

Hoffman2 Happy Hour: R and Python

Charles Peterson

👏👏 Welcome! 💡

In this workshop, we will go over how to use R and Python on Hoffman2

We will explore:

- Using R and Python
- Installing Packages
- Basic Anaconda
- Jupyter and RStudio
- ❤️ Suggestions and Feedback, Email
cpeterson@oarc.ucla.edu







Access the Workshop Files

This presentation and accompanying materials are available on  [UCLA OARC GitHub Repository](#)

You can view the slides in:

-  PDF format - Python-R.pdf
-  HTML format: [Workshop Slides](#)

Each file provides detailed instructions and examples on the various topics covered in this workshop.

Note:  This presentation was built using [Quarto](#) and RStudio.



Running Python on Hoffman2

Python

Hoffman2 supports running  Python applications.

It is **HIGHLY** recommended to use a version of python that has been built and tested by Hoffman2 staff.

These versions of python can be accessed by using `module load` commands. This means that using system python builds (i.e. `/usr/bin/python`) are NOT recommended

- See all versions of Python installed on Hoffman2

```
1 modules_lookup -m python
```

- Load a Python module

```
1 module load python/3.7.3
2 which python3
```

- This example shown:
 - Python version 3.7.3
 - Location of python
 - `/u/local/apps/python/3.7.3/gcc-4.8.5/bin/python3`
 - Location of the Hoffman2 installed python

Running Python on Hoffman2

Once you load the Python module, you can run Python with either an interactive (qssh) or Batch job (qsub)

Interactive session

Get access to a compute node and load python module

```
1 qssh -l h_data=10G
2 module load python/3.7.3
```

Run Python

- Start a python session
- Run a python single command
- Run a Python file

```
1 python3
```

```
1 python3 -c "a = 1 + 3 ; print a"
```

```
1 python3 foo.py
```



Note

If you are using Hoffman2 build python, like most python builds from source, when using Python version 3.x, the command is `python3` while version 2.7 is `python`

Running Python

Batch job

You can create a job script to run python as a Batch job

For example, we create a job script, named `foopy.job`

```
1 #!/bin/bash
2 #$ -cwd
3 #$ -o joblog.$JOB_ID
4 #$ -j y
5 #$ -l h_rt=1:00:00,h_data=10G
6 #$ -pe shared 1
7
8 # load the job environment:
9 . /u/local/Modules/default/init/modules.sh
10 module load python/3.7.3
11
12 python3 foo.py
```

Then we will submit this job

```
1 qsub foopy.job
```



Note

See that this job script used one core `-pe shared 1`.

You can request more cores with modifying `-pe`, but your Python code **MUST** also be modified for your Python code to actually use multiple.

See our workshop on [Big Data](#) for some ways to parallelize your Python code.

Python Package Installation

The builds of Python on Hoffman2 ONLY have the basic compiler/interpreter and a few basic packages.

- Most python application will need to installed by the user
 - Pandas, SciPy, Scikit-learn, and TensorFlow, etc.
- Hoffman2 staff do NOT install packages into the python builds that are supported.
 - This can cause conflicts with different packages

User installed packages

- Users are also not able to install packages in the main Python build directories.
 - This is because of packages of different versions can cause conflicts with other package dependencies, causing Python to break and unusable.
- While users do not install packages into the main Python build directories, users may install packages into their own directories
 - \$HOME, \$SCRATCH, or any Project directories.
- There are a few ways a user can accomplish this.
 - Using pip package manager
 - Using Python Virtual Environments
 - Using anaconda

Using pip package manager

If you would like to use the package, scikit-learn, you can install this package via the pip (PyPI) package manager.

```
1 module load python/3.7.3
2 pip3 install scikit-learn --user
```

You will notice the `--user` flag on pip3. It will ensure that you will install the package in your \$HOME directory.

Normally, pip will try to install in the main Python build, in which, users do not have write access and you may see errors when building your package. When using `--user` by default, it will install it in \$HOME/.local.

Installing with pip

If you want to install this package in another directory, you can install this package in any directory you have write access.

For example, if you want to install this package in `$SCRATCH/python-pkg`,

```
1 module load python/3.7.3
2 pip3 install scikit-learn -t $SCRATCH/python-pkg
3 export PYTHONPATH=$SCRATCH/python-pkg:$PYTHONPATH
```

When running your jobs, you will have to make sure that the variable `$PYTHONPATH` is updated to have any custom locations of your packages.

Warning

Users will need be aware when they are installing multiple packages in `$HOME/.local` with `pip3 --user`.

Installing conflicting packages with different versions of python will let to errors.

When working on different projects, it is best to use Virtual Environment or Anaconda to install python packages and dependencies.

Python Virtual Environment

One great way to manage your Python packages is using Python's **Virtual Environment** feature. This will allow users to install and manage Python inside an environment that they control and is located in their own directories. It is an isolated runtime environment so user can set unique versions and dependencies of applications.

This is good when you are running multiple applications that requires different application environments.

Creating a Virtual Environment

To create a environment, you will first load the base 🐍 Python version on Hoffman2 you want to use. For example, if you want to use python/3.7.3:

```
1 module load python/3.7.3
```

Then create a directory location you want to install the env.

```
1 mkdir -pv $HOME/myenv
2 python3 -m venv $HOME/myenv/mypython3.7.3
```

This will create an env named mypython3.7.3. Once it is created, you can activate the env by:

```
1 source $HOME/myenv/mypython3.7.3/bin/activate
```

This is a bash shell script that will active your new python environment. From here, you can managing packages with pip or and other installtion method.

Virtual Environment

Note

Since this is a new, custom python env that you created, you don't need to add `--user` flags in pip. All new libraries and packages will be installed by default inside the new environment (`$HOME/myenv/mypython3.7.3`)

More information about using  Python Virtual Environments can be found [here](#)

Tip

For python version 3.x, the commands `python` and `python3` are the same in virtual environments

You can run `which python` and `which python3` to make sure you are running the python in the correct location

R on Hoffman2

Setting up R

Hoffman2 supports running R applications.

You can see all the available version of R on Hoffman2 by running:

```
1 modules_lookup -m R
```

Then you can load R

```
1 module load R/4.2.2
```

In order to use the correct build of R, they will need to have gcc or intel modules loaded first, based on what was shown by [modules_lookup](#). This will ensure the gcc and intel libraries and the correct version is correctly loaded.

Running R on Hoffman2

Once you load the R module, you can run R with either an interactive (qrsh) or Batch job (qsub)

Interactive session

Get access to a compute node

```
1 qrsh -l h_data=10G
```

Load python module

```
1 module load R/4.2.2
```

Run R

- Start a R session
- Run a R single command
- Run a R file

```
1 R
```

```
1 R -e "a = 1 +3 ; print
```

```
1 R CMD BATCH fooR.R  
2 Rscript fooR.R
```

Running R

Batch job

You can create a job script to run R as a Batch job

For example, we create a job script, named `fooR.job`

```
1 #!/bin/bash
2 #$ -cwd
3 #$ -o joblog.$JOB_ID
4 #$ -j y
5 #$ -l h_rt=1:00:00,h_data=10G
6 #$ -pe shared 1
7
8 # load the job environment:
9 . /u/local/Modules/default/init/modules.sh
10
11 module load R/4.2.2
12
13 R CMD BATCH fooR.R
```

Then we will submit this job

```
1 qsub fooR.job
```


Like the python job before, this request one core.

There are ways to parallelize your R code, like [snow](#), [sparkR](#), [parallel](#)

R package installation

When installing R packages, a common way is to use

```
1 install.packages('PKG_name')
```

Typically, when you run R, it will install the new packages in the main R global directory. Though, on Hoffman2 (and other  HPC resources), you will not be able to modify this directory.

Example install:

```
1 install.packages("dplyr")
```

- R will prompt you with a new path that will be located in your \$HOME directory. This directory path is determined by the \$R_LIBS_USER.
- Each R module on Hoffman2 has a unique \$R_LIBS_USER to avoid conflicts when using different versions of R.
- R may also ask you to select a CRAN mirror. You can choose [1](#) to use the CRAN https cloud server

Common Errors

Sometimes R can error during the installing process. R can output a lot of output during the `install.packages()` step.

This output can be very long, though, looking through this output can give you clues to what you need to do to fix it.

Look for lines that look like C/C++ errors, lines that have 'ERROR, or 'no such file or directory'.

For example, the R package 'glmnet' will error when using the R/4.2.2 module with no gcc module, since it will use the default gcc 4.8.5 version that is too old for the 'glmnet' package.

The message will say "Error: C++17 standard requested but CXX17 is not defined". Where you will need a version of gcc that uses at least a C++17 standard.

To fix this, you can use R/4.2.2 with gcc version 10.2.0

```
1 module load gcc/10.2.0
2 module load R/4.2.2
```

External packages

When installing packages, you may need to add external software and libraries to successfully install the package.

For example, if you need to install the `nloptr` package, you will need to have the `nlopt` libraries already installed and loaded in your shell session. On Hoffman2, we already have this library and you will need to load the module before you try to install the package

If you run

```
1 install.packages("nloptr")
```

You may see an error line

```
1 g++: error: nlopt/lib/libnlopt.a: No such file or directory
```

To fix this, you will load the `nlopt` module with R

```
1 module load nlopt
2 module load gcc/10.2.0
3 module load R/4.2.2
```


R package location

Sometimes you will want to use a different directory location to install your packages. Maybe you are running low in your \$HOME directory so you have group space with large storage space. You can update `.libPaths()` to have a directory location to install and find your R packages

```
1 # Assign the current library paths to a variable
2 current_paths <- .libPaths()
3
4 # Append the new directory to the library paths
5 new_library_path <- "/path/to/new/library"
6 updated_paths <- c(current_paths, new_library_path)
7
8 # Update the library search paths
9 .libPaths(updated_paths)
```

If you decided to use this new directory to store your R packages, make sure that you have all your R scripts updated to find this directory everytime you run R.

Using Anaconda

Anaconda

Anaconda is a very popular Python and R distribution. This is a great option for simplifying package management and pipelines. Hoffman2 does have Anaconda installed, from which, the user can create their own conda environments.

TO create a environment, you will first need to load the Anaconda module

```
1 module load anaconda3/2020.11
```

Warning

There is NO need to load other python or R module. Your anaconda environment will have a build of python and/or R so adding other modules may have conflicts.

Note

There is NO need to run conda init. Anaconda is environment is already load when you run the conda.sh script. When you run conda init, it will modify your .bashrc file and possibly break your created env.

Creating a Conda Environment

After you load anaconda, you will now create your new environment

```
1 conda create --name mypython3.7 python=3.7
2 conda activate mypython3.7
```

This will create a python env, version 3.7, with the name mypython3.7. Then it will activate the new environment. From here you can install any package you want with conda install or pip install. Do NOT use `--user` with pip in this case. You want the package to install in the conda env, not in `$HOME/.local`

```
1 conda activate mypython3.7
2 conda install scipy=0.15.0
3 pip install matplotlib
```

By default, conda envs are installed in `$HOME/.conda` directories. You can create env in other directories by

```
1 conda create -p $SCRATCH/mypython python=3.7
2 conda activate $SCRATCH/mypython
```

This will install the conda env in a custom location. You can install your conda env in a shared PROJECT directory so they can be shared with other users.

For more information, we had done a workshop on using [Anaconda on Hoffman2](#) that you can review.

Jupyter and Rstudio



Jupyter on Hoffman2

Jupyter can be executed on Hoffman2. This process involves running the `jupyter notebook/lab` command on a Hoffman2 node, forwarding the Jupyter port to your local machine, and accessing Jupyter via your local web browser.

However, for a more streamlined experience, we have created a script, `h2jupynb`, which automatically creates a Jupyter session on a Hoffman2 compute node.

- Download `h2jupynb`

```
1 wget https://raw.githubusercontent.com/rdauria/jupyter-notebook/main/h2jupynb
2 chmod u+x h2jupynb
```

- Run `jupyter`

```
1 python h2jupynb -u joebruin -t 5 -m 10
```

More information can be found on [our website](#)

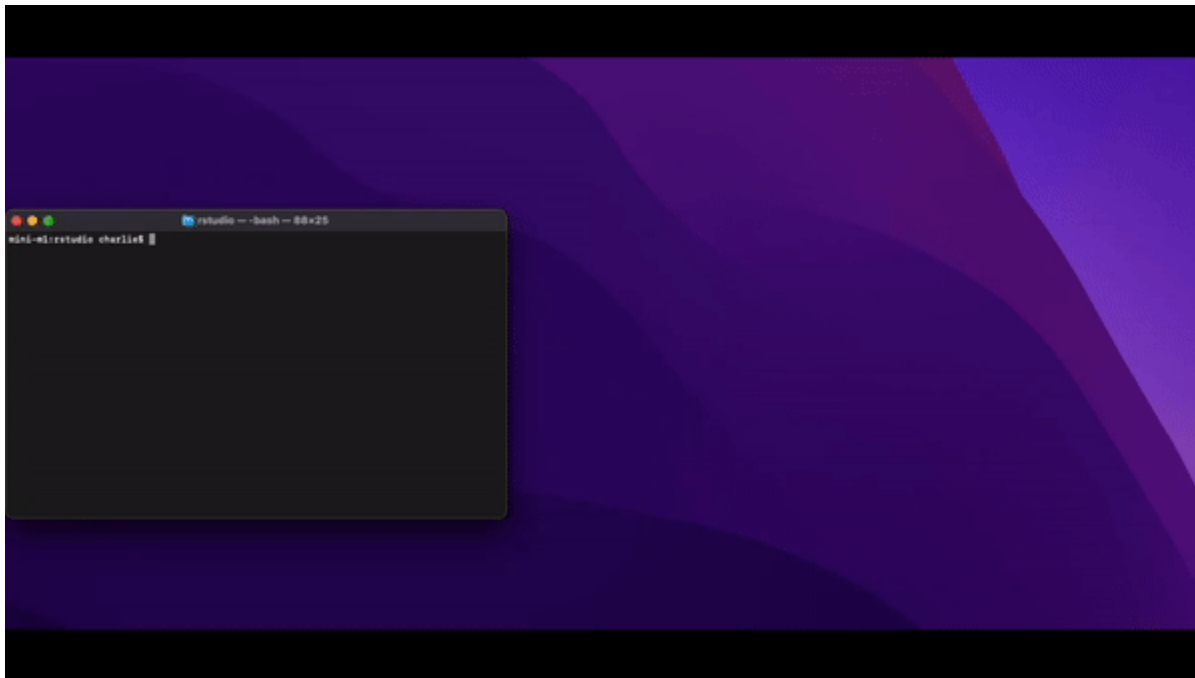


RStudio on Hoffman2

You can conveniently interact with Hoffman2 using RStudio Server, allowing you to utilize R in a familiar, intuitive environment.

Find detailed instructions for using RStudio on Hoffman2 on our [GitHub page](#)

We've also conducted a dedicated workshop on using [Rstudio on Hoffman2](#). Feel free to explore this resource for additional insights and usage tips.





Thanks for Joining!



Questions? Comments?

- cpeterson@oarc.ucla.edu
- Look at for more Hoffman2 workshops at <https://idre.ucla.edu/calendar>
- Fill our [assessment form](#)



