STAT 113: Lab 3

RMarkdown, Correlation and Regression

Last Revised September 22, 2017

Note: Parts of this lab are modified by Colin Dawson from source material by Andrew Bray, Mine Çetinkaya-Rundel, and the UCLA statistics department which accompanies the OpenIntro statistics textbooks. This handout as well as the source material is covered by a CreativeCommons Attribution-ShareAlike 3.0 Unported license.

Lab Summary The movie *Moneyball* focuses on the "quest for the secret of success in baseball". It follows a low-budget team, the Oakland Athletics, who believed that underused statistics, such as a player's ability to get on base, better predict the ability to score runs than typical statistics like home runs, RBIs (runs batted in), and batting average. Obtaining players who excelled in these underused statistics turned out to be much more affordable for the team.

The goal of this lab is to explore various simple linear regression models to predict the number of runs scored by baseball teams in a season, using a variety of common team level measures of a team's offensive performance.

What to Turn In There will be exercises along the way do work on during lab (and finish at home if needed), but you only need to turn in written answers to the questions at the end, in the section titled "Homework". You should turn in two files: an `.Rmd` RMarkdown source file (you'll learn what this is) and a `.pdf` output file.

RMarkdown for Reproducible Research

In the first two labs we learned about script files as a way to keep track of what we have done. It is good practice to record exactly what you did so that you can reproduce it later. This extends to plots and graphs as well.

Last time, you created several plots. If we wanted to present our models and descriptions of this data, we'd like to include up-to-date plots which (a) use the most current data we have, and (b) reflect exactly any manipulations we did in the code. A script only helps us so much here: at some point we would need to cut and paste graphs into a Word document or some such writeup, and we run the risk of getting things out of sync. Also, if we have run code at the console, or read in data using a dropdown menu, or anything else not recorded in the script, someone else using the script may not get the same results because there is something left out.

A powerful solution is to use an **RMarkdown** document, where code, plots, and descriptive text can be interleaved in a file, all of which get integrated ("Knit") into a well-formated document that includes all of the above (or, you can omit the code and just show the output). Every time you make a change, just re-Knit, and the changes will be updated in the document. Moreover, the Markdown file is completely self-contained; the document will only have access to what is explicitly written down in the file, and it is "walled off" from anything you do in the rest of your RStudio session. At first this may be annoying, because you will undoubtedly get error messages arising from forgetting to include the line that reads in the data, or loads a package, or something else, in your document. But the fact that you get errors when you leave something out is a *good* thing; it forces you to record exactly what is needed to get your results.

R Markdown Workflow

Once you have done this a few times you will develop your own habits, but these are good habits to develop the first few times you use Markdown.

1. From the RStudio File menu, select New File > R Markdown.... Now you have several options. From the list on the lefthand side, select From Template, and then on the right, pick mosaic fancy to get a template that includes a bunch of examples showing you how to include various kinds of content. As you create your own documents you will want to delete much of the extra stuff in the middle (starting with the line `## Using RMarkdown` and ending before `* File creation date`), but reading this template is a good way to learn about how Markdown works.

2. The first thing you should do after creating a new Markdown document is to save the file. Do File > Save As... and give your document a name.

3. Before editing anything, press the Knit HTML button in the document toolbar (or type Ctrl-Shift-k / Cmd-Shift-k). If you are on the server or if you happen to have LaTeX installed on your computer, you could choose Knit PDF instead. If on the server you may get a message about a blocked pop-up. Allow it to show, and you will get a new window with a mix of code, text, and plots. If you prefer, you can change the setting that puts the output in a new window and have it show up in the lower right pane of RStudio instead. You will find this option under Tools > Global Options... > R Markdown in a dropdown menu that says "Show output preview in..."

4. The first thing you should edit is the Title and Author fields at the top of the document. Give the document a title such as "STAT 113: Lab 3", and put your name inside the quotes on the Author line. Re-Knit to see your change reflected in the document.

5. When you make a change, you can test that your code works as expected by first sending that chunk's code to the console by clicking the little green triangle (looks like a "play" button) in the upper right corner of the chunk. This will execute all the code in that chunk in the console. If this chunk depends on earlier ones, you may first need to click the button to the left that send all code in previous chunks to the console. If it works as expected, "Knit" again to update your output.

Notice that there is also a reference guide with lots more info on customizing formats, etc., under Help Markdown Quick Reference that will pop up in the Help tab.

> **Note** You should *not* include code in your Markdown document that you would not want to be run over and over again, or that produces output that you would not put in a writeup. This includes `install.package()` lines, lines that access documentation (e.g., `?EmployedACS`), or generate dialogue boxes.

Code Chunks

Look over the original source document. One thing that jumps out is that all of the R code in the document comes in little "chunks" with some beginning and ending markers like the following:

~~~
```{r, include = TRUE, echo = FALSE}
xyplot(births ~ date, data = Births78)
```
~~~

The triple tick marks are backquotes, which are found on the same key as the tilde (∼) on most keyboards. The letter `r` inside the curly braces indicates that the code inside is R code. The other pieces are optional and control things like whether the code itself should be displayed, or just the output, whether the code should be run or *just* displayed, etc.

Anything written *outside* these triple-quote "fences" is just interpreted as regular text. You can type whatever you want as though you were working in a word processor and it will be displayed as regular text. Unlike in an RScript, you do *not* need to precede your text with a # sign, as long as it is outside a "code chunk".

To create a new code chunk, place the cursor where you want the code to go, and click the Chunks button at the upper right of the toolbar, and select Insert Chunk (you will also see a keyboard shortcut that you could use instead if you prefer). This will create the triple-backquote "fence" that tells the computer that anything inside is code (and is interpreted as in an RScript), and anything outside is ordinary text.

You can also look over some of the text formatting tips given in the template, such as how to make text bold or italicized, how to include section headings, etc. All of that applies to text that is not in a code chunk. Inside a code chunk it is just as if you are working in a script file. There is also a reference guide with lots more info on customizing formats, etc., under Help Markdown Quick Reference that will pop up in the Help tab.

Protip You should *not* include code in your Markdown document that you would not want to be run over and over again, or that produces output that you would not put in a writeup. This includes any `install.packages()` lines, lines that access documentation (e.g., `?EmployedACS`), and any `View()` commands. Run these kinds of lines at the console.

Markdown Exercises

1. Create a new code chunk just below the first one to include `library()` lines for the packages we are using (i.e., `mosaic`, `Stat2Data`, `Lock5Data`). Re-Knit to make sure you don't get any errors.

2. Create a new code chunk and create a box plot of the `Income` variable from the `EmployedACS` dataset (in the `Lock5Data` package). (Recall the `bwplot()` function uses the form `function(~Variable, data = DataSet)`.) Press Knit. You will probably get an error saying that the dataset cannot be found. This is because with a Markdown document, unlike a script, prerequisite commands (e.g., loading data) must be in the document itself; the Knit procedure will not be able to use any data, packages, or variables that you have loaded at the console, from another file, or from buttons or menus. In other words, a Markdown document must be like a completely self-contained R session. This can be frustrating at times, but it is a good thing, because it means that your document is portable, and does not depend on the specific state of your workspace to run, so your results are reproducible exactly. Add the `data()` line loading the `EmployedACS` dataset to your code chunk above the boxplot command.

3. Modify the code chunk options (inside the curly braces, after the `r`) to add `echo = FALSE` (separate the `r` and this option with a comma. This will suppress the code itself from the output but retain results. Here are some other options you can use, for future reference:

`include = FALSE`	Suppress all code and output from the document
`echo = FALSE`	Suppress code but include output
`eval = FALSE`	Don't run the code, just display it (useful if you have some code that isn't working and you want to disable it without deleting it)
`message = FALSE`	Suppress any messages displayed when the code is run (useful for chunks that load packages)
`warning = FALSE`	Suppress warning messages that result from the code (make sure the warnings don't point to a problem first)

4. Outside the code chunk, write a sentence (it should *not* have a `#` in front if it is outside a chunk) estimating the five number summary of the `Income` variable from the boxplot.

5. In a new code chunk, calculate the actual five number summary using the `fivenum()` function (the structure is the same as the box plot function). Store the results in a named variable, e.g., `incomeStats`.

6. You can include values from a variable in regular text by surrounding the command that returns the value of interest with single backquotes, and including a lower-case `r` just inside the quotes. For example, if you typed the sentence:

   ``The mean income in the sample is `r meanIncome` `` (not in a code chunk) and you had defined a variable called `meanIncome`, when you Knit the document, the value would be substituted in. Include a sentence stating the value of the median income in the data (you can either use `median()` or get the value in the appropriate position of the five-number summary, as in `incomeStats[1]`).

Using Hitting Stats to Predict MLB Team Performance

The Data

We will use a dataset from the 2011 Major League Baseball season. In addition to runs scored (`Runs`), there are seven traditionally used variables in the data set: `AtBats`, `Hits`, `HomeRuns`, `BattingAvg`, `Strikeouts`, `StolenBases`, and `Wins`. There are also three newer variables: on base percentage (`OBP`), slugging percentage (`SLG`), and on-base plus slugging (`OPS`). For the first portion of the analysis we'll consider the seven traditional variables. At the end of the lab, you'll work with the newer variables on your own.

The data is located at `http://colindawson.net/data/mlb11.csv` Load the `mosaic` package and read the data in with `read.file()` as `MLB11`:

```r
library("mosaic")
MLB11 <- read.file("http://colindawson.net/data/mlb11.csv")
```

Exercise 1 What type of plot would you use to display the relationship between `Runs` and one of the other quantitative variables? Plot this relationship using the variable `AtBats` as the predictor. Looking at your plot, describe the relationship between these two variables. Make sure to discuss the form, direction, and strength of the relationship as well as any unusual observations. Does the relationship look linear? If you knew a team's `AtBats`, would you be comfortable using a linear model to predict the number of runs? If the relationship does look linear, quantify the strength of the relationship by computing the correlation coefficient.

Finding the Best-Fit Line

Type the following **at the console** (not in your Markdown document) to produce an interactive plot that will let you draw your own regression line and then will show you the residuals associated with it. You will need to send the `read.file()` command to the console first so the data is visible, if you haven't already.

```r
## If you are using your own installation you may want to log
## in to the server to do this section (just through Ex. 2)
## since installing the oilabs package can take awhile
library("oilabs")
plot_ss(y = Runs, x = AtBats, data = MLB11)
```

You may need to expand the plot window in the lower right of RStudio to see the whole graph.

After running this command, you'll be prompted to click two points on the plot to define a line. Once you've done that, the line you specified will be shown in black and the residuals in blue. Note that there are 30 residuals, one for each of the 30 observations. Recall that the residuals are the difference between the observed values and the values predicted by the line.

The most common way to do linear regression is to select the line that minimizes the sum of squared residuals. To visualize the squared residuals, you can rerun the plot command and add the argument `showSquares = TRUE`.

```r
plot_ss(y = Runs, x = AtBats, data = MLB11, showSquares = TRUE)
```

Note that the sum of squared residuals is displayed in the console when you choose your line.

Exercise 2 Re-run the line above a few times (at the console), selecting different lines, and see how small you can get the sum of squared residuals (SSR) to be. Write down the prediction equation for your best line (in the form $\hat{y} = \hat{a} + \hat{b}x$).

It is rather cumbersome to try to get the correct least squares line, i.e. the line that minimizes the sum of squared residuals, through trial and error. As we discussed in class, the best line is the solution to a multivariable calculus problem; but we can use the `lm()` function in R to fit the linear model (a.k.a. regression line).

We will want to use the resulting regression model later, so we'll save the result to a named R object.

```r
AtBatsModel <- lm(Runs ~ AtBats, data = MLB11)
```

You can display the model coefficients (that is, intercept and slope) by calling the `coef()` function on the model

```
coef(AtBatsModel)
```

Exercise 3 Write down the prediction equation for the best fit line (in the form $\hat{y} = \hat{a} + \hat{b}x$) found by `lm()`. Did you get close with your trial-and-error approach? Plot the best fit line over the data using `xyplot()` as we have done before (using the argument `type = c("p","r")`) to visualize it. If a team manager saw the least squares regression line and not the actual data, how many runs would they predict for a team with 5,578 at-bats?

From now on we will use `lm()` *instead* of `plot_ss()` to find the best line directly. (The `plot_ss()` was just to illustrate how different lines give different sums of squared errors.)

Exercise 4 Fit a new model that uses `HomeRuns` to predict `Runs`. Using the estimates from the R output, write the equation of the regression line. What does the slope tell us in the context of the relationship between success of a team and its home runs?

Measuring fit with R^2

Definition: R^2 One measure of how well the model fits the data is the sum of the squared residuals. Another is called the **coefficient of determination**, denoted by R^2. This is a number that ranges from 0 to 1 which indicates what proportion of the total variability in the response variable is linearly related to the explanatory variable. The proportion of variability that is *not* linearly related; that is the "random" part that's not explained by the model, is represented by the ratio between the variance of the residuals and the variance of the response variable by itself, and R^2 is 1 minus this proportion:

$$R^2 = 1 - s^2_{residuals}/s^2_y$$

It turns out that it can also be computed by squaring the correlation coefficient.

The R^2 value of a model can be found as follows:

```
rsquared(AtBatsModel)
```

Verify that we get the same thing if we look at ratios of variances:

```
s2.residuals <- residuals(AtBatsModel) %>% var()
s2.y <- var(~Runs, data = MLB11)
1 - s2.residuals / s2.y
```

Note: It is possible you will get a cryptic warning when you use the `var()` function as above. You can ignore this; it is caused by an odd interaction between R packages, but it doesn't hurt anything. In Markdown, you can suppress the warning from appearing in the output using the chunk option `warning = FALSE`

Verify that you also get the same thing if you square the correlation coefficient:

```
r <- cor(Runs ~ AtBats, data = MLB11)
r^2
```

Exercise 5 Compare the R^2 value for the `AtBats` model to the R^2 value for the `HomeRuns` model. Which explanatory variable does a better job of accounting for the number of runs scored?

Assessing Model Quality

As we have seen, not every linear model is appropriate, even if the residuals are small. We should check at least two things:

- **Linearity:** Is there a "leftover" pattern in the residuals which is associated with the explanatory variable or with the predicted values? If so, the relationship is likely not linear.
- **Approximate "Normality":** Are the residuals approximately bell-shaped ("Normally distributed")? If not, the best fit line may not be reliable, due to skew or outliers.

To check for linearity, we should plot the residuals against the explanatory or fitted values:

```
## Plotting residuals against explanatory variable
xyplot(residuals(AtBatsModel) ~ AtBats, data = MLB11, type = c("p", "r"))

## Plotting residuals against fitted values
xyplot(residuals(AtBatsModel) ~ fitted.values(AtBatsModel), type = c("p", "r"))
## Note that we can omit the data= argument in the second case, since
## the model object stores the residuals and fitted values with it
```

Do you see any pattern?

To check for Normality (bell-shapedness) we can create a histogram of the residuals, with an overlaid Normal curve:

```
histogram(~residuals(AtBatsModel), data = MLB11, fit = "normal")
```

An alternative plot we can use to assess Normality is called a `Quantile-Quantile` plot. It plots the quantiles of a theoretical Normal distribution against the actual quantiles of the residuals. If the fit is normal, the residuals should fall on a straight line. If the values are very curved or form an S-shape, that is a sign that the residual distribution is skewed, or has values that are more extreme than expected.

```
## QQ Plot
plot(AtBatsModel, which = 2)
```

Exercise 6 Produce residual diagnostic plots for the HomeRuns model you created above. Does the fit look roughly linear? Are the residuals roughly Normal?

Homework

1. Choose another "traditional" variable from MLB11 that you think might be a good predictor of Runs. Produce a scatterplot of the two variables and fit a linear model. At a glance, does there seem to be a linear relationship?
2. How does this relationship compare to the relationship between Runs and AtBats? Use the R^2 values from the two model summaries to compare. Does your variable seem to predict Runs better than AtBats? How can you tell?
3. Now that you can summarize the linear relationship between two variables, investigate the relationships between Runs and each of the other five traditional variables. Which variable best predicts Runs? Support your conclusion using the graphical and numerical methods we've discussed (for the sake of conciseness, only include output for the best variable, not all five).
4. Now examine the three "newer" variables: on-base percentage (OBP), slugging percentage (SLG) and on-base-plus-slugging (OPS). These are the statistics used by the author of *Moneyball* to predict a team's success. In general, are they more or less effective at predicting runs than the old variables? Explain using appropriate graphical and numerical evidence. Of all ten variables you've analyzed, which seems to be the best predictor of runs? Does the model using that variable satisfy the conditions of linearity and near-normality?