8/29/2024

Muhammad Usman

Assessment

Datum Labs

# Python Tasks

**Problem 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].

**Solution:**

# Created a function which relace None values

def replaceNone(listNum):

  # Loop on the list of values having None

  for i in range(len(listNum)):

    # Check if current value is none and it is not first index and assign the non none value to the index where none is present

    if listNum[i]==None and i!=0:

      listNum[i]=listNum[i-1]

  return listNum

numList=[1,None,1,2,None]

print(replaceNone(numList))

**Approach:**

* First there cannot be none values at the start of the list
* Second I loop over the list check the current is none then replace that with previous index value.
* Finally it will return the list replacing “None” with previous non none value

**Other Approach:**

* It can also be done using pandas data frame and list comprehension as well.

**Output:**

****

**Problem 2:**

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"].

**Solution:**

def find\_mismatched\_words(string1, string2):

    # Split the strings into lists of words

    words1 = string1.split()

    words2 = string2.split()

    # Create a dictionary to store words in lowercase and their original cases

    word\_dict1 = {word.lower(): word for word in words1}

    word\_dict2 = {word.lower(): word for word in words2}

    # Find words that are present in both strings but have different cases

    mismatched\_words = []

    for word\_lower in word\_dict1:

        if word\_lower in word\_dict2 and word\_dict1[word\_lower] != word\_dict2[word\_lower]:

            mismatched\_words.extend([word\_dict1[word\_lower], word\_dict2[word\_lower]])

    return mismatched\_words

#Calling the function

string1 = "Datumlabs is an awesome place"

string2 = "Datumlabs.io Is an AWESOME place"

mismatched\_words = find\_mismatched\_words(string1, string2)

print(mismatched\_words)

**Approach:**

* First split the strings into words list.
* Second I created dictionary of words in lower case.
* Finally I loop through first string check if it is present and check case of both if they are not equal to each other then store them into list and finally return the list of mismatch strings in both

**Other Approach:**

* It can also be done using list comprehension such as :
* Split both strings into lists of words.
* Use a list comprehension to filter words from string1 that have a matching word in string2 but with different cases.

**Output:**

****

**Problem 3:**

3.Character Frequency Counter: Create a function to count the occurrences of a specific

character in a string.

Example: 'mississippi', 's' should return 3.

**Solution:**

def frequencyCounter(word, character):

    # Check if the word length is 0 which there is not any word

    if len(word) == 0:

        print("Word length is equal to 0.")

        return 0

    # initialize with 0 to keep track of no of characters present in word string

    count = 0

    # Iterate through the word and count occurrences of the character

    for i in range(len(word)):

        if word[i] == character:

            count += 1

    return count

# Usage of function

word=""

word=input("Enter the word for which you want to count number of characteritics : ")

character = ""

character=input("Enter the character which you want to count: ")

print(frequencyCounter(word, character)

**Approach:**

* First check the word length is 0 then return0.
* Second loop over the each word and check if the character matches in word and count +1 if matched .

**Other Approach:**

* It can also be done using python in build count function which will return specific character count in the word

**Output:**

****

**Problem 4:**

4. 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'.

**Solution:**

def NthLargest(dictionaryNum, n):

    # Sort the dictionary by value in descending order and convert to a list of tuples

    sortedDictionary = sorted(dictionaryNum.items(), key=lambda x: x[1], reverse=True)

    # Check if the list has at least n elements

    if n > len(sortedDictionary) or n <= 0:

        print("Invalid input: 'n' is out of range.")

        return

    # Get the n-th largest item (1-based index)

    nth\_largest\_item = sortedDictionary[n - 1]

    # Print the key and value of the n-th largest item

    key, value = nth\_largest\_item

    print(f"The {n}-th largest key-value pair is: Key: {key}, Value: {value}")

# Usage of function

NthLargest({'a': 1, 'b': 44, 'c': 3, 'd': 4, 'e': 50}, 2)

**Approach:**

* First check the word length is 0 then return0.
* Second loop over each word and check if the character matches in word and count +1 if matched.

**Other Approach:**

* It can also be done using python in build count function which will return specific character count in the word

**Output:**

****

# SQL Tasks

**Problem 1:**

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.

**Solution:**

WITH "CustomersWhoBoughtBoth" AS (

    SELECT customer\_id

    FROM "public"."CustomerPurchases"

    WHERE product\_id IN ('A', 'B') AND payment\_status = 'Paid'

    GROUP BY customer\_id

    HAVING COUNT(DISTINCT product\_id) = 2

),

"TotalPaidCustomers" AS (

    SELECT COUNT(DISTINCT customer\_id) AS total\_customers

    FROM "public"."CustomerPurchases"

    WHERE payment\_status = 'Paid'

)

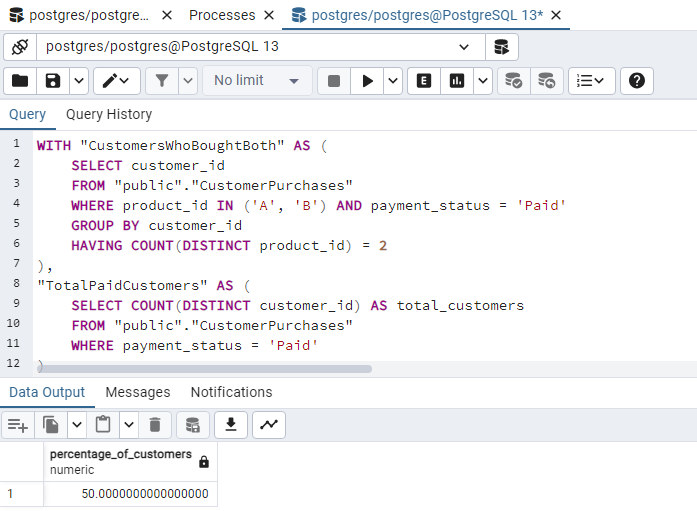
SELECT

    (SELECT COUNT(DISTINCT customer\_id) FROM "CustomersWhoBoughtBoth") \* 100.0 / (SELECT total\_customers FROM "TotalPaidCustomers") AS percentage\_of\_customers;

**Approach:**

* Identify customers who purchased both products A and B and then calculate the percentage of these customers against the total number of paid customers

**Output:**

****

**Problem 2:**

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.

**Solution:**

WITH "SalesOnPromotionDays" AS (

    SELECT s.sale\_id, s.amount

    FROM "Sales" s

    JOIN "Promotions" p ON s.promotion\_id = p.promotion\_id

    WHERE s.sale\_date = p.start\_date OR s.sale\_date = p.end\_date

),

"SalesData" AS (

    SELECT SUM(amount) AS "sales\_on\_promotion\_days"

    FROM "SalesOnPromotionDays"

),

"TotalSales" AS (

    SELECT SUM(amount) AS "total\_sales"

    FROM "Sales"

)

SELECT

    (sd.sales\_on\_promotion\_days \* 100.0) / ts.total\_sales AS "percentage\_of\_sales"

FROM

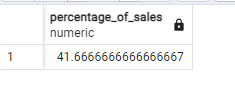
    "SalesData" sd,

    "TotalSales" ts;

**Approach:**

* I first identify sales that occurred on the first and last days of promotions, then calculate the total sales on those days and compare it to the overall sales to determine the percentage.

**Output:**

****

**Problem 3:**

3. Top 5 Complementary Products for Product A: Identify the top 5 products bought

alongside Product A.

**Solution:**

WITH "CustomersWhoBoughtA" AS (

    SELECT DISTINCT customer\_id

    FROM "CustomerPurchases"

    WHERE product\_id = 'A'

),

"OtherProductsBought" AS (

    SELECT cp.product\_id, COUNT(\*) AS purchase\_count

    FROM "CustomerPurchases" cp

    JOIN "CustomersWhoBoughtA" cba ON cp.customer\_id = cba.customer\_id

    WHERE cp.product\_id != 'A'

    GROUP BY cp.product\_id

)

SELECT

    product\_id, purchase\_count

FROM

    "OtherProductsBought"

ORDER BY

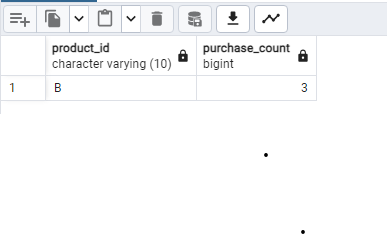
    purchase\_count DESC

LIMIT 5;

**Approach:**

* I'm identifying customers who bought Product A and then finding other products they purchased. First, I select customers who bought Product A. Then, I count how many times these customers bought other products, excluding Product A. Finally, I order these other products by the number of times they were purchased, showing the top 5 most frequently bought products.

**Output:**

****

# Pyspark Tasks

**Problem 1:**

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

2024.

**Solution:**

user\_activity\_df=spark.read.csv('UserActivity.csv',header=True,inferSchema=True)

#user\_activity\_df.show()

user\_activity\_df\_24 = user\_activity\_df.filter((year(col('activity\_date')) == 2024) & (month(col('activity\_date')) == 1))

#user\_activity\_df\_24.show()

user\_activity\_df\_24\_unique=user\_activity\_df\_24.select('user\_id').distinct().count()

print(f'Monthly Active User for January 2024 :{user\_activity\_df\_24\_unique}')

**Approach:**

* Identify customers who purchased both products A and B and then calculate the percentage of these customers against the total number of paid customers

**Output:**

****

**Problem 2:**

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

2024.

**Solution:**

from pyspark.sql.functions import sum as spark\_sum

sales\_df=spark.read.csv('Sales.csv',header=True,inferSchema=True)

#sales\_df.show()

df\_filtered = sales\_df.filter((year(col('sale\_date')) == 2024) & (month(col('sale\_date')) == 1))

# Sum the sales\_amount for jan 2024

total\_sales\_revenue = df\_filtered.agg(spark\_sum('amount').alias('total\_sales\_revenue')).collect()[0]['total\_sales\_revenue']

print(f'Total sales revenue for January 2024: {total\_sales\_revenue}')

**Approach:**

* Identify customers who purchased both products A and B and then calculate the percentage of these customers against the total number of paid customers

**Output:**

****

**Problem 3:**

3. Average Sale Amount Per Category for January 2024:Average sale amount per category in

January 2024.

**Solution:**

from pyspark.sql.functions import  avg

sales\_df=spark.read.csv('Sales.csv',header=True,inferSchema=True)

# filter data for jan 2024

sales\_df\_24 = sales\_df.filter((year(col('sale\_date')) == 2024) & (month(col('sale\_date')) == 1))

# Calculate average sale amount per category

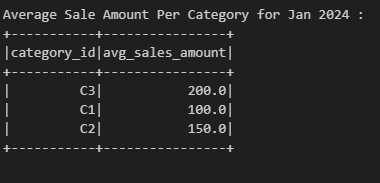
avg\_sales\_per\_categ= sales\_df\_24.groupBy('category\_id').agg(avg('amount').alias('avg\_sales\_amount'))

print("Average Sale Amount Per Category for Jan 2024 :")

# Show the result

avg\_sales\_per\_categ.show()

**Output:**

****

**Problem 4:**

4. Number of New Users in January 2024: Count of users who joined in January 2024.

**Solution:**

from pyspark.sql.functions import col, month, year

user\_data=spark.read.csv('Users.csv',header=True,inferSchema=True)

user\_data.show()

# filter users who join in Jan 2024

user\_jan\_24 = user\_data.filter((year(col('join\_date')) == 2024) & (month(col('join\_date')) == 1))

print(f'Count of user in January 2024 : {user\_jan\_24.count()} ')

**Output:**

**A black background with white text

Description automatically generated**

**Problem 5:**

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

January 2024.

**Solution:**

sales\_df=spark.read.csv('Sales.csv',header=True,inferSchema=True)

categories\_df=spark.read.csv('Categories.csv',header=True,inferSchema=True)

# Filter data for January 2024

sales\_df\_filtered = sales\_df.filter((year(col('sale\_date')) == 2024) & (month(col('sale\_date')) == 1))

# Calculate total sales per category

total\_sales\_per\_category = sales\_df\_filtered.groupBy('category\_id').agg(spark\_sum('amount').alias('total\_sales'))

# Join with categories table to get category names

result\_df = total\_sales\_per\_category.join(categories\_df, on='category\_id', how='inner')

# Find the top-selling category

top\_selling\_category = result\_df.orderBy(col('total\_sales').desc()).first()

print(f"Top Selling Product Category in January 2024: {top\_selling\_category['category\_name']} with total sales of ${top\_selling\_category['total\_sales']}")

**Output:**

****