# Curating Code

Talya Cooper
New York University

Greg Janée
University of California, Santa Barbara

Kay P Maye
Tulane University

Nick Ruhs
Florida State University

Seth Erickson
University of California, Santa Barbara

# Case Study: Willoughby-Hoye Scripts

"While preparing a manuscript, to our surprise, attempts by team members to replicate these results produced different [results] despite using the same [source code] outlined by Willoughby et al." (Neupane et al., 2019)

# Case Study: “Willoughby-Hoye Scripts”

```python
def read_gaussian_outputfiles():
    list_of_files = []
    for file in glob.glob('*.out'):
        list_of_files.append(file)
    return list_of_files
```

Excerpt of original python code from (Willoughby et al., 2014)

```python
def read_gaussian_outputfiles():
    list_of_files = []
    for file in glob.glob('*.out'):
        list_of_files.append(file)
    # ...
    list_of_files.sort()
    return list_of_files
```

Excerpt of revised code (Yuhen Luo and Rui Sun, 2019)

*Is it realistic to expect a curator to catch this sort of mistake?*

# Case Study: “Willoughby-Hoye Scripts”

“This simple glitch in the original script calls into question the conclusions of a significant number of papers on a wide range of topics in a way that cannot be easily resolved from published information **because the operating system is rarely mentioned.**” (Neupane et al., 2019)

# Learning Outcomes (pt.1)

By the end of this workshops, you should be able to …

- Identify and explain real-world examples of how differences between **computing environments** lead to **reproducibility problems**.
- Navigate significant **differences between major operating systems** and understand how they can affect the reusability of software
- Identify several **common programming languages** used in research computing and the file types and tools associated with each.

# Learning Outcomes (pt .2)

- Identify and document **dependencies** used in research software for common programming languages.
- Identify and improve different types of **documentation** associated with source code and research software.
- Describe differences between **software licenses** and non-software licenses, and differences between different kinds of open source software licenses.
- Implement commonly used **project organization strategies** used in the research community.

# Key Terms

- **Source Code** - *plain text*, written in a programming language, that can be run on a computer using an interpreter or compiled into runnable machine code by a compiler.
  - Example: contents of a python/R “script”
- **Software** - encompasses source code, compiled programs, operating systems, libraries, dependencies, packages, scripts, etc..
  - Example: Linux or Windows operating systems; the NumPy package.
- **Source Code Repository** - a centralized location where code is stored and shared, usually, often conjunction with a version control system. (Not to be confused with a data repository!).
  - Example: GitHub and GitLab

# Research Software

"Research Software includes **source code files, algorithms, scripts, computational workflows and executables that were created during the research process or for a research purpose**. Software components (e.g., operating systems, libraries, dependencies, packages, scripts, etc.) that are used for research but were not created during or with a clear research intent should be considered software in research and not Research Software. This differentiation may vary between disciplines. The minimal requirement for achieving computational reproducibility is that all the computational components (Research Software, software used in research, documentation and hardware) used during the research are identified, described, and made accessible to the extent that is possible." (Gruenpeter et al, 2021)

# Overview

1. Introduction
2. Computing Platforms
3. Programming Languages
4. Dependencies
5. Documentation
6. Licensing *
7. Project Structure *
8. Exercise

* - not covered due to time constraints

# Example Dataset

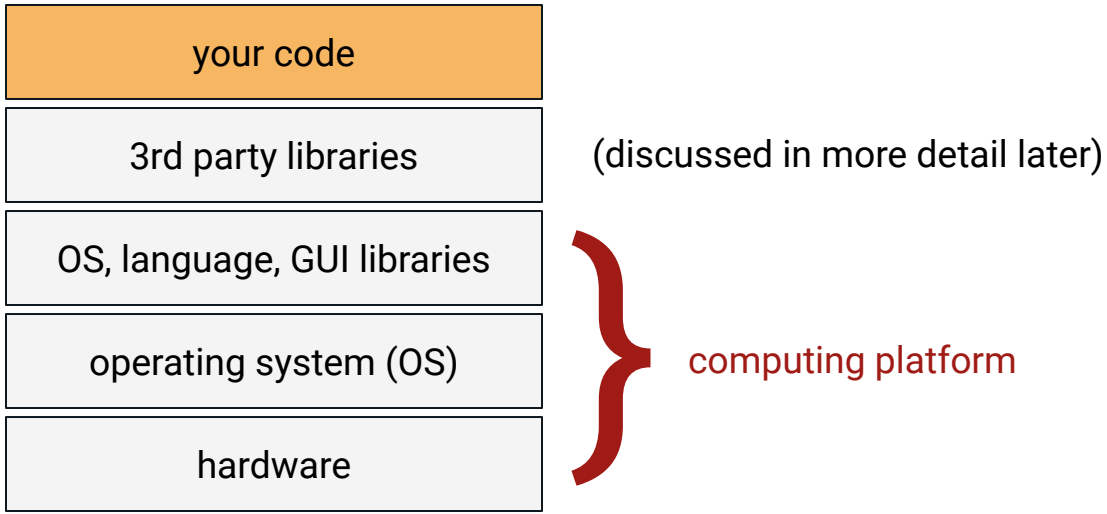
- *Code, data, and metadata document for the manuscript: "Density-dependence in wolf resource selection study designs"*
- The submission includes data and analysis code from study of wolf habitat preference.
- We'll return to this example in each section of the workshop.
- See files in course materials.
- This is not an authentic dataset! (though it is derived from one)

# 2. Computing Platforms

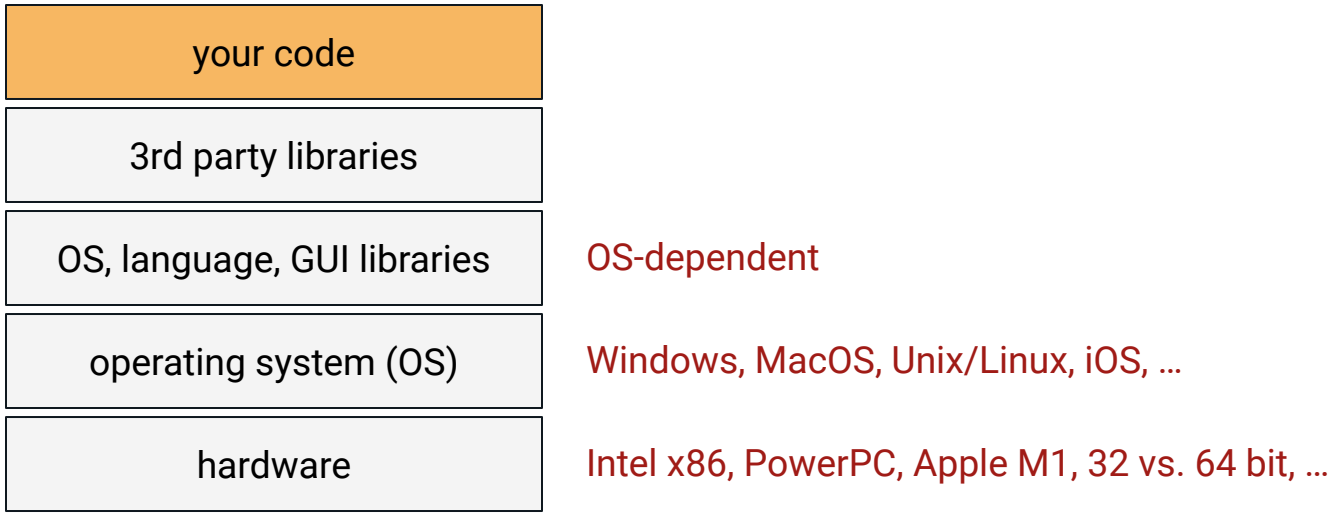
# Learning Outcomes

- Understand elements of computing platforms
- Understand differences between compiled and interpreted languages
- Understand obstacles to reproducibility
- Understand implications for curation

# Computing platforms



# Central obstacle to reproducibility



# Compiled vs interpreted languages

- Portability of code depends greatly on whether the code is written in a **compiled** language or in an **interpreted** language

# Compiled languages

- Must be “compiled” into a “binary/executable/program/app”
  - Once compiled, the binary is entirely platform-specific and it is directly executable; the source code is not needed
- Compiled examples:
  - Languages: C, C++, FORTRAN, Go, Rust
  - Programs: most every program/app on your laptop or phone
- Code in compiled languages is often platform-dependent
- Compilation process is difficult

# Interpreted languages

- Require a program called an “interpreter” to run
  - Code is directly executed from source code every time
- Examples: Python, R, Stata, SAS, MATLAB, Java, Perl, Mathematica
- Typically excellent portability

# Language type comparison

| | Compiled | Interpreted |
|---|---|---|
| **Distribution form** | Source code | Source code |
| **Your responsibility** | Compile code **and** 3rd party libraries | Load 3rd party libraries from central repository |
| **OS provides (intrinsically or by add-on)** | Compiler(s), package manager | Interpreter |

# Where platform incompatibilities get addressed

| | Compiled | Interpreted |
|---|---|---|
| **Distribution form** | Source code | Source code |
| **Your responsibility** | Compile code **and** 3rd party libraries | Load 3rd party libraries from central repository |
| **OS provides (intrinsically or by add-on)** | Compiler(s), package manager | Interpreter |

# Containerization

- Relatively new technology in libraries & curation
- A “container image” bundles a program with all its library dependencies, down to the operating system libraries
  - and possibly data as well

# Computing platforms


(binary) container image

# Containerization features

- Images are run from/within a **container engine** (e.g., Docker)
- Images can be stored in repositories
  - Can serve as building blocks, have their own dependencies
- Images are opaque
- Images retain dependencies on OS, machine architecture

# Language type comparison

| | Compiled | Interpreted | Containerized |
|---|---|---|---|
| **Distribution form** | Source code | Source code | Container image |
| **Your responsibility** | Compile code **and** 3rd party libraries | Load 3rd party libraries from central repository | None |
| **OS provides (intrinsically or by add-on)** | Compiler(s), package manager | Interpreter | Container engine (that supports container's OS) |

# Implications for curation

- Interpreted code is highly portable
  - Document language and dependency versions
- Compiled code is likely **not** runnable without significant work
  - Document the environment it was developed in
- Container images: reproducible, but opaque
  - Capture build file

# Four additional incompatibility sources

- Pathnames
- Character encoding
- Text file format
- Nondeterminism

# Pathnames

- All platforms support similar concepts of **files** (named unit of information storage) and **directories** (named collection of files), organized hierarchically
- **Pathnames** are used for identification
  - /Users/alice/my_project/output_data/final.csv

# Pathname incompatibilities

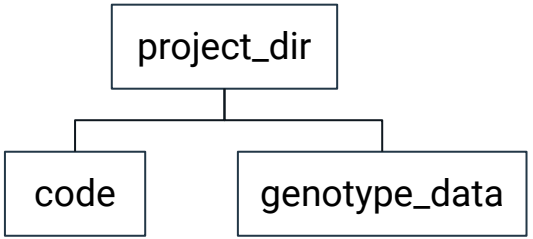
- Repertoire of allowable characters differs across platforms
- Case-sensitivity differs
  - README.txt, Readme.txt, REadME.TXT
- Pathname syntax
  - Unix: /Users/alice/my_project/output_data/final.csv
  - Windows: C:\Users\alice\my_project\output_data\final.csv

# Absolute vs. relative pathnames

- Absolute pathname
  - Identifies starting from filesystem root
    - /Users/alice/my_project/output_data/final.csv
- Relative pathname
  - Identifies relative to another (contextual) directory
    - output_data/final.csv

# Quiz: Which are good path/filenames?

- AC/DC Voltages.dat
- Buñuel Luis.xml
- /raid/data/spot/554/228/5542281050329184915 1J-9.tiff
- ../genotype_data/hap_G100.A1.O_X.R
- BT_Palmyra–2015.csv (look close)



# Implications for curation

- Watch out for absolute pathnames
  - Actual example:
  - `setwd("C:\\Users\\soneil\\Documents") # change me`
- Encourage use of simple characters that transport well
  - Yes: A-Z a-z 0-9 . - _
  - No: é, – (en dash), (space), 😀

# Other platform incompatibilities

- Character mis-encoding
  - “Beno√ÆtLalibertÃ©” instead of “Benoît Laliberté”
- Subtle differences in plain text file formats

# Character encoding

- Text characters are internally represented as numbers; the assignment of characters to numbers is called an **encoding**
  - E.g., 65 = “A”
- Unicode UTF-8 is the *de facto* standard encoding today, but older, platform-specific encodings are still encountered

# Encoding misinterpretation example

- Given “Benoît Laliberté”
- Encoded as UTF-8:
  - 66 101 110 111 195 174 116 32 76 97 108 105 98 101 114 116 195 169
  - B e n o î t L a l i b e r t é
- Decoded as Windows-1252:
  - B e n o Ã ® t L a l i b e r t Ã ©
- Decoded as MacRoman:
  - B e n o √ Æ t L a l i b e r t √ ©

# Implication for curation

- Watch out for character encoding problems
- Researchers don’t always recognize, or know how to fix

# Text file format

- Text files differ in how lines are delimited internally
  - Unix:
    - This is a line LF
      So is this LF
  - MacOS
    - This is a line CR
      So is this CR
  - Windows
    - This is a line CR LF
      So is this CR LF

# Implication for curation

- In most cases, interpreted languages (Python, R) automatically detect text file format and adapt automatically
- Incompatibilities still surface, though, and may cause errors

# Nondeterminism

- Code is generally deterministic: same input produces same output every time
- Nondeterministic code: results can vary
  - Intended
  - Artifact of internal choices, ambiguity
  - A bug

# Nondeterministic: intended

- Monte Carlo simulation

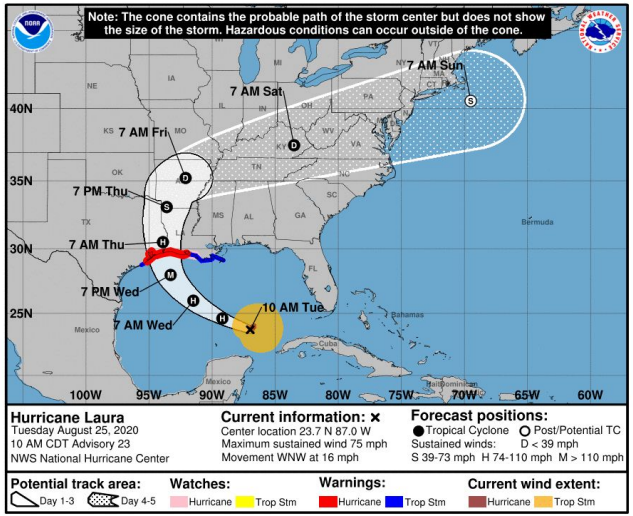



Image source: [NOAA, National Hurricane Center]

# Nondeterministic: artifact

- Consider sorting these students in descending order by grade:
  - 
| Student | Grade |
|---|---|
| Alice | 90 |
| Bob | 75 |
| Cindy | 90 |

- “Alice,Cindy,Bob” or “Cindy,Alice,Bob”?

# Nondeterministic: bug

- Code may depend on behavior that is not guaranteed
- Ex: documentation on Python's "os.listdir" function:
  - "Return a list containing the names of the entries in the directory given by path. The list is in arbitrary order, and does not include the special entries '.' and '..' even if they are present in the directory."

# Implication for curation

- Code that produces (slightly) different results may be intentional or unintentional, may or may not be significant, may or may not reflect a bug
- Requires further investigation

# Discussion: deposit example (1/2)

- Platform: R
  - Interpreted language, portability should be good
  - But documentation of platform could be better: version of R is documented in README, but library versions are missing
  - Consider using `renv` to automatedly capture all version information
- Files and pathnames
  - Reads and writes CSV files, no issues there
  - But uses absolute pathname to set working directory
    - Line 48: `setwd("C:\\Users\\soneil\\Documents")`
  - In this case, not needed, but generally, replace with a relative pathname

# Discussion: deposit example (2/2)

- Repeatability
  - Sets random number seed, but that line is buried in the code
    - Line 81: set.seed(0123)
  - Better to set at top of file, or read in from input file
- Hardware requirements are specified in the README.

# 3. Programming Languages

# Learning Outcomes

- Describe common programming languages used in academic research and their applications
- Identify key file types and extensions associated with MATLAB, Python, and R
- Introduce common Integrated Development Environments (IDE) and Notebooks for these programming languages

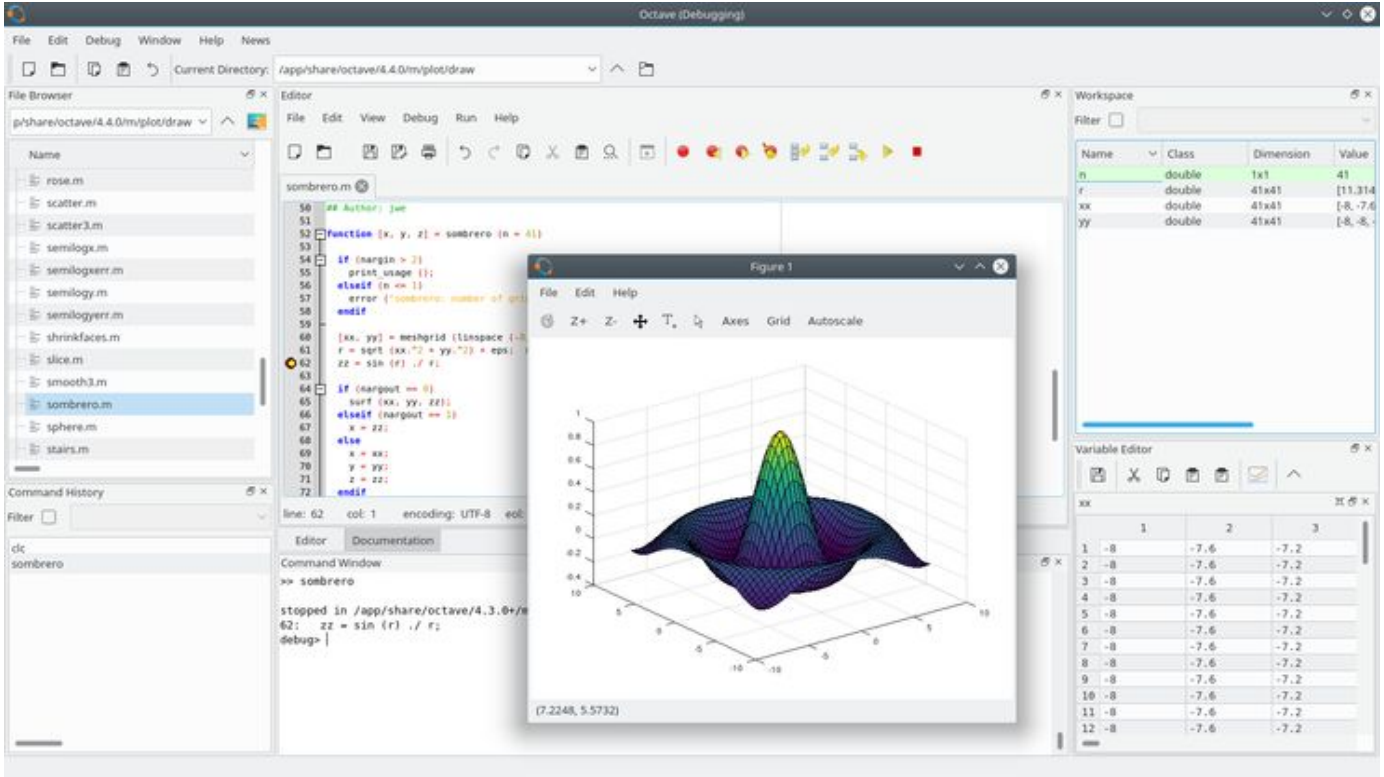
# What is a Programming Language?

- A computer language used to develop software programs, scripts, or other sets of instructions for computers or computing devices to perform specific tasks
- During today's workshop, we will be focusing on MATLAB, Python and R
- Focusing on these languages as they are the most commonly encountered by DCN curators
- All three languages (and many encountered by curators) are interpreted languages

# MATLAB Overview

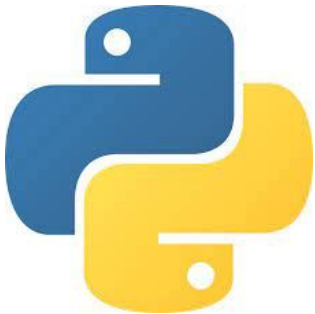
- Numerical computing environment built around the MATLAB scripting language
- Licensed through Mathworks as a proprietary, paid product
- Several open-source alternatives with similar (but not always exact) syntax exist, with GNU Octave being the most common
- Many external toolboxes have been developed for MATLAB, most of which require a separate subscription

# GNU Octave: “Open Source MATLAB”



# Python Overview

- High-level, general purpose programming language with applicability cross a variety of field and research areas
- Easily extendable to other programming languages (i.e. C++)
- Applicability is enhanced by a large number of externally developed libraries and packages.
- Open-source software (Libraries are also open-source)



# R Overview

- R is an open-source programming language that is specifically designed for complex data and statistical analysis
- The programming language itself (and its default console) is often referred to as Base R
- Community-developed “packages” have greatly enhanced the capabilities of R

# Common Research Applications

| MATLAB | Python | R |
|---|---|---|
| Data Analysis | | |
| Data Visualization | | |
| Machine Learning | | |
| Data Modeling | Web Development | Mapping |
| Simulation | Automation or Scripting | Spatial data analysis |
| Linear Algebra | Text Mining | Text mining |
| | Financial Analysis | |

# Common Research Applications

**What are some research applications that you have seen for these languages that are not listed previously?**

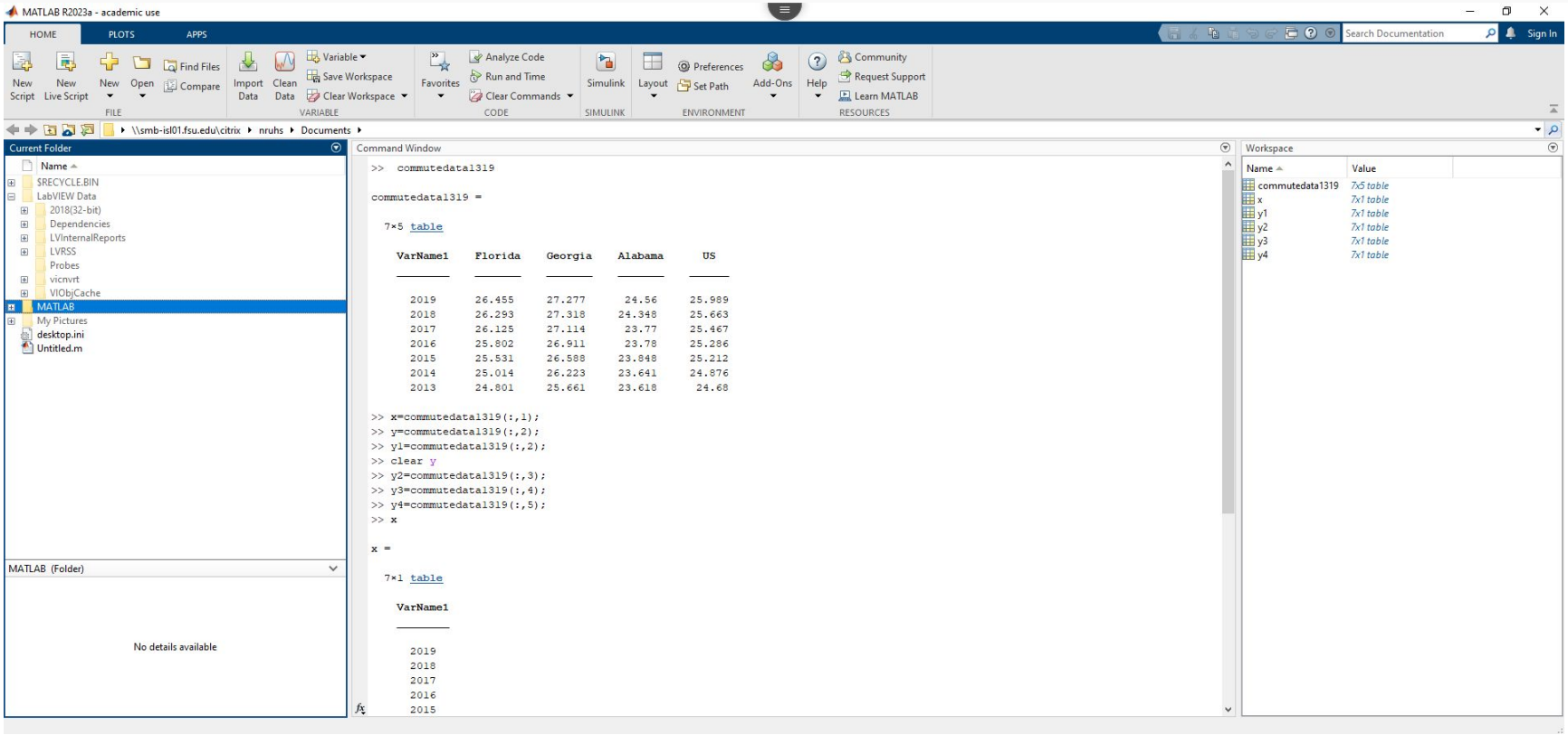
# File Formats

- Each programming language has a specific file format for programming files or scripts.
  - MATLAB - .m
  - Python - .py
  - R - .r
- The scripts can be read as plain text files (.txt) by a text editor, but require the specific software to run
- MATLAB also uses .mat files to store MATLAB formatted data

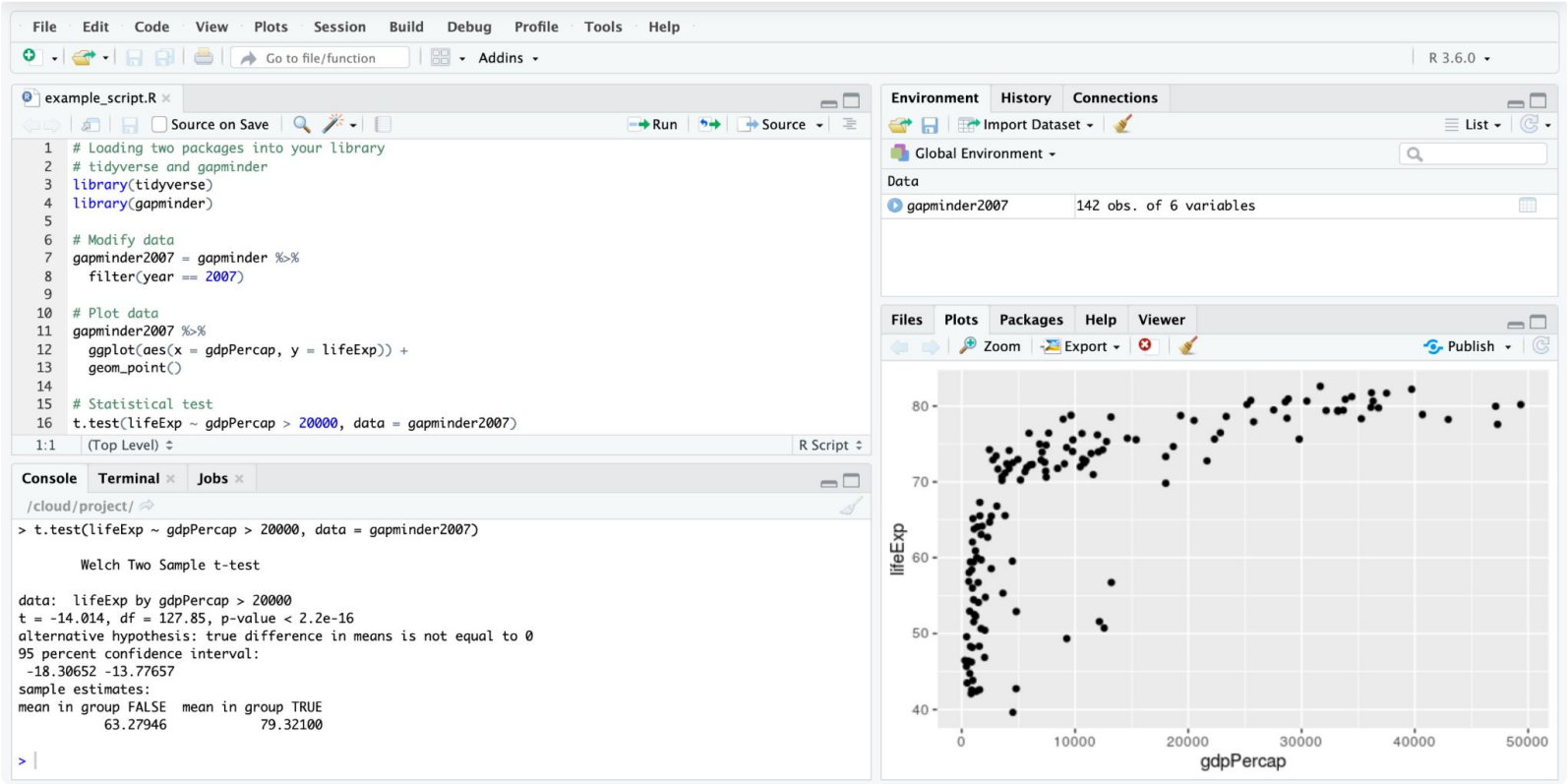
# Integrated Development Environments

- Software applications that provide a suite of tools for code and software development
- IDEs combine several programming tasks into one application
- IDEs allow you to develop, write, and check code written in a particular programming language all in one application
- Some IDEs are specific to certain programming languages, while others are compatible with multiple languages

# MATLAB


MATLAB R2023a - academic use
HOME
PLOTS
APPS
Search Documentation
Sign In
New Script
New Live Script
New
Open
Find Files
Compare
Import Data
Clean Data
Variable
Save Workspace
Clear Workspace
Favorites
Analyze Code
Run and Time
Clear Commands
Simulink
Layout
Preferences
Set Path
Add-Ons
Help
Community
Request Support
Learn MATLAB
FILE
VARIABLE
CODE
SIMULINK
ENVIRONMENT
RESOURCES
\\smb-isl01.fsu.edu\citrix ▸ nruhs ▸ Documents ▸
Current Folder
Name
$RECYCLE.BIN
LabVIEW Data
2018(32-bit)
Dependencies
LVInternalReports
LVRSS
Probes
vicnvrt
VIObjCache
MATLAB
My Pictures
desktop.ini
Untitled.m
MATLAB (Folder)
No details available
Command Window
>> commutedata1319
commutedata1319 =
7×5 table
VarName1 Florida Georgia Alabama US
2019 26.455 27.277 24.56 25.989
2018 26.293 27.318 24.348 25.663
2017 26.125 27.114 23.77 25.467
2016 25.802 26.911 23.78 25.286
2015 25.531 26.588 23.848 25.212
2014 25.014 26.223 23.641 24.876
2013 24.801 25.661 23.618 24.68
>> x=commutedata1319(:,1);
>> y=commutedata1319(:,2);
>> y1=commutedata1319(:,2);
>> clear y
>> y2=commutedata1319(:,3);
>> y3=commutedata1319(:,4);
>> y4=commutedata1319(:,5);
>> x
x =
7×1 table
VarName1
2019
2018
2017
2016
2015
Workspace
Name
Value
commutedata1319 7x5 table
x 7x1 table
y1 7x1 table
y2 7x1 table
y3 7x1 table
y4 7x1 table

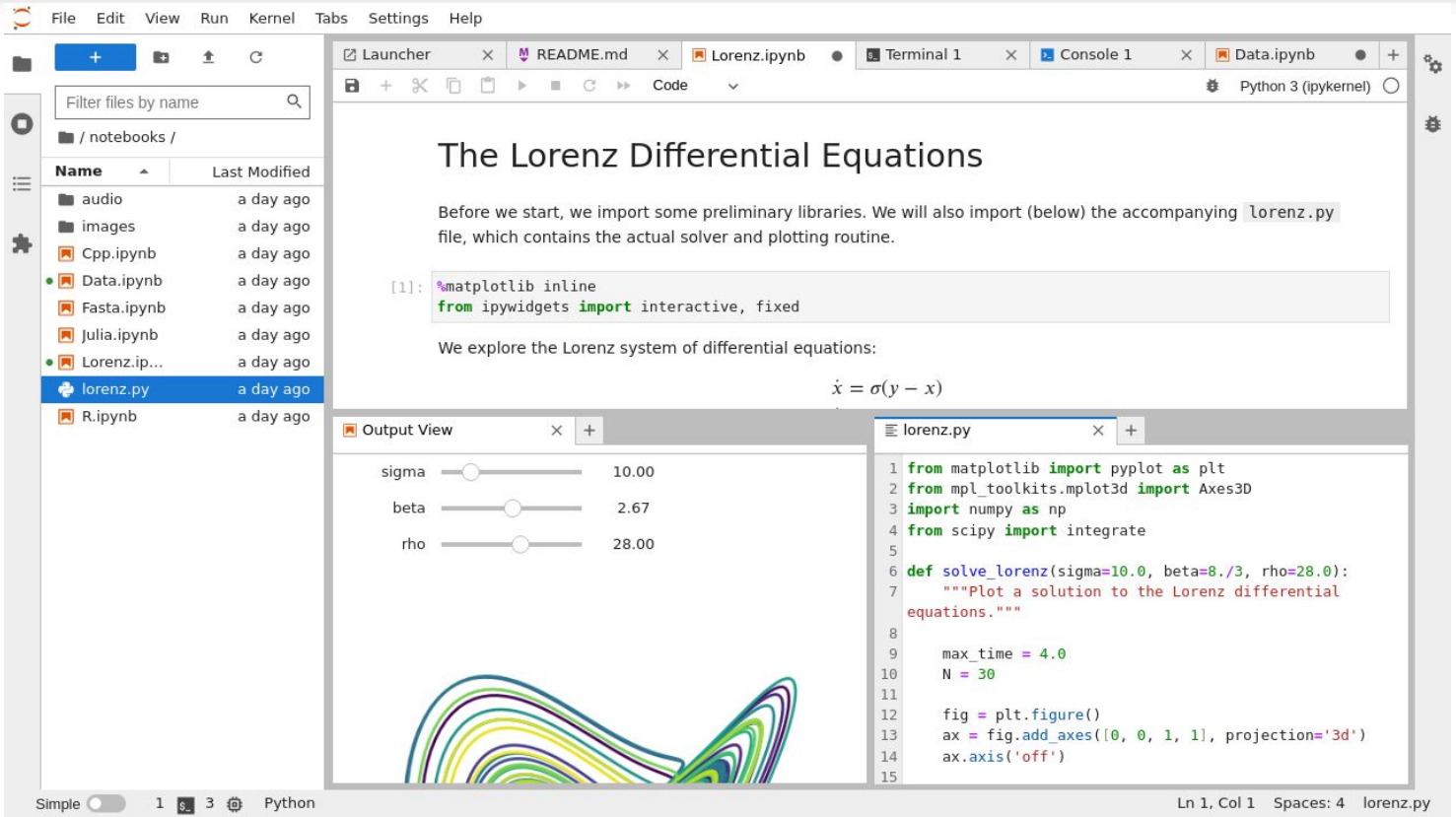
# R Studio


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 3.6.0
example_script.R
Source on Save
Run
Source
1 # Loading two packages into your library
2 # tidyverse and gapminder
3 library(tidyverse)
4 library(gapminder)
5
6 # Modify data
7 gapminder2007 = gapminder %>%
8 filter(year == 2007)
9
10 # Plot data
11 gapminder2007 %>%
12 ggplot(aes(x = gdpPercap, y = lifeExp)) +
13 geom_point()
14
15 # Statistical test
16 t.test(lifeExp ~ gdpPercap > 20000, data = gapminder2007)
1:1 (Top Level) R Script
Console Terminal Jobs
/cloud/project/
> t.test(lifeExp ~ gdpPercap > 20000, data = gapminder2007)
Welch Two Sample t-test
data: lifeExp by gdpPercap > 20000
t = -14.014, df = 127.85, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-18.30652 -13.77657
sample estimates:
mean in group FALSE mean in group TRUE
63.27946 79.32100
>
Environment History Connections
Import Dataset
List
Global Environment
Data
gapminder2007 142 obs. of 6 variables
Files Plots Packages Help Viewer
Zoom
Export
Publish
lifeExp
gdpPercap
80
70
60
50
40
0
10000
20000
30000
40000
50000

# Notebooks

- Notebooks are “executable papers” combining richly formatted text, runnable code, and data visualization in a single file.
- Support different programming languages, including Python and R.
- Text is formatted using markdown
- Notebook files use special file extensions (e.g., “.ipynb”, “.Rmd”)
- Not simple text files: need to be “rendered” with the notebook application: Jupyter Notebooks, RStudio Notebooks, Quarto.

# Jupyter


File Edit View Run Kernel Tabs Settings Help
Launcher
README.md
Lorenz.ipynb
Terminal 1
Console 1
Data.ipynb
Code
Python 3 (ipykernel)
Filter files by name
/ notebooks /
Name
Last Modified
audio a day ago
images a day ago
Cpp.ipynb a day ago
Data.ipynb a day ago
Fasta.ipynb a day ago
Julia.ipynb a day ago
Lorenz.ip... a day ago
lorenz.py a day ago
R.ipynb a day ago
The Lorenz Differential Equations
Before we start, we import some preliminary libraries. We will also import (below) the accompanying lorenz.py file, which contains the actual solver and plotting routine.
[1]:
%matplotlib inline
from ipywidgets import interactive, fixed
We explore the Lorenz system of differential equations:
$$\dot{x} = \sigma(y - x)$$
Output View
sigma 10.00
beta 2.67
rho 28.00
lorenz.py
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from scipy import integrate
def solve_lorenz(sigma=10.0, beta=8./3, rho=28.0):
"""Plot a solution to the Lorenz differential equations."""
max_time = 4.0
N = 30
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.axis('off')
Simple 1 3 Python
Ln 1, Col 1 Spaces: 4 lorenz.py

# Curation Questions

- Are you able to identify the **correct programming language** for the code files?
- Are source code files named appropriately, with a **file extension** that matches the programming language?
- Are you able to **open** the code files and **view** them using the correct programming language?

# Discussion: Deposit Example

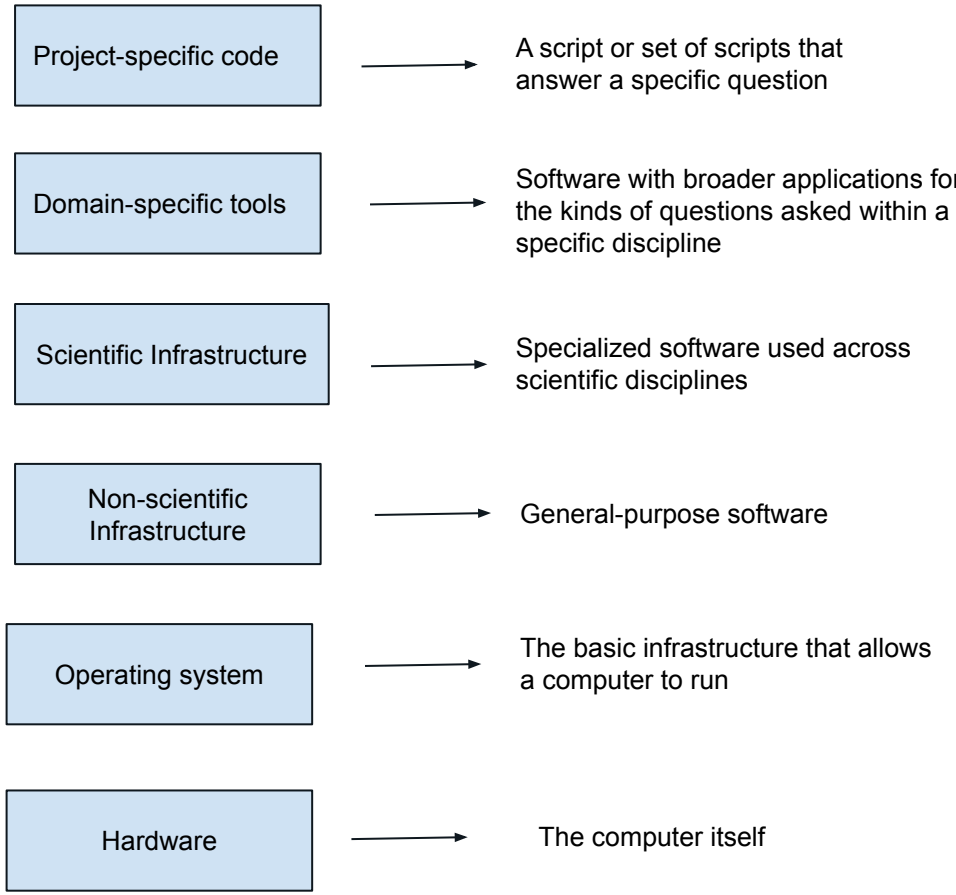
Let's revisit our deposit example (Code, data, and metadata document for the manuscript: **Density-dependence in wolf resource selection study designs**)

- File extension
  - Script uses a .txt extension in the deposit
  - While the code can be read in a text editor, it needs to be run using the programming language it was written in.
  - File extension should reflect the programming language
  - A better name for the script would be `code.R`

# 4. Dependencies

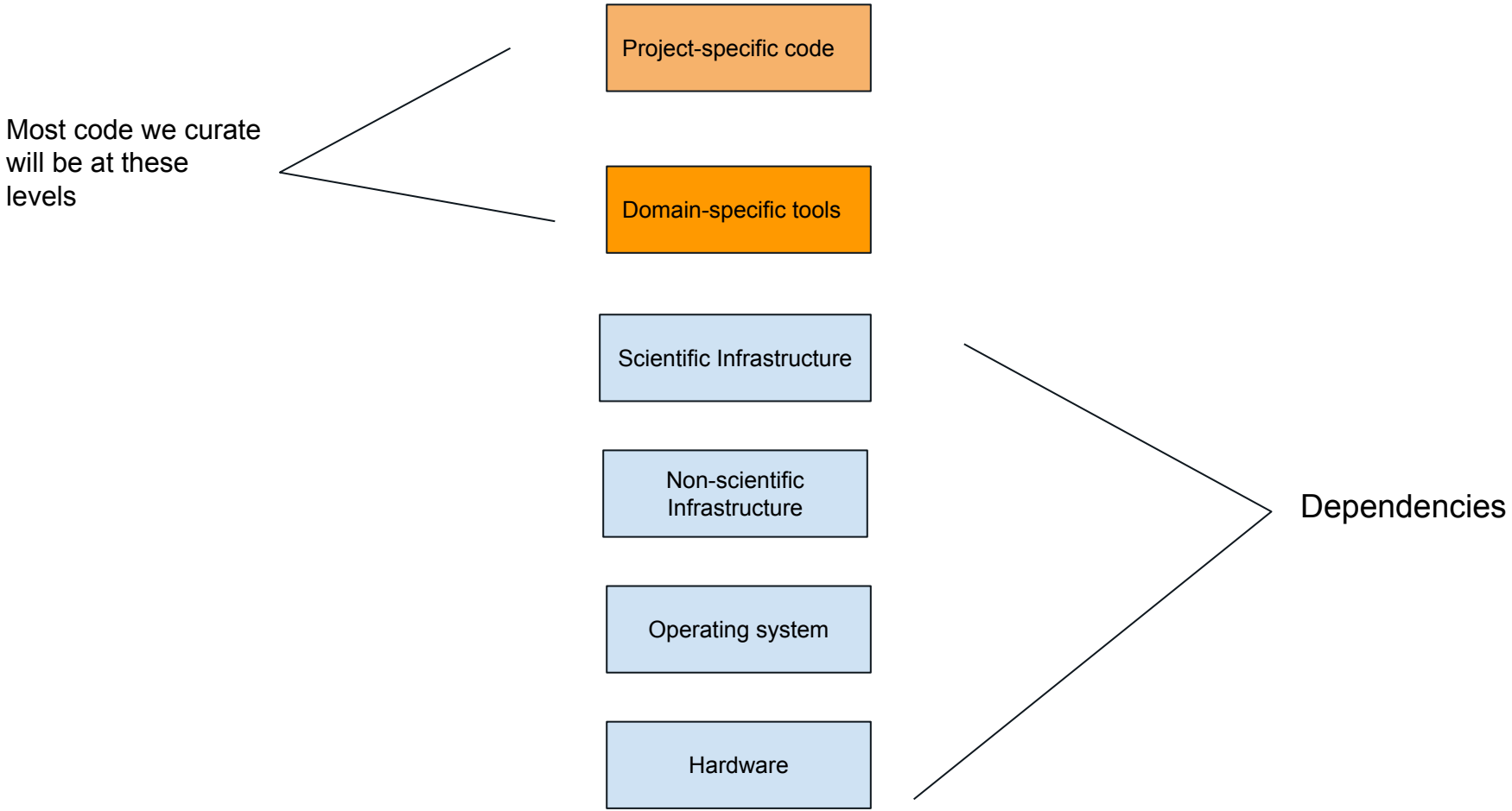
# Learning Outcomes

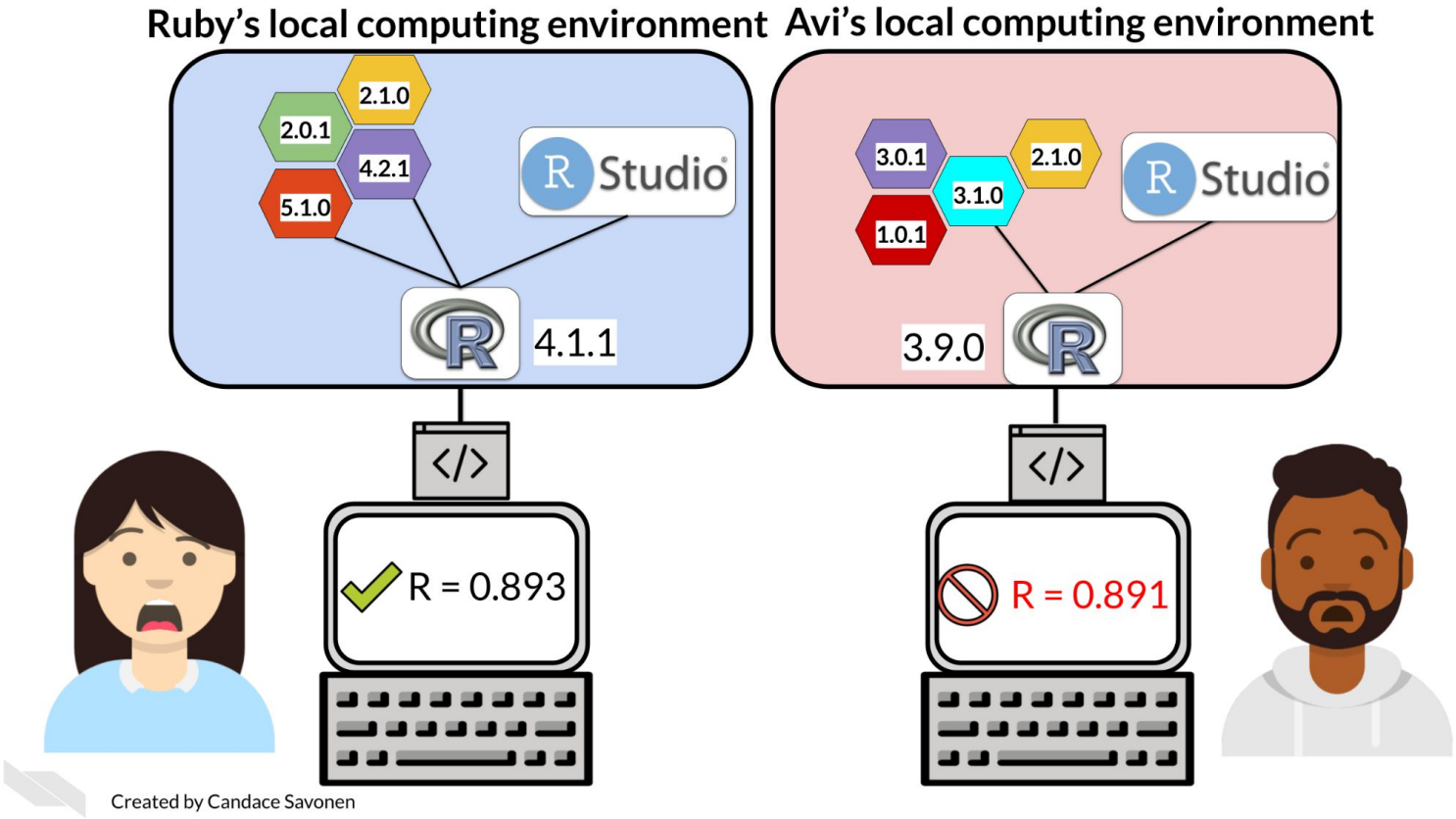
- Define “dependency” and learn why dependencies are used
- Identify when dependencies are being used in code
- Gain familiarity with package managers and versioning
- Learn several ways to record dependencies–via README, programmatically, and through the use of containerization




*Note: not all research software will necessarily use domain-specific tools*

Hinsen, Konrad. (2019). Dealing With Software Collapse. Computing in Science & Engineering. 21. 104-108. 10.1109/MCSE.2019.2900945.


Most code we curate will be at these levels
Project-specific code
Domain-specific tools
Scientific Infrastructure
Non-scientific Infrastructure
Operating system
Hardware
Dependencies


Ruby's local computing environment
2.1.0
2.0.1
4.2.1
5.1.0
RStudio
4.1.1
R = 0.893
Avi's local computing environment
3.0.1
2.1.0
3.1.0
1.0.1
RStudio
3.9.0
R = 0.891
Created by Candace Savonen

# Activity - Which Python?

- Open a Terminal on a Mac, or a command prompt on a Windows PC
- At the prompt, type

```
python --version
```

# Why use dependencies if they cause so many problems?



# Loading dependencies in code

## Dependencies in R

```
# Load libraries
  library(RMark)
  library(knitr)
  library(lme4)
  library(ggplot2)
  library(doBy)
```

## Dependencies in MATLAB

```
--------------------------------------------------------------------------------
MATLAB Version: 9.13.0.2080170 (R2022b) Update 1
MATLAB License Number: [redacted]
Operating System: macOS  Version: 12.6 Build: 21G115
Java Version: Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
--------------------------------------------------------------------------------
MATLAB                                                Version 9.13        (R2022b)
Control System Toolbox                                Version 10.12       (R2022b)
Deep Learning Toolbox                                 Version 14.5        (R2022b)
Signal Processing Toolbox                             Version 9.1         (R2022b)
Statistics and Machine Learning Toolbox               Version 12.4        (R2022b)
>> untitled
```

## Dependencies in Python

```
: from math import floor
  import pandas as pd
  import numpy as np
  import matplotlib.pyplot as plt
  %matplotlib inline
```

# Data and analysis code for "Capturing High Resolution Plant Movement in the Field"

Heuschele, Deborah; Furuta, Daniel; Smith, Kevin; Marchetto, Peter (2022)

**Persistent link to this item**
https://doi.org/10.13020/7e6x-8f36
https://hdl.handle.net/11299/226273

**Services**
Full Metadata (xml)
View Usage Statistics

**Keywords**
agronomy

**Collection period**
2019 to 2020

**Date completed**
2022

**Title**
Data and analysis code for "Capturing High Resolution Plant Movement in the Field"

**Published Date**
2022-02-07

**Authors**
Heuschele, Deborah
Furuta, Daniel
Smith, Kevin
Marchetto, Peter

**Group**
USDA Agricultural Research Service

**Author Contact**
Heuschele, Deborah (Jo.Heuschele@usda.gov)

**Type**
Dataset
Field Study Data
Programming Software Code

**Abstract**
This is the dataset and analysis code for the paper "Capturing High Resolution Plant Movement in the Field", in which we demonstrate a system to capture high resolution plant motion with small grains in natural conditions.

**Description**
Plant motion data, wind speed and direction data, and analysis code used for the paper.

**Funding information**
Sponsorship: MNDRIVE Robotics Sensing and Advanced Manufacturing initiative; Minnesota Department of Agriculture Grant No. 122130; University of Minnesota Rapid Agricultural Response Fund Grant No. AES00RR234

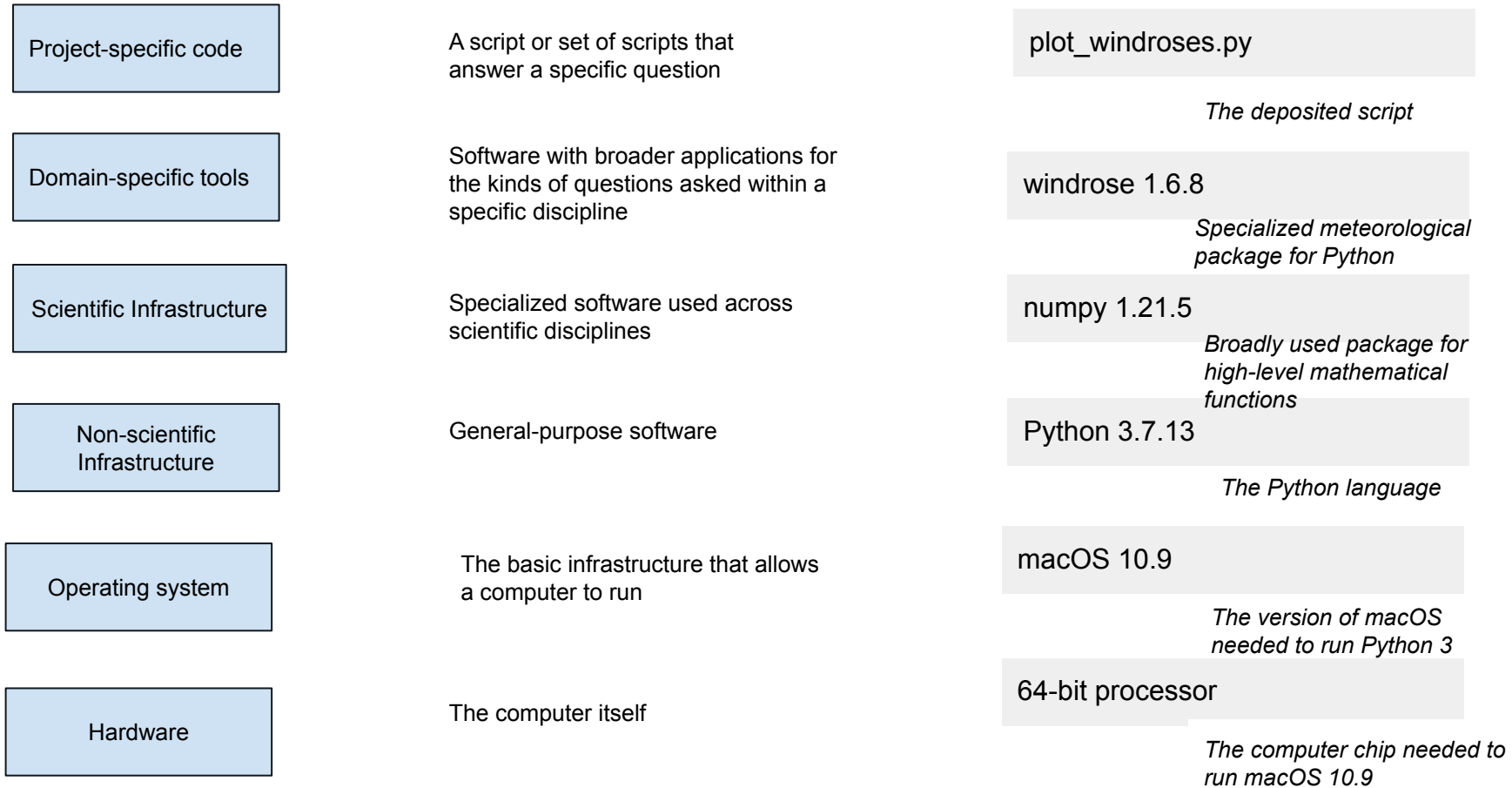


**Suggested Citation**

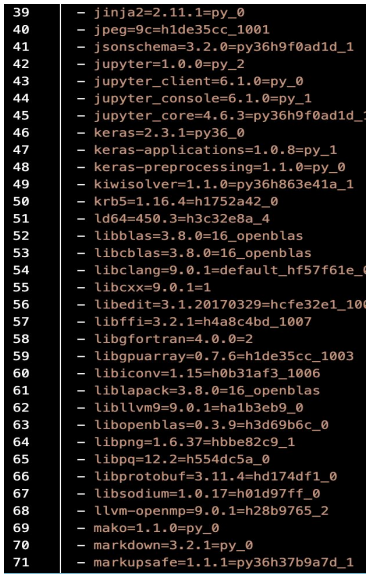
```
The scripts were developed with Python 3.7.13 and the following non-core package versions:
Python 3.7.13
numpy 1.21.5
dateutil 2.8.2
matplotlib 3.5.1
pandas 1.3.5
seaborn 0.11.2
windrose 1.6.8
```

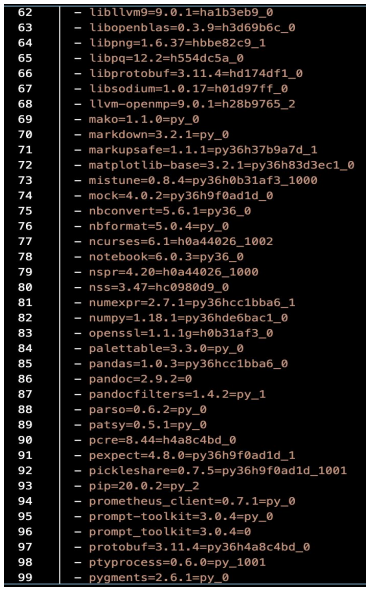
Users > talyacooper > Downloads > analysis_code > visualization > plot_windroses.py > ...

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jan  5 10:29:02 2022

This script creates windroses showing wind trends over the collected data.
To adapt to your needs, replace the file locations, date ranges,
and wind ranges to your situation.
"""

import windrose
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sn
import numpy as np

```


Project-specific code
A script or set of scripts that answer a specific question
plot_windroses.py
The deposited script
Domain-specific tools
Software with broader applications for the kinds of questions asked within a specific discipline
windrose 1.6.8
Specialized meteorological package for Python
Scientific Infrastructure
Specialized software used across scientific disciplines
numpy 1.21.5
Broadly used package for high-level mathematical functions
Non-scientific Infrastructure
General-purpose software
Python 3.7.13
The Python language
Operating system
The basic infrastructure that allows a computer to run
macOS 10.9
The version of macOS needed to run Python 3
Hardware
The computer itself
64-bit processor
The computer chip needed to run macOS 10.9

```
: from math import floor
  import pandas as pd
  import numpy as np
  import matplotlib.pyplot as plt
  %matplotlib inline
```


Swiger, B. M., Liemohn, M. W., & Ganushkina, N. Y. (2020). Improvement of Plasma Sheet Neural Network Accuracy With Inclusion of Physical Information. Frontiers in Astronomy and Space Sciences, 7. https://doi.org/10.3389/fspas.2020.00042

# Packages and package managers

**Package**
A collection of code, data, and documentation that can be distributed and re-used. Also referred to in some languages as a **library** or **module**.

**Package manager**
A program that does its best to keep track of the different software installed on a computer and their dependencies on one another.

If you download a dependency as a **package** from a **package manager,** the **package manager** should ensure that you also install that package's **direct AND indirect dependencies** (and those dependencies' dependencies)



The Comprehensive R Archive Network

# Documenting Dependencies - README

### Hints for the Installation

`librosa` uses `soundfile` and `audioread` to load audio files.

📝 Note that older releases of `soundfile` (prior to 0.11) do not support MP3, which will cause librosa to fall back on the `audioread` library.

### `soundfile`

If you're using `conda` to install librosa, then audio encoding dependencies will be handled automatically.

If you're using `pip` on a Linux environment, you may need to install `libsndfile` manually. Please refer to the SoundFile installation documentation for details.

### `audioread` and MP3 support

To fuel `audioread` with more audio-decoding power (e.g., for reading MP3 files), you may need to install either *ffmpeg* or *GStreamer*.

📝*Note that on some platforms,* `audioread` *needs at least one of the programs to work properly.*

If you are using Anaconda, install *ffmpeg* by calling

```
conda install -c conda-forge ffmpeg
```

https://librosa.org/

# When to document dependencies in a README

- It never hurts!
- Always specify version information
- Document API and data dependencies in the README
- Documenting dependencies in the README may be sufficient for a simple script that calls a small number of external packages or libraries
- README is the place to explain specific issues, challenges, or instructions

# Documenting Dependencies–Programmatic

## Python

main ▾ tile2net / requirements-dev.txt

Mary-h86 initial commit

1 contributor

32 lines (31 sloc) | 271 Bytes

```
1  centerline
2  jupyter
3  geopandas
4  geopy
5  matplotlib
6  momepy
7  numba
8  opencv-python
9  osmnx
10 pillow
11 pyarrow
12 rasterio
13 scikit-image
14 scikit-learn
15 shapely
16 tqdm
17 torch
18 torchvision
```

## R

### Set up package dependencies for compatibility with `renv`


Source: R/tar_renv.R

Write package dependencies to a script file (by default, named `_targets_packages.R` in the root project directory). Each package is written to a separate line as a standard `library()` call (e.g. `library(package)`) so `renv` can identify them automatically.

### Usage

```
tar_renv(
  extras = c("bs4Dash", "clustermq", "future", "gt", "markdown", "pingr", "rstudi
    "shiny", "shinybusy", "shinyWidgets", "visNetwork"),
  path = "_targets_packages.R",
  callr_function = callr::r,
  callr_arguments = targets::tar_callr_args_default(callr_function),
  envir = parent.frame(),
  script = targets::tar_config_get("script")
)
```

# When to create programmatic documentation of dependencies

- It never hurts!
- Particularly helpful for more complex code that calls a large number of external packages and libraries
- Required for certain kinds of virtual environments, like Binder, if a depositor will want their code to function in that setting

# Capturing dependencies: Containerizing

- Containerization tools like Docker and Singularity create a complete image of a piece of software's runtime environment–including all of the correct versions of the software's dependencies
- ReproZip is another tool designed for packaging software applications for reproducibility; it might be overkill for a simple script but is excellent for the preservation of more extensive projects that will not be updated.

# When to capture dependencies with a container

- This is often done when software is still in development, to ensure developers are making changes in the exact same environment with the same versions of dependencies
- Don’t forget to document how to use/run the container!
- If the software is no longer being developed, is complex and highly environment-specific, and you want to be able to run it in the future, preserving it in a ReproZip package is one of the best ways to ensure future reproducibility
    - Almost a “special collections” level of research software preservation

# Discussion: deposit example

- README indicates a now-obsolete version of R, which may make it harder to run the code.
- No version information for dependencies
  - Because the version of R is older, someone who wanted to reuse this code would need to do some research to find the correct version of the dependencies that would work with that version of R – could lead to some issues with indirect dependencies, too!

# Dependency issues to watch out for

What are some dependency-related issues you might look out for while curating?

What are some dependency-related issues we notice in our code example?

# Dependency issues to watch out for

- Dependencies are not described anywhere in **documentation**
- Dependencies are called in the code but are **not actually used** when the code is run
- No **version information** is provided about dependencies
- When the code is run, **dependencies are no longer available** or an error message pops up when it is installed
- Dependencies are **proprietary software** and cannot be accessed
- Code requires **specialized hardware or firmware** (eg. a GPU, an AR/VR device.)
- Code calls resources from elsewhere and requires an **API or an internet connection**
- Code contains **private information**, particularly an API key

# Code Curation Questions

- Are hardware and software **dependencies** adequately documented?
- Are **files referenced** in the source code included in the submission?

# 5. Reviewing Documentation

# Learning Outcomes

By the end of this module, participants will be able to:

- Identify types of research software documentation and how they can be useful during the curation process
- Make suggestions for research software documentation (README, Commenting, Markdown, etc.)
- Review and offer edits to README files/outlines for research software

# Reviewing Documentation as a Curator

Each step of the curation process involves reviewing provided documentation to help understand the structure and function of code deposit.

**Check** files and read documentation (risk mitigation, file inventory, appraisal/selection)

**Understand** the data (or try to), if not… (run files/environment, QA/QC issues, readme)

**Request** missing information or changes (tracking provenance of any changes and why)

**Augment** metadata for findability (DOIs, metadata standards, discoverability)

**Transform** file formats for reuse (data preservation, conversion tools, data viz)

**Evaluate** for FAIRness (licenses, responsibility standards, metrics for tracking use)

**Document** your curation activities (Curator Log, correspondence)

# What is documentation?

**Documentation is the use of text to describe the software's original environment, intended use, and outputs.**

*Documentation occurs throughout the research software lifecycle, so it is important to help researchers understand the value of documenting their processes and procedures before curation as this information is often referenced during the curation process.*

## Internal Documentation

*Comments and notes made within the code/software*

## External Documentation

*Comments and notes made outside of the code/software*

# External Documentation

# README Files

README files are used to describe code included in a code repository. The main goal of a README file is provide information on the following:

- Description of Project
- Contributors
- Technical Requirements (System and Language Dependencies & Working Directory)
- Any Known Bugs/Issues
- Output Files Created and Storage Location
- Naming Conventions
- License
- Acknowledgements
- Contact Information
- Any information significant to the structure, usability, and context of the code file(s)

**The goal is to facilitate reuse.**

# README Exemplar

*See file in workshop materials*

# Whole Group Activity

# README File Investigation

As a group, we will review the README file below and consider –

- The usefulness of the file
- Potential edits to the file

---

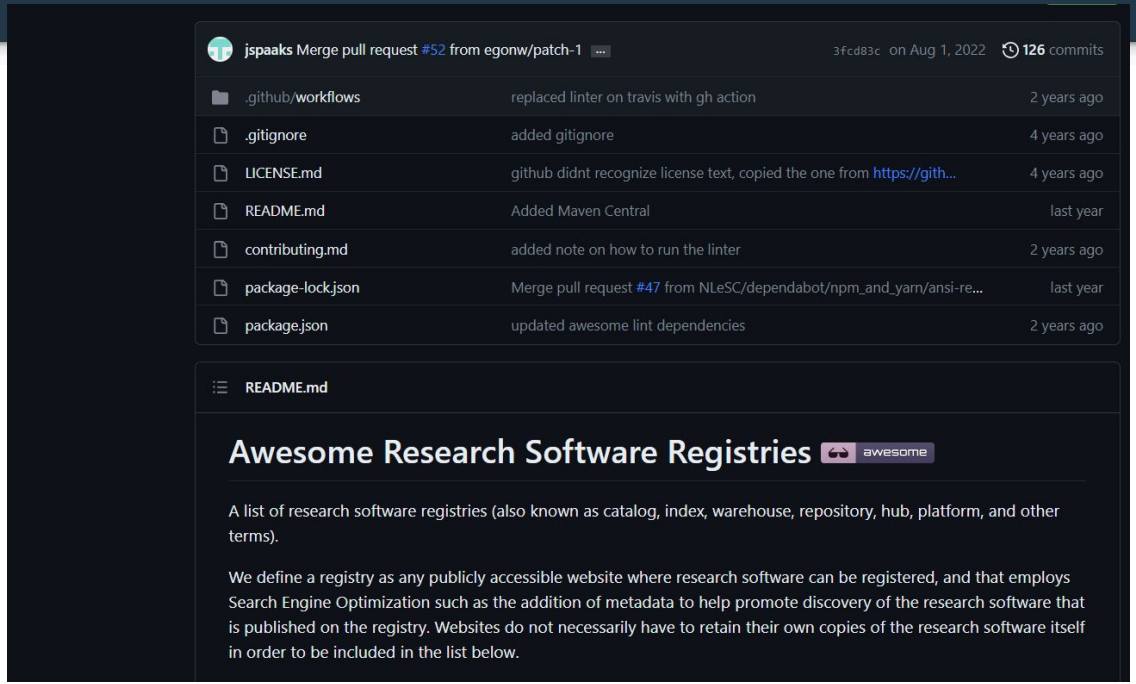
**Attitude** is a Python module for fitting the orientation of planes! Its core mission is to compute planar fits and transform them (*along with meaningful error distributions*) into a form that can be used for geology. It is designed to support the collection of structural measurements from remote sensing data.

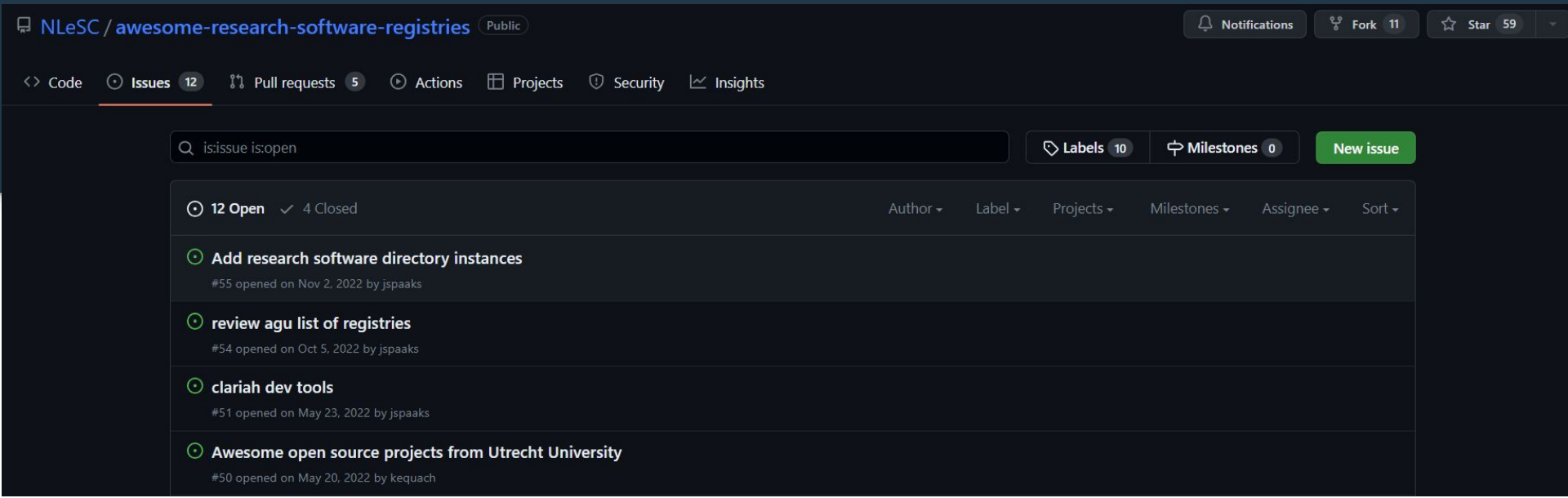
The documentation for the module can be found at https://davenquinn.github.io/Attitude, and a preprint publication describing the underlying statistical method is on [EarthArXiV](https://eartharxiv.org/4enzu/).

# Reviewing GitHub Documentation

# READMEs on GitHUB

GitHub uses the README file as the front page of a code repository.

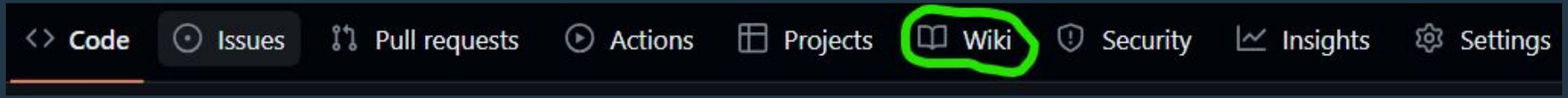





The researcher may use the ISSUES menu to add further documentation about a software.

Issues can be created and addressed by all members in a repository.

# GitHub Wiki




Researchers may use GitHub wikis to provide software documentation

## About wikis

You can host documentation for your repository in a wiki, so that others can use and contribute to your project.

Wikis are available in public repositories with GitHub Free and GitHub Free for organizations, and in public and private repositories with GitHub Pro, GitHub Team, GitHub Enterprise Cloud and GitHub Enterprise Server. For more information, see "GitHub's plans."

Every repository on GitHub.com comes equipped with a section for hosting documentation, called a wiki. You can use your repository's wiki to share long-form content about your project, such as how to use it, how you designed it, or its core principles. A README file quickly tells what your project can do, while you can use a wiki to provide additional documentation. For more information, see "About READMEs."

# External Documentation: File Names

File names can be consistently arranged to ensure discoverability and reusability of research software. File naming conventions should be documented in the README file (more to come!).. Potential metadata include:

- Date (following the ISO 8601 standards: YYYYMMDD or YYYY-MM-DD)
- Time
- Sample/Subject/Image ID Numbers
- File Description
- Versioning Information (mostly for files that do not undergo automatic version control)
- Output
- Data type
- Project/Project Component Title
- Researcher/Assistant Name

**Abbreviations** will come in handy as you want your file names to be as short as possible.

Avoid using spaces; instead, **use underscores(_) or hyphens(-).**

**Use one case when possible** to help in retrieval

# Sample File Names

- **YYYY-MM-DD_ProjectTile_SubjectID_Version.FileType**
- *2012-12-20_birdmiganalysis_sue_v1.csv*

- **YYYYMMDD_ProjectTitle.FileType**
- *20221015bookclustering_analysis.py*
  - *Versioning information for code files is typically included in version control protocols.*

If you notice that a deposit has a standard file naming structure, suggest that this be structure be outlined in the README file.

# File Names

## WARNING!

Be sure to remind researchers to avoid absolute paths! Relative paths are preferred.

Absolute Path: "C:\\Users\\rholmes\\Documents.csv")
Relative Path:(“..\Documents.csv")

# Internal Documentation Strategies

# Internal Documentation Strategies: Commenting

Comments are natural language text strings that describe software code. **Commenting is useful for annotating software code as they help readers understand the complexities of the software's language.** Comments are usually made when–

- An action will occur that results in a new or a modified variable, including adding columns to data frames or other similar edits
- An action will occur that results in an output, both saved to your system or printed in the console
- User input will be required
- Conditions needs must be met before running a line/block of code
- An output is to be expected
- An explanation of a process needs to be provided (why was a certain block of code run?)
- Users need to change alter code for their purposes

# Identifying Comments in Python

Comments in Python are created begin with a hashtag (#).

```
In [1]:
import pandas as pd

In [2]:
#import data file
df = pd.read_excel("C:/Users/MayeKaypounyers/Box/Librarianship/faculty_research_interests.xlsx")
```

Additionally, you can make block comments by surrounding text with triple double quotes (“””).

```
In [ ]:
"""
Creation Date: 10/15/2022
Modification Date: 6/12/2023

Depednencies and Requirements
- Python 3.7 or higher
- Pandas 1.0.3 or higher
- IDE: Jupyter

Other required files:
- faculty_research_interests.xlsx

Summary: This code was created to restructure faculty research
interest data.

"""
```

# Identifying Comments in R

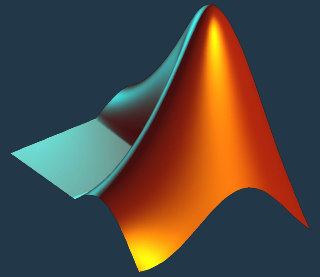
To create comments in R, begin the comment text with a hashtag (#).

```r
# Function to calculate the circumference of a circle
circumference <- function(radius) {
  2 * pi * radius
}

# Function to calculate the area of a circle
area <- function(radius) {
  pi * radius^2
}
```

Additionally, you can make block comments by surrounding text with triple double quotes (“””).

```
"""
Author: Jayden Eaves
Contact: jeaves@ggolden.com
Affiliation: Gee's Golden Code Academy

Creation Date: 6/7/2023
License Information: None
Dependencies: R (version 2.3.1 or higher)

Other Required Files:
- data.csv (input file)
- output.png (output visualization)

Summary: This code calculates the dimensions of a circle based on its circumference
"""
```

# Identifying Comments in MATLAB



To create comments in MATLAB, begin the comment text with a percent symbol (%).

```
% Calculate the circumference
circumference = 2 * pi * radius;

% Calculate the area
area = pi * radius^2;
```

Additionally, you can make block comments by surrounding text in the %{ %} construction.

```
%{
this code
will generate a new set
of dimensions for your research
%}
```

# Internal Documentation: Comments

```python
# Function to calculate the factorial of a number
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# input
num = int(input("Enter a number: "))

# Check the input
if num < 0:
    print("Factorial is undefined for negative numbers.")
elif num == 0:
    print("The factorial of 0 is 1.")
else:
    result = factorial(num)
    print(f"The factorial of {num} is {result}.")
```

```python
# Function to calculate the factorial of a number
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Get input from user
num = int(input("Enter a number: "))

# Check whether the input is non-negative
if num < 0:
    print("Factorial is undefined for negative numbers.")
elif num == 0:
    print("The factorial of 0 is 1.")
else:
    result = factorial(num)
    print(f"The factorial of {num} is {result}.")
```

# Internal Documentation: Comments

```r
# Create a list of numbers
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)


even_sum <- 0


for (number in numbers) {
  # Check the number
  if (number %% 2 == 0) {
    # Check eveness
    even_sum <- even_sum + number
  }
}

cat("The sum of even numbers in the list is:", even_sum, "\n")
```

# Internal Documentation: Comments

```r
# Create a list of numbers
numbers <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# Initialize a variable to store the sum of even numbers
even_sum <- 0

# Iterate through the list of numbers
for (number in numbers) {
  # Check if the number is even
  if (number %% 2 == 0) {
    # If it's even, add it to the sum
    even_sum <- even_sum + number
  }
}

# Print the sum of even numbers
cat("The sum of even numbers in the list is:", even_sum, "\n")
```

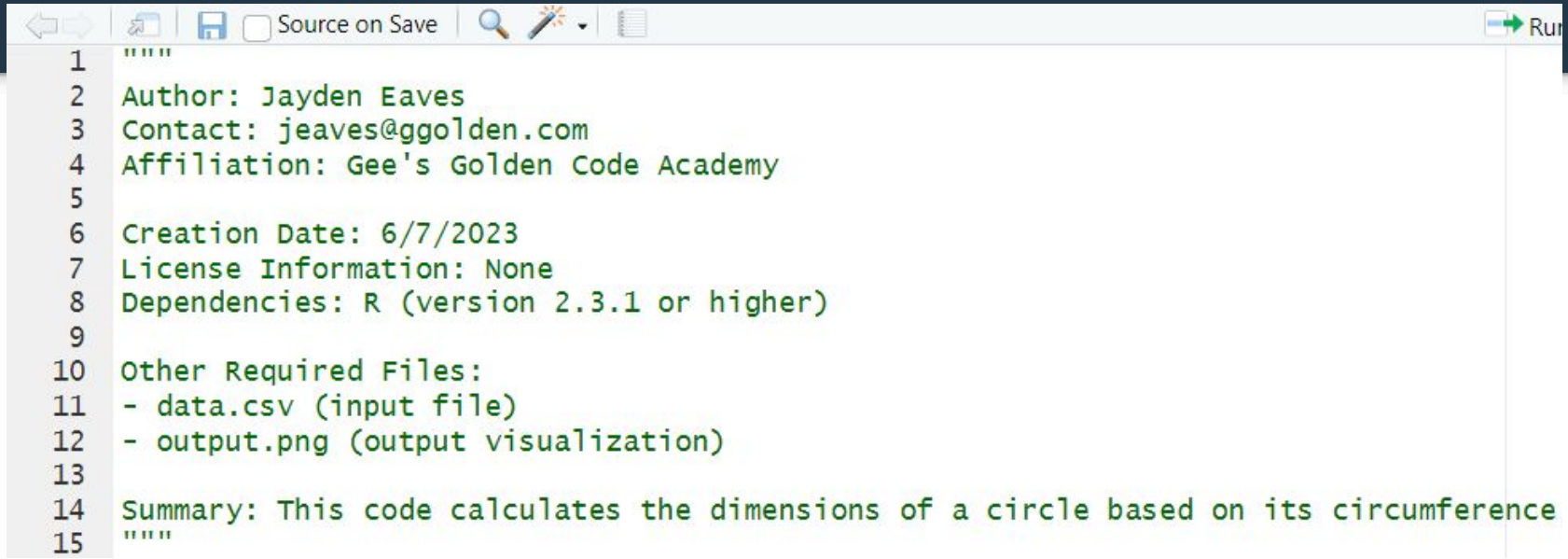
# Internal Documentation: Header Text

When reviewing a code deposit, the header can provide useful information about the content of a code file. This text usually includes –

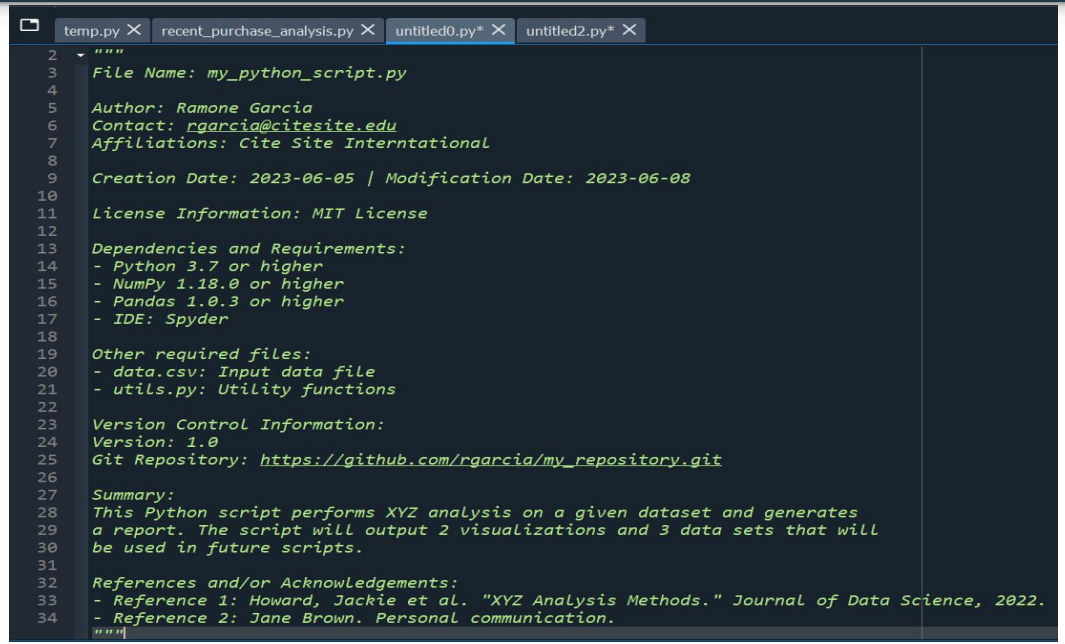
- File Name
- Author Information (name, contact, affiliations)
- Creation Date and Modification Dates
- License Information
- Dependencies and Requirements
- Other Required Files
- Version Control Information
- Summary
- References and/or Acknowledges

**Header Texts are often condensed/individualized versions of the README file.**

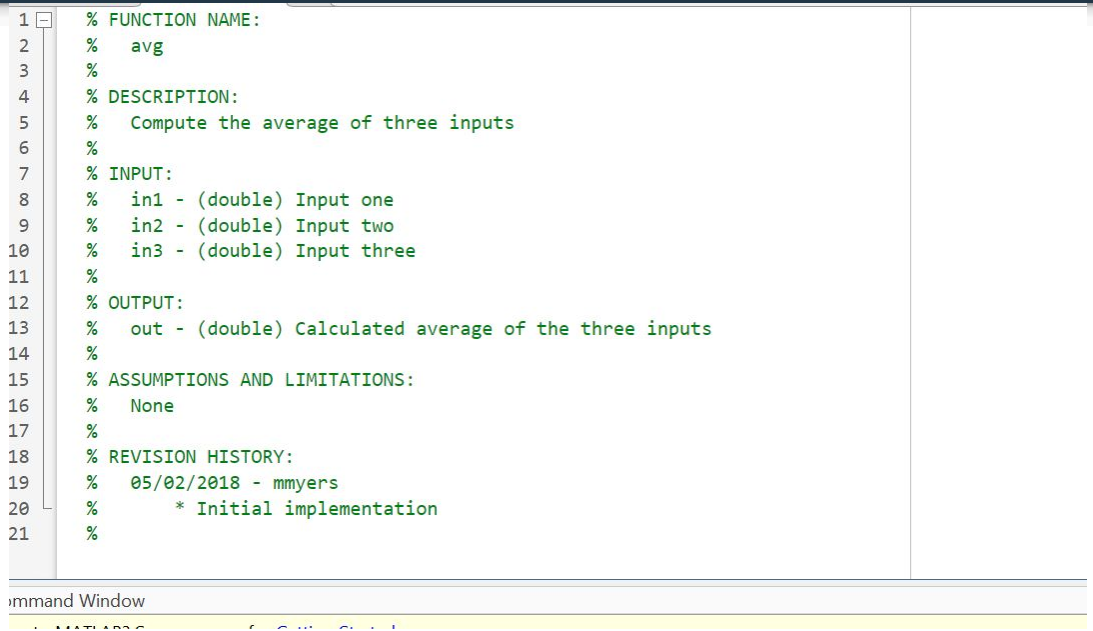
# R Header Text Example



# Python Header Text Example



# MATLAB Header Text Example



```
% FUNCTION NAME:
%   avg
%
% DESCRIPTION:
%   Compute the average of three inputs
%
% INPUT:
%   in1 - (double) Input one
%   in2 - (double) Input two
%   in3 - (double) Input three
%
% OUTPUT:
%   out - (double) Calculated average of the three inputs
%
% ASSUMPTIONS AND LIMITATIONS:
%   None
%
% REVISION HISTORY:
%   05/02/2018 - mmyers
%       * Initial implementation
%
```

# Best Practice Resources

Google Style Guides
PEP8 (Python)
MatLab Style Guide
ISO 8601

# Small Group Activity

# Comment Edits

The following code snippet creates a new column based on values in another column. What internal documentation edits might you suggest for the author?

```r
# Load the required libraries
library(dplyr)

# Read the data file
data <- read.csv("input_data.csv")

# Create a new column
data$new_column <- ifelse(data$old_column > 10, "Greater than 10", "Less than or equal to 10")

# Remove a column
data <- data[, -ncol(data)]

# Display the head of the data frame
head(data)
write.csv(data, "output_data.csv", row.names = FALSE)
```

# Potential Curation Actions

| | |
|---|---|
| **Create meaningful header text.** | Ask depositor, “Do you anticipate any of these files being separated?” |
| **Ensure interoperable metadata** | If metadata standards are used, link to documentation |
| **Promote related literature** | Ensure that literature is referenced in documentation (README) |
| **Ensure human-approachable code** | Provide depositors with access to best practices resources like PEP8 |
| **Ensure system interoperability** | Suggest system and directory information be included in documentation (README) |

# Discussion: Deposit Example

Visit the code example included in the workshop materials (*Code, data, and metadata document for the manuscript: Density-dependence in wolf resource selection study designs*) and consider the following:

1. How can the README file be edited to give a clear explanation of what the code does and its expected output?

2. What are the benefits and disadvantages of installing libraries at the end of the code?

3. What edit(s) can be made to avoid including the researcher's working directory?

# Questions, Concerns, Comments

# Thanks for completing this module with us!

# Final Exercise: Following a Checklist

- Are **file paths in the source code portable**?
- Are source code files named appropriately, with a **file extension** that matches the programming language?
- Are hardware and software **dependencies** adequately documented?
- Are **files referenced** in the source code included in the submission?
- Are important functions and components **documented with comments** in the source code?
- Does the code include **README**?
- Does the code include a **license** and is it appropriate?
- Are files in the submission **organized in a consistent and logical manner**?

# Additional Slides

The following slides were not covered during the pilot workshop. They are included for reference

- Draft Software Licenses Section
- Draft Project Structure Section
- Markdown/Markup

# Software Licenses

Note: This section was not covered in the pilot workshop due to time constraints. Slides are included for reference and to facilitate future work.

# What is a software license?

- *A license is legal permission for others to copy, modify, and share your work*

# What is a software license?

- The Open Source Definition: https://opensource.org/osd
  - Allows free distribution
  - Applies to software source code
  - Allows derived works
  - (...)
- Common Open Source Licenses in Science:
  - BSD (Astropy, NumPy, Jupyter Notebook, scikit-learn)
  - MIT (Julia, GPT-2, GitLab)
  - Apache (TensorFlow, Natural Language Toolkit)
  - GPL (R, FFTW, SageMath, Linux)

# Copyleft vs Permissive Licenses

- Copyleft licenses: copyright licenses that include “share alike” restrictions
  - GPL: derived works must be distributed under the GPL
  - “Lesser” GPL:  provides an exception for linked libraries
  - Affero GPL: covers distribution via web-based applications

- Permissive licenses place few restrictions on use
  - BSD (3-clause): derived works must include original copyright and license text
  - MIT: explicitly permits sublicensing

tl;dr Legal (https://tldrlegal.com) has useful, short summaries of these licenses

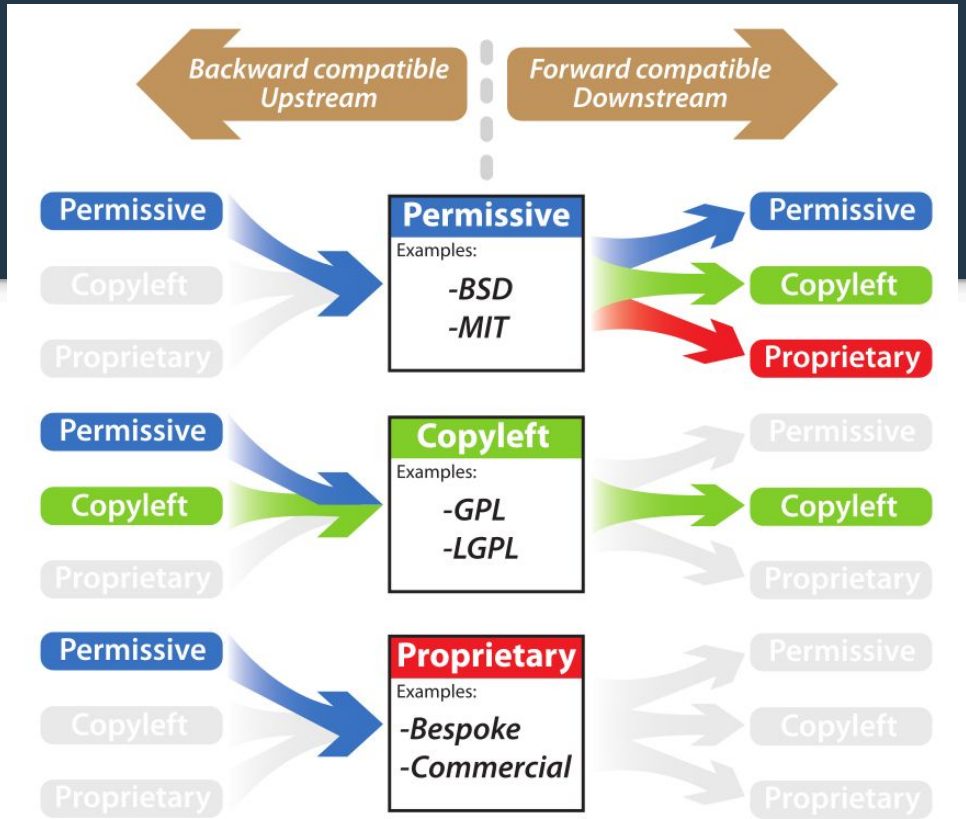
# Example: BSD 3-Clause

Copyright <YEAR> <COPYRIGHT HOLDER>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met.
1. **Redistributions of source code** must retain the above copyright notice, this list of conditions and the following disclaimer.
2. **Redistributions in binary form** must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to **endorse or promote products** derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR…

# License compatibility



# Project Structure

Note: This section was not covered in the pilot workshop due to time constraints. Slides are included for reference and to facilitate future work.

# Objectives

- Suggest some principles to follow for organizing projects
- Provide guiding questions to ask yourself and a depositor about project structure as you curate

# Three rules of project organization

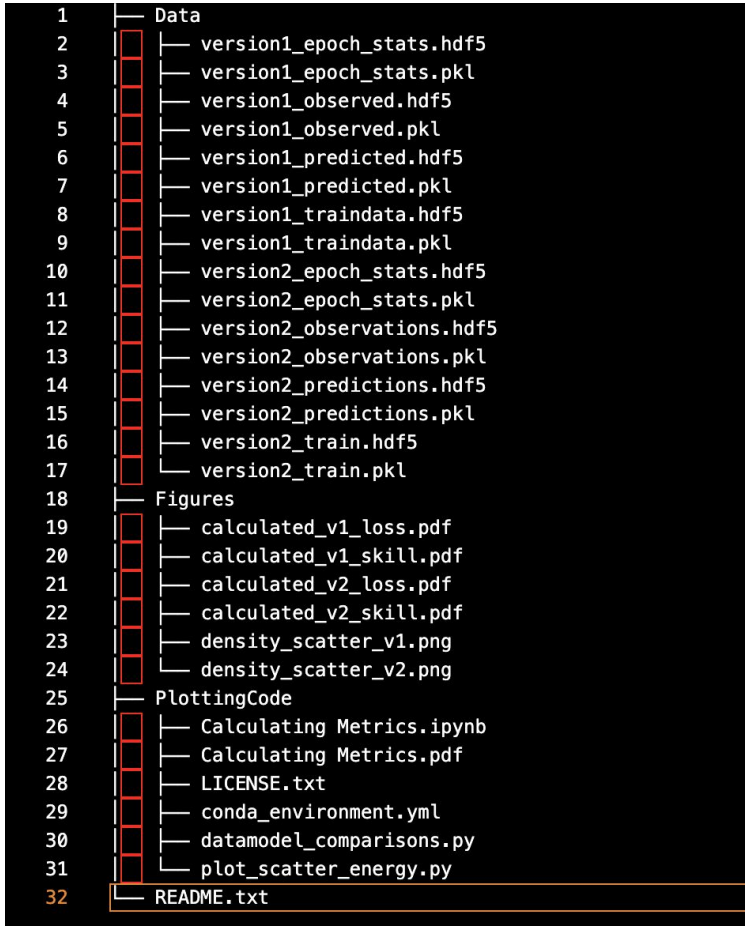
- Separate directories by function.
- Separate files into inputs and outputs.
- Make directories portable

Gentzkow M, Shapiro JM. Code and Data for the Social Sciences: A Practitioner's Guide; 2014. Available from: http://web.stanford.edu/~gentzkow/research/CodeAndData.pdf.

Data = “input”

Figures = “output”

Plotting code = scripts

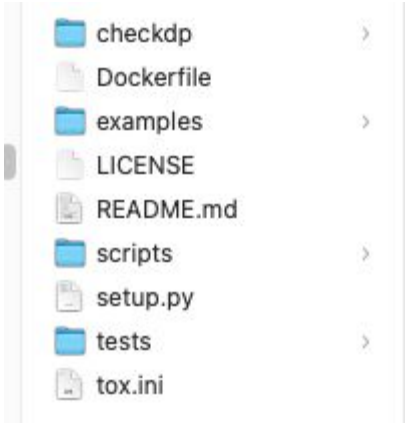



Swiger, B. M., Liemohn, M. W., & Ganushkina, N. Y. (2020). Data From Improvement of Plasma Sheet Neural Network Accuracy With Inclusion of Physical Information. DOI: 10.7302/559r-t639

# Other kinds of project organization

Common file organization for academic papers (example)


Common organizational structure for a software package (from CheckDP)




Wang, Y., Z. Ding, D. Kifer, and Danfeng Zhang. 2020. CheckDP: An Automated and Integrated Approach for Proving Differential Privacy or Finding Precise Counterexamples. DOI: https://doi.org/10.1145/3372297.3417282

# Key questions

- Does the project structure make sense for the type of project you are curating?
- Does the project structure conform to similar types of projects (code and data/software libraries/etc.)?
- Can you clearly describe how the files are organized/why they are organized in a particular way?
- Is the project structure described in the README?

# Be CAREFUL when you reorganize a project!

- Altering the structure of a project will cause it to BREAK if you do not make corresponding changes to the code itself!

```
# Load predictions and observations of test data.

# Start with Version 1.
observed_fname = \
    '../Data/version1_observed.pkl'

predicted_fname = \
    '../Data/version1_predicted.pkl'
```

What would happen if I decided to make a subdirectory in “Data” called “version1” for all the version1 data?

# Code Curation Questions

- Are files in the submission **organized in a consistent and logical manner**?

# Markdown/Markup

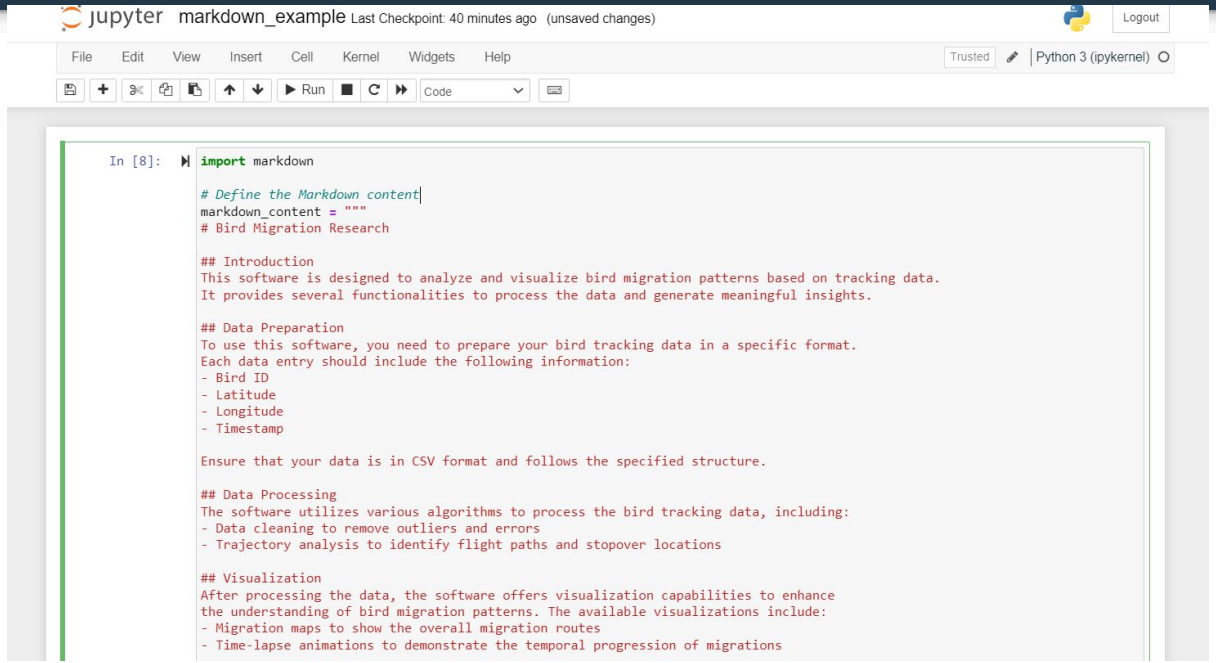
Note: This section was not covered in the pilot workshop due to time constraints. Slides are included for reference and to facilitate future work.

# Internal Documentation Strategies: Markdown/Markup

Markdown is a language used to add visual features of text. The markdown process varies depending on the development environment. In Python and R, you can use packages to help create markdown text. However, you can also use certain development environments to render markdown specifications. MATLAB is similar as it natively supports markdown.

# Markdown  Library in Python

Install and import the markdown library to render text. Content can be stored in a variable.



# Markdown Library in Python

Once text is stored, use the markdown library to output an HTML rendering.

```
## Getting Started
Follow the steps below to get started with the software:
1. Install the required Python packages: `pip install pandas matplotlib`
2. Prepare your bird tracking data in CSV format.
3. Configure the software with the data file path and desired parameters.

## Conclusion
This software provides a comprehensive framework for bird migration research.
It simplifies the process of data analysis and visualization, enabling researchers to gain valuable insights.

For detailed documentation and code examples,
refer to the project's GitHub repository: [Bird Migration Research Software](https://github.com/your/repository)

---
"""

# Convert Markdown to HTML
html_content = markdown.markdown(markdown_content)

# Save the HTML content to a file
with open("bird_migration_research.html", "w") as f:
    f.write(html_content)

print("Markdown content converted and saved as HTML.")
```

```
Markdown content converted and saved as HTML.
```

# Markdown Library in Python

## Bird Migration Research

### Introduction

This software is designed to analyze and visualize bird migration patterns based on tracking data. It provides several functionalities to process the data and generate meaningful insights.

### Data Preparation

To use this software, you need to prepare your bird tracking data in a specific format. Each data entry should include the following information: - Bird ID - Latitude - Longitude - Timestamp

Ensure that your data is in CSV format and follows the specified structure.

### Data Processing

The software utilizes various algorithms to process the bird tracking data, including: - Data cleaning to remove outliers and errors - Trajectory analysis to identify flight paths and stopover locations - Statistical analysis to extract migration patterns and trends

### Visualization

After processing the data, the software offers visualization capabilities to enhance the understanding of bird migration patterns. The available visualizations include: - Migration maps to show the overall migration routes - Time-lapse animations to demonstrate the temporal progression of migrations - Statistical charts to present migration trends and patterns

### Getting Started

Follow the steps below to get started with the software: 1. Install the required Python packages: `pip install pandas matplotlib` 2. Prepare your bird tracking data in CSV format. 3. Configure the software with the data file path and desired parameters. 4. Run the software using the provided Python script.

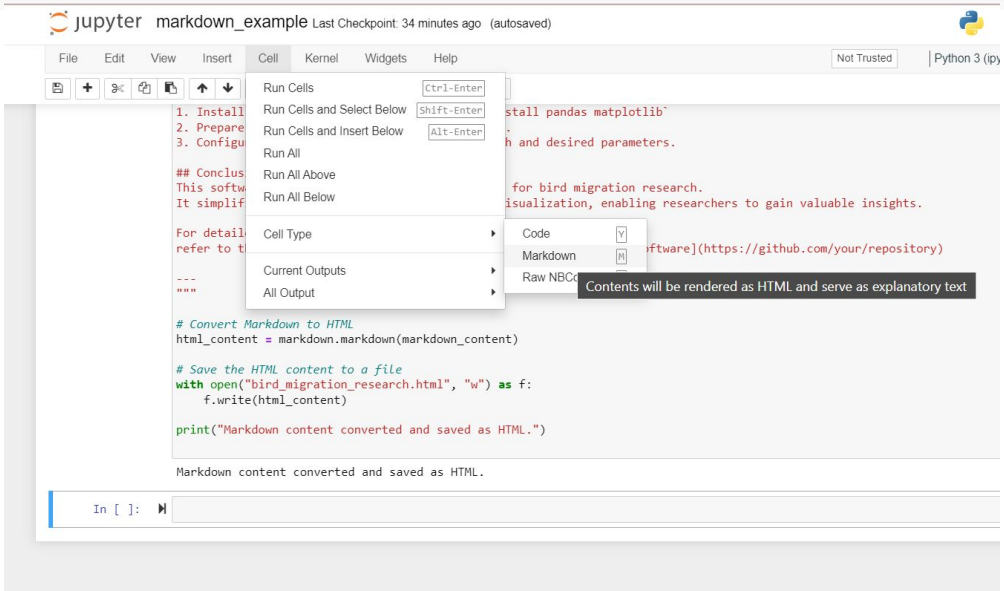
### Conclusion

This software provides a comprehensive framework for bird migration research. It simplifies the process of data analysis and visualization, enabling researchers to gain valuable insights into bird behavior and migration patterns.

For detailed documentation and code examples, refer to the project's GitHub repository: Bird Migration Research Software

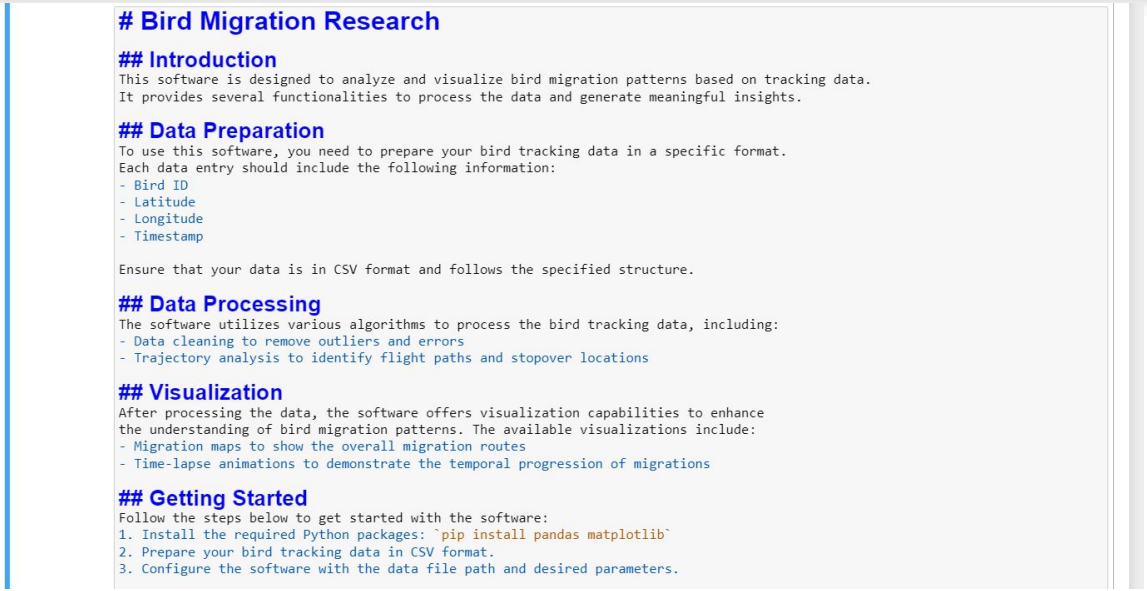
# Markdown Cells in Python – Jupyter Notebook

Some Integrated Development Environments (IDEs) allow you to create markdown cells within your code.



# Markdown Cells in Python – Jupyter Notebook

Use special characters to establish headings and other visual features.



```
# Bird Migration Research

## Introduction
This software is designed to analyze and visualize bird migration patterns based on tracking data.
It provides several functionalities to process the data and generate meaningful insights.

## Data Preparation
To use this software, you need to prepare your bird tracking data in a specific format.
Each data entry should include the following information:
- Bird ID
- Latitude
- Longitude
- Timestamp

Ensure that your data is in CSV format and follows the specified structure.

## Data Processing
The software utilizes various algorithms to process the bird tracking data, including:
- Data cleaning to remove outliers and errors
- Trajectory analysis to identify flight paths and stopover locations

## Visualization
After processing the data, the software offers visualization capabilities to enhance
the understanding of bird migration patterns. The available visualizations include:
- Migration maps to show the overall migration routes
- Time-lapse animations to demonstrate the temporal progression of migrations

## Getting Started
Follow the steps below to get started with the software:
1. Install the required Python packages: `pip install pandas matplotlib`
2. Prepare your bird tracking data in CSV format.
3. Configure the software with the data file path and desired parameters.
```

# Markdown Cells in Python – Jupyter Notebook

## Bird Migration Research ¶

### Introduction

This software is designed to analyze and visualize bird migration patterns based on tracking data. It provides several functionalities to process the data and generate meaningful insights.

### Data Preparation

To use this software, you need to prepare your bird tracking data in a specific format. Each data entry should include the following information:

- Bird ID
- Latitude
- Longitude
- Timestamp

Ensure that your data is in CSV format and follows the specified structure.

### Data Processing

The software utilizes various algorithms to process the bird tracking data, including:

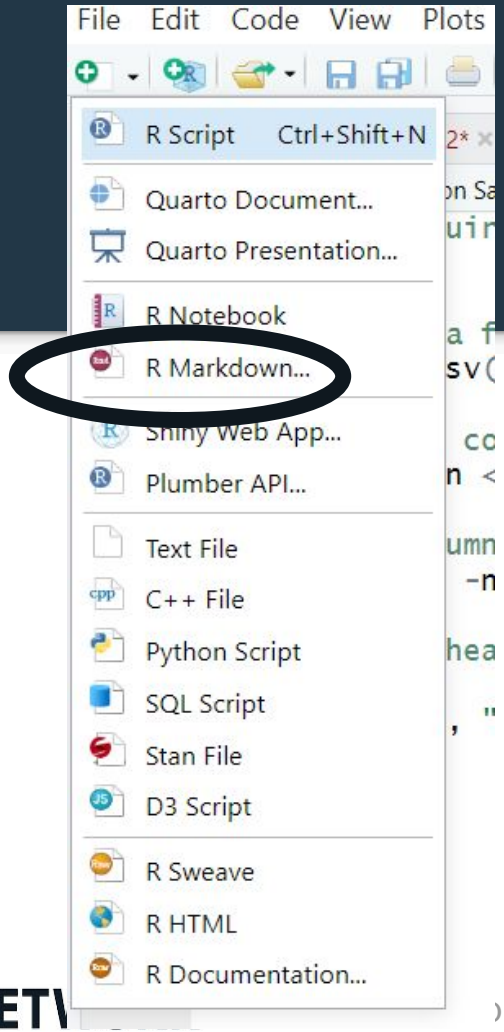
- Data cleaning to remove outliers and errors
- Trajectory analysis to identify flight paths and stopover locations

### Visualization

After processing the data, the software offers visualization capabilities to enhance the understanding of bird migration patterns. The available visualizations include:

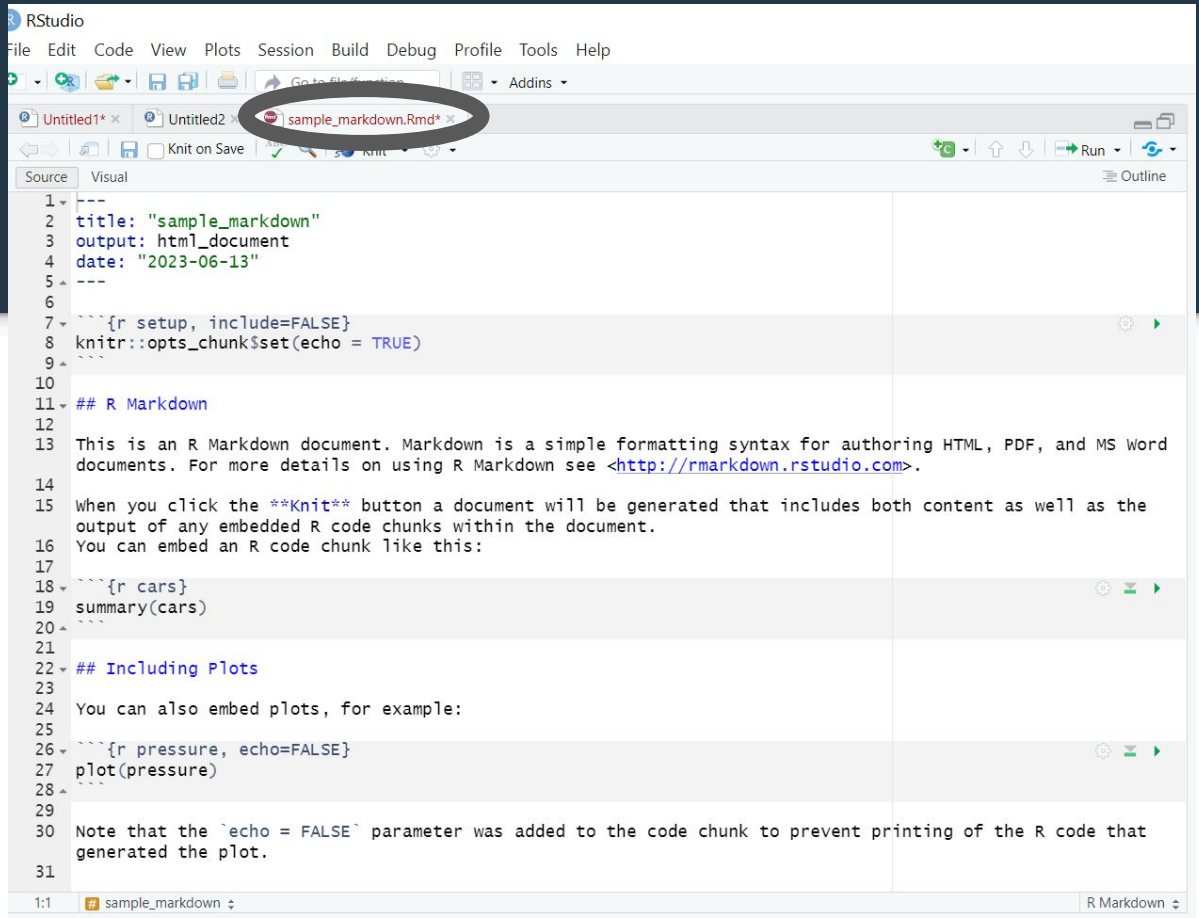
# Markdown in R

R Studio allows you to create markup files that can include snippets of R code. (.rmd)



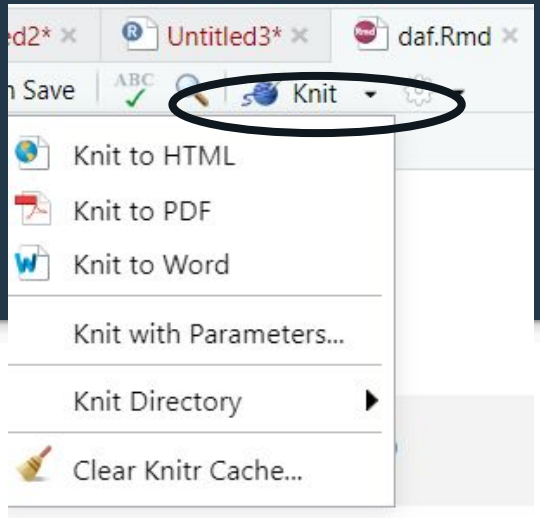
# Markdown in R

R Studio allows you to create markup files that can include snippets of R code. (.rmd)



# Markup in R

Select Knit to render your file in HTML or some other format.




## sample_markdown

2023-06-13

### R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

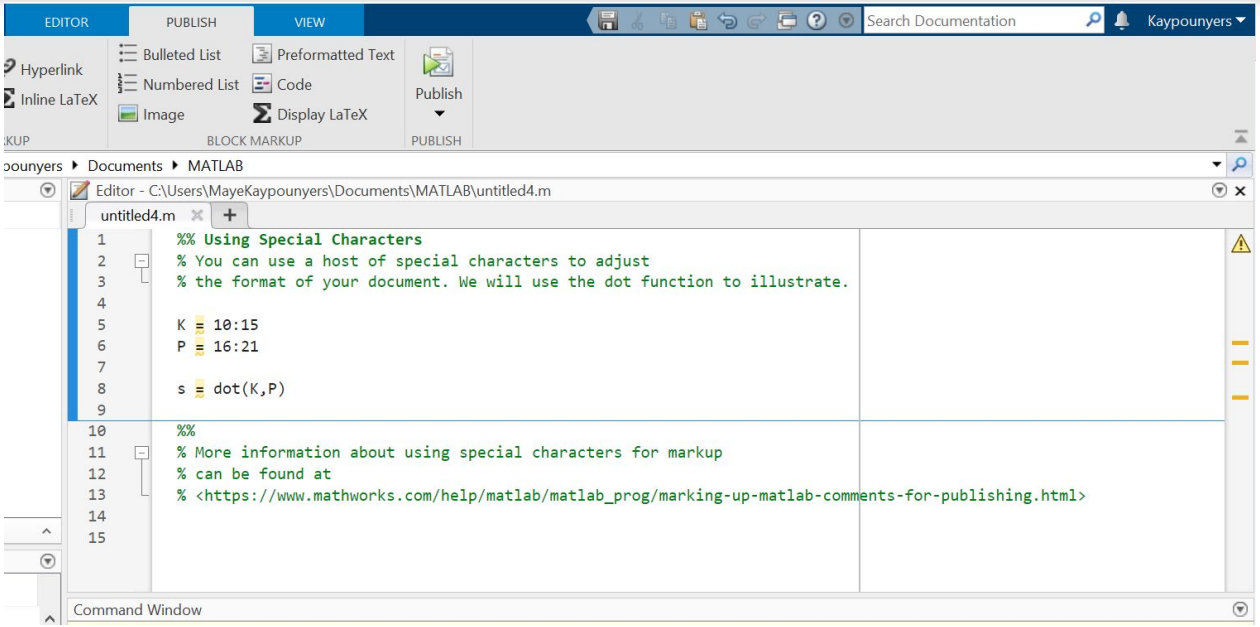
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

# Markdown in MATLAB

MATLAB natively supports markdown, so be sure to consult the use of markdown symbols for the system (ex. Levels of % symbol)



# Markdown in MATLAB

After creating your file, select the publish button to render the text in HTML.

## Using Special Characters

You can use a host of special characters to adjust the format of your document. We will use the dot function to illustrate.

```
K = 10:15
P = 16:21

s = dot(K,P)
```