Assignment: Big Mart Sales Analysis(DA-3)

Roll No: 41145

Batch: R1

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Linear Regression

Linear regression is a regression model that estimates the relationship between one independent variable and one dependent variable using a straight line.

Example

You are a social researcher interested in the relationship between income and happiness. You survey 500 people whose incomes range from 15k to 75k and ask them to rank their happiness on a scale from 1 to 10. Your independent variable (income) and dependent variable (happiness) are both quantitative, so you can do a regression analysis to see if there is a linear relationship between them.

In [3]:

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [22]:

```
#Combine test and train into one file
train['source']='train'
test['source']='test'
data = pd.concat([train, test],ignore_index=True)

# print the details of the data
print(train.shape, test.shape, data.shape)
```

```
(8523, 13) (5681, 12) (14204, 13)
```

In [23]:

```
data.head()
```

Out[23]:

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

In [24]:

```
# Numerical data summary:
data.describe()
```

Out[24]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	11765.000000	14204.000000	14204.000000	14204.000000	8523.000000
mean	12.792854	0.065953	141.004977	1997.830681	2181.288914
std	4.652502	0.051459	62.086938	8.371664	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.710000	0.027036	94.012000	1987.000000	834.247400
50%	12.600000	0.054021	142.247000	1999.000000	1794.331000
75%	16.750000	0.094037	185.855600	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

Data Cleaning

In [25]:

```
# Check missing values & print the count of resp. null values
data.apply(lambda x: sum(x.isnull()))
```

Out[25]:

```
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
```

Filling missing values

So, we have missing values in Item_Weight and Outlet_Size column. We need to impute these missing values with appropriate ones. We usually replace missing values in numerical columns with mean and in categorical columns with mode.

In [26]:

```
data.Item_Outlet_Sales = data.Item_Outlet_Sales.fillna(data.Item_Outlet_Sales.mean())
```

In [27]:

```
data.Item_Weight = data.Item_Weight.fillna(data.Item_Weight.mean())
```

In [28]:

```
data['Outlet_Size'].value_counts()
```

Out[28]:

```
Medium    4655
Small     3980
High      1553
Name: Outlet_Size, dtype: int64
```

In [29]:

```
data.Outlet_Size = data.Outlet_Size.fillna('Medium')
```

In [30]:

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

Out[30]:

```
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
source                       0
dtype: int64
```

In [31]:

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

In [32]:

```
#Item type combine:
data['Item_Identifier'].value_counts()
data['Item_Type_Combined'] = data['Item_Identifier'].apply(lambda x: x[0:2])
data['Item_Type_Combined'] = data['Item_Type_Combined'].map({'FD':'Food',
                                                              'NC':'Non-Consumable',
                                                              'DR':'Drinks'})
data['Item_Type_Combined'].value_counts()
```

Out[32]:

```
Food              10201
Non-Consumable     2686
Drinks             1317
Name: Item_Type_Combined, dtype: int64
```

Numerical and One-Hot Coding of Categorical variables

In [33]:

```
#Import library:
# Label Encoder - This approach is very simple and it involves converting each value in a
column to a number.
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
#New variable for outlet
data['Outlet'] = le.fit_transform(data['Outlet_Identifier'])
var_mod = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Item_Type_Combined','
Outlet_Type','Outlet']
le = LabelEncoder()
for i in var_mod:
    data[i] = le.fit_transform(data[i])
```

In [34]:

```
#One Hot Coding:
# In this strategy, each category value is converted into a new column and assigned a 1 o
r 0 (notation for true/false) value to the column.
data = pd.get_dummies(data, columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_S
ize','Outlet_Type','Item_Type_Combined','Outlet'])
```

In [35]:

```
data.head()
```

Out[35]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Item_Outlet_
0	FDA15	9.30	0.016047	Dairy	249.8092	OUT049	1999	373
1	DRC01	5.92	0.019278	Soft Drinks	48.2692	OUT018	2009	44
2	FDN15	17.50	0.016760	Meat	141.6180	OUT049	1999	209
3	FDX07	19.20	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	73
4	NCD19	8.93	0.000000	Household	53.8614	OUT013	1987	99

In [36]:

```
data.dtypes
```

Out[36]:

```
Item_Identifier                object
Item_Weight                   float64
Item_Visibility               float64
Item_Type                      object
Item_MRP                      float64
Outlet_Identifier              object
Outlet_Establishment_Year       int64
Item_Outlet_Sales             float64
source                         object
Item_Fat_Content_0              uint8
Item_Fat_Content_1              uint8
Item_Fat_Content_2              uint8
Item_Fat_Content_3              uint8
Item_Fat_Content_4              uint8
Outlet_Location_Type_0          uint8
Outlet_Location_Type_1          uint8
Outlet_Location_Type_2          uint8
Outlet_Size_0                   uint8
Outlet_Size_1                   uint8
Outlet_Size_2                   uint8
Outlet_Type_0                   uint8
```

```
Outlet_Type_0                  uint8
Outlet_Type_1                  uint8
Outlet_Type_2                  uint8
Outlet_Type_3                  uint8
Item_Type_Combined_0           uint8
Item_Type_Combined_1           uint8
Item_Type_Combined_2           uint8
Outlet_0                       uint8
Outlet_1                       uint8
Outlet_2                       uint8
Outlet_3                       uint8
Outlet_4                       uint8
Outlet_5                       uint8
Outlet_6                       uint8
Outlet_7                       uint8
Outlet_8                       uint8
Outlet_9                       uint8
dtype: object
```

Exporting Data

In [37]:

```
import warnings
warnings.filterwarnings('ignore')
#Drop the columns which have been converted to different types:
data.drop(['Item_Type','Outlet_Establishment_Year'],axis=1,inplace=True)

#Divide into test and train:
train = data.loc[data['source']=="train"]
test = data.loc[data['source']=="test"]

#Drop unnecessary columns:
test.drop(['Item_Outlet_Sales','source'],axis=1,inplace=True)
train.drop(['source'],axis=1,inplace=True)

#Export files as modified versions:
train.to_csv("train_modified.csv",index=False)
test.to_csv("test_modified.csv",index=False)
```

Model Building

In [38]:

```
# Reading modified data
train2 = pd.read_csv("train_modified.csv")
test2 = pd.read_csv("test_modified.csv")
```

In [39]:

```
train2.head()
```

Out[39]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Item_Outlet_Sales	Item_Fat_Content_0	Item_Fat_Co
0	FDA15	9.30	0.016047	249.8092	OUT049	3735.1380	0	
1	DRC01	5.92	0.019278	48.2692	OUT018	443.4228	0	
2	FDN15	17.50	0.016760	141.6180	OUT049	2097.2700	0	
3	FDX07	19.20	0.000000	182.0950	OUT010	732.3800	0	
4	NCD19	8.93	0.000000	53.8614	OUT013	994.7052	0	

In [40]:

```
X_train = train2.drop(['Item_Outlet_Sales', 'Outlet_Identifier','Item_Identifier'], axis
=1)
y_train = train2.Item_Outlet_Sales
```

In [41]:

```
X_test = test2.drop(['Outlet_Identifier','Item_Identifier'], axis=1)
```

In [42]:

```
X_train.head()
```

Out[42]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Fat_Content_0	Item_Fat_Content_1	Item_Fat_Content_2	Item_Fat_Content_3
0	9.30	0.016047	249.8092	0	1	0	0
1	5.92	0.019278	48.2692	0	0	1	0
2	17.50	0.016760	141.6180	0	1	0	0
3	19.20	0.000000	182.0950	0	0	1	0
4	8.93	0.000000	53.8614	0	1	0	0

In [43]:

```
y_train.head()
```

Out[43]:

```
0    3735.1380
1     443.4228
2    2097.2700
3     732.3800
4     994.7052
Name: Item_Outlet_Sales, dtype: float64
```

Linear Regression Model:

In [56]:

```
# Fitting Multiple Linear Regression to the training set
from sklearn.linear_model import  LinearRegression
from sklearn.metrics import r2_score
from sklearn import metrics
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[56]:

```
LinearRegression()
```

In [57]:

```
# Predicting the test set results
y_pred = regressor.predict(X_test)
```

In [58]:

```
y_pred
```

Out[58]:

```
array([1848.53604783, 1472.81670435, 1875.65285894, ..., 1809.18796433,
       3565.6645235 , 1267.46171871])
```

In [80]:

```
regressor.score(X_train,y_train)
```

Out[80]:

```
0.6127122806844048
```

In [81]:

```
lr_accuracy = round(regressor.score(X_train,y_train) * 100,2)
lr_accuracy
```

Out[81]:

```
61.27
```

In [82]:

```
r2_score(y_train, regressor.predict(X_train))
```

Out[82]:

```
0.6127122806844046
```

In [83]:

```
print("RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(y_train, regressor.predict(X_tr
ain))))
```

```
RMSE : 1062
```

In [84]:

```
submission = pd.DataFrame({
'Item_Identifier':test2['Item_Identifier'],
'Outlet_Identifier':test2['Outlet_Identifier'],
'Item_Outlet_Sales': y_pred
},columns=['Item_Identifier','Outlet_Identifier','Item_Outlet_Sales'])
```

In [63]:

```
submission.to_csv('submission1.csv',index=False)
```

Decision Tree Model:

In [64]:

```
# Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(max_depth=15,min_samples_leaf=300)
regressor.fit(X_train, y_train)
```

Out[64]:

```
DecisionTreeRegressor(max_depth=15, min_samples_leaf=300)
```

In [65]:

```
# Predicting the test set results
y_pred = regressor.predict(X_test)
y_pred
```

Out[65]:

```
array([1673.98398729, 1349.51290433,  471.30684669, ..., 1892.06614452,
       3805.94860417, 1349.51290433])
```

In [66]:

```
tree_accuracy = round(regressor.score(X_train,y_train),2)
tree_accuracy
```

Out[66]:

```
0.59
```

In [67]:

```
r2_score(y_train, regressor.predict(X_train))
```

Out[67]:

```
0.5884050821570486
```

In [68]:

```
print("RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(y_train, regressor.predict(X_tr
ain))))
```

```
RMSE : 1095
```

In [69]:

```
submission = pd.DataFrame({
'Item_Identifier':test2['Item_Identifier'],
'Outlet_Identifier':test2['Outlet_Identifier'],
'Item_Outlet_Sales': y_pred
},columns=['Item_Identifier','Outlet_Identifier','Item_Outlet_Sales'])
```

In [70]:

```
submission.to_csv('submission2.csv',index=False)
```

Random Forest Model:

In [71]:

```
# Fitting Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100,max_depth=6, min_samples_leaf=50,n_jo
bs=4)
regressor.fit(X_train, y_train)
```

Out[71]:

```
RandomForestRegressor(max_depth=6, min_samples_leaf=50, n_jobs=4)
```

In [72]:

```
# Predicting the test set results
y_pred = regressor.predict(X_test)
y_pred
```

Out[72]:

```
array([1660.36142353, 1366.37445379,  593.00865621, ..., 1937.99949775,
       3630.89534885, 1297.418435  ])
```

In [73]:

```
rf_accuracy = round(regressor.score(X_train,y_train),2)
rf_accuracy
```

Out[73]:

```
0.61
```

In [74]:

```
r2_score(y_train, regressor.predict(X_train))
```

Out[74]:

```
0.6127122806844048
```

In [75]:

```
print("RMSE : %.4g" % np.sqrt(metrics.mean_squared_error(y_train, regressor.predict(X_tr
ain))))
```

```
RMSE : 1062
```

In [76]:

```
submission = pd.DataFrame({
'Item_Identifier':test2['Item_Identifier'],
'Outlet_Identifier':test2['Outlet_Identifier'],
'Item_Outlet_Sales': y_pred
},columns=['Item_Identifier','Outlet_Identifier','Item_Outlet_Sales'])
```

In [77]:

```
submission.to_csv('submission3.csv',index=False)
```