R Winter Assignment (Solutions)

Winter 2020, Pre-Assignment*

Due January 27, 2020

Combined with the assigned primers, this set of exercises will help you get you up and running with doing basic data analysis in R. The Math Camp program will go over the exercise to clarify any points of confusion.[1]

Submission

Do your work in the rstudio.cloud environment described below and submit only the saved `.R` file to the Assignment Page which we will announce we created. More details on how to do this are provided at the beginning and end of this assignment.

Where are we? Where are we headed?

Before you start this practice problem set, you should complete the following RStudio Primers:

1. Visualization Basics
2. Programming Basics
3. Work with Tibbles
4. Isolating Data with dplyr
5. Creating Variables and dataframes

If you are already familiar with R, you can just briefly review these tutorials. After this assignment, we'll be offering several more sessions to cover more tools and concepts.

*This exercise was originally written by Shiro Kuriwaki and Dan Levy. Copyright: CC-BY.

[1] That said, please feel free to contact Shiro (kuriwakig.harvard.edu) if you have any questions in the meantime.

Problem 1: Familiarize with the Style Guide

Learning any language requires following its form and style. Throughout the course, we will be enforcing a style guidelines on how R code should be written. Before writing any code, read and try to internalize Book I ("Analyses") of tidyverse style guide (`https://style.tidyverse.org`), especially chapters 1 and 2.

Code style matters. Following the style guide in programming is like following the rules of proper punctuation in English. Please adhere to the guidelines in this style guide for all subsequent code you write (including this assignment).

Problem 2: Loading a Spreadsheet in RStudio

The interactive windows in the primers got you started in R, but was also restrictive. Most of your data analysis work will involve programming in R on a designated interface called RStudio. Follow the steps below to get set up and load a dataset. The page below contains some screenshots to supplement.

1. **Create a rstudio.cloud account**: Go to `https://rstudio.cloud`. Please create a new account for yourself. You will use this account for math camp, so we advise you use your HKS email.

2. **Sign into the Class Space**: Once you have signed in, join the Math Camp "space" through the access link `https://rstudio.cloud/spaces/43920/join?access_code=gGOt6lQJSmo%2BGOgbIAVueWEVTYJ8Tpq%2FmsER4pI6`. By joining this group, you can access R material shared with the group.

3. **Copy a Project**: In the Projects tab at the tab (expand your browser window is wide enough), go to the Assignment `01_Winter-Assignment` (Figure 1(a)), and click "Start".

4. **Understanding the GUI and R**: It will take 30 seconds to about a full minute for a new window to finish loading (Figure 1(b)). Welcome to RStudio!

   RStudio is a *GUI* (Graphical User Interface) for R. R is not a GUI; it's the program. A GUI allows users to interface with the software using graphical aids like buttons and tabs. Most daily software is a GUI (like Microsoft Word or the Control Panel). RStudio is also an *IDE* (Integrated Development Environment) meaning that it provides shortcuts to advanced tools for working with R.

   The *Console* is the core window through which you can observe R operating (through the GUI). All your results, commands, errors, warnings get shown here.

5. **Open a Script**: From the Toolbar's `File`, click to `New File`, then `R Script` (Figure 1(c)). This will create a blank file with the `.R` file extension. Please enter your code for this assignment in this file, and submit it (see the end of this assignment for more details). We call this type of file a "script". It is a plain (i.e. no formatting added on) text file with executable code. Please save your script, in this case

with the file name as your last name followed by your first name, separated by an underscore, all in lower case (e.g. `kuriwaki_shiro.R`).

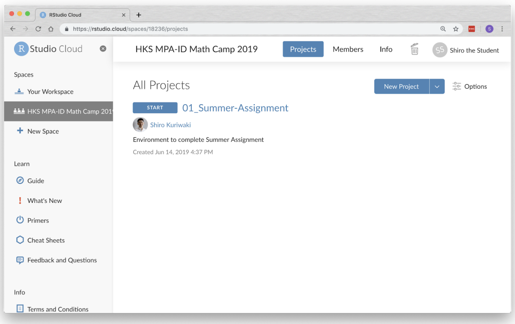
6. **Read in a Dataset**: Here, we'll first rely on the convenience features that the GUI provides. At the bottom right pane, you should see a "Files" tab (Figure 1(d)). Click through to the folders `data`, then `input`, and click on the filename `WEO-2018.xlsx,"` Choose `Import Dataset...` (Figure 1(e)). This starts the process of structuring a piece of `R` code to read in the file. One thing you want to **change** is the name you assign to your imported dataset. Because you will be typing in the object name many times, pick a short and informative name (like `weo`), as recommended in the style guide (Figure 1(f)).

   You'll see a preview of the spreadsheet and the command that produces it (Figure 1(g)). The bottom-right button, "Import", will send the code directly into the Console. To make your script replicable, copy the first two lines of this inserted code (the `library(readxl)` command and the line that involves `read_excel()`) to the beginning of your script.

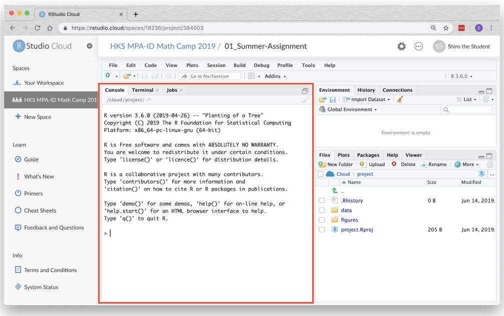
   All objects created in R will appear in the "Environment" pane (top-right), along with information like variable names. After you have imported the dataset, it can also be available for browsing on a tab right next to your R script.

7. **Load packages**: As you saw in the in primers (Programming Basics), a R session needs to load a package every time before using it (the exception is the base package, which is pre-loaded). Start your script by loading the `tidyverse` package, i.e. with typing `library(tidyverse)` at the top of your R script. To see if your code works and see its output, highlight your code in the R script and click on the "Run" Icon (or the hot-key `command` + `Enter`). "Running" (or executing) code sends the command to R and the output should be displayed in the Console.

(a) Open the Assignment Project

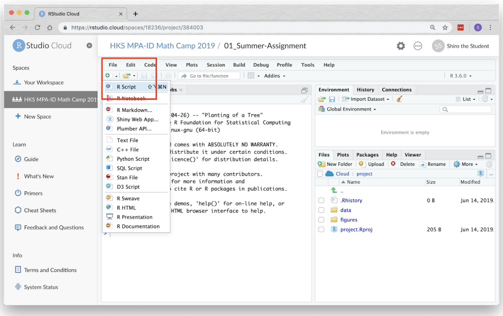


(b) RStudio GUI/IDE and the Console (highlighted)

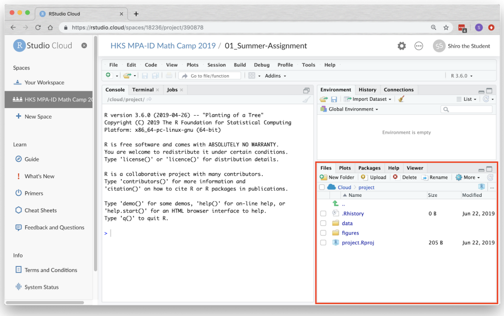


```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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.

>
```

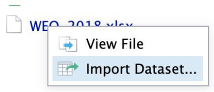
(c) Open a New R Script



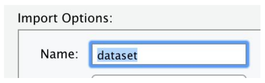
(d) The Files pane (highlighted) to find the spreadsheet



(e) Navigate to the data file and Import



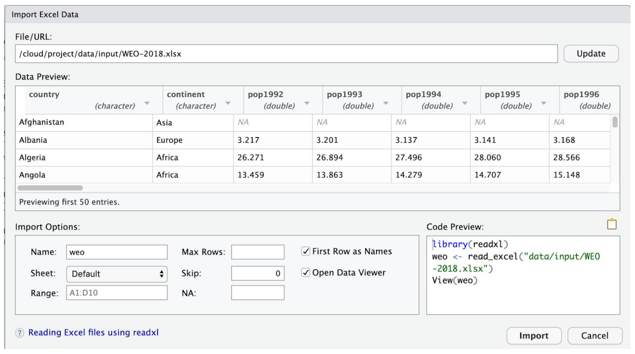
(f) Change the assigned name to an informative one



Import Options:

Name: dataset

(g) Preview the dataset before completing the Import



Import Excel Data

File/URL:

/cloud/project/data/input/WEO-2018.xlsx

Update

Data Preview:

country (character)	continent (character)	pop1992 (double)	pop1993 (double)	pop1994 (double)	pop1995 (double)	pop1996 (double)
Afghanistan	Asia	NA	NA	NA	NA	NA
Albania	Europe	3.217	3.201	3.137	3.141	3.168
Algeria	Africa	26.271	26.894	27.496	28.060	28.566
Angola	Africa	13.459	13.863	14.279	14.707	15.148

Previewing first 50 entries.

Import Options:

Name: weo
Max Rows:
✓ First Row as Names
Sheet: Default
Skip: 0
✓ Open Data Viewer
Range: A1:D10
NA:

Code Preview:

```
library(readxl)
weo <- read_excel("data/input/WEO
-2018.xlsx")
View(weo)
```

Reading Excel files using readxl

Import Cancel

Figure 1: Example Screenshots Corresponding to Problem 2

Problem 3: Sorting by Values

The following questions are based on a recent version of the World Economic Outlook dataset published by the International Monetary Fund (IMF), which you just read in. Each row in the spreadsheet is a country, with total GDP for a given year adjusted for purchasing power parity (with the `rgdp` column prefix) and total population (with the `pop` column prefix). GDP values are in millions of 2011 international dollars, so you can directly compare values in different years. Population values are in millions of persons.

(1) Write a command (connected by pipes) that (i) first sorts the dataset from lowest to highest real GDP in 2017, and then (ii) outputs a two-column dataset of the country and its GDP.

```
weo %>%
  arrange(rgdp2017) %>%
  select(country, rgdp2017)
```

```
## # A tibble: 191 x 2
##    country               rgdp2017
##    <chr>                    <dbl>
##  1 Tuvalu                    38.1
##  2 Nauru                    142.
##  3 Marshall Islands         172.
##  4 Kiribati                 207.
##  5 Palau                    261.
##  6 Micronesia               315.
##  7 Tonga                    536.
##  8 São Tomé and Príncipe    617.
##  9 Vanuatu                  701.
## 10 Dominica                 718.
## # ... with 181 more rows
```

(2) Write a command that is the same as (1) but now sorts it in descending order of 2017 GDP (highest to lowest).

```
weo %>%
  arrange(desc(rgdp2017)) %>%
  select(country, rgdp2017)
```

```
## # A tibble: 191 x 2
##    country         rgdp2017
##    <chr>              <dbl>
##  1 China          21094859.
##  2 United States  17662233.
##  3 India           8615891.
##  4 Japan           4944922.
##  5 Germany         3799057.
##  6 Russia          3650599.
##  7 Indonesia       2953727.
##  8 Brazil          2951498.
##  9 United Kingdom  2654284.
## 10 France          2582983.
## # ... with 181 more rows
```

(3) The `arrange()` command can sort on more than one variable. To rank countries within their continent, write a command that sorts the countries by continent (in alphabetical order), then by 2017 GDP in descending order. Remember different inputs ("arguments") to a function are separated by commas.

```
weo %>%
  arrange(continent, desc(rgdp2017)) %>%
  select(continent, country, rgdp2017)
```

```
## # A tibble: 191 x 3
##    continent country      rgdp2017
##    <chr>     <chr>           <dbl>
##  1 Africa    Egypt        1094122.
##  2 Africa    Nigeria      1019039.
##  3 Africa    South Africa  697330.
##  4 Africa    Algeria       576494.
##  5 Africa    Morocco       271959.
##  6 Africa    Ethiopia      182365.
##  7 Africa    Angola        173327.
##  8 Africa    Sudan         170360.
##  9 Africa    Kenya         148596.
## 10 Africa    Tanzania      147711.
## # ... with 181 more rows
```

(4) Write a command that shows African countries in descending order of their 2017 GDP (Use the variable `continent` to filter on African countries).

```
weo %>%
  filter(continent == "Africa") %>%
  arrange(desc(rgdp2017)) %>%
  select(country, rgdp2017)
```

```
## # A tibble: 55 x 2
##    country      rgdp2017
##    <chr>           <dbl>
##   1 Egypt        1094122.
##   2 Nigeria      1019039.
##   3 South Africa  697330.
##   4 Algeria       576494.
##   5 Morocco       271959.
##   6 Ethiopia      182365.
##   7 Angola        173327.
##   8 Sudan         170360.
##   9 Kenya         148596.
##  10 Tanzania      147711.
## # ... with 45 more rows
```

Problem 4: GDP per capita

Create a new tibble object called `weo_percap` that is the same as the main dataset but also includes:

- A variable called `gdp_percap_2017` that is the country's GDP per capita in 2017,
- A variable called `gdp_percap_1992`, which is the same as above but for 1992, and
- A variable called `growth_2017_1992` which indicates the rate of growth between the two variables above, following the definition below.

Operationalize the rate of growth between two periods a and $b > a$ as:

$$\text{growth}_{b,a} = \frac{\text{GDP per capita}_b - \text{GDP per capita}_a}{\text{GDP per capita}_a}$$

```
weo_percap <- weo %>%
  mutate(gdp_percap_2017 = rgdp2017 / pop2017,
         gdp_percap_1992 = rgdp1992 / pop1992,
         growth_2017_1992 = gdp_percap_2017 - gdp_percap_1992)
```

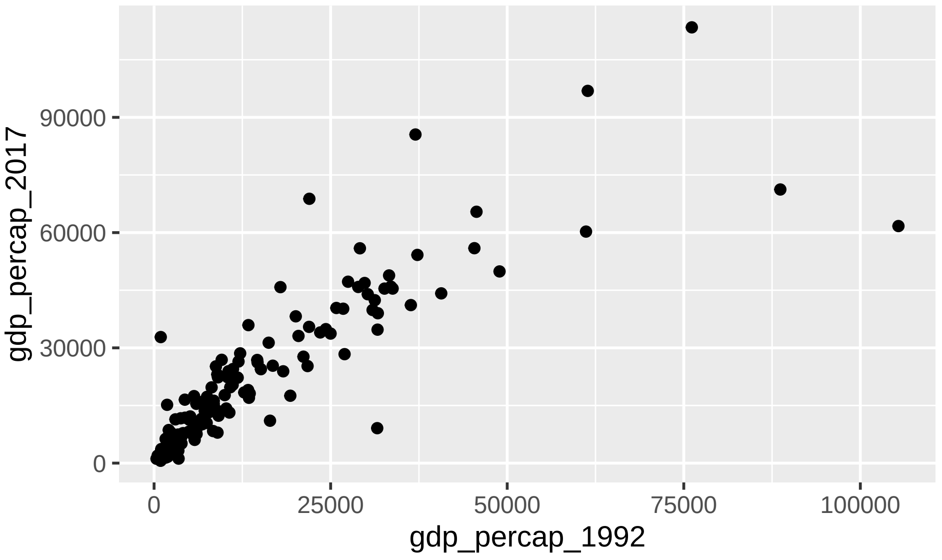
You will be using this new object for the subsequent problems.

Problem 5: Graphing

(1) Make a scatterplot that shows a countries 1992 GDP per capita on the x-axis and its 2017 GDP per capita on the y-axis.[2]

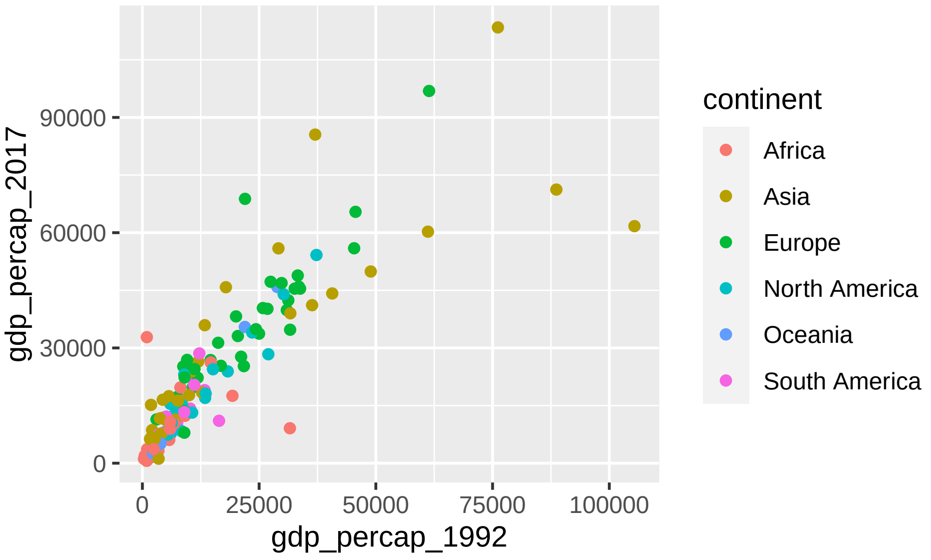
```
ggplot(weo_percap, aes(gdp_percap_1992, gdp_percap_2017)) +
  geom_point()
```

[2] You might notice that the scatterplot itself is not as informative as it could be. In later sessions, we will spend time discussing the nuts and bolts of making a high-quality graphic that is informative and user-friendly.



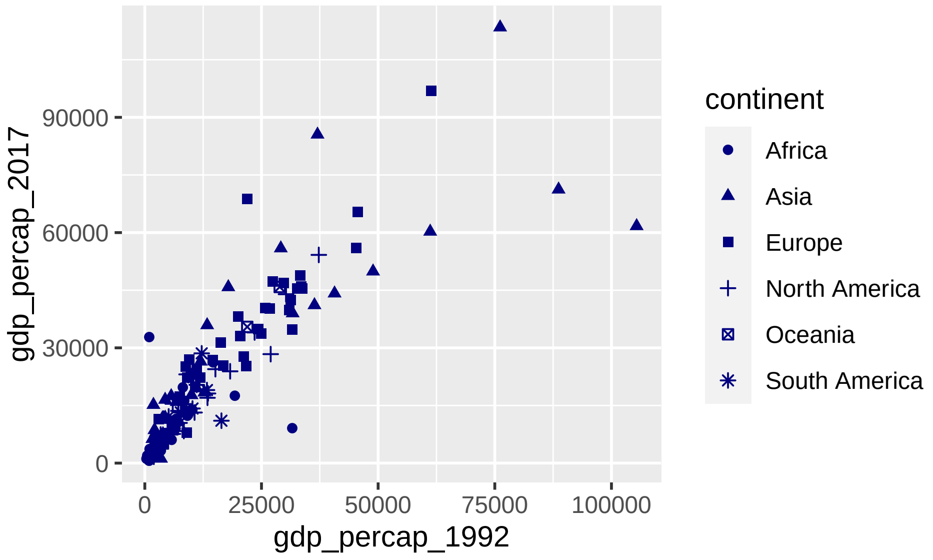
(2) Show the same figure, but coloring the points by continent. That is, countries of the same continent should have the same color.

```
ggplot(weo_percap,
       aes(x = gdp_percap_1992, y = gdp_percap_2017, color = continent)) +
  geom_point()
```



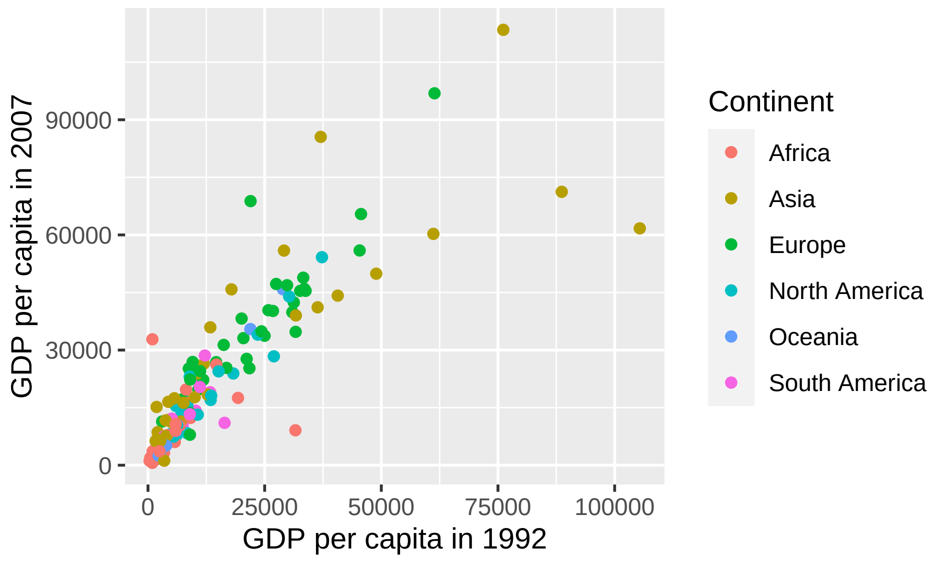
(3) Show the same figure, but assigning a different shape of point for different continents. Also, set the color for all countries to navy by using the color name `"navy"`.

```
ggplot(weo_percap,
       aes(x = gdp_percap_1992, y = gdp_percap_2017, shape = continent)) +
  geom_point(color = "navy")
```



(4) To make a figure presentable in a professional setting, you will need to many, many more additions to a graph. At the very least, you need to label your axes with actual words. Update the graph in part (2) by adding an informative x-axis title and a y-axis title. This can be specified by adding a `labs()` layer to your graph. For an example, open the help page for the `labs` function, and scroll to the bottom of the page under "Examples".

```
ggplot(weo_percap, aes(x = gdp_percap_1992,
                       y = gdp_percap_2017,
                       color = continent)) +
  geom_point() +
  labs(x = "GDP per capita in 1992",
       y = "GDP per capita in 2007",
       color = "Continent")
```



Problem 6: Mean and Median

(1) Let's start to think about summarizing variables using summary statistics. Write code that reports the mean of country-level GDP per capita of 2017 in one column and the median for 2017 in another. Pick column names that are sufficiently self-explanatory but also concise.

```
weo_percap %>%
  summarize(mean_gcap_2017 = mean(gdp_percap_2017),
            median_gcap_2017 = median(gdp_percap_2017))
```

```
## # A tibble: 1 x 2
##   mean_gcap_2017 median_gcap_2017
##            <dbl>            <dbl>
## 1         18915.           11590.
```

(2) Write code that indicates which countries have missing values for 1992 GDP per capita and 2017 GDP per capita.

```
weo_percap %>%
  filter(is.na(gdp_percap_1992)) %>%
  select(country)
```

```
## # A tibble: 25 x 1
##    country
##    <chr>
##  1 Afghanistan
##  2 Bosnia and Herzegovina
##  3 Czech Republic
##  4 Estonia
##  5 Georgia
##  6 Iraq
##  7 Kosovo
##  8 Liberia
##  9 Lithuania
## 10 Macao
## # ... with 15 more rows
```

```
weo_percap %>%
  filter(is.na(gdp_percap_2017)) %>%
  select(country)
```

```
## # A tibble: 0 x 1
## # ... with 1 variable: country <chr>
```

(3) Try the same as (1) but now with replacing 2017 with 1992. You should initially notice that you get a missing value for both summary statistics. This is because by default, `mean()` and `median()` report a missing value if at least one of its input values is missing, and as you probably found in part (2), some countries have have missing values for 1992 GDP per capita. Now, modify your code so that you change this default and ignore the missing values in your computation. As the help page for the functions indicate, the relevant argument is `na.rm` (for NA - remove). Change its logical value from `FALSE` (its default) to `TRUE`, while making sure to follow the style guide for proper spacing (style guide section 2.2.3).

```
weo_percap %>%
  summarize(mean_gcap_1992 = mean(gdp_percap_1992, na.rm = TRUE),
            median_gcap_1992 = median(gdp_percap_1992, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   mean_gcap_1992 median_gcap_1992
##            <dbl>            <dbl>
## 1         12894.            6819.
```

Problem 7: slice() and filter()

Much of data analysis is understanding how new functions work through reading the documentation and experimentation. The function `slice()` is part of the tidyverse and allows you to filter rows by their position. Notice that `slice()` and `filter()` are similar in that they subset rows of a dataset, but differ in the types of input they require — the former asks for positions, the latter asks for conditions.

Check out the help page of `slice()` (recall, e.g., by typing `?slice` in the Console). Then, write a command that shows the countries with the top three and bottom three 2017 GDPs, thereby combining the output in the first two R exercises.

```
weo %>%
 arrange(desc(rgdp2017)) %>%
 select(country, rgdp2017) %>%
 slice(c(1:3, 189:191))
```

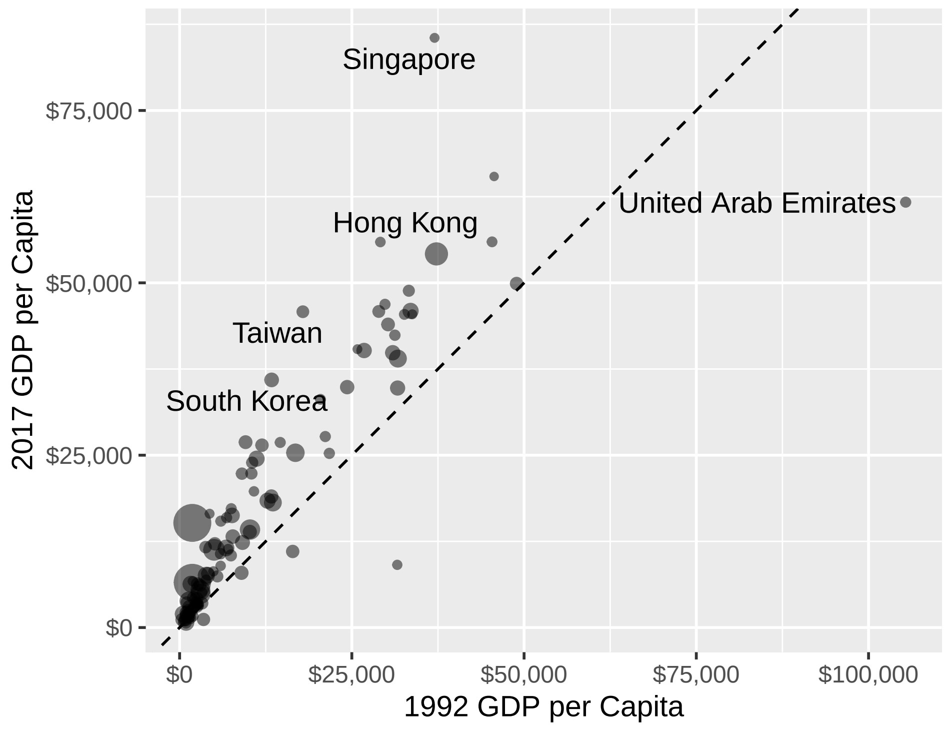
```
## # A tibble: 6 x 2
##   country            rgdp2017
##   <chr>                 <dbl>
## 1 China           21094859.
## 2 United States   17662233.
## 3 India            8615891.
## 4 Marshall Islands      172.
## 5 Nauru                 142.
## 6 Tuvalu                 38.1
```

[Optional and Challenging] More Graphing

Note: This problem is optional; it involves some commands not covered in the primers.

Make a graph like the one shown in Figure 2. Follow both the graphical components of the graph shown as you see them, as well as the description of the measures as described in the Figure caption. *Hint:* Check out the packages `ggrepel` and `scales` to implement some of the features.

```
# only use large enough countiries
ds_large <- weo_percap %>%
  filter(pop1992 > 5 | pop2017 > 5)

# subset to show country names. Only use the top 5.
sub_show <- ds_large %>%
  mutate(abs_change = abs(gdp_percap_2017 - gdp_percap_1992)) %>%
  arrange(desc(abs_change)) %>%
  select(country, gdp_percap_2017, gdp_percap_1992) %>%
  slice(1:5)

# Creat the graph, but add a layer of geom_text_repel and input the subset there
gg_scatter <- ggplot(ds_large, aes(gdp_percap_1992, gdp_percap_2017)) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  geom_point(aes(size = pop2017), alpha = 0.5) +
  coord_equal() +
  geom_text_repel(data = sub_show, aes(label = country)) +
  scale_x_continuous(labels = dollar) +
  scale_y_continuous(labels = dollar) +
  guides(size = FALSE) +
  labs(x = "1992 GDP per Capita",
       y = "2017 GDP per Capita",
       caption = "Points sized by 2017 population and labels show top 5 countries
       with the most absolute change. Only countries with population at least 5 million\n

ggsave("gg_scatter.pdf", gg_scatter, width = 5, height = 5)
```



Points sized by 2017 population and labels show top 5 countries
with the most absolute change. Only countries with population at least 5 million
in 1992 or 2017 shown. Dotted line indicates 45−degree line.

Figure 2: Changes in GDP per capita between 1992 and 2017.

Submitting (and Survey)

Before you submit, please complete this brief survey as so that we understand better you background and can design the R activities in math camp accordingly.

https://harvard.az1.qualtrics.com/jfe/form/SV_0jLN9xCSLAWn2Ul

Once you have completed or made an attempt for all the problem, please clean up your R script, download it from the cloud, and submit it to Canvas.

Math camp instructors will check and provide comments for your code. You should follow these guidelines to clean up your final submission (and should do so for all future scripts):

- Delete any failed attempts or duplicative code.
- Label the relevant question number by comment (e.g., `## Problem 1.1 -------`. Follow the style guide for the exact format).
- You do not need to submit anything other than the R script (i.e., figures or numbers not necessary), but the results should be "reproducible" from the script. This means that an instructor who receives your script should be able to run it and reproduce correct answers. To preview this, try restarting R (Toolbar `Session > Restart`) and running your entire code at once (e.g., Select All Text and Run, or `Run All` by the hot-key `option + command + R`. Before you do this, though, make sure you explicitly load the `tidyverse` package in your code by adding `library(tidyverse)` to the beginning of your file.
- Follow other guidelines from the style guide, such as breaking up long lines and properly using spaces.
- As we mentioned in the beginning, please name your script with your last name followed by your first name, all in lower case. This helps graders sort through all submissions.

After editing your code, save it to the main project folder, and then download it by right-clicking the file icon (in the Files pane), and selecting `Export` (Figure 3). Download the script and attach it to your Canvas submission.

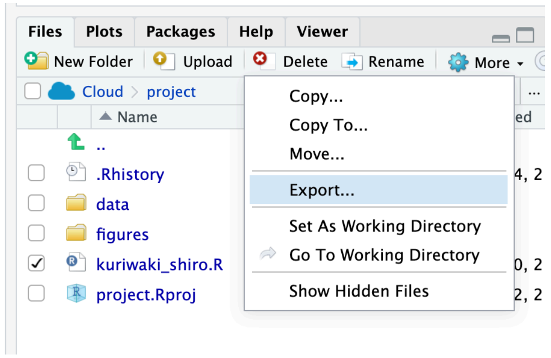


Figure 3: Downloading your final script