Quant II Recitation

Antonella Bandiera `aab639@nyu.edu`

January, 2018

Logistics

- Office hours will be on Tuesdays 4-6
- I have created a recitation repository on github where you will find this presentation:
  `https://github.com/antobandiera/Quant-II`
- All homework will be submitted to me in hard copy
- All replication code will be submitted to me via email
- **Code should be well commented**
- I will only accept high quality tables and plots
- A great deal of the code is work by Drew Dimmery

Installing R

- It depends on whether you are a Mac or Windows user
- For MAC, go to `http://cran.r-project.org/bin/macosx/`
- For Windows, go to `http://cran.r-project.org/bin/windows/base/`
- You should choose an editor and learn how it works
- I use RStudio, but there is vim, emacs, Notepad++, etc.

When things break

- Documentation - Ex: `?lm`
- Google
- CRAN (Reference manuals, vignettes, etc) - Ex: `http://cran.r-project.org/web/packages/AER/index.html`
- JSS - Ex: `http://www.jstatsoft.org/v27/i02`
- Stack Overflow - `http://stackoverflow.com/questions/tagged/r`
- Listservs - `http://www.r-project.org/mail.html`

Resources

- The Art of R Programming - N. Matloff
- Modern Applied Statistics with S - W. Venables and B. Ripley
- Advanced R Programming - forthcoming, H. Wickham
- The R Inferno - P. Burns
- Rdataviz - a talk by P. Barberá on ggplot2
- Basic Intro to R - also by P. Barberá
- Jamie Monogan

R

- If you see a $+$ means a parenthesis or bracket is open.
- R is case sensitive
- Use / in path names. Not \.
- R is an object-oriented programming language
  - Data
  - Procedures
- Object's procedures can acces and modify the data fields of objects

Using Third-party Code

- Relevant commands are: `install.packages` and `library`
- Find the appropriate packages and commands with Google and via searching in R:

```
?covariance
??covariance
install.packages("sandwich")
library("sandwich")
?vcovHC
```

Data types

- Character - strings
- Double / Numeric - numbers
- Logical - true/false
- Factor - unordered categorical variables

Character

Character

```
my.name <- "Antonella"
paste("My","name","is","Antonella", sep=" ")
```

```
## [1] "My name is Antonella"
```

```
as.character(99)
```

```
## [1] "99"
```

```
class(my.name)
```

```
## [1] "character"
```

Numeric

Numeric

```
num <- 99.867
class(num)
```

```
## [1] "numeric"
```

```
round(num, digits=2)
```

```
## [1] 99.87
```

```
pi
```

```
## [1] 3.141593
```

```
exp(1)
```

```
## [1] 2.718282
```

Numeric

- `sin`, `exp`, `log`, `factorial`, `choose`, are some useful mathematical functions
- You probably noticed that "<-" is an assignment operator
- It lets you store objects and use them later on
- You can also use "="
- To know what is stored you can use the ls() function

```
ls()
```

```
## [1] "my.name" "num"
```

- To remove something, rm(object)
- To remove everything that is stored use rm(list=ls())

Logical

- The logical type allows us to make statements about truth

Logical

- The logical type allows us to make statements about truth

```r
2 == 4
```

```
## [1] FALSE
```

```r
class(2==4)
```

```
## [1] "logical"
```

Logical

- The logical type allows us to make statements about truth

```
2 == 4
```

```
## [1] FALSE
```

```
class(2==4)
```

```
## [1] "logical"
```

```
my.name != num
```

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

Logical

- The logical type allows us to make statements about truth

```
2 == 4
```

```
## [1] FALSE
```

```
class(2==4)
```

```
## [1] "logical"
```

```
my.name != num
```

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

```
"34" == 34
```

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

- `==`, `!=`, `>`, `<`, `>=`, `<=`, `!`, `&`, `|`, `any`, `all`, etc

Objects

- Many functions will return objects rather than a single datatype.

Objects

- Many functions will return objects rather than a single datatype.

```
X <- 1:100; Y <- rnorm(100,X)
out.lm <- lm(Y~X)
class(out.lm)
```

```
## [1] "lm"
```

- Objects can have other data embedded inside them

Objects

- Many functions will return objects rather than a single datatype.

```
X <- 1:100; Y <- rnorm(100,X)
out.lm <- lm(Y~X)
class(out.lm)
```

```
## [1] "lm"
```

- Objects can have other data embedded inside them

```
out.lm$coefficients
```

```
## (Intercept)           X
##  -0.2055415   1.0031144
```

Data Structures

- There are other ways to hold data, though:
  - Vectors
  - Lists
  - Matrices
  - Dataframes

Vectors

- Almost everything in R is a vector.

Vectors

- Almost everything in R is a vector.

```r
as.vector(4)
```

```
## [1] 4
```

```r
4
```

```
## [1] 4
```

Vectors

- Almost everything in R is a vector.

```r
as.vector(4)
```

```
## [1] 4
```

```r
4
```

```
## [1] 4
```

- We can combine elements in vectors with `c`, for concatenate:

Vectors

- Almost everything in R is a vector.

```
as.vector(4)
```

```
## [1] 4
```

```
4
```

```
## [1] 4
```

- We can combine elements in vectors with `c`, for concatenate:

```
vec <- c("a","b","c")
vec
```

```
## [1] "a" "b" "c"
```

Vectors

- Almost everything in R is a vector.

```
as.vector(4)
```

```
## [1] 4
```

```
4
```

```
## [1] 4
```

- We can combine elements in vectors with `c`, for concatenate:

```
vec <- c("a","b","c")
vec
```

```
## [1] "a" "b" "c"
```

```
c(2,3,vec)
```

Vectors (cont.)

- Sometimes R does some weird stuff:

Vectors (cont.)

- Sometimes R does some weird stuff:

```
c(1,2,3,4) + c(1,2)
```

```
## [1] 2 4 4 6
```

- It “recycles” the shorter vector:

Vectors (cont.)

- Sometimes R does some weird stuff:

```r
c(1,2,3,4) + c(1,2)
```

```
## [1] 2 4 4 6
```

- It "recycles" the shorter vector:

```r
c(1,2,3,4) + c(1,2,1,2)
```

```
## [1] 2 4 4 6
```

Vectors (cont.)

- Sometimes R does some weird stuff:

```
c(1,2,3,4) + c(1,2)
```

```
## [1] 2 4 4 6
```

- It "recycles" the shorter vector:

```
c(1,2,3,4) + c(1,2,1,2)
```

```
## [1] 2 4 4 6
```

```
c(1,2,3,4) + c(1,2,3)
```

```
## Warning in c(1, 2, 3, 4) + c(1, 2, 3): longer object len
## multiple of shorter object length
```

```
## [1] 2 4 6 5
```

More Vectors

- We can index vectors in several ways

More Vectors

- We can index vectors in several ways

```r
vec[1]
```

```
## [1] "a"
```

More Vectors

- We can index vectors in several ways

```
vec[1]
```

```
## [1] "a"
```

```
names(vec) <- c("first","second","third")
vec
```

```
##  first second  third
##    "a"    "b"    "c"
```

More Vectors

- We can index vectors in several ways

```
vec[1]
```

```
## [1] "a"
```

```
names(vec) <- c("first","second","third")
vec
```

```
##  first second  third
##    "a"    "b"    "c"
```

```
vec["first"]
```

```
## first
##   "a"
```

Creating Vectors

```r
vector1 <- 1:5
vector1
```

```
## [1] 1 2 3 4 5
```

```r
vector1 <- c(1:5,7,11)
vector1
```

```
## [1]  1  2  3  4  5  7 11
```

```r
vector2 <- 1:7
vector2
```

```
## [1] 1 2 3 4 5 6 7
```

Creating Vectors

```r
cbind(vector1,vector2)
```

```
##      vector1 vector2
## [1,]       1       1
## [2,]       2       2
## [3,]       3       3
## [4,]       4       4
## [5,]       5       5
## [6,]       7       6
## [7,]      11       7
```

```r
rbind(vector1,vector2)
```

```
##         [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## vector1    1    2    3    4    5    7   11
## vector2    1    2    3    4    5    6    7
```

Missingness

Missingness

```
vec[1] <- NA
vec
```

```
##  first second  third
##     NA    "b"    "c"
```

Missingness

```
vec[1] <- NA
vec
```

```
##  first second  third
##     NA    "b"    "c"
```

```
is.na(vec)
```

```
##  first second  third
##   TRUE  FALSE  FALSE
```

Missingness

```r
vec[1] <- NA
vec
```

```
##  first second  third
##     NA    "b"    "c"
```

```r
is.na(vec)
```

```
##  first second  third
##   TRUE  FALSE  FALSE
```

```r
vec[!is.na(vec)] # vec[complete.cases(vec)]
```

```
## second  third
##    "b"    "c"
```

Lists

- Lists are similar to vectors, but they allow for arbitrary mixing of types and lengths.

Lists

- Lists are similar to vectors, but they allow for arbitrary mixing of types and lengths.

```
listie <- list(first = vec, second = num)
listie
```

```
## $first
##  first second  third
##     NA    "b"    "c"
##
## $second
## [1] 99.867
```

Lists

```r
listie[[1]]
```

```
##  first second  third
##     NA    "b"    "c"
```

```r
listie$first
```

```
##  first second  third
##     NA    "b"    "c"
```

Basic Functions

```r
a <- c(1,2,3,4,5)
a
```

```
## [1] 1 2 3 4 5
```

```r
sum(a)
```

```
## [1] 15
```

```r
max(a)
```

```
## [1] 5
```

```r
min(a)
```

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

Basic Functions

```r
length(a)
```

```
## [1] 5
```

```r
length <- length(a)
b <- seq(from=0,to=5,by=.5)
c <- rep(10,27)
d <- runif(100)
```

More later # Matrices

- $$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$
- A_{ij}
- $A_{1,2} = 3$
- $A_{1,\cdot} = (1, 3)$

Basic Functions

```
length(a)
```

```
## [1] 5
```

```
length <- length(a)
b <- seq(from=0,to=5,by=.5)
c <- rep(10,27)
d <- runif(100)
```

More later # Matrices

- $$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$
- A_{ij}
- $A_{1,2} = 3$
- $A_{1,\cdot} = (1, 3)$

Matrix Operations

- Its very easy to manipulate matrices:

Matrix Operations

- Its very easy to manipulate matrices:

```
solve(A) #A^{-1}
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

Matrix Operations

- Its very easy to manipulate matrices:

```
solve(A) #A^{-1}
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
10*A
```

```
##      [,1] [,2]
## [1,]   10   30
## [2,]   20   40
```

Matrix Operations

```
B<-diag(c(1,2)) #Extract or replace diagonal of a matrix
B
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    2
```

Matrix Operations

```r
B<-diag(c(1,2)) #Extract or replace diagonal of a matrix
B
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    2
```

```r
A%*%B
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    8
```

More Matrix Ops.

More Matrix Ops.

```r
t(A) # A'
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

More Matrix Ops.

```r
t(A) # A'
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```r
rbind(A,B)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
## [3,]    1    0
## [4,]    0    2
```

More Matrix Ops.

```r
cbind(A,B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    1    0
## [2,]    2    4    0    2
```

More Matrix Ops.

```r
cbind(A,B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    1    0
## [2,]    2    4    0    2
```

```r
c(1,2,3)%x%c(1,1) # Kronecker Product
```

```
## [1] 1 1 2 2 3 3
```

Naming Things

Naming Things

```
rownames(A)
```

```
## NULL
```

Naming Things

```r
rownames(A)
```

```
## NULL
```

```r
rownames(A)<-c("a","b")
colnames(A)<-c("c","d")
A
```

```
##   c d
## a 1 3
## b 2 4
```

Naming Things

```
rownames(A)
```

```
## NULL
```

```
rownames(A)<-c("a","b")
colnames(A)<-c("c","d")
A
```

```
##   c d
## a 1 3
## b 2 4
```

```
A[,"d"]
```

```
## a b
## 3 4
```

Naming things

- Matrices are vectors:

Naming things

- Matrices are vectors:

```r
A[3]
```

```
## [1] 3
```

Dataframes

- The workhorse
- Basically just a matrix that allows mixing of types.
- R has a bunch of datasets . . .

```
# data() gives you all the datasets
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Spec
## 1          5.1         3.5          1.4         0.2  set
## 2          4.9         3.0          1.4         0.2  set
## 3          4.7         3.2          1.3         0.2  set
## 4          4.6         3.1          1.5         0.2  set
## 5          5.0         3.6          1.4         0.2  set
## 6          5.4         3.9          1.7         0.4  set
```

Dataframes

- But you will generally work with your own datasets

```
getwd()
```

```
## [1] "/Users/antobandiera/Dropbox/Teaching NYU/QUANT II/R
```

```
setwd("/Users/antobandiera/Dropbox/Teaching NYU/QUANT II/R
```

- R can read any number of file types (.csv, .txt, etc.)

```
#.CSV
dat.csv <- read.csv("http://stat511.cwick.co.nz/homeworks/a
```

Dataframes

```r
#STATA
require(foreign)
```

```
## Loading required package: foreign
```

```
## Warning: package 'foreign' was built under R version 3.2
```

```r
dat.data <- read.dta("https://stats.idre.ucla.edu/stat/data
```

Control Flow

- loops
- if/else
- functions
- useful stock functions to know

Loops

- for loops - a way to say "do this for each element of the index"
- "this" is defined in what follows the "for" expression

Loops

- for loops - a way to say "do this for each element of the index"
- "this" is defined in what follows the "for" expression

```
for(i in 1:5) {
  cat(i*10," ")
}
```

```
## 10  20  30  40  50
```

Loops

- for loops - a way to say “do this for each element of the index”
- “this” is defined in what follows the “for” expression

```
for(i in 1:5) {
  cat(i*10," ")
}
```

```
## 10  20  30  40  50
```

```
for(i in 1:length(vec)) {
  cat(vec[i]," ")
}
```

```
## NA  b  c
```

Loops

- for loops - a way to say "do this for each element of the index"
- "this" is defined in what follows the "for" expression

```
for(i in 1:5) {
  cat(i*10," ")
}
```

```
## 10  20  30  40  50
```

```
for(i in 1:length(vec)) {
  cat(vec[i]," ")
}
```

```
## NA  b  c
```

```
for(i in vec) {
  cat(i," ")
}
```

If/Else

If/Else

```
if(vec[2]=="b") print("Hello World!")
```

```
## [1] "Hello World!"
```

If/Else

```
if(vec[2]=="b") print("Hello World!")
```

```
## [1] "Hello World!"
```

```
if(vec[3]=="a") {
  print("Hello World!")
} else {
  print("!dlroW olleH")
}
```

```
## [1] "!dlroW olleH"
```

Vectorized If/Else

- Conditional execution on each element of a vector

Vectorized If/Else

- Conditional execution on each element of a vector

```
vec <- letters[1:3]
new <- vector(length=length(vec))
for(i in 1:length(vec)) {
  if(vec[i]=="b") {
    new[i] <- 13
  } else {
    new[i] <- 0
  }
}
new
```

```
## [1]  0 13  0
```

Vectorized If/Else

```r
new <- ifelse(vec=="b",13,0)
new
```

```
## [1]  0 13  0
```

Functions

- $f : X \to Y$
- Functions in R are largely the same. ("Pure functions")

Functions

- $f : X \rightarrow Y$
- Functions in R are largely the same. ("Pure functions")

```
add3 <- function(X) {
  return(X+3)
}
add3(2)
```

```
## [1] 5
```

Functions

```r
my.function <- function(x,y,z){
  out <- (x + y)*z
  return(out)
}
my.function(x = 5, y = 10, z = 3)
```

```
## [1] 45
```

```r
my.function(2,2,2)
```

```
## [1] 8
```

Useful Functions

- Note: Most functions don't do complete case analysis by default (usually option na.rm=TRUE)
- `print`, `cat`, `paste`, `with`, `length`, `sort`, `order`, `unique`, `rep`, `nrow`, `ncol`, `complete.cases`, `subset`, `merge`, `mean`, `sum`, `sd`, `var`, `lag`,`lm`, `model.matrix`,`coef`, `vcov`, `residuals`, `vcovHC` (from `sandwich`), `ivreg` (from `AER`), `countrycode` (from`countrycode`),`summary`, `pdf`, `plot`, Tools from `plm`, and many more.

Distributional Functions

- `?Distributions`
- They have a consistent naming scheme.
- `rnorm`, `dnorm`, `qnorm`, `pnorm`
- `rdist` - generate random variable from dist
- `ddist` - density function of dist
- `qdist` - quantile function of dist
- `pdist` - distribution function of dist
- look at documentation for parameterization

Distributional Functions

- ?Distributions
- They have a consistent naming scheme.
- rnorm, dnorm, qnorm, pnorm
- rdist - generate random variable from dist
- ddist - density function of dist
- qdist - quantile function of dist
- pdist - distribution function of dist
- look at documentation for parameterization

```r
rnorm(16)
```

```
##  [1] -2.09736833 -1.34493702  0.25430391  0.15071400  0.
##  [6]  0.08692846  0.98243228 -1.14323734 -1.22621817 -1.
## [11]  0.71850181  0.91158074 -0.29711217 -2.16165357  0.
## [16]  0.77515527
```

The *apply family

- These functions allow one to *efficiently* perform a large number of actions on data.
- `apply` - performs actions on the rows or columns of a matrix/array (1 for rows, 2 for columns)
- `sapply` - performs actions on every element of a vector
- `tapply` - performs actions on a vector by group
- `replicate` - performs the same action a given number of times

apply

```
A
```

```
##   c d
## a 1 3
## b 2 4
```

```
apply(A,1,sum) # margin = 1, row
```

```
## a b
## 4 6
```

```
apply(A,2,mean) # margin = 2, column
```

```
##   c   d
## 1.5 3.5
```

Working With Data

- Input
- Output

Input

Input

```r
setwd("/Users/antobandiera/Dropbox/Teaching NYU/QUANT II/RI
dir()
```

```
## [1] "apsrtable.png"          "clt_notes_updated.pdf" "fig
## [4] "iris.csv"               "presentation.html"     "pre
## [7] "presentation.R"         "presentation.Rmd"      "sta
```

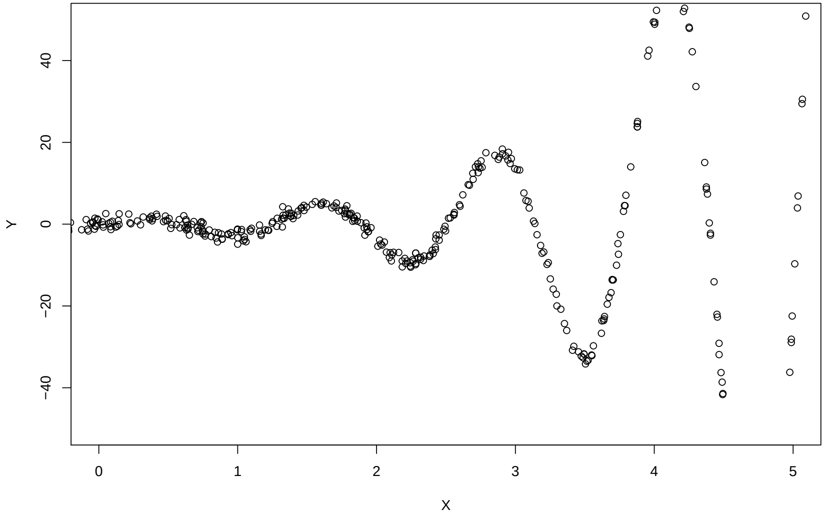
```r
iris <- read.csv("iris.csv")
```

Simulate some Data

```r
set.seed(1023) # Important for replication
X <- rnorm(1000,0,5)
Y <- sin(5*X)*exp(abs(X)) + rnorm(1000)
dat <- data.frame(X,Y)
```

Plot

```
plot(X,Y,xlim=c(0,5),ylim=c(-50,50))
```



Regression Output

```r
dat.lm<-lm(Y~X,data=dat)
dat.lm
```

```
##
## Call:
## lm(formula = Y ~ X, data = dat)
##
## Coefficients:
## (Intercept)            X
##     -216634       183687
```

Regression Output

```r
summary(dat.lm)
```

```
##
## Call:
## lm(formula = Y ~ X, data = dat)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -209776557    -418849     201212     816899    9079253
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -216634     212126  -1.021    0.307
## X             183687      43470   4.226  2.6e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
##
## Residual standard error: 6707000 on 998 degrees of freed
```

Pretty Output

- How do we get LaTeX output?
- The `xtable` package:

Pretty Output

- How do we get LaTeX output?
- The xtable package:

```r
require(xtable)
```

```
## Loading required package: xtable
```

```
## Warning: package 'xtable' was built under R version 3.2.
```

Pretty Output

```r
xtable(dat.lm)
```

```
## % latex table generated in R 3.2.2 by xtable 1.8-2 packa
## % Fri Jan 26 09:36:35 2018
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrr}
##   \hline
##  & Estimate & Std. Error & t value & Pr($>$$|$t$|$) \\
##   \hline
## (Intercept) & -216633.6722 & 212125.4622 & -1.02 & 0.307
##   X & 183687.1735 & 43469.5839 & 4.23 & 0.0000 \\
##    \hline
## \end{tabular}
## \end{table}
```

xtable

- `xtable` works on any sort of matrix

xtable

- xtable works on any sort of matrix

```r
xtable(A)
```

```
## % latex table generated in R 3.2.2 by xtable 1.8-2 packa
## % Fri Jan 26 09:36:35 2018
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrr}
##   \hline
##  & c & d \\
##   \hline
## a & 1.00 & 3.00 \\
##   b & 2.00 & 4.00 \\
##    \hline
## \end{tabular}
## \end{table}
```

Pretty it up

- Now let's make some changes to what `xtable` spits out:

Pretty it up

- Now let's make some changes to what `xtable` spits out:

```
print(xtable(dat.lm,digits=1),booktabs=TRUE)
```

```
## % latex table generated in R 3.2.2 by xtable 1.8-2 packa
## % Fri Jan 26 09:36:35 2018
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrr}
##   \toprule
##  & Estimate & Std. Error & t value & Pr($>$$|$t$|$) \\
##   \midrule
## (Intercept) & -216633.7 & 212125.5 & -1.0 & 0.3 \\
##   X & 183687.2 & 43469.6 & 4.2 & 0.0 \\
##    \bottomrule
## \end{tabular}
## \end{table}
```

apsrtable

- Read the documentation - there are many options.

```
require(apsrtable)
```

```
## Loading required package: apsrtable
```

```
dat.lm2 <- lm(Y~X+0,data=dat)
apsrtable(dat.lm,dat.lm2)
```

```
## \begin{table}[!ht]
## \caption{}
## \label{}
## \begin{tabular}{ l D{.}{.}{2}D{.}{.}{2} }
## \hline
##   & \multicolumn{ 1 }{ c }{ Model 1 } & \multicolumn{ 1
##  %               & Model 1       & Model 2      \\
## (Intercept)  & -216633.67   &               \\
##              & (212125.46)  &               \\
```

apsrtable

```r
library(png)
library(grid)
img <- readPNG("apsrtable.png")
grid.raster(img)
```

	Model 1	Model 2
(Intercept)	-216633.67	
	(212125.46)	
X	183687.17*	182921.71*
	(43469.58)	(43464.06)
N	1000	1000
R^2	0.02	0.02
adj. R^2	0.02	0.02
Resid. sd	6706998.86	6707143.05

stargazer

```r
require(stargazer)
```

```
## Loading required package: stargazer

##
## Please cite as:

##  Hlavac, Marek (2015). stargazer: Well-Formatted Regress

##  R package version 5.2. http://CRAN.R-project.org/packag
```

stargazer

```r
stargazer(dat.lm,dat.lm2)
```

```
##
## % Table created by stargazer v.5.2 by Marek Hlavac, Harv
## % Date and time: Fri, Jan 26, 2018 - 14:36:43
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lcc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{2}{c}{\textit{Dependent variable:}} \\
## \cline{2-3}
## \\[-1.8ex] & \multicolumn{2}{c}{Y} \\
## \\[-1.8ex] & (1) & (2)\\
## \hline \\[-1.8ex]
##  X & 183,687.200$^{***}$ & 182,921.700$^{***}$ \\
##   & (43,469.580) & (43,464.060) \\
```

stargazer

```r
img <- readPNG("stargazer.png")
grid.raster(img)
```

	Dependent variable:	
	Y	
	(1)	(2)
X	$183{,}687.200^{***}$	$182{,}921.700^{***}$
	(43,469.580)	(43,464.060)
Constant	−216,633.700	
	(212,125.500)	
Observations	1,000	1,000
R^2	0.018	0.017
Adjusted R^2	0.017	0.016
Residual Std. Error	6,706,999.000 (df = 998)	6,707,143.000 (df = 999)
F Statistic	17.856^{***} (df = 1; 998)	17.712^{***} (df = 1; 999)
Note:		*p<0.1; **p<0.05; ***p<0.01

Both

- Both packages are good (and can be supplemented with `xtable` when it is easier)
- Get pretty close to what you want with these packages, and then tweak the LaTeX directly.

Plotting

- It's all about coordinate pairs.
- `plot(x,y)` plots the pairs of points in `x` and `y`
- Notable options:
  - `type` - determines whether you plot points, lines or whatnot
  - `pch` - determines plotting character
  - `xlim` - x limits of the plot (likewise for `y`)
  - `xlab` - label on the x-axis
  - `main` - main plot label
  - `col` - color
  - A massive number of options. Read the docs.
- Some objects respond specially to `plot`. Try `plot(dat.lm)`

Homework

- Helpful functions: as.list(), sample()
- How to use column X of dataset ‘data’, data$X
- Nested for loops: first index by samples, then number of simulations