Going from notebooks to production code

Catherine Nelson and Robert Masson

https://github.com/catherinenelson1/notebooks-to-production

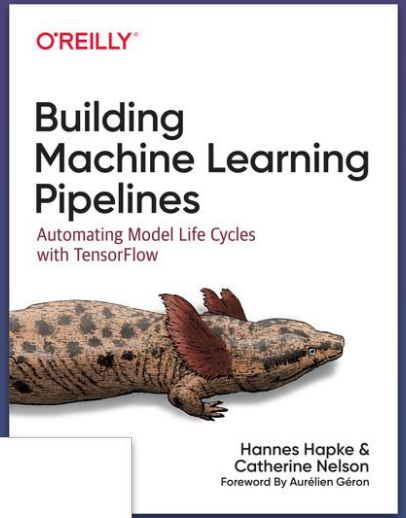
About Catherine

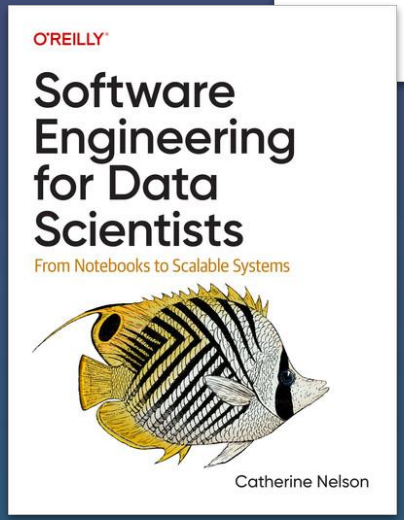
~10 years in data science with a strong focus on ML

Author of two O'Reilly books

Transitioning to startup AI Engineer role

Previously Principal Data Scientist at SAP Concur





https://github.com/catherinenelson1/notebooks-to-production

About Robert

12 years in data science with a focus on product analytics

10 years at FB/Meta, now at Atlassian working on Search, Chat and third party integrations

https://github.com/catherinenelson1/notebooks-to-production

Agenda for today

- We'll give an intro presentation
- *Your turn: download the repo and get set up*
- We'll give a short presentation on modular code
- *Your turn: convert part of a notebook into modular functions*
- We'll give a short presentation on unit tests
- *Your turn: write tests for your new functions*
- We'll wrap up with some final thoughts

https://github.com/catherinenelson1/notebooks-to-production

Introduction and setup

https://github.com/catherinenelson1/notebooks-to-production

Differences between notebooks and production code

- Notebooks are fantastic for exploration and experimentation
- Notebooks mix code, outputs and documentation together
- Production code is modular and reusable - forces you to break things into logical pieces
- This makes it easier to test, and guarantees that your results are what you expect
- Know when to graduate your code from exploration to something reliable and maintainable

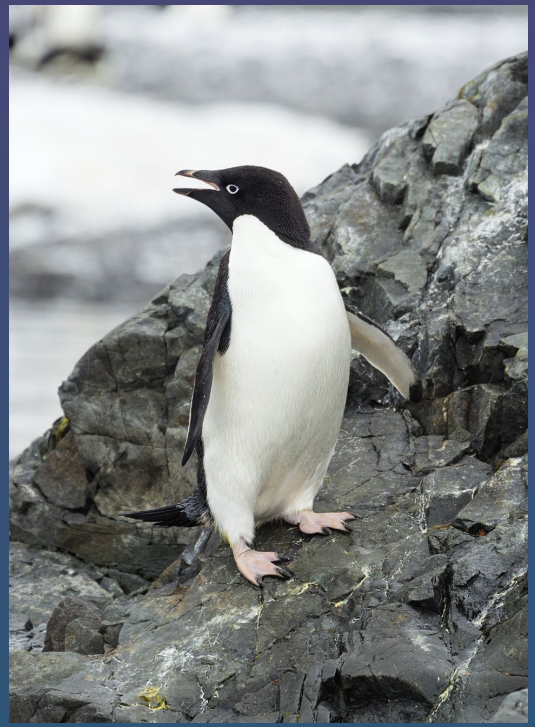
https://github.com/catherinenelson1/notebooks-to-production

Steps to go from notebooks to production

- Separate code out into different tasks
- Organize the code into functions
- Write the tests so that you know your code does what you expect it to do
- Make it shareable by sharing the modules it depends on (the requirements)
- Use CI/CD tools to deploy it

https://github.com/catherinenelson1/notebooks-to-production

Our example notebook – let's classify penguins!



By Andrew Shiva / Wikipedia, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=46714803

Penguins can be hard to tell apart

Given data on flipper size, bill length, bill depth, body mass, build a model to predict penguin species

Features: flipper size, bill length, body mass

Labels: 3 different species

Put the model in production

https://github.com/catherinenelson1/notebooks-to-production

Example notebook [DEMO]

https://github.com/catherinenelson1/notebooks-to-production

Your turn:

Clone the repo

Install the requirements in a virtual environment

Ensure that you can run the notebook to the end

https://github.com/catherinenelson1/notebooks-to-production

Use a virtual environment

```
python -m venv notebooks-to-prod

# Mac or Linux
source notebooks-to-prod/bin/activate

# Windows
notebooks-to-prod\Scripts\activate.bat

pip install -r requirements.txt
```

https://github.com/catherinenelson1/notebooks-to-production

What you should get

Make a prediction on new data

```
[30]: scaled_data = scaler.transform([[40, 17, 190, 3500]])

[31]: encoder.inverse_transform(lr.predict(scaled_data))[0]

[31]: 'Adelie'

[32]: encoder.inverse_transform(rf.predict(scaled_data))[0]

[32]: 'Adelie'

[33]: lr.predict_proba(X_test[:4])

[33]: array([[2.43316874e-03, 1.08422977e-02, 9.86724534e-01],
              [9.99192006e-01, 7.71119169e-04, 3.68744421e-05],
              [9.95382486e-01, 4.57623344e-03, 4.12810025e-05],
              [8.75121168e-01, 1.19155948e-01, 5.72288365e-03]])

[34]: rf.predict_proba(X_test[:4])

[34]: array([[0.  , 0.  , 1.  ],
              [0.99, 0.01, 0.  ],
              [1.  , 0.  , 0.  ],
              [0.53, 0.47, 0.  ]])
```

roduction

Modular code

https://github.com/catherinenelson1/notebooks-to-production

Principles of modular code

Modular code means your code is broken into small independent parts – it's the opposite of having all the code for your project in one giant script

As far as possible, make sure that each function or class has one well-defined purpose – one "thing" that it does

You should be able to describe it in a short sentence, like "this function creates a data visualization"

If you start using the word "and" in your description, this may be a clue to break it down into multiple functions

"This function cleans the data and creates a data visualization" should be divided into one function that cleans the data and another that creates the visualization

https://github.com/catherinenelson1/notebooks-to-production

Identify the steps in your notebook

What are the steps you need to keep?

Write a function stub for each step

For each function, what are the inputs and outputs?

Add in the code

Think about how it could go wrong and write a test

https://github.com/catherinenelson1/notebooks-to-production

Instructions for exercise 1

- Look at the notebook and try to identify the different tasks
- In the exercises folder, simple_script.py copies (most of) the code from the notebook into a script; *run it and see its output*
- In the exercises folder, choose which file you're going to work on
  - Choose modular_code_exercise if you want to implement 2 functions
  - Choose modular_code_function_stubs.py if you want to implement all the functions yourself
- Implement the functions based on the code in the notebook or in simple_script.py
- Run your code and check it gives you the same answer as simple_script.py

https://github.com/catherinenelson1/notebooks-to-production

Steps in the notebook

Load penguin data

Clean missing values

Scale/encode features and labels

Train model

Predict on new penguins

https://github.com/catherinenelson1/notebooks-to-production

From steps to function stubs

```
def download_data():
    pass

def clean_data():
    pass

def preprocess_data():
    pass

def train_model():
    pass

def predict_new_data():
    pass
```

https://github.com/catherinenelson1/notebooks-to-production

Solution for Exercise 1 [DEMO]

https://github.com/catherinenelson1/notebooks-to-production

Unit tests

https://github.com/catherinenelson1/notebooks-to-production

Why write tests?

A test is code that calls a function and checks that it does what it's supposed to do – it's a safety net for your code

Tests give you evidence and confidence that your code works correctly by verifying expected behavior

For tiny one-off experiments, tests may not be essential

But you should use them if your code is part of a larger system, other people are changing your code, or other code depends on yours to return a certain result

Testing gives you guarantees that your code is working, and you'll know if a change that you or someone else makes breaks it

https://github.com/catherinenelson1/notebooks-to-production

What goes in a test

1. Arrange: Set up everything you'll need to run the function, for example, load some data.
2. Act: Run the function you're testing.
3. Assert: Check that the result of running the function is what you expect.
4. Cleanup: Make sure the test doesn't leave any trace behind. For example, if you have opened a file, make sure to close it.

https://github.com/catherinenelson1/notebooks-to-production

An example

```
def weighted_mean(num_list, weights):
    running_total = 0
    for i in range(len(num_list)):
        running_total += (num_list[i] * weights[i])
    return (running_total/sum(weights))
```

https://github.com/catherinenelson1/notebooks-to-production

An example

```python
def test_weighted_mean():

    list_a = [1, 2, 4]
    list_b = [1, 2, 4]

    result = weighted_mean(list_a, list_b)

    assert result == 3
```

https://github.com/catherinenelson1/notebooks-to-production

An example

```
def test_weighted_mean():

    result = weighted_mean([1, 2, 4], [1, 2, 4])
    assert result == 3

    empty_list_result = weighted_mean([], [])
    assert not empty_list_result

    wrong_types_result = weighted_mean(['one', 2, 4], [1, 2, 4])
    assert not wrong_types_result
```

https://github.com/catherinenelson1/notebooks-to-production

Use a testing framework

- A testing framework handles finding and running your tests automatically
- Without it, you'd need to run every test function manually or write boilerplate code to run them
- This doesn't scale
- We'll use Pytest, a popular Python testing framework

https://github.com/catherinenelson1/notebooks-to-production

Instructions for exercise 2

- In the exercises folder, go to test_modular_code.py
- Complete the tests for clean_data and preprocess_data
- Use the Arrange/Act/Assert/Cleanup structure
- Test for some things you think might go wrong - poorly formatted data, missing data…
- Run the tests using `pytest test_modular_code.py`

https://github.com/catherinenelson1/notebooks-to-production

Solution for Exercise 2 [DEMO]

https://github.com/catherinenelson1/notebooks-to-production

Other topics we didn't cover

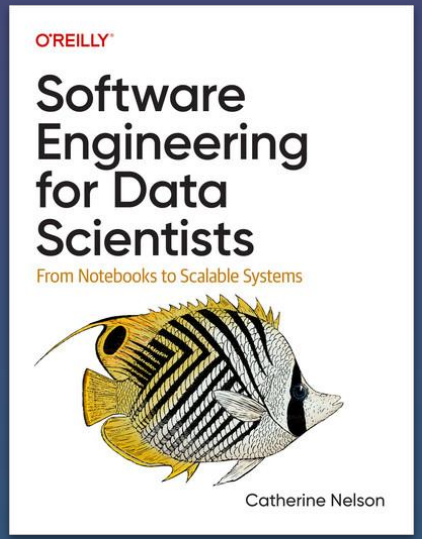
- Deployment
- Automation (CI/CD)
- Efficiency
- Logging
- Error handling

https://github.com/catherinenelson1/notebooks-to-production

But what about AI?
[DEMO]

https://github.com/catherinenelson1/notebooks-to-production

Further reading



https://github.com/catherinenelson1/notebooks-to-production