# Problem Statement

The Data Scientists at BigMart have collected sales for one thousands five hundred and fifty-nine(1559) products across ten stores in different cities. Also, certain attributes of each product and store have been defined.

## Aim

> The aim 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 stires which palyt a key role in increasing sales.

## Understand the problem

Q. What could affect the target variabke 'Sasles'?

1. The time of the week as they are usually busier.
2. Higher sales according to the time of the day that is, morning and late evenings.
3. Higher sales during the end of the year
4. Store size and location
5. Items with more shelf space sell more.

## Import required libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
```

In [2]:

```
train = pd.read_csv('bigmart_train.csv')
```

In [3]:

```
1 train.head(10)
```

Out[3]:

| | Item_Identifier | Item_Weight | Item_Fat_Content | Item_Visibility | Item_Type | Item_MRP | Outlet_ |
|---|---|---|---|---|---|---|---|
| **0** | FDA15 | 9.300 | Low Fat | 0.016047 | Dairy | 249.8092 | |
| **1** | DRC01 | 5.920 | Regular | 0.019278 | Soft Drinks | 48.2692 | |
| **2** | FDN15 | 17.500 | Low Fat | 0.016760 | Meat | 141.6180 | |
| **3** | FDX07 | 19.200 | Regular | 0.000000 | Fruits and Vegetables | 182.0950 | |
| **4** | NCD19 | 8.930 | Low Fat | 0.000000 | Household | 53.8614 | |
| **5** | FDP36 | 10.395 | Regular | 0.000000 | Baking Goods | 51.4008 | |
| **6** | FDO10 | 13.650 | Regular | 0.012741 | Snack Foods | 57.6588 | |
| **7** | FDP10 | NaN | Low Fat | 0.127470 | Snack Foods | 107.7622 | |
| **8** | FDH17 | 16.200 | Regular | 0.016687 | Frozen Foods | 96.9726 | |
| **9** | FDU28 | 19.200 | Regular | 0.094450 | Frozen Foods | 187.8214 | |

In [4]:

```
1 train.shape
```

Out[4]:

```
(8523, 12)
```

In [5]:

```
1 train.isnull().sum()
```

Out[5]:

```
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 [6]:

```
1 train['Item_Fat_Content'].unique()
```

Out[6]:

```
array(['Low Fat', 'Regular', 'low fat', 'LF', 'reg'], dtype=object)
```

In [7]:

```
1 train['Outlet_Establishment_Year'].unique()
```

Out[7]:

```
array([1999, 2009, 1998, 1987, 1985, 2002, 2007, 1997, 2004], dtype=in
t64)
```

In [8]:

```
1 train['Outlet_Age'] = 2018-train['Outlet_Establishment_Year']
```

In [9]:

```
1 train.head()
```

Out[9]:

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

In [10]:

```
1 train['Outlet_Size'].unique()
```

Out[10]:

```
array(['Medium', nan, 'High', 'Small'], dtype=object)
```

In [11]:

```
1 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]:

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

Out[12]:

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

In [13]:

```
train['Outlet_Size'].mode()[0]
```

Out[13]:

```
'Medium'
```

In [14]:

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

In [15]:

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

In [16]:

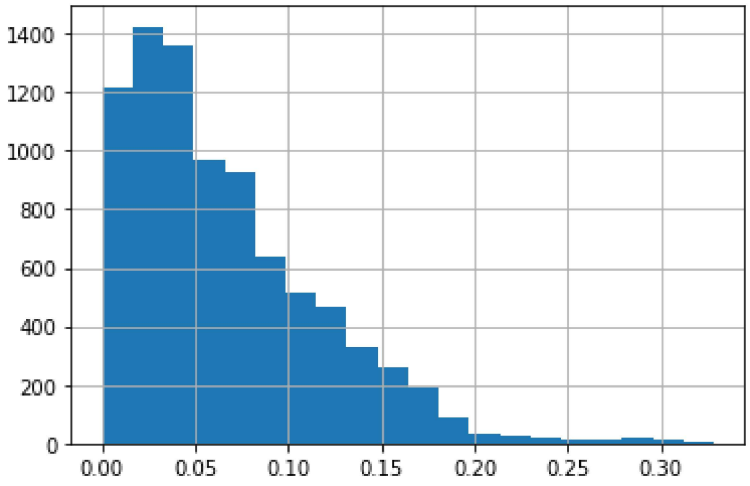
```
train.isnull().sum()
```

Out[16]:

```
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
Outlet_Age                   0
dtype: int64
```

In [17]:

```
train['Item_Visibility'].hist(bins = 20);
```




In [18]:

```
Q1 = train['Item_Visibility'].quantile(0.25)
Q3 = train['Item_Visibility'].quantile(0.75)

IQR = Q3- Q1
filt_train = train.query('(@Q1 - 1.5*@IQR) <= Item_Visibility <= (@Q3 + 1.5 * @
```

In [19]:

```
1 filt_train
```

Out[19]:

| | Item_Identifier | Item_Weight | Item_Fat_Content | Item_Visibility | Item_Type | Item_MRP | Outlet_Identifier |
|---|---|---|---|---|---|---|---|
| **0** | FDA15 | 9.300 | Low Fat | 0.016047 | Dairy | 249.8092 | OUT049 |
| **1** | DRC01 | 5.920 | Regular | 0.019278 | Soft Drinks | 48.2692 | OUT018 |
| **2** | FDN15 | 17.500 | Low Fat | 0.016760 | Meat | 141.6180 | OUT049 |
| **3** | FDX07 | 19.200 | Regular | 0.000000 | Fruits and Vegetables | 182.0950 | OUT010 |
| **4** | NCD19 | 8.930 | Low Fat | 0.000000 | Household | 53.8614 | OUT013 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **8518** | FDF22 | 6.865 | Low Fat | 0.056783 | Snack | 214.5218 | OUT013 |

In [20]:

```
1 filt_train.shape, train.shape
```

Out[20]:

```
((8379, 13), (8523, 13))
```

In [21]:

```
1 train = filt_train
2 train.shape
```

Out[21]:

```
(8379, 13)
```

In [22]:

```
1 train['Item_Visibility_bins'] = pd.cut(train['Item_Visibility'], [0.000, 0.065,
```

In [23]:

```
1 train['Item_Visibility_bins'].value_counts()
```

Out[23]:

```
Low Viz     4403
Viz         2557
High Viz     893
Name: Item_Visibility_bins, dtype: int64
```

In [24]:

```
1 train['Item_Visibility_bins'] = train['Item_Visibility_bins'].replace(np.NaN, '
```

In [25]:

```
train['Item_Fat_Content'] = train['Item_Fat_Content'].replace(['Low fat', 'LF']
```

In [26]:

```
train['Item_Fat_Content'] = train['Item_Fat_Content'].replace('reg', 'Regular')
```

In [27]:

```
train.head()
```

Out[27]:

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

In [28]:

```
le = LabelEncoder()
```

In [29]:

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

Out[29]:

```
array(['Low Fat', 'Regular', 'low fat'], dtype=object)
```

In [30]:

```
train['Item_Fat_Content'] = le.fit_transform(train['Item_Fat_Content'])
```

In [31]:

```
train['Item_Visibility_bins'] = le.fit_transform(train['Item_Visibility_bins'])
```

In [32]:

```
train['Outlet_Size'] = le.fit_transform(train['Outlet_Size'])
```

In [33]:

```
train['Outlet_Location_Type'] = le.fit_transform(train['Outlet_Location_Type'])
```

In [34]:

```
dummy = pd.get_dummies(train['Outlet_Type'])
```

In [35]:

```
dummy.head()
```

Out[35]:

| | Grocery Store | Supermarket Type1 | Supermarket Type2 | Supermarket Type3 |
|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 0 |
| **1** | 0 | 0 | 1 | 0 |
| **2** | 0 | 1 | 0 | 0 |
| **3** | 1 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0 | 0 |

In [36]:

```
train = pd.concat([train, dummy], axis = 1)
```

In [37]:

```
train.dtypes
```

Out[37]:

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content               int32
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                    int32
Outlet_Location_Type           int32
Outlet_Type                   object
Item_Outlet_Sales            float64
Outlet_Age                     int64
Item_Visibility_bins           int32
Grocery Store                  uint8
Supermarket Type1              uint8
Supermarket Type2              uint8
Supermarket Type3              uint8
dtype: object
```

In [38]:

```
train = train.drop(['Item_Identifier', 'Item_Type','Outlet_Identifier','Outlet_
```

In [39]:

```
1 train.head()
```

Out[39]:

| | Item_Weight | Item_Fat_Content | Item_Visibility | Item_MRP | Outlet_Size | Outlet_Location_Typ |
|---|---|---|---|---|---|---|
| **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 | |

In [40]:

```
1 train.columns
```

Out[40]:

```
Index(['Item_Weight', 'Item_Fat_Content', 'Item_Visibility', 'Item_MR
P',
       'Outlet_Size', 'Outlet_Location_Type', 'Item_Outlet_Sales',
       'Outlet_Age', 'Item_Visibility_bins', 'Grocery Store',
       'Supermarket Type1', 'Supermarket Type2', 'Supermarket Type3'],
      dtype='object')
```

In [41]:

```
1 train.isnull().sum()
```

Out[41]:

```
Item_Weight             0
Item_Fat_Content        0
Item_Visibility         0
Item_MRP                0
Outlet_Size             0
Outlet_Location_Type    0
Item_Outlet_Sales       0
Outlet_Age              0
Item_Visibility_bins    0
Grocery Store           0
Supermarket Type1       0
Supermarket Type2       0
Supermarket Type3       0
dtype: int64
```

In [42]:

```
1 X = train.drop('Item_Outlet_Sales', axis = 1)
2 y = train.Item_Outlet_Sales
```

In [43]:

```
test = pd.read_csv('bigmart_test.csv')
test['Outlet_size'] = test['Outlet_Size'].fillna('Medium')
```

In [44]:

```
test['Item_Visibility_bins'] = pd.cut(test['Item_Visibility'],[0.000, 0.065, 0.
```

In [45]:

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

In [46]:

```
test['Item_Visibility'] = test['Item_Visibility_bins'].fillna('Low Viz')
test['Item_Visibility_bins'] = le.fit_transform(test['Outlet_Location_Type'])
```

In [47]:

```
dummy = pd.get_dummies(test['Outlet_Type'])
test = pd.concat([test, dummy])
```

In [48]:

```
X_test = test.drop(['Item_Identifier', 'Item_Type','Outlet_Type', 'Outlet_Estab
```

In [49]:

```
X.columns, X_test.columns
```

Out[49]:

```
(Index(['Item_Weight', 'Item_Fat_Content', 'Item_Visibility', 'Item_MR
P',
        'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Age',
        'Item_Visibility_bins', 'Grocery Store', 'Supermarket Type1',
        'Supermarket Type2', 'Supermarket Type3'],
       dtype='object'),
 Index(['Item_Weight', 'Item_Fat_Content', 'Item_Visibility', 'Item_MR
P',
        'Outlet_Identifier', 'Outlet_Size', 'Outlet_Location_Type',
        'Outlet_size', 'Item_Visibility_bins', 'Item_weight', 'Grocery
Store',
        'Supermarket Type1', 'Supermarket Type2', 'Supermarket Type
3'],
       dtype='object'))
```

In [50]:

```
from sklearn import model_selection
xtrain,xtest,ytrain,ytest= model_selection.train_test_split(X, y, test_size = 0
```

In [51]:

```
lin = LinearRegression()
```

In [52]:

```
lin.fit(xtrain, ytrain)
print(lin.coef_)
lin.intercept_
```

```
[    3.35662016    25.60838646  -103.62415364    15.96855715
    16.78809725     7.94373052    -2.29584866    23.21038545
 -1754.50307086   219.60428213  -122.61674151  1657.51553023]
```

Out[52]:

```
-209.60516380835816
```

In [53]:

```
predictions = lin.predict(xtest)
print(sqrt(mean_squared_error(ytest, predictions)))
```

```
1118.48645487984
```

In [54]:

```
from sklearn.linear_model import Ridge
```

In [55]:

```
ridgereg = Ridge(alpha = 0.001, normalize = True)
ridgereg.fit(xtrain, ytrain)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_base.
py:145: FutureWarning: 'normalize' was deprecated in version 1.0 and w
ill be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Ridge())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.step
s}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * n_samples.
  FutureWarning,
```

Out[55]:

```
Ridge(alpha=0.001, normalize=True)
```

In [56]:

```
print(sqrt(mean_squared_error(ytrain, ridgereg.predict(xtrain))))
print(sqrt(mean_squared_error(ytest, ridgereg.predict(xtest))))
print('R2 Value/Coefficient of Detemination : {}'.format(ridgereg.score(xtest,y
```

```
1139.452862059927
1118.4290612607788
R2 Value/Coefficient of Detemination : 0.5486035026497414
```

In [57]:

```
from sklearn.linear_model import Lasso
```

In [58]:

```
lasso =Lasso(alpha = 0.001, normalize = True)
```

In [59]:

```
lasso.fit(xtrain, ytrain)
print(sqrt(mean_squared_error(ytrain, lasso.predict(xtrain))))
print(sqrt(mean_squared_error(ytest, lasso.predict(xtest))))
print('R2 Value.Coefficient of Determination : {}'.format(lasso.score(xtest, yt
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_base.
py:145: FutureWarning: 'normalize' was deprecated in version 1.0 and w
ill be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Lasso())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.step
s}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
  FutureWarning,

1139.4522016188898
1118.4800243829711
R2 Value.Coefficient of Determination : 0.5485623644143902
```

In [60]:

```
from sklearn.linear_model import ElasticNet
```

In [61]:

```
Elas = ElasticNet(alpha = 0.001, normalize =True)
Elas.fit(xtrain, ytrain)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_base.
py:145: FutureWarning: 'normalize' was deprecated in version 1.0 and w
ill be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a
preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), ElasticNet())

If you wish to pass a sample_weight parameter, you need to pass it as
a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.step
s}
model.fit(X, y, **kwargs)

Set parameter alpha to original_alpha * np.sqrt(n_samples) if l1_ratio
is 1, and to original_alpha * n_samples if l1_ratio is 0. For other va
lues of l1_ratio, no analytic formula is available.
  FutureWarning,
```

Out[61]:

```
ElasticNet(alpha=0.001, normalize=True)
```

In [62]:

```
print(sqrt(mean_squared_error(ytrain, Elas.predict(xtrain))))
print(sqrt(mean_squared_error(ytest, Elas.predict(xtest))))
print('R2 Value.Coefficient of Determination : {}'.format(Elas.score(xtest, yte
```

```
1478.2335442319848
1429.1345660993047
R2 Value.Coefficient of Determination : 0.26296596158006647
```

In [ ]:

```

```