

ETL Pipeline

User Manual

Date: 06.03.2020

Author: Bernardo Di Chiara

Status: Draft

Change History:

Version	Status	Date	Author	Comments / List of main changes
0.1	Draft	14.11.2019	Bernardo Di Chiara	First version (the code has not been peer-reviewed and this the document has not been reviewed)
0.2	Draft	06.03.2020	Bernardo Di Chiara	DB password changed in sections 5, 6, 7 and 9. Information about a new EC2 instance has been added in sections 6 and 9. Public accessibility has been changed to No and the security group has been changed in the RDS instance (section 9). List of needed libraries updated in section 5. The snapshots of the local test cases have been changed (section 7).

Contents

1. Scope of the Project	4
2. Results of the Exploratory Data Analysis	4
3. Design Principles and Assumptions.....	5
4. Solution Description.....	6
4.1. Structure of the SQL Tables.....	7
4.2. Configurable Parameters	7
4.3. Console Feedback.....	7
5. SW Environment	8
6. How to Run the Pipeline	9
7. Testing Plan	11
7.1. Testing Tools	11
7.2. Testing Environment	11
7.3. Test Cases.....	13
7.3.1. TC-1_Tables-Creation_Interrupting-the-Python-Modules.....	13
7.3.2. TC-2_Files-are-Modified-in-the-Watched-Directory	17
7.3.3. TC-3_Files-are-Deleted-in-the-Watched-Directory.....	19
7.3.4. TC-4_Directories-are-Added-in-the-Watched-Directory	21
7.3.5. TC-5_Aggregate-Data_Handling-Missing-Values_New-File-with-Same-Name.....	23
7.3.6. TC-6_Full-Dump_TESTING-is-False	27
8. Possible Future Improvements	30
8.1. Issues for Customer Feedback	30
8.2. Backlog Items	30
9. Annex 1: AWS RDS and EC2 Instance Configuration	32

1. Scope of the Project

The scope of this project is to build an ETL pipeline to extract data from a disk where CSV files are loaded daily. The purpose of the pipeline is to store the content of those files into a repository for further processing, together with some aggregate data.

The file name of the CSV file contains the date and this information shall be stored together with the content of the file in the repository.

The aggregate data shall be stored in a separate table and consists in the mean price in the last 30 days for each of the temperature groups.

The solution shall be tolerant to change of scheme and data untidiness.

2. Results of the Exploratory Data Analysis

An exploratory data analysis of the provided samples has been made.

The origin CSV files are named with the date in the format yyyy-mm-dd.csv (where yyyy is the year, mm is the month and dd is the day). No file is missing and the file names are correct and consistent.

165 files have been analyzed covering data from July 1st to December 12th 2018.

Each file contains 10 entries.

The variable 'temperature' contains categorical data (LOW, MEDIUM, HIGH).

The variable 'skipped_beat' contains integer kind of data with values ranging from 0 to 7

The variable 'at_risk' contains integer from 0 to 1 (Boolean kind of variable).

The variable 'price' contains decimal data.

The columns are not always in the same order in the origin files and in certain files some column (like the column price) is missing. (The columns price starts to appear from the file containing data from August 31st).

There are some missing values in the column 'skipped_beat'.

3. Design Principles and Assumptions

The design approach follows the Lean principles and aims to produce a first usable version with a minimum set of requirements to get customer feedback. Therefore, the following choices have been done.

Since the amount of data currently received every day is reasonable, the code has been written in Python with no usage of external framework (the Exploratory Data Analysis has been performed in Jupiter Notebook).

Also, since there is not a long history yet (the amount of files to be loaded into the database is not huge), the data is currently stored in a My SQL database.

The following assumptions have been made:

- It is safe to rely that the origin CSV files are named correctly
- The variable 'at_risk' can have only two values (this has allowed to use a Boolean variable in the database – this can be changed easily, if needed)
- The variable 'skipped_beat' has only integer values (again, this has to do with the definition of the corresponding variable in the database and can be changed easily)
- No additional variables are foreseen in the near future in addition to the ones mentioned in this file (changing this would require a more dynamic way to define the table structure of daily dump table, may be by using SQLAlchemy)
- Each day 10 entries are produced (this is related to the way the content of the aggregate table data is calculated – removing this assumption would require to build the table based on the values of the column containing the date rather than on the number of sorted entries)

The module testing has been done gradually during the SW development stage.

4. Solution Description

The solution consists of two python modules: `daily_data.py` and `aggregate_data.py`

`daily_data.py`

- monitors continuously a defined subdirectory for any new CSV files
- extracts the content of such files into a pandas dataframe
- extracts the date from the file name and appends it into a separate column in the dataframe
- does some formatting to the dataframe
- sends the data into the table 'Daily_Dump' of a defined MySQL database

`aggregate_data.py`

- daily calculates aggregate data (average price of the last 30 days) by reading from the 'Daily_Dump' table
- refreshes (clears and inserts) the information into a dedicated table ('Current_Mean_Price') in the same database

The solution tolerates the fact that the columns in the CSV files might not be always in the same order or might even be missing. The solution tolerates missing data in certain cells.

It has been decided to leave the entries with partial missing data into the table 'Daily_Dump'. Data cleansing can be done later if needed.

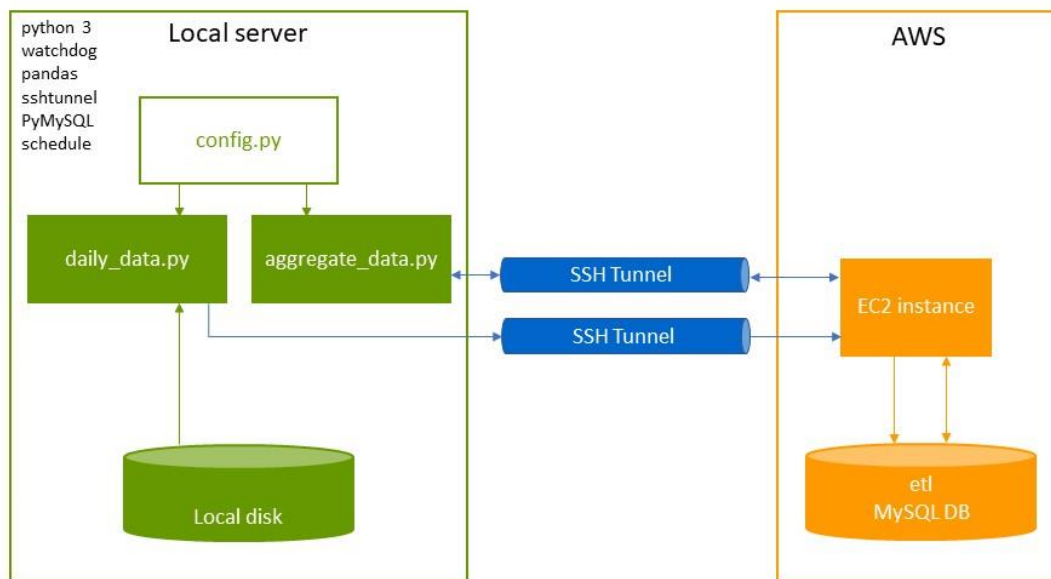


FIG. 1. SW Architecture.

4.1. Structure of the SQL Tables

The table 'Daily_Dump' contains the following columns:

- 'id' (integer) a running integer (primary key)
- 'temperature' (categorical)
- 'at_risk' (boolean)
- 'skipped_beat' (integer)
- 'price' (decimal)
- 'date' (date) - the content is built by using the file name of the origin CSV file

The columns 'temperature', 'at_risk', 'skipped_beat' and 'price' contain the information fetched from the CSV file.

The table 'Daily_Dump' is updated whenever there are changes in the origin disk.

The table 'Current_Mean_Price' has the following columns:

- 'id' (integer) a running integer (primary key)
- 'temperature_group' (categorical) - the group in the temperature column for which aggregate data has been calculated
- 'mean_price' (float) - the mean price in the last 30 days for a certain temperature group

The table 'Current_Mean_Price' is updated daily.

4.2. Configurable Parameters

It is possible to configure easily the following data by editing the separate file `config.py` that is called by both modules:

- Credentials for the remote SSH server used to access the database (ssh user, ssh host, ssh port, path to the private key)
- Credentials for the remote database (user, host, port, password, database name)
- Whether to use the remote database or a local database (for testing)
- Path of the subdirectory to watch
- interval in seconds between two consecutive checks of the watched directory (watchdog delay)
- Whether to update the aggregate table normally once per day or every minute (the last option is used in testing)
- Time of the day when to refresh the aggregate table
- Credentials for the local database (user, host, password, database name)

4.3. Console Feedback

Both python modules provide useful console feedback when run in Linux terminals.

5. SW Environment

The SW has been written by using Jupyter Notebook server 6.0.1 and Python 3.6.8 (running on Linux Ubuntu 18.04).

The MySQL database runs version 5.7.22.

The following python libraries are required:

- watchdog (0.9.0)
- pandas (0.25.2)
- sshtunnel (0.1.5)
- PyMySQL (0.9.3)
- schedule (0.6.0)

The requirement.txt file containing the required libraries is provided.

The credentials to access the database are the following ones:

- user: admin
- host: `maindbname.id.region.rds.amazonaws.com`
- port: 3306
- user password: `remote_db_user_password`
- database name: etl

The SQL database is hosted in AWS and is accessible through an EC2 instance by using SSH. The credentials to access the EC2 instance are the following ones:

- User ubuntu
- HostName `ssh_server_public_ip`
- Port 22
- PreferredAuthentications publickey
- IdentityFile `/home/user/.ssh/RDSkey.pem`

For details about the configuration of the RDS instance and of the EC2 instance in AWS, see Annex 1.

A Shell script (`provide_csv.sh`) has been created to facilitate testing. This script allows copying files from an origin directory into a destination directory.

NOTE: In this version of the file, for security reasons, the value of certain credentials has been hidden. You have to create your own database in order to test this feature.

6. How to Run the Pipeline

Environment

The modules `daily_data.py` and `aggregate_data.py`, the file `config.py` as well as the script `provide_csv.sh` shall be in a dedicated directory. The same directory shall contain the following subdirectories:

- The directory 'test_orig' containing all the 165 CSV files (from July 1st to December 12nd)
- The empty subdirectory called 'test_dest' which will receive the files from `provide_csv.sh` and which will be watched by `daily_data.py`

A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal) is needed. For the SSH server credentials and the remote database credentials, see section 5.

Three more terminals are needed to run the two python modules and the shell script.

Note that in normal conditions those 165 files will be loaded during the corresponding 165 days. In here, for obvious practical reasons, a 1-day time interval has been reduced to 10 seconds. Reducing it further would not be tolerated by the micro AWS instances that have been used in this project.

This test case is used to verify that the feature works end-to-end with the AWS instances and that no conflict arises when the two modules run simultaneously from the beginning.

Preconditions

The database etl contains no tables.

Execution

- **Edit** the file `config.py` by setting the parameter `WATCHDOG_DELAY` to 1 and by setting the parameter `TESTING` to 'True'
- In the same file check that `DIRECTORY_TO_WATCH = "test_dest/"` and `LOCAL_TESTING = False`
- Check that the parameter `DELAY` in `provide_csv.sh` is set to 10
- In a separate terminal (while in the dedicated directory) run the `daily_data.py` module
 - `python3 daily_data.py`
- In a separate terminal (while in the dedicated directory) run the `aggregate_data.py` module
 - `python3 aggregate_data.py`
- In a separate terminal (while in the dedicated directory) run the shell script
 - `./provide_csv.sh`
- Check the terminal where `daily_data.py` is running and wait till the last file (2018-12-12.csv) has been loaded into the database (*).
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
 - It shall return 1650 entries
- Check the terminal where `aggregate_data.py` has been running. You should see the following messages: "The aggregate table has been updated at yyy-mm-dd-hh:mm:ss.xxxxxx"
- Check the content of the table 'Current_Mean_Price' in the database
 - `SELECT COUNT(id) FROM Current_Mean_Price;`
 - You should see 3 entries
 - `SELECT * FROM Current_Mean_Price;`
 - You should see the values shown in FIG.14 in section 7.3.6 (TC-6). If not, wait one minute and give the last command again.

(*) The module execution takes approximatively 18 minutes.

Postconditions

- Interrupt [daily_data.py](#) and [aggregate_data.py](#) (CTRL+C.)
- Edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY back to 60 and by setting the parameter TESTING back to 'False'

7. Testing Plan

7.1. Testing Tools

The shell script `provide_csv.sh` allows copying files from one folder to another one with a configurable delay. It can be used to simulate the dumping of origin csv files into the origin disk.

Make sure you have set the right permission for the file with: `chmod u+x provide_csv.sh`

7.2. Testing Environment

Python 3.7 is needed with the following libraries:

- watchdog (0.9.0)
- pandas (0.25.2)
- sshtunnel (0.1.5)
- PyMySQL (0.9.3)
- schedule (0.6.0)

The modules `daily_data.py` and `aggregate_data.py`, the file `config.py` as well as the script `provide_csv.sh` shall be in the same directory as well as the following subdirectories:

- The directory 'test_orig' containing the CSV files
- An empty subdirectory called 'test_dest' which will receive the files

It is possible to use different names for such directory by editing the file `config.py` and by changing the values of the relevant parameters.

The MySQL database and the EC2 instance whose credentials are mentioned in section 5 are used. For convenience the Linux commands to create the HSS tunnel and to access the database (once in the EC2 server) are copied below. It is possible to change the database credential and to use another MySQL database by editing the parameters of the file `config.py`.

In `/home/user/.ssh/config`:

```
Host AWS_EC2
    User ubuntu
    HostName ssh_server_public_ip
    Port 22
    PreferredAuthentications publickey
    IdentityFile /home/user/.ssh/RDSkey.pem
```

```
$ ssh AWS_EC2
```

```
$ mysql -u admin -p -h maindbname.id.region.rds.amazonaws.com etl
```

Password: `remote_db_user_password`

NOTE: In this version of the file, for security reasons, the value of certain credentials has been hidden. You have to create your own database in order to test this feature.

The figure below shows an example setting with Linux Ubuntu 18.04.

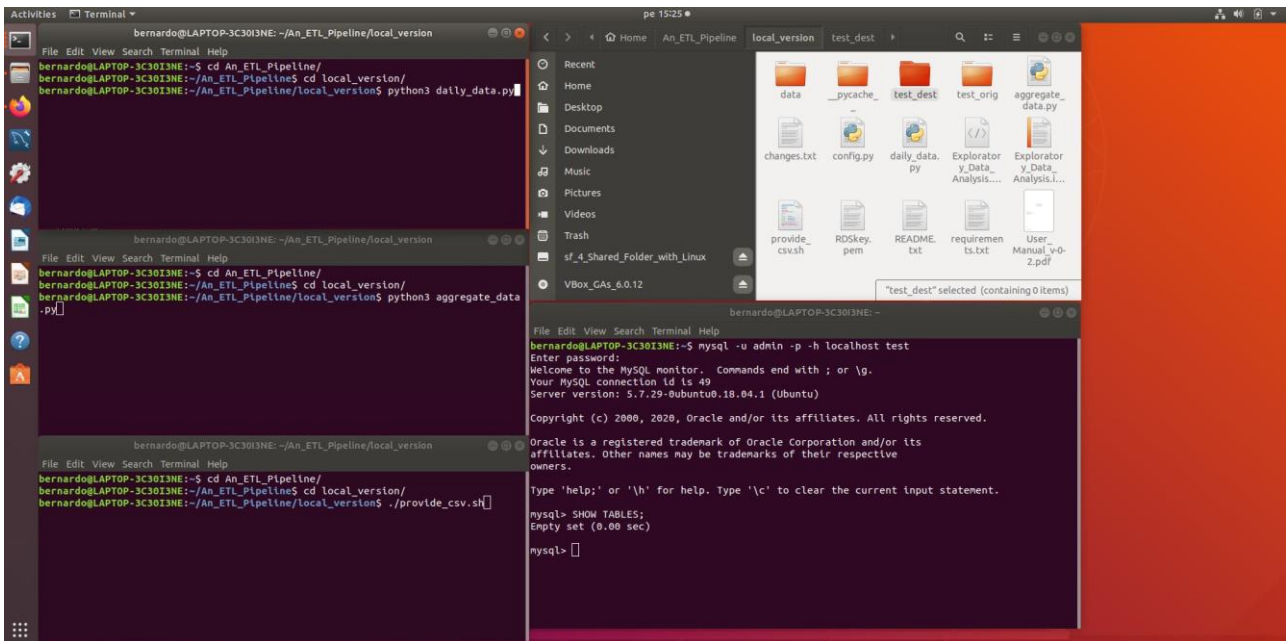


FIG. 2. Running in Linux Ubuntu.

7.3. Test Cases

The following cases have been run on the last version of the SW with a local MySQL database.

The end-to-end test case is documented in section 6.

In this first version of the code, no stress test cases are needed.

7.3.1. TC-1_Tables-Creation_Interrupting-the-Python-Modules

Description

This test case verifies that the following aspects:

- the two tables are correctly created in the database
- the basic daily dump mechanism works
- console feedback works correctly
- interrupting the python modules works correctly

Test Environment

The modules `daily_data.py` and `aggregate_data.py`, the file `config.py` as well as the script `provide_csv.sh` shall be in the same directory as well as the following subdirectories:

- The directory 'test_orig' containing the first 31 CSV files (from July 1st to July 31st)
- An empty subdirectory called 'test_dest' which will receive the files from `provide_csv.sh` and which will be watched by `daily_data.py`

A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal). For the credentials, see section 6.

Preconditions

The database etl contains no tables.

```
DROP TABLE Daily_Dump;  
DROP TABLE Current_Mean_Price;  
SHOW TABLES;
```

Execution

- **Edit** the file `config.py` by setting the parameter `WATCHDOG_DELAY` to 1, the parameter `TESTING` to 'True' and the parameter `LOCAL_TESTING` to 'True'
- In the same file check that `DIRECTORY_TO_WATCH = "test_dest/"`
- Check that the parameter `DELAY` in `provide_csv.sh` is set to 1
- In a separate terminal, while in the dedicated directory, run the `daily_data.py` module
 - `python3 daily_data.py`
- In a separate terminal, while in the dedicated directory, run the shell script
 - `./provide_csv.sh`
- Check the terminal where `daily_data.py` is running and wait till the last file (2018-07-31.csv) has been loaded into the database
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- In a separate terminal, while in the dedicated directory, run the `aggregate_data.py` module
 - `python3 aggregate_data.py`

- Wait 1 minute till you see the message “The aggregate table has been updated at yyyy-mm-dd-hh:mm:ss,xxxxxx” and then check the content of the table ‘Current_Mean_Price’ in the database
 - `SELECT COUNT(id) FROM Current_Mean_Price;`
- Interrupt the module `daily_data.py` by using CTRL+C
- Interrupt the module `aggregate_data.py` by using CTRL+C

Expected Result

- The table ‘Daily_Dump’ contains 310 entries
- The table ‘Current_Mean_Price’ exists and contains 3 entries
- After interrupting the python modules, the following messages are visible in 2 terminals running the 2 python modules:
 - “Monitoring has been interrupted”
 - “Scheduling has been interrupted”

The figures below show what you should see in the terminals

```
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 daily_data.py
Watching the directory: test_dest/
Press CTRL + C to interrupt

New file detected: test_dest/2018-07-01.csv
at 2020-03-06 15:26:20.282086
File loaded into the database

New file detected: test_dest/2018-07-02.csv
at 2020-03-06 15:26:21.289803
File loaded into the database

New file detected: test_dest/2018-07-03.csv
at 2020-03-06 15:26:22.289378
File loaded into the database
```

Clip

```
New file detected: test_dest/2018-07-29.csv
at 2020-03-06 15:26:48.449046
File loaded into the database

New file detected: test_dest/2018-07-30.csv
at 2020-03-06 15:26:49.456645
File loaded into the database

New file detected: test_dest/2018-07-31.csv
at 2020-03-06 15:26:50.461493
File loaded into the database
^C
Monitoring has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$
```

FIG. 3. Terminal running daily.data.py


```

bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ ./provide_csv.sh
Script's start time: 15:26:19
The file 2018-07-01.csv has been copied
The file 2018-07-02.csv has been copied
The file 2018-07-03.csv has been copied
The file 2018-07-04.csv has been copied
The file 2018-07-05.csv has been copied
The file 2018-07-06.csv has been copied
The file 2018-07-07.csv has been copied
The file 2018-07-08.csv has been copied
The file 2018-07-09.csv has been copied
The file 2018-07-10.csv has been copied
The file 2018-07-11.csv has been copied
The file 2018-07-12.csv has been copied
The file 2018-07-13.csv has been copied
The file 2018-07-14.csv has been copied
The file 2018-07-15.csv has been copied
The file 2018-07-16.csv has been copied
The file 2018-07-17.csv has been copied
The file 2018-07-18.csv has been copied
The file 2018-07-19.csv has been copied
The file 2018-07-20.csv has been copied
The file 2018-07-21.csv has been copied
The file 2018-07-22.csv has been copied
The file 2018-07-23.csv has been copied
The file 2018-07-24.csv has been copied
The file 2018-07-25.csv has been copied
The file 2018-07-26.csv has been copied
The file 2018-07-27.csv has been copied
The file 2018-07-28.csv has been copied
The file 2018-07-29.csv has been copied
The file 2018-07-30.csv has been copied
The file 2018-07-31.csv has been copied
Script's end time: 15:26:50
Script's duration:
00:00:31
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

```

FIG. 4. Terminal running provide_csv.sh

```

bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 aggregate_data
.py
The schedules for updating the aggregate table have been set
Press CTRL + C to interrupt
The aggregate table has been updated at 2020-03-06 15:34:00.893309
^C
Scheduling has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

```

FIG. 5. Terminal running aggregate.data.py

```

mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_test |
+-----+
| Daily_Dump      |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|          310 |
+-----+
1 row in set (0.00 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_test      |
+-----+
| Current_Mean_Price |
| Daily_Dump          |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(id) FROM Current_Mean_Price;
+-----+
| COUNT(id) |
+-----+
|           3 |
+-----+
1 row in set (0.00 sec)

mysql>

```

FIG. 6. Terminal running mysql

Postconditions

Once the test is run:

- Edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY back to 60, the parameter TESTING back to 'False' and the parameter LOCAL_TESTING back to 'False'

7.3.2. TC-2_Files-are-Modified-in-the-Watched-Directory

Description

This test case verifies that file modification is not a watched event.

Test Environment

The module `daily_data.py` and the file `config.py` shall be in a dedicated directory. The same directory shall contain the a subdirectory called 'test_dest' which will be watched by `daily_data.py`

A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal). For the credentials, see section 6.

Preconditions

The test case TC-1 has been run.

Execution

- **Edit** the file `config.py` by setting the parameter `WATCHDOG_DELAY` to 1 and the parameter `LOCAL_TESTING` to 'True'
- In the same file check that `DIRECTORY_TO_WATCH = "test_dest/"`
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- In a separate terminal run the `daily_data.py` module
 - `python3 daily_data.py`
- Modify the name of one of the files in the watched directory (for example, change 2018-07-10.csv into 2019-07-10.csv)
- Check the terminal where `daily_data.py` is running and wait 10 seconds
- Check again the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- Interrupt the module `daily_data.py` by using CTRL+C

Expected Result

- `daily_data.py` shows no detected file in the terminal
- The table 'Daily_Dump' contains 310 entries both before and after the modification in the watched directory

The figure below shows what you should see in the terminals.

```
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 daily_data.py
Watching the directory: test_dest/
Press CTRL + C to interrupt
^C
Monitoring has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version/test_dest
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ ls
2018-07-01.csv 2018-07-08.csv 2018-07-15.csv 2018-07-22.csv 2018-07-29.csv
2018-07-02.csv 2018-07-09.csv 2018-07-16.csv 2018-07-23.csv 2018-07-30.csv
2018-07-03.csv 2018-07-10.csv 2018-07-17.csv 2018-07-24.csv 2018-07-31.csv
2018-07-04.csv 2018-07-11.csv 2018-07-18.csv 2018-07-25.csv
2018-07-05.csv 2018-07-12.csv 2018-07-19.csv 2018-07-26.csv
2018-07-06.csv 2018-07-13.csv 2018-07-20.csv 2018-07-27.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ mv 2018-07-10.csv 2019-07-10.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ ls
2018-07-01.csv 2018-07-07.csv 2018-07-14.csv 2018-07-20.csv 2018-07-26.csv 2019-07-10.csv
2018-07-02.csv 2018-07-08.csv 2018-07-15.csv 2018-07-21.csv 2018-07-27.csv
2018-07-03.csv 2018-07-09.csv 2018-07-16.csv 2018-07-22.csv 2018-07-28.csv
2018-07-04.csv 2018-07-11.csv 2018-07-17.csv 2018-07-23.csv 2018-07-29.csv
2018-07-05.csv 2018-07-12.csv 2018-07-18.csv 2018-07-24.csv 2018-07-30.csv
2018-07-06.csv 2018-07-13.csv 2018-07-19.csv 2018-07-25.csv 2018-07-31.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$

bernardo@LAPTOP-3C30I3NE: ~
File Edit View Search Terminal Help
mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|          310 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|          310 |
+-----+
1 row in set (0.00 sec)

mysql>
```

FIG. 7. Console feedback and SQL queries

Postconditions

- Once the test is run, edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY back to 60 and the parameter LOCAL_TESTING back to 'False'

7.3.3. TC-3_Files-are-Deleted-in-the-Watched-Directory

Description

This test case verifies that file deletion is not a watched event.

Test Environment

The module `daily_data.py` and the file `config.py` shall be in a dedicated directory. The same directory shall contain the a subdirectory called 'test_dest' which will be watched by `daily_data.py`

A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal). For the credentials, see section 6.

Preconditions

The test case TC-1 (optionally also TC-2) has been run.

Execution

- **Edit** the file `config.py` by setting the parameter `WATCHDOG_DELAY` to 1 and the parameter `LOCAL_TESTING` to 'True'
- In the same file check that `DIRECTORY_TO_WATCH = "test_dest/"`
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- In a separate terminal run the `daily_data.py` module
 - `python3 daily_data.py`
- Delete one of the files in 'test_dest' directory (for example, the file you have modified in TC-1)
- Check the terminal where `daily_data.py` is running and wait 10 seconds
- Check again the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- Interrupt the module `daily_data.py` by using CTRL+C

Expected Result

- `daily_data.py` shows no detected file in the terminal
- The table 'Daily_Dump' contains 310 entries both before and after the modification in the watched directory

The figure below shows what you should see in the terminals.

```
bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 daily_data.py
Watching the directory: test_dest/
Press CTRL + C to interrupt
^C
Monitoring has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version/test_dest
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ ls
2018-07-01.csv 2018-07-07.csv 2018-07-14.csv 2018-07-20.csv 2018-07-26.csv 2019-07-10.csv
2018-07-02.csv 2018-07-08.csv 2018-07-15.csv 2018-07-21.csv 2018-07-27.csv
2018-07-03.csv 2018-07-09.csv 2018-07-16.csv 2018-07-22.csv 2018-07-28.csv
2018-07-04.csv 2018-07-11.csv 2018-07-17.csv 2018-07-23.csv 2018-07-29.csv
2018-07-05.csv 2018-07-12.csv 2018-07-18.csv 2018-07-24.csv 2018-07-30.csv
2018-07-06.csv 2018-07-13.csv 2018-07-19.csv 2018-07-25.csv 2018-07-31.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ rm 2019-07-10.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ ls
2018-07-01.csv 2018-07-06.csv 2018-07-12.csv 2018-07-17.csv 2018-07-22.csv 2018-07-27.csv
2018-07-02.csv 2018-07-07.csv 2018-07-13.csv 2018-07-18.csv 2018-07-23.csv 2018-07-28.csv
2018-07-03.csv 2018-07-08.csv 2018-07-14.csv 2018-07-19.csv 2018-07-24.csv 2018-07-29.csv
2018-07-04.csv 2018-07-09.csv 2018-07-15.csv 2018-07-20.csv 2018-07-25.csv 2018-07-30.csv
2018-07-05.csv 2018-07-11.csv 2018-07-16.csv 2018-07-21.csv 2018-07-26.csv 2018-07-31.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$

bernardo@LAPTOP-3C30I3NE: ~
File Edit View Search Terminal Help
mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|        310 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|        310 |
+-----+
1 row in set (0.00 sec)

mysql>
```

FIG. 8. Consolle feedback and SQL queries

Postconditions

- Once the test is run, edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY back to 60 and the parameter LOCAL_TESTING back to 'False'

7.3.4. TC-4_Directories-are-Added-in-the-Watched-Directory

Description

This test case is similar to the previous two cases and verifies that sub-directory addition is not a watched event.

Test Environment

The module `daily_data.py` and the file `config.py` shall be in a dedicated directory. The same directory shall contain the a subdirectory called 'test_dest' which will be watched by `daily_data.py`

A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal). For the credentials, see section 6.

Preconditions

The test case TC-1 (optionally also TC-2 and TC-3) has been run.

Execution

- **Edit** the file `config.py` by setting the parameter `WATCHDOG_DELAY` to 1 and the parameter `LOCAL_TESTING` to 'True'
- In the same file check that `DIRECTORY_TO_WATCH = "test_dest/"`
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- In a separate terminal run the `daily_data.py` module
 - `python3 daily_data.py`
- Add a subdirectory in the 'test_dest' directory (`mkdir temp`) and wait 10 seconds
- Check again the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- Interrupt the module `daily_data.py` by using CTRL+C

Expected Result

The table 'Daily_Dump' still contains 310 entries

Postconditions

- Once the test is run, edit the file `config.py` by setting the parameter `WATCHDOG_DELAY` back to 60 and the parameter `LOCAL_TESTING` back to 'False'

The figure below shows what you should see in the terminals.


```
bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 daily_data.py
Watching the directory: test_dest/
Press CTRL + C to interrupt
^C
Monitoring has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version/test_dest
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ ls
2018-07-01.csv 2018-07-07.csv 2018-07-14.csv 2018-07-20.csv 2018-07-26.csv 2019-07-10.csv
2018-07-02.csv 2018-07-08.csv 2018-07-15.csv 2018-07-21.csv 2018-07-27.csv
2018-07-03.csv 2018-07-09.csv 2018-07-16.csv 2018-07-22.csv 2018-07-28.csv
2018-07-04.csv 2018-07-11.csv 2018-07-17.csv 2018-07-23.csv 2018-07-29.csv
2018-07-05.csv 2018-07-12.csv 2018-07-18.csv 2018-07-24.csv 2018-07-30.csv
2018-07-06.csv 2018-07-13.csv 2018-07-19.csv 2018-07-25.csv 2018-07-31.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ rm 2019-07-10.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$ ls
2018-07-01.csv 2018-07-06.csv 2018-07-12.csv 2018-07-17.csv 2018-07-22.csv 2018-07-27.csv
2018-07-02.csv 2018-07-07.csv 2018-07-13.csv 2018-07-18.csv 2018-07-23.csv 2018-07-28.csv
2018-07-03.csv 2018-07-08.csv 2018-07-14.csv 2018-07-19.csv 2018-07-24.csv 2018-07-29.csv
2018-07-04.csv 2018-07-09.csv 2018-07-15.csv 2018-07-20.csv 2018-07-25.csv 2018-07-30.csv
2018-07-05.csv 2018-07-11.csv 2018-07-16.csv 2018-07-21.csv 2018-07-26.csv 2018-07-31.csv
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version/test_dest$

bernardo@LAPTOP-3C30I3NE: ~
File Edit View Search Terminal Help
mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|        310 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|        310 |
+-----+
1 row in set (0.00 sec)

mysql>
```

• FIG. 9. Console feedback and SQL queries

7.3.5. TC-5_Aggregate-Data_Handling-Missing-Values_New-File-with-Same-Name

Description

This test case verifies the following issues:

- the code correctly calculates aggregate data
- the code can cope with missing values in the price column
- replacing a file with a file with the same name in the watched directory does not trigger new entries in the database
- when there are no new entries in the Daily_Dump' table, the content of the 'Current_Mean_Price' table does not change

The actual values of the aggregate data columns can be verified with an independent calculation (in this test case there are only 30 entries containing price data and those are the entries related to dates from August 31st to September 2nd)

Test Environment

The modules [daily_data.py](#) and [aggregate_data.py](#), the file [config.py](#) as well as the script [provide_csv.sh](#) shall be in the same directory as well as the following subdirectories:

- The directory 'test_orig' containing the 64 CSV files from July 1st to September 2nd
- The subdirectory called 'test_dest' which will receive the files from [provide_csv.sh](#) and which will be watched by [daily_data.py](#). This directory contains already the first 31 CSV files (from July 1st to July 31st)
- A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal). For the credentials, see section 6.

Preconditions

- The table 'Daily_Dump' contains 310 entries
- The test case TC-1 has been run
- Note: if TC-2 and TC-3 have been run after TC-1, it is important to reset the files in the 'test_dest' directory as they were after TC-1.

Execution

- **Edit** the file [config.py](#) by setting the parameter WATCHDOG_DELAY to 1, the parameter TESTING to 'True' and the parameter LOCAL_TESTING to 'True'
- In the same file check that DIRECTORY_TO_WATCH = "test_dest/"
- Check that the parameter DELAY in [provide_csv.sh](#) is set to 1
- In a separate terminal run the [daily_data.py](#) module
 - `python3 daily_data.py`
- In a separate terminal run the shell script
 - `./provide_csv.sh`
- Check the terminal where [daily_data.py](#) is running and verify that the first file that is detected (2018-08-01.csv). Then wait till the last file (2018-09-02.csv) has been loaded into the database
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- In a separate terminal run the [aggregate_data.py](#) module
 - `python3 aggregate_data.py`

- Check the terminal where `aggregate_data.py` is running and wait till the following message is shown: “The aggregate table has been updated at yyyy-mm-dd-hh:mm:ss.xxxxxx”
- Check the content of the table ‘Current_Mean_Price’ in the database
 - `SELECT * FROM Current_Mean_Price;`
- Wait for one minute that a new message appears in the terminal where `aggregate_data.py` is running and check again the content of the ‘Current_Mean_Price’ table
- Interrupt the module `aggregate_data.py` by using CTRL+C
- Interrupt the module `daily_data.py` by using CTRL+C

Expected Result

- The table ‘Daily_Dump’ contains 640 entries
- The table ‘Current_Mean_Price’ contains the values contained in FIG. 8. in the ‘mean_price’ column
- The content of the ‘Current_Mean_Price’ table does not change after one minute

The figure below shows what you should see in the terminals.

The figure consists of two terminal window screenshots. The top window shows the execution of `python3 daily_data.py`. It displays messages for detecting and loading four CSV files from the `test_dest/` directory into the database, with timestamps for each file. The bottom window shows the execution of `python3 aggregate_data.py`. It displays messages indicating that the schedules for updating the aggregate table have been set, and that the table has been updated twice with specific timestamps. The process is then interrupted with `^C`, resulting in a 'Scheduling has been interrupted' message.

```

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 daily_data.py
Watching the directory: test_dest/
Press CTRL + C to interrupt

New file detected: test_dest/2018-08-01.csv
at 2020-03-06 16:48:59.942488
File loaded into the database

New file detected: test_dest/2018-08-02.csv
at 2020-03-06 16:49:00.962790
File loaded into the database

New file detected: test_dest/2018-08-03.csv
at 2020-03-06 16:49:01.970021
File loaded into the database

New file detected: test_dest/2018-08-04.csv
at 2020-03-06 16:49:02.955081
File loaded into the database

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 aggregate_data
.py
The schedules for updating the aggregate table have been set
Press CTRL + C to interrupt
The aggregate table has been updated at 2020-03-06 16:51:00.869925
The aggregate table has been updated at 2020-03-06 16:52:00.866346
^C
Scheduling has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

```

FIG. 10. Console feedback for the python modules.


```
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ ./provide_csv.sh
Script's start time: 16:48:28
The file 2018-07-01.csv has been copied
The file 2018-07-02.csv has been copied
The file 2018-07-03.csv has been copied
The file 2018-07-04.csv has been copied
The file 2018-07-05.csv has been copied
```

clip

```
The file 2018-08-29.csv has been copied
The file 2018-08-30.csv has been copied
The file 2018-08-31.csv has been copied
The file 2018-09-01.csv has been copied
The file 2018-09-02.csv has been copied
Script's end time: 16:49:32
Script's duration:
00:01:04
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$
```

FIG. 11. Console feedback for the shell script.

```

mysql> SELECT * FROM Current_Mean_Price;
+-----+-----+-----+
| id | temperature_group | mean_price |
+-----+-----+-----+
| 1 | LOW | NULL |
| 2 | MEDIUM | NULL |
| 3 | HIGH | NULL |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
| 310 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
| 640 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM Current_Mean_Price;
+-----+-----+-----+
| id | temperature_group | mean_price |
+-----+-----+-----+
| 4 | LOW | 107378.89971354131 |
| 5 | MEDIUM | 38763.76077792506 |
| 6 | HIGH | 20359.37687755542 |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
mysql> SELECT * FROM Current_Mean_Price;
+-----+-----+-----+
| id | temperature_group | mean_price |
+-----+-----+-----+
| 7 | LOW | 107378.89971354131 |
| 8 | MEDIUM | 38763.76077792506 |
| 9 | HIGH | 20359.37687755542 |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> 

```

FIG. 12. SQL queries.

Postconditions

Once the test is run:

- Edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY back to 60, the parameter TESTING back to 'False' and the parameter LOCAL_TESTING back to 'False'

7.3.6. TC-6_Full-Dump_TESTING-is-False

Description

This test case verifies that everything works fine also when the parameter TESTING is set to false in [config.py](#)

Also, it allows to check the result after dumping the complete list of csv files that have been provided.

Test Environment

The modules [daily_data.py](#) and [aggregate_data.py](#), the file [config.py](#) as well as the script [provide_csv.sh](#) shall be in the same directory as well as the following subdirectories:

- The directory 'test_orig™' containing all the 165 CSV files (from July 1st to December 12nd)
- The subdirectory called 'test_dest' which will receive the files from [provide_csv.sh](#) and which will be watched by [daily_data.py](#)
- A connection to the database through an SSH connection to the EC2 server (for example, from a Linux terminal). For the credentials, see section 6.

Preconditions

- The table 'Daily_Dump' contains 640 entries
- The test case TC-1 and TC-5 have been run
- In [config.py](#) TESTING = 'False'

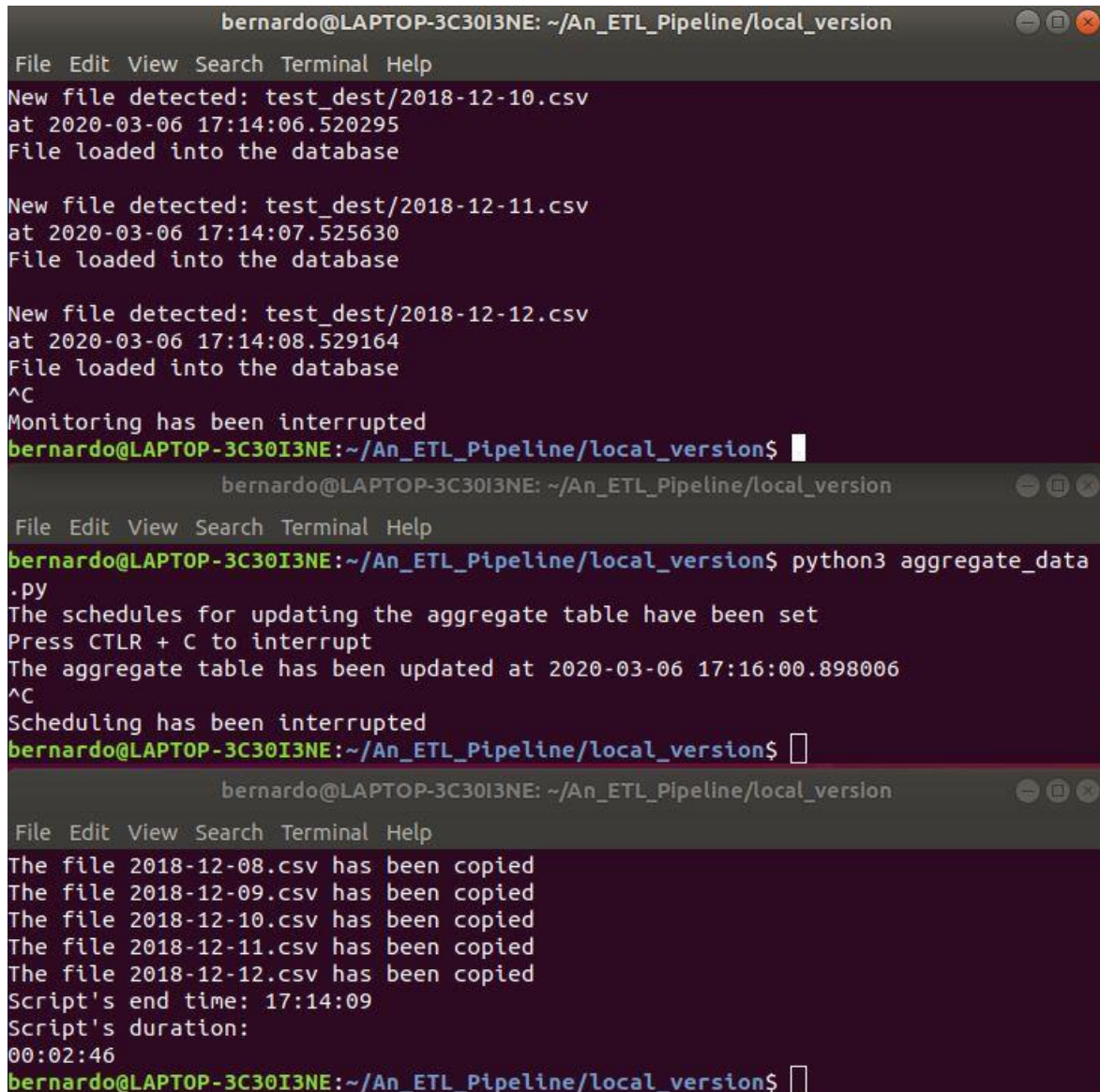
Execution

- Edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY to 1
- In the same file check that DIRECTORY_TO_WATCH = "test_dest/" and LOCAL_TESTING = True
- Check that the parameter DELAY in [provide_csv.sh](#) is set to 1
- In a separate terminal run the [daily_data.py](#) module
 - `python3 daily_data.py`
- In a separate terminal run the shell script
 - `./provide_csv.sh`
- Check the terminal where [daily_data.py](#) is running and wait till the last file (2018-12-12.csv) has been loaded into the database
- Check the content of the table 'Daily_Dump' in the database
 - `SELECT COUNT(id) FROM Daily_Dump;`
- Now edit the file [config.py](#) by setting the parameter TIME at the current hour and a couple of minutes ahead
- In a separate terminal run the [aggregate_data.py](#) module
 - `python3 aggregate_data.py`
- Check the terminal where [aggregate_data.py](#) is running and wait till the following message is shown: "The aggregate table has been updated at yyyy-mm-dd-hh:mm:ss.xxxxxx"
- Check the content of the table 'Current_Mean_Price' in the database
 - `SELECT * FROM Current_Mean_Price;`
- Interrupt the module [aggregate_data.py](#) by using CTRL+C
- Interrupt the module [daily_data.py](#) by using CTRL+C

Expected Result

- The table 'Daily_Dump' contains 1650 entries
- The table 'Current_Mean_Price' contains the values contained in FIG. 15. in the 'mean_price' column

The figure below shows what you should see in the terminals



```
bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
New file detected: test_dest/2018-12-10.csv
at 2020-03-06 17:14:06.520295
File loaded into the database

New file detected: test_dest/2018-12-11.csv
at 2020-03-06 17:14:07.525630
File loaded into the database

New file detected: test_dest/2018-12-12.csv
at 2020-03-06 17:14:08.529164
File loaded into the database
^C
Monitoring has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$ python3 aggregate_data
.py
The schedules for updating the aggregate table have been set
Press CTRL + C to interrupt
The aggregate table has been updated at 2020-03-06 17:16:00.898006
^C
Scheduling has been interrupted
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$

bernardo@LAPTOP-3C30I3NE: ~/An_ETL_Pipeline/local_version
File Edit View Search Terminal Help
The file 2018-12-08.csv has been copied
The file 2018-12-09.csv has been copied
The file 2018-12-10.csv has been copied
The file 2018-12-11.csv has been copied
The file 2018-12-12.csv has been copied
Script's end time: 17:14:09
Script's duration:
00:02:46
bernardo@LAPTOP-3C30I3NE:~/An_ETL_Pipeline/local_version$
```

FIG. 13. Console feedback.

```
mysql> SELECT COUNT(id) FROM Daily_Dump;
+-----+
| COUNT(id) |
+-----+
|      1650 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT * FROM Current_Mean_Price;
+-----+-----+-----+
| id | temperature_group | mean_price |
+-----+-----+-----+
| 10 | LOW               | 37569.44525745997 |
| 11 | MEDIUM           | 29441.747246939583 |
| 12 | HIGH              | 37697.31742568427 |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> 
```

FIG. 14. SQL queries.

Postconditions

Once the test is run:

- Edit the file [config.py](#) by setting the parameter WATCHDOG_DELAY back to 60, the parameter TIME back to '00:00' and the parameter LOCAL_TESTING back to 'False'

8. Possible Future Improvements

The current version of the pipeline is just a first beta version.

8.1. Issues for Customer Feedback

The first thing to be done is to get customer feedback. Also, the following questions should be asked from the customer:

- Is it possible to get more background information about the exact meaning of all the variables?
- Confirmation that 'at_risk' can have only two values
- Confirmation that 'skipped_beat' has only integer values
- Confirming that it is possible to rely on the correct naming of the CSV files
- Confirmation that no additional variables are foreseen in the future, in addition to 'temperature', 'at_risk', 'skipped_beat' and 'price'
- Confirmation that the wished aggregate data is regarding the last 30 calendar days (rather than the calendar month)
- Elaborating if the amount of collected CSV files per day is limited to 1, both now and in the future
- Elaborating if the amount of entries per CSV file is limited to 10, both now and in the future

Based on the answers to those question, modifications to the code might be needed.

8.2. Backlog Items

The following possible enhancements (in decreasing priority order) have already been identified.

Issues which are specific to the database configuration and connection to the database are in italic.

In addition to what listed below, it is worth to add that automatizing the test cases would increase efficiency when modifying the code (creating regression tests).

- ROBUSTNESS
 - Finding a solution that verifies that the input data for calculating aggregate data is from the last 30 days (in case the number of entries for each day changes): in `aggregate_data.py` in the SQL command that creates a temporary table (sql2), rather than using the LIMIT statement, using a WHERE statement which checks the value of the date column. This would also allow to change the logic to aggregate data on a calendar month basis.
 - The solution could be more tolerant to schema changes and support new unforeseen columns (may be by using SQLAlchemy)
 - An if condition could be added to the event Handler in `daily_data.py` to filter the call to the function fetch (which reads the file) only for csv files (may be also some check of the data format might be added)
- EFFICIENCY
 - Using Apache Airflow for the orchestration
- COMPATIBILITY
 - Using docker compose for testing
- SCALABILITY
 - Adding iteration loops by using id ranges In `aggregate_data.py` in the query that creates the temporary table (sql2).
- AVAILABILITY
 - *In the AWS RDS instance the 'Backup retention period' shall be increased and /or automatic snapshots shall be set*

- USABILITY
 - It would be useful to add the column 'time_stamp' to the 'Current_Mean_Price' table containing the day and time when the aggregate data has been calculated.
- SCALABILITY (medium term)
 - Should the size of the files grow, at the moment it is possible to increase manually the value of the parameter WATCHDOG_DELAY in [daily_data.py](#). A more scalable solution would be to send the content of the file to a AWS SQS queue and then pull the content from there into the SQL database
 - *In the AWS RDS instance:*
 - *'DB instance class' and 'Allocated storage' shall be upgraded as needed*
 - *'Storage autoscaling' shall be enabled*
- SCALABILITY (long term)
 - Should the amount of data increase dramatically, a more scalable solution using for example Apache Spark is to be considered. Also, should the amount of data to be stored grow over 1 terabyte, a Big Data storage solution like, for example, Snowflake should be considered

9. Annex 1: AWS RDS and EC2 Instance Configuration

NOTE: In this version of the file, for security reasons, the value of certain credentials has been hidden. You have to create your own database in order to test this feature.

One EC2 instance has been created in Amazon Web Service (AWS) <https://aws.amazon.com/> to connect to the database through SSH:

- Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0fc20dd1da406780b (64-bit x86)
- Contains python 3.6.9
- t2micro (1 CPU 2.5GHz, 1 GB)
- id: i-xxxxxxxxxxxxxxxxxx
- User name: **ubuntu**
- Auto-assign public IP enabled
- Public IP address **ssh_server_public_ip**
- Public DNS (IPv4): ec2-xxx-xxx-xxx-xxx.region.compute.amazonaws.com
- Private IP: **hss_server_private_ip**
- Network: vpc-vpc__id
- Subnet: default (subnet-xxxxxxx)
- Shutdown behavior: stop
- Termination protection disabled
- CloudWatch detailed monitoring disabled
- Tenacy: Shared - Run a shared hardware instance
- No elastic inference accelerator
- T2/T3 unlimited disabled
- Storage: Root, /dev/sda1, 8 G, General Purpose SSD, IOPS 100/3000
- Security group:
 - Access to the SSH server by everyone (by SSH key): TCP **22** 0.0.0.0/0 (SSH access by SSH key by everyone)
- Key pair name: RDSkey
- Private key file: **RDSkey.pem**

A MySQL database has been created in Amazon Web Service (AWS) <https://aws.amazon.com/> by using the same VPC:

- Engine type: MySQL
- Engine version: 5.7.22
- Templates: Free tier ¹
- DB instance identifier: etl
- **Master user: admin**
- **Password: remote_db_user_password**
- **Endpoint: maindbname.id.region.rds.amazonaws.com**
- Database port: 3306
- **Database name: etl**
- DB instance class: db.t2.micro (1 vCPU, 1GiB RAM)
- Storage type: General purpose SSD
- Allocated storage: 20 GiB
- Storage autoscaling: disabled
- Multi-AZ deployment: No
- VPC: Default VPC (vpc-vpc__id)
- Subnet group: default-vpc__id
- Public accessibility: No
- Security group:
 - SSH to DB: TCP 3306 hss_server_private_ip/32 (MySQL access from the EC2 instance)
- Availability zone: no preferences
- IAM DB authentication: disabled
- DB parameter group: default.mysql5.7
- Option group: default:mysql-5-7
- Automatic back up enabled
- Backup retention period: 1 days
- Backup window Start Time: 01:11 UTC
- Backup window Duration: 1 hours
- Copy tags to snapshots: yes
- Enhanced monitoring enabled
- Granularity: 60 seconds
- Monitoring Role: default
- Log exports disabled
- Auto minor version upgrade: enabled
- Maintenance window Start Day: Sunday
- Maintenance window Start Time: 00:04 UTC
- Maintenance window duration: 1 hours
- Deletion protection: enabled

¹ Limits for free usage:

- instance class: db.t2.micro
- 20 Gb General Purpose SSD
- 20 Gb for backup and snapshots
- 750 hours / month