Econometrics: Multiple Equation Estimation **ARE212**: Section Syllabus

Professor	Max Auffhammer	**Office hours**	Thursdays, 5:00PM-6:00PM
GSI	Dan Hammer	**OH location**	234 GIANNINI
Section time	Fridays, 11:00AM-12:00PM	**e-mail**	danhammer@berkeley.edu
Section location	285 CORY	**twitter**	@econohammer

The objective of section is to introduce R for econometrics, illustrating the lecture notes with applied examples. Actual data work. Each section, I will present a coded example. I hope, however, that the sections will be interactive. Bring your laptop, if possible! We will work through the example and extensions together. Interrupt me with questions.

The following outline of section topics is based on the progression from previous years. The outline might change, but the final section notes will always be posted on bSpace at least one week in advance of section. The development version of the notes and R code will be posted to my Github repository:

github.com/danhammer/ARE212

You do not need to know anything about Github to productively and successfully engage in section. But you will be able to see the evolution of the notes on Github, as well as the structure and protocol associated with production code. If you want to collaborate on the notes, join Github and send me an e-mail. I will help get you started, if needed.[1]

February 1	Preliminaries and setup
February 8	Matrix operations in R
February 15	OLS regression from first principles
February 22	Goodness of fit
March 1	Hypothesis testing
March 8	Returns to education, empirical example
March 15	Efficiency of GLS
March 22	Large sample properties of OLS
April 5	Testing for heteroskedasticity
April 12	Feasible generalized least squares
April 19	Serial correlation
April 26	Instrumental variables
May 3	Spatial analysis in R

The R code for each section will be posted on both bSpace and Github, and should run on any machine and any operating system. Please feel free to e-mail me with any questions.

Office hours: I do not use my office on campus. In fact, I'm not sure where it is. Instead, I'll be in 234 Giannini from 5:00pm - 6:00pm on Thursdays.

E-mail policy: If you cannot attend regular office hours, please e-mail me with questions! I'll respond promptly over e-mail or in person within 48 hours. If I can't answer in less than a paragraph, then I'll ask you to come to office hours. If you cannot attend those office hours, please e-mail me with questions! If I

[1] A side note: I have reserved the handle `auffhammer`, just in case Max ever wants to join. I will try dearly to extract the rents associated with absolute scarcity. He claims he'll just get another handle; but this is not a credible threat, since `auffhammer` is an awesome Github handle.

can't answer in less than a paragraph, I'll ask you to come to office hours. It's turtles all the way down. I will not hold office hours on Thursday, March 21 as I will be at an out-of-town conference.

Laptops: Please bring your laptop, if possible. You will be able to follow along in the notes without a laptop, but it will be helpful to write code in real time. Note, however, that AirBears is an unsecured network, and I can easily spy on your network usage. I will *never* outwardly embarrass someone for being on Facebook. But you should be embarrassed if you are on Facebook during section. Twitter is totally fine.

Homework: Problem sets will be collected at the end of class on the specified date. Ultimately, Max will decide the deductions for late homework. I will not stray from his stated policies.

Grading: The problem sets will be graded on a 100 point scale. A face-plant fail or no-show will be assigned a 0; a tried-and-came-close will be assigned a 70; and a good-showing will be assigned a 100.

Attendance: You are not required to come. I hope the sections are helpful, but I carry no conception that the sections will be uniformly helpful. Only come if the sections are helpful to you.

Quizzes: There are no quizzes. But I scared you, didn't I?

Additional resources: There are many online, free resources to learn `R` and basic econometrics; and there even exist resources that do both at once. I have listed a few helpful resources for both writing code and scripting econometric routines.

1. **Econometrics in R**, `cran.r-project.org/doc/contrib/Farnsworth-EconometricsInR.pdf`
2. **R Style Guide**, `google-styleguide.googlecode.com/svn/trunk/google-r-style.html`
3. **Econometrics**, *Bruce Hansen*, `www.ssc.wisc.edu/~bhansen/econometrics/Econometrics.pdf`

Academic integrity: Directly from the Berkeley site: These are some basic expectations of students with regards to academic integrity: ◦ Any work submitted should be your own individual thoughts, and should not have been submitted for credit in another course unless you have prior written permission to re-use it in this course from this instructor. ◦ All assignments must use "proper attribution," meaning that you have identified the original source and extent or words or ideas that you reproduce or use in your assignment. This includes drafts and homework assignments! ◦ If you are unclear about expectations, ask your instructor or GSI. ◦ Do not collaborate or work with other students on assignments or projects unless you have been given permission or instruction to do so.

Special accommodations: If you should require any disability-related accommodations during our sections, lecture, or exams, please see me privately. You will ultimately need to procure an accommodations letter from the Disabled Students Program (`dsp.berkeley.edu`), which will be sent directly to Max.

Stata is the primary statistics package for empirical research in economics. The most powerful functions for data manipulation in Stata are `sort`, `reshape`, `collapse`, and `merge`. It seems like R is becoming more prevalent in economics, however, as the types of analysis required for empirical research become more varied, and must rely more heavily on a wider developer base. Also, the statistics department does not use Stata at all, as far as I can tell. Only R. It makes sense, then, to spend a little time learning how to replicate the `sort`, `reshape`, `collapse`, and `merge` capabilities in R.

First create a sample data frame for manipulation and print the output:

```
(X <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), x1=c(5,3,6,2), x2=c(6,5,1,4)))
```

```
  id t x1 x2
1  1 1  5  6
2  1 2  3  5
3  2 1  6  1
4  2 2  2  4
```

Sort

One clear difference between Stata and R is that, with Stata, the data are very rigid, aligned as a rectangular data frame. This structure often simplifies operations on economic data. The data in R, however, can take many forms, including lists and vectors. And there can be multiple data sets in memory simultaneously. Simple operations, then, must be more explicitly defined. Sorting data is not as simple as `sort data`, but it's not too much more complicated. Instead, we arrange the data according to a list of indices.

First, consider the indexing of a data frame, and suppose that we want to extract just the second row of X:

```
X[2, ]
```

```
  id t x1 x2
2  1 2  3  5
```

And the second through fourth rows:

```
X[2:4, ]
```

```
  id t x1 x2
2  1 2  3  5
3  2 1  6  1
4  2 2  2  4
```

And then suppose that we want to collect the second and fourth rows *and* we want to reverse the order:

```
X[c(4,2), ]
```

```
  id t x1 x2
4  2 2  2  4
2  1 2  3  5
```

Sorting is just a simple extension. If we collect the indices of an ordered variable, then we can sort the entire data frame. The `order` function facilitates this index ordering.

```
order(X$t)
```

```
[1] 1 3 2 4
```

The sorted data can be achieved by appropriately applying the ordered indices.

```
X[order(X$t), ]
```

```
  id t x1 x2
1  1 1  5  6
3  2 1  6  1
2  1 2  3  5
4  2 2  2  4
```

Reshape

The Stata reshape command has two basic options: wide and long. Suppose that a variable x is indexed along two dimensions, i and j. In wide format, each observation is uniquely identified by the i index, and the j indexed attributes are separate variables or columns. In long format, each observation is uniquely identified by a different i, j combination. All else equal, there will be more observations or rows in long format and more variables or columns in wide format. Consider, for example, the data frame `X`, and let `id` be indexed by i and `t` be indexed by j. The data frame is therefore already in long format, as each observation is identified by an individual identifier *and* and time period: individual 1 in period 1 is represented as a different observation than individual 1 in period 2. Suppose that we want the unit of observation to be only the unique IDs, converting the data frame to wide format. There are many, many ways to do this in `R`, but we will use the `reshape` command:

```
(wide <- reshape(X, idvar="id", timevar="t", direction="wide"))
```

```
  id x1.1 x2.1 x1.2 x2.2
1  1    5    6    3    5
3  2    6    1    2    4
```

Note that the arguments in the `reshape` command indicate that the natural use case is for panel data, where i indexes the unit ID variable and j indexes the time variable. Reshaping data is actually pretty fun — like playing the Creator (see Figure 1). Even for a simple data frame like `X`, there are $2^3 = 8$ different formats, since we can view the index on the `x` characteristic as yet another index. With respect to that index, then, the data frame `X` is actually in wide format. Consider the data frame where each observation represents a different combindation of the three indices, where the third is the 1 or 2 that follows the `x` variable. Instead of reshape, we can "melt" the data to long format:

```
library(reshape)
(llong <- melt(X, id=c("id","t")))
```

```
  id t variable value
1  1 1       x1     5
2  1 2       x1     3
3  2 1       x1     6
4  2 2       x1     2
5  1 1       x2     6
6  1 2       x2     5
7  2 1       x2     1
8  2 2       x2     4
```

So far, we've seen the same data frame `X` with dimension 4×4, 2×5, and 8×4 — all representing the exact same information. Each representation may be useful in a different context.

Calvin and Hobbes by Watterson
First there was nothing...
...then there was Calvin!
Calvin, the mighty god, creates the universe with pure will!
From utter nothingness comes swirling form! Life begins where once was void!
But Calvin is no kind and loving god! He's one of the old gods! He demands sacrifice!
Yes, Calvin is a god of the underworld! And the puny inhabitants of earth displease him!
The great Calvin ignores their pleas for mercy and the doomed writhe in agony!
HAVE YOU SEEN HOW ABSORBED CALVIN IS WITH THOSE TINKERTOYS? HE'S CREATING WHOLE WORLDS OVER THERE!
I'LL BET HE GROWS UP TO BE AN ARCHITECT.

Figure 1: Playing the creator with `reshape`

Collapse

Now suppose that we want to transform the data frame to reflect the average value of x_1 and x_2 across time periods $t \in \{1, 2\}$. In Stata, we'd use the `collapse` command; the comparable command, both in function and in name, within `R` is `aggregate`.

```
(aggdata <-aggregate(X, by=list(X$id), FUN=mean))
```

```
  Group.1 id   t x1  x2
1       1  1 1.5  4 5.5
2       2  2 1.5  4 2.5
```

The `Group.1` and `t` variables are redundant, here, for this very simple data set. The `aggregate` function supports much more complicated queries, however, and adding the group variables becomes useful. A common query in Stata is to collapse the data by multiple variables. The extension for `aggregate` is clear, since the `by` parameter is a list.

Merge

Consider another data frame `Y` with another characteristic for each nunit and time period.

```
(Y <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), x3=c(4,3,4,3)))
```

```
  id t x3
1  1 1  4
2  1 2  3
3  2 1  4
4  2 2  3
```

To merge `X` and `Y` into a single data frame, we use the `merge` function. This is comparable to Stata's merge command, except that in `R`, you can have two data frames stored in memory; you do not need to save one to disk. Additionally, in `R`, the data sets do not need to be sorted before the merge. Together, these enhancements save many lines of code.

```
merge(X, Y, by=c("id", "t"))
```

```
  id t x1 x2 x3
1  1 1  5  6  4
2  1 2  3  5  3
3  2 1  6  1  4
4  2 2  2  4  3
```

A convenient aspect of `merge` in `R` is that, unlike Stata, the basis variables need not be named the same. Instead of a generic `by` parameter, we can specify the merging variables with `by.x` and `by.y`, which refer to the first and second data frames, respectively.

```
names(Y) <- c("a", "b", "x3")
merge(X, Y, by.x=c("id", "t"), by.y=c("a", "b"))
```

```
  id t x1 x2 x3
1  1 1  5  6  4
2  1 2  3  5  3
3  2 1  6  1  4
4  2 2  2  4  3
```

Merges in economic data, especially panel data, are often used to attribute static characteristics to the time series. The target data may be organized by time and unit, whereas the new data frame may be at just the unit level. The two data frames need not ber perfectly aligned, as in the previous examples. Consider a new, unit-level data frame `Z` that contains static characteristics (that do not depend on time).

```
(Z <- data.frame(id=c(1,2), x4=c("yes", "no")))
```

```
  id  x4
1  1 yes
2  2  no
```

We can merge this into the panel data frame `X` using the same syntax.

```
merge(X, Z, by=c("id"))
```

```
  id t x1 x2  x4
1  1 1  5  6 yes
2  1 2  3  5 yes
3  2 1  6  1  no
4  2 2  2  4  no
```

Professor	Max Auffhammer	**Office hours**	Thursdays, 5:00PM-6:00PM
GSI	Dan Hammer	**OH location**	234 GIANNINI
Section time	Fridays, 11:00AM-12:00PM	**e-mail**	`danhammer@berkeley.edu`
Section location	285 CORY	**twitter**	`@econohammer`

The objective of section is to introduce `R` for econometrics, illustrating the lecture notes with applied examples. Actual data work. Each section, I will present a coded example. I hope, however, that the sections will be interactive. Bring your laptop, if possible! We will work through the example and extensions together. Interrupt me with questions.

The following outline of section topics is based on the progression from previous years. The outline might change, but the final section notes will always be posted on bSpace at least one week in advance of section. The development version of the notes and `R` code will be posted to my Github repository:

`github.com/danhammer/ARE212`

You do not need to know anything about Github to productively and successfully engage in section. But you will be able to see the evolution of the notes on Github, as well as the structure and protocol associated with production code. If you want to collaborate on the notes, join Github and send me an e-mail. I will help get you started, if needed.[1]

February 1	Preliminaries and setup
February 8	Matrix operations in `R`
February 15	OLS regression from first principles
February 22	Goodness of fit
March 1	Hypothesis testing
March 8	Returns to education, empirical example
March 15	Efficiency of GLS
March 22	Large sample properties of OLS
April 5	Testing for heteroskedasticity
April 12	Feasible generalized least squares
April 19	Serial correlation
April 26	Instrumental variables
May 3	Spatial analysis in `R`

The `R` code for each section will be posted on both bSpace and Github, and should run on any machine and any operating system. Please feel free to e-mail me with any questions.

Office hours: I do not use my office on campus. In fact, I'm not sure where it is. Instead, I'll be in 234 Giannini from 5:00pm - 6:00pm on Thursdays.

E-mail policy: If you cannot attend regular office hours, please e-mail me with questions! I'll respond promptly over e-mail or in person within 48 hours. If I can't answer in less than a paragraph, then I'll ask you to come to office hours. If you cannot attend those office hours, please e-mail me with questions! If I

[1] A side note: I have reserved the handle `auffhammer`, just in case Max ever wants to join. I will try dearly to extract the rents associated with absolute scarcity. He claims he'll just get another handle; but this is not a credible threat, since `auffhammer` is an awesome Github handle.

can't answer in less than a paragraph, I'll ask you to come to office hours. It's turtles all the way down. I will not hold office hours on Thursday, March 21 as I will be at an out-of-town conference.

Laptops: Please bring your laptop, if possible. You will be able to follow along in the notes without a laptop, but it will be helpful to write code in real time. Note, however, that AirBears is an unsecured network, and I can easily spy on your network usage. I will *never* outwardly embarrass someone for being on Facebook. But you should be embarrassed if you are on Facebook during section. Twitter is totally fine.

Homework: Problem sets will be collected at the end of class on the specified date. Ultimately, Max will decide the deductions for late homework. I will not stray from his stated policies.

Grading: The problem sets will be graded on a 100 point scale. A face-plant fail or no-show will be assigned a 0; a tried-and-came-close will be assigned a 70; and a good-showing will be assigned a 100.

Attendance: You are not required to come. I hope the sections are helpful, but I carry no conception that the sections will be uniformly helpful. Only come if the sections are helpful to you.

Quizzes: There are no quizzes. But I scared you, didn't I?

Additional resources: There are many online, free resources to learn `R` and basic econometrics; and there even exist resources that do both at once. I have listed a few helpful resources for both writing code and scripting econometric routines.

1. **Econometrics in R**, `cran.r-project.org/doc/contrib/Farnsworth-EconometricsInR.pdf`
2. **R Style Guide**, `google-styleguide.googlecode.com/svn/trunk/google-r-style.html`
3. **Econometrics**, *Bruce Hansen*, `www.ssc.wisc.edu/~bhansen/econometrics/Econometrics.pdf`

Academic integrity: Directly from the Berkeley site: These are some basic expectations of students with regards to academic integrity: ○ Any work submitted should be your own individual thoughts, and should not have been submitted for credit in another course unless you have prior written permission to re-use it in this course from this instructor. ○ All assignments must use "proper attribution," meaning that you have identified the original source and extent or words or ideas that you reproduce or use in your assignment. This includes drafts and homework assignments! ○ If you are unclear about expectations, ask your instructor or GSI. ○ Do not collaborate or work with other students on assignments or projects unless you have been given permission or instruction to do so.

Special accommodations: If you should require any disability-related accommodations during our sections, lecture, or exams, please see me privately. You will ultimately need to procure an accommodations letter from the Disabled Students Program (`dsp.berkeley.edu`), which will be sent directly to Max.

The objective of this section is to review the syllabus and the `R` environment. With remaining time, I will introduce the Github repository for these sections, submit basic code puzzles, and make a shameless pitch for the `ecohack.org` conference.

Download `R`: The download of `R` will vary by operating system, but it will begin here in any event:

`cran.r-project.org`

The online documentation and installer routines are comprehensive. If you are new to `R`, then it might make sense to use the Mac or Windows distribution, along with the built-in editor to write and evaluate code. For the tech-oriented, the Linux distribution is very flexible; and I'd use Emacs with the ESS package for editing. If you are interested in using the Linux distribution and are having trouble with the setup, please see me.

In order to download specific packages that are not bundled with the base distribution of `R`, such as the `foreign` package, you'll enter the following commands to install and load the package:

```
install.packages("foreign")
library(foreign)
```

Once `foreign` is loaded, you'll have access to all of its constituent functions, including `read.dta` which will convert a Stata data file into an `R` data frame.

Github repo: I will primarily use bSpace to disseminate the section notes. You don't need to know anything about Github to productively and successfully engage in this section. The final version of the notes will be posted at least one week before section as a PDF on Blackboard. That said, if you want to review an advanced, rough copy of the notes, you can browse the Github repo for this section, and the address is found in the syllabus. Github is an immensely useful collaborative coding site. You will find a full revision history of the code and notes; and if you see any problems, you can submit a patch. This will provide a gentle but useful introduction to the type of open source project that is common in the Bay Area tech industry. Many of the facilities developed for collaborative coding are incredibly valuable for joint research projects.

Linear algebra puzzles: These notes will provide a code illustration of the Linear Algebra review in Chapter 1 of the lecture notes. Don't worry if you can't solve these puzzles. Come back to them later, once we have gone over `R` code in more detail. There are many correct ways to solve these puzzles. We will go over a few solutions in section.

1. Let $\mathbf{I}_5$ be a 5×5 identity matrix. Demonstrate that $\mathbf{I}_5$ is symmetric and idempotent using simple functions in `R`.
2. Generate a 2×2 idempotent matrix $\mathbf{X}$, where $\mathbf{X}$ is not the identity matrix. Demonstrate that $\mathbf{X} = \mathbf{XX}$.
3. Generate two random variables, $\mathbf{x}$ and $\mathbf{e}$, of dimension $n = 100$ such that $\mathbf{x}, \mathbf{e} \sim N(0,1)$. Generate a random variable $\mathbf{y}$ according to the data generating process $y_i = x_i + e_i$. Show that if you regress $\mathbf{y}$ on $\mathbf{x}$ using the canned linear regression routine `lm()`, then you will get an estimate of the intercept β_0 and the coefficient on $\mathbf{x}$, β_1, such that $\beta_0 = 0$ and $\beta_1 = 1$.
4. Show that if $\lambda_1, \lambda_2, \ldots, \lambda_5$ are the eigenvectors of a 5×5 matrix $\mathbf{A}$, then $\text{tr}(\mathbf{A}) = \sum_{i=1}^{5} \lambda_i$.

This first section is meant to give a brief introduction to data manipulation in R that will support the work in future sections. It may not be exciting, but it's incredibly exciting.

Creating matrices

There are a variety of data objects in R, including numbers, vectors, matrices, strings, and dataframes. We will mainly be working with vectors and matrices, which are quick to create and manipulate in R. The `matrix` function will create a matrix, according to the supplied arguments.

```
matrix(1:6, ncol=2)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

The `ncol` option specifies the number of columns for the output matrix; and the default behavior of `matrix` is to cycle through by column. To cycle through by rows, you'll have to set the optional argument `byrow=TRUE`.

```
matrix(1:6, ncol=3, byrow=TRUE)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Suppose we wanted to check to see if the first matrix was equal to the transpose of the second. This is clearly the case — we can see that it is. But in code, it would be cumbersome to check this condition using the previous two commands. Instead, we can assign the matrices to variables for use in subsequent manipulations. The `<-` operator assigns the arbitrary object to the supplied variable:

```
A <- matrix(1:6, ncol=2)
B <- matrix(1:6, ncol=3, byrow=TRUE)
```

The `=` operator also assigns values, with a slightly different behavior; and it is common practice to use the `=` assignment for function arguments.[1] The `==` comparison operator will yield `TRUE` or `FALSE`:

```
A == t(B)
```

```
     [,1] [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
[3,] TRUE TRUE
```

Note that `t()` will return the tranpose of the supplied matrix. Each element is checked individually, and each is identical in matrix $\mathbf{A}$ and $\mathbf{B}'$. To check the truthiness of the statement that all elements are identical, we need only to employ the `all` function:

```
all(A == t(B))
```

```
[1] TRUE
```

We can get a list of all the object currently available in memory with the `ls()` function, which is useful as the assignments begin to accumulate:

```
ls()
```

Note that without assignment, the transpose of $\mathbf{B}$, or `t(B)`, is created on the fly, remaining anonymous.

[1]See the Google style sheet for a description of other standard practices in R.

Matrix operations

Matrix muliplication in R is bound to %*%, whereas scalar multiplication is bound to *. Consider the product $\mathbf{BA}$:

```
B %*% A
```

```
     [,1] [,2]
[1,]   14   32
[2,]   32   77
```

The dimensions have to line up properly for matrix multiplication to be appropriately applied, otherwise R returns an error, as is the case with the product $\mathbf{BA}'$:

```
B %*% t(A)
```

```
Error in B %*% t(A) : non-conformable arguments
```

If scalar multiplication is applied to matrices of exactly the same dimensions, then the result is element-wise multiplication. This type of operation is sometimes called the Hadamard product, denoted $\mathbf{B} \circ \mathbf{A}'$:

```
B * t(A)
```

```
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

More common, if we want to scale all elements by a factor of two, say, we just multiply a matrix by a scalar; but note that `class(2)` must be not be `matrix` but rather `numeric` so as to avoid a non-conformable error:

```
A * 2
```

```
     [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
```

```
A * matrix(2)
```

```
Error in A * matrix(2) : non-conformable arrays
```

Consider a more complicated operation, whereby each column of a matrix is multiplied element-wise by another, fixed column. We encounter this situation frequently in time series analysis to test for parameter instability. Here, each column of a particular matrix is multiplied in-place by a fixed column of residuals. Let $\mathbf{e}$ be a vector defined as an increasing sequence of length three:

```
e <- 1:3
```

Note first that the default sequence in R is a column vector, and not a row vector. We would like to `apply` a function to each column of $\mathbf{A}$, specifically a function that multiplies each column in-place by $\mathbf{e}$. We must supply a 2 to ensure that the function is applied to the second dimension (columns) of $\mathbf{A}$:

```
apply(A, 2, function(x) {x * e})
```

```
     [,1] [,2]
[1,]    1    4
[2,]    4   10
[3,]    9   18
```

The function that is applied is anonymous, but it could also be bound to a variable – just as a matrix is bound to a variable:

```
whoop <- function(x) {x * e}
apply(A, 2, whoop)
```

```
      [,1] [,2]
 [1,]    1    4
 [2,]    4   10
 [3,]    9   18
```

We will often need to define an identity matrix of dimension n, or $\mathbf{I}_n$. This is quick using `diag`:

```
I <- diag(5)
```

There are many ways to calculate the trace of $\mathbf{I}_5$. One method has been bundled into a function, called `tr()`, that is included in a packaged called `psych` which is not included in the base distribution of `R`. We will need to grab and call the library to have access to the function, installing it with the command `install.packages("psych")`. For this, you'll need an internet connection.

```
library(psych)
tr(I)
```

```
 [1] 5
```

Linear algebra puzzles

1. Let $\mathbf{X} = [1 \;\; 2 \;\; 3]$, $\mathbf{Y} = [2 \;\; 3 \;\; 4]$, and $\mathbf{Z} = [3 \;\; 4 \;\; 7]$. Define $\mathbf{W} = [\mathbf{X}' \;\; \mathbf{Y}' \;\; \mathbf{Z}']$. Calculate $\mathbf{W}^{-1}$. If you cannot take the inverse, explain why not and adjust $\mathbf{W}$ so that you *can* take the inverse. *Hint*: the `solve()` function will return the inverse of the supplied matrices.

2. Show, somehow, that $(\mathbf{X}')^{-1} = (\mathbf{X}^{-1})'$.

3. Generate a 3×3 matrix $\mathbf{X}$, where each element is drawn from a standard normal distribution. Let $\mathbf{A} = \mathbf{I}_3 - \frac{1}{n}\iota\iota'$ be a demeaning matrix, with ι a 3×1 vector of ones. First show that $\mathbf{A}$ is idempotent and symmetric. Next show that each row of the matrix $\mathbf{XA}$ is the deviation of each row and $\mathbf{X}$ from its mean. Finally, show that $(\mathbf{XA})(\mathbf{XA})' = \mathbf{XAX}'$, first through algebra and then `R` code.

4. Demonstrate from random matrices that $(\mathbf{XYZ})^{-1} = \mathbf{Z}^{-1}\mathbf{Y}^{-1}\mathbf{X}^{-1}$.

5. Let $\mathbf{X}$ and $\mathbf{Y}$ be square 20×20 matrices. Show that $tr(\mathbf{X} + \mathbf{Y}) = tr(\mathbf{X}) + tr(\mathbf{Y})$.

6. Generate a diagonal matrix $\mathbf{X}$, where each element on the diagonal is drawn from $U[10, 20]$. Show that the decomposition of $\mathbf{X} = \sqrt{\mathbf{X}}\sqrt{\mathbf{X}}$. That is, calculate the Cholesky decomposition, for example, of $\mathbf{X}$ through the `chol()` function and multiply it by itself to return $\mathbf{X}$.

7. Demonstrate that for any scalar c and any square matrix $\mathbf{X}$ of dimension n that $\det(c\mathbf{X}) = c^n \det(\mathbf{X})$.

8. Demonstrate that for an $m \times m$ matrix $\mathbf{A}$ and a $p \times p$ matrix $\mathbf{B}$ that $\det(\mathbf{A} \otimes \mathbf{B}) = \det(\mathbf{A})^p \det(\mathbf{B})^m$. *Hint*: Note that $\otimes$ indicates the Kronecker product. Google the appropriate `R` function.

This section is intended to introduce linear regression in R. The first step is to read the data set auto.csv from the relative path. You can also download it from here. We set the option header to TRUE, which treats the first line of the CSV file as variable names, rather than an observation.

```
data <- read.csv("../data/auto.csv", header=TRUE)
```

We can read the names from the data set; but they aren't much help.

```
names(data)
```

```
 [1] "V1" "V2" "V3"
```

We can replace the column headers with more descriptive variable names. The next command is an example of destructuring, sort of, in a language where destructuring does not really exist. There are three elements in both the original and new list; and each element in the original list is reassigned based on its position in the list, e.g., the V1 header is replaced by price.

```
names(data) <- c("price", "mpg", "weight")
```

To get a sense of the data, list the first six observations:

```
head(data)
```

```
   price mpg weight
 1  4099  22   2930
 2  4749  17   3350
 3  3799  22   2640
 4  4816  20   3250
 5  7827  15   4080
 6  5788  18   3670
```

With the columns appropriately named, we can refer to particular variables within the data set using the unique indexing in R, where data objects tend to be variants of lists and nested lists.

```
head(data$mpg)
```

```
 [1] 22 17 22 20 15 18
```

Now for the analysis, a linear model using lm(). We specify the model by referring to the column headers within the specified data set — along with a 1 to indicate a column of ones. R is unique for intelligently snapping the dimensions of comparable matrices.

```
lm(price ~ 1 + mpg + weight, data=data)
```

```

 Call:
 lm(formula = price ~ 1 + mpg + weight, data = data)

 Coefficients:
 (Intercept)          mpg       weight
    1946.069      -49.512        1.747
```

Note that we do not need to refer to the data set in calling a vector, e.g., data$price. Instead, we "attach" the data within the linear model. We can do the same thing for the R session, which tends to simplify notation. The columns can be referenced directly after the data set has been attached.

```
attach(data)
head(mpg)
```

```
 [1] 22 17 22 20 15 18
```

The use of canned routines is not permitted for most of this class; you'll have to write the econometric routines from first principles. First, create matrices of the data, since we will be working mainly with matrix operations. Let $\mathbf{y}$ be the dependent variable, price, and let $\mathbf{X}$ be a matrix of the other car characteristics, along with a column of ones prepended. The `cbind()` function binds the columns horizontally and coerces the `matrix` class.

```
y <- matrix(price)
X <- cbind(1, mpg, weight)
```

Check that the number of observations are the same in both the dependent vector $\mathbf{y}$ and the cofactor matrix $\mathbf{X}$.

```
dim(X)[1] == nrow(y)
```

```
 [1] TRUE
```

Before proceeding, we should `detach` the data frame, so that if we reload the `R` script, there won't be any naming conflicts. The `X` and `y` objects will persist, as will `data$mpg`, but we will no longer be able to call the variables without referencing the data frame.

```
detach(data)
```

Using the matrix operations described in the previous section, we can quickly estimate the ordinary least squared parameter vector.

```
beta <- solve(t(X) %*% X) %*% t(X) %*% y
print(beta)
```

```
              [,1]
       1946.068668
 mpg    -49.512221
 weight   1.746559
```

This vector matches the coefficient vector from the canned routine, thankfully. Digging deeper into the numbers, consider the projection matrix $\mathbf{P} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$ and the residual maker matrix $\mathbf{M} = \mathbf{I}_n - \mathbf{P}$

```
n <- nrow(y)
P <- X %*% solve(t(X) %*% X) %*% t(X)
M <- diag(n) - P
```

`R` is useful for checking the properties of these matrices, including whether $\mathbf{M}$ is symmetric, that is, whether $\mathbf{M} = \mathbf{M}'$. The function `all.equal()` does not test **exact** equality, but instead whether the supplied objects are "close enough" to be considered the same. The problem is the limits of machine precision, and rounding at the tail ends of floating point numbers.

```
all.equal(M, t(M))
```

```
 [1] TRUE
```

If we want to test for exact equality, we set the tolerance to zero, and the function will return a message with the mean relative difference between elements — which is clearly very close to zero.

```
all.equal(M, t(M), tol=0)
```

```
[1] "Mean relative difference: 7.266263e-15"
```

The residual maker matrix should also be idempotent, or $\mathbf{M} = \mathbf{M}\mathbf{M}'$.

```
all.equal(M, M %*% t(M))
```

```
[1] TRUE
```

Finally, we can examine the different components of the variation in the dependent variable, as they relate to the OLS estimate. Specifically, we can show that the total sum of square is equal to the sum of the residual and estimated sum of squares:

$$\mathbf{y}'\mathbf{y} = \hat{\mathbf{y}}'\hat{\mathbf{y}} + \mathbf{e}'\mathbf{e} \tag{1}$$

First, define the relevant variables:

```
e <- M %*% y
y.hat <- P %*% y
rss <- t(e) %*% e
ess <- t(y.hat) %*% y.hat
tss <- t(y) %*% y
```

Then check the condition in Eq. (1):

```
all.equal(tss, ess + rss)
```

```
[1] TRUE
```

Additional puzzles

1. Write a function `wt.coef()` that will return the OLS coefficient on weight from the regression of car price on the covariate matrix described above.
2. Adjust the function to return a list of coefficients from the same linear regression, appropriately named.
3. Find the estimate of the covariance matrix $\sigma^2(\mathbf{X}'\mathbf{X})^{-1}$ and show that the residuals and covariate matrix are orthogonal.
4. **Partitioned regression**: Generate a 100×5 matrix $\mathbf{X}$ *including* a column of ones for the intercept. Additionally, generate a vector $\mathbf{y}$ according to the generating process:

$$y_i = 1 + x_{1i} + 2x_{2i} + 3x_{3i} + 4x_{4i} + \epsilon_i,$$

where $\epsilon_i \sim N(0,1)$. Let $\mathbf{Q}$ be the first three columns of $\mathbf{X}$ and let $\mathbf{N}$ be the final two columns. In addition, let

$$\begin{aligned}\hat{\gamma}_1 &= (\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'\mathbf{y} \quad \text{and} \quad \mathbf{f} = \mathbf{y} - \mathbf{Q}\hat{\gamma}_1 \\ \hat{\gamma}_2 &= (\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'\mathbf{N} \quad \text{and} \quad \mathbf{g} = \mathbf{N} - \mathbf{Q}\hat{\gamma}_2 \\ \hat{\gamma}_3 &= \mathbf{f}\cdot\mathbf{g}/||\mathbf{g}||^2 \quad \text{and} \quad \mathbf{e} = \mathbf{f} - \mathbf{g}\hat{\gamma}_3\end{aligned}$$

Show that $\hat{\beta} = [\hat{\gamma}_1 - \hat{\gamma}_2\hat{\gamma}_3 \quad \hat{\gamma}_3]$. Note that the total dimension of $\hat{\beta}$ is 5.

$$\max \int_0^T \left[p \cdot f(x_{1t}, x_{2t}) - c_1(R_1)x_{1t} - c_2 x_{2t} - k \cdot y_t\right] e^{-\delta t}\, dt$$

subject to

$$\begin{aligned}\dot{R} &= g(x_{2t}) - x_{1t} \\ x_{2,t+1} &= y_t\end{aligned}$$

The objective of this section is to (1) study the behavior of the centered R^2 as cofactors are incrementally included in the regression, and (2) use `R` to check the behavior of Problem 2 in the optional section of the first problem set.

Centered R^2

First, we create a random matrix, where each element is drawn from a standard uniform distribution — another context to practice the `function()` structure. The function `randomMat()` generates a long vector of length $n \cdot k$ and then reshapes it into an $n \times k$ matrix.

```
randomMat <- function(n, k) {
  v <- runif(n*k)
  matrix(v, nrow=n, ncol=k)
}
```

The function bound to `randomMat` behaves as we would expect:

```
randomMat(3,2)
```

```
          [,1]      [,2]
 [1,] 0.4688696 0.8325487
 [2,] 0.3989330 0.4329744
 [3,] 0.3218055 0.2732040
```

Another useful function for this section will be to create a square demeaning matrix $\mathbf{A}$ of dimension n. The following function just wraps a few algebraic maneuvers, so that subsequent code is easier to read.

```
demeanMat <- function(n) {
  ones <- rep(1, n)
  diag(n) - (1/n) * ones %*% t(ones)
}
```

As is described in the notes, pre-multiplying a matrix $\mathbf{B}$ by $\mathbf{A}$ will result in a matrix $\mathbf{C} = \mathbf{AB}$ of deviations from the column means of $\mathbf{B}$. Check that this is true. This may seem like a roundabout way to check the equivalence of the matrices; but it provides the opportunity to practice the `apply` function.

```
A <- demeanMat(3)
B <- matrix(1:9, nrow=3)
col.means <- apply(B, 2, mean)
C <- apply(B, 1, function(x) {x - col.means})
all.equal(A %*% B, t(C))
```

```
 [1] TRUE
```

Alright, we're ready to apply the functions to real data in order to calculate the centered R^2. First, read in the data to conform to equation (2.37) on page 14 of the lecture notes, and identify the number of observations n for later use:

```
data <- read.csv("../data/auto.csv", header=TRUE)
names(data) <- c("price", "mpg", "weight")
y <- matrix(data$price)
X2 <- cbind(data$mpg, data$weight)
n <- nrow(X2)
```

The centered R^2 is defined according to equation (2.41) as follows:

$$R^2 = \frac{\mathbf{b}_2' \mathbf{X}_2^{*\prime} \mathbf{X}_2^* \mathbf{b}_2}{\mathbf{y}^{*\prime} \mathbf{y}^*}, \tag{1}$$

where $\mathbf{y}^* = \mathbf{A}\mathbf{y}$, $\mathbf{X}_2^* = \mathbf{A}\mathbf{X}_2$, and $\mathbf{b}_2 = (\mathbf{X}_2^{*\prime}\mathbf{X}_2^*)^{-1}\mathbf{X}_2^{*\prime}\mathbf{y}^*$. Noting that $\mathbf{A}$ is both symmetric and idempotent, we can rewrite Equation (1) in terms of matrices already defined, thereby simplifying the subsequent code dramatically. From my limited experience with programming, the best code is that which reflects the core idea of the procedure; more time spent with a pen and paper and not in `R` will almost always yield more readable code, and more readable code yields fewer errors and suggests quick extensions. That said, note that $\mathbf{X}_2^{*\prime}\mathbf{X}_2^* = \mathbf{X}_2'\mathbf{A}'\mathbf{A}\mathbf{X}_2 = \mathbf{X}_2'\mathbf{A}\mathbf{A}\mathbf{X}_2 = \mathbf{X}_2'\mathbf{A}\mathbf{X}_2$ and similarly that $\mathbf{y}^{*\prime}\mathbf{y}^* = \mathbf{y}'\mathbf{A}\mathbf{y}$ and $\mathbf{X}_2^{*\prime}\mathbf{y}^* = \mathbf{X}_2'\mathbf{A}\mathbf{y}$. If we write a more general function, though, we can apply it to an arbitrary dependent vector and associated cofactor matrix:

```r
R.squared <- function(y, X) {
  n <- nrow(X)
  A <- demeanMat(n)
  xtax <- t(X) %*% A %*% X
  ytay <- t(y) %*% A %*% y
  b2 <- solve(xtax) %*% t(X) %*% A %*% y
  return(t(b2) %*% xtax %*% b2 / ytay)
}

R.squared(y, X2)
```

```
          [,1]
 [1,] 0.2933891
```

Without some penalty for addtional cofactors, the centered R^2 will monotonically increase with the number of columns in the cofactor matrix $\mathbf{X}$. This function is plotted in Figure (1), mostly as an introduction to very simple plots in `R`.

```r
n <- nrow(X2); k.max <- 70
X.rnd <- randomMat(n, k.max)
res <- rep(0, k.max)

for (i in 1:70) {
  X.ext <- cbind(X2, X.rnd[, seq(i)])
  res[i] <- R.squared(y, X.ext)
}

plot(res, type = "l", lwd = 3, col = "blue",
     xlab = "num. of additional columns",
     ylab = "R-squared value")
```

It may be difficult to get a sense of the shape of the curve based on a single draw for the random matrix. We can calculate the relationship between R^2 and the number of cofactors — or we can bootstrap an estimate for each index, which we will do in a subsequent section to illustrate bootstrapping in `R`.

Sum of squared residuals

Suppose that $\mathbf{b}$ is the 2×1 least squared coefficient vector in the regression of $\mathbf{y}$ on $\mathbf{X}_2$. Suppose that $\mathbf{c}$ is some other 2×1 vector. We are asked to show that

$$(\mathbf{y} - \mathbf{X}\mathbf{c})'(\mathbf{y} - \mathbf{X}\mathbf{c}) - (\mathbf{y} - \mathbf{X}\mathbf{b})'(\mathbf{y} - \mathbf{X}\mathbf{b}) = (\mathbf{c} - \mathbf{b})'\mathbf{X}'\mathbf{X}(\mathbf{c} - \mathbf{b}) \tag{2}$$

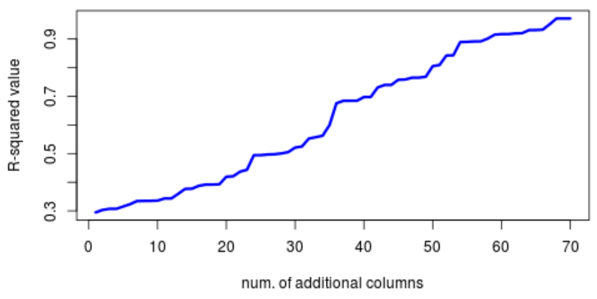

Figure 1: R^2 rises monotonically as a function of columns

We could prove this with matrix algebra. In fact, we are asked to prove this fact with matrix algebra in the problem set. But matrix algebra is for chumps — or very smart and kind people. Let's check the equality using `R` by choosing any arbitrary **c**.

```
b <- solve(t(X2) %*% X2) %*% t(X2) %*% y
c <- c(-3, 5)
```

For simplicity of notation, define **M** and **N** to be the following:

```
M <- y - X2 %*% b
N <- y - X2 %*% c
```

Now, we can check both sides of the equality:

```
lhs <- floor(t(N) %*% N - t(M) %*% M)
rhs <- floor(t(c - b) %*% t(X2) %*% X2 %*% (c - b))
all(lhs == rhs)
```

```
[1] TRUE
```

Note, however, that the order of **c** and **b** doesn't matter:

```
rhs.alt <- floor(t(b - c) %*% t(X2) %*% X2 %*% (b - c))
all(lhs == rhs.alt)
```

```
[1] TRUE
```

This result is because we are effectively looking at the sum of the squared difference between the two vectors. The ordering in the difference calculation doesn't matter if it is subsequently squared. Consider the property $\mathbb{V}(\mathbf{cX}) = \mathbf{c}\mathbb{V}(\mathbf{X})\mathbf{c}'$ for a vector **c** or $\mathbb{V}(a\mathbf{X}) = a^2\mathbb{V}(\mathbf{X})$ for a scalar a. The right-hand side of Equation (**??**) is effectively a nested, squared matrix, which has to yield positive entries (and a postive scalar if the result is 1×1):

```
G <- X2 %*% (b - c)
t(G) %*% G
```

```
           [,1]
[1,] 6209641706
```

Additional puzzles

1. **Partitioned regression**: Generate a 100×4 matrix $\mathbf{X}$ *including* a column of ones for the intercept. Additionally, generate a vector $\mathbf{y}$ according to the generating process:

$$y_i = 1 + x_{1i} + 2x_{2i} + 3x_{3i} + \epsilon_i,$$

where $\epsilon_i \sim N(0,1)$. Let $\mathbf{Q}$ be the first three columns of $\mathbf{X}$ and let $\mathbf{N}$ be the final column. In addition, let

$$\begin{aligned}
\hat{\gamma}_1 &= (\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'\mathbf{y} \quad \text{and} \quad \mathbf{f} = \mathbf{y} - \mathbf{Q}\hat{\gamma}_1 \\
\hat{\gamma}_2 &= (\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'\mathbf{N} \quad \text{and} \quad \mathbf{g} = \mathbf{N} - \mathbf{Q}\hat{\gamma}_2 \\
\hat{\gamma}_3 &= \mathbf{f} \cdot \mathbf{g}/||\mathbf{g}||^2 \quad \text{and} \quad \mathbf{e} = \mathbf{f} - \mathbf{g}\hat{\gamma}_3
\end{aligned}$$

Show that $\hat{\beta} = [\hat{\gamma}_1 - \hat{\gamma}_2\hat{\gamma}_3 \quad \hat{\gamma}_3]$. Note that the total dimension of $\hat{\beta}$ is 4.

Answer:

```
X <- cbind(1, randomMat(100, 3))
e <- rnorm(100)

beta <- c(1, 1, 2, 3)
y <- X %*% beta + e

Q <- X[, 1:3]
N <- X[, 4]
gamma.1 <- solve(t(Q) %*% Q) %*% t(Q) %*% y
gamma.2 <- solve(t(Q) %*% Q) %*% t(Q) %*% N
f <- y - Q %*% gamma.1
g <- N - Q %*% gamma.2
gamma.3 <- as.numeric(crossprod(f,g)/crossprod(g,g))
e <- f - g * gamma.3

(b <- c(gamma.1 - gamma.2 * gamma.3, gamma.3))
```

```
 [1] 1.2704760 0.8533298 1.6991382 3.0427751
```

This is an introducton to basic hypothesis testing in `R`. We have shown that, with a certain set of assumptions, the difference between the OLS estimator and the true parameter vector is distributed normally as shown in expression (2.63):

$$(\mathbf{b} - \beta)|\mathbf{X} \sim N(\mathbf{0},\ \sigma^2 \cdot (\mathbf{X}'\mathbf{X})^{-1})$$

We have also shown that $s^2 = \mathbf{e}'\mathbf{e}/(n-k)$ is an unbiased estimator of σ^2 in Section 2.3.4 of the lecture notes. The purpose of the section is not to rehash the lectures, but instead to use the results to practice indexing in `R`.

```
data <- read.csv("../data/auto.csv", header=TRUE)
names(data) <- c("price", "mpg", "weight")
y <- matrix(data$price)
X <- cbind(1, data$mpg, data$weight)
```

For reference, consider the regression output, using data we've seen before:

```
res <- lm(price ~ 1 + mpg + weight, data = data)
summary(res)
```

```

Call:
lm(formula = price ~ 1 + mpg + weight, data = data)

Residuals:
   Min     1Q Median     3Q    Max 
 -3332  -1858   -504   1256   7507 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)   
(Intercept) 1946.0687  3597.0496   0.541  0.59019   
mpg          -49.5122    86.1560  -0.575  0.56732   
weight         1.7466     0.6414   2.723  0.00813 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2514 on 71 degrees of freedom
Multiple R-squared: 0.2934,   Adjusted R-squared: 0.2735 
F-statistic: 14.74 on 2 and 71 DF,  p-value: 4.425e-06
```

We print the full results to examine the F-statistic. Under normal circumstances, consider just printing `coef(summary(res))`. In order to perform individual t-tests, we will first have to identify the standard errors for each coefficient, noting the distribution in (2.63). The variance of the error, σ^2, can be numerically estimated, as shown below:

```
n <- nrow(X); k <- ncol(X)
P <- X %*% solve(t(X) %*% X) %*% t(X)
e <- (diag(n) - P) %*% y
s2 <- t(e) %*% e / (n - k)
print(s2)
```

```
        [,1]
 [1,] 6320340
```

The vector of standard errors matches those reported from `R`'s canned routine `lm()`, which is encouraging.

```
vcov.mat <- as.numeric(s2) * solve(t(X) %*% X)
se <- sqrt(diag(vcov.mat))
print(se)
```

```
[1] 3597.0495988   86.1560389    0.6413538
```

We can now use the vector of standard errors to perform the individual t-tests.

```
b <- solve(t(X) %*% X) %*% t(X) %*% y
apply(b / se, 1, function(t) {2 * pt(-abs(t), df = (n - k))})
```

```
[1] 0.590188628 0.567323727 0.008129813
```

Great! We have replicated the `Pr(>|t|)` column of the canned output. Now let's try to replicate the full regression F-statistic. This is a joint test of coefficient significance; are the coefficients jointly different from a zero vector? Max has a great description as to why this is different from three separate tests of significance. For now, note that we are testing joint significance by setting

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{r} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

This is great. This simplifies the hell out of equation (2.81), which is fairly daunting at first:

$$F = \frac{(\mathbf{Rb}-\mathbf{r})'[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}']^{-1}(\mathbf{Rb}-\mathbf{r})/J}{s^2} = \frac{\mathbf{b}'(\mathbf{X}'\mathbf{X})\mathbf{b}/J}{s^2} \tag{2}$$

```
F <- t(b) %*% (t(X) %*% X) %*% b / (s2*3)
print(F)
```

```
         [,1]
[1,] 158.1714
```

Well shit. This is much larger than the reported F-statistic of 14.74. What happened? The problem is that we also included the intercept, whereas `R` assumes that this shouldn't be included in the joint test. Simplification failed. Let's try again.

```
R <- rbind(c(0, 1, 0), c(0, 0, 1)); J <- 2
select.var <- solve(R %*% solve(t(X) %*% X) %*% t(R))
F <- t(R %*% b) %*% select.var %*% (R %*% b) / (s2 * J)
print(c(F, pf(F, 2, 71, lower.tail=FALSE)))
```

```
[1] 1.473982e+01 4.424878e-06
```

It worked! And the probability of observing the F-statistic with degrees of freedom $J = 2$ and $n - k = 71$ is printed as well.

Puzzle (sort of)

We will replicate the results of a sort of silly study that examines the causes of McCarthyism, using a *path model*. The study was published in the *American Political Science Review* by Gibson (1988) and reprinted in Freedman (2009) to illustrate path models, which are essentially a simple graphical framework to keep track of direct and indirect causation. The path model is recreated in Figure (1), and shows that tolerance scores of both the masses and elites directly impact repression. This is a theoretical framework. The unlabeled connection between the tolerance nodes indicates association rather than causation. The repression and tolerance scores have been standardized, so that they have mean equal to zero and standard deviation equal to one.

The purpose of this exercise is to build up an intuition of the relationship between the OLS estimates and covariate correlations.
Gibson reports the correlation matrix for the path diagram:

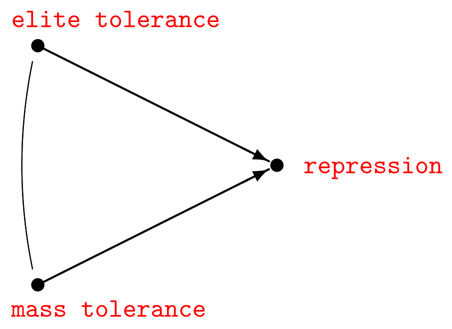

Figure 1: Path model, causes of McCarthyism, reproduced from Freedman (2009)

	Masses	Elite	Repress
Masses	1.00	0.52	-0.26
Elite	0.52	1.00	-0.42
Repress	-0.26	-0.42	1.00

His model for political repression for $n = 36$ states is given by:

$$\texttt{repression} = \beta_1 \cdot \texttt{mass tolerance} + \beta_2 \cdot \texttt{elite tolerance} + \epsilon, \tag{3}$$

Denote `mass tolerance` as M, `elite tolerance` as E, and `repression` as R, such that Equation (3) becomes $R = \beta_1 M + \beta_2 E + \epsilon$. Finally, let $\mathbf{X} = [M \quad E]$, so that $R = \mathbf{X}\beta + \epsilon$.

Here is the kicker. Since, all variables were standardized, we know that

$$\frac{1}{n}\sum_{i=1}^{n} E_i = 0 \qquad \text{and} \qquad \frac{1}{n}\sum_{i=1}^{n} E_i^2 = 1$$

This is true, also, for M and R. Being careful about matrix multiplication, we can compute the following:

$$\mathbf{X}'\mathbf{X} = \begin{bmatrix} \sum_{i=1}^n M_i^2 & \sum_{i=1}^n M_i E_i \\ \sum_{i=1}^n M_i E_i & \sum_{i=1}^n E_i^2 \end{bmatrix} = n \begin{bmatrix} 1 & r_{ME} \\ r_{ME} & 1 \end{bmatrix} = n \begin{bmatrix} 1 & 0.52 \\ 0.52 & 1 \end{bmatrix} \tag{4}$$

$$\mathbf{X}'R = \begin{bmatrix} \sum_{i=1}^n M_i R_i \\ \sum_{i=1}^n E_i R_i \end{bmatrix} = n \begin{bmatrix} r_{MR} \\ r_{ER} \end{bmatrix} = n \begin{bmatrix} -0.26 \\ -0.42 \end{bmatrix} \tag{5}$$

We don't even need the actual data to compute the OLS coefficients! Specifically, $\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'R$:

```
n <- 36
xtx <- n * matrix(c(1, 0.52, 0.52, 1), ncol = 2)
xtr <- n * matrix(c(-0.26, -0.42), ncol = 1)
(b <- solve(xtx) %*% xtr)
```

```
            [,1]
[1,] -0.05701754
[2,] -0.39035088
```

Given standardized tolerance and repression scores, we can use the following formula from page 85 in Freedman to calculate the model variance: $\hat{\sigma}^2 = 1 - \hat{\beta}_1^2 - \hat{\beta}_2^2 - 2\hat{\beta}_1\hat{\beta}_2 r_{ME}$

```
p <- 3
(sigma.hat.sq <- (n / (n - p)) * (1 - b[1]^2 - b[2]^2 - 2 * b[1] * b[2] * 0.52))
```

```
 [1] 0.8958852
```

There is an implicit intercept, since the scores are standardized, so that $p = 3$. We compute the standard errors from the estimated covariance matrix, $\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}$. Note that $\mathbb{V}(\hat{\beta}_k|\mathbf{X}) = \hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}_{kk}$:

```
vcov.mat <- sigma.hat.sq * solve(xtx)
se1 <- sqrt(vcov.mat[1,1])
se2 <- sqrt(vcov.mat[2,2])
pt(b[1]/se1, n - p)
pt(b[2]/se2, n - p)
```

```
 [1] 0.3797346
 [1] 0.02109715
```

The coefficient on `mass tolerance` is not significant, but the coefficient on `elite tolerance` is significant. But are the two coefficients significantly different from each other? Let $\mathbf{R} = [1 \quad -1]$ and $\mathbf{r} = [0]$. Then the following test statistic will test that the two coefficients are equal.

$$F = \frac{(\mathbf{R}\hat{\beta} - \mathbf{r})'[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}']^{-1}(\mathbf{R}\hat{\beta} - \mathbf{r})}{\hat{\sigma}^2} \tag{6}$$

```
R <- t(matrix(c(-1, 1))); r <- 0
G <- R %*% b - r
(F <- (G %*% R %*% solve(xtx) %*% t(R) %*% t(G))/sigma.hat.sq)
```

```
            [,1]
 [1,] 0.01435461
```

The test statistic follows the F-distribution, and is not significant at any reasonable p-value. So, while `elite tolerance` may be significant in the regression of repression on tolerance, it is not significantly different than the insignificant variable `mass tolerance`.

The purpose of this section is to estimate the returns to education using `R`. There is nothing valid about the results found in this section; but the empirical application gives us a chance to explore categorical dummies and the `ggplot` package. First, as always, we load the required libraries.

```
library(foreign)
library(ggplot2)
library(xtable)
```

We can then read the wage data directly from the online repository for the supplementary data sets for the Wooldridge (2002) text. You will need an internet connection. We only need the `wage`, `educ`, and `age` variables, and we omit all observations with missing observations.

```
f <- "http://fmwww.bc.edu/ec-p/data/wooldridge/wage2.dta"
data <- read.dta(f)
data <- data[ , c("wage", "educ", "age")]
data <- na.omit(data)
```

A quick visualization reveals the distribution of wages in the data set:

```
hist(data$wage, xlab = "wage", main = "", col = "grey", border = "white")
```

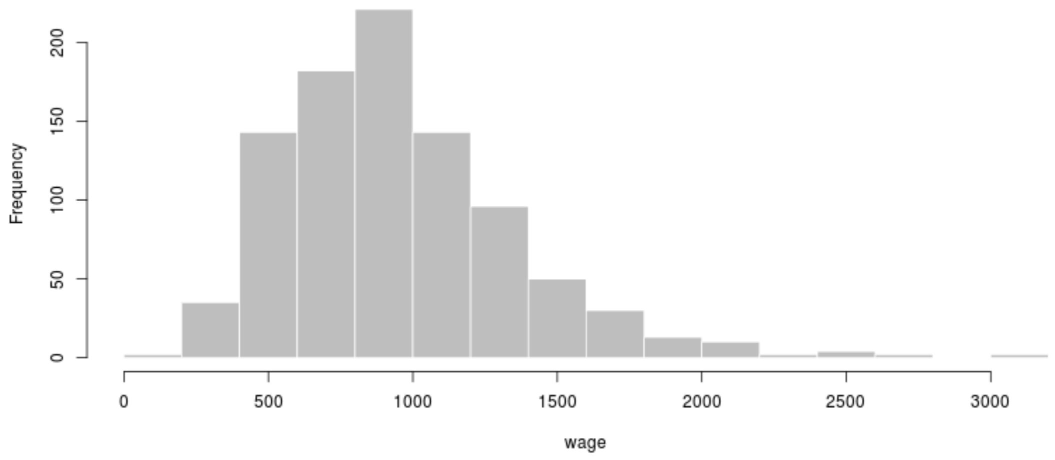

Figure 1: Wage histogram

Roughly following page 38 of the lecture notes, we create a rough measure of educational attainment from the `educ` variable.

```
e1 <- ifelse(data$educ %in% 1:12, 1, 0)
e2 <- ifelse(data$educ %in% 13:14, 1, 0)
e3 <- ifelse(data$educ %in% 15:16, 1, 0)
e4 <- ifelse(data$educ %in% 17:18, 1, 0)
```

The categorical education variables sum to one, and the `lm()` function will force-drop one of the variables. Note that the intercept in this regression reflects the mean wage of the `e4` class. The other coefficients reflect the relative wages of the other three classes.

```
coef(summary(m1 <- lm(wage ~ 1 + e1 + e2 + e3 + e4, data = data)))
```

```
               Estimate Std. Error  t value      Pr(>|t|)
(Intercept) 1196.95876   38.93314 30.743953 7.969254e-144
e1          -350.46396   42.67867 -8.211689  7.239709e-16
e2          -229.97111   49.22796 -4.671555  3.429280e-06
e3           -90.50748   47.64240 -1.899726  5.777793e-02
```

Suppose we want to estimate the premium on education, relative to the least educated class. We can then specify the following regression and print the output directly to LaTeX using the `xtable` package[1]:

```
xtable(m2 <- lm(wage ~ 1 + e2 + e3 + e4, data = data))
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	846.4948	17.4837	48.42	0.0000
e2	120.4929	34.8322	3.46	0.0006
e3	259.9565	32.5528	7.99	0.0000
e4	350.4640	42.6787	8.21	0.0000

None that the coefficients, now, indicate the premium over the base level of education for all subsequent levels. Consider, for example, the premium on the `e4` class. The average wage for people in this class, the class with the highest educational attainment levels, is found by:

```
mean(data[e4 == 1, c("wage")])
```

```
[1] 1196.959
```

This is equivalent to adding the coefficient on `e4` to the intercept from the well-specified regression above. Specifically:

```
b <- m2$coefficients
b[["(Intercept)"]] + b[["e4"]]
```

```
[1] 1196.959
```

This equality only holds because there are no other covariates in the regression. If we condition on age, for example, then the simple addition does not yield an average wage. For illustration, consider the previous regression with `age` and squared `age` as cofactors. Note also the manner by which the `lm()` function accepts a nested function to specify squared `age` within the line:

```
coef(summary(lm(wage ~ 1 + e2 + e3 + e4 + age + I(age^2), data = data)))
```

```
                Estimate  Std. Error     t value     Pr(>|t|)
(Intercept) -148.0041478 1660.163718 -0.08915033 9.289817e-01
e2           142.0919504   34.603321  4.10630965 4.374401e-05
e3           276.4462130   32.327306  8.55147690 4.954754e-17
e4           329.3717601   42.427654  7.76313861 2.181854e-14
age           38.4978013  100.467589  0.38318628 7.016693e-01
I(age^2)      -0.2572671    1.508597 -0.17053405 8.646273e-01
```

It looks like age has a positive but diminishing effect on wage. This makes sense, maybe, but the coefficients are not significantly different from zero. Why might this be the case? This is where some non-parametric graphing comes in handy.

```
(g <- ggplot(data, aes(x=age, y=wage)) + geom_smooth(method="loess", size=1.5))
```

We use the `ggplot2` package instead of the base `R` plotting facilities. The plots reveal a reasonable relationship between wage and age, but there is a significant amount of variation in wage, relative to the short time frame of age.

```
(g <- g + geom_point())
```

[1]Note that this is a little superfluous, but it's worth examining the different ways to export tables.

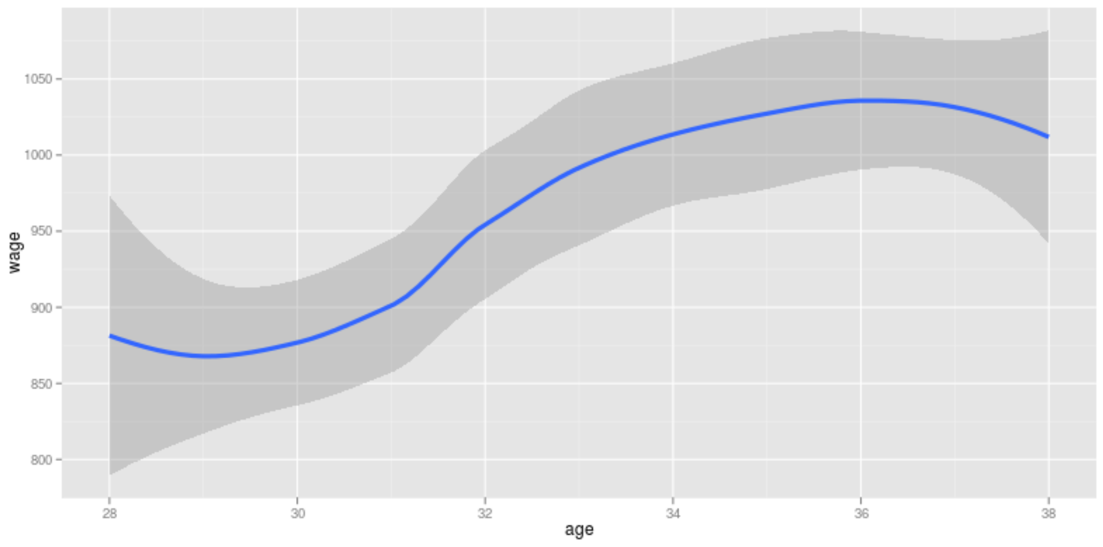

Figure 2: Smoothed line

Maximum Likelihood

Suppose that I tell you that $\mathbf{X}$ is a random vector, where each $\mathbf{X}_i$ is generated from a common density function $\theta/(\theta + \mathbf{X}_i)^2$. We want to estimate θ by maximum likelihood, given the data set `mle.txt` from Freedman (2009). It can be shown that the log-likelihood function is given by the following, where n is the number of observations:

$$\mathscr{L}_n(\theta) = n \log \theta - 2 \sum_{i=1}^{n} \log(\theta + \mathbf{X}_i)$$

First, we will plot the log-likelihood function as a function of θ, and then find the maximum with `optimize`.

```
data <- read.csv("mle.txt", header = FALSE)

logLik <- function(theta, X = data) {
  n <<- nrow(data)
  n * log(theta) - 2 * sum(log(theta + X))
}
```

To maximize this function with respect to θ, we don't have to do any math. And in fact, for this function, there is no explicit function for the maximum likelihood estimate, and we have to find the estimate through numerical optimization.

```
suppressWarnings(opt <- optimize(logLik, interval=c(-100, 100), maximum=TRUE))
(theta.hat <- opt$maximum)
```

```
 [1] 22.50976
```

We can compute the asymptotic variance in a variety of ways, but perhaps the most direct is $[-\mathscr{L}_n''(\hat{\theta})]^{-1}$:

```
dd.logLik <- function(theta, X = data) {
  -1 * (n / theta^2) + 2 * sum(1 / (theta + X)^2)
}

(asy.var <- -1 / dd.logLik(theta.hat))
```

```
 [1] 30.12326
```

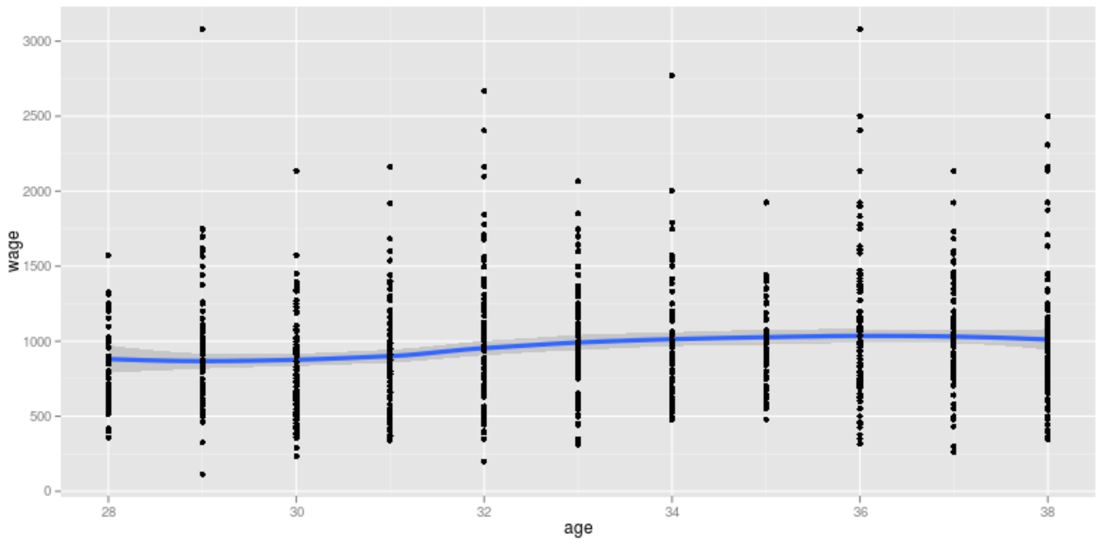

Figure 3: Smoothed line with points

This section will briefly outline two general concepts, GLS and `ggplot`. As an addendum, we will go over the recurring `R` questions that cropped up during the midterm. We will examine the characteristics of generalized least squares (GLS), and specifically the efficiency gains from a special case of GLS, weighted least squares (WLS). This is the econometric concepts. We will then recreate the graphs from Figures 2.6 and 2.7, roughly, in the notes using `ggplot2` a very popular graphing package in `R`. This is part is optional, especially since it is only a very brief treatment of the package — there is a lot more to learn.

Let $x \sim U(0, 2000)$ and $\epsilon \sim N(0, (x/1000)^2)$. The underlying data generating process in (2.102) is $y_i = \alpha + x_i\beta + \epsilon$, where $\alpha = 0.5$ and $\beta = 1.5$. The objective is to plot the simulated sampling distribution of the OLS estimator applied to $B = 10,000$ draws, each of size $n = 1000$. First, let's generate the sample data for one draw.

```
n <- 1000
x <- runif(n, min=0, max=2000)
eps <- rnorm(n, 0, sqrt((x/1000)^2))
y <- 0.5 + x*1.5 + eps
```

Now we can calculate the standard OLS parameter vector $[\hat{\alpha} \quad \hat{\beta}]'$ by noting that $\mathbf{X}$ is just the x vector bound to a column of ones. We will only examine $\hat{\beta}$ for this section, rather than both parameters.

```
X <- cbind(1, x)
params <- solve(t(X) %*% X) %*% t(X) %*% y
beta <- params[2]
print(beta)
```

```
 [1] 1.500033
```

Let's package this into a function, called `rnd.beta`, so that we can collect the OLS parameter for an arbitrary number of random samples, noting that n is a constant so we may as well keep it out of the function so that 1000 is not reassigned thousands of times to n.

```
rnd.beta <- function(i) {
  x <- runif(n)
  eps <- rnorm(n, 0, sqrt(x/10))
  y <- 0.5 + x*1.5 + eps
  X <- cbind(1, x)
  params <- solve(t(X) %*% X) %*% t(X) %*% y
  beta <- params[2]
  return(beta)
}
```

Since there aren't any supplied arguments, the function will return an estimated $\hat{\beta}$ from a a different random sample for each call:

```
rnd.beta()
rnd.beta()
```

```
 [1] 1.503556
 [1] 1.496052
```

This is convenient for bootstrapping without loops, but rather applying the function to a list of effective indices.[1] Now replicating the process for B draws is straightforward:

[1]This is much more comfortable for me, with a background in functional programming. There is some inherent value, however, in keeping code compact by mapping across indices rather than incrementing an index within a `for` loop; the code is more readable and less prone to typos.

```
B <- 1000
beta.vec <- sapply(1:B, rnd.beta)
mean(beta.vec)
```

```
 [1] 1.500078
```

Alright. Looking good. The average of the simulated sample is much closer to β than any individual call of `rnd.beta`, suggesting that the distribution of the simulated parameters will be appropriately centered. Now, let's create another, similar function that returns the WLS estimates.

```
rnd.wls.beta <- function(i) {
  x <- runif(n)
  y <- 0.5 + x*1.5 + rnorm(n, 0, sqrt(x/10))
  C <- diag(1/sqrt(x/10))
  y.wt <- C %*% y
  X.wt <- C %*% cbind(1, x)
  param.wls <- solve(t(X.wt) %*% X.wt) %*% t(X.wt) %*% y.wt
  beta <- param.wls[2]
  return(beta)
}
wls.beta.vec <- sapply(1:B, rnd.wls.beta)
```

We now have two collections of parameter estimates, one based on OLS and another based on WLS. It is straightforward to plot two separate histograms using `R`'s core histogram plotting function `hist()`. However, we can use this to introduce a more flexible, powerful graphing package called `ggplot2`.

```
library(ggplot2)
labels <- c(rep("ols", B), rep("wls", B))
data <- data.frame(beta=c(beta.vec, wls.beta.vec), method=labels)
ggplot(data, aes(x=beta, fill=method)) + geom_density(alpha=0.2)
```

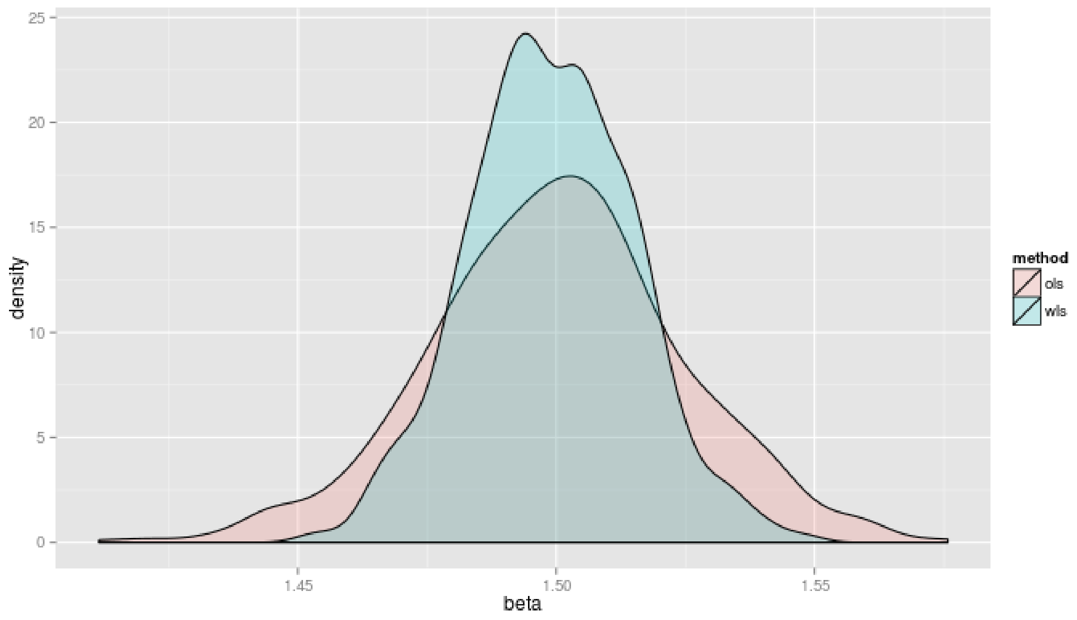

Figure 1: Relative efficiency of WLS

A review of loops, a loopy review

There were many questions about `R` code on the midterm, and especially code that will traverse a sequence of indices. There are many ways to loop over a sequence and iterate on a function — and we do this a lot in ARE212 to examine the behavior of different estimators. I used `sapply()` to iterate over a sequence. There are other ways to do this. Let's consider an equivalent for-loop.

```
res <- rep(NA, B)
for (i in seq(B)) {
  res[i] <- rnd.wls.beta(i)
}
print(head(res))
```

```
 [1] 1.521475 1.485235 1.521788 1.528059 1.473549 1.484154
```

This is fine. It works. You have to pay attention, however, to the expressed indices instead of using them for implicit increments; and you have to preallocate a results vector which also can get confusing over large, sprawling projects. Also, there are more lines of code (LOC) which is often used as a metric for inefficiency.

Another thing that might help avoid loops is something called *vectorized operations*. Instead of looping through indices, is there a way to formulate the problem in terms of element-wise operations? Suppose, for example, that I want to generate a binary vector that indicates whether the value in one variable exceeds the value in another.

```
x <- rnorm(10); y <- rnorm(10)
(D <- ifelse(x > y, 1, 0))
```

```
  [1] 0 1 0 0 1 1 1 1 1 1
```

The function `ifelse()` operates on each individual element of the equal-length `x` and `y` vectors. No looping necessary. This comes up a lot; and you should definitely make use of (and document) this behavior when available. This doesn't always work, however, especially when we are trying to generate a random vector. Suppose we want to create a vector where the elements are pulled from different distributions, depending on the value of `D`.

```
factor(x <- ifelse(D == 1, runif(1), rnorm(1)))
```

```
  [1] -2.02376587549482 0.285012832842767 -2.02376587549482 -2.02376587549482
  [5] 0.285012832842767 0.285012832842767 0.285012832842767 0.285012832842767
  [9] 0.285012832842767 0.285012832842767
 Levels: -2.02376587549482 0.285012832842767
```

What happened? There are only two values in this vector. The random value was generated before the vectorized boolean check. This is equivalent, then, to the code by which we generated `D` in the first place. To fix this problem, we may have to apply the function to each value within the reference vector:

```
randme <- function(d) {
  if (d == 1) {
    r <- runif(1)
  } else {
    r <- rnorm(1)
  }
  return(r)
}

factor(x <- sapply(D, randme))
```

```
  [1] -0.65209157185594  0.154534797882661  0.323631716951203  -0.281180265781314
  [5] 0.254660100210458  0.0384278814308345 0.579372271196917  0.513248251518235
  [9] 0.302135252160951  0.867873660288751
 10 Levels: -0.65209157185594 -0.281180265781314 ... 0.867873660288751
```

This is the fun (read: optional!) section to show how to manipulate and visualize spatial data in R. It seems more fun to start with a data story; and work through some basic analysis.

Data.gov is a data repository administered by the US government with over 445,000 geographic data sets. One dat set is the geographic coordinates and characteristics of 7,863 farmers marketsin the United States.[1] Suppose we are interested in examining the effect of state boundaries on the characteristics of farmers markets. Do state boundaries have a substantive impact on the character of farmers markets, or are the no better than arbitrary lines. There are rigorous ways to address this question, but we will just classify and plot farmers markets, looking for spatial patterns that follow state boundaries.

First, we export the farmers market data set as a CSV file, saving it to the data directory as `farmers-mkts.csv`. Let's just plot the distribution of farmers markets on a a base map of the United States. To do this, you will need to install and then load the following libraries:

```
library(maps)
library(maptools)
library(RColorBrewer)
library(classInt)
library(gpclib)
library(mapdata)
```

The following code plots the map, and presents the results in Figure 1.

```
data <- read.csv("../data/farmers-mkts.csv", header=TRUE)
map("state", interior = FALSE)
map("state", boundary = FALSE, col = "gray", add = TRUE)
points(data$x, data$y, cex = 0.2, col="blue")
```

The distribution of farmers markets across the US is neat to see, but there are so many points that it is difficult to visually glean any useful information, as seen in the following figure. So, instead, lets only consider the farmers markets in Colorado, Utah, New Mexico, and Arizona. There are 354 farmers markets in these four states.

```
statelist <- c("New Mexico", "Colorado", "Arizona", "Utah")
idx <- is.element(data$State, statelist)
state.data <- data[idx, ]
dim(state.data)
```

```
 [1] 354  32
```

Each column of the `state.data` data frame contains information on a different attribute of the farmers markets. The last 24 columns are binary variables with entries `"Y"` or `"N"`, indicating whether the market sells cheese, for example, or accepts credit cards. These are the attributes of interest. The idea is whether we can predict the state of the farmers market, purely based on the characteristics. If so, then the state boundaries are correlated with the attributes for some reason — which is not included in the scope of this example. We can extract these characteristics into a matrix **X** and recode the string variables to 0-1 binary variables. Note that the default numeric levels of the string variables are 2 and 3, so subtracting 2 will yield the desired result.

```
X <- state.data[, 8:ncol(state.data)]
for(i in names(X)) {
  X[[i]] <- as.numeric(X[[i]]) - 2
}
```

[1] https://explore.data.gov/d/wfna-38ey

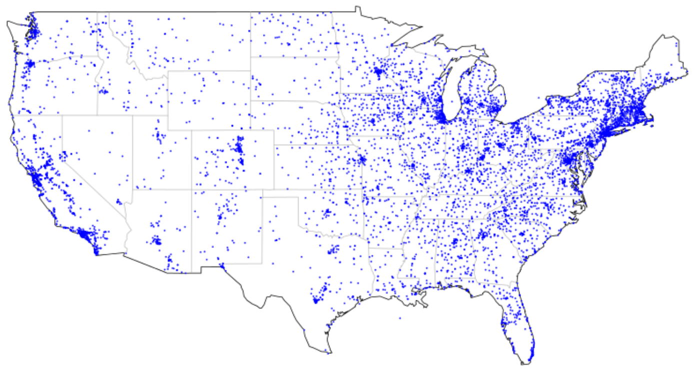

Figure 1: Distribution of US farmers markets

We want to categorize the markets based on their characteristics into four distinct buckets, indicating one of the four states. For this, we can use K-Means clustering, which is a simple call in R. The resulting object cl is a list with various output attributes of the clustering, including to which of the four clusters each market was assigned, indicated by cl$cluster. This is the vector we will use to color the points on the map. There are many other attributes that are collected in cl; three of them are listed below.

```
cl <- kmeans(X, 4)
names(cl)[1:3]
```

```
 [1] "cluster" "centers" "totss"
```

The following code plots the points by zooming in on the four states, and coloring each market by the predicted state. There is slight variation in the output map, as R selects the exact colors on the fly. Below, we save the resulting image to inserts/limited-mkts.png and display one such image, produced previously. We set the number of colors to 4, one for each state; and the brewer.pal() function sets separate color codes for the number of supplied classes according to the color scheme Set1.

```
nclr <- 4
clr.set <- brewer.pal(nclr, "Set1")
```

There exactly four cluster values to match the four-color palette. However, the classIntervals() assigns each color in the palette to a range of values for the more general case. The findColours() function then maps the color codes to each point in the cluster vector; and this is the schema that is used to color the points on the map.

```
class <- classIntervals(cl$cluster, nclr, style = "pretty")
colcode <- findColours(class, cl$cluster)
```

The map is generated in much the same way as the full US map, except that the extent is limited by a latitude-longitude bounding box.

```
map("state", interior = FALSE,
    xlim = c(-117, -101), ylim = c(28, 43))
map("state", boundary = FALSE, col="gray", add = TRUE,
    xlim = c(-117, -101), ylim = c(28, 43))
points(state.data$x, state.data$y, pch = 16, col= colcode, cex = 1)
```

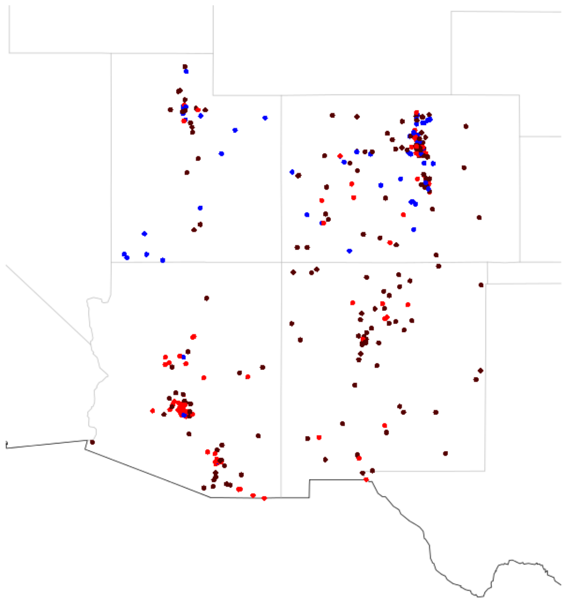

Figure 2: Distribution of US farmers markets

This is cool. It is clear that there is some separable variation in the characteristics of the markets. The red points seem to be clustered in New Mexico, the lower-right state. Even markets right across the border have characteristics that are different enough to “correctly” cluster them by state. There must be something about state regulations that are particularly imposing in New Mexico.

This is just one of many neat examples about how to use `R` to examine spatial variation; and to effectively uncover some spatial, data generating processes that are not obvious from looking at the data in a matrix or spreadsheet. More to come!

The purpose of this section is to showcase the ability to scrape and process web data using R. This tutorial draws heavily from a post on a great blog by Pascal Mickelson and Scott Chamberlain, two biologists and experienced R users. Suppose we want to find the number of available environmental economics journals. There are too many. Definitely. But suppose we want to find out just how many. To do this, we can visit crossref.org, which is a citation-linking network with a list of all journals and their Digital Object Identifiers (DOIs). We will query the list from within R and then parse the returned content to list journals with certain attributes. For this, we'll need to load the following libraries:

```
library(XML)
library(RCurl)
library(stringr)
```

The next step is to repeatedly query crossref.org for journal titles. Try to copy and paste the base URL address (`baseurl`) into your browser: `http://oai.crossref.org/OAIHandler?verb=ListSets`. The result is a long XML form. The function `getURL` in the following code pulls this response into R as a string, and the outer functions `xmlParse` and `xmlToList` convert the output into an R data structure. There are too many entries to fit into a single query, so the `while` loop continues to query until there are no more results. The final results are stored in `nameslist`.

```
token <- "characters"
nameslist <- list()
options(show.error.messages = FALSE)
while (is.character(token) == TRUE) {
    baseurl <- "http://oai.crossref.org/OAIHandler?verb=ListSets"
    if (token == "characters") {
      tok2 <- NULL
    } else {
      tok2 <- paste("&resumptionToken=", token, sep = "")
    }
    query <- paste(baseurl, tok2, sep = "")
    crsets <- xmlToList(xmlParse(getURL(query)))
    names <- as.character(sapply(crsets[[4]], function(x) x[["setName"]]))
    nameslist[[token]] <- names
    if (class(try(crsets[[2]]$.attrs[["resumptionToken"]])) == "try-error") {
        stop("no more data")
    }
    else {
      token <- crsets[[2]]$.attrs[["resumptionToken"]]
    }
}
```

How many journal titles are collected by this query? We first concatenate the results into a single list, and then find the total length:

```
allnames <- do.call(c, nameslist)
length(allnames)
```

```
 [1] 28004
```

Now, suppose that we are looking for just those journals with *economic* in the title. We rely on regular expressions, a common way to parse strings, from within R. The following code snippet detects strings with some variant of *economic*, both lower- and upper-case, and selects those elements from within the `allnames` list.

```
econtitles <- as.character(allnames[str_detect(allnames, "^[Ee]conomic|\\s[Ee]conomic")])
length(econtitles)
```

```
 [1] 444
```

What in the hell? So many! I suppose that this is a good thing: at least one of the 440 journals should accept my crappy papers. If I blindly throw a dart in a bar, it may not hit the dartboard, but it will almost certainly hit one of the 440 patrons. Here is a random sample of ten journals:

```
sample(econtitles, 10)
```

```
 [1] "Tourism Economics"
 [2] "Journal of Contemporary Accounting & Economics"
 [3] "Annals of Public and Cooperative Economics"
 [4] "American Economic Journal Economic Policy"
 [5] "Scottish Economic & Social History"
 [6] "Journal of Housing Economics"
 [7] "Economic Theory"
 [8] "International Journal of Social Economics"
 [9] "Construction Management and Economics"
[10] "North American Review of Economics and Finance"
```