Introduction to Bioinformatics Summer Series

Amy Stonelake Ph.D., Peter Fitzgerald Ph.D., Alexandra L Emmons Ph.D., & Joe Wu Ph.D.
BTEP/GAU/CCR/NCI/NIH - email ncibtep@mail.nih.gov
Bioinformatics Training and Education Program

Table of Contents

Lesson 5

Lesson 6

Course Overview

Course Description

A series of 6 stand-alone lessons introducing learners to various aspects of bioinformatics. Lessons will be held on Tuesdays at 1 PM in June and July 2023. You can attend any class, just one class, or all classes. Lessons will not be hands on, and thus, no software installation is necessary.

Lesson Topics

1. **Introduction to Bioinformatics Resources at NCI** (06/13, Instructor Amy Stonelake, PhD (BTEP))

   In this lesson, attendees will be introduced to the different bioinformatics resources available to them at NCI. This includes:

   1. Bioinformatics Training Classes and Events
   2. Online Learning Platforms Licenses (Coursera and Dataquest)
   3. Using the NIH High Performance Compute Cluster Biowulf
   4. Next-Gen Sequencing software purchased by the Office of Science Technology and Resources (OSTR)
   5. Available Cloud Resources

2. **Central Dogma of Molecular Biology: Analyzing DNA, RNA, and Proteins** (Rescheduled 06/29, Instructor Amy Stonelake, PhD (BTEP))

   Starting with the classic Central Dogma of Molecular Biology, we will look at how each of the components (DNA, RNA, protein) is measured and analyzed. Next-Gen Sequencing (NGS) techniques, analysis tools available, and some history of how it all started with the Human Genome Project will be discussed.

3. **Keeping your Data FAIR: Organizing, Managing, and Sharing your Data** (06/27, Instructor Peter FitzGerald, PhD (GAU))

   Participants will learn about FAIR principles (findability, accessibility, interoperability, and reusability) and how they apply to scientific data. Given the NIH Data Sharing Policy for intramural scientists, methods for organizing, managing, and sharing data in bioinformatics projects will be discussed.

4. **Introduction to High Performance Computing at NIH: Biowulf** (07/11, Instructor Alex Emmons, PhD (BTEP))

Attendees will learn about high performance computing (HPC) with the NIH cluster Biowulf. This resource contains hundreds of open source bioinformatics tools and biological databases. Participants will understand why it's important to be able to work on an HPC cluster for Next-Gen Sequencing data analyses.

5. **Introduction to R and Python Programming Languages** (07/18, Instructor Alex Emmons, PhD (BTEP))

   In this class, participants will learn about the R and Python programming languages, and how each is used in bioinformatics research. The advantages of each language will be discussed, and how to choose which is most applicable to your data analyses. Learning resources for beginners will be provided and questions answered.

6. **Managing Bioinformatics Projects with Jupyter Notebook** (07/25, Instructor Amy Stonelake, PhD (BTEP))

   Participants will learn how Jupyter Lab Notebooks can be used to organize, manage, and share their bioinformatics analyses projects. The instructor will demonstrate how to install, launch, and interact with Jupyter Lab Notebooks. This will include managing code, data, and visualizations, which are kept all in one place within the Notebook. A great class for those starting a new bioinformatics project.

Classes will be recorded and made available within 48 hours of the event on the BTEP Video Archive *(https://bioinformatics.ccr.cancer.gov/btep/btep-video-archive-of-past-classes/)*.

Lesson 1

Introduction to Bioinformatics Resources

Lesson 2

The Central Dogma: Analyzing DNA, RNA, and Proteins

Lesson 3

Keeping your Data FAIR: Organizing, Managing, and Sharing your Data

Lesson 4

Introduction to High Performance Computing at NIH: Biowulf

Learning Objectives

1. Understand the components of an HPC system. How does this compare to your local desktop?
2. Learn about Biowulf, the NIH HPC cluster.
3. Learn about the command line interface and resources for learning.

What is a high performance cluster (HPC)?

> A collection of standalone computers that are networked together. They will frequently have software installed that allow the coordinated running of other software across all of these computers. This allows these networked computers to work together to accomplish computing tasks faster. --- hpc-intro (Software carpentries) *(https://carpentries-incubator.github.io/hpc-intro/files/jargon.html#p6)*

When using an HPC

- We use a command line interface and a Secure shell protocol (SSH) to establish a remote connection to the login node / head node
    - HPCs are remote resources that require connections using slow or intermitten interfaces (over WIFI and VPNs). It is more practical to guide functionality over the command line using plain text. *(https://carpentries-incubator.github.io/hpc-intro/11-connecting/index.html)* Most of us are likely used to a graphical user interface (GUI), which is a point-and-click interface, so we will describe how the CLI differs a bit later.
- the cluster head node distributes compute tasks using a scheduling system (e.g., SLURM).

Note

Slurm stands for Simple Linux Utility for Resource Management.

What is Biowulf?

Biowulf is the high performance computing (HPC) system at NIH.

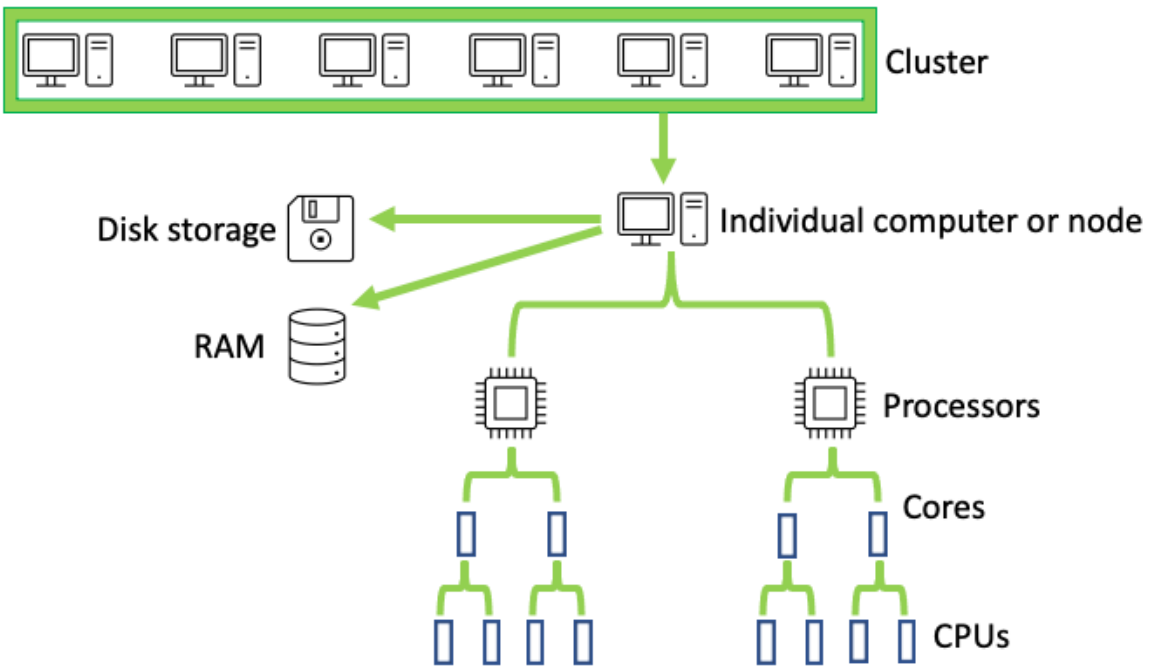
- The NIH high-performance compute cluster is known as “Biowulf”
- It is a 95,000+ processor Linux cluster
- Can perform large numbers of simultaneous jobs
- Jobs can be split among several nodes
- Scientific software *(https://hpc.nih.gov/apps/)* (600+) and databases *(https://hpc.nih.gov/apps/db.php)* are already installed
- Can only be accessed on NIH campus or via VPN

When should we use Biowulf?

You should use Biowulf when:

- Software is unavailable or difficult to install on your local computer and is available on Biowulf.
- You are working with large amounts of data that can be parallelized to shorten computational time AND/OR
- You are performing computational tasks that are memory intensive.

Example of High Performance Computing Structure



Essentially Biowulf is a scaled up version of your local computer.

In Biowulf, many computers make up a cluster. Each individual computer or node has disk space for storage and random access memory (RAM) for running tasks. The individual computer is composed of processors, which are further divided into cores, and cores are divided into CPUs.

Info

Information on the NIH HPC architecture and hardware here *(https://hpc.nih.gov/systems/hardware.html)*.

Node types on the HPC

- login node (head node)
  - Used for submitting resource intensive tasks as jobs
  - Editing and compiling code
  - File management and data transfers on a small scale
- compute nodes (worker nodes)
  - For computational processes
  - Requires interaction with a job scheduling system (SLURM)
  - Batch jobs, sinteractive sessions
- Data transfer node (For Biowulf, this is Helix.)

Info

`sinteractive` - work on biowulf compute nodes interactively; suitable for testing/debugging cpu-intensive code, Pre/post-processing of data, and using graphical applications.
`sbatch` - for submitting shell scripts via jobs, taking away any interactive component.
`swarm` - used for runnning embarassingly parallel code as independent jobs.

The Data transfer node: Helix

- Used for data transfers and file management on a large scale.
- 48 core system with 1.5 TB of main memory
- direct internet connection
- Helix should be used when
  - you are transferring >100 GB using `scp`
  - gzipping a directory containing >5K files, or > 50 GB
  - copying > 150 GB of data from one directory to another.
  - uploading or downloading data from the cloud.
- For more information on data transfers see hpc.nih.gov *(https://hpc.nih.gov/docs/transfer.html)*.

Biowulf Data Storage

Summary of file storage options

	Location	Creation	Backups	Space	Available from
/home	network (NFS)	with Biowulf/Helix account	yes	16 GB default quota	B,C,H
/lscratch (nodes)	local	created by user job	no	~850 GB shared	C
/scratch	network (NFS)	created by user	no	100 TB shared	B,H
/data	network (GPFS/NFS)	with Biowulf/Helix account	no	100 GB default quota	B,C,H

H = helix, B = biowulf login node, C = biowulf compute nodes

- You may request more space on `/data`, but this requires a legitimate justification.
- More information on data storage here *(https://hpc.nih.gov/storage/)*.

Important

Data storage on the HPC system should not be for archival purposes.

Note

Though there aren't true back-ups of your data directories, there are snapshots with a view of your home and data directories at a specific point in time. You can learn more about snapshots *(https://hpc.nih.gov/storage/backups.html)* in the HPC documentation.

Applications on Biowulf

- Bioinformatics applications and other programs are available on Biowulf via **modules**.
- View a list of available applications here *(https://hpc.nih.gov/apps/)*.

Info

Loading software as environment modules allows us to better control our computational environment and easily use a large number of programs and even different versions of the same programs. Modules alter the user's environment varibables such as the executaion path.

Getting an NIH HPC account

- If you do not already have a Biowulf account, you can obtain one by following the instructions here *(https://hpc.nih.gov/docs/accounts.html)*.
- NIH HPC accounts are available to all NIH employees and contractors listed in the NIH Enterprise Directory.
- Obtaining an account requires PI approval and a nominal fee of $35 per month.
- Accounts are renewed annually contigent upon PI approval.

The Command Line Interface (CLI)

What is Unix?

- Unix is a proprietary operating system like Windows or MacOS (Unix based).
- There are many Unix and Unix-like operating systems, including open source Linux and its multiple distributions.
- Biowulf computational nodes use a Unix-like (Linux) operating system (distributions RHEL8/Rocky8).
- Biowulf requires knowledge and use of the command line interface (shell) to direct computational functionality.
- To work on the command line we need to be able to issue Unix commands to tell the computer what we want it to do.

> **Tip**
>
> A basic foundation of Unix is advantageous for most scientists, as many bioinformatics open-source tools are available or accessible by command line on Unix-like systems.

Accessing your local terminal or command prompt

Mac OS

- Type `cmd + spacebar` and search for "terminal". Once open, right click on the app logo in the dock. Select `Options` and `Keep in Dock`.

Windows 10 or greater

> You can start an SSH session in your command prompt by executing ssh user@machine and you will be prompted to enter your password. ---Windows documentation *(https://docs.microsoft.com/en-us/windows/terminal/tutorials/ssh?source=recommendations)*

To find the Command Prompt, type `cmd` in the search box (lower left), then press `Enter` to open the highlighted Command Prompt shortcut.

How much Unix do we need to learn?

To work on Biowulf you really need to understand the following:

1. Directory navigation: what the directory tree is, how to navigate and move around with `cd`

2. Absolute and relative paths: how to access files located in directories
3. What simple Unix commands do: `ls`, `mv`, `rm`, `mkdir`, `cat`, `man`
4. Getting help: how to find out more on what a unix command does
5. What are “flags”: how to customize typical unix programs `ls` vs `ls -l`
6. Shell redirection: what is the standard input and output, how to “pipe” or redirect the output of one program into the input of the other --- Biostar Handbook *(https://www.biostarhandbook.com/introduction-to-unix.html)*

Connecting to Biowulf

- To connect to Biowulf, we use a secure shell (SSH) protocol.
- used to open an encrypted network connection between two machines, allowing you to send & receive text and data without having to worry about prying eyes. *(https://carpentries-incubator.github.io/hpc-intro/11-connecting/index.html)*
- `man ssh`

Establishing a remote connection

```
ssh username@biowulf.nih.gov
```

"username" = NIH/Biowulf login username.

> **Note**
>
> If this is your first time logging into Biowulf, you will see a warning statement with a yes/no choice. Type "yes".

Type in your password at the prompt. **The cursor will not move as you type your password!**

SLURM commands

You will also need to know commands specific to the Biowulf job scheduling system:

- `sbatch` submit slurm job
- `swarm` submit a swarm of commands to cluster
- `sinteractive` allocate an interactive session
- `sjobs` show brief summary of queued and running jobs
- `squeue` display status of slurm batch job
- `scancel` delete slurm jobs

How to load / unload a module

- To see a list of available software in modules use

```
module avail
module avail [appname|string|regex]
module -d
```

- To load a module

```
module load appname
module load appname/version
```

- To see loaded modules

```
module list
```

- To unload modules

```
module unload appname
module purge #(unload all modules)
```

> **Note**
>
> You may also create and use your own modules.

Getting help on Biowulf: NIH HPC Documentation

The NIH HPC systems are well-documented at hpc.nih.gov *(https://hpc.nih.gov/)*.

- User guides *(https://hpc.nih.gov/docs/user_guides.html)*
- Training documentation *(https://hpc.nih.gov/training/)*
- How To *(https://hpc.nih.gov/docs/how_to.html)* docs

Additional HPC help

- Contact staff@hpc.nih.gov *(mailto:staff@hpc.nih.gov)*
  The HPC team welcomes questions and is happy to offer guidance to address your concerns.

Monthly Zoom consult sessions

- The HPC team offers montly zoom consult sessions. "All problems and concerns are welcome, from scripting problems to node allocation, to strategies for a particular project, to anything that is affecting your use of the HPC systems. The Zoom details are emailed to all Biowulf users the week of the consult." *(https://hpc.nih.gov/training/)*

- Bioinformatics Training and Education Program
  If you experience any difficulties or challenges, especially with different bioinformatics applications, please do not hesitate to email us at BTEP *(mailto:ncibtep@nih.gov)*.

Learning Unix: Classes / Courses

- Introduction to Biowulf (May – Jun, 2023) *(https://bioinformatics.ccr.cancer.gov/docs/biowulf-introduction-summer-2023/index.html)*
- Introduction to Unix on Biowulf (Jan – Feb, 2023) *(https://bioinformatics.ccr.cancer.gov/docs/unix-on-biowulf-2023/index.html)*
- Bioinformatics for Beginners: Module 1 Unix/Biowulf *(https://bioinformatics.ccr.cancer.gov/docs/b4b/Module1_Unix_Biowulf/Lesson1/)*

Additional Unix Resources:

- BashScripting_LinuxCommands from the NIH HPC team *(https://hpc.nih.gov/training/handouts/BashScripting_LinuxCommands.pdf)*
- Fosswire linux reference sheet *(https://bioinformatics.ccr.cancer.gov/docs/b4b/fosswire_reference.pdf)*

Key points

- Biowulf is the high performance computing cluster at NIH.
- To work on Biowulf, you will need to use the command line interface, which requires some knowledge of unix commands.
- When you apply for a Biowulf account you will be issued two primary storage spaces:
  1. `/home/$User` (16 GB)
  2. `/data/$USER` (100 GB).
- Hundreds of pre-installed bioinformatics programs are available through the `module` system.
- Computational tasks on Biowulf should be submitted as a job (`sbatch`, `swarm`) or through an interactive session (`sinteractive`).
- Do not run computational tasks on the login node.

Lesson 5

Learning Objectives

1. Learn about popular programming languagues in bioinformatics
2. Compare advantages and disadvantages of Python and R
3. Discuss what you will need to learn to use these languages
4. Discuss learning resources

Choosing a programming language

What is a programming language?

> A programming language is a formal language that specifies a set of instructions for a computer to perform specific tasks. It's used to write software programs and applications, and to control and manipulate computer systems.
>
> Key features of programming languages include:
> * Syntax (rules and structure used to write code)
> * Data types (type of values that can be stored in a program)
> * Variables (named memory locations that can store values)
> * Operators (symbols used to perform operations on values)
> * Control Structures (statements used to control the flow of a program)
> * Libraries (collections of pre-written code used to perform common tasks and speed up development)
> * Paradigms (programming styles / philosophies) --- GeeksforGeeks *(https://www.geeksforgeeks.org/introduction-to-programming-languages/)*

Examples include C++, C#, Perl, Java, Ruby, Python, Julia, and R.
More on paradigms, here *(https://medium.com/@LiliOuakninFelsen/functional-vs-object-oriented-vs-procedural-programming-a3d4585557f3)*.

Why learn programming?

Do all molecular scientists need to learn a programming language?

- Absolutely not.

BUT

- We are in a big data era, and learning to code can be extremely beneficial, especially if you do not have access to bioinformatics analysts to analyze the data for you or expensive licensed software.

Which programming language should I learn?

1. Bash
    - Most of bioinformatics can be done by understanding specific software applications and running those applications in a pipeline, usually using some form of bash scripting. Bash as a scripting language is fairly important for processing biological data, though arguably, not a formal programming language *(https://stackoverflow.com/questions/28693737/is-bash-a-programming-language)*.
2. Python or R
    - Depending on your goals, you may lean toward one programming language over another. For example:
        - Interested in statistics and data visualization? R may be for you.
        - Interested in software development and machine learning? A more general language like Python may be a better fit.

Check out this video *(https://omgenomics.com/programming-languages/)*!

> **Tip**
>
> Ultimately, what language you choose to learn will depend on what you actually want to do with your skills. If you want to become a bioinformatics analyst and are interested in developing scripts / programs for the greater community, you should probably learn bash, R, and python. If you are not developing pipelines or scripts for others to use, you can probably pick your poison. Though, you will likely still need to know some degree of all three.

What is R?

- released in 1993
- a computational language and environment for statitical computing and graphics.
    - complex statistical functions easily accessible
    - easy to get started, but more difficult to learn
- Key features:
    - open-source
    - extensible (Packages on CRAN (> 19,000 packages), Github, Bioconductor)

    - wide community
    - Maintained by a network of collaborators - The R Core Team

Check out more on The R Project for Statistical Computing website *(https://www.r-project.org/about.html)*.

What is Python?

- developed as early as 1991
- high-level, popular, general-purpose programming language that has a readable and easy to learn syntax
- Key features:
    - easy to read
    - easy to learn
    - interpreted
    - multi-platform
    - wide community
    - open source libraries (> 300,000)
- Two major versions (`python2` and `python3`)
- Not as easy to just start analyzing data

Check out more at https://www.python.org/ *(https://www.python.org/)*.
Also, check out this primer for biologists *(https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.0030199)*.

Advantages of R and Python

R Programming

- Data Visualization (Base R and ggplot2)
    - additional packages that enhance these, especially for -omics data
- More packages for data science / bioinformatics
    - Bioconductor
- Report generation
    - R Markdown
    - Quarto
- more popular among scientists and academics (i.e., non-programmers)

Python

- More consistent syntax (generally a right way to do something)
- Large data manipulation (generally more efficient)
    - shines in machine learning (scikit-learn *(https://scikit-learn.org/stable/)*)
- Report Generation
    - Jupyter Notebook
- More popular among software developers and across multiple domains

> **Tip**
>
> There is no right answer to the question, "which programming language should I learn, R or Python?". They are both valuable programming languages with different strengths and weaknesses. Choosing one or the other will come down to several factors such as your analysis goals, the time you have to learn, and what those around you are using.

Check out this comparison from Toward Data Science, *Python vs R: The Basics (https://towardsdatascience.com/python-vs-r-the-basics-d754c45c1596)*, author Sidney Kung:

	Python	R
General	Python is a general-purpose programming lanuage for data analysis and scientific computing.	R is a functional programming enviornment and language for statstical computing and graphics.
Objective	Data Science, Web Developoment, Embedded Systems	Data Science & Statistical Modeling
IDE	**iPython, Pycharm, Jupyter Notebook, Spyder**	**Rstudio, R GUI, R KWARD**
Data Collection	Supports CSV files, **SQL**, **JSON**, and webscraping with **BeautifulSoup.**	Can also import csv files with built-in **readr** library. R's library **RCurl** provides a simple way to make API requests, similar to Python's **requests** package.
Data Analysis	Orgnaize dataframes with **Pandas** filtering, sorting. Python takes a more streamlined approach for data science projects.	Complex data visualizaiton tools make the exploratory data analysis (EDA) process much more complex than Python.
Essential Packages & Libraries	**Numpy, Pandas, matplotlib, scipy, scikit-learn, TensorFlow**	**caret, stringr, ggplot2, knitr, tidyverse, markdown, shiny, forcats, haven**
Database Handling Capacity	Can easily handle large data because there are less constraints for memory usage	R computes everything in memory, so its capabilities are limited by RAM size. A major downfall of R is the inability to handle massive amounts of data
Data Visualization	Despite the capabilities of data visualization tools like **Matplotlib** and **Seaborn**, Python fails to measure up to data visualization features of R.	Developed by and for statisticians, R has complex data visualzatioon features.
Syntax	The 'zen of python' is that there's a proper way to write code.	R doesn't have this set of rules. Also indexing starts at 1, which can be considered unconventional for general programmers.
Learning Curve	Simple and readable code structure makes it easier for beginners to learn. It also allows for object-oriented programing. It also offers a wide range of data structures that you wouldn't expect from a general-purpose language.	R's functional syntax isn't easy for beginners, but not too challenging for those well versed in programming. It also offers a few data structures, but fails to handle large amounts of data.

What do you need to know to learn R or Python?

Installation

If you intend to use through Biowulf, no installation necessary.

R:

- Use this guide *(https://bioinformatics.ccr.cancer.gov/docs/rtools/)*.

Python:

- You can download directly from https://www.python.org/downloads/ *(https://www.python.org/downloads/)*.

How do we execute our code?

With both R and Python, code is executed

- interactively line by line from the command line
- interactively in an IDE
- as a script submitted from the command line or in an IDE

For python, to get started from the command line:

```
python
quit()
```

For R, to get started from the command line:

```
R
q()
```

What is an IDE?

An IDE is an integrated development environment. IDEs generally include features such as:

- Console
- File access
- Environment / variable view
- Data view
- Plotting window
- History
- Autocomplete
- Debugging
- Markdown

IDEs make coding easier. They increase productivity and facilitate project management. Using an IDE will allow you to more effectively organize code and results as you tackle data analysis problems.

IDEs for R and Python

R

- RStudio *(https://posit.co/products/open-source/rstudio/)*
- VS Code*
    - R *(https://code.visualstudio.com/docs/languages/r)*
    - Python *(https://code.visualstudio.com/docs/languages/python)*

Python

- JupyterLab / Jupyter Notebook *(https://jupyter.org/try)**
    - Can be used with C++, Julia, GNU octave, R, Ruby, and Scheme
- Spyder *(https://www.spyder-ide.org/)*
- iPython *(https://ipython.org/)*
- Google colab *(https://colab.google/)*

Elements of programming with python or R

- libraries
- syntax
- variables

- functions
- data types
- loops and conditionals

Libraries

R Packages can be found at:

- CRAN *(https://cran.r-project.org/)*
    - METACRAN *(https://www.r-pkg.org/)*- to search for packages
- Bioconductor *(https://bioconductor.org/)*
- Github

Python

- Python Package Index (PyPI) *(https://pypi.org/)*

Bioconductor

- A repository for R packages related to biological data analysis *(https://bioconductor.org/)*, primarily bioinformatics and computational biology.
- a great place to search for -omics packages and pipelines.
- Released every 6 months and work with a specific version of R.
    - included packages are "mutually compatible, traceable, and guaranteed to function for the associated version of R"
    - Package types: Software, annotation, experimental data, workflows

---

Bioinformatics related python packages

- Biopython *(https://biopython.org/)*
- Bioconda *(https://bioconda.github.io/)*
    - Conda, as a package management and environment management system was created for python but now can be used for any language.

R Syntax

- more functional
    - built around functions (`function_name()`)
- Case sensitive
- white space insensitive (rules for line continuation)
- `<-` or = assignment operators
- # used for comments
- keywords or words with special meaning (`?reserved`)
    - for example, `if`, `else`, `repeat`, `while`, `function`, `for`, `in`, `next`, and `break` are used for control-flow statements and declaring user-defined functions.
- statement grouping with `{}`
- indexing starts with 1

Python Syntax

- more object oriented (`.` is an operator and should not be used to name variables)
- = assignment operator
- 33 reserved words `help("keywords")`
- lists use brackets `[]`, dictionaries use `{}`
- indentation is important (4 spaces) - defines blocks of code
- indexing starts with 0

Compare the code

A syntax comparison from Dataquest: https://www.dataquest.io/blog/python-vs-r/ *(https://www.dataquest.io/blog/python-vs-r/)*.

> **Note**
>
> R code can be run using python with the `rpy2` library. Python code can be executed through R using the `reticulate` package.

Variables

Essentially named storage that can be manipulated.

Rules for R variables:

1. Avoid spaces or special characters EXCEPT '_' and '.'
2. No numbers or underscores at the beginning of an object name.
3. Avoid common names with special meanings (See ?Reserved) or assigned to existing functions (These will auto complete).
4. Case sensitive

Rules for Python variables:

1. Contains alpha-numeric characters and underscores
2. Must start with a letter or the underscore character
3. cannot start with a number
4. case-sensitive

Functions

Used to perform specific tasks.

R:

```
product <- function(a,b){
  c<- a*b
  c
}
product(5,7)
```

```
[1] 35
```

Python:

```
def product(a,b):
  c = a*b
  return c

print(product(5,7))
```

```
35
```

Code example from https://www.r-bloggers.com/2017/05/r-vs-python-different-similarities-and-similar-differences/ *(https://www.r-bloggers.com/2017/05/r-vs-python-different-similarities-and-similar-differences/)*

Data Types

R:
Data types: integer, numeric, character, and logical
Data structures: vectors, lists, data frames, matrices.

```
x <- c(1,2,3)
typeof(x)
## [1] "double"
class(x)
## [1] "numeric"
is.vector(x)
## [1] TRUE
```

Python:
Data types: Integers, Floats, Long, Complex, Strings, booleans (TRUE, FALSE)
Data structures: arrays, tuples, lists, dictionaries

```
import numpy as np
x = [1,2,3]
x = np.array(x)
print(type(x))
```

```
<class 'numpy.ndarray'>
```

Loops and conditionals

Loops - used to iterate over a sequence

R:

```
fruit <- c('apples','bananas','cantaloupe')

for(i in fruit) {
  print(i)
}
```

```
[1] "apples"
[1] "bananas"
[1] "cantaloupe"
```

Python:

```
fruit=['apples', 'bananas', 'cantaloupe'] #Loop for a list of fruits

for i in fruit:
    print(i)
```

```
apples
bananas
cantaloupe
```

Conditionals - code is executed based on conditions

R:

```
x<-3
y<-5

if(x<y){
  print(paste(x, 'is less than', y))
} else{
  print(paste(x, 'is not less than', y))
}
```

```
[1] "3 is less than 5"
```

Python:

```
x=3
y=5

if x<y:
  print(x, 'is less than', y)
else:
  print(x, 'is not less than', y)
```

```
3 is less than 5
```

Resources to learn

BTEP and Others

- Check the NIH Bioinformatics Calendar *(https://bioinformatics.ccr.cancer.gov/btep/)* for upcoming events including courses or lessons on python and R.
- Past BTEP courses
    - Class documentation *(https://bioinformatics.ccr.cancer.gov/btep/class-documents/)*
    - Video Archive *(https://bioinformatics.ccr.cancer.gov/btep/btep-video-archive-of-past-classes/)*
- NIH library *(https://www.nihlibrary.nih.gov/training/calendar)*
- NIAID Bioinformatics Resources *(https://bioinformatics.niaid.nih.gov/resources)*

Dataquest and Coursera

- Dataquest - great for learning programming skills
- Coursera - great for learning more specific skills

  Click here *(https://bioinformatics.ccr.cancer.gov/btep/self-learning/)* for license information.

Books and other resources:

- See this list *(https://bioinformatics.ccr.cancer.gov/docs/rintro/References/Resources/)* for introductory R material.
- A Primer for Computational Biology, Shawn T. O'Neil *(https://open.oregonstate.education/computationalbiology/)*

Sources

1. https://www.datacamp.com/blog/python-vs-r-for-data-science-whats-the-difference#gs.JrY_3bk
2. https://shiring.github.io/r_vs_python/2017/01/22/R_vs_Py_post

3. https://realpython.com/python-ides-code-editors-guide/
4. https://medium.com/@hamza_33678/programming-for-bioinformatics-r-vs-python-52969a1f7a49#:~:text=While%20both%20R%20and%20Python,in%20keeping%20RAM%20co
5. https://towardsdatascience.com/python-vs-r-the-basics-d754c45c1596
6. https://www.dataquest.io/blog/python-vs-r/
7. Learning Python for Data Science: What to Learn and Why, Cindy Sheffield, NIH Library

Lesson 6

Managing Bioinformatics Projects with Jupyter Lab

Learning Objectives

After this class, participants will have obtained the foundation needed to start using Jupyter Lab as an all-in-one place to maintain code, output, and other description of analysis steps. Participants will be able to

- Start a Jupyter Lab session
- Describe the Jupyter Lab interface
- Initiate a Jupyter Notebook
- Know how to access Jupyter Lab
- Know how to create formatted text and code in a Jupyter Notebook
- Describe ways to export and share a Jupyter Notebook

Course recap

So far, this course series has addressed several areas that are important for anyone venturing into bioinformatics. These include:

- Available bioinformatics resources and tools at NIH
- Various assays to measure the different components of the Central Dogma of Biology (ie. whole genome sequencing, RNA sequencing)
- Data management
- High performance computing systems (Biowulf at NIH *(https://hpc.nih.gov/systems/)*)
- Programming languages such as R and Python

Start Jupyter Lab

To start Jupyter Lab, type the following into the command prompt. The `--no-browser` option prevents a web browser from opening.

```
jupyter lab --no-browser
```

Copy and paste any of the following URLs into a web browser to start using Jupyter. These URLs will be different for every Jupyter Lab session.

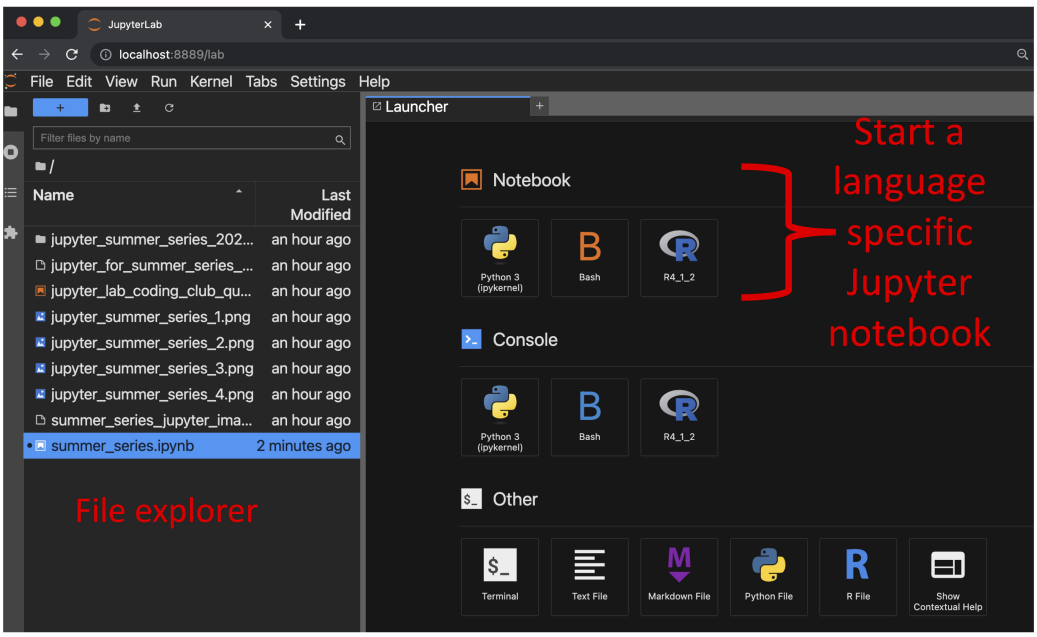
```
```

```
To access the server, open this file in a browser:
        file:///Users/wuz8/Library/Jupyter/runtime/jpserver-30985-ope
    Or copy and paste one of these URLs:
        http://localhost:8890/lab?token=1952fcce201164f8368f2666f2f2(
        http://127.0.0.1:8890/lab?token=1952fcce201164f8368f2666f2f2(
```

Tip

Start a Jupyter Lab session in the project folder. This folder will contain input and analysis output.

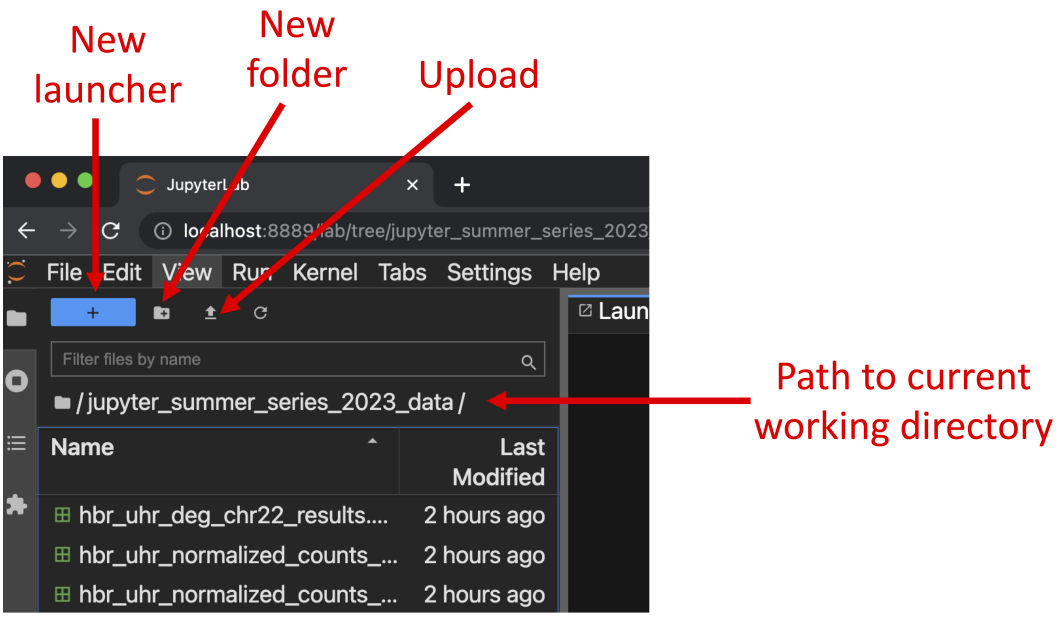
Jupyter Lab interface



Jupyter Lab is compatible with many languages

- Bash, Python, and R
- See https://github.com/jupyter/jupyter/wiki/Jupyter-kernels *(http://localhost:5808/)* for a list of Jupyter compatible languages

Jupyter Lab file explorer



Start a Python Jupyter Notebook

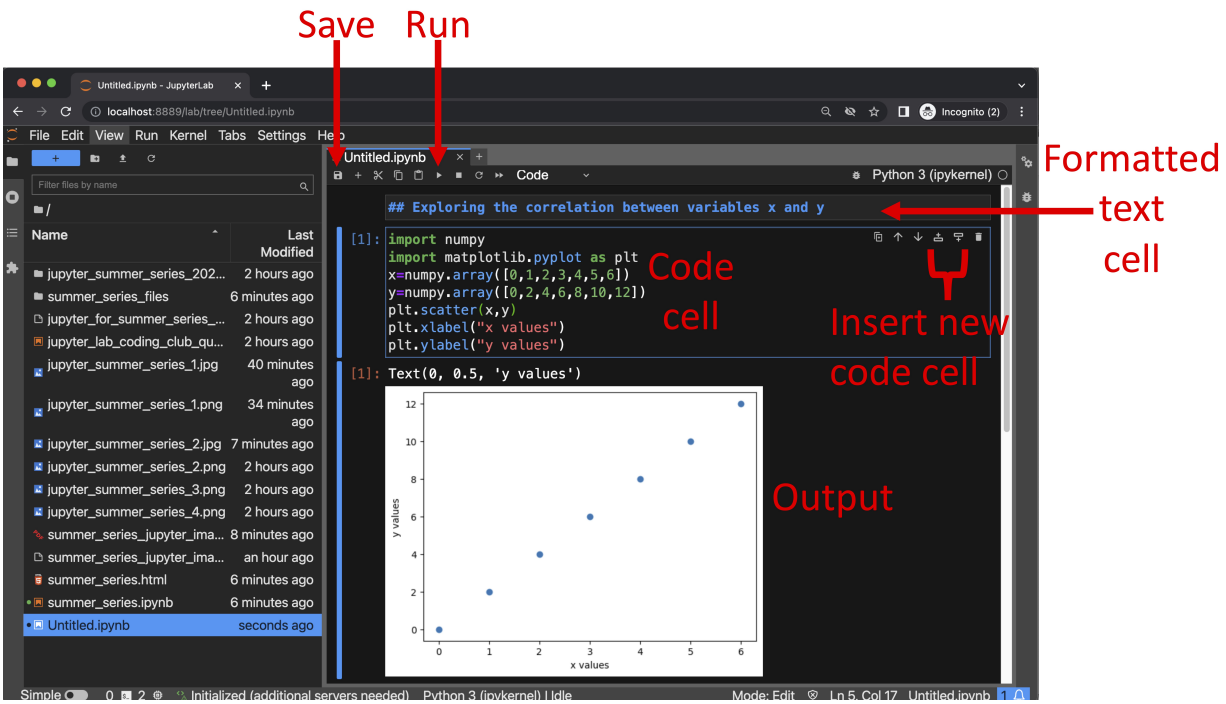
Click on the "Python 3 (ipykernel)" tab to start a Python Jupyter Notebook. The Jupyter Notebook is a part of Jupyter Lab.

The note book is where users

- Write code
- View output
- Document analysis steps using formatted text written in markdown

> **Note**
>
> "Markdown is a lightweight markup language for creating formatted text using a plain-text editor." -- https://en.wikipedia.org/wiki/Markdown *(https://en.wikipedia.org/wiki/Markdown)*

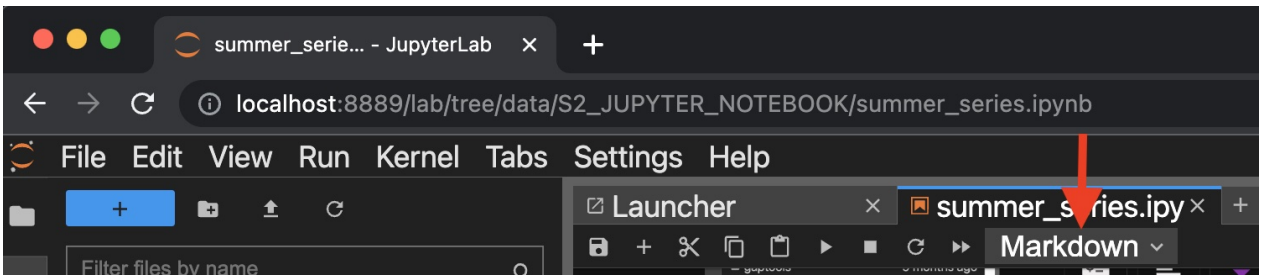
Keeping code, output and analysis steps all in one place

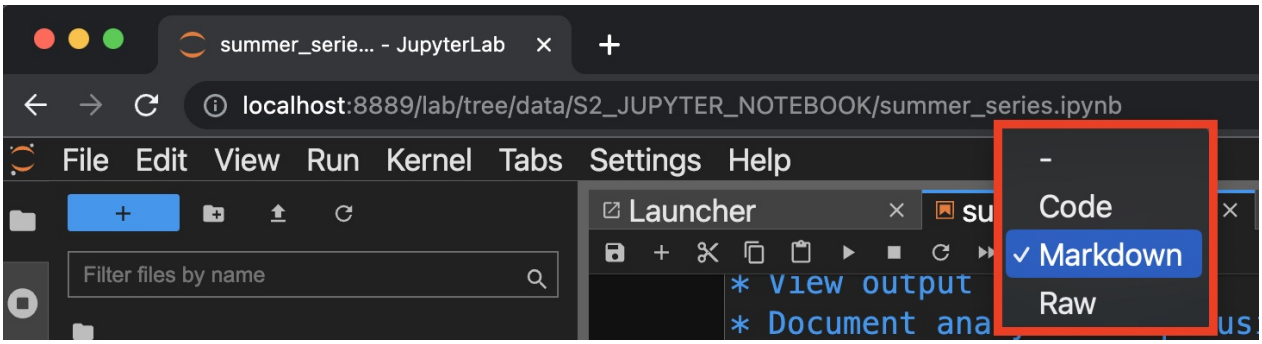


Note

A new Jupyter Notebook is given the name "Untitled". Change this to something meaningful either using the save icon on the notebook menu bar or right-clicking on the "Untitled" notebook in the file explorer and choose "Rename". Jupyter Notebooks have extension ipynb, which stands for interactive Python notebook.

Changing between markdown and code





Ways to access and use Jupyter Lab

- Install on local machine (see https://github.com/jupyter/jupyter/wiki/Jupyter-kernels *(https://github.com/jupyter/jupyter/wiki/Jupyter-kernels)*)
- Available on Biowulf (see https://hpc.nih.gov/apps/jupyter.html *(https://hpc.nih.gov/apps/jupyter.html)*)

Writing formatted text

See https://www.markdownguide.org/basic-syntax/ *(https://www.markdownguide.org/basic-syntax/)* for a markdown guide.

Custom heading sizes

Use # to specify heading level

```
# Heading level 1 (largest)
## Heading level 2 (second largest)
### Heading level 3 (third largest)
...
```

Lists

Un-ordered lists: use * or -

```
- DNA
- RNA
- protein
- metabolite
```

Ordered list: use numbers

```
1. Obtain sequencing data
2. Perform pre-alignment QC
3. Adapter and/or quality trim
3. Align sequencing data to reference genome
4. Obtain gene expression count matrix
5. Run differential expression analysis
6. Pathway analysis
```

Insert images

```
<img src="image_path" />
```

Insert links

```
[Description of website](insert url)
```

Code and visualization

Import data using Pandas

Pandas *(https://pandas.pydata.org)* is a Python package used for working with tabular data. The dataset used here is the differential gene expression analysis results from the HBR and UHR study *(https://rnabio.org/module-01-inputs/0001/05/01/RNAseq_Data/)*. To work with this, users will need to import it using the `read_csv` function of Pandas as the data is in a csv file (hbr_uhr_deg_chr22_with_significance.csv located in the folder jupyter_summer_series_2023_data). The path to this file is used as the **argument** for the `read_csv` function.

```
# Load the Pandas package
import pandas
```

```
# Import the data

hbr_uhr_deg_chr22=pandas.read_csv("./jupyter_summer_series_2023_data/

# View the first several lines of hbr_uhr_deg_chr22
hbr_uhr_deg_chr22.head()
```

	name	log2FoldChange	PAdj	-log10PAdj	significance
0	SYNGR1	-4.6	5.200000e-217	216.283997	down
1	SEPT3	-4.6	4.500000e-204	203.346787	down
2	YWHAH	-2.5	4.700000e-191	190.327902	down
3	RPL3	1.7	5.400000e-134	133.267606	ns
4	PI4KA	-2.0	2.900000e-118	117.537602	down

Construct volcano plot using Seaborn

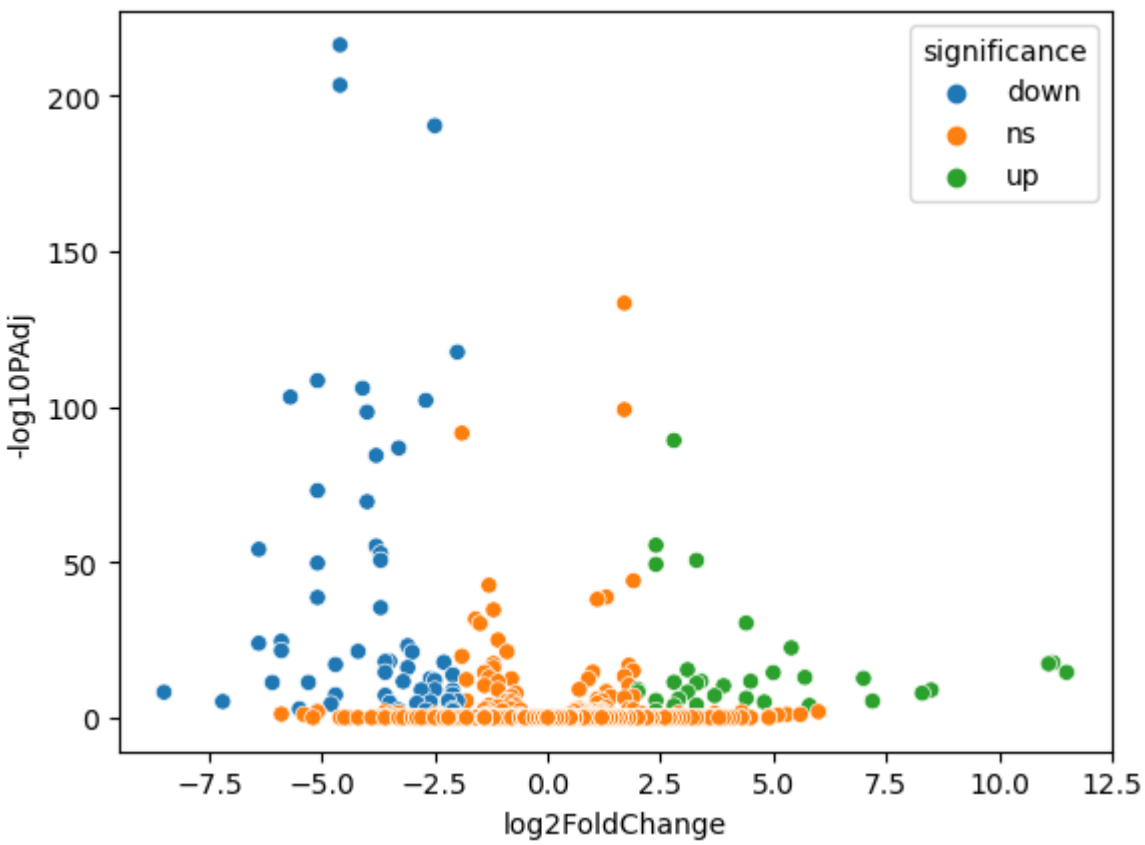
Seaborn *(https://seaborn.pydata.org)* is a popular visualization package for Python. Users can use its `scatterplot` function to generate scatter plots (in this case a volcano plot, which is a special type of scatter plot). The `scatterplot` function will take on arguments:

- Data: hbr_uhr_deg_chr22 (differential gene expression analysis results)
- `x`: x-axis values (ie. gene expression log2FoldChange)
- `y`: y-axis values (ie. -log10 of adjusted p-value)
- `hue`: color dots by whether gene expression is up, down, or has no change (see signifcance column of the data)

```
# Load the seaborn plotting package
import seaborn

seaborn.scatterplot(hbr_uhr_deg_chr22,x="log2FoldChange", y="-log10PA
```

```
<Axes: xlabel='log2FoldChange', ylabel='-log10PAdj'>
```



The volcano plot is a special scatter plot that depicts gene expression change versus the statistical significance of the change.

Construct heatmap using Seaborn

This exercise will use Seaborn's `clustermap` function to construct a gene expression heatmap of top differentially expressed genes in the HBR and UHR study. Heatmaps are another common visualization in RNA sequencing and allow users to identify clusters of samples with similar gene expression patterns.

First, import the dataset using `pandas.read_csv`. The `clustermap` function of seaborn takes the following arguments and options.

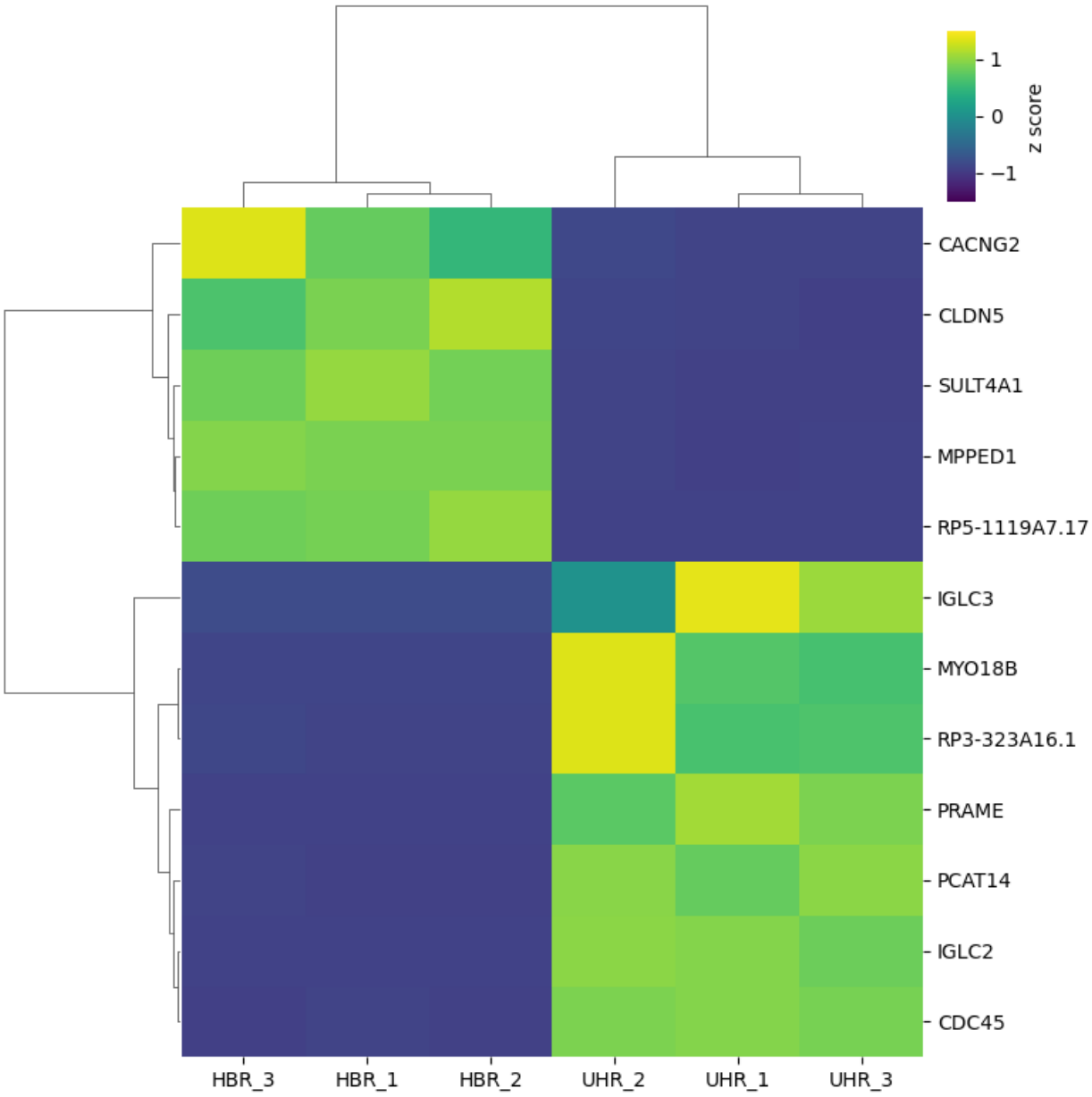
- Data: hbr_uhr_top_deg_normalized_counts
- `z_score`: z-score scale the gene expression counts
- `cmap`: specify a coloring scheme (ie. viridis)
- `figsize`: specify figure size
- `cbar_kws`: specify the title for the heatmap color bar using a key-value pair
- `cbar_pos`: specify coordinate to place the heatmap color bar

```
# Import the data

hbr_uhr_top_deg_normalized_counts=pandas.read_csv("./jupyter_summer_s
```

```
seaborn.clustermap(hbr_uhr_top_deg_normalized_counts,z_score=0,cmap='
                   figsize=(8,8),vmin=-1.5, vmax=1.5,cbar_kws=({"labe
                   cbar_pos=(0.855,0.8,0.025,0.15))
```

```
<seaborn.matrix.ClusterGrid at 0x1a3bd2190>
```



R code in a Python Jupyter Notebook

Using the `rpy2.ipython` package, users can run R code inside a Python Jupyter Notebook.

```
# Load rpy2.ipython
%load_ext rpy2.ipython
```

Using R to generate a principal components plot

Here, R will be used to generate principal components plot for the HBR and UHR study. Principal components plots are a popular way to visualize how samples in RNA sequencing cluster based on gene expression.

```
%%R
# Load packages using the library command
library(ggfortify)
```

```
Loading required package: ggplot2
```

```
%%R
# Import gene expression data using read.csv and store it as variable
counts <- read.csv("./jupyter_summer_series_2023_data/hbr_uhr_normal
```

```
%%R
# Look at the first few lines of counts
head(counts)
```

```
    Samples Treatment SULT4A1 MPPED1 PRAME IGLC2 IGLC3 CDC45 CLDN5 PCAT14
1 HBR_1.bam       HBR   375.0  157.8   0.0   0.0   0.0   2.6  77.6    0.0
2 HBR_2.bam       HBR   343.6  158.4   0.0   0.0   0.0   1.0  88.5    0.0
3 HBR_3.bam       HBR   339.4  162.6   0.0   0.0   0.0   0.0  67.2    1.2
4 UHR_1.bam       UHR     3.5    0.7 568.9 488.6 809.7 155.0   1.4  139.8
5 UHR_2.bam       UHR     6.9    3.0 467.3 498.0 313.8 152.5   2.0  154.4
6 UHR_3.bam       UHR     2.6    2.6 519.2 457.5 688.0 149.9   0.0  155.1
  RP5.1119A7.17 MYO18B RP3.323A16.1 CACNG2
1          53.0    0.0          0.0   42.7
2          57.6    0.0          0.0   35.0
3          51.9    0.0          1.2   56.6
4           0.0   59.5         51.9    0.0
5           0.0   84.2         76.2    1.0
6           0.0   56.5         53.1    0.0
```

The `autoplot` command takes the following arguments

- Principal components analysis results, which are stored in hbr_uhr_pca
- `data`: Expression counts table, which is stored as counts
- `colour`: column in the expression counts table to color the samples by (here color by Treatment)
- `size`: specify size of the dots

The layer theme was added to the principal components plot to customize the font sizes.

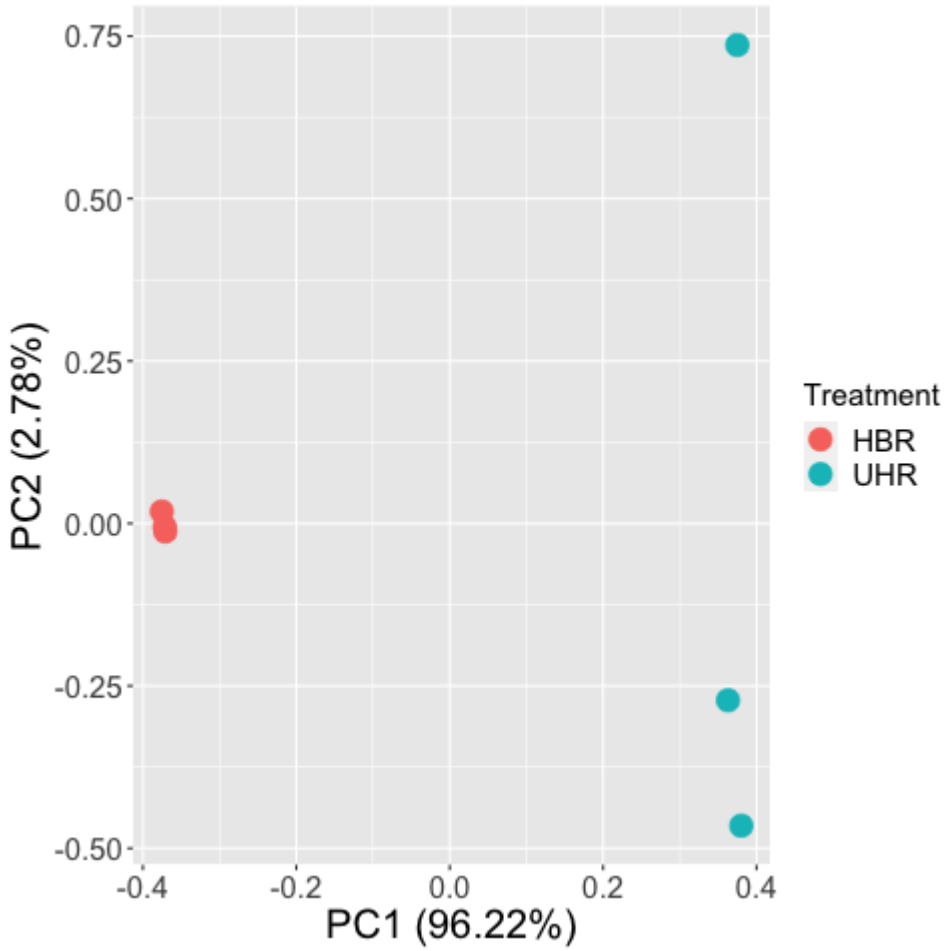
```
%%R

# Run principal components analysis on counts using the prcomp funct
```

```
hbr_uhr_pca <- prcomp(counts[3:14],scale.=TRUE,center=TRUE)

# Construct principal components plot.

autoplot(hbr_uhr_pca,data=counts,colour="Treatment",size=5)+
        theme(axis.title=element_text(size=20),
              axis.text=element_text(size=15),
              legend.title=element_text(size=15),
              legend.text=element_text(size=15))
```



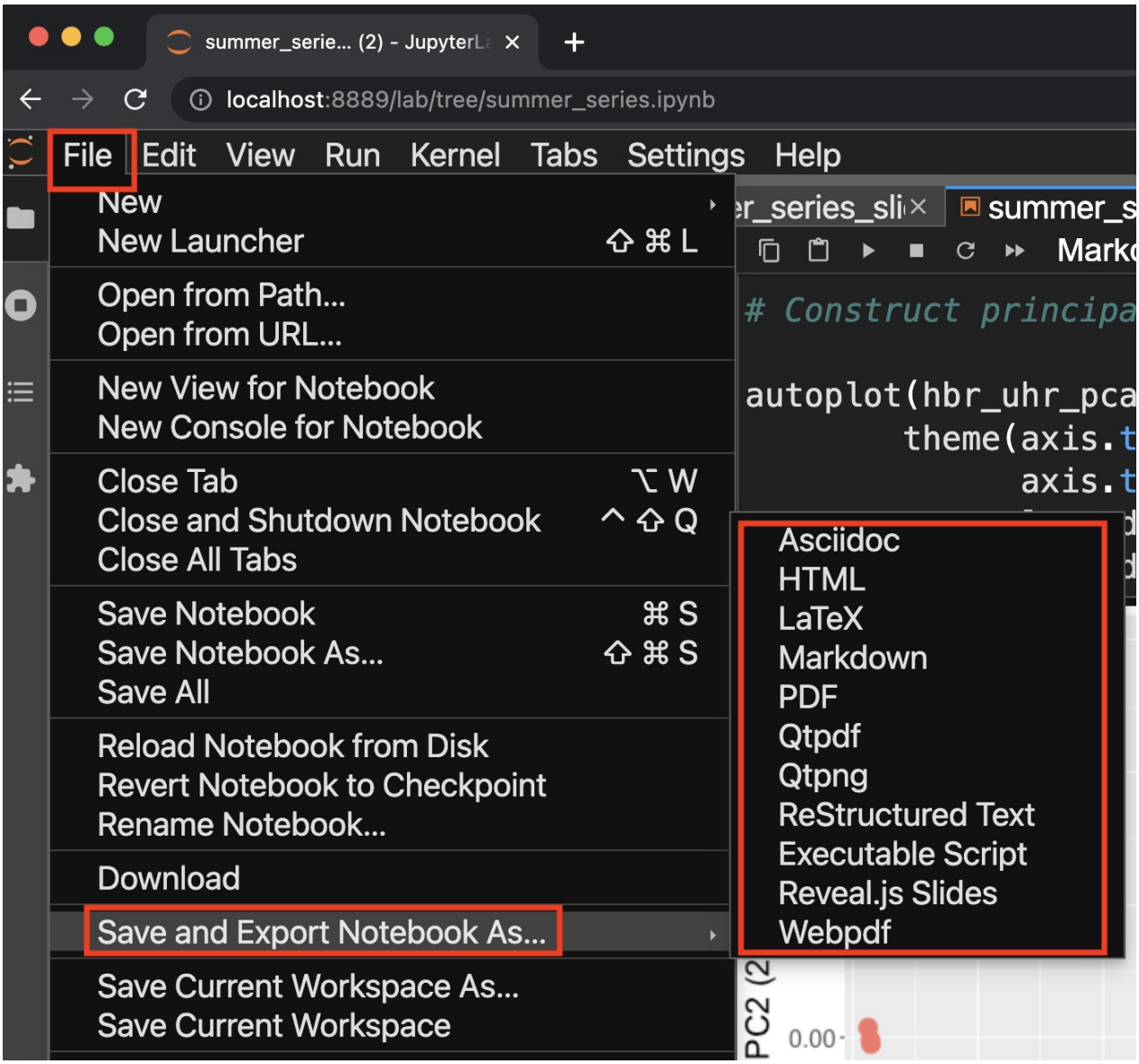
Running Unix commands

Users can run Unix commands within a Python Jupyter Notebook. To do this start a code block with "!" followed by the Unix command. For instance, use the `pwd` command in the code block below to list the present working directory.

```
!pwd
```

```
/Users/wuz8/Documents/jupyter_summer_series_2023
```

Exporting Jupyter Notebook using GUI



Exporting Jupyter Notebook using command line

Use the `jupyter nbconvert` command at the command prompt to convert Jupyter Notebook to various available formats, including html, pdf, and slides. The format is specified after the `--to` option.

```
jupyter nbconvert --to format
```

Sharing Jupyter Notebook

- Github *(https://github.com)*
  - Static notebook (ie. users will not be able to run)
- Binder *(https://mybinder.org)*
  - Provide data
  - Provide list of packages
  - Users can run the notebook
  - Example *(https://mybinder.org/v2/gh/ncbi/workshop-ncbi-data-with-python/main?filepath=notebooks%2Fworkshop.py)*

Download example data and Jupyter Notebook

The example data and Jupyter Notebook are inside a zip file, so unzip it after downloading to access the content.

Example data