```
Asad Ashraf Karel
```

[https://datahack.analyticsvidhya.com/contest/practice-problem-big-mart-sales-iii/ (https://datahack.analyticsvidhya.com/contest/practice-problem-big-mart-sales-iii/)](https://datahack.analyticsvidhya.com/contest/practice-problem-big-mart-sales-iii/)

In [1]:

```python
from warnings import filterwarnings
filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
import statsmodels
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.anova import anova_lm
from statsmodels.formula.api import ols
from statsmodels.tools.eval_measures import rmse

from scipy import stats
from scipy.stats import shapiro

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

import scipy.stats as st

from statsmodels.formula.api import ols

plt.rcParams["figure.figsize"] = [15,8]
```

Importing the datasets:

In [2]:

```python
train = pd.read_csv('Bigmart_train.csv')
test = pd.read_csv('Bigmart_test.csv')
```

In [3]:

```python
train.shape, test.shape
```

Out[3]: ((8523, 12), (5681, 11))

Exploratory Data Analysis

1. Univariate Analysis:

In [4]:

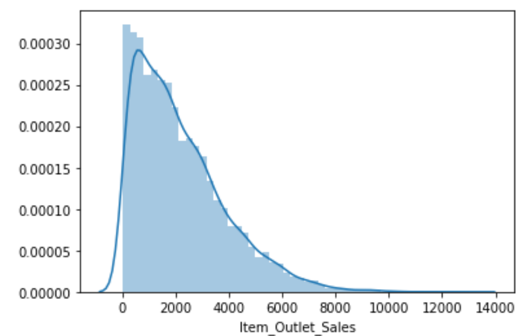
```python
train.head()
```

Out[4]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OU
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OU
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OU
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OU
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OU

In [5]:

```python
sns.distplot(train.Item_Outlet_Sales)
plt.show()
```



There are some stuffs which are having a huge sale values.

In [6]:
```
train.Item_Outlet_Sales.describe()
```

Out[6]:
```
count     8523.000000
mean      2181.288914
std       1706.499616
min         33.290000
25%        834.247400
50%       1794.331000
75%       3101.296400
max      13086.964800
Name: Item_Outlet_Sales, dtype: float64
```

In [7]:
```
train.Item_Outlet_Sales.mean()
```

Out[7]:
```
2181.2889135750365
```

Making the test information set for the prediction:

In [8]:
```
solution = pd.DataFrame({'Item_Identifier': test.Item_Identifier,
                        'Outlet_Identifier':test.Outlet_Identifier,
                        'Item_Outlet_Sales': train.Item_Outlet_Sales.mean()}
```

In [9]:
```
solution.head()
```

Out[9]:

	Item_Identifier	Outlet_Identifier	Item_Outlet_Sales
0	FDW58	OUT049	2181.288914
1	FDW14	OUT017	2181.288914
2	NCN55	OUT010	2181.288914
3	FDQ58	OUT017	2181.288914
4	FDY38	OUT027	2181.288914

In [10]:
```
solution.to_csv('Basemodel.csv', index=False)
```

Your score for this submission is (RMSE) : 1773.8251377790564

In [11]:
```
train.columns
```

Out[11]:
```
Index(['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', 'Item_Outlet_Sales'],
      dtype='object')
```

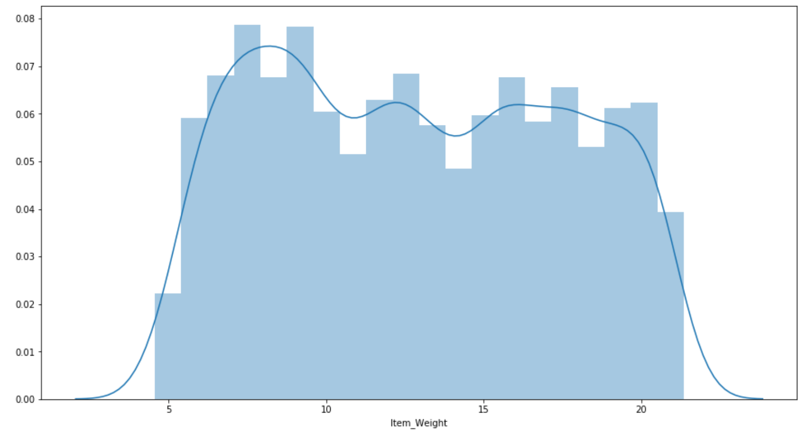
fig, axes = plt.subplots(nrows=2,ncols=2,figsize=(15,10))

sns.distplot(train.Item_Weight, ax=[0][0]) sns.distplot(train.Item_Visibility, ax=[0][1]) sns.distplot(test.Item_Visibility, ax=[1][0]) sns.distplot(train.Item_MRP, ax=[1][1])

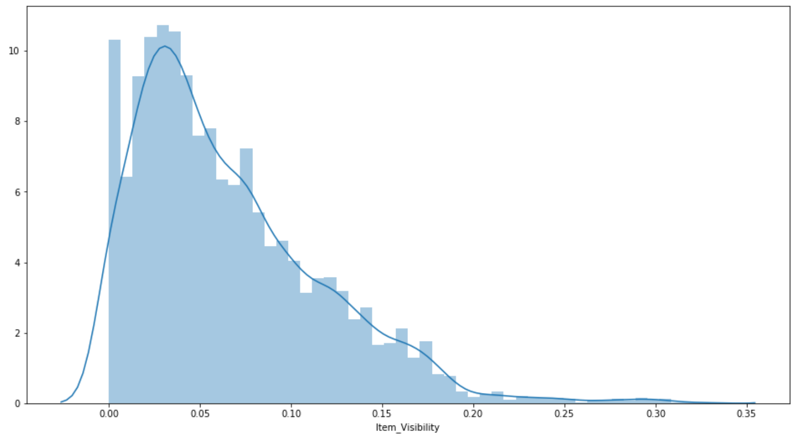
axes[0][0].title.set_text('train.Item_Weight') axes[0][1].title.set_text('train.Item_Visibility') axes[1][0].title.set_text('test.Item_Visibility') axes[1][1].title.set_text('train.Item_MRP')

plt.show()

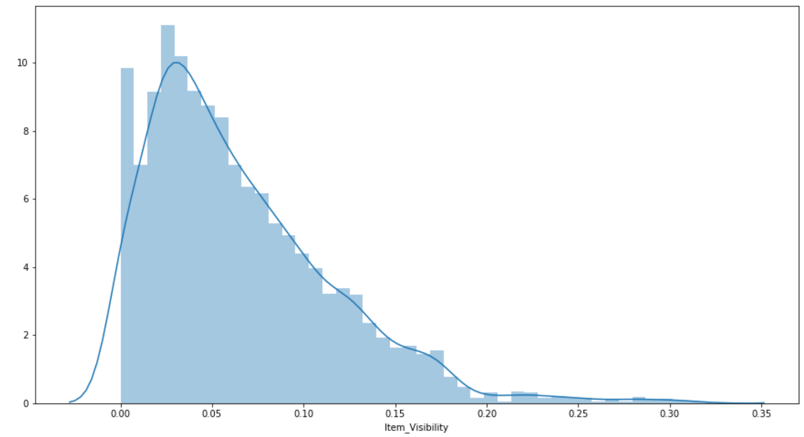
In [12]:
```
plt.rcParams["figure.figsize"] = [15,8]
sns.distplot(train.Item_Weight.dropna())
plt.show()
```



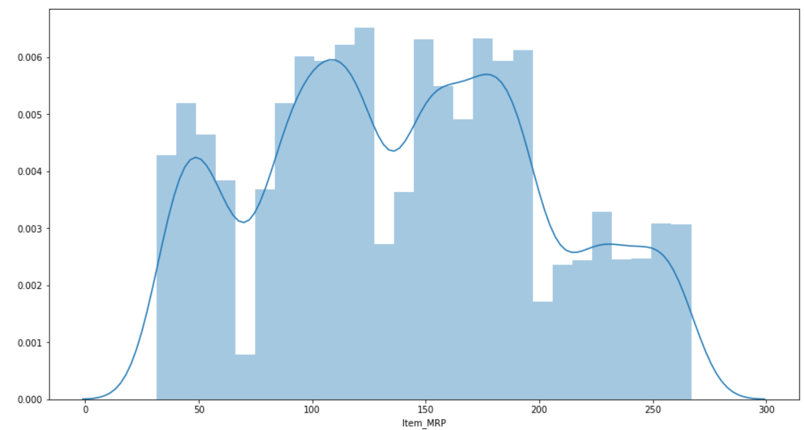
In [13]:
```
sns.distplot(train.Item_Visibility)
plt.show()
```



In [14]:
```
sns.distplot(test.Item_Visibility)
plt.show()
```



In [15]:
```
sns.distplot(train.Item_MRP)
plt.show()
```



Tentetively we had a glance that, our data has some behaviours, with respect to market appearance.

Dealing with the features:

In [16]:
```
train.columns
```

Out[16]:
```
Index(['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', 'Item_Outlet_Sales'],
      dtype='object')
```

In [17]:
```
train.Item_Fat_Content.value_counts()
```

Out[17]:
```
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```

In [18]:
```
test.Item_Fat_Content.value_counts()
```

Out[18]:
```
Low Fat    3396
Regular    1935
LF          206
reg          78
low fat      66
Name: Item_Fat_Content, dtype: int64
```

Here we observe an uncertainty into the distribution of the categories.

Let's resolve this:

In [19]:
```
train.Item_Fat_Content.replace(to_replace = ['LF','reg','low fat'], value =
                              inplace=True)

test.Item_Fat_Content.replace(to_replace = ['LF','reg','low fat'], value = [
                              inplace=True)
```

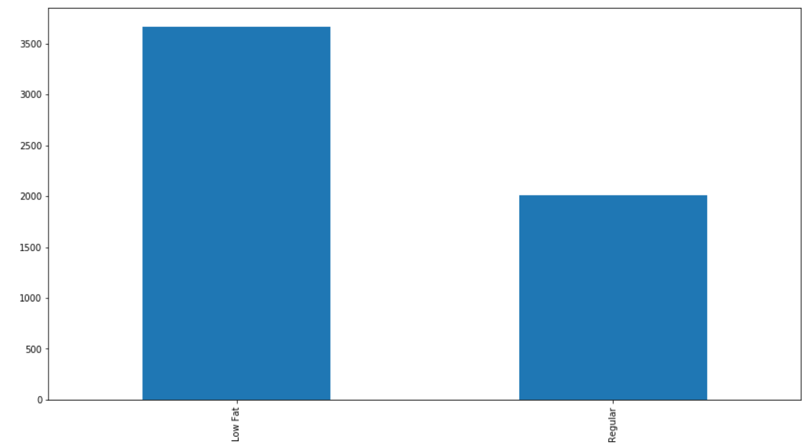
In [20]:
```
print(pd.DataFrame(train.Item_Fat_Content.value_counts())) ;print(); print(p
```

```
         Item_Fat_Content
Low Fat              5517
Regular              3006

         Item_Fat_Content
Low Fat              3668
Regular              2013
```

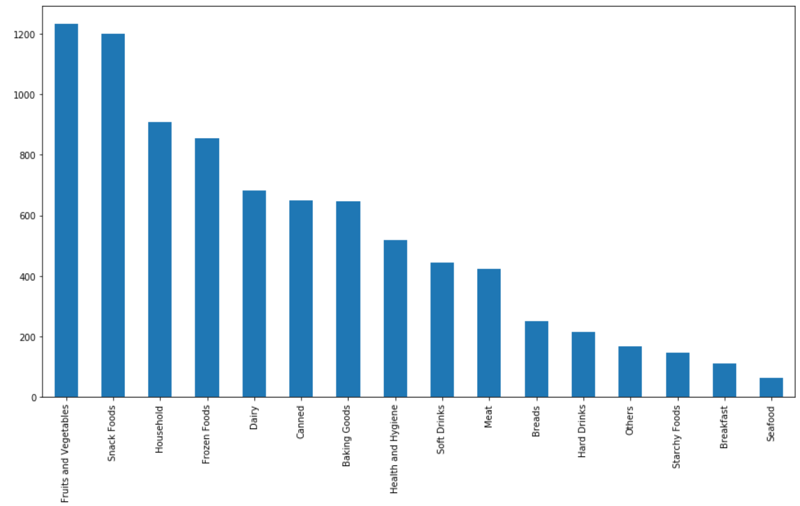
Its all set with the proper categories.

In [21]:
```
test.Item_Fat_Content.value_counts().plot(kind='bar')
plt.show()
```



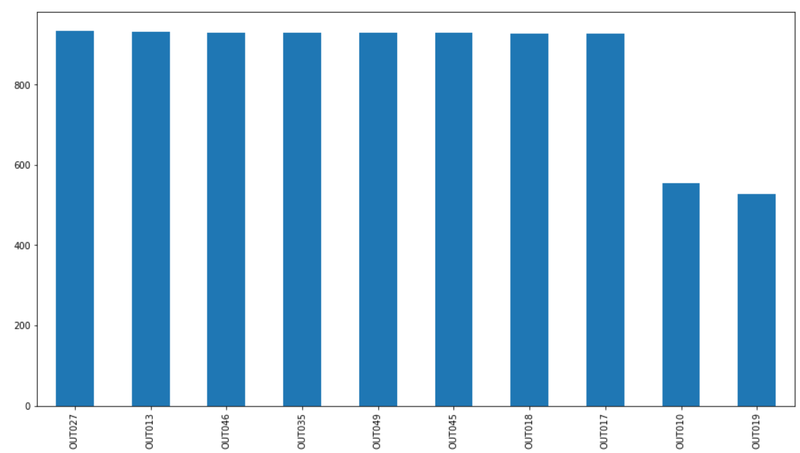
Let's find out, which kind of stuff has a huge sale:

In [22]:
```
train.Item_Type.value_counts().plot(kind='bar')
plt.show()
```



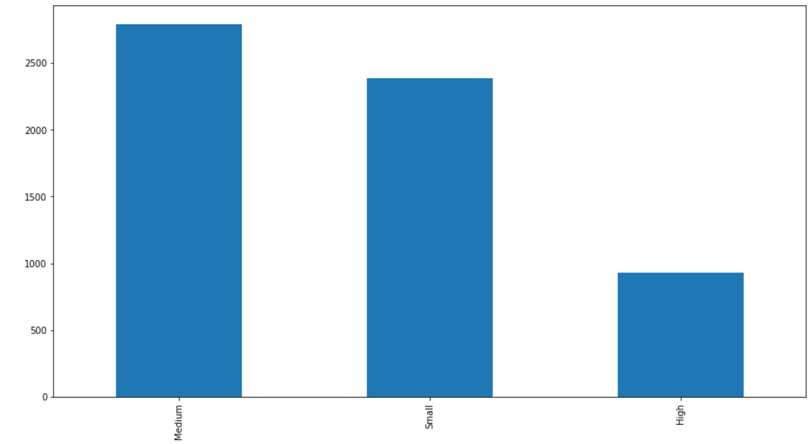
Let's see, which identifier has a huge demand:

In [23]:
```
train.Outlet_Identifier.value_counts().plot(kind='bar')
plt.show()
```



Let's see what size of the stuff is in demand:

In [24]:
```
train.Outlet_Size.value_counts().plot(kind='bar')
plt.show()
```



Summary:

- Low sale is 33 approx and high sale is 13k approx.
- We observed that fruits vegitables and snak foods have the highest sales all over. Even seafoods are in the range of sales, but since they are quite expensive, they are bought occasionally. Rest all are on their respective sales.
- OUT027 has a huge demand of the sales with respect to the other oulets.
- The things are bought under the medium quantity. We can say that **The people who are visiting bars are from city area and little far from the mart, thats why they are not storing stuff.**

---

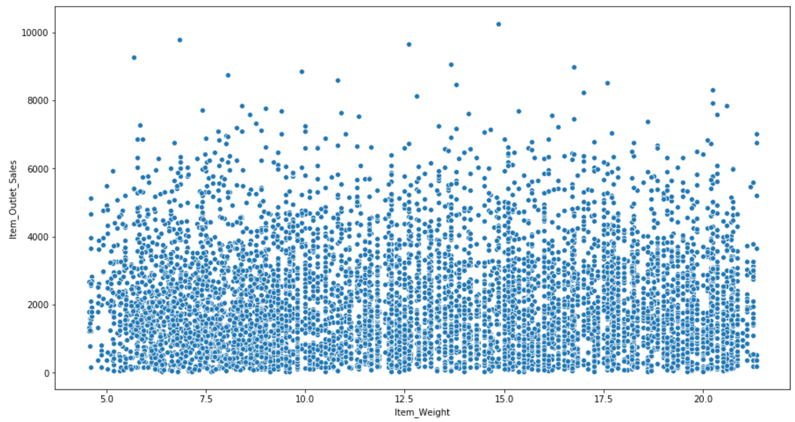
2. Bivariate Analysis:

- Num vs Num (Pred vs TGT)
- Cat vs Num (Cat vs TGT)

In [25]:
```
train.columns
```

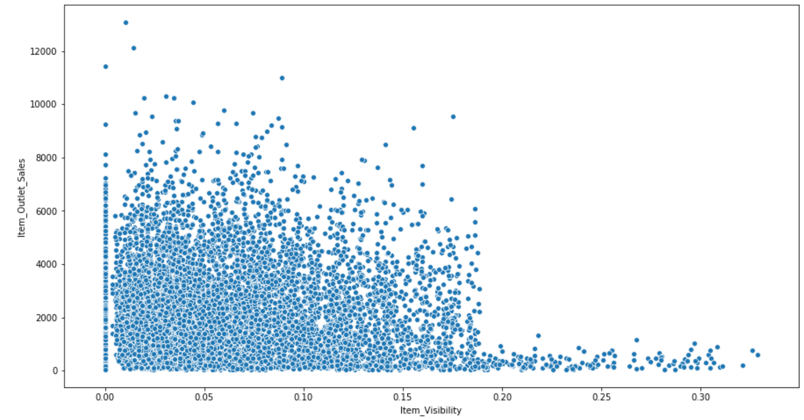
Out[25]:
```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier',
       'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

In [26]:
```
sns.scatterplot(train.Item_Weight, train.Item_Outlet_Sales)
plt.show()
```

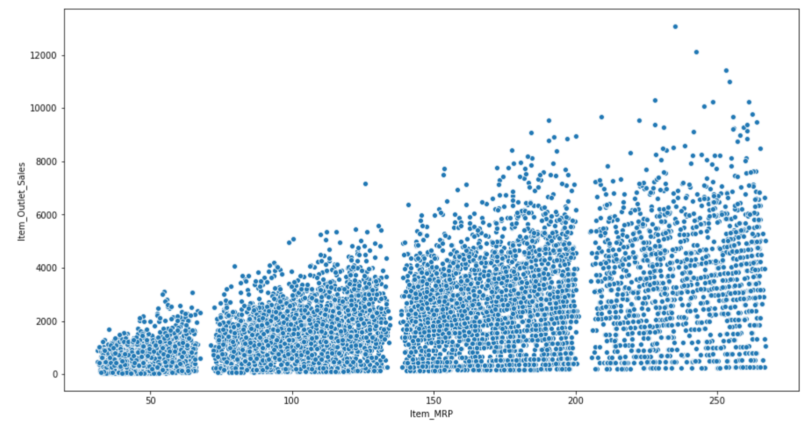


No specific relation seems to be weight. Which mean all the stuffs are being sold are essentails.

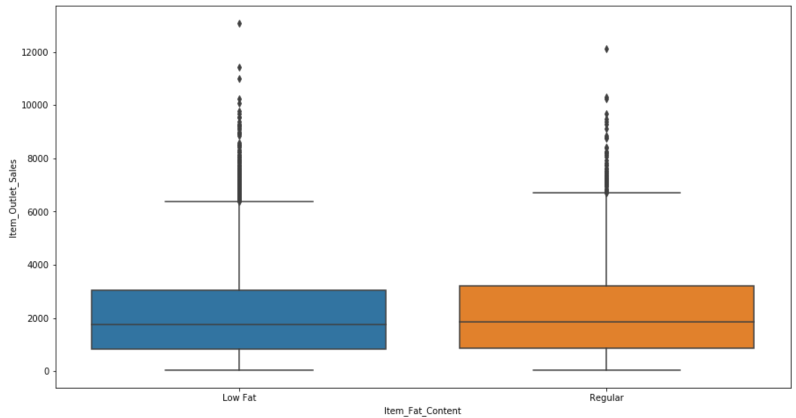
In [27]:
```
sns.scatterplot(train.Item_Visibility, train.Item_Outlet_Sales)
plt.show()
```



In [28]:
```
sns.scatterplot(train.Item_MRP, train.Item_Outlet_Sales)
plt.show()
```



In [29]:
```
sns.boxplot(train.Item_Fat_Content, train.Item_Outlet_Sales)
plt.show()
```



In [30]:
```
train.groupby('Item_Fat_Content')['Item_Outlet_Sales'].describe()
```

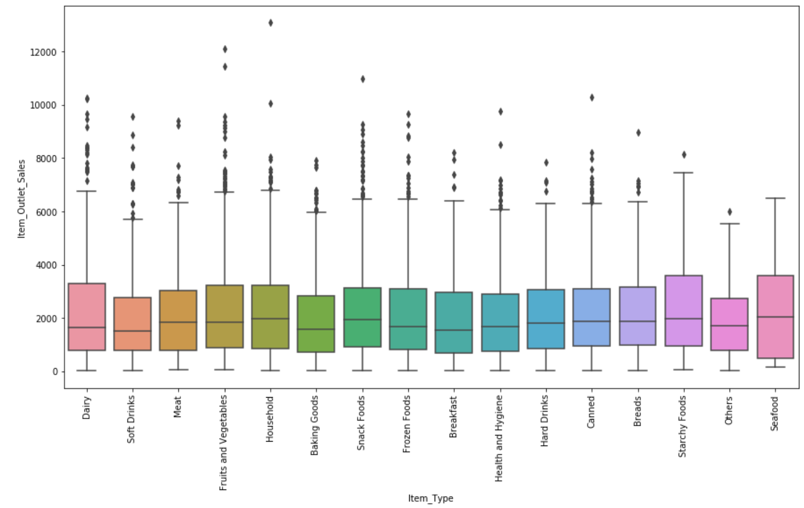
Out[30]:

Item_Fat_Content	count	mean	std	min	25%	50%	75%	
Low Fat	5517.0	2157.711534	1697.973824	33.2900	826.2578	1765.0358	3050.69560	130
Regular	3006.0	2224.561170	1721.480865	33.9558	857.5504	1844.5989	3198.66965	121

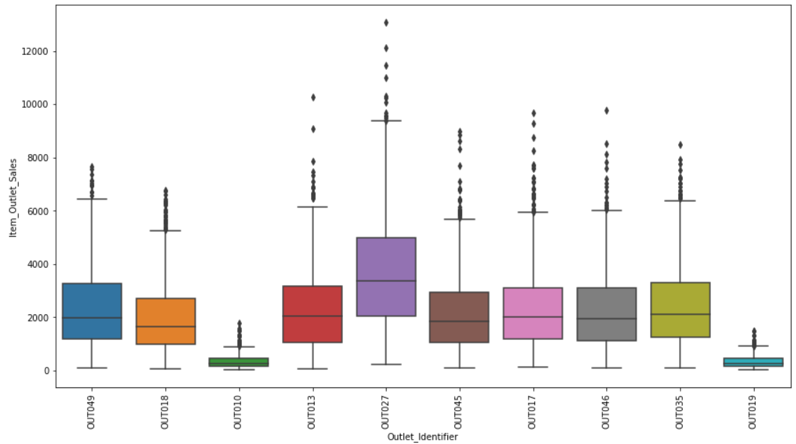
In [31]:
```
train.columns
```

Out[31]:
```
Index(['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', 'Item_Outlet_Sales'],
      dtype='object')
```

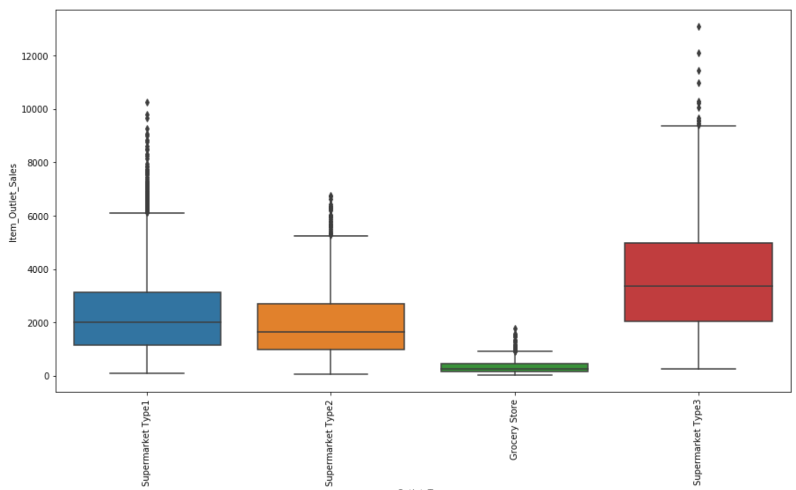

In [32]:
```
sns.boxplot(train.Item_Type, train.Item_Outlet_Sales)
plt.xticks(rotation=90)
plt.show()
```



In [33]:
```
sns.boxplot(train.Outlet_Identifier, train.Item_Outlet_Sales)
plt.xticks(rotation=90)
plt.show()
```



In [34]:
```
sns.boxplot(train.Outlet_Type, train.Item_Outlet_Sales)
plt.xticks(rotation=90)
plt.show()
```



Summary:

- Supermarket type3 has a huge sale, since grocery store has a very less sales. Here we can again say so accurately that the people are visiting from supermarkets, hence this data is belong to **city**, hence people are not visiting to the grocery or stationary.
- Again OUT027 has the high in demand. It is biggest revenue generator.
- Even fruits vegitables and snak foods have the highest sales.
- Both the things are being sold similarly, either **Low Fat** or **Regular**.
- As price increasing, sales quantity is also increasing. Which mean there are few things which the tremedously important, which are having no concern with price.
- Some stuffs are more likely present into the shops but their sale is not that much. Hence we can say, the **Electronics** things are not bought casually.

Dealing with missing values:

In [35]:
```
train.isnull().sum()
```

Out[35]:
```
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

Weight & Size have the nan values. Let's check how are the affecting our data.

In [36]:
```
Null = pd.DataFrame(train.isnull().sum(), columns=['Null_values'])
Null['Effect_of_values'] = (Null.Null_values/train.shape[0]) * 100
Null
```

Out[36]:

	Null_values	Effect_of_values
Item_Identifier	0	0.000000
Item_Weight	1463	17.165317
Item_Fat_Content	0	0.000000
Item_Visibility	0	0.000000
Item_Type	0	0.000000
Item_MRP	0	0.000000
Outlet_Identifier	0	0.000000
Outlet_Establishment_Year	0	0.000000
Outlet_Size	2410	28.276428
Outlet_Location_Type	0	0.000000
Outlet_Type	0	0.000000
Item_Outlet_Sales	0	0.000000

Only 2 attributes are having null values, even they are not affecting the data much. 17-28 % respectively.

Hence rather dropping the columns, we can deal with the nan values.

In [37]:
```
train_test = pd.concat((train,test), ignore_index=False)
```

In [38]:
```
train_test.shape
```

Out[38]:
```
(14204, 12)
```

We purposely concatenated both train and test to deal with null values.

In [39]:
```
train_test.isnull().sum()
```

Out[39]:
```
Item_Identifier                 0
Item_Weight                  2439
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  4016
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales            5681
dtype: int64
```

In [40]:
```
train_test.Item_Weight.mean()
```

Out[40]:
```
12.792854228644991
```

In [41]:
```
train_test.loc[train_test.Item_Identifier=="NCD19",'Item_Weight']
```

Out[41]:
```
4       8.93
522     8.93
802     8.93
2129    8.93
2907    8.93
3428    8.93
149      NaN
1944    8.93
5377    8.93
Name: Item_Weight, dtype: float64
```

In [42]:
```
train_test['Item_Weight'] = train_test.groupby('Item_Identifier')['Item_Weig
```

In [43]:
```
train_test.loc[train_test.Outlet_Size.isnull(), 'Outlet_Location_Type'].uniq
```

Out[43]:
```
array(['Tier 3', 'Tier 2'], dtype=object)
```

In [44]:
```
train_test.loc[train_test.Outlet_Size.isnull(), 'Outlet_Type'].unique()
```

Out[44]:
```
array(['Grocery Store', 'Supermarket Type1'], dtype=object)
```

In [45]:
```
train_test.Outlet_Type.unique()
```

Out[45]:
```
array(['Supermarket Type1', 'Supermarket Type2', 'Grocery Store',
       'Supermarket Type3'], dtype=object)
```

Looking above information, we see that Supermarkets and Grocery Store have the nan values, with Teir1 and Teir 2.

Let's deal with these null values:

In [46]:
```
pd.DataFrame(train_test.groupby(['Outlet_Location_Type','Outlet_Type'])['Out
```

Out[46]:

			Outlet_Size
Outlet_Location_Type	**Outlet_Type**	**Outlet_Size**	
Tier 1	**Grocery Store**	**Small**	880
	Supermarket Type1	**Medium**	1550
		Small	1550
Tier 2	**Supermarket Type1**	**Small**	1550
Tier 3	**Supermarket Type1**	**High**	1553
	Supermarket Type2	**Medium**	1546
	Supermarket Type3	**Medium**	1559

We have nan values to Teir2 and Teir3 with grocery store and supermarket type1. And even we are observing to fill small as our type. Because grocery always have small and supermarket type1 has also small. Hence we go with **small** only.

In [47]:
```
train_test.loc[train_test.Outlet_Type=='Grocery Store', 'Outlet_Size'] = 'sm
train_test.loc[train_test.Outlet_Type=='Supermarket Type1', 'Outlet_Size'] =
```

In [48]:
```
train_test.isnull().sum()
```

Out[48]:
```
Item_Identifier                0
Item_Weight                    0
Item_Fat_Content               0
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Size                    0
Outlet_Location_Type           0
Outlet_Type                    0
Item_Outlet_Sales           5681
dtype: int64
```

Now we see, our data has no null values. 5681 are the values of test dataset which we are about to check our model. Hence we are not disturbing this attribute. We are about to resplit the data into the `train` and `test` format. According to our previous nan evaluation,we saw no null values into the sales attribute, hence we strongly prefer this as the test values. Hence we are not discturbing this attribute right now.

Feature Engineering

In [49]:
```
train_test.Item_Visibility.describe()
```

Out[49]:
```
count    14204.000000
mean         0.065953
std          0.051459
min          0.000000
25%          0.027036
50%          0.054021
75%          0.094037
max          0.328391
Name: Item_Visibility, dtype: float64
```

In [50]:
```
print(pd.DataFrame(train_test[train_test.Item_Visibility==0]['Outlet_Size'].
print(pd.DataFrame(train_test[train_test.Item_Visibility==0]['Item_Outlet_Sa
```

```
        Outlet_Size
small           680
Medium          199

   Item_Outlet_Sales
3           732.3800
4           994.7052
5           556.6088
```

Item_Visibility must not be zero, if that is being sold with appropriate price and size.

Let's deal with 0 in visibility:

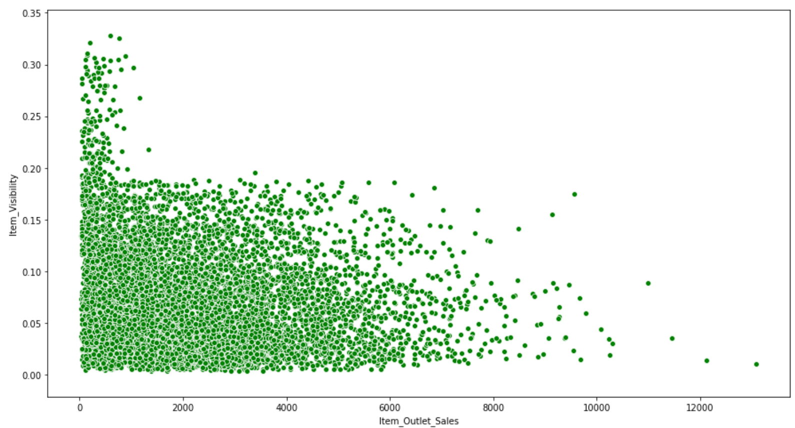
In [51]:
```
train_test['Item_Visibility'] =train_test.groupby('Item_Identifier')['Item_V
```

In [52]:
```
train_test.Item_Visibility.describe()
```

Out[52]:
```
count    14204.000000
mean         0.069710
std          0.049728
min          0.003575
25%          0.031145
50%          0.057194
75%          0.096930
max          0.328391
Name: Item_Visibility, dtype: float64
```

Now let's see the graph of these both attributes:

In [53]:
```
sns.scatterplot(train_test.Item_Outlet_Sales, train_test.Item_Visibility, co
plt.show()
```



We have dropped the stright line of 0. Now graph looks quiet possessive.

As much the machine is simpler, that much it is good. Hence we try our best to give it as lesser categories as possible.

Here we are about to deal with again 2 attributes, which are unnccesserily elaborated into the data. If we manage them, we do not see any hurdle to our data for `Machine Learning modelling` .

In [54]:
```
train_test.head()
```

Out[54]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OU
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OU
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OU
3	FDX07	19.20	Regular	0.017834	Fruits and Vegetables	182.0950	OU
4	NCD19	8.93	Low Fat	0.009780	Household	53.8614	OU

In [55]:
```
train_test.Item_Type.unique()
```

Out[55]:
```
array(['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
       'Household', 'Baking Goods', 'Snack Foods', 'Frozen Foods',
       'Breakfast', 'Health and Hygiene', 'Hard Drinks', 'Canned',
       'Breads', 'Starchy Foods', 'Others', 'Seafood'], dtype=object)
```

In [56]:
```
perishables = ['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
               'Frozen Foods','Breakfast', 'Breads', 'Seafood']

def perish(x):
    if(x in perishables):
        return("Perishables")
    else:
        return("Non Perishables")

train_test['Item_Type_Cat'] = train_test.Item_Type.apply(perish)
```

In [57]:
```
train_test['Outlet_wintage'] =  2013 - train_test.Outlet_Establishment_Year
```

In [58]:
```
def out(x):
    if x == 'OUT027':
        return 'OUT027'
    else:
        return 'Others'
```

In [59]:
```
train_test['Outlet_Identifier_Cat'] = train_test['Outlet_Identifier'].apply(
```

In [60]:
```
mylist = []

for i in train_test.Item_Identifier:
    mylist.append(i[:2])
```

In [61]:
```
train_test['Item_Identifier_Cat'] = mylist
```

Let's drop unnecessery attributes:

In [62]:
```
df = train_test.drop(['Item_Type','Item_Identifier','Outlet_Identifier','Out
```

In [63]:
```
df.head()
```

Out[63]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Type	Ou
0	9.30	Low Fat	0.016047	249.8092	small	Tier 1	Su
1	5.92	Regular	0.019278	48.2692	Medium	Tier 3	Su
2	17.50	Low Fat	0.016760	141.6180	small	Tier 1	Su
3	19.20	Regular	0.017834	182.0950	small	Tier 3	
4	8.93	Low Fat	0.009780	53.8614	small	Tier 3	Su

Here we are observing that `NC` are also showing `Low Fat`. Let's manage that.

In [64]:
```
df.loc[df.Item_Identifier_Cat=='NC','Item_Fat_Content'] = 'Non Edible'
```


In [65]:
```
df.head()
```

Out[65]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Type	Ou
0	9.30	Low Fat	0.016047	249.8092	small	Tier 1	Su
1	5.92	Regular	0.019278	48.2692	Medium	Tier 3	Su
2	17.50	Low Fat	0.016760	141.6180	small	Tier 1	Su
3	19.20	Regular	0.017834	182.0950	small	Tier 3	
4	8.93	Non Edible	0.009780	53.8614	small	Tier 3	Su

`We have dealt with the complete data and we arranged each and every attribute of the data.` **Let's jump on some Machine Learning models**.

In [66]:
```
df.shape, train.shape, test.shape
```

Out[66]: ((14204, 12), (8523, 12), (5681, 11))

In [67]:
```
new_train = df[:8523] ; new_test = df[8523:]
```

In [68]:
```
new_train.shape, new_test.shape
```

Out[68]: ((8523, 12), (5681, 12))

In [69]:
```
new_test = new_test.drop('Item_Outlet_Sales', axis=1)
```

`We have resplitted the data into train and test sets.`

In [70]:
```
new_train.head()
```

Out[70]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Type	Ou
0	9.30	Low Fat	0.016047	249.8092	small	Tier 1	Su
1	5.92	Regular	0.019278	48.2692	Medium	Tier 3	Su
2	17.50	Low Fat	0.016760	141.6180	small	Tier 1	Su
3	19.20	Regular	0.017834	182.0950	small	Tier 3	
4	8.93	Non Edible	0.009780	53.8614	small	Tier 3	Su

In [71]:
```
new_test.head()
```

Out[71]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Type	Ou
0	20.750	Low Fat	0.007565	107.8622	small	Tier 1	Su
1	8.300	Regular	0.038428	87.3198	small	Tier 2	Su
2	14.600	Non Edible	0.099575	241.7538	small	Tier 3	
3	7.315	Low Fat	0.015388	155.0340	small	Tier 2	Su
4	13.600	Regular	0.118599	234.2300	Medium	Tier 3	Su

In [72]:
```
train_dummies = pd.get_dummies(new_train, drop_first=True)
test_dummies = pd.get_dummies(new_test, drop_first=True)
```

In [73]:
```
train_dummies.shape , test_dummies.shape
```

Out[73]: ((8523, 17), (5681, 16))

`See how we managed the data with dummies.` **Now it seems better for further Machine Learning processes.**

Model building:

``````````````````````````````````

In [74]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error, r2_score

LR = LinearRegression()

X = train_dummies.drop('Item_Outlet_Sales', axis=1)
y = train_dummies.Item_Outlet_Sales

xtrain, xtest, ytrain, ytest = train_test_split(X,y, test_size = 0.20, rando

ypred = LR.fit(xtrain,ytrain).predict(xtest)

RMSE = round(np.sqrt(mean_squared_error(ytest, ypred)),4)
RMSE
```

Out[74]: 1068.5986

By setting `random_state = 42` , we got such error `1068.5986` .We observe some changes according to random_state variations.

### Let's try another model for submission:

In [75]:
```
from sklearn.ensemble import BaggingRegressor
Bagg = BaggingRegressor()

pred = Bagg.fit(X,y).predict(test_dummies)
```

In [76]:
```
Prediction = solution.copy()

Prediction['Item_Outlet_Sales'] = pred
```

### Predicted sales values:
```

In [77]:
```
Prediction.head()
```

Out[77]:

| | Item_Identifier | Outlet_Identifier | Item_Outlet_Sales |
|---|---|---|---|
| **0** | FDW58 | OUT049 | 1666.09792 |
| **1** | FDW14 | OUT017 | 1155.76222 |
| **2** | NCN55 | OUT010 | 1047.43656 |
| **3** | FDQ58 | OUT017 | 1766.50056 |
| **4** | FDY38 | OUT027 | 6107.31682 |

In [78]:
```
Prediction.to_csv('Predictions.csv', index=False)
```

`Awesome...`

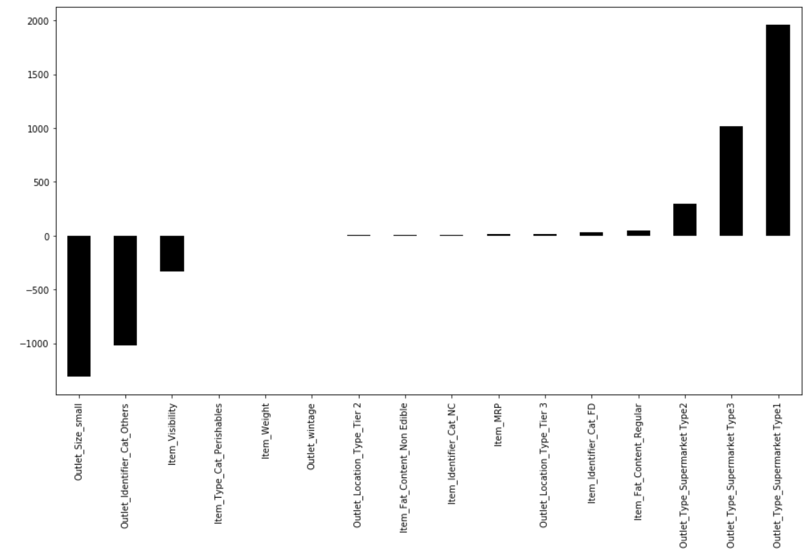
## Your score for this submission is : 1247.3659110049714.

This error seems good to be considered.





In [79]:
```
pd.Series(LR.coef_, X.columns).sort_values().plot(kind='bar', color='black')
plt.show()
```




Here we see the negative slope of the features. Hence we prefer doing `Regularization` along the data to avoid `Overfitting` .

## Let's check the quality of the model along L1 and L2 Regularization:

In [80]:
```
lasso = Lasso(alpha=1.0, normalize=True)
train_predicted = lasso.fit(xtrain,ytrain).predict(xtrain)
test_predicted = lasso.fit(xtrain,ytrain).predict(xtest)

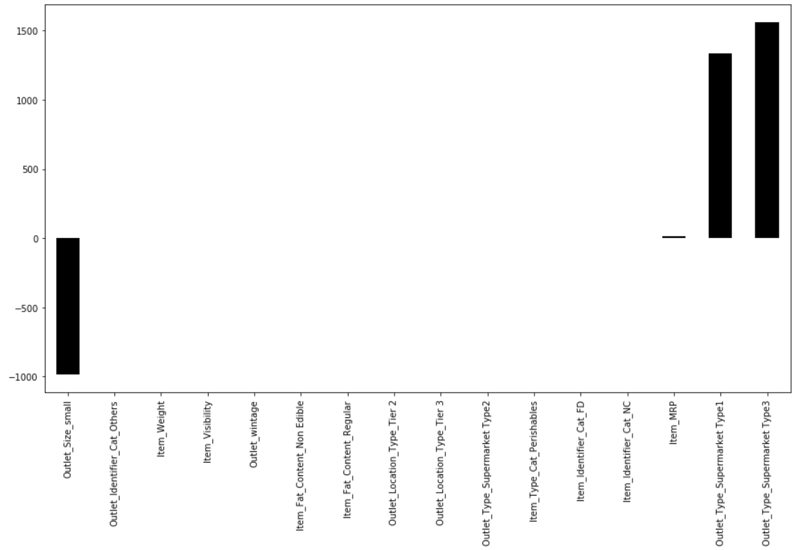
print('Train_RMSE :', np.sqrt(mean_squared_error(ytrain, train_predicted)))
print('Test_RMSE :', np.sqrt(mean_squared_error(ytest, test_predicted)))
```

```
Train_RMSE : 1169.5031210990107
Test_RMSE : 1090.3083445498125
```

`It is quite amazing that our model gives a good accuracy to our unknown data, which means we are avoiding overfitting with a good predicting values.` **But still we are facing around $1k of error, but that seems okay.**

### Let's see how our features are contributing to our model?

In [81]:
```
pd.Series(lasso.coef_, X.columns).sort_values().plot(kind='bar', color= 'bla
plt.show()
```




**Still we can observe that not more featurs are contributing in the prediction to the model. Indeed Lasso is making the coefficients so negligible who seem to be not used as doubt. That is the reason we see here a giant variation.**

## Let's try something else:

In [82]:
```
ridge = Ridge(alpha=1.0, normalize=True)
train_predicted = ridge.fit(xtrain,ytrain).predict(xtrain)
test_predicted = ridge.fit(xtrain,ytrain).predict(xtest)

print('Train_RMSE :', round(np.sqrt(mean_squared_error(ytrain, train_predict
print('Test_RMSE :', round(np.sqrt(mean_squared_error(ytest, test_predicted)
```

```
Train_RMSE : 1340.05
Test_RMSE : 1262.4022
```

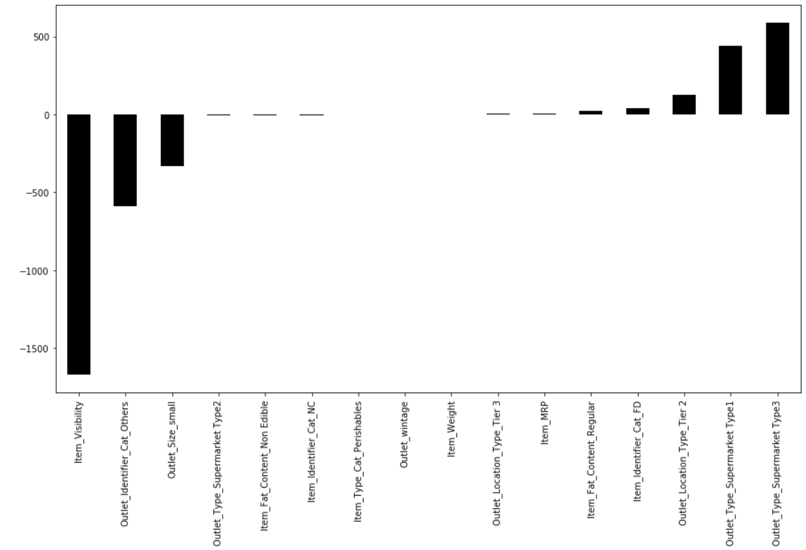
Same output but different `RMSE` .

**Let's see the contribution:**





In [83]:
```
pd.Series(ridge.coef_, X.columns).sort_values().plot(kind='bar', color='blac
plt.show()
```




**Still we face a little high error, but thats absolutely okay, because Lasso did not consider all necessery attributes but Ridge does not mak coefficient's value zero at all. Though we are getting a good prediction with the huge contribution in the model for the predictions.**

---

`The game of alpha`

To get a good fit with accuracy as well as best features we make some models to be experimented. All mathematics behind alpha is just the coefficient which eventually deals with independent varaibles. Alpha value affects the model accuracy.

**Let's observe different alpha for best fit model:**

In [84]:
```python
def lasso_coef(alpha):
    df = pd.DataFrame()
    df['Features'] = X.columns

    for i in alpha:
        lasso = Lasso(alpha=i)
        lasso.fit(xtrain,ytrain)
        column_name = 'Alpha = %f'% i
        df[column_name] = lasso.coef_
    return df
```

In [85]:
```python
lasso_coef([0.1, 0.01, 0.001, 0.5, 0.99, 1, 2, 5, 10])
```

Out[85]:

| | Features | Alpha = 0.100000 | Alpha = 0.010000 | Alpha = 0.001000 | Alpha = 0.500000 | Alpha = 0.99000 |
|---|---|---|---|---|---|---|
| 0 | Item_Weight | -1.219242 | -1.241792e+00 | -1.244130 | -1.122258 | -1.05445 |
| 1 | Item_Visibility | -294.383753 | -3.308086e+02 | -334.455191 | -131.579388 | -0.00000 |
| 2 | Item_MRP | 15.638303 | 1.563772e+01 | 15.637654 | 15.640739 | 15.64205 |
| 3 | Outlet_wintage | -0.756271 | -6.807401e-01 | -0.673170 | -0.752548 | -0.81891 |
| 4 | Item_Fat_Content_Non Edible | 0.000007 | 2.070690e-07 | -41.465849 | 0.720556 | 0.00000 |
| 5 | Item_Fat_Content_Regular | 45.545178 | 4.598600e+01 | 46.030500 | 43.645727 | 41.58283 |
| 6 | Outlet_Size_small | -1557.661093 | -1.072638e+03 | -1024.182017 | -1611.275996 | -1610.05551 |
| 7 | Outlet_Location_Type_Tier 2 | 6.548277 | 7.801031e+00 | 7.926860 | 3.992542 | 0.30627 |
| 8 | Outlet_Location_Type_Tier 3 | 15.246074 | 1.580058e+01 | 15.855914 | 11.534133 | 7.11646 |
| 9 | Outlet_Type_Supermarket Type1 | 1958.619638 | 1.958779e+03 | 1958.795543 | 1958.542570 | 1955.46521 |
| 10 | Outlet_Type_Supermarket Type2 | 50.356535 | 5.371664e+02 | 585.801984 | 0.000000 | 0.00000 |
| 11 | Outlet_Type_Supermarket Type3 | 1792.518587 | 2.277421e+03 | 2325.951866 | 1738.351002 | 1736.09204 |
| 12 | Item_Type_Cat_Perishables | -5.383671 | -5.402778e+00 | -5.397356 | -5.256148 | -3.28964 |
| 13 | Outlet_Identifier_Cat_Others | -0.097573 | -1.007011e-01 | -0.014862 | -0.083900 | -0.07686 |
| 14 | Item_Identifier_Cat_FD | 30.157715 | 3.219951e+01 | 32.426103 | 20.958144 | 15.39447 |
| 15 | Item_Identifier_Cat_NC | 13.939266 | 1.623012e+01 | 57.949367 | 3.010833 | 0.00000 |

**Here we can obserev that as the value of alpha is getting up the coefficient values going down, ultimately we face lose in those features as well which are crucial for the accuracy of the model to make absolute predictions.**





In [86]:
```python
def rig_coef(alpha):
    df1 = pd.DataFrame()
    df1['Features'] = X.columns

    for i in alpha:
        ridge = Ridge()
        ridge.fit(xtrain,ytrain)
        column_name = 'Alpha = %f'% i
        df1[column_name] = ridge.coef_

    return df1
```

In [87]:
```python
rig_coef([0.1, 0.01, 0.5, 0.99, 1, 2, 5, 10,20,30])
```

Out[87]:

| | Features | Alpha = 0.100000 | Alpha = 0.010000 | Alpha = 0.500000 | Alpha = 0.990000 | Alpha = 1.000000 |
|---|---|---|---|---|---|---|
| 0 | Item_Weight | -1.240722 | -1.240722 | -1.240722 | -1.240722 | -1.240722 |
| 1 | Item_Visibility | -323.058811 | -323.058811 | -323.058811 | -323.058811 | -323.058811 |
| 2 | Item_MRP | 15.637788 | 15.637788 | 15.637788 | 15.637788 | 15.637788 |
| 3 | Outlet_wintage | -0.717466 | -0.717466 | -0.717466 | -0.717466 | -0.717466 |
| 4 | Item_Fat_Content_Non Edible | 8.187175 | 8.187175 | 8.187175 | 8.187175 | 8.187175 |
| 5 | Item_Fat_Content_Regular | 45.971392 | 45.971392 | 45.971392 | 45.971392 | 45.971392 |
| 6 | Outlet_Size_small | -1311.225461 | -1311.225461 | -1311.225461 | -1311.225461 | -1311.225461 |
| 7 | Outlet_Location_Type_Tier 2 | 8.543016 | 8.543016 | 8.543016 | 8.543016 | 8.543016 |
| 8 | Outlet_Location_Type_Tier 3 | 16.411699 | 16.411699 | 16.411699 | 16.411699 | 16.411699 |
| 9 | Outlet_Type_Supermarket Type1 | 1954.608765 | 1954.608765 | 1954.608765 | 1954.608765 | 1954.608765 |
| 10 | Outlet_Type_Supermarket Type2 | 294.042789 | 294.042789 | 294.042789 | 294.042789 | 294.042789 |
| 11 | Outlet_Type_Supermarket Type3 | 1017.182673 | 1017.182673 | 1017.182673 | 1017.182673 | 1017.182673 |
| 12 | Item_Type_Cat_Perishables | -5.467649 | -5.467649 | -5.467649 | -5.467649 | -5.467649 |
| 13 | Outlet_Identifier_Cat_Others | -1017.182673 | -1017.182673 | -1017.182673 | -1017.182673 | -1017.182673 |
| 14 | Item_Identifier_Cat_FD | 32.316586 | 32.316586 | 32.316586 | 32.316586 | 32.316586 |
| 15 | Item_Identifier_Cat_NC | 8.187175 | 8.187175 | 8.187175 | 8.187175 | 8.187175 |

**The opposite situation we are observing:** As the alpha values are increasing we do not see the coefficient values descreasing, which seems to be good for many attributes.

## Let's observe how the errors are getting affected by alpha variation:

In [88]:
```
alpha = []
R2 = []
Train_RMSE = []
Test_RMSE = []

alphas = [0.1, 0.01, 0.02, 0.0005, 0.09, 0.001, 0.5, 0.99, 1, 2, 5, 10]

for i in alphas:
    alpha.append(i)

    lasso = Lasso(alpha=i, normalize=True)
    ypred_train = lasso.fit(xtrain,ytrain).predict(xtrain)
    ypred_test = lasso.fit(xtrain,ytrain).predict(xtest)
    Train_RMSE.append(round(np.sqrt(mean_squared_error(ytrain, ypred_train))
    Test_RMSE.append(round(np.sqrt(mean_squared_error(ytest, ypred_test)),4)
    R2.append(r2_score(ytest, ypred_test))
```

In [89]:
```
df2 = pd.DataFrame({'Alpha':alpha, 'R2':R2, 'Train_RMSE':Train_RMSE, 'Test_R
df2.sort_values(by='Alpha', ascending=True)
```

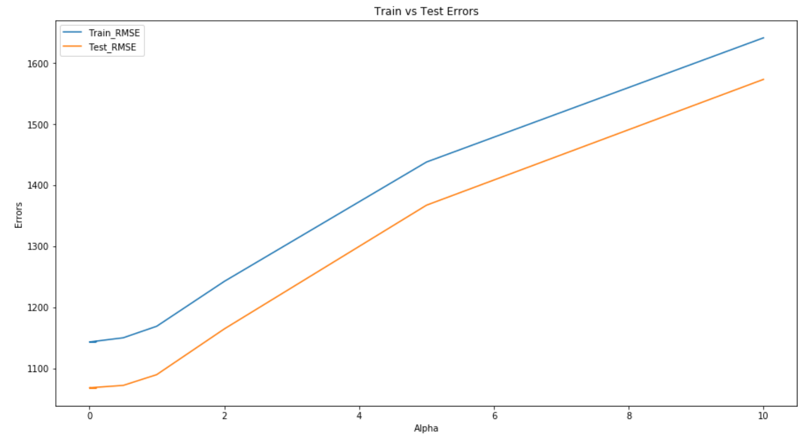
Out[89]:

| | Alpha | R2 | Train_RMSE | Test_RMSE |
|---|---|---|---|---|
| **3** | 0.0005 | 0.579877 | 1143.5892 | 1068.5884 |
| **5** | 0.0010 | 0.579885 | 1143.5892 | 1068.5782 |
| **1** | 0.0100 | 0.580007 | 1143.5964 | 1068.4227 |
| **2** | 0.0200 | 0.580122 | 1143.6165 | 1068.2769 |
| **4** | 0.0900 | 0.580357 | 1143.8756 | 1067.9780 |
| **0** | 0.1000 | 0.580358 | 1143.9325 | 1067.9766 |
| **6** | 0.5000 | 0.576838 | 1150.4027 | 1072.4466 |
| **7** | 0.9900 | 0.563020 | 1169.0018 | 1089.8161 |
| **8** | 1.0000 | 0.562625 | 1169.5031 | 1090.3083 |
| **9** | 2.0000 | 0.500696 | 1242.7682 | 1164.9431 |
| **10** | 5.0000 | 0.311961 | 1438.1595 | 1367.5047 |
| **11** | 10.0000 | 0.089069 | 1641.5099 | 1573.4940 |





In [90]:
```
plt.plot(df2.Alpha, df2.Train_RMSE, label='Train_RMSE')
plt.plot(df2.Alpha, df2.Test_RMSE, label='Test_RMSE')
plt.xlabel('Alpha') ; plt.ylabel('Errors') ; plt.title('Train vs Test Errors
plt.legend()
plt.show()
```



In [91]:
```
alpha1 = []
R2_1 = []
RMSE_Train = []
RMSE_Test = []

alphas = [0.1, 0.01, 0.02, 0.0005, 0.09, 0.001, 0.5, 0.99, 1, 2, 5, 10]

for i in alphas:
    alpha1.append(i)

    ridge = Ridge(alpha=i, normalize=True)
    pred_train = ridge.fit(xtrain,ytrain).predict(xtrain)
    pred_test = ridge.fit(xtrain,ytrain).predict(xtest)
    RMSE_Train.append(round(np.sqrt(mean_squared_error(ytrain,pred_train)),4
    RMSE_Test.append(round(np.sqrt(mean_squared_error(ytest,pred_test)),4))
    R2_1.append(r2_score(ytest, pred_test))
```

In [92]:
```
df3 = pd.DataFrame({'Alpha':alpha1, 'R2':R2_1, 'Train_RMSE':RMSE_Train, 'Tes
df3.sort_values(by='Alpha', ascending=True)
```

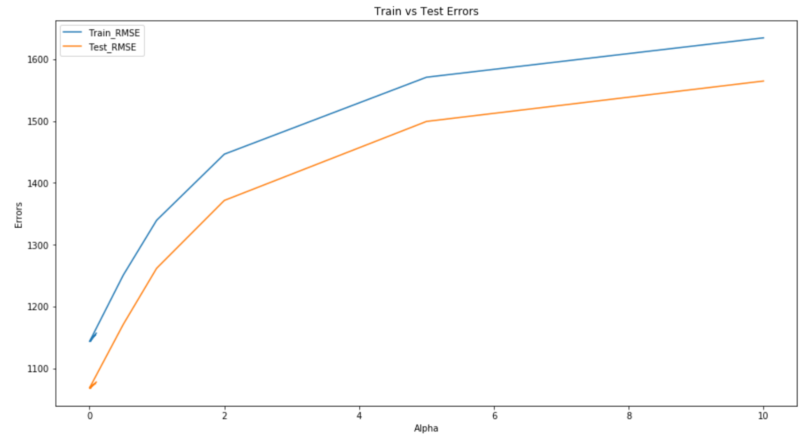
Out[92]:

| | Alpha | R2 | Train_RMSE | Test_RMSE |
|---|---|---|---|---|
| **3** | 0.0005 | 0.579898 | 1143.5897 | 1068.5621 |
| **5** | 0.0010 | 0.579925 | 1143.5914 | 1068.5271 |
| **1** | 0.0100 | 0.580245 | 1143.8016 | 1068.1206 |
| **2** | 0.0200 | 0.580253 | 1144.3889 | 1068.1107 |
| **4** | 0.0900 | 0.573944 | 1154.9060 | 1076.1082 |
| **0** | 0.1000 | 0.572481 | 1156.9575 | 1077.9531 |
| **6** | 0.5000 | 0.495443 | 1250.7221 | 1171.0554 |
| **7** | 0.9900 | 0.415062 | 1338.5759 | 1260.8906 |
| **8** | 1.0000 | 0.413658 | 1340.0500 | 1262.4022 |
| **9** | 2.0000 | 0.307773 | 1446.4829 | 1371.6606 |
| **10** | 5.0000 | 0.172790 | 1570.9349 | 1499.4435 |
| **11** | 10.0000 | 0.099102 | 1634.6735 | 1564.8044 |





In [93]:
```
plt.plot(df3.Alpha, df3.Train_RMSE, label='Train_RMSE')
plt.plot(df3.Alpha, df3.Test_RMSE, label='Test_RMSE')
plt.xlabel('Alpha') ; plt.ylabel('Errors') ; plt.title('Train vs Test Errors
plt.legend()
plt.show()
```




## Summary:

- If we increase alpha value along the Lasso Regularization, we can obsreve that R_squared values are going down which is affecting model accuracy, thats why we see increase in errors.
- If we increase alpha value along the Ridge Regularization, we are observing the same phenomenon. But here we care of some accuracy.

```
Hence we recommend using Ridge Regularization for nD regression.
```

Where n = 3,4,5....

### A bit curriculum for Cross Validation with Lasso and Ridge

## Regularization:

In [94]:
```
from sklearn.linear_model import LassoCV, RidgeCV, ElasticNet
from sklearn.metrics import mean_squared_error
```

In [95]:
```
lassocv = LassoCV(cv=5, normalize=True)

pred_train = lassocv.fit(xtrain,ytrain).predict(xtrain)
pred_test = lassocv.fit(xtrain,ytrain).predict(xtest)

print('Train_RMSE :', np.sqrt(mean_squared_error(ytrain, pred_train)))
print('Test_RMSE :', np.sqrt(mean_squared_error(ytest, pred_test)))
print('\nAlpha value selected by the model :',round(lassocv.alpha_,1))
```

```
Train_RMSE : 1143.9472249977373
Test_RMSE : 1067.9773521827715

Alpha value selected by the model : 0.1
```

In [96]:
```
ridgcv = RidgeCV(cv=5, normalize=True)

pred_train = ridgcv.fit(xtrain,ytrain).predict(xtrain)
pred_test = ridgcv.fit(xtrain,ytrain).predict(xtest)

print('Train_RMSE :', np.sqrt(mean_squared_error(ytrain, pred_train)))
print('Test_RMSE :', np.sqrt(mean_squared_error(ytest, pred_test)))
print('\nAlpha value selected by the model :',ridgcv.alpha_)
```

```
Train_RMSE : 1156.9574988996396
Test_RMSE : 1077.9531244934415

Alpha value selected by the model : 0.1
```

In [97]:
```
enet = ElasticNet(alpha=0.00001, normalize=True, max_iter=10000)

pred_train = enet.fit(xtrain,ytrain).predict(xtrain)
pred_test = enet.fit(xtrain,ytrain).predict(xtest)

print('Train_RMSE :', np.sqrt(mean_squared_error(ytrain, pred_train)))
print('Test_RMSE :', np.sqrt(mean_squared_error(ytest, pred_test)))
```

```
Train_RMSE : 1145.7328403732351
Test_RMSE : 1068.743128774703
```

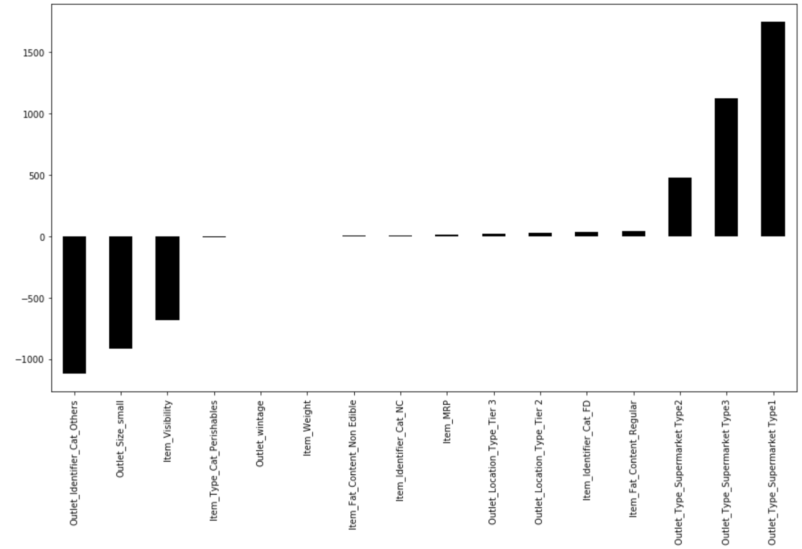
Whether we put `cv = 5,10,100` we se no change in our values. Ridge was giving a little high error compared to the Lasso. Eventually similar case we are observing here in `RidgeCV` and `LassoCV` . But we see a bit difference in the values of the errors.

From `ElasticNet` model, we observe the sme scenario, but the point is alpha. from `0.00001` onwards it gives the similar error. And the errors are followed by Lasso and Ridge as well. But still the value of alpha matters.





In [98]:
```
pd.Series(enet.coef_, X.columns).sort_values().plot(kind='bar', color='black
plt.show()
```




Here we see the property of ElasticNet followed with Ridge Regresssion. Even this model does not ignore the features with low contribution.

`So we say, Ridge Regularization is quite good for less error and best selection of features.`

---

A question always in the mind of selection of best alpha for the model. Let's resolve this issue:

In [99]:
```
from sklearn.model_selection import GridSearchCV
```

In [100]:
```
alpha = [{'alpha':[0.1, 0.01, 0.02, 0.0005, 0.09, 0.001, 0.5, 0.99, 1, 2, 5,

gridcv = GridSearchCV(estimator= Ridge(normalize=True), param_grid=alpha , c

gridcv.fit(xtrain, ytrain)
```

Out[100]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=True, random_state=None,
                             solver='auto', tol=0.001),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.1, 0.01, 0.02, 0.0005, 0.09, 0.001, 0.5,
                                    0.99, 1, 2, 5, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [101]:
```
gridcv.best_params_
```

Out[101]:
```
{'alpha': 0.001}
```

## Let's see the alpha value accuracy:

In [102]:
```
ridge = Ridge(alpha=0.001, normalize=True)
train_predicted = ridge.fit(xtrain,ytrain).predict(xtrain)
test_predicted = ridge.fit(xtrain,ytrain).predict(xtest)

print('Train_RMSE :', round(np.sqrt(mean_squared_error(ytrain, train_predict
print('Test_RMSE :', round(np.sqrt(mean_squared_error(ytest, test_predicted)
```

```
Train_RMSE : 1143.5914
Test_RMSE : 1068.5271
```

**Grid model errors:**

- Train_RMSE : 1340.05
- Test_RMSE : 1262.4022

**This is the error we were getting by alpha value what we were issumed, but model selected alpha was more accurate.**

---

### Predicted valaues of Items outlet sales:





In [103]:
```
pd.read_csv('C:/Users/hi/Downloads/b.csv').head(10)
```

Out[103]:

| | Item_Identifier | Outlet_Identifier | Item_Outlet_Sales |
|---|---|---|---|
| **0** | FDW58 | OUT049 | 1942.00544 |
| **1** | FDW14 | OUT017 | 1547.58552 |
| **2** | NCN55 | OUT010 | 897.76472 |
| **3** | FDQ58 | OUT017 | 1965.70792 |
| **4** | FDY38 | OUT027 | 6552.00464 |
| **5** | FDH56 | OUT046 | 2104.86012 |
| **6** | FDL48 | OUT018 | 630.97866 |
| **7** | FDC48 | OUT027 | 1959.04992 |
| **8** | FDN33 | OUT045 | 1050.89872 |
| **9** | FDA36 | OUT017 | 2272.84146 |

---

****