PyCharm & Anaconda

This explains the steps needed to get a working setup for PyCharm & Anaconda (for Python).

Anaconda is a strong package manager for Python and other software, PyCharm is the most powerful IDE (Integrated Development Environment) currently available for Python (better then e.g. Spyder, it is also newer).

For the course Datenanalyse, this tools should be used. In case other tools are strongly preferred, this is to be noted to the assistants.

Anaconda

Download and install Anaconda (-Navigator, a Graphical User Interface available in certain OS). This is a package manager that takes care of Python packages (via conda install) but also works well with pip (the default Python package installer).

You can install it anywhere, it requires only user privileges (on Unix, may be different on Windows).

Anaconda provides the possibility to create so called “virtual environments”: in short, you can have different “working spaces” for different projects and in each different (versions of) packages installed. This also saves you in case you ever screw up an environment, you can simply delete and recreate it. When working together with PyCharm, we won’t directly notice this setup step, instead Conda is well integrated into PyCharm which fully takes care of it.

PyCharm

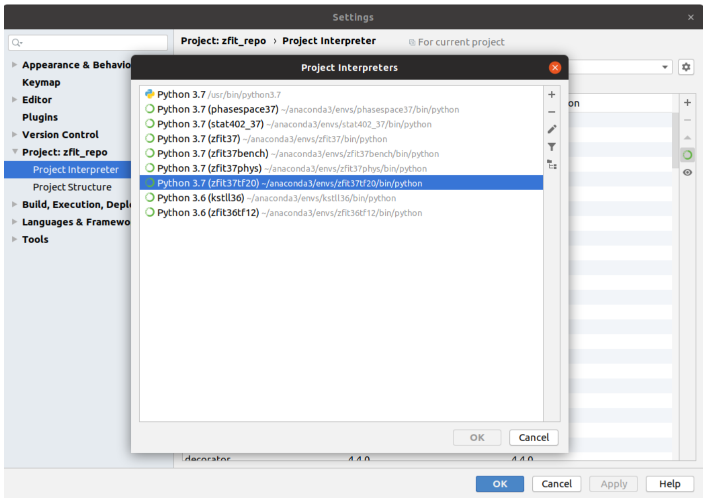
Download and install. The free version is sufficient for our use-case as it contains nearly all of the features. However, as a student using your e-mail you can even get the professional license for free.

After the installation, open it and either create a new project (in a new directory) or open an existing directory (tip: use the “data analysis” directory, or “exercises”, do not make a project for each single exercise).

When creating a new project, specify the location and use the menu (hidden first) below to define the project interpreter. Follow the instructions in the next section.

PyCharm is nothing else than a (fancy) text editor that allows you to edit .py files. In addition, it knows a lot about the Python language and takes a large amount of the work of writing code. More on that later. It is important to understand, that PyCharm itself is not Python, but just a text editor. We need to have separately Python installed (and PyCharm, being convenient, can execute a script using Python for us).

If you have skipped specifying the interpreter in the new project or also to proceed after the creation in order to install more packages, you can go to “settings”→ “project” → “project interpreter”, select from the drop-down menu of Python interpreters “show all” and then the “+” to add an interpreter. The following window should show up (with just the first environment, Python).



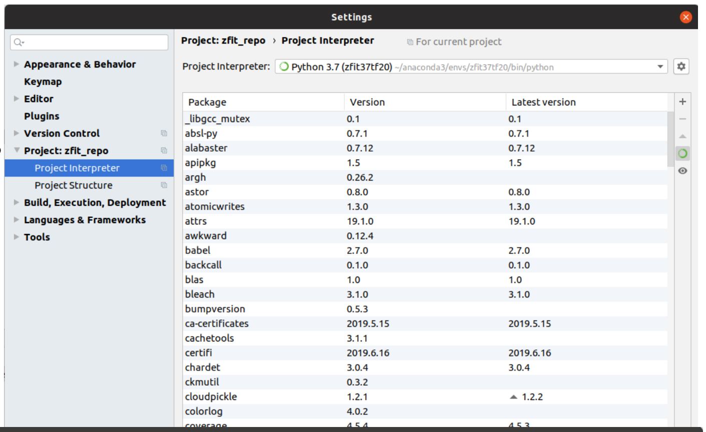
Project: zfit_repo › Project Interpreter   For current project

Python 3.7 /usr/bin/python3.7
Python 3.7 (phasespace37) ~/anaconda3/envs/phasespace37/bin/python
Python 3.7 (stat402_37) ~/anaconda3/envs/stat402_37/bin/python
Python 3.7 (zfit37) ~/anaconda3/envs/zfit37/bin/python
Python 3.7 (zfit37bench) ~/anaconda3/envs/zfit37bench/bin/python
Python 3.7 (zfit37phys) ~/anaconda3/envs/zfit37phys/bin/python
Python 3.7 (zfit37tf20) ~/anaconda3/envs/zfit37tf20/bin/python
Python 3.6 (kstll36) ~/anaconda3/envs/kstll36/bin/python
Python 3.6 (zfit36tf12) ~/anaconda3/envs/zfit36tf12/bin/python

Setting up the interpreter

On the left, click on “Conda Environment”. On the right, select (or keep selected) “New environment”, Python version should be 3.7. Check if “Conda executable” has a path (/home/.../anaconda3/bin/conda or similar). If it is empty, Anaconda was not installed properly.

Click “Ok”. The environment will now be created. Press again ok. Now we assume you are in the setting menu

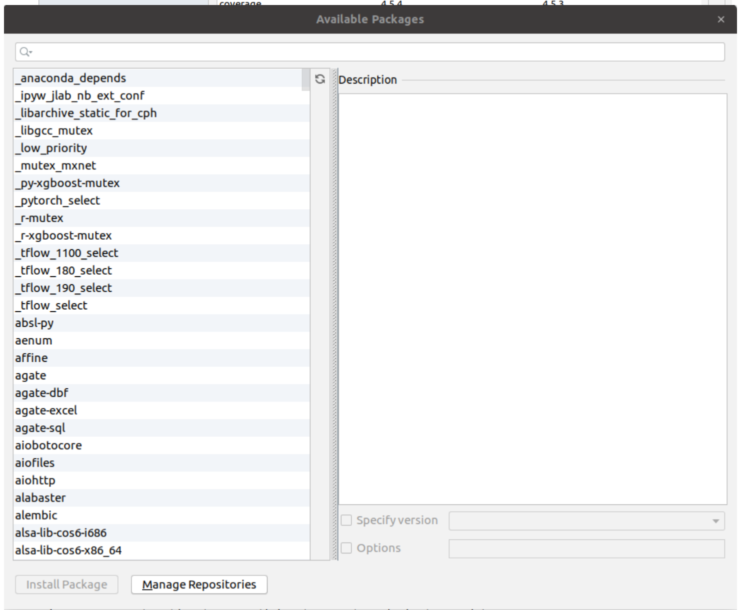


Project: zfit_repo › Project Interpreter   For current project

Project Interpreter: Python 3.7 (zfit37tf20) ~/anaconda3/envs/zfit37tf20/bin/python

Package	Version	Latest version
_libgcc_mutex	0.1	0.1
absl-py	0.7.1	0.7.1
alabaster	0.7.12	0.7.12
apipkg	1.5	1.5
argh	0.26.2	
astor	0.8.0	0.8.0
atomicwrites	1.3.0	1.3.0
attrs	19.1.0	19.1.0
awkward	0.12.4	
babel	2.7.0	2.7.0
backcall	0.1.0	0.1.0
blas	1.0	1.0
bleach	3.1.0	3.1.0
bumpversion	0.5.3	
ca-certificates	2019.5.15	2019.5.15
cachetools	3.1.1	
certifi	2019.6.16	2019.6.16
chardet	3.0.4	3.0.4
ckmutil	0.3.2	
cloudpickle	1.2.1	▲ 1.2.2
colorlog	4.0.2	

Click here on the “+”, which opens a menu to install any package you need.

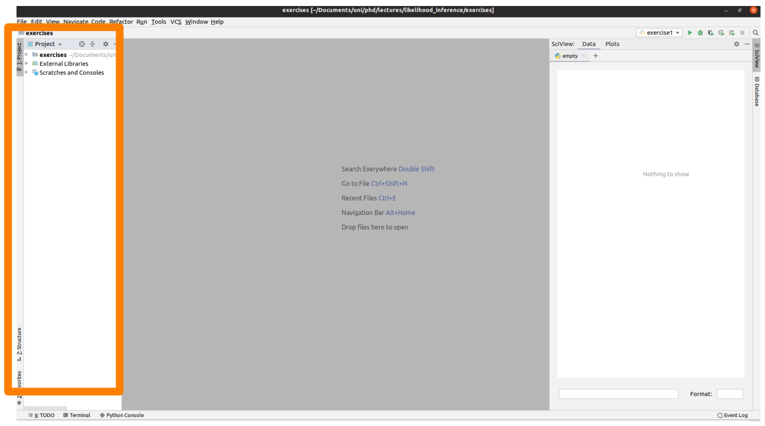


_anaconda_depends
_ipyw_jlab_nb_ext_conf
_libarchive_static_for_cph
_libgcc_mutex
_low_priority
_mutex_mxnet
_py-xgboost-mutex
_pytorch_select
_r-mutex
_r-xgboost-mutex
_tflow_1100_select
_tflow_180_select
_tflow_190_select
_tflow_select
absl-py
aenum
affine
agate
agate-dbf
agate-excel
agate-sql
aiobotocore
aiofiles
aiohttp
alabaster
alembic
alsa-lib-cos6-i686
alsa-lib-cos6-x86_64

In case it does not work, you can also go to the anaconda navigator and install packages from there.

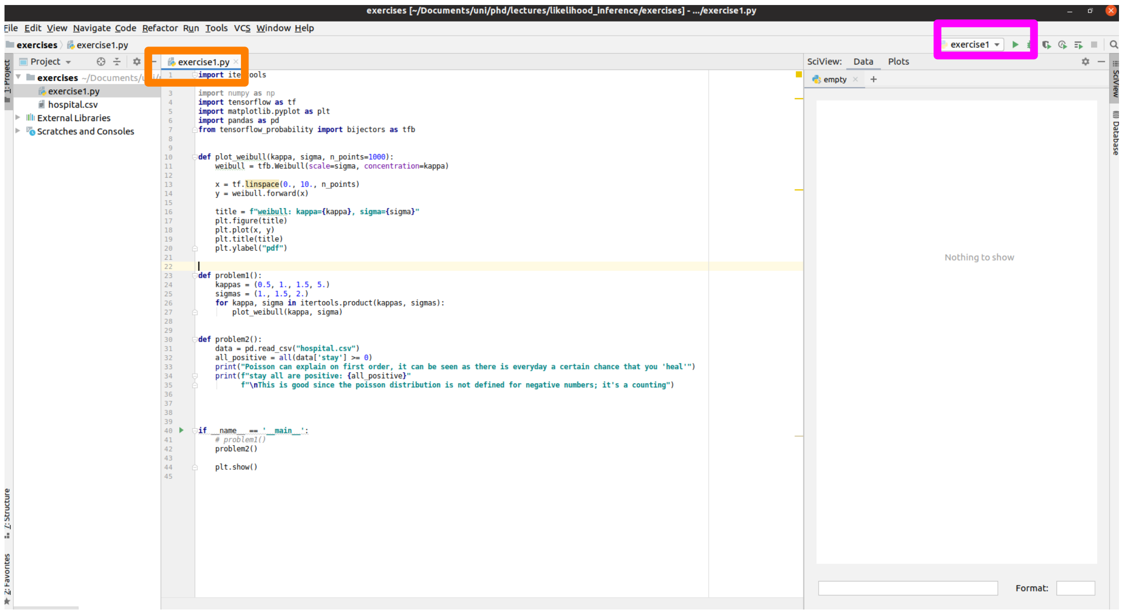
Creating and executing code

On the left side of the IDE, you should see a project navigator bar



Right-click on the folder that is there and create with "new" a new Python file (or open the folder by double-clicking and select an existing one). Start coding.

To execute the file, (usually required for the first time only), do a right-click on the file name (e.g. exercise1.py in this case) and on the drop-down menu select "run exercise1.py" (orange rectangle)



```python
import ite ools

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow_probability import bijectors as tfb


def plot_weibull(kappa, sigma, n_points=1000):
    weibull = tfb.Weibull(scale=sigma, concentration=kappa)

    x = tf.linspace(0., 10., n_points)
    y = weibull.forward(x)

    title = f"weibull: kappa={kappa}, sigma={sigma}"
    plt.figure(title)
    plt.plot(x, y)
    plt.title(title)
    plt.ylabel("pdf")


def problem1():
    kappas = (0.5, 1., 1.5, 5.)
    sigmas = (1., 1.5, 2.)
    for kappa, sigma in itertools.product(kappas, sigmas):
        plot_weibull(kappa, sigma)


def problem2():
    data = pd.read_csv("hospital.csv")
    all_positive = all(data['stay'] >= 0)
    print("Poisson can explain on first order, it can be seen as there is everyday a certain chance that you 'heal'")
    print(f"stay all are positive: {all_positive}"
          f"\nThis is good since the poisson distribution is not defined for negative numbers; it's a counting")




if __name__ == '__main__':
    # problem1()
    problem2()

    plt.show()
```

Every other time you can also run the script from the upper right (pink rectangle). If you run several scripts, you can also choose from the small drop-down menu which one to run.

PyCharm tools and tricks

There are endless powerful tools available in PyCharm. For the beginning, let's just list a few important ones:

- Ctrl + Q shows the docs of a function
- Ctrl + P (when placed inside a function call, e.g. “np.array(*curser_here*)”) will show the “signature” of the function; which arguments does it take and at which are you entering currently
- Alt + Enter shows (if available, usually when you see a red/yellow light bulb) options to solve problems or restructure things.
- Alt + Space (very important!): shows the auto-completion menu. Typing more letters brings the adequate method. Press enter to confirm.
- If you go on the menu “Code”, there is a “reformat code” (maybe change the shortcut for that in the settings). Do that regularly! Any code you hand in should be formatted in the way this function formats your code (if you don’t like it, please ask your assistant, namely Jonas).
- Ctrl + B brings you directly to the function definition (attention: you may end up in the definition of numpy or matplotlib functions, not your own. Don’t change anything there! These files as show yellowish.
- Advanced: To go step-wise through the code, instead of pressing the green arrow, you can press the bug (“käfer”), which launches the debugger. If you set a red dot before starting it by clicking on the left border of the file, it will stop there and allow you line by line execution of the and to even step inside functions.
- Way more: VCS integration, remote files, powerful refactoring, unittest integration etcetc