Stats 701 Assignment #2 Solutions

Josh Errickson

Due April 7, 2017, submitted via Canvas.

Libraries I used in my solutions:

```
library(stringr)
library(jsonlite)
library(ggplot2)
library(rvest)
library(tidyr)
library(mice)
library(testthat)
```

Instructions:

- Submit **both** your RMarkdown file **and** your output file (pdf or html, your choice). The Rmarkdown should compile and show all work.
- If you use an online resource or collaborate with classmates, provide attribution.
- If you cannot finish a question, show your work and explain/demonstrate what is causing your issue for partial credit.
- Compared to assignment 1, my solutions for these exercises are substantially longer, though both are under 100 lines. Again, only provided for a reference; if you're going far beyond that without seeing the end, come see me to clarify.

Exercise 1

On this github page, https://github.com/jhallen/joes-sandbox/tree/master/editor-perf, user jhallen ran some performance tests on various text editors, producing a number of tables. There are two "sets": the first measures how much memory (RSS) loading files with or without syntax highlighting takes; the second times performing an operation. However, it's hard to get an overview because the results are not combined (e.g. nano loads a file and jumps to the end very quickly, but is one of the slowest to perform a search and replace).

Using ggplot2, produce some graphics which combine these results to tell a fuller story across all browsers. You can graph the RSS and time results separately (though if you can come up with a clever way of combining them that doesn't produce visual overload, I'll give some extra credit for that).

Extra Credit: Some of the tests are not performed on some editors or doesn't run to completion (e.g look at the "Time to load 3 GB file" table). Include that information in your visualization.

Notes:

- There is no correct answer here.
- I was able to scrape this with rvest. However, it may be faster for you to scrape this manually with copy/paste into text or Excel. This is fine, but make sure you describe in words what you did (since there's no corresponding R code).
- It may be useful (but definitely not necessary) to work on the log scale (or log(x + 1) to ensure all values are positive).
- Ignore all comments (drop the Comments column, make comments in variables `NA`) with the exception of the few "at least X minutes" - Convert those to their max (e.g. "at least 5 minutes" -> 300 seconds).

Tips:

- The `stat` and `position` options may be useful.
- The `gather` function from the "tidyr" package may be useful for converting from wide to long data (A lot of geom's that I tested required long data). The basic syntax is:

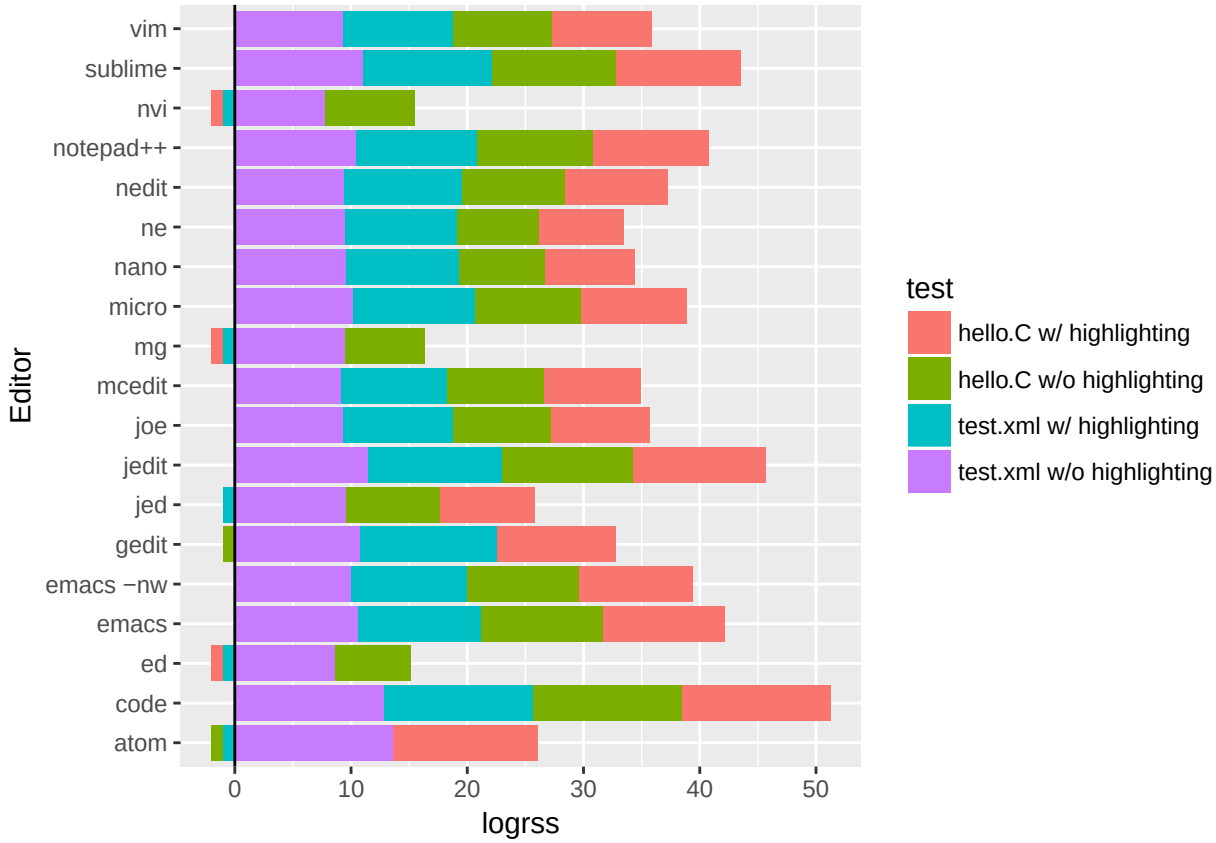
```
gather(<data>, <key colname>, <value colname>, <variables>)
```

You make up the `<key colname>` and `<value colname>`, and the list of `<variables>` is the variables to make long (all other variables are constant) and is just a vector of *unquoted* variable names.

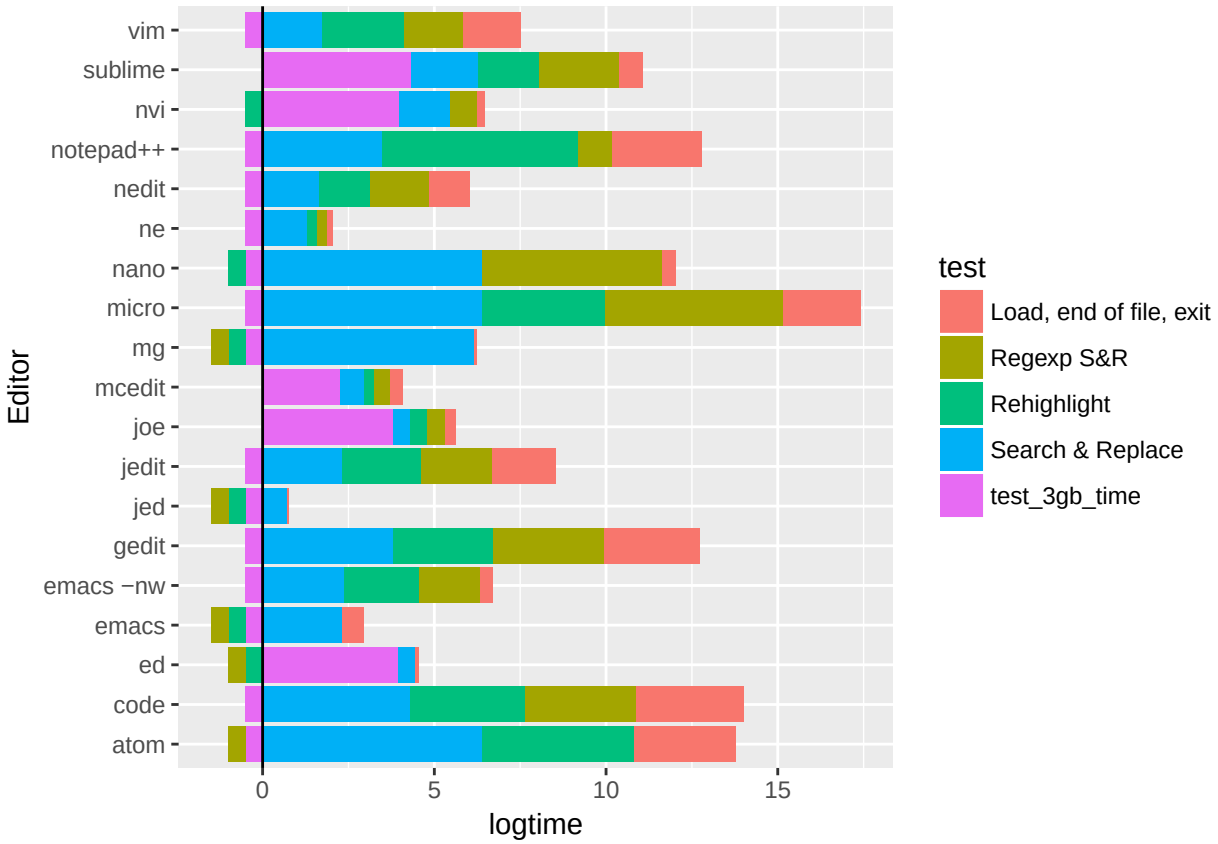
Solution

```
library(rvest)
r <- read_html("https://github.com/jhallen/joes-sandbox/tree/master/editor-perf")

tables <- sapply(3:12, function(i) {
  html_table(html_nodes(r, "table")[i])[[1]]
})
# Rename everything:
names(tables[[1]])[2] <- "hello_syntax_RSS"
names(tables[[2]])[2] <- "hello_no_syntax_RSS"
names(tables[[3]])[2] <- "test_syntax_RSS"
names(tables[[4]])[2] <- "test_no_syntax_RSS"
names(tables[[5]])[2] <- "test_eof_time"
tables[[5]] <- tables[[5]][, -3]
names(tables[[6]])[2] <- "test_rehighlight_time"
names(tables[[7]])[2] <- "test_search_time"
names(tables[[8]])[2] <- "test_regexp_time"
names(tables[[9]])[2] <- "test_3gb_time"
names(tables[[10]])[2] <- "test_paragraph_time"

a <-Reduce(function(...) merge(..., by = "Editor", all.x = TRUE, all.y = TRUE), tables)

a$test_rehighlight_time <- as.numeric(str_replace(a$test_rehighlight_time,
                                                  "^[a-zA-Z]+[:print:]*$",
                                                  "300"))
a$test_search_time <- as.numeric(str_replace(a$test_search_time,
                                             "^[a-zA-Z]+[:print:]*$",
                                             "600"))
a$test_3gb_time <- as.numeric(a$test_3gb_time)
## Warning: NAs introduced by coercion
a$test_paragraph_time <- str_replace(a$test_paragraph_time,
                                     "crashes",
                                     "NA")
a$test_paragraph_time <- as.numeric(str_replace(a$test_paragraph_time,
                                                "65\\*",
                                                "65"))
## Warning: NAs introduced by coercion

rss <- a[, 1:5]
rss <- gather(rss, test, rss, -Editor)
rss$test <- str_replace(rss$test, "hello_syntax_RSS", "hello.C w/ highlighting")
rss$test <- str_replace(rss$test, "hello_no_syntax_RSS", "hello.C w/o highlighting")
rss$test <- str_replace(rss$test, "test_syntax_RSS", "test.xml w/ highlighting")
```

```
rss$test <- str_replace(rss$test, "test_no_syntax_RSS", "test.xml w/o highlighting")
time <- a[, c(1, 6:10)]
time <- gather(time, test, time, -Editor)
time$test <- str_replace(time$test, "test_eof_time", "Load, end of file, exit")
time$test <- str_replace(time$test, "test_rehighlight_time", "Rehighlight")
time$test <- str_replace(time$test, "test_search_time", "Search & Replace")
time$test <- str_replace(time$test, "test_regexp_time", "Regexp S&R")
time$test <- str_replace(time$test, "test_paragraph_time", "Load 3Gb file")

rss$logrss <- log(rss$rss)
time$logtime <- log(time$time + 1)
rss$logrss[is.na(rss$logrss)] <- -1
time$logtime[is.na(time$logtime)] <- -.5

ggplot(rss) +
  geom_bar(aes(x = Editor, y = logrss, fill = test), stat = "identity") +
  coord_flip() + geom_hline(yintercept = 0)
```




```
ggplot(time) +
  geom_bar(aes(x = Editor, y = logtime, fill = test), stat = "identity") +
  coord_flip() + geom_hline(yintercept = 0)
```



The trick with the negative values on the barcharts incorporates missing values; the blocks of color to the left of the vertical line at 0 indicate a missing value.

Exercise 2

Grab data on all automotive facilities in Southeast Michigan from here: http://maps-semcog.opendata.arcgis.com/datasets/e71268f136dc4538abb4cffb2594277c_0 (Download Dataset on the right below the map.) This contains data on all 51 automotive plants in the area. Let's focus on EMPLOYMENT, YEARBUILT, SQUAREFEET and attempt to answer the question: Are more recent plants built more efficiently to accommodate more workers, controlling for the size of the facility.

There are probably better statistical approaches to determine this, but for our purposes, we will address this with a linear model predicting EMPLOYMENT based upon YEARBUILT and SQUAREFEET. (Take `log` transforms of both EMPLOYMENT and SQUAREFEET.)

```
lm(log(EMPLOYMENT) ~ log(SQUAREFEET) + YEARBUILT, data = ...)
```

1) Clean up this data. Restrict yourself to currently operating plants. Take special note of the COMMENTS.
2) Do you think the missing data is MCAR, MAR or MNAR? Remember, we can only answer this by considering the data collection methods.
3) Use "mice" to multiply impute these three columns. Hint: Be sure to clean the data first!
4) Report the coefficients from $2+k$ regression models:
   a) The original data.
   b) Each of your k imputations.
   c) The pooled model.

You only need do $k = 5$ iterations. Feel free to do more if you'd like, but I don't want a massive table!

Extra Credit: Create some sort of visualization for these coefficients, specifically visualizing the original and pooled estimates relative to the distribution of imputation estimates. Use $k \geq 100$ if attempting this. There are certainly many ways to accomplish this; my solution involved `facet_wrap`.

Extra Credit 2: Create a scatter plot between log(SQUAREFEET) and log(EMPLOYMENT), plotting all the least square lines. Clearly distinguish the fit from the original data and the pooled results versus the various imputations. This is a *much* trickier extra credit that probably isn't worth the time! (At least, it took me nearly an hour to solve this.)

Notes:

- When cleaning small data, its often quicker to manually make modifications, either in Excel or by directly referencing the cells, e.g. `data[4,6] <- ...`. Be sure to document this in your submission!
- Take special note of COMMENTS!
- If you make decisions how to handle bad data, be explicit about what decision you make.
- Different variables can have different types of missingness.
- I may not have said this during the MI lecture, but there are arguments both ways on whether to impute raw data or to impute transformed data. You may choose either.

Hints:

- Don't assume non-missing data is actually not missing.
- Use `print = FALSE` in your mice call to suppress that annoying output.

Hints for extra credit:

- Using `gather` may again be handy for getting the data in a format ggplot likes (then again, it might not, depending on your solution).
- To ensure certain points are "on top" of the other points, you can add a second `geom_whatever` which operates on a subsetted data. If this occurs later in the `ggplot` stack, it will plot on top.
- The `scales = "free"` argument in `facet_` may be useful (or `scales = "free_x"` for just "x", same for "y").

Solution

- EMPLOYMENT has no missing data as the two 0's are closed plants, so we drop them.
- Some of the YEARBUILT appears to be MAR, because we know that two are from leased spaces, so the YEARBUILT variable is not really imformative. The others we don't have any information about, so we can't really determine, but there's also no evidence that MAR isn't a valid assumption.
- Similarly with SQUAREFEET, the two plants with 0 footage has no evidence against MAR, so we can assume it.
- Note that we don't have evidence that either are not MNAR! So its an assumption that we are MAR, we don't have any evidence of it.

```
data <- read.csv("~/Downloads/Automotive_Facilities.csv")

# The two Trenton plants "share" employees. Let's split the employees
# over them relative to their size.
wmean <- weighted.mean(data$EMPLOYMENT[str_detect(data$NAME_2, "Trenton")],
                       data$SQUAREFEET[str_detect(data$NAME_2, "Trenton")])
data$EMPLOYMENT[str_detect(data$NAME_2, "Trenton")] <- c(wmean,
          sum(data$EMPLOYMENT[str_detect(data$NAME_2, "Trenton")]) - wmean)

# Remove the two closed plants, and make SQFT of 0 missing.
data <- data[!str_detect(data$NAME_2, "Mack[:print:]*II|Performance"),]
data$SQUAREFEET[data$SQUAREFEET == 0] <- NA
```

```r
# Drop extraneous data
data <- data[,c("EMPLOYMENT", "YEARBUILT", "SQUAREFEET")]

# original model
mod1 <- lm(log(EMPLOYMENT) ~ log(SQUAREFEET) + YEARBUILT, data = data)
coef1 <- mod1$coef

# Number of imputations:
k <- 100

mdata <- mice(data, m = k, print = FALSE)
# Imputed model
mods <- with(mdata, lm(log(EMPLOYMENT) ~ log(SQUAREFEET) + YEARBUILT))
coefs <- sapply(mods$analyses, coef)

# Pooled model
modp <- pool(mods)
coefp <- modp$qbar

allcoef <- data.frame(t(cbind(coef1, coefs, coefp)))
```

```
## Warning in data.row.names(row.names, rowsi, i): some row.names duplicated:
## 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38
## --> row.names NOT used
```

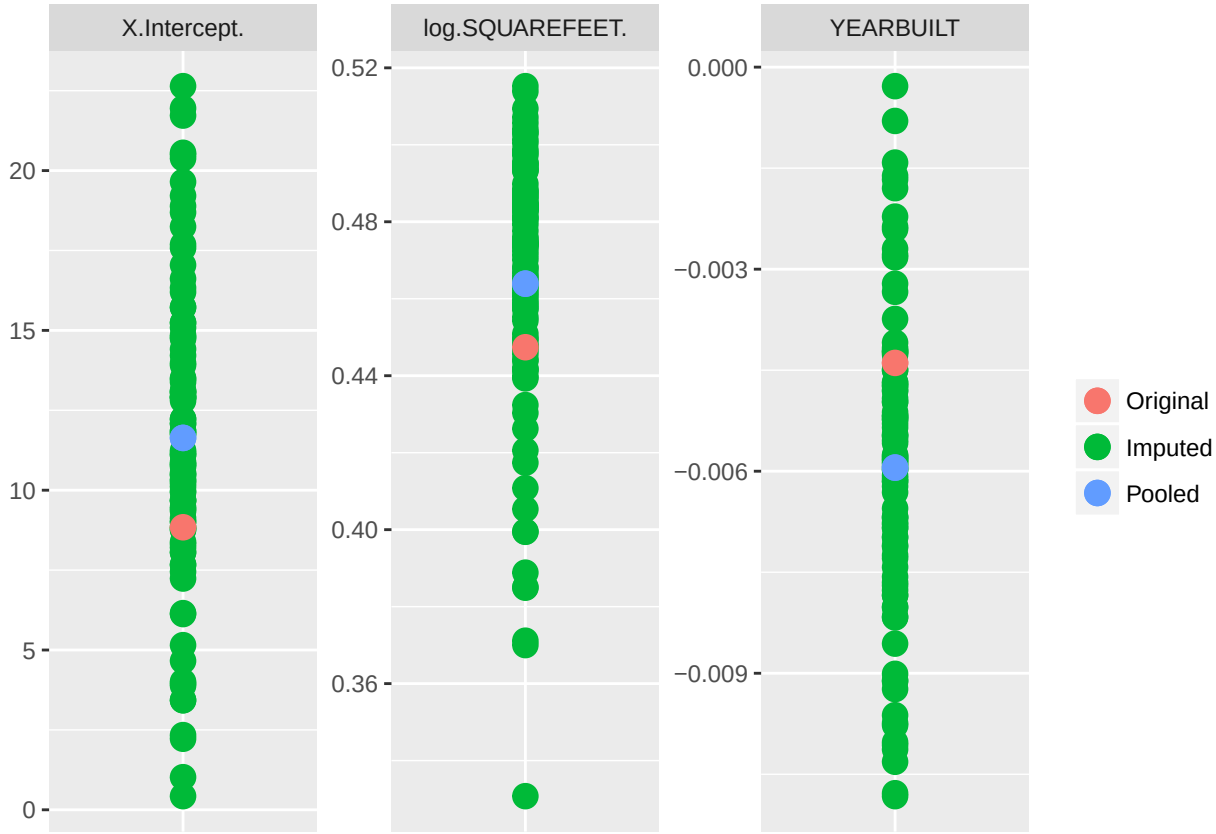
```r
allcoef$model <- c("Original", str_c("Imp ", 1:k), "Pooled")
# Just print out a few since there's 100 imputations for the EC
allcoef[c(1:6, nrow(allcoef)), ]
```

```
##     X.Intercept. log.SQUAREFEET.    YEARBUILT    model
## 1       8.839952       0.4474029 -0.004392039 Original
## 2      13.384104       0.4833568 -0.006986926    Imp 1
## 3      10.464632       0.4496139 -0.005231882    Imp 2
## 4      15.718396       0.4419889 -0.007840820    Imp 3
## 5       8.063507       0.4827702 -0.004269906    Imp 4
## 6      17.044366       0.4466546 -0.008560096    Imp 5
## 102    11.635642       0.4639234 -0.005946281   Pooled
```

Extra credit solution:

```r
# Reshape the data into a format the ggplot likes
allcoeflong <- gather(allcoef, variable, coef, 1:3)

# Change "model" and "variable" into factor to more easily control plotting order.
allcoeflong$variable <- as.factor(allcoeflong$variable)
allcoeflong$variable <- relevel(allcoeflong$variable, "X.Intercept.")

# model has imputation number saved from above, drop them.
allcoeflong$model <- str_replace(allcoeflong$model, " [0-9]+", "uted")
allcoeflong$model <- as.factor(allcoeflong$model)
allcoeflong$model <- relevel(allcoeflong$model, "Original")

# These various `theme` options aren't neccessary, it just cleans things up a bit.
ggplot(allcoeflong, aes(x = variable, y = coef, color = model)) +
  geom_point(size = 4) + facet_wrap(~ variable, scales = "free") +
```

```
  geom_point(data = allcoeflong[allcoeflong$model != "Imputed",], size = 4) +
  theme(axis.ticks.x = element_blank(),
        axis.text.x = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        legend.title = element_blank())
```

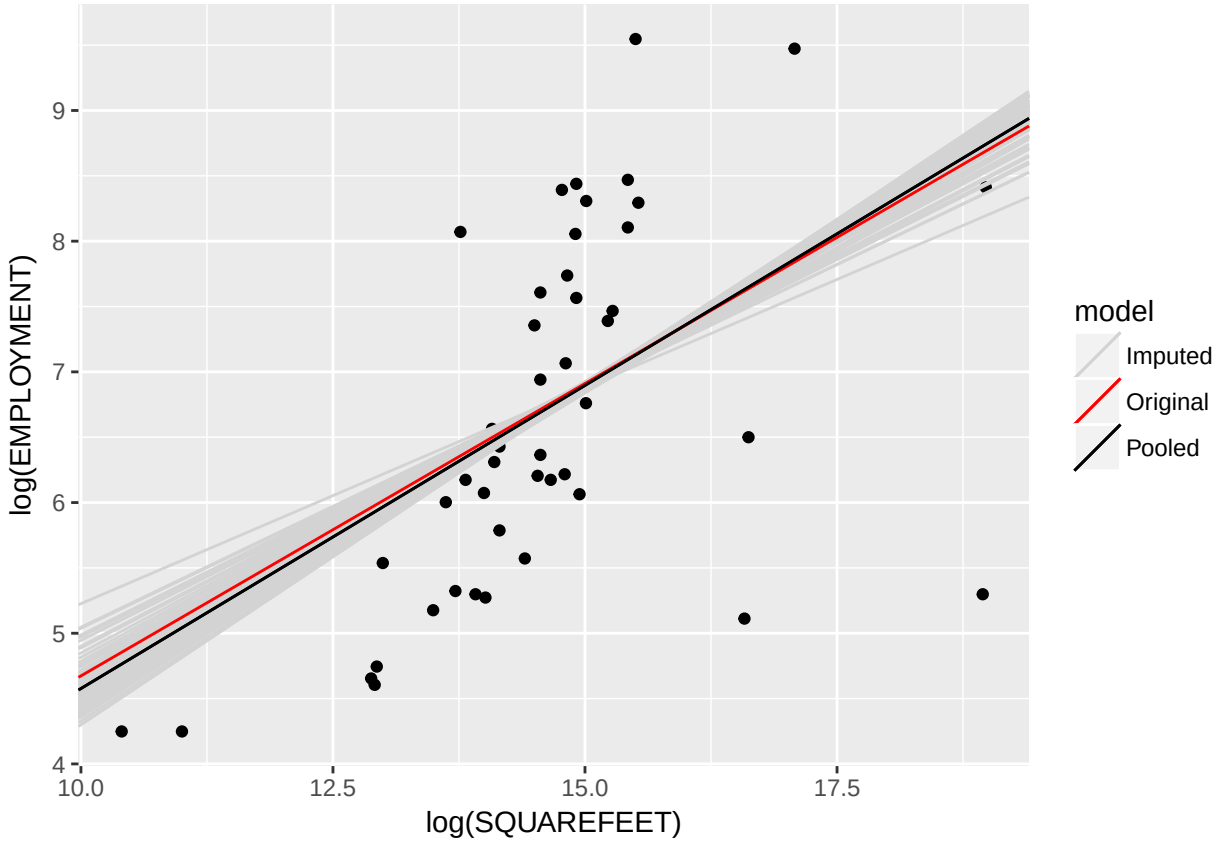


Extra credit 2:

```
# We can clean up the coefficients
twowaycoef <- allcoef
names(twowaycoef) <- c("intercept", "slope", "year", "model")
twowaycoef$model <- str_replace(twowaycoef$model, " [0-9]+", "uted")

# Let's take the intercept averaged over all years.
twowaycoef$intercept <- twowaycoef$intercept +
  mean(data$YEARBUILT, na.rm = TRUE)*twowaycoef$year

# Two geom_abline's foce the original and pooled lines on top.
# The scale_color_manual defines the colors.
ggplot(data, aes(x = log(SQUAREFEET), y = log(EMPLOYMENT))) + geom_point() +
  geom_abline(data = twowaycoef,
              aes(intercept = intercept,
                  slope = slope,
                  color = model)) +
  geom_abline(data = twowaycoef[twowaycoef$model != "Imputed",],
              aes(intercept = intercept,
                  slope = slope,
```

```
                  color = model)) +
scale_color_manual(values = c('lightgrey', 'red', 'black'))
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



Exercise 3

Use the testthat package to add unit tests to the `getMovieData` function from the first assignment. You may use your function from that assignment, or my solution if you prefer.

Important If you run into any failing tests (indicating a bug), you *are not required to fix the code*, however, do identify what the bug or oversight might be.

Notes:

- For whatever reason, placing `expect_*` directly in RMarkdown doesn't work, even passing chunk option `error = TRUE`. To get around this, wrap all your `expect_*` statements in a `test_that` call as well as passing `error = TRUE` as a chunk option. I will explain when we talk about testing inside packages the actual purpose of the `test_that` function.

````
```{r, error = TRUE}}
test_that(desc = "my tests", {
  expect_*(...)
  expect_*(...)
})
```
````

- I don't want to put any sort of limitation like "You must have at least k tests", but I would be surprised if anyone managed to provide complete code coverage with less than 20-30 tests. For reference, I wrote 45 tests, although I tend to be a bit overzealous in testing. On the other hand, I did it rapidly so I'm sure I didn't test every corner.
- In case I forget to mention it in class, as opposed to the normal programming principle of DRY (Don't Repeat Yourself) where you should put any code you'll use more than once into a function, in testing, you Do Repeat Yourself.

Solution

```
library(stringr)
library(jsonlite)
getMovieData <- function(title) {
  # Replace spaces with +
  title2 <- str_replace_all(title, "[:space:]+", "+")
  ## Vectorized:
  output <- sapply(title2, function(i) {
    url <- str_c("http://www.omdbapi.com/?t=", i)
    data <- fromJSON(url)
    # Break if a movie title isn't found
    if (data$Response == "False") {
      warning(str_c("Movie \"",
                    str_replace_all(i, "[+]+", " "),
                    "\" not found!"))
      return(c(str_replace_all(i, "[+]+", " "), NA, NA))
    }
    data$Runtime <- str_replace(data$Runtime, " min", "")
    return(c(data$Title, data$Year, data$Runtime))
  })
  output <- data.frame(t(output), stringsAsFactors = FALSE)
  # Clean up names on output.
  colnames(output) <- c("title", "year", "minutes")
  rownames(output) <- NULL
  # Convert to numeric (Note that " min" was stripped earlier.)
  output$year <- as.numeric(output$year)
  output$minutes <- as.numeric(output$minutes)
  return(output)
}

library(testthat)
test_that(desc = "getMovieData tests", {
### Check output type
expect_error(getMovieData(), "is missing")
m <- getMovieData("star wars")
expect_is(m, "data.frame")
expect_equal(nrow(m), 1)
expect_equal(ncol(m), 3)
expect_identical(names(m), c("title", "year", "minutes"))
expect_is(m$title, "character")
expect_is(m$year, "numeric")
expect_is(m$minutes, "numeric")
```

```
### Check capitalization and spacing
expect_identical(m, getMovieData("Star Wars"))
expect_identical(m, getMovieData("Star+Wars"))
expect_identical(m, getMovieData("Star%20Wars"))
expect_identical(m, getMovieData("Star   Wars"))
expect_identical(m, getMovieData("  Star Wars"))
expect_identical(m, getMovieData("Star Wars  "))
# These last two tests fail because I should have used `str_trim`.

### Check if correct data. Visiting
### http://www.omdbapi.com/?t=Star+Wars:+Episode+IV+-+A+New+Hope returns
### `{"Title":"Star Wars: Episode IV - A New Hope","Year":"1977","Rated":"PG",
###   "Released":"25 May 1977","Runtime":"121 min"`, etc
expect_true(str_detect(m$title[1], "Star Wars"))
expect_identical(m$title[1], "Star Wars: Episode IV - A New Hope")
expect_equal(m$year, 1977)
expect_equal(m$minutes, 121)

### Check useful error
expect_warning(getMovieData("Statistics the Movie"), "Statistics the Movie")
expect_warning(getMovieData("Statistics the Movie"), "not found")
expect_warning(getMovieData("Statistics  the Movie"), "Statistics  the Movie")
# Test fails because when I re-introduce the spaces, I add one regardless of original.
# Probably should save the original title for this step instead of regenerating it.

### Make sure with warning, still output something
suppressWarnings(s <- getMovieData("Statistics the Movie"))
expect_is(s, "data.frame")
expect_equal(nrow(s), 1)
expect_equal(ncol(s), 3)
expect_identical(names(s), c("title", "year", "minutes"))
expect_is(s$title, "character")
expect_is(s$year, "numeric")
expect_is(s$minutes, "numeric")
expect_identical(s$title[1], "Statistics the Movie")
expect_equal(s$year, as.numeric(NA))
expect_equal(s$minutes, as.numeric(NA))

### Vectorized results - valid movies
# (Side note: At this point I'm confident that each individual movie result is
# well tested. I simply need to check that running a vectorized version outputs
# as expected; I don't need to check the values.)
v <- getMovieData(c("Goodfellas", "Goldmember"))
expect_is(v, "data.frame")
expect_equal(nrow(v), 2)
expect_equal(ncol(v), 3)
expect_identical(names(v), c("title", "year", "minutes"))
expect_is(v$title, "character")
expect_is(v$year, "numeric")
expect_is(v$minutes, "numeric")
expect_true(v$title[1] != v$title[2])
expect_equal(sum(is.na(v)), 0)
```

```
### Make sure error in vectorized results still returns properly
expect_warning(e <- getMovieData(c("Goldmember", "Statistics the Movie")), "not found")
expect_is(e, "data.frame")
expect_equal(nrow(e), 2)
expect_true(isTRUE(all(is.na(e[2,2:3]))))
expect_true(isTRUE(all(!is.na(e[1,]))))
})
```

```
## Error: Test failed: 'getMovieData tests'
## * `m` not identical to getMovieData("  Star Wars").
## Component "title": 1 string mismatch
## Component "year": Mean relative difference: 0.0005058169
## Component "minutes": Mean relative difference: 0.1983471
##
## * `m` not identical to getMovieData("Star Wars  ").
## Component "title": 1 string mismatch
## Component "year": 'is.NA' value mismatch: 1 in current 0 in target
## Component "minutes": Mean relative difference: 0.7520661
##
## * `warnings` does not match "Statistics  the Movie".
## Actual value: "Movie "Statistics the Movie" not found!"
```

Exercise 4

Ensure your system is prepare for R package development! Install the necessary tools:

- Windows: RTools
- Mac: XCode (You may be able to just run `xcode-select --install` at the command line instead)
- Linux: install `r-base-dev` package from apt-get, etc.

Test this making sure this returns `TRUE`:

```
library(devtools)
has_devel()
```

```
## '/Library/Frameworks/R.framework/Resources/bin/R' --no-site-file  \
##   --no-environ --no-save --no-restore --quiet CMD SHLIB foo.c
```

```
##
```

```
## [1] TRUE
```