Mini_Project

May 10, 2023

1 Importing Required Libraries

[1]:
```
import os #paths to file
import numpy as np # linear algebra
import pandas as pd # data processing
import warnings# warning filter


#ploting libraries
import matplotlib.pyplot as plt
import seaborn as sns

#feature engineering
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

#train test split
from sklearn.model_selection import train_test_split

#metrics
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score as R2
from sklearn.model_selection  import cross_val_score as CVS


#ML models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Lasso


#default theme and settings
sns.set(context='notebook', style='darkgrid', palette='deep',␣
 ↪font='sans-serif', font_scale=1, color_codes=False, rc=None)
pd.options.display.max_columns
```

```
#warning hadle
warnings.filterwarnings("always")
warnings.filterwarnings("ignore")
```

2 Creating File Paths

[2]:
```
#list all files under the input directory
for dirname, _, filenames in os.walk('/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

[3]:
```
#path for the training set
tr_path = "Train.csv"
#path for the testing set
te_path = "Test.csv"
```

3 Data Pre-Processing

[4]:
```
# read in csv file as a DataFrame
tr_df = pd.read_csv(tr_path)
# explore the first 5 rows
tr_df.head()
```

[4]:

	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
0	Supermarket Type1	3735.1380

```
1  Supermarket Type2           443.4228
2  Supermarket Type1          2097.2700
3      Grocery Store           732.3800
4  Supermarket Type1           994.7052
```

[5]:
```
# read in csv file as a DataFrame
te_df = pd.read_csv(te_path)
# explore the first 5 rows
te_df.head()
```

[5]:

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

	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	\
0	107.8622	OUT049	1999	Medium	
1	87.3198	OUT017	2007	NaN	
2	241.7538	OUT010	1998	NaN	
3	155.0340	OUT017	2007	NaN	
4	234.2300	OUT027	1985	Medium	

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

[6]:
```
print(f"training set (row, col): {tr_df.shape}\n\ntesting set (row, col):␣
↪{te_df.shape}")
```

```
training set (row, col): (8523, 12)

testing set (row, col): (5681, 11)
```

[7]:
```
#column information
tr_df.info(verbose=True, null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8523 non-null   object
 1   Item_Weight                7060 non-null   float64
```

```
 2   Item_Fat_Content           8523 non-null   object
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object
 7   Outlet_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                6113 non-null   object
 9   Outlet_Location_Type       8523 non-null   object
 10  Outlet_Type                8523 non-null   object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

[8]:
```
#summary statistics test
te_df.describe()
```

[8]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
count	4705.000000	5681.000000	5681.000000	5681.000000
mean	12.695633	0.065684	141.023273	1997.828903
std	4.664849	0.051252	61.809091	8.372256
min	4.555000	0.000000	31.990000	1985.000000
25%	8.645000	0.027047	94.412000	1987.000000
50%	12.500000	0.054154	141.415400	1999.000000
75%	16.700000	0.093463	186.026600	2004.000000
max	21.350000	0.323637	266.588400	2009.000000

[9]:
```
#summary statistics train
tr_df.describe()
```

[9]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	\
count	7060.000000	8523.000000	8523.000000	8523.000000	
mean	12.857645	0.066132	140.992782	1997.831867	
std	4.643456	0.051598	62.275067	8.371760	
min	4.555000	0.000000	31.290000	1985.000000	
25%	8.773750	0.026989	93.826500	1987.000000	
50%	12.600000	0.053931	143.012800	1999.000000	
75%	16.850000	0.094585	185.643700	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

4 Missing Value Treatment

[10]:
```
#missing values in decsending order
print("Train:\n")
print(tr_df.isnull().sum().sort_values(ascending=False),"\n\n",tr_df.isnull().
 ↪sum()/tr_df.shape[0] *100,"\n\n")
print("Test:\n")
print(te_df.isnull().sum().sort_values(ascending=False),"\n\n",te_df.isnull().
 ↪sum()/te_df.shape[0] *100,"\n\n")
```

```
Train:

Outlet_Size                 2410
Item_Weight                 1463
Item_Identifier                0
Item_Fat_Content               0
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Location_Type           0
Outlet_Type                    0
Item_Outlet_Sales              0
dtype: int64

 Item_Identifier             0.000000
Item_Weight                 17.165317
Item_Fat_Content             0.000000
Item_Visibility              0.000000
Item_Type                    0.000000
Item_MRP                     0.000000
Outlet_Identifier            0.000000
Outlet_Establishment_Year    0.000000
Outlet_Size                 28.276428
Outlet_Location_Type         0.000000
Outlet_Type                  0.000000
Item_Outlet_Sales            0.000000
dtype: float64


Test:

Outlet_Size                 1606
Item_Weight                  976
Item_Identifier                0
Item_Fat_Content               0
Item_Visibility                0
```

```
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Location_Type            0
Outlet_Type                     0
dtype: int64

 Item_Identifier               0.000000
Item_Weight                   17.180074
Item_Fat_Content               0.000000
Item_Visibility                0.000000
Item_Type                      0.000000
Item_MRP                       0.000000
Outlet_Identifier              0.000000
Outlet_Establishment_Year      0.000000
Outlet_Size                   28.269671
Outlet_Location_Type           0.000000
Outlet_Type                    0.000000
dtype: float64
```

[11]:
```
print("Outlet_Size:\n", tr_df.Outlet_Size.value_counts(), "\n\n")
print("Item_Weight:\n", tr_df.Item_Weight.value_counts(), "\n\n")
```

```
Outlet_Size:
 Medium    2793
Small     2388
High       932
Name: Outlet_Size, dtype: int64


Item_Weight:
 12.150    86
17.600    82
13.650    77
11.800    76
15.100    68
          ..
7.275      2
7.685      1
9.420      1
6.520      1
5.400      1
Name: Item_Weight, Length: 415, dtype: int64
```

[12]:
```
print("test mode, train mode\n",[tr_df['Outlet_Size'].mode().values[0],␣
 ↪te_df['Outlet_Size'].mode().values[0]])
```

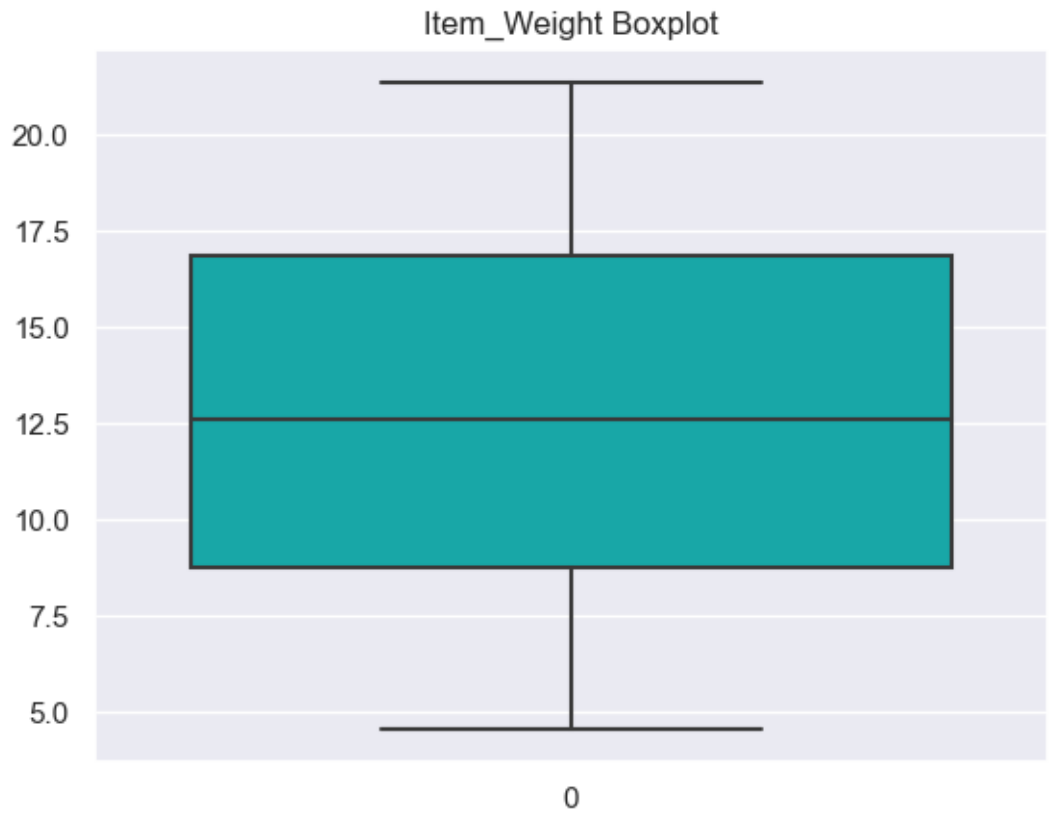
```
test mode, train mode
 ['Medium', 'Medium']
```

[13]:
```
#train
tr_df['Outlet_Size'] = tr_df['Outlet_Size'].fillna(
tr_df['Outlet_Size'].dropna().mode().values[0])

#test
te_df['Outlet_Size'] = te_df['Outlet_Size'].fillna(
te_df['Outlet_Size'].dropna().mode().values[0])

#checking if we filled missing values
tr_df['Outlet_Size'].isnull().sum(),te_df['Outlet_Size'].isnull().sum()
```

[13]:
```
(0, 0)
```

[14]:
```
# I personally prefer a vertical view and a cyan color
sns.boxplot(data=tr_df['Item_Weight'],orient="v", color = 'c')
plt.title("Item_Weight Boxplot")
```

[14]:
```
Text(0.5, 1.0, 'Item_Weight Boxplot')
```



[15]:
```
#train
tr_df['Item_Weight'] = tr_df['Item_Weight'].fillna(
tr_df['Item_Weight'].dropna().mean())

#test
te_df['Item_Weight'] = te_df['Item_Weight'].fillna(
te_df['Item_Weight'].dropna().mean())

#checking if we filled missing values
tr_df['Item_Weight'].isnull().sum(),te_df['Item_Weight'].isnull().sum()
```

[15]:
```
(0, 0)
```

[16]:
```
print("train:\n")
print(tr_df.info())
print("\n\ntest:\n")
print(te_df.info())
```

```
train:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8523 non-null   object
 1   Item_Weight                8523 non-null   float64
 2   Item_Fat_Content           8523 non-null   object
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object
 7   Outlet_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                8523 non-null   object
 9   Outlet_Location_Type       8523 non-null   object
 10  Outlet_Type                8523 non-null   object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
None


test:

<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 5681 entries, 0 to 5680
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            5681 non-null   object
 1   Item_Weight                5681 non-null   float64
 2   Item_Fat_Content           5681 non-null   object
 3   Item_Visibility            5681 non-null   float64
 4   Item_Type                  5681 non-null   object
 5   Item_MRP                   5681 non-null   float64
 6   Outlet_Identifier          5681 non-null   object
 7   Outlet_Establishment_Year  5681 non-null   int64
 8   Outlet_Size                5681 non-null   object
 9   Outlet_Location_Type       5681 non-null   object
 10  Outlet_Type                5681 non-null   object
dtypes: float64(3), int64(1), object(7)
memory usage: 488.3+ KB
None
```

5 Data Exploration

[17]:
```
#list of all the numeric columns
num = tr_df.select_dtypes('number').columns.to_list()
#list of all the categoric columns
cat = tr_df.select_dtypes('object').columns.to_list()

#numeric df
BM_num =  tr_df[num]
#categoric df
BM_cat = tr_df[cat]

#print(num)
#print(cat)

[tr_df[category].value_counts() for category in cat[1:]]
```

[17]:
```
[Low Fat    5089
 Regular    2889
 LF          316
 reg         117
 low fat     112
 Name: Item_Fat_Content, dtype: int64,
 Fruits and Vegetables    1232
 Snack Foods              1200
 Household                 910
 Frozen Foods              856
 Dairy                     682
```

```
Canned                    649
Baking Goods              648
Health and Hygiene        520
Soft Drinks               445
Meat                      425
Breads                    251
Hard Drinks               214
Others                    169
Starchy Foods             148
Breakfast                 110
Seafood                    64
Name: Item_Type, dtype: int64,
OUT027    935
OUT013    932
OUT049    930
OUT046    930
OUT035    930
OUT045    929
OUT018    928
OUT017    926
OUT010    555
OUT019    528
Name: Outlet_Identifier, dtype: int64,
Medium    5203
Small     2388
High       932
Name: Outlet_Size, dtype: int64,
Tier 3    3350
Tier 2    2785
Tier 1    2388
Name: Outlet_Location_Type, dtype: int64,
Supermarket Type1    5577
Grocery Store        1083
Supermarket Type3     935
Supermarket Type2     928
Name: Outlet_Type, dtype: int64]
```

[18]:
```
#train
tr_df['Item_Fat_Content'].replace(['LF', 'low fat', 'reg'],
                                  ['Low Fat','Low Fat','Regular'],inplace =␣
 ↪True)
#test
te_df['Item_Fat_Content'].replace(['LF', 'low fat', 'reg'],
                                  ['Low Fat','Low Fat','Regular'],inplace =␣
 ↪True)

#check result
```

```
tr_df.Item_Fat_Content.value_counts()
```

[18]:
```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

[19]:
```
tr_df.head()
```

[19]:

	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
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	Medium	Tier 3	Grocery Store	732.3800
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

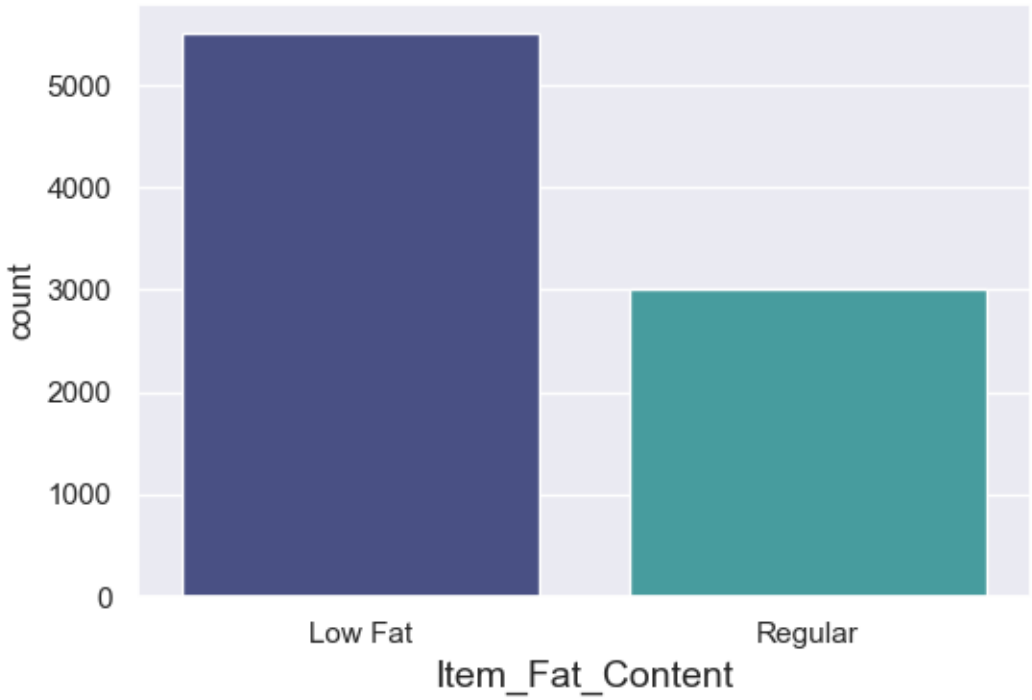
[20]:
```
#creating our new column for both datasets
tr_df['Outlet_Age'], te_df['Outlet_Age']= tr_df['Outlet_Establishment_Year'].
 ↪apply(lambda year: 2020 - year), te_df['Outlet_Establishment_Year'].
 ↪apply(lambda year: 2020 - year)

##uncomment to check result
tr_df['Outlet_Age'].head
te_df['Outlet_Age'].head
```

[20]:
```
<bound method NDFrame.head of 0       21
1       13
```

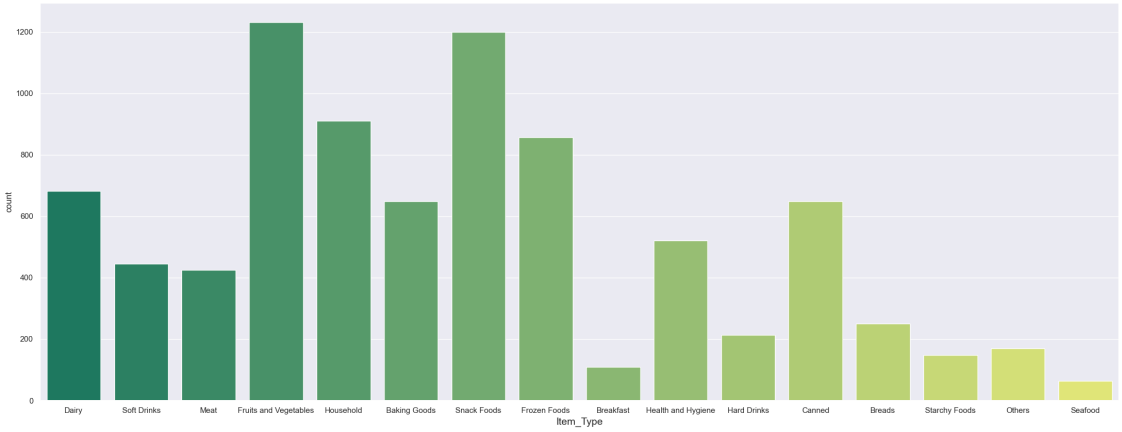
```
2       22
3       13
4       35
        ..
5676    23
5677    11
5678    18
5679    13
5680    18
Name: Outlet_Age, Length: 5681, dtype: int64>
```

6 Data Visualisation

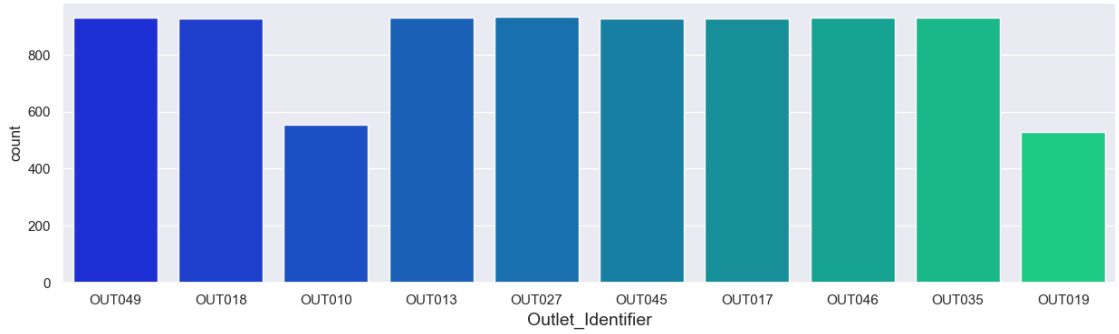
[21]:
```
#categorical columns:
['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier',
 'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type']

plt.figure(figsize=(6,4))
sns.countplot(x='Item_Fat_Content' , data=tr_df ,palette='mako')
plt.xlabel('Item_Fat_Content', fontsize=14)
plt.show()
```



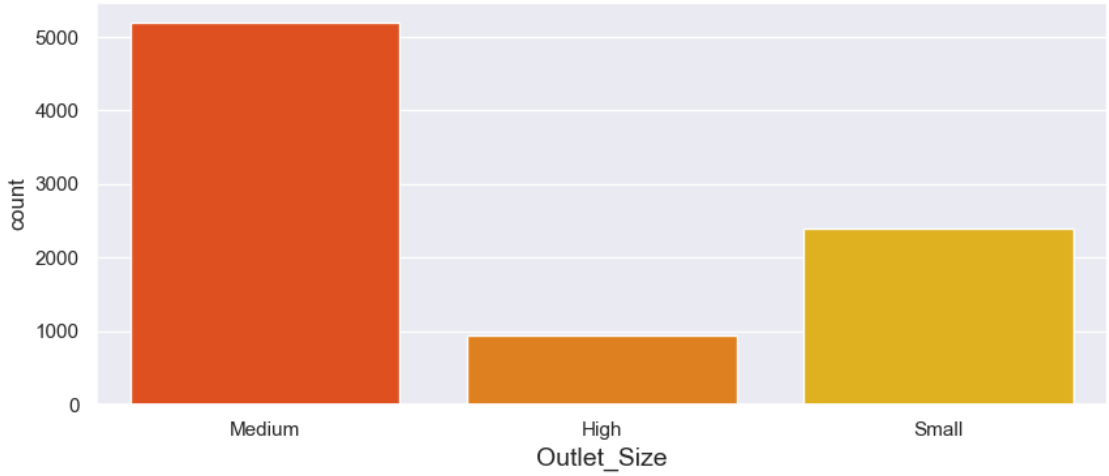
[22]:
```
plt.figure(figsize=(27,10))
sns.countplot(x='Item_Type' , data=tr_df ,palette='summer')
plt.xlabel('Item_Type', fontsize=14)
plt.show()
```



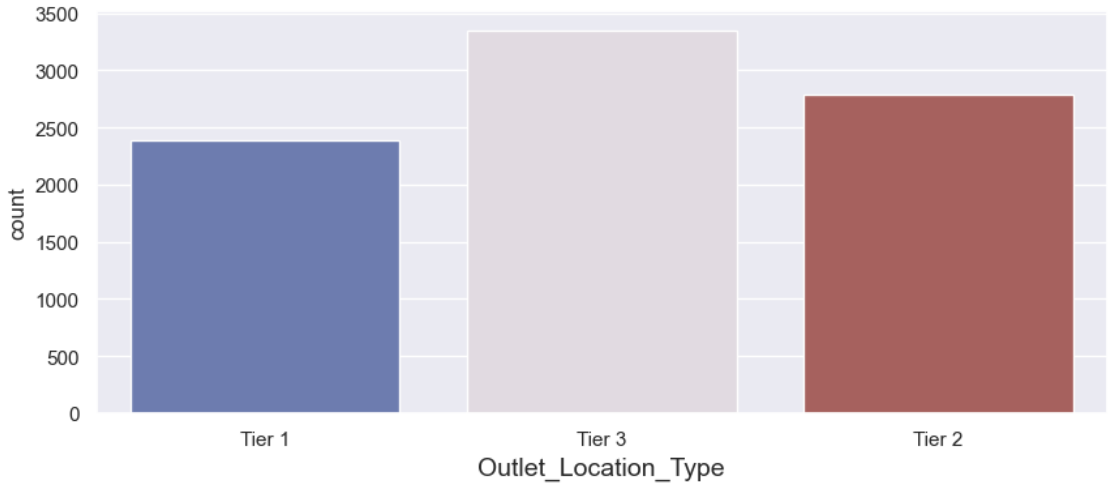
[23]:
```
plt.figure(figsize=(15,4))
sns.countplot(x='Outlet_Identifier' , data=tr_df ,palette='winter')
plt.xlabel('Outlet_Identifier', fontsize=14)
plt.show()
```



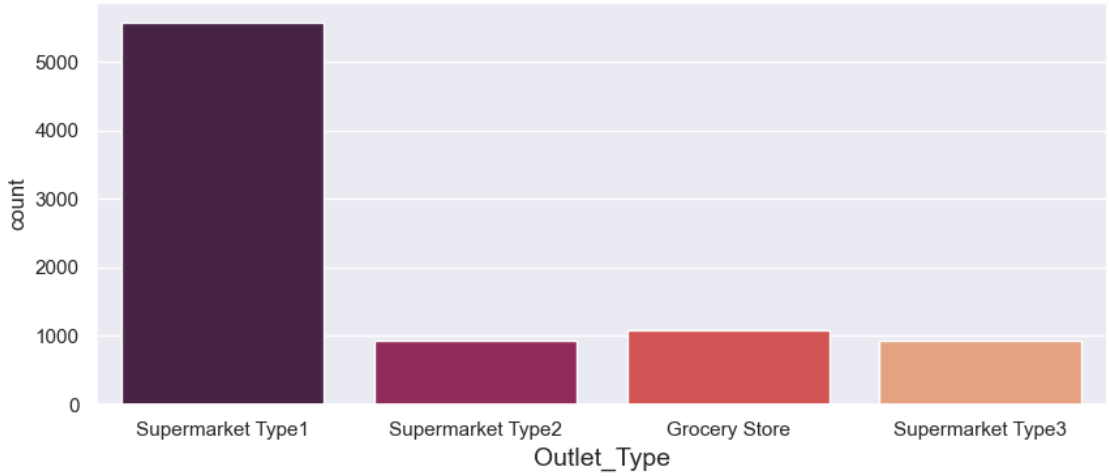
[24]:
```
plt.figure(figsize=(10,4))
sns.countplot(x='Outlet_Size' , data=tr_df ,palette='autumn')
plt.xlabel('Outlet_Size', fontsize=14)
plt.show()
```



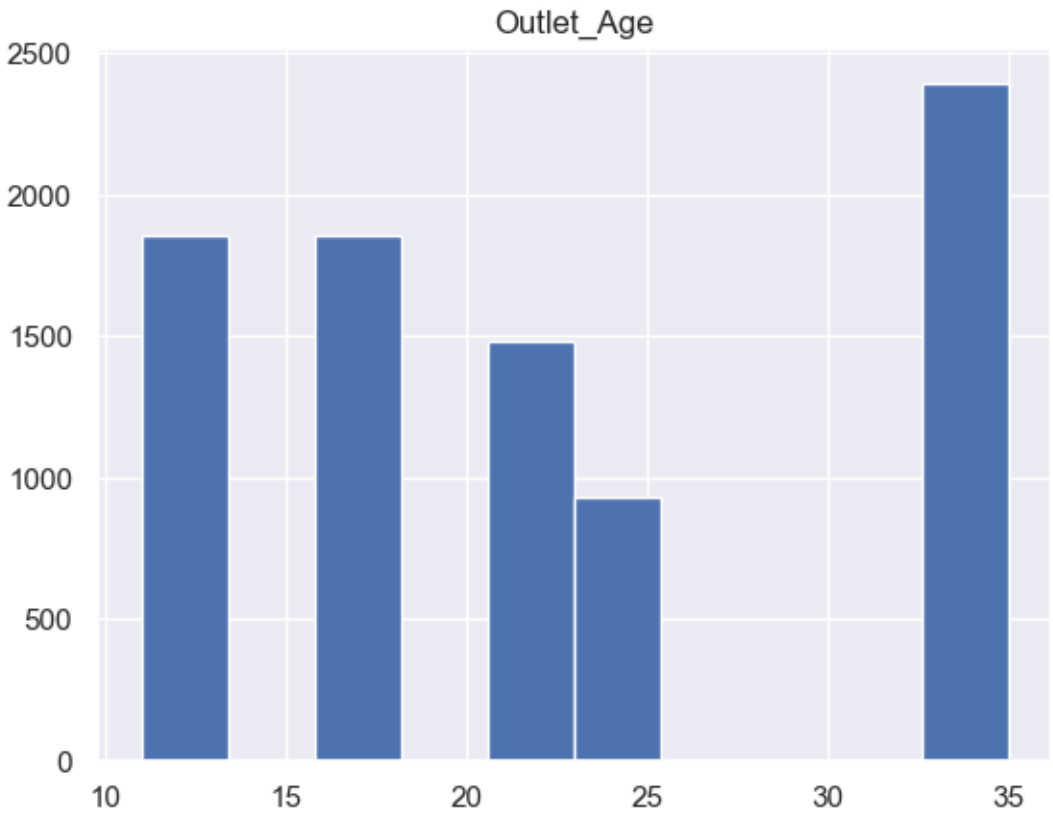
[25]:
```
plt.figure(figsize=(10,4))
sns.countplot(x='Outlet_Location_Type' , data=tr_df ,palette='twilight_shifted')
plt.xlabel('Outlet_Location_Type', fontsize=14)
plt.show()
```



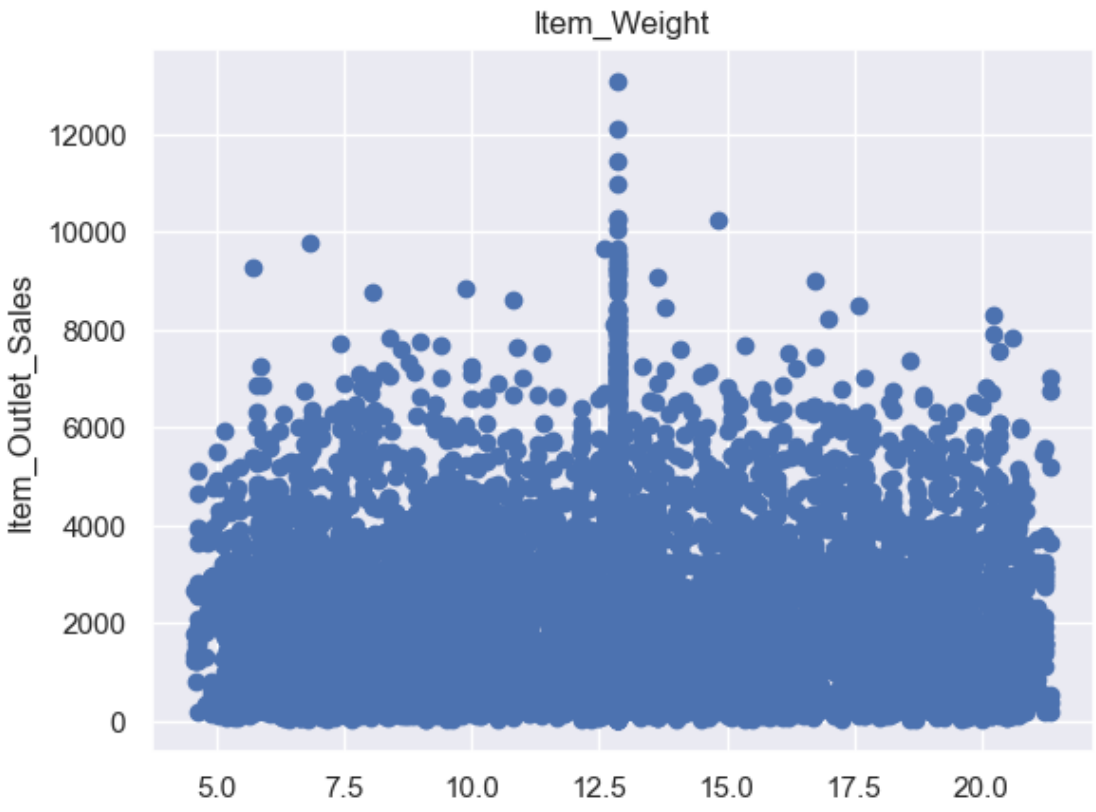
[26]:
```
plt.figure(figsize=(10,4))
sns.countplot(x='Outlet_Type' , data=tr_df ,palette='rocket')
plt.xlabel('Outlet_Type', fontsize=14)
plt.show()
```

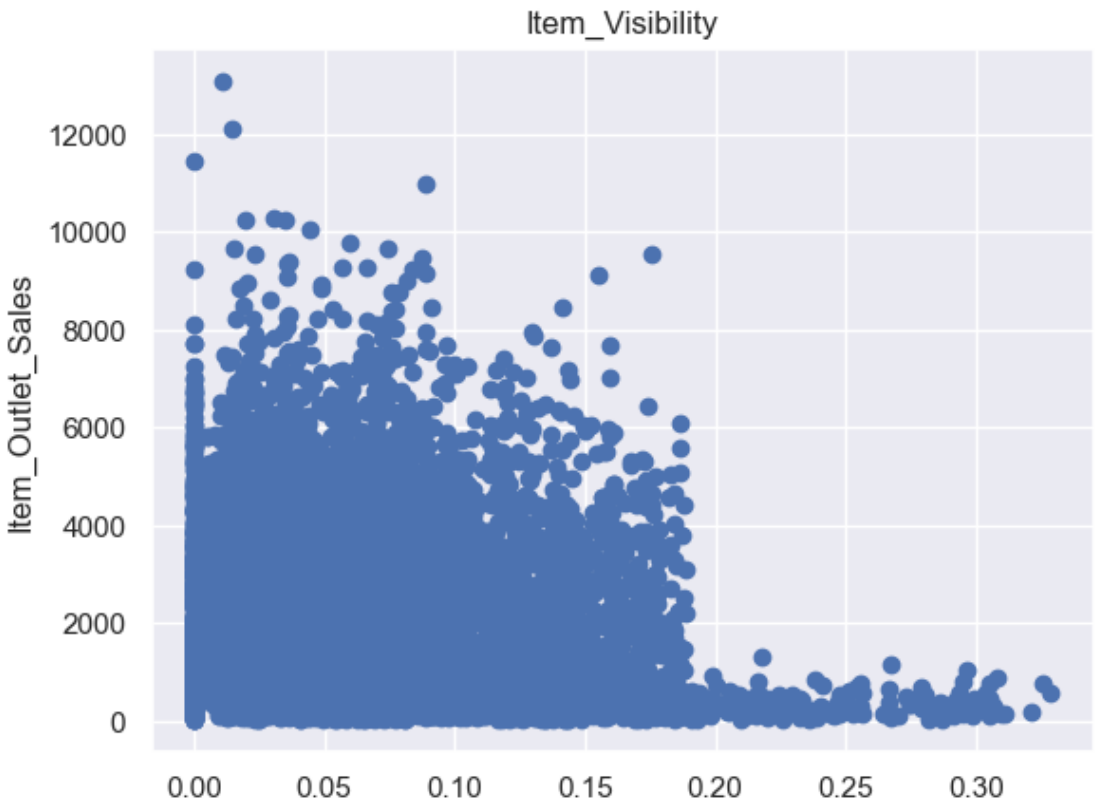


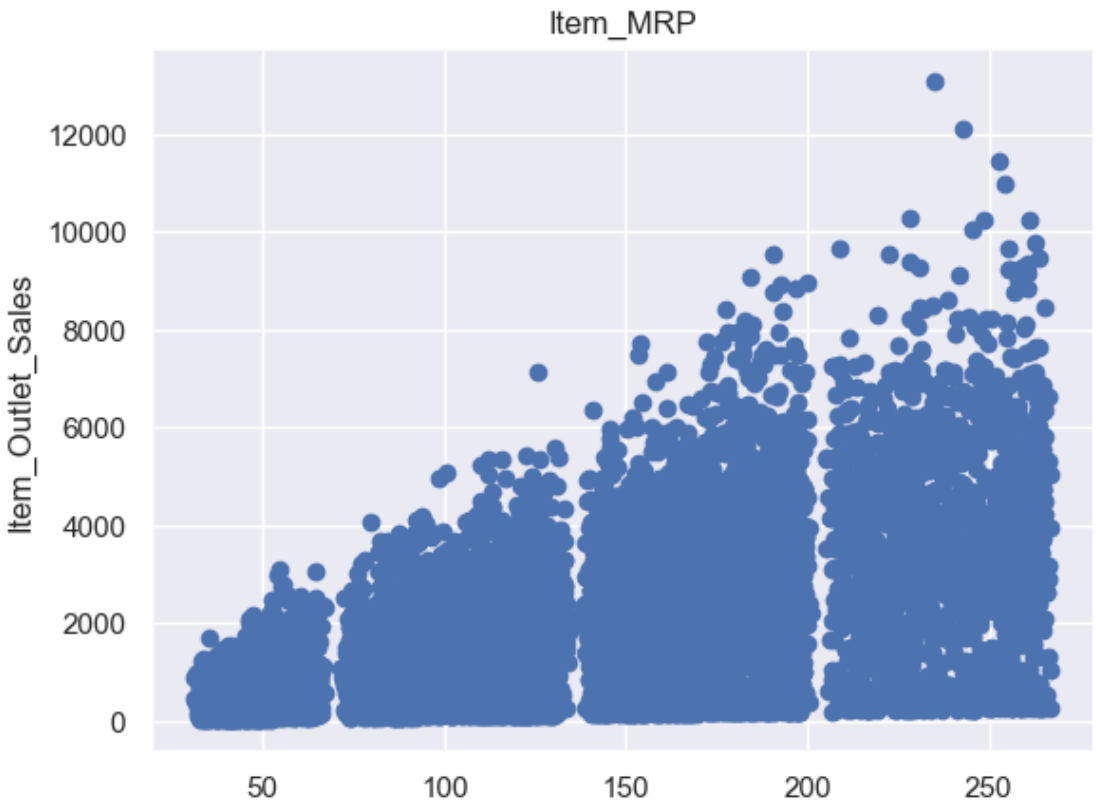
[27]:
```
#list of all the numeric columns
num = tr_df.select_dtypes('number').columns.to_list()
#numeric df
BM_num =  tr_df[num]

plt.hist(tr_df['Outlet_Age'])
plt.title("Outlet_Age")
plt.show()
```



[28]:
```
#because of the variability of the unique values of the numeric columns a␣
 ↪scatter plot with the target value will be of use
for numeric in BM_num[num[:3]]:
    plt.scatter(BM_num[numeric], BM_num['Item_Outlet_Sales'])
    plt.title(numeric)
    plt.ylabel('Item_Outlet_Sales')
    plt.show()
```

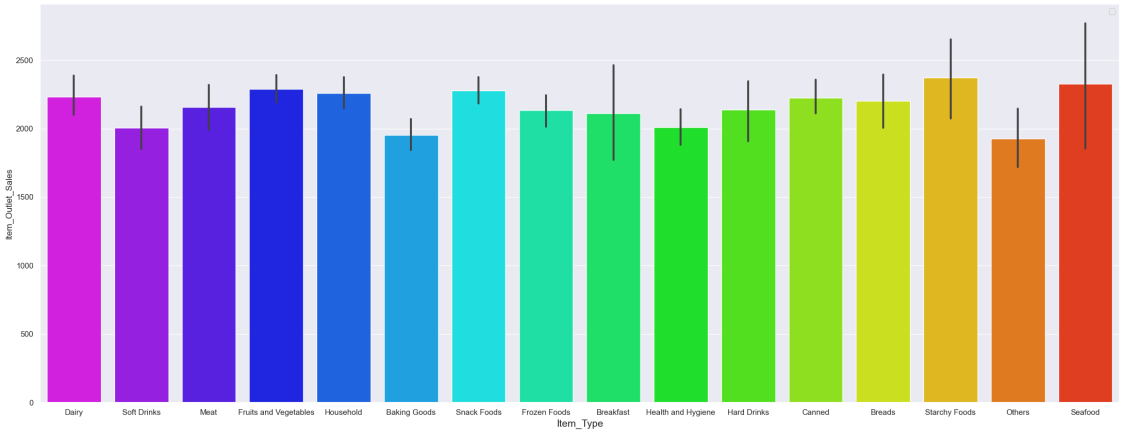

Item_Weight
Item_Outlet_Sales
12000
10000
8000
6000
4000
2000
0
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Item_Visibility
Item_Outlet_Sales
12000
10000
8000
6000
4000
2000
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30



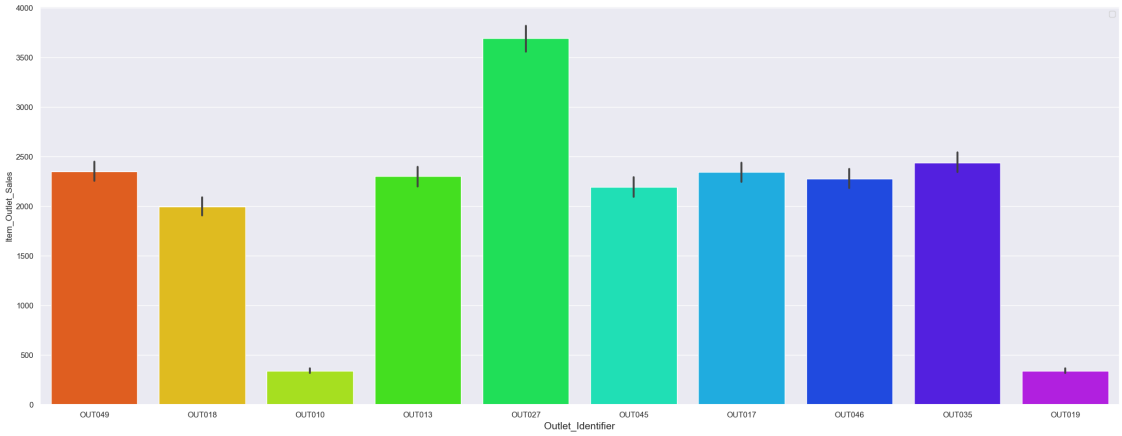
[29]:
```
plt.figure(figsize=(27,10))
sns.barplot('Item_Type' ,'Item_Outlet_Sales', data=tr_df␣
 ↪,palette='gist_rainbow_r')
plt.xlabel('Item_Type', fontsize=14)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



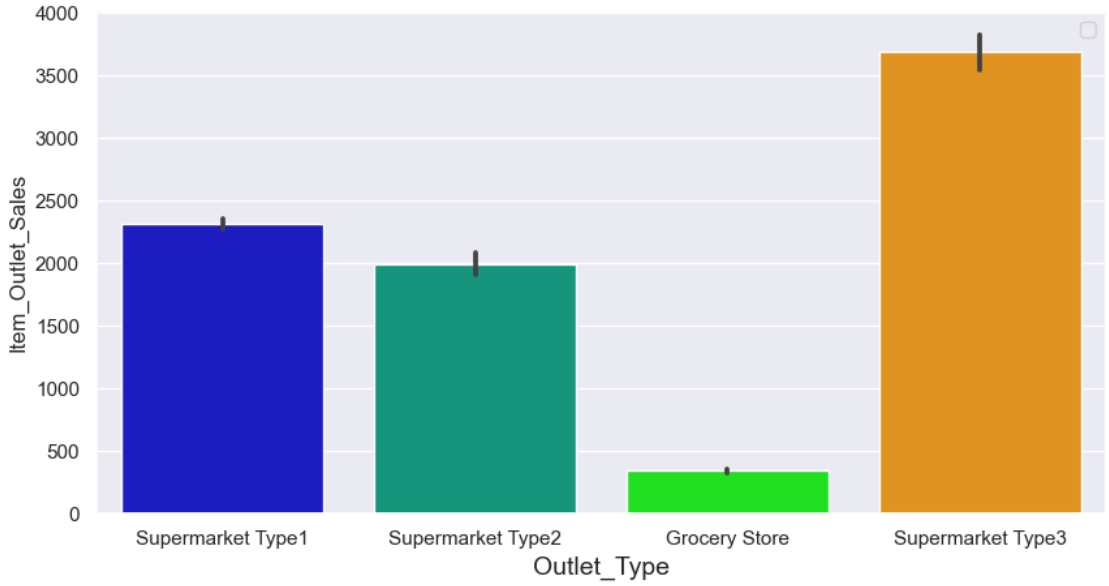
[30]:
```
plt.figure(figsize=(27,10))
sns.barplot('Outlet_Identifier' ,'Item_Outlet_Sales', data=tr_df␣
 ↪,palette='gist_rainbow')
plt.xlabel('Outlet_Identifier', fontsize=14)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



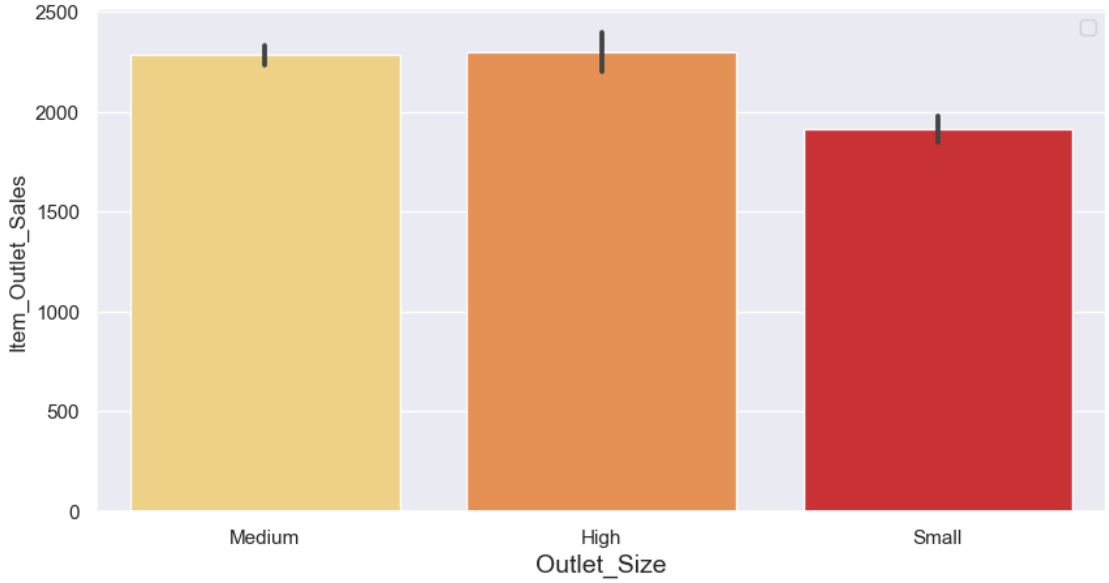
[31]:
```
plt.figure(figsize=(10,5))
sns.barplot('Outlet_Type' ,'Item_Outlet_Sales', data=tr_df␣
 ↪,palette='nipy_spectral')
plt.xlabel('Outlet_Type', fontsize=14)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



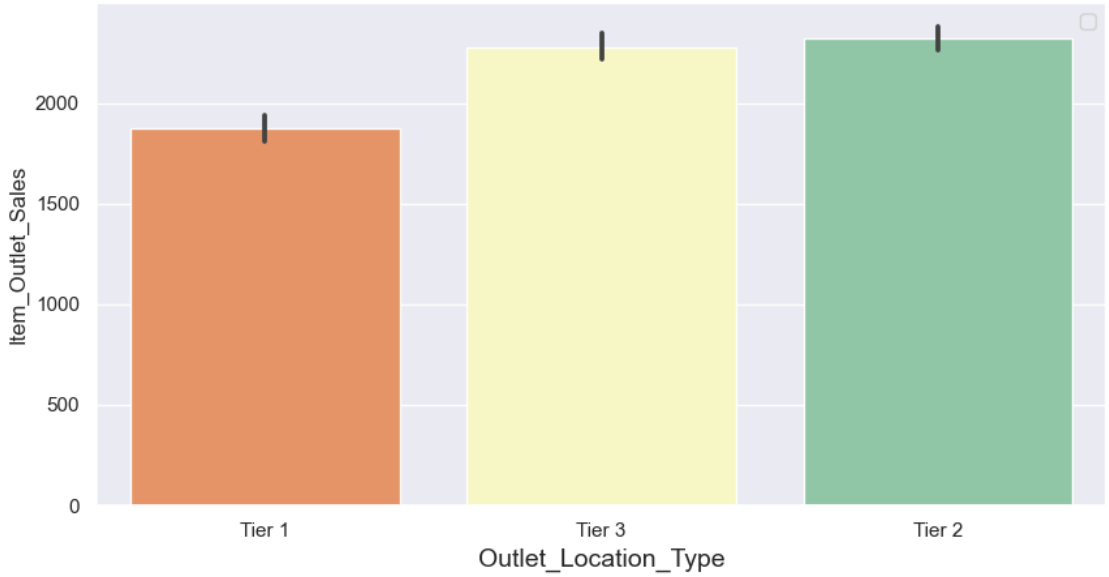
[32]:
```
plt.figure(figsize=(10,5))
sns.barplot('Outlet_Size' ,'Item_Outlet_Sales', data=tr_df ,palette='YlOrRd')
plt.xlabel('Outlet_Size', fontsize=14)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



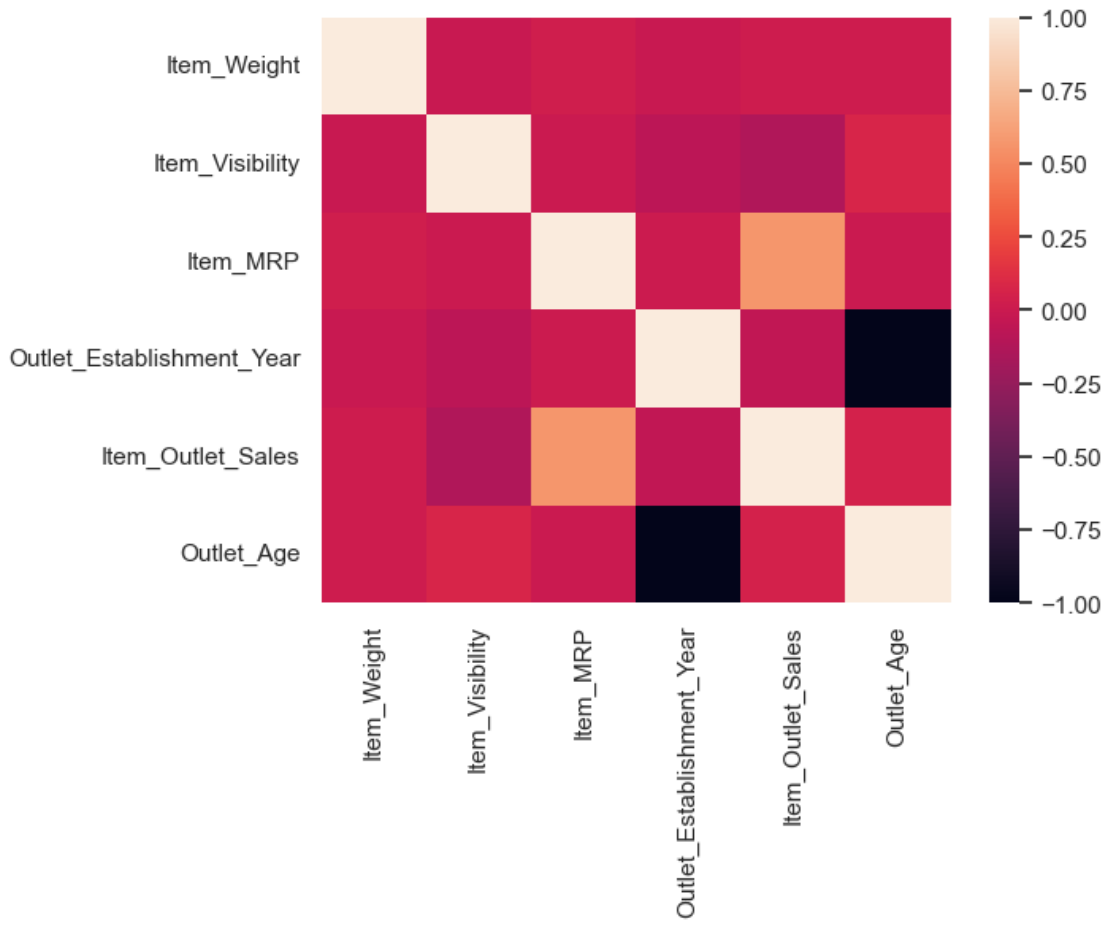
[33]:
```
plt.figure(figsize=(10,5))
sns.barplot('Outlet_Location_Type' ,'Item_Outlet_Sales', data=tr_df␣
 ↪,palette='Spectral')
plt.xlabel('Outlet_Location_Type', fontsize=14)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



[34]:
```
#plotting the correlation matrix
sns.heatmap(tr_df.corr() ,cmap='rocket')
```

[34]:
```
<AxesSubplot:>
```



7 Feature Engineering

[35]:
```
BM_cat.apply(lambda x: x.nunique()) #checking the number of unique values in␣
↪each column
```

[35]:
```
Item_Identifier         1559
Item_Fat_Content           5
Item_Type                 16
Outlet_Identifier         10
Outlet_Size                3
Outlet_Location_Type       3
Outlet_Type                4
dtype: int64
```

[36]:
```
#lable encoding

le = LabelEncoder()
```

```
Label = ['Item_Fat_Content','Outlet_Size','Outlet_Location_Type']

for i in Label:
    tr_df[i] = le.fit_transform(tr_df[i])
    te_df[i] = le.fit_transform(te_df[i])

tr_df.head()
```

[36]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	\
0	FDA15	9.30	0	0.016047	
1	DRC01	5.92	1	0.019278	
2	FDN15	17.50	0	0.016760	
3	FDX07	19.20	1	0.000000	
4	NCD19	8.93	0	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	1	0	
1	2009	1	2	
2	1999	1	0	
3	1998	1	2	
4	1987	0	2	

	Outlet_Type	Item_Outlet_Sales	Outlet_Age
0	Supermarket Type1	3735.1380	21
1	Supermarket Type2	443.4228	11
2	Supermarket Type1	2097.2700	21
3	Grocery Store	732.3800	22
4	Supermarket Type1	994.7052	33

[37]:
```
#one hot encoding
cols = ['Item_Type','Outlet_Type']
# Apply one-hot encoder
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
tr_oh = pd.DataFrame(OH_encoder.fit_transform(tr_df[cols])).astype('int64')
te_oh = pd.DataFrame(OH_encoder.fit_transform(te_df[cols])).astype('int64')


#get feature columns
tr_oh.columns = OH_encoder.get_feature_names(cols)
te_oh.columns = OH_encoder.get_feature_names(cols)
```

```
# One-hot encoding removed index; put it back
tr_oh.index = tr_df.index
te_oh.index = te_df.index

# Add one-hot encoded columns to our main df new name: tr_fe, te_fe (means␣
 ↪feature engeenired)
tr_fe = pd.concat([tr_df, tr_oh], axis=1)
te_fe = pd.concat([te_df, te_oh], axis=1)
```

[38]:
```
# Dropping irrelevant columns

tr_fe  = tr_fe.
 ↪drop(['Item_Identifier','Outlet_Identifier','Outlet_Establishment_Year','Outlet_Type','Item
te_fe = te_fe.
 ↪drop(['Item_Identifier','Outlet_Identifier','Outlet_Establishment_Year','Outlet_Type','Item
```

[39]:
```
tr_fe.head()
```

[39]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	\
0	9.30	0	0.016047	249.8092	1	
1	5.92	1	0.019278	48.2692	1	
2	17.50	0	0.016760	141.6180	1	
3	19.20	1	0.000000	182.0950	1	
4	8.93	0	0.000000	53.8614	0	

	Outlet_Location_Type	Item_Outlet_Sales	Outlet_Age	\
0	0	3735.1380	21	
1	2	443.4228	11	
2	0	2097.2700	21	
3	2	732.3800	22	
4	2	994.7052	33	

	Item_Type_Baking Goods	Item_Type_Breads	…	Item_Type_Meat	\
0	0	0	…	0	
1	0	0	…	0	
2	0	0	…	1	
3	0	0	…	0	
4	0	0	…	0	

	Item_Type_Others	Item_Type_Seafood	Item_Type_Snack Foods	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

```
   Item_Type_Soft Drinks  Item_Type_Starchy Foods  Outlet_Type_Grocery Store  \
0                      0                        0                          0
1                      1                        0                          0
2                      0                        0                          0
3                      0                        0                          1
4                      0                        0                          0

   Outlet_Type_Supermarket Type1  Outlet_Type_Supermarket Type2  \
0                              1                              0
1                              0                              1
2                              1                              0
3                              0                              0
4                              1                              0

   Outlet_Type_Supermarket Type3
0                              0
1                              0
2                              0
3                              0
4                              0

[5 rows x 28 columns]
```

8 Model Development

[40]:
```
y = tr_fe['Item_Outlet_Sales']
X = tr_fe.drop('Item_Outlet_Sales', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.8,␣
 ↪random_state = 0)
```

[41]:
```
def cross_val(model_name,model,X,y,cv):

    scores = CVS(model, X, y, cv=cv)
    print(f'{model_name} Scores:')
    for i in scores:
        print(round(i,2))
    print(f'Average {model_name} score: {round(scores.mean(),4)}')
```

9 Linear Regression

[42]:
```
#model
LR = LinearRegression(normalize=True)

#fit
LR.fit(X_train, y_train)
```

```
#predict
y_predict = LR.predict(X_test)

#score variables
LR_MAE = round(MAE(y_test, y_predict),2)
LR_MSE = round(MSE(y_test, y_predict),2)
LR_R_2 = round(R2(y_test, y_predict),4)
LR_CS  = round(CVS(LR, X, y, cv=5).mean(),4)

print(f" Mean Absolute Error: {LR_MAE}\n")
print(f" Mean Squared Error: {LR_MSE}\n")
print(f" R^2 Score: {LR_R_2}\n")
cross_val(LR,LinearRegression(),X,y,5)
```

```
 Mean Absolute Error: 838.07

 Mean Squared Error: 1286129.02

 R^2 Score: 0.5592

LinearRegression(normalize=True) Scores:
0.57
0.55
0.55
0.56
0.56
Average LinearRegression(normalize=True) score: 0.558
```

[44]:
```
Linear_Regression=pd.DataFrame({'y_test':y_test,'prediction':y_predict})
Linear_Regression.to_csv("Linear Regression.csv")
```

[45]:
```
#model
RFR= RandomForestRegressor(n_estimators=200,max_depth=5,␣
 ↪min_samples_leaf=100,n_jobs=4,random_state=101)
#fit
RFR.fit(X_train, y_train)
#predict
y_predict = RFR.predict(X_test)

#score variables
RFR_MAE = round(MAE(y_test, y_predict),2)
RFR_MSE = round(MSE(y_test, y_predict),2)
RFR_R_2 = round(R2(y_test, y_predict),4)
RFR_CS  = round(CVS(RFR, X, y, cv=5).mean(),4)
```

```
print(f" Mean Absolute Error: {RFR_MAE}\n")
print(f" Mean Squared Error: {RFR_MSE}\n")
print(f" R^2 Score: {RFR_R_2}\n")
cross_val(RFR,RandomForestRegressor(),X,y,5)
```

```
 Mean Absolute Error: 1030.27

 Mean Squared Error: 1964025.66

 R^2 Score: 0.3268

RandomForestRegressor(max_depth=5, min_samples_leaf=100, n_estimators=200,
                      n_jobs=4, random_state=101) Scores:
0.57
0.53
0.52
0.55
0.57
Average RandomForestRegressor(max_depth=5, min_samples_leaf=100,
n_estimators=200,
                      n_jobs=4, random_state=101) score: 0.5472
```

10 Random Forest Regressor

[46]:
```
#model
LS = Lasso(alpha = 0.05)
#fit
LS.fit(X_train,y_train)

#predict
y_predict = LS.predict(X_test)

#score variables
LS_MAE = round(MAE(y_test, y_predict),2)
LS_MSE = round(MSE(y_test, y_predict),2)
LS_R_2 = round(R2(y_test, y_predict),4)
LS_CS  = round(CVS(LS, X, y, cv=5).mean(),4)

print(f" Mean Absolute Error: {LS_MAE}\n")
print(f" Mean Squared Error: {LS_MSE}\n")
print(f" R^2 Score: {LS_R_2}\n")
cross_val(LS,Lasso(alpha = 0.05),X,y,5)
```

```
 Mean Absolute Error: 838.07

 Mean Squared Error: 1285554.86
```

```
 R^2 Score: 0.5594

Lasso(alpha=0.05) Scores:
0.57
0.55
0.55
0.56
0.56
Average Lasso(alpha=0.05) score: 0.5581
```

[47]:
```
Lasso_Regressor=pd.DataFrame({'y_test':y_test,'prediction':y_predict})
Lasso_Regressor.to_csv("Lasso Regressor.csv")
```

11 XGBOOST

[48]:
```
import xgboost as xgb

# model
xgb_model = xgb.XGBRegressor(n_estimators=200, max_depth=5, learning_rate=0.1,␣
 ↪subsample=0.5, colsample_bytree=0.5, random_state=101)

# fit
xgb_model.fit(X_train, y_train)

# predict
y_predict = xgb_model.predict(X_test)

# score variables
XGB_MAE = round(MAE(y_test, y_predict), 2)
XGB_MSE = round(MSE(y_test, y_predict), 2)
XGB_R_2 = round(R2(y_test, y_predict), 4)
XGB_CS = round(CVS(xgb_model, X, y, cv=5).mean(), 4)

print(f" Mean Absolute Error: {XGB_MAE}\n")
print(f" Mean Squared Error: {XGB_MSE}\n")
print(f" R^2 Score: {XGB_R_2}\n")
print(f" Cross Validation Score: {XGB_CS}\n")
```

```
 Mean Absolute Error: 836.48

 Mean Squared Error: 1383190.92

 R^2 Score: 0.5259

 Cross Validation Score: 0.5652
```

[50]:
```
XGB_Regressor = pd.DataFrame({'y_test': y_test, 'prediction': y_predict})
XGB_Regressor.to_csv("XGBoost Regressor.csv")
```

12 Conclusion

[51]:
```
MAE = [LR_MAE, RFR_MAE, LS_MAE, XGB_MAE]
MSE = [LR_MSE, RFR_MSE, LS_MSE, XGB_MSE]
R_2 = [LR_R_2, RFR_R_2, LS_R_2, XGB_R_2]
Cross_score = [LR_CS, RFR_CS, LS_CS, XGB_CS]

Models = pd.DataFrame({
    'models': ["Linear Regression", "Random Forest Regressor", "Lasso␣
 ↪Regressor", "XGBoost Regressor"],
    'MAE': MAE, 'MSE': MSE, 'R^2': R_2, 'Cross Validation Score': Cross_score})

Models.sort_values(by='MAE', ascending=True)
```

[51]:

	models	MAE	MSE	R^2	Cross Validation Score
3	XGBoost Regressor	836.48	1383190.92	0.5259	0.5652
0	Linear Regression	838.07	1286129.02	0.5592	0.5580
2	Lasso Regressor	838.07	1285554.86	0.5594	0.5581
1	Random Forest Regressor	1030.27	1964025.66	0.3268	0.5920