

Git & CI/CD

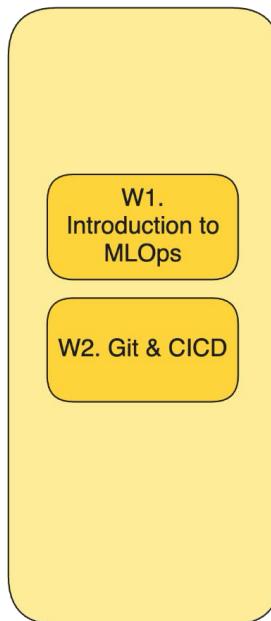
Sprint 1 | Week 2

INFO9023 - ML Systems Design

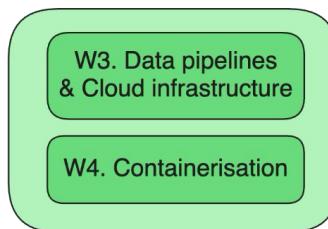
Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our roadmap

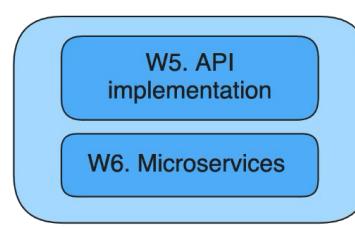
Sprint 1: Project organisation



Sprint 2: Cloud & containerisation



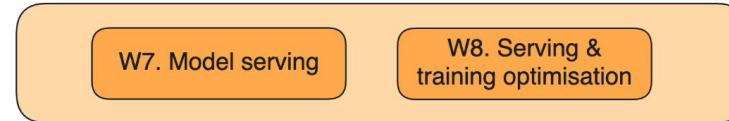
Sprint 3: API implementation



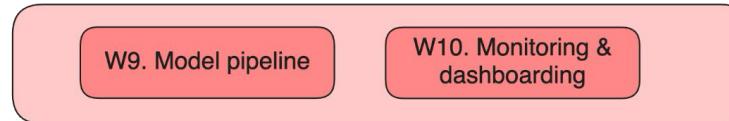
Sprint 6: LLMOps



Sprint 4: Model serving & optimisation



Sprint 5: Pipeline & monitoring



Agenda

What will we talk about today

Lecture

- Git: code versioning & conventions
- CICD
- Code testing
- Data sources

Directed Work

- Git & CICD

Git: code versioning & conventions

Why does versioning matter?

Name	Date modified	Type	Size
final-draft	8/1/2017 3:48 PM	Microsoft Word D...	16 KB
final-draft2	8/3/2017 3:49 PM	Microsoft Word D...	12 KB
final-draft-final	8/3/2017 10:22 PM	Microsoft Word D...	13 KB
final-modified	8/8/2017 3:49 PM	Microsoft Word D...	13 KB
real-final-draft	8/9/2017 2:22 PM	Microsoft Word D...	13 KB
real-final-draft-v2	8/10/2017 12:33 PM	Microsoft Word D...	16 KB
final	8/11/2017 11:59 PM	Microsoft Word D...	72 KB



Julien Chaumond
@julien_c

Modern ML engineering is 90% knowledge of git

[Traduci il Tweet](#)

4:40 PM · 20 ago 2021 · Twitter for iPhone



Versioning applied on different parts of your solution.



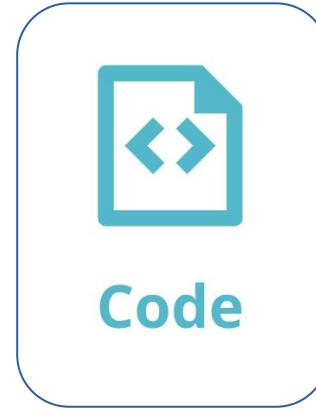
Data

+



Model

+



Code

Focus for today!

Code versioning & Git

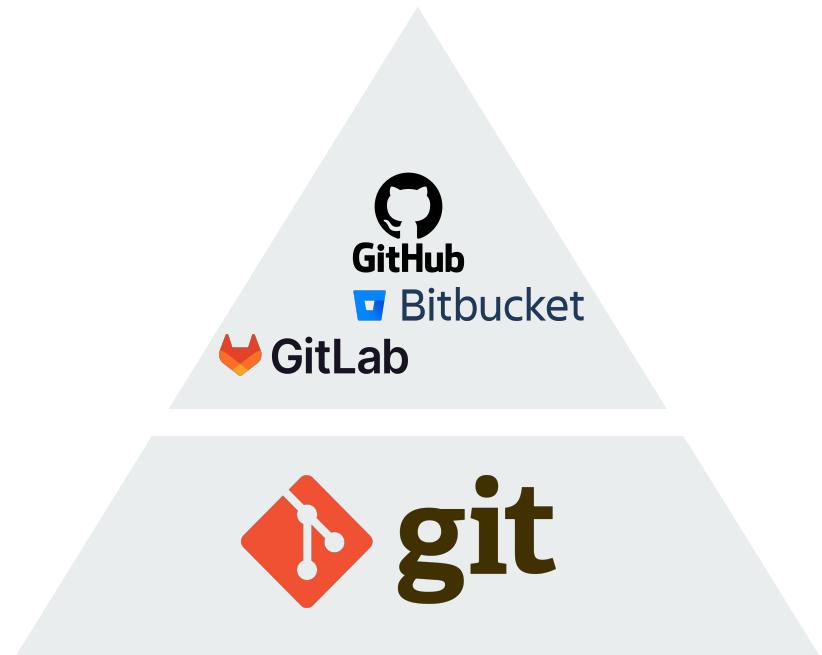
Code versioning: Tracking, organising and controlling changes in the code. Enables:

- Collaboration
- Track history
- Revert
- Versioning

Git: Standard solution for code versioning

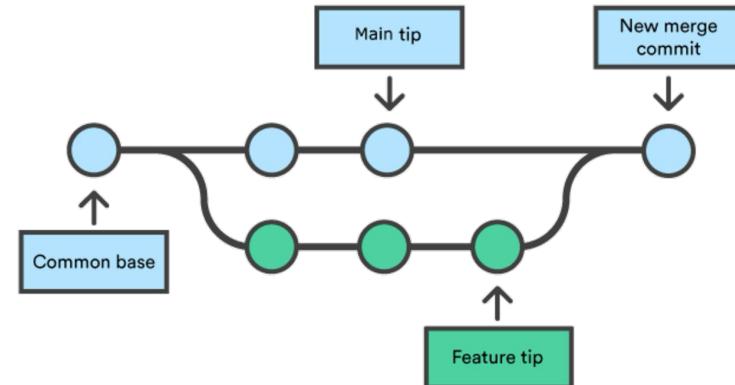
Source Code Management (SCM): Tools on top of git to manage people, process, and automation around Git. Examples:

- Github
- Gitlab
- Bitbucket
- ...



Git branching and merging

- **Goal:** work at the same time with multiple versions of a repository
 - A branch is an independent line of development
 - As default, a repository has a unique branch called **main**
 - Another branch is used to work at changes and extensions before a merge with the main
 - For each issue X (feature, bug, etc.):
 - **Create** a new branch X
 - **Implement** X
 - **Merge** X changes with the main

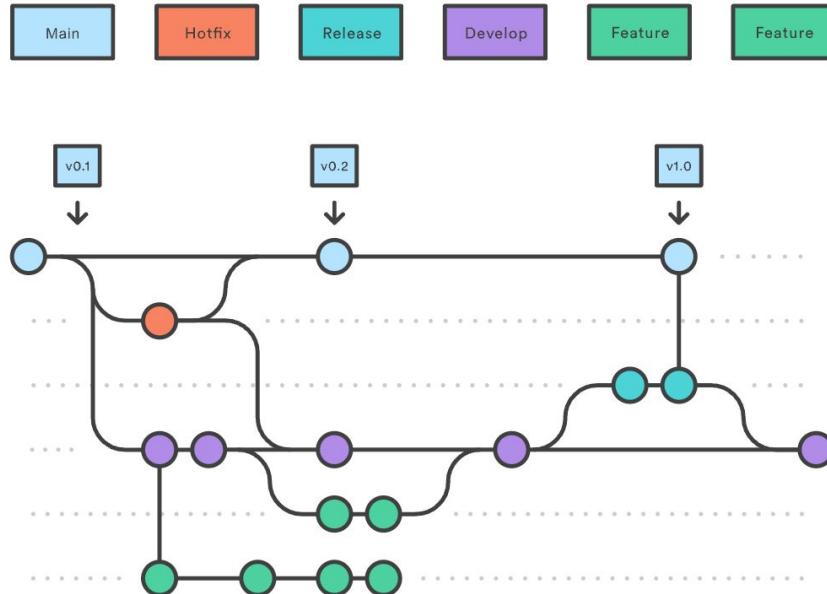


GitFlow

Set of practices around Git.

4 types of primary branches

1. **Main** branch stores the official release history (commits with a **version number**)
2. **Develop** branch serves as working branch for new features
3. **Feature** branch gets created each time a specific new features needs to be implemented
4. **Release** cycle - only bug fixes and documentation added to this branch
5. **Hotfix** to quickly patch and maintain production release



<https://nvie.com/posts/a-successful-git-branching-model/>

<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

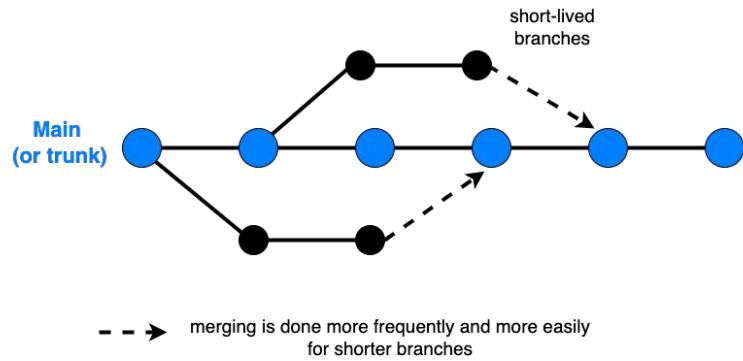
Git Trunk-based development

- Developers merge **small, frequent** updates to a core “**trunk**” or main branch
- Based on automatic building and testing of the solution (requires CICD)
- In trunk-based development, code review should be performed immediately (within a day)

Some advocates name it as the new thing...

Trunk-based development

StatusNeo



GitFlow vs Trunk based

GitFlow	Trunk based
Large feature branches	Small, frequent updates to a core “trunk”
Active collaboration	CICD
Multiple base branch	Automatic testing
Large PRs	Small and frequent PRs
Different phases of development (dev, release, hotfix, ...)	No planned code freeze or pause of integration
Still very popular	Documented as the next best thing



Often, teams operate in between.

Traditional Gitflow framework while trying to work with as small branches as possible.

GitFlow vs Trunk based

GitFlow	
Large feature branches	Large feature branches
Active collaboration	Active collaboration
Multiple base branch	Multiple base branch
Large PRs	Large PRs
Different phases of development (hotfix, ...)	Pause of integration
Still very popular	First thing

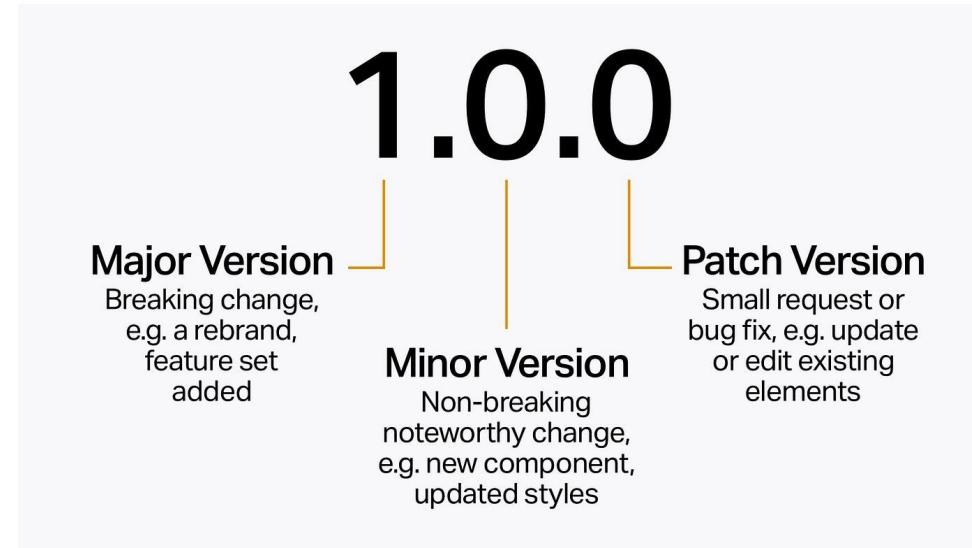
A cartoon illustration of SpongeBob SquarePants. He is smiling broadly with his arms raised. A vibrant rainbow arches over his head, with white stars scattered along its curve. The background is a simple brown gradient. In the center, the words "KEEP YOUR BRANCHES SHORT AND CONCISE" are written in large, bold, white capital letters.

↔ imgflip.com ↔

Often, teams operate in between.

Traditional Gitflow framework while trying to work with as small branches as possible.

Product versioning convention



The .gitignore file

Issues with Git:

- Pain to erase memory
 - *Don't ever push any sensitive information on git!*
- Limited storage space (~2 GB)



The .gitignore file

Issues with Git:

- Pain to erase memory
 - *Don't ever push any sensitive information on git!*
- Limited storage space (~2 GB)

.gitignore file!

- Patterns for ignoring files to be submitted to git
- Usually set with standard file patterns
- Many examples: <https://github.com/github/gitignore/tree/main>

Any of the characters in square brackets (.pyc, .pyo, .pyd)

Remove files per extension

Remove files per location



```
# These are some examples of commonly ignored file patterns.

# Compiled Python bytecode
*.py[cod]

# Log files
*.log

# Environment
*.env
venv/

# Data
data/

# Secrets
secrets/

# Jupyter Notebook
.ipynb_checkpoints
```

The .dockerignore file

Fun fact...

Nothing to do with gitignore!
It will remove files from your
Docker image

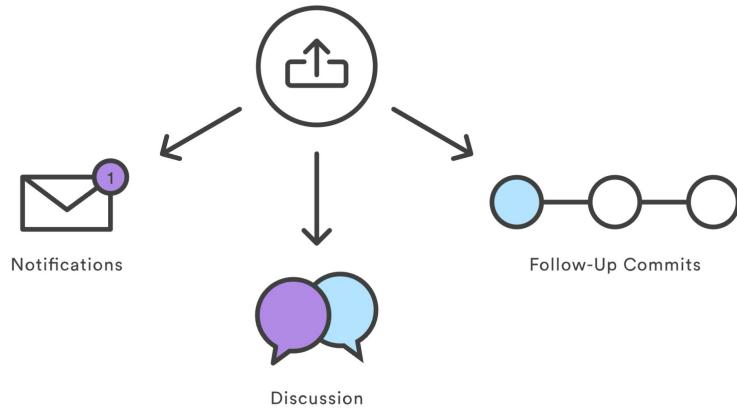
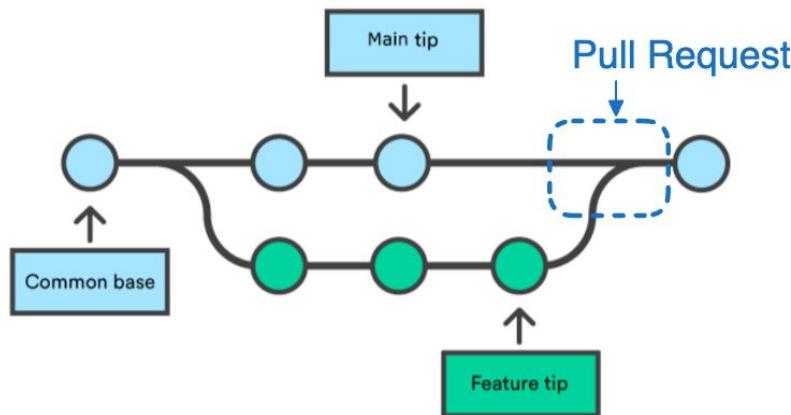
A screenshot of a GitHub pull request titled "Remove markdown from docker ignore #54". The pull request has been merged by ThomasVrancken. It shows two commits: one changing .dockerignore and one changing .gcloudignore. The .dockerignore file contains rules to ignore .sh, README.md, docs/*.md, Dockerfile, and __pycache__. The .gcloudignore file contains rules to ignore *.swp, *.swo, and documentation files like README.md.

```
diff --git a/.dockerignore b/.dockerignore
--- a/.dockerignore
+++ b/.dockerignore
@@ -1,4 +1,5 @@
**/*.sh
- *.md
+ README.md
+ docs/*.md
Dockerfile
**/__pycache__/

diff --git a/.gcloudignore b/.gcloudignore
--- a/.gcloudignore
+++ b/.gcloudignore
@@ -15,8 +15,8 @@ __pycache__/
*.swp
*.swo
-
- # Ignore documentation
- *.md
+
+ # Ignore documentation (but not prompt .md files in app/agentic/prompts/)
+ README.md
docs/
```

Pull requests

- **Pull Request** ⇒ Process to merge branch into main
- Select members that need to **review the codes** before the branch is merged to main.



What is the correct name...

Pull request == Merge request → Synonyms



Pull requests: Code review

Why it is important

- Benefits of code review:
 - **Quality Assurance:** Is this actually the best implementation?
 - **Knowledge sharing:** Aligned team on how the whole code base works.
 - **Component alignment:** Possible catch of logic flaws outside of the feature being reviewed.
 - **Coding standards:** Consistent over code base
 - **Mentorship:** Guidance for less experienced developers
 - **Documentation:** Guarantees that codes are readable
 - **Team morale:** We work as a group, less isolated features
- Tech-lead needs to **dedicate time to code review**

Strong correlation between developer team maturity and enforcement of code reviews!

Be smart - shouldn't hinder team efficiency (especially in early stages of a project)

Pull requests: Code review

What reviewers should look at

- **Design:** Is the code well-designed and appropriate for your system?
- **Functionality:** Does the code behave as the author likely intended? Is the way the code behaves good for its users?
- **Complexity:** Could the code be made simpler? Would another developer be able to easily understand and use this code when they come across it in the future?
- **Tests:** Does the code have correct and well-designed automated tests?
- **Naming:** Did the developer choose clear names for variables, classes, methods, etc.?
- **Comments:** Are the comments clear and useful?
- **Style:** Does the code follow our [style guides](#)?
- **Documentation:** Did the developer also update relevant documentation?

Docstring documentation

Used to document your function. “*Would I be comfortable that someone else can easily reuse this code?*

Describe the general **functionality**, the input **arguments** and what it **returns**. Example styles:

```
"""
This is a reST style.

:param param1: this is a first param
:param param2: this is a second param
:returns: this is a description of what is returned
:raises KeyError: raises an exception
"""
```

[reStructuredText](#) (reST)

```
"""
This is an example of Google style.

Args:
    param1: This is the first param.
    param2: This is a second param.

Returns:
    This is a description of what is returned.

Raises:
    KeyError: Raises an exception.
"""
```

[Google](#)

Docstring do

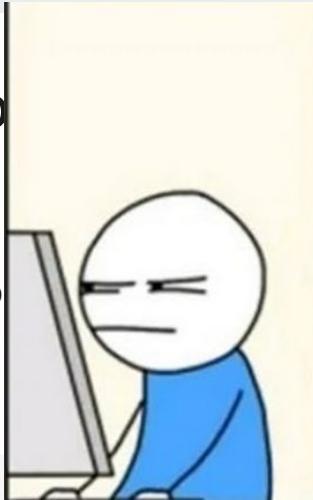
Used to document yo

Describe the general

```
....  
This is a reST style.
```

```
:param param1: this is a firs  
:param param2: this is a sec  
:returns: this is a descripti  
:raises KeyError: raises an e  
....
```

[reStructuredText](#)



**When I wrote this code,
only God & I understood
what it did.**



use this code?
yles:

**Now.....
only God knows.**

What is wrong with the docstring in this code snippet?



Pathetic.
Imgflip.com

```
def calculate_area(length, width):
    '''Calculates the area of a rectangle. This method takes two arguments, the length and
width of the rectangle, and returns the area.
    Arguments:
        length - the length of the rectangle.
        width - the width of the rectangle.
    Returns: The area of the rectangle calculated using the formula length*width.'''
    return length * width
```

Markdown documentation

Two levels, two purposes (1/2)

1. README.md | Front door

General introduction to the repository/project. Does not contain actual component documentation

- Project Overview
- Setup & Installation
- Usage
- Project Structure
- Contributors

```
# Foobar
Foobar is a Python library for dealing with word pluralization.

## Installation

Use the package manager \[pip\](https://pip.pypa.io/en/stable/) to install foobar.

```bash
pip install foobar
```

## Usage

```python
import foobar

returns 'words'
foobar.pluralize('word')

returns 'geese'
foobar.pluralize('goose')
returns 'phenomenon'
foobar.singularize('phenomena')
```

```

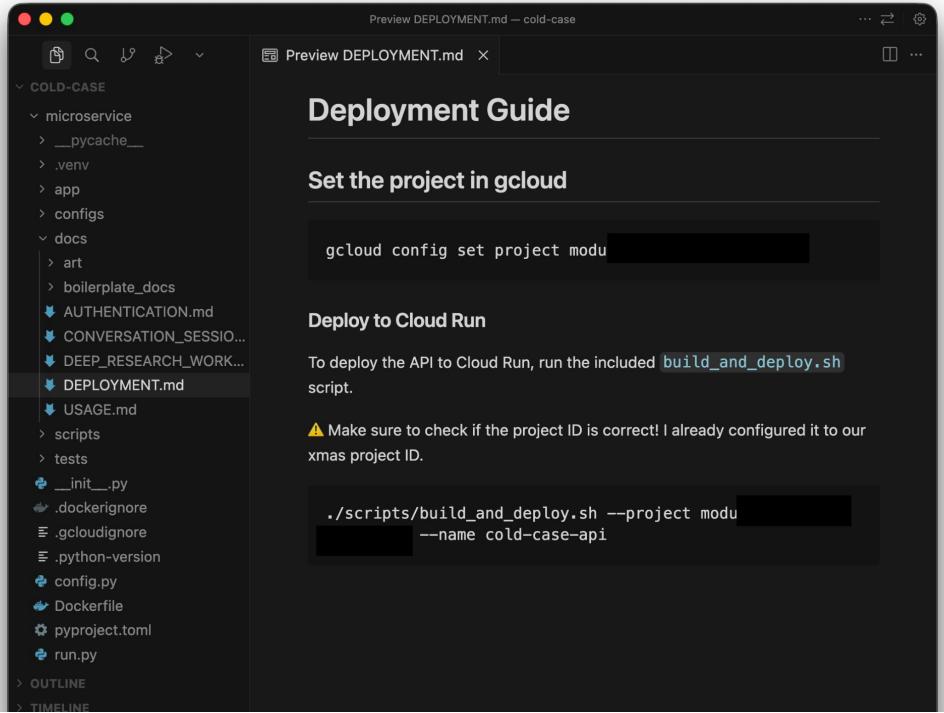
Markdown documentation

Two levels, two purposes (2/2)

2. docs/*.md | Deep dives

Deeper dives into key documentation, different files per topics:

- Architecture
- Key design decisions
- Deployment
- Infrastructure
- ...



The screenshot shows a Mac OS X desktop environment. On the left, a file browser window titled 'Preview DEPLOYMENT.md — cold-case' displays the directory structure of a project named 'cold-case'. The 'docs' folder contains several files: AUTHENTICATION.md, CONVERSATION_SESSIO..., DEEP_RESEARCH_WORK..., DEPLOYMENT.md (which is selected), and USAGE.md. Other files in the 'docs' folder include art, boilerplate_docs, and scripts. The 'DEPLOYMENT.md' file is also shown in the preview window on the right.

Deployment Guide

Set the project in gcloud

```
gcloud config set project modu
```

Deploy to Cloud Run

To deploy the API to Cloud Run, run the included `build_and_deploy.sh` script.

⚠ Make sure to check if the project ID is correct! I already configured it to our xmas project ID.

```
./scripts/build_and_deploy.sh --project modu  
--name cold-case-api
```

Markdown documentation

Two levels, two purposes (2/2)

Rubber ducking: Tell yourself what exactly you're doing here. Good way to voice a problem. (Usually for debugging but also for documentation).



The screenshot shows a Mac desktop interface. On the left is a file browser window titled 'COLD-CASE' containing a directory structure for a Python project. On the right is a preview window titled 'Preview DEPLOYMENT.md — cold-case'. The preview content includes:

Deployment Guide

Set the project in gcloud

```
gcloud config set project modu
```

Deploy to Cloud Run

To deploy the API to Cloud Run, run the included `build_and_deploy.sh` script.

⚠ Make sure to check if the project ID is correct! I already configured it to our xmas project ID.

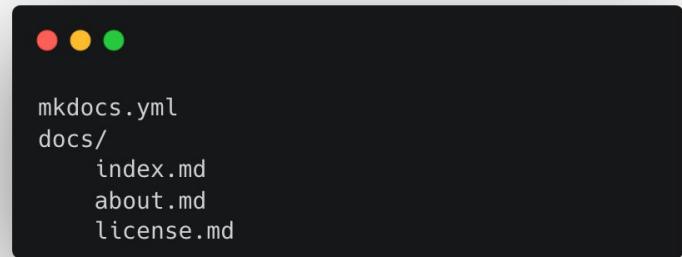
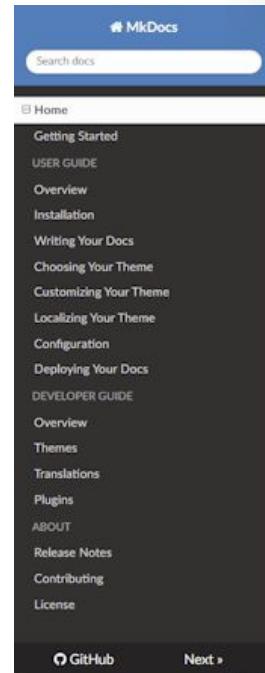
```
./scripts/build_and_deploy.sh --project modu  
--name cold-case-api
```

Markdown documentation

You can use tools such as **MkDocs** or **docusaurus** to produce a full documentation page from your directory.

It takes in a certain structure of documentation and will generate an HTML page.

Identifies docstrings and other types of documentation directly from your codebase.



A screenshot of the MkDocs landing page. The header includes the 'MkDocs' logo, a 'Docs > Home' breadcrumb, an 'Edit on GitHub' button, and a 'Next' button. The main content area has a section titled 'MkDocs' with the subtext 'Project documentation with Markdown.' Below this is a paragraph about MkDocs being a fast, simple, and beautiful static site generator. There are two green buttons for 'Getting Started' and 'User Guide', followed by a 'Features' section. Under 'Features', there are sections for 'Great themes available' (mentioning built-in themes like mkdocs and readthedocs), 'Easy to customize' (mentioning customization via themes and plugins), and 'Many configuration options are available'. The footer contains the 'MkDocs' logo and a 'GitHub' link.

Scripting vs notebooks

Scripts (*.py files):

- **Stateless**
 - Scripts do *not* save variables and what you run in the **global state**. Therefore they are **stateless** (unlike notebooks which are **stateful**)
 - Scripts require passing variables to functions and classes. Healthy to **structure** your implementation.
- **Linear**
 - Scripts run everything **from beginning to end linearly**, which takes away the risk of running cells in the wrong order
- **Reproducibility**
 - Easier to **reproduce** and **run tests** with different parameters with scripts.

"Only a Sith deals in absolutes"

⇒ notebooks aren't bad, they just serve a different purpose ⇒ **Experimentation** (which requires being messy)

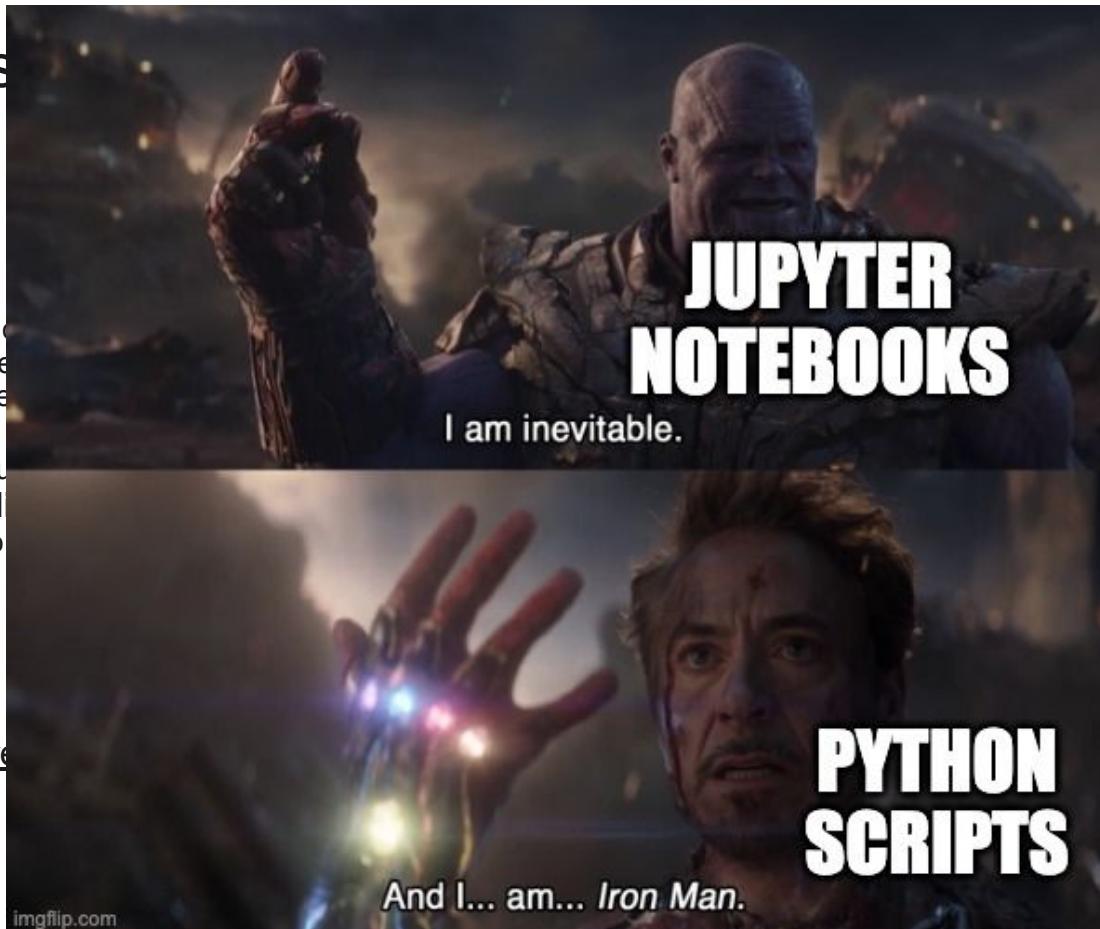
Scripting vs Notebooks

Attributes of scripts:

- **Stateless**
 - Scripts don't remember which are run before.
 - Scripts repeat the same thing every time.
- **Linear**
 - Scripts run sequentially from top to bottom.
- **Reproducible**
 - Easier to reproduce results.

"Only a sith deals in

⇒ notebooks are
messy)



s (unlike notebooks
tation.

ells in the wrong order

quires being

Code convention: Python PEP8

- Ensures a consistent code quality across a team.
- Set of rules on styling, indenting, naming conventions and documentation
- Integrate/automate in your code editor (IDE) to make it easy.
Often enforced PEP8 during pull request submission.
- 🧑 Developers can be judgy... Make your life easy, adapt clean codes

*Tools to automatically check compliance with
PEP8 will be covered in the CICD directed
work!*



Code convention: Python PEP8

- Ensures a consistent style
- Set of rules on style, documentation
- Integrate/automate
- Often enforced by tools
- 🧑 Developers can write better codes



ject):

pected 2 blank lines after class or function definition, found 1

Alt+Shift+Enter More actions... Alt+Enter

Code convention: Python PEP8

- Code Layout and Indentation
 - Use 4 spaces per indentation level. Avoid mixing tabs and spaces
- Line Length
 - Limit lines to 79 characters, and 72 characters for comments or docstrings
- Blank Lines
 - Surround top-level function and class definitions with two blank lines. Use a single blank line to separate logical sections within a function or method.
- Imports
 - Group imports into three sections: standard library, third-party libraries, and local application/library-specific imports. Each group should be separated by a blank line, and avoid wildcard imports (from module import *).
- Naming Conventions
 - Use snake_case for variable and function names, UPPERCASE for constants, and CamelCase for class names. Avoid single-character variable names, except for counters or iterators.
- Comments and Docstrings
 - Write clear and concise comments that explain why something is done, not how. Use triple double-quotes (""""") for module, class, and method docstrings.
- Function and Variable Annotations
 - Use type hints for function arguments and return values (e.g., def add(x: int, y: int) -> int:).
- Exceptions
 - Use specific exception types rather than a generic except: block. Always clean up resources (e.g., files) with context managers like with open().

What is wrong with this code snippet?



Pathetic.
Imgflip.com

```
import sys, os

def some_function(x,y):
    print('x+y=',x+y)
    if (x==0):
        print("x is 0!")
        x=x+1
    for i in range(0,10): print(i)
    try:os.listdir('.')
    except Exception as e:print(e);sys.exit(1)
```

Cookie cutters: Initiating your repository

Many [**cookie cutters**](#) exist to easily set up specific kinds of code repositories.

The [Cookiecutter Data Science](#) project sets up a repository structure tailored to data science projects.

 **Disclaimer:** A bit old and subjective. E.g. saving data locally is not best practice.

```
LICENSE           <- The top-level README for developers using this project.  
README.md  
  
notebooks         <- Jupyter notebooks.  
  
references        <- Data schemas, manuals, and all other explanatory materials.  
  
reports           <- Generated analysis as HTML, PDF, LaTeX, etc.  
    figures        <- Generated graphics and figures to be used in reporting  
  
requirements.txt  <- The requirements file for reproducing the analysis environment, e.g.  
                    generated with `pip freeze > requirements.txt`  
  
setup.py          <- makes project pip installable (pip install -e .) so src can be imported  
src               <- Source code for use in this project.  
.gitignore        <- Specify rules for code versioning
```

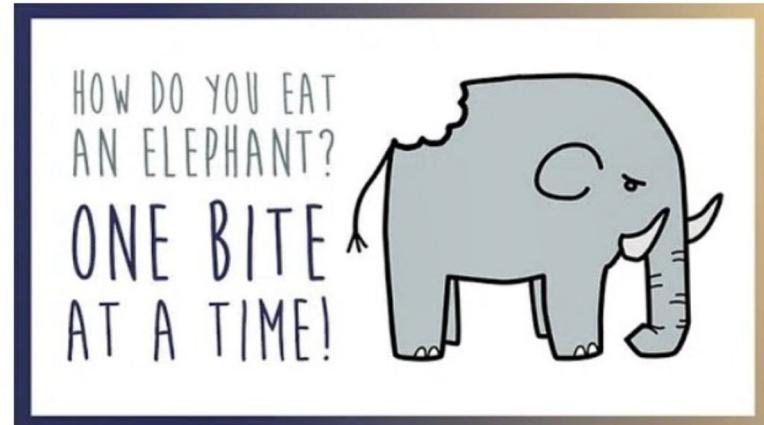
CICD

Why do we need CICD

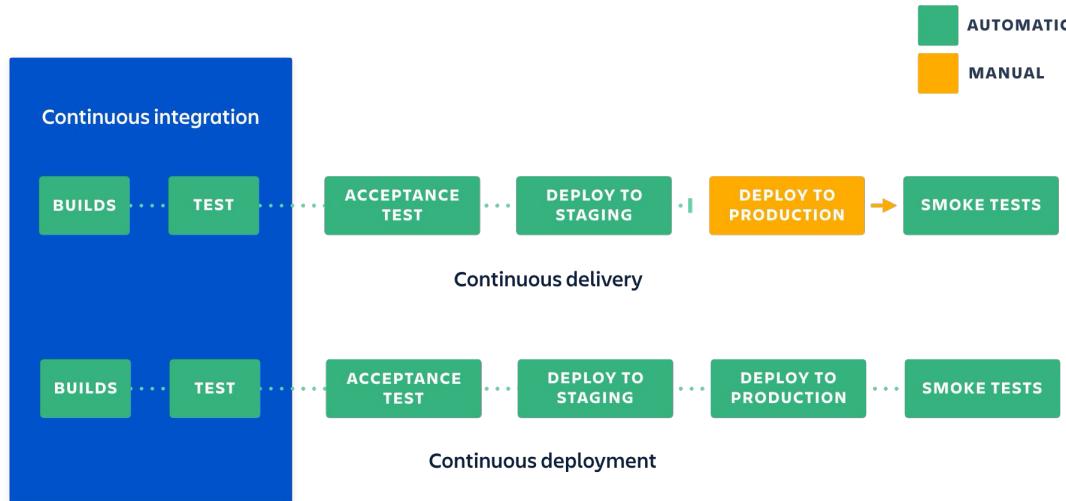
It is better to ship a small change than shipping a GIANT change.

Making frequent changes mean having to *frequently redeploy* your solution ⇒ Automate **deployment**

You want to make sure that the *system works before deploying it* ⇒ Automate **testing**

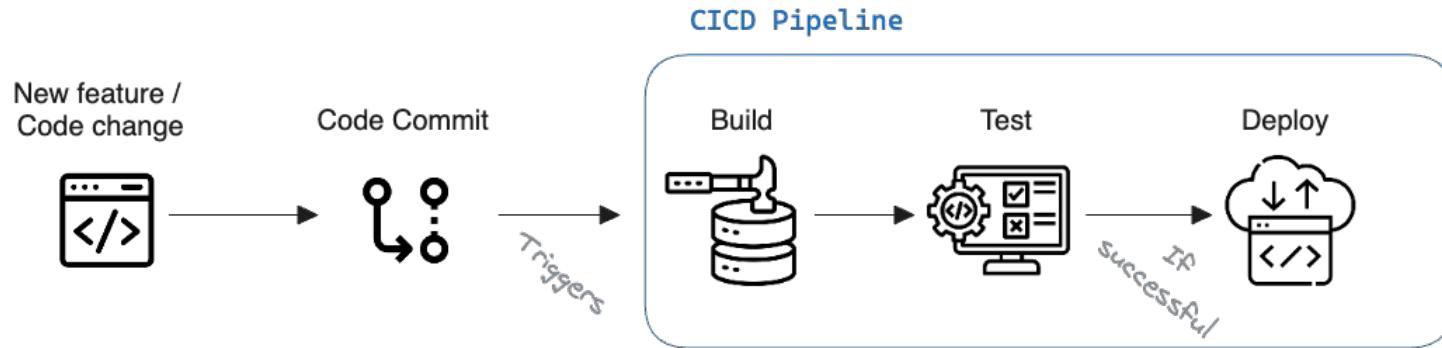


Continuous Integration (CI), Continuous Delivery (CD) & Continuous Deployment (CD)



CICD Pipeline

Allows you to continuously work on your application and efficiently deploy new changes to it.



CICD Pipeline

Integrated in code versioning tools



GitHub Actions



CI CD



Bitbucket Pipelines

CICD: Build stage

We combine the source code and its dependencies to build a runnable instance of our product that we can potentially ship to our end users.

Build the **Docker containers** we covered in a previous lecture.

Failure to pass the build stage is an indicator of a fundamental problem in a project's configuration, and it's best to address it immediately.

CICD: Test stage

(covered in next section).

CICD: Deploy stage

If our codes passed the previous steps, we can **deploy** the feature we built.

For example, if we updated the logic of an API this part of the pipeline would deploy the API in the Cloud. That could also be a ML model training pipeline.

In development we typically have resources in different **environments**.

Example CI/CD pipeline with Github Actions

```
name: Build and Test

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Set up Python Environment
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'
      - name: Install Dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run Tests
        run: |
          python manage.py test
```

```
deploy:
  needs: [test]
  runs-on: ubuntu-latest

  steps:
    - name: Checkout source code
      uses: actions/checkout@v2

    - name: Generate deployment package
      run: zip -r deploy.zip . -x '*.git*'

    - name: Deploy to EB
      uses: einaregilsson/beanstalk-deploy@v20
      with:

        // Remember the secrets we embedded? this is how we access them
        aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

        // Replace the values here with your names you submitted in one of
        // The previous sections
        application_name: django-github-actions-aws
        environment_name: django-github-actions-aws

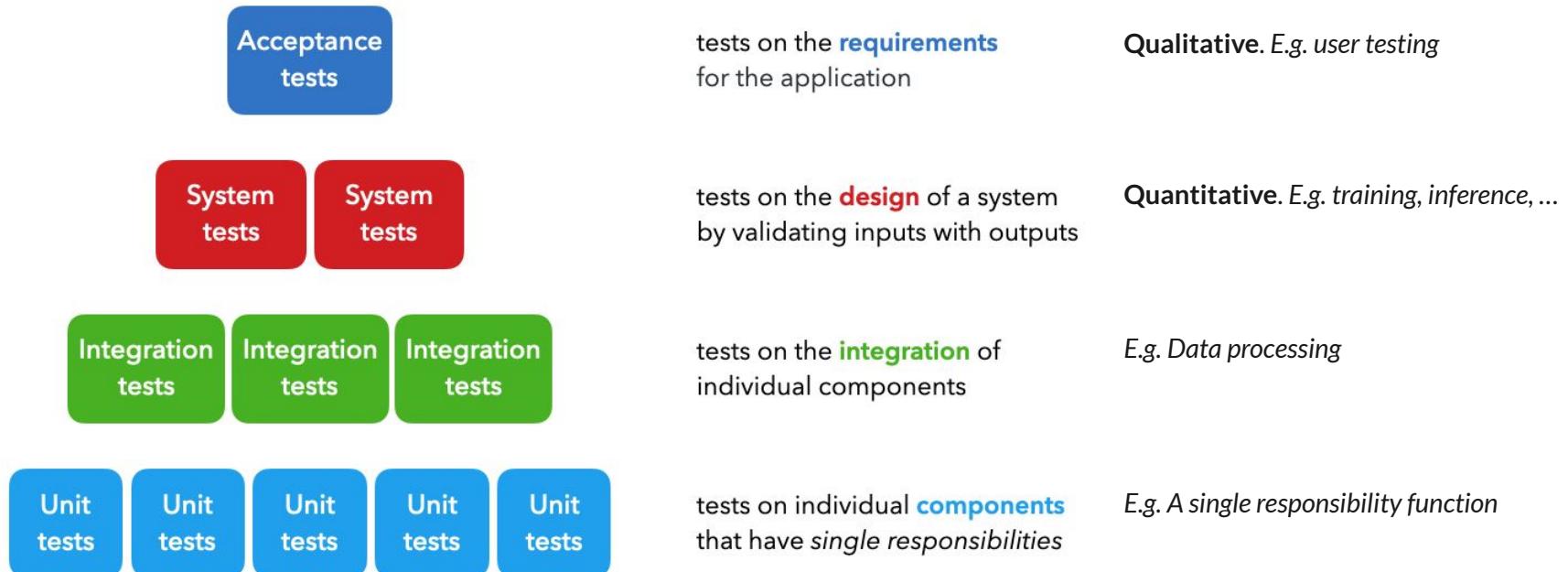
        // The version number could be anything. You can find a dynamic way
        // Of doing this.
        version_label: 12348
        region: "us-east-2"
        deployment_package: deploy.zip
```

Code testing

Why should we have automatic code tests?

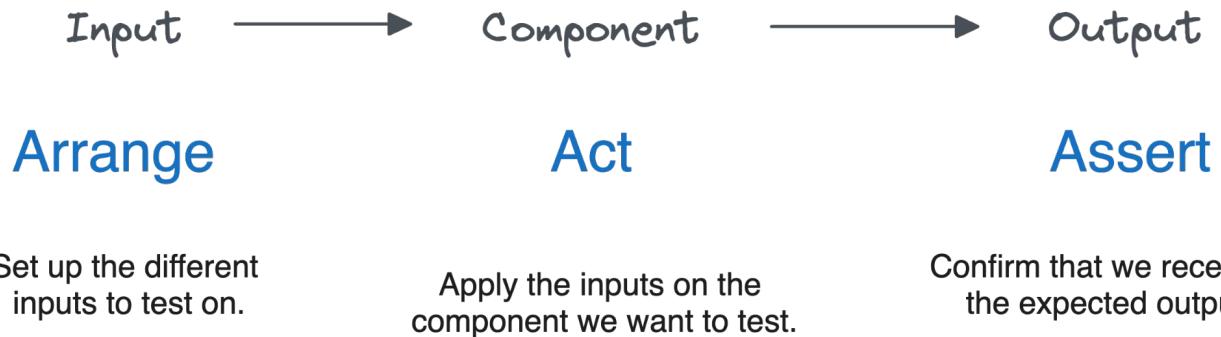
- **Catch bugs early:** Identify problems early in the development cycle, making it easier and cheaper to fix them.
- **Facilitate Refactoring:** Makes it safer to apply changes, as developers know that there is a safety net to deploying changes.
- **Improve Code Quality:** Writing tests forces developers to consider edge cases and error conditions, leading to more robust code.
- **Documentation:** Tests can serve as documentation, showing how a piece of code is intended to be used.

Different types of code testing levels



How to compose tests?

"Arrange-Act-Assert" - a pattern for arranging and formatting code in Unit Test methods





Testing best practices

- **Atomic:** Single Responsibility Principle ([SRP](#)) states that “a module (or function) should be responsible to one, and only one, actor” → Allows for *clear testing*
- **Compose:** Create tests as you implement methods! Catch errors early on and reliably.
- **Reuse:** Reuse similar tests across different projects (can maintain a single repo for tests)
- **Regression:** Test against known errors. If a new error occur → Create a new test to prevent it from happening again in the future.
- **Coverage:** We want to ensure 100% coverage for our codebase. This doesn't mean writing a test for every single line of code but rather accounting for every single line.
- **Automate:** Not only run tests manually but also as part of, for example, your CICD pipelines.

CICD: Test stage

Let's look at two **tools** to enable two **types** of testing

1. **Functionality:** Test components of your application to catch bugs
 - E.g. [Pytest](#)
2. **Linters:** Check your codes *without running it*, to flag programming errors (type errors), bugs, stylistic errors, and suspicious construct.
 - E.g [Ruff](#), pylint, black, ...

Pytest

Pytest makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries.

By default, pytest identifies files starting with `test_` or ending with `_test.py` as test files.

```
# my_math_module.py

def add(a, b):
    """Add two numbers together."""
    return a + b
```

```
# test_my_math_module.py

from my_math_module import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(-1, -1) == -2
```

Pytest

You can then run your tests by just using the `pytest` command in your root directory.

```
● → pytest git:(main) ✘ pytest
=====
platform darwin -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: /Users/thomasvrarancken/Documents/project/ulg/github/info9023-mlops/my_labs/misc/pytest
collected 1 item

test_my_math_module.py . [100%]

===== 1 passed in 0.00s =====
○ → pytest git:(main) ✘
```

Pytest

```
# test_my_math_module.py

from my_math_module import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(-1, -1) == -2
    assert add(-1, 10) == -2
```

Pytest

```
# test_my_math_module.py

from my_math_module import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(-1, -1) == -2
    assert add(-1, 10) == -2
```

```
④ → pytest git:(main) ✘ pytest
=====
platform darwin -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: /Users/thomasvrancen/Documents/project/ulg/github/info9023-mlops/my_labs/misc/pytest
collected 1 item

test_my_math_module.py F [100%]

=====
FAILURES =====
test_add -----
>     assert add(-1, 10) == -2
E     assert 9 == -2
E         + where 9 = add(-1, 10)

test_my_math_module.py:9: AssertionError
=====
short test summary info =====
FAILED test_my_math_module.py::test_add - assert 9 == -2
===== 1 failed in 0.04s =====
```

Pytest

```
● ● ●  
import pytest  
  
# Use the parametrize decorator to test a list of arguments  
@pytest.mark.parametrize('num1, num2, expected',[(3,5,8),  
(-2,-2,-4), (-1,5,4), (3,-5,-2), (0,5,5)])  
def test_sum(num1, num2, expected):  
    assert sum(num1, num2) == expected
```

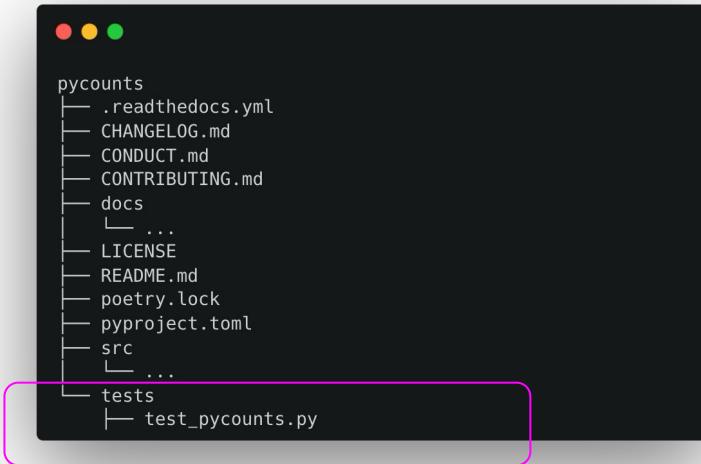
```
collected 6 items  
  
test_example.py::test_sum[3-5-8] PASSED [ 16%]  
test_example.py::test_sum[-2--2--4] PASSED [ 33%]  
test_example.py::test_sum[-1-5-4] PASSED [ 50%]  
test_example.py::test_sum[3--5--2] PASSED [ 66%]  
test_example.py::test_sum[0-5-5] PASSED [ 83%]  
test_example.py::test_sum_output_type PASSED [100%]  
  
===== 6 passed in 0.04 seconds =====
```

One can use the **parameterized** decorator to run the test on a series of parameters.

Define a set of tests to run using methods defined in your codes (e.g. API logic).

Pytest

- Tests are defined as functions prefixed with `test_` and contain one or more statements using `assert` to test against an expected result or raises a particular error
- Tests are put in files of the form `test_*.py` or `*_test.py`
- Tests are usually placed in a directory called `tests/` in a package's root.



```
pycounts
├── .readthedocs.yml
├── CHANGELOG.md
├── CONDUCT.md
├── CONTRIBUTING.md
├── docs
│   └── ...
├── LICENSE
├── README.md
├── poetry.lock
├── pyproject.toml
└── src
    └── ...
        └── tests
            └── test_pycounts.py
```

Ruff

Linters are **static code analysers**

⇒ It analyses your code without actually running it. It checks for errors, enforces a coding standard, looks for code smells, and can make suggestions about how the code could be refactored.

Code convention: Python PEP8

- Ensures a consistent code quality across a team.
- Set of rules on styling, indenting, naming conventions and documentation
- Integrate/automate in your code editor (IDE) to make it easy.
Often enforced PEP8 during pull request submission.
- 🧑 Developers can be judgy... Make your life easy, adapt clean codes

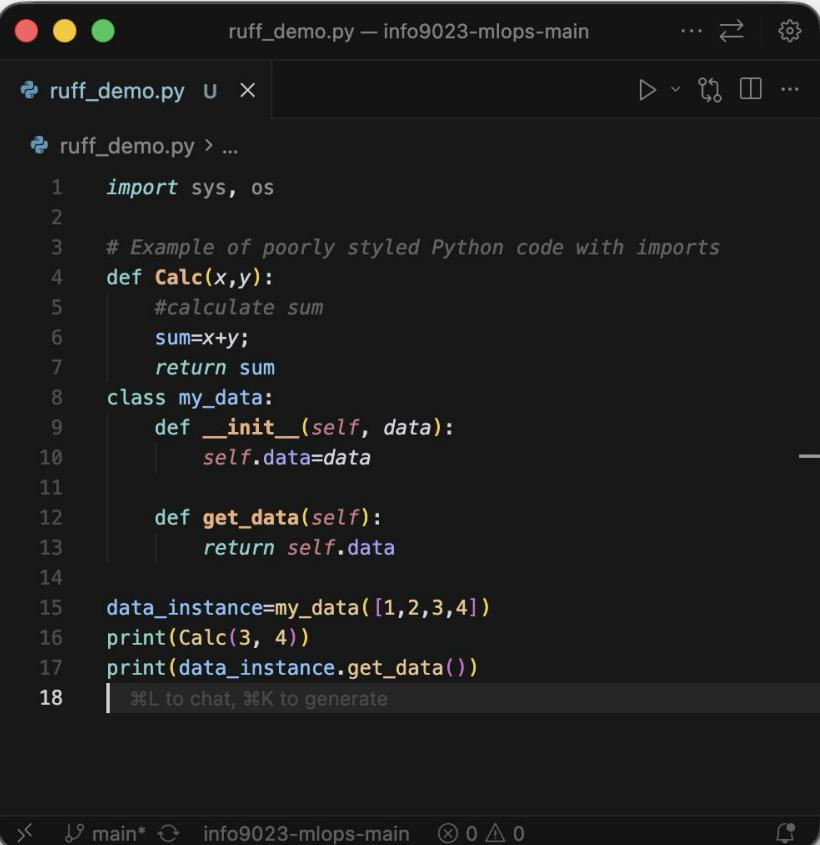
*Tools to automatically check compliance with
PEP8 will be covered in the CICD lecture*

<https://peps.python.org/pep-0008/>



Ruff

Example



A screenshot of a terminal window titled "ruff_demo.py – info9023-mlops-main". The window shows a Python script with various annotations. Annotations include:

- Line 1: A note above the import statement: "# Example of poorly styled Python code with imports".
- Line 4: A note above the function definition: "def Calc(x,y):".
- Line 5: A note above the sum assignment: "#calculate sum".
- Line 6: A note above the sum assignment: "sum=x+y;".
- Line 7: A note above the return statement: "return sum".
- Line 8: A note above the class definition: "class my_data:".
- Line 9: A note above the __init__ method: "def __init__(self, data):".
- Line 10: A note above the self.data assignment: "self.data=data".
- Line 12: A note above the get_data method: "def get_data(self):".
- Line 13: A note above the return statement: "return self.data".
- Line 15: A note above the data_instance assignment: "data_instance=my_data([1,2,3,4])".
- Line 16: A note above the print statement: "print(Calc(3, 4))".
- Line 17: A note above the final print statement: "print(data_instance.get_data())".
- Line 18: A note above the prompt: "%L to chat, %K to generate".

```
1 import sys, os
2
3 # Example of poorly styled Python code with imports
4 def Calc(x,y):
5     #calculate sum
6     sum=x+y;
7     return sum
8 class my_data:
9     def __init__(self, data):
10        self.data=data
11
12     def get_data(self):
13         return self.data
14
15     data_instance=my_data([1,2,3,4])
16     print(Calc(3, 4))
17     print(data_instance.get_data())
18 | %L to chat, %K to generate
```

Ruff

Example

```
info9023-mlops-main — thomasvrancen@Thomass-MacBook-Pro — ..23-mlops-main --zsh — 113x30
[→ info9023-mlops-main git:(main) ✘ ruff check ruff_demo.py --select=ALL --output-format=concise
warning: `incorrect-blank-line-before-class` (D203) and `no-blank-line-before-class` (D211) are incompatible. Ignoring `incorrect-blank-line-before-class`.
warning: `multi-line-summary-first-line` (D212) and `multi-line-summary-second-line` (D213) are incompatible. Ignoring `multi-line-summary-second-line`.
ruff_demo.py:1::1: E401 [*] Multiple imports on one line
ruff_demo.py:1::1: D100 Missing docstring in public module
ruff_demo.py:1::1: I001 [*] Import block is un-sorted or un-formatted
ruff_demo.py:1::8: F401 [*] 'sys' imported but unused
ruff_demo.py:1::13: F401 [*] 'os' imported but unused
ruff_demo.py:4::5: N802 Function name 'Calc' should be lowercase
ruff_demo.py:4::5: ANN001 Missing return type annotation for public function 'Calc'
ruff_demo.py:4::5: D103 Missing docstring in public function
ruff_demo.py:4::10: ANN001 Missing type annotation for function argument `x`
ruff_demo.py:4::12: ANN001 Missing type annotation for function argument `y`
ruff_demo.py:6::5: A001 Variable 'sum' is shadowing a Python builtin
ruff_demo.py:6::12: E703 [*] Statement ends with an unnecessary semicolon
ruff_demo.py:7::12: RET504 Unnecessary assignment to 'sum' before `return` statement
ruff_demo.py:8::7: N801 Class name 'my_data' should use CapWords convention
ruff_demo.py:8::7: D101 Missing docstring in public class
ruff_demo.py:9::9: ANN204 Missing return type annotation for special method `__init__`
ruff_demo.py:9::9: D107 Missing docstring in `__init__`
ruff_demo.py:9::24: ANN001 Missing type annotation for function argument `data`
ruff_demo.py:12::9: ANN201 Missing return type annotation for public function `get_data`
ruff_demo.py:12::9: D102 Missing docstring in public method
ruff_demo.py:16::1: T201 'print' found
ruff_demo.py:17::1: T201 'print' found
Found 22 errors.
[*] 5 fixable with the `--fix` option (4 hidden fixes can be enabled with the `--unsafe-fixes` option).
→ info9023-mlops-main git:(main) ✘
```

The screenshot shows a terminal window on the left and a code editor window on the right. The terminal window displays the command-line output of the Ruff code analysis, listing 22 errors and 5 fixable issues. The code editor window shows the Python file ruff_demo.py with syntax highlighting and line numbers. The code itself contains several style violations, such as multiple imports on one line, missing docstrings, and non-CapWords class names.

```
ruff_demo.py — info9023-mlops-main
ruff_demo.py > ...
import sys, os
# Example of poorly styled Python code with imports
def Calc(x,y):
    #calculate sum
    sum=x+y;
    return sum
class my_data:
    def __init__(self, data):
        self.data=data
    def get_data(self):
        return self.data
data_instance=my_data([1,2,3,4])
print(Calc(3, 4))
print(data_instance.get_data())
#L to chat, #K to generate
```

Best practice: Local pre-commit

pre-commit package.

```
(venv) → madewithml git:(dev) ✘ git add .
(venv) → madewithml git:(dev) ✘ git commit -m "added pre-commit hooks"
trim trailing whitespace.....Passed
fix end of files.....Passed
check for merge conflicts.....Passed
check yaml.....Passed
check for added large files.....Passed
check yaml.....Passed
clean.....Passed
```

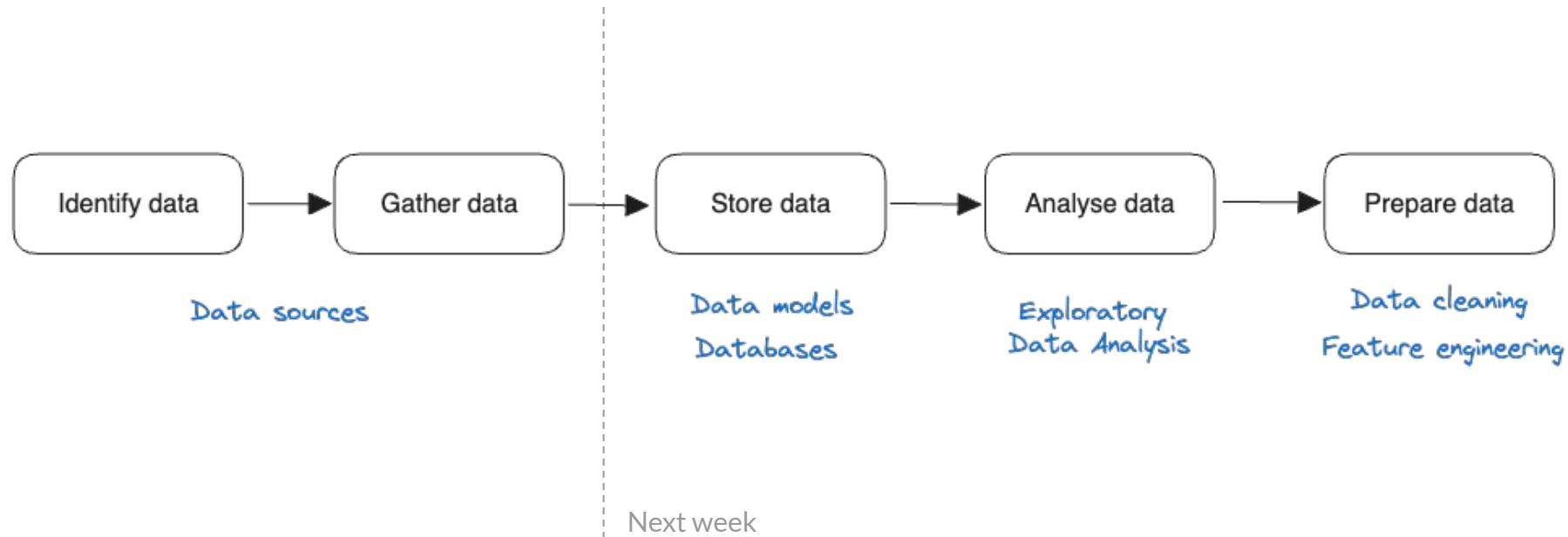


When should you use testing?

Is it more important when using **Gitflow** or **trunk based** code versioning?

Data sources

Overall data process



Data source

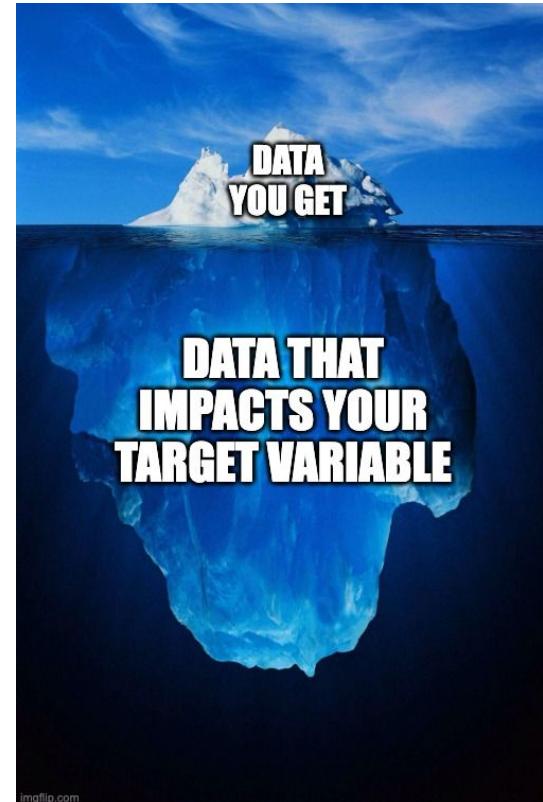
Really important to make sure you understand **all factors** that could be impacting your system.

Spend time discussing with **process experts** and challenge them on what input your model should use.

Example:

- Demand forecasting

What data would you need to predict demand?



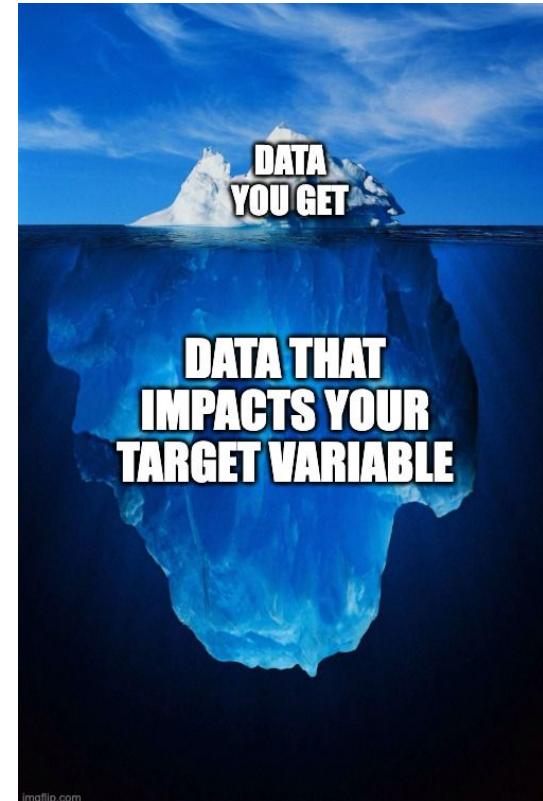
Data source

Really important to make sure you understand **all factors** that could be impacting your system.

Spend time discussing with **process experts** and challenge them on what input your model should use.

Example:

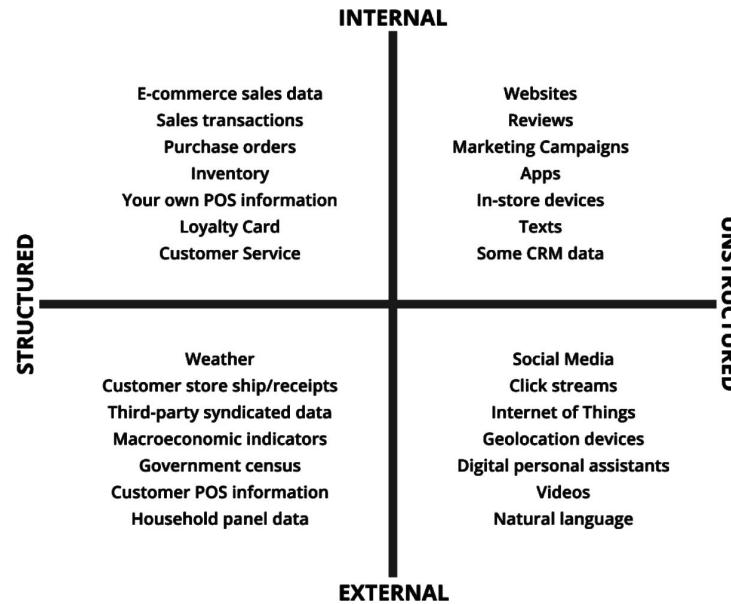
- Demand forecasting
 - It will vary *a lot* depending on industry
 - Fashion → *trends*
 - Ice cream → *weather*
 - Stocks → *macroeconomics*
 - ...
 - Ask sales people how they estimate demand.



Data source

Internal data already belong to the organisation building the model.

Split in two dimensions



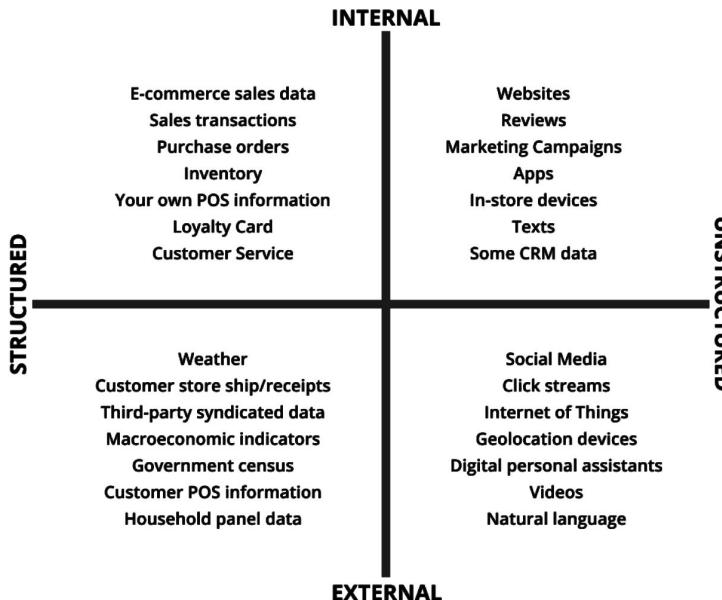
External data needs to be gathered, scraped or purchased externally.

Data source

Split in two dimensions

Structured data

- Usually tabular organised data
- Schema clearly defined
- Easy to search and analyse
- Schema changes will cause a lot of troubles
- Expensive to setup and store
- Stored in a data warehouse



Unstructured data

- Data doesn't have to follow a schema
- All non-tabular data (computer vision, NLP, audio, video, ...)
- Fast arrival
- Cheap to store large volumes
- Can handle data from any source
- Harder to extract information
- Stored in a data lake



Labeled data

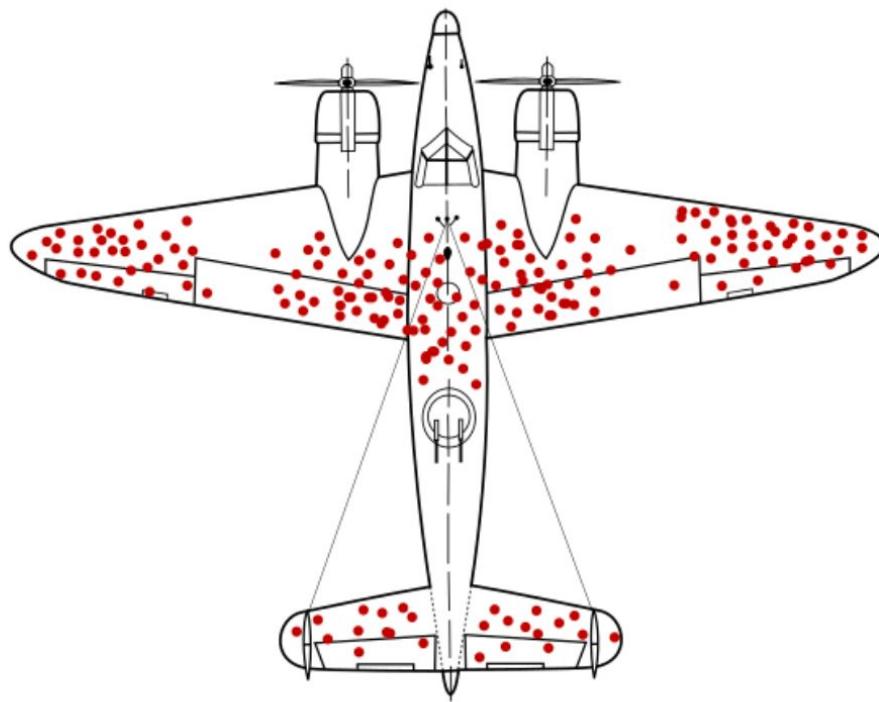
Labeled data contains observations with a value for your **target variable**.

You need

- Enough labeled data
- Good quality labels
- Truly representing what you are trying to model.

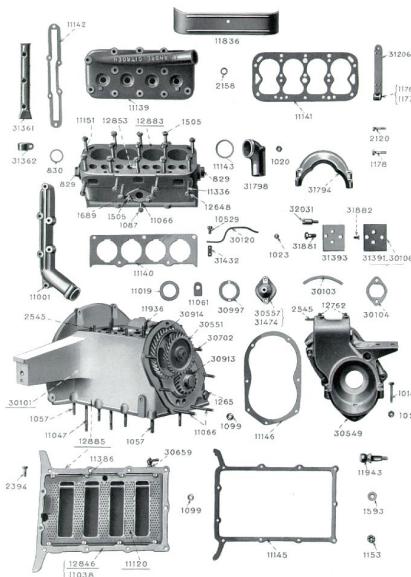
Representative data is hard to gather...

Careful to bias when collecting data



Setting up a data collection and/or labelling system

Example 1: Building a physical system



Context: Company selling hundreds of different *engine parts*.

Goal: Automatically classify physical parts using computer vision.

How would you approach it?

Setting up a data collection and/or labelling system

Example 1: Building a physical system



Context: Company selling hundreds of different *engine parts*.

Goal: Automatically classify physical parts using computer vision.

Approach:

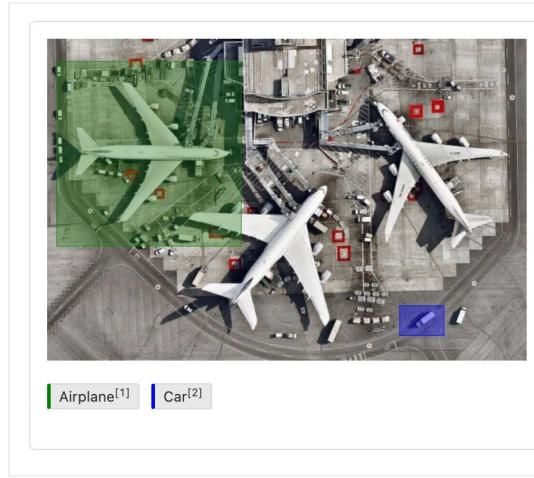
1. Build a box with clear lighting and multiple cameras
2. Build a UI to easily take a snap picture and enter product code
3. Each new and returning products had to go through it for a summer

Result:

>100k product images.

Setting up a data collection and/or labelling system

Example 2: Build a labeling tool and hire a 3rd party company (or student workers)



Airplane^[1] Car^[2]

Results ✖

Nothing selected

Regions (2) ! Sort

1 Rectangle 155.00 x 155.00 ✖

2 Rectangle 37.00 x 25.00 ✖

Relations (0)

No Relations added yet

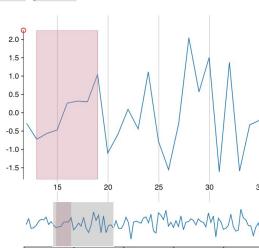
To have faith is to trust yourself to the water

Positive^[1]
 Negative^[2]
 Neutral^[3]

Time Series classification

Growth^[1]
 Decay^[2]

Run^[3] Walk^[4]



Results ✖

Nothing selected

Regions (1) ! Sort

1 TS 13 - 19 ✖

Relations (0)

No Relations added yet



Label Studio

Quality challenge in manual data labelling

Beware of bias and data quality when collecting the data! Surprisingly hard to get a lot of people to label in a consistent way - inherent bias...

Task: Classify the following online comment as “toxic” or “not toxic.”

Comment: “1. People still eat at Pizza Hut? Gross. 2. It is shameful how this country[...].”

Toxic Not toxic

The diagram shows a grid of 24 people (3 rows by 8 columns) reacting to a single online comment. The comment is: "1. People still eat at Pizza Hut? Gross. 2. It is shameful how this country[...]".

Annotations from the top row (4 people per row):

- THIS IS NON-CONSEQUENTIAL. NON-TOXIC.
- WHO IS THIS PERSON?
- TOXIC: THEY'RE HEAPING SHAME ON THE WHOLE COUNTRY AT ONCE.
- THIS SEEMS TOXIC IN ONLY CERTAIN CONTEXTS...

Annotations from the middle row (4 people per row):

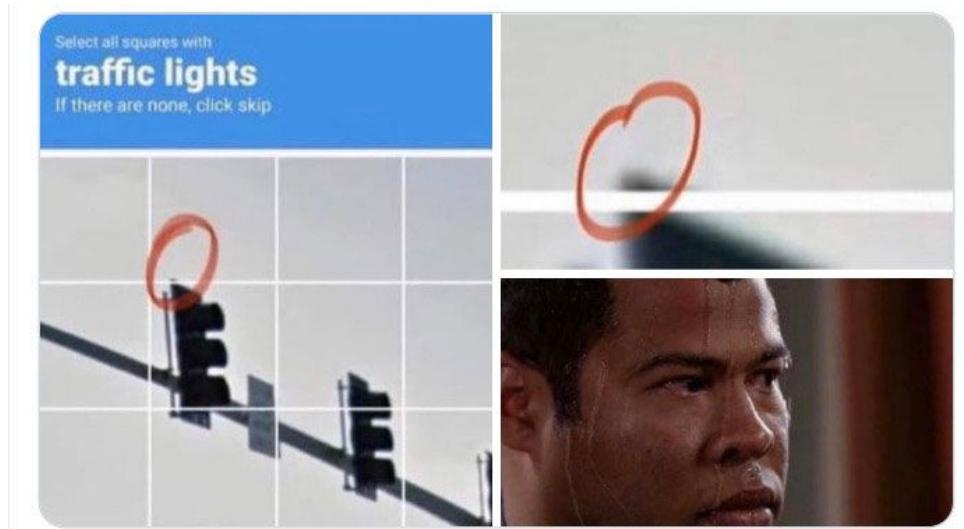
- EATING AT PIZZA HUT MAY BE GROSS TO SOME, BUT WHAT'S REALLY GROSS IS THE GROSS GENERALIZATION. TOXIC.
- TOXICITY IS DEFINED DIFFERENTLY FOR DIFFERENT SOCIAL MEDIA COMPANIES!
- NON-TOXIC. IT'S MERELY AN OPINION.

Annotations from the bottom row (4 people per row):

- NON-TOXIC, BUT INSULTING. NO LINE CROSSED HERE.
- TOXIC. I LOVE PIZZA HUT.
- TELL ME WHERE IT WAS POSTED AND I'LL TELL YOU IF IT'S TOXIC.
- I GOT FOOD POISONING FROM THE CORNER DELI. NOW THAT WAS TOXIC!

Quality challenge in manual data labelling

Beware of bias and data quality when collecting the data! Surprisingly hard to get a lot of people to label in a consistent way - inherent bias...



Quality challenge in manual data labelling

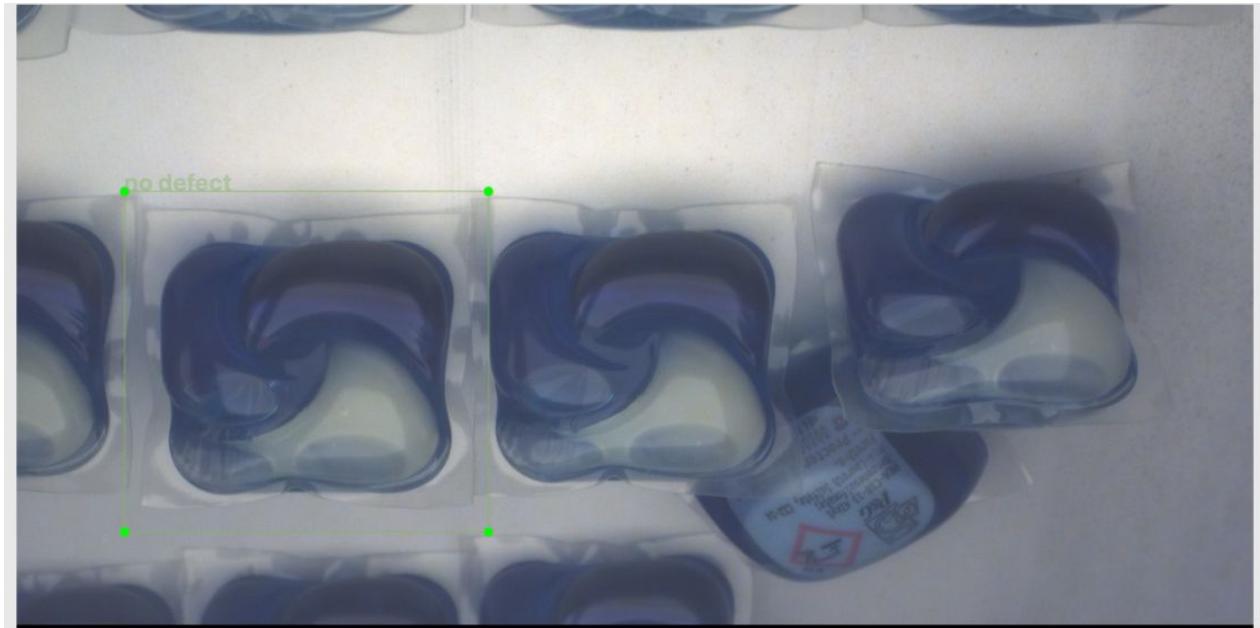
Beware of bias and data quality when collecting the data! Surprisingly hard to get a lot of people to label in a consistent way - inherent bias...

Solutions:

- **Set strict guidelines**
 - Explain in detail what should happen
 - Give a lot of examples
- **Review sessions** of difficult examples with different labelers
- **Overlap samples** which are then labeled by multiple labelers
 - Remove or average observations with different labels
- **Calculate bias**
 - Standard deviation of labels
 - Distribution



Be really careful with your labeling strategy as it's hard to come back from it...



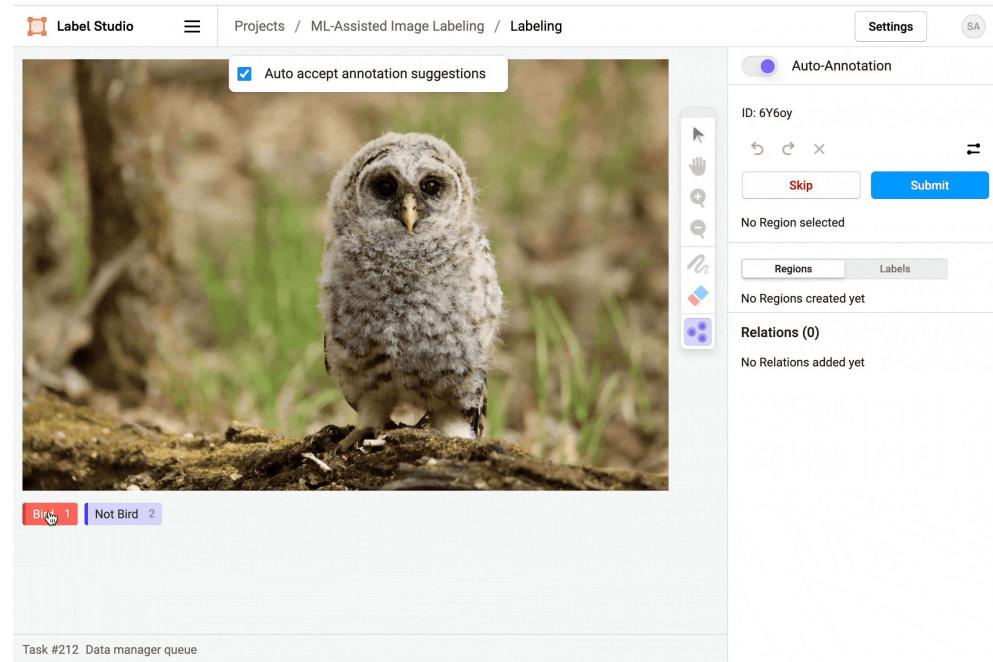
Detecting defect in dishwasher tab pouches.

Did not label overlapping pouches...

Data pseudo-labelling

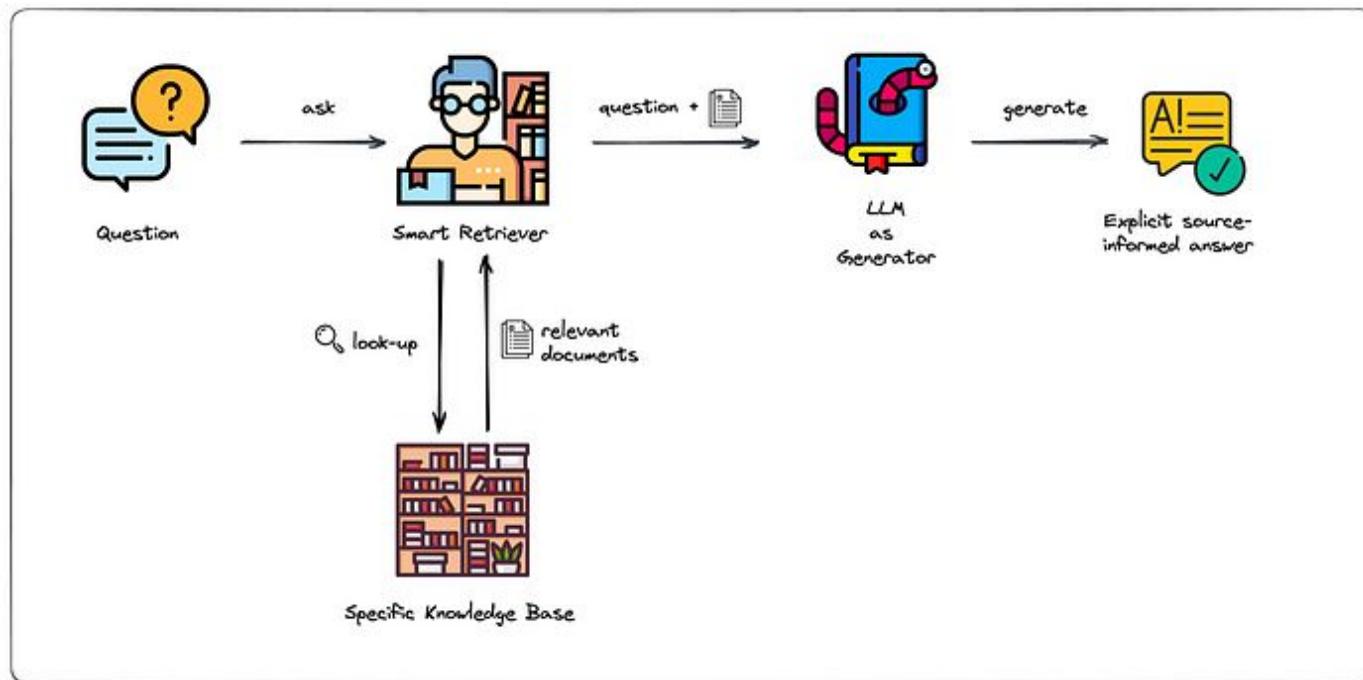
Use an unsupervised general model to produce some possible labels and get them verify.

Can lead to overfitting on general model !



Retrieval Augmented Generation (RAG)

Side note



Retrieval Augmented Generation (RAG)

Side note

 Google Cloud
3,141,475 followers
3d • 

We are excited to highlight how [Mediahuis Nederland](#) partnered with Google Cloud to build NewsSearch.

Researching a news story used to be a bottleneck, taking up to 30 minutes just to find background information in the archives. Today, thanks to generative AI, that process happens in seconds.

Powered by Vertex AI and Gemini models, NewsSearch goes beyond simple keyword matching. It acts as an intelligent assistant that transforms vast archives into a dynamic journalistic asset.

As Bart Janssens, Senior Project Lead at Mediahuis, puts it: "Innovation and journalistic rigor can not only coexist but also reinforce each other. With AI as a companion, our journalists are opening a new era in the press."

Read the full case study, here → <https://goo.gle/3M552MF>

 **MEDIAHUIS**

Mediahuis strengthens its commitment to

 You
Give me 5 questions about solar panels

 NewsSearch
Here are five questions about solar panels, based on recent articles in De Telegraaf:

1. What is the current payback period for solar panels, taking into account the abolition of the net metering scheme in 2027 and any feed-in tariffs from energy suppliers? [[11](#), [14](#)]
2. Are my roof and location suitable for solar panels, and what is the minimum number of panels I would need to cover a significant portion of my energy consumption? [[2](#), [3](#), [5](#)]
3. What factors should I consider when choosing a reliable solar panel supplier and comparing quotes? [[2](#), [3](#), [5](#), [12](#)]
4. Which type of inverter is most suitable for my situation, and should I consider purchasing a home battery to store the generated energy? [[2](#), [5](#), [7](#), [12](#), [22](#)]

Searching through the archive... • Last updated on Nov 17, 2024, 6:49:49 PM 

[Most relevance] [Most recent]

[1] • the telegraph • Jun 27, 2016
Energy consultation: many questions about solar panels
Do I get a subsidy if I buy solar panels? And how long will it take before I earn them back? Will I really be better off if I switch energy suppliers and if so, how much?


[2] • the telegraph • Jul 2, 2022
7 questions: Reduce your alarmingly high energy bill by purchasing solar panels
Many homeowners want to reduce their shockingly high energy bills by purchasing solar panels. With the current high electricity prices, an investment of a few thousand euros will be earned back within about five years. But what should you pay attention to when purchasing solar panels? Seven questions.

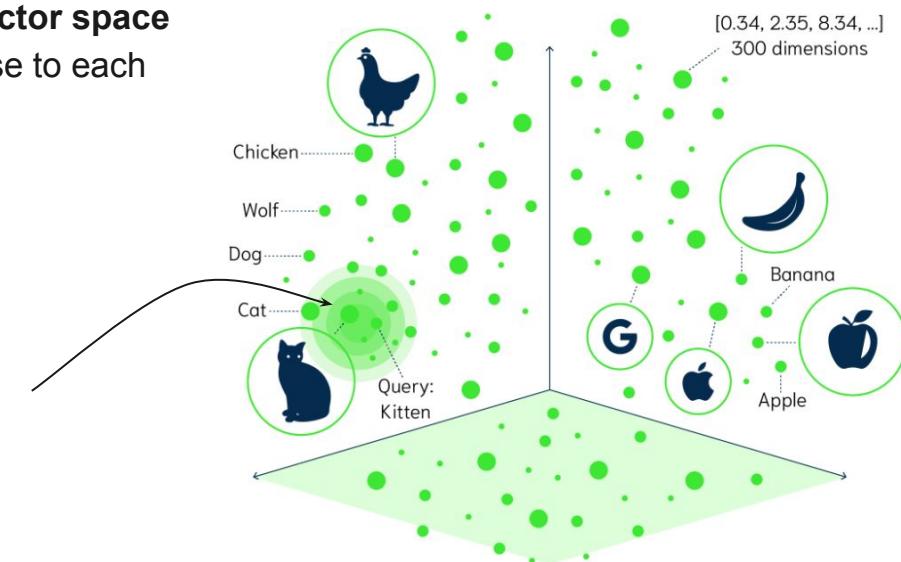
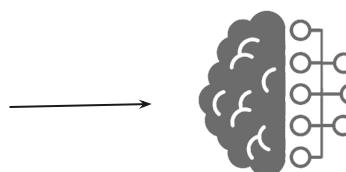

[3] • the telegraph • May 24, 2013
Five important questions about purchasing solar panels
Is the Dutch climate suitable and when will the costs of the panels be recouped? OverGeld answers the four most important questions about solar panels.

[4] • the telegraph • Jun 2, 2015
Many questions about solar panels during energy consultation
The annual energy consultation hour of De Telegraaf and DFTgeld.nl received many questions about solar panels on Monday. "How many panels should you buy? And what about the payback period?" many readers wanted to know from the energy experts from Gaslicht.com, Vereniging Eigen Huis and ACM/ConsuWijzer.nl who were present at the consultation hour.

Breakout exercise

Semantic Search works with **text embedding**. An ML model is trained to represent any text in a **vector space** where texts tackling similar topics will be close to each others.

Query:
“How often should I
feed my cat?”

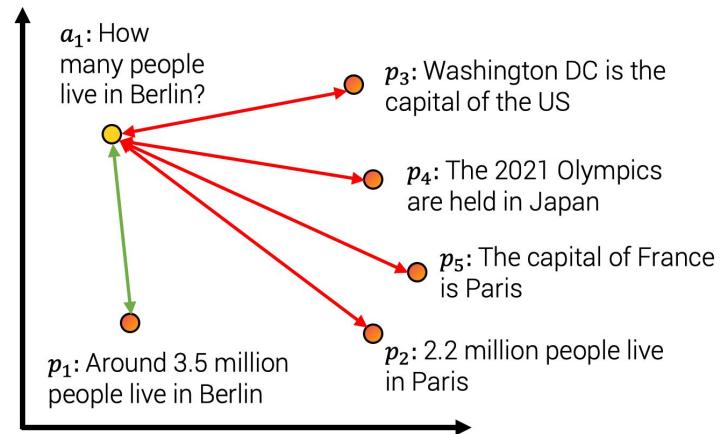


Breakout exercise

You can find **pre-trained** embedding models (OpenAI, Huggingface, Cohere, ...).

You can **fine-tune** these models on your own data to make them better at understanding **domain specific language**.

What you need: A data set of **Question-Answer pairs**.



Breakout exercise

Exercise:

- Team of 2-3 students
- You got 5min

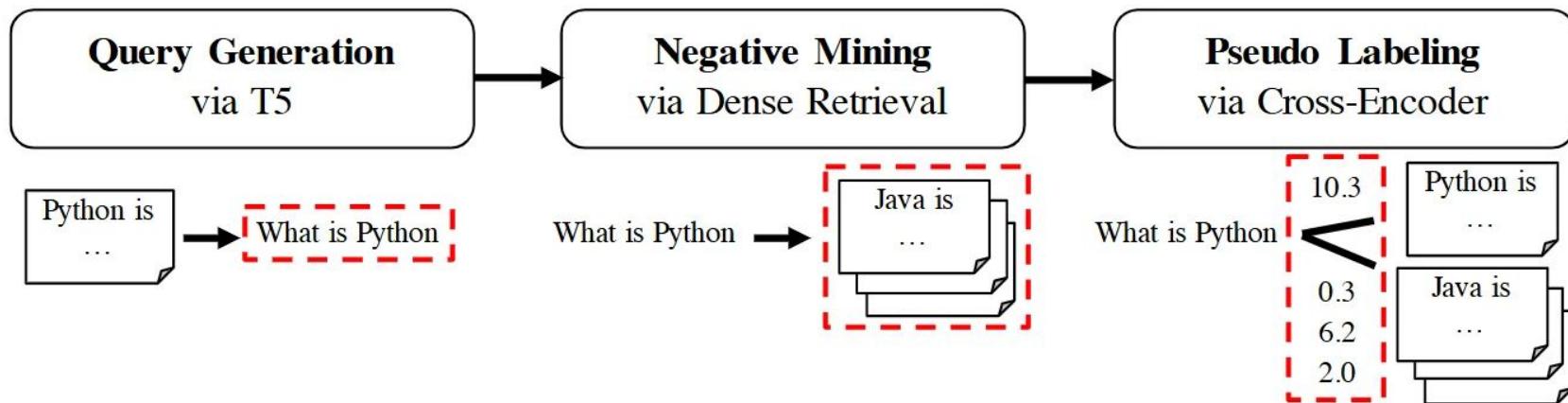
Context:

- Assume you work for a **publishing company** that has **300k news articles** from the last 30 years
- They want to build a **semantic search engine**
- The documents use domain specific language so you want to **fine-tune a text embedding model**
- For that, you need a labeled data set of **Question and Answer pairs**
 - We want it to specifically use the **language** and **domain** of the newspaper publisher
 - The publisher only has documents

Goal:

- Try to come up with a couple of strategies to **create a labeled dataset**

Generative Pseudo Labeling



Wrap-up

Lecture summary

| Topic | Concepts | To know for... | |
|------------------------------------|---|----------------|------|
| | | Project | Exam |
| Git: code versioning & conventions | <ul style="list-style-type: none">• Git Flow vs Trunk based• Git conventions• Python convention (PEP8) | Yes | |
| CICD | <ul style="list-style-type: none">• Continuous Integration (CI), Continuous Delivery (CD) and Continuous Deployment (CD) | Yes | Yes |
| Code testing | <ul style="list-style-type: none">• Levels of testing: Unit, Integration, System and Acceptance• Functionality testing (Pylint)• Linters (ruff) | Yes | Yes |
| Data sources | <ul style="list-style-type: none">• Internal vs external• Structured vs unstructured• Data labelling techniques | | Yes |
| DW: Git & CICD | | Yes | |

Project objective for sprint 1



| # | Week | Work package |
|-----|------|---|
| 1.1 | W01 | <p>Pick a team</p> <ul style="list-style-type: none">• Try to mix skills and experience• If you didn't find one let one of the teachers know and we'll allocate you to one |
| 1.2 | W01 | <p>Select a use case</p> <p>Source options</p> <ul style="list-style-type: none">• Previous course• Kaggle Datasets• ... <p>Make sure to pick a use case where data is available. A requirement for this project is to train and deploy your own ML model. It is not an option to work with LLMs, as there is too much overhead in training and serving those. Note that the ML modeling itself won't be a big part of the course. You can pick data from one of your previous projects if it is available. Ideally pick something with interesting data and a real world application.</p> |
| 1.3 | W01 | Find a cool name for your project ✨ |
| 1.4 | W01 | <p>Fill in the project card template (docx version is in this course's github repository, under "project/project_card_template.docx") and send it via email to the teaching staff.</p> <p>The teaching staff needs to approve and provide feedback on your project - if the project is deemed not feasible then we will ask you to come up with a new one.</p> |

Project card templates

We will provide you with some feedback by Today EOD !

[Project name]

INFO9023 - Project card

Purpose of this document is to briefly explain what your project will be about so the teaching staff can validate it and provide feedback. It is not part of grading, no need to spend too much time editing it (just a couple of sentences per section).

Make a copy and send it to t.vrancken@uliege.be and Matthias.Pirlet@uliege.be by the 26/02/2024.

Project description

[Short description of your project]

Project data

[Short description of the data you'll use]

Project objective for sprint 1

| # | Week | Work package |
|-----|------|--|
| 1.6 | W02 | <p>Build a simple CICD pipeline using Github Actions with the following:</p> <ol style="list-style-type: none">1. Uses pre-commit hooks2. Code formatting checks (your choice of tools from pylint, ruff, black or something else)3. A placeholder step to run pytests that you might implement in the future <p>If you have any questions on the steps to follow, feel free to ask the teaching staff.</p> |

Next week

- Guest lecture on **cloud infrastructure** !
- First DW on **Google Cloud** !



Project deliverables

Deliverables

- **3 Milestone (MS) meetings:**
 - MS 1: Present work from sprint 1 & 2
 - Present your general use case (BMC), the data preparation and the result from your model experimentation.
 - MS 2: Present work from sprint 3 & 4
 - Present your architecture for model deployment and automated training.
 - MS 3: Overall presentation
 - Whole project, inc. sprint 5
- Each MS presentation will be accompanied with a *code submission on Github*. Teaching staff will not go over all codes but key information can be provided in README + check that everything is well implemented.

No handovers before MS meetings. The work packages per sprints are there to guide you, but you're free to implement it at your pace. Teaching staff is there to provide support every week.

Directed work:

Git & CI/CD

MADE IT THROUGH THE LECTURE



SEE YOU NEXT WEEK !

imgflip.com