Lecture Note for Applied Econometrics

Yuta Toyama

Last updated: 2019/05/16

Contents

Chapter 1

Preface

Welcome to Applied Econometrics using R!

1.1 About this

This lecture note is maintained by Yuta Toyama.

1.2 Update: April 23, 2019

- Upload chapter 6 (review of statistics)

1.3 Update: April 16, 2019

- Update chapter 3 (basic programming)
- Upload chapter 4 (Data frame)
- Upload chapter 5 (Exercise 1)

1.4 Acknowledgement (as of April 16, 2019)

- Chapter 2 through 4 are largely based on Applied Statistics with `R`. https://daviddalpiaz.github.io/appliedstats/
- Chapter 6 is based on "Introduction to Econometrics with R". https://www.econometrics-with-r.org/index.html

Chapter 2

Introduction to the course

2.1 What is econometrics?

1. Estimating economic relationships
    1. Demand curve $\log(Q_t) = \alpha_0 + \alpha_1 P_t + \epsilon_t$
    2. Production function $Y_{it} = A_{it} K_{it}^{\alpha} L_{it}^{\beta}$
2. Testing economic theory
    - Does adverse selection exists in insurance markets?
    - Are consumers rational?
3. Determine the effect of a given intervention (causal inference)
    - What is the effect of increasing minimum wage on employment?
    - Do mergers increase the output price?
    - Does democracy cause economic growth? (a series of works by Acemoglu, Robinsohn, and their co-authors).
    - Effects of going to private colleges on your future earnings.
    - Note: Some questions may have underlying economic models, others may not.
4. Describe the data (prediction/forecasting)
    - How does the distribution of wage look like?
    - Relationship between electricity consumption and temperature (possibly nonlinear).
    - Related to machine learning (ML).

2.2 Why do we need to learn computation

1. Conduct statistical and empirical analysis using your own data set
    1. Construct the data set
    2. Describe the data
    3. Run regression or estimate an economic object
    4. Make tables and figures that show the results of your analysis.
2. Verify the econometric theory through numerical simulations.
    - Ex. Asymptotic theory considers the case when the sample size is large enough (i.e., $N \to \infty$)
        - Law of large numbers, central limit theorem
        - How well is the asymptotic approximation?
    - **Monte Carlo simulations**

- We will learn both aspects in this course.

2.3 Why do we use R?

- Many alternatives: Stata, Matlab, Python, etc…

1. Free software!!
   - Stata and Matlab are expensive.
   - Though you can use Matlab through the campus license from this April.
2. Good balance between flexibility in programming and easy-to-use for econometric analysis
   - Stata is easy to use for econometric analysis, but hard to write your own program.
   - Matlab is the opposite.
   - You can do everything with R, including data construction, regression analysis, and complicated structural estimation.
3. Many users
   - Popular in engineering.
   - Many packages being developed (especially important for recently popular tools.)

- Note: Python seems also good, though I have not used it before.

Chapter 3

Introduction of `R` and `R studio`

3.1 Getting Started

- You can use R/R studio in the PC room.
- However, I strongly recommend you install R/Studio in your laptop and bring it to the class.
- Install in the following order
  1. R: https://www.r-project.org/
  2. Rstudio: https://www.rstudio.com/
- Now open Rstudio.

3.2 Helps

- The RStudio team has developed a number of "cheatsheets" for working with both `R` and RStudio.
- This particular cheatsheet for Base `R` will summarize many of the concepts in this document.

3.3 Quick tour of Rstudio

- There are four panels
  1. Source: Write your own code here.
  2. Console:
  3. Environment/History:
  4. Files/Plots/Packages/Help:
- In the Source panel,
  - Write your own code.
  - Save your code in `.R` file
  - Click `Run` command to run your entire code.
- In the concole panel,
  - After clicking `Run` in the source panel, your code is evaluated.
  - You can directly type your code here to implement.

3.4 Basic Calculations

To get started, we'll use `R` like a simple calculator.

Addition, Subtraction, Multiplication and Division

Math	R	Result
$3+2$	`3 + 2`	5
$3-2$	`3 - 2`	1
$3 \cdot 2$	`3 * 2`	6
$3/2$	`3 / 2`	1.5

Exponents

Math	R	Result
3^2	`3 ^ 2`	9
$2^{(-3)}$	`2 ^ (-3)`	0.125
$100^{1/2}$	`100 ^ (1 / 2)`	10
$\sqrt{100}$	`sqrt(100)`	10

Mathematical Constants

Math	R	Result
π	`pi`	3.1415927
e	`exp(1)`	2.7182818

Logarithms

- Note that we will use ln and log interchangeably to mean the natural logarithm.
- There is no `ln()` in R, instead it uses `log()` to mean the natural logarithm.

Math	R	Result
$\log(e)$	`log(exp(1))`	1
$\log_{10}(1000)$	`log10(1000)`	3
$\log_2(8)$	`log2(8)`	3
$\log_4(16)$	`log(16, base = 4)`	2

Trigonometry

Math	R	Result
$\sin(\pi/2)$	`sin(pi / 2)`	1
$\cos(0)$	`cos(0)`	1

3.5 Getting Help

- In using R as a calculator, we have seen a number of functions: `sqrt()`, `exp()`, `log()` and `sin()`.
- To get documentation about a function in R, simply put a question mark in front of the function name and RStudio will display the documentation, for example:

```
?log
?sin
?paste
?lm
```

3.6 Installing Packages

- One of the main strengths of `R` as an open-source project is its package system.
- To install a package, use the `install.packages()` function.
  - Think of this as buying a recipe book from the store, bringing it home, and putting it on your shelf.

```
install.packages("ggplot2")
```

- Once a package is installed, it must be loaded into your current `R` session before being used.
  - Think of this as taking the book off of the shelf and opening it up to read.

```
library(ggplot2)
```

- Once you close `R`, all the packages are closed and put back on the imaginary shelf.
- The next time you open `R`, you do not have to install the package again, but you do have to load any packages you intend to use by invoking `library()`.

Chapter 4

Data and Programming

4.1 Data Types

`R` has a number of basic data *types*.

- Numeric
  - Also known as Double. The default type when dealing with numbers.
  - Examples: `1`, `1.0`, `42.5`
- Logical
  - Two possible values: `TRUE` and `FALSE`
  - You can also use `T` and `F`, but this is *not* recommended.
  - `NA` is also considered logical.
- Character
  - Examples: `"a"`, `"Statistics"`, `"1 plus 2."`

4.2 Data Structures

- `R` also has a number of basic data *structures*.
- A data structure is either
  - homogeneous (all elements are of the same data type)
  - heterogeneous (elements can be of more than one data type).

Dimension	**Homogeneous**	**Heterogeneous**
1	Vector	List
2	Matrix	Data Frame
3+	Array	

4.3 Vectors

4.3.1 Basics of vectors

- Many operations in `R` make heavy use of **vectors**.
  - Vectors in `R` are indexed starting at `1`.
- The most common way to create a vector in `R` is using the `c()` function, which is short for "combine.""

```
c(1, 3, 5, 7, 8, 9)
```

```
## [1] 1 3 5 7 8 9
```

- If we would like to store this vector in a **variable** we can do so with the **assignment** operator `=`.
  - The variable `x` now holds the vector we just created, and we can access the vector by typing `x`.

```
x = c(1, 3, 5, 7, 8, 9)
x
```

```
## [1] 1 3 5 7 8 9
```

```
# The following does the same thing.
x <- c(1, 3, 5, 7, 8, 9)
x
```

```
## [1] 1 3 5 7 8 9
```

- The operator `=` and `<-` work as an assignment operator.
  - You can use both. This does not matter usually.
  - If you are interested in the weird cases where the difference matters, check out The R Inferno.
- In `R` code the line starting with `#` is `comment`, which is ignored when you run the fode.

- A vector based on a sequence of numbers.
- The quickest and easiest way to do this is with the `:` operator, which creates a sequence of integers between two specified integers.

```
(y = 1:100)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

- By putting parentheses around the assignment,
  - `R` both stores the vector in a variable called `y` and
  - automatically outputs `y` to the console.

4.3.2 Useful functions for creating vectors

- Use the `seq()` function for a more general sequence.

```
seq(from = 1.5, to = 4.2, by = 0.1)
```

```
##  [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1
## [18] 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2
```

- Here, the input labels `from`, `to`, and `by` are optional.

```
seq(1.5, 4.2, 0.1)
```

```
##  [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1
## [18] 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2
```

- The `rep()` function repeat a single value a number of times.

```
rep("A", times = 10)
```

```
##  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

- The `rep()` function can be used to repeat a vector some number of times.

```
rep(x, times = 3)
```

```
##  [1] 1 3 5 7 8 9 1 3 5 7 8 9 1 3 5 7 8 9
```

- We have now seen four different ways to create vectors:
  1. `c()`
  2. `:`
  3. `seq()`
  4. `rep()`
- They are often used together.

```
c(x, rep(seq(1, 9, 2), 3), c(1, 2, 3), 42, 2:4)
```

```
##  [1]  1  3  5  7  8  9  1  3  5  7  9  1  3  5  7  9  1  3  5  7  9  1  2
## [24]  3 42  2  3  4
```

- The length of a vector can be obtained with the `length()` function.

```
length(x)
```

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

```
length(y)
```

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

4.3.3 Subsetting

- Use square brackets, `[]`, to obtain a subset of a vector.
- We see that `x[1]` returns the first element.

```
x
```

```
## [1] 1 3 5 7 8 9
```

```
x[1]
```

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

```
x[3]
```

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

- We can also exclude certain indexes, in this case the second element.

```
x[-2]
```

```
## [1] 1 5 7 8 9
```

- We can subset based on a vector of indices.

```
x[1:3]
```

```
## [1] 1 3 5
```

```
x[c(1,3,4)]
```

```
## [1] 1 5 7
```

- We could instead use a vector of logical values.

```
z = c(TRUE, TRUE, FALSE, TRUE, TRUE, FALSE)
z
```

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

```
x[z]
```

```
## [1] 1 3 7 8
```

4.4 Vectorization

- One of the biggest strengths of `R` is its use of vectorized operations.
  - Frequently the lack of understanding of this concept leads of a belief that `R` is *slow*.
  - `R` is not the fastest language, but it has a reputation for being slower than it really is.)
- When a function like `log()` is called on a vector `x`, a vector is returned which has applied the function to each element of the vector `x`.

```
x = 1:10
x + 1
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

```
2 * x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
2 ^ x
```

```
##  [1]    2    4    8   16   32   64  128  256  512 1024
```

```
sqrt(x)
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
##  [8] 2.828427 3.000000 3.162278
```

```
log(x)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

4.5 Logical Operators

Operator	Summary	Example	Result
`x < y`	x less than y	`3 < 42`	TRUE
`x > y`	x greater than y	`3 > 42`	FALSE
`x <= y`	x less than or equal to y	`3 <= 42`	TRUE
`x >= y`	x greater than or equal to y	`3 >= 42`	FALSE
`x == y`	xequal to y	`3 == 42`	FALSE
`x != y`	x not equal to y	`3 != 42`	TRUE
`!x`	not x	`!(3 > 42)`	TRUE
`x | y`	x or y	`(3 > 42) | TRUE`	TRUE
`x & y`	x and y	`(3 < 4) & ( 42 > 13)`	TRUE

- Logical operators are vectorized.

```
x = c(1, 3, 5, 7, 8, 9)
```

```
x > 3
```

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

```
x < 3
```

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

```
x == 3
```

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

```
x != 3
```

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

```
x == 3 & x != 3
```

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

```
x == 3 | x != 3
```

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

- This is extremely useful for subsetting.

```
x[x > 3]
```

```
## [1] 5 7 8 9
```

```
x[x != 3]
```

```
## [1] 1 5 7 8 9
```

4.5.0.1 Short exercise

1. Create the vector $z = (1, 2, 1, 2, 1, 2)$, which has the same length as x.
2. Pick up the elements of x which corresponds to 1 in the vector z.

4.6 Matrices

4.6.1 Basics

- `R` can also be used for **matrix** calculations.
  - Matrices have rows and columns containing a single data type.
- Matrices can be created using the `matrix` function.

```
x = 1:9
x
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
X = matrix(x, nrow = 3, ncol = 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

- We are using two different variables:
  - lower case `x`, which stores a vector and
  - capital `X`, which stores a matrix.
- By default the `matrix` function reorders a vector into columns, but we can also tell `R` to use rows instead.

```
Y = matrix(x, nrow = 3, ncol = 3, byrow = TRUE)
Y
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

- a matrix of a specified dimension where every element is the same, in this case `0`.

```
Z = matrix(0, 2, 4)
Z
```

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

- Matrices can be subsetted using square brackets, `[]`.
- However, since matrices are two-dimensional, we need to specify both a row and a column when subsetting.
- Here we get the element in the first row and the second column.

```
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
X[1, 2]
```

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

- We could also subset an entire row or column.

```
X[1, ]
```

```
## [1] 1 4 7
```

```
X[, 2]
```

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

- Matrices can also be created by combining vectors as columns, using **`cbind`**, or combining vectors as rows, using **`rbind`**.

```
x = 1:9
rev(x)
```

```
## [1] 9 8 7 6 5 4 3 2 1
```

```
rep(1, 9)
```

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

```
rbind(x, rev(x), rep(1, 9))
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## x    1    2    3    4    5    6    7    8    9
##      9    8    7    6    5    4    3    2    1
##      1    1    1    1    1    1    1    1    1
```

- When using **`rbind`** and **`cbind`** you can specify "argument" names that will be used as column names.

```
cbind(col_1 = x, col_2 = rev(x), col_3 = rep(1, 9))
```

```
##       col_1 col_2 col_3
##  [1,]     1     9     1
##  [2,]     2     8     1
##  [3,]     3     7     1
##  [4,]     4     6     1
```

```
##  [5,]     5     5     1
##  [6,]     6     4     1
##  [7,]     7     3     1
##  [8,]     8     2     1
##  [9,]     9     1     1
```

4.6.2 Matrix calculations

- Perform matrix calculations.

```
x = 1:9
y = 9:1
X = matrix(x, 3, 3)
Y = matrix(y, 3, 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
Y
```

```
##      [,1] [,2] [,3]
## [1,]    9    6    3
## [2,]    8    5    2
## [3,]    7    4    1
```

```
X + Y
```

```
##      [,1] [,2] [,3]
## [1,]   10   10   10
## [2,]   10   10   10
## [3,]   10   10   10
```

```
X - Y
```

```
##      [,1] [,2] [,3]
## [1,]   -8   -2    4
## [2,]   -6    0    6
## [3,]   -4    2    8
```

```
X * Y
```

```
##      [,1] [,2] [,3]
## [1,]    9   24   21
## [2,]   16   25   16
## [3,]   21   24    9
```

```
X / Y
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.1111111 0.6666667 2.333333
## [2,] 0.2500000 1.0000000 4.000000
## [3,] 0.4285714 1.5000000 9.000000
```

- Note that `X * Y` is **not** matrix multiplication.
- It is element by element multiplication. (Same for `X / Y`).
- Matrix multiplication uses `%*%`.

- `t()` which gives the transpose of a matrix

```
X %*% Y
```

```
##      [,1] [,2] [,3]
## [1,]   90   54   18
## [2,]  114   69   24
## [3,]  138   84   30
```

```
t(X)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

- `solve()` which returns the inverse of a square matrix if it is invertible.

```
Z = matrix(c(9, 2, -3, 2, 4, -2, -3, -2, 16), 3, byrow = TRUE)
Z
```

```
##      [,1] [,2] [,3]
## [1,]    9    2   -3
## [2,]    2    4   -2
## [3,]   -3   -2   16
```

```
solve(Z)
```

```
##             [,1]        [,2]       [,3]
## [1,]  0.12931034 -0.05603448 0.01724138
## [2,] -0.05603448  0.29094828 0.02586207
## [3,]  0.01724138  0.02586207 0.06896552
```

- To verify that `solve(Z)` returns the inverse, we multiply it by `Z`.
  - We would expect this to return the identity matrix.
  - However we see that this is not the case due to some computational issues.
  - However, `R` also has the `all.equal()` function which checks for equality, with some small tolerance which accounts for some computational issues.

```
solve(Z) %*% Z
```

```
##              [,1]          [,2]         [,3]
## [1,] 1.000000e+00 -6.245005e-17 0.000000e+00
## [2,] 8.326673e-17  1.000000e+00 5.551115e-17
## [3,] 2.775558e-17  0.000000e+00 1.000000e+00
```

```
diag(3)
```

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

```
all.equal(solve(Z) %*% Z, diag(3))
```

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

4.6.2.1 Exercise

- Solve the following simultanoues equations using matrix calculation

$$2x_1 + 3x_2 = 105x_1 + x_2 = 20$$

- Hint: You can write this as $Ax = y$ where A is the 2-times-2 matrix, x and y are vectors with the length of 2.

4.6.3 Getting information for matrix

- `R` has a number of matrix specific functions for obtaining dimension and summary information.

```
X = matrix(1:6, 2, 3)
X
```

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

```
dim(X)
```

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

```
rowSums(X)
```

```
## [1]  9 12
```

```
colSums(X)
```

```
## [1]  3  7 11
```

```
rowMeans(X)
```

```
## [1] 3 4
```

```
colMeans(X)
```

```
## [1] 1.5 3.5 5.5
```

- The `diag()` function can be used in a number of ways. We can extract the diagonal of a matrix.

```
diag(Z)
```

```
## [1]  9  4 16
```

- Or create a matrix with specified elements on the diagonal. (And `0` on the off-diagonals.)

```
diag(1:5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    2    0    0    0
## [3,]    0    0    3    0    0
## [4,]    0    0    0    4    0
## [5,]    0    0    0    0    5
```

- Or, lastly, create a square matrix of a certain dimension with `1` for every element of the diagonal and `0` for the off-diagonals.

```
diag(5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

4.7 Lists

- A list is a one-dimensional heterogeneous data structure.
  - It is indexed like a vector with a single integer value,
  - but each element can contain an element of any type.

```
# creation
list(42, "Hello", TRUE)
```

```
## [[1]]
## [1] 42
##
## [[2]]
## [1] "Hello"
##
## [[3]]
## [1] TRUE
```

```
ex_list = list(
  a = c(1, 2, 3, 4),
  b = TRUE,
  c = "Hello!",
  d = function(arg = 42) {print("Hello World!")},
  e = diag(5)
)
```

- Lists can be subset using two syntaxes,
  1. the $ operator, and
  2. square brackets [].

```
# subsetting
ex_list$e
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

```
ex_list[1:2]
```

```
## $a
## [1] 1 2 3 4
##
## $b
## [1] TRUE
```

```
ex_list[1]
```

```
## $a
## [1] 1 2 3 4
```

```
ex_list[c("e", "a")]
```

```
## $e
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
```

```
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
##
## $a
## [1] 1 2 3 4
```

```
ex_list["e"]
```

```
## $e
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

```
ex_list$d
```

```
## function(arg = 42) {print("Hello World!")}
```

4.8 Data Frames

- We will talk about Dataframe in the next chapter.

4.9 Programming Basics -Control flow-

4.9.1 if/else

- The if/else syntax is:

```
if (...) {
  some R code
} else {
  more R code
}
```

- Example: To see whether x is large than y.

```
x = 1
y = 3
if (x > y) {
  z = x * y
  print("x is larger than y")
} else {
  z = x + 5 * y
  print("x is less than or equal to y")
}
```

```
## [1] "x is less than or equal to y"
```

```
z
```

```
## [1] 16
```

- `R` also has a special function `ifelse()`
  - It returns one of two specified values based on a conditional statement.

```
ifelse(4 > 3, 1, 0)
```

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

- The real power of `ifelse()` comes from its ability to be applied to vectors.

```
fib = c(1, 1, 2, 3, 5, 8, 13, 21)
ifelse(fib > 6, "Foo", "Bar")
```

```
## [1] "Bar" "Bar" "Bar" "Bar" "Bar" "Foo" "Foo" "Foo"
```

4.10 `for` loop

- A `for` loop repeats the same procedure for the specified number of times

```
x = 11:15
for (i in 1:5) {
  x[i] = x[i] * 2
}

x
```

```
## [1] 22 24 26 28 30
```

- Note that this `for` loop is very normal in many programming languages.
- In `R` we would not use a loop, instead we would simply use a vectorized operation.
  - `for` loop in `R` is known to be very slow.

```
x = 11:15
x = x * 2
x
```

```
## [1] 22 24 26 28 30
```

4.11 Functions

- To use a function,
  - you simply type its name,
  - followed by an open parenthesis,
  - then specify values of its arguments,
  - then finish with a closing parenthesis.
- An **argument** is a variable which is used in the body of the function.

```
# The following is just a demonstration, not the real function in R.
function_name(arg1 = 10, arg2 = 20)
```

- We can also write our own functions in `R`.
- Example: "standardize" variables

$$\frac{x - \bar{x}}{s}$$

- When writing a function, there are three thing you must do.

1. Give the function a name. Preferably something that is short, but descriptive.
2. Specify the arguments using `function()`
3. Write the body of the function within curly braces, `{}`.

```
standardize = function(x) {
  m = mean(x)
  std = sd(x)
  result = (x - m) / std
  return(result)
}
```

- Here the name of the function is `standardize`,
- The function has a single argument `x` which is used in the body of function.
- Note that the output of the final line of the body is what is returned by the function.
- Let's test our function
- Take a random sample of size `n = 10` from a normal distribution with a mean of `2` and a standard deviation of `5`.

```
test_sample = rnorm(n = 10, mean = 2, sd = 5)
test_sample
```

```
##  [1]  -1.152080  -1.902363   7.496638  -3.575524 -10.786826   5.679470
##  [7]   0.639967   1.931371   6.142081   7.622131
```

```
standardize(x = test_sample)
```

```
##  [1] -0.40376996 -0.53204981  1.07494844 -0.81811925 -2.05107485
##  [6]  0.76425745 -0.09737386  0.12342460  0.84335249  1.09640474
```

- The same function can be written more simply.

```
standardize = function(x) {
  (x - mean(x)) / sd(x)
}
```

- When specifying arguments, you can provide default arguments.

```
power_of_num = function(num, power = 2) {
  num ^ power
}
```

- Let's look at a number of ways that we could run this function to perform the operation `10^2` resulting in 100.

```
power_of_num(10)
```

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

```
power_of_num(10, 2)
```

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

```
power_of_num(num = 10, power = 2)
```

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

```
power_of_num(power = 2, num = 10)
```

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

- Note that without using the argument names, the order matters. The following code will not evaluate to the same output as the previous example.

```
power_of_num(2, 10)
```

```
## [1] 1024
```

- Also, the following line of code would produce an error since arguments without a default value must be specified.

```
power_of_num(power = 5)
```

- To further illustrate a function with a default argument, we will write a function that calculates sample variance two ways.
- By default, the function will calculate the unbiased estimate of σ^2, which we will call s^2.

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x - \bar{x})^2$$

- It will also have the ability to return the biased estimate (based on maximum likelihood) which we will call $\hat{\sigma}^2$.

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x - \bar{x})^2$$

```
get_var = function(x, unbiased = TRUE) {

  if (unbiased == TRUE){
    n = length(x) - 1
  } else if (unbiased == FALSE){
    n = length(x)
   }

  (1 / n) * sum((x - mean(x)) ^ 2)
}
```

```
get_var(test_sample)
```

```
## [1] 34.20838
```

```
get_var(test_sample, unbiased = TRUE)
```

```
## [1] 34.20838
```

```
var(test_sample)
```

```
## [1] 34.20838
```

- We see the function is working as expected, and when returning the unbiased estimate it matches `R`'s built in function `var()`. Finally, let's examine the biased estimate of σ^2.

```
get_var(test_sample, unbiased = FALSE)
```

```
## [1] 30.78754
```

Chapter 5

Data frame

5.1 Introduction

- A **data frame** is the most common way that we store and interact with data in this course.

```
example_data = data.frame(x = c(1, 3, 5, 7, 9, 1, 3, 5, 7, 9),
                          y = c(rep("Hello", 9), "Goodbye"),
                          z = rep(c(TRUE, FALSE), 5))
```

- A data frame is a **list** of vectors.
  - Each vector must contain the same data type
  - The difference vectors can store different data types.

```
example_data
```

```
##    x       y     z
## 1  1   Hello  TRUE
## 2  3   Hello FALSE
## 3  5   Hello  TRUE
## 4  7   Hello FALSE
## 5  9   Hello  TRUE
## 6  1   Hello FALSE
## 7  3   Hello  TRUE
## 8  5   Hello FALSE
## 9  7   Hello  TRUE
## 10 9 Goodbye FALSE
```

- `write.csv` save (or export) the dataframe in `.csv` format.

5.2 Load csv file

- We can also import data from various file types in into `R`, as well as use data stored in packages.
- Read `csv` file into R.
  - `read.csv()` function as default
  - `read_csv()` function from the `readr` package. This is faster for larger data.

```
# install.packages("readr")
#library(readr)
#example_data_from_csv = read_csv("example-data.csv")
example_data_from_csv = read.csv("example-data.csv")
```

- Note: This particular line of code assumes that the file `example_data.csv` exists in your current working directory.
- The current working directory is the folder that you are working with. To see this, you type

```
getwd()
```

```
## [1] "C:/Users/Yuta/Dropbox/Teaching/2019S_Applied_Econometrics_JPN_ENG/Material_Github"
```

- If you want to set the working directory, use `setwd()` function

```
setwd(dir = "directory path" )
```

5.3 Examine dataframe

- Inside the `ggplot2` package is a dataset called `mpg`. By loading the package using the `library()` function, we can now access `mpg`.

```
library(ggplot2)
```

- Three things we would generally like to do with data:
  - Look at the raw data.
  - Understand the data. (Where did it come from? What are the variables? Etc.)
  - Visualize the data.
- To look at the data, we have two useful commands: `head()` and `str()`

```
head(mpg, n = 10)
```

```
## # A tibble: 10 x 11
##    manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4      1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4      1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4      2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4      2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4      2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4      2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4      3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 q~   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 q~   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 q~   2    2008     4 manu~ 4        20    28 p     comp~
```

- The function `str()` will display the "structure" of the data frame.
  - It will display the number of **observations** and **variables**, list the variables, give the type of each variable, and show some elements of each variable.
  - This information can also be found in the "Environment" window in RStudio.

```
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
```

```
##  $ hwy          : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl           : chr  "p" "p" "p" "p" ...
##  $ class        : chr  "compact" "compact" "compact" "compact" ...
```

- `names()` function to obtain names of the variables in the dataset

```
names(mpg)
```

```
##  [1] "manufacturer" "model"        "displ"        "year"
##  [5] "cyl"          "trans"        "drv"          "cty"
##  [9] "hwy"          "fl"           "class"
```

- To access one of the variables **as a vector**, we use the `$` operator.

```
mpg$year
```

```
##   [1] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 2008 1999 1999 2008
##  [15] 2008 1999 2008 2008 2008 2008 2008 1999 2008 1999 1999 2008 2008 2008
##  [29] 2008 2008 1999 1999 1999 2008 1999 2008 2008 1999 1999 1999 1999 2008
##  [43] 2008 2008 1999 1999 2008 2008 2008 2008 1999 1999 2008 2008 2008 1999
##  [57] 1999 1999 2008 2008 2008 1999 2008 1999 2008 2008 2008 2008 2008 2008
##  [71] 1999 1999 2008 1999 1999 1999 2008 1999 1999 1999 2008 2008 1999 1999
##  [85] 1999 1999 1999 2008 1999 2008 1999 1999 2008 2008 1999 1999 2008 2008
##  [99] 2008 1999 1999 1999 1999 1999 2008 2008 2008 2008 1999 1999 2008 2008
## [113] 1999 1999 2008 1999 1999 2008 2008 2008 2008 2008 2008 2008 1999 1999
## [127] 2008 2008 2008 2008 1999 2008 2008 1999 1999 1999 2008 1999 2008 2008
## [141] 1999 1999 1999 2008 2008 2008 2008 1999 1999 2008 1999 1999 2008 2008
## [155] 1999 1999 1999 2008 2008 1999 1999 2008 2008 2008 2008 1999 1999 1999
## [169] 1999 2008 2008 2008 2008 1999 1999 1999 1999 2008 2008 1999 1999 2008
## [183] 2008 1999 1999 2008 1999 1999 2008 2008 1999 1999 2008 1999 1999 1999
## [197] 2008 2008 1999 2008 1999 1999 2008 1999 1999 2008 2008 1999 1999 2008
## [211] 2008 1999 1999 1999 1999 2008 2008 2008 2008 1999 1999 1999 1999 1999
## [225] 1999 2008 2008 1999 1999 2008 2008 1999 1999 2008
```

```
mpg$hwy
```

```
##   [1] 29 29 31 30 26 26 27 26 25 28 27 25 25 25 25 24 25 23 20 15 20 17 17
##  [24] 26 23 26 25 24 19 14 15 17 27 30 26 29 26 24 24 22 22 24 24 17 22 21
##  [47] 23 23 19 18 17 17 19 19 12 17 15 17 17 12 17 16 18 15 16 12 17 17 16
##  [70] 12 15 16 17 15 17 17 18 17 19 17 19 19 17 17 17 16 16 17 15 17 26 25
##  [93] 26 24 21 22 23 22 20 33 32 32 29 32 34 36 36 29 26 27 30 31 26 26 28
## [116] 26 29 28 27 24 24 24 22 19 20 17 12 19 18 14 15 18 18 15 17 16 18 17
## [139] 19 19 17 29 27 31 32 27 26 26 25 25 17 17 20 18 26 26 27 28 25 25 24
## [162] 27 25 26 23 26 26 26 26 25 27 25 27 20 20 19 17 20 17 29 27 31 31 26
## [185] 26 28 27 29 31 31 26 26 27 30 33 35 37 35 15 18 20 20 22 17 19 18 20
## [208] 29 26 29 29 24 44 29 26 29 29 29 29 23 24 44 41 29 26 28 29 29 29 28
## [231] 29 26 26 26
```

- We can use the `dim()`, `nrow()` and `ncol()` functions to obtain information about the dimension of the data frame.

```
dim(mpg)
```

```
## [1] 234  11
```

```
nrow(mpg)
```

```
## [1] 234
```

```
ncol(mpg)
```

```
## [1] 11
```

5.4 Subsetting data

- Subsetting data frames can work much like subsetting matrices using square brackets, `[,]`.
- Here, we find fuel efficient vehicles earning over 35 miles per gallon and only display `manufacturer`, `model` and `year`.

```
mpg[mpg$hwy > 35, c("manufacturer", "model", "year")]
```

```
## # A tibble: 6 x 3
##   manufacturer model       year
##   <chr>        <chr>      <int>
## 1 honda        civic       2008
## 2 honda        civic       2008
## 3 toyota       corolla     2008
## 4 volkswagen   jetta       1999
## 5 volkswagen   new beetle  1999
## 6 volkswagen   new beetle  1999
```

- An alternative would be to use the `subset()` function, which has a much more readable syntax.

```
subset(mpg, subset = hwy > 35, select = c("manufacturer", "model", "year"))
```

- Lastly, we could use the `filter` and `select` functions from the `dplyr` package which introduces the `%>%` operator from the `magrittr` package.

```
library(dplyr)
mpg %>%
filter(hwy > 35) %>%
select(manufacturer, model, year)
```

- I will give you an assignment about `dplyr` package in the `DataCamp` as a makeup lecture.

Chapter 6

Exercise 1

- Due date: April 22th (Monday) 11pm

Rules for Problem Sets

- If you are enrolled in Japanese class (i.e., Wednesday 2nd), you can use both Japanese and English to write your answer.
- Submit your solution through `CourseN@vi`.
- Submit both your answer and R script.
- Using `Rmarkdown` would be appreciated, though not mandatory.
  - `Rmarkdown` introduction in Japanese: https://kazutan.github.io/kazutanR/Rmd_intro.html
  - `Rmarkdown` introduction in English: https://rmarkdown.rstudio.com/articles_intro.html
- I might cover `Rmarkdown` in the course later.

6.1 Update (as of 10am, April 18th)

- Please calculate the standard deviation, not the variance in your simulation. The parameter you set when drawing the random number is the mean μ and the standard deviation σ in normal distribution.
- Use `seq` function to create the sequence of the sample sizes that you use in the simulation.

6.2 Question: Examine the law of large numbers through numerical simulations

Consider the random sample of $\{x_i\}_{i=1}^N$ drawn from the random variable X. The law of large numbers implies that

$$\frac{1}{N}\sum_{i=1}^{N} x_i \xrightarrow{p} E[X]$$

In other words, the sample mean converges to the population mean in probability as the sample size goes to infinity (i.e., $N \to \infty$).

Similary, the sample variance also converges to the population variance in probability

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \xrightarrow{p} V[X]$$

(Note on 4/18) This implies that the sample standard deviation also converges to the population standard deviation

(This is an application of the law of large numbers, though it is a bit involved to prove this.)

The goal of this problem set is to demonstrate these two properties through numerical simulations. Here is what we are going to do:

1. For a certain sample size N, draw N random numbers from the normal distribution with known mean and standard deviation.
2. Calculate the sample mean and the sample variance for the "data" you draw.
3. Repeat this for many different sample sizes.
4. Examine to see whether the sample mean and **standard deviation** are getting closer to the true value, which you set when you draw the random numbers, as the sample size gets larger.

6.2.1 How to implement

I explain how to implement this in `R` step by step below.

1. Prepare a function like this
    1. There are two inputs: (1) a vector that contains the data $\{x_i\}_{i=1}^N$ and (2) the indicator of whether you calculate the mean or the standard deviation.

```
fun_something = function(firstinput, secondinput){

  # Two inputs: firstinput, secondinput
  # One output: output

  # Do something.

  return(output)

}
```

    2. Use if/else sentence. Example:

```
# "secondinput" is the name of the input variable in your function
if ( secondinput == "mean"){
  # calculate mean of the data (firstinput)

} else if ( secondinput == "sd"){
  # Calculate standard deviation of the data (firstinput)
}
```

    3. Use `return` function to define the output of the function.
2. Construct a vector that contains the sample size you want to use in your simulation. For example:

```
samplesize_vec = seq(from = 100, to = 100000, by = 100)
```

Here, let's try 100 different sample sizes that ranges from 100 to 100000.

3. Prepare two vectors that contain the result in the `forloop` below. Since we are trying 100 different sample sizes, let's create a vector with the length of 100.

```
# Hint:
# numeric(k) returns a zero vector with the length of k
# length( vector) returns the length of `vector`

# result_mean = ....
# result_sd = ....
```

4. To create the random draw from the normal distribution, use below

```
# You can choose the mean and the standard deviation as you like.
rnorm(n = 100, mean = 2, sd = 5)
```

4. Use `forloop` to calculate both mean and the standard deviation for each sample size. For example:

```
for (i in 1:length(samplesize_vec)){

  # Draw the random number

  # Calculate the mean using the function you construct.

  # Calculate the standard deviation using the function you construct.

}
```

5. Plot the result with `ggplot2`.
    1. Install the package if you have not done it yet.
    2. Load `ggplot2` by `library(ggplot2)`
    3. Use `qplot` command to make a figure

```
# Create plot and save it as the variable `plot1`
plot1 <- qplot(x = samplesize_vec, y = yourresult, geom = "line")

# print "plot1"
print(plot1)

# save the plot as PNG file
ggsave(file = "filename.png", plot = plot1)
```

6.2.2 What to submit

Your answer should include

1. The true value of mean and variance you choose in your simulation.
2. The plot that describes the relationship between the sample mean (variance) and the sample size.
3. Explain what the plots from your simulation indicate.

Chapter 7

A Review of Statistics

Acknowledgement: This chapter is largely based on chapter 3 of "Introduction to Econometrics with R". https://www.econometrics-with-r.org/index.html

The goal of this chapter is

1. Review of important concepts in statistics
    1. Estimation
    2. Hypothesis testing
2. Review of tools from probability theory
    1. Law of large numbers
    2. Central limit theorem

7.1 Estimation

- Estimator: A mapping from the sample data drawn from an unknown population to a certain feature in the population
- Example: Consider hourly earnings of college graduates Y .
- You want to estimate the mean of Y, defined as $E[Y] = \mu_y$
- Draw a random sample of n i.i.d. (identically and independently distributed) observations $Y_1, Y_2, \ldots, Y_N$
- How to estimate $E[Y]$ from the data?
- Idea 1: Sample mean

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i,$$

- Idea 2: Pick the first observation of the sample.
- Question: How can we say which is better?

7.1.1 Properties of the estimator

Consider the estimator $\hat{\mu}_N$ for the unknown parameter μ.

1. Unbiasdeness: The expectation of the estimator is the same as the true parameter in the population.

$$E[\hat{\mu}_N] = \mu$$

2. Consistency: The estimator converges to the true parameter in probability.

$$\forall \epsilon > 0, \lim_{N\to\infty} Prob(|\hat{\mu}_N - \mu| < \epsilon) = 1$$

- Intuition: As the sample size gets larger, the estimator and the true parameter is close with probability one.
- Note: a bit different from the usual convergence of the sequence.

7.1.2 Sample mean $\bar{Y}$ is unbiased and consistent

- Showing these two properties using mathmaetics is straightforward:
  - Unbiasedness: Take expectation.
  - Consistency: Law of large numbers.
- Let's examine these two properties using R.
- Step 1: Prepare a population. Here, I prepare income and age data from PUMS 5% sample of U.S. Census 2000.
  - PUMS: Public Use Microdata Sample
  - Download the example data here as a .csv file. Put this file in the same folder as your R script file.

```
# Use "readr" package
library(readr)
pums2000 <- read_csv("data_pums_2000.csv")
```

```
## Parsed with column specification:
## cols(
##   AGE = col_double(),
##   INCTOT = col_double()
## )
```

- We treat this dataset as **population**.

```
pop <- as.vector(pums2000$INCTOT)
```

- *Population* mean and standard deviation

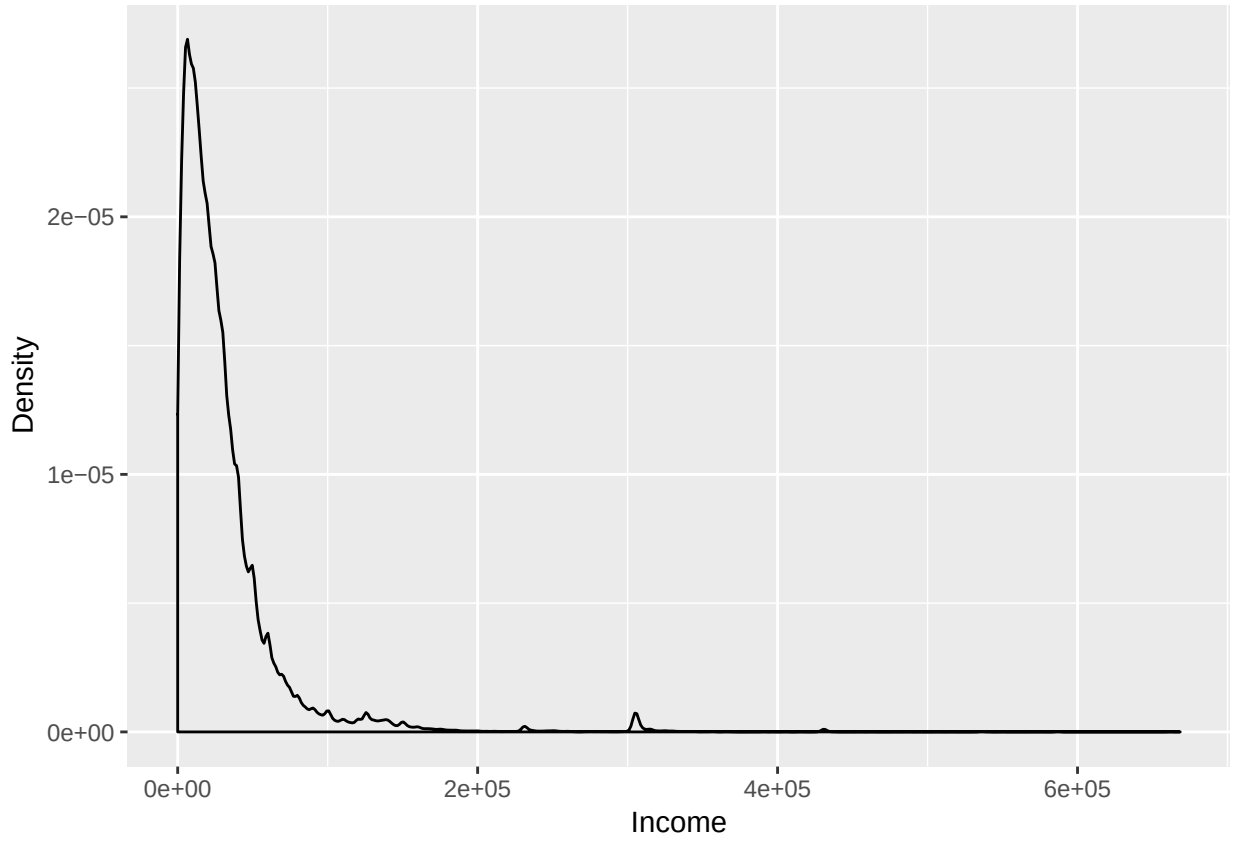
```
pop_mean = mean(pop)
pop_sd   = sd(pop)

# Average income in population
pop_mean
```

```
## [1] 30165.47
```

```
# Standard deviation of income in population
pop_sd
```

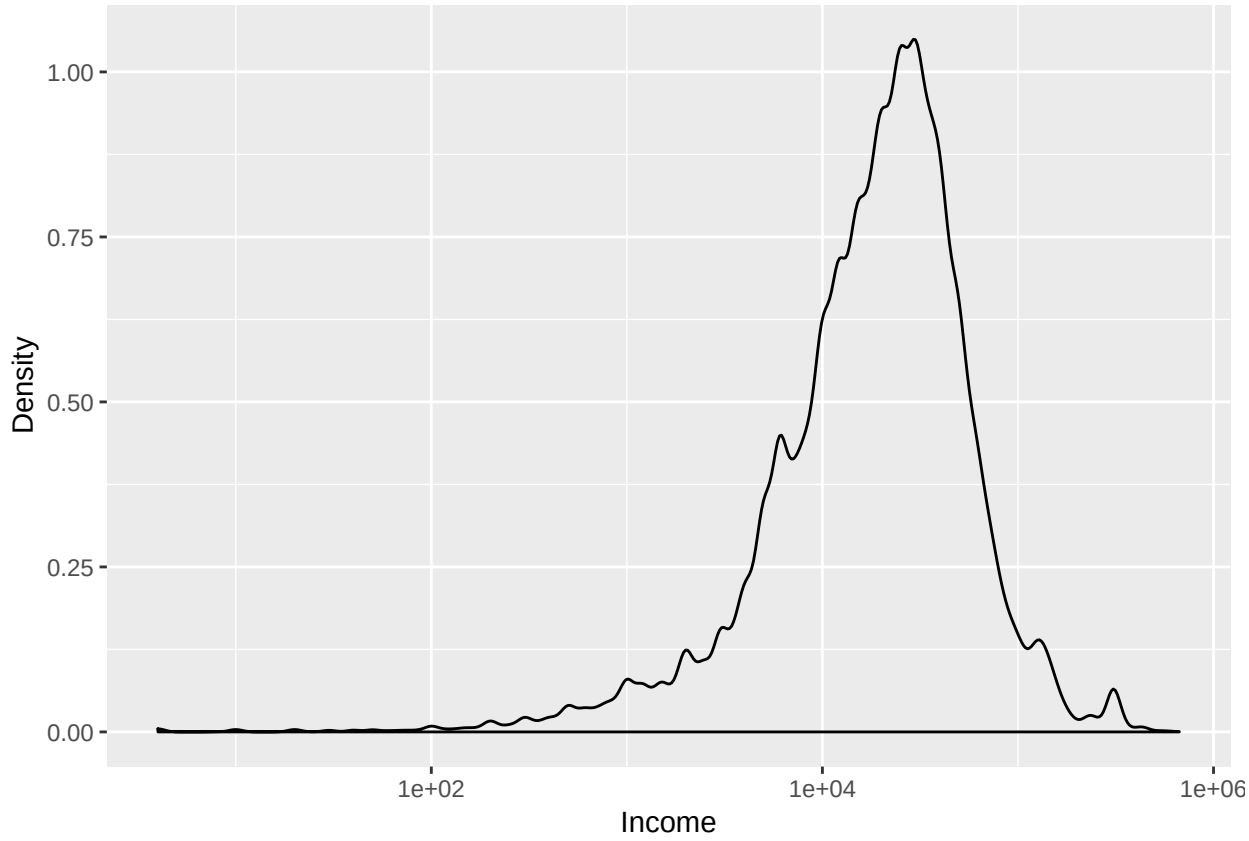
```
## [1] 38306.17
```

```
# income distribution in population
# Note that the unit is in USD.
library("ggplot2")
qplot(pop, geom = "density",
      xlab = "Income",
      ylab = "Density")
```



- The distribution has a long tail.
- Let's plot the distribution in *log* scale

```
# `log` option specifies which axis is represented in log scale.
qplot(pop, geom = "density",
      xlab = "Income",
      ylab = "Density",
      log = "x")
```



- Let's investigate how close the sample mean constucted from the random sample is to the true population mean.
- Step 1: Draw random sample*s* from this population and calculate $\bar{Y}$ for each sample.
  - Set the sample size N.
- Step 2: Repeat 2000 times. You now have 2000 sample means.

```
# Set the seed for the random number. This is needed to maintaine the reproducibility of the results.
set.seed(123)

# draw random sample of 100 observations from the variable pop
test <- sample(x = pop, size = 100)

# Use loop to repeat 2000 times.
Nsamples = 2000
result1 <- numeric(Nsamples)

for (i in 1:Nsamples ){

  test <- sample(x = pop, size = 100)
  result1[i] <- mean(test)

}

# Simple approach
result1 <- replicate(expr = mean(sample(x = pop, size = 10)), n = Nsamples)
result2 <- replicate(expr = mean(sample(x = pop, size = 100)), n = Nsamples)
result3 <- replicate(expr = mean(sample(x = pop, size = 500)), n = Nsamples)
```

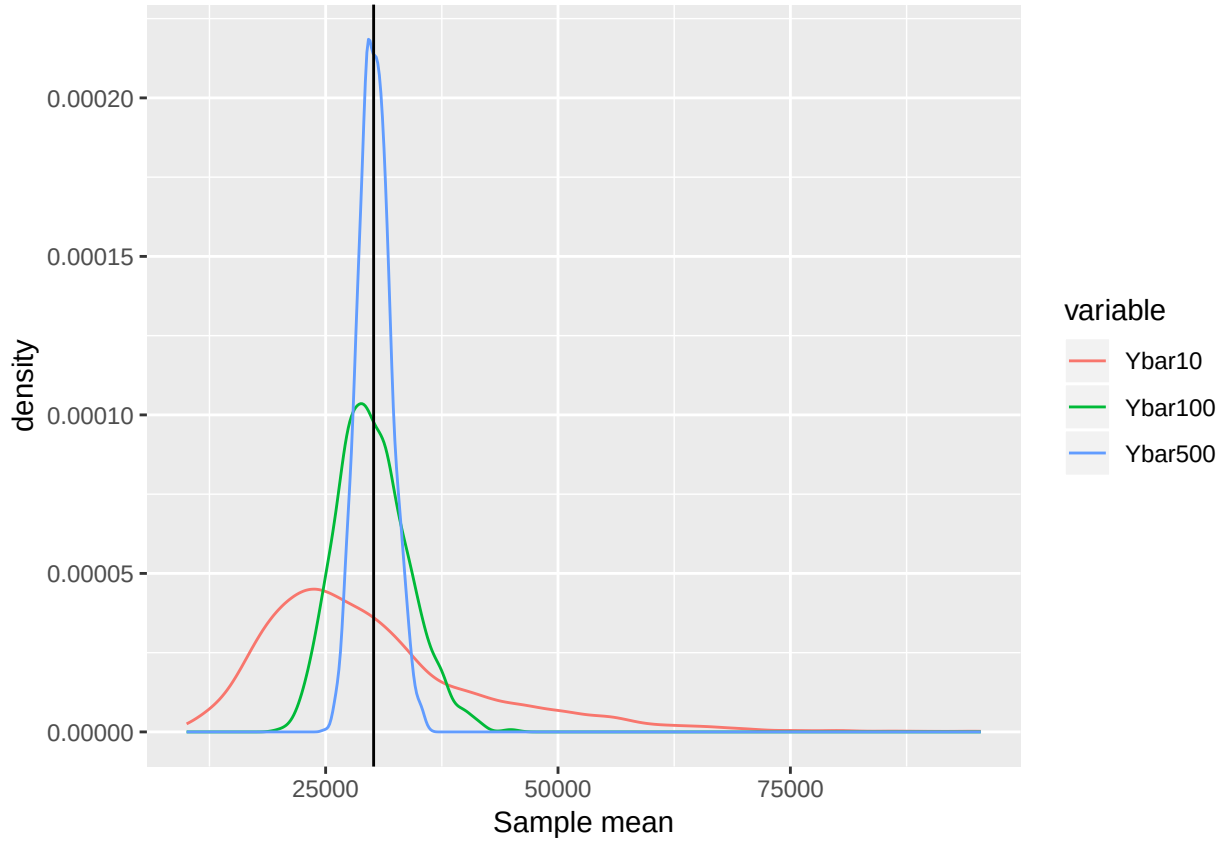
```
# Create dataframe

result_data <- data.frame(  Ybar10 = result1,
                            Ybar100 = result2,
                            Ybar500 = result3)
```

- Step 3: See the distribution of those 2000 sample means.

```
# Use reshape library
# install.packages("reshape")
library("reshape")

# Use "melt" to change the format of result_data
data_for_plot <- melt(data = result_data, variable.name = "Variable" )
```

```
## Using  as id variables
```

```
# Use "ggplot2" to create the figure.
# The variable `fig` contains the information about the figure
fig <-
  ggplot(data = data_for_plot) +
  xlab("Sample mean") +
  geom_line(aes(x = value, colour = variable ),   stat = "density" ) +
  geom_vline(xintercept=pop_mean ,colour="black")

# Display the figure
plot(fig)
```



- Observation 1: Regardless of the sample size, the average of the sample means is close to the population mean. **Unbiasdeness**
- Observation 2: As the sample size gets larger, the distribution is concentrated around the population mean. **Consistency (law of large numbers)**

7.2 Hypothesis Testing

7.2.1 Central limit theorem

- Cental limit theorem: Consider the i.i.d. sample of $Y_1, \cdots, Y_N$ drawn from the random variable Y with mean μ and variance σ^2. The following Z converges in distribution to the normal distribution.

$$Z = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \frac{Y_i - \mu}{\sigma} \xrightarrow{d} N(0,1)$$

  In other words,

$$\lim_{N \to \infty} P\left(Z \leq z\right) = \Phi(z)$$

- The central limit theorem implies that if N is large **enough**, we can **approximate** the distribution of $\bar{Y}$ by the standard normal distribution with mean μ and variance σ^2/N **regardless of the underlying distribution of Y.**
- Let's examine this property through simulation!!
- Use the same example as before. Remember that the underlying income distribution is clearly NOT normal.
  - Population mean $\mu = 3.0165467 \times 10^4$ and standard deviation $\sigma = 3.8306171 \times 10^4$. Use these numbers.

```
# Set the seed for the random number
set.seed(124)

# define function for simulation
f_simu_CLT = function(Nsamples, samplesize, pop, pop_mean, pop_sd ){

  output = numeric(Nsamples)
  for (i in 1:Nsamples ){
    test <- sample(x = pop, size = samplesize)
    output[i] <- ( mean(test) - pop_mean ) / (pop_sd / sqrt(samplesize))
  }

  return(output)

}

# Comment: You can do better without using forloop. Let me know if you come with a good idea.

# Run simulation
Nsamples = 2000
result_CLT1 <- f_simu_CLT(Nsamples, 10, pop, pop_mean, pop_sd )
result_CLT2 <- f_simu_CLT(Nsamples, 100, pop, pop_mean, pop_sd )
result_CLT3 <- f_simu_CLT(Nsamples, 1000, pop, pop_mean, pop_sd )

# Random draw from standard normal distribution as comparison
```

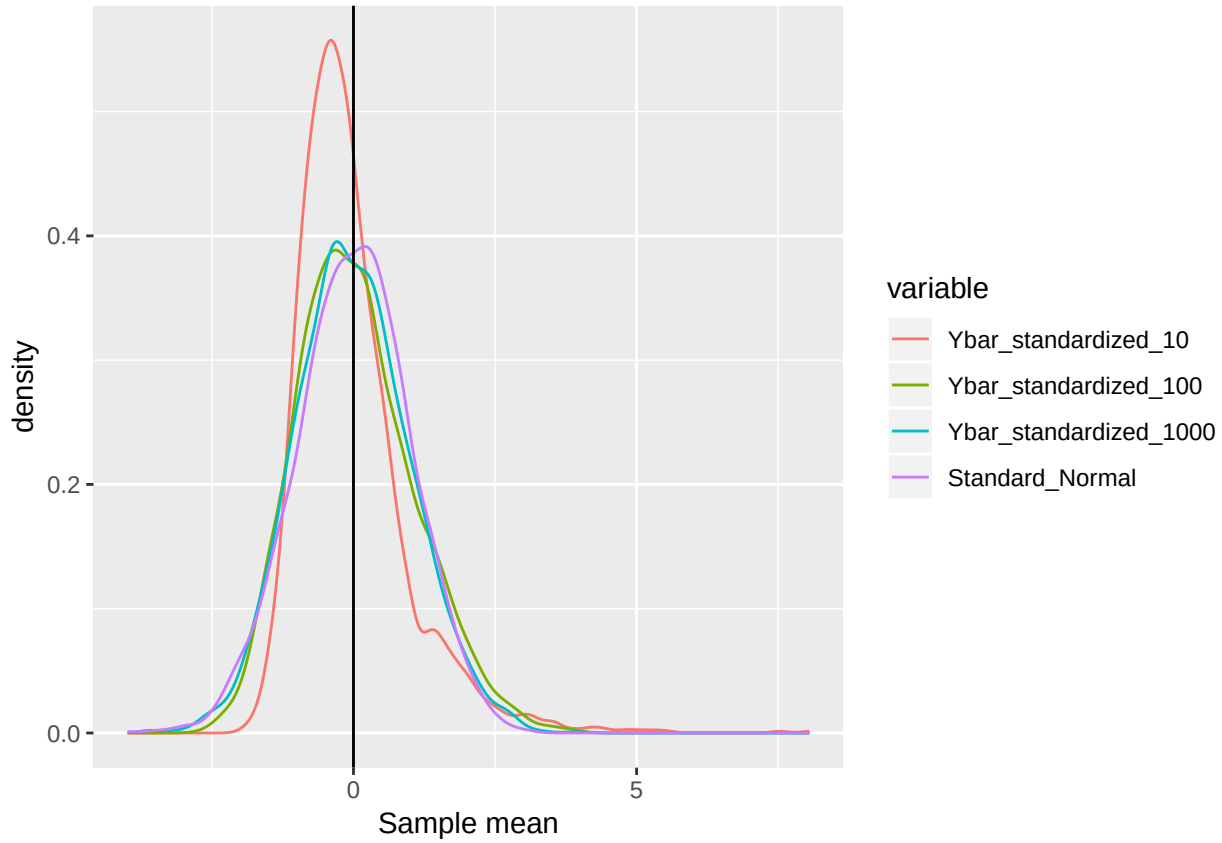
```
result_stdnorm = rnorm(Nsamples)

# Create dataframe
result_CLT_data <- data.frame(  Ybar_standardized_10 = result_CLT1,
                                Ybar_standardized_100 = result_CLT2,
                                Ybar_standardized_1000 = result_CLT3,
                                Standard_Normal = result_stdnorm)

# Note: If you wanna quicky plot the density, type `plot(density(result1))`.
```

- Now take a look at the distribution.

```
# Use "melt" to change the format of result_data
data_for_plot <- melt(data = result_CLT_data, variable.name = "Variable" )
```

```
## Using  as id variables
```

```
# Use "ggplot2" to create the figure.
fig <-
  ggplot(data = data_for_plot) +
  xlab("Sample mean") +
  geom_line(aes(x = value, colour = variable ),    stat = "density" ) +
  geom_vline(xintercept=0 ,colour="black")

plot(fig)
```



- As the sample size grows, the distribution of Z converges to the standard normal distribution.

7.2.2 Hypothesis testing

To be added.

Chapter 8

Linear Regression 1: Theory

8.1 Regression framework

- Let Y_i be the dependent variable and X_{ik} be k-th explanatory variable.
  - We have K explantory variables (along with constant term)
  - i is an index for observations. $i = 1, \cdots, N$.
  - Data (sample): $\{Y_i, X_{i1}, \ldots, X_{iK}\}_{i=1}^N$
- **Linear regression model** is defined as

$$Y_i = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_K X_{Ki} + \epsilon_i$$

  - ϵ_i: error term (unobserved)
  - β: coefficients
- **Assumptions for Ordinaly Least Squares (OLS) estimation**
  1. Random sample: $\{Y_i, X_{i1}, \ldots, X_{iK}\}$ is i.i.d. drawn sample
     - i.i.d.: identically and independently distributed
  2. ϵ_i has zero conditional mean
$$E[\epsilon_i | X_{i1}, \ldots, X_{iK}] = 0$$
  3. Large outliers are unlikely: The random variable Y_i and X_{ik} have finite fourth moments.
  4. No perfect multicollinearity: There is no linear relationship betwen explanatory variables.
- OLS estimators are the minimizers of the sum of squared residuals:

$$\min_{\beta_0, \cdots, \beta_K} \frac{1}{N} \sum_{i=1}^{N} (Y_i - (\beta_0 + \beta_1 X_{i1} + \cdots + \beta_K X_{iK}))^2$$

- Using matrix notation, we have the following analytical formula for the OLS estimator

$$\hat{\beta} = (X'X)^{-1} X'Y$$

where

$$\underbrace{X}_{N \times (K+1)} = \begin{pmatrix} 1 & X_{11} & \cdots & X_{1K} \\ \vdots & \vdots & & \vdots \\ 1 & X_{N1} & \cdots & X_{NK} \end{pmatrix}, \underbrace{Y}_{N \times 1} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_N \end{pmatrix}, \underbrace{\beta}_{(K+1) \times 1} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_K \end{pmatrix}$$

8.2 Theoretical Properties of OLS estimator

- We briefly review theoretical properties of OLS estimator.

1. **Unbiasdness**: Conditional on the explantory variables X, the expectation of the OLS estimator $\hat{\beta}$ is equal to the true value β.
$$E[\hat{\beta}|X] = \beta$$
2. **Consistency**: As the sample size N goes to infinity, the OLS estimator $\hat{\beta}$ converges to β in probability
$$\hat{\beta} \xrightarrow{p} \beta$$
3. **Asymptotic normality**: Will talk this later

8.3 Interpretation and Specifications of Linear Regression Model

- Remember that
$$Y_i = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_K X_{Ki} + \epsilon_i$$
- The coefficient β_k captures the effect of X_k on Y **ceteris paribus (all things being equal)**
- Equivalently,
$$\frac{\partial Y}{\partial X_k} = \beta_k$$
if X_k is continuous random variable.
- If we can estimate β_k without bias, we can obtain **causal effect** of X_k on Y.
  - This is of course very difficult task. We will see this more later.
- We will see several specifications that are frequently used in empirical analysis. 1. Nonlinear term 1. log specification 2. dummy (categorical) variables 3. interaction terms

8.3.1 Nonlinear term

- We can capture non-linear relationship between Y and X in a linearly additive form
$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3 + \epsilon_i$$
- As long as the error term ϵ_i appreas in a additively linear way, we can estimate the coefficients by OLS.
  - Multicollinarity could be an issue if we have many polynomials (see later).
  - You can use other non-linear variables such as $log(x)$ and $\sqrt{x}$.

8.3.2 log specification

- We often use `log` variables in both dependent and independent variables.
- Using `log` changes the interpretation of the coefficient β in terms of scales.

Dependent variable	Explanatory variable	interpretation
Y	X	1 unit increase in X causes β units change in Y
$\log Y$	X	1 unit increase in X causes $100\beta\%$ incchangerease in Y
Y	$\log X$	1% increase in X causes $\beta/100$ unit change in Y
$\log Y$	$\log X$	1% increase in X causes $\beta\%$ change in Y

8.3.3 Dummy variable

- A dummy variable takes only 1 or 0. This is used to express qualititative information
- Example: Dummy variable for race
$$white_i = \begin{cases} 1 & if\ white \\ 0 & otherwise \end{cases}$$
- The coefficient on a dummy variable captures the difference of the outcome Y between categories

- Consider the linear regression

$$Y_i = \beta_0 + \beta_1 white_i + \epsilon_i$$

The coefficient β_1 captures the difference of Y between white and non-white people.

8.3.4 Interaction term

- You can add the interaction of two explanatory variables in the regression model.
- For example:

$$wage_i = \beta_0 + \beta_1 educ_i + \beta_2 educ_i \times white_i + \epsilon_i$$

where $wage_i$ is the earnings of person i and $educ_i$ is the years of schooling for person i.
- The effect of $educ_i$ is

$$\frac{\partial wage_i}{\partial educ_i} = \beta_1 + \beta_2 white_i,$$

- This allows for heterogenous effects of education across races.

8.4 Measures of Fit

- We often use R^2 as a measure of the model fit.
- Denote **the fitted value** as $\hat{y}_i$

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_K X_{iK}$$

  - Also called prediction from the OLS regression.
- R^2 is defined as

$$R^2 = \frac{SSE}{TSS},$$

where

$$SSE = \sum_i (\hat{y}_i - \bar{y})^2,\ TSS = \sum_i (y_i - \bar{y})^2$$

- R^2 captures the fraction of the variation of Y explained by the regression model.
- Adding variables always (weakly) increases R^2.
- In a regression model with multiple explanatory variables, we often use **adjusted** R^2 that adjusts the number of explanatory variables

$$\bar{R}^2 = 1 - \frac{N-1}{N-(K+1)} \frac{SSR}{TSS}$$

where

$$SSR = \sum_i (\hat{y}_i - y_i)^2 (= \sum_i \hat{u}_i^2),$$

8.5 Statistical Inference

- Notice that the OLS estimators are **random variables**. They depend on the data, which are random variables drawn from some population distribution.
- We can conduct statistical inferences regarding those OLS estimators: 1. Hypothesis testing 2. Constructing confidence interval
- I first explain the sampling distribution of the OLS estimators.

8.5.1 Distribution of the OLS estimators based on asymptotic theory

- Deriving the exact (finite-sample) distribution of the OLS estimators is very hard.
  - The OLS estimators depend on the data Y_i, X_i in a complex way.
  - We typically do not know the distribution of Y and X.
- We rely on **asymptotic** argument. We approximate the sampling distribution of the OLS esimator based on the cental limit theorem.
- Under the OLS assumption, the OLS estimator has **asymptotic normality**

$$\sqrt{N}(\hat{\beta} - \beta) \xrightarrow{d} N(0, V)$$

where

$$\underbrace{V}_{(K+1)\times(K+1)} = E[\mathbf{x}_i'\mathbf{x}_i]^{-1} E[\mathbf{x}_i'\mathbf{x}_i \epsilon_i^2] E[\mathbf{x}_i'\mathbf{x}_i]^{-1}$$

and

$$\underbrace{\mathbf{x}_i}_{(K+1)\times 1} = \begin{pmatrix} 1 \\ X_{i1} \\ \vdots \\ X_{iK} \end{pmatrix}$$

- We can **approximate** the distribution of $\hat{\beta}$ by

$$\hat{\beta} \sim N(\beta, V/N)$$

- The above is joint distribution. Let V_{ij} be the (i, j) element of the matrix V.
- The individual coefficient β_k follows

$$\hat{\beta}_k \sim N(\beta_k, V_{kk}/N)$$

8.5.1.1 Estimation of Asymptotic Variance

- V is an unknown object. Need to be estimated.
- Consider the estimator $\hat{V}$ for V using sample analogues

$$\hat{V} = \left(\frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i'\mathbf{x}_i \right)^{-1} \left(\frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i'\mathbf{x}_i \hat{\epsilon}_i^2 \right) \left(\frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i'\mathbf{x}_i \right)^{-1}$$

where $\hat{\epsilon}_i = y_i - (\hat{\beta}_0 + \cdots + \hat{\beta}_K X_{iK})$ is the residual.
- Technically speaking, $\hat{V}$ converges to V in probability. (Proof is out of the scope of this course)
- We often use the (asymptotic) **standard error** $SE(\hat{\beta}_k) = \sqrt{\hat{V}_{kk}/N}$.
- The standard error is an estimator for the standard deviation of the OLS estimator $\hat{\beta}_k$.

8.5.2 Hypothesis testing

- OLS estimator is the random variable.
- You might want to test a particular hypothesis regarding those coefficients.
  - Does x really affects y?
  - Is the production technology the constant returns to scale?
- Here I explain how to conduct hypothesis testing.
- Step 1: Consider the null hypothesis H_0 and the alternative hypothesis H_1

$$H_0 : \beta_1 = k, H_1 : \beta_1 \neq k$$

where k is the known number you set by yourself.

- Step 2: Define **t-statistic** by

$$t_n = \frac{\hat{\beta}_1 - k}{SE(\hat{\beta}_1)}$$

- Step 3: We reject H_0 is at α-percent significance level if

$$|t_n| > C_{\alpha/2}$$

where $C_{\alpha/2}$ is the $\alpha/2$ percentile of the standard normal distribution.

  - We say we **fail to reject** H_0 if the above does not hold.

8.5.2.1 Caveats on Hypothesis Testing

- We often say $\hat{\beta}$ is **statistically significant** at 5% level if $|t_n| > 1.96$ when we set $k = 0$.
- Arguing the statistical significance alone is not enough for argument in empirical analysis.
- Magnitude of the coefficient is also important.
- Case 1: Small but statistically significant coefficient.
  - As the sample size N gets large, the SE decreases.
- Case 2: Large but statistically insignificant coefficient.
  - The variable might have an important (economically meaningful) effect.
  - But you may not be able to estimate the effect precisely with the sample at your hand.

8.5.2.2 F test

- We often test a composite hypothesis that involves multiple parameters such as

$$H_0 : \beta_1 + \beta_2 = 0,\ H_1 : \beta_1 + \beta_2 \neq 0$$

- We use **F test** in such a case (to be added).

8.5.3 Confidence interval

- 95% confidence interval

$$CI_n = \left\{ k : |\frac{\hat{\beta}_1 - k}{SE(\hat{\beta}_1)}| \leq 1.96 \right\} = \left[\hat{\beta}_1 - 1.96 \times SE(\hat{\beta}_1), \hat{\beta}_1 + 1.96 \times SE(\hat{\beta}_1) \right]$$

- Interpretation: If you draw many samples (dataset) and construct the 95% CI for each sample, 95% of those CIs will include the true parameter.

8.5.4 Homoskedasticity vs Heteroskedasticity

- So far, we did not put any assumption on the variance of the error term ϵ_i.
- The error term ϵ_i has **heteroskedasticity** if $Var(u_i|X_i)$ depends on X_i.
- If not, we call ϵ_i has **homoskedasticity**.
- This has an important implication on the asymptotic variance.
- Remember the asymptotic variance

$$\underbrace{V}_{(K+1)\times(K+1)} = E[\mathbf{x}_i'\mathbf{x}_i]^{-1} E[\mathbf{x}_i'\mathbf{x}_i\epsilon_i^2] E[\mathbf{x}_i'\mathbf{x}_i]^{-1}$$

Standard errors based on this is called **heteroskedasticity robust standard errors**/

- If homoskedasticity holds, then

$$V = E[\mathbf{x}_i'\mathbf{x}_i]^{-1}\sigma^2$$

where $\sigma^2 = V(\epsilon_i)$.

- In many statistical packages (including R and Stata), the standard errors for the OLS estimators are calcualted under homoskedasticity assumption as a default.
- However, if the error has heteroskedasticity, the standard error under homoskedasticity assumption will be **underestimated**.
- In OLS, **we should always use heteroskedasticity robust standard error.**
  - We will see how to fix this in R.

Chapter 9

Linear Regression 2: Implementation in R

9.1 Implementation in R

9.1.1 Preliminary: packages

- We use the following packages:
  - `AER` :
  - `dplyr` : data manipulation
  - `stargazer` : output of regression results

```
# Install package if you have not done so
# install.packages("AER")
# install.packages("dplyr")
# install.packages("stargazer")
# install.packages("lmtest")

# load packages
library("AER")
library("dplyr")
library("stargazer")
library("lmtest")
```

9.1.2 Empirical setting: Data from California School

- Question: How does the student-teacher ratio affects test scores?
- We use data from California school, which is included in `AER` package.
  - See here for the details: https://www.rdocumentation.org/packages/AER/versions/1.2-6/topics/CASchools

```
# load the the data set in the workspace
data(CASchools)
```

- Use `class()` function to see `CASchools` is `data.frame` object.

```
class(CASchools)
```

```
## [1] "data.frame"
```

- We take 2 steps for the analysis.

  - Step 1: Look at data (descriptive analysis)
  - Step 2: Run regression

9.1.3 Step 1: Descriptive analysis

- It is always important to grasp your data before running regression.
- `head()` function give you a first overview of the data.

```
head(CASchools)
```

```
##   district                          school  county grades students
## 1    75119              Sunol Glen Unified Alameda  KK-08      195
## 2    61499            Manzanita Elementary   Butte  KK-08      240
## 3    61549     Thermalito Union Elementary   Butte  KK-08     1550
## 4    61457 Golden Feather Union Elementary   Butte  KK-08      243
## 5    61523        Palermo Union Elementary   Butte  KK-08     1335
## 6    62042         Burrel Union Elementary  Fresno  KK-08      137
##   teachers calworks   lunch computer expenditure    income   english  read
## 1    10.90   0.5102  2.0408       67    6384.911 22.690001  0.000000 691.6
## 2    11.15  15.4167 47.9167      101    5099.381  9.824000  4.583333 660.5
## 3    82.90  55.0323 76.3226      169    5501.955  8.978000 30.000002 636.3
## 4    14.00  36.4754 77.0492       85    7101.831  8.978000  0.000000 651.9
## 5    71.50  33.1086 78.4270      171    5235.988  9.080333 13.857677 641.8
## 6     6.40  12.3188 86.9565       25    5580.147 10.415000 12.408759 605.7
##    math
## 1 690.0
## 2 661.9
## 3 650.9
## 4 643.5
## 5 639.9
## 6 605.4
```

- Alternatively, you can use `browse()` to see the entire dataset in browser window.

9.1.3.1 Create variables

- Create several variables that are needed for the analysis.
- We use `dplyr` for this purpose.

```
CASchools %>%
  mutate( STR = students / teachers ) %>%
  mutate( score = (read + math) / 2 ) -> CASchools
```

9.1.3.2 Descriptive statistics

- There are several ways to show descriptive statistics
- The standard one is to use `summary()` function

```
summary(CASchools)
```

```
##    district           school                  county      grades
##  Length:420         Length:420         Sonoma     : 29   KK-06: 61
##  Class :character   Class :character   Kern       : 27   KK-08:359
##  Mode  :character   Mode  :character   Los Angeles: 27
##                                        Tulare     : 24
##                                        San Diego  : 21
##                                        Santa Clara: 20
```

```
##                                         (Other)    :272
##     students          teachers          calworks          lunch
##  Min.   :   81.0   Min.   :   4.85   Min.   : 0.000   Min.   :  0.00
##  1st Qu.:  379.0   1st Qu.:  19.66   1st Qu.: 4.395   1st Qu.: 23.28
##  Median :  950.5   Median :  48.56   Median :10.520   Median : 41.75
##  Mean   : 2628.8   Mean   : 129.07   Mean   :13.246   Mean   : 44.71
##  3rd Qu.: 3008.0   3rd Qu.: 146.35   3rd Qu.:18.981   3rd Qu.: 66.86
##  Max.   :27176.0   Max.   :1429.00   Max.   :78.994   Max.   :100.00
##
##     computer       expenditure       income          english
##  Min.   :   0.0   Min.   :3926   Min.   : 5.335   Min.   : 0.000
##  1st Qu.:  46.0   1st Qu.:4906   1st Qu.:10.639   1st Qu.: 1.941
##  Median : 117.5   Median :5215   Median :13.728   Median : 8.778
##  Mean   : 303.4   Mean   :5312   Mean   :15.317   Mean   :15.768
##  3rd Qu.: 375.2   3rd Qu.:5601   3rd Qu.:17.629   3rd Qu.:22.970
##  Max.   :3324.0   Max.   :7712   Max.   :55.328   Max.   :85.540
##
##       read            math            STR            score
##  Min.   :604.5   Min.   :605.4   Min.   :14.00   Min.   :605.5
##  1st Qu.:640.4   1st Qu.:639.4   1st Qu.:18.58   1st Qu.:640.0
##  Median :655.8   Median :652.5   Median :19.72   Median :654.5
##  Mean   :655.0   Mean   :653.3   Mean   :19.64   Mean   :654.2
##  3rd Qu.:668.7   3rd Qu.:665.9   3rd Qu.:20.87   3rd Qu.:666.7
##  Max.   :704.0   Max.   :709.5   Max.   :25.80   Max.   :706.8
##
```

- This returns the desriptive statistics for all the variables in dataframe.
- You can combine this with `dplyr::select`

```
CASchools %>%
  select(STR, score) %>%
  summary()
```

```
##       STR            score
##  Min.   :14.00   Min.   :605.5
##  1st Qu.:18.58   1st Qu.:640.0
##  Median :19.72   Median :654.5
##  Mean   :19.64   Mean   :654.2
##  3rd Qu.:20.87   3rd Qu.:666.7
##  Max.   :25.80   Max.   :706.8
```

- You can do a bit lengthly thing manually like this.

```
# compute sample averages of STR and score
avg_STR <- mean(CASchools$STR)
avg_score <- mean(CASchools$score)

# compute sample standard deviations of STR and score
sd_STR <- sd(CASchools$STR)
sd_score <- sd(CASchools$score)

# set up a vector of percentiles and compute the quantiles
quantiles <- c(0.10, 0.25, 0.4, 0.5, 0.6, 0.75, 0.9)
quant_STR <- quantile(CASchools$STR, quantiles)
quant_score <- quantile(CASchools$score, quantiles)
```

```
# gather everything in a data.frame
DistributionSummary <- data.frame(Average = c(avg_STR, avg_score),
                                  StandardDeviation = c(sd_STR, sd_score),
                                  quantile = rbind(quant_STR, quant_score))

# print the summary to the console
DistributionSummary
```

```
##               Average StandardDeviation quantile.10. quantile.25.
## quant_STR    19.64043          1.891812      17.3486     18.58236
## quant_score 654.15655         19.053347     630.3950    640.05000
##             quantile.40. quantile.50. quantile.60. quantile.75.
## quant_STR       19.26618     19.72321      20.0783     20.87181
## quant_score    649.06999    654.45000     659.4000    666.66249
##             quantile.90.
## quant_STR       21.86741
## quant_score    678.85999
```

- My personal favorite is to use `stargazer` function.

```
stargazer(CASchools, type = "text")
```

```
##
## ================================================================================
## Statistic    N    Mean    St. Dev.     Min    Pctl(25)  Pctl(75)     Max
## --------------------------------------------------------------------------------
## students    420 2,628.793 3,913.105    81        379      3,008    27,176
## teachers    420  129.067   187.913    4.850    19.662    146.350  1,429.000
## calworks    420  13.246    11.455     0.000     4.395    18.981    78.994
## lunch       420  44.705    27.123     0.000    23.282    66.865    100.000
## computer    420  303.383   441.341      0        46       375.2     3,324
## expenditure 420 5,312.408  633.937  3,926.070 4,906.180 5,601.401 7,711.507
## income      420  15.317     7.226     5.335    10.639    17.629    55.328
## english     420  15.768    18.286       0        1.9      23.0       86
## read        420  654.970   20.108    604.500   640.400   668.725   704.000
## math        420  653.343   18.754      605      639.4     665.8      710
## STR         420  19.640     1.892    14.000    18.582    20.872    25.800
## score       420  654.157   19.053    605.550   640.050   666.662   706.750
## --------------------------------------------------------------------------------
```

- You can choose summary statistics you want to report.

```
CASchools %>%
  stargazer( type = "text", summary.stat = c("n", "p75", "sd") )
```

```
##
## ==================================
## Statistic    N  Pctl(75)  St. Dev.
## ----------------------------------
## students    420   3,008   3,913.105
## teachers    420  146.350   187.913
## calworks    420  18.981    11.455
## lunch       420  66.865    27.123
## computer    420   375.2    441.341
## expenditure 420 5,601.401  633.937
## income      420  17.629     7.226
```

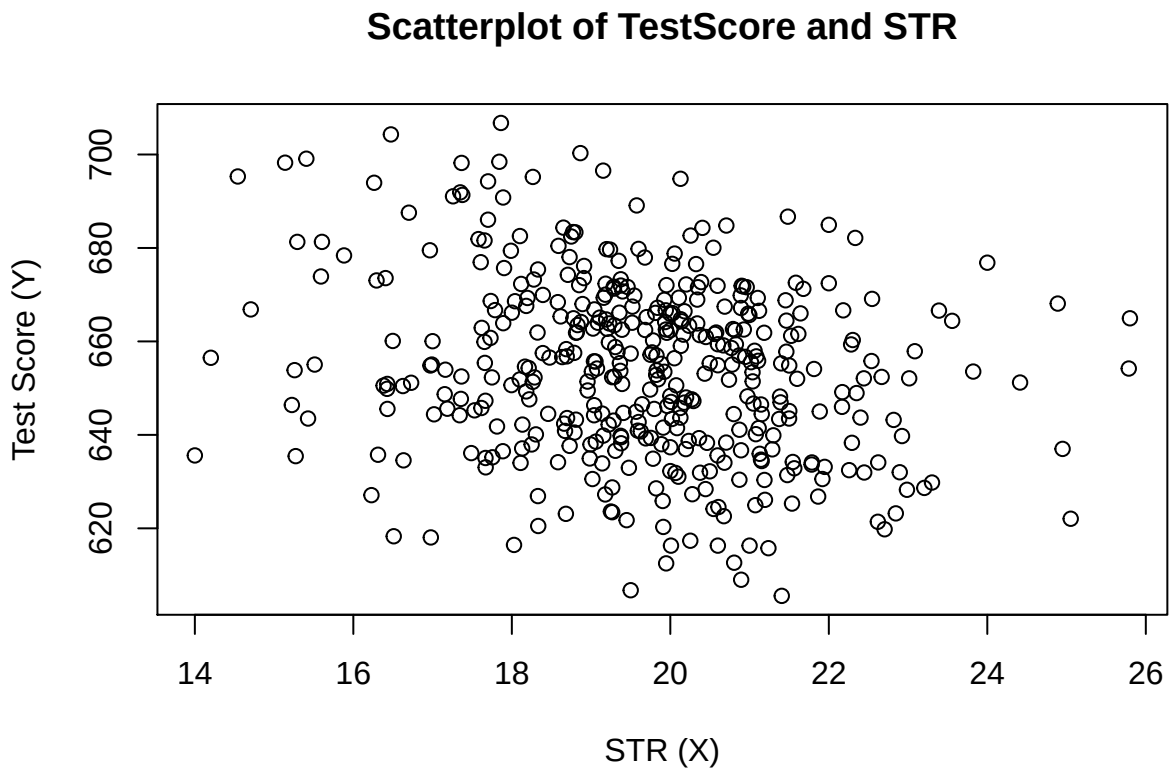
```
## english     420   23.0     18.286
## read        420  668.725   20.108
## math        420   665.8    18.754
## STR         420  20.872     1.892
## score       420  666.662   19.053
## ----------------------------------
```

- See https://www.jakeruss.com/cheatsheets/stargazer/#the-default-summary-statistics-table for the details.
- We will use `stargazer` to report regression results.

9.1.3.3 Scatter plot

- Let's see how test score and student-teacher-ratio is correlated.

```
plot(score ~ STR,
     data = CASchools,
     main = "Scatterplot of TestScore and STR",
     xlab = "STR (X)",
     ylab = "Test Score (Y)")
```



- Use `cor()` to compute the correlation between two numeric vectors.

```
cor(CASchools$STR, CASchools$score)
```

```
## [1] -0.2263627
```

9.1.4 Step 2: Run regression

9.1.4.1 Simple linear regression

- We use `lm()` function to run linear regression
- First, consider the simple linear regression

$$score_i = \beta_0 + \beta_1 size_i + \epsilon_i$$

  where $size_i$ is the class size (student-teacher-ratio).
  - From now on we call student-teacher-ratio (STR) class size.
- To run this regression, we use `lm`

```
# First, we rename the variable `STR`
CASchools %>%
  dplyr::rename( size = STR) -> CASchools

# Run regression and save results in the varaiable `model1_summary`
model1_summary <- lm( score ~ size, data = CASchools)

# See the results
summary(model1_summary)
```

```
##
## Call:
## lm(formula = score ~ size, data = CASchools)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -47.727 -14.251   0.483  12.822  48.540
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 698.9329     9.4675  73.825  < 2e-16 ***
## size         -2.2798     0.4798  -4.751 2.78e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.58 on 418 degrees of freedom
## Multiple R-squared:  0.05124,    Adjusted R-squared:  0.04897
## F-statistic: 22.58 on 1 and 418 DF,  p-value: 2.783e-06
```

- Interpretations
  - An increase of one student per teacher leads to 2.2 point decrease in test scores.
  - p value is very small. The effect of the class size on test score is significant. Note: Be careful. These standard errors are NOT heteroskedasiticity robust. We will come back to this point soon.
  - $R^2 = 0.051$, implying that 5.1% of the variance of the dependent variable is explained by the model.
- You can add more variable in the regression (will see this soon)

9.1.4.2 Correction of Robust standard error

- We use `vcovHC()` function, a partof the package `sandwich`, to obtain the robust standard errors.
  - The package `sandwich` is automatically loaded if you load `AER` package.

```
# compute heteroskedasticity-robust standard errors
vcov <- vcovHC(model1_summary, type = "HC1")
```

```r
# get standard error: the square root of the diagonal element in vcov
robust_se <- sqrt(diag(vcov))
robust_se
```

```
## (Intercept)        size
##  10.3643617   0.5194893
```

- Notice that robust standard errors are larger than the one we obtained from `lm`!
- How to combine the robust standard errors with the original summary? Use `coeftest()` from the package `lmtest`

```r
# load `lmtest`
library(lmtest)

# Combine robust standard errors
coeftest(model1_summary, vcov. = vcov)
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 698.93295   10.36436 67.4362 < 2.2e-16 ***
## size         -2.27981    0.51949 -4.3886 1.447e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.1.4.3 Report by Stargazer

- `stargazer` is useful to show the regression result.

```r
# load
library(stargazer)

# Create output by stargazer
stargazer::stargazer(model1_summary, type ="text")
```

```
##
## ================================================
##                         Dependent variable:
##                     ---------------------------
##                                  score
## ------------------------------------------------
## size                            -2.280***
##                                  (0.480)
##
## Constant                       698.933***
##                                  (9.467)
##
## ------------------------------------------------
## Observations                      420
## R2                                0.051
## Adjusted R2                       0.049
## Residual Std. Error         18.581 (df = 418)
## F Statistic             22.575*** (df = 1; 418)
## ================================================
## Note:                *p<0.1; **p<0.05; ***p<0.01
```

- Use robust standard errors in stargazer output

```
# Prepare robust standard errors in list
rob_se <- list( sqrt(diag(vcovHC(model1_summary, type = "HC1") ) ) )

# generate regression table.

stargazer( model1_summary,
           se = rob_se,
           type = "text")
```

```
##
## ================================================
##                          Dependent variable:
##                      ---------------------------
##                                  score
## ------------------------------------------------
## size                            -2.280***
##                                  (0.519)
##
## Constant                       698.933***
##                                  (10.364)
##
## ------------------------------------------------
## Observations                      420
## R2                               0.051
## Adjusted R2                      0.049
## Residual Std. Error       18.581 (df = 418)
## F Statistic           22.575*** (df = 1; 418)
## ================================================
## Note:                *p<0.1; **p<0.05; ***p<0.01
```

9.1.4.4 Full results

Taken from https://www.econometrics-with-r.org/7-6-analysis-of-the-test-score-data-set.html

```
# load the stargazer library

# estimate different model specifications
spec1 <- lm(score ~ size, data = CASchools)
spec2 <- lm(score ~ size + english, data = CASchools)
spec3 <- lm(score ~ size + english + lunch, data = CASchools)
spec4 <- lm(score ~ size + english + calworks, data = CASchools)
spec5 <- lm(score ~ size + english + lunch + calworks, data = CASchools)

# gather robust standard errors in a list
rob_se <- list(sqrt(diag(vcovHC(spec1, type = "HC1"))),
               sqrt(diag(vcovHC(spec2, type = "HC1"))),
               sqrt(diag(vcovHC(spec3, type = "HC1"))),
               sqrt(diag(vcovHC(spec4, type = "HC1"))),
               sqrt(diag(vcovHC(spec5, type = "HC1"))))

# generate a LaTeX table using stargazer
stargazer(spec1, spec2, spec3, spec4, spec5,
          se = rob_se,
          digits = 3,
```

```
          header = F,
          column.labels = c("(I)", "(II)", "(III)", "(IV)", "(V)"),
          type ="text",
          keep.stat = c("N", "adj.rsq"))
```

```
##
## ===============================================================
##                               Dependent variable:
##              -----------------------------------------------
##                                     score
##                 (I)        (II)      (III)       (IV)       (V)
##                 (1)        (2)        (3)        (4)        (5)
## ---------------------------------------------------------------
## size         -2.280***   -1.101**  -0.998***  -1.308***  -1.014***
##               (0.519)    (0.433)    (0.270)    (0.339)    (0.269)
##
## english                 -0.650***  -0.122***  -0.488***  -0.130***
##                          (0.031)    (0.033)    (0.030)    (0.036)
##
## lunch                              -0.547***             -0.529***
##                                     (0.024)               (0.038)
##
## calworks                                      -0.790***    -0.048
##                                                (0.068)    (0.059)
##
## Constant     698.933*** 686.032*** 700.150*** 697.999*** 700.392***
##               (10.364)   (8.728)    (5.568)    (6.920)    (5.537)
##
## ---------------------------------------------------------------
## Observations    420        420        420        420        420
## Adjusted R2    0.049      0.424      0.773      0.626      0.773
## ===============================================================
## Note:                                   *p<0.1; **p<0.05; ***p<0.01
```

- The coefficient on the class size decreases as we add more explantory variables. Can you explain why? (Hint: omitted variable bias)