Big Mart Sales

1. Loading important files

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

2. Loading Dataset

In [2]:
```
df_train=pd.read_csv(r'D:\New folder\New folder (2)\bigmart\Train.csv')
df_test =pd.read_csv(r'D:\New folder\New folder (2)\bigmart\Test.csv')
```

In [3]:
```
df_train.head()
```

Out[3]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	M
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	M
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	M
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 [4]:
```
df_test.head()
```

Out[4]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	M
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	2007	
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	1998	
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	2007	
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	1985	M

. Checking Information

In [5]:
```
df_train.shape
```

Out[5]: (8523, 12)

In [6]:
```
df_test.shape
```

Out[6]: (5681, 11)

Checking Describe

In [7]:

```
1 df_train.describe()
```

Out[7]:

	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 [8]:

```
1 df_test.describe()
```

Out[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

Checking Info

In [9]:
```
df_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 [10]:

```
df_test.info()
```

```
<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                4705 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                4075 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
```

. Checking missing values

In [11]:

```
df_train.isnull().sum()
```

Out[11]:

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

In [12]:
```
df_test.isnull().sum()
```

Out[12]:
```
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
```

Filling Missing Values

In [13]:
```
# 'Item_Weight' has a numerical values so will will fill it by using mean imputation
df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
df_test['Item_Weight'].fillna(df_test['Item_Weight'].mean(),inplace=True)
```

In [14]:
```
df_train['Item_Weight'].mode()
```

Out[14]:
```
0    12.857645
dtype: float64
```

In [15]:
```
# Outlet_Size has a categorical values so we will fill it by using mode imputation
df_test['Outlet_Size'].mode()
```

Out[15]:
```
0    Medium
dtype: object
```

In [16]:
```
df_train['Outlet_Size'].fillna(df_train['Outlet_Size'].mode()[0],inplace=True)
```

In [17]:
```
df_test['Outlet_Size'].fillna(df_test['Outlet_Size'].mode()[0],inplace=True)
```

In [18]:
```
df_train.isnull().sum()
```

Out[18]:
```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

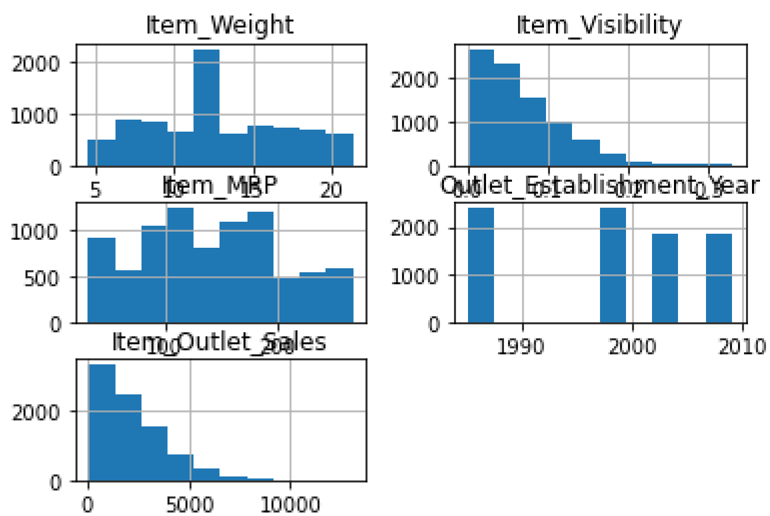
Data Reduction

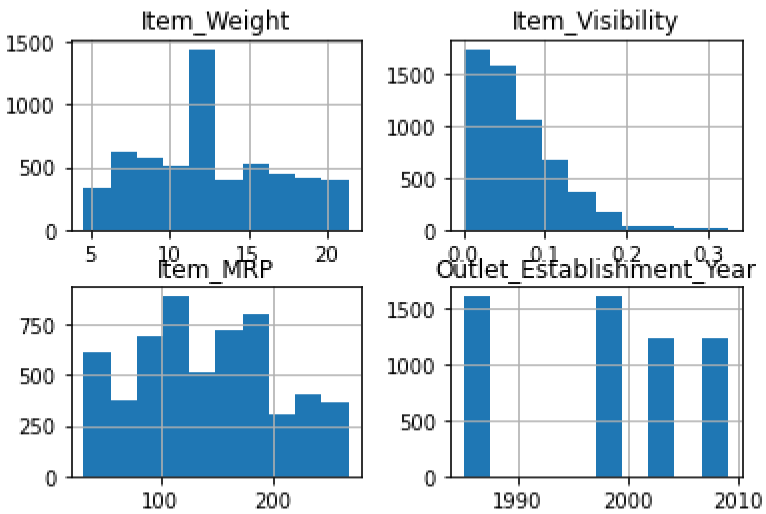
In [19]:
```
# We dont need the two columns i.e, ['Item_Identifier'] ,['Outlet_Identifier'] so we will drop this columns
df_train.drop(columns=(['Item_Identifier','Outlet_Identifier']),inplace=True)
df_test.drop(columns=(['Item_Identifier','Outlet_Identifier']),inplace=True)
```

EDA(Exploratory Data Analysis)

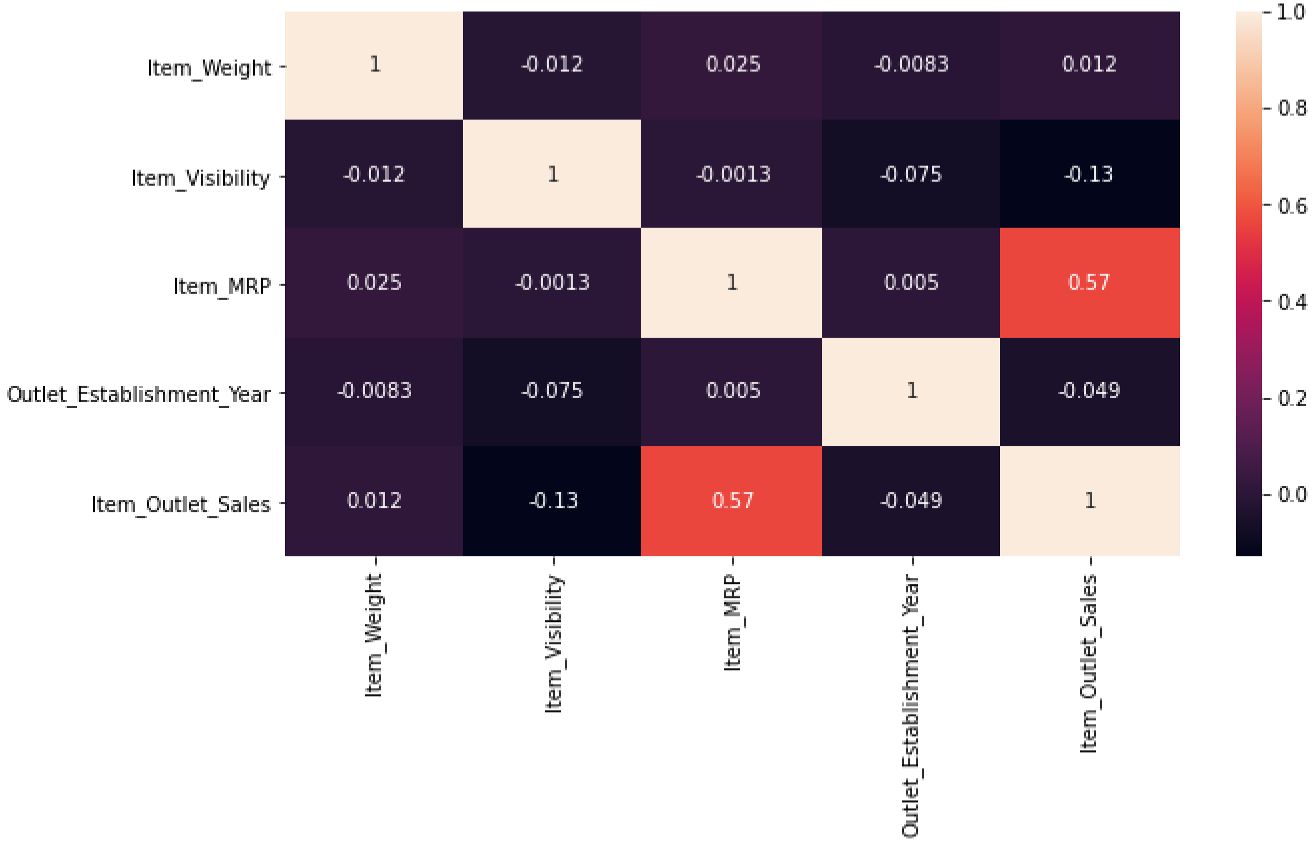
In [20]:
```
df_train.hist()   ,df_test.hist()
```

Out[20]:
```
(array([[<AxesSubplot:title={'center':'Item_Weight'}>,
         <AxesSubplot:title={'center':'Item_Visibility'}>],
        [<AxesSubplot:title={'center':'Item_MRP'}>,
         <AxesSubplot:title={'center':'Outlet_Establishment_Year'}>],
        [<AxesSubplot:title={'center':'Item_Outlet_Sales'}>,
         <AxesSubplot:>]], dtype=object),
 array([[<AxesSubplot:title={'center':'Item_Weight'}>,
         <AxesSubplot:title={'center':'Item_Visibility'}>],
        [<AxesSubplot:title={'center':'Item_MRP'}>,
         <AxesSubplot:title={'center':'Outlet_Establishment_Year'}>]],
       dtype=object))
```




Item_Weight
1500
1000
500
0
5
10
15
20
Item_Visibility
1500
1000
500
0
0.0
0.1
0.2
0.3
Item_MRP
750
500
250
0
100
200
Outlet_Establishment_Year
1500
1000
500
0
1990
2000
2010

In [21]:
```
plt.figure(figsize =(10,5))
sns.heatmap(df_train.corr(),annot=True)
plt.show()
```



In [22]:

```
plt.figure(figsize=(10,5))
sns.heatmap(df_test.corr(),annot=True)
plt.show()
```

In [23]:

```
df_train.head()
```

Out[23]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Ou
0	9.30	Low Fat	0.016047	Dairy	249.8092	1999	Medium	Tier 1	Su
1	5.92	Regular	0.019278	Soft Drinks	48.2692	2009	Medium	Tier 3	Su
2	17.50	Low Fat	0.016760	Meat	141.6180	1999	Medium	Tier 1	Su
3	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	1998	Medium	Tier 3	
4	8.93	Low Fat	0.000000	Household	53.8614	1987	High	Tier 3	Su

Preprocessing task before model building

Label Encoding

In [24]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [25]:
```
df_train['Item_Fat_Content'] = le.fit_transform(df_train['Item_Fat_Content'])
df_train['Item_Type'] = le.fit_transform(df_train['Item_Type'])
df_train['Outlet_Size'] = le.fit_transform(df_train['Outlet_Size'])
df_train['Outlet_Type'] = le.fit_transform(df_train['Outlet_Type'])
df_train['Outlet_Location_Type'] = le.fit_transform(df_train['Outlet_Location_Type'])
```

In [26]:
```
df_train
```

Out[26]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type
0	9.300	1	0.016047	4	249.8092	1999	1	0
1	5.920	2	0.019278	14	48.2692	2009	1	2
2	17.500	1	0.016760	10	141.6180	1999	1	0
3	19.200	2	0.000000	6	182.0950	1998	1	2
4	8.930	1	0.000000	9	53.8614	1987	0	2
...	...	...	...	...	...	...	...	...
8518	6.865	1	0.056783	13	214.5218	1987	0	2
8519	8.380	2	0.046982	0	108.1570	2002	1	1
8520	10.600	1	0.035186	8	85.1224	2004	2	1
8521	7.210	2	0.145221	13	103.1332	2009	1	2
8522	14.800	1	0.044878	14	75.4670	1997	2	0

8523 rows × 10 columns

Splitting our data into train and test

In [27]:
```
x=df_train.drop(['Item_Outlet_Sales'],axis=1)
y=df_train['Item_Outlet_Sales']
```

In [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=101)
```

Standarisation

In [30]:
```
x.describe()
```

Out[30]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.00
mean	12.857645	1.369354	0.066132	7.226681	140.992782	1997.831867	1.170832	1.11
std	4.226124	0.644810	0.051598	4.209990	62.275067	8.371760	0.600327	0.81
min	4.555000	0.000000	0.000000	0.000000	31.290000	1985.000000	0.000000	0.00
25%	9.310000	1.000000	0.026989	4.000000	93.826500	1987.000000	1.000000	0.00
50%	12.857645	1.000000	0.053931	6.000000	143.012800	1999.000000	1.000000	1.00
75%	16.000000	2.000000	0.094585	10.000000	185.643700	2004.000000	2.000000	2.00
max	21.350000	4.000000	0.328391	15.000000	266.888400	2009.000000	2.000000	2.00

In [31]:
```
# Here our data are not close like Item_Weight is very high where  Outlet_Location_Type is very small. To o
```

In [32]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [33]:
```
x_train_std=sc.fit_transform(x_train)
```

In [34]:
```
x_test_std=sc.fit_transform(x_test)
```

In [35]:
```
x_train_std
```

Out[35]:
```
array([[ 1.52290029, -0.57382672,  0.68469729, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.23985603, -0.57382672, -0.09514748, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 1.54667616,  0.97378032, -0.00838589, ..., -0.28872895,
        -0.13870429, -0.25964107],
       ...,
       [-0.08197107, -0.57382672, -0.9191623 , ...,  1.37953713,
        -1.36527477, -0.25964107],
       [-0.74888428,  0.97378032,  1.21363058, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 0.67885683, -0.57382672,  1.83915356, ..., -0.28872895,
         1.08786619,  0.98524841]])
```

In [36]:
```
y_train
```

Out[36]:
```
3684     163.7868
1935    1607.2412
5142    1510.0344
4978    1784.3440
2299    3558.0352
          ...
599     5502.8370
5695    1436.7964
8006    2167.8448
1361    2700.4848
1547     829.5868
Name: Item_Outlet_Sales, Length: 6818, dtype: float64
```

In [37]:
```
y_test
```

Out[37]:
```
8179     904.8222
8355    2795.6942
3411    1947.4650
7089     872.8638
6954    2450.1440
           ...
1317    1721.0930
4996     914.8092
531      370.1848
3891    1358.2320
6629    2418.1856
Name: Item_Outlet_Sales, Length: 1705, dtype: float64
```

In [38]:
```
import joblib
```

In [ ]:
```

```

Model Building

In [39]:
```
# Using Alogithm of Linear Regresion
```

In [40]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
```

In [41]:
```
lr.fit(x_train_std,y_train)
```

Out[41]:
```
LinearRegression()
```

In [42]:
```
y_pred_lr=lr.predict(x_test_std)
```

In [43]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [44]:
```
print(r2_score(y_test,y_pred_lr))
print( mean_absolute_error(y_test,y_pred_lr))
print(np.sqrt(mean_squared_error(y_test,y_pred_lr)))
```

```
0.5020054018117118
885.7810720872159
1164.9965315232587
```

In [45]:
```
# Using Alogithm of Random Forest Regresion
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=1000) #wrote after finding the valuye of n_estimator by applying
                                            #hyperparameter tuning
```

In [ ]:
```
rf.fit(x_train,y_train)
```

In [ ]:
```
y_pred_rf=rf.predict(x_test)
```

In [ ]:
```
print(r2_score(y_test,y_pred_rf))
print( mean_absolute_error(y_test,y_pred_rf))
print(np.sqrt(mean_squared_error(y_test,y_pred_rf)))
```

Hyperparameter tuning

In [50]:

```
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
#define model and parameter
model=RandomForestRegressor()
n_estimators=[10,100,1000]
max_depth=range(1,31)
min_sample_leaf=np.linspace(0.1,1.0)
max_features=['auto','sqrt','log2']

#define grid search
grid=dict(n_estimators=n_estimators)
#cv=RepeatedStratifiedKFold(n_splits=5,n_repeats=3, random_state=101)
grid_search_forest=GridSearchCV(estimator=model,param_grid=grid,n_jobs=-1,
                               scoring='r2', error_score=0, verbose=2, cv=2)
grid_search_forest.fit(x_train_std,y_train)
# Summerize result
print(f'Best:{grid_search_forest.best_score_:.3f}using{grid_search_forest.best_params_}')
means=grid_search_forest.cv_results_['mean_test_score']
stds=grid_search_forest.cv_results_['std_test_score']
params=grid_search_forest.cv_results_['params']

for mean,stdev, param in zip(means,stds,params):
    print(f'{mean:.3f}({stdev:.3f})with{param}')
```

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Best:0.550using{'n_estimators': 1000}
0.517(0.011)with{'n_estimators': 10}
0.542(0.007)with{'n_estimators': 100}
0.550(0.005)with{'n_estimators': 1000}
```

In [51]:

```
grid_search_forest.best_params_
```

Out[51]:

```
{'n_estimators': 1000}
```

In [52]:

```
grid_search_forest.best_score_
```

Out[52]:

```
0.5498216664771529
```

In [53]:

```
y_pred_rf_grid= grid_search_forest.predict(x_test_std)
```

In [54]:
```
r2_score(y_test,y_pred_rf_grid)
```

Out[54]: 0.5506195416916224

In [ ]:
```

```

By: Ashwin Gorakhnath Dubey

In [ ]:
```

```