Datenanalyse (PHY231) — Herbstsemester 2021

PyCharm and Anaconda *jonas.eschle@physik.uzh.ch*

1 Setting up PyCharm and Anaconda

This documents explains the steps needed to get a working setup for PyCharm and Anaconda (for Python). It is *NOT* required to use *this exact* setup to solve the exercises, you are completely free in your choice. However, the setup is strongly recommended.

Anaconda is a package manager for Python and other software and allows to install packages in a simple manner, PyCharm is a powerful IDE (Integrated Development Environment) for Python and used for the actual coding.

1.1 Anaconda

Instruction: Download and install Anaconda (-Navigator, a Graphical User Interface).

`Explanation:` 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, no idea about Windows). Anaconda provides the possibility to create "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 step but PyCharm takes care of it.

1.2 PyCharm

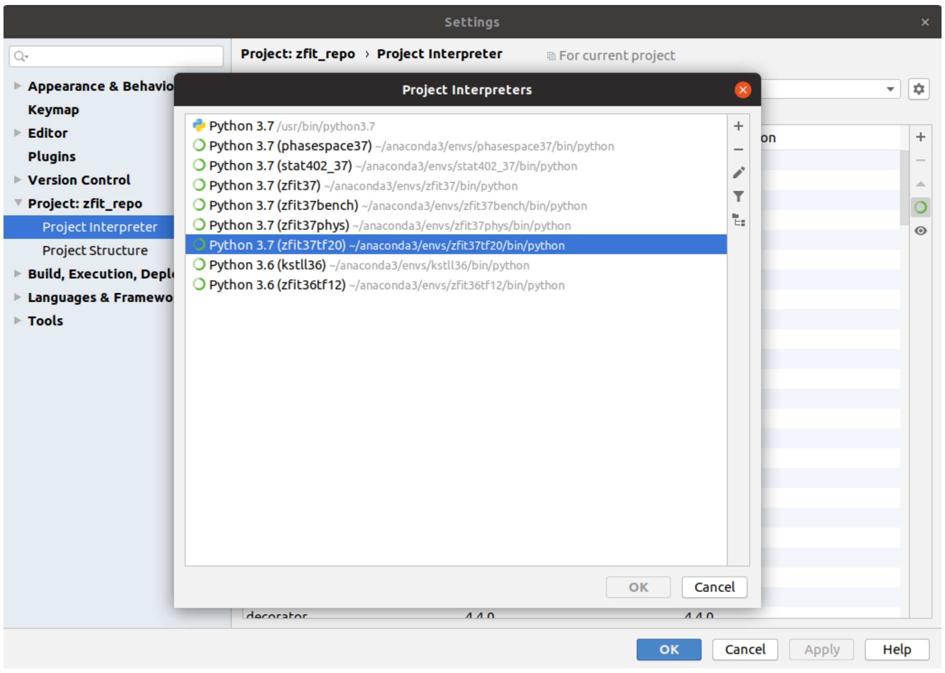
Instruction: Download and install PyCharm (either the free Community Edition or the Professional also free for students).

`Explanation:` 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.

Instruction: After installation, open PyCharmand either create a new project (in a new directory) or open an existing directory (hint: use the "data analysis" directory, or "exercises", do not make a project for each single exercise). If 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.

`Explanation:` 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 around writing code. More on that later in 2. 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).

Instruction: If you have skipped specifying the interpreter in the "new project" section or also to proceed after the creation in order to install more packages, 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).



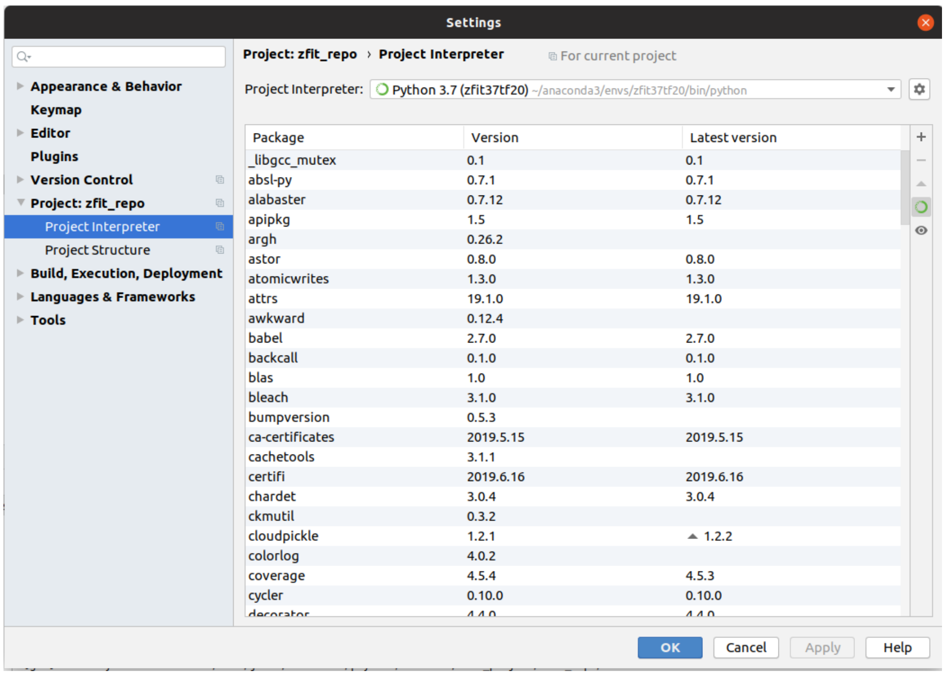
```
Python 3.7 /usr/bin/python3.7
Python 3.7 (phasespace37) ~/anaconda3/envs/phasespace37/bin/python
Python 3.7 (stat402_37) ~/anaconda3/envs/stat402_37/bin/python
Python 3.7 (zfit37) ~/anaconda3/envs/zfit37/bin/python
Python 3.7 (zfit37bench) ~/anaconda3/envs/zfit37bench/bin/python
Python 3.7 (zfit37phys) ~/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
```

1.3 Setting up the interpreter

Instruction: On the left, click on "Conda Environment". On the right, select (or keep selected) "New environment", Python version should be 3.9. Check if "Conda executable" has a path (such as /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.

`Explanation:` This created a virtual environment. This is an encapsulated collection of packages that are just seen if we "enter" the environment. Next, let's install packages into it.

Instruction: Now we assume you are in the setting menu

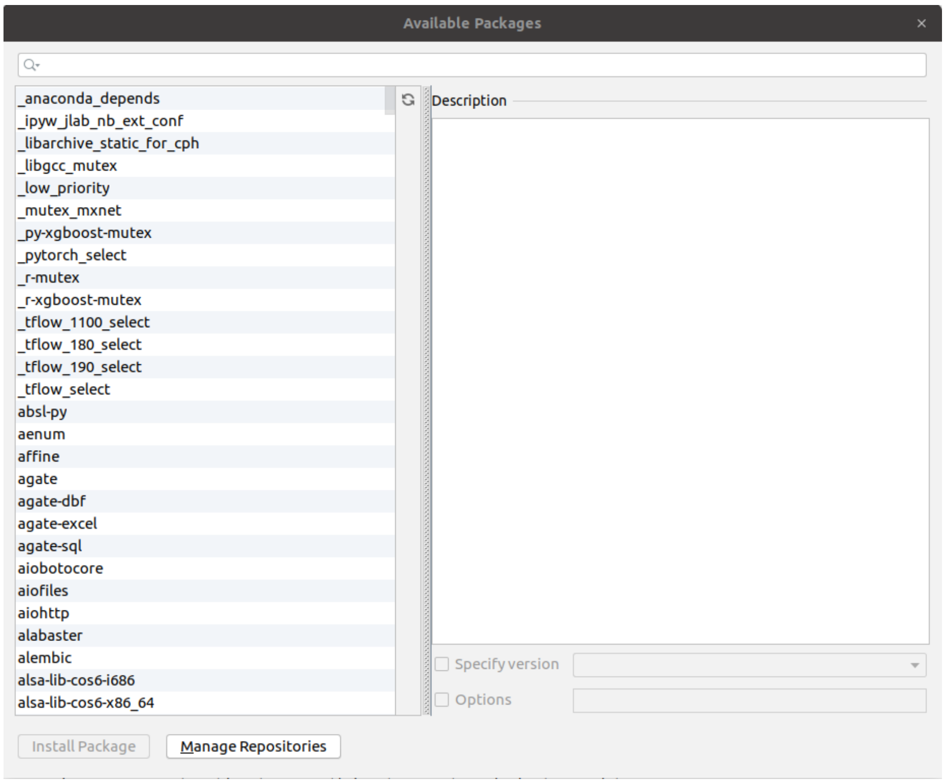


Project Interpreter: Python 3.7 (zfit37tf20) ~/anaconda3/envs/zfit37tf20/bin/python

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	
coverage	4.5.4	4.5.3
cycler	0.10.0	0.10.0
decorator	4.4.0	4.4.0

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

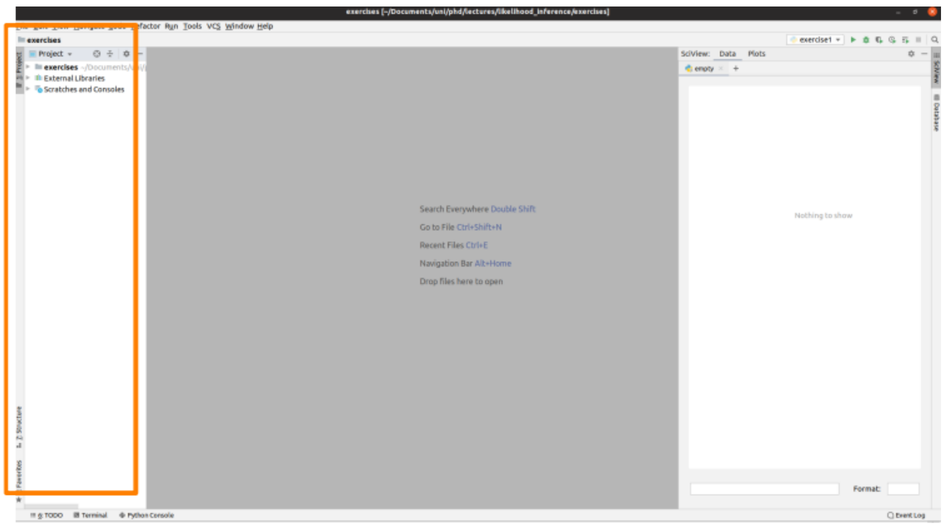
In case it does not work, you can also go to the anaconda navigator and install packages from there.



_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

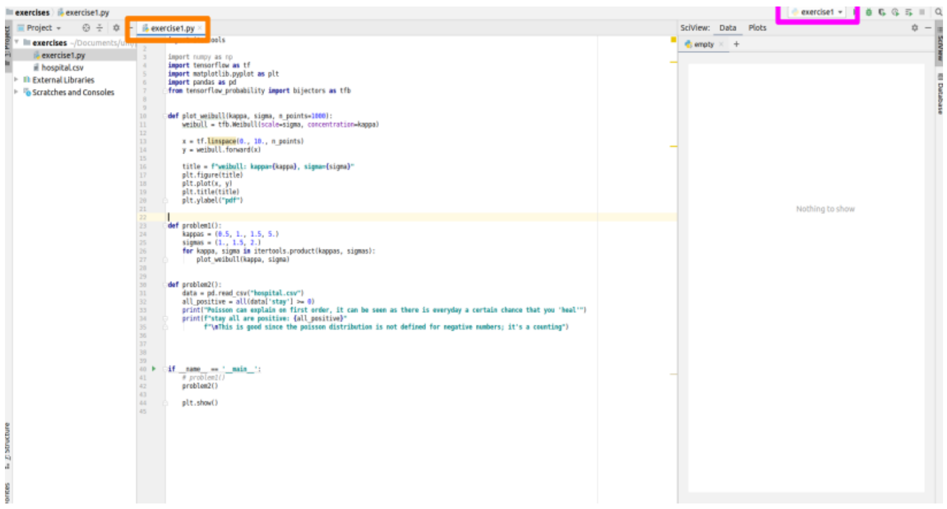
1.4 Creating and executing code

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



Instruction: 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). Now you can start to code (enter `print("hello world")`).

Instruction: Execute the file `for the first time`, (required, can also be executed later on like this), 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
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()
```

Explanation: When we ran the script, PyCharm created now a default "run configuration". This contains information about which Python version to use, where is the file located and more. Every other time from now on 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.

Instruction: Make sure that all of this works and tell the assistant about it.

2 Extra: What can PyCharm do?

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

- Ctrl + Q shows the docs of a function (you want to add this to your own functions? Checkout e.g. : `http://google.github.io/styleguide/pyguide.html#doc-function-raises` for an example)
- 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.
- Using the debugger (slightly 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

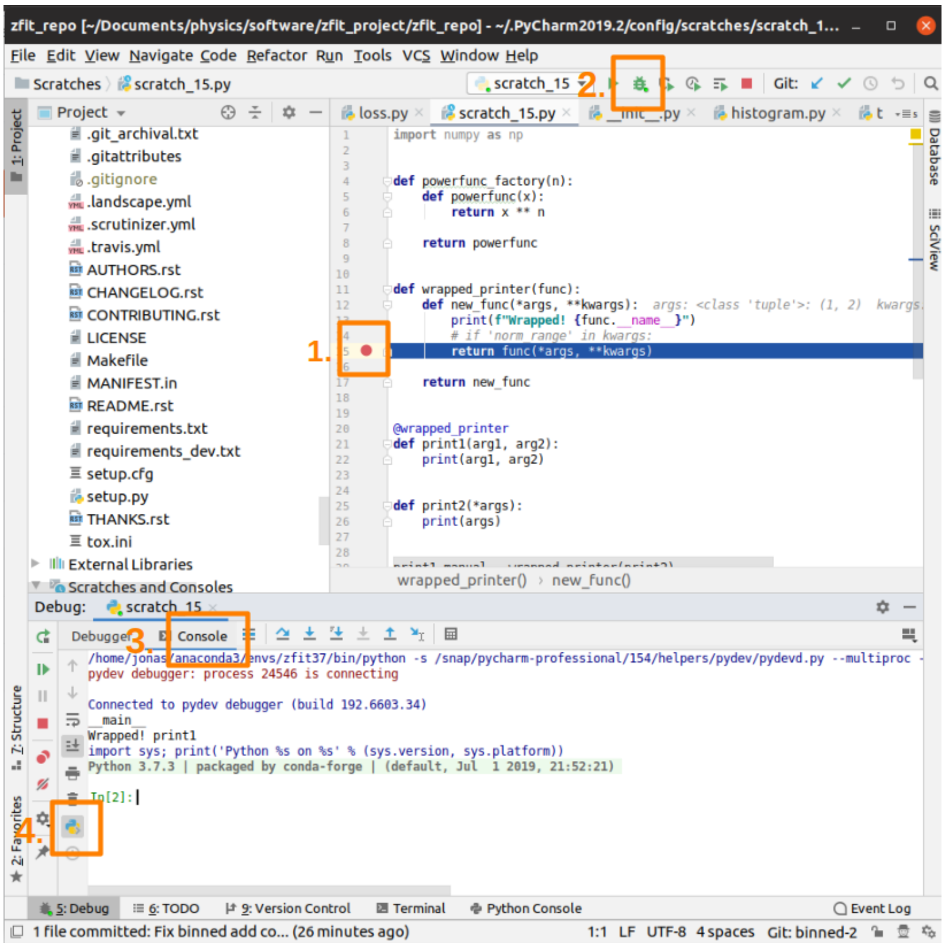
3 Extra: Using the debugger

This section can also be read at a later stage, as soon as you start using it for coding.

Instead of running the whole script, we can step through it. This is extremely helpful to understand (and therefore also debug) code.

To use the debugger, we need to run in "debug" mode and we need to tell until where it should run before we step in, as depicted in Fig. 3.

1. **Breakpoint:** We need to set a break point by clicking where the red dot is located in the image and a red dot should appear. Clicking again and it disappears. This is the line where the code execution will stop on (note that it only works on lines that have code, not empty ones).
2. **Run the debugger:** To run the script in debug mode, we can use the small bug (Käfer) symbol in the upper right. The code will run and stop at the (first) breakpoint encountered. A blue line will mark the current line (that is the line which will be executed next but was not yet.) Above our breakpoint, in grey and on the right of our code, PyCharm visually displays (as a help) the values of the variables.
3. **Debugger views:** The debugger offers us two views: the "Debugger" console that we are in by default shows an overview over the used variables (feel free to click around).


zfit_repo [~/Documents/physics/software/zfit_project/zfit_repo] - ~/.PyCharm2019.2/config/scratches/scratch_1...
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Scratches scratch_15.py
scratch_15
2.
Git:
Project
loss.py scratch_15.py histogram.py
1.
3.
4.
Debug: scratch_15
Debugger Console
5: Debug 6: TODO 9: Version Control Terminal Python Console Event Log
1 file committed: Fix binned add co... (26 minutes ago) 1:1 LF UTF-8 4 spaces Git: binned-2

```
import numpy as np


def powerfunc_factory(n):
    def powerfunc(x):
        return x ** n

    return powerfunc


def wrapped_printer(func):
    def new_func(*args, **kwargs):  args: <class 'tuple'>: (1, 2)  kwargs
        print(f"Wrapped! {func.__name__}")
        # if 'norm_range' in kwargs:
        return func(*args, **kwargs)

    return new_func


@wrapped_printer
def print1(arg1, arg2):
    print(arg1, arg2)


def print2(*args):
    print(args)
```

```
/home/jonas/anaconda3/envs/zfit37/bin/python -s /snap/pycharm-professional/154/helpers/pydev/pydevd.py --multiproc -
pydev debugger: process 24546 is connecting

Connected to pydev debugger (build 192.6603.34)
__main__
Wrapped! print1
import sys; print('Python %s on %s' % (sys.version, sys.platform))
Python 3.7.3 | packaged by conda-forge | (default, Jul  1 2019, 21:52:21)

In[2]: 
```

Figure 1: Using the debugger: 1) set at least one break-point; the execution of the code will stop at this line 2) start the script in debug mode 3) switch to 'Console' 4) activate the interactive Python shell by pressing the Python button. Now you can enter any command in the prompt (but be aware that if you change a value here, this is changed in the running code now). Typically, you want to print out something or checkout objects. To continue in the code, you can use the button on the right of 3) such as 'stepping to the next line'. You can also continue the execution to the next break-point by using the green 'resume' arrow on the left of 3).

Now click on the other tab, the "Console". This is now an interactive Python console which `takes place right at where the execution stopped.`

4. **Interactive console:** Here we can code anything. Inspect variables, evaluate expressions and more. Assigning to a variable here will make the variable like this in the running code!

To continue the code execution, there are a few options:

- Continue: The green arrow on the left, the "play button" continues the code until the next breakpoint is hit.
- Step over (default F8): To execute the current line and go to the next one, use the first blue arrow (just next to 3.)
- Step into (default F7): In order to step `inside` a function, use the arrow next to "step over". This will bring us "one stack down" into the function. If there is no function call, this is equivalent to "step over".