Colleen Roe
2019 ITMD Annual Workshop
April 17, 2019

An Introduction to the R Programming Language

Goals of this Session

- Introducing the R “ecosystem”
- Giving you the information you need to get R and related software installed
- Compare and constrast the R and “standard” programming languages like Java
- Provide a high level overview of major language structures/features
- Demonstrate how to use R effectively
- Point out some available resources to help you learn more

Agenda

- Getting set up to work in R technologies
- R Packages: how to find and use them
- Understanding how R is different
- The basics of programming in R
- Handling simple file IO in R
- Actually programming in R
- Wrap up and questions

Getting Started in R Technologies

R Project for Statistical Computing

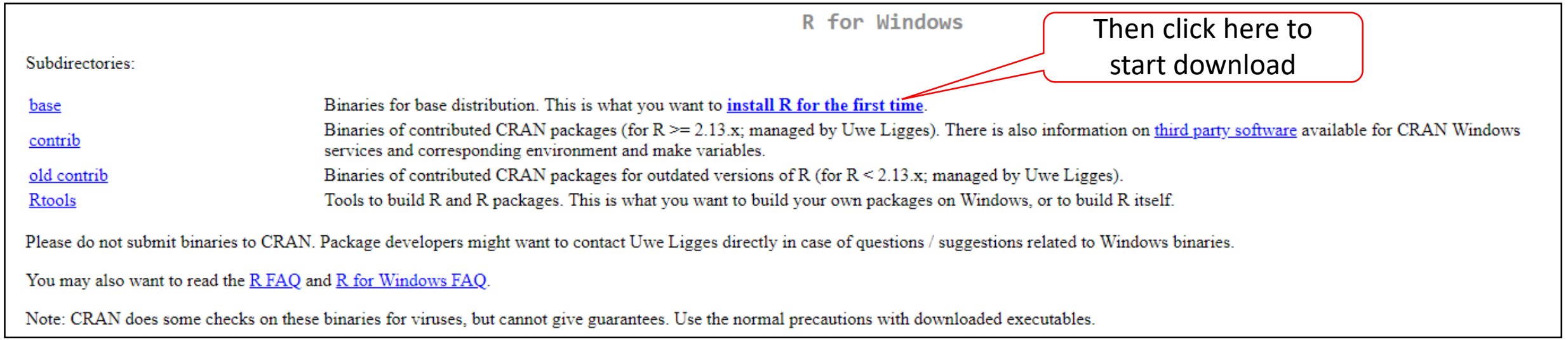
- R is focused on statistical and graphical computation and is supported by the *R Foundation for Statistical Computing*
- R is a free programming language developed and maintained by a world wide group of scientists and engineers
- The R website is: https://r-project.org from this page links are available for R downloads for Linux, Windows, and MacOs
- If you need help, the world wide R users community can provide some mighty fine help so join in and access it
- There are R users groups in larger cities all over the world, including Portland (https://www.meetup.com/portland-r-user-group/), that sponsor talks, seminars, and events

Other Important R Resources

- RStudio is a free and very popular R software development environment (SDE): https://www.rstudio.com/products/rstudio/download/
- If you use Eclipse, the StatET SDE plugin is also available: http://www.walware.de

Getting Set Up

- R itself is available to down load from the CRAN* site:
  - https://cran.r-project.org/mirrors.html
  - Pick a mirror site geographically close
  - Click on the URL
  - On the download page, click on the link: Download R for Windows (or Linux or MacOS)
  - You will get this page:



*Comprehensive R Archive Network

Getting Set Up

Next page:

Click here. Version number may vary.

R-3.5.3 for Windows (32/64 bit)

Download R 3.5.3 for Windows (79 megabytes, 32/64 bit)

Installation and other instructions

New features in this version

After download, read these instructions and install

If you want to double-check that the package you have [illegible] uted by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master serve windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.htm.

R Packages: how to find and use them

R Packages

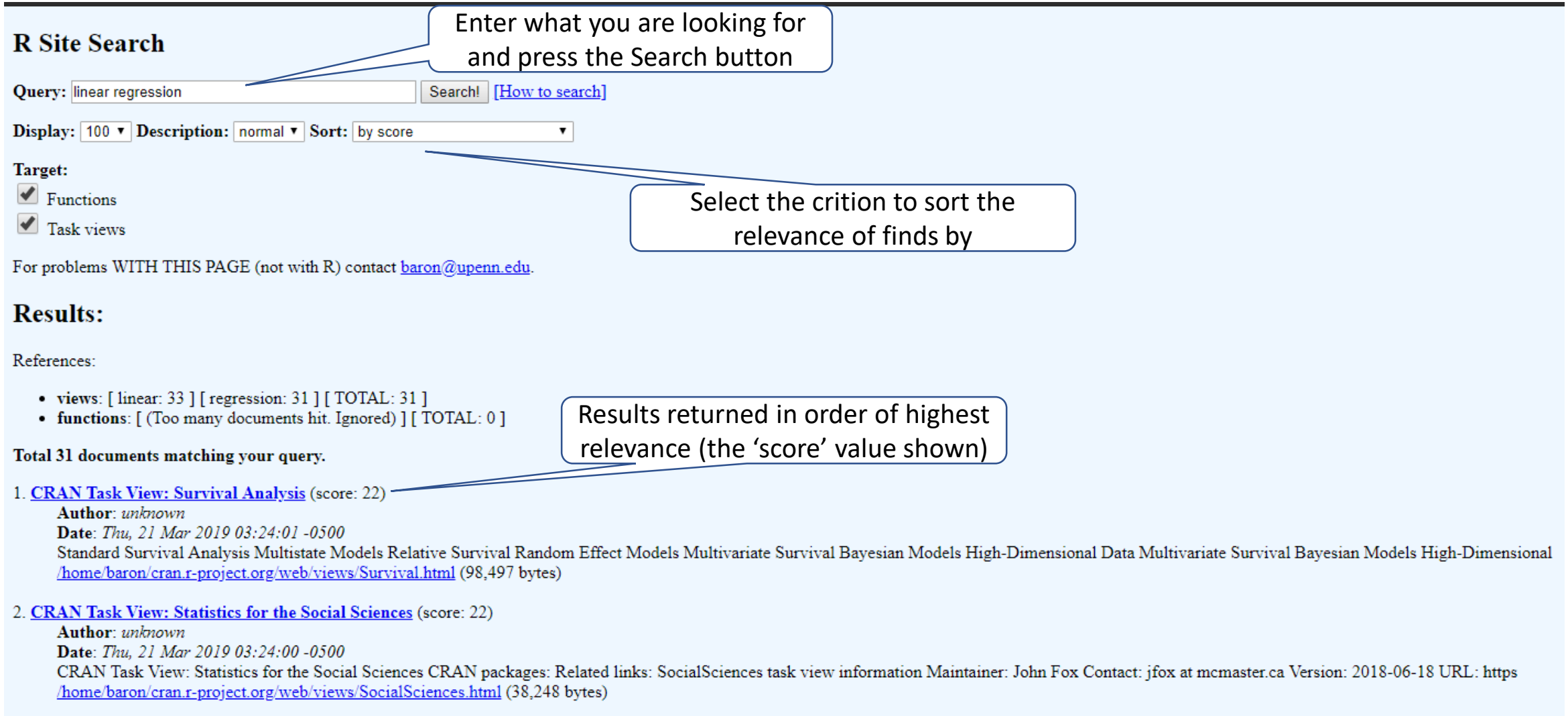
- What is a package?
  - Packages are collections of R functions, data, documentation, and compiled code in a well-defined format that implement a specific functionality, for example: ANOVA
  - The directory in which packages are stored is called the “library” (`C:\Program Files\R\R-3.4.3\library`)
  - R installs with a standard set of packages which provide much functionality
  - Others packages are available to download and install at your option
  - Once installed, they have to be loaded into the session to be used (more on this later)
  - Most packages implement mathematical, statistical and/or graphical functionality
- In addition to the basic language, R has over 15,000 add-on packages and this number is increasing rapidly
  - Packages add to R core functionality
  - Effectively, packages allow R to serve as an extensible language

R Packages

- Anyone can write a package for R
- Packages are carefully curated by a board of scientists:
  - There are standards for what a package must contain (code, documentation, examples, etc.)
  - Submissions must meet these standards to even be considered
  - Functions are vetted by board members and others to ensure scientifc correctness and code quality
- This vetting process ensures that only quality packages become available to the R community
- BONUS: Authors of packages often directly answer requests for help – another great resource of the R community

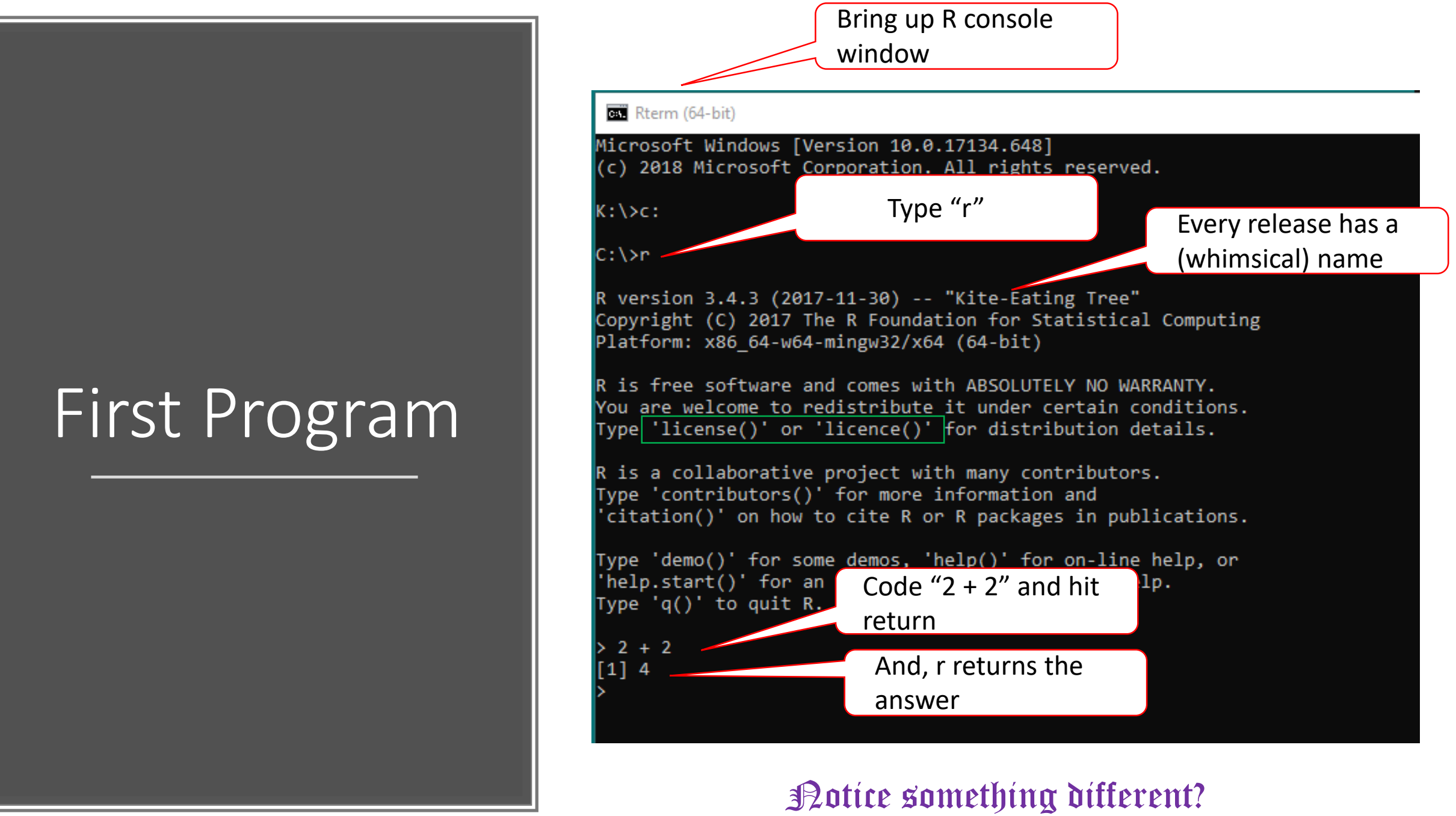
Finding a Suitable Package: The R Search Site

Go to http://finzi.psych.upenn.edu/search.html - the R search site



Understanding how R is different

First Program



Notice something different?

Differences: R vs Java or C#

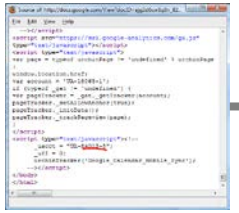
Actually, there are a lot of differences:

R	Java or C#
• Object oriented • *Interperted* • *Vector or scalar* • *Dynamic typing* • *Malleable function invocation* • *Images*	• Object oriented • Compiled • Scalar • Declarative typing • Strict function invocations

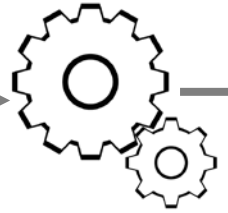
Interpreted vs Compiled Software

Compiled Language

Code File

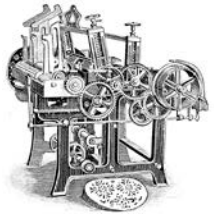


Compiler

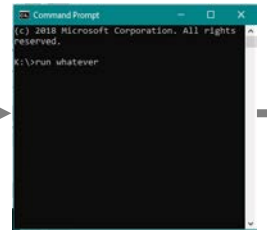


Binary File

Create Runnable



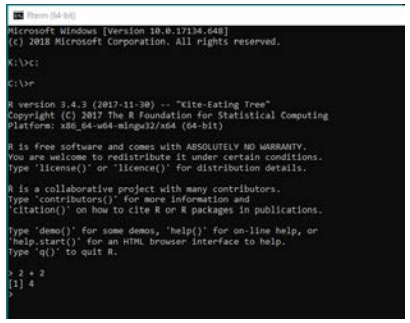
Run Time Environment



Result

Interpreted Language

Interpreter



Result

R Features

Dynamic Typing

Note that the type of z changed between assignments

```
x <- c(5,5,2)
> y <- c(8,5,14)
>
> z <- sum(x)
> print(z)
[1] 12
>
> z <- sum(y)
> print(z)
[1] 27
>
> z <- x + y
> print(z)
[1] 13 10 16
>
> print(mean(y))
[1] 9
> print(sd(x))
[1] 1.732051
>
```

Create two vectors

Sum each vector and print this value

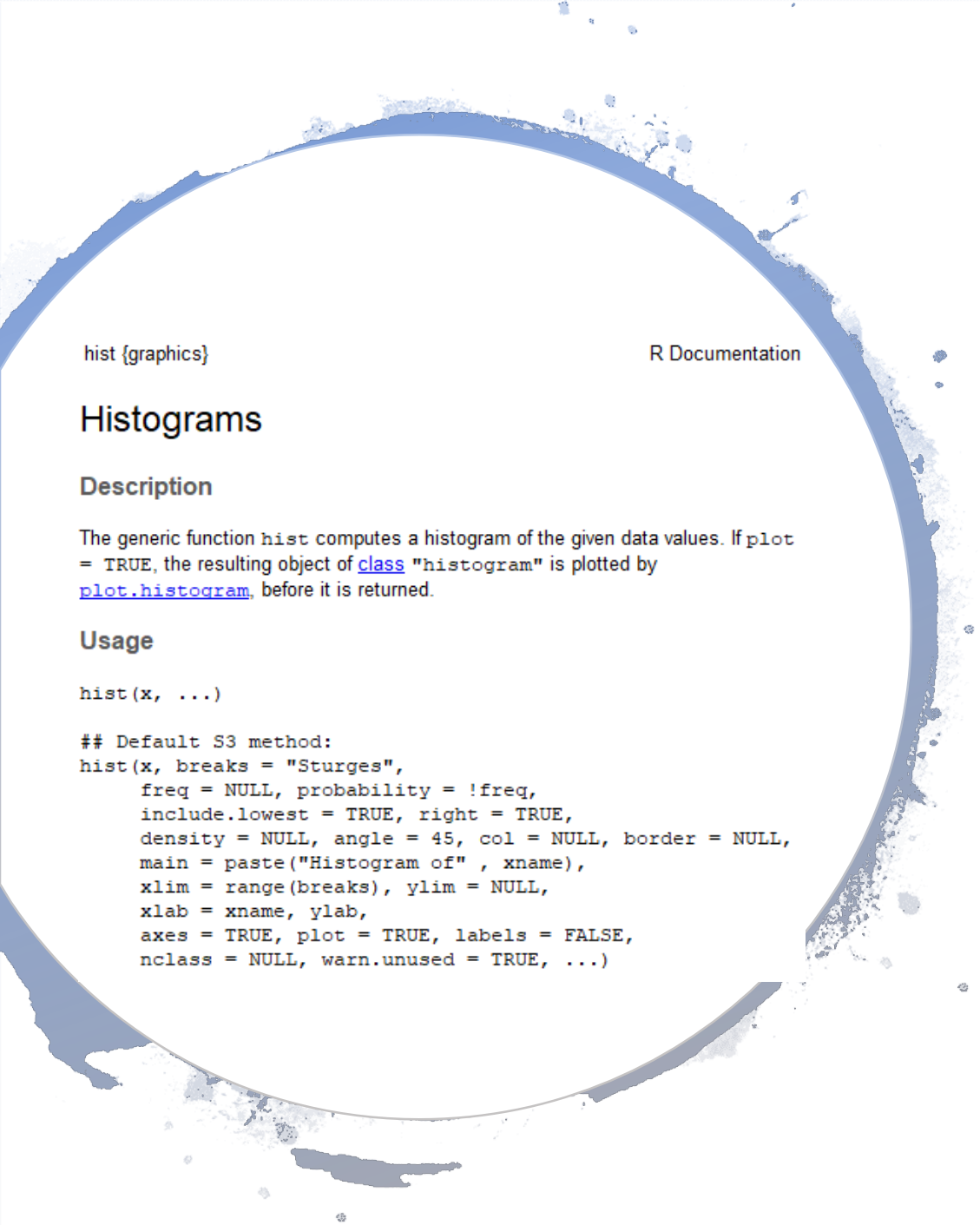
Add the two vectors and print new value

Calculate the mean and std dev and print

Vector Operations

(less looping and simpler code)

The “>” is the command line prompt



R Function Calls

- Define pData as:
- `pData <- 1:10`
- The following are all valid invocations of the `hist` function
- 1. `hist(pData)`
- 2. `hist(x = pData)`
- 3. `hist(plot=TRUE, x = pData)`
- 4. `hist(pData, plot=TRUE,axes=FALSE)`
- The following rules apply:
  - As long as you enter data in the order defined, you do not have to name the parameter (1,4)
  - If you are satisified with the default value of the parameter, you can skip it (1,2,3,4)
  - You can enter parameters in any mixed order if you specifically name them (3,4)

Histograms

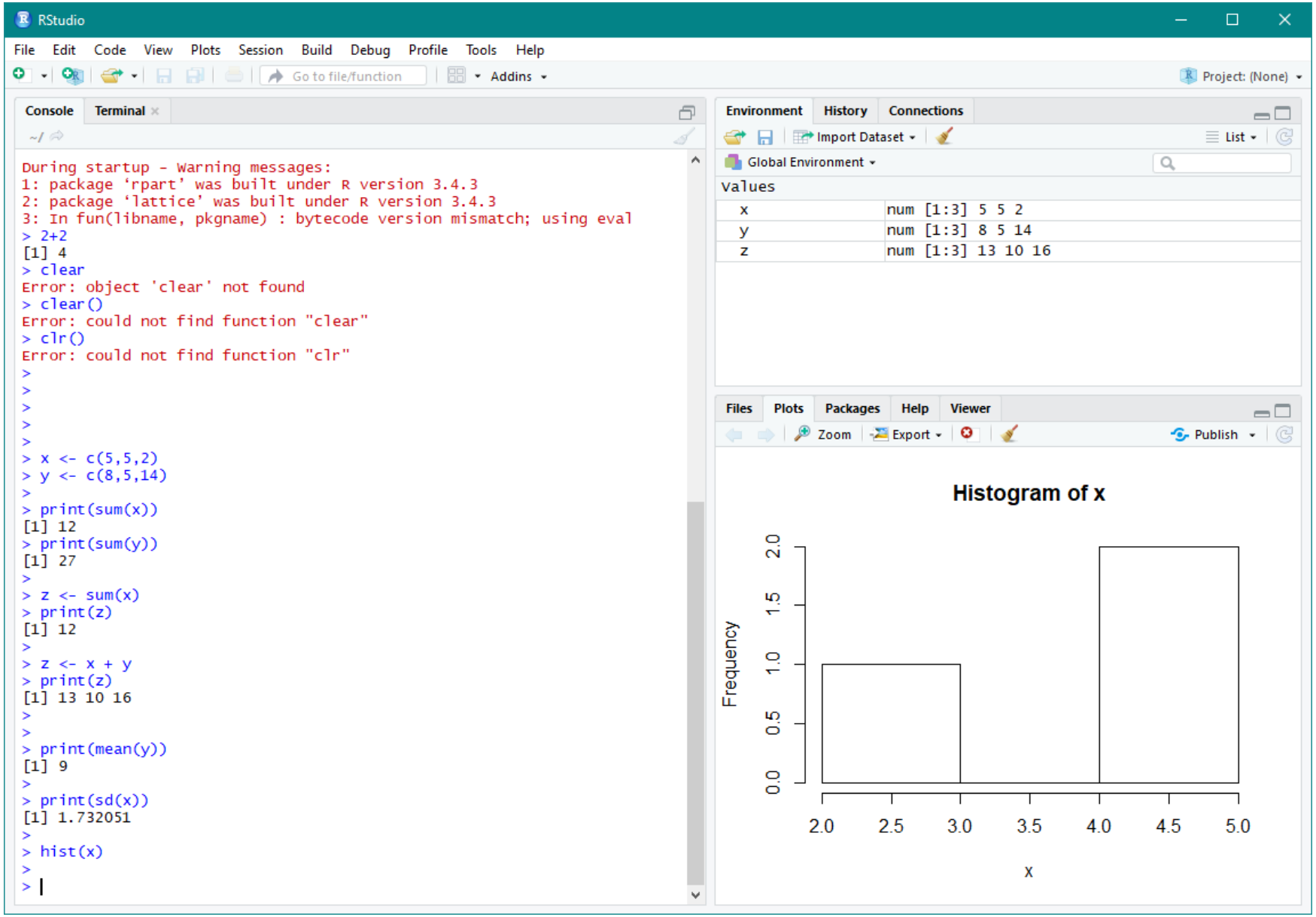
Description

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of class `"histogram"` is plotted by `plot.histogram`, before it is returned.

Usage

```
hist(x, ...)

## Default S3 method:
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

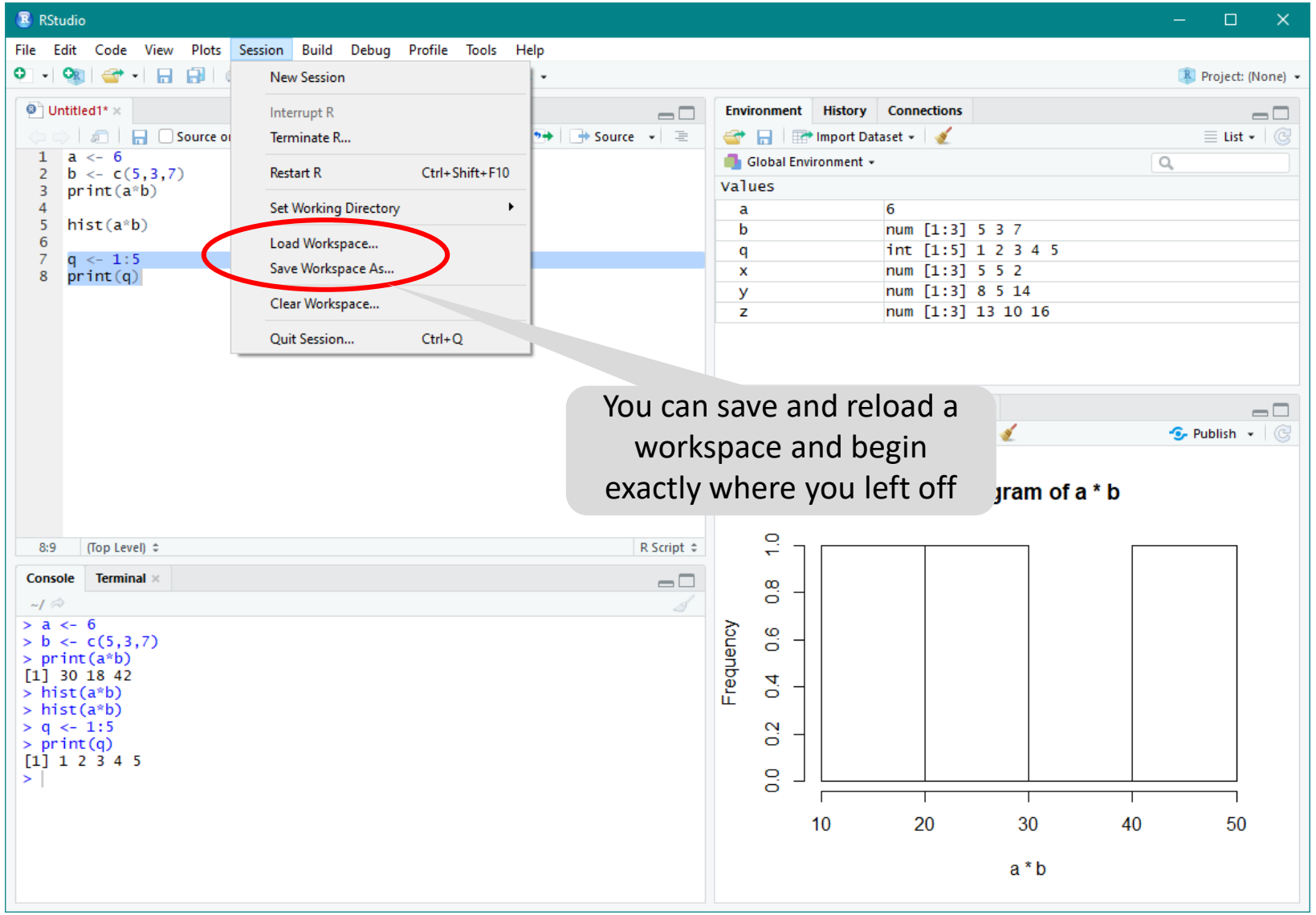
RStudio



You can download it for free from here:

https://www.rstudio.com/products/rstudio/download/

The Workspace/Image

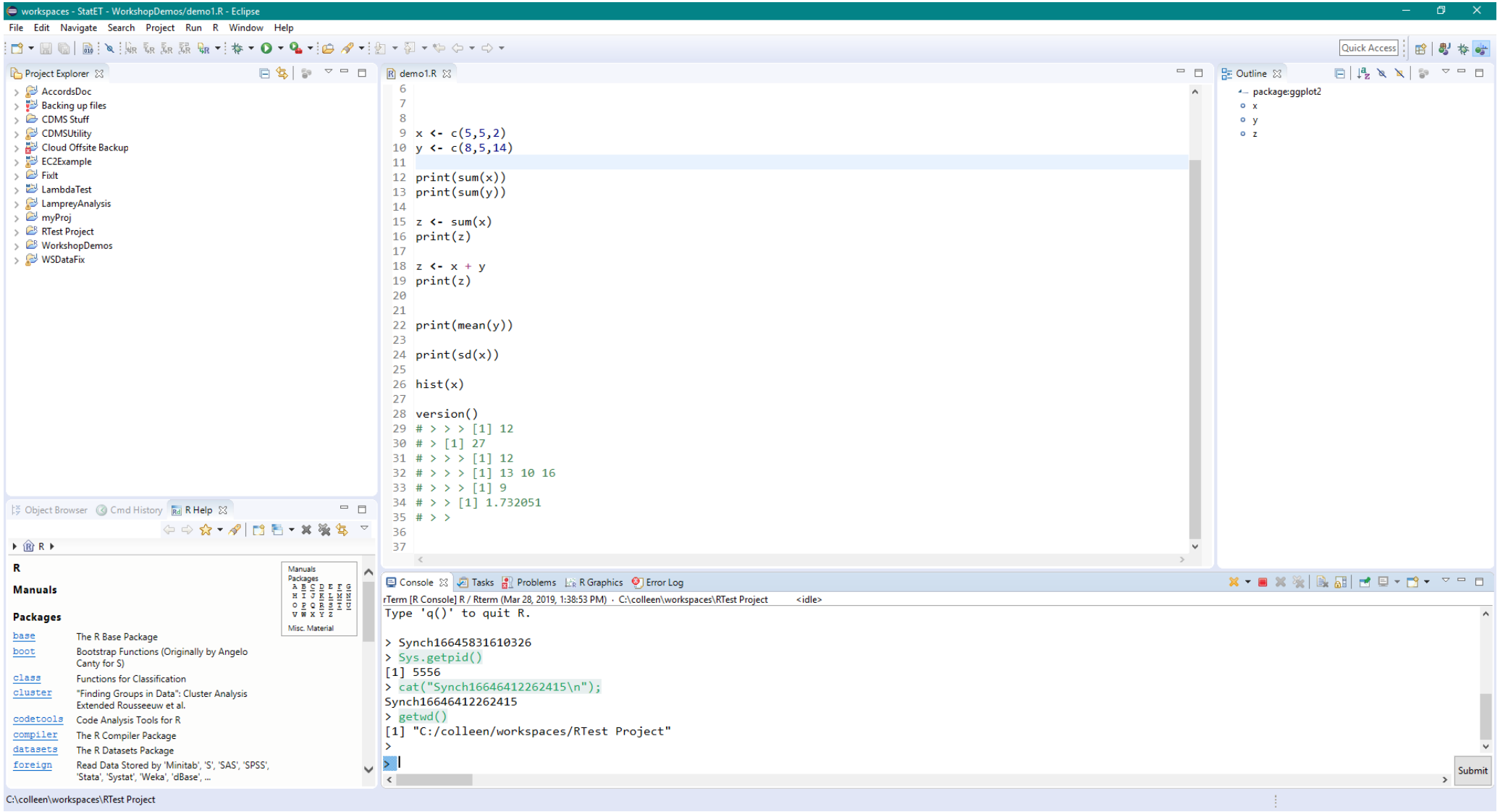


You can save and reload a workspace and begin exactly where you left off

An R Workspace contains <u>all</u> the state information present in R at the time

- All the objects that have been created
- All the functions that have been created
- The workspace can be saved to a file as an “image”
- And later reloaded so you return to you last known state

Eclipse/StatET



Download from here:

http://walware.de

(and click on the link in the upper left corner that will take you to the English pages)

Warning: StatET is not a simple installation. Knowledge of Java, Eclipse, and Unix/Linux needed

Walking tour of

The basics of programming in R

Simple Coding Comparison: R vs Java

R

```
x <- 1:20
sumX <- sum(x)
sumX2 <- sum(x^2)
sd <- sqrt((sumX2 - (sumX^2/20))/(19))
print(sd)
```

Java

```
public class sd {

        public static void main(String[] args) {

        double x[] =      R shortcuts specifying the initial sequence
        {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

        double sumX = 0.0;
        double sumX2 = 0.0;

        for(double d: x) {                          No need to write loop in R
                sumX = sumX + d;
                sumX2 = sumX2 + d*d;
        }

        double sd = Math.sqrt((sumX2 -(sumX*sumX/20))/19);
        System.out.println("sd = " + Double.toString(sd));

        }
}
```

Data Structures

In addition to the usual built-in data scalars (e.g., int, long, float, double, boolean, char, string), R also provides native data structures:

- Vectors (contain scalars of a single type)
- Matrices (only 2 dimensions, a single type of data)
- Arrays (multiple dimensions, single type of data)
- Data Frames (similar to matrix but columns can contain different types of data)
- List (ordered collection of objects of any type)
- and Factors – for example, the Make of a car could be represented by a factor such as (Ford, GM, Toyota, BMW, Mercedes)

Creating Data Structures

Create a vector using the construct function: c()

v <- c(1,2,3) creates a vector containing 1, 2, and 3 in that order
v[2] gives you the second element in the vector : 2

Create a matrix by combining rows or columns: rbind() or cbind()

```
r1 <- c(1,2)                    OR      c1 <- c(1,3)
r2 <- c(3,4)                            c2 <- c(2,4)
m <- rbind(r1, r2)                      m <- cbind(c1, c2)
```

Either approach creates the matrix:

```
|1 2|
|3 4|
```

You can access elements in the matrix by indices: m[1,2] yields the value 3

Creating Data Strucures

If the value of m2 is:|1 2 3| then m2[2:3,2:3] gives you: |5 6|
|4 5 6| |8 9|
|7 8 9|

And m2[-2,-2] gives you: | 1 3|
| 7 9|

It is also possible to give rows and columns names rather than numbers

Creating Data Structures II

Data frames are commonly created by reading CSV files using the `read.table` function, for example:

```
fish <- read.table("c:/colleen/workshop/fishData.csv",
                   sep=",",header=T)
```

```
  SampleDate Species Length Age     PitTagCode
1  9/11/2012    FACH   50.0 0.1 3D9.1C2DD8DE20
2  9/11/2012    FACH   51.5 0.1 3D9.1C2DF1DA39
```

NOTE: the columns can take the names on the header columns in the file, optionally

`write.table()` can write a data frame as a CSV file

Accessing Data in Data Frames

- An element can be accessed like an element in a matrix: `fish[3,4]`
- An entire column can be accessed by name: `fish$Age or fish[,4]`
- Or, a row `fish[2,]`
- So, it becomes easy to get statics on data in data frame:
  - `sd(fish$Age)`
  - `mean(fish$Age)`
  - `hist(fish$Age)`
- Or, in one fell swoop
  - `summary(fish)`

Note how R determines the data type in a column and reports appropriately

```
    SampleDate   Species        Length          Age                PitTagCode
5/9/2012 :  76   FACH:2910   Min.   :50.00   Min.   :0.1000   384.3B23AF4811:   2
9/10/2013:  75   SPCH:2056   1st Qu.:64.00   1st Qu.:0.3000   384.1B79764260:   1
9/3/2014 :  75   SUCH:1928   Median :71.50   Median :1.1000   384.1B79764278:   1
9/23/2013:  74               Mean   :70.57   Mean   :0.7503   384.1B79764284:   1
9/19/2012:  73               3rd Qu.:78.00   3rd Qu.:1.2000   384.1B79764299:   1
5/8/2012 :  71               Max.   :90.00   Max.   :2.3000   384.1B797642AD:   1
(Other)  :6450                                                (Other)       :6887
>
```

Accessing Data in Data Frames II

- Create a new data frame by taking columns 1,3, and 5 from rows 10 thru 21
  - `x <- fish[c(10:21),c(1,3,5)]`
- Create a new data frame where the Species is FACH and we only want columns Age and Length
  - `y <- subset(fish, Species = “FACH”, select(“Age”, “Length”))`
- Create a new data frame where the length is a minimum of 50
  - `z <- subset(fish, Age >= 50)`
- Now, let’s remove data from existing data frame: we remove rows 4,5, and 6 and all of column 5 (PITTagID)
  - `w <- fish[-c(4,5,6), -c(5)]`
- All this with absolutely no looping in the code
- These are just a samples, there are many other techniques and functions to prune data frames

Data Access Features: An Analogy

Java and C#

VS

R

What Else Do Native Data Structures Buy You?

- Built-in functionality for sorting, merging, aggregating, reshaping, and subsetting data
- Conventional languages like Java and C# do not provide all this functionality
- Example: built-in sorting of data

Sorting Data

To sort a data frame in R, use the **order( )** function. By default, sorting is ASCENDING. Prepend the sorting variable by a minus sign to indicate DESCENDING order. Here are some examples.

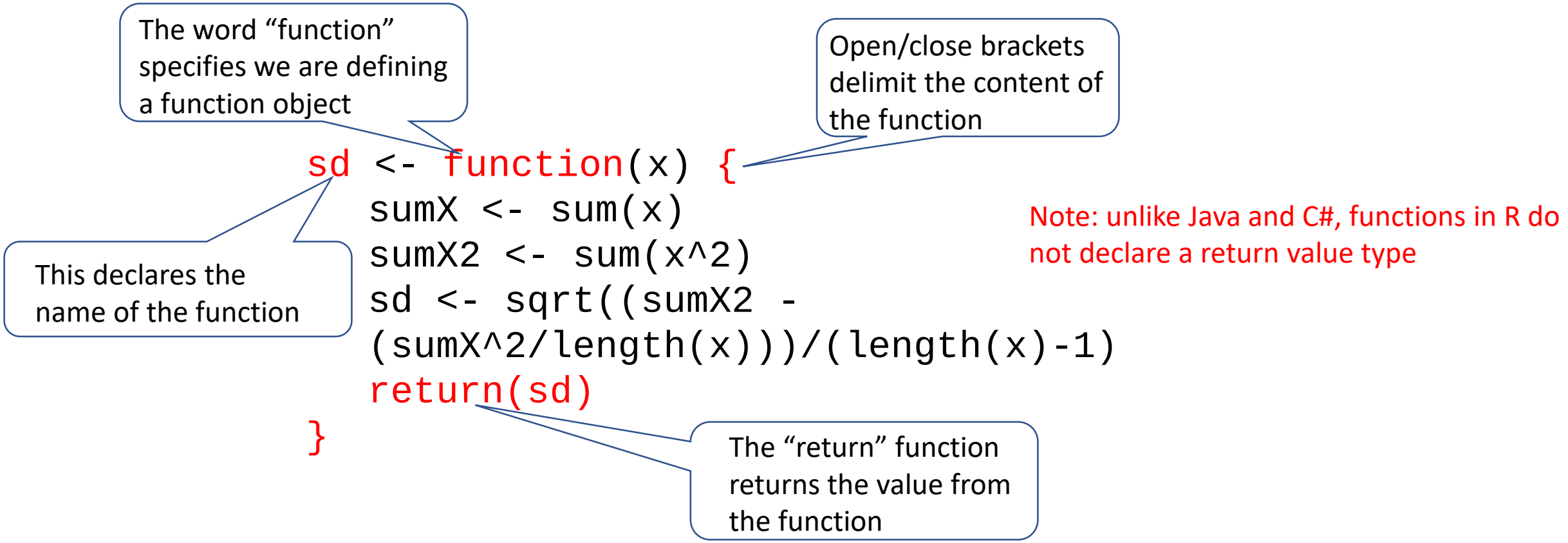
`mtcars` is a built-in sample data frame with data on cars

```
# sorting examples using the mtcars dataset
attach(mtcars)

# sort by mpg
newdata <- mtcars[order(mpg),]

# sort by mpg and cyl
newdata <- mtcars[order(mpg, cyl),]

#sort by mpg (ascending) and cyl (descending)
newdata <- mtcars[order(mpg, -cyl),]

detach(mtcars)
```

R is Functional Programming

To script reusable code , we package the code in a particular type of object called a “function”

Here is a function definition:



R Operators

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Conditional Logic

Like Java and C#, R has an if-else statement for example:

```
// discount list price by 5% only if buyer
// is a member
FinalPrice <- if(member) ListPrice * 0.95
              else ListPrice
```

But, since R deals with vectors. It also has a vectorized if-else statement:

```
ifelse(v1, v2, v3)
```

where:

- v1 is a vector of logical values
- v2 is a vector of values to be returned if the corresponding value in v1 is TRUE
- v3 is a vector of values to be returned if the corresponding value in v1 is FALSE

`Ifelse` returns a vector of the values calculated for each position

`ifelse` Statement

Consider this `ifelse` statement:

```
ifelse(c(3,6,12) < 5, 4:6, 7:9)
```

The arguments to it are:

```
v1 = TRUE, FALSE, FALSE
v2 = 4, 5, 6
v3 = 7, 8, 9
```

The vector returned from the statement is:

```
4, 8, 9
```

Note: again, we used no loops

R Control Structures

if-else

```
if (cond) expr
if (cond) expr1 else expr2
```

for

```
for (var in seq) expr
```

while

```
while (cond) expr
```

switch

```
switch(expr, …)
```

ifelse

```
ifelse(test, yes, no)
```

“Looping” the R Way

1. We build a 5x6 matrix

```
> x <- matrix(rnorm(30), nrow=5, ncol=6)
> x
           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
[1,]  0.2386702  1.3499769  1.6661533 -0.4196998 -0.6559358 -0.2736569
[2,] -1.8832814  0.1200928 -0.9730539  0.7075804 -0.1711574 -0.5475355
[3,] -0.7571133  2.1929553  1.2340384 -0.3150873  0.3112674 -0.1292675
[4,] -0.9543883 -0.8519280 -1.5907856 -1.2084175 -0.4594546  0.7904125
[5,]  0.5425458 -0.5738047 -1.1163553  1.6020351  2.0274718  2.7624321
```

2. We want the average of each column, so could loop:

```
avgs <- c()
for( I in 1:5) {
        colsum = 0
        for(j in 1:6) {
                colsum = colsum + x[I,j]
        }
        avgcol = colsum/5
        avgs <- c(avgs, avgcol)
}
summary(avgs)
```

Or we could…

…use the `apply` function: `apply(values,mode,func)`

The function to be applied to data

1 = row-wise
2 = column-wise

```
means <- apply(x,2,mean)
summary(means)
```

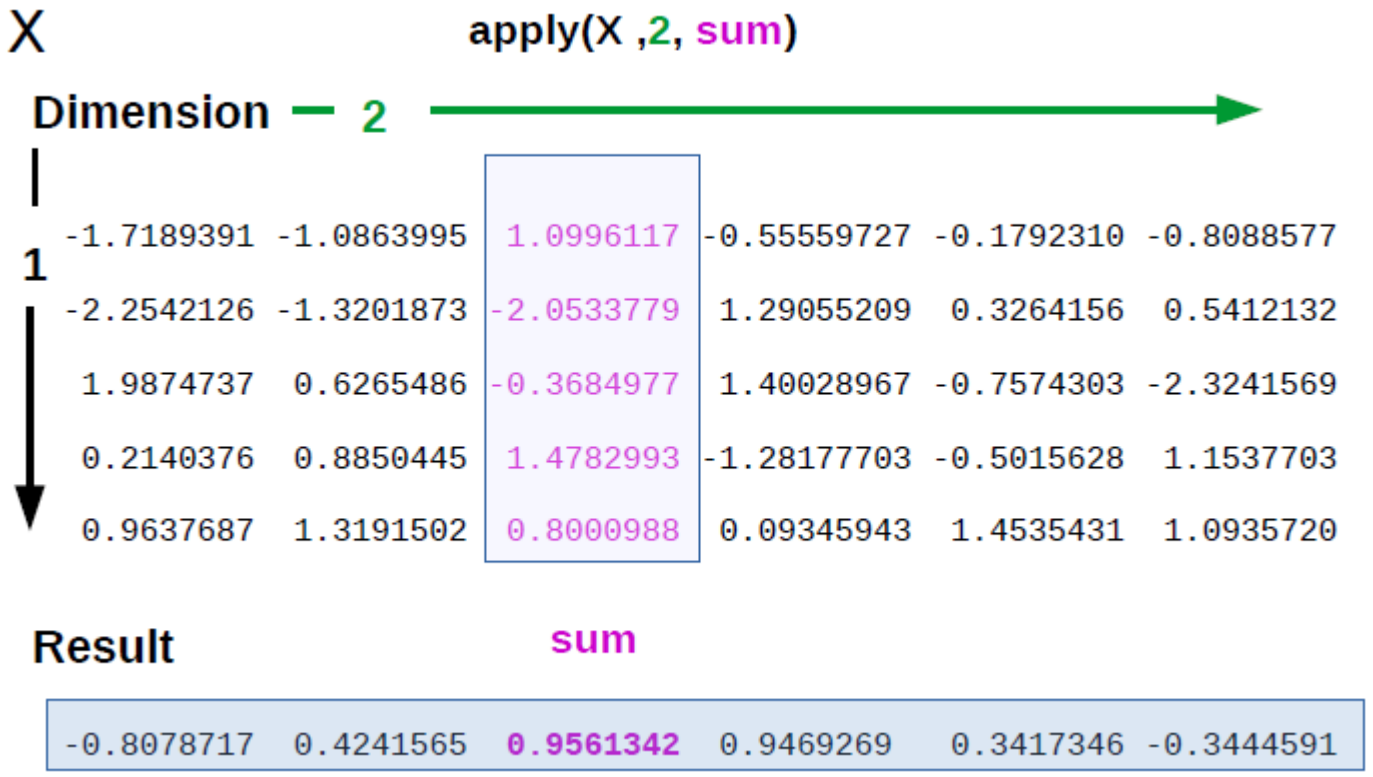
```
[1] -2.8135670  2.2372923 -0.7800031  0.3664110  1.0521914  2.6023847
```

Advantages of the apply function family:

- No need for writing nested loops, only *one* line of code
- The `apply` functions are generally faster than loops since they run in compiled code not interpreted code

Visualization of `apply` Function

Visually, this is what is happening:



The `apply` Family of Functions

R provides a mechanism more efficient than looping: the apply functions

Function Name	*Objects the Function Works On*	*What the Function Sees as Compo-nents*	*Result Type*
apply	Matrix	Rows or columns	Vector, matrix, array, or list
	Array	Rows, columns, or any dimension	Vector, matrix, array, or list
	Data frame	Rows or columns	Vector, matrix, array, or list
sapply	Vector	Elements	Vector, matrix, or list
	Data frame	Variables	Vector, matrix, or list
	List	Components	Vector, matrix, or list
lapply	Vector	Elements	List
	Data frame	Variables	List
	List	Components	List

There are other “apply” functions, but these three are the most commonly used

Basic Statistics I

Central Tendency and Variability

Function	What it Calculates
mean(x)	Mean of the numbers in vector x.
median(x)	Median of the numbers in vector x
var(x)	Estimated variance of the population from which the numbers in vector x are sampled
sd(x)	Estimated standard deviation of the population from which the numbers in vector x are sampled
scale(x)	Standard scores (z-scores) for the numbers in vector x

Basic Statistics II

Relative Standing

Function	What it Calculates
sort(x)	The numbers in vector x in increasing order
sort(x)[n]	The nth smallest number in vector x
rank(x)	Ranks of the numbers (in increasing order) in vector x
rank(-x)	Ranks of the numbers (in decreasing order) in vector x
rank(x, ties.method= "average")	Ranks of the numbers (in increasing order) in vector x, with tied numbers given the average of the ranks that the ties would have attained
rank(x, ties.method= "min")	Ranks of the numbers (in increasing order) in vector x, with tied numbers given the minimum of the ranks that the ties would have attained
rank(x, ties.method = "max")	Ranks of the numbers (in increasing order) in vector x, with tied numbers given the maximum of the ranks that the ties would have attained
quantile(x)	The 0^{th}, 25^{th}, 50^{th}, 75^{th}, and 100^{th} percentiles (i.e, the *quartiles*) of the numbers in vector x. (That's not a misprint: quantile(x) returns the quartiles of x.)

Basic Statistics III

t-tests

Function	What it Calculates
t.test(x,mu=n, alternative = "two.sided")	Two-tailed t-test that the mean of the numbers in vector x is different from n.
t.test(x,mu=n, alternative = "greater")	One-tailed t-test that the mean of the numbers in vector x is greater than n.
t.test(x,mu=n, alternative = "less")	One-tailed t-test that the mean of the numbers in vector x is less than n.
t.test(x,y,mu=0, var.equal = TRUE, alternative = "two.sided")	Two-tailed t-test that the mean of the numbers in vector x is different from the mean of the numbers in vector y. The variances in the two vectors are assumed to be equal.
t.test(x,y,mu=0, alternative = "two.sided", paired = TRUE)	Two-tailed t-test that the mean of the numbers in vector x is different from the mean of the numbers in vector y. The vectors represent matched samples.

And also basic ANOVA and correlation analysis functions

Advanced Statistics

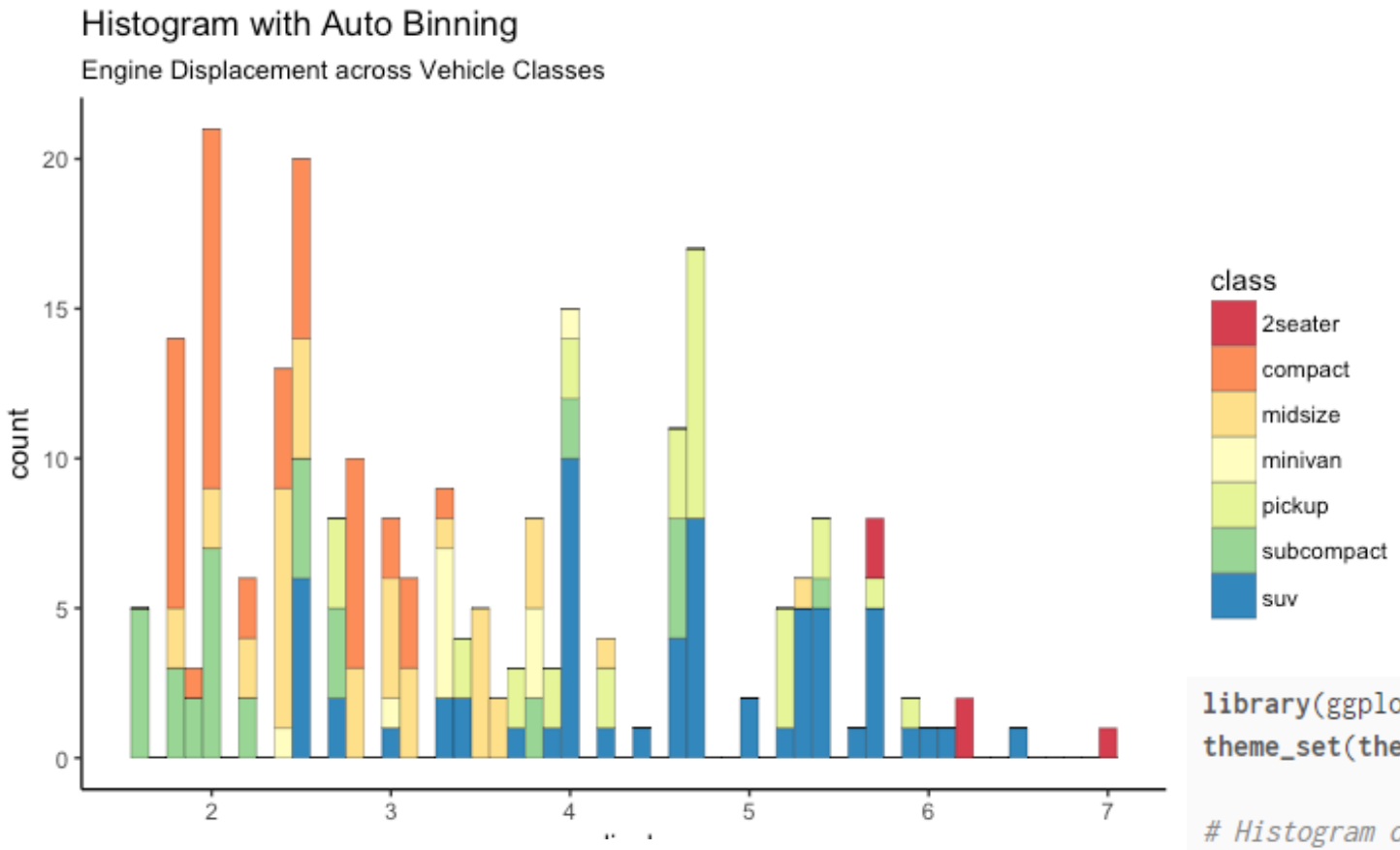
STATISTICS

- Descriptive Statistics
- Frequencies & Crosstabs
- Correlations
- t-tests
- Nonparametric Statistics
- Multiple Regression
- Regression Diagnostics
- ANOVA/MANOVA

ADVANCED STATISTICS

- Generalized Linear Models
- Discriminant Function
- Time Series
- Factor Analysis
- Correspondence Analysis
- Multidimensional Scaling
- Cluster Analysis
- Tree-Based Models
- Bootstrapping
- Matrix Algebra

Graphic using ggplot2




```
library(ggplot2)
theme_set(theme_classic())

# Histogram on a Continuous (Numeric) Variable
g <- ggplot(mpg, aes(displ)) + scale_fill_brewer(palette = "Spectral")

g + geom_histogram(aes(fill=class),
                   binwidth = .1,
                   col="black",
                   size=.1) +  # change binwidth
  labs(title="Histogram with Auto Binning",
       subtitle="Engine Displacement across Vehicle Classes")
```

Handling simple file IO in R

Importing Data from Excel Spreadsheet

1. Export you Excel data into a CSV file with variable names in first row
2. Import your data from the file with this code
3. It is also possible to read it in directly from a .xlxs file

```
# note the / instead of the \ in file name even though we are on Windows

fishData <- read.table("c:/colleen/workshop/fishData.csv", header=T,sep=",")
```

Writing Data to Output

```
# write a tab delimited text output file
write.table(mydata, "c:/mydata.txt", sep="\t")



# write to an Excel file format

# we have to explicitly load the xlsx library into the workspace
library(xlsx)

write.xlsx(mydata, "c:/mydata.xlsx")
```

Actually Programming in R

Putting It All Together

- We are going to read a CSV file exported from Excel containing fish data ~7000 records
- The fields are:
  - `SampleDate`
  - `Species`
  - `Length`
  - `Age`
  - `PITTagCode`
- We will put the data into a data frame
- We will calculate basic statistics
- We will create a series of graphs
- We will explore other graphical possibilities using ggplot2

ggplot2 Resource

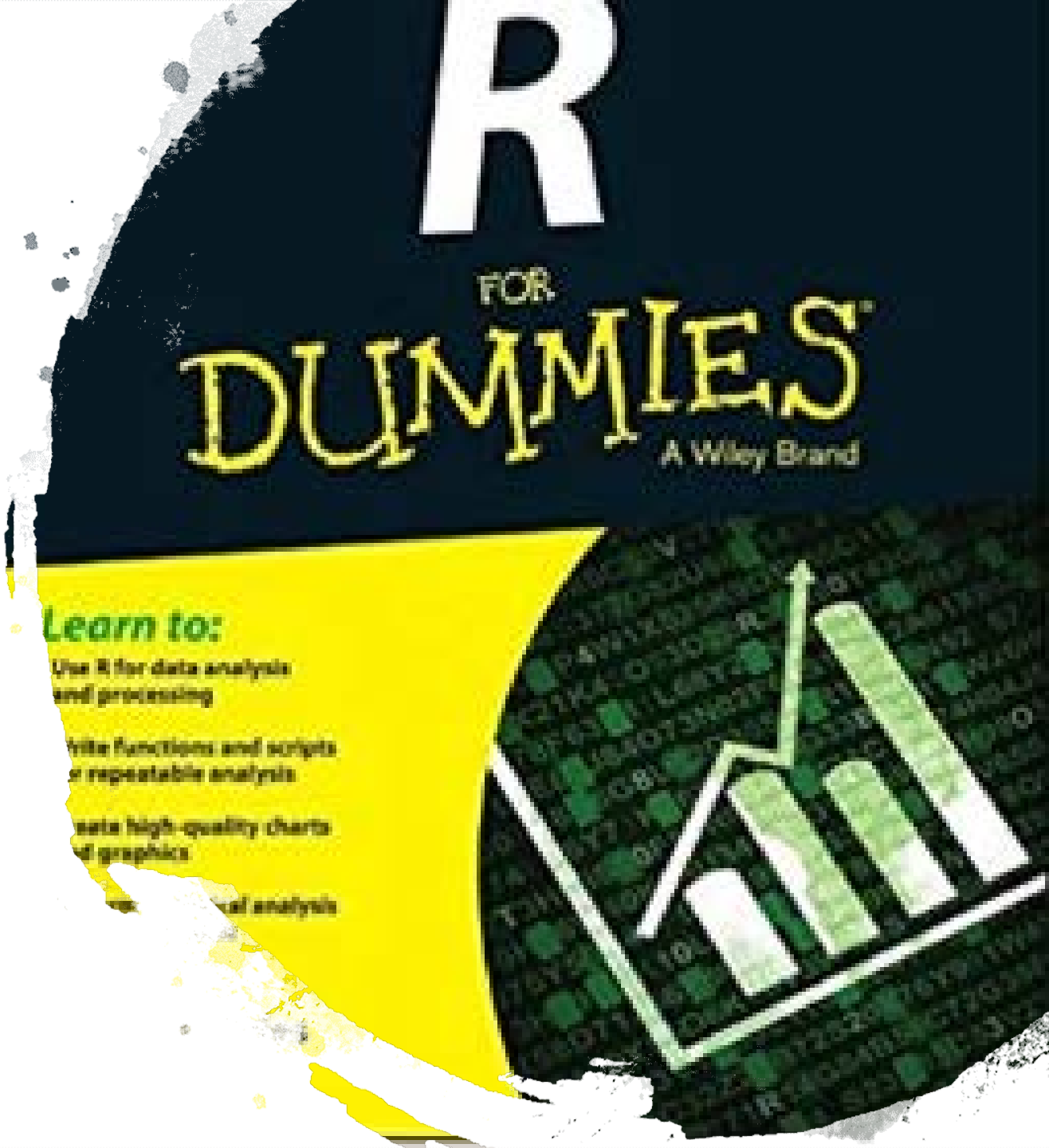
This website is a terrific resource for someone trying to learn ggplot2:

http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html#Calendar%20Heat%20Map

Wrap up and questions

Suggested Reading

- Available at Amazon new, used, and in Kindle books
- Written by two people who have worked on R for many years
- Best general reference for R I've found to date:
  - Good general advice for programming conventions
  - Lots of examples
  - Clear explanations
- And, get the second edition, it's a big improvement over the first



Suggested Free Online Tutorial

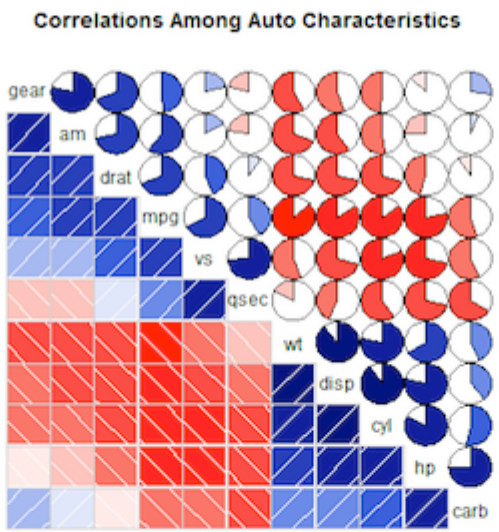
Quick-R by DataCamp

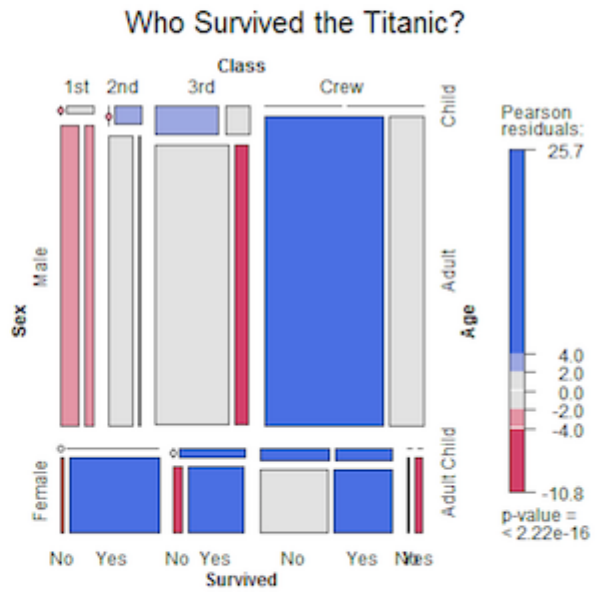
R Tutorial | R Interface | Data Input | Data Management | Statistics | Advanced Statistics | Graphs | Advanced Graphs

SITE CONTENTS

- Learning R
- R Tutorial
- R Interface
- Data Input
- Data Management
- Statistics
- Advanced Statistics
- Graphs
- Advanced Graphs

About Quick-R





R is an elegant and comprehensive statistical and graphical programming language. Unfortunately, it can also have a steep learning curve. I created this website for both

https://www.statmethods.net/index.html

Any Questions?