R for STATA Users

Chris Newton

Contents

Chapter 1

Introduction

R for STATA Users is a book designed to take researchers that are already is profecient in STATA, and show them how to do the same analyses in R. The hope is that by starting with what the reader is already comfrotable and showing how to replicate their code in R, the transition to open source software will be gentle. After covering the basics, the book then moves on to more R specific lessons that may introduce techniques that aren't commonly seen in STATA.

This book will show snipets of STATA code, such as:

```
reg dep_var ind_var control_var1 control_var3
```

followed by the corresponding R code to produce the same result. In this case:

```
lm(dep_var ~ ind_var + control_var1 + control_var2,
    data = example.data)
```

There will also be addition answer and explanations. For example, the linear model above can be written differently in R.

```
lm(example.data$dep_var ~ example.data$ind_var +
    example.data$control_var1 + example.data$control_var2)
```

These models are exactly the same. They all run OLS regression In R, you can either write the variables by themselves and then specifly which dataframe they come from, or you can write the dataframe, and $ symbol, and then the variable name, making wure there are no spaces between them. More on this later.

1.1 Who is this book for?

This book is for anyone that is already profecient in STATA that would like to learn how to conduct statistical analyses in R. It is primarily written for researchers, educations, and students in quatitative fields.

1.2 Who isn't this book for?

This books assumes that you are already comfortable with STATA and statistics. If you don't already know STATA, this book will not make a lot of sense as the underlying concepts will not be explained. If you are not already familiar with the math and intuition behind statistical models, this book will not be of much use. In fact, there will be no pure math covered what so ever. The point of this book is not to explain why you should do something statistically, only to show you how to do something you're already familiar with in STATA using R.

1.3 Setting up R

There are already a number of great tutorials on how install and setup R. As this book is unlikely to provide an even better tutorial, instead of taking up space with redundant information, I instead recommend checking out one of these tutorials:

- *Hands-On Programming with R*, by Garrett Grolemund: **A Installing R and RStudio**
- *R for Data Science*, by Hadley Wickham & Garrett Grolemund: **1.4 Prerequisites**

Chapter 2

Why switch to R

If you've gotten this far, I'm assuming that you're already profeceint in STATA. Maybe you're a seasoned researcher with scores of publications on you CV. Maybe you're a grad student, recently emerged from the gauntlet of stats class after stats class, having learned STATA along the way. Perhaps you're asking yourself, Why would I re-learn how to do something I already know?

The answer for many of you is: You shouldn't. Some people don't need to leard R, especially considering that they've already learned how to do everything they need to do in STATA. For everyone else, here are some good reasons to make the switch (or at least learn *some* R).

1. **R is free**

R is an open source programing language available for Windows, Mac, and Linux operating systems. This means that anyone can download it, use it, publish results, develop packages, and other fun stuff without spending any money. Seriously, it's 100% free. 100%. All versions, updates, extra packages, even some books, free. No need for temporary lisences, shelling out for perpetual liscences, buyng new versions, getting your institution to buy it for you, or borrowing that sketchy thumb drive that one person has (not that any reader every pirated anything). It's all free.

2. **R is free**

Seriously, though. It's free. Even if this doesn't matter to you, if you teach, it likely matters to your students. College is expensive, especially if you go for a really long time, like getting a Ph.D. Even for students with scholarships and funding, the financial burden can be tough. Not having to by software, on top of buying an overpriced statistics textbook (the latest edition only!), can make a big difference.

3. **R is the preferred language of statisticians and methodologists in many fields**

If you want to be at the forefront of statistics or your chosen field, there's a good chance that the latest developments are going to come in the form of new R packages before they spreads to any other language. Even before R packages are published, people often post their work on GitHub to be downloaded and used as you will.

4. **R is a programing language**

Getting comfortable in R means learning some fundanamentals of how to write basic code. While with can be extended to developing entire programs in R, learning a bit about functions and loops may mean suffering through a bit of a learning curve, but it will make you life easier down the road. Especially when it come to tedious repetitive tasks, learing a bit about coding can save you lots of time and energy. Learning R means getting comfortable with some of the more basic coding principals.

This also means that learning other prograing languages will be easier. All languages employ the same basic logic, with some variation. Understading how one languages works, means it will be a lot easier to learn another. With maching learning, text analysis, and web development becoming increaasingly popular in Python, this techniques may be some you'll need to learn down the road. Learning the fundamentals in this book can serve at the floaties in the shallow end before diving in the deep end.

5. **Graphics**

R's libraies from visualization – ggplot2 in particular – can produce everything from publication ready graphs, to maps, to animated 3D graphs. The possibilites are vast with other programing builing libraries to imitate the R's. While a web developer creating data visulizations may prefer something such as D3.js that easily runs in a browser, for most researcher, it's hard to beat R when it comes to visualizing you datasets and results.

6. **Boredom**

Sometimes we just want to do things a bit different. Tired of using STATA all the time, use R?

7. **To be condescending to your collegues**

- Oh, you use STATA? That's cute. I use R, like a *real* statistician.

2.1 In conclusion...

As you can see, some of these reasons are better than others. Maybe they all fit your situation, maybe none do. For those that are commited, let write some R code.

Chapter 3

Going Through a Project from Start to Finish

To start off, we're going to look at an example analysis. This will go step-by-step through loading data, exploring the dataset, running a regression, and commuicating the results. You may not understand everything and this point, but you'll start to get familiar with R's syntax and won't have to wait 100 pages before trying out something useful. The following chapters will look at each step in detail to explain exactly what is happening, how it relates to STATA commands, and why we're doing it this way.

3.1 Loading packages

Once R is installed, you now have what is called base-R. Base-R comes ready to go with a number of statistical functions, visualizations, and even some sample datasets. There is, however, plenty more that can be done by loading additional packages. Packages are developed by the comunity of R users and typically hosted on CRAN (The **C**omprehensive **R** **A**rchive **N**etwork). For effeciency, additional packages have to be installed and then called when you want to use them. R doesn't automatically installing or loading additional package as this would take up *a lot* of memory with with packages that you'll never use.

If a package exists on CRAN, it can be installed by writing `install.packages("package.name")`. Most packages that you'll want to use will be hosted on CRAN, but occasionally, new packages that are being developed are only on GitHub. If this is the case, the authos will include instructions on how to install the package in the README.md file of the GitHub repository.

You only have to intall a package once, but you have to call it everytime you open up R. It's the norm to list all of the packages that you'll be using at the very top of you R script. You call a package with the command `library(package.name)`. For this example, were going to use the packages `gt`, `kableExtra`, and `modelsummary` for making regression tables, `dplyr` for data manipulation and pipes (`%>%`) which allow us to string commands together, `tidyr` for datat cleaning and wrangling, and `ggplot2` for making graphs. We can import `dplyr`. `tidyr`, and `ggplot2`, by calling `tidyverse`[1], which automatically loads a collection of packages. So starting out, our script should look like this:

```
library(gt)
library(kableExtra)
library(modelsummary)
library(tidyverse)
```

3.2 Loading and prepping the data

For this example, we are going to use one of the preloaded datasets that comes with R. While you'll never use one of these datasets for actual research, it's easier to use something that everyone already has to get started with an example. The next chapter will show you how to load data that in STATA's .dta format, as well as other common formats.

For this example, we'll use the *Titanic* dataset. This will be obnoxiously familiary for anyone that has done some tutiorials on machine learning. For those unfamiliar, it contains variables on the age, gender, and ticket class for those that were on the titanic, as well as whether or not they survived. To access the data, we type `data("Titanic")`. You should now see `Titanic` in the `Enivronment` tab in RStudio. It's not quite ready yet, however, as it is not in a `data.frame` or `tibble` format. If we typw `class(Titanic)`, we see that it's `table`. This can be converted with the commands `data.frame(Titanic)` or `as_tibble(Titanic)`

For this example, let's convert the table into a data frame. If you simple type `data.frame(Titanic)`, the table will be converted to a data frame, and then printed into the console. We don't want this. We want to store the data fame as an object that we can analyze. In R, you stare and object by first typing an object name of you choosing, followed by the assignment operation (`<-`) and then what you want to be stored in the object. It best to choose descriptive names for objects, so it's easy to remember what they are. Let's use `titanic.data`. The code should hten look like this:

[1]For more information on the tidyverse and how to use the various packages, see *R for Data Science*, by Hadley Wickham & Garrett Grolemund.

```
data("Titanic")
titanic.data <- data.frame(Titanic)
```

You should now see an object called `titanic.data` in you `Environment` with 32 observations or 5 variables. If we want to look at the entire dataset, we can type `view(titanic.data)`. If we only want to see the first few rows, we can type `head(titanic.data)` and the last few can be seen with `tail(titanic.data)`. A frequency table can be seen with `table(titanic.data)`. Let's check that out.

```
table(titanic.data)
```

As you can see, it's hard to glean any information from this as frequency is already a variable, with the other variables being collapsed. We can treate a subset without frequency by typing `titanic.subset <- titanic.data[(1:4)]`. This creates a new data frame called `titanic.subset` with only contains the first four columns of the `titanic.data` data frame. Now trying `table(titanic.subset)`, we see eveything has a ferequency of one. To use this data for a regression, let's expand it so that we have one observation for everyone that was on board. We can do this using the `tidyr` function `uncount()`. This function takes a data frame, and a variable, and then replicated each row so there are as many duplicate rows as the number of the variable supplied. The syntax is

```
uncount(data, weights, .remote = TRUE, .id = NULL)
```

where `data` is the data frome `weights` is the variable that has the count of rows to duplicate, `.remove` deletes the variable supplied to `weights` (TRUE by defalut) and `.id` creates a new ID for each row. For our data, let's type:

```
titanic.expanded <- uncount(titanic.data, Freq)
```

Now let's explore our expanded data. We already used `view()` to look at the full dataset, but with 2201 observations, it can be hard to tell much about what going on. Instead, we're going to generate summary statistics with `summary(titanic.expanded)`. This shows of the level of each variable, the number of ovservations at the level, and, implicitly, that there are no missing values. If there were missing values, the last row of each variable would read `NA's:` followed by the number of rows for which that variable didn't have a value.

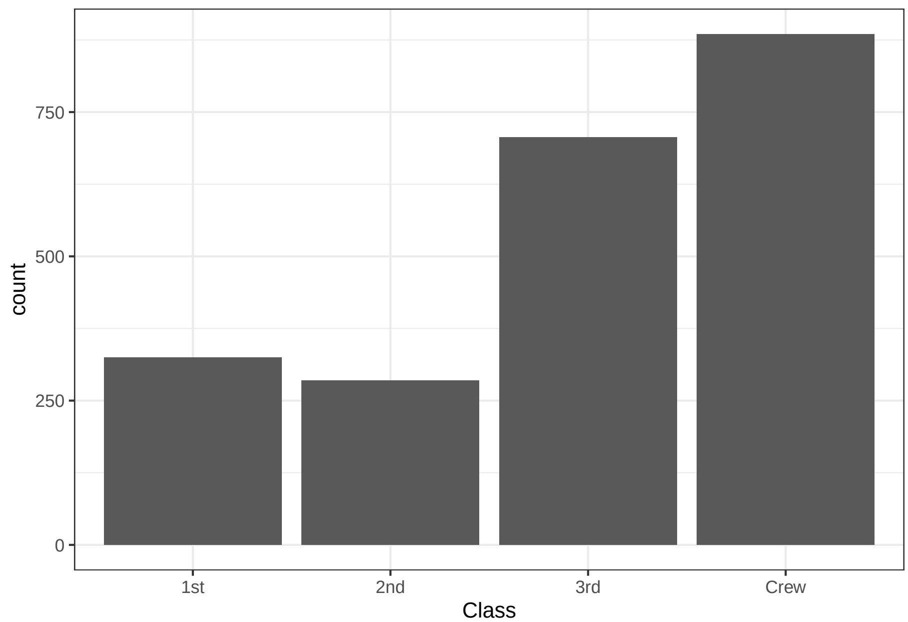
Now were have a data frame that we can analyze, we no longer need the original data, or the subset we created. We can get rid of these with `rm(list = c('Titanic', 'titanic.data', 'titanic.subset'))`. R can use a lot of memory on your computer, so it's best to get rid of any objects that you're no longer using.

3.3 Visualization

Let's look at our data using some plots. First, we're going to check the distribution of our variables. Given that all of our variables are factors, a histogram it the was to go. Using `ggplot2`, we can do this:

```
ggplot() +
    geom_histogram(data = titanic.expanded,
        aes(x = Class), stat = 'count') +
    theme_bw()
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```

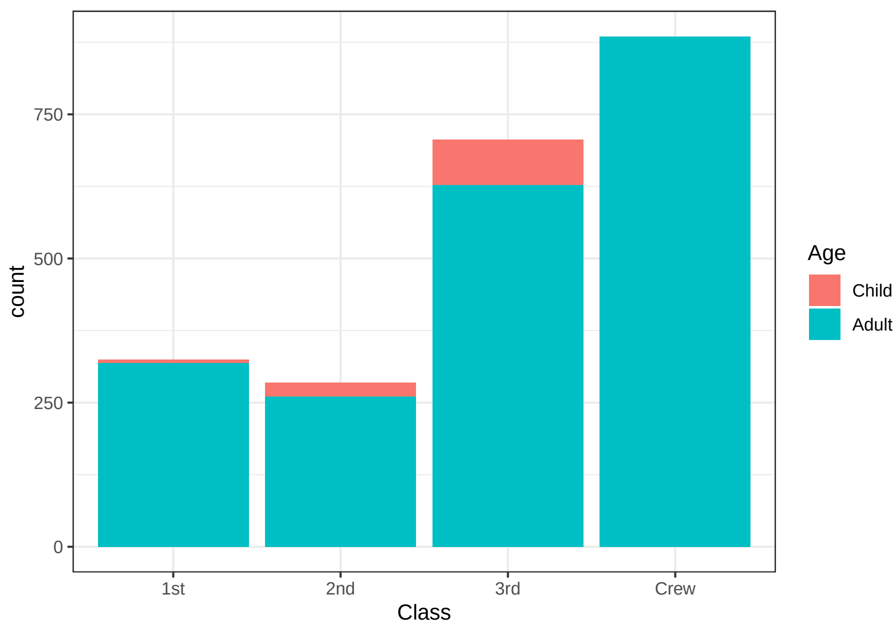


The above calls a plot (`ggplot()`) and then says that we're going to make a histogram (`geom_histogram()`). We're going to use the `titanic.expanded` data, and we're want to see the variable `Class`. `aes()` is responsible for creating the mapping, in other words, with the variables that are being plotting. We include `stat = 'count'` as we're looking at the frequenqy of each level of the variable `Class`. Finally, `theme_bw()` styles the graph. This part is optional, and there are plent of other themes you can choose from,in including custom themes that you can make yourself. `ggplot2` uses the *grammar of graphics* which layers different aspects of a vizualiztion on top of each other. Each layer is connected with a `+`. While you could keep everything on one line and the code will still run, it is best to end each line with a `+` and the start on the next line with an indent. This keeps the code organized and easy to read.

Now say you also wanted to see how may within each class were chilren and how many were adults. This could be done by changing the `fill`.

```
ggplot() +
    geom_histogram(data = titanic.expanded,
        aes(x = Class, fill = Age), stat = 'count') +
    theme_bw()
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



3.4 Modeling

We're going to build a model to predict whether or not someone would survive bacsed on the variables we have. `Survived` is a binary variable, so we'll estimate a logit model.

```
titanic.logit <- glm(Survived ~ Class + Sex + Age,
    data = titanic.expanded, family = 'binomial')
summary(titanic.logit)
```

```
##
## Call:
```

```
## glm(formula = Survived ~ Class + Sex + Age, family = "binomial",
##     data = titanic.expanded)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0812  -0.7149  -0.6656   0.6858   2.1278
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.6853     0.2730   2.510   0.0121 *
## Class2nd     -1.0181     0.1960  -5.194 2.05e-07 ***
## Class3rd     -1.7778     0.1716 -10.362  < 2e-16 ***
## ClassCrew    -0.8577     0.1573  -5.451 5.00e-08 ***
## SexFemale     2.4201     0.1404  17.236  < 2e-16 ***
## AgeAdult     -1.0615     0.2440  -4.350 1.36e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2769.5  on 2200  degrees of freedom
## Residual deviance: 2210.1  on 2195  degrees of freedom
## AIC: 2222.1
##
## Number of Fisher Scoring iterations: 4
```

Unpacking the above command, `glm()` calls a generalized linear model, with `Survived` as the dependent variable, `Class`, `Sex`, and `Age`, as inependent variables, using the `titanic.expanded` data frame. `family = 'binomial'` declares that the model is a logit, and we save this as an object called `titanic.logit`. The `summary()` command gives us the statistical information we're want to know about the model.

3.5 Reporting

Now have results, we need to communicate them. Let's start with a nice table. Typing `modelsummary(titanic.logit, stars = TRUE)` gives us a basic table, but the variable names aren't formated nicely. We can change this by creating an object with new names, and adding `coef_map = independent.var.names` to `modelsummary()`:

```
independent.var.names = c(
    'Class2nd' = 'Second Class',
    'Class3rd' = 'Third Class',
```

```
    'ClassCrew' = 'Crew',
    'SexFemale' = 'Sex (Female)',
    'AgeAdult' = 'Age (Adult)'
)

modelsummary(titanic.logit, stars = TRUE,
    coef_map = independent.var.names)
```

	Model 1
Second Class	-1.018***
	(0.196)
Third Class	-1.778***
	(0.172)
Crew	-0.858***
	(0.157)
Sex (Female)	2.420***
	(0.140)
Age (Adult)	-1.062***
	(0.244)
Num.Obs.	2201
AIC	2222.1
BIC	2256.2
Log.Lik.	-1105.031

* $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$

And say we have multiple models, such as one for each independent variable plus our original model, we can report all of them like this:

```
models = list(
    model1 <- glm(Survived ~ Class,
        data = titanic.expanded, family = 'binomial'),
    model2 <- glm(Survived ~ Sex,
        data = titanic.expanded, family = 'binomial'),
    model3 <- glm(Survived ~ Age,
        data = titanic.expanded, family = 'binomial'),
    model4 <- glm(Survived ~ Class + Sex + Age,
        data = titanic.expanded, family = 'binomial')
)

independent.var.names = c(
    'Class2nd' = 'Second Class',
```

```
    'Class3rd' = 'Third Class',
    'ClassCrew' = 'Crew',
    'SexFemale' = 'Sex (Female)',
    'AgeAdult' = 'Age (Adult)'
)

modelsummary(models, stars = TRUE,
    coef_map = independent.var.names)
```

	Model 1	Model 2	Model 3	Model 4
Second Class	-0.856***			-1.018***
	(0.166)			(0.196)
Third Class	-1.596***			-1.778***
	(0.144)			(0.172)
Crew	-1.664***			-0.858***
	(0.139)			(0.157)
Sex (Female)		2.317***		2.420***
		(0.120)		(0.140)
Age (Adult)			-0.880***	-1.062***
			(0.197)	(0.244)
Num.Obs.	2201	2201	2201	2201
AIC	2596.6	2339.0	2753.9	2222.1
BIC	2619.3	2350.4	2765.3	2256.2
Log.Lik.	-1294.278	-1167.494	-1374.948	-1105.031

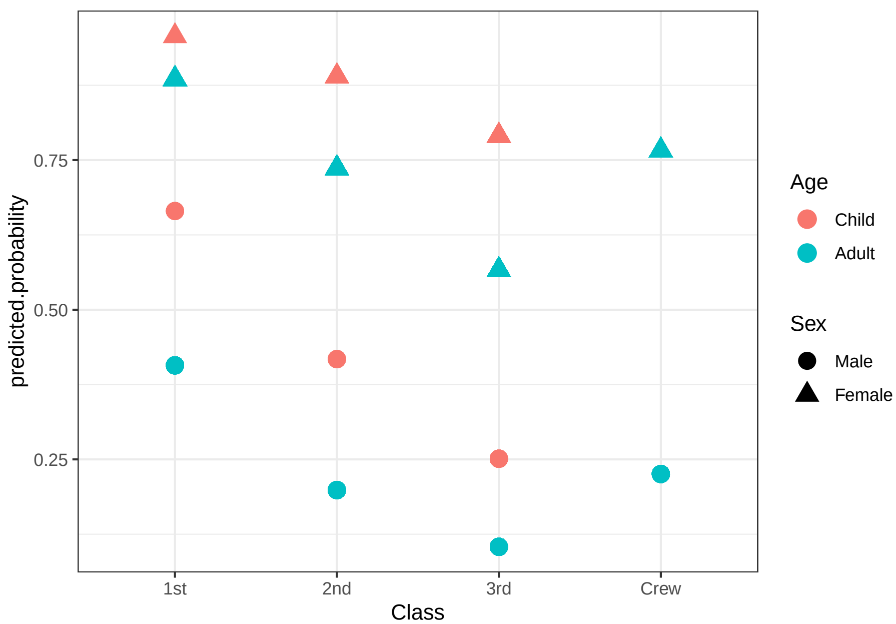
* $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$

We can graph our results using ggplot2, but first we need to calculate the predicted probailities.

```
titanic.predictions <- cbind(titanic.expanded,
    predict(titanic.logit, newdata = titanic.expanded,
        type = 'link', se = TRUE))

titanic.predictions <- within(titanic.predictions, {
    predicted.probability <- plogis(fit)
    }
)

ggplot(titanic.predictions, aes(Class, predicted.probability)) +
    geom_point(aes(color = Age, shape = Sex), size = 4) +
    theme_bw()
```



And there you have it. A complete project from start to finish in R. There are of course plenty of other things we could have done, but this chapter is about getting a taste for R. In future chapters we'll go much futher in depth to each step and still only cover a portion of what's possible in R.

Chapter 4

Dealing with .dta

4.1 Loading packages

The first step for moving from STATA to R is getting access to your .dta files. There are multiple packages that can read and write .dta files, were going to use `haven`.

Haven is not part of base-R, so it has to be installed if you haven't done so before. One a package has been installed on a system, however, you don't need to reinstall it. You should already know now to install packages, from the previous chapter. With have in stalled, put `library(haven)` at the top of you R script. R scripts (as with scripts in all programming languages) with the first line and goes through the program. As long as `library(haven)` appears in you file before you attept to use it, you should be fine. That being said, the convention is to put all packages used at the beginning of your script, with each being on a new line (ex: *figure 1*).

figure 1

```
library(dplyr)
library(ggplot2)
library(haven)
```

4.2 Using haven to import STATA files

Below the packages that you'll be using, you then need to import your data. This is done with the command `read_dta()`. As you probably want to actually load the data into the environment, and not just print the observations in the colsole, you'll have to assign the data a name. In R, names can contain uppercase and lowercase letter, numbers, underscores, and periods. An object name cannot, contain spaces, begin with a number, or contain symbols such as $ or %. Also, names cannot be the same as a function in base-R or any of the packages you are using.

```
my.data <- read_dta('data.dta') # this works
MyDaTa <- read_dta('data.dta') # so does this

# this doesn't work, sum() is a function in base-R
 sum <- read_dta('data.dta')

data2 <- read_dta('data.dat') # this is fine
2data <- read_dta('data.dat') # this isn't
```

Names are assigned by using `<-`, typically with a space before and after (though this isn't necessary, it keeps the code clean and easy to read). To load your data, give it a name (that conforms with R's rules) followed by `<-` and then `read_dta()`. If your data is in your working directory, you can simply write the file name inside either single (”) or double ("") quotes. To find you working directory, type `getwd()` into the console.

If your file is in a subdirectory of your workind directory, you can simple speficy the subdirectiory. For example, if you keep all of your datasets in a folder 'data', you would type `read_dta('data/data.dta')`. If you data is outside of your working directory, you can specify the complete file path. for example `read_dta('~/home/user/Desktop/datasets/data.dta')`.

4.3 Dealing with errors

The `read_dta()` function supports STATA versions 8-15. If you import your file and it doesn't look right, there may be an issue with interpreting the version. This can be fixed by adding a comma after the file name, followed by `version =` and then the version of STATA that wrote the file For example, importing a file from STATA 10 would look like this:

```
stata.data <- read_dta('data/data.dta', version = 10)
```

If there's still an issue, it might be the econding. Before STATA 14, files just relied on the default decosing of the system when writing a file. This means that a file written on Windows may not have the same encoding as one written on Mac or Linux. If you get the meggage `"Unable to convert string to the requested encoding"`, it's probably because STATA saved the default Windows encoding, windows-1252. To fix this, add `encoding = "latin1"` after the version (again seperated with a comma).

```
stata.data <- read_dta('data/data.dta', version = 10, encoding = "latin1")
```

Of course, if you saved the file on you own computer or the file was save using STATA 14 or newer. This shouldn't be a problem.

4.4 But there's still problems

If you're still having errors at this point, the best option is probable to quit. Unistall R, throw your laptop into the sea, fish it our because you're worried about pollution, chuck it in rice because you realize that you started learning R to save money.

Or, start practicing th single most important programming skill there is: looking up the answer on the internet. When something doesn't run, R prints and error message

4.5 Other data formats

Chapter 5

Linear Regression

Let's start with plain old OLS. In STATA this is done using the command `reg`. In R, OLS is run using the commend `lm()`. This is part of base-R, which means that there are no packages to install, you just start R and you're ready to go.

While STATA seperates each part of the regression with a space, R wraps everything in parentheses. You don't have to include any spaces between the elements inside the pathenses, but it best to do so, for readability. You do have to add `~`, `+`, and `:`. `~` goes between the dependent variabale, and the rest of the equation, `+` seperates the rest of the variables in an additive model, and `:` indicates a multiplitcative interation. So `lm(y ~ x1 + x2 + x1:x2, data = stata.data)` is the same as `gen x1x2 = x1*x2` followed by `reg y x1 x2 x1x2` in STATA.

This R code could also be written:

```
lm(stata.data$y ~ stata.data$x1 + stata.data$x2 + stata.data$x1:stata.data$x2)
```

And just to add even more variety, one could also write:

```
attach(stata.data)
lm(y ~ x1 + x2 + x1:x2)
```

5.1 Which way should you write your variables?

Starting with `attach(stata.data)` is likely to be most comfortable for STATA users. This method loads a single dataframe into the environment (in this case stata.data) and now any variable you reference is assumed to belong to that dataframe. If you call a variable that doesn't exist in the dataframe, you see the error `Error in eval(predvars, data, env) : object 'variable' not` `found` where `'variable` is that name of the non-existant variable you tried to call.

If you want to switch to another dataframe, you simple write `detach(stata.data)` and then attach another dataframe (i.e. `attach(stata.data2)`). Note that `detach()` will not remove the data from the environment, it only removes it from being the default for calling variables. If you want to remove the data completly, type `rm(stata.data)`.

While this may be the most similar to the way you're used to working with dataframes in STATA, one of the advantages of R is that you can work with various different dataframes at the same time.