Report

Introduction

Scripts for the assignment will be found in the Code folder. The functions.py file contains the functions used to answer the business questions. The execute_this.py file was used to derive the results presented in the next sections. The only required package to run functions.py and execute_this.py is pandas. To run execute_this.py one needs to navigate to the directory it is stored and execute it. Every sql file found in the code file was used to create the table or view it is named after in a Postgres database. Functionality of the functions.py is presented in the next sections. The assignment solutions are implemented based on the fact that the dataset is static and some assumptions presented in the Assumptions section.

Overview of the Data :

The dataset consists of three columns, production_line_id , status and timestamp. Production_line_id column contains 4 distinct values , status column contains three distinct values and timestamp columns contains status measurements per production_line_id taken in the same day in 15 minute intervals (most of them).

Assumptions Made about the Dataset :

Every production_line starts the day with a start status and ends with an end status. In the case that the status of the first measurement of the day is ON this will be regarded as START, in the case that the last measurement of the day is ON this will be regarded as STOP, the transformation will be performed for every production_line_id.

Production lines are assumed to be working serially. As a consequence all of the production_lines need to be working in order for the production floor to be up and running.

The time interval of interest is assumed to be the timespan during which the measurements were observed, namely from 2020-10-07 01:00:00 to 2020-10-07 06:00:00.

functions.py

In the Code folder one can find the functions.py file. functions.py contain the functions used to preprocess the data as well as answer all three business questions. The five functions are : augment_data,transform_data,bq1,bq2 and bq3.

Augment_data function is designed to get as input the raw dataset in dataframe format with column names the names of the first row of the dataset.

Transform data is designed to get as input the result of the augment_data function.

Bq1,bq2 and bq3 are designed to get as input the result of the augment_data function.

What follows is an explanation of the logic implemented in every function.

augment_data: This function gets as input the raw dataset in dataframe format. At first it sorts dataset values by production_line_id and timestamp. It then checks the chronologically first row for every production_line_id , if the status is not START it replaces it with START. As a last step it checks the chronologically last row for every production_line_id , if the status is not STOP it replaces it with STOP. This is done to normalize the data according to the first assumption. It returns a transformed dataframe with the same format as the raw data dataframe i.e. with columns production_line_id,status, timestamp.

transform_data function is designed to get as input the result of the augment_data function. Transforms the timestamp column . Creates a new column named current start, it copies the Start time value only from the rows with status START and forward fills the rows that left empty. That way every row has the corresponding starting time in the current start column(this happens because the data is sorted). It then selects the rows with status STOP. By this point rows with status STOP have timestamp , the stopping time and current start , the corresponding starting time. A new column named Duration which is the time passed between starting time and stopping time. A reordering and renaming is performed and the function returns a dataframe ready to answer the business questions. The dataframe returned has columns production_line_id,Start_Time,Stop_Time,Duration.

bq1 function is designed to get as input the result of the transform_data function and a line_id. It simply selects the subset of the dataset that have production_line_id the line_id requested. This question is implemented in SQL aswell with a simple view : VIEW_LINE_47_RESULTS

Results:

```
Results for Question 1:
   production_line_id          Start_Time           Stop_Time        Duration
36           gr-np-47 2020-10-07 01:33:20 2020-10-07 02:03:20 0 days 00:30:00
45           gr-np-47 2020-10-07 02:15:02 2020-10-07 04:15:02 0 days 02:00:00
51           gr-np-47 2020-10-07 05:00:00 2020-10-07 05:55:17 0 days 00:55:17
```

bq2 function is designed to get as input the result of the transform_data function.

bq2 treats starts and stops as events and assigns scores on them. Specifically assigns +1 point to every start of a production line and -1 to every stop to a production line. Sorting the events in chronological order and calculating the cumulative sum of the scores we get the number of active production lines as these are lines that have started and not stopped. In our specific case and according to the second hypothesis we need 4 production lines running. To calculate the duration we shift the time column by 1 to bring both the timestamp of the current event and the next in the same row . That way we calculate the duration for every time interval. By summing the duration of the time intervals with 4 production lines running we have the total uptime of the production floor. The total downtime will be 5 full hours minus the uptime due to the third hypothesis.

Results:

```
Results for Question 2:
  Uptime_Duration Downtime_Duration
0 0 days 01:55:02   0 days 03:04:58
```

bq3 function is designed to get as input the result of the transform_data function.

It groups by production_line_id and sums Duration column. It then selects the argmin.

The result will be 5 full hours minus the Uptime Duration of the argmin. As a last step it converts the result to a dataframe for easier handling downstream. This question has been answered with SQL too in the view : VIEW_most_downtime

Results:

```
Results for Question 3:
  production_line_id        Duration
0           gr-np-22 0 days 02:20:00
```

All results provided are taken from the execute_this.py script . execute_this.py reads the raw dataset uses the functions in the functions.py file and prints the results.

Data Warehouse Implementation

To implement the above logic in a data warehouse several tables and views were used.

Namely:

Table tb_stg_rawdata : This table is used for storing the staging phase, the result of the augment_data function.

Contains columns : prodiction_line_id,status and timestamp.

Table tb_mdl_linedata : This table is used for storing the results of the transform_data function. This table is used to feed bg1,bg2 and bg3 functions.

Contains columns : production_line_id, start_time, stop_time, duration.

Table tb_ tb_mdl_line47data: This table is used to store the results for the first business question.

Contains columns : production_line_id, start_time, stop_time, duration.

View view_line_47_results : An implementation of the first business question in SQL.

Table tb_mdl_uptime_downtime : this table is used to store the results for the second business question.

Contains columns : uptime, downtime.

Table tb_ tb_mdl_most_downtime : This table is used to store the results for the second business question.

Contains columns : production_line_id, duration.

View view_most_downtime : An implementation of the third business question in SQL.

Nikos Nteits