Big Mart Sales Prediction

Capstone Project Report

Instructor: Rouzbeh Razavi

Authored by

Manasa Chelukala

Reg No: 811186837

Contents

INTRODUCTION:

Forecasting sales has always been a critical area to focus on. To maintain the efficiency of marketing organizations, all suppliers must use an efficient and optimal forecasting method. Manual material handling of this work may result in significant errors, leading to poor organizational management, and, more importantly, would be time consuming, which is not desired in today's fast-paced environment. The primary goal of business sectors is to attract the target audience. As a result, it is critical that the firm has already demonstrated its ability to achieve this goal through the use of a prediction model.

Big Mart is a massive global store network. Big Mart trends are critical because data scientists use them to identify potential centers based on product and location. To achieve the best results, data scientists can use a computer to predict Big Mart sales. Many companies rely heavily on their data and require market forecasting. Forecasting entails analyzing data from a variety of sources, such as consumer trends, purchasing behavior, and other factors. This study would also help businesses manage their finances more effectively.

And this is where machine learning can truly shine. To forecast sales using various machine learning algorithms, we use data mining approaches such as discovery, data exploration, data cleaning, feature engineering, model building, and testing. Pre-processing raw data acquired by a large mart for missing data, abnormalities, and outliers is the approach used in this method. The data will then be used to train an algorithm to create a model.

PROBLEM STATEMENT:

Big Mart's data scientists gathered 2013 sales data for 1559 products from 10 stores in various cities. Certain characteristics of each product and store have also been defined. The goal is to create a predictive model and determine the sales of each product in a specific store.

Big Mart will use this model to try to understand the properties of products and stores that are important in increasing sales.

LITERATURE:

Research on sales prediction has been done and some of them has been discussed below:

To predict sales, the general linear approach, decision tree approach, and good gradient approach were used[1]. The initial data set considered had many entries, but the final data set used for analysis was much smaller than the original due to non-usable data, redundant entries, and insignificant sales data.

Sales forecasting using linear regression and the Random Forest algorithm, which included data collection and translation into processed data[3]. Finally, they predicted which model would yield the best results.

Three modules, hive, R programming, and tableau[4], were used to forecast sales. By analyzing the store's history, you can gain a better understanding of the store's revenue and make changes to the target to make it more successful. Key values are obtained within the diagram to reduce all intermediate values by reducing the intermediate key feature to obtain the results.

In his research, Mohit Gurnani demonstrates that composite models outperform individual models. He also claimed that decomposition mechanisms outperform hybrid mechanisms [5].

In his research, J. Scott Armstrong discussed predicting solutions to intriguing and difficult sales forecasting problems [6].

In his research, Samaneh Beheshti-Kashi reviewed various approaches on the predictive potential of consumer-generated content and search queries [7].

Gopal Behera conducted a thorough study on Big Mart sales forecasting and provided prediction metrics for various existing models [8].

In this project, we will use linear regression, Decision Tree, and random forest methodologies to pre-process raw data obtained from a large mart for missing data, anomalies, and outliers. The final results will then be predicted using an algorithm. ETL stands for Extract, Transform, and Load, and we finally compare all of the models to predict which model produces the most accurate results.

Assumptions or Observations:

1. Sales are higher during weekends.
2. Higher sales during morning and late evening.
3. Higher sales during end of the year.
4. Store size affects the sales.
5. Location of the store affects the sales.
6. Items with more shelf space sell more

EXPLORATORY DATA ANALYSIS:

It is advantageous to combine train and test data in order to explore data in each dataset and thus merge train and test data for data visualization and feature engineering.

Importing Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score,cross_val_predict
from sklearn import metrics
import statsmodels.api as sm
```

```
In [95]: #importing test and train data
df_train=pd.read_csv("C:\\Users\\HP\\Desktop\\Capstone Project\\train.csv")
df_test=pd.read_csv("C:\\Users\\HP\\Desktop\\Capstone Project\\test.csv")
```

```
In [9]: df_train['source'] = 'train'
df_test['source'] = 'test'
df=pd.concat([df_train,df_test], ignore_index=True)
```

```
In [10]: df.head()
```

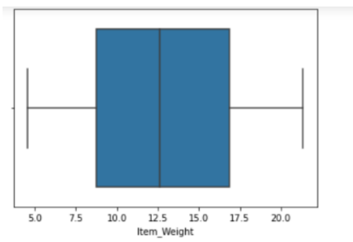
Out[10]:

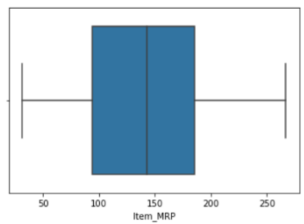
	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	

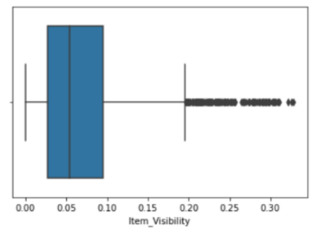
For the exploratory method, univariate analysis and bivariate analysis are to be conducted to obtain data information.

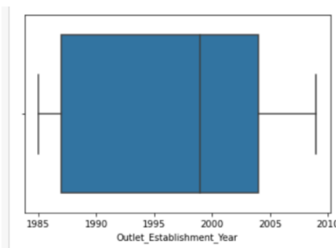
Checking for the Outliers

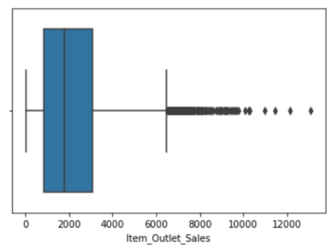
```
#Checking for the outliers
for i in df_train.describe().columns:
  sns.boxplot(df_train[i].dropna())
  plt.show()
```





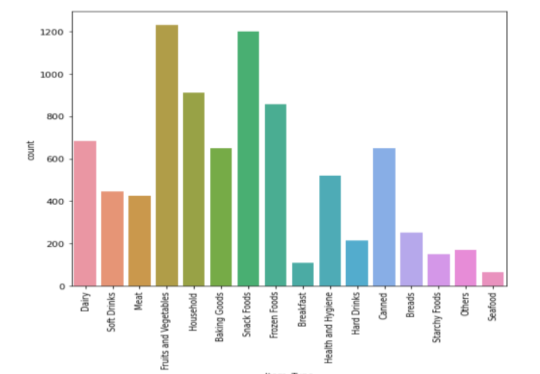


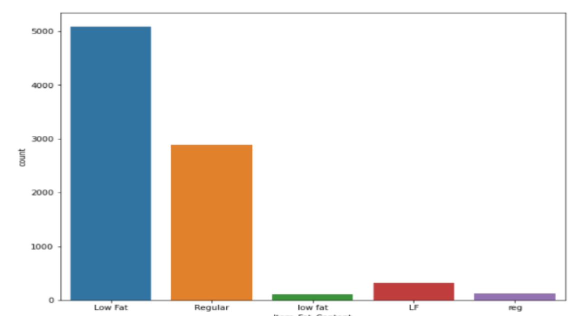


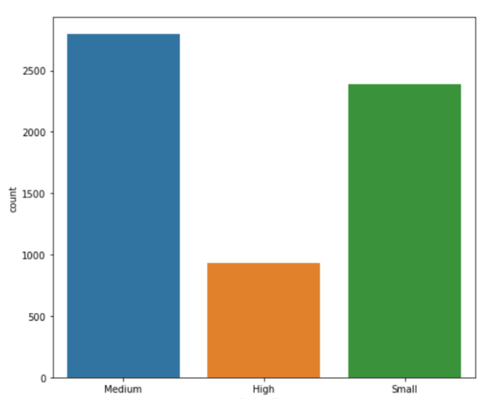


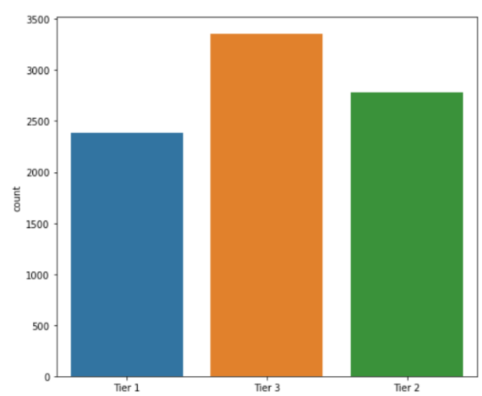
Univariate Analysis

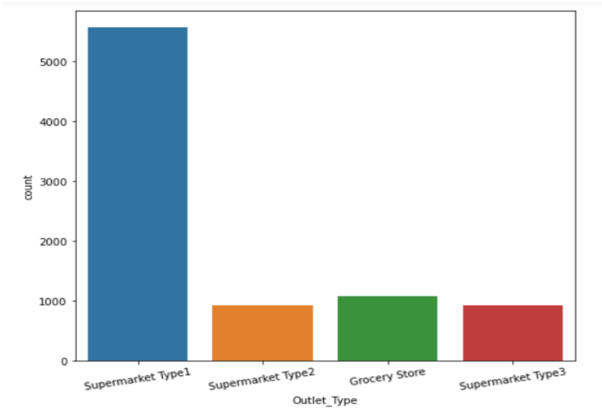
We will start off by plotting and exploring all the individual variables to gain some insights.

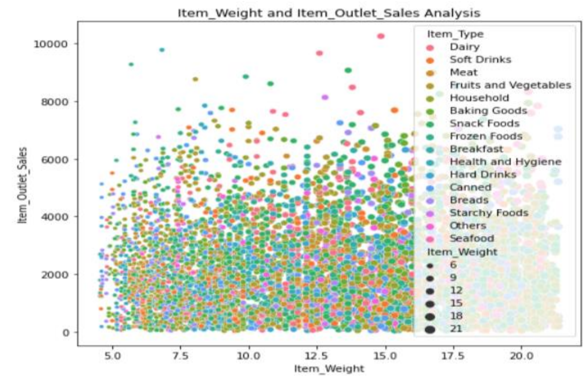








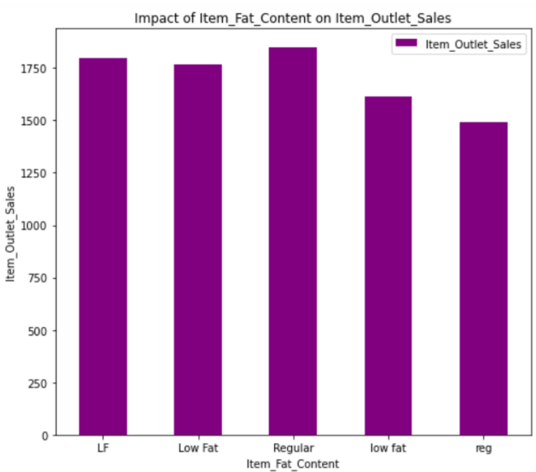


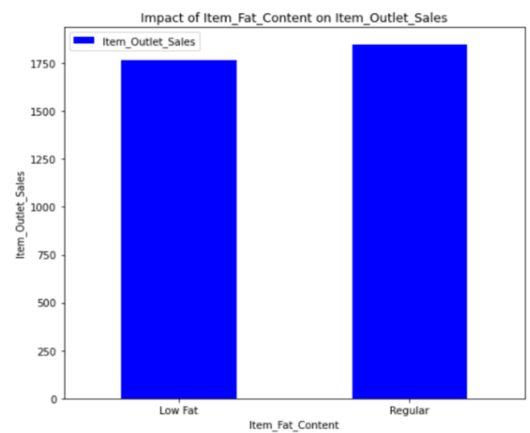


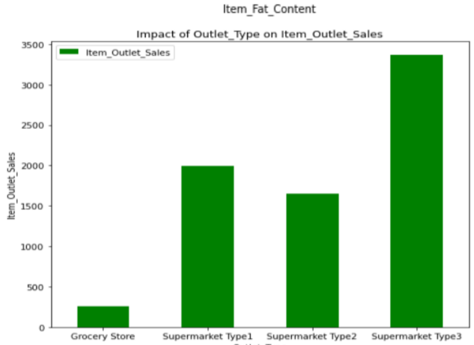
Several observations were made during the Univariate Analysis, and they are as follows: The categories 'LF,' 'low fat,' and 'Low Fat' are interchangeable, as are'reg' and 'Regular.' As a result, they can be combined into one, and low fat items are nearly twice as expensive as regular items. Fruit and snacks are the most popular items in the Item Type column. The variable goal is skewed to the right. These items are not edible, but they are all labeled as lowfat or regular.

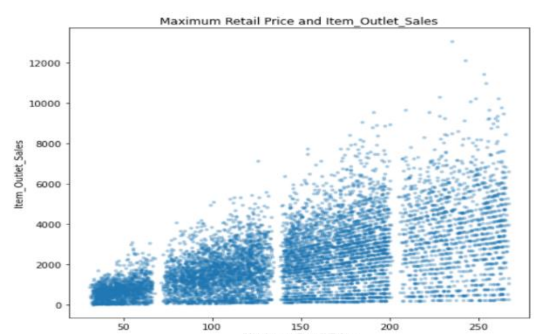
Bivariate Analysis:

After going over each feature individually, let's go over them again in relation to the target variable. Bar plots will be used here for both continuous or numeric variables and categorical variables.









Bivariate analysis revealed a clear relationship between product weight and sales, as well as item fat content and sales. Products with visibility of less than 0.2 generate a significant amount of sales. Individuals have chosen a low fat category over others. In the relationship between the item identifiers and the outlet size, the items are purchased more frequently as the outlet size increases. The item's exposure indicates that more visible items sell for less money.

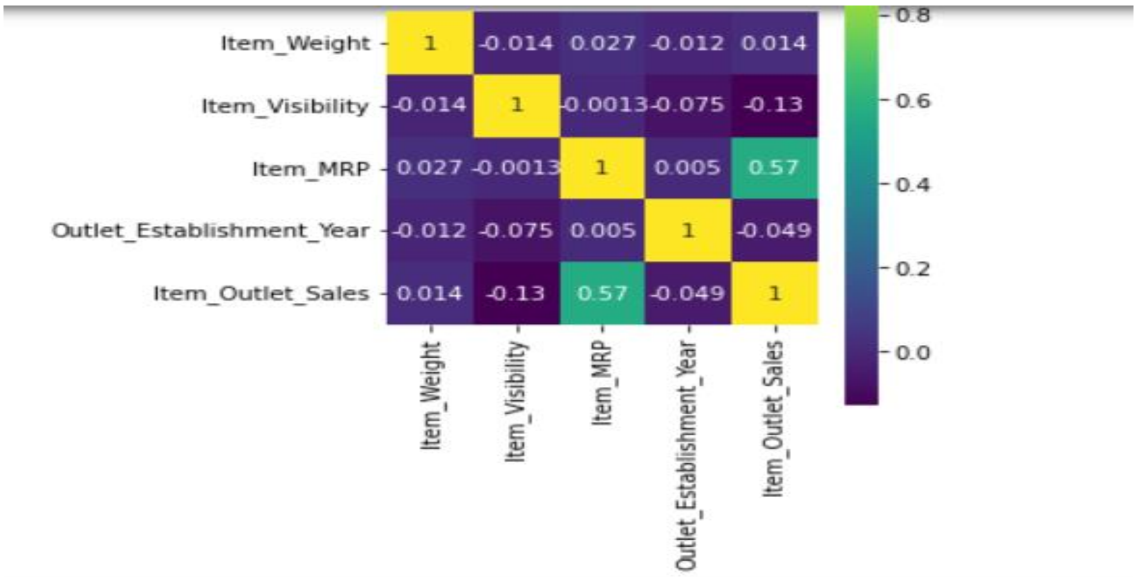
Correlation between different attributes

In [116]:
```
df_train.corr()
```

Out[116]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.000000	-0.014048	0.027141	-0.011588	0.014123
Item_Visibility	-0.014048	1.000000	-0.001315	-0.074834	-0.128625
Item_MRP	0.027141	-0.001315	1.000000	0.005020	0.567574
Outlet_Establishment_Year	-0.011588	-0.074834	0.005020	1.000000	-0.049135
Item_Outlet_Sales	0.014123	-0.128625	0.567574	-0.049135	1.000000

In [81]:
```
plt.figure(figsize=(4,5))
sns.heatmap(df_train.corr(),vmax=1, square=True,annot=True, cmap='viridis')
plt.title('Correlation between different attributes')
plt.show()
```



The heatmap above shows that the greater the dependability of the target variable on the corresponding attribute, the greater the intensity of the attribute's color in relation to the target variable. As a result, Item outlet sales is less dependent on Item visibility and more dependent on Item MRP.

DATA CLEANING:

During the data visualization phase, it is discovered that the attributes Item Weight and Outlet Size have missing values. Data pre-processing is required in order to fill missing values in the data so that it can be used by a machine learning model, which increases the model's efficiency. The missing values that correspond to the Item Weight were filled by averaging the weight of the specific item, while the missing values that correspond to the Outset Size were filled by using the mode of the outlet size of a specific type of outlet. The dataset was created using Big Mart 2013 sales results, and there are a total of 12 attributes.

Treating the Missing values

Missing data can have a significant impact on predictive model development because missing values may contain important information that can aid in making better predictions. As a result, missing data imputation becomes essential. Depending on the problem and the data, there are various methods for dealing with missing values.

Item_Weight:

From the boxplot,We saw that the item weight column is approximately normal, so we can replace the missing values with the column Mean.

```
In [41]: df['Item_Weight'].mean() #replacing the NaN values with this mean
Out[41]: 12.792854228644991
```

```
In [42]: df['Item_Weight'].fillna(df['Item_Weight'].mean(), inplace=True) #missing values are replaced with the mean using fillna functio
```

Outlet_Size:

We will fill in the missing values in Outlet Size with the most frequently occurring item, in this case Medium.

```
In [43]: df['Outlet_Size'].value_counts()
Out[43]: Medium    4655
         Small     3980
         High      1553
         Name: Outlet_Size, dtype: int64
```

```
In [44]: df['Outlet_Size'].fillna('Medium', inplace=True)
```

```
In [45]: df.isnull().sum() #No null values in Outlet_Size
Out[45]: 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            5681
         source                          0
         dtype: int64
```

FEATURE ENGINEERING:

Feature Engineering is a technique for using domain data understanding to build functions that work with machine learning algorithms. When feature engineering is done correctly, the predictive capability of machine learning algorithms is improved by creating raw data features that aid in the machine learning process. Correction of incorrect values is also a part of feature engineering.

We can investigate the Item Type variable. Previously, we saw that the Item Type variable has 16 categories, which could be very useful in analysis. So combining them is a good idea. One option is to manually assign each a new category. But there is a catch. The Item Identifier, or the unique ID of each item, begins with either FD, DR, or NC. If you look at the categories, they appear to be Food, Drinks, and Non-Consumables. So I've created a new column using the Item Identifier variable:

In [78]:
```
df['Item_Type'].value_counts()
```

Out[78]:
```
Fruits and Vegetables    2013
Snack Foods              1989
Household                1548
Frozen Foods             1426
Dairy                    1136
Baking Goods             1086
Canned                   1084
Health and Hygiene        858
Meat                      736
Soft Drinks               726
Breads                    416
Hard Drinks               362
Others                    280
Starchy Foods             269
Breakfast                 186
Seafood                    89
Name: Item_Type, dtype: int64
```

Item types are either Food, Drinks or Non-Consumables

A closer look at each Item Identifier reveals that they begin with "FD", "DR" (Drinks), or "NC" (Non-Consumables).

In [79]:
```
df['Item_Identifier'].value_counts()
```

Out[79]:
```
FDU15    10
FDS25    10
FDA38    10
FDW03    10
FDJ10    10
         ..
FDR51     7
FDM52     7
DRN11     7
FDH58     7
NCW54     7
Name: Item_Identifier, Length: 1559, dtype: int64
```

To improve our analysis, we will create three new categories in addition to the existing 16 categories.

In [80]:
```
#Changing only the first 2 characters (category ID)

df['New_Item_type'] = df['Item_Identifier'].apply(lambda x: x[0:2])
```

In [81]:
```
#Renaming the categories:

df['New_Item_type'] = df['New_Item_type'].map({'FD':'Food','NC':'Non-Consumabl
df['New_Item_type'].value_counts()
```

Out[81]:
```
Food              10201
Non-Consumable     2686
Drinks             1317
Name: New_Item_type, dtype: int64
```

If a product is non-consumable then why associate a fat-content to them? We will remove it. #Mark non-consumables as separate category in low_fat.

In [82]:
```
df.loc[df['New_Item_type']=='Non-Consumable','Item_Fat_Content']= "Non-Edible"
df['Item_Fat_Content'].value_counts()
```

Out[82]:
```
Low Fat       6499
Regular       5019
Non-Edible    2686
Name: Item_Fat_Content, dtype: int64
```

Modifying Item_Visibility

We have noticed that the minimum value in this case is 0, which makes no sense. Consider it a piece of missing information and link it to the product's low visibility.

In [74]:
```
df[df['Item_Visibility']==0]['Item_Visibility'].count()
```

Out[74]:
```
879
```

In [75]:
```
df['Item_Visibility'].fillna(df['Item_Visibility'].median(), inplace=True)
```

Normally, the more visible a product is, the more likely it is to sell. Based on that hypothesis, we can calculate the importance of a product in a given store by averaging the importance of the same product in all other stores.

In [83]:
```
Item_Visibility_Avg = df.pivot_table(values='Item_Visibility', index='Item_Identifier')
```

In [84]:
```
Item_Visibility_Avg
```

Out[84]:

	Item_Visibility
Item_Identifier	
DRA12	0.034938
DRA24	0.045646
DRA59	0.133384
DRB01	0.079736
DRB13	0.006799
...	...
NCZ30	0.027302
NCZ41	0.056396
NCZ42	0.011015
NCZ53	0.026330
NCZ54	0.081345

Correcting incorrect values is also part of feature engineering. The visibility of the item had a minimum value of 0 in the device dataset, which is unacceptable because the item should be accessible to all. As a result, it was replaced by the column's mean.

```
In [237]: function = lambda x: x['Item_Visibility']/Item_Visibility_Avg['Item_Visibility'][Item_Visibility_Avg.index == x['Item_Identifier'
          df['Item_Visibility_Avg'] = df.apply(function,axis=1).astype(float)
```

```
In [239]: df['Item_Visibility_Avg'].describe()
```

```
Out[239]: count    14204.000000
          mean         1.000000
          std          0.348382
          min          0.000000
          25%          0.921522
          50%          0.962037
          75%          1.042007
          max          3.010094
          Name: Item_Visibility_Avg, dtype: float64
```

Determining the Years of Operations of a Store

We wanted to add a new column that showed how long a store had been open. This is possible by doing the following:

```
In [222]: df['Outlet_Establishment_Year'].value_counts()
```

```
Out[222]: 1985    2439
          1987    1553
          1999    1550
          1997    1550
          2004    1550
          2002    1548
          2009    1546
          2007    1543
          1998     925
          Name: Outlet_Establishment_Year, dtype: int64
```

```
In [53]: df['Outlet_Years'] = 2013-df['Outlet_Establishment_Year']
         df['Outlet_Years'].describe()
```

```
Out[53]: count    14204.000000
         mean        15.169319
         std          8.371664
         min          4.000000
         25%          9.000000
         50%         14.000000
         75%         26.000000
         max         28.000000
         Name: Outlet_Years, dtype: float64
```

Because Outlet Years is a new column, we must consider how long the store has been open rather than the year it was founded.

Categorical Variables Transformation

As a result, since scikit-learn only accepts numerical variables, I converted all nominal variable types to numeric types. I also wanted the variable Outlet Identifier. As a result, I created a new variable called 'Outlet,' which is the same as the Outlet Identifier. Outlet Identifier should be left alone because it is required in the submission file.

We will use the LabelEncoder function to convert all categorical variables into numeric types (Values of 0 or 1) so that we can build models on them.

```
In [240]: from sklearn.preprocessing import LabelEncoder
          label = LabelEncoder()

          #New variable for outlet
          df['Outlet'] = label.fit_transform(df['Outlet_Identifier'])
          varib = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','New_Item_type','Outlet_Type','Outlet']
          for i in varib:
            df[i] = label.fit_transform(df[i])
```

```
In [241]: df.head()
```

Out[241]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location
0	FDA15	9.30	0	0.016047	Dairy	249.8092	OUT049	1999	1	
1	DRC01	5.92	2	0.019278	Soft Drinks	48.2692	OUT018	2009	1	
2	FDN15	17.50	0	0.016760	Meat	141.6180	OUT049	1999	1	
3	FDX07	19.20	2	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	1	
4	NCD19	8.93	1	0.000000	Household	53.8614	OUT013	1987	0	

One-Hot Coding of Categorical variables

The term "one-hot-coding" refers to the process of creating dummy variables, one for each category of a categorical variable. Item Fat Content, for example, has three categories: 'Low Fat,' 'Regular,' and 'Non-Edible.' One hot coding operation will remove this variable and create three new variables. Each will contain binary numbers 0 (if the category does not exist) and 1. (if category is present). This is possible with Pandas' 'get dummies' function.

```
In [ ]: #Dummy Variables:

df = pd.get_dummies(df, columns =['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','New_Item_type','Outlet_Type','Outlet']
df.dtypes
```

```
Item_Identifier               object
Item_Weight                  float64
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Item_Outlet_Sales            float64
source                        object
Outlet_Years                   int64
Item_Visibility_Avg          float64
Item_Fat_Content_0             uint8
Item_Fat_Content_1             uint8
Item_Fat_Content_2             uint8
Outlet_Location_Type_0         uint8
Outlet_Location_Type_1         uint8
Outlet_Location_Type_2         uint8
Outlet_Size_0                  uint8
Outlet_Size_1                  uint8
Outlet_Size_2                  uint8
New_Item_type_0                uint8
New_Item_type_1                uint8
New_Item_type_2                uint8
Outlet_Type_0                  uint8
Outlet_Type_1                  uint8
Outlet_Type_2                  uint8
Outlet_Type_3                  uint8
Outlet_0                       uint8
Outlet_1                       uint8
Outlet_2                       uint8
Outlet_3                       uint8
Outlet_4                       uint8
Outlet_5                       uint8
Outlet_6                       uint8
```

We can see that all variables have been converted to floats, and each category has a new variable. Consider the three columns formed by Item Fat Content.

```
In [ ]: df[['Item_Fat_Content_0','Item_Fat_Content_1','Item_Fat_Content_2']].head(10)
```

	Item_Fat_Content_0	Item_Fat_Content_1	Item_Fat_Content_2
0	1	0	0
1	0	0	1
2	1	0	0
3	0	0	1
4	0	1	0
5	0	0	1
6	0	0	1
7	1	0	0
8	0	0	1
9	0	0	1

Therefore, we can notice that each row has only one column with the value 1, which corresponds to the category in the original variable.

In [307]:
```
#Droping the columns which are converted to different types:
df.drop(['Item_Type','Outlet_Establishment_Year'],axis=1,inplace=True)
```

Fields Item_Type & Outlet_Establishment._Year are dropped as they are of object type.

MODEL BUILDING:

The dataset is now ready for predictive modeling after Data Cleaning and Feature Engineering have been completed. To learn how to forecast values, the algorithm is fed into the training set. After creating a target variable to forecast, testing data is fed into the model. The predictive models are created by utilizing

- Linear Regression
- Decision Tree
- Random Forest

The data should be re-converted into train and test data sets. Both of these should be exported as modified data sets in order to be reused across multiple sessions.

```
#Divide into test and train:
train = df.loc[df['source']=="train"]
test = df.loc[df['source']=="test"]
```

```
#Droping the unnecessary columns:
test.drop(['Item_Outlet_Sales','source'],axis=1,inplace=True)
train.drop(['source'],axis=1,inplace=True)
```

```
#Export files as modified versions:
train.to_csv("train_modified.csv",index=False)
test.to_csv("test_modified.csv",index=False)
```

- Removing the Item Outlet sales from test data and the source.
- And Removing source from train data.

```
X = train.drop(['Item_Outlet_Sales','Item_Identifier','Outlet_Identifier'],axis=1)
Y = train['Item_Outlet_Sales']
```

Linear Regression:

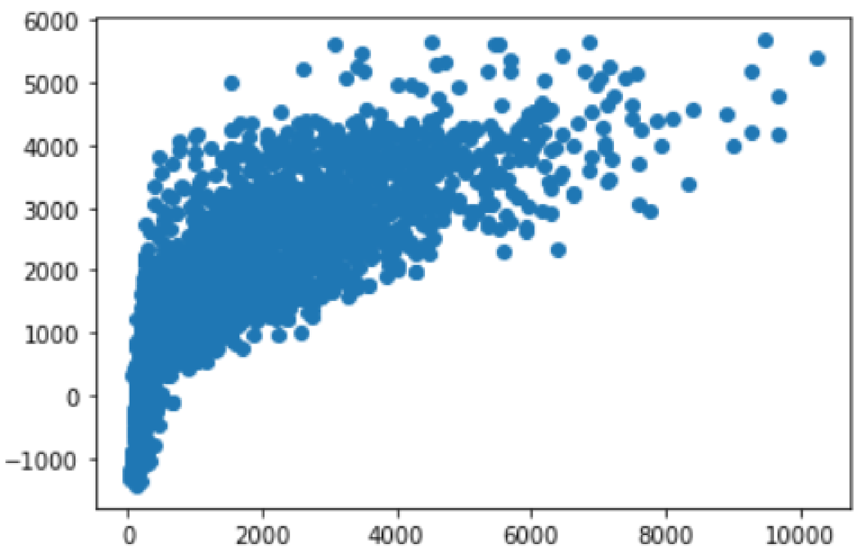
Linear regression is a crucial and widely used regression technique. It's one of the most fundamental regression methods. One of its primary benefits is the ease with which the results can be interpreted. $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$. Where Y is the predicted variable X - Prediction variables (0, 1...r) - Errors at Random No matter how well the model is trained, tested, and validated, there will always be a difference between observed and predicted results, which is irreducible error, so we cannot rely entirely on the learning algorithm's predicted results. A successful linear regression model requires data to meet several conditions.

```
lm = LinearRegression()
```

```
model = lm.fit(X_train,Y_train)
predictions = lm.predict(X_test)
predictions[:5]
```

```
array([1362.80792719,  703.90859258,  856.21656123, 4237.8455983 ,
       3352.78672996])
```

Plotting the Model



Evaluating the Model

```
#Accuracy of the Model
print("Linear Regression Model Score:",model.score(X_test,Y_test))
```

```
Linear Regression Model Score: 0.5693160178336899
```

```
original_values = Y_test
#Root mean squared error
rmse = np.sqrt(metrics.mean_squared_error(original_values,predictions))
print("Linear Regression Model R2 score: ",metrics.r2_score(original_values,predictions))
print("Linear Regression Model RMSE value: ", rmse)
```

```
Linear Regression Model R2 score:  0.5693160178336899
Linear Regression Model RMSE value:  1090.0675781890661
```

Therefore,The RMSE value of the Linear Regression Model is 1090.0675.

Linear Regression Model with Cross Validation

```
# Performing 6-fold cross validation
Score = cross_val_score(model,X,Y,cv=5)
print("Linear Regression Model Cross Validated Score: ",Score)
```

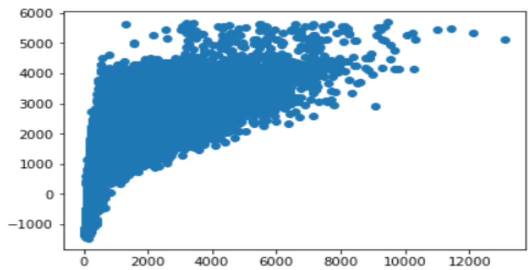
```
Linear Regression Model Cross Validated Score:  [0.57118971 0.55527549 0.54760813 0.56585116 0.56484981]
```

```
#Predicting with cross_val_predict
predictions = cross_val_predict(model,X,Y,cv=6)
predictions[:5]
```

```
array([4010.25243997,  587.58661364, 2338.51646505, 1042.84768269,
        982.61412886])
```

Plotting the Model

```
plt.scatter(Y,predictions)
plt.show()
```



Evaluating the Model

```
accuracy = metrics.r2_score(Y,predictions)
print("Linear Regression Model R2 with Cross Validation: ",accuracy)
```

```
Linear Regression Model R2 with Cross Validation:  0.5613484407669818
```

```
rmse = np.sqrt(metrics.mean_squared_error(y,predictions))
print("Linear Regression Model RMSE with Cross Validation:",rmse)
```

```
Linear Regression Model RMSE with Cross Validation: 1130.1616188829596
```

Decision Tree:

In a top-down approach, it's a simple model with little bias that can be used to create a classifier model, with the root node being the first to be considered. It's a popular machine learning model. A tuple recursive classifier is a decision tree. It is a powerful method of multi-variable analysis and a powerful approach for data mining. In a variety of areas, this approach depicts the variables involved in achieving a specific goal, as well as the motivations for achieving the goal and the means of execution.

```
tree = DecisionTreeRegressor(max_depth=15, min_samples_leaf=100)

tree.fit(X_train,Y_train)
tree_pred = tree.predict(X_test)
tree_pred[:5]
```

```
array([1141.52015273,  659.5803528 ,  659.5803528 , 5204.69368986,
       3048.67928523])
```

```
Results = pd.DataFrame({'Actual':Y_test,'Predicted':tree_pred})
Results.head()
```

	Actual	Predicted
7503	1743.0644	1141.520153
2957	356.8688	659.580353
7031	377.5086	659.580353
1084	5778.4782	5204.693690
856	2356.9320	3048.679285

Evaluating the Model

```
print('R2 Score of Decision Tree Model:' , metrics.r2_score(Y_test,tree_pred))
print('Mean Absolute Error of Decision Tree Model:', metrics.mean_absolute_error(Y_test, tree_pred))
print('Mean Squared Error of Decision Tree Model:', metrics.mean_squared_error(Y_test, tree_pred))
print('Root Mean Squared Error of Decision Tree Model:', np.sqrt(metrics.mean_squared_error(Y_test, tree_pred)))
```

```
R2 Score of Decision Tree Model: 0.5893639233847621
Mean Absolute Error of Decision Tree Model: 743.0158588866356
Mean Squared Error of Decision Tree Model: 1132935.608935475
Root Mean Squared Error of Decision Tree Model: 1064.3944799441017
```

Therefore,The RMSE value of the Decision Tree Model is 1064.3944.

Random Forest:

The random forest algorithm is a highly accurate method for forecasting sales. For forecasting the outcomes of machine learning projects, it is simple to use and understand. Because of their decision tree-like hyperparameters, random forest classifiers are used in sales prediction. A tree model is analogous to a decision-making tool. A random forest model is built for each learner using a random set of rows and a few randomly chosen factors. The final forecast may be based on the best guesses of all the individual students. When dealing with a regression problem, the final forecast may be the average of all previous predictions.

```
rf = RandomForestRegressor(random_state=43)
```

```
rf.fit(X_train,Y_train)
```

```
RandomForestRegressor(random_state=43)
```

```
predictions = rf.predict(X_test)
predictions[:5]
```

```
array([ 802.25571 ,  947.160422,  785.630684, 4729.730014, 2750.240034])
```

```
results = pd.DataFrame({'Actual':Y_test,'Predicted':predictions})
results.head()
```

	Actual	Predicted
7503	1743.0644	802.255710
2957	356.8688	947.160422
7031	377.5086	785.630684
1084	5778.4782	4729.730014
856	2356.9320	2750.240034

Evaluating the Model

```
rmse = np.sqrt(metrics.mean_squared_error(Y_test,predictions))
print("Random Forest Model RMSE :",rmse)
```

```
Random Forest Model RMSE : 1100.474842761797
```

```
print("Random Forest Model R2 Score:",metrics.r2_score(Y_test,predictions))
```

```
Random Forest Model R2 Score: 0.5610529724532947
```

The RMSE value of the Random Forest Model is 1100.474.

CONCLUSION

Machine Learning techniques are used in this framework to predict future sales from data of previous years. The goal of this project was to discuss how machine learning algorithms such as linear regression, decision trees, and random forests can be used to build different machine learning models. Sales results have been predicted using these algorithms. A detailed description has been provided of how noisy data is removed, as well as the algorithms that are used to predict the result. According to the accuracy predicted by different models, decision trees and random forests are the best. Using our predictions, big stores can refine their methodologies and strategies, resulting in increased profitability.

REFERENCES

[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-DecisionMaking).

[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.