cct College Dublin
Computing • IT • Business

Assessment Cover Page

Student Full Name	CHAN CHEE XIANG
Student Number	2024463
Module Title	INTERGRADE ML & VIS CA2
Assessment Title	INTERGRADE ML & VIS CA2
Assessment Due Date	2025-05-21
Date of Submission	2025-05-21

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on academic misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source.

I declare it to be my own work and that all material from third parties has been appropriately referenced.

I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

Introduction

This report explains the Integrated ML & VIS CA2 Jupyter Notebook, which helps businesses make better decisions through data analysis. The notebook has six main parts that work together. First, it handles data collection and cleaning to prepare the information. Second, it does exploratory data analysis (EDA) to find important patterns and trends. Third, feature engineering improves the data for better results. Fourth, market basket analysis shows which products customers often buy together. Fifth, the recommendation system suggests products customers might like. Finally, create a dashboard visualizes all these findings with interactive charts, maps and table to make information easy to understand.

Each part helps in different ways from preparing data to showing final results until giving businesses complete insights about customer behaviour, popular products, and sales patterns. This helps shops make better decision about what to sell, how to arrange products, and how to recommend items to customers.

Data Understanding

Data contain total 21 features, 2 feature are datetime64[ns], 3 feature are float64, 3 features are int64, 13 feature are object. Total file size 1.6 MB, 9994 entries and no missing values found.

```
In [3]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
```

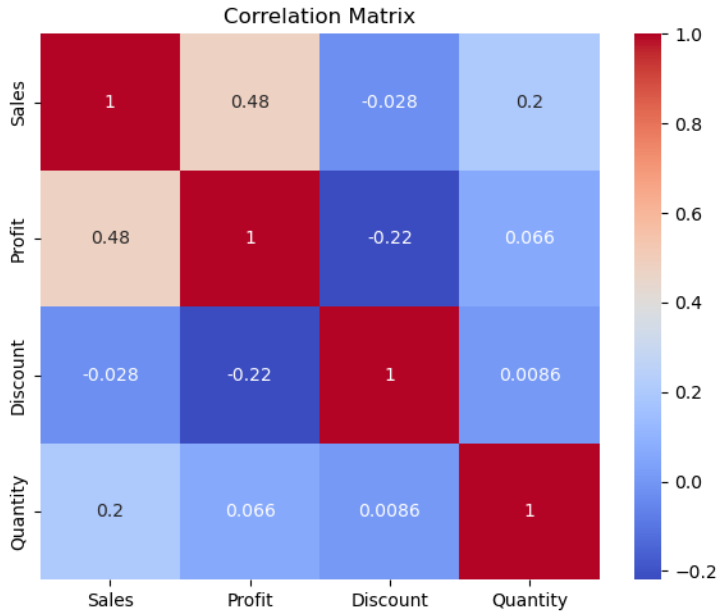
Data Preprocessing and Feature Engineering

```
In [49]: # New temporal features
         df['Order Year']    = df['Order Date'].dt.year
         df['Order Month']   = df['Order Date'].dt.month
         df['Order Weekday'] = df['Order Date'].dt.day_name()
```

```
In [50]: print(df.dtypes)
```

```
Row ID                  int64
Order ID               object
Order Date     datetime64[ns]
Ship Date      datetime64[ns]
Ship Mode              object
Customer ID            object
Customer Name          object
Segment                object
Country                object
City                   object
State                  object
Postal Code             int64
Region                 object
Product ID             object
Category               object
Sub-Category           object
Product Name           object
Sales                 float64
Quantity                int64
Discount              float64
Profit                float64
Order Year              int32
Order Month             int32
Order Weekday          object
dtype: object
```

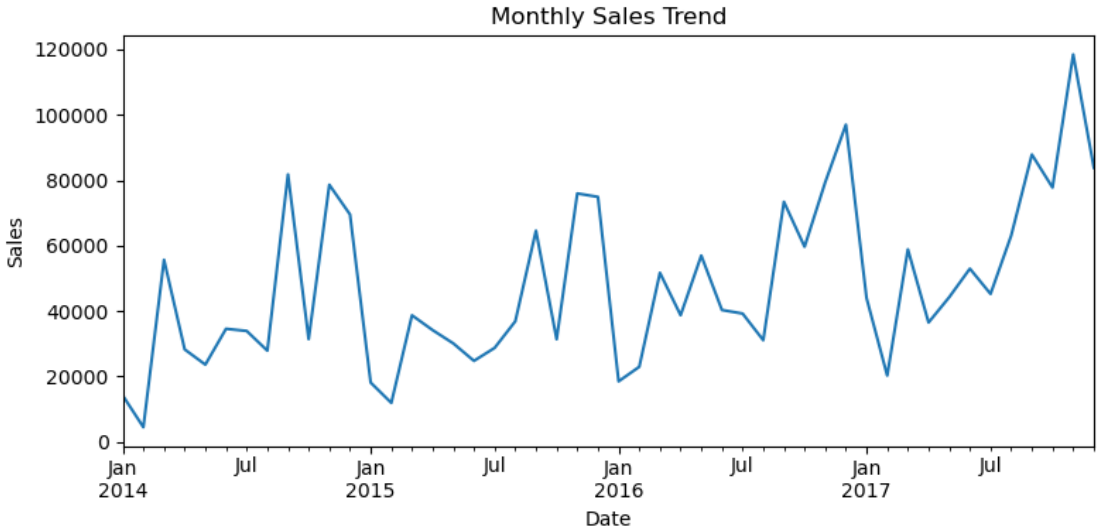
To extract Order Date into " Order Year, Order Month , and Order Weekday" as called temporal feature for visualization convenient purposes



Above graphs shows the correlation heatmap to understand the relation score between the feature, higher sales normally lead higher profits, but in the heatmap shows it doesn't perfectly align, it meaning that other costs may dilute in the profit margins.

Discount and profit have a weak correlation (-0.22), it seems discount were reducing the company profit, it looks like possible lower margins on discounted items.

Overall Trend



Above graph shows the overall sales trend by month for the company, it shows sales are fluctuated, the chart shows significant peaks and valleys, indicating seasonal or periodic variations in sales. Overall shows to be a sales were upward over the years compare from 2014 to 2017.

Market Basket Analysis – Apriori Rule

Step 1 - Create a basket to aggregate the Order ID and Sub-Category into a list

```
In [18]: # group by Order ID and aggregate Product into a list
         basket = df.groupby("Order ID")["Sub-Category"].apply(list)
         basket.head(10)
```

```
Out[18]: Order ID
         CA-2014-100006                                  [Phones]
         CA-2014-100090                          [Tables, Binders]
         CA-2014-100293                                   [Paper]
         CA-2014-100328                                 [Binders]
         CA-2014-100363                        [Fasteners, Paper]
         CA-2014-100391                                   [Paper]
         CA-2014-100678    [Art, Chairs, Envelopes, Accessories]
         CA-2014-100706                [Accessories, Furnishings]
         CA-2014-100762              [Art, Labels, Paper, Paper]
         CA-2014-100860                                  [Labels]
         Name: Sub-Category, dtype: object
```

Above graphs shows the coding for grouping Order ID and Sub-Category into a list, this step apply is for understand which products are frequently bought together and it also called association rules, example "If customers buy X item, they're possibility will buy Y item." In the business insight it possibility could give understanding which item are selling the most and given promotion to the customer.

Step 2 - Transaction Encoding Process

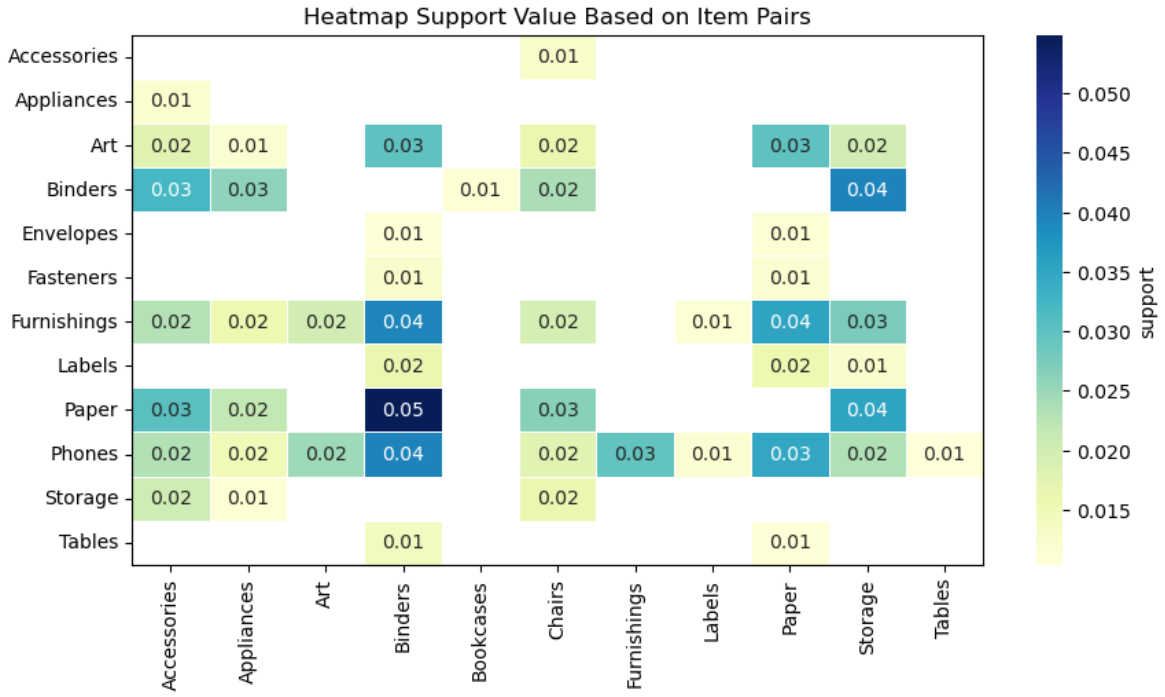
Convert the code a list of purchased items into a binary matrix format, which is essential for association rule mining

```
In [19]: # Create an object te by calling a method TransactionEncoder()
te = TransactionEncoder()

# Call fit() method to train the model
te_array = te.fit(basket).transform(basket)

# Transform te_array into dataframe
basket_df = pd.DataFrame(te_array, columns = te.columns_)
basket_df
```

Out[19]:

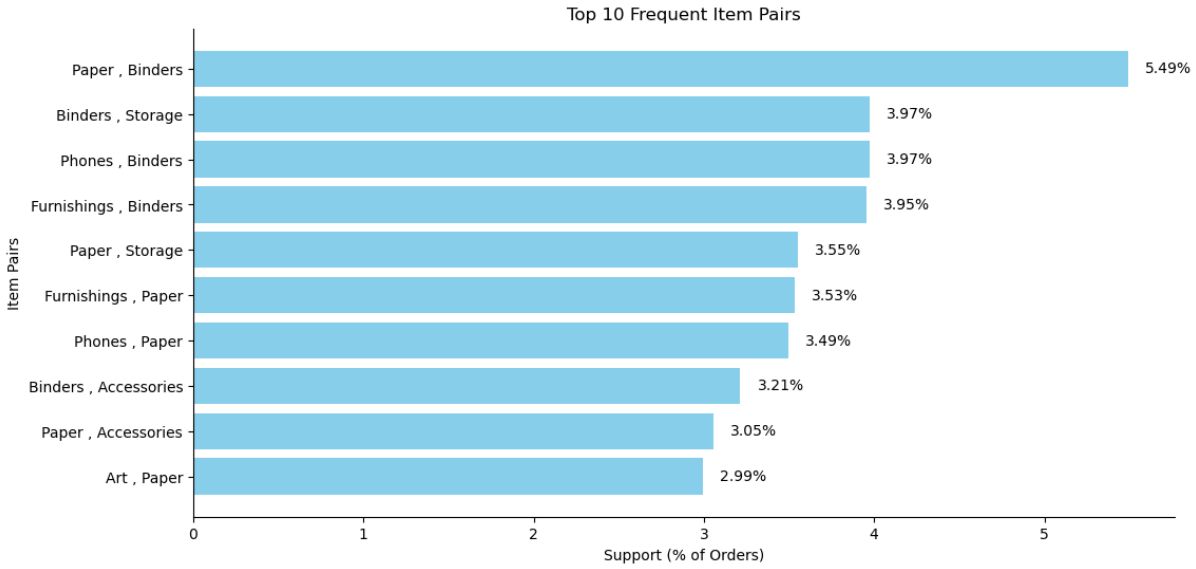
	Accessories	Appliances	Art	Binders	Bookcases	Chairs	Copiers	Envelopes	Fast
0	False	False	False	False	False	False	False	False	
1	False	False	False	True	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	True	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	

Step 3 - Find the frequent pair set



The Heatmap shows the support value that two item categories are purchased together. As per review that, "Paper and Binders" has the highest support value in the heatmap (0.05). It means that, customers frequent purchase Paper and Binders together. Besides, "Storage & Binders" and "Paper & Storage"are also show relatively in high support (0.04), it means "Storage" is often buy with "Binders and Paper" too.

Some categories have very low support with others. This means are either niche purchases or they are less often buy together with other items.



Above graph shows Paper and Binders dominate as a high-demand products, it frequent pair with Storage, Phones, or Accessories. Other top pairs like Paper & Binders has approx. 3.97% support it reflects consistent customer behaviour.

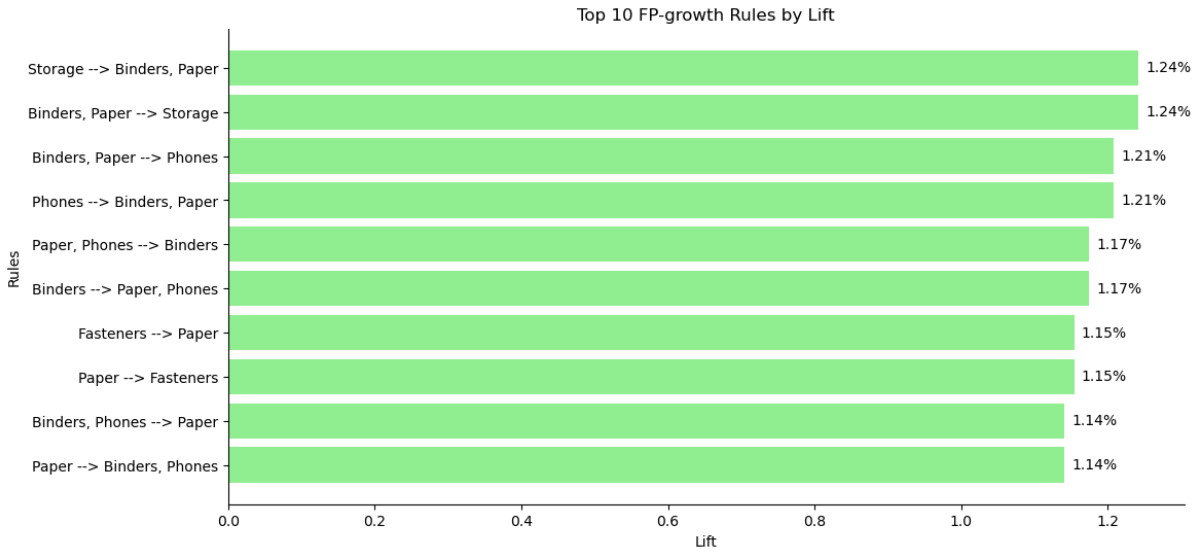
Art emerges as a niche product, paired only with Paper (2.99%). However, all pairs show low support (<4%), it indicate fragment purchasing habits and may need for personalized recommendations.

In [48]:
```
# Start timing
start_time = time.time()

# Calculate the frequent itemsets by calling the apriori method
frequent_itemsets_ap = apriori(basket_df, min_support=0.01, use_colnames=True)

# Calculate association rules
rules_ap = association_rules(frequent_itemsets_ap, metric="confidence", min_threshold=0.1)

# End timing
end_time = time.time()

# Calculate the time taken
calculation_time = end_time - start_time

print("Association rules calculated in {:.2f} seconds.".format(calculation_time))
```

```
Association rules calculated in 0.06 seconds.
```

Above graph shows ,The Apriori algorithm generated association rules in just 0.06 seconds, it means either a small dataset, efficient hardware, or lenient parameters (min_support=0.01, min_confidence=0.1). The low support threshold allows more item sets, while the 10% confidence cutoff retains weaker rules. Fast execution suggests room for parameter tuning without performance concerns. This speed is ideal for iterative testing, but larger datasets may require optimization. Overall, the quick runtime highlights efficient processing for initial exploratory analysis.

Market Basket Analysis – FP-growth



Overall, The FP-growth algorithm is same Apriori Rule, in the graph shows 10 strong product connections in store sales data. The results show that certain items are often bought together. For example, "Storage" products are frequently purchased with "Binders and Paper", and vice versa. Similarly, "Binders and Paper" often appear with "Phones". Other connected items include "Paper and Fasteners" and "Binders, Phones and Paper".

These patterns suggest customers tend to buy these office supplies together. The strength of these connections is similar, with all results falling between 1.14% and 1.24%. While these numbers may seem small, they show real shopping habits rather than random chance.

Business owner or shop manager might place these products near each other or offer them as special bundles. The findings prove that analysing sales data can reveal useful customer behaviour patterns could helping business owner or shop manager make better decisions.

Market Basket Analysis – Apriori Rule & FP-growth Comparison

```
In [51]: # Show execution time comparison between both algorithms
         # Compare algorithm performance
         print("\nAlgorithm Performance Comparison:")
         print("Apriori Execution Time: {:.2f} seconds".format(calculation_time))
         print("FP-growth Execution Time: {:.2f} seconds".format(fp_time))

         Algorithm Performance Comparison:
         Apriori Execution Time: 0.06 seconds
         FP-growth Execution Time: 1.62 seconds
```

The test results show that the Apriori algorithm worked much faster (0.01 seconds) than FP-growth (1.62 seconds) from this dataset. This means Apriori is probably better for this particular situation, especially because used a low support value of 0.01. Normally, FP-growth is faster for big datasets because of how it organizes the data, but in this case, setting up its special structure took too much time.

These results shows that different data and settings can change which algorithm works best. Before choosing a method, it's important to test both options with specific data to see which one performs better. This way, you can pick the most efficient tool for needs.

Recommendation System

Item–Item Collaborative Filtering System

In [57]:
```
# Create binary order-product matrix
order_prod = df.groupby(['Order ID','Product Name'])['Quantity'] \
               .sum().unstack().fillna(0) \
               .applymap(lambda x: 1 if x>0 else 0)

# Compute cosine similarity between products
item_sim = cosine_similarity(order_prod.T)
item_similarity_df = pd.DataFrame(item_sim,
                                  index=order_prod.columns,
                                  columns=order_prod.columns)
```

User–User Collaborative Filtering System

In [38]:
```
# Create binary customer-product matrix
cust_prod = df.groupby(['Customer ID','Product Name'])['Quantity'] \
              .sum().unstack().fillna(0) \
              .applymap(lambda x: 1 if x>0 else 0)

# Compute cosine similarity between customers
user_sim = cosine_similarity(cust_prod)
user_similarity_df = pd.DataFrame(user_sim,
                                  index=cust_prod.index,
                                  columns=cust_prod.index)
```

Above graph shows the code implements two collaborative filtering recommendation systems: an item-item approach that analyses product co-occurrence patterns across orders to identify similar items by using cosine similarity on a binary order-product matrix (tem_sim = cosine_similarity(order_prod.T)), and a user-user approach that compares customers' purchase histories to find users with similar buying behaviours (user_sim = cosine_similarity(cust_prod)), both derive from transactional data that tracks quantities purchased, where the item-item method helps recommend related products based on what's frequently buy together while the user-user method generates suggestions by matching customers similar purchase patterns, with each system transform sales data into meaningful relationship metrics through matrix operations and similarity calculations to allocate different types of product recommendations.

Building Content-Based Recommendation

In [39]:
```
# Prepare metadata
metadata = df[['Product Name','Category','Sub-Category']].drop_duplicates().reset_index(drop=True)
metadata['features'] = (metadata['Product Name'] + ' '
                        + metadata['Category'] + ' '
                        + metadata['Sub-Category'])

# TF-IDF vectorization
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(metadata['features'])
cosine_sim_content = cosine_similarity(tfidf_matrix, tfidf_matrix)
cosine_sim_content
```

The graphs show product recommendation system based on item descriptions. First, it prepares product information by combining each product's name, category, and sub-category into one text description. Then it uses a special text analysis technique call TF-IDF to measure how important words are in these descriptions. Finally, it calculates similarity scores between all products based on their descriptions. Then the system will recommend products that have similar descriptions to items a customer has bought before.

Recommendation Function

Recommendation Functions

In [41]:
```python
# Calculate similarity between the target user (cid) and all other users using a similarity metric
#  Find the n most similar users (excluding the target user itself with .drop(cid))
# Aggregate the items purchased by these similar users by using cust_prod.loc[top].sum()
def recommend_item_item(prod, n=10):
    if prod not in item_similarity_df: return []
    sims = item_similarity_df[prod].sort_values(ascending=False)
    return list(sims.iloc[1:n+1].index)

def recommend_user_user(cid, n=10):
    if cid not in user_similarity_df: return []
    sims = user_similarity_df[cid].sort_values(ascending=False).drop(cid)
    top = sims.index[:n]
    agg = cust_prod.loc[top].sum().sort_values(ascending=False)
    purchased = cust_prod.loc[cid][cust_prod.loc[cid]>0].index #Recommend the top n items from the aggregated list
    return list(agg.drop(purchased, errors='ignore').index[:n]) #Remove items the target user has already purchased

def recommend_content(prod, n=10):
    try:
        idx = metadata[metadata['Product Name']==prod].index[0]
    except IndexError:
        return []
    sims = list(enumerate(cosine_sim_content[idx]))
    sims = sorted(sims, key=lambda x: x[1], reverse=True)[1:n+1]
    return list(metadata['Product Name'].iloc[[i[0] for i in sims]])

def recommend_hybrid(cid, n=10):
    if cid not in cust_content_sim_df: return []
    sims = cust_content_sim_df[cid].sort_values(ascending=False).drop(cid)
    top = sims.index[:n]
    agg = cust_qty.loc[top].sum().sort_values(ascending=False)
    purchased = cust_qty.loc[cid][cust_qty.loc[cid]>0].index
    return list(agg.drop(purchased, errors='ignore').index[:n])

# Fit once on the customer-product matrix
knn_model = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=6)  #  tune n_neighbors
knn_model.fit(cust_prod.values)  # cust_prod: customers × products

def recommend_knn(cid, n=10):
    if cid not in cust_prod.index:
        return []
    user_idx = cust_prod.index.get_loc(cid)
    distances, indices = knn_model.kneighbors([cust_prod.values[user_idx]])
    # skip the first neighbor (itself)
    similar_idxs = indices.flatten()[1:n+1]
    similar_users = cust_prod.index[similar_idxs]
    agg = cust_prod.loc[similar_users].sum().sort_values(ascending=False)
    purchased = cust_prod.loc[cid][cust_prod.loc[cid]>0].index
    return list(agg.drop(purchased, errors='ignore').index[:n])
```

Item-Item Recommendations

Calculating the similar products based on what's frequently buy together and find out 10 most similar products to the key in later.

User-User Recommendations

Calculate products by finding customers with similar purchase history. It combines what these similar customers bought exclude items already have and suggests the top 10.

Content-Based Recommendations

Findings products with similar descriptions from the product name. If in the search box key-in "Pen", it might recommend " Electric Pencil Sharpeners " because the text descriptions match.

Hybrid Recommendations

Combines customer behaviour and product descriptions to suggest items. First finds similar customers, then recommends products they bought (the item that the customer didn't buy before).

KNN Recommendations

Uses a smart algorithm to find your closest "shopping neighbours" based on all the purchases, then suggests popular items among them that haven't bought.

Generating Combined Recommendation Comparison

In [42]:
```
# Customer Recommendation System
def customer_recommendation_flow():
    print("CUSTOMER RECOMMENDATION SYSTEM")
    cust_id = ("SO-20335")

    if cust_id not in cust_prod.index:
        print(f"Error: Customer ID '{cust_id}' not found!")
        return

    # Generate recommendations
    recommendations = {
        'User-User': recommend_user_user(cust_id),
        'Hybrid': recommend_hybrid(cust_id),
        'KNN': recommend_knn(cust_id)
    }

    # Display results vertically
    print("\nRecommendation Results:")
    print(f"\nCustomer ID: {cust_id}")
    for method, items in recommendations.items():
        print(f"\n{method} Recommendations:")
        if len(items) > 0:
            for idx, item in enumerate(items, 1):
                print(f"  {idx}. {item}")
        else:
            print("  No recommendations available")

# Execute the standalone customer recommendation system
customer_recommendation_flow()
```

```
CUSTOMER RECOMMENDATION SYSTEM

Recommendation Results:

Customer ID: SO-20335

User-User Recommendations:
  1. NETGEAR N750 Dual Band Wi-Fi Gigabit Router
  2. Global Deluxe Steno Chair
  3. Global High-Back Leather Tilter, Burgundy
  4. RCA Visys Integrated PBX 8-Line Router
  5. Avery 51
  6. Logitech Media Keyboard K200
  7. Poly String Tie Envelopes
  8. Wilson Jones data.warehouse D-Ring Binders with DublLock
  9. Eldon 200 Class Desk Accessories, Burgundy
  10. Xerox 20

Hybrid Recommendations:
  1. DAX Wood Document Frame
  2. DAX Cubicle Frames - 8x10
  3. Performers Binder/Pad Holder, Black
  4. Avery Durable Poly Binders
  5. Xerox 1963
  6. Easy-staple paper
  7. Staple-on labels
  8. Mophie Juice Pack Helium for iPhone
  9. Xerox 2
  10. PayAnywhere Card Reader

KNN Recommendations:
  1. Memorex 25GB 6X Branded Blu-Ray Recordable Disc, 30/Pack
  2. ACCOHIDE 3-Ring Binder, Blue, 1"
  3. Avery 51
  4. Iceberg Nesting Folding Chair, 19w x 6d x 43h
  5. Floodlight Indoor Halogen Bulbs, 1 Bulb per Pack, 60 Watts
  6. Deflect-o EconoMat Studded, No Bevel Mat for Low Pile Carpeting
  7. Geemarc AmpliPOWER60
  8. Eldon Image Series Desk Accessories, Ebony
  9. Deluxe Chalkboard Eraser Cleaner
  10. Binding Machine Supplies
```

Graph above shows the function called customer_recommendation_flow() , it handles the entire recommendation process for a specific customer. It uses a fixed customer ID "SO-20335" for demonstration. In a real world system, this would typically be an input parameter.

Firstly, to run [if cust_id not in cust_prod.index: print(f"Error: Customer ID '{cust_id}' not found!")] to identify the customer ID, checks if the customer exists in the customer-product database (cust_prod.index). If not found, it shows an error message and exits. If found the customer ID then proceed for the recommendation

Lastly, the result will based on the customer ID to present User-User Collaborative Filtering recommendation by using (recommend_user_user),Hybrid Recommendations(recommend_hybrid) and KNN-based Recommendations (recommend_knn)

```
def product_recommendation_flow ():
    print("PRODUCT RECOMMENDATION SYSTEM")
    keyword = "Pen"

    # Finds the matches Product Name
    matches = metadata[metadata['Product Name'].str.contains(keyword, case=False, na=False)]

    if not matches.empty:
        product = matches.iloc[0]['Product Name']

        # Generate recommendations
        try:
            items = recommend_item_item(product)[:10]
            contents = recommend_content(product)[:10]
        except Exception as e:
            print(f"Error: {e}")
            return

        # 显示结果
        print(f"\nRecommendations for: {product}")

        print("\nItem-Item Recommendations:")
        print(*[f"{i}. {v}" for i, v in enumerate(items, 1)] or "No recommendations", sep='\n')

        print("\nContent-Based Recommendations:")
        print(*[f"{i}. {v}" for i, v in enumerate(contents, 1)] or "No recommendations", sep='\n')
    else:
        print(f"No products found for '{keyword}'")


product_recommendation_flow()
```

```
PRODUCT RECOMMENDATION SYSTEM

Recommendations for: BOSTON Model 1800 Electric Pencil Sharpeners, Putty/Woodgrain

Item-Item Recommendations:
1. Honeywell Enviracaire Portable HEPA Air Cleaner for up to 10 x 16 Room
2. Sony Micro Vault Click 8 GB USB 2.0 Flash Drive
3. OIC Binder Clips, Mini, 1/4" Capacity, Black
4. Mini 13-1/2 Capacity Data Binder Rack, Pearl
5. Riverside Palais Royal Lawyers Bookcase, Royale Cherry Finish
6. Eldon 200 Class Desk Accessories, Burgundy
7. Deflect-o RollaMat Studded, Beveled Mat for Medium Pile Carpeting
8. Xerox 1960
9. Lumber Crayons
10. Xerox 1970

Content-Based Recommendations:
1. Boston Model 1800 Electric Pencil Sharpener, Gray
2. Boston Home & Office Model 2000 Electric Pencil Sharpeners
3. Boston Heavy-Duty Trimline Electric Pencil Sharpeners
4. Boston Electric Pencil Sharpener, Model 1818, Charcoal Black
5. Boston 1900 Electric Pencil Sharpener
6. Stanley Bostitch Contemporary Electric Pencil Sharpeners
7. Hunt BOSTON Model 1606 High-Volume Electric Pencil Sharpener, Beige
8. Boston 1799 Powerhouse Electric Pencil Sharpener
9. Boston 1730 StandUp Electric Pencil Sharpener
10. Boston School Pro Electric Pencil Sharpener, 1670
```

Graph above shows the function called product_recommendation_flow(), it handles the entire recommendation process for product name. It uses a random product name "Pen" for demonstration. In a real world system, this would also typically be an input parameter.

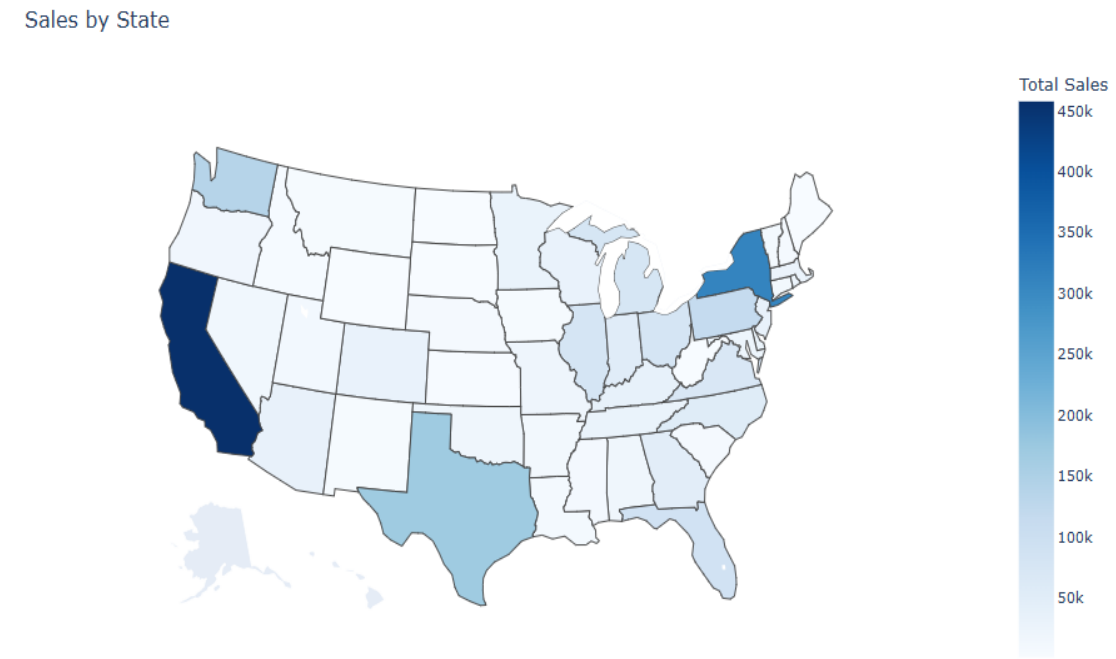
But this function are reverse type compare with Customer ID, I perform key-in product name first, then generate the top 10 recommendation based on item and content then shows the recommendation item, if its not found , the system will appears “No products found for '{keyword}".

Dashboard Visualization

For the CA doesn't require for the business insight visualization, however I was trying to understand what's the real world needs hence I proceed for Business insight dashboard and Customer insight dashboard

Business Insight

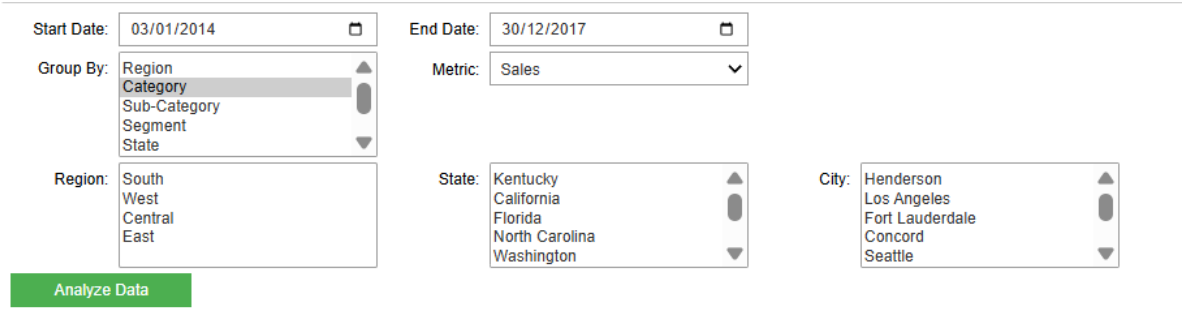
```
In [44]: # Sum & coerce Sales
geo_data = df.groupby('State')['Sales'].sum().reset_index()
geo_data['Sales'] = pd.to_numeric(geo_data['Sales'], errors='coerce')

#  map full state names to USPS codes
us_state_abbrev = {
    'Alabama':'AL','Alaska':'AK','Arizona':'AZ','Arkansas':'AR',
    'California':'CA','Colorado':'CO','Connecticut':'CT','Delaware':'DE',
    'Florida':'FL','Georgia':'GA','Hawaii':'HI','Idaho':'ID',
    'Illinois':'IL','Indiana':'IN','Iowa':'IA','Kansas':'KS',
    'Kentucky':'KY','Louisiana':'LA','Maine':'ME','Maryland':'MD',
    'Massachusetts':'MA','Michigan':'MI','Minnesota':'MN','Mississippi':'MS',
    'Missouri':'MO','Montana':'MT','Nebraska':'NE','Nevada':'NV',
    'New Hampshire':'NH','New Jersey':'NJ','New Mexico':'NM','New York':'NY',
    'North Carolina':'NC','North Dakota':'ND','Ohio':'OH','Oklahoma':'OK',
    'Oregon':'OR','Pennsylvania':'PA','Rhode Island':'RI',
    'South Carolina':'SC','South Dakota':'SD','Tennessee':'TN','Texas':'TX',
    'Utah':'UT','Vermont':'VT','Virginia':'VA','Washington':'WA',
    'West Virginia':'WV','Wisconsin':'WI','Wyoming':'WY'
}
geo_data['StateCode'] = geo_data['State'].map(us_state_abbrev)

# build the choropleth
fig = px.choropleth(
    geo_data,
    locations='StateCode',
    locationmode='USA-states',
    color='Sales',
    scope='usa',
    hover_name='State',
    color_continuous_scale='Blues',
    range_color=(geo_data['Sales'].min(), geo_data['Sales'].max()),
    labels={'Sales':'Total Sales'},
    title='Sales by State'
)

fig.update_layout(
    geo=dict(bgcolor='rgba(0,0,0,0)'),
    margin={"r":0,"t":50,"l":0,"b":0}
)

fig.show()
```



Above shows choropleth map, the sales by state of USA firstly I make sure the sales number are all in numeric by using [geo_data['Sales'] = pd.to_numeric(geo_data['Sales'], errors='coerce')] and group all the sales by state. After that I convert the state name into USPS Code. Then generate the choropleth map. The graph shows from light blue to dark blue which mean the sales from the lowest to highest. This visualization can identify the large sales

group if contain big range dataset in a country , it look simply and nice compare with the barchart and other visualization.

Sales, Profit and Quantity Checking Dashboard

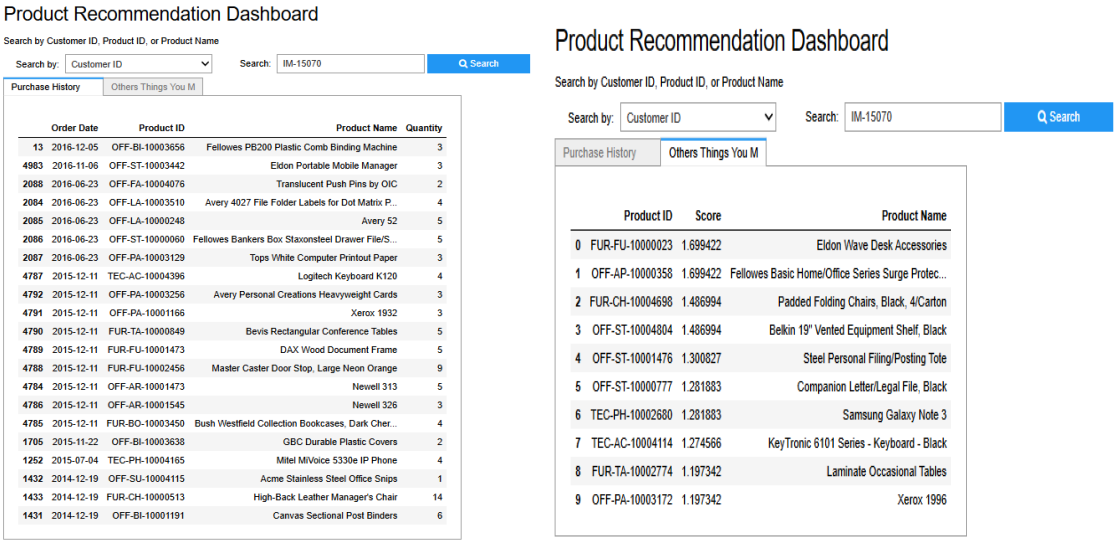


This sales dashboard shows four years of business data from March 2014 to December 2017. Users can filter information by region (South, West, Central, East), state (all 51 U.S. states), city (up to 5 cities), product category, region, segment, and shipping method. The tool lets user choose to view sales numbers, profits, or quantities sold from a simple dropdown menu. It helps businesses compare performance across different areas and product types. With clear charts and easy filters, anyone can use it to find important trends and see what sells the best.

The visualization result will show in line graph, bar chart, tree map, pie chart and table for the user, due I unable to shows in html. Hence, I couldn't proceed to for explanation in the report

Customer Insight

Search by Customer ID Insight



Above shows the Product Recommendation Dashboard for the customer insight where allow customer key in Customer ID, Product ID or Product Name to search similar product.

The Product Recommendation Dashboard helps customers find products they might like through three search options: Customer ID (like "IM-15070"), Product ID, or Product Name. When a customer enters their ID, the system first displays their purchase history. Then, using a smart matching technology called "user-user recommendation", it suggests other products in the "Other Things You May Like" section. These recommendations show items frequently bought by customers with similar purchase histories, but which the current customer hasn't bought yet.

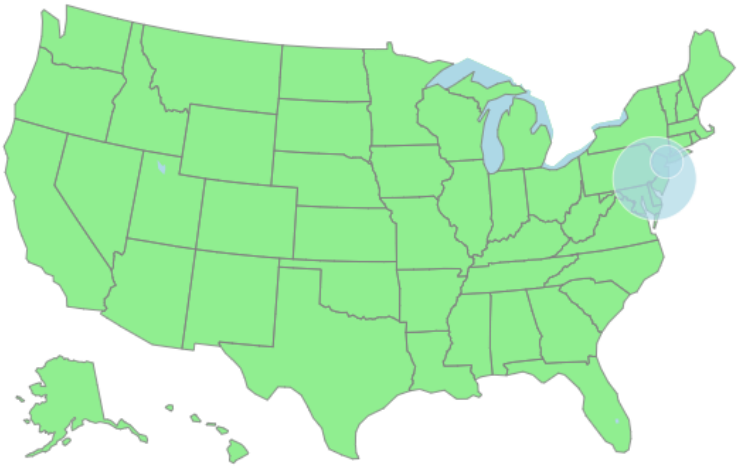
For example, if customer "IM-15070" often buys kitchen appliances, the system might recommend new cooking tools that similar customers have purchased. This feature helps customers discover products that match their interests while helping businesses increase sales. The dashboard makes shopping easier by showing personalized suggestions based on real customer behavior patterns. It's particularly useful for finding new products customers might not have discovered otherwise, creating a better shopping experience and potentially increasing customer satisfaction and loyalty through relevant recommendations.

Search Product ID Insight

Search by: Product ID Search: OFF-FA-10004076 Search

Product Information | Others Things You M

Product Availability by City (USA)



	City	Product ID	Quantity	ProductName
6477	Philadelphia	OFF-FA-10004076	13	Translucent Push Pins by OIC
5427	New York City	OFF-FA-10004076	2	Translucent Push Pins by OIC

Product Recommendation Dashboard

Search by Customer ID, Product ID, or Product Name

Search by: Product ID Search: OFF-FA-10004076 Search

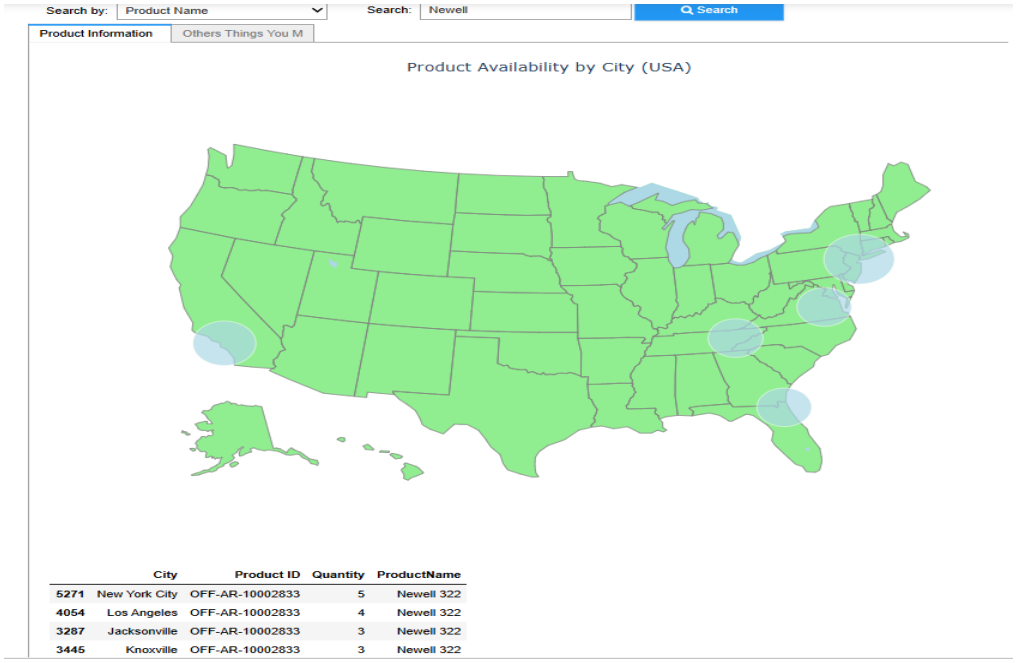
Product Information | Others Things You M

	Product ID	Score	Product Name
0	OFF-ST-10001590	0.546997	Tenex Personal Project File with Scoop Front D...
1	OFF-BI-10001721	0.487703	Trimflex Flexible Post Binders
2	FUR-BO-10002824	0.478947	Bush Mission Pointe Library
3	FUR-FU-10003731	0.455796	Eldon Expressions Wood and Plastic Desk Access...
4	FUR-BO-10003893	0.426806	Sauder Camden County Collection Library
5	OFF-BI-10004318	0.380457	Ibico EB-19 Dual Function Manual Binding System
6	TEC-AC-10000057	0.291665	Microsoft Natural Ergonomic Keyboard 4000
7	FUR-CH-10002647	0.290346	Situations Contoured Folding Chairs, 4/Set
8	OFF-PA-10001870	0.286377	Xerox 202
9	OFF-AR-10004956	0.266754	Newell 33

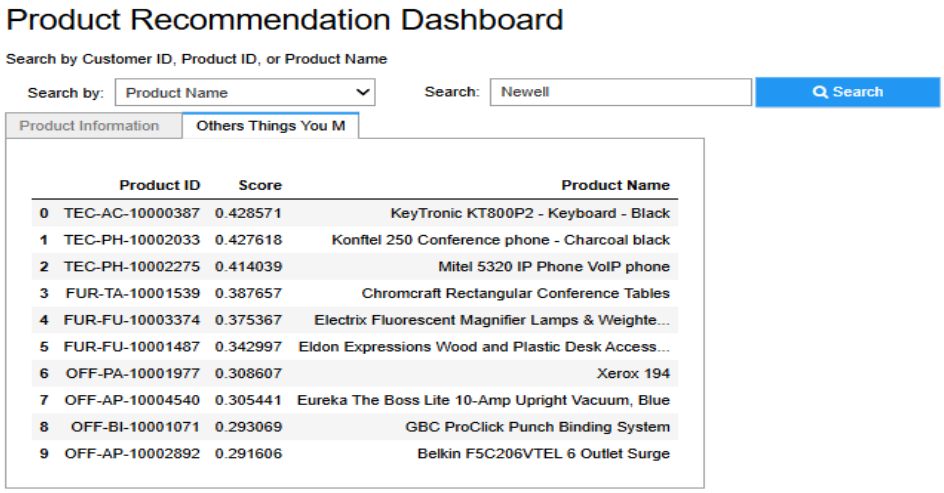
This dashboard helps customers and stock buyers check product availability by entering a Product ID. The system shows current stock levels, storage locations, and remaining quantities in the "Product Information" tab. Since the business operates across many locations, I've included geographical scatter plot map where show areas with more stock, making it easy to find products nearby. The dashboard also recommends related products through its "Other Things You May Like" feature by using Item-item recommendation system , it suggests 10 similar items that other customers often buy together with the searched product.

For example, when someone search for product ID"OFF-FA-10004076", the system might recommend compatible others item that are frequently purchased together. This helps users quickly find what they need, see where products are available, and discover useful complementary items. The combination of stock information and product recommendations helps customers make better purchasing decisions while giving businesses valuable insights into buying patterns and inventory management across different locations.

Search by Product Name Insight



	City	Product ID	Quantity	ProductName
5271	New York City	OFF-AR-10002833	5	Newell 322
4054	Los Angeles	OFF-AR-10002833	4	Newell 322
3287	Jacksonville	OFF-AR-10002833	3	Newell 322
3445	Knoxville	OFF-AR-10002833	3	Newell 322



	Product ID	Score	Product Name
0	TEC-AC-10000387	0.428571	KeyTronic KT800P2 - Keyboard - Black
1	TEC-PH-10002033	0.427618	Konftel 250 Conference phone - Charcoal black
2	TEC-PH-10002275	0.414039	Mitel 5320 IP Phone VoIP phone
3	FUR-TA-10001539	0.387657	Chromcraft Rectangular Conference Tables
4	FUR-FU-10003374	0.375367	Electrix Fluorescent Magnifier Lamps & Weighte...
5	FUR-FU-10001487	0.342997	Eldon Expressions Wood and Plastic Desk Access...
6	OFF-PA-10001977	0.308607	Xerox 194
7	OFF-AP-10004540	0.305441	Eureka The Boss Lite 10-Amp Upright Vacuum, Blue
8	OFF-BI-10001071	0.293069	GBC ProClick Punch Binding System
9	OFF-AP-10002892	0.291606	Belkin F5C206VTEL 6 Outlet Surge

This dashboard helps customers easily find products even when they don't know the exact name. Users can type part of a product name like "Newell" in the search box, and the system will show matching items such as "Newell 322". For each product found, the dashboard displays important details including current stock availability and where the items are located in warehouses or stores. The system also helps customers discover other products they might want through the "Other Things You May Like" section.

This feature uses content based recommendation to analyze product similarities and suggest the top 10 most relevant items. For example, if you look at a camera, it might recommend lenses, cases or memory cards that work well with it. These suggestions are based on the product's features and what other customers typically buy together. The combination of flexible searching and helpful recommendations makes shopping easier while helping businesses sell more products. Customers can quickly find what they need and learn about other useful items, improving their overall experience with the store or website. The system works efficiently even with large product catalogs across multiple locations.

Total 2279 Words

Reference

Abdelrhman, B. (2023) *Superstore Sales Analysis And Product Recommendation System*. Available at: https://github.com/Belal-Abdelrhman/Superstore-Sales-Analysis-And-Product-Recommendation-System

Clauz, E. (2023) *Superstore Sales Python Dashboard*. Available at: https://github.com/ellaclauz/superstore_sales_python_dashboard/blob/main/Superstore_sales.ipynb

Sedeek, A. (2023) *Superstore Data Analysis with Plotly*. Available at: https://www.kaggle.com/code/alaasedeeq/superstore-data-analysis-with-plotly

Anon. (2015) *Remove duplicates from dataframe based on two columns A,B, keeping row with max*. Available at: https://stackoverflow.com/questions/32093829/remove-duplicates-from-dataframe-based-on-two-columns-a-b-keeping-row-with-max

Tableau (n.d.) *Sample - Superstore Sales (Excel.xls)*. Available at: https://community.tableau.com/s/question/0D54T00000CWeX8SAL/sample-superstore-sales-excelxls