
klib.clean_column_names(df_train) # cleans and standardizes column names, also called inside data_cleaning()
```

Out[38]:

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type | outlet_type | item_outlet_s |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9.300 | Low Fat | 0.016047 | Dairy | 249.8092 | 1999 | Medium | Tier 1 | Supermarket Type1 | 3735.1 |
| 1 | 5.920 | Regular | 0.019278 | Soft Drinks | 48.2692 | 2009 | Medium | Tier 3 | Supermarket Type2 | 443.4 |
| 2 | 17.500 | Low Fat | 0.016760 | Meat | 141.6180 | 1999 | Medium | Tier 1 | Supermarket Type1 | 2097.2 |
| 3 | 19.200 | Regular | 0.000000 | Fruits and Vegetables | 182.0950 | 1998 | Medium | Tier 3 | Grocery Store | 732.3 |
| 4 | 8.930 | Low Fat | 0.000000 | Household | 53.8614 | 1987 | High | Tier 3 | Supermarket Type1 | 994.7 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 8518 | 6.865 | Low Fat | 0.056783 | Snack Foods | 214.5218 | 1987 | High | Tier 3 | Supermarket Type1 | 2778.3 |
| 8519 | 8.380 | Regular | 0.046982 | Baking Goods | 108.1570 | 2002 | Medium | Tier 2 | Supermarket Type1 | 549.2 |
| 8520 | 10.600 | Low Fat | 0.035186 | Health and Hygiene | 85.1224 | 2004 | Small | Tier 2 | Supermarket Type1 | 1193.1 |
| 8521 | 7.210 | Regular | 0.145221 | Snack Foods | 103.1332 | 2009 | Medium | Tier 3 | Supermarket Type2 | 1845.5 |
| 8522 | 14.800 | Low Fat | 0.044878 | Soft Drinks | 75.4670 | 1997 | Small | Tier 1 | Supermarket Type1 | 765.6 |

8523 rows × 10 columns

df_train=klib.convert_datatypes(df_train) # converts existing to more efficient dtypes, also called inside data_cleaning()
df_train.info()

```
In [40]: df_train=klib.convert_datatypes(df_train) # converts existing to more efficient dtypes, also called inside data_cleaning()
         df_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   item_weight                8523 non-null   float32
 1   item_fat_content           8523 non-null   category
 2   item_visibility            8523 non-null   float32
 3   item_type                  8523 non-null   category
 4   item_mrp                   8523 non-null   float32
 5   outlet_establishment_year  8523 non-null   int16
 6   outlet_size                8523 non-null   category
 7   outlet_location_type       8523 non-null   category
 8   outlet_type                8523 non-null   category
 9   item_outlet_sales          8523 non-null   float32
dtypes: category(5), float32(4), int16(1)
memory usage: 192.9 KB
```

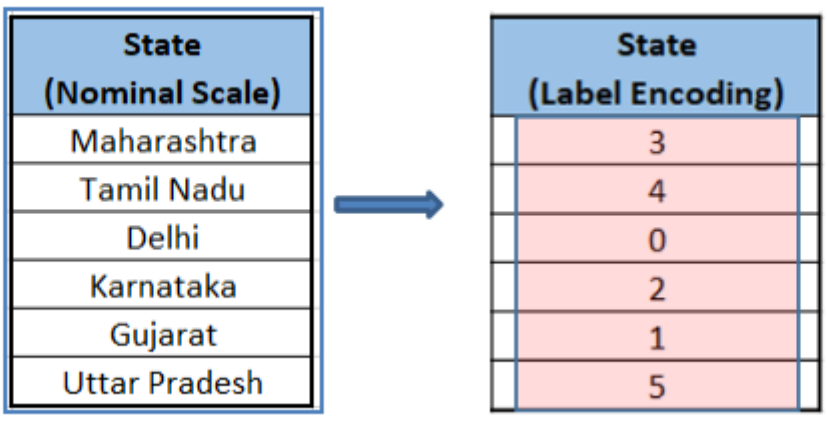
# 5) Pre Processing task

## Label Encoding

n label encoding in Python, we replace the categorical value with a numeric value between **0 and the number of classes minus 1.** If the categorical variable value contains 5 distinct classes, we use (0, 1, 2, 3, and 4).

To understand label encoding with an example, let us take COVID-19 cases in India across states. If we observe the below data frame, the State column contains a categorical value that is not very machine-friendly and the rest of the columns contain a numerical value. Let us perform Label encoding for State Column.

From the below image, after label encoding, the numeric value is assigned to each of the categorical values. You might be wondering why the numbering is not in sequence (Top-Down), and the answer is that the numbering is assigned in alphabetical order. Delhi is assigned 0 followed by Gujarat as 1 and so on.



As Label Encoding in Python is part of data preprocessing, hence we will take an help of preprocessing module from sklearn package and import LabelEncoder class as below:

And then:

1. Create an instance of LabelEncoder() and store it in labelencoder variable/object
2. Apply fit and transform which does the trick to assign numerical value to categorical value and the same is stored in new column called “State_N”
3. Note that we have added a new column called “State_N” which contains numerical value associated to categorical value and still the column called State is present in the dataframe. This column needs to be removed before we feed the final preprocess data to machine learning model to learn

```
In [42]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
```

```
In [43]: df_train['item_fat_content']= le.fit_transform(df_train['item_fat_content'])
         df_train['item_type']= le.fit_transform(df_train['item_type'])
         df_train['outlet_size']= le.fit_transform(df_train['outlet_size'])
         df_train['outlet_location_type']= le.fit_transform(df_train['outlet_location_type'])
         df_train['outlet_type']= le.fit_transform(df_train['outlet_type'])
```

```
In [44]: df_train.head(5)
```

Out[44]:

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type | outlet_type | item_outlet_sales |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9.300000 | 1 | 0.016047 | 4 | 249.809204 | 1999 | 1 | 0 | 1 | 3735.137939 |
| 1 | 5.920000 | 2 | 0.019278 | 14 | 48.269199 | 2009 | 1 | 2 | 2 | 443.422791 |
| 2 | 17.500000 | 1 | 0.016760 | 10 | 141.617996 | 1999 | 1 | 0 | 1 | 2097.270020 |
| 3 | 19.200001 | 2 | 0.000000 | 6 | 182.095001 | 1998 | 1 | 2 | 0 | 732.380005 |
| 4 | 8.930000 | 1 | 0.000000 | 9 | 53.861401 | 1987 | 0 | 2 | 1 | 994.705200 |

# Splitting the data into train and test

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

It can be used for classification or regression problems and can be used for any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values.

The train-test procedure is appropriate when there is a sufficiently large dataset available.

The scikit-learn Python machine learning library provides an implementation of the train-test split evaluation procedure via the train_test_split() function.

The function takes a loaded dataset as input and returns the dataset split into two subsets.

```
1...
2# split into train test sets
3train, test = train_test_split(dataset, ...)
```

Ideally, you can split your original dataset into input (*X*) and output (*y*) columns, then call the function passing both arrays and have them split appropriately into train and test subsets.

```
1...
2# split into train test sets
3X_train, X_test, y_train, y_test = train_test_split(X, y, ...)
```

The size of the split can be specified via the "*test_size*" argument that takes a number of rows (integer) or a percentage (float) of the size of the dataset between 0 and 1.

The latter is the most common, with values used such as 0.33 where 33 percent of the dataset will be allocated to the test set and 67 percent will be allocated to the training set.

```
1...
2# split into train test sets
3X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

We can demonstrate this using a synthetic classification dataset with 1,000 examples.

The complete example is listed below.

```
1# split a dataset into train and test sets
2from sklearn.datasets import make_blobs
3from sklearn.model_selection import train_test_split
4# create dataset
5X, y = make_blobs(n_samples=1000)
6# split into train test sets
7X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
8print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Running the example splits the dataset into train and test sets, then prints the size of the new dataset.

We can see that 670 examples (67 percent) were allocated to the training set and 330 examples (33 percent) were allocated to the test set, as we specified.

```
1(670, 2) (330, 2) (670,) (330,)
```

Alternatively, the dataset can be split by specifying the "*train_size*" argument that can be either a number of rows (integer) or a percentage of the original dataset between 0 and 1, such as 0.67 for 67 percent.

```
1...
2# split into train test sets
3X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.67)
```

```
In [45]: X=df_train.drop('item_outlet_sales',axis=1)

In [46]: Y=df_train['item_outlet_sales']

In [47]: from sklearn.model_selection import train_test_split

         X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=101, test_size=0.2)
```

# Standardization

Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

For instance, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the l1 and l2 regularizers of linear models) may assume that all features are centered around zero or have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

One solution to this issue is standardization. Consider columns as variables. If a column is standardized, mean value of the column is subtracted from each value and then the values are divided by the standard deviation of the column. The resulting columns have a standard deviation of 1 and a mean that is very close to zero. Thus, we end up having variables (columns) that have almost a normal distribution. Standardization can be achieved by StandardScaler.

The functions and transformers used during preprocessing are in sklearn.preprocessing package. Let's import this package along with numpy and pandas.

```
import numpy as np
import pandas as pdfrom sklearn import preprocessing
```

We can create a sample matrix representing features. Then transform it using a StandardScaler object.

```
a = np.random.randint(10, size=(10,1))
b = np.random.randint(50, 100, size=(10,1))
c = np.random.randint(500, 700, size=(10,1))X = np.concatenate((a,b,c), axis=1)
X
```

```
array([[  1,  53, 605],
       [  8,  57, 582],
       [  4,  99, 524],
       [  9,  66, 576],
       [  5,  51, 680],
       [  3,  54, 622],
       [  1,  83, 574],
       [  4,  74, 509],
       [  5,  74, 617],
       [  4,  70, 629]])
```

X represents the values in a dataframe with 3 columns and 10 rows. Columns represent features. The mean and standard deviation of each column:

```
X.mean(axis=0)
```

```
array([  4.4,  68.1, 591.8])
```

```
X.std(axis=0)
```

```
array([ 2.45764115, 14.48067678, 47.9578982 ])
```

The columns are highly different in terms of mean and standard deviation.

We can now create a StandardScaler object and fit X to it.

```
sc = preprocessing.StandardScaler().fit(X)
```

X can be transformed by applying transform method on StandardScaler object.

```
X_standardized = sc.transform(X)
X_standardized
```

```
array([[-1.38344038, -1.04276894,  0.27524142],
       [ 1.46481923, -0.76653876, -0.2043459 ],
       [-0.16275769,  2.13387817, -1.41374002],
       [ 1.87171346, -0.14502085, -0.32945564],
       [ 0.24413654, -1.18088403,  1.83911312],
       [-0.56965192, -0.9737114 ,  0.62971901],
       [-1.38344038,  1.02895743, -0.37115888],
       [-0.16275769,  0.40743952, -1.72651436],
       [ 0.24413654,  0.40743952,  0.52546089],
       [-0.16275769,  0.13120934,  0.77568037]])
```

Let's calculate the mean and standard deviation of transformed features.

```
X_standardized.mean(axis=0)
```

```
array([-1.60982339e-16,  3.44169138e-16,  9.65894031e-16])
```

```
X_standardized.std(axis=0)
```

```
array([1., 1., 1.])
```

Mean of each feature is very close to 0 and all the features have unit (1) variance. Please note that standard deviation is the square root of variance. A standard deviation of 1 indicates variance is 1.

I would like to emphasize a very important point here. Consider we are working on a supervised learning task so we split the dataset into training and test subsets. In that case, we only fit training set to the standard scaler object, not the entire dataset. We, of course, need to transform the test set but it is done with transform method.

- StandardScaler.fit(X_train)
- StandardScaler.transform(X_train)

- StandardScaler.transform(X_test)

Fitting the entire dataset to the standard scaler object causes the model to learn about test set. However, models are not supposed to learn anything about test set. It destroys the purpose of train-test split. In general, this issue is called data leakage.

Data Leakage in Machine Learning
How to detect and avoid data leakage
towardsdatascience.com

When we transform the test set, the features will not have exactly zero mean and unit standard deviation because the scaler used in transformation is based on the training set. The amount of change in the test set is the same as in the training set. Let's create a sample test set and transform it.

```
X_test = np.array([[8, 90, 650], [5, 70, 590], [7, 80, 580]])
X_test
```

```
array([[  8,  90, 650],
       [  5,  70, 590],
       [  7,  80, 580]])
```

```
X_test_transformed = sc.transform(X_test)
X_test_transformed
```

```
array([[ 1.46481923,  1.51236025,  1.21356444],
       [ 0.24413654,  0.13120934, -0.03753292],
       [ 1.057925  ,  0.8217848 , -0.24604915]])
```

The mean and standard deviation of the columns of test set:

```
X_test_transformed.mean(axis=0)
```

```
array([0.92229359, 0.8217848 , 0.30999412])
```

```
X_test_transformed.std(axis=0)
```

```
array([0.50748627, 0.5638525 , 0.64456665])
```

# 6) Model Building

## 1) Linear regression

It can be referred to as a parametric ML technique which is used to predict a continuous or dependent variable since a dataset of independent variables is provided. This technique is said to be parametric as different predictions are made based on the dataset.

```
In [60]: from sklearn.linear_model import LinearRegression
         lr= LinearRegression()

In [61]: lr.fit(X_train_std,Y_train)

Out[61]: LinearRegression
         LinearRegression()

In [62]: Y_pred_lr=lr.predict(X_test_std)

In [63]: r2_score(Y_test,Y_pred_lr)

Out[63]: 0.5041875773270634

In [64]: print(r2_score(Y_test,Y_pred_lr))
         print(mean_absolute_error(Y_test,Y_pred_lr))
         print(np.sqrt(mean_squared_error(Y_test,Y_pred_lr)))

         0.5041875773270634
         880.99990440845
         1162.4412631603452
```

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Before attempting to fit a linear model to observed data, a modeler should first determine whether or not there is a relationship between the variables of interest. This does not necessarily imply that one variable *causes* the other (for example, higher SAT scores do not *cause* higher college grades), but that there is some significant association between the two variables. A scatterplot can be a helpful tool in determining the strength of the relationship between two variables. If there appears to be no association between the proposed explanatory and dependent variables (i.e., the scatterplot does not indicate any increasing or decreasing trends), then fitting a linear regression model to the data probably will not provide a useful model. A valuable numerical measure of association between two variables is the correlation coefficient, which is a value between -1 and 1 indicating the strength of the association of the observed data for the two variables.

A linear regression line has an equation of the form $\boldsymbol{Y = a + bX}$, where $\boldsymbol{X}$ is the explanatory variable and $\boldsymbol{Y}$ is the dependent variable. The slope of the line is $\boldsymbol{b}$, and $\boldsymbol{a}$ is the intercept (the value of $\boldsymbol{y}$ when $\boldsymbol{x} = 0$).

## 2) Random forest regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

```
In [66]: from sklearn.ensemble import RandomForestRegressor
         rf= RandomForestRegressor()

In [67]: rf.fit(X_train_std,Y_train)

Out[67]: RandomForestRegressor
         RandomForestRegressor()

In [68]: Y_pred_rf= rf.predict(X_test_std)

In [69]: r2_score(Y_test,Y_pred_rf)

Out[69]: 0.5498450972136788

In [70]: print(r2_score(Y_test,Y_pred_rf))
         print(mean_absolute_error(Y_test,Y_pred_rf))
         print(np.sqrt(mean_squared_error(Y_test,Y_pred_rf)))

         0.5498450972136788
         779.8420976075874
         1107.6264243242706
```

We will use the sklearn module for training our random forest regression model, specifically the RandomForestRegressor function. The RandomForestRegressor documentation shows many different parameters we can select for our model. Some of the important parameters are highlighted below:

- n_estimators — the number of decision trees you will be running in the model
- criterion — this variable allows you to select the criterion (loss function) used to determine model outcomes. We can select from loss functions such as mean squared error (MSE) and mean absolute error (MAE). The default value is MSE.
- max_depth — this sets the maximum possible depth of each tree
- max_features — the maximum number of features the model will consider when determining a split
- bootstrap — the default value for this is True, meaning the model follows bootstrapping principles (defined earlier)
- max_samples — This parameter assumes bootstrapping is set to True, if not, this parameter doesn't apply. In the case of True, this value sets the largest size of each sample for each tree.
- Other important parameters are min_samples_split, min_samples_leaf, n_jobs, and others that can be read in the sklearn's

# 3) XG boost regressor

Before running XGBoost, we must set three types of parameters: general parameters, booster parameters and task parameters.

- **General parameters** relate to which booster we are using to do boosting, commonly tree or linear model
- **Booster parameters** depend on which booster you have chosen
- **Learning task parameters** decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks.
- **Command line parameters** relate to behavior of CLI version of XGBoost.

```
In [72]: from xgboost import XGBRegressor
         xg= XGBRegressor()

In [73]: xg.fit(X_train_std, Y_train)

Out[73]: XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)

In [74]: Y_pred_xg= xg.predict(X_test_std)

In [75]: r2_score(Y_test,Y_pred_xg)

Out[75]: 0.5313160637898305

In [76]: print(r2_score(Y_test,Y_pred_xg))
         print(mean_absolute_error(Y_test,Y_pred_xg))
         print(np.sqrt(mean_squared_error(Y_test,Y_pred_xg)))

         0.5313160637898305
         800.45557
         1130.1923
```

The results of the regression problems are continuous or real values. Some commonly used regression algorithms are Linear Regression and Decision Trees. There are several metrics involved in regression like root-mean-squared error (RMSE) and mean-squared-error (MAE). These are some key members of XGBoost models, each plays an important role.

- RMSE: It is the square root of mean squared error (MSE).
- MAE: It is an absolute sum of actual and predicted differences, but it lacks mathematically, that's why it is rarely used, as compared to other metrics.

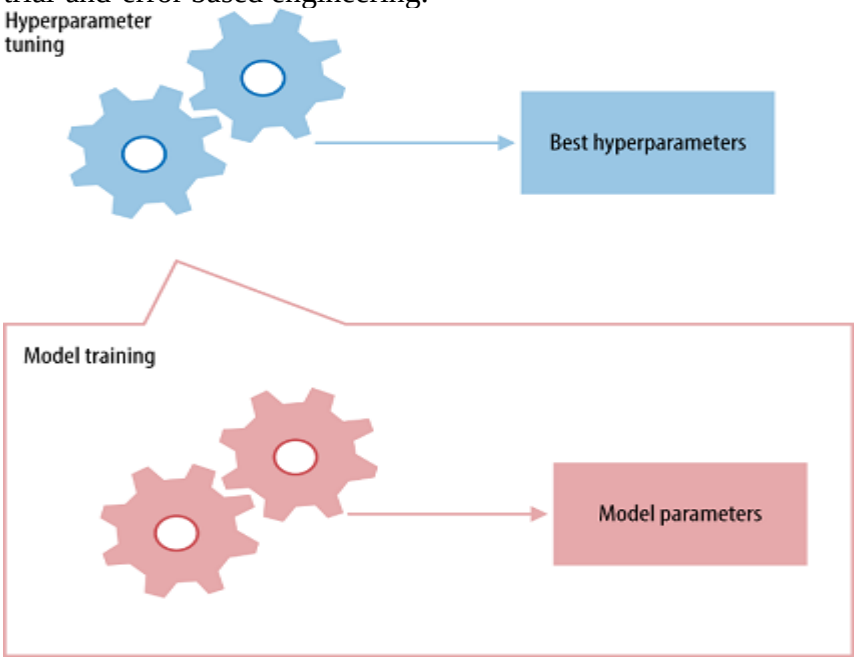
XGBoost is a powerful approach for building supervised regression models. The validity of this statement can be inferred by knowing about its (XGBoost) objective function and base learners. The objective function

contains loss function and a regularization term. It tells about the difference between actual values and predicted values, i.e how far the model results are from the real values. The most common loss functions in XGBoost for regression problems is reg:linear, and that for binary classification is reg:logistics. Ensemble learning involves training and combining individual models (known as base learners) to get a single prediction, and XGBoost is one of the ensemble learning methods. XGBoost expects to have the base learners which are uniformly bad at the remainder so that when all the predictions are combined, bad predictions cancels out and better one sums up to form final good predictions

# 7) Hyper Parameter tuning

Grid search is the simplest algorithm for hyperparameter tuning. Basically, we divide the domain of the hyperparameters into a discrete grid. Then, we try every combination of values of this grid, calculating some performance metrics using cross-validation. The point of the grid that maximizes the average value in cross-validation, is the optimal combination of values for the hyperparameters.

The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as we might turn the knobs of an AM radio to get a clear signal (or your parents might have!). While model *parameters* are learned during training — such as the slope and intercept in a linear regression — *hyperparameters* must be set by the data scientist before training. In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. (The parameters of a random forest are the variables and thresholds used to split each node learned during training). Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error based engineering.



Hyperparameters and Parameters

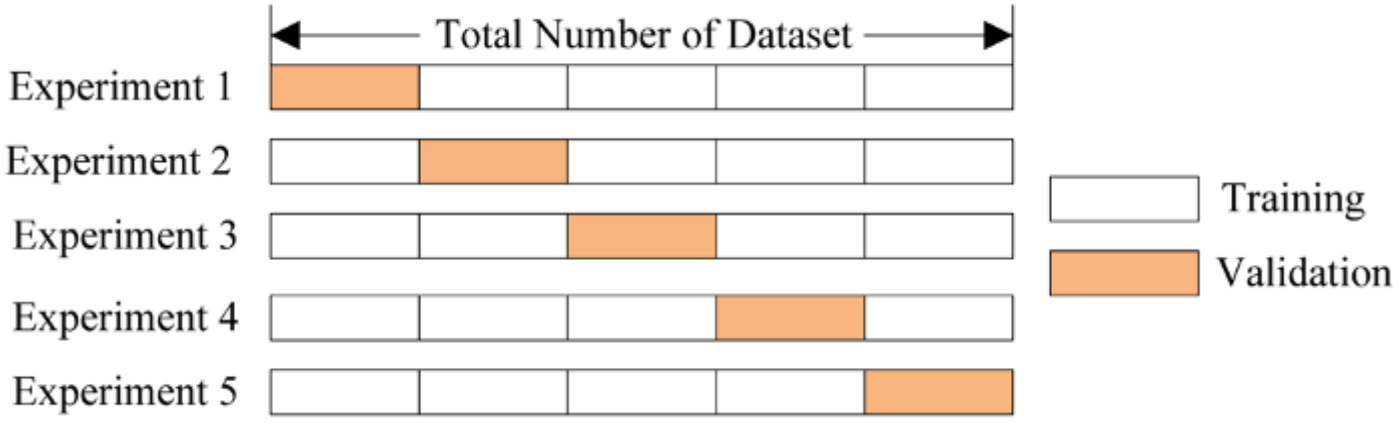
Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most fundamental problems in machine learning: overfitting.

If we optimize the model for the training data, then our model will score very well on the training set, but will not be able to generalize to new data, such as in a test set. When a model performs highly on the training set but poorly on the test set, this is known as overfitting, or essentially creating a model that knows the training set very well but cannot be applied to new problems. It's like a student who has memorized the simple problems in the textbook but has no idea how to apply concepts in the messy real world.

An overfit model may look impressive on the training set, but will be useless in a real application. Therefore, the standard procedure for hyperparameter optimization accounts for overfitting through cross validation.

Cross Validation

The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data). As an example, consider fitting a model with K = 5. The first iteration we train on the first four folds and evaluate on the fifth. The second time we train on the first, second, third, and fifth fold and evaluate on the fourth. We repeat this procedure 3 more times, each time evaluating on a different fold. At the very end of training, we average the performance on each of the folds to come up with final validation metrics for the model.




5 Fold Cross Validation (Source)

For hyperparameter tuning, we perform many iterations of the entire K-Fold CV process, each time using different model settings. We then compare all of the models, select the best one, train it on the full training set, and then evaluate on the testing set. This sounds like an awfully tedious process! Each time we want to assess a different set of hyperparameters, we have to split our training data into K fold and train and evaluate K times. If we have 10 sets of hyperparameters and are using 5-Fold CV, that represents 50 training loops. Fortunately, as with most problems in machine learning, someone has solved our problem and model tuning with K-Fold CV can be automatically implemented in Scikit-Learn.

Random Search Cross Validation in Scikit-Learn

Usually, we only have a vague idea of the best hyperparameters and thus the best approach to narrow our search is to evaluate a wide range of values for each hyperparameter. Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold CV with each combination of values.

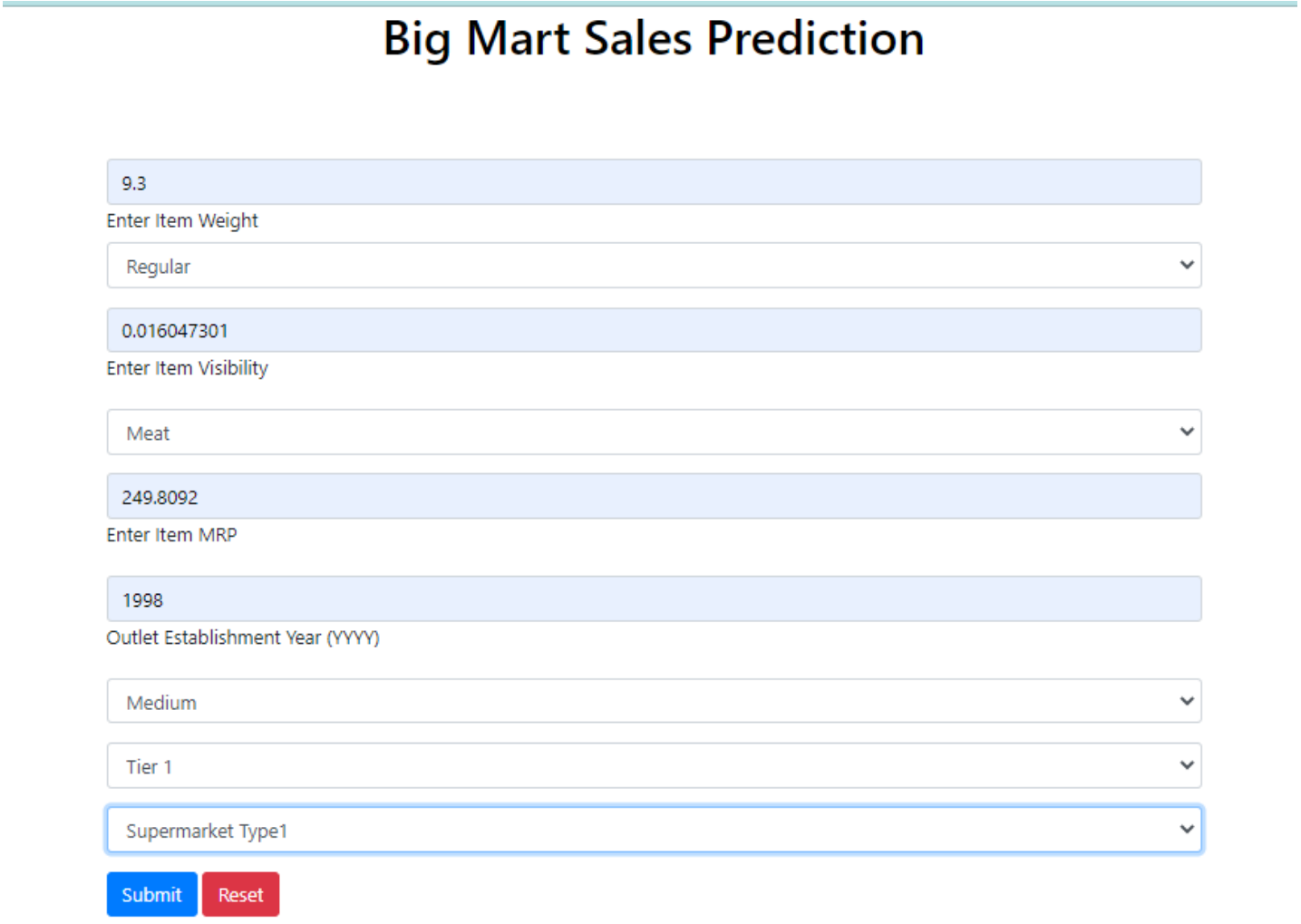
In [84]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

# define models and parameters
model = RandomForestRegressor()
n_estimators = [10, 100, 1000]
max_depth=range(1,31)
min_samples_leaf=np.linspace(0.1, 1.0)
max_features=["auto", "sqrt", "log2"]
min_samples_split=np.linspace(0.1, 1.0, 10)

# define grid search
grid = dict(n_estimators=n_estimators)
grid_search_forest = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                        scoring='r2',error_score=0,verbose=2,cv=2)
grid_search_forest.fit(X_train_std, Y_train)

# summarize results
print(f"Best: {grid_search_forest.best_score_:.3f} using {grid_search_forest.best_params_}")
means = grid_search_forest.cv_results_['mean_test_score']
stds = grid_search_forest.cv_results_['std_test_score']
params = grid_search_forest.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
 print(f"{mean:.3f} ({stdev:.3f}) with: {param}")
```

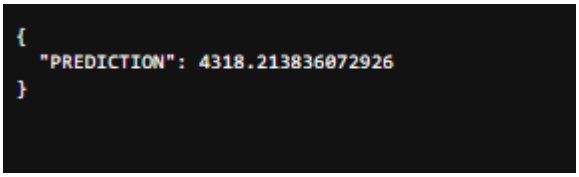
```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best: 0.550 using {'n_estimators': 100}
0.512 (0.010) with: {'n_estimators': 10}
0.550 (0.006) with: {'n_estimators': 100}
0.550 (0.006) with: {'n_estimators': 1000}
```

Random search allowed us to narrow down the range for each hyperparameter. Now that we know where to concentrate our search, we can explicitly specify every combination of settings to try. We do this with GridSearchCV, a method that, instead of sampling randomly from a distribution, evaluates all combinations we define.

# 8) Deployment and Result

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.




## Done !

# Conclusion

The objective of this framework is to predict the future sales from given data of the previous year's using machine Learning techniques. In this paper, we have discussed how different machine learning models are built using different algorithms like Linear regression, Random forest regressor, and XG booster algorithms. These algorithms have been applied to predict the final result of sales. We have addressed in detail about how the noisy data is been removed and the algorithms used to predict the result. Based on the accuracy predicted by different models we conclude that the random forest approach is the best models. Our predictions help big marts to refine their methodologies and strategies which in turn helps them to increase their profit.

# Reference


[1]"Applied Linear Statistical Models", Fifth Edition by Kutner, Nachtsheim, Neter and L, Mc Graw Hill India, 2013.

[2]Demchenko, Yuri & de Laat, Cees & MembreyPeter,"Defining architecture components of the Big Data Ecosystem", 2014.

[3] Blog: Big Sky, "The Data Analysis Process: 5 Steps To Better Decision Making", (URL: https://www.bigskyassociates.com/blog/bid/372186/The-Data- Analysis-Process-5-Steps-To-Better-Decision Making).

[4] Blog: Dataaspirant, "HOW THE RANDOM FOREST ALGORITHM WORKS IN MACHINE LEARNING", (URL: https://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/.).

[5] Mohit Gurnani, Yogesh Korke, Prachi Shah, Sandeep Udmale, Vijay Sambhe, Sunil Bhirud, "Forecasting of sales by using fusion of machine learning techniques",2017 International Conference on Data Management, Analytics and Innovation (ICDMAI), IEEE, October 2017.

[6] Armstrong J,"Sales Forecasting", SSRN Electronic Journal, July 2008.

[7] Samaneh Beheshti-Kashi, Hamid Reza Karimi, Klaus-Dieter Thoben, Michael Lütjen, "A survey on retail sales forecasting and prediction in fashion markets", Systems Science & Control Engineering: An Open Access Journal. 3. 154-161. 10.1080/21642583.2014.999389.

[8]Gopal Behera, Neeta Nain, "A Comparative Study of Big Mart Sales Prediction", 4th International Conference on Computer Vision and Image Processing, At MNIT Jaipur, September 2019.

# Acknowledgements

I am profoundly grateful to Prof. Vishal Patil for his expert guidance and continuous encouragement throughout to see that this project rights its target.

I would like to express deepest appreciation towards Dr. Varsha Shah, Principal RCOE, Mumbai and Prof. Vishal Patil HOD Prof. Shiburaj Pappu Department whose invaluable guidance supported me in this project.

At last I must express my sincere heartfelt gratitude to all the staff members of Computer Engineering Department who helped us directly or indirectly during this course of work.

~Utsav Kuntalwad

Srushti Sawant

Mayur Kyatham

Prerna Shakwar