NIDDK-CR Data Science and Meet the Expert Webinar Series

National Institute of Diabetes and Digestive and Kidney Diseases

Central Repository

National Institute of Diabetes and Digestive and Kidney Diseases

Central Repository

About the Series

- Aims to accelerate data science and AI-driven biomedical research by fostering collaboration between biomedical researchers and experts in the field
- Monthly webinar held on the **last Thursday of each month**

Upcoming Webinars

- Spatial Transcriptomics Computational Methods
- Use of AI in Metadata Harmonization
- Tools for whole-slide imaging pathomics
- Fusion Tool Demo

Learn more about the webinar series, register for future webinars, and access past webinars materials and recordings

National Institute of Diabetes and Digestive and Kidney Diseases

Central Repository

The Analytics Workbench V1.0

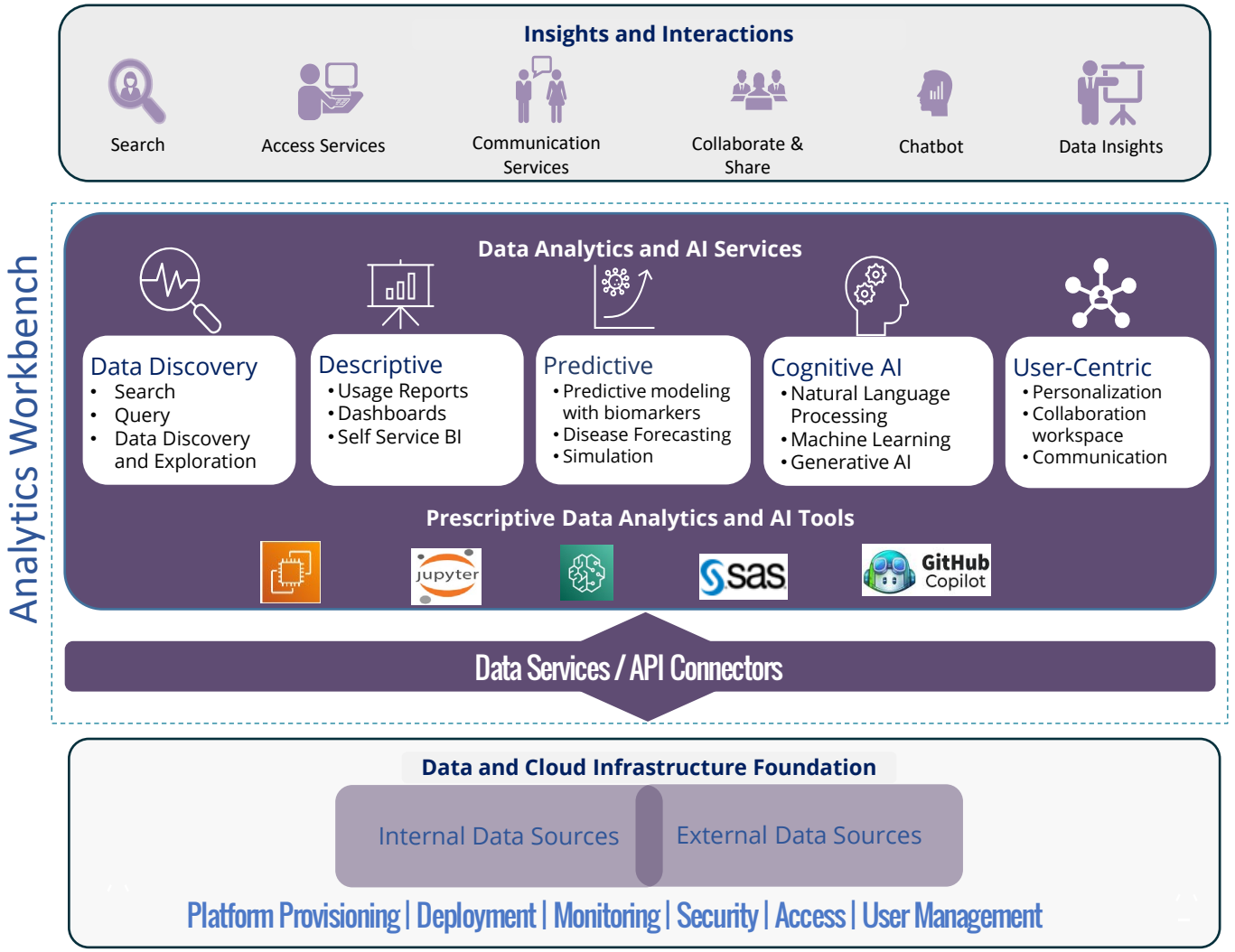
Streamlining *end-to-end data science lifecycle and discovery of data-driven biomedical insights.*

Innovation and ease of use

A cloud-based analytics environment where researchers and data scientists can access a suite of integrated analytics tools and cloud computing resources to participate in data challenges and AI innovation.

Expected Benefits of Analytics Workbench:

- **Promote Collaboration**
- **Support AI Innovation**
- **Minimize Data Movement**
- **Improve User Experience**
- **Discover Data Insights**
- **Advance NIDDK Research Mission**



National Institute of Diabetes and Digestive and Kidney Diseases

Central Repository

Data Science Centric Challenge Series

Goals of NIDDK-CR Data-science centric challenge series

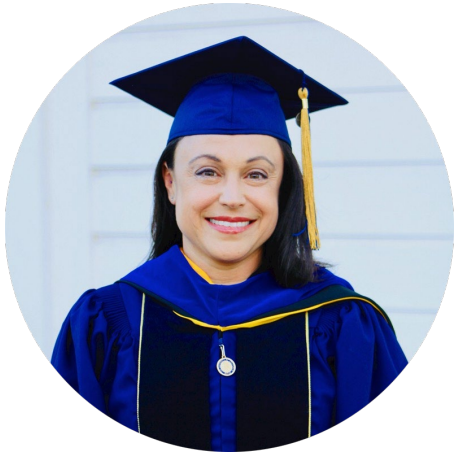
- Develop tools, approaches, models and/or methods to increase data interoperability and usability for artificial intelligence (AI) and machine learning (ML) applications
- Augment and enhance existing data for future secondary research, including data-driven discovery by AI/ML researchers
- Discover innovative approaches to enhance the utility of datasets for AI/ML applications

Visit our website for more information on our data-centric movement and to learn more about our past data-challenges

National Institute of Diabetes and Digestive and Kidney Diseases

Central Repository

Meet the Expert



Dr. Courtney D. Shelley, PhD, is a Health Data Scientist at Booz Allen Hamilton, where she focuses on data science education and AI-readiness of health-related data. She has supported the NIH Office of Data Science Strategy to develop online data science learning resources for pre-college and collegiate audiences, and to assess data science education across US universities to promote collaborative research between biomedical researchers and AI professionals. Prior to working at Booz Allen Hamilton, Dr. Shelley worked at Los Alamos National Laboratory, where she received the Postdoctoral Distinguished Performance Award for COVID-19 response efforts at local, state, and federal levels, as well as conducted research in suicide prevention with the support of the Department of Veterans Affairs and Million Veteran Program. She completed her PhD in Epidemiology with a focus on causal inference at University of California, Davis.

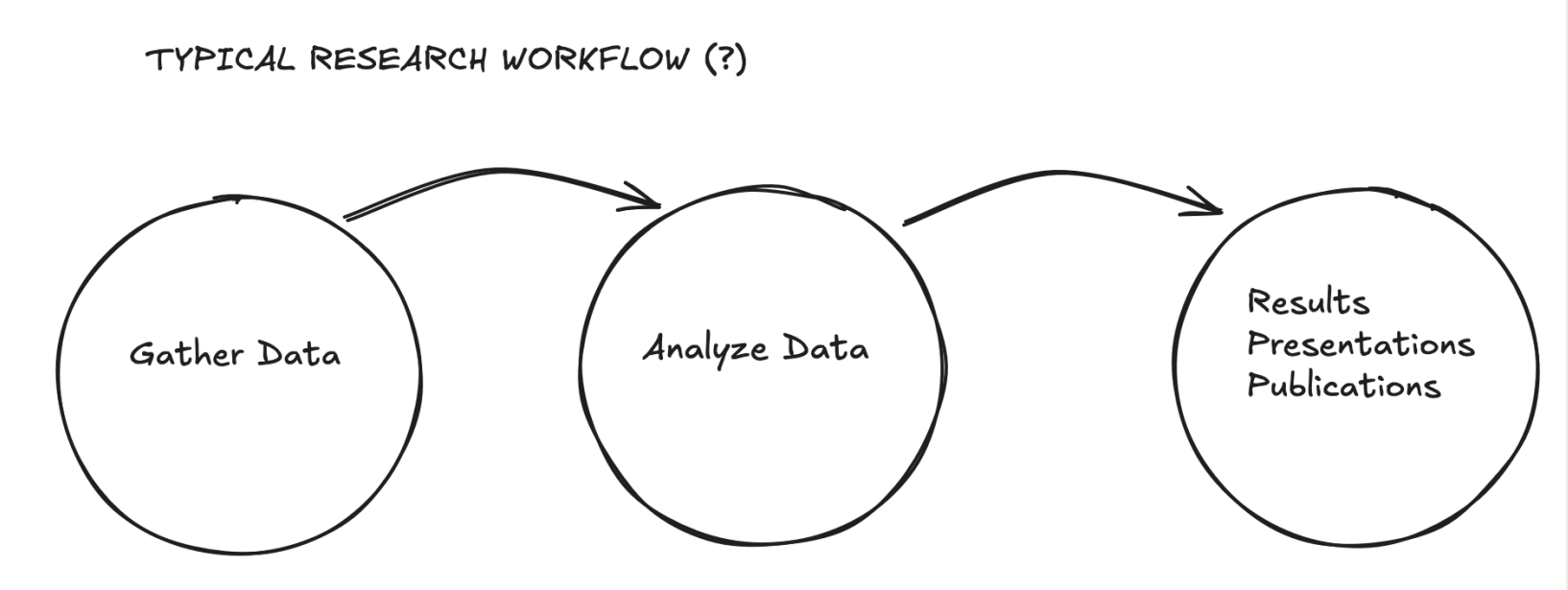
DESIGN PATTERNS FOR REPRODUCIBLE RESEARCH

Courtney D. Shelley, PhD
shelley_courtney@bah.com

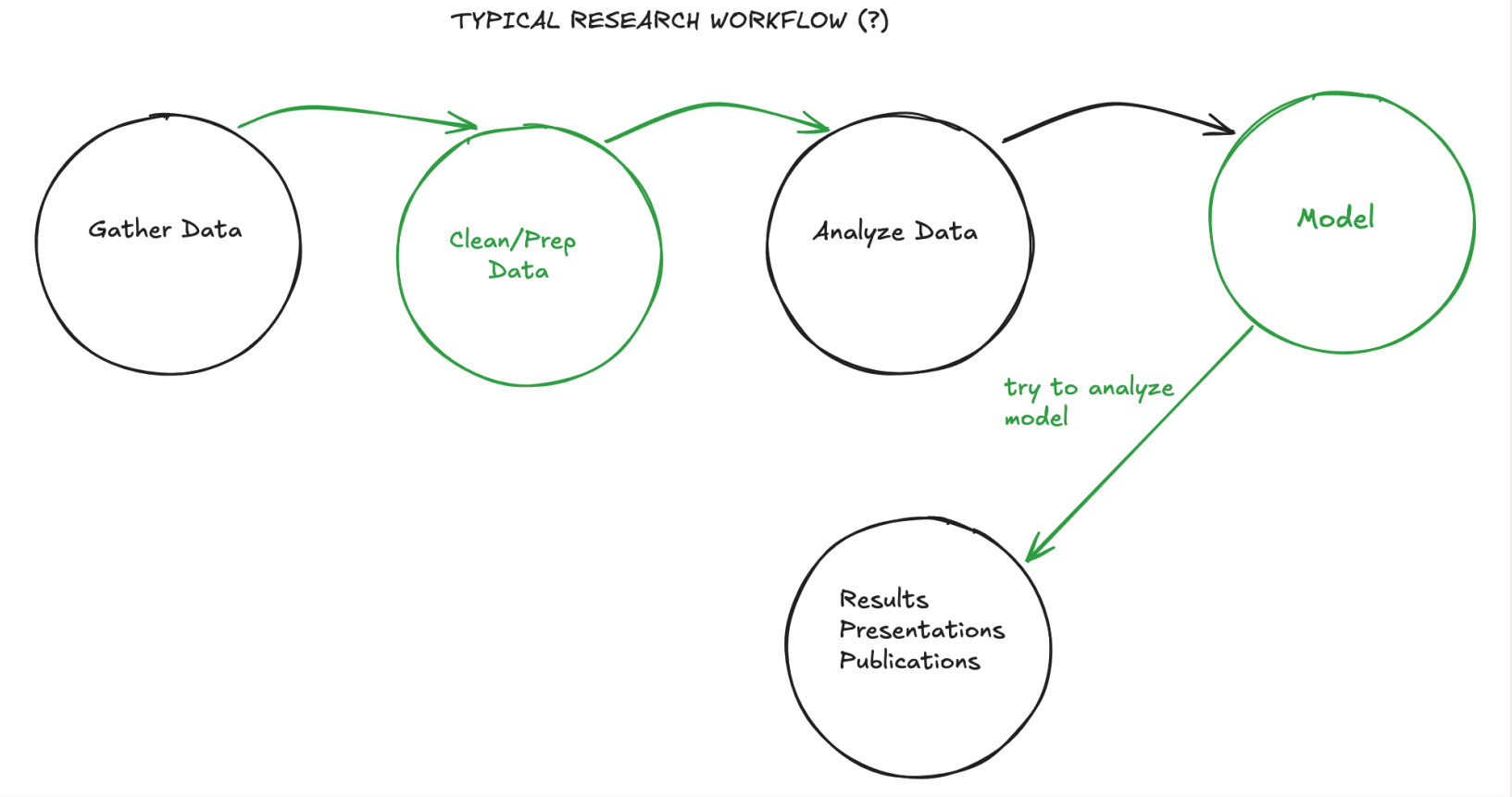
AGENDA

- Research in the Computational Sciences
- Why should research be reproducible?
- Design Patterns
  - *Pattern #1: Literate Programming*
  - *Pattern #2: Everything is Code*
  - *Pattern #3: Version Control and Provenance*
  - *Pattern #4: Code to Package*
  - *Pattern #5: Social Coding*

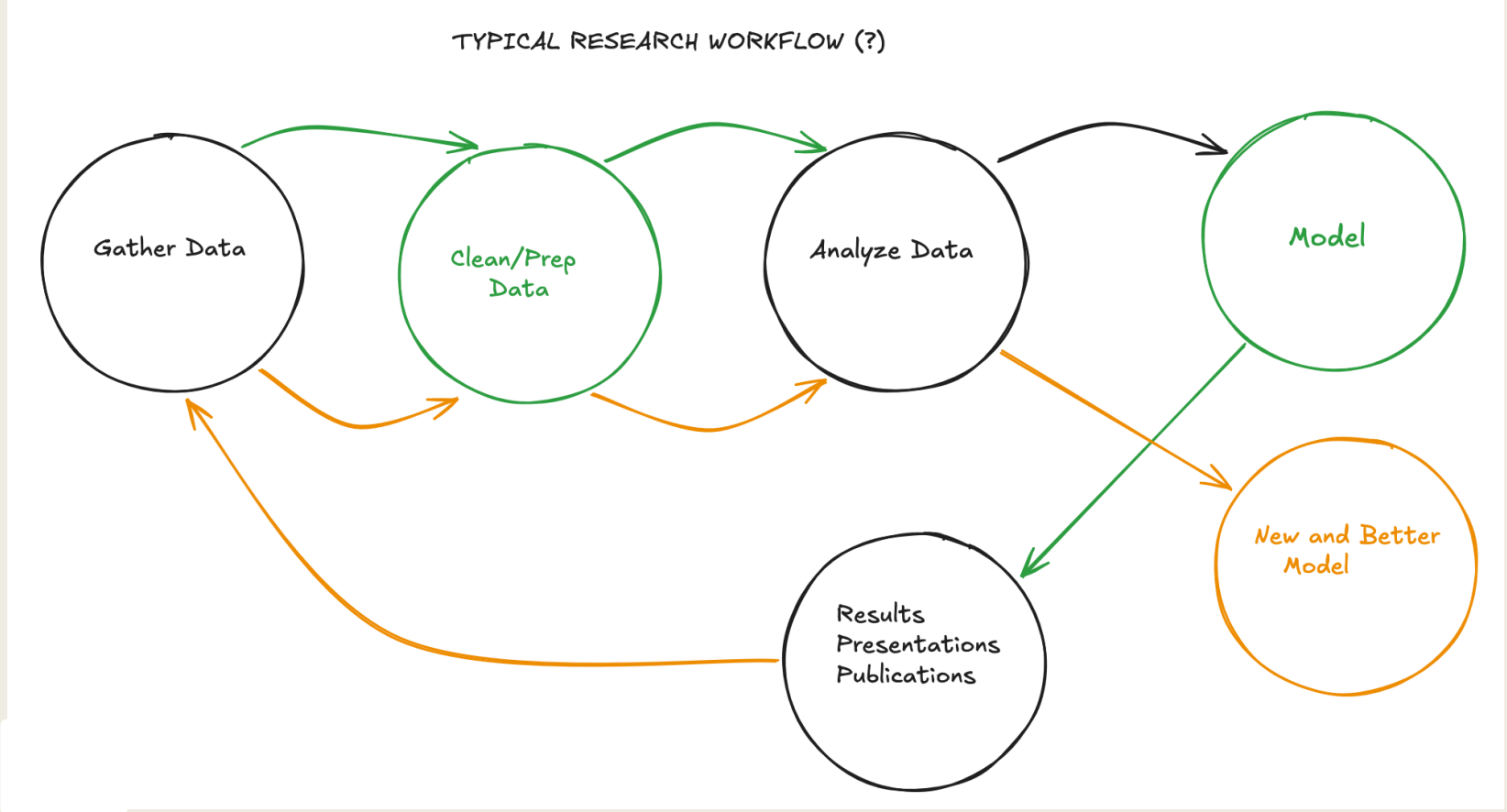
What we're taught …



Add the data science ...



But really ...


TYPICAL RESEARCH WORKFLOW (?)
Gather Data
Clean/Prep Data
Analyze Data
Model
New and Better Model
Results
Presentations
Publications

What is research?

Research is **replicable** if there is sufficient information available for independent researchers to use the same procedures and new data to obtain comparable results.

What is research?

Research is **replicable** if there is sufficient information available for independent researchers to use the same procedures and new data to obtain comparable results.

Research is "**the full software environment, code, and data that produced the results**".[1] Slideshows, journal articles, and presentations are ***merely advertising***.[2]

[1] Donoho 2010. *An invitation to reproducible computational research.* Biostatistics 11(3):385-8
[2] Gandrud 2020. *Reproducible Research with R and Rstudio (3rd Ed.)*

What is research?

Research is **replicable** if there is sufficient information available for independent researchers to use the same procedures and new data to obtain comparable results.

Research is "**the full software environment, code, and data that produced the results**".[1] Slideshows, journal articles, and presentations are ***merely advertising***.[2]

In computational sciences, this means "the data and code used to make a finding are available and they are sufficient for an independent researcher to recreate the findings". This is "**really reproducible research**".[3]

[1] Donoho 2010. *An invitation to reproducible computational research.* Biostatistics 11(3):385-8
[2] Gandrud 2020. *Reproducible Research with R and Rstudio (3rd Ed.)*
[3] Peng 2011. *Reproducible Research in Computational Science.* Science 334(6060):1226-7.

Why should research be reproducible?

1. For science.
2. For you.
3. For community.
4. For impact.

Why should research be reproducible?

1. For science.
    - Reproducibility leads to replication, validation, and extension.
2. For you.
3. For community.
4. For impact.

Why should research be reproducible?

1. For science.
    - Reproducibility leads to replication, validation, and extension.
2. For you.
    - The someone who most likely needs to reproduce your research is you!
3. For community.
4. For impact.

Why should research be reproducible?

1. For science.
    - Reproducibility leads to replication, validation, and extension.
2. For you.
    - The someone who most likely needs to reproduce your research is you!
3. For community.
    - FAIR[1] isn't just for data!
4. For impact.

[1] Findable, Interoperable, Accessible, and Reusable.

Why should research be reproducible?

1. For science.
    - Reproducibility leads to replication, validation, and extension.
2. For you.
    - The someone who most likely needs to reproduce your research is you!
3. For community.
    - FAIR isn't just for data!
4. For impact.
    - "The goal of reproducible research is to have more impact with our research".[2]

[1] Findable, Interoperable, Accessible, and Reusable.

[2] Vanderwalle et al 2007. *Experiences with Reproducible Research in Various Facets of Signal Processing Research.* IEEE ICASSP 4:1253-6.

Design Patterns

In software engineering, a **design pattern** is a reusable, generalized blueprint or template used to solve commonly occurring problems in code.

Design patterns represent best practices to help you write cleaner, more maintainable, and scalable analyses.

Our goal is a reproducible, FAIR[1] workflow.

[1] Findable, Accessible, Interoperable, Reusable

Design Patterns

In software engineering, a **design pattern** is a reusable, generalized blueprint or template used to solve commonly occurring problems in code.

Design patterns represent best practices to help you write cleaner, more maintainable, and scalable analyses.

Our goal is a reproducible, FAIR[1] workflow.

1. Literate Programming
2. Everything as Code
3. Version Control and Provenance
4. Code to Package
5. Social Coding

[1] Findable, Accessible, Interoperable, Reusable

1. Literate Programming

- **The Pattern:** Combine narrative text (*e.g., hypotheses, methodology, and interpretation*) with live, executable code blocks.
- **Implementation:** When the document is rendered, the code is executed and the latest outputs (tables, charts, and statistics) are dynamically embedded.
- **Value:** Eliminates the human error of manually copying and pasting results into a manuscript.

1. Literate Programming

RMarkdown

- Pre-installed with RStudio (an IDE for R)
- Used to create dynamic, reproducible documents, reports, and presentations
- Consists of **plain text** and **code chunks**:
  - **Plain text** uses Markdown to format text, including mathematical notation
  - **Code chunks** contain executable R code

1. Literate Programming

MARKDOWN

- Allows formatting of text chunks

- **Headings:** Add 1 to 6 # symbols at the start of a line to create different heading sizes (e.g., # Header 1, ### Header 3).
- **Bold:** Wrap your text in double asterisks or underscores: **bold**.
- **Italics:** Wrap your text in single asterisks or underscores: *italics*.
- **Lists:** Use numbers for ordered lists (e.g., 1. First) and hyphens (-), asterisks (*), or plus signs (+) for bullet points.
- **Links:** Place the display text in square brackets and the URL in parentheses: [Google](https://www.google.com).
- **Images:** Add an exclamation mark before the link format: ![Alt text](image-url.jpg).
- **Blockquotes:** Start a line with a > to quote text.
- **Math:** Within text (i.e., inline), wrap LaTeX notation in single dollar signs: $\sum$ will render as Σ when knit

1. Literate Programming

PSEUDOCODE

- A simple, human-readable description of code
- Plan code before writing it – easier to change
- Document what, why, and how

In a code chunk:

1. Use comments (#) to answer:
    - # why: *a brief, active description*
    - # what: *input(s) -> output(s)*
    - # how:
2. Review (preferably by explaining to someone else) that this would achieve your goal if coded up.
    - ***Yes?*** Move on. ***No?*** Redo.
3. Replace # how with base R functions.
3. Code up.
4. Test.
    - Check at every step that code is doing what you intend.
    - Comment with what you're doing and what you expect to see.

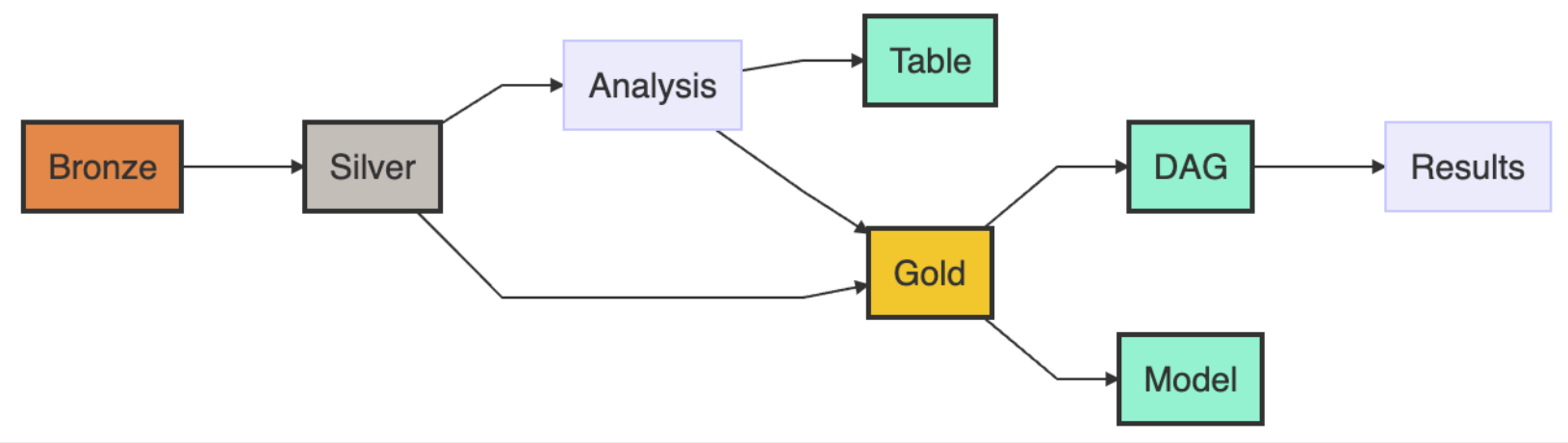
1. Literate Programming

CODE CHUNK OPTIONS

Call	Accepted Value	Notes
eval =	TRUE / FALSE	Execute code. FALSE is show code without running
echo =	TRUE / FALSE	Display source code. FALSE is hide code
include =	TRUE / FALSE	FALSE executes code chunk without showing any output
results =	“markup” “hide” “asis”	
message =	TRUE / FALSE	Hide messages
warning =	TRUE / FALSE	Hide warnings
error =	TRUE / FALSE	TRUE will continue to knit document even if an error occurs
fig.width = fig.height =	*numeric in inches*	Default is 7 in wide and 5 in high, alternative is fig.dim = c(8, 6)
fig.align =	“left” ”right” “center”	

2. Everything as Code

- **The Pattern:** All data acquisition, cleaning, and analysis steps are executed using scripts or compiled code rather than point-and-click graphical tools.
- **Implementation:** Break research into a directed acyclic graph (DAG) of sequential steps.
- **Value:** If raw data updates, the entire pipeline automatically re-evaluates without human intervention.

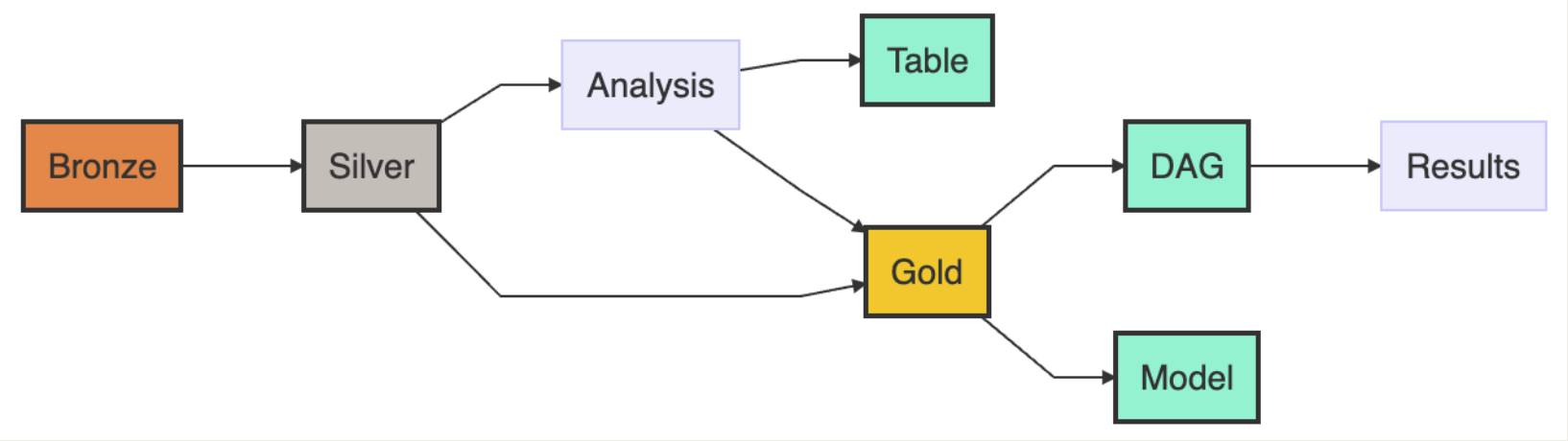


1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
3. Version Control and Provenance
4. Code to Package
5. Social Coding

2. Everything as Code

MERMAID DIAGRAMS

- *"Diagrams as Code"*
- Mermaid is an open-source diagramming tool
  - mermaid.ai for GUI web app
  - DiagrammeR for mermaid in R

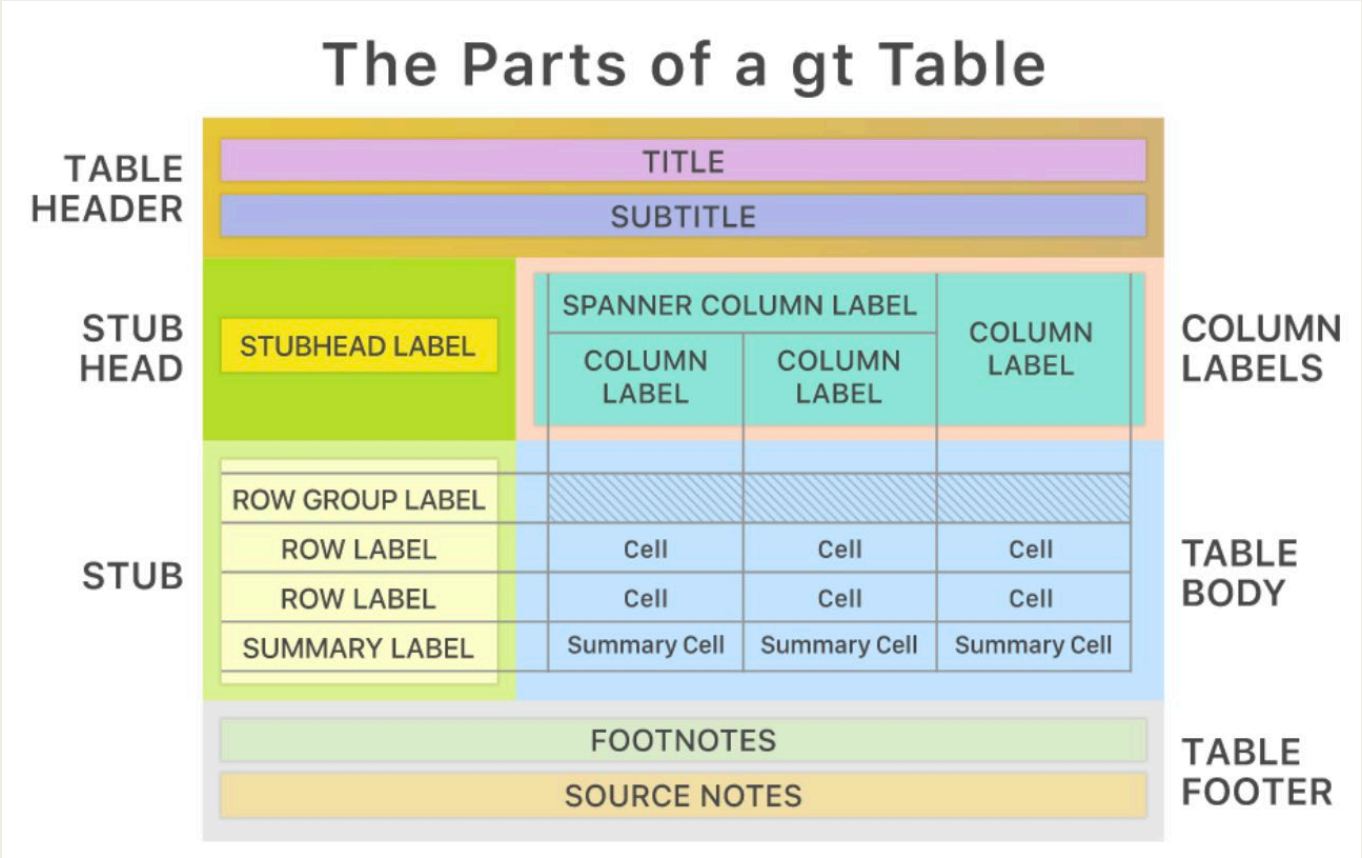


1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
3. Version Control and Provenance
4. Code to Package
5. Social Coding

2. Everything as Code

`gt`

- *"Tables as Code"*
- The `gt` package provides a "grammar of graphics" for tables



1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
   - `gt` for Tables
3. Version Control and Provenance
4. Code to Package
5. Social Coding

2. Everything as Code

LaTeX (pronounced LAH-tek)

- *"Math as Code"*
- *A special Markup language for math*
- *Two modes:*
  - *Inline. Surround the symbol or short equation in single $*
    - *The basic linear model formula is $y = ax + b$*
  - *Display Mode.*
    - *Create an indented chunk surrounded by double $$*

The basic linear model formula is:

```
$$
p_x = P(D|X = x) = \frac{1}{1+e^{-(a + bx)}}
$$
```

- Display mode can handle very complex equations and will render indented and italicized.

2. Everything as Code

LaTeX (pronounced LAH-tek)

- *"Math as Code"*
- *A special Markup language for math*

My favorite resource is Art of Problem Solving's LaTeX wiki

You can render Greek symbols, math symbols, calculus, vertical fractions, large (display style) brackets, aligned equations.

Most things start with a backslash (\) followed by a fairly easy-to-remember name, such as \beta, \sum, \frac{}{}.

Get as complicated as you need but **[pro tip] build it up and knit often!**

*(** Sometimes I even create a new .Rmd just to knit my equation if it takes a long time to knit the full doc)*

2. Everything as Code

Inline Code

- *Code results can be inserted directly into text*
  1. *Name the result you want to display*
  2. *Call inline with `` `r <name>` ``*
- *Also capable of doing some simple formatting with round(), paste(), or sprintf()*

3. Version Control and Provenance

- **The Pattern:** Track every modification made to the research code, text, and configuration files over time.
- **Implementation:** Maintain repositories with branching and commit tagging, ensuring major iterations of the paper or study correspond to specific code states.
- **Value:** Allows researchers to roll back broken analyses and provides a clear audit trail of the research's evolution.

3. Version Control and Provenance

`devtools`

- A collection of R packages that "just work"
- Designed to simplify and speed up package development
- `usethis` package automates package and project setup tasks

In your console, type:

```
library(devtools)
create_package("/file-path")

# Will open a new R session
library(devtools)

usethis::use_git_config(
    user.name = "GitHub.username",
    user.email = "name@email.com")

usethis::use_github()
```

3. Version Control and Provenance

In the terminal, type:

```
mkdir data
echo "/data" >> .gitignore
git add .gitignore
git commit -m "Update .gitignore rules"
git push
```

This created a subfolder in your .Rproj folder and then added it to the .gitignore file so Git will not track changes.

3. Version Control and Provenance

Your new workflow!

Do some work.

Run the following in terminal:

```
git status
    Shows status of all tracked files
git add <file-name>
   NOTE: Don't commit your bronze & silver data
   files
git commit -m "short-message"
git push
```

Commit. Commit! COMMIT!!

1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
   - `gt` for Tables
   - LaTex
3. Version Control and Provenance
   - `devtools`
4. Code to Package
5. Social Coding

4. Code to Package

- **The Pattern:** Write all code with the goal of eventually releasing to CRAN as an R Package.
- **Implementation:** Use code comments as auto-documentation for functions. Adopt a test-driven workflow.
- **Value:** A springboard to industry best practices of DevOps and productionizing. R package release is the endgame of FAIR research.

4. Code to Package

R packages contain functions, tests, and documentation

- You've used **functions** when writing R code throughout your career. They are reusable code to perform a certain task.
- Functions contain **documentation** users can access with *?function_name* in the console
- Good functions should also have unit tests. Unit tests record the intended functionality of your code. If you want to change it later (called **refactoring**), unit tests ensure the functionality remains the same.

4. Code to Package

We've already laid the groundwork to make our code functions because we reproducibly recorded tests and documentation.

1. I made a copy of our niddk_book4.Rmd into a .R file.

```
cp niddk_book4.Rmd functions.R
```

(add – commit –push)

2. I then removed all the narrative text and left just the code within the ```r ```

4. Code to Package

```
```{r extract_to_bronze, include = FALSE}
## EXTRACT DATASET
# why:  Extract dataset from UCI Machine Learning Repository and archive
# what: Input dataset name. Output bronze dataset.
# how:  fetch_ucirepo()    # downloads dataset and metadata

#install.packages("ucimlrepo",
#                 repos = c('https://coatless-rpkg.r-universe.dev',
#                           'https://cloud.r-project.org'))

library(ucimlrepo)
library(jsonlite)

chronic_kidney_disease <- fetch_ucirepo(name = "Chronic Kidney Disease")

# Save a bronze version as a JSON file
str(chronic_kidney_disease)
write_json(chronic_kidney_disease,
           path = here("data", paste0("ckd_bronze_", date, ".json")), pretty =
TRUE)

# Follow UCI MLR website instructions to construct dataset
ckd_data = chronic_kidney_disease$data
X = ckd_data$features
Y = ckd_data$targets

ckd = cbind(X, Y)
class(ckd)  # test that class equals "data.frame"

dim(ckd)    # test that dimensions equals (400   25)
```
```

1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
   - `gt` for Tables
   - LaTex
3. Version Control and Provenance
   - `devtools`
4. Code to Package
   - Functions
5. Social Coding

4. Code to Package

Convert our pseudocode notes into auto-documentation with roxygen2 notation

```
#----------- extract_to_bronze() --------------#
#' why:  Extract dataset from UCI Machine Learning
Repository and archive
#' what: Input dataset name. Output bronze dataset.

extract_to_bronze <- function(name) {
  mlr_data <- fetch_ucirepo(name = name)

  just_data = mlr_data$data
  X = just_data$features
  Y = just_data$targets

  dataset = cbind(X, Y)

  return(dataset)
}
```

4. Code to Package

```
#' Extract dataset from UCI Machine Learning Repository
and archive
#'
#' @param name A string. References a dataset
#'    downloadable from UCI machine learning repository
#' @returns A dataset, which is a subset of input list
#' @examples
#' extract_to_bronze('Chronic Kidney Disease')
extract_to_bronze <- function(name) {
  mlr_data <- fetch_ucirepo(name = name)

  just_data = mlr_data$data
  X = just_data$features
  Y = just_data$targets

  dataset = as.data.frame(cbind(X, Y))

  return(dataset)
}
```

1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
   - `gt` for Tables
   - LaTex
3. Version Control and Provenance
   - `devtools`
4. Code to Package
   - Functions
   - Auto-document
5. Social Coding

4. Code to Package

Unit Tests

- So far, we've been performing manual, ad hoc testing but our ultimate goal is a DevOps best practice of **automated testing**
- Unit tests is a formal way of testing our smallest code
- Supports two advanced design patterns: refactoring and test-driven development

- **Unit tests lead to:**
  - Fewer bugs – ensure code works before pushing to GitHub
  - Test-driven development – write the test first and then write code that passes the test
  - Robust code to refactor against – Refactoring is the process of rewriting code to make it cleaner, more readable, and more efficient

4. Code to Package

```
# unit test for extract_to_bronze
class(ckd)   # test that class equals "data.frame"
dim(ckd)     # test that dimensions equals (400   25)
```

Initial set-up:

```
usethis::use_testthat(3)
      # adds a tests folder to your repo
```

DO NOT EDIT tests/testhtat.R!

Workflow:

```
usethis::use_r('function')
      # creates and opens R/function.R
usethis::use_test('function')
      # creates and opens
      # tests/testthat/test-function.R
```

4. Code to Package

In <function>.R, include function code with documentation (add-commit-push!)

In test-<function>.R, cut and paste the # test thats we originally wrote.

You'll see the test_that function structure in a provided example.

- Each test describes what it's testing (e.g., "multiplication works")
- Each test has one more expectations (e.g., expect_equal(2*2, 4)

```
# unit test for extract_to_bronze
class(ckd)   # test that class equals "data.frame"
dim(ckd)     # test that dimensions equals (400   25)
```

```
test_that("data.frame is correctly assembled", {
  expect_equal(class(extract_to_bronze('Chronic Kidney Disease')), "data.frame")
  expect_equal(dim(extract_to_bronze("Chronic Kidney Disease")), c(400, 25))
})
```

1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
   - `gt` for Tables
   - LaTex
3. Version Control and Provenance
   - `devtools`
4. Code to Package
   - Functions
   - Auto-document
   - Unit tests
5. Social Coding

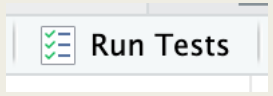
4. Code to Package

A test groups together multiple expectations to test the function output from a simple function. Each test should cover a single unit of functionality.

A test is created with test_that(description, code)

You want to arrange things such that, when a test fails, you'll know what's wrong and where in your code to look for the problem.

```
Your workflow in function window:
1. Edit function, test files
2. In test function, run tests manually with "Run
   Tests"
```




```
Your workflow in console:
1. usethis::use_r(), usethis::use_test()
2. devtools::load_all()
3. devtools::check()
```

1. Literate Programming
   - Jupyter Notebooks
   - Markdown
   - Pseudocode
2. Everything as Code
   - Mermaid Diagrams
   - `gt` for Tables
   - LaTex
3. Version Control and Provenance
   - `devtools`
4. Code to Package
   - Functions
   - Auto-document
   - Unit tests
5. Social Coding

5. Social Coding

- **The Pattern**: Developers collaborate and build projects incrementally, working together to document decisions, plan execution, and generate incremental improvements.

- **Implementation**: Use GitHub features of Issues, Pull Requests, and main and development branching to develop, and GitHub Actions to automatically run tests, trigger peer reviews, and merge code.

- **Value:** Cultivate collective code ownership and code quality through peer evaluation collaboration across the research team.

5. Social Coding

- GitHub Issues
- Pull Requests
- Main and development branches
- GitHub Actions

1. Literate Programming
2. Everything as Code
3. Version Control and Provenance
4. Code to Package
5. Social Coding

1. Literate Programming

Jupyter Notebook

- An interactive, browser-based web application
- Create and share documents containing text, live code, equations, and visualizations
- Consists of **cells**:
  - **Text cells** use Markdown to format text, including mathematical notation
  - **Code cells** can be in R, Python, Julia, or over 100 other languages, specified by a "kernel" when you begin the notebook.
  - **Raw cells** insert unrendered code into your document.