

Python for Data Science

Session #2

Outline

1. Python environment
 - a. Pyenv
 - b. Conda
 - c. PyPI
2. Version control system
 - a. Git
 - b. GitHub, GitLab, Bitbucket
 - c. Github playground
 - d. Data Version Control DVC
3. Integrated Development Environment
 - a. VSCode
4. Exercises

Assignment 1: Tips & Comments

- GitHub Repositories missing... (**56 out of 63**)
- README files
 - **Clear** and **understandable**
- Exercices without statement
- The most straightforward solution isn't always the **best** (even if it works!)
- Use **text blocks** to introduce the exercices (in Jupyter)
- Solutions Assignment 1 [here](#)

Python environment

Every single project is built on a specific version of Python. Same happens with the libraries.

Python 3.12.5

rasterio 1.3.11

setuptools 75.1.0

notebook 7.2.2

opencv-python 4.10.0.84

pillow 10.4.0

Python environment

But, why don't I simply use a recent version of Python and libraries for all my projects?

ImportError

```
Traceback (most recent call last):
  File "script.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

AttributeError

```
Traceback (most recent call last):
  File "script.py", line 10, in <module>
    result = pandas.DataFrame.from_csv('data.csv')
AttributeError: type object 'DataFrame' has no attribute 'from_csv'
```

SyntaxError

```
File "script.py", line 5
  print(f'Python {version} detected')
                                     ^
SyntaxError: invalid syntax
```

TypeError

```
Traceback (most recent call last):
  File "script.py", line 15, in <module>
    result = math.pow(2)
TypeError: pow expected 2 arguments, got 1
```



Python environment

In order to navigate between different versions of Python and libraries, we require a tool that allows us to easily switch between them.



+ virtualenv



Pyenv + Virtualenv

What each of them does:

Feature	pyenv	virtualenv
Purpose	Manage multiple Python versions	Create isolated environments for dependencies
Python Version Management	YES	NO
Virtual Environment Creation	NO	YES
Scope	System-wide or per-project version	Project-specific dependency isolation
Works With	Entire Python interpreter	A specific Python version (but isolated)

Pyenv + Virtualenv

Grab your laptop and let's install all we need to create our first project !

1. Install pyenv (brew or apt)
2. Install virtualenv plugin

```
git clone https://github.com/pyenv/pyenv-virtualenv.git $(pyenv root)/plugins/pyenv-virtualenv
```

3. Set up system (bashrc or zshrc)

```
export PATH="$HOME/.pyenv/bin:$PATH"  
eval "$(pyenv init --path)"  
eval "$(pyenv init -)"  
eval "$(pyenv virtualenv-init -)"
```

4. `pyenv install <version>` + `pyenv virtualenv <version> myenv` + `pyenv activate myenv`

Pyenv + Virtualenv

So far you learnt:

- Install your tools
- Use pyenv and virtualenv to create a new environment

Conda

Conda is a powerful command line tool for package and environment management (Windows, macOS, and Linux). For each environment you can define the Python version.

There exist three different conda installers:

- **Anaconda** (commercial use requires a paid license)
 - Comes with a ton of pre-installed libraries for data science, Anaconda navigator and a GUI
- **Miniconda** (free for any use, BSD license)
 - Comes with just conda and its minimal dependencies
- **Miniforge** (free for any use, BSD license)
 - Comes preconfigured for use with the conda-forge channel (this package is built by the community behind conda-forge).

Conda

Grab again your laptop, and let's install miniconda !

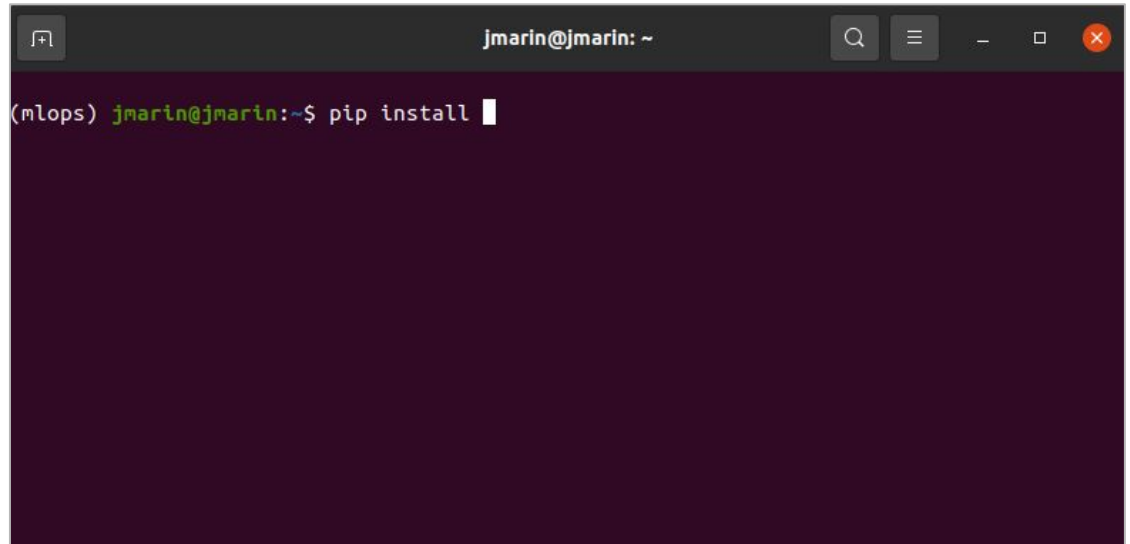
Conda

So far you learnt:

- Install your tools
- Create a new environment with Python and basic dependencies
- Install from conda and use a channel

Pypi

Pypi is of the largest and most used package indices in the Python community, counting with **572,339** projects. With just a simple command line you can install any of the libraries available.

A screenshot of a terminal window. The window title bar shows "jmarin@jmarin: ~". The terminal has a dark purple background. The prompt is "(mlops) jmarin@jmarin:~\$". The command "pip install" has been entered, followed by a cursor. The window includes standard Linux window controls (search, menu, zoom, close) in the top right corner.

Pypi

Let's create a **requirements.txt** file in our first repository. Let's grab the laptop !

Pypi

So far you learnt:

- Check that you have pip in your environment and install it (ensure pip)
- Install and uninstall a library
- Create a requirements file

Version Control System

Any decent software project requires a control on how the code evolves. Otherwise, there's no track on what was changed, who changed it, what was the purpose, or how we can go back to a working version when something breaks.

In 2005, Linus Torvalds created an open-source distributed version control system called **Git**.



According to the 2023 Stack Overflow Developer Survey, **90%** of professional developers reported using Git, making it the dominant version control system.

Version Control System

Github platform was created back in 2008, with a user-friendly interface that quickly gained a lot of popularity in the community (acquired by Microsoft in 2018).

100+ million

Developers

4+ million

Organizations

420+ million

Repositories



Version Control System

GitLab and **BitBucket** platforms are an alternative to Github. Although the later offers enterprise-oriented features (Github Enterprise), Gitlab and BitBucket are the usual choices for the private sector, focusing on team collaboration, DevOps, and project management.



Github is the usual choice for open-source projects and individual developers.

Version Control System

So far, you have already created an account in **Github**, a first project and pushed some code,



Let's see how the process should have looked like...

Version Control System

Install Git first in your laptop/PC/server

Go to <https://git-scm.com/> and proceed as described (check what OS you are using)

Once installed, setup Git config

```
git config --global user.name "Your Name"  
git config --global user.email "your-email@example.com"
```

Let's create a project now !

Version Control System

Let's create a repository (alternatively you can use github-cli)



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner * Repository name *

Great repository names are short and memorable. Need inspiration? How about [miniature-octo-carnival](#) ?

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

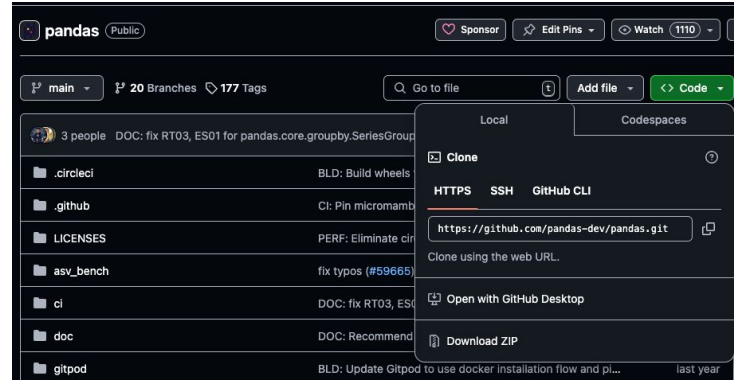
☐ You are creating a public repository in your personal account.

Create repository

Version Control System

To work locally with yours or any other project you will have the following options:

1. Clone it via https
2. Clone it via ssh (with a SSH Key)
3. Clone it using Github CLI
4. Or Download it as a zip



Version Control System

How do we do our first push?

1. Get into the repository and write down or modify any existing file(s) (use `git status`)
2. Add them like this:
`git add .` or simply `git add <file>.py`
3. Create a commit
`git commit -m "Describe the content or changes"`
4. Push them to your remote repository
`git push origin main`

Version Control System

How to download the latest version of yours or any other remote repository:

git fetch : It downloads **changes** from the remote repository but **does not merge** them into your local branch. You can **review** changes before merging.

git pull : It combines both *git fetch* and *git merge*. It basically downloads **changes** and **automatically merges** them into your current local branch.

Version Control System

If we want to work in a specific feature, change or solve an issue, we will need to create a new **branch**:

```
git checkout -b new_feature
```

Use now `git status` or `git branch` to see where you are at

From there, we will do our changes and pushes. Once the branch work is finished, we may want to merge it with the main branch:

```
git checkout main
```

 to go back to the main branch

```
git merge new_feature
```

 it will not complain if there are not conflicts (main vs new branch)

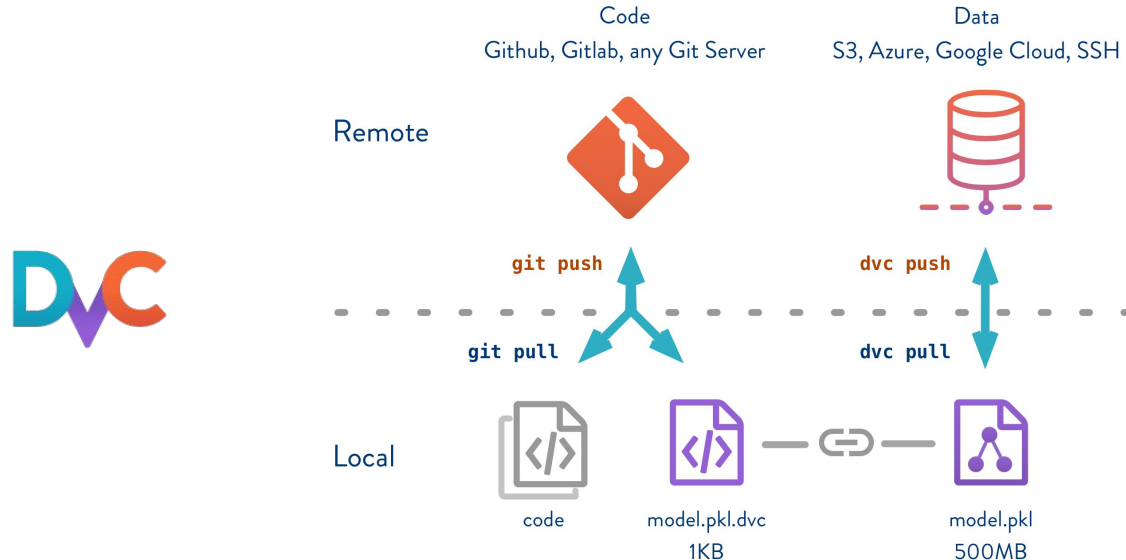
```
git branch -d new_branch
```

 to remove the branch (optional) and then

```
git push origin main
```

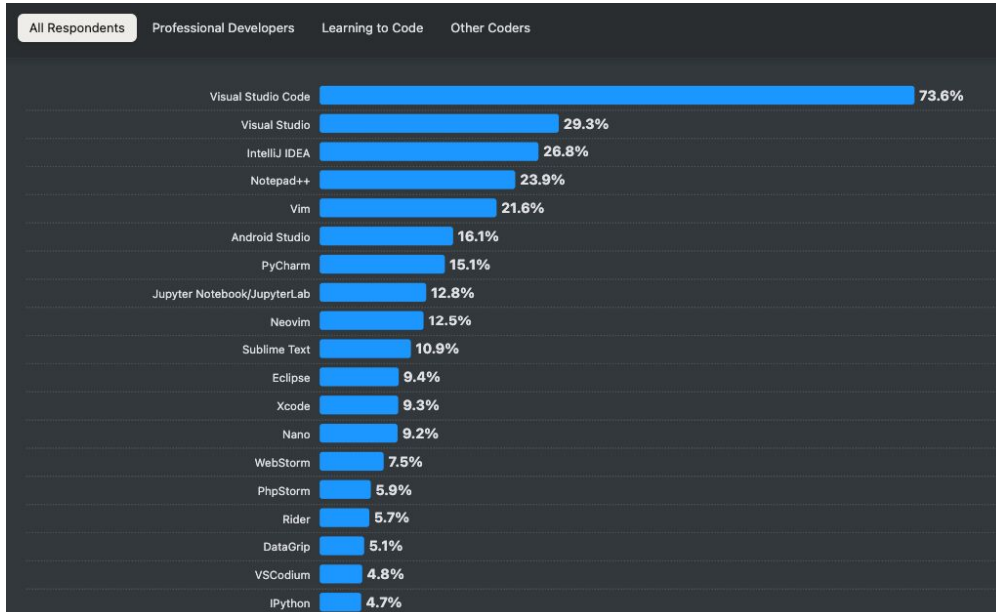
Version Control System

When working with data, you may want to also keep control of what version of your data you worked with at every stage of your code. One of the most extended one is DVC:



Integrated Development Environment

These are the most used IDEs out there in 2024 (stackoverflow) are:

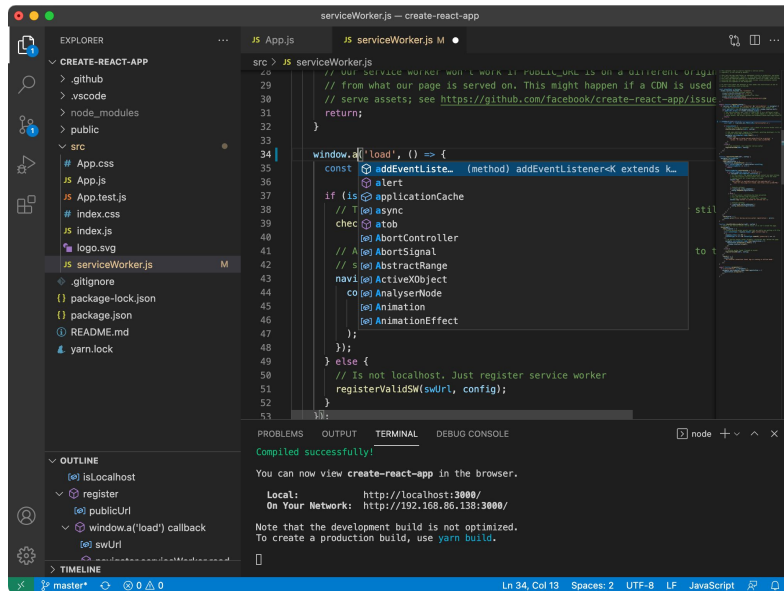


Where the most used ones for data science with Python are:

- JupyterLab Notebook & Jupyter Notebook
- Visual Studio Code
- PyCharm
- Spyder
- Google Colab

Integrated Development Environment

Our choice for this course is VSCode, an open-source IDE developed and maintained by Microsoft. It supports Python and other languages, and comes with multiple extensions and plugins.



Exercises

Let's install it in our Laptop, create a new repository for Session #2 and do some work!

General hints:

- Repository name
 - Checkout that the name was not already used by other developers
 - Usually try that the name makes sense with the purpose (e.g. pandas comes from panel data)
- Reproducibility
 - As de facto you should always include a README.md file
 - Write down a concise and well-written description (purpose, people, functionalities...)
 - Add a step by step guide to use your repository
 - Add requirements.txt / dockerfile if needed
- Structure
 - Create an easy to navigate structure in your repository (folders for jupyter notebook, main folder code, etc)
- Logo (optional)
- Create a professional and good portfolio

Exercises

You will update your first repository and for each exercise (after creating the readme) create an individual branch.

Exercise 1: FizzBuzz

1. **Write a FizzBuzz Function:** Create a function `fizzbuzz(n)` that takes an integer `n` as a parameter.
2. **Implement FizzBuzz Logic:** The function should print:
 - o "Fizz" for multiples of 3
 - o "Buzz" for multiples of 5
 - o "FizzBuzz" for multiples of both 3 and 5
 - o The number itself for other numbers
3. **Call the Function:** Call the function for numbers 1 to 20.

Exercise 2: Basic Data Filtering

1. **Create a List of Mixed Data Types:** Create a list that contains a mix of integers, strings, and floats.
2. **Filter the List:** Use list comprehension to create a new list that contains only the integers from the original list.
3. **Print the New List:** Output the filtered list of integers.

Exercises

Exercise 3: Simple To-Do List

1. **Create an Empty List:** Start with an empty list called `todo_list`.
2. **Define Functions:**
 - A function `add_task(task)` that adds a task to the list.
 - A function `show_tasks()` that prints all tasks in the list.

Exercise 4: Temperature Converter

1. **Define a Conversion Function:** Write a function `celsius_to_fahrenheit(celsius)` that converts Celsius to Fahrenheit.
2. **Print the Result:** Output the converted temperature for 22°F, 46°F, 51°F and 76°F.