Data Lake

REVIEW

CODE REVIEW 8

HISTORY

Meets Specifications

Dear student,

Thank you for the extremely extraordinary effort you have put into this project. We do appreciate your hard work!
Now, you have indeed mastered **data lake and data streams with spark** which is a much-needed skill nowadays!
I wish you the best of luck in your future endeavors!

I've added `some extra tips for improvements at each rubric item` and also at the `Code Review` tab, kindly, check them and apply as you see appropriate.
We look forward to no less than this great submission for the upcoming submissions.

`FURTHER READINGs`

- REFERENCE #1: **Data Lake vs Data Warehouse.**
- REFERENCE #2: **Data Lake - how to enable Advanced Analytics and Machine Learning**.
- REFERENCE #3: **ETL vs ELT explained.**

`GitHub`

If you didn't add the project to `GitHub` then you probably want to.
Adding your `project` to `GitHub` will really pay off at some point of time. Take it from me, one of the reasons I got my current role is that I kept a clean and organized `GitHub` to attract potential employees all the way.
Read this **short article on why you should be adding your project to `GitHub`.**

`KNOWLEDGE`

If you ever had any questions or you happen to face any future issues .. kindly refer to Knowledge platform seeking help and your questions will be answered by me or one of my fellow mentors as soon as published.

`STAY SAFE`

Stay safe and take care of yourself and all your beloved ones during these pandemic times.

ETL

The script, etl.py, runs in the terminal without errors. The script reads song_data and load_data from S3, transforms them to create five different tables, and writes them to partitioned parquet files in table directories on S3.

RUBRIC ITEM MEETS SPECIFICATION

The script runs perfectly with no `interpretation(1)` or `run-time(2)` errors. Excellent job.

REFERENCES

- REFERENCE #1: **Difference between interpretation and compilation.**

- REFERENCE #2: **Definition of run-time error.**

Each of the five tables are written to parquet files in a separate analytics directory on S3. Each table has its own folder within the directory. Songs table files are partitioned by year and then artist. Time table files are partitioned by year and month. Songplays table files are partitioned by year and month.

RUBRIC ITEM MEETS SPECIFICATION

- Tables are written to `parquet files`.
  Speaking of which, here is an interesting article about `parquet` files.
- `Songs table` files are partitioned by **year and then artist**. `Time table` files are partitioned by **year and month**. `Songplays table` files are partitioned by **year and month**.
  Here is my favorite article that explains `partitioning` and **why do we actually commit partitioning**?

Good job!

Each table includes the right columns and data types. Duplicates are addressed where appropriate.

RUBRIC ITEM MEETS SPECIFICATION

- You chose the right columns for each table.
- You also handled `duplicates` when appropriate.

Here is an article that shows **10 reasons why you must not have any duplicates in your database**.

Dropping duplicates tip: What is the difference between `distinct()` | `drop_duplicates()` and `dropDuplicates()`

`dropDuplicates()` can take an input which is the column name upon which you will decide whether the value is a duplicate or not. `distinct()` takes no inputs. The `drop_duplicates()` works in the same manner as `distinct()`.

Code Quality

The README file includes a summary of the project, how to run the Python scripts, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.

RUBRIC ITEM MEETS SPECIFICATION

Dear student,

To pass this rubric item, two aspects are being evaluated, the `README.md` file and the addition of `docstrings`.
You have put a great effort into your submission and met both aspects. Excellent job!

`Firstly: README.md`

You've created an informative `README.md` file that indeed contains the **minimum specifications.**

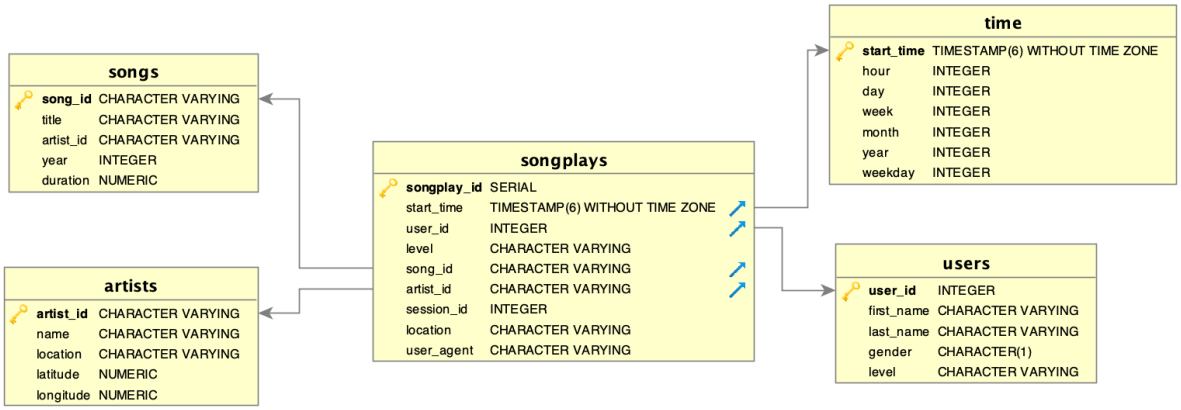
MINIMAL REQUIREMENTS:

There are three minimal aspects of the README file:

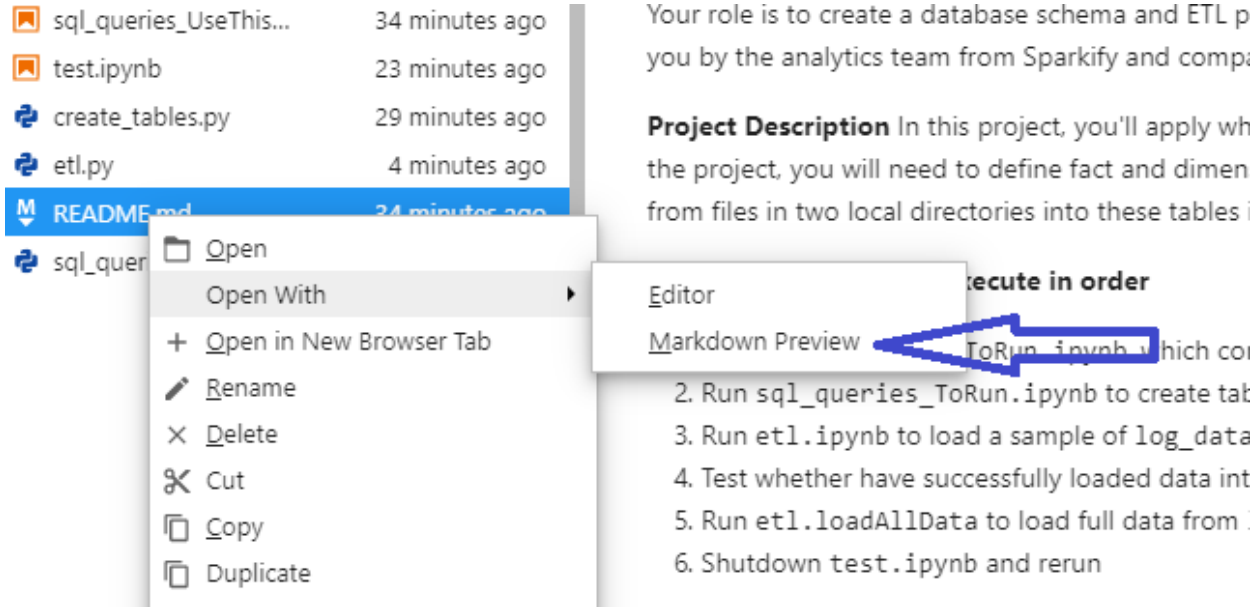
- It **contains a summary of the project.**
- It **explains all the files in the repository.**
- It mentions in detail **how to run the project.**

IMPROVEMENTS AND TIPS

- Always take care of grammatical and structural errors, like capitalization of the first letter.
  I also would highly suggest you use any online grammatical checker tool **(like Grammarly)**.
- Add a screenshot or an image (ER Diagram) showing how the fact and dimension tables are connected.
  Example:



- You can use an online markdown editor like this one.
- You can preview your README in your Udacity workplace as below:



- `READMEs` references:
  - This is a list of great `READMEs` on `GitHub`.
  - This is an article on how to write a good `README` for your `GitHub` project.
  - Medium article: Guide on how to write `READMEs`.

`Secondly: Docstrings`

Excellent job adding `docstrings` to your function.

TIP FOR `DOCSTRINGS`

- Did you know, you can create automated documentation to your code **by adding docstrings to functions**. We are using that a lot in my team every day at work! Check this. It's a very informative article 😄

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

RUBRIC ITEM MEETS SPECIFICATION

- Your **project code is clean** and for the most part, follows the `PEP8 style guidelines`.
- I can clearly see your code structured into logical functions. Your function names clearly specify what your code is going to do. Great effort here!

Tip for distinguishment: Make use of the `pycodestyle`

- `pycodestyle` is a tool to check your Python code against some of the style conventions in `PEP 8`.
- Have a look at the official documentation of the `pycodestyle`, here - **[Trust me, running it is a five minutes job!]**.
- To install and use it, just execute these commands in your **workspace terminal**:

```
pip install pycodestyle
pycodestyle etl.py
```

- One of the `PEP8` guidelines is not to have lines longer than `79 characters [which happens a lot in everyone’s code while coding in python!]`, but there are cases where your lines are somewhere around `80 ~ 85 characters`. In this particular case, the `pycodestyle` takes an argument called `--max-line-length` that will get rid of all the `long lines warnings` that you will get running your script. **But be careful with that, don't ignore too long lines!**