Lab3-Task2: Guided solution for Add Missing Dates

Overview

The aim of this exercise is to enrich flight data with the most recent dates that match the recorded day of the week and the day of the month. This allows for a richer, more complete dataset that can be used for time-series analysis, trend recognition, and other forms of data analytics.

1. **Initialize Spark Session**: Configurations for running Spark are set, including setting the driver memory.
2. **Load Data**: The matched flight data is read from an S3 path.
3. **Generate Date DataFrame**: A sequence of dates for the entire year 2020 is generated.
4. **Enrich Date DataFrame**: Additional columns for day of the week and day of the month are added.
5. **Find Most Recent Dates**: For each (day of week, day of month) pair, find the most recent corresponding date.
6. **Join with Flight Data**: The most recent date information is joined with the original flight data.
7. **Save Enriched Data**: The enriched data is saved back to S3 in Parquet format.

Step 1: Import Statements

importing necessary modules and libraries. The first three lines are specific to PySpark, whereas **datetime** is from Python's standard library.

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import types as T
from datetime import date, timedelta
from pyspark.sql import Row
```

Step 2: Define Function to Generate Dates

```
def get_dates_df():
    dummy_df = spark.createDataFrame([Row(dummy='x')])
    in_dates_df = dummy_df.select(F.explode(F.sequence(F.lit("2020-01-01").cast(T.DateType()), F.lit("2020-12-31").cast(T.DateType()))).alias("flight_date"))

    return in_dates_df
```

break down the function **get_dates_df()** line-by-line to understand its workings:

```
def get_dates_df():
```

This line defines a new function called **get_dates_df**, which does not take any arguments.

```
dummy_df = spark.createDataFrame([Row(dummy='x')])
```

1. **spark.createDataFrame():** This function creates a new Spark DataFrame.
2. [**Row(dummy='x')]:** This is a list containing one row with a single column named dummy, which has a value of 'x'. The DataFrame will have just this one row. The purpose of creating this DataFrame with a single row is to have a "base" DataFrame that you can operate on to generate a range of dates.

```
in_dates_df = dummy_df.select(F.explode(F.sequence(F.lit("2020-01-01").cast(T.DateType()), F.lit("2020-12-31").cast(T.DateType()))).alias("flight_date"))
```

1. **dummy_df.select():** This function is used to select and manipulate columns.
2. **F.explode():** This function takes an array column and creates a new row for each element in the array. In this case, it's being used to create a new row for each date in the generated sequence.
3. **F.sequence():** This function generates a sequence of numbers or dates. Here, it's generating a sequence of dates from "2020-01-01" to "2020-12-31".
4. **F.lit("2020-01-01").cast(T.DateType()):** F.lit() creates a column with a constant value, "2020-01-01" in this case. cast(T.DateType()) converts this constant to a DateType. This is done for both the start and end dates.
5. **.alias("flight_date"):** This part renames the new column containing the expanded dates to "flight_date".

The result is a DataFrame with a single column called "flight_date" containing all the dates from "2020-01-01" to "2020-12-31".

```
return in_dates_df
```

This line returns the DataFrame in_dates_df as the output of the function.

The overall function generates a DataFrame with a sequence of dates from the start of the year 2020 to the end, which can be used for joining or other operations with other DataFrames.

Step 3: Spark Session Initialization

```
spark =
SparkSession.builder.master("local").appName('ex3_add_dates').getOrCreate()
```

Step 4: Load Data

read data from a Parquet file located in S3 and stores it in a DataFrame called **flights_df**.

```
flights_df = spark.read.parquet('s3a://spark/data/stg/flight_matched/')
```

Step 5: Create Date DataFrame

calls the get_dates_df function to generate a DataFrame containing the dates for the year 2020.

```
dates_df = get_dates_df()
```

Step 6: Add Day of Week and Day of Month to Dates DataFrame

- Two new columns day_of_week and day_of_month are added to dates_df using PySpark's withColumn method.
- The actual values are calculated using PySpark functions dayofweek and dayofmonth.

```
dates_full_df = dates_df \
    .withColumn('day_of_week', F.dayofweek(F.col('flight_date'))) \
    .withColumn('day_of_month', F.dayofmonth(F.col('flight_date')))
```

Step 7: Find Max Date for Each Combination of Day of Week and Day of Month

This block performs a groupBy operation on dates_full_df based on the day of the week and day of the month. Then, it finds the maximum date for each group using PySpark's agg and max functions.

```
max_date_df = dates_full_df \
    .groupBy(F.col('day_of_week'), F.col('day_of_month')) \
    .agg(F.max(F.col('flight_date')).alias('flight_date'))
```

Step 8: Join Flight Data with Max Date Data

Performs an inner join between **flights_df** and **max_date_df** based on the common columns **day_of_week** and **day_of_month**. The result is stored in **enriched_flights_df**.

```
enriched_flights_df = flights_df.join(max_date_df, ['day_of_week', 'day_of_month'])
```

Step 9: Save Data to Parquet Format

This line saves the **enriched_flights_df** DataFrame into a Parquet file. The **mode='overwrite'** option means it will overwrite the file if it already exists.

```
enriched_flights_df.write.parquet('s3a://spark/data/transformed/flights/',
mode='overwrite')

spark.stop()
```

Step 10 - Full Code Solution

Data Completion
Folder Name: exercises_three
File Name: add_missing_dates.py

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import types as T
from datetime import date, timedelta
from pyspark.sql import Row


def get_dates_df():
    dummy_df = spark.createDataFrame([Row(dummy='x')])
    in_dates_df = dummy_df.select(F.explode(F.sequence(F.lit("2020-01-
01").cast(T.DateType()), F.lit("2020-12-
31").cast(T.DateType()))).alias("flight_date"))

    return in_dates_df


spark =
SparkSession.builder.master("local").appName('ex3_add_dates').getOrCreate()

flights_df = spark.read.parquet('s3a://spark/data/stg/flight_matched/')
dates_df = get_dates_df()

dates_full_df = dates_df \
    .withColumn('day_of_week', F.dayofweek(F.col('flight_date'))) \
    .withColumn('day_of_month', F.dayofmonth(F.col('flight_date')))

max_date_df = dates_full_df \
    .groupBy(F.col('day_of_week'), F.col('day_of_month')) \
    .agg(F.max(F.col('flight_date')).alias('flight_date'))

enriched_flights_df = flights_df.join(max_date_df, ['day_of_week', 'day_of_month'])

enriched_flights_df.write.parquet('s3a://spark/data/transformed/flights/',
mode='overwrite')

spark.stop()
```