R: From nil to 50 in 50 min

Dr. Kim Cuddington
Dept. Biology
University of Waterloo
19.03.14

The pros of using R!

- It is 100% FREE
- **Huge** community of users, which means lots of help and support.
- Available on virtually all platforms: OS X, Unix/ Linux, and Windows XP, Vista, and 7!
- Documentation is available in many languages
- Publication quality graphics with default options

The pros of using R! cont.

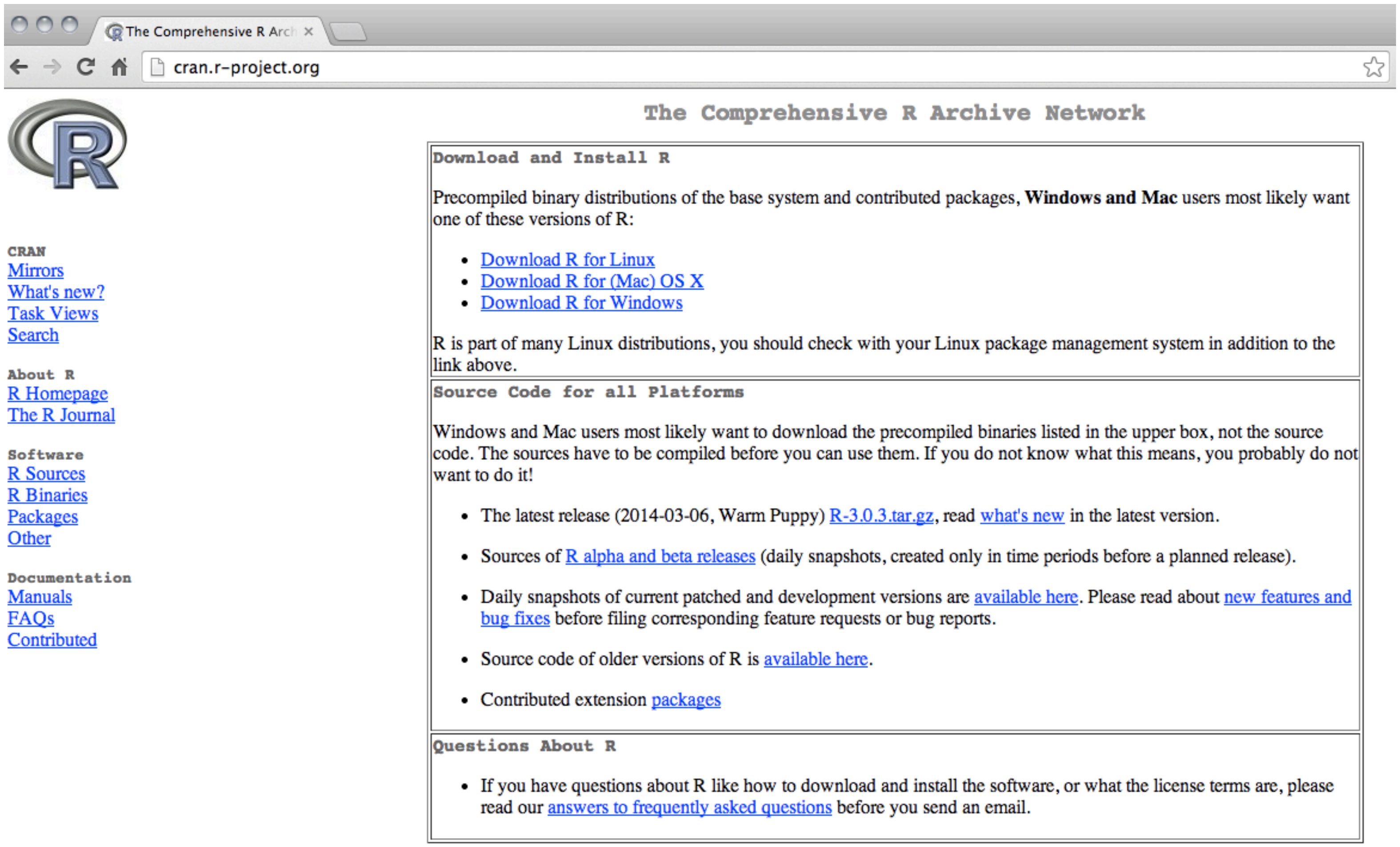
- R library database has virtually every type of statistical analysis that has ever been conceived. The most cutting edge biostatistical ideas are always implemented in R first.
- R is very flexible. All models and graphics can be completely customized and modified for the analysis you're working on.

The cons of R …

- Data manipulation is poor for editing
- Steep learning curve
- The programming is largely done by statisticians not computer scientists … so can be slow during computationally intensive tasks

Get R: Its free!

- Use web browser to access
  - http://cran.us.r-project.org/
- Download "base" package



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

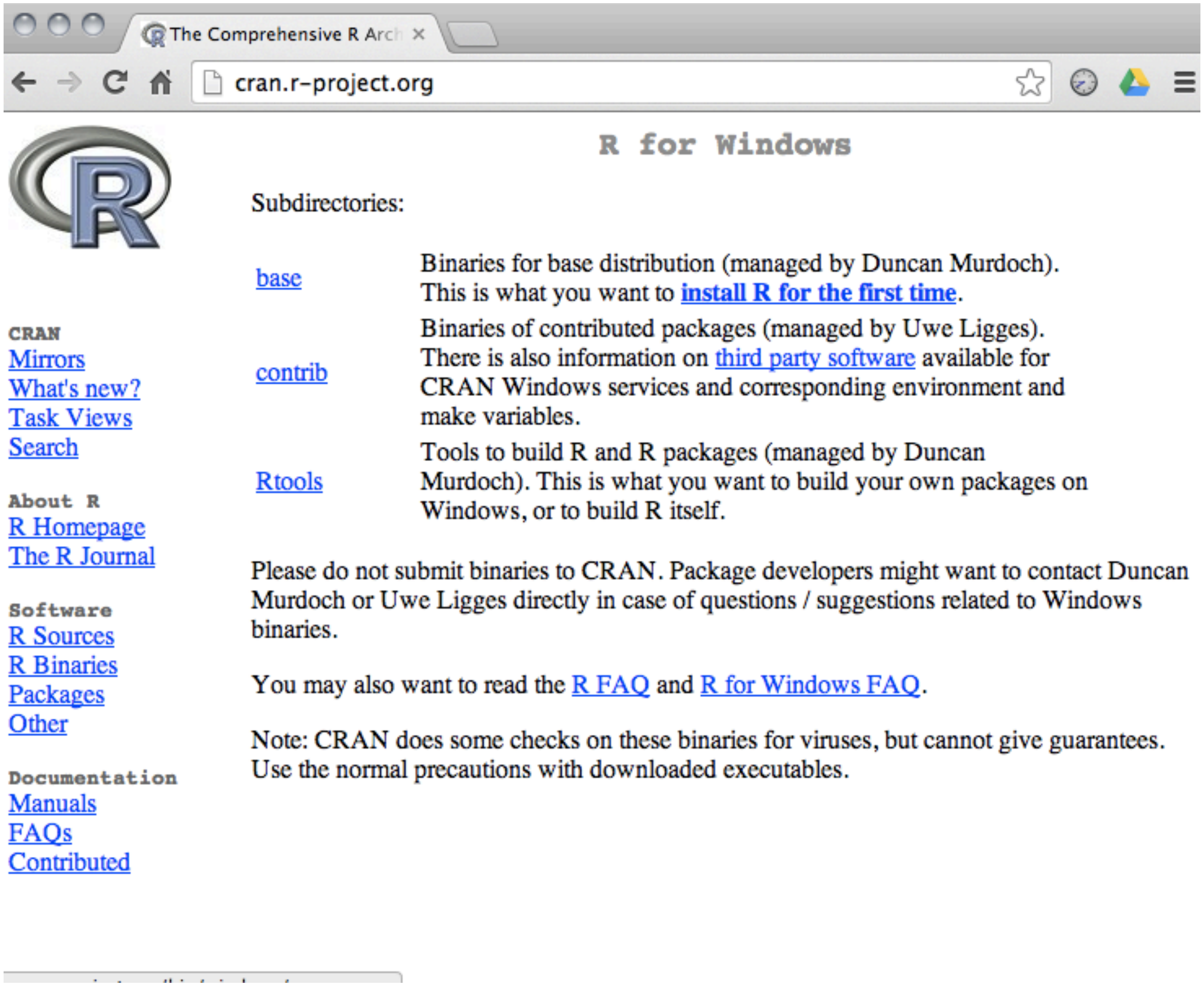
Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2014-03-06, Warm Puppy) R-3.0.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.



R for Windows

Subdirectories:

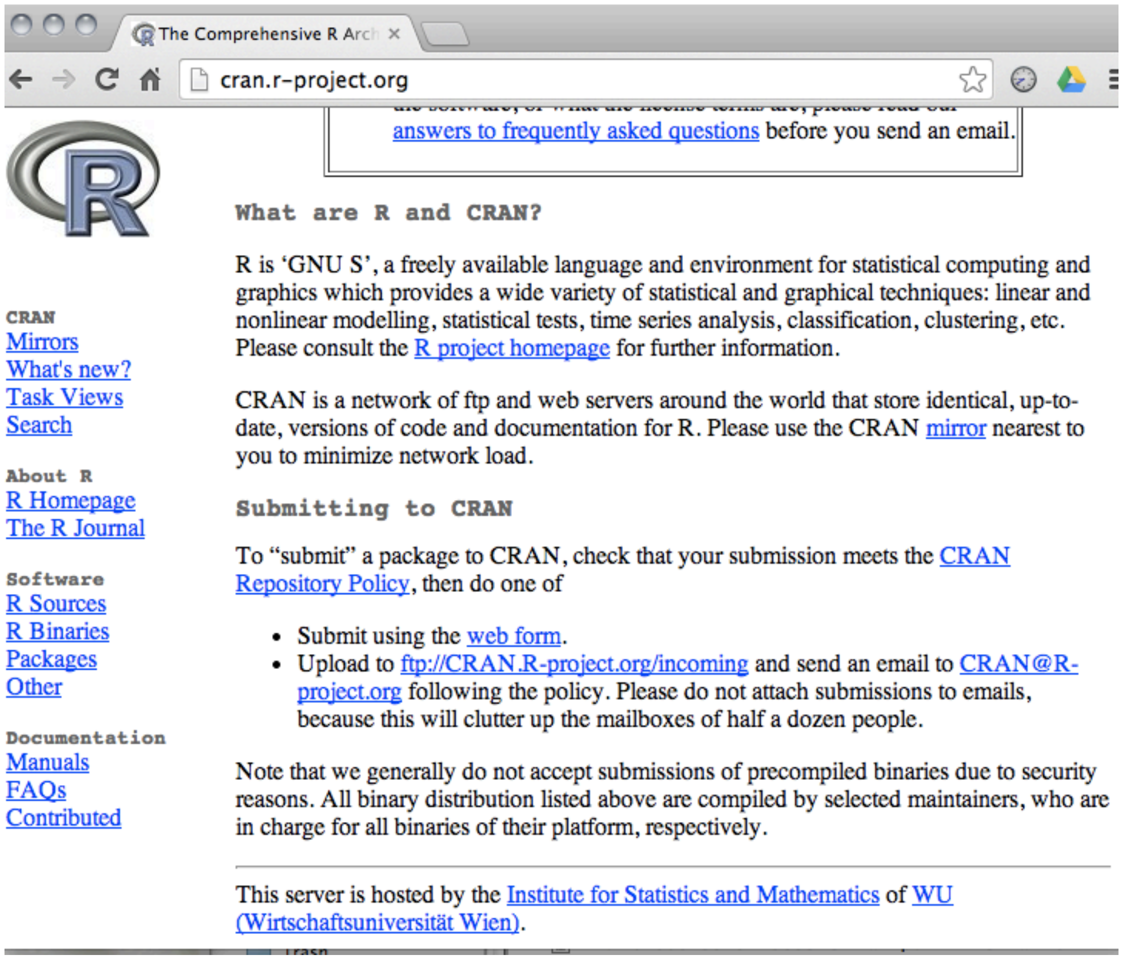
base	Binaries for base distribution (managed by Duncan Murdoch). This is what you want to **install R for the first time**.
contrib	Binaries of contributed packages (managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
Rtools	Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch or Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

What is R??



answers to frequently asked questions before you send an email.

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

What are R and CRAN?

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

Submitting to CRAN

To "submit" a package to CRAN, check that your submission meets the CRAN Repository Policy, then do one of

- Submit using the web form.
- Upload to ftp://CRAN.R-project.org/incoming and send an email to CRAN@R-project.org following the policy. Please do not attach submissions to emails, because this will clutter up the mailboxes of half a dozen people.

Note that we generally do not accept submissions of precompiled binaries due to security reasons. All binary distribution listed above are compiled by selected maintainers, who are in charge for all binaries of their platform, respectively.

This server is hosted by the Institute for Statistics and Mathematics of WU (Wirtschaftsuniversität Wien).

R History

R is a comprehensive statistical and graphical programming language and is a dialect of the S language:

1988 - S2: RA Becker, JM Chambers, A Wilks

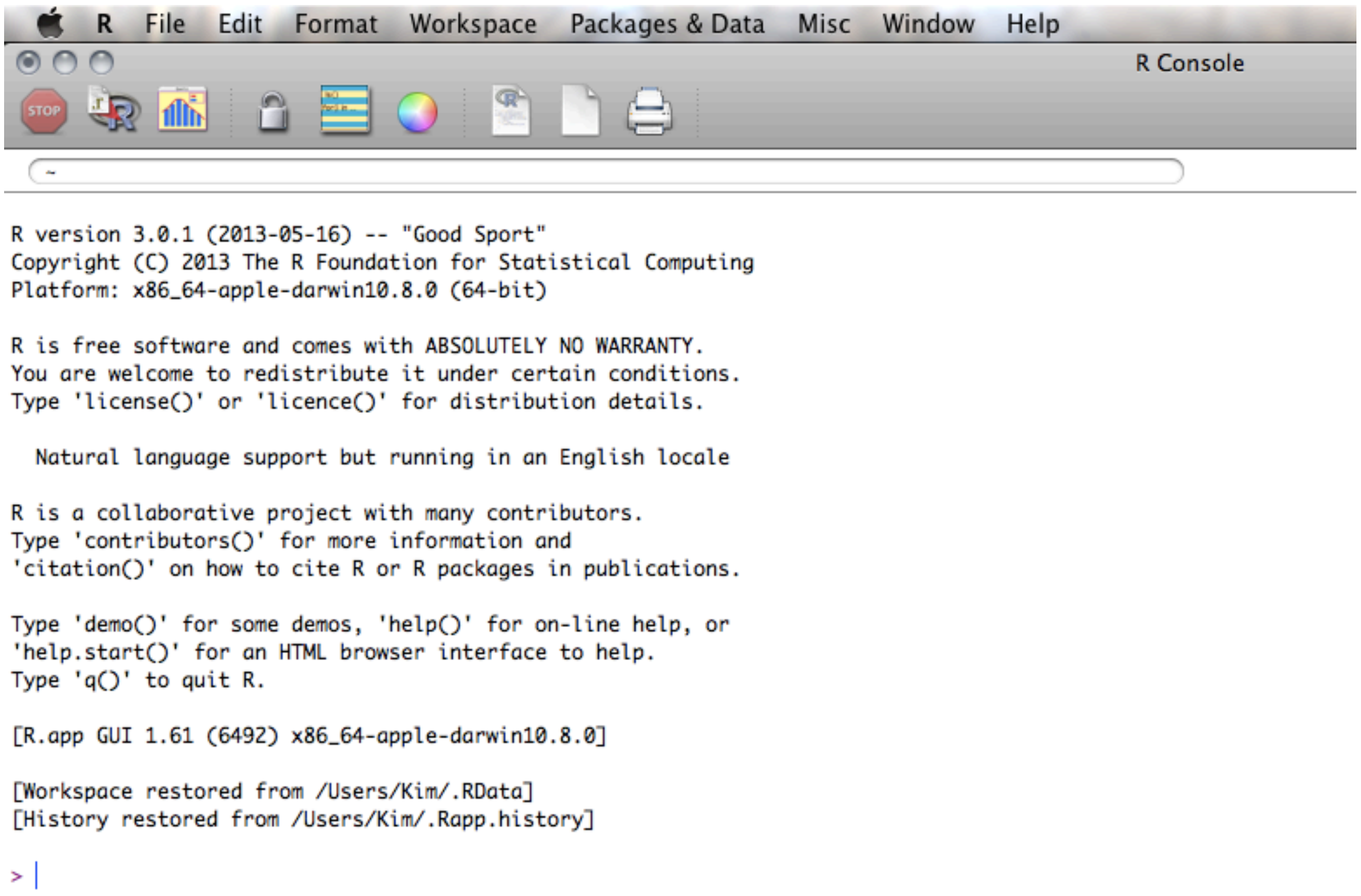
1992 - S3: JM Chambers, TJ Hastie

1998 - S4: JM Chambers

R: initially written by Ross Ihaka and Robert Gentleman at Dep. of Statistics of U of Auckland, New Zealand during 1990s.

Since 1997: international "R-core" team of 15 people with access to common CVS archive.

Open R!



```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (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.

  Natural language support but running in an English locale

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.

[R.app GUI 1.61 (6492) x86_64-apple-darwin10.8.0]

[Workspace restored from /Users/Kim/.RData]
[History restored from /Users/Kim/.Rapp.history]

>
```

R overview

- You can enter commands one at a time at the command prompt (>) or run a set of commands from a source file.
- To quit R, use

```
>q()
```

R syntax

- This table shows the most common operators.

`+, -, *, /, ^`	Arithmetic
`!=, &, |, ==, >, >=, <, <=`	Logical
`:`	Sequence

Basic command line operations

- Simple arithmetic:

```
> 5+3
[1] 8
> ((2 + 4)^2)/3
[1] 12
```

- More advanced functions

```
> sqrt(2)
[1] 1.414214

> log(2.74)
[1] 1.007958
```

- Creating list of consecutive numbers (1-15)

```
1:15
```

Find the sine of π

Hint: use “sin” and “pi”

builtins() lists all the functions in R

Data in

- R allows users to **create variables**, which are essentially named computer memory.
- For example, you may store the number of species in a sample in a variable.
- Variables are identified by a name assigned when they are created.

Variable assignment

- To assign a value to a variable enter the variable name to the left of a left-pointing arrow
- Typed with the "less than" followed by a "dash” with the value behind the arrow. For example: speciesA <- 137
- Recent versions of R allow you to use the = sign for an assignment, i.e. speciesB = 137

Notes about variable assignment

- No default output for assignment

```
> coffee.mon<-3
> coffee.tues=4
>
> coffee.mon
[1] 3
>
```

- Variable names are case-sensitive

```
> coffee.Mon
Error: object 'coffee.Mon' not found
>
```

More than just integers!

- R also allows the creation of variables that contain floating point or real numbers, characters, or special characters interpreted as "logical" values. For example

```
x <- 1.2345
small.value <- 1.0e-10
species.name <- 'Pinus contorta'
conifer <- TRUE
```

Sets of variables: Data structures are sets of variables organized in a particular way

1. Vectors
2. Matrices
3. Dataframes (most like an excel spreadsheet)
4. Lists

Vectors

Vectors are one-dimensional ordered sets composed of a single data type. Data types include integers, real numbers, and strings (character variables)

Vectors are often read in as data or produced as the result of analysis, but you can produce one simply using the **c()** function, which stands for "combine." For example

```
> coffeeweek<-c(3,2,5,8,6,4,1)
> week<-c("m","t","w","th","f","s","sn")
>
> week
[1] "m"  "t"  "w"  "th" "f"  "s"  "sn"
> coffeeweek
[1] 3 2 5 8 6 4 1
```

Create a vector of the number of coffees you had on each day of last week, and a vector of the names of the week

Create a graph of your coffee consumption as:

barplot(coffeeweek, names=week)

Matrices

- **Matrices** are multidimensional ordered sets composed of a single data type, equivalent to the concept of matrix in linear algebra.
- A matrix can be constructed by using both the matrix() function and the c() function

```
> beverage = matrix(c( coffeeweek, 2, 5, 9, 2, 7, 9, 2), nrow=7, ncol=2)
> beverage
     [,1] [,2]
[1,]    3    2
[2,]    2    5
[3,]    5    9
[4,]    8    2
[5,]    6    7
[6,]    4    9
[7,]    1    2
```

Create a matrix of the number of coffees you had on each day of last week, and the number of servings of your fave beverage

Hint: you will need to use the matrix and c functions

Referencing individual items in a vector or matrix

- Individual items can be identified by subscript (numbered 1 - n), which is indicated by a number (or numeric variable) within square brackets.
- For example, if the number of plant species per plot is stored in a vector **veg**, then **veg[37]** equals the number of species at position 37.
- Matrices are specified in the order "row, column", so that **veg[23,48]** equals row 23, column 48 in matrix veg.

Referencing whole rows or columns in a matrix

- Individual rows or columns within a matrix can be referred to by implied subscript, where the value of the desired row or column is specified, but other values are omitted.
- For example, **veg[,3]** represents the third column of matrix veg, as the row number before the comma was omitted. Similarly, **veg [5,]** represents row 5, as the column after the comma was omitted.

So we can reference each column in our beverage matrix independently as

```
plot(beverage[,1], beverage[,2])
```

Referencing groups of items in a vector or matrix

- In addition, a number of specialized subscripts can be used:

veg[1:10]	= veg[1] through veg[10]
veg[-3]	= all of vector veg except veg[3]
veg[a:b,c:d]	= a submatrix of veg from row a to b and column c to d

- It's even possible to specify specific subsets of rows and columns that are not adjacent, veg[c(1,7,10),c (3,6,12)] = a submatrix consisting # of rows 1,7 and 10, and columns # 3, 6, as 12 from matrix veg.

How would I produce a plot that excluded a questionable point?

Hint: involves a – sign

Surprise! We've been using functions

- Many things, if not most things, are accomplished in R by using such *function calls*.
- A function is a set of commands that R will execute every time you type the function name.
- The format is that of a function name followed by parentheses containing one or more arguments.
- For example, if you type **plot(richness, cover)** in the command window where height and richness are vectors of the same length, R interprets **plot** as a function (since it recognizes the name), and richness and cover as the particular arguments you want to use in this instance of a function call.

Functions to manipulate data

- A large number of **functions** exist **for manipulating vectors, and** by extension, **matrices**. For example, if veg is a vegetation matrix of 100 sample plots and 200 species (plots as rows and species as columns), we can perform the following:
- x <- max(veg[,3]) assigns the maximum value of column 3 among all rows to the variable x
- y <- sum(veg[,5]) assigns the sum of column 5 in all rows to y
- logveg <- log(veg+1) creates a new matrix called "logveg" with all values the log of the respective values in veg (+1 to avoid log(0) which is undefined)

Using conditions to manipulate data

- In addition, R supports logical subscripts, where the subscript is applied whenever the logical function is true. Logical operators include:

> for "greater than”	>= for "greater than or equal to”
< for "less than”	<= for "less than or equal to”
== for "equal to”	!= for "not equal to”
& for "and”	| for "or”

- For example q <- sum(veg[,8]>10) assigns q the number of plots where the value of column 8 is greater than 10 (i.e., veg[,8]>0 is evaluated as 1 (true) or 0 (false) for each item, so that the sum is of 0's and 1's and gives us the number of items that satisfy the condition).

Let’s convert our beverage count data to volume consumed

Coffee=250 ml

Tea=200 ml

Working with vectors

- To convert vectors, just multiply their assigned names by a constant. Lets save these results to new variables:

```
volcoffee<-beverage[,1]*250
voltea<-beverage[,2]*200
```

- We can now create a daily vector of fluid intake:

```
fluids<-volcoffee+voltea
```

- Or even calculate total fluids per week or mean fluids per day

```
weekfluids<-sum(fluids)
Mfluids<-mean(fluids)
```

- When you are operating with vectors like this, the vectors must be of equal length!

Statistical questions

- Is my coffee and tea consumption statistically different?

Getting help from R

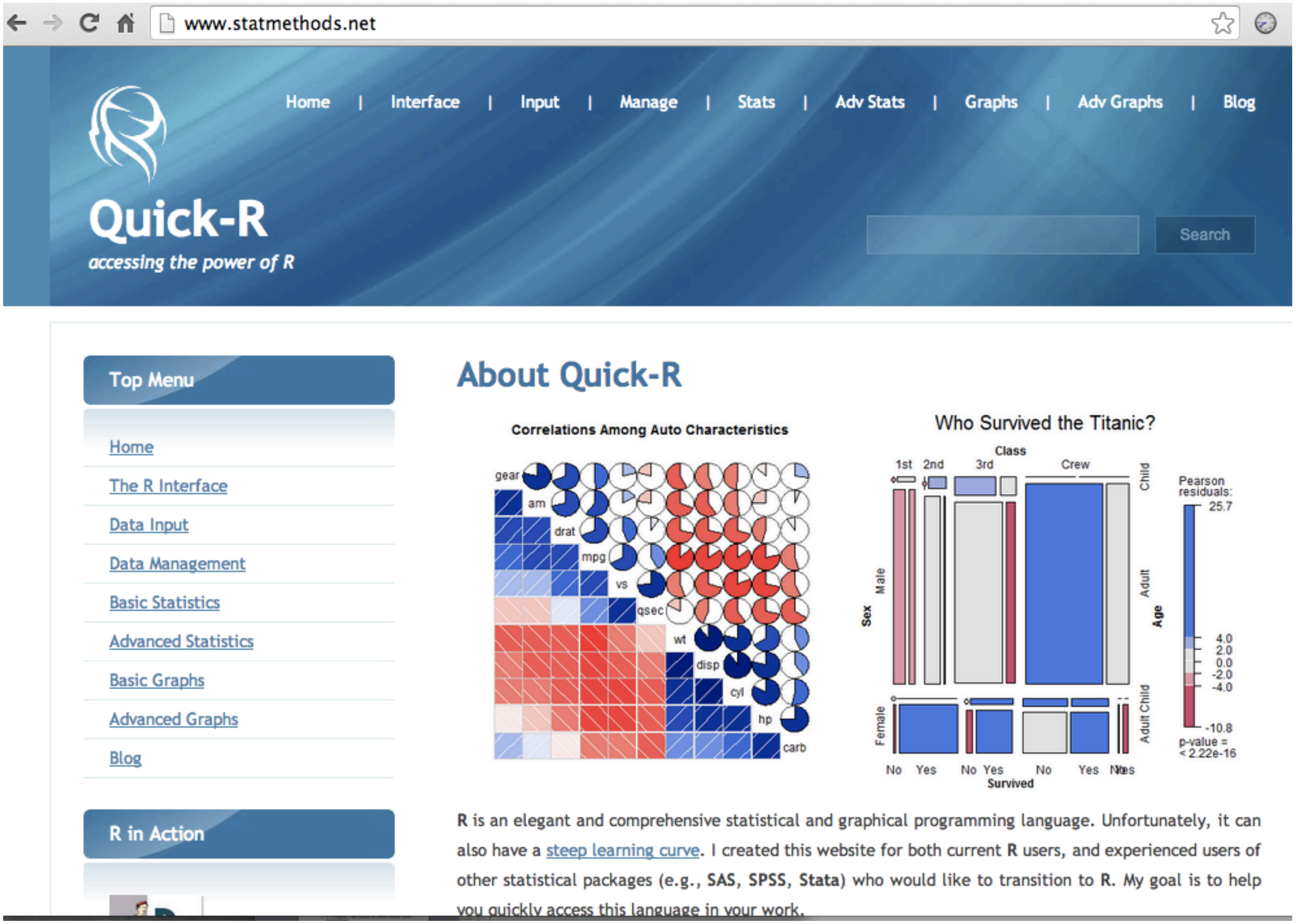
- Once R is installed, there is a comprehensive built-in help system. At the program's command prompt you can use any of the following:

help.start()	= general help
help(foo)	= help about function foo
?foo	= same thing
apropos("foo")	= list all functions containing string foo
example(foo)	=show an example of function foo

Sadly….

```
> apropos("t-test")
character(0)
```

```
> apropos("t.test")
[1] "bartlett.test"   "pairwise.t.test"
   "power.t.test"    "t.test"
```



www.statmethods.net

Home | Interface | Input | Manage | Stats | Adv Stats | Graphs | Adv Graphs | Blog

Quick-R

accessing the power of R

Search

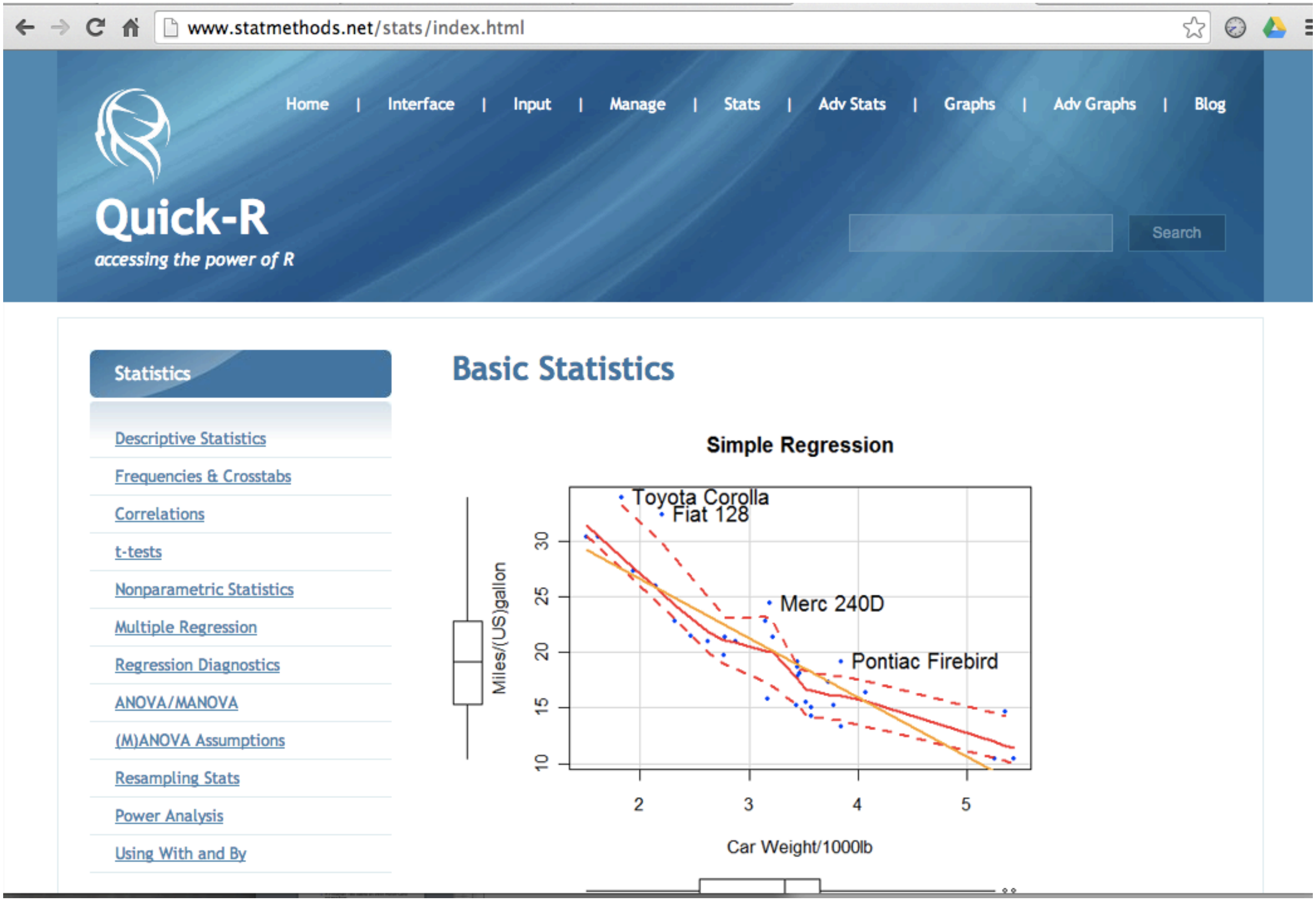
Top Menu

- Home
- The R Interface
- Data Input
- Data Management
- Basic Statistics
- Advanced Statistics
- Basic Graphs
- Advanced Graphs
- Blog

R in Action

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 current R users, and experienced users of other statistical packages (e.g., SAS, SPSS, Stata) who would like to transition to R. My goal is to help you quickly access this language in your work.



www.statmethods.net/stats/index.html

Home | Interface | Input | Manage | Stats | Adv Stats | Graphs | Adv Graphs | Blog

Quick-R

accessing the power of R

Search

Statistics

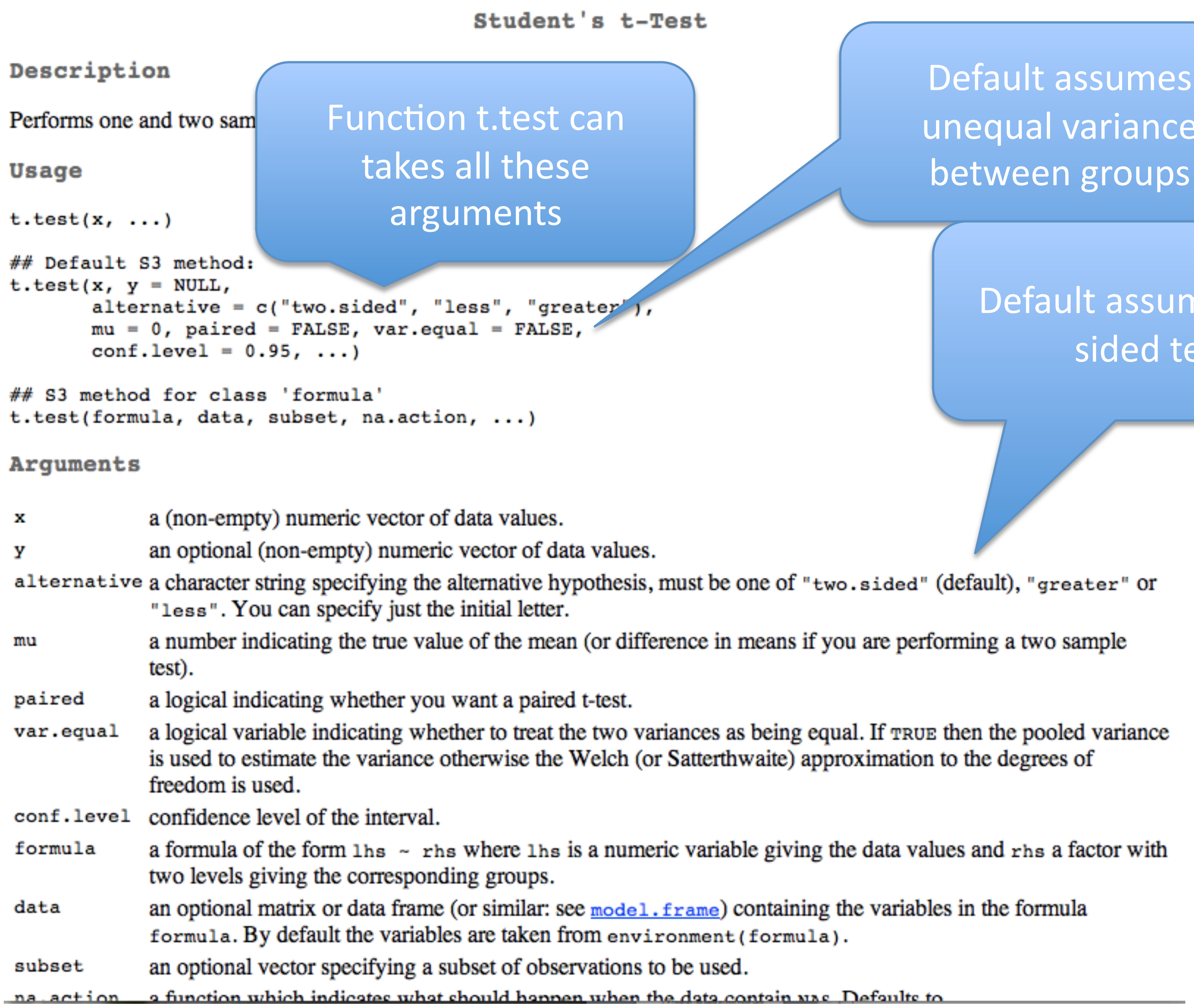
- Descriptive Statistics
- Frequencies & Crosstabs
- Correlations
- t-tests
- Nonparametric Statistics
- Multiple Regression
- Regression Diagnostics
- ANOVA/MANOVA
- (M)ANOVA Assumptions
- Resampling Stats
- Power Analysis
- Using With and By

Basic Statistics

t.test()

- The t.test( ) function produces a variety of t-tests. Unlike most statistical packages, the default assumes unequal variance and applies the Welsh df modification.# independent 2-group t-test
  t.test(y~x) # where y is numeric and x is a binary factor
- # independent 2-group t-test
  t.test(y1,y2) # where y1 and y2 are numeric
- # paired t-test
  t.test(y1,y2,paired=TRUE) # where y1 & y2 are numeric
- # one sample t-test
  t.test(y,mu=3) # Ho: mu=3

R help page: ?t.test



Student's t-Test

Description

Performs one and two sam

Usage

```
t.test(x, ...)

## Default S3 method:
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)

## S3 method for class 'formula'
t.test(formula, data, subset, na.action, ...)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`y`	an optional (non-empty) numeric vector of data values.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.
`mu`	a number indicating the true value of the mean (or difference in means if you are performing a two sample test).
`paired`	a logical indicating whether you want a paired t-test.
`var.equal`	a logical variable indicating whether to treat the two variances as being equal. If `TRUE` then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used.
`conf.level`	confidence level of the interval.
`formula`	a formula of the form `lhs ~ rhs` where `lhs` is a numeric variable giving the data values and `rhs` a factor with two levels giving the corresponding groups.
`data`	an optional matrix or data frame (or similar: see `model.frame`) containing the variables in the formula `formula`. By default the variables are taken from `environment(formula)`.
`subset`	an optional vector specifying a subset of observations to be used.
`na.action`	a function which indicates what should happen when the data contain `NA`s. Defaults to

Function t.test can takes all these arguments

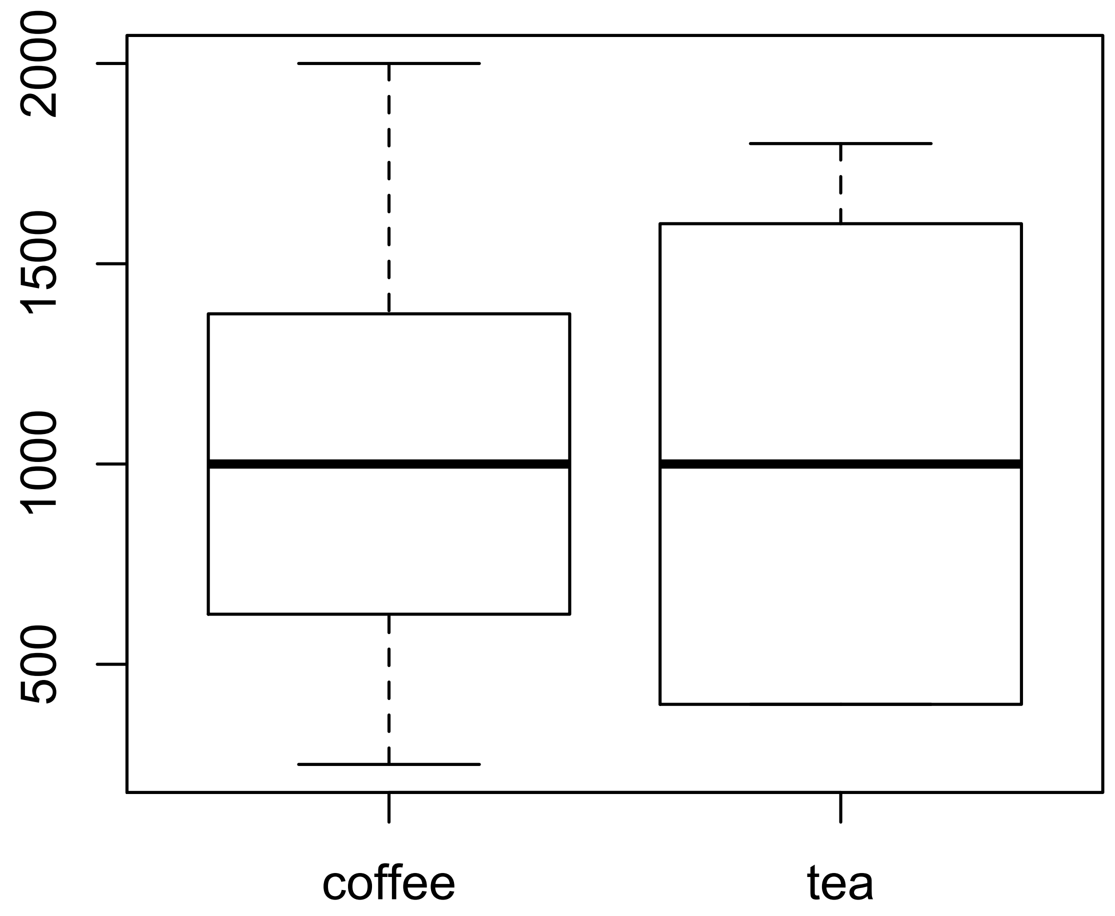
Default assumes unequal variance between groups

Default assumes two-sided test

Significantly different?

```
> t.test(volcoffee,voltea)

     Welch Two Sample t-test

data:  volcoffee and voltea
t = 0.0214, df = 11.939, p-value = 0.9833
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -721.5768  735.8625
sample estimates:
mean of x mean of y
 1035.714  1028.571
```

```
>boxplot(volcoffee, voltea, names=c("coffee", "tea"))
```



Is there a significant correlation between your coffee and tea drinking?

Hint: Use the cor.test() function

Correlation?

```
>cor.test(volcoffee,voltea)

     Pearson's product-moment correlation

data:  volcoffee and voltea
t = 0.3807, df = 5, p-value = 0.719
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.6698773  0.8175696
sample estimates:
      cor
0.1678578
```

Importing data

- One major use of R is to analyze previously collected data. Most often, you will import your data from another program like Excel.
- If you save your datafile as .csv formt, it is particularly easy to import into R as a dataframe.

Dataframes

- **Dataframes** are one to multi-dimensional sets, and can be composed of different data types (although all data in a single column must be of the same type).
- In addition, each column and row in a data frame may be given a label or name to identify it.
- Data frames are equivalent to a flat file database, and similar to spreadsheets.

Ways to import

- Use the OS to find the file

datafilename <- file.choose()

- Specify the name and address of the local file

e.g., datafilename <- "Desktop/epi.big5.txt”

The read the file into memory

person.data <- read.table(datafilename, header=TRUE) #read the data file

- Alternatively, to read in a comma delimited file:

person.data <- read.table(datafilename,header=TRUE,sep=",")

person.data <- read.csv(datafilename,header=TRUE)

Three ways to find and open a .csv file in R

1 Enter the command **mydata = read.csv(file.choose()**) in the Console window. This command opens a directory window and allows you to choose the file you wish to open.

2 If you know the location of your file, you could enter **mydata =read.csv("C:/Users/yourusername/Downloads/ hogweed.csv”)**

3 You could also simply change the directory that R is looking at. The command **getwd()** allows you to see which directory R is currently focusing on, and the command **setwd("C:/Users/ yourusername/Downloads /")** or similar should change the directory to the right location. Then you could enter **mydata = read.csv("hogweed.csv"**), since R would already be focusing on the correct directory.

Download the hogweed.csv file from ecotheory.uwaterloo.ca/Rintro.htm

Import the datafile into R

Data checking

- Once you have successfully read your data file into R, take a look at it!
- Simply type **mydata** to see if the data file was read in properly.
- Some datasets will be too large for this approach to be useful (the top of the data will scroll right off the page). In that case, there are a number of commands to look at a portion of the dataset.

Three ways to examine the dataset you read in.

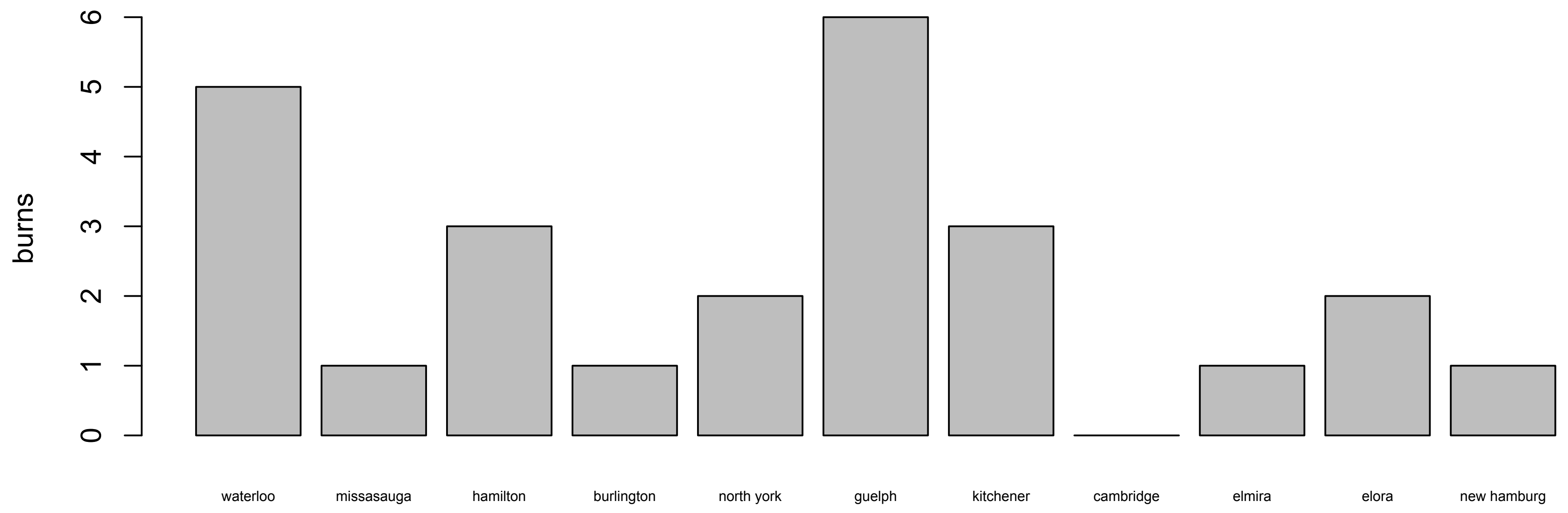
- You could use a command like **names (mydata),** but this has one obvious shortcoming, what is it?
- Instead, try typing **head(mydata).** What does this show you?
- How about **tail(mydata)?**
- Now for the big one, type **str(mydata).**

Three ways to reference a column in a dataframe

- The data file contains a bunch of different kinds of information (location, density of plants, number of reported burns)
- We can access each part of this data independently, by calling the specific component of the data frame (our data is frame data rather than matrix data since it contains both text and numbers).
- The command **mydata[,"reported.burns"] will give only the burns column of the mydata set, but so would mydata[, 3], since burns is the third column in our dataset. Finally the command mydata $reported.burns will also access the burns column**

Do a preliminary plot

- barplot(hogdata$burns, names=hogdata$area, cex.names=.5, ylab="burns")



Scripts

- We could just keep entering commands line by line in the command window, but often you will want to repeat analyses, or keep a permanent record.
- Let’s create a **Script.** A script is a nothing more than a record of a set of commands that you would like R to complete.
- File-> New Document

Summary stats

- The **summary()** command calculates summary statistics for our data, such as the mean and median.
- For the text-based components, R just tells us how many times each of the different text words appear. For the numeric data, R reports the max, min, mean and median.
- If you don't want to summarize everything, you can calculate the mean and standard deviation of particular variables using the **mean(yourvariable) and sd(yourvariable)** commands.

Start your script

Add the commands to calculate summary data

Running scripts

- run your script by either selecting the all the script text (Cntr+A or Cmd+A) or part of it (use the mouse) and sending it to the console window (Cntrl+R or Cmd+return)
- or by setting the working directory appropriately and typing **source (“myscript.r”)**

ANOVA

- if x is categorical data lm(y~x) performs a oneway ANOVA, and the ANOVA table can be seen by typing anova(linearmodeloutput)

- Can also use aov(y~x)

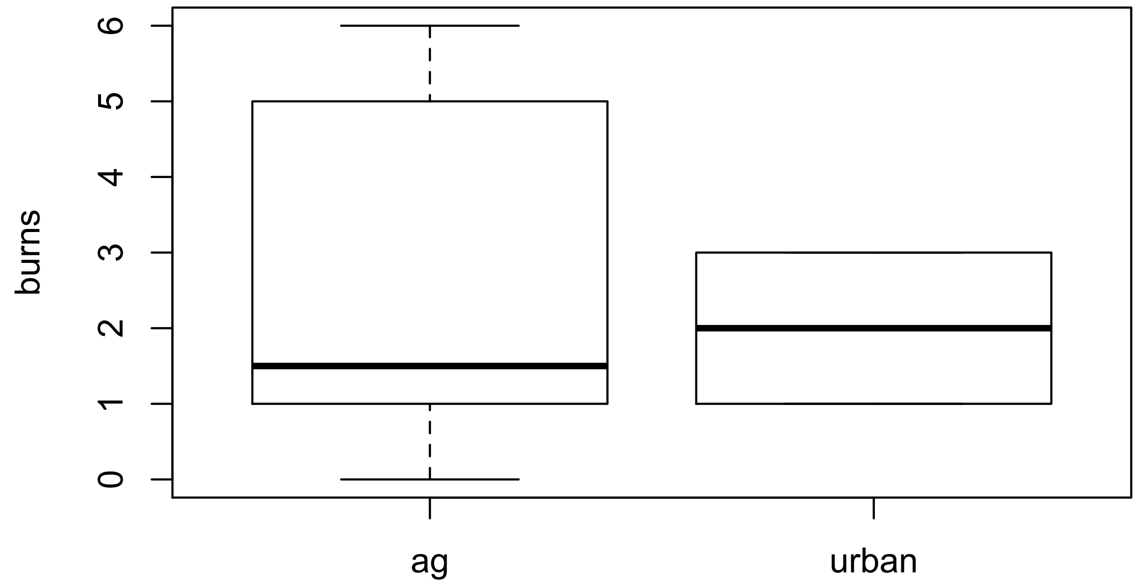
ANOVA (one-way)

- Does the incidence of hogweed burns differ between agricultural and urban areas?

```
anova(lm(burns~land.use,
  data=hogdata))
boxplot(burns~land.use,
  data=hogdata, ylab="burns")
```

Add the anova commands to your script and implement them

Is there a significant effect of land use on hogweed burns?



```
> anova(lm(burns~land.use, data=hogdata))
Analysis of Variance Table

Response: burns
          Df Sum Sq Mean Sq F value Pr(>F)
land.use   1  0.682  0.6818  0.1832 0.6787
Residuals  9 33.500  3.7222
```

Regression

- Regression in R is completed using the linear model lm() command. The basic syntax for a regression analysis in R is: **lm(Y~model)** where Y is the object containing the numeric dependent variable to be predicted and model is the formula for the chosen statistical fit and contains the numeric independent variable.
- For example, if x and Y are numeric data, the statement lm(Y~x) is shorthand for a request to R to fit the statistical model $Y= mx + b + e$ to the data, where the best fit line is estimated using least-squares regression.
- The results of the regression can be seen by typing summary(myregressionoutput).

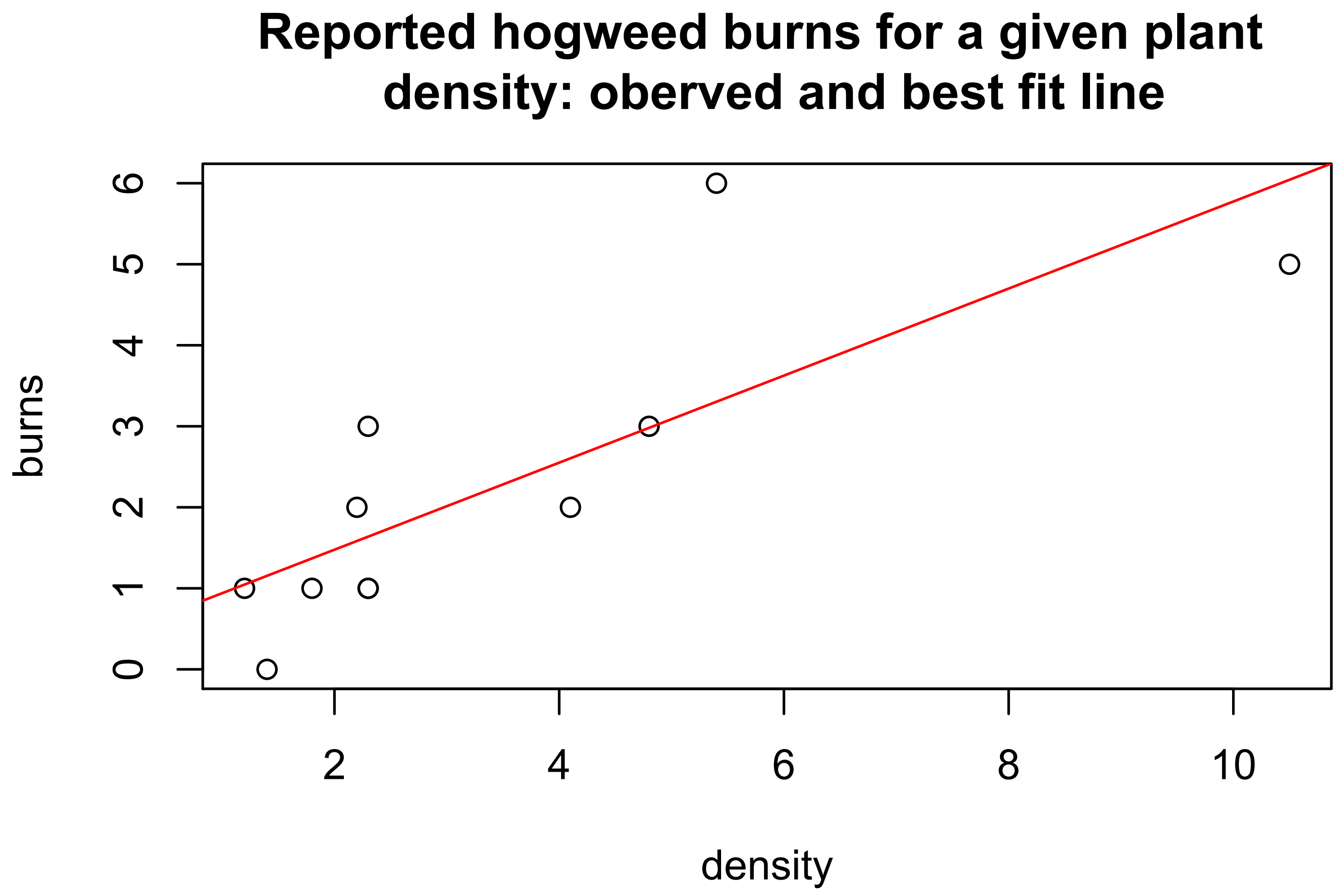
Regression

- Does hogweed density predict the number of hogweed burns?

```
lreg.burns<-lm(burns~density,
  data=hogdata)
summary(lreg.burns)
```

```
> summary(lreg.burns)

Call:
lm(formula = burns ~ density, data = hogdata)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.1543 -0.6378 -0.3692  0.2175  2.6968 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)   
(Intercept)   0.4022     0.6095    0.66  0.52582   
density       0.5372     0.1406    3.82  0.00409 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 1.204 on 9 degrees of freedom
Multiple R-squared:  0.6185,  Adjusted R-squared:  0.5761 
F-statistic: 14.59 on 1 and 9 DF,  p-value: 0.00409
```

```
plot(burns~density, data=hogdata)
abline(lreg.burns, col="red")
title("Reported hogweed burns for a given plant
  \n density: observed and best fit line")
```



Lists

- **Lists** are compound objects of associated data.
- Like data frames, they need not contain only a single data type, but can include strings (character variables), numeric variables, and even such things as matrices and data frames.
- In contrast to data frames, list items do not have a row-column structure, and items need not be the same length; some can be a single value, and others a matrix.

ANOVA (two-way)

- Does land use and the pattern of distribution predict hogweed burns?

ANOVA WARNING:

- R provides Type I sequential SS, not the default Type III marginal SS reported by SAS and SPSS.
- In a nonorthogonal design with more than one term on the right hand side of the equation order will matter (i.e., A+B and B+A will produce different results)!
- We will need use the drop1( ) function to produce the familiar Type III results. It will compare each term with the full model.

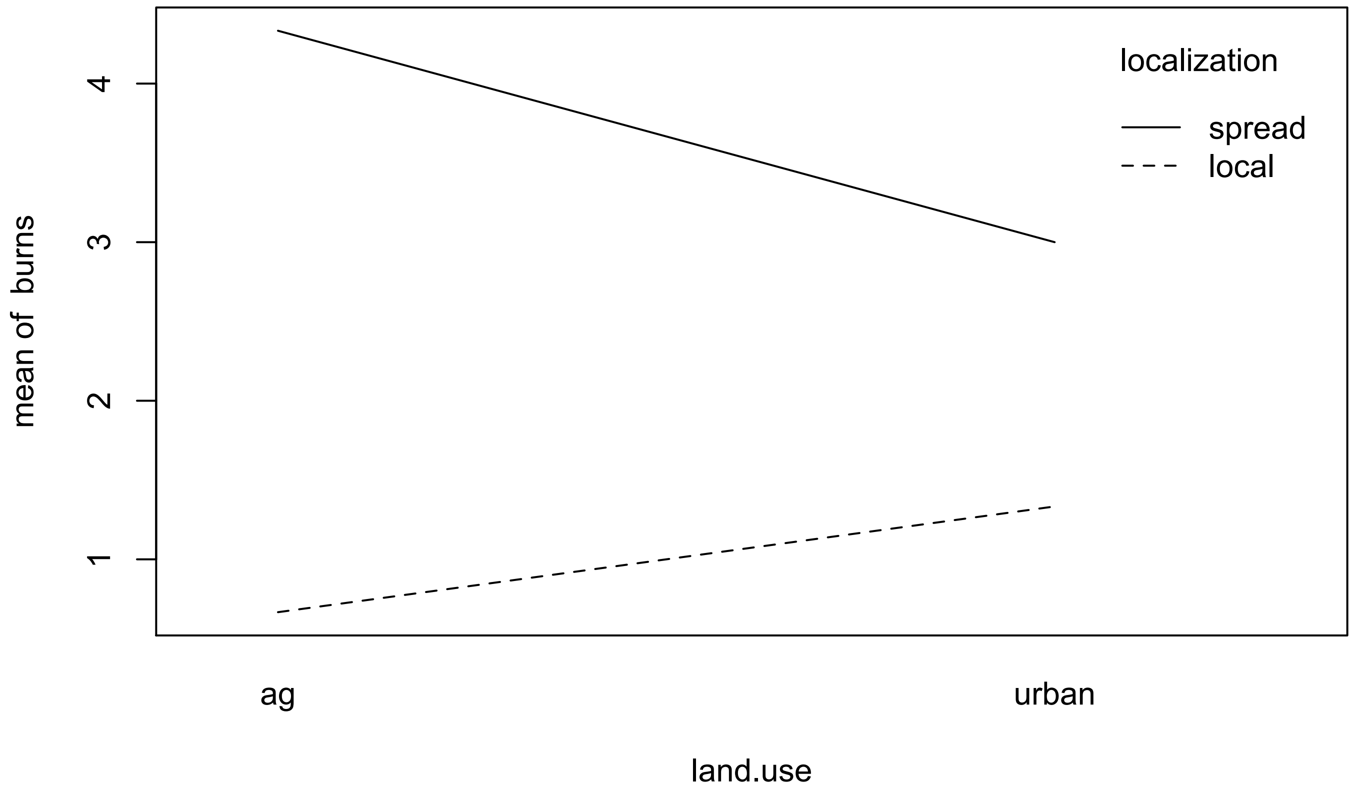
ANOVA (two-way)

```
an2.burns<-lm
  (burns~land.use*localization,
  data=hogdata)
#Type I SS
anova(an2.burns)
#Type III SS
drop1(an2.burns,~.,test="F")
```

```
> #Type I SS
> anova(an2.burns)
Analysis of Variance Table

Response: burns
                     Df  Sum Sq Mean Sq F value   Pr(>F)
land.use              1  0.6818  0.6818  0.4773 0.511911
localization          1 20.8333 20.8333 14.5833 0.006552 **
land.use:localization  1  2.6667  2.6667  1.8667 0.214125
Residuals             7 10.0000  1.4286
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> #Type III SS
> drop1(an2.burns,~.,test="F")
Single term deletions

Model:
burns ~ land.use * localization
                     Df Sum of Sq    RSS     AIC F value   Pr(>F)
<none>                             10.000  6.9516
land.use              1    0.6667 10.667  5.6615  0.4667 0.516490
localization          1   20.1667 30.167 17.0973 14.1167 0.007101 **
land.use:localization  1    2.6667 12.667  7.5519  1.8667 0.214125
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Interaction plot

- `attach(hogdata)`
- `interaction.plot(land.use, localization, burns)`



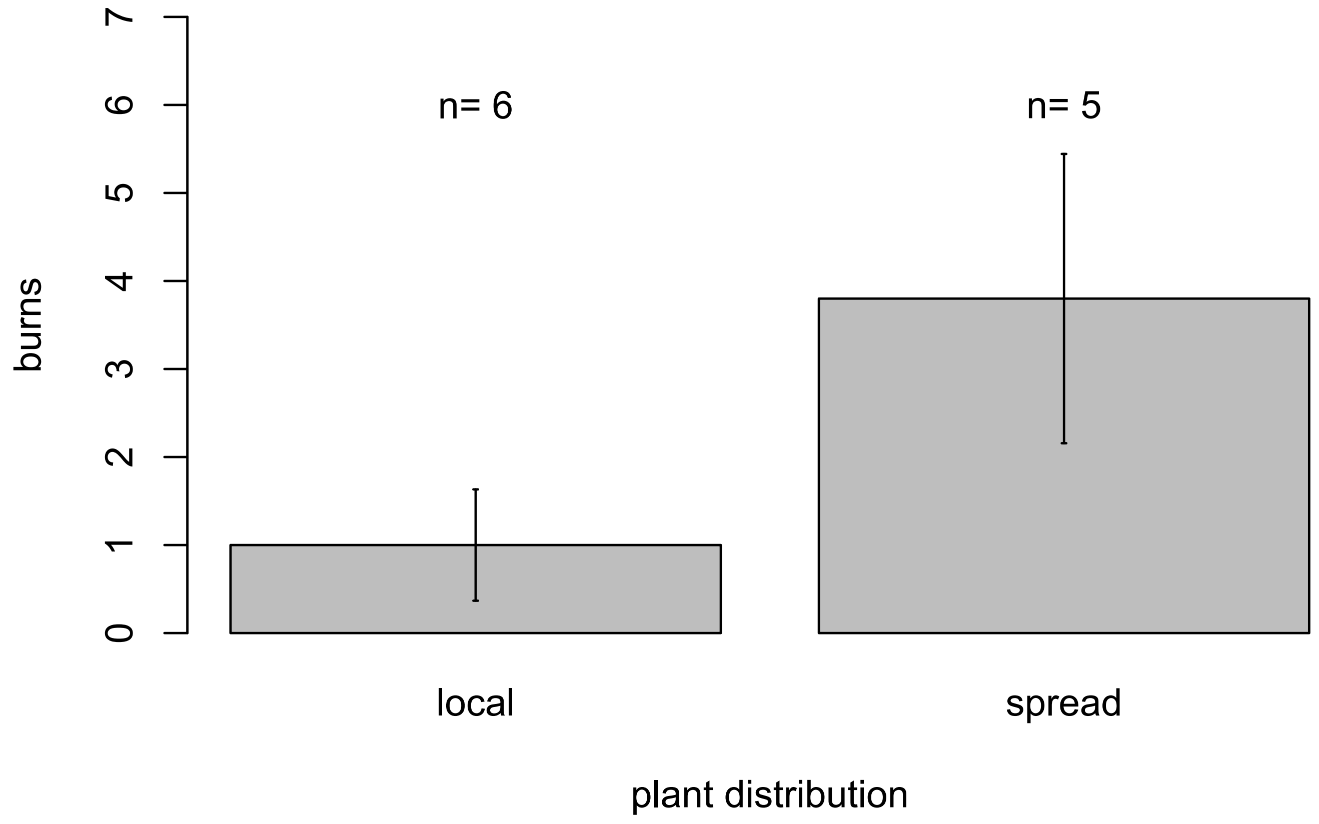
Post-hoc comparison of means

- `TukeyHSD(aov(an2.burns))`

```
 Tukey multiple comparisons of means
   95% family-wise confidence level

Fit: aov(formula = an2.burns)

$localization
             diff      lwr      upr     p adj
spread-local 2.75 1.038611 4.461389 0.0067177
```

Barplot with error bars

```
mean.burns<-tapply(hogdata$burns, list
  (hogdata$localization), mean)
sd.burns<-tapply(hogdata$burns, list
  (hogdata$localization), sd)
n.burns<-tapply(hogdata$burns, list
  (hogdata$localization), length)
mids<-barplot(mean.burns, xlab="plant
  distribution", ylab="burns", ylim=c(0,
  7))
arrows(mids, mean.burns-sd.burns, mids,
  mean.burns+sd.burns, code=3, angle=90,
  length=0.01)
text(mids, 6, paste("n=", n.burns))
```



n= 6
n= 5
burns
0
1
2
3
4
5
6
7
local
spread
plant distribution

Pub quality figures

- Default plot will save as .pdf
- output graph to jpeg file

```
jpeg("c:/mygraphs/myplot.jpg")
plot(x)
dev.off()
```

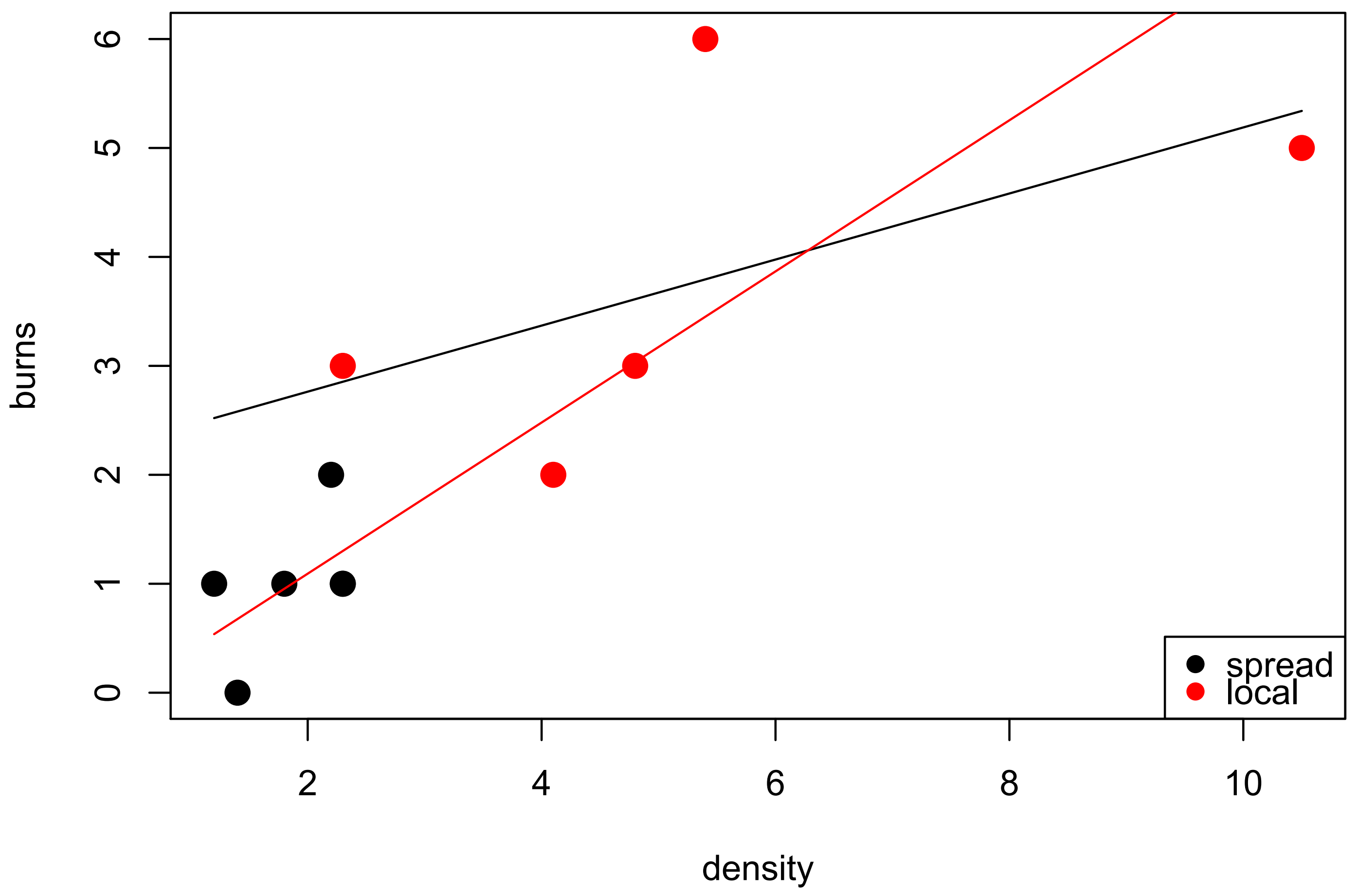
ANCOVA

- Is there a different relationship between hogweed density and hogweed burns for areas with concentrated occurrence vs. areas with widespread occurrence (i.e. different slopes of the linear relationship)?

```
anc.burns<-lm
  (burns~density*localization,
  data=hogdata)
anova(anc.burns)
```

Plot data and regression lines

```r
plot(burns~density, data=hogdata, pch=19, cex=1.5,
  col=c("black", "red")[hogdata$localization],
  xlab="density", ylab="burns")
legend("bottomright", legend=c("spread", "local"),
  col=c("black", "red"), pch=c(19, 19))
new.x<-expand.grid(density=unique(hogdata$density),
  localization=levels(hogdata$localization))
anclines<-predict(anc.burns, newdata=new.x)
preds<-data.frame(new.x,anclines)
idx <- order(preds$density)
sorted <- preds[idx,]
lines(anclines~density, data=subset(sorted,
  localization=="spread"))
lines(anclines~density, subset(sorted,
  localization=="local"), col="red")
```


burns
0
1
2
3
4
5
6
2
4
6
8
10
density
spread
local

A small selection of resources

- http://cran.r-project.org/manuals.html
- http://www.statmethods.net/
- http://plantecology.syr.edu/fridley/bio793/nichenotes_portal.html
- http://www.instantr.com/
- http://www.ats.ucla.edu/stat/r/
- https://www.coursera.org/course/compdata
- http://tryr.codeschool.com/