> DATA ARE BECOMING THE NEW RAW MATERIAL OF BUSINESS

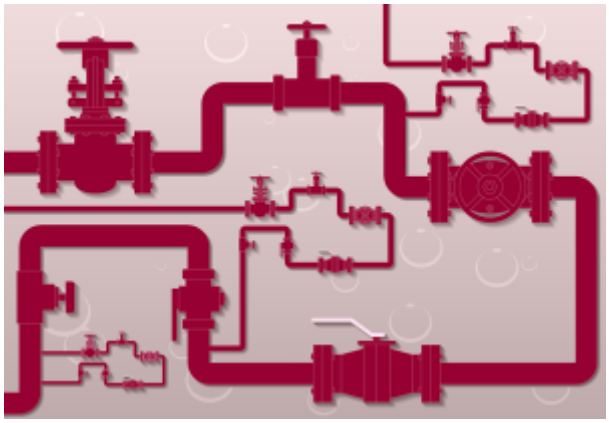
— THE ECONOMIST

Three best practices for building successful data pipelines

Posted by Michael (http://www.thedataincubator.com/team.html#michael) on September 17, 2015

On September 15th, O'Reilly Radar featured an article written by Data Incubator founder Michael Li. The article can be found where it was originally posted here (http://radar.oreilly.com/2015/09/three-best-practices-for-building-successful-data-pipelines.html).

Building a good data pipeline can be technically tricky. As a data scientist who has worked at Foursquare and Google, I can honestly say that one of our biggest headaches was locking down our Extract, Transform, and Load (ETL) process



(https://en.wikipedia.org/wiki/Extract,_transform,_load).

At The Data Incubator (http://www.thedataincubator.com/?utm_source=oreilly&utm_medium=link&utm_campaign=datapipeline), our team has trained more than 100 talented Ph.D. data science fellows who are now data scientists at a wide range of companies, including Capital One (https://blog.thedataincubator.com/2015/08/columbia-astrophysics-postdoc-moves-to-capital-one-labs-alumni-spotlight-on-brian-farris/), the New York Times (https://blog.thedataincubator.com/2015/02/alumni-spotlight-dorian-goldman-using-a-pure-math-background-to-solve-problems-for-the-new-york-times/), AIG, and Palantir (https://blog.thedataincubator.com/2015/02/moving-to-palantir-from-mathematics-alumni-spotlight-on-justin-bush/). We commonly hear from Data Incubator alumni and hiring managers that one of their biggest challenges is also implementing their own ETL pipelines.

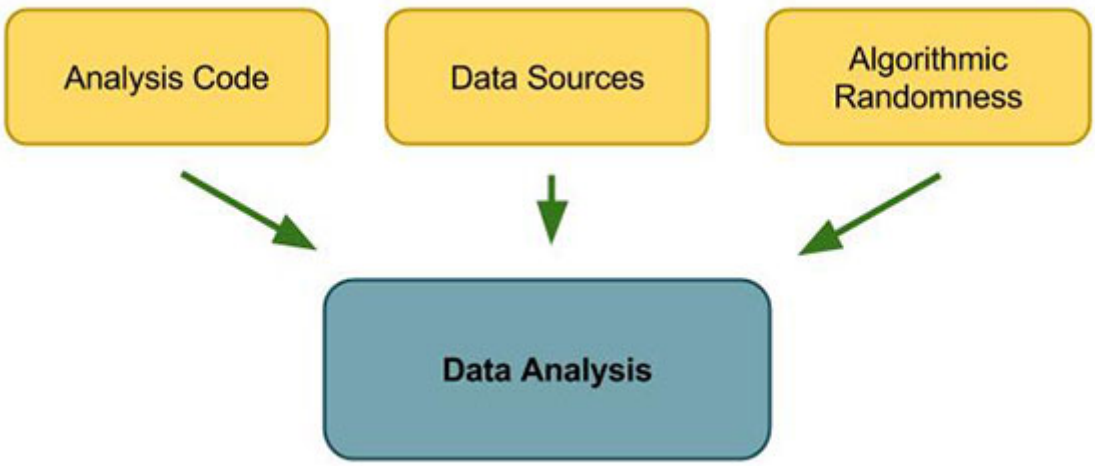
Drawn from their experiences and my own, I've identified three key areas that are often overlooked in data pipelines, and those are making your analysis:

1. Reproducible
2. Consistent
3. Productionizable

While these areas alone cannot guarantee good data science, getting these three technical aspects of your data pipeline right helps ensure that your data and research results are both reliable and useful to an organization.

Ensuring reproducibility by providing a reliable audit trail

To ensure the reproducibility of your data analysis, there are three dependencies that need to be locked down: analysis code, data sources, and algorithmic randomness.



Science that cannot be reproduced by an external third party is just not science — and this does apply to data *science*. One of the benefits of working in data science is the ability to apply the existing tools from software engineering. These tools let you isolate all the dependencies of your analyses and make them reproducible.

Dependencies fall into three categories:

1. **Analysis code** — All analysis code should be checked into source control. Every analysis consists of innumerable assumptions made during both ETL and modelling. These are impossible to document exhaustively, and the rapid iteration cycles enabled by faster tools means the results and documented processes are often out of sync. Like with soft engineering, what was run is what was in code, and having that time stamped and recorded in a version control system makes analysis much more reproducible.
2. **Data sources** — While the analysis code needs to be locked down, so does the data source. (You've likely encountered analyses that open up a mysterious "sales.dat" file that no one can find.) Even if you *can* locate your data files, you might find a version that's encoded differently than the one originally used (e.g. using JSON rather than XML), thus making your otherwise carefully source-controlled analysis code much less useful. Locking down the exact file is the first step in data consistency.
3. **Algorithmic randomness** — Is your data being randomly sub-sampled? Even seemingly deterministic analyses can have hidden randomness. For example: check whether your gradient descent starts at a random initial condition. These techniques all depend on a random number generator. The problem with algorithmic randomness is that even if we freeze the input data and analysis code, the outputs will still vary randomly. This eliminates reproducibility because another person trying to reproduce the result can never answer the question "is the new answer different because I did something wrong or because of algorithmic randomness?" To promote reproducibility, set the random number generator's "seed" (https://en.wikipedia.org/wiki/Random_seed) to an arbitrary, but fixed, value (that's checked into source control) before running the analysis so that results are consistent and discrepancies can be attributed to code or data.

By locking down analysis code, data sources, and algorithmic randomness, we can ensure that the entire analysis can be re-run by anyone. The benefits, however, go beyond scientific reproducibility; making the analysis fully documented provides a reliable audit trail, which is critical for data-driven decision-making.

Establishing consistency in data

How we establish the consistency of data sources is a bit trickier than establishing reproducibility. Establishing consistency is of fundamental importance because data science obeys the maxim "garbage in, garbage out," and, as we have discussed, access to the correctly formatted data is

just as important as access to the right analysis code.

There are generally two ways of establishing the consistency of data sources. The first is by checking-in all code *and data* into a single revision control repository. The second method is to reserve source control for code and build a pipeline that explicitly depends on external data being in a stable, consistent format and location.

Checking data into version control is generally considered verboten for production software engineers, but it has a place in data analysis. For one thing, it makes your analysis very portable by isolating all dependencies into source control. Here are some conditions under which it makes sense to have both code and data in source control:

Small data sets — A lot of data analysis either fully or partially depends on a few small data sets. Even if you are performing an analysis on a large amount of data, sub-sampling to a smaller data set can be sufficient. In this case, it may make sense to keep your data checked into source control rather than building an expensive pipeline to manage it. Obviously, if your data set is large, it becomes impractical to check it into source control.

Regular analytics — A lot of data analytics are simply one-off reporting, and it may not make much sense to set up a pipeline simply for the sake of reproducibility. On the other hand, if this is an analysis that needs to occur daily, it doesn't make sense to check-in each day's data to an ever-growing version control repository.

Fixed source — If you control the data source or know that it will likely remain fixed, this might be a good candidate for setting up a full pipeline. On the other hand, if your data source is managed by an external party and subject to frequent changes, it may not be worth setting up and maintaining a consistent pipeline. For example, if your data includes files that need to be downloaded from a government website, it may make sense to check those into source control, rather than maintaining a custom script that downloads the files each time.

While source-control systems are considered forbidden on the production end, source-control systems like git can easily handle tens of megabytes of data without significant performance lag. Services like GitHub's large file storage system (https://github.com/blog/1986-announcing-git-large-file-storage-lfs) promise to make handling large data in source control even easier in the future. Whether you choose to check in all code and data into a single revision control repository or just put your code under source control and lock down your data sources in an external pipeline, securing your data sources is the key to consistent data and reproducible analysis.

Productionizability: Developing a common ETL

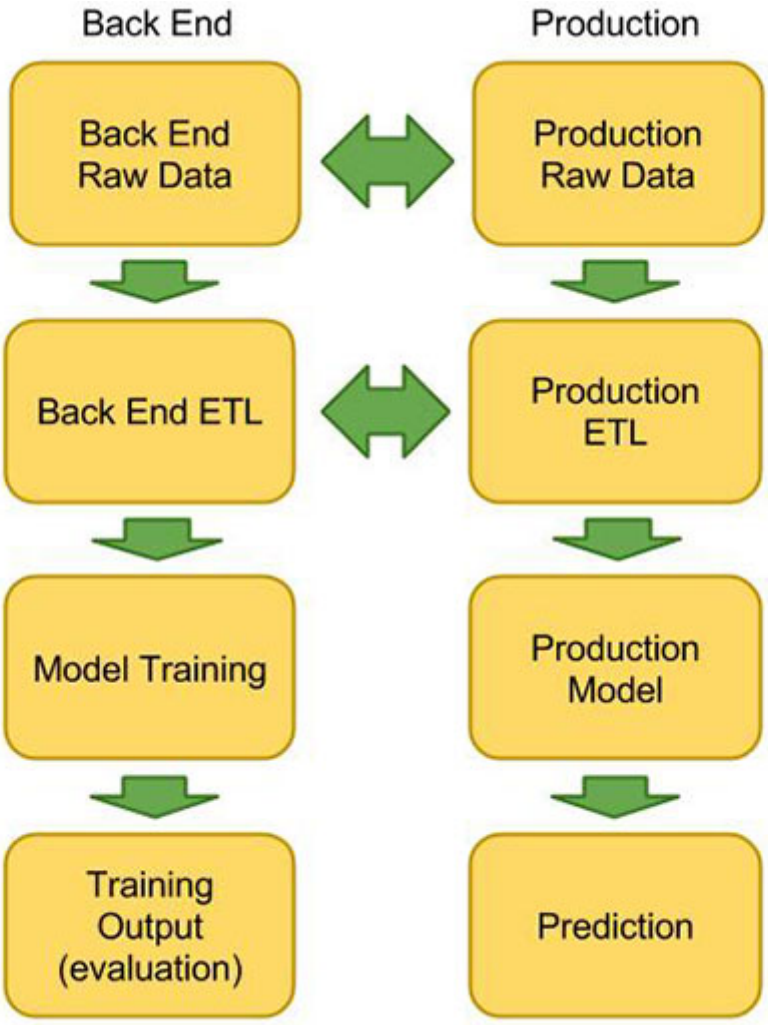
Of course data science that isn't deployed is useless, and the ability to productionize results is always a central concern of good data scientists. From a data pipelining perspective, one of the key concerns is the development of a common ETL process shared by production and research.

As a simple example, we may want to join-in user data and purchase data to feed into a recommender model for a website. This join would need to happen in two environments: in research, where we need the data for training, and in production, where we need the data for

predictions and recommendations. Ideally the ETL code that joins these two chunks of data (and the myriad of data normalization decisions embedded in this code) would be shared between the two environments. In practice, this faces two major challenges:

1. **Common data format** — In practice, there are a number of constraints to establishing a common data format. You'll have to pick a data format that plays well with production and Hadoop (https://hadoop.apache.org/) (or whatever back end datastore you use). Being able to use the same data formats reduces the number of unnecessary data transformations, and helps prevent introducing bugs that contaminate data purity. Using the same data formats also reduces the number of data schemas and formats that your data team has to learn and maintain.
2. **Isolating library dependencies** — You will want to isolate library dependencies used by your ETL in production. On most research environments, library dependencies are either packaged with the ETL code (e.g. Hadoop) or provisioned on each cluster node (e.g. mrjob (https://github.com/Yelp/mrjob)). Reducing these dependencies reduces the overhead of running an ETL pipeline. However, production code typically supports a much larger set of of functionality (HTML templating, Web frameworks, task queues) that are not used in research, and vice versa. Data engineers and scientists need to be careful about isolating the ETL production code from the rest of the codebase to keep the dependencies isolated so that the code can be run efficiently on both the front and back ends.

While sharing ETL code between production and research introduces some complexities, it also greatly reduces the potential for errors, by helping guarantee that the transformed data used for model training is the same as the transformed data the models will use in production. Even after locking down your data sources and ensuring data consistency, having separate ETL code can lead to differences in modelling input data that renders the outputs of models completely useless. Common ETL code can go a long way to ensuring reproducible results in your analysis.



Back end and production sharing data formats and ETL code make research rapidly productionizable.

.

Focusing on the science

With the high cost of data science, managers need to ensure that their data analytics are both sound and useful. Ultimately, achieving this depends on external factors, like the quality of the team and the quality of the underlying data, which may be outside of the manager's control. Data analysis is hard enough without having to worry about the correctness of your underlying data or its future ability to be productionizable. By employing these engineering best practices of making your data analysis reproducible, consistent, and productionizable, data scientists can focus on science, instead of worrying about data management.

Editor's Note: The Data Incubator is a data science education company. We offer a free eight-week Fellowship (https://www.thedataincubator.com/fellowship.html) *helping candidates with PhDs and masters degrees enter data science careers. Companies can hire talented data*