PROJECT REPORT

"Eveready Products Sales Analysis

by using

Machine Learning

&

Deep Learning Models"

Submitted towards the partial fulfilment of the criteria for award of Post Graduate Program in Data Analytics

Submitted By:

G.VIGNESHWARAN

Course and Batch: PGA-04-APRIL-2021

IMARTICUS LEARNING

Abstract

This project comes under Retail Sales Domain. In this project I am going to take Eveready Industries India Ltd sales dataset consist of Organization Code, Region, Town, Month, Sale Type, Payment Term, Customer Name, Group code, Costs section code, Free, Quantity, all over India sales data. As the dealers, distributers or sales person would overcome the problem of how much quantity of sales will happen for each product for the upcoming months, this project will help to over this problem.

Acknowledgements

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guided us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the PGA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: May, 2021 G.Vigneshwaran

Place: Coimbatore

Certificate of Completion

I hereby certify that the project titled "**Eveready Products Sales Analysis by using Machine learning & Deep learning Models"** was undertaken and completed under my supervision by Thulasidass from the batch of PGA 04 COI

Date: May, 2021

Place:

Coimbatore

Table of Contents

CHAPTER 1
INTRODUCTION

1.1Title & Objective of the study

***Eveready Products Sales Analysis by using Machine Learning & Deep Learning Models.*,**

In this project have used various machine learning regression models like Logistic Regression, Gradient Boosting Regressor, Extreme Gradient Boosting Regressor, KNN Regressor, Random Forest Regressor & Artificial Neural Network to predict the sales quantity values. The quantity value prediction is important for many business and administrative decision makers especially in Marketing & Sales. To promote business, sales and buyer behavior can help business decision makers in Warehouse, marketing campaigns, budget and resource planning.

1.2 Need of the Study

In this project, the main purpose is to predict quantity of the sales for the month has happen and for knowledge of what quantity sales to happen in future, so that the dealers, distributers or sales person will overcome the problem of how much quantity of sales will happen for each product for the current month or upcoming months, In this project, we work on the simpler problem that is to predict the quantity of sales for the month using Machine Learning & Deep Learning Models finding out the best.

1.3 Business or Enterprise under study

Eveready Industries India Limited (EIIL) is one of India's leading consumer goods companies with its products and brands being household names over the past century. Over the decades it has been the leader in the dry cell batteries and flashlights markets in India. So, we took all consumer goods to quantity which sells every month across various bigger supermarkets and retail outlets alone.

1.4 Business Model of Enterprise

Selecting the relevant variables from the dataset and arranging their values in order of importance to creating models to predict the probability of quantity of sales of an individual by performing different types of algorithms on the data.

1.5 Data Sources

A dataset provided by the Eveready Industries India (Coimbatore sales division) will be utilized for analysis,

The data contains:

Po Number: Purchase order Number of the product.

Sale Type: Sales done by cash, credit or return to vendor.

Month: Sales done in a month.

Cost section code: Code for production identification.

Group code: Product code identification.

Type code: Subcode of product code

Product code: Separate code for Stock Knowledge Unit.

Description: Description for Stock Knowledge Unit.

Lot Number: Product manufactured in the same batch or contain a common material.

Customer Name: Buyer name & details.

Nsv: Net sale value of the product.

Unit Weight: Measurements of Stock Knowledge Unit.

Lp: Landing price of the product.

Payment Term: Number of credit days given or cash and carry.

Data Set Description:

Contains 13159 rows and 15 columns.

1.6 Tools & Techniques

Tools: Jupyter Notebook.

Techniques:

- Logistic Regression
- Gradient Boosting Regressor
- Extreme Gradient Boosting Regressor
- KNN Regressor
- Random Forest Regressor
- Artificial Neural Network

CHAPTER 2
DATA PREPARATION AND UNDERSTANDING

One of the first steps we engaged in was to outline the sequence of steps that we will be following for our project. Each of these steps are elaborated below
After importing the required libraries, a sequence of steps was followed to perform data preprocessing.

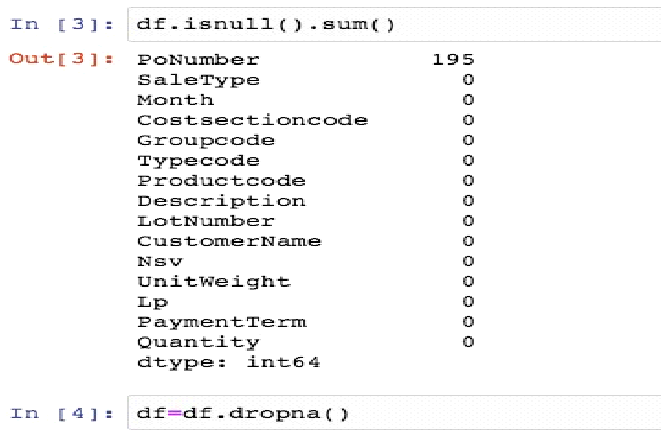
2.1 Feature Engineering

Feature engineering using **domain knowledge** of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process. As the Feature Engineering is an art

2.1.0. Exploratory Data Analysis (EDA)

We will explore our Data set and perform the exploratory data analysis.

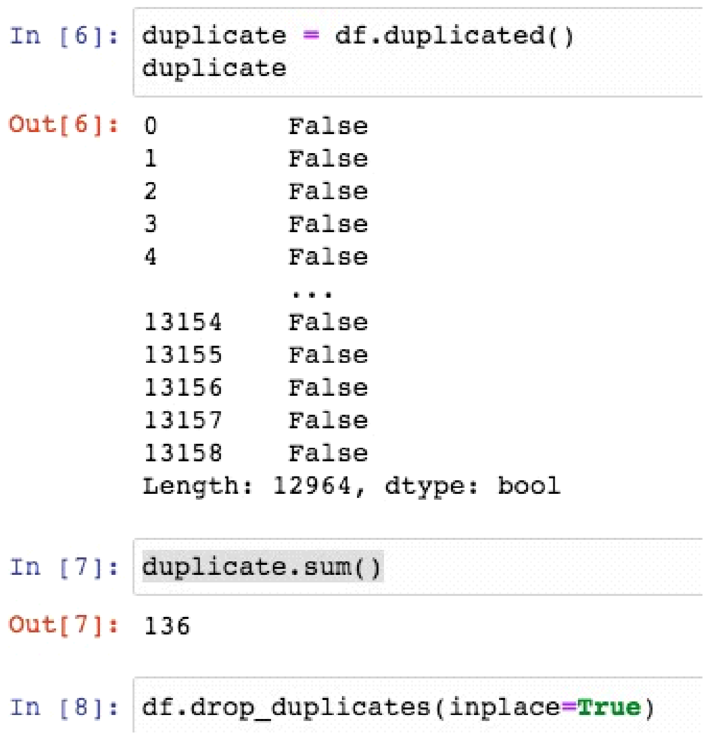
2.1.1. Handling missing value



```
In [3]: df.isnull().sum()

Out[3]: PoNumber             195
        SaleType               0
        Month                  0
        Costsectioncode        0
        Groupcode              0
        Typecode               0
        Productcode            0
        Description            0
        LotNumber              0
        CustomerName           0
        Nsv                    0
        UnitWeight             0
        Lp                     0
        PaymentTerm            0
        Quantity               0
        dtype: int64

In [4]: df=df.dropna()
```

We can see that we have various missing values in the respective columns. There are various ways of treating your missing values in the data set. And which technique to use when is actually dependent on the type of data you are dealing with.

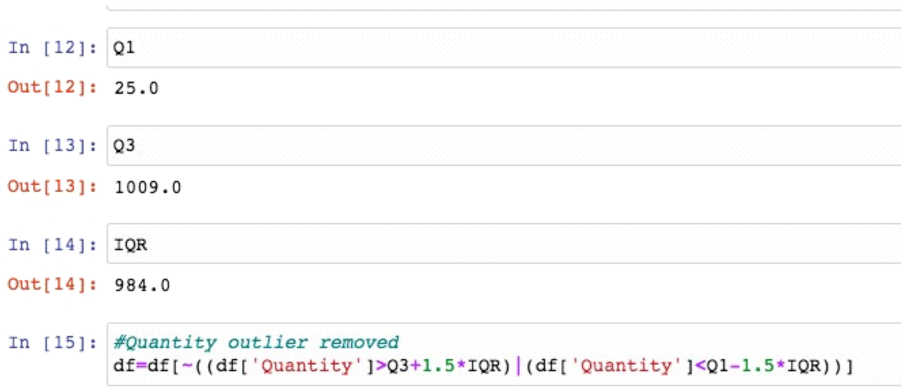
2.1.2. Handling Duplicate records



```
In [6]:  duplicate = df.duplicated()
         duplicate

Out[6]:  0        False
         1        False
         2        False
         3        False
         4        False
                  ...
         13154    False
         13155    False
         13156    False
         13157    False
         13158    False
         Length: 12964, dtype: bool

In [7]:  duplicate.sum()

Out[7]:  136

In [8]:  df.drop_duplicates(inplace=True)
```

Since we have 136 duplicate records in the data, we will remove this from the data set so that we get only distinct records. Post removing the duplicate, we will check whether the duplicates have been removed from the data set or not.

2.1.3. Handling Outlier

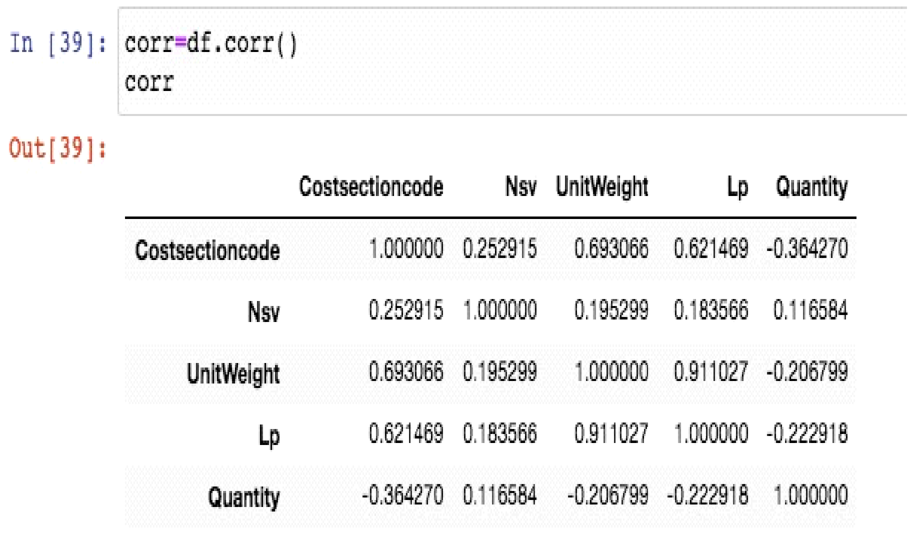


```
In [12]: Q1
Out[12]: 25.0

In [13]: Q3
Out[13]: 1009.0

In [14]: IQR
Out[14]: 984.0

In [15]: #Quantity outlier removed
         df=df[~((df['Quantity']>Q3+1.5*IQR)|(df['Quantity']<Q1-1.5*IQR))]
```

Outliers, being the most extreme observations, may include the sample maximum or sample minimum, or both, depending on whether they are extremely high or low. However, the sample maximum and minimum are not always outliers because they may not be unusually far from other observations.

2.1.4. Correlation

Correlation explains how one or more variables are related to each other. These variables can be input data features which have been used to forecast our target variable.

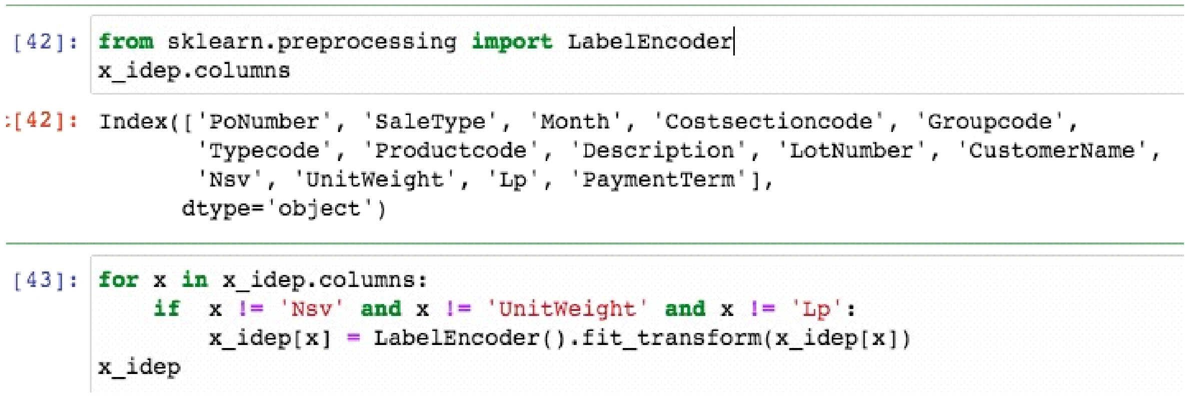


```
In [39]: corr=df.corr()
         corr
Out[39]:
```

	Costsectioncode	Nsv	UnitWeight	Lp	Quantity
Costsectioncode	1.000000	0.252915	0.693066	0.621469	-0.364270
Nsv	0.252915	1.000000	0.195299	0.183566	0.116584
UnitWeight	0.693066	0.195299	1.000000	0.911027	-0.206799
Lp	0.621469	0.183566	0.911027	1.000000	-0.222918
Quantity	-0.364270	0.116584	-0.206799	-0.222918	1.000000

2.1.5. Encoding

There are many ways to convert categorical values into numerical values. But we used Label-Encoder are used to convert text or categorical data into numerical data in which the model expects and perform better with.



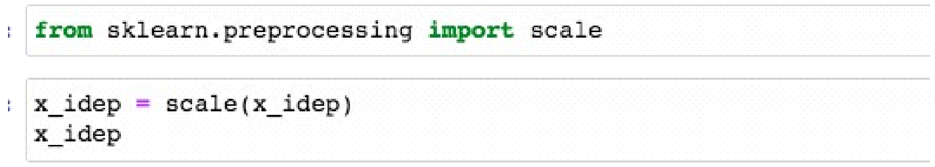
```
[42]: from sklearn.preprocessing import LabelEncoder
      x_idep.columns
```

```
:[42]: Index(['PoNumber', 'SaleType', 'Month', 'Costsectioncode', 'Groupcode',
              'Typecode', 'Productcode', 'Description', 'LotNumber', 'CustomerName',
              'Nsv', 'UnitWeight', 'Lp', 'PaymentTerm'],
             dtype='object')
```

```
[43]: for x in x_idep.columns:
          if  x != 'Nsv' and x != 'UnitWeight' and x != 'Lp':
              x_idep[x] = LabelEncoder().fit_transform(x_idep[x])
      x_idep
```

2.1.6. Normalizing and Scaling

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since the range of values of data may vary widely, it becomes a necessary step in data pre-processing while using machine learning algorithms.



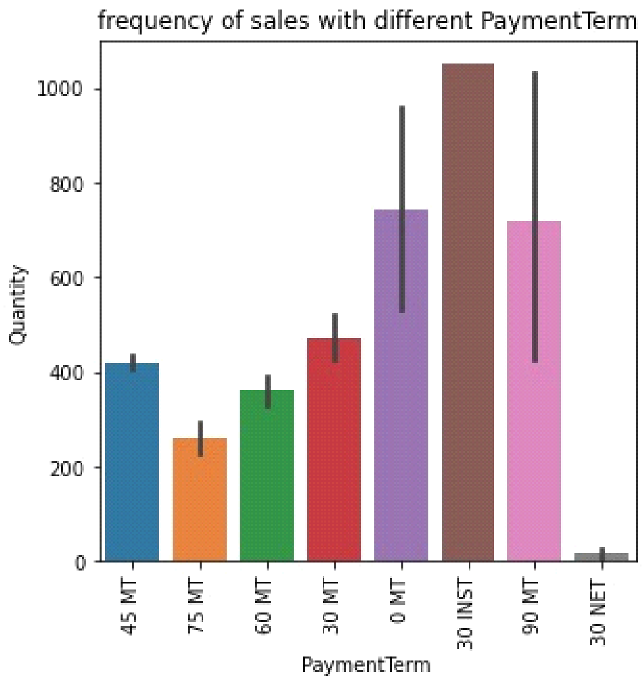
```
: from sklearn.preprocessing import scale
```

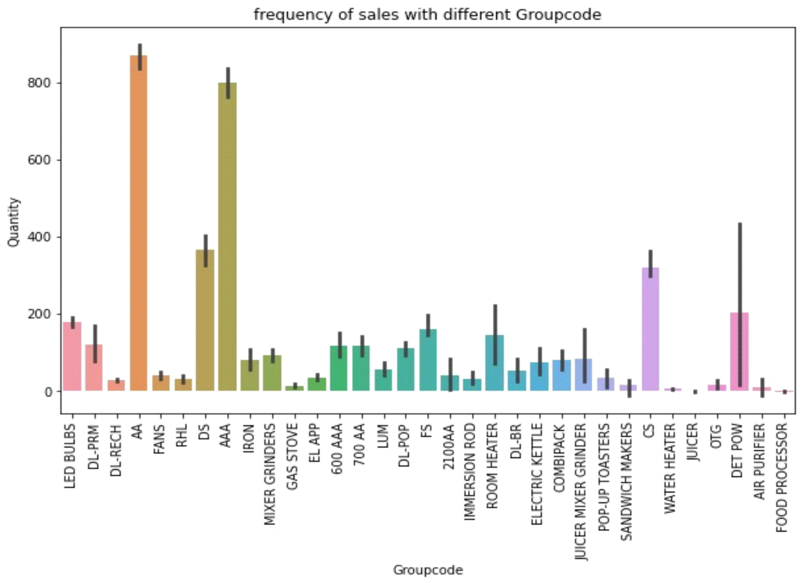
```
: x_idep = scale(x_idep)
  x_idep
```

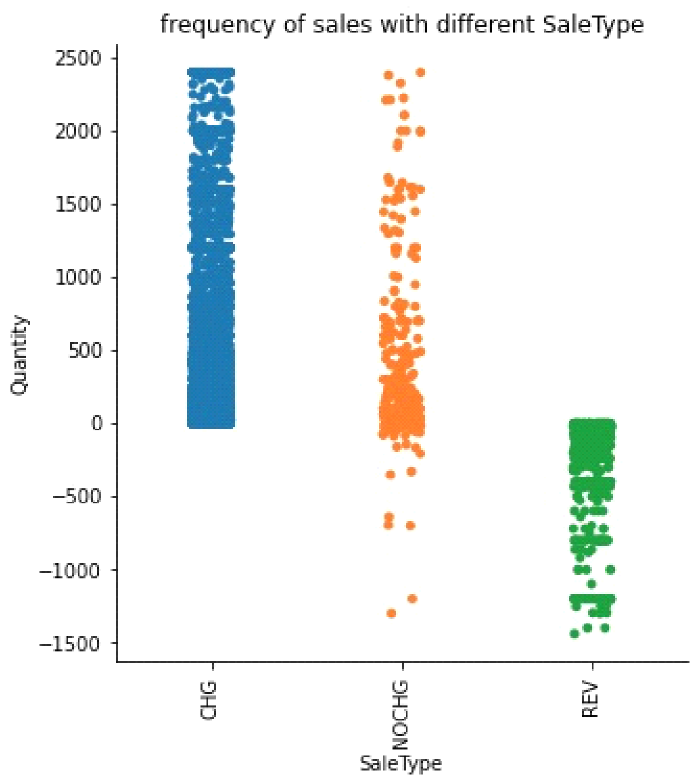
2.2 Data Visualization

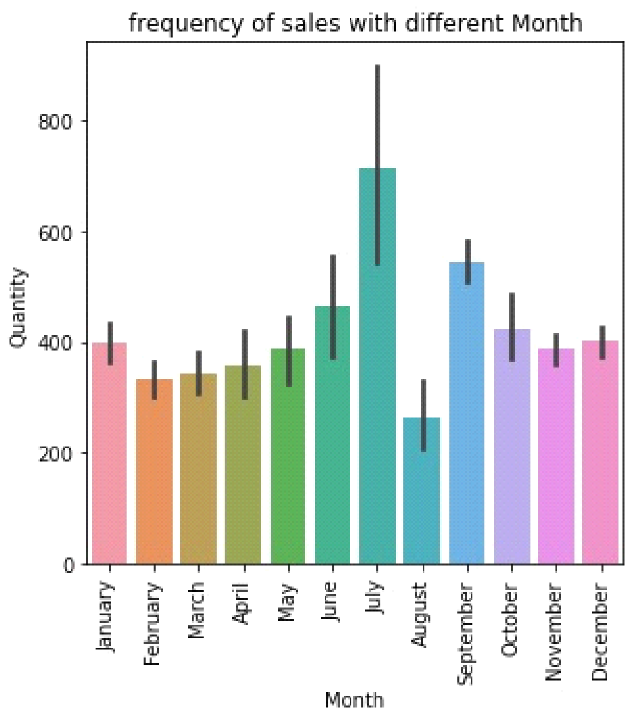
2.2.1. Bivariate Analysis

When we talk about bivariate analysis, it means analyzing 2 variables. Since we know there are numerical and categorical variables, there is a way of analyzing these variables as shown below:





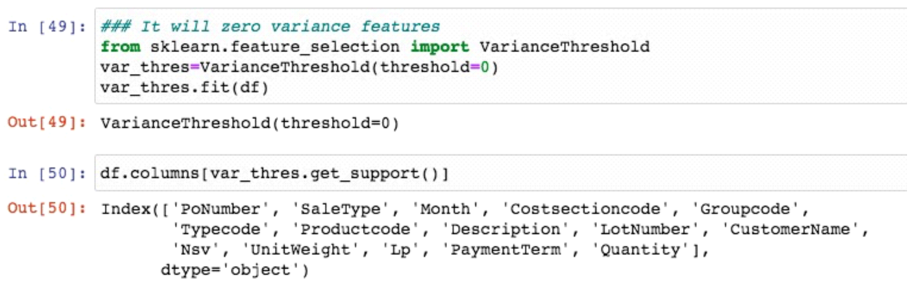

frequency of sales with different SaleType
Quantity
2500
2000
1500
1000
500
0
−500
−1000
−1500
CHG
NOCHG
REV
SaleType


frequency of sales with different Month
Quantity
800
600
400
200
0
January
February
March
April
May
June
July
August
September
October
November
December
Month

2.3 Feature Selection

2.3.1 Dropping Constant Features Using Variance Threshold

The variance threshold is a simple baseline approach to feature selection. It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e., features that have the same value in all samples. We assume that features with a higher variance may contain more useful information, but note that we are not taking the relationship between feature variables or feature and target variables into account, which is one of the drawbacks of filter methods.

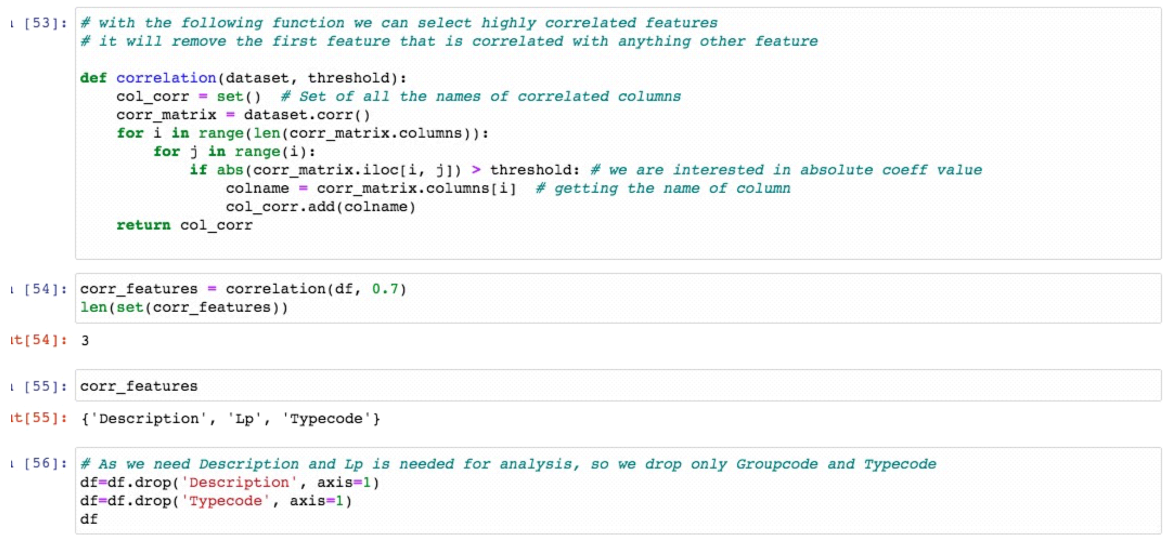


```
In [49]: ### It will zero variance features
         from sklearn.feature_selection import VarianceThreshold
         var_thres=VarianceThreshold(threshold=0)
         var_thres.fit(df)

Out[49]: VarianceThreshold(threshold=0)

In [50]: df.columns[var_thres.get_support()]

Out[50]: Index(['PoNumber', 'SaleType', 'Month', 'Costsectioncode', 'Groupcode',
                'Typecode', 'Productcode', 'Description', 'LotNumber', 'CustomerName',
                'Nsv', 'UnitWeight', 'Lp', 'PaymentTerm', 'Quantity'],
               dtype='object')
```

No features that have the same value in all samples, none of the feature is removed as of now.

2.3.2 Dropping Constant Features Using Pearson Correlation

A Pearson correlation is a number between -1 and 1 that indicates the extent to which variables are correlated; we can predict one from the other. Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information. We will use the Pearson Correlation here.



```
ı [53]: # with the following function we can select highly correlated features
        # it will remove the first feature that is correlated with anything other feature

        def correlation(dataset, threshold):
            col_corr = set()  # Set of all the names of correlated columns
            corr_matrix = dataset.corr()
            for i in range(len(corr_matrix.columns)):
                for j in range(i):
                    if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                        colname = corr_matrix.columns[i]  # getting the name of column
                        col_corr.add(colname)
            return col_corr

ı [54]: corr_features = correlation(df, 0.7)
        len(set(corr_features))

ıt[54]: 3

ı [55]: corr_features

ıt[55]: {'Description', 'Lp', 'Typecode'}

ı [56]: # As we need Description and Lp is needed for analysis, so we drop only Groupcode and Typecode
        df=df.drop('Description', axis=1)
        df=df.drop('Typecode', axis=1)
        df
```

Pearson correlations are suitable only for metrics:

The correlation coefficient has values between -1 to 1

- A value closer to 0 implies weaker correlation (exact 0 implying no correlation)
- A value closer to 1 implies stronger positive correlation

2.3.3 Information gain - mutual information In Regression

Mutual Information

Estimate mutual information for a continuous target variable.

Mutual information (MI) between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.

The function relies on nonparametric methods based on entropy estimation from k-nearest neighbours' distances

Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable.

In short

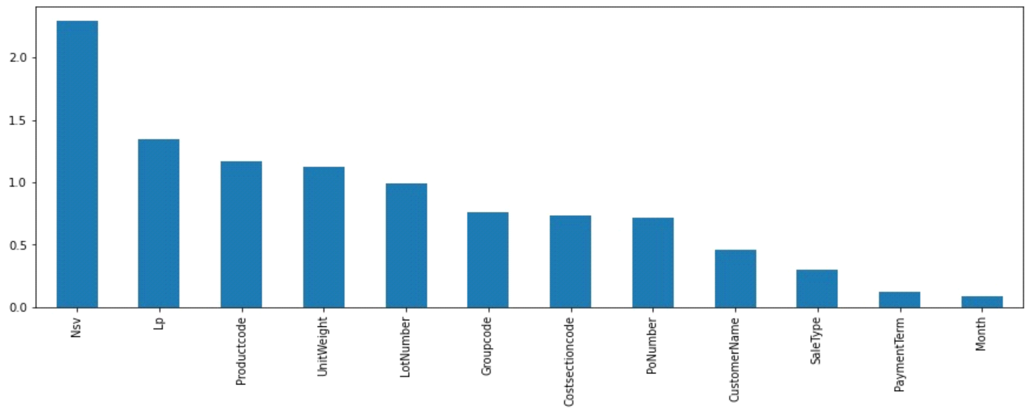
A quantity called mutual information measures the amount of information one can obtain from one random variable given another.

The mutual information between two random variables X and Y

```
In [59]: from sklearn.feature_selection import mutual_info_regression
         # determine the mutual information
         mutual_info = mutual_info_regression(X_train.fillna(0), y_train)
         mutual_info

Out[59]: array([0.71457177, 0.30012616, 0.08749829, 0.7381715 , 0.76452324,
                1.16580687, 0.99126793, 0.45798268, 2.2945013 , 1.12084913,
                1.34609808, 0.12292555])
```

```
In [60]: mutual_info = pd.Series(mutual_info)
         mutual_info.index = X_train.columns
         mutual_info.sort_values(ascending=False)
```

```
In [61]: 
         mutual_info.sort_values(ascending=False).plot.bar(figsize=(15,5))

Out[61]: <AxesSubplot:>
```



```
ı [66]: # As we found in mutual_info_regression 70 percent of fields are considered for analysis,
        # even though month is less related with dependent variable its need for sales analysis
        # so we drop only SaleType and PaymentTerm
        df=df.drop('SaleType', axis=1)
        df=df.drop('PaymentTerm', axis=1)
```

```
ı [67]: # as we splitted the x_idep during feature selection and will use it for all model builbing ,
        # so we drop the values in x_idep as well as df(the main data)
        x_idep=x_idep.drop('SaleType', axis=1)
        x_idep=x_idep.drop('PaymentTerm', axis=1)
```

```
ı [68]: #we drop LotNumber as we will not be able known the LotNumber in future months Quantity perdiction
        df=df.drop('LotNumber', axis=1)
        x_idep=x_idep.drop('LotNumber', axis=1)
```

```
ı [69]: # As we generally we know Productcode is mostly similar to group code,
        #so we drop Productcode
        df=df.drop('Productcode', axis=1)
        x_idep=x_idep.drop('Productcode', axis=1)
```

```
ı [70]: # As we will get purchase order number only after pruchase is made
        # but we are trying to find the sales Quantity for upcoming months
        df=df.drop('PoNumber', axis=1)
        x_idep=x_idep.drop('PoNumber', axis=1)
```

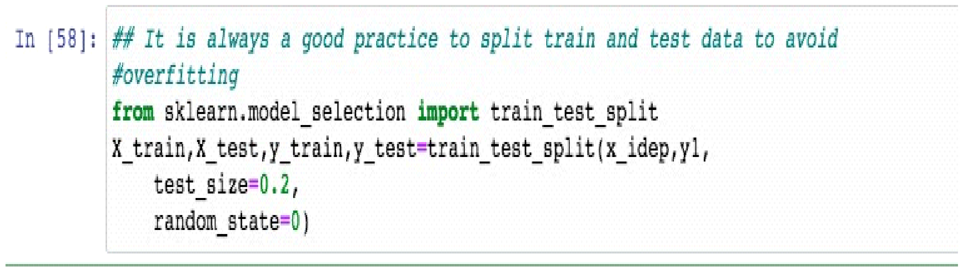
CHAPTER 3

MODEL BUILDING USING MACHINE LEARNING AND HYPERPARAMETER TUNING.

Process commonly used in Machine Learning Models.

3.0.1 Train-Test Split

In the development of machine learning models, it is desirable that the trained model perform well on new, unseen data. In order to simulate the new, unseen data, the available data is subjected to data splitting whereby it is split to 2 portions (sometimes referred to as the train-test split). Particularly, the first portion is the larger data subset that is used as the training set.

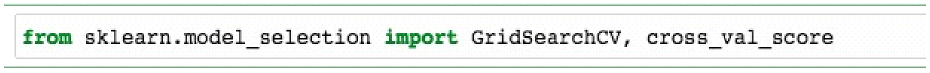


```
In [58]: ## It is always a good practice to split train and test data to avoid
         #overfitting
         from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test=train_test_split(x_idep,y1,
             test_size=0.2,
             random_state=0)
```

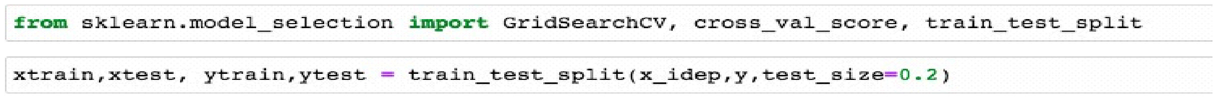
Once train-test split is done it is used same for all upcoming machine learning models

3.0.2 Cross-Validation

In order to make the most economical use of the available data, an ***N-fold cross-validation (CV)*** is normally used whereby the dataset is partitioned to *N* folds (*i.e.* commonly 5-fold or 10-fold CV are used). In such *N*-fold CV, one of the fold is left out as the testing data while the remaining folds are used as the training data for model building.

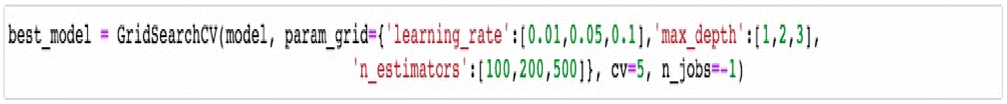


```
from sklearn.model_selection import GridSearchCV, cross_val_score
```



```
from sklearn.model_selection import GridSearchCV, cross_val_score, train_test_split
```

```
xtrain,xtest, ytrain,ytest = train_test_split(x_idep,y,test_size=0.2)
```

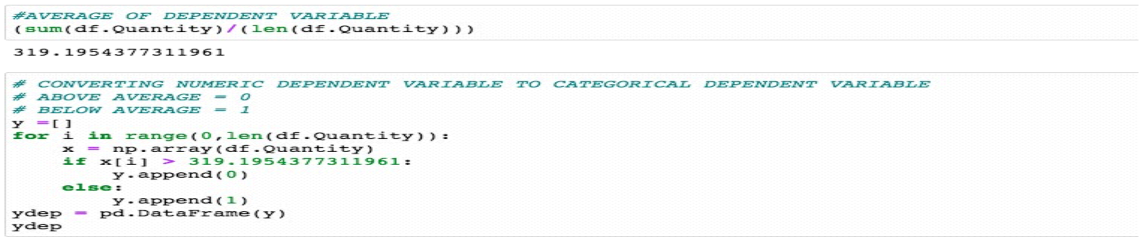


```
best_model = GridSearchCV(model, param_grid={'learning_rate':[0.01,0.05,0.1],'max_depth':[1,2,3],
                                             'n_estimators':[100,200,500]}, cv=5, n_jobs=-1)
```

3.1 Logistic Regression

For our data set no need of logistic regression but divide the dependent variable and took average by using average the independent variable is splinted into o's and 1's then model is fit with logistic regression and train-test split which is already done where the model is trained and before predicting the model is converted into normal values by Predicted y-test value by multiplying the y-test with Predicted probability value, so that the result is viewed by the normal quantity values than 0's and 1's.

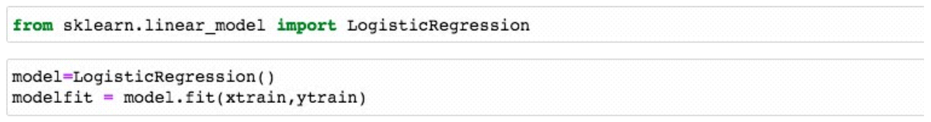
To predict the logistic Regression



```
#AVERAGE OF DEPENDENT VARIABLE
(sum(df.Quantity)/(len(df.Quantity)))
```

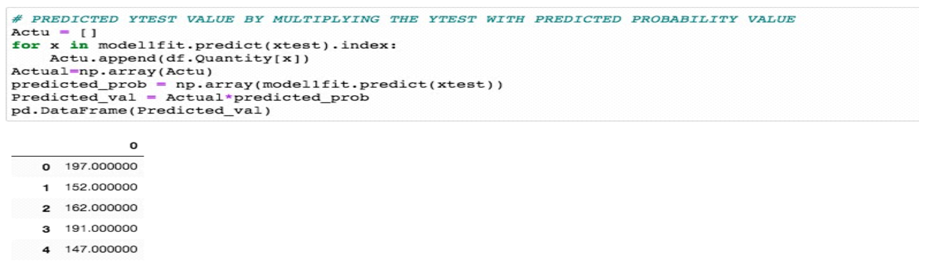
```
319.1954377311961
```

```
# CONVERTING NUMERIC DEPENDENT VARIABLE TO CATEGORICAL DEPENDENT VARIABLE
# ABOVE AVERAGE = 0
# BELOW AVERAGE = 1
y =[]
for i in range(0,len(df.Quantity)):
    x = np.array(df.Quantity)
    if x[i] > 319.1954377311961:
        y.append(0)
    else:
        y.append(1)
ydep = pd.DataFrame(y)
ydep
```



```
from sklearn.linear_model import LogisticRegression
```

```
model=LogisticRegression()
modelfit = model.fit(xtrain,ytrain)
```



```
# PREDICTED YTEST VALUE BY MULTIPLYING THE YTEST WITH PREDICTED PROBABILITY VALUE
Actu = []
for x in modellfit.predict(xtest).index:
    Actu.append(df.Quantity[x])
Actual=np.array(Actu)
predicted_prob = np.array(modellfit.predict(xtest))
Predicted_val = Actual*predicted_prob
pd.DataFrame(Predicted_val)
```

	0
0	197.000000
1	152.000000
2	162.000000
3	191.000000
4	147.000000

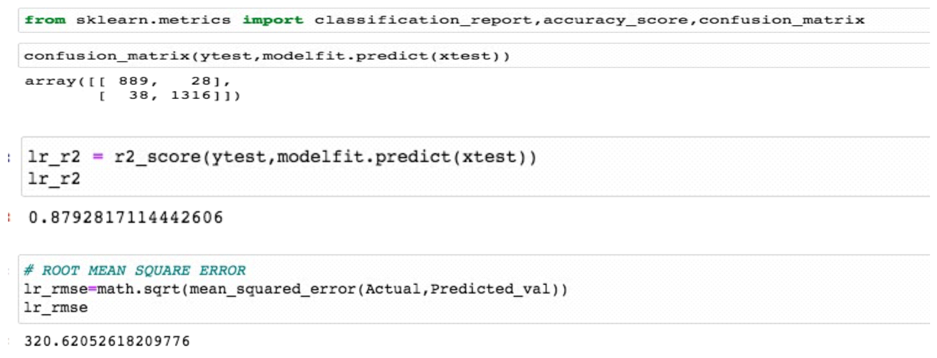
3.1.1Confusion Matrix

Confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

3.1.2 How to Calculate a Confusion Matrix

1. You need a test dataset or a validation dataset with expected outcome values.
2. Make a prediction for each row in your test dataset.
3. From the expected outcomes and predictions count:

The number of correct predictions for each class.



```
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
```

```
confusion_matrix(ytest,modelfit.predict(xtest))
```

```
array([[ 889,   28],
       [  38, 1316]])
```

```
lr_r2 = r2_score(ytest,modelfit.predict(xtest))
lr_r2
```

```
0.8792817114442606
```

```
# ROOT MEAN SQUARE ERROR
lr_rmse=math.sqrt(mean_squared_error(Actual,Predicted_val))
lr_rmse
```

```
320.62052618209776
```

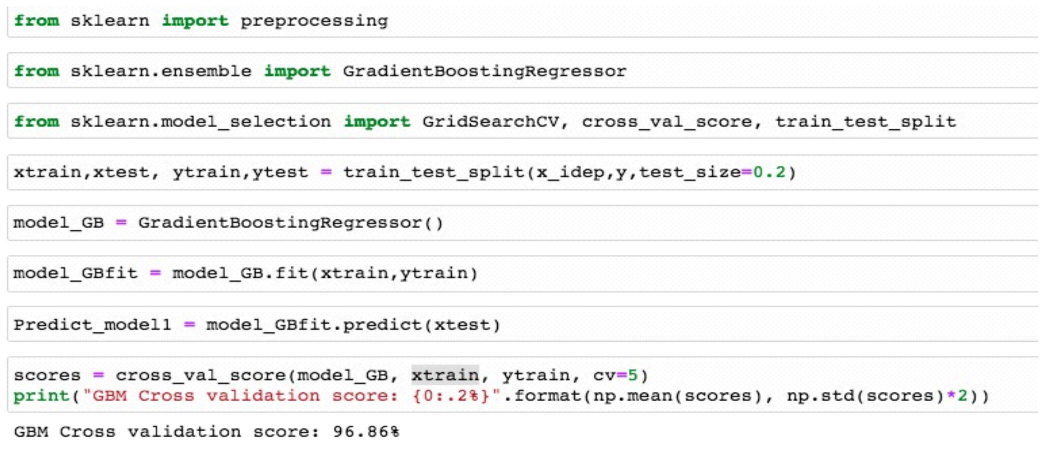
This algorithm works best in the segment we get Root Mean Square Error is 320.6, and We apply some model in this dataset, we see which one the best accuracy.

3.2 Gradient Boosting Regressor

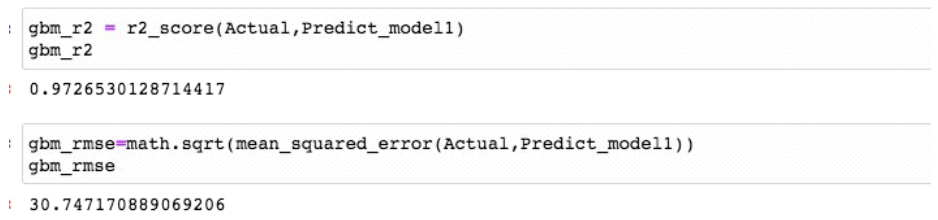
A Gradient Boosting Machine combines the predictions from multiple decision trees to generate the final predictions. In this model that all the weak learners in a gradient boosting machine are decision trees.

Here – **the nodes in every decision tree take a different subset of features for selecting the best split.** This means that the individual trees aren't all the same and hence they are able to capture different signals from the data.

Additionally, each new tree takes into account the errors or mistakes made by the previous trees. So, every successive decision tree is built on the errors of the previous trees. This is how the trees in a gradient boosting machine algorithm are built sequentially.



```
from sklearn import preprocessing

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model_selection import GridSearchCV, cross_val_score, train_test_split

xtrain,xtest, ytrain,ytest = train_test_split(x_idep,y,test_size=0.2)

model_GB = GradientBoostingRegressor()

model_GBfit = model_GB.fit(xtrain,ytrain)

Predict_model1 = model_GBfit.predict(xtest)

scores = cross_val_score(model_GB, xtrain, ytrain, cv=5)
print("GBM Cross validation score: {0:.2%}".format(np.mean(scores), np.std(scores)*2))

GBM Cross validation score: 96.86%
```



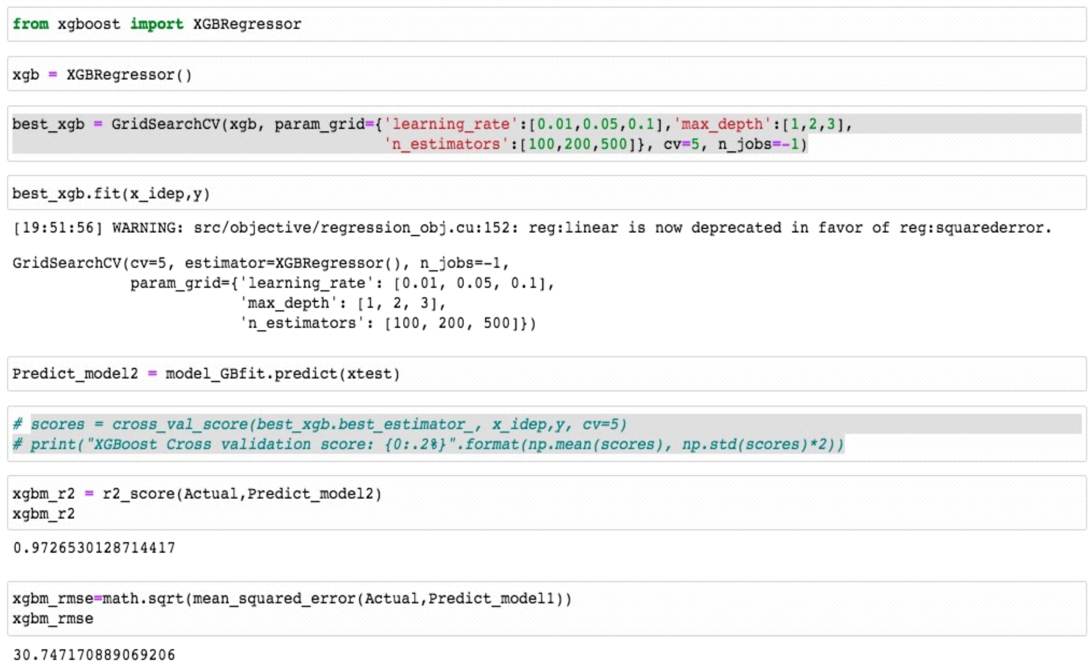
```
: gbm_r2 = r2_score(Actual,Predict_model1)
  gbm_r2

: 0.9726530128714417

: gbm_rmse=math.sqrt(mean_squared_error(Actual,Predict_model1))
  gbm_rmse

: 30.747170889069206
```

We get a Root Mean Square Error in GBM is 30.74 so we apply some model in this dataset, we see which one the best accuracy.

3.3 Extreme Gradient Boosting Regressor

XGBoost is another popular boosting algorithm. In fact, XGBoost is simply an improvised version of the GBM algorithm! The working procedure of XGBoost is the same as GBM. The trees in XGBoost are built sequentially, trying to correct the errors of the previous trees.

Additionally, if you are using the XGBM algorithm, you don't have to worry about imputing missing values in your dataset. **The XGBM model can handle the missing values on its own**. During the training process, the model learns whether missing values should be in the right or left node.



```
from xgboost import XGBRegressor
```

```
xgb = XGBRegressor()
```

```
best_xgb = GridSearchCV(xgb, param_grid={'learning_rate':[0.01,0.05,0.1],'max_depth':[1,2,3],
                                         'n_estimators':[100,200,500]}, cv=5, n_jobs=-1)
```

```
best_xgb.fit(x_idep,y)
```

```
[19:51:56] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

GridSearchCV(cv=5, estimator=XGBRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.05, 0.1],
                         'max_depth': [1, 2, 3],
                         'n_estimators': [100, 200, 500]})
```

```
Predict_model2 = model_GBfit.predict(xtest)
```

```
# scores = cross_val_score(best_xgb.best_estimator_, x_idep,y, cv=5)
# print("XGBoost Cross validation score: {0:.2%}".format(np.mean(scores), np.std(scores)*2))
```

```
xgbm_r2 = r2_score(Actual,Predict_model2)
xgbm_r2
```

```
0.9726530128714417
```

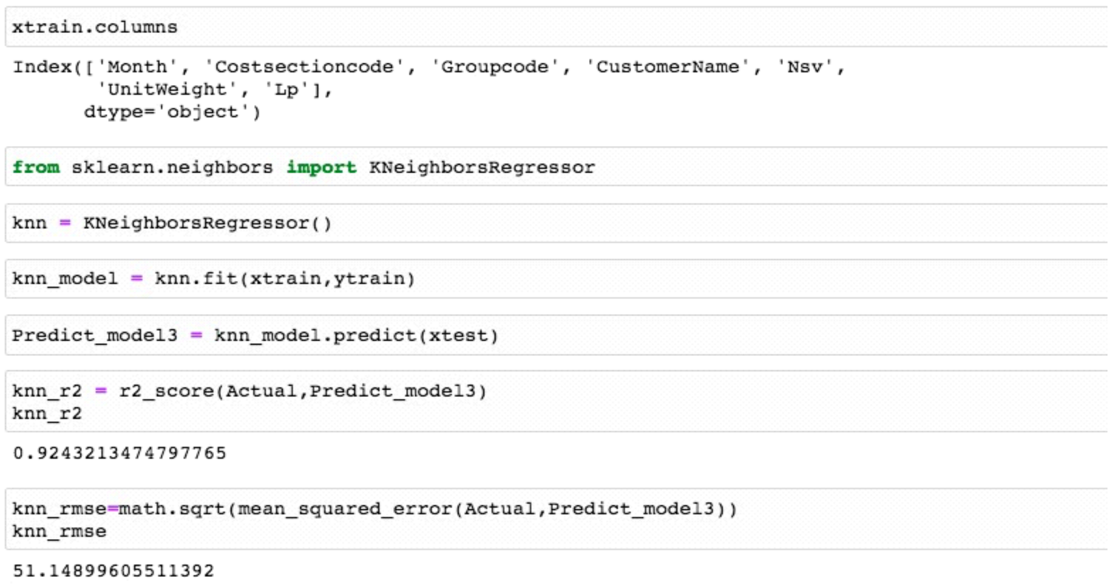
```
xgbm_rmse=math.sqrt(mean_squared_error(Actual,Predict_model1))
xgbm_rmse
```

```
30.747170889069206
```

We get an Root Mean Square Error in XGBM **is 30.74, so we apply some model in this dataset, we see which one the best accuracy.**

3.4 K Nearest Neighbour Regressor

K Nearest Neighbors algorithm. It is one of the simplest and widely used classification algorithms in which a new data point is classified based on similarity in the specific group of neighboring data points. This gives a competitive result.
For a given data point in the set, the algorithms find the distances between this and all other K numbers of datapoint in the dataset close to the initial point and votes for that category that has the most frequency. Usually, Euclidean distance is taking as a measure of distance. Thus, the end resultant model is just the labeled data placed in a space. This algorithm is popularly known for various applications like genetics, forecasting, etc. The algorithm is best when more features are present.



```
xtrain.columns

Index(['Month', 'Costsectioncode', 'Groupcode', 'CustomerName', 'Nsv',
       'UnitWeight', 'Lp'],
      dtype='object')

from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor()

knn_model = knn.fit(xtrain,ytrain)

Predict_model3 = knn_model.predict(xtest)

knn_r2 = r2_score(Actual,Predict_model3)
knn_r2

0.9243213474797765

knn_rmse=math.sqrt(mean_squared_error(Actual,Predict_model3))
knn_rmse

51.14899605511392
```

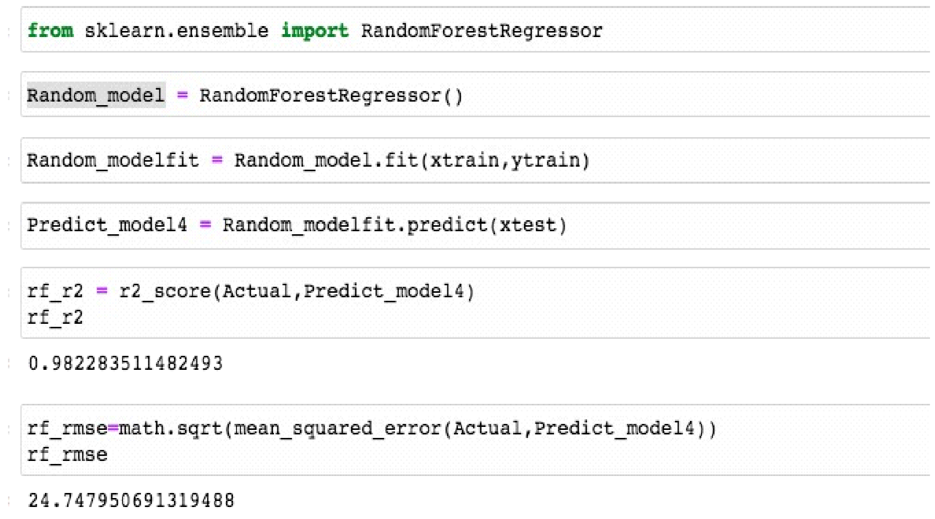
We get a Root Mean Square Error in KNN is 51.14, so we apply some model in this dataset, we see which one the best accuracy.

3.5 Random Forest Regressor

Random Forest is also a “Tree”-based algorithm that uses the qualities features of multiple Decision Trees for making decisions.

This data set by implementing the Random Forest Regression in place of the Decision Tree Regression. Additionally, the Random Forest algorithm is also very *fast* and *robust* than other regression models.

To summarize in short, The Random Forest Algorithm merges the output of multiple Decision Trees to generate the final output.



```
from sklearn.ensemble import RandomForestRegressor

Random_model = RandomForestRegressor()

Random_modelfit = Random_model.fit(xtrain,ytrain)

Predict_model4 = Random_modelfit.predict(xtest)

rf_r2 = r2_score(Actual,Predict_model4)
rf_r2

0.982283511482493

rf_rmse=math.sqrt(mean_squared_error(Actual,Predict_model4))
rf_rmse

24.747950691319488
```

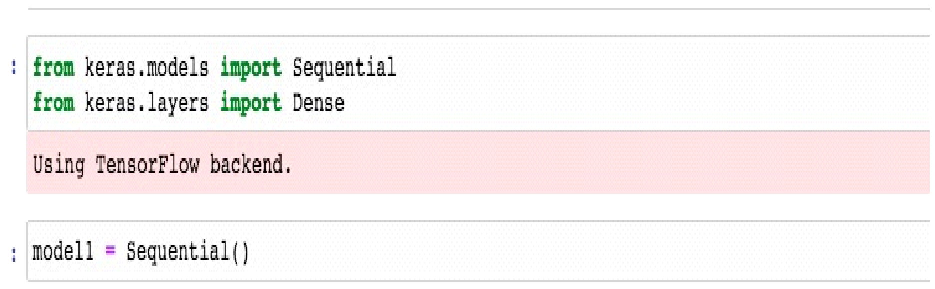
We get a Root Mean Square Error in Random Forest Regression is 24.74 is the Good one to fit the model in this sales quantity prediction.

CHAPTER 4

MODEL BUILDING USING DEEP LEARNING.

Artificial Neural Network

Artificial Neural Network is capable of learning any nonlinear function. Hence, these networks are popularly known as **Universal Function Approximators**. ANNs have the capacity to learn weights that map any input to the output.
perceptron (or neuron) can be imagined as a Logistic Regression. Artificial Neural Network, or ANN, is a group of multiple perceptron's/ neurons at each layer. ANN is also known as a **Feed-Forward Neural network** because inputs are processed only in the forward direction:
As you can see here, ANN consists of 3 layers – Input, Hidden and Output. The input layer accepts the inputs, the hidden layer processes the inputs, and the output layer produces the result. Essentially, each layer tries to learn certain weights.



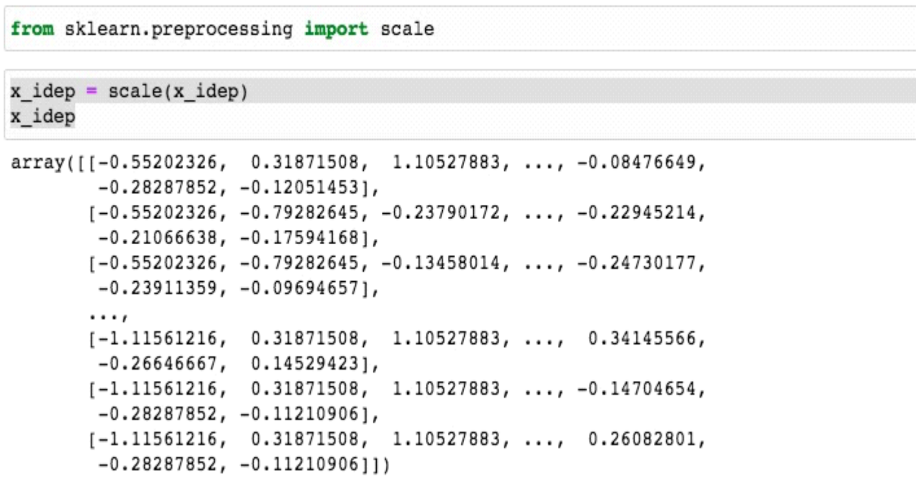
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
model1 = Sequential()
```

4.1. Data Preprocessing

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since the range of values of data may vary widely, it becomes a necessary step in data preprocessing while using deep learning algorithms.



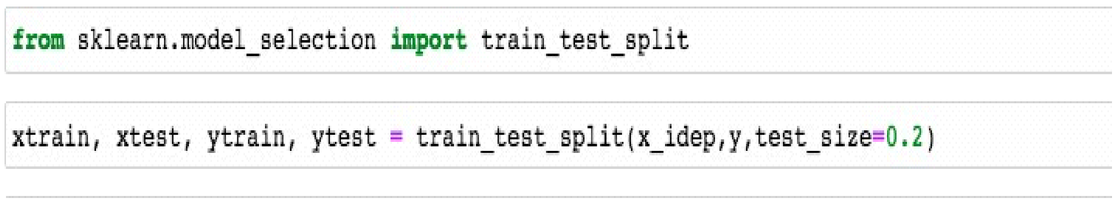
```
from sklearn.preprocessing import scale
```

```
x_idep = scale(x_idep)
x_idep
```

```
array([[-0.55202326,  0.31871508,  1.10527883, ..., -0.08476649,
        -0.28287852, -0.12051453],
       [-0.55202326, -0.79282645, -0.23790172, ..., -0.22945214,
        -0.21066638, -0.17594168],
       [-0.55202326, -0.79282645, -0.13458014, ..., -0.24730177,
        -0.23911359, -0.09694657],
       ...,
       [-1.11561216,  0.31871508,  1.10527883, ...,  0.34145566,
        -0.26646667,  0.14529423],
       [-1.11561216,  0.31871508,  1.10527883, ..., -0.14704654,
        -0.28287852, -0.11210906],
       [-1.11561216,  0.31871508,  1.10527883, ...,  0.26082801,
        -0.28287852, -0.11210906]])
```

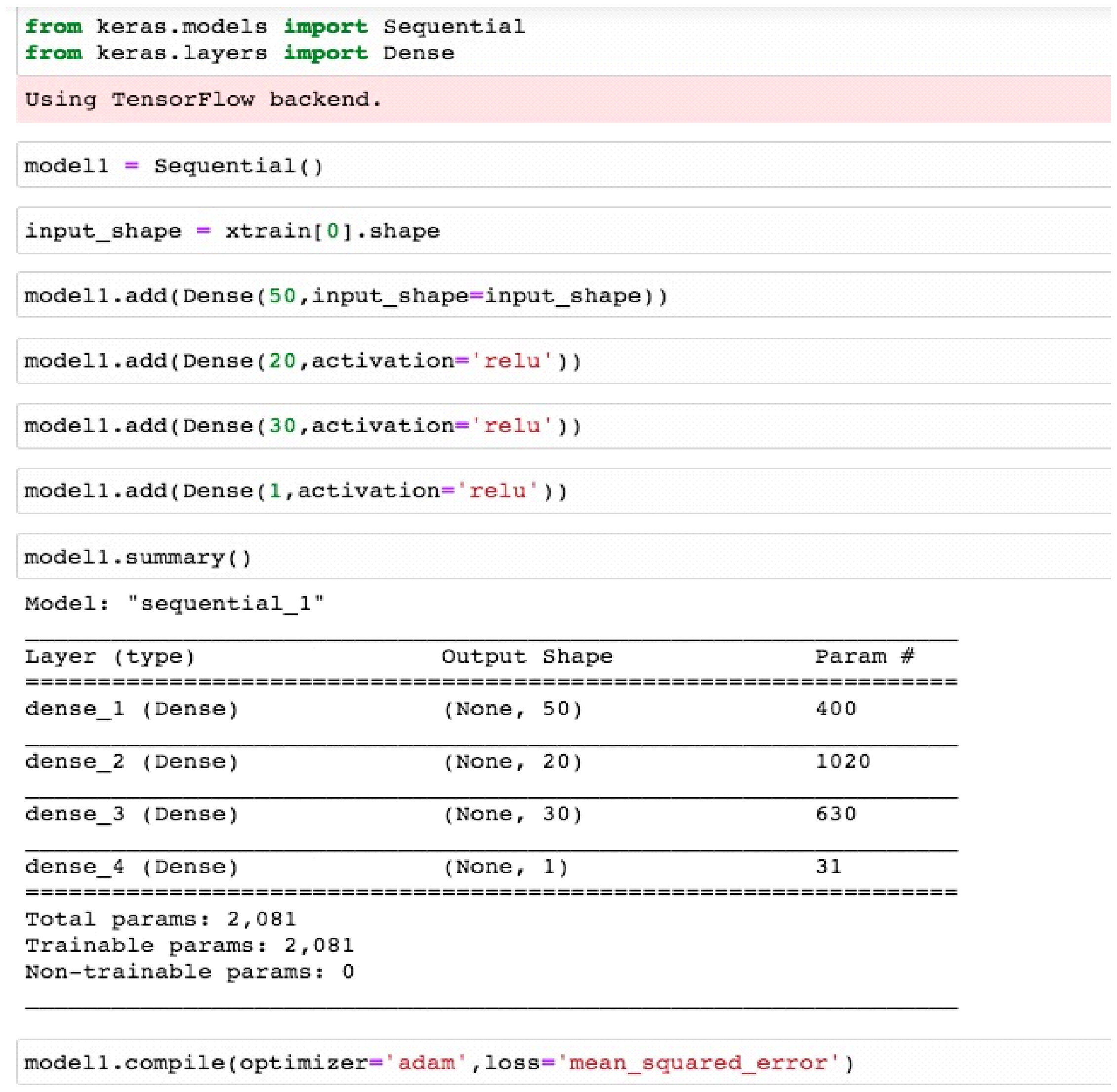
4.2. Training and Model Fitting

The first thing we need to do before building a model is to create a model object itself, this object will be an instance of the class called Sequential.



```
from sklearn.model_selection import train_test_split
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x_idep,y,test_size=0.2)
```



```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
model1 = Sequential()
```

```
input_shape = xtrain[0].shape
```

```
model1.add(Dense(50,input_shape=input_shape))
```

```
model1.add(Dense(20,activation='relu'))
```

```
model1.add(Dense(30,activation='relu'))
```

```
model1.add(Dense(1,activation='relu'))
```

```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 50)                400       
_________________________________________________________________
dense_2 (Dense)              (None, 20)                1020      
_________________________________________________________________
dense_3 (Dense)              (None, 30)                630       
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 31        
=================================================================
Total params: 2,081
Trainable params: 2,081
Non-trainable params: 0
_________________________________________________________________
```

```
model1.compile(optimizer='adam',loss='mean_squared_error')
```

We use model.fit() method to train the model, we pass three arguments inside the method which are

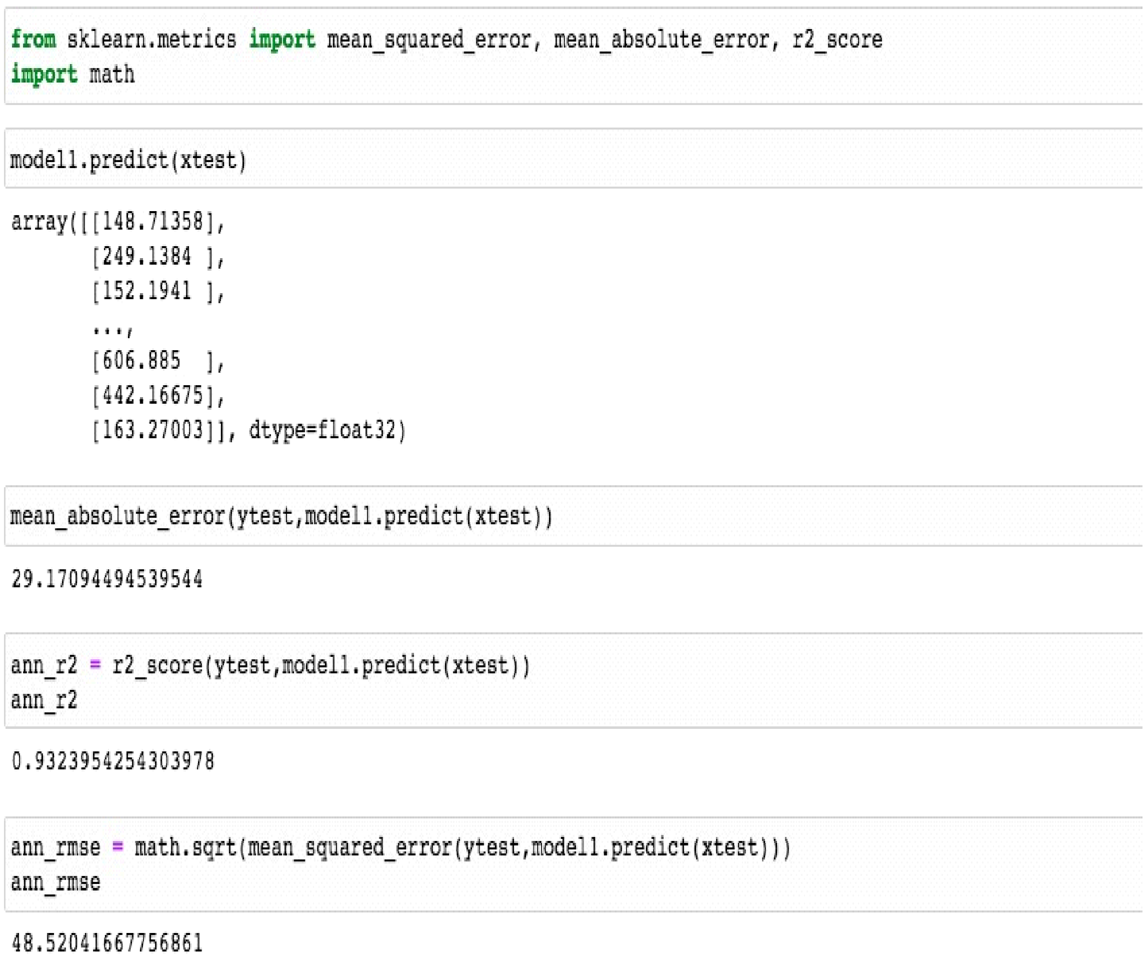
input → x_train is the input that is fed to the network

output → this contains the correct answers for the x_train i.e. y_train

no.of.epochs → It means the number of times you are going to train the network with the dataset.

4.3. Evaluating the Model

We are going to evaluate the performance of the model by applying it to the test set. Here evaluate method returns two arguments one is the loss incurred in the test set by the prediction and the other one is the accuracy.



```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import math
```

```
model1.predict(xtest)
```

```
array([[148.71358],
       [249.1384 ],
       [152.1941 ],
       ...,
       [606.885  ],
       [442.16675],
       [163.27003]], dtype=float32)
```

```
mean_absolute_error(ytest,model1.predict(xtest))
```

```
29.17094494539544
```

```
ann_r2 = r2_score(ytest,model1.predict(xtest))
ann_r2
```

```
0.9323954254303978
```

```
ann_rmse = math.sqrt(mean_squared_error(ytest,model1.predict(xtest)))
ann_rmse
```

```
48.52041667756861
```

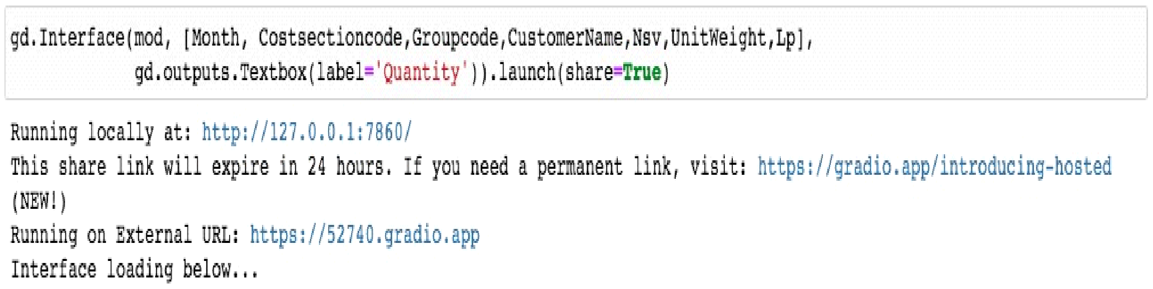
CHAPTER 5

DEPLOYMENT

Deployment is the method by which integrate a machine learning model into an existing production environment to make practical business decisions based on data. It is one of the last stages in the machine learning life cycle.

5.1. Gradio

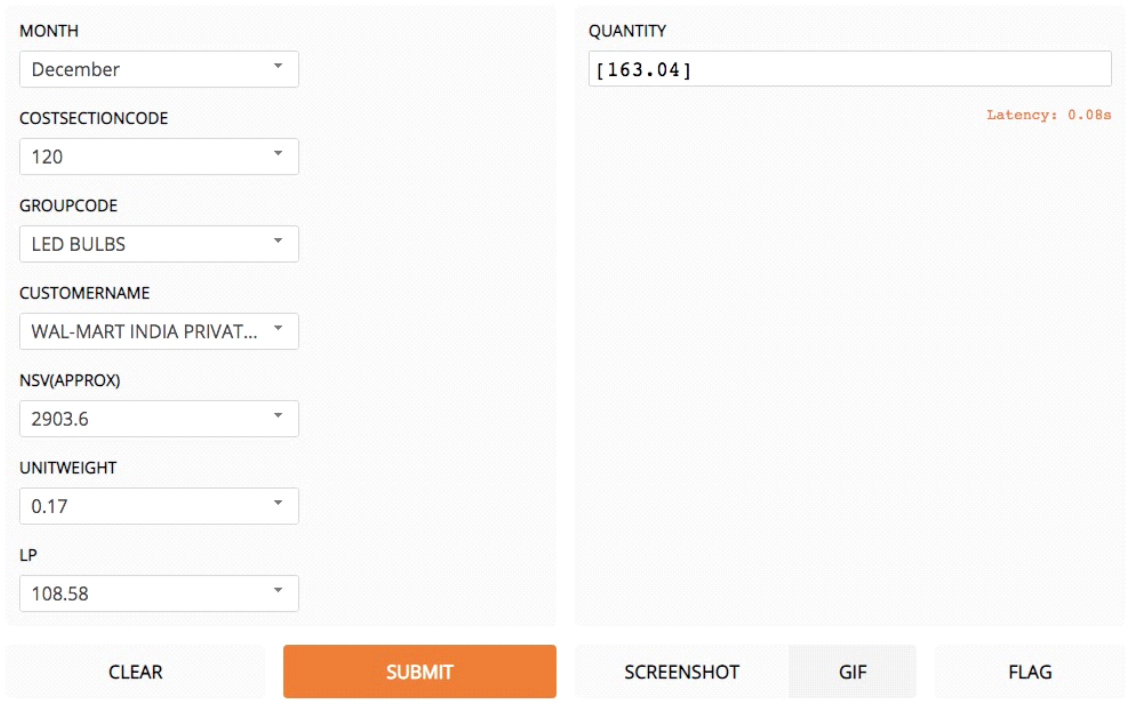
Gradio helps in building an online GUI in a couple of lines of code which is convenient for showing exhibitions of the model presentation. It is quick, simple to set up, and prepared to utilize, and shareable as the public connection which anybody can get to distantly and parallelly run the model in your machine. Gradio **works with a wide range of media-text, pictures, video, and sound.**



```
gd.Interface(mod, [Month, Costsectioncode,Groupcode,CustomerName,Nsv,UnitWeight,Lp],
             gd.outputs.Textbox(label='Quantity')).launch(share=True)
```

```
Running locally at: http://127.0.0.1:7860/
This share link will expire in 24 hours. If you need a permanent link, visit: https://gradio.app/introducing-hosted
(NEW!)
Running on External URL: https://52740.gradio.app
Interface loading below...
```

By entering the values of independent variables (Month, Cost section code, Group code, Customer Name, Nsv, Unit Weight, Lp) and output dependent variable (Quantity) is gets predicted using machine learning model and displays on the quantity output textbox using Gradio user interface


MONTH
December

COSTSECTIONCODE
120

GROUPCODE
LED BULBS

CUSTOMERNAME
WAL-MART INDIA PRIVAT...

NSV(APPROX)
2903.6

UNITWEIGHT
0.17

LP
108.58

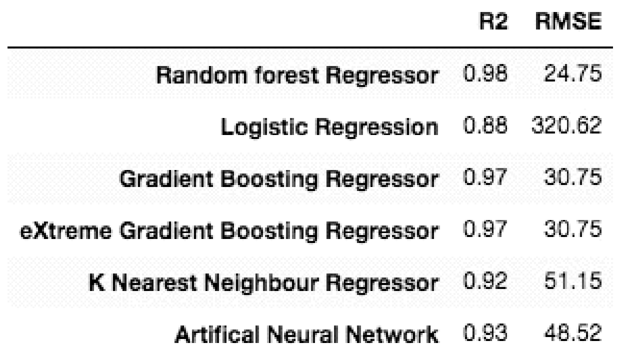
QUANTITY
[163.04]

Latency: 0.08s

CHAPTER 6

CONCLUSION

Out of the Machine learning and Deep learning algorithms used, Random Forest Regressor model gave us the least RMSE score.
With the help of RMSE score we can say Random Forest Regressor model has least error when compared to other machine learning and deep learning models.



	R2	RMSE
Random forest Regressor	0.98	24.75
Logistic Regression	0.88	320.62
Gradient Boosting Regressor	0.97	30.75
eXtreme Gradient Boosting Regressor	0.97	30.75
K Nearest Neighbour Regressor	0.92	51.15
Artifical Neural Network	0.93	48.52