Importing relavant Libraries

In [1]:

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

In [2]:

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

sns.set()
%matplotlib inline
```

In [3]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection  import cross_val_score
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

In [4]:

```
train = pd.read_csv('Train.csv')
test = pd.read_csv('Test.csv')
```

Data Inspection

In [5]:

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

Out[5]:

```
((8523, 12), (5681, 11))
```

As said above we have 8523 rows and 12 columns in Train set whereas Test set has 5681 rows and 11 columns.

In [6]:

```
test.apply(lambda x: sum(x.isnull()))
```

Out[6]:

```
Item_Identifier                 0
Item_Weight                   976
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  1606
Outlet_Location_Type            0
Outlet_Type                     0
dtype: int64
```

In [7]:

```
test.isnull().sum()/test.shape[0] *100
```

Out[7]:

```
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
```

We have 17% and 28% of missing values in Item weight and Outlet_Size columns respectively

In [8]:

```
train.info()
```

```
<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
```

In [9]:

```
categorical = train.select_dtypes(include =[np.object])
print("Categorical Features in Train Set:",categorical.shape[1])

numerical= train.select_dtypes(include =[np.float64,np.int64])
print("Numerical Features in Train Set:",numerical.shape[1])
```

```
Categorical Features in Train Set: 7
Numerical Features in Train Set: 5
```

In [10]:

```
categorical = test.select_dtypes(include =[np.object])
print("Categorical Features in Test Set:",categorical.shape[1])

numerical= test.select_dtypes(include =[np.float64,np.int64])
print("Numerical Features in Test Set:",numerical.shape[1])
```

```
Categorical Features in Test Set: 7
Numerical Features in Test Set: 4
```

In [11]:

```
train.describe()
```

Out[11]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

In [12]:

```
test.describe()
```

Out[12]:

	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

Data Cleaning

1. Item Size

In [13]:

```
train.columns
```

Out[13]:

```
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 [14]:

```
train['Item_Weight'].isnull().sum(),test['Item_Weight'].isnull().sum()
```

Out[14]:

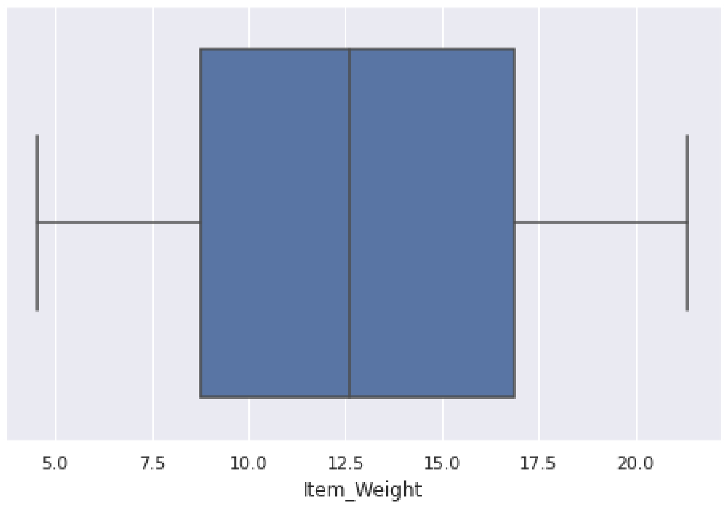
```
(1463, 976)
```

In [15]:

```
plt.figure(figsize=(8,5))
sns.boxplot('Item_Weight', data=train)
```

Out[15]:

```
<AxesSubplot:xlabel='Item_Weight'>
```

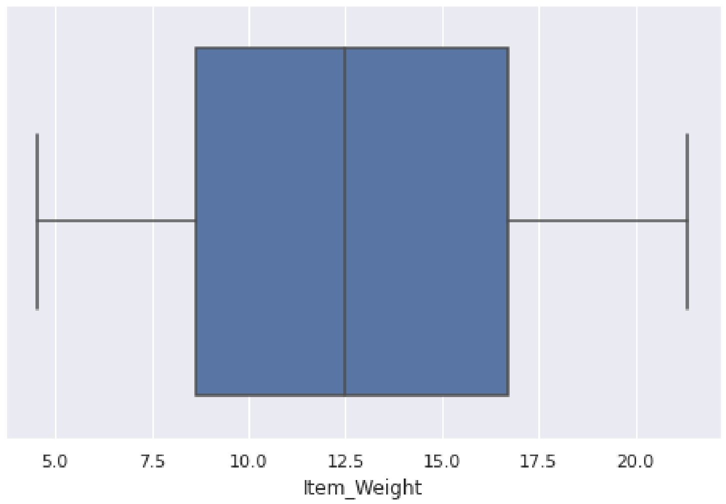


In [16]:

```
plt.figure(figsize=(8,5))
sns.boxplot('Item_Weight', data=test)
```

Out[16]:

```
<AxesSubplot:xlabel='Item_Weight'>
```



The Box Plots above clearly show no "Outliers" and hence we can impute the missing values with "Mean"

In [17]:

```
train['Item_Weight']= train['Item_Weight'].fillna(train['Item_Weight'].mean())
test['Item_Weight']= test['Item_Weight'].fillna(test['Item_Weight'].mean())
```

In [18]:

```
train['Item_Weight'].isnull().sum(),test['Item_Weight'].isnull().sum()
```

Out[18]:

```
(0, 0)
```

We have succesfully imputed the missing values from the column Item_Weight.

2. Outlet_Size

In [19]:

```
train['Outlet_Size'].isnull().sum(),test['Outlet_Size'].isnull().sum()
```

Out[19]:

```
(2410, 1606)
```

In [20]:

```
print(train['Outlet_Size'].value_counts())
print('******************************************')
print(test['Outlet_Size'].value_counts())
```

```
Medium    2793
Small     2388
High       932
Name: Outlet_Size, dtype: int64
******************************************
Medium    1862
Small     1592
High       621
Name: Outlet_Size, dtype: int64
```

Since the outlet_size is a categorical column, we can impute the missing values by "Mode"(Most Repeated Value) from the column.

In [21]:

```
train['Outlet_Size']= train['Outlet_Size'].fillna(train['Outlet_Size'].mode()[0])
test['Outlet_Size']= test['Outlet_Size'].fillna(test['Outlet_Size'].mode()[0])
```

In [22]:

```
train['Outlet_Size'].isnull().sum(),test['Outlet_Size'].isnull().sum()
```

Out[22]:

```
(0, 0)
```

We have succesfully imputed the missing values from the column Outlet_Size.

Exploratory Data Analysis

In [23]:

```
train.head()
```

Out[23]:

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

In [24]:

```
train['Item_Fat_Content'].value_counts()
```

Out[24]:

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

We see there are some irregularities in the column and it is needed to fix them

In [25]:

```
train['Item_Fat_Content'].replace(['low fat','LF','reg'],['Low Fat','Low Fat','Regular'],inplace =
True)
test['Item_Fat_Content'].replace(['low fat','LF','reg'],['Low Fat','Low Fat','Regular'],inplace =
True)
```

In [26]:

```
train['Item_Fat_Content']= train['Item_Fat_Content'].astype(str)
```

In [27]:

```
train['Years_Established'] = train['Outlet_Establishment_Year'].apply(lambda x: 2020 - x)
test['Years_Established'] = test['Outlet_Establishment_Year'].apply(lambda x: 2020 - x)
```

In [28]:

```
train.head()
```

Out[28]:

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

Univariate Analysis

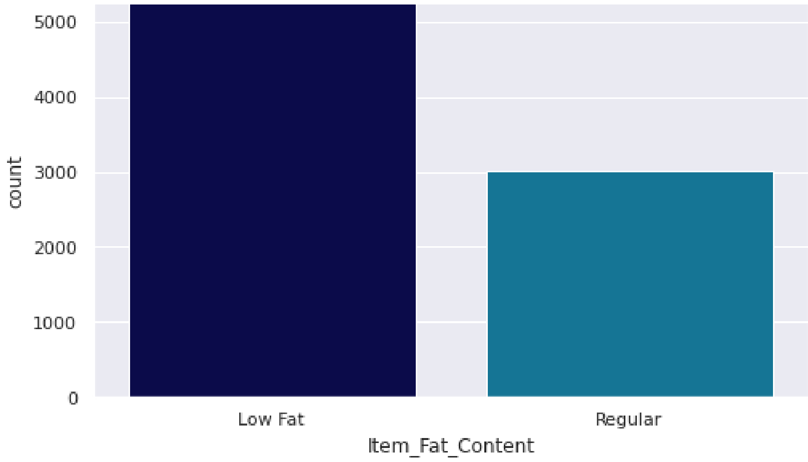
1. Item fat content

In [29]:

```
plt.figure(figsize=(8,5))
sns.countplot('Item_Fat_Content',data=train,palette='ocean')
```

Out[29]:

```
<AxesSubplot:xlabel='Item_Fat_Content', ylabel='count'>
```



Observations:

1. Low fat items are bought more than regular

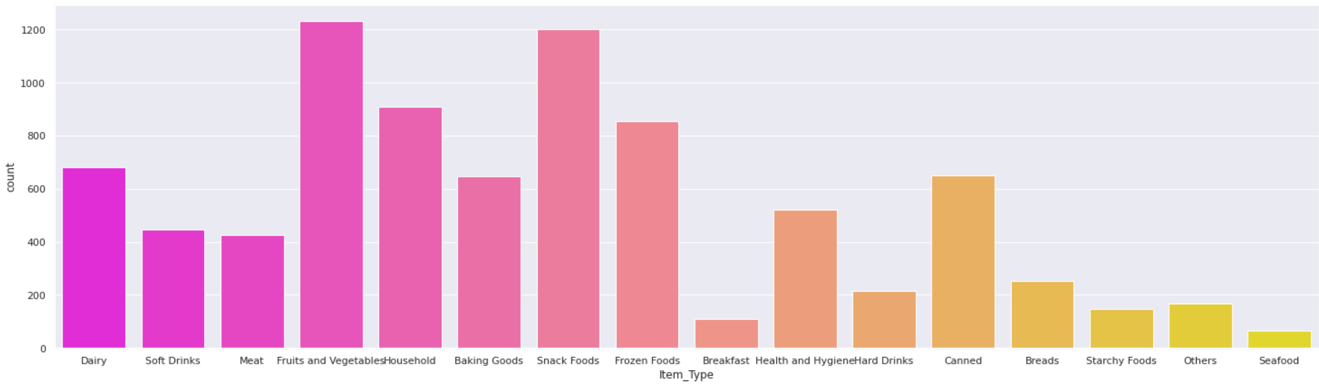
2. Item Type

In [30]:

```
plt.figure(figsize=(25,7))
sns.countplot('Item_Type',data=train,palette='spring')
```

Out[30]:

```
<AxesSubplot:xlabel='Item_Type', ylabel='count'>
```



Observations:

1. Fruits and vegetables are largely sold as people tend to use them on a daily basis
2. Snack food too have a good sale.

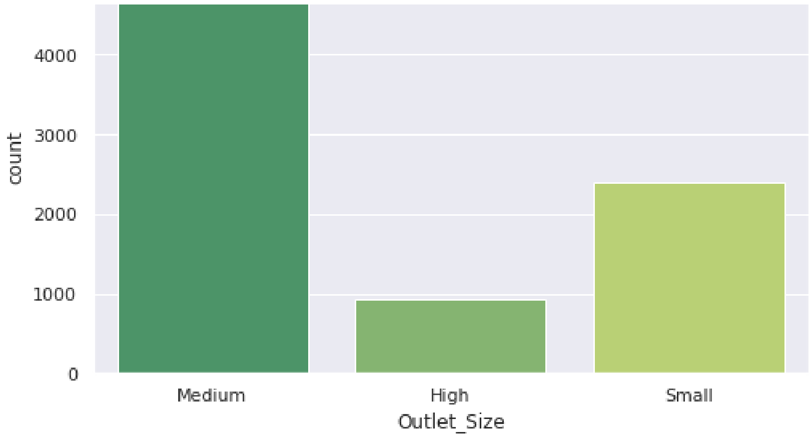
3. Outlet Size

In [31]:

```
plt.figure(figsize=(8,5))
sns.countplot('Outlet_Size',data=train,palette='summer')
```

Out[31]:

```
<AxesSubplot:xlabel='Outlet_Size', ylabel='count'>
```



Observations:

1. Te Outlets are more of Medium size

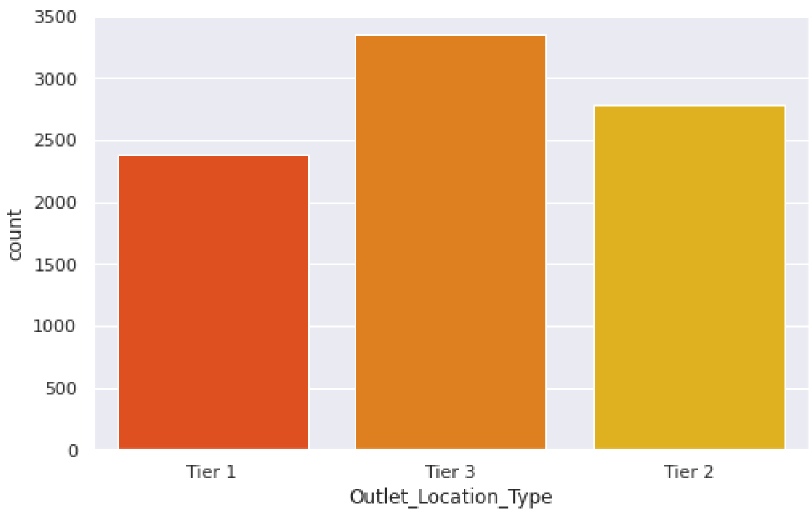
4. Outlet location type

In [32]:

```
plt.figure(figsize=(8,5))
sns.countplot('Outlet_Location_Type',data=train,palette='autumn')
```

Out[32]:

```
<AxesSubplot:xlabel='Outlet_Location_Type', ylabel='count'>
```



Observations:

```
1. Outlets are maximum in number in Tier 3 cities
```

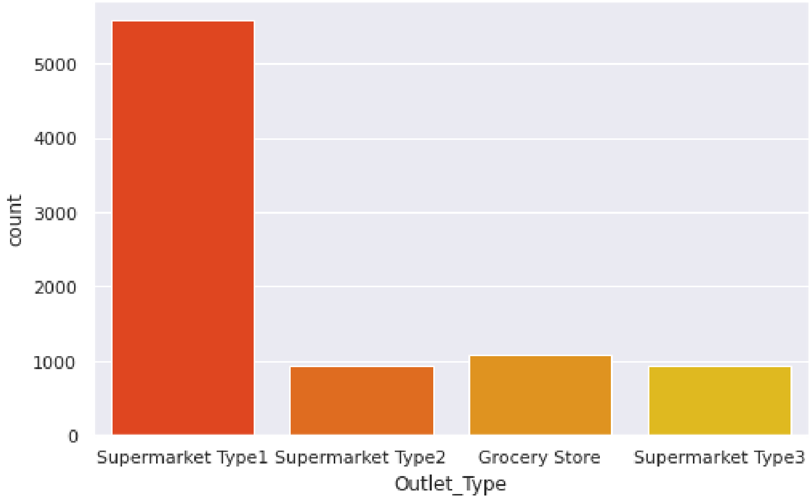
5. Outlet Type

In [33]:

```
plt.figure(figsize=(8,5))
sns.countplot('Outlet_Type',data=train,palette='autumn')
```

Out[33]:

```
<AxesSubplot:xlabel='Outlet_Type', ylabel='count'>
```



Observations:

```
1. The outlets are more of Supermarket Type 1
```

Bivariate Analysis

In [34]:

```
train.columns
```

Out[34]:

```
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', 'Years_Established'],
      dtype='object')
```

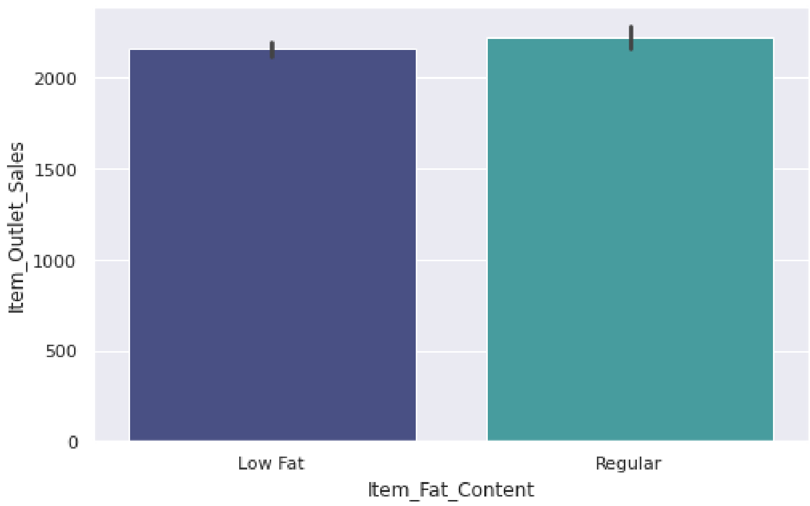
1. Item fat

In [35]:

```
plt.figure(figsize=(8,5))
sns.barplot('Item_Fat_Content', 'Item_Outlet_Sales', data=train, palette='mako')
```

Out[35]:

```
<AxesSubplot:xlabel='Item_Fat_Content', ylabel='Item_Outlet_Sales'>
```



Observations

1. Both low and regular fat conten items have high sales

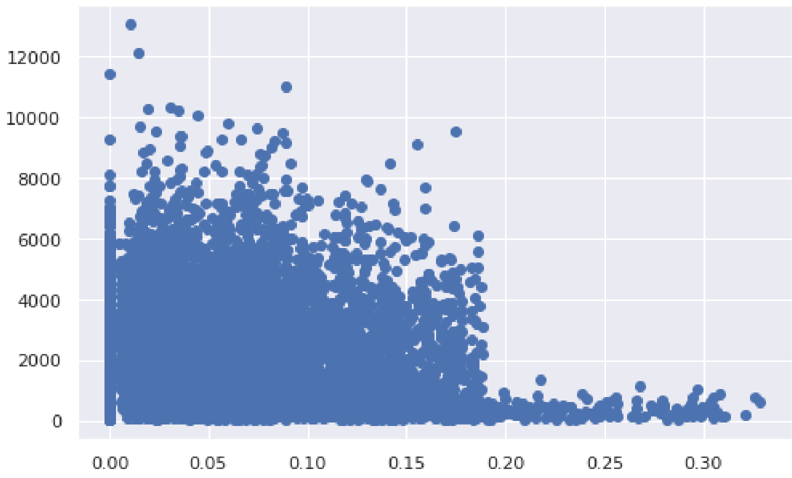
2. Item Visibility

In [36]:

```
plt.figure(figsize=(8,5))
plt.scatter('Item_Visibility','Item_Outlet_Sales', data=train)
```

Out[36]:

```
<matplotlib.collections.PathCollection at 0x7f018ce0e130>
```



Observations

1. Item visibility has a minimum value of 0.This makes n practical sense coz when a product is being sold in a store, its visibility cannot be 0.

Let us consider as a missing value and impute it by mean visibility value of that item.

In [37]:

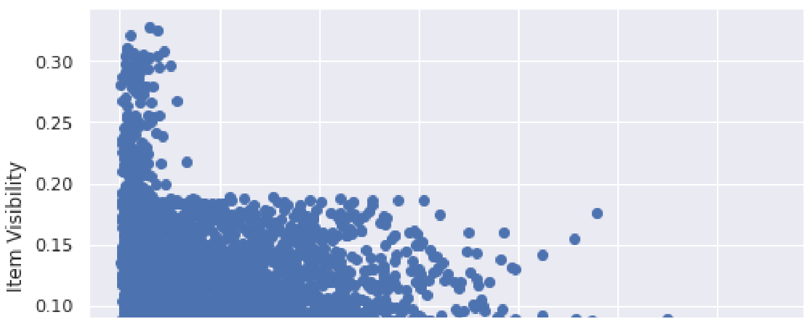
```
train['Item_Visibility']= train['Item_Visibility'].replace(0,train['Item_Visibility'].mean())
test['Item_Visibility']= test['Item_Visibility'].replace(0,test['Item_Visibility'].mean())
```

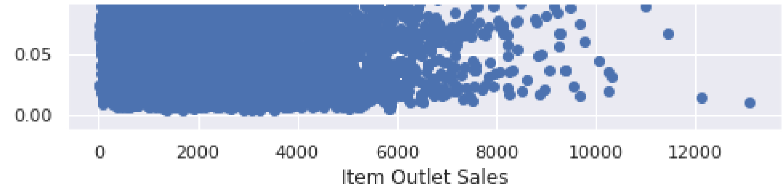
In [38]:

```
plt.figure(figsize=(8,5))
plt.scatter(y='Item_Visibility',x='Item_Outlet_Sales',data=train)
plt.xlabel('Item Outlet Sales')
plt.ylabel('Item Visibility')
```

Out[38]:

```
Text(0, 0.5, 'Item Visibility')
```





We can see that now visibility is not exactly zero and it has some value indicating that Item is rarely purchased by the customers.

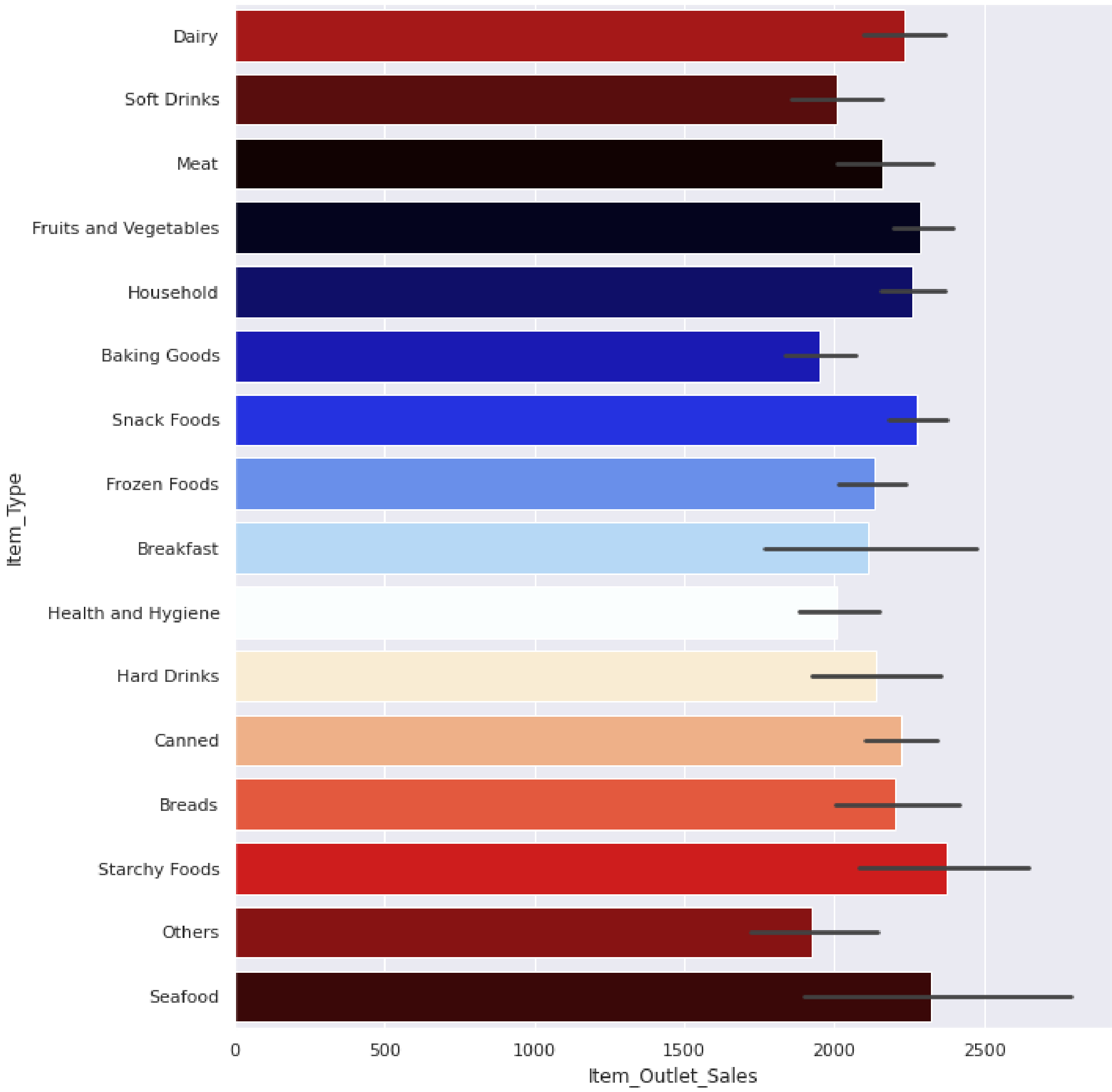
3. Item Type

In [40]:

```
plt.figure(figsize=(10,12))
sns.barplot(y='Item_Type', x='Item_Outlet_Sales', data=train, palette='flag')
```

Out[40]:

```
<AxesSubplot:xlabel='Item_Outlet_Sales', ylabel='Item_Type'>
```



The products available were Fruits-Veggies and Snack Foods but the sales of Seafood and Starchy Foods seems higher and hence the sales can be improved with having stock of products that are most bought by customers.

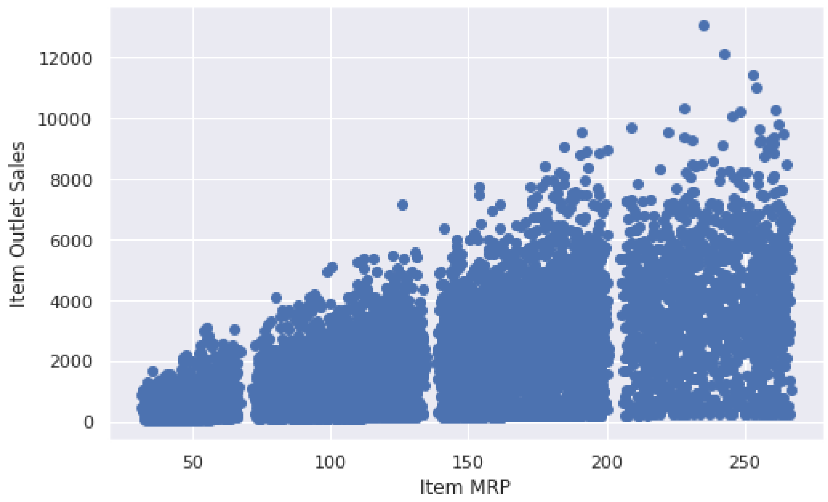
In [41]:

```
plt.figure(figsize=(8,5))
plt.scatter(y='Item_Outlet_Sales',x='Item_MRP',data=train)
```

```
plt.xlabel('Item MRP')
plt.ylabel('Item Outlet Sales')
```

Out[41]:

```
Text(0, 0.5, 'Item Outlet Sales')
```



Observation

1. Items MRP ranging from 200-250 dollars is having high Sales.

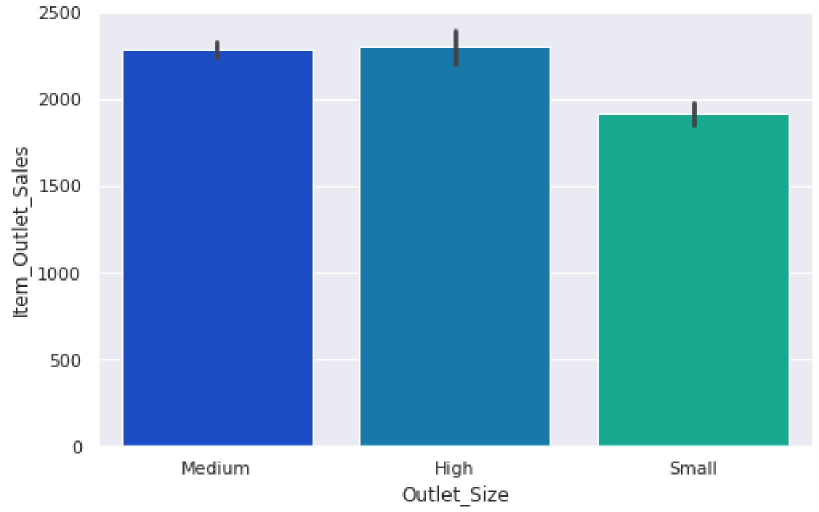
4. Outlet Size

In [42]:

```
plt.figure(figsize=(8,5))
sns.barplot(x='Outlet_Size',y='Item_Outlet_Sales',data=train,palette='winter')
```

Out[42]:

```
<AxesSubplot:xlabel='Outlet_Size', ylabel='Item_Outlet_Sales'>
```



Observations:

1. Sales is greater for medium and high outlet size

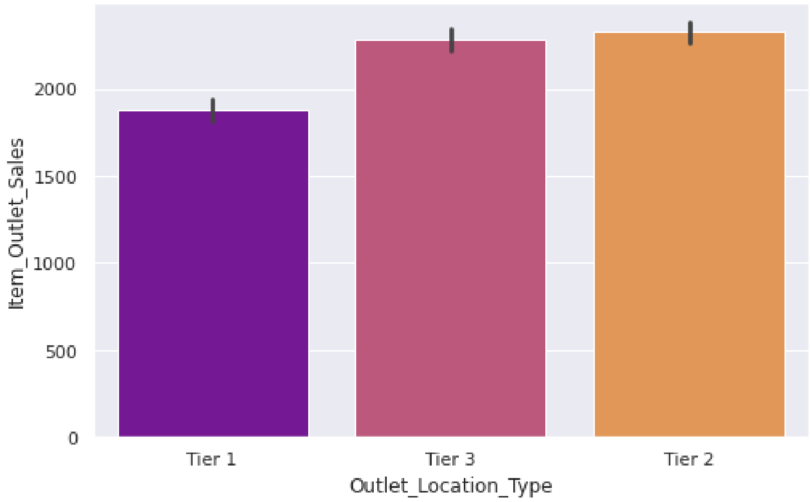
5.Outlet Location Type

In [44]:

```
plt.figure(figsize=(8,5))
sns.barplot(x='Outlet_Location_Type',y='Item_Outlet_Sales',data=train,palette='plasma')
```

Out[44]:

<AxesSubplot:xlabel='Outlet_Location_Type', ylabel='Item_Outlet_Sales'>



Obseravtions:

1. The Outlet Sales tend to be high for Tier3 and Tier 2 location types but we have only Tier3 locations maximum Outlets.

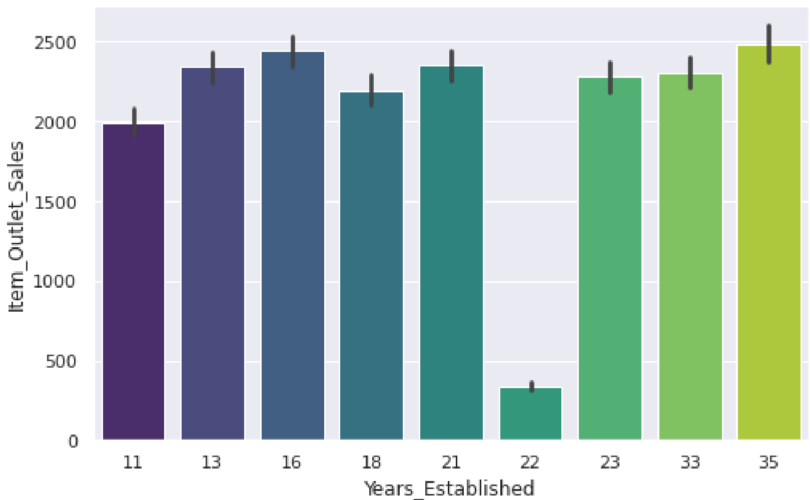
6. Years established

In [45]:

```
plt.figure(figsize=(8,5))
sns.barplot(x='Years_Established',y='Item_Outlet_Sales',data=train,palette='viridis')
```

Out[45]:

<AxesSubplot:xlabel='Years_Established', ylabel='Item_Outlet_Sales'>



Observations:

1. It is quiet evident that Outlets established 35 years before is having good Sales margin.
2.We also have a outlet which was established before 22 years has the lowest sales margin, so established years wouldn't improve the Sales unless the products are sold according to customer's interest.

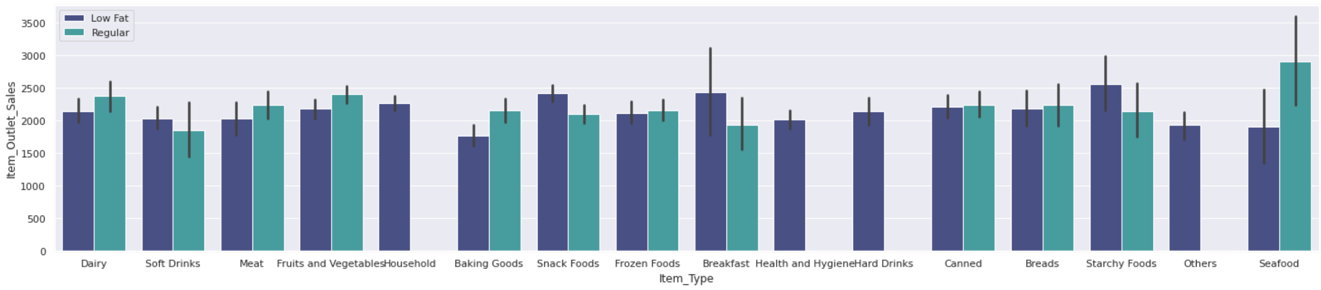
Multivariate Analysis

In [46]:

```
plt.figure(figsize=(25,5))
sns.barplot('Item_Type','Item_Outlet_Sales',hue='Item_Fat_Content',data=train,palette='mako')
plt.legend()
```

Out[46]:

```
<matplotlib.legend.Legend at 0x7f018c71b070>
```

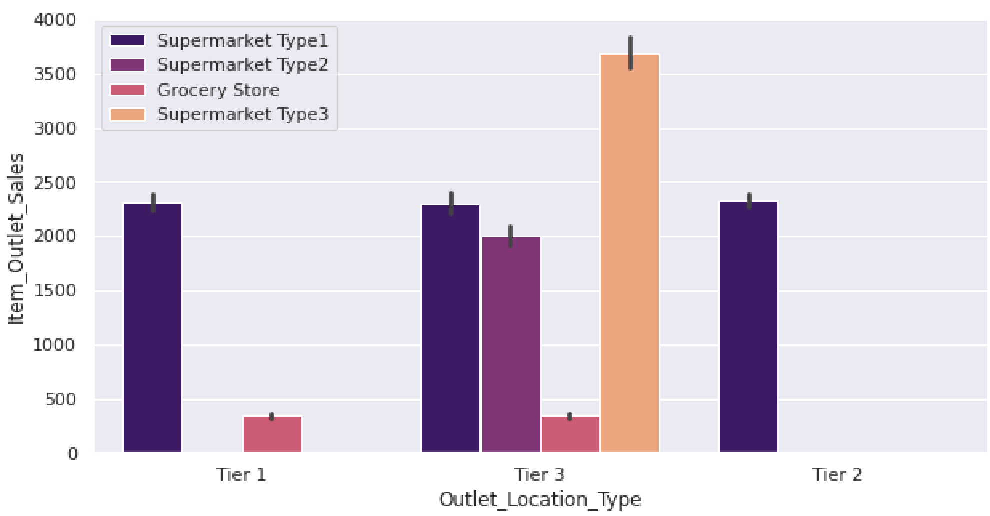


In [47]:

```
plt.figure(figsize=(10,5))
sns.barplot('Outlet_Location_Type','Item_Outlet_Sales',hue='Outlet_Type',data=train,palette='magma'
)
plt.legend()
```

Out[47]:

```
<matplotlib.legend.Legend at 0x7f018c71b3d0>
```



Observations:

1. The Tier-3 location type has all types of Outlet type and has high sales margin.

Feature Engineering

In [48]:

```
train.head()
```

Out[48]:

Out[46]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	O
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	
3	FDX07	19.20	Regular	0.066132	Fruits and Vegetables	182.0950	OUT010	1998	
4	NCD19	8.93	Low Fat	0.066132	Household	53.8614	OUT013	1987	

In [56]:

```
le = LabelEncoder()
var_mod = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Outlet_Type','Item_Type']

for i in var_mod:
    train[i] = le.fit_transform(train[i])

for i in var_mod:
    test[i] = le.fit_transform(test[i])
```

In [57]:

```
train.head()
```

Out[57]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_S
0	9.30	0	0.016047	4	249.8092	1	0	1	3735.1
1	5.92	1	0.019278	14	48.2692	1	2	2	443.4
2	17.50	0	0.016760	10	141.6180	1	0	1	2097.2
3	19.20	1	0.066132	6	182.0950	1	2	0	732.3
4	8.93	0	0.066132	9	53.8614	0	2	1	994.7

There are some columns that needs to be dropped as they don't seem helping our analysis.

In [51]:

```
train = train.drop(['Item_Identifier','Outlet_Identifier','Outlet_Establishment_Year'],axis=1)
test= test.drop(['Item_Identifier','Outlet_Identifier','Outlet_Establishment_Year'],axis=1)
```

In [52]:

```
train.columns
```

Out[52]:

```
Index(['Item_Weight', 'Item_Fat_Content', 'Item_Visibility', 'Item_Type',
       'Item_MRP', 'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type',
       'Item_Outlet_Sales', 'Years_Established'],
      dtype='object')
```

In [58]:

```
X= train[['Item_Weight','Item_Fat_Content','Item_Visibility','Item_Type','Item_MRP','Outlet_Size',
'Outlet_Location_Type','Outlet_Type','Years_Established']]
y= train['Item_Outlet_Sales']
```

In [59]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=22)
```

Feature Scaling

In [60]:

```
features= ['Item_Weight','Item_Fat_Content','Item_Visibility','Item_Type','Item_MRP','Outlet_Size'
,'Outlet_Location_Type','Outlet_Type','Years_Established']
```

Linear Regression

Preparing the model and importing necessary packages

In [62]:

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
```

Fitting the model

In [64]:

```
reg.fit(X_train,y_train)
```

Out[64]:

```
LinearRegression()
```

Finding accuracy of Linear regression model

In [65]:

```
reg.score(X_test,y_test)
```

Out[65]:

```
0.4946245671867815
```

Gradient Boosting Regressor

Preparing the model and importing necessary packages

In [66]:

```
from sklearn.ensemble import GradientBoostingRegressor
grad= GradientBoostingRegressor(n_estimators=100)
```

Fitting the model

In [67]:

```
grad.fit(X_train,y_train)
```

Out[67]:

```
GradientBoostingRegressor()
```

Finding the accuracy of Gradient Boosting Regressor

In [68]:

```
grad.score(X_test,y_test)
```

Out[68]:

```
0.5713935192940436
```

Random Forest Regressor

Preparing the model and importing necessary pacakges

In [69]:

```
from sklearn.ensemble import RandomForestRegressor
ran=RandomForestRegressor(n_estimators=50)
```

Fitting the model

In [70]:

```
ran.fit(X_train,y_train)
```

Out[70]:

```
RandomForestRegressor(n_estimators=50)
```

Finding accuracy of Random Forest Model

In [71]:

```
ran.score(X_test,y_test)
```

Out[71]:

```
0.5216971567098128
```

Conclusion

We are given a Big_Mart dataset The aim is to build a predictive model and find out the sales of each product at a particular store. Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales.

First we explore the data performing EDA using various data vizualization tools and draw the necessary conclusions from univariate,bivariate and multivariate analysis

After EDA, we perform feature engineering and feature scaling. Intead of using one-hot encoder, we instead use label encoder as the categorical data has been handeled and using one-hot encoder wont make much difference.We then built three models over our datasets and find which one performs the best.

Linear regression accuray score: 0.4946245671867815

GradientBoostingRegressor accuracy Score: 0.5713935192940436

RandomForestRegressor accuracy score: 0.5216971567098128

From the above results we conclude that GradientBoostingRegressor has performed the best, thus boosting algorithms efficient for most of the predictive cases.

In [ ]:

```

```