BigMart

November 9, 2022

1 BigMart Project

1.1 Table of Contents

Introduction

1.1.1 Project Description

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 of this data science project 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.

Hypothesis Generation

1. ITEM TYPE Item type holds a lot of significance in determining its price. Price differs by supply and demand which differs in item type. #### 2. ITEM SIZE People with stores far away tends to buy large size items to decrease the times of shopping.This will get us to the third feature. #### 3. STORE PLACE Store place is a great factor also. It determines the class of the buyers and what type of profducts they are interested in. #### 4. OUTLET PLACE High class people tend to buy imported products more than lower classes. #### 5. RETAIL PRICE I think this is one of the greatest factors of determining a product's sale.

Import Packages & Loading Data

[1]:
```
# Import needed packages
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.stats import skew, norm
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import mean_absolute_error
%matplotlib inline

from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgb
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold, cross_val_score
from mlxtend.regressor import StackingCVRegressor
from xgboost import XGBRegressor
```

[2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train['source'] = 'train'
test['source'] = 'test'

data = pd.concat([train, test], ignore_index=True)
data.head()
```

[2]:
```
  Item_Identifier  Item_Weight Item_Fat_Content  Item_Visibility  \
0           FDA15         9.30          Low Fat         0.016047
1           DRC01         5.92          Regular         0.019278
```

```
2          FDN15        17.50          Low Fat          0.016760
3          FDX07        19.20          Regular          0.000000
4          NCD19         8.93          Low Fat          0.000000

               Item_Type  Item_MRP Outlet_Identifier  \
0                  Dairy  249.8092            OUT049
1            Soft Drinks   48.2692            OUT018
2                   Meat  141.6180            OUT049
3  Fruits and Vegetables  182.0950            OUT010
4              Household   53.8614            OUT013

   Outlet_Establishment_Year Outlet_Size Outlet_Location_Type  \
0                       1999      Medium               Tier 1
1                       2009      Medium               Tier 3
2                       1999      Medium               Tier 1
3                       1998         NaN               Tier 3
4                       1987        High               Tier 3

         Outlet_Type  Item_Outlet_Sales source
0  Supermarket Type1          3735.1380  train
1  Supermarket Type2           443.4228  train
2  Supermarket Type1          2097.2700  train
3      Grocery Store           732.3800  train
4  Supermarket Type1           994.7052  train
```

Exploratory Data Analysis

[3]:
```
data.shape
```

[3]: (14204, 13)

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            14204 non-null  object
 1   Item_Weight                11765 non-null  float64
 2   Item_Fat_Content           14204 non-null  object
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object
 7   Outlet_Establishment_Year  14204 non-null  int64
 8   Outlet_Size                10188 non-null  object
 9   Outlet_Location_Type       14204 non-null  object
```

```
 10  Outlet_Type                 14204 non-null  object
 11  Item_Outlet_Sales           8523 non-null   float64
 12  source                      14204 non-null  object
dtypes: float64(4), int64(1), object(8)
memory usage: 1.4+ MB
```

[5]:
```
data.describe()
```

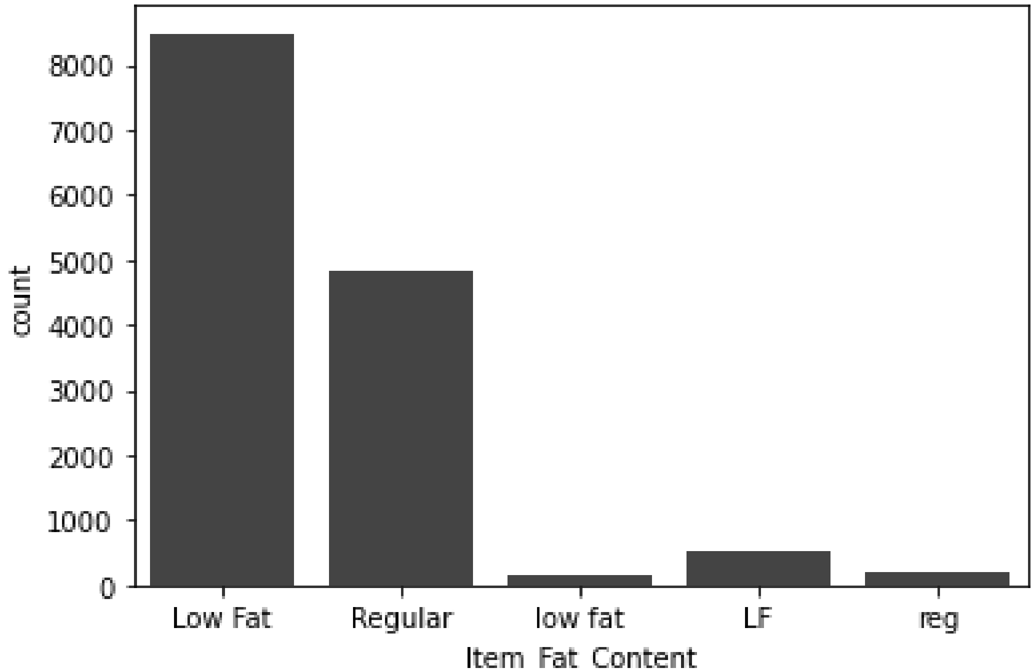
[5]:
```
        Item_Weight  Item_Visibility      Item_MRP  Outlet_Establishment_Year  \
count  11765.000000     14204.000000  14204.000000               14204.000000
mean      12.792854         0.065953    141.004977                1997.830681
std        4.652502         0.051459     62.086938                   8.371664
min        4.555000         0.000000     31.290000                1985.000000
25%        8.710000         0.027036     94.012000                1987.000000
50%       12.600000         0.054021    142.247000                1999.000000
75%       16.750000         0.094037    185.855600                2004.000000
max       21.350000         0.328391    266.888400                2009.000000

       Item_Outlet_Sales
count        8523.000000
mean         2181.288914
std          1706.499616
min            33.290000
25%           834.247400
50%          1794.331000
75%          3101.296400
max         13086.964800
```

[6]:
```
data.nunique()
```

[6]:
```
Item_Identifier               1559
Item_Weight                    415
Item_Fat_Content                 5
Item_Visibility              13006
Item_Type                       16
Item_MRP                      8052
Outlet_Identifier               10
Outlet_Establishment_Year        9
Outlet_Size                      3
Outlet_Location_Type             3
Outlet_Type                      4
Item_Outlet_Sales             3493
source                           2
dtype: int64
```

Univariate Exploration

[7]:
```
sb.countplot(data=data, x='Item_Fat_Content',color='#444444');
plt.show()
```

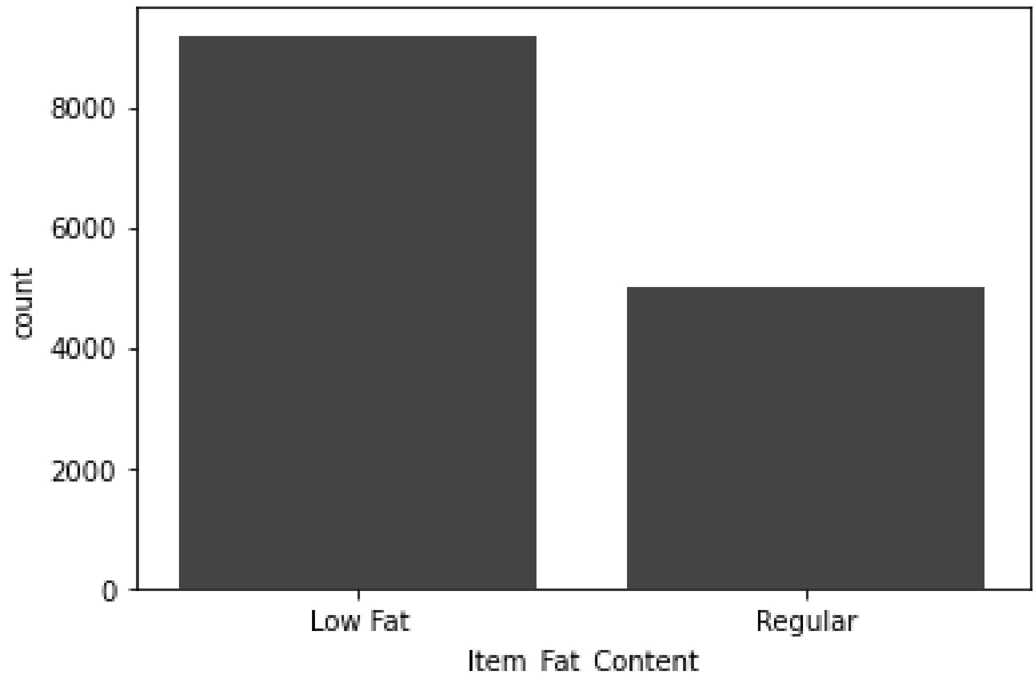


As shown, low fat and regular have a lot of values so let us combine them.

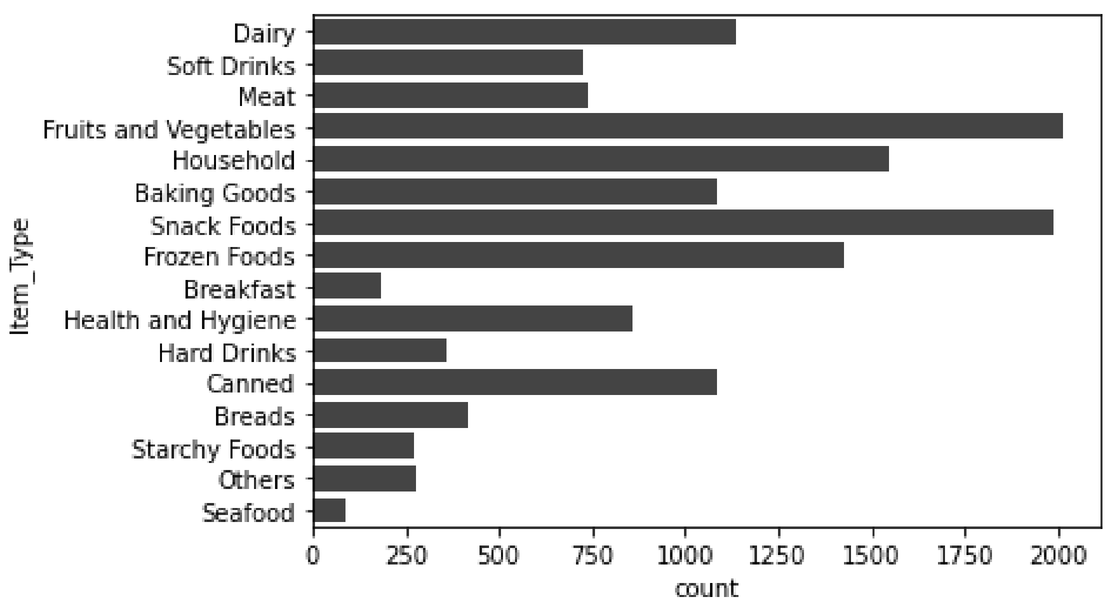
[8]:
```
data['Item_Fat_Content']=data['Item_Fat_Content'].replace('low fat','Low Fat')
data['Item_Fat_Content']=data['Item_Fat_Content'].replace('LF','Low Fat')
data['Item_Fat_Content']=data['Item_Fat_Content'].replace('reg','Regular')
data['Item_Fat_Content'].unique()
```

[8]:
```
array(['Low Fat', 'Regular'], dtype=object)
```

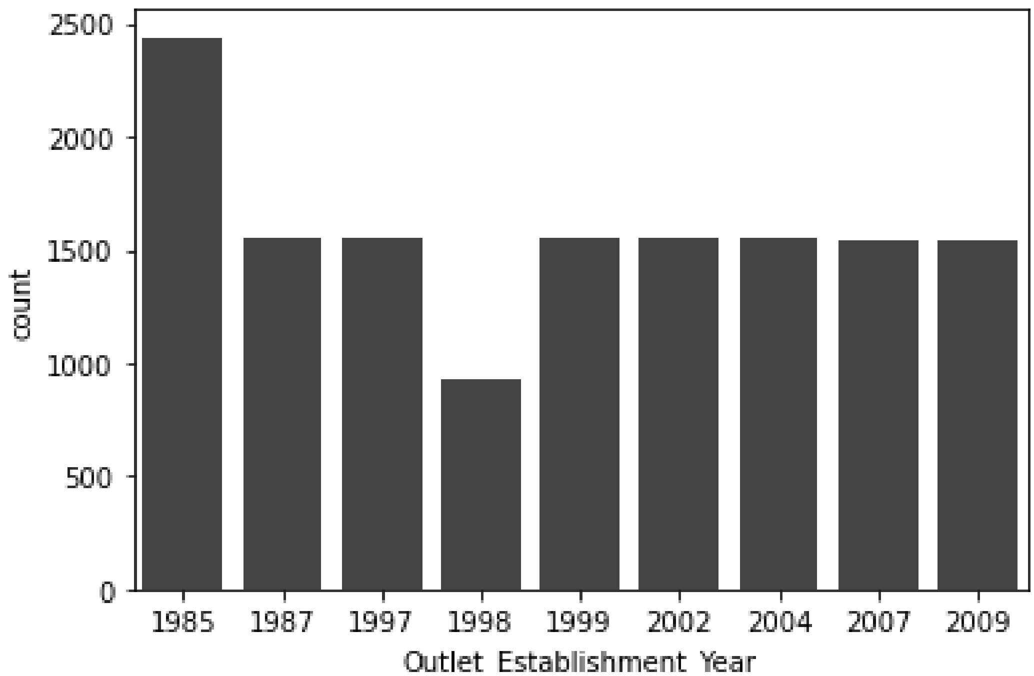
[9]:
```
sb.countplot(data=data, x='Item_Fat_Content',color='#444444');
plt.show()
```



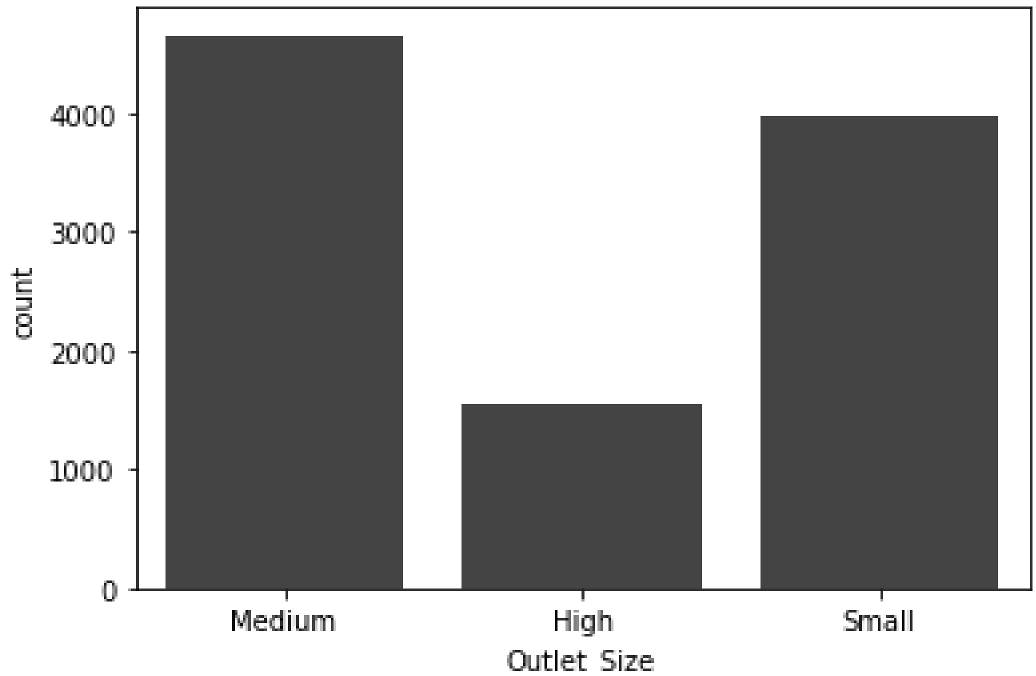
[10]:
```
sb.countplot(data=data, y='Item_Type',color='#444444');
plt.show()
```



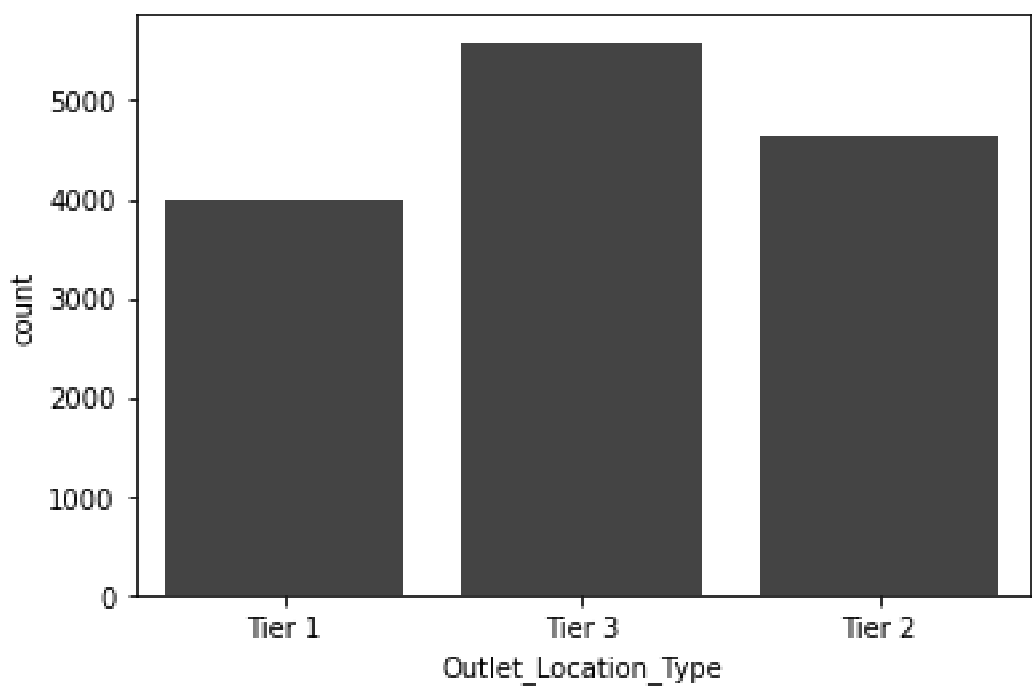
[11]:
```
sb.countplot(data=data, x='Outlet_Establishment_Year',color='#444444');
plt.show()
```



[12]:
```
sb.countplot(data=data, x='Outlet_Size',color='#444444');
plt.show()
```

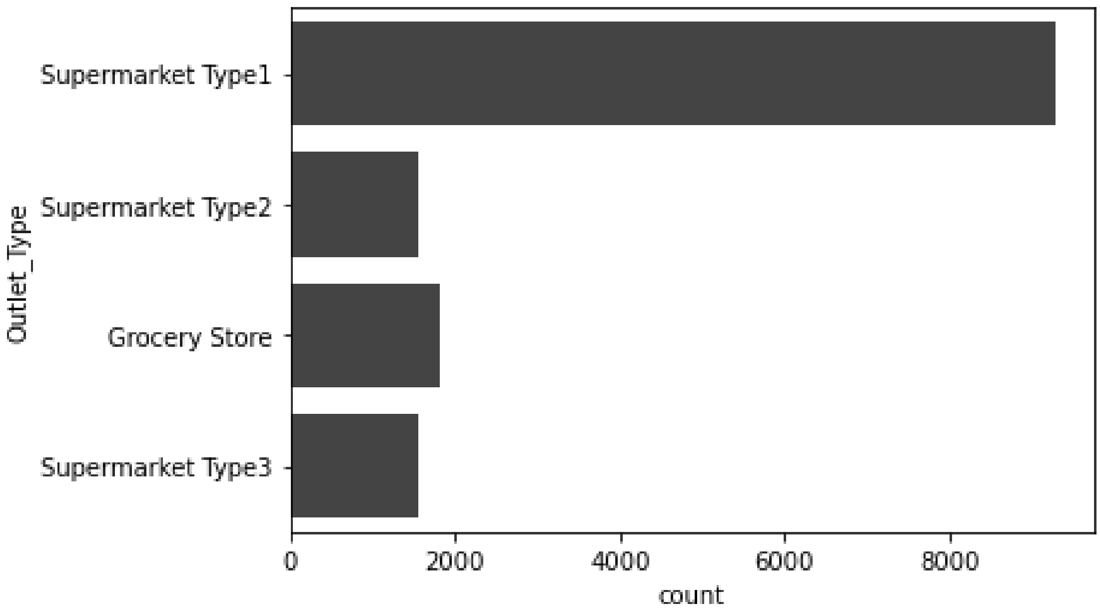


[13]:
```
sb.countplot(data=data, x='Outlet_Location_Type',color='#444444');
plt.show()
```



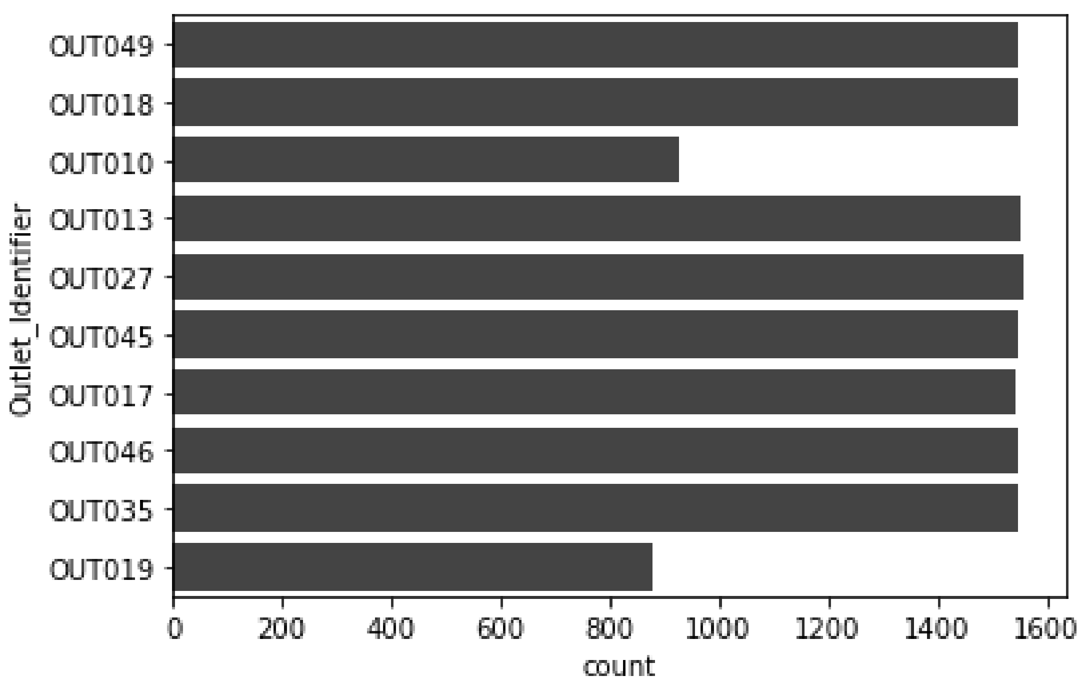
[14]:

```
sb.countplot(data=data, y='Outlet_Type',color='#444444');
plt.show()
```

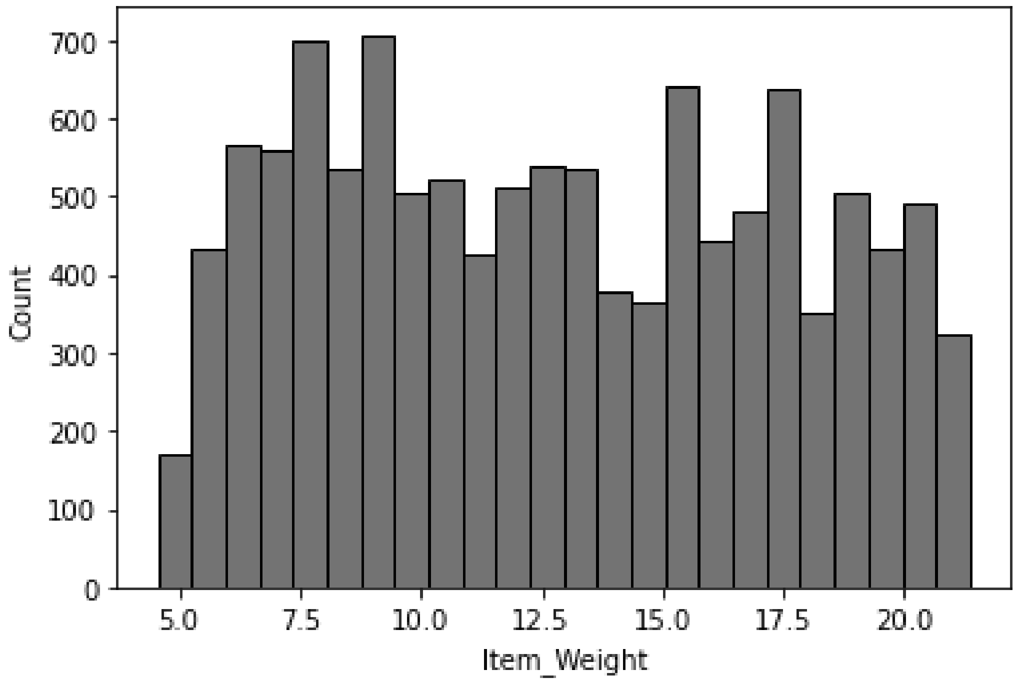


[15]:

```
sb.countplot(data=data, y='Outlet_Identifier',color='#444444');
plt.show()
```

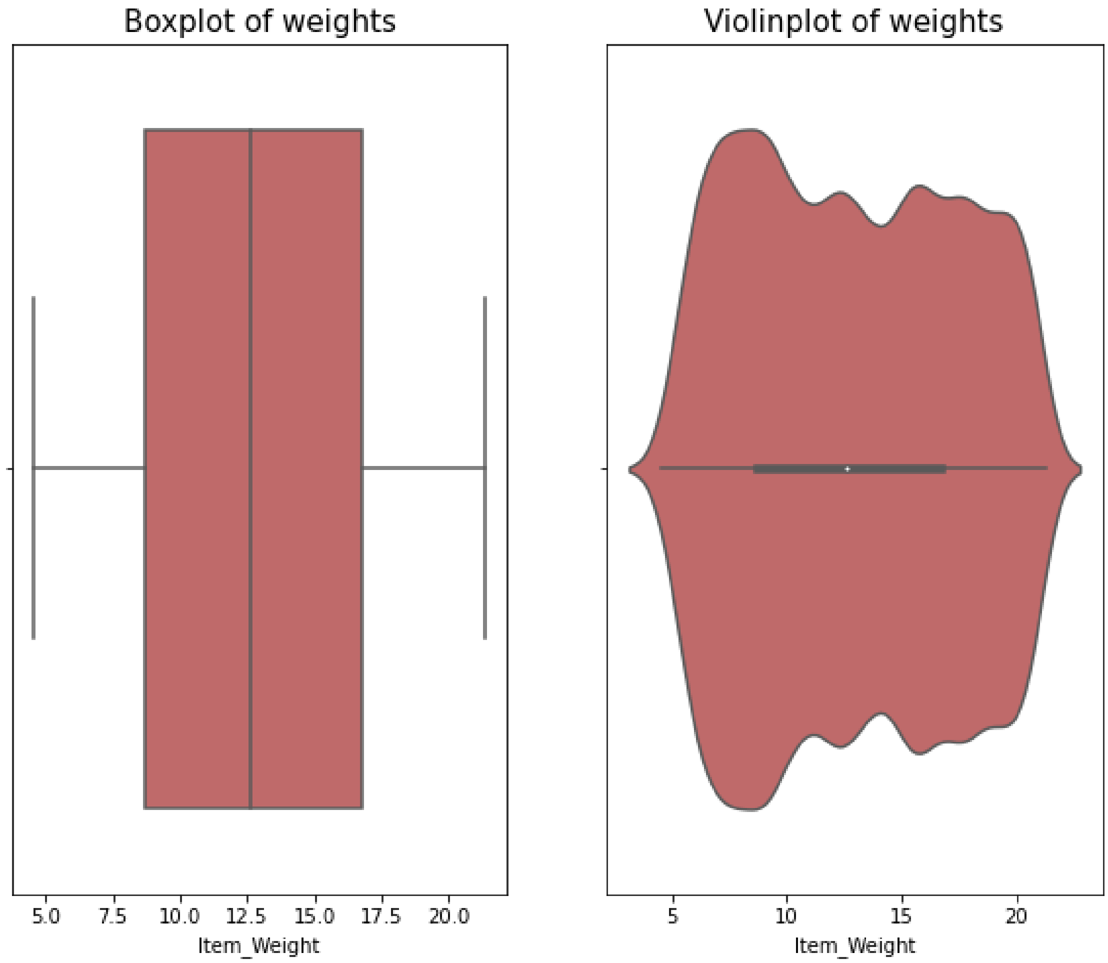


[16]:
```
sb.histplot(x=data['Item_Weight'],color='#444444')
plt.show()
```

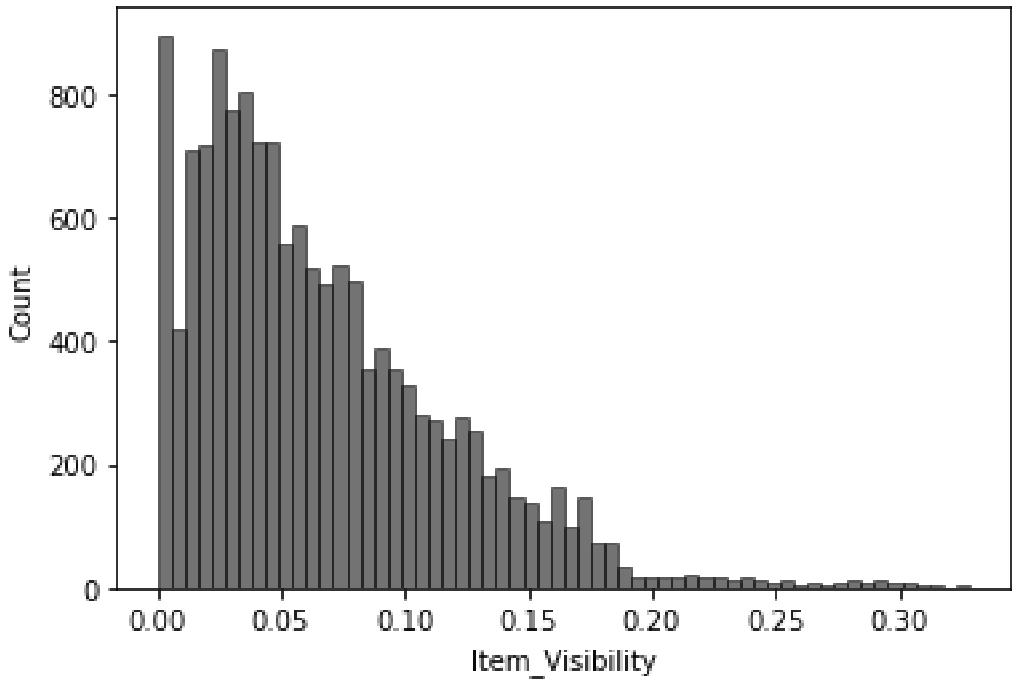


[17]:
```
fig1=plt.figure(figsize=(10,8))
ax1=fig1.add_subplot(121)
sb.boxplot(x = data['Item_Weight'],ax=ax1,color='indianred')
ax1.set_title('Boxplot of weights',size=15)

ax2=fig1.add_subplot(122)
sb.violinplot(x = data['Item_Weight'],ax=ax2,color='indianred')
ax2.set_title('Violinplot of weights',size=15)
```

[17]:
```
Text(0.5, 1.0, 'Violinplot of weights')
```

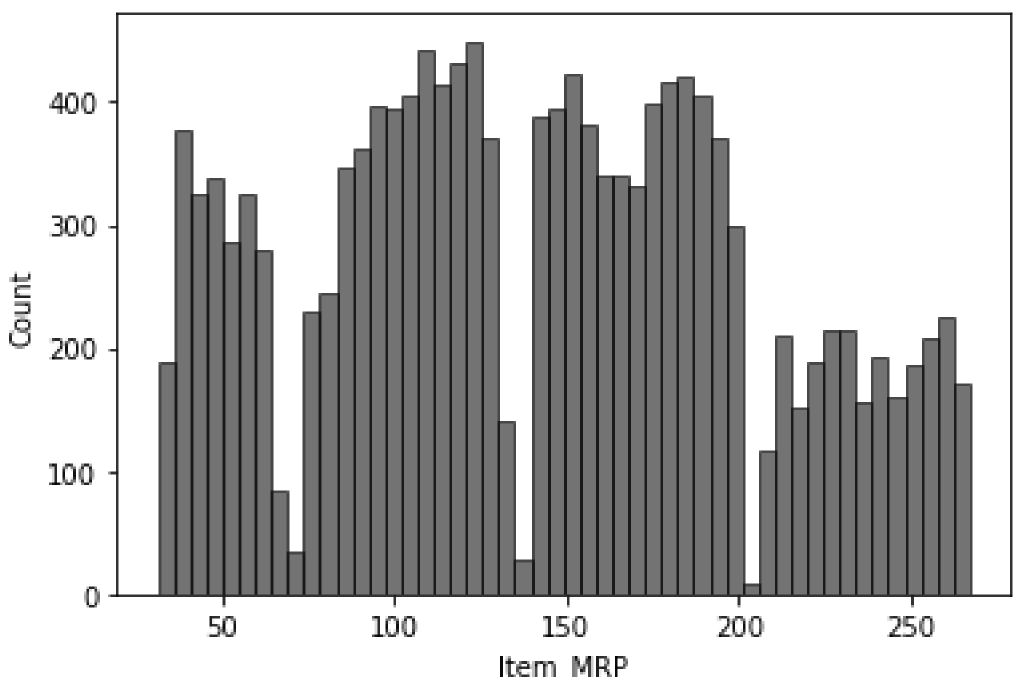


[18]:
```
sb.histplot(x=data['Item_Visibility'],color='#444444')
plt.show()
```



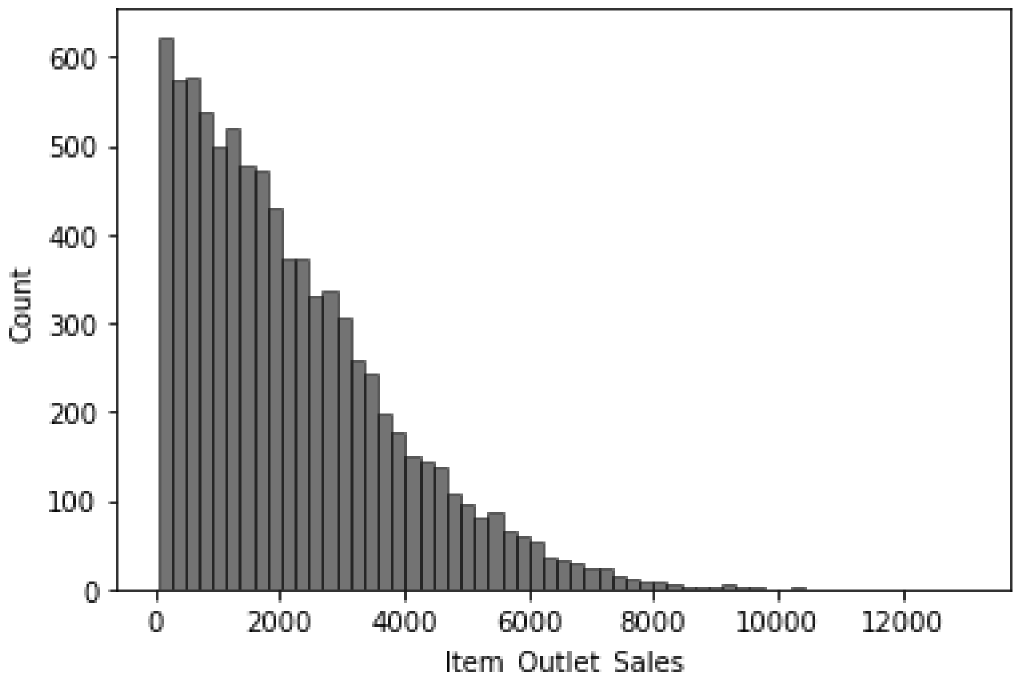
We can see that this a rightly skewed distribution. We could transform it later.

[19]:
```
sb.histplot(x=data['Item_MRP'],color='#444444', bins = 50)
plt.show()
```



It is visible that the data is clustered into four clusters. We could use that in feature engineering.

[20]:
```
sb.histplot(x=data['Item_Outlet_Sales'],color='#444444')
plt.show()
```

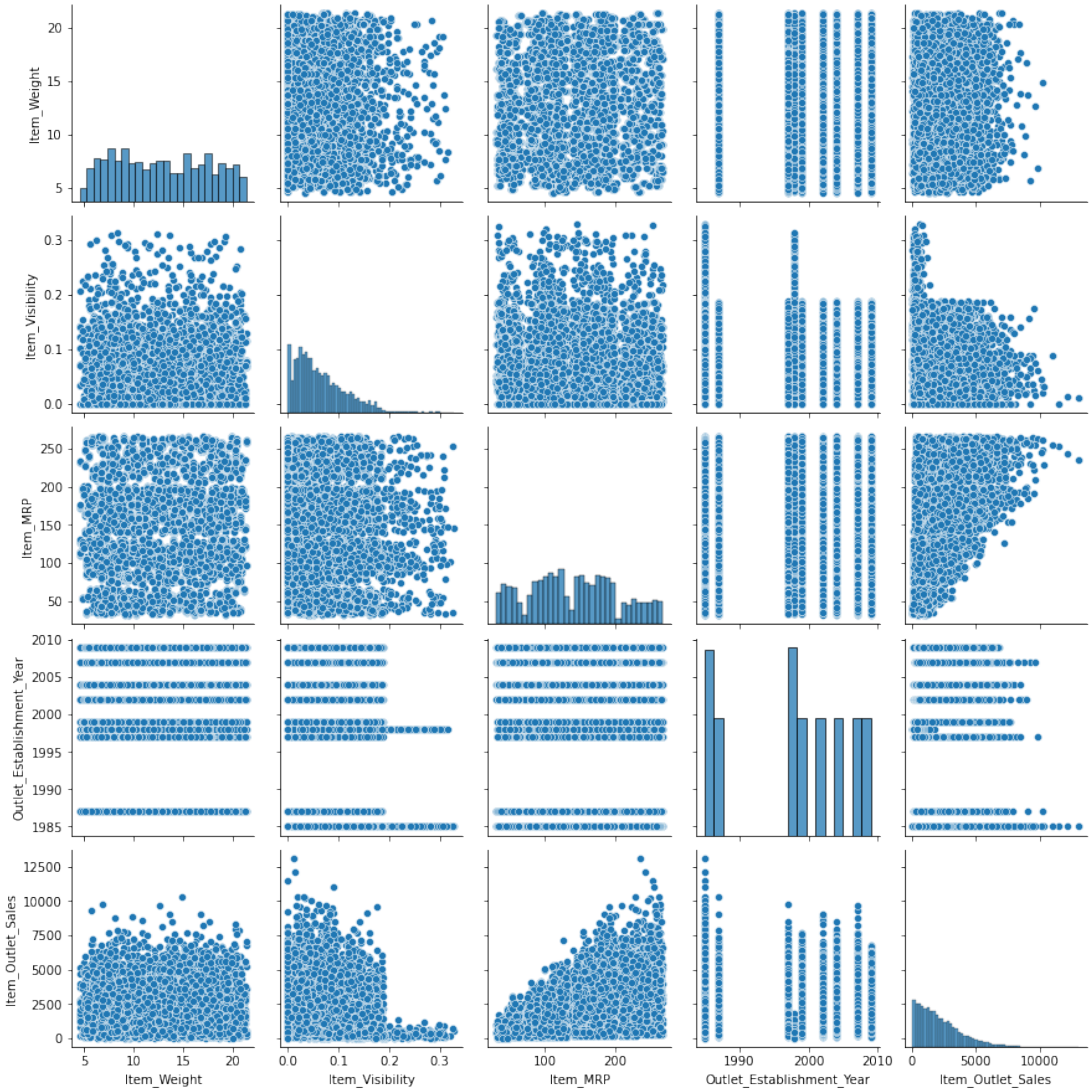


Our target is also skewed to the right

Bivariate Exploration

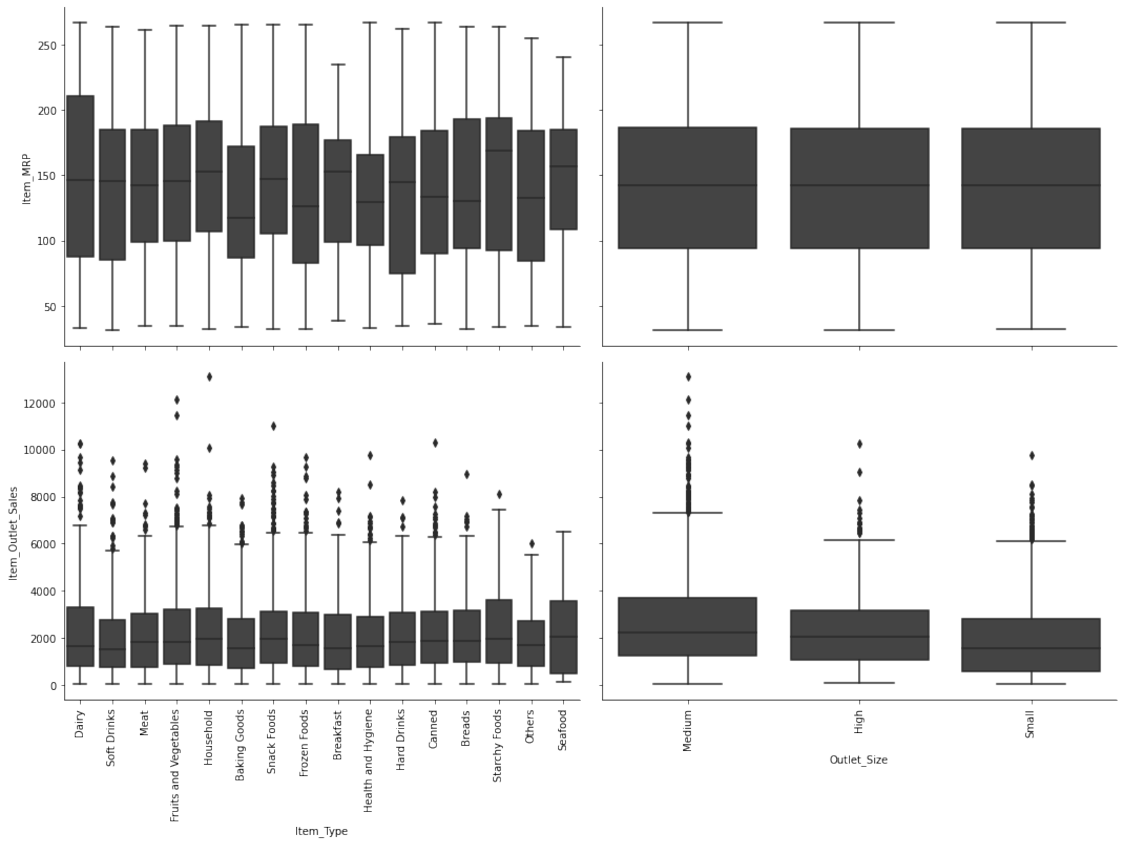
[21]:
```
sb.pairplot(data)
```

[21]: <seaborn.axisgrid.PairGrid at 0x2b829130400>



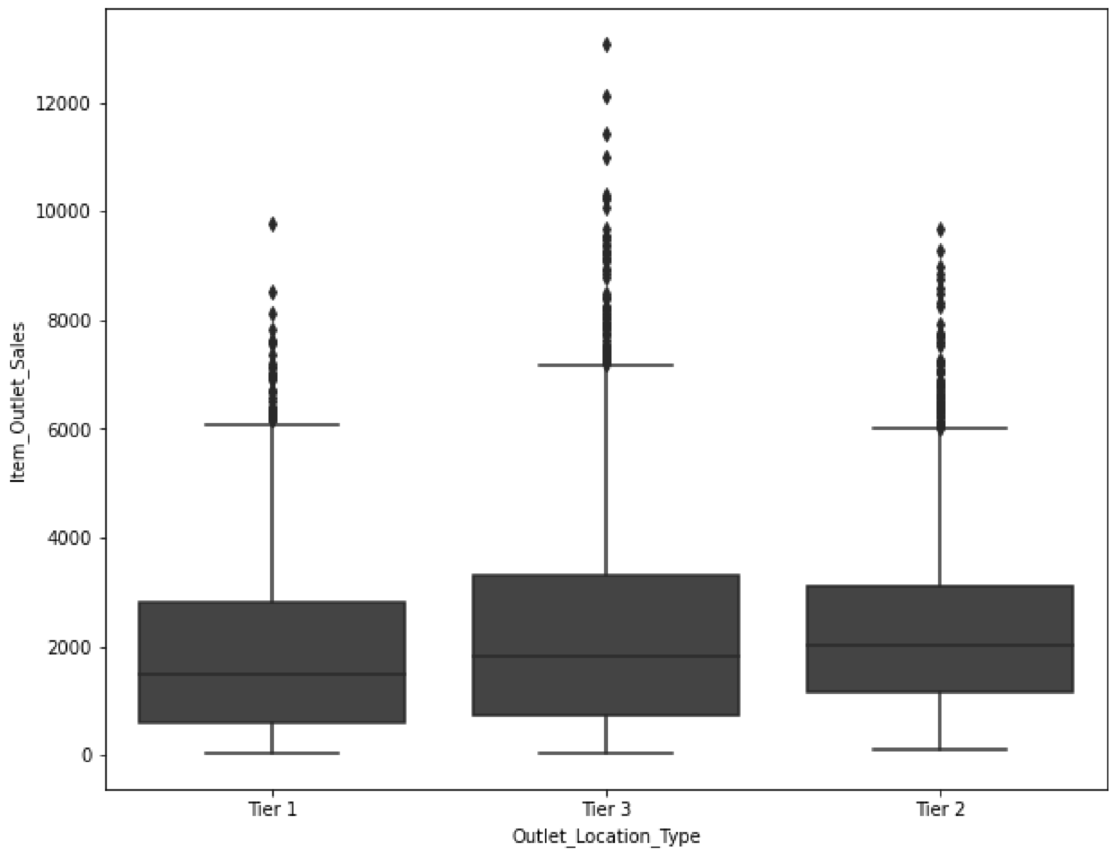
We could infer some points: 1. Item_Visibility is inversely proportional to the sales 2. Item_MRP is directly proportional to the sales

[22]:
```python
g = sb.PairGrid(data, y_vars = ['Item_MRP','Item_Outlet_Sales'], x_vars =␣
↪['Item_Type','Outlet_Size'], height = 5, aspect = 1.5)
g.map(sb.boxplot,color='#444444')

for ax in g.axes.flatten():
    # rotate x axis labels
    ax.set_xticklabels(ax.get_xticklabels(), rotation=90)

plt.show()
```



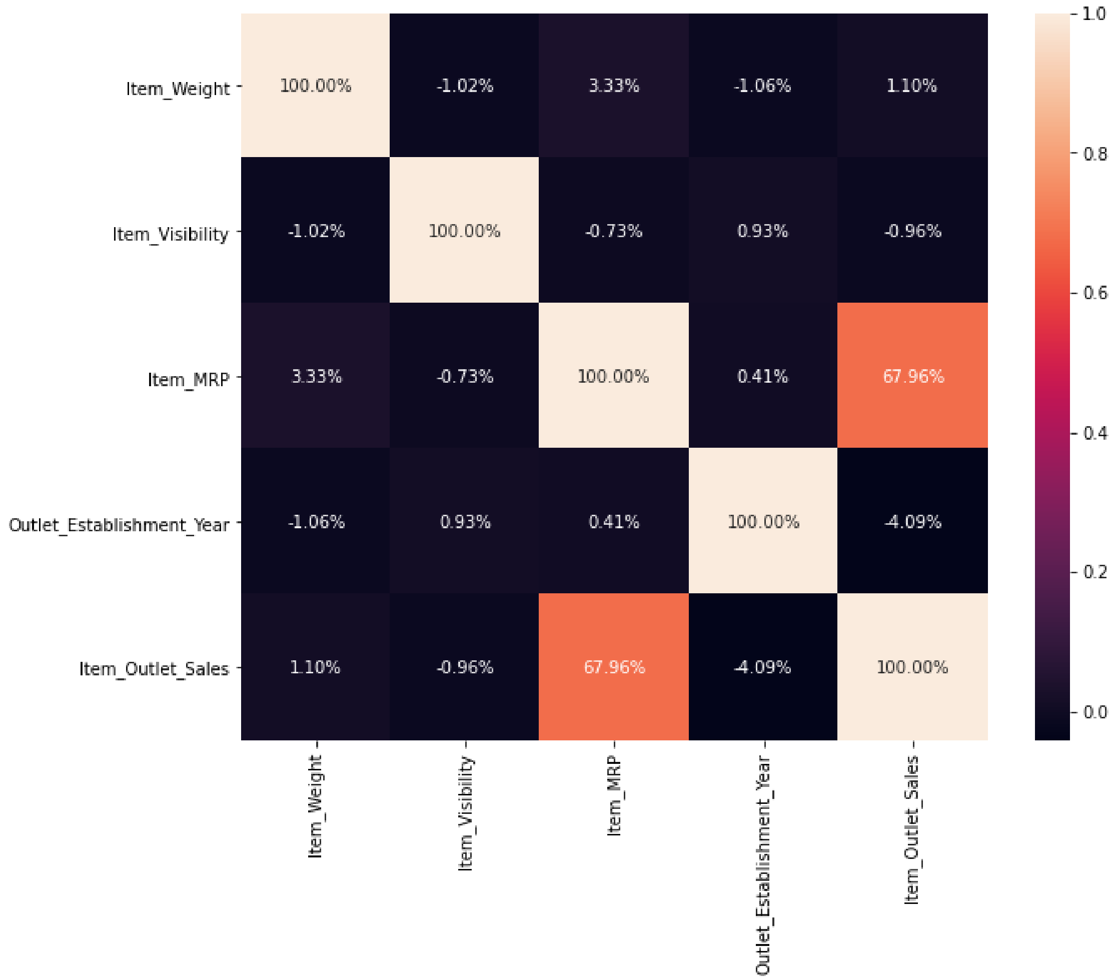
[23]:
```
plt.figure(figsize=(10,8))
sb.boxplot(x = 'Outlet_Location_Type',y = 'Item_Outlet_Sales',data=data, color␣
↪= '#444444')
```

[23]: <AxesSubplot:xlabel='Outlet_Location_Type', ylabel='Item_Outlet_Sales'>



[24]:
```
corrs=data.dropna().corr()
plt.figure(figsize=(10,8))
sb.heatmap(corrs,annot=True,fmt='.2%')
```

[24]: <AxesSubplot:>



Missing Value Treatment

[13]:
```
data.apply(lambda x: sum(x.isnull()))
```

[13]:
```
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
source                          0
```

```
dtype: int64
```

There are a lot of missing values in weight and size features. Weight is a continuous feature. Hence, we will treat it by grouping the data by the item identifier and finding its weight mean. For size feature, We will do the same thing but we will look for the mode.

[14]:
```python
def fast_mode(df, key_cols, value_col):
    """
    Calculate a column mode, by group, ignoring null values.

    Parameters
    ----------
    df : pandas.DataFrame
        DataFrame over which to calcualate the mode.
    key_cols : list of str
        Columns to groupby for calculation of mode.
    value_col : str
        Column for which to calculate the mode.

    Return
    ------
    pandas.DataFrame
        rows for the mode of value_col per key_cols group. If ties,
        returns the one which is sorted first.
    """
    return (df.groupby(key_cols + value_col).size()
                .to_frame('counts').reset_index()
                .sort_values('counts', ascending=False)
                .drop_duplicates(subset=key_cols)).drop(columns='counts')
```

[15]:
```python
group = ['Item_Identifier']
mode = fast_mode(data, group, ['Outlet_Size']).set_index(group)
```

[16]:
```python
def fillMode(row):
    try:
        if math.isnan(row['Outlet_Size']):
            row['Outlet_Size']=mode.loc[row['Item_Identifier']][0]
    except:pass
    return row
data=data.apply(fillMode,axis=1)
data['Outlet_Size'].isnull().sum()
```

[16]: 0

[17]:
```python
means = data.groupby('Item_Identifier').Item_Weight.mean()
```

[18]:
```python
def fillMean(row):
    try:
```

```
        if math.isnan(row['Item_Weight']):
            row['Item_Weight']=means.loc[row['Item_Identifier']]
    except:pass
    return row
data=data.apply(fillMean,axis=1)
data['Item_Weight'].isnull().sum()
```

[18]: 0

Outliers Removal

```
[19]: def check_outliers (df, features):
    Q1 = df[features].quantile(0.25)
    Q3 = df[features].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5*IQR
    upper_bound = Q3 + 1.5*IQR
    q_lower_bound=df[features][(df[features])<lower_bound].count()
    q_upper_bound=df[features][(df[features])>upper_bound].count()
    return lower_bound,upper_bound

def remove_outliers (df, features, lower, upper):
    df = df[(df[features]<=upper)]
    df = df[(df[features]>=lower)]
    return df
```

```
[20]: for i in ['Item_MRP','Item_Weight','Item_Visibility']:
    lower , upper = check_outliers(data,i)
    data = remove_outliers(data, i, lower, upper)
```

```
[21]: data.shape
```

[21]: (13943, 13)

Feature Engineering

1.1.2 Features Tranformation

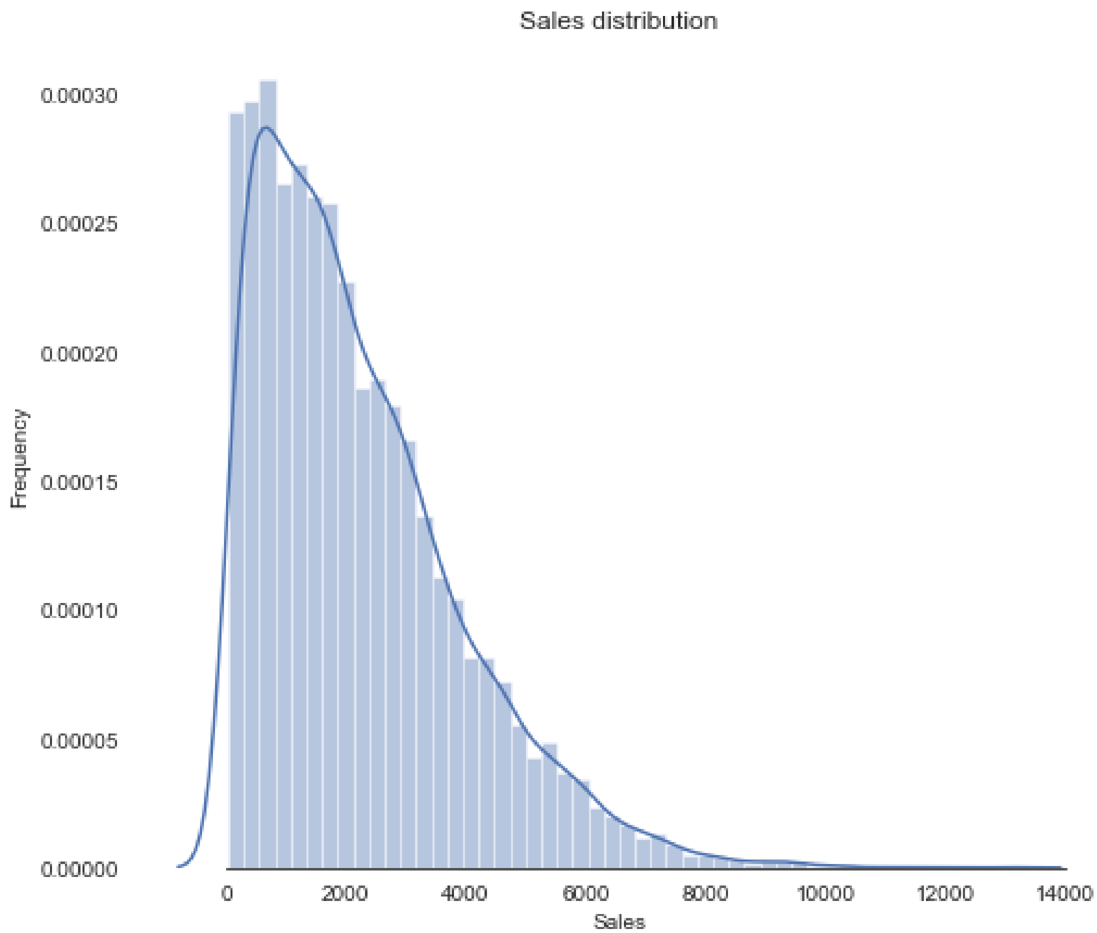
Here we are going to apply transformations to certain continous features in the data in order to meake them more normally distributed. This will help the model.

1. Sales

```
[34]: sb.set_style("white")
sb.set_color_codes(palette='deep')
f, ax = plt.subplots(figsize=(8, 7))
#Check the new distribution
sb.distplot(data['Item_Outlet_Sales'], color="b");
ax.xaxis.grid(False)
```

```
ax.set(ylabel="Frequency")
ax.set(xlabel="Sales")
ax.set(title="Sales distribution")
sb.despine(trim=True, left=True)
plt.show()
```

```
C:\Users\C\conda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



[35]:

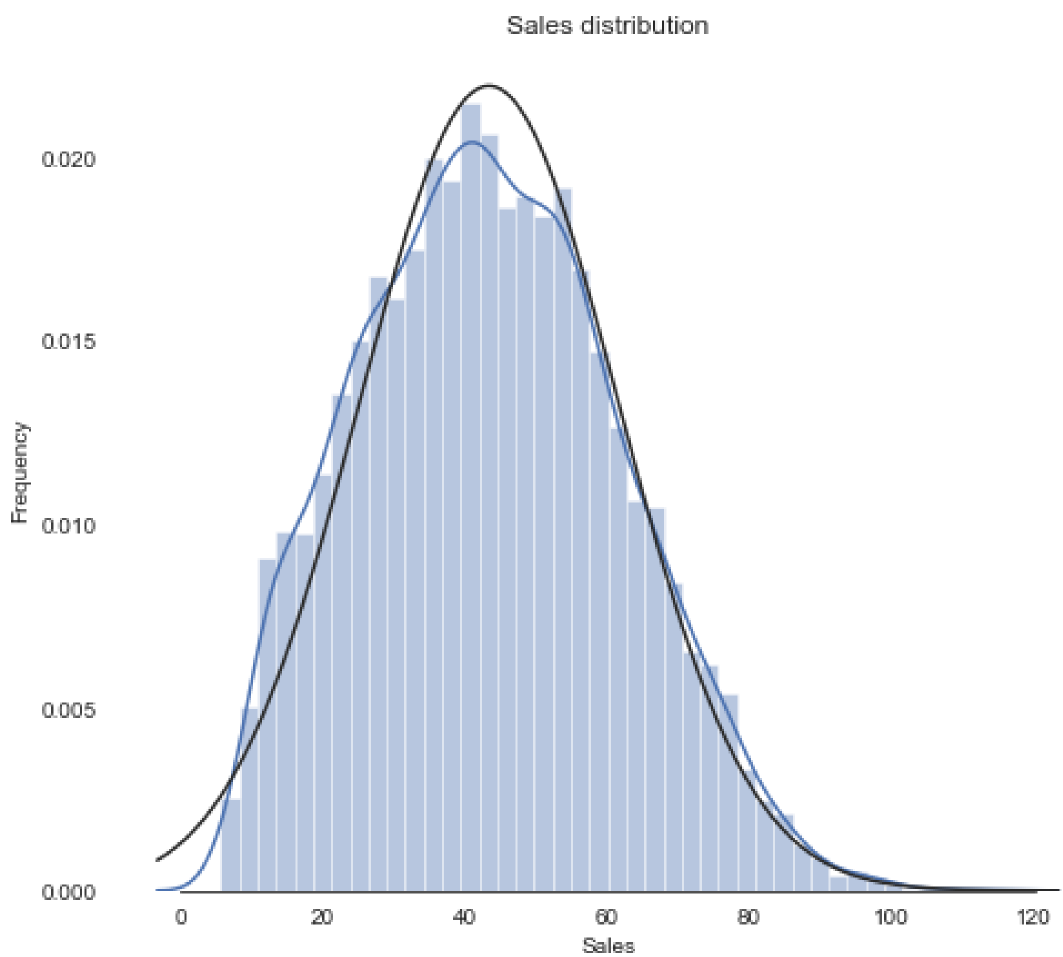
```
# Skew and kurt
print("Skewness: %f" % data['Item_Outlet_Sales'].skew())
print("Kurtosis: %f" % data['Item_Outlet_Sales'].kurt())
```

```
Skewness: 1.170912
Kurtosis: 1.613675
```

[22]:
```
# Square root transformation - sqrt(x)
data['Sqrt_Sales'] = data.Item_Outlet_Sales**(1/2)
```

[37]:
```
sb.set_color_codes(palette='deep')
f, ax = plt.subplots(figsize=(8, 7))
#Check the new distribution
sb.distplot(data['Sqrt_Sales'] , fit=norm, color="b");

# Get the fitted parameters used by the function
ax.xaxis.grid(False)
ax.set(ylabel="Frequency")
ax.set(xlabel="Sales")
ax.set(title="Sales distribution")
sb.despine(trim=True, left=True)

plt.show()
```

```
C:\Users\C\conda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

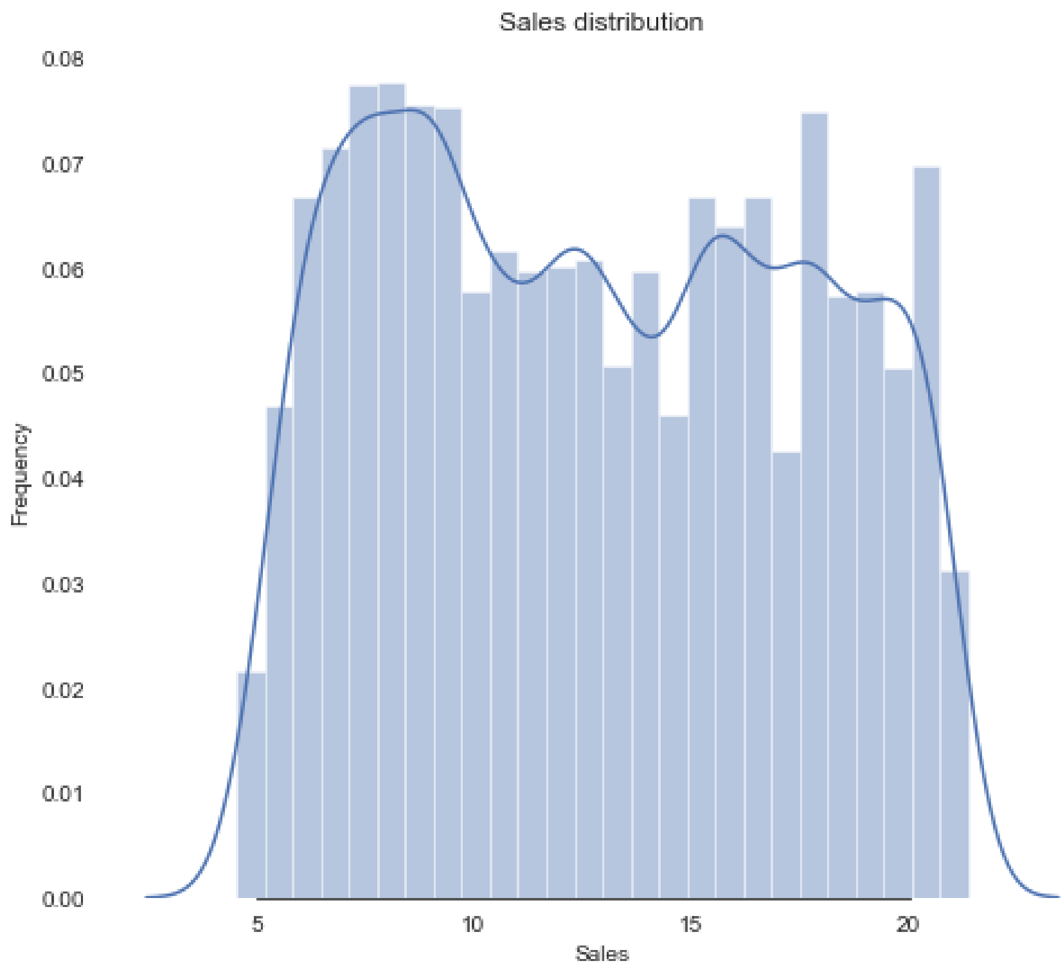


2. Item_Weight

[38]:
```
sb.set_style("white")
sb.set_color_codes(palette='deep')
f, ax = plt.subplots(figsize=(8, 7))
#Check the new distribution
sb.distplot(data['Item_Weight'], color="b");
ax.xaxis.grid(False)
ax.set(ylabel="Frequency")
ax.set(xlabel="Sales")
ax.set(title="Sales distribution")
sb.despine(trim=True, left=True)
plt.show()
# Skew and kurt
print("Skewness: %f" % data['Item_Weight'].skew())
print("Kurtosis: %f" % data['Item_Weight'].kurt())
```

```
C:\Users\C\conda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
```

```
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



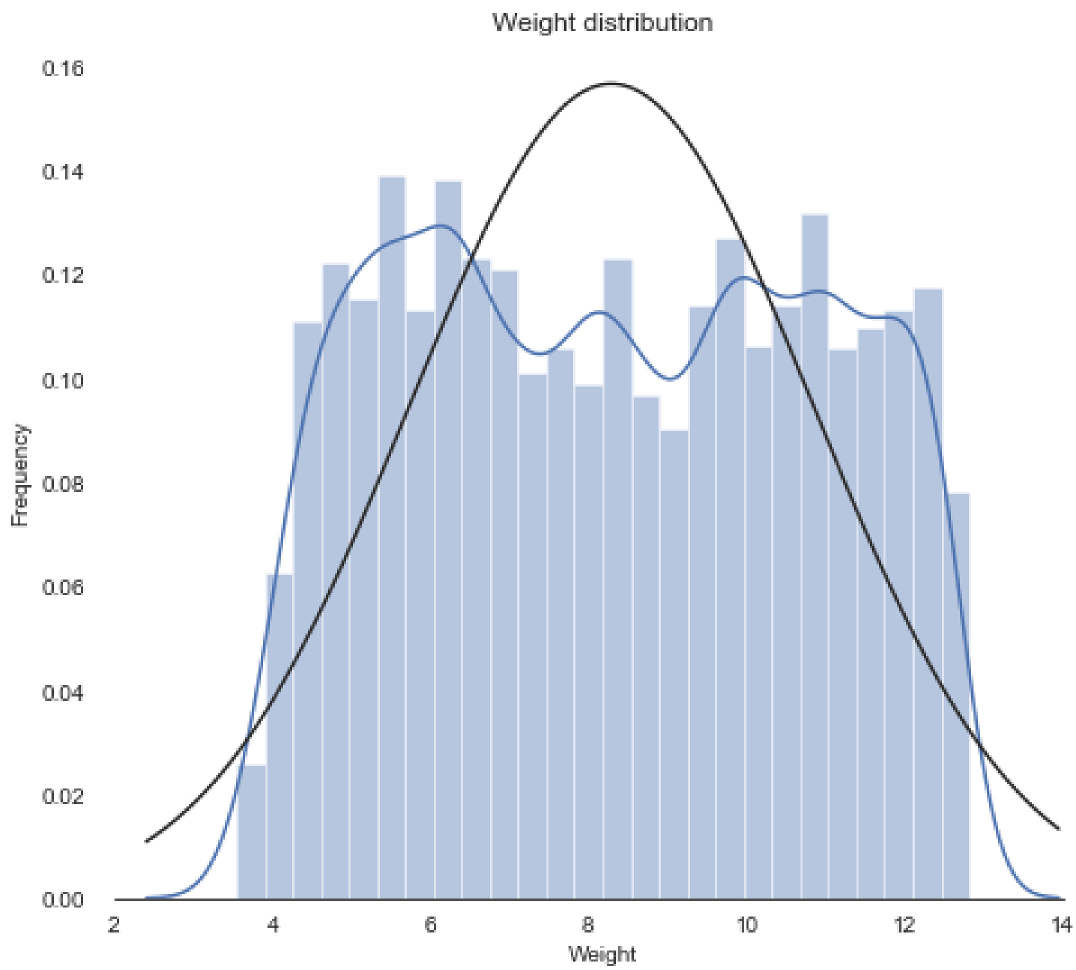

```
Skewness: 0.099016
Kurtosis: -1.227968
```

[23]:
```
# Exp transformation
data['Item_Weight'] = data.Item_Weight**(1/1.2)
```

[40]:
```
sb.set_color_codes(palette='deep')
f, ax = plt.subplots(figsize=(8, 7))
#Check the new distribution
sb.distplot(data['Item_Weight'] , fit=norm, color="b");

# Get the fitted parameters used by the function
```

```
ax.xaxis.grid(False)
ax.set(ylabel="Frequency")
ax.set(xlabel="Weight")
ax.set(title="Weight distribution")
sb.despine(trim=True, left=True)

plt.show()
```

```
C:\Users\C\conda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



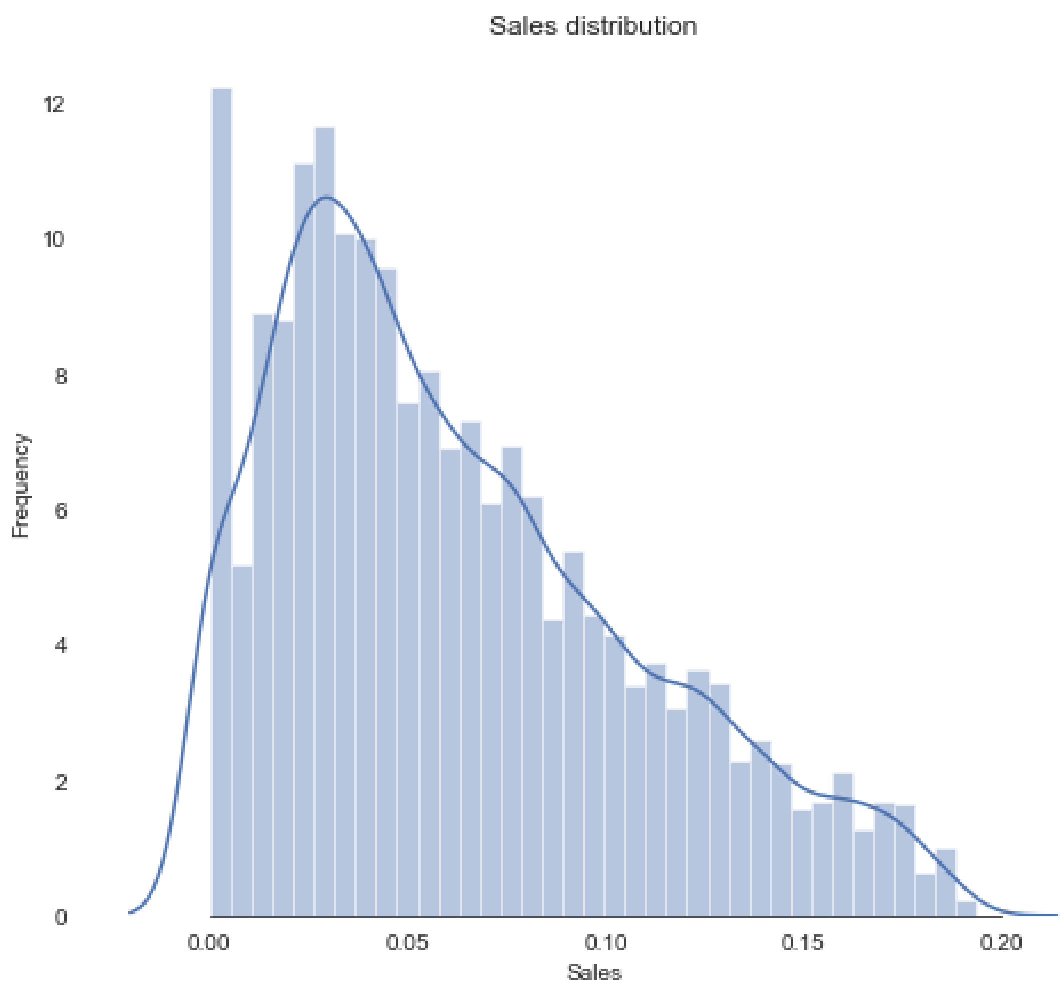
We will choose :- 'Item_Weight_exp', 'Item_Visibility_sqaure', 'Item_MRP_exp', 'Item_Outlet_Sales_sqaure', 'How_old_Outlet_sqaure'

3. Item_Visibility

[41]:

```
sb.set_style("white")
sb.set_color_codes(palette='deep')
f, ax = plt.subplots(figsize=(8, 7))
#Check the new distribution
sb.distplot(data['Item_Visibility'], color="b");
ax.xaxis.grid(False)
ax.set(ylabel="Frequency")
ax.set(xlabel="Sales")
ax.set(title="Sales distribution")
sb.despine(trim=True, left=True)
plt.show()
# Skew and kurt
print("Skewness: %f" % data['Item_Visibility'].skew())
print("Kurtosis: %f" % data['Item_Visibility'].kurt())
```

```
C:\Users\C\conda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



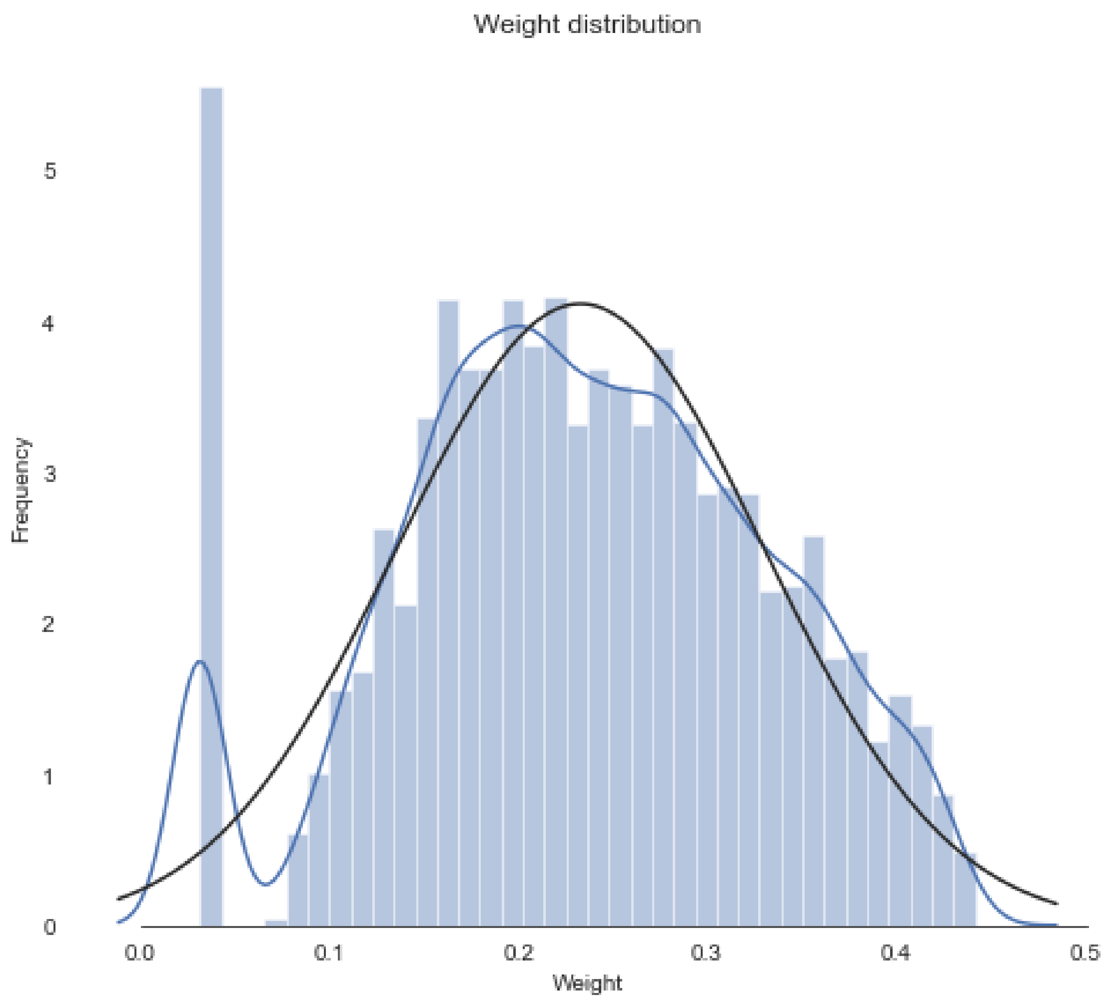

```
Skewness: 0.730801
Kurtosis: -0.248291
```

[24]:
```
# Exp transformation
data['Item_Visibility'] = (data.Item_Visibility+0.001)**(1/2)
```

[43]:
```
sb.set_color_codes(palette='deep')
f, ax = plt.subplots(figsize=(8, 7))
#Check the new distribution
sb.distplot(data['Item_Visibility'] , fit=norm, color="b");

# Get the fitted parameters used by the function
ax.xaxis.grid(False)
ax.set(ylabel="Frequency")
ax.set(xlabel="Weight")
ax.set(title="Weight distribution")
sb.despine(trim=True, left=True)
```

```
plt.show()
```

```
C:\Users\C\conda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



1.1.3 Adding Features

There are 4 clusters in the Item_MRP feature. I'll categorize them into a new feature

[25]:
```
def binn(x):
    if x<75: return 0
    elif x<145: return 1
    elif x<200: return 2
```

```
    else: return 3
data['Item_MRP_Bin_qcut'] = data['Item_MRP'].apply(binn)
data[['Item_MRP', 'Item_MRP_Bin_qcut']].head()
```

[25]:

	Item_MRP	Item_MRP_Bin_qcut
0	249.8092	3
1	48.2692	0
2	141.6180	1
3	182.0950	2
4	53.8614	0

Item_Identifier has a lot of values. Encoding them will cause dimentionality curse. To solve this, I'll code them by the first three letters then I'll Frequency code them into a new feature.

[26]:
```
data['Item_Code'] = data['Item_Identifier'].apply(lambda x: x[0:3])
data[['Item_Identifier', 'Item_Code']].head()
```

[26]:

	Item_Identifier	Item_Code
0	FDA15	FDA
1	DRC01	DRC
2	FDN15	FDN
3	FDX07	FDX
4	NCD19	NCD

[27]:
```
# Frequency encoding using value_counts function
Item_Code_freq = data['Item_Code'].value_counts(normalize=True)

# Mapping the encoded values with original data
data['Item_Code_freq'] = data['Item_Code'].apply(lambda x : Item_Code_freq[x])

print('The sum of Item_Code_freq variable:', sum(Item_Code_freq))
data[['Item_Code', 'Item_Code_freq']].head(6)
```

```
The sum of Item_Code_freq variable: 1.0000000000000002
```

[27]:

	Item_Code	Item_Code_freq
0	FDA	0.029047
1	DRC	0.005236
2	FDN	0.024098
3	FDX	0.032776
4	NCD	0.006383
5	FDP	0.027397

I'll add another feature that averages the sales of each item type & id

[28]:
```
data['Item_Outlet_Sales_Mean'] = data.groupby(['Item_Identifier',␣
↪'Item_Type'])['Sqrt_Sales']\
                                    .transform(lambda x: x.mean())
```

```
data[['Item_Identifier','Item_Type','Item_Outlet_Sales','Item_Outlet_Sales_Mean']].
↪tail()
```

[28]:
```
      Item_Identifier           Item_Type  Item_Outlet_Sales  \
14199           FDB58         Snack Foods                NaN
14200           FDD47       Starchy Foods                NaN
14201           NCO17  Health and Hygiene                NaN
14202           FDJ26              Canned                NaN
14203           FDU37              Canned                NaN

       Item_Outlet_Sales_Mean
14199               43.888017
14200               60.369285
14201               43.613733
14202               50.848006
14203               44.363552
```

Encoding

Outlet_Size

[29]:
```
label_encoder = preprocessing.LabelEncoder()
data['Outlet_Size']= label_encoder.fit_transform(data['Outlet_Size'])
```

[30]:
```
data['Outlet_Size'].unique()
```

[30]:
```
array([1, 0, 2])
```

One-Hot-Encoding

[31]:
```
data = pd.get_dummies(data=data,␣
↪columns=['Outlet_Type','Item_Fat_Content','Outlet_Identifier','Item_Type','Outlet_Location_
↪drop_first=True)
```

[32]:
```
data
```

[32]:
```
      Item_Identifier  Item_Weight  Item_Visibility  Item_MRP  \
0               FDA15     6.413116         0.130565  249.8092
1               DRC01     4.401507         0.142402   48.2692
2               FDN15    10.860881         0.133267  141.6180
3               FDX07    11.733233         0.031623  182.0950
4               NCD19     6.199779         0.031623   53.8614
…                 …          …                …        …
14199           FDB58     7.095632         0.120401  141.3154
14200           FDD47     5.420150         0.379461  169.1448
14201           NCO17     6.812921         0.272999  118.7440
14202           FDJ26     9.710528         0.031623  214.6218
14203           FDU37     6.527843         0.325146   79.7960
```

```
       Outlet_Establishment_Year  Outlet_Size  Item_Outlet_Sales source  \
0                           1999            1          3735.1380  train
1                           2009            1           443.4228  train
2                           1999            1          2097.2700  train
3                           1998            1           732.3800  train
4                           1987            0           994.7052  train
…                              …          …                 …     …
14199                       1997            2                NaN   test
14200                       2009            1                NaN   test
14201                       2002            1                NaN   test
14202                       2007            2                NaN   test
14203                       2002            1                NaN   test

       Sqrt_Sales  Item_MRP_Bin_qcut  … Item_Type_Health and Hygiene  \
0       61.115775                  3  …                            0
1       21.057607                  0  …                            0
2       45.795961                  1  …                            0
3       27.062520                  2  …                            0
4       31.538947                  0  …                            0
…             …                …  …                          …
14199         NaN                  1  …                            0
14200         NaN                  2  …                            0
14201         NaN                  1  …                            1
14202         NaN                  3  …                            0
14203         NaN                  1  …                            0

       Item_Type_Household  Item_Type_Meat  Item_Type_Others  \
0                        0               0                 0
1                        0               0                 0
2                        0               1                 0
3                        0               0                 0
4                        1               0                 0
…                      …             …               …
14199                    0               0                 0
14200                    0               0                 0
14201                    0               0                 0
14202                    0               0                 0
14203                    0               0                 0

       Item_Type_Seafood  Item_Type_Snack Foods  Item_Type_Soft Drinks  \
0                      0                      0                      0
1                      0                      0                      1
2                      0                      0                      0
3                      0                      0                      0
4                      0                      0                      0
…                    …                    …                    …
14199                  0                      1                      0
```

```
14200                    0                       0                       0
14201                    0                       0                       0
14202                    0                       0                       0
14203                    0                       0                       0

       Item_Type_Starchy Foods  Outlet_Location_Type_Tier 2  \
0                            0                            0
1                            0                            0
2                            0                            0
3                            0                            0
4                            0                            0
…                          …                          …
14199                        0                            0
14200                        1                            0
14201                        0                            1
14202                        0                            1
14203                        0                            1

       Outlet_Location_Type_Tier 3
0                                0
1                                1
2                                0
3                                1
4                                1
…                              …
14199                            0
14200                            1
14201                            0
14202                            0
14203                            0

[13943 rows x 43 columns]
```

Scaling

[33]:
```
data.drop(columns =␣
↪['Item_Identifier','Item_Outlet_Sales','Item_Code'],inplace=True)
```

[34]:
```
# Separate target from predictors
y = data[['Sqrt_Sales','source']]
X = data.drop(['Sqrt_Sales','source'], axis=1)
ss = RobustScaler()
df_scaled = pd.DataFrame(ss.fit_transform(X),columns = X.columns)
```

[35]:
```
data = df_scaled
data[['Sqrt_Sales','source']] = y
data.head()
```

[35]:

```
   Item_Weight  Item_Visibility  Item_MRP  Outlet_Establishment_Year  \
0    -0.419788        -0.737045  1.171129                   0.000000
1    -0.877040        -0.650828 -1.023223                   0.588235
2     0.591216        -0.717366 -0.006849                   0.000000
3     0.789508        -1.457757  0.433862                  -0.058824
4    -0.468281        -1.457757 -0.962335                  -0.705882

   Outlet_Size  Item_MRP_Bin_qcut  Item_Code_freq  Item_Outlet_Sales_Mean  \
0          0.0                2.0        0.177258                1.453640
1          0.0               -1.0       -0.933110               -0.829408
2          0.0                0.0       -0.053512               -0.468787
3          0.0                1.0        0.351171                0.543592
4         -1.0               -1.0       -0.879599               -1.038699

   Outlet_Type_Supermarket Type1  Outlet_Type_Supermarket Type2  …  \
0                            0.0                            0.0  …
1                           -1.0                            1.0  …
2                            0.0                            0.0  …
3                           -1.0                            0.0  …
4                            0.0                            0.0  …

   Item_Type_Meat  Item_Type_Others  Item_Type_Seafood  Item_Type_Snack Foods  \
0             0.0               0.0                0.0                    0.0
1             0.0               0.0                0.0                    0.0
2             1.0               0.0                0.0                    0.0
3             0.0               0.0                0.0                    0.0
4             0.0               0.0                0.0                    0.0

   Item_Type_Soft Drinks  Item_Type_Starchy Foods  \
0                    0.0                      0.0
1                    1.0                      0.0
2                    0.0                      0.0
3                    0.0                      0.0
4                    0.0                      0.0

   Outlet_Location_Type_Tier 2  Outlet_Location_Type_Tier 3  Sqrt_Sales  \
0                          0.0                          0.0   61.115775
1                          0.0                          1.0   21.057607
2                          0.0                          0.0   45.795961
3                          0.0                          1.0   27.062520
4                          0.0                          1.0   31.538947

   source
0   train
1   train
2   train
3   train
```

```
4   train

[5 rows x 40 columns]
```

Splitting

[36]:
```
# divide into train and test
train_s = data.loc[data['source']=='train']
test_s = data.loc[data['source']=='test']

# drop unnecessary columns
train_s.drop(['source'], axis=1, inplace=True)
test_s.drop(['Sqrt_Sales', 'source'], axis=1, inplace=True)
```

```
C:\Users\C\conda\lib\site-packages\pandas\core\frame.py:4308:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

[37]:
```
# Separate target from predictors
y = train_s.Sqrt_Sales
X = train_s.drop(['Sqrt_Sales'], axis=1)# Divide data into training and␣
↪validation subsets

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
↪test_size=0.2, random_state=42)
```

[38]:
```
X.shape
```

[38]:
```
(8375, 38)
```

Modeling

[39]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.feature_selection import RFECV

estimator = DecisionTreeRegressor(max_depth=10, min_samples_leaf=100)
selector = RFECV(estimator=estimator, cv=5)
selector = selector.fit(X_train, y_train)
```

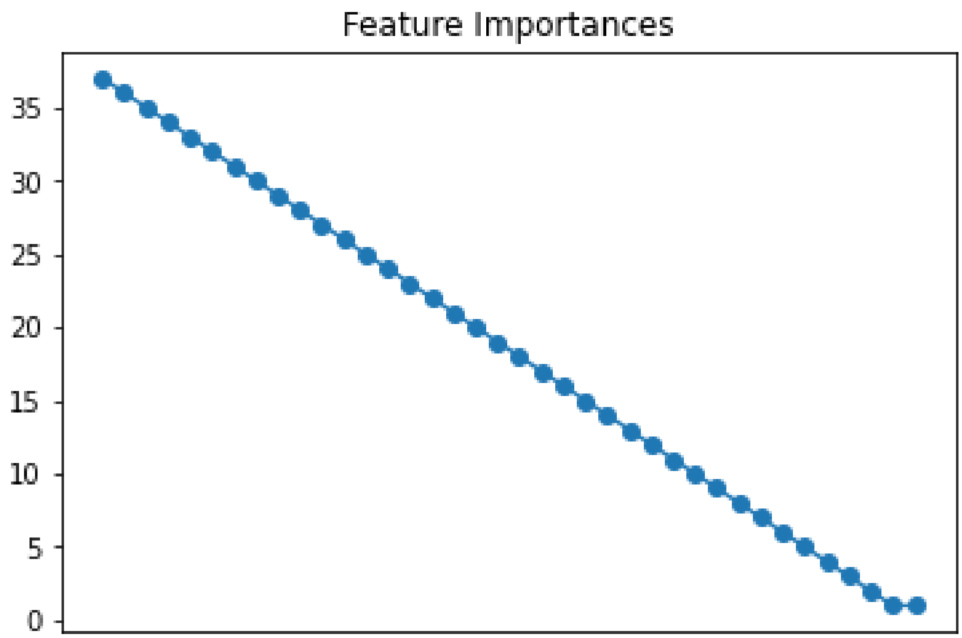
1.1.4 Feature Selection

We are going the best 20 feature to feed them to our model.

[40]:

```python
# visualise result
coefs = pd.Series(selector.ranking_, X_train.columns).
↪sort_values(ascending=False)
ax=coefs.plot(title='Feature Importances', marker='o', rot=90)
ax.axes.get_xaxis().set_visible(False)
print("Feature Ranking:\n %s"% coefs)
```

```
Feature Ranking:
 Outlet_Identifier_OUT017           37
Outlet_Identifier_OUT046           36
Outlet_Identifier_OUT045           35
Outlet_Identifier_OUT013           34
Outlet_Identifier_OUT018           33
Item_Type_Breads                   32
Item_Type_Household                31
Outlet_Type_Supermarket Type3      30
Item_Type_Health and Hygiene       29
Outlet_Type_Supermarket Type2      28
Item_Type_Hard Drinks              27
Outlet_Type_Supermarket Type1      26
Item_Type_Fruits and Vegetables    25
Item_Type_Starchy Foods            24
Outlet_Identifier_OUT035           23
Item_Type_Soft Drinks              22
Outlet_Identifier_OUT027           21
Item_Type_Snack Foods              20
Outlet_Identifier_OUT019           19
Item_Type_Seafood                  18
Item_Type_Others                   17
Item_Type_Meat                     16
Item_MRP_Bin_qcut                  15
Item_Type_Breakfast                14
Item_Type_Canned                   13
Item_Type_Dairy                    12
Item_Type_Frozen Foods             11
Outlet_Location_Type_Tier 2        10
Outlet_Location_Type_Tier 3         9
Outlet_Identifier_OUT049            8
Outlet_Size                         7
Item_Visibility                     6
Item_Fat_Content_Regular            5
Item_Outlet_Sales_Mean              4
Outlet_Establishment_Year           3
Item_Code_freq                      2
Item_MRP                            1
Item_Weight                         1
dtype: int32
```



[41]:
```
features = list(coefs.index[-20:])
not_features = list(coefs.index[:-20])
```

[42]:
```
features
```

[42]:
```
['Outlet_Identifier_OUT019',
 'Item_Type_Seafood',
 'Item_Type_Others',
 'Item_Type_Meat',
 'Item_MRP_Bin_qcut',
 'Item_Type_Breakfast',
 'Item_Type_Canned',
 'Item_Type_Dairy',
 'Item_Type_Frozen Foods',
 'Outlet_Location_Type_Tier 2',
 'Outlet_Location_Type_Tier 3',
 'Outlet_Identifier_OUT049',
 'Outlet_Size',
 'Item_Visibility',
 'Item_Fat_Content_Regular',
 'Item_Outlet_Sales_Mean',
 'Outlet_Establishment_Year',
 'Item_Code_freq',
 'Item_MRP',
 'Item_Weight']
```

[43]:
```
X = X[features]
test = test_s[features]
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
↪test_size=0.2, random_state=42)
```

[44]:
```
def error(true, pred):
    print('RMSE: ', np.sqrt(mean_squared_error(true, pred)))
    print('MAE: ', mean_absolute_error(true, pred))

def visualize(true,pred):
    plt.figure(figsize=(20, 7))
    ax = sb.kdeplot(true, color="r", label="Actual Value")
    sb.kdeplot(pred, color="b", label="Fitted Values" , ax=ax)
    plt.title('Actual vs Fitted Values for Price')
    plt.legend()
    plt.show()
    plt.close()
```

1.2 Linear Regression

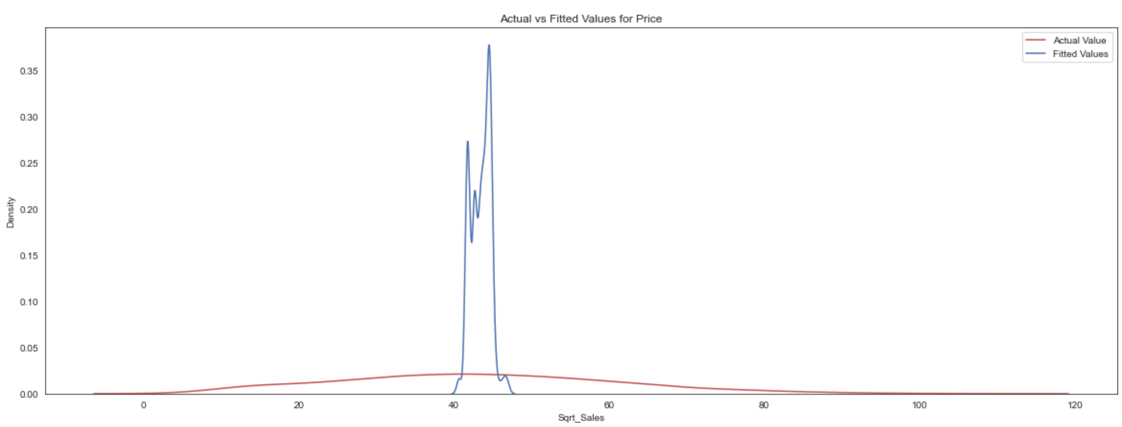
[45]:
```
from sklearn.linear_model import LinearRegression

reg_lin=LinearRegression()
reg_lin.fit(X_train,y_train)
reg_preds = reg_lin.predict(X_test)
error(y_test**2,reg_preds**2)
```

```
RMSE:  1712.1060972960797
MAE:  1275.3624039499932
```

[46]:
```
reg_lin.score(X_train,y_train)
```

[46]: 0.0044331515990807535

[64]:
```
visualize(y_test,reg_preds)
```

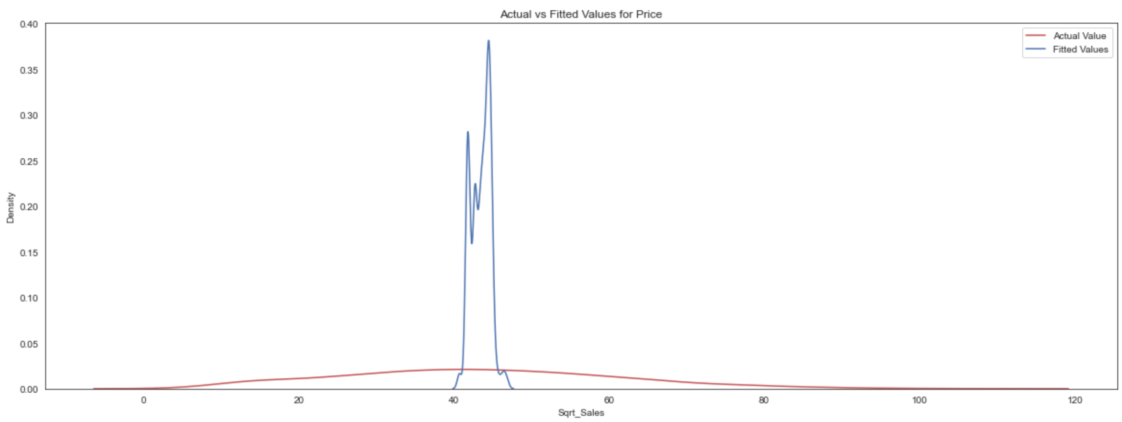


1.3 Ridge Regression

[47]:
```
from sklearn.linear_model import RidgeCV

reg_rid=RidgeCV(cv=10)
reg_rid.fit(X_train,y_train)
reg_rid_preds = reg_rid.predict(X_test)
error(y_test**2,reg_rid_preds**2)
```

```
RMSE:  1711.9818409281402
MAE:  1275.3070166485863
```

[66]:
```
visualize(y_test,reg_rid_preds)
```



1.4 Lasso Regression

[48]:
```
from sklearn.linear_model import Lasso

reg_las=Lasso()
reg_las.fit(X_train,y_train)
reg_las_preds = reg_las.predict(X_test)
error(y_test**2,reg_las_preds**2)
```

```
RMSE:  1706.839454357763
MAE:  1272.5707754163175
```

[49]:
```
reg_las.score(X_test,y_test)
```

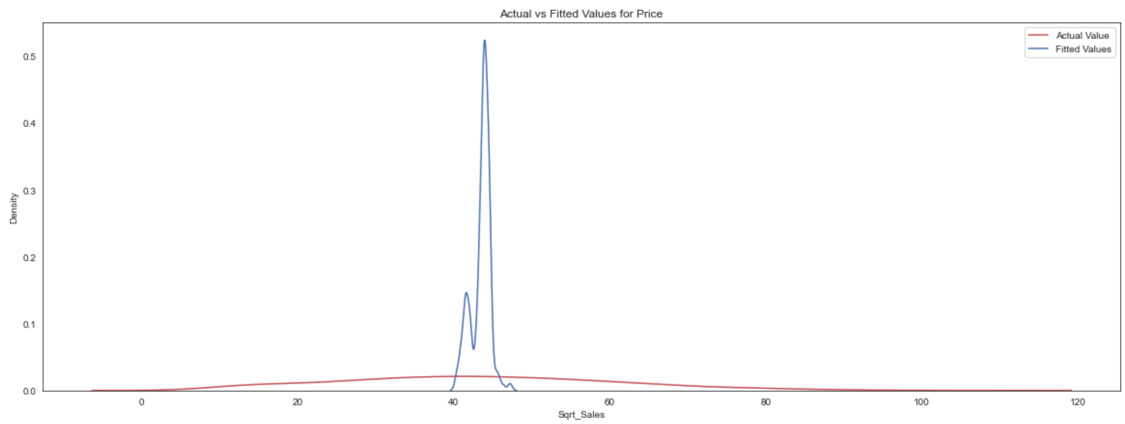
[49]: -0.0004706312433240267

1.5 Random Forrest

[50]:
```
rf = RandomForestRegressor(n_estimators=200,
                           max_depth=5,
                           random_state=42)
# Preprocessing of training data, fit model
rf.fit(X_train, y_train)
# Preprocessing of validation data, get predictions
rf_preds = rf.predict(X_test)
# Evaluate the model
error(y_test**2,rf_preds**2)
```

```
RMSE:  1709.985871556812
MAE:  1274.2896135367498
```

[64]:
```
visualize(y_test,rf_preds)
```



[65]:
```
rf.score(X_test,y_test)
```

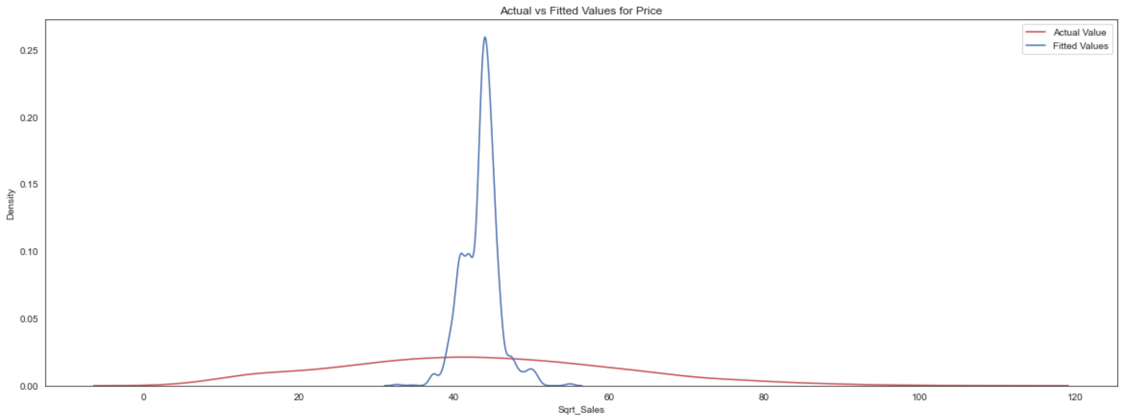
[65]: -0.007050661540812353

1.6 XGB

[51]:
```
xgboost = XGBRegressor()
# Preprocessing of training data, fit model
xgboost.fit(X_train, y_train)
# Preprocessing of validation data, get predictions
xgboost_preds = xgboost.predict(X_test)
# Evaluate the model
error(y_test**2,xgboost_preds**2)
```

```
RMSE:  1810.9876742129602
MAE:  1353.2496915813208
```

[71]:
```
visualize(y_test,xgboost_preds)
```

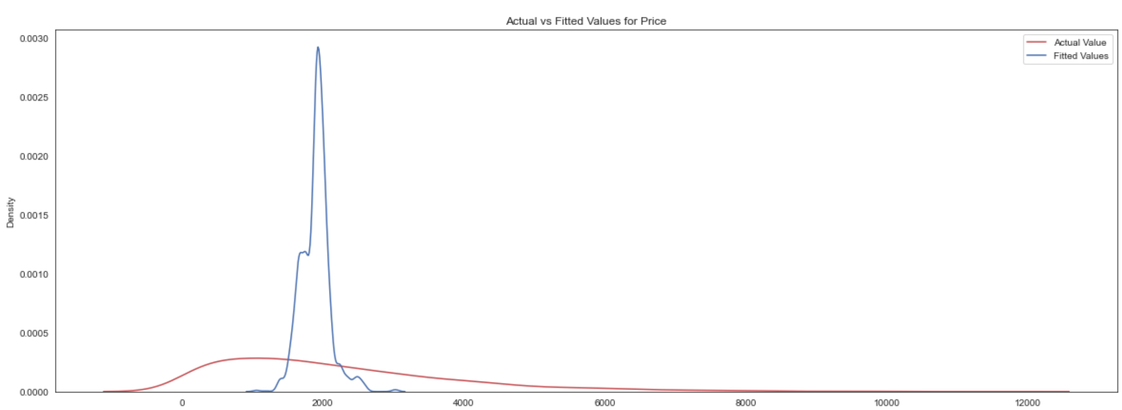


1.7 GradientBoostingRegressor

[52]:
```
gbr = GradientBoostingRegressor(random_state=42,
                                subsample = 1,
                                n_estimators=500,
                                max_depth = 6)
gbr.fit(X_train, y_train)
# Preprocessing of validation data, get predictions
gbr_preds = gbr.predict(X_test)
# Evaluate the model
error(y_test**2,gbr_preds**2)
```

```
RMSE:  1851.0696061142567
MAE:  1403.5454876338197
```

[73]:
```
visualize(y_test**2,gbr_preds**2)
```



1.8 Test Model

[80]:
```
Test_Predictions = pd.DataFrame(rf.predict(test)**2)
Test_Predictions.columns = ['Item_Outlet_Sales']
```

[81]:
```
Test_Predictions.to_csv('Test_Output')
```