Reordering Assistant Report

1. Data Collection

Task: Gather or simulate user order data and product information.

Methodology & Implementation:

- Instead of generating synthetic data, a dummy e-commerce dataset was downloaded from Kaggle.
- The dataset contained realistic order details such as:
    - user_id, order_id, product_id, product_name, category, order_date, quantity.
- This dataset served as a representative sample of user purchase history for training and testing recommendation models.

Tools Used: Kaggle (for dataset), Python, Pandas.

Challenges:

- Dataset cleaning required because Kaggle data often contains inconsistencies, missing values, and duplicates.
- Mapping dataset fields to project requirements (ensuring column names matched with ML pipeline).

Outcome:

Final CSV dataset successfully collected and validated with the filename **`amazon_preprocessed_first_100.csv`**. The dataset was directly used in preprocessing for building recommendation and bundling models.

2. Data Preprocessing

Task: Prepare data for ML model training.

Methodology & Implementation:

- The dataset **`amazon_preprocessed_first_100.csv`** (collected in Task 1) was used as input.
- **Cleaning:**
    - Removed duplicate entries.
    - Handled missing values to avoid bias in the model.

- **Encoding:**
    - Transformed categorical features such as `category` into numerical form using label encoding/one-hot encoding.

- **Feature Engineering:**

  Derived additional features such as:

    - **Order frequency** per user.
    - **Product associations** to capture co-purchase behavior.

Tools Used: Pandas, NumPy, Scikit-learn.

Challenges:

- Dealing with inconsistent or missing values in the Kaggle dataset.
- Choosing the right encoding strategy without losing interpretability.
- Ensuring that the preprocessed dataset was compatible with both recommendation and bundling models.

Outcome:
A clean and preprocessed dataset was successfully generated from **`amazon_preprocessed_first_100.csv`**, ready for machine learning model training.

3. ML Model for Order Recommendation

Task: Recommend products for reordering based on past orders.

Methodology & Implementation:

- The preprocessed dataset (**`amazon_preprocessed_first_100.csv`**) was used as input.
- A **collaborative filtering approach** was applied using matrix factorization (SVD).

- Steps:
    1. Uploaded and read the dataset into Google Colab.
    2. Cleaned column names (removed hidden spaces).
    3. Extracted dataset info (rows, columns, unique users).
    4. Ensured that each user receives **exactly 10 product recommendations**.
    5. Retrieved and displayed top-10 recommendations for a selected user (`AE3CFONNMANNC5QPYIAXV67EUYUQ`).

Tools Used: Scikit-learn, Surprise (for recommendation), Pandas, Google Colab.

Challenges:

- Handling the cold-start problem for new users/products without history.
- Ensuring fair distribution so that each user gets a fixed number of recommendations.

Outcome:

- A total of **1,040 recommendations** were generated for **104 unique users**.
- Each user successfully received **10 personalized recommendations**.
- Example (for user `AE3CFONNMANNC5QPYIAXV67EUYUQ`):

Rank	Product ID	Score
1	B09Q5SWVBJ	1.51e-15
2	B07JH1C41D	6.99e-16
3	B08Y1TFSP6	6.34e-16
4	B082LZGK39	6.25e-16
5	B09NKZXMWJ	6.19e-16

Final recommendations were saved in **recommendations_top10.csv**, which contains four columns:

- user_id, product_id, rank, score.

Code:

```
# Step 1: Upload the file

from google.colab import files

import pandas as pd



uploaded = files.upload()

file_name = list(uploaded.keys())[0]  # uploaded file ka naam auto le
lo



# Step 2: Read CSV & clean column names

df = pd.read_csv(file_name)

df.columns = df.columns.str.strip()  # hidden spaces remove



# Step 3: Basic info
```

```python
print("Total Rows:", df.shape[0])

print("Total Columns:", df.shape[1])

print("Columns:", df.columns.tolist())

print("\nUnique Users:", df['user_id'].nunique())


# Step 4: Check each user has 10 recommendations

rec_per_user = df.groupby('user_id')['product_id'].count().describe()

print("\nRecommendations per User:\n", rec_per_user)


# Step 5: Example - Show recommendations for a specific user

user_id = "AE3CFONNMANNC5QPYIAXV67EUYUQ"  # apna user_id yahan dal do

user_recs = df[df['user_id'] == user_id].sort_values(by="rank")

if not user_recs.empty:

    print("\nTop 10 Recommendations for", user_id)

    print(user_recs)

else:

    print("\nNo recommendations found for", user_id)
```

4. ML Model for Smart Bundling

Task: Suggest product bundles to enhance reordering.

Methodology & Implementation:

- Used **Apriori algorithm** to identify frequent co-purchased items and generate bundling rules.
- Steps followed:
    1. Uploaded and cleaned dataset (orders_clean_preprocessed.csv).
    2. Filtered out rare products (kept only those appearing in ≥5 orders).

3. Converted order history into **transaction format**.
4. Applied **one-hot encoding** for basket analysis.
5. Ran Apriori with **min_support = 0.005** to find frequent itemsets.
6. Generated **association rules** with minimum confidence = 0.2.
7. Selected **top bundles** using lift ≥ 1.1 (better than random chance).
8. Exported final deliverables for frequent itemsets, association rules, and top bundles.

Tools Used: MLxtend, Pandas, NumPy, Google Colab.

Challenges:

- Very sparse dataset: most products appear in few orders.
- Normalizing product names (typo/case mismatches).
- Balancing between high support vs. meaningful lift values.

Outcome:

- Dataset processed:
  - **Rows:** 904
  - **Transactions (orders):** 816
  - **Unique items kept:** 100

- **Frequent Itemsets:** Found 100+ itemsets, including single products and combinations.
- **Association Rules:** Generated with strong lift values (up to **46x better than random**).
- **Top Bundles (Pairs):** Example rules:

Antecedent (If bought)	Consequent (Also buy)	Support Count
Wayona Nylon Braided 3A Lightning Cable	Wayona Nylon Braided USB to Lightning Cable	5
Flix USB Type-C Data Cable	Flix Micro USB Cable	6
Amazon Basics HDMI Cable (6 Feet, Black)	Amazon Basics HDMI Cable (6 Feet, 2-Pack)	5
Acer 80 cm Android TV	Acer 127 cm 4K Ultra HD TV	6

Deliverables generated:

- **frequent_itemsets.csv** → All frequent itemsets with support values.
- **association_rules.csv** → All rules with support, confidence, lift, leverage, conviction.
- **bundles_top_pairs.csv** → Best 1→1 bundle suggestions.
- **bundles_top_triplets.csv** → Best 2→1 bundle suggestions.

Code:

```
# ==============================================

# STEP 0: Setup (Install + Imports)

# ==============================================

!pip install mlxtend --quiet


import pandas as pd

import numpy as np

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent_patterns import apriori, association_rules

from google.colab import files



# ==============================================

# STEP 1: Upload and Load Orders Dataset

# ==============================================

uploaded = files.upload()  # Upload "orders_clean_preprocessed.csv"

file_name = list(uploaded.keys())[0]  # Automatically get uploaded file
name

df = pd.read_csv(file_name)



# ==============================================

# STEP 2: Define Important Columns

# ==============================================

ORDER_COL = 'order_id'       # Each order

ITEM_COL  = 'product_name'   # Product name in that order
```

```
# Keep only required columns & clean data
df = df[[ORDER_COL, ITEM_COL]].dropna().astype(str)


print(f"✅ Rows: {len(df)}")
print(f"✅ Columns: {df.columns.tolist()}")
df.head()


# ============================================
# STEP 3: Filter Rare Products (Optional)
# ============================================
MIN_ITEM_FREQ = 5   # Products appearing in less than 5 orders will be dropped
item_counts = df[ITEM_COL].value_counts()
keep_items = item_counts[item_counts >= MIN_ITEM_FREQ].index
df = df[df[ITEM_COL].isin(keep_items)].copy()


print(f"✅ Unique items kept: {df[ITEM_COL].nunique()}")


# ============================================
# STEP 4: Create Transactions List
# ============================================
transactions = (
    df.groupby(ORDER_COL)[ITEM_COL]
      .apply(lambda x: sorted(set(x)))  # Remove duplicates in same order
```

```
    .tolist()
)

print(f"✅ Total transactions: {len(transactions)}")
print(transactions[:3])  # Peek at first 3

# ============================================
# STEP 5: One-Hot Encode Transactions
# ============================================
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
basket_df = pd.DataFrame(te_array, columns=te.columns_).astype(bool)

print(f"✅ Basket shape: {basket_df.shape}")

# ============================================
# STEP 6: Run Apriori Algorithm
# ============================================
MIN_SUPPORT = 0.005  # Itemset appears in at least 2% of orders
freq_itemsets = apriori(
    basket_df,
    min_support=MIN_SUPPORT,
    use_colnames=True,
    low_memory=True
)
```

```
freq_itemsets['itemset_len'] = freq_itemsets['itemsets'].apply(len)

freq_itemsets = freq_itemsets.sort_values(['itemset_len','support'],
ascending=[True, False]).reset_index(drop=True)


print("✅ Top Frequent Itemsets:")

print(freq_itemsets.head(10))


# ============================================

# STEP 7: Generate Association Rules

# ============================================

MIN_CONF = 0.2  # Minimum confidence 30%

rules = association_rules(freq_itemsets, metric='confidence',
min_threshold=MIN_CONF)


rules['antecedent_len'] = rules['antecedents'].apply(len)

rules['consequent_len'] = rules['consequents'].apply(len)

rules['support_count']  = (rules['support'] *
len(transactions)).round().astype(int)


# Helper function to make frozensets readable

def fs_to_str(fs): return ", ".join(sorted(list(fs)))


rules_tidy = rules.assign(

    antecedents_str = rules['antecedents'].apply(fs_to_str),

    consequents_str = rules['consequents'].apply(fs_to_str)
```

```
)


print("✅ Top Rules:")

print(rules_tidy[['antecedents_str','consequents_str','support','suppor
t_count','confidence','lift']].head(10))


# ============================================

# STEP 8: Select Top Bundles (Smart Bundling)

# ============================================

MIN_LIFT = 1.1  # Better than random chance


bundle_rules = rules_tidy.query("consequent_len == 1 and antecedent_len
>= 1 and lift >= @MIN_LIFT")


# Top pairs (1 item → 1 more)

top_pairs = (bundle_rules.query("antecedent_len == 1")

             .sort_values(['lift','confidence','support'],
ascending=False)

             .head(50))


# Top triplets (2 items → 1 more)

top_triplets = (bundle_rules.query("antecedent_len == 2")

                .sort_values(['lift','confidence','support'],
ascending=False)

                .head(50))


print("✅ Top Bundles (Pairs):")
```

```
print(top_pairs[['antecedents_str','consequents_str','support_count','c
onfidence','lift']].head(10))


# ============================================

# STEP 9: Export Deliverables

# ============================================

# Frequent Itemsets

freq_export = freq_itemsets.assign(

    itemsets_str=freq_itemsets['itemsets'].apply(lambda s: ",
".join(sorted(list(s))))

freq_export[['itemsets_str','support','itemset_len']].to_csv("/content/
frequent_itemsets.csv", index=False)


# All Association Rules

rules_export =
rules_tidy[['antecedents_str','consequents_str','support','support_coun
t','confidence','lift','leverage','conviction','antecedent_len','conseq
uent_len']]

rules_export.to_csv("/content/association_rules.csv", index=False)


# Top Pairs & Triplets

top_pairs[['antecedents_str','consequents_str','support_count','confide
nce','lift']].to_csv("/content/bundles_top_pairs.csv", index=False)

top_triplets[['antecedents_str','consequents_str','support_count','conf
idence','lift']].to_csv("/content/bundles_top_triplets.csv",
index=False)


print("✅ Saved Deliverables:")

print("frequent_itemsets.csv")
```

```python
print("association_rules.csv")

print("bundles_top_pairs.csv")

print("bundles_top_triplets.csv")



# ============================================

# STEP 10: (Optional) Product-wise Suggestions

# ============================================

def suggest_addons_for(product, top_n=5, min_lift=1.1):

    out =
(bundle_rules[bundle_rules['antecedents_str'].str.contains(rf'\b{pd.util.escape_regex(product)}\b')]

           .query("lift >= @min_lift")

           .sort_values(['lift','confidence','support'],
ascending=False)


[['antecedents_str','consequents_str','support_count','confidence','lift']]

           .head(top_n))

    return out



# Example usage:

# suggest_addons_for("Bread", top_n=5)
```

6. Testing

Task: Validate recommendation accuracy, bundling relevance, and usability.

Methodology & Implementation:
Testing was performed in two major parts:

Part 1: Recommendation Accuracy Test

- Uploaded recommendations_top10.csv (model predictions) and orders_clean_preprocessed.csv (ground truth).
- Evaluated performance using Precision, Recall, and F1-score per user.
- Used scikit-learn for metrics calculation.
- Saved user-level results in part1_accuracy_report.csv.

Results (Part 1):

- Recommendations file: 1040 rows
- Orders file: 904 rows
- Overall averages:
  - Precision: 0.0317
  - Recall: 0.1218
  - F1 Score: 0.0497

🔎 Interpretation:

- The recommendation system is retrieving some correct products, but precision and recall are relatively low.
- Indicates that further tuning (e.g., hybrid models, re-ranking) can improve accuracy.

```
# ==============================

# Part 1: Recommendation Accuracy Test (Real Data)

# Precision / Recall / F1 Score

# ==============================



# Step 1: Install required libs

!pip install scikit-learn --quiet



# Step 2: Import libraries

import pandas as pd

import numpy as np

from sklearn.metrics import precision_score, recall_score, f1_score
```

```
from google.colab import files


print("Upload your recommendations_top10.csv file")
uploaded = files.upload()  # Upload recommendations file
recommendations_file = list(uploaded.keys())[0]


print("\nUpload your orders_clean_preprocessed.csv file")
uploaded = files.upload()  # Upload orders file
orders_file = list(uploaded.keys())[0]


# Step 3: Load files (upload in Colab before running)
recommendations = pd.read_csv("recommendations_top10.csv")   # Model
output
orders = pd.read_csv("orders_clean_preprocessed.csv")        # Ground
truth


print("Recommendations file:", recommendations.shape)
print("Orders file:", orders.shape)


# Step 4: Ensure columns exist
# recommendations file should have: user_id, product_id
# orders file should have: user_id, product_id
assert "user_id" in recommendations.columns
assert "product_id" in recommendations.columns
assert "user_id" in orders.columns
assert "product_id" in orders.columns
```

```python
# Step 5: Evaluate precision/recall for each user
metrics = []
for user in recommendations["user_id"].unique():
    rec_items = set(recommendations[recommendations["user_id"]==user]["product_id"].tolist())
    true_items = set(orders[orders["user_id"]==user]["product_id"].tolist())

    if not true_items:  # skip users without ground truth
        continue

    all_items = sorted(list(rec_items | true_items))
    y_true = [1 if item in true_items else 0 for item in all_items]
    y_pred = [1 if item in rec_items else 0 for item in all_items]

    precision = precision_score(y_true, y_pred, zero_division=0)
    recall = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)

    metrics.append([user, precision, recall, f1])

# Step 6: Create report
report = pd.DataFrame(metrics, columns=["user_id","Precision","Recall","F1"])
print("\n=== Overall Averages ===")
```

```python
print("Precision:", report["Precision"].mean())
print("Recall    :", report["Recall"].mean())
print("F1 Score :", report["F1"].mean())


# Step 7: Save detailed report
report.to_csv("part1_accuracy_report.csv", index=False)
print("\n✅ Saved: part1_accuracy_report.csv")
```

```
=== Overall Averages ===
Precision: 0.03173076923076923
Recall    : 0.12179487179487178
F1 Score : 0.04969709777402084
```

Part 2: Bundle Relevance Test

- Uploaded association rule outputs:
  - `association_rules.csv`
  - `frequent_itemsets.csv`
  - `bundles_top_pairs.csv`
  - `bundles_top_triplets.csv`
- Tested rules against thresholds:
  - min_support = 0.005
    min_confidence = 0.2
  - min_lift = 1.0
- Filtered valid vs. failed rules.

- Saved reports:
    - `part2_valid_rules_report.csv`
    - `part2_failed_rules_report.csv`

Results (Part 2):

- Association Rules: 16 total
- Frequent Itemsets: 108
- Top Pairs: 16
- Top Triplets: 0
- Testing summary:
    - Passed (Valid): 16
    - Failed: 0

🔎 Interpretation:

- All generated rules met minimum thresholds for support, confidence, and lift.
- Example:
    - *Acer 80 cm TV → Acer 127 cm 4K TV* (Confidence: 0.60, Lift: 34.97)
    - *Flix Type-C Cable → Flix Micro USB Cable* (Confidence: 0.67, Lift: 45.33)
- Indicates strong and reliable bundle recommendations.

Tools Used:

- Model testing: Scikit-learn, Pandas
- Bundling test: MLxtend, Pandas
- Environment: Google Colab

Roles:

- AI Developer → Model & Bundling testing
- Web Developer → Web app functionality (not covered here)
- App Developer → Cross-device testing (not covered here)

Outcome:

- Part 1: Model recommendations evaluated, baseline performance established.
- Part 2: Bundling rules validated, all rules passed quality thresholds.
- Generated detailed reports for further analysis and debugging.

```
# ==============================

# Part 2: Bundle Relevance Test

# ==============================
```

```
import pandas as pd

from google.colab import files



# Step 1: Upload association rules + frequent itemsets

print("Upload association_rules.csv")

uploaded = files.upload()

assoc_file = list(uploaded.keys())[0]



print("\nUpload frequent_itemsets.csv")

uploaded = files.upload()

freq_file = list(uploaded.keys())[0]



# Optional: upload top bundles

print("\nUpload bundles_top_pairs.csv")

uploaded = files.upload()

pairs_file = list(uploaded.keys())[0]



print("\nUpload bundles_top_triplets.csv")

uploaded = files.upload()

triplets_file = list(uploaded.keys())[0]



# Step 2: Load files

association_rules = pd.read_csv(assoc_file)

frequent_itemsets = pd.read_csv(freq_file)

top_pairs = pd.read_csv(pairs_file)
```

```python
top_triplets = pd.read_csv(triplets_file)


print("Association Rules:", association_rules.shape)

print("Frequent Itemsets:", frequent_itemsets.shape)

print("Top Pairs:", top_pairs.shape)

print("Top Triplets:", top_triplets.shape)


# Step 3: Check support/confidence/lift distribution

print("\n=== Association Rules (Sample) ===")

print(association_rules.head())


# Step 4: Define thresholds (testing criteria)

min_support = 0.005

min_confidence = 0.2

min_lift = 1.0


# Step 5: Filter valid rules

valid_rules = association_rules[

    (association_rules["support"] >= min_support) &

    (association_rules["confidence"] >= min_confidence) &

    (association_rules["lift"] >= min_lift)

]


# Step 6: Testing summary

total_rules = association_rules.shape[0]
```

```python
passed = valid_rules.shape[0]
failed = total_rules - passed


print("\n=== Testing Summary ===")
print("Total Rules    :", total_rules)
print("Passed (Valid):", passed)
print("Failed         :", failed)


# Step 7: Save detailed report
valid_rules.to_csv("part2_valid_rules_report.csv", index=False)
print("\n✅ Saved: part2_valid_rules_report.csv")


# (Optional) Also save failed rules for debugging
failed_rules = association_rules.drop(valid_rules.index)
failed_rules.to_csv("part2_failed_rules_report.csv", index=False)
print("✅ Saved: part2_failed_rules_report.csv")
```

```
                                    consequents_str   support  
support_count  \
0  Acer 127 cm (50 inches) I Series 4K Ultra HD A...  0.007353  
6
1  Acer 80 cm (32 inches) I Series HD Ready Andro...  0.007353  
6
2        Flix Micro Usb Cable For Smartphone (Black)  0.007353  
6
3  Flix (Beetel) Usb To Type C Pvc Data Sync And ...  0.007353  
6
```

```
4  OnePlus 108 cm (43 inches) Y Series 4K Ultra H...  0.007353
6



   confidence       lift  leverage  conviction  antecedent_len
consequent_len

0    0.600000  34.971429  0.007143    2.457108               1
1

1    0.428571  34.971429  0.007143    1.728554               1
1

2    0.666667  45.333333  0.007191    2.955882               1
1

3    0.500000  45.333333  0.007191    1.977941               1
1

4    0.461538  34.237762  0.007138    1.832108               1
1



=== Testing Summary ===

Total Rules   : 16

Passed (Valid): 16

Failed        : 0
```

Report

Approach:

Our project followed a structured approach to build and evaluate a recommendation system. We started by preparing and cleaning the dataset to ensure accuracy and consistency. The main focus was on extracting meaningful features such as order frequency and product associations. A machine learning pipeline was then designed to train models for generating recommendations. Finally, evaluation metrics like precision, recall, and F1 score were used to measure the system's effectiveness.

Implementation:

- **Data Preprocessing:** We handled missing values, removed duplicates, and encoded categorical features to make the dataset suitable for training.

- **Feature Engineering:** Derived additional features like order frequency and product relationships to improve model performance.
- **Model Training:** Machine learning models were trained on the cleaned dataset to generate product recommendations.
- **Evaluation:** Using scikit-learn, we calculated precision, recall, and F1 scores to assess the accuracy of recommendations.
- **Tools Used:** Python, Pandas, NumPy, Scikit-learn, Google Colab.

Challenges:

- Ensuring data quality was a challenge due to missing values and duplicates.
- Feature selection required careful consideration to avoid overfitting and to ensure relevant recommendations.
- Achieving a balance between precision and recall was difficult, as improving one often reduced the other.
- Computational limitations in training larger models within the available environment.

Outcomes

- A functional recommendation system was developed with measurable accuracy.
- Overall averages achieved:
    - Precision: 0.0317
    - Recall: 0.1217
    - F1 Score: 0.0497
- The project successfully demonstrated the end-to-end process of building, training, and evaluating a machine learning model for recommendations.
- Documentation was created to record methodology, tools, challenges, and results for future improvements.