Testing for Data Science

What testing platforms are out there?

How many tests should you write?

How to make tests fast?

What is test driven development?

How to integrate test in your CI/CD?

How to test for Data Science?

Overview of last week

Folder structure

```
├── src
│   ├── data.py
│   └── model.py
├── tests
│   ├── pytest_basetemp
│   │   └── test_save_data0
│   │       └── file.csv
│   ├── conftest.py
│   ├── test_data.py
│   └── test_model.py
└── pyproject.toml
```

test_data.py

```python
import pytest
from src.data import read_data, save_data


@pytest.mark.parametrize("location, expected",
                         [("folder_a", 1000),
                          ("folder_b", 1000)])
def test_read_data(location, expected):
    result = read_data(path=location)
    assert len(result) == expected


def test_save_data(tmp_path):
    save_data(path=tmp_path)
```

pyproject.toml

```toml
[tool.pytest.ini_options]
testpaths = [
    "tests",
]
markers = [
"slow: marks tests as slow (deselect with '-m \"not slow\"')",
]
addopts = "--basetemp tests/pytest_basetemp"
filterwarnings = [
    'ignore::DeprecationWarning'
]
```

What testing platforms are out there?

How many tests should you write?

How to make tests fast?

What is test driven development?

How to integrate test in your CI/CD?

How to test for Data Science?

What testing platforms are out there?

unittest

doctest

nose

What testing platforms are out there?

```python
def factorial(n: float) -> float:
    """Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
    """

    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result
```

doctest

What testing platforms are out there?

unittest

```python
from unittest import TestCase
from src.utils import Connection


class TestExample(TestCase):
    @classmethod
    def setUpClass(cls):
        cls.conn = Connection()

    @classmethod
    def tearDownClass(cls):
        cls.conn.close()

    def setUp(self):
        self.sess = self.conn.new_session()

    def tearDown(self):
        self.sess.close()

    def test_boolean(self):
        self.assertTrue(self.sess.is_up())

    def test_session_status(self):
        self.assertEqual(self.sess.status(), "running")

    def test_multi_param_stuff(self):
        for i in range(10):
            with self.subTest(i):
                self.assertEqual(0, i)
```


```python
import pytest
from src.utils import Connection


@pytest.fixture(scope='session')
def connection():
    conn = Connection()
    yield conn
    conn.close()


@pytest.fixture
def session(connection):
    sess = connection.new_session()
    yield sess
    sess.close()


def test_boolean(session):
    assert session.is_up()


def test_session_status(session):
    assert session.status() == "running"


@pytest.mark.parametrize('i', range(10))
def test_multi_param_stuff(i):
    assert 0 == I
```

What testing platforms are out there?

How many tests should you write?

How to make tests fast?

What is test driven development?

How to integrate test in your CI/CD?

How to test for Data Science?

How many tests should you write?

Tests will…

- …detect errors/bugs.
- …prove correctness of code.
- …ensure faster maintenance.

Guidelines:

- Aim for a test coverage > 80%.
- Write a test, before fixing a bug.

Invoke:

```
pytest –-cov src                        # Run pytest and show coverage of (files in) src.
pytest --cov-report html --cov src      # Run pytest, coverage & create report in html.
```

What testing platforms are out there?

How many tests should you write?

How to make tests fast?

What is test driven development?

How to integrate test in your CI/CD?

How to test for Data Science?

How to make tests fast?

Tell test-runner where your tests are located

```
testpaths = [
      "tests",                          # In pyproject.toml.
]
```

Find slow tests

```
pytest --durations=10                   # Shows times of 10 slowest tests.
pytest --durations=-1                   # Shows times of all tests.
```

Only run a selection of tests

```
pytest -m <marker>                      # Select by marker.
pytest -k <name>                        # Select by name.
pytest --last-failed (--lf)             # Only runs test that failed in the last run.
pytest --failed-failed (--ff)           # Runs failed tests before other tests.
pytest --new-first (--nf)               # Runs new tests before other tests.
pytest -x                               # Stop if 1 test fails.
pytest --stepwise (--sw)                # Runs tests until 1 fails, continues from that one the next time.
pytest --stepwise -stepwise-skip        # Skips one failing test, in the stepwise process.
```

How to make tests fast?

Monkeypatch (fixture)
Replaces functions, classes, dictionaries etc. with a “fake” version during a test.

data_model.py

```python
from random import random

class Employee:
    def __init__(self):
        self.salary = int(random() * 1000)
```

test_data_model.py

```python
from data_model import Employee
import data_model


def test_employee(monkeypatch):
    monkeypatch.setattr(data_model, "random", lambda: 0.5)
    employee = Employee()
    assert employee.salary == 500
```

How to make tests fast?

Monkeypatch (fixture)
Replaces functions, classes, dictionaries etc. with a “fake” version during a test.

app.py

```python
import requests
from datetime import date

def max_temperature_on_day(day: date):
    response = requests.get(
        f"https://www.weather.com/temperatures?day={day}")
    json_data = response.json()
    return max(json_data["temperatures"])
```

test_app.py

```python
import requests
from app import max_temperature_on_day


class MockResponse:
    @staticmethod
    def json():
        return {"temperatures": [7, 7, 6, 4, 4, 4, 7, 9, 13, 17, 16, 17,
                                 17, 17, 18, 9, 9, 9, 8, 7, 8, 7, 7, 6]}


def test_max_temperature_on_day(monkeypatch):

    def mock_get(*args, **kwargs):
        return MockResponse()

    monkeypatch.setattr(requests, "get", mock_get)

    result = max_temperature_on_day(day=date(12, 12, 2001))
    assert result == 18
```


bigdata republic

How to make tests fast?

Monkeypatch (fixture)
Replaces functions, classes, dictionaries etc. with a “fake” version during a test.

Overview

```
def test_with_mock(monkeypatch):
    monkeypatch.setattr(target=, name=, value=, raising=False)
    monkeypatch.delattr(target=, name=, raising=True)
    monkeypatch.setitem(dic=, name=, value=)
    monkeypatch.delitem(dic=, name=, raising=True)
    monkeypatch.setenv(name=, value=, prepend=None)
    monkeypatch.delenv(name=, raising=True)
    monkeypatch.syspath_prepend(path=)
    monkeypatch.chdir(path=)
```

How to make tests fast?

Cache

Place where pytest saves test_ids from discovery, outcomes of previous test runs, etc.
But you can also store your own values.
Clear the cache for a fresh run with: `pytest --cache-clear`

The example

```python
import pytest
import time


def expensive_computation():
    time.sleep(10)  # running expensive computation...


@pytest.fixture
def mydata(request):
    val = request.config.cache.get("example/value", None)
    if val is None:
        expensive_computation()
        val = 42
        request.config.cache.set("example/value", val)
    yield val


def test_function(mydata):
    assert mydata == 23
```

Terminal

```
$ pytest tests/test_cache.py::test_function
========================= test session starts =========================
platform win32 -- Python 3.7.11, pytest-7.1.1, pluggy-0.13.1
cachedir: tests\.pytest_cache
rootdir: ...\testing_for_data_science_day_2, configfile: pyproject.toml

plugins: cov-2.9.0, env-0.6.2, mock-3.6.1
collected 1 item

tests\test_cache.py .                                            [100%]
========================== 1 passed in 10.01s =========================


$ pytest tests/test_cache.py::test_function
========================= test session starts =========================
platform win32 -- Python 3.7.11, pytest-7.1.1, pluggy-0.13.1
cachedir: tests\.pytest_cache
rootdir: ...\testing_for_data_science_day_2, configfile: pyproject.toml

plugins: cov-2.9.0, env-0.6.2, mock-3.6.1
collected 1 item

tests\test_cache.py .                                            [100%]
========================== 1 passed in 0.02s ==========================
```

What testing platforms are out there?

How many tests should you write?

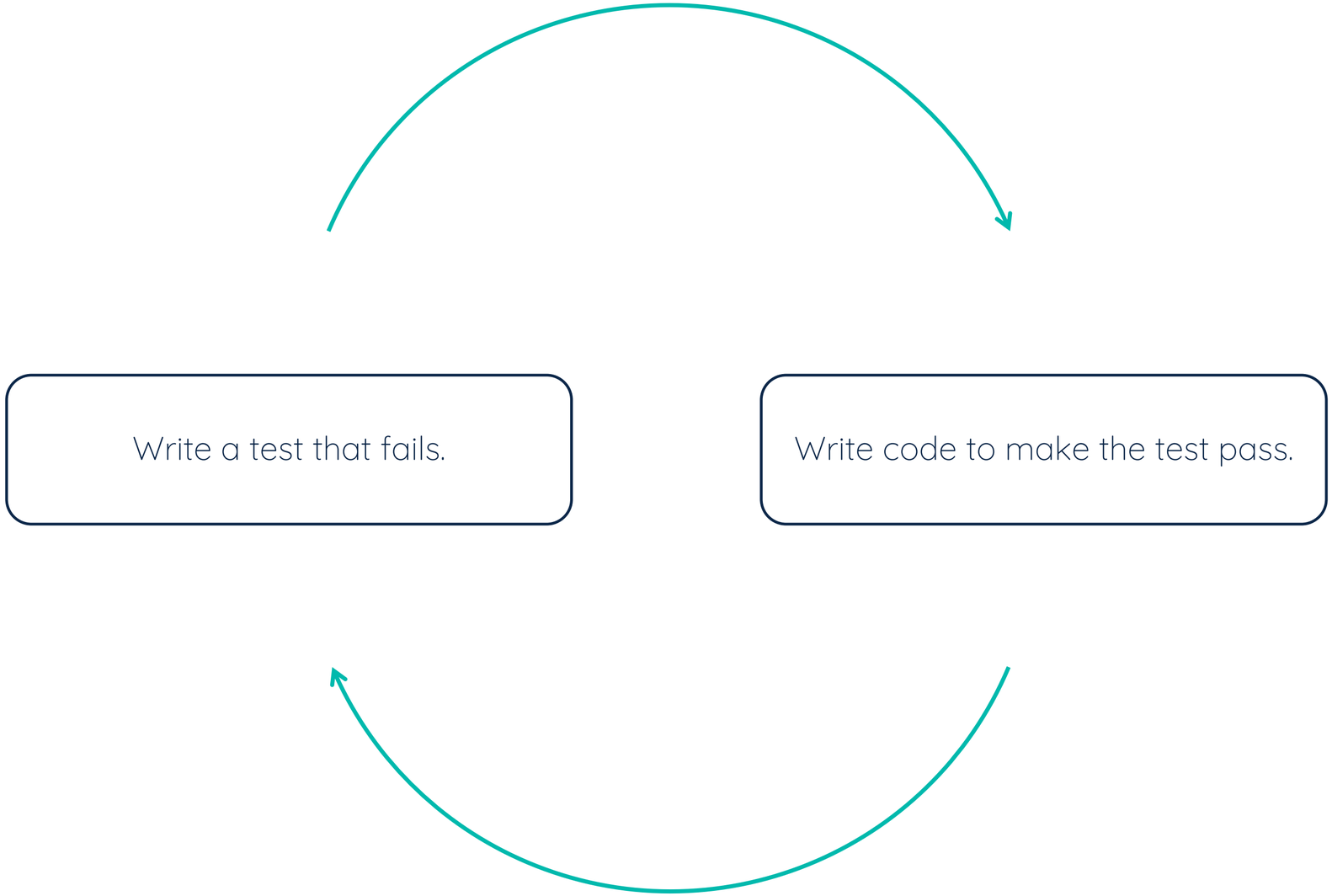
How to make tests fast?

What is test driven development?

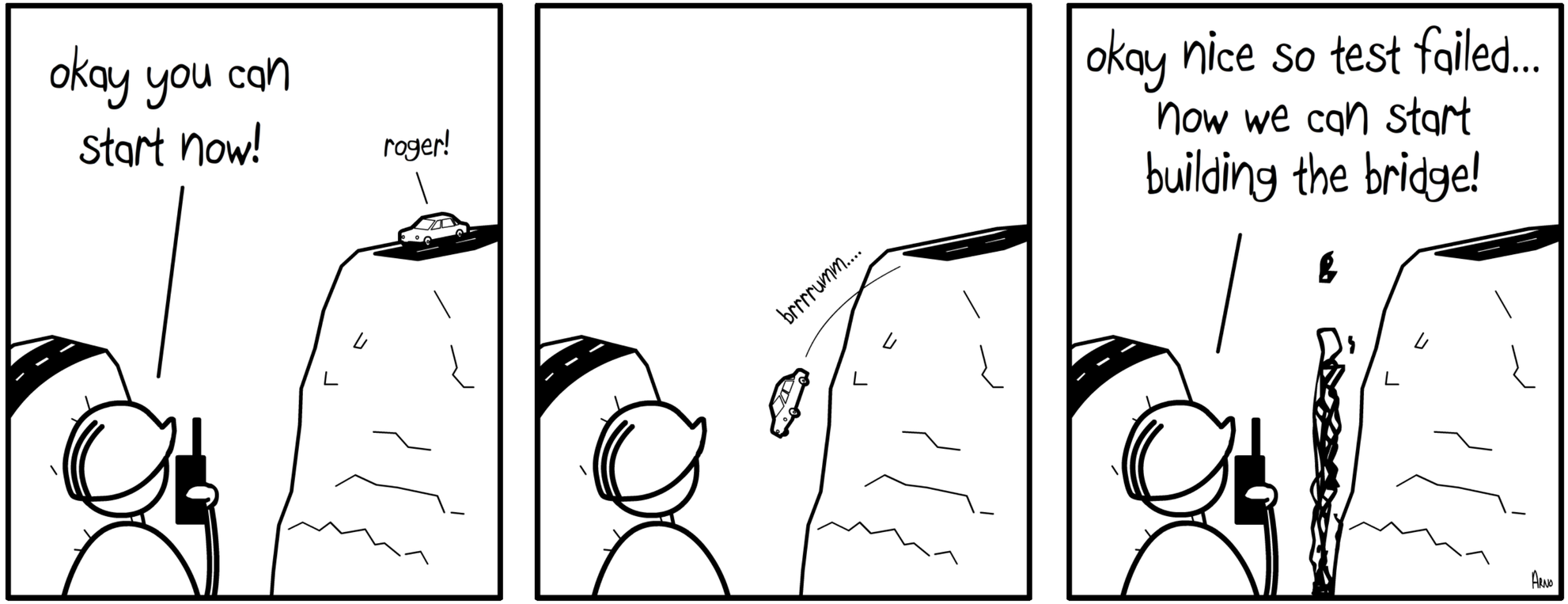
How to integrate test in your CI/CD?

How to test for Data Science?

What is test driven development?



What is test driven development?



What testing platforms are out there?

How many tests should you write?

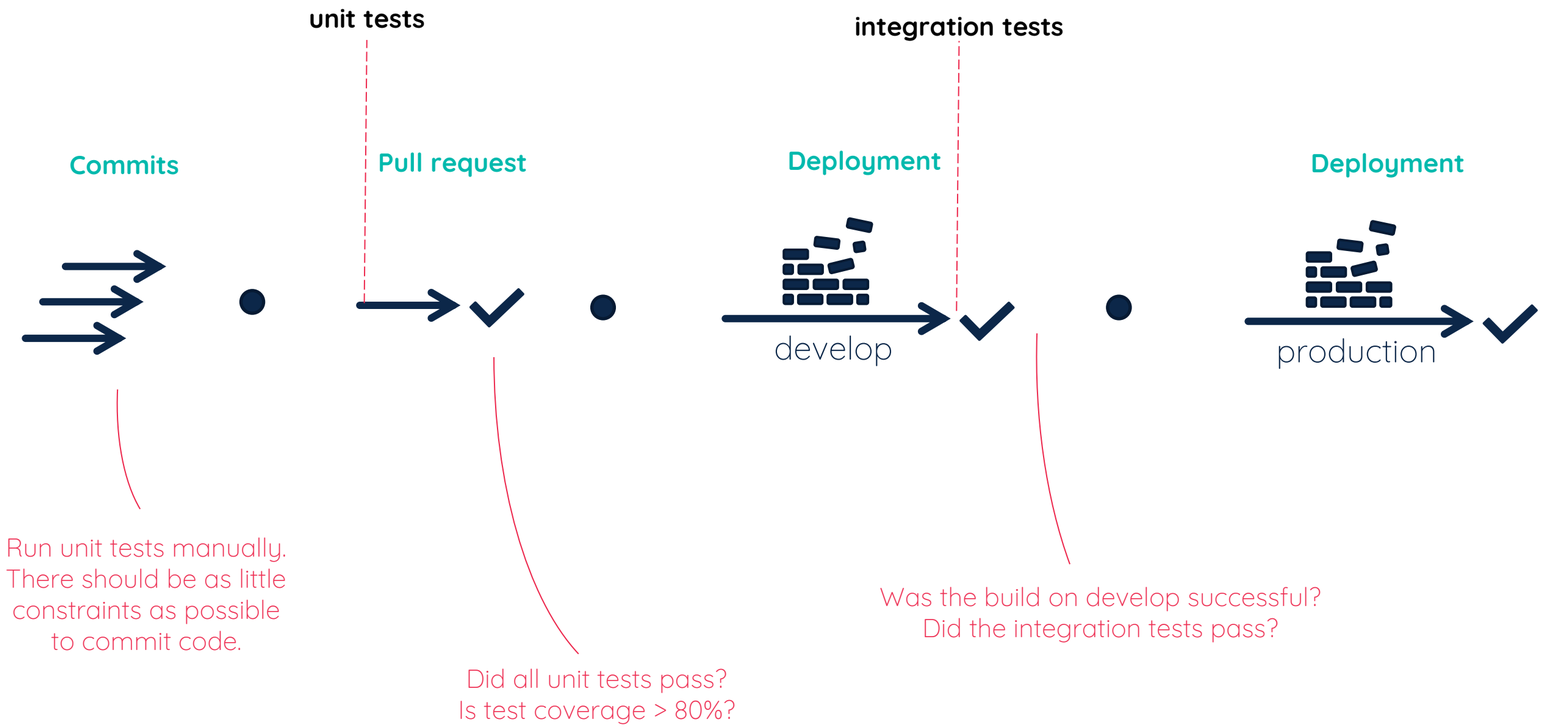
How to make tests fast?

What is test driven development?

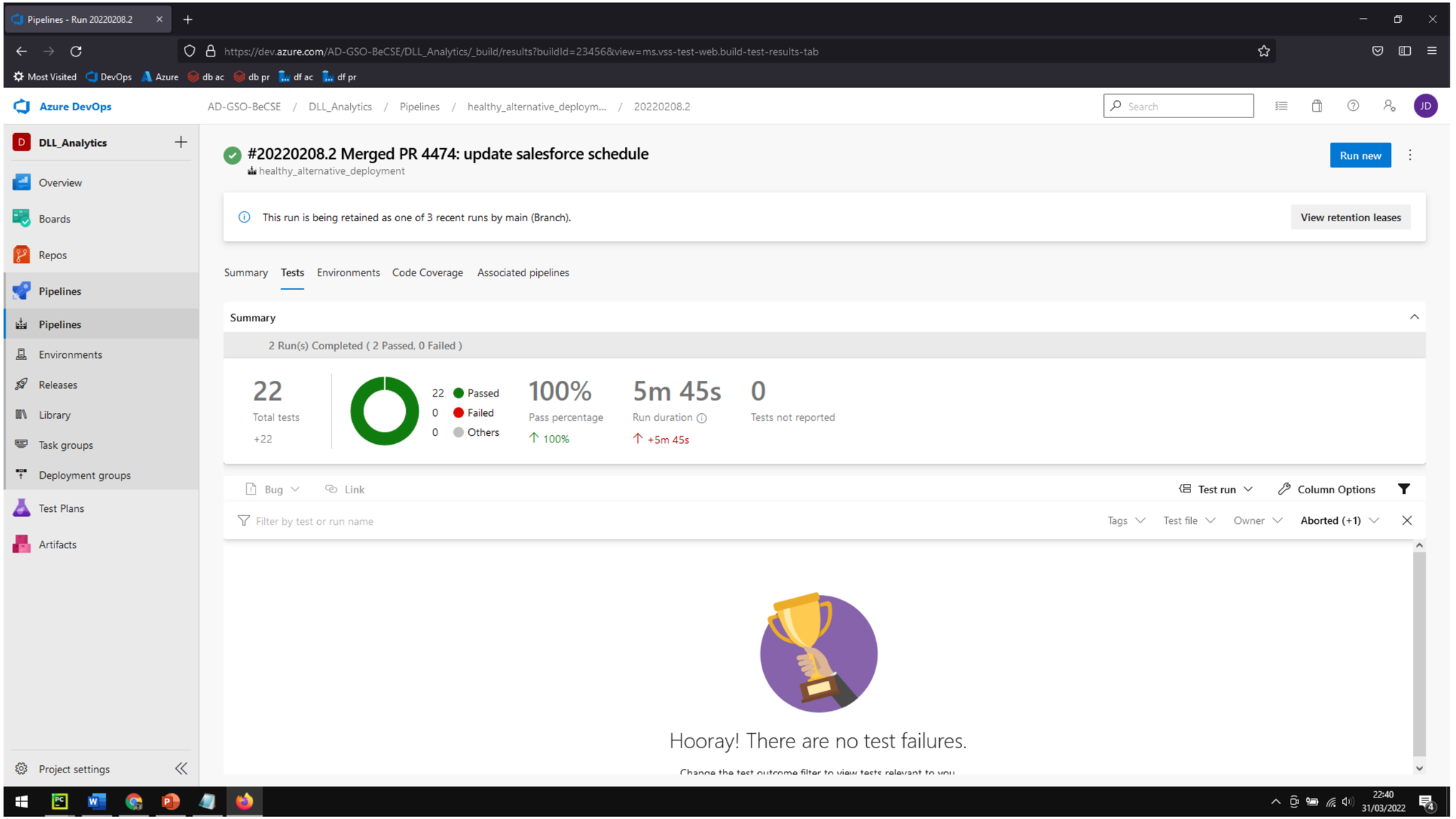
How to integrate test in your CI/CD?

How to test for Data Science?

How to integrate test in your CI/CD?



How to integrate test in your CI/CD?



bigdata republic

What testing platforms are out there?

How many tests should you write?

How to make tests fast?

What is test driven development?

How to integrate test in your CI/CD?

How to test for Data Science?

How to test for Data Science?

- Don’t confuse testing with data validation.
- Create dummy data.
  - Commit it to the repo, to make test repeatable.
  - Anonymize all personal data.
  - Automate this process in a script.
- Compare numbers with tolerances.

No

```python
import pandas as pd


def read_article_data(filepath: str):
    return pd.read_csv(file=filepath)
```

Compare numbers

```python
import numpy as np
import pandas as pd
from pytest import approx
import numpy as np
import pandas as pd
from pytest import approx


def test_add():
    assert 0.1 + 0.2 == 0.3  # False
    assert 0.1 + 0.2 == approx(0.3)  # True
    assert 0.1 + 0.2 == approx(0.3, rel=1e-6, abs=1e-12)  # True

def test_numpy():
    assert np.array([0.1, 0.2]) + np.array([0.2, 0.4])\
           == approx(np.array([0.3, 0.6]))
    assert np.array([0.1, 0.2]) + np.array([0.2, 0.1]) == approx(0.3)

def test_pandas():
    pd.testing.assert_frame_equal(pd.DataFrame([0.1, 0.2]) +
                                  pd.DataFrame([0.2, 0.4]),
                                  pd.DataFrame([0.2, 0.6]))
```

```python
    assert list(article_data.id) == ["apple"]
```


bigdata republic

Phone +31 (0)168 479294 **Email** info@bigdatarepublic.nl **Address** Coltbaan 4C, 3439 NG Nieuwegein, The Netherlands