### ****Epic: Analyze and Optimize App Card Exposure and Cross-Sell Opportunities****

#### ****Ticket 1: Analyze Card Exposure on App (Reinforced Learning vs BAU Logic)****

**Description:**  
Analyze which cards users are exposed to when they visit the app. Break down by card category, reinforced learning limb, and business-as-usual (BAU) logic.

**Tasks:**

* Gather app card exposure data from logs or a database.
* Categorize the cards based on RL or BAU logic.
* Create a visualization/dashboard for easy breakdown.

**Story Points:** 5  
**Priority:** High

#### ****Ticket 2: Assess Traceability of Assigned Limbs****

**Description:**  
Analyze the limbs (RL/BAU) users were assigned to and the effect on their purchase behavior via the springboard app. Develop traceability tables to connect assignments to purchase outcomes.

**Tasks:**

* Extract limb assignment and purchase data.
* Build traceability tables (CNBA-focused).
* Summarize findings on causal relationships.

**Story Points:** 8  
**Priority:** High

#### ****Ticket 3: Evaluate Confounding Variables in Cross-Sell Conversions****

**Description:**  
Identify and control for confounding variables affecting RL limb and BAU performance for cross-sell. Perform causal modeling to validate the impact.

**Tasks:**

* Identify key variables and relationships.
* Run statistical tests to check for confounders.
* Implement causal modeling techniques.

**Story Points:** 8  
**Priority:** Medium

#### ****Ticket 4: Analyze Pricing Policy Effects on Broadband Cross-Sell****

**Description:**  
Examine how different pricing policies (TNBA) influence broadband cross-sell conversion rates.

**Tasks:**

* Collect data on pricing models and broadband conversions.
* Segment users by pricing experience (e.g., TNBA vs CNBA).
* Test for significant relationships and create insights.

**Story Points:** 8  
**Priority:** Medium

#### ****Ticket 5: Link Digital Touchpoints to Cross-Sell Outcomes****

**Description:**  
Investigate how digital touchpoints (e.g., cards shown via CNBA) and pricing models combine to influence cross-sell outcomes.

**Tasks:**

* Map digital touchpoints and pricing models to customer journeys.
* Analyze combined effects of cards, digital experience, and pricing on conversions.
* Provide actionable recommendations.

**Story Points:** 13  
**Priority:** High

#### ****Ticket 6: Create SQL-Based Conversion Metrics Dashboard****

**Description:**  
Develop SQL queries to automate tracking and reporting of broadband cross-sell conversion metrics.

**Tasks:**

* Design SQL queries to calculate key metrics (e.g., CTR, conversion rate).
* Build a dashboard for ongoing reporting.
* Validate data accuracy with the team.

**Story Points:** 8  
**Priority:** Medium

#### ****Ticket 7: Conduct Randomized Experiments on Cross-Sell Outcomes****

**Description:**  
Design and implement randomized experiments to measure causal effects of RL/BAU limbs on cross-sell behavior.

**Tasks:**

* Develop experimental design (e.g., A/B testing).
* Assign user groups and ensure randomization.
* Analyze results and validate findings.

**Story Points:** 13  
**Priority:** High

### Task 1: ****Analyze App Card Exposure****

**Goal:** Identify the cards shown to users based on reinforced learning or business-as-usual logic.

**Relevant Columns:**

* request\_time (TIMESTAMP): Timestamp of when the recommendation was requested.
* request\_channel (STRING): Channel where the recommendation was displayed.
* request\_section (STRING): Section (e.g., broadband, add-ons) relevant to the card shown.
* experiment\_id (STRING): Identifies the experiment (e.g., RL or BAU).
* experiment\_limb\_id (STRING): Experiment limb for RL or BAU exposure.
* pricing\_strategy (STRING): Logic behind the pricing shown for the recommendation.

### Task 2: ****Assess Limbs Assigned and Traceability****

**Goal:** Understand user assignments to RL/BAU and analyze purchase behavior via the springboard app.

**Relevant Columns:**

* household\_mrc\_avg (FLOAT): Average household revenue, indicating value segment.
* experiment\_id (STRING): Links to the limb assigned.
* experiment\_limb\_id (STRING): Identifies the treatment/control group.
* product\_name (STRING): Name of the recommended product.
* action\_id (STRING): Action identifier for traceability.
* add\_to\_cart\_flag (INTEGER): Indicates if the product was added to the cart.
* recommended\_flag (INTEGER): Flags if the action was recommended.
* session\_id (STRING): Unique identifier for the session to track purchases.

### Task 3: ****Control Confounding Variables****

**Goal:** Check and control for variables affecting performance.

**Relevant Columns:**

* household\_region (STRING): Regional segmentation for users.
* household\_social\_grade (STRING): Social grade classification.
* request\_journey (STRING): Customer journey type (e.g., regrade, acquisition).
* customer\_limb\_detail (STRING): Detailed description of RL or BAU model used.
* prev\_service\_product\_revenue (NUMERIC): Baseline revenue for analysis.
* curr\_service\_product\_revenue (NUMERIC): Current revenue after cross-sell.

### Task 4: ****Analyze Pricing Policy Effects on Broadband Cross-Sell****

**Goal:** Explore the impact of TNBA pricing models on conversion rates.

**Relevant Columns:**

* pricing\_tier (STRING): Tier of the pricing strategy.
* pricing\_floor (FLOAT): Minimum price for the product.
* pricing\_ceiling (FLOAT): Maximum price for the product.
* target\_price (FLOAT): Final price recommended.
* promo\_period (STRING): Promotional period for the price.
* product\_type (STRING): Type of product recommended (e.g., broadband, add-ons).
* action\_recommended\_price\_flag (INTEGER): Indicates if a specific price strategy was recommended.

### Task 5: ****Link Digital Touchpoints to Outcomes****

**Goal:** Analyze how digital touchpoints and pricing models impact outcomes.

**Relevant Columns:**

* request\_location (STRING): Location of the recommendation in the app.
* request\_channel (STRING): Channel where interaction happened.
* product\_name (STRING): Product shown to the user.
* curr\_service\_revenue (FLOAT): Current service revenue post-cross-sell.
* prev\_service\_product\_revenue (NUMERIC): Previous revenue for comparison.

### Task 6: ****SQL Dashboard for Conversion Metrics****

**Goal:** Build queries to automate reporting.

**Relevant Columns:**

* add\_to\_cart\_flag (INTEGER): Tracks whether users added the product to their cart.
* recommended\_flag (INTEGER): Tracks the success of recommendations.
* transaction\_date (DATE): Transaction date for purchase tracking.
* curr\_service\_arpu (NUMERIC): Average revenue per user for the current period.
* prev\_service\_arpu (NUMERIC): ARPU before the recommendation.

### Task 7: ****Design Randomized Experiments****

**Goal:** Set up A/B tests for RL vs. BAU.

**Relevant Columns:**

* experiment\_id (STRING): Unique ID for experiments.
* experiment\_limb\_id (STRING): Identifies the control or treatment group.
* product\_transaction\_flag (INTEGER): Tracks conversion success.
* transaction\_date (DATE): Timing of transactions.

If you need further mapping or additional queries for these tasks, let me know!

##### You said:

Now write the detailed code for task 1

###### ChatGPT said:

ChatGPT

Here’s the detailed Python code for **Task 1: Analyze App Card Exposure**. The code is structured to extract, process, and visualize the data. It uses **SQLAlchemy** for querying the database, **pandas** for data manipulation, and **matplotlib** for visualization.

Analyze Card Exposure

99

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

# Import necessary libraries

import pandas as pd

from sqlalchemy import create\_engine

import matplotlib.pyplot as plt

import seaborn as sns

# Database connection setup

DATABASE\_URI = "postgresql+psycopg2://username:password@host:port/database"

engine = create\_engine(DATABASE\_URI)

# Step 1: Extract data from the database

query = """

SELECT

request\_time,

request\_channel,

request\_section,

experiment\_id,

experiment\_limb\_id,

pricing\_strategy

FROM

app\_recommendations

WHERE

request\_time BETWEEN '2024-11-01' AND '2024-11-30';

"""

# Execute query and load data into a DataFrame

data = pd.read\_sql(query, engine)

# Step 2: Data preprocessing

# Convert request\_time to datetime

data['request\_time'] = pd.to\_datetime(data['request\_time'])

# Extract the date and hour for analysis

data['date'] = data['request\_time'].dt.date

data['hour'] = data['request\_time'].dt.hour

# Map experiment\_limb\_id to labels for easier interpretation

data['experiment\_limb'] = data['experiment\_limb\_id'].map({

'limb\_1': 'Reinforced Learning',

'limb\_2': 'Business-As-Usual'

})

# Step 3: Exploratory Data Analysis (EDA)

# 1. Count the number of cards shown per limb

distribution\_by\_limb = data.groupby('experiment\_limb')['experiment\_id'].count().reset\_index()

distribution\_by\_limb.columns = ['Experiment Limb', 'Number of Cards']

# 2. Distribution of cards by request section and limb

cards\_by\_section = data.groupby(['request\_section', 'experiment\_limb']).size().reset\_index(name='count')

# Step 4: Visualization

# Bar plot for the number of cards shown per limb

plt.figure(figsize=(10, 6))

sns.barplot(x='Experiment Limb', y='Number of Cards', data=distribution\_by\_limb)

plt.title('Number of Cards Shown by Experiment Limb', fontsize=16)

plt.xlabel('Experiment Limb', fontsize=12)

plt.ylabel('Number of Cards', fontsize=12)

plt.show()

# Heatmap for card distribution by request section and limb

pivot\_table = cards\_by\_section.pivot('request\_section', 'experiment\_limb', 'count')

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

sns.heatmap(pivot\_table, annot=True, fmt='d', cmap='Blues')

plt.title('Card Distribution by Request Section and Experiment Limb', fontsize=16)

plt.xlabel('Experiment Limb', fontsize=12)

plt.ylabel('Request Section', fontsize=12)

plt.show()

# Step 5: Save results

# Save the distribution\_by\_limb to a CSV file

distribution\_by\_limb.to\_csv('card\_distribution\_by\_limb.csv', index=False)

# Save the cards\_by\_section to a CSV file

cards\_by\_section.to\_csv('card\_distribution\_by\_section.csv', index=False)

print("Analysis and visualization completed. Results saved to CSV files.")

The code extracts card exposure data, processes it to group by experimental limbs and request sections, and generates visualizations to understand the exposure patterns. Let me know if you need modifications or additional analysis!

##### You said:

the data is in billion records

###### ChatGPT said:

ChatGPT

For datasets with billions of records, efficient processing and scalability are essential. Here's how you can adapt the code for large-scale data, leveraging **distributed processing** and **efficient query design**:

### ****Optimized Approach for Task 1: Analyze App Card Exposure****

1. **Database Query Design**:
   * Use SQL to filter and aggregate data at the database level before bringing it into Python.
   * Only retrieve the columns and rows required for the analysis to minimize data transfer.
2. **Distributed Computing**:
   * Use distributed data processing tools like **PySpark** or **Dask** to handle large datasets in Python.
   * Perform aggregations and joins efficiently on distributed clusters.
3. **Incremental Processing**:
   * Break down processing into chunks by date, region, or user groups.
   * Use batch processing to analyze smaller subsets.
4. **Visualization**:
   * Use summary statistics or pre-aggregated data for visualization instead of raw records.

### ****Code Implementation with PySpark****

python

Copy code

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count, when

# Step 1: Initialize Spark Session

spark = SparkSession.builder \

.appName("Analyze App Card Exposure") \

.config("spark.sql.shuffle.partitions", "200") \

.getOrCreate()

# Step 2: Load Data from Database

# Replace with your database connection details

jdbc\_url = "jdbc:your\_database\_url"

db\_properties = {

"user": "your\_user",

"password": "your\_password",

"driver": "com.mysql.jdbc.Driver"

}

table\_name = "your\_table\_name"

# Load data with selected columns

columns\_to\_load = ["request\_time", "request\_section", "experiment\_limb\_id", "request\_channel"]

df = spark.read.jdbc(jdbc\_url, table=table\_name, properties=db\_properties)

# Step 3: Data Filtering and Aggregation

# Filter by date range and required sections

filtered\_df = df.filter(

(col("request\_time") >= "2024-01-01") &

(col("request\_time") <= "2024-01-31")

)

# Aggregate data to count requests by section and experimental limb

aggregated\_df = filtered\_df.groupBy("experiment\_limb\_id", "request\_section") \

.agg(count("\*").alias("request\_count"))

# Step 4: Save Aggregated Data for Visualization

aggregated\_df.write.csv("aggregated\_card\_exposure.csv", header=True)

# Step 5: Visualize Results (Optional in Python)

# Load the aggregated data into Pandas for visualization

import pandas as pd

import matplotlib.pyplot as plt

pandas\_df = pd.read\_csv("aggregated\_card\_exposure.csv")

pivot\_table = pandas\_df.pivot(index="request\_section", columns="experiment\_limb\_id", values="request\_count")

# Plot as a bar chart

pivot\_table.plot(kind="bar", figsize=(12, 6))

plt.title("Card Exposure by Experimental Limb and Request Section")

plt.xlabel("Request Section")

plt.ylabel("Request Count")

plt.legend(title="Experiment Limb")

plt.tight\_layout()

plt.show()

### ****Key Features of the Code****:

1. **Scalability**:
   * **PySpark** processes large datasets in a distributed manner.
   * Aggregations are performed within the database or Spark cluster, minimizing data transfer.
2. **Incremental Loading**:
   * Filters (request\_time) reduce the dataset size.
   * Data is grouped by experiment\_limb\_id and request\_section to create a smaller, manageable dataset.
3. **Storage Efficiency**:
   * Saves the aggregated results (CSV) for downstream analysis or visualization.
4. **Visualization**:
   * Results are visualized in Python after processing in Spark.

### ****Benefits****:

* **Efficient Handling**: Spark handles billions of records efficiently.
* **Reduced Overhead**: Pre-aggregated results reduce memory usage.
* **Reusability**: The code can process other time ranges or subsets with minor changes.

Let me know if you need assistance with setting up this environment or further optimization!

##### You said:

More detailed analysis

###### ChatGPT said:

ChatGPT

For handling **billions of records**, a detailed and scalable analysis requires a systematic approach that focuses on **data pipeline optimization, distributed processing, and detailed insights generation**. Here's a comprehensive plan and extended implementation:

### ****Plan for Detailed Analysis of App Card Exposure****

1. **Extract, Filter, and Load (ETL)**:
   * Query only the necessary columns (minimizing unnecessary data transfer).
   * Apply time and category filters to reduce data volume at the source.
2. **Data Aggregation**:
   * Group the data by key dimensions (experiment\_limb\_id, request\_section, request\_channel) and calculate metrics (e.g., count, mean response times).
3. **Explore Exposure Metrics**:
   * Total exposures per section and limb.
   * Relative exposure distribution (percentage share per section).
   * Time-based exposure trends (hourly, daily, or weekly patterns).
4. **Investigate Reinforcement Learning (RL) vs. BAU Logic**:
   * Compare RL and BAU exposures in different sections.
   * Analyze effectiveness by adding downstream metrics (e.g., conversion rates).
5. **Optimize for Large Data Volume**:
   * Use **partitioning** (e.g., by request\_time) for efficient data processing.
   * Leverage **PySpark** for distributed computing.

### ****Detailed Analysis Code****

Below is a more detailed implementation using **PySpark**, focusing on aggregations, filtering, and insights extraction.

#### ****1. Set Up Spark Session****

python

Copy code

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count, sum, when, lit, date\_format

# Initialize Spark session

spark = SparkSession.builder \

.appName("Detailed Card Exposure Analysis") \

.config("spark.sql.shuffle.partitions", "200") \

.getOrCreate()

#### ****2. Load Data from Database****

python

Copy code

# Database connection details

jdbc\_url = "jdbc:your\_database\_url"

db\_properties = {

"user": "your\_user",

"password": "your\_password",

"driver": "com.mysql.jdbc.Driver"

}

# Define table name and columns to load

table\_name = "your\_table\_name"

columns\_to\_load = ["request\_time", "request\_section", "experiment\_limb\_id", "request\_channel", "add\_to\_cart\_flag"]

# Load data with only required columns

df = spark.read.jdbc(jdbc\_url, table=table\_name, properties=db\_properties)

#### ****3. Apply Filters****

python

Copy code

# Define time range and other filters

filtered\_df = df.filter(

(col("request\_time") >= "2024-01-01") &

(col("request\_time") <= "2024-01-31") &

(col("request\_section").isNotNull()) &

(col("experiment\_limb\_id").isNotNull())

)

#### ****4. Aggregation for Analysis****

python

Copy code

# Aggregate request counts by section and experiment limb

aggregated\_df = filtered\_df.groupBy("experiment\_limb\_id", "request\_section") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("add\_to\_cart\_rate")

)

# Show sample results

aggregated\_df.show(truncate=False)

#### ****5. Time-Based Exposure Analysis****

python

Copy code

# Extract day and hour from request\_time

time\_df = filtered\_df.withColumn("day", date\_format(col("request\_time"), "yyyy-MM-dd")) \

.withColumn("hour", date\_format(col("request\_time"), "HH"))

# Aggregate requests by day and hour

time\_trend\_df = time\_df.groupBy("day", "hour") \

.agg(count("\*").alias("hourly\_requests"))

# Show time-based trends

time\_trend\_df.show(truncate=False)

#### ****6. Analyze Reinforcement Learning (RL) vs. BAU Logic****

python

Copy code

# Compare RL and BAU request counts

rl\_bau\_df = filtered\_df.groupBy("experiment\_limb\_id", "request\_section") \

.agg(count("\*").alias("request\_count"))

# Calculate percentage distribution

total\_requests = rl\_bau\_df.agg(sum("request\_count").alias("total")).collect()[0]["total"]

rl\_bau\_df = rl\_bau\_df.withColumn("percentage", (col("request\_count") / lit(total\_requests)) \* 100)

# Show RL vs. BAU comparison

rl\_bau\_df.orderBy("experiment\_limb\_id", "request\_section").show(truncate=False)

#### ****7. Save Aggregated Results****

python

Copy code

# Save aggregated data to CSV for further use

aggregated\_df.write.csv("output/card\_exposure\_analysis.csv", header=True)

# Save time trends

time\_trend\_df.write.csv("output/time\_trends.csv", header=True)

#### ****8. Visualization in Python (Optional)****

python

Copy code

import pandas as pd

import matplotlib.pyplot as plt

# Load aggregated data

df = pd.read\_csv("output/card\_exposure\_analysis.csv")

# Pivot for plotting

pivot\_df = df.pivot(index="request\_section", columns="experiment\_limb\_id", values="total\_requests")

# Plot bar chart

pivot\_df.plot(kind="bar", figsize=(12, 6), stacked=True)

plt.title("Card Exposure by Section and Experimental Limb")

plt.xlabel("Request Section")

plt.ylabel("Total Requests")

plt.legend(title="Experiment Limb")

plt.tight\_layout()

plt.show()

### ****Optimizations for Billion Records****

1. **Partitioning and Bucketing**:
   * Partition by request\_time or experiment\_limb\_id to distribute processing.
   * Use bucketing for highly cardinal columns like request\_section.
2. **Incremental Processing**:
   * Process data incrementally by date (daily or weekly batches).
   * Save intermediate results in distributed storage (e.g., HDFS, S3).
3. **Cluster Configuration**:
   * Ensure sufficient cluster size for processing.
   * Optimize Spark configuration (e.g., executor memory, number of partitions).
4. **Pre-aggregation**:
   * Perform initial aggregations at the database level if supported.

### ****Expected Outputs****

1. **Card Exposure by Section and Experimental Limb**:
   * Total requests.
   * Add-to-cart counts and rates.
2. **Time Trends**:
   * Hourly and daily exposure trends.
3. **RL vs. BAU Effectiveness**:
   * Exposure and performance comparison.

This approach ensures scalability and actionable insights while managing the challenges of large-scale data processing. Let me know if you need further refinements!

##### You said:

write the code for each task

###### ChatGPT said:

ChatGPT

Below is the code for each task using **PySpark** and **Python** for detailed analysis of the dataset containing billions of records. Each task is modularized for clarity and scalability.

### ****Task 1: Analyze Card Exposure on App****

python

Copy code

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count, sum, when, date\_format

# Initialize Spark session

spark = SparkSession.builder \

.appName("Analyze App Card Exposure") \

.config("spark.sql.shuffle.partitions", "200") \

.getOrCreate()

# Load data

df = spark.read.jdbc(

url="jdbc:your\_database\_url",

table="your\_table\_name",

properties={"user": "your\_user", "password": "your\_password", "driver": "com.mysql.jdbc.Driver"}

)

# Filter for relevant columns and data

filtered\_df = df.select("request\_time", "request\_section", "experiment\_limb\_id") \

.filter((col("request\_time") >= "2024-01-01") & (col("request\_time") <= "2024-01-31"))

# Aggregate requests by section and experimental limb

aggregated\_df = filtered\_df.groupBy("experiment\_limb\_id", "request\_section") \

.agg(count("\*").alias("total\_requests"))

# Save aggregated results

aggregated\_df.write.csv("output/task1\_card\_exposure.csv", header=True)

### ****Task 2: Assess Limbs Assigned and Traceability****

python

Copy code

# Add columns for traceability and calculate downstream metrics

traceability\_df = df.select("experiment\_limb\_id", "request\_section", "add\_to\_cart\_flag") \

.groupBy("experiment\_limb\_id", "request\_section") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("conversion\_rate")

)

# Save results

traceability\_df.write.csv("output/task2\_traceability.csv", header=True)

### ****Task 3: Evaluate Confounding Variables in Cross-Sell Conversions****

python

Copy code

# Add social grade and regional segmentation for confounders

confounding\_df = df.select(

"household\_region", "household\_social\_grade", "experiment\_limb\_id", "request\_section", "add\_to\_cart\_flag"

) \

.groupBy("household\_region", "household\_social\_grade", "experiment\_limb\_id", "request\_section") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("conversion\_rate")

)

# Save results

confounding\_df.write.csv("output/task3\_confounding\_variables.csv", header=True)

### ****Task 4: Analyze Pricing Policy Effects on Broadband Cross-Sell****

python

Copy code

pricing\_df = df.select("pricing\_tier", "pricing\_floor", "pricing\_ceiling", "experiment\_limb\_id", "add\_to\_cart\_flag") \

.groupBy("pricing\_tier", "experiment\_limb\_id") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("conversion\_rate"),

sum("pricing\_floor").alias("avg\_pricing\_floor"),

sum("pricing\_ceiling").alias("avg\_pricing\_ceiling")

)

# Save results

pricing\_df.write.csv("output/task4\_pricing\_effects.csv", header=True)

### ****Task 5: Link Digital Touchpoints to Cross-Sell Outcomes****

python

Copy code

digital\_touchpoints\_df = df.select(

"request\_channel", "request\_location", "experiment\_limb\_id", "add\_to\_cart\_flag"

) \

.groupBy("request\_channel", "request\_location", "experiment\_limb\_id") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("conversion\_rate")

)

# Save results

digital\_touchpoints\_df.write.csv("output/task5\_digital\_touchpoints.csv", header=True)

### ****Task 6: SQL Dashboard for Conversion Metrics****

python

Copy code

# Conversion metrics (save as table for SQL-based reporting/dashboard)

conversion\_metrics\_df = df.select(

"experiment\_limb\_id", "request\_section", "add\_to\_cart\_flag"

) \

.groupBy("experiment\_limb\_id", "request\_section") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("conversion\_rate")

)

# Save table to database for dashboard integration

conversion\_metrics\_df.write.jdbc(

url="jdbc:your\_database\_url",

table="conversion\_metrics",

mode="overwrite",

properties={"user": "your\_user", "password": "your\_password", "driver": "com.mysql.jdbc.Driver"}

)

### ****Task 7: Conduct Randomized Experiments****

python

Copy code

# Compare RL vs. BAU experiment limbs

rl\_bau\_experiment\_df = df.select("experiment\_limb\_id", "request\_section", "add\_to\_cart\_flag") \

.groupBy("experiment\_limb\_id", "request\_section") \

.agg(

count("\*").alias("total\_requests"),

sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)).alias("add\_to\_cart\_count"),

(sum(when(col("add\_to\_cart\_flag") == 1, 1).otherwise(0)) / count("\*") \* 100).alias("conversion\_rate")

)

# Save RL vs. BAU comparison

rl\_bau\_experiment\_df.write.csv("output/task7\_rl\_vs\_bau.csv", header=True)

### ****Summary of Outputs****

1. **Task 1**: Aggregated card exposure data saved as task1\_card\_exposure.csv.
2. **Task 2**: Traceability insights saved as task2\_traceability.csv.
3. **Task 3**: Confounding variable analysis saved as task3\_confounding\_variables.csv.
4. **Task 4**: Pricing effects saved as task4\_pricing\_effects.csv.
5. **Task 5**: Digital touchpoints and outcomes saved as task5\_digital\_touchpoints.csv.
6. **Task 6**: SQL-based conversion metrics saved in a database table (conversion\_metrics).
7. **Task 7**: RL vs. BAU experiment results saved as task7\_rl\_vs\_bau.csv.

These modular tasks ensure scalability and efficient processing for large datasets while generating actionable insights. Let me know if you'd like to delve deeper into any specific task!