datathon

May 18, 2024

[ ]:
```
import pandas as pd

source = pd.read_csv('./UrbanEdgeApparel.csv')
```

1 Calculate Individual Metrics

1.1 Total Sales Volumn of Each Product

[ ]:
```
df = source.copy()

sales_volume = df.groupby('Product ID')['Total Selling Price'].sum()

# Normalize sales volume
sales_volume_min = sales_volume.min()
sales_volume_max = sales_volume.max()

sales_volume_normalized = (sales_volume - sales_volume_min) / (sales_volume_max␣
 ↪- sales_volume_min)

print(sales_volume_normalized.sort_values(ascending=False))
```

```
Product ID
Prod_1009    1.000000
Prod_1140    0.915352
Prod_2015    0.800323
Prod_1000    0.467872
Prod_2024    0.466712
               …
Prod_131     0.000000
Prod_144     0.000000
Prod_143     0.000000
Prod_113     0.000000
Prod_100     0.000000
Name: Total Selling Price, Length: 302, dtype: float64
```

1.2 Sales Growth Rate of Each Product

[ ]:
```
df = source.copy()

# Calculate sales growth rate
df['Order Data'] = pd.to_datetime(df['Order Date'])

df['YearMonth'] = df['Order Data'].dt.to_period('M')

monthly_sales = df.groupby(['Product ID', 'YearMonth'])['Total Selling Price'].
 ↪sum().reset_index()

monthly_sales['SGR'] = monthly_sales.groupby('Product ID')['Total Selling␣
 ↪Price'].pct_change()

monthly_sales = monthly_sales.dropna()

# Calculate the average sales growth rate
average_sgr = monthly_sales.groupby('Product ID')['SGR'].mean().reset_index()
average_sgr.columns = ['Product ID', 'Average SGR']

# Normalize the average sales growth rate
average_sgr_min = average_sgr['Average SGR'].min()
average_sgr_max = average_sgr['Average SGR'].max()
average_sgr['Normalized Average SGR'] = (average_sgr['Average SGR'] -␣
 ↪average_sgr_min) / (average_sgr_max - average_sgr_min)

print(average_sgr.sort_values('Normalized Average SGR', ascending=False))
```

```
     Product ID  Average SGR  Normalized Average SGR
159  Prod_53518    15.133270                1.000000
204   Prod_7001    10.099871                0.678033
211   Prod_7350     8.572111                0.580308
75    Prod_2070     6.787932                0.466181
194   Prod_6207     5.990834                0.415194
..          …           …                       …
61   Prod_20063    -0.350000                0.009595
54   Prod_14010    -0.357593                0.009109
163  Prod_54026    -0.422222                0.004975
45   Prod_13000    -0.500000                0.000000
30    Prod_1093    -0.500000                0.000000

[282 rows x 3 columns]
```

1.3 Quantity Sold of Each Product

[ ]:
```
df = source.copy()

df['Order Date'] = pd.to_datetime(df['Order Date'])

df['YearMonth'] = df['Order Date'].dt.to_period('M')

monthly_quantity = df.groupby(['Product ID', 'YearMonth'])['Product Quantity'].
 ↪sum().reset_index()
monthly_quantity.columns = ['Product ID', 'YearMonth', 'Quantity Sold']

monthly_quantity['QSI'] = monthly_quantity.groupby('Product ID')['Quantity␣
 ↪Sold'].transform(lambda x: (x - x.min()) / (x.max() - x.min()))

monthly_quantity = monthly_quantity.dropna()

print(monthly_quantity.head())

average_qsi = monthly_quantity.groupby('Product ID')['QSI'].mean().reset_index()
average_qsi.columns = ['Product ID', 'Average QSI']

print(average_qsi.head())
```

```
  Product ID YearMonth  Quantity Sold       QSI
1  Prod_1000   2013-04            135  0.056943
2  Prod_1000   2013-05            521  0.226018
3  Prod_1000   2013-06            136  0.057381
4  Prod_1000   2013-07            128  0.053876
5  Prod_1000   2013-08            116  0.048620
   Product ID  Average QSI
0   Prod_1000     0.234903
1  Prod_10021     0.370526
2   Prod_1003     0.287332
3   Prod_1005     0.168096
4   Prod_1007     0.149496
```

1.4 Average Selling Price of Each Product

[ ]:
```
df = source.copy()

product_sales = df.groupby('Product ID')['Total Selling Price'].sum().
 ↪reset_index()
product_quantity = df.groupby('Product ID')['Product Quantity'].sum().
 ↪reset_index()

product_sales = product_sales.merge(product_quantity, on='Product ID')
```

```
product_sales = product_sales[product_sales['Product Quantity'] > 0]

product_sales['Average Selling Price'] = product_sales['Total Selling Price'] /␣
 ↪product_sales['Product Quantity']

# Normalize the average selling price
min_asp = product_sales['Average Selling Price'].min()
max_asp = product_sales['Average Selling Price'].max()
product_sales['Normalized Average Selling Price'] = (product_sales['Average␣
 ↪Selling Price'] - min_asp) / (max_asp - min_asp)

product_sales = product_sales[['Product ID', 'Average Selling Price',␣
 ↪'Normalized Average Selling Price']]

print(product_sales.sort_values('Normalized Average Selling Price',␣
 ↪ascending=False))
```

```
     Product ID  Average Selling Price  Normalized Average Selling Price
20    Prod_1028             125.000000                          1.000000
172  Prod_53318              50.000000                          0.400000
156  Prod_52518              43.884615                          0.351077
173  Prod_53518              33.666102                          0.269329
171  Prod_53118              33.000000                          0.264000
..          …                     …                                 …
24     Prod_105               0.000000                          0.000000
65     Prod_144               0.000000                          0.000000
34     Prod_113               0.000000                          0.000000
50     Prod_131               0.000000                          0.000000
0      Prod_100               0.000000                          0.000000

[302 rows x 3 columns]
```

1.5 Shipping States of Each Product

[ ]:
```
df = source.copy()

unique_states = df.groupby('Product ID')['Shipping State'].nunique().
 ↪reset_index()

unique_states.columns = ['Product ID', 'Number of Unique States']

# Normalize the number of unique states
min_states = unique_states['Number of Unique States'].min()
max_states = unique_states['Number of Unique States'].max()
unique_states['Normalized Unique States'] = (unique_states['Number of Unique␣
 ↪States'] - min_states) / (max_states - min_states)
```

```
print(unique_states.sort_values('Normalized Unique States', ascending=False))
```

```
     Product ID  Number of Unique States  Normalized Unique States
6     Prod_1009                       64                  1.000000
132   Prod_4402                       55                  0.857143
1     Prod_1000                       54                  0.841270
4     Prod_1005                       54                  0.841270
35    Prod_1140                       54                  0.841270
..          …                        …                         …
242   Prod_7800                        1                  0.000000
49   Prod_13000                        1                  0.000000
50     Prod_131                        1                  0.000000
52     Prod_132                        1                  0.000000
0      Prod_100                        1                  0.000000

[302 rows x 3 columns]
```

1.6 Shipping Countries of Each Product

[ ]:
```
df = source.copy()

unique_countries = df.groupby('Product ID')['Shipping Country'].nunique().
 ↪reset_index()

unique_countries.columns = ['Product ID', 'Number of Unique Countries']

# Normalize the number of unique countries
min_countries = unique_countries['Number of Unique Countries'].min()
max_countries = unique_countries['Number of Unique Countries'].max()
unique_countries['Normalized Unique Countries'] = (unique_countries['Number of␣
 ↪Unique Countries'] - min_countries) / (max_countries - min_countries)

print(unique_countries.sort_values('Normalized Unique Countries',␣
 ↪ascending=False))
```

```
     Product ID  Number of Unique Countries  Normalized Unique Countries
6     Prod_1009                          16                     1.000000
35    Prod_1140                          12                     0.733333
4     Prod_1005                          12                     0.733333
132   Prod_4402                          12                     0.733333
1     Prod_1000                          10                     0.600000
..          …                           …                            …
135  Prod_48016                           1                     0.000000
147   Prod_5040                           1                     0.000000
156  Prod_52518                           1                     0.000000
157   Prod_5300                           1                     0.000000
```

```
301  Prod_99300                          1                    0.000000

[302 rows x 3 columns]
```

1.7 Cancalllation Rate of Each Product

[ ]:
```
df = source.copy()

total_orders = df.groupby('Product ID')['Order ID'].count().reset_index()
total_orders.columns = ['Product ID', 'Total Orders']

completed_orders = df[df['Order Status'] == 'Completed']

completed_order_counts = completed_orders.groupby('Product ID')['Order ID'].
 ↪count().reset_index()
completed_order_counts.columns = ['Product ID', 'Completed Orders']

order_stats = total_orders.merge(completed_order_counts, on='Product ID',␣
 ↪how='left')
order_stats['Completed Orders'] = order_stats['Completed Orders'].fillna(0)  #␣
 ↪Fill NaN values with 0

order_stats['Completion Rate'] = order_stats['Completed Orders'] /␣
 ↪order_stats['Total Orders']

print(order_stats.sort_values('Completion Rate', ascending=False))
```

```
     Product ID  Total Orders  Completed Orders  Completion Rate
0      Prod_100             1                 1         1.000000
72    Prod_2010            94                94         1.000000
114   Prod_3507            43                43         1.000000
233  Prod_74003            19                19         1.000000
119   Prod_3516             3                 3         1.000000
..          …             …                 …              …
98   Prod_30200             8                 6         0.750000
234  Prod_74010             6                 4         0.666667
96    Prod_3013            36                22         0.611111
173  Prod_53518            19                11         0.578947
32    Prod_1093             6                 3         0.500000

[302 rows x 4 columns]
```

1.8 Number of Companies bought each product

[ ]:
```
df = source.copy()
```

```
unique_companies = df.groupby('Product ID')['Company ID'].nunique().
 ↪reset_index()

unique_companies.columns = ['Product ID', 'Number of Unique Companies']

min_companies = unique_companies['Number of Unique Companies'].min()
max_companies = unique_companies['Number of Unique Companies'].max()
unique_companies['Normalized Unique Companies'] = (unique_companies['Number of␣
 ↪Unique Companies'] - min_companies) / (max_companies - min_companies)

print(unique_companies.sort_values('Normalized Unique Companies',␣
 ↪ascending=False))
```

```
     Product ID  Number of Unique Companies  Normalized Unique Companies
6     Prod_1009                         472                     1.000000
35    Prod_1140                         345                     0.730932
1     Prod_1000                         334                     0.707627
246   Prod_7900                         286                     0.605932
132   Prod_4402                         276                     0.584746
..          …                           …                          …
45    Prod_1171                           1                     0.002119
34     Prod_113                           1                     0.002119
273  Prod_86912                          1                     0.002119
0      Prod_100                          1                     0.002119
46   Prod_11800                          0                     0.000000

[302 rows x 3 columns]
```

2 Combine Metrics to Calculate Overall Product Performance

2.1 Weights for Each Metric

[ ]:
```
Weights = {
    'Sales Volume': 0.25,
    'Average Sales Growth': 0.2,
    'Average Quantity Sold': 0.1,
    'Average Selling Price': 0.1,
    'Number of Unique States Sold': 0.1,
    'Number of Unique Countries Sold': 0.1,
    'Number of Unique Companies Sold': 0.05,
    'Completion Rate': 0.1
}
```

- Sales Volume (0.25):
  - High Impact: Directly correlates with revenue and market demand. Products with higher sales volumes generate more revenue and indicate strong market presence.
- Average Sales Growth Rate (0.2):

  - Trend Indicator: Reflects how the product is performing over time. Steady growth indicates increasing market acceptance and potential for future success.
- Average Quantity Sold (0.10):
  - Demand Stability: Indicates consistent demand for the product, which is crucial for ongoing sales.
- Average Selling Price (0.10):
  - Profitability: Higher selling prices can lead to greater profitability per unit sold, making it an important metric.
- Number of Unique States Sold (0.10):
  - Geographic Reach: Shows market penetration within a country, indicating broader acceptance and distribution capabilities.
- Number of Unique Countries Sold (0.10):
  - International Reach: Similar to states, but indicates global market penetration and acceptance, which is vital for products aiming for international markets.
- Number of Unique Companies Sold (0.05):
  - Market Versatility: Selling to a diverse range of companies can indicate product versatility and broader market appeal, but less critical than other metrics.
- Cancellation Rate (0.10):
  - Reliability and Satisfaction: High completion rates are crucial for maintaining customer trust and ensuring repeat business.

2.2 Build the performance matrix

[ ]:
```
performance = pd.DataFrame(source['Product ID'].unique(), columns=['Product␣
↪ID'])

# 1. Sales Volume
performance = performance.merge(sales_volume_normalized, on='Product ID',␣
↪how='left').fillna(0)

# 2. Average Sales Growth
performance = performance.merge(average_sgr[['Product ID', 'Normalized Average␣
↪SGR']], on='Product ID', how='left').fillna(0)

# 3. Average Quantity Sold
performance = performance.merge(average_qsi, on='Product ID', how='left').
↪fillna(0)

# 4. Average Selling Price
performance = performance.merge(product_sales[['Product ID', 'Normalized␣
↪Average Selling Price']], on='Product ID', how='left').fillna(0)

# 5. Number of Unique States Sold
performance = performance.merge(unique_states[['Product ID', 'Normalized Unique␣
↪States']], on='Product ID', how='left').fillna(0)

# 6. Number of Unique Countries Sold
```

```
performance = performance.merge(unique_countries[['Product ID', 'Normalized␣
 ↪Unique Countries']], on='Product ID', how='left').fillna(0)

# 7. Number of Unique Companies Sold
performance = performance.merge(unique_companies[['Product ID', 'Normalized␣
 ↪Unique Companies']], on='Product ID', how='left').fillna(0)

# 8. Completion Rate
performance = performance.merge(order_stats[['Product ID', 'Completion Rate']],␣
 ↪on='Product ID', how='left').fillna(0)

print(performance.head())
```

```
   Product ID  Total Selling Price  Normalized Average SGR  Average QSI  \
0   Prod_5030             0.010459                0.140325     0.058856
1  Prod_70018             0.001103                0.062569     0.099662
2   Prod_1000             0.467872                0.054752     0.234903
3   Prod_5080             0.043329                0.123600     0.117056
4   Prod_5002             0.053146                0.071799     0.239942

   Normalized Average Selling Price  Normalized Unique States  \
0                          0.030215                  0.317460
1                          0.015461                  0.206349
2                          0.034900                  0.841270
3                          0.023658                  0.698413
4                          0.021658                  0.555556

   Normalized Unique Countries  Normalized Unique Companies  Completion Rate
0                     0.133333                     0.078390         0.992701
1                     0.066667                     0.048729         1.000000
2                     0.600000                     0.707627         0.980116
3                     0.133333                     0.237288         0.984305
4                     0.400000                     0.296610         0.987782
```

2.3 Calculate the overall performance score for each product

[ ]:
```
performance['Performance Score'] = (
  performance['Total Selling Price'] * Weights['Sales Volume'] +
  performance['Normalized Average SGR'] * Weights['Average Sales Growth'] +
  performance['Average QSI'] * Weights['Average Quantity Sold'] +
  performance['Normalized Average Selling Price'] * Weights['Average Selling␣
 ↪Price'] +
  performance['Normalized Unique States'] * Weights['Number of Unique States␣
 ↪Sold'] +
  performance['Normalized Unique Countries'] * Weights['Number of Unique␣
 ↪Countries Sold'] +
```

```
  performance['Normalized Unique Companies'] * Weights['Number of Unique␣
  ↪Companies Sold'] +
  performance['Completion Rate'] * Weights['Completion Rate']
)

print(performance['Performance Score'].sort_values(ascending=False))
```

```
9      0.636478
25     0.561396
39     0.460047
2      0.432419
47     0.400785
         …
178    0.100908
233    0.100106
251    0.100106
252    0.100106
211    0.100106
Name: Performance Score, Length: 302, dtype: float64
```