R Training Session

Ben Berger

January 27, 2021

Who am I?

Ben Berger

- G3, Public Policy Ph.D.
- Research Interests: Health care, innovation, drug regulation
- Hometown: Harrisburg, PA
- Hobbies: Board games, photography, eating spicy food
- R experience: 7+ years

Today

- Introduce R.
- Discuss programming style.
- Review important `Base R` concepts and syntax.
- Review select `tidyverse` commands.
- Do some exercises along the way.
- Let's start a new project now: call it `r_tutorial`.

R

- Programming language designed for statistical analysis.
- Free and open source since 1995.
- Many useful packages included with R installation.
- Lots more on CRAN!
- Tidyverse: popular collection of packages designed to work well together.

R vs. Stata

- R not built around single dataset.
- Allows for many types and instances of objects.
- Free and open source.
- Large and diverse array of packages.

R Packages

- Dates: `lubridate`
- Databases: `DBI`, `odbc`, `RMySQL`
- Web: `httr`, `XML`, `jsonlite`
- Modeling: `survival`, `glmnet`, `randomForest`, `caret`
- HTML and PDF documents: `rmarkdown`, `stargazer`, `xtable`
- Visualization: `ggplot2`, `leaflet`, `network3D`, `maps`
- Interactive web apps: R Shiny

Programming Style

On Programming Style

Your computer doesn't care about your programming style, but it helps to be consistent and follow the programming conventions.

Future users (including yourself!) will appreciate understandable code.

Which of these commands is easier to read?

```
# 1.
y<-2 ^ 2+3 *( 4 +2 )

# 2.
y <- 2^2 + 3 * (4 + 2)
```

On Programming Style

- Most important factor is *readability*.
- Best practices:
    - Follow the style guide.
    - Comment your code clearly and extensively.
    - Write code that is understandable!

Some Problematic Code

- This script plots the opening price of Tesla stock in each day of December 2020.
- What are some (stylistic) things wrong with this code?

```
# Read daily Tesla stock price data.
muskData.df<-read.csv( "TSLA.csv"  )

# Keep only dates in December 2020.
library(tidyverse)
X_Æ_A_12<- filter(muskData.df,
Date>="2020-12-01" & Date <= '2020-12-31')

# Plot
ggplot(X_Æ_A_12,aes(x=Date,y=Open)) +
  geom_col()
```

Some Problematic Code

- The object names are very confusing. What is an `X_Æ_A_12` anyway?
- `library(tidyverse)` isn't at the top of the script!
- Inconsistent use of single and double quotes.
- Inconsistent spacing & doesn't follow style guide.
- Comments are mostly okay, but doesn't explain that it plots opening prices.

Some Better Code

```r
library(tidyverse)
library(lubridate)

# Read daily Tesla stock price data.
tesla_price <- read_csv("TSLA.csv")

# Keep only dates in December 2020.
tesla_price_dec2020 <- tesla_price %>%
  filter(year(Date) == 2020 & month(Date) == 12)

# Plot opening prices on each day.
ggplot(tesla_price_dec2020,
       aes(x = Date, y = Open)) +
  geom_col()
```

Introduction to R

Basic Data Types

All objects in R are created by combining a few basic data types.

```r
# Numeric
479
-0.3

# Character
"Hello, World!"
"479"
"TRUE"
"Dr. Jill Biden"

# Logical
TRUE
FALSE
```

Vectors

String together data elements of the same type using `c()` to create vectors.

```
# Numeric Vector
v_num <- c(1, 0.2, -30)
# Character Vector
v_char <- c("Larry", "Moe", "Curly")
# Logical Vector
v_lgl <- c(TRUE, FALSE, FALSE)
```

`NA` (missing value) is a special element which can be in any type of vector.

```
# Numeric Vector
c(1, 2, 3, NA)
# Character Vector
c("Larry", NA, "Curly")
```

Subset Vectors

Subset vectors using single square brackets [].

```
# Subset using indices
v_char[c(1,3)]
```

```
[1] "Larry" "Curly"
```

```
v_char[1:3]
```

```
[1] "Larry" "Moe"   "Curly"
```

Subset Vectors

```r
# Recall:
print(v_lgl)
```

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

```r
# Subset using logical vectors
v_char[v_lgl]
```

```
[1] "Larry"
```

```r
v_char[v_num < 0]
```

```
[1] "Curly"
```

Tibbles/Data Frames

Tibbles are “well-behaved” data frames. Tidyverse functions (e.g. `read_xlsx`) all load data into tibble. I will refer to them interchangably, but generally will use tibbles.

```
library(tidyverse)
data_tibble <- tibble(num = v_num,
                      char = v_char,
                      lgl = v_lgl)
print(data_tibble)
```

Squashing Some Bugs

What is this script supposed to do? What is wrong with it?

```
v1 <- 1:3
v2 <- c("a", "b")
my_tibble <- tibble(v1, v2)
```

```
Error: Tibble columns must have compatible sizes.
* Size 3: Existing data.
* Size 2: Column at position 2.
i Only values of size one are recycled.
```

Viewing Data Frames

- You can browse data frames/tibbles by clicking on the object name in the RStudio environment panel OR using `View` function.
- Do not put `View` in your scripts. Like `install.packages` you should input it directly in the console.

Extracting Columns

Two options: `$` or `[[]]`.

```
data_tibble$num
```

```
[1]   1.0   0.2 -30.0
```

```
data_tibble[["num"]]
```

```
[1]   1.0   0.2 -30.0
```

Exercise 1

Exercise 1

Now copy `WEO-2018.xlsx` into your project directory, and load the World Economic Outlook data using `read_excel`.

```
library(tidyverse)
library(readxl)

# Read data
weo <- read_excel("WEO-2018.xlsx")
```

Exercise 1

For this exercise, let's use `summary` to understand the the distribution of population in 1992.

1. Make a new object called `pop1992` by extracting it from `weo` using `$` or `[[]]`.
2. Use the `summary` function to get summary statistics for `pop1992`.
3. Print the first 5 elements of `pop1992`.
4. Lastly, tell me how you could do 1 and 2 on one line.

Exercise 1

```r
# Extract vector from `weo`
pop1992 <- weo$pop1992
pop1992 <- weo[["pop1992"]]
# Summarize
summary(pop1992)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
   0.041    1.956    5.929   31.362   19.421 1171.710       24
```

```r
# First 5 Elements
pop1992[1:5]
```

```
[1]     NA  3.217 26.271 13.459  0.062
```

Functions

Functions allow you to automate common tasks in a more powerful and general way than copy-and-pasting. Writing a function has three big advantages over using copy-and-paste:

- *You can give a function an evocative name that makes your code easier to understand.*
- *As requirements change, you only need to update code in one place, instead of many.*
- *You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).*

(R for Data Science)

Functions

- Functions have three main components: a *name*, *arguments*, and a *body*.
- The *name* is the phrase you use to call the function.
- The *arguments* are the inputs to the function.
- The *body* translates the inputs into an output.

```
function_name <- function(arg_1, arg_2, ...) {
   Function body
}
```

Function Examples

```r
# Define function to square a number
square <- function(x){
  x^2
}
# Alternatively, specify `return`
square <- function(x){
  return(x^2)
}
square(10)
```

```
[1] 100
```

Function Examples

```r
# Define function to calculate standard deviation
my_sd <- function(x, drop_na = F){
  if(drop_na){
    x <- x[!is.na(x)]
  }
  n <- length(x)
  x_bar <- mean(x)
  resid <- x - x_bar
  resid_2 <- resid^2
  var <- sum(resid_2) / (n - 1)
  std_dev <- sqrt(var)
  return(std_dev)
}

my_sd(pop1992, drop_na = T)
```

```
[1] 116.5915
```

Exercise 2

Exercise 2

Replace the `...` to define a new function called `my_mean`. Use the functions `sum` and `length`.

```
my_mean <- function(x){
  ...
}
```

To test your function, run `my_mean(1:5)`. Your function should return 3.

Exercise 2

```r
my_mean <- function(x){
  sum <- sum(x)
  n <- length(x)
  mean <- sum/n
  return(mean)
}
```

Exercise 2

Now redefine `my_mean` to remove NA values from `x` before calculating the mean. Test it in `pop1992` and compare to the output of `mean(pop1992, na.rm = TRUE)`.

Hints:

- Subset vectors using `[]`.
- You can determine which elements of a vector are NAs using `is.na`.
- The "NOT" operator is `!`. E.g. `!is.na(x)` is TRUE when `x` is not NA.

Exercise 2

```
my_mean <- function(x){
  x <- x[!is.na(x)]
  sum <- sum(x)
  n <- length(x)
  mean <- sum/n
  return(mean)
}
```

Formula Notation, T-Tests, and Linear Models

`Base R` includes numerous functions for statistical estimation and testing.

- One sample t-test:
  - `t.test(x, mu)`
  - `mu = 0` by default

R uses a special formula notation for estimation of most tests and models.

- Two sample t-test: D is binary
  - `t.test(y ~ D)`
- Bivariate regression: $Y_i = \beta_0 + \beta_1 X_i + u_i$.
  - `lm(y ~ x, data)`
  - β_0 is always included by default.

Formula Notation, T-Tests, and Linear Models

```r
# Load Motor Trend car data
data(mtcars)
# 1 sample t-test of MPG
t.test(mtcars$mpg, mu = 20)
```

```

	One Sample t-test

data:  mtcars$mpg
t = 0.08506, df = 31, p-value = 0.9328
alternative hypothesis: true mean is not equal to 20
95 percent confidence interval:
 17.91768 22.26357
sample estimates:
mean of x
 20.09062
```

Formula Notation, T-Tests, and Linear Models

```r
# 2 sample t-test of MPG between automatic/manual transmission
t.test(mpg ~ am, data = mtcars)
```

```

    Welch Two Sample t-test

data:  mpg by am
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.280194  -3.209684
sample estimates:
mean in group 0 mean in group 1 
       17.14737        24.39231 
```

Exercise 3

Exercise 3

If you haven't already, load the mtcars data using `data(mtcars)`. Then estimate the bivariate regression of miles per gallon on weight. Interpret for me the coefficient on weight. Is the intercept meaningful in this model?

Hint: View documentation on the dataset by entering `?mtcars` in the console.

Pro tip: `Ctrl-1` and `Ctrl-2` allow you to flip between the editor and console without use of a trackpad.

Exercise 3

```r
# Estimate linear regression of MPG on weight and intercept
m1 <- lm(mpg ~ wt, data = mtcars)
summary(m1)
```

```

Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.5432 -2.3647 -0.1252  1.4096  6.8727 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,	Adjusted R-squared:  0.7446 
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

Formula Notation, T-Tests, and Linear Models

```r
# Extract coefficients
coefs <- coef(m1)
coefs <- m1$coefficients
# Extract standard errors
se <- sqrt(diag(vcov(m1)))
# Obtain predicted values
mpg_predictions <- predict(m1, mtcars)
# Obtain residuals
m1_residuals <- resid(m1)
m1_residuals <- m1$residuals
```

Some Important Base `R` functions to know.

- `ls`: objects in R's memory.
- `dir`: files in current directory.
- `class`: class of an object (i.e. data frame, numeric, character).
- `length`: number of elements in a vector.
- `nrow`: number of rows in a data frame.
- `rep(x, times)` repeat the value `x` (`times` = # of repeats)
- `table`: counts unique values in vector.
- `cor`: correlation between 2+ vectors.
- `lm`: estimate a linear model.

tidyverse and dplyr

- `tidyverse` = `dplyr` + `tidyr` + `readr` + `ggplot2` + more.
- `dplyr` is a package within `tidyverse` that makes working directly with data frames easier.
- Key functions: `select`, `filter`, `slice`, `mutate`, `arrange`, `rename`, `summarize`.
- First argument of each is a dataset.
- Pipe operator %>%: create a chain of verbs to change data.
- Passes previous object into next function as first argument.
- e.g. `data %>% select(name)` = `select(data, name)`.

Subsetting Data Frames

```r
## By Position: (Rows 1 and 3) ##
# dplyr/Tidyverse
slice(data_tibble, c(1, 3))
```

```
# A tibble: 2 x 3
    num char  lgl
  <dbl> <chr> <lgl>
1     1 Larry TRUE
2   -30 Curly FALSE
```

```r
# Base R
data_tibble[c(1, 3),]
```

```
# A tibble: 2 x 3
    num char  lgl
  <dbl> <chr> <lgl>
1     1 Larry TRUE
2   -30 Curly FALSE
```

Subsetting Data Frames

```
## Using logical vector ##
# dplyr/Tidyverse
filter(data_tibble, lgl == TRUE)
```

```
# A tibble: 1 x 3
    num char  lgl
  <dbl> <chr> <lgl>
1     1 Larry TRUE
```

```
# Base R
data_tibble[data_tibble$lgl == TRUE,]
```

```
# A tibble: 1 x 3
    num char  lgl
  <dbl> <chr> <lgl>
1     1 Larry TRUE
```

Selecting Columns

```r
# dplyr/Tidyverse
select(data_tibble, num, char)
```

```
# A tibble: 3 x 2
    num char
  <dbl> <chr>
1   1   Larry
2   0.2 Moe
3 -30   Curly
```

```r
# Base R
data_tibble[, c("num", "char")]
```

```
# A tibble: 3 x 2
    num char
  <dbl> <chr>
1   1   Larry
2   0.2 Moe
3 -30   Curly
```

Adding Columns

```r
# dplyr/Tidyverse
data_tibble <- mutate(data_tibble, new_var = toupper(char))
data_tibble
```

```
# A tibble: 3 x 4
    num char  lgl   new_var
  <dbl> <chr> <lgl> <chr>
1   1   Larry TRUE  LARRY
2   0.2 Moe   FALSE MOE
3 -30   Curly FALSE CURLY
```

Exercise 4

Exercise 4

Let's write a script that computes the mean, median, 10th percentile, 90th percentile, and the standard deviation of GDP per capita in 2017 for each continent.

Which continent has the greatest variance in national GDP per capita?

Start with these steps:

- Select country, continent, and the 2017 columns (`select`).
- Calculate GDP per capita (`mutate`).
- Create a new object called `weo_2017` using the modified data (`<-`).

You will likely want to use the pipe operator (%>%) a couple times. RStudio shortcuts for this are Cmd+Shift+M on Mac and Ctrl+Shift+M on PC!

Exercise 4

```r
# Create new data frame
weo_2017 <- weo %>%
  # Select 2017 columns
  select(country, continent, pop2017, rgdp2017) %>%
  # Calculate GDP per capita
  mutate(rgdppc2017 = rgdp2017 / pop2017)
```

Exercise 4

```
head(weo_2017)
```

```
# A tibble: 6 x 5
  country             continent     pop2017 rgdp2017 rgdppc2017
  <chr>               <chr>           <dbl>    <dbl>      <dbl>
1 Afghanistan         Asia           35.5     63353.      1783.
2 Albania             Europe          2.88    32763.     11392.
3 Algeria             Africa         41.5    576494.     13879.
4 Angola              Africa         28.2    173327.      6151.
5 Antigua and Barbuda North America   0.091    2174.     23893.
6 Argentina           South America  44.1    838221.     19015.
```

Exercise 4

- Now we want to summarize the data separately for each continent. Use the functions `mean`, `median`, `quantile`, and `sd` as well as the `dplyr` functions `group_by` and `summarize`.
- `quantile` takes two arguments to return a particular percentile. Remember you can type `?quantile` in the console to quickly read the function's documentation!

Exercise 4

```r
# Summarize 2017 GDP per Capita
# Mean, Median, 10th & 90th percentiles, Std. Dev.
weo_2017_summary <- weo_2017 %>%
  group_by(continent) %>%
  summarize(mean = mean(rgdppc2017),
            median = median(rgdppc2017),
            perc_10 = quantile(rgdppc2017, 0.1),
            perc_90 = quantile(rgdppc2017, 0.9),
            sd = sd(rgdppc2017))
```

Exercise 4

```r
print(weo_2017_summary)
```

```
# A tibble: 6 x 6
  continent        mean median perc_10 perc_90     sd
  <chr>           <dbl>  <dbl>   <dbl>   <dbl>  <dbl>
1 Africa          5581.  3180.   1173.  13262.  6553.
2 Asia           25913. 15443.   3141.  61415. 27956.
3 Europe         32727. 30082.  11471.  52041. 18019.
4 North America  17672. 14484.   5536.  33446. 13148.
5 Oceania        10939.  5109.   2071.  31274. 13884.
6 South America  14455. 13194.   7737.  21955.  6393.
```

Exercise 5 (Advanced)

Reshaping Data with `tidyr`

- Now let's try to do something new with the `weo` data and plot the evolution of (real) GDP per capita from 1992 to 2017 separately for each continent. Which continent's economies grew the most on average?
- What is the issue with just taking the object `weo`, calculating GDP per capita in each year, and feeding it into `ggplot`?

Reshaping Data with tidyr

```
weo %>%
  select(1:10) %>%
  head
```

```
# A tibble: 6 x 10
  country continent pop1992 pop1993 pop1994 pop1995 pop1996 pop1997
  <chr>   <chr>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1 Afghan~ Asia       NA      NA      NA      NA      NA       NA
2 Albania Europe      3.22    3.20    3.14    3.14    3.17     3.15
3 Algeria Africa     26.3    26.9    27.5    28.1    28.6     29.0
4 Angola  Africa     13.5    13.9    14.3    14.7    15.1     15.6
5 Antigu~ North Am~   0.062   0.063   0.065   0.067   0.068    0.07
6 Argent~ South Am~  33.4    33.9    34.4    34.8    35.2     35.6
# ... with 1 more variable: pop1999 <dbl>
```

Reshaping Data with tidyr

- What is the brute force way of calculate GDP per capita in each year?

```
weo %>%
  mutate(rgdppc1992 = rgdp1992/pop1992,
         rgdppc1993 = rgdp1993/pop1993,
         rgdppc1994 = rgdp1994/pop1994,
         rgdppc1995 = rgdp1995/pop1995,
         rgdppc1996 = rgdp1996/pop1996,
         ...)
```

- What are some issues with this approach?

Reshaping Data with tidyr

- tidyr is another useful package within tidyverse
- We can use tidyr function pivot_longer to reshape this wide data in long form.
- Here, long form means that each country, continent, and year will have its own unique row of data.

```
# Rename each variable in WEO data
# to have "_" between variable name and year.
weo_underscored <- weo %>%
  rename_at(vars(starts_with(c("pop", "rgdp"))),
            ~ str_replace(., "(^[[:alpha:]]*)", "\\1_"))

# Pivot data LONG so that each year and variable is on its own row.
weo_long <- weo_underscored %>%
  pivot_longer(cols = -c("country", "continent"),
               names_to = c("var", "year"),
               names_sep = "_")
```

Reshaping Data with tidyr

Is the data usable yet?

```
weo_long %>%
  filter(country == "United States") %>%
  head
```

```
# A tibble: 6 x 5
  country       continent     var   year  value
  <chr>         <chr>         <chr> <chr> <dbl>
1 United States North America pop   1992   257.
2 United States North America pop   1993   260.
3 United States North America pop   1994   263.
4 United States North America pop   1995   266.
5 United States North America pop   1996   270.
6 United States North America pop   1997   273.
```

Reshaping Data with tidyr

```r
# Pivot data WIDE so that each row has a column for pop and GDP.
weo_final <- weo_long %>%
  pivot_wider(
    names_from = var,
    values_from = value
  ) %>%
  # Calculate GDP per Capita
  mutate(rgdppc = rgdp / pop)
```

Reshaping Data with tidyr

```
weo_final %>%
  filter(country == "United States") %>%
  head
```

```
# A tibble: 6 x 6
  country       continent     year    pop      rgdp rgdppc
  <chr>         <chr>         <chr> <dbl>     <dbl>  <dbl>
1 United States North America 1992   257.  9573371. 37283.
2 United States North America 1993   260.  9836237. 37810.
3 United States North America 1994   263. 10233448. 38862.
4 United States North America 1995   266. 10511642. 39450.
5 United States North America 1996   270. 10910701. 40473.
6 United States North America 1997   273. 11400224. 41786.
```

Finishing Exercise 5

Finally we can use our data! The last steps before plotting are (1.) calculating average GDP/capita for each continent and year and (2.) calculating its percent growth since 1992.

Finishing Exercise 5

```r
weo_average <- weo_final %>%
  group_by(continent, year) %>%
  # Calculate average GDP per capita for each continent and year
  summarize(rgdppc = mean(rgdppc, na.rm = T)) %>%
  # Calculate percent growth since 1992
  mutate(rgdppc_base = rgdppc[year == "1992"],
         rgdppc_growth = 100 * (rgdppc - rgdppc_base) / rgdppc_base
  )
```

Finishing Exercise 5

And now we can plot our data!

```
head(weo_average)
```

```
# A tibble: 6 x 5
# Groups:   continent [1]
  continent year  rgdppc rgdppc_base rgdppc_growth
  <chr>     <chr>  <dbl>       <dbl>         <dbl>
1 Africa    1992   3904.       3904.         0
2 Africa    1993   3843.       3904.        -1.56
3 Africa    1994   3810.       3904.        -2.43
4 Africa    1995   3785.       3904.        -3.07
5 Africa    1996   3887.       3904.        -0.448
6 Africa    1997   4016.       3904.         2.86
```

Finishing Exercise 5

```r
my_plot <- ggplot(weo_average,
                  aes(x = as.integer(year),
                      y = rgdppc_growth,
                      group = continent,
                      color = continent)) +
  geom_line() +
  scale_x_continuous(
    breaks = c(1992, seq(1995, 2020, 5)),
    minor_breaks = NULL
  ) +
  scale_color_discrete(name = "Continent") +
  labs(
    x = "Year",
    y = "Real GDP/capita growth since 1992"
  ) +
  theme_minimal()
```

Finishing Exercise 5