# 1. Basic R and Basic Concepts

Junlong Aaron Zhou (jlzhou@nyu.edu)

February 04, 2021

## About R

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

## 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)
library("sandwich")
require(sanwich)
sanwich::vcovHC
?vcovHC
```

## Install from other sources

- If you want to install packages on GitHub:

```
require(devtools)
install_github("wch/ggplot2")
```

- If you have a complied package downloaded on your computer (tar.gz):
- Tools -> Install Packages -> Find the location

- R cmd install *package name*

# Data types

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

# Character

```r
my.name <- "Aaron"
paste("My","name","is","Aaron")
```

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

```r
name.sentence <- paste0("My","name","is","Aaron")
as.character(99)
```

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

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

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

# Numeric

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

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

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

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

```r
as.numeric("99") + 1
```

```
## [1] 100
```

```r
pi
```

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

```r
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 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

```r
2 == 4
```

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

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

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

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

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

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

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

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

# Data Structures

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

# 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:

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

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

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

```
## [1] "2" "3" "a" "b" "c"
```

# More Vectors

- We can index vectors in several ways

```r
vec[1]
```

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

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

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

```r
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 <- seq(1, 7, 1)
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
```

```
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

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

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

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

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

```
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.

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

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

# Lists

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

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

```
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

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

```r
A <- matrix(c(1,2,3,4),nrow=2,ncol=2)
A
```

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

```r
A[1,2]
```

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

```r
A[1,]
```

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

```r
A[1:2,]
```

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

# Matrix Operations

- Its very easy to manipulate matrices:

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

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

```r
10*A
```

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

# 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.

```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.

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

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

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

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

- How to generate the OLS estimates with $X$ and $Y$?

# 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
```

# Array

- An array is similar to a matrix in many ways

```
array1 <- array(c(1,2,3,4,5,6,7,8), c(2,2,2))
array1
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

```
array1[,2,]
```

```
##      [,1] [,2]
## [1,]    3    7
## [2,]    4    8
```

# Dataframes

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

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

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

# Dataframes

- But you will generally work with your own datasets

```r
getwd()
```

```
## [1] "/Users/junlong/Dropbox/Teaching/2021_Spring_Quant_2/lab1"
```

```r
setwd("/Users/junlong/Dropbox/Teaching/2021_Spring_Quant_2/lab1")
```

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

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

# Dataframes

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

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

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

# Dataframes

```r
# add variables
dat.data[, "intercept"] <- rep(1, nrow(dat.data))
# change the name of a variable
names(dat.data)[6] <- "constant"
# delete variables
dat.data <- dat.data[, -6]
# sort on one variable
dat.data <- dat.data[order(dat.data[, "mpg"]), ]


# remove all missing values
dat.data.complete <- dat.data[complete.cases(dat.data), ]
# Or similarly
dat.dat.nona <- na.omit(dat.data)

dim(dat.data.complete)
```

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

```r
dim(dat.dat.nona)
```

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

```r
# select a subset
dat.data.subset <- dat.data[dat.data[, "make"] == "AMC", ]
dat.data.subset <- dat.data[1:3, ]
```

# Objects

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

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

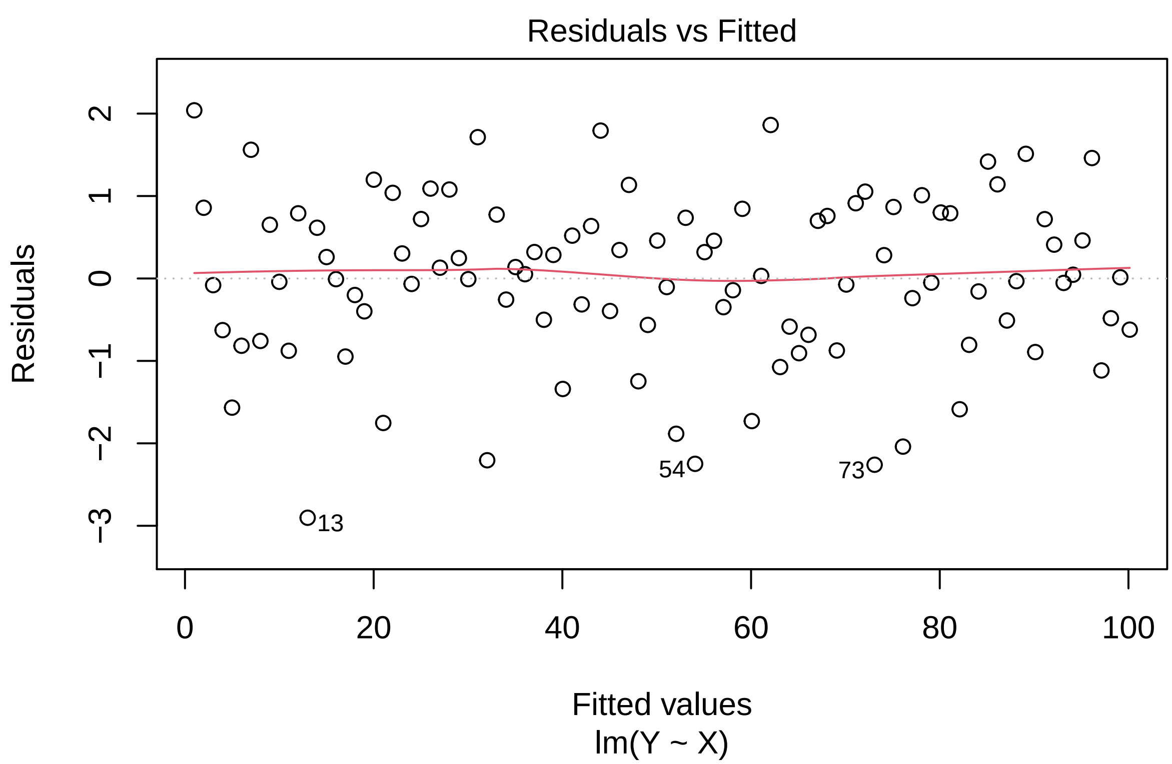
```
## [1] "lm"
```

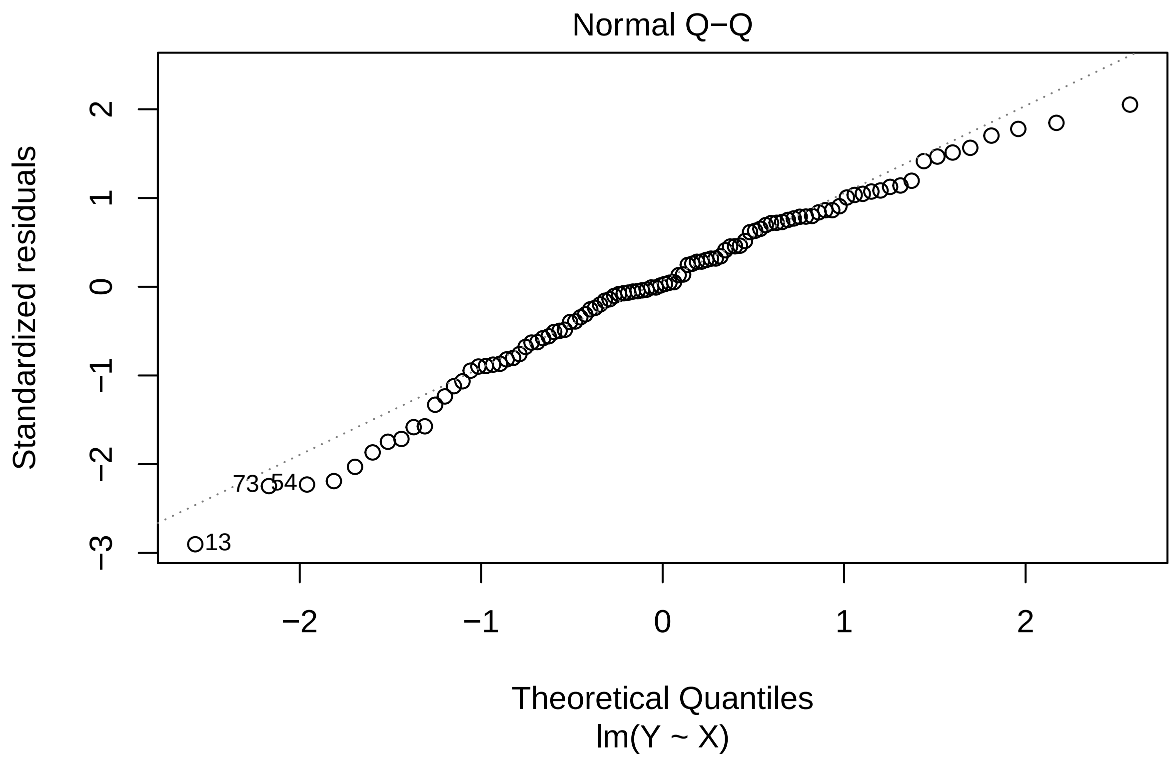
```r
predict(out.lm)
```

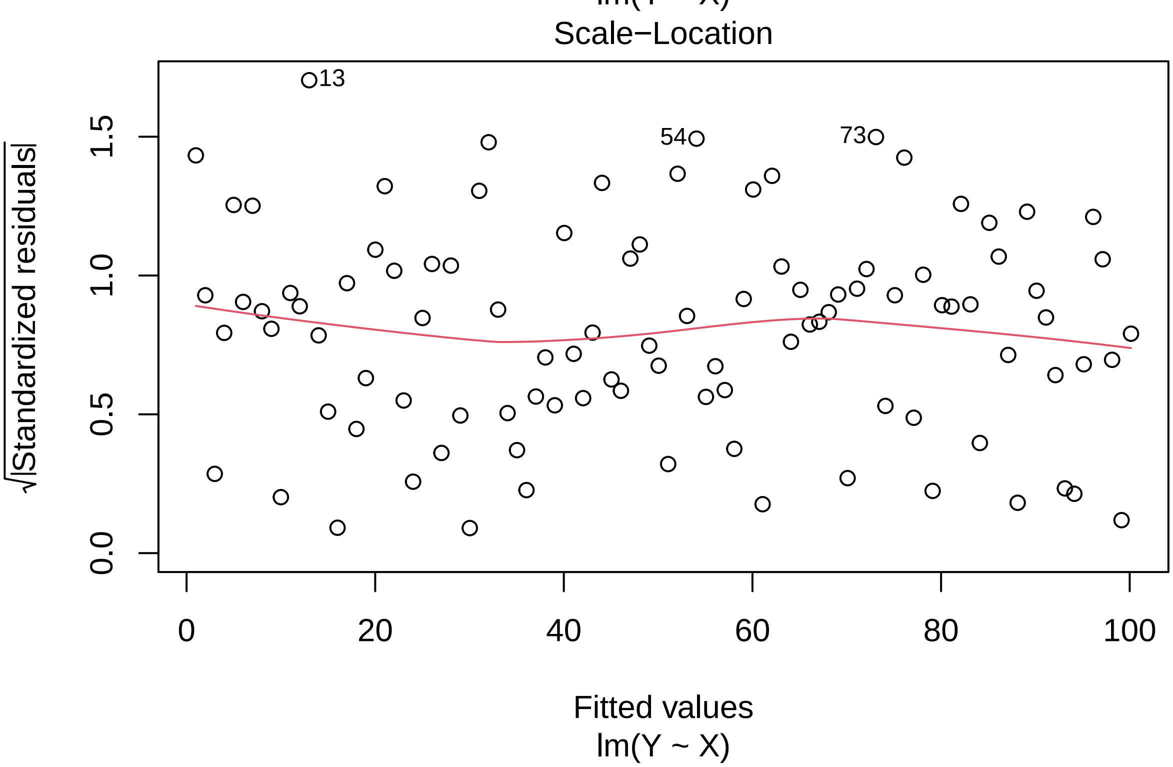
```
##          1          2          3          4          5          6
##  0.9836897  1.9852681  2.9868465  3.9884249  4.9900032  5.9915816
##          7          8          9         10         11         12
##  6.9931600  7.9947384  8.9963168  9.9978952 10.9994735 12.0010519
##         13         14         15         16         17         18
## 13.0026303 14.0042087 15.0057871 16.0073655 17.0089438 18.0105222
##         19         20         21         22         23         24
## 19.0121006 20.0136790 21.0152574 22.0168357 23.0184141 24.0199925
##         25         26         27         28         29         30
## 25.0215709 26.0231493 27.0247277 28.0263060 29.0278844 30.0294628
##         31         32         33         34         35         36
## 31.0310412 32.0326196 33.0341979 34.0357763 35.0373547 36.0389331
##         37         38         39         40         41         42
## 37.0405115 38.0420899 39.0436682 40.0452466 41.0468250 42.0484034
```

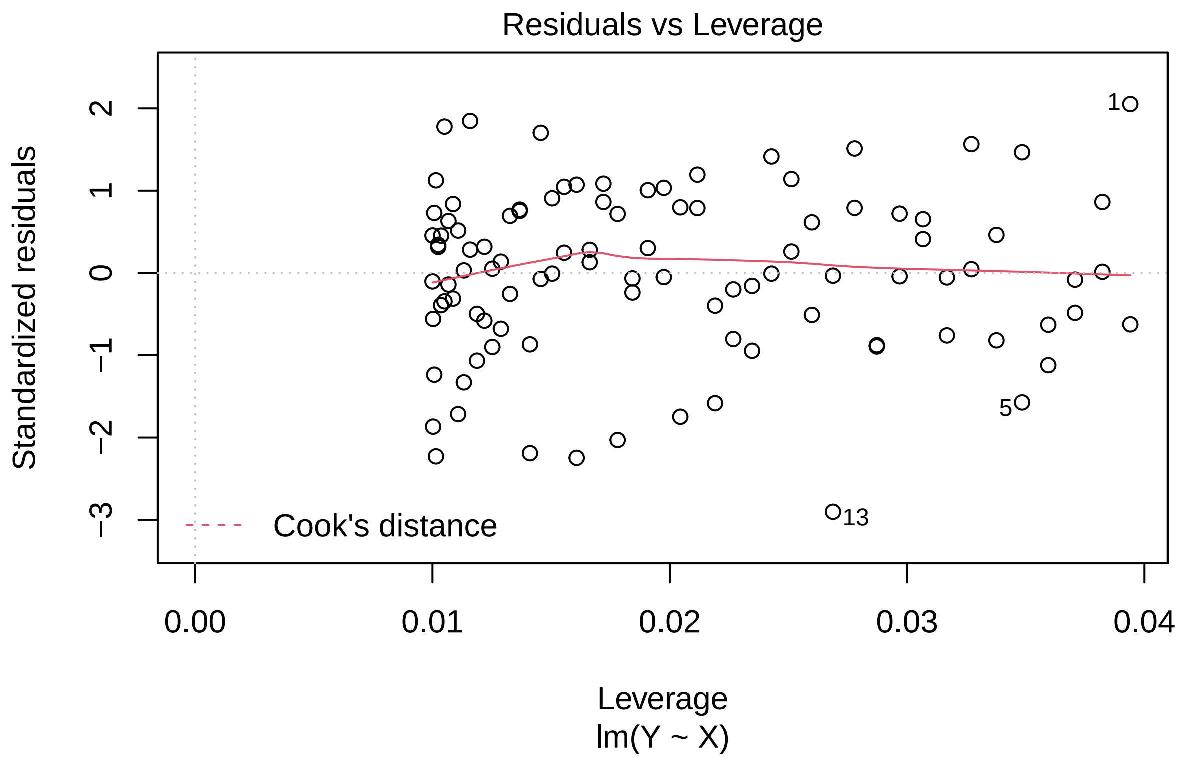
```
##         43         44         45         46         47         48
##  43.0499818  44.0515601  45.0531385  46.0547169  47.0562953  48.0578737
##         49         50         51         52         53         54
##  49.0594521  50.0610304  51.0626088  52.0641872  53.0657656  54.0673440
##         55         56         57         58         59         60
##  55.0689223  56.0705007  57.0720791  58.0736575  59.0752359  60.0768143
##         61         62         63         64         65         66
##  61.0783926  62.0799710  63.0815494  64.0831278  65.0847062  66.0862845
##         67         68         69         70         71         72
##  67.0878629  68.0894413  69.0910197  70.0925981  71.0941765  72.0957548
##         73         74         75         76         77         78
##  73.0973332  74.0989116  75.1004900  76.1020684  77.1036467  78.1052251
##         79         80         81         82         83         84
##  79.1068035  80.1083819  81.1099603  82.1115387  83.1131170  84.1146954
##         85         86         87         88         89         90
##  85.1162738  86.1178522  87.1194306  88.1210089  89.1225873  90.1241657
##         91         92         93         94         95         96
##  91.1257441  92.1273225  93.1289009  94.1304792  95.1320576  96.1336360
##         97         98         99        100
##  97.1352144  98.1367928  99.1383712 100.1399495
```

```r
plot(out.lm)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Y ~ X)
13
73
54
−2
−1
0
1
2
−3


Scale−Location
√|Standardized residuals|
Fitted values
lm(Y ~ X)
13
54
73
0.0
0.5
1.0
1.5
0
20
40
60
80
100




```r
summary(out.lm)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.90240 -0.59314  0.02267  0.74177  2.03962
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.017889   0.204307  -0.088     0.93
## X            1.001578   0.003512 285.157   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.014 on 98 degrees of freedom
## Multiple R-squared:  0.9988, Adjusted R-squared:  0.9988
## F-statistic: 8.131e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

- Objects can have other data embedded inside them

```r
out.lm$coefficients
```

```
## (Intercept)           X
## -0.01788866  1.00157838
```

Show results properly using stargazer.

```r
library(stargazer)
```

```
##
## Please cite as:
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
##  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
```

```r
stargazer(out.lm) #  This code gives you a latex code
```

```
##
## % Table created by stargazer v.5.2.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harva
## % Date and time: Thu, Feb 04, 2021 - 15:33:53
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{1}{c}{\textit{Dependent variable:}} \\
## \cline{2-2}
## \\[-1.8ex] & Y \\
## \hline \\[-1.8ex]
##  X & 1.002$^{***}$ \\
##   & (0.004) \\
##   & \\
##  Constant & $-$0.018 \\
##   & (0.204) \\
##   & \\
## \hline \\[-1.8ex]
## Observations & 100 \\
## R$^{2}$ & 0.999 \\
## Adjusted R$^{2}$ & 0.999 \\
## Residual Std. Error & 1.014 (df = 98) \\
## F Statistic & 81,314.570$^{***}$ (df = 1; 98) \\
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{1}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
## \end{tabular}
## \end{table}
```

```r
stargazer(out.lm, type = "text") # This gives you a table in text
```

```
##
## ================================================
##                          Dependent variable:
##                      ---------------------------
##                                   Y
## ------------------------------------------------
## X                              1.002***
##                                 (0.004)
##
## Constant                        -0.018
##                                 (0.204)
```

```
##
## -----------------------------------------------
## Observations                       100
## R2                               0.999
## Adjusted R2                      0.999
## Residual Std. Error       1.014 (df = 98)
## F Statistic         81,314.570*** (df = 1; 98)
## ===============================================
## Note:               *p<0.1; **p<0.05; ***p<0.01
```

You can always includes latex code directly.

Table 1:

| | *Dependent variable:* |
|---|---|
| | Y |
| X | $1.004^{***}$ |
| | (0.003) |
| | |
| Constant | $-0.172$ |
| | (0.200) |
| | |
| Observations | 100 |
| $R^2$ | 0.999 |
| Adjusted $R^2$ | 0.999 |
| Residual Std. Error | 0.991 (df = 98) |
| F Statistic | $85{,}529.470^{***}$ (df = 1; 98) |
| *Note:* | $^{*}$p<0.1; $^{**}$p<0.05; $^{***}$p<0.01 |

# Control Flow

- loops
- if/else

# Loops

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

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

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

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

```
## NA  b  c
```

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

```
## NA  b  c
```

# If/Else

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

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

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

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

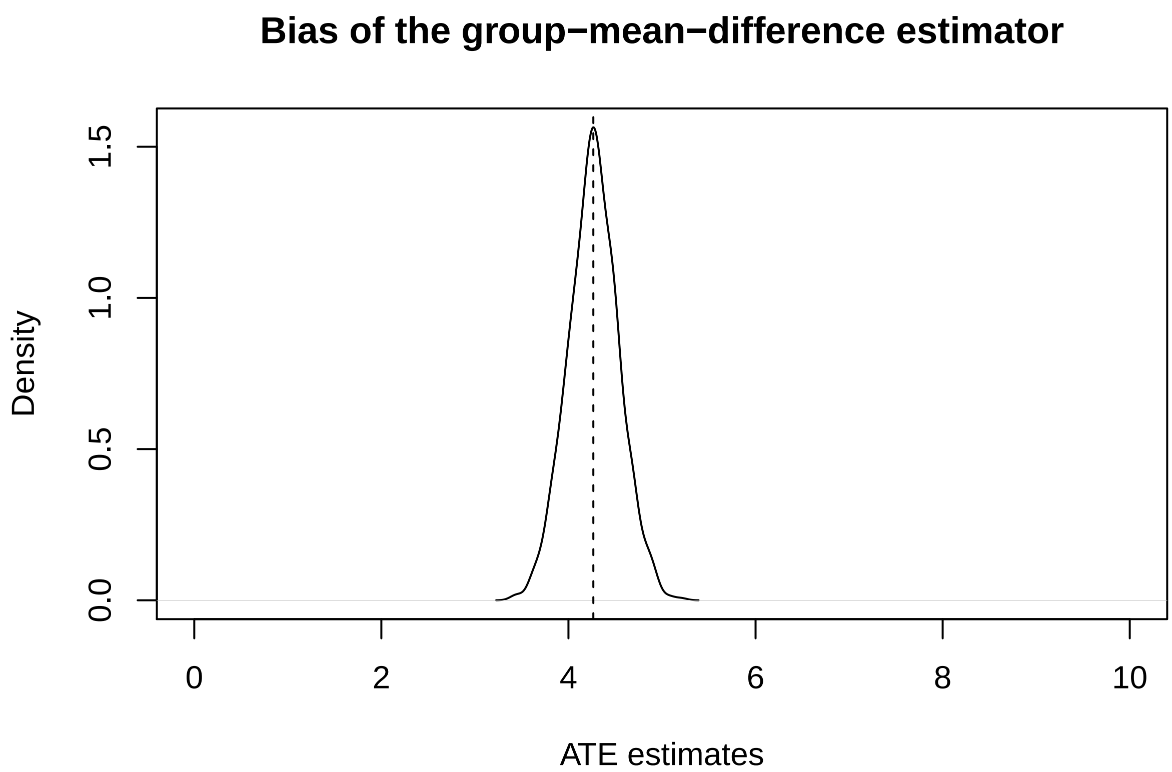
# Basic concepts

- When do we want ATT rather than ATE?
- What does conditional independence mean?
- What do we mean by "control?"
- When we have the population, does regression estimate still have uncertainty?
- Lesson One: Whenever you don't understand something, simulate!

# A toy example

```
N_pop <- 1000
X1 <- rnorm(N_pop, 3, 1)
Y0 <- abs(rnorm(N_pop, 5, 2))
Y1 <- Y0 + rnorm(N_pop, 0, 5) + 4 # PATE = 4 by construction
TE <- Y1 - Y0
ATE <- mean(TE)
```

```
nboots <- 1000
ATE_est <- ATT <- rep(NA, nboots)
for (i in 1:nboots){
  D <- rep(0, N_pop)
  D[sample(N_pop, 300)] <- 1
  Y <- D*Y1 + (1-D)*Y0
  data.pop <- data.frame(Y=Y, D=D)
  ATT[i] <- mean(TE[D==1])
  ATE_est[i] <- mean(Y[D==1]) - mean(Y[D==0])
}
plot(density(ATE_est), main = "Bias of the group-mean-difference estimator", xlab = "ATE estimates", xli
abline(v = ATE, lty = 2)
```



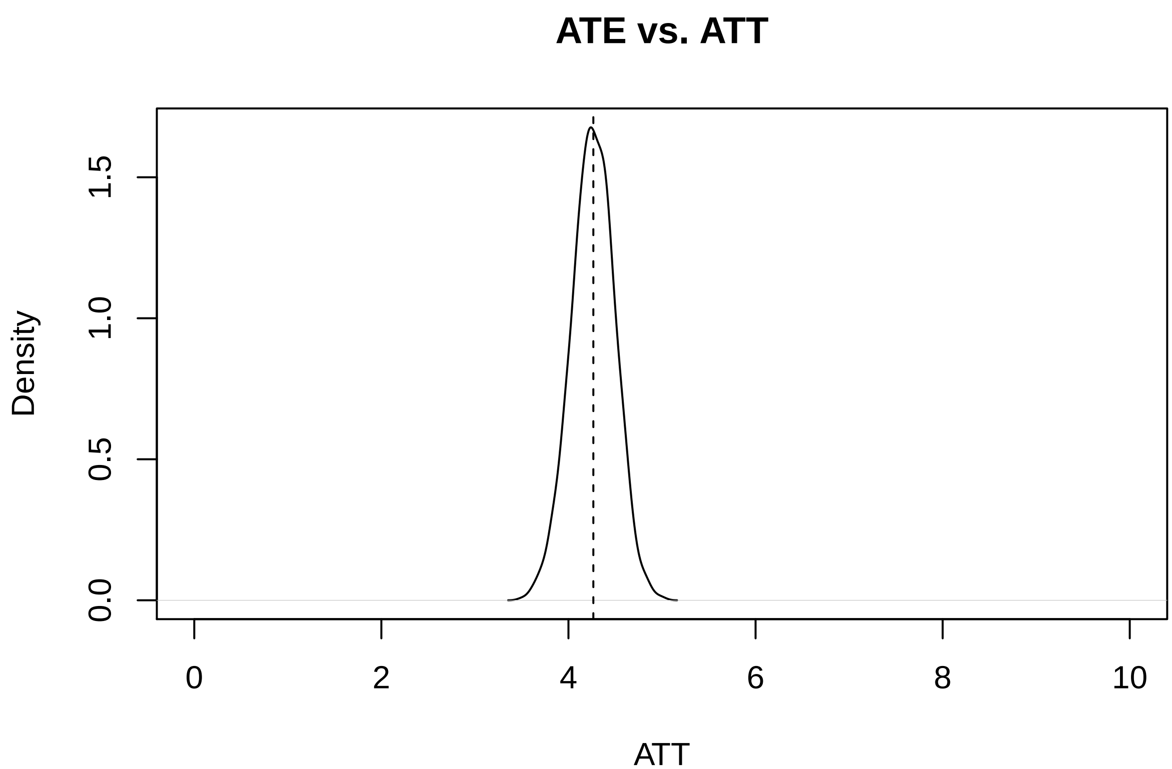

```r
cat("The true ATE is", ATE, ", and the average of ATE estimates is", mean(ATE_est), "\n")
```

```
## The true ATE is 4.266077 , and the average of ATE estimates is 4.268728
```

```r
cat("The sampling variance of ATE estimates is", var(ATE_est), "\n")
```

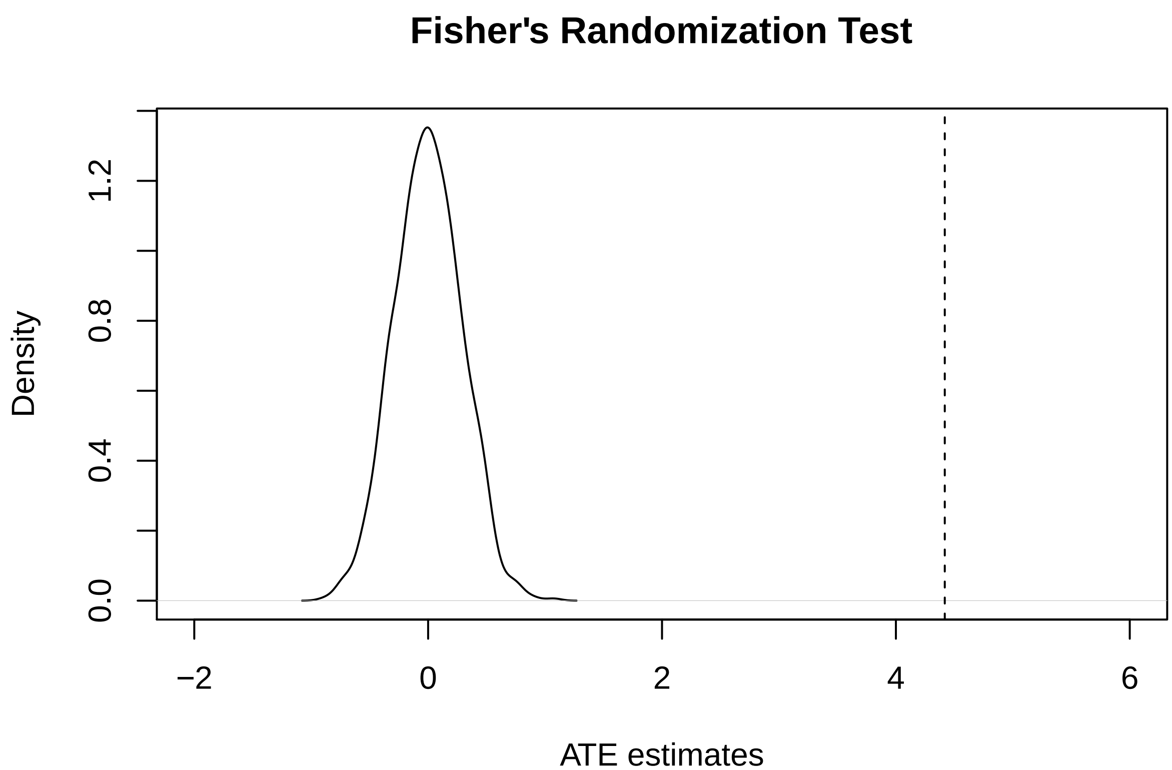
```
## The sampling variance of ATE estimates is 0.07252279
```

```r
plot(density(ATT), main = "ATE vs. ATT", xlab = "ATT", xlim = c(0, 10))
abline(v = ATE, lty = 2)
```



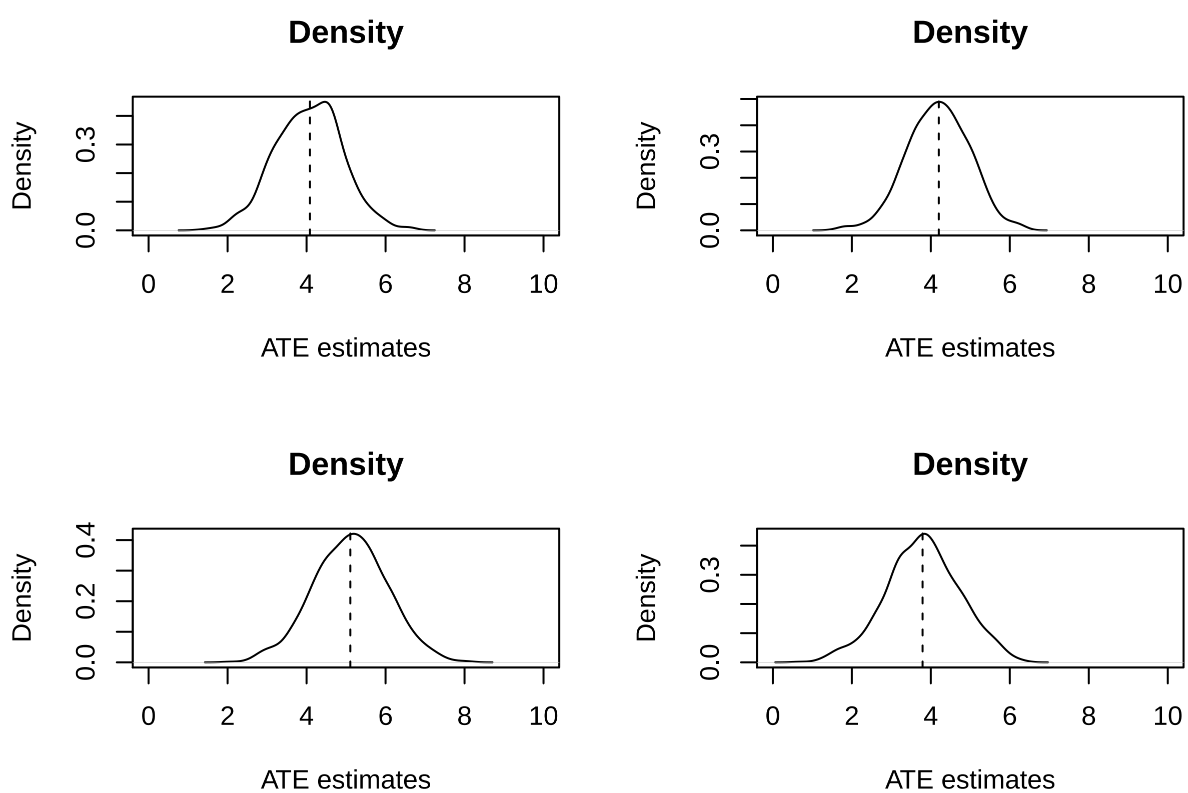

```
nboots <- 1000
D <- rep(0, N_pop)
D[sample(N_pop, 500)] <- 1
Y <- D*Y1 + (1-D)*Y0
ATE_original <- mean(Y[D==1]) - mean(Y[D==0])

ATE_boot <- rep(NA, nboots)
for (i in 1:nboots){
  D_boot <- rep(0, N_pop)
  D_boot[sample(N_pop, 500)] <- 1
  ATE_boot[i] <- mean(Y[D_boot==1]) - mean(Y[D_boot==0])
}
plot(density(ATE_boot), main = "Fisher's Randomization Test", xlab = "ATE estimates", xlim = c(-2, 6))
abline(v = ATE_original, lty = 2)
```




```
cat("The 95% confidence interval under the null distribution is", quantile(ATE_boot, c(0.025, 0.975)),
```

```
## The 95% confidence interval under the null distribution is -0.558905 0.5380508
```




```
## The total variance of ATE estimates is 1.016305
```

```
## The variance of ATE estimates due to design is 0.7718686
## The variance of ATE estimates due to sampling is 0.326606
## The average variance estimates is 1.006966
```