Introduction to R

Daniel Cullen
Department of Economics
University of California, Santa Barbara

September 25, 2018

INTRODUCTION

LEARNING OBJECTIVES

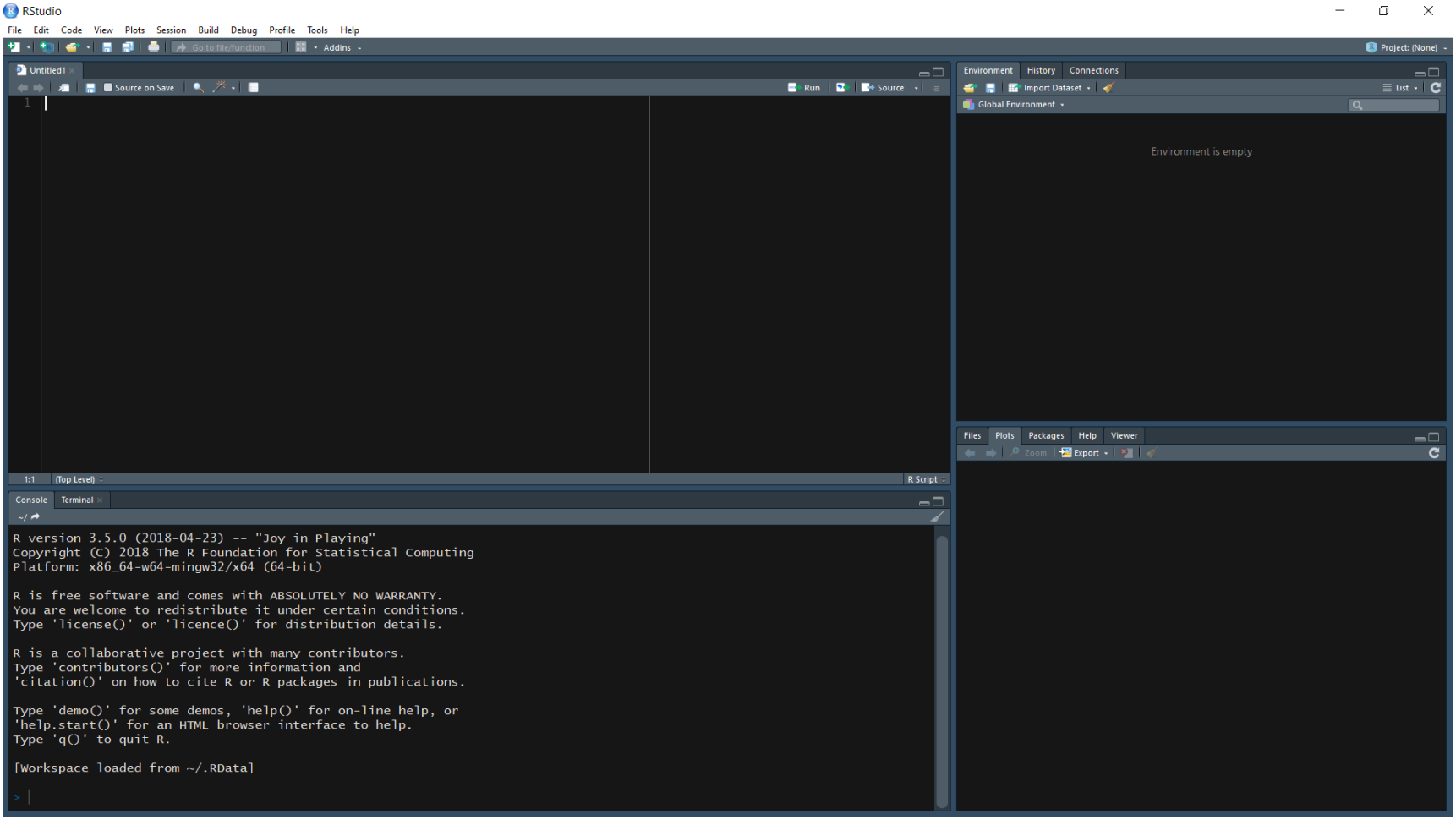
- Become familiar with RStudio
- Understand R syntax and data structures
- Visualize and summarize data using R

R AND RSTUDIO

What is R?
R is a programming language for statistical computing and graphics. R is effective at handling data, performing data analysis, and creating visuals to summarize data.

What is RStudio?
RStudio is an open-source Integrated Development Environment (IDE). Rstudio provides an environment to make using R easier.

RSTUDIO


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1
Source on Save
Run
Source
Environment History Connections
Import Dataset
Global Environment
Environment is empty
Files Plots Packages Help Viewer
Export
Console Terminal
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[Workspace loaded from ~/.RData]

RSTUDIO

The interface of R Studio is split into four parts

- R Console: This area shows the output of code you run and you can directly write codes in the console.
- R Script: The space to write code and save to file. To run this code, select the lines of code and press Ctrl + Enter or click on the Run button at the top right corner of R Script.
- R environment: Displays the set of external elements added, including data, variables, functions, etc.
- Files/Plots/Packages/Viewer: Display the graphs created during data analysis.

ORGANIZING YOUR WORKING DIRECTORY

Check current working directory:

```
getwd()
```

Change working directory:

```
setwd("directory")
```

To organize your working directory for a particular analysis, you should separate the original data (raw data) from produced datasets. For instance, you may want to create a `raw_data/` directory within your working directory that stores the raw data, and have a `produced_data/` directory for intermediate datasets and a `figures/` directory for the plots you generate.

USING R

There are two main ways of interacting with R in RStudio using the console or using the script editor. Best practice is to use the script editor and save the script so you have a record of the commands you ran and can reproduce the output. It is also recommended to thoroughly comment the code using `#`.

PACKAGES

To install a package type:

```
install.packages("package name")
```

To load package type:

```
library("package name")
```

ASSIGNMENT OPERATOR

To assign values to variables use the assignment operator, `<-`. We can assign the value of `5` to `x` by typing:

```
x <- 5
```

We can create another variable, `y`, and set it equal to `2`

```
y <- 2
```

Then we can create a third variable, `z`, and set it equal to the sum of `x` and `y`

```
z <- x + y
```

We can see the result by typing `z` in the console:

```
>z
[1] 7
```

DATA TYPES

R contains six data types:

- `numeric` : for any numeric value
- `character` : for text values
- `integer` : for integer numbers
- `logical` : for `TRUE` and `FALSE`
- `complex` : for complex numbers with real and imaginary parts
- `raw` : holds raw bytes as hex digits

DATA STRUCTURES

Common data structures in R are:

- Vectors `c()` : an ordered list of R objects, indexed by an integer beginning at 1
- Factors `factor` : similar to a vector but each element is categorical (set number of levels)
- Matrix `matrix` : similar to a vector but indexed by two integers
- Arrays `array` : similar to a matrix but can have more than two dimensions
- Lists `list` : similar to a vector, but elements do not need to be the same type
- Data Frames `data.frame` : like a matrix but does not assume all columns have the same type

VECTORS

A vector is a ordered collection of values of the same data type. You can create a vector using the `c()` function which concatenates elements.

```
> c(1, 2, 3, 4, 5)
[1] 1 2 3 4 5
> c("a", "b", "c", "d", "e")
[1] "a" "b" "c" "d" "e"
> c(T, T, F, F, T)
[1]  TRUE TRUE FALSE FALSE TRUE
```

VECTORS

You can create a vector of a sequence using the `:` symbol or the `seq()` function

```
> 1:5
[1] 1 2 3 4 5
> 5:1
[1] 5 4 3 2 1
> seq(1, 5)
[1] 1 2 3 4 5
> seq(1, 5, by = 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

FACTORS

`factor()` transforms a vector into a factor. A factor can also be ordered with the option `ordered = T` or the function `ordered()`.

```
> factor(c("yes", "no", "undecided", "no",
           "yes", "undecided", "no"))
[1] yes no undecided no yes undecided no
Levels: no undecided yes
> factor(c("yes", "no", "undecided", "no",
   "yes", "undecided", "no"), ordered = T)
[1] yes no undecided no yes undecided no
Levels: no < undecided < yes
> ordered(c("yes", "no", "undecided", "no",
      "yes", "undecided", "no"))
[1] yes no undecided no yes undecided no
Levels: no < undecided < yes
```

MATRIX

`matrix()` creates a matrix of data in which you enter a vector of data, the number of rows and/or columns and you can specify if you want R to read your vector by row or by column (the default option).

```
> matrix(data = NA, nrow = 5, ncol = 5, byrow
    = T)
     [,1] [,2] [,3] [,4] [,5]
[1,]   NA   NA   NA   NA   NA
[2,]   NA   NA   NA   NA   NA
[3,]   NA   NA   NA   NA   NA
[4,]   NA   NA   NA   NA   NA
[5,]   NA   NA   NA   NA   NA
```

MATRIX

`matrix()` creates a matrix of data in which you enter a vector of data, the number of rows and/or columns and you can specify if you want R to read your vector by row or by column (the default option).

```
> matrix(data = 1:15, nrow = 5, ncol = 5,
    byrow = T)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]    1    2    3    4    5
[5,]    6    7    8    9   10
```

MATRIX

`matrix()` creates a matrix of data in which you enter a vector of data, the number of rows and/or columns and you can specify if you want R to read your vector by row or by column (the default option).

```
> matrix(data = 1:15, nrow = 5, ncol = 5,
    byrow = F)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11    1    6
[2,]    2    7   12    2    7
[3,]    3    8   13    3    8
[4,]    4    9   14    4    9
[5,]    5   10   15    5   10
```

MATRIX

`cbind()` and `rbind()` combine vectors into matrices column by column or row by row :

```
> v1 <- 1:4
> v2 <- 4:1
> v2
[1] 4 3 2 1
> cbind(v1,v2)
     v1 v2
[1,]  1  4
[2,]  2  3
[3,]  3  2
[4,]  4  1
> rbind(v1,v2)
   [,1] [,2] [,3] [,4]
v1    1    2    3    4
v2    4    3    2    1
```

ARRAYS

An array is composed of n dimensions where each dimension is a vector of R objects of the same type.

```
z <- array(1:27, dim = c(3, 3, 3))
> z
, , 1

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

, , 2

[,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

[,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

ARRAYS

R arrays are accessed by integer index. The third dimension of a 3 by 3 array is:

```
> z[,,3]
[,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

Specifying two of the three dimensions returns an array on one dimension.

```
> z[,3,3]
[1] 25 26 27
```

Specifying three of three dimension returns an element of the 3 by 3 by 3 array.

```
> z[3,3,3]
[1] 27
```

LISTS

A list is a collection of R objects. `list()` creates a list. `unlist()` transform a list into a vector. The objects in a list do not have to be of the same type or length.

```
> x <- c(1:4)
> y <- FALSE
> z <- matrix(c(1:4),nrow=2,ncol=2)
> myList <- list(x,y,z)
> myList
 [[1]]
[1] 1 2 3 4

 [[2]]
[1] FALSE

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

DATA FRAMES

A dataframe is a list of variables/vectors of the same length.

```
> v1 <- 1:5
> v2 <- c(T, T, F, F, T)
> df <-  data.frame(v1, v2)
> print(df)
  v1    v2
1  1  TRUE
2  2  TRUE
3  3 FALSE
4  4 FALSE
5  5  TRUE
```

For Loops

For loops allow us to repeat an action multiple times. For example we can print the square of every number from 1 to 5:

```
for (i in 1:5){
  print(i^2)
}
```

For Loops

We can also put the results of our loop in a list. First we create a vector of zeros, then we fill in with values

```
n <-  5
x <- rep(0,n)
for (i in 1:n){
  x[i] <- i^2
}

>print(x)
[1]  1  4  9 16 25
```

For Loops

We can use a loop to find the expected value of rolling a die 1000 times.

```
nsides  <- 6
ntrials <- 1000
trials  <- rep(0, ntrials)
for (i in 1:ntrials){
  trials[i] <- sample(1:nsides, 1)
}

> mean(trials)
[1] 3.496
```

FUNCTIONS

Functions are a key feature in R. They are sections of code that take input, process it, and return a results.
A function consists of the name of the function followed by parentheses:

```
function_name(input)
```

The input are called arguments. The arguments are either objects on which the function performs a task or options that alter the function.

BASIC FUNCTIONS

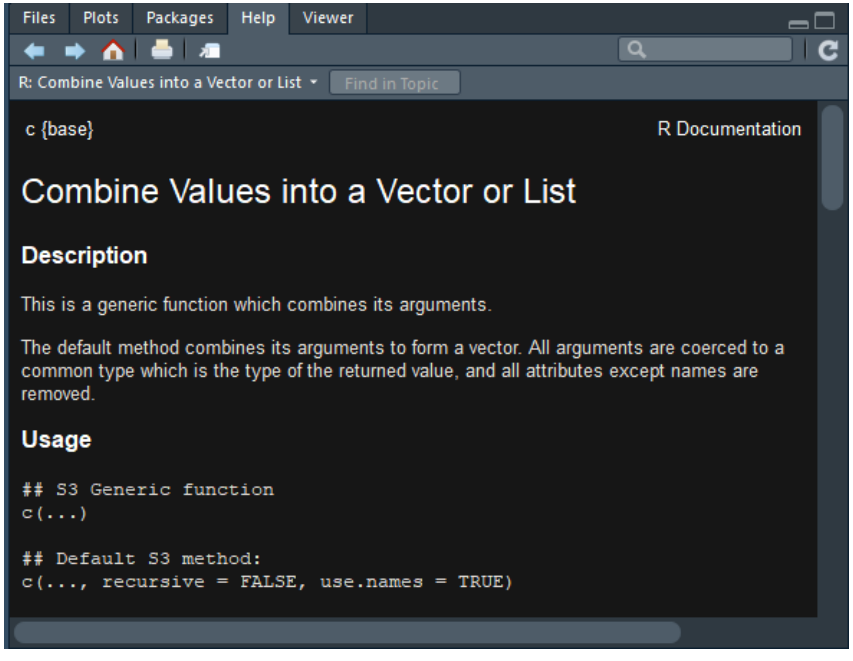
We have already seen some basic function. `getwd()` is a function to display the current working directory and takes no arguments. We also used the `c()` function to combine data into vectors. The arguments `c()` takes are a collection of numbers, characters, or strings. You can also use the function to add elements to an existing vector.

```
> country  <- c("Canada", "United States",
    "Mexico")
> country  <- c(country, "France")
> country
[1] "Canada" "United States" "Mexico" "France"
```

FUNCTION HELP

To find information about a function use the `?` followed by the name of the function. This will open the help manual in the bottom right panel of R studio that provides a description of the function and its arguments.

```
> ?c
```



USER-DEFINED FUNCTIONS

Functions are a useful tool when there is a task you need to repeat multiple times.

```
function_name <- function(argument1, argument2) {
  Code that does something
  return(something)
}
```

User-defined Function Example

Create a function to find the square of a number.

```
squared <- function(x) {
  square <- x * x
  return(square)
}
```

Now we can use our created function to find the square of any number

```
  > squared(5)
  [1] 25
  > squared(13)
  [1] 169
```

LOADING DATA

R can import data of various file types. A common file type for data is a comma-separated values (`.csv`) file. To import a csv file and assign it to a data frame named `df` type

```
df <- read.csv("cps_2016.csv")
```

VIEWING DATA

There are various ways to inspect a data frame, such as:

- `str(df)` gives a very brief description of the data
- `names(df)` gives the name of each variables
- `summary(df)` gives some very basic summary statistics for each variable
- `head(df)` shows the first few rows
- `tail(df)` shows the last few rows.

```
View(df)
head(df, n = 20) # n = 20 prints the first 20
   lines in the R console
```

VIEWING DATA

Other useful functions to inspect a data frame:

- `length(df)` gives number of variables
- `length(df$var)` gives number of observations for a variable, `var`
- `dim(df)` gives the dimensions of the data frame
- `nrow(df)` gives some very basic summary statistics for each variable
- `ncol(df)` shows the first few rows

```
> # View dimensions of data
> length(df)
[1] 13
> length(df$incwage)
[1] 185487
> dim(df)
[1] 185487     13
```

ROW AND COLUMN BIND

A row or column can be added to a data frame using the `rbind` or `cbind` commands.

```
mydata <- cbind(mydata, myVector)
mydata <- rbind(mydata, myVector)
```

Creating and Removing Variables

A new variable can be added to a data frame

```
mydata$newVar <- oldvar
```

A variable can be deleted from a dataset by assigning NULL to the variable

```
mydata$x <- NULL
```

RENAMING VARIABLES

To rename a variable you must redefine the vector of names in a data frame

```
df <- data.frame(x = 1:10, y = 11:20)

> names(df)
[1] "x" "y"

names(df) <- c("Var1", "Var2")
```

Subsetting Data

The `subset` command is used to create a subset of a data frame. The first argument is the name of the dataset, the second argument is a logical condition which says what to include in the new dataset, and the last argument is the list of variable which will be included in the new dataset.

```
N <- 100
x1 <- rnorm(N)
x2 <- 1 + rnorm(N) + x1
x3 <- rnorm(N) + x2
mydat <- data.frame(x1, x2, x3)
subset(x = mydat, subset = x1 > 0 & x2 < 0,
   select = c(x1, x2))
```

RESHAPING DATA

The `reshape()` command reshapes a dataset in a wide or long format.

```
> country  <- c("Canada", "United States", "Mexico")
> gdp_2000 <- c(1, 2, 3)
> gdp_2010 <- c(2, 4, 6)
> gdp_data <- data.frame(country, gdp_2000, gdp_2010)
> gdp_data # wide format
       country gdp_2000 gdp_2010
1       Canada        1        2
2 United States        2        4
3       Mexico        3        6
>
> # long format
> gdp_long <- reshape(data = gdp_data, varying = list(2:3) , v.names = "gdp",
    direction = "long")
> gdp_long
         country time gdp id
1.1       Canada    1   1  1
2.1 United States    1   2  2
3.1       Mexico    1   3  3
1.2       Canada    2   2  1
2.2 United States    2   4  2
3.2       Mexico    2   6  3
```

MERGING DATA

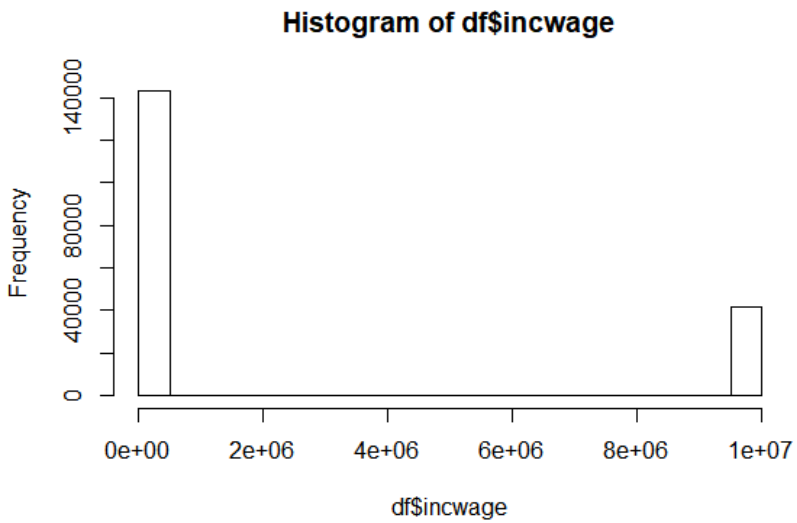
The `merge()` command can be used to combine two data frames.

```
> capitals <- data.frame(country = c("Canada", "United States", "Mexico"),
+                        capital = c("Ottawa", "Washington DC", "Mexico City"))
> capitals
       country       capital
1       Canada        Ottawa
2 United States Washington DC
3       Mexico   Mexico City
> leaders  <- data.frame(country = c("Canada", "United States", "Mexico"),
+                         leader = c("Justin Trudeau", "Donald Trump", "Enrique Pena
    Nieto"))
> leaders
       country             leader
1       Canada     Justin Trudeau
2 United States       Donald Trump
3       Mexico Enrique Pena Nieto
> final <- merge(capitals, leaders, by = "country")
> final
       country       capital             leader
1       Canada        Ottawa     Justin Trudeau
2       Mexico   Mexico City Enrique Pena Nieto
3 United States Washington DC       Donald Trump
```

LOOKING FOR MISSING DATA

To view a histogram of a variable in your data frame use the `hist()` command.

```
hist(df$incwage)
```



REMOVING MISSING DATA

In the CPS data missing income data is coded as 9999999 we can subset the data to exclude all observation with an income equal to 9999999.

```
df1 <- subset(df,
    subset = incwage !
    = 9999999)
hist(df1$incwage)
```