

Basic R and Basic Concepts

Giacomo Lemoli*(gl1759@nyu.edu)

January 31, 2022

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")
```

*This note builds on and revises material created by Aaron Zhou

- 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

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

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

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

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

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

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

Numeric

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

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

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

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

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

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

```
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 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

```
2 == 4
```

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

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

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

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

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

```
"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.

```
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)
```

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

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

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

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

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

```
vector2 <- seq(1, 7, 1)  
vector2
```

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

Creating Vectors

```
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

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

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

```
sum(a)
```

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

```
max(a)
```

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

```
min(a)
```

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

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}
- $A_{1,2} = 3$
- $A_{1,.} = (1, 3)$

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

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

```
## [2,] 2 4
```

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

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

```
A[1,]
```

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

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

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

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

```
## [2,] 2 4
```

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
```

```
A%*%B
```

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

```
## [1,] 1 6
```

```
## [2,] 2 8
```

More Matrix Ops.

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

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

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

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

```
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

```
# 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

```
getwd()
```

```
## [1] "C:/Users/giacco/Dropbox/NYU/TA Work/Quant 2 Spring 2022/Lab material/lab1"
setwd("C:/Users/giacco/Dropbox/NYU/TA Work/Quant 2 Spring 2022/Lab material/lab1")
```

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

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

Dataframes

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

```
## Caricamento del pacchetto richiesto: foreign
dat.data <- read.dta("https://stats.idre.ucla.edu/stat/data/test.dta")
```

Dataframes

```
# 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
```

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

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

```
# 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.

```
# Remember to always set a seed when generating random numbers
set.seed(1)

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

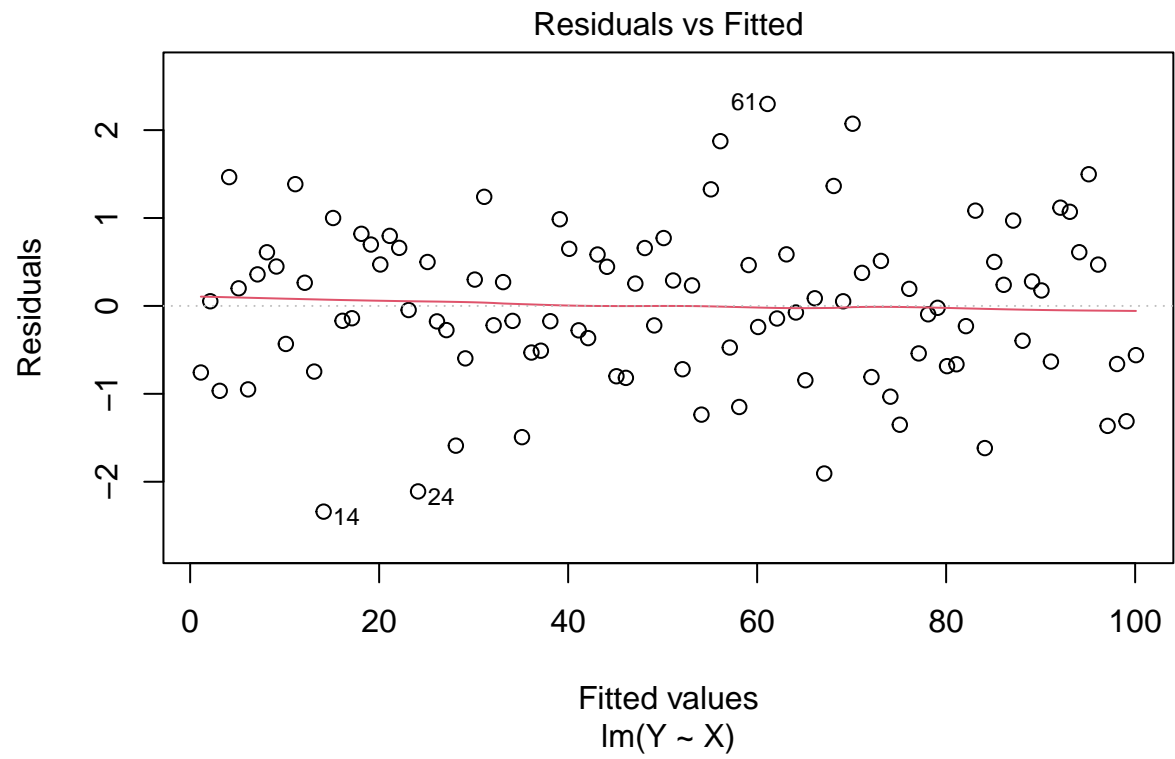
```
## [1] "lm"
```

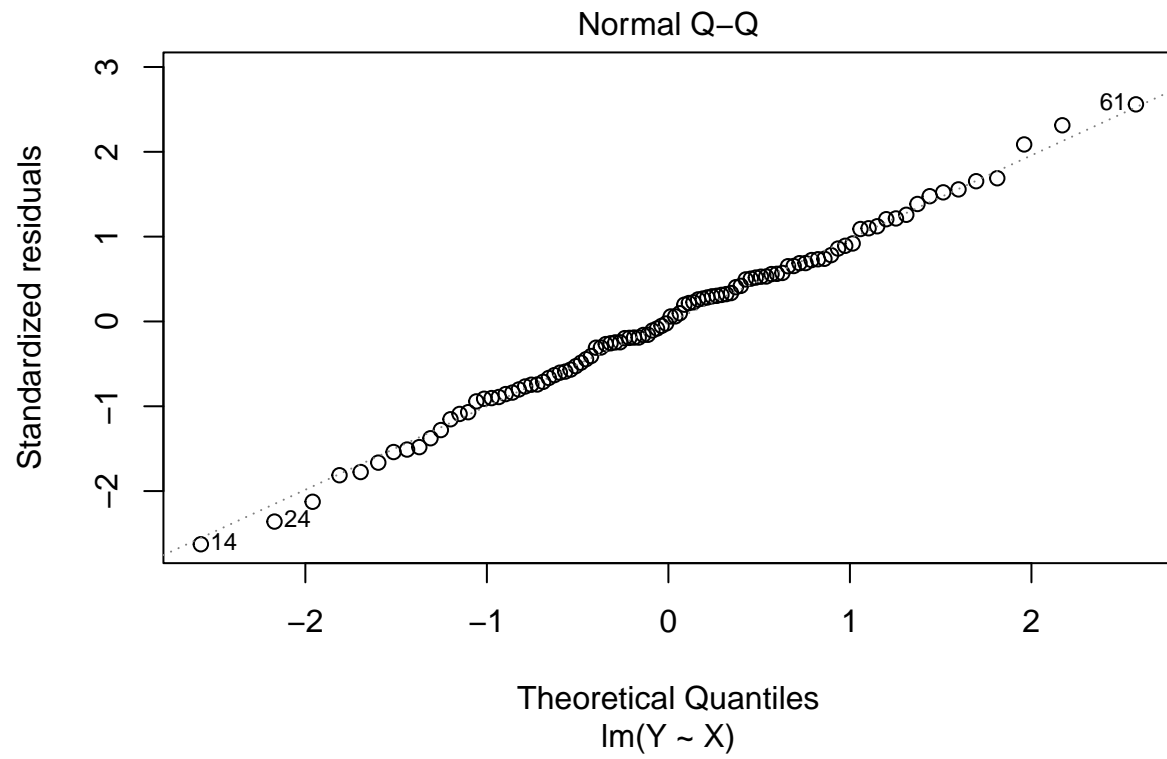
```
predict(out.lm)
```

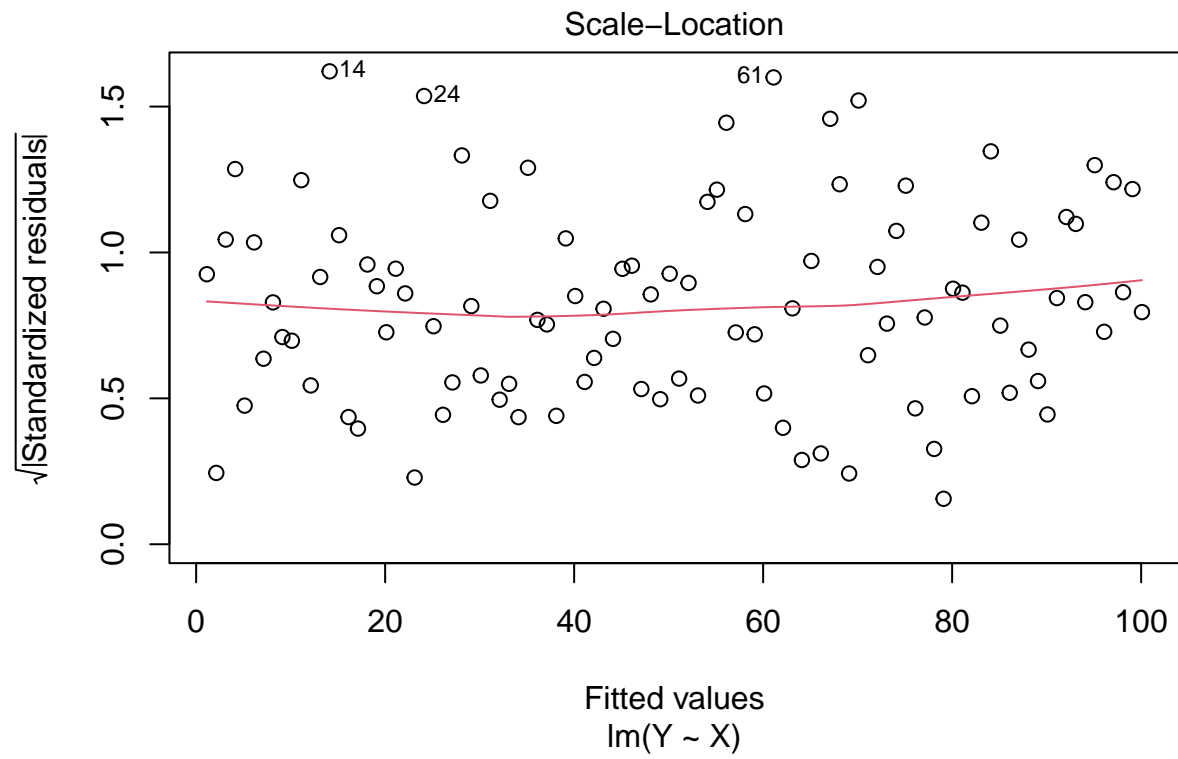
```
##          1          2          3          4          5          6          7
##  1.131215  2.130764  3.130313  4.129862  5.129410  6.128959  7.128508
##          8          9         10         11         12         13         14
##  8.128057  9.127606 10.127155 11.126704 12.126253 13.125802 14.125351
##         15         16         17         18         19         20         21
```

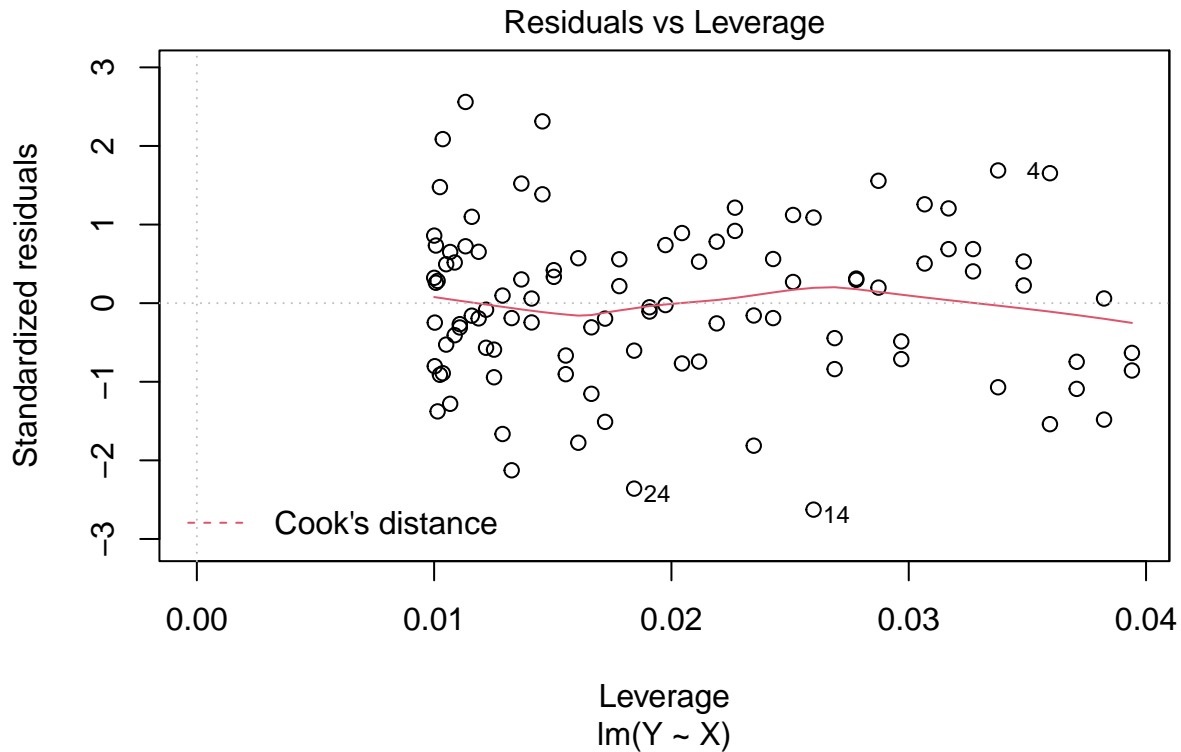
```
## 15.124900 16.124449 17.123998 18.123547 19.123096 20.122645 21.122194
##          22          23          24          25          26          27          28
## 22.121742 23.121291 24.120840 25.120389 26.119938 27.119487 28.119036
##          29          30          31          32          33          34          35
## 29.118585 30.118134 31.117683 32.117232 33.116781 34.116330 35.115879
##          36          37          38          39          40          41          42
## 36.115428 37.114977 38.114526 39.114075 40.113623 41.113172 42.112721
##          43          44          45          46          47          48          49
## 43.112270 44.111819 45.111368 46.110917 47.110466 48.110015 49.109564
##          50          51          52          53          54          55          56
## 50.109113 51.108662 52.108211 53.107760 54.107309 55.106858 56.106407
##          57          58          59          60          61          62          63
## 57.105955 58.105504 59.105053 60.104602 61.104151 62.103700 63.103249
##          64          65          66          67          68          69          70
## 64.102798 65.102347 66.101896 67.101445 68.100994 69.100543 70.100092
##          71          72          73          74          75          76          77
## 71.099641 72.099190 73.098739 74.098288 75.097836 76.097385 77.096934
##          78          79          80          81          82          83          84
## 78.096483 79.096032 80.095581 81.095130 82.094679 83.094228 84.093777
##          85          86          87          88          89          90          91
## 85.093326 86.092875 87.092424 88.091973 89.091522 90.091071 91.090620
##          92          93          94          95          96          97          98
## 92.090169 93.089717 94.089266 95.088815 96.088364 97.087913 98.087462
##          99         100
## 99.087011 100.086560
```

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









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

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
```

	Min	1Q	Median	3Q	Max
	-2.34005	-0.60584	0.01551	0.58514	2.29747

```
##
## Coefficients:
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.131666	0.181897	0.724	0.471
X	0.999549	0.003127	319.640	<2e-16 ***

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9027 on 98 degrees of freedom
## Multiple R-squared:  0.999, Adjusted R-squared:  0.999
## F-statistic: 1.022e+05 on 1 and 98 DF, p-value: < 2.2e-16
```

- Objects can have other data embedded inside them

```
out.lm$coefficients
```

```
## (Intercept)          X
##  0.1316657    0.9995489
```

Show results properly using stargazer.

```
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
```

```
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.harvard.edu
```

```
## % Date and time: lun, gen 31, 2022 - 00:14:06
```

```
## \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 & 1.000 $\text{\textasciitimes}$  \\\
```

```
## & (0.003) \\\
```

```
## & \\\
```

```
## Constant & 0.132 \\\
```

```
## & (0.182) \\\
```

```
## & \\\
```

```
## \hline \[-1.8ex\]
```

```
## Observations & 100 \\\
```

```
## R $\text{\textasciitimes}$ 2 & 0.999 \\\
```

```
## Adjusted R $\text{\textasciitimes}$ 2 & 0.999 \\\
```

```
## Residual Std. Error & 0.903 (df = 98) \\\
```

```
## F Statistic & 102,169.400 $\text{\textasciitimes}$  (df = 1; 98) \\\
```

```
## \hline
```

```
## \hline \[-1.8ex\]
```

```
## \textit{Note:} & \multicolumn{1}{r}{ $\text{\textasciitimes}$  p < 0.1;  $\text{\textasciitimes}$  p < 0.05;  $\text{\textasciitimes}$  p < 0.01} \\\
```

```
## \end{tabular}
```

```
## \end{table}
```

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

```
##
```

```
## =====
```

```
## Dependent variable:
```

```
## -----
```

```
## Y
```

```
## -----
```

```
## X 1.000***
```

```
## (0.003)
```

```
##
```

```
## Constant 0.132
```



```
##                                (0.182)
##
## -----
## Observations                100
## R2                          0.999
## Adjusted R2                 0.999
## Residual Std. Error        0.903 (df = 98)
## F Statistic                102,169.400*** (df = 1; 98)
## =====
## Note:                       *p<0.1; **p<0.05; ***p<0.01
```

You can always includes latex code directly.

Table 1:	
	<i>Dependent variable:</i>
	Y
X	1.000*** (0.003)
Constant	0.132 (0.182)
Observations	100
R ²	0.999
Adjusted R ²	0.999
Residual Std. Error	0.903 (df = 98)
F Statistic	102,169.400*** (df = 1; 98)
<i>Note:</i>	*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

```
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, " ")
}
```

```
## NA b c
```

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"
```

```
for(i in 2:length(vec)){
  if(vec[i]=="b") {
    print("Hello World!")
  }
  else {
    print("!dlroW olleH")
  }
}
```

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

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

Functions

- Function perform a set of operations internally and return only the output we want
- They are especially useful when we want to reiterate the same actions without copying the same lines of code forever

```
# Simple function to estimate descriptive statistics
```

```
descr <- function(var){
  mean <- mean(var, na.rm=T)
  sd <- sd(var, na.rm=T)
  N <- length(var)
  return(round(c(mean, sd, N), 2))
}
```

```
# Apply it to our data
```

```
apply(dat.data[,c("mpg", "weight", "price")], 2, descr)
```

```
##      mpg  weight  price
## [1,] 19.20 3250.00 5058.00
## [2,]  3.11  541.62 1606.72
## [3,]  5.00    5.00   5.00
```

Since data frames are R objects, we can code our own estimators (one of the reasons people like R).

```
# Function to compute OLS coefficients
ols <- function(X, y) {
  # Input: vector or matrix X, vector y
  # Returns: coefficient vector for OLS regression  $(X'X)^{-1}X'y$ 

  # Create a column for the intercept
  if ( !all(as.matrix(X)[,1] == 1) ) {X <- cbind(1, X)}
  out <- solve(t(X) %*% X) %*% t(X) %*% y
  rownames(out) <- c('Intercept', rep('', nrow(out)-1 ))
  return( t(out) )
}

# Let's estimate regression coefficients by OLS
round(ols(X, Y), 4)
```

```
##      Intercept
## [1,]  0.1317 0.9995
```

```
lm(Y ~ X)
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Coefficients:
## (Intercept)          X
##      0.1317      0.9995
```

Simulations

- A very important tool
- Theoretical properties of estimators can be better understood by observing them in simulated data
- For empirical researchers, simulating the DGP allows to study properties of research designs such as power (especially important when designing an experiment or study)
- So, if you don't understand something, simulation is a valid option

An example

Recall the Neyman estimator of the sampling variance of the difference in means:

$$\hat{V} = \frac{\hat{s}_{Y_1}^2}{n_1} + \frac{\hat{s}_{Y_0}^2}{n_0}$$

We have seen that, under random sampling from a super-population, this estimator is unbiased for the sampling variance of the difference in means given by both random sample variation **and** randomization within sample (Imbens and Rubin, Ch.6).

$$E[\hat{V}] = \frac{\sigma_{Y_1}^2}{n_1} + \frac{\sigma_{Y_0}^2}{n_0} = V[\bar{Y}_1 - \bar{Y}_0]$$

Moreover, if there is treatment effect heterogeneity (i.e. the unit-level treatment effects are not constant), it is a (upward) biased estimator of the sampling variance of difference in means given by randomization only, for a fixed sample (Imbens and Rubin Ch.6).

$$E_D[\hat{V}|S] = V_D[\bar{Y}_1 - \bar{Y}_0|S] + \frac{s_p^2}{n} \geq V_D[\bar{Y}_1 - \bar{Y}_0|S]$$

Let's observe these properties through simulation.

```
# Set seed
set.seed(123)

# Assume a large super population
N_pop <- 100000

# We simulate the potential outcomes for each observation
Y0 <- abs(rnorm(N_pop, mean = 5, sd = 2))
Y1 <- Y0 + rnorm(N_pop, 0, 5) + 4

# Note that the PATE is ~ 4 by construction
TE <- Y1 - Y0
(PATE <- mean(TE))

## [1] 4.026077

# Now, we extract a random sample from this super-population
# Say, our experimental sample
Nsample <- 1000
pop <- data.frame(Y0 = Y0, Y1 = Y1, TE = TE)
sample <- pop[sample(nrow(pop), size = Nsample),]
```

```
# What is the SATE?
(SATE <- mean(sample$TE))
```

```
## [1] 3.910847
```

```
# Now, we can simulate the randomization distribution over this sample.
# In practice, we re-assign the treatment N times and at each iteration we compute an estimate
# for the SATE using the new **observed** values
```

```
# Number of iterations
Nboot <- 10000
```

```
# An empty vector where to store the estimates
dim <- vars <- rep(NA, Nboot)
```

```
# Start loop
for(i in 1:Nboot){
  # Treatment assignment to half units (complete randomization)
  sample$D <- 0
  sample$D[sample(Nsample, Nsample/2)] <- 1

  # Observed potential outcomes
  sample$Y <- sample$D*sample$Y1 + (1-sample$D)*sample$Y0

  # Compute simple difference in means (estimate for SATE) and store it
  dim[i] <- mean(sample$Y[sample$D==1]) - mean(sample$Y[sample$D==0])

  # Compute variance (estimate for V(SATE)) and store it
  vars[i] <- var(sample$Y[sample$D==1])/(Nsample/2) + var(sample$Y[sample$D==0])/(Nsample/2)
}
```

```
# We now have a simulated randomization distribution of differences in means
# We know this is unbiased for the SATE
mean(dim)
```

```
## [1] 3.905932
```

```
# What is the variance of this estimator in the randomization distribution?
var(dim)
```

```
## [1] 0.03920887
```

```
# What is the expected value of the variance estimator we computed?
mean(vars)
```

```
## [1] 0.06285642
```

```
# Finally, let's compute the sampling variance of DIM under both randomization and sampling distribution
# Here the simulation has two levels:
```

```
# (i) we do an outer loop where we randomly draw samples from the super population
# (ii) we do an inner loop where for each sample we compute the randomization distribution, as before
```

```
# Number of sample draws to do
Nsampling <- 20
```

```

# Matrices and vectors where to store the results
dim <- vars <- matrix(NA, Nsampling, Nboot)

# Begin loop
for(j in 1:Nsampling){
  # New random sample
  sample <- pop[sample(nrow(pop), size = Nsample),]

  # Inner loop: randomization distribution
  for(i in 1:Nboot){
    # Treatment assignment to half units (complete randomization)
    sample$D <- 0
    sample$D[sample(Nsample, Nsample/2)] <- 1

    # Observed potential outcomes
    sample$Y <- sample$D*sample$Y1 + (1-sample$D)*sample$Y0

    # Compute simple difference in means (estimate for SATE) and store it
    dim[j,i] <- mean(sample$Y[sample$D==1]) - mean(sample$Y[sample$D==0])

    # Compute variance (estimate for V(SATE)) and store it
    vars[j,i] <- var(sample$Y[sample$D==1])/(Nsample/2) + var(sample$Y[sample$D==0])/(Nsample/2)
  }
}

# We know that the DIM is unbiased also for the PATE
mean(dim)

## [1] 4.059777

# What is the variance of this estimator in the sampling and randomization distribution?
var(as.vector(dim))

## [1] 0.06019168

# Note that we can also compute the "true" (theoretical) value
# using the population values
var(Y1)/(Nsample/2) + var(Y0)/(Nsample/2)

## [1] 0.06595247

# What is the expected value of the Neyman variance estimator?
mean(vars)

## [1] 0.06691493

```

Note that under sampling and randomization distribution the variance of the DIM estimator is larger! In fact, we have two sources of uncertainty (or noise) to account for. But we are able to estimate it without bias using sample quantities.