PART B

Mini Project:	Implement a machine learning algorithm for a specific domain.

Big Mart Sales Prediction

Introduction:

Big Mart is a big supermarket chain. The data have been collected in 2018, including 1559 products across 10 stores in different cities. They are provided to challenge and make predictions model to ensure success of the business.

Objective:

The objective is to build a predictive model and find out the sales of each product at a particular store. Big Mart will use this model to understand the properties of products and stores which play a key role in increasing sales.

Frame the problem:

Before looking at the data it is important to understand how does the company expect to use and benefit from this model? This first brainstorming helps to determine how to frame the problem, what algorithms to select and measure the performance of each one.
For now, we can categorize our Machine Learning (ML) system as:

Supervised Learning task: we are given labeled training data (e.g. we already know some of the sale price, for some products and stores);

Regression task: our algorithm is expected to predict the sale price for a given product and store.

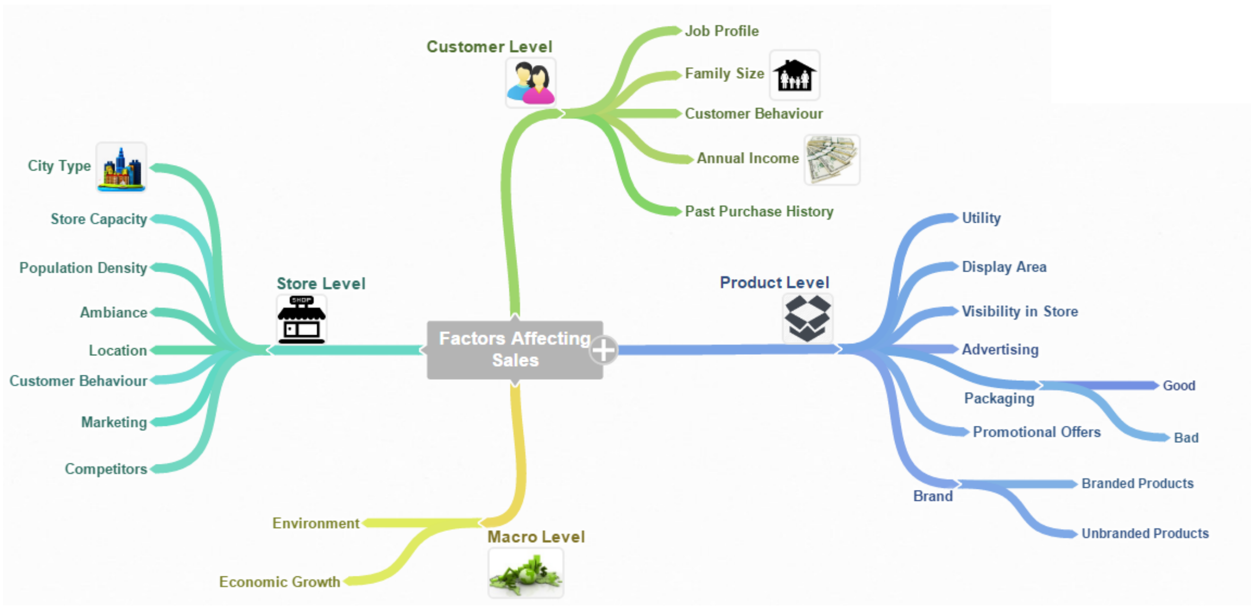
Plain Batch learning: since there is no continuous flow of data coming into our system, there is no particular need to adjust to changing data rapidly, and the data is small enough to fit in memory, so plain batch learning should work.

Performance Measure:

Usually for regression problems the typical performance measure is the RootMean Square Error (RMSE). This function gives an idea of how much error the system makes in its predictions with higher weight for large errors.

Make Assumptions:

After framing our problem and deciding on the performance measure, it is good to make some assumptions on what possible outcomes we might expect from our analysis according to the available data. Therefore, by knowing the goal we should think which possible factors might affect the sales prediction outcome. We can start by diving the process into four levels: **Store level, Product level, Customer level and Macro level.**



Store Level Hypotheses:

1. **City type:** Stores located in urban or Tier 1 cities should have higher sales because of the higher income levels of people there.

2. **Population Density:** Stores located in densely populated areas should have higher sales because of more demand.

3. **Store Capacity:** Stores which are very big in size should have higher sales as they act like one-stop-shops and people would prefer getting everything from one place

4. **Competitors:** Stores having similar establishments nearby should have less sales because of more competition.

5. **Marketing:** Stores which have a good marketing division should have higher sales as it will be able to attract customers through the right offers and advertising.
6. **Location:** Stores located within popular marketplaces should have higher sales because of better access to customers.
7. **Customer Behavior:** Stores keeping the right set of products to meet the local needs of customers will have higher sales.
8. **Ambiance:** Stores which are well-maintained and managed by polite and humble people are expected to have higher footfall and thus higher sales.

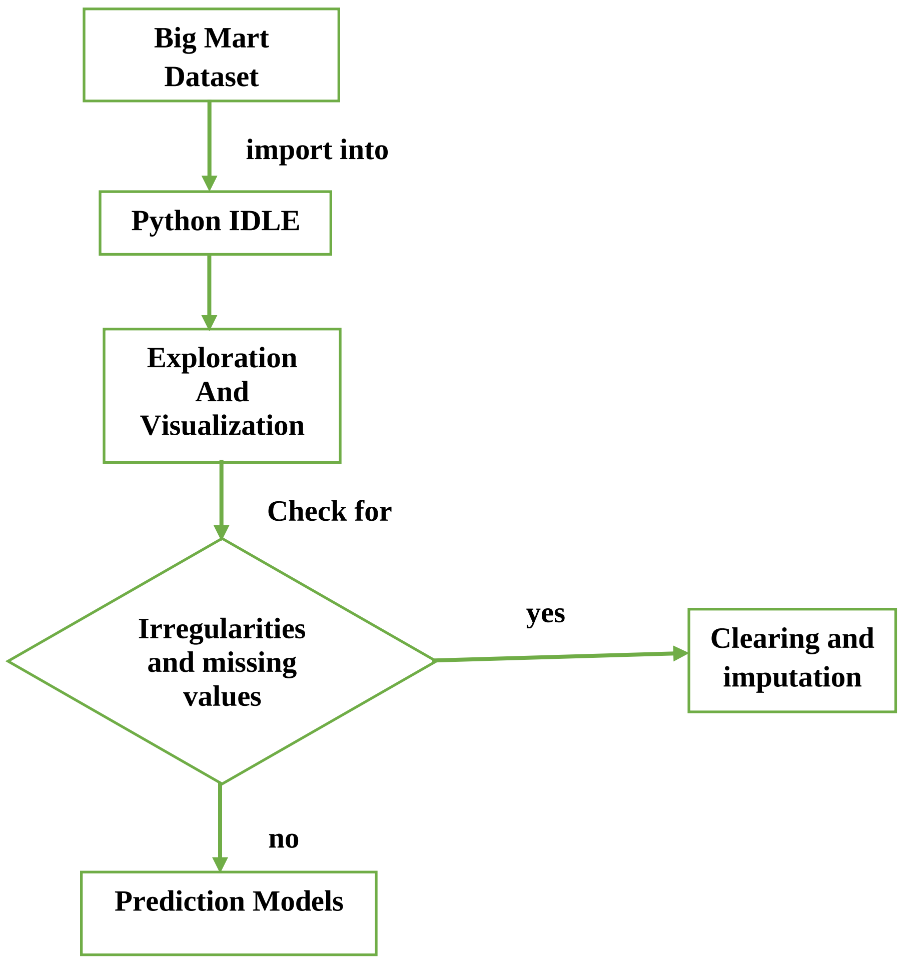
Product Level Hypotheses:

1. **Brand:** Branded products should have higher sales because of higher trust in the customer.
2. **Packaging:** Products with good packaging can attract customers and sell more.
3. **Utility:** Daily use products should have a higher tendency to sell as compared to the specific use products.
4. **Display Area:** Products which are given bigger shelves in the store are likely to catch attention first and sell more.
5. **Visibility in Store:** The location of product in a store will impact sales. Ones which are right at entrance will catch the eye of customer first rather than the ones in back.
6. **Advertising:** Better advertising of products in the store will should higher sales in most cases.
7. **Promotional Offers:** Products accompanied with attractive offers and discounts will sell more.

Organization of Data analysis process:

Our goal is to identify the most important variables and to define the best regression model for predicting out target variable. Hence, this analysis will be divided into five stages:

1. Data Collection
2. Data Analysis
3. Data Pre processing
4. Exporting data
5. Modelling and Validation
6. Visualization
7. Conclusion



1. Collection of Data:

This dataset contains **8523 observations** and **12 features.**
From the image below most of the features follow into the segments “Product” and “Store”, therefore we can exclude our previous assumptions for the other two segments considered since we do not have data to support them.
It was a belief that from this first look at the data, the variables that will have higher impact on the product’s sale price are: Item_Visibility , Item_Type , OutletSize , Outlet_Location_Type , Outlet_Type .
The target variable is Item_Outlet_Sales .

Name	Type	Subtype	Description	Segment	Expectation
Item_Identifier	Numeric	Discrete	Unique Product ID	Product	Low Impact
Item_weight	Numeric	Continuous	Weight of product	Product	Medium Impact
Item_Fat_Content	Categorical	Ordinal	Wether the product is low fat or not	Product	Medium Impact
Item_Visibility	Numeric	Continuous	% of total display area in store allocated to this product	Product	High Impact
Item_Type	Categorical	Nominal	Category to which product belongs	Product	High Impact
Item_MRP	Numeric	Discrete	Maximum Retail Price (list price) of product	Product	Medium Impact
Outlet_Identifier	Numeric	Discrete	Unique Store ID	Store	Low Impact
Outlet_Establishment_Year	Numeric	Discrete	Year in which store was established	Store	Low Impact
Outlet_Size	Categorical	Ordinal	Size of the store	Store	High Impact
Outlet_Location_Type	Categorical	Ordinal	Type of city in which the store is located	Store	High Impact
Outlet_Type	Categorical	Ordinal	Grocery store or some sort of supermarket	Store	High Impact
Item_Outlet_Sales	Numeric	Discrete	Sales of product in particular store. This is the outcome variable to be predicted	Product	Target

2. Data Analysis:

The process of evaluating data using analytical and logical reasoning to examine each component of the data provided. This form of analysis is just one of the many steps that must be completed when conducting a research experiment. Data from various sources is gathered, reviewed, and then analyzed to form some sort of finding or conclusion. There are a variety of specific data analysis method, some of which include data mining, text analytics, business intelligence, and data visualizations.

Importing the packages and csv files

```
[1]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

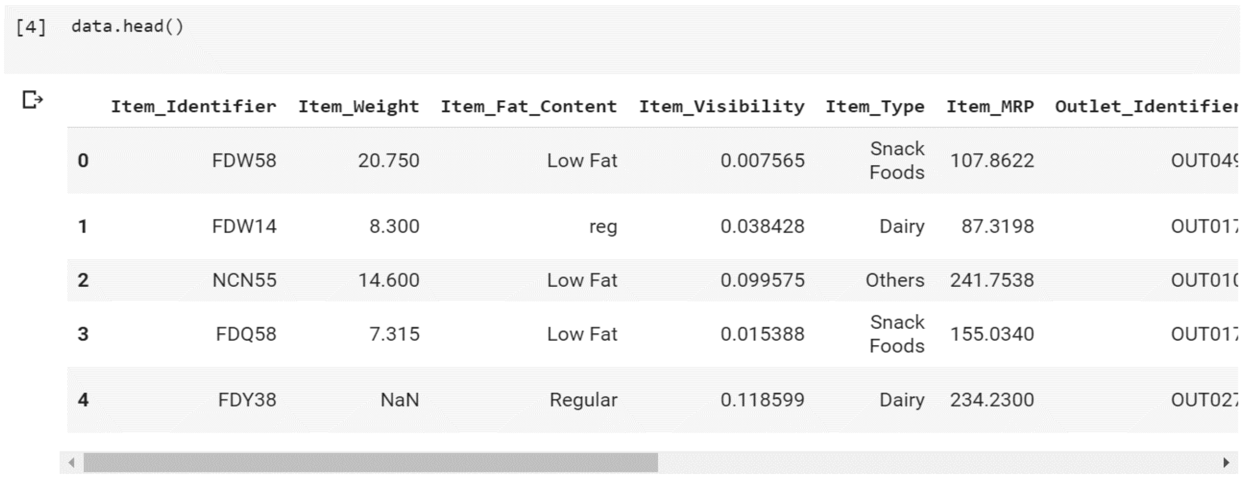
```
[2]  train = pd.read_csv("/content/Test.csv")
     test = pd.read_csv("/content/Train.csv")
```

Lets combine them into a dataframe ‘data’ with a ‘source’ column .

```
[3]  train['source']='train'
     test['source']='test'
     data = pd.concat([train, test],ignore_index=True)
     print(train.shape, test.shape, data.shape)
```

```
(5681, 13) (8523, 13) (14204, 13)
```

Displaying first five rows of the data.csv file



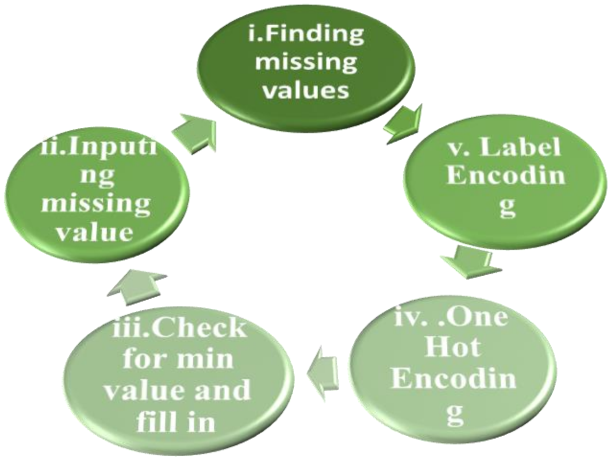
```
[4]  data.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027

Thus we can see that data has same #columns but rows equivalent to both test and train. One of the key challenges in any data set is missing values. Lets start by checking which columns contain missing values.

3. Data Preprocessing

Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. With data preprocessing, we convert raw data into a clean data set.



- Regarding the variables which were thought to have high impact on the product’s sale price. Item_Visibility does not have a high positive correlation as expected, quite the opposite. As well, there are no big variations in the sales due to theItem_Type . On the other hand, it was possible to see that the size, location and type of store could have a positive impact on sales.

- If we look at variable Item_Identifer , we can see different groups of letters per each product such as ‘FD’ (Food), ‘DR’(Drinks) and ‘NC’ (Non-Consumable). From this we can create a new variable.
- Regarding Item_Visibility there are items with the value zero. This does not make lot of sense, since this is indicating those items are not visible on the store.
- Similar, Item_Weight and Outlet_Size seem to present NaN values.
- There seems to be 1562 unique items only available in a single store.
- Item_Fat_Content has vale “low fat” writen in different manners.
- For Item_Type we try to create a new feature that does not have 16 unique values.
- Outlet_Establishment_Year besides being a hidden category, its values vary from 1985 to 2009 . It must be converted to how old the store is to better see the impact on sales.

Need of Data Preprocessing
• For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
• Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen.

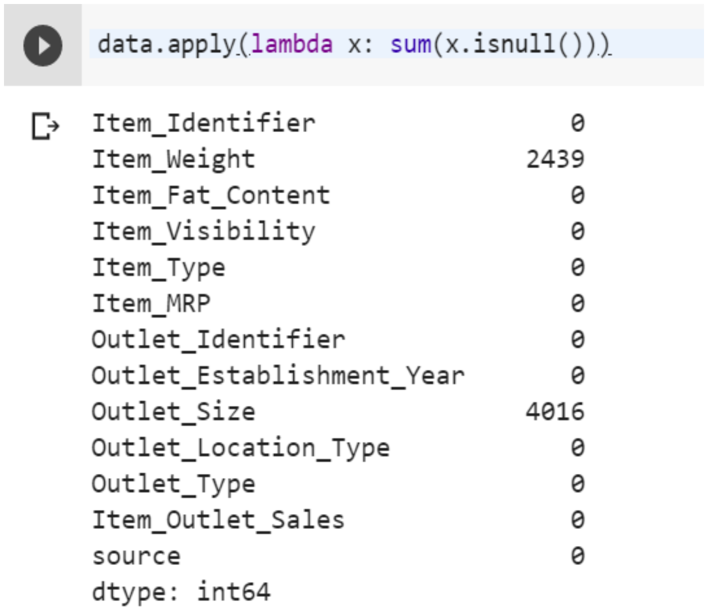
i. Finding Missing Values

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real life scenario. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different user being surveyed may choose not to share their income, some user may choose not to share the address in this way many datasets went missing.

To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()
- dropna()
- fillna()
- replace()
- interpolate()

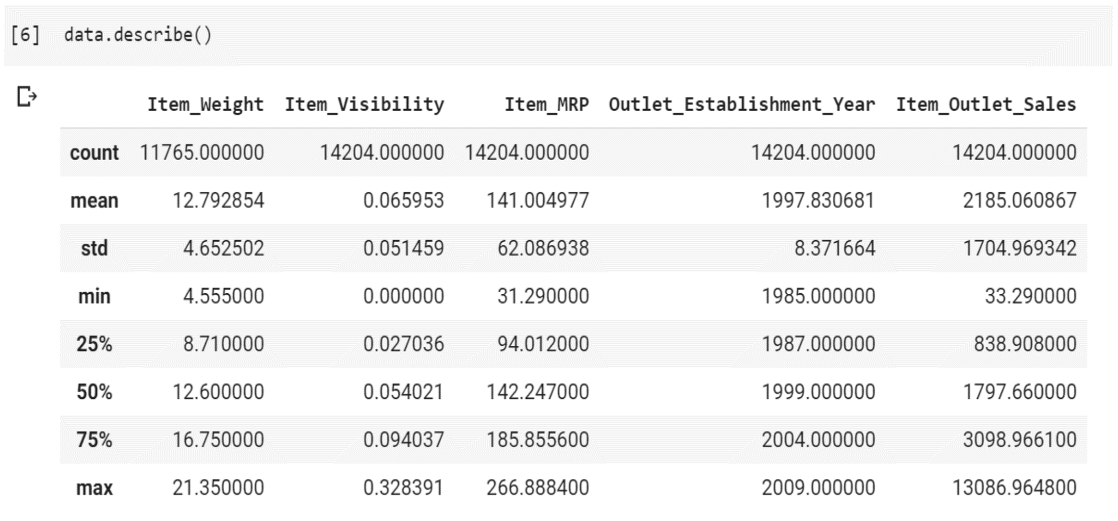
To check for sum of the missing values



```
data.apply(lambda x: sum(x.isnull()))
```

```
Item_Identifier                 0
Item_Weight                  2439
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  4016
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
source                          0
dtype: int64
```

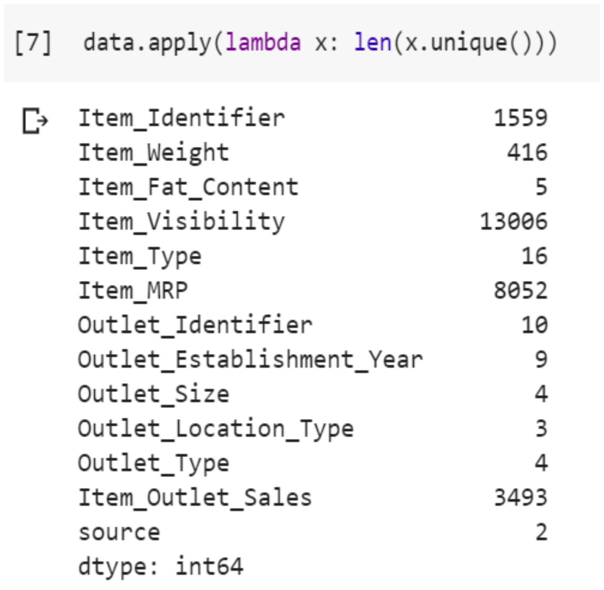
Describe the dataset



```
[6]  data.describe()
```

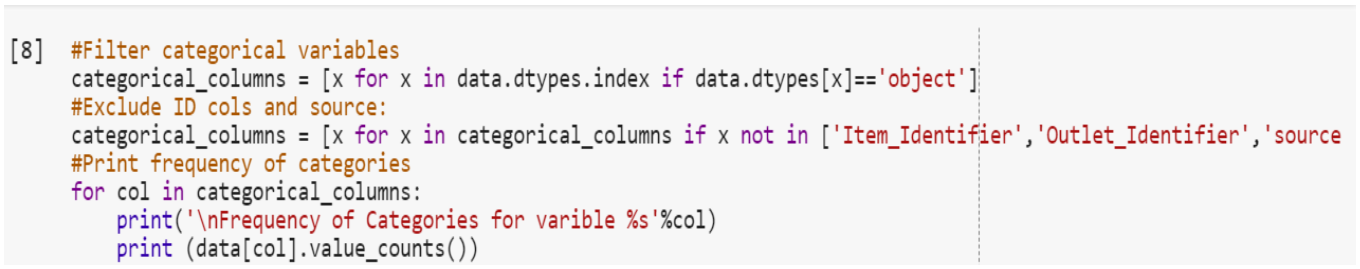
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	11765.000000	14204.000000	14204.000000	14204.000000	14204.000000
mean	12.792854	0.065953	141.004977	1997.830681	2185.060867
std	4.652502	0.051459	62.086938	8.371664	1704.969342
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.710000	0.027036	94.012000	1987.000000	838.908000
50%	12.600000	0.054021	142.247000	1999.000000	1797.660000
75%	16.750000	0.094037	185.855600	2004.000000	3098.966100
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

The number of unique values in each nominal (categorical) variable

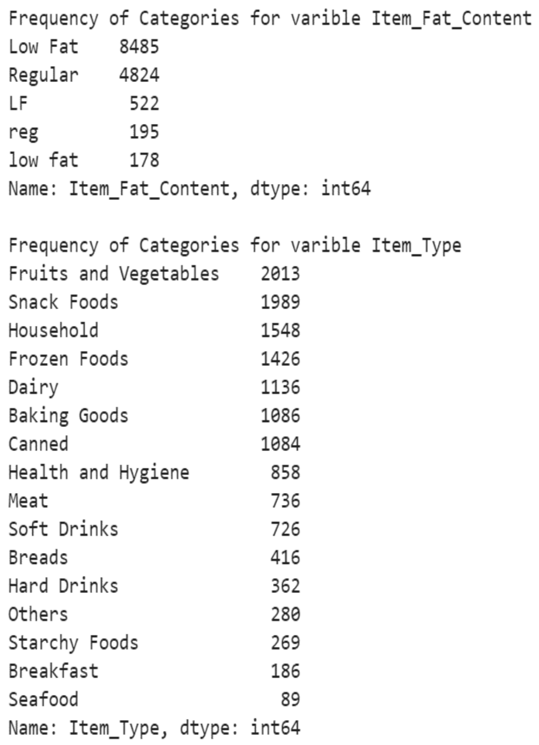


```
[7]  data.apply(lambda x: len(x.unique()))

     Item_Identifier               1559
     Item_Weight                    416
     Item_Fat_Content                 5
     Item_Visibility              13006
     Item_Type                       16
     Item_MRP                      8052
     Outlet_Identifier               10
     Outlet_Establishment_Year        9
     Outlet_Size                      4
     Outlet_Location_Type             3
     Outlet_Type                      4
     Item_Outlet_Sales             3493
     source                           2
     dtype: int64
```

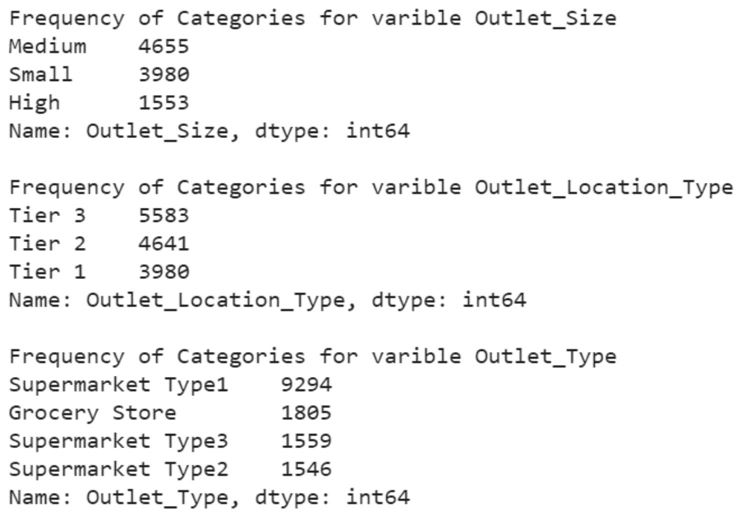
Frequency of different categories in each nominal variable.



```
[8]  #Filter categorical variables
     categorical_columns = [x for x in data.dtypes.index if data.dtypes[x]=='object']
     #Exclude ID cols and source:
     categorical_columns = [x for x in categorical_columns if x not in ['Item_Identifier','Outlet_Identifier','source
     #Print frequency of categories
     for col in categorical_columns:
         print('\nFrequency of Categories for varible %s'%col)
         print (data[col].value_counts())
```



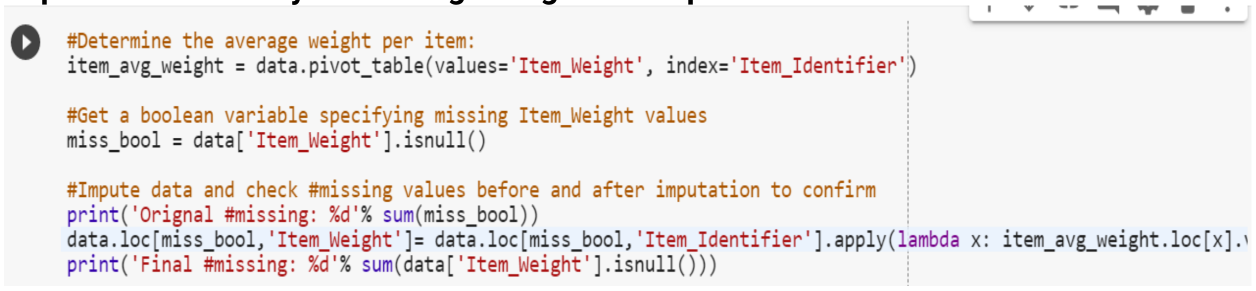
```
Frequency of Categories for varible Item_Fat_Content
Low Fat    8485
Regular    4824
LF          522
reg         195
low fat     178
Name: Item_Fat_Content, dtype: int64

Frequency of Categories for varible Item_Type
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
```



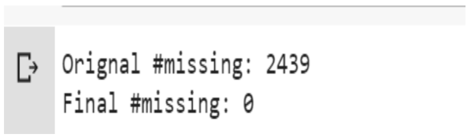
```
Frequency of Categories for varible Outlet_Size
Medium    4655
Small     3980
High      1553
Name: Outlet_Size, dtype: int64

Frequency of Categories for varible Outlet_Location_Type
Tier 3    5583
Tier 2    4641
Tier 1    3980
Name: Outlet_Location_Type, dtype: int64

Frequency of Categories for varible Outlet_Type
Supermarket Type1    9294
Grocery Store        1805
Supermarket Type3    1559
Supermarket Type2    1546
Name: Outlet_Type, dtype: int64
```

ii. Imputing missing values using mean and mode

We found two variables with missing values – Item_Weight and Outlet_Size. Lets impute the former by the average weight of the particular item.

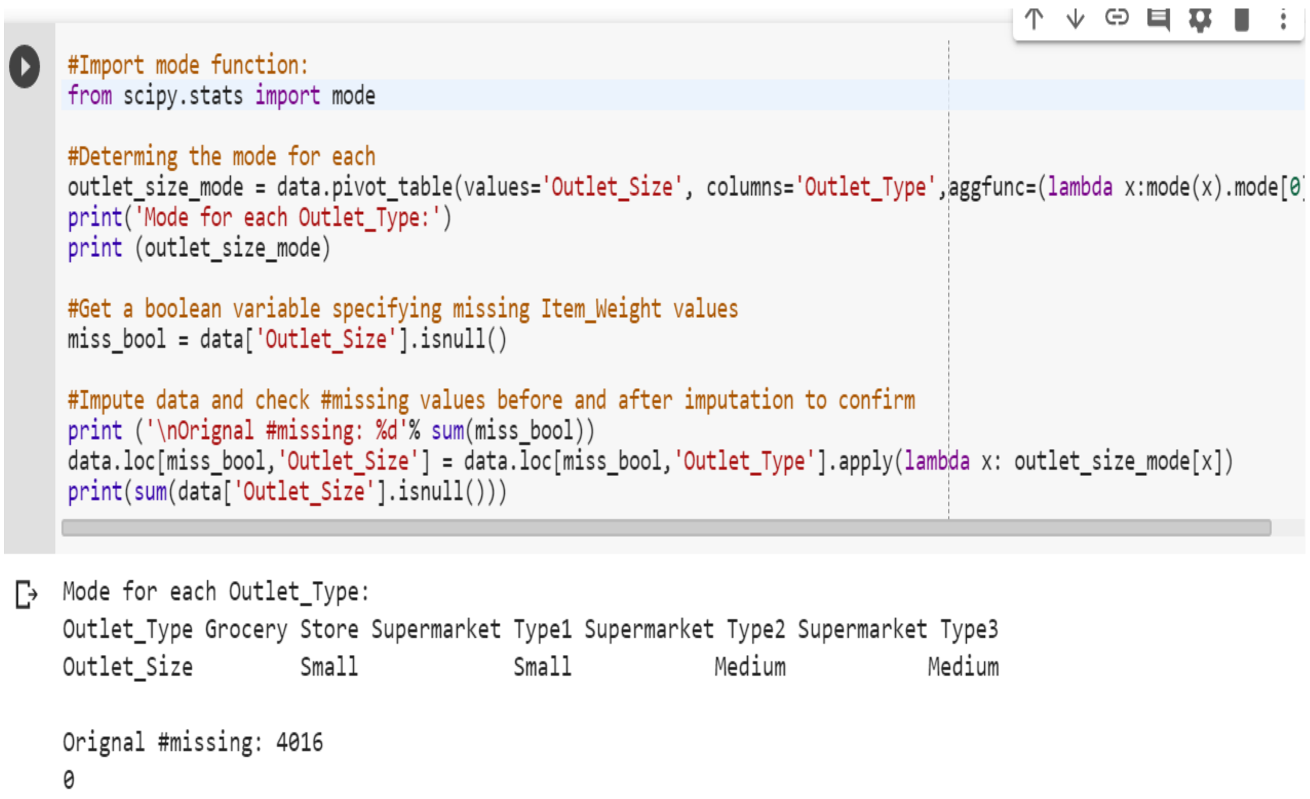


```python
#Determine the average weight per item:
item_avg_weight = data.pivot_table(values='Item_Weight', index='Item_Identifier')

#Get a boolean variable specifying missing Item_Weight values
miss_bool = data['Item_Weight'].isnull()

#Impute data and check #missing values before and after imputation to confirm
print('Orignal #missing: %d'% sum(miss_bool))
data.loc[miss_bool,'Item_Weight']= data.loc[miss_bool,'Item_Identifier'].apply(lambda x: item_avg_weight.loc[x].
print('Final #missing: %d'% sum(data['Item_Weight'].isnull()))
```



```
Orignal #missing: 2439
Final #missing: 0
```

impute Outlet_Size with the mode of the Outlet_Size for the particular type of outlet.

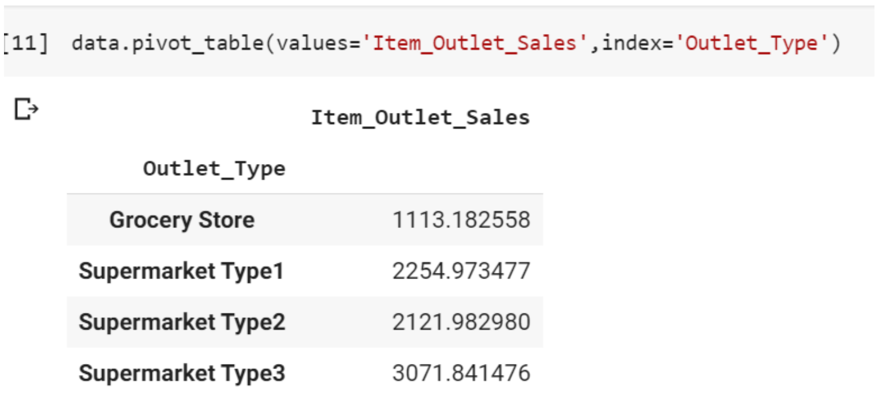


```
#Import mode function:
from scipy.stats import mode

#Determing the mode for each
outlet_size_mode = data.pivot_table(values='Outlet_Size', columns='Outlet_Type',aggfunc=(lambda x:mode(x).mode[0
print('Mode for each Outlet_Type:')
print (outlet_size_mode)

#Get a boolean variable specifying missing Item_Weight values
miss_bool = data['Outlet_Size'].isnull()

#Impute data and check #missing values before and after imputation to confirm
print ('\nOrignal #missing: %d'% sum(miss_bool))
data.loc[miss_bool,'Outlet_Size'] = data.loc[miss_bool,'Outlet_Type'].apply(lambda x: outlet_size_mode[x])
print(sum(data['Outlet_Size'].isnull()))
```

```
Mode for each Outlet_Type:
Outlet_Type Grocery Store Supermarket Type1 Supermarket Type2 Supermarket Type3
Outlet_Size         Small             Small            Medium            Medium

Orignal #missing: 4016
0
```

iii. Standardizing the dataset

Combining Outlet Type



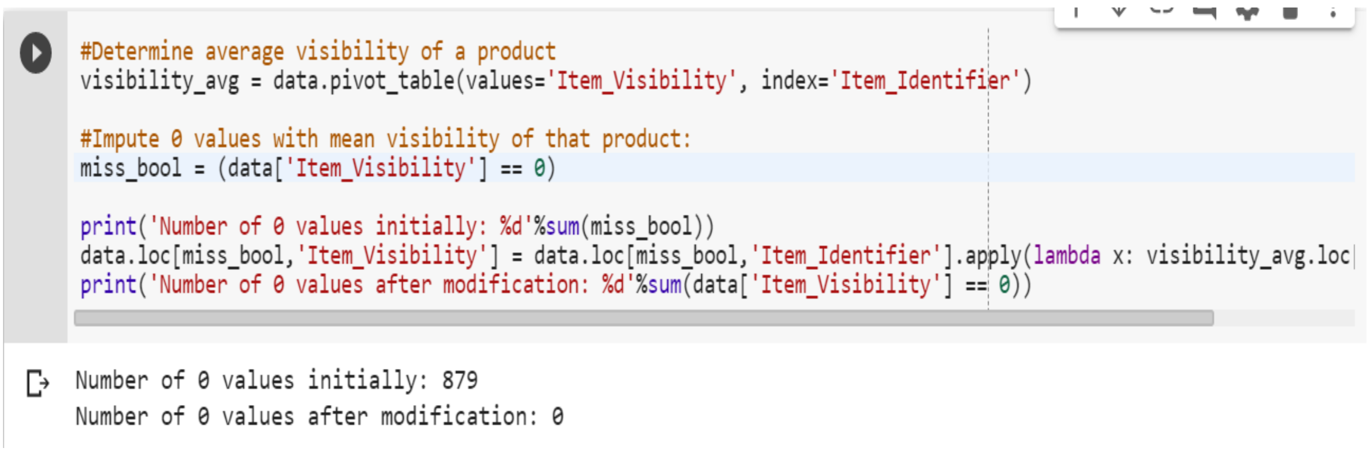
```
[11] data.pivot_table(values='Item_Outlet_Sales',index='Outlet_Type')
```

	Item_Outlet_Sales
Outlet_Type	
Grocery Store	1113.182558
Supermarket Type1	2254.973477
Supermarket Type2	2121.982980
Supermarket Type3	3071.841476

This shows significant difference between them so we'll leave them as it is.

Modify Item_Visibility

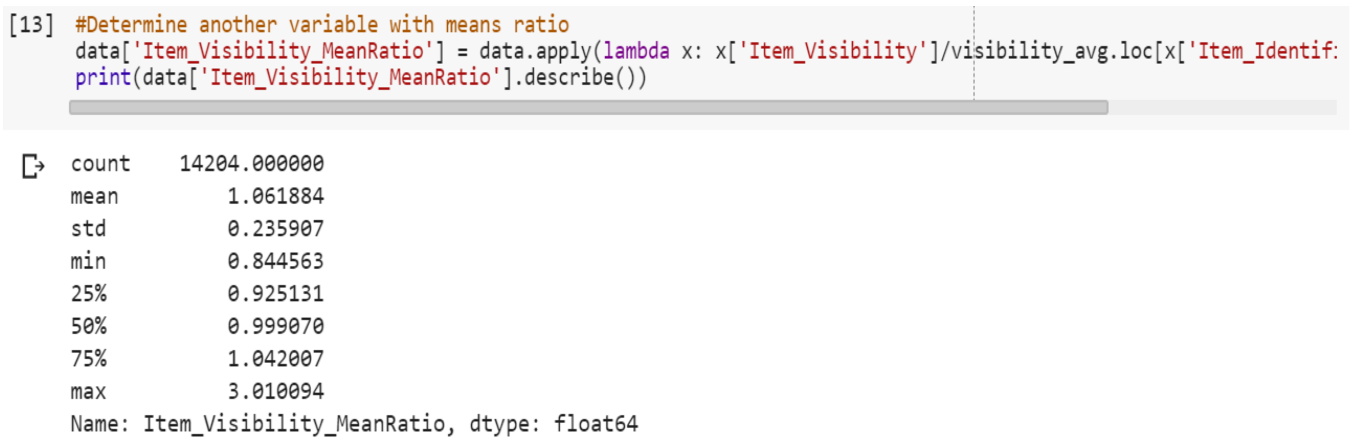
We noticed that the minimum value here is 0, which makes no practical sense. Lets consider it like missing information and impute it with mean visibility of that product.



```
#Determine average visibility of a product
visibility_avg = data.pivot_table(values='Item_Visibility', index='Item_Identifier')

#Impute 0 values with mean visibility of that product:
miss_bool = (data['Item_Visibility'] == 0)

print('Number of 0 values initially: %d'%sum(miss_bool))
data.loc[miss_bool,'Item_Visibility'] = data.loc[miss_bool,'Item_Identifier'].apply(lambda x: visibility_avg.loc
print('Number of 0 values after modification: %d'%sum(data['Item_Visibility'] == 0))
```

```
Number of 0 values initially: 879
Number of 0 values after modification: 0
```

Products with higher visibility are likely to sell more. But along with comparing products on absolute terms, we should look at the visibility of the product in that particular store as compared to the mean visibility of that product across all stores. This will give some idea about how much importance was given to that product in a store as compared to other stores. We can use the 'visibility_avg' variable made above to achieve this.

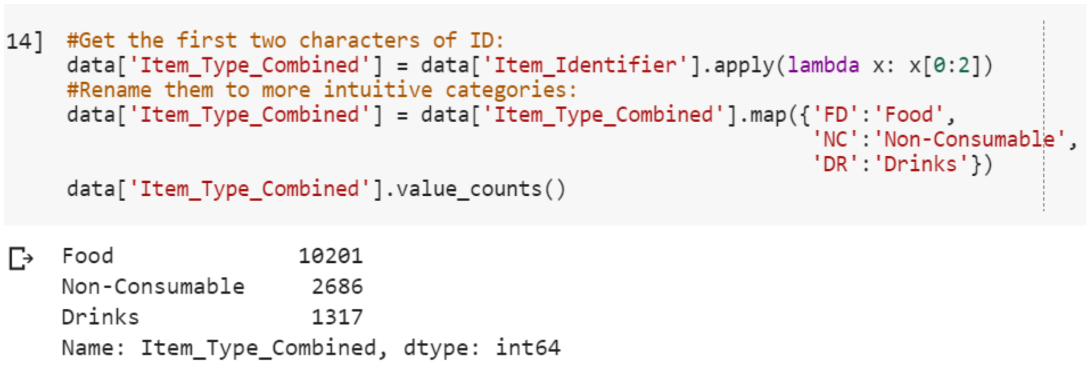


```
[13] #Determine another variable with means ratio
     data['Item_Visibility_MeanRatio'] = data.apply(lambda x: x['Item_Visibility']/visibility_avg.loc[x['Item_Identif
     print(data['Item_Visibility_MeanRatio'].describe())
```

```
count    14204.000000
mean         1.061884
std          0.235907
min          0.844563
25%          0.925131
50%          0.999070
75%          1.042007
max          3.010094
Name: Item_Visibility_MeanRatio, dtype: float64
```

Create a broad category of Type of Item

Earlier we saw that the Item_Type variable has 16 categories which might not prove to be very useful in our analysis. So it's a good idea to combine them. If we look closely to the Item_Identifier of each item we see that each one starts with either "FD" (Food), "DR" (Drinks) or "NC" (Non-Consumables). Therefore, we can group the items within these 3 categories

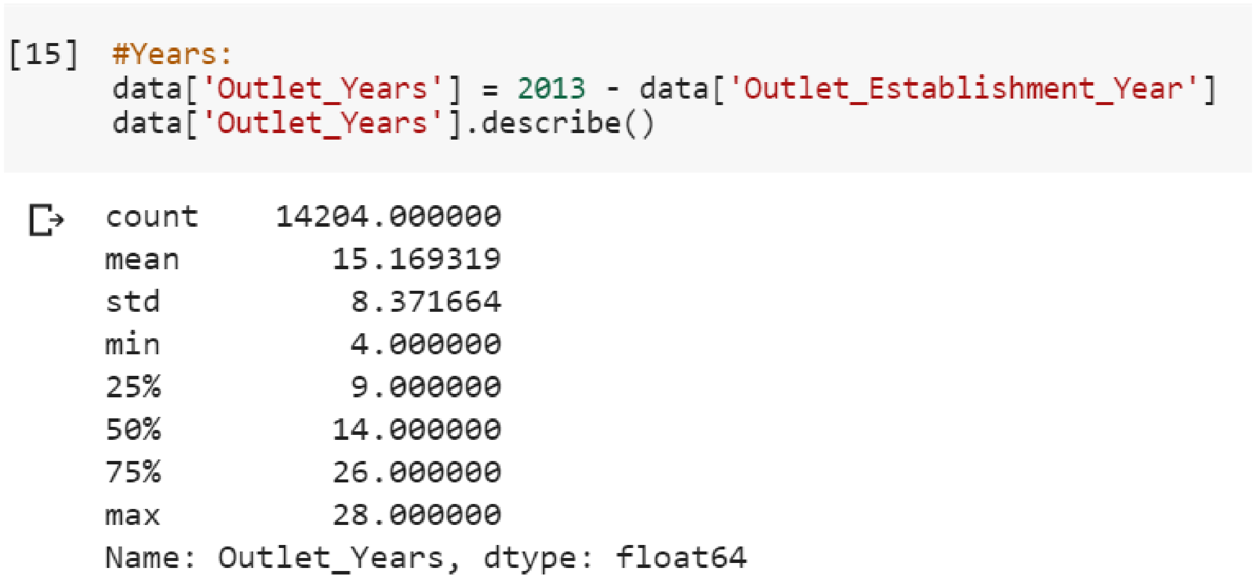


```
14]  #Get the first two characters of ID:
     data['Item_Type_Combined'] = data['Item_Identifier'].apply(lambda x: x[0:2])
     #Rename them to more intuitive categories:
     data['Item_Type_Combined'] = data['Item_Type_Combined'].map({'FD':'Food',
                                                                  'NC':'Non-Consumable',
                                                                  'DR':'Drinks'})
     data['Item_Type_Combined'].value_counts()
```

```
Food              10201
Non-Consumable     2686
Drinks             1317
Name: Item_Type_Combined, dtype: int64
```

Determine the years of operation of a store

We talked about using how long has been working instead of the date of start. Remember that the data we have is from 2013. Thus we must consider this year into our calculations-

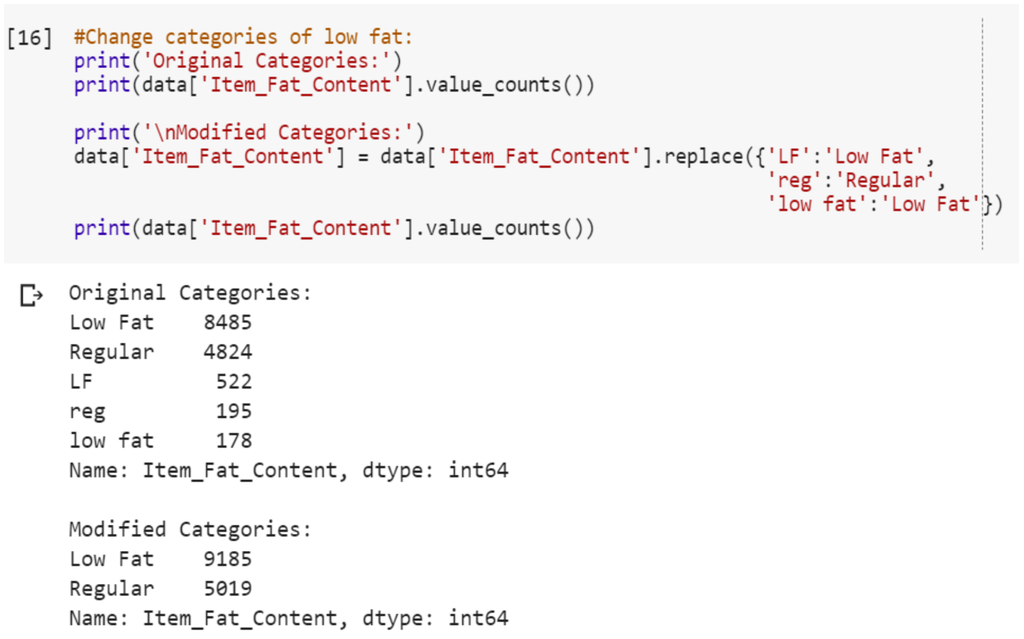


```
[15]  #Years:
      data['Outlet_Years'] = 2013 - data['Outlet_Establishment_Year']
      data['Outlet_Years'].describe()
```

```
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
```

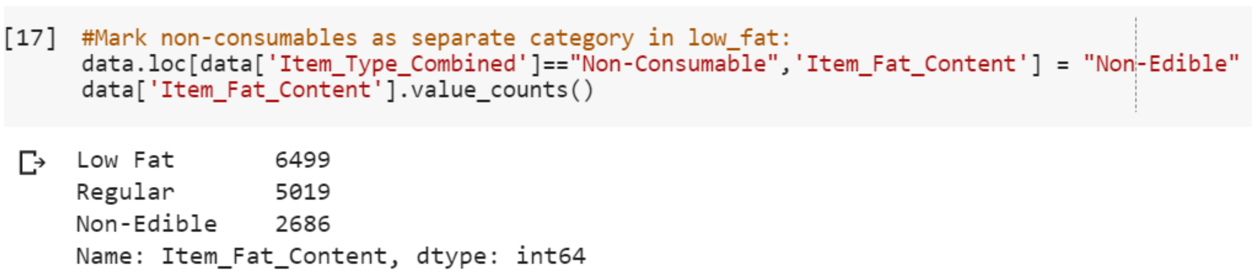
Modify categories of Item_Fat_Content

We found typos and difference in representation in categories of Item_Fat_Content variable. This can be corrected as:



```
[16] #Change categories of low fat:
     print('Original Categories:')
     print(data['Item_Fat_Content'].value_counts())

     print('\nModified Categories:')
     data['Item_Fat_Content'] = data['Item_Fat_Content'].replace({'LF':'Low Fat',
                                                               'reg':'Regular',
                                                               'low fat':'Low Fat'})
     print(data['Item_Fat_Content'].value_counts())
```

```
Original Categories:
Low Fat    8485
Regular    4824
LF          522
reg         195
low fat     178
Name: Item_Fat_Content, dtype: int64

Modified Categories:
Low Fat    9185
Regular    5019
Name: Item_Fat_Content, dtype: int64
```



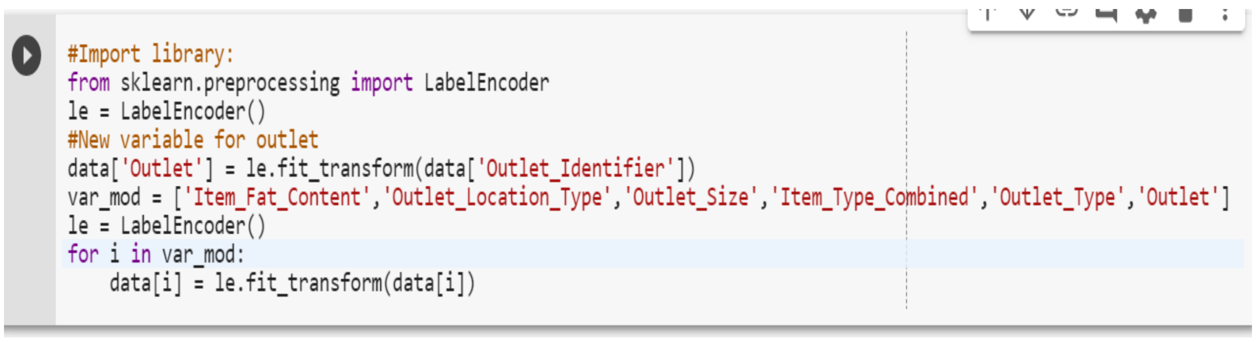
```
[17] #Mark non-consumables as separate category in low_fat:
     data.loc[data['Item_Type_Combined']=="Non-Consumable",'Item_Fat_Content'] = "Non-Edible"
     data['Item_Fat_Content'].value_counts()
```

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

iv. Label Encoding

Since scikit-learn only accepts numerical variables, we need to convert all categories of nominal variables into numeric types. Let’s start with turning all categorical variables into numerical values using LabelEncoder() (Encode labels with value between 0 and n_classes-1). After that, we can use get_dummies to generate dummy variables from these numerical categorical variables

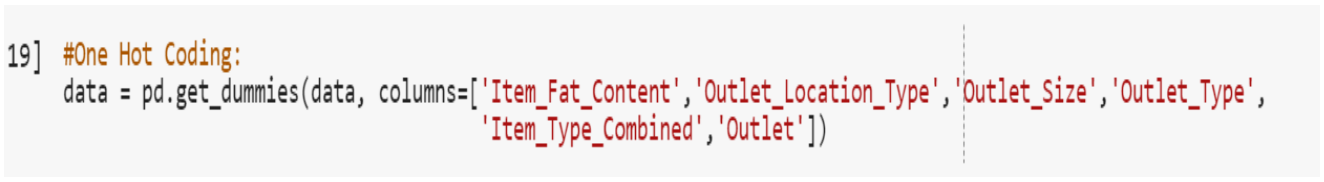
Coding all categorical variables as numeric using ‘LabelEncoder’ from sklearn’s preprocessing module



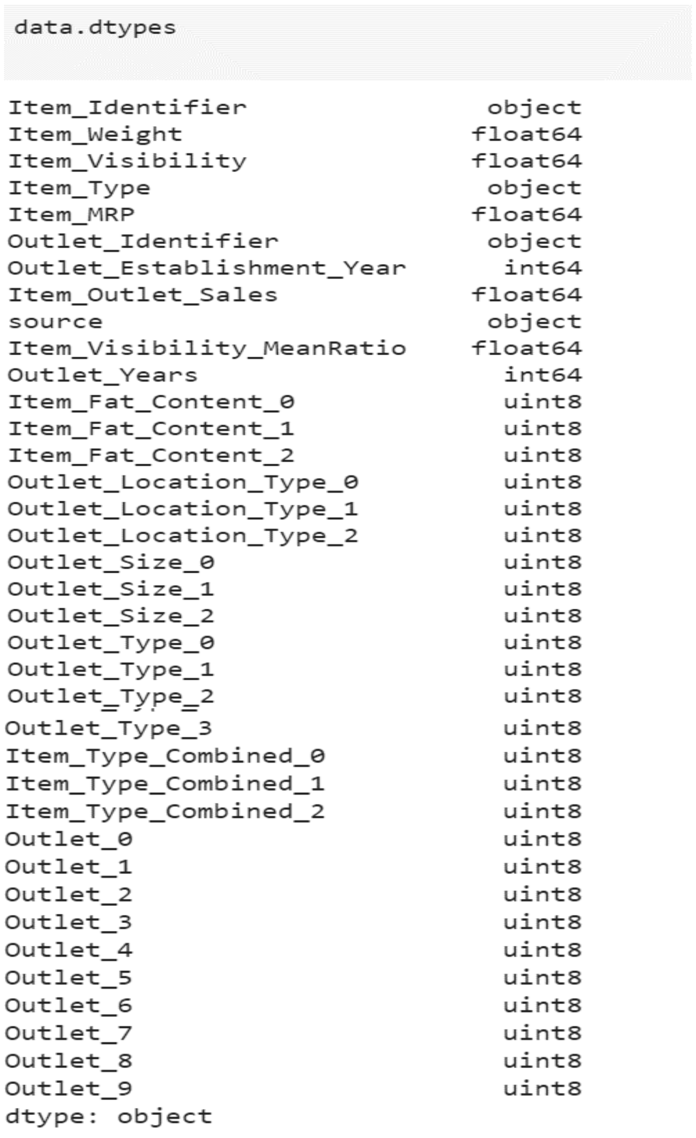
```
#Import library:
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#New variable for outlet
data['Outlet'] = le.fit_transform(data['Outlet_Identifier'])
var_mod = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Item_Type_Combined','Outlet_Type','Outlet']
le = LabelEncoder()
for i in var_mod:
    data[i] = le.fit_transform(data[i])
```

One-Hot-Coding refers to creating dummy variables, one for each category of a categorical variable. For example, the Item_Fat_Content has 3 categories — LowFat,Regular,Non-Edible. One hot coding will remove this variable and generate 3 new variables. Each will have binary numbers — 0 (if the category is not present) and 1(if category is present). This can be done using get_dummies function of Pandas.

v. One Hot Encoding



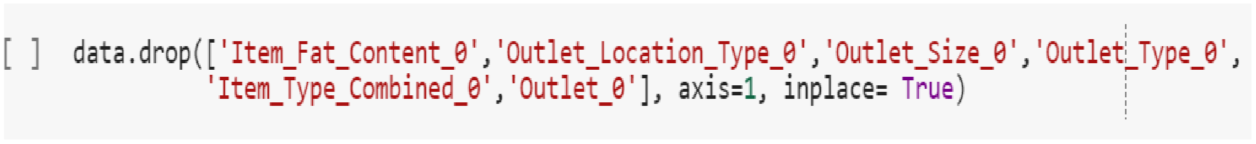
```
19] #One Hot Coding:
    data = pd.get_dummies(data, columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Outlet_Type',
                                         'Item_Type_Combined','Outlet'])
```



```
data.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
Item_Visibility_MeanRatio    float64
Outlet_Years                   int64
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
Outlet_Type_0                  uint8
Outlet_Type_1                  uint8
Outlet_Type_2                  uint8
Outlet_Type_3                  uint8
Item_Type_Combined_0           uint8
Item_Type_Combined_1           uint8
Item_Type_Combined_2           uint8
Outlet_0                       uint8
Outlet_1                       uint8
Outlet_2                       uint8
Outlet_3                       uint8
Outlet_4                       uint8
Outlet_5                       uint8
Outlet_6                       uint8
Outlet_7                       uint8
Outlet_8                       uint8
Outlet_9                       uint8
dtype: object
```

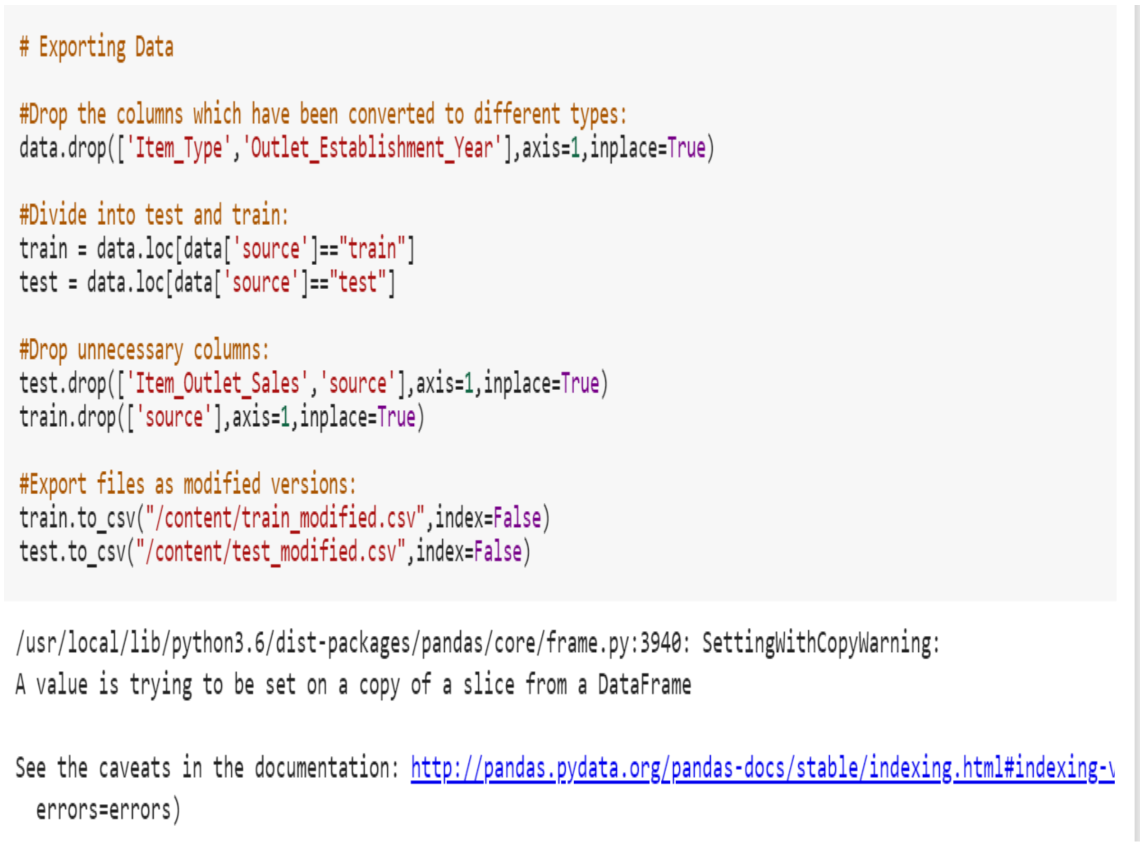
Dropping unwanted columns



```
[ ]  data.drop(['Item_Fat_Content_0','Outlet_Location_Type_0','Outlet_Size_0','Outlet_Type_0',
               'Item_Type_Combined_0','Outlet_0'], axis=1, inplace= True)
```

4. Exporting data

Final step is to convert data back into train and test data sets. Its generally a good idea to export both of these as modified data sets so that they can be re-used for multiple sessions. This can be achieved using following code:



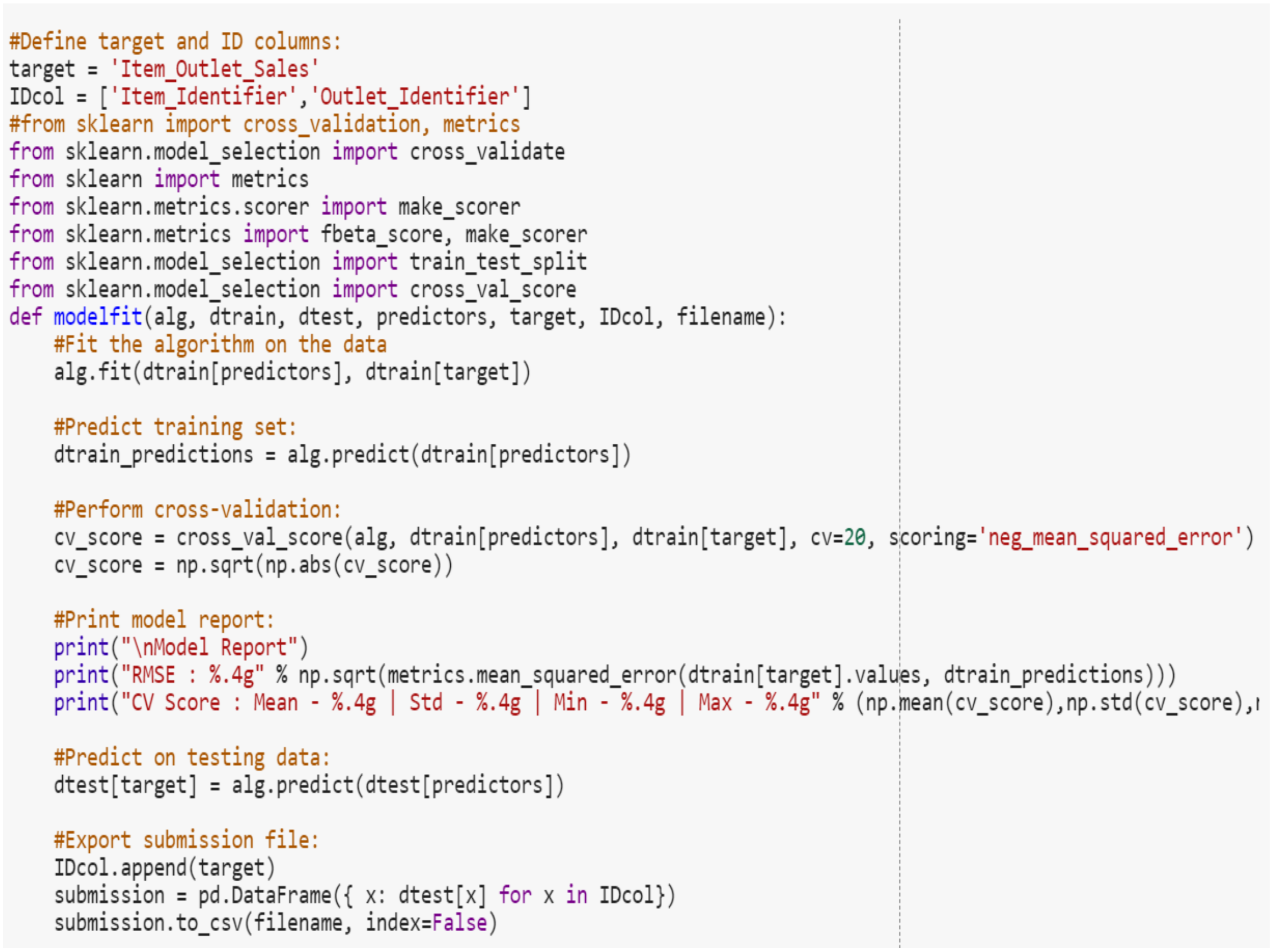
```
# Exporting Data

#Drop the columns which have been converted to different types:
data.drop(['Item_Type','Outlet_Establishment_Year'],axis=1,inplace=True)

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

#Drop 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("/content/train_modified.csv",index=False)
test.to_csv("/content/test_modified.csv",index=False)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3940: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame


See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-
  errors=errors)
```

5. Model Building and validation

Now that we have the data ready, its time to start making predictive models, through 6 models including linear regression, decision tree and random forest which can get you into Top 20 ranks in this competition

Lets start by making a **baseline model**. Baseline model is the one which requires no predictive model and its like an informed guess. For instance, in this case lets predict the sales as the overall average sales. This can be done as:



```
#Define target and ID columns:
target = 'Item_Outlet_Sales'
IDcol = ['Item_Identifier','Outlet_Identifier']
#from sklearn import cross_validation, metrics
from sklearn.model_selection import cross_validate
from sklearn import metrics
from sklearn.metrics.scorer import make_scorer
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
def modelfit(alg, dtrain, dtest, predictors, target, IDcol, filename):
    #Fit the algorithm on the data
    alg.fit(dtrain[predictors], dtrain[target])

    #Predict training set:
    dtrain_predictions = alg.predict(dtrain[predictors])

    #Perform cross-validation:
    cv_score = cross_val_score(alg, dtrain[predictors], dtrain[target], cv=20, scoring='neg_mean_squared_error')
    cv_score = np.sqrt(np.abs(cv_score))

    #Print model report:
    print("\nModel Report")
    print("RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(dtrain[target].values, dtrain_predictions)))
    print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_score),np.std(cv_score),

    #Predict on testing data:
    dtest[target] = alg.predict(dtest[predictors])

    #Export submission file:
    IDcol.append(target)
    submission = pd.DataFrame({ x: dtest[x] for x in IDcol})
    submission.to_csv(filename, index=False)
```

Cross Validation is a technique which involves reserving a particular sample of a dataset on which you do not train the model. Later, you test your model on this sample before finalizing it.
Here are the steps involved in cross validation:

- You *reserve* a sample data set
- Train the model using the remaining part of the dataset
- Use the reserve sample of the test (validation) set. This will help you in gauging the effectiveness of your model's performance. If your model delivers a positive result on validation data

6. Prediction and Visualization

i. Linear Regression Model

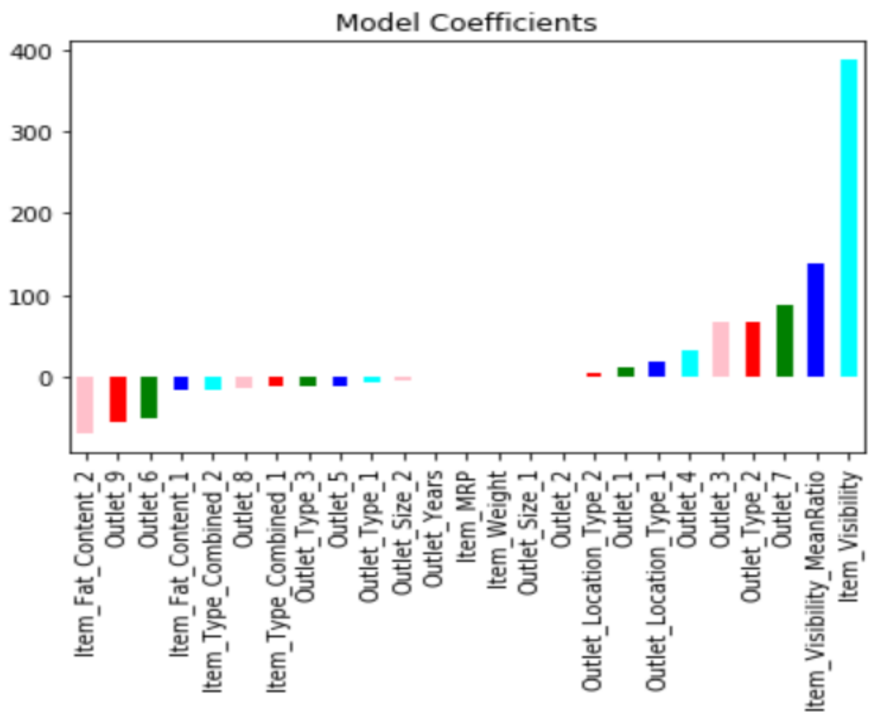
The basic idea of this algorithm is to fit a straight line between the selected features in training dataset and a continuous target variable i.e. sales. This algorithm finds a line that best fits the data. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). The equation of the regression line is represented by:

h(Xi)=B0+B1Xi

where h(Xi) is the expected value for ith observation.

```
from sklearn.linear_model import LinearRegression, Ridge
predictors = [x for x in train.columns if x not in [target]+IDcol]
# print predictors
alg1 = LinearRegression(normalize=True)
modelfit(alg1, train, test, predictors, target, IDcol, 'alg1.csv')
coef1 = pd.Series(alg1.coef_, predictors).sort_values()
coef1.plot(kind='bar', title='Model Coefficients',color=['pink', 'red', 'green', 'blue', 'cyan'])
```

```
Model Report
RMSE : 1701
CV Score : Mean - 1703 | Std - 94.13 | Min - 1536 | Max - 1863
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-v
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9e6af9438>
```

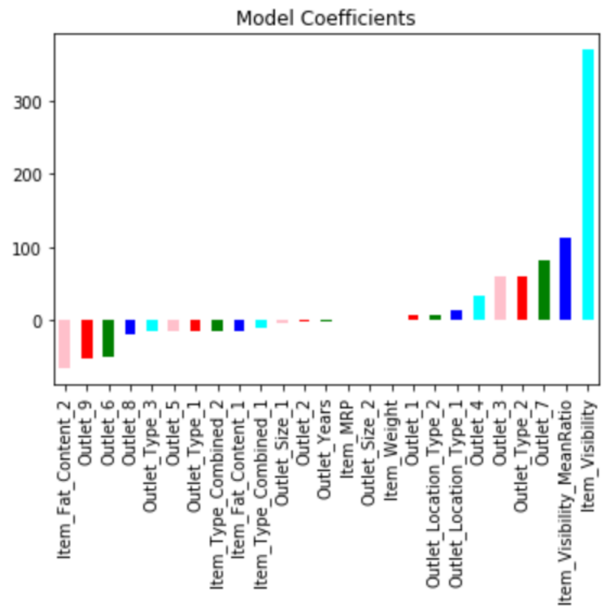


ii. Ridge Regression

It is a technique for analyzing multiple regression data. When multicollinearity occurs, least squares estimates are unbiased. A degree of bias is added to the regression estimates, and a result, ridge regression reduces the standard errors.

```
predictors = [x for x in train.columns if x not in [target]+IDcol]
alg2 = Ridge(alpha=0.05,normalize=True)
modelfit(alg2, train, test, predictors, target, IDcol, 'alg2.csv')
coef2 = pd.Series(alg2.coef_, predictors).sort_values()
coef2.plot(kind='bar', title='Model Coefficients',color=['pink', 'red', 'green', 'blue', 'cyan'])
```

```
Model Report
RMSE : 1701
CV Score : Mean - 1703 | Std - 94.16 | Min - 1535 | Max - 1862
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9e3bcaf28>
```

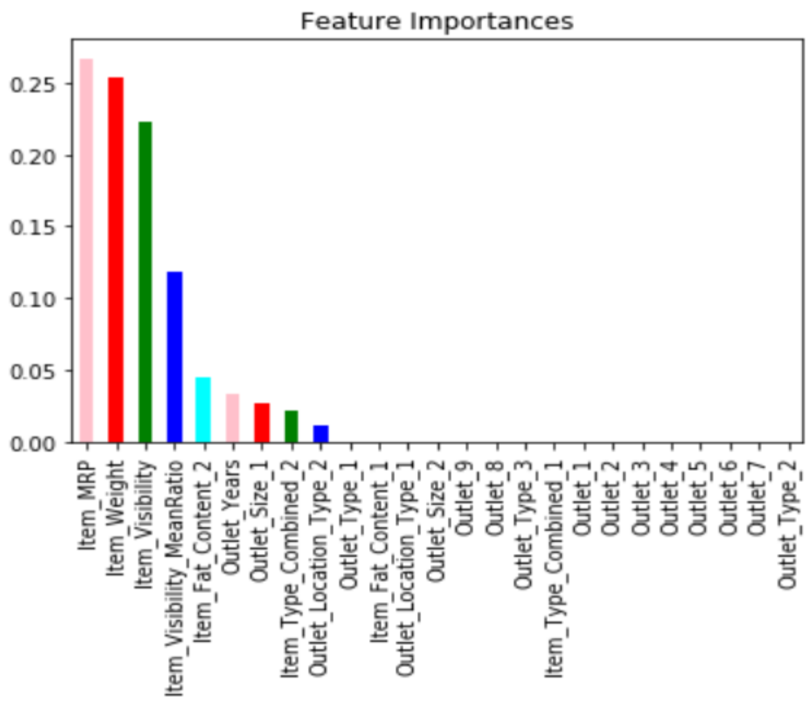


iii. Decision Tree

The **Decision Tree Analysis** is a schematic representation of several decisions followed by different chances of the occurrence. Simply, a tree-shaped graphical representation of decisions related to the investments and the chance points that help to investigate the possible outcomes is called as a decision tree analysis.

```python
from sklearn.tree import DecisionTreeRegressor
predictors = [x for x in train.columns if x not in [target]+IDcol]
alg3 = DecisionTreeRegressor(max_depth=15, min_samples_leaf=100)
modelfit(alg3, train, test, predictors, target, IDcol, 'alg3.csv')
coef3 = pd.Series(alg3.feature_importances_, predictors).sort_values(ascending=False)
coef3.plot(kind='bar', title='Feature Importances',color=['pink', 'red', 'green', 'blue', 'cyan'])
```

```
Model Report
RMSE : 1670
CV Score : Mean - 1735 | Std - 99.38 | Min - 1567 | Max - 1880
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

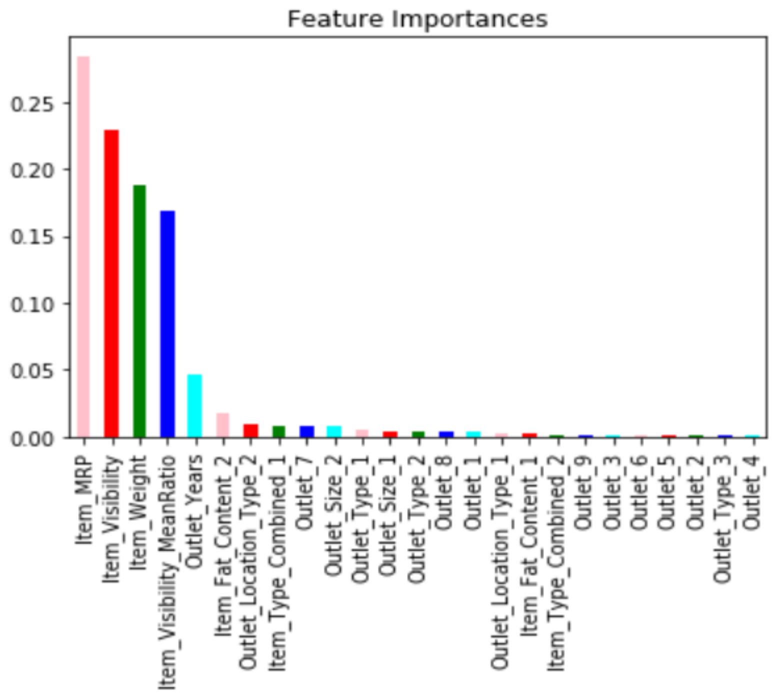


iv. Random Forest model

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as **bagging**. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

```python
from sklearn.ensemble import RandomForestRegressor
predictors = [x for x in train.columns if x not in [target]+IDcol]
alg5 = RandomForestRegressor(n_estimators=200,max_depth=5, min_samples_leaf=100,n_jobs=4)
modelfit(alg5, train, test, predictors, target, IDcol, 'alg5.csv')
coef5 = pd.Series(alg5.feature_importances_, predictors).sort_values(ascending=False)
coef5.plot(kind='bar', title='Feature Importances',color=['pink', 'red', 'green', 'blue', 'cyan'])
```

```
Model Report
RMSE : 1683
CV Score : Mean - 1702 | Std - 93.82 | Min - 1537 | Max - 1857
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9e1be69b0>
```



7. Conclusion

In the end, the Decision Tree Regressor ensemble was used to make the final predictions for Item Outlet Sales at Big Mart. This model produced the lowest RMSE. Therefore, it should be the model that will make the best predictions.

There were other conclusions that can be made from this report's analysis. First there is a moderate correlation between an Item's MRP at a Big Mart location and that item's sales at that location. Also the smallest locations produced the lowest sales. However, the largest location did not produce the highest sales. The location that produced the highest sales was the OUT027 location. This location was Supermarket Type3 and its size was medium. This outlet performed much better than any other location. Its median Item_Outlet_Sales were 3364.95. The location that was second was the OUT035 location, which had a median Item_Outlet_Sales of 2109.25.

If Big Mart were to try to increase sales at all locations, it may consider switching more locations to Supermarket Type3. Other things Big Mart could do to increase sales is to see which Items had the highest sales. They may also consider how product visibility affected outlet sales. However, the model built in this report should be good for helping Big Mart predict future sales at its locations.

References

https://www.analyticsvidhya.com/blog/2016/02/bigmart-sales-solution-top-20/

https://colab.research.google.com/drive/1mo0MkF4JeAkTjY80tJPS4JKaATUH5-cX#scrollTo=XEE826lRkhIj

https://rpubs.com/C_Westby1984/344479

https://mc.ai/project-1-bigmart-sale-prediction/

https://github.com/aman1002/BigMart-Sales-Prediction/blob/master/bigmart.ipynb

https://www.youtube.com/watch?v=ZnvurTaXZVY

https://link.springer.com/chapter/10.1007/978-1-4842-2614-8_5

https://www.analyticsvidhya.com/blog/2016/02/bigmart-sales-solution-top-20/

https://mef-bda503.github.io/gpj-datamunglers-2/files/20171218_BigMart_Ece.html

http://rstudio-pubs-static.s3.amazonaws.com/376328_f9d8432b815d4bfa97740c48f50f3149.html

https://medium.com/diogo-menezes-borges/project-1-bigmart-sale-prediction-fdc04f07dc1e

https://rpubs.com/Pragi/finalproject1

https://mc.ai/project-1-bigmart-sale-prediction/