Machine Learning - Prescriptive Modeling

Market Basket Analysis (Customer Purchase Behavior)

What products are frequently bought together?

1. Overview

In today's highly competitive retail environment, understanding customer purchasing behavior is crucial for maximizing sales and enhancing customer satisfaction. This analysis implements Market Basket Analysis (MBA) using transactional data to uncover product combinations that are frequently purchased together. By applying data transformation, grouping, and combinatorial analysis, this approach enables businesses to reveal valuable product affinities that can inform cross-selling strategies, in-store placements, and promotional bundling.

2. Goal

- Identify frequently co-purchased products to uncover hidden patterns in customer behavior.
- Determine the top 2-product and 3-product combinations most often bought together.
- Generate actionable insights that can improve product placement, cross-selling opportunities, and marketing promotions.
- Lay the groundwork for recommendation systems that can personalize customer experiences.

3. Business Challenge

- Lack of visibility into which products drive co-purchases.
- Missed revenue opportunities due to ineffective product bundling.
- Difficulty in designing targeted promotions based on real buying patterns.
- Underperformance of campaigns due to guesswork in product relationships.

4. Methodology

- Filter Transaction Data: Isolate orders that contain multiple items by identifying repeated Order IDs.
- Group Products by Order: Concatenate product names purchased in the same order into a single field for analysis.

- Clean Data: Remove duplicates to ensure accurate pattern detection.
- Apply Combinatorial Analysis: Use the itertools.combinations function to find frequent 2-product and 3-product sets.
- Aggregate & Rank: Use collections.Counter to count and sort the most common combinations.
- Derive Insights: Identify the top 10–20 most frequent item combinations to support marketing and product placement strategies.

Let's take a deep dive into the data

Import necessary libraries

In [10]:
```
import pandas as pd
import os
import glob
```

In [11]:
```
folder_path = r"C:\Monthly_Sales"

# Retrieve all CSV files from the folder using glob
all_files = glob.glob(os.path.join(folder_path, "*.csv"))

# All CSV files combined as one DataFrame
all_data = pd.concat([pd.read_csv(file) for file in all_files], ignore_index=True)

# Merged DataFrame saved into a new CSV
output_file = os.path.join(folder_path, "all_data.csv")
all_data.to_csv(output_file, index=False)

print("All files integrated into:", output_file)
```

```
All files integrated into: C:\Monthly_Sales\all_data.csv
```

Load the updated DataFrame

In [13]:
```
# Skip Blank Rows if present in the dataset

df = pd.read_csv(r'C:\Monthly_Sales\all_data.csv', skip_blank_lines=True)
df.head()
```

Out[13]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address
0	175667	iPhone	1	700.0	04/24/24 19:12	135 Meadow St, Boston, MA 02215
1	175668	AA Batteries (4-pack)	1	5.84	04/20/24 13:45	592 4th St, San Francisco, CA 94016
2	175669	AA Batteries (4-pack)	1	5.84	04/28/24 09:17	632 Park St, Dallas, TX 75001
3	175670	AA Batteries (4-pack)	2	5.84	04/23/24 14:06	131 Pine St, San Francisco, CA 94016
4	175671	Samsung Odyssey Monitor	1	409.99	04/23/24 12:13	836 Forest St, Boston, MA 02215

Data Cleaning Process

Thoroughly clean and standardize the data to eliminate errors, ensure consistency, and build a solid foundation for meaningful insights.

Find and remove rows with NaN values

In [16]:
```
df.isna().sum()
```

Out[16]:
```
Order ID            23744
Product Name        23744
Units Purchased     23746
Unit Price          23746
Order Date          23747
Delivery Address    23748
dtype: int64
```

In [17]:
```
# If Nan value is present in Order ID and Unit Purchased, it will be impossible to
# Therefore, drop Nan values in Order ID and Units Purchased.

df.dropna(subset=['Order ID', 'Units Purchased'], inplace=True)

# Check if Nan value is present
df.isna().sum()
```

Out[17]:
```
Order ID            0
Product Name        0
Units Purchased     0
Unit Price          0
Order Date          1
Delivery Address    2
dtype: int64
```

In [18]:
```
# Further check if any NaN values or blank rows are present
```

```
blank_rows_na = df[df.isnull().any(axis=1)]
blank_rows_na
```

Out[18]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address
2195228	Charging Cable	1	14.95	05/24/24 07:04	852 Hickory St, San Francisco, CA 94016	NaN
3001506	150766	iPhone	1	7	NaN	NaN

Find and remove rows with duplicate values

In [20]:
```
# Find duplicate values

df.duplicated()
```

Out[20]:
```
0          False
1          False
2          False
3          False
4          False
           ...
9144140     True
9144141     True
9144142     True
9144143     True
9144144     True
Length: 9120399, dtype: bool
```

In [21]:
```
# Check again for duplicated values

df.drop_duplicates(inplace = True)

# Check again for duplicated values
df.duplicated()
```

Out[21]:
```
0          False
1          False
2          False
3          False
4          False
           ...
172530     False
2195228    False
3001506    False
6370083    False
6403571    False
Length: 171546, dtype: bool
```

Verify and fix incorrect data types in the dataset

In [23]:
```
# check for data types

df.dtypes
```

Out[23]:
```
Order ID            object
Product Name        object
Units Purchased     object
Unit Price          object
Order Date          object
Delivery Address    object
dtype: object
```

Fix incorrect data types

In [25]:
```
df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%y %H:%M', errors

df['Units Purchased'] = pd. to_numeric(df['Units Purchased'], errors='coerce')

df['Unit Price'] = pd. to_numeric(df['Unit Price'], errors='coerce')
```

In [26]:
```
# Verify the presence of NaN values remaining in the columns as a result of using e

df.isna().sum()
```

Out[26]:
```
Order ID            0
Product Name        0
Units Purchased     1
Unit Price          2
Order Date          3
Delivery Address    2
dtype: int64
```

In [27]:
```
df = df.dropna()
```

Change the data type to optimize memory usage (Optional)

In [29]:
```
df['Order ID'] = pd.to_numeric(df['Order ID'], downcast='integer')
df['Product Name'] = df['Product Name'].astype('category')
df['Units Purchased'] = df['Units Purchased']. astype('int8')
df['Unit Price'] = pd.to_numeric(df['Unit Price'], downcast='float')
df['Delivery Address'] = df['Delivery Address'].astype('category')
```

Expand the dataset with supplementary columns

In [31]:
```
# Add Month and Year
df['Month'] = df['Order Date'].dt.month
df['Year'] = df['Order Date'].dt.year

df
```

Out[31]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Year
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	2024
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	2024
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	2024
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	2024
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	2024
...	...	...	...	...	...	...	...	...
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	2024
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	2024
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	2024
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	2024

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Year
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	2024

171543 rows × 8 columns

In [32]:
```
# Add Total Sales column

df['Total Sales'] = df['Units Purchased'] * df['Unit Price']
df.head()
```

Out[32]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Year	Total Sales
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	2024	700.00000
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	2024	5.84000
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	2024	5.84000
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	2024	11.68000
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	2024	409.98999

Format Unit Price and Total Sales to 2 decimal places

In [34]:
```
df['Unit Price'] = df['Unit Price'].apply(lambda x: "%.2f" % x)

df['Total Sales'] = df['Total Sales'].apply(lambda x: "%.2f" % x)
df.head()
```

Out[34]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Year	Total Sales
0	175667	iPhone	1	700.00	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	2024	700.00
1	175668	AA Batteries (4-pack)	1	5.84	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	2024	5.84
2	175669	AA Batteries (4-pack)	1	5.84	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	2024	5.84
3	175670	AA Batteries (4-pack)	2	5.84	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	2024	11.68
4	175671	Samsung Odyssey Monitor	1	409.99	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	2024	409.99

In [35]:
```
# Convert from string to numeric

df['Unit Price'] = pd.to_numeric(df['Unit Price'])
df['Total Sales'] = pd.to_numeric(df['Total Sales'])
```

In [36]:
```
df.dtypes
```

Out[36]:
```
Order ID                    int32
Product Name             category
Units Purchased              int8
Unit Price                float64
Order Date          datetime64[ns]
Delivery Address         category
Month                       int32
Year                        int32
Total Sales               float64
dtype: object
```

Organize Data by Order Date Chronologically and Reindex

In [38]:
```
df = df.sort_values(by = 'Order Date')

df = df.reset_index(drop=True)
df
```

Out[38]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Year	Tot Sal
0	160155	Alienware Monitor	1	400.99	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	2024	400.
1	151041	AAA Batteries (4-pack)	1	4.99	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	2024	4.
2	146765	AAA Batteries (4-pack)	1	4.99	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	2024	4.
3	145617	Amana Washing Machine	1	600.00	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	2024	600.
4	156535	iPhone	1	700.00	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	2024	700.
...	...	...	...	...	...	...	...	...	
171538	297748	iPhone	1	700.00	2025-01-01 02:37:00	258 Forest St, Los Angeles, CA 90001	1	2025	700.
171539	284606	Bose SoundSport Headphones	1	99.99	2025-01-01 02:50:00	211 Johnson St, Boston, MA 02215	1	2025	99.
171540	302330	AA Batteries (4-pack)	1	5.84	2025-01-01 03:03:00	665 6th St, San Francisco, CA 94016	1	2025	5.
171541	284711	AA Batteries (4-pack)	1	5.84	2025-01-01 03:19:00	250 8th St, San Francisco, CA 94016	1	2025	5.

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Year	Tot Sal
171542	303626	USB-C Charging Cable	3	11.95	2025-01-01 04:43:00	651 Lakeview St, Dallas, TX 75001	1	2025	35.

171543 rows × 9 columns

Top 20 product pairs most frequently ordered together

In [40]:
```
# Filter rows with the same Order ID and add .copy() to avoid settingwithcopyWarnin
df2 = df[df['Order ID'].duplicated(keep=False)].copy()

# Group Product Name with same Order ID
df2['Grouped'] = df2.groupby('Order ID')['Product Name'].transform(lambda x: ','.jo

# Drop any duplicates that may occur as a result of the grouping
df3 = df2[['Order ID', 'Grouped']].drop_duplicates()

df3.head(10)

# Top 20 product pairs most frequently ordered together
from itertools import combinations
from collections import Counter

count = Counter()

for row in df3['Grouped']:
    row_list = row.split(',')
    count.update(Counter(combinations(row_list, 2)))

top_combinations = count.most_common(20)

top_combinations
```

Out[40]:
```
[(('iPhone', 'Lightning Charging Cable'), 473),
 (('USB-C Charging Cable', 'Google Phone'), 469),
 (('Google Phone', 'USB-C Charging Cable'), 465),
 (('Lightning Charging Cable', 'iPhone'), 443),
 (('iPhone', 'Galaxy buds Headphones'), 240),
 (('Galaxy buds Headphones', 'iPhone'), 216),
 (('Google Phone', 'Galaxy buds Headphones'), 195),
 (('Galaxy buds Headphones', 'Google Phone'), 178),
 (('Samsung Galaxy Phone', 'USB-C Charging Cable'), 177),
 (('iPhone', 'Apple Airpods Headphones'), 173),
 (('USB-C Charging Cable', 'Samsung Galaxy Phone'), 168),
 (('Apple Airpods Headphones', 'iPhone'), 147),
 (('Bose SoundSport Headphones', 'Google Phone'), 111),
 (('Google Phone', 'Bose SoundSport Headphones'), 94),
 (('USB-C Charging Cable', 'Galaxy buds Headphones'), 86),
 (('Galaxy buds Headphones', 'USB-C Charging Cable'), 82),
 (('Galaxy buds Headphones', 'Samsung Galaxy Phone'), 69),
 (('Lightning Charging Cable', 'Galaxy buds Headphones'), 66),
 (('Galaxy buds Headphones', 'Lightning Charging Cable'), 66),
 (('Samsung Galaxy Phone', 'Galaxy buds Headphones'), 65)]
```

Top 10 sets of three (3) products most commonly ordered together

In [42]:
```
from itertools import combinations
from collections import Counter

count = Counter()

for row in df3['Grouped']:
    row_list = row.split(',')
    count.update(Counter(combinations(row_list, 3)))

# Collect top 10 most common combinations
top_combinations = count.most_common(10)

# Display results
top_combinations
```

Out[42]:
```
[(('Google Phone', 'USB-C Charging Cable', 'Galaxy buds Headphones'), 22),
 (('iPhone', 'Lightning Charging Cable', 'Galaxy buds Headphones'), 17),
 (('Galaxy buds Headphones', 'USB-C Charging Cable', 'Google Phone'), 17),
 (('iPhone', 'Lightning Charging Cable', 'Apple Airpods Headphones'), 15),
 (('Galaxy buds Headphones', 'Lightning Charging Cable', 'iPhone'), 11),
 (('Google Phone', 'USB-C Charging Cable', 'Bose SoundSport Headphones'), 10),
 (('Lightning Charging Cable', 'iPhone', 'Galaxy buds Headphones'), 10),
 (('iPhone', 'Galaxy buds Headphones', 'Lightning Charging Cable'), 10),
 (('USB-C Charging Cable', 'Google Phone', 'Galaxy buds Headphones'), 9),
 (('Bose SoundSport Headphones', 'USB-C Charging Cable', 'Google Phone'), 9)]
```

Key Insights

1. Strong Accessory Dependency:

- iPhone is frequently bought with the Lightning Charging Cable (473 times) and Apple AirPods Headphones (173 times), confirming a strong dependency on accessories.
- Similar behavior observed for Google Phone and USB-C Charging Cable (469 times), and Samsung Galaxy Phone with the same cable (177 times).

2. Bundling Behavior Around Devices:

- Buyers often purchase phones with multiple accessories (e.g., Google Phone + USB-C Cable + Headphones combinations).
- Example: Google Phone + USB-C Charging Cable + Bose SoundSport Headphones is a top 3-product combo.

3. Cross-Brand Headphone Love:

- Galaxy Buds Headphones appear frequently with iPhone, Google Phone, and Samsung Galaxy Phone, suggesting customers aren't loyal to brand-specific headphones. They mix and match based on price, design, or preference.

4. Repetitive Combination Patterns:

- Most of the high-frequency combinations are symmetrical (e.g., iPhone + Lightning Cable and Lightning Cable + iPhone)—reinforcing the reliability of the insight.

5. Accessory-Driven Purchases:

- Accessories like charging cables and headphones dominate the co-purchase behavior. These aren’t just afterthoughts—they're clearly planned purchases.

Strategic Recommendations

1. Create Product Bundles:

- Offer ready-made bundles (Example, iPhone + Lightning Cable + AirPods) with small discounts to promote upselling and convenience.

2. Optimize Online Store Suggestions:

- Implement "Frequently Bought Together" suggestions at checkout using the most frequent 2- and 3-product combinations.

3. Improve In-Store Product Placement:

- Place high-affinity products (Example, USB-C Cable and Google Phone) adjacent on shelves to encourage impulse add-ons.

4. Run Targeted Promotions:

- Create cross-promotion campaigns such as "Buy a Google Phone and get 20% off USB-C Cable and Galaxy Buds."

5. Inventory Forecasting:

- Forecast accessory demand based on device sales, as customers are highly likely to buy them together.