IntRo & RefreshR

An Introduction to Data Handling and Analysis using R

Blake Massey and Joe Drake

January 17, 2019

What is R?

Integrated suite of software facilities for data manipulation, calculation, and graphical display.

- R Project

Background

THE UNIVERSITY OF AUCKLAND

Dr. Robert Gentleman

Dr. Ross Ihaka

1993 - Project to extend S Language

1995 - First public release

1997 - Comprehensive R Archive Network (CRAN) started

2000 - R 1.0.0 version

2019 - R 3.5.2 (current version)

Why R?

Cons

Dependent on writing code

Pros

Free, open-source, and cross-platform

Extended via packages (>13,500 on CRAN)

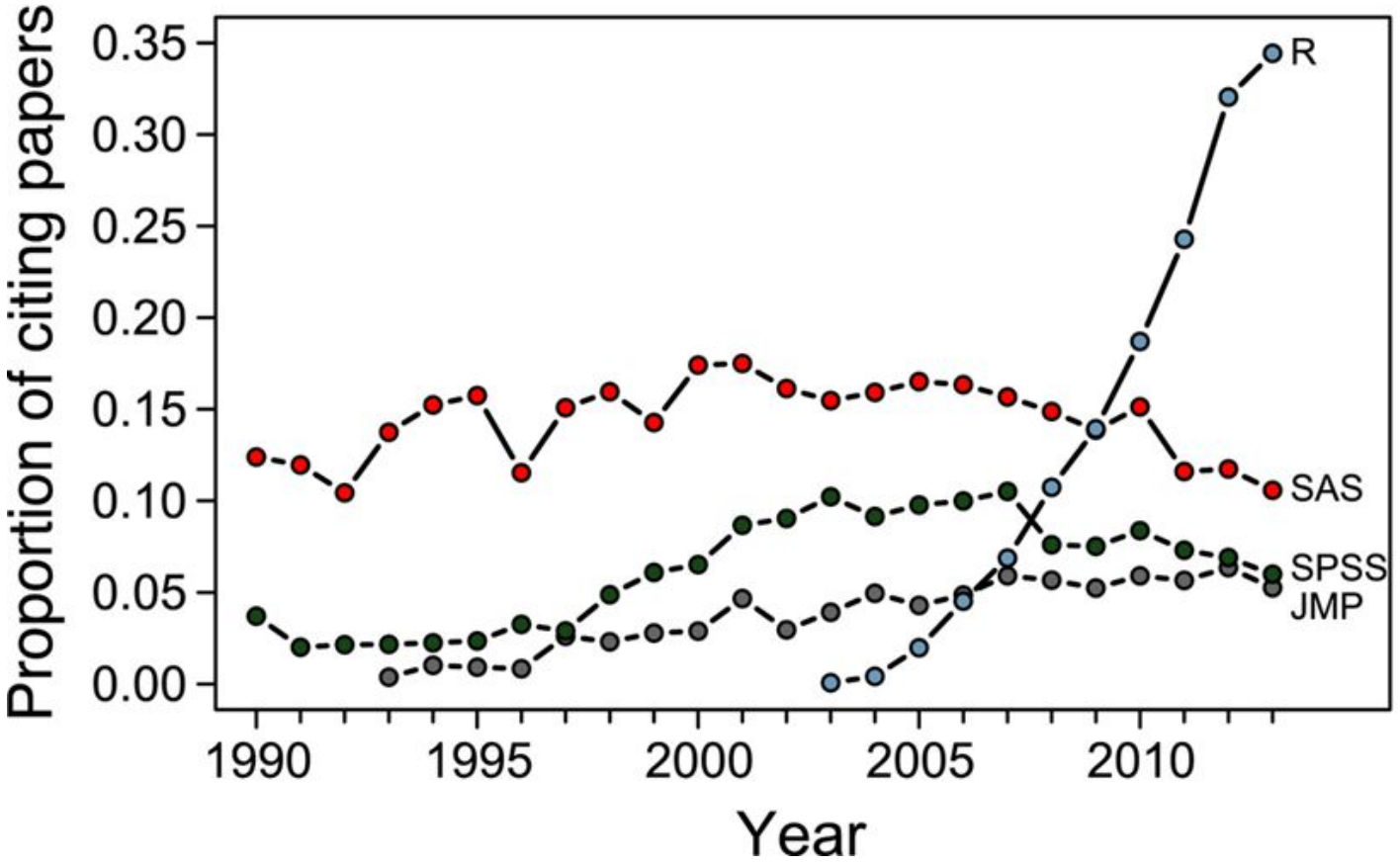
Large community (StackOverflow, Github, etc.)

ECOSPHERE
AN ESA OPEN ACCESS JOURNAL

The mismatch between current statistical practice and doctoral training in ecology

Justin C. Touchon, Michael W. McCoy

First published: 17 August 2016



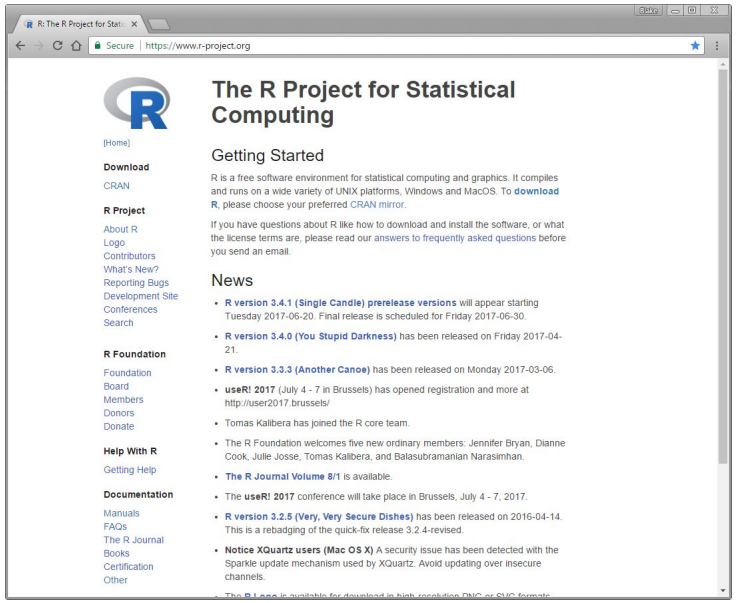
Changes in the usage of four leading statistical programs from 1990 to 2013.

Software: R and RStudio

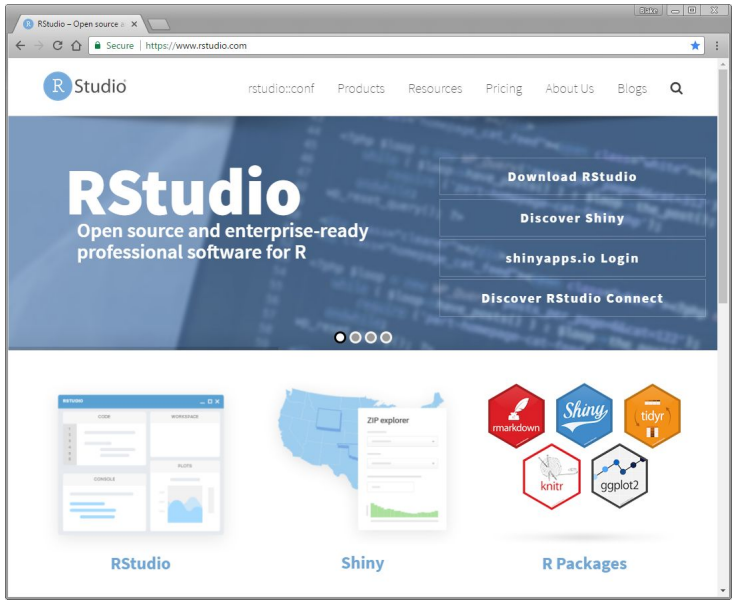
Language and Environment

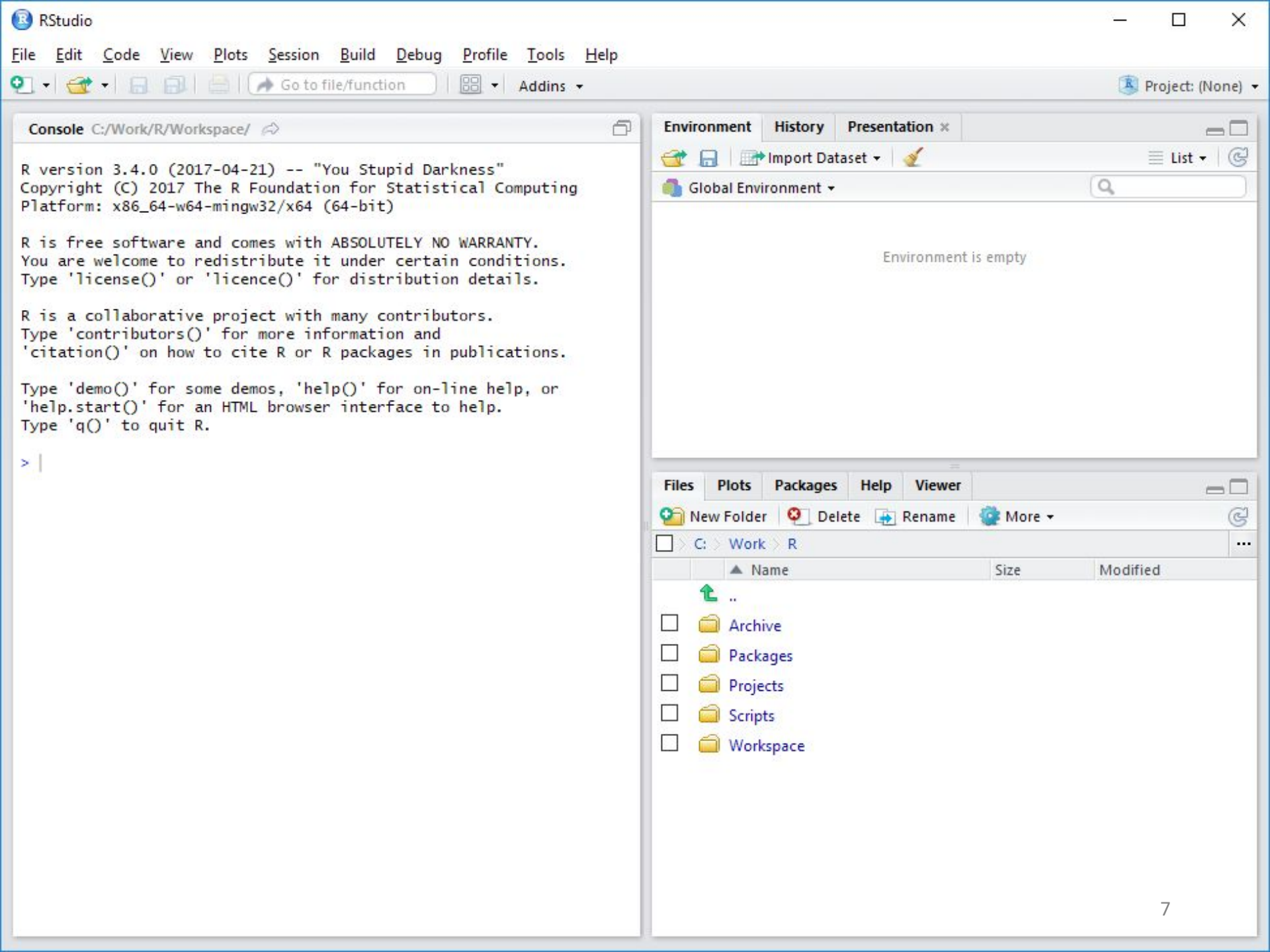
Integrated Development Environment (IDE) for R

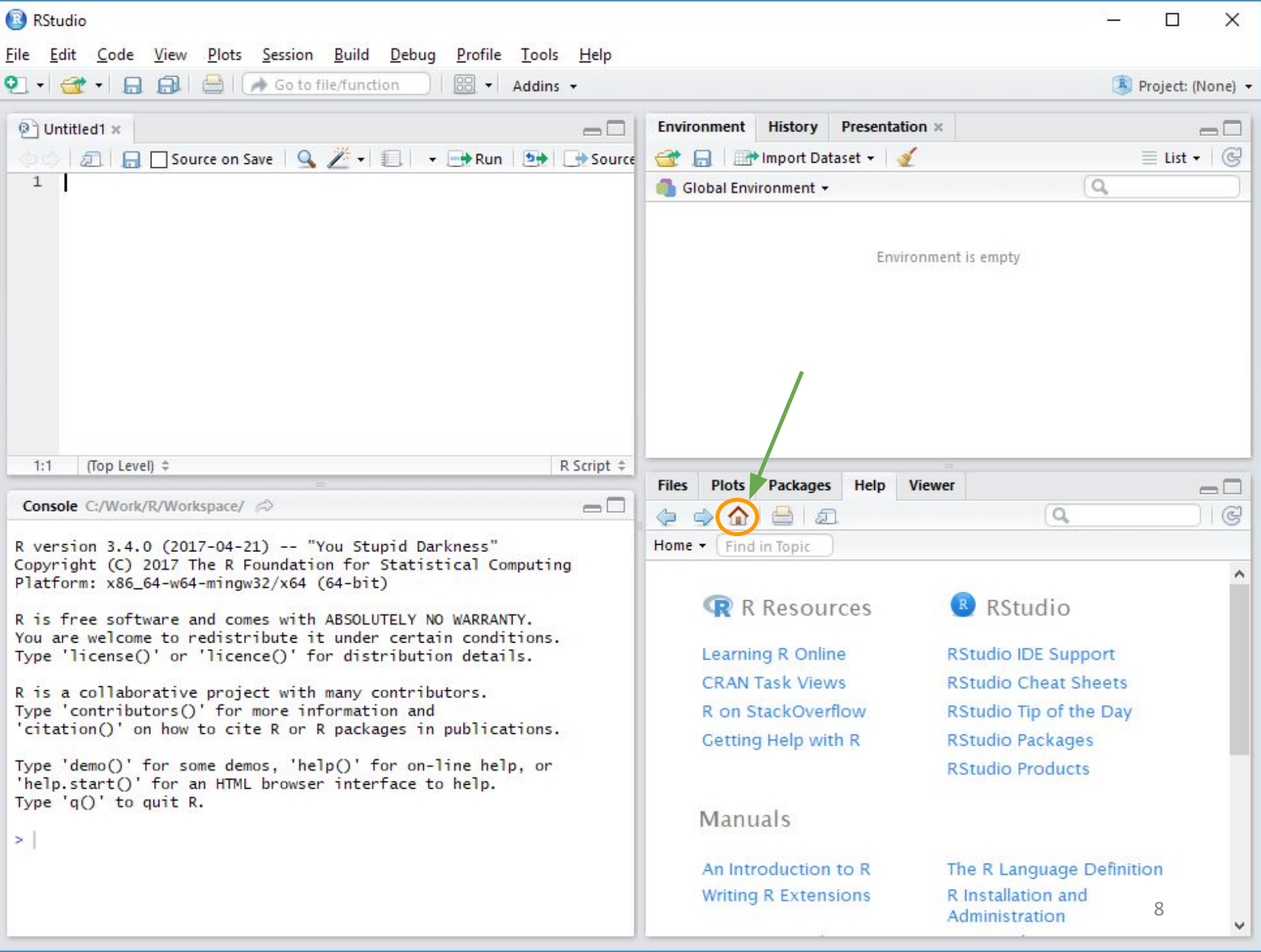
r-project.org



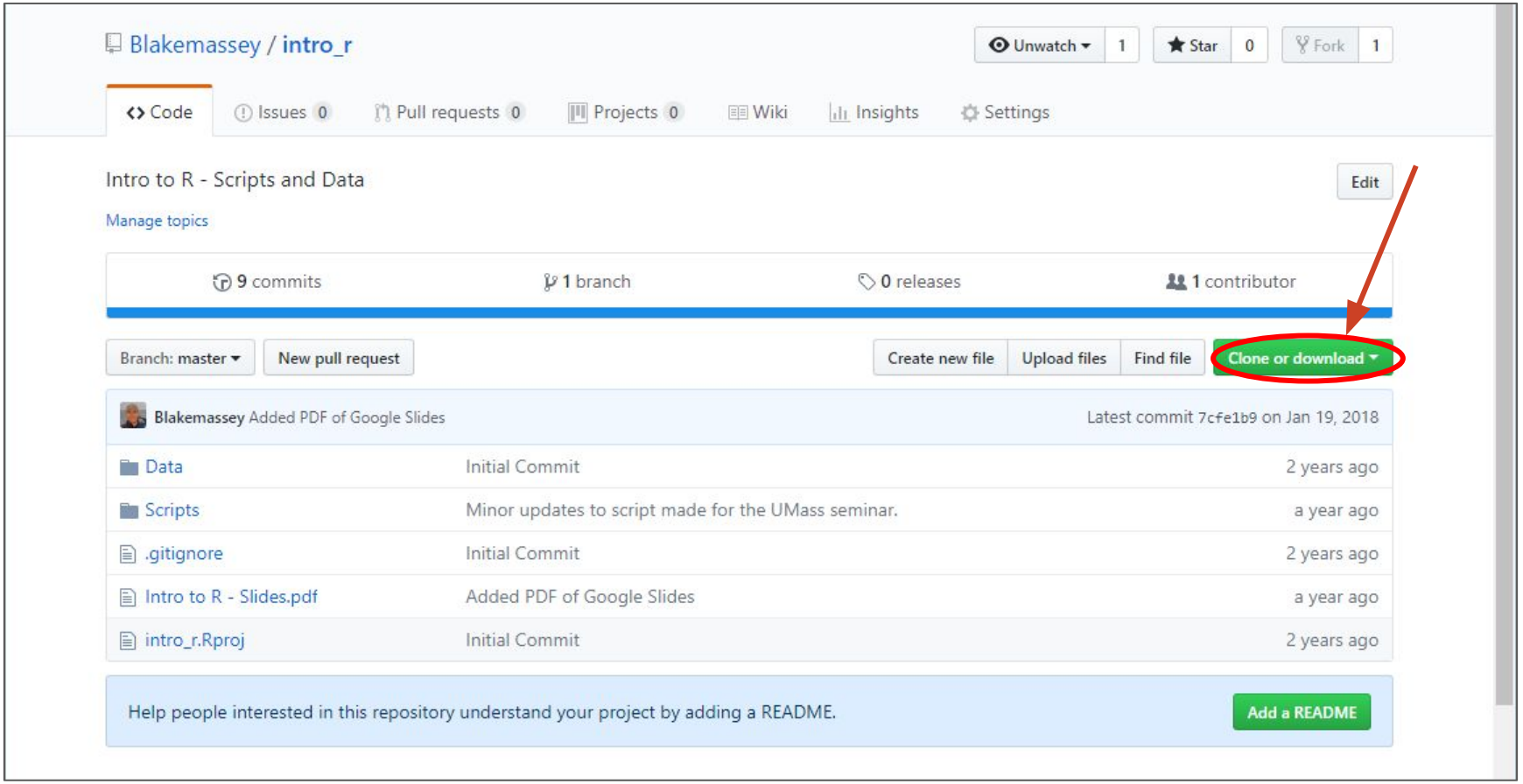
rstudio.com




RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console C:/Work/R/Workspace/
R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment History Presentation
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
New Folder
Delete
Rename
More
C: Work R
Name
Size
Modified
..
Archive
Packages
Projects
Scripts
Workspace


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
Console C:/Work/R/Workspace/
R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment History Presentation
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
Home
Find in Topic
R Resources
Learning R Online
CRAN Task Views
R on StackOverflow
Getting Help with R
RStudio
RStudio IDE Support
RStudio Cheat Sheets
RStudio Tip of the Day
RStudio Packages
RStudio Products
Manuals
An Introduction to R
Writing R Extensions
The R Language Definition
R Installation and Administration

github.com/blakemassey/intro_r



Working Directory

```
# Working Directory
getwd()  # show current working directory
setwd()  # set working directory
```

Windows Users:

Single backslashes in filenames cause problems:

"*C:\Work\myfile.R*"

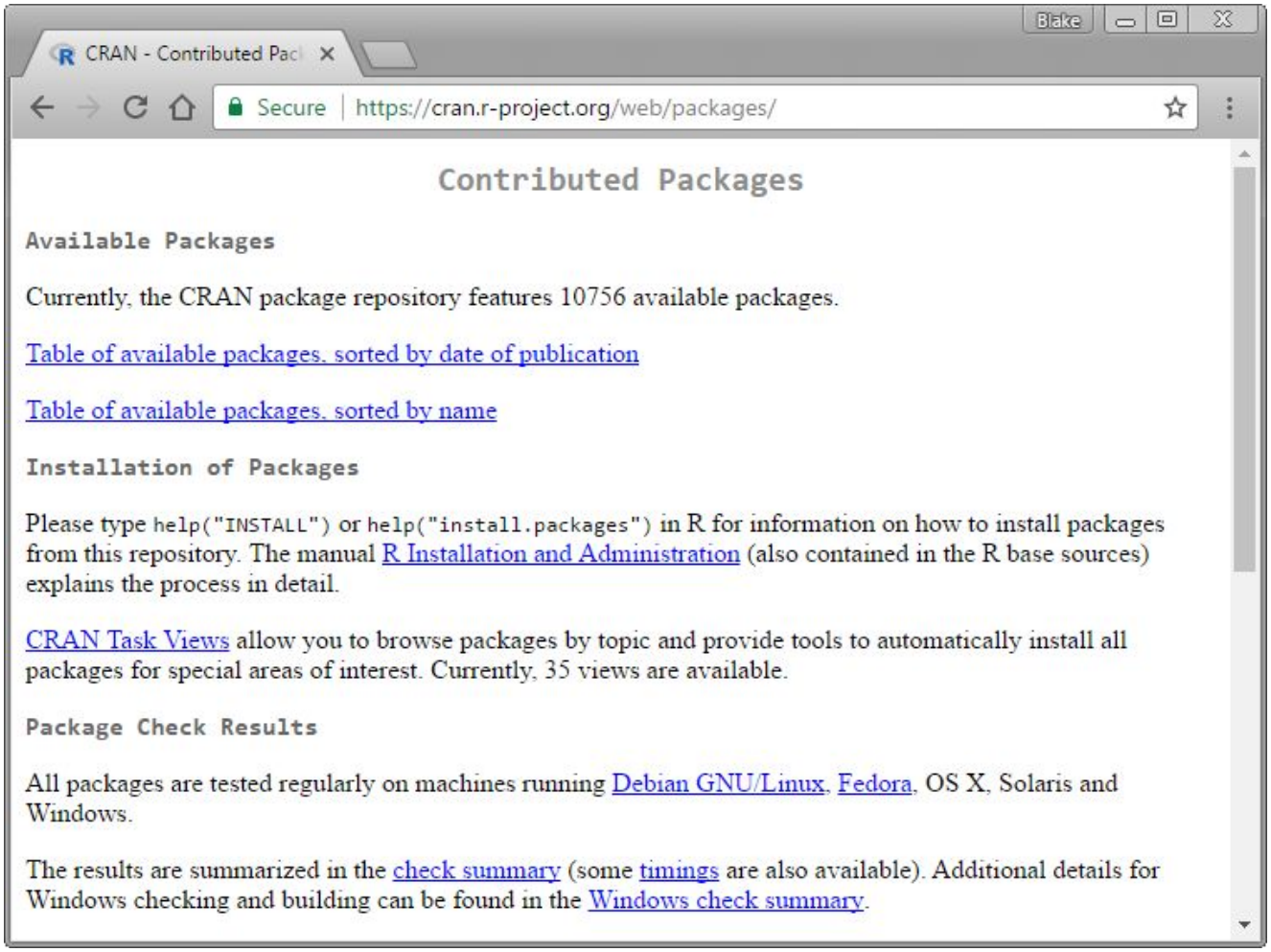
R reads "\" as an escape character. Instead, use:

"*C:\\Work\\myfile.R*"

"*C:/Work/myfile.R*"

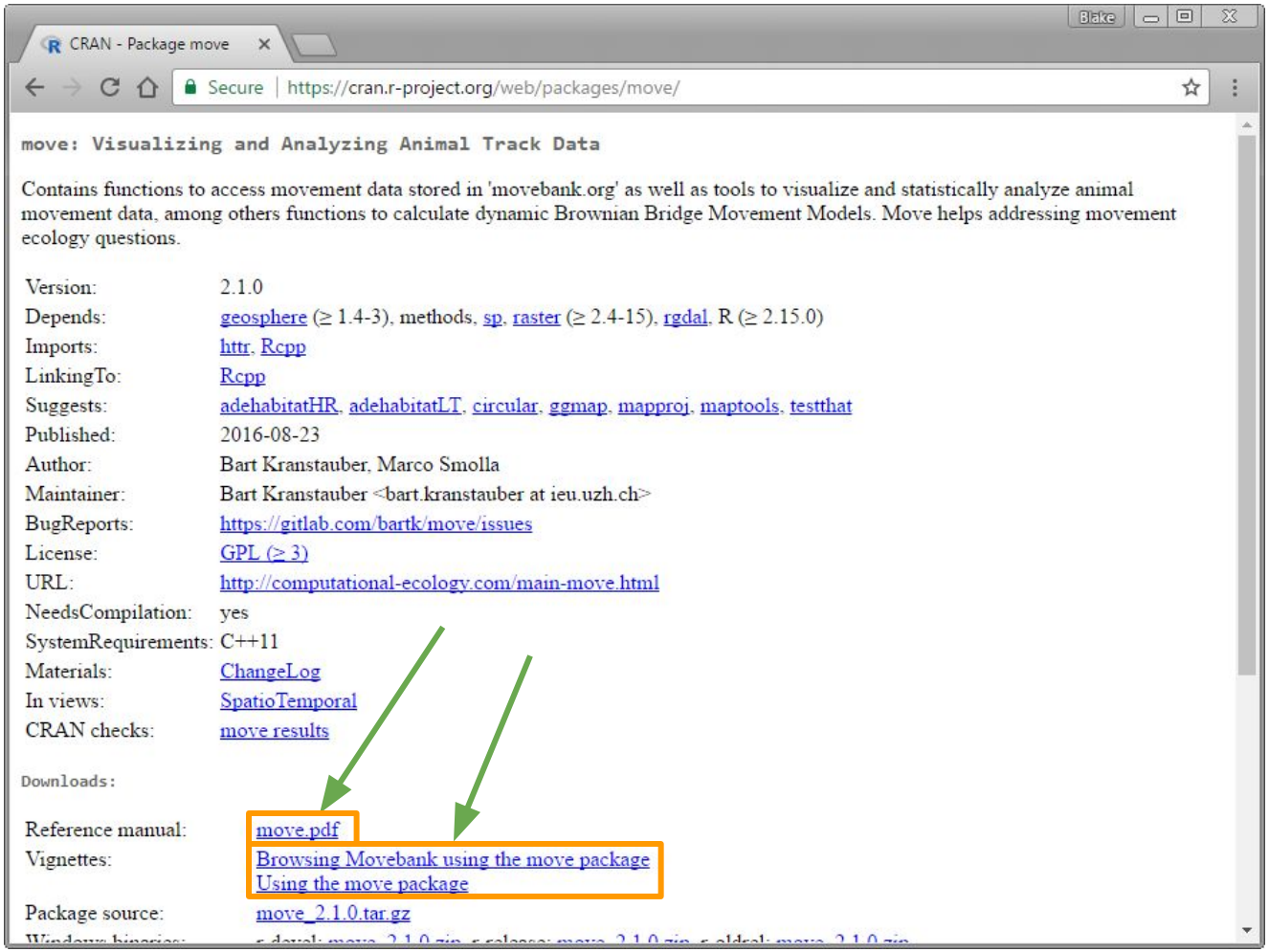
CRAN Packages

cran.r-project.org/web/packages



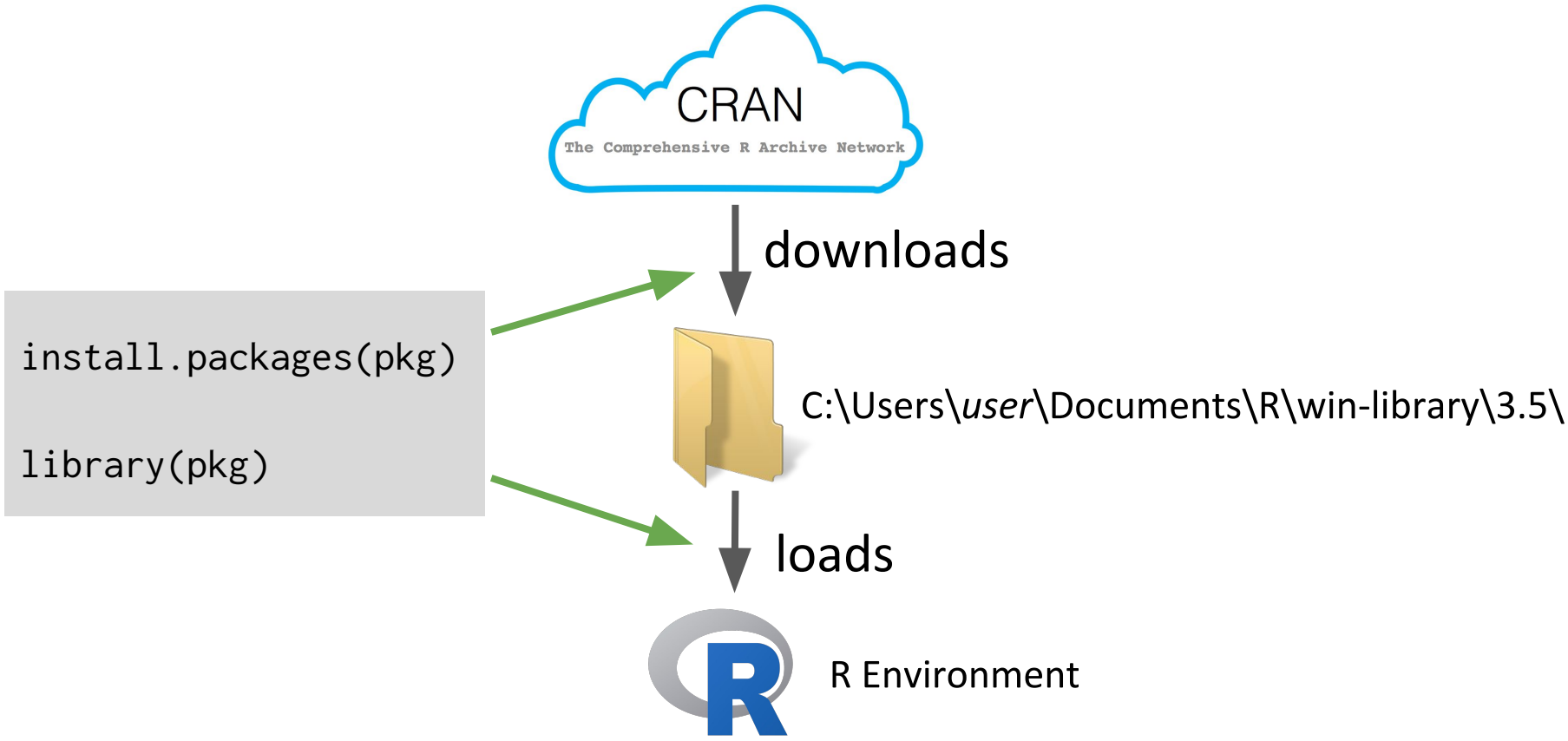
Reference Manuals and Vignettes

cran.r-project.org/web/packages/*pkg*



Packages: Installing and Loading

```
?install.packages() # downloads package
?library()          # loads package into R environment
```



‘Swirl’ package

swirlstats.com

Swirl: Basic Building Blocks

```
install.packages("swirl")
library(swirl)

# Open "R Programming, Lesson 1: Basic Building Blocks"
swirl()

# Open and go through the lesson:
# 1: R Programming: The basics of programming in R
# 1: Basic Building Blocks

# Leave swirl by using the bye() function
bye()
```

To understand computations in R, two slogans are helpful:

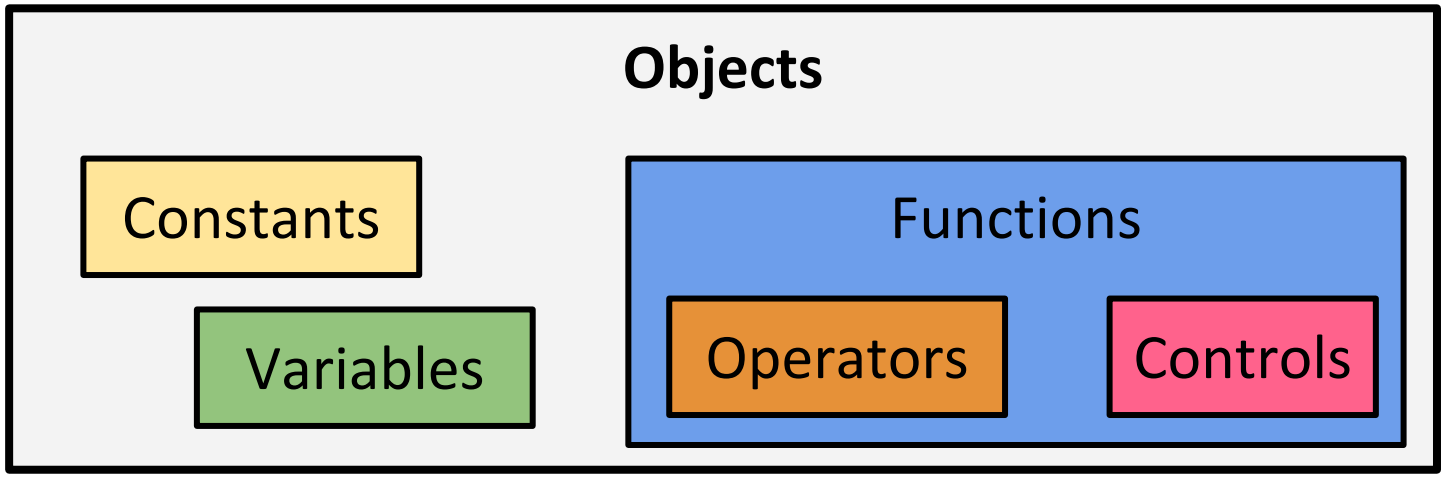
1) Everything that exists is an object.

2) Everything that happens is a function call.

- John Chambers (CRAN Core Team)

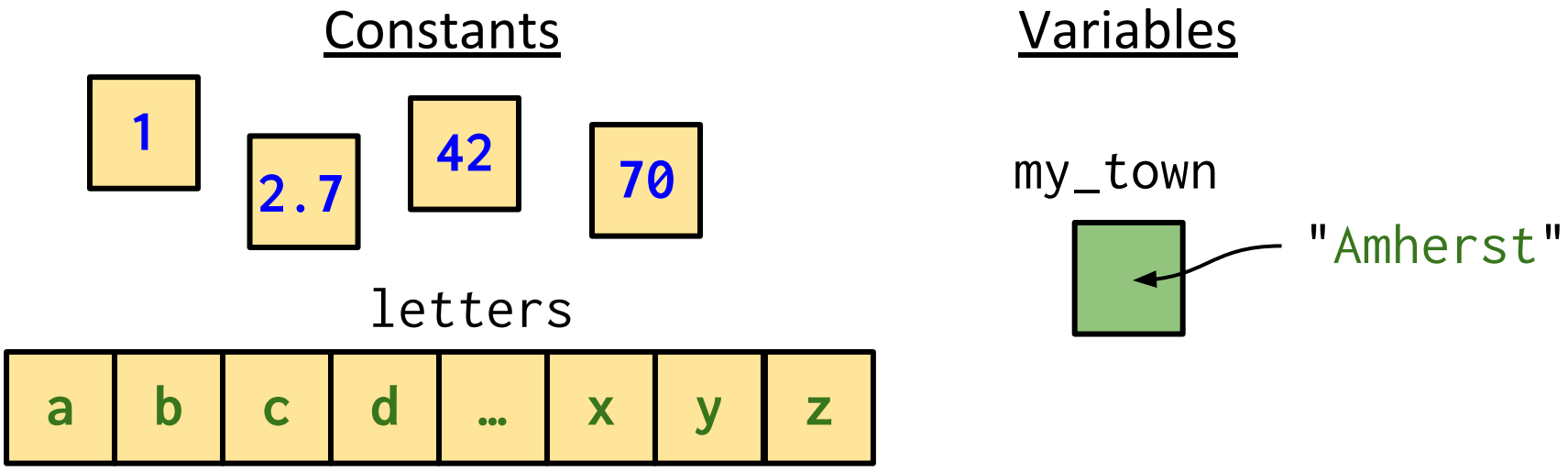
Basic R Syntax

```
# Examples of Basic R Syntax
5             # a constant
my_number     # a variable
sqrt(25) # a function (which is also an object!)
?'<-'         # an operator (a type of function)
?'if'         # a control statement (a type of function)
```



Constants vs Variables

```
# Constants
42          # built-in constants (numeric)
letters     # built-in constants (characters)

# Variables
my_town <- "Amherst"      # variable created by user
```



Basic Data Classes

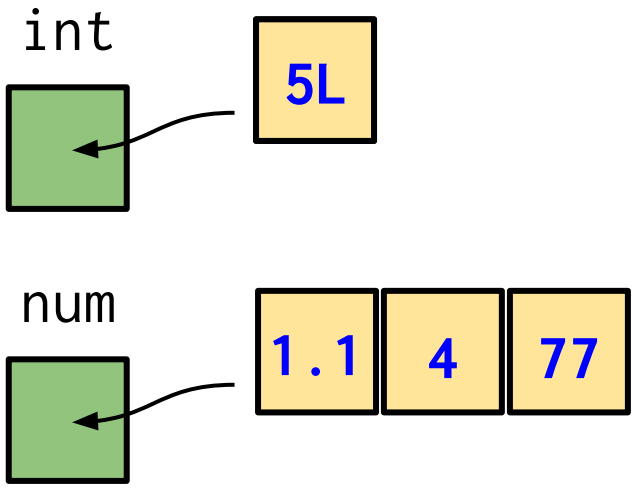
```
# Basic Data Classes (R automatically recognizes these)
?character  # strings
?integer    # whole numbers
?numeric    # real numbers; aka "double"
?logical    # TRUE or FALSE
?complex    # complex numbers; i.e., (n + ni)
?raw        # raw byte value

# Special Values
TRUE    # true
FALSE   # false
NA      # not available
NaN     # not a number
NULL    # list of zero length (an empty list)
Inf     # infinity
-Inf    # negative infinity
```

Built-in Constants

```
# Built-in Constants (Numeric)
1.3             # numeric sequence with or without a decimal
5L              # L suffix means coerce to integer class
2.1e-4          # numeric seq, followed by e or E, then n or -n
pi              # 3.14159…

# Built-in Constants (Characters)
letters         # 26 lowercase letters
LETTERS         # 26 uppercase letters
month.name      # Month names full
month.abb       # Month names abbreviations (3 letters)
```

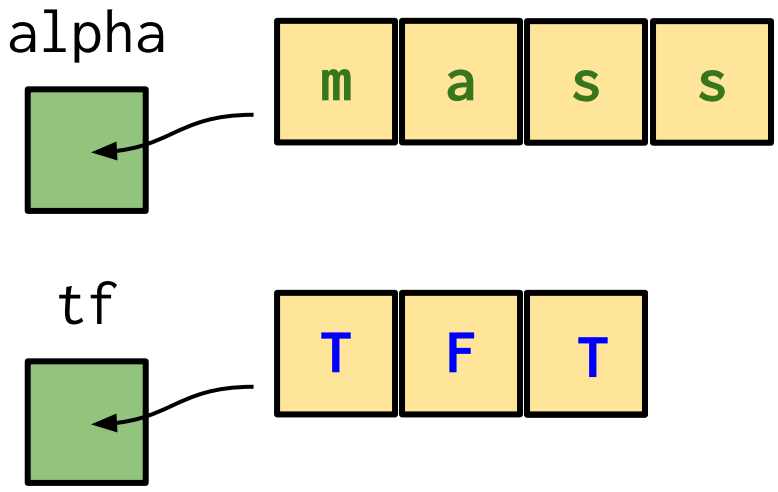
Creating Variables

```
int <- 5L     # an L suffix coerces numeric elements to integer
class(int)
num <- c(1.1, 4, 77)   # 'c' is function that combines elements
class(num)
```

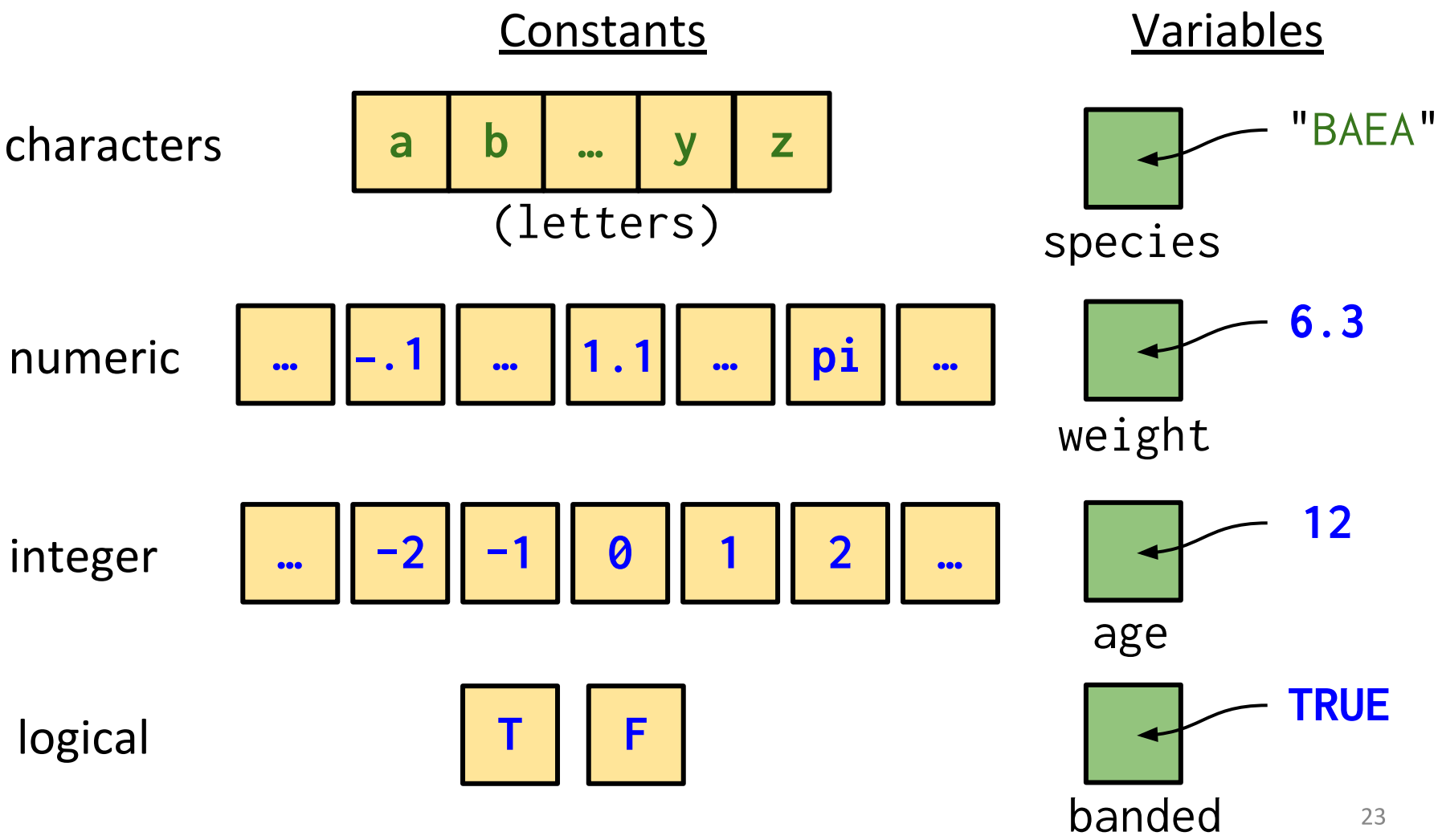


Creating Variables

```
alpha <- c("m", "a", "s", "s")
class(alpha)
tf <- c(TRUE, FALSE, TRUE)  # notice no quotes
class(tf)
```



Constants/Variables and Classes



Atomic Vectors

```
# Atomic Vectors
my_log <- c(TRUE, FALSE, FALSE)              # logical
my_int <- c(1L, 6L, 17L)                     # integer
my_num <- c(1.4, 3.15, 17.082)               # numeric
my_chr <- c("Amherst", "Site 27", "Bombus") # character
```

All elements of an atomic vector must be the same type.

Atomic Vector Coercion

```
# Atomic Vectors
c(TRUE, FALSE, FALSE, 6L)                # integer
c(1L, 6L, 17L, 37.621)                   # numeric
c(1.4, 3.15, 17.082, "Summer")           # character
c("Amherst", "Site 27", "Bombus", TRUE)  # character
```

When you attempt to combine different types they will be coerced to the most flexible type.

Flexibility: logical < integer < numeric < character

Classes and NA Tests

```
# Tests for Classes
is.logical(c(TRUE, FALSE, FALSE))
is.integer(c(1L, 6L, 17L))
is.numeric(c(1.4, 3.15, 17.082))
is.character(c("Amherst", "Site 27", "Bombus"))

# Tests for NA
is.na(c(TRUE, FALSE, FALSE))
is.na(c(1L, 6L, FALSE, 17L))
```

Date and Time Classes

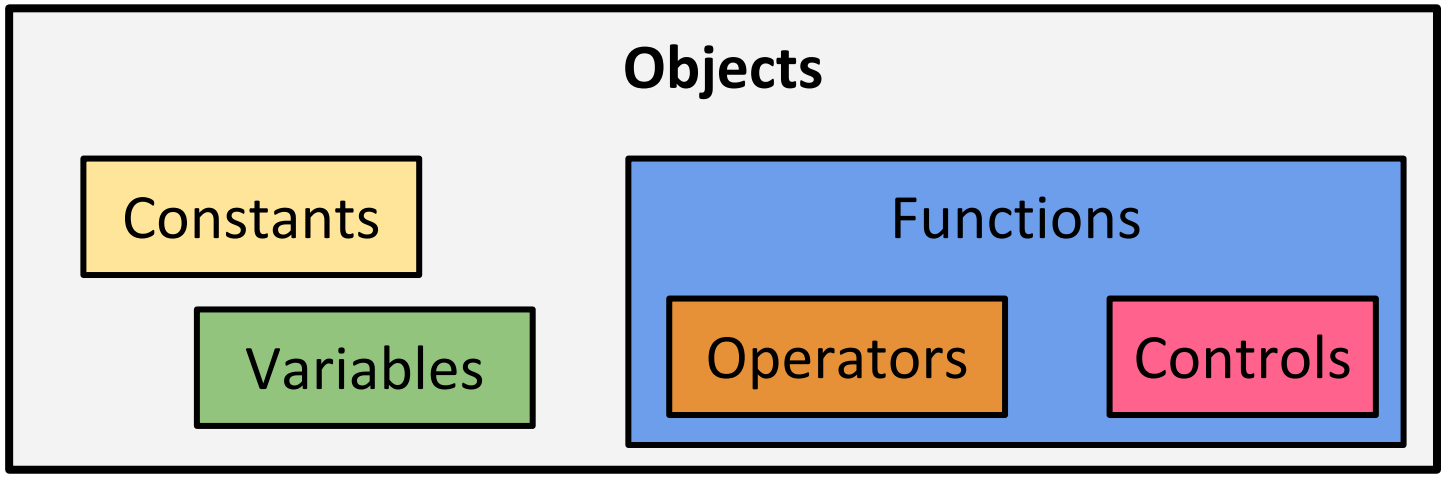
```
# Date Classes
as.Date("1995-02-16")
as.POSIXct("1995-02-16 12:45:30", tz = "GMT")

now <- Sys.time()
class(now)
```

"Portable Operating System Interface" *calendar time*

Excellent package for date/time operations: 'lubridate'

Functions

```
# R Base Functions
builtins()      # list all built-in objects (mostly functions!)
class(print)    # function class is "function"
class(`+`)      # operator class is "function"
class(`if`)     # control class is "function"
```



Basic Operators (Arithmetic)

```
# Unary versus binary operations
 -3     # Unary operations have one operand
8 - 4   # Binary operations have two operands

# Arithmetic Operators Examples
 +3     # positive (unary)
 -7     # negative (unary)
2 + 4   # addition
7 - 2   # subtraction
9 / 3   # division
2 * 6   # multiplication
5 ^ 2   # exponentiation
```

Basic Operators (Logical)

```
# Logical Operators (Return TRUE or FALSE)
?`<`     # less greater than
?`>`     # less greater than
?`<=`    # less than or equal to
?`>=`    # greater than or equal to
?`==`    # equal to
?`!=`    # not equal to
?`&`     # and (vectorized)
?`&&`    # and (non-vectorized)
?`|`     # or (vectorized)
?`||`    # or (non-vectorized)
?`%in%`  # match
```

Basic Operators

```
# Special Operators
?`!`      # negation
?`~`      # used for model formulae
?`%%`     # special operators (infix type)
```

Basic Operators (Logical)

```
# Logical Operators Examples
3 < 5
7 > 4
7 <= 7
7 >= 7
3 == 3L
4 != 5
TRUE & TRUE
FALSE && c(TRUE, FALSE, TRUE)
TRUE | FALSE
FALSE || c(TRUE, FALSE, TRUE)
"g" %in% c("e", "f", "g")
!"z" %in% c("e", "f", "g")
```

Basic Operators (Assignment)

```
# Assignment/Creation Operators
?`:`     # create sequence
?`<-`    # assign in current environment
?`<<-`   # assign in parent environment
?`=`     # assign (use inside of function arguments)

# Assignment/Creation Operators
3:7                # create sequence
my_var <- "g"      # assign in current environment
glob_var <<- 5.12  # assign in parent environment
list(x = 1:4)      # assign (use inside of function arguments)
```

Basic Operators (Assignment)

```
# Indexing/Extraction Operators
?`[`      # index (returns list)
?`[[`     # index (returns list component)
?`$`      # extract named column or component
?`::`     # use a particular function from a package

# Create a list
lt <- list(x = 1:4, "a", c(TRUE, FALSE, TRUE), c(2.3, 5.9))

# Indexing/Extraction Operators
lt[1]            # index (returns list)
Lt[[1]]          # index (returns list component)
lt$x             # extract named column or component
stats::ecdf      # use a particular function from a package
```

Operators as Functions

```
# Example of Infix Operator and Matching Function
6 + 2      # infix operator
`+`(6, 2)  # function, need backticks around name

# Example of Infix Operator and Matching Function
4:9        # infix operator
`:`(4, 9)  # function, need backticks around name

# Example of Infix Operator and Matching Function
var_1 <- "Amherst"      # infix operator
`<-`(var_1, "Amherst")  # function, need backticks around name
var_1
```

Basic Functions

```
# Example of Function Arguments
round(6.248)              # press tab to see function's arguments
round(6.248, digits = 0) # equivalent to previous line
round(6.248, digits = 2) # rounds to 2 decimal places

# Examples of Arguments
?print        # x = R object; "..." = additional arguments
?length       # x = R object
?rep.int      # x = vector; times = number of times to repeat
?Sys.Date()   # no arguments
```

Basic Mathematical Functions

```
# Mathematical Functions
?abs        # absolute value
?log        # log to base e
?exp        # antilog
?log        # log to base n
?log10      # log to base 10
?sqrt       # square root
?cos        # cosine of x in radians; also sin(x), tan(x)
?factorial  # x!
?floor      # greatest integer less than x
?ceiling    # smallest integer greater than x
?trunc      # closest integer to x between x and 0
?round      # round the value of x to an integer
?signif     # give x to 6 digits scientific notation
```

Basic Math Examples

```
# Mathematical Functions (Examples)
abs(-23)         # 23
log(5)           # 1.609438
exp(4)           # 54.59815
log(6, 2)        # 2.584963
log10(7)         # 0.845098
sqrt(25)         # 5
cos(pi)          # -1
factorial(4)     # 24
floor(6.23)      # 6
ceiling(9.19)    # 10
trunc(-3.2)      # -3
round(3.2178, digits=2)   # 3.22
signif(4/3, digits=6)     # 1.33333
```

Options

```
# Show Current Options
options()             # prints all option settings

# Change Options
options(digits = 10)  # default was 6
5 / 3                 # 10 digits shown
```

Vector Sequences

```
# Vectors of sequences
c(2, 4, 6)
2:6
seq(from = 2, to = 3, by = 0.5)
rep(1:2, times = 3)
rep(1:2, each = 3)
```

```
# Open "R Programming, Lesson 3: Sequence of Numbers"
swirl()

# Open and go through the lesson:
# 1: R Programming: The basics of programming in R
# 3: Sequence of Numbers

# Leave swirl by using the bye() function
bye()
```

Control Statements

```
# Control Statements
?'if'       # if(cond) expr
?'else'     # if(cond) expr1  else  expr2
?'for'      # for(var in seq) expr
?'while'    # while(cond) expr
?'repeat'   # repeat expr
?'break'    # break expr
?'next'     # repeat expr
```

Control the flow of execution of a script

Indexing By Logic

```
# Create sample
ints <- sample(1:10, size = 10, replace = FALSE)

# Create Logical Vectors
ints > 5
ints < 7

# Determine Vector Positions Using Logical Vectors
which(ints > 5)
which(ints < 7)

# Test for Any or All in Logical Vectors
any(ints < 0)
all(ints > 0)
```

Indexing By Logic

```
# Create vector
x <- 6:10

# Equal to or not equal to
x[x == 7]
x[x != 7]
x[x == 6 | x == 8]

# Greater/less than
x[x > 7]
x[x < 8]
x[x > 7 & x < 9]
```

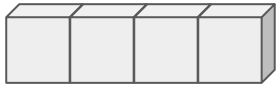
Subsetting Vectors

```
# Open "R Programming, Lesson 6: Subsetting Vectors"
swirl()

# Open and go through the lesson:
# 1: R Programming: The basics of programming in R
# 6: Subsetting Vectors

# Leave swirl by using the bye() function
bye()
```

Data Structures

[pos]

Vector

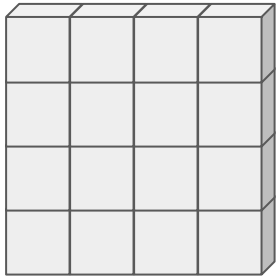


List

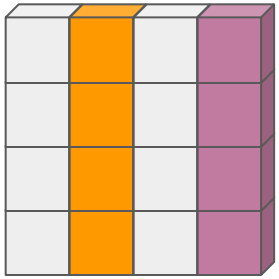
Vectors
Data frames
Arrays
Lists

[row, col]

Matrix



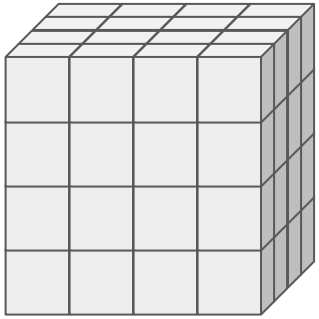
Data frame



Columns can be different data types.

[row, col, slice]

Array

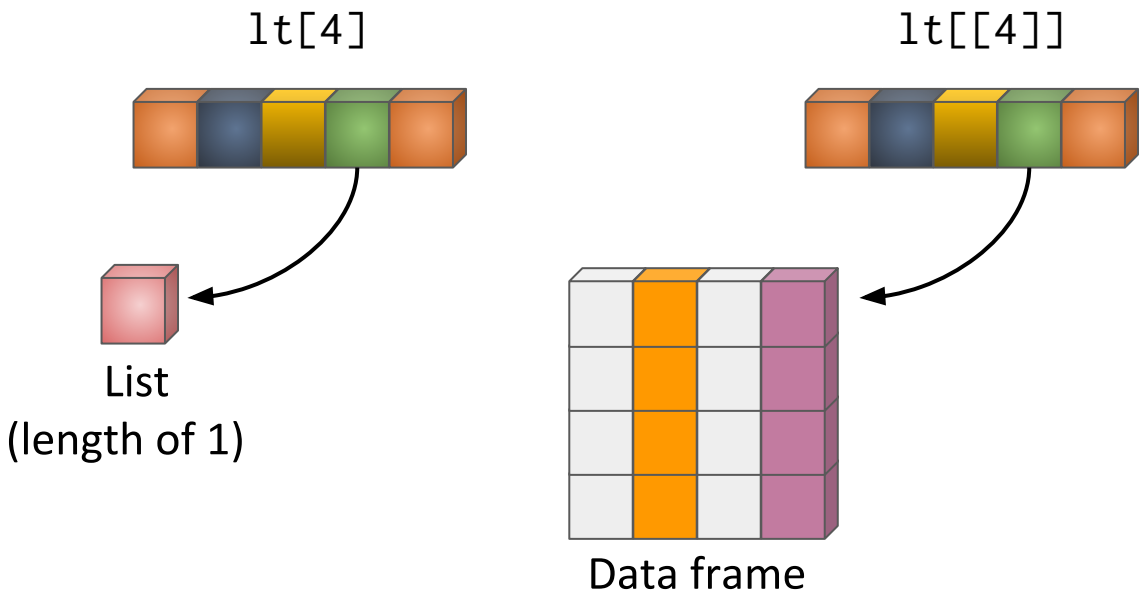


Creating Objects

```r
# Create a vec
vec <- seq(from = 4, to = 16, by = 4)

# Create a matrix
mt <- matrix(1:16, nrow=4)

# Create a data frame
df <- data.frame(x = c(4:7), y = c(letters[1:4]), x2 = c(11:14),
    z = c(TRUE, FALSE, FALSE, TRUE))

# Create a list
lt <- list(1:3, "a", c(TRUE, FALSE, TRUE), df, c(2.3, 5.9))
```

Indexing By Position

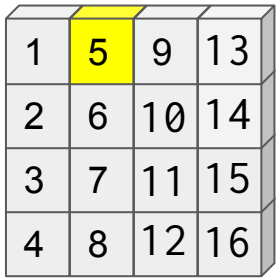
```
# Lists
lt <- list(1:3, "a", c(TRUE, FALSE, TRUE), df, c(2.3, 5.9))
lt[4]        # single bracket returns a list
lt[[4]]      # double bracket returns list components
```



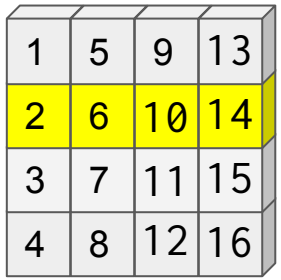
Indexing By Position

```
# Matrices / Data frames
mt <- matrix(1:16, nrow = 4)
mt[1, 2]    # returns element
mt[2, ]     # returns row
mt[, 3]     # returns column
```

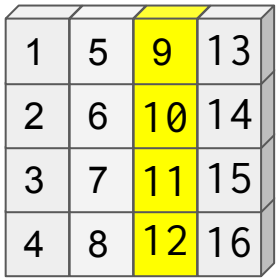
mt[1,2]



mt[2,]

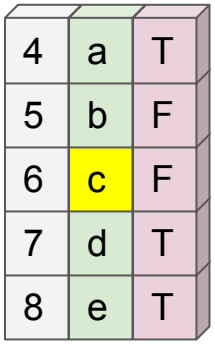


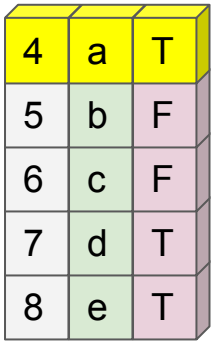
mt[, 3]

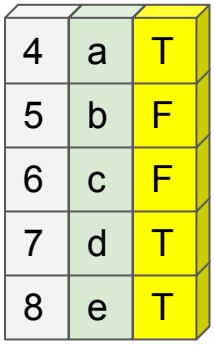


Indexing By Position

```
# Names: Data frames
df <- data.frame(x = c(4:8), y = c(letters[1:5]), z = c(TRUE,
  FALSE, FALSE, TRUE, TRUE))
df[3, 2]    # returns element
df[1, ]     # returns row
df[, 3]     # returns column
```

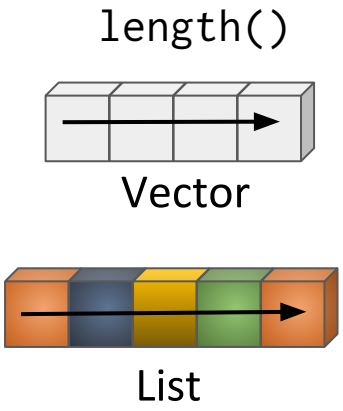


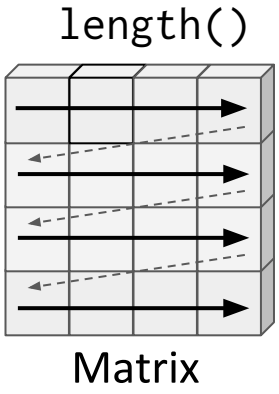


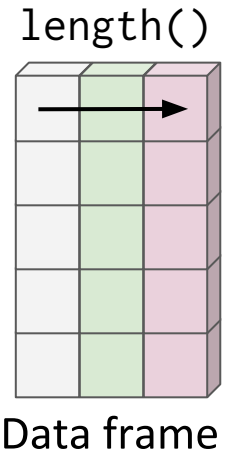


Object Attributes: length()

```
# Object Attributes
length(vec) # number of elements in vector
length(lt)  # number of elements in list
length(mt)  # number of elements in matrix
length(df)  # number of columns in data frame
```

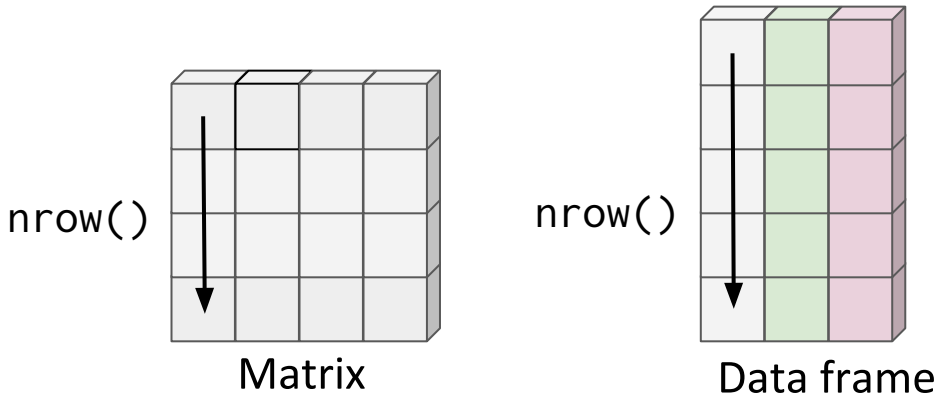






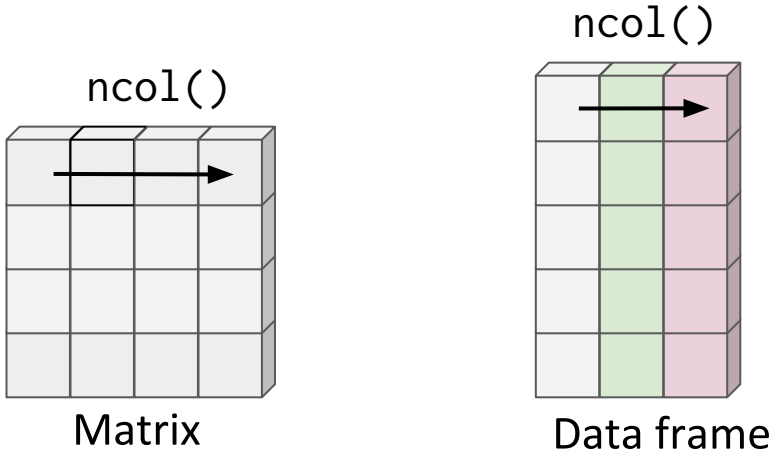
Object Attributes: nrow()

```
# Object Attributes
nrow(mt)    # number of rows in matrix
nrow(df)    # number of rows in data frame
```



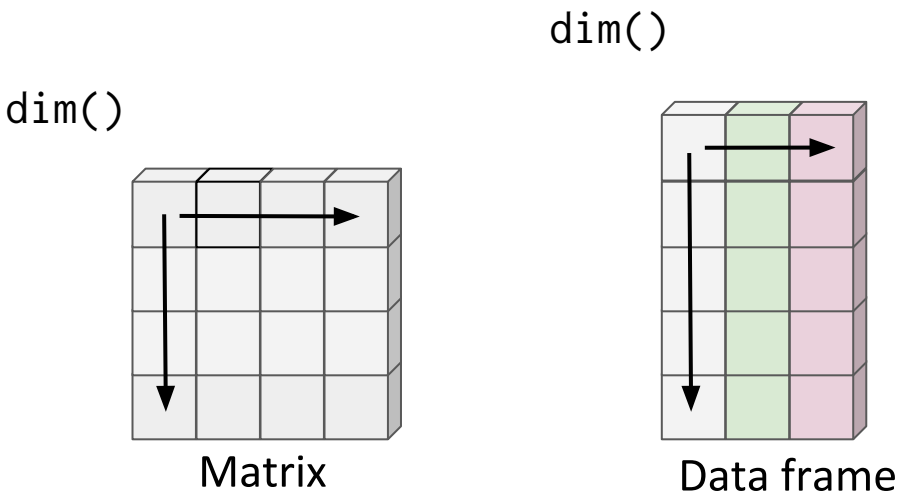
Object Attributes: ncol()

```
# Object Attributes
ncol(mt)    # number of columns in matrix
ncol(df)    # number of columns in data frame
```



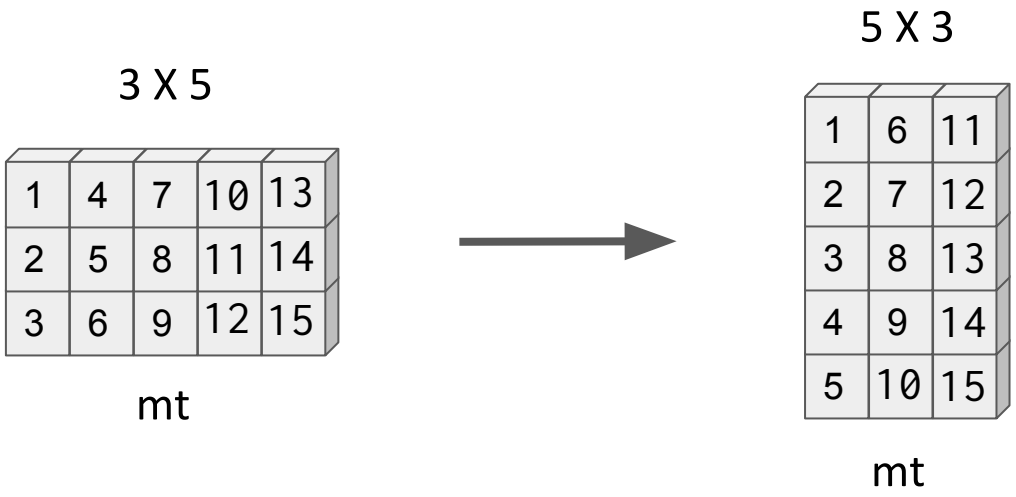
Object Attributes: dim()

```
# Attributes: dim()
dim(mt)    # number of rows, columns in matrix
dim(df)    # number of rows, columns in data frame
```



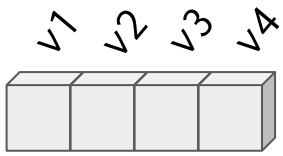
Object Attributes: dim()

```
# Change Matrix Dimensions
mt <- matrix(1:15, nrow = 3, ncol = 5)  # create matrix
dim(mt) <- c(5, 3)  # change dimensions
```

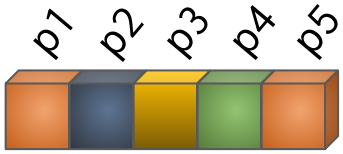


Vector and List Names

```
# Names: Vector and List
vec <- 5:8   # vector
names(vec) <- c("v1", "v2", "v3", "v4")  # set names

lt <- list(1:3, "a", c(TRUE, FALSE, TRUE), vec, c(2.3, 5.9)) # list
names(lt) <- c("p1","p2", "p3", "p4", "p5")  # set names
```



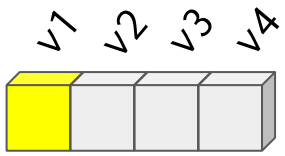
vec



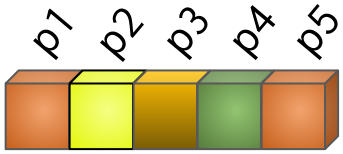
lt

Vector and List Names

```
# Indexing by Name: Vectors and Lists
vec["v1"]   # Index vector using name in brackets
lt["p1"]    # Index list using name in brackets
lt$p2       # Index list using $name
```



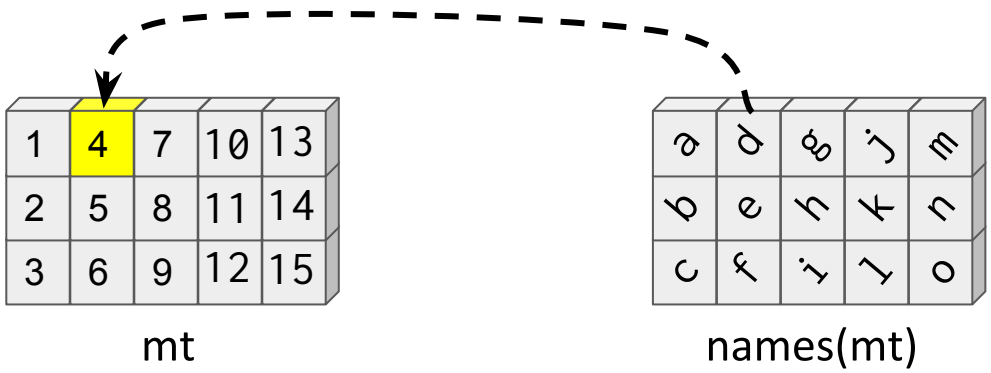
vec



lt

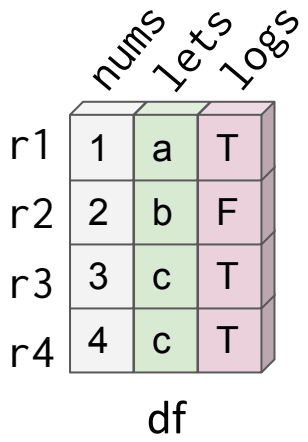
Matrix Names

```
# Names: Matrices
mt <- matrix(1:15, nrow = 3, ncol = 5)  # create matrix
names(mt) <- letters[1:15]              # set names of matrix
mt["d"]                                 # locate element by name
```



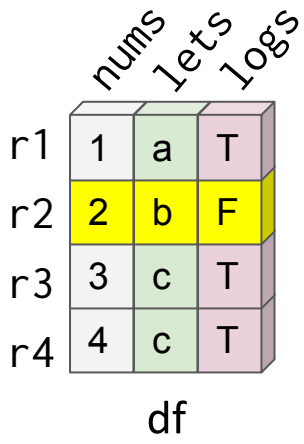
Data Frame Names

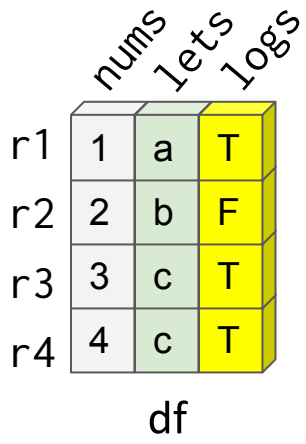
```
# Names: Data frames
df <- data.frame(nums = c(1:4), lets = c("a", "b", "c", "c"),
  logs = c(T, F, T, T))                    # create 3-col data frame
rownames(df) <- c("r1", "r2", "r3", "r4")  # set row names
```

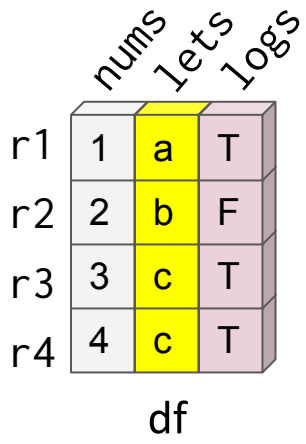


Data Frame Names

```
# Names: Data frames
df["r2",]
df[,"logs"]
df$lets
```

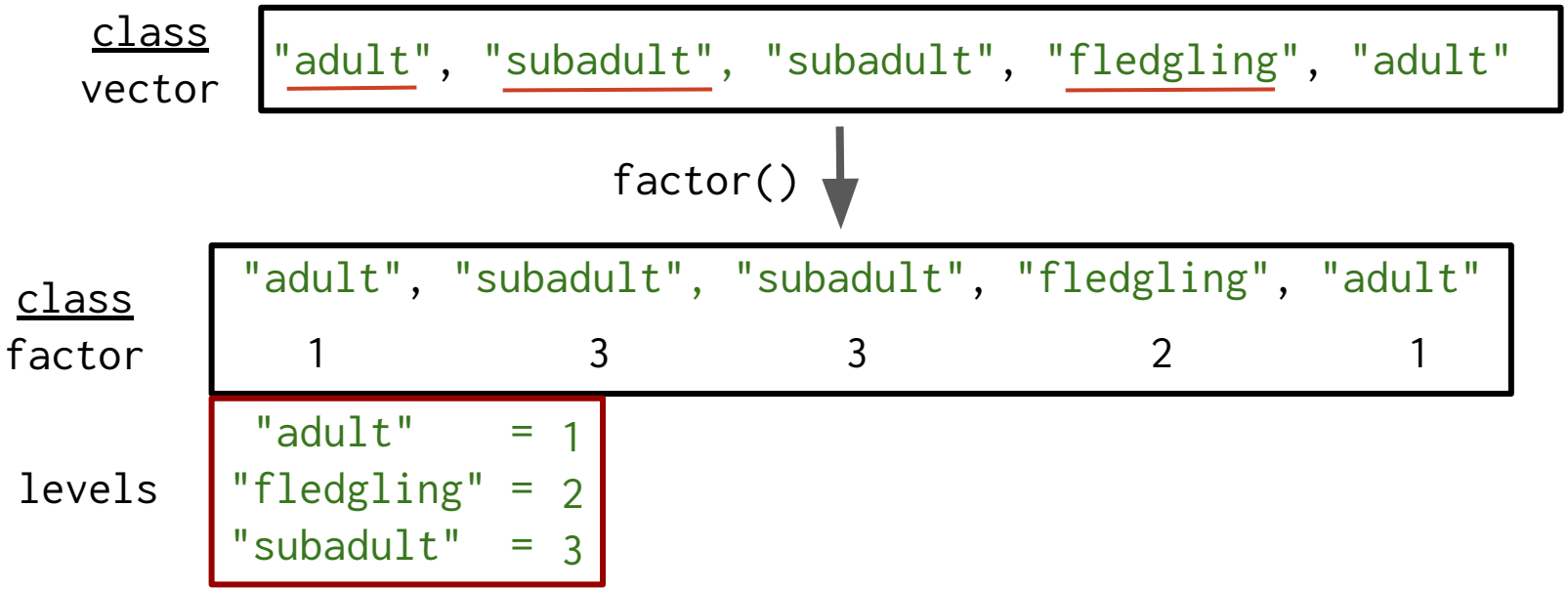






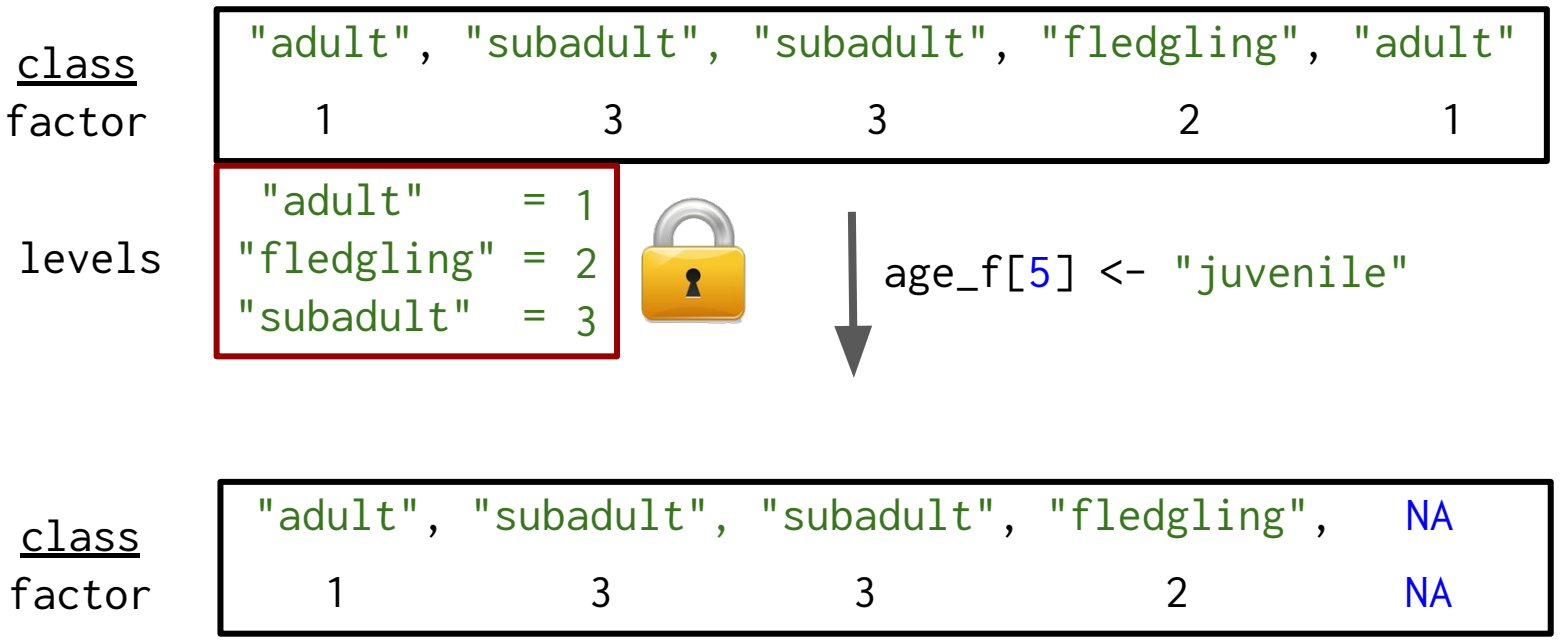
Factor and Levels

```
# Factor and Levels
age <- c("adult", "subadult", "subadult", "fledgling", "adult")
age_f <- factor(age)  # make into class Factor
levels(age_f)         # print levels
```



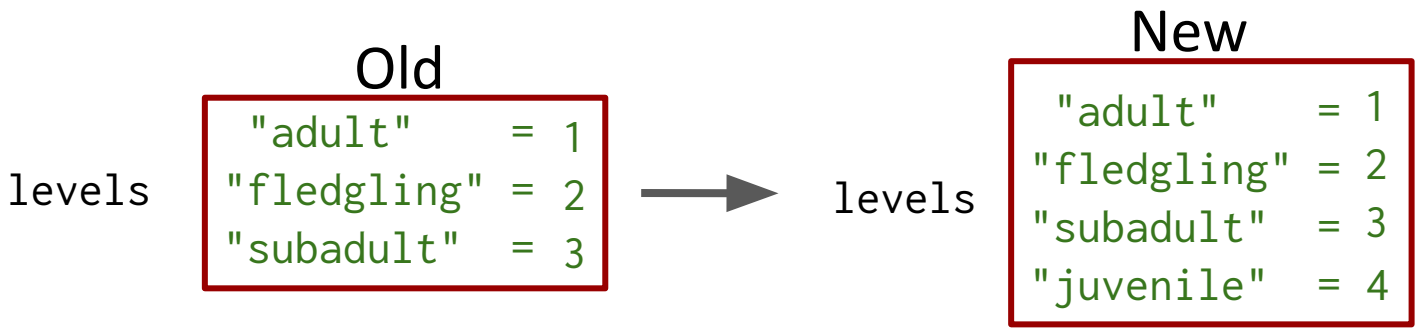
Factor and Levels

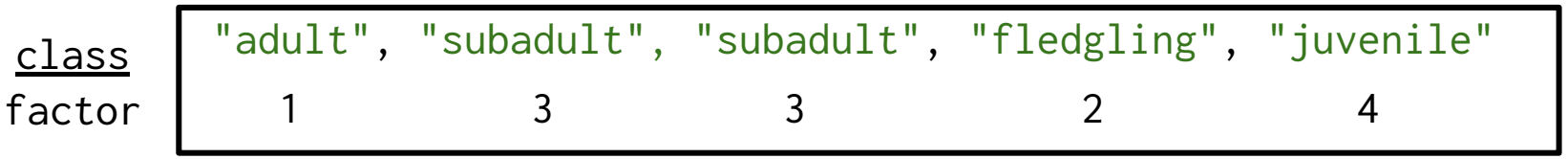
```
# Factor and Levels
age_f[5] <- "juvenile"  # Saved as an NA
```



Factor and Levels

```
# Factor and Levels
levels(age_f) <- c("adult", "subadult", "fledgling", "juvenile")
age_f[5] <- "juvenile"
```



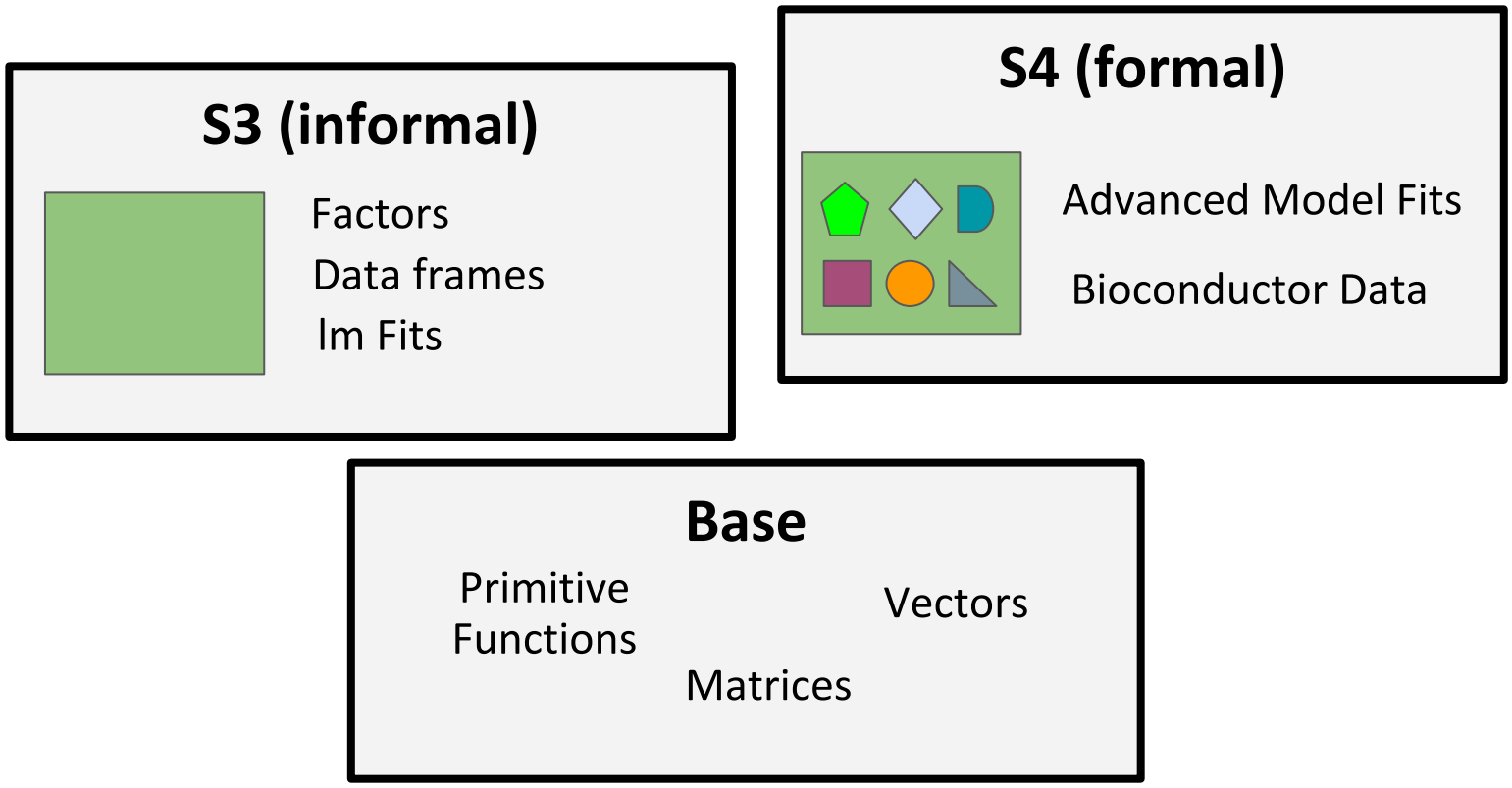


Class Transformations

```
# Transform
age_ch <- as.character(age_f)

# Class Transformations: Using as.xxx()
chr <- c("TRUE", "FALSE", "FALSE")
tf <- as.logical(chr)
int <- as.integer(tf)

chr <- c("1", "0", "1")
int <- as.integer(chr)
tf <- as.logical(int)

chr <- c("5.41", "94.3", "1.63"))
num <- as.numeric(chr)
```

Base, S3, and S4 Object Classes

```
# Object Classes
```



Objects in Environment

```
# Objects in Current Environment
?ls
ls()
objects()  # equivalent to: ls()

# Removing Objects
?rm
rm(vec, mt)

# Removing All Objects in Current Environment
rm(list = ls())
```

Object Names - Validity

```
# Valid Object Names
Ab_.3 <- 1    # letters, numbers, underscores, and dots only
.is_ok <- 2   # must start with letter or dot
a3 <- 3       # numbers only allowed after an initial letter
a <- 4        # R is case sensitive
A <- 5        # a is not equal to A

# Reserved Words
?reserved     # not allowed as names
```

Getting Help in R

```
# Search for function help files
help(as.integer)
?as.integer

# Search within help files
help.search("kernel")
??"kernel"
```

Style Guide - Variables Names

```
# Good names
day_one <- Sys.Date()
day_1 <- Sys.Date()

# Bad names
First_day_of_the_month <- Sys.Date()
Dayone <- Sys.Date()
Djm1 <- Sys.Date()

# Avoid using names of existing variables and functions
```

Style Guide - Code Syntax

```r
# Bad: No Spaces
df<-data.frame(feet=c(5,6,5,6),inches=c(10,2,8,4))
average<-mean(df$feet/12+df$inches,na.rm=TRUE)

# Good: Spaces
df <- data.frame(feet = c(5, 6, 5, 6), inches = c(10, 2, 8, 4))
average <- mean(df$feet / 12 + df$inches, na.rm = TRUE)

# Google's R Style Guide:
browseURL("http://google.github.io/styleguide/Rguide.xml")

# Advanced R - Style Guide (Hadley Wickham):
browseURL("http://adv-r.had.co.nz/Style.html")
```

Swirl: Matrices and Data Frames

```
# Open "R Programming, Lesson 7: Matrices and Data Frames"
swirl()

# Open and go through the lesson:
# 1: R Programming: The basics of programming in R
# 7: Matrices and Data Frames

# Leave swirl by using the bye() function
bye()
```

15 minute coffee break...

Part 2 - Data Exploration

Importing Data

Delimited Files (.csv, .txt)

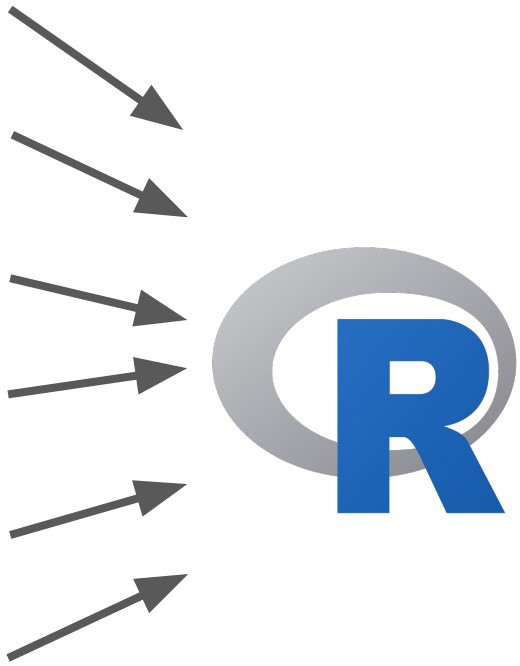
Excel and Access

Statistical Software

SQL Databases

Geospatial Data

Data from the Web



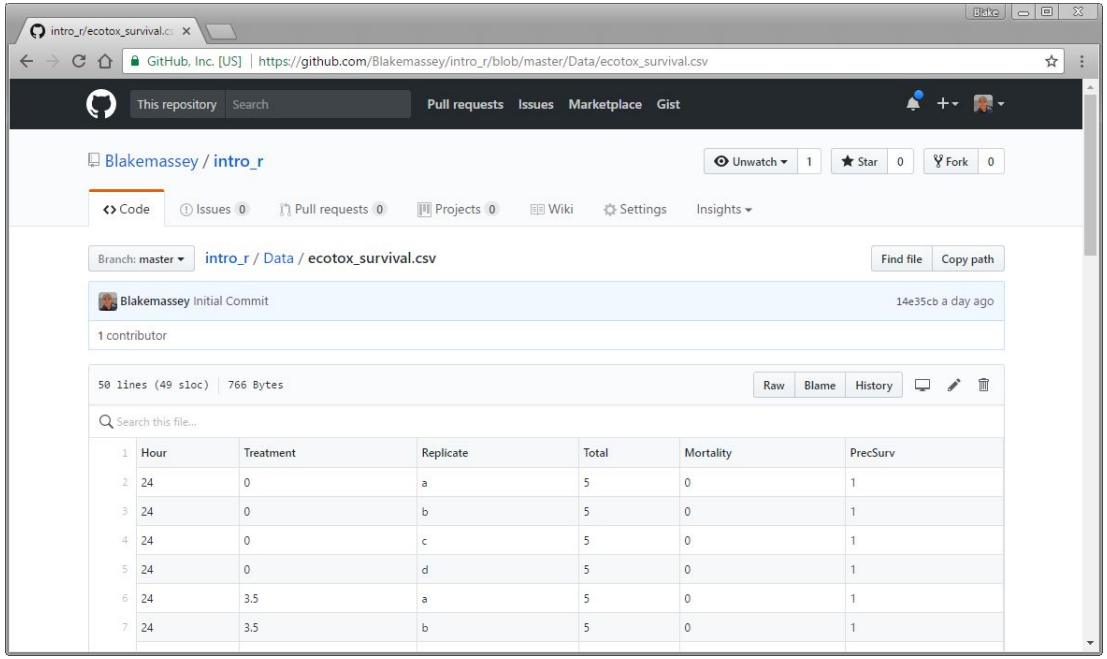
Importing Data

```
# Reading Tables
?read.table()   # reads tabular data
?read.csv()     # reads comma delimited files
?read.csv2()    # reads semi-colon delimited files
?read.delim()   # reads files with any delimiter

# Reading Excel Files
install.packages("readxl")    # Part of the Tidyverse
library(readxl)
```

Importing Data

```
data <- read.csv(file = "Data/blood_iron.csv")
data2 <- read.csv(file = "Data/farm_data.csv")
```

R needs either the file path or the name of the file in the working directory to load external data.



Exploring Data

```
# Data Exploration
str(data)       # summary information on the data structure.
head(data)      # first 6 lines of data
tail(data)      # last 6 lines of data
summary(data)  # gives summary of data
unique(data$dose) # gives all unique values
class(data$dose)  # tells you the class of the data
table(data$dose)  # build a table of counts at each factor
is.na(data)    # figure out if there are any NA's in the object
```

Finding NAs

```
# How to deal with: NAs
sum(c(1, NA, 4)) # This comes to NA
sum(c(1, NA, 4), na.rm=T) # This actually comes to 5
```

NAs can often derail functions - so make sure they aren't in there!

```
# How to deal with NAs
?is.na
is.na(c(1:3, NA, 5))
```

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

```
?any
any(is.na(c(1:3, NA, 5)))
```

```
[1] TRUE
```

Find then subset it! Or fill in the missing value if you can!

Data Manipulation 1

```
# Changing values in vectors, matrices, & data frames
vect <- seq(1:10) # create a sequence to manipulate
```

```
> vect
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
vect[3] <- 9    # for changing the 3rd element to 9
```

```
> vect
 [1]  1  2  9  4  5  6  7  8  9 10
```

```
mt <- matrix(1:9, nrow=3) # create a matrix
mt[1,1] <- 9    # for changing the element in row 1 and col 1 to 9
```

```
  > mt
      [,1] [,2] [,3]
 [1,]    1    4    7
 [2,]    2    5    8
 [3,]    3    6    9
```

```
> mt
     [,1] [,2] [,3]
[1,]    9    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Data Manipulation 2

```
# Add cols to new data
df1 <- data.frame(good=1:3, bad=4:6) # create a data frame
df2 <- data.frame(ugly=c("a","b","c")) # create a second data frame
df3 <- cbind(df1, df2)  # cbind those data frames together
df3$var <- c("wa", "wa", "wa")  # for making new column
```

Data Manipulation 3

```
# Change column names
colnames(df3) <- c("Good", "Bad") # changing all names
```

```
> df3
  Good Bad NA NA
1    1   4  a  wa
2    2   5  b  wa
3    3   6  c  wa
```

```
names(df3)[3:4] <- c("&", "Ugly")   # if you know column position
```

```
> df3
  Good Bad & Ugly
1    1   4 a   wa
2    2   5 b   wa
3    3   6 c   wa
```

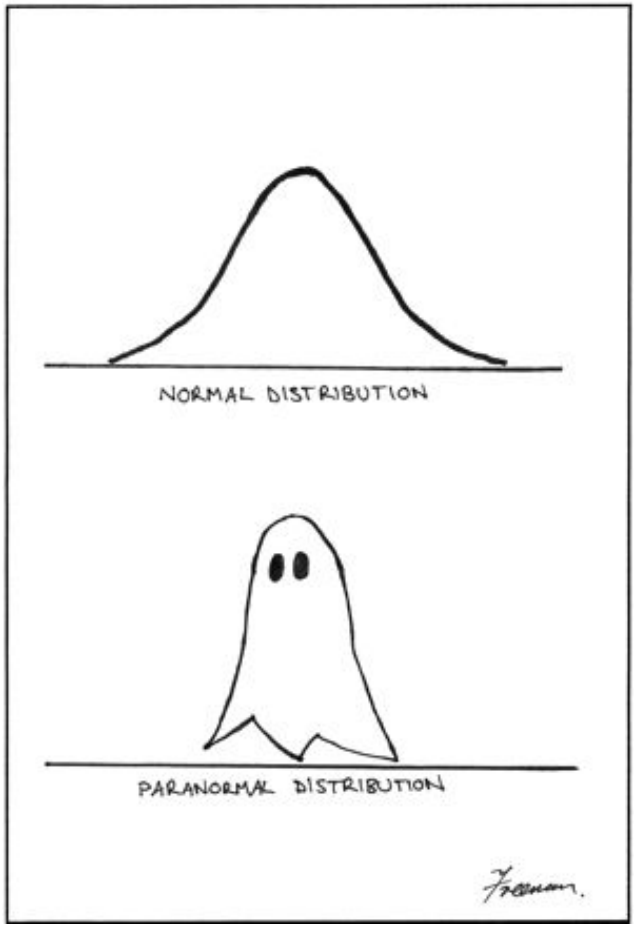
Data Transformation

```
# Create Numeric Vectors
x <- c(0, 1.55, 2.9, 34, 600)

# Transform Data
log(x)              # zeroes produce -Inf
log(x + .001)       # prevents -Inf results
log10(x + .001)     # log base 10
exp(x)              # raises constant e to power x
(x)^2               # raises x to 2nd power
```

Data Standardization

```
# Standardize Data
scale(x)                    # scale + center object x
scale(x, center=FALSE)      # no centering performed
scale(x, scale=FALSE)       # no scaling performed

# Z Transformation
x_stand <- (x - mean(x)) / sd(x)# Z-score standardization
```

Data Distributions



Data Distributions

```
# Probability Distributions
?pnorm()# probability (cumulative dist. function)
?qnorm()# quantile or inverse c.d.f.
?dnorm()# density
?rnorm()# random variable with specified distribution
rnorm(n, mean=0, sd=1)   # default args

# Fill in an appropriate distribution after initial letter
# (e.g., rbinom, pbeta, qgeom, dgamma, rpois, etc…)

# Note: runif = Random sample of uniform distribution

# For more information on the differences between these R functions
# see:  http://seankross.com/notes/dpqr/
```

Visual Data Exploration - Plotting

No better way to get familiar with data!

Critical for exploring statistical assumptions!

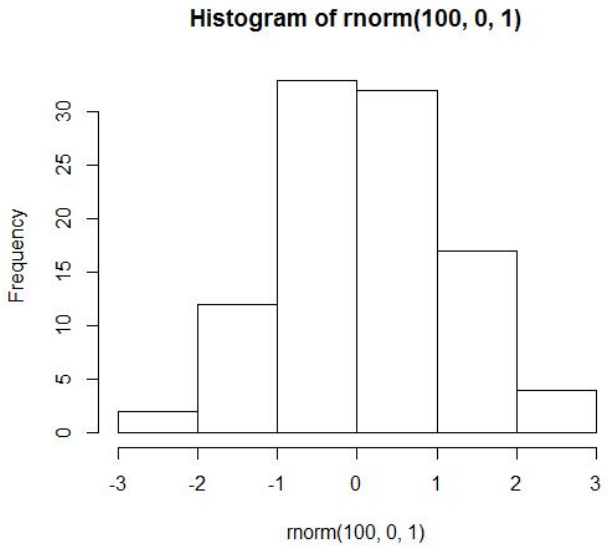
Basic Plots

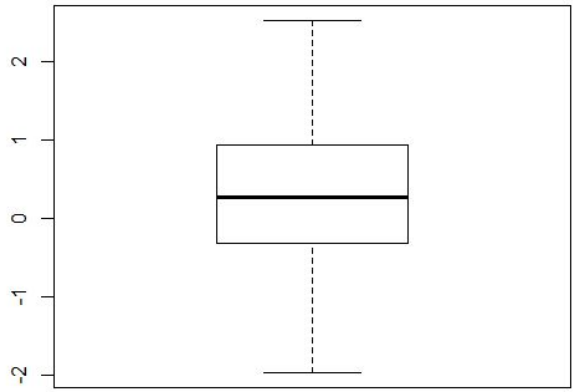
```
?hist()     # histogram
?boxplot()  # boxplot
?dotchart() # Cleveland's dotplot
?plot()     # scatterplot
```

Above are some common, useful plots.
Many others are available.

Passing Functions as Args

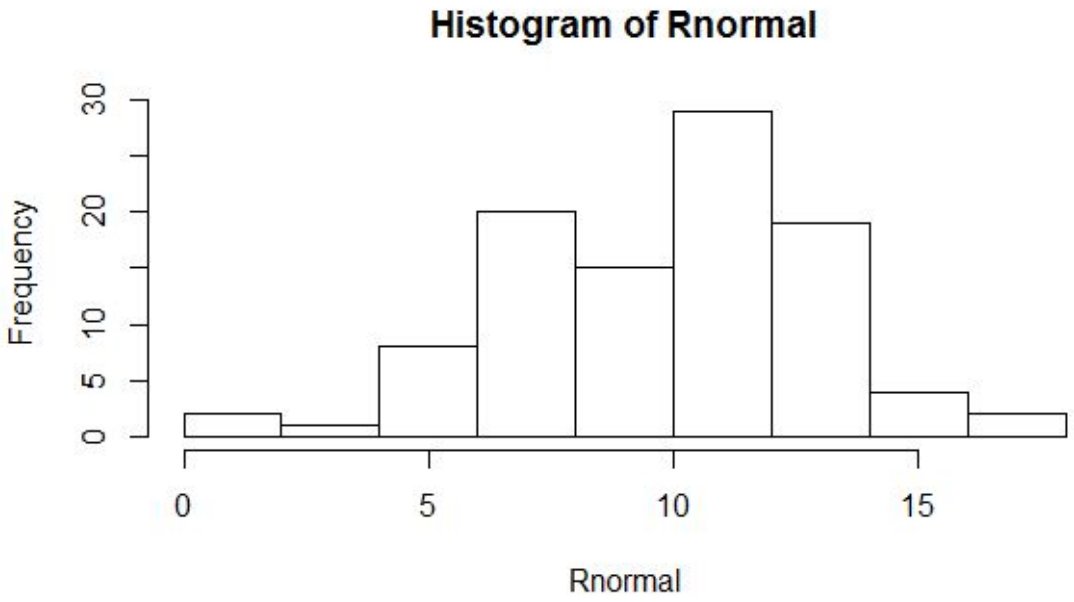
```
# Functions can be passed as arguments to other functions

rnorm(100, 0, 1) # A normal distribution

hist(rnorm(100, 0, 1)) # histogram of 100 normally dist values

boxplot(rnorm(100)) # boxplot of 100 normally dist values
```





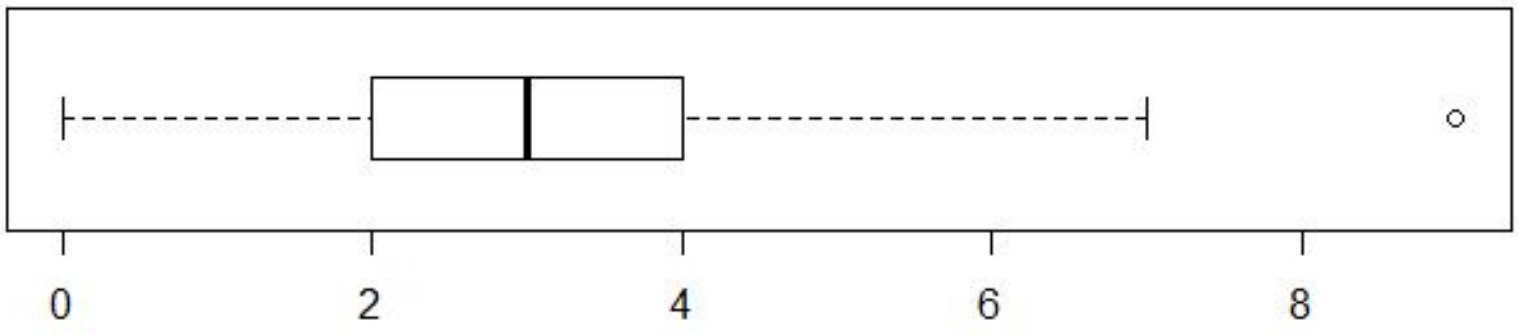
Histogram

```
hist(rnorm(100, 0, 1)) # histogram of 100 normally dist values

Rnormal <- rnorm(n=100, mean=10, sd=3)

hist(Rnormal)
```



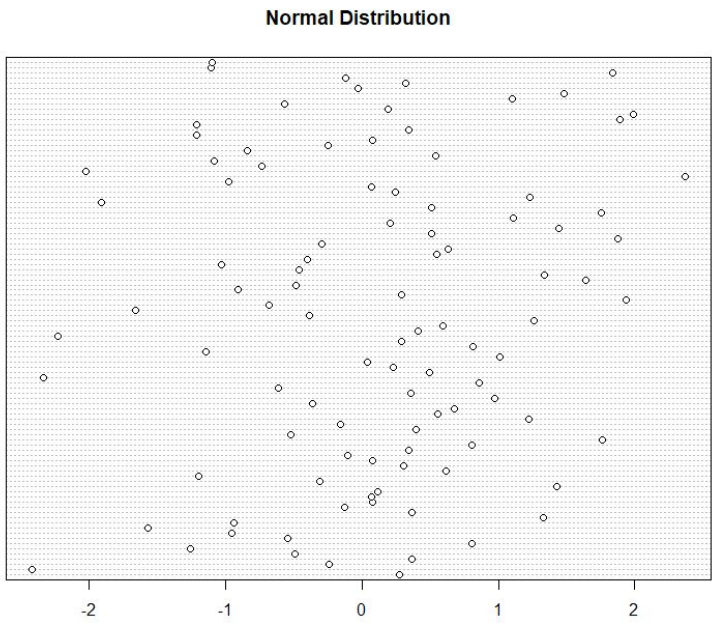
Boxplots

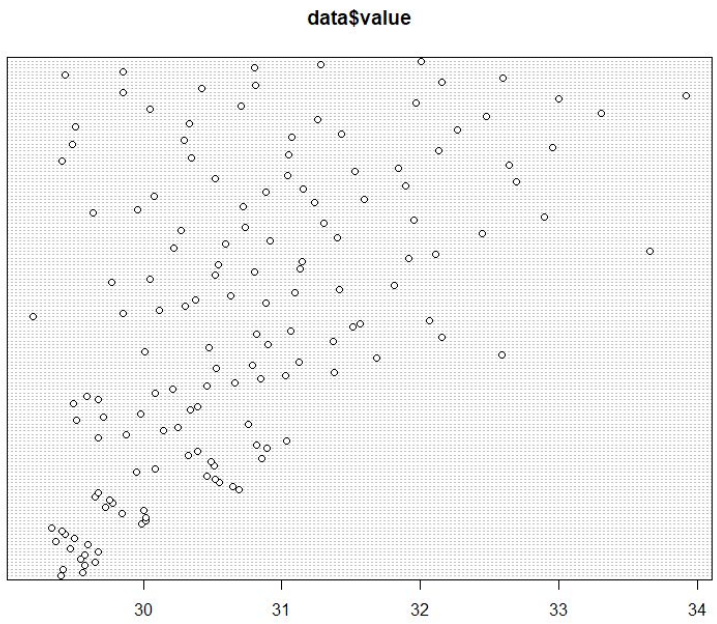
```
Rpoisson <- rpois(100,3)  # Assign random Poisson distribution

boxplot(Rpoisson)     # boxplot of 100 Poisson distributed values
                      # with a lambda of 3

boxplot(Rpoisson, horizontal=TRUE) # Horizontally oriented plot
```



Dot Charts

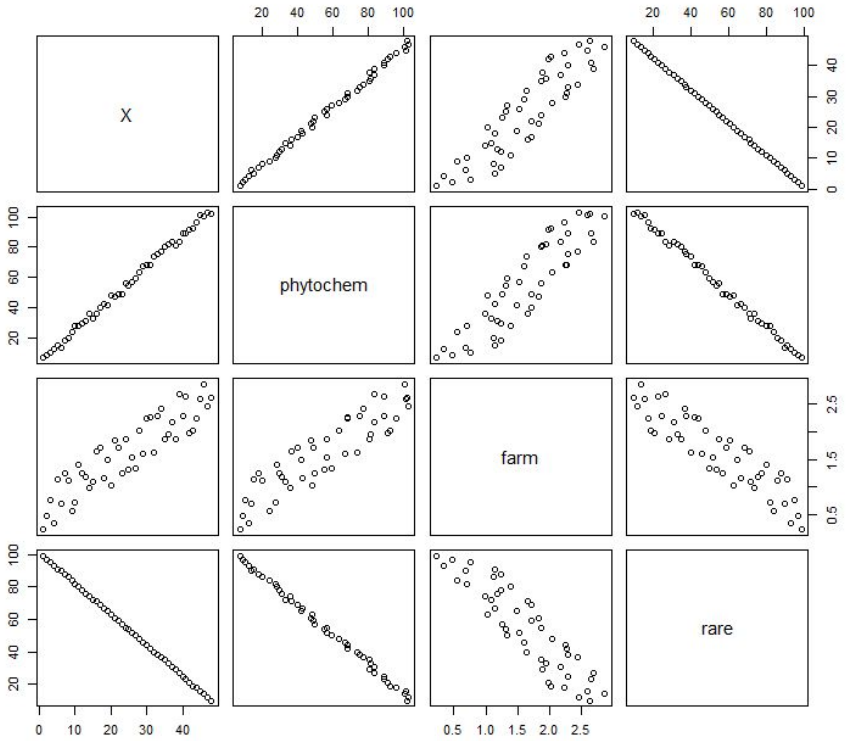
```
# Cleveland dot charts can show dist. (better than bar charts)

dotchart(rnorm(100)) # Dotchart of 100 normally dist values

dotchart(data$value)  # Dotchart of a column in data
```





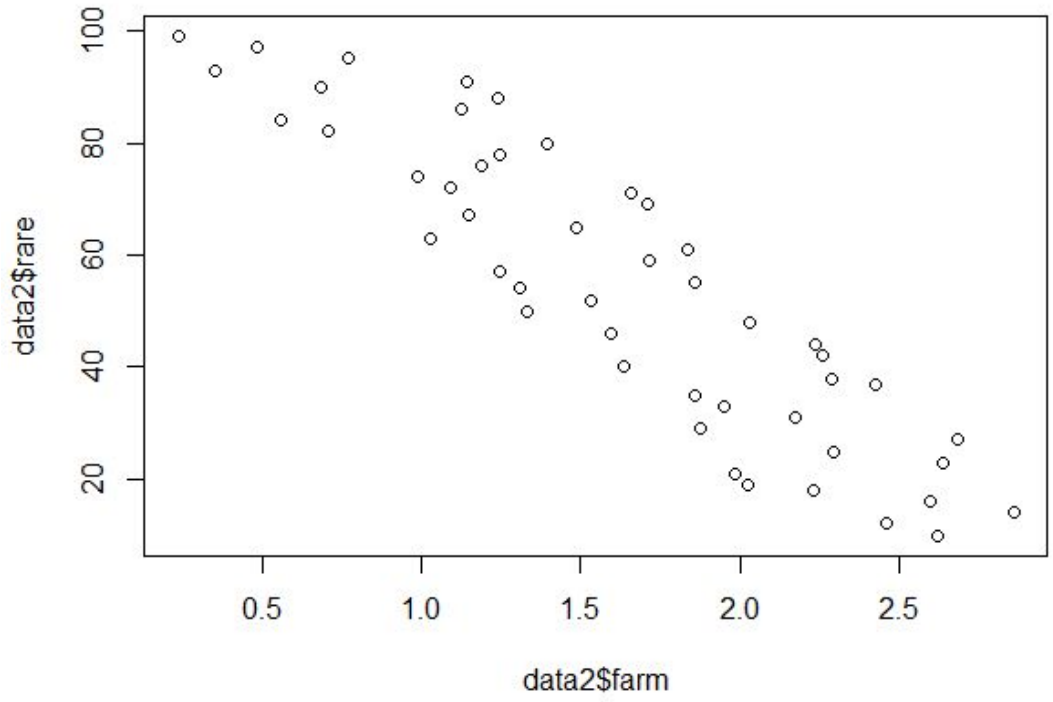
Generic Plots

```
plot(x, y)      # scatter plot of vectors x and y
plot(data2)     # R will create a correlation plot (> 2 columns)
```



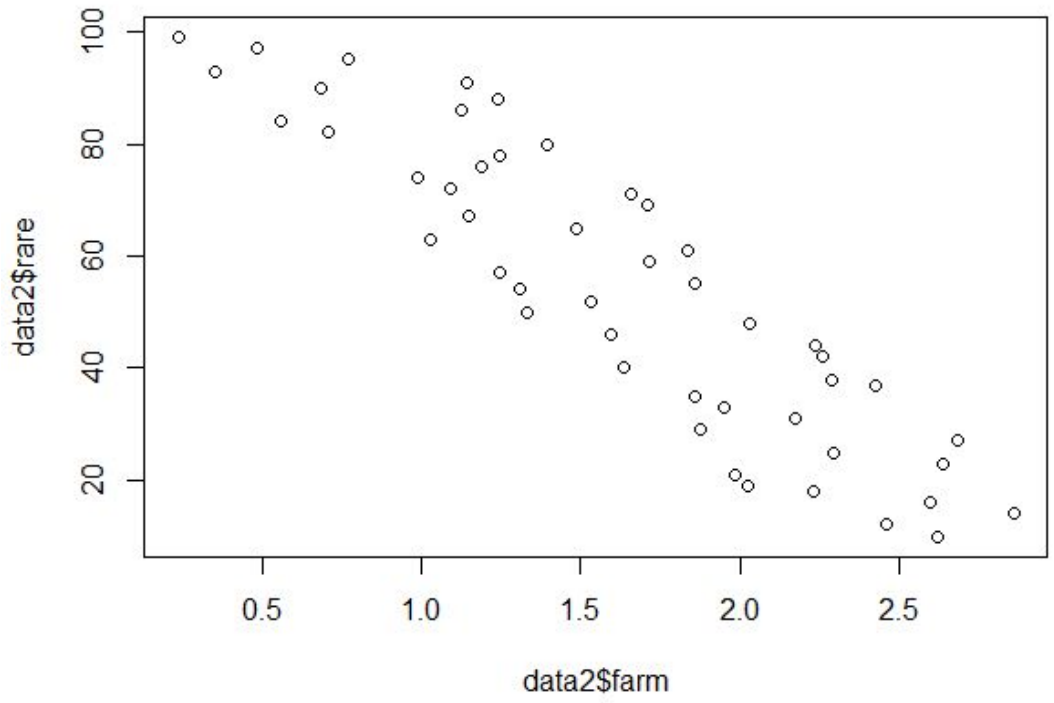
Generic Plots

```
plot(x = data2$farm,  # a column named farm in data2
     y = data2$rare)  # a column named rare in data2
```



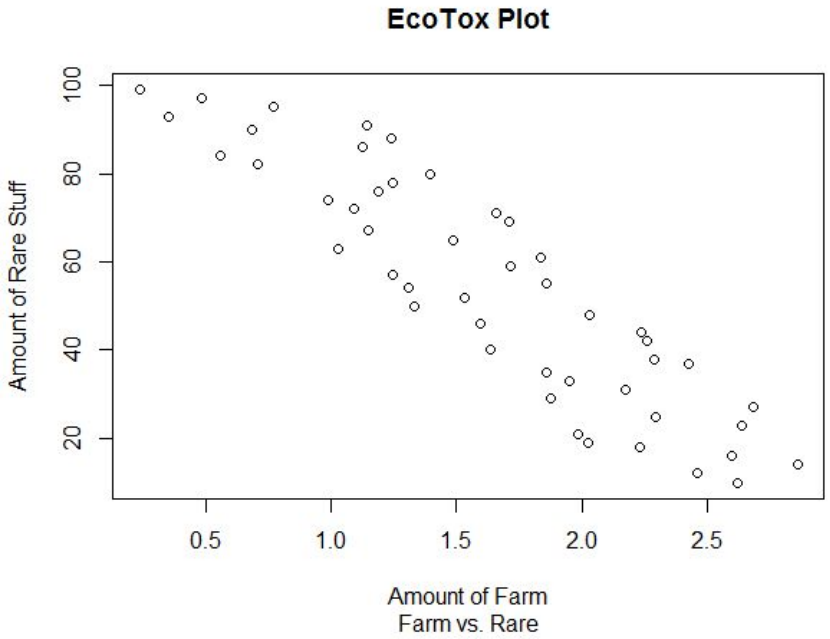
Generic Plots

```
farm <- data2$farm      # assign data to its own object
rare <- data2$rare      # assign data to its own object
plot(farm, rare)    # plot() recognizes them
```

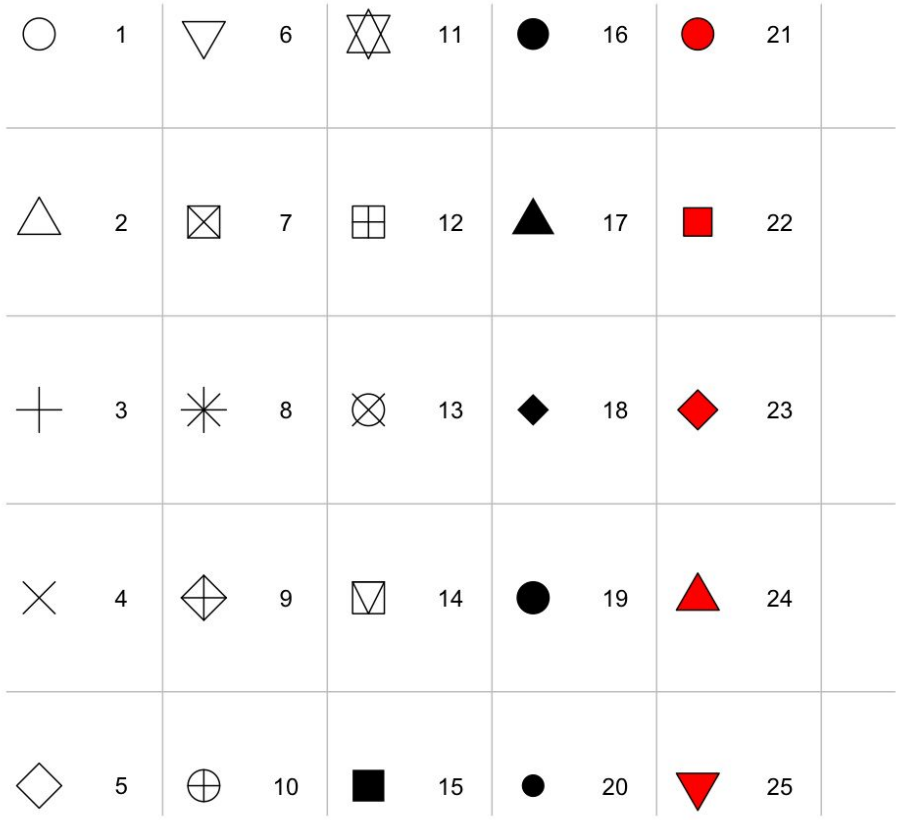


Plot Labels

```
plot(farm, rare,
     main = "EcoTox Plot",              #This is the main title
     sub = "Farm vs. Rare",             #This is the sub title
     xlab = "Amount of Farm",           #This is the x-axis title
     ylab = "Amount of Rare Stuff")     #This is the y-axis title
```



Point Shapes



http://www.ling.upenn.edu/~joseff/rstudy/week4.html#Basics

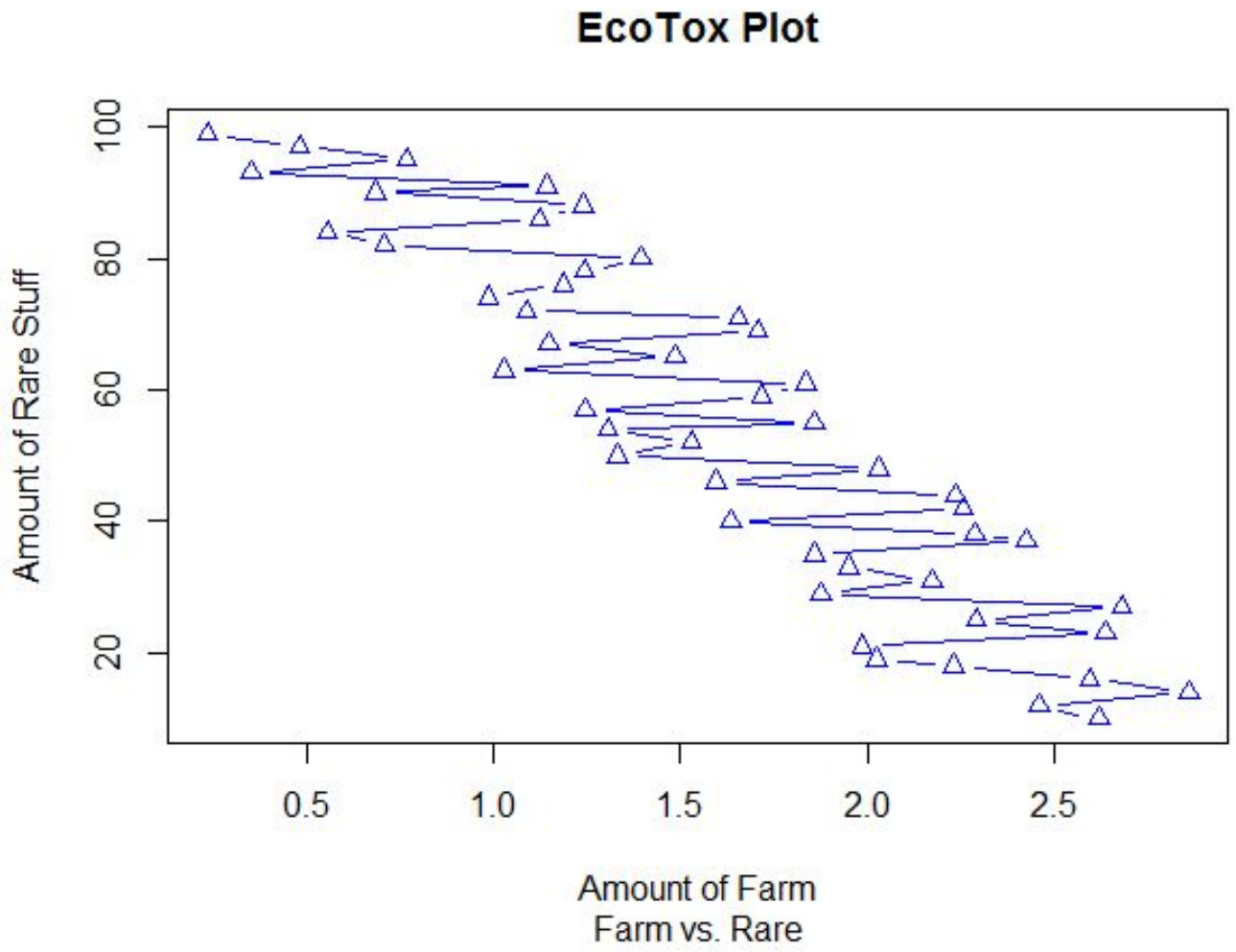
Point Shapes

```
par(mar = c(1,1,1,1))
plot(x = c(rep(1,5),rep(2,5),rep(3,5),rep(4,5),rep(5,5)),
     y = rep(5:1,5),
     axes = F,
     xlab = "",ylab = "",
     xlim = c(1,6),ylim = c(1,5),
     pch = 1:25,
     bg = "red",cex = 3)
text(x = c(rep(1,5),rep(2,5),rep(3,5),rep(4,5),rep(5,5))+.5,
     y = rep(5:1,5),
     labels = as.character(1:25))
abline(h = (1:5)+0.5,col = "grey")
abline(v = (1:5)+0.75,col = "grey")
```

Plot Shapes + Sizes

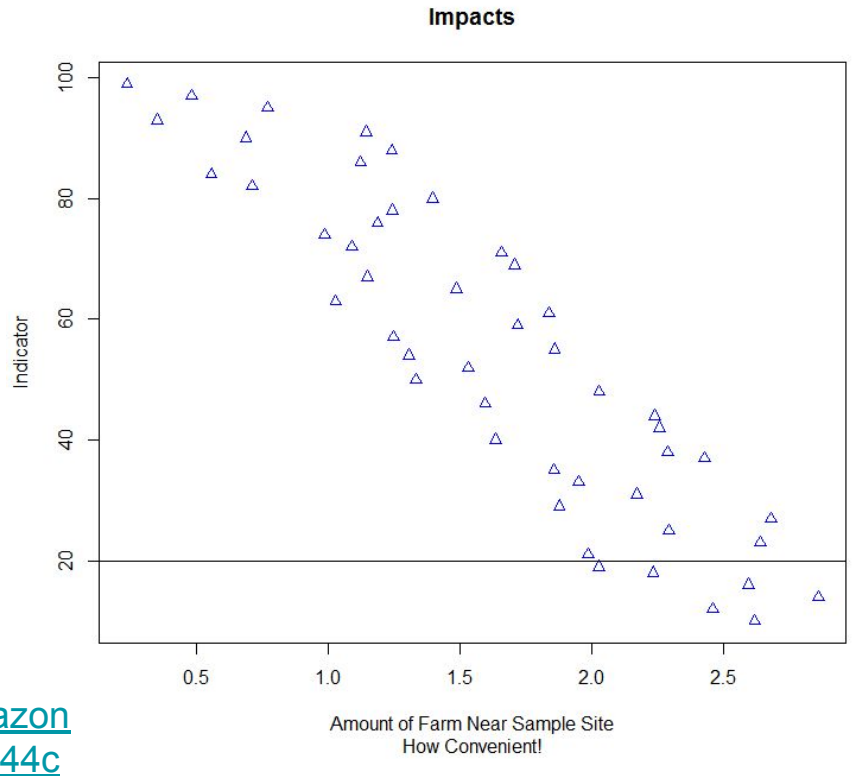
```
plot(farm, rare,
     main = "EcoTox Plot",                #This is the main title
     sub = "Farm vs. Rare",               #This is the sub title
     xlab = "Amount of Farm",             #This is the x-axis title
     ylab = "Amount of Rare Stuff",       #This is the y-axis title
     pch = 2,                             #This changes point shapes
     col = "blue",                        #This changes point color
     type = "b")                          #This changes line type
```

Plot Shapes + Sizes



Graphical Parameters

```
par(mfrow=c(1,1))        # change # of graphs displayed (row, col)
par(cex=1)               # change all elements size
abline()                 # add a vertical or horizontal line
```



More on base graphics:
http://rstudio-pubs-static.s3.amazonaws.com/7953_4e3efd5b9415444ca065b1167862c349.html

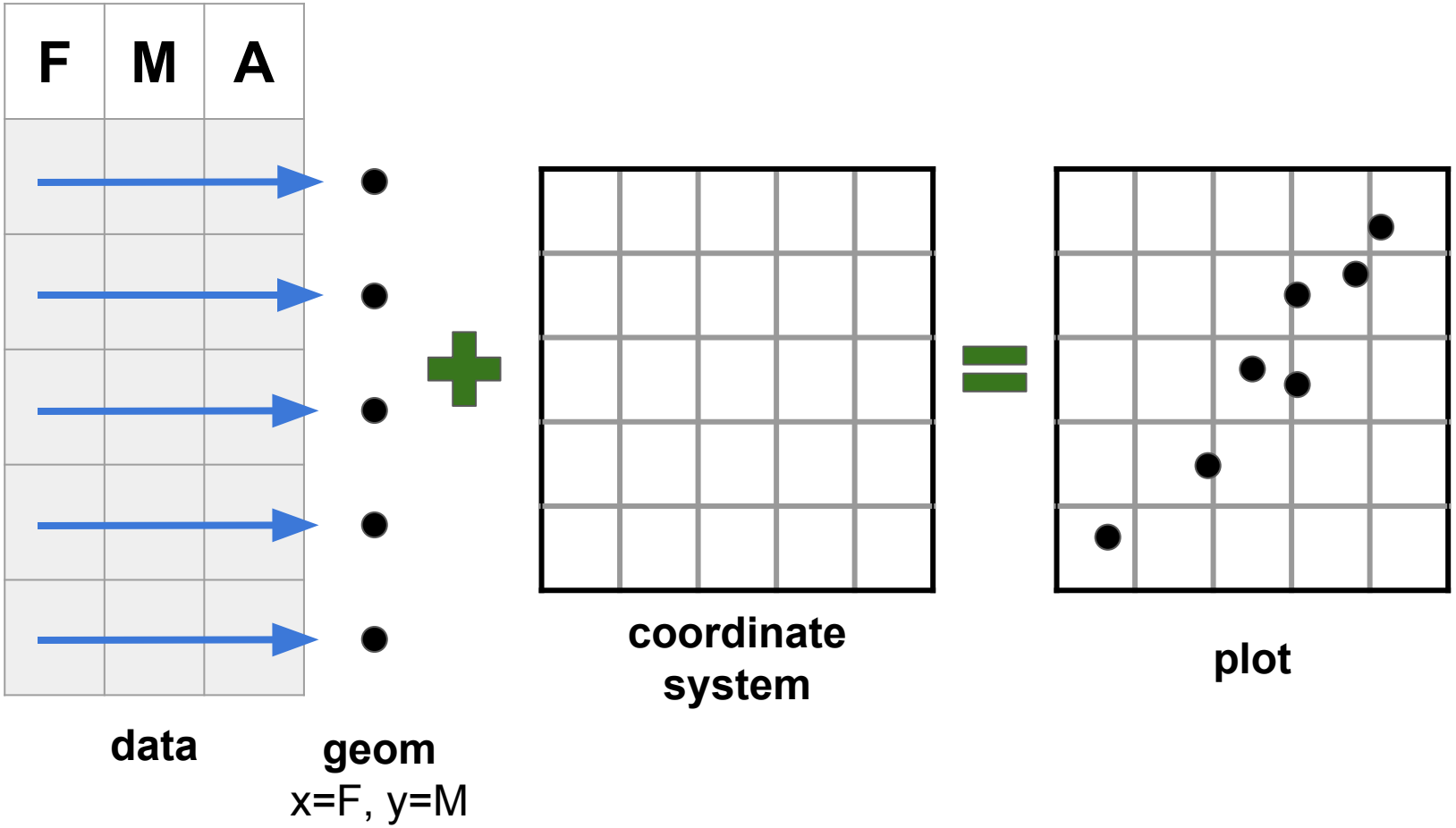
Plots (all put together…)

```
myplot <- plot(
  x = xdata,
  y = ydata,
  main = "This is the main title",
  col = "blue",
  type = "s" # plot type
  xlab = "This is the x-axis label",
  ylab =  "This is the y-axis label",
  cex = 1 # size of points
  pch = 2 # shape of points
)
```

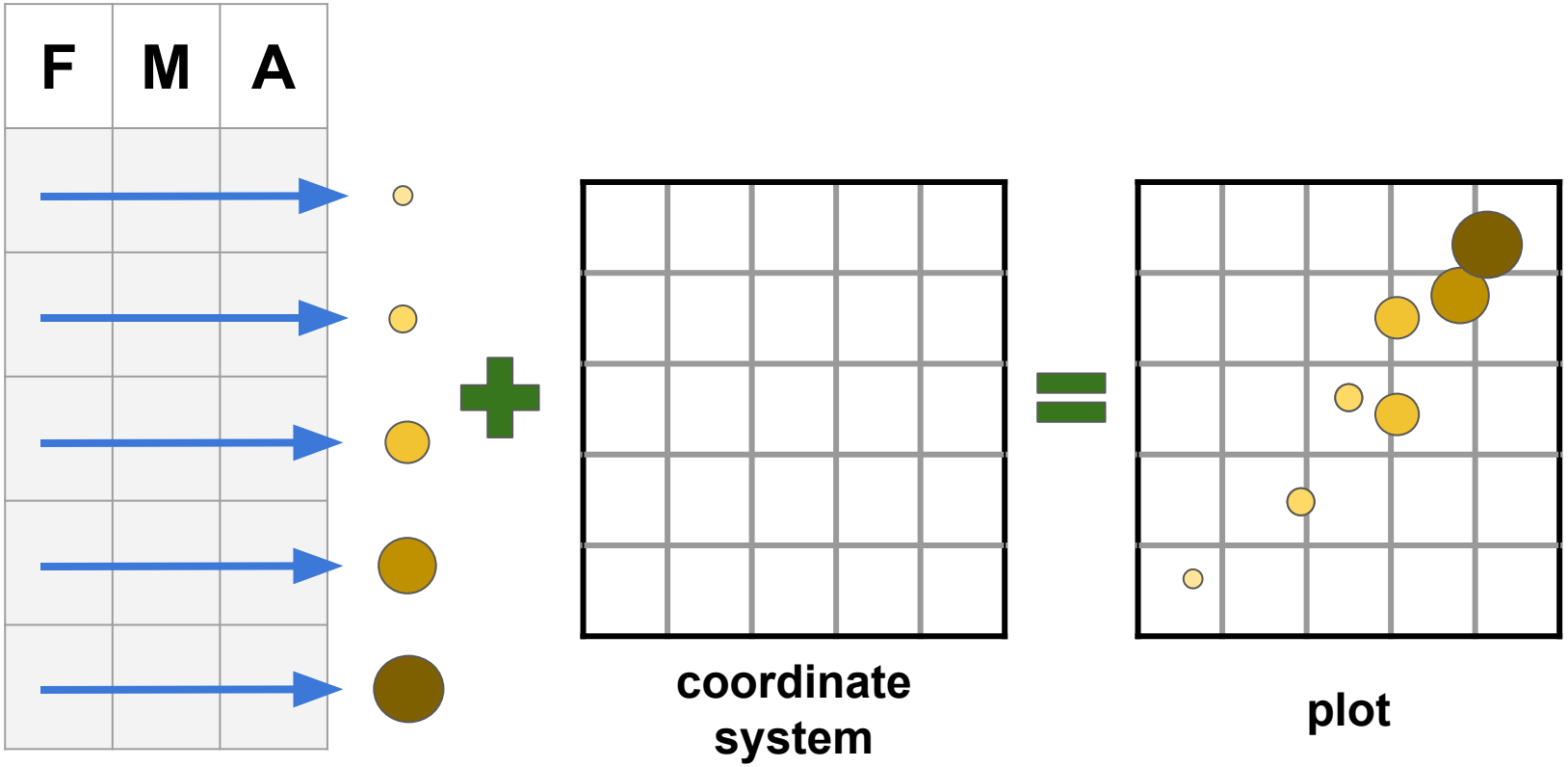
A big problem with the base plotting is that you can't store it.

That brings us to: ggplot2

ggplot2

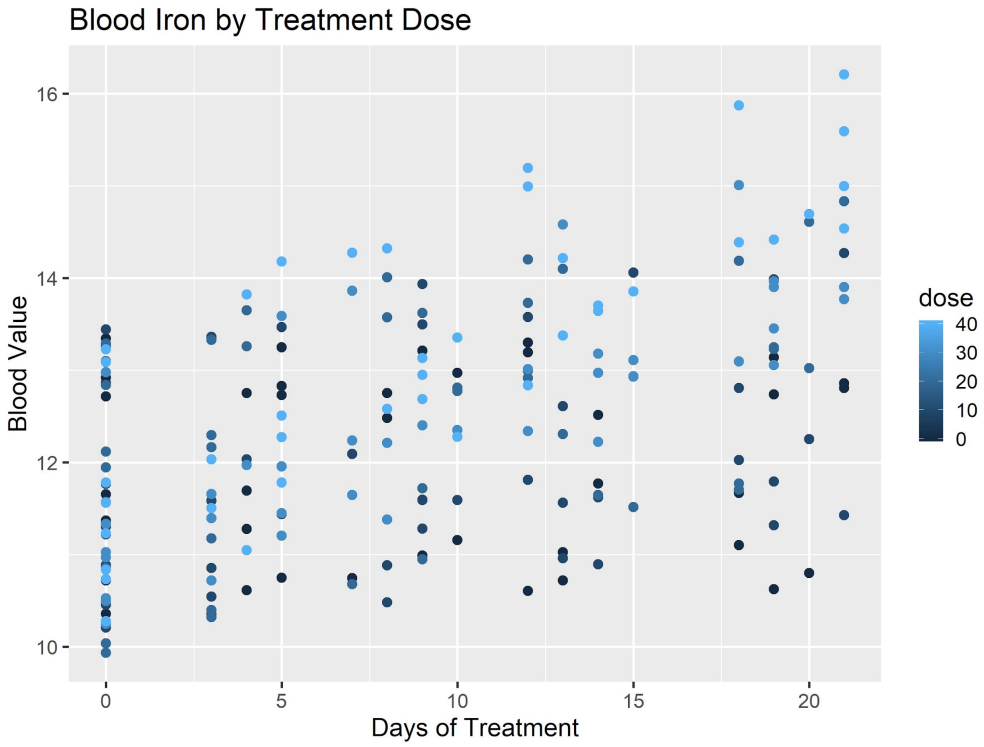


ggplot2



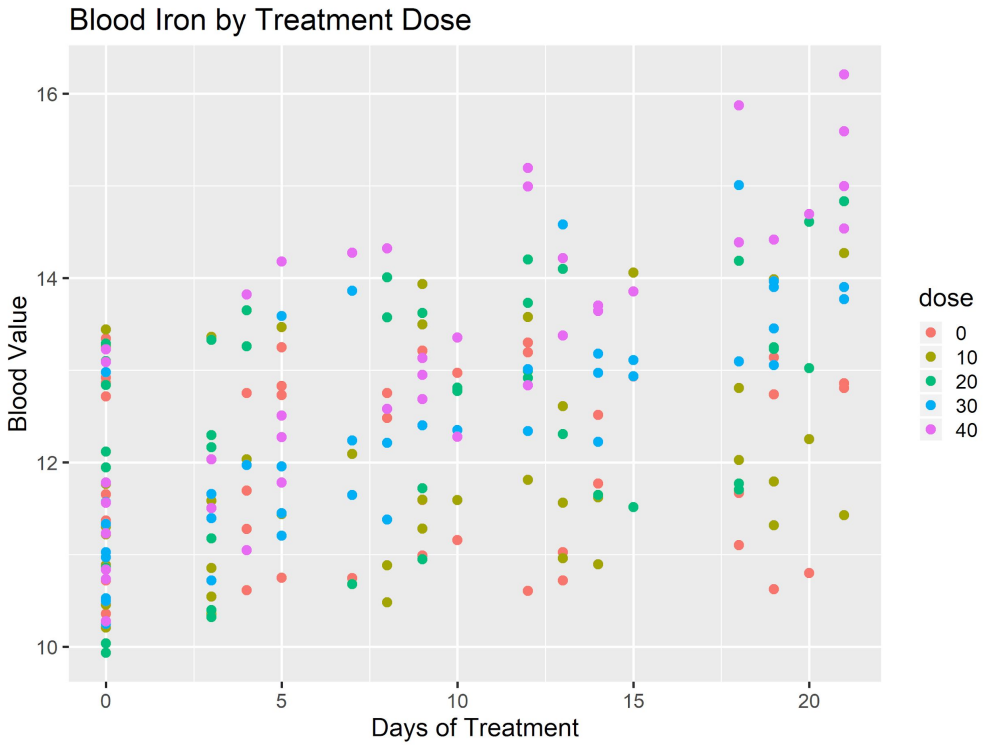
ggplot2 Example

```
library(ggplot2)
gg1 <- ggplot(data = data, aes(x = day, y = value, color = dose)) +
  geom_point(size = 3) + ggtitle("Blood Iron by Treatment Dose") +
  xlab("Days of Treatment") + ylab("Blood Value")
```



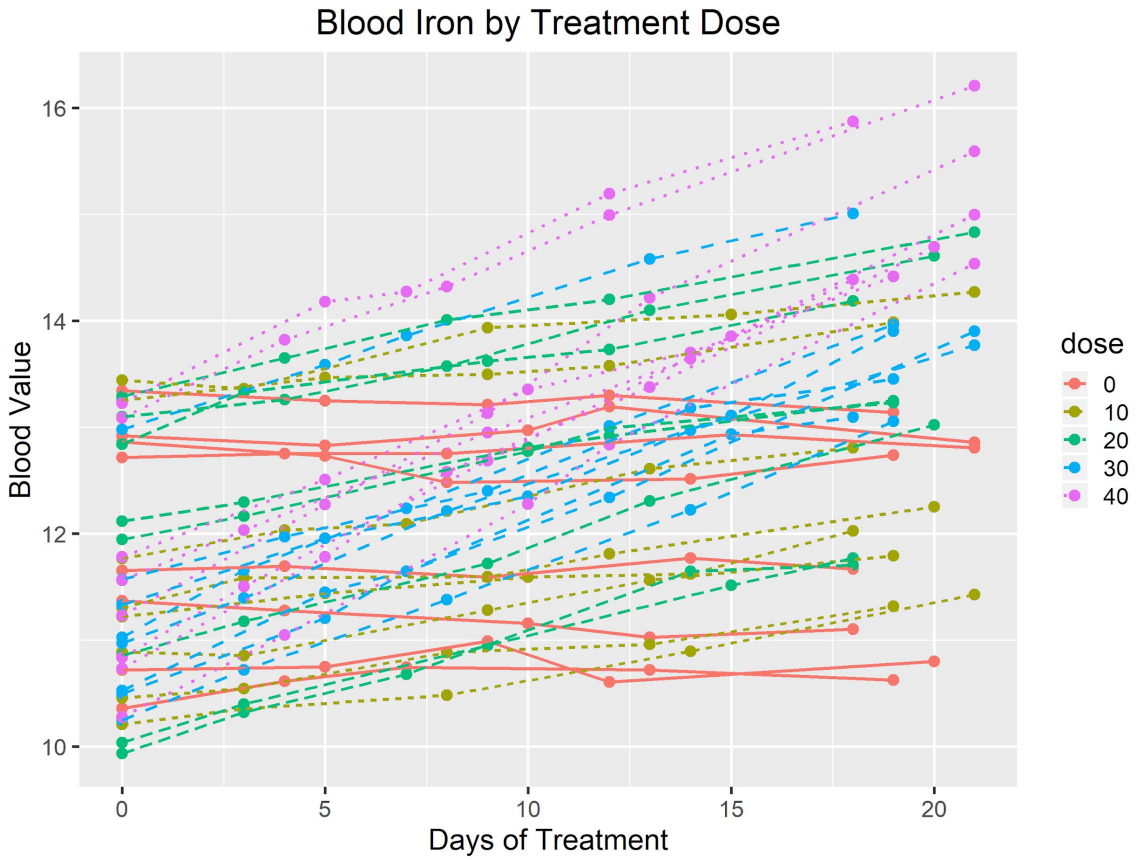
ggplot2 Example

```
data$dose <- factor(data$dose)
gg1 <- ggplot(data = data, aes(x = day, y = value, color = dose)) +
  geom_point(size = 3) + ggtitle("Blood Iron by Treatment Dose") +
  xlab("Days of Treatment") + ylab("Blood Value")
```



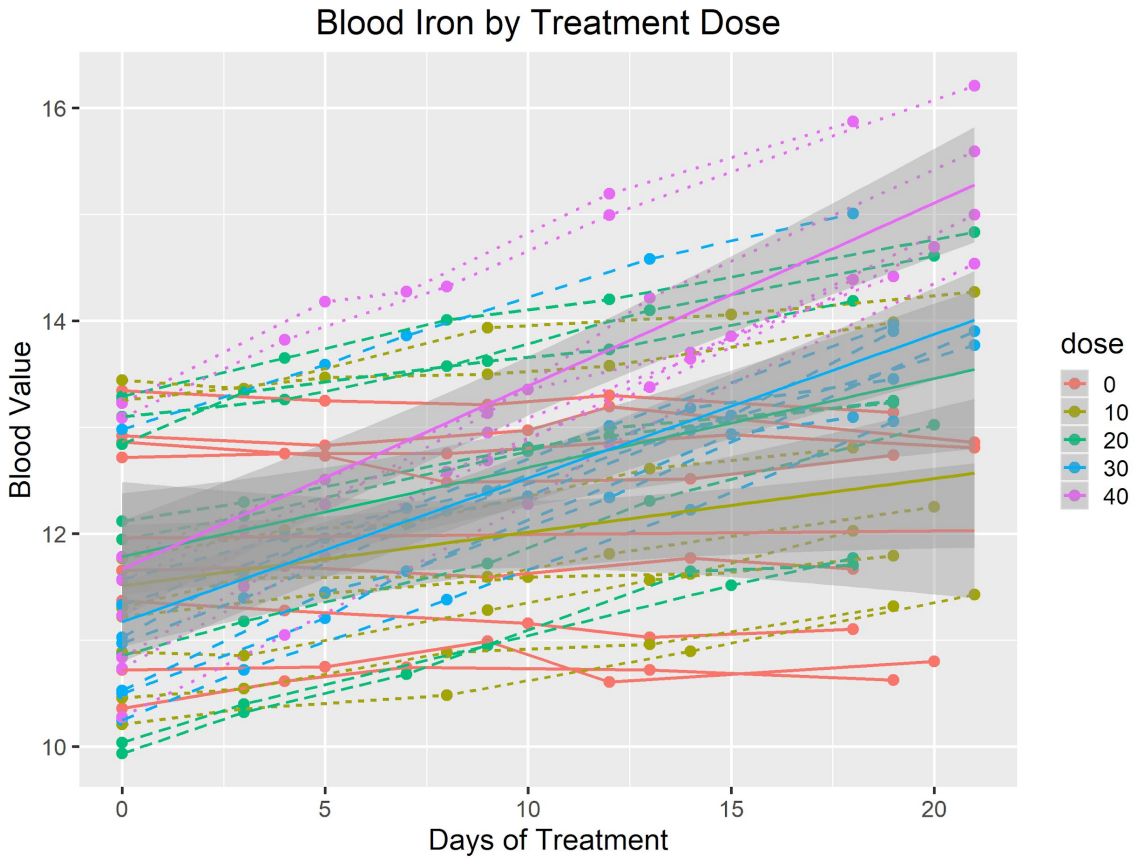
ggplot2 Example

```
gg2 <- gg1 + geom_line(aes(group = id, linetype = dose), size = 1)+
   theme(plot.title = element_text(hjust = 0.5))
```



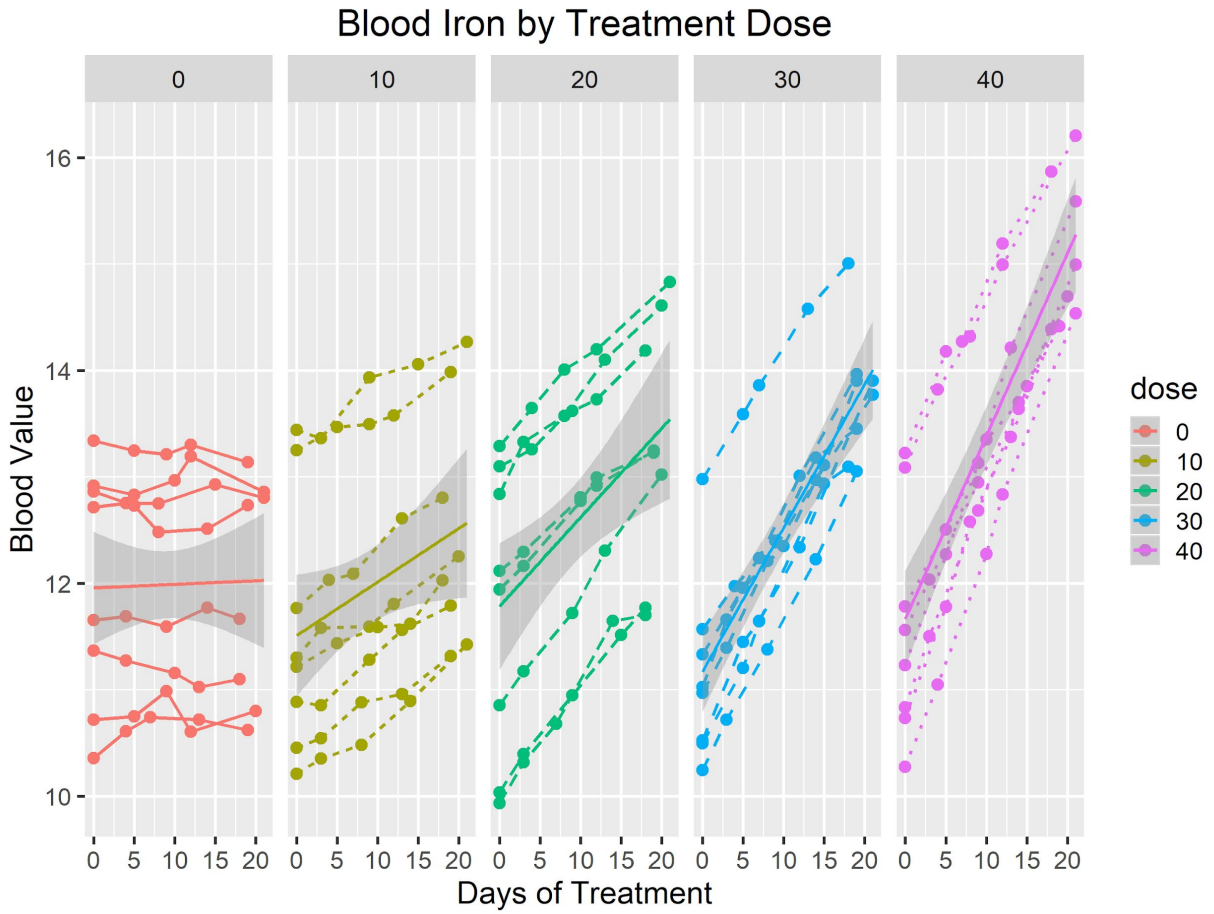
ggplot2 Example

```
gg3 <- gg2 + geom_smooth(method = "glm", aes(group = dose),
  formula = y ~ x)
```



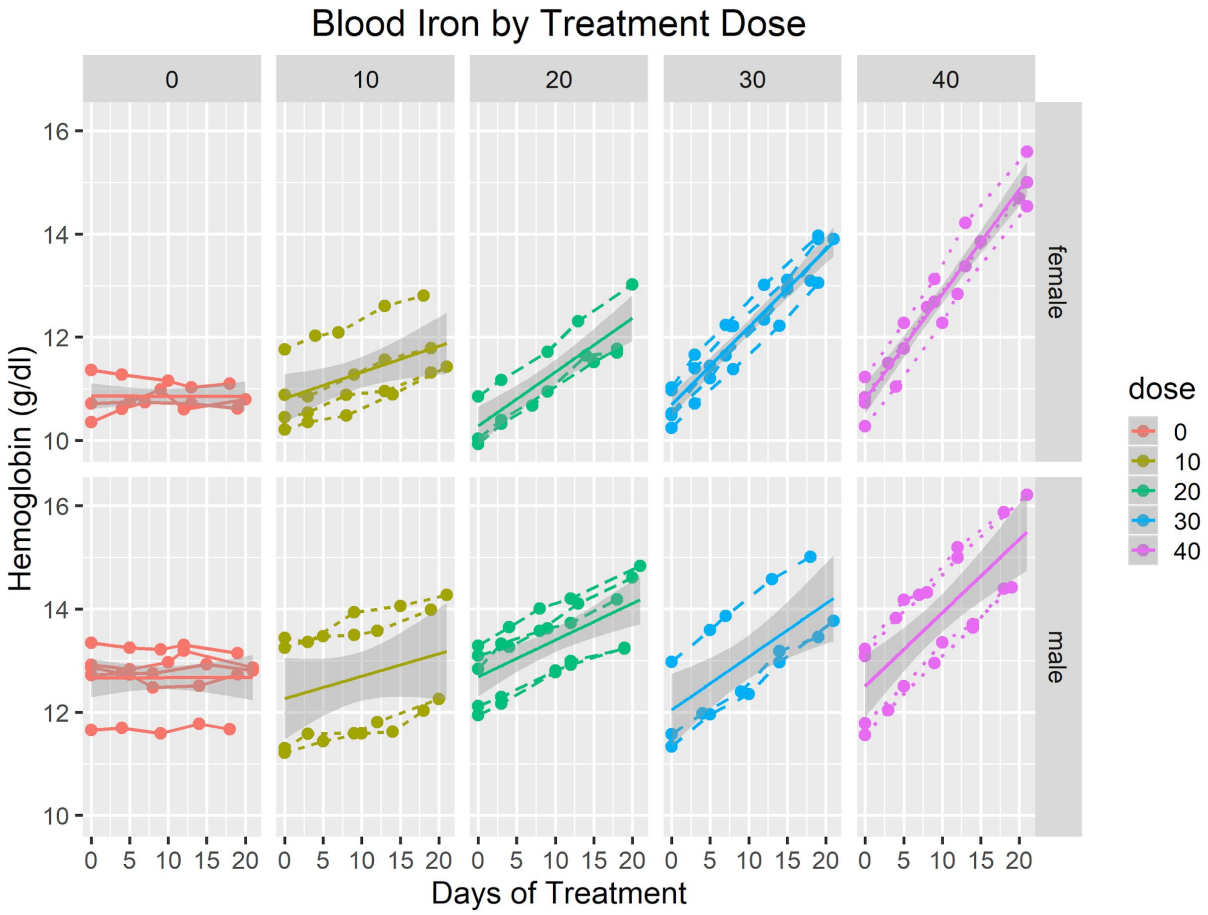
ggplot2 Example

```
gg4 <- gg3 + facet_grid(. ~ dose)
```



ggplot2 Example

```
gg5 <- gg4 + facet_grid(sex ~ dose) + ylab("Hemoglobin (g/dl)")
```



Best Practices

- Use common sense and BE CONSISTENT!
- R is about reproducibility - write code as scripts
- Start with a header:
  - Copyright,
  - Authors,
  - Date created,
  - Version of pkgs used,
  - etc...

Style Guide - Object Names

General Suggestions

Use lowercase, or camelcase (lowerCamel, UpperCamel)
Use underscore "_" to separate words
Use nouns for variable names
Use verbs for function names
Be concise and meaningful

“There are only two hard things in Computer Science: cache invalidation and naming things.”

– Phil Karlton

PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.

THIS IS ***THE*** CORRECT WAY TO WRITE NUMERIC DATES:

2013-02-27

THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13

20130227 2013.02.27 27.02.13 27-02-13

27.2.13 2013. II. 27. $27\frac{1}{2}$-13 2013.158904109

MMXIII-II-XXVII MMXIII $\frac{\text{LVII}}{\text{CCCLXV}}$ 1330300800

$((3+3)\times(111+1)-1)\times 3/3-1/3^3$ 2013

10/11011/1101 02/27/20/13 $\overset{2}{0}\overset{3}{1}\overset{1}{2}\overset{4}{3}\overset{67}{7}_{5}{}_{8}$

xkcd.com/1179

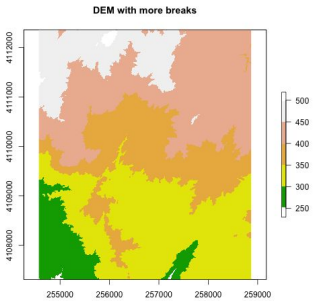
Best Practices

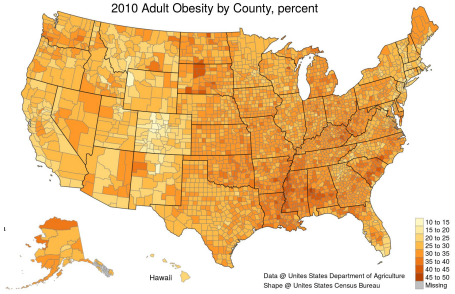
- Use comments (#)!
- Write why, not what

Specific Topics

Spatial Data Packages

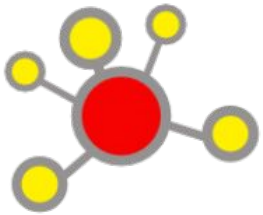
- sf
- raster
- tmap

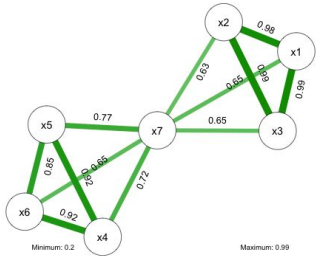




Network Analysis

- igraph
- qgraph





Task Views

- cran.r-project.org/web/views/Distributions.html

Skills for Being a Better R User

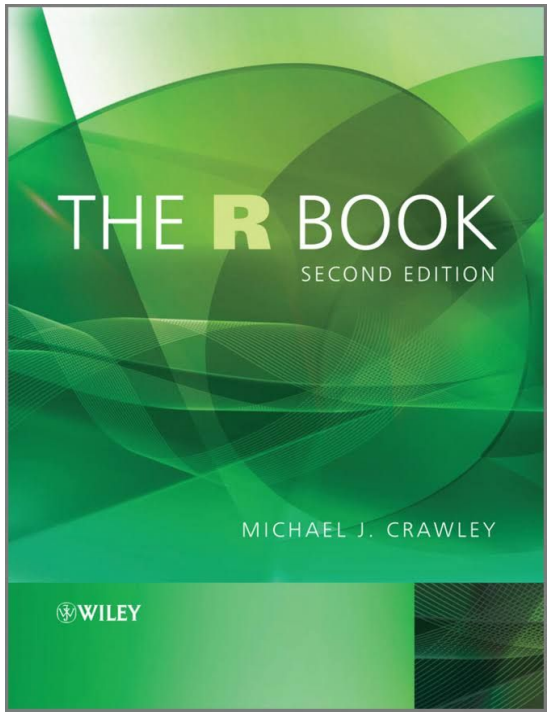
Code Specific:

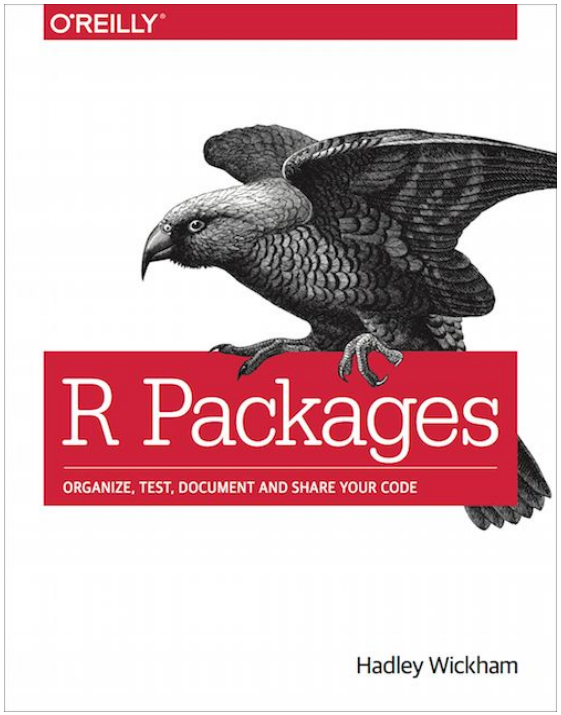
- Control statements `if(), ifelse(), while()`
- For loops `for (i in 1:10) print(i)`
- Writing user-defined functions `my_fun <- function(x){...}`
- Checking and handling errors `try(), tryCatch(), break()`

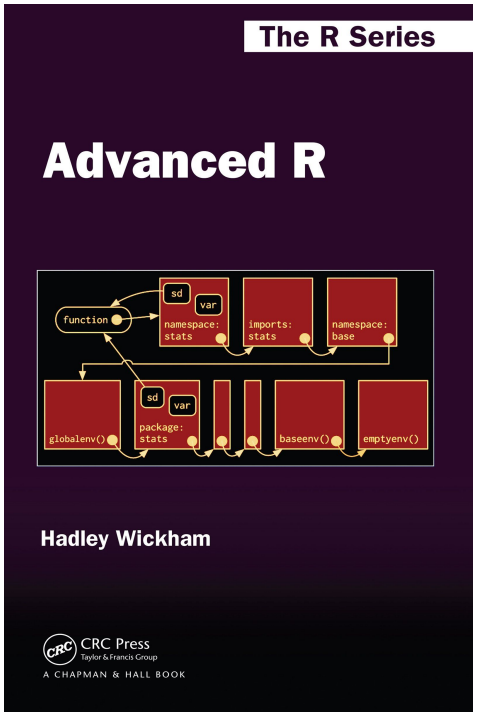
General Usability/Functionality

- Version control (git/GitHub)
- RStudio Projects
- R Markdown

Resources: Books







Many, many more on general R and specific R uses!

Online Classes

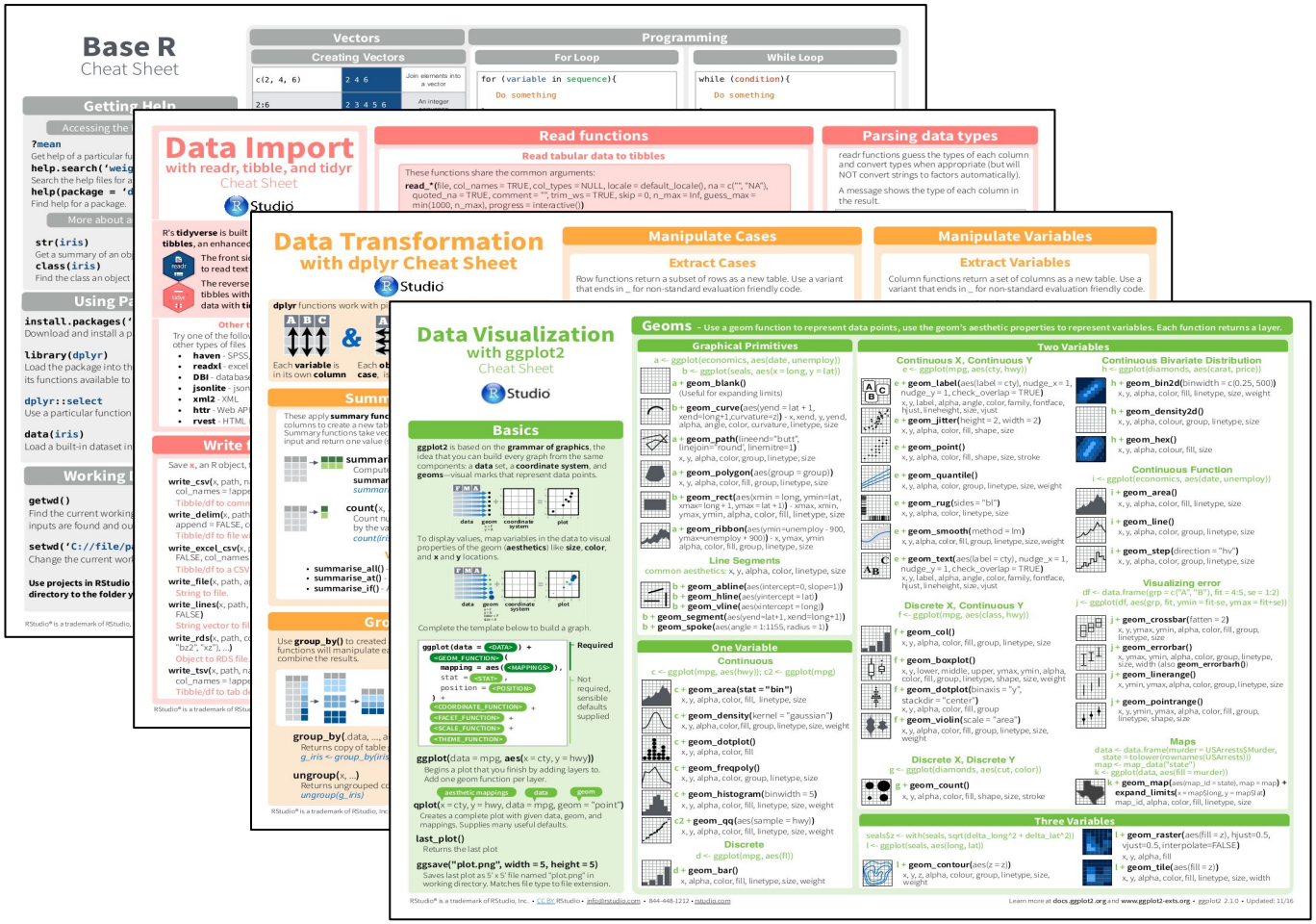
datacarpentry.org

datacamp.com

{swirl}

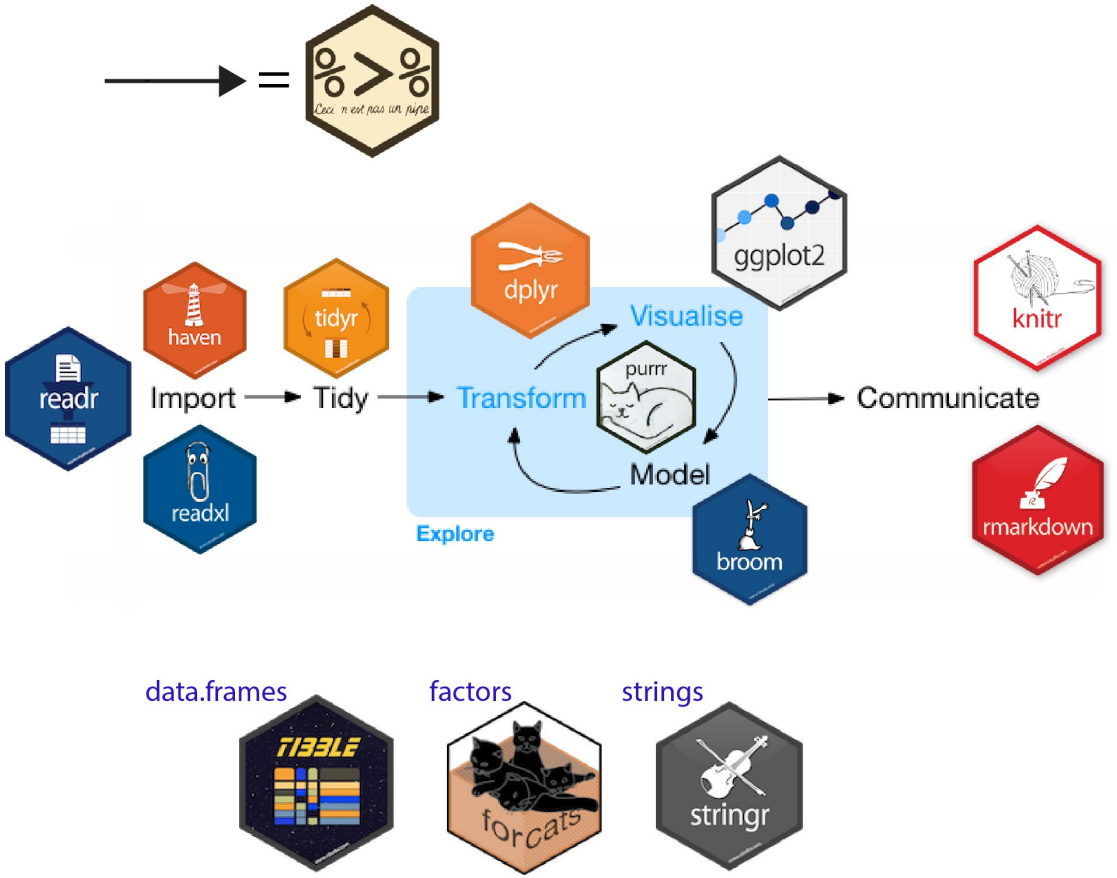
swirlstats.com

RStudio: Cheat Sheets



rstudio.com/resources/cheatsheets

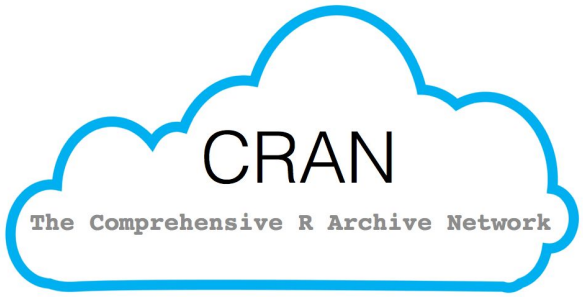
Resources: The Tidyverse

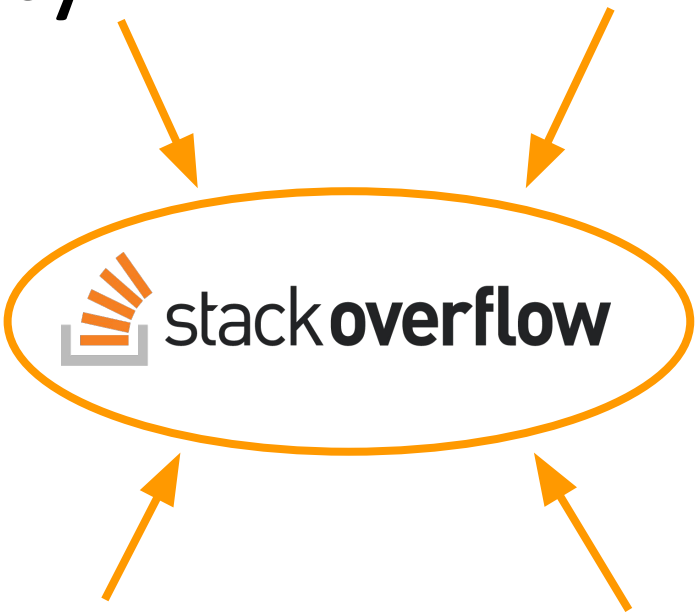


tidyverse.org

Hadley Wickham

R Community





@Rbloggers

#Rstats

#DataScience

Some Takeaways….

Blake	Joe
R/RStudio continue evolving	Metadata is king
Use external resources	Reproducibility
Coding = problem-solving	No single answer

THANKS EVERYONE!