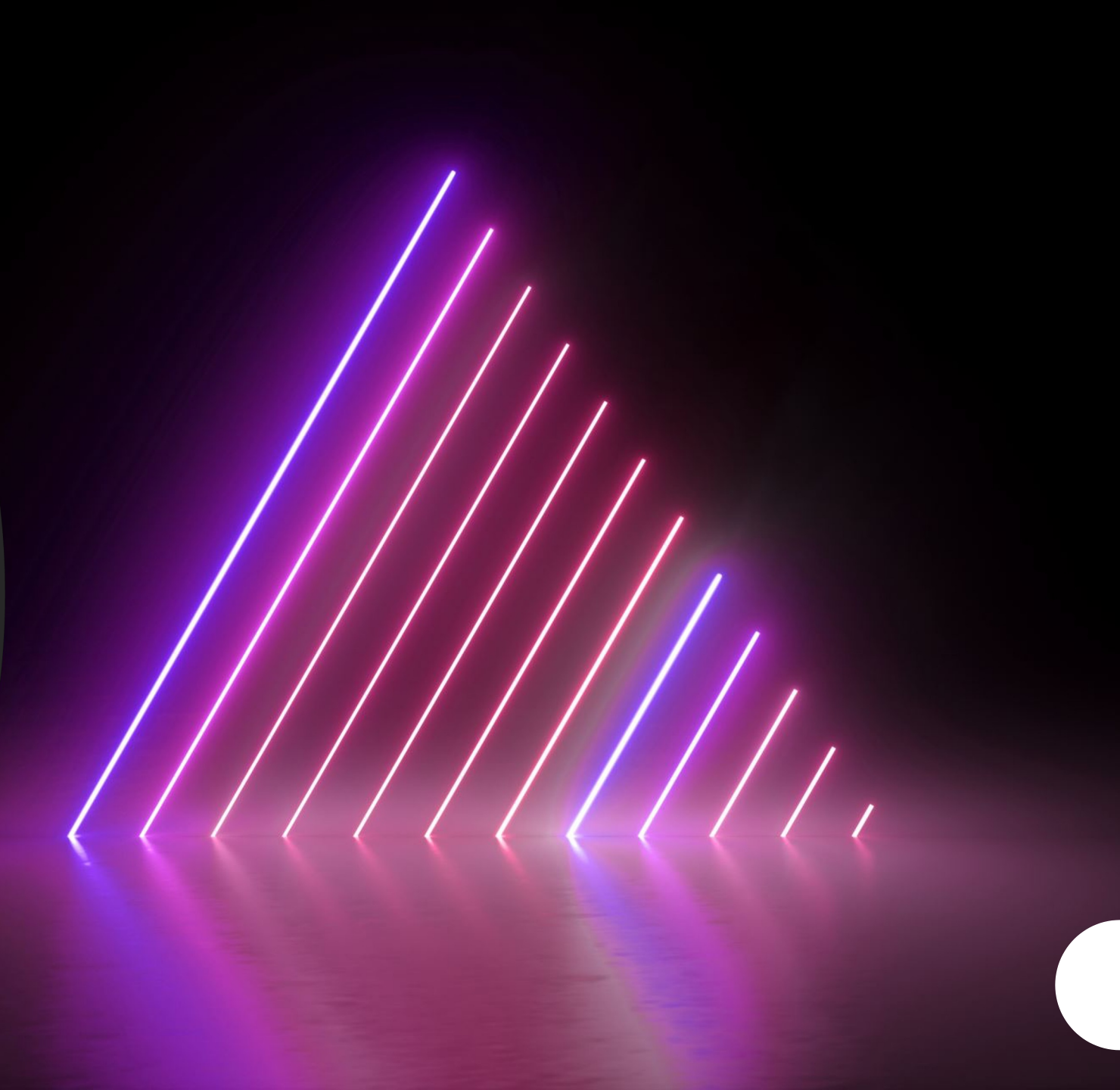
Going Pro

Modern development tools to help you build better software faster

Made with the help of the Microsoft AI slide designer



Agenda

- Andy's philosophy of software engineering
- When is a single script good enough?
- Sample exercise – using an IDE to fix legacy code

Who am I?

- Husband / therapy animal to the managing director
- Founder and CEO of Handshake Health
- Former data scientist / data science executive with:

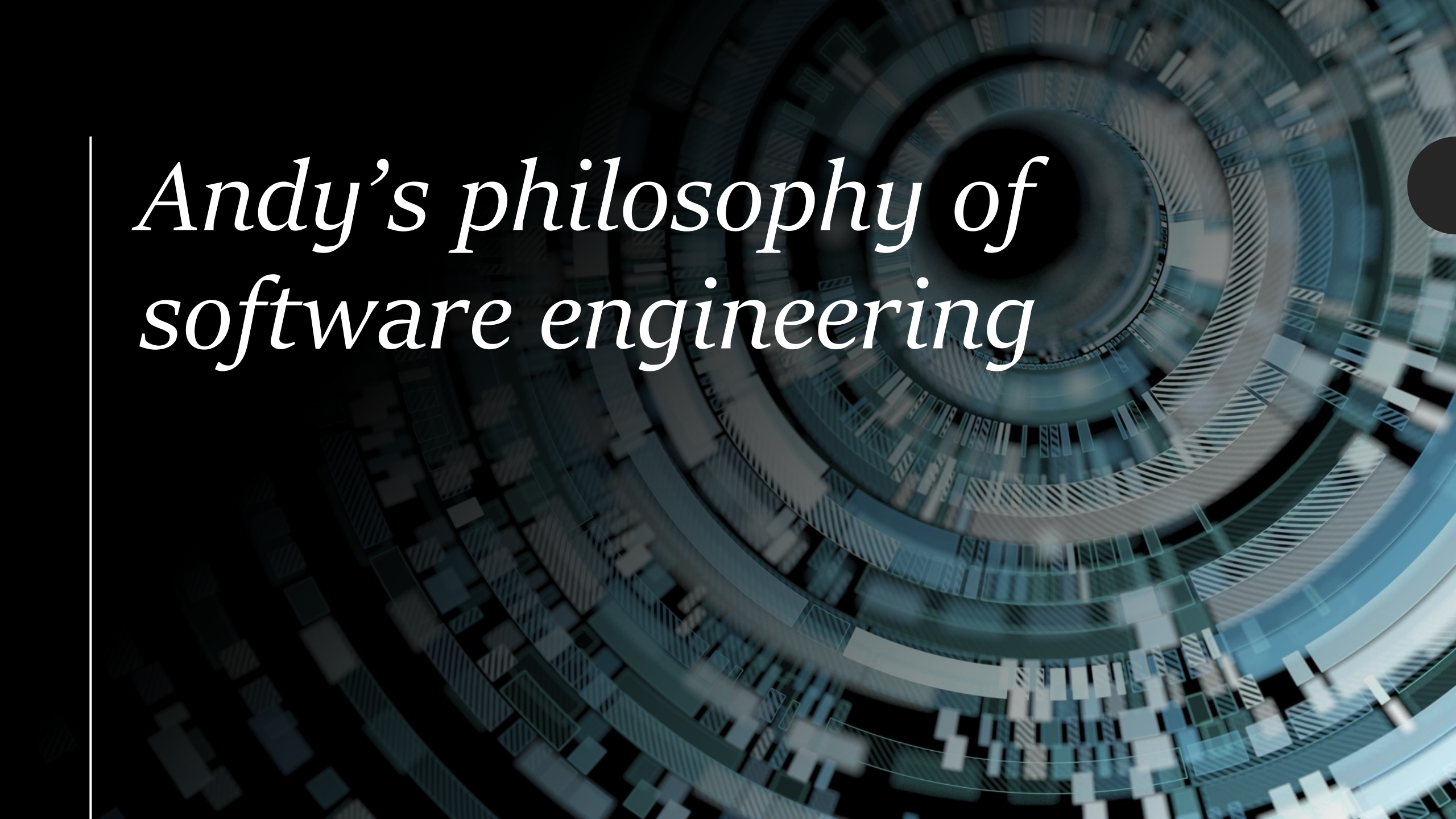
  Simon Kucher & Partners

  Moss (Berlin Fintech)

  Boston Consulting Group

  Nike
- PhD and postdoc in biomedical engineering
- Undergrad in music and chemistry
- Generally fun person

Andy's philosophy of software engineering



Time is valuable

Good software engineering saves time. How?

- Helping a new developer onboard faster
- Isolate problems in the code
- Breaking big problems into smaller ones
- Making collaboration easier

Things change

Changes in your code are inevitable:

- You'll want to run your code in different conditions
- The purpose / scope of your project will change
- Packages you use will change
- OS updates are your worst enemy
- Other people will change your code and break it

Changes necessary but time consuming! We want to make changes as easy as possible.

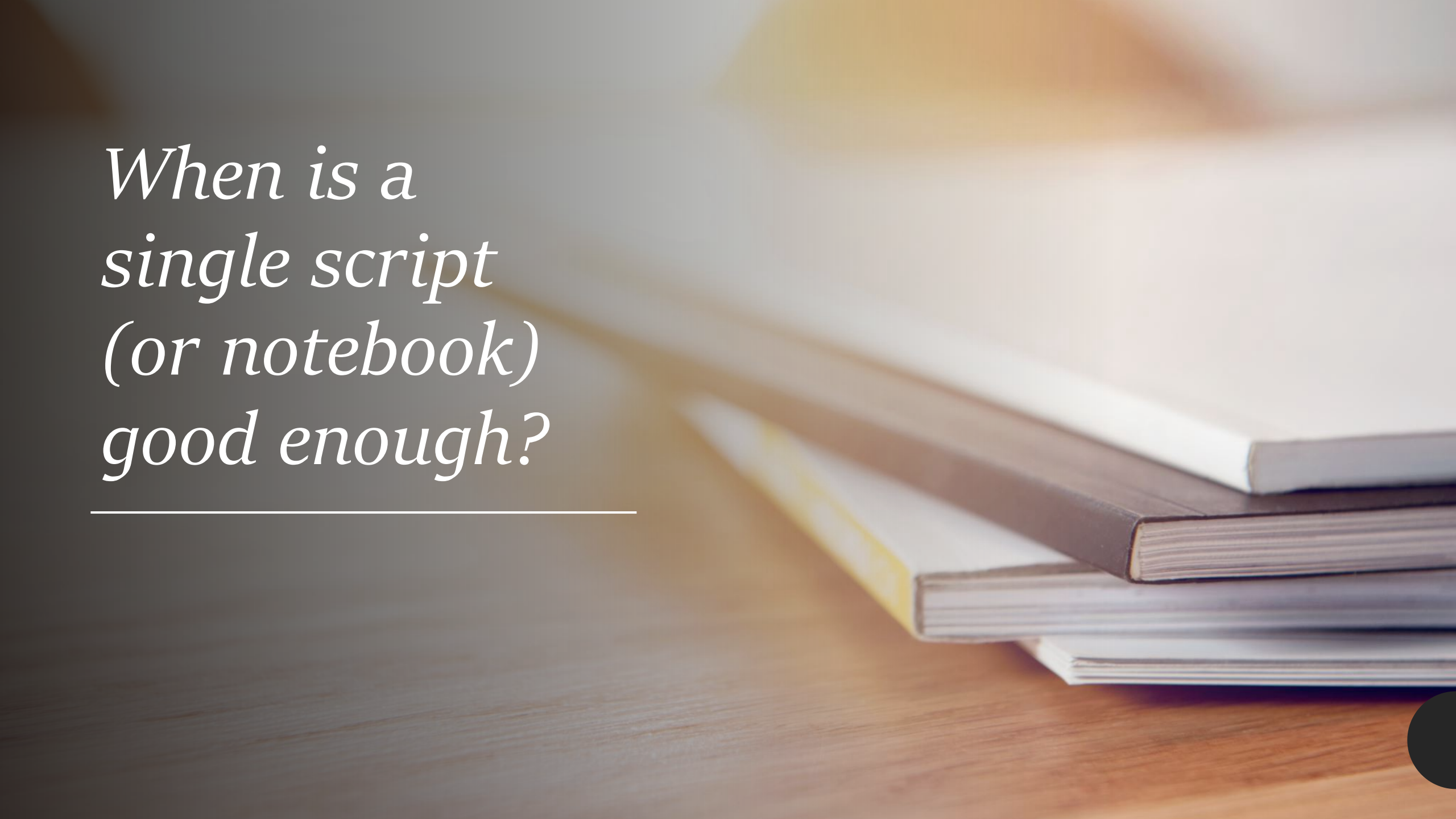
Takeaway: Rules are only good if they help

Rules have two good uses:

- They save you time (sometimes in the long run)
- They make change easier to navigate

Take away:

- Just because some ~~asshole~~ unpleasant person on the internet can only imagine one way to do things, it doesn't mean that they're right.


When is a single script (or notebook) good enough?

Single developer

What this means:

- Only one person will work on this code

When you've moved past this:

- You have collaborators on the code
- Any result from your code will be published

How a single script / notebook limits you:

- It's hard to isolate errors in a single huge script
- Merge conflicts in notebooks are HORRIFIC

Plan in development

What this means:

- You’re not sure what the end product is or there are unanswered technical questions

When you’ve moved past this:

- You can draw your end-to-end process
- you know all the technologies you’re going to use

How a single script / notebook limit you:

- Notebooks can’t be broken up into smaller tasks

It only gotta work once!

What this means:

- You have a task that needs code (like plotting a figure or copying files) but isn't a software product

When you've moved past it:

- When it gotta work twice!

How a single script / notebook limits you:

- Versioning and configuration get really messy
- It's hard to know what you did when you want to re-create a previous analysis

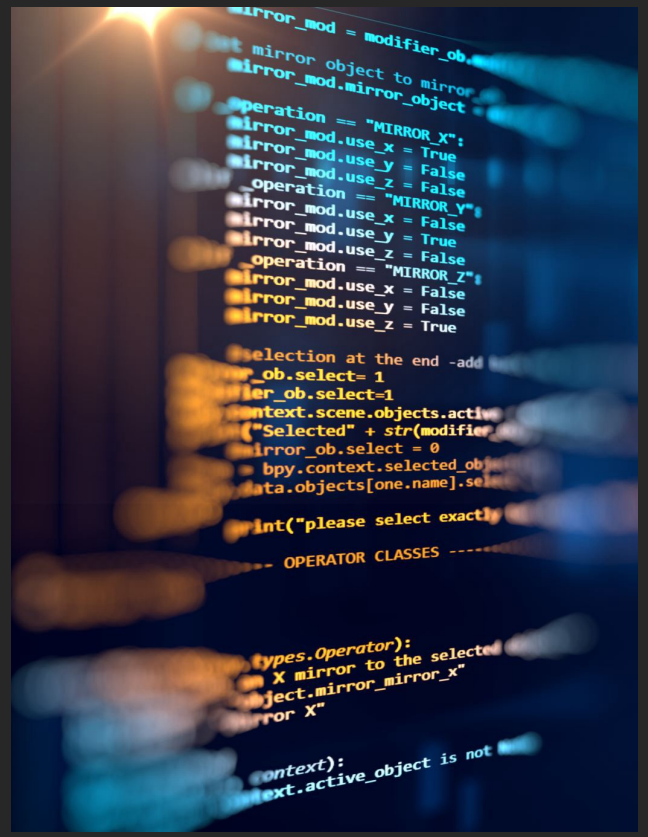
Take away:

You need to move past a notebook when:

- There are results from your code that will be published
- Your code will be used by someone else
- You want to reuse or run your code in a slightly different context than the one you developed it in (new data, new model, etc.)

Sample exercise

Using an IDE to fix legacy code



PyCharm Downloads

Tools

- IDE (integrated development environment) – we're using pycharm community edition
- Environment management – Anaconda
- Testing – we're using pytest
- Linting – we're using black
- AI (optional) – chatgpt free tier is great

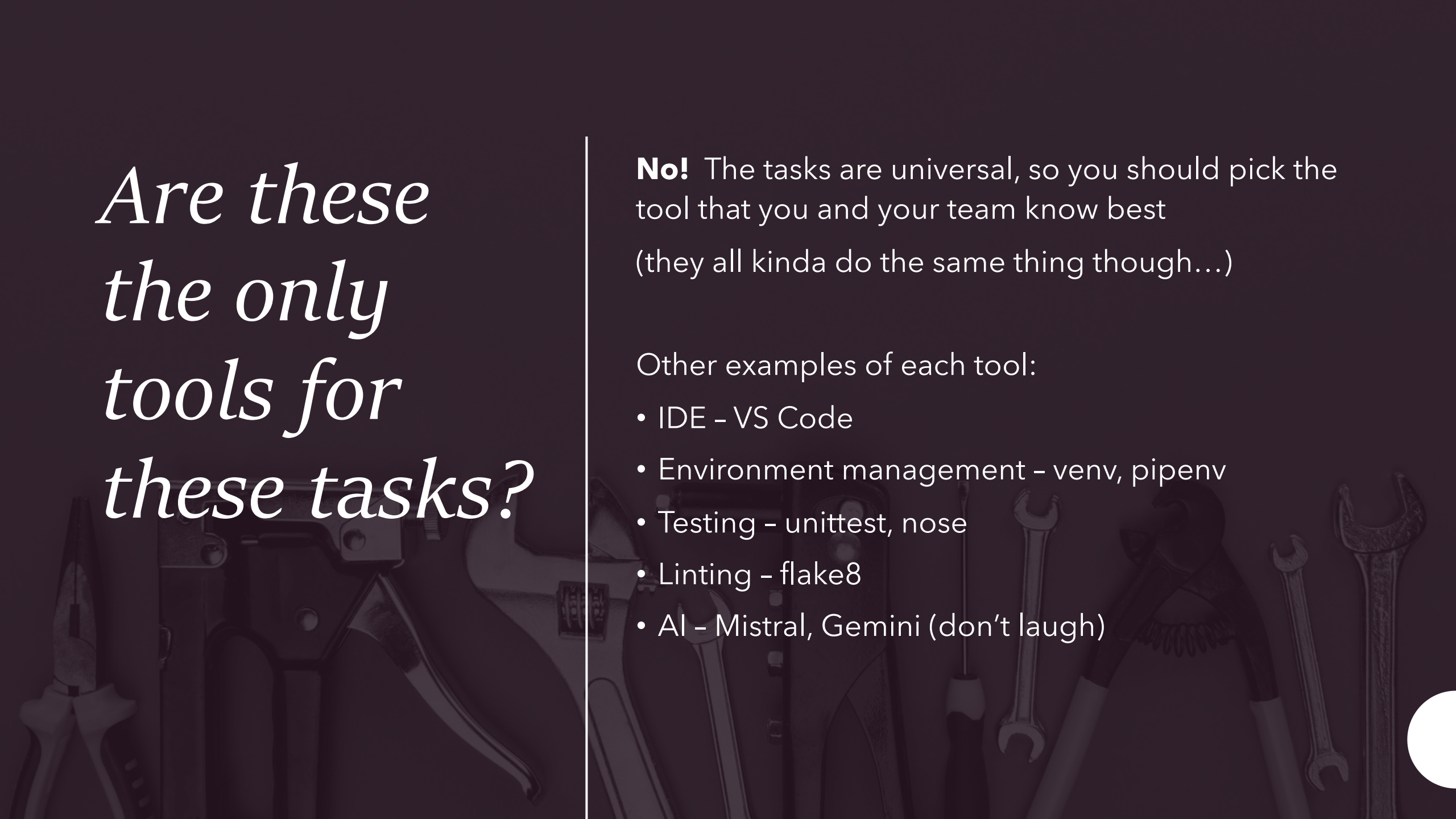
Are these the only tools for these tasks?

No! The tasks are universal, so you should pick the tool that you and your team know best

(they all kinda do the same thing though…)

Other examples of each tool:

- IDE – VS Code
- Environment management – venv, pipenv
- Testing – unittest, nose
- Linting – flake8
- AI – Mistral, Gemini (don’t laugh)



What are we doing?

We're going to refactor a sample data processing notebook into a more robust pipeline. How?

1. Set up our environment
2. Clean up the code
3. Setup command line runs
4. Write an end-to-end test

Environment setup

Key tasks

- Creating a conda environment
- Managing packages
- Using a requirements file
- Troubleshooting from the command line

Further reading:

- Configure which conda channels you use
- (Pro): Create a remote environment on an astro-node

Clean up the code

Key Tasks:

- Remove unused code
- Remove unused imports
- Reformat (lint) the code

Further reading:

- Configure Black using command line arguments
- Look at other formatting tools for other standards

Setup command line runs

Key Tasks:

- Clear out hard coded values
- Set input variables in the run configuration
- Take a look at our outputs

Further reading:

- Put complex inputs into a config file
- Configure command line inputs

Write an end-to-end test

Background – what is an end-to-end test?

- A full run of your code which goes from input to output
- An end-to-end test should run as quickly as possible while covering as much of the codebase as possible (this is a judgement call)

Why must we test these crappy scripts?

- Tests ensure code correctness when the code is changed.
- You shouldn't use testing for scientific questions (e.g. did my model fit converge? Is my data quality good enough?).
- Note – tests often feel trivial, it should feel like, "did I turn out the lights before I left the house?"

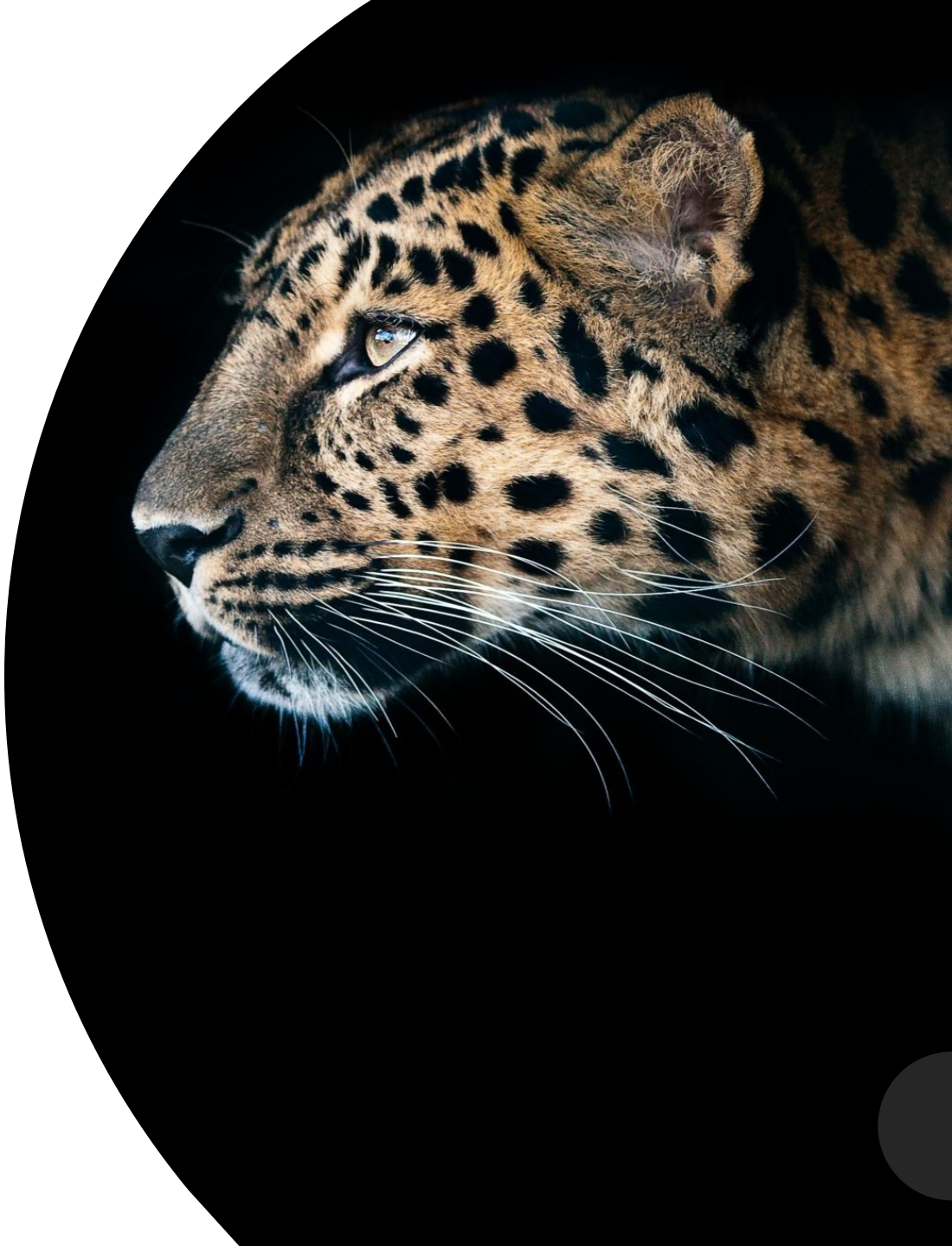
Key tasks:

- Write the test (optional – with AI help)
- Run the test
- Interpret output

Further reading:

- Test coverage
- Mocking

Where do you go from here?



Other stuff that's useful

- Continuous Integration / Continuous Development – integrate testing into your version control
- Code coverage – how much of your code is actually tested
- Code profiling – runs your test script to see where there are inefficiencies in your code
- Remote deployment – how to run your code on another machine (pycharm does this in the pro version)

Thank you!

Questions?