What is Reproducible Research?

A working definition for today:

A set of techniques that allow an analysis to be exactly recreated

Reproducible vs. Replicable

**Replicable** means if we **did the experiment again**, our result would be the same

- Generate Data
- Analysis 1
- Result
- Generate Data Again
- Analysis 2
- Same Result

**Reproducible** means that if we **use the same data and methods**, we get the identical result

- Generate Data
- Analysis
- Identical Result
- Reproduce Analysis
Ideal Analysis Workflow

Collect data → Format data → Analysis → Get input (sponsor, editor, teammate) → Publish or present results

Reality

Collect data → Format data → Analysis → Get input (sponsor, editor, teammate) → Publish or present results

New or updated data → Edit database (remove rows) → "We know that test X was bad, so remove it"

"Here's some new data. Can you just do it all over?"

Never think about project again
Reality

1. Collect data
2. Format data
3. Analysis
4. Get input (sponsor, editor, teammate)
5. Publish or present results

- New or updated data
- Edit database (remove rows)

"We know that test X was bad, so remove it"

"Here's some new data. Can you just do it all over?"
But why?

<table>
<thead>
<tr>
<th>I am a:</th>
<th>Why should I:</th>
<th>Because it benefits:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researcher/Analyst</td>
<td>Learn to use reproducible research techniques?</td>
<td>You</td>
</tr>
<tr>
<td>Team Leader/Project Leader</td>
<td>Encourage my team to use reproducible research techniques?</td>
<td>Your team/company</td>
</tr>
</tbody>
</table>

Reality

- Hand project off to new person
- Justify 6-month old decisions
- Tasked with similar analysis

Collect data

Format data

Analysis

Get input (sponsor, editor, teammate)

Publish or present results

New or updated data

Edit database (remove rows)

“We know that test X was bad, so remove it”

“Here’s some new data. Can you just do it all over?”

Never think about project again
## Benefits of Reproducible Research

<table>
<thead>
<tr>
<th>Benefits for you</th>
<th>Benefits for your team/company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Now</strong></td>
<td></td>
</tr>
<tr>
<td>Better work habits</td>
<td>Better teamwork</td>
</tr>
<tr>
<td>Changes (to code or data) are easier to make</td>
<td>Easier to hand off projects (e.g., when someone leaves or transitions)</td>
</tr>
<tr>
<td><strong>In the Future</strong></td>
<td></td>
</tr>
<tr>
<td>Easier to pick up projects again later or repeat analyses</td>
<td>Easier to respond to sponsor feedback</td>
</tr>
<tr>
<td></td>
<td>Supports transparency</td>
</tr>
<tr>
<td></td>
<td>More cumulative impact</td>
</tr>
</tbody>
</table>
Isn’t it extra work?

Credit: xkcd

**Same Data + Same Methods = Same Results**

- **Same Data**: The original inputs for the analysis are preserved
- **Same Methods**: Analysis tools or scripts are saved in a way that they can be applied directly to your data
- **Same Results**: All of your figures, tables, and conclusions can be reproduced by another researcher (or you in the future)

As you listen today, think about...

**Same Data**

Keep raw data Read Only

Organize your files
Use descriptive names

**Same Methods**

Document every decision
Stay organized and write readable code
Combine analysis and report generation

**Motivating Example**

**Rigid-Hulled Inflatable Boats**

The Navy wants to acquire a new rigid-hulled inflatable boat (RHIB). They have designed a test to measure the time required to launch the boats under different conditions.
(Fake) RHIB Test Data

<table>
<thead>
<tr>
<th>light</th>
<th>length</th>
<th>200kg-2pass</th>
<th>200kg-4pass</th>
<th>100kg-2pass</th>
<th>100kg-4pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>7</td>
<td>14</td>
<td>17</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>day</td>
<td>13</td>
<td>16</td>
<td>18</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>night</td>
<td>7</td>
<td>20</td>
<td>25</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>night</td>
<td>13</td>
<td>21</td>
<td>25</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

Disclaimers!

- The rest of this talk will be much more technical
- We will present a **NON EXHAUSTIVE** list of techniques **USING R**
- You don’t have to use all of this right away
Organizing files

All project-related files and scripts should be in a single overarching project directory

RHIB_Analysis/
|-- data_raw/
|-- data_clean/
|-- docs/
|-- figures/
|-- lib/
|-- munge/
|-- reports/
|-- src/
README
RHIB_Analysis.Rproj
R Projects – Why use them?

R projects make it straightforward to divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents.

.Rproj sets project-specific variables and formatting niceties

**RHIB_Analysis.Rproj**

Version: 1.0

- RestoreWorkspace: Default
- SaveWorkspace: Default
- AlwaysSaveHistory: Default
- EnableCodeIndexing: Yes
- UseSpacesForTab: Yes
- NumSpacesForTab: 2
- Encoding: UTF-8
- RnwWeave: Sweave
- LaTeX: pdfLaTeX

How do I set up an R Project?

.Rproj files should be created automatically for you
Version Control – What?

Turn project (directory) into a database (repository)

Git is Ctrl-s on steroids

Version Control – Why?

- Traverse project history
- Collaborate asynchronously
- Work offline
- Decentralize storage of data and code
- Preserve sanity

Version control should be ubiquitous

- Data
- Source Code
- Graphs
- Manuscripts
- Presentations
- Bibliographies

Set yourself up for git success

To take full advantage of version control:

- Data – .csv
- Source Code – .R, .py
- Graphs – .R, .py
- Manuscripts – markdown
- Presentations – markdown
- Bibliographies – BibTeX

Plain text is light, readable, portable, and universal

Resources to get started

Git is built into Rstudio

happygitwithr.com
Managing your data

Raw data folder should be treated as Read-Only
Consider removing ‘write’ permissions from raw data files
Plain text data are best for short and medium duration projects

Reshaping Data

Recall our example dataset…

<table>
<thead>
<tr>
<th></th>
<th>light</th>
<th>length</th>
<th>200kg-2pass</th>
<th>200kg-4pass</th>
<th>100kg-2pass</th>
<th>100kg-4pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>7</td>
<td>14</td>
<td>17</td>
<td>12</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>day</td>
<td>13</td>
<td>16</td>
<td>18</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>night</td>
<td>7</td>
<td>20</td>
<td>25</td>
<td>19</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>night</td>
<td>13</td>
<td>21</td>
<td>25</td>
<td>18</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

This “wide” format is common – logical to record data in this way

Reshaping Data

“Wide” format is good for answering simple questions

What was the time to launch a 13 m boat during the day when loaded with 200 kg and 4 passengers?
18 minutes
Reshaping Data

... but not good for answering more complex questions

How does the time to launch a 13 m boat with 4 passengers change between night and day?

Reshaping Data – Tidy Data

Prepared Data

<table>
<thead>
<tr>
<th>light</th>
<th>length</th>
<th>load</th>
<th>passengers</th>
<th>launch_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>day</td>
<td>7</td>
<td>200</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>day</td>
<td>7</td>
<td>200</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>day</td>
<td>7</td>
<td>100</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>day</td>
<td>7</td>
<td>100</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>day</td>
<td>13</td>
<td>200</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>day</td>
<td>13</td>
<td>200</td>
<td>4</td>
<td>18</td>
</tr>
</tbody>
</table>

Notes on Reshaping Data

dplyr and tidyr offer simple functions for reshaping data

```r
RHB %>%
gather("loadpassengers", "launch_time", -light, -length) %>%
separate(loadpassengers, into = c("load", "passengers"), sep = "-") %>%
```
# extract numeric portion of load and passenger columns
```r
mutate(load = as.numeric(str_extract(load, "[:digit:]{3,}")),
       passengers = as.numeric(str_extract(passengers, "[:digit:]{1,}")))```

```r
## # A tibble: 6 x 5
## #  light length load passengers launch_time
## # <chr> <int> <int> <int>
## 1 day    7  200   2   14
## 2 day    7  200   4   17
## 3 day    7  100   2   12
## 4 day    7  100   4   16
## 5 day   13  200   2   16
## 6 day   13  200   4   18
```

What if I have to change one or two points?

Reproducible workflows can include data editing:

Use code to change data points (not manual modification), which is much easier to do when your data is in tidy format

Document your steps and reasoning

```r
# Per email conversation on 1 March 2019, remove the 13m 200kg 2 passenger night trial
RHB_tidy <- RHB_tidy[!which(RHB_tidy$light == "night" & RHB_tidy$length == 13 & RHB_tidy$load == 200 & RHB_tidy$passengers == 2),]
```
Document everything

Consider comment headers on all scripts

What does the script do? Who wrote it? Last modified? etc.

```
# This script generates RHIB launch time estimates with confidence intervals.
#
# Author: John Doe
# Created: 22 Feb 2019
# Modified: 1 April 2019
```

Make your code easy to read

```r
my_fun <- function(x=3,y=3*2^2){y%%x+1}
```

Style guides are available – style.tidyverse.org
Nested parentheses can be difficult to follow

```r
bop_on(scoop_up(hop_through(little_bunny, forest), field mice), head)
```

Adding white space and new lines help

```r
bop_on(
  scoop_up(
    hop_through(little_bunny, forest),
    field mice),
  head
)
```

*(Example from Hadley Wickham)*

Too many intermediate steps leads to nondescript variable names and cluttered workspaces

```
foofoo <- little_bunny
bunnyHop <- hop_through(foofoo, forest)
bunnyHopScoop <- scoop_up(bunnyHop, field mice)
foofooFinal <- bop_on(bunnyHopScoop, head)
```

Use the pipe operator to “chain” steps together

```
foofoo <- little_bunny %>%
  hop_through(forest) %>%
  scoop_up(field_mice) %>%
  bop_on(head)
```

*(Example from Hadley Wickham)*
Turn repeated code into functions

D.R.Y. - Don’t repeat yourself
Copy/paste can introduce errors that are difficult to track down

Do not repeat data munging code
Save data creation code as function

```r
reshape_RHIB_data <- function(wide_data){
  wide_data %>%
    gather("loadpassengers", "launch_time", -light, -length) %>%
    separate(loadpassengers, into = c("load", "passengers"), sep = "-") %>%
    # extract numeric portion of load and passenger columns
    mutate(load = as.numeric(str_extract(load, "[[:digit:]{3,}]")),
           passengers = as.numeric(str_extract(passengers, "[[:digit:]{1,}]")))
}

RHIB_tidy <- reshape_RHIB_data(RHIB)
```

Use good file-naming practices

<table>
<thead>
<tr>
<th>BAD</th>
<th>GOOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>update.R</td>
<td>01clean_data.R</td>
</tr>
</tbody>
</table>
Use good file-naming practices

- Letters, numbers, periods, hyphens, and underscores only
- Please, no whitespace!
- Machine readable
- Human readable
- Plays well with default ordering

See “naming things” talk by Jenny Bryan

Use seeds to make analyses “reproducibly random”

```r
x <- rnorm(1e5, mean = 0, sd = 1)
y <- 2*x + rnorm(1e5, mean = 0, sd = .01)

lm(y ~ x)$coefficients[2]
```

```
## x
## 1.999983
```

```r
set.seed(4850)
x <- rnorm(1e5, mean = 0, sd = 1)
y <- 2*x + rnorm(1e5, mean = 0, sd = .01)

lm(y ~ x)$coefficients[2]
```

```
## x
## 2.000088
```

```r
set.seed(4850)
x <- rnorm(1e5, mean = 0, sd = 1)
y <- 2*x + rnorm(1e5, mean = 0, sd = .01)

lm(y ~ x)$coefficients[2]
```

```
## x
## 2.000088
```
R Markdown Primer

R Markdown allows you to combine your code, its results, and written narrative in a single document. Directly links your analysis with the reporting of your results, allowing you to see the code the generated each figure or produced a value.

R Markdown Primer

An R Markdown file is a plain text file with the extension .Rmd
There are 3 types of content in an .Rmd file:

1. Header
2. R code
3. Text with simple formatting, such as **bold** or *italics*

**R Markdown Primer**

```r
library(tidyverse)
library(here)
```

```{r, echo = FALSE}
load(here("data_clean", "RHIB_tidy.Rds"))
```

Launch time data from `r nrow(RHIB_tidy)` trials were used to assess rigid-hulled inflatable boat launch times. The distribution of launch times by boat length are shown in the figure below.

```{r, echo = FALSE}
RHIB_tidy %>%
  ggplot(aes(x = as.factor(length), y = launch_time, fill = light)) +
  geom_boxplot() +
  labs(x = "Length",
       y = "Launch Time",
       fill = "Light Condition",
       title = "Launch Times by Boat Length and Light Condition")
```

**R Markdown Primer**

knitr and Pandoc do the heavy lifting
R Markdown Primer

New R Markdown

- Document
- Presentation
- Shiny
- From Template

**Title:** Rigid-Hulled Inflatable Boat Analysis

**Author:** Andrew Flack

**Default Output Format:**

- HTML
  
  Recommended format for authoring (you can switch to PDF or Word output anytime).

- PDF
  
  PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

- Word
  
  Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).
R Markdown Primer

```r
---
title: "Rigid-Hulled Inflatable Boat Analysis"
author: "Andrew Flack"
date: "February 27, 2019"
output: html_document
---

```r
library(tidyverse)
library(here)

```r
## Data Exploration

```r
load(here("data_clean", "RHIB_tidy.Rds"))

Launch time data from 'r nrow(RHIB_tidy)' trials were used to assess rigid-hulled inflatable boat launch times. The distribution of launch times by boat length are shown in figure below.

```r
ggplot(aes(x = as.factor(length), y = launch_time, fill = light)) + geom_boxplot() + labs(x = "Length", y = "Launch Time", fill = "Light Condition", title = "Launch Times by Boat Length and Light Condition")
Rigid-Hulled Inflatable Boat Analysis
Andrew Flack
February 27, 2019

Data Exploration
Launch time data from 15 trials were used to assess rigid-hulled inflatable boat launch times. The distribution of launch times by boat length are shown in figure below.

Launch Times by Boat Length and Light Condition

Light Condition
- day
- night
You are told that there was a problem with one trial, and it should be removed from your analysis. You update your analysis, and also update the numbers in your report.

But you don’t catch all references to the number of trials...
Replace “magic numbers” throughout your narrative with inline code

Inline code:
The average launch time is `r mean(RHIB_tidy$launch_time)` minutes.

Rendered code:
“The average launch time is 18.375 minutes.”

“This sounds great for simple analyses, but…”

…my analysis is spread across multiple scripts!

Won’t the act of creating the .Rmd report require me to copy/paste code?

Code that resides in other scripts can be referenced in the .Rmd

Tag a section of code in your script...

```r
## @knitr tidy_RHIB_data
RHIB_tidy <- RHIB %>%
  gather("loadpassengers", "launch_time", -light, -length) %>%
  separate(loadpassengers, into = c("load", "passengers"), sep = "-") %>%
  # extract numeric portion of load and passenger columns
  mutate(load = as.numeric(str_extract(load, "[:digit:]{3,}")),
         passengers = as.numeric(str_extract(passengers, "[:digit:]{1,}")))
```

... then reference that tag in your R Markdown document

```
# Analysis
```{r, include = FALSE}
knitr::read_chunk("01_read_and_tidy.R")
```{r tidy_RHIB_data}
You can pass parameters into reports to set values for key inputs
Cache code chunks when computations take a long time to run

# Analysis
```
r, include = FALSE, cache = TRUE
knitr::read_chunk("01_read_and_tidy.R")
<<tidy_RHIB_data>>
```
Cached chunks are evaluated only when necessary
Use your organization's template by saving a reference document in your project directory

This also works with PowerPoint templates
Restart R and re-run code throughout development

As you work through an analysis, you may accumulate temporary variables, unused objects, and other artifacts in your environment.

Running your code in a fresh R session ensures that there are no lurking dependencies.

Starting scripts with `rm(list = ls())` is not a good habit

```r
# clear the workspace
rm(list = ls())
```

Deletes user-created objects from the environment, but does **not** create a new R session.

If you plan to share your analysis, it’s bad form to wipe a collaborator’s environment!

Unit Test Functions

The `testthat` package makes it easy to test that your functions actually do what you think they do, and it is easy to integrate into your workflow.

`my_function.R`
my_function <- function(a, b){
  sum(a, b)
}

Unit Test Functions

test_my_function.R
test_that('output values are correct', {
  expect_equal(my_function(1, 1), 2)
  expect_equal(my_function(0, 0), 0)
  expect_equal(my_function(-1, -1), -2)
  expect_equal(my_function(-1, 1), 0)
})

test_that('data types correct', {
  expect_is(my_function(1, 1), 'numeric')
})

Defensive Programming

New data can vary in ways that don’t break your code
Think about ways that new data could differ, and verify assumptions about data input to analysis pipelines

Sometimes called “assertions-based” programming

Defensive Programming

`assertr` package enables assertions-based checks within pipelines

```r
reshape_RHIB_data <- function(wide_data){
  wide_data %>%
    # verify that input data has columns named "light" and "length"
    verify(has_all_names("light", "length")) %>%
    gather("loadpassengers", "launch_time", -light, -length) %>%
    # verify that input data has at least 3 digit load, at least 1 digit passengers,
    # and that they are separated by "-
    verify(str_detect(loadpassengers, "[:digit:]{3,}kg-[:digit:]{1,}pass")) %>%
    separate(loadpassengers, into = c("load", "passengers"), sep = "-") %>%
    # extract numeric portion of load and passenger columns
    mutate(load = as.numeric(str_extract(load, "[:digit:]{3,}")),
           passengers = as.numeric(str_extract(passengers, "[:digit:]{1,}"))) %>%
    # verify that all recorded values for launch time are positive
    verify(launch_time > 0)
}
```
Considerations when a collaborator runs your code on their computer

1. They need to know how to run your code
2. The working directory and file paths need to work properly
3. Required packages and dependencies must be installed and loaded

1. **They need to know how to run your code**

Do your scripts need to be run in a certain order?
If so, describe that order in your README

# README

1. Run `munge/01_read_and_tidy.R`
2. Run `munge/02_remove_bad_trial.R`
3. Open `reports/RHIB_Analysis_Report.Rmd` and click the "knit" button.

OR

Consider a `run_all.R` script
run_all.R

```r
# This script runs the full RHIB Analysis and generates the report.
#
# Author: John Doe
# Created: 1 March 2019
# Modified: 1 April 2019

# Load required packages
library(tidyverse)

# Source custom functions
source("lib/calculate_foobar_metric.R")

# Clean and prepare data
source("munge/01_read_and_tidy.R")
source("munge/02_remove_bad_trial.R")
source("munge/03_add_new_test_data.R")

# Generate report
knit("reports/RHIB_Analysis_Report.Rmd")
```

2. The working directory and file paths need to work properly

Your analysis should be portable – your collaborator should be able to save your analysis anywhere on their computer and run it without error.

R Project files offer a simple solution.

The `here` package automatically detects the root directory of your analysis project and helps write platform-independent file paths.

```r
library(here)

here()
```

```
## [1] "C:/Users/aflack/Desktop/reproducible-research-mini-tutorial"
```

2. The working directory and file paths need to work properly

What’s wrong with `setwd()`?

```r
setwd("/Users/andrew/my_projects/2019/foo/bar/")
```

Using `setwd()` makes your code brittle and no longer portable.

Your collaborator has a different directory structure and your script won’t work.

You might move the file or change your directory structure and your script won’t work.
3. Required packages and dependencies must be installed and loaded

```r
install_if_needed <- function(required_pkg){
  is_installed <- required_pkg %in% installed.packages()
  if(!is_installed){
    message("Attempting to download and install: ", required_pkg)
    install.packages(required_pkg,
      dependencies = TRUE,
      repos = "https://cran.revolutionanalytics.com")
  }
}

pkgs <- c("devtools", "stringr", "lubridate",  # utilities
  "rvest", "httr",                    # data acquisition
  "readxl", "readr",                  # data loading
  "dplyr", "tidyr",                   # data wrangling
  "ggplot2")                           # data visualization

invisible(lapply(pkgs, install_if_needed))
invisible(lapply(pkgs, library, character.only = TRUE))
```

(Example from Wil Doane (IDA))

3. Required packages and dependencies must be installed and loaded

If your code requires a specific version of a package:

- Document specific version requirements in README (sessionInfo() can be helpful here)
- Check for proper package versions in your run_all.R script and stop if necessary
Wrap up

Basic principles can be implemented in any analysis workflow

Whether you’re using R, JMP, Excel, or any other tool, basic principles of reproducible research can still be applied

Excel

Use comments liberally

Make a new copy of an Excel calculator for each new analysis rather than using it interactively
(Example from Jon Bell (IDA))

Excel

Document data transformation steps with Power Query
JMP

Create scripts when possible
Resources

These slides were produced entirely in R Markdown

Reproduce the slides and inspect the source code
Resources

A full end-to-end analysis, including a dynamically-generated report, is also included in the repo.

Questions?

SessionInfo()

```r
sessionInfo()
```

```
## R version 3.4.2 (2017-09-28)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats graphics grDevices utils datasets methods base
##
## other attached packages:
```

42
# References