21CSE322T - MULTIVARIATE TECHNIQUES FOR DATA ANALYTICS

SEMESTER-V

KHADEEJA NADA

RA2112704010027

Mtech Integrated CSE with Data Science

Academic Year:2021-2026

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section3 of UGC Act,1956)

SRM NAGAR KATTANKULATHUR – 603203

KANCHEEPURAM DISTRICT

`

LIST OF EXERCISES

Ex No : 1
DATE: 2-8-2023

UNIVARIATE, BIVARIATE AND MULTIVARIATE DATA ANALYSIS

AIM: To perform univariate, bivariate and multivariate data analysis using the given dataset.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the data frame. (Dataset used :BigMart Sales Prediction)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Perform the univariate, bivariate and multivariate analysis and display the results.
5. Analyze the displayed output.

DATASET DESCRIPTION:

1. City type: Stores located in urban or Tier 1 cities should have higher sales because of the higher income levels of people there.
2. Population Density: Stores located in densely populated areas should have higher sales because of more demand.
3. Store Capacity: Stores which are very big in size should have higher sales as they act like one-stop-shops and people would prefer getting everything from one place
4. Competitors: Stores having similar establishments nearby should have less sales because of more competition.
5. Marketing: Stores which have a good marketing division should have higher sales as it will be able to attract customers through the right offers and advertising.
6. Location: Stores located within popular marketplaces should have higher sales because of better access to customers.
7. Customer Behavior: Stores keeping the right set of products to meet the local needs of customers will have higher sales.

8. Ambiance: Stores which are well-maintained and managed by polite and humble people are expected to have higher footfall and thus higher sales.

PROGRAM WITH OUTPUT:

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

```
from google.colab import files
uploaded=files.upload()
```

Choose Files Test.csv

- **Test.csv**(text/csv) - 527709 bytes, last modified: 8/2/2023 - 100% done

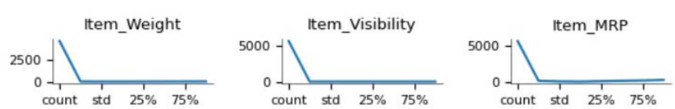
```
Saving Test.csv to Test.csv
```

```
df=pd.read_csv('Test.csv')
df.head()
```

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

	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

Values

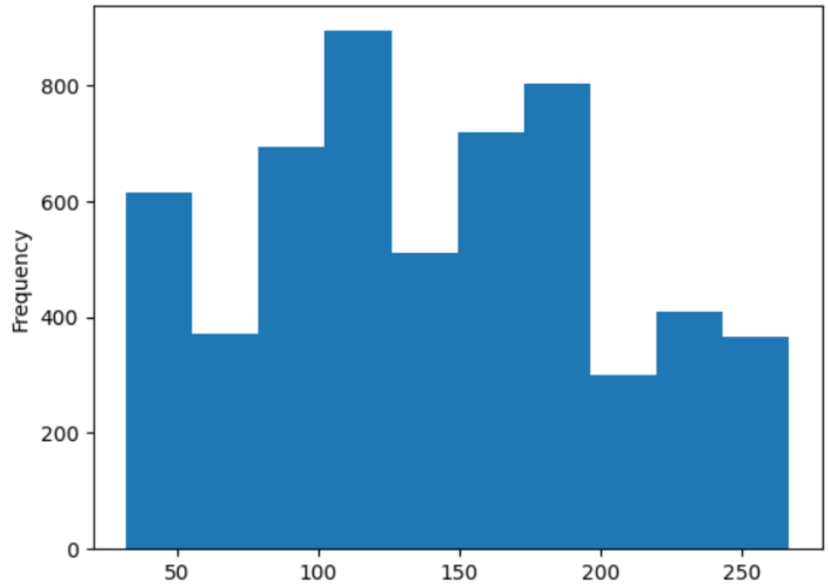



```
df['Item_MRP'].describe()
```

```
count    5681.000000
mean      141.023273
std        61.809091
min        31.990000
25%        94.412000
50%       141.415400
75%       186.026600
max       266.588400
Name: Item_MRP, dtype: float64
```

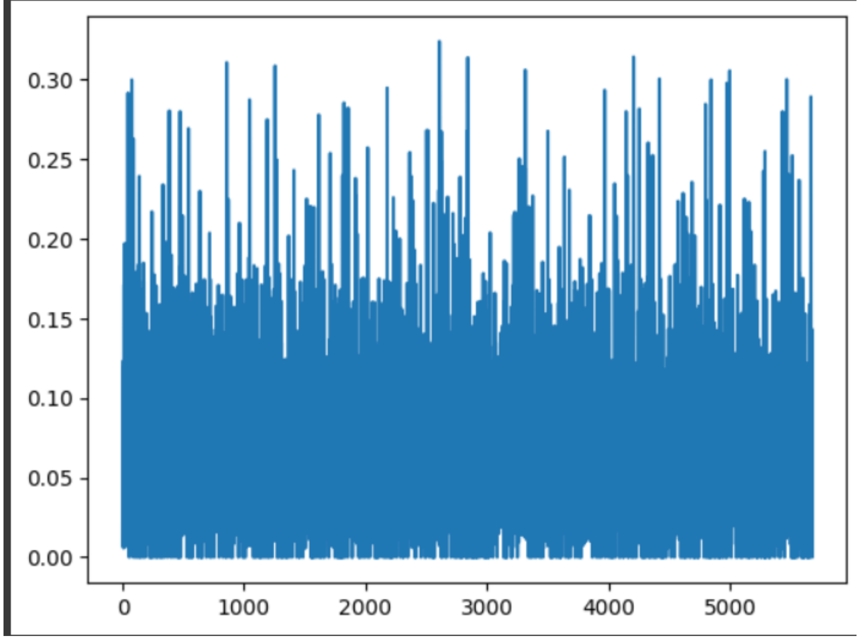
```
df['Item_MRP'].plot(kind="hist")
```

```
<Axes: ylabel='Frequency'>
```



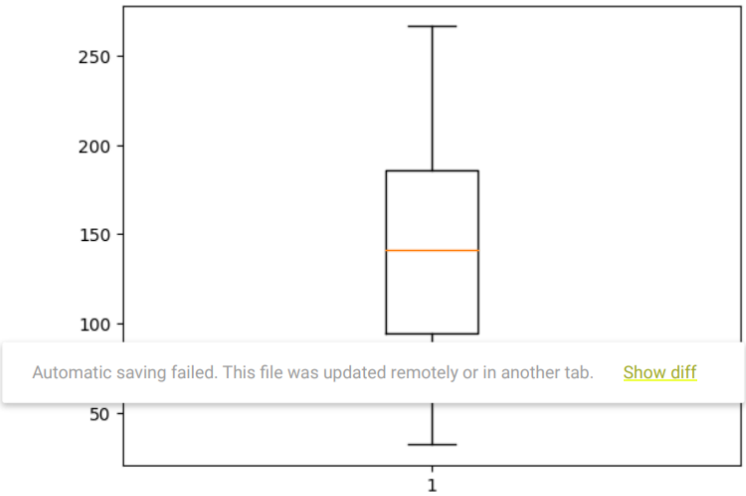

```
df['Item_Visibility'].plot(kind="line")
```

```
df['Item_Visibility'].plot(kind="line")
<Axes: >
```

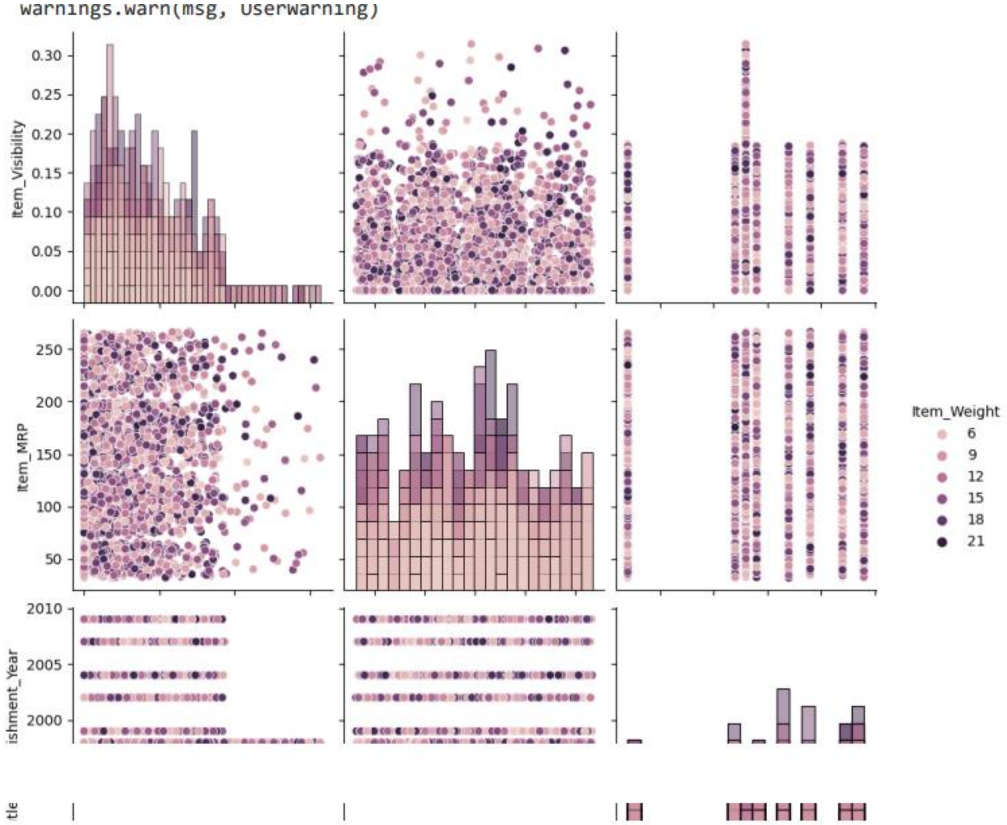


```
plt.boxplot(df["Item_MRP"])
```

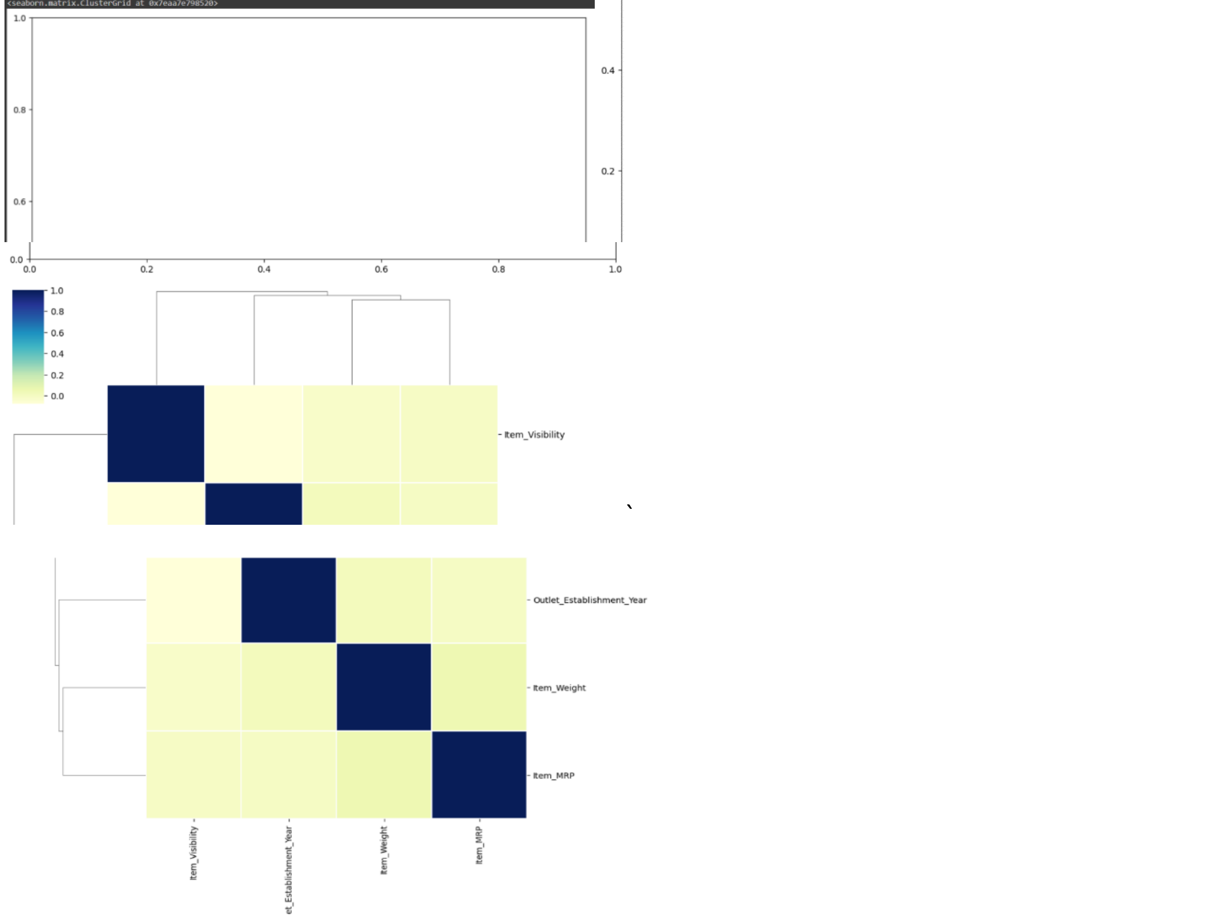
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7eaaa8e29600>,
  <matplotlib.lines.Line2D at 0x7eaaa8e298a0>],
 'caps': [<matplotlib.lines.Line2D at 0x7eaaa8e29b40>,
  <matplotlib.lines.Line2D at 0x7eaaa8e29de0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7eaaa8e29360>],
 'medians': [<matplotlib.lines.Line2D at 0x7eaaa8e2a080>],
 'fliers': [<matplotlib.lines.Line2D at 0x7eaaa8e2a320>],
 'means': []}
```




```
warnings.warn(msg, UserWarning)
```



```
cormat = df.corr(method='spearman')
f, ax = plt.subplots(figsize=(12, 10))
cg = sns.clustermap(cormat, cmap="YlGnBu", linewidths=0.1);
plt.setp(cg.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
cg

<ipython-input-27-38f12a9fa6d9>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future
  cormat = df.corr(method='spearman')
<seaborn.matrix.ClusterGrid at 0x7eaa7e798520>
```



CONCLUSION:

The univariate, bivariate and multivariate data analysis has been done using the given dataset and the results have been analyzed using the above visualizations.

Ex No : 2
DATE: 2-8-2023

DESCRIPTIVE DATA ANALYSIS

AIM: To perform Descriptive Statistics Analysis using the given dataset.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the data frame. (Dataset used :BigMart Sales Prediction)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Perform the Descriptive Statistical analysis
5. Analyze the displayed output.

PROGRAM WITH OUTPUT

```
df.describe(include='all')
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outl
count	5681	4705.000000	5681	5681.000000	5681	5681.000000	5681	
unique	1543	NaN	5	NaN	16	NaN	10	
top	DRF48	NaN	Low Fat	NaN	Snack Foods	NaN	OUT027	
freq	8	NaN	3396	NaN	789	NaN	624	
mean	NaN	12.695633	NaN	0.065684	NaN	141.023273	NaN	
std	NaN	4.664849	NaN	0.051252	NaN	61.809091	NaN	
min	NaN	4.555000	NaN	0.000000	NaN	31.990000	NaN	
25%	NaN	8.645000	NaN	0.027047	NaN	94.412000	NaN	
50%	NaN	12.500000	NaN	0.054154	NaN	141.415400	NaN	
75%	NaN	16.700000	NaN	0.093463	NaN	186.026600	NaN	

```
df.dtypes
```

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content              object
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                   object
Outlet_Location_Type          object
Outlet_Type                   object
dtype: object
```

```
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
```

```
df['Item_Weight'].mean()
```

```
12.695633368756642
```

```
df['Item_Weight'].mode()
```

```
0    10.5
Name: Item_Weight, dtype: float64
```

```
df['Item_Weight'].mode()[0]
```

```
10.5
```

```
df.describe(include=np.number)
```

```
df.describe(include=np.number)
```

	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

```
df.skew()
```

```
<ipython-input-45-9e0b1e29546f>:1: FutureWarning: The default value of numeric_only in [
  df.skew()
Item_Weight                  0.129975
Item_Visibility              1.238312
Item_MRP                     0.136182
```

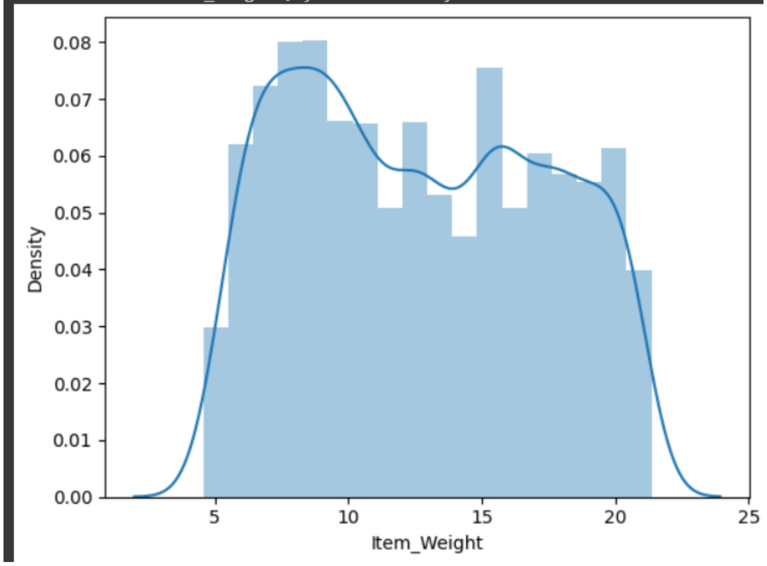


```
Outlet_Establishment_Year   -0.396306
dtype: float64
```

```
df.kurt()
```

```
<ipython-input-46-8bd0d54cd88d>:1: FutureWarning: The default value of numeric_only in [
  df.kurt()
Item_Weight                 -1.226412
Item_Visibility              2.040199
Item_MRP                    -0.900203
Outlet_Establishment_Year   -1.206132
dtype: float64
```

```
<Axes: xlabel='Item_Weight', ylabel='Density'>
```




```
sns.distplot(df['Item_Weight'])
```

```
df.corr()
```

```
<ipython-input-48-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
  df.corr()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
Item_Weight	1.000000	-0.019089	0.049809	0.018918
Item_Visibility	-0.019089	1.000000	-0.014013	-0.097040
Item_MRP	0.049809	-0.014013	1.000000	-0.007233
Outlet_Establishment_Year	0.018918	-0.097040	-0.007233	1.000000

```
df.cov()
```

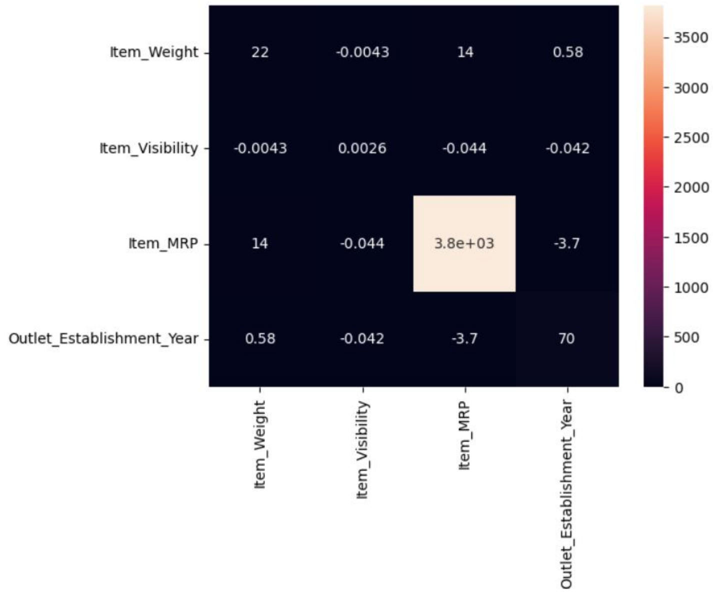
```
<ipython-input-30-6f98a29763d5>:1: FutureWarning: The default value of numeric_only in DataFrame.cov is deprecated. In a future ve
  df.cov()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
Item_Weight	21.760812	-0.004303	14.338892	0.581425
Item_Visibility	-0.004303	0.002627	-0.044391	-0.041640
Item_MRP	14.338892	-0.044391	3820.363680	-3.742832
Outlet_Establishment_Year	0.581425	-0.041640	-3.742832	70.094664

```
df['Item_Weight'].var()
```

```
21.760812343231528
```

```
sns.heatmap(df.cov(),annot=True)
plt.show()
```



CONCLUSION:

The predicted output is displayed using the Descriptive Statistical Analysis model trained with the given dataset and results are verified.

Ex No : 3
DATE: 9/08/2023

LINEAR REGRESSION

AIM: To perform prediction with Linear regression using BigMart Sales prediction Dataset.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe.
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the linear regression model.
5. Load the test dataset and predict the value using the model
6. Plot the output using scatterplot.
7. Display the results of the output predicted from the model.

DATASET DESCRIPTION:

For this Simple linear regression experiment,BigMart Sales Dataset is used. And linear regression model is built.

PROGRAM AND OUTPUT:

```
def mean(values):
    return sum(values)/float(len(values))
```

```
def variance(values,mean):
    return sum([(x-mean)**2 for x in values])
```

```
def covariance(x,mean_x,y,mean_y):
    covar=0.0
    for i in range(len(x)):
        covar+=(x[i]-mean_x) * (y[i]-mean_y)
    return covar
```

```
def coefficients(dataset):
    b1=covariance(x,mean_x,y,mean_y)/variance(x,mean_x)
    b0=mean_y-b1*mean_x
    return[b0,b1]
```

```
def simple_linear_regression(train,test):
    for row in test:
        ytest = b0 + b1 * row[0]
```

```
coefficients:b0=6.068,b1=0.942
Regression equation of y on x : y=6.068+0.942x
```

```
test=[[55]]
result=simple_linear_regression(dataset,test)
print('Value of y when x=55 is %.3f' % (result))
```

```
Value of y when x=55 is 57.899
```

```
[ ]     return ytest
```

```
[ ]  dataset=[[40,38],[50,60],[38,55],[60,70],[65,60],[50,48],[35,30]]
     x=[row[0] for row in dataset]
     y=[row[1] for row in dataset]
     mean_x=mean(x)
     mean_y=mean(y)
```

```
[ ]  variance_x=variance(x,mean_x)
     variance_y=variance(y,mean_y)
     print('x stats:mean=%.3f variance=%.3f' % (mean_x,variance_x))
     print('y stats:mean=%.3f variance=%.3f' % (mean_y,variance_y))
```

```
x stats:mean=48.286 variance=773.429
y stats:mean=51.571 variance=1155.714
```

```
[ ]  covar = covariance(x,mean_x,y,mean_y)
     print('covariance: %.3f' % (covar))
```

```
covariance: 728.857
```

```
[ ]  b0,b1 = coefficients(dataset)
     print('coefficients:b0=%.3f,b1=%.3f' % (b0,b1))
     print('Regression equation of y on x : y=%.3f+%.3fx '% (b0, b1))
```

```
coefficients:b0=6.068,b1=0.942
Regression equation of y on x : y=6.068+0.942x
```

```
[ ]  test=[[55]]
     result=simple_linear_regression(dataset,test)
     print('Value of y when x=55 is %.3f' % (result))
```

```
Value of y when x=55 is 57.899
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
dataset=[[40,38],[50,60],[38,55],[60,70],[65,60],[50,48],[35,30]]
x=np.array([row[0] for row in dataset]).reshape(-1,1)
y=np.array([row[1] for row in dataset])
model = LinearRegression()
model.fit(x, y)
```

```
▾ LinearRegression
LinearRegression()
```

```
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
```

```
coefficient of determination: 0.5943115020664733
```


```
import numpy as np
from sklearn.linear_model import LinearRegression
dataset=[[40,38],[50,60],[38,55],[60,70],[65,60],[50,48],[35,30]]
x=np.array([row[0] for row in dataset]).reshape(-1,1)
y=np.array([row[1] for row in dataset])
model = LinearRegression()
model.fit(x, y)
```

```
▾ LinearRegression
LinearRegression()
```

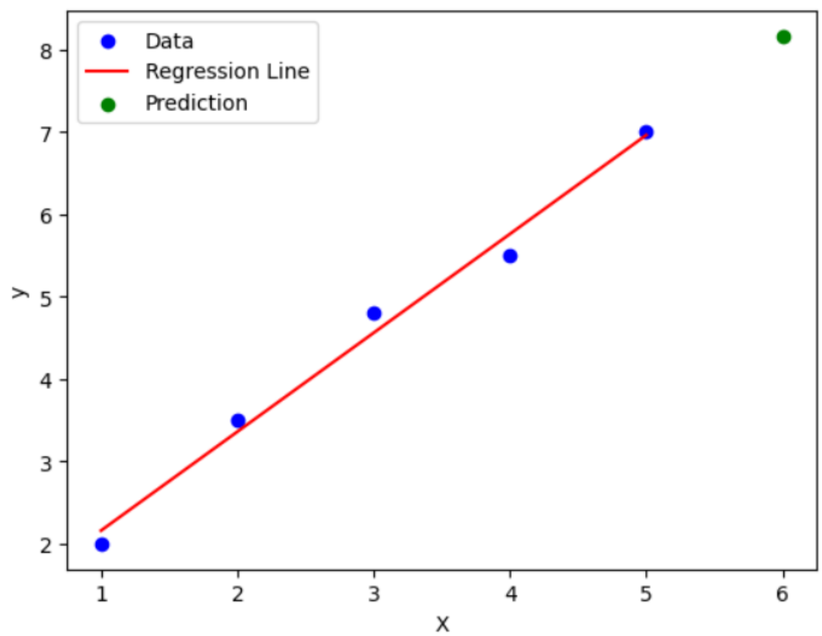
```
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
```

```
coefficient of determination: 0.5943115020664733
```

```
[ ] y_pred = model.predict([[55]])
    print('predicted response:', y_pred)
```

```
predicted response: [57.89878094]
```

```
[ ] import matplotlib.pyplot as plt
    X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
    y = np.array([2, 3.5, 4.8, 5.5, 7])
    model = LinearRegression()
    model.fit(X, y)
    X_new = np.array([6]).reshape(-1, 1)
    y_pred = model.predict(X_new)
    plt.scatter(X, y, color='blue', label='Data')
    plt.plot(X, model.predict(X), color='red', label='Regression Line')
    plt.scatter(X_new, y_pred, color='green', label='Prediction')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.show()

    print(f"Prediction for X={X_new[0][0]}: {y_pred[0]}")
```




```
Prediction for X=6: 8.16
```

<u>Conclusion:</u> We have successfully done the Linear Regression of the BigMart Sales prediction dataset.

Ex No : 4
DATE: 11/09/2023

PRINCIPAL COMPONENT ANALYSIS

AIM: To perform Principal component analysis using BigMart sales dataset.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe. (Dataset used :BigMart Sales Dataset)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the PCA model.
5. Display the heatmap for the correlation.
6. Load the test dataset and predict the value using the model
7. Display the results of the output predicted from the model.

DATASET DESCRIPTION:

1. City type: Stores located in urban or Tier 1 cities should have higher sales because of the higher income levels of people there.
2. Population Density: Stores located in densely populated areas should have higher sales because of more demand.
3. Store Capacity: Stores which are very big in size should have higher sales as they act like one-stop-shops and people would prefer getting everything from one place
4. Competitors: Stores having similar establishments nearby should have less sales because of more competition.
5. Marketing: Stores which have a good marketing division should have higher sales as it will be able to attract customers through the right offers and advertising.
6. Location: Stores located within popular marketplaces should have higher sales because of better access to customers.
7. Customer Behavior: Stores keeping the right set of products to meet the local needs of customers will have higher sales.

8. Ambiance: Stores which are well-maintained and managed by polite and humble people are expected to have higher footfall and thus higher sales.

PROGRAM AND OUTPUT:

```
[4] df = pd.read_csv('Test.csv')
```

```
[5] df.shape
```

```
(5681, 11)
```

```
[6] df.head()
```

›m_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Ou‹
FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	
FDW14	8.300	reg	0.038428	Dairy	87.3198	
NCN55	14.600	Low Fat	0.099575	Others	241.7538	
FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	
FDY38	NaN	Regular	0.118599	Dairy	234.2300	

```
[7] df.corr()
```

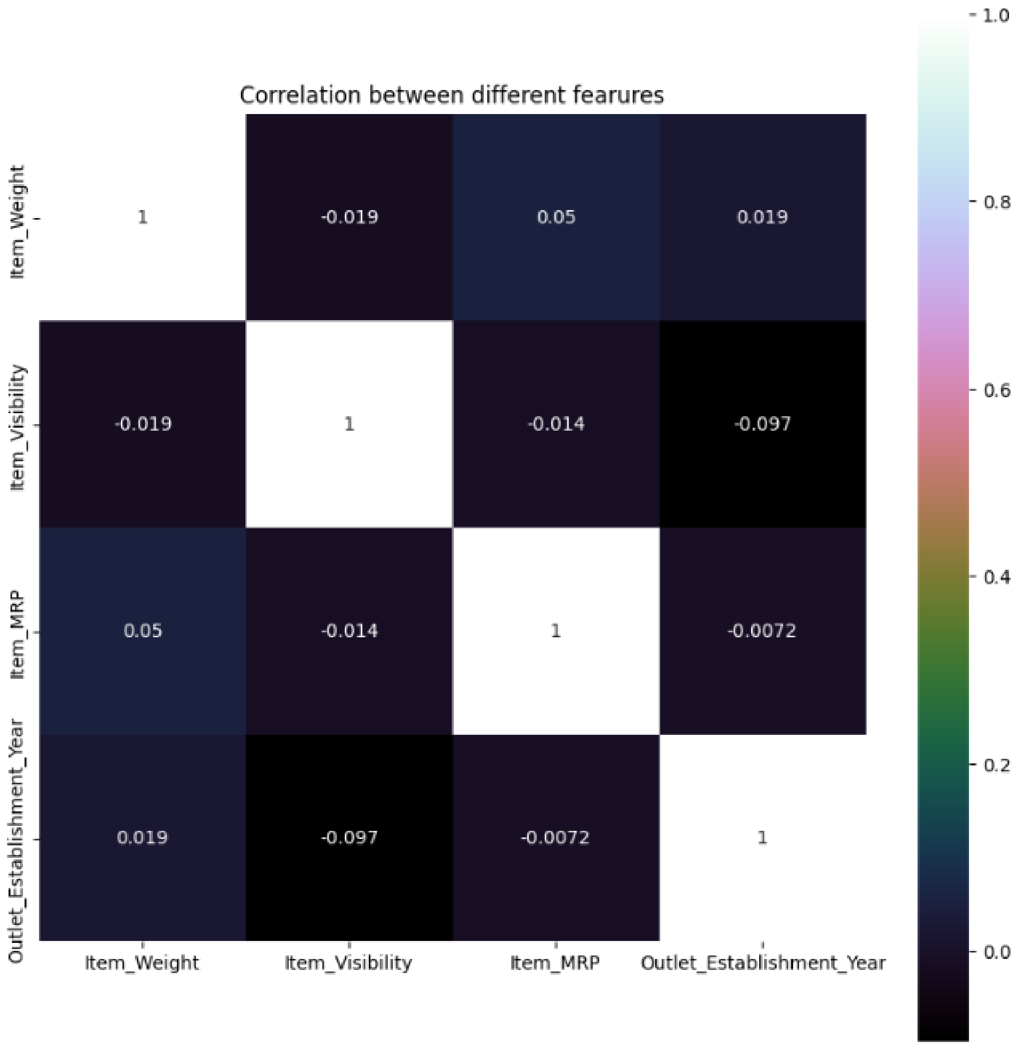
```
<ipython-input-7-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid col
  df.corr()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
Item_Weight	1.000000	-0.019089	0.049809	0.018918
Item_Visibility	-0.019089	1.000000	-0.014013	-0.097040
Item_MRP	0.049809	-0.014013	1.000000	-0.007233
Outlet_Establishment_Year	0.018918	-0.097040	-0.007233	1.000000

Visualising correlation using Seaborn library

```
[8] correlation = df.corr()
    plt.figure(figsize=(10,10))
    sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='cubehelix')

    plt.title('Correlation between different fearures')
```

```
<ipython-input-8-effb445d3340>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid col
  correlation = df.corr()
Text(0.5, 1.0, 'Correlation between different fearures')
```




```
[10] columns_names=df.columns.tolist()
     print("Columns names:")
     print(columns_names)

     Columns names:
     ['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']
```

Doing some visualisation before moving onto PCA

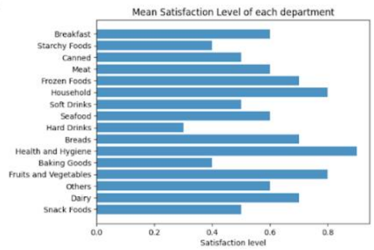
```
df['Item_Type'].unique()

array(['Snack Foods', 'Dairy', 'Others', 'Fruits and Vegetables',
       'Baking Goods', 'Health and Hygiene', 'Breads', 'Hard Drinks',
       'Seafood', 'Soft Drinks', 'Household', 'Frozen Foods', 'Meat',
       'Canned', 'Starchy Foods', 'Breakfast'], dtype=object)
```

```
[12] item=df.groupby('Item_Type').sum
     item

     <bound method GroupBy.sum of <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7bb77bfc2470>>
```

```
[14] groupby_item=df.groupby('Item_Type').mean()
     groupby_item

     <ipython-input-14-fdcdd1015e55>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
       groupby_item=df.groupby('Item_Type').mean()
```

Item_Type	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
Baking Goods	12.272418	0.066016	129.052304	1997.970320
Breads	10.866799	0.074570	142.054893	1997.903030
Breakfast	13.759603	0.077746	132.481526	1998.144737
Canned	12.393565	0.067399	136.915788	1997.409195
Dairy	12.955040	0.070183	145.325889	1998.112335
Frozen Foods	12.101543	0.069854	133.409842	1997.380702
Fruits and Vegetables	13.146659	0.068661	143.867282	1997.943662
Hard Drinks	11.844417	0.066824	137.608950	1997.628378

```python
item_type = ('Snack Foods', 'Dairy', 'Others', 'Fruits and Vegetables', 'Baking Goods', 'Health and Hygiene', 'Breads', 'Hard Drinks', 'Seafood', 'Soft Drinks', 'Household', 'Frozen Foods', 'Meat', 'Canned', 'Starchy Foods', 'Breakfast')

# Assuming you have a list of satisfaction levels for each item type
satisfaction_levels = [0.5, 0.7, 0.6, 0.8, 0.4, 0.9, 0.7, 0.3, 0.6, 0.5, 0.8, 0.7, 0.6, 0.5, 0.4, 0.6]

y_pos = np.arange(len(item_type))

plt.barh(y_pos, satisfaction_levels, align='center', alpha=0.8)
plt.yticks(y_pos, item_type)
plt.xlabel('Satisfaction level')
plt.title('Mean Satisfaction Level of each department')

plt.show()
```



PCA

[ ]

[26]
```python
df.head()
```

	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
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	1998	NaN	Tier 3	Grocery Store
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3

[27]
```python
df_drop=df.drop(labels=['Item_Type','Item_Weight'],axis=1)
df_drop.head()
```

	Item_Identifier	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	FDW58	Low Fat	0.007565	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	FDW14	reg	0.038428	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	NCN55	Low Fat	0.099575	241.7538	OUT010	1998	NaN	Tier 3	Grocery Store
3	FDQ58	Low Fat	0.015388	155.0340	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	FDY38	Regular	0.118599	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3

df.drop() is the method to drop the columns in our dataframe

Now we need to bring "left" column to the front as it is the label and not the feature.

[28]
```python
cols = df_drop.columns.tolist()
cols
```

```
['Item_Identifier',
 'Item_Fat_Content',
 'Item_Visibility',
 'Item_MRP',
 'Outlet_Identifier',
 'Outlet_Establishment_Year',
 'Outlet_Size',
```

```
 'Outlet_Size',
 'Outlet_Location_Type',
 'Outlet_Type']
```

```
[31] cols.insert(0, cols.pop(cols.index('Outlet_Size')))
```

```
[32] cols
```

```
['Outlet_Size',
 'Item_Identifier',
 'Item_Fat_Content',
 'Item_Visibility',
 'Item_MRP',
 'Outlet_Identifier',
 'Outlet_Establishment_Year',
 'Outlet_Location_Type',
 'Outlet_Type']
```

```
df_drop = df_drop.reindex(columns= cols)
```

```
[34] X = df_drop.iloc[:,1:8].values
     y = df_drop.iloc[:,0].values
     X
```

```
array([['FDW58', 'Low Fat', 0.007564836, ..., 'OUT049', 1999, 'Tier 1'],
       ['FDW14', 'reg', 0.038427677, ..., 'OUT017', 2007, 'Tier 2'],
       ['NCN55', 'Low Fat', 0.099574908, ..., 'OUT010', 1998, 'Tier 3'],
       ...,
       ['NCO17', 'Low Fat', 0.073528561, ..., 'OUT045', 2002, 'Tier 2'],
       ['FDJ26', 'Regular', 0.0, ..., 'OUT017', 2007, 'Tier 2'],
       ['FDU37', 'Regular', 0.104720151, ..., 'OUT045', 2002, 'Tier 2']],
      dtype=object)
```

```
[35] y
```

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

```
[36] np.shape(X)
```

```
(5681, 7)
```

```
[37] np.shape(y)
```

```
(5681,)
```

CONCLUSION:

The predicted output is displayed using the Principal Component Analysis model trained with the given dataset and results are verified. Thus the PCA is used to reduce the dimension of the dataset.

Ex No : 5
DATE: 23/08/2023

FACTOR ANALYSIS

AIM: To perform Factor analysis using personality BigMart Sales Prediction Dataset dataset.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe. (Dataset used :BigMart Sales Prediction Dataset)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the Factor analysis model.
5. Display the scatterplot for the eigenvalues.
6. Load the test dataset and predict the value using the model
7. Display the results of the output predicted from the model.

DATASET DESCRIPTION:

Personality dataset is used here for performing factor analysis. It consists of 5 factors with 5 sub factors in each factor, based on the eigen value most suitable factor is picked among the 5.

PROGRAM AND OUTPUT:

In [3]:
```
!pip install factor_analyzer
import pandas as pd
from factor_analyzer import FactorAnalyzer
import matplotlib.pyplot as plt
```

```
Collecting factor_analyzer
  Downloading factor_analyzer-0.5.0.tar.gz (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.5/42.5 kB 1.2 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from factor_analyzer) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from factor_analyzer) (1.11.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from factor_analyzer) (1.23.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from factor_analyzer) (1.2.2)
Collecting pre-commit (from factor_analyzer)
  Downloading pre_commit-3.5.0-py2.py3-none-any.whl (203 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 203.7/203.7 kB 7.6 MB/s eta 0:00:00
```

In [5]:
```
df=pd.read_csv("Test.csv")
```

In [6]:
```
df.columns
```

Out[6]:
```
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'],
      dtype='object')
```

In [7]:
```
df.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

In [8]:
```
df.dropna(inplace=True)
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3099 entries, 0 to 5677
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Weight                3099 non-null   float64
 1   Item_Fat_Content           3099 non-null   object
 2   Item_Visibility            3099 non-null   float64
 3   Item_Type                  3099 non-null   object
 4   Item_MRP                   3099 non-null   float64
 5   Outlet_Establishment_Year  3099 non-null   int64
 6   Outlet_Size                3099 non-null   object
 7   Outlet_Location_Type       3099 non-null   object
 8   Outlet_Type                3099 non-null   object
dtypes: float64(3), int64(1), object(5)
memory usage: 242.1+ KB
```

In [10]:
```
df.head()
```

Out[10]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type
0	20.750	Low Fat	0.007565	Snack Foods	107.8622	1999	Medium	Tier 1
5	9.800	Regular	0.063817	Fruits and Vegetables	117.1492	1997	Small	Tier 1
6	19.350	Regular	0.082602	Baking Goods	50.1034	2009	Medium	Tier 3
13	4.785	Low Fat	0.092738	Breads	122.3098	1999	Medium	Tier 1
14	16.750	LF	0.021206	Hard Drinks	52.0298	1987	High	Tier 3

In [11]:
```
fa=FactorAnalyzer(n_factors=6, rotation="varimax")
```

In [12]:
```
df=df.dropna()
```

In [17]:
```
features = ['Item_Weight', 'Item_Fat_Content', 'Item_Visibility', 'Item_Type', 'Item_MRP',
            'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type']

X = df[features]
```

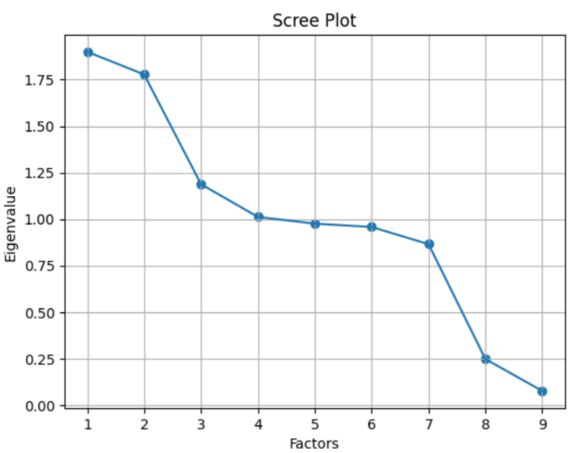
In [18]:
```
fa = FactorAnalyzer(n_factors=3, rotation='varimax')
fa.fit(X)
```

Out[18]:
```
FactorAnalyzer(rotation='varimax', rotation_kwargs={})
```

In [19]:
```
ev, v=fa.get_eigenvalues()
ev
```

Out[19]:
```
array([1.89928956, 1.77763671, 1.18754347, 1.01193277, 0.97572083,
       0.95781427, 0.86561383, 0.24787273, 0.07657583])
```

In [20]:
```
    plt.scatter(range(1, len(ev) + 1), ev)
    plt.plot(range(1, len(ev) + 1), ev)
    plt.title('Scree Plot')
    plt.xlabel('Factors')
    plt.ylabel('Eigenvalue')
    plt.grid()
    plt.show()
```



In [21]:
```
fa=FactorAnalyzer(6, rotation='varimax')
fa.fit(X)
fa.loadings_
```

Out[21]:
```
array([[-1.42928360e-02,  9.39458109e-03, -3.50697247e-03,
        -9.04606083e-02,  2.93460829e-02,  1.87150919e-01],
       [-1.50298409e-04,  3.63175130e-03, -8.00582259e-02,
         2.79209386e-01, -4.02274724e-02, -5.28589190e-02],
```

```
       [ 5.11049647e-03, -8.16034489e-03, -1.08879587e-02,
         1.48708602e-01,  1.86382307e-02, -2.33861901e-02],
       [ 4.01511123e-03, -7.95244326e-03,  8.15774941e-01,
        -2.05708159e-01,  3.71179462e-04,  9.82857323e-02],
       [ 9.02707362e-03, -5.61458218e-03,  3.66441119e-02,
        -7.15897785e-03, -2.28579957e-02,  2.32942212e-01],
       [-3.02179588e-01,  8.99584732e-01, -1.44320562e-02,
        -3.55781810e-02,  3.99339502e-02,  8.08703194e-03],
       [-9.16570071e-01,  2.77061426e-01,  1.04959945e-02,
         2.09698228e-02,  2.78550835e-01,  4.27115707e-04],
       [ 8.07433159e-01,  2.61342539e-01,  1.65114620e-02,
         5.95136923e-02,  2.45408754e-01, -1.37160337e-02],
       [ 4.09349760e-01,  8.73709721e-01,  2.48890300e-03,
        -1.50840011e-02,  9.96376142e-04,  2.95395210e-03]])
```

In [22]:

```
index_values =df.columns[0:25]
```

In [24]:

```
column_values=['Factor1','Factor2','Factor3','Factor4','Factor5','Factor6']
df1=pd.DataFrame(fa.loadings_,index_values,column_values)
df1
```

Out[24]:

	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
Item_Weight	-0.014293	0.009395	-0.003507	-0.090461	0.029346	0.187151
Item_Fat_Content	-0.000150	0.003632	-0.080058	0.279209	-0.040227	-0.052859
Item_Visibility	0.005110	-0.008160	-0.010888	0.148709	0.018638	-0.023386
Item_Type	0.004015	-0.007952	0.815775	-0.205708	0.000371	0.098286
Item_MRP	0.009027	-0.005615	0.036644	-0.007159	-0.022858	0.232942
Outlet_Establishment_Year	-0.302180	0.899585	-0.014432	-0.035578	0.039934	0.008087
Outlet_Size	-0.916570	0.277061	0.010496	0.020970	0.278551	0.000427
Outlet_Location_Type	0.807433	0.261343	0.016511	0.059514	0.245409	-0.013716
Outlet_Type	0.409350	0.873710	0.002489	-0.015084	0.000996	0.002954

CONCLUSION

The predicted output is displayed using the Factor Analysis model trained with the given dataset and results are verified. Thus the Factor Analysis is used to reduce the dimension of the dataset by selecting the most important factors in the dataset based on the latent features.

Ex No : 6
DATE: 29/08/2023

LINEAR PROGRAMMING

AIM: To perform Linear programming in python for the given equations with the constraints and get the optimized values.

PROCEDURE:

1. Import the necessary library functions.
2. If pulp is not available use pip install method and install pulp library and import the entire package
3. Give the required constraints and maximization function to the model
4. View the model constraints and verify it.
5. View the status of the model.
6. Print the results which are calculated by the model
7. Get the optimized values of the given equation and constraints.

LINEAR PROGRAMMING

In [19]:
```
from pulp import LpProblem, LpVariable, lpSum, LpMaximize, LpMinimize
```

In [20]:
```
# Creating a LP problem
prob = LpProblem("Example_LP_Problem", LpMaximize)
```

In [21]:
```
x1 = LpVariable("x1", lowBound=0)  # Variable x1 >= 0
x2 = LpVariable("x2", lowBound=0)  # Variable x2 >= 0
```

In [22]:
```
 # Defining the objective function
prob += 3 * x1 + 2 * x2, "Objective"
```

In [23]:
```
 prob += 3 * x1 + 2 * x2, "Objective"
```

8.

In [ ]:
```
# Defining constraints
prob += 2 * x1 + x2 <= 8, "Constraint_1"
prob += 4 * x1 - 5 * x2 >= -10, "Constraint_2"
prob += -2 * x1 + 3 * x2 == 3, "Constraint_3
```

In [14]:
```
prob.solve()
```

Out[14]: 1

In [16]:
```
# Print the results
print("Status:", prob.status)
print("Objective value:", lpSum([3 * x1, 2 * x2]).value())
print("Decision variables:")
print("x1 =", x1.value())
print("x2 =", x2.value())
```

```
Status: 1
Objective value: 13.375
Decision variables:
x1 = 2.625
x2 = 2.75
```

CONCLUSION: Thus the Linear programming method using python was implemented and the results of various equations and optimized calues was verified successfully.

Ex No : 7
DATE: 5/9/2023

TRANSPORTATION PROBLEM

AIM: To perform Transportation problem in python for the given equations with the constraints and get the optimized values.

PROCEDURE:

1. Import the necessary library functions.
2. If pulp is not available use pip install method and install pulp library and import the entire package
3. Give the required constraints and maximization function to the model
4. View the model constraints and verify it.
5. View the status of the model
6. Print the results which are calculated by the model
7. Get the optimized values of the given equation and constraints.

PROGRAM WITH OUTPUT

Transportation problem

In [2]:
```
pip install pulp
```

```
Collecting pulp
  Downloading PuLP-2.7.0-py3-none-any.whl (14.3 MB)
     ---------------------------------------- 14.3/14.3 MB 23.3 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-2.7.0
```

In [3]:
```
import pulp
```

In [4]:
```
from pulp import*
```

In [5]:
```
from pulp import LpProblem, LpVariable, lpSum, LpMinimize
```

In [6]:
```
# Defining the transportation problem
def solve_transportation_problem(costs, supply, demand):
    prob = LpProblem("Transportation Problem", LpMinimize)
```

In [7]:
```
   suppliers = ["Supplier1", "Supplier2"]
consumers = ["Consumer1", "Consumer2", "Consumer3"]
```

In [8]:
```
x = LpVariable.dicts("shipment", (suppliers, consumers), lowBound=0, cat="Integer")
```

In [9]:
```
# Defining the objective function
costs = {
    ("Supplier1", "Consumer1"): 10,
    ("Supplier1", "Consumer2"): 7,
    ("Supplier1", "Consumer3"): 4,
    ("Supplier2", "Consumer1"): 2,
    ("Supplier2", "Consumer2"): 6,
    ("Supplier2", "Consumer3"): 9,
}
```

In [10]:
```
   prob = LpProblem("Transportation Problem", LpMinimize)
prob += lpSum([x[i][j] * costs[i, j] for i in suppliers for j in consumers]), "Total Cost"
```

```
In [11]: supplies = {
             "Supplier1": 100,
             "Supplier2": 150,
         }
```

```
In [12]: for i in suppliers:
             prob += lpSum(x[i][j] for j in consumers) == supplies[i], f"Supply_{i}"
```

```
In [13]: # Defining demand constraints
         demands = {
             "Consumer1": 50,
             "Consumer2": 100,
             "Consumer3": 100,
         }
```

```
In [14]: for j in consumers:
             prob += lpSum(x[i][j] for i in suppliers) == demands[j], f"Demand_{j}"
```

```
In [15]: # Solving the problem
         prob.solve()

Out[15]: 1
```

```
In [16]: # Printing the results
         print("Status:", LpStatus[prob.status])

         Status: Optimal
```

```
In [17]: for i in suppliers:
             for j in consumers:
                 print(f"Amount from {i} to {j}: {value(x[i][j])}")

         Amount from Supplier1 to Consumer1: 0.0
         Amount from Supplier1 to Consumer2: 0.0
         Amount from Supplier1 to Consumer3: 100.0
         Amount from Supplier2 to Consumer1: 50.0
         Amount from Supplier2 to Consumer2: 100.0
         Amount from Supplier2 to Consumer3: 0.0
```

```
In [18]: print("Total Cost:", value(prob.objective))

         Total Cost: 1100.0
```

RESULT: Thus the transportation problem method using python was implemented and the results of various equations and optimized values was verified successfully.

Ex No : 8
DATE: 12/09/2023

ASSIGNMENT PROBLEM

AIM: To perform Assignment problem in python for the given equations with the constraints and get the optimized values.

PROCEDURE:

1. Import the necessary library functions.
2. If pulp is not available use pip install method and install pulp library and import the entire package
3. Give the required constraints and maximization function to the model
4. View the model constraints and verify it.
5. View the status of the model
6. Print the results which are calculated by the model
7. Get the optimized values of the given equation and constraints.

PROGRAM WITH OUTPUT

ASSIGNMENT PROBLEM

In [27]:
```
from pulp import LpProblem, LpVariable, lpSum, LpMinimize
```

In [28]:
```
prob = LpProblem("Assignment_Problem", LpMinimize)
```

In [29]:
```
workers = ['Worker1', 'Worker2', 'Worker3']
tasks = ['Task1', 'Task2', 'Task3']
```

In [30]:
```
x = LpVariable.dicts("assignment", [(i, j) for i in workers for j in tasks], 0, 1, LpMinimize)
```

In [31]:
```
# Setting the objective function
prob += lpSum(x[i, j] for i in workers for j in tasks)
```

In [32]:
```
# Adding constraints
for i in workers:
    prob += lpSum(x[i, j] for j in tasks) == 1  # Each worker is assigned to exactly one task
```

In [33]:
```
for j in tasks:
    prob += lpSum(x[i, j] for i in workers) == 1  # Each task is assigned to exactly one worker
```

In [34]:
```
# Solving the problem
prob.solve()
```

Out[34]: 1

In [35]:
```
# Printing the results
print("Status:", prob.status)
print("Objective value:", lpSum(x[i, j] for i in workers for j in tasks).value())
print("Assignment:")
for i in workers:
    for j in tasks:
        if x[i, j].value() == 1:
            print(f"{i} is assigned to {j}")
```

```
Status: 1
Objective value: 3.0
Assignment:
Worker1 is assigned to Task2
Worker2 is assigned to Task3
Worker3 is assigned to Task1
```

RESULT: Thus the assignment problem method using python was implemented and the results of various equations and optimized values was verified successfully.

Ex No : 9
DATE: 26/09/2023

HIERARCHICAL CLUSTERING

AIM: To perform Hierarchical Clustering using agglomerative clustering with four types of linkage methods ward,single,average,complete using BigMart Sales prediction Datset

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe.
3. Load the training dataset and fit the data into the hierarchical clustering ,agglomerative clustering model.
4. Display the scatterplot for the two columns.
5. Display the dendogram using agglomerative model.
6. Fit and predict the point in agglomerative clustering model.
7. Display the scatter plot of the clusters.

DATASET DESCRIPTION:

We use BigMart sales prediction dataset from Kaggle for performing the hierarchical clustering methods.

PROGRAM AND OUTPUT:

In [31]:
```
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
```

In [32]:
```
data = pd.read_csv('Test.csv')
```

In [33]:
```
features = ['Item_MRP', 'Item_Weight', 'Item_Visibility']
X = data[features]
```

In [34]:
```
# Handle missing values or other preprocessing steps if necessary
# For simplicity, let's fill missing values with the mean
X.fillna(X.mean(), inplace=True)
```

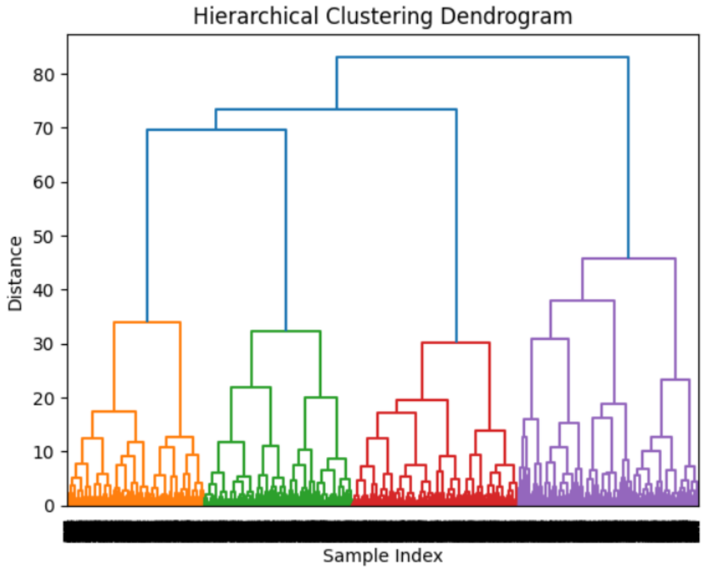
In [35]:
```

# Standardize the data
X = (X - X.mean()) / X.std()
```

In [36]:
```
# Hierarchical clustering using AgglomerativeClustering with Ward's method
agglomerative_cluster = AgglomerativeClustering(n_clusters=3, linkage='ward')
agglomerative_cluster.fit(X)
```

Out[36]:
```
AgglomerativeClustering(n_clusters=3)
```

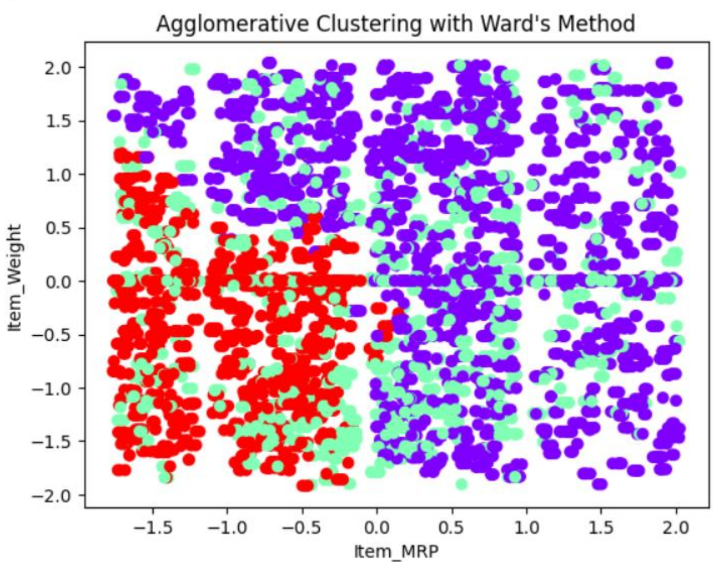
In [37]:

```
# Plot the dendrogram
linked = linkage(X, 'ward')
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



In [38]:

```
# Below, we use the first two features for illustration purposes
plt.scatter(X['Item_MRP'], X['Item_Weight'], c=agglomerative_cluster.labels_, cmap='rainbow')
plt.title('Agglomerative Clustering with Ward\'s Method')
plt.xlabel('Item_MRP')
plt.ylabel('Item_Weight')
plt.show()
```



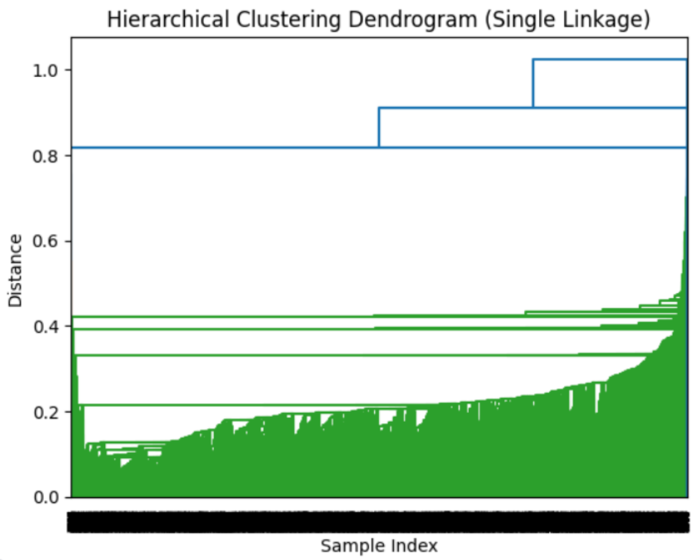
In [39]:

```
# Hierarchical clustering using AgglomerativeClustering with single linkage method
single_linkage_cluster = AgglomerativeClustering(n_clusters=3, linkage='single')
single_linkage_cluster.fit(X)
```

Out[39]: AgglomerativeClustering(linkage='single', n_clusters=3)

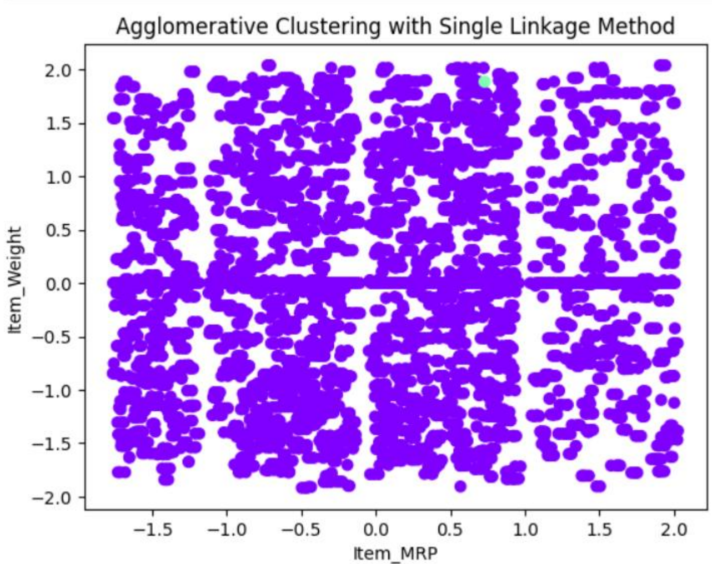
In [40]:

```
# Plot the dendrogram with single linkage method
linked = linkage(X, 'single')
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Single Linkage)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



In [41]:

```
# Below, we use the first two features for illustration purposes
plt.scatter(X['Item_MRP'], X['Item_Weight'], c=single_linkage_cluster.labels_, cmap='rainbow')
plt.title('Agglomerative Clustering with Single Linkage Method')
plt.xlabel('Item_MRP')
plt.ylabel('Item_Weight')
plt.show()
```



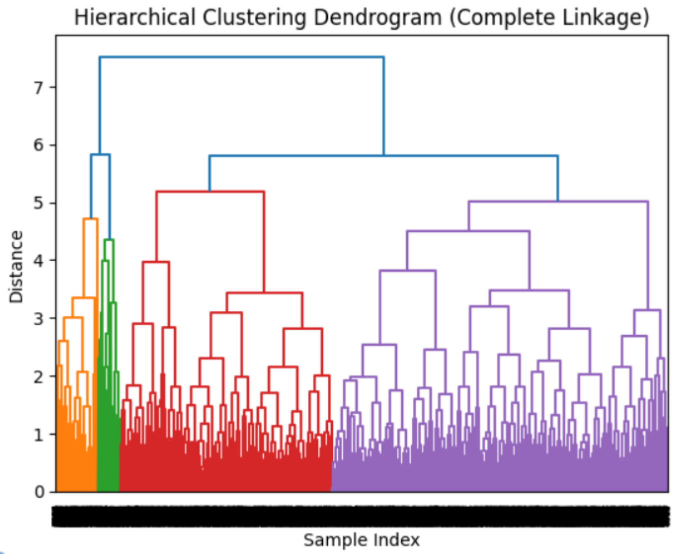
In [42]:

```
# Hierarchical clustering using AgglomerativeClustering with complete linkage method
complete_linkage_cluster = AgglomerativeClustering(n_clusters=3, linkage='complete')
complete_linkage_cluster.fit(X)
```

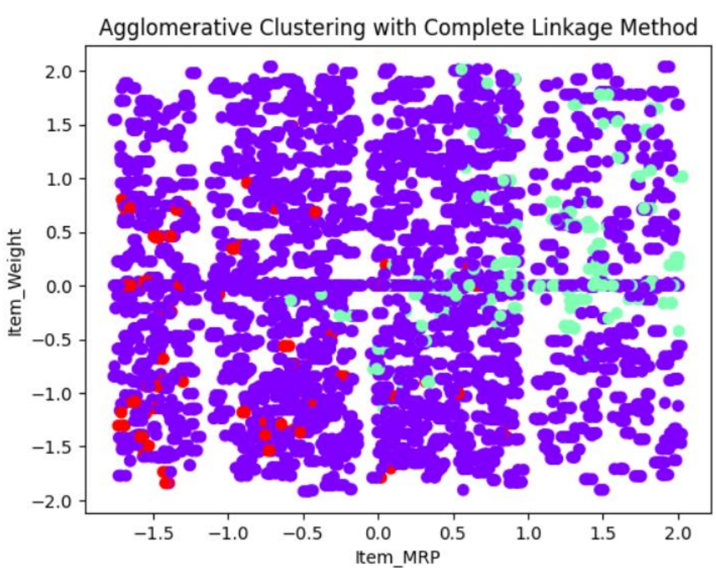
Out[42]:

```
AgglomerativeClustering(linkage='complete', n_clusters=3)
```

In [43]:
```python
# Plot the dendrogram with complete linkage method
linked = linkage(X, 'complete')
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Complete Linkage)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



In [44]:
```python
# Below, we use the first two features for illustration purposes
plt.scatter(X['Item_MRP'], X['Item_Weight'], c=complete_linkage_cluster.labels_, cmap='rainbow')
plt.title('Agglomerative Clustering with Complete Linkage Method')
plt.xlabel('Item_MRP')
plt.ylabel('Item_Weight')
plt.show()
```

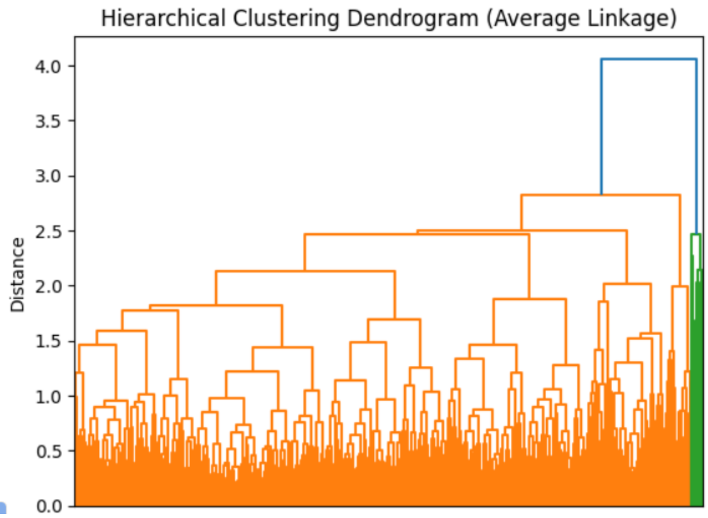


In [45]:
```python
# Hierarchical clustering using AgglomerativeClustering with average linkage method
average_linkage_cluster = AgglomerativeClustering(n_clusters=3, linkage='average')
average_linkage_cluster.fit(X)
```

Out[45]:
```
AgglomerativeClustering(linkage='average', n_clusters=3)
```

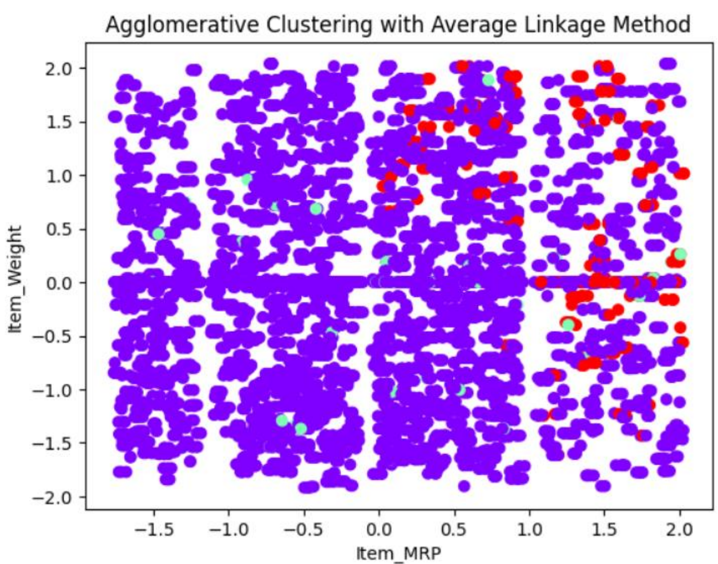
In [46]:

```
# Plot the dendrogram with average linkage method
linked = linkage(X, 'average')
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Average Linkage)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



In [47]:

```
# Below, we use the first two features for illustration purposes
plt.scatter(X['Item_MRP'], X['Item_Weight'], c=average_linkage_cluster.labels_, cmap='rainbow')
plt.title('Agglomerative Clustering with Average Linkage Method')
plt.xlabel('Item_MRP')
plt.ylabel('Item_Weight')
plt.show()
```



CLUSTER EVALUATION

```
In [49]: from sklearn import datasets
         from sklearn.metrics import silhouette_score
```

```
In [50]: from sklearn.cluster import KMeans
```

```
In [51]: iris=datasets.load_iris()
         x=iris.data
         y=iris.data
```

```
In [52]: km=KMeans(n_clusters=3,random_state=42)
         km.fit_predict(x)
```

```
Out[52]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
                2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
                2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

```
In [53]: score =silhouette_score(x,km.labels_,metric='euclidean')
```

```
In [54]: print('Silhoutte Score: %.3f'% score)

Silhoutte Score: 0.553
```

CONCLUSION:

Thus the given dataset is clustered using the hierarchical clustering with agglomerative clustering method with 4 different types of linkage methods called as wards, single, complete , average linkage methods and results were verified.

Ex No : 10
DATE: 17/10/2023

K-MEANS CLUSTERING

AIM: To perform Non-hierarchical clustering using K-Means algorithm

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe. (Dataset used :BigMart Sales prediction dataset)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the K-Means clustering model.
5. Display the scatterplot for the two columns.
6. Using min-maxscaler find the number of cluster required and plot the graph.
7. With the help of the elbow diagram , find the number of clusters needed and do the k-means clustering.

DATASET DESCRIPTION:

The K-Means clustering is implemented using theBigMart sales prediction datset. Dataset consists ofdiferent attributes where we are taking item mrp and item visibility.

PROGRAM WITH OUTPUT:

In [109…
```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
# %matplotlib inline
```

In [110…
```
km=KMeans(n_clusters=3)
km
```

Out[110…
```
KMeans(n_clusters=3)
```

In [113…
```
y_predicted = km.fit_predict(df[['Item_MRP', 'Item_Visibility']])
y_predicted
```

Out[113… array([0, 0, 1, ..., 0, 1, 0], dtype=int32)

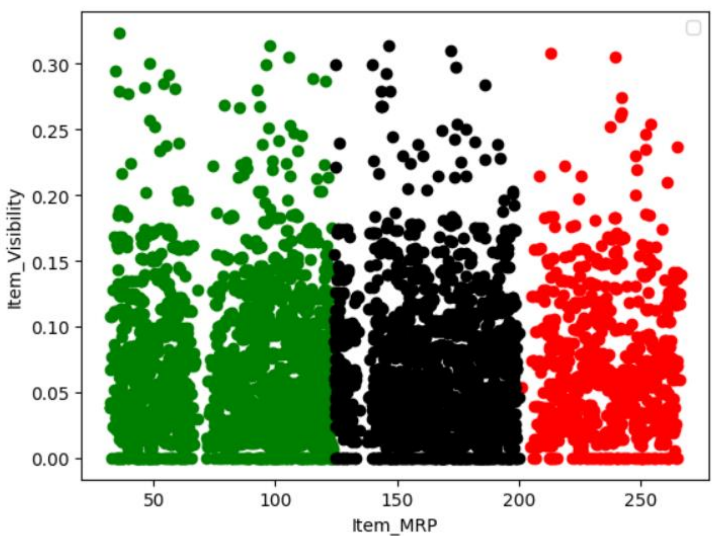
In [117…
```
df['cluster'] = y_predicted
df.head()
```

Out[117…

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Ou
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	
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	

In [118…
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Item_MRP,df1['Item_Visibility'],color='green')
plt.scatter(df2.Item_MRP,df2['Item_Visibility'],color='red')
plt.scatter(df3.Item_MRP,df3['Item_Visibility'],color='black')
#plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='+',label='centroid')
plt.xlabel('Item_MRP')
plt.ylabel('Item_Visibility')
plt.legend()
```

Out[118… <matplotlib.legend.Legend at 0x7cdb2eb33ac0>



In [119…
```
scaler = MinMaxScaler()
scaler.fit(df[['Item_Visibility']])
df['Item_Visibility']=scaler.transform(df['Item_Visibility'].values.reshape(-1,1))
df
scaler.fit(df[['Item_MRP']])
```

Out[119… MinMaxScaler()

In [120…

```
df.Item_MRP = scaler.transform(df['Item_MRP'].values.reshape(-1,1))
df
```

Out[120…

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year
0	FDW58	20.750	Low Fat	0.023374	Snack Foods	0.323413	OUT049	1999
1	FDW14	8.300	reg	0.118737	Dairy	0.235849	OUT017	2007
2	NCN55	14.600	Low Fat	0.307674	Others	0.894140	OUT010	1998
3	FDQ58	7.315	Low Fat	0.047548	Snack Foods	0.524488	OUT017	2007
4	FDY38	NaN	Regular	0.366458	Dairy	0.862069	OUT027	1985
...	...	...	...	...	...	...	...	...
5676	FDB58	10.500	Regular	0.041702	Snack Foods	0.466011	OUT046	1997
5677	FDD47	7.600	Regular	0.441825	Starchy Foods	0.584637	OUT018	2009
5678	NCO17	10.000	Low Fat	0.227194	Health and Hygiene	0.369798	OUT045	2002
5679	FDJ26	15.300	Regular	0.000000	Canned	0.778487	OUT017	2007
5680	FDU37	9.500	Regular	0.323573	Canned	0.203778	OUT045	2002

5681 rows × 12 columns

In [121…

```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['Item_MRP','Item_Visibility']])
y_predicted
```

Out[121…

```
array([1, 1, 2, ..., 1, 2, 0], dtype=int32)
```

In [122…

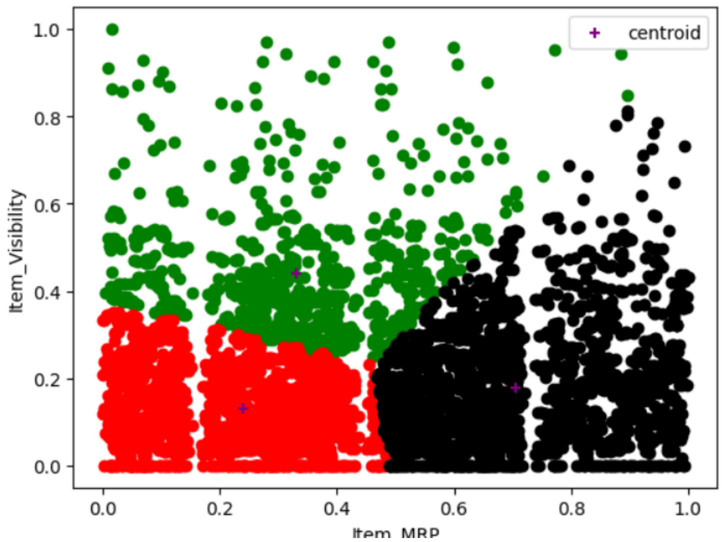
```
df['cluster'] = y_predicted
df
```

Out[122…

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year
0	FDW58	20.750	Low Fat	0.023374	Snack Foods	0.323413	OUT049	1999
1	FDW14	8.300	reg	0.118737	Dairy	0.235849	OUT017	2007
2	NCN55	14.600	Low Fat	0.307674	Others	0.894140	OUT010	1998
3	FDQ58	7.315	Low Fat	0.047548	Snack Foods	0.524488	OUT017	2007
4	FDY38	NaN	Regular	0.366458	Dairy	0.862069	OUT027	1985
...	...	...	...	...	...	...	...	...
5676	FDB58	10.500	Regular	0.041702	Snack Foods	0.466011	OUT046	1997
5677	FDD47	7.600	Regular	0.441825	Starchy Foods	0.584637	OUT018	2009
5678	NCO17	10.000	Low Fat	0.227194	Health and Hygiene	0.369798	OUT045	2002

```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Item_MRP,df1['Item_Visibility'],color='green')
plt.scatter(df2.Item_MRP,df2['Item_Visibility'],color='red')
plt.scatter(df3.Item_MRP,df3['Item_Visibility'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='+',label='centroid')
plt.xlabel('Item_MRP')
plt.ylabel('Item_Visibility')
plt.legend()
```

Out[123… <matplotlib.legend.Legend at 0x7cdb2b8c2a10>



CONCLUSION:

Thus the given dataset is clustered using k-means clustering algorithm and 3 clusters has been grouped.