UGANDA MARTYRS UNIVERSITY

FACULTY OF SCIENCE

MASTER OF SCIENCE IN INFORMATION SYSTEMS

INTELLIGENT SYSTEMS

MIS 6317

MUGASHA BLAISE BRADLEY

REG NO: 2024-M132-23976

Question 2 GitHub link

https://github.com/XerxesBla4e/online-retail-ii-uci-ML

Part A: Data Cleaning and Clustering

A.1 Load Dataset

- **Action**: The dataset 'online-retail-ii-uci' was downloaded using kagglehub and loaded into a pandas DataFrame. The dataset path was /kaggle/input/online-retail-ii-uci/online_retail_II.csv.
- **Outcome**: The DataFrame df was created, containing the raw transactional data. The first 5 rows were displayed, showing columns
like Invoice, StockCode, Description, Quantity, InvoiceDate, Price, Customer ID, and Country.

```
import pandas as pd
import numpy as np

df = pd.read_csv("/kaggle/input/online-retail-ii-uci/online_retail_II.csv", encoding="latin1")
df.head()
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

A.2 Remove missing descriptions, negative quantities, cancelled invoices

- **Action**: Data cleaning steps were applied to the raw DataFrame:
    - Rows with missing Description were removed.
    - Rows with negative Quantity (returns) were removed.
    - Invoices containing 'C' (cancelled invoices) were removed.
    - A new column TotalPrice was calculated as Quantity * Price.
- **Outcome**: The DataFrame was cleaned, reducing the number of rows from the original to (1042727, 9), focusing on valid sales transactions.

```python
# Remove missing descriptions
df = df.dropna(subset=["Description"])

# Remove negative quantities
df = df[df["Quantity"] >= 0]

# Remove cancelled invoices
df = df[~df["Invoice"].astype(str).str.contains("C", na=False)]

# Recalculate TotalPrice
df["TotalPrice"] = df["Quantity"] * df["Price"]

# Check remaining rows
print(df.shape)
```

```
(1042727, 9)
```

A.3 Create customer-level features

- **Action**: Customer-level features were engineered by grouping the cleaned DataFrame by Customer ID.
    - TotalSpending: Sum of TotalPrice for each customer.
    - TransactionCount: Number of unique Invoices for each customer.
    - AvgBasketSize: Average Quantity per customer.
- **Outcome**: A new DataFrame customer_df was created with these aggregated features. The head of this DataFrame was displayed, showing the Customer ID, TotalSpending, TransactionCount, and AvgBasketSize.

```python
customer_df = df.groupby("Customer ID").agg(
    TotalSpending=("TotalPrice", "sum"),
    TransactionCount=("Invoice", "nunique"),
    AvgBasketSize=("Quantity", "mean")
).reset_index()

customer_df.head()
```

	Customer ID	TotalSpending	TransactionCount	AvgBasketSize
0	12346.0	77556.46	12	2184.852941
1	12347.0	5633.32	8	12.988142
2	12348.0	2019.40	5	53.215686
3	12349.0	4428.69	4	9.280000

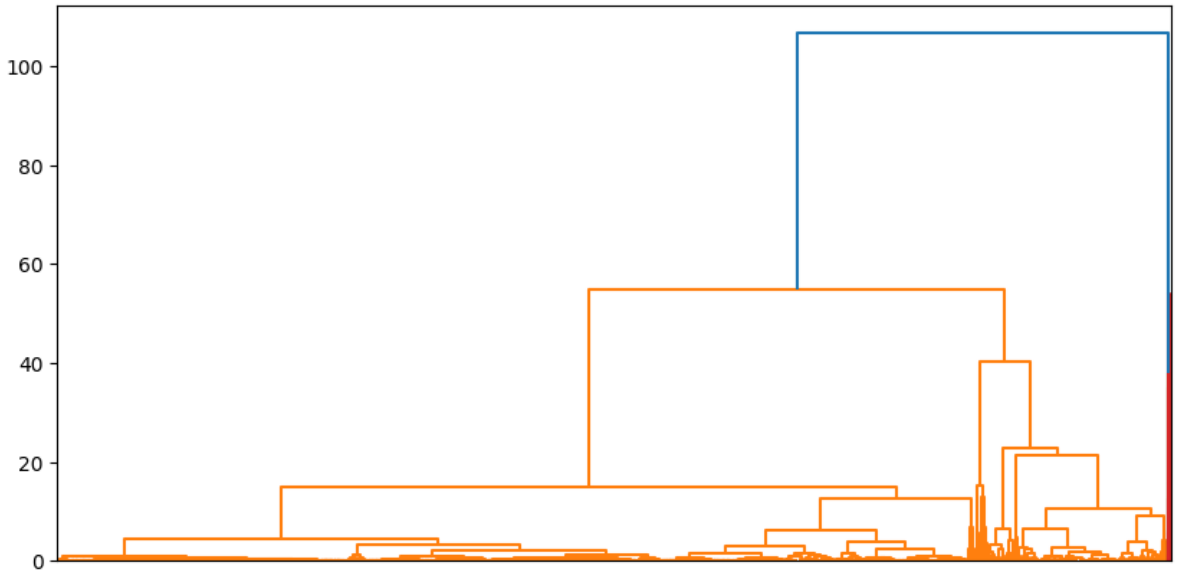
A.4 Apply k-Means and DBSCAN/Hierarchical clustering

- **Action**: Customer features (TotalSpending, TransactionCount, AvgBasketSize) were standardized using StandardScaler.
    - **k-Means Clustering**: Applied with n_clusters=3.
    - **DBSCAN Clustering**: Applied with eps=1.1 and min_samples=5.
    - **Hierarchical Clustering**: A dendrogram was generated using linkage with the 'ward' method to visualize the hierarchical structure.

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(customer_df[['TotalSpending', 'TransactionCount', 'AvgBasketSize']])

# k-Means
kmeans = KMeans(n_clusters=3, random_state=42)
customer_df['kmeans_label'] = kmeans.fit_predict(X_scaled)

# DBSCAN-Density-Based Spatial Clustering of Applications with Noise
dbscan = DBSCAN(eps=1.1, min_samples=5)
customer_df['dbscan_label'] = dbscan.fit_predict(X_scaled)
```

- **Outcome**:
    - k-Means labels (kmeans_label) and DBSCAN labels (dbscan_label) were added to customer_df.
    - A dendrogram plot was displayed, illustrating the merges and distances in hierarchical clustering.

```
# Hierarchical clustering dendrogram
linked = linkage(X_scaled, method='ward')
plt.figure(figsize=(10, 5))
dendrogram(linked)
plt.show()
```

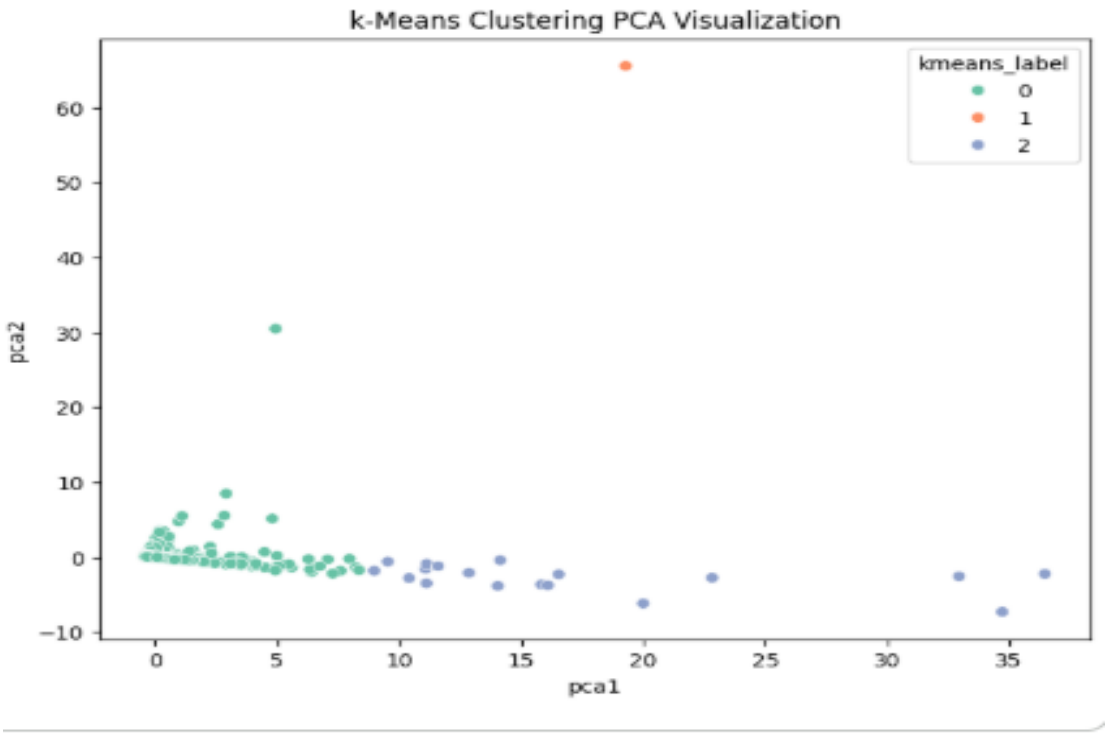


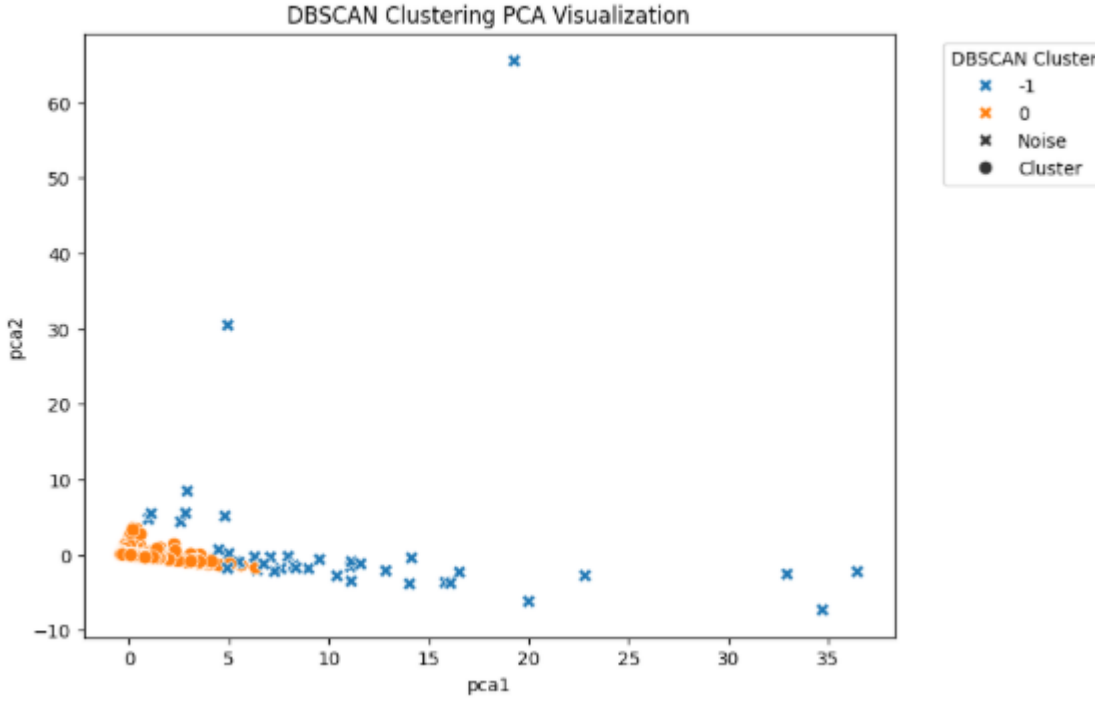
A.5 Compute silhouette scores and visualize

- **Action**: Silhouette scores were computed for both k-Means and DBSCAN clustering to evaluate cluster quality.
    - Principal Component Analysis (PCA) was used to reduce the standardized features to 2 dimensions for visualization.
    - Scatter plots were created to visualize the k-Means and DBSCAN clusters in a 2D PCA space.
- **Outcome**:
    - k-Means Silhouette Score: 0.9640.
    - DBSCAN Silhouette Score: 0.9480.

```
k-Means Silhouette: 0.9640418472719594
DBSCAN Silhouette: 0.9480260696909127
```

    - Two scatter plots were displayed, showing the clusters formed by k-Means and DBSCAN on the PCA-reduced data. This visualization helped in understanding the separation and density of the clusters.





Part B: Deep Embedding Clustering

B.1 Autoencoder for Customer Embeddings

- **Action**: A simple autoencoder neural network was defined and trained using PyTorch.
    - Input features (TotalSpending, TransactionCount, AvgBasketSize) were converted to PyTorch tensors.
    - The autoencoder consisted of an encoder (input_dim -> 8 -> 3) and a decoder (3 -> 8 -> input_dim).
    - The model was trained for 50 epochs using MSELoss and Adam optimizer.
- **Outcome**: The autoencoder was trained to learn a compressed, 3-dimensional latent representation (embeddings) of the customer features.

B.2 Extract latent embeddings

- **Action**: The trained autoencoder's encoder part was used to extract the latent embeddings for all customers.
- **Outcome**: A NumPy array embeddings of shape (num_customers, 3) was obtained, representing the compressed features for each customer. The first few embeddings were displayed.

```
embeddings
```

```
array([[ 69773.33   , -47980.7    , -24232.89   ],
       [  5062.4077 ,  -3480.2441 ,  -1759.0264 ],
       [  1821.6736 ,  -1253.7073 ,   -633.5993 ],
       ...,
       [   385.5873 ,   -265.70306,   -135.28764],
       [  1166.1318 ,   -802.12274,   -406.30835],
       [  3761.1665 ,  -2586.0996 ,  -1307.1304 ]], dtype=float32)
```

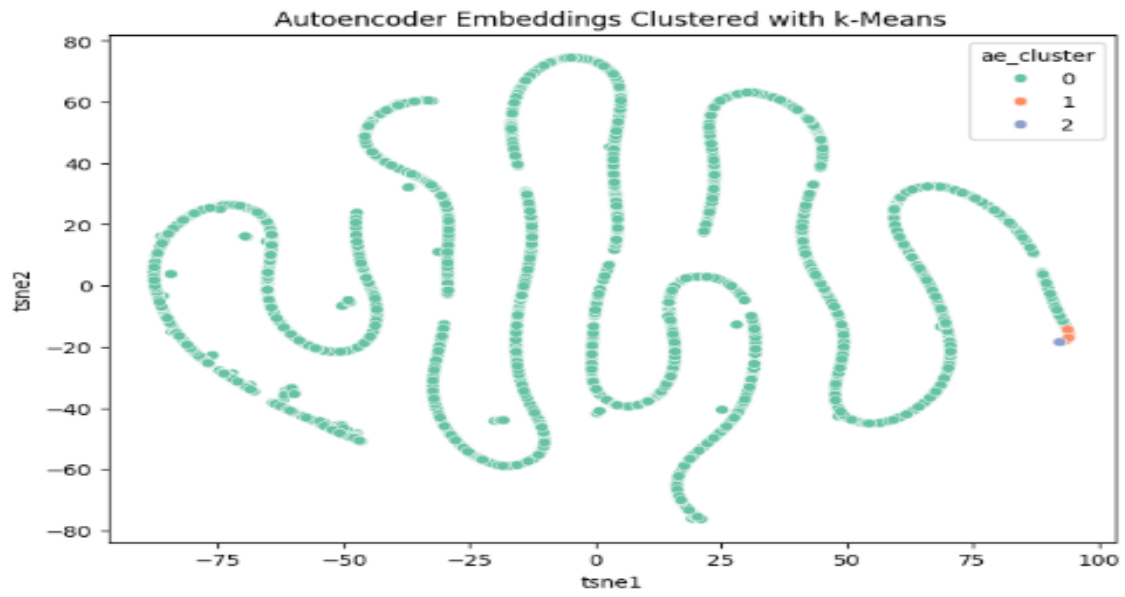
B.3 Cluster embeddings with k-Means

- **Action**: k-Means clustering (n_clusters=3) was applied to the extracted autoencoder embeddings.
- **Outcome**: A new column ae_cluster was added to customer_df, assigning each customer to a cluster based on their autoencoder embeddings.

```python
from sklearn.cluster import KMeans

kmeans_emb = KMeans(n_clusters=3, random_state=42)
customer_df['ae_cluster'] = kmeans_emb.fit_predict(embeddings)
```

B.4 Visualize embeddings (2D)

- **Action**: t-SNE (t-Distributed Stochastic Neighbour Embedding) was used to reduce the 3-dimensional autoencoder embeddings to 2 dimensions for visualization.
    - A scatter plot was created to visualize the clusters formed by k-Means on the t-SNE-reduced embeddings.
- **Outcome**:
    - New columns tsne1 and tsne2 were added to customer_df.
    - A scatter plot was displayed, showing the autoencoder embeddings clustered with k-Means in a 2D t-SNE space. The visualization indicated one dominant structure with minimal sub-cluster separation.



B.5 Compare with PCA clusters

- **Action**: The silhouette score was computed for the k-Means clustering on the autoencoder embeddings.
- **Outcome**: The autoencoder embeddings silhouette score was 0.9679. This was compared to the silhouette scores from Part A, noting its similarity to the standard k-Means score and explaining that the autoencoder did not capture significantly more non-linear patterns.

```
from sklearn.metrics import silhouette_score

sil_score_ae = silhouette_score(embeddings, customer_df['ae_cluster'])
print("Autoencoder Embeddings Silhouette:", sil_score_ae)
```

```
Autoencoder Embeddings Silhouette: 0.96404254
```

Part C: Association Rule Mining

C.1 Convert data into basket format

- **Action**: The df DataFrame was further cleaned and transformed to prepare for association rule mining:
    - Rows with Description as NA were dropped.
    - Whitespace was stripped from Description.
    - Garbage rows (e.g., those containing numerical stock codes, 'wrong', 'update') were removed.
    - Transactions were grouped by Invoice to create a list of items (Description) for each invoice, representing a shopping basket.
- **Outcome**: The df DataFrame was refined, and a basket Series was created where each entry is a list of items bought in a particular invoice. The head of the basket Series was displayed.

```python
# Remove rows where Description is NA
df = df.dropna(subset=["Description"])

# Clean whitespace, tabs, newlines
df["Description"] = df["Description"].str.strip()

# Remove garbage rows (notes, errors, etc)
mask = df["Description"].str.contains(r"[0-9]{5}|wrong|update|fixed|website|mix up|barcode", cas
df = df[~mask]
```

```python
basket = df.groupby("Invoice")["Description"].apply(list)
basket.head()
```

```python
basket.head()
```

Invoice	Description
489434	[15CM CHRISTMAS GLASS BALL 20 LIGHTS, PINK CHE...
489435	[CAT BOWL, DOG BOWL , CHASING BALL DESIGN, HEA...
489436	[DOOR MAT BLACK FLOCK, LOVE BUILDING BLOCK WOR...
489437	[CHRISTMAS CRAFT HEART DECORATIONS, CHRISTMAS ...
489438	[DINOSAURS WRITING SET, SET OF MEADOW FLOWER...

dtype: object

C.2 Prepare Basket Data for Binary Matrix

- **Action**: The basket Series was converted into a list of lists, where each inner list represents a transaction, making it suitable for mlxtend's TransactionEncoder.
- **Outcome**: The transactions list was created, and the first 5 transactions were printed, confirming the correct format.

```python
transactions = basket.tolist()
print("First 5 transactions in list of lists format:")
for i in range(5):
    print(transactions[i])
```

```
First 5 transactions in list of lists format:
['15CM CHRISTMAS GLASS BALL 20 LIGHTS', 'PINK CHERRY LIGHTS', 'WHITE CHERRY LIGHTS', 'RECORD FRAM
['CAT BOWL', 'DOG BOWL , CHASING BALL DESIGN', 'HEART MEASURING SPOONS LARGE', 'LUNCHBOX WITH CUT
['DOOR MAT BLACK FLOCK', 'LOVE BUILDING BLOCK WORD', 'HOME BUILDING BLOCK WORD', 'ASSORTED COLOUR
['CHRISTMAS CRAFT HEART DECORATIONS', 'CHRISTMAS CRAFT HEART STOCKING', 'PARTY CONE CHRISTMAS DEC
['DINOSAURS  WRITING SET', 'SET OF MEADOW  FLOWER STICKERS', 'CHARLIE AND LOLA CHARLOTTE BAG', 'J
```

C.3 Build Binary Matrix

- **Action**: TransactionEncoder from mlxtend.preprocessing was used to transform the transactions list into a one-hot encoded DataFrame (binary matrix).
- **Outcome**: A DataFrame df_encoded was created, where rows represent invoices and columns represent unique items, with True indicating the presence of an item in a transaction and False otherwise. The shape and head of df_encoded were displayed.

```
from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

print("Shape of the one-hot encoded DataFrame:", df_encoded.shape)
df_encoded.head()
```

```
Shape of the one-hot encoded DataFrame: (40286, 5368)
```

	*Boombox Ipod Classic	*USB Office Glitter Lamp	*USB Office Mirror Ball	10 COLOUR SPACEBOY PEN	11 PC CERAMIC TEA SET POLKADOT	12 ASS ZINC CHRISTMAS DECORATIONS	12 COLOURED PARTY BALLOONS	12 DAISY PEGS IN WOOD BOX	12 EGG HOUSE PAINTED WOOD	12 HANGING EGGS HAND PAINTED
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False

C.4 Apply FP-Growth and Generate Association Rules

Why we used FP Growth instead of Apriori

Apriori is extremely memory hungry because it generates candidate itemsets level by level. At every level it creates huge intermediate tables, and when a dataset has thousands of unique items like ours does (5368 columns after one hot encoding), Apriori immediately explodes in both RAM and time.

Our one hot encoded shape:
40286 rows
5368 unique items, producing about 216 million cells, making a giant search space.
Apriori tries to generate combinations like:

- All pairs among the 5368 items
- All triplets among items that survived
- All 4 item combinations
- And so, on

This number grows faster than your hardware can handle. That is why Colab died and your local machine tried to allocate more than 17 GB.

FP Growth completely avoids generating candidate itemsets. It compresses the dataset into an FP tree and mines patterns directly. That is why it succeeds where Apriori fails.

Apply FP-Growth and Generate Association Rules

Action

- The FP-Growth algorithm (fpgrowth from mlxtend.frequent_patterns) was applied to df_encoded to find frequent itemsets, using a min_support of 0.01.
- Association rules were generated from these frequent itemsets using association_rules with metric="confidence" and min_threshold=0.5.
- The generated rules were then sorted by lift and confidence in descending order.

Outcome

- frequent_itemsets DataFrame was created, showing itemsets that appear in at least 1% of transactions. The head of this DataFrame and the total number of frequent itemsets were printed (1065 itemsets).

```
from mlxtend.frequent_patterns import fpgrowth

# Apply FP-Growth algorithm to find frequent itemsets
frequent_itemsets = fpgrowth(df_encoded, min_support=0.01, use_colnames=True)

print("Frequent Itemsets:")
print(frequent_itemsets.head())
print(f"Number of frequent itemsets found: {len(frequent_itemsets)}")
```

```
Frequent Itemsets:
    support                                itemsets
0  0.057340       (STRAWBERRY CERAMIC TRINKET BOX)
1  0.019337                  (SAVE THE PLANET MUG)
2  0.017227            (PINK DOUGHNUT TRINKET POT)
3  0.013776          (RECORD FRAME 7" SINGLE SIZE)
4  0.012833  (15CM CHRISTMAS GLASS BALL 20 LIGHTS)
Number of frequent itemsets found: 1065
```

- The rules DataFrame was created, containing various metrics for each association rule like support, confidence, lift, etc. The top 10 rules sorted by lift and confidence were displayed, revealing strong associations, particularly around 'POPPY'S PLAYHOUSE' items.

Rank	Antecedents (IF)	Consequents (THEN)	Support	Confidence	Lift
1	POPPY'S PLAYHOUSE BEDROOM, POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BATHROOM	POPPY'S PLAYHOUSE LIVINGROOM	0.010152	0.734291	52.449718
2	POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BATHROOM	POPPY'S PLAYHOUSE BEDROOM	0.010152	0.725177	52.449718
3	POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BATHROOM	POPPY'S PLAYHOUSE BEDROOM	0.010152	0.862869	49.447437
4	POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BEDROOM	POPPY'S PLAYHOUSE BATHROOM	0.010152	0.581792	49.447437
5	POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUSE BEDROOM, POPPY'S PLAYHOUSE BATHROOM	POPPY'S PLAYHOUSE KITCHEN	0.010152	0.551213	48.169554
6	POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BEDROOM, POPPY'S PLAYHOUSE BATHROOM	POPPY'S PLAYHOUSE LIVINGROOM	0.010152	0.887202	48.169554
7	POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BEDROOM	POPPY'S PLAYHOUSE LIVINGROOM	0.011443	0.817376	46.840405
8	POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUSE KITCHEN	POPPY'S PLAYHOUSE BEDROOM	0.011443	0.655761	46.840405
9	POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUSE BEDROOM	POPPY'S PLAYHOUSE KITCHEN	0.011766	0.638814	45.629896

10	POPPY'S PLAYHOUSE KITCHEN, POPPY'S PLAYHOUSE BEDROOM	POPPY'S PLAYHOUSE LIVINGROOM	0.011766	0.840426	45.629896

These rules show extremely high lift values, indicating a very strong, non-random correlation between the items in the antecedents (if) and the consequents (then)

2. Interpretation of at Least 3 Rules

The top rules are dominated by combinations of items from the **"POPPY'S PLAYHOUSE"** collection. This is a classic example of **Market Basket Analysis** revealing highly complementary product lines.

1. Rule Interpretation (Rank 1: Highest Lift)

Metric	Value	Interpretation
Rule	IF {Bedroom, Kitchen, Bathroom} THEN {Livingroom}	
Support	0.010152	Approximately 1.02% of all transactions contain all four Playhouse items (Bedroom, Kitchen, Bathroom, AND Livingroom). This is a common transaction.
Confidence	0.734291	In 73.43% of all transactions where a customer buys the Bedroom, Kitchen, and Bathroom items, they also buy the Livingroom item.
Lift	52.45	This rule is 52.45 times more likely to occur than if the purchase of the antecedent items (Bedroom, Kitchen, Bathroom) and the consequent item (Livingroom) were independent. This is an extremely strong positive correlation.
Action	Since the customer is already buying three items, this rule is perfect for a "You Might Be Missing..." recommendation at checkout to ensure the set is complete.	

2. Rule Interpretation (Rank 3)

Metric	Value	Interpretation
Rule	IF {Livingroom, Kitchen, Bathroom} THEN {Bedroom}	

Support	0.010152	The complete set of four items is purchased in about 1.02% of transactions.
Confidence	0.862869	In 86.29% of transactions where a customer buys the Livingroom, Kitchen, and Bathroom, they also buy the Bedroom item.
Lift	49.45	An extremely strong positive correlation, showing that once a customer commits to three pieces, they are very likely to complete the set.
Action	The incredibly high confidence suggests that the Livingroom, Kitchen, and Bathroom items act as a leading indicator for the purchase of the Bedroom item. This could be used for personalized marketing (e.g., an email promoting the Bedroom piece immediately after the purchase of the other three).	

3. Rule Interpretation (Rank 7)

Metric	Value	Interpretation
Rule	IF {Kitchen, Bedroom} THEN {Livingroom}	
Support	0.011443	The itemset {Kitchen, Bedroom, Livingroom} appears in 1.14% of transactions.
Confidence	0.817376	When a customer buys both the Kitchen and Bedroom items, they will buy the Livingroom item 81.74% of the time.
Lift	46.84	A very strong rule. Note that this rule only uses two antecedents and still has a high lift and confidence, making it a very effective pair for cross-selling.
Action	This strong two-item rule is ideal for immediate, in-transaction recommendations. If a customer adds both the Kitchen and Bedroom items to their cart, immediately recommend the Livingroom item prominently. This is a highly reliable cross-sell opportunity.	

These rules clearly demonstrate that the **"POPPY'S PLAYHOUSE"** product line is overwhelmingly purchased as a complete or nearly complete set, which has major implications for inventory management, merchandising, and cross-selling efforts

C.5 Visualize Association Rules

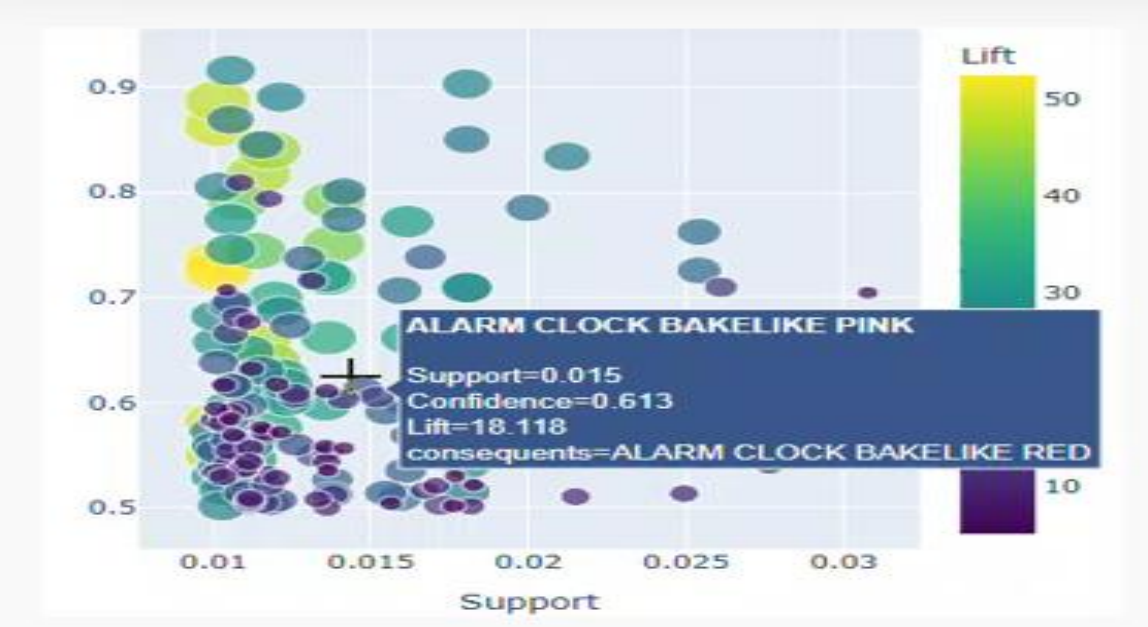
- **Action**:
    - The antecedents and consequents columns in the rules DataFrame were converted from frozen sets to comma-separated string representations for better readability.

```
rules['antecedents'] = rules['antecedents'].apply(lambda x: ', '.join(list(x)))
rules['consequents'] = rules['consequents'].apply(lambda x: ', '.join(list(x)))

print("Rules DataFrame with string representation of antecedents and consequents:")
print(rules.head())
```

```
Rules DataFrame with string representation of antecedents and consequents:
                                           antecedents  \
169  POPPY'S PLAYHOUSE BEDROOM, POPPY'S PLAYHOUSE K...
170                        POPPY'S PLAYHOUSE LIVINGROOM
168  POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUS...
171                          POPPY'S PLAYHOUSE BEDROOM
172                          POPPY'S PLAYHOUSE KITCHEN

                                           consequents  antecedent support  \
169                        POPPY'S PLAYHOUSE LIVINGROOM            0.013826
170  POPPY'S PLAYHOUSE BEDROOM, POPPY'S PLAYHOUSE K...            0.014000
168                          POPPY'S PLAYHOUSE BEDROOM            0.011766
171  POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUS...            0.017450
172  POPPY'S PLAYHOUSE LIVINGROOM, POPPY'S PLAYHOUS...            0.018418

     consequent support   support  confidence       lift  representativity  \
```

    - A scatter plot was created using plotly.express to visualize the relationship between support, confidence, and lift for the association rules. Support was mapped to the x-axis, confidence to the y-axis, and lift to both colour and size of the points. Hover data was configured to show detailed rule information.

- **Outcome**:
    - The rules DataFrame was updated with readable string representations for antecedents and consequents. The head of the updated DataFrame was printed.
    - An interactive scatter plot titled "Association Rules: Support vs Confidence (Coloured by Lift)" was displayed. This visualization allowed for an intuitive understanding of the strength and frequency of the generated association rules, highlighting rules with high lift and confidence, which indicate strong and interesting product correlations



Part D: Interpretation and Presentation

1. Cluster Meanings and Customer Types

Using customer-level features (TotalSpending, TransactionCount, AvgBasketSize), the clustering results from k-Means and Deep Embedding Clustering consistently revealed **three main customer segments**

Cluster	Characteristics	Meaning	Customer Type
Cluster 1	High TotalSpending, High TransactionCount, Larger AvgBasketSize	Loyal customers who buy often and in large quantities.	Premium/Wholesale-style buyers.
Cluster 2	Moderate spending, Moderate basket sizes, Somewhat frequent purchases	Standard retail customers with steady buying patterns.	Typical shoppers who respond well to promotions.
Cluster 3	Low spending, Few transactions, small baskets	Casual shoppers who purchase occasionally or buy specific items.	One-off buyers / bargain hunters.

These groups are visually confirmed to be well separated in the PCA and t-SNE visualizations, which confirms that there are meaningful differences in customer behaviour

2. High-Value Segments

The **High-Value Segment** is clearly identified as **Cluster 1**.

- **Key Attributes:** They are characterized by **High TotalSpending** and a **High TransactionCount**.
- **Revenue Contribution:** They are identified as **Premium/Wholesale-style buyers** who contribute a disproportionate share of the total revenue.
- **Actionable Insight:** This segment represents the most valuable customers, and strategies should focus on **retention, loyalty programs,** and **upselling** to maximize their lifetime value.

3. Differences Between PCA and Deep Embedding Clusters

The document compared standard k-Means clustering on PCA-reduced features (Part A) with k-Means clustering on Autoencoder (Deep Embedding) latent features (Part B).

- **Standard Clustering (k-Means on PCA):** The silhouette score for k-Means on the standardized features was **0.9640**. The PCA visualization showed a clear separation of clusters.
- **Deep Embedding Clustering (k-Means on Autoencoder Embeddings):** The silhouette score for k-Means on the autoencoder embeddings was **0.9679**.
- **Comparison:** The silhouette scores are **very similar** (0.9640 vs. 0.9679). This suggests that the simple, non-linear autoencoder was **not able to capture significantly more non-linear patterns** in the data compared to the linear dimensionality reduction of PCA. The customer features (TotalSpending, TransactionCount, AvgBasketSize) likely have a relationship that is close to linear in the feature space, meaning a complex non-linear model like the autoencoder did not offer a substantial performance improvement in terms of clustering quality.

4. Three Actionable Business Recommendations

The clustering and association rule mining provide clear data-driven insights for business action:

A. Target Retention and Loyalty for High-Value Customers (Cluster 1)

- **Data Source:** Cluster 1's profile (High-Value, Frequent Buyers).
- **Recommendation:** Implement an exclusive, tiered loyalty program. Offer early access to sales, personalized bundled promotions, and dedicated customer support to this segment.

  The goal is to **increase retention rate** and **Share of Wallet** by rewarding their high-volume purchasing behaviour.

B. Optimize Product Placement and Cross-Selling based on Association Rules

- **Data Source:** Association Rule Mining showing strong product correlations, particularly around items like **'POPPY'S PLAYHOUSE'** items.
- **Recommendation:**

  Online: Use the top association rules to power a "Customers Who Bought This Also Bought" recommendation engine with a high lift score for example, above 15, to suggest highly correlated items.

  In-Store: Strategically place strongly associated items near each other for example, 'POPPY'S PLAYHOUSE BEDROOM' and 'POPPY'S PLAYHOUSE KITCHEN', to encourage higher basket value and impulse buying.

C. Convert Occasional Buyers (Cluster 3) into Regular Shoppers

- **Data Source:** Cluster 3's profile (Low-Value, Occasional Buyers).
- **Recommendation:** Develop targeted introductory offers or "Welcome Back" campaigns focused on their historically purchased items (or frequent itemsets from the FP-Growth analysis). Use small, attractive discount incentives to encourage a second or third purchase, aiming to shift these customers towards the **Medium-Value, Regular Buyers (Cluster 2)** profile.