**Project Review**

**Meets Specifications**

Greetings Learner  
Congratulations on passing your project!  This is indeed very impressive and you should be proud of yourself! All the specifications of this project was met accurately and successfully! You have shown great determination and effort throughout the project and I want you to know that we appreciate it very much! It also shows that you have a good understand in this field now. Hope you keep up the good work and wish you all the best for your future.

**Extra Materials**

Below are some additional links to help you deepen your understanding on the related concepts:

* [Airflow: Lesser Known Tips, Tricks, and Best Practises](https://medium.com/datareply/airflow-lesser-known-tips-tricks-and-best-practises-cf4d4a90f8f)
* [Airflow Tips, Tricks & Pitfalls](https://caserta.com/data-blog/airflow-tips-tricks-pitfalls/)
* [Getting started with Apache Airflow](https://towardsdatascience.com/getting-started-with-apache-airflow-df1aa77d7b1b)
* [DAG Writing Best Practices in Apache Airflow](https://www.astronomer.io/guides/dag-best-practices/)

**General**

**DAG can be browsed without issues in the Airflow UI**

Awesome! DAG can be browsed without issues in the Airflow UI.

**Suggestions**

To make the DAG even more compact, you could try to use the [SubDag](https://www.astronomer.io/guides/subdags" \t "_blank) operator with the dimension loads and hide the repetitive parts behind that. Depending on your set up, using a **subdag** operator could make your DAG cleaner.

**The dag follows the data flow provided in the instructions, all the tasks have a dependency and DAG begins with a start\_execution task and ends with a end\_execution task.**

Excellent work!  The DAG’s graph view all the task have a dependency and DAG begins with astart\_execution task and ends with a end\_execution task.

**Pro Tip**

To set groups of dependencies, you can use lists or tuples as well.

**For example:**

d1 >> d2 >> (d3, d4)

d1 >> d2 >> [d3, d4]

**Suggestions**  
Dependencies are one of Airflow's most powerful and popular features - they allow for previously long, brittle jobs to be broken down into granular parts that are safer, more modular, and reusable. You can check this [link](https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/external_task_sensor.html) to know more about **Managing Dependencies in Apache Airflow**.

**Dag configuration**

**DAG contains default\_args dict, with the following keys:**

* **Owner**
* **Depends\_on\_past**
* **Start\_date**
* **Retries**
* **Retry\_delay**
* **Catchup**

Good job defining the default\_args dictionary as required.   
If a dictionary of default\_args is passed to a DAG, it will apply them to any of its operators. This makes it easy to apply a common parameter to many operators without having to type it many times.  
External Resources  
[Backfill and Catchup](https://airflow.apache.org/docs/apache-airflow/stable/scheduler.html#backfill-and-catchup)

**The DAG object has default args set**

Nice work! The DAG object contains a binding to the default args.

**The DAG should be scheduled to run once an hour**

Good work scheduling the DAG to run once an hour as required.

**Learning Notes**

A DAG Run is an object representing an instantiation of the DAG in time. Each DAG may or may not have a schedule, which informs how DAG Runs are created. schedule\_interval is defined as a DAG argument, which can be passed a cron expression as a str, a datetime.timedelta object, or one of of the following cron "presets". You may refer to this DAG Runs documentation for more details on scheduling a DAG.

**External Resources**

* [Airflow Scheduling & Triggers](https://airflow.apache.org/scheduler.html)
* [Scheduling Tasks in Airflow](https://www.astronomer.io/guides/scheduling-tasks/)

**Staging the data**

**There is a task that to stages data from S3 to Redshift. (Runs a Redshift copy statement)**

Nice work  the stage operator contains a template field that allows it to load timestamped files from S3 based on the execution time and run backfills.

**Extra Materials**

[Templating and Macros in Airflow](https://www.astronomer.io/guides/templating)  
[How to add template variable in the Operator task](https://stackoverflow.com/questions/46645001/how-to-add-template-variable-in-the-filename-of-an-emailoperator-task-airflow)

**Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically**

Good job dynamically generating copy statements using params as opposed to static SQL statements.

**The operator contains logging in different steps of the execution**

logging.info shows the status of staging load. Nice work!

**Suggestions**

Here is a nice documentation about [Logging in Airflow](https://www.astronomer.io/guides/logging)  
A nice discussion about [Adding logs to Airflow Logs](https://stackoverflow.com/questions/40120467/adding-logs-to-airflow-logs)

**The SQL statements are executed by using a Airflow hook**

Good job dynamically generating copy statement using params as opposed to static SQL statements.

**Suggestions**

Note that Hooks are interfaces to external platforms and databases, implementing a common interface when possible and acting as building blocks for operators. Check out the following links for more info.

* [Automate AWS Tasks using Airflow Hooks](https://www.sicara.ai/blog/2019-01-28-automate-aws-tasks-thanks-to-airflow-hooks)
* [Source code for airflow.hooks.S3\_hook](https://airflow.readthedocs.io/en/1.9.0/_modules/airflow/hooks/S3_hook.html)

**Loading dimensions and facts**

**Dimensions are loaded with on the LoadDimension operator**

Nice work the stage operator contains a template field that allows it to load timestamped files from S3 based on the execution time and run backfills.

**Facts are loaded with on the LoadFact operator**

There is a separate functional operator for facts **LoadFactOperator** as well.

**Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically**

The parameters were used to add some dynamic functionality to the operators and allow to run various SQL statements instead of hardcoded SQL statement. Well done!

**The DAG allows to switch between append-only and delete-load functionality**

Nice! The DAG allows to switch between append-only and delete-load functionality.

**Suggestions**  
Dimension loads are often done with the truncate-insert pattern where the target table is emptied before the load. Thus, having a parameter that allows switching between insert modes when loading dimensions. Fact tables are usually so massive that they should only allow append type functionality. That is why there is a need to check for parameter value passed and if it says delete before insertion, delete the values from table before insertion or append otherwise.

**Data Quality Checks**

**Data quality check is done with correct operator**

Great work! The operator that runs a check on the fact or dimension table(s) after the data has been loaded is **DataQualityOperator**.

**The DAG either fails or retries n times**

Correct!  Exceptions are raised in case the expected result was not met, this will fail the task and will lead to retry logic as per the config passed to the dag.

**Operator uses params to get the tests and the results, tests are not hard coded to the operator**

Nice work! The Operator uses params to get the tests and the results and you've passed an array of checks and runs them in a loop. Well done!  Proper amendments made.