Writing clean, responsible code

Why Clean Code?

- Switch mindset
  - coding by yourself -> coding collaboratively.
  - code living on your laptop -> code hosted live and automated

- Reproducibility
  - Someone trying to help you debug an error on Stackoverflow
  - You working on a project with multiple people
  - You receiving legacy code from someone else, or passing on your work to your successor

- Scalability
  - Just because it works on 1 csv file doesn't mean it won't run forever for the full dataset
  - Writing high performance code means less computing power, less memory issues, and less $$!

Technical Debt

- Technical Debt is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer. (Wikipedia)

- Harsh deadlines can drive you to write sloppy, stream of consciousness code, but the hidden cost (technical debt) incurred of that can result in larger failures in the future when your code is automated.

Tenets of Clean Code

- Code is simple, clear, and concise.
  - Naming conventions should be explicit
  - Functions do one thing and one thing only. (e.g. “get_data”, “train_model”)
  - Do not repeat yourself! (If you find that your are repeating yourself, look to a loop)
  - Try to catch errors (e.g.`try`) instead of throwing default error codes

- Format correctly. Follow the style guides:
  - Pythonic code: https://docs.python-guide.org/writing/style/
  - PEP8 guide for Python: https://www.python.org/dev/peps/pep-0008/

- Code is clearly documented and commented.

Tenets of Clean Code

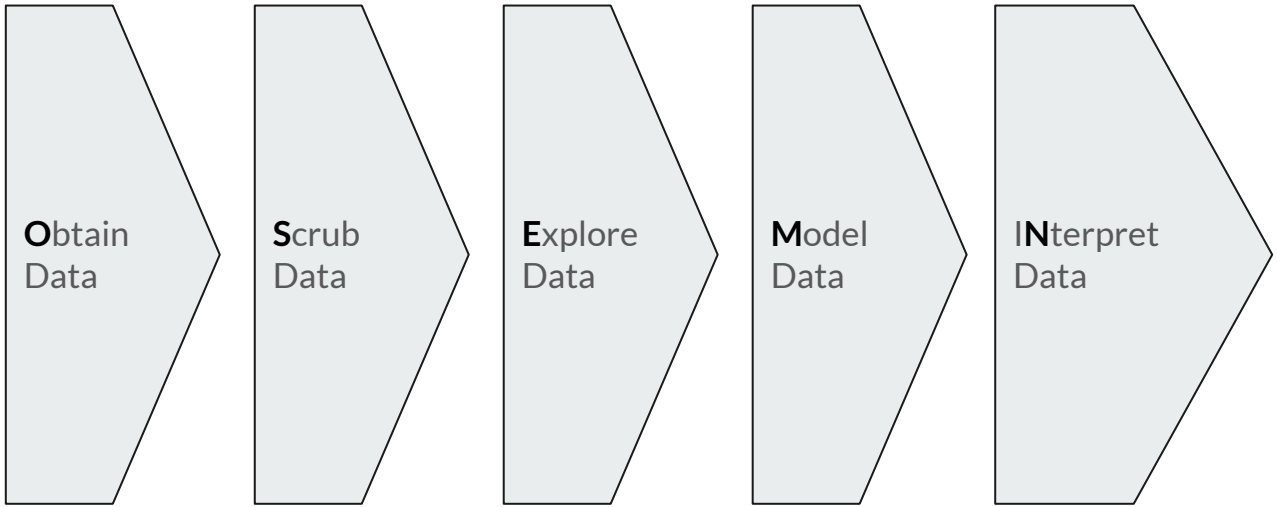
- The code is easily testable.
  - A testing unit (unit test) should focus on one tiny bit of functionality and prove it correct.
  - All tests are easy to understand and easy to change. Avoid “glamor tests”.
  - More here: https://docs.python-guide.org/writing/tests/

- Code review.
  - Have a friend, classmate, coworker review your code. Doesn’t matter if that person has more, less, or equal experience than you.
  - Working alone? Try the rubber duck debugging method (https://en.wikipedia.org/wiki/Rubber_duck_debugging)
  - Learn version control! Git all the things!!

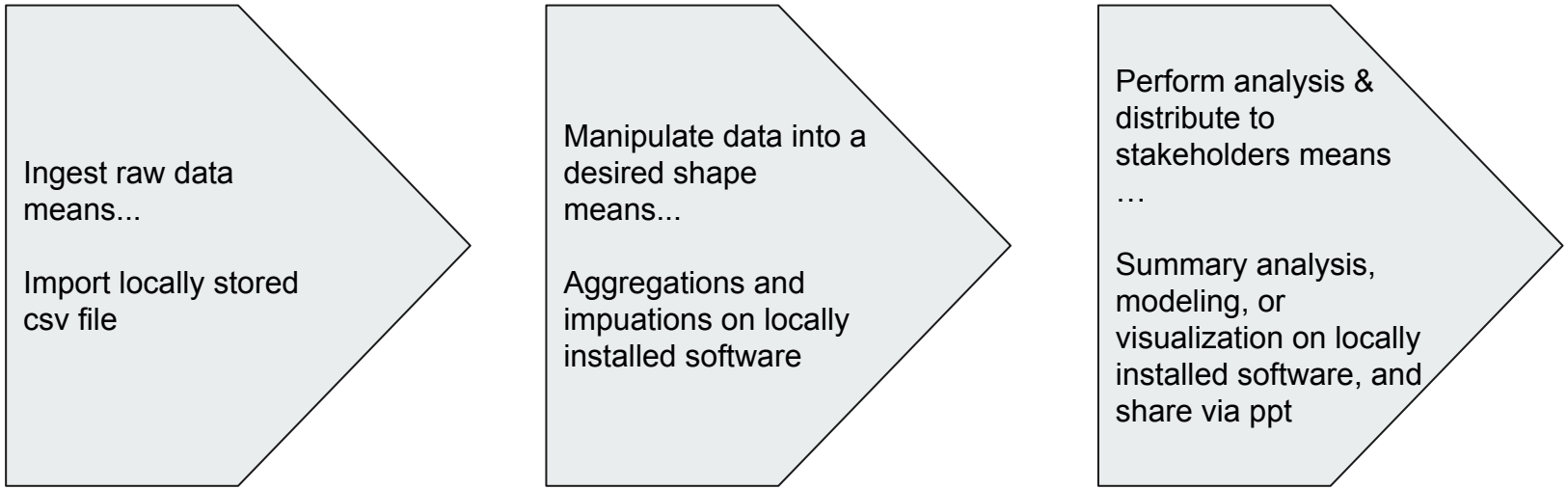
- Read the book Clean Code for more details and check out this guide specific for Python data work.

Complexities in your data workflow

The OSEMN framework for data workflow

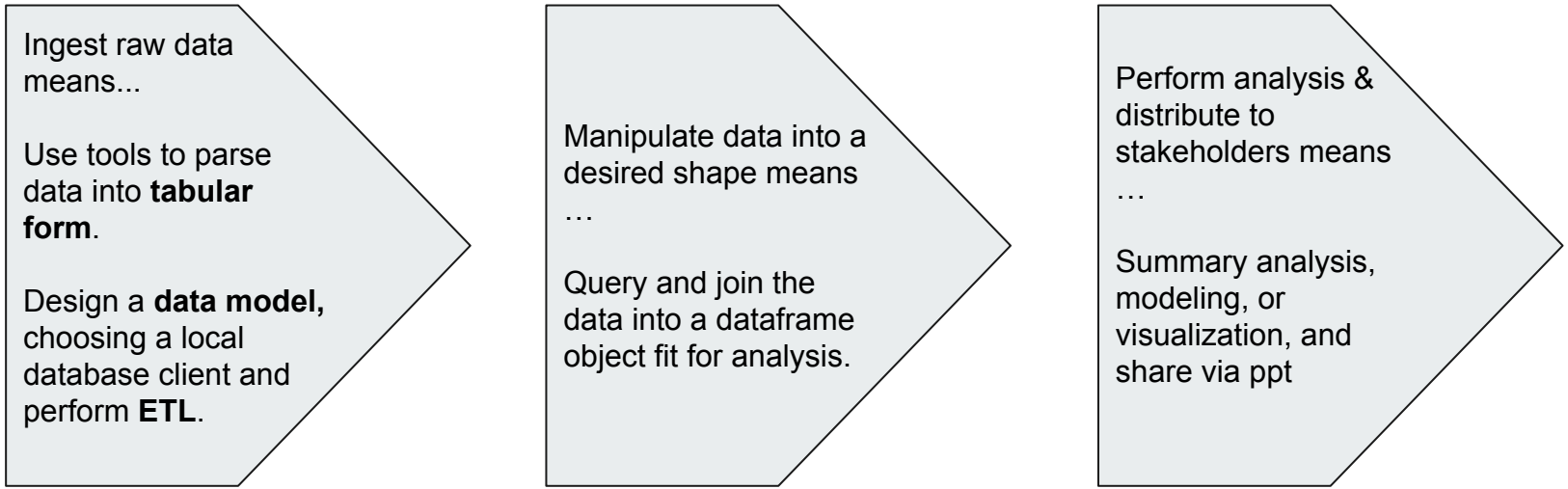


A simple example

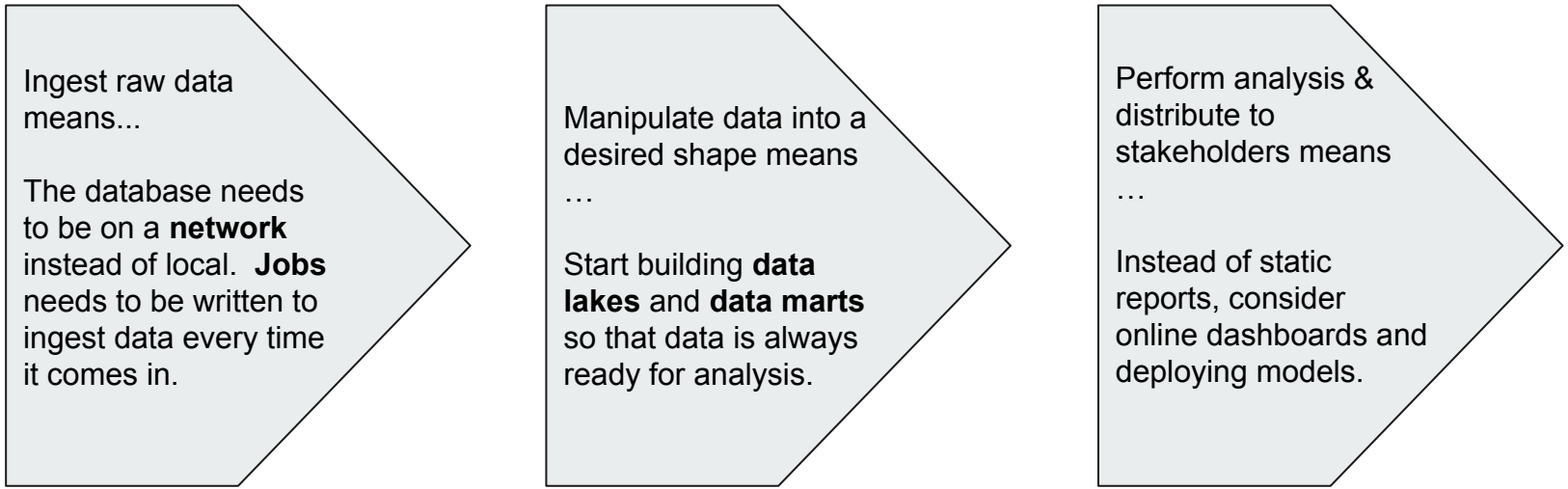


But things can get complicated very quickly...

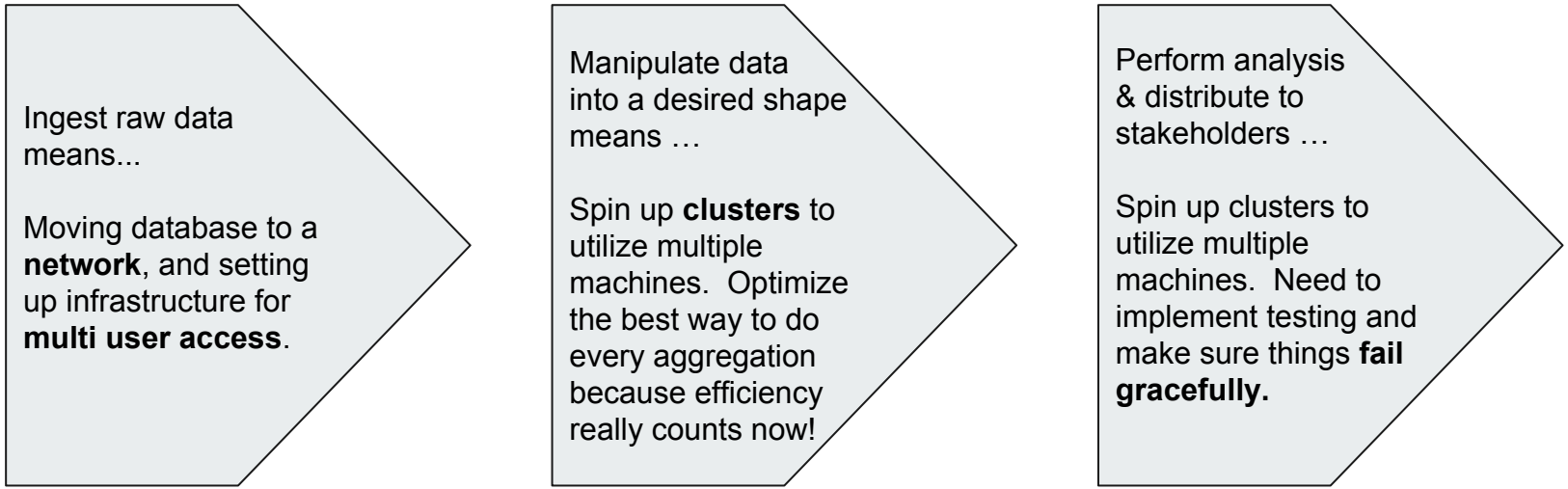
What if the data is more than a simple text file?



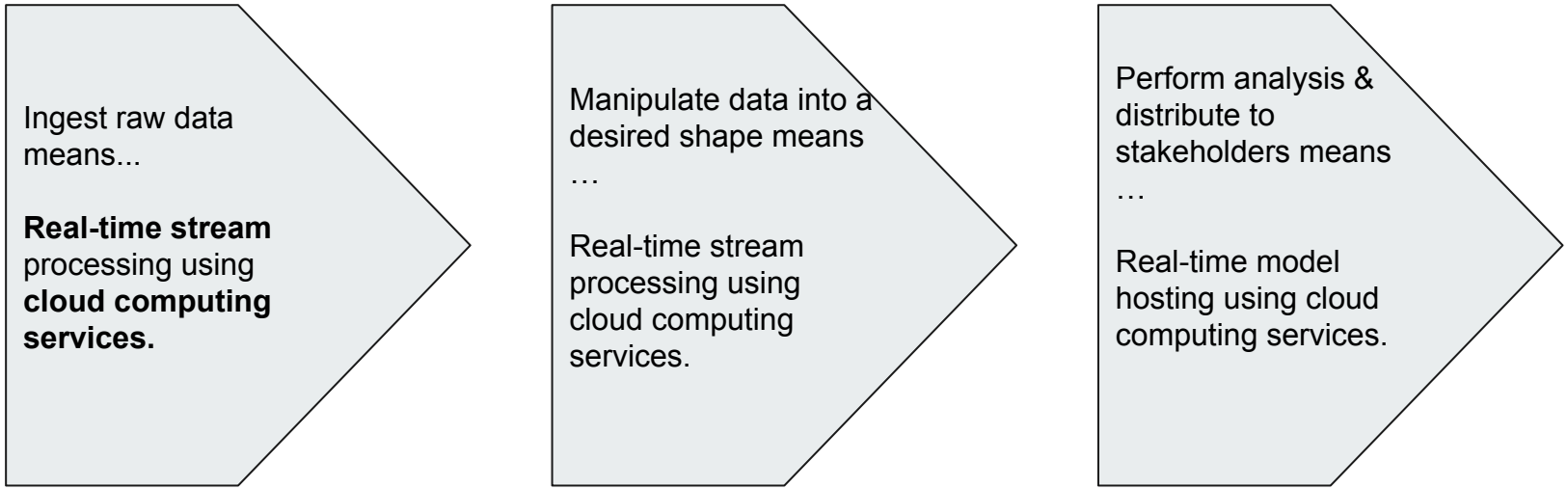
What if new data is periodically available?



What if every step is too large to be handled on a single machine?



What if stakeholder wants real-time data access and you want to save money?



Major players in cloud computing …

Amazon Web Services	Google Cloud Platform	Microsoft Azure

More on the design of data architecture? Check out the following two presentations:

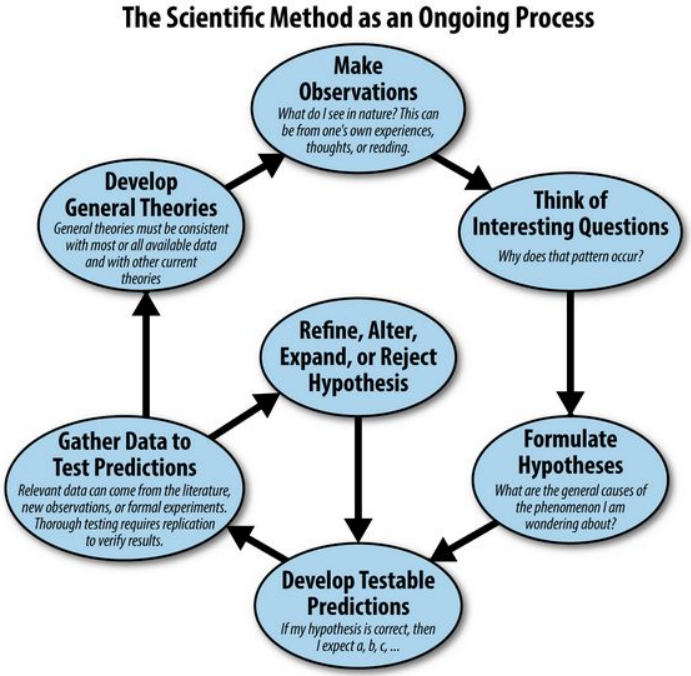
here and here

Complexities in working on a data team

A typical workflow for the solo data scientist ...

.. can be iterative because data scientists tend to come from an academic background and are used to the scientific method.

This works if it's solo work, but in a team, this workflow will change.



Source: Agile Data Science 2.0

A typical project team …

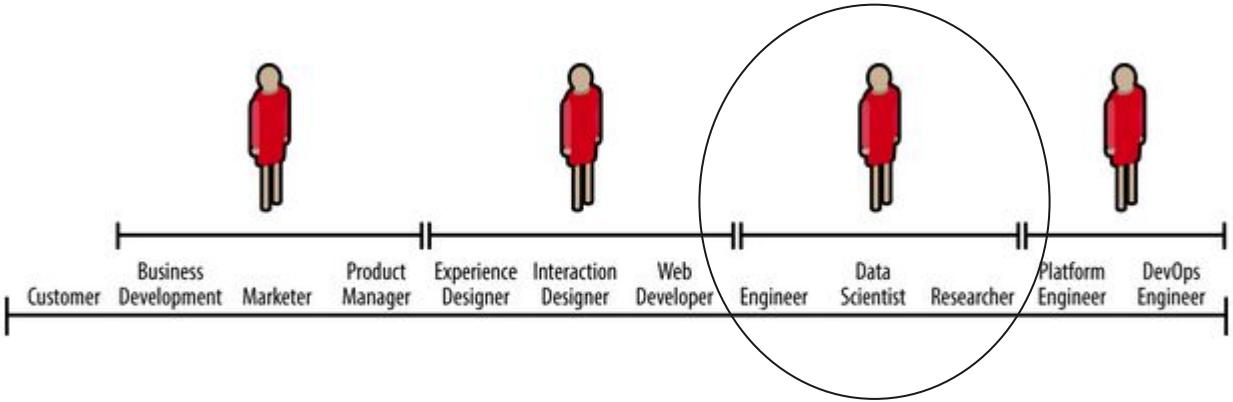
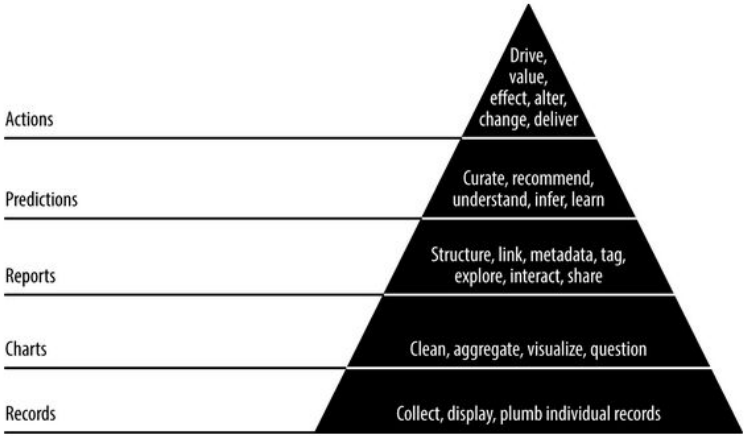


Figure 1-9. Broad roles in an Agile Data Science team

Source: Agile Data Science 2.0

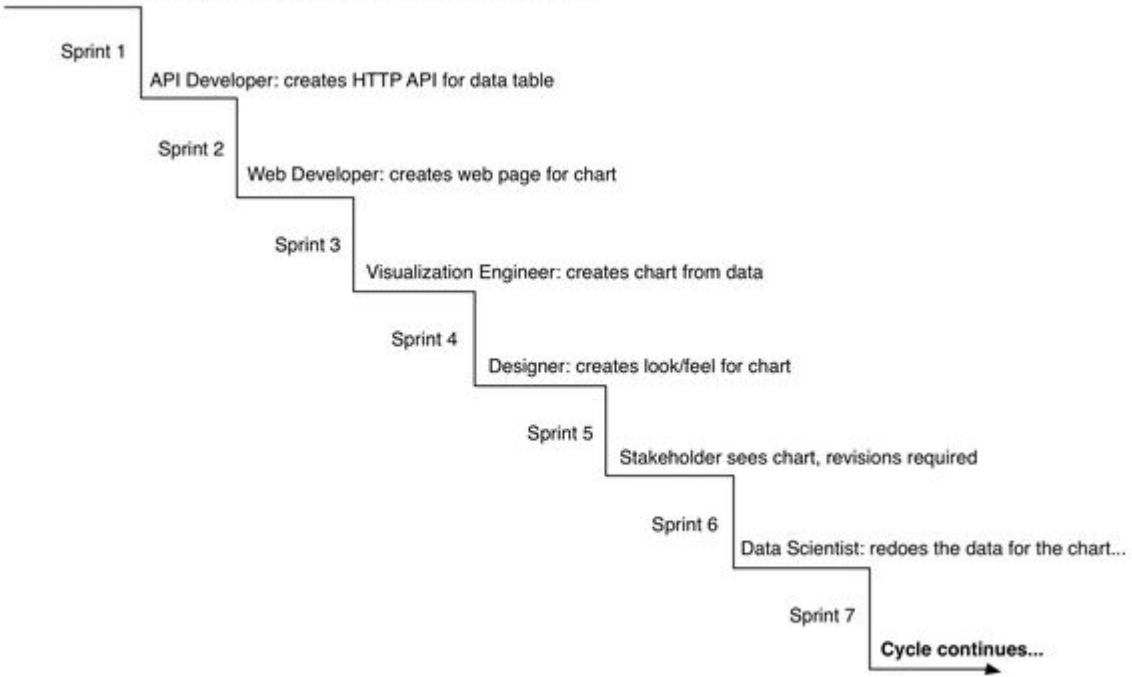
The Agile Data Science Manifesto

- Iterate, iterate, iterate: tables, charts, reports, predictions
- Ship intermediate output. Even failed experiments have output.
- Get meta. Describe the process, not just the end state.
- Climb up and down the data-value pyramid as we work.



Source: Agile Data Science 2.0

When you are a data scientist on a team …

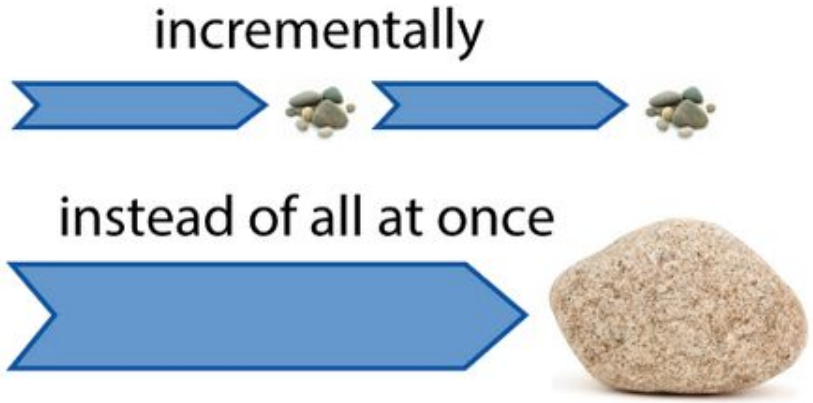


Source: Agile Data Science 2.0

What is Agile?

Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end.

It works by breaking projects down into little bits of user functionality called user stories, prioritizing them, and then continuously delivering them in short 1-2 week cycles called iterations.



Source: http://www.agilenutshell.com/what_is_agile

User Story

User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

As a < type of user >, I want < some goal > so that < some reason >.

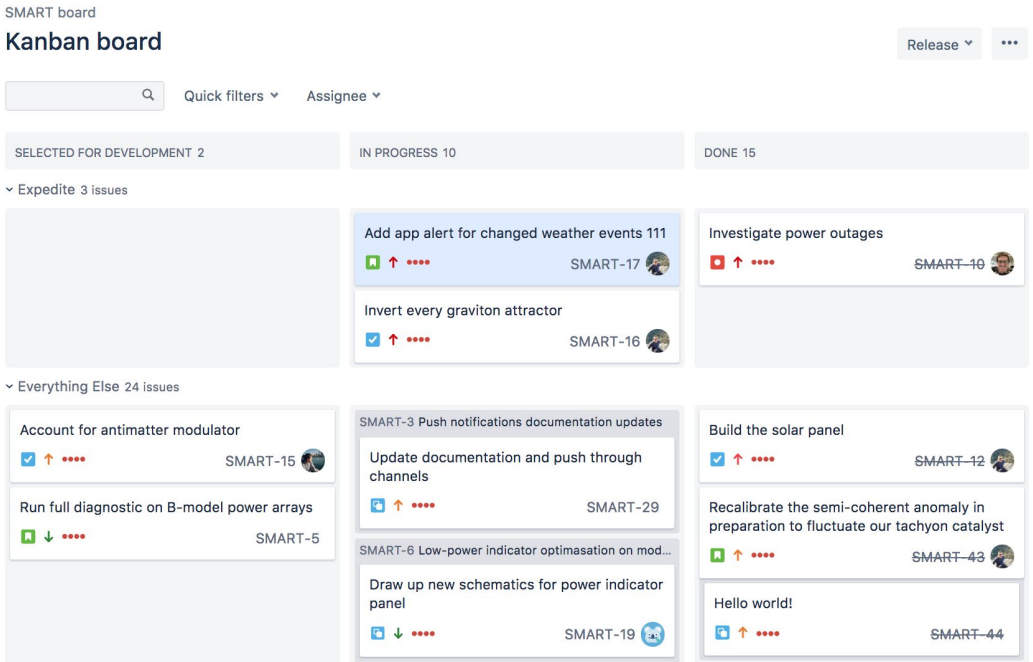
As a **data analyst**, I want **data in a certain format** so that **I can train a model on it.**

These user stories are then converted into bite-sized “tasks” with clear direction on how to accomplish this goal and then assigned to someone on the team.

Source: https://www.mountaingoatsoftware.com/agile/user-stories

Task Board

The **task board** is a visual display of the progress of the team during a **sprint**. It presents a snapshot of the current sprint backlog allowing everyone to see which tasks remain to be started, which are in progress and which are done.



Source: https://confluence.atlassian.com/get-started-with-jira-software/common-jira-software-boards-937185142.html

Any more questions? Unmute yourself and ask!