ML Basics

Python Setup

Learning goals

- Understand the different package management systems
- Be comfortable in settig up a python environment
- Know how to make your work reproducible with Python

0. INSTALLATION OF PYTHON

Python can be installed in many ways, e.g. by

- directly downloading it from python.org:
  https://www.python.org/downloads/
- downloading Miniconda:
  https://docs.conda.io/en/latest/miniconda.html
- installing Python via the Anaconda distribution:
  https://docs.anaconda.com/anaconda/install/

0. PACKAGE MANAGEMENT IN PYTHON

Unlike in R/RStudio (which you might be used to), package management in Python is a little more of a mess: There are *two* package management systems:

- `pip` (+ PyPi)
- `conda` (+ Anaconda repository)
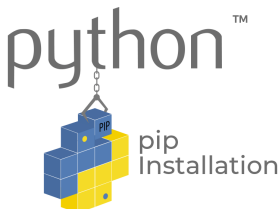
0. PACKAGE MANAGEMENT IN PYTHON

`pip`

- Installs packages from the Python Package Index (PyPi)
- Pip packages are source distributions (compiler needed) or wheels[1]
- Installs dependencies in a recursive, serial loop
- Limited to Python software
- No built-in support for **virtual environments** (but possible)
- $> 150k$ packages available on PyPi

[1]tl;dr: wheels are smaller + install faster than a source distribution and do not require a compiler

0. PACKAGE MANAGEMENT IN PYTHON

conda

- Installs packages from the Anaconda repository
- Conda packages are binaries, no need for compilers
- SAT solver to verify requirements of all packages
- Not limited to Python software
- Easier to create/manage **virtual environments**
- $\sim$ 1.5k packages available in the Anaconda repository

1. PYTHON VIA THE CONSOLE

Start Python in Terminal / Console / Command Line

```
1 python
```

Approximate result:
```
Python 3.7.11 (default, Jul 27 2021, 07:03:16)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Similar to R console:

```
1 1+1
```

Exit with Ctrl-D or

```
1 exit()
```

2. ANACONDA

Anaconda is a widely used open-source distribution of Python:
`https://www.anaconda.com`

3. MINICONDA

We recommend downloading and installing *Miniconda*, since it

- directly brings `conda` (as well as `pip`)
- is a minimal installer (only the core parts, unlike Anaconda)

Source: https://www.mrdbourke.com/
get-your-computer-ready-for-machine-learning-using-anaconda-miniconda-and-conda/

4. SPYDER

Spyder is a complete IDE for Python (very similar to RStudio)

- directly comes with the Anaconda distribution
- can also be installed from https://www.spyder-ide.org/
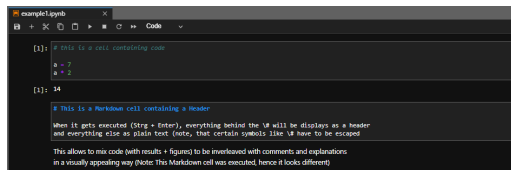
5. JUPYTER NOTEBOOKS/LAB

- web application that lets you develop Python code in your browser
- combine code, text, images, and other media
- execute your code in smaller parts (cells)
- Useful for explorative analyses and for beginning stages of a project, where you need frequent feedback


```
1 conda install jupyterlab
```

5. JUPYTER NOTEBOOKS/LAB

Code cells vs. Markdown cells

Useful for creating and changing figures quickly
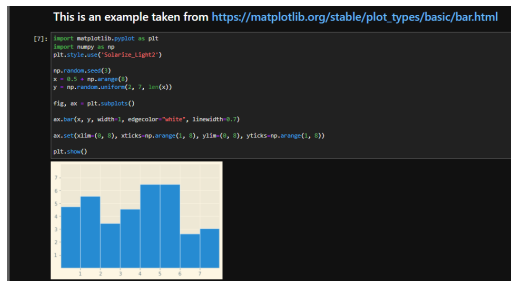
5. JUPYTER NOTEBOOKS/LAB

- When you click into a cell and you see your cursor, then you are in **Edit Mode**
- Write your Code/Explanations and execute with `Ctrl+Enter`
- Hit `Esc` → Cursor disappears, cell is still selected
  (you are in **Command mode** now)
- Now there are, among others, certain shortcuts available:
  - `A` will insert a new cell *above* the current one
  - `B` will insert one *below*
  - `C` copies the selected cell
  - `X` cuts the selected cell
  - `V` pastes copied/cut cell below
  - `M` switches the cell mode to *Markdown*
  - `Y` switches it to *Code*
  - `Up` selects cell above
  - `Down` selects cell below

6. VISUAL STUDIO CODE

- Widely used editor for many programming languages
- Main advantage: supports Jupyter Notebooks, takes care of setting up server
- install from `https://code.visualstudio.com/`
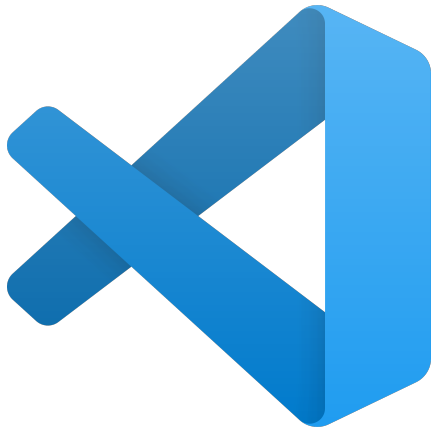
7. R-STUDIO

RStudio can also be used as an IDE for Python:

- Install the `reticulate` packages in R
- Select the desired Python interpreter at
  `Tools > Global Options > Python`
- `New File > Python Script`

8. VIRTUAL ENVIRONMENTS

Assume the following scenario:

- We have two ongoing projects, Project A and Project B
- Both projects need some package pkg_xy
- Project A needs v1.2.0 while Project B requires v2.1.3
- What can we do about this?

Solution: **Work with virtual environments**

8. VIRTUAL ENVIRONMENTS

Virtual Environments allow you to

- have different entire version of Python running on your machine
  (Python 2, Python 3.x, Python 3.y, ..)
- maintain different versions of packages for your projects

They can be created/activated/deactivated via

```
conda create --name some_name python=3.9
conda activate some_name
conda deactivate some_name
```

Useful commands: Conda Cheat Sheet

8. VIRTUAL ENVIRONMENTS

Different versions of Python

```
C:\Users\ri85vex>conda activate helloworld

(helloworld) C:\Users\ri85vex>python
Python 3.8.3 (default, May 19 2020, 06:50:17) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

(helloworld) C:\Users\ri85vex>conda activate inhouse

(inhouse) C:\Users\ri85vex>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Different (versions of) packages

```
(inhouse) C:\Users\ri85vex>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import matplotlib
>>> exit()

(inhouse) C:\Users\ri85vex>conda activate helloworld

(helloworld) C:\Users\ri85vex>python
Python 3.8.3 (default, May 19 2020, 06:50:17) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import matplotlib
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'matplotlib'
>>>
```

9. MANAGING PACKAGES

With `conda`:

```
1 conda install pandas
2 conda install pandas==1.3.5
3 conda update pandas
4 conda remove pandas
5 conda list
```

With `pip`:

```
1 pip install pandas
2 pip install pandas==1.3.5
3 pip install pandas --upgrade
4 pip uninstall pandas
5 pip freeze
```

9. MANAGING PACKAGES

Instruct `conda` which version you want to have installed:

- Exact (1.3.5)

```
1 conda install pandas ==1.3.5
```

- Fuzzy (1.3.0, 1.3.1, etc.)

```
1 conda install pandas ==1.3
```

- Greater or equal (1.3.5 or higher)

```
1 conda install "pandas >=1.3.5"
```

- OR (1.3.4 or 1.3.5)

```
1 conda install "pandas ==1.3.4|1.3.5"
```

- AND (1.3.3, 1.3.4, but not 1.4)

```
1 conda install "pandas >=1.3.3 ,<1.4"
```

Standardized way of sharing code requirements with others publicly across platforms:

- `environment.yml` (conda)
- `requirements.txt` (pip)

10. REPRODUCIBILITY

- Create an environment:[2]
  (with specific Python version and some packages)

```
1  conda create -n my_env python=3.7 pandas numpy scikit-learn
```

- Activate the conda environment:

```
1  conda activate my_env
```

- Export the specifications of your conda environment:

```
1  conda env export > environment.yml
```

---

[3]-n is the short version of the --name flag

10. REPRODUCIBILITY

Result (abbreviated to fit on the slide):

```
 1  name: my_env
 2  channels:
 3    - defaults
 4  dependencies:
 5    - blas=1.0=mkl
 6    - bottleneck=1.3.2=py37h2a96729_1
 7    - ...
 8    - numpy=1.21.2=py37hfca59bb_0
 9    - numpy-base=1.21.2=py37h0829f74_0
10    - openssl=1.1.1l=h2bbff1b_0
11    - packaging=21.3=pyhd3eb1b0_0
12    - pandas=1.3.4=py37h6214cd6_0
13    - pip=21.2.4=py37haa95532_0
14    - pyparsing=3.0.4=pyhd3eb1b0_0
15    - python=3.7.11=h6244533_0
16    - python-dateutil=2.8.2=pyhd3eb1b0_0
17    - pytz=2021.3=pyhd3eb1b0_0
18    - scikit-learn=1.0.1=py37hf11a4ad_0
19    - scipy=1.7.1=py37hbe87c03_2
20    - ...
21  prefix: C:\Users\ri85vex\Miniconda3\envs\my_env
```

10. REPRODUCIBILITY

- *Note 1:* As you can see, this also includes all sorts of packages that are automatically installed when creating an environment
- *Note 2:* We observe some cryptic stuff after the version of each package
- *Note 3:* Those packages installed via pip are **also handled by this file** (see next slide)
- *Note 4:* The prefix is ignored when using the .yml-file for creation. So you can simply delete it after the export.

10. REPRODUCIBILITY

- Install some package via `pip`:

```
1 pip install matplotlib
2
```

- Result (abbreviated to fit on the slide):

```
1  name: my_env
2  channels:
3    - defaults
4  dependencies:
5    - blas=1.0=mkl
6    - bottleneck=1.3.2=py37h2a96729_1
7    - ca-certificates=2021.10.26=haa95532_2
8    - certifi=2021.10.8=py37haa95532_0
9    - icc_rt=2019.0.0=h0cc432a_1
10   - intel-openmp=2021.4.0=haa95532_3556
11   - ...
12   - pip:
13     - cycler==0.11.0
14     - fonttools==4.28.5
15     - kiwisolver==1.3.2
16     - matplotlib==3.5.1
17     - pillow==9.0.0
18 prefix: C:\Users\ri85vex\Miniconda3\envs\my_env
19
```

10. REPRODUCIBILITY

- Using the "–from-history" flag

```
1 conda env export --from-history > environment.yml
2
```

- allows you to create a reduced version of the `.yml`-file containing only the packages (+ version) you explicitly installed (**via conda**):

```
1 name: my_env
2 channels:
3   - defaults
4 dependencies:
5   - scikit-learn
6   - numpy
7   - python=3.7
8   - pandas
9 prefix: C:\Users\ri85vex\Miniconda3\envs\my_env
10
```

10. REPRODUCIBILITY

- Export specifications of the `pip` packages:

```
1 pip freeze > requirements.txt
2
```

- Can be installed in a conda environment via:

```
1 pip install -r requirements.txt
2
```

- **But**: Requiring others to execute these two steps will create a lot of work!

10. REPRODUCIBILITY

- Best practice: Create `.yml`-file manually

```
1  name: my_env
2  channels:
3    - defaults
4  dependencies:
5    - python=3.7.11
6    - numpy=1.21.2
7    - pandas=1.3.4
8    - scikit-learn=1.0.1
9    - pip=21.2.4
10   - pip:
11     # works for regular pip packages
12     - matplotlib==2.0.0
13     # also works for whole requirements-files:
14     - -r requirements.txt
15
```

- Finally: Create environment from `.yml`-file:

```
1  conda env create -f environment.yml
2
```