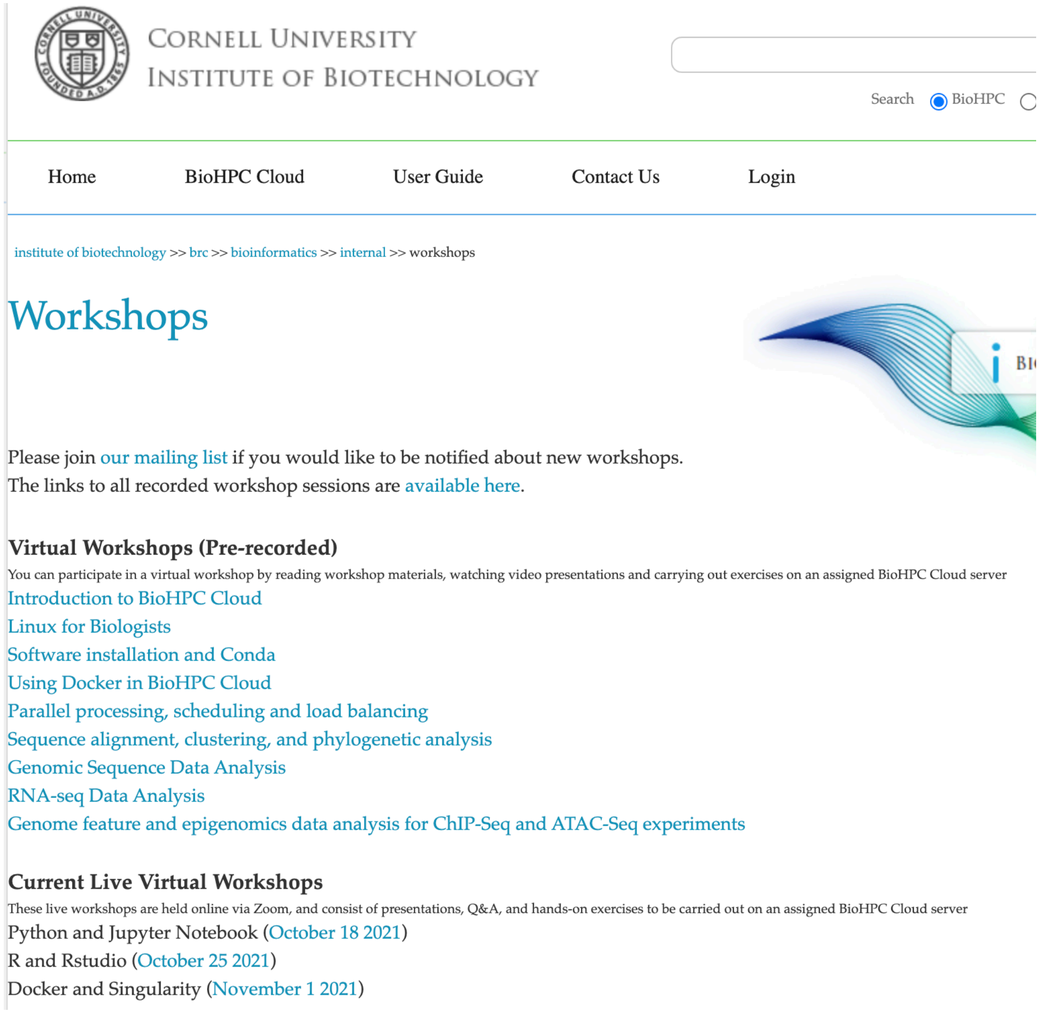
Sept 28, 2022

Programming basics
with Bash, Python, and R

Suzy Strickler
srs57@cornell.edu

- https://www.rstudio.com/products/rstudio/download/#download
- sudo dpkg -i rstudio-2021.09.0-351-amd64.deb

Office hour 12:30 - 1:30 pm today in G22 PS
See Canvas for link and schedule



Homework Solutions

Please log on to the william server using the credentials that will be supplied to you individually via email. The address for William is: william.sgn.cornell.edu

Please answer the below questions and store them in a text file called "class1_assignment.txt". You can either create the text file on your local computer and transfer it to william, or you can use one of the available text editors on william to create your answer file.

1) What is the hard disk size of William? du -lh

2) How much of the hard disk is currently free? df -lh

3) How much RAM does william have? free -m

4) Which users are currently running processes on William? ps -aux |cut -f1 -d " " |sort -u

5) How many CPUs is the process using the most memory using? top -> shift + m

6) How many people have an account on William? who

7) Which user is using the most hard disk space? du -sh *

8) What is the path to the file: canu-smartdenovo_pass5.fasta locate

9) Which version(s) of Python are installed on the system? python

10) Which version of R is on the system? R

What are scripts?

- A text document that contains instructions to be executed by a program
- Usually combine existing components
- Usually use an interpreter instead of compiler to translate to machine code
  - **Bash**, **Python**, Perl, JavaScript, PHP, Ruby

Why use scripts?

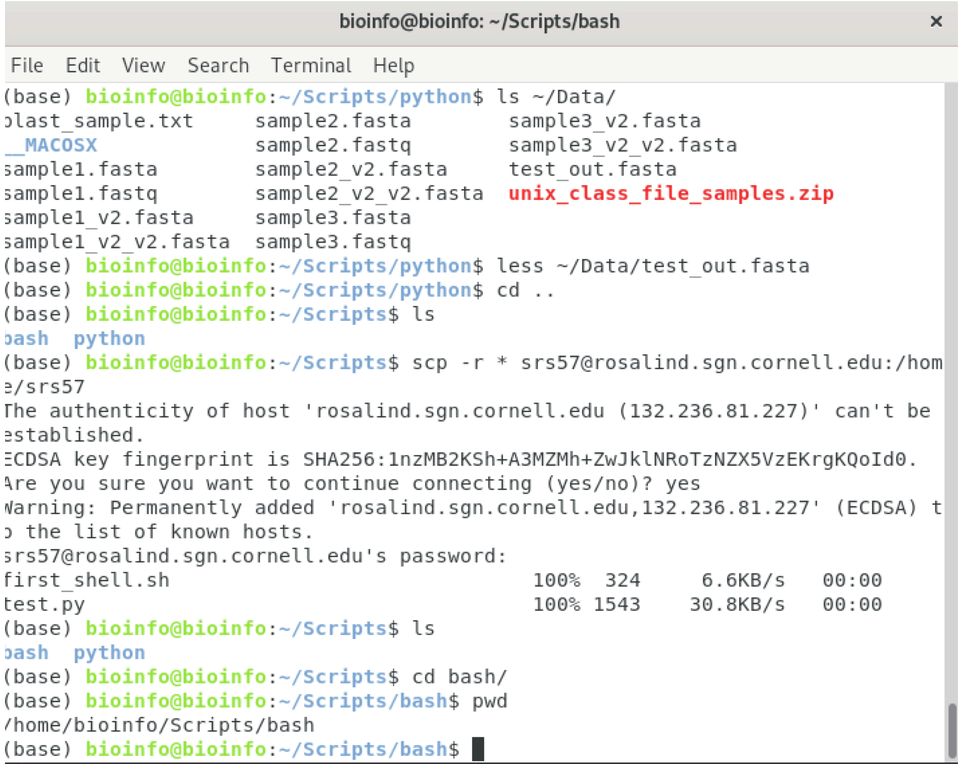
- Automation of step-by-step processes
- Reproducibility
- Share with others

Text Editors for Programming

- Don’t write code in Word!
- Vi, Vim, Emacs
- **Atom**
- Sublime
- Visual Studio

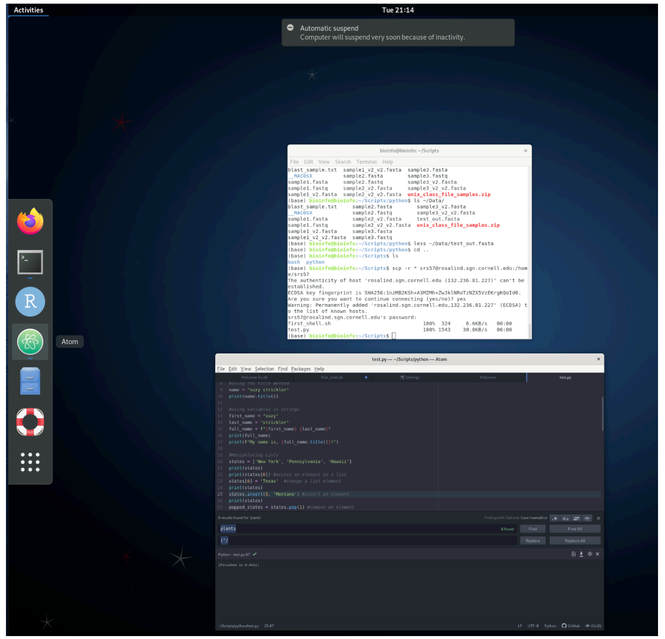
Simple Scripting with Bash

- Bash is a Unix shell, both a command language and scripting language
- Configuration file, bash.rc, is written in bash
- A bash shell script contains a list of commands
  - Since we are using the bash shell in our VM, these are the same commands we went over last time



Let’s create a bash shell script

- Open Atom on your VM
- File -> Add Project Folder
  - Select /home/bioinfo/Scripts
  - New folder: “bash”
  - Select “ok”
- File -> New file



Let’s create a bash shell script

https://github.com/bcbc-group/PLSCi7202/blob/master/first_shell.sh

We’ll call it first_shell.sh

Welcome Guide | first_shell.sh

```
#!/bin/sh
#list contents of "Desktop"
cd ~/Desktop
ls

#find all the fasta files/folders
locate *fasta

#for all the files in ~/Data, replace Atha_ with Athaliana
cd ~/Data

for file in `dir -d *.fasta`; do
  echo $file
  new_file=`echo "$file" |sed 's/.fasta/_v2.fasta/'`
  sed 's/Atha_/Athaliana/g' $file > $new_file
done

```

We can run the script from within Atom
Shift + Ctl + b

We can also run it from the command line
bash ~/Scripts/bash/first_shell.sh

Why use bash?

- It is useful for chaining other programs together.
- This allows one to automate tasks and build pipelines.

```
#!/bin/sh

#move to annotation directory
cd /opt/annotation

#get some files
cp /scratch/Botany2020NMGWorkshop/annotation/2transfer/list.txt .
cp /scratch/Botany2020NMGWorkshop/annotation/2transfer/uniprot_sprot_plants.fasta .

#run Portcullis: https://bioconda.github.io/recipes/portcullis/README.html
#docker pull maplesond/portcullis:stable #already done with Adrian
docker run --rm -v /scratch/annotation_output:/data maplesond/portcullis:stable portcullis full -t 7 -v /data/contig_15.fasta /data/SRR5046448_contig15.sort.bam

#run Mikado pipeline: https://bioconda.github.io/recipes/mikado/README.html
#docker pull cyverseuk/mikado #already done with Adrian
docker run --rm -v /scratch/annotation_output:/data cyverseuk/mikado mikado configure --list list.txt --reference data/contig_15.fasta --mode permissive --scori
docker run --rm -v /scratch/annotation_output:/data cyverseuk/mikado mikado prepare --json-conf /data/configuration.yaml

makeblastdb -in uniprot_sprot_plants.fasta -dbtype prot
blastx -max_target_seqs 5 -num_threads 7 -query mikado_prepared.fasta -outfmt 5 -db uniprot_sprot_plants.fasta -evalue 0.000001 2> blast.log | sed '/^$/d' | gzi

screen -L /opt/TransDecoder-TransDecoder-v5.5.0/TransDecoder.LongOrfs -t mikado_prepared.fasta

screen -L hmmscan --cpu 7 --domtblout pfam.domtblout Pfam-A.hmm mikado_prepared.fasta.transdecoder_dir/longest_orfs.pep

blastp -query mikado_prepared.fasta.transdecoder_dir/longest_orfs.pep -db uniprot_sprot_plants.fasta -max_target_seqs 1 -outfmt 6 -evalue 1e-5 -num_threads 7 >

/opt/TransDecoder-TransDecoder-v5.5.0/TransDecoder.Predict -t mikado_prepared.fasta --retain_blastp_hits blastp.outfmt6 --cpu 7  --retain_pfam_hits pfam.domtblo

docker run --rm -v /scratch/annotation_output:/data cyverseuk/mikado mikado serialise --json-conf /data/configuration.yaml --xml /data/mikado.blast.xml.gz --orf
```

The Python Language

- First released in 1991
- Efficient and elegant - fewer lines of code
- Object-oriented
- Runs on many operating systems
- Monty Python’s Flying Circus
- Python 2 (discontinued 2020) vs Python 3 - not completely backward compatible

Setting up Python

- Linux - usually already installed but may need updating
- OSX
- Windows
- What version is on our VM?

Code Libraries

- Modularizing code
  - Breaking apart problem into individual units
  - Easier to maintain
  - Reusable
- Modules, Packages - grouping of modules, Library - a collection of packages
- Libraries we will use:
  - BioPython, NumPy, Pandas in later classes
- Package Repos
  - Python Package Index (PyPi) and Anaconda

Comments

- Allows you to add human readable text to your code
- In Python comment lines start with: #
- Allows you and others to understand your code

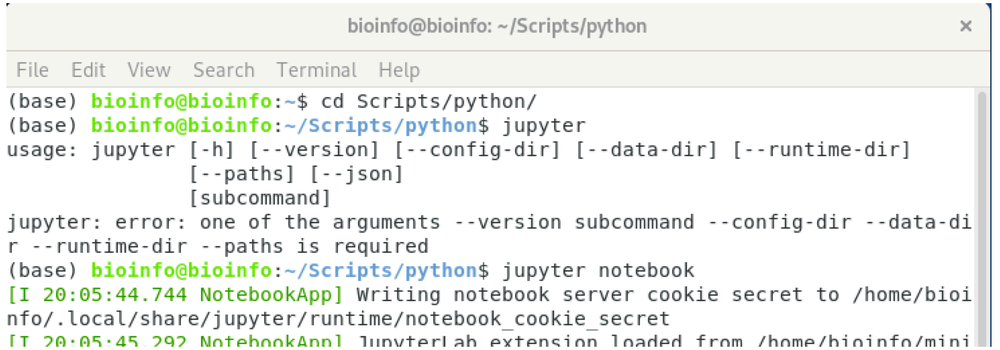
Jupyter Notebooks

- Starting the notebook server

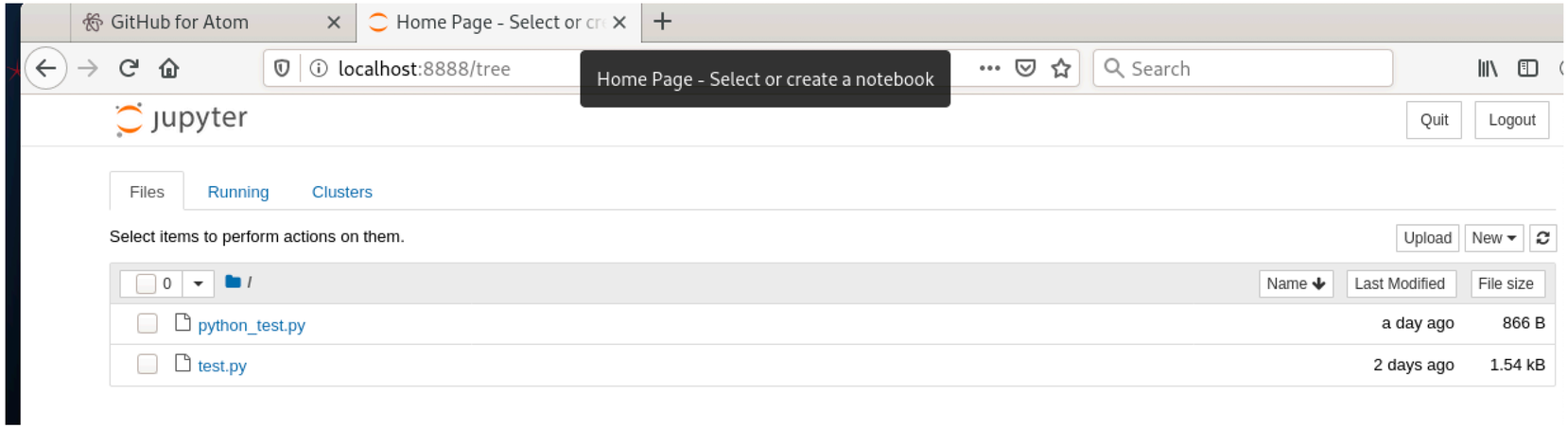
mkdir ~/Scripts/python

cd ~/Scripts/python/

jupyter notebook

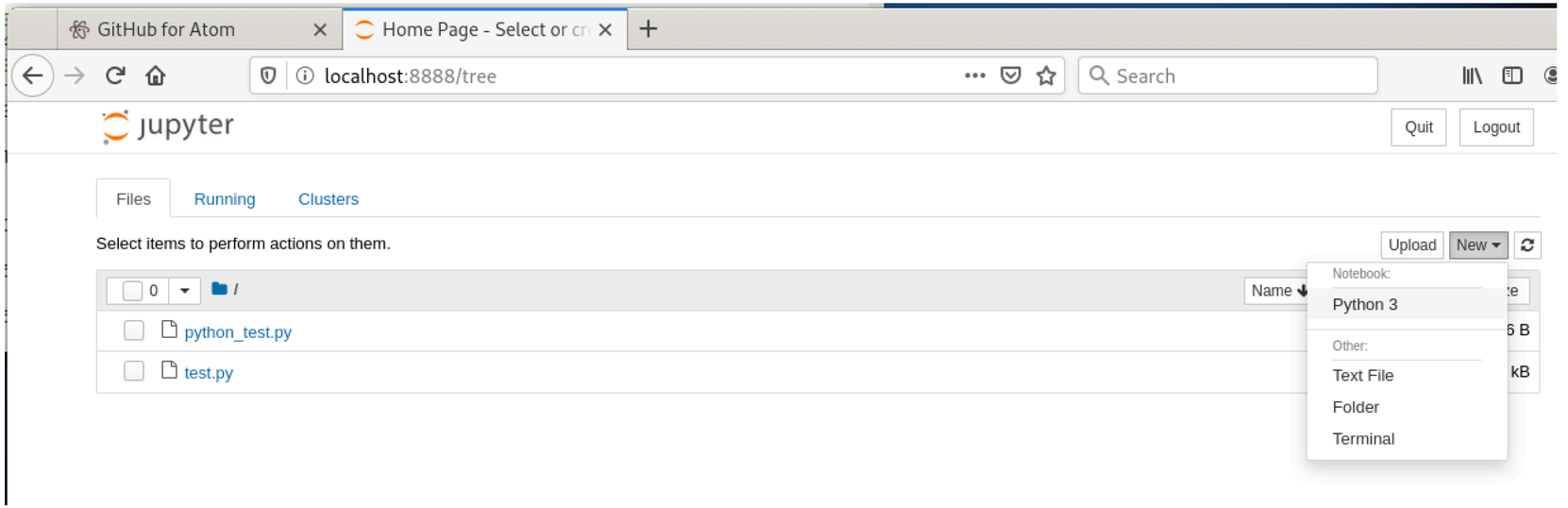


- Notebook Dashboard

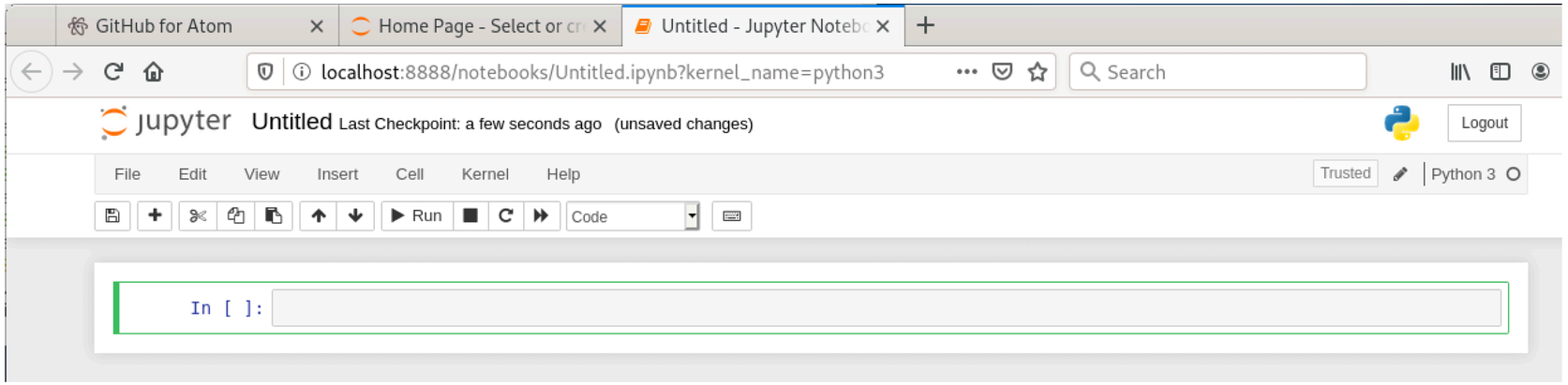


Jupyter Notebooks

- Create a new notebook

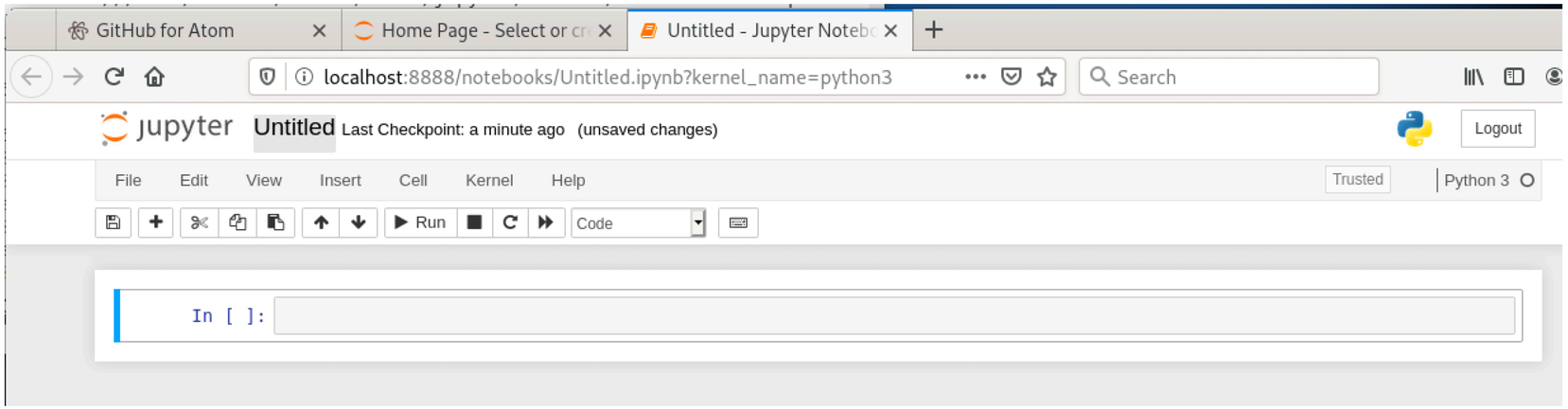


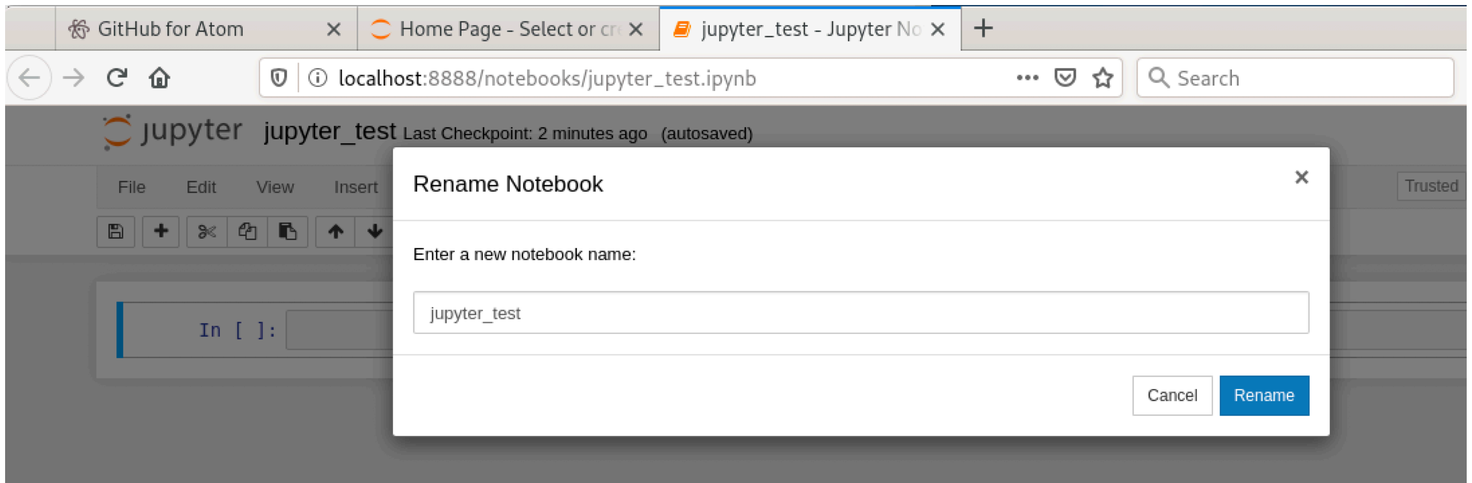
Jupyter Notebooks



Jupyter Notebooks

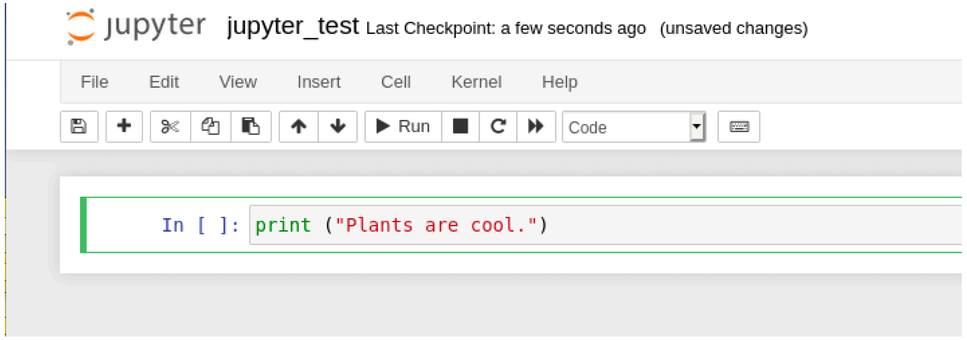
- Name the notebook





Jupyter Notebooks

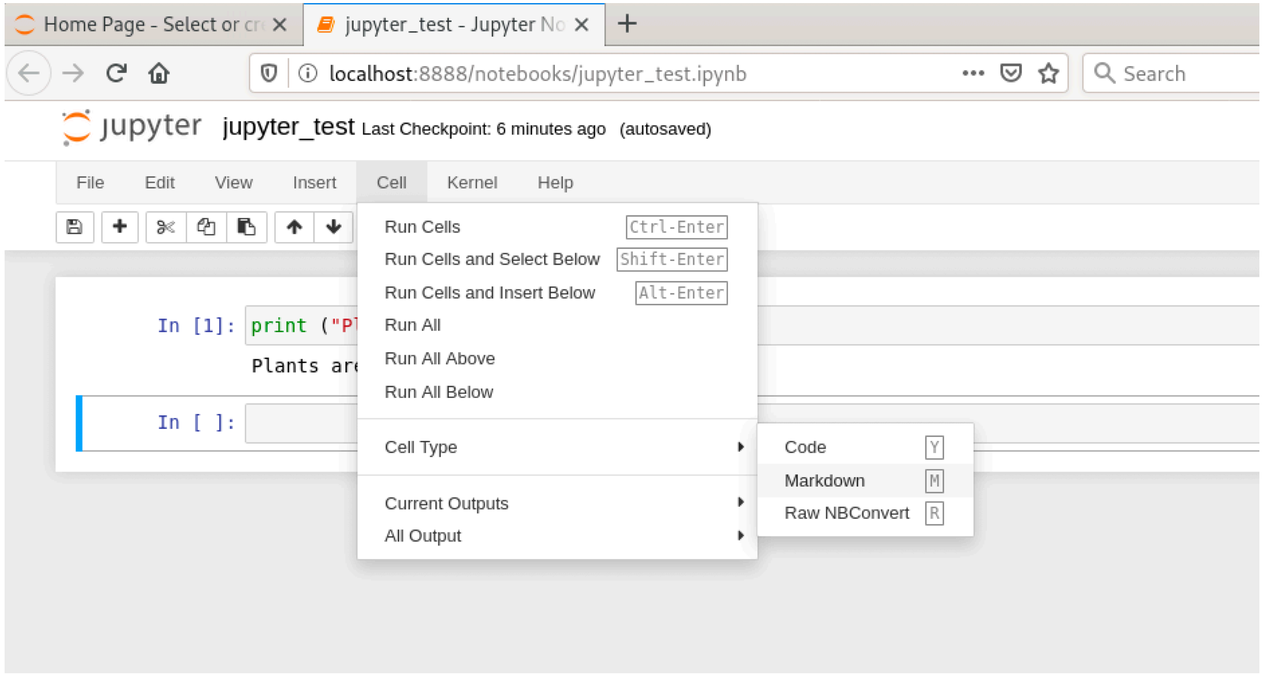
- Writing code and executing cells



Shift + enter

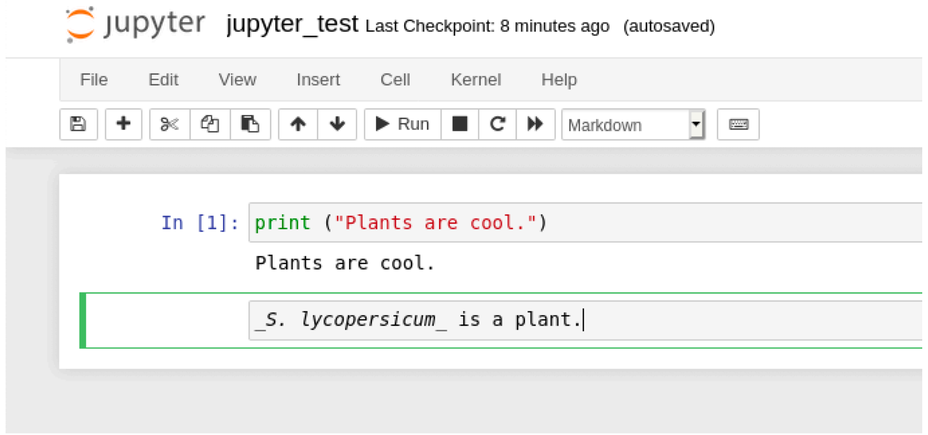
Jupyter Notebooks

- There are four cell types: Code, Markdown, Raw NBConvert, and Heading. Code and Markdown are the main types used.

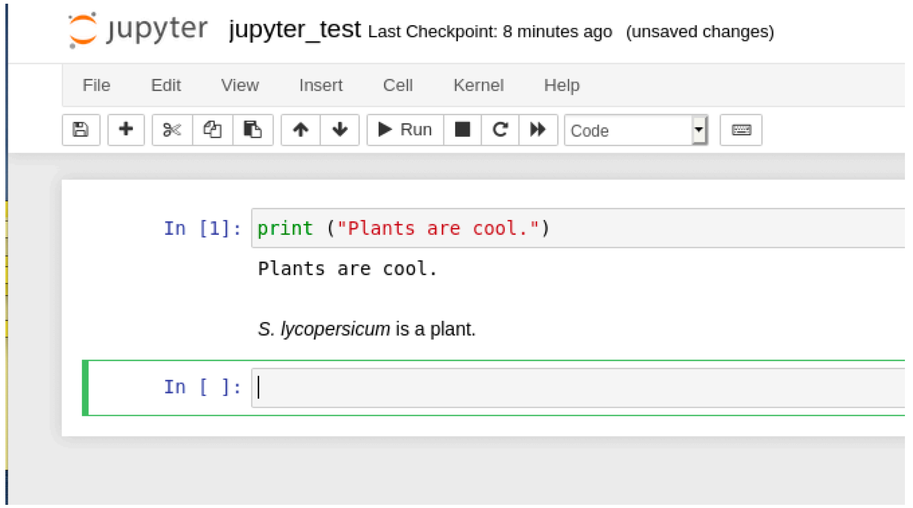


Jupyter Notebooks

- Markdown
- Esc + m

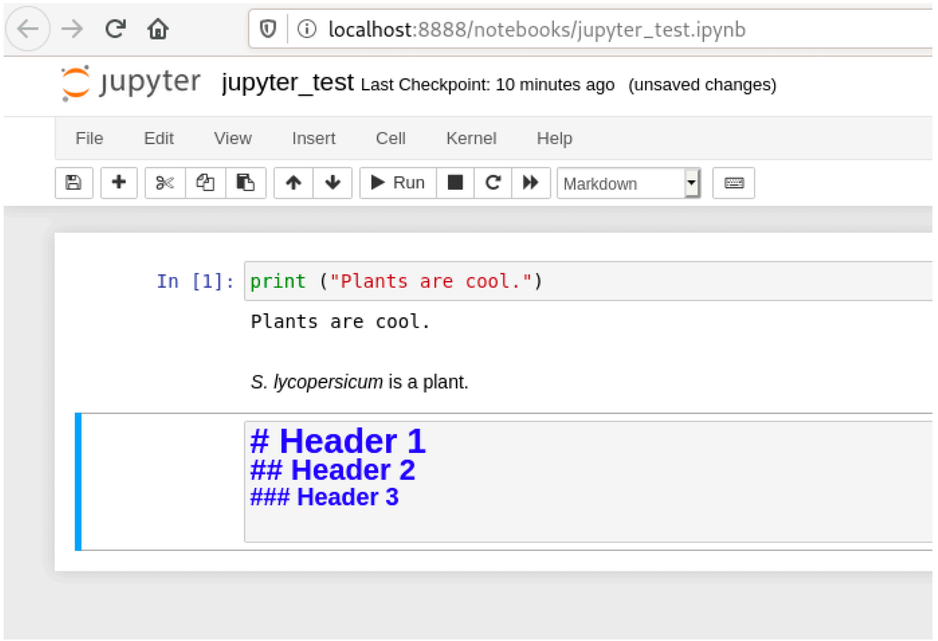


Shift + enter

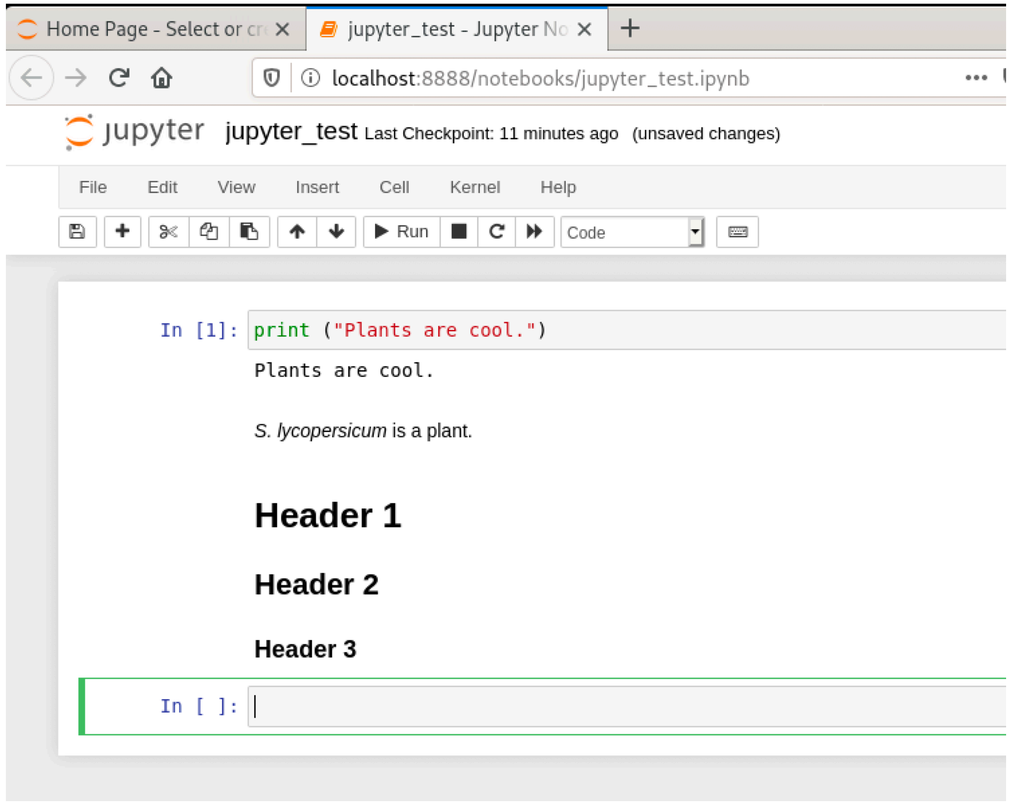


Jupyter Notebooks

- Jupyter supports Markdown which is based on html

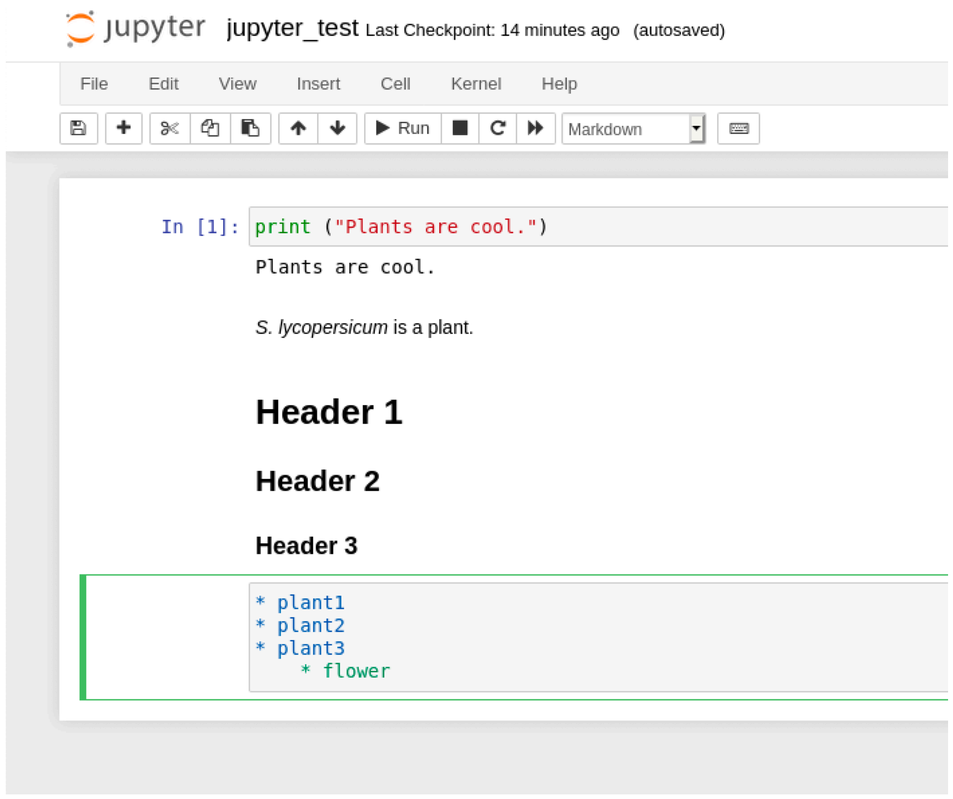


Shift + enter

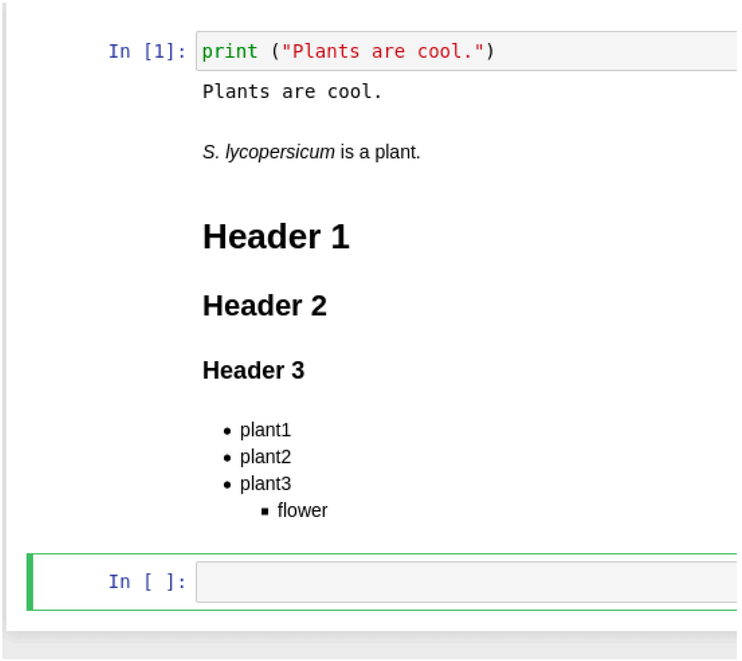


Jupyter Notebooks

- Markdown



Shift + enter



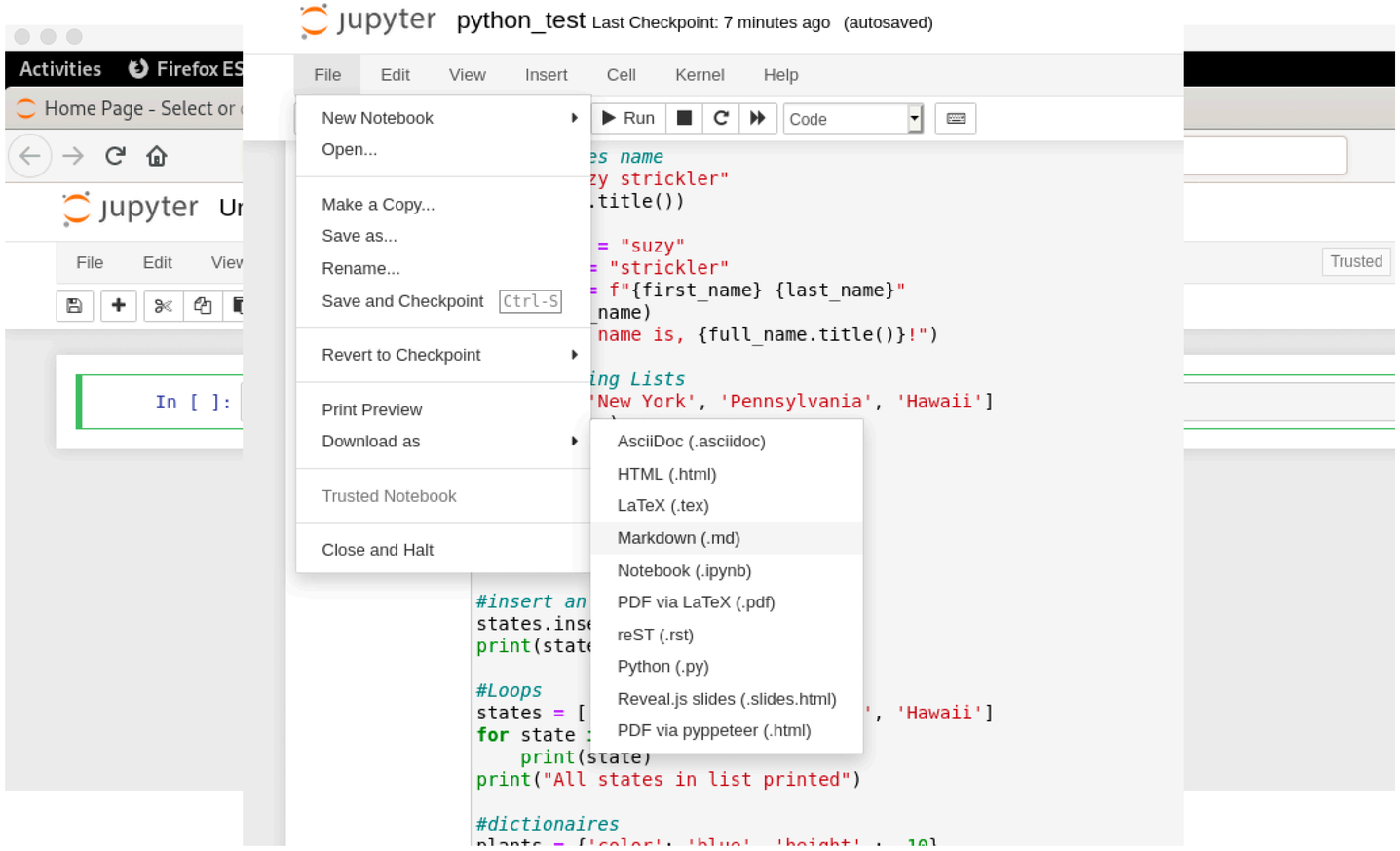
Jupyter Notebooks

- nbconvert can convert notebooks to pdf, HTML, LaTeX, and other formats

```Shell
$ jupyter nbconvert py_examples.ipynb --to pdf
```

- Can also use menu to download as various file types

Jupyter Notebooks



Jupyter Notebooks

- Extensions add functionality to a notebook

Shell

```
$ jupyter nbextension install EXTENSION_NAME
```

This only installs the extension but does not make it active. You will need to enable an extension after installing it by running the following:

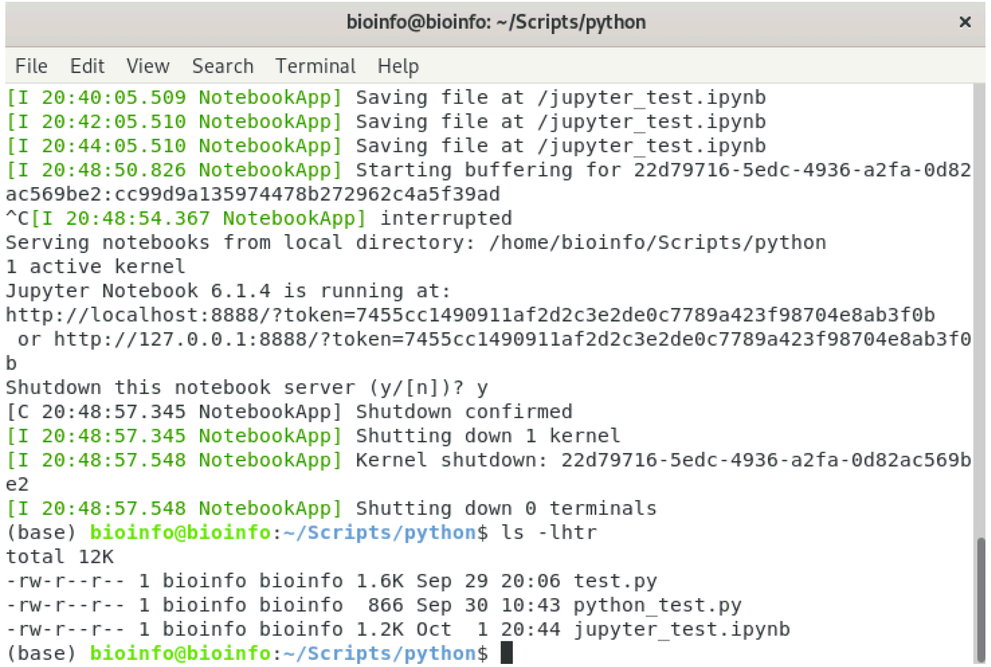
Shell

```
$ jupyter nbextension enable EXTENSION_NAME
```

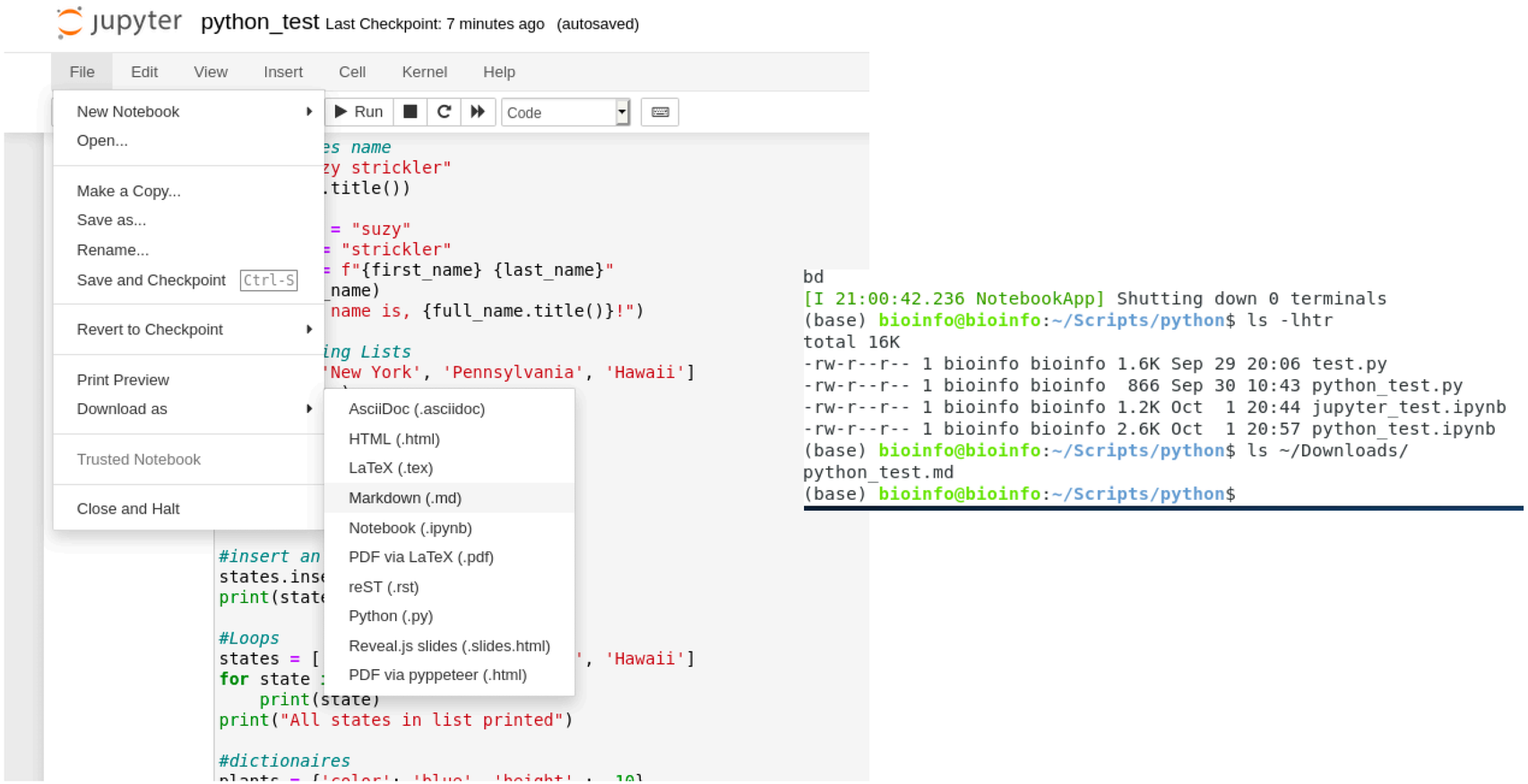
- Examples: Table of Contents, Variable Inspector, ExecuteTime
  - https://towardsdatascience.com/jupyter-notebook-extensions-517fa69d2231

Jupyter Notebooks

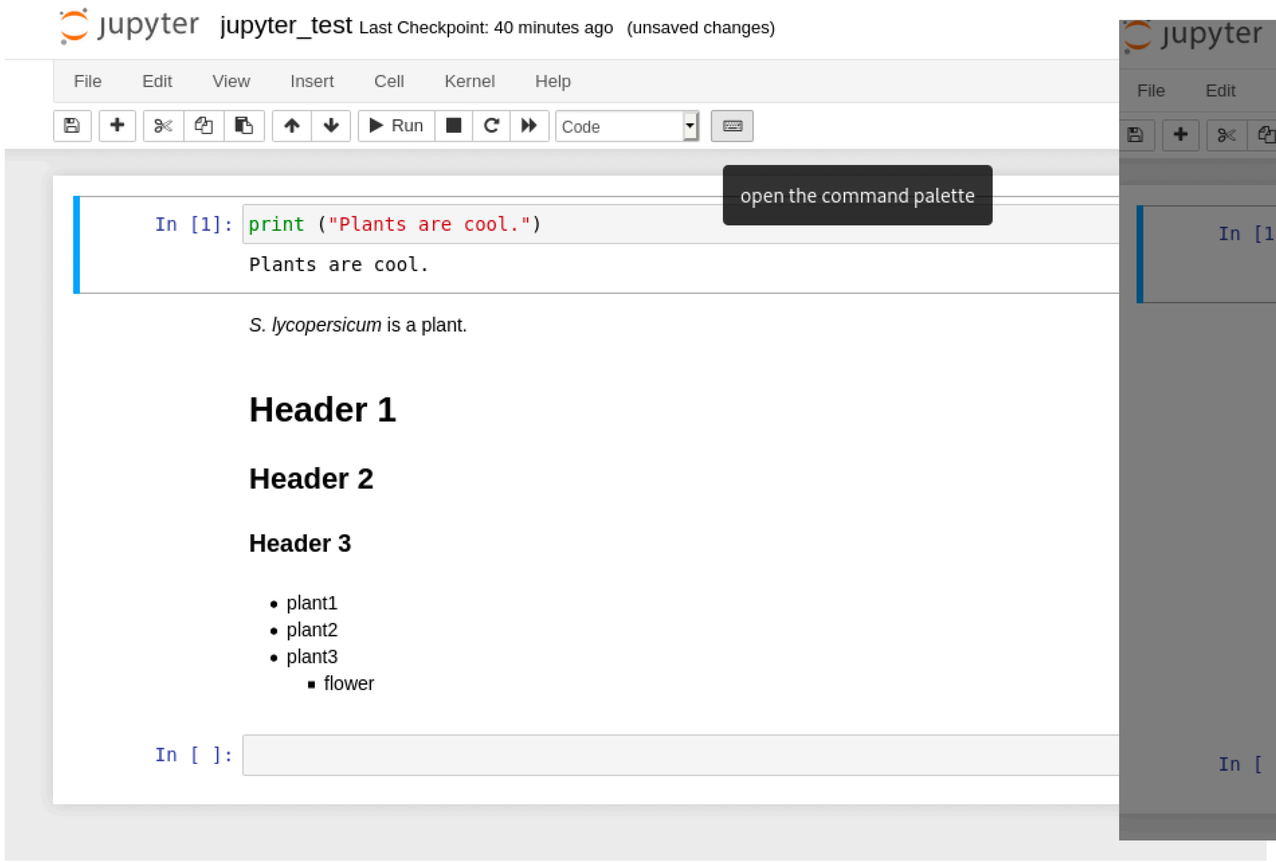
- Notebook is stored with extension .ipynb (Json format)

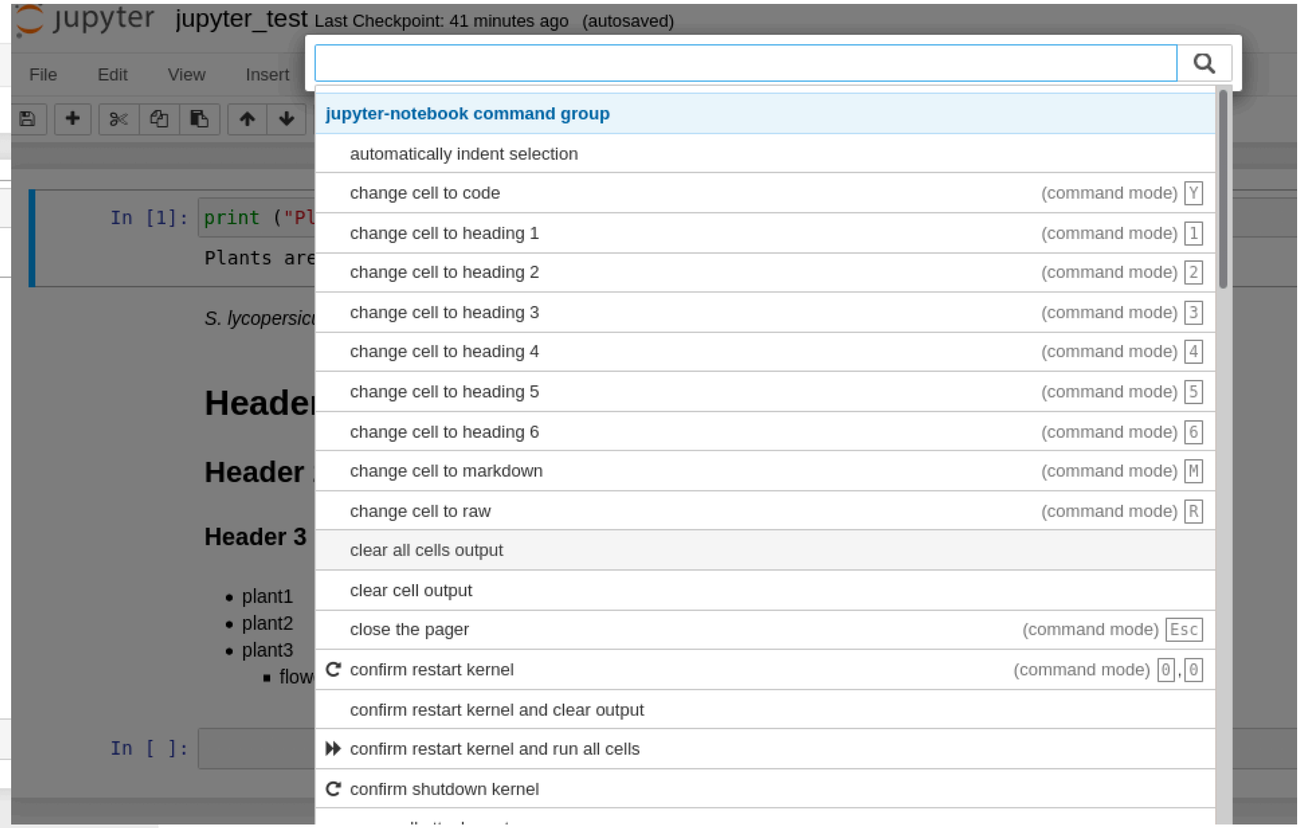


Jupyter Notebooks



Jupyter Notebooks





Jupyter Resources

- https://jupyter.readthedocs.io/en/latest/use/use-cases/data_science.html

Jupyter for Data Science

The purpose of this page is to highlight kernels and other projects that are central to the usage of Jupyter in data science. This page is not meant to be comprehensive or unbiased, but rather to provide an opinionated view of the usage of Jupyter in data science based on our interactions with users.

The following Jupyter kernels are widely used in data science:

- **python**
  - IPython (GitHub Repo)
- **R**
  - IRkernel (Documentation, GitHub Repo)
  - IRdisplay (GitHub Repo)
  - repr (GitHub Repo)
- **Julia**
  - IJulia Kernel (GitHub Repo)
  - Interactive Widgets (GitHub Repo)
- Bash (GitHub Repo)

Jupyter Notebooks

- Opening an existing notebook

  jupyter notebook notebook.ipynb

  https://github.com/bcbc-group/PLSCi7202/blob/master/first_python.py

Now let's look at some example Python code

- Can run programs in Atom (ctl+shift + b)
- Can run Python in a terminal session

```
>>> quit()
(base) bioinfo@bioinfo:~/Scripts/python$ python
Python 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Can run as a Python script from the command line

```
base) bioinfo@bioinfo:~/Scripts/python$ python test.py
```

- **print("Python is fun!")**

```
1  #testing out python
2  print("Python is fun!")
```

Variables

- Labels that are assigned to values
- Names contain only letters, numbers, and underscores. They cannot start with a number and cannot contain spaces
- Do not use function names, for example print
- Keep names short but descriptive
- Example:

message = “Python is fun!”

print(message)

```
4  #using variables
5  message = "Python is fun!"
6  print(message)
```

Strings

- A series of characters
- Are surrounded by single or double quotes
- Example:

  “This is a string”

  ‘This is also a string’

Methods

- An action Python can perform on a piece of data
- Example:

name = “suzy strickler”

print(name.title())

```
 8  #using the title method
 9  name = "suzy strickler"
10  print(name.title())
```

Variables in Strings

- Example:

first_name = “suzy”

last_name = “strickler”

full_name = f”{first_name} {last_name}”

print(full_name)

print(f”My name is, {full_name.title()}!”)

```
12  #using variables in strings
13  first_name = "suzy"
14  last_name = "strickler"
15  full_name = f"{first_name} {last_name}"
16  print(full_name)
17  print(f"My name is, {full_name.title()}!")
18
```

Lists

- A collection of items in a particular order
- Example:

  states = [‘New York’, ‘Pennsylvania’, ‘Hawaii’]

  print(states)

- Accessing elements of a list:

  print(states[0])

- Changing an element

  states[0] = ‘Texas’

  print(states)

```
19  #Manipulating Lists
20  states = ['New York', 'Pennsylvania', 'Hawaii']
21  print(states)
22  print(states[0]) #access an element in a list
23  states[0] = 'Texas'  #change a list element
24  print(states)
25  states.insert(0, 'Montana') #insert an element
26  print(states)
27  popped_states = states.pop(1) #remove an element
28  print(popped_states)
29  print(states)
```

Lists cont’d

- Adding elements

  states.insert(0, ‘Montana’)

  print(states)

- Removing elements

  popped_states = states.pop(1)

  print(popped_states)

```
19  #Manipulating Lists
20  states = ['New York', 'Pennsylvania', 'Hawaii']
21  print(states)
22  print(states[0]) #access an element in a list
23  states[0] = 'Texas'  #change a list element
24  print(states)
25  states.insert(0, 'Montana') #insert an element
26  print(states)
27  popped_states = states.pop(1) #remove an element
28  print(popped_states)
29  print(states)
```

Loops

- Repeat an action with every item in a list
- Indentation
- Example

for state in states:

print(state)

print(“All states in list printed”)

```
31  #Loops
32  states = ['New York', 'Pennsylvania', 'Hawaii']
33  for state in states:
34      print(state)
35      print("All states in list printed")
36
```

If statements

```
for state in states:

    if state == ‘Hawaii’:

        print(state.upper())

    elif state == “New York”:

        print(state.lower()

    else:

        print(state.title())
```

```
37  #Loops - if statements
38  states = ['New York', 'Pennsylvania', 'Hawaii']
39  for state in states:
40      if state == 'Hawaii':
41          print(state.upper())
42      elif state == 'Montana':
43          print(state.lower())
44      else: print(state.title())
```

Dictionaries

- A collection of key-value pairs
- Example:

plants = {‘color’: ‘blue’, ‘height’: 10}

print(plants[‘color’])

#add a new key-value pair

plants[‘sepal’] = 3

print(plants)

#remove a key-value pair

del plants[‘color’]

```
46  #Dictionaries
47  plants = {'color': 'blue', 'height': 10}
48  print(plants['color'])
49  plants['sepal'] = 3  #add a new key-value pair
50  print(plants)
51  del plants['color']  #remove a key-value pair
52  print(plants)
53
```

Functions

- Blocks of code that do a specific job
- docstring = describes what the function does
- Example

def weather():

“””Find out the weather”””

answer = input(“How is the weather?”)

print(answer)

```
54  #example of a function
55  def weather():
56      """Find out the weather"""
57      answer = input("How is the weather?")
58      print(answer)
59  weather()
```

- Module - storing function in a separate file and then import it (**import weather**)

Reading from a file

- In many cases, you will want your script to operate on some input file
- Example:

```
fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")

if fasta_file.mode == "r":

    contents = fasta_file.read()

    print(contents)

fasta_file.close()
```

Writing output to a file

- Example:

```
fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")

if fasta_file.mode == "r":

    contents = fasta_file.read()

   outF = open(“/home/bioinfo/Date/test_out.fasta”, “w”)

    outF.write(contents)

    print(contents)

fasta_file.close()
```

```
61  #reading in a file
62  fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")
63  if fasta_file.mode == "r":
64      contents = fasta_file.read()
65      outF = open("/home/bioinfo/Data/test_out.fasta", "w")
66      outF.write(contents)
67  fasta_file.close()
68  outF.close()
```

General Introduction - R and RStudio

- What is R?
  - A programming, ‘scripting’ language

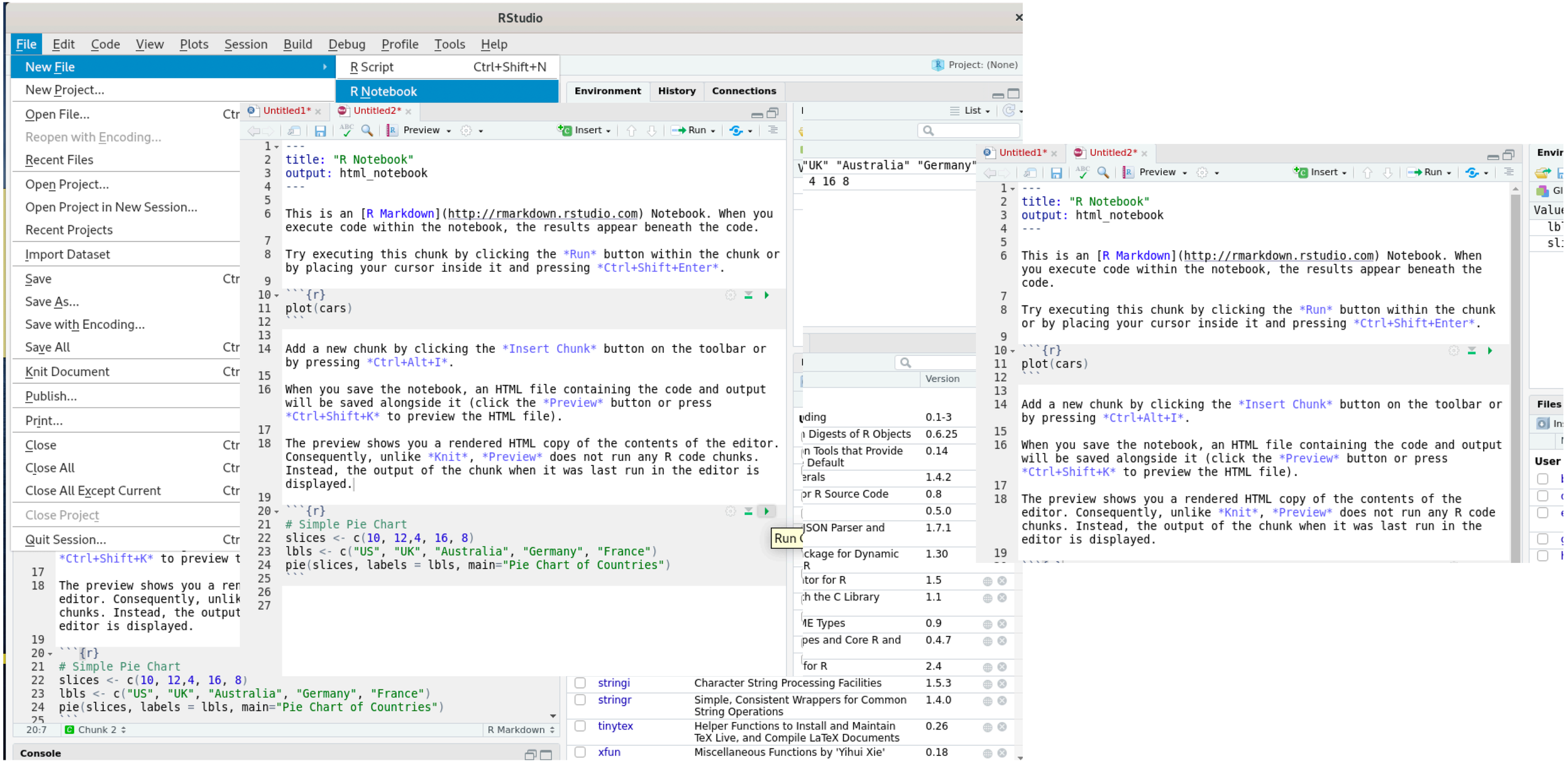
- Why use R?
  - Community
  - Free, open-source
  - Useful for statistics and data visualization

- How to use R?
  - We will use RStudio, a user-friendly interface

Install R Studio

- https://www.rstudio.com/products/rstudio/download/#download
- sudo dpkg -i rstudio-2021.09.0-351-amd64.deb

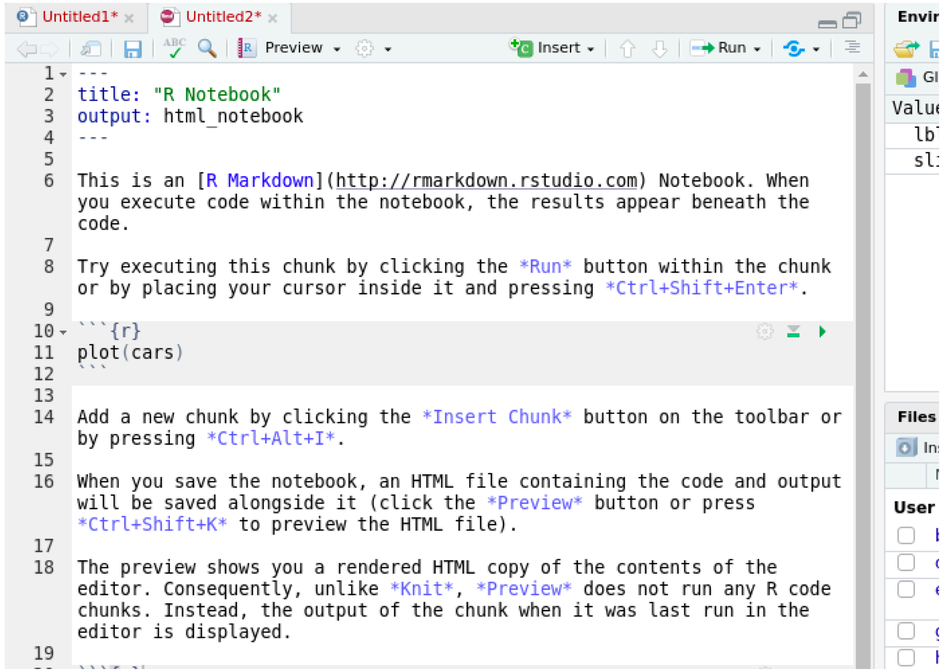
R Markdown

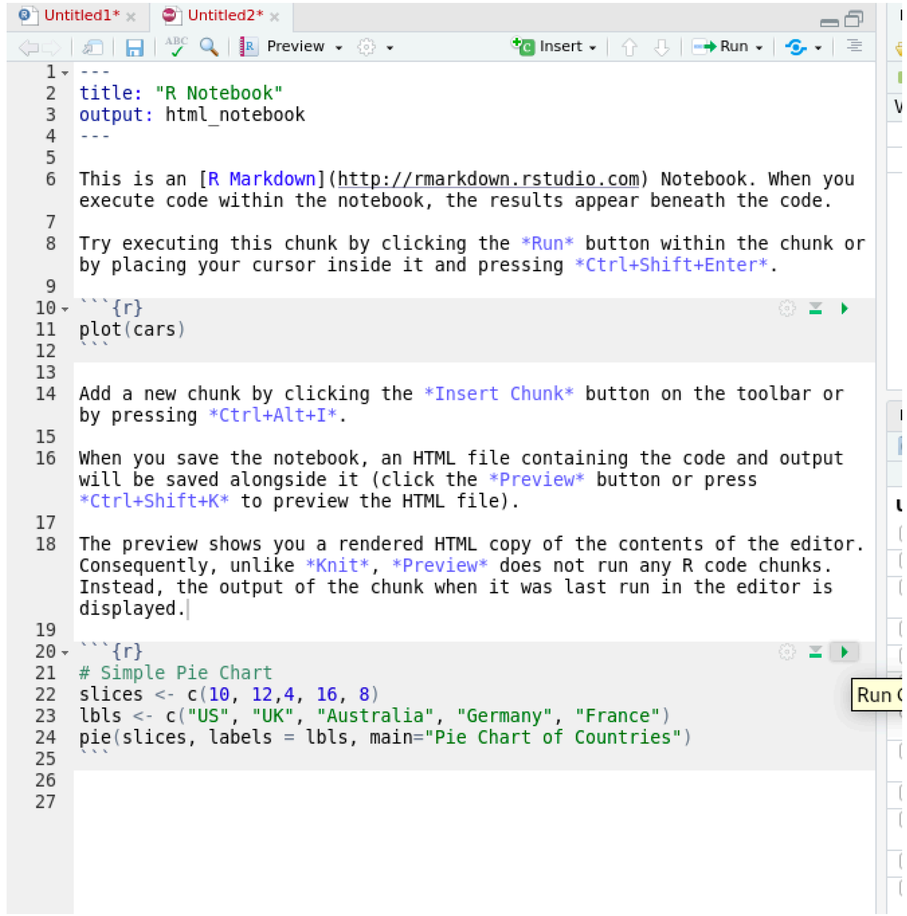

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
New File
New Project...
Open File...
Reopen with Encoding...
Recent Files
Open Project...
Open Project in New Session...
Recent Projects
Import Dataset
Save
Save As...
Save with Encoding...
Save All
Knit Document
Publish...
Print...
Close
Close All
Close All Except Current
Close Project
Quit Session...
R Script Ctrl+Shift+N
R Notebook
Project: (None)
Environment History Connections
Untitled1* Untitled2*
Preview Insert Run
---
title: "R Notebook"
output: html_notebook

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
```{r}
plot(cars)
```
Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
```{r}
# Simple Pie Chart
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```
20:7 Chunk 2 R Markdown
Console
"UK" "Australia" "Germany"
4 16 8
Version
stringi Character String Processing Facilities 1.5.3
stringr Simple, Consistent Wrappers for Common String Operations 1.4.0
tinytex Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents 0.26
xfun Miscellaneous Functions by 'Yihui Xie' 0.18

R Markdown

R notebook

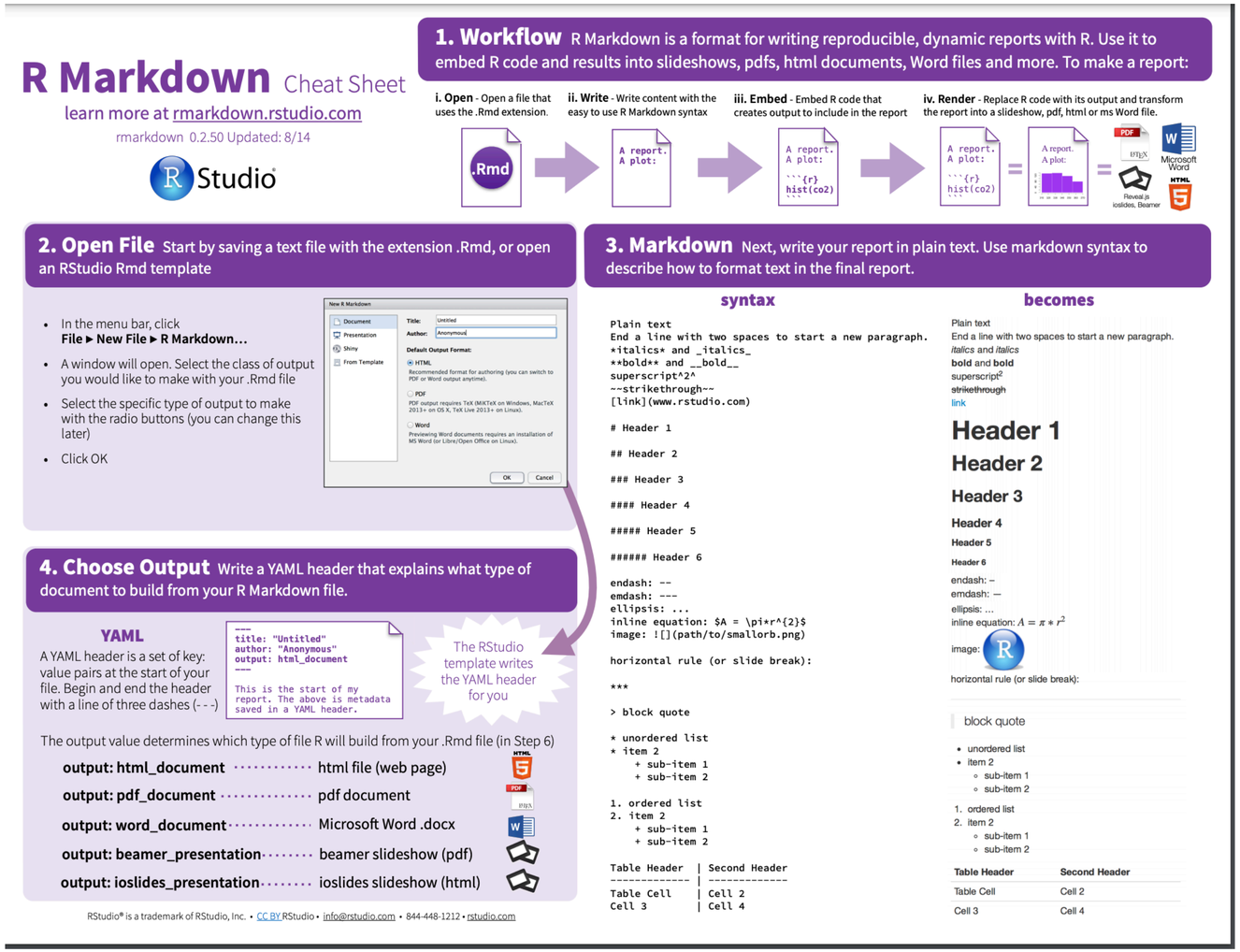





```
# Simple Pie Chart
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```

R Markdown Cheat Sheet

https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf



R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com

rmarkdown 0.2.50 Updated: 8/14

RStudio

1. Workflow R Markdown is a format for writing reproducible, dynamic reports with R. Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more. To make a report:

i. Open - Open a file that uses the .Rmd extension.

ii. Write - Write content with the easy to use R Markdown syntax

iii. Embed - Embed R code that creates output to include in the report

iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.

.Rmd

```
A report.
A plot:
```

```
A report.
A plot:
```{r}
hist(co2)
```
```

```
A report.
A plot:
```{r}
hist(co2)
```
```

```
A report.
A plot:
```

PDF, LaTeX, Microsoft Word, Reveal.js, ioslides, Beamer, HTML

2. Open File Start by saving a text file with the extension .Rmd, or open an RStudio Rmd template

- In the menu bar, click **File ▶ New File ▶ R Markdown…**
- A window will open. Select the class of output you would like to make with your .Rmd file
- Select the specific type of output to make with the radio buttons (you can change this later)
- Click OK

3. Markdown Next, write your report in plain text. Use markdown syntax to describe how to format text in the final report.

syntax

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
superscript^2^
~~strikethrough~~
[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi*r^{2}$
image: ![](path/to/smallorb.png)

horizontal rule (or slide break):

***

> block quote

* unordered list
* item 2
    + sub-item 1
    + sub-item 2

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2

Table Header  | Second Header
------------- | -------------
Table Cell    | Cell 2
Cell 3        | Cell 4
```

becomes

Plain text
End a line with two spaces to start a new paragraph.
italics and *italics*
bold and **bold**
superscript^2
~~strikethrough~~
link

Header 1
Header 2
Header 3
Header 4
Header 5
Header 6

endash: –
emdash: —
ellipsis: …
inline equation: $A = \pi * r^2$
image:
horizontal rule (or slide break):

> block quote

- unordered list
- item 2
    - sub-item 1
    - sub-item 2

1. ordered list
2. item 2
    - sub-item 1
    - sub-item 2

Table Header	Second Header
Table Cell	Cell 2
Cell 3	Cell 4

4. Choose Output Write a YAML header that explains what type of document to build from your R Markdown file.

YAML

A YAML header is a set of key: value pairs at the start of your file. Begin and end the header with a line of three dashes (- - -)

```
---
title: "Untitled"
author: "Anonymous"
output: html_document
---

This is the start of my
report. The above is metadata
saved in a YAML header.
```

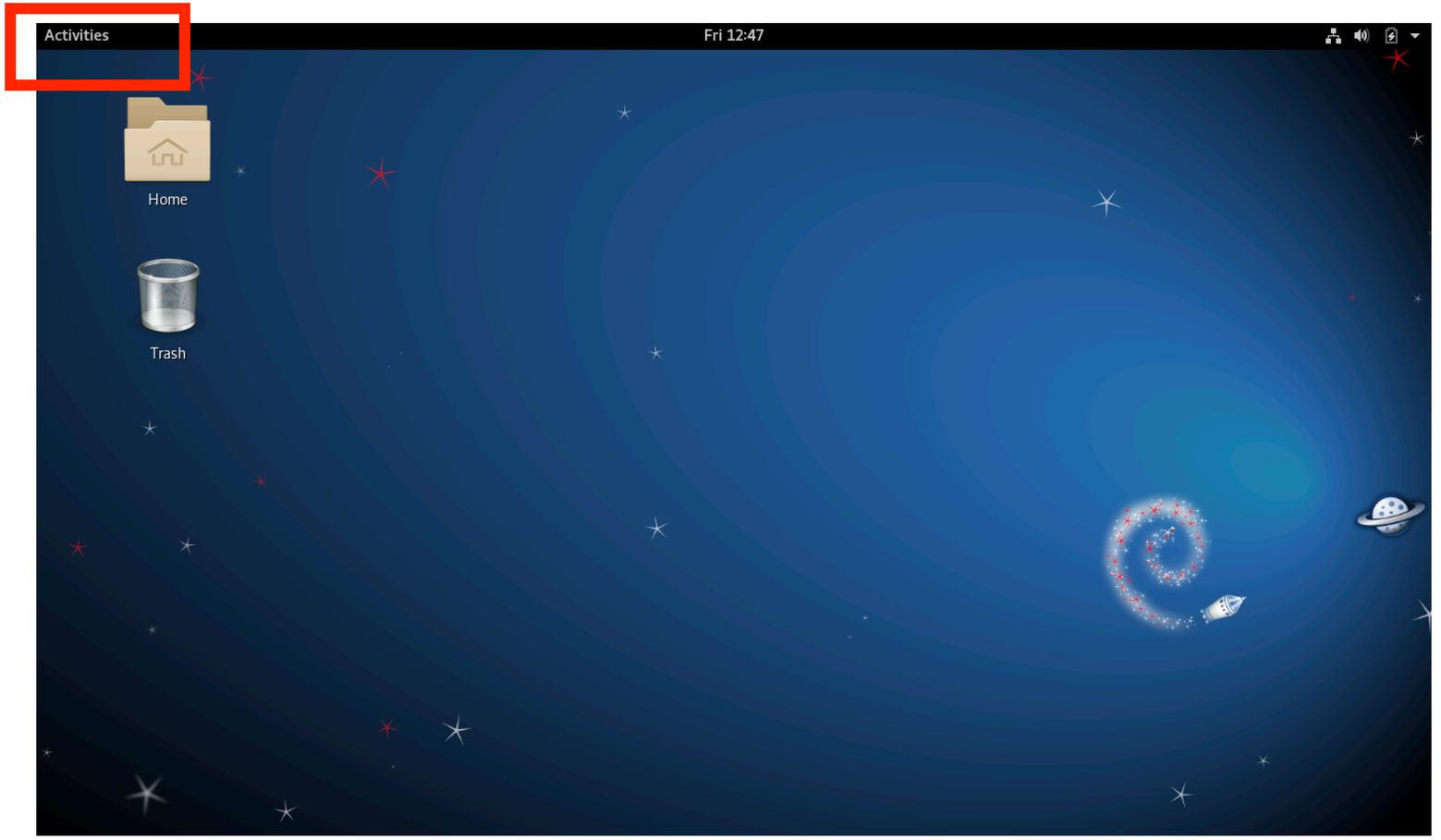
The RStudio template writes the YAML header for you

The output value determines which type of file R will build from your .Rmd file (in Step 6)

output: html_document	html file (web page)
output: pdf_document	pdf document
output: word_document	Microsoft Word .docx
output: beamer_presentation	beamer slideshow (pdf)
output: ioslides_presentation	ioslides slideshow (html)

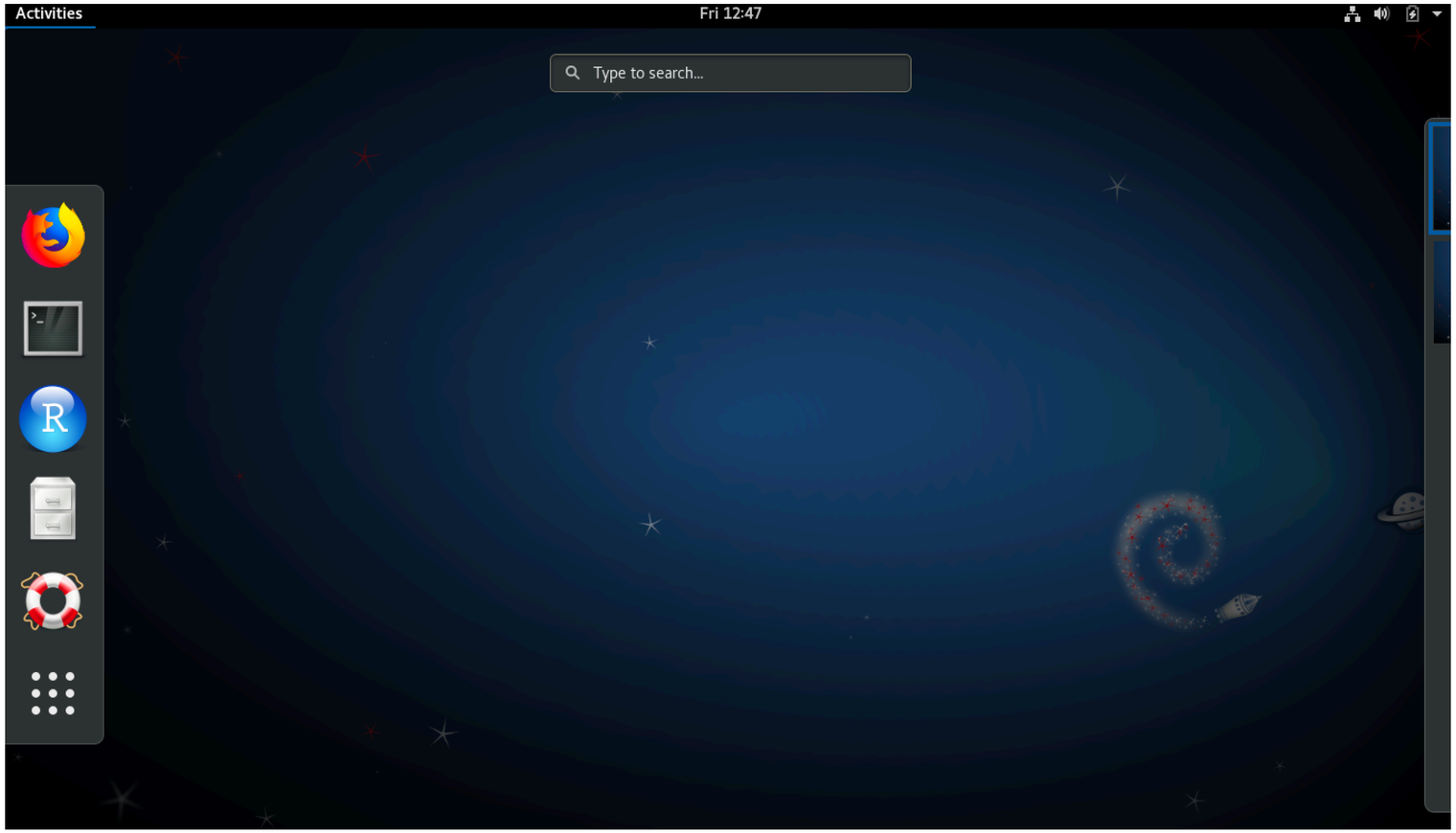
General Introduction - R and RStudio

- Let’s open RStudio



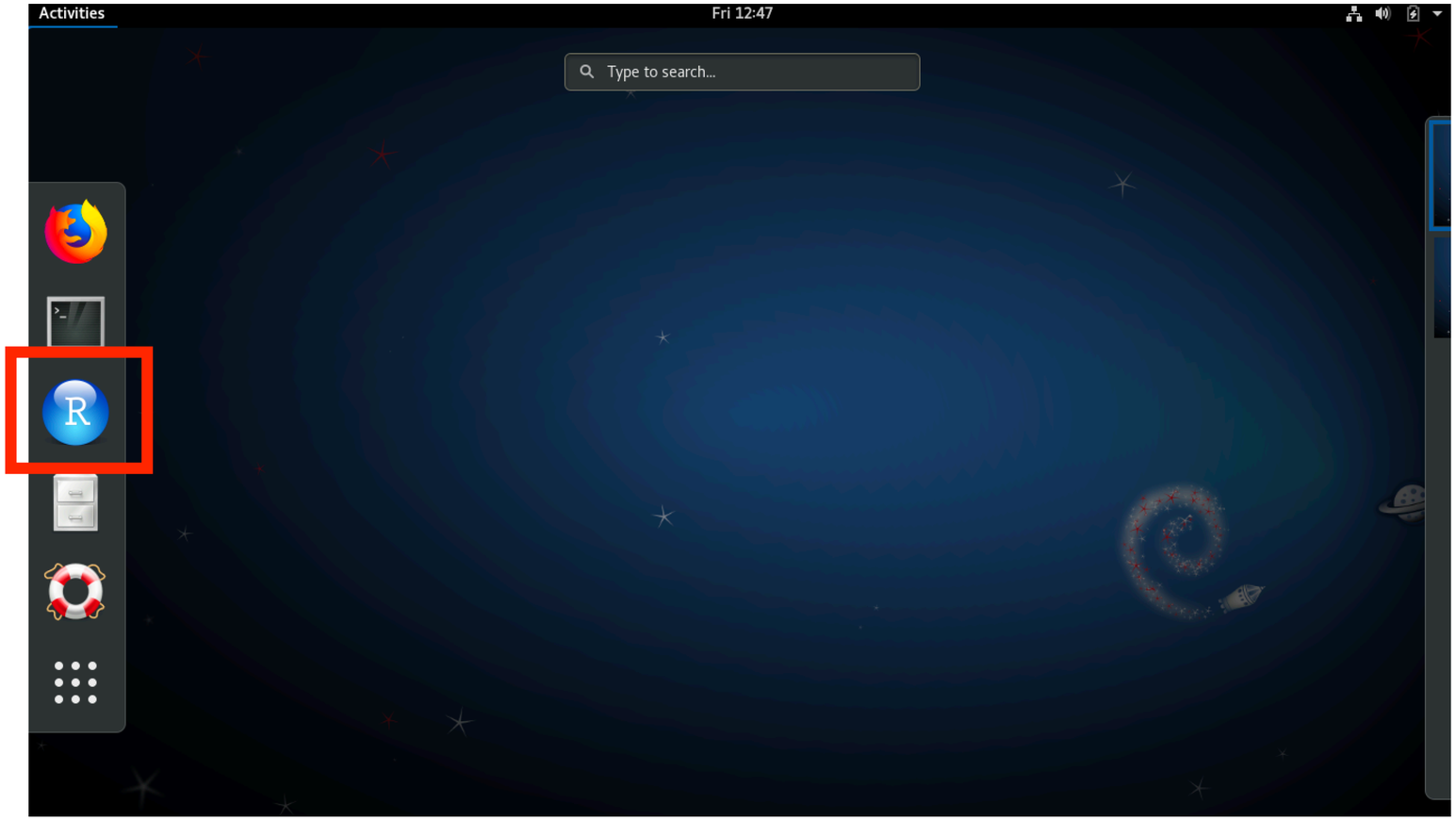
General Introduction - R and RStudio

- Let’s open RStudio



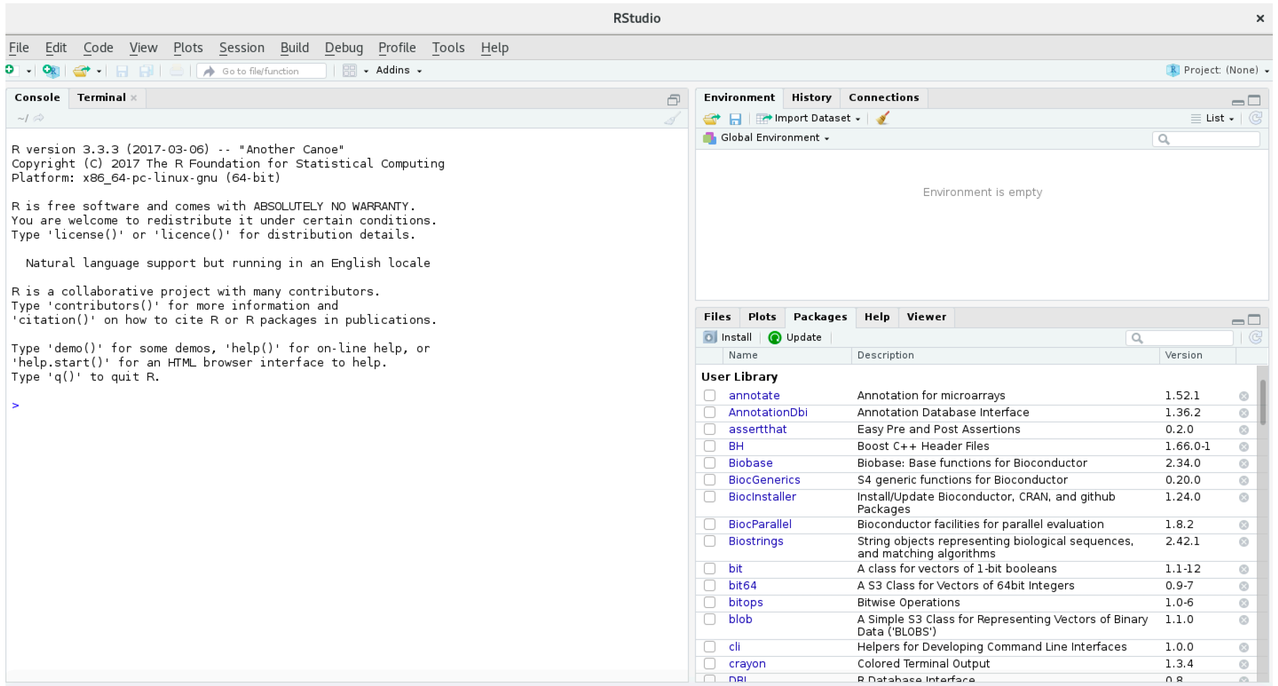
General Introduction - R and RStudio

- Let’s open RStudio



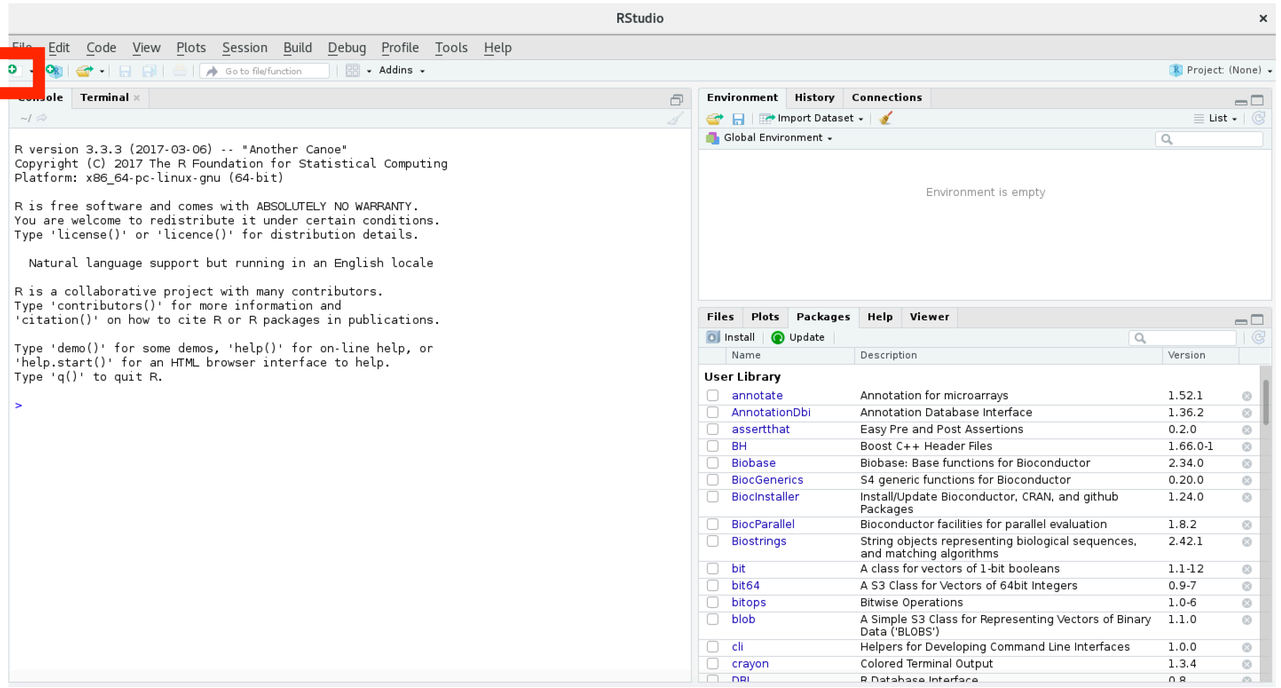
General Introduction - R and RStudio

- Getting oriented in RStudio



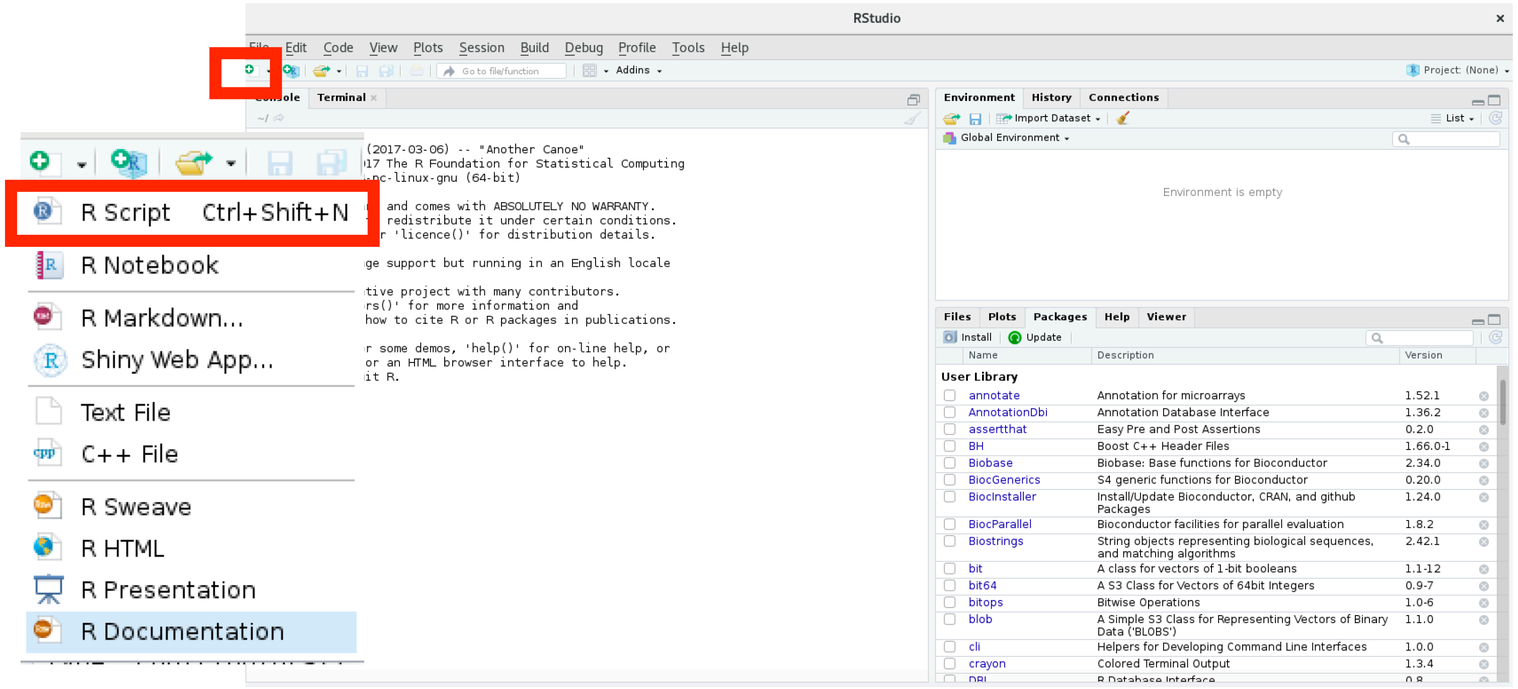
General Introduction - R and RStudio

- Getting oriented in RStudio



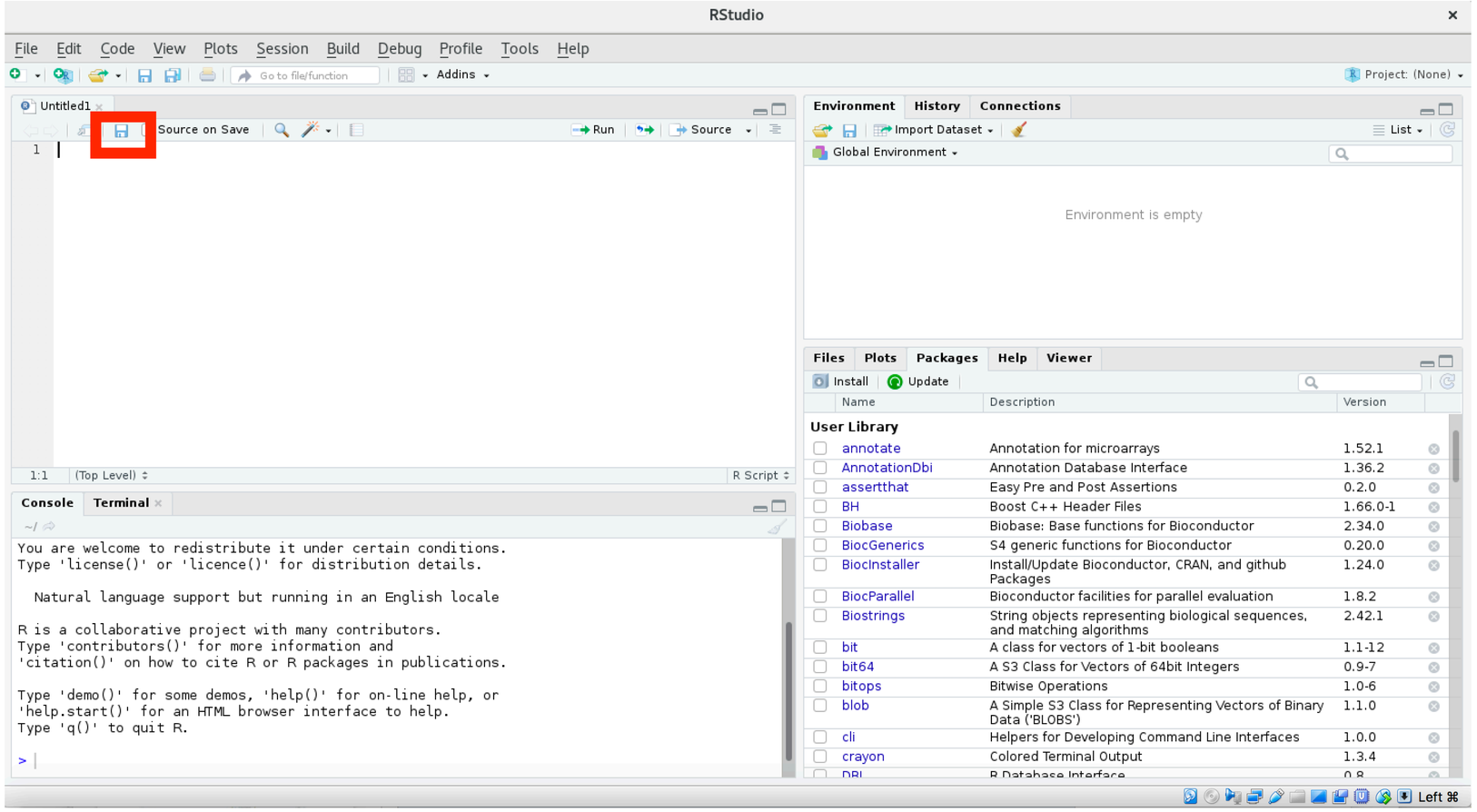
General Introduction - R and RStudio

- Getting oriented in RStudio



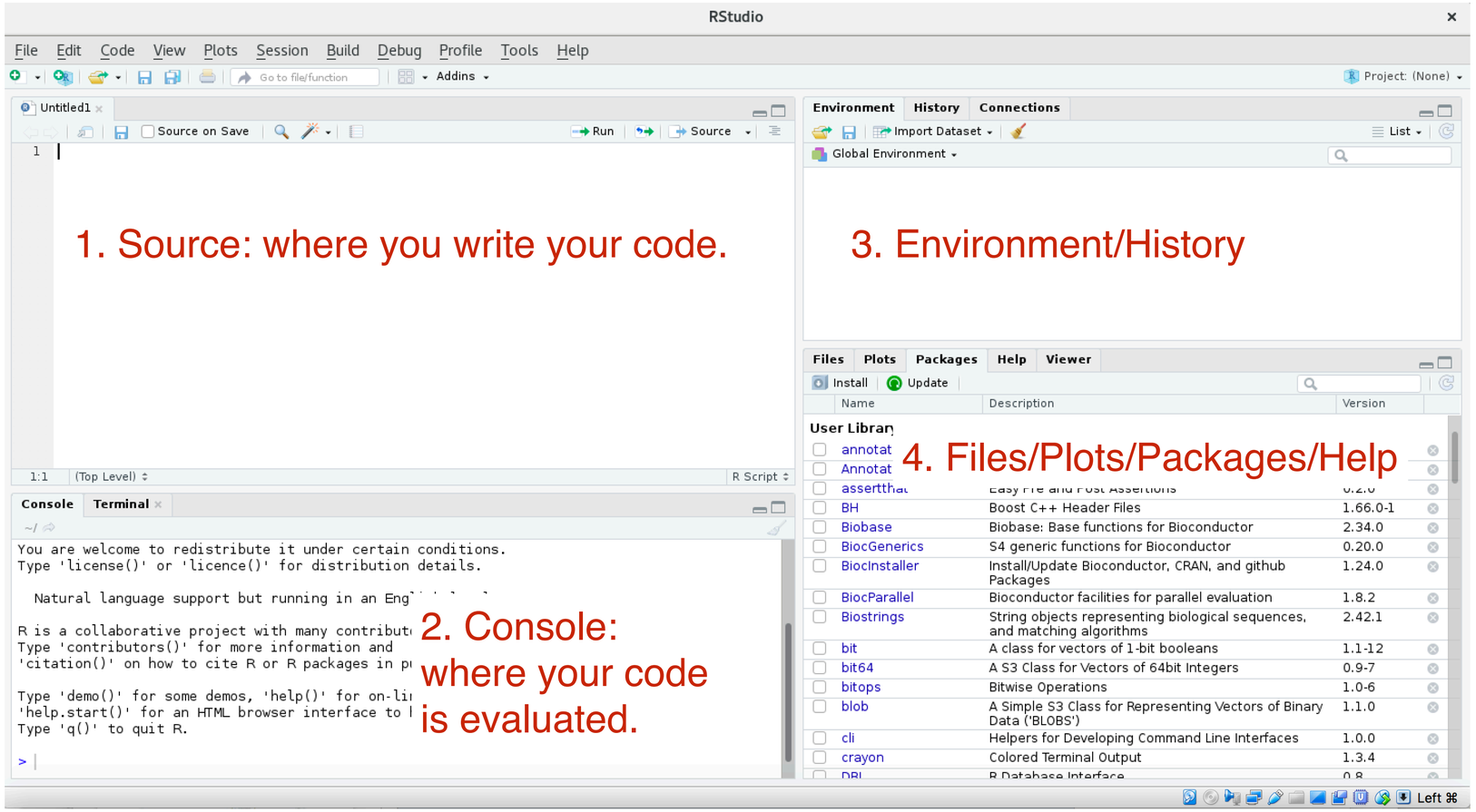
General Introduction - R and RStudio

- Getting oriented in RStudio



General Introduction - R and RStudio

- Getting oriented in RStudio



Location & Working Directory

To see your working directory (similar to pwd in Terminal):

```
getwd()
```

To set your working directory (similar to cd in Terminal):

```
setwd("./")
```

To list what is in the directory (similar to ls in Terminal):

```
dir()
```

BCBC

BTI

Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

```
56+29
```

Multiplication:

```
3*12
```

Division:

```
40/8
```

Exponents:

```
2^12
```

Try to calculate the following:

```
( (8 / 4) - 4.8 + (12 / (2+1) ) )^3
```

Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

```
56+29
```

Multiplication:

```
3*12
```

Division:

```
40/8
```

Exponents:

```
2^12
```

Try to calculate the following:

```
( (8 / 4) - 4.8 + (12 / (2+1) ) )^3
```

1.728

BCBC

BTI

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

```
number.of.round.progeny.peas <- number.of.total.progeny.peas * (1 - ratio.wrinkled)

number.of.round.progeny.peas
```

BCBC

BTI

Scalar Variables & Data Types

- A few tips
  - Choose meaningful variable names
  - Separate words using dots or underscores (no spaces allowed)
  - Variable names are case-sensitive

Scalar Variables & Data Types

We can use different data types in our variables. To check the type of a variable, use class():

Numeric

```
my_lucky_number <- 7

class(my_lucky_number)

class(number.of.round.progeny.peas)
```

Character

```
cool_fruit <- "tomato"

class(cool_fruit)
```

BCBC

BTI

Scalar Variables & Data Types

Logical

```
my_name_is_Adrian <- TRUE

class(my_name_is_Adrian)
```

Scalar Variables & Data Types

Logical

```
my_name_is_Adrian <- TRUE

class(my_name_is_Adrian)
```

What would class() tell you for the following?

```
tomato_is_the_best_fruit <- "TRUE"

class(tomato_is_the_best_fruit)
```

BCBC

BTI

Functions

- A function is basically just a set of instructions
  - Takes some input
  - Does some things
  - Gives an output

- We can access functions in a few ways…

Functions

- We can define our own custom functions

Let's define and use a simple function:

```
my_awesome_function <- function(input1){
  output <- input1^12 * 5 + 3
  return(output)
}

my_awesome_function(3)

my_awesome_function(5)

my_awesome_new_variable <- my_awesome_function(7)

my_awesome_new_variable
```

BCBC

BTI

Functions

- R also has ‘pre-made’ functions… can you think of any examples that we’ve used?

Functions

- R also has ‘pre-made’ functions… can you think of any examples that we’ve used?

To access R documentation for a function, etc.:

```
?class

?plot
```

Functions

- We can also install packages and load libraries to access sets of additional functions…

Packages

- Let’s try installing/loading a package called ggplot2:

```
install.packages("ggplot2")

library(ggplot2)
```

Data Structures

- Vectors
- Matrices
- Data frames
- Lists

Data Structures: Vectors

```
vector1 <- c(1,2,5.3,6,-2,4)
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector2 <- c("one","two","three")
vector2
```

```
## [1] "one"   "two"   "three"
```

```
vector3 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
vector3
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE
```

```
vector4 <- rep(1:3, each=2)
vector4
```

```
## [1] 1 1 2 2 3 3
```

Data Structures: Vectors

```
vector5 <- rep(c(1, 0, 7), times = 5) #repeats c(1,0,7) vector 5 times
vector5
```

```
##  [1] 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7
```

```
vector6 <- rep(c(0, 4), times = c(5,3)) # you could tell R how many times to repeat each value.
vector6
```

```
## [1] 0 0 0 0 0 4 4 4
```

```
vector7 <- rep(1:4,length.out=6) # you could also repeat the vector to the specifield length even if th
vector7
```

```
## [1] 1 2 3 4 1 2
```

```
vector8 <- seq(from = 7.5, to = 2.5, by = -0.5) # vector with a numbers 7.5 to 2.5 in steps of 0.5.
vector8
```

```
##  [1] 7.5 7.0 6.5 6.0 5.5 5.0 4.5 4.0 3.5 3.0 2.5
```

Data Structures: Vectors

We can transpose a vector:

```
t(vector1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2  5.3    6   -2    4
```

We can access elements in a vector using the element's vector position:

```
vector1[c(2,4)]
```

```
## [1] 2 6
```

We can add, subtract, multiply, divide, square root, and exponentiate vectors and scalars of numeric class. These can be done using operations:

```
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector7
```

```
## [1] 1 2 3 4 1 2
```

```
vector1 + vector7
```

```
## [1]  2.0  4.0  8.3 10.0 -1.0  6.0
```

Data Structures: Vectors

We can apply functions on vectors:

```
mean(vector1)
```

```
## [1] 2.716667
```

```
median(vector1)
```

```
## [1] 3
```

Data Structures: Matrices

We can create matrix using matrix(c(), byrow=, ncol=). Matrices can mix element class types.

```
matrix1 <- matrix(c(1,3,2,2,8,9), ncol=2)
matrix1
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    8
## [3,]    2    9
```

```
matrix2 <- matrix(c(1,3,2,2,8,9), ncol=2, byrow = T)
matrix2
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    2
## [3,]    8    9
```

Data Structures: Matrices

We can access elements in matrix using element position:

```
c(matrix1[1,2], matrix1[2,2],matrix1[3,1])
```

```
## [1] 2 8 2
```

Matrix operations (matrix multiplication, etc.) can also be performed.

Transpose of matrices (It is basically to interchange of rows and columns:

```
t(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    2    8    9
```

We can also create vectors from a matrix:

```
matrix1[2, ]
```

```
## [1] 3 8
```

```
matrix1[ ,2]
```

```
## [1] 2 8 9
```

BTI

Data Structures: Data Frames

Data frames are used for storing data tables. It is usually a list of equal vectors of the same length.

```
c <- c(2, 3, 5)
d <- c("aa", "bb", "cc")
e <- c(TRUE, FALSE, TRUE)
df <- data.frame(c, d, e)
colnames(df) <- c("numeric", "character", "logical") # Add column names
df
```

```
##   numeric character logical
## 1       2        aa    TRUE
## 2       3        bb   FALSE
## 3       5        cc    TRUE
```

BCBC

BTI

Reading Data into R

- Use functions such as:
  - read.table()
  - read.csv()

R Package Repos

- CRAN

```
> install.packages("fortunes")
```

- Bioconductor

```
source("https://bioconductor.org/biocLite.R")
biocLite("S4Vectors")
```

- GitHub

```
library(githubinstall)
githubinstall("AnomalyDetection")
```

When would you use bash vs R vs Python?

- Designing a data analysis pipeline that runs several commands and programs
- Writing a more sophisticated program to deploy a machine learning approach on a large data set
- Performing statistical analysis on a large data set

When would you use bash vs R vs Python?

- Bash
  - Simple, great performance wise, good for piping output from one command into another command
  - Gets complicated with larger projects, not very elegant
- Python
  - Is more general purpose and can be used beyond data analysis, elegant
  - Not as many libraries as R
- R
  - Great for visualization (plotting) and for statistical analysis
  - Can be slow, code not as elegant, can be more difficult to learn
- Also depends on what you are comfortable with, some people prefer R, some prefer Python.

- All scripts from today will be added to GitHub here (in addition to on Canvas):

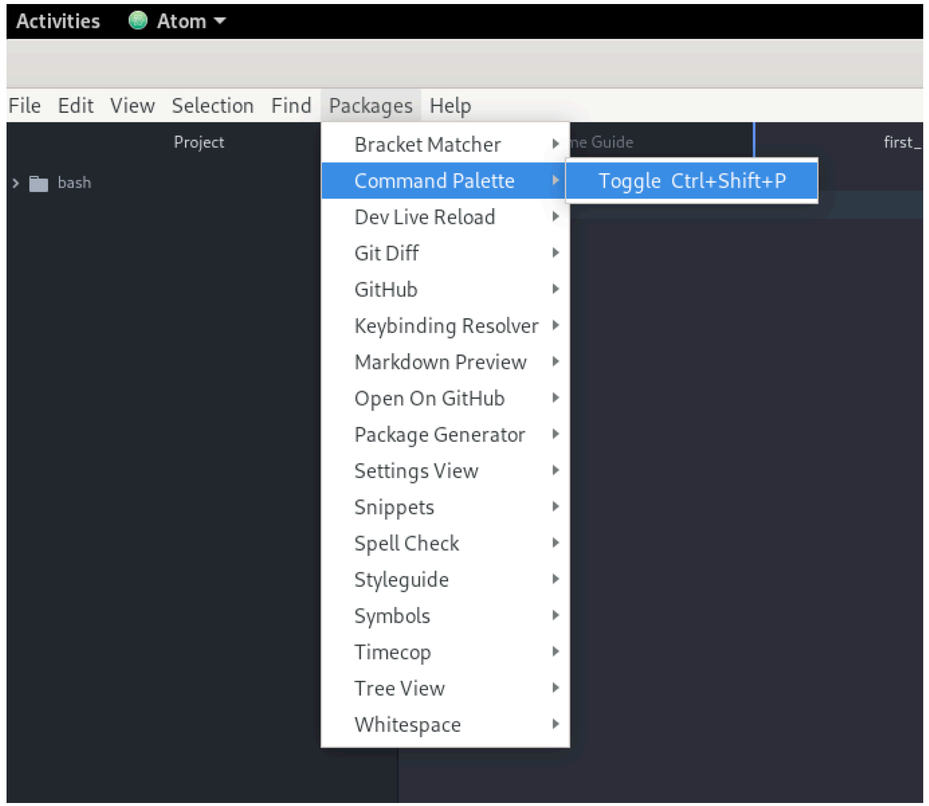
https://github.com/bcbc-group/PLSCi7202

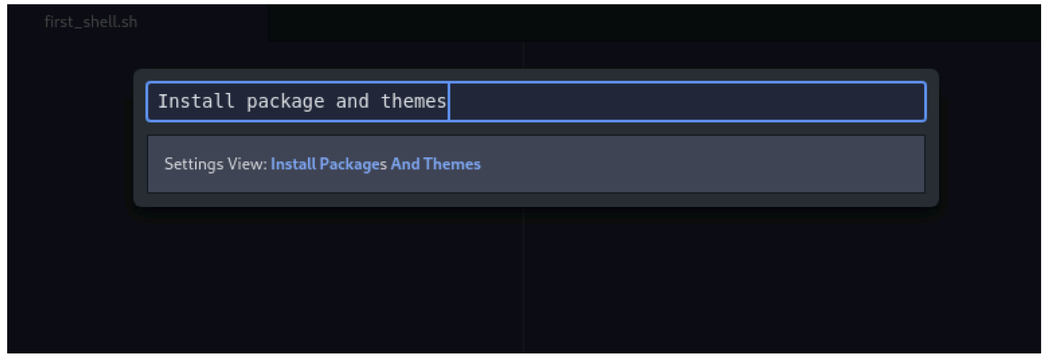
Assignment for next time

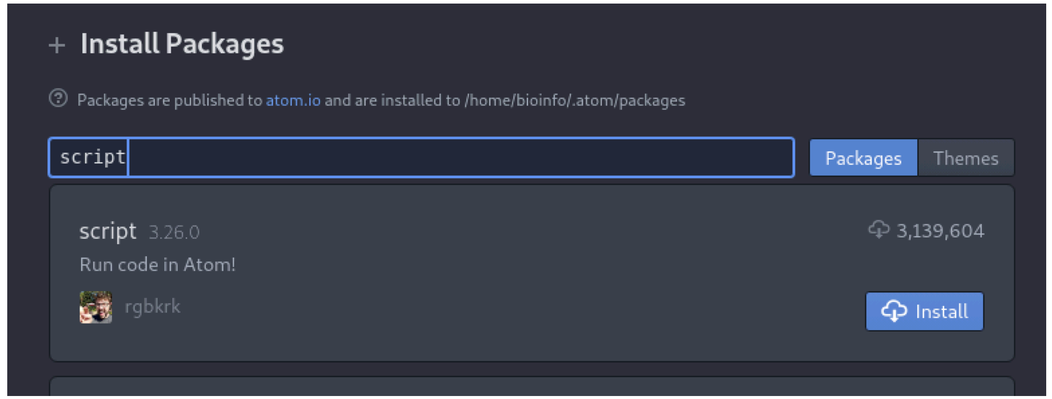
- Write an R or Python script that performs a task that will be useful for your research. If relevant, you can automate it with a bash shell script. Feel free to discuss with others!

- Please complete your script(s) before next class. We will use these in our next class to demonstrate best practices.

Install script Package in Atom







- Ctl + shift + b