Problem Description

- Sales Prediction is utilized to anticipate offers of various items sold at different outlets in various urban areas of a Big Mart Company. Predicting the correct interest for an item is difficult as merchants have constrained time, inventory space and cash for the dealers.

- The interest of an item relies upon numerous components like value, popularity, time, outlet type, outlet area and other features.Successfully predicting the sales for different products can help business in various ways such as inventory planning, know which products to focus on, etc

- Some of the challenging factors like lack of historical data, consumer-oriented markets face uncertain demands, and short life cycle of prediction methods result in inaccurate calculations of various segments of business.

- Our aim is to develop a predictive model for predicting the sales of each product at a particular outlet. This model is built from **Big Mart's** 2013 sales data for 1559 products across 10 stores. We will use one of the datahacks organized by Analytics Vidhya, to test our models

- Our vision is to then expand from sales analysis to other analytical tasks like Productions Planning, Inventory Optimization, etc and create an all in one business analytics platform

Technology Landscape Assessment

Patents :

Following inventions inspired us to undertake this study

1. Predictive & Profile Learning Sales Automation Analytics system
   Reference (2020)
2. Apparatus for providing sales forecasting information based on network (2016)
3. Product sales management control device and product sales management control program (2018)
4. A kind of logistics supply chain needing forecasting method based on big data (2018)
5. Demand forecasting using weighted mixed machine learning models (2017)

Published Literature :

Latest research on use of ML in Sales Analytics

1. Sales Prediction using Linear and KNN Regression (2020)Reference
2. Sales-forecasting of Retail Stores using Machine Learning Techniques (2018)
3. Machine-Learning Models for Sales Time Series Forecasting (2019)
4. Sales forecasting by combining clustering and machine-learning techniques for computer retailing (2016)
5. Integration of Machine Learning Insights into Organizational Learning: A Case of B2B Sales Forecasting (2016)

Potential Customer Segment:

- Small Businesses (As they may not afford a whole Analytics team)
- E-Commerce startups
- Big Retail stores

Libraries/Toolkits to be used (all open) :

- NumPy
- Pandas
- Matplotlib
- Jupyter Notebook
- Seaborn
- Scipy

Market Size: The business analytics market was valued at 67.92 billion USD at 2020 and is expected to reach 103.65 billion by 2026 at a CAGR of 7.3 %

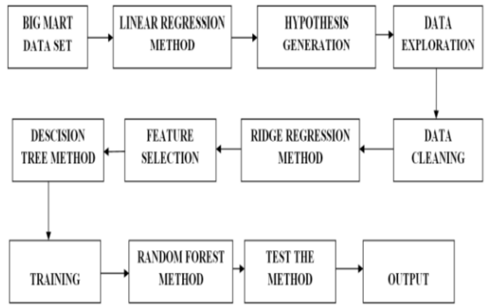
Project Plan

- We will use **Big Mart's** 2013 sales data for 1559 products across 10 stores.
- Our aim is to build an interface for the Big companies like Big mart to help them manipulate their sales strategies for their outlets in different cities and countries by building a predictive model and predict the sales of each product at a particular outlet.
- We plan to also provide a report feature in our interface which will give an overall understanding of the sales and importance of various attributes which will help Big mart understand properties of products and outlets.
- We will provide an interface to predict the sales which would need the required attributes as an input which will help Big mart play around with their strategies.

Project Workflow

- Technology Landscape assessment
- Marketing Aspect assessment
- Product building code Backend :
    - Data Analysis and Feature Engineering
    - Create and Clean Dataset to build the final model
- Model Building: We plan to implement regression as well as Decision tree based models on train dataset to predict the sales.
- Frontend Development : Create a interface for final product where companies can upload data and receive an overall analysis report as output

Conceptual Design



We plan to use various regression models for predicting sales. The input training and test data will be uploaded by the user on the website in the form of 2 csv files (the files need to follow certain rules mentioned in the user manual). On receiving input files, a model will be trained in the backend using python libraries sklearn, numpy, scikit and pandas. The output (predicted sales and insights) will then be displayed on the webpage. The web application will be built using the Django framework

Attributes we plan to focus for analysis:

Store Level Hypotheses	***Product Level Hypotheses***
► City type /Location	Brand
► Population density	Packaging
► Store capacity	Utility
► Competitors	Display Area
► Marketing	Visibility in Store

Models tried: Baseline Model, Linear Regression, Ridge Regression, Decision Tree Random Forest model

Random forest seemed to be the most successful model in the datahack. So, we have decided to use this in the web application. More details on model selection are discussed in the next section

Note: Screenshots of UI and output visualization, unit testing report, model training and testing report are all discussed in the next section

1.1 Data Dictionary

For the Datahack, we have Train (8523) and Test (5681) data set. Train data set has both input & output variable(s). We need to predict the sales for the Test data set. We can conclude that this is a **supervised machine**

learning regression problem.

We will explore the problem in the following stages:

- **Hypothesis Generation** – understanding the problem better by brainstorming possible factors that can impact the outcome
- **Exploratory Data Analysis** – looking at categorical and continuous feature summaries and thus, making inferences about the data.
- **Data Cleaning** – imputing missing values in the data
- **Feature Engineering** – modifying existing variables and creating new ones for analysis
- **Model Building** – making predictive models on the data using regression techniques

1.3 Hypothesis Generation

1.3.1 Store Level Hypotheses

- **City type**: Stores located in urban or Tier 1 cities should have higher sales because of the higher income levels of people there.
- **Population Density**: Stores located in densely populated areas should have higher sales because of more demand.
- **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
- **Competitors**: Stores having similar establishments nearby should have less sales because of more competition.
- **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.
- **Location**: Stores located within popular marketplaces should have higher sales because of better access to customers.

1.3.2 Product Level Hypotheses

- **Brand**: Branded products should have higher sales because of higher trust in the customer.
- **Packaging**: Products with good packaging can attract customers and sell more.
- **Utility**: Daily use products should have a higher tendency to sell as compared to the specific use products.
- **Display Area**: Products which are given bigger shelves in the store are likely to catch attention first and sell more.
- **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.

We can think about other parameters like advertising, promotional offers, etc. which might impact the sales. For now, we will proceed towards exploratory data analysis.

1.4 Exploratory Data Analysis

We will look at the data and try to identify the information which we hypothesized vs the available data. As given on the competition website, the `train` dataset has the following attributes:

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

On the other hand, the `test` dataset has all the above mentioned attributes except `Item_Outlet_Sales`. As a part of the competition, we have to predict the `Item_Outlet_Sales` for all the items in the `test` dataset and submit a csv file with three columns, as given below:

- `Item_Identifier` – Unique product ID
- `Outlet_Identifier` – Unique store ID
- `Item_Outlet_Sales` – Sales of the product in the particular store. This is the outcome variable to be predicted.

Let's start by loading the required libraries and data. The `train` and `test` data sets are available on the competition website.

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

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
train = pd.read_csv("train_v9rqX0R.csv")
train.sample(10)
```

[2]:
```
     Item_Identifier  Item_Weight Item_Fat_Content  Item_Visibility  \
4161           DRI23       18.850          Low Fat         0.137973
344            FDJ22          NaN          Low Fat         0.092464
4985           FDT14       10.695          Regular         0.127621
2636           FDT33        7.810          Regular         0.034044
5815           FDT57       15.200          Low Fat         0.019031
3250           FDX38       10.500          Regular         0.048167
6057           DRA59          NaN          Regular         0.127308
```

```
5794            FDI09        20.750          Regular          0.000000
1151            FDK43         9.800          Low Fat          0.026993
3434            FDT39         6.260          Regular          0.009924

        Item_Type  Item_MRP Outlet_Identifier  Outlet_Establishment_Year  \
4161  Hard Drinks  158.4578            OUT017                       2007
344   Snack Foods  190.9504            OUT019                       1985
4985        Dairy  119.2440            OUT013                       1987
2636  Snack Foods  168.7158            OUT049                       1999
5815  Snack Foods  235.5248            OUT035                       2004
3250        Dairy   48.8376            OUT013                       1987
6057  Soft Drinks  186.6924            OUT027                       1985
5794      Seafood  239.9880            OUT045                       2002
1151         Meat  127.3020            OUT017                       2007
3434         Meat  152.8366            OUT017                       2007

     Outlet_Size Outlet_Location_Type        Outlet_Type  Item_Outlet_Sales
4161         NaN               Tier 2  Supermarket Type1          1444.1202
344        Small               Tier 1      Grocery Store           383.5008
4985        High               Tier 3  Supermarket Type1          3475.4760
2636      Medium               Tier 1  Supermarket Type1          2673.8528
5815       Small               Tier 2  Supermarket Type1          4740.4960
3250        High               Tier 3  Supermarket Type1           671.1264
6057      Medium               Tier 3  Supermarket Type3          7033.5112
5794         NaN               Tier 2  Supermarket Type1          2636.5680
1151         NaN               Tier 2  Supermarket Type1          2277.0360
3434         NaN               Tier 2  Supermarket Type1          3778.4150
```

[3]:
```
print("train data has {} rows and {} columns".format(train.shape[0], train.
 ↪shape[1]))
```

```
train data has 8523 rows and 12 columns
```

[4]:
```
test = pd.read_csv("test_AbJTz2l.csv")
test.head()
```

[4]:
```
  Item_Identifier  Item_Weight Item_Fat_Content  Item_Visibility    Item_Type  \
0           FDW58       20.750          Low Fat         0.007565  Snack Foods
1           FDW14        8.300              reg         0.038428        Dairy
2           NCN55       14.600          Low Fat         0.099575       Others
3           FDQ58        7.315          Low Fat         0.015388  Snack Foods
4           FDY38          NaN          Regular         0.118599        Dairy

   Item_MRP Outlet_Identifier  Outlet_Establishment_Year Outlet_Size  \
0  107.8622            OUT049                       1999      Medium
1   87.3198            OUT017                       2007         NaN
2  241.7538            OUT010                       1998         NaN
```

```
3  155.0340           OUT017                      2007          NaN
4  234.2300           OUT027                      1985       Medium

  Outlet_Location_Type        Outlet_Type
0               Tier 1  Supermarket Type1
1               Tier 2  Supermarket Type1
2               Tier 3      Grocery Store
3               Tier 2  Supermarket Type1
4               Tier 3  Supermarket Type3
```

[5]:
```python
print("test data has {} rows and {} columns".format(test.shape[0], test.
 ↪shape[1]))
```

```
test data has 5681 rows and 11 columns
```

Its generally a good idea to combine both `train` and `test` datasets into one, perform feature engineering and then divide them later again. We will combine `train` and `test` into a dataframe `data` with a `source` column specifying where each observation belongs.

[6]:
```python
train['source'] = 'train'
test['source'] = 'test'
data = pd.concat([train, test], axis = 0)
print(train.shape, test.shape, data.shape)
```

```
(8523, 13) (5681, 12) (14204, 13)
```

Thus we can see that `data` has same number of columns (as that in `train` dataset) but rows equivalent to both `test` and `train` taken together.

[7]:
```python
data.sample(5)
```

[7]:
```
     Item_Fat_Content Item_Identifier  Item_MRP  Item_Outlet_Sales  \
770           Regular           FDH53   80.9592                NaN
2926              reg           FDV28   35.3558                NaN
5118          Regular           FDC29  112.7176          1832.2816
377           Regular           FDF45   57.7904          1464.7600
5557          Low Fat           FDC20   56.2272           559.2720

                  Item_Type  Item_Visibility  Item_Weight  \
770            Frozen Foods         0.019230        20.50
2926           Frozen Foods         0.160052        16.10
5118           Frozen Foods         0.024088          NaN
377   Fruits and Vegetables         0.012195        18.20
5557  Fruits and Vegetables         0.024069        10.65

      Outlet_Establishment_Year Outlet_Identifier Outlet_Location_Type  \
770                        1999            OUT049               Tier 1
2926                       2002            OUT045               Tier 2
5118                       1985            OUT027               Tier 3
```

```
377                         1987          OUT013              Tier 3
5557                        2009          OUT018              Tier 3

     Outlet_Size        Outlet_Type source
770       Medium  Supermarket Type1   test
2926         NaN  Supermarket Type1   test
5118      Medium  Supermarket Type3  train
377         High  Supermarket Type1  train
5557      Medium  Supermarket Type2  train
```

One of the key challenges in any dataset is missing values. We will begin by checking which columns contain missing values.

[8]:
```
print(data.isna().sum())
```

```
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
dtype: int64
```

As we know that the `Item_Outlet_Sales` is the target variable and its missing values are the ones which are present in the `test` dataset. So we will leave this column as it is. However, we need to impute the missing values in `Item_Weight` and `Outlet_Size`.

Next, we will have a look at some basic statistics for numerical variables.

[9]:
```
data.describe()
```

[9]:
```
            Item_MRP  Item_Outlet_Sales  Item_Visibility   Item_Weight  \
count  14204.000000        8523.000000     14204.000000  11765.000000
mean     141.004977        2181.288914         0.065953     12.792854
std       62.086938        1706.499616         0.051459      4.652502
min       31.290000          33.290000         0.000000      4.555000
25%       94.012000         834.247400         0.027036      8.710000
50%      142.247000        1794.331000         0.054021     12.600000
75%      185.855600        3101.296400         0.094037     16.750000
max      266.888400       13086.964800         0.328391     21.350000

       Outlet_Establishment_Year
```

```
count                14204.000000
mean                  1997.830681
std                      8.371664
min                   1985.000000
25%                   1987.000000
50%                   1999.000000
75%                   2004.000000
max                   2009.000000
```

From the above mentioned statistics, we can observe that

- `Item_Visibility` has a minimum value of zero. This makes no practical sense because when a product is being sold in a store, the visibility cannot be 0.
- `Outlet_Establishment_Year` varies 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.
- The lower `count` of `Item_Weight` and `Item_Outlet_Sales` substantiates the fact that there are missing values in these two columns.

Next, we will have a look at the number of unique values in each of the columns.

[10]:
```
data.apply(lambda x : len(x.unique()))
```

[10]:
```
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
dtype: int64
```

The above mentioned result tells us that 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. We will explore it further using the frequency of different categories in each variable. Also, we will not include `Item_Identifier`,`Outlet_Identifier`,`source` for obvious reasons.

[11]:
```
# 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']]

for col in categorical_columns:
    print('Frequency of categories for {}'.format(col))
    print(data[col].value_counts())
    print("="*50)
```

```
Frequency of categories for 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 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 Outlet_Location_Type
Tier 3    5583
Tier 2    4641
Tier 1    3980
Name: Outlet_Location_Type, dtype: int64
==================================================
Frequency of categories for Outlet_Size
Medium    4655
Small     3980
High      1553
Name: Outlet_Size, dtype: int64
==================================================
```

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

The above mentioned output provides us with the following observations:

- `Item_Fat_Content`: Some of `Low Fat` values are incorrectly coded as `low fat` and `LF`. Also, some of `Regular` values are mentioned as `regular`.
- `Item_Type`: Not all categories have substantial numbers. Maybe, combining them can give better results.

1.5 Data Cleaning

Here, we will deal with the imputation of missing values. In the previous section, we noticed that there are two variables with missing values – `Item_Weight` and `Outlet_Size`. We will impute the `Item_Weight` by the average weight of the particular item.

[12]:
```
# Determine the average weight per item
item_avg_weight = data.groupby('Item_Identifier')['Item_Weight'].mean()

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

# Impute data
data.loc[miss_bool,'Item_Weight'] = data.loc[miss_bool,'Item_Identifier'].
 ↪apply(lambda x : item_avg_weight[x])
```

[13]:
```
print(data['Item_Weight'].isna().sum())
```

```
0
```

Now, we will impute the `Outlet_Size` with the mode of the `Outlet_Size` for the particular type of outlet.

[14]:
```
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]))

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

# Impute data
```

```
data.loc[miss_bool,'Outlet_Size'] = data.loc[miss_bool,'Outlet_Type'].
 ↪apply(lambda x : outlet_size_mode[x])
```

[15]:
```
print(data.isna().sum())
```

```
Item_Fat_Content               0
Item_Identifier                0
Item_MRP                       0
Item_Outlet_Sales           5681
Item_Type                      0
Item_Visibility                0
Item_Weight                    0
Outlet_Establishment_Year      0
Outlet_Identifier              0
Outlet_Location_Type           0
Outlet_Size                    0
Outlet_Type                    0
source                         0
dtype: int64
```

The above mentioned output confirms that there are no missing values now. Remember, `Item_Outlet_Sales` is the target variable and its missing values are the ones which are present in the `test` dataset.

1.6 Feature Engineering

In the section **Exploratory Data Analysis**, we noticed that the minimum value of `Item_Visibility` is 0, which makes no practical sense. We will treat the zero entries as missing values and hence we need to impute these with mean visibility of that particular item.

[16]:
```
print(len(data[data['Item_Visibility'] == 0]))
```

```
879
```

[17]:
```
# Determine average visibility of a product
visibility_avg = data.groupby('Item_Identifier')['Item_Visibility'].mean()

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

data.loc[miss_bool,'Item_Visibility'] = data.loc[miss_bool,'Item_Identifier'].
 ↪apply(lambda x : visibility_avg[x])
```

[18]:
```
print(len(data[data['Item_Visibility'] == 0]))
```

```
0
```

In the section **Hypothesis Generation**, we inferred that products with higher visibility are likely to sell more. But along with comparing products on absolute terms, we need to 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 will add a new column `Item_Visibility_MeanRatio` by using the `visibility_avg` variable defined above.

[19]:
```
# Determine another variable with means ratio
data['Item_Visibility_MeanRatio'] = data.apply(lambda x : x['Item_Visibility']/
↪visibility_avg[x['Item_Identifier']], axis=1)
data['Item_Visibility_MeanRatio'].describe()
```

[19]:
```
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
```

In the section **Exploratory Data Analysis** we saw that `Item_Type` variable has 16 categories which might prove to be very useful in analysis. So it might be a good idea to combine them. If we look at the entries of `Item_Identifier`, i.e. the unique ID of each item, it starts with either `FD`, `DR` or `NC`. If we see the categories, these look like being Food, Drinks and Non-Consumables. So we can use used `Item_Identifier` variable to create a new column.

[20]:
```
data['Item_Identifier'].sample(10)
```

[20]:
```
3212    FDI44
3503    FDL45
7334    NCB55
3645    FDU32
6777    FDV60
2987    NCK42
3097    FDI36
2496    FDQ26
1116    FDL39
3594    DRC36
Name: Item_Identifier, dtype: object
```

[21]:
```
# 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()
```

[21]:
```
Food              10201
Non-Consumable     2686
Drinks             1317
Name: Item_Type_Combined, dtype: int64
```

[22]:
```
data.sample(8)
```

[22]:
```
     Item_Fat_Content Item_Identifier  Item_MRP  Item_Outlet_Sales  \
7943          Low Fat           NCE31   33.6216           934.7832
1785          Regular           FDQ28  155.0656                NaN
1228          Low Fat           FDU52  154.4630          2503.4080
5647          Regular           DRY23   43.8112                NaN
1352          Low Fat           NCZ06  253.8698                NaN
7221          Low Fat           FDP33  256.3672           255.6672
6032          Low Fat           NCP06  151.4366          1511.3660
953           Regular           FDZ23  185.4240           745.6960

         Item_Type  Item_Visibility  Item_Weight  Outlet_Establishment_Year  \
7943     Household         0.183948        7.670                       1985
1785  Frozen Foods         0.105800       14.000                       1985
1228  Frozen Foods         0.064031        7.560                       2009
5647   Soft Drinks         0.109318        9.395                       2002
1352     Household         0.094353       19.600                       2002
7221   Snack Foods         0.156304       18.700                       1985
6032     Household         0.039246       20.700                       1997
953   Baking Goods         0.112986       17.750                       1998

     Outlet_Identifier Outlet_Location_Type Outlet_Size        Outlet_Type  \
7943            OUT027               Tier 3      Medium  Supermarket Type3
1785            OUT019               Tier 1       Small      Grocery Store
1228            OUT018               Tier 3      Medium  Supermarket Type2
5647            OUT045               Tier 2       Small  Supermarket Type1
1352            OUT045               Tier 2       Small  Supermarket Type1
7221            OUT019               Tier 1       Small      Grocery Store
6032            OUT046               Tier 1       Small  Supermarket Type1
953             OUT010               Tier 3       Small      Grocery Store

     source  Item_Visibility_MeanRatio Item_Type_Combined
7943  train                   0.870493     Non-Consumable
1785   test                   1.679003               Food
1228  train                   1.028941               Food
5647   test                   0.924160             Drinks
```

```
1352   test                        0.924160     Non-Consumable
7221   train                       1.679003               Food
6032   train                       0.929633     Non-Consumable
953    train                       1.464117               Food
```

Now, we will add another new column depicting the years of operation of a store.

```
[23]: # Big Mart has collected 2013 sales data for 1559 products across 10 stores in␣
      ↪different cities
      # Years of operation since 2013
      data['Outlet_Years'] = 2013 - data['Outlet_Establishment_Year']
      data['Outlet_Years'].describe()
```

```
[23]: 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
```

Remember, in `Item_Fat_Content`, we noticed that some of `Low Fat` values are incorrectly coded as `low fat` and `LF`. Also, some of `Regular` values are mentioned as `regular`. Here, we will fix these.

```
[24]: # Change categories of low fat
      print('Original Categories of Item_Fat_Content:')
      print('-'*40)
      print(data['Item_Fat_Content'].value_counts())
```

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

```
[25]: data['Item_Fat_Content'] = data['Item_Fat_Content'].replace({'LF':'Low Fat',
                                                                     'reg':'Regular',
                                                                     'low fat':'Low␣
      ↪Fat'})
      print('Modified Categories of Item_Fat_Content:')
      print('-'*40)
      print(data['Item_Fat_Content'].value_counts())
```

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

Earlier, we saw there were some non-consumables as well and a fat-content should not be specified for them. So we will create a separate category for such kind of observations.

[26]:
```
data['Item_Type_Combined'].value_counts()
```

[26]:
```
Food              10201
Non-Consumable     2686
Drinks             1317
Name: Item_Type_Combined, dtype: int64
```

[27]:
```
# 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()
```

[27]:
```
Low Fat       6499
Regular       5019
Non-Edible    2686
Name: Item_Fat_Content, dtype: int64
```

[28]:
```
data.head(5)
```

[28]:
```
  Item_Fat_Content Item_Identifier  Item_MRP  Item_Outlet_Sales  \
0          Low Fat           FDA15  249.8092          3735.1380
1          Regular           DRC01   48.2692           443.4228
2          Low Fat           FDN15  141.6180          2097.2700
3          Regular           FDX07  182.0950           732.3800
4       Non-Edible           NCD19   53.8614           994.7052

               Item_Type  Item_Visibility  Item_Weight  \
0                  Dairy         0.016047         9.30
1            Soft Drinks         0.019278         5.92
2                   Meat         0.016760        17.50
3  Fruits and Vegetables         0.017834        19.20
4              Household         0.009780         8.93

   Outlet_Establishment_Year Outlet_Identifier Outlet_Location_Type  \
0                       1999            OUT049               Tier 1
1                       2009            OUT018               Tier 3
2                       1999            OUT049               Tier 1
3                       1998            OUT010               Tier 3
4                       1987            OUT013               Tier 3
```

```
  Outlet_Size        Outlet_Type source  Item_Visibility_MeanRatio  \
0      Medium  Supermarket Type1  train                   0.931078
1      Medium  Supermarket Type2  train                   0.933420
2      Medium  Supermarket Type1  train                   0.960069
3       Small      Grocery Store  train                   1.000000
4        High  Supermarket Type1  train                   1.000000

  Item_Type_Combined  Outlet_Years
0               Food            14
1             Drinks             4
2               Food            14
3               Food            15
4     Non-Consumable            26
```

1.6.1 Encoding of Categorical Variables

Since `scikit-learn` accepts only numerical variables, we will converted all categorical variables into numeric ones. Also, we will create a new variable `Outlet` same as `Outlet_Identifier`. `Outlet_Identifier` should remain as it is, because it will be required in the submission file.

Now, we will begin with the encoding of all categorical variables as numeric using `LabelEncoder`.

[29]:
```
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'
le = LabelEncoder()
for i in var_mod:
    data[i] = le.fit_transform(data[i])
```

[30]:
```
data.head(5)
```

[30]:
```
   Item_Fat_Content Item_Identifier  Item_MRP  Item_Outlet_Sales  \
0                 0           FDA15  249.8092          3735.1380
1                 2           DRC01   48.2692           443.4228
2                 0           FDN15  141.6180          2097.2700
3                 2           FDX07  182.0950           732.3800
4                 1           NCD19   53.8614           994.7052

              Item_Type  Item_Visibility  Item_Weight  \
0                 Dairy         0.016047         9.30
1           Soft Drinks         0.019278         5.92
2                  Meat         0.016760        17.50
```

```
3  Fruits and Vegetables         0.017834        19.20
4              Household         0.009780         8.93

   Outlet_Establishment_Year Outlet_Identifier  Outlet_Location_Type  \
0                       1999            OUT049                     0
1                       2009            OUT018                     2
2                       1999            OUT049                     0
3                       1998            OUT010                     2
4                       1987            OUT013                     2

   Outlet_Size  Outlet_Type source  Item_Visibility_MeanRatio  \
0            1            1  train                   0.931078
1            1            2  train                   0.933420
2            1            1  train                   0.960069
3            2            0  train                   1.000000
4            0            1  train                   1.000000

   Item_Type_Combined  Outlet_Years  Outlet
0                   1            14       9
1                   0             4       3
2                   1            14       9
3                   1            15       0
4                   2            26       1
```

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 with binary values.

[31]:
```python
# One Hot Coding
data = pd.get_dummies(data,␣
↪columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Outlet_Type',
                              'Item_Type_Combined','Outlet'])
```

We will have a look at the data types of columns after encoding.

[32]:
```python
data.dtypes
```

[32]:
```
Item_Identifier               object
Item_MRP                     float64
Item_Outlet_Sales            float64
Item_Type                     object
Item_Visibility              float64
Item_Weight                  float64
Outlet_Establishment_Year      int64
```

```
Outlet_Identifier              object
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
```

To visualize the effect of one-hot encoding, we will have a look at the three columns formed from `Item_Fat_Content`.

[33]:
```
data[['Item_Fat_Content_0','Item_Fat_Content_1','Item_Fat_Content_2']].head(10)
```

[33]:

	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

Final step is to convert data back into `train` and test datasets. It will be a good idea to export both of these as modified data sets so that they can be re-used for multiple sessions.

[34]:
```
# 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 for further use
# train.to_csv("train_modified.csv",index=False)
# test.to_csv("test_modified.csv",index=False)
```

1.7 Model Building

We will 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, we will predict the sales as the overall average sales.

[35]:
```
# Mean based baseline model
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("baseline_model.csv",index=False)
```

The above mentioned baseline model resulted into a leaderboard score of 1773.82513777906.

Now, we will implement other models. For this, we would define a generic function which takes the algorithm and data as input and makes the model, performs cross-validation and generates the submission file.

[36]:
```
# Define target and ID columns
target = 'Item_Outlet_Sales'
IDcol = ['Item_Identifier','Outlet_Identifier']

from sklearn import metrics
```

```
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', n_jobs=1)
    cv_score = np.sqrt(np.abs(cv_score))

    print("Model report:")
    print("-"*40)
    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)
```

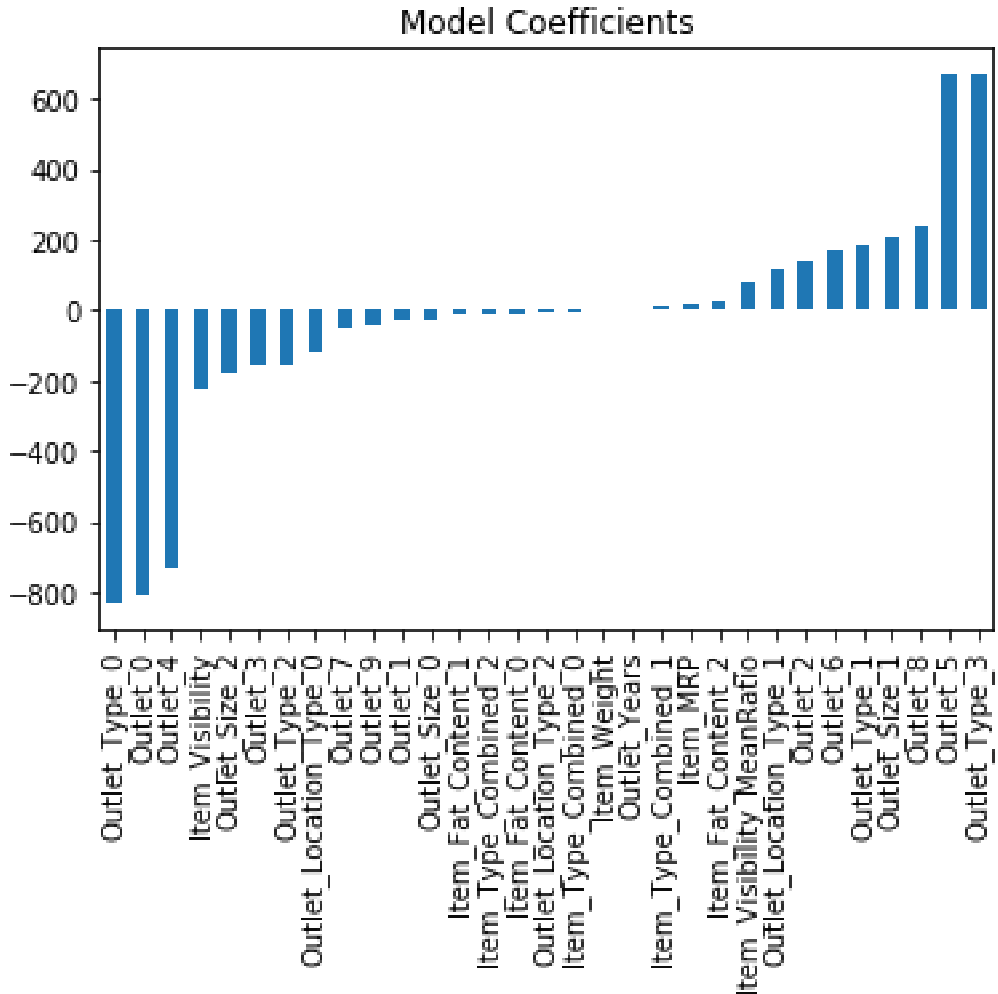
1.7.1 Linear Regression Model

[37]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
predictors = [x for x in train.columns if x not in [target]+IDcol]

# print predictors
lin_reg = LinearRegression(normalize=True)
modelfit(lin_reg, train, test, predictors, target, IDcol, 'lin_reg.csv')
coef1 = pd.Series(lin_reg.coef_, predictors).sort_values()
coef1.plot(kind='bar', title='Model Coefficients')
```

```
Model report:
----------------------------------------
RMSE : 1127
CV Score : Mean - 1129 | Std - 43.61 | Min - 1075 | Max - 1213
```

[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5606415e50>



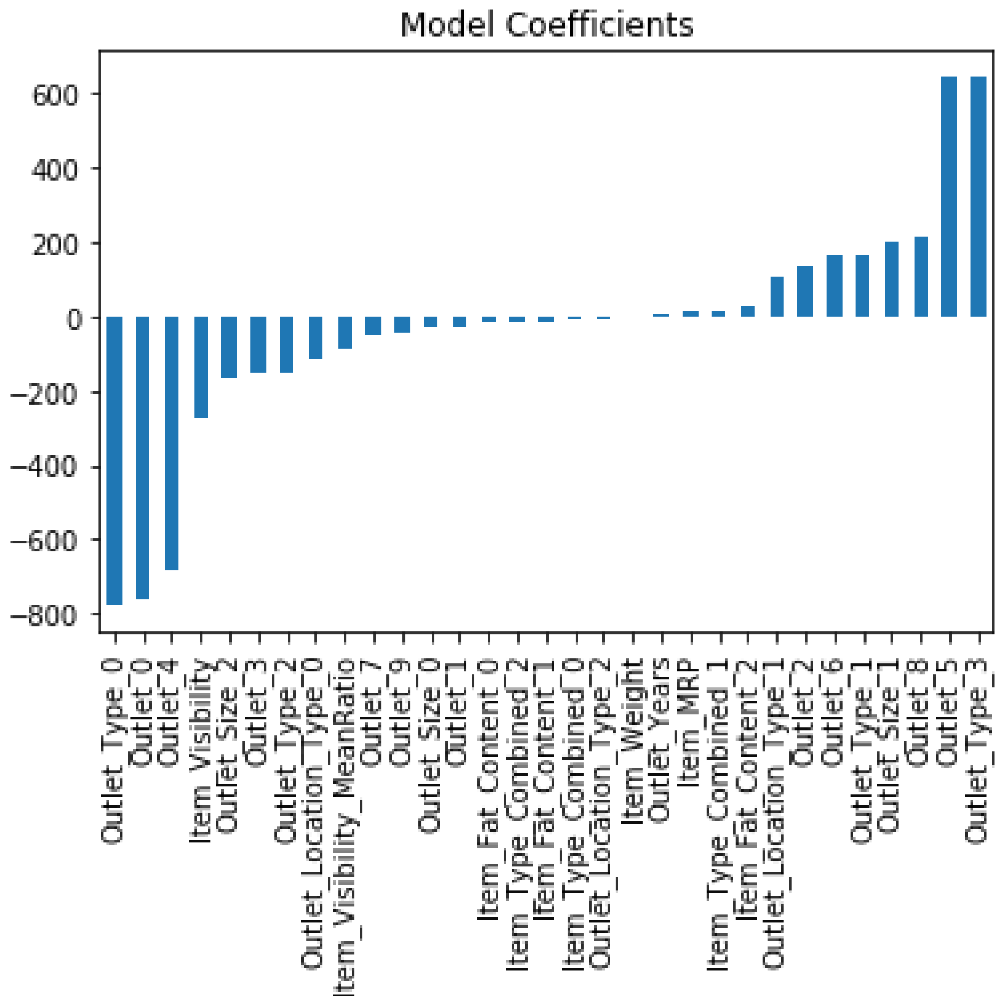
The above mentioned model predicted some of the sales as negative values. Since the submission on Analytics Vidhya accepted only positive values, we could not submit this model. Though there were ways to get rid of the negative values, we preferred to submit the tree-based models, which yielded a decent score.

1.7.2 Ridge Regression Model

[38]:
```
predictors = [x for x in train.columns if x not in [target]+IDcol]
ridge_reg = Ridge(alpha=0.05,normalize=True)
modelfit(ridge_reg, train, test, predictors, target, IDcol, 'ridge_reg.csv')
coef2 = pd.Series(ridge_reg.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
```

[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7f56062a0d50>



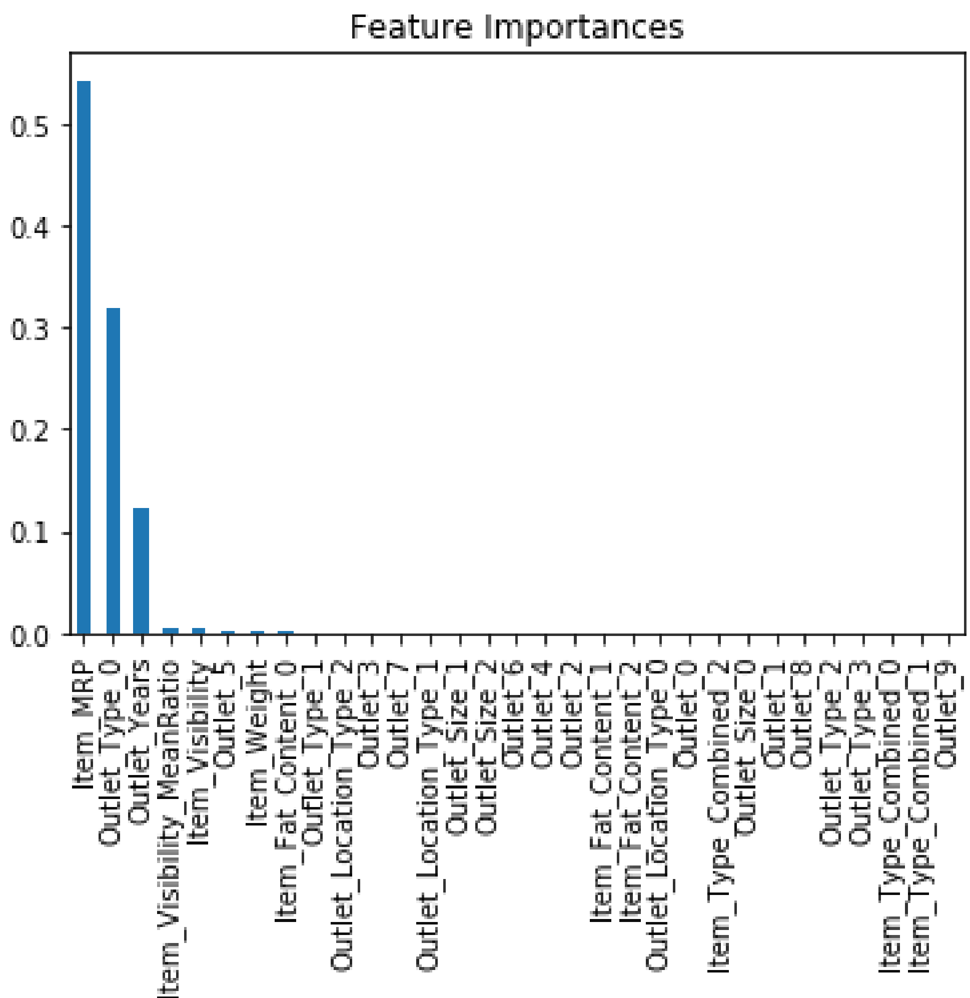
1.7.3 Decision Tree Model

[39]:
```
from sklearn.tree import DecisionTreeRegressor
predictors = [x for x in train.columns if x not in [target]+IDcol]
dec_tree = DecisionTreeRegressor(max_depth=15, min_samples_leaf=100)
modelfit(dec_tree, train, test, predictors, target, IDcol, 'dec_tree.csv')
coef3 = pd.Series(dec_tree.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
```

[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f56060ca250>



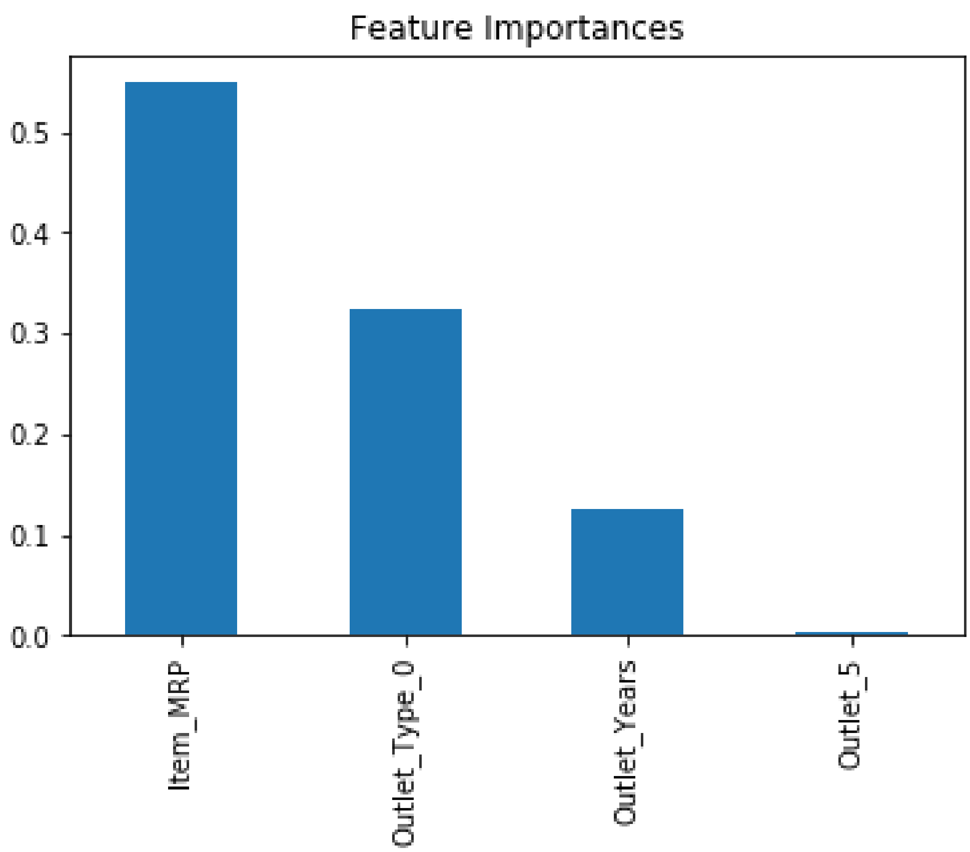
The above mentioned decision tree model resulted into a leaderboard score of 1162.49724969602. Here you can see that the RMSE is 1058 and the mean CV error is 1091. It tells us that the model is slightly overfitting. So, we will try making a decision tree with just top 4 variables, a `max_depth` of 8 and `min_samples_leaf` as 150.

[40]:
```
predictors = ['Item_MRP','Outlet_Type_0','Outlet_5','Outlet_Years']
dec_tree_2 = DecisionTreeRegressor(max_depth=8, min_samples_leaf=150)
```

```
modelfit(dec_tree_2, train, test, predictors, target, IDcol, 'dec_tree_2.csv')
coef4 = pd.Series(dec_tree_2.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
```

[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7f56040afed0>



The above mentioned decision tree model resulted into a leaderboard score of 1156.89167845091.

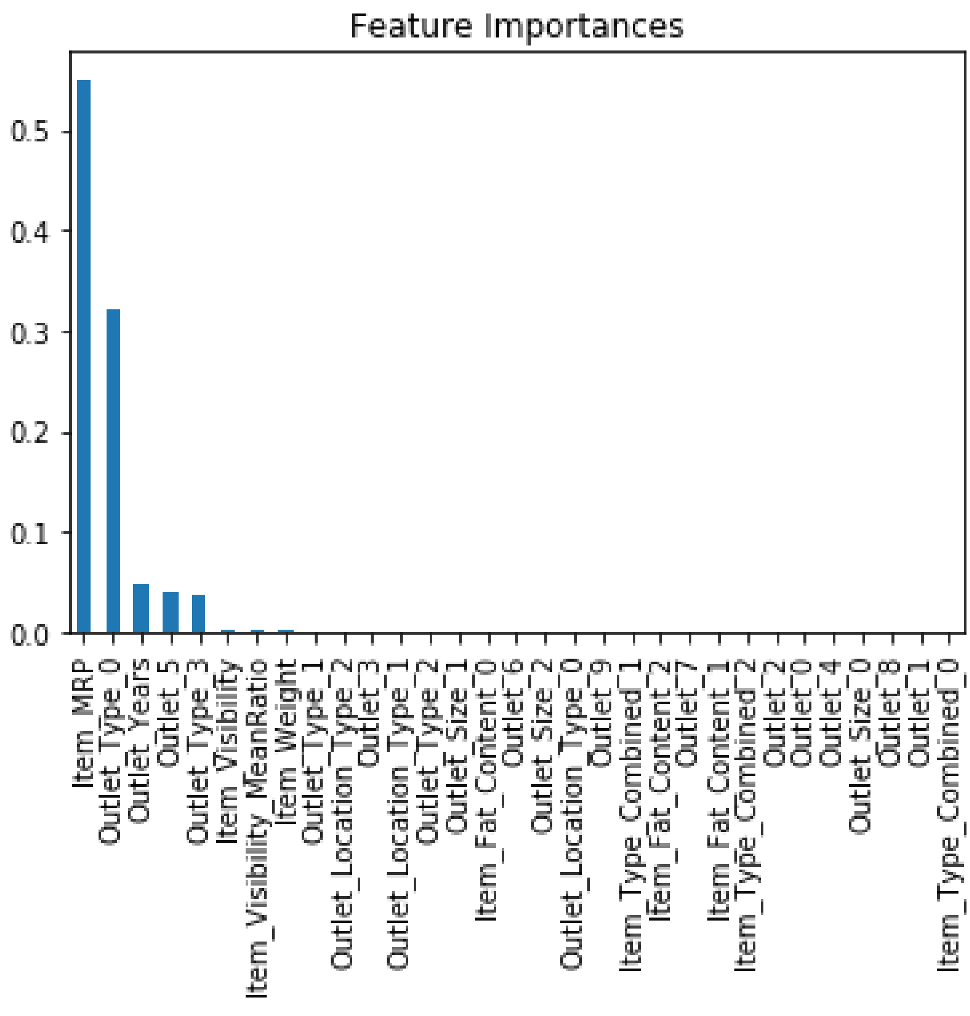
1.7.4 Random Forest Model

[41]:
```
from sklearn.ensemble import RandomForestRegressor
predictors = [x for x in train.columns if x not in [target]+IDcol]
```

```
rand_for = RandomForestRegressor(n_estimators=400,max_depth=6,␣
 ↪min_samples_leaf=100,n_jobs=4)
modelfit(rand_for, train, test, predictors, target, IDcol, 'rand_for.csv')
coef6 = pd.Series(rand_for.feature_importances_, predictors).
 ↪sort_values(ascending=False)
coef6.plot(kind='bar', title='Feature Importances')
```

```
Model report:
----------------------------------------
RMSE : 1068
CV Score : Mean - 1083 | Std - 43.88 | Min - 1020 | Max - 1160
```

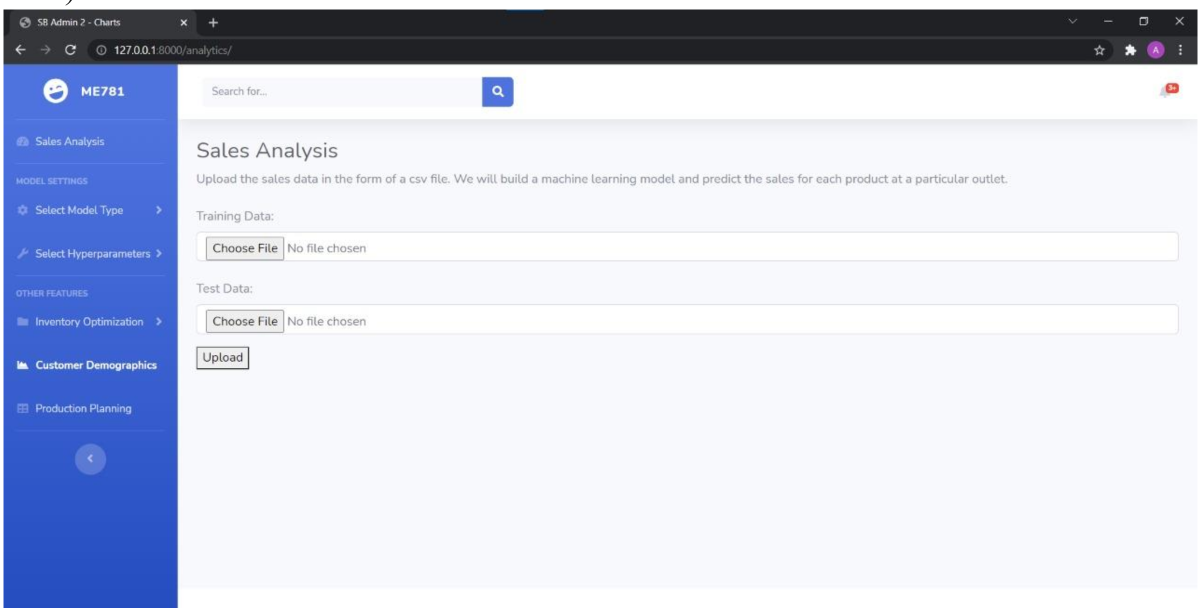
[41]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f55ef6cd610>`



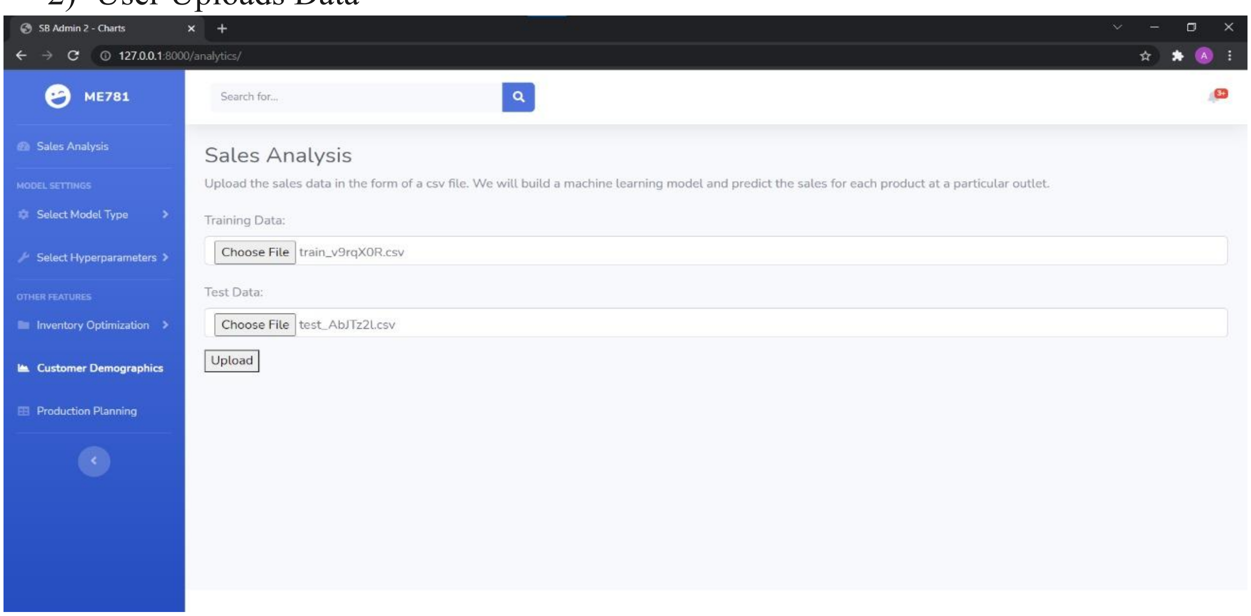
The above mentioned random forest model resulted into a leaderboard score of 1152.96165866681. Next, we also tried TPOT (Tree-based Pipeline Optimization Tool). Though it enhanced the score by reducing the RMSE, it took more than 1.5 hours to train, which is not acceptable for our final product as we need realtime outputs for users. Hence, it was not used

UI and Output Visualization

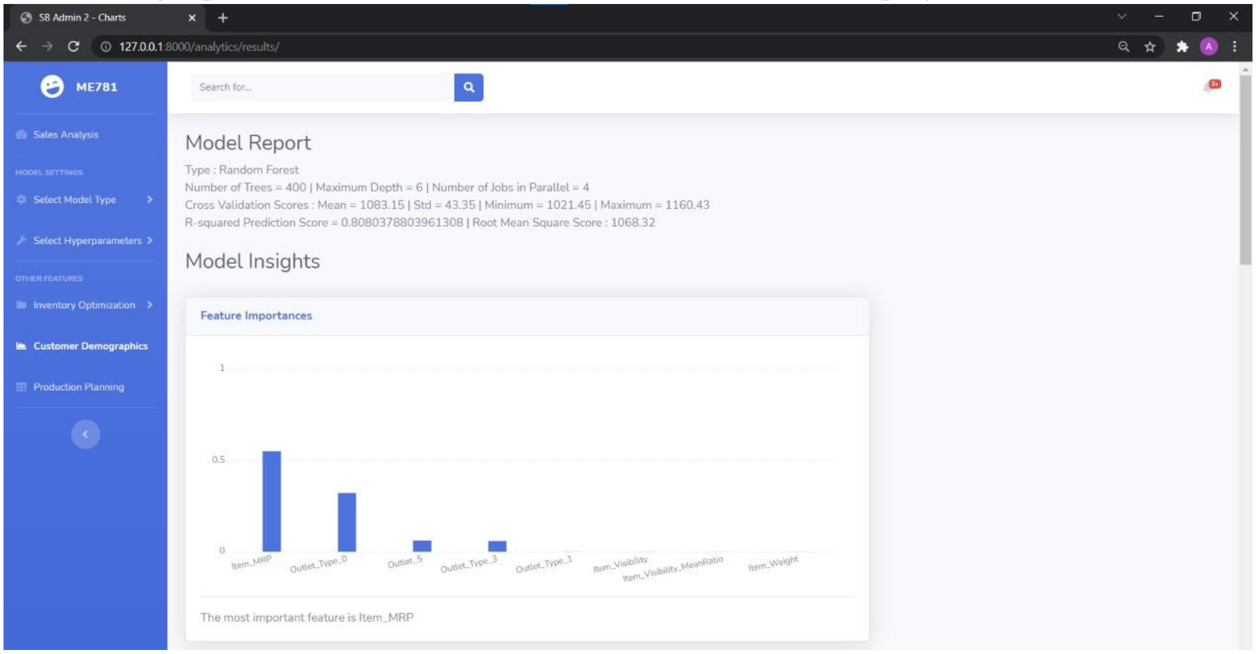
1) Frontend

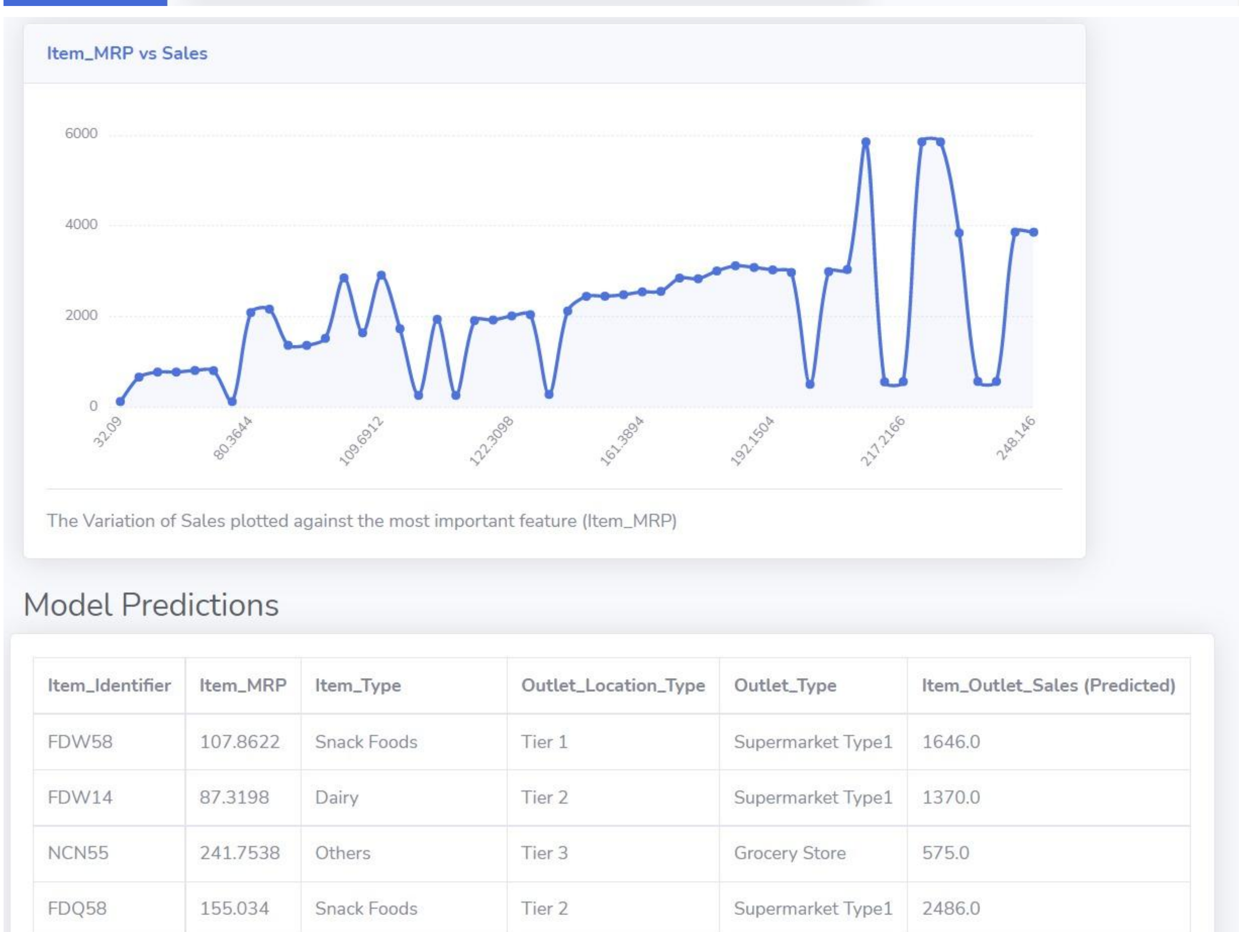


2) User Uploads Data



3) On Clicking Upload, the model is trained and the results are displayed





User Manual

A sample training file is shown for reference :

	A	B	C	D	E	F	G	H	I	J	K	L
1	Item_Iden	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment	Outlet_Size	Outlet_Location_	Outlet_Type	Item_Outlet_Sales
2	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
3	DRC01	5.92	Regular	0.019278216	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
4	FDN15	17.5	Low Fat	0.016760075	Meat	141.618	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.27
5	FDX07	19.2	Regular	0	Fruits and Veg	182.095	OUT010	1998		Tier 3	Grocery Store	732.38
6	NCD19	8.93	Low Fat	0	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052
7	FDP36	10.395	Regular	0	Baking Goods	51.4008	OUT018	2009	Medium	Tier 3	Supermarket Type2	556.6088
8	FDO10	13.65	Regular	0.012741089	Snack Foods	57.6588	OUT013	1987	High	Tier 3	Supermarket Type1	343.5528
9	FDP10		Low Fat	0.127469857	Snack Foods	107.7622	OUT027	1985	Medium	Tier 3	Supermarket Type3	4022.7636
10	FDH17	16.2	Regular	0.016687114	Frozen Foods	96.9726	OUT045	2002		Tier 2	Supermarket Type1	1076.5986
11	FDU28	19.2	Regular	0.09444959	Frozen Foods	187.8214	OUT017	2007		Tier 2	Supermarket Type1	4710.535
12	FDY07	11.8	Low Fat	0	Fruits and Veg	45.5402	OUT049	1999	Medium	Tier 1	Supermarket Type1	1516.0266
13	FDA03	18.5	Regular	0.045463773	Dairy	144.1102	OUT046	1997	Small	Tier 1	Supermarket Type1	2187.153
14	FDX32	15.1	Regular	0.1000135	Fruits and Veg	145.4786	OUT049	1999	Medium	Tier 1	Supermarket Type1	1589.2646
15	FDS46	17.6	Regular	0.047257328	Snack Foods	119.6782	OUT046	1997	Small	Tier 1	Supermarket Type1	2145.2076
16	FDF32	16.35	Low Fat	0.0680243	Fruits and Veg	196.4426	OUT013	1987	High	Tier 3	Supermarket Type1	1977.426
17	FDP49	9	Regular	0.069088961	Breakfast	56.3614	OUT046	1997	Small	Tier 1	Supermarket Type1	1547.3192

Columns containing information on the product should start with Item_ , whereas columns containing information on the outlet (where the product is sold) should start with Outlet_ . The Item_Outlet_Sales column should be named as it is. Missing data is allowed. The uploaded file must be of csv format. Same procedure must be followed for the testing file and no columns should be missing except Item_Outlet_Sales. Sample test file :-

	A	B	C	D	E	F	G	H	I	J	K	L
1	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	
2	FDW58	20.75	Low Fat	0.007564836	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1	
3	FDW14	8.3	reg	0.038427677	Dairy	87.3198	OUT017	2007		Tier 2	Supermarket Type1	
4	NCN55	14.6	Low Fat	0.099574908	Others	241.7538	OUT010	1998		Tier 3	Grocery Store	
5	FDQ58	7.315	Low Fat	0.015388393	Snack Foods	155.034	OUT017	2007		Tier 2	Supermarket Type1	
6	FDY38		Regular	0.118599314	Dairy	234.23	OUT027	1985	Medium	Tier 3	Supermarket Type3	
7	FDH56	9.8	Regular	0.063817206	Fruits and Vegetable	117.1492	OUT046	1997	Small	Tier 1	Supermarket Type1	
8	FDL48	19.35	Regular	0.082601537	Baking Goods	50.1034	OUT018	2009	Medium	Tier 3	Supermarket Type2	
9	FDC48		Low Fat	0.015782495	Baking Goods	81.0592	OUT027	1985	Medium	Tier 3	Supermarket Type3	
10	FDN33	6.305	Regular	0.123365446	Snack Foods	95.7436	OUT045	2002		Tier 2	Supermarket Type1	
11	FDA36	5.985	Low Fat	0.005698435	Baking Goods	186.8924	OUT017	2007		Tier 2	Supermarket Type1	
12	FDT44	16.6	Low Fat	0.103569075	Fruits and Vegetable	118.3466	OUT017	2007		Tier 2	Supermarket Type1	
13	FDQ56	6.59	Low Fat	0.10581147	Fruits and Vegetable	85.3908	OUT045	2002		Tier 2	Supermarket Type1	
14	NCC54		Low Fat	0.171079215	Health and Hygiene	240.4196	OUT019	1985	Small	Tier 1	Grocery Store	