Rbootcamp/Workshop

Michael Kleinsasser

9/9/2019

Mike's Personal Introduction

- R programmer for the Department of Biostatistics
- Write and mantain R packages for faculty and students
- Consult faculty and students on writing R packages, optimization

Rbootcamp Introduction

- Load, view, manipulate a data set
- Apply to modeling functions, inspect outputs
- Some details of data structures and getting help
- After slides: practice using lab#0 script and data set
    - The end of that script contains some exercises

R Basics: Big Picture

- R is a sophisticated calculator for statistics

Chambers (2016) Extending R:

- Everything that exists in R is an object
- Everything that happens in R is a function call

Obtain a basic working knowledge of R objects and functions,

- Google the rest!

Use R as a calculator

Standard operations:

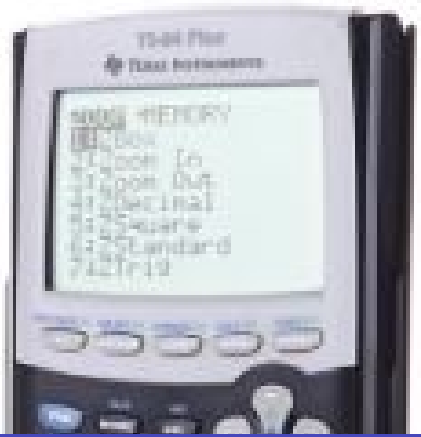
```
# multiply *, divide /, add + subtract -
18056.983 - 1005.118 + 22.53
```

```
## [1] 17074.4
```

```
( (pi - 3.14) / (3.14) )  * 100
```

```
## [1] 0.05072145
```

- Can do anything your Ti84 does, and then some



Using R's sophisticated functions

How is R extended beyong basic calculator functionality?

MANY functions to carry out statistical analyses have been implemented.

To use these packages, you need to install, load and attach them:

```
install.packages("readr")    # read text data
install.packages("dplyr")    # manipulate data
```

Load the packages into your R session:

```
library(readr)
library(dplyr)
```

Now, we can use the functions from those packages:

```
mutate() # from dplyr

read_csv() # from readr
```

Example data for this tutorial

The Effect of Vitamin C on Tooth Growth in Guinea Pigs:

The response is the length of odontoblasts (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, orange juice or ascorbic acid (a form of vitamin C and coded as VC).

Format: - A data frame with 60 observations on 3 variables.

- [,1] len numeric Tooth length
- [,2] supp factor Supplement type (VC or OJ).
- [,3] dose numeric Dose in milligrams/day

Read in data

For this tutorial we use the ToothGrowth data set:

- Two ways to read in data:

1. Point and click in RStudio << Do this for now (easy)
2. read.csv()

```
ToothGrowth = read.csv("./slides/ToothGrowth.csv")
ToothGrowth = read.csv("slides/ToothGrowth.csv")
ToothGrowth = read.csv("ToothGrowth.csv")
```

Create object

Create object with "assign" operator

- arrow then minus sign <-
- single equal sign =

```
ToothGrowth <- read.csv("./slides/ToothGrowth.csv")

# equivalently
ToothGrowth = read.csv("./slides/ToothGrowth.csv")
```

Note that ToothGrowth is an arbitrary name. I can choose whatever name I desire. Choose names that help you remember what the object is.

Inspect the data

The first step with any data you encounter is to view the dataset and take a look at what type of information is contained within.

To do that, use the
'View' function. Note that this function, unlike most other R function, is capitalized.

```
View(ToothGrowth)
```

View is a good way to spot check your work!

Useful functions for inspecting your data

First, let's see how many observations and variables are contained in the dataset. The 'dim' function will return two numbers: the first is the number of rows (observations) and the second is the number of columns (variables)

```
# insert the object to get 'dimension' of
dim()
```

Useful functions for inspecting your data

Dig deeper into the data. Recall, each row is an observation (Guinea Pig). Columns are variables, e.g. what we measured on each pig.

- We need to know what kind of data these columns contain
- Next slide has some classes of data:

Classes of ToothGrowth

The classes of variables (columns of data):

- numeric, comes in two flavors: integer, numeric
- character
- logical
- complex
- factor

```
class(ToothGrowth$len)
class(ToothGrowth$supp)
class(ToothGrowth$dose)
```

Note: The modeling functions you use will care what data types you send in

Classes of ToothGrowth, in detail

```
class(ToothGrowth$len)
```

Tooth cell length is 'numeric', meaning meaning R is treating it as a continuous numeric variable

```
class(ToothGrowth$supp)
```

Supplement type is a 'factor', indicating that R is treating supp as a categorical variable (restricted set of levels, unordered)

```
class(ToothGrowth$dose)
```

Inspect types of data in ToothGrowth

We saw the classes of the columns (variables) of ToothGrowth

But what about the object ToothGrowth itself? Find out what it is:

```
class()  # insert the correct object.
```

How do these objects work?

The data.frame

The **data.frame** is the most common way to store and work with data in R

- Not surprising: they are designed for this purpose
- Most modeling functions work on data.frames

Composed of a 'list' of equal length atomic vectors (can be of any type, but within the vector each element must be of the same type)

```
# look at the top of the data.frame (first few rows):
head(ToothGrowth)

# bottom
tail(ToothGrowth)
```

Working with data set: simple summaries

Now that we have taken a surface glance at the dataset, let's look deeper into some of the variables in the data.

One important statistic to find for variables is the sample average (mean).

Let's start out by finding the mean for the response 'tooth cell length'.

```
ToothGrowth$len # This call returns the vector of length data

# Pass length data into the function 'mean'
mean(ToothGrowth$len)  # returns sample mean
```

Working with data set: simple summaries

Next, create a histogram to see the distribution of the length variable

```
hist(ToothGrowth$len, main="Histogram for Length",
     xlab="Length", col="gray")
```

The distribution for Length can also be viewed using a boxplot:

```
boxplot(, ylab="Length")    # insert the correct object
```

Working with data set: simple summaries

Histograms and boxplots are used for numeric variables such tooth length.

The dataset also contains categorical varaibles as well, for example supplement (supp)

First, view the levels for the supp variable:

```
levels(ToothGrowth$supp)
```

Create a table with counts for each level of the categorical variable using the table function:

```
table(ToothGrowth$supp)
```

A barchart is used to visualize counts for the categorical variable: Note that the col argument lets you change the colors of each bar

```
barplot(table(ToothGrowth$supp), col=c("red", "blue"))
```

Working with data set: manipulate data

The variable names that come with a dataset can often be cumbersome.

We can rename a variable to something simpler and more intuitive using the rename function. The example below changes the variable name from "supp" to "supplement."

```
ToothGrowth2 = rename(ToothGrowth, supplement=supp)
```

Note: saved this as ToothGrowth2 to make explicit something in R:

Copy on modify. Can also make a new copy and resave with the same name, thus overwriting the original data:

```
#ToothGrowth = rename(ToothGrowth, supplement=supp)
```

Working with data set: manipulate data

We can create new variables in the dataset using the mutate function.

Imagine that we would like to indicate pigs who were given a dose level greater than 0.5. Might do this because the client asked whether there is a difference in tooth cell length comparing dose $= 0.5$ to dose > 0.5.

We will create a new variable called 'dose_gr05' that is '1' if the dose is over .5 and '0' if dose is equal to 0.5.

```
ToothGrowth <- mutate(ToothGrowth, dose_gr05=ifelse(dose>0.5,1
View(ToothGrowth)
```

Note: We used the 'ifelse' function to create the new variable.

This function takes three arguments: 1. The logical expression (here, if dose > 0.5), 2. the value if the expression is TRUE (here, 1) 3. and the value if the expression is false (here 0).

Common R functions

R has a huge collection of packages:

- 6,000+ packages for data analysis build (on CRAN alone)

Example: lm (linear models)

- Use ?lm to read help documentation

lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.

Fit a linear model with lm

- Use built-in data set ToothGrowth
- ?ToothGrowth for help:

The Effect of Vitamin C on Tooth Growth in Guinea Pigs

Description

The response is the length of odontoblasts (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, orange juice or ascorbic acid (a form of vitamin C and coded as VC).

Usage

```
ToothGrowth
```

Format

A data frame with 60 observations on 3 variables.

[,1] len numeric Tooth length
[,2] supp factor Supplement type (VC or OJ).
[,3] dose numeric Dose in milligrams/day

Using lm() function for linear models

Call lm on the data and formula, store result "lm" object:

```
tooth_fit = lm(formula = len ~ supp + dose,
               data = ToothGrowth)
```

Formulas in R:

```
len ~       # Response column name, ~ for "="
  supp +    # First predictor name + for "+"
  dose      # second predictor name
```

Many R functions use the formula argument.

Getting detailed information

Basic "print" of model:

```
print(tooth_fit)     # equivalent to tooth_fit
```

```
##
## Call:
## lm(formula = len ~ supp + dose, data = ToothGrowth)
##
## Coefficients:
## (Intercept)       suppVC         dose
##       9.272       -3.700        9.764
```

Detailed summary:

```
summary(tooth_fit)
```

```
##
## Call:
## lm(formula = len ~ supp + dose, data = ToothGrowth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.600 -3.700  0.373  2.116  8.800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.2725     1.2824   7.231 1.31e-09 ***
## suppVC       -3.7000     1.0936  -3.383   0.0013 **
## dose          9.7636     0.8768  11.135 6.31e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

Understanding R classes

- What is this thing?

```
class(tooth_fit)
```

- What are the methods for this object?

```
methods(class = "lm")
```

- What is its structure? (i.e., what's in it)

```
str(tooth_fit)
```

Understanding R classes

- What is this thing?

```
class(tooth_fit)
```

```
## [1] "lm"
```

Understanding R classes

- What are the methods for this object?

```
methods(class = "lm")
```

```
##  [1] add1             alias            anova            case.names
##  [5] coerce           confint          cooks.distance   deviance
##  [9] dfbeta           dfbetas          drop1            dummy.coef
## [13] effects          extractAIC       family           formula
## [17] hatvalues        influence        initialize       kappa
## [21] labels           logLik           model.frame      model.matrix
## [25] nobs             plot             predict          print
## [29] proj             qr               residuals        rstandard
## [33] rstudent         show             simulate         slotsFromS3
## [37] summary          variable.names   vcov
## see '?methods' for accessing help and source code
```

Understanding R classes

- What is its structure? (i.e., what's in it)

```
str(tooth_fit)
```

```
## List of 13
##  $ coefficients : Named num [1:3] 9.27 -3.7 9.76
##   ..- attr(*, "names")= chr [1:3] "(Intercept)" "suppVC" "dose"
##  $ residuals    : Named num [1:60] -6.25 1.05 -3.15 -4.65 -4.05 ...
##   ..- attr(*, "names")= chr [1:60] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:60] -145.73 14.33 47.16 -3.86 -3.26 ...
##   ..- attr(*, "names")= chr [1:60] "(Intercept)" "suppVC" "dose" "" ...
##  $ rank         : int 3
##  $ fitted.values: Named num [1:60] 10.5 10.5 10.5 10.5 10.5 ...
##   ..- attr(*, "names")= chr [1:60] "1" "2" "3" "4" ...
##  $ assign       : int [1:3] 0 1 2
##  $ qr           :List of 5
##   ..$ qr   : num [1:60, 1:3] -7.746 0.129 0.129 0.129 0.129 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:60] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:3] "(Intercept)" "suppVC" "dose"
##   .. ..- attr(*, "assign")= int [1:3] 0 1 2
##   .. ..- attr(*, "contrasts")=List of 1
##   .. .. ..$ supp: chr "contr.treatment"
##   ..$ qraux: num [1:3] 1.13 1.11 1.11
##   ..$ pivot: int [1:3] 1 2 3
##   ..$ tol  : num 1e-07
##   ..$ rank : int 3
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 57
##  $ contrasts    :List of 1
##   ..$ supp: chr "contr.treatment"
##  $ xlevels      :List of 1
```

Understanding R classes

- Pull something out of the "lm" fit object:

```
tooth_fit$fitted.values        # y_hat's for the linear model
```

```
##        1        2        3        4        5        6        7        8
## 10.45429 10.45429 10.45429 10.45429 10.45429 10.45429 10.45429 10.45429
##        9       10       11       12       13       14       15       16
## 10.45429 10.45429 15.33607 15.33607 15.33607 15.33607 15.33607 15.33607
##       17       18       19       20       21       22       23       24
## 15.33607 15.33607 15.33607 15.33607 25.09964 25.09964 25.09964 25.09964
##       25       26       27       28       29       30       31       32
## 25.09964 25.09964 25.09964 25.09964 25.09964 25.09964 14.15429 14.15429
##       33       34       35       36       37       38       39       40
## 14.15429 14.15429 14.15429 14.15429 14.15429 14.15429 14.15429 14.15429
##       41       42       43       44       45       46       47       48
## 19.03607 19.03607 19.03607 19.03607 19.03607 19.03607 19.03607 19.03607
##       49       50       51       52       53       54       55       56
## 19.03607 19.03607 28.79964 28.79964 28.79964 28.79964 28.79964 28.79964
##       57       58       59       60
## 28.79964 28.79964 28.79964 28.79964
```

Extracting data from model objects

Some generic extraction methods:

```
coef(tooth_fit)                 # model coefficients

coef(summary(tooth_fit))        # adds test statistics, p-values

vcov(tooth_fit)                 # variance/covariance matrix
```

Note: depending on implementation, these may not be available - Check methods with "methods(object)" before attempting

Extracting data from model objects (in detail)

Extract coefficients, test stats, and p-values

```
coef(summary(tooth_fit))   # adds test statistics, p-values
```

```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)  9.272500  1.2823649  7.230781 1.312335e-09
## suppVC      -3.700000  1.0936045 -3.383307 1.300662e-03
## dose         9.763571  0.8768343 11.135025 6.313519e-16
```

Predict new values

- predict() function is generic and works with many models
- Pass in a new data.frame with the same column names:

```
to_predict = data.frame(dose = 0.5, supp = "VC")
```

```
predict(tooth_fit, newdata = to_predict)
```

```
##        1
## 10.45429
```

Predict new values (example 2)

- predict() function is generic and works with many models
- Pass in a new data.frame with the same column names:

```
to_predict = data.frame(dose = seq(0,1,0.1), supp = "OJ")
```

```
predict(tooth_fit, newdata = to_predict)
```

```
##        1        2        3        4        5        6        7
##  9.27250 10.24886 11.22521 12.20157 13.17793 14.15429 15.13064 16
##        9       10       11
## 17.08336 18.05971 19.03607
```

Summary

What have we learned?

- Read in some data
- Work with it
- What is it?
- Manipulate it
- Fit a model to it
- Inspect the results

Questions?

- Open the lab#0 and run through the script with a data set
  - Do the exercises at the end