PROJECT REPORT

ABOUT BIGMART_SALE DATA

Programming language: Python (Jupyter Notebook).

Data-Preprocessing:

1.

a) Clean all null values

- Find and count all missing value:

```
In [3]: sales.isnull().sum()
```

```
Out[3]: 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
```

- Handle missing values in Item_Weight column:

Step 1: Calculate mean of Item_weight group by Item_Identifier:

```
mean_weight = sales.pivot_table(index = 'Item_Identifier', values = 'Item_Weight', aggfunc = 'mean
mean_weight.reset_index(inplace=True)
print(mean_weight)
```

```
     Item_Identifier  Item_Weight
0              DRA12       11.600
1              DRA24       19.350
2              DRA59        8.270
3              DRB01        7.390
4              DRB13        6.115
...              ...          ...
1550           NCZ30        6.590
1551           NCZ41       19.850
1552           NCZ42       10.500
1553           NCZ53        9.600
1554           NCZ54       14.650

[1555 rows x 2 columns]
```

Step 2: Fill null values in the Item_Weight corresponding to Item_Identifier and mean of Item_Weight have already calculated above.

```
index = sales[sales['Item_Weight'].isnull()].index.tolist()
for i in range(len(index)):
    for j in range(len(mean_weight['Item_Identifier'])):
        if mean_weight['Item_Identifier'][j] == sales['Item_Identifier'][index[i]]:
            sales['Item_Weight'][index[i]] = mean_weight['Item_Weight'][j]
```

However, there are still 4 null values because they are not compatible with any Item_Identifier. We will fill this 4 null values with the mean of all Item_Weight.

There are still 4 null values

In [10]:
```
other = sales[sales['Item_Weight'].isnull()].index.tolist()
print(other)
```

```
[927, 1922, 4187, 5022]
```

In [11]:
```
for i in range(len(other)):
    sales['Item_Weight'][other[i]] = sales['Item_Weight'].mean()
```

⇨ So now all null values in Item_Weight column have been filled in.

- Handle missing values in Outlet_Size column:
  Outlet_Size is categorical variable including High, Medium, Small.

  We will group Outlet_ Type and Outlet_Size.

```
pd.crosstab(sales['Outlet_Size'],sales['Outlet_Type'])
```

Outlet_Type Outlet_Size	Grocery Store	Supermarket Type1	Supermarket Type2	Supermarket Type3
High	0	932	0	0
Medium	0	930	928	935
Small	528	1860	0	0

  We will fill Nan values in Out_Size is Small if Outlet_Type is Grocery Store and Supermarket Type1. Medium if Outlet_Type is Supermarket Type2 and 3.

```
#chon small cho GS va ST1, medium cho ST2 va ST3
index_2 = sales[sales['Outlet_Size'].isnull()].index.tolist()

for j in range(len(index_2)):
    if sales['Outlet_Type'][index_2[j]] == 'Grocery Store':
        sales['Outlet_Size'][index_2[j]] = 'Small'
    elif sales['Outlet_Type'][index_2[j]] == 'Supermarket Type1':
        sales['Outlet_Size'][index_2[j]] = 'Small'
    elif sales['Outlet_Size'][index_2[j]] == 'Supermarket Type2':
        sales['Outlet_Size'][index_2[j]] = 'Medium'
    elif sales['Outlet_Size'][index_2[j]] == 'Supermarket Type3':
        sales['Outlet_Size'][index_2[j]] = 'Medium'
```

⇨ So all null values in Outlet_Size have been filled in.

⇨ All null values have been filled in now.

- **Clean noise data:**

- Looking at the Item_Identidier column, we will select only first two characters.

⇨ Now, we have:

```
FD    6125
NC    1599
DR     799
Name: Item_Identifier, dtype: int64
```

- Looking at the Item_Fat_Content column:

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

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

We will change 'LF' to 'Low Fat', 'reg' to 'Regular', 'low fat' to 'Low Fat'

⇨ We have:

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

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

- Looking at the Item_Visibility:

```
sales['Item_Visibility'].isin([0]).sum()
```

```
526
```

We have 526 [0] values, we will change value 0 to median of Item_Visibility.

```
# change sales['Item_Visibility']
sales['Item_Visibility'] = sales['Item_Visibility'].replace([0],[sales['Item_Visibility'].median()])
```

We can see that Year column have very large value but it haven't a lot of meaning, it will affect our prediction result later, so we will take current year (2021) minus year in data.

```
# year
for i in range(len(sales['Outlet_Establishment_Year'])):
    sales['Outlet_Establishment_Year'][i] = 2021 - sales['Outlet_Establishment_Year'][i]
```

b) All column except Item_Outlet_Sales are independent variables and Item_Outlet_Sales are dependent variable.

```
sales.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                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
```

'Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier', 'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type' are categorical variables.
Others are continuous variable.

c) Seafood in both Low Fat content and Regular content usually sold cheaper than other type of foods.

Fruits and Vegetables have high retailing price and sales in small types of supermarket.
Low Fat content have higher sale in small store.
Fruits and Vegetables have highest demand in small, medium and high store.
Snack food havs highest demand in small store and and highest sold in Tier3 city.

2. Working with Categorical variable:

We will see overall about out data:

```
sales.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                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
```

Now, we will be handling with each Dtype namely ‘object’.
Step 1: We need to see the distribution of each feature.

```
In [24]:
categories = []
for i in sales.columns:
    if sales.dtypes[i] == 'object':
        categories.append(i)
print(categories)
```

```
['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier', 'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type']
```

```
In [26]:
counts = []
for i in categories:
    counts.append(sales[i].value_counts())
print(counts)
```

```
[FD    6125
NC    1599
DR     799
Name: Item_Identifier, dtype: int64, Low Fat    5517
Regular    3006
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
OUT046    930
OUT035    930
OUT049    930
OUT045    929
OUT018    928
OUT017    926
OUT010    555
OUT019    528
Name: Outlet_Identifier, dtype: int64, Small     4798
Medium    2793
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]
```

- **Get_dummies:**
  We decide use ‘get_dummies’ for Item_Identifier, Item_Fat_Content, Outlet_Size and Outlet_Location_Type because we can see each column that we select how have very few categories from 3 to 4 so it wont create too many new columns.

```
sales = pd.get_dummies(sales)
```

- **Encoding categorical variables:**
  With Item_Type, Outlet_Identifier and Outlet_Type we will encode it.
  we will call package sklearn.preprocessing and call library LabelEncoder.

```
# ma hoa gia tri
from sklearn.preprocessing import LabelEncoder
lb =  LabelEncoder()
sales['Item_Type'] = lb.fit_transform(sales['Item_Type'])
sales['Outlet_Identifier'] = lb.fit_transform(sales['Outlet_Identifier'])
sales['Outlet_Type'] = lb.fit_transform(sales['Outlet_Type'])
```

Dairy	4
Soft Drinks	14
Meat	10
Fruits and Vegetables	6
Household	9
Baking Goods	0
Snack Foods	13
Frozen Foods	5
Breakfast	2
Health and Hygiene	8
Hard Drinks	7
Canned	3
Breads	1
Starchy Foods	15
Others	11
Seafood	12

OUT049	9
OUT018	3
OUT010	0
OUT013	1
OUT027	5
OUT045	7
OUT017	2
OUT046	8
OUT035	6
OUT019	4

Supermarket Type1	1
Supermarket Type2	2
Grocery Store	0
Supermarket Type3	3

So now, we have data shape (8523,19).
All categorical variables now become numeric variable.

c) We can see that ‘Item_Outlet_Sales’ column is continuous variable.

We decided to change into categorical variable (3 categories):

Group A : Item_Outlet_Sales < 1000.

Group B : 1000 <= Item_Outlet_Sales < 10000.

Group C : Item_Outlet_Sales >= 10000.

In [32]:
```python
A = []
B = []
C = []
for i in range(len(sales['Item_Outlet_Sales'])):
    if sales['Item_Outlet_Sales'][i] < 1000:
        A.append(i)
    elif sales['Item_Outlet_Sales'][i] < 10000:
        B.append(i)
    else:
        C.append(i)
```

In [33]:
```python
for i in A:
    sales['Item_Outlet_Sales'][i] = str(sales['Item_Outlet_Sales'][i]).replace(str(sales['Item_Outlet_Sales'][i]),"A")
```

In [34]:
```python
for i in B:
    sales['Item_Outlet_Sales'][i] = str(sales['Item_Outlet_Sales'][i]).replace(str(sales['Item_Outlet_Sales'][i]),"B")
```

In [35]:
```python
for i in C:
    sales['Item_Outlet_Sales'][i] = str(sales['Item_Outlet_Sales'][i]).replace(str(sales['Item_Outlet_Sales'][i]),"C")
```

In [36]:
```python
print(sales['Item_Outlet_Sales'].value_counts())
```

```
B    6011
A    2504
C       8
Name: Item_Outlet_Sales, dtype: int64
```

After that We will be encoding it: A is 0, B is 1, C is 2.

```python
sales['Item_Outlet_Sales'] = lb.fit_transform(sales['Item_Outlet_Sales'])
print(sales['Item_Outlet_Sales'].value_counts())
```

```
1    6011
0    2504
2       8
Name: Item_Outlet_Sales, dtype: int64
```

d) + e) :

- **Split data into training and testing data:**

- 70% for training data and 30% for testing data.

```python
# chia independent(X) and dependent(y) variables
# y is continuous data
X = sales.drop(['Item_Outlet_Sales'],axis = 1)
y = sales['Item_Outlet_Sales']
```

```python
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size = 0.7, random_state = 42)
```

3.

- **Looking at the dependent variable:**

  Item_Outlet_Sales column is dependent variable and it is what we will predict.

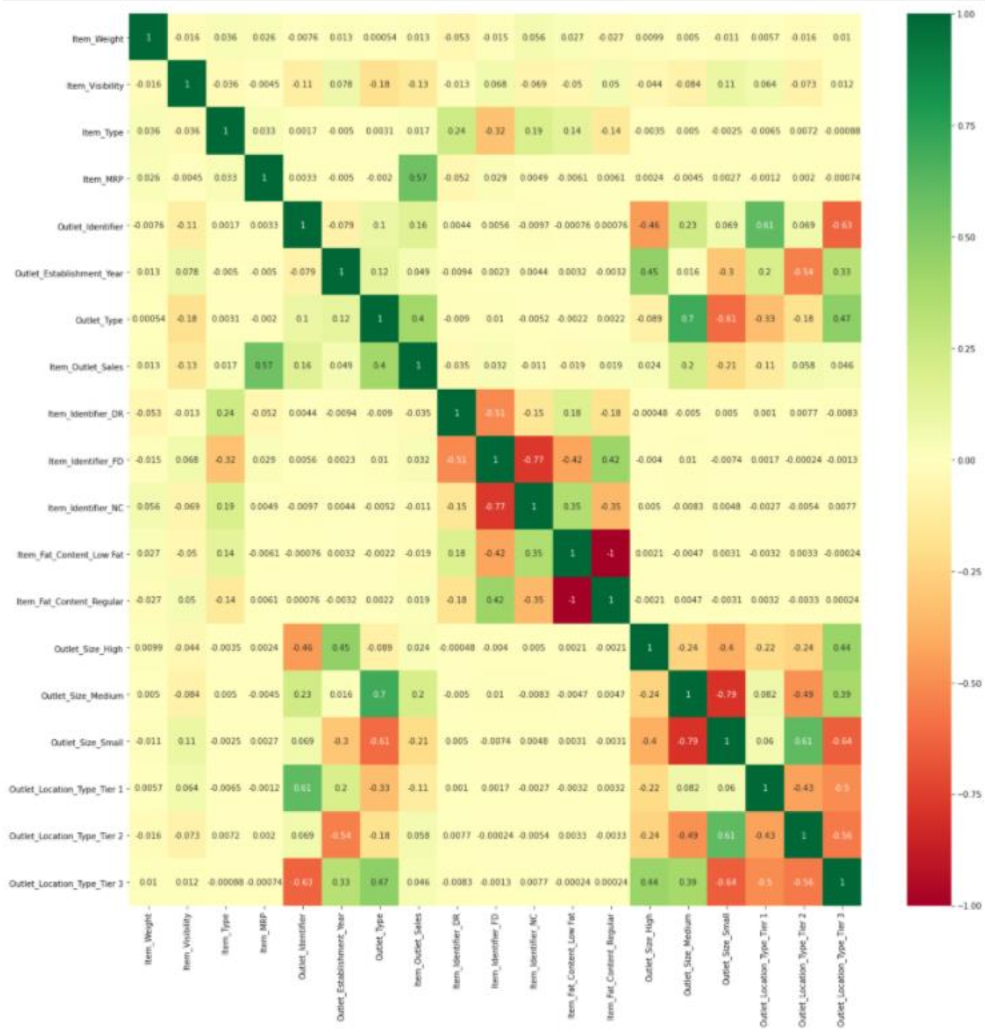
  We can see that it is continuous variable but after preprocessing data, Item_Outlet_Sales column is become categorical variable with 3 categories.

  Advantages: It is easier to predict and gets high score.

  Disadvantages: We can't predict exactly price of Item_Outlet_Sales, we can only predict at this price, which group it belongs to

- **Before go to build model we draw correlation table of all independent variables**

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
corrmat = sales.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(sales[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



As corr() between Item_Outlet_sales and Item MRP ~ 0.57 means when Item MRP increases leading to increase in Item_Outlet_sales

- **Feature selection:**

```
In [37]:
y=y.astype('int')
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
bestfeatures = SelectKBest(score_func=chi2)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['name','Score']  #naming the dataframe columns
print(featureScores.nlargest(18,'Score'))
```

```
                           name          Score
3                      Item_MRP  155069.429794
5     Outlet_Establishment_Year   10829.620835
2                     Item_Type    8560.004638
4             Outlet_Identifier    6021.474301
0                   Item_Weight    5610.966685
7            Item_Identifier_DR    3099.961747
12             Outlet_Size_High    2785.709977
9            Item_Identifier_NC    2631.188116
6                   Outlet_Type    2561.045867
13           Outlet_Size_Medium    2366.494962
15  Outlet_Location_Type_Tier 1    2288.506128
16  Outlet_Location_Type_Tier 2    2265.608036
11     Item_Fat_Content_Regular    2190.530020
17  Outlet_Location_Type_Tier 3    1963.546577
14            Outlet_Size_Small    1520.963954
10     Item_Fat_Content_Low Fat    1193.535117
8            Item_Identifier_FD     939.850544
1               Item_Visibility     119.562013
```

As we can see all feature have very high score so we wil use all independent variables for building model

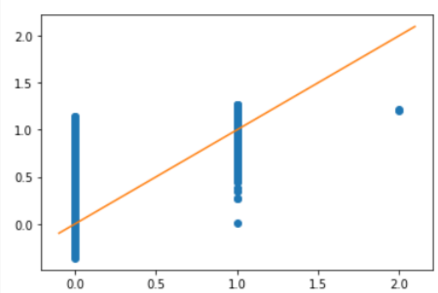
4. **Building model:**

- **Linear Regression:**

liner model

```
In [44]:
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train,y_train)
y_predic = lr.predict(X_test)
print("train_score: {}".format(lr.score(X_train,y_train)))
print("test_score: {}".format(lr.score(X_test,y_test)))
plt.plot(y_test,y_predic,'o')
axes = plt.gca()
plt.plot(axes.get_xlim(),axes.get_xlim(), '-')
plt.show()
```

```
train_score: 0.4672934323357183
test_score: 0.4730000852332201
```



- **Logistic Regression:**

Logistic regression

```
In [45]:
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression(C=10^9).fit(X_train,y_train)
print("train_score: {}".format(lg.score(X_train,y_train)))
print("test_score: {}".format(lg.score(X_test,y_test)))
```

```
train_score: 0.8826684545759302
test_score: 0.8826750097770825
```

- **KNN:**

KNN

In [46]:
```
from sklearn.neighbors import KNeighborsClassifier
kn =KNeighborsClassifier().fit(X_train,y_train)
print("train_score: {}".format(kn.score(X_train,y_train)))
print("test_score: {}".format(kn.score(X_test,y_test)))
```

```
train_score: 0.8853503184713376
test_score: 0.8513883457176379
```

- **DecisionTreeClassifier:**

decision tree

In [51]:
```
from sklearn.tree import DecisionTreeClassifier

dc = DecisionTreeClassifier(max_depth=4).fit(X_train,y_train)
print("train_score: {}".format(dc.score(X_train,y_train)))
print("test_score: {}".format(dc.score(X_test,y_test)))
#seem that i = 4 is the best
```

```
train_score: 0.8923902111967817
test_score: 0.8920610089949159
```

- **RandomForestClassifier:**

RandomForest

In [52]:
```
from sklearn.ensemble import RandomForestClassifier
rc = RandomForestClassifier(max_depth=10,n_estimators = 100).fit(X_train,y_train)
print("train_score: {}".format(rc.score(X_train,y_train)))
print("test_score: {}".format(rc.score(X_test,y_test)))
```

```
train_score: 0.914683204827355
test_score: 0.8932342588971451
```

- **Model evaluation for RandomForestClassifier:**

model evaluation for randomforest

In [53]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth': [1,2,3,4,5,6,7,8,9,10],
'n_estimators': [10,50,100,150]}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
```

```
Test set score: 0.89
```

In [54]:
```
print("Best parameters: {}".format(grid_search.best_params_))
print("Best test score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'max_depth': 10, 'n_estimators': 100}
Best test score: 0.89
```

In [55]:
```
rr = RandomForestClassifier(max_depth=9,n_estimators = 50).fit(X_train,y_train)
print("train_score: {}".format(rr.score(X_train,y_train)))
print("test_score: {}".format(rr.score(X_test,y_test)))
```

```
train_score: 0.9066376131411331
test_score: 0.8936253421978881
```

- **Support vector machine Classifier:**

support vector machine

In [47]:
```
from sklearn.svm import SVC
svm = SVC(kernel= 'linear', random_state=1, C=0.1)
svm.fit(X_train, y_train)
print("train_score: {}".format(svm.score(X_train,y_train)))
print("test_score: {}".format(svm.score(X_test,y_test)))
```

```
train_score: 0.8835065370432451
test_score: 0.8904966757919437
```

- **Model evaluation for Support vector machine Classifier:**

model evaluation for svm

In [48]:
```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

In [49]:
```
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
```

```
Test set score: 0.89
Best parameters: {'C': 100, 'gamma': 0.001}
```

In [50]:
```
svm = SVC(C=100, gamma = 0.001)
svm.fit(X_train, y_train)
print("train_score: {}".format(svm.score(X_train,y_train)))
print("test_score: {}".format(svm.score(X_test,y_test)))
```

```
train_score: 0.8918873617163929
test_score: 0.8920610089949159
```

- **Inconclusion** : DecisionTreeClassifier, RandomForestClassifier, SupportVectorMachineClassifier have highest prediction score.

--END--