

pies_2023



***Bioinformatics Training
& Education Program***

Table of Contents

Course overview

● Course Overview	6
● Example data used in this course	7

Lesson 1

● Lesson 1: Short introduction to Python, signing onto Biowulf, and starting Jupyter Lab	8
● Learning objectives	8
● What is Python and why use it?	8
● Signing onto Biowulf	9
● Change into Biowulf data directory	9
● Request an interactive session	9
● Load Jupyter	11
● Start Jupyter Lab	12
● Jupyter Lab - file explorer and launcher	12
● Jupyter Notebook - cells	13
● Python education resources	13
● Python command syntax	14
● Example of a Python command with and without options	14
● Finding help for Python commands	14
● Example of using help	14
● Copy class data to data directory	15

Lesson 2

● Lesson 2: Python data types and structures	16
● Learning objectives	16
● Signing onto Biowulf	16
● Change into data directory and copy course data	16
● Request interactive session	17
● Load Jupyter	18
● Start Jupyter Lab	19
● Python data types and data structures	20
● Identifying data type and structure in Python	20
● Variable assignments	21
● Conditionals	21
● Data frames	23
● Importing tabular data with Pandas	23
● Lists and tuples	24
● List versus tuples (mutable versus immutable)	25
● Arrays	25
● Range	26
● Dictionaries	27

Lesson 3

● Lesson 3: Data wrangling using Python	28
● Learning objectives	28
● Importing tabular data using Pandas	28

● Get dimensions of a data frame	30
● Row indices/names	31
● Data wrangling	32
● Subsetting	32
● Subsetting by integer positions	32
● Subsetting using column names	34
● Summary statistics of data frames	35
● Replacing column names	36
● Mathematical operations on data frames and filtering	36
● Removing and adding columns to a data frame	37

Lesson 4

● Lesson 4: Data visualization using Python	42
● Learning objectives	42
● Python data visualization tools	42
● Visualization using Seaborn	42
● Load packages	42
● Modify the basic plot elements with Seaborn.	43
● Constructing biologically relevant plots	47

Practice questions

Lesson 2 practice	57
● Lesson 2 practice questions	57
● Question 1	57
● Question 2	57

● Question 3	57
● Question 4	58
● Question 5	58
● Question 6	58

Lesson 3 practice **60**

● Lesson 3 practice questions	60
● Question 1	60
● Question 2	60
● Question 3	60
● Question 4	61
● Question 5	61
● Question 6	61
● Question 7	62
● Question 8	62

Lesson 4 practice **63**

● Lesson 4 practice questions	63
● Question 1	63
● Question 2	63
● Question 3	64
● Question 4	64

Finding help

● Finding help	66
----------------	----

Course Overview

Welcome to the Python Introductory Education Series (PIES) course. This course is composed of four lessons (see schedule below) and is meant to help those with no or limited experience in Python get started using this general purpose scripting language for data analyses. Each one-hour lesson will be followed by an optional one-hour help session. At the end of this course series, participants should

- Have obtained a broad overview of Python, including
 - Familiarity with tools used to write Python code
 - Knowledge of Python command syntax
 - Ability to find help for Python commands
 - Knowledge of where to find Python packages
 - Familiarity with self-learning resources
- Be able to describe Python data types and structures and provide examples of where some of the data structures are used
- Know how to work with and wrangle tabular data
- Be able to construct data visualizations

Lesson schedule:

- Lesson 1: Short introduction to Python, signing onto Biowulf, and starting Jupyter Lab (Tuesday, August 15, 2023) (https://bioinformatics.ccr.cancer.gov/docs/pies-2023/pies_lesson1/)
 - Lesson 1 recording (<https://cbiit.webex.com/cbiit/ldr.php?RCID=28b10cbe0179993cd0008f1300a1a9ed>)
- Lesson 2: Python data types and structures (Thursday, August 17, 2023) (https://bioinformatics.ccr.cancer.gov/docs/pies-2023/pies_lesson2/)
 - Lesson 2 recording (<https://cbiit.webex.com/cbiit/ldr.php?RCID=41f35ca8d9d251425edd765389b47c32>)
- Lesson 3: Data wrangling using Python (Tuesday, August 22, 2023) (https://bioinformatics.ccr.cancer.gov/docs/pies-2023/pies_lesson3/)
 - Lesson 3 recording (<https://cbiit.webex.com/cbiit/ldr.php?RCID=0749d0a1a34b9dbcc3abfbb6b34292ff>)
- Lesson 4: Data visualization using Python (Thursday, August 24, 2023) (https://bioinformatics.ccr.cancer.gov/docs/pies-2023/pies_lesson4/)
 - Lesson 4 recording (<https://cbiit.webex.com/cbiit/ldr.php?RCID=f6dc3393c95acb10a4ffb2a3b1be6a29>)

A Biowulf account is needed for this class. Visit the [Biowulf User Dashboard \(https://hpc.nih.gov/auth/dashboard/\)](https://hpc.nih.gov/auth/dashboard/) to unlock an inactive account. For instructions on obtaining a Biowulf account, visit <https://hpc.nih.gov/docs/accounts.html> (<https://hpc.nih.gov/docs/accounts.html>).

Example data used in this course

[Download data used in this course](#)

Lesson 1: Short introduction to Python, signing onto Biowulf, and starting Jupyter Lab

Learning objectives

After this lesson, participants will

- Be able to describe Python and provide rationale for using it
- Know how to start a Jupyter Lab session on Biowulf (Jupyter Lab will be used to interact with Python throughout this course)
- Be familiar with places for getting Python packages
- Become familiar with navigating the Jupyter Lab environment
- Be able to describe Python command syntax
- Know how to find help for Python commands
- Become familiar with continuing and self-learning resources

What is Python and why use it?

- Scripting language
 - Facilitates reuse and reproducibility
- Can be used to analyze large datasets
- Extensive external packages that can be used for
 - Data wrangling
 - Data visualization
 - Single cell RNA sequencing analysis
 - Working with biological sequences
 - Interfacing with bioinformatics databases
- Strong support community
- Easy to learn

Note

Python packages can be found at [The Python Package Index \(https://pypi.org\)](https://pypi.org).

Signing onto Biowulf

In this course series, participants will interact with Python through Jupyter Lab on Biowulf. Thus, the first step is to sign onto Biowulf using `ssh`. Replace username with participant's own Biowulf username.

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

- Mac: use `ssh` through the Terminal
- Windows: use `ssh` through the command prompt

Change into Biowulf data directory

Use `cd` to change into the participant's data directory on Biowulf. Again, replace username with participant's Biowulf username.

```
cd /data/username
```

Request an interactive session

Request an interactive session using `sinteractive` with the following options.

- `--gres=lscratch:5`: to allocate 5gb of local temporary/scratch storage space
- `--mem=2gb`: to request 2gb of memory or RAM
- `--tunnel`: to open up a channel of communication between local machine and Biowulf to allow interaction with applications like Jupyter Lab

```
sinteractive --gres=lscratch:5 --mem=2g --tunnel
```

After resources for the interactive session has been granted, users will see the information similar to that shown in Figure 1.

```
wuz8 — wuz8@biowulf:/data/wuz8 — ssh wuz8@biowulf.nih.gov — 93x25
[wuz8@biowulf wuz8]$ sinteractive --gres=lscratch:5 --mem=2gb --tunnel
salloc: Pending job allocation 6385785
salloc: job 6385785 queued and waiting for resources
salloc: job 6385785 has been allocated resources
salloc: Granted job allocation 6385785
salloc: Waiting for resource configuration
salloc: Nodes cn4275 are ready for job
srun: error: x11: no local DISPLAY defined, skipping
error: unable to open file /tmp/slurm-spank-x11.6385785.0
slurmstepd: error: x11: unable to read DISPLAY value

Created 1 generic SSH tunnel(s) from this compute node to
biowulf for your use at port numbers defined
in the $PORTn ($PORT1, ...) environment variables.

Please create a SSH tunnel from your workstation to these ports on biowulf.
On Linux/macOS, open a terminal and run: Copy and paste into new terminal (Mac) or command prompt (Windows)

ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov

For Windows instructions, see https://hpc.nih.gov/docs/tunneling
```

Figure 1: After interactive session resources have been allocated, users will see a ssh command that looks like that enclosed in the red rectangle. Open a new terminal (if working on a Mac) or command prompt (if working on a Windows computer) and then copy and paste this ssh command into the new terminal.

After copying and pasting the ssh command shown in Figure 1 to a new terminal or command prompt, hit enter to supply password and log in to Biowulf. This will complete the tunnel.

Hit enter after copying and pasting into a new terminal (Mac) or command prompt (Windows) to provide password and sign onto Biowulf, which will complete the tunnel.

```
(base) NCI-02227565-ML:~ wuz8$ ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov
Enter passphrase for key '/Users/wuz8/.ssh/id_rsa':
Last login: Tue Aug 15 16:24:28 2023 from 10.248.80.125
[wuz8@biowulf ~]$
```

Figure 2: Hit enter after copying and pasting the ssh command to a new terminal to provide password and log into Biowulf. This will complete the tunnel.

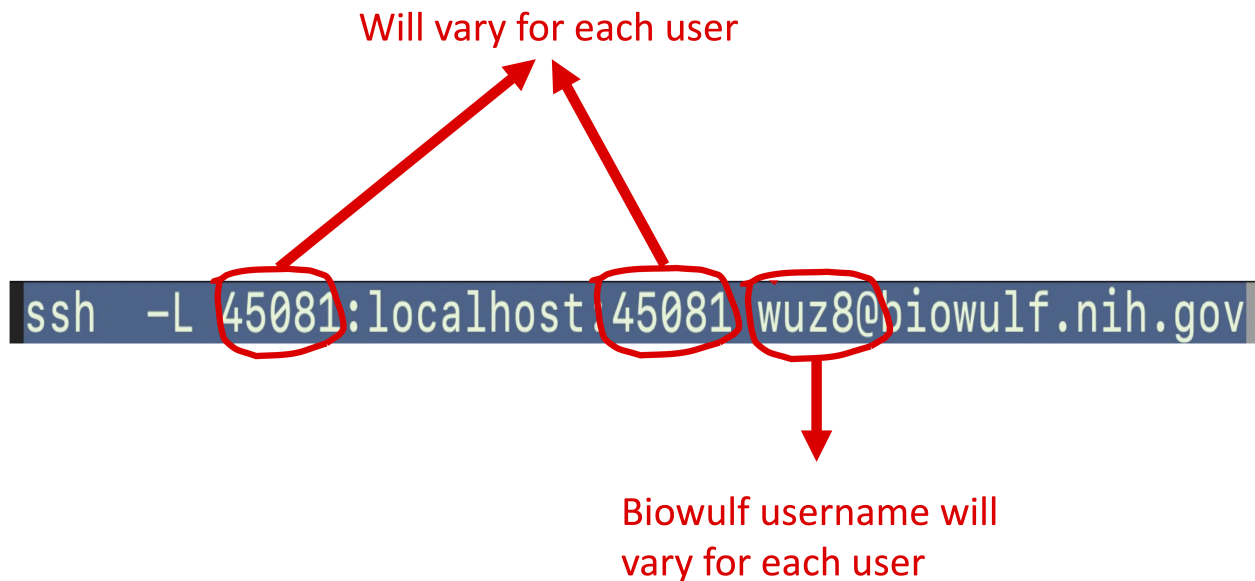


Figure 3: In the ssh command shown in Figure 1 and Figure 2, the numbers preceding and following "localhost" will differ depending on user. Also, the Biowulf username will differ for each user (wuz8 is the instructor's Biowulf username).

Load Jupyter

After the tunnel has been created, go back terminal (Mac) or command prompt (Windows) with the Biowulf interactive session and activate Jupyter (see Figure 4).

```
module load jupyter
```

```
salloc: job 6385785 queued and waiting for resources
salloc: job 6385785 has been allocated resources
salloc: Granted job allocation 6385785
salloc: Waiting for resource configuration
salloc: Nodes cn4275 are ready for job
srun: error: x11: no local DISPLAY defined, skipping
error: unable to open file /tmp/slurm-spank-x11.6385785.0
slurmstepd: error: x11: unable to read DISPLAY value

Created 1 generic SSH tunnel(s) from this compute node to
biowulf for your use at port numbers defined
in the $PORTn ($PORT1, ...) environment variables.

Please create a SSH tunnel from your workstation to these ports on biowulf.
On Linux/MacOS, open a terminal and run:

    ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov

For Windows instructions, see https://hpc.nih.gov/docs/tunneling

[wuz8@cn4275 wuz8]$ module load jupyter
[+] Loading git 2.39.2 ...
[+] Loading jupyter
[wuz8@cn4275 wuz8]$
```

Figure 4: Go back to the terminal (Mac) or command prompt (Windows) with the interactive session (look for `cn####` at the prompt). Do `module load jupyter` from here.

Start Jupyter Lab

Use the command below to start a Jupyter Lab session. Copy and paste either of the http links to a local browser to interact with Jupyter (see Figure 5).

```
jupyter lab --ip localhost --port $PORT1 --no-browser
```

```
[wuz8@cn4275 wuz8]$ jupyter lab --ip localhost --port $PORT1 --no-browser
To access the server, open this file in a browser:
  file:///spin1/home/linux/wuz8/.local/share/jupyter/runtime/jpserver-363837-open.html
Or copy and paste one of these URLs:
  http://localhost:45081/lab?token=ad4b828f83a0fd8ad468cadaed56590b8a34f7f0418e76f3
  or http://127.0.0.1:45081/lab?token=ad4b828f83a0fd8ad468cadaed56590b8a34f7f0418e76f3
```

Copy either of the http links to local browser

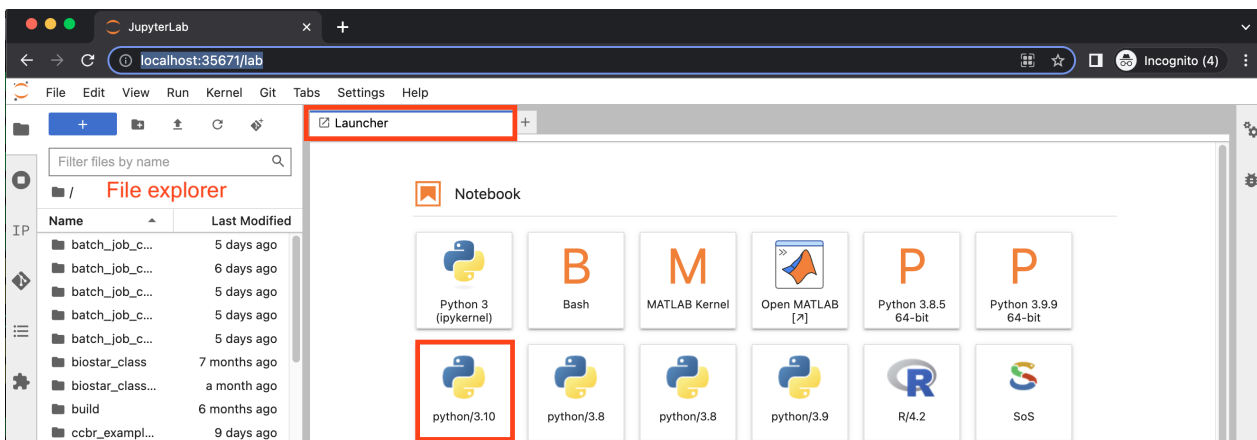
Figure 5: Start a Jupyter lab session using `jupyter lab --ip localhost --port $PORT1 --no-browser` and copy and paste either one of the http links to a local browser.

Warning

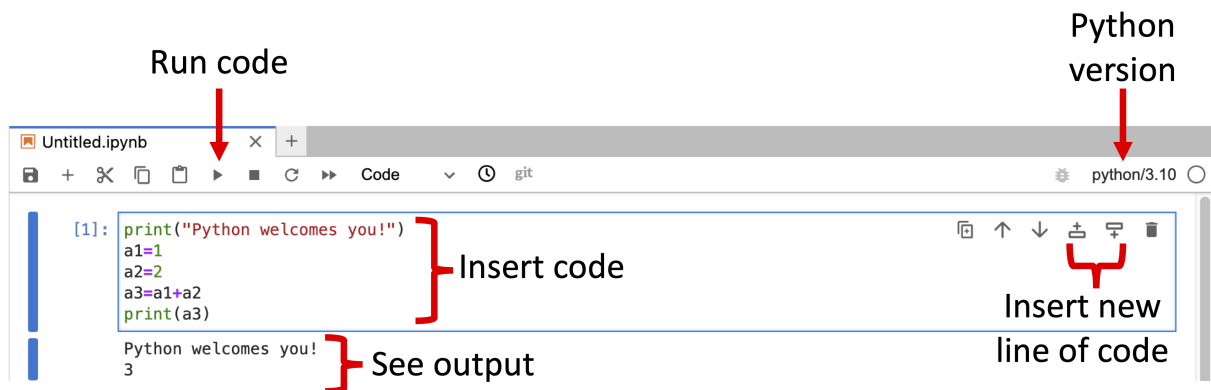
The URLs change with each Jupyter Lab session, so please do not copy from the examples shown below. Copy from the URLs provided in the Biowulf interactive session terminal instead.

Jupyter Lab - file explorer and launcher

- File explorer
- Launcher for starting language specific notebooks (for this course series, choose the python/3.10 notebook)



Jupyter Notebook - cells



Python education resources

- Coursera
 - Programming for Everybody (Getting Started with Python)
 - Instructor: Charles Severance, PhD (University of Michigan)
 - Data Analysis with Python
 - Instructor: IBM staff
 - Includes data wrangling and regression analysis
 - Data Visualization with Python
 - Instructor: IBM staff
 - Introduces data visualization using packages such as Matplotlib and Seaborn
- Dataquest
 - <https://www.dataquest.io/course/introduction-to-python/> (*https://www.dataquest.io/course/introduction-to-python/*)
 - <https://www.dataquest.io/path/data-scientist/> (*https://www.dataquest.io/path/data-scientist/*)
 - <https://www.dataquest.io/path/data-analyst/> (*https://www.dataquest.io/path/data-analyst/*)

Visit the [self learning resources page](https://bioinformatics.ccr.cancer.gov/btep/self-learning/) (*https://bioinformatics.ccr.cancer.gov/btep/self-learning/*) to request a Dataquest or Coursera license.

Python command syntax

The command syntax for Python is composed of the

- Command
- Argument, which is enclosed in the parentheses and what the command will act on
- Options, which is enclosed in parentheses and alters the way the command runs

```
command(argument, options)
```

Example of a Python command with and without options

```
print("Hello", "welcome to Python")
```

```
Hello welcome to Python
```

Include option `sep` to place a comma between "Hello" and "welcome to Python".

```
print("Hello", "welcome to Python", sep=", ")
```

```
Hello, welcome to Python
```

Finding help for Python commands

The `help` command can be used to view documentations for Python commands. It follows the Python command syntax. Insert the command in which help is needed into the parentheses.

```
help()
```

Example of using help

```
help(print)
```


Help on built-in function print in module builtins:

```
print(...)  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

Copy class data to data directory

The example datasets used for this course series reside in `/data/classes/BTEP/pies_2023_data`. Make a copy in your data directory.

```
cp -r /data/classes/BTEP/pies_2023_data ./pies_2023
```

Lesson 2: Python data types and structures

Learning objectives

After this class, participants will

- Be able to describe some common Python data types and structures
- Be able to identify Python data types
- Become familiar with variable assignment
- Be able to use conditional operators and if-else statements
- Be able to load packages
- Know how to import tabular data
- Know how to view tabular data
- Become familiar with constructing a for loop in Python

Signing onto Biowulf

Sign onto Biowulf using the `ssh` command. Replace username with user's Biowulf ID.

```
ssh username@biowul.nih.gov
```

Change into data directory and copy course data

Replace username with user's Biowulf ID.

```
cd /data/username
```

The `cp` command below will copy `pies_2023_data` in `/data/classes/` to the user's data directory (denoted as `.` as this should be present working directory) and save it as a folder called `pies_2023`.

```
cp -r /data/classes/BTEP/pies_2023_data ./pies_2023
```

Change into `pies_2023`.

```
cd pies_2023
```

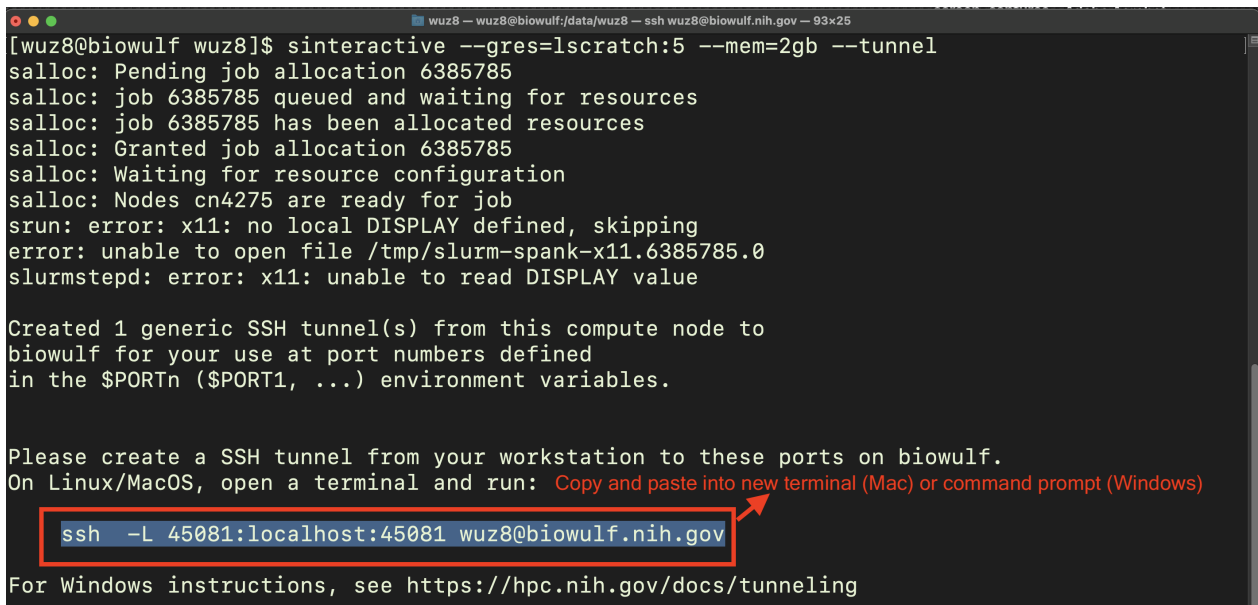
Request interactive session

Stay in the `/data/username/pies_2023` folder and request an interactive session using `sinteractive` with the following options.

- `--gres=lscratch:5`: to allocate 5gb of local temporary/scratch storage space
- `--mem=2gb`: to request 2gb of memory or RAM
- `--tunnel`: to open up a channel of communication between local machine and Biowulf to allow interaction with applications like Jupyter Lab

```
sinteractive --gres=lscratch:5 --mem=2g --tunnel
```

After resources for the interactive session has been granted, users will see the information similar to that shown in Figure 1.



```
wuz8@biowulf wuz8]$ sinteractive --gres=lscratch:5 --mem=2gb --tunnel
salloc: Pending job allocation 6385785
salloc: job 6385785 queued and waiting for resources
salloc: job 6385785 has been allocated resources
salloc: Granted job allocation 6385785
salloc: Waiting for resource configuration
salloc: Nodes cn4275 are ready for job
srun: error: x11: no local DISPLAY defined, skipping
error: unable to open file /tmp/slurm-spank-x11.6385785.0
slurmstepd: error: x11: unable to read DISPLAY value

Created 1 generic SSH tunnel(s) from this compute node to
biowulf for your use at port numbers defined
in the $PORTn ($PORT1, ...) environment variables.

Please create a SSH tunnel from your workstation to these ports on biowulf.
On Linux/MacOS, open a terminal and run: Copy and paste into new terminal (Mac) or command prompt (Windows)

ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov

For Windows instructions, see https://hpc.nih.gov/docs/tunneling
```

Figure 1: After interactive session resources have been allocated, users will see a `ssh` command that looks like that enclosed in the red rectangle. Open a new terminal (if working on a Mac) or command prompt (if working on a Windows computer) and then copy and paste this `ssh` command into the new terminal.

After copying and pasting the `ssh` command shown in Figure 1 to a new terminal or command prompt, hit enter to supply password and log in to Biowulf. This will complete the tunnel.

Hit enter after copying and pasting into a new terminal (Mac) or command prompt (Windows) to provide password and sign onto Biowulf, which will complete the tunnel.

```
(base) NCI-02227565-ML:~ wuz8$ ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov
Enter passphrase for key '/Users/wuz8/.ssh/id_rsa':
Last login: Tue Aug 15 16:24:28 2023 from 10.248.80.125
[wuz8@biowulf ~]$
```

Figure 2: Hit enter after copying and pasting the ssh command to a new terminal to provide password and log into Biowulf. This will complete the tunnel.

Will vary for each user

```
ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov
```

Biowulf username will vary for each user

Figure 3: In the ssh command shown in Figure 1 and Figure 2, the numbers preceding and following "localhost" will differ depending on user. Also, the Biowulf username will differ for each user (wuz8 is the instructor's Biowulf username).

Load Jupyter

Warning

Make sure to stay in the /data/username/pies_2023 folder for this step.

After the tunnel has been created, go back terminal (Mac) or command prompt (Windows) with the Biowulf interactive session and activate Jupyter (see Figure 4).

```
module load jupyter
```

```

salloc: job 6385785 queued and waiting for resources
salloc: job 6385785 has been allocated resources
salloc: Granted job allocation 6385785
salloc: Waiting for resource configuration
salloc: Nodes cn4275 are ready for job
srun: error: x11: no local DISPLAY defined, skipping
error: unable to open file /tmp/slurm-spank-x11.6385785.0
slurmstepd: error: x11: unable to read DISPLAY value

Created 1 generic SSH tunnel(s) from this compute node to
biowulf for your use at port numbers defined
in the $PORTn ($PORT1, ...) environment variables.

Please create a SSH tunnel from your workstation to these ports on biowulf.
On Linux/MacOS, open a terminal and run:

    ssh -L 45081:localhost:45081 wuz8@biowulf.nih.gov

For Windows instructions, see https://hpc.nih.gov/docs/tunneling

[wuz8@cn4275 wuz8]$ module load jupyter
[+] Loading git 2.39.2 ...
[+] Loading jupyter
[wuz8@cn4275 wuz8]$

```

Figure 4: Go back to the terminal (Mac) or command prompt (Windows) with the interactive session (look for `cn####` at the prompt). Do `module load jupyter` from here.

Start Jupyter Lab

Warning

Make sure to stay in the `/data/username/pies_2023` folder for this step.

Use the command below to start a Jupyter Lab session. Copy and paste either of the http links to a local browser to interact with Jupyter (see Figure 5).

```
jupyter lab --ip localhost --port $PORT1 --no-browser
```

```

[wuz8@cn4275 wuz8]$ jupyter lab --ip localhost --port $PORT1 --no-browser
To access the server, open this file in a browser:
  file:///spin1/home/linux/wuz8/.local/share/jupyter/runtime/jpserver-363837-open.html
Or copy and paste one of these URLs:
  http://localhost:45081/lab?token=ad4b828f83a0fd8ad468cadaed56590b8a34f7f0418e76f3
  or http://127.0.0.1:45081/lab?token=ad4b828f83a0fd8ad468cadaed56590b8a34f7f0418e76f3

```

Copy
either
of the
http
links to
local
browser

Figure 5: Start a Jupyter lab session using `jupyter lab --ip localhost --port $PORT1 --no-browser` and copy and paste either one of the http links to a local browser.

Python data types and data structures

An important step to learning any new programming language and data analysis is to understand its data types and data structures. Common data types and structures that will be encountered include the following.

- Text (str)
- Numeric
 - int (ie. integers)
 - float (ie. decimals)
- Boolean (True or False)
 - conditionals
 - filtering criteria
 - command options
- Data frames
- Lists
- Arrays
- Tuples
- Range
- Dictionaries

Identifying data type and structure in Python

The command `type` can be used to identify data types and structures in Python.

```
type(100)
```

```
int
```

```
type(3.1415926)
```

```
float
```

```
type("bioinformatics")
```

```
str
```

Variable assignments

In Python, variables are assigned to values using "=". Users can assign variables to integers, float, or string.

```
perfect=100  
perfect
```

```
100
```

```
mole=6.02e23  
mole
```

```
6.02e+23
```

```
btep_class="Python Introductory Education Series"  
btep_class
```

```
'Python Introductory Education Series'
```

The command `type(btep_class)` will return `str` because the variable `btep_class` is text.

```
type(btep_class)
```

```
str
```

Conditionals

Conditionals evaluate the validity of certain conditions and operators include:

- `==`: is equal to?
- `>`: is greater than?
- `>=`: is greater than or equal to?
- `<`: is less than?
- `<=`: is less than or equal to?

- `!=`: is not equal to?
- `and`
- `or`

The command below will evaluate if the variable `perfect` is equal to the variable `mole` and returns the Boolean value, `False`.

```
perfect==mole
```

```
False
```

If statements are also conditionals and are used to instruct the computer to do something if a condition is met. To have the computer do something when the condition is not met, use `elif` (else if) or `else`.

The command below will accomplish the following:

- Use `if` to evaluate if `perfect==mole`, if yes then indicate using `print` that the two variables are equal
- In the case that `perfect` does not equal `mole`, use `elif` (which stands for else if) to evaluate if `perfect>mole`, if yes then use the `print` statement to indicate that `perfect` is greater than `mole`
- `else` when the previous two conditions are not met, use `print` to indicate that `perfect` is less than `mole`

```
if perfect==mole:
    print(perfect, "is equal to", mole)
elif perfect>mole:
    print(perfect, "is greater than", mole)
else:
    print(perfect, "is less than", mole)
```

```
100 is less than 6.02e+23
```

Note

The `print` command can be used to print variables by not enclosing in quotes.

A `:` is required after `if`, `elif`, and `else`. The command(s) to execute when conditions are met are placed on a separate line but tab indented.

Data frames

Often, in bioinformatics and data science, data comes in the form of rectangular tables, which are referred to as data frames. Data frames have the following property.

- Study variable(s) form the columns
- Observation(s) form rows
- Can have a mix of data types (strings and numeric) but **each column/study variable can contain only one data type**
- Limited to one value per cell

A popular package for working with data frames in Python is **Pandas** (<https://pandas.pydata.org>).

To load a Python package use the `import` command followed by the package name (ie. pandas).

```
import pandas
```

Sometimes the name of the package is long, so users might want to shorten it by creating an alias. The alias "pd" is often used for the Pandas package. To add an alias, just append `as` followed by the user defined alias to the package import command.

```
import pandas as pd
```

Importing tabular data with Pandas

This exercise will use the `read_csv` function of Pandas to import a comma separated value (csv) file called `hbr_uhr_chr22_rna_seq_counts.csv`, which contains RNA sequencing gene expression counts from the **Human Brain Reference (hbr) and Universal Human Reference (uhr) study** (https://rnabio.org/module-01-inputs/0001/05/01/RNAseq_Data/).

```
hbr_uhr_chr22_counts=pandas.read_csv("./hbr_uhr_chr22_rna_seq_counts
```

Note

If a Python package was imported using an alias (ie. `pd` for Pandas) then use the alias to call the package. For instance, `pd.read_csv` rather than `pandas.read_csv` when the `pd` alias is used for Pandas.

Take note of the way the csv import command is constructed. First the user specifies the name of package (ie. `pandas`) and then the function within the package (ie. `read_csv`). The package name and function name is separated by a period.

Next, use `type` to find out the data type or structure for `hbr_uhr_chr22_counts`.

```
type(hbr_uhr_chr22_counts)
```

```
pandas.core.frame.DataFrame
```

Take a look at the first few rows of `hbr_uhr_chr22_counts`.

```
hbr_uhr_chr22_counts.head()
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam	UHR_3.bam
0	U2	0	0	0	0	0	0
1	CU459211.1	0	0	0	0	0	0
2	CU104787.1	0	0	0	0	0	0
3	BAGE5	0	0	0	0	0	0
4	ACTR3BP6	0	0	0	0	0	0

Figure 1: Example of a data frame.

Because `hbr_uhr_chr22_counts` is a Pandas data frame, it is possible to append one of the many Pandas commands to it. For instance, the `head` function was appended to display the first five rows of `hbr_uhr_chr22_counts`. The data frame name and function is separated by a period. This is perhaps one of the most appealing aspects of Python syntax. Note that the `head` function was followed by `()`. If the parentheses is blank, then by default the first five lines will be shown. There will be more examples of the Pandas `head` function in a subsequent lesson.

Lists and tuples

Lists and tuples are one dimensional collections of data. The tuple is an immutable list, in which the elements cannot be modified.

To create a list, enclose the contents in square brackets.

```
sequencing_list=["whole genome", "rna", "whole exome"]
```

To create a tuple, enclose the contents in parentheses.

```
sequencing_tuple=("whole genome", "rna", "whole exome")
```

Lists and tuples are indexed and can contain duplicates. The first item in a list or tuple has an index of 0, the second item has an index of 1, and the last item has an index of n-1 where n is the number of items. Indices can be used to recall items in a list or tuple.

```
sequencing_list[1]
```

```
'rna'
```

List versus tuples (mutable versus immutable)

```
sequencing_list[1]="single cell RNA"
```

```
sequencing_list
```

```
['whole genome', 'single cell RNA', 'whole exome']
```

```
sequencing_tuple[1]="single cell RNA"
```

```
TypeError                                Traceback (most recent call)
Cell In[48], line 1
----> 1 sequencing_tuple[1]="single cell RNA"

TypeError: 'tuple' object does not support item assignment
```

Instructions for modifying Python lists can be found at the [W3 school \(https://www.w3schools.com/python/python_lists.asp\)](https://www.w3schools.com/python/python_lists.asp)

Arrays

Given a list of numbers, it is difficult to perform mathematical operations. For instance

```
list_of_numbers=[1,2,3,4,5]
```

Multiplying `list_of_numbers` by 2 will duplicate this list. However, multiplying a list of numbers by two should double every number in that list. Thus, the expected result is [2,4,6,8,10]. To resolve this, convert the list to an array using the package [numpy \(https://numpy.org\)](https://numpy.org).

```
list_of_numbers*2
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

Use the `array` function of `numpy` to convert `list_of_numbers` to an array called `array_of_numbers`.

```
array_of_numbers=numpy.array(list_of_numbers)
```

```
array_of_numbers*2
```

```
array([ 2,  4,  6,  8, 10])
```

The array of numbers shown here is a one dimensional array. A special case of arrays is the matrix, which is two dimensional. Like data frames, matrices store values in columns and rows. Matrices are encountered in computation and are used to store numeric values ([see here for more on matrices \(https://youtu.be/IZcyZHomFQc\)](https://youtu.be/IZcyZHomFQc)).

Range

Ranges can be used to for subsetting data (ie. extract data in rows 5 thru 10 of a data frame) or applied to iterate over a task in things like a `for` loop.

For instance, a `for` loop can be used to iterate over `sequencing_list_new` and print the 3rd to 5th entries.

```
sequencing_list_new=["whole genome", "rna", "whole exome","single ce`
```

```
for i in range(2,5):  
    print(sequencing_list_new[i])
```

```
whole exome  
single cell rna  
chip
```

Dictionaries

Dictionaries are key-value pairs and these are encountered as ways to specify options in some Python packages.

```
my_dictionary={"apples":"red","oranges":"orange","bananas":"yellow"}
```

Lesson 3: Data wrangling using Python

Learning objectives

After this lesson, participants will

- Be able to import tabular data into Python using Pandas
- Be able to explore and modify tabular data through various data wrangling approaches, including
 - retrieving dimensions
 - subsetting
 - obtaining column statistics
 - replacing column names
 - performing mathematical operations
 - filtering
 - removing and adding columns

Importing tabular data using Pandas

Pandas (<https://pandas.pydata.org>) is a popular Python package used to work with tabular data.

To work with Pandas, first activate it using the `import` command.

```
import pandas
```

Sometimes the name of the package is long, so users might want to shorten it by creating an alias. The alias "pd" is often used for the Pandas package. To add an alias, just append `as` followed by the user defined alias to the package import command. If importing a package using an alias, then the package needs to be called using the assigned alias. For instance, if `pd` was used to import `pandas`, then use `pd.read_csv` to import a csv file.

```
import pandas as pd
```

This exercise will use the `read_csv` function of Pandas to import a comma separated value (csv) file called `hbr_uhr_chr22_rna_seq_counts.csv`, which contains RNA sequencing gene expression counts from the **Human Brain Reference (hbr) and Universal Human Reference (uhr) study** (https://rnabio.org/module-01-inputs/0001/05/01/RNAseq_Data/). This data will be stored as the variable `hbr_uhr_chr22_counts`.

```
hbr_uhr_chr22_counts=pandas.read_csv("./hbr_uhr_chr22_rna_seq_counts
```

Take a look at the first few rows of `hbr_uhr_chr22_counts` by appending the `head` attribute to `hbr_uhr_chr22_counts`.

```
hbr_uhr_chr22_counts.head()
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam	UHR_3.bam
0	U2	0	0	0	0	0	0
1	CU459211.1	0	0	0	0	0	0
2	CU104787.1	0	0	0	0	0	0
3	BAGE5	0	0	0	0	0	0
4	ACTR3BP6	0	0	0	0	0	0

Figure 1: The first five rows of `hbr_uhr_chr22_counts`. The first column contains genes and the subsequent columns contain gene expression counts for each of the samples. The left most column of this data frame contains the row indices or names.

Because `hbr_uhr_chr22_counts` is a Pandas data frame (`type(hbr_uhr_chr22_counts)`, see lesson 2), it is possible to append one of the many Pandas commands to it. For instance, the `head` function was appended to display the first five rows of `hbr_uhr_chr22_counts`. The data frame name and function is separated by a period. This is perhaps one of the most appealing aspects of Python syntax. Note that the `head` function was followed by `()`. If the parentheses are blank, then the default first five lines will be shown. To view the first 10 rows of `hbr_uhr_chr22_counts` do the following.

```
hbr_uhr_chr22_counts.head(10)
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam	UHR_3.bam
0	U2	0	0	0	0	0	0
1	CU459211.1	0	0	0	0	0	0
2	CU104787.1	0	0	0	0	0	0
3	BAGE5	0	0	0	0	0	0
4	ACTR3BP6	0	0	0	0	0	0
5	5_8S_rRNA	0	0	0	0	0	0
6	AC137488.1	0	0	0	0	0	0
7	AC137488.2	0	0	0	0	0	0
8	CU013544.1	0	0	0	0	0	0
9	CT867976.1	0	0	0	0	0	0

Figure 2: Include an integer inside the parentheses of `pandas.DataFrame.head()` function to view the specified number of lines in a tabular dataset.

The function `tail` can be used to view by default the bottom five lines of a tabular dataset. Similar to `head`, the number of lines shown can be customized by specifying an integer inside the parentheses.

```
hbr_uhr_chr22_counts.tail()
```

Get dimensions of a data frame

Pandas data frames have a function `shape` that informs of the number of rows and number of columns in a data frame (in other words the dimensions). To get the dimensions for `hbr_uhr_chr22_counts`, do the following

```
hbr_uhr_chr22_counts.shape
```

The `hbr_uhr_chr22_counts` data frame has 1335 rows and 7 columns.

```
(1335, 7)
```

Note

The elements in tabular data can be referred to by their row and column positions.

The `size` function returns the number elements in a data frame. For instance, `hbr_uhr_chr22_counts` has 1335 rows and 7 columns, which means that it has 1335 times 7 elements (or 9345).

Row indices/names

Figure 2 shows the first 10 rows of `hbr_uhr_chr22_counts`. The left most column, which contains labels starting with "0" is referred to as the row indices or row names. Users can specify a column in the dataset as the row indices or row names using the `index_col` options in `read_csv`. For instance, the `hbr_uhr_chr22_rna_seq_counts.csv` dataset could be imported with gene names as the row indices. To do this, add the `index_col=0` option to `read_csv`. Gene names in `hbr_uhr_chr22_rna_seq_counts.csv` is the first column and is denoted as column "0" in Python. Thus, setting `index_col=0` ensures that the gene names will be set as the row indices or row names (see Figure 3).

```
hbr_uhr_chr22_counts_1=pandas.read_csv("./hbr_uhr_chr22_rna_seq_count
```

	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam	UHR_3.bam
Geneid						
U2	0	0	0	0	0	0
CU459211.1	0	0	0	0	0	0
CU104787.1	0	0	0	0	0	0
BAGE5	0	0	0	0	0	0
ACTR3BP6	0	0	0	0	0	0
...
ACR	0	0	0	0	2	0
AC002056.5	0	0	0	0	0	0
AC002056.3	0	0	0	0	0	0
RPL23AP82	41	59	54	32	23	34
RABL2B	74	62	54	68	50	47

Figure 3. The `index_col=0` option in `pandas.read_csv` sets the gene names as row names in the imported data frame.

Data wrangling

Subsetting

The command below will subset the expression counts for the RABL2B gene.

```
hbr_uhr_chr22_counts[hbr_uhr_chr22_counts["Geneid"]=="RABL2B"]
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam
1334	RABL2B	74	62	54	68	54

The "|" symbol can be used as the "or" operator so to also subset the counts for RPL23AP82

```
hbr_uhr_chr22_counts[(hbr_uhr_chr22_counts["Geneid"]=="RABL2B") | (hbr_uhr_chr22_counts["Geneid"]=="RPL23AP82")]
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam
1333	RPL23AP82	41	59	54	32	54
1334	RABL2B	74	62	54	68	54

Alternatively, use the `isin` function and provide a list of genes to retrieve.

```
hbr_uhr_chr22_counts[hbr_uhr_chr22_counts["Geneid"].isin(["RABL2B", "RPL23AP82"])]
```

Use "." to reference a column.

```
hbr_uhr_chr22_counts[hbr_uhr_chr22_counts.Geneid=="RABL2B"]
```

Subsetting by integer positions

Given that the elements in a data frame are referenced by its row and column positions, what would be the approach for extracting the element in row 60 and column 5? The solution is the command below, which returns a result of 2. The row and column numbers are enclosed in "[]" and separated by a comma.

```
hbr_uhr_chr22_counts.iloc[60,5]
```

```
2
```

The above method for subsetting the element in row 60 and column 5 of `hbr_uhr_chr22_counts` is great if the goal is to extract the value and do numeric operation on it. But what if the user wants to return the element along with the corresponding gene in data frame format?

To do this, enclose the row and column indices to extract in their own inner set of square brackets as shown below. Column 0, which contains the gene name is also included in the brackets containing the column indices of interest.

```
hbr_uhr_chr22_counts.iloc[[60],[0,5]]
```

	Geneid	UHR_2.bam
60	CCT8L2	2

Pandas offers different approaches for subsetting rectangular data. One method is `iloc`.

`iloc` is a "purely integer-location based indexing for selection by position" -- <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html#> (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html#>). The row and column positions are enclosed in "[]".

`iloc` allows for retrieval of elements in multiple rows and columns. For instance, the following can be used to retrieve the elements in rows 60 and 65 and columns 0, 4, 5, and 6 in `hbr_uhr_chr22_counts`. Note that the row and column positions are enclosed in an outer set of "[]". Within the outer set of "[]" the first set of "[]" enclose a comma separated list of row positions while the second set of "[]" enclose a comma separated list of column positions.

```
hbr_uhr_chr22_counts.iloc[[60,65],[0,4,5,6]]
```

	Geneid	UHR_1.bam	UHR_2.bam	UHR_3.bam
60	CCT8L2	1	2	0
65	SLC25A15P5	2	2	4

To get the first three rows of `hbr_uhr_chr22_counts` do the following. Note that it retrieves the rows with indices 0, 1, and 2.

```
hbr_uhr_chr22_counts.iloc[:3]
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bar
0	U2	0	0	0	0	0
1	CU459211.1	0	0	0	0	0
2	CU104787.1	0	0	0	0	0

What will be the output for `hbr_uhr_chr22_counts.iloc[[3],:]`?

{{Sdet}}

Solution{{Esum}}

The row with an index of 3 will be retrieved.

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bar
3	BAGE5	0	0	0	0	0

{{Edet}}

Subsetting using column names

Panda's `loc` function allows for subsetting by row or column names. For instance, to retrieve the gene id column, do the following. The ":" denotes get every row.

```
hbr_uhr_chr22_counts.loc[:, ['Geneid']]
```

```

Geneid
0      U2
1      CU459211.1
2      CU104787.1
3      BAGE5
4      ACTR3BP6
...    ...
1330   ACR
1331   AC002056.5
1332   AC002056.3
1333   RPL23AP82
1334   RABL2B

```

To retrieve the counts for the gene `SLC25A15P5`, use the following where `SLC25A15P5` is the subsetting criteria, where

- `hbr_uhr_chr22_counts.loc[:, 'Geneid']` extracts the Geneid column.
- `=="SLC25A15P5"` will filter out the row with the `SLC25A15P5` gene.

```
hbr_uhr_chr22_counts[hbr_uhr_chr22_counts.loc[:, 'Geneid'] == "SLC25A15P5"]
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam
65	SLC25A15P5	0	0	0	2	0

To retrieve counts for more than one gene, enclose the genes of interest in a list and use the `isin` function to filter out the rows containing the genes in the list.

```
hbr_uhr_chr22_counts[hbr_uhr_chr22_counts.loc[:, 'Geneid'].isin(["SLC25A15P5", "CCT8L2"])]
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam
60	CCT8L2	0	0	0	1	0
65	SLC25A15P5	0	0	0	2	0

To find all of the SLC genes in `hbr_uhr_chr22_counts`, the following could be used where `str.startswith` searches for text that starts a pattern (ie. "SLC"). Other options for pattern matching include `str.endswith` and `str.contains`.

```
hbr_uhr_chr22_counts.loc[hbr_uhr_chr22_counts.loc[:, 'Geneid'].str.startswith("SLC")]
```

	Geneid	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam
54	SLC9B1P4	0	0	0	0	0
65	SLC25A15P5	0	0	0	2	0
109	SLC25A18	100	111	74	6	0
181	SLC25A1	32	50	41	226	0
249	SLC9A3P2	0	0	0	0	0
268	SLC7A4	19	25	14	9	0
494	SLC2A11	54	63	46	28	0
726	SLC35E4	18	32	26	21	0
783	SLC5A1	0	0	0	0	0
795	SLC5A4	7	12	5	13	0
955	SLC16A8	9	13	11	11	0
1046	SLC25A17	39	39	40	119	0
1099	SLC25A5P1	0	0	1	0	0

Summary statistics of data frames

```
hbr_uhr_chr22_counts.describe()
```

	HBR_1.bam	HBR_2.bam	HBR_3.bam	UHR_1.bam	UHR_2.bam	UHR_3.bam
count	1335.000000	1335.000000	1335.000000	1335.000000	1335.000000	1335.000000
mean	29.530337	36.264419	32.084644	50.694382	33.419476	33.419476
std	99.177874	120.617793	108.237694	197.575081	122.598310	122.598310
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000
75%	8.000000	10.000000	9.000000	13.000000	12.000000	12.000000
max	1532.000000	1797.000000	1637.000000	4027.000000	2406.000000	2406.000000

Replacing column names

To view the column headings of a data frame use the `columns` function. For instance,

```
hbr_uhr_chr22_counts.columns
```

```
HBR_1.bam
HBR_2.bam
HBR_3.bam
UHR_1.bam
UHR_2.bam
UHR_3.bam
```

The `str.replace` function can be used to replace a string with something else. Here, it is used to remove ".bam" from the sample names in the column heading.

```
hbr_uhr_chr22_counts.columns=hbr_uhr_chr22_counts.columns.str.replace
```

Mathematical operations on data frames and filtering

Pandas enables mathematical operations on data frames. For instance, one might want to sum the total counts across all samples for each gene. The `sum` function can be used to do this. Setting `axis=1` will sum up the counts for each row or gene. Because the `Geneid` column is a string, it is necessary to first subset only the sample columns.

```
hbr_uhr_chr22_counts.loc[:, ['HBR_1', 'HBR_2', 'HBR_3', 'UHR_1', 'UHR_2', 'UHR_3']].sum(axis=1)
```

Below, genes with zero counts across all samples are removed from `hbr_uhr_chr22_counts` and stored as `hbr_uhr_chr22_counts_filtered`. To accomplish this set `hbr_uhr_chr22_counts.loc[:, ['HBR_1', 'HBR_2', 'HBR_3', 'UHR_1', 'UHR_2', 'UHR_3']].sum(axis=1) != 0` and use as a filter criteria.

```
hbr_uhr_chr22_counts_filtered=hbr_uhr_chr22_counts.loc[hbr_uhr_chr22_
```

Removing and adding columns to a data frame

For this exercise, stay in the `/data/username/pies_2023` folder, which should be the present working directory (use `pwd` to check). If not in the `/data/username/pies_2023` folder, change into it. Copy the `hbr_uhr_deg_chr22.csv` and `hcc1395_deg_chr22.csv` files from `/data/classes/BTEP/pies_2023_data` to the `/data/username/pies_2023` directory.

```
cp /data/classes/BTEP/pies_2023_data/hbr_uhr_deg_chr22.csv .
```

```
cp /data/classes/BTEP/pies_2023_data/hcc1395_deg_chr22.csv .
```

The file `hcc1395_deg_chr22.csv` will be needed for the practice questions.

This exercise will use the differential gene expression analysis table from the hbr and uhr study.

```
hbr_uhr_deg_chr22=pandas.read_csv("./hbr_uhr_deg_chr22.csv")
```

The `info()` function will retrieve information regarding the `hbr_uhr_deg_chr22` data frame, which includes the column names.

```
hbr_uhr_deg_chr22.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1335 entries, 0 to 1334
Data columns (total 18 columns):
#   Column                Non-Null Count  Dtype
---  -
0   name                   1335 non-null   object
1   baseMean               1335 non-null   float64
2   baseMeanA              1335 non-null   float64
3   baseMeanB              1335 non-null   float64
4   foldChange             971 non-null    float64
5   log2FoldChange         971 non-null    float64
```

```

6   lfcSE          971 non-null    float64
7   stat           971 non-null    float64
8   PValue        971 non-null    float64
9   PAdj          971 non-null    float64
10  FDR           639 non-null    float64
11  falsePos      639 non-null    float64
12  HBR_1.bam     1335 non-null   float64
13  HBR_2.bam     1335 non-null   float64
14  HBR_3.bam     1335 non-null   float64
15  UHR_1.bam     1335 non-null   float64
16  UHR_2.bam     1335 non-null   float64
17  UHR_3.bam     1335 non-null   float64
dtypes: float64(17), object(1)
memory usage: 187.9+ KB

```

The `hbr_uhr_deg_chr22` table contains differential gene expression analysis results. Relevant columns include

- `name`: gene names
- `log2FoldChange`: the gene expression change between the two treatment groups
- `PAdj`: the adjusted p-value associated with statistical confidence of the expression change
- The columns labeled with the sample names (ie. columns 12 through 17) are the normalized gene expression counts

Use `str.replace` to remove `".bam"` from the sample names in columns 12 through 17.

```
hbr_uhr_deg_chr22.columns=hbr_uhr_deg_chr22.columns.str.replace(".bam", "")
```

To drop columns in a Pandas data frame, use the `.drop` function and specify the name(s) of the column(s) to remove. The example below removes columns `baseMean`, `baseMeanA`, and `baseMeanB`

```
hbr_uhr_deg_chr22.drop(columns=["baseMean", "baseMeanA", "baseMeanB"])
```

Subset the `name`, `log2FoldChange`, and `PAdj` columns in `hbr_uhr_deg_chr22` and save to a new data frame `hbr_uhr_deg_chr22_1`.

```
hbr_uhr_deg_chr22_1=hbr_uhr_deg_chr22.loc[:,["name", "log2FoldChange", "PAdj"]]
```

```
hbr_uhr_deg_chr22_1.head()
```


	name	log2FoldChange	PAdj
0	SYNGR1	-4.6	5.200000e-217
1	SEPT3	-4.6	4.500000e-204
2	YWHAH	-2.5	4.700000e-191
3	RPL3	1.7	5.400000e-134
4	PI4KA	-2.0	2.900000e-118

Next, add a column called "-log10PAdj" to `hbr_uhr_deg_chr22_1`, which will contain the negative of log10 of the values in the PAdj column. "-log10PAdj" is used in volcano plots that depict gene expression change versus statistical confidence. To calculate -log10PAdj, the package `numpy` will be used. [Numpy \(https://numpy.org\)](https://numpy.org) enables scientific calculations.

```
import numpy
```

```
hbr_uhr_deg_chr22_1["-log10PAdj"]=numpy.negative(numpy.log10(hbr_uhr_
```

Take a look at the first several lines of `hbr_uhr_deg_chr22_1`

```
hbr_uhr_deg_chr22_1.head()
```

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

Other methods for adding new column to a Pandas data frame include `insert` and `assign`.

The final task for this lesson is to add a column that indicates whether a gene is up regulated, down regulated, or has no change based on the `log2FoldChange` and `PAdj` values. The criteria are as follows.

- `PAdj >= 0.01`: no change (marked as `ns` in the column)
- Absolute value of `log2FoldChange < 2`: no change (marked as `ns` in the column)
- `log2FoldChange >= 2` and `PAdj < 0.01`: (up regulated)
- `log2FoldChange <= 2` and `PAdj < 0.01`: (down regulated)

To code this in Python, the first step is to drop the NA values from the `hbr_uhr_deg_chr22_1` using `dropna`.

```
hbr_uhr_deg_chr22_1=hbr_uhr_deg_chr22_1.dropna()
```

Next, create a list called `significance_criteria` that contains the criteria shown above. In the criteria list below, "&" is the Boolean for "and". To calculate the absolute value of `log2FoldChange`, `numpy.absolute` is used.

```
significance_criteria=[(hbr_uhr_deg_chr22_1["PAdj"]>=0.01),
                       (numpy.absolute(hbr_uhr_deg_chr22_1["log2FoldChange"])>=2) & (
hbr_uhr_deg_chr22_1["log2FoldChange"]>=2) & (
hbr_uhr_deg_chr22_1["log2FoldChange"]<=-2) & (
```

Then, create a list called `significance_status` that indicates whether the criteria are ns (not significant), up, or down. These statuses have to correspond to the order in which the criteria were listed in `significance_criteria`.

```
significance_status=["ns","ns","up","down"]
```

Finally, `numpy.select` will be used to assign values to the significance column.

```
hbr_uhr_deg_chr22_1["significance"]=numpy.select(significance_criteria,
```

```
hbr_uhr_deg_chr22_1.head(4)
```

	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

Write this data frame to a csv file in the `/data/username/pies_2023` folder, which should be the present working directory. Replace `username` with the user's Biowulf account ID. The `to_csv` command in Pandas is used to write data frames to csv files. Setting `index=False` ensures that the csv file will not have row names.

```
hbr_uhr_deg_chr22_1.to_csv("./hbr_uhr_deg_chr22_with_significance_le
```

This lesson has shown the participants various data wrangling approaches using the Python package Pandas. The capability of Pandas expand to more than what is covered here, participants are encouraged to check out the [Pandas documentations \(https://pandas.pydata.org/docs/\)](https://pandas.pydata.org/docs/) to learn more.

Lesson 4: Data visualization using Python

Learning objectives

This lesson will provide participants with enough knowledge to start using Python for data visualization. Specifically, participants should

- Be able to use the package Seaborn to
 - Construct plots that range from very basic to elegant as well as biologically relevant
 - Customize plots including altering font size and adding custom annotations

Python data visualization tools

Seaborn (<https://seaborn.pydata.org>) is a popular Python plotting package, which is the tool that will be introduced in this lesson. Seaborn is an extension of and builds on *Matplotlib* (<https://matplotlib.org>) and is oriented towards statistical data visualization. However, there are other packages, including those that are domain specific, implement grammar of graphics, and are used for creating web-based visualization dashboards. A non-exhaustive list of Python plotting packages is shown below.

- *Matplotlib* (<https://matplotlib.org>)
- *Plotnine*: implements grammar of graphics for those familiar with R's *ggplot2* (<https://plotnine.readthedocs.io/en/stable/>)
- *bioinfokit*: genomic data visualization (<https://github.com/reneshbedre/bioinfokit>)
- *pygenomeviz*: visualize comparative genomics data (<https://moshi4.github.io/pyGenomeViz/>)
- *Dash bio*: create interactive data visualizations and web dashboards (<https://dash.plotly.com/dash-bio>)

Visualization using Seaborn

Load packages

```
import pandas
import numpy
import matplotlib.pyplot as plt
import seaborn
```

Modify the basic plot elements with Seaborn.

To plot using Seaborn, start the command with `seaborn` followed by the plot type, separated by a period.

```
seaborn.plot_type
```

This section will use Seaborn's `scatterplot` to explore how to work with and modify basic elements of plotting. The foundations learned in this section form the basis for creating advanced and elegant plots.

The data that will be plotted is a point located at 5 on the x axis and 5 on the y axis. To generate x and y, `numpy.array` was used. Here, x and y are single element arrays that store the number 5.

```
x=numpy.array([5])  
y=numpy.array([5])
```

Plot x and y using Seaborn's `scatterplot` function (see Figure 1 for results), which takes data frames or Numpy arrays as input. Here, x will be plotted on the x axis, and y will be plotted on the y axis. The plot can be stored as a variable, which in this example is `plot0`.

```
plot0=seaborn.scatterplot(x=x, y=y)  
plt.show()
```

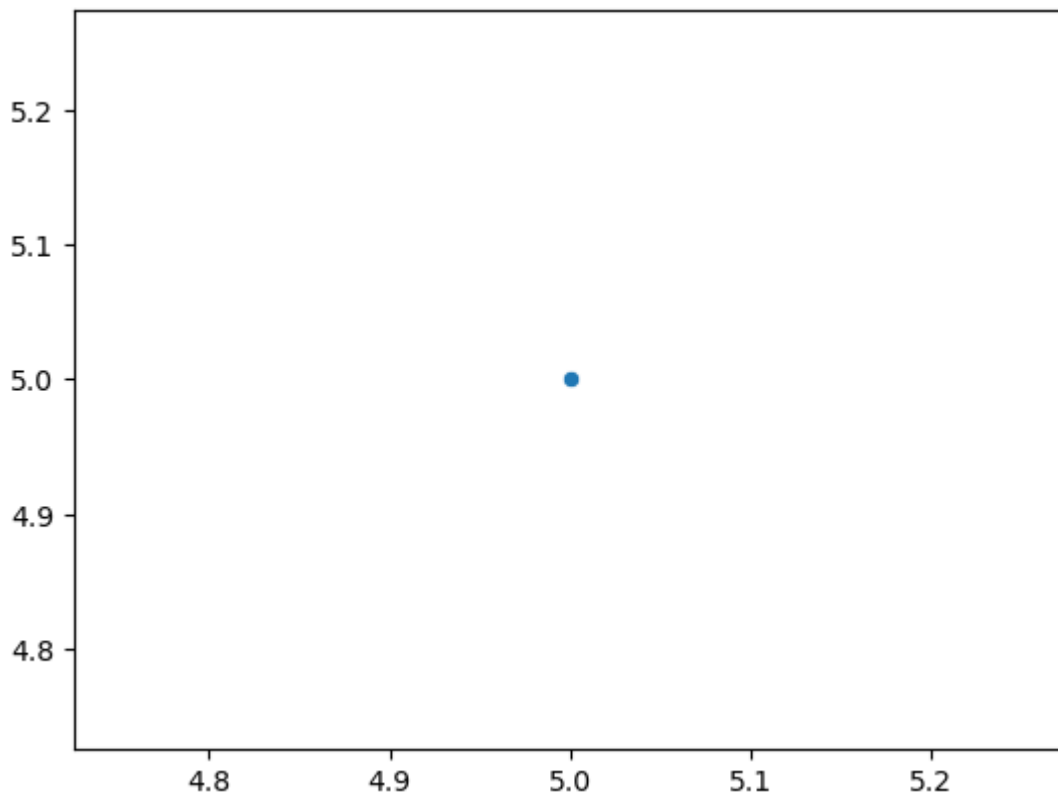


Figure 1

The plot in Figure 1 has no axes labels. Axes labels are an integral part of an informative data visualization. It might also be useful to include meaningful x and y limits. To do this, append the various `.set*` attributes to the plot. See Figure 2a for result.

- `set_xlabel`: specify x axis label (`size` is used to set the label font size)
- `set_ylabel`: specify y axis
- `set_xlim`: sets the x axis limits
- `set_ylim`: sets the y axis limits
- `set_xticks`: sets the location of x axis tick marks
- `set_xticklabels`: sets the x axis tick mark labels, `size` is used to set the tick mark label font size
- `set_yticks`: sets the location of y axis tick marks
- `set_yticklabels`: sets the y axis tick mark labels, `size` is used to set the tick mark label font size

```
plot0=seaborn.scatterplot(x=x, y=y)
plot0.set_xlabel("x axis", size=14)
plot0.set_ylabel("y axis", size=14)
plot0.set_xlim(0,10)
plot0.set_ylim(0,10)
plot0.set_xticks([0,2,4,6,8,10])
plot0.set_xticklabels(labels=["0","2","4","6","8","10"], size=15)
```

```
plot0.set_yticks([0,2,4,6,8,10])
plot0.set_yticklabels(labels=["0","2","4","6","8","10"], size=15)
plt.show()
```

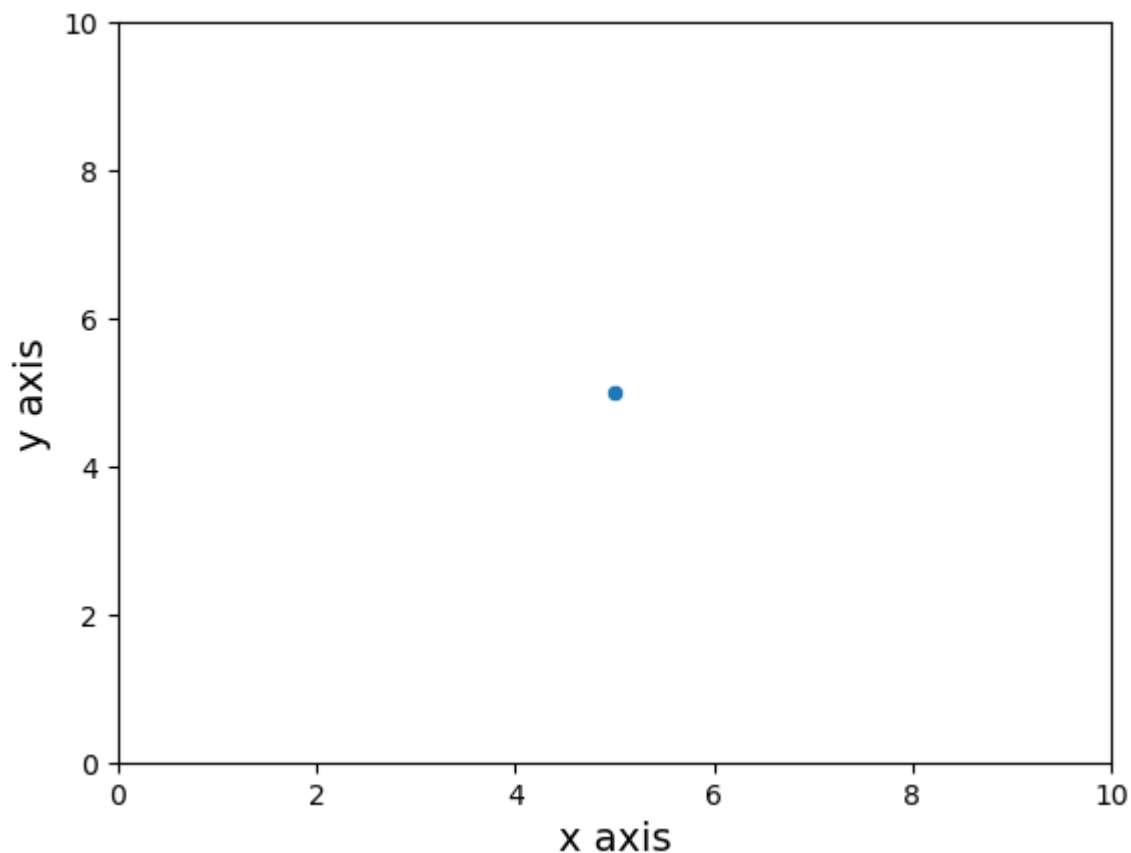


Figure 2

The `plotting_context` of a Seaborn plot contains parameters that determine scaling of plot elements (see https://seaborn.pydata.org/generated/seaborn.plotting_context.html (https://seaborn.pydata.org/generated/seaborn.plotting_context.html)). To view these parameters, do the following, which will return the plot scaling parameters as a dictionary.

```
print(seaborn.plotting_context())
```

```
{'font.size': 12.0, 'axes.labelsize': 12.0, 'axes.titlesize': 12.0,
```

These parameters can be changed using the `set_context` function by providing a customized dictionary and assigning it to the `rc` argument.

```
help(seaborn.set_context)
```

Help on function `set_context` in module `seaborn.rcmod`:

```
set_context(context=None, font_scale=1, rc=None)
```

Set the parameters that control the scaling of plot elements.

This affects things like the size of the labels, lines, and other of the plot, but not the overall style. This is accomplished using `matplotlib rcParams` system.

The base context is "notebook", and the other contexts are "paper" and "poster", which are version of the notebook parameters scaled values. Font elements can also be scaled independently of (but relative to) the other values.

See `:func:`plotting_context`` to get the parameter values.

Parameters

`context` : dict, or one of {paper, notebook, talk, poster}

A dictionary of parameters or the name of a preconfigured set

`font_scale` : float, optional

Separate scaling factor to independently scale the size of the font elements.

`rc` : dict, optional

Parameter mappings to override the values in the preset seaborn context dictionaries. This only updates parameters that are considered part of the context definition.

To change the x and y axes tick label font size to 20, use `seaborn.set_context(rc={'xtick.labelsize': 20, 'ytick.labelsize': 20})` prior to constructing a Seaborn plot.

The code above can be modified to generate a more complex scatter plot that has more points. For instance, the inputs for x and y can be changed to numeric arrays of five 6 elements each.

```
x=numpy.array([0,1,2,3,4,5])
y=numpy.multiply(2,x)
print("x is a numeric array composed of: ", x)
print("y is a numeric array composed of: ", y)
```

```
x is a numeric array composed of: [0 1 2 3 4 5]
y is a numeric array composed of: [ 0  2  4  6  8 10]
```


The code used to generate Figure 2 can then be run again with modifications to the x and y axes limits to generate the plot shown in Figure 3. To produce a line plot representation of Figure 3, simply change the plot type to `lineplot` (`seaborn.lineplot`).

```
plot0=seaborn.scatterplot(x=x, y=y)
plot0.set_xlabel("x axis", size=14)
plot0.set_ylabel("y axis", size=14)
plot0.set_xlim(0,6)
plot0.set_ylim(0,12)
plot0.set_xticks([0,2,4,6])
plot0.set_xticklabels(labels=["0","2","4","6"], size=15)
plot0.set_yticks([0,2,4,6,8,10,12])
plot0.set_yticklabels(labels=["0","2","4","6","8","10","12"], size=15)
plt.show()
```

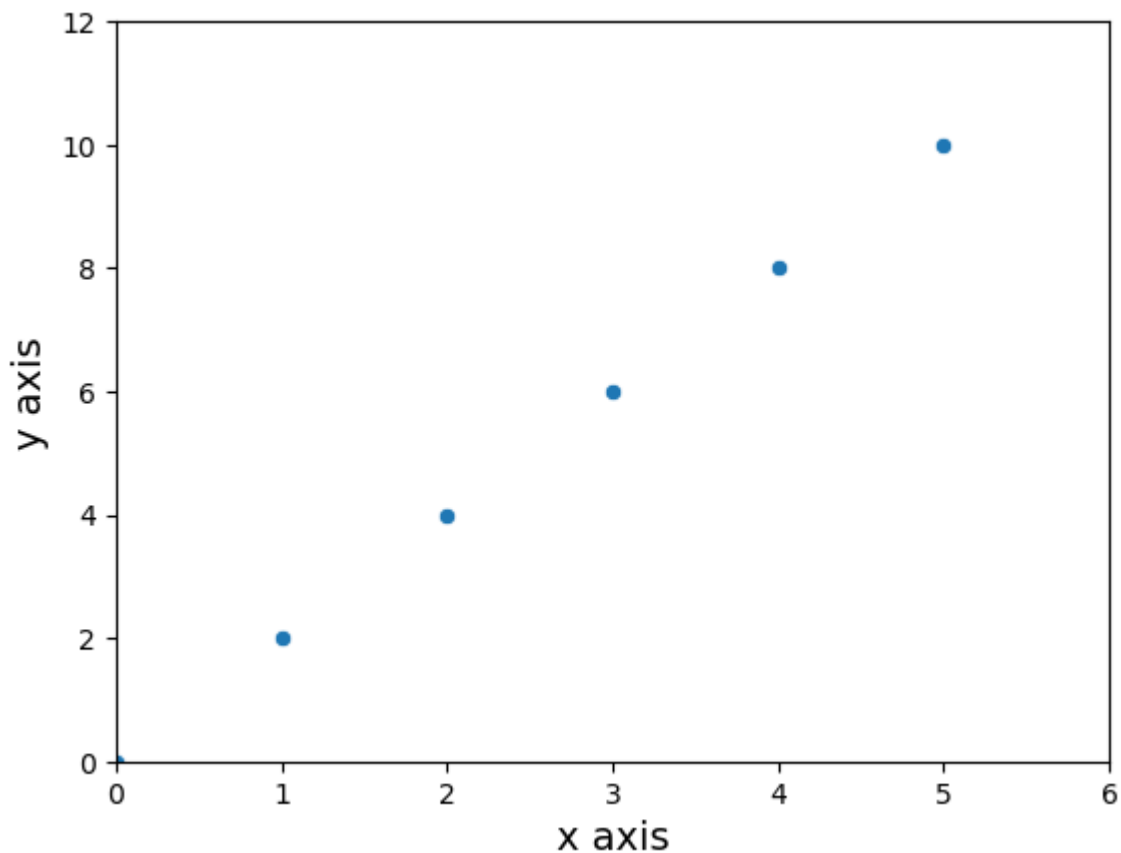


Figure 3

Constructing biologically relevant plots

The next exercise is to practice creating a scatter plot on a biologically relevant dataset. Namely, the differential expression results from the hbr and uhr RNA sequencing study will be used to create a scatter plot depicting log₂ fold change of gene expression on the x axis and

negative \log_{10} of the adjusted p-values on the y axis. This special case of scatter plot is called a volcano plot.

Step one is to import the data using Panda's `read_csv` command.

```
hbr_uhr_deg_chr22=pandas.read_csv("./hbr_uhr_deg_chr22_with_significa
```

Now, review the contents of this data table by doing the following.

```
hbr_uhr_deg_chr22.head(4)
```

	name	log2FoldChange	PAdj	$-\log_{10}PAdj$	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	down

To create the volcano plot, provide the following arguments. See Figure 4 for result.

- The data frame (ie. `hbr_uhr_deg_chr22`)
- What to plot on the x axis (ie. `log2FoldChange`)
- What to plot on the y axis (ie. `"-log10PAdj"`)

```
plot1=seaborn.scatterplot(hbr_uhr_deg_chr22,x="log2FoldChange", y="-
```

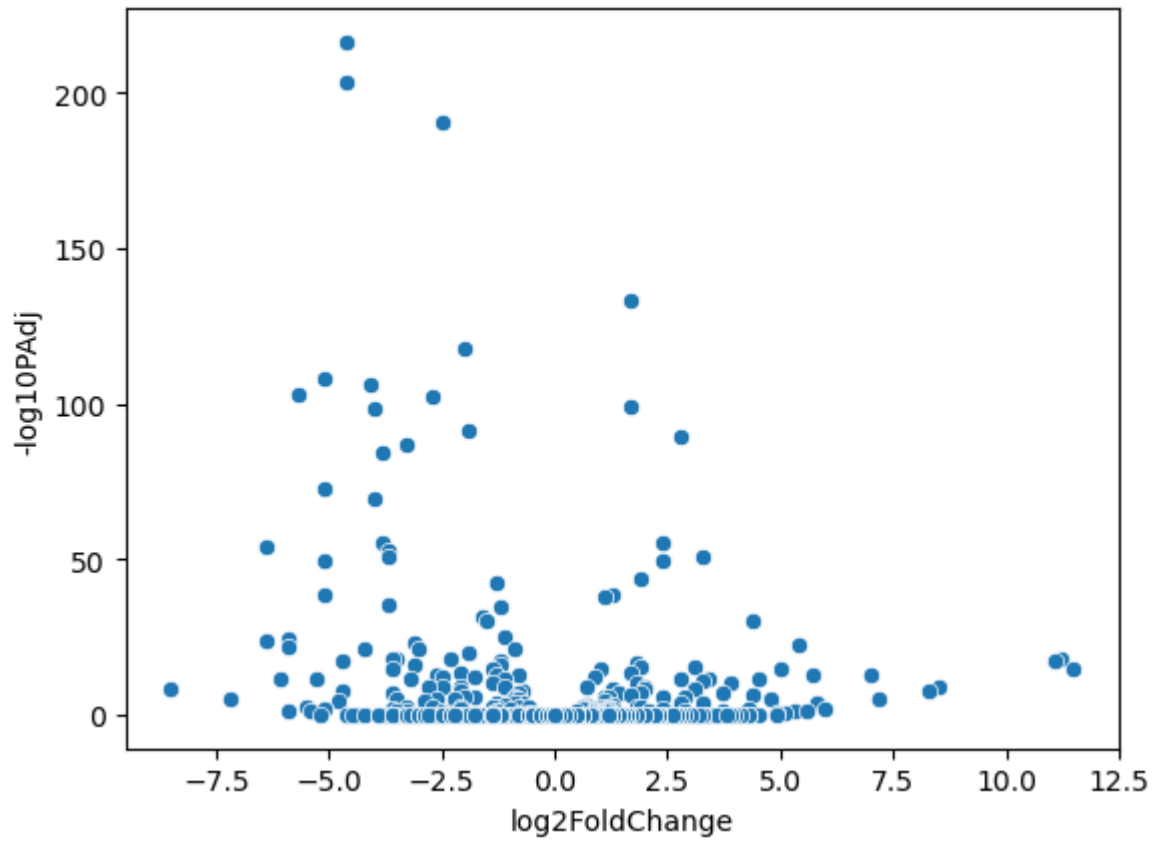


Figure 4

The volcano plot in Figure 4 does not help with visualizing the up, down, or non-significant genes. Fortunately, the hue option can be used to distinguish these. See Figure 5.

```
plot1=seaborn.scatterplot(hbr_uhr_deg_chr22,x="log2FoldChange", y="-log10PAdj", hue="log2FoldChange")
```

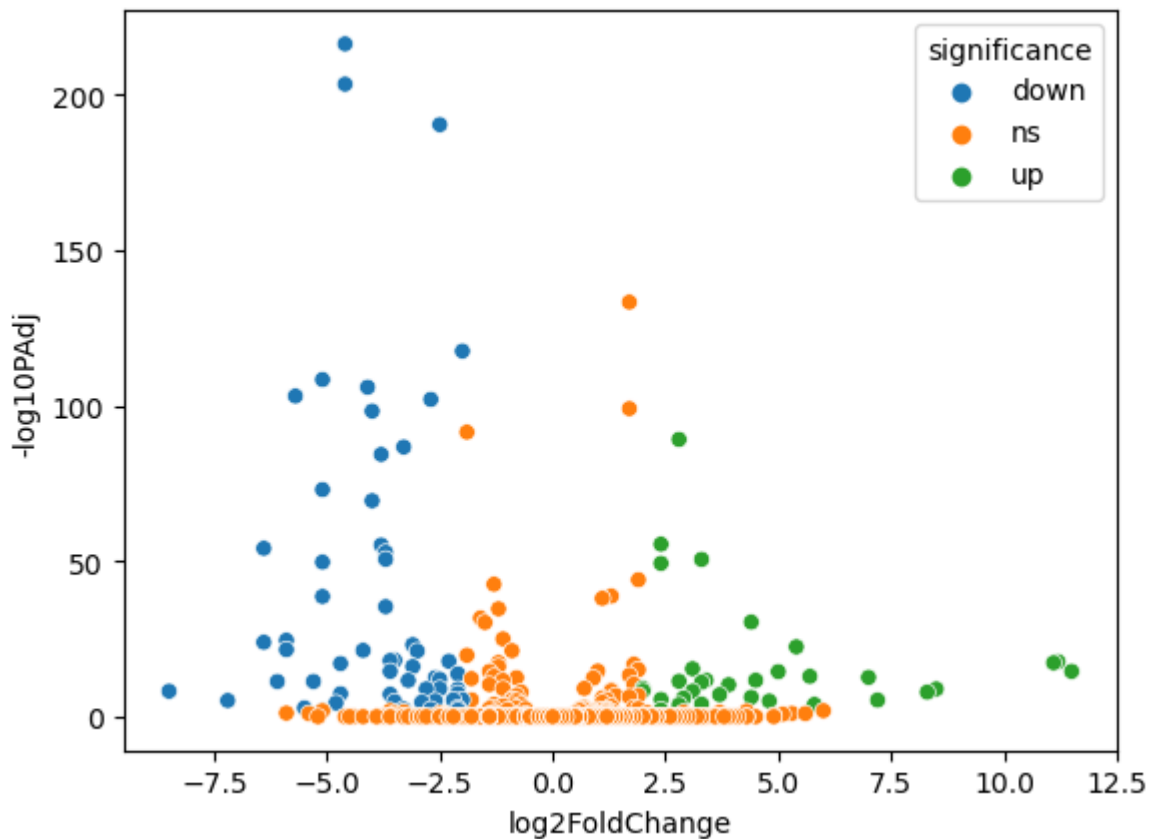


Figure 5

It would be informative to label some of the top significant differentially expressed genes in the volcano plot. To do this, import the file `hbr_uhr_deg_chr22_top_genes.csv` and assign it to the data frame `hbr_uhr_deg_chr22_top_genes`.

```
hbr_uhr_deg_chr22_top_genes=pandas.read_csv("./hbr_uhr_deg_chr22_top_
```

```
hbr_uhr_deg_chr22_top_genes
```

The table contains the top two differentially expressed genes according to the adjusted p-value (PAdj). The task to do is to label the points corresponding to these two genes on the volcano plot. The values for `log2FoldChange` and `-log10PAdj` will serve as the x and y coordinates for plotting the gene name.

	name	log2FoldChange	PAdj	-log10PAdj	significance
0	XBP1	2.8	7.300000e-90	89.136677	up
1	SYNGR1	-4.6	5.200000e-217	216.283997	down

To label the two top differentially expressed genes, start by constructing the volcano plot from Figure 5. Then, use a for loop to iterate through the name column in the data frame `hbr_uhr_deg_chr22_top_genes`. In the for loop

- `i`: the number that keeps track of the row number in the data frame `hbr_uhr_deg_chr22_top_genes` and is used to
 - reference the x coordinate or `log2FoldChange` value in that row
 - reference the y coordinate or `-log10PAdj` value in that row
- `enumerate`: iterate through the name column in `hbr_uhr_deg_chr22_top_genes` and stores the name to variable `gene_name`. `i` is incremented as it iterates through the name column within the for loop

```
plot1=seaborn.scatterplot(hbr_uhr_deg_chr22,x="log2FoldChange", y="-log10PAdj")
for i, gene_name in enumerate(hbr_uhr_deg_chr22_top_genes["name"]):
    plot1.text(hbr_uhr_deg_chr22_top_genes["log2FoldChange"][i],
              hbr_uhr_deg_chr22_top_genes["-log10PAdj"][i],gene_name)
```

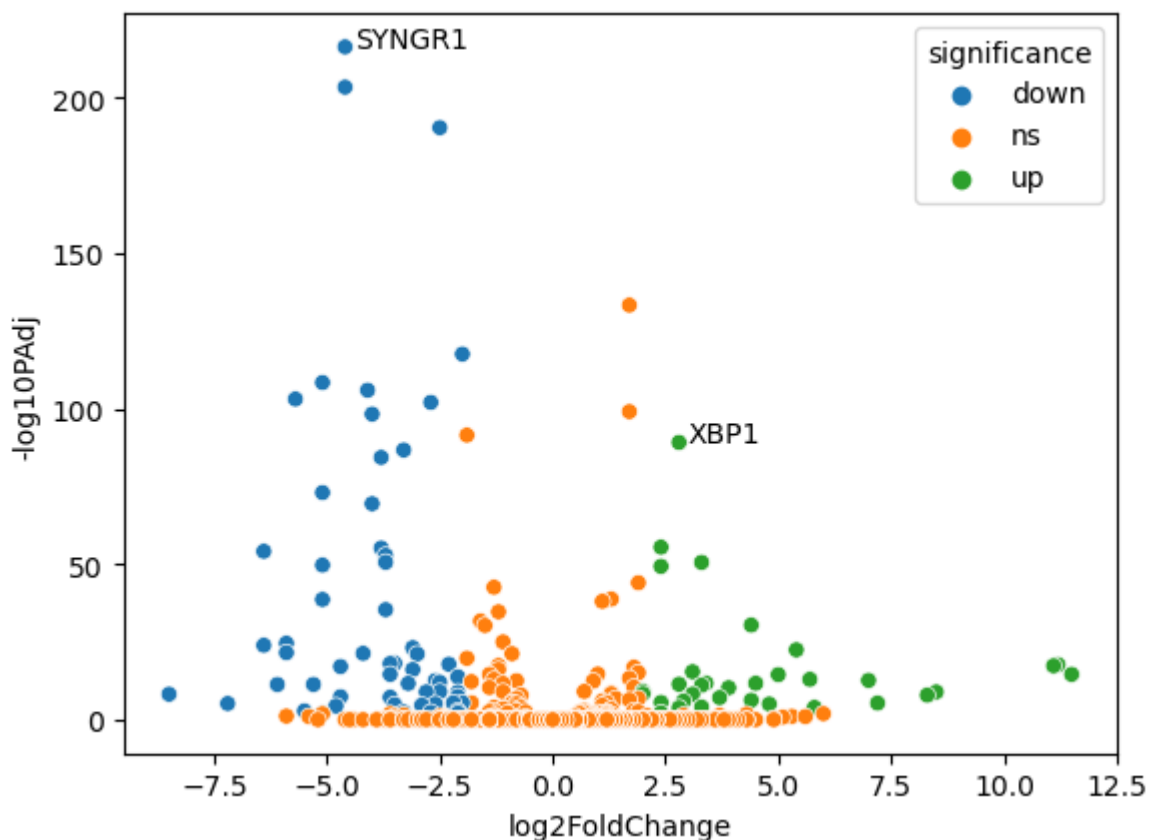


Figure 6

The next visualization is the heatmap and dendrogram combination, which helps with visualizing clusters and patterns. Heatmap and dendrogram can be used in RNA sequencing studies to inspect whether there are cluster of genes with similar expression patterns among treatment groups. The normalized counts for the top differential expressed genes in the hbr and uhr study will be used to construct a heatmap/dendrogram using Seaborn's `clustermap`.

Import the data.

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

The `seaborn.clustermap` command below generates a clustermap of the top differential expressed genes in the hbr and uhr study. The arguments and options are as follows.

- Argument: The dataset (ie. `hbr_uhr_top_deg_normalized_counts`)
- Options:
 - `z_score=0`: scale the rows by z-score
 - `cmap`: specify color palette (ie. `viridis`)
 - `figsize`: specify figure size
 - `vmin`: minimum value on the color scale bar
 - `vmax`: maximum value on the color scale bar
 - `cbar_kws`: dictionary containing key value pair that specifies the title to the color scale bar
 - `cbar_pos`: coordinates for placement of the color scale bar

```
plot4=seaborn.clustermap(hbr_uhr_top_deg_normalized_counts,z_score=0  
                          figsize=(8,8),vmin=-1.5, vmax=1.5,cbar_kws=(  
                          cbar_pos=(0.855,0.8,0.025,0.15))
```

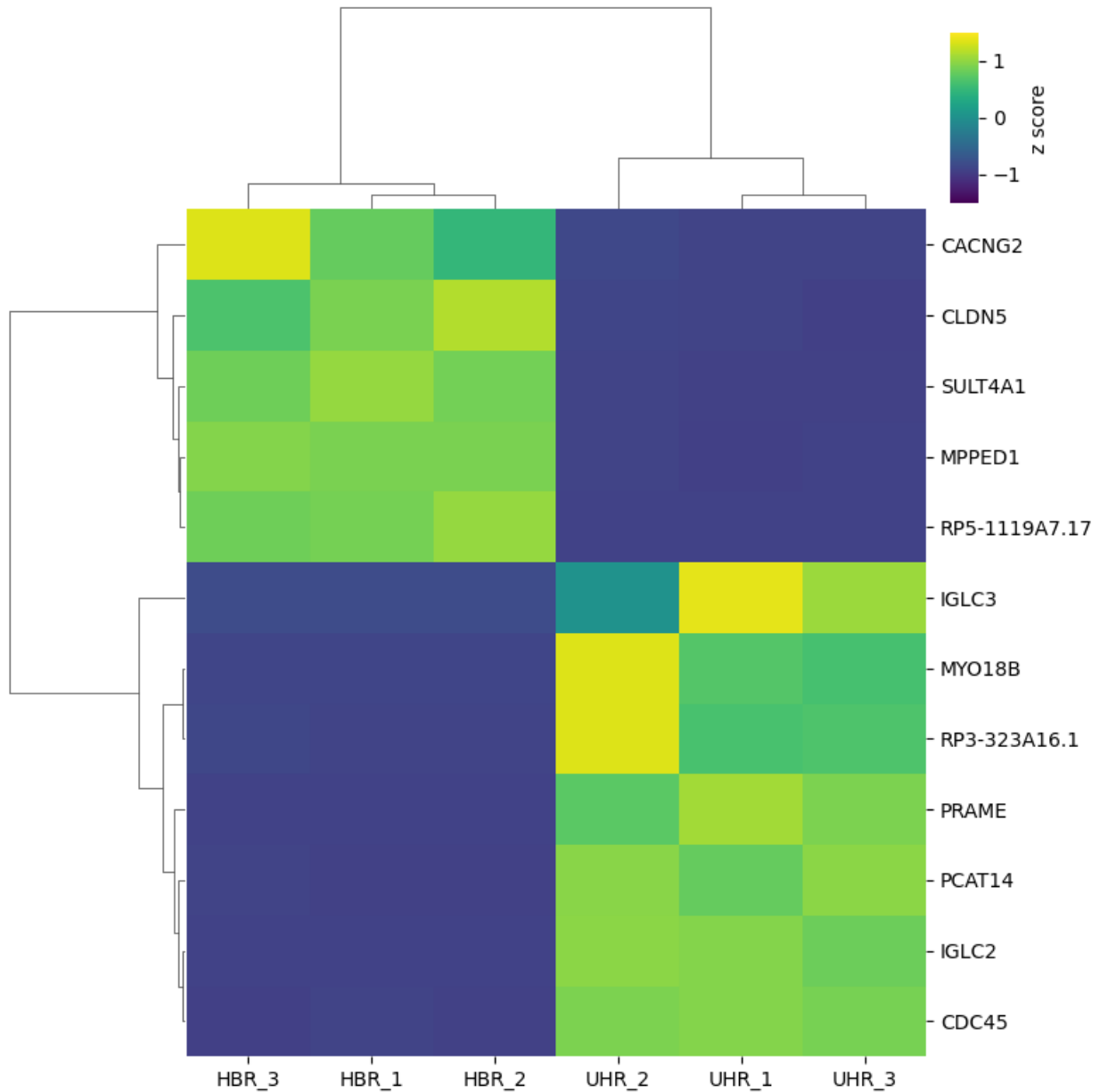


Figure 9: Expression heatmap of the top 12 differentially expressed genes in the HBR and UHR study

Below, a Pandas Series, called `samples` that contains a mapping of colors to study samples is created.

```
samples=pandas.Series({"HBR_1":"orangered", "HBR_2":"orangered", "HBR_3":"orangered", "UHR_1":"lightcoral", "UHR_2":"lightcoral", "UHR_3":"lightcoral"})
```

Then a variable, `column_colors` is created that contains a mapping of the `hbr_uhr_top_deg_normalized_counts` column headings to the colors specified in `samples`. This is accomplished using the `map` command.

```
column_colors=hbr_uhr_top_deg_normalized_counts.columns.map(samples)
```

The option `col_colors`, which is set to `column_colors` is added to display a color bar on the top of the heatmap that helps to distinguish treatment groups (ie. hbr or uhr).

Other options added include

- `ax_heatmap.set_xticklabels`: allows for customizing the x axis labels' fontsize and rotation. This requires using `ax_heatmap.get_xmajorticklabels()` to get the x axis tick labels
- `ax_cbar.tick_params`: sets the size for the color scale bar labels
- `ax_col_colors.set_title`: sets the title and location bar displaying the treatment group to color mapping

```
plot4=seaborn.clustermap(hbr_uhr_top_deg_normalized_counts,z_score=0
                        figsize=(8,8),vmin=-1.5, vmax=1.5,cbar_kws={
                        col_colors=column_colors, cbar_pos=(0.855,0.8
plot4.ax_heatmap.set_xticklabels(plot4.ax_heatmap.get_xmajorticklabel
plot4.ax_cbar.tick_params(labelsiz=12)
plot4.ax_col_colors.set_title("treatment",x=-0.1,y=0.01)
plt.show()
```

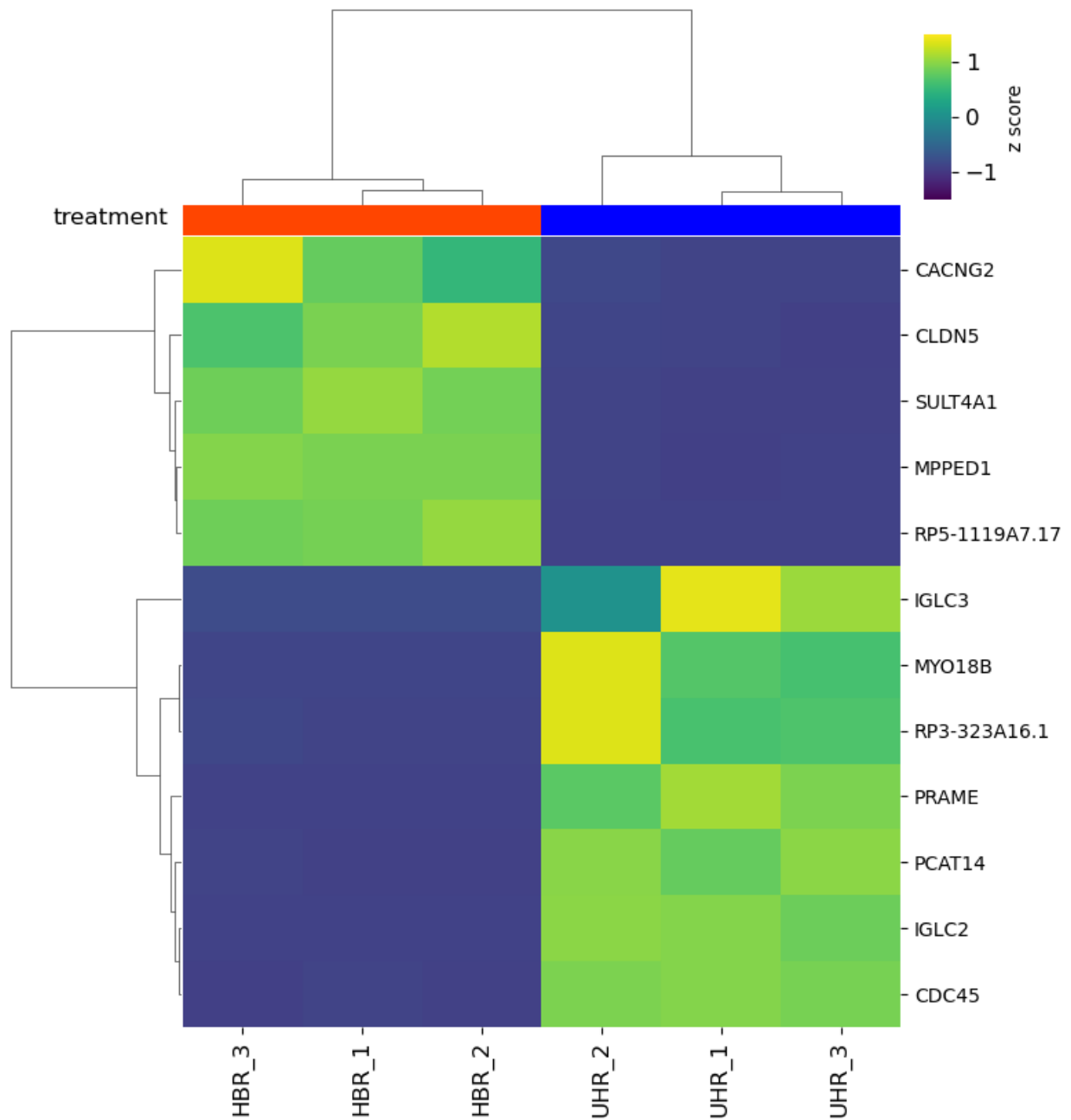



Figure 10: Expression heatmap of the top 12 differentially expressed genes in the HBR and UHR study with treatment group annotations.

Practice questions



Lesson 2 practice questions

Question 1

Create a variable that stores the value for pi.

{{Sdet}}

Solution{{Esum}}

```
pi=3.14
```

{{Edet}}

Question 2

What data type is stored in the variable pi? And why?

{{Sdet}}

Solution{{Esum}}

```
type(pi)
```

```
float
```

The variable pi has decimals, thus it is a float.

{{Edet}}

Question 3

Create a variable that stores Avogadro's number.

{{Sdet}}

Solution{{Esum}}

```
avogadro=6.02e23
```



```
{{Edet}}
```

Question 4

How do we check if Avogadro's number is greater than pi?

```
{{Sdet}}
```

Solution{{Esum}}

```
avogadro > pi
```

```
True
```

```
{{Edet}}
```

Question 5

Use the `if` statement to print out something if Avogadro's number is greater than pi.

```
{{Sdet}}
```

Solution{{Esum}}

```
if avogadro > pi:  
    print("Avogadro's number is greater than pi")  
else:  
    print("No conclusion can be made")
```

```
Avogadro's number is greater than pi
```

```
{{Edet}}
```

Question 6

Create a list of five random things that you can think of and then use a `for` loop to print each item in the list.

```
{{Sdet}}
```



```
Solution{{Esum}}
```

```
town=["Curry","Thompson","Green","Igoudala","Durant"]
```

```
for player in range(0,5):  
    print(town[player])
```

Alternative solution

```
for player in town:  
    print(player)
```

```
{{Edet}}
```



Lesson 3 practice questions

Question 1

Import `hcc1395_chr22_rna_seq_counts.csv` and store it as `hcc1395_chr22_counts`.

{{Sdet}}

Solution{{Esum}}

```
import pandas
```

```
hcc1395_chr22_counts=pandas.read_csv("./hcc1395_chr22_rna_seq_counts
```

{{Edet}}

Question 2

How many rows and columns are in `hcc1395_chr22_counts`?

{{Sdet}}

Solution{{Esum}}

```
hcc1395_chr22_counts.shape
```

```
(1335, 7)
```

{{Edet}}

Question 3

What are the column names in `hcc1395_chr22_counts` and how to view the first 10 rows of this data set?

{{Sdet}}

Solution{{Esum}}

```
hcc1395_chr22_counts.head(10)
```



Alternatively, use `hcc1395_chr22_counts.columns` to get the column headings for this data frame.

```
{{Edet}}
```

Question 4

How many genes start with the letter "C" in `hcc1395_chr22_counts`?

```
{{Sdet}}
```

Solution{{Esum}}

```
hcc1395_chr22_counts.loc[hcc1395_chr22_counts.loc[:, 'Geneid'].str.starts
```

```
{{Edet}}
```

Question 5

Import `hcc1395_deg_chr22.csv` and store it as `hcc1395_deg_chr22`.

```
{{Sdet}}
```

Solution{{Esum}}

```
hcc1395_deg_chr22=pandas.read_csv("./hcc1395_deg_chr22.csv")
```

```
{{Edet}}
```

Question 6

Remove ".bam" from the column headers of `hcc1395_deg_chr22`.

```
{{Sdet}}
```

Solution{{Esum}}

```
hcc1395_deg_chr22.columns=hcc1395_deg_chr22.columns.str.replace(".bam", "")
```

```
{{Edet}}
```

Question 7



Subset out the following columns from `hcc1395_deg_chr22` and store it as `hcc1395_deg_chr22_1`.

- `name`
- `log2FoldChange`
- `PAdj`

{{Sdet}}

Solution{{Esum}}

```
hcc1395_deg_chr22_1=hcc1395_deg_chr22.loc[:,["name", "log2FoldChange'
```

Use the `.head` function to check of the subsetting was done correctly.

```
hcc1395_deg_chr22_1.head()
```

{{Edet}}

Question 8

Add a column to `hcc1395_deg_chr22_1` that contains the negative log10 of the `PAdj` value.

{{Sdet}}

Solution{{Esum}}

```
import numpy
```

```
hcc1395_deg_chr22_1["-log10PAdj"]=numpy.negative(numpy.log10(hcc1395_
```

{{Edet}}



Lesson 4 practice questions

Question 1

Create a volcano plot for the differential expression analysis results for the hcc1395 data (hint: import hcc1395_deg_chr22_with_significance.csv)

{{Sdet}}

Solution{{Esum}}

```
import pandas
import matplotlib.pyplot as plt
import seaborn
```

```
hcc1395_deg_chr22=pandas.read_csv("./hcc1395_deg_chr22_with_significa
```

```
plot1=seaborn.scatterplot(hcc1395_deg_chr22,x="log2FoldChange", y="-"
plt.show()
```

{{Edet}}

Question 2

Label the two most differential expressed genes in the volcano plot. As a hint, first import hcc1395_deg_chr22_top_genes.csv.

{{Sdet}}

Solution{{Esum}}

```
hcc1395_deg_chr22_top_genes=pandas.read_csv("./hcc1395_deg_chr22_top_
```

```
plot1=seaborn.scatterplot(hcc1395_deg_chr22,x="log2FoldChange", y="
for i, gene_name in enumerate(hcc1395_deg_chr22_top_genes["name"]):
    plot1.text(hcc1395_deg_chr22_top_genes["log2FoldChange"][i],
              hcc1395_deg_chr22_top_genes["-log10PAdj"][i],gene_name)
plt.show()
```

{{Edet}}

Question 3

Import hcc1395_top_deg_normalized_counts.csv and create an expression heatmap. Use the Viridis color palette.

{{Sdet}}

Solution{{Esum}}

```
hcc1395_top_deg_normalized_counts=pandas.read_csv("./hcc1395_top_deg_
```

```
plot2=seaborn.clustermap(hcc1395_top_deg_normalized_counts,z_score=0
                          figsize=(8,8),vmin=-1.5, vmax=1.5,cbar_kws=(
plt.show()
```

{{Edet}}

Question 4

Add a bar on the top of the heatmap that shows which treatment group the samples belong to.

{{Sdet}}

Solution{{Esum}}

```
samples=pandas.Series({"hcc1395_normal_rep1":"orangered", "hcc1395_n
column_colors = hcc1395_top_deg_normalized_counts.columns.map(sample
plot2=seaborn.clustermap(hcc1395_top_deg_normalized_counts,z_score=0
                          figsize=(8,8),vmin=-1.5, vmax=1.5,cbar_kws=(
                          col_colors=column_colors, cbar_pos=(0.05,0.8
plot2.ax_heatmap.set_xticklabels(plot2.ax_heatmap.get_xmajorticklabel
plot2.ax_cbar.tick_params(labelsize=12)
plot2.ax_col_colors.set_title("treatment",x=1.09,y=-0.3)
plt.show()
```

{{Edet}}

Finding help

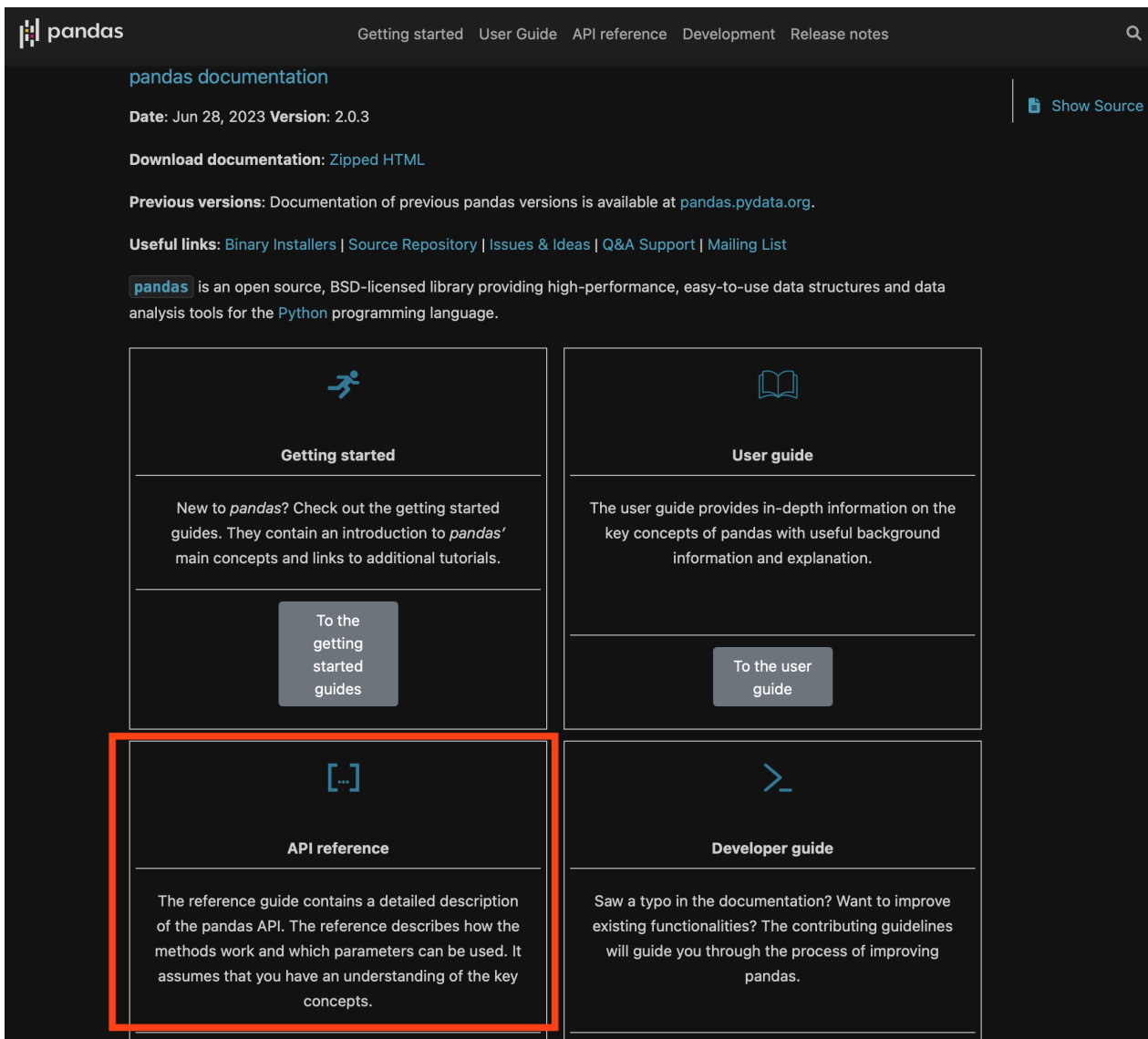
The document provides useful links where participants can find help for the Python packages that were addressed during the course series.

Pandas - package for working with tabular data (<https://pandas.pydata.org>)

- Pandas API reference gives instructions for each command (<https://pandas.pydata.org/docs/reference/index.html>). To get to the API reference, either
 - Navigate to the the Documentation section at the Pandas homepage and click on API reference (Figure 1).
 - OR, click on the the Documentation tab at the top of the Pandas homepage and click on the tile labeled API reference in the subsequent page (Figure 2).

The screenshot shows the Pandas homepage with a dark blue header. The 'Documentation' link in the header is highlighted with a red box. Below the header, the main content area features the 'pandas' logo and a description: 'pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.' A blue button labeled 'Install pandas now!' is visible. To the right, the 'Latest version: 2.0.3' section lists links for 'What's new in 2.0.3', 'Release date: Jun 28, 2023', 'Documentation (web)', and 'Download source code'. Below this, 'Follow us' includes icons for GitHub, Medium, and Twitter. The 'Get the book' section shows the cover of 'Python for Data Analysis' by Wes McKinney. At the bottom, three columns of navigation links are shown: 'Getting started' (Install pandas, Getting started), 'Documentation' (User guide, API reference, Contributing to pandas, Release notes), and 'Community' (About pandas, Ask a question, Ecosystem). A red arrow points to the 'API reference' link in the Documentation column. At the bottom left, it says 'With the support of:'.

Figure 1



The screenshot shows the pandas documentation website. At the top, there is a navigation bar with links for "Getting started", "User Guide", "API reference", "Development", and "Release notes". Below the navigation bar, the page title is "pandas documentation". The date is "Jun 28, 2023" and the version is "2.0.3". There is a "Show Source" link. Below this, there are links for "Download documentation: Zipped HTML", "Previous versions: Documentation of previous pandas versions is available at pandas.pydata.org.", and "Useful links: Binary Installers | Source Repository | Issues & Ideas | Q&A Support | Mailing List". A brief description of pandas is provided: "pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language." Below the description, there are four main navigation cards: "Getting started" (with a person icon), "User guide" (with a book icon), "API reference" (with a code icon), and "Developer guide" (with a code icon). Each card contains a short description and a button to navigate to that section. The "API reference" card is highlighted with a red border.

Figure 2

Seaborn for data visualization (<https://seaborn.pydata.org/index.html>)

- Seaborn API reference gives instructions for each command (<https://seaborn.pydata.org/api.html>). To get to the Seaborn API reference, click on API at the top of the Seaborn website.

seaborn: statistical data visualization

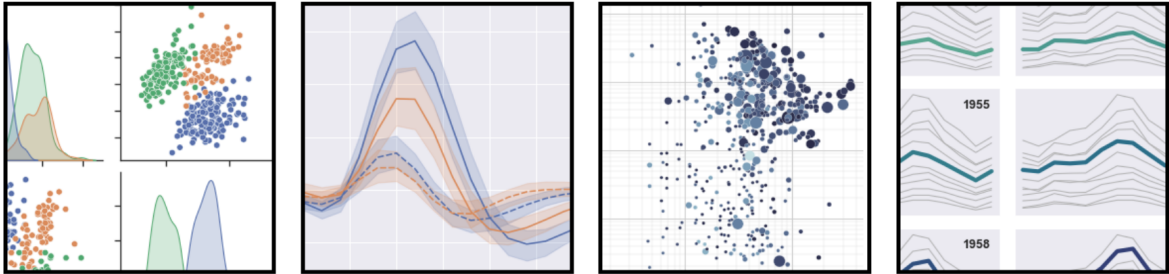


Figure 3

Numpy for scientific computing (<https://numpy.org/doc/stable/index.html>)

- [Numpy API reference \(https://numpy.org/doc/stable/reference/index.html\)](https://numpy.org/doc/stable/reference/index.html). To get to this, select Documentation at the top of the Numpy homepage (Figure 4) and then click on either of the links to the API reference (Figure 5).

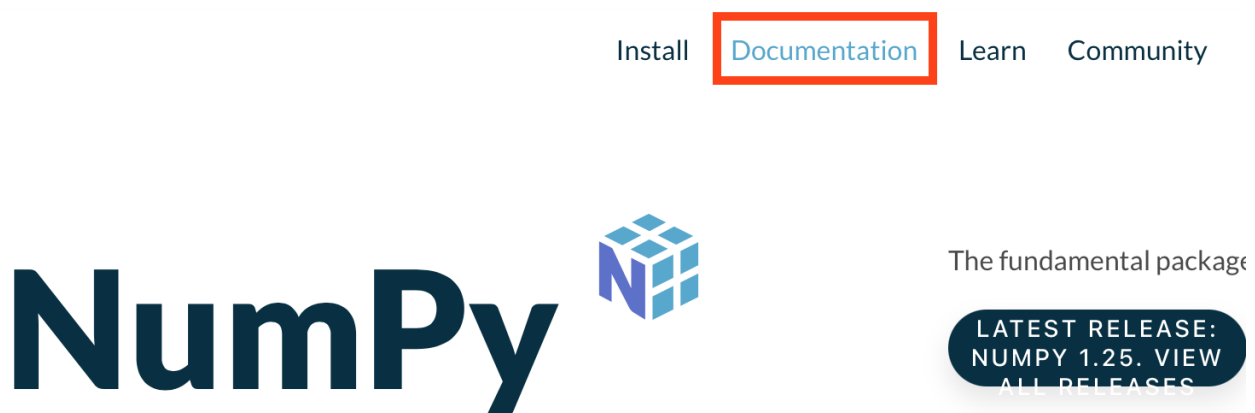


Figure 4

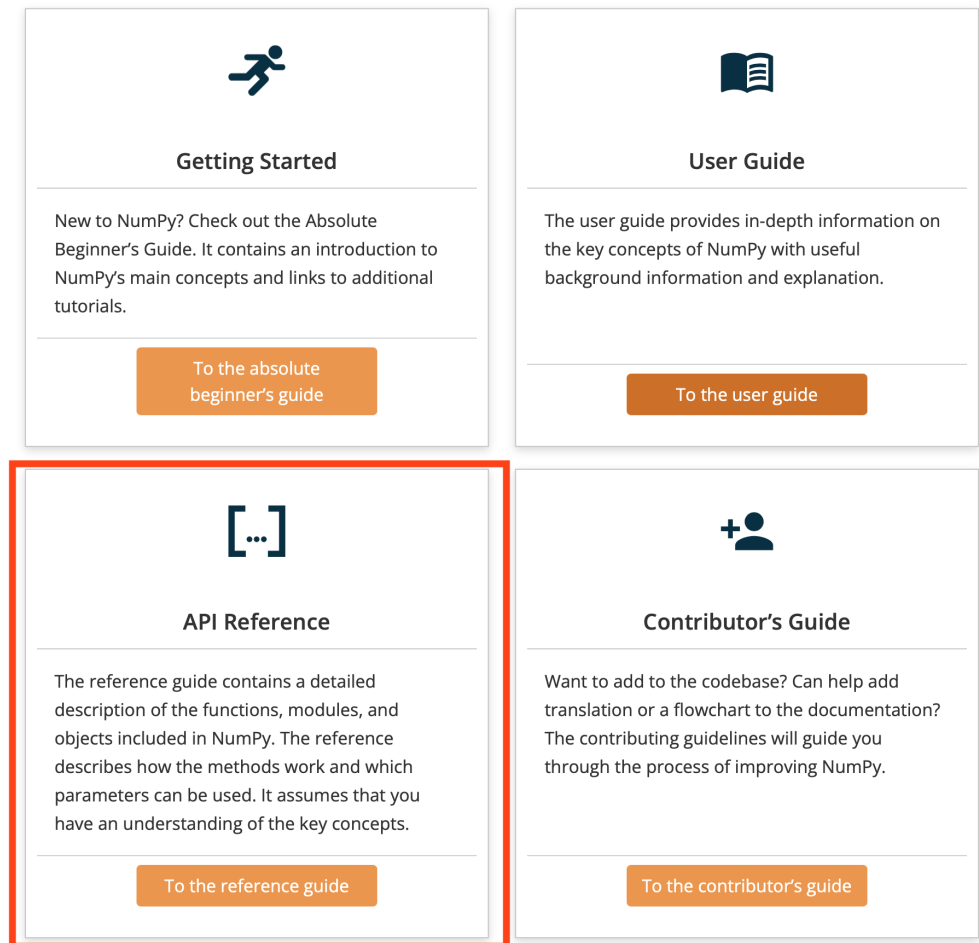


Figure 5

Matplotlib for data visualization (<https://matplotlib.org>)

- [Matplotlib API reference \(https://matplotlib.org/stable/api/index\)](https://matplotlib.org/stable/api/index). To get to this, click on reference at the top of the Matplotlib homepage (Figure 6).

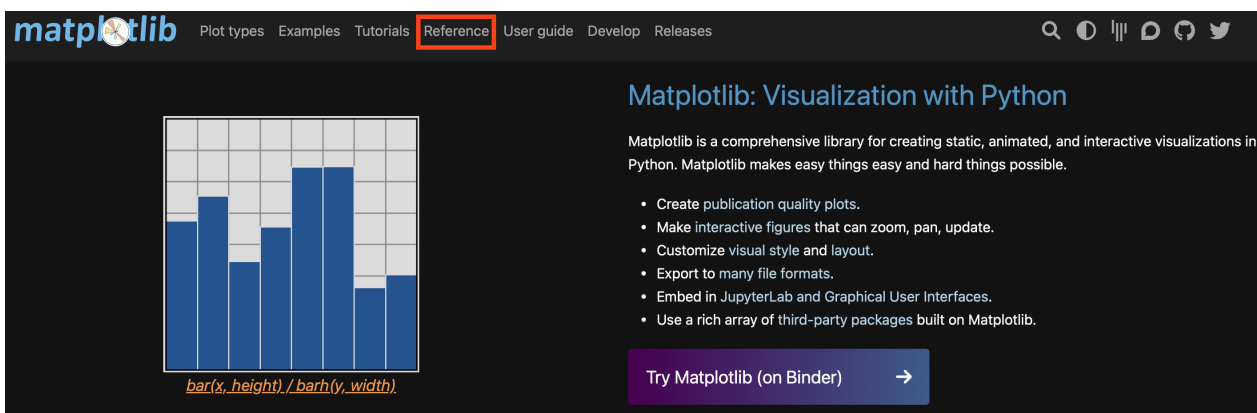


Figure 6