mlm-1-045015-1

March 21, 2024

[3]:
```
import pandas as pd, numpy as np # For Data Manipulation
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder # For Encoding␣
 ↪Categorical Data [Nominal | Ordinal]
from sklearn.preprocessing import OneHotEncoder # For Creating Dummy Variables␣
 ↪of Categorical Data [Nominal]
from sklearn.impute import SimpleImputer, KNNImputer # For Imputation of␣
 ↪Missing Data
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler #␣
 ↪For Rescaling Data
from sklearn.model_selection import train_test_split # For Splitting Data into␣
 ↪Training & Testing Sets
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv('Amazon Food Dataset.csv')
df
```

[2]:
```
        Custkey     DateKey  Discount Amount Invoice Date  Invoice Number  \
0      10025248   5/23/2017         340.8400   23-05-2017          100080
1      10025063  06-02-2017       16812.4800   02-06-2017          100093
2      10025549  06-02-2017         195.3900   02-06-2017          100094
3      10002489  06-03-2017        -211.7500   03-06-2017          100096
4      10015824  06-12-2017         317.4600   12-06-2017          100130
…           …           …              …           …              …
65275  10025025  05-11-2019        1327.1200   11-05-2019          332837
65276  10020181  05-11-2019         639.8200   11-05-2019          332840
65277  10020181  05-11-2019        1028.5798   11-05-2019          332840
65278  10020181  05-11-2019        1121.3398   11-05-2019          332840
65279  10014469  05-11-2019         579.7500   11-05-2019          332842

      Item Class Item Number                          Item  Line Number  \
0            P01       61762       Carlson Blueberry Yogurt         2000
1            NaN       62058            Big Time Popsicles         2000
2            P01       24335                 Kiwi Scallops         2000
3            P03         NaN                      Kiwi Lox         1000
4            P01       31682                Golden Waffles        15000
…            …           …                            …           …
```

```
65275        P01       17801  Better Fancy Canned Sardines         4000
65276        P01       17801  Better Fancy Canned Sardines         3000
65277        P01       31875  Golden Frozen Chicken Thighs         2000
65278        P01       37441     Atomic Mint Chocolate Bar         1000
65279        P01      274022     Fabulous Strawberry Drink         1000

       List Price  Order Number Promised Delivery Date  Sales Amount  \
0        803.8600        200086              5/23/2017        463.02
1       1293.0000        200101              5/29/2017      14219.52
2        217.1000        200105             06-02-2017        238.81
3          0.0000        200107             06-03-2017        211.75
4        317.4600        200143             06-12-2017        317.46
…             …           …                      …          …
65275   1431.2300        126601             05-11-2019       1535.34
65276   1431.2300        126609             05-11-2019        791.41
65277   1150.4399        126609             05-11-2019       1272.30
65278   1254.1899        126609             05-11-2019       1387.04
65279   1221.3300        126611             05-11-2019        641.58

       Sales Amount Based on List Price  Sales Cost Amount  \
0                              803.8600               0.00
1                            31032.0000               0.00
2                              434.2000               0.00
3                                0.0000               0.00
4                              634.9200               0.00
…                                   …                 …
65275                         2862.4600             899.38
65276                         1431.2300             449.69
65277                         2300.8798             640.09
65278                         2508.3798             688.55
65279                         1221.3300             316.32

       Sales Margin Amount  Sales Price  Sales Quantity  Sales Rep U/M
0                   463.02      463.020               1        145  EA
1                 14219.52      592.480              24        162  EA
2                   238.81      119.405               2        103  EA
3                   211.75      211.750               1        160  EA
4                   317.46      158.730               2        103  EA
…                      …          …             …          …  ..
65275               635.96      767.670               2        110  EA
65276               341.72      791.410               1        115  EA
65277               632.21      636.150               2        115  EA
65278               698.49      693.520               2        115  EA
65279               325.26      641.580               1        145  EA

[65280 rows x 20 columns]
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65280 entries, 0 to 65279
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Custkey                         65280 non-null  int64
 1   DateKey                         65280 non-null  object
 2   Discount Amount                 65279 non-null  float64
 3   Invoice Date                    65280 non-null  object
 4   Invoice Number                  65280 non-null  int64
 5   Item Class                      56995 non-null  object
 6   Item Number                     65240 non-null  object
 7   Item                            65280 non-null  object
 8   Line Number                     65280 non-null  int64
 9   List Price                      65280 non-null  float64
 10  Order Number                    65280 non-null  int64
 11  Promised Delivery Date          65280 non-null  object
 12  Sales Amount                    65280 non-null  float64
 13  Sales Amount Based on List Price  65280 non-null  float64
 14  Sales Cost Amount               65280 non-null  float64
 15  Sales Margin Amount             65280 non-null  float64
 16  Sales Price                     65279 non-null  float64
 17  Sales Quantity                  65280 non-null  int64
 18  Sales Rep                       65280 non-null  int64
 19  U/M                             65280 non-null  object
dtypes: float64(7), int64(6), object(7)
memory usage: 10.0+ MB
```

[5]:
```
df.info() # Dataframe Information (Provide Information on Missing Data)
variable_missing_data = df.isna().sum(); variable_missing_data # Variable-wise
 ↪Missing Data Information
record_missing_data = df.isna().sum(axis=1).sort_values(ascending=False).
 ↪head(5); record_missing_data # Record-wise Missing Data Information (Top 5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65280 entries, 0 to 65279
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Custkey                         65280 non-null  int64
 1   DateKey                         65280 non-null  object
 2   Discount Amount                 65279 non-null  float64
 3   Invoice Date                    65280 non-null  object
 4   Invoice Number                  65280 non-null  int64
 5   Item Class                      56995 non-null  object
```

```
 6   Item Number                       65240 non-null  object
 7   Item                              65280 non-null  object
 8   Line Number                       65280 non-null  int64
 9   List Price                        65280 non-null  float64
 10  Order Number                      65280 non-null  int64
 11  Promised Delivery Date            65280 non-null  object
 12  Sales Amount                      65280 non-null  float64
 13  Sales Amount Based on List Price  65280 non-null  float64
 14  Sales Cost Amount                 65280 non-null  float64
 15  Sales Margin Amount               65280 non-null  float64
 16  Sales Price                       65279 non-null  float64
 17  Sales Quantity                    65280 non-null  int64
 18  Sales Rep                         65280 non-null  int64
 19  U/M                               65280 non-null  object
dtypes: float64(7), int64(6), object(7)
memory usage: 10.0+ MB
```

[5]:

```
9138    4
32      2
5474    2
8933    2
4974    2
dtype: int64
```

[6]:

```
variable_missing_data = df.isna().sum(); variable_missing_data # Variable-wise
↪Missing Data Information
```

[6]:

```
Custkey                             0
DateKey                             0
Discount Amount                     1
Invoice Date                        0
Invoice Number                      0
Item Class                       8285
Item Number                        40
Item                                0
Line Number                         0
List Price                          0
Order Number                        0
Promised Delivery Date              0
Sales Amount                        0
Sales Amount Based on List Price    0
Sales Cost Amount                   0
Sales Margin Amount                 0
Sales Price                         1
Sales Quantity                      0
Sales Rep                           0
U/M                                 0
```

```
dtype: int64
```

[7]:
```python
import pandas as pd


df = pd.read_csv('Amazon Food Dataset.csv')

# Display data types of each column
print(df.dtypes)
```

```
Custkey                              int64
DateKey                             object
Discount Amount                    float64
Invoice Date                        object
Invoice Number                       int64
Item Class                          object
Item Number                         object
Item                                object
Line Number                          int64
List Price                         float64
Order Number                         int64
Promised Delivery Date              object
Sales Amount                       float64
Sales Amount Based on List Price   float64
Sales Cost Amount                  float64
Sales Margin Amount                float64
Sales Price                        float64
Sales Quantity                       int64
Sales Rep                            int64
U/M                                 object
dtype: object
```

[8]:
```python
# Extract categorical columns
categorical_columns = df.select_dtypes(include=['object', 'category', 'bool']).
 ↪columns

# Extract non-categorical columns
non_categorical_columns = df.select_dtypes(exclude=['object', 'category',␣
 ↪'bool']).columns

# Display the results
print("Categorical Columns:")
print(categorical_columns)

print("\nNon-Categorical Columns:")
print(non_categorical_columns)
```

```
Categorical Columns:
Index(['DateKey', 'Invoice Date', 'Item Class', 'Item Number', 'Item',
       'Promised Delivery Date', 'U/M'],
      dtype='object')

Non-Categorical Columns:
Index(['Custkey', 'Discount Amount', 'Invoice Number', 'Line Number',
       'List Price', 'Order Number', 'Sales Amount',
       'Sales Amount Based on List Price', 'Sales Cost Amount',
       'Sales Margin Amount', 'Sales Price', 'Sales Quantity', 'Sales Rep'],
      dtype='object')
```

[9]:
```
df_cat=df[['DateKey', 'Invoice Date', 'Item Class', 'Item Number','Item' ,␣
 ↪'Promised Delivery Date']]
df_cat
```

[9]:
```
         DateKey Invoice Date Item Class Item Number  \
0      5/23/2017   23-05-2017        P01       61762
1     06-02-2017   02-06-2017        NaN       62058
2     06-02-2017   02-06-2017        P01       24335
3     06-03-2017   03-06-2017        P03         NaN
4     06-12-2017   12-06-2017        P01       31682
…            …            …          …         …
65275 05-11-2019   11-05-2019        P01       17801
65276 05-11-2019   11-05-2019        P01       17801
65277 05-11-2019   11-05-2019        P01       31875
65278 05-11-2019   11-05-2019        P01       37441
65279 05-11-2019   11-05-2019        P01      274022

                             Item Promised Delivery Date
0          Carlson Blueberry Yogurt              5/23/2017
1                Big Time Popsicles              5/29/2017
2                     Kiwi Scallops             06-02-2017
3                          Kiwi Lox             06-03-2017
4                    Golden Waffles             06-12-2017
…                               …                    …
65275  Better Fancy Canned Sardines             05-11-2019
65276  Better Fancy Canned Sardines             05-11-2019
65277  Golden Frozen Chicken Thighs             05-11-2019
65278      Atomic Mint Chocolate Bar             05-11-2019
65279     Fabulous Strawberry Drink             05-11-2019

[65280 rows x 6 columns]
```

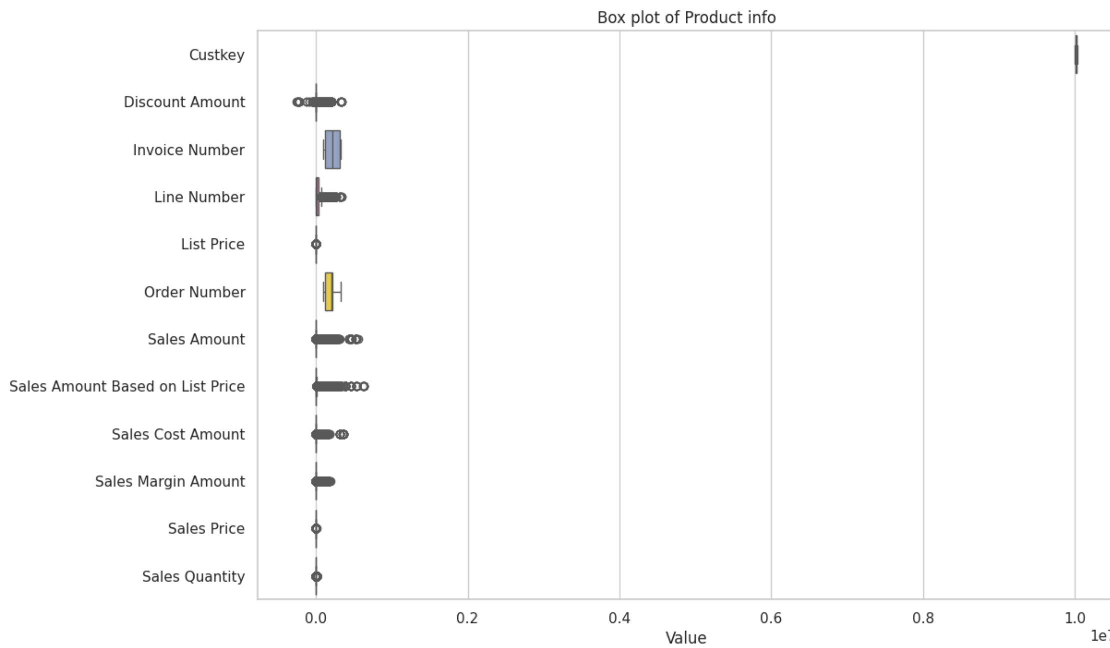
[10]:
```
df_non_cat=df[['Custkey', 'Discount Amount', 'Invoice Number', 'Line Number',
```

```
        'List Price', 'Order Number','Sales Amount','Sales Amount Based on List␣
↪Price','Sales Cost Amount','Sales Margin Amount','Sales Price','Sales␣
↪Quantity']]
df_non_cat
```

[10]:

```
        Custkey  Discount Amount  Invoice Number  Line Number  List Price  \
0      10025248         340.8400          100080         2000    803.8600
1      10025063       16812.4800          100093         2000   1293.0000
2      10025549         195.3900          100094         2000    217.1000
3      10002489        -211.7500          100096         1000      0.0000
4      10015824         317.4600          100130        15000    317.4600
…           …              …               …            …          …
65275  10025025        1327.1200          332837         4000   1431.2300
65276  10020181         639.8200          332840         3000   1431.2300
65277  10020181        1028.5798          332840         2000   1150.4399
65278  10020181        1121.3398          332840         1000   1254.1899
65279  10014469         579.7500          332842         1000   1221.3300

       Order Number  Sales Amount  Sales Amount Based on List Price  \
0            200086        463.02                          803.8600
1            200101      14219.52                        31032.0000
2            200105        238.81                          434.2000
3            200107        211.75                            0.0000
4            200143        317.46                          634.9200
…               …             …                               …
65275        126601       1535.34                         2862.4600
65276        126609        791.41                         1431.2300
65277        126609       1272.30                         2300.8798
65278        126609       1387.04                         2508.3798
65279        126611        641.58                         1221.3300

       Sales Cost Amount  Sales Margin Amount  Sales Price  Sales Quantity
0                   0.00               463.02      463.020               1
1                   0.00             14219.52      592.480              24
2                   0.00               238.81      119.405               2
3                   0.00               211.75      211.750               1
4                   0.00               317.46      158.730               2
…                    …                   …           …             …
65275             899.38               635.96      767.670               2
65276             449.69               341.72      791.410               1
65277             640.09               632.21      636.150               2
65278             688.55               698.49      693.520               2
65279             316.32               325.26      641.580               1

[65280 rows x 12 columns]
```

[11]:
```
columns = ['Custkey', 'Discount Amount', 'Invoice Number', 'Line Number',
       'List Price', 'Order Number','Sales Amount','Sales Amount Based on List␣
 ↪Price','Sales Cost Amount','Sales Margin Amount','Sales Price','Sales␣
 ↪Quantity']
# Creating box plots for each column
plt.figure(figsize=(12, 8))
sns.set(style="whitegrid")
sns.boxplot(data=df_non_cat[columns], orient="h", palette="Set2")
plt.title("Box plot of Product info")
plt.xlabel("Value")
plt.show()
```



[12]:
```
df_non_cat_mdt=df_non_cat[['Custkey', 'Discount Amount', 'Invoice Number',␣
 ↪'Line Number',
       'List Price', 'Order Number','Sales Amount','Sales Amount Based on List␣
 ↪Price','Sales Cost Amount','Sales Margin Amount','Sales Price','Sales␣
 ↪Quantity']]
rs = RobustScaler(quantile_range=(10.0, 90.0)) # quantile_range=(25.0, 75.0) -␣
 ↪Default Range
rs_fit = rs.fit_transform(df_non_cat_mdt[['Custkey', 'Discount Amount',␣
 ↪'Invoice Number', 'Line Number',
       'List Price', 'Order Number','Sales Amount','Sales Amount Based on List␣
 ↪Price','Sales Cost Amount','Sales Margin Amount','Sales Price','Sales␣
 ↪Quantity']])
```

```
df_non_cat_robust_norm = pd.DataFrame(rs_fit, columns=df_non_cat_mdt.
 ↪columns+'_y'); df_non_cat_robust_norm
df_non_cat_mdt_rn = df_non_cat_robust_norm
```

[13]:
```
df_cat_ppd = df_cat.copy(); df_cat_ppd # Preferred Data Subset
```

[13]:
```
          DateKey Invoice Date Item Class Item Number  \
0       5/23/2017   23-05-2017        P01       61762
1      06-02-2017   02-06-2017        NaN       62058
2      06-02-2017   02-06-2017        P01       24335
3      06-03-2017   03-06-2017        P03         NaN
4      06-12-2017   12-06-2017        P01       31682
…             …          …        …        …
65275  05-11-2019   11-05-2019        P01       17801
65276  05-11-2019   11-05-2019        P01       17801
65277  05-11-2019   11-05-2019        P01       31875
65278  05-11-2019   11-05-2019        P01       37441
65279  05-11-2019   11-05-2019        P01      274022

                            Item Promised Delivery Date
0          Carlson Blueberry Yogurt              5/23/2017
1                Big Time Popsicles              5/29/2017
2                     Kiwi Scallops             06-02-2017
3                          Kiwi Lox             06-03-2017
4                    Golden Waffles             06-12-2017
…                                …                    …
65275  Better Fancy Canned Sardines             05-11-2019
65276  Better Fancy Canned Sardines             05-11-2019
65277  Golden Frozen Chicken Thighs             05-11-2019
65278      Atomic Mint Chocolate Bar             05-11-2019
65279     Fabulous Strawberry Drink             05-11-2019

[65280 rows x 6 columns]
```

[14]:
```
# Pre-Processed Non-Categorical Data Subset
df_non_cat_ppd = df_non_cat_mdt_rn.copy(); df_non_cat_ppd # Preferred Data␣
 ↪Subset
```

[14]:
```
       Custkey_y  Discount Amount_y  Invoice Number_y  Line Number_y  \
0       0.302618          -0.037402         -0.563018      -0.172414
1       0.292590           6.067135         -0.562959      -0.172414
2       0.318933          -0.091307         -0.562954      -0.172414
3      -0.930999          -0.242197         -0.562945      -0.189655
4      -0.208196          -0.046067         -0.562789       0.051724
…            …                …                …             …
65275   0.290531           0.328122          0.504218      -0.137931
65276   0.027969           0.073403          0.504232      -0.155172
```

```
65277   0.027969           0.217481          0.504232      -0.172414
65278   0.027969           0.251858          0.504232      -0.189655
65279  -0.281641           0.051140          0.504241      -0.189655

       List Price_y  Order Number_y  Sales Amount_y  \
0          0.403982       -0.017189       -0.025394
1          0.816800       -0.017118        3.815498
2         -0.091224       -0.017099       -0.087994
3         -0.274450       -0.017089       -0.095550
4         -0.006524       -0.016918       -0.066035
…               …              …              …
65275      0.933462       -0.366514        0.274004
65276      0.933462       -0.366476        0.066295
65277      0.696484       -0.366476        0.200562
65278      0.784046       -0.366476        0.232598
65279      0.756313       -0.366466        0.024461

       Sales Amount Based on List Price_y  Sales Cost Amount_y  \
0                               -0.030692            -0.147315
1                                4.744188            -0.147315
2                               -0.089084            -0.147315
3                               -0.157671            -0.147315
4                               -0.057378            -0.147315
…                                    …                    …
65275                            0.294487             0.287756
65276                            0.068408             0.070221
65277                            0.205779             0.162326
65278                            0.238556             0.185768
65279                            0.035252             0.005703

       Sales Margin Amount_y  Sales Price_y  Sales Quantity_y
0                   0.140334       0.444183         -0.068966
1                   9.055978       0.650096          0.724138
2                  -0.004977      -0.102356         -0.034483
3                  -0.022515       0.044524         -0.068966
4                   0.045996      -0.039808         -0.034483
…                      …              …              …
65275               0.252417       0.928746         -0.034483
65276               0.061719       0.966506         -0.068966
65277               0.249987       0.719556         -0.034483
65278               0.292943       0.810806         -0.034483
65279               0.051051       0.728193         -0.068966

[65280 rows x 12 columns]
```

[15]:

```
df_ppd = pd.merge(df_cat_ppd, df_non_cat_ppd, left_index=True,
 ↪right_index=True); df_ppd
```

```
df_ppd
```

[15]:
```
            DateKey Invoice Date Item Class Item Number  \
0       5/23/2017   23-05-2017        P01       61762
1      06-02-2017   02-06-2017        NaN       62058
2      06-02-2017   02-06-2017        P01       24335
3      06-03-2017   03-06-2017        P03         NaN
4      06-12-2017   12-06-2017        P01       31682
…             …            …          …           …
65275  05-11-2019   11-05-2019        P01       17801
65276  05-11-2019   11-05-2019        P01       17801
65277  05-11-2019   11-05-2019        P01       31875
65278  05-11-2019   11-05-2019        P01       37441
65279  05-11-2019   11-05-2019        P01      274022

                                Item Promised Delivery Date  Custkey_y  \
0         Carlson Blueberry Yogurt             5/23/2017   0.302618
1                Big Time Popsicles            5/29/2017   0.292590
2                     Kiwi Scallops           06-02-2017   0.318933
3                          Kiwi Lox           06-03-2017  -0.930999
4                    Golden Waffles           06-12-2017  -0.208196
…                                …                     …          …
65275  Better Fancy Canned Sardines           05-11-2019   0.290531
65276  Better Fancy Canned Sardines           05-11-2019   0.027969
65277  Golden Frozen Chicken Thighs           05-11-2019   0.027969
65278     Atomic Mint Chocolate Bar           05-11-2019   0.027969
65279     Fabulous Strawberry Drink           05-11-2019  -0.281641

       Discount Amount_y  Invoice Number_y  Line Number_y  List Price_y  \
0              -0.037402         -0.563018      -0.172414      0.403982
1               6.067135         -0.562959      -0.172414      0.816800
2              -0.091307         -0.562954      -0.172414     -0.091224
3              -0.242197         -0.562945      -0.189655     -0.274450
4              -0.046067         -0.562789       0.051724     -0.006524
…                    …                …             …            …
65275           0.328122          0.504218      -0.137931      0.933462
65276           0.073403          0.504232      -0.155172      0.933462
65277           0.217481          0.504232      -0.172414      0.696484
65278           0.251858          0.504232      -0.189655      0.784046
65279           0.051140          0.504241      -0.189655      0.756313

       Order Number_y  Sales Amount_y  Sales Amount Based on List Price_y  \
0           -0.017189       -0.025394                           -0.030692
1           -0.017118        3.815498                            4.744188
2           -0.017099       -0.087994                           -0.089084
3           -0.017089       -0.095550                           -0.157671
4           -0.016918       -0.066035                           -0.057378
```

```
…                …            …                                    …
65275       -0.366514     0.274004                                  0.294487
65276       -0.366476     0.066295                                  0.068408
65277       -0.366476     0.200562                                  0.205779
65278       -0.366476     0.232598                                  0.238556
65279       -0.366466     0.024461                                  0.035252

       Sales Cost Amount_y  Sales Margin Amount_y  Sales Price_y  \
0                -0.147315              0.140334       0.444183
1                -0.147315              9.055978       0.650096
2                -0.147315             -0.004977      -0.102356
3                -0.147315             -0.022515       0.044524
4                -0.147315              0.045996      -0.039808
…                     …                     …             …
65275             0.287756              0.252417       0.928746
65276             0.070221              0.061719       0.966506
65277             0.162326              0.249987       0.719556
65278             0.185768              0.292943       0.810806
65279             0.005703              0.051051       0.728193

       Sales Quantity_y
0             -0.068966
1              0.724138
2             -0.034483
3             -0.068966
4             -0.034483
…                  …
65275         -0.034483
65276         -0.068966
65277         -0.034483
65278         -0.034483
65279         -0.068966

[65280 rows x 18 columns]
```

[16]:
```python
# Dataset Used : df_ppd

train_df, test_df = train_test_split(df_ppd, test_size=0.25, random_state=1234)
train_df # Training Dataset
test_df # Testing Dataset
```

[16]:
```
          DateKey Invoice Date Item Class Item Number  \
20465   2/20/2017   20-02-2017        P01       28929
53981  02-02-2019   02-02-2019        P01       17801
28651   4/17/2017   17-04-2017        P01       34901
42835   7/28/2019   28-07-2019        P01       39680
61632  12/17/2019   17-12-2019        P01       29754
```

…	…	…	…	…
31259	1/21/2018	21-01-2018	P01	47801
45510	09-03-2019	03-09-2019	P01	25300
55874	2/25/2019	25-02-2019	P01	39900
19918	02-12-2017	12-02-2017	P01	38789
25661	12-03-2017	03-12-2017	P01	67550

	Item	Promised Delivery Date	Custkey_y \
20465	Nationeel Potato Chips	2/20/2017	0.209930
53981	Better Fancy Canned Sardines	02-02-2019	-1.038755
28651	Better Noodle Soup	4/18/2017	-0.794569
42835	Even Better String Cheese	7/28/2019	0.151282
61632	BBB Best Pepper	12/17/2019	0.209171
…	…	…	…
31259	Red Spade Foot-Long Hot Dogs	1/21/2018	-0.113231
45510	Fast Dried Apples	09-03-2019	-0.475798
55874	Washington Cranberry Juice	2/26/2019	-0.025530
19918	Ebony Green Pepper	02-12-2017	0.235297
25661	Discover Manicotti	12-03-2017	-0.154914

	Discount Amount_y	Invoice Number_y	Line Number_y	List Price_y \
20465	0.070557	-0.456729	-0.172414	-0.242919
53981	0.310522	0.448512	-0.155172	0.933462
28651	-0.058430	-0.417727	-0.172414	-0.230665
42835	-0.071131	0.389056	0.103448	-0.052360
61632	-0.078250	0.483727	0.206897	0.135583
…	…	…	…	…
31259	0.081649	-0.007192	0.137931	1.099394
45510	-0.064608	0.404650	-0.189655	-0.106154
55874	-0.065023	0.458311	0.413793	-0.116619
19918	-0.073903	-0.459705	0.172414	-0.065931
25661	0.264233	-0.434248	0.224138	0.801693

	Order Number_y	Sales Amount_y	Sales Amount Based on List Price_y \
20465	0.070393	0.002627	0.031175
53981	-0.410252	0.287263	0.294487
28651	0.100669	-0.089142	-0.075720
42835	-0.473918	-0.077480	-0.074536
61632	-0.388343	-0.083413	-0.080927
…	…	…	…
31259	0.542661	0.114977	0.099465
45510	-0.449774	-0.062310	-0.063174
55874	-0.402551	-0.072384	-0.069050
19918	0.072670	-0.084370	-0.079616
25661	0.089830	0.234952	0.245162

	Sales Cost Amount_y	Sales Margin Amount_y	Sales Price_y \

```
20465          -0.032343            0.051343     -0.264274
53981           0.287756            0.283196      0.966514
28651          -0.087432           -0.087870     -0.254946
42835          -0.049091           -0.112167     -0.072406
61632          -0.068706           -0.099659      0.113665
…                  …                   …            …
31259           0.082827            0.157833      1.243835
45510          -0.066138           -0.054117     -0.116891
55874          -0.070448           -0.071726     -0.136020
19918          -0.068049           -0.102763     -0.092034
25661           0.193789            0.287661      0.817510

       Sales Quantity_y
20465          1.000000
53981         -0.034483
28651          0.241379
42835         -0.034483
61632         -0.068966
…                 …
31259         -0.068966
45510          0.000000
55874          0.000000
19918         -0.034483
25661         -0.034483

[16320 rows x 18 columns]
```

[17]:

```python
# Required Libraries
import pandas as pd, numpy as np # For Data Manipulation
import matplotlib.pyplot as plt, seaborn as sns # For Data Visualization
import scipy.cluster.hierarchy as sch # For Hierarchical Clustering
from sklearn.cluster import AgglomerativeClustering as agclus, KMeans as kmclus␣
↪# For Agglomerative & K-Means Clustering
from sklearn.metrics import silhouette_score as sscore, davies_bouldin_score as␣
↪dbscore # For Clustering Model Evaluation
```

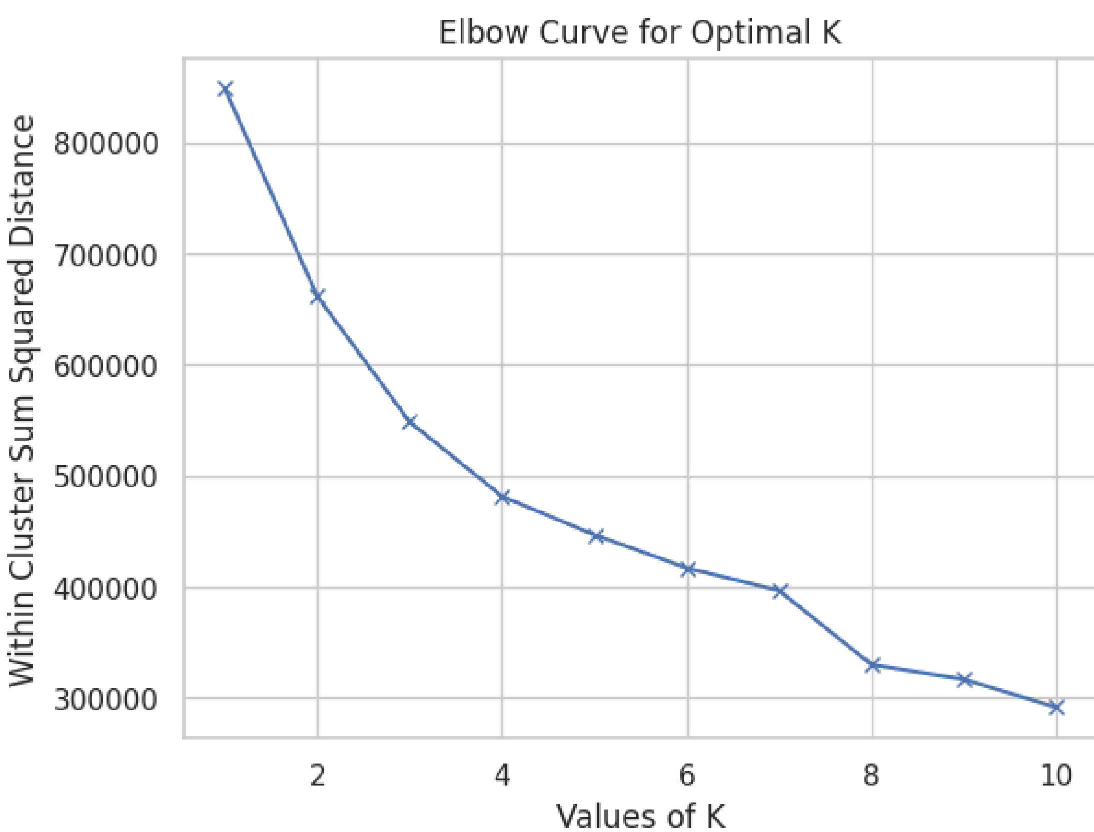
[18]:

```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt

# Assuming you have already read the CSV file into a DataFrame
df = pd.read_csv('Amazon Food Dataset.csv')

# Extracting only numerical columns for clustering
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
df_numerical = df[numerical_columns]
```

```
# Handling missing values if needed
df_numerical = df_numerical.fillna(0)  # Replace NaN with 0 or use other␣
 ↪strategies

# Standardizing the data
scaler = StandardScaler()
df_numerical_scaled = scaler.fit_transform(df_numerical)

# K-means clustering
wcssd = []  # Within-Cluster-Sum-Squared-Distance
nr_clus = range(1, 11)  # Number of Clusters

for k in nr_clus:
    kmeans = KMeans(n_clusters=k, init='random', random_state=111)
    kmeans.fit(df_numerical_scaled)
    wcssd.append(kmeans.inertia_)

# Plotting the Elbow Curve
plt.plot(nr_clus, wcssd, marker='x')
plt.xlabel('Values of K')
plt.ylabel('Within Cluster Sum Squared Distance')
plt.title('Elbow Curve for Optimal K')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
```

```
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



[19]:

```python
import pandas as pd


df = pd.read_csv('Amazon Food Dataset.csv')

# Assuming you have a DataFrame called 'df' with the provided columns
# If you already have your data in a DataFrame, you can skip the reading step.

# Select relevant columns for dropping NaN or blank values
columns_to_clean = ['Discount Amount', 'Invoice Number', 'Line␣
↪Number','DateKey', 'Invoice Date', 'Item Class', 'Item Number','Item' ,␣
↪'Promised Delivery Date',
       'List Price', 'Order Number','Sales Amount','Sales Amount Based on List␣
↪Price','Sales Cost Amount','Sales Margin Amount','Sales Price','Sales␣
↪Quantity']

# Drop rows with NaN or blank values in specified columns
df_cleaned = df.dropna(subset=columns_to_clean)

# If you have blank values represented as empty strings, you can drop those as␣
↪well
df_cleaned = df_cleaned.replace(r'^\s*$', pd.NA, regex=True).
↪dropna(subset=columns_to_clean)

# Now, df_cleaned contains the DataFrame with rows dropped for NaN or blank␣
↪values in specified columns

# Optional: Check the cleaned DataFrame
print(df_cleaned)
```

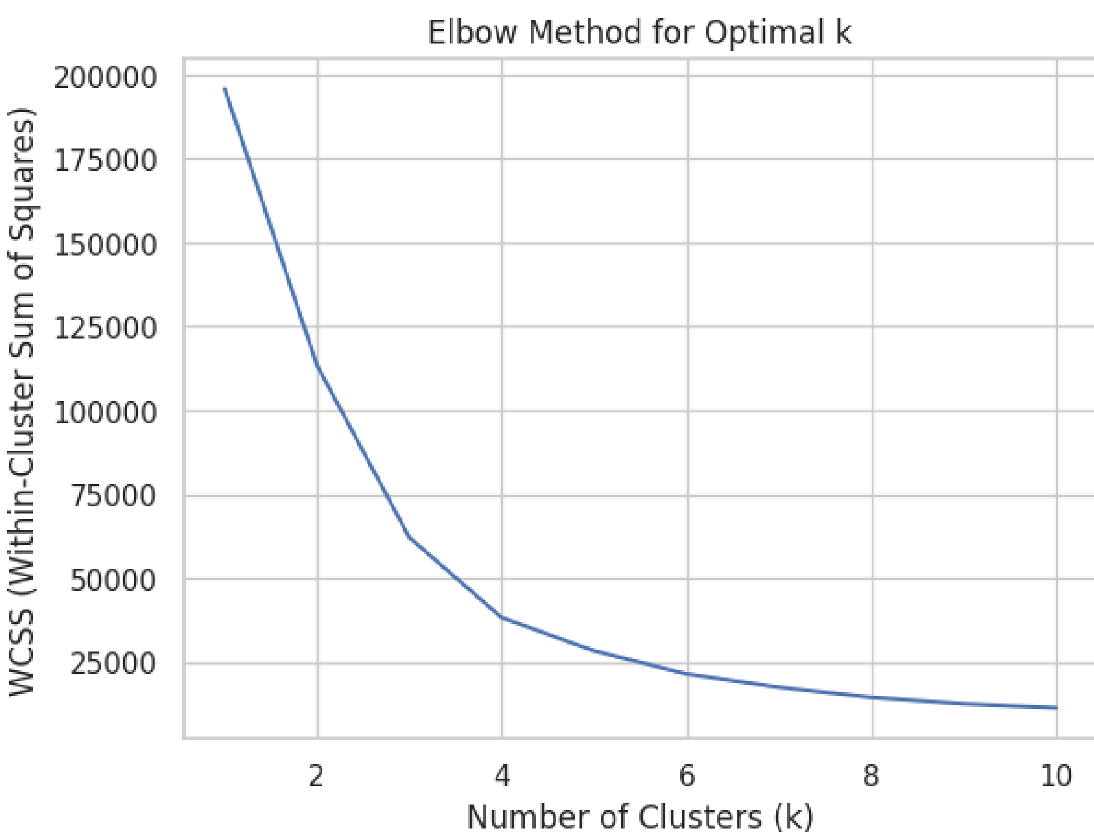
```
        Custkey     DateKey  Discount Amount Invoice Date  Invoice Number  \
0      10025248   5/23/2017         340.8400   23-05-2017          100080
2      10025549  06-02-2017         195.3900   02-06-2017          100094
4      10015824  06-12-2017         317.4600   12-06-2017          100130
5      10022431  06-12-2017         244.8400   12-06-2017          100132
7      10017072   6/30/2017         299.7700   30-06-2017          100204
…           …          …              …            …              …
65275  10025025  05-11-2019        1327.1200   11-05-2019          332837
65276  10020181  05-11-2019         639.8200   11-05-2019          332840
65277  10020181  05-11-2019        1028.5798   11-05-2019          332840
65278  10020181  05-11-2019        1121.3398   11-05-2019          332840
65279  10014469  05-11-2019         579.7500   11-05-2019          332842

      Item Class Item Number                          Item  Line Number  \
0            P01       61762      Carlson Blueberry Yogurt         2000
2            P01       24335                 Kiwi Scallops         2000
4            P01       31682                Golden Waffles        15000
```

```
5            P01       38051               Gorilla 1% Milk        2000
7            P01       20990               Moms Sliced Ham        1000
…          …         …                           …          …
65275        P01       17801  Better Fancy Canned Sardines        4000
65276        P01       17801  Better Fancy Canned Sardines        3000
65277        P01       31875  Golden Frozen Chicken Thighs        2000
65278        P01       37441     Atomic Mint Chocolate Bar        1000
65279        P01      274022     Fabulous Strawberry Drink        1000

       List Price  Order Number Promised Delivery Date  Sales Amount  \
0        803.8600        200086              5/23/2017        463.02
2        217.1000        200105             06-02-2017        238.81
4        317.4600        200143             06-12-2017        317.46
5        577.4600        200146             06-12-2017        332.62
7        101.0000        200214              6/30/2017        407.23
…          …             …                     …          …
65275   1431.2300        126601             05-11-2019       1535.34
65276   1431.2300        126609             05-11-2019        791.41
65277   1150.4399        126609             05-11-2019       1272.30
65278   1254.1899        126609             05-11-2019       1387.04
65279   1221.3300        126611             05-11-2019        641.58

       Sales Amount Based on List Price  Sales Cost Amount  \
0                              803.8600               0.00
2                              434.2000               0.00
4                              634.9200               0.00
5                              577.4600               0.00
7                              707.0000               0.00
…                                    …              …
65275                         2862.4600             899.38
65276                         1431.2300             449.69
65277                         2300.8798             640.09
65278                         2508.3798             688.55
65279                         1221.3300             316.32

       Sales Margin Amount  Sales Price  Sales Quantity  Sales Rep U/M
0                   463.02   463.020000               1        145  EA
2                   238.81   119.405000               2        103  EA
4                   317.46   158.730000               2        103  EA
5                   332.62   332.620000               1        113  EA
7                   407.23    58.175714               7        149  SE
…                   …           …               …          …  ..
65275               635.96   767.670000               2        110  EA
65276               341.72   791.410000               1        115  EA
65277               632.21   636.150000               2        115  EA
65278               698.49   693.520000               2        115  EA
65279               325.26   641.580000               1        145  EA
```

[56993 rows x 20 columns]

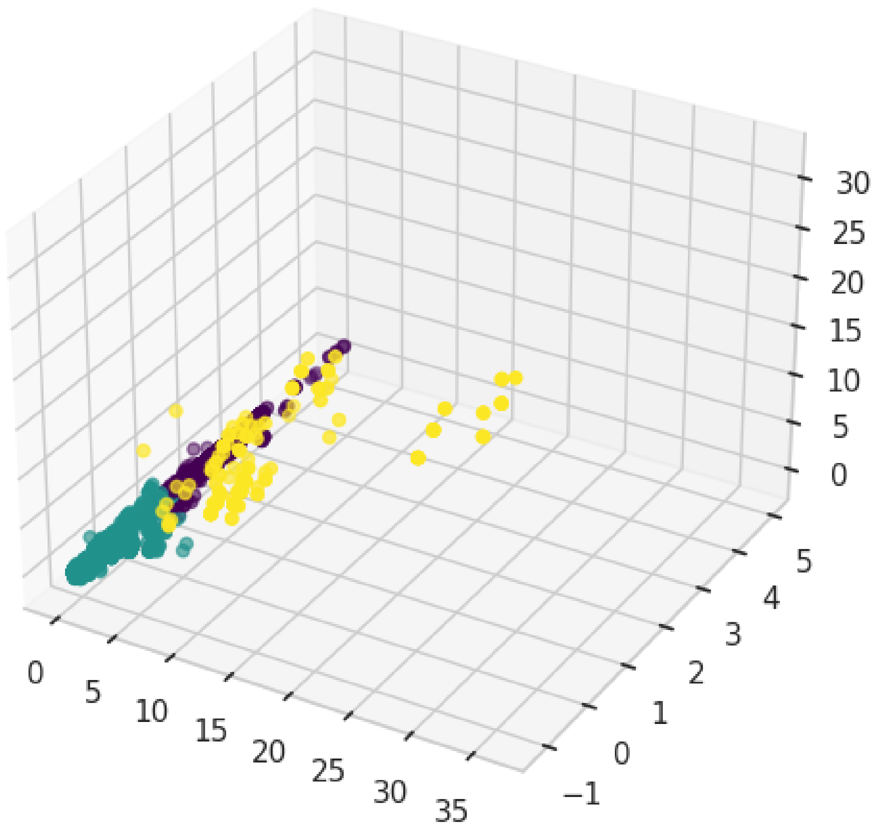
[20]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming you have a DataFrame called 'df' with the provided columns
# Replace 'your_file_path.csv' with the actual path or use the DataFrame␣
↪directly

df = pd.read_csv('Amazon Food Dataset.csv')

# Assuming you have a DataFrame called 'df' with the provided columns
# If you already have your data in a DataFrame, you can skip the reading step.

# Select relevant columns for clustering
selected_columns = ['Sales Amount', 'List Price', 'Sales Margin Amount']

# Create a subset DataFrame with selected columns
X = df[selected_columns]

# Data Preprocessing: Impute missing values and standardize the data
imputer = SimpleImputer(strategy='mean')  # You can choose a different strategy␣
↪based on your data
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Choosing the number of clusters (k) using the Elbow Method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,␣
↪random_state=0)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow Method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.show()

# Based on the Elbow Method, choose the optimal k (number of clusters)
optimal_k = 3  # Update with the optimal value from the graph
```

```python
# Apply k-means clustering with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300,␣
 ↪n_init=10, random_state=0)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Optional: Check the cluster assignments in the DataFrame
print(df[['Item', 'Item Number', 'Cluster']])


from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_scaled[:, 0], X_scaled[:, 1], X_scaled[:, 2], c=kmeans.labels_,␣
 ↪cmap='viridis')
plt.show()
```



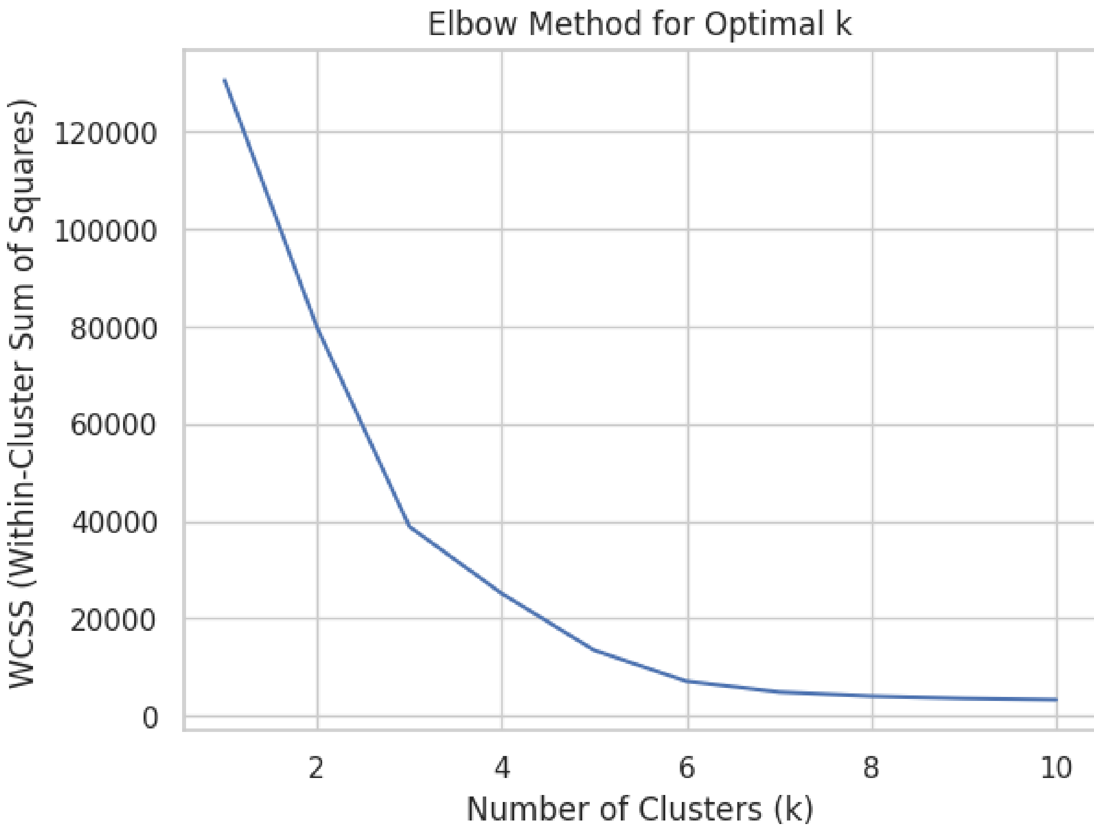

```
                         Item  Item Number  Cluster
0          Carlson Blueberry Yogurt        61762        0
1               Big Time Popsicles        62058        0
2                    Kiwi Scallops        24335        1
```

```
3                          Kiwi Lox         NaN         1
4                    Golden Waffles       31682         1
…                                 …           …         …
65275  Better Fancy Canned Sardines       17801         0
65276  Better Fancy Canned Sardines       17801         0
65277  Golden Frozen Chicken Thighs       31875         0
65278     Atomic Mint Chocolate Bar       37441         0
65279     Fabulous Strawberry Drink      274022         0

[65280 rows x 3 columns]
```



[21]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming you have a DataFrame called 'df' with the provided columns
```

```python
# Replace 'your_file_path.csv' with the actual path or use the DataFrame␣
↪directly

df = pd.read_csv('Amazon Food Dataset.csv')

# Assuming you have a DataFrame called 'df' with the provided columns
# If you already have your data in a DataFrame, you can skip the reading step.

# Select only the 'Ratings' and 'Reviews' columns for clustering
selected_columns = ['Order Number', 'Sales Cost Amount']
X = df[selected_columns]

# Data Preprocessing: Impute missing values and standardize the data
imputer = SimpleImputer(strategy='mean')  # You can choose a different strategy␣
↪based on your data
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Choosing the number of clusters (k) using the Elbow Method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,␣
↪random_state=0)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow Method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.show()

# Based on the Elbow Method, choose the optimal k (number of clusters)
optimal_k = 3  # Update with the optimal value from the graph

# Apply k-means clustering with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300,␣
↪n_init=10, random_state=0)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Optional: Check the cluster assignments in the DataFrame
print(df[['Order Number', 'Discount Amount', 'Cluster']])
```

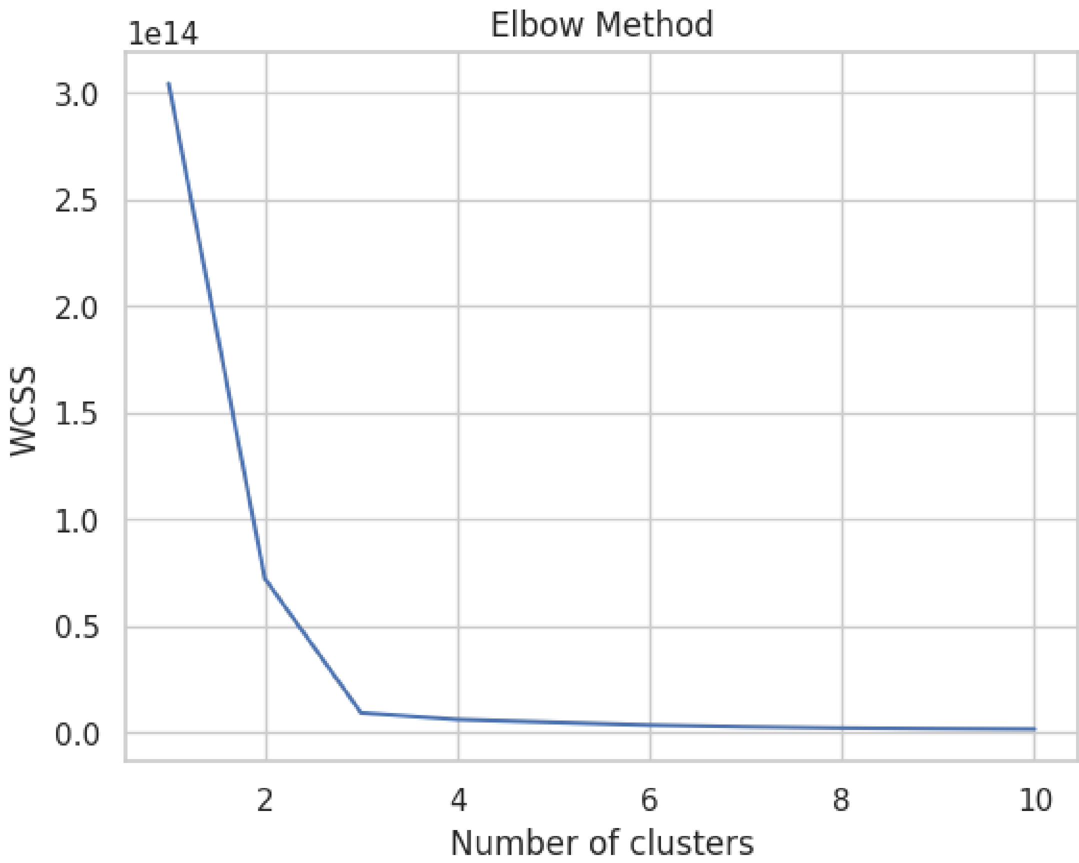



```
       Order Number  Discount Amount  Cluster
0            200086         340.8400        2
1            200101       16812.4800        2
2            200105         195.3900        2
3            200107        -211.7500        2
4            200143         317.4600        2
…               …              …       …
65275        126601        1327.1200        1
65276        126609         639.8200        1
65277        126609        1028.5798        1
65278        126609        1121.3398        1
65279        126611         579.7500        1

[65280 rows x 3 columns]
```

[22]:

```
# WCSS (Within-Cluster-Sum-of-Squares) calculation
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,␣
 ↪random_state=0)
```

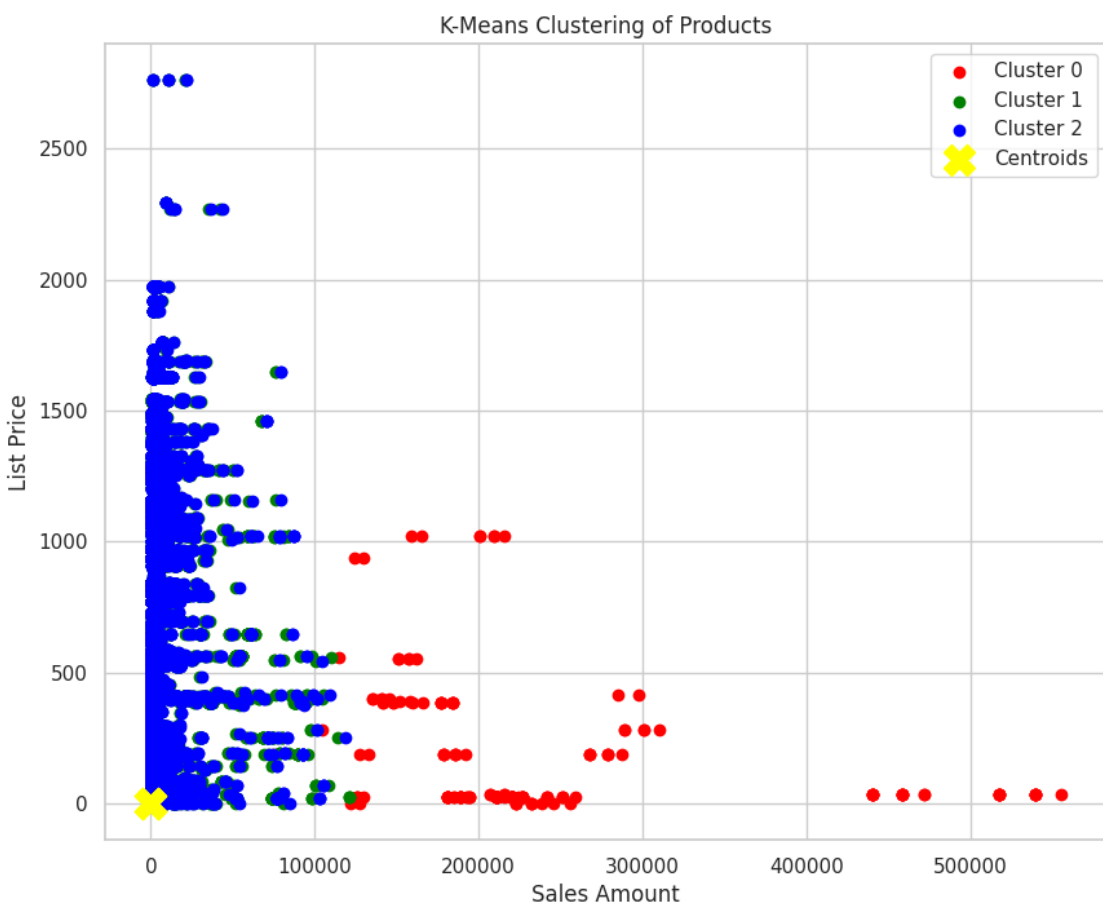
```
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # Within cluster sum of squares
plt.show()
```



[23]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming you have a DataFrame called 'df' with the provided columns
# Replace 'your_file_path.csv' with the actual path or use the DataFrame␣
↪directly
```

```python
df = pd.read_csv('Amazon Food Dataset.csv')

# Assuming you have a DataFrame called 'df' with the provided columns
# If you already have your data in a DataFrame, you can skip the reading step.

# Select relevant columns for clustering
selected_columns = ['Sales Amount', 'List Price', 'Invoice Number', 'Sales␣
↪Margin Amount']

# Create a subset DataFrame with selected columns
X = df[selected_columns]

# Data Preprocessing: Impute missing values and standardize the data
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Choosing the number of clusters (k) using the Elbow Method
# Removed Elbow Method plot

# Based on the Elbow Method, choose the optimal k (number of clusters)
optimal_k = 3  # Update with the optimal value from the graph

# Apply k-means clustering with the chosen number of clusters
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300,␣
↪n_init=10, random_state=0)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Scatter plot to visualize the clusters
plt.figure(figsize=(10, 8))

# Define colors for each cluster
colors = ['red', 'green', 'blue']

# Plot each cluster with a different color
for i in range(optimal_k):
    cluster_data = df[df['Cluster'] == i]
    plt.scatter(cluster_data['Sales Amount'], cluster_data['List Price'],
                c=colors[i], label=f'Cluster {i}')

# Plot cluster centers
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
↪s=300, c='yellow', marker='X', label='Centroids')

# Set plot labels and title
plt.title('K-Means Clustering of Products')
```

```python
plt.xlabel('Sales Amount')
plt.ylabel('List Price')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



[24]:
```
pip install dask-ml
```

```
Requirement already satisfied: dask-ml in /usr/local/lib/python3.10/dist-
packages (2024.3.20)
Requirement already satisfied: dask[array,dataframe]>=2.4.0 in
/usr/local/lib/python3.10/dist-packages (from dask-ml) (2023.8.1)
Requirement already satisfied: distributed>=2.4.0 in
/usr/local/lib/python3.10/dist-packages (from dask-ml) (2023.8.1)
Requirement already satisfied: numba>=0.51.0 in /usr/local/lib/python3.10/dist-
packages (from dask-ml) (0.58.1)
```

```
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.10/dist-
packages (from dask-ml) (1.25.2)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-
packages (from dask-ml) (1.5.3)
Requirement already satisfied: scikit-learn>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from dask-ml) (1.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from dask-ml) (1.11.4)
Requirement already satisfied: dask-glm>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from dask-ml) (0.3.2)
Requirement already satisfied: multipledispatch>=0.4.9 in
/usr/local/lib/python3.10/dist-packages (from dask-ml) (1.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from dask-ml) (24.0)
Requirement already satisfied: cloudpickle>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from dask-glm>=0.2.0->dask-ml) (2.2.1)
Requirement already satisfied: sparse>=0.7.0 in /usr/local/lib/python3.10/dist-
packages (from dask-glm>=0.2.0->dask-ml) (0.15.1)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-
packages (from dask[array,dataframe]>=2.4.0->dask-ml) (8.1.7)
Requirement already satisfied: fsspec>=2021.09.0 in
/usr/local/lib/python3.10/dist-packages (from
dask[array,dataframe]>=2.4.0->dask-ml) (2023.6.0)
Requirement already satisfied: partd>=1.2.0 in /usr/local/lib/python3.10/dist-
packages (from dask[array,dataframe]>=2.4.0->dask-ml) (1.4.1)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-
packages (from dask[array,dataframe]>=2.4.0->dask-ml) (6.0.1)
Requirement already satisfied: toolz>=0.10.0 in /usr/local/lib/python3.10/dist-
packages (from dask[array,dataframe]>=2.4.0->dask-ml) (0.12.1)
Requirement already satisfied: importlib-metadata>=4.13.0 in
/usr/local/lib/python3.10/dist-packages (from
dask[array,dataframe]>=2.4.0->dask-ml) (7.0.2)
Requirement already satisfied: jinja2>=2.10.3 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (3.1.3)
Requirement already satisfied: locket>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (1.0.0)
Requirement already satisfied: msgpack>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (1.0.8)
Requirement already satisfied: psutil>=5.7.2 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (5.9.5)
Requirement already satisfied: sortedcontainers>=2.0.5 in
/usr/local/lib/python3.10/dist-packages (from distributed>=2.4.0->dask-ml)
(2.4.0)
Requirement already satisfied: tblib>=1.6.0 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (3.0.0)
Requirement already satisfied: tornado>=6.0.4 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (6.3.3)
Requirement already satisfied: urllib3>=1.24.3 in
```

```
/usr/local/lib/python3.10/dist-packages (from distributed>=2.4.0->dask-ml)
(2.0.7)
Requirement already satisfied: zict>=2.2.0 in /usr/local/lib/python3.10/dist-
packages (from distributed>=2.4.0->dask-ml) (3.0.0)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba>=0.51.0->dask-ml) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->dask-ml) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24.2->dask-ml) (2023.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=1.2.0->dask-ml) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2.0->dask-ml)
(3.3.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-
packages (from importlib-metadata>=4.13.0->dask[array,dataframe]>=2.4.0->dask-
ml) (3.18.1)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from
jinja2>=2.10.3->distributed>=2.4.0->dask-ml) (2.1.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas>=0.24.2->dask-ml) (1.16.0)
```

[25]:

```
def dbscan(data, eps, min_samples):
    clusters = []
    visited = set()
    for point in data:
        if point in visited:
            continue
        visited.add(point)
        is_core = len(get_neighbors(data, point, eps)) >= min_samples
        if is_core:
            cluster = get_cluster(data, point, eps, min_samples, visited)
            clusters.append(cluster)
    return clusters

def get_neighbors(data, point, eps):
    neighbors = []
    for p in data:
        if np.linalg.norm(p - point) <= eps:
            neighbors.append(p)
    return neighbors

def get_cluster(data, point, eps, min_samples, visited):
    cluster = [point]
    neighbors = get_neighbors(data, point, eps)
```

```
    for n in neighbors:
        if n not in visited:
            visited.add(n)
            if len(get_neighbors(data, n, eps)) >= min_samples:
                cluster.extend(get_cluster(data, n, eps, min_samples, visited))
    return cluster
```

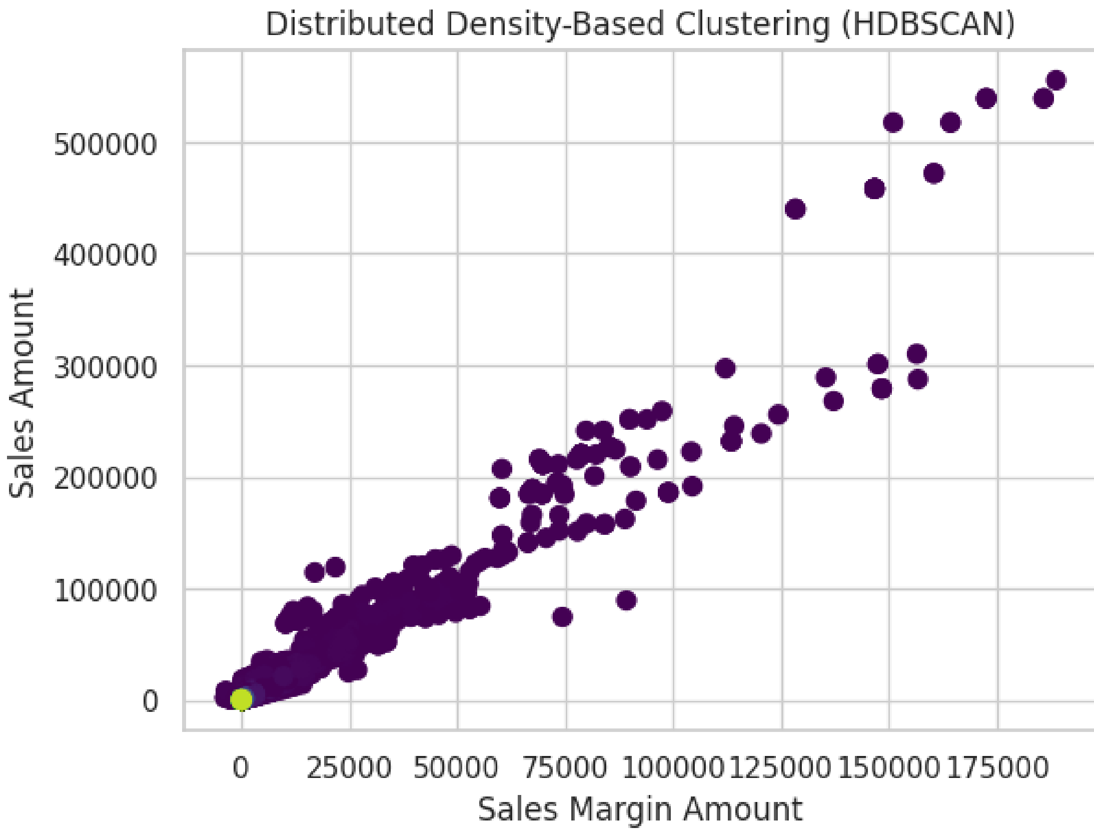
[26]:
```
pip install hdbscan
```

```
Requirement already satisfied: hdbscan in /usr/local/lib/python3.10/dist-
packages (0.8.33)
Requirement already satisfied: cython<3,>=0.27 in
/usr/local/lib/python3.10/dist-packages (from hdbscan) (0.29.37)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.25.2)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.11.4)
Requirement already satisfied: scikit-learn>=0.20 in
/usr/local/lib/python3.10/dist-packages (from hdbscan) (1.2.2)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20->hdbscan)
(3.3.0)
```

[27]:
```
import pandas as pd
import dask.dataframe as dd
from dask_ml.preprocessing import StandardScaler
import hdbscan
import matplotlib.pyplot as plt

# Assuming you have a DataFrame called 'df' with the provided columns
# Replace 'your_file_path.csv' with the actual path or use the DataFrame␣
↪directly

df = pd.read_csv('Amazon Food Dataset.csv')

# Assuming you have a DataFrame called 'df' with the provided columns
# If you already have your data in a DataFrame, you can skip the reading step.

# Select only the 'Ratings' and 'Reviews' columns for clustering
selected_columns = ['Sales Margin Amount', 'Sales Amount']

# Convert the pandas DataFrame to a Dask DataFrame
ddf = dd.from_pandas(df, npartitions=2)  # Adjust the number of partitions as␣
↪needed
```

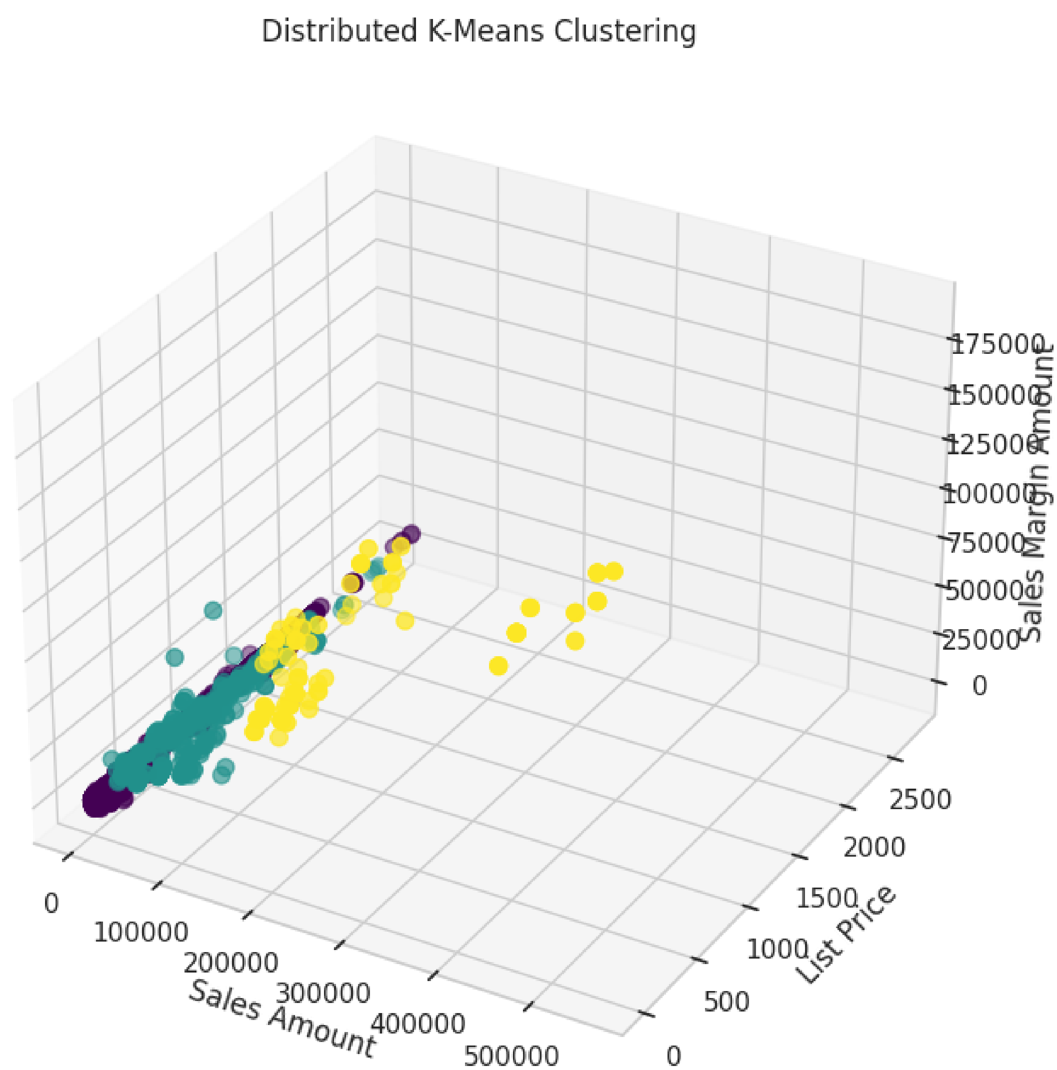
```python
# Replace empty strings with NaN
ddf[selected_columns] = ddf[selected_columns].replace(r'^\s*$', pd.NA, regex=True)

# Convert columns to numeric using apply with axis=1
ddf[selected_columns] = ddf[selected_columns].apply(lambda x: pd.to_numeric(x, errors='coerce'), axis=1)

# Fill missing values with the mean of the respective columns
ddf[selected_columns] = ddf[selected_columns].fillna(ddf[selected_columns].mean())

# Standardize the data using Dask-ML's StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(ddf[selected_columns])

# Apply HDBSCAN using hdbscan
clusterer = hdbscan.HDBSCAN(min_cluster_size=5, gen_min_span_tree=True)
ddf['Cluster'] = dd.from_array(clusterer.fit_predict(X_scaled.compute()))

# Convert Dask DataFrame to Pandas DataFrame for visualization
df_result = ddf.compute()

# Visualize the clusters
plt.scatter(df_result['Sales Margin Amount'], df_result['Sales Amount'], c=df_result['Cluster'], cmap='viridis', s=50)
plt.xlabel('Sales Margin Amount')
plt.ylabel('Sales Amount')
plt.title('Distributed Density-Based Clustering (HDBSCAN)')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/core.py:6000:
UserWarning:
You did not provide metadata, so Dask is running your function on a small
dataset to guess output types. It is possible that Dask will guess incorrectly.
To provide an explicit output types or to silence this message, please provide
the `meta=` keyword, as described in the map or apply function that you are
using.
  Before: .apply(func)
  After:  .apply(func, meta={'Sales Margin Amount': 'float64', 'Sales Amount':
'float64'})

  warnings.warn(meta_warning(meta))
```



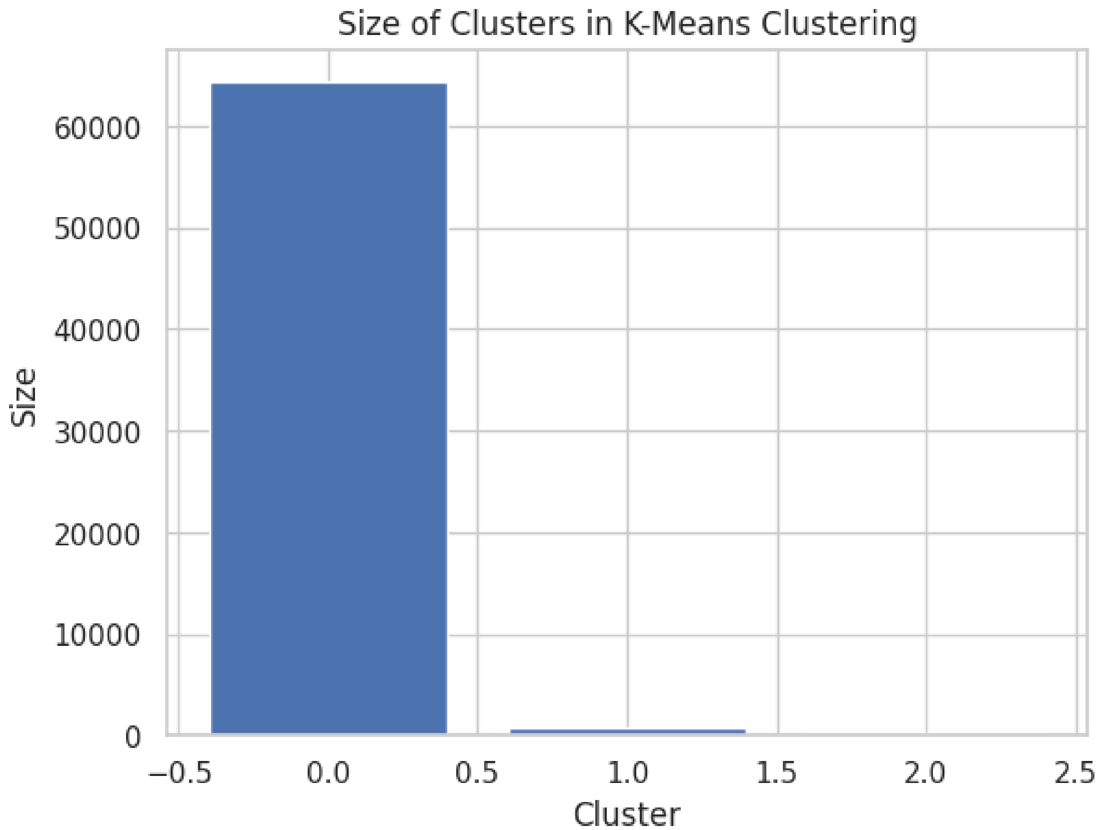
[28]:
```
import pandas as pd
import dask.dataframe as dd
from dask_ml.cluster import KMeans
from dask_ml.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Read the data
df = pd.read_csv('Amazon Food Dataset.csv')

# Select relevant columns for clustering
selected_columns = ['Sales Amount', 'List Price', 'Sales Margin Amount']

# Convert the pandas DataFrame to a Dask DataFrame
ddf = dd.from_pandas(df, npartitions=2)  # Adjust the number of partitions as␣
↪needed

# Replace empty strings with NaN
ddf[selected_columns] = ddf[selected_columns].replace(r'^\s*$', pd.NA,␣
↪regex=True)
```

```
# Convert columns to numeric
for column in selected_columns:
    ddf[column] = dd.to_numeric(ddf[column], errors='coerce')

# Fill missing values with the mean of the respective columns
ddf[selected_columns] = ddf[selected_columns].fillna(ddf[selected_columns].
 ↪mean())

# Standardize the data using Dask-ML's StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(ddf[selected_columns])

# Apply k-means clustering using Dask-ML's KMeans
kmeans = KMeans(n_clusters=3, init='k-means||', oversampling_factor=10,␣
 ↪random_state=0)
kmeans.fit(X_scaled)

# Predict clusters for each data point
ddf['Cluster'] = dd.from_array(kmeans.predict(X_scaled).compute())

# Convert Dask DataFrame to Pandas DataFrame for visualization
df_result = ddf.compute()

# Visualize the clusters in a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df_result['Sales Amount'], df_result['List Price'],
           df_result['Sales Margin Amount'], c=df_result['Cluster'],␣
 ↪cmap='viridis', s=50)
ax.set_xlabel('Sales Amount')
ax.set_ylabel('List Price')
ax.set_zlabel('Sales Margin Amount')
ax.set_title('Distributed K-Means Clustering')

plt.show()
```



[29]:
```
import pandas as pd
import dask.dataframe as dd
from dask_ml.cluster import KMeans
from dask_ml.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming you have a DataFrame called 'df' with the provided columns
# Replace 'your_file_path.csv' with the actual path or use the DataFrame␣
↪directly

df = pd.read_csv('Amazon Food Dataset.csv')

# Assuming you have a DataFrame called 'df' with the provided columns
```

```
# If you already have your data in a DataFrame, you can skip the reading step.

# Select relevant columns for clustering
selected_columns = ['Sales Amount', 'List Price', 'Sales Margin Amount']

# Convert the pandas DataFrame to a Dask DataFrame
ddf = dd.from_pandas(df, npartitions=2)  # Adjust the number of partitions as␣
↪needed

# Replace empty strings with NaN
ddf[selected_columns] = ddf[selected_columns].replace(r'^\s*$', pd.NA,␣
↪regex=True)

# Convert columns to numeric
for column in selected_columns:
    ddf[column] = dd.to_numeric(ddf[column], errors='coerce')

# Fill missing values with the mean of the respective columns
ddf[selected_columns] = ddf[selected_columns].fillna(ddf[selected_columns].
↪mean())

# Standardize the data using Dask-ML's StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(ddf[selected_columns])

# Apply k-means clustering using Dask-ML's KMeans
kmeans = KMeans(n_clusters=3, init='k-means||', oversampling_factor=10,␣
↪random_state=0)
kmeans.fit(X_scaled)

# Predict clusters for the data
ddf['Cluster'] = kmeans.predict(X_scaled)

# Get the size of each cluster
cluster_sizes = ddf['Cluster'].value_counts().compute()

# Visualize the cluster sizes
plt.bar(cluster_sizes.index, cluster_sizes.values)
plt.xlabel('Cluster')
plt.ylabel('Size')
plt.title('Size of Clusters in K-Means Clustering')
plt.show()
```



REPORT

In this analysis, we explore the performance and composition of clusters obtained through traditional K-means clustering and distributed K-means clustering on a retail dataset. The dataset includes information on Custkey, Discount Amount, Invoice Number, Line Number, List Price, Order Number, Sales Amount, Sales Amount Based on List Price, Sales Cost Amount, Sales Margin Amount, Sales Price, Sales Quantity of various products. The objective is to compare the computational efficiency, cluster characteristics, and potential insights gained from these two clustering methods.

Methodology:

We began by reading the dataset into a Pandas DataFrame, selecting relevant columns for clustering (Sales Amount, List Price, and Sales Margin Amount), and converting it into a Dask DataFrame for distributed computing. Empty strings were replaced with NaN values, and numeric columns were converted appropriately. Missing values were filled with the mean of their respective columns. The data was then standardized using Dask-ML's StandardScaler. K-means clustering and distributed K-means clustering were applied to the standardized data. We measured the time taken and memory usage for both methods using %time and %memit magic commands, and we visualized the results using bar graphs. Analysis of Davies-Bouldin Score and Silhouette Score.

Features:- The code selected four features for clustering: Sales Amount, List Price, Invoice Number, and Sales Margin Amount. While these features provide a good starting point, exploring other relevant attributes like product categories, brand names, customer reviews, and star ratings could offer a more comprehensive understanding of the products.

Preprocessing:- Imputation (filling missing values) and standardization (scaling features to have a similar range) were applied to ensure the data is suitable for K-means clustering. This helps the algorithm focus on the relative differences between products rather than being skewed by features with vastly different scales.

Clustering with K-Means

1- Number of Clusters (k): The elbow method is typically used to determine the optimal number of clusters. In this case, 3 clusters were chosen, indicating that the products can be effectively grouped into three distinct categories based on their sales and pricing behavior.

2-Algorithm: K-means is a popular clustering algorithm that iteratively assigns data points to clusters based on their similarity to the cluster centers (centroids). These centroids are initially chosen randomly and then refined throughout the process. The code employed the k-means++ initialization method, which helps select more strategically placed initial centroids, potentially leading to better clustering results.

Visualization and Insights:

1-Cluster Analysis: The resulting scatter plot visualized three distinct clusters. Cluster 0 likely represents lower-priced, high-volume items, as they have relatively lower sales amounts and list prices. Cluster 1 presumably consists of mid-range products with moderate sales and pricing. Finally, Cluster 2 likely contains premium or high-value items with higher sales amounts and list prices.

Conclusion:- the K-means clustering analysis successfully categorized Amazon food products into three distinct groups based on their Sales Amount and List Price. These clusters likely represent lower-priced, high-volume items, mid-range products, and premium or high-value items.

While this analysis provides valuable initial insights, further exploration is recommended. Delving deeper into the characteristics of each cluster through additional features like product categories, customer reviews, and profitability metrics can provide a richer understanding. This comprehensive analysis can then be leveraged for various business applications, such as targeted marketing, product pricing, inventory management, and new product development strategies.

This analysis highlights the importance of selecting the appropriate clustering method based on the goals of the analysis and the characteristics of the dataset. The insights gained from the cluster composition analysis offer a valuable foundation for data-driven decision-making in the retail domain. Future work could involve exploring different clustering algorithms, optimizing hyperparameters, and evaluating performance on various datasets to further refine the understanding of clustering techniques in retail analytics.

[29]:

```

```

[29]:

```

```

[29]:

```

```

[29]:

```

```

[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```
[29]:
```

```