Workshop Environment Installation

For the installation process, we will be working from the terminal window: this is the **Terminal** application for Mac and Linux, and the **Anaconda command prompt** for Windows.

The following steps will guide you through this process.

Step 1: Download Anaconda

We will use Anaconda to automatically handle all the Python installation and management for us. Anaconda is a distribution of software that comes with **Conda**, **Python**, and many **scientific packages** that we will be using for data science.

To download Anaconda, go to anaconda.com/download and download the **Python 3.x version**. For specific instructions on how to install Anaconda for macOS, Windows, or Linux you can consult docs.continuum.io/anaconda/install.

> **For windows users**: We've seen that the newest version of Anaconda can create some issues on Windows machines. These issues are only restricted to anaconda and lead to the error popup message "python.exe - Entry Point Not Found" and that "pythoncom39.dll" cannot be found. To prevent this message and have anaconda and Jupyter running without any issues, we recommend to download a previous anaconda version (which is listed on this page) and can be downloaded directly via this link: Anaconda3-2021.05-Windows-x86_64.exe.

> **Miniconda instead of Anaconda**: When installing Anaconda, we automatically get Python, Conda, and 150 of the most popular Python packages for science, math and data analysis. If we want a lighter alternative to Anaconda for a faster installation, or we want to manage the package installation ourselves, we can instead use **Miniconda** which comes with only Python and Conda. Miniconda can be downloaded from conda.io/miniconda.html. If you choose to install Miniconda instead of Anaconda, make sure to download the Python 3.x version for your system.

Step 2 - Download the environment file

When starting a new Python project, it's always a good idea to create a **virtual environment** specifically for the project. We can think of an environment as an isolated copy of Python that maintains its own files and directories. This makes working on multiple projects much less troublesome by ensuring that each project is cleanly separated and there are no problems with dependencies between them.

The nice thing about Anaconda is that it comes with **Conda**: a package and **environment manager** that we can use to create environments for our projects. To create the Conda environment, we will use a `workshop.yml` environment file which contains the list of packages that you need for the course and workshop with their version number.

> This file tells Conda which libraries to download and most importantly which versions. Setting up your environment with this file will ensure that you are using the same versions as these course notes which is important for ensuring that you will have the same outputs. To get the list of installed libraries and their versions, you can take a look at the `.yml` file by opening it with a text editor.

Step 3 - Create the environment

We will now type a few commands to install the course environment using the `.yml` file.

- If you are on macOS and Linux: open a **Terminal** application.
- If you are on Windows: open the **Anaconda command prompt** from the Start menu.

If you correctly installed Anaconda or Miniconda in step 1, you should be able to list the current Conda environments available on your machine by typing the following command

```
conda env list
```

> **Troubleshooting**: If this command doesn't work for you, then it's likely that your terminal doesn't find `conda`. In that case, verify that you have Anaconda or Miniconda installed.

If you didn't previously install any Conda environment, it should only display the default one called **base**, or sometimes **root**, along with its installation path which may vary depending on your machine and whether you installed Anaconda or Miniconda.

```
# conda environments:
#
base                  *  C:\Users\user\anaconda3
```

Before installing the course environment, make sure you have an up-to-date version of `conda` by running

```
conda update conda
```

Note that you can always retrieve Conda's version with

```
conda -V
```

and then verify that the release is a recent one on this page: List of Conda releases

Let's now **create the environment** by running the following command.

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

Note that you need to be in the same folder as the `workshop.yml` file to run this command.

> **Troubleshooting**: If you cannot locate your `.yml` file from the terminal or the Anaconda prompt, use the File Explorer to find its absolute path. Open the explorer and go to the file (ex. in your Downloads folder), right click on it and open the details window. You should be able to copy/paste the absolute path from there and use it in the command:

```
# On Windows, it will look something like
conda env create -f C:\Users\your_username\Downloads\workshop.yml
# On macOS
conda env create -f /Users/your_username/Downloads/workshop.yml
```

After installing the course environment, you should see an `workshop` entry in the list of available environments - this is the course environment that we will use for the exercises and projects.

```
conda env list

# conda environments:
#
base                  *  C:\Users\user\anaconda3
workshop                 C:\Users\user\anaconda3\envs\workshop
```

Step 4 - Work with Jupyter notebooks within the course environment

Let's now try to run a few commands in Python using the course environment. First, in the terminal (mac, linux) or Anaconda command prompt (windows), **activate the course environment** by running

```
conda activate workshop
```

Let's now open the Jupyter lab and run a few Python commands. **After activating** the environment, run the following command in the terminal (mac, linux) or Anaconda command prompt (windows):

```
jupyter lab
```

You should see the application opening up in your default web browser. Let's select `Python [conda env:workshop]` under the Notebook tab. This will open a Notebook using the course environment. Notebooks are the main type of document that we will work with.

Now what you see at the very top of the notebook is what we call a cell. It represents an executable block of code. Let's type in the code to load and display the titanic data.

We can now run it by executing the cell. This can be done by pressing the play button from the menu or even simpler by pressing Shift+Enter (make sure that the cell is active first, meaning that your cursor is inside it).

```
import seaborn as sns
data = sns.load_dataset('titanic')
data.head(5)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	dec
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Na
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Na
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Na

Step 5 - Let's try a few libraries

```
import scipy
import scipy.stats as stats
from scipy.stats import norm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pymc as pm
```

You should have the same versions of the following libraries. **Note the last digit on the right can be different in your environment**:

```
pd.__version__
```

```
'1.5.3'
```

```
np.__version__
```

```
'1.24.4'
```

```
scipy.__version__
```

```
'1.9.3'
```

```
sns.__version__
```

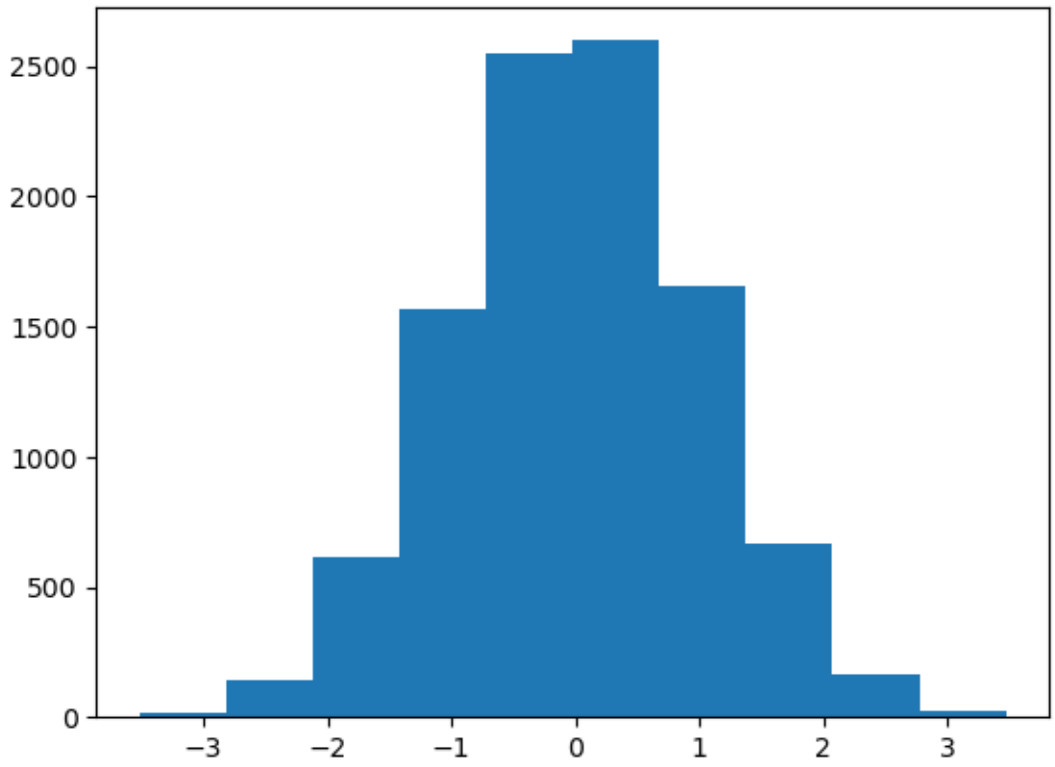
```
'0.11.2'
```

```
pm.__version__
```

```
'5.5.0'
```

The following codes should run without any issue:

```
# plot probability distribution
plt.hist(norm.rvs(0, 1, size = (10000,1)));
```




```
with pm.Model() as model:

    # Priors for unknown model parameters
    theta = pm.Beta("theta",
                    alpha = 10,
                    beta  = 10,
                    shape = 2)

    # Likelihood (sampling distribution) of observations
    obs = pm.Binomial("y",
                      n        = [10,10],
                      observed = [2,2],
                      p        = theta,
                      shape    = 2)
```

```
    # Difference between variants
    relative_uplift = pm.Deterministic("uplift",
                                       theta[1] / theta[0] - 1)

    # Draw samples from the prior
    trace = pm.sample(draws=10)
```

```
Only 10 samples in chain.
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [theta]
Sampling 4 chains for 1_000 tune and 10 draw iterations (4_000 + 40 draws total) took 19 seco
The number of samples is too small to check convergence reliably.
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
trace
```

```
Inference data with groups:
    > posterior
    > sample_stats
    > observed_data
```