**D.E ASSESSMENT SOLUTION DOCUMENT**

**PYTHON**

**Q 1.) Fill None Values: Given a list, replace None values with the previous non-None value. If consecutive Nones occur, fill each with the last non-None value. Example: [1, None, 1, 2, None] becomes [1, 1, 1, 2, 2].**

num = [1,2,3,None,4,None]

for i in range(len(num)):

if num[i] == None:

num[i] = num[i-1]

print(num)

Q2.) **Mismatched Words Finder: Write a function that returns a list of words present in two strings that don't match in case. Example: Input: "Datumlabs is an awesome place", "Datumlabs.io Is an AWESOME place". Output: ["is", "Is", "awesome", "AWESOME"].**

def mismatched\_words():

a = "Cars are fast"

b = "Bikes are Fast"

not\_matched = []

str\_a = a.split()

str\_b = b.split()

for words in str\_a:

if words not in string\_b:

not\_matched.append(words)

for words in str\_b:

if words not in string\_a:

not\_matched.append(words)

print(not\_matched)

mismatched\_words()

**Q 3.) Character Frequency Counter: Create a function to count the occurrences of a specific character in a string. Example: 'mississippi', 's' should return 3**

def counter():

a = "Python is a programming language"

counter = a.count('a')

print(counter)

counter()

**Q4.) Nth Largest Value Key Finder: Write a function to find the key of the nth largest value in a dictionary. Example: For {a: 1, b: 2, c: 100, d: 30}, and n = 2, return 'd'**

def key\_finder():

num = {'a':1000, 'b': 55,'c': 455,'d':37,'e':900, 'f':999}

n2 = sorted(num.keys())[-1]

print(n2)

key\_finder()

**SQL QUESTIONS:**

**1. Percentage of Paid Customers Who Bought Both Product A and Product B: Given a table CustomerPurchases with columns customer\_id, product\_id, purchase\_date, price, and payment\_status, calculate the percentage of customers who bought both productsA and B and paid for them.**

FIRST QUERY OUT OF CUSTOMER WHO PAID (FOR PRODUCT A and B, including Unpaid- Need clarity)

SELECT ROUND(CAST(PAID\_CUSTOMERS AS DECIMAL) / CAST(TOTAL\_CUSTOMERS AS DECIMAL) \* 100, 2) AS PAID\_PERCENTAGE

FROM (

SELECT count(cp1.customer\_id) AS PAID\_CUSTOMERS

FROM customerpurchases cp1

**FIRST QUERY**

I’ve used self-join here where customers bought both Product A and Product B and filtered with Product\_id ‘A’ and ‘B’ in the first and joining with where payment status = paid for both products in Sub1 sub query.

Sub2, used the same logic but excluded payment status to get all customers who bought Product A and B

**SECOND QUERY**

I’ve used the same logic overall but for denominator I’ve grabbed all distinct customers.

JOIN customerpurchases cp2 ON cp1.customer\_id = cp2.customer\_id

WHERE TRIM(cp1.product\_id) = 'A'

AND TRIM(cp2.product\_id) = 'B'

AND cp1.payment\_status = cp2.payment\_status

) SUB1

JOIN (

SELECT Count(cp1.customer\_id) TOTAL\_CUSTOMERS

FROM customerpurchases cp1

JOIN customerpurchases cp2 ON cp1.customer\_id = cp2.customer\_id

WHERE TRIM(cp1.product\_id) = 'A'

AND TRIM(cp2.product\_id) = 'B'

) SUB2

ON 1 = 1

**Ans: 66.67%**

SECOND QUERY OUT OF ALL DISTINCT CUSTOMERS (NEED CLARITY)

SELECT ROUND(CAST(PAID\_CUSTOMERS AS DECIMAL) / CAST(TOTAL\_CUSTOMERS AS DECIMAL) \* 100, 2) AS PAID\_PERCENTAGE

FROM (

SELECT count(cp1.customer\_id) AS PAID\_CUSTOMERS

FROM customerpurchases cp1

JOIN customerpurchases cp2 ON cp1.customer\_id = cp2.customer\_id

WHERE TRIM(cp1.product\_id) = 'A'

AND TRIM(cp2.product\_id) = 'B'

AND cp1.payment\_status = cp2.payment\_status

) SUB1

JOIN (

SELECT count(DISTINCT customer\_id) AS TOTAL\_CUSTOMERS

FROM customerpurchases

) SUB2

ON 1 = 1

**ANSWER**: 50 %

2. Percentage of Sales Attributed to Promotions on First and Last Days: With the Sales table (columns: sale\_id, product\_id, sale\_date, amount, promotion\_id) and Promotions table (columns: promotion\_id, start\_date, end\_date, discount\_rate), compute the percentage of sales attributed to promotions on their first and last days

SELECT \*

FROM (

SELECT Round(fd\_amount / total\_sales \* 100, 2) AS First\_day\_percentage

FROM (

SELECT total\_amount AS FD\_amount

FROM (

SELECT SUM(amount) AS total\_amount

,CASE

The green highlighted part will retrieve Percentage for **First Day promotions**. The inner join on promotion id will exclude NONE in promotion\_id. Then I’ve used a case statement with if sale date matches with start date this will be considered first day and if not that will be considered last day.

The blue highlighted part will retrieve Percentage for **Last Day promotions**.

The purple highlighted part will retrieve sum of both first and last day promotions which will be used as a denominator.

WHEN s.sale\_date = pr.start\_date

THEN 'First Day'

WHEN s.sale\_date > start\_date

AND s.sale\_date <= end\_date

THEN 'Last days'

END AS PR\_days

FROM Sales s

INNER JOIN promotions pr ON TRIM(pr.promotion\_id) = TRIM(s.promotion\_id)

GROUP BY PR\_days

) FD

WHERE pr\_days = 'First Day'

) join\_1

JOIN (

SELECT Sum(total\_amount) Total\_sales

FROM (

SELECT SUM(amount) AS total\_amount

,CASE

WHEN s.sale\_date = pr.start\_date

THEN 'First Day'

WHEN s.sale\_date > start\_date

AND s.sale\_date <= end\_date

THEN 'Last days'

END AS PR\_days

FROM Sales s

INNER JOIN promotions pr ON TRIM(pr.promotion\_id) = TRIM(s.promotion\_id)

GROUP BY PR\_days

) Total\_amount

) Join\_2 ON 1 = 1

) CL\_JOIN

JOIN (

SELECT Round(fd\_amount / total\_sales \* 100, 2) AS Last\_day\_percentage

FROM (

SELECT total\_amount AS FD\_amount

FROM (

SELECT SUM(amount) AS total\_amount

,CASE

WHEN s.sale\_date = pr.start\_date

THEN 'First Day'

WHEN s.sale\_date > start\_date

AND s.sale\_date <= end\_date

THEN 'Last days'

END AS PR\_days

FROM Sales s

INNER JOIN promotions pr ON TRIM(pr.promotion\_id) = TRIM(s.promotion\_id)

GROUP BY PR\_days

) FD

WHERE pr\_days = 'Last days'

) join\_1

JOIN (

SELECT Sum(total\_amount) Total\_sales

FROM (

SELECT SUM(amount) AS total\_amount

,CASE

WHEN s.sale\_date = pr.start\_date

THEN 'First Day'

WHEN s.sale\_date > start\_date

AND s.sale\_date <= end\_date

THEN 'Last days'

END AS PR\_days

FROM Sales s

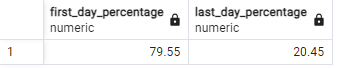
INNER JOIN promotions pr ON TRIM(pr.promotion\_id) = TRIM(s.promotion\_id)

GROUP BY PR\_days

) Total\_amount

) Join\_2 ON 1 = 1

) CL\_JOIN\_2 ON 2 = 2



3**.) 3. Top 5 Complementary Products for Product A: Identify the top 5 products bought alongside Product A (NEED CLARITY ON ALONGSIDE PRODUCTS)**

SELECT

s1.product\_id,

s2.product\_id,

COUNT(\*) AS frequency

FROM

There’s only product B which is bought 3 times.

I would need some clarity on these queries which I will discuss on call

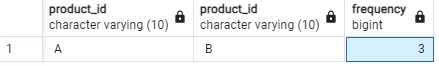
customerpurchases s1

JOIN

customerpurchases s2 ON s1.customer\_id = s2.customer\_id AND s1.product\_id <> s2.product\_id

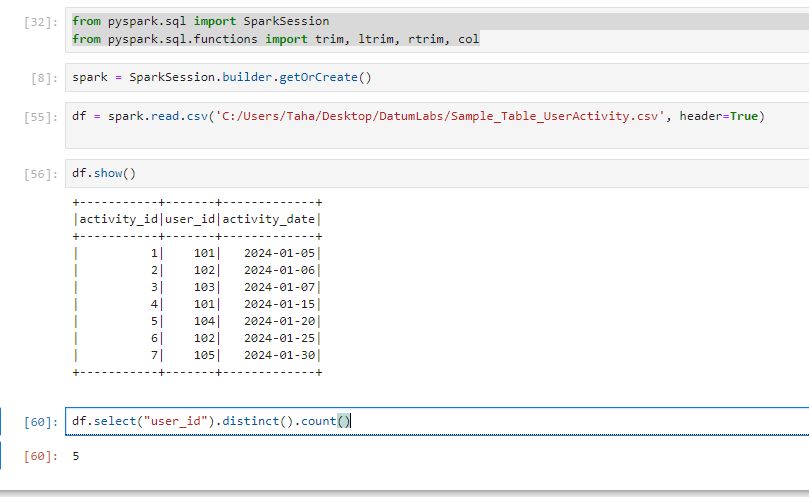
where Trim(s1.product\_id) = 'A'

group by s1.product\_id,s2.product\_id



PYSPARK

Q1 1. **Monthly Active Users (MAU) for January 2024: Count of unique users active in January 2024.**



from pyspark.sql import SparkSession

from pyspark.sql.functions import trim, ltrim, rtrim, col

spark = SparkSession.builder.getOrCreate()

df = spark.read.csv('C:/Users/Taha/Desktop/DatumLabs/Sample\_Table\_UserActivity.csv', header=True)

df.show()

df.select("user\_id").distinct().count()

**2. Total Sales Revenue for January 2024: Sum of sales in January 2024.**



from pyspark.sql import SparkSession

from pyspark.sql.functions import trim, ltrim, rtrim, col

from pyspark.sql.types import IntegerType,DateType, StringType

df = spark.read.csv('C:/Users/Taha/Desktop/DatumLabs/Sample\_Table\_Sales.csv', header=True)

total\_sales = df.select(sum(df["amount"])).show()

**3. Average Sale Amount Per Category for January 2024.**



df = df.select("amount","category\_id")

df\_grouped = df.groupby("category\_id").agg({"amount":"avg"})

df\_grouped\_renamed = df\_grouped.withColumnRenamed("avg(amount)", "AVG\_Sale")

Avg\_Sale = df\_grouped\_renamed

Avg\_Sale.show()

4. NOT CLEAR

5. Top Selling Product Category in January 2024: Product category with highest sales in January 2024.



sales = spark.read.csv('C:/Users/Taha/Desktop/DatumLabs/Sample\_Table\_Sales.csv', header=True)

category = spark.read.csv('C:/Users/Taha/Desktop/DatumLabs/Sample\_Table\_Categories.csv', header=True)

join\_data = sales.join(category, trim(sales.category\_id) == trim(category.category\_id), how ="inner") \

.select("product\_id",sales["category\_id"]," category\_name","amount").groupby(" category\_name").agg({"amount":"sum"}) \

.orderBy("sum(amount)",ascending = False).limit(1)

Top\_Selling\_Category = join\_data.withColumnRenamed("sum(amount)", "Highest\_Sales")

Top\_Selling\_Category.show()