

## Datum Labs Assessment

### Python Questions:

1.

```
def fill_none_value(array): #function definition with array as
#argument
    if len(array) == 0: # handling a case for length of array=0
        return f"List is empty"
    else:
        for i in range(len(array)):
            if(array[i] == None):
                array[i] = array[i-1] # if current element is None then
#fill with the last element
            if (array[0] == None):
                array[i] = array[i+1]

        return array

array1 = [None,2,3,None,None] #function usage
fill_none_value(array1)
```

Code output:

[2, 2, 3, 3, 3]

2.

```
def split_and_return_list(string1, string2): #function declaration
#taking two strings as arguments.
    words1 = string1.split() #list
    words2 = string2.split() #list
    mismatched_Words = []
    for i in range (len(words2)): #condition checking the case.
        if (words1[i] != words2[i]):
            mismatched_Words.append(words1[i])

    return f"{mismatched_Words}"
```

```
# Example usage
input_string = "This is a sample string" #defining string
input_string1 = "THIS IS A sample string" #defining string

split_and_return_list(input_string, input_string1) #function usage
```

**Code output:**

```
['This', 'is', 'a']
```

**3.**

```
def get_char_frequency(str): #function definition
    frequency = {} #deifning of dictionary
    for i in str: #increment each character in the string
        if i in frequency:
            frequency[i] += 1
        else:
            frequency[i] = 1 #if not present then make frequency to one.
    return f"Frequency of each character in list is {frequency}"

def get_specific_char_frequency(str, char): #function for computing
#frequency of specific character.
    count = 0
    for i in str:
        if i == char:
            count += 1

    return f"Count of {char} in string {str} is {count}"

my_string = "This is This is usman usman"
get_specific_char_frequency(my_string, "i")
```

**Code ouput:**

**Count of i in string This is This is usman usman is 4**

**SQL Questions:**

**1.**

```

SELECT

    customers_with_both_paid,

    total_unique_customers.unique_customers,

    ROUND((customers_with_both_paid * 100.0 / total_unique_customers.unique_customers), 2) AS
percentage

FROM (

    SELECT

        COUNT(DISTINCT cp1.customer_id) AS customers_with_both_paid

    FROM

        CustomerPurchases AS cp1

    WHERE

        cp1.product_id = 'ProductA'

        AND cp1.payment_status = 'Paid'

        AND EXISTS (

            SELECT 1

            FROM CustomerPurchases AS cp2

            WHERE cp1.customer_id = cp2.customer_id

            AND cp2.product_id = 'ProductB'

            AND cp2.payment_status = 'Paid'

        )

    ) AS customers_with_paid_number,

(

    SELECT

        COUNT(DISTINCT customer_id) AS unique_customers

    FROM

        CustomerPurchases

    ) AS total_unique_customers;

```

**Code output:**

## Output

customers_with_both_paid	unique_customers	percentage
3	6	50

**3.**

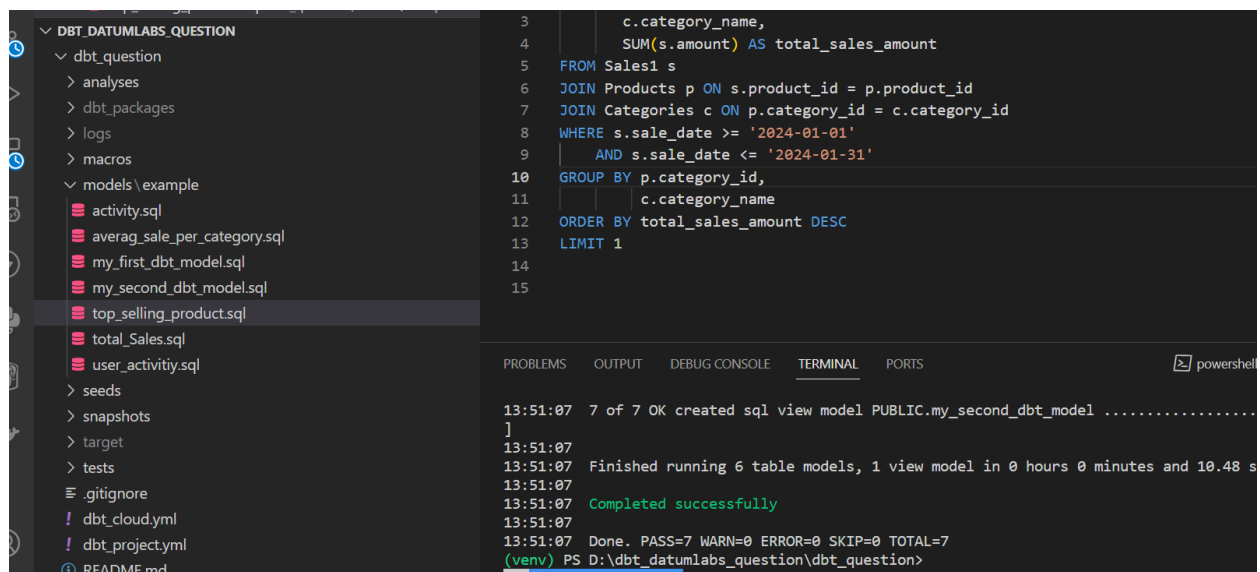
```
SELECT
    cp.product_id AS complementary_product_id,
    COUNT(*) AS frequency
FROM
    CustomerPurchases cp
JOIN
    CustomerPurchases cp_a ON cp.customer_id = cp_a.customer_id
WHERE
    cp_a.product_id = 'ProductA'
    AND cp.product_id <> 'ProductA'
GROUP BY
    cp.product_id
ORDER BY
    frequency DESC
LIMIT 5;
```

## Code output

complementary_product_id	frequency
ProductB	4

## Dbt Question:

The following is the screenshot of the overall environment.



The screenshot shows a VS Code editor with a DBT project structure on the left and a terminal window on the right. The project structure includes a folder named 'DBT\_DATUMLABS\_QUESTION' with subfolders 'dbt\_question', 'analyses', 'dbt\_packages', 'logs', 'macros', and 'models'. The 'models' folder contains several SQL files, including 'top\_selling\_product.sql'. The terminal window shows the execution of a DBT command, resulting in a successful completion of 7 models and 1 view model in 10.48 seconds.

```
3      c.category_name,  
4      SUM(s.amount) AS total_sales_amount  
5  FROM Sales1 s  
6  JOIN Products p ON s.product_id = p.product_id  
7  JOIN Categories c ON p.category_id = c.category_id  
8  WHERE s.sale_date >= '2024-01-01'  
9        AND s.sale_date <= '2024-01-31'  
10 GROUP BY p.category_id,  
11          c.category_name  
12 ORDER BY total_sales_amount DESC  
13 LIMIT 1  
14  
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
13:51:07 7 of 7 OK created sql view model PUBLIC.my_second_dbt_model .....  
13:51:07 ]  
13:51:07 Finished running 6 table models, 1 view model in 0 hours 0 minutes and 10.48 s  
13:51:07 Completed successfully  
13:51:07 Done. PASS=7 WARN=0 ERROR=0 SKIP=0 TOTAL=7  
(venv) PS D:\dbt_datumlabs_question\dbt_question>
```

1.

```
{{ config(materialized='table') }}  
SELECT COUNT(DISTINCT user_id) AS active_users  
FROM USERACTIVITY  
  
WHERE activity_date >= '2024-01-01' and activity_date <= '2024-01-30'
```

Code output in the snowflake database

• COMPUTE\_WH

1 Row. Updated just now

↺

	ACTIVE_USERS
1	5

2.

```
{{ config(materialized='table') }}
```

```
SELECT SUM(amount) AS total_sales_revenue
FROM Sales
WHERE sale_date >= '2024-01-01'
      AND sale_date <= '2024-01-31'
```

Code output:

• COMPUTE\_WH

1 Row. Updated just now

↺

	TOTAL_SALES_REVENUE
1	168.00

3.

```
{{ config(materialized='table') }}
```

```
SELECT category_id,
       AVG(amount) AS average_sale_amount
FROM SALES1
GROUP BY category_id
```

#### Code output:

• COMPUTE\_WH 3 Rows • Updated just now 

	CATEGORY_ID	AVERAGE_SALE_AMOUNT
1	C1	100.000000
2	C3	200.000000
3	C2	150.000000

4.

```
{{ config(materialized='table') }}  
SELECT user.user_id,  
       user.user_name,  
       activity.activity_id,  
       activity.activity_date,  
       user.join_date  
FROM USERACTIVITY activity  
JOIN USERS user ON user.user_id = activity.user_id  
WHERE user.join_date >= '2024-01-01'  
      AND user.join_date <= '2024-01-31'
```

#### Code output:

No output as there were no users who joined in the January.

5.



```

{{ config(materialized='table') }}
SELECT p.category_id,
       c.category_name,
       SUM(s.amount) AS total_sales_amount
FROM Sales1 s
JOIN Products p ON s.product_id = p.product_id
JOIN Categories c ON p.category_id = c.category_id
WHERE s.sale_date >= '2024-01-01'
      AND s.sale_date <= '2024-01-31'
GROUP BY p.category_id,
         c.category_name
ORDER BY total_sales_amount DESC
LIMIT 1

```

#### Code output:

• COMPUTE\_WH 1 Row. Updated just now



	CATEGORY_ID	CATEGORY_NAME	TOTAL_SALES_AMOUNT
1	C2	Clothing	300