Technical Assessment for Data Engineering Role

Datum Labs

Name: Muhammad Hamza
Email: m.hamza455878@gmail.com
Phone No#: 0303-4168680
Code Repo: https://github.com/MuhammadHamza45/Data-Engineering-Assessment-Datum-labs.git

Python Questions

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:

```
# replace None values with the previous non-None value

def ReplaceNoneValue (ls):
    preVal = None
    for i in range(len(ls)):
        if ls[i] == None:
            ls[i] = preVal
            preVal = ls[i]
        else:
            preVal = ls[i]
    return ls

ls = [1, None, 1, 2, None]
ls = ReplaceNoneValue(ls)
print(ls)
```

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:

```
# Write a function that returns a list of words present in two strings that don't match in case.

def MismatchedWordsFinder (str1, str2):
    ls1 = str1.split()
    ls2 = str2.split()
    resLs = []
    minLenght = min(len(ls1), len(ls2))
    for i in range(minLenght):
        if ls1[i].lower() == ls2[i].lower() and ls1[i] != ls2[i]:
            resLs.append(ls1[i])
            resLs.append(ls2[i])
```

```
    return resLs

str1 = "Datumlabs is an awesome place"
str2 = "Datumlabs.io Is an AWESOME place"
result = MismatchedWordsFinder (str1, str2)
print(result)
```

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:

```
# Create a function to count the occurrences of a specific character in a string.

def CharFrequencyCounter (inputStr, countChar):
    occurrences = inputStr.count(countChar)
    return occurrences

inputStr = "mississippi"
countChar = "s"
print(CharFrequencyCounter(inputStr, countChar))
```

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:

```
# Write a function to find the key of the nth largest value in a dictionary.

def NthLargestValueKeyFinder (inputDic, n):
    sortDic = sorted(inputDic.items(), key = lambda item: item[1], reverse = True)
    if n <= len(inputDic):
        return sortDic[n-1][0]
    else:
        return None

inputDic = {
    "a": 1,
    "b": 2,
    "c": 100,
    "d": 30
}
n = 2
result = NthLargestValueKeyFinder(inputDic, n)
print(result)
```

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.

   Solution:

```
WITH CustomerBoughtAB AS (
        SELECT COUNT(C.customer_id) AS TotalCustomerWithAB
                FROM (
                SELECT customer_id
                        FROM CustomerPurchases
                        WHERE payment_status = 'Paid' AND
                        product_id IN ('A', 'B')
                        GROUP BY customer_id
                        HAVING COUNT(DISTINCT product_id) = 2
                ) C
),
TotalPaid AS (
        SELECT COUNT(DISTINCT customer_id) AS TotalPaidCustomer
                FROM CustomerPurchases
)

SELECT (c.TotalCustomerWithAB * 100) / t.TotalPaidCustomer AS [Percentage Paid Customers AB]
FROM CustomerBoughtAB c
Cross JOIN TotalPaid t
```

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:

```
SELECT (SP.SalesAmount * 100) / TS.TotalAmount AS [Percentage Sales On Promo Days]
        FROM (
                SELECT SUM(s.amount) AS SalesAmount
                        FROM dbo.Sales s
                        JOIN dbo.Promotions p
                        ON s.promotion_id = p.promotion_id
                        WHERE s.sale_date = p.start_date
                        OR s.sale_date = p.end_date
                ) AS SP
        CROSS JOIN (
                SELECT SUM(amount) AS TotalAmount
                        FROM dbo.Sales
                ) TS
```

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

Solution:

```
SELECT C.[Complementary Products]
        FROM (
                SELECT product_id AS [Complementary Products],
                        SUM(price) AS Sales, ROW_NUMBER() OVER(ORDER BY SUM(price)) AS
ProdRank
                        from dbo.CustomerPurchases
                        WHERE customer_id in (
                                SELECT customer_id
                                        FROM [dbo].[CustomerPurchases]
                                        WHERE product_id = 'A')
                        AND product_id != 'A'
                        GROUP BY product_id
        ) C
        WHERE C.ProdRank <= 5
```

DBT/PysparkMetricsCalculation:

1. MonthlyActiveUsers(MAU)forJanuary2024:CountofuniqueusersactiveinJanuary 2024.

Solution:

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

from pyspark.sql.functions import to_date, col
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

# Define the schema and create DataFrame
schema = StructType([
   StructField("activity_id", IntegerType(), True),
   StructField("user_id", IntegerType(), True),
   StructField("activity_date", StringType(), True)  # Use StringType here
])

data = [
   (1, 101, "2024-01-05"),
   (2, 102, "2024-01-06"),
   (3, 103, "2024-01-07"),
   (4, 101, "2024-01-15"),
   (5, 104, "2024-01-20"),
   (6, 102, "2024-01-25"),
   (7, 105, "2024-01-30")
]
```

```
df_userActivity = spark.createDataFrame(data, schema=schema)

# Convert activity_date col to date type
df_userActivity = df_userActivity.withColumn("activity_date",
to_date(df_userActivity["activity_date"], "yyyy-MM-dd"))
#df_userActivity.show()

# Filter data for January 2024
filter_data = df_userActivity.filter((col("activity_date") >= "2024-01-01") & (col("activity_date")
<= "2024-01-31"))
#filter_data.show()

# Count unique user_id from filter_data
mau = filter_data.select("user_id").distinct().count()

print(f"MAU for January 2024: {mau}")
```

2. TotalSalesRevenueforJanuary2024:SumofsalesinJanuary2024.
   Solution:

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

from pyspark.sql.functions import to_date, col, sum
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType

# Define the schema and create DataFrame for Sales
schema = StructType([
   StructField("sale_id", IntegerType(), True),
   StructField("product_id", StringType(), True),
   StructField("sale_date", StringType(), True),  # Use StringType for the date initially
   StructField("amount", FloatType(), True),
   StructField("category_id", StringType(), True)
])

data = [
   (1, "P001", "2024-01-01", 100.00, "C1"),
   (2, "P002", "2024-01-05", 150.00, "C2"),
   (3, "P001", "2024-01-10", 100.00, "C1"),
   (4, "P003", "2024-01-15", 200.00, "C3"),
   (5, "P002", "2024-01-20", 150.00, "C2")
]

df = spark.createDataFrame(data, schema=schema)
```

```
# Convert sale_date col to date type
df = df.withColumn("sale_date", to_date(df["sale_date"], "yyyy-MM-dd"))
#df.show()

# Filter data for January 2024
filter_data = df.filter((col("sale_date") >= "2024-01-01") & (col("sale_date") <= "2024-01-31"))
#filter_data.show()

# Total sales revenue for January 2024
total_sale =
filter_data.agg(sum("amount").alias("TotalSalesRevenue")).collect()[0]["TotalSalesRevenue"]

print(f"Total Sales Revenue for January 2024: {total_sale}")
```

3. AverageSaleAmountPerCategoryforJanuary2024:Averagesaleamountpercategoryin January2024.
   Solution:

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

from pyspark.sql.functions import to_date, col, avg
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType

# Define the schema and create DataFrame for sales
schema = StructType([
   StructField("sale_id", IntegerType(), True),
   StructField("product_id", StringType(), True),
   StructField("sale_date", StringType(), True),  # Use StringType for the date initially
   StructField("amount", FloatType(), True),
   StructField("category_id", StringType(), True)
])

data = [
   (1, "P001", "2024-01-01", 100.00, "C1"),
   (2, "P002", "2024-01-05", 150.00, "C2"),
   (3, "P001", "2024-01-10", 100.00, "C1"),
   (4, "P003", "2024-01-15", 200.00, "C3"),
   (5, "P002", "2024-01-20", 150.00, "C2")
]

df_sales = spark.createDataFrame(data, schema=schema)

# Define the schema and create DataFrame for category
schema_cat = StructType([
   StructField("category_id", StringType(), True),
```

```
    StructField("category_name", StringType(), True)
])

# Create data
data_cat = [
    ("C1", "Electronics"),
    ("C2", "Clothing"),
    ("C3", "HomeAppliances")
]

# Create the DataFrame
df_category = spark.createDataFrame(data_cat, schema=schema_cat)

# Convert sale_date col to date type
df_sales = df_sales.withColumn("sale_date", to_date(df_sales["sale_date"], "yyyy-MM-dd"))
#df.show()

# Filter data for January 2024
filter_data = df_sales.filter((col("sale_date") >= "2024-01-01") & (col("sale_date") <= "2024-01-
31"))
#filter_data.show()

# Rename the 'amount' column in sales DataFrame to avoid ambiguity
filter_data = filter_data.withColumnRenamed("amount", "sale_amount")

# sales data by category
sales_cat = filter_data.join(df_category, "category_id")
#sales_cat.show()

# average sale amount per category
avg_sale_cat = sales_cat.groupBy("category_id",
"category_name").agg(avg("sale_amount").alias("AverageSaleAmount"))
avg_sale_cat.show()
```

4. NumberofNewUsersinJanuary2024:CountofuserswhojoinedinJanuary2024.
   Solution:

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

from pyspark.sql.types import StructType, StructField, StringType, IntegerType
from pyspark.sql.functions import to_date

# Define the schema for the DataFrame for Users
schema = StructType([
    StructField("user_id", IntegerType(), True),
```

```
    StructField("user_name", StringType(), True),
    StructField("join_date", StringType(), True)  # Use StringType initially for the date
])

# Create data
data = [
    (101, "Alice", "2023-05-10"),
    (102, "Bob", "2023-06-15"),
    (103, "Charlie", "2023-07-20"),
    (104, "Dana", "2023-08-25"),
    (105, "Emily", "2024-01-30")
]

# Create the DataFrame
df = spark.createDataFrame(data, schema=schema)

# Convert the 'join_date' column to DateType
df = df.withColumn("join_date", to_date(df["join_date"], "yyyy-MM-dd"))
#df.show()

# Filter users who joined in January 2024
january_new_users = df.filter((col("join_date") >= "2024-01-01") & (col("join_date") <= "2024-
01-31")).count()

print(f"Number of New Users in January 2024: {january_new_users}")
```

5. TopSellingProductCategoryinJanuary2024:Productcategorywithhighestsalesin January2024.
   Solution:

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

from pyspark.sql.functions import to_date, col, sum, desc
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType

# Define the schema and create DataFrame for sales
schema = StructType([
    StructField("sale_id", IntegerType(), True),
    StructField("product_id", StringType(), True),
    StructField("sale_date", StringType(), True),  # Use StringType for the date initially
    StructField("amount", FloatType(), True),
    StructField("category_id", StringType(), True)
])

data = [
```

```
    (1, "P001", "2024-01-01", 100.00, "C1"),
    (2, "P002", "2024-01-05", 150.00, "C2"),
    (3, "P001", "2024-01-10", 100.00, "C1"),
    (4, "P003", "2024-01-15", 200.00, "C3"),
    (5, "P002", "2024-01-20", 150.00, "C2")
]

df_sales = spark.createDataFrame(data, schema=schema)

# Define the schema and create DataFrame for category
schema_cat = StructType([
    StructField("category_id", StringType(), True),
    StructField("category_name", StringType(), True)
])

# Create data
data_cat = [
    ("C1", "Electronics"),
    ("C2", "Clothing"),
    ("C3", "HomeAppliances")
]

# Create the DataFrame
df_category = spark.createDataFrame(data_cat, schema=schema_cat)

# Convert sale_date col to date type
df_sales = df_sales.withColumn("sale_date", to_date(df_sales["sale_date"], "yyyy-MM-dd"))
#df.show()

# Filter data for January 2024
filter_data = df_sales.filter((col("sale_date") >= "2024-01-01") & (col("sale_date") <= "2024-01-
31"))
#filter_data.show()

# Rename the 'amount' column in sales DataFrame to avoid ambiguity
filter_data = filter_data.withColumnRenamed("amount", "sale_amount")

# sales data by category
sales_cat = filter_data.join(df_category, "category_id")
#sales_cat.show()

# average sale amount per category
avg_sale_cat = sales_cat.groupBy("category_id",
"category_name").agg(sum("sale_amount").alias("TotalSales"))
```

```
Top_Selling_Pro_Cat = avg_sale_cat.orderBy(desc("TotalSales")).limit(1)
#Top_Selling_Pro_Cat.show()
print(f"Top Selling Product Category in January 2024:
{Top_Selling_Pro_Cat.collect()[0]['category_name']}")
```
