0. Setup Python

Machine Learning and Deep Learning with Python

Dr. Ludwig Bothmann, Dr. Matthias Aßenmacher

April 29, 2022

0. Installation of Python

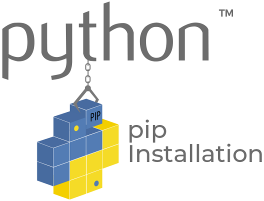
Python can be installed in many ways, e.g. by

- .. 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* (concurring) package management systems:

- `pip` (+ PyPi)
- `conda` (+ Anaconda repository)



0. Package management in Python

Key facts (`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

Key facts (`conda`)

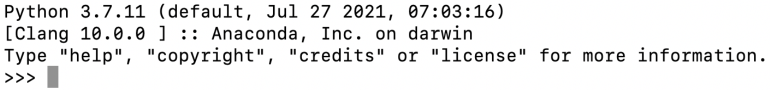
- Installs packages from the Anaconda repository
- Conda packages are binaries, no need for compilers
- SAT solver to verify requirements of all packages
- Not limited to Python software
- Easier to create/manage **virtual environments**
- $\sim$ 1.5k packages available in the Anaconda repository

1. Python via the console

- Start Python in Terminal / Console / Command Line

```
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
```

```
## 2
```

- Exit with Ctrl-D or

```
exit()
```

2. Anaconda

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

3. Miniconda

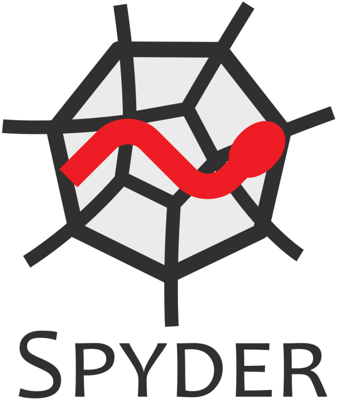
We recommend downloading and installing *Miniconda*, since it

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

Source: https://www.mrdbourke.com/

4. Spyder

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



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

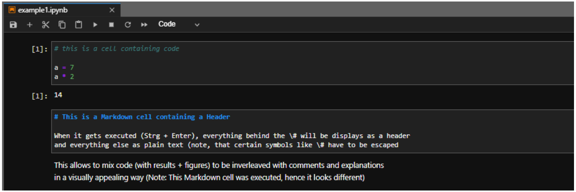
5. Jupyter Notebooks/Lab

A Jupyter Notebook is a slightly different way of writing and presenting Python codes. It is a web application that lets you develop Python code in your browser.
Useful for explorative analyses and for beginning stages of a project, where you need frequent feedback.

```
conda install jupyterlab
```

5. Jupyter Notebooks/Lab

- Code cells vs. Markdown cells

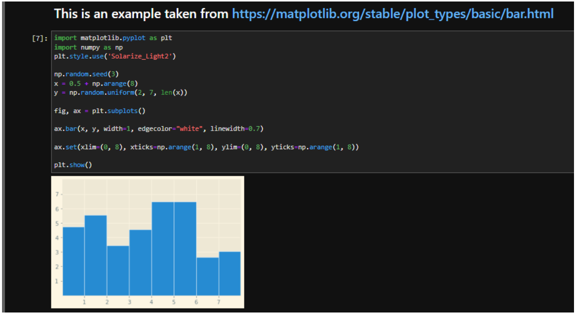


```
[1]: # this is a cell containing code

     a = 7
     a * 2

[1]: 14
```

```
# This is a Markdown cell containing a Header

When it gets executed (Strg + Enter), everything behind the \# will be displays as a header
and everything else as plain text (note, that certain symbols like \# have to be escaped
```

This allows to mix code (with results + figures) to be inverleaved with comments and explanations
in a visually appealing way (Note: This Markdown cell was executed, hence it looks different)

- Creating e.g. figures



This is an example taken from https://matplotlib.org/stable/plot_types/basic/bar.html

```
[7]: import matplotlib.pyplot as plt
     import numpy as np
     plt.style.use('Solarize_Light2')

     np.random.seed(3)
     x = 0.5 + np.arange(8)
     y = np.random.uniform(2, 7, len(x))

     fig, ax = plt.subplots()

     ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)

     ax.set(xlim=(0, 8), xticks=np.arange(1, 8), ylim=(0, 8), yticks=np.arange(1, 8))

     plt.show()
```

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. RStudio

It is (meanwhile) also possible to use good old RStudio as an IDE for writing Python code:

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

6. RStudio

You can also use Python in conjunction with R Markdown (as you will observe quite often in our slides)

```
import pandas as pd

x = pd.DataFrame({"a": [1,2,3,4],
                  "b": ["c", "d", "e", "f"]})
x
```

```
##    a  b
## 0  1  c
## 1  2  d
## 2  3  e
## 3  4  f
```

7. 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**

- You can also tell RStudio to activate environments automatically by ticking the box at:

☒ `Automatically activate project-local Python environments`

7. 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

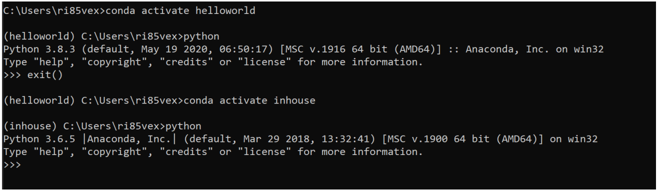
They can be created/activated/deactivated via

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

Useful commands: Conda Cheat Sheet

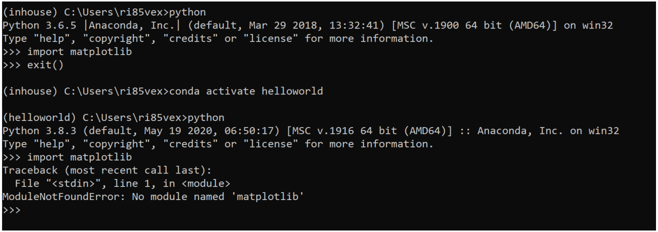
7. Virtual Environments

- 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'
>>>
```

8. Managing packages

- Via `conda`:

```
conda install pandas
conda install pandas==1.3.5
conda update pandas
conda remove pandas
conda list
```

- Via `pip`:

```
pip install pandas
pip install pandas==1.3.5
pip install pandas --upgrade
pip uninstall pandas
pip freeze
```

8. Managing packages

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

- Exact (1.3.5)

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

- Fuzzy (1.3.0, 1.3.1, etc.)

```
conda install pandas=1.3
```

- Greater or equal (1.3.5 or higher)

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

- OR (1.3.4 or 1.3.5)

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

- AND (1.3.3, 1.3.4, but not 1.4)

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

9. Reproducibility

Creating isolated environments and running different versions of Python as well as different packages (or versions) of them works perfectly fine *(for us)*.

What if we want to share our code with others? Make it publicly available? Across platforms?

There needs to be some standardized way to do so, right?[2]

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

[2] Rhetorical question ;-)

9. Reproducibility

- Create an environment:[3]
  (with specific Python version and some packages)

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

- Activate the conda environment:

```
conda activate my_env
```

- Export the specifications of your conda environment:

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

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

9. Reproducibility

- Result (abbreviated to fit on the slide):

```
name: my_env
channels:
  - defaults
dependencies:
  - blas=1.0=mkl
  - bottleneck=1.3.2=py37h2a96729_1
  - ...
  - numpy=1.21.2=py37hfca59bb_0
  - numpy-base=1.21.2=py37h0829f74_0
  - openssl=1.1.1l=h2bbff1b_0
  - packaging=21.3=pyhd3eb1b0_0
  - pandas=1.3.4=py37h6214cd6_0
  - pip=21.2.4=py37haa95532_0
  - pyparsing=3.0.4=pyhd3eb1b0_0
  - python=3.7.11=h6244533_0
  - python-dateutil=2.8.2=pyhd3eb1b0_0
  - pytz=2021.3=pyhd3eb1b0_0
  - scikit-learn=1.0.1=py37hf11a4ad_0
  - scipy=1.7.1=py37hbe87c03_2
  - ...
prefix: C:\Users\ri85vex\Miniconda3\envs\my_env
```

9. 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.

9. Reproducibility

- Install some package via `pip`:

```
pip install matplotlib
```

- Result (abbreviated to fit on the slide):

```
name: my_env
channels:
  - defaults
dependencies:
  - blas=1.0=mkl
  - bottleneck=1.3.2=py37h2a96729_1
  - ca-certificates=2021.10.26=haa95532_2
  - certifi=2021.10.8=py37haa95532_0
  - icc_rt=2019.0.0=h0cc432a_1
  - intel-openmp=2021.4.0=haa95532_3556
  - ...
  - pip:
    - cycler==0.11.0
    - fonttools==4.28.5
    - kiwisolver==1.3.2
    - matplotlib==3.5.1
    - pillow==9.0.0
prefix: C:\Users\ri85vex\Miniconda3\envs\my_env
```

9. Reproducibility

- Using the `--from-history` flag

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

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

```
name: my_env
channels:
  - defaults
dependencies:
  - scikit-learn
  - numpy
  - python=3.7
  - pandas
prefix: C:\Users\ri85vex\Miniconda3\envs\my_env
```

9. Reproducibility

- Export specifications of the pip packages:

```
pip freeze > requirements.txt
```

- Can be installed in a conda environment via:

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

- **But**: Requiring others to execute these two steps is cumbersome!

9. Reproducibility

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

```
name: my_env
channels:
  - defaults
dependencies:
  - python=3.7.11
  - numpy=1.21.2
  - pandas=1.3.4
  - scikit-learn=1.0.1
  - pip=21.2.4
  - pip:
    # works for regular pip packages
    - matplotlib==2.0.0
    # also works for whole requirements-files:
    - -r requirements.txt
```

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

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