BigMartSales

Problem Statement

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.

Please note that the data may have missing values as some stores might not report all the data due to technical glitches. Hence, it will be required to treat them accordingly.

Data

We have train (8523) and test (5681) data set, train data set has both input and output variable(s). You need to predict the sales for test data set. Variable Description 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 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 Outlet_Location_Type The type of city in which the store is located Outlet_Type Whether the outlet is just a grocery store or some sort of supermarket Item_Outlet_Sales Sales of the product in the particulat store. This is the outcome variable to be predicted.

Evaluation Metric: Your model performance will be evaluated on the basis of your prediction of the sales for the test data (test.csv), which contains similar data-points as train except for the sales to be predicted. Your submission needs to be in the format as shown in "SampleSubmission.csv". We at our end, have the actual sales for the test dataset, against which your predictions will be evaluated. We will use the Root Mean Square Error value to judge your response.

RoadMap

#

Data Summary: Exploration, Visalization, Data Imputation

Preprocessing / Feature Engineering: Deriving the column, Transfomation, FEature selection

Dividing our data interms of trainig and testing

Building the model: linear regression, Ensemble regressor methods

In [1]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:

```
import seaborn as sns
```

In [4]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
#Read CSV

train = pd.read_csv('Train_UWu5bXk.csv')
```

```
train = pd.read_csv('Train_UWu5bXk.csv')
test = pd.read_csv('Test_u94Q5KV.csv')
```

In [5]:

```
train.head()
```

Out[5]:

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

In [2]:

```
#creates a new column 'source' with values as 'train' for all train data and 'test' for all test d
ata
train['source']='train'
test['source']='test'

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

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

data.isnull().sum()
```

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

Out[2]:

```
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
```

In [3]:

```
data.describe()
```

Out[3]:

	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

	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Establishment_Year
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

In [4]:

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

Out[4]:

```
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
```

In [5]:

```
#Filter categorical variables and exclude Item_Identifier etc
categorical_columns=[x for x in data.dtypes.index if data.dtypes[x]== 'object']
categorical_columns=[x for x in categorical_columns if x not in
['Item_Identifier','Outlet_Identifier','source']]

for col in categorical_columns:
    print "\nFrequency of categories for variable %s"%col
    print data[col].value_counts()
```

```
Frequency of categories for variable Item_Fat_Content
Low Fat    8485
Regular    4824
LF          522
reg         195
low fat     178
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
dtype: int64

Frequency of categories for variable Outlet_Location_Type
Tier 3    5583
Tier 2    4641
Tier 1    3980
dtype: int64

Frequency of categories for variable Outlet_Size
```

```
Frequency of categories for variable Outlet_Size
Medium    4655
Small     3980
High      1553
dtype: int64

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

In [6]:

```
item_avg_wt=data.pivot_table(values='Item_Weight',index='Item_Identifier')
print item_avg_wt

miss_bool=data['Item_Weight'].isnull()
# print "hi", sum(miss_bool)
print miss_bool

data.loc[miss_bool,'Item_Weight']=data.loc[miss_bool,'Item_Identifier'].apply(lambda x : item_avg_w
t[x])
print sum(data['Item_Weight'].isnull())
```

```
Item_Identifier
DRA12             11.600
DRA24             19.350
DRA59              8.270
DRB01              7.390
DRB13              6.115
DRB24              8.785
DRB25             12.300
DRB48             16.750
DRC01              5.920
DRC12             17.850
DRC13              8.260
DRC24             17.850
DRC25              5.730
DRC27             13.800
DRC36             13.000
...
NCY30             20.250
NCY41             16.750
NCY42              6.380
NCY53             20.000
NCY54              8.430
NCZ05              8.485
NCZ06             19.600
NCZ17             12.150
NCZ18              7.825
NCZ29             15.000
NCZ30              6.590
NCZ41             19.850
NCZ42             10.500
NCZ53              9.600
NCZ54             14.650
Name: Item_Weight, Length: 1559, dtype: float64
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7       True
8      False
9      False
10     False
11     False
12     False
13     False
14     False
...
14189    False
```

```
14190    False
14191    False
14192     True
14193    False
14194    False
14195    False
14196    False
14197    False
14198    False
14199    False
14200    False
14201    False
14202    False
14203    False
Name: Item_Weight, Length: 14204, dtype: bool
0
```

In [7]:

```
visibility_avg=data.pivot_table(values='Item_Visibility',index='Item_Identifier')
miss_bool=(data['Item_Visibility']==0)

print 'Number of zeros 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 zeros after imputing %d'%sum(data['Item_Visibility']==0)
```

```
Number of zeros initially 879
Number of zeros after imputing 0
```

In [8]:

```
# determine another feature
data['Item_Visibility_MeanRatio']=data.apply(lambda x: x['Item_Visibility']/visibility_avg[x['Item_
Identifier']],axis=1)
print data['Item_Visibility_MeanRatio'].describe()
print visibility_avg[[0,10]]#[data[1]['Item_Identifier']]
```

```
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
Item_Identifier
DRA12              0.034938
DRC13              0.028408
Name: Item_Visibility, dtype: float64
```

In [9]:

```
#capturing initial two characters of string and separating the data
data['Item_Type_Combined']=data['Item_Identifier'].apply(lambda x:x[0:2])
#Renaming 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()
```

Out[9]:

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

In [10]:

```
data['Outlet_Years']=2013-data['Outlet_Establishment_Year']
data['Outlet_Years'].describe()
```

Out[10]:

```
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
```

In [11]:

```
print 'Original Categories :'
print data['Item_Fat_Content'].value_counts()

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

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

 Modified Categories :
```

Out[11]:

```
Low Fat    9185
Regular    5019
dtype: int64
```

In [12]:

```
#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()
```

Out[12]:

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

In [13]:

```
#Outlet Identifier is the store number /name so changed it also to numeric
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data['Outlet']=le.fit_transform(data['Outlet_Identifier'])
var_mod=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Item_Type_Combined','Outlet_Type'
,'Outlet']
for i in var_mod:
    data[i]=le.fit_transform(data[i])

data=pd.get_dummies(data,columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Item_Typ
e_Combined','Outlet_Type','Outlet'])
```

In [14]:

```
data.dtypes#['Item_Fat_Content']
```

```
data.dtypes#['Item_Fat_Content']
```

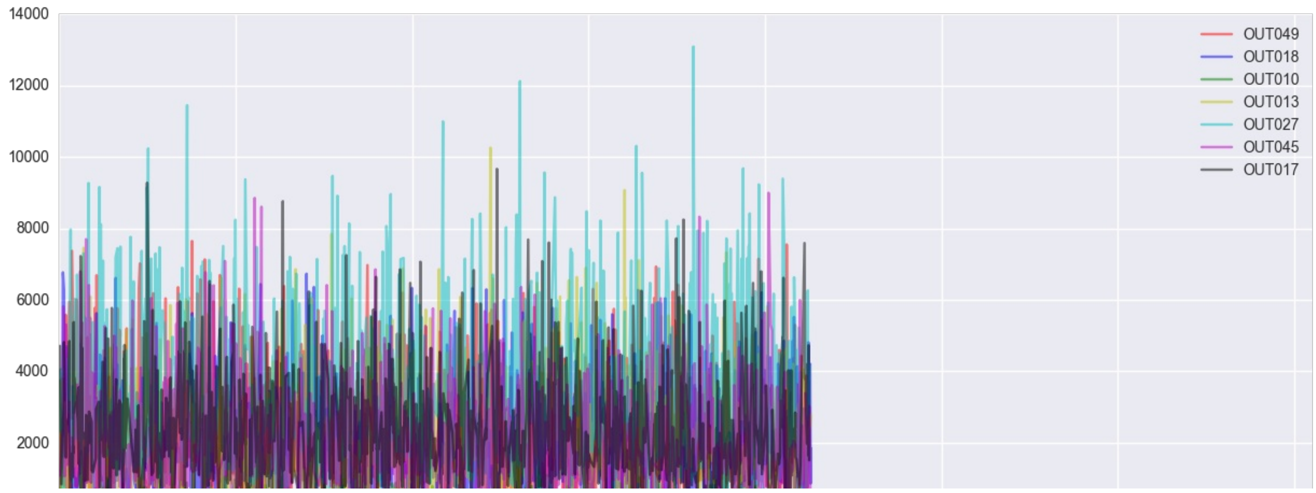
Out[14]:

```
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          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_Size_3               float64
Item_Type_Combined_0        float64
Item_Type_Combined_1        float64
Item_Type_Combined_2        float64
Outlet_Type_0               float64
Outlet_Type_1               float64
Outlet_Type_2               float64
Outlet_Type_3               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
dtype: object
```

Sales across different outlets

In [15]:

```python
colors = ['r', 'b', 'g', 'y', 'c', 'm', 'k']

for i, s in enumerate(data.Outlet_Identifier.unique()[:7]):
    data[data.Outlet_Identifier == s].Item_Outlet_Sales.plot(c=colors[i], figsize=(15, 6), label=s,
alpha=.5)

plt.legend(loc='best');
```



Stores have varying sales

In [16]:

```
data.Outlet_Identifier.unique()
```

Out[16]:

```
array(['OUT049', 'OUT018', 'OUT010', 'OUT013', 'OUT027', 'OUT045',
       'OUT017', 'OUT046', 'OUT035', 'OUT019'], dtype=object)
```

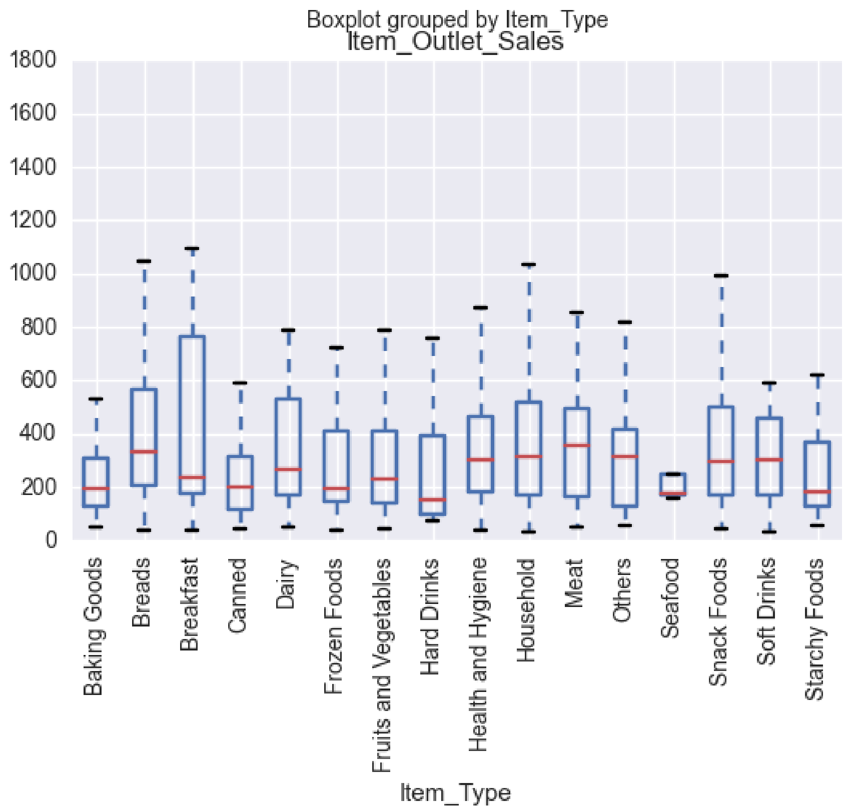
In [17]:

```
data_OUT049 = data[data.Outlet_Identifier == 'OUT049']
data_OUT018 = data[data.Outlet_Identifier == 'OUT018']
data_OUT010 = data[data.Outlet_Identifier == 'OUT010']
data_OUT013 = data[data.Outlet_Identifier == 'OUT013']
data_OUT027 = data[data.Outlet_Identifier == 'OUT027']
data_OUT045 = data[data.Outlet_Identifier == 'OUT045']
data_OUT017 = data[data.Outlet_Identifier == 'OUT017']
data_OUT035 = data[data.Outlet_Identifier == 'OUT035']
data_OUT019 = data[data.Outlet_Identifier == 'OUT019']
data_OUT046 = data[data.Outlet_Identifier == 'OUT046']
```

Correlation between item type and sales for a particular store

In [18]:

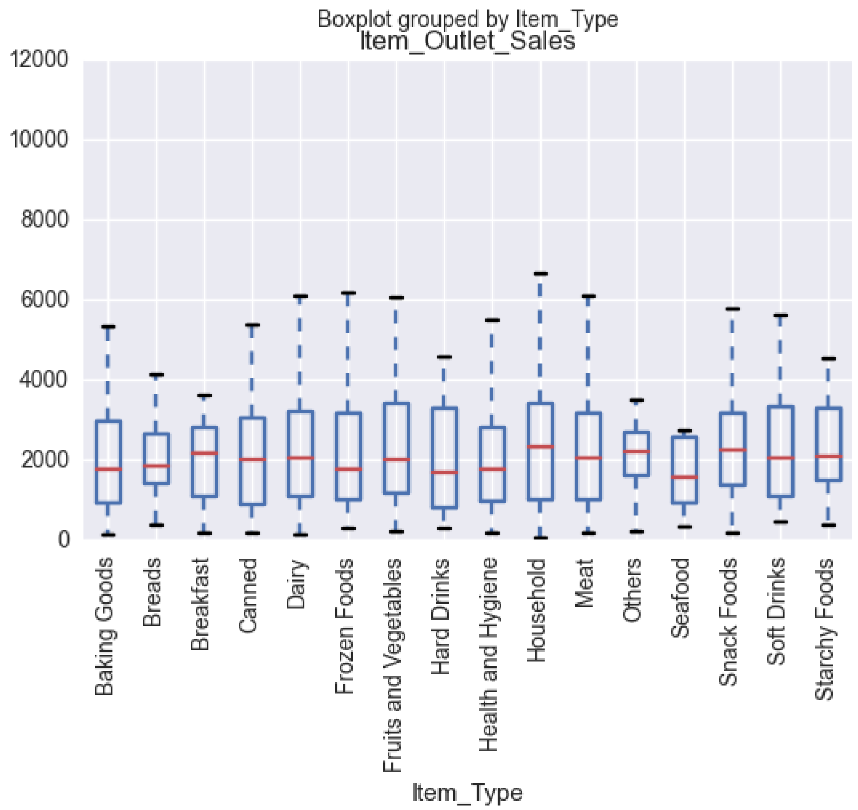
```
data_OUT010.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



In [19]:

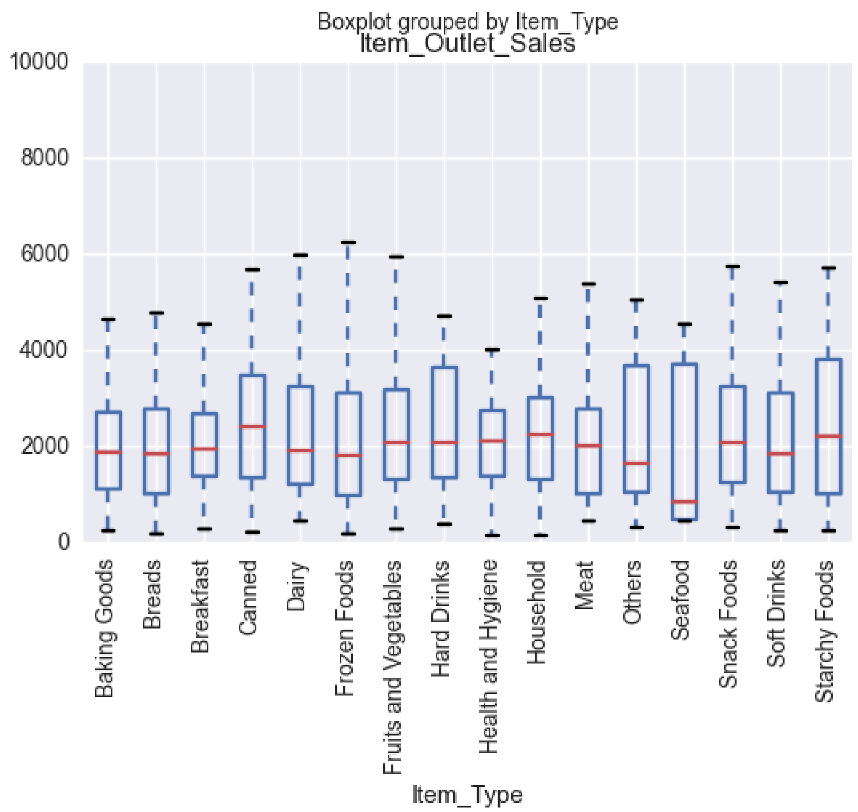
```
data_OUT013.boxplot(column='Item_Outlet_Sales', by='Item_Type')
```

```
plt.xticks(rotation=90);
```



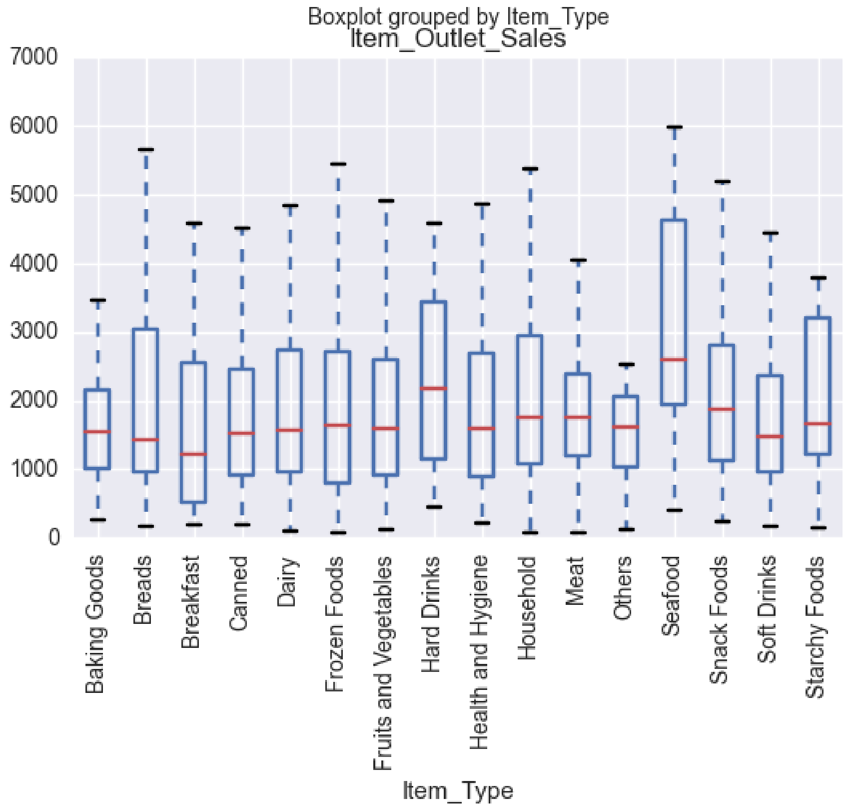
In [20]:

```
data_OUT017.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



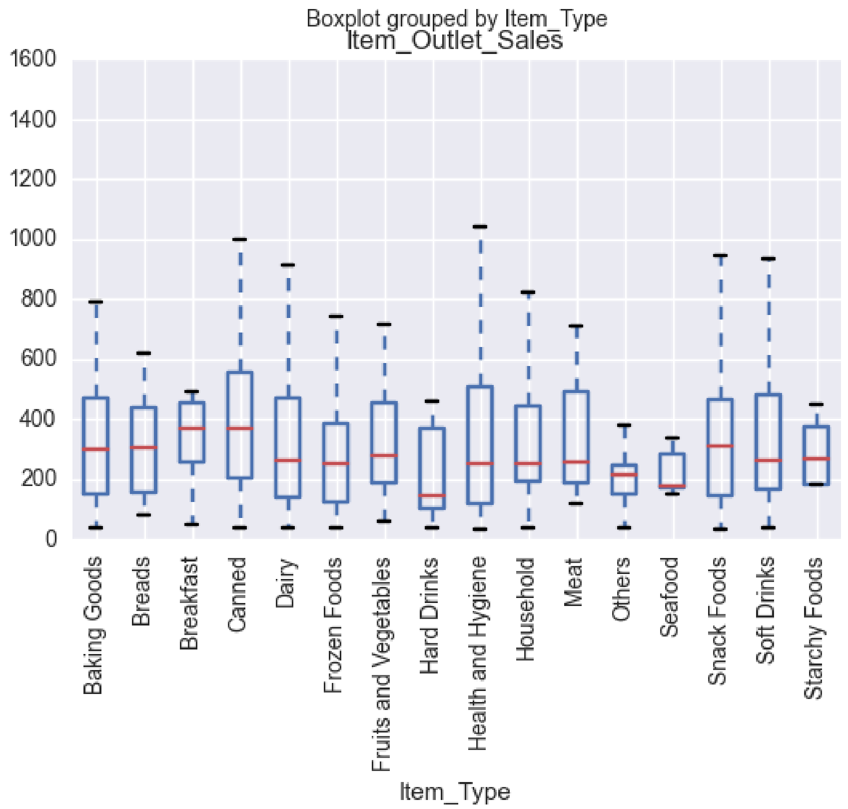
In [21]:

```
data_OUT018.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



In [22]:

```
data_OUT019.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



In [23]:

```
data_OUT027.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rdata:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAiIAAAIECAYAAAAkQUUBAAAABHNCSVQICAgIfAhki
AAlwSFlzAAAPYQAAD2EBqD+naQAAIABJREFUeJzs3XmcHHWd//HXHMlMZpIJE64EgoHw45JKCJiwjCuqwK67ym/lioso14
```

u7ubkRERESSUJv0AYiIiEh6KRARERGRxCgQERERkcQoEBEREZHEKBARERGRxCgQERERkcQoEBEREZHEKBARERGRxCgQERERkcT
IpXCzLqyFr0L3A/8p7u3Fznv7wCfdPeD81h3OHCCu99UzGNKipl1Ap9296dypHUBB7j7nwuQTxdwkLs/ldQ1NbODgN/3s8pt7n5
yiQ5HpChUIiIyOMcCY4FtgSOBjwKXJ3pEvX0B+FbSB1FEScxLkdQ1nU6438YC46Jlx8aWnZXAMYkUlEpERAZnmbsvjl4vNLMfA
X03wmLLVJH0AVSiRa+rua4HM/YaZwcb3oEjFUyAiMjSr43+YWSNwMeEJegzwO+Br7v62mZ0C/AzYw91nm9muwIvAvwAvA3Oi7a4
RgB3AOe4+/rsTM1sEqEkZh/CD9U0d78hKsq/JVqnC9jR3d/K2nYEcA3wOWAVcBFwA7AToZR0DjAV+Drwf939TDM7AvgusCvwBn
Bu98X7e8J4A/ufnH0947RPnZ097ei4zgZuBDYEvgN8JVMdZaZfQK4Ctgd+DvwHXe/N3a8FwBnRMd2QT/vRcYhZnYHocTgt8Cp7t
5mZo8Br7r7hlIEM/st8IK7T+1rZ31dUzO7EPgK0AT8kfA+z4ut93ngEmD76Jy/DdwMfAyYCfybuy/I43z6ZGbfBj7v7nvHlp0D
OXuBw712ouUgqpmRAZnw5OxmW0BnAn8IpZ+PXA08O/AJGAY8Bszq3H3m4G/EL74AW4E7nH338a2n0oIEI4FjiMENRsxs90I7Qa
D4CfAf4kZkdTSjKPxt4m/BD/HaOc/gJMBH4NPCvwLn0fuL/ODAB+ImZfQr4NXAbsBfwc+AuM9s3WrebgatLLgG+BhwM7E0IfDC
TLBwC/BhYBpwm5kdEKX/B/CfhB/TQ4D/AzQMkNdXgNOATxJ+YDPX+5eE60q079HAPwJ3DrC/XtfUzM4AvkgIHD8GLAIeNbO62H
A18CPkt4L6cTAtGPE6pZzh0g33zcCexpZh+KLfs8G5/TJl17kVJRiYjI4DxsZusJP9xNhAarXwEws1bCD+Vh7v5ktOx4YB7hB+
9R4D+Al8zsv4APEYKWuG+4+1+ibS8k/DhklwJ8GZjp7pnls6Lg5Fx3v9/M2oD1uYrvzWwkIUG6zN2fZadATySterV7v5GlH4Z8
/u/pMo7Soz+yjwDcKPcT6+7+6PRPs7E3jMzL5K+IF83N2vjdabEwU4ZwN/is71Snd/MNr2ZMAHyOsCd38sdm6PR3neB1xnZh+PG
rMeDbi7/62/nbn72uxrambnAqdlGsya2VeABcBhwIPRple5+7NR+gvAa+7+jvXxNKs4bE3eeY2TOEUrXvm9kOhOD0s7HVNvXa
5SESkREBucUwlPl3oQn20eB6Wa2JbAL4TP1dGZld19O+p8OHcLfp9Wdb+p8dezwS2jEpeoKfUYbd4HpG/ZPIYwK7
AcODZ2LIZOdZ7M2ubTc0vI/u86gjXazfgSDNbmflH+IHMPOHvRqi+AjZcv7YB8nom9voFwgPB939PePeBhwo82hJKDXw3iHIANw
2hFKhzDG/R6iK2yW26pzY6w42vqadDFyyk6872fic/uDu78bSN/Xai5SESkREBme+u2d+YGab2UxgKeGHOK+nyDo2Dvr3AdYTqh
r+K2vddVnbEa0b10HvqpS62Pr9yew/vn2uhpidfbyO55c5p+xqmVzfK7nOqyt6/Qvg+1nHs7af43s/x/7j4tcrk1dmmzuBK6Lu0
IcApw+wr1wy5/c5Ni6dqQHigWX8nCGcbzHcTaia2zk6puuz0ody7UWKTiUiIkPTTfgc1QGzCV/6kzKJZrY54QnTo7+PIrTN+Cxw
vJlljwuyb+z1foKO8XWzahn/Uz+U3KWjaJnh/h94GWWNpOOfaRfV7vR8frwC7uPifzj9
+vho3VcIXaQBMLMPAFvQv3iVx8eANfSUTvwW2IxQrfRSpvopDxuuaVSyshgYFzvmeYRqtF362L5o3H0hob3QKYQ2PNmNTTf12o
ERAZn86iRH8Aowg9aLfA/7t5uZjcBPzWzLwPLCT9ObxHq5UcBPwUucfdHzewa4AYz2zO2/5+Y2WRCMf/FhIal2a4FzjKzS4HbC
FBps4foB1oNbMPAm/Ee924+yozuxX4cXSMtVEe/TU4vQr4k5nNIFRtHAEcQ2j3AqGa5wQz+xUhWPpujn18z8zmEp62fwLc6u6r
xa4Ewzu4TQS2g/whP6KZnrQWjX8RLhh/MqBi5ZmGZmCwk/uD8BrnX3NdH5d5jZb4BzCL1Y8rXRNSX0bLrUzBZHx/Ut4B9ix52t
uJ2Ab6T0BPqUXdfkZW2qddepCRUIiIyOL8mNEpcQGh/sAtwuLvPjdK/ATwrfcnQvfeQ6PxIL5P+EH7UbTud4BmQtfKTBDwK+A
hQpXNDe4+LVq+IVCIuogeQWgY+VfCj+B/uvvt0bq/I5R8/JXQliXbN6Q4K03wH/HeVVQ0/1xUYBSdSo9d8JPVFeBk4A/sXdn4hWuR
J4Hngq2td3s/cB3EqoBngYeJzQ24ioa/GRwOHRvi8htJ25M0r/JaGx7k8IXWSfIFz7/lxB+GH9f9H638xKvQPuOuAfYTl31Nr
D0HrqRcB/sCPxTjiAgIzvQy6en0WDcS3iwzNXmZZOuvUip1HR3JzFIoYjExcbe2G6oY0vkkdfRwGOxsST2JwRNTbnGLClAfgUb
opKgL+YzXH6liLrvvgBs5e6rY8vL6tqL5KKqGZH0mQp81symEaqXLgfuL0YQUk6ixpz7E6pkymIIfDNrIXQD78vK/uYxinrwHA
cCvwyHoSIVAoFIiLloxs2THA31d1rzWwzQrXETe7+xwLlczyhPcHzqYSxU3aNFv+dUD1znbt3bMI+LwA63
f2K6O/bCJP9jR/EPvYAbnT3f8hK2olQnXJfVOWa9sdCFVlnwa2BlYQujdfkWuSvQGO4ztE72M/q11GGFumL98hd5ubjBrgJsJ1
H0ybF5GyoUBEpAy4+5tEXSszCeSqYzCeSqTPdh/BD//MC5vU34NCh7CMalfRhQi+NnxHacXQDB0avTzCzz7j7j7/AF+iLN9l/DjGzfY+u
N/oXcvH6JBzkb2tVHUCHkGoXHx+YSeMFsBk4Hfm9m/ZIa1H4R+j93dv0I0IN6mcPeVQGs/6WoHKGVPgYhIecruyTFV9nDAg1
R3fyW2/HEz+wVhEK3/Ag7b7ahH0nde5fBkYDh7j7qsxCM7uPbKDbdwsw5Gub1vImVHgYhImTGzbxE+F0dxFk6fitKWIrPp8AH/U0E9n9nDAg1
1fMu4FuZNgJRtcC/EtpCXALsTBg74rG9iSMfXKWu2fyyee49iDMnXJ+VhAChNFPo+HprzOzg939D31VU0SNKb/j7t+NXg
NcZGZT3T3nAG1R9+b/jM5pEWVvEcvSuTT/a+8zy1sYQSjI2+F6P9fpOe6qf4cXwlWl5L6MZ7qbvf01cGebxvIwi9qo4kTFL3B
zd/9R7j2KVD4V24mUr5n0jA3eYzgG8FwpP5q8BRhYtF/pZ5
27Cj/o/x5YNVMWSqU75OTmqVgCigOAGwtD6RxDGZTmPcD4Q2kvcHL2eGHudj98SZj1+2szOMbN9MpPYufvj7v7T2HF8jTCC6b3
wjtbtYAvzSzbfs49nzet6sJ1/ccQjuV3wCXm9mJgzgPkYqiEhGRMNUNPpaZkO1Vd3/NzlfyqxrZrMI1SKfcfeHosVN
InZHo3W2Z0wz83J7n5btGwqITjZhTBGRj4yDUff7OfYl5vZMsL4Ghn9VlO4+9NR+5i3MxPyxUXtUi4Ernf3TOPax81sKfBzM/u
Ru//NzOZH++u1jwHyHwAkE0GZmvyM0vn08tvp44DJ33zA8ejRo2HOEgc3uzjr2gd63w939YcKMwY+6e2b7p6I5YBYN5I
xEKokCEZHKYoQJ1y41s/jn9iyPLKbzRvl7wQhoM4CvVnyF+i2/DvOuGvd91mJnUhwlYO9ZNgmtdS53kVRwNzrfPyTK8wIzt8GS
y7pg/OVWHKBwu78PrvsQJe7CmW3OHcQvwNSoCC8hwYIrWBTvEXKrY5aaY2HiEYGjAlc/b9U9XXkVErnh+2cc9NUo2I1vWZDWxqPDb6DbhqR3+m7Prfk8
gbj7dVFPnX8izEXzScIosse2ZXu/o1ovW8ARL2adgU+CGTGJck1mV0+79vDhKq4r0RD2T8EPOTulw7lnETKnapmRCrL5tH/1x
K63sb/jQTGxVeON7qM6XNcijy9Gf3fZ5faDj7LYC5fa2zCTLn/hAbn/c7hGBlm0Jk4u4d7n6/u5/h7nsR5gr6I/D1qH0Zraz
T1OCOSeIFSlZAKMXMHXQO9b5tjPJvQ6Gk/oYj3bzKab2V6FODeRcqQSEZHK8l70/zcIP4BxNYT5bYrtN4Rqhn8
/iEzRkqNu2de99mVtg+Zc/8i8HpWWg1DqL6I2oK8QRhWf6MSCHefbWZnEkYJMxKvB/wYtSgdXdCm4/+jr3f983d3y
MBfB9M9uO0MbmQkK7ng9v6vmJlDOViIiUt+zRTl8jVLHs5O7Z/4ZHOdmZ7oeHaHsIkLLsAN0fHkDnmw6P/c32vDvi+mdkwM3Mz+3
4HYqgfk2H2uCe0yJQwzCCUI22Wde2Yenx2j9QY9Qmw0quxbwOSouiVbpsfMK4TeLLsAN0fHkDnmw6P/c32vDvi+mdkwM3Mz+3p
0TG+7+7WE+WO2H+w5iVQKlYiIlLfMk/QRZtbm7i+a2bcJs/auJ7SP2IxQNL8tYbTUUjiNUJ3wlJn9jDCZ2nrg44SutQuAf4ut/
wChW+qNZnYFISCZSmgfEa/KeA84wMw+6e5PxjN096Vmd hlwSTQ0+pOEc/4uIYB5KbYPzOwLwAx3fyPPczqTUFrxgpn9ONpfHaG
NmEBquvRft+Ezgjahj7HqGny5ei4+hV0uPu6wd639x9rZlNJ3Rffp8wEZ0RJhnss0uwSKVTiYhI+YnPzPoKofThdMIMqrj7zcA
CD/6/0NodzCHMBT63Bz7yN53Psv65e5thNFZ/5NQsnEXoWvqUYQn/P3d/e3Y+rMIP9Q7En6EzyCMWLogK/9LCdUdD5jZ9tnn4e5
Tga8Tqn4eJJS8PAUcGI0yCuFH+1ngNkJVSL7n9DzwEcJMu2dE+78vOs+z3P1rsdWPJpRm3E6YwXgc8DHC+xUv6Ykfez7v29cIX
DPIcwe/G1CF+TTEKlSg55918waiMY3yH5iibrXvUoYoOf22PJDCf3jxxOKVyfHn1LM7GxgCmECrruBMzLzVETjG/yM8MXTQZjz
cpBnqeIiIiUoUFVzURBwS+B3cn9FDWN8GTQHdtme8KkWhcCjwAXRX/vHaUfFy07nlCHehthIqgzol1cDuxL6Bq3I3C7mc11918
thFpH9m9hFydz2NWxHNVVMRzGxLwgisA/nfWImKiJRQ3oFI1CI854yVUfoBwKcIXeniJgPPuPtV0XonAe+Y2YHRbJdAVdlBmE
s1OBR81sCqF+9hTgMHd/EXgxqiM+HVAgIlJY9wI7DLDOE4TPeaX4LGEI+P50Ex50BjW7rogUxmBKRA4k1J1eQNY2EIa
ytpuIrEPuLt3mNnzwKSoYdZ+RHNDRJ4GhhNKTOqAYW8KNN0NN21SMG5e5/jglSqaATZ2xI+DBHpR96NVd39enc/J9N2I8u3CK
+H8+RNpbQIC1uEbAdYabLxni6u68Dlkbp44B3o2XxbRvNbHNEREREkog25+25UZXMqYSbPXJoIk0HFrSHURTfF/s6VXtdHGgxcl
0iIiJlbkiBSDSR002E6b2XxJLi4wJ00jtoaCSMJNgZ/Z2d3gCsJlTL5EojSh9Qd3d3d01NIaa7EBERkUHI68d3qCUi2xMmotrL
ULWsCrjezz7v7Zwl97cdlbTeWMPDSUkIwMpZoyOZoQqjNgYWEEpEtzKw2NnrhWKDD3d8jD8uWtVNbO/hApK6ulpaWEbS1dbB+f
a7BHotD+Spf5at8la/yrYZ8W1ub81pvqIHI24RZJzNqCK3qfwz8V7RsBrEBfsysCdiHUIrSbWbPAp+gp0HrJMKQzS8R2rCsjZZ
j9IPAPKe3rurq5uurkGP17TB+vVdrFtXujde+Spf5at8la/yTVO+QwpEovkZ5sSXmdk6YLG7L4wW3QJMMbPzCCMqTgXmxAZDu5
7PErhEar1wE3untntL/bCSUsJxEasJ4DnDiU4xYREZHyUPQh3qOhi48FTiKUZLQShkfOpN9FmMHzBuBR4C/AubFdfJ0wkusfCN
iT3X3+4t93CIiIlJ8m1Qi4u59BjC5xiJw90fomb0y1zbTCKOy5krrIJSAnDjY4xQREZHypknvREREJDEKREREREQxCkREREQkM
REJDEKREREREQxCkREREQkMQpEREREJDEKREREREQxCkREREQkMQpEREREJDEKREREREQxCkREREQkMQpEREREJDEKREREREQx
EQkMQpEREREJDEKREREREQxCkREREQkMQpEREREJDEKREREREQxCkREREQkMQpEREREJDEKREREREQxCkREREQkMQpEREREJDE
RCQxCkREREQkMQpEREREJDEKREREREQxCkREREQkMQpEREREJDH1g93AzBqAmcDX3P3JaNlY4HbCsM9LgW+7+yOxbQ4CbgYmA
Y8AX3X3NQfZ/YZRY4ijh3u5XL2fFwRegWCdlmbNaNgovSFe6WRE+vJZqzbaxqSmnLRZAOvpk/6I2bLbSZnVYr9ZLAiaSv/eMy2
Y7O5vxNLPBqYAo4C7gTPcJOsNKawR+BhwLdABXuPuVgz9VERERKTeDKhGJgLxEpxwuEm5qxcy0DrEO1Aei1/HeLBh4PfAPsBFwD
SfR3Zp/HRdt8GfgUMBG4LJbt5cC+wAzgKzlZ/hYaMs84AAyPLcTBTz1ZV3Q7mOTP/dp8UCrwSLEtt/LjnjtbhdNcJNLOh4WAt+zmSZqMqBzkP9wv53E/W9NYb5I5pS/+tzCNE4pafDx
D3f0p4CzgKnd/KEo/FXjDzDYDZgI3Au3AMnfPIpdzzcyavRMrOLfnCcObxCmlYMMZ7Dz0VAVZxuddO0B9pVkJBCNN61ZdCxH6tvZ07o9vUqAlzXvthwkJHg6Byrv1
ZWR2hlOSp2LZPA8OBvaN/w4A/x9KnAx8bxHGLiIhcq7RMTdr8+8NrP48rnA3FjaVsC/AVOjRWOBBVm7WwRsB4wGGuPp7r9Ozj
G6QDvuvu6rG0bzWxd1+a7/GLiIhI+R0Y9X+mNkIQpXJAuCGaHETsCZr1TVAQ5RGP+11faQRpQ+otraG2tqafFbdSF1d7Ub/l4
ryVb7KV/kqX+WbnwLFoiY2UjjgN8AHgQPcgN8QYCmwcdmm5dfRtwAEJvnSPXvrQIT4UCMspU9vtAGE+EMloaRmxydsOhfJVvsp
+Spf5ZuGfAsSiJhZC6GHj0vmZuj/dlSJcdZWQEf9ha9ch5ZhB9mb9dDo+SpNY
BW1sH69d3DbxBgShf5at8la/yVb7VkG9ra3Ne6w05EDGzuBeYEfgk+7+etYqvzWJaXAzg5ZcIjW

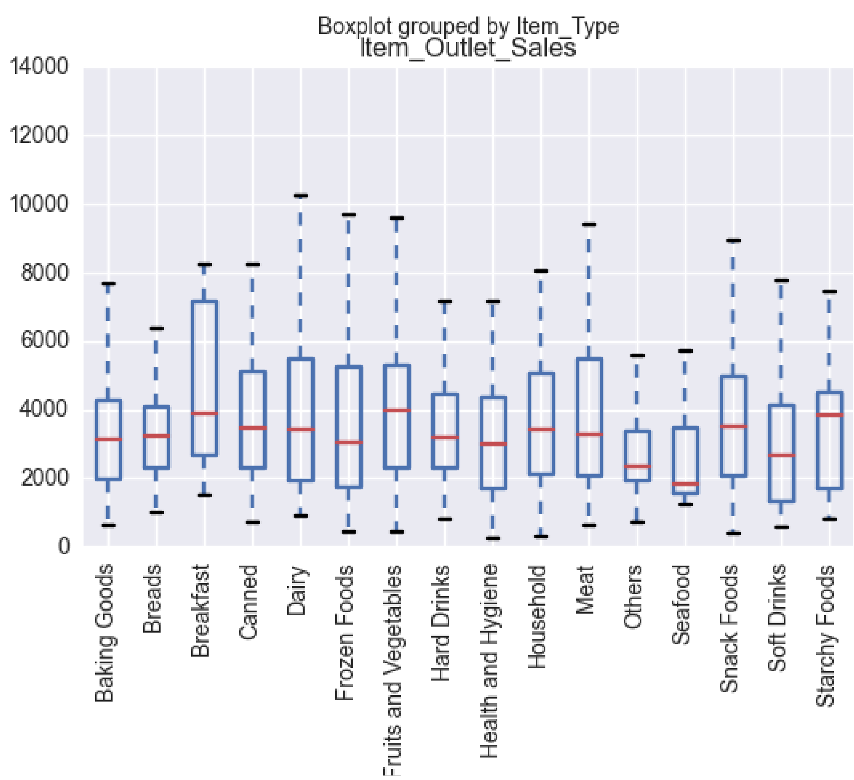
XRsumR+KHAM/ke3xdXd1UdXUP8qx6rF/IxbplpXvjla/yvb/Kv/kq3zT1W4gSkvUAg4B/BtqlcU0A3hf3ZcAtWBQzOw94gNC1dU
mMDTgWuAGM3uF0LbkOuDGTNWOmd0OXB91+90OOAc4sQDHLSIiIgkrRAuUYwnddR8gBBKZf/fAhl41xxK6+D4DtBLGHiFKvwv4Aa
Fx66PAX4BzY/v/OmEk1z8A1xBKUu5HREREKt4mlYi4e23s9eF5rP8IsGs/6dOAaX2kdRBKQE4c7HGKiIhIedOkdyIiIpIYBSIiI
KSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAUiIiIikhgFI
pIYBSIiIiKSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAU
khgFIiIiIpIYBSIiIiKSGAUiIiIikhgFIiIiIpIYBSIiIiKSGAUiIiIikpj6pA9ApFTa2lYwa9brANTV1dLSMoK2tg7Wr+8C4E
Mf2oWWltFJHqKISOoMOhAxswZgJvA1d38yWjYeuAmYCMwFznb3x2LbHApcDYwHZgCT3f2NWPrZwBRGRgFHA3cIa7d0RpjcDPgGOBI
79y8KcqadbWtoIJE/ZkxYr3+lxn9OjNmDnzZQUjIiIlNKiqmSgouBPYHeiOltUA9wMLMLGAnAL4D7zOwDUfr2UfrNwH7AkujvzD6
Cvgx8ihDMXBbL9nJgX+Bg4KvARdE2iIiIuUHyLhExs92BX+ZIOhjYCZgYlYlWL80MwOAU4GLgYmA8+4+1XRfk4C3jGzA939KeAs4C
p3fyhKPxV41MymAHXAKcBh7v4v4i8KKZXQacDvx6k85YUqmlZTQzZ77MrFmvM//ddm596DUAJh+xG2PHNAGqmhERScJgqmYOBH4H
AC0x5ZPBGZmqlIifwImxdKfyiS4e4eZPQ9MMrPphFKSqbFtnwaGA3sTApFhwJ9j6dOBbw/iuEWAEIxMmLA/E4BjP/spWlubWb6
nXXrupI+NBGR1Mo7EHH36zOvzSyeNA5YmLX6YmC76PVYQrVN3KIofTTQGE9393VmtjS2/bvuvi5r7z7/CIiIlJ+Ct
FrpglYk7VsDdCQR3pT7O9c6XV9pBHbf79qa2uora3JZ9WN1NXVbvR/qShf5at81a/yVb5pyrcQgUgHsHnWsgZ6qm866R00NALLo
RypDcAqwnVMrnSiNIHNGZMMzU1gw9EMlpaRmzytkOhfJWv8lW+ylf5piHfQgQi84E9spaNpae6Zj6h+iY7/XlgKSEYGQu8DmBm9
YTAZiGhRGQLM6t1967Yth3u3nc/zJhly9o9uUQke5yJUlC+ylf5Kl/lq3yrId/W1ua81itEIP0cL6ZNbp7poTjAHoaqM6I/gb
AzJqAfYCp7t5tZs8Cn4itPwlYC7Pbbo2/PbvZ/I9uK6ubrq6ujflvABYv74rkcaMylf5Kl/lq3yVbxryLUQg8gQwD7jVzI
4HHEnoCXNClH4LMMXMzgMeIPSQmZMDIPSQmZMZDA24FrjBzF4hNFq9DrgxE9SY2e3A9VG33G33+2Ac4ATC3DcIiIikrAht0CJqkyO
ETjG3d+O0ucSRkU9iVCS0QocHdv+LuAHwA3Ao8BfgHNjWXydMJLrH4BrCCUp9yMiIlVb5NKRNy9Nuvv2cBB/az/CLBrP+nTgG
l9pHUQSkBOHPyRivS2aNlq7v3jHIYPq+eoA3Zki5bGouanOW5ERPqmSe8kdVZ1ruXZvy0G4NAJ2xY1ENEcNyIi/VMgIlKlVBIjI
pVAgYhIEcXnuIHSBQQqiRGRSqFARKTIMnPcANTX12qOGxGRGAUiIiVSykaymm1YRCqFAhGREiIlI1noKYnZbMEKWl8IpS8f3mtf
VFHzldJQGyCpFgpEREQqjNoASTVRICKpM7JxGPvvDXh9XTPGJY0ocjzRZr9Qx+2lRqNSsZLqjSVSDApEREQqkHp
jSbUY8lwzIiIiIptKJSIiVU5tYkSnQkEQSmRpAICtYkRkXKmQESkRBQQiIj0pjYiIiIkhiViIiIiPRBI9gWnwIRSZ3ZC1Zw6l
jopP015LlUNN3PxaMRbEtDVTMiIiKSGJWIiIiI5BAfwXZZ2xqeeHE+9fV1HLj3ODZrHg6oaqYQFIiIiIj0IT6C7eH/qBFsi0GBi
EiVUxsCESlnCkRESkQBgYhIb2qsKiIiIolRICIiIiKJUdWMpM7WrU2cfuyeNI9sYKUkfsjsiQ6H6WSqdARFJn5IhhfHT3rdX
7mepdKqaERERkcSoRERERGRGQA6vVWPApERKqc2hCISDlTICJSIkkFBGpDICLlTIFIiiU1vXVap9VWQCAi0lvBAhEz2wX4fAxYC
/OEobD9wETATmAme7+2OxbGAzOAye7+Riz9bGAKMBAGKMAq4GzjD3TsKdexplNT01ppWW0RE4grZa+Z+Z+YAnwEeAs4FIzO9rMaq
K0BcAE4BfAfWb2AQ2AzAz2z5KvxnYL9ZZ2Kvxn/YL9rH/Zmdmtlx/wEXAl4FPEYKZywp43syri/jTS/NZufr9pA9HZEh0P0ulK0iJiJlt(
ewKfM7dZwOzzewR4BCgDdgJmBiVYvzQzA4BTgYuBiYDz7j7VdG+TgLeMbMD3f0pQlBzlbs/FKWfCjxqZlPcvbMQx59G8emt57/
bzq0PvQbA5CN2Y+yYJqA4VSTxfCGZqplFy1fz03tfBtT6XSqf7mepdIWqmlkGzAFONrPzgZ2BfwC+RSjBmJlVlfInYFL0eiLwV
bB3TvM7HlgkplNJ5SSTI1t+zQwHNg7ei2bKDO99WYLVtD6QggCPrzXvkX/IotPq11fr2m1pbKltc2TSKEUJBBx9/VmdgzwJKEE
w641d1vNbNrgIVZmywGtotejtjVU28QtitJHA43xdHdfZ2ZLo3QFIiKSGLV5Sg91gy+eQlXNNBEakT4O/BDYE7jGzH4H4HjADWZG2
BmiIXjf1k94U+7uv7ftVW1tDbW1NPqtupK6udqP/SyWJfOtjedXW1lBfX7q8dt7Ft3L1+/jf32NE03DGbz2SpobSdSar9s9Rvvo
OfX2EPrI5AAAgAElEQVRt0e6ztN3PSeW72agGJu05rleJVyU1Ur/3Ct+xrgeTPbFrgA+D2Ul3PJEgu4zcOWm3aNeVbUwpnZw4kPNUmpmH++6
A8SiNHegOwOp8DGzOmmZqawQciGS0tyUS+m3/m2tb/PX/++BIC9eSnf8s5Iw/m2tb/jJPX8X4Iv9/mSO9jJZAIz
8G2Y0v/ZF6tn6PW1mbmzn2T1157rc91dPLKCkQ+AMyQ/gQmQxZA37ug/CQyl6bVSmGrPJ8VW/3Fz4/1YaaxNdIfs/4gLmNhHcsoJ3YA7
o79fBzCzekJgk13dk9OyeZxNsq95N95tVRPVf9uDEDBVbZ0LYYlF8efvAGxVIEue7cmVPG+f29jUlO9/Z81cw7Y7nALj4lI8
yflxLSfJN6nwhLZ+jenbZ5cN95tvVRVGvud5f5Vuu+eYbFBcqEPk78EEzG+bua6NlY4ElwEeAXQEmKHaW/wjQdKnwmcSimxQGwatel
8w1d27zexZ4BOx8cBa4GX8jmwrq5uurq6rfyoF6A4/aiMIlyo9DtY3U18j2BNHyOkspX76/yrfR8
CxWIPEDopvtzM/seIQj5JqHXzZJPAPODc/cko/VrgBjN7dk80hFCtytsxBAQdRnXdrXyrOtby+rz3aB7
ZwPZbNDFiuAb6FRGBdPXGKlSvmTX6Kg/BZ419Iq5xN1vAjCzowgDhVL0HzAKOcfe3o23nmtmxwPfB1tcqz2/D00x3zG0i3m0GR
25Bzi3EMctyUpq/IOdtxnNHRccqm7DUlBJBda6n6tP2npjFeyTEg3J/tk+0mYDB/Wz7SOEUpS+0qcB04Z4z4iANKUwQqIoWlgcVEN
o3KwiNpi0BFRCR/pSzxKocRqEtJgYiIiMgASl3ilaYRqBWIRJKae0Wk2NSGQKRyzV6wgkvvmAlUb5WfApGYpOZ6eSZp6r5SGAgIP
0iyNqZCciIkAynTYUiIiIiEhinTYUiEhikqqqWLRsNff+cQ7Dh9Vz1AE7skVLY0nyFSkG3c9S6RSI5KDdpnqvbqs61PPu3xQAcOm
fXFLQSQVWOt+lkJJqtuwApEcRo4Yxkd33RQ5kDasQERERGQA6vVWPApEUmR15zoWLus9Rfj8JT3L3l6yaqPZWm
G3TKUpdRuCpKZK0BQNUo2SaipQymEd9KuSEqs713HudX9m9Zp1/a538wN/6Zp1/a538wN/6zOtqaGey077uIKRTZRUo8JStiFIqtW9pmiQapVU
FSDutQW7Q9S1lZuKx9wCBkIKvX5C5RkfxkAoLpf11Ae8fapA9HRKQs6NE2hU48fFe23bJ5o2X1dbWMGtXIypWdvapm5i9p57aH
UqFSmqqhHi+HWvW8daiVTQ2DWPs6EYahtUVLV8RGToFIim07ZbN7LzNxl/ISUyqlFRVxcjGYey/21YMH1ZP84hhJckZTZKaKiGT
7+wFK7jlqZlAFxedtEfVjxSs+1kqnQKRKHDT3SmkkNf7B1mOaO004vdT7XQoqqcBa97NUOrOHSXSGyEEehIgUlRl8gMglVYRwRiGT
ENo0e9UVERAagofSLR4GISJVTGwKRKJVTGwKRoUuqKjjNTQ74xEylRSAYEIhIn10i+VBjVREREeUmMS
QkMWq7IMWgkhYRERFRRhAhkQtuPSOmUBp61yluiUVWOt+lkqnQCRFSt2GQKsRmqSO7BDMp4wrOyGIi.
wKRMpAW9sKZs16HYC6ulpaWkbQ1tbB+mgWXM0aKkOhNgQilSuppgKlHElWgUjC2tpWMGHCnqxY8V6f64wevRkZ76sYKTCpS0g
YIpWrlCPJahwRESkKTQInIvkoWImImTUAPwK+AKwDbnb3b0Vp44GbInAXOBsd38stu2hwNXAGNd34ilnw1MAUYBdwNnuH
TIvb0jKamTNfVtWMSIGoJEakshSyauZHwKsKhAPwEtwK/M7E13vxG4H3GgJmAACA9xnZru5+zwz2z5KvxB4BLGo+ntvADM7Llp2PLA
YuA24DDijgVMeeqJaW0UyYsD8A9fW1qVRxatTZ+x7J40j2BAhEzawBZ+Dxz+2EqmD1EEx4EfNr
TM/g7sBEyMSjF+aGaHACcDwFwOTgWfc/apouVcD/apou5OAd8zsQd/8zsQHd/CjjLuMrdH4rSTWUeZZiOPPPpieq6jZyxDA+uvvWqQn4pDSSC
qx1P0ulK1QbkQOAOwANnf/Y2aPmE98OAEbuQDsIW8yN7rBMSNUf6GdC3A7EmdOgsyJFgHmYHTvuv3GeyITBumEKvHS6mR7BbCQUDEf
wK+BQwDbgUuBcYC7PWXwxsF70eCzZZ6f1s8wCxfR9SU8F5cU4FmLPO6TvxMXCCeCZdWHIxJdCsf+FCrXIj3g6U8X6PhFRET6llZeb6VUqEBkb
/TVAQ/S6qZ/0ptjffW0vIv1QGwKRypVUU4FSjiRbqEBkHaGB6hfdR5A1Aj1q8BjwOZZ6zcA7dHRdTnoHFY3A8iiNHOkNhCBnQLW
TW1uSz6gb1dT01VrW1NdTXl6aX8zvLVnPPE7MZVl/HsQeOZ8vNCvejET+n+rraXudUF6XX1fU+14G2LcQxlfI6Q//nWyzbbNHMm
Z/bixFNwxm3RXPJznezUQ1M2nNcr95YxTZ6ZAMf3X1rhtXX0dI8vGTnm9R9lbb7OW35JvX+dq5dv6Hx9af3365k+W671UjO/vw
+JfneKFQgshDozAQhkdeBDwDgT2y1h9LT3jb14mnYhcm5LlnwaQ1ZPLzJoLQxdhJH0aqUxl0aDRpIPA35VZ6bGxbuGjOmmZqawQUio9p6CmCamxtobW0e1I
HbGp55dREAxxy0cOHzjZ/TqFGNFe67paV38JPPvtkM5plJe57hc51ssra3NbL9ta8nyy6XU53vhzluWLL+MpO6XxXK
CoQmQE0OmtmH3H1WtGw4Xwza4z1cjmAngaoM6K/ATCzJmAfYKq7d5dtdsd8ufqvSWi16/2huMyAx7hA1sSsW+G0ZP8IDwBVJ
/ezduEUM9/4vleu7Oy171zjl+S7bSGOqZTXKj6/nfIOmggQQi7u5m9iBw
5mdRijhOA+4BHgsnAfcambfArjuhA++5NR+rXADWfl23e3q6q8tQQ57MudsG7urpL
h2umPnG971ufVef+16fIy3fbQdr85bGDW0XthjdWLLrvKpjLa/Pe4/mkQ1sv0UTI4aXfxqaDXNdZ+RbGFtF9VerurEl9b+h+ru73
N61844p9nQt5xx4PXEPomrsauMbdfwpgZkcBNwPPPAbOAYu9z9bQB3n2tmxxJGVp0KTAeOzuzU3e8ysx0jJV8bgHuAcwt43Llouu
8g0XLV/PTe18G1PpdCiepRsG6n6XSFSwQcfc2c2QinHCTnSZgMH9bPtI8Cu/aRPG2F5YN/SjzoexXu2XdFKKSsl
Ms5eoGXzwKRESqXDm0IRCpdEmVeKwhqYC+kcpIGm64NEsqIFAbAimkJJKpI0jxLeVJNBUo5kqwCkViVLG0BgYbErj5JVpGk
yNFAgIlVrdec6Fi7r3ed+/pKeVWr97i4cWOaNUQ7MxPkNkZOWtfYtnx3cBR/lhn2lNDfVcdt
rH9eVdYaqxJKZc7uc0V5FI8egbvVhJTzKqKhcvaB/zSHsjqNeEJdodt9KUqAytmYF1O97OqSKTQFIjxUG24uG7su2UtG81X
Xy6hRjaxc2dmrKHv+knZue/i1Uh6iVIFStQFK+/2c1Gy0UjwKRKTqbbtlwlcRl1WdoiohG3
x6EqKFFgSAUG5tCGQ9EhbyUTaer2Vkr5xRAooqYCgnNoQSOVSkBVIUgMw3+kOSFF+e/CXECbaz8rZzt2tv63MYUkP/wAeS1rnOw7LJtaeIhv
suJBUQlEsbAk0CJyL5qLxvd6koaW67UC4BQdoUuyQmrYG1SLHo0yBFVQ5VFZIuxSyxMyGsUFbJLB2szA1QAgQnn6u82ygaE4XKr6Ctgsim0qBdfKk
TmoSrxKT1crB9VtF4eqKqSaKLBORjEnB1WJVzJ0pURENkHaA+til0wkQSVeyAgIiig1bMkolyUC4lXkk1FSjlSLIKRMqI2qZ
dGQ2FLJyqXEK6mmAqUcSVbfDGVEbVOkmmhIbBHJwpEtQBdeAFEbeWBAyW2jO/VpP1VMHUgO/Yx+b2w
UgOeqISESlPsxes4Evfe5wjz/kNs+evSPpwpwpAD0qC8iIpJS5TCrArIREmScvgik9KbvwAF194xE1CvqGIqZlOBchlJVp/+MqK

2KVJpyuWLTKRaFbMbfLmMJKtPfhhRuAtSaZL8I1NJjEjhJDmSrD6FlIIQpfw10UmMSGEIOZJsqj+BeqISKZxSfpGVS5Gy1Axdw
9FzexBYLG7nxT9PR64CZgIzAXOdvfHYusfClwNjAdmAJPd/Y1Y+tnAFGAUcDdwhrsPuU+tnqhEqkO5TE4mIpumoL+gZvZvwOHAk
dHfNcD9wEvABOAY4D4z283d55nZ9lH6hcAjwEXR33tH2x8XLTseWBzt9zLgjKEeq56oRKpDuUxOJqWhyUGrT8ECETMbA1wOPBtk
EzAxKsX4oZkdApwMXAxMBp5x96uifZwEvGNmB7r7U8BZwFXu/lCUfirlqirwqJlNcffOQh27nqhERCpDtU4OmuamAoU86iuA24FtgZp
2URgZlZVyp+ASbH0pzIJ7t5hZs8Dk8xsOrAfMDW27dydOFOvBSP1Gl+YYTkeqgkonCSXtTgYIcsZl9CjgA2Au4Hsj8go4
FmatvhjYLno9FliQlb4oSh8NNMbT3X2dmS2N0gsWiJRS2m84EakO1VoykYS0NxUY8i+ZmTUSgo+vuXunmXXHkpuANVmbrAEa8kh
viv3d1/YVJ+03nBRPw/r3GbN2Bd3z3qSjc+RGaXV1tdSPamT1yk7WZ5W0dS9ZxbjOJSwbpvtJJGlpbCpQiEfqi4DnYj1hamJpnc
rPUbgPZYenZQ0Qgsj9LIkd4ArM734Gpra6itrem1vL6udqPX9fUbz/9XF6XX1fWeF3CgbfsT3/aUz+7Gdltt/INRW1tDc7MD7e1
r6Orq3ijt7cWruPnBvw0536TOV/kWL9+6tWs4be69NHa9T9dNDzMv7y2DE4DO2uHUrd23Is5X+ZYm34H0l28xFSvfcnh/dxg7i
p233fihoK6ulpaWEbS1dfR6kKiG+6oQgci/AmPNbGX0dwOAmX0O+D6we9Y0D+6Y+mprplPqL7JTn8eWEoIRsYCr0f7rAc2p3d1T5/
GjGmmpqZ3IDKqraegZdSoRlpbm3utA9DS0rvuM99tc4lvu9vOW2A7ZMdp/Ww7d9mQ8s08MTcuXUD9sFW91lm5MESR2TdF49K2DU
/MQznfpK5zGvIdObKBtrzX7n8/lXC+yrc0+eYrV76lUOh80/b+lst9VYhA5KDYfmqAaUA3cB6wA3C+mTXGerkcQE8D1RnR3wCYW
ROwDzDV3bvbvN7FngE7H1JwFrCd2B87JsWXvOEpGVKzs3er18+caNR/uLQAfatj9J5du2ZPmGJ+YVVz7MX/PeMsg8Mbct2ZflLfnX
jKXtOieVb/u6Wq7b4VjGrF3B5CN2Z9sts6pmYiVt67NK2uYvWcXPH3iVZcNGc/662oo4X+VbmnwH0l++xVSsfNP2/q5c2bnhAXX
Fq68xf0n+3xsrYlW6feWbb3Ay5EDE3d+K/21mq4Bud59J8m8C84Bbzex4wjGEnjAnRKvfAkws/wb9OABwg9ZOa4+5NR+rXADWb2Cq
HR6nXAjYPputvV1d2rigPYqJ5t3fquPhtarc+Rlu+2uZRDvkNRieeblnzX1A1nYd2WdG2zPcNz9AIb1drMuhyNCruGrWBh45JNz
fY66m65ymfPOVK99iKebkoGl7f9e2t294QF17/cO8mfeWQeYBdW37Pqxbt+lzoxWj20V39A937zKzo4CbgeeAWcAx7v52I
ljKw6FZgOHJ3ZkbvfZWY7AjcQnzuAc4twjFXvZrGERuemE86fNfeT8x1tbSMaqQtR2PG+UtWcevDr7Fs2Gi+0ahueiKSHE0OWr
0KHohkhnaP/T2bUH3T1/qPALv2kz6NUN0jQ5R5Yq75wI6MGMQTc03jChY2Li3loYpImStmyYSURk8oOrOERGRQVPJRHUohwfU
EhEREYlRICIiIKJUdWMiIhIRCMUl54CESk6fbBFZLCSmBy0u7OjICMUd3fuQ5guTfKhQESKKskPtgIgKQbdV8WnyUHTRe+QVC
U92UgxpPG+SqJkIqnJQculO2vaKBCRotIHW6RylUPJRKlnoy2H7qxpo0BEii6JD7YCICmGJO+rJKqEkiqZiNt2y+Ze29bX19La
szyHN8bUnkUiCRAdcyloScbKYYk7qtyqBIqdcmEpIcCkRIrhy8UkUJSYJ0OKpmQYk11IKIvUJGhUWBdGqpqlGqW2kAkqS9QfaG
VaNUq9QGIknSF0p1S1NJmwJrERmq1AYi+gKVYkhjVUVSgXWaAj6RapbaQARUMiFSqdIY8IlUq1QHIiKFppI2EZHBUSAiUmAqaS
+BXwi1UOBiIhUJAV8ItWhNukDEBERkfRSICIiIiKJUdVMCs1Z0NZrWX1dLaPa1vvQ5Z4SIiBRPmrujKxBJiY7YDJp3Pj4rwSMREZ
dHV9VMSmyzefPAKw2gqaGecWOGvh8REZEMlYikxJiWRq746sdZsLSdEQ0bv+3x6bpPOWI3xo5pyrmPcWOaaWrULSMiUkhp746uX
UUGdPSyJiWxn7X2W7Lkeyw9agSHZGIiEC6u6OrakZEREQSo0BEREREEqOqGRER6VOau5VKaSgQSVNI2/kmJanrrPe3+qS9
W6mUhgKREKvbeB5pO9+kJHd9f6f6KyFAVLBAxs22BHwMMHAx3AXcC33H2NmY0HbgsNQi42MAUHiSiEOBs939sdi2xgldjln42MAU
YBdwNnOHuHYU69l+lqx/E8tm5nm81Suo66/2tbnnv66/2tbmnvViqlUZBAxMxqgHuApcAhALcB64FzfgfuAlYAL
wDHCfme3m7vPMbPbPso/ULgEeCi6O9+9o30fFyF8dPetaW1tZnmOt7mfDUY7nW42Sus56f6tfm
SmkU6tNvwMeArd19CYCZTQWuMLOHgZ2AiVEpxg7k8BZwFXuftDUfqpwKNmNsXdOwt0/C
WVtvE80na+SUnqOqf1/VWbGJHCKFQgshD4p0wQEqkhtE6aCDyfVZXyJ2BS9Hoi8QFQf2A6bGtn0aGE4oMXm6
IjKgcmgTowCoNHSdS6cggYi7rwDibT5qgdOBx4FxwIKsTRYD4stWYD20Wvx+ZIXxSljwYa4+nuvs7MlkbpCkREpGSSahNTDgFQGug6J6N
FbOXAfsA+wNfB9Zkpa8BGqW9NreX1dz1huc99ZudHfme0Wt62hvX0NXV3dG9s3T1Rvupry/MuH
DxY6itrSnYfvNRF+VdV1e6PIt5vvF953qP+jvfgbYtx3zzPaZS3lfV+P5uNaaJq844gAXvtvdq1/L2k1Xc/MDfAPjykbuzzRa5g
41xmw++TcwHthp6tVZTYz0f2GpURdzPSeWr65zM91XBAxEzm0Zo1/F5d3/VzDoaJBoVz9oVm6aM
pregUjt4p5itP/76Ov57CqnUaMaaW0tTIv/UW098VZzc0PB9jsYLS2la91ezPON73vB8g5GjerdhmHh8twdr5a1r+3ZzyDf33i
+/W2b6zrnu+1gJXVfler9LfV1bm1t5oM7Zn+1wai5yza8tvGbYzuMGdR+B8rzlgs+zbzFK2nOCmLmvtPGNXe/BMAZn9+bHca
zHdluNonnEsEHlm7bPUTlc51Keb7l8XxU0EDGza4CmAV+AR4vnAHlm7+1rmrjiW0K8uR/jyjyFO5n9PfrUR71hhMBmIXlYtqw9
4lIS0NdPpv3q6mxnpHD61i+vDB1gytX9rS9bW9fU7fU7D95qOurpaWlhG0tXX0VLVm14fdP9r2zyflau7BzUccXPKde2
/V3ngbbdVEndV8XMN43XuR4Yv1XvL/uVK4dveL3FqAa2asldWPx+5/u83/n/oPJM4+coieuc1PkWO998NyEAq8K/ufm8
saQZwvpk1xnq5HEBPA9UZ0d+Z/TQRqnWmunu3mT0LfCK/2CK2/iCC6urp7Va0AjG4eXpBuh8PrawvW3XzlsYN43lsMbqxoN
+7OqYy2vz3uP5pENbL9FEyOGl6YrZTHPd+sCjIfS1FDPlqNHDOq44g3Y1q3v6nPb9TnS8tl2sJK6r4qZbzle5/h+u7q6S3adi5l
vkp+jzNDya9+cw8r23uOX1IxqZGWO8UvWxoaWr5T3N6n7uVw+T3N6n7uVw+R4UaR2Q3wjgglwLTzWxsLPlJYB5wq9FmCj9FmCKmZ
PEDoITPH3Z+M0q8FbjCzVwiNjpa2CDj/89W+ZXkN2mEb++deVq/IxxPN7MfLzXLLi5RqV1bbdW+n1CdQ3Wb3GHHZYDqzOdZ28hkgW7pLmsMXtfiyQg8yrVjmXkyTQMMjn1z1n72IT0r/k74gvM3p2C5UI0HNzG
bM5Hth9C+j293rzOwo4GbgOWAWcIy7vw3g7nPN7nPN7FjCyKpTgenA0ZkduPtEgcjAOhbcg9hEHSRPqVdKCp7n9SDZL+HEn1K1T3
WnAtH7SZwMH9ZP+CLDrpu5fpFyo+1/pKeCrPhpavvSS/BxVfrmzSBnR3CulUY4BXzHnbEojDS1ffOXyOVIgIlJAaptSGuUY8CXV
YBUGlU43Uul8+Rvu1ECkx16sWngK9HtTcKLpeqt2q8zuXyOar8T2ERVGPkm0tb2wpmzXqd+e+2s3xhGNTtlb/W8W50w33oQ7vQ
qIW51KeFPBVr3KpMkiDcvgcKRDJIanIt5TjebS1rWDChD1ZseK9jZZPv7Pn9ejRmzFz5stFC0aSGr8kLYFmRlLXOal8pTK
qlyoDKQ19M5SRpMbzSEo1jl9SjpK6zPmkys2aFKplcQ/mqakZEklIOVQZSGzQyPVh
waSkhEBEpV+XSOLeUFIiIlEja2qakjdrdErEyKZKe+NcfVJESiRtbVOSEnIh
2Cy6GXXzGvczk2zi3l50iBSA5pecIQEZHykObGufqFzSENTxhppkBTZOhK+TlSL7/qpm/gMrLzNqO544JDU10knNT5pi3QTOo6p
+1+TptSf47Uy696KRApA5lGWKBIX0RE0kWBSML+xGWCIiIklRICKPlQ3d/0REHUgGf2sRIMSQ1
Lk4pP0cKRMqAGmFJNUm6zVPaj4j4FQKWR1HUu9bg48ZLjWx8K45dMjo1fUozPrwIRSQWVPJWG2jyJVK6kSo4ViOSgJ4ziSuqJWS
XdFsrqT4KRKSk9MRceouWrebe85H+LB6jjpgR7YYYYPTGoVDJU3Urh7ZWUjxJfX4ViMQkXbctUGyrOtfy7N8WA3DohG2LGoiAS
Rd9XUgxJfH4ViET0pF4aemIWGbqkvq/0+S2dNAWaCkSk5PTELFK59PktqQjZZjqprY4IptGgUiMIn0phqSKWNI
auyGpgE9tcaqbAs3iUSAiUkRJFbGqd4NYcQD+kd/Hwb4GjusuAN8QboCTQUiIiIiOSigLw0FIiJFRa9qKdpOU1iowkUJRI
ZEkVsaapaBeSaYujJ+bqpoC+NCoiEDGzRuBnwLFlFAB3CFu1+Z7FGJSLIW3W3dHKZ20BfRJqIhABLgc2Bc4NCoeMt3M3nb3e5MlhEEDI+Z7FGJSLIW3WfHKUNwRuNM5rr7xW7QhF
T0xiwxd2QciZtYMnAic5u4vufuVA+A3GBRH3VUiPeIPIOQWqKRUDXUTHebKDzRPQcaTfa5HXVWKYHU0UCjj5wVuAA8LVCYsrsXhiAsQDGUUFnqvvQcaP0
QIazWzzhI5JRERECqASSkSagDVZQyJ/Nwy0cW1tDbW1NYPtK6ydqP/S0X5Kl/lq3yVr/JNU76ePV1lrPVfkJfNPvFfDJaJM4wlD/HKkEoInBvFjqbKUF1hsQTbdDu/mQSDNiWNUXGfTUzvK5KnndKMyrEYIrZJtz//JyZ5r6hLEwIL5Dp/jfjlHIlCNAT4J2evVEmaMb4
OkLwcRKwHXAbcKK735/kcYmIiMjQVEIbEYCvA9cBfwDeA6YqCBEREEal8FVEiIiIIIItWpEtqIiIiIISJVSICIiIIKJUSAiIiIiiiVF
IiIolRICIiIiKJqZTuuyVhZocBL7j7j7IjM7BTGoeA74nru/X8RWNPWuHuHme0NfBqY6e6ee6/e/L2KeU4EfuXt71wvIW4CJ3P6dYYYesSs
5ntDJwGfCj6/3DA3f1PdRcH+xGPsZCcyYEg3iWXAOFGugc9wM+Stsx+2cmNo9e9eB7F8Jb1tQu+3x19/6U4NRXZuhSutt9bLfMcnU4nHNVHx5c
AR1B3wM3AgcPuf3D3E3/uF+71HAnYJWEbXAesh+eAWCMwxdL2LEr8hWzXsMlLzmYjK3TXHHjH5s9RmsPZ7Zy07C9DHffqVDSZ5Zi
Tpf6yNLNRwLeBW4FZwO3OOzPuBLJjHu/u8Ksu3HpgMPOTub5nzZJbF8zyzWtAxmdiWDwMLI3k3RLAACAQSUVVRUE1ESwe3Qc5wBfcPdfFz
4juV8J930c95e7fLVS+WcdwC3CWu6/MWt4tVvAE+AF4DHiUMNjkX36koyUHqHQ7e51hco36xj+Afh
v4HjC+T5P+HoBr7o7vcUKd//AK4l/B61AU8DVP2P2AW5094KUP8CNWP8xsP8IXXFXCEcECEEDsVfX7Xu/
riZ/RCY5+57mNkNHVJ2OhbAM8Hvv73hzrtBO+aarpm7HXuxB+HH4WB5HuKuLnfBz4CnAVeB84+OBPi4B1pjFgfO/cve/FDGvuGsGsJXx63
Al8kfKhPBj4XpR8kR8ZleGeUxs5cL67X2NmAxDDcwBDAhHCdBOZ4KOe8GMxD3iBcD/vA+
wE3FfAPDGzSYTSnhgrSFLKuRdpuwVHHEpvVMt5s1wGjCA4eXXCYpRCrZldAvFpCvCIHA
wiTsO4AfIFWxXUlECE8FH/J3Z80s2sIJTCHRwH3nYTrAXdHr4es4B58b+X0LgUDB
RVU8tgJm9Cezn7u8zD8ZVPQ8tf8WwMJ2ekr7FmuVBQ44FPh/l1wbcRQhKXihSnhB+gA92dzezacBv8MT8N
NVbfn+K3C0u79ozZucCj7j7NDHd+/JhexHw/DDyYYnLHDA1exDZAQQkpv0EH48cuYAqyP/d1NeJA4
t8D5xn2K8L3xlpkdA/wmenPZtmWbfQGJ3wx7HXh25+PzRz/ZAuPYV8+mSvF+5+I/CUwF+yfL/lPCd3+SvF
IQuk8hElgNytivol8bygQ6fESIeJdBmwJ3BcVQ6/jeEr+r4vvFQX6yTUSnaTcVg9dgDSbl
+sTN19LaFk62EzG0YISo4EppvZPOCXxFLCq3PKPO+LCJxxAoG6UpipG+Z9Ejz8ALC0vRk
LtqsJC+BHwkE4QAuHt3VKpa0C9ud38RGA9gZk8Ax7j78kLmyjuD0YERUBXQQocqC6LiyS2cKaStgaY7layjuD/MiYI+oyuIjhNJc

6CkFKxYHjo8CvO0Jv0fDCdOdFDTgy5LI94YCkR6nESLrHYBvuvubZvZTQtHjMUXMdyrwC8J78aC7P2dmVxJugH8rVqbxuk8z26j
KVvcJPAlcaWaT3f3t6Fh2JhTbP1ykPDeIPsyHEYoaPwMsAX5LFCCZ2Tfd/ZoCZvl74CZgFdAF/MbMDiEUYz9SwHzKJd+/EEoD2
ghByf1mthfh/X2qiPl+G7gtqkodBnzJzHYifIb+vYj5ziMEXK9nLf8cxX2QOCjX8uj+/oi7P12krO8nlCR2EOb9etDMPk+oRv7
vIuUJ8ADwczM7kzAjew0h8LwmOp5iuZL/3955h9lVVX34nQSkyygoBQRKHQ6MTxqQQQkSYldBEpiUiJ6Fmrvl9UQTxQQkSYldBEpUiOd0EFAKQFRu
1A8CAiKKwALFEMCPGnoICTDfH2vf3DMndyYhOfuemTvrfZ48mTn3zqw9M/gOf9/fus/Yqv+Uptm78QHqHpEPw1MjuGe3ug/8+5wXOMrP
HUvRta7yeMBe17BvhiCTM7GFghdLln+YqUi3YvULSbcAnCymCi4EzzOzfGU3Xlfv8YfreYyWNwzeUefHalR9lskkKI2+Jp93eT
WvYtNhRIWl3PAJWpSOyI/77XAzYxMxeT7n+e/DNJhd12f0h/vtbGdghdaAdgOf0d+vzK2cAM7ta0lP4a/kRPM1pwJoZb8rgufwr
U03X3/DIz6rAKvhrLQuS1sDrNZbF30PFIs6J+KEiB7sCe+Cvq3NTp9/HgDOA4zLZBNgFT4XcDjQKUyfhxZR75jJqZqdJugtvHrg
RbtGA+ua2V0Z7d4qaQHgo2bWaCgYgafjckYWa9k3BnXXjKTtmMY/qpn9ukK705yCyJAqaKzhHUAp1/sg3lGyr6TFgcfNLNdG1r(
/LN7ZAPAPM3s8s73X8dPcKOAWM3uvxXNWB75hZkeUHwumDUkzpzRY8VpX2sBXNrO/1rW2XEhaBtgBfz13447QeTkPEuk9+yxeQ
HglniJaGDgY2DClcXLYvQaPGD+W4/tPg/2P4gW74K3Zb/b1/IGKpMNa7UOStgZOMLOq649qZbBHRH5CT0dEwHg8pDoRD9fPiYek
NE8DbdIr22pdH0/qumrtwnkobiOeVPARcBwyR91Mxy5pivBI4zs17D5ak47t7eHp9WJF1Iz79psdWz+Dlm9oMZtdfHOrYF9sY37
pXxk+wLZparIBjgekmbmVkxnzyPpGPwaMnMOYymlMT38UjEzDR/1114B02237OZPYqfVNvJssC2KWT/IK5DdGaqKTgWTxfl4EvA
FE58O0g1ez/GO4OGACbpfDMrp8Vm1E7tbcPAbpI+ZmZ7pTUthac318TTYJXRH/arQe2ImNlSjY8lHY7Xg+zY0DqQNAceDqy6IKz
YDbMJfrPYk55taafQrJTOQS25T0mL4NoDH0v/rsM38S9J+oaZ/T2T6c2BozN97zLv0nzzzokX9I0G7sfDySsBa+G52CxI2hWvPz
oGD+l242m40yTNYmaHZzI9J3CLpA3MhZh2xG+Mr+Kv9Vycj7ca3gQ0Tsm93UBmiJRKbcUUG7iZrVu1/cR4mh0zhqeVb8RfY1Uem
sqcCVwu6Rz8QDWh+KCZZakDkrQm3gH1D/y9NBP+HvqxpK9btaJ1/aFteA3gjynt9RwwHE9fL5ehmaD2/WpQp2aKpND96uWQY/J
EHzCzOTPZfZakX1K6/nm8eHXBHHaTjZVo5j7fSUVJE3PmPiVdh0djdsGL3VbAIzDnAoub2TqZ7B6Cv7lPofUGmisF9jtcrfeo0v
G+a2dcz2X0c2M/MbpDraqxgZk/JFRrPN7NPZbI7K57+WhJ3CJYFjgJOKadsKrb7FrC5mf0op142CrcOn8andudJ8kn6LOzx74N0N
P/2wAH5grdt4gW9MDMsowNkXQ/nlL9Sen6ccBXzGyN1l85w3afBHYttQ03BPR+3eg+zGR7frwlfWXgOxkbCIo2a9mvBnVEpMRrw
eBcu5zTeD5jHZnp3W4eh7ypWUASMWxDxU+/7OkWSStlrHAb01gtdQy3LA7KYXuc+pbNBQuv9nisZwpsGILa5Hf4zfoXCzKlK9l(
fs47mMmtkESZvjYeRdgLUqPq32xuv4yTE7vUWTUv3CkDa11O4FXIJHgc7GiwxfwjsdshXY53I0poFlcUmDMhfgv4tctK1tWD0V(
iwPDE+1SI1UYxbFXmrar8IRaXIUcJ6ktejZHrYVsF1GuxcDF6cTe9HukWQUzKmx6n48Li5WzusKb/nMRRbp+GngcfwmMfnNLWkI
fpJ9OKPd+/CaickKjMnuvsBfqjTUIlXRSIlMAq6S9AjNDTRXquJI4JTU3vlkq2LkHMjFy/bCRcQWTNdexFsusxU9m9lzwOTooX
EwTLA65ZJhrtgayju0A8j1XjhBe45a7yeBlbDZceLfIG8B8V2tg2vQ+vmiXsLjzdGCuRyRGrZr8IRSZjZeZKeBnbCW9TAq9/Xy
3yi24+mrPp86doL+As9Z1HhSPx0fAAtqu4z2j0bOEfS/njBmSStjddvnJ/LqJmNyfW9p8IewB9SlOBh/GdeGY+EZQlzFuzeKGkI
3Kk8E79hzE71hYx3TOP1nHngg4CFSGJPjWhbw27GosJD8N/1z/C8+lC8ruBwSRNzFganCMz38GLko/AC8FZigVXarKvGawRwVk
VN6K1X8R/9wdnsgltbBvuTRumzdSyX0WNSAvSG3xooX+7HTa7SI6Imb3UBnsTcOGjx9KJ9hdmdqOkLfGC3VxV90jaA3eAPpkuvY
z55gZn3moD+knQ+ABc3sxanktnPeqBq53q3wan9wB/dyyzQArmB3NjycvTR+6HgcuMTM3spptw6SM9srZnZ7JrvPAruZ2XWl65s
p5nZYpnsLocPrnwaD90vjXcBbg1slPHnraXGK9neHr/5L4XXeBlwkpnlFFJr2G5727CkJZmyC6yxgGwFyXXsVxERSfQVYgV+bh
mntkpaEU+RDC2sZVZgRTPbta+vnQHqqrrHXLl0pKQ5gZkyvsDXpdnxlCslMFXM7CVJN+O6D0PxMHZWJyTZfQf4ZW47Repqo23ce
M7+pJ4fcwsmdMF4APgrMX1J/D6glyMxNM/h6o55O+Hac/6BZ4+yEFdNV6Y2UV4KqittKttuGRzfzwKNI5mF1iRbHt0HftVOCJNe
x4jUTWUKskg5NNp7HHaBngU/gYezf9P6VM8ytwLEpOnEPXgx1Hj5/pVVxVmWUPf1iGL1KT794KuzrhCifs5MFSfPgm+fGuFM0FJ
hbPjRr0ypvlJKmdaZKt5nlqplpWxttkdStczo+EbYLT0OdIGl2vOMgVwHpaGB/ST9qRPPkc3b2o+JanBKr4LosZX6JywHkoq4aI
yRtih8UlybdIHEF6l9ltNnOtuEi+wEHmNkJmb5/S9q5XxUJR6TJzsAPSyHWh+TD6E4jX73Gznh72DnyibhfxR2BK/C+8VzUUnVf
l6efcssj8MjTEJoFurPgQw5zvRdOw1NQS5uZpbUsg+eYT8ZHbFfFtBZH5szHbo4PY8veRlviF/jfdiXcse4mzZ/BowfbZrI7HLg
LWE8uLNaFd9/NQusOrap4Gb/5l9Vbv4inTnJRS42XpJ2BE/G/5XE0D4pnSPqImeXSuDgJOL2XtuFf0FNzpEpmBa7K9L37op371W
EWlSV4h1PprD3h7CtUwulXQwPhG2Smn5I4CTU5htKPDVwimuXVX3tXj6eA57Jnwk/Ml4gd1ieHHy2hntbgx8rfGmBlfilM+1uY
kK39gpdD0FKb890XqqneaibW20JbbAT2z/KKQMHpW0C3BzLqOpxmopmrU4E/Bizty1OMfh3RzHkN7LqYZib9wBy4KZHSnpNTxlI
ve1jm5xiuXXXymz26liOnVkv6JF6vmckTqahu+DNhd0n45ywJa0Lb9qkg4Ik3qCrE+B3wGGIuHGlcGLsXDnFWLEh2AnxBfwztmI
Q3EdLP/EjF9lpRl6e/KrCGmT0k6XvAY2Z2hqQn8AhQrhDrBDzKVOYDMurESJoZ7yTZDU/1IWkscKJVO124TC1ttLgi5PgW14eQe
8zs5fxk2TbSBHU/+Lv6fH46dzwqG7Ol G47a7yKLIDv0WVG4weKXNTVNjw3ftP/ToqUTyw8lrMNvpb9KhyRJnWFWM/HJZO3x4ey3
Zw2mK9Rfd+2Ab+T1GixOzV1zzRo5PJzzuaoy9OfhDtg4L+HlfA6mVvwCEkursPDx9tamnMjaRhez/D7jHZPwduwD8QjbUPxDfVw
SQuaWa5Tc11ttNcBR0v6fuOCpCXwUH6237N8gOUIvNh7VnrWwuSsxcHMrgeuz/X9e0P+R12e9PPmqvEq8Tdcz+mQ0vXtSK+1TN
TVNvwkvZcD5Nw3a9mvwhFJlEKsjfawPwKX5gyxmtkxqQ7lHTO7Ty6luwteJ1K1M7Alfvr/aPq8USfRXfo8J3V5+o2I1374zIjvS
DoZdzZbnaSr4kB8ps8TKaQNrsh4I76Z5eK7+BjvopbH39Lv/DLyhe+3z/R9p8YeeKHmODwK8iD+Or+JvL/nXwPz4rUT5WLNSm8Y
aipvTjGMrEwu5U1Jw/FajdfwNFyZXI7I/sCtqR7lPvx3sDqwIhl1j8zsouRo7QnsQ7Nt+Ac524Z7U+9tA7XsV6EjUiJ5f5Orss(
na3Yc8omLe5nZFJXukubM5Xyp7zkdOWdzLIOfHs/EbxoP4Cf3OYGjzeywPr68CvvLU9A/KOZgM9l7Dp+9Up5htCpwg5l9Iqf9up
D0GZq6KWaZx9VLegdY1cyypzWTFk43HuEqFnqXDxPdufQ8JD2PT7HOOfCtYavHPiRpabymq6gjcmbmmra2kfbkPc3sTTUn4pbJI
aWt+1VERBJ1tS0l23WMa3+6Fydka7zoLMvQrBo9/Qn473guvDj123hueZyZtco9V0ZqIV2dpg7BpyS9bGaVtkknvYMGJwEXSPox
P0Pn56PIspw9szavc/wCpm9spUWogrTVWktMizZvZB+hg8Bff30nOyDTUE/oUrjLaD3YDN8NfSXXgq91prgwBigdmAa9tka6ykI
zsGUkX4AenfdpkezJtbBvuavFxj1Qf+SPWbduvioQj0qSWtiXVN659V0nzmtleaR1L4XnANYFTM9ksCl6tir/+sgpeyScKnwws1
+LhB/DZK9mQK2DehDsCD+A/8xZ4rcbaFUfcnm1x7c8trp1NtV0GR+BjChof90bV4dcxNAuux0zFbmW1KfLJqw2uAH4t6Si8lfb9
PN7M6q7JrZ2cDZkuYGNsCdkuNTzdfVwNVm9nRV9nrhEjy9m/V9kxgKfF3Sn/GU3xmSWjpduRzNdrYNm9n2hU8fBUa1O9rT5v1qM
GINKmlbQlvAdvJfFz7McnuKEmv4oWsh2eyuwbwR0kfwzt3huOFm8uZWblCvEraJngl6et4gdUo/DT5GJ7XnhsvtvsBXhy8rpndU
7X9xGl4++hOjQ6S1NFyHl5QWuX8hloG+5XahjcDDsqdEkksgetpND5uF7e3uHZuL8+tfFptimSOAkZJmgXXHtoUOCClTq6pskZI
YcazgysIenbuPNXdLyqrvE6mZ4O8/29PC/n9Oy62oYPBn6b6Xv3RTv3q8mEI9KklrYl6hvX/m9JX8Rv1Nvg6pPteOG3U/DqUHw
WxYGl6+Pwm8ntksbhhZsbZFrD6rhg3eQ2VnNJ7BH4iaMybBoG+6WI1Er0HUGYEdYA2tKyW/p5e5sl1A1MlDTUzN7v5Tkf1m7lzs
0YmbvSvojHpF6C1dbPYhqp7MWC5676V1osdKIl5kdLulU/ODwH7xl9uW+v6py6mobHgX8NO0TY8xs4tS+oCLatl8VCUekSV1tlu
0c196oui9yOx4dGJ5SUY0USa4x0+0UvFoB+NFUnvNLMgpeAf+H1/6Ui70+Q0ZJbElr4PUgyzJlN9REvPUyB2fi7ejn4M5OsT28(
RFiado3fXViLZNknQNrrNR6cAySUPxFv9heJ3ZMLzQPWdd2VzJ5sY0pyn/Hk9hVOrkF9PDkrYDrjCzHn/XpCmyY5V2k+1XgVdT
K/ZYazEUM53Yc1FX2/D6+CF1B2hrG3wt+1U4Ik3qarNs57j2dWh9arm38HijAj+XI9JOwavZ8ehHX7yMS7zn4mxcAfMQeuoQ/J
x8YV3wnPYYvMjuStzZXRgP+WZrd8RnNYH/3K3IFU3YGf9Z98Bfzw0doNNwgcBbaKrqtprRMl1IWgS/8X8s/bsObzX9kqRvmNnf
+/r66bC1cfq3Fu7QX4unOu/O9V6ST2OdDf+dXgg80qJWY0W8xi1XfdkE4GRJxREN4HvmUngLdQ5qaRumvjb4WvarcEQSyfNeO7U
aSazcwez2z3kRR5aUhED8Wr4SuXiDaztav8ftNJOwWvuug9ZD/ZJnkr0U8A5sA36cZm+QJeAHdiRrvLAtsmfZwHgXfN7Ez5dNZj
d7JBWpNWxwBbG1mdxeu3Srph8CVZnZs0sC4mQodEeAMXJV3F1xboxtPdZ6L35SrbKN9Go9m3YUrPv8jXe/G6zYmP7HiyNPaeFFu
g95qNS6p0GaZX+JTla/Co8Un4qf0DXHNnCyY2WhJK+Ntw437wh34ay1bIamZ3Z6iTHMBb+bUsipRy34VjgieCimE/ObGw1MAC0q
an7H1j3ST2N7M2jquPdleEb9hNW7+jSFwK5nZrpnMbp/p+/bG1pL6CpHPk9O4uXrs4fI5PwsAE3KG7AuMp1lIaHia6kb8JpJthF
oS+Db6BvAI+0oZMDXA9mUovrH9D8G78JfKRiu2sCq5nZ+2rOuJmUCs8fqtgW+Pq/mv71RWUOoZldKekv+P7wFB4RGIJHG99NT3v
bXOo+F2sBXzezeySth2vh3C3pQDxFdXVVhtJreEzjnpAKr9vSNixpQTxquRnexdm4/jQe2RyRs422rv1q0DsiciXTgyUtb2b/x
TfrOQpPeUbSclXnlQssxNRP7ZUj6VC8I+d5vAXyWXwmSTeQbVaFmd3ex5oWqtjcWKZtA6n0RinpTmBj6zmDY1YzyzkVtcytwLGS
Cn0Q6XdB6wEa7aWyl9tUlLegDY18zuqtpugd/S1E25H79proJHJa6RNAeefq16btR4/P3zROm6qDinXnOR7FhgV3zsxL2F6w8D5
nZmZntd9GsLXsU11u6G785P4Cn5qriXxTmcAGkyNoVGe8DSFoSj7a8DZzDlF1+2wL/I2mNKg/H/WG/GtSOiHy+y8/wvvhizvOzq
avkk7hU9J74mOscXIzXiFxC6+K+XKfXnfHq6HPkst9fxW9QV9B7VfwMk/RKRuCRmEautxGJmZ8KX5NmtnhV3+tD8mWmPHm/mMS
ZnmrTGvbCQ+Vb4HnfHfHX+Af4670yprFN+hZJ62Rsk94Drwe5iebvfiJe07AfPrvp81Qfxj8bOEfS/vjrWame4Gi8VX3Ak4pxr8
OjPxfhehqv4jVHqwAnpBq3jVoVk1bEQ8D3gKNw5+dreB3U4pnslTkZd+6zOSJ4DdPDeFfhu6XHrkpRthvwbsAq04u171eD2hHBN
2DzOzS0vVuADN7TtLP8c6LXI7I1vjNYZteHs/liMyHR3/A3+Srm9mlkg7GZ5Hksnsu/rprFA7uj7fB7YTnooMKMLPnKNQnSFoHT

m8TsnZrYDa26TN7B1gp1QHshTeQvyvQm79mvSvartHpuL2s/CCzt/jJ+mT8Hx7J7A3/tpZtkVdxEWptfO29LyTMq3hQOAGSW/j
e9P+kh7BO0tGZbLZbtYEvtXCCQHAzCamzsfL27us/Ax2R2RZfLBdkRfpqYPwJzJuKDWe2p/Di73G4pLFK+PdBW+QSd49sSqwhpk
9JOl7wGNmdoakJ/CT+v9mtD1okPQ+sJCZvQiQTqqPSFoM12SYs0JztbdJpxz65cDlZpZN76AVZjYSGJmKC4e2qQaonewAHNhbca
Pv+eGdFFkck1YMsjqcMXpa0Cl5HMY7OuTHPQ2t15CJj8UhURzHYHZEJlEJSZvaZ0nNmxYWCspPa5L6NF49el7nI73xc72F7/KR
4s6T/4iHPhzPanYR3F4AXUa6EhzxvwSMkwXSSHLuGAnAXHs4tF3AuTLMYuyr6Q5v0PsBWwJ3ygX9X4E5J7qF3M+Fh8j+Y2VhJR
0raAq9b2LOUdx+oLEGzlbM3HsQPNtkws9clLZI0cm7GI7pPpQLLTmAIUxcDfJ+8Apu1MNgdkb8DW+JzXnpjE3pvV5suUuHc8Xha
BrxOZCRefDU7/oIcIWl96znGvTLM7Ji0Yb9jZvelot1d8DqRnJMdR+Nh1f3wzfo7kk7G8/fjM9ptN8VunS78vbZ5ap+dTMU1QN
fgN40uvMtgNM0ZMOApx7eosMMgUXubtJn9DvidfGDXBvj7+m5Jz+An5lHTojw7HZyUbN0raSW8Pf2QtIbT8QLDgc5rwKfou6h7F
rW2VWKpHnxwtS18dfRMFxyfAlJ38pwaCs78bMBl8mnLTeoWtIeYBFJfYkNLlCxvQZ17FeTGeyOyGl4VOBJM7uy/KCkjfBWqo0rt
nsyLle8G37z3RO/afwJD4N24/oEh1OtDkEPrDA90szOpz3FdcOB6/EBYefgP/s4PFWQqw6n3bTq1nkB2L3Fcyt7Y6eK/iMAUgH
y5WUFzIzU2ibdwMzGS7oKd6hfwot09wYOkXQ38GMzK3e4zAhb49O5/ybpAOAmMxsh6Qb8YNEJ3AAcKumbraIPciXoQ8k7lfc0fI
+cD3gG3yO3p3mIq3KPbiXm2OpAmCMSU+mhdxqpZb8qMqgdETO7WtKRuKd7CF6fMA4XclkDb0U8yMxaTTCdETbFK8zvA5A0Gq9NC
MJqVrJ5BHh2AykrbFN+nP4jUiewAvmNmxmewtibfGfRbP9Y5PkZElgAfMrNVMhwFHjXU/xTVcJOkz8unOn8VbL9f3h6zqOpxa2c
LpM6OdfHU5qb43nYVLnh1G+7onoM7werl20wPswMvpBTN+nhEBDyq2Skpg8PwtudbU2Hq/XjXzIJ418zPcFXZnNGfbwJrm9lrBk
WV9L+Uem+Yfkmnk+NuoZWLl6H3SKD2hEBMLOjJF2Le9er44PmXgXuBLYzs3/08eXTy3z45t1Yw8uSxtMztPkmvsllId2gDsXTUi
wTfM+4DRJs1T5ZpTUhes57A6sm9JNjTTMrsC3cNn3ezso31sr8lH1N+LtrN/EQ8tLAmdJ2ialMiqhP2xk+AluNtzR2AmPTBS7D9
6Q9Ds8Elklo4Ff4EXes+OaJcvjaZlcc3Xaipn9X6rLGIl3BRVTbO/jqb6NzSxbaiYxW4tr89NayG7AkSl1OCAY9I4IQHI29m2z2
Ik0G7ae4LaCx/1fEPqT8fMRkl6FU/RHF6hrT3xEPYm5ZoXM9swpcAuwqMluYWRBgvH49G8kZLexPPZP0kFyUcAlTki/YS9gGvM7
O3enmA+XbrqCdM/xG/QKwM7mNkLKUUzEU+9dgSpY2ZTSZ/Af9aP4dHjB9rggIBLCpwiaZf0+ZyS1sXbpq/o/cuCgUA4IvWxRtJW
AD9hDAVWk/SpdO3jme0vigtPlRmTwfbOwB5mdkOrB83s+rR570U4IlWxHK2nRv8Bd1IGPJIWLXx6F/BxSS1fu1UqUbb4vpuULu
/XqZG9pLZ541SfWD0H4NHbB/BOx7/hh7nzcMG6YAATjkh9XNXiWllYLSf34RNZD2tcSEVnUWdG4gAAHfxJREFU+1K9DPZiTL397
3b8ZBlUw9N4GqKsjPgtXEekExgzjc/rJlPLYxKYalXACYCZ5ZpiPWhIUZhXzGxfST/D61K+gMsvXJjE7HLZnrlRt9fisZXN7K+
5bA8mwhGpgZpnRjTYA5eW3wDXSjkTb4mbneons74AfJq+ixU/SYYZKP0F+TCrmSm1sOY6qeMqphcl4aeZge/Lh3l9B5fK7gRaFf
c9gjtb2QZVlliHno7ITPi6Gu2mwXSSBOIuw1uhl8WFF7fCxfGexevM9pH0FTObmhDY9HK9pM2Kzk5qJT4GT8vNnMOopAuAvcqzk
8xsy0x278SVaq+0vEMMe9DV3d2REcRgGpA0G/A/+HjrobjA2CVW8chpSSfieeVvmNnEFo/PjCvcPmVmVc5QqB35HJbzaK1W221m
cSJJK2Ah62Lf9+TG91aFdpZdOrPcjI6Xo21vAm0c6ZPqzV04foi3WbWlqmtnUjaN76Op3bvxYtV/wv8E1jLfMrxWcBcZpalY0fS
/+KHhw1Sx86OwLF4Q8NwM/tDhba+iHe4deEzkvbCRzIUWRpvQZ+rKrulNeyLO3sr4d1mlwNX5VYLjohIQtKFNE82jVNro4B0Ev4
uMrMHqlheZUj6UFgezP7ZRvMHYWnex6QdDre/vc6fmpcBZd2nxtPFXUaI/FNdEPyDsyaAjN7mPZEP8aUPu9NvCxbiqQ/YWbdkk
YCf6VN4+M7lC3xAuB7ACR9A5gLGFlIl/wK1znJxXp4hOCu5OQui+9np/SWspkB3iHpACX2p2dTQzcuUHhAxXYnY2YnAiemURBb4
KXZ0r6E+6UXGtmlQtPhiPS5C28yv0+vCVvCK72+WVcqGcR4CeStjKz62tbZXUsxNTVMCvBzF6VtDo+tfMEes45eR1/gR9ubRw73
UYWAb5pZm2ty5B0G1N3rK80syoKD4uzLzbBtWn2xAsLJ+Knq1PwgYeDhW9R/XDBwcYn8E66Buvhe1ZxPtjzwBy5FmBmEyRtjrdj
4JHYnLNw1oAWDrZvA3Y3MxezWSrT5JS7fFJHLAhCrg+MFHSZcBPq+yWCkekyWeAo8zssOJF+TTa1c1sA0k/fX3OsERuRivEbkF
P9H22DSrlvI1s1fw6ag/xn/X8+A1If82s6nNVxjI3IVP1Wx3gehduLjWb/GITBfuWG+NF0W/g6sKDzezC2bEkJk93/g4FRNuUUr
//K+knfEuno5yRiS1+rvOhbe3RjfHjDF5MGdKd20A3GtmxblGq1FxPVByAoo0InyTcOn3R9LnVUu8X41Pjn4GWBw/DLcdSZ/Fh
QG/DXwObyTYA2+wmA+vJ/wDPsC0EsIRabI2npMr81tc+At80FKndHZsjZ8utunl8SxSvklk6tEc37ufcgcuIrYB8CQeIYDmRpar
+KreA77jOJFSXcB3zOzr0j6M150N0OOSInZaV3ANw8Vp2UK6dSuwv+z4HOainVO3WaWa37SEaXPu/G/8QNm9mQmm4OFi4FTk+r
1V0lR6caDkpbDxRgvqdhub3LudxQ+brzmquRVXEr/brzT8H96G5uQa+aLpIdx5+MvuLbTb8ysOCTzVUnn4HVvlRGOSJN/4x5geQ
DepjS7PT6LS7EPePqJGuZg4Ot4Tcwn6DmwqrGR5XJEVsLVgsvcQdOZfhDf8KrkYuDidPP4G/5zfgE4EpdYr5KuFh9f1sdzKsfMI
oLJgyyXxE+x/zazN3LaHSQcjdeOXYgfmg41s8tg8giMffAoW6Uzqsqq0ql9eB4zs/T51sAdxUhgReyG7wdfTZ+Xa0SKZHFEgN/
gKaF/9/Gcm3AdqsqIrplE6m64Dg9j30+zRmR1vGjqWXxU/Ulm1te03gGHpPlxJ2wocJ1VP8kyaDOSGq/jPRviWoVujrXMbGX5uF
pjzWxYhXZnwjfTHXH5bfD27ZHJVkdtOJI+gku870bzYPcenv7auVWXWDDjJBn9mXLreCT11mvxff+wdO0evHtlo1z1IvKhlau0s
4W2ZH9+XNahLXID4YgUSDoLPwSWx/OB/wTONbOxkpYFljSznBMms5JObcfjaRloTq68Gw+pD0n/1i9LsQfTj6TP4DN1cg+fK9r8
PH5yeQPv3hgCrIifMDfCb5o3462AlXVOSdoGnyI9Ds8nA7zcaQ5IA0mn4rULu+NF7kOBL+Lvq6vNLOpEBjCS/gaMMrMRpes/wS
MHldVJtLDdhRfoFtvvb87QrVO0+VX8vrBgi4ezyQ2EIzKIkHQuHiY/FhcD2hNYAb9x7ICnCs4AhpnZOnWts5MoDZ/bEFgGd3b3E
SodPtfC9kdxp7PoWI8ys7clLY6Hm/9Wsc3XgNUaYexOR9JLwFZmdlvp+jr477rVhh4MECS9DSxfTlXIJ4n/w8xaDeKrwu6n8EiN
cAdkKH6QGQusZ2bPZbL7ON7t1hjk2INcg/miRiSRFOv2xSuBZ8JDUpPbHiuujq6LTfFw4n0AkkbjNS/nN7zslHt8qL4ldhy1DZ8
zs9clXY5vLI8CszGwuXaUHARpO9KOtp6Tr/tVIbQum7sJbx7JhjYGO7Ml9PxG9GztbhqzsBTmus1WnjTHKVLgNOALTLZXQyPiI
yy8ckSa/xp2QS5lSeKpTwkbzUWh1M7OXJY3HN80Gb+JpmqAaahk+J2lWXPtge9yhHgacIGl24DsZ9QkWAA4BfirpRXq2hXebWSt
9oHMn4HjJG3bUJ9Mh5rjcKcsGNgcjMu8r4cXd3fhUeQ1yecMgBesfrH4PjWzVyQdCGRL6eKtum2XGwhHpMl6eBFf1QPf+hvlKu
yGyFWQh7qGz/0CV4FcCbgH/xv/FG/JGwlkkcTG2/oqbe3rDUnbMY2v3Vztjnjnxq3Ac5KeSNeG4X/vjTPZDNqEmd0kaUW8+HoZ
vDX7IbwQOecYgXG4Fk2ZeWlKAFRCaXDjM8C5qXnj3zTvF1nlBsIRafJf2qQ0WjNrSGoIAnXhucfVUk4SoOUY9WC6qWv43BbApmk
nwT4qaRe8SDULjXbWVqSZQlXyE3o6IsJrn/6Fb9bDcBXf0eTTxXk2FbKvj4tRTaBZVDgY9pOOx8z+SQupfvUxmbcCRuEOwe40J5
d/EY9yXlGxrfLgxnuBT+GDSBtklRsIR6TJfrim/mH0FJ4C8g/raiNXtbh2adtXMUgws6slPYW/vh7BJdANWLPq4XMl5sRvymWGk
PF9L58y/BM8GjOEZp3VrPiNet6qbJnZUgW7h+NCTDua2Wvp2hy4dkmlaahe1FSnmK8jqRNTUYOKFq9n8L9z5a/nEofh2kM3Fey
+h0cbK+3EMrO1y9ckzWpmE9LHi+WWdAhHpEmjaLBVPr8jhnWZWS2SwYMZSYu0Gj4naVZJ38+YMrgOOFrS5EGCKRIzktav8ar4JS
sdRVe/H0iLtO9IfDdjHaH46MYXmtcSN1BR+HFuntUaKuspgpwFq7AXKy3ipTnwKeW13NyAraXNByP7L2DC+W9ncsmQOqmuxJPNx
6YLt+fDlNbmtmzOeyGI9IkTi5BDp6WdCV+Ui/Kjs+D12vkckT2wDfRcfiJ6kHgo/gJq8qbcpm1gK+b2T2pwO8GM7s7Fdl9E5+n
kYPXcAHCx0rX18QHo1VGq/RTmrb7u8x1A0H7afvrOSm5vmJm75kPDJ0EfA1YUdLvMjsjZ+O1aycWri2drp9JprqncEQSGdsZg+
DTwIOSvm1mf89lRNIZwKVmdk+KDGyRxNSWxt/rZmblG3XVdOHDysBbhlfGBfOuxCMTO2eyexRwnqS16Cktvtxwz
xEQUb4KmgxyVtjx8onsVTrYdL+kquyATwJWAlM5vckp66dX6KH2ayMKgdEUkfAAua2Yvp497IpigXDAo2xcXjRkvaMymZ5gjbDw
PQMcDlwmZk9gle/t4uH8DTUUcDD+EluJD5NNBtmdp6kp4GdcPVa8Jqc9XIq2AYdTztfz0fgh5Y1gSdSjdOpeLHqWmY2SdJZeGt4
3V/BOu7JGylJAywF8VTCoHRFgXZqFbH0JlkWuN5gR3jOzg5KA3EVJbfXgqo2Y2dckzYd3zGwF7C/pUfyUNapNBdcHAjckRcpfpz
U8gg/JGpXTsJn9CVcJDoKqaOfreUtgBzO7B0DSN3BRvJGF7pxfATdUbLfIaXhkcWmaEZAV8RqsE3IZHdSOiJndXvh0XKuaS5
L0SzBBmdq2kVfHCtz+RwcFNQ7LOAc6RtABNp+SoNAjvMny09ytV2072704Fb7MmwbxVgM3wWpXLc9iEycPnvg802qQbHSwN/YM
fVGirrF/She+lmycRt8lkLEYO2kCbX8+foGckYj1cUuKPhWvPA3NUbHcyZnZScrp2Bg7AR0M8CextZpfksjuoHZESd0ja0Mzubl
9D1cGOpdYO/aVhYMZO7E38wAmNm/JK2OOwtL5zSc8rxnAWelAriN8bTFKcAsGe2+TgrjppkYp+eyVeB83Om6iaYy8hQttRVxBFN
6kS/gg+/KhCMywGnj6/k5vCNnbBp4twFwr5mNKzxnNQrq2FUjaX/gcjM7J5eNVoQj0uRQ4CZJW+Hqcmfgp6tfACP6+sIg6I1Gj3
6SVl8SbwP/t5l9j7yCZpORtDJ+k94EDylfWfH3nyZdDfJKvG8ObJbSM1kxs8Vz2wj6B+m9cxo+/qMsyFd17eDFwKmSDsEl3hfBN
Uwaa1kOvxdli0zgAozZ5l/1RjgiiTSU7P9w1bpZ8MmHS3eQkFlQAyll8AtgN5rvt/ckXYrLRFcq11yw+yXc+dgMWAiPFBwJXGdm
71Rsrj/oarxOs7shCKril/hra0umnEFWNUcDcwMX4imZQ83sMpg8jHQfXAPo6IxruAyfEzUCGJNrfyrT1d09eOswJS3a4vJaeNj
R6xMMhCaYHSafiIdbdcanxobhU80jgajOrTCUx6RxsjnfpLIDXNV0GXJVxyF1va3kTWKFduhpJun4LvDvpSTN7rx12g85G0jvA8
M3rWB6Yycz+mtnOGDxq2ops3aODPSIypo/HTk7/oEOUVYNa+B9gKzMrTmL9Q9rgRlGtXPOfgPvxCMxvzOy/FX7v/s5BeOTnnwCN
+TqJaL8PppeH8VquWh2RnPpDJbZvk50eDHZHJNRUg9wMAV5scf0lvDWvSoaZWnDwbn/f7Cwfd0LCDqSXwHnS7qQ1jPIOqoYudRJ2gN
C+WyO6gdkWlRU005/pXoO3oSBL3xZ+A4Sdum6nskzYuLEt3W51d+SAaxE1LbBhp0PAfiiqZb9fJ4RzkikpbCC2KLQyu78LrJ+c
nkMwxqR6SIpDXwArtlaf7yG0zEpy0GwYdlH3yA1HOSnkjXhgFPkWluQ7vpD7oadW2gQWczCDukzsXfK8fjpQn7A4vhisVr5zIak
/GoxwF4e+P3gYVxBcwN61tWMMB5FVgOH5C1NDABeBy42cz6GiswkOgPuhq1bKBB51IcPpc+XwlX4H4JH3CYdRJuTawKrGFmDyUc
rcfM7Ix0iPoxkGVcQjgiTZYFtjWzxyQ9CLxrzmemE92xwPr1Li8YoDwDwKbGpm1wHX1b2YHPSTU2MtG2jQefST4XN1MQmfZA1geFr

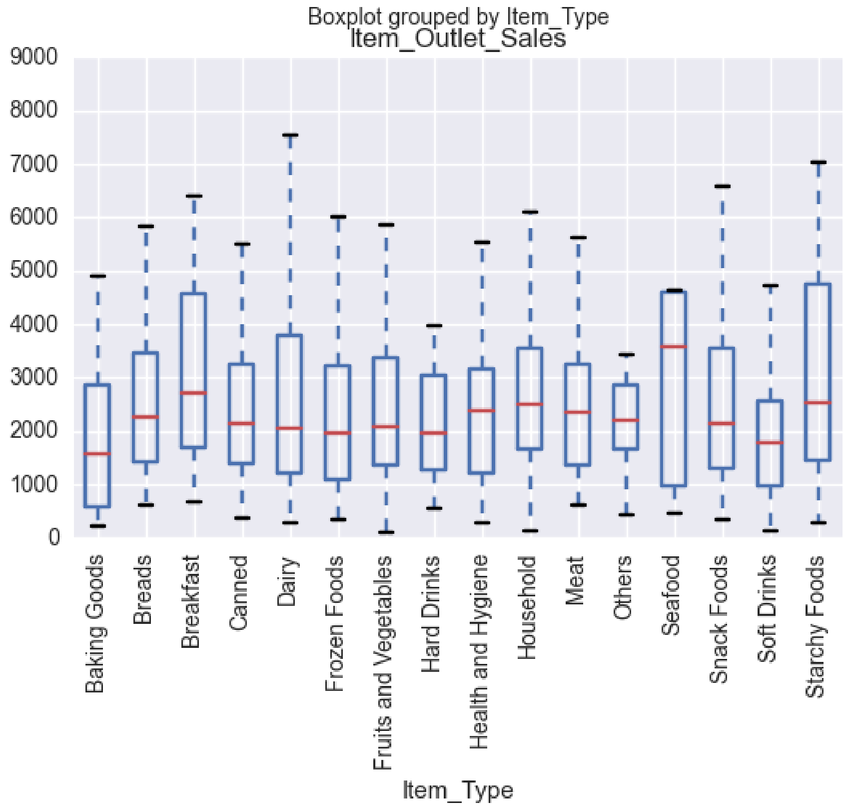
tNJs3KicckSbjgffTxwasANyIdyF0VB4waCvvk1HFtIikcs1JS1ExADPra7bSQKSWDTToSPrD8Lm6GI3P09kPny78HUknA5/H7
5FZCEekya3AsZL2AO4Bhks6D9gIn0gYBNPD74GbJV2Pp/4mpOuNGSg/r9BWcSbFJ/DUyJW4Mz0Jvzlvg0u8dxq1bKBBR9Ifhs/
VxXDgenxi9zm4Ls84YE4yCqmFI9JkL1w6dwvgbGBHPBf4AR7aDYLp4XP4FMuFcZ2LBl14dKIyR8TMjmt8nKIjPzazc4vPkXQn7
q7/YRaNtCgI6l9+FxdmNmjkj6LD/kbn4Z0ro0Phb0nl91wRBJpmNE6jc8lrY3nB183s2fqWlcwsGnMmqmBL+ATNMvcjxfPdhR1
aBBR1L78Lm6kPQUsGpjOreZvQXcIGlhSS+a2QI57IYj0gJJ8wPfxtVUO7LAMMhHijpsbGavFa7NlmHGS188iM+M2KVhV9Lc+Ly
Zu/v8ygFK6kIanz5+i84MnQf56Q/D59qGpC1xZwtgceB0SRNKT1scyDY2YVA7IqkI6Xhg63TpYryN925gdlyPYISk9c3sjnpWG
gx8pHTtRUltm72CR0P+ADwv6Uk8FTQMP8VFB1gQ9E5/GD7XTu7EHZFGYXtZR6sbeAQXd8vCoHZE8Gr6L+CTUcfjeeXR+MyOHfA
/wBnA4RTSNkHQ30lt6Evh+e1l0uVHgFsKBXdBEJRI74990r8yvwYuyz18rp2Y2Yv4/a4x9O74dmukDHZHZFNgIzO7D0DSaHwuy
NzTp5wA/Vt8QgmD7M7F385Pb7utcSBJ1AG4fPtZ0k4HZUQcBtZfwAnl3AbbA7IvNRKDgys5cljcd/8Q3exNM0QTBgSJvIabjQ1
ylhztyGq2kuYCl8HEMPfRTzOzOWhYVBP2cgoDbhnj0tO0CboPdEYGmiFmDbqaUqw6CD8vWkl5PH7d99gq+ibyOayK8mclGv0HS
sAF9C4eN6SNywmCgURDwO3L1CTgFo4IrCGp0ZbVhXfKrCbpU+nax+tZVjCAGcuU+eV2z15ZCljezJ7M9P37G8fhheZHmdkbdS8
AYQtQu4hSMCV7W4dmnbVxF0DP1k9srD+JC9weKIzAecGU5IEHxoahdwG9SOiJlFuDboVH4FnC/pQtwZmVh8MGNKqC6ux1WRT6x
IUEwwKhdwG1QOyJB0MEciBeZbdXL4wPeEUlOVqOea2bgeEmb4TLvHxSe2m1mP2j3+oJggFC7gFs4IkHQgfST9FBuip0xb9B0rr
eU4QBFNSu4BbV3d3NIgEQSeSRhUMwwuwwW/KswArmdmI2haWAUlrAfeUxdokzQqsb2ZX17OyIBi4SFoemCm3gFtERIKgA5G0E3
2qIvAfci4daO4nb8KK7l0rXl8E1EmZr+4qCYIDTLgG3cESCoDM5GDg2/fsPsDowJ17EemyN66oMSbvhzlaDFyS1eurN7VlREAT
zgiQdCZfBK4yMzelfRXYDUzu1LScFxv48Z6l1cJZwH/xFNOt+JdM68WHu8G3gY6VpY7CDqBcESCoDN5EVgAGAMYsBJwJfBfXOx
Nm3cAdAJI+DTxjZh/0/VVBEPQ3olg1CDoQSScB3wJ+gAsRXYwru26MF6suX+PyKqHQvttFs1OmPL68i2jfDYJ+TQh6BUFnchAw
pjfzG4GzgfOxmtFdqtzYRXSRdPx6ML3s67CvyGl5wRB0A+JiEgQBEEQBLURNSJBEAx4JA0BNsHbdYu6KbMCK5rZ+nWtLQiCvgl
JAiCTuA0YEfgIXwuxt3AksDHgONrXFcQBFMhakSCIOgEtga2NbM18CF/uwGLAr8FXq9zYUEQ9E04IkEwSJA0f5qu2YnMDdyfPv
sKqZvYeLt+1R26qCIJgq4YgEQQciaWFJV0haUdKsku4EXgCekvS5uteXgf8AK6ePHwW+kD7uAj5ey4qCIJgmokYkCDqTs4C5gH
9sDngC8C2+LKqmvXtbBMnABcKukHwBXAA5LeA74E3FPryoIg6JOIiARBZ7IusKuZjQU2A641s/uAk2hGCzoGMzsfF3B70swexX
/mhfB0zXZ1ri0Igr6JiEgQdCYTgNkkzYtHP76brn+aDi3eNLOG3Pu8wC3AzSH5HgT9n3BEgqAzuQZPUbwDvAb8XtJWwKn4zJmOJ
IHAzsDcwLDAN+LuktYE8ze7fO9QVB0DuRmgmCzmRXXNL9dmAdM3sH19Q4A9inxnXl4hC8/mUHPBrUDfwSWA+vHwmCoJ8SjkgQd
YHA+ea2d6pZgIzOxsX/hpR68rysAOws5ldD3wAYGa34vUhW9W5sCAI+iZSM0HQIUhaClgAb1k9HPi7pHGlp30j5bs297VZWcB4
8trr8GzNnmtQRB8CEIRyQIOoeF8SLNBle1eM7bwMntWU5b+TOwv6SdGxckzQ0cA9xW26qCIJgqMX03CDoQSWOAVczs5ZqX0hYkI
7XoviAmaPp4+fBjY2s//UuLwgCPogHJEgCDqCJF+/LrA0Hu014I/RwhsE/ZtwRIKgQ5D0HzwK8kr6uDe6zJ6JGJ
6hyPwGpDGxx1NL85WN16s2+NaOF5B0H+JiEgQBAMSSdu3uHwWcCjwUuFat5n9qi2LCoLgQxOOSBB0IJJuY8rowOQ3u5mt2/ZFt
QFJbwIrmNlTda8lCIJpI1IzQdCZ3FH6fEZgQmAx6G1w3NHtX04QBEFrwhEJgg7EzA5gEzA5vdT2lM7YAjm/neoIgCHojJN6DYHBxJz5/JQi
CoF8QEZEg6EAkLdri8tzA/sCY9q4mD5K2o1D3gtfDzARsNvFsQBNNOOCJB0JmM6eX6MM6eX6M8CObVxHTo6gpyMC8AKwe
iMSBP2U6JoJgg5E0nLAW4VL3cBE4IVQGg2CoD8REZEg6ExuADuAYzs4fqXkgQBEFfRLFqEHQmHwCz1L2IIAiCqRERkSDoTH4P3Cz
ZEK63oUrjf68roUFQRAUCUckCDqTzwEPAgsDHqL4hi1SAIgiAIaiNqRIKgQ5B0p6R5Stdmq2s9QRAE00I4Ik
XwZ+Ejp2ouSlqhjMUEQBNNCOCJBEARBENRGOCJBEARBENRGOCJBEARBENRGtO8GQWextaTX08cxBC4Ign5PtO8GQYcgaQxTDoH
M/t0G5YUBEEwVcIRCYIgCIKgNqJGJAiCIAiC2ghHJAiCIAiC2ghHJAiCIAiC2ghHJAiCIAiC2ghHJAiCIAiC2ghHJAiCIAiC2g
IpkDSRcBiZraOpI8Dm5jZBTWu53bgK308ZYzxXC/IBiAhCMSBEFvNESGTgAWB2pzRIDNgJnTx4sCfwE2B+5J196vY1FBEMw44
gEQTA1uupegJm92vhY0uzpw3Fm9mIvXxIEwQAhHJEgCFrRDXRJuhD4PoCk981sqKQuYH9gF+ATwBPA8WZ2WXre2sAtwJbACGAR
DSwHXAA8D1gInCqmR1TxWIlbQJcBXzazMYWro8G7gTOAp5KP8tBwBLAw8C+ZnZP4fk7pDUuBowGrGmllIUAdBJqJYNQiCVjR
OwJ/AZPgSyUHjsad0J2B5YDTgXAcD2wDrAivi
MXT838EHA8cBiwDHII7LcdVtMYgCFoQjkgQBL1iZm8CE4BJZvaipDmAvYG9zexaM3vAZxAvYG9zexGM/uPmV0EnIJHEor8zMz+amb3An8G3jKzA83s
X8Cx6TnLVrTO94GLKTgieAmLL2b2eOHasWZ2hZn1hmjDg8tQ1Bk10zjG7KETmPH2nPHPQcB7Y4P0cgFBMHy+OzCMyOudTiXXbtn43k3BwdqWQuJPEnaBVt7cxbuAMUvSc+WkW/Tm4nF/fCspCIUF1IkgpGA8LQCIDhfPdE
+E/ARSbMUrv2r8PF44D+NT8xsgiSA4vNnlAuAfSWtCjwAbAuUUz+3FdbwnqQHgOUkzQd8EjhO0tGa/w0YBWuNQiCRDgiQR
BMC40aiUYU9dt4aqNBo6B1YuHapF6+RxbM7DFJ9+FRkTmABYBRpaeV1zQT3nHT+Ln2xutbinQBYwmCIAuRmgmCYGp003Q0Hgfew
VGnmr8A9YH9ptKUWc7Cj4vADbFC2WvNrM3So9/ofFBSrd8Hvhr6r55CfhM6edaGa+Jqb1zKAg6lYiIBEHQG42b71vAwnJ
UKTd/Au2HWxIs8p9YB046b+eXAycAOuPZImaMlvYB3xBwMzAacmx4bkR4fC/wBUEFRARESCIGhFN80IkIxkXA7
MA/JS0ADMeLU48EHsULOn9mZkeWvr6371fV+qYgFddeA7wC3NziKWfgAm34AwsCa5nZC+lrTwL/BM4DXdSdqlw3UEQlOj
7o72+CAIOockB3+nmR1auLY4riOiOytpndWdPSgiBoQjkgQBL1iZm8CE4BJZvaipDmAvYG9zexGM/uPmV0EnIJHEor8zMz+amb3An8G3jKzA83s
tabZpqNM4ABgG7GRmz7X6PtO7wCAI8hGpmSAIgiAIaiOKVYMgcIIgqI1wRIIgqI1wRIIgqI1wRIIgqI1wRIIgcI
1wRIIgCIIgqI1wRIIgCIIgqI1wRIIgCIIgqI3/B9ifa7YBSWs0AAAAAElFTkSuQmCCotation=90);



Item_Type

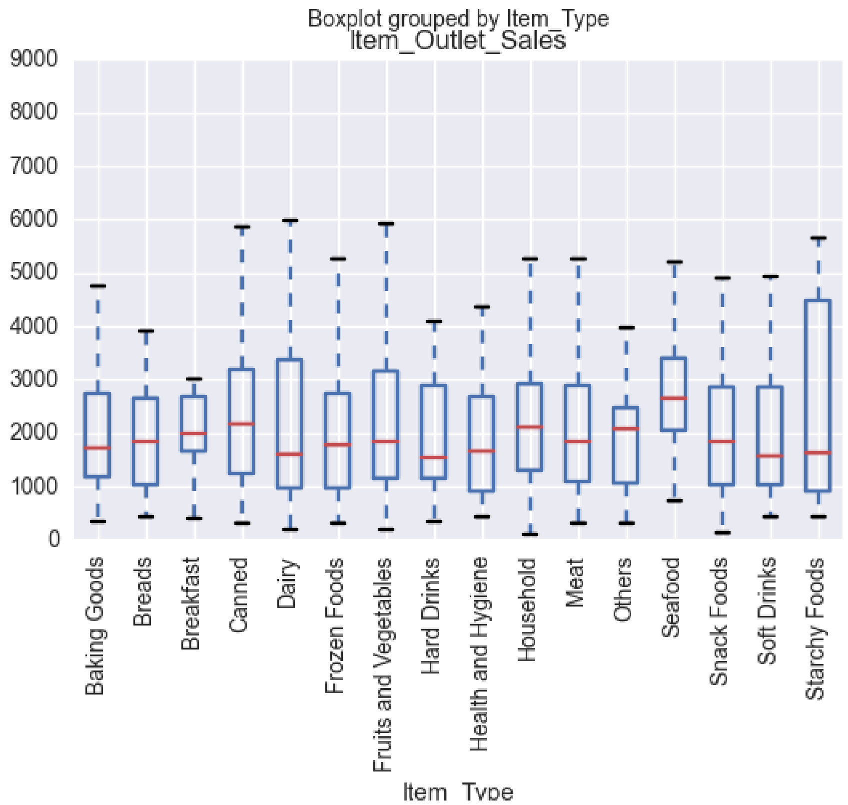
In [24]:

```
data_OUT035.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



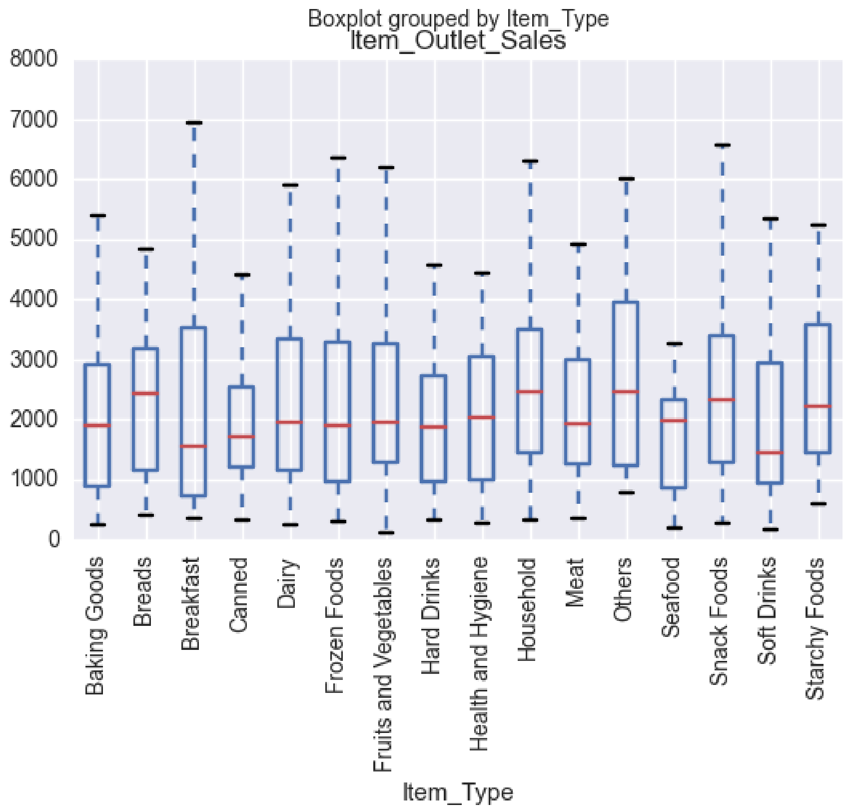
In [25]:

```
data_OUT045.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



In [26]:

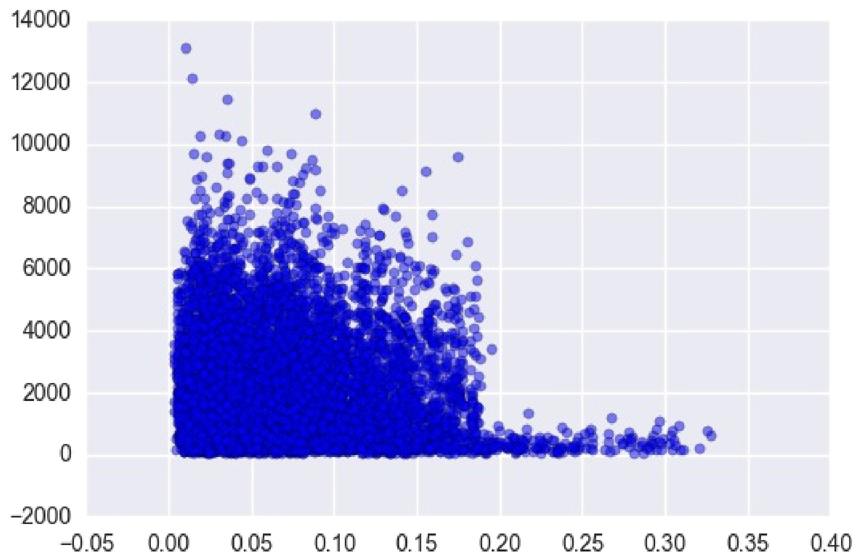
```
data_OUT049.boxplot(column='Item_Outlet_Sales', by='Item_Type')
plt.xticks(rotation=90);
```



Analysis of item visibility with item outlet sales

In [27]:

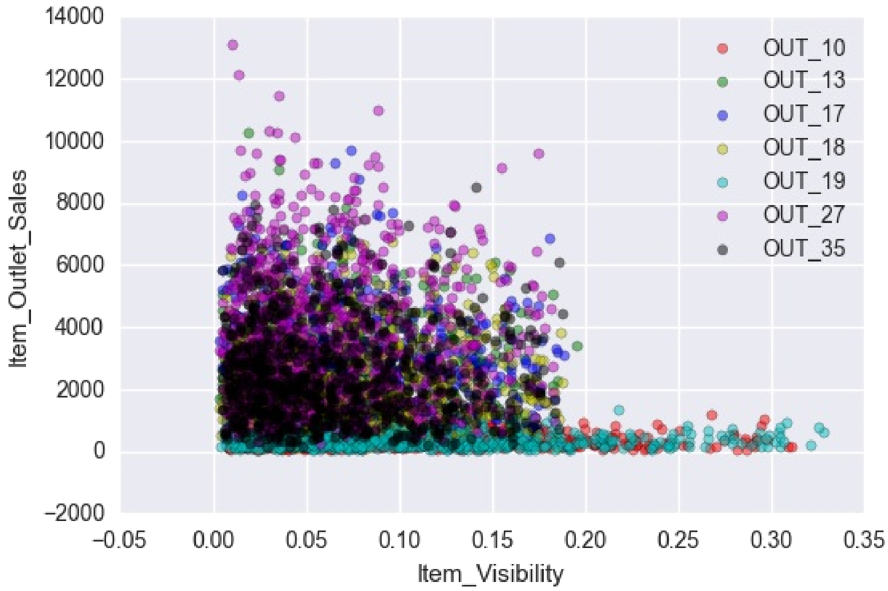
```
plt.scatter(data.Item_Visibility, data.Item_Outlet_Sales, alpha=.5);
```



In [28]:

```
ax = data_OUT010.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='r', alpha=
0.5, label='OUT_10')
data_OUT013.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='g', ax=ax, alph
```

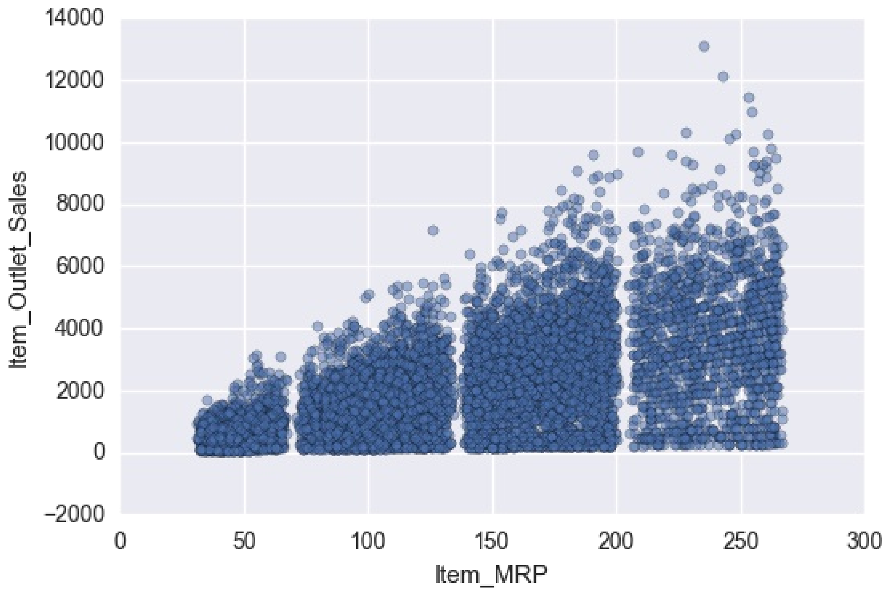
```
a=0.5, label='OUT_13')
data_OUT017.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='b', ax=ax, alph
a=0.5, label='OUT_17')
data_OUT018.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='y', ax=ax, alph
a=0.5, label='OUT_18')
data_OUT019.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='c', ax=ax, alph
a=0.5, label='OUT_19')
data_OUT027.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='m', ax=ax, alph
a=0.5, label='OUT_27')
data_OUT035.plot(kind='scatter', x = 'Item_Visibility', y = 'Item_Outlet_Sales', c='k', ax=ax, alph
a=0.5, label='OUT_35')
plt.legend(loc='best');
```



Correlation between item prices and sales

In [29]:

```
data.plot(kind='scatter', x = 'Item_MRP', y = 'Item_Outlet_Sales', alpha=.5);
```

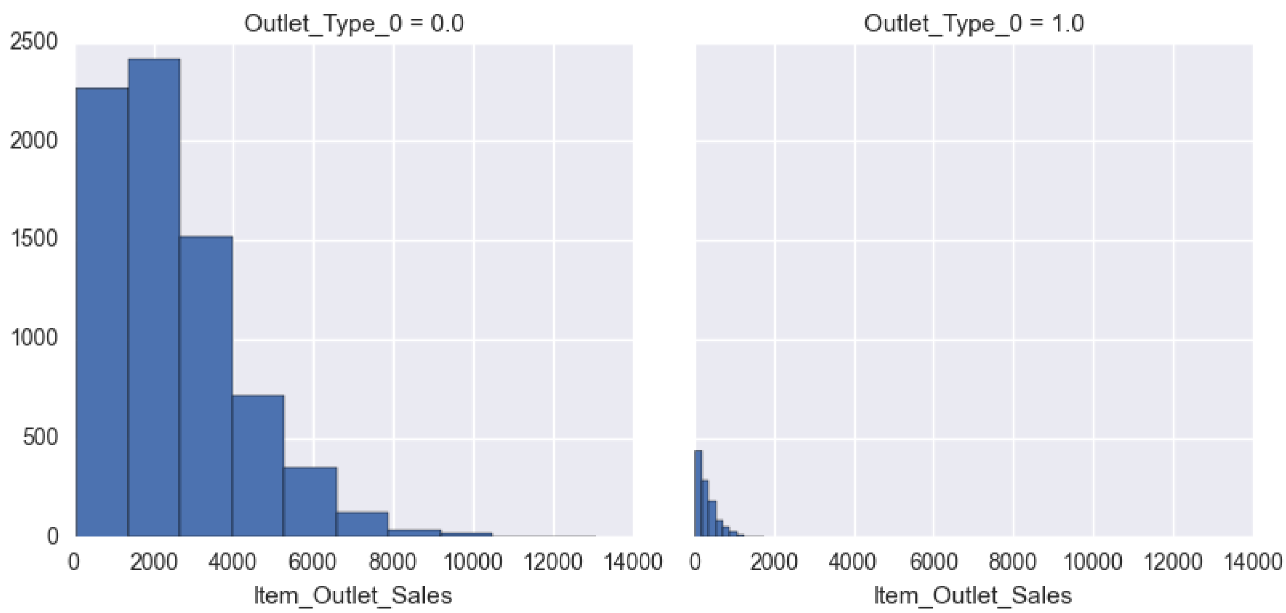


In [30]:

```
sns.FacetGrid(data, col='Item_Type', size=4, col_wrap=5) \
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
```

In [31]:

```
sns.FacetGrid(data, col='Outlet_Type_0', size=4) \
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
```



In [ ]:

```
Numerical:

    1 column:
```

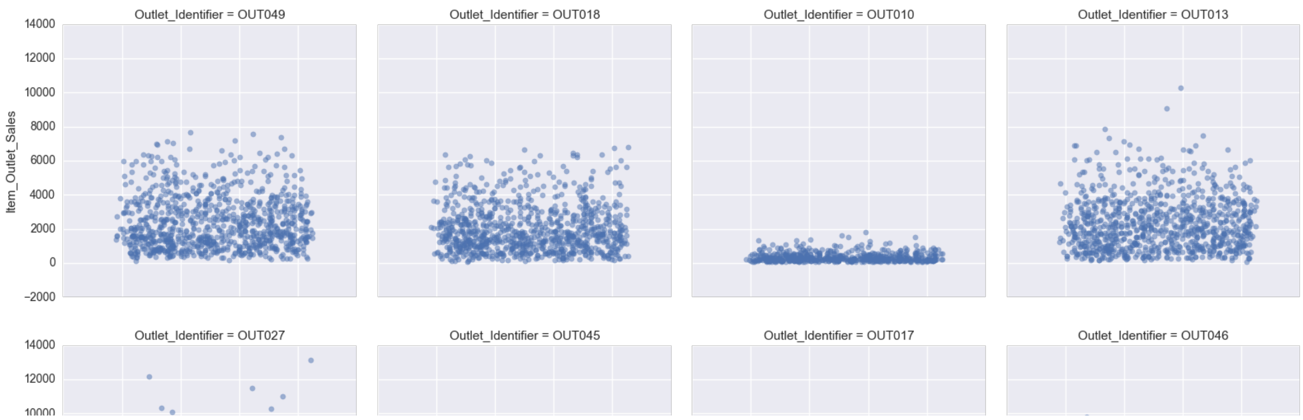
```
    1-column:
          histgram, density plot, gaussion plot

             multivariate analysis: Scatter, Histogram


Categorical:
    1- Bar,Factor plot, Pie chart,
```

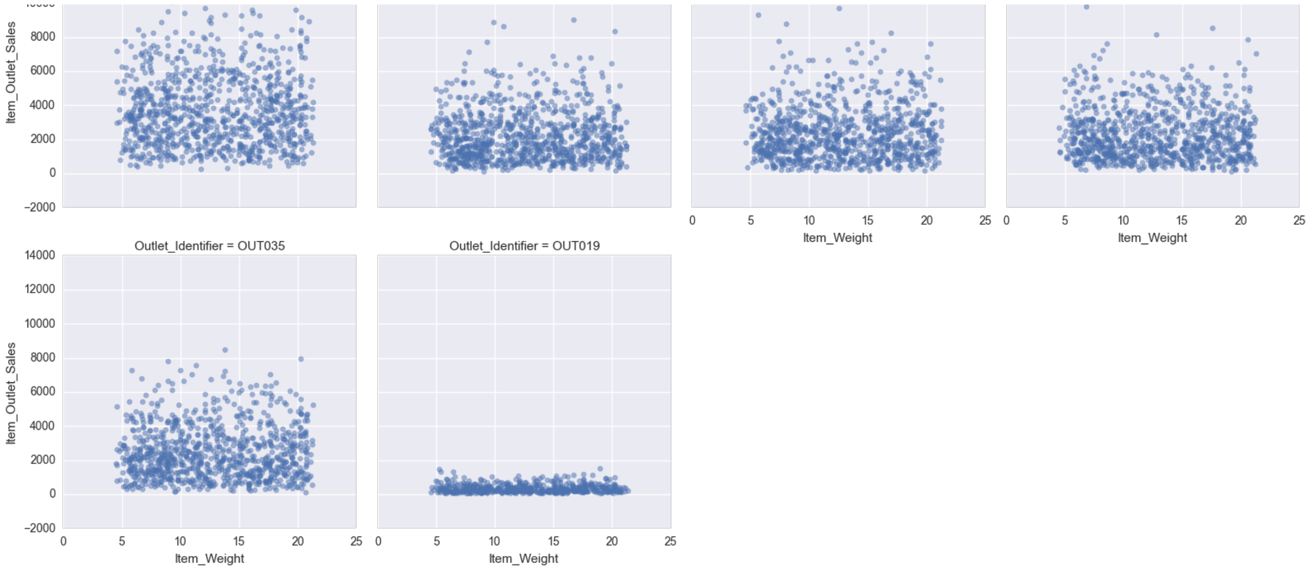
In [32]:

```
sns.FacetGrid(data, col='Outlet_Identifier', col_wrap=4, size=4) \
    .map(plt.scatter, 'Item_MRP', 'Item_Outlet_Sales', c='b', alpha=0.5) \
    .add_legend();
```

In [33]:

```
sns.FacetGrid(data, col='Outlet_Identifier', col_wrap=4, size=4) \
    .map(plt.scatter, 'Item_Weight', 'Item_Outlet_Sales', c='b', alpha=0.5) \
    .add_legend();
```





In [34]:

```
#drop the columns whose data types are converted.  Item type which is of
#16 categories has been reduced to 3 categories and instead of the establishment year
#we consider age of the stores for the analysis.

data = data.drop('Item_Type' , axis = 1)
```

In [35]:

```
data.head()
```

Out[35]:

	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Establishment_Year	Outlet_Identifier
0	FDA15	249.8092	3735.1380	0.016047	9.30	1999	OUT049
1	DRC01	48.2692	443.4228	0.019278	5.92	2009	OUT018
2	FDN15	141.6180	2097.2700	0.016760	17.50	1999	OUT049
3	FDX07	182.0950	732.3800	0.017834	19.20	1998	OUT010
4	NCD19	53.8614	994.7052	0.009780	8.93	1987	OUT013

5 rows × 37 columns

In [36]:

```
data.head()
```

Out[36]:

	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Establishment_Year	Outlet_Identifier
0	FDA15	249.8092	3735.1380	0.016047	9.30	1999	OUT049
1	DRC01	48.2692	443.4228	0.019278	5.92	2009	OUT018
2	FDN15	141.6180	2097.2700	0.016760	17.50	1999	OUT049
3	FDX07	182.0950	732.3800	0.017834	19.20	1998	OUT010
4	NCD19	53.8614	994.7052	0.009780	8.93	1987	OUT013

5 rows × 37 columns

In [37]:

```
#Divide the data set back to train and test
train=data.loc[data['source']=='train']
```

```
test=data.loc[data['source']=='test']
train.head
#Drop unnecessary columns
test.drop(['source','Item_Outlet_Sales'],axis=1,inplace=True)
train.drop(['source'],axis=1,inplace=True)
```

In [38]:

```
#Creating a 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
```

In [39]:

```
base1['Item_Outlet_Sales']=base1['Outlet_Identifier'].apply(lambda x: train.loc[train['Outlet_Ident
ifier']==x]
                                                                        ['Item_Outlet_Sales'].mean())
```

In [ ]:

```
y = ax+ b
```

In [40]:

```
#function to create submission file
#define target and ID Columns for submission
target = 'Item_Outlet_Sales'
IDCol = ['Item_Identifier','Outlet_Identifier']
from sklearn import cross_validation, metrics
```

In [41]:

```
# Linear Regression
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import AdaBoostRegressor
%matplotlib inline

predictors = [x for x in train.columns if x not in [target]+IDCol]
#print predictors
alg1= LinearRegression(normalize=True)
```

In [42]:

```
#fit the algorithm on the data
alg1.fit(train[predictors],train[target])
```

Out[42]:

```
LinearRegression(copy_X=True, fit_intercept=True, normalize=True)
```

In [43]:

```
#predict training data set
train_predictions = alg1.predict(train[predictors])
```

In [44]:

```
#performs cross validation
cv_score =
cross_validation.cross_val_score(alg1,train[predictors],train[target],cv=20,scoring='mean_squared_e
rror')
cv_score=np.sqrt(np.abs(cv_score))
```

In [45]:

```
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_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))
```

```
Model Report :
RMSE : 1127
CV Score : Mean - 1132 | Std - 45.02 | Min - 1075 | Max - 1209
```

In [46]:

```
#Predict on testing data
test[target]=alg1.predict(test[predictors])
```

In [47]:

```
#Ridge Regression
predictors = [x for x in train.columns if x not in [target]+IDCol]
alg2=Ridge(alpha=0.05,normalize=True)

#fit the algorithm on the data
alg2.fit(train[predictors],train[target])
```

Out[47]:

```
Ridge(alpha=0.05, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=True, solver='auto', tol=0.001)
```

In [48]:

```
#predict training data set
train_predictions = alg2.predict(train[predictors])

#Predict on testing data
test[target]=alg2.predict(test[predictors])
```

In [49]:

```
#performs cross validation
cv_score =
cross_validation.cross_val_score(alg2,train[predictors],train[target],cv=20,scoring='mean_squared_e
rror')
cv_score=np.sqrt(np.abs(cv_score))
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_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))
```

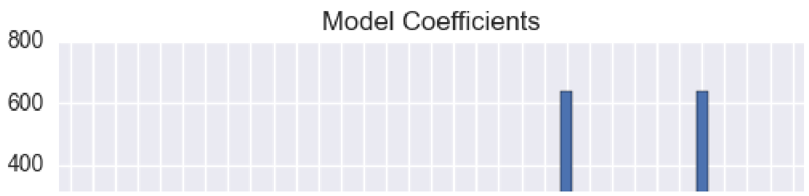
```
Model Report :
RMSE : 1129
CV Score : Mean - 1130 | Std - 44.63 | Min - 1076 | Max - 1217
```

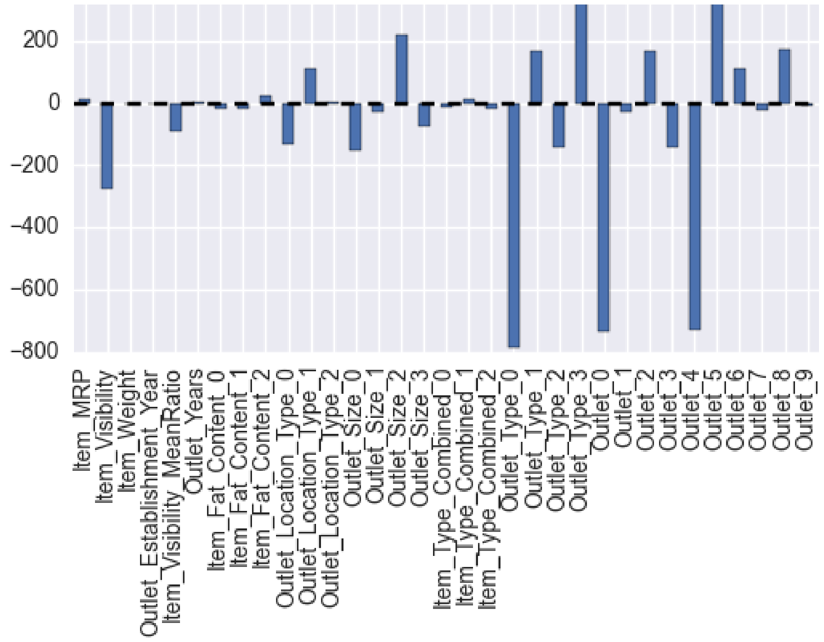
In [50]:

```
coef2 = pd.Series(alg2.coef_,predictors)
coef2.plot(kind='bar',title = 'Model Coefficients')
```

Out[50]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f25f908>
```





In [51]:

```
ada1=AdaBoostRegressor(base_estimator=alg1,learning_rate=0.9)
#fit the algorithm on the data
ada1.fit(train[predictors],train[target])
```

Out[51]:

```
AdaBoostRegressor(base_estimator=LinearRegression(copy_X=True, fit_intercept=True,
normalize=True),
         learning_rate=0.9, loss='linear', n_estimators=50,
         random_state=None)
```

In [52]:

```
#predict training data set
train_predictions = ada1.predict(train[predictors])

#Predict on testing data
test[target]=ada1.predict(test[predictors])
```

In [53]:

```
cv_score =
cross_validation.cross_val_score(ada1,train[predictors],train[target],cv=20,scoring='mean_squared_e
rror')
cv_score=np.sqrt(np.abs(cv_score))
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_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))
```

```
Model Report :
RMSE : 1127
CV Score : Mean - 1157 | Std - 40.8 | Min - 1076 | Max - 1224
```

In [54]:

```
#DecisionTree Model
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)
ada2=AdaBoostRegressor(base_estimator=alg3)#,learning_rate=0.9)


ada2.fit(train[predictors],train[target])
```

Out[54]:

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(compute_importances=None, criterion='mse',
max_depth=15,
           max_features=None, max_leaf_nodes=None, min_density=None,
           min_samples_leaf=100, min_samples_split=2, random_state=None,
           splitter='best'),
         learning_rate=1.0, loss='linear', n_estimators=50,
         random_state=None)
```

In [55]:

```
alg3=DecisionTreeRegressor(max_depth=15,min_samples_leaf=100)
alg3.fit(train[predictors],train[target])
```

Out[55]:

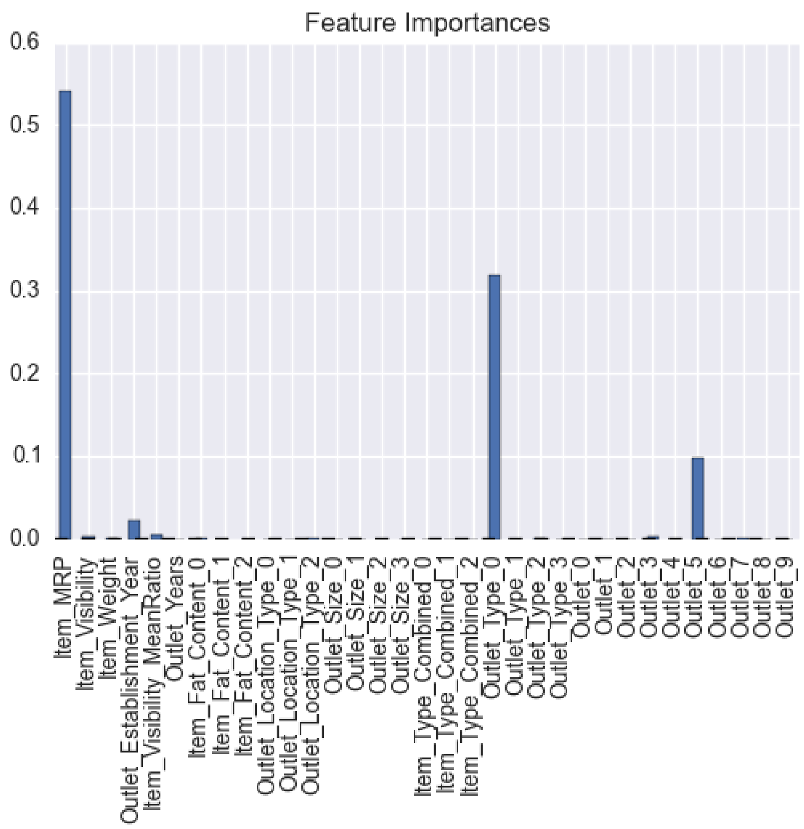
```
DecisionTreeRegressor(compute_importances=None, criterion='mse', max_depth=15,
           max_features=None, max_leaf_nodes=None, min_density=None,
           min_samples_leaf=100, min_samples_split=2, random_state=None,
           splitter='best')
```

In [56]:

```
coef5=pd.Series(alg3.feature_importances_,predictors)
coef5.plot(kind='bar',title='Feature Importances')
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ad5fcc0>
```



In [57]:

```
cv_score =
cross_validation.cross_val_score(ada2,train[predictors],train[target],cv=20,scoring='mean_squared_e
rror')
cv_score=np.sqrt(np.abs(cv_score))
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_predictions))
print "CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g"%(np.mean(cv_score),np.std(cv
```

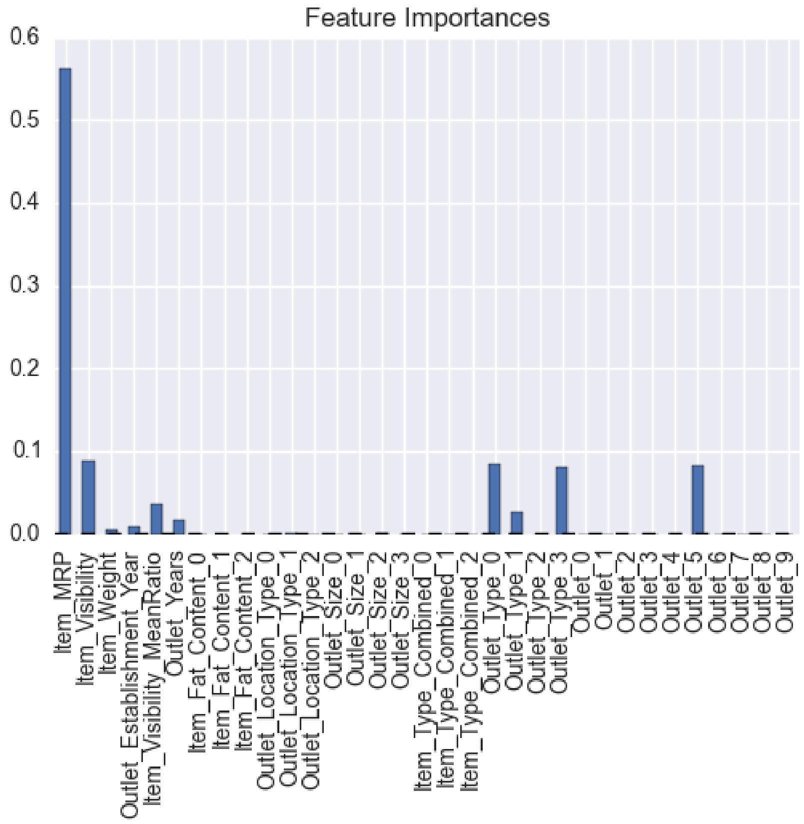
```
_score),np.min(cv_score),np.max(cv_score))
```

```
Model Report :
RMSE : 1127
CV Score : Mean - 1250 | Std - 39.42 | Min - 1180 | Max - 1340
```

In [58]:

```
#RandomForestRegressor
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=3,min_samples_leaf=300,n_jobs=4)
ada2=AdaBoostRegressor(base_estimator=alg5)#,learning_rate=0.9)
#fit the algorithm on the data
ada2.fit(train[predictors],train[target])
coef5=pd.Series(ada2.feature_importances_,predictors)
coef5.plot(kind='bar',title='Feature Importances')
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_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))
```

```
Model Report :
RMSE : 1127
CV Score : Mean - 1250 | Std - 39.42 | Min - 1180 | Max - 1340
```

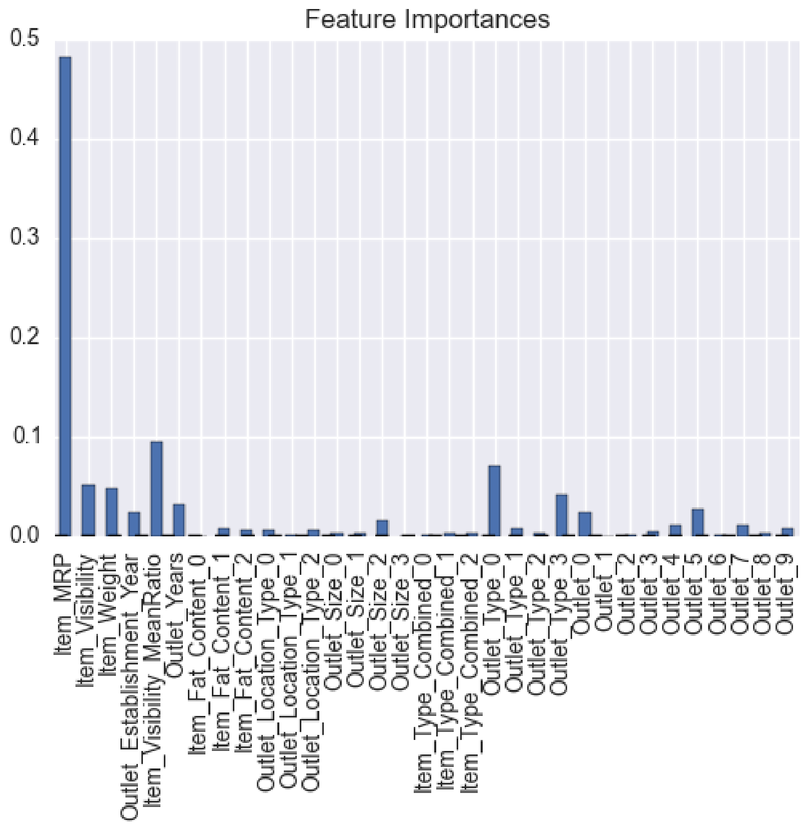


In [ ]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.grid_search import GridSearchCV

predictors=[x for x in train.columns if x not in [target]+IDCol]
alg1= LinearRegression(normalize=True)
alg7=GradientBoostingRegressor(learning_rate=0.1,max_depth=5,random_state=10,
                              n_estimators=70,min_samples_split=1000,min_samples_leaf=50,
                               max_leaf_nodes=7,max_features='sqrt')#,init=alg1)#loss='huber'
alg7.fit(train[predictors],train[target])
# param_test1 = {'n_estimators':range(20,81,10)}
# gsearch1 = GridSearchCV(estimator = GradientBoostingRegressor(learning_rate=0.1,
```

```
min_samples_split=500,min_samples_leaf=50,max_depth=8,max_features='sqrt',subsample=0.8,random_sta
),
# param_grid = param_test1, scoring='mean_squared_error',n_jobs=4,iid=False, cv=5)
# gsearch1.fit(train[predictors],train[target])

coef5=pd.Series(alg7.feature_importances_,predictors)
coef5.plot(kind='bar',title='Feature Importances')
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_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))
```

```
Model Report :
RMSE : 1127
CV Score : Mean - 1250 | Std - 39.42 | Min - 1180 | Max - 1340
```



In [ ]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.grid_search import GridSearchCV

param_test2 = {'max_depth':range(5,16,2), 'min_samples_split':range(200,1001,200)}
gsearch2 = GridSearchCV(estimator = GradientBoostingRegressor(learning_rate=0.1, n_estimators=70, m
ax_features='sqrt', subsample=0.8, random_state=10),
param_grid = param_test2, scoring='mean_squared_error',n_jobs=4,iid=False, cv=5)
gsearch2.fit(train[predictors],train[target])
gsearch2.grid_scores_, gsearch2.best_params_, gsearch2.best_score_
#Print model report
print "\nModel Report :"
print "RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(train[target].values,train_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))
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.grid_search import GridSearchCV

param_test3 = {'min_samples_split':range(1000,2100,200), 'min_samples_leaf':range(30,71,10)}
gsearch3 = GridSearchCV(estimator = GradientBoostingRegressor(learning_rate=0.1,
```

```
n_estimators=70,max_depth=5,max_features='sqrt', subsample=0.8, random_state=10),
param_grid = param_test3, scoring='mean_squared_error',n_jobs=4,iid=False, cv=5)
gsearch3.fit(train[predictors],train[target])
gsearch3.grid_scores_, gsearch3.best_params_, gsearch3.best_score_
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.grid_search import GridSearchCV

param_test4 = {'max_features':range(2,20,2)}
gsearch4 = GridSearchCV(estimator = GradientBoostingRegressor(learning_rate=0.1,
n_estimators=70,max_depth=5, min_samples_split=1000, min_samples_leaf=50, subsample=0.8,
random_state=10),
param_grid = param_test4, scoring='mean_squared_error',n_jobs=4,iid=False, cv=5)
gsearch4.fit(train[predictors],train[target])
gsearch4.grid_scores_, gsearch4.best_params_, gsearch4.best_score_
```

In [ ]:

```
 7+ Models === the less rmse we be chosen for depoyment
```

In [ ]:

```
#Export submission file
filename= "Submission.csv"
IDCol.append(target)
submission = pd.DataFrame({x : test[x] for x in IDCol})
submission.to_csv("C:/Users/vamsi/Bigmartsales/"+filename,index=False)
```