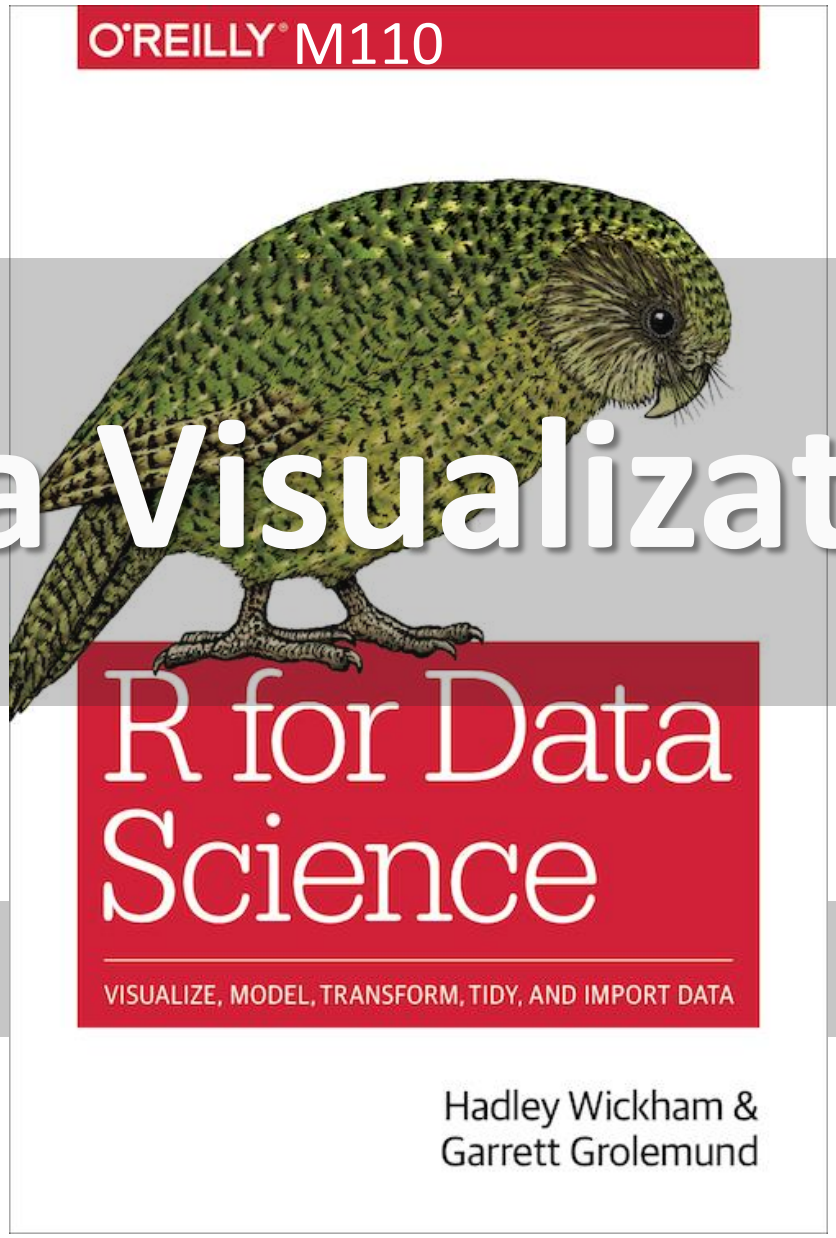
Data Visualization



https://r4ds.had.co.nz/

Why use R and its visualization packages

- R is the most popular language in the world of data science.
- R is an interpreted language. Hence, we can run code without any compiler.
- R is a vector language, so anyone can add functions to a single vector without putting in a loop. Hence, R is powerful and faster than other languages.

For data viz, R allows:

- Reproducibility
- Create many variations and iterations without having to start from scratch

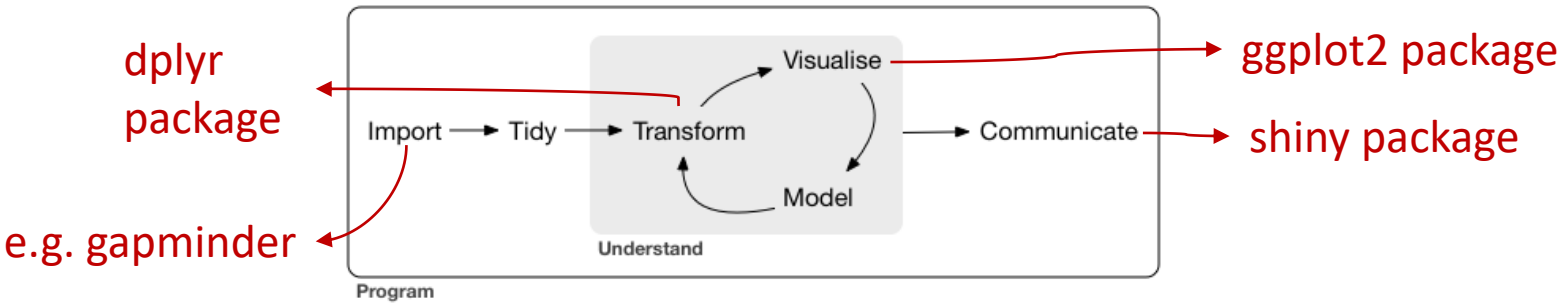
The workflow for visualizing with R

- **import** your data into R
- **tidy** your data
- **transform** your data (e.g. narrow in on observations of interest, create new variables, calculate a set of summary statistics)

wrangling

- **visualize**
- **model** (e.g. to answer questions you've made)
- **communicate**

data viz



To start

- Download the most recent version of **R** https://www.r-project.org/
- Once R is installed, download and install the IDE **R Studio** https://www.rstudio.com/
- In R Studio, install add-on packages for R:
  - **Tidyverse** - tidyverse.org

    install.packages("tidyverse") # includes ggplot2
  - **Shiny**

    install.packages("shiny")

    install.packages("rsconnect") # For publishing apps online
- You may also want to create a shinyapps.io account

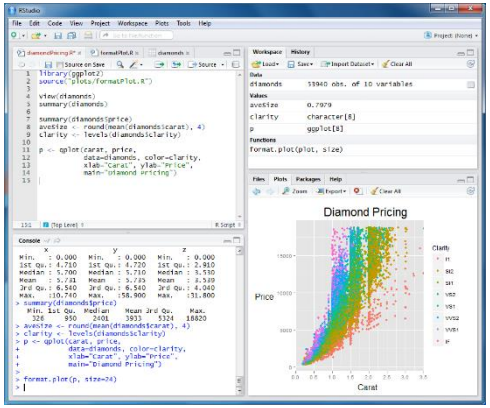
In R Studio, type at R's command prompt (located in the window named "Console")

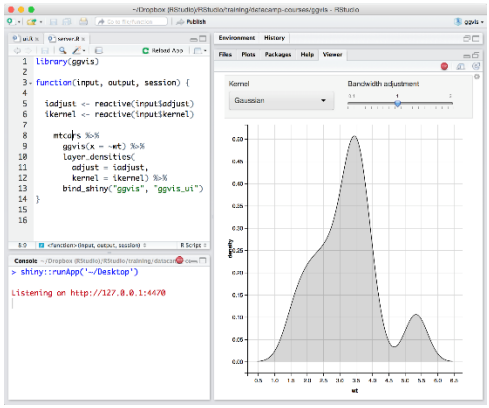
R resources

- The R Project for Statistical Computing https://www.r-project.org/
- The Comprehensive R Archive Network - https://cran.rstudio.com/
- TutorialsPoint - Learn R programming https://www.tutorialspoint.com/r/index.htm
- Swirl R package for learning R programming and data science http://swirlstats.com/
- Grolemund, G., & Wickham, H. (2017). ***R for Data Science***. O'Reilly Media. Retrieved from https://r4ds.had.co.nz/
- Codeschool R courses http://tryr.codeschool.com
- DataCamp course https://www.datacamp.com/

RStudio resources

- RStudio - https://www.rstudio.com/
- RStudio videos https://resources.rstudio.com/
- Rstudio's IDE features: https://rstudio.com/products/rstudio/features
- Rstudio's IDE video presentation: https://fast.wistia.net/embed/iframe/520zbd3tij?videoFoam=true
- RStudio Cheatsheets

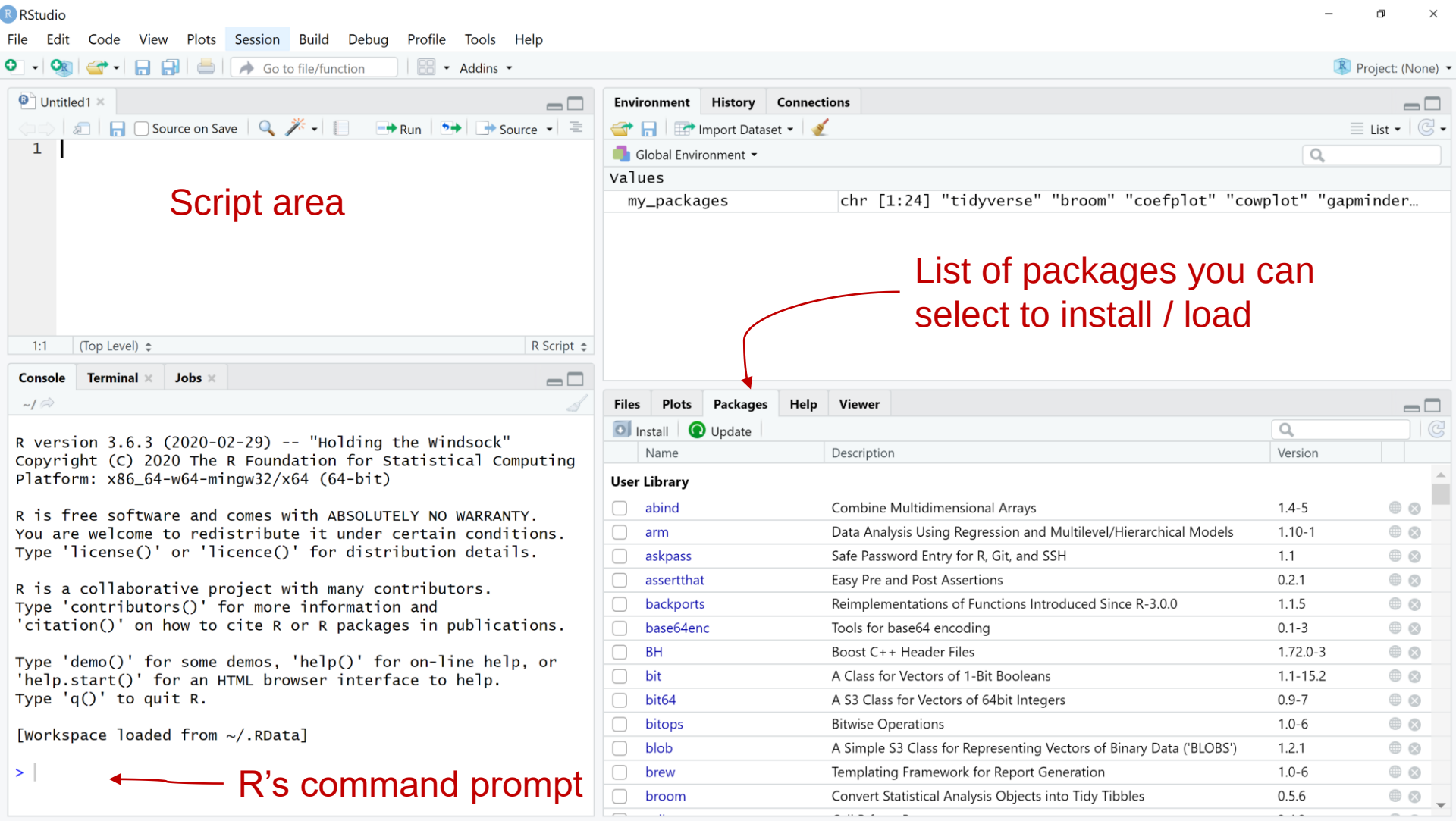




R Studio

Note: run as administrator!

- This is what RStudio looks like once installed



R Studio - installing packages

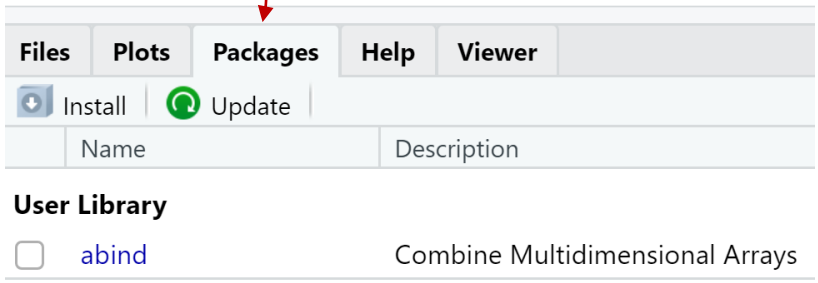
Note: R is case sensitive!

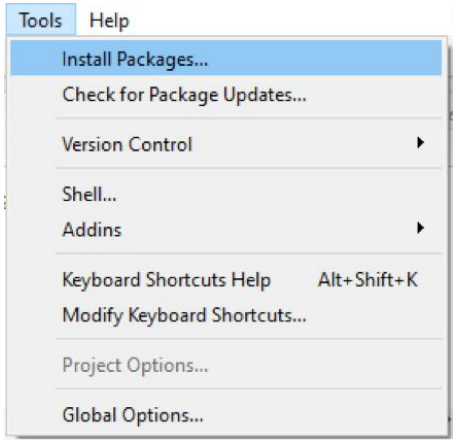
- Install at command line
  > install.packages("tidyverse")

the package name

- Or install more than one packages at a time
  > install.packages(c("tidyverse", "ggplot2", "dplyr"))

- Or from the Packages tab, or the menu





- To view the current library path(s) location
  > .libPaths()

R Studio - loading packages

- You only need to install a package once, but you need to **reload** it every time you start a new session. E.g.

```
> library(tidyverse) # load the package tidyverse
> library(dplyr)
> library(gapminder)
```

at R's command prompt

- Or, altogether in a script file and then highlight and Ctrl + Enter

```
library(tidyverse)
library(dplyr)
library(gapminder)
```

- To see which packages are loaded in a session

```
> (.packages()) # listed by most recently loaded first
```

tidyverse

- **tidyverse** is a collection of packages that are useful for many tasks like data management, data reshaping, data formatting, memory management, etc.
  It includes:
  - **ggplot2** (contains the functions used to create plots)
  - dplyr (contains functions to work with data, e.g. subset, aggregate data)
  - tidyr
  - readr
  - purrr
  - tibble
  - stringr
  - Forcats

https://www.tidyverse.org/packages

ggplot2

- **ggplot** is a system for declaratively creating graphics, based on The Grammar of Graphics.

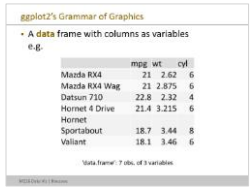
  We provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the rest.

  https://ggplot2.tidyverse.org/

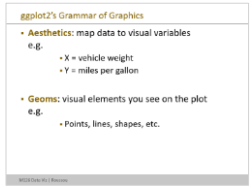
ggplot2 = the Grammar of Graphics plot

The Grammar of Graphics components:

1. **Data**: what we want to visualize. In R, data is stored in a data.frame, a rectangular collection of variables (in the columns) and observations (in the rows).



2. **Aesthetics**: visual properties of geoms e.g. x and y positions, colors, shapes, transparency, etc.



3. **Geoms**: the geometrical object that a plot uses to represent data, e.g. points, lines, areas, polygons, etc.

ggplot2's Grammar of Graphics

- A **data** frame with columns as variables
  e.g.

	mpg	wt	cyl
Mazda RX4	21	2.62	6
Mazda RX4 Wag	21	2.875	6
Datsun 710	22.8	2.32	4
Hornet 4 Drive	21.4	3.215	6
Hornet Sportabout	18.7	3.44	8
Valiant	18.1	3.46	6

'data.frame': 7 obs. of 3 variables

ggplot2's Grammar of Graphics

- **Aesthetics**: map data to visual variables

  e.g.

  - X = vehicle weight
  - Y = miles per gallon

- **Geoms**: visual elements you see on the plot

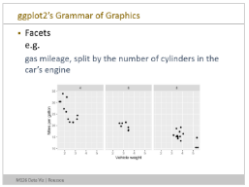
  e.g.

  - Points, lines, shapes, etc.

ggplot2 = the Grammar of Graphics plot

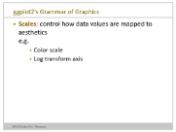
The Grammar of Graphics components (cont'd):

4. **Statistical transformations** (stats): The stats summarize data in many useful ways. E.g., binning and counting to create a histogram, regression line for regression analysis, etc.



5. **Facets**: allow to break up the data into subsets to visualize as small multiples.

6. **Scales**: Scales map values in the data space to values in the aesthetic space, whether it is color, size, or shape.



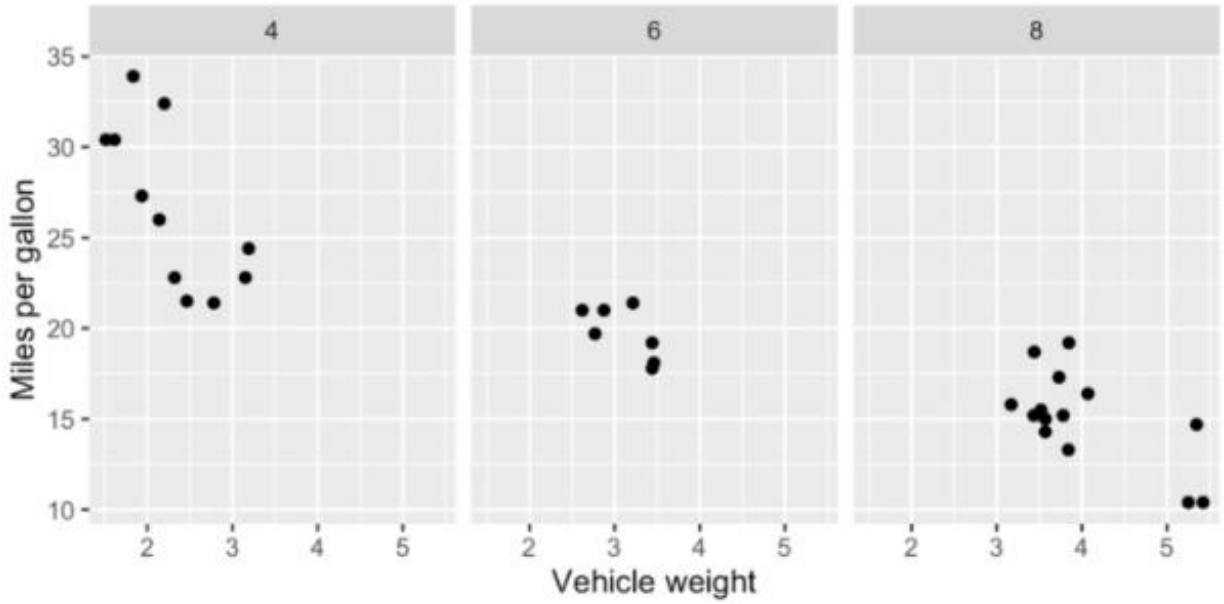
7. **Coordinates**: controls how data coordinates are mapped to the visual plane of graphics

ggplot2's Grammar of Graphics

- Facets

  e.g.

  gas mileage, split by the number of cylinders in the car's engine



ggplot2's Grammar of Graphics

- **Scales**: control how data values are mapped to aesthetics

  e.g.

  - Color scale
  - Log transform axis

Flow of action in ggplot2

Start with a table of data →

map the variables you want to display to aesthetics like position, color or shape →

choose one or more geoms to draw the graph

Steps for making a plot with ggplot2

1. **Assign data**: declare the input data.frame (e.g., data=acs).
2. **Assign** a variable to aes **or set** the **aesthetics** (with aes()): assign or set the x-axis variable. E.g., to make a histogram or density curve using the x-axis variable only. To make a scatter plot, assign the y-axis variable together. You can assign a variable to color, fill, or size variable (e.g., color=sex) or set the variable(e.g., color="black").
3. **Specify the geom**(s): select various geoms layer by layer. E.g., select points, lines, areas, or polygons layer by layer.

Because the coordinate system and scales have default values, one can draw a plot without setting them. You can change the coordinate system or scales if needed.
Stats and facets can be added as desired.

ggplot2_cheat-sheet.pdf

ggplot2 installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just ggplot2:
install.packages("ggplot2")

# Or the development version from GitHub:
install.packages("devtools")
devtools::install_github("tidyverse/ggplot2")
```

Example of making a plot with ggplot2

- Let's make some plots

(see e-class > Documents > ASSIGNMENTS - PROJECTS - TOOLS > R - ggplot2 – shiny)

ggplot2 resources

- ggplot2: Elegant Graphics for Data Analysis by Hadley Wickham, https://ggplot2-book.org/
- R Graphics Cookbook by Winston Chang

- Online documentation: https://docs.ggplot2.org/current

- https://ggplot2.tidyverse.org

plotly

- plotly is an R package to create a variety of interactive graphics
- There are two main ways to creating a plotly object:
  - by transforming a **ggplot2** object (via *ggplotly()*) into a **plotly** object
  - by directly initializing a **plotly** object with *plot_ly() / plot_geo() / plot_mapbox()*

https://plotly-r.com/overview.html

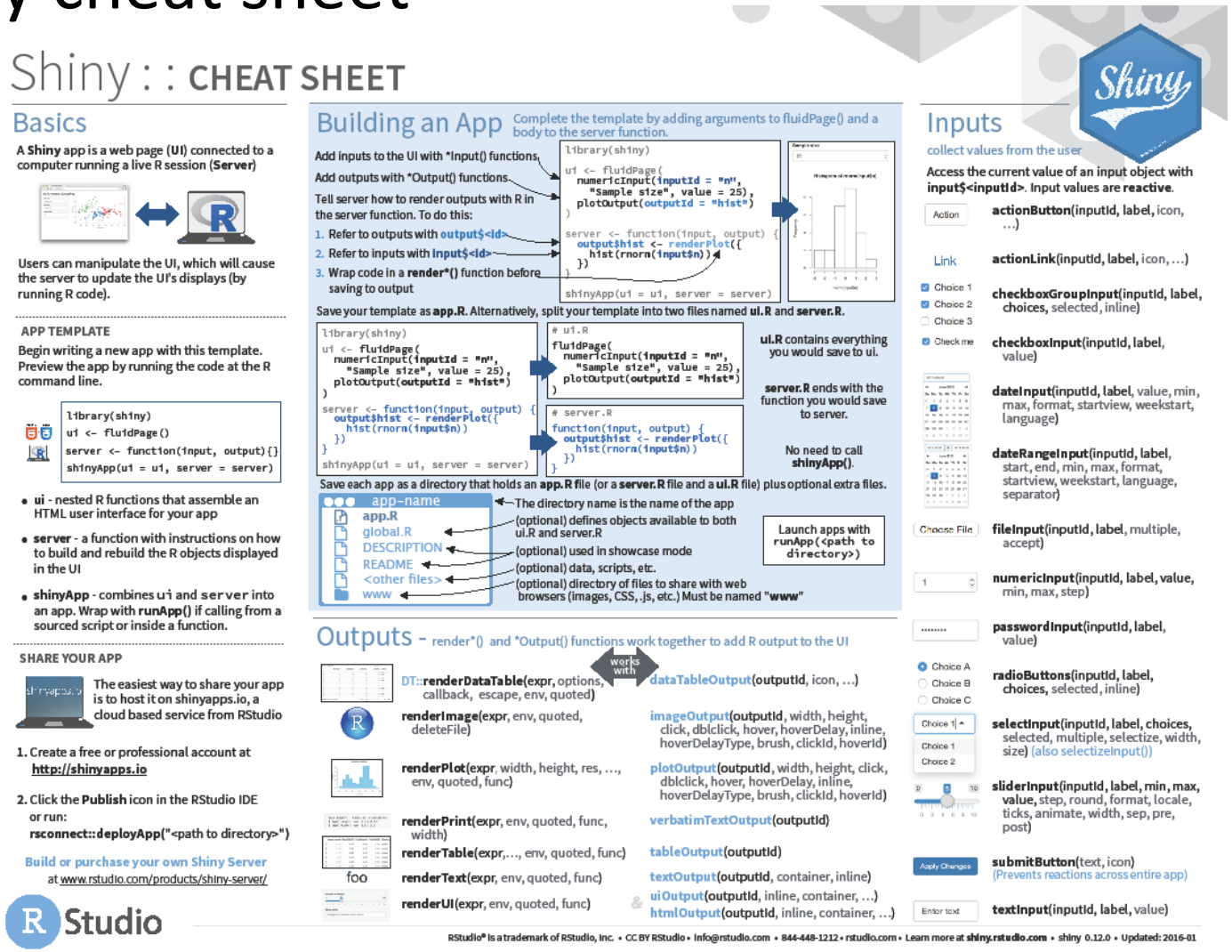
Shiny

- Shiny is merely an R package like dplyr or ggplot2. The package is used to create web-applications, but uses the R language rather than javascript or HTML5, which are traditionally used for web applications.

- By using R, Shiny provides an efficient method of creating web applications designed around data presentation and analysis.

Shiny resources

- Shiny cheat sheet



Shiny : : CHEAT SHEET

Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

APP TEMPLATE

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines `ui` and `server` into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

SHARE YOUR APP

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at http://shinyapps.io
2. Click the **Publish** icon in the RStudio IDE or run: **rsconnect::deployApp("<path to directory>")**

Build or purchase your own Shiny Server at www.rstudio.com/products/shiny-server/

RStudio

Building an App

Complete the template by adding arguments to fluidPage() and a body to the server function.

Add inputs to the UI with *Input() functions.
Add outputs with *Output() functions.
Tell server how to render outputs with R in the server function. To do this:
1. Refer to outputs with `output$<id>`
2. Refer to inputs with `input$<id>`
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

ui.R contains everything you would save to ui.

```
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

server.R ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

app-name
- app.R — The directory name is the name of the app
- global.R — (optional) defines objects available to both ui.R and server.R
- DESCRIPTION — (optional) used in showcase mode
- README — (optional) data, scripts, etc.
- <other files> — (optional) data, scripts, etc.
- www — (optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

Launch apps with `runApp(<path to directory>)`

Outputs - render*() and *Output() functions work together to add R output to the UI

works with

- DT::**renderDataTable**(expr, options, callback, escape, env, quoted) — **dataTableOutput**(outputId, icon, ...)
- **renderImage**(expr, env, quoted, deleteFile) — **imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
- **renderPlot**(expr, width, height, res, ..., env, quoted, func) — **plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
- **renderPrint**(expr, env, quoted, func, width) — **verbatimTextOutput**(outputId)
- **renderTable**(expr,..., env, quoted, func) — **tableOutput**(outputId)
- **renderText**(expr, env, quoted, func) — **textOutput**(outputId, container, inline)
- **renderUI**(expr, env, quoted, func) — & **uiOutput**(outputId, inline, container, ...) **htmlOutput**(outputId, inline, container, ...)

Inputs

collect values from the user

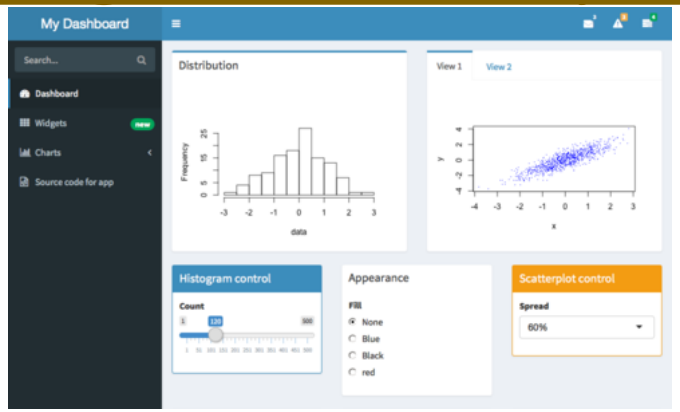
Access the current value of an input object with **input$<inputId>**. Input values are **reactive**.

- Action — **actionButton**(inputId, label, icon, ...)
- Link — **actionLink**(inputId, label, icon, ...)
- Choice 1, Choice 2, Choice 3 — **checkboxGroupInput**(inputId, label, **choices**, selected, inline)
- Check me — **checkboxInput**(inputId, label, value)
- **dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)
- **dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
- Choose File — **fileInput**(inputId, label, multiple, accept)
- 1 — **numericInput**(inputId, label, value, min, max, step)
- ••••••• — **passwordInput**(inputId, label, value)
- Choice A, Choice B, Choice C — **radioButtons**(inputId, label, **choices**, selected, inline)
- Choice 1, Choice 1, Choice 2 — **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())
- **sliderInput**(inputId, label, min, max, **value**, step, round, format, locale, ticks, animate, width, sep, pre, post)
- Apply Changes — **submitButton**(text, icon) (Prevents reactions across entire app)
- Enter text — **textInput**(inputId, label, value)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **shiny.rstudio.com** • shiny 0.12.0 • Updated: 2016-01

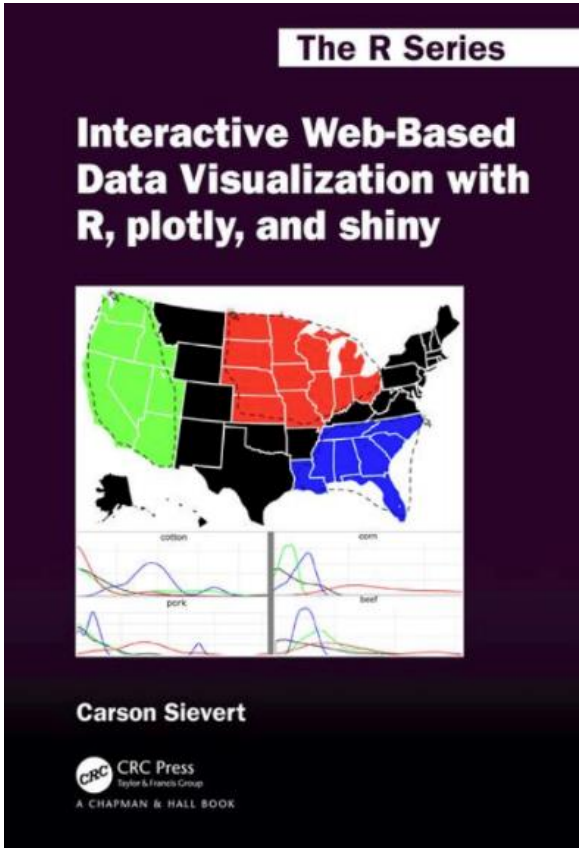
Shiny resources

- Shiny gallery: https://shiny.rstudio.com/gallery/
- 2-3 hour Shiny Tutorial: https://shiny.rstudio.com/tutorial/
- shinyApps.io platform: www.shinyapps.io
- Shinydashboard https://rstudio.github.io/shinydashboard/



Shiny resources

- Interactive web-based data visualization with R, plotly, and shiny: https://plotly-r.com/



Learn ggplot2 using Shiny App

- Moon, K.-W. (2016). *Learn ggplot2 Using Shiny App*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-53019-2

- http://r-graph.com/
- Online app: https://cardiomoon.shinyapps.io/ggplot2new/
- *or*, in Rstudio: shiny::runGitHub("cardiomoon/ggplot2new")

Other R stuff

- Leaflet: an open-source JavaScript library for mobile-friendly interactive maps
  https://leafletjs.com/
- Leaflet R package to integrate and control Leaflet maps in R:
  https://rstudio.github.io/leaflet/

Thank you!

mroussou@di.uoa.gr

http://eclass.uoa.gr/courses/DI411/