**Data Lineage**

**Definition**

The data lineage of a dataset describes the discrete steps involved in the creation, movement, and calculation of that dataset.

**Why is Data Lineage important?**

1. **Instilling Confidence:** Being able to describe the data lineage of a particular dataset or analysis will build confidence in data consumers (engineers, analysts, data scientists, etc.) that our data pipeline is creating meaningful results using the correct datasets. If the data lineage is unclear, its less likely that the data consumers will trust or use the data.
2. **Defining Metrics:** Another major benefit of surfacing data lineage is that it allows everyone in the organization to agree on the definition of how a particular metric is calculated.
3. **Debugging:** Data lineage helps data engineers track down the root of errors when they occur. If each step of the data movement and transformation process is well described, it's easy to find problems when they occur.

In general, data lineage has important implications for a business. Each department or business unit's success is tied to data and to the flow of data between departments. For e.g., sales departments rely on data to make sales forecasts, while at the same time the finance department would need to track sales and revenue. Each of these departments and roles depend on data, and knowing where to find the data. Data flow and data lineage tools enable data engineers and architects to track the flow of this large web of data.

**Schedules**

Pipelines are often driven by schedules which determine what data should be analyzed and when.

**Why Schedules**

* Pipeline schedules can reduce the amount of data that needs to be processed in a given run. It helps scope the job to only run the data for the time period since the data pipeline last ran. In a naive analysis, with no scope, we would analyze all of the data at all times.
* Using schedules to select only data relevant to the time period of the given pipeline execution can help improve the quality and accuracy of the analyses performed by our pipeline.
* Running pipelines on a schedule will decrease the time it takes the pipeline to run.
* An analysis of larger scope can leverage already-completed work. For. e.g., if the aggregates for all months prior to now have already been done by a scheduled job, then we only need to perform the aggregation for the current month and add it to the existing totals.

**Selecting the time period**

Determining the appropriate time period for a schedule is based on a number of factors which you need to consider as the pipeline designer.

1. **What is the size of data, on average, for a time period?** If an entire years worth of data is only a few kb or mb, then perhaps its fine to load the entire dataset. If an hours worth of data is hundreds of mb or even in the gbs then likely you will need to schedule your pipeline more frequently.
2. **How frequently is data arriving, and how often does the analysis need to be performed?** If our bikeshare company needs trip data every hour, that will be a driving factor in determining the schedule. Alternatively, if we have to load hundreds of thousands of tiny records, even if they don't add up to much in terms of mb or gb, the file access alone will slow down our analysis and we’ll likely want to run it more often.
3. **What's the frequency on related datasets?** A good rule of thumb is that the frequency of a pipeline’s schedule should be determined by the dataset in our pipeline which requires the most frequent analysis. This isn’t universally the case, but it's a good starting assumption. For example, if our trips data is updating every hour, but our bikeshare station table only updates once a quarter, we’ll probably want to run our trip analysis every hour, and not once a quarter.

### Schedules in Airflow

##### Start Date

Airflow will begin running pipelines on the start date selected. Whenever the start date of a DAG is in the past, and the time difference between the start date and now includes more than one schedule intervals, Airflow will automatically schedule and execute a DAG run to satisfy each one of those intervals. This feature is useful in almost all enterprise settings, where companies have established years of data that may need to be retroactively analyzed.

##### End Date

Airflow pipelines can also have end dates. You can use an end\_date with your pipeline to let Airflow know when to stop running the pipeline. End\_dates can also be useful when you want to perform an overhaul or redesign of an existing pipeline. Update the old pipeline with an end\_date and then have the new pipeline start on the end date of the old pipeline.

# Common Questions

**Wouldn't creating a new DAG for every feature change become cumbersome because feature changes or bugs happen all the time?**

**Is time the only type of partition? Can you partition other types, such as events or values?**

**Partitioning**

**Schedule partitioning**

Not only are schedules great for reducing the amount of data our pipelines have to process, but they also help us guarantee that we can meet timing guarantees that our data consumers may need.

**Logical partitioning**

Conceptually related data can be partitioned into discrete segments and processed separately. This process of separating data based on its conceptual relationship is called logical partitioning. With logical partitioning, unrelated things belong in separate steps. Consider your dependencies and separate processing around those boundaries.

Also worth mentioning, the data *location* is another form of logical partitioning. For example, if our data is stored in a key-value store like Amazon's S3 in a format such as: s3://<bucket>/<year>/<month>/<day> we could say that our date is logically partitioned by time.

**Size Partitioning**

Size partitioning separates data for processing based on desired or required storage limits. This essentially sets the amount of data included in a data pipeline run. Size partitioning is critical to understand when working with large datasets, especially with Airflow.

### Why Data Partitioning?

Pipelines designed to work with partitioned data fail more gracefully. Smaller datasets, smaller time periods, and related concepts are easier to debug than big datasets, large time periods, and unrelated concepts. Partitioning makes debugging and rerunning failed tasks much simpler. It also enables easier redos of work, reducing cost and time.

Another great thing about Airflow is that if your data is partitioned appropriately, your tasks will naturally have fewer dependencies on each other. Because of this, Airflow will be able to parallelize execution of your DAGs to produce your results even faster.

In this demonstration we upgrade our demonstration DAG to work on logically partitioned data. The data that we use in this lesson has been pre-partitioned in Amazon Web Services (AWS) S3 by creation date. The partition follows the format: <year>/<month>/<day>/<file>.csv.

In practice, it is often best to have Airflow process pre-partitioned data. If your upstream data sources cannot partition data, it is possible to write an Airflow DAG to partition the data. However, it is worth keeping in mind memory limitations on your Airflow workers. If the size of the data to be partitioned exceeds the amount of memory available on your worker, the DAG will not successfully execute.

**Examples of Data Quality Requirements**

* Data must be a certain size
* Data must be accurate to some margin of error
* Data must arrive within a given timeframe from the start of execution
* Pipelines must run on a particular schedule
* Data must not contain any sensitive information