

# Setting up Anaconda for *Introduction to Machine Learning*

**Author:** Marco Baumann **Version:** March 2024

## General Notes

This guide is intended to give you a starting point to set-up a Python environment with Anaconda which can be used for your projects.

## What is Anaconda

Anaconda is a distribution for R and Python allowing users to easily install and manage packages and libraries. It is specifically tailored towards scientific applications and a great tool to manage and create execution environments for machine learning. However, it is not the only option. According to personal preference, you might also choose to setup a virtual environment using VENV instead (see: <https://docs.python.org/3/library/venv.html>), which is somewhat more lightweight.

## Anaconda vs Miniconda

Both Anaconda and Miniconda provide very similar functionality. However, Miniconda uses less memory as it ships with fewer preinstalled dependencies. This guide is applicable for both Anaconda *and* Miniconda. So, if you are low on memory, you might want to use Miniconda instead.

## Installing Anaconda

Anaconda is supported on Linux, MacOS and Windows. First, download the Anaconda installer from:

**Anaconda:** <https://www.anaconda.com/download>

**Miniconda:** <https://docs.anaconda.com/free/miniconda/miniconda-install/>

Follow the respective instructions provided on <https://docs.anaconda.com/free/anaconda/install/> for your respective operating system.

If you use a graphical installer, beware that we need Anaconda to be added to our PATH variables. On some systems, there is a check-box that is not set by default, which does exactly that. Anaconda itself does not recommend checking the box, but rather manually adding it to your PATH variables. In most cases, checking the box should not lead to any issues and simplify installation. If you want to be safe, however, you should add it manually (see below on how to do that). If you check the box and run into issues, you might need to remove and reinstall Anaconda.

## Verifying installation

Before actually working with Anaconda and setting up our environment, we need to ensure your Anaconda installation is working as expected.

Open a terminal and try:

```
1 | conda --version
```

This command should return the currently installed Anaconda version and should look something like: `conda 24.1.2`.

Should this fail and you get an error message stating that the command `conda` is not known/found, you probably do not have Anaconda added to your PATH variables. PATH variables are locations where your computer will look for commands and programs. For your system to recognize the `conda` command, you hence first need to tell your device where to find it. The process of adding PATH variables differs depending on the operating system. For a comprehensive guide on how to do it on different platforms, see: <https://www.geeksforgeeks.org/how-to-setup-anaconda-path-to-environment-variable/>. The solution for *Linux* should also be applicable to *MacOS* users. Make sure to retry `conda --version` in a *fresh* terminal after setting environment variables or, alternatively, reloading `~/ .bashrc` on Unix systems.

# Setting up an environment

## Updating Anaconda and package manager

Before actually installing anything, we first make sure Anaconda is up to date and its package manager knows about recent package versions. Please run:

```
1 | conda update -n base anaconda # if this does not work, ignore it (e.g. miniconda)
2 | conda update -n base conda # this should work! -> updates package manager
```

## Creating an environment

Next, we can set up an Anaconda execution environment. For that, use the command:

```
1 | conda create -n <name> python=<python_version>
```

where `name` is the name you want to give to your environment and `python_version` is the Python-version you want to use.

**Important Note:** We highly recommend using a well established Python version, since many packages do not yet support the newest Python releases. This can cause various compatibility issues. Good versions to use are e.g.

`v3.10` or similar.

**Example:** Creating a new environment called `iml` using python `v3.10` looks like:

```
1 | conda create -n IML python=3.10
```

If everything works fine, you should see a list of packages to be downloaded and installed. Confirm with `y`.  
Congrats, you've just created an Anaconda environment.

## Activating/Enabling environments

Anaconda environments are often not active by default, but need to be activated. Activating an environment will tell your system to use the Anaconda environment as Python runtime in the current terminal instead of the system default.

An environment can be **activated** using

```
1 | conda activate <name>
```

Where `name` again refers to the name of the environment you want to activate. Most likely, you'll see `(<name>)` on the left of your input prompt after having executed this command. This means environment `<name>` is active. Anaconda also has a `base` environment, which sometimes is activated by default. After activating your environment try:

```
1 | python --version
```

You should now see the Python version you previously chose (or a subversion of it) when creating the environment. Sometimes, you need to use the `python3` instead of the `python` command for it to work, depending on your system and installation.

Using:

```
1 | python example.py
```

executes the local Python file: `example.py` in the activated environment. This allows you to execute Python files in the desired environment using your terminal. You need to replace `example.py` with the path to the file you want to execute (just a filename without a path assumes the file is located in the current working directory, as is standard for shells).

## Installing packages

At this point, you've already got everything you need to start installing packages and libraries for the course. One of Anaconda's main strength is its huge collection of packages/libraries for data science and the offered simplicity of their installation.

By default, Anaconda already comes with a bunch of pre-installed packages with every newly created environment.

1. Make sure your previously created environment is enabled (i.e. you have run `conda activate <name>` in the *current shell session*)
2. You can list all packages that are already installed in your environment with:

```
1 | conda list
```

3. To install new packages, simply run:

```
1 | conda install <packetname>
```

This should install a packet. Anaconda uses a transaction based installation system. Hence, if an installation fails, it will safely revert all changes.

Some frequently used and very useful packages are:

- `numpy` (Python's numerical array and matrix computations library)
- `scikit-learn` (this is the sklearn packet)
- `matplotlib` (plotting library for data visualizations)
- `pandas` (allows working with dataframes and `.csv` files)
- `seaborn` (another great library for visualizations)
- `scipy` (includes useful metrics and tools for scientific calculations)
- `pytorch` (see section below for details and installation)

Instead of separately installing all packages as in:

```
1 | conda install numpy
2 | conda install scikit-learn
3 | ...
```

you can also install multiple packets with a single `install` command.

```
1 | conda install numpy scikit-learn pandas # and so on (only requires a single line)
```

Further packages you might require can be installed in the same manner. If conda package manager is not able to find the packet (very unlikely for any packet you'll need for the course), you can try using different channels like `conda install -c conda-forge <packetname>` or use python's `pip` to install a packet in your environment using: `pip install <packetname>`. It might make sense to update `pip` before doing so with:

```
1 | pip install --upgrade pip # only required if you work with pip
```

Also, make sure you have a working Internet connection during installation, as it is required to download the packets.

4. Rerun `conda list` in your environment to check that all packets have been properly installed.

## Installing PyTorch

PyTorch is an extremely powerful library for more advanced machine learning and deep learning algorithms and will be used in later projects.

PyTorch also includes a vast collection of pretrained models and popular ML-datasets, as well as preprocessing suites and much more.

PyTorch needs special care to install, because you can either run it on CPU or GPU. If you have a device with a dedicated GPU supporting CUDA (unfortunately not available on MAC), we recommend using the GPU installation of PyTorch, as it is **significantly** faster. However, you should also be fine to work with the *default* installation which uses the CPU.

For installation, follow: <https://pytorch.org/get-started/locally/>

Select: *Stable*, then your OS, then *Conda*, then you CUDA version for GPU or *default/CPU* for CPU. This then produces the installation command you should run in the activated Anaconda environment. On Unix systems, you can use `nvidia-smi` to find out which version of CUDA your NVIDIA drivers are compatible with (required NVIDIA card and associated drivers).

An example for Mac is:

**NOTE:** Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

PyTorch Build	Stable (2.2.1)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 5.7	Default
Run this Command:	<code>conda install pytorch::pytorch torchvision torchaudio -c pytorch</code>			

## Using the environment in IDEs

If you are using an IDE, you need to properly configure it so it uses your Anaconda environment to execute Python code. For most IDEs this is very straight forward and also well documented on the Internet. We will be demonstrating setting up *Visual Studio Code* and *JetBrains* IDEs here. For other IDEs please refer to their respective online guides.

### VS Code

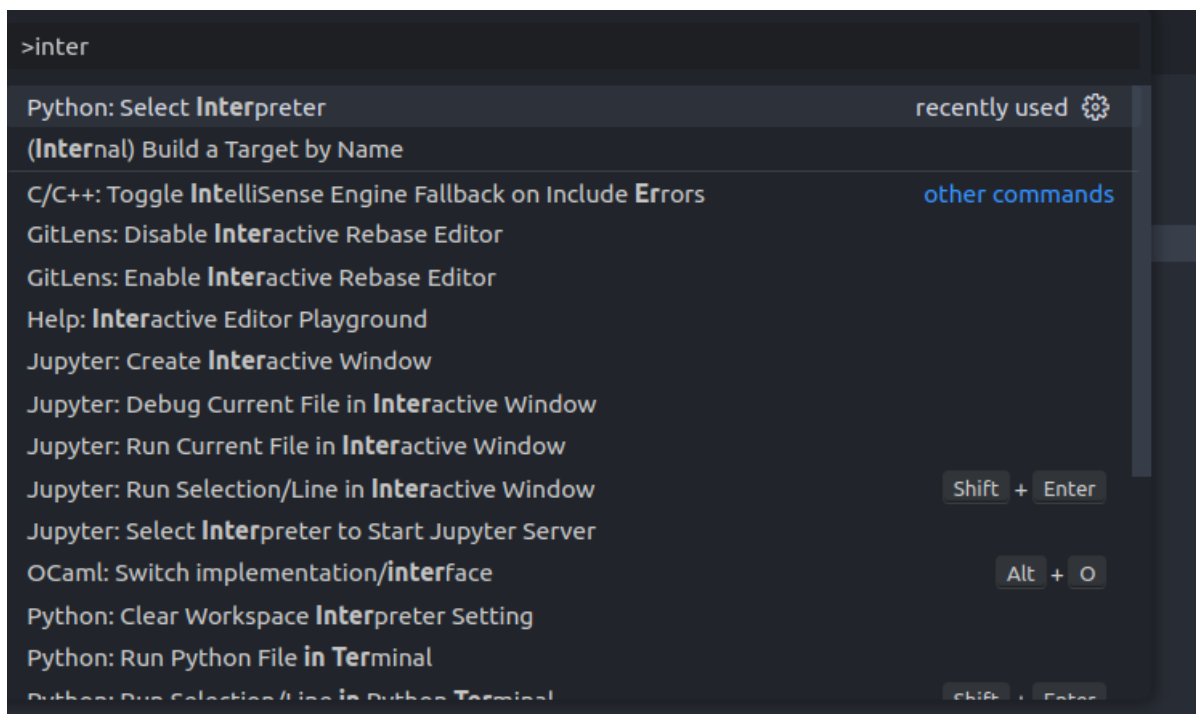
You can either directly run your files in from the console as mentioned above:

```
1 | python <local path to .py file>
```

Note, you need to have your environment enabled when doing so (remember `conda activate <name>`). If the environment is not activated, the system will execute the command with the default Python installation (which probably does not have the correct packages installed).

Alternatively, configure VSCode so that the `run` button automatically uses your Anaconda setup.

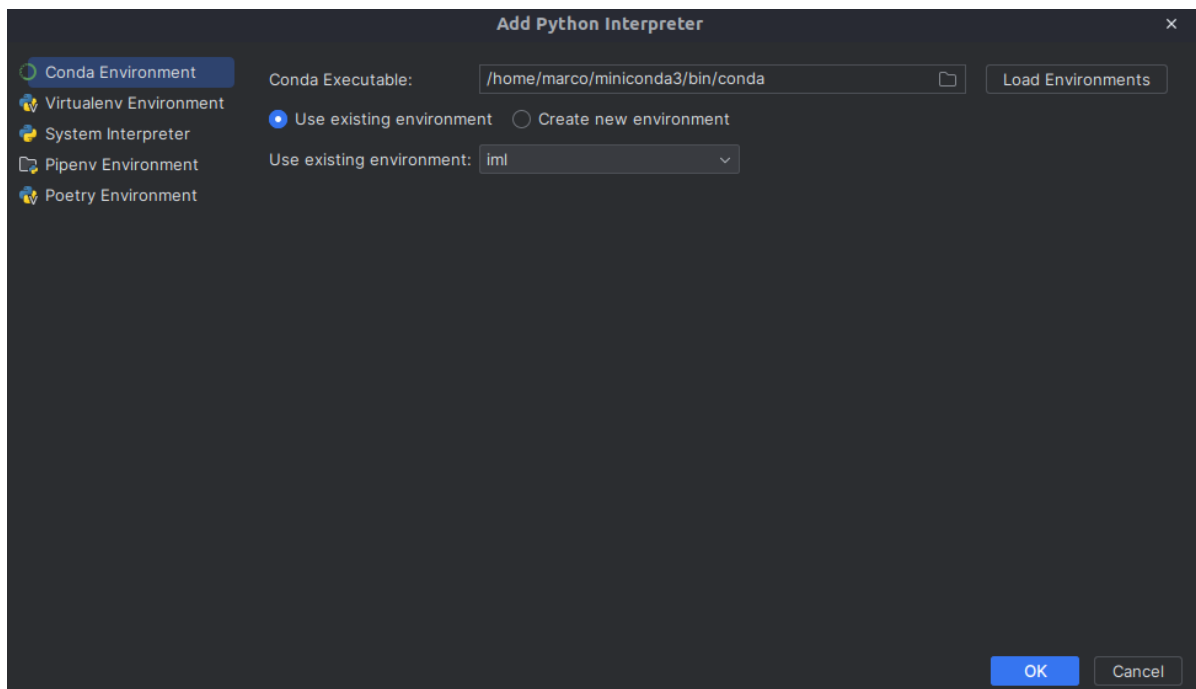
For that, press `Ctrl+Shift+P` (or `cmd` instead of `ctrl` on Apple-devices) to open the search dialog. Type `interpreter` and select: `Python: Select Interpreter`.



A drop-down list should appear, including your Anaconda environment. Select it. After that, pressing the `run` button in the IDE should execute your code in the correct environment. It is possible that you need to have the Python extension installed for this to work.

## JetBrains IDEs

As before, you can use your terminal to execute code. Alternatively, right-click on your project folder and choose `interpreter`. If your anaconda environment is not yet listed, select `Add local interpreter`. This opens a window (see img) in which you can choose your Anaconda environment:



Select `Use existing environment` and select your environment from the drop-down list. Confirm with `ok`. After some indexing, your IDE is properly configured.

This approach should work for various JetBrains IDEs like: *IntelliJ*, *PyCharm* or *DataSpell*