Visualizing Data with R

Biocomputing Working Group

3/28/14

Laura Crothers & Sean Maguire

Download R

- If you haven't already, download and install R!
  - Go to http://www.r-project.org/
  - Under "**Getting Started**", select "**Download R**"
  - Select a CRAN mirror near you
  - Download the right version for your operating system
  - Follow the R installation instructions

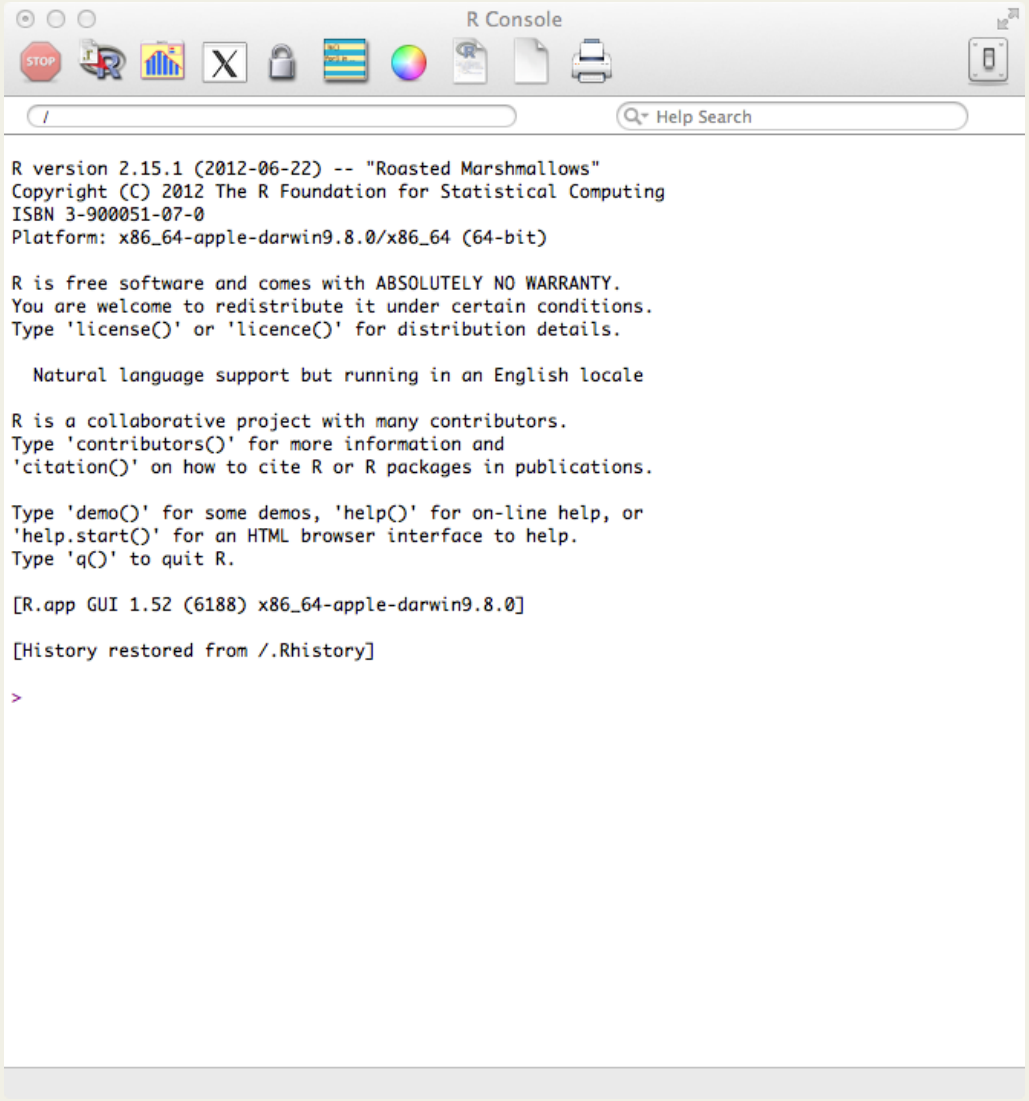
- **Optional**: Download RStudio:
  - https://www.rstudio.com/ide/

The R environment....

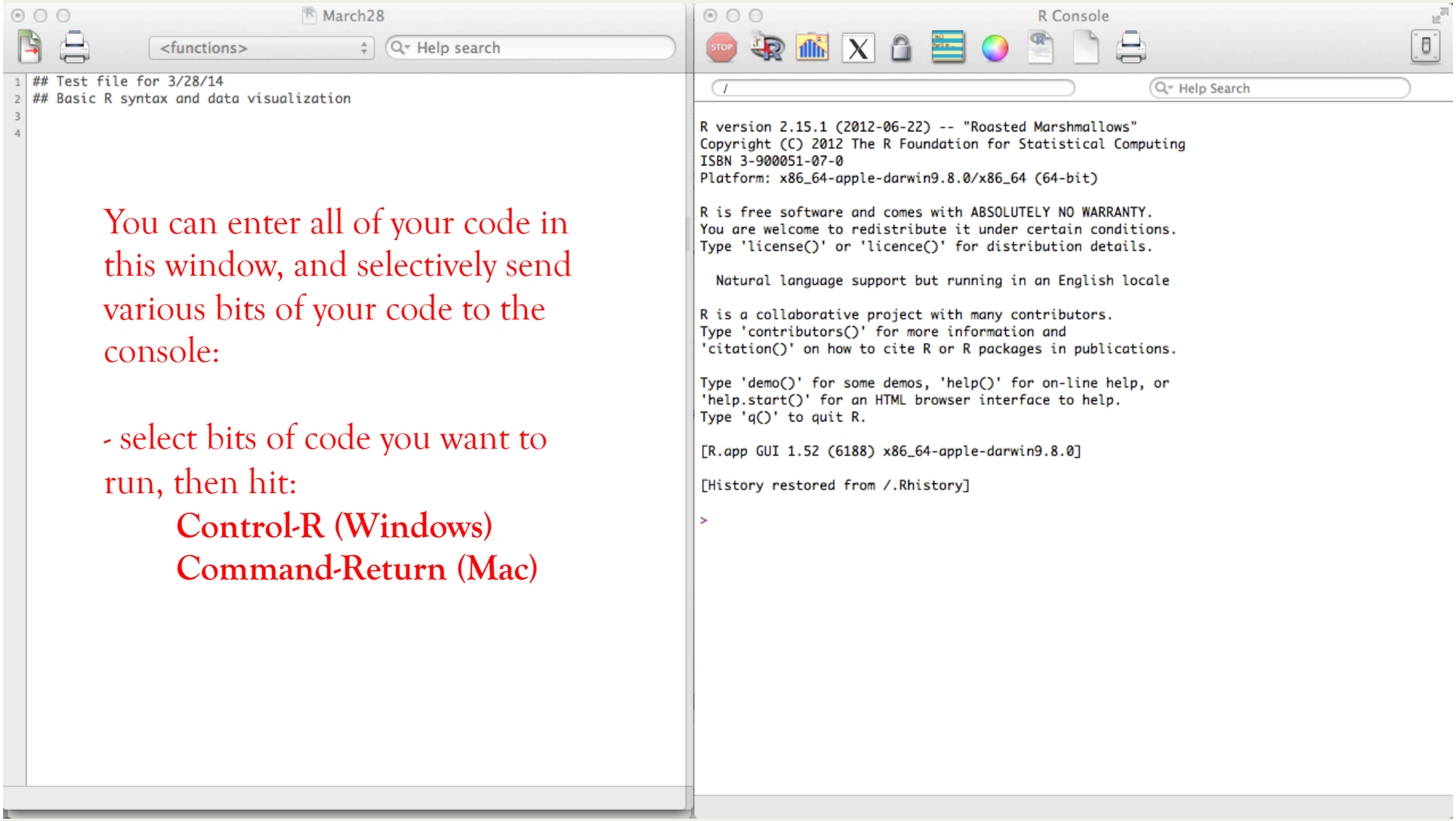
You can type your commands directly into the console window

But, it is usually better to hit:
Control-N (Windows)
Command-N (Mac)

This opens up a new R file.



The R environment....



A few warnings/cool things about R

- It is a case-sensitive language
  - **`DARWIN`**, **`Darwin`**, and **`darwin`** will be treated as three different objects

- It is syntactically flexible
  - e.g., you can assign new objects using several methods
    (these methods are largely equivalent, though there are a few exceptions with the equals sign so try to stick with the arrows):

    `->` **`<-`** **`=`**

- It comes with a wide variety of built-in functions

- An even wider variety of awesome packages can be downloaded too, but each often has its own weird syntactic quirks and error messages (or lack thereof)
  - **Use new/unfamiliar packages at your own risk!**

Commenting & R Documentation

In order to comment out a section of your code, use **#**:

```
# this code won't run, even if you select it and send it to the console
 (but it will appear in the console as text)
```

```
?function.name
    # this opens up a documentation file for a function named function.name
```

```
example(function.name)
    # provides you with several examples of this function and gives you the
    code to produce those examples
```

Exercise:

1) Bring up the documentation file for the **plot** function

2) Bring up examples of the **plot** function

Creating and manipulating vectors

VECTOR: 1D objects that contain one data type (numeric, logical, character, integer.... etc).

```
c('a','f','z')
    # creates a vector of these characters

c(1,200,3,5540)->numbers.galore
    # create a vector of these numbers and assign it to the object
    called numbers.galore

seq(1,20)->one.to.twenty
    # create a sequence of numbers from 1 to 20, assign them to the
     object called one.to.twenty

one.to.twenty[7]
    # return the 7th index in the vector one.to.twenty
    # NOTE: R is a little weird, it starts its indices at 1, NOT 0
```

Exercise:

1) Return the 8th index in a vector you create named ***icancount*** that contains a sequence of numbers from 8 to 1000.
2) Now try creating a vector using this code: **c('a', 0, FALSE)->test**
3) What does R return if you type **test** in the console and hit enter?
4) Now try **class(test)** to get information on what data type the vector is.

Creating and manipulating matrices

MATRIX: 2D arrays which can contain only one data type (usually numeric)

```
matrix(
c(2,3,52,10,6,1,100,1,2),      # combine these data elements
nrow=3,                        # use this many rows
ncol=3,                        # use this many columns
byrow=TRUE)->a                 # assign this matrix to object a


print(a)                       # print matrix a
a                              # print matrix a
```

LET'S GET A TASTE OF R'S VISUALIZATION CAPABILITIES

Exercise:

1) Use the code above to make matrix a
2) Then type in:

```
persp(a, col="gray")          # creates a perspective plot of matrix a
```

3) Now try:

```
image(a)                      # creates a heat map of matrix a
```

4) Now try:

```
example(persp)
example(image)
```

Subsetting 101

Subsetting is the key to mastering R!

5 ways to subset

1) Blank – include everything
2) Positive integer – include specific elements
3) Negative integer – exclude specific elements
4) Logicals – include TRUE, exclude FALSE
5) Names – include named elements

We'll be using R-blocks to learn about subsetting and to explore data types a bit more

INSTALL R-BLOCKS:

```
lapply(FUN=install.packages,c("devtools"))
devtools::install_github('rblocks', 'ramnathv')
library('rblocks')
```

Subsetting Vectors

Including everything

```
b_vector <- make_block(5, type = "vector")
    # create a vector block called b_vector

b_vector
    # print b_vector
```

Leave [ ] blank = include everything:

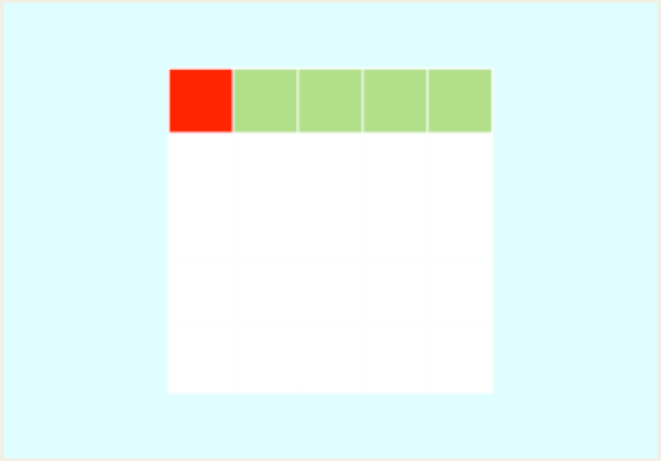
```
display(B_vector[])
```

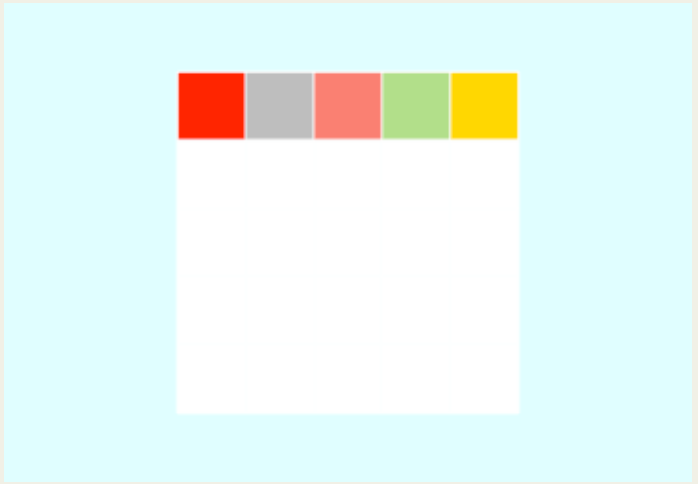
Subsetting Vectors

Including specific element(s)

Positive integer = include selected element

```
b_vector[1]<-"red”
    # make first index in b_vector red
b_vector

b_vector[c(5,2,3)]<-c("gold","gray","salmon")
b_vector
```



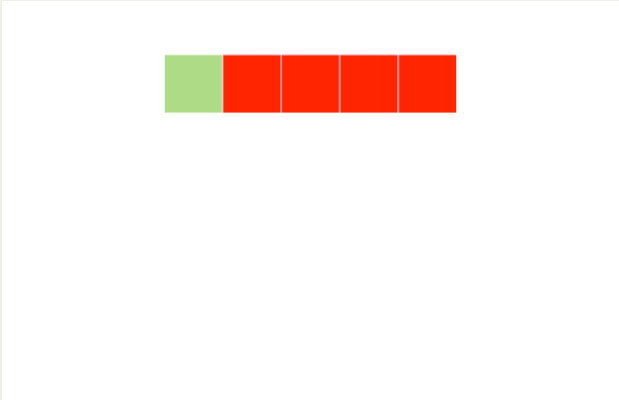


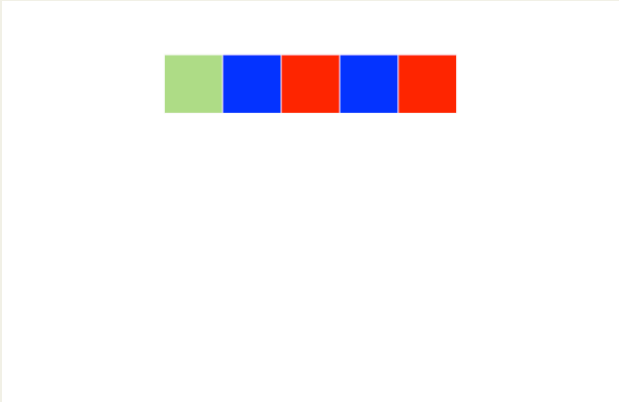
Subsetting Vectors

Excluding specific element(s)

Negative integer = exclude selected element

```
b_vector <- make_block(5, type = "vector")
b_vector[-1]<-"red"
b_vector
b_vector[-c(1,3,5)]<-"blue"
b_vector
```



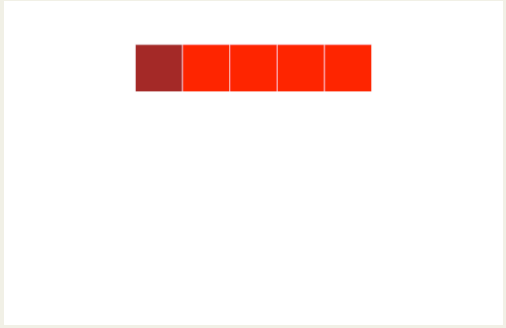


Logicals - include TRUE

```
B_vector

b_vector=="blue”
>>> [1] FALSE  TRUE FALSE  TRUE FALSE

b_vector[b_vector=="blue"]<-"red"
b_vector
```

Names – include named element

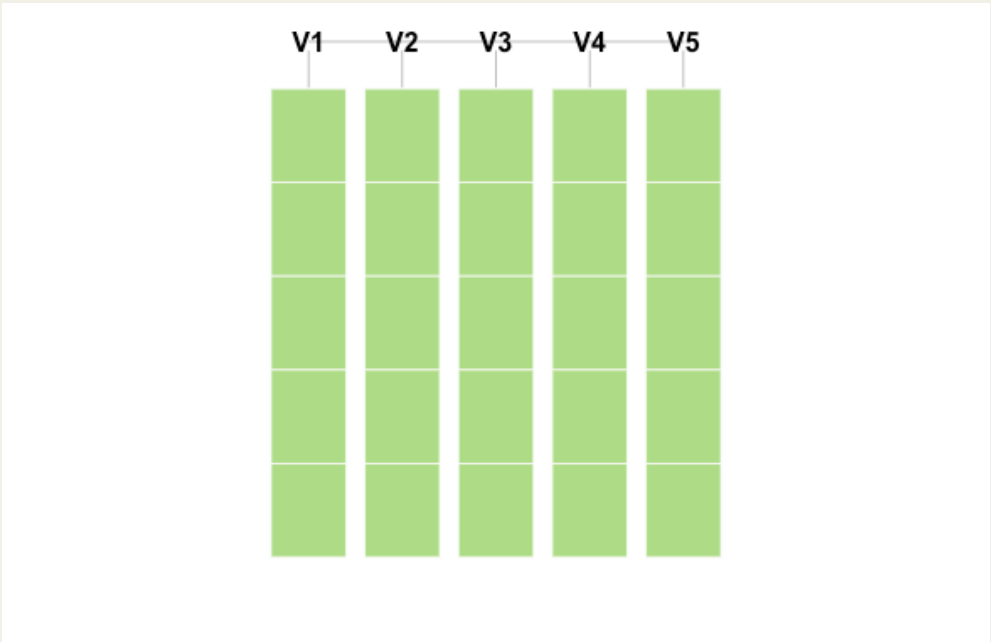
```
names(b_vector)<-c("dog","cat","chicken","cow","monkey")
b_vector[] #now we have named our vector
 >>>      dog           cat       chicken       cow    monkey
 >>> "#b2df8a"     "red"     "red"     "red"     "red"

b_vector["dog"]<-"brown"
```



Subsetting Dataframes

- Data frames are 2D objects that can contain different data types in separate columns.
  - Most commonly used and most important data structure

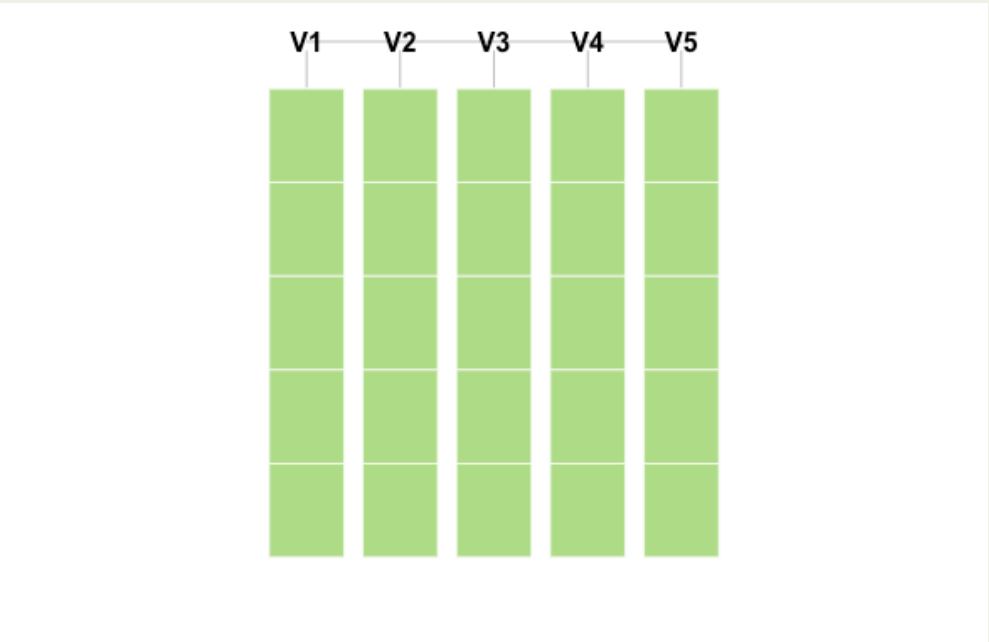
```
b_dataframe <- make_block(5, 5, type = "data.frame”)
```



Subsetting Dataframes

- Leave [ ] blank = include everything
  - R uses a [Rows, Columns] convention
  - So [,] means include all rows all columns

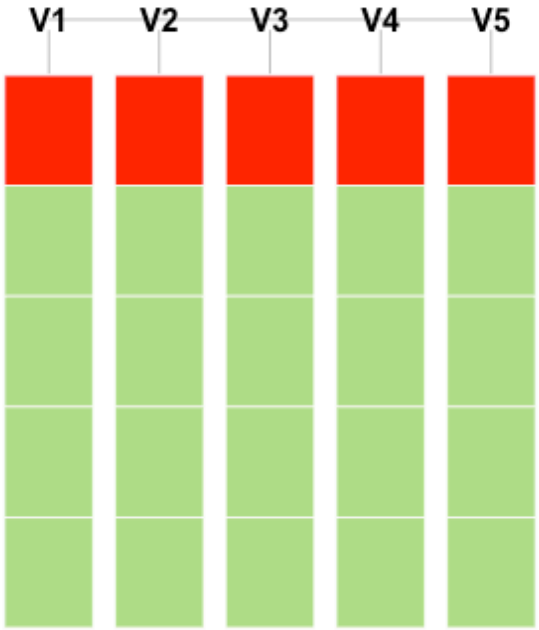
```
display(b_dataframe[,])
```



Subsetting Dataframes *Include*

Entering integers within the brackets = include specific rows or columns

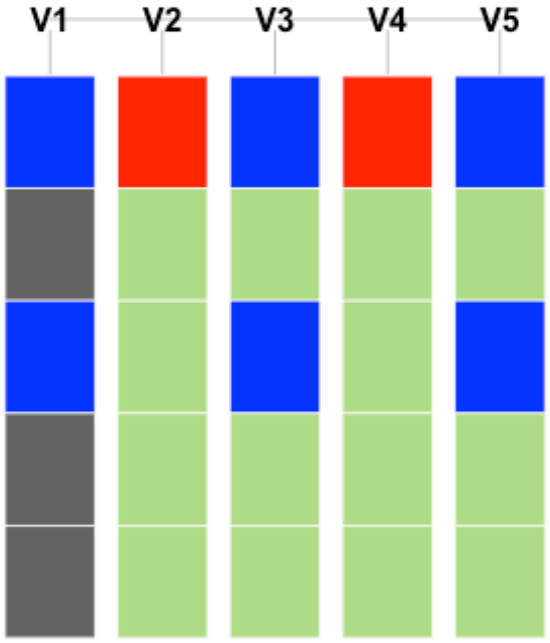
```
b_dataframe[1,]<-"red"
    # first row, all columns red

b_dataframe
```



Subsetting Dataframes

Include

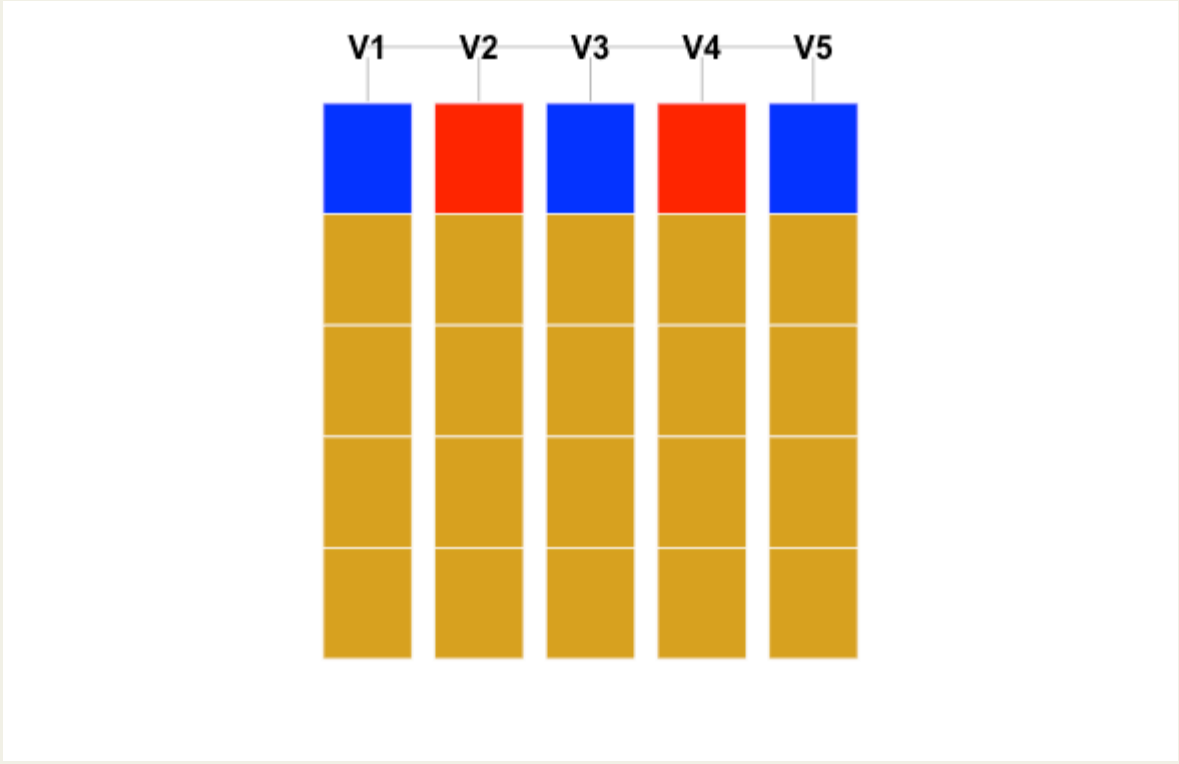
```
b_dataframe[c(1,3),c(1,3,5)]<-"blue"
    # specific rows and column selections

b_dataframe
```



Subsetting Dataframes *Exclude*

Negative Integers = exclude specific rows or columns

```
b_dataframe[-1,]<-'goldenrod'
    # exclude the first row all other rows and columns red

b_dataframe
```

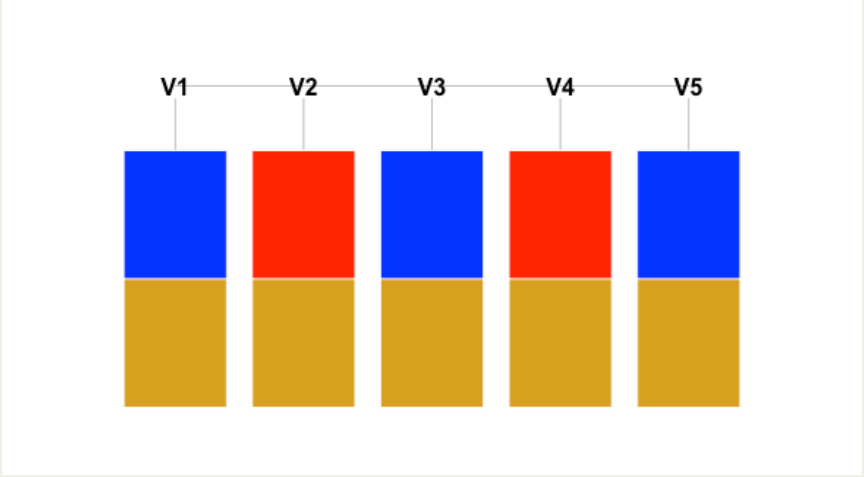


Logicals - include TRUE

```
b_dataframe$animals<-c("cat","cat","dog","dog","dog")
as.data.frame(b_dataframe)
```

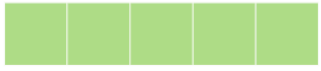
	V1	V2	V3	V4	V5	animals
1	blue	red	blue	red	blue	cat
2	Goldenrod	goldenrod	goldenrod	goldenrod	goldenrod	cat
3	goldenrod	goldenrod	goldenrod	goldenrod	goldenrod	dog
4	Goldenrod	goldenrod	goldenrod	goldenrod	goldenrod	dog
5	goldenrod	goldenrod	goldenrod	goldenrod	goldenrod	dog

```
b_dataframe[b_dataframe$animals=="cat",1:5]
```



Names – include named component

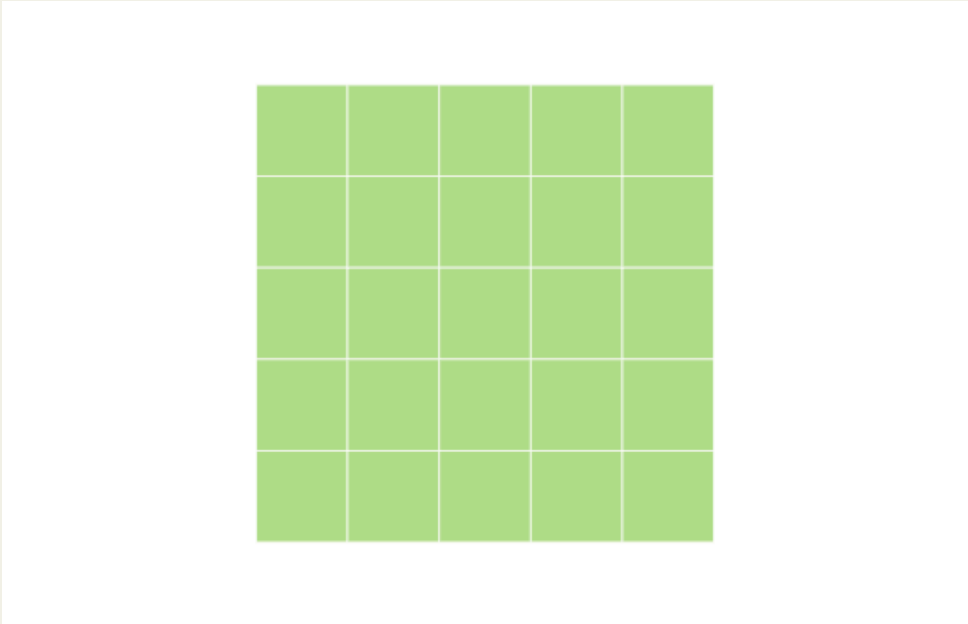
```
b_dataframe <- make_block(5, 5, type = "data.frame")
display(b_dataframe$V1)
display(b_dataframe[,"V1"])
display(b_dataframe[,"V1",drop=FALSE])
```



Subsetting Matrices

Subsetting them is pretty much exactly the same as a dataframe so we won't do that here, but try it out on you own!

```
b_matrix <- make_block(5, 5, type = "matrix")
b_matrix
```

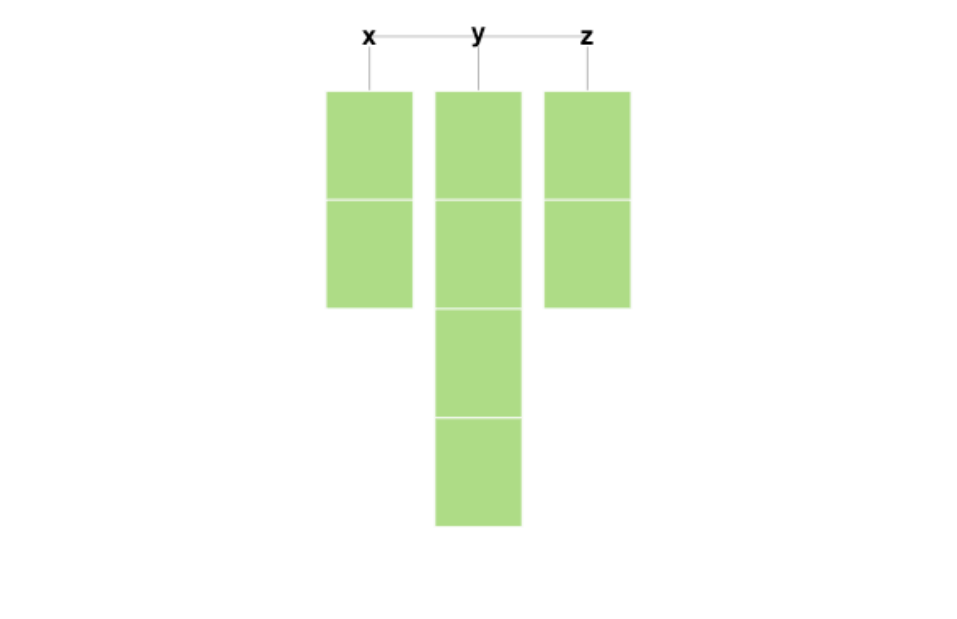


Subsetting Lists

Lists are super flexible data structures that can contain anything, even other lists.

- Mixing of different types, file structures, objects etc., is fine!

```
b_list <- make_block(list(x = 1:2, y = LETTERS[1:4], z = c(T, F)))
b_list
```

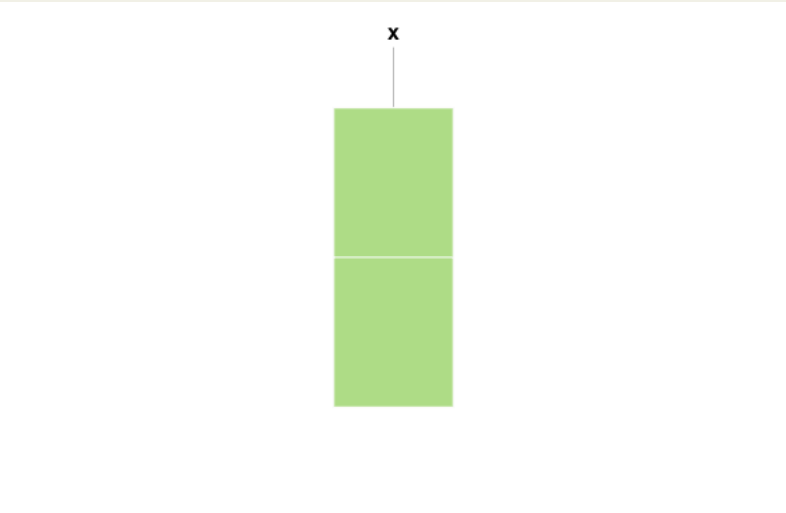


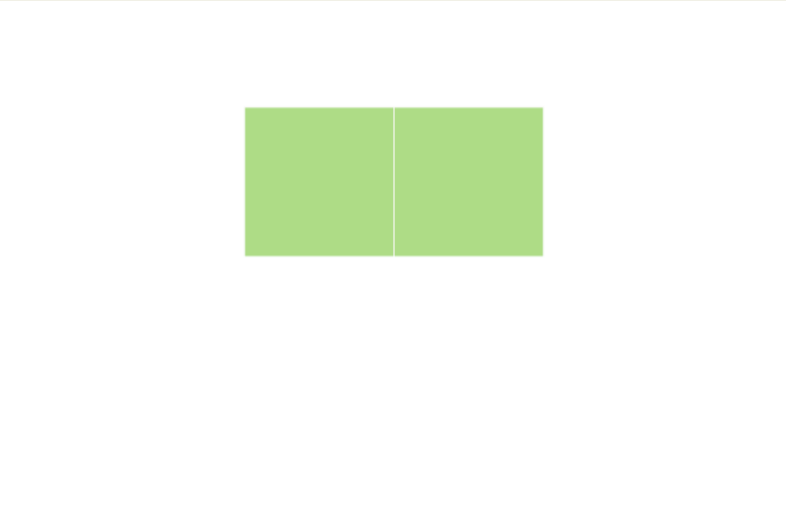
Subsetting Lists

Include

Remember, positive integers means include

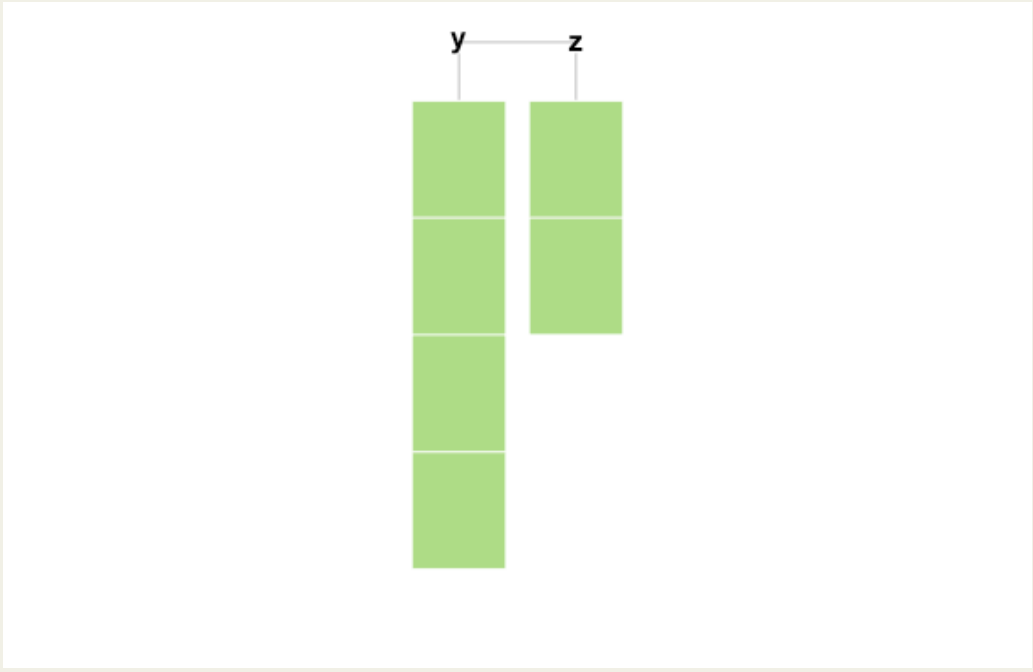
```
display(b_list[1])
display(b_list[[1]])
```





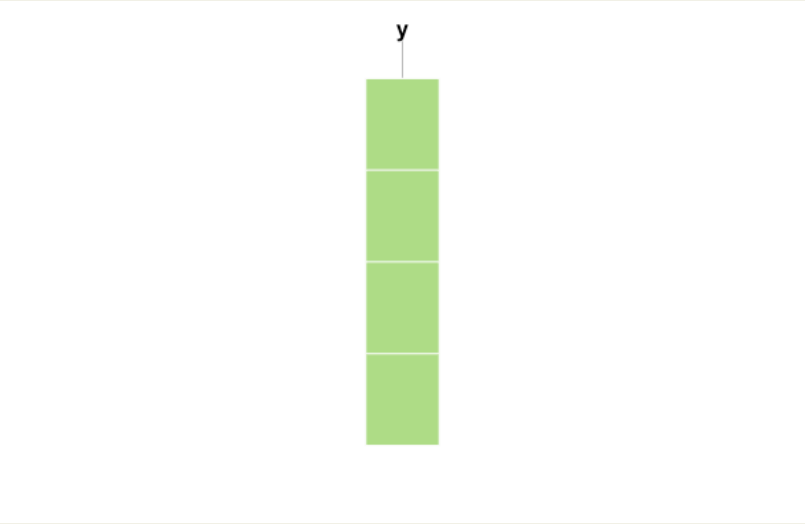
Negative integers means exclude

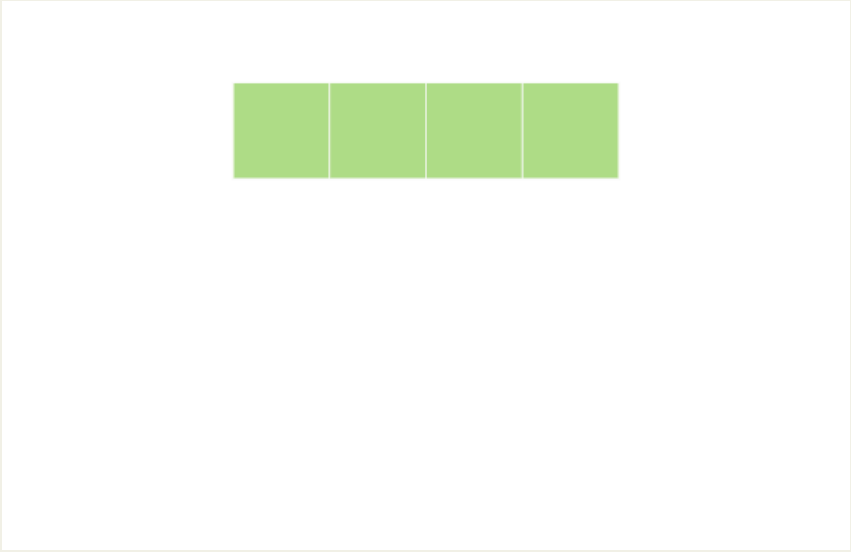
```
display(b_list[-1])
```



Names – include named element

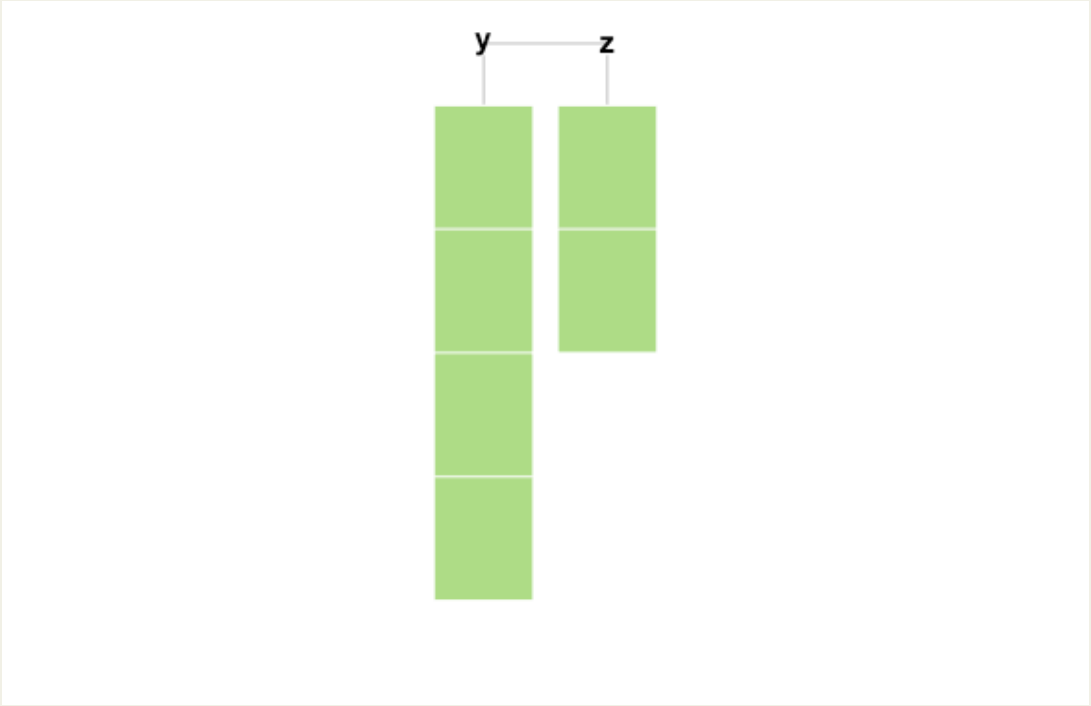
```
display(b_list["y"])
display(b_list[["y"]])
```





Logicals – include TRUE

```
names(b_list)!="x”
>>>>> [1] FALSE  TRUE  TRUE

display(b_list[(names(b_list)!="x")])
```



Reading in a dataset from Excel

- Let's set R's working directory:
  - Control-D (Windows)
  - Command-D (Mac)
  - Go to the folder that contains the Excel .csv file

- Now enter:

```
read.csv("sampledata.csv")->mydata
    # this makes R read in that comma delimited data to a
    dataframe called mydata
```

Getting to know your data

```
fix(mydata)
    # get a nice little data editing window to pop up
    # need to close the window once you're done looking at it

names(mydata)
    # give me the names of each of the columns in mydata

mydata$variable1
    # returns the vector within the dataframe mydata called variable1

head(mydata)
    # returns first part of mydata
```

Exercise:
From now on we'll be using one of R's built-in datasets called mtcars

1) What happens if you type in **mtcars[,1]** ?
2) What happens if you type in **mtcars[,1:2]**?
3) What happens if you type in **mtcars[1,]** ?

Remember, R uses a row, column convention

Basic Operations

- How to add two vectors within a dataframe:

```
mydata$variable1+mydata$variable2
```

- How to multiply a vector by a constant:

```
mydata$variable1*10
```

- There are many constants built-in to R (e.g., `pi`)

Exercise:

1) Multiply the first two vectors in the **mtcars** dataset by each other.

2) What happens if you subtract **pi** from **mtcars$mpg**?

Let's use subsetting to create a new dataframe

```
mydata[mydata$variable1>0,]-> mydata.subset

    # create a new subsetted dataframe of mydata, only including entries
    where the value of variable1 is greater than 0, and assign it to
    mydata.subset
```

Exercise:

1) Create a subset of the ***mtcars*** dataset that only includes entries where the vector containing "miles per gallon" information is greater than 20.

2) Name the new data subset ***efficient,*** and print ***efficient*** in the console

Creating a new categorical or numeric variable using if-else

```
ifelse(mydata$variable>x, 1, 0)
    #if entries in variable exceed x, assign them a 1, if not, assign them a 0

ifelse(mydata$variable>x, 'a', 'b')->newcategorical.variable
    # can also create factors by using ' ' around words
```

Exercise:

1) Using the ifelse function, create a new categorical variable called ***fuelefficient*** that has two categories:
   - 'efficient' for cars with mpg above 20
   - 'gas.guzzler' for cars with mpg below 20

Let's get to visualizing!

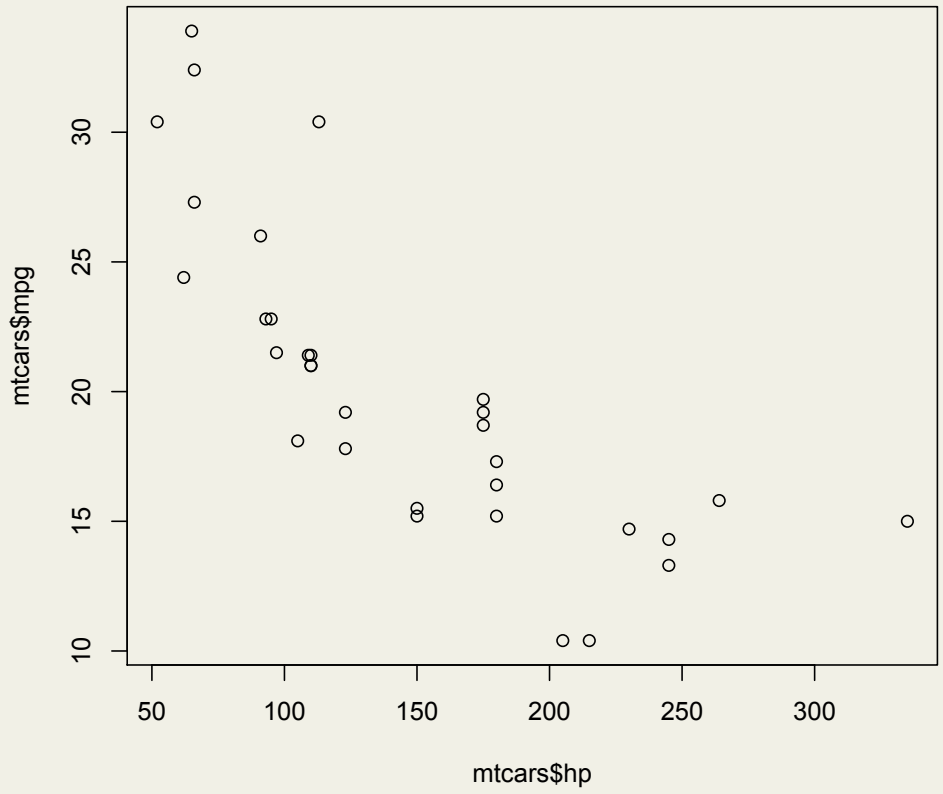
```
plot(mydata$y.variable ~ mydata$x.variable)
plot(mydata$x.variable, mydata$y.variable)
    # make a scatterplot of y.variable as a function of x.variable
```

These mean the same thing to R

Exercise:

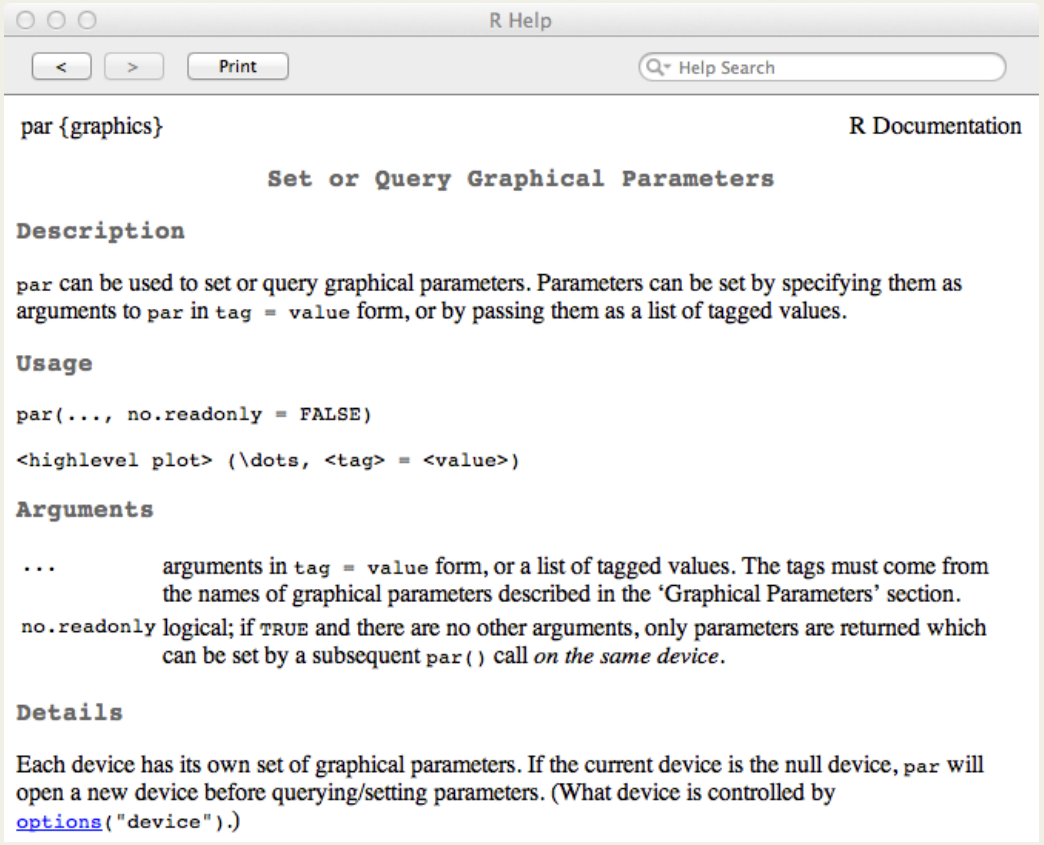
Use the **mtcars** dataset to plot miles per gallon as a function of horsepower.
(HINT: use ?mtcars to learn more about the dataset)

Not too pretty, is it? Let's make it a little more informative



Plot Aesthetics

```
?par
    # open up the documentation
file for the graphics parameters
```



R Help

< > Print Help Search

par {graphics} R Documentation

Set or Query Graphical Parameters

Description

`par` can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to `par` in `tag = value` form, or by passing them as a list of tagged values.

Usage

```
par(..., no.readonly = FALSE)

<highlevel plot> (\dots, <tag> = <value>)
```

Arguments

`...`	arguments in `tag = value` form, or a list of tagged values. The tags must come from the names of graphical parameters described in the 'Graphical Parameters' section.
`no.readonly`	logical; if `TRUE` and there are no other arguments, only parameters are returned which can be set by a subsequent `par()` call *on the same device*.

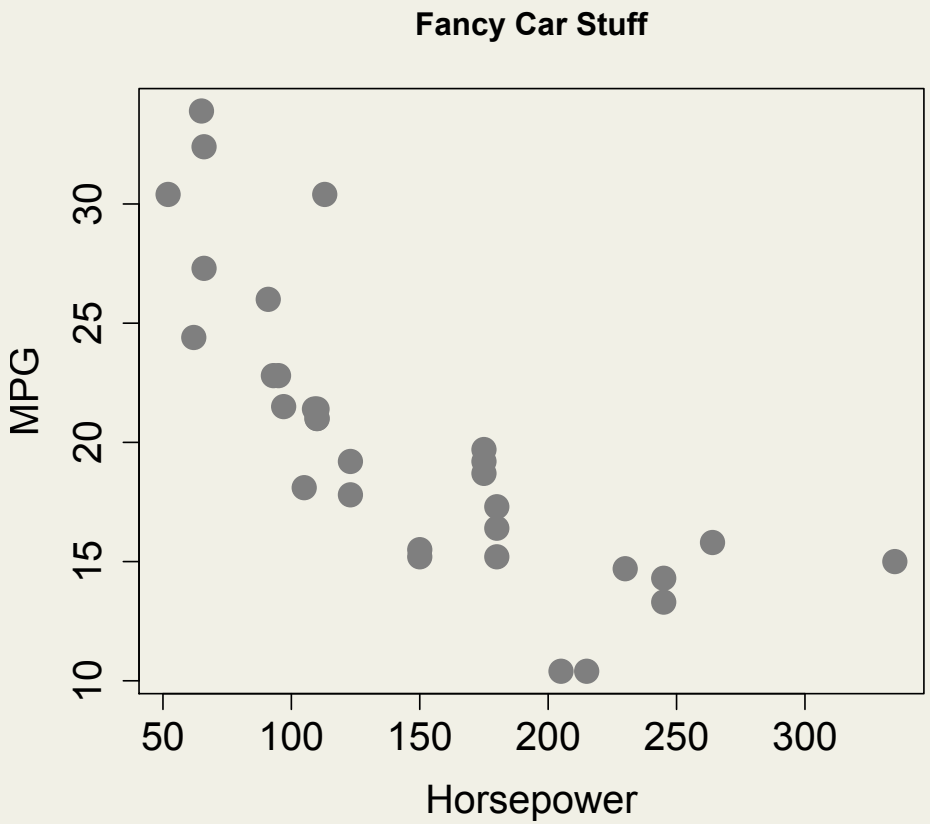
Details

Each device has its own set of graphical parameters. If the current device is the null device, `par` will open a new device before querying/setting parameters. (What device is controlled by `options("device")`.)

```
col
    # specify color

pch
    # specify point shape

cex
    # specify point size

main
    # title of plot

xlab
    # x axis label

ylab
    # y axis label
```

Scatterplots

```
plot(mtcars$mpg~mtcars$hp,
col="gray50", pch=19, cex=2,
cex.axis=2, main="Fancy Car
Stuff", ylab="MPG",
cex.lab=2,
xlab="Horsepower")
```



Exercise:

Create a scatterplot of mpg and horsepower using green points and that only plots cars with horsepower greater than 150

HINT: If you click on the Console, and hit the up arrow, the console will show the last command you entered, which you can then go in and edit

Plotting best-fit lines

- A quick regression:

```
lm(mtcars$mpg~mtcars$hp)->model1
    # regressing mpg on horsepower, saving resulting model to object model1

summary(model1)
    # provides you with summaries of any object, in this case, your linear
    regression model1

plot(model1)
    # plots useful validation plots of your model (hit ENTER to toggle
    through them
```

- Now let's plot the scatterplot and the line for the model:

```
plot(mtcars$mpg~mtcars$hp)      # don't close the plot window!
abline(model1)                  # plots the line for the regression
abline(v=200, col="red")
    # plots a vertical line at the value of 200 on the x-axis
```

Multi-panel plots

```
par(mfcol = c(1, 2))
    # open a graphics window, that will have one row, two columns. Don't
    close this empty window!

plot(mtcars$mpg~mtcars$hp)
    # first plot

boxplot(mtcars$mpg~mtcars$am)
    # second plot
```

Exercise:

Create a 3 row, 1 column graphic with three different plots of your choice. Try adding axis labels and color to each of them.

If you're feeling fancy: ggplot2

```
install.packages("ggplot2")
    # install the package ggplot 2
    # can also use the Package Installer

library(ggplot2)
    # loads the ggplot2 package

?ggplot2::diamonds
    # call documentation file for the diamonds dataset within the
    ggplot2 package

qplot(x, y, data = mydata)
```

Try this:

```
qplot(carat, price, data = diamonds, colour = color)
```

Many other plots can be done with the base R graphics

```
boxplot(mydata$y.variabley~mydata$x.variable)
    # plots a numerical y-variable as a function of a factor x-variable

dotchart(mydata$variable)
    # plots a Cleveland dot plot (a scatterplot for one variable)

pairs(dataframe)
    # creates scatterplots of all of the variables within a dataframe

barplot(table)
    # creates a barplot using a table
```

Free Resources, Classes, Tutorials

tryr.codeschool.com

Teaches the basics of R in an interactive way (*Codeacademy* imitation)

Free Resources, Classes, Tutorials

Roger Peng's 4 week long R Class

Totally free, starts April 7th, and May 5th, and June 2nd!

JOHNS HOPKINS
BLOOMBERG SCHOOL of PUBLIC HEALTH

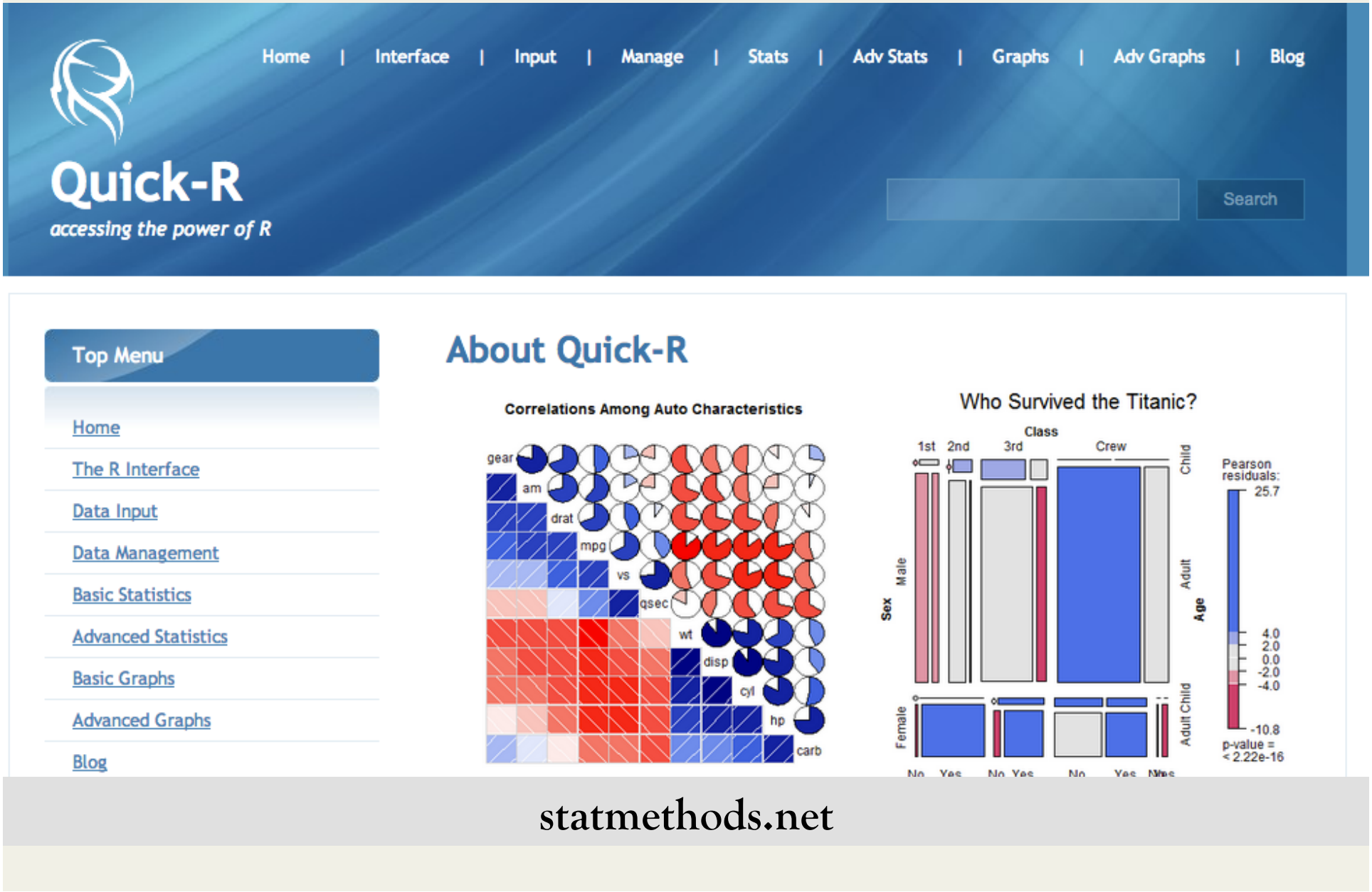
R Programming

Part of the "Data Science" Specialization »

Learn how to program in R and how to use R for effective data analysis. This is the second course in the Johns Hopkins Data Science Specialization.

www.coursera.org/course/rprog

Free Resources, Classes, Tutorials

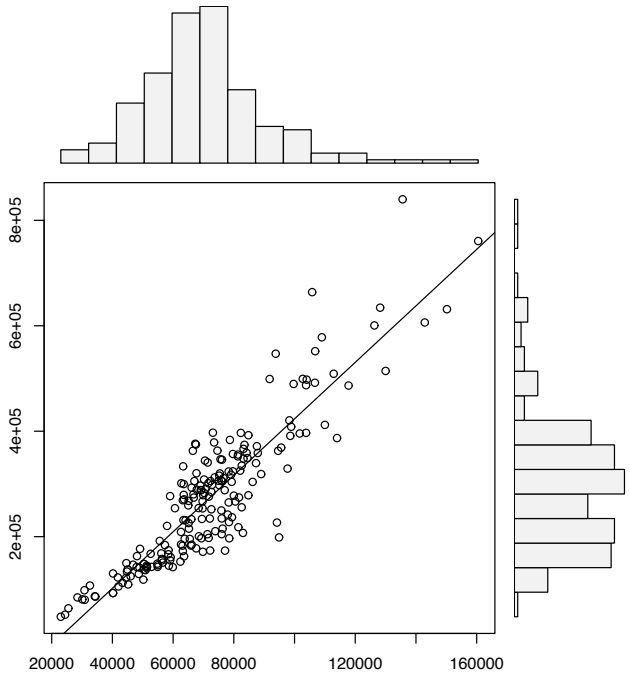


statmethods.net

Free Resources, Classes, Tutorials

simpleR – *Using R for Introductory Statistics*

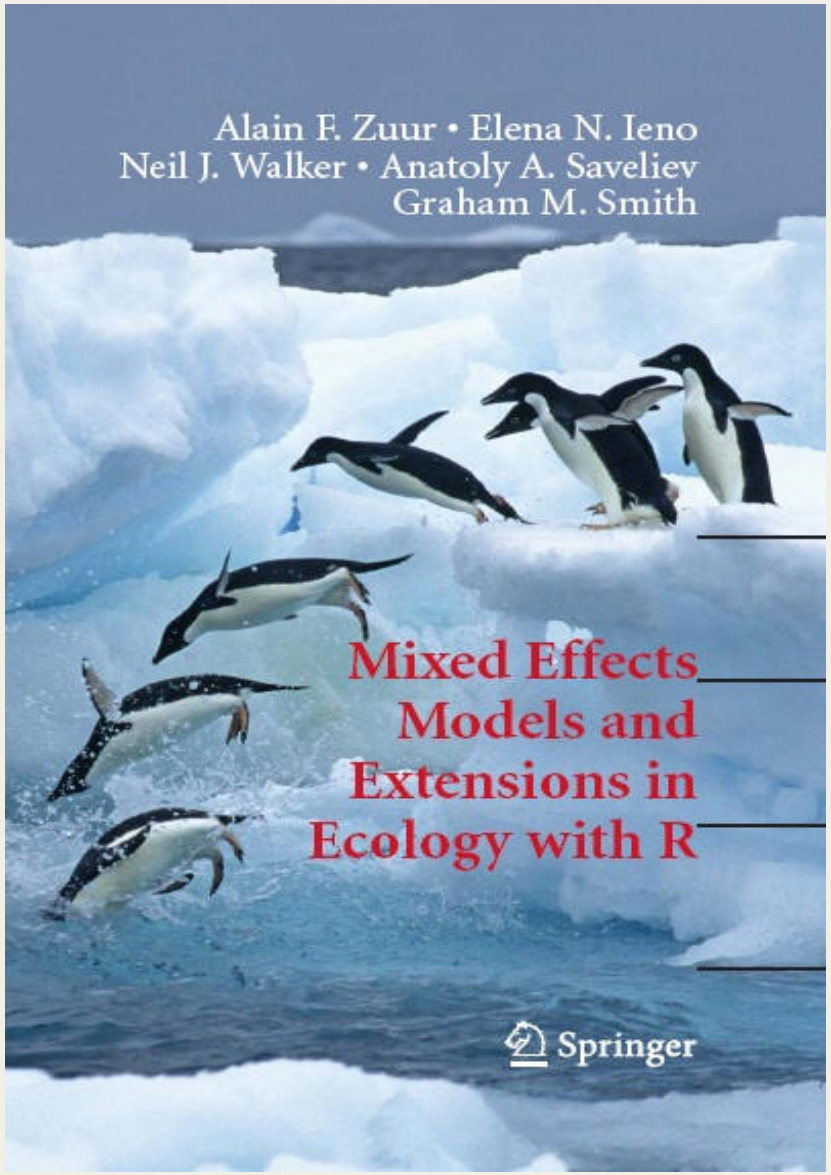
John Verzani



Free R book

Download here:
ub.edu/stat/docencia/EADB/simple-12-oneside-letter.pdf

Free Resources, Classes, Tutorials



Teaches linear regression, GLM, GAM, GEE, GLMM, in R, using ecological examples

Comes with sample R code for every chapter of book

Download PDF through UT Austin library