CEPC Analysis Tutorial

Environment Setup

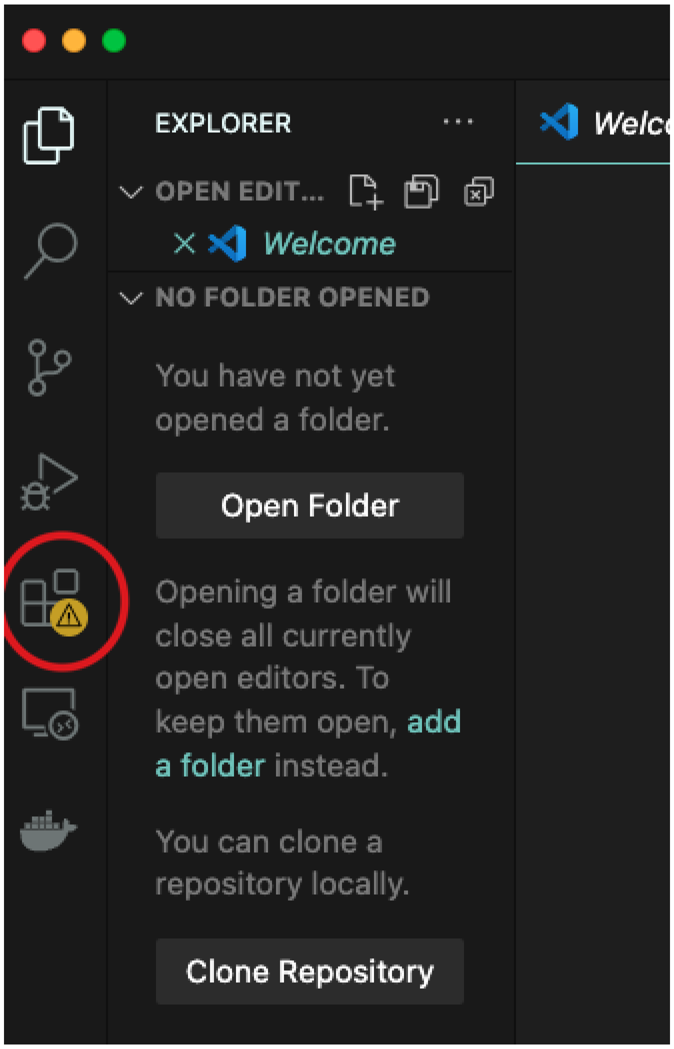
Vscode + Remote-ssh + Anaconda + uproot/Root/Awkward

Install Vscode on your local computer

Download and install Vscode: https://code.visualstudio.com/Download

Install the related plugins

Search the plugins you want to install through the following side navigation



You have not yet opened a folder.

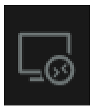
Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

Install the following plugins:

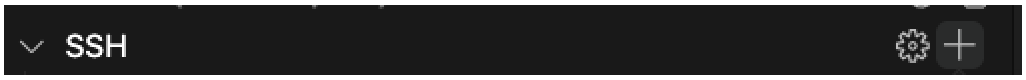
```
Remote-ssh, Jupyter
```

Once it's done, reopen the Vscode and you should see the following navigation

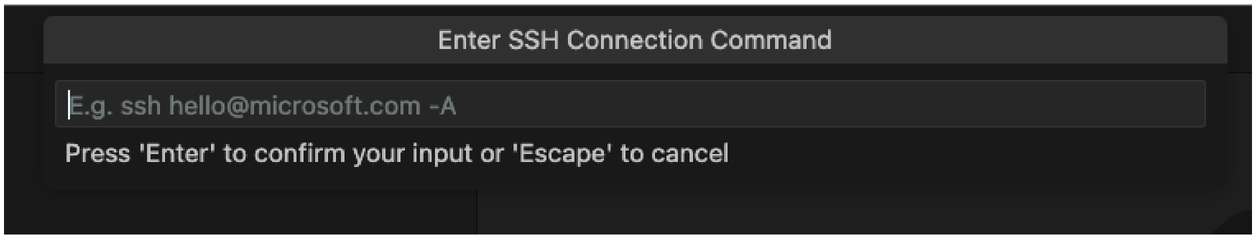


Setup your lxlogin account

Add your lxlogin account through the Remote Explorer:



Once you click “new remote”, you should see:

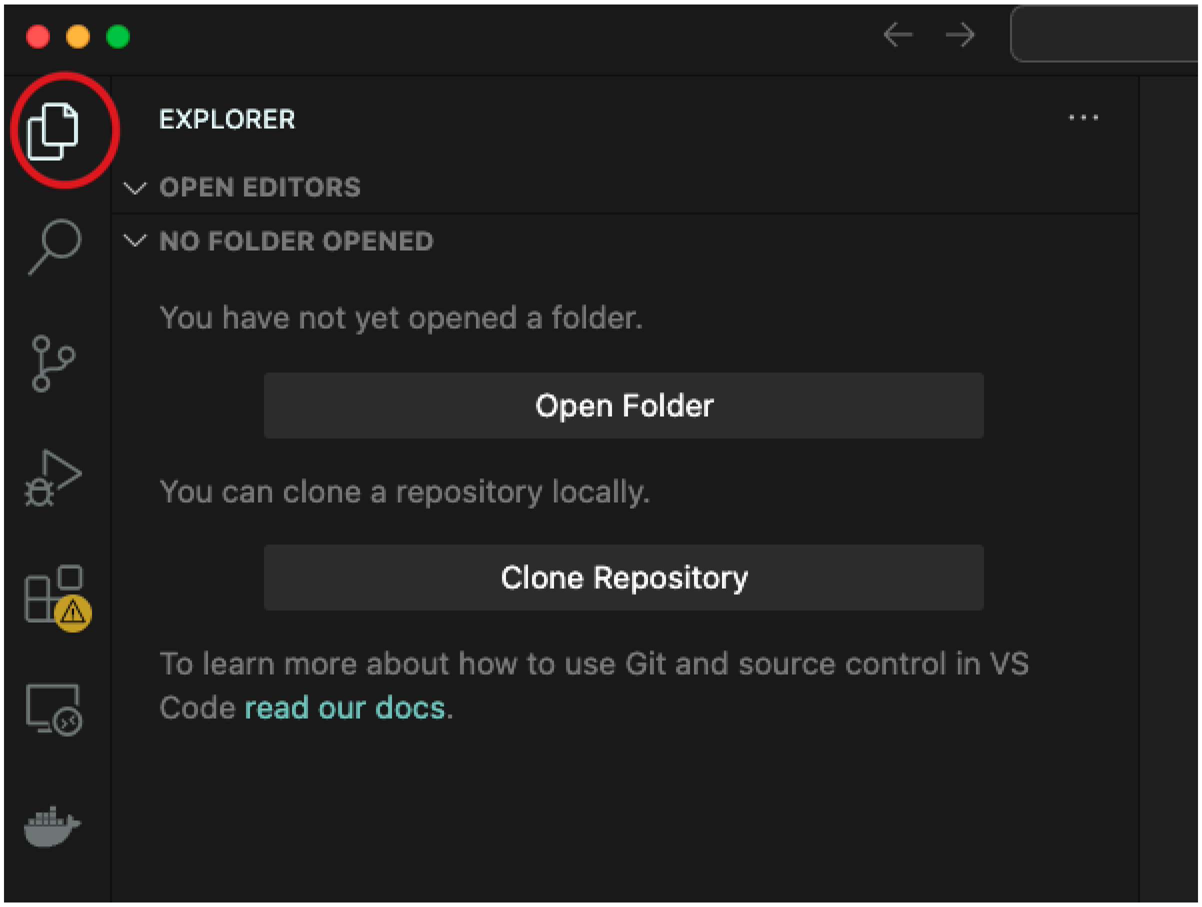

Enter SSH Connection Command

E.g. ssh hello@microsoft.com -A

Press 'Enter' to confirm your input or 'Escape' to cancel

Log in your lxlogin account through “ssh -Y username@login.ihep.ac.cn”

Click “Open Folder”, choose your base folder and you should see all the files located in this folder (e.g. /publicfs/cms/user/{your name}).



You have not yet opened a folder.

You can clone a repository locally.

To learn more about how to use Git and source control in VS Code read our docs.

> You could also access the bash terminal through “Command + j”

Install Anaconda into your lxlogin workplace

Download the Anaconda package to your workplace, the version of the package can be found: https://repo.anaconda.com/archive/

Select the correct version you want and download it through

```
wget https://repo.anaconda.com/archive/Anaconda3-2023.09-0-Linu
```

Install Anaconda through following commends:

```
chmod +x Anaconda3-2023.09-0-Linux-x86_64.sh
./Anaconda3-2023.09-0-Linux-x86_64.sh
```

If you install it successfully, the command “conda -V” will show you the version of the anaconda you installed.

Install the virtual analysis environment

```
conda create -n myenv_name python=3.8
conda activate myenv_name
```

Then you will go into your analysis environment, remember to reactivate your env after you login your IHEP account (myenv_name can be any string you like, I will use “test” for teaching reasons)

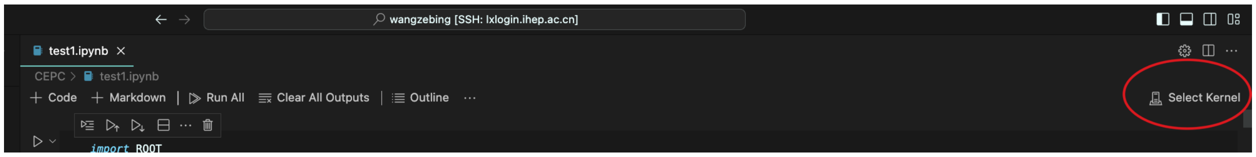
Install the analysis required package:

```
conda activate myenv_name

conda install mamba

mamba install root
mamba install uproot
mamba install pandas
mamba install matplotlib
mamba install mplhep
pip install awkward-pandas
```

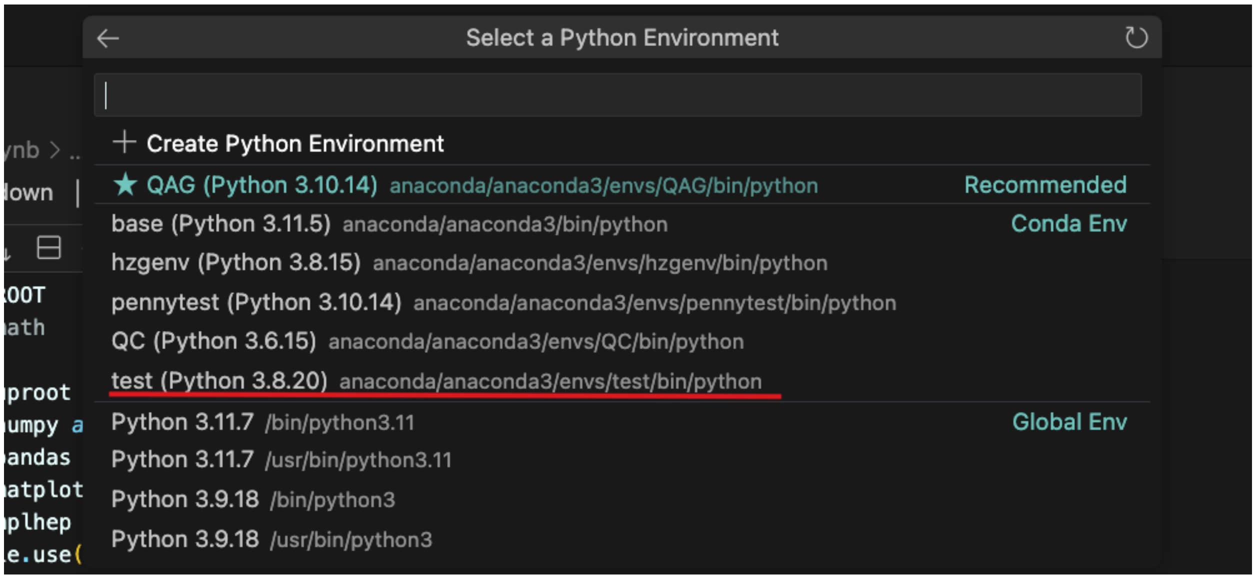
Once it’s done, create a jupyter notebook and test the installation:

```
touch test.ipynb
```

Select the python env as your jupyter kernel.



```
import ROOT
```



Select a Python Environment

+ Create Python Environment

★ QAG (Python 3.10.14) anaconda/anaconda3/envs/QAG/bin/python Recommended

base (Python 3.11.5) anaconda/anaconda3/bin/python Conda Env

hzgenv (Python 3.8.15) anaconda/anaconda3/envs/hzgenv/bin/python

pennytest (Python 3.10.14) anaconda/anaconda3/envs/pennytest/bin/python

QC (Python 3.6.15) anaconda/anaconda3/envs/QC/bin/python

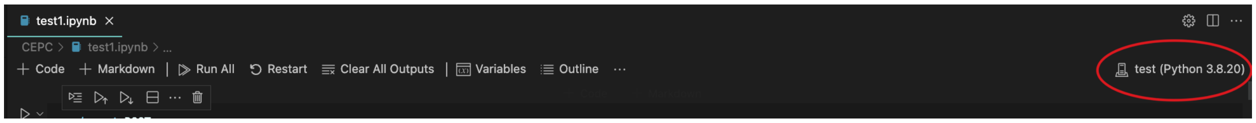
test (Python 3.8.20) anaconda/anaconda3/envs/test/bin/python

Python 3.11.7 /bin/python3.11 Global Env

Python 3.11.7 /usr/bin/python3.11

Python 3.9.18 /bin/python3

Python 3.9.18 /usr/bin/python3



if you can’t find the correct kernel, follow the below tutorial to add your kernel:

https://www.cnblogs.com/chester-cs/p/14653149.html

Run the following commands to test:

```
import ROOT
import math

import uproot
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import mplhep as hep
plt.style.use(hep.style.CMS)
```

```
import awkward as ak
```

If it’s successful running, congratulations! You are ready for the data analysis.

Awkward tutorial

What is Awkward Array for? How does it compare to other libraries?

Python's builtin lists, dicts, and classes can be used to analyze arbitrary data structures, but at a cost in speed and memory. Therefore, they can't be used (easily) with large datasets.

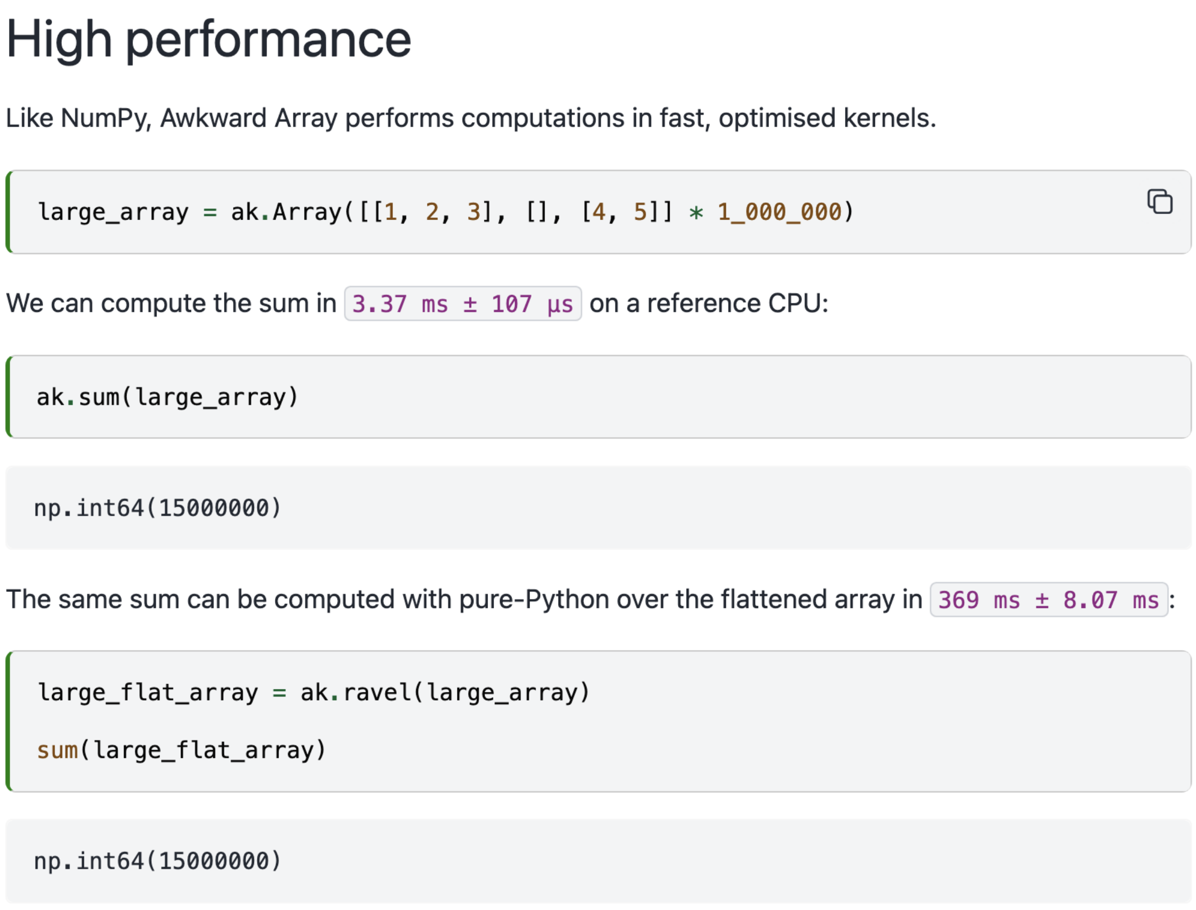
Pandas DataFrames (as well as Polars, cuDF, and Dask DataFrame) are well-suited to tabular data, including tables with relational indexes, but not arbitrary data structures. If a DataFrame is filled with Python's builtin types, then it offers no speed or memory advantage over Python itself.

NumPy is ideal for rectangular arrays of numbers, but not arbitrary data structures. If a NumPy array is filled with Python's builtin types, then it offers no speed or memory advantage over Python itself.

Apache Arrow (pyarrow) manages arrays of arbitrary data structures (including those in Polars, cuDF, and to some extent, Pandas), with great language interoperability and interprocess communication, but without manipulation functions oriented toward data analysts.

Awkward Array is a data analyst-friendly extension of NumPy-like idioms for arbitrary data structures. It is intended to be used interchangeably with NumPy and share data with Arrow and DataFrames. Like NumPy, it simplifies and accelerates computations that transform arrays into arrays—all computations over elements in an array are compiled. Also like NumPy, imperative-style computations can be accelerated with Numba.

Note that there is also a ragged array library with simpler (but still non-rectangular) data types that more closely adheres to array APIs.



High performance

Like NumPy, Awkward Array performs computations in fast, optimised kernels.

```
large_array = ak.Array([[1, 2, 3], [], [4, 5]] * 1_000_000)
```

We can compute the sum in `3.37 ms ± 107 μs` on a reference CPU:

```
ak.sum(large_array)
```

```
np.int64(15000000)
```

The same sum can be computed with pure-Python over the flattened array in `369 ms ± 8.07 ms`:

```
large_flat_array = ak.ravel(large_array)

sum(large_flat_array)
```

```
np.int64(15000000)
```

https://awkward-array.org/doc/main/getting-started/index.html