**MARKET BASKET ANALYSIS**

[1]:

*#This is a kaggle notebook.*

**import numpy as np** *# linear algebra*

**import pandas as pd** *# data processing, CSV file I/O (e.g. pd.read\_csv)*

**import os**

**for** dirname, \_, filenames **in** os.walk('/kaggle/input'):

**for** filename **in** filenames: print(os.path.join(dirname, filename))

[2]:

/kaggle/input/market-basket-analysis/Assignment-1\_Data.xlsx

/kaggle/input/market-basket-analysis/Assignment-1\_Data.csv

# Market Basket Analysis Project

## Overview

This notebook is part of a project focused on market basket analysis. We will begin by loading and preprocessing the dataset.

## Dataset Information

The dataset is stored in the file Assignment-1\_Data.xlsx located at

/kaggle/input/market-basket-analysis/. It contains information related to market transac- tions.

## Loading the Dataset

Let’s start by loading the dataset into a DataFrame using pandas.

**import pandas as pd**

*# Load the dataset*

dataset\_path = '/kaggle/input/market-basket-analysis/Assignment-1\_Data.xlsx' df = pd.read\_excel(dataset\_path)

# Initial Exploration

We’ll perform an initial exploration of the dataset to understand its structure and characteristics.

[3]:

*# Display basic information about the dataset* print("Number of rows and columns:", df.shape) print("**\n**Data Types and Missing Values:") print(df.info())

print("**\n**First few rows of the dataset:") print(df.head())

Number of rows and columns: (522064, 7)

Data Types and Missing Values:

<class 'pandas.core.frame.DataFrame'> RangeIndex: 522064 entries, 0 to 522063 Data columns (total 7 columns):

# Column Non-Null Count Dtype

1. BillNo 522064 non-null object
2. Itemname 520609 non-null object
3. Quantity 522064 non-null int64
4. Date 522064 non-null datetime64[ns]
5. Price 522064 non-null float64
6. CustomerID 388023 non-null float64
7. Country 522064 non-null object

dtypes: datetime64[ns](1), float64(2), int64(1), object(3) memory usage: 27.9+ MB

None

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| First few  BillNo | | rows of the dataset:  Itemname | Quantity |  | Date | \ |
| 0 536365 | | WHITE HANGING HEART T-LIGHT HOLDER | 6 | 2010-12-01 | 08:26:00 |  |
| 1 536365 | | WHITE METAL LANTERN | 6 | 2010-12-01 | 08:26:00 |  |
| 2 536365 | | CREAM CUPID HEARTS COAT HANGER | 8 | 2010-12-01 | 08:26:00 |  |
| 3 536365 | | KNITTED UNION FLAG HOT WATER BOTTLE | 6 | 2010-12-01 | 08:26:00 |  |
| 4 536365 | | RED WOOLLY HOTTIE WHITE HEART. | 6 | 2010-12-01 | 08:26:00 |  |
|  | Price | CustomerID Country | | | | |
| 0 | 2.55 | 17850.0 United Kingdom | | | | |
| 1 | 3.39 | 17850.0 United Kingdom | | | | |
| 2 | 2.75 | 17850.0 United Kingdom | | | | |
| 3 | 3.39 | 17850.0 United Kingdom | | | | |
| 4 | 3.39 | 17850.0 United Kingdom | | | | |

[4]:

# Preprocessing

We’ll preprocess the data to ensure it’s ready for analysis.

*#Check Missing Values*

print("Missing Values:")

print(df.isnull().sum())

*#Drop Rows with Missing Values*

df.dropna(inplace=**True**)

[5]:

Missing Values: BillNo 0

Itemname 1455

Quantity 0

Date 0

Price 0

CustomerID 134041

Country 0

dtype: int64

*# Convert dataframe into transaction data*

transaction\_data = df.groupby(['BillNo', 'Date'])['Itemname'].apply(**lambda** x:␣

↪', '.join(x)).reset\_index()

*#Drop Unnecessary Columns*

columns\_to\_drop = ['BillNo', 'Date'] transaction\_data.drop(columns=columns\_to\_drop, inplace=**True**)

*# Save the transaction data to a CSV file* transaction\_data\_path = '/kaggle/working/transaction\_data.csv' transaction\_data.to\_csv(transaction\_data\_path, index=**False**)

[6]:

*# Display the first few rows of the transaction data* print("**\n**Transaction Data for Association Rule Mining:") print(transaction\_data.head())

transaction\_data.shape

Transaction Data for Association Rule Mining:

Itemname

* 1. WHITE HANGING HEART T-LIGHT HOLDER, WHITE META…
  2. HAND WARMER UNION JACK, HAND WARMER RED POLKA DOT
  3. ASSORTED COLOUR BIRD ORNAMENT, POPPY'S PLAYHOU…
  4. JAM MAKING SET WITH JARS, RED COAT RACK PARIS …
  5. BATH BUILDING BLOCK WORD

[6]: (18192, 1)

[7]:

# 

## 4.1 Formatting the transaction data in a suitable format for analysis

Developing the preprocessed data into analysis. Split the ‘Itemname’ column in transaction\_data into individual items using str.split(', ', expand=True).Concatenate the original DataFrame (transaction\_data) with the items DataFrame (items\_df) using pd.concat.Drop the original ‘Itemname’ column since individual items are now in separate columns.Display the resulting DataFrame.

*# Split the 'Itemname' column into individual items*

items\_df = transaction\_data['Itemname'].str.split(', ', expand=**True**)

*# Concatenate the original DataFrame with the new items DataFrame*

transaction\_data = pd.concat([transaction\_data, items\_df], axis=1)

*# Drop the original 'Itemname' column*

transaction\_data = transaction\_data.drop('Itemname', axis=1)

*# Display the resulting DataFrame*

print(transaction\_data.head())

0 1 \

1. WHITE HANGING HEART T-LIGHT HOLDER WHITE METAL LANTERN
2. HAND WARMER UNION JACK HAND WARMER RED POLKA DOT
3. ASSORTED COLOUR BIRD ORNAMENT POPPY'S PLAYHOUSE BEDROOM
4. JAM MAKING SET WITH JARS RED COAT RACK PARIS FASHION
5. BATH BUILDING BLOCK WORD None

2 3 \

1. CREAM CUPID HEARTS COAT HANGER KNITTED UNION FLAG HOT WATER BOTTLE
2. None None
3. POPPY'S PLAYHOUSE KITCHEN FELTCRAFT PRINCESS CHARLOTTE DOLL
4. YELLOW COAT RACK PARIS FASHION BLUE COAT RACK PARIS FASHION
5. None None

4 5 \

1. RED WOOLLY HOTTIE WHITE HEART. SET 7 BABUSHKA NESTING BOXES
2. None None
3. IVORY KNITTED MUG COSY BOX OF 6 ASSORTED COLOUR TEASPOONS
4. None None
5. None None

6 7 \

1. GLASS STAR FROSTED T-LIGHT HOLDER None
2. None None
3. BOX OF VINTAGE JIGSAW BLOCKS BOX OF VINTAGE ALPHABET BLOCKS
4. None None

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 9 … | | | | | 534 | 535 | 536 | \ |
| 0 | None | | None | … | None | None | None | |
| 1 | None | | None | … | None | None | None | |
| 2 HOME | BUILDING BLOCK WORD | | LOVE BUILDING BLOCK WORD | … | None | None | None | |
| 3 | None | | None | … | None | None | None | |
| 4 | None | | None | … | None | None | None | |
| 537 | 538 | 539 540 | 541 542 543 | | | | | |
| 0 None | None | None None | None None None | | | | | |
| 1 None | None | None None | None None None | | | | | |
| 2 None | None | None None | None None None | | | | | |
| 3 None | None | None None | None None None | | | | | |
| 4 None | None | None None | None None None | | | | | |

[8]:

[5 rows x 544 columns]

# Association Rules - Data Mining

## Converting Items to Boolean Columns

To prepare the data for association rule mining, we convert the items in the transaction\_data DataFrame into boolean columns using one-hot encoding. This is achieved through the pd.get\_dummies function, which creates a new DataFrame (df\_encoded) with boolean columns representing the presence or absence of each item.

*# Convert items to boolean columns*

df\_encoded = pd.get\_dummies(transaction\_data, prefix='', prefix\_sep='').

↪groupby(level=0, axis=1).max()

*# Save the transaction data to a CSV file*

df\_encoded.to\_csv('transaction\_data\_encoded.csv', index=**False**)

[9]:

## Association Rule Mining

We apply the Apriori algorithm to perform association rule mining on the encoded transaction data. The min\_support parameter is set to 0.007 to filter out infrequent itemsets. The resulting frequent itemsets are then used to generate association rules based on a minimum confidence threshold of 0.5.Finally, we print the generated association rules.

*# Load transaction data into a DataFrame*

df\_encoded = pd.read\_csv('transaction\_data\_encoded.csv')

**from mlxtend.frequent\_patterns import** apriori, association\_rules

*# Association Rule Mining*

frequent\_itemsets = apriori(df\_encoded, min\_support=0.007, use\_colnames=**True**)

rules = association\_rules(frequent\_itemsets, metric="confidence",␣

↪min\_threshold=0.5)

*# Display information of the rules* print("Association Rules:") print(rules.head())

Association Rules:

antecedents consequents \

1. (CHOCOLATE BOX RIBBONS) (6 RIBBONS RUSTIC CHARM)
2. (60 CAKE CASES DOLLY GIRL DESIGN) (PACK OF 72 RETROSPOT CAKE CASES)
3. (60 TEATIME FAIRY CAKE CASES) (PACK OF 72 RETROSPOT CAKE CASES)
4. (ALARM CLOCK BAKELIKE CHOCOLATE) (ALARM CLOCK BAKELIKE GREEN)
5. (ALARM CLOCK BAKELIKE CHOCOLATE) (ALARM CLOCK BAKELIKE PINK)

antecedent support consequent support support confidence lift \ 0 0.012368 0.039193 0.007036 0.568889 14.515044

1 0.018525 0.054529 0.010059 0.543027 9.958409

2 0.034631 0.054529 0.017315 0.500000 9.169355

3 0.017150 0.042931 0.011379 0.663462 15.454151

4 0.017150 0.032652 0.009125 0.532051 16.294742

leverage conviction zhangs\_metric

|  |  |
| --- | --- |
| 0 0.006551 2.228676 | 0.942766 |
| 1 0.009049 2.068984 | 0.916561 |
| 2 0.015427 1.890941 | 0.922902 |
| 3 0.010642 2.843862 | 0.951613 |
| 4 0.008565 2.067210 | 0.955009 |

[10]:

# Visualization

## Visualizing Market Basket Analysis Results

We use matplotlib and seaborn libraries to create a scatterplot visualizing the results of the market basket analysis. The plot depicts the relationship between support, confidence, and lift for the generated association rules.

**import matplotlib.pyplot as plt import seaborn as sns**

*# Plot scatterplot for Support vs. Confidence*

plt.figure(figsize=(12, 8))

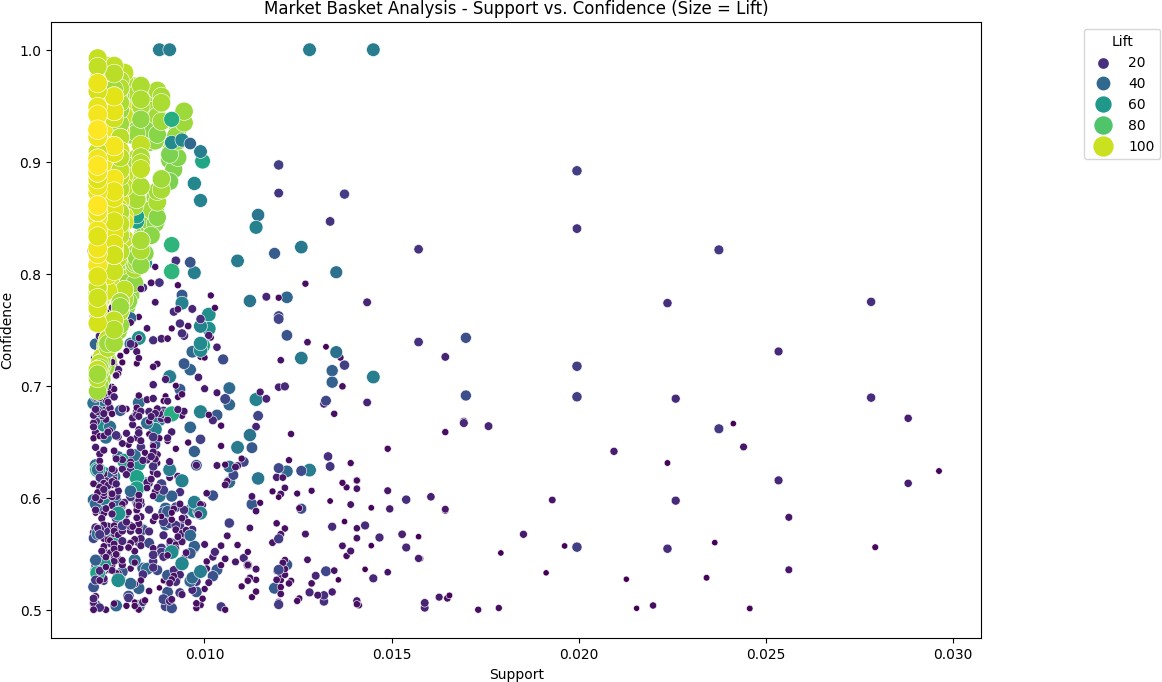
sns.scatterplot(x="support", y="confidence", size="lift", data=rules,␣

↪hue="lift", palette="viridis", sizes=(20, 200))

plt.title('Market Basket Analysis - Support vs. Confidence (Size = Lift)') plt.xlabel('Support')

plt.ylabel('Confidence')

plt.legend(title='Lift', loc='upper right', bbox\_to\_anchor=(1.2, 1)) plt.show()



[11]:

## Interactive Market Basket Analysis Visualization

We leverage the Plotly Express library to create an interactive scatter plot visualizing the results of the market basket analysis. This plot provides an interactive exploration of the relationship between support, confidence, and lift for the generated association rules.

**import plotly.express as px**

*# Convert frozensets to lists for serialization* rules['antecedents'] = rules['antecedents'].apply(list) rules['consequents'] = rules['consequents'].apply(list)

*# Create an interactive scatter plot using plotly express*

fig = px.scatter(rules, x="support", y="confidence", size="lift", color="lift", hover\_name="consequents",

title='Market Basket Analysis - Support vs. Confidence', labels={'support': 'Support', 'confidence': 'Confidence'})

*# Customize the layout*

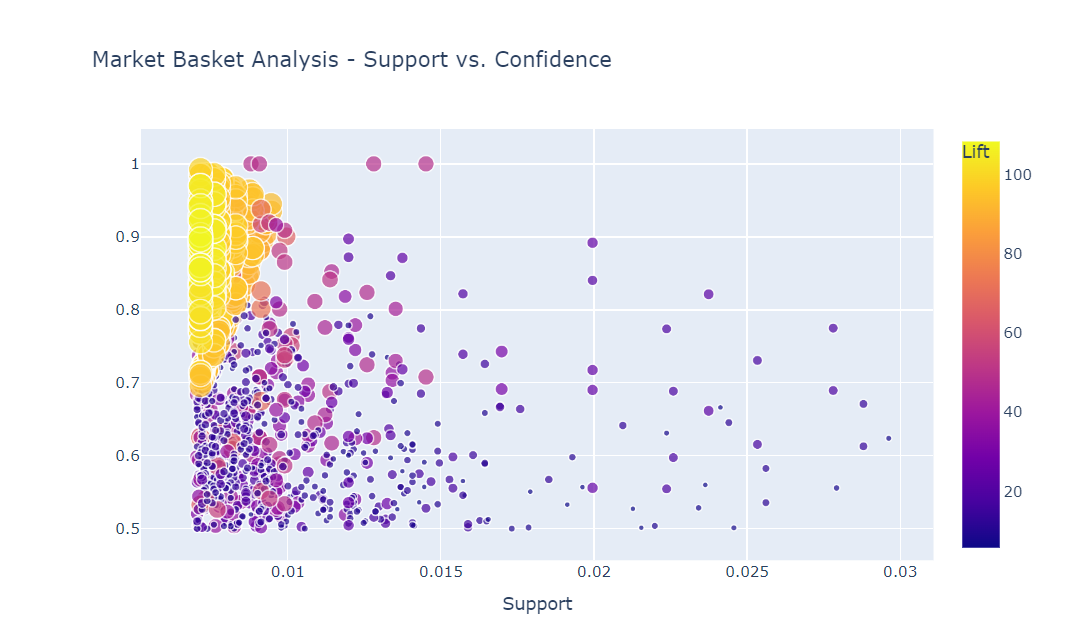
fig.update\_layout( xaxis\_title='Support', yaxis\_title='Confidence', coloraxis\_colorbar\_title='Lift',

showlegend=**True**

)

*# Show the interactive plot*

fig.show()



[12]:

## Interactive Network Visualization for Association Rules

We utilize the NetworkX and Plotly libraries to create an interactive network graph visualizing the association rules. This graph represents relationships between antecedent and consequent items, showcasing support as edge weights.

**import networkx as nx**

**import matplotlib.pyplot as plt import plotly.graph\_objects as go**

*# Create a directed graph*

G = nx.DiGraph()

*# Add nodes and edges from association rules*

**for** idx, row **in** rules.iterrows(): G.add\_node(tuple(row['antecedents']), color='skyblue') G.add\_node(tuple(row['consequents']), color='orange') G.add\_edge(tuple(row['antecedents']), tuple(row['consequents']),␣

↪weight=row['support'])

*# Set node positions using a spring layout*

pos = nx.spring\_layout(G)

*# Create an interactive plot using plotly*

edge\_x = [] edge\_y = []

**for** edge **in** G.edges(data=**True**): x0, y0 = pos[edge[0]]

x1, y1 = pos[edge[1]] edge\_x.append(x0) edge\_x.append(x1) edge\_x.append(**None**) edge\_y.append(y0) edge\_y.append(y1) edge\_y.append(**None**)

edge\_trace = go.Scatter( x=edge\_x, y=edge\_y,

line=dict(width=0.5, color='#888'), hoverinfo='none',

mode='lines')

node\_x = [] node\_y = []

**for** node **in** G.nodes(): x, y = pos[node] node\_x.append(x) node\_y.append(y)

node\_trace = go.Scatter( x=node\_x, y=node\_y, mode='markers', hoverinfo='text', marker=dict(

showscale=**True**, colorscale='YlGnBu', size=10, colorbar=dict(

thickness=15,

title='Node Connections', xanchor='left', titleside='right'

)

)

)

*# Customize the layout*

layout = go.Layout( showlegend=**False**, hovermode='closest', margin=dict(b=0, l=0, r=0, t=0),

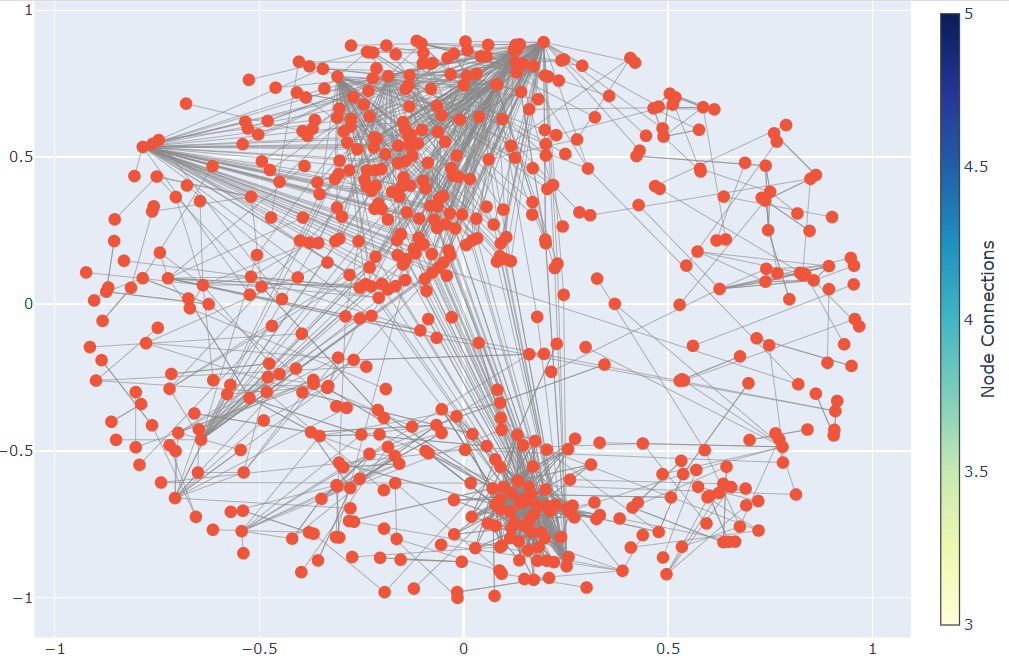
)

*# Create the figure*

fig = go.Figure(data=[edge\_trace, node\_trace], layout=layout)

*# Show the interactive graph*

fig.show()



[13]:

## Interactive Sunburst Chart for Association Rules

We use Plotly Express to create an interactive sunburst chart visualizing association rules. This chart represents the relationships between antecedent and consequent items, showcasing lift as well as support through color intensity.

**import plotly.express as px**

*# Combine antecedents and consequents into a single column for each rule*

rules['rule'] = rules['antecedents'].astype(str) + ' -> ' +␣

↪rules['consequents'].astype(str)

*# Create a sunburst chart*

fig = px.sunburst(rules, path=['rule'], values='lift',

title='Market Basket Analysis - Sunburst Chart', color='support', color\_continuous\_scale='YlGnBu')

*# Customize the layout*

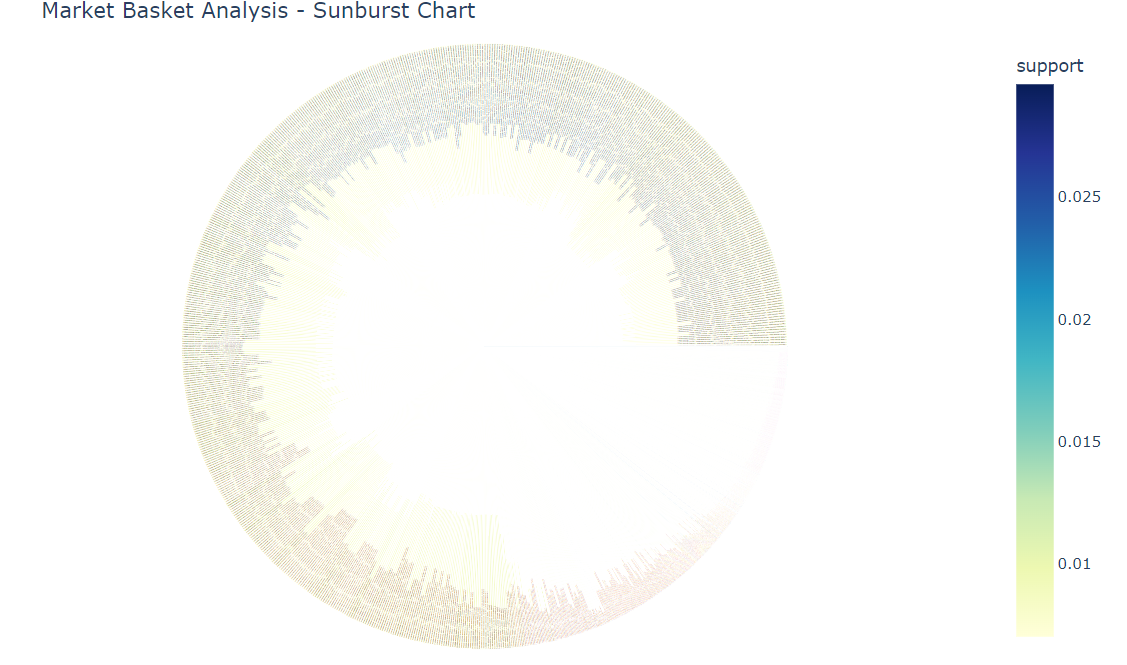
fig.update\_layout(

margin=dict(l=0, r=0, b=0, t=40),

)

*# Show the interactive plot*

fig.show()



Team Members :

1. Muhammed Shafi T

2. Naila PK

3. Nithya V

4. Sreejith A

5. Vipinraj MR