



[content/uploads/2016/02/Comp-4.jpg\)](https://www.analyticsvidhya.com/wp-content/uploads/2016/02/Comp-4.jpg)

We will explore the problem in following stages:

1. **Hypothesis Generation** – understanding the problem better by brainstorming possible factors that can impact the outcome
2. **Data Exploration** – looking at categorical and continuous feature summaries and making inferences about the data.
3. **Data Cleaning** – imputing missing values in the data and checking for outliers
4. **Feature Engineering** – modifying existing variables and creating new ones for analysis
5. **Model Building** – making predictive models on the data

You can also check out a full hands-on solution to this practice problem on our [Trainings platform](https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+BigMS01+2018_1/about) ([https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+BigMS01+2018\\_1/about](https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+BigMS01+2018_1/about)).

Without further ado, lets get started!

## 1. Hypothesis Generation

This is a very pivotal step in the process of analyzing data. This involves understanding the problem and making some hypothesis about what could potentially have a good impact on the outcome. This is done BEFORE looking at the data, and we end up creating a laundry list of the different analysis which we can potentially perform if data is available. Read more about hypothesis generation [here](https://www.analyticsvidhya.com/blog/2015/09/hypothesis-testing-explained/) (<https://www.analyticsvidhya.com/blog/2015/09/hypothesis-testing-explained/>).

## The Problem Statement

Understanding the problem statement is the first and foremost step. You can view this in the [competition page](http://datahack.analyticsvidhya.com/contest/practice-problem-big-mart-sales-iii) (<http://datahack.analyticsvidhya.com/contest/practice-problem-big-mart-sales-iii>) but I'll iterate the same here:

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

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

So the idea is to find out the properties of a product, and store which impacts the sales of a product. Let's think about some of the analysis that can be done and come up with certain hypothesis.

## **The Hypotheses**

I came up with the following hypothesis while thinking about the problem. These are just my thoughts and you can come-up with many more of these. Since we're talking about stores and products, lets make different sets for each.

### **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.

These are just some basic 15 hypothesis I have made, but you can think further and create some of your own. Remember that the data might not be sufficient to test all of these, but forming these gives us a better understanding of the problem and we can even look for open source information if available.

Lets move on to the data exploration where we will have a look at the data in detail.

## 2. Data Exploration

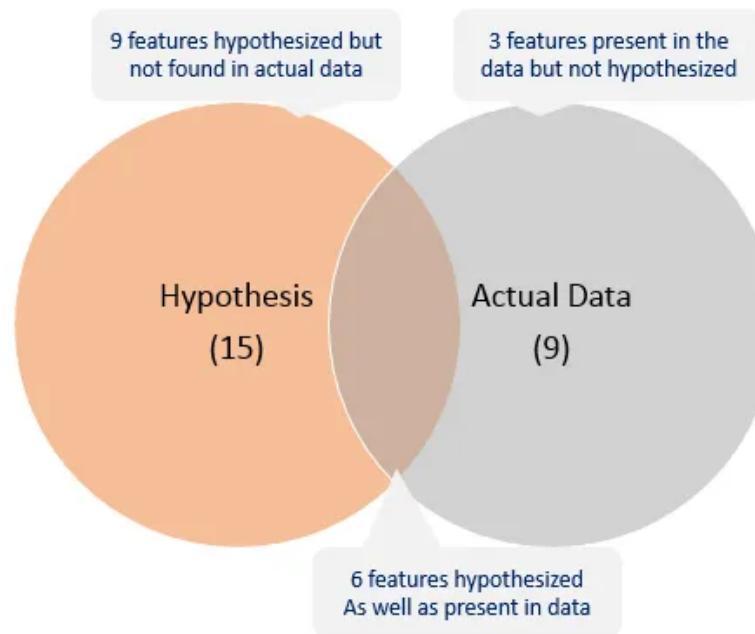
We'll be performing some basic data exploration here and come up with some inferences about the data. We'll try to figure out some irregularities and address them in the next section. If you are new to this domain, please refer our [Data Exploration Guide](https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/) (<https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/>).

The first step is to look at the data and try to identify the information which we hypothesized vs the available data. A comparison between the data dictionary on the competition page and our hypotheses is shown below:

Variable	Description	Relation to Hypothesis
Item_Identifier	Unique product ID	ID Variable
Item_Weight	Weight of product	Not considered in hypothesis
Item_Fat_Content	Whether the product is low fat or not	Linked to 'Utility' hypothesis. Low fat items are generally used more than others
Item_Visibility	The % of total display area of all products in a store allocated to the particular product	Linked to 'Display Area' hypothesis. More inferences about 'Utility' can be derived from this.
Item_Type	The category to which the product belongs	Not considered in hypothesis
Item_MRP	Maximum Retail Price (list price) of the product	ID Variable
Outlet_Identifier	Unique store ID	Not considered in hypothesis
Outlet_Establishment_Year	The year in which store was established	Linked to 'Store Capacity' hypothesis
Outlet_Size	The size of the store in terms of ground area covered	Linked to 'City Type' hypothesis.
Outlet_Location_Type	The type of city in which the store is located	Linked to 'Store Capacity' hypothesis again.
Outlet_Type	Whether the outlet is just a grocery store or some sort of supermarket	Outcome variable
Item_Outlet_Sales	Sales of the product in the particular store. This is the outcome variable to be predicted.	

(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/0.-data-dictionary-1.png>).

We can summarize the findings as:



(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/venn.png>).

You will invariably find features which you hypothesized, but data doesn't carry and vice versa. You should look for open source data to fill the gaps if possible. Let's start by loading the required libraries and data. You can download the data from the [competition page](http://datahack.analyticsvidhya.com/contest/practice-problem-bigmart-sales-prediction) (<http://datahack.analyticsvidhya.com/contest/practice-problem-bigmart-sales-prediction>).

```

import pandas as pd
import numpy as np

#Read files:
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

```

It's generally a good idea to combine both train and test data sets into one, perform feature engineering and then divide them later again. This saves the trouble of performing the same steps twice on test and train. Lets combine them into a dataframe 'data' with a 'source' column specifying where each observation belongs.

```

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

```

(8523, 13) (5681, 12) (14204, 13) [\(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/1.-shape.png\)](https://www.analyticsvidhya.com/wp-content/uploads/2016/02/1.-shape.png)

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.

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

Item_Fat_Content	0
Item_Identifier	0
Item_MRP	0
Item_Outlet_Sales	5681
Item_Type	0
Item_Visibility	0
Item_Weight	2439
Outlet_Establishment_Year	0
Outlet_Identifier	0
Outlet_Location_Type	0
Outlet_Size	4016
Outlet_Type	0
source	0
<b>dtype:</b>	<b>int64</b>

[\(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/2.-missing.png\)](https://www.analyticsvidhya.com/wp-content/uploads/2016/02/2.-missing.png)

Note that the Item\_Outlet\_Sales is the target variable and missing values are ones in the test set. So we need not worry about it. But we'll impute the missing values in Item\_Weight and Outlet\_Size in the data cleaning section.

Lets look at some basic statistics for numerical variables.

```
data.describe()
```

Enroll Now

	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Establishment_Year
count	14204.000000	8523.000000	14204.000000	11765.000000	14204.000000
mean	141.004977	2181.288914	0.065953	12.792854	1997.830681
std	62.086938	1706.499616	0.051459	4.652502	8.371664
min	31.290000	33.290000	0.000000	4.555000	1985.000000
25%	94.012000	834.247400	0.027036	8.710000	1987.000000
50%	142.247000	1794.331000	0.054021	12.600000	1999.000000
75%	185.855600	3101.296400	0.094037	16.750000	2004.000000
max	266.888400	13086.964800	0.328391	21.350000	2009.000000

(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/3.-describe.png>).

Some observations:

1. **Item\_Visibility** has a min value of zero. This makes no practical sense because when a product is being sold in a store, the visibility cannot be 0.
2. **Outlet\_Establishment\_Years** vary from 1985 to 2009. The values might not be apt in this form. Rather, if we can convert them to how old the particular store is, it should have a better impact on sales.
3. The lower 'count' of Item\_Weight and Item\_Outlet\_Sales confirms the findings from the missing value check.

Moving to nominal (categorical) variable, lets have a look at the number of unique values in each of them.

```
data.apply(lambda x: len(x.unique()))
```

Item_Fat_Content	5
Item_Identifier	1559
Item_MRP	8052
Item_Outlet_Sales	3494
Item_Type	16
Item_Visibility	13006
Item_Weight	416
Outlet_Establishment_Year	9
Outlet_Identifier	10
Outlet_Location_Type	3
Outlet_Size	4
Outlet_Type	4
source	2
<b>dtype: int64</b>	

[unique.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/02/4-.unique.png))

This tells us that there are **1559 products and 10 outlets/stores** (which was also mentioned in problem statement). Another thing that should catch attention is that **Item\_Type has 16 unique values**. Let's explore further using the frequency of different categories in each nominal variable. I'll exclude the ID and source variables for obvious reasons.

```
#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 variable %s'%col
    print data[col].value_counts()
```



Enroll Now

```
Frequency of Categories for variable 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 variable 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
```

<https://www.analyticsvidhya.com/wp->

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

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

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

<content/uploads/2016/02/5-categories.png>)

The output gives us following observations:

1. **Item\_Fat\_Content**: Some of 'Low Fat' values mis-coded as 'low fat' and 'LF'. Also, some of 'Regular' are mentioned as 'regular'.
2. **Item\_Type**: Not all categories have substantial numbers. It looks like combining them can give better results.
3. **Outlet\_Type**: Supermarket Type2 and Type3 can be combined. But we should check if that's a good idea before doing it.

### 3. Data Cleaning

This step typically involves imputing missing values and treating outliers. Though outlier removal is very important in regression techniques, advanced tree based algorithms are impervious to outliers. So I'll leave it to you to try it out. We'll focus on the imputation step here, which is a very important step.

Note: We'll be using some Pandas library extensively here. If you're new to Pandas, please go through [this article](https://www.analyticsvidhya.com/blog/2016/01/12-pandas-techniques-python-data-manipulation/) (<https://www.analyticsvidhya.com/blog/2016/01/12-pandas-techniques-python-data-manipulation/>).

#### Imputing Missing Values

We found two variables with missing values – Item\_Weight and Outlet\_Size. Lets impute the former by the average weight of the particular item. This can be done as:

```
#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 'Original #missing: %d' % sum(miss_bool)  
data.loc[miss_bool, 'Item_Weight'] = data.loc[miss_bool, 'Item_Identifier'].apply(lambda x: item_avg_weight[x])  
print 'Final #missing: %d' % sum(data['Item_Weight'].isnull())
```

Original #missing: 2439

Final #missing: 0

[\\_\(https://www.analyticsvidhya.com/wp-content/uploads/2016/02/7.-imputation1.png\)](https://www.analyticsvidhya.com/wp-content/uploads/2016/02/7.-imputation1.png)

This confirms that the column has no missing values now. Lets impute Outlet\_Size with the mode of the Outlet\_Size for the particular type of outlet.



Enroll Now

```

#Import mode function:
from scipy.stats import mode

#Determining 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 '\nOriginal #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** Small

**Supermarket Type1** Small

**Supermarket Type2** Medium

**Supermarket Type3** Medium

<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/8.->

**Name: Outlet\_Size, dtype: object**

**Original #missing: 4016**

0

imputation2.png).

This confirms that there are no missing values in the data. Lets move on to feature engineering now.

## 4. Feature Engineering

We explored some nuances in the data in the data exploration section. Lets move on to resolving them and making our data ready for analysis. We will also create some new variables using the existing ones in this section.

### Step 1: Consider combining Outlet\_Type

During exploration, we decided to consider combining the Supermarket Type2 and Type3 variables. But is that a good idea? A quick way to check that could be to analyze the mean sales by type of store. If they have similar sales, then keeping them separate won't help much.

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

```

Outlet_Type
Grocery Store      339.828500
Supermarket Type1  2316.181148
Supermarket Type2  1995.498739
Supermarket Type3  3694.038558
Name: Item_Outlet_Sales, dtype: float64

```

[content/uploads/2016/02/9.-outlet-type.png\)](content/uploads/2016/02/9.-outlet-type.png)

This shows significant difference between them and we'll leave them as it is. Note that this is just one way of doing this, you can perform some other analysis in different situations and also do the same for other features.

## Step 2: 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[x])
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** [https://www.analyticsvidhya.com/wp-content/uploads/2016/02/10.-visib-1.png\)](https://www.analyticsvidhya.com/wp-content/uploads/2016/02/10.-visib-1.png)

So we can see that there are no values which are zero.

In step 1 we hypothesized that 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.

```

#Determine another variable with means ratio
data['Item_Visibility_MeanRatio'] = data.apply(lambda x: x['Item_Visibility']/visibility_avg[x['Item_Identifier']], axis=1)
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
```

([content/uploads/2016/02/11.-visb-2.png\).](https://www.analyticsvidhya.com/wp-</a></p></div><div data-bbox=)

Thus the new variable has been successfully created. Again, this is just 1 example of how to create new features. I highly encourage you to try more of these, as good features can drastically improve model performance and they invariably prove to be the difference between the best and the average model.

### Step 3: Create a broad category of Type of Item

Earlier we saw that the Item\_Type variable has 16 categories which might prove to be very useful in analysis. So its a good idea to combine them. One way could be to manually assign a new category to each. But there's a catch here. If you look at the Item\_Identifier, i.e. the unique ID of each item, it starts with either FD, DR or NC. If you see the categories, these look like being Food, Drinks and Non-Consumables. So I've used the Item\_Identifier variable to create a new column:

```
#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
```

([content/uploads/2016/02/12.-item\\_type.png\).](https://www.analyticsvidhya.com/wp-</a></p></div><div data-bbox=)

Another idea could be to combine categories based on sales. The ones with high average sales could be combined together. I leave this for you to try.

### Step 4: Determine the years of operation of a store

We wanted to make a new column depicting the years of operation of a store. This can be done as:

```
#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
```

<content/uploads/2016/02/13.-years-of-oper.png>)

This shows stores which are 4-28 years old. Notice I've used 2013. Why? Read the problem statement carefully and you'll know.

## Step 5: 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:

```
#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  https://www.analyticsvidhya.com/wp-
```

### Modified Categories:

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

<content/uploads/2016/02/14.-fat-content1.png>)

Enroll Now

Now it makes more sense. But hang on, in step 4 we saw there were some non-consumables as well and a fat-content should not be specified for them. So we can also create a separate category for such kind of observations.

```
#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
```

[\[content/uploads/2016/02/15.-fat-content-2.png\\)\]\(#\)](https://www.analyticsvidhya.com/wp-</a></p></div><div data-bbox=)

## Step 6: Numerical and One-Hot Coding of Categorical variables

Since scikit-learn accepts only numerical variables, I converted all categories of nominal variables into numeric types. Also, I wanted Outlet\_Identifier as a variable as well. So I created a new variable 'Outlet' same as Outlet\_Identifier and coded that. Outlet\_Identifier should remain as it is, because it will be required in the submission file.

Lets start with 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 – 'Low Fat', 'Regular' and '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.

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

Lets look at the datatypes of columns now:

```
data.dtypes
```

Enroll Now

Item_Identifier	object
Item_MRP	float64
Item_Outlet_Sales	float64
Item_Visibility	float64
Item_Weight	float64
Outlet_Identifier	object
source	object
Item_Visibility_MeanRatio	float64
Outlet_Years	int64
Item_Average_OutletType_Sales	float64
Item_Fat_Content_0	float64
Item_Fat_Content_1	float64
Item_Fat_Content_2	float64
Outlet_Location_Type_0	float64
Outlet_Location_Type_1	float64
Outlet_Location_Type_2	float64
Outlet_Size_0	float64
Outlet_Size_1	float64
Outlet_Size_2	float64
Outlet_Type_0	float64
Outlet_Type_1	float64
Outlet_Type_2	float64
Outlet_Type_3	float64
Item_Type_Combined_0	float64
Item_Type_Combined_1	float64
Item_Type_Combined_2	float64
Outlet_0	float64
Outlet_1	float64
Outlet_2	float64
Outlet_3	float64
Outlet_4	float64
Outlet_5	float64
Outlet_6	float64
Outlet_7	float64
Outlet_8	float64
Outlet_9	float64
<b>dtype: object</b>	

<content/uploads/2016/02/16.-final-dtypes.png>)

Here we can see that all variables are now float and each category has a new variable. Lets look at the 3 columns formed from Item\_Fat\_Content.

```
data[['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

([<content/uploads/2016/02/17.-final-fat-content-compare.png>\)](https://www.analyticsvidhya.com/wp-</a></p>
</div>
<div data-bbox=)

You can notice that each row will have only one of the columns as 1 corresponding to the category in the original variable.

## Step 7: 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:

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

With this we come to the end of this section. If you want all the codes for exploration and feature engineering in an iPython notebook format, you can download the same from my [GitHub repository](#) ([https://github.com/aarshayj/Analytics\\_Vidhya/tree/master/Articles/Getting\\_Started\\_with\\_BigMart\\_Sales\(AV\\_Datahacks\)](https://github.com/aarshayj/Analytics_Vidhya/tree/master/Articles/Getting_Started_with_BigMart_Sales(AV_Datahacks))).

Now that we have the data ready, its time to start making predictive models. I will take you through 6 models including linear regression, decision tree and random forest which can get you into Top 20 ranks in this competition (I mean ranks as of today because after reading this article, I'm sure many new leaders will emerge).

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:

```
#Mean based:  
mean_sales = train['Item_Outlet_Sales'].mean()  
  
#Define a dataframe with IDs for submission:  
base1 = test[['Item_Identifier','Outlet_Identifier']]  
base1['Item_Outlet_Sales'] = mean_sales  
  
#Export submission file  
base1.to_csv("alg0.csv",index=False)
```

## Public Leaderboard Score: 1773

Seems too naive for you? If you look at the public LB now, you'll find 4 players below this number. So making baseline models helps in setting a benchmark. If your predictive algorithm is below this, there is something going seriously wrong and you should check your data.

If you participated in AV datahacks or other short duration hackathons, you'll notice first submissions coming in within 5-10 mins of data being available. These are nothing but baseline solutions and no rocket science.

Taking overall mean is just the simplest way. You can also try:

1. Average sales by product
2. Average sales by product in the particular outlet type

These should give better baseline solutions.

Since I'll be making many models, instead of repeating the codes again and again, I would like to define a **generic function** which takes the algorithm and data as input and **makes the model, performs cross-validation and generates submission**. If you don't like functions, you can choose the longer way as well. But I have a tendency of using functions a lot (actually I over-use sometimes :D). So here is the function:



Enroll Now

```

#define target and ID columns:
target = 'Item_Outlet_Sales'

IDcol = ['Item_Identifier','Outlet_Identifier']

from sklearn import cross_validation, metrics

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_validation.cross_val_score(alg, dtrain[predictors], dtrain[target], cv=20, scoring='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),np.min(cv_score),np.max(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)

```

I've put in self-explanatory comments. Please feel free to discuss in comments if you face difficulties in understanding the code. If you're new to the concept of cross-validation, read more about it [here](#) (<https://www.analyticsvidhya.com/blog/2015/11/improve-model-performance-cross-validation-in-python-r/>).

## Linear Regression Model

Lets make our first linear-regression model. Read more on Linear Regression [here](#) (<https://www.analyticsvidhya.com/blog/2015/10/regression-python-beginners/>).



```

from sklearn.linear_model import LinearRegression, Ridge, Lasso
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')

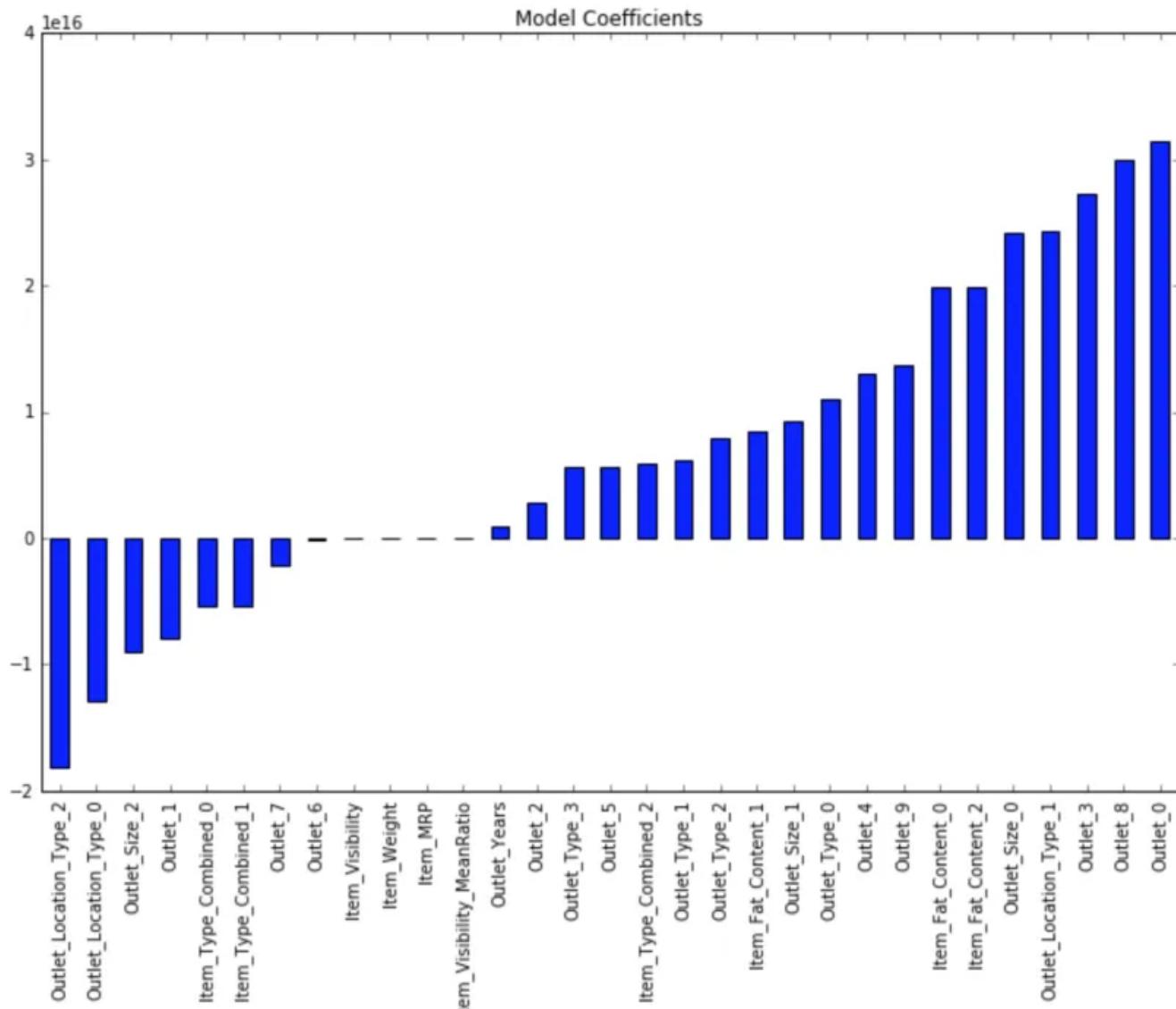
```

### Model Report

RMSE : 1128

CV Score : Mean - 1129 | Std - 44.16 | Min - 1074 | Max - 1218

<matplotlib.axes.\_subplots.AxesSubplot at 0x10b4f3650>



(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/1.-lin-reg.png>)

Public LB Score: 1202

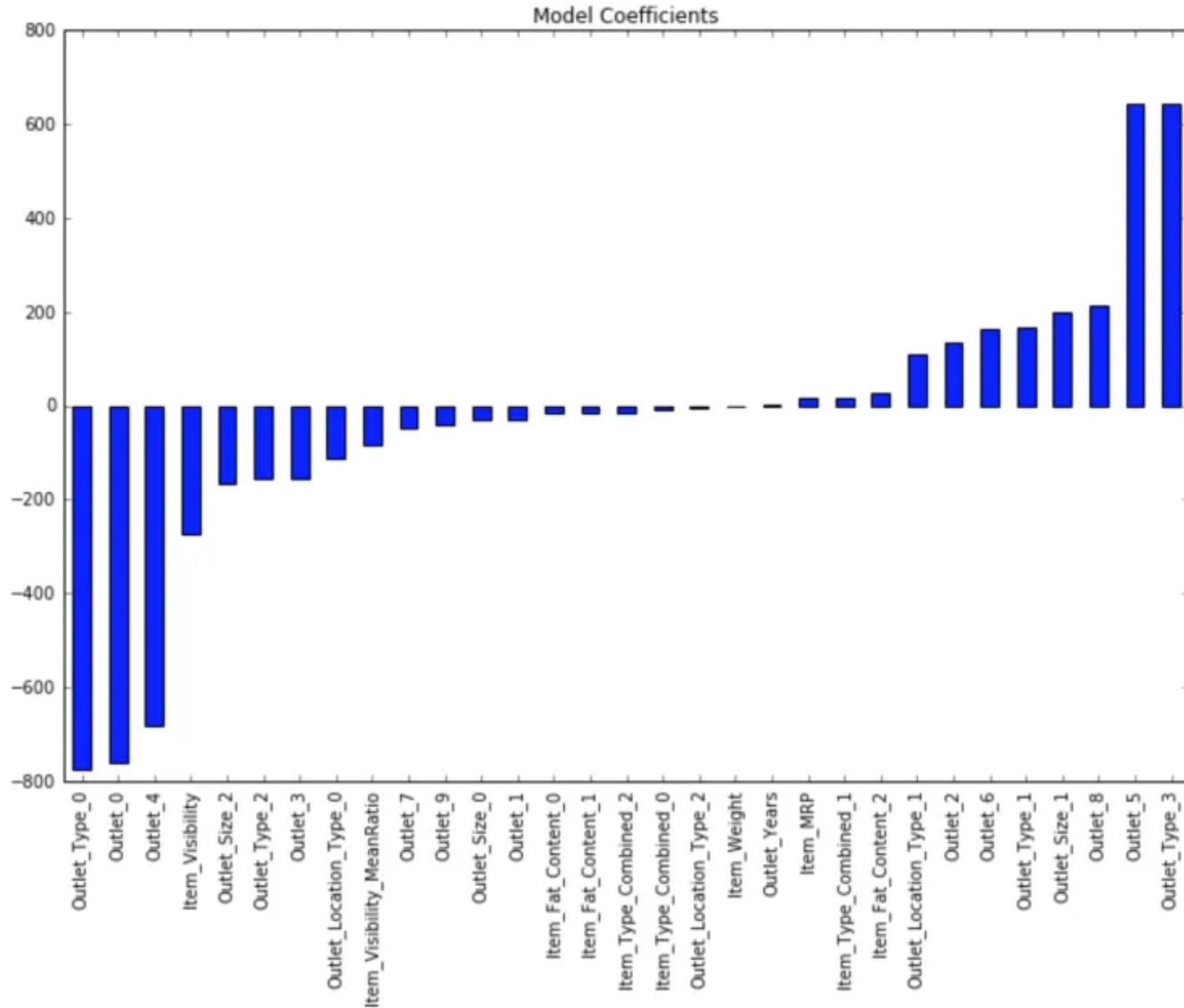
Enroll Now

We can see this is better than baseline model. But if you notice the coefficients, they are very large in magnitude which signifies overfitting. To cater to this, lets use a ridge regression model. You should read [this article](https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/) (<https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/>) if you wish to learn more about Ridge & Lasso regression techniques.

## Ridge Regression Model:

```
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')
```

Model Report  
RMSE : 1129  
CV Score : Mean - 1130 | Std - 44.6 | Min - 1076 | Max - 1217  
<matplotlib.axes.\_subplots.AxesSubplot at 0x10cf61450>



Though the regression coefficient look better now, the score is about the same. You can tune the parameters of the model for slightly better results but I don't think there will be a significant improvement. Even the cross-validation score is same so we can't expect way better performance.

## Decision Tree Model

Lets try out a decision tree model and see if we get something better.

```
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')
```

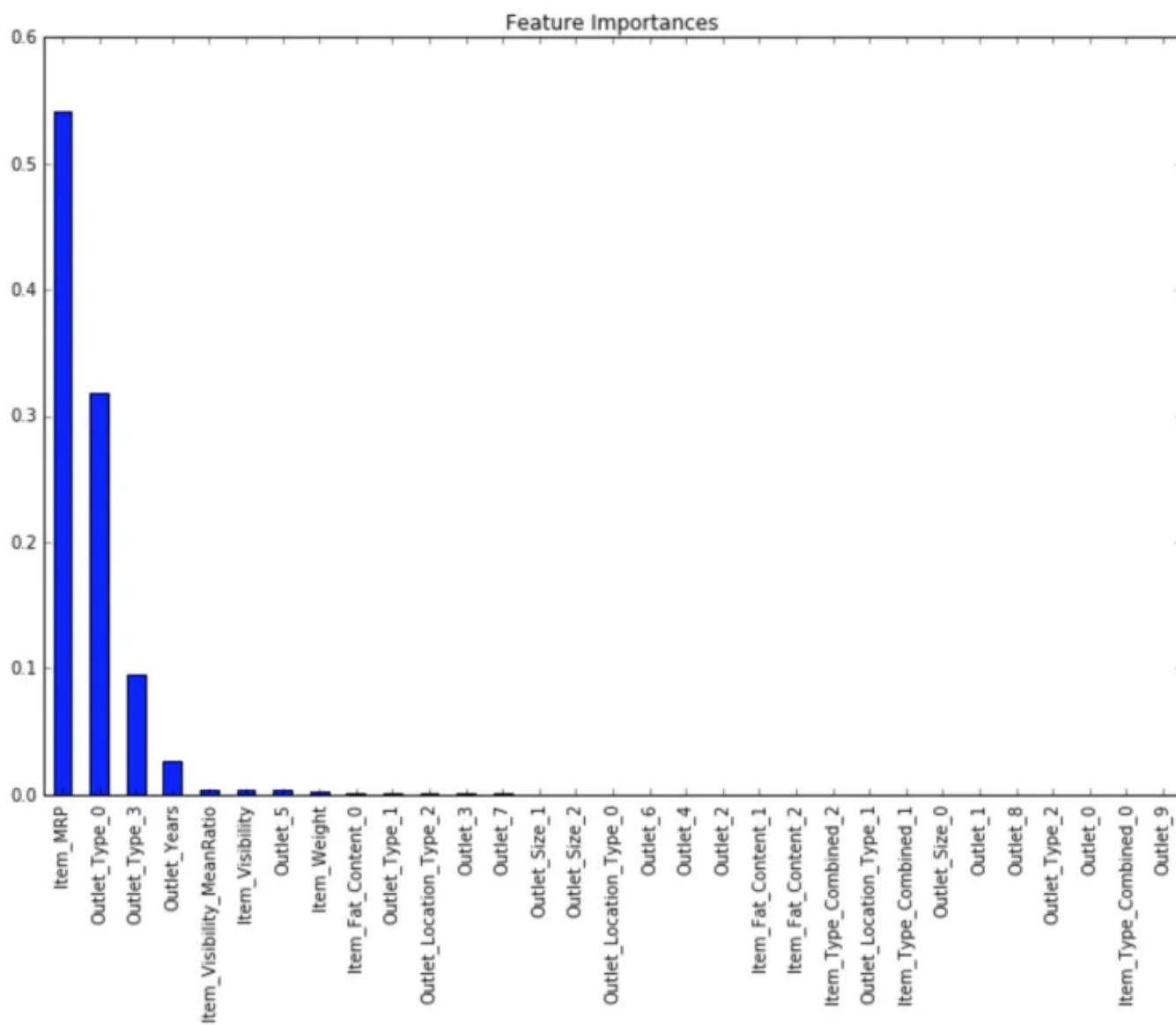


## Model Report

RMSE : 1058

CV Score : Mean - 1091 | Std - 45.42 | Min - 1003 | Max - 1186

<matplotlib.axes.\_subplots.AxesSubplot at 0x10d687250>



(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/3.-dt-1.png>)

Public LB Score: 1162

Here you can see that the RMSE is 1058 and the mean CV error is 1091. This tells us that the model is slightly overfitting. Lets try making a decision tree with just top 4 variables, a max\_depth of 8 and min\_samples\_leaf as 150.

```
predictors = ['Item_MRP', 'Outlet_Type_0', 'Outlet_5', 'Outlet_Years']

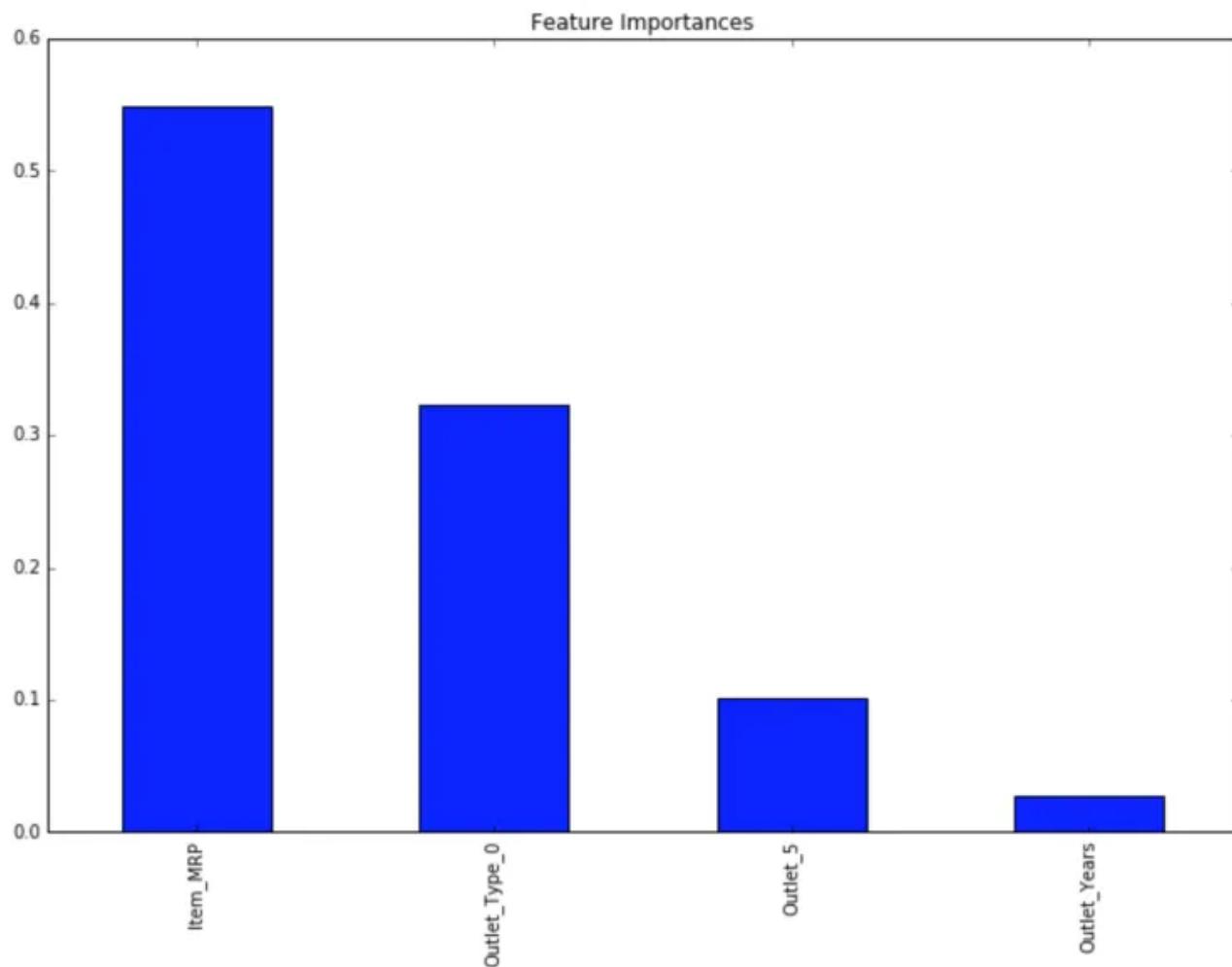
alg4 = DecisionTreeRegressor(max_depth=8, min_samples_leaf=150)

modelfit(alg4, train, test, predictors, target, IDcol, 'alg4.csv')

coef4 = pd.Series(alg4.feature_importances_, predictors).sort_values(ascending=False)

coef4.plot(kind='bar', title='Feature Importances')
```

```
Model Report
RMSE : 1071
CV Score : Mean - 1096 | Std - 43.3 | Min - 1027 | Max - 1172
<matplotlib.axes._subplots.AxesSubplot at 0x10da94b10>
```



(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/4.-dt2.png>).

**Public LB Score: 1157**

You can fine tune the model further using other parameters. I'll leave this to you.

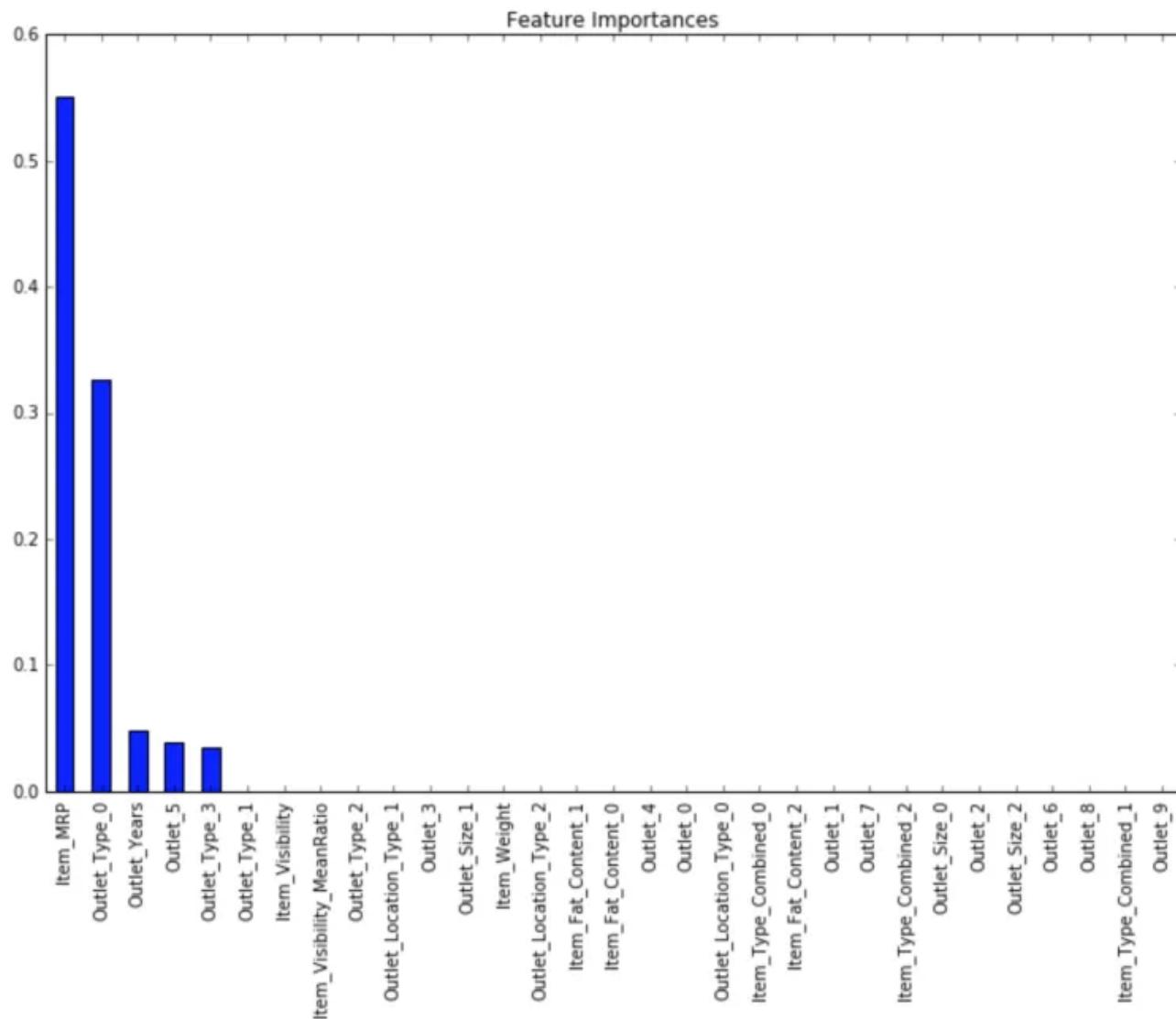
## Random Forest Model

Lets try a random forest model as well and see if we get some improvements. Read more about random forest [here](#) (<https://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/>).

```
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')
```

Model Report  
RMSE : 1073  
CV Score : Mean - 1084 | Std - 43.79 | Min - 1020 | Max - 1160

<matplotlib.axes.\_subplots.AxesSubplot at 0x10df14d10>



(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/5.-rf1-1.png>).

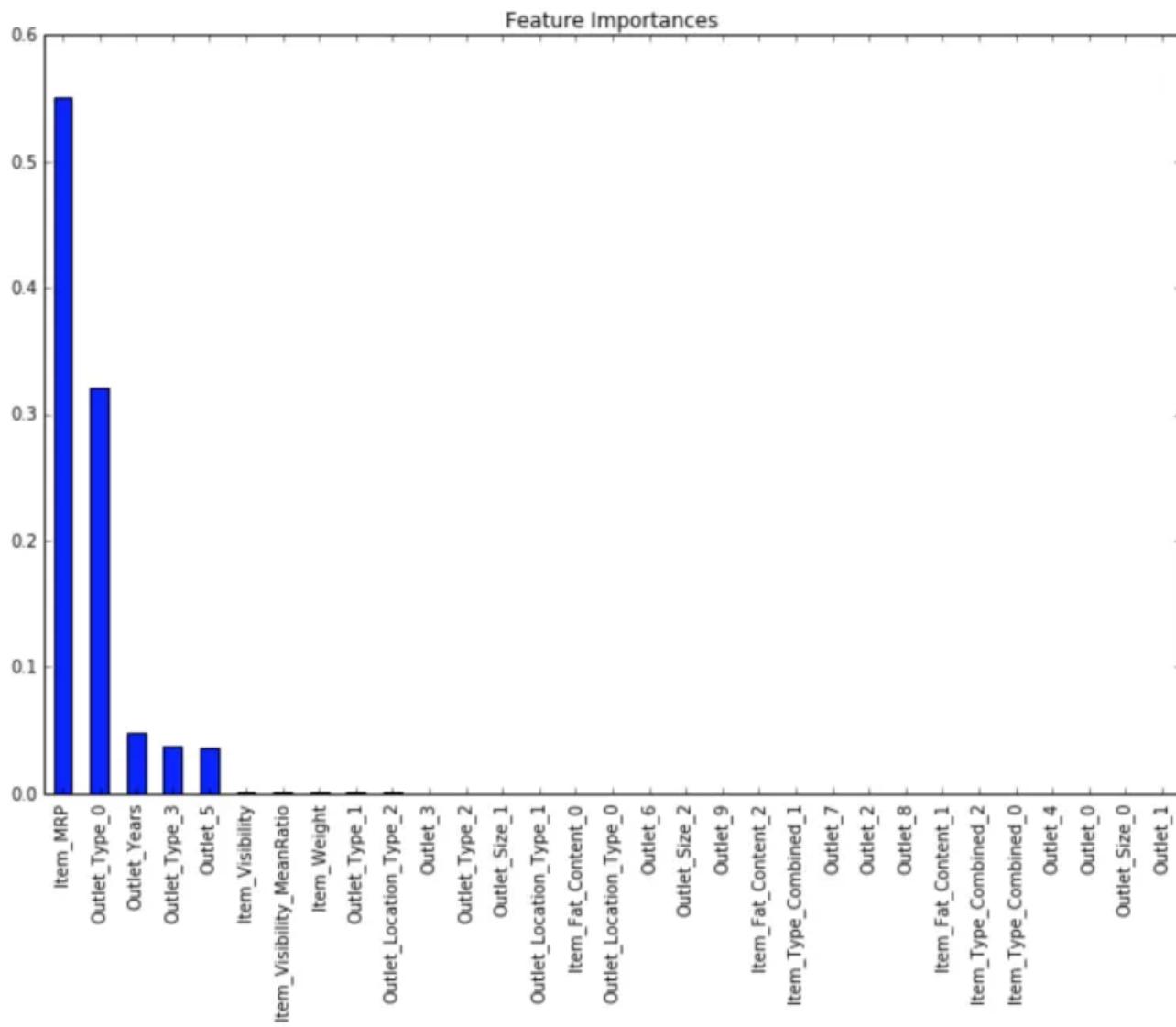
Public LB Score: 1154

You might feel this is a very small improvement but as our model gets better, achieving even minute improvements becomes exponentially difficult. Lets try another random forest with max\_depth of 6 and 400 trees. Increasing the number of trees makes the model robust but is computationally expensive.

```
predictors = [x for x in train.columns if x not in [target]+IDcol]
alg6 = RandomForestRegressor(n_estimators=400,max_depth=6, min_samples_leaf=100,n_jobs=4)
modelfit(alg6, train, test, predictors, target, IDcol, 'alg6.csv')
coef6 = pd.Series(alg6.feature_importances_, predictors).sort_values(ascending=False)
coef6.plot(kind='bar', title='Feature Importances')
```

Model Report  
RMSE : 1068  
CV Score : Mean - 1083 | Std - 43.77 | Min - 1020 | Max - 1162

<matplotlib.axes.\_subplots.AxesSubplot at 0x10df12450>



(<https://www.analyticsvidhya.com/wp-content/uploads/2016/02/6.-rf2-1.png>)

LB Score: 1152

Again this is an incremental change but will help you get a jump of 5-10 ranks on leaderboard. You should try to tune the parameters further to get higher accuracy. But this is good enough to get you into the top 20 on the LB as of now. I tried a basic GBM with little tuning and got into the top 10. I leave it to you to refine with score with better algorithms like [GBM](#) (<https://www.analyticsvidhya.com/blog/2015/09/complete-guide-boosting-methods/>) and [XGBoost](#) (<https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/>) and try [ensemble](#) (<https://www.analyticsvidhya.com/blog/2015/10/trick-right-model-ensemble/>). techniques.

With this we come to the end of this section. If you want all the codes for model building in an iPython notebook format, you can download the same from my [GitHub repository](#) ([https://github.com/aarshayj/Analytics\\_Vidhya/tree/master/Articles/Getting\\_Started\\_with\\_BigMart\\_Sales\(AV\\_Datahacks\)](https://github.com/aarshayj/Analytics_Vidhya/tree/master/Articles/Getting_Started_with_BigMart_Sales(AV_Datahacks))).

End Notes

Enroll Now