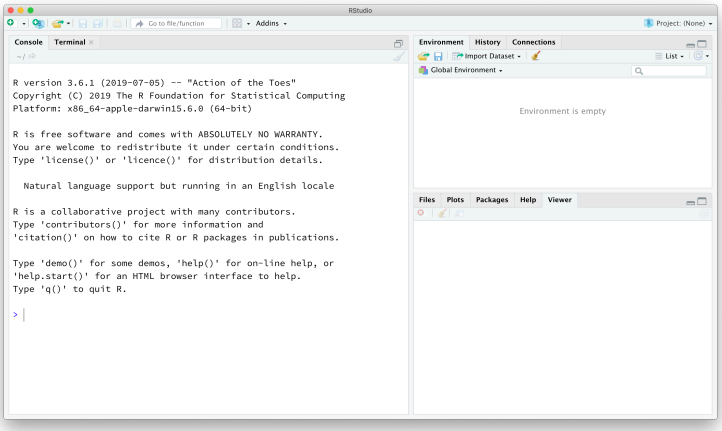
Intro to R / Baseball

Zachary Treisman

Overview

This lab introduces some basic data concepts, and you will begin exploring data in R. We will access R via the program RStudio which is an interface, or IDE (Interactive Development Environment).

When you start RStudio the first time, you will see something like this:



The window is split into three panels. Soon the left side of the window will grow another panel. Each panel has multiple tabs.

The panel in the upper right shows your workspace - the data, variables and special functions that you are currently working with - in the *Environment* tab. The *History* tab lists the commands that you've previously entered.

The lower right panel has a simple browser in the *Files* tab, displays graphics in the *Plots* tab, helps you install and load additional libraries of functions and data in the *Packages* tab, and lets you navigate the *Help* system.

The panel on the left is where the action happens. The main tab shows the *Console*. The `>` is called the *prompt*. The prompt is prompting you to type a command. Initially, interacting with R is all about typing commands and interpreting the output.

The Console and entering commands

To get you started, enter the following commands at the R prompt (i.e. right after `>` on the console). You can either type them in manually or copy and paste them from this document.

```
2+3
2*3
2^3
sqrt(9)  # This is a comment.
#R will ignore any code preceded by a number sign.
```

Two very important commands: `<-` is called "left assignment". Whatever is on the left of the arrow becomes a name you can use to refer to whatever is on the right of the arrow. You can use `=` instead, but the arrow can add clarity. The other arrow `->` for "right assignment" can also be used. `c()` stands for "concatenate" or maybe "combine". It makes a vector of whatever is inside the parentheses.

```
x <- c(1,3,2,5) # Define a vector 1, 3, 2, 5 and call it x.
# Notice that x just appeared in your workspace in the upper right.
x # Report the current value of x.
x+2 # Add 2 to each element of x. y <- c(4,2,-3,0)
x+y # Add the two vectors.
```

R is a language for doing statistics. So of course some of the first commands you will use are for basic statistical computations.

```
mean(x)
sum(x)
max(x)
min(x)
length(x)
```

We can redefine `x`, even in terms of the current value of `x`.

```
x <- c(1,6,2)
x
x <- x*2
x
```

Any command or object that comes with R has a help file. Type a question mark before something that you would like to know about. For example try `?mean`.

For vector and matrix multiplication use `%*%` instead of `*`. Transpose is `t()`.

```
A <- matrix(data=c(1,2,3,4), nrow=2, ncol=2)
x <- c(10, 100)
t(x)%*%x # dot product A%*%A  A%*%x
```

To get the entries of a matrix, vector or other data structure, use square brackets.

```
x[1]
A[1,2]
(t(x)%*%x)[1,1]
```

In your previous encounters with statistics, you used distributions such as the normal distribution (also called the Gaussian distribution) and the uniform distribution. R knows about these and many more. For any distribution, there are four R functions: the distribution function (d), the cumulative probability function (p), the quantile function (q) and the random number generator (r).

```
dunif(x=0.1, min=0, max=1) # Uniform Distribution: Constant 1/(max-min).
dnorm(x=0.1, mean=0, sd=1) # Normal Distribution: The bell curve.
pnorm(q=0, mean=0, sd=1)   # Half of the possible values are less than the mean.
qnorm(p=0.5, mean=0, sd=1) # And the mean is the number where half of possible
                           # values are less than it.

x<-runif(50) # Generate 50 random numbers from a uniform distribution.
y<-rnorm(50,mean=2*x+5,sd=.1) # Put these random inputs into a linear function
                              # and add Gaussian noise.
plot(y~x) # Show the data.
cor(x,y) # Correlation coefficient
```

R Packages

R is an open-source programming language, meaning that users can contribute packages that make our lives easier, and we can use them for free. For this lab, and many others in the future, we will use the following:

- **ggplot2** for graphics
- **dplyr** for reshaping and summarizing data

- **openintro** for some nice data sets

In the lower right hand corner click on the *Packages* tab. You can search for packages to install and load here. Alternatively you can type commands into the console.

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("openintro")
```

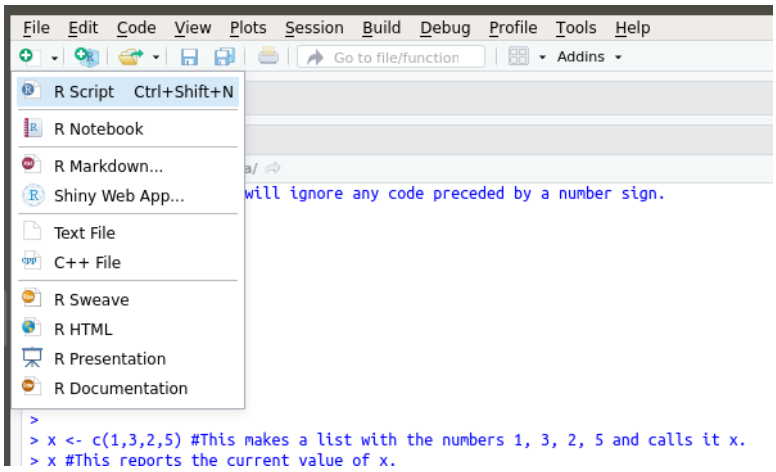
You only need to *install* packages once, but you need to *load* them each time you relaunch R. Load packages with the `library` function.

```
library(ggplot2)
library(dplyr)
library(openintro)
```

A common mistake comes from the fact that when you install a package the name of the package goes in quotes but when you load a package from your library the name does not get quotes. The reason for this difference is legitimate but not really important at this point. If it bugs you, ask me to explain.

R scripts

From the drop down menu in the upper left with the green + on a white square, create a new R Script. This will give you a panel in the upper left of your RStudio window where you can input R code. Unlike commands typed into the console, this code can be edited and will not be executed until you tell R to do so.



In the upper right of your screen, choose the *History* tab and select some or all of the code you just entered. You can use Shift+Mouse to select multiple lines. Press the *To Source* button - you should see those lines you just entered appear in your script window. This is handy for when you are trying various things in the console and find some commands that you want to save. Since you only need to install packages once on your computer, you probably want to either delete the lines with the `install.packages` commands. Alternatively, comment them out by putting a `#` at the beginning of each line.

You can run commands from your script by putting your cursor on the line that you would like to run and either clicking on the *Run* button in the upper right of the script window, or using *Control+Enter* (*Command+Enter* on a Mac). Run a command from your script and observe that the command and its output appeared in the console. You can run just part of a line or multiple lines by selecting exactly what you want to run from your script with the mouse.

As you work in R, you will find yourself going back and forth between entering commands at the prompt and composing scripts.

A Data Science Example

In this class we are developing tools used for data science. We will learn an outline of a primary data science task by exploring some real data.

Load the data

Today, we will use an example data set from the openintro package. Next week, we will learn how to import our own data. The data that we'll use are some batting statistics from the 2018 Major League Baseball season. Here's how to load and view the data.

```
data("mlb_players_18") # Load the data into your Environment.

head(mlb_players_18) # Look at the first 6 rows of data as output.
View(mlb_players_18) # Explore the data set in spreadsheet form in RStudio.
?mlb_players_18 # Information about the data.
```

Now is a good time to take a look at the data. In the spreadsheet that opened in the upper left, scroll around and see what you can figure out. This is also a good time to look for any problems that the data might have, like missing values or bad variable names. Luckily, this is a nice clean dataset. Clicking on column headers will sort the data by that column. Who were the top three home run hitters in 2018?

The more you know about baseball, the more you can see in the data. This is called *domain knowledge*, and it is a key part of data science. If you lack domain knowledge, it's helpful to

acquire some - either by asking a domain expert (baseball fans in the room, please identify yourselves) or if no domain experts are available, you might try something like the following AI prompt: *I'm looking at the mlb_players_18 data set in the openintro package in R. I don't know much about the game of baseball. Please give me a 3-minute summary of the domain knowledge that would be helpful to understand this data set, and some links for further reading.*

Recall some of the different types of variable:

- Categorical
- Numeric (Discrete or Continuous)

What are the types of the variables in this data set?

A very useful command in R is `summary()`. You can ask for a summary of a variable, or a data set. To refer to a variable in a data set, you can use a `$` like this: `data$variable`.

```
summary(mlb_players_18$games)
summary(mlb_players_18)
```

Ask the data a question

As with any science, we are ideally motivated by a question. The fourth variable in the data set (number of games played) presents an opportunity. A reasonable question that these data can help us address is: Based on batting statistics, which players were underplayed or overplayed in 2018?

Establish a metric

In order to investigate our question, we have to decide how we are going to measure if a player was overplayed or underplayed. In order to do that, we'll have to figure out how to guess, based on a players batting statistics, how many games we expect them to play. Then we can just calculate the difference:

overplayed = games - expected_games

Explore the data for good predictors of games

Now we need to figure out what other variables in the data are helpful predictors of how many games a player plays. One immediate candidate is how many runs they scored - this should be highly correlated with how many games they played. Let's take a look.

ggplot

We made a scatter plot above using the `plot()` command, but the ggplot2 package is much better in many ways than the tools included in "base R" so we are going to use it for most of our visualizations. The "gg" stands for Grammar of Graphics, and the key ideas are that

- **variables** in the data are mapped to **aesthetics** in a graphic, such as position, color, and shape; and
- **aesthetics** are represented by various **geometries**, such as points, boxplots, and histograms.

The format used in the `ggplot()` function is

```
ggplot(data, aes(x, y, ...))+
  geometry
```

In our example, the data are `mlb_players_18` and we will use `R` (runs) for our x variable and `games` for our y variable, and represent each player by a point. The command to enter into R is this:

```
ggplot(mlb_players_18, aes(R, games))+
  geom_point()
```

Your reaction to this plot should be:

1. There's definitely a correlation - runs scored does correlate with games played.
2. There's definitely more to the story.

Adding in some more variables is a good way to learn more. Color and size are good aesthetics, as well as opacity (which is called `alpha` in ggplot). A domain expert told me that some good variables to look at are `OBP`, `position`, and `SLG`.

```
ggplot(mlb_players_18, aes(R, games, size=OBP, color=position, alpha = SLG))+
  geom_point()
```

You are encouraged to experiment with this graphic. Try some other variables. See if you pick up on anything else that might be part of the story we are trying to discover.

Linear regression

When you see a scatter plot like this one, you are probably tempted to compute the correlation and a regression line (also called a line of best fit). That's a great idea - here's how to do it:

```
cor(mlb_players_18$R, mlb_players_18$games)

lm1 <- lm(games~R, data=mlb_players_18) # lm is for linear model
lm1

ggplot(mlb_players_18, aes(R, games))+
  geom_point()+
  geom_smooth(method = "lm") # fits a cubic spline curve by default
```

Interpret the coefficients of the linear model `lm1` in terms of baseball. If a player scored 50 runs during the 2018 season, how many games does this model estimate that they played?

R can apply a model to data. We can create a new variable in `mlb_players_18` with the number of games predicted by `lm1`.

```
mlb_players_18$pred1 <- predict(lm1)
```

Look at the data again and you will see the new column on the far right. Now we can use the model to give an answer to our question.

```
mlb_players_18$overplayed1 <- mlb_players_18$games - mlb_players_18$pred1
# Use the sorting feature in the spreadsheet view to find the player with the
# highest value of overplayed. Or you can use the following code.
x <- which.max(mlb_players_18$overplayed1)
mlb_players_18$name[x]
```

This model hands the dubious honor to third baseman Yadiel Rivera of the Miami Marlins. Before we send him back to the Minor Leagues, perhaps we should investigate a more sophisticated model.

Filtering data

Looking at the plot you made earlier that included `position`, note that the pitchers seem to be doing something different. Ask your domain expert what is going on, and if it might make sense to leave them out of this particular model. They will probably say yes. Here's how to filter out the pitchers, and create a new data set called `batters18`. The `%>%` is called a pipe

- it feeds `mlb_players_18` into the `filter` function. These commands are from the `dplyr` package.

```
batters18 <- mlb_players_18 %>%
  filter(position!="P") # != means "is not equal to"
```

Are there other positions that we should pay attention to or treat differently? We can make a table and a boxplot:

```
table(batters18$position)

ggplot(data = batters18, aes(x = position, y = games)) +
  geom_boxplot()
```

If you see anything curious, maybe your domain expert can explain what is going on.

Now make a new model without the pitchers.

```
lm2 <- lm(games~R, data=batters18)
batters18$overplayed2 <- lm2$residuals # A shortcut.
# For any model, residual=actual-predicted.
x <- which.max(batters18$overplayed2)
batters18$name[x]
```

Poor Yadiel.

Beyond the one-variable linear function

As you have learned in your math classes, linear functions with one input are great, but they can be limited. We will be moving past $y = mx + b$ in our pursuit of useful models. Here are two more models.

```
lm3 <- lm(games~poly(R,3), data=batters18) # fit a degree 3 polynomial
batters18$overplayed3 <- lm3$residuals # residual=actual-predicted.
x <- which.max(batters18$overplayed3)
batters18$name[x]

lm4 <- lm(games~poly(R,3) + OPS + RBI, data=batters18) # include more variables
batters18$overplayed4 <- lm4$residuals # residual=actual-predicted.
x <- which.max(batters18$overplayed4)
batters18$name[x]
```

Poor Yadiel.

Assignment

Your assignment is to produce a document that uses the investigations performed in this lab, and reports on the question, "Who were the most underplayed batters in the 2018 MLB season?" Note that underplayed is just the negative of overplayed, so you don't have to do any new analysis. If you are so inclined, playing with the models is encouraged - try different variables, only look at left fielders, …get creative and it will serve you well as you develop your data skills.

- You are encouraged to work together on this, as long as the work is equitably shared. I will gladly accept lab reports with up to three names on them. Everyone should turn in their own copy of the lab report on Canvas - even if that means three identical submissions.
- Your audience is both future you and present me. I want to have an easy time looking at your report and knowing what you figured out and what you are struggling with. Future you wants a useful reference that will jog their memory when the time comes to apply this knowledge.
- You don't need to discuss the work that you did in the Overview section, unless you explored and learned something that you want either me or future you to know about.
- If you include code and output, use a `fixed width (typewriter) font` for code and output and another font for text.
- The export button in the Plots tab opens a dialog so you can save your graphics.
- If you want to use Quarto or RMarkdown, I encourage this. https://quarto.org/docs/get-started/hello/rstudio.html