Data Science And AI for Economists

Lecture 2: Introduction to R and RStudio

Zhaopeng Qu
Business School, Nanjing University
September 10 2025

Roadmap

Today we will cover the following topics:

- Coding skills in the age of AI
- Programming Language:
  - R Basics and Data Management
- Version Control, Git and GitHub
  - How to Solve the conflicts?
  - How to use branches?

Coding skills in the age of AI

Coding skills in the age of AI

Do we still need to master R syntax in the age of powerful AI tools?

Yes, we still need R fundamentals — but with a strategic approach

AI Tools Have Limitations

- AI generates code, but you need to understand and verify its output
- Without basic syntax knowledge, you cannot judge if AI-generated code is correct or suitable
- Debugging and modifying AI code still requires solid programming foundations

Remember: **AI is not always correct and efficient**. You have to check it before using it.

The Key Question: **What to Learn** and **How to Learn**?

- Focus on **core concepts** rather than memorizing every detail
- Learn to **collaborate with AI** more effectively and efficiently

Strategic Learning Approach for the AI Era

Essential Skills to Master for Data Science and Economists:

- Core Concepts: data structures, formats, and variables (vectors, data.frames, lists, characters, etc.)
- Basic Rules: set up working directory, use Git and GitHub for code management
- Fundamental Operations: data import, export, cleaning, transformation, visualization
- Logical Thinking: decompose problems and design analysis workflows
- Package Ecosystem: basic usage of tidyverse, ggplot2, and other key packages

"I cannot trust AI to help me clean and analyze data with just a simple prompt."

"You always have to monitor and guide AI to do the right things."

Strategic Learning Approach for the AI Era (cont.)

Areas Where AI Can Handle the Heavy Lifting:

- Complex syntax implementation: nested loops, intricate conditional statements
- Advanced programming techniques: code optimization, parallelization, memory management
- Custom function development: specialized functions with multiple parameters
- Package-specific syntax: obscure function arguments and advanced features
- Code debugging and troubleshooting: identifying and fixing programming errors

Recommended Learning Strategy:

1. Master Fundamentals → Collaborate with AI → Boost Efficiency
2. Learn to **read and understand code**, not just write it
3. Focus on the **core data analysis workflow**, let AI handle implementation details
4. Keep up with the latest developments in this **rapidly evolving** field

In summary, focusing on **the right things** and **the right way to do things** is crucial.

We will learn how to use AI to assist with coding and data analysis later in the course.

R, RStudio and Settings

Intro RStudio

- The most popular IDE for R
  - 如果把R认为是发动机的话，IDE可以认为是操作面板，两者共同构成数据分析的“战车”。
  - 所以发动机可以都是R,但可以使用不同品牌或公司的操作面板(IDE)。
  - 所有操作面板中，RStudio是最容易上手的，使用者最多的。
- Also Free(for basic version)
- Combine RStudio with R Markdown ,Quarto and LaTeX to easily create scientific documents and presentations using tools like xaringan.
- Download it here: RStudio

Introduction to RStudio

Installing RStudio

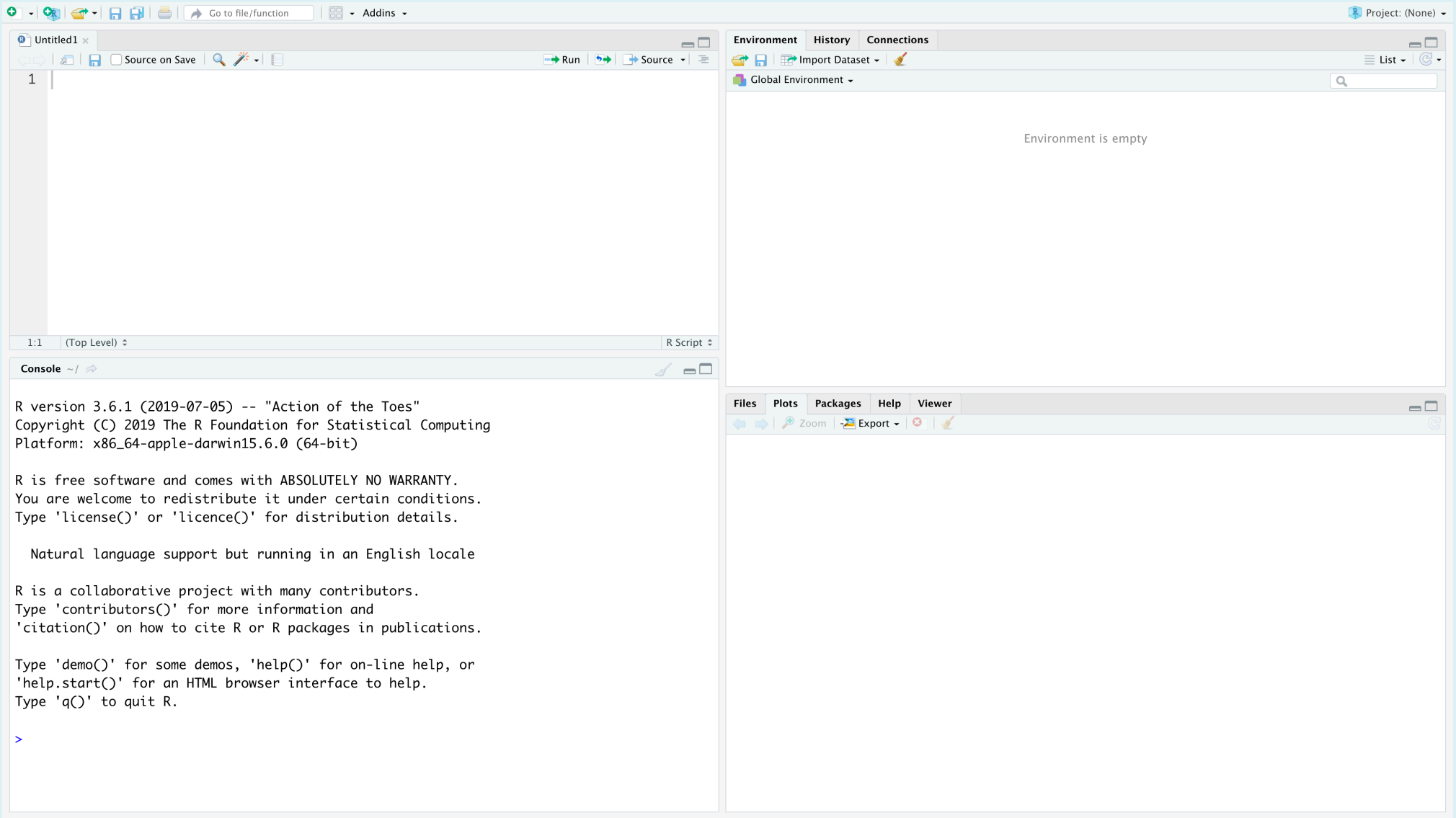
- 注意事项(for Windows)
  - 官网下载
  - 安装路径里不要有中文
  - 目录最好不是program files,不然后面可能会由于“管理员权限”影响使用。

Introduction to RStudio

RStudio's basic interface

- Console(Command Window)
- R Script or Rmarkdown(Code Editor)
- Environment(Environment Window)
  - Import Data
  - History
  - Connection: data sources
  - Git
- Others (Files, Plots, Packages, Help, Viewer)
  - Files
  - Plots
  - Packages
  - Help
  - Viewer

RStudio's basic interface



Introduction to RStudio

Three ways to interact with R

1. Write commands in the command window(Console): simple commands
2. Write commands in the R script(R Script): complex commands or functions
3. Write commands in the Rmarkdown or Quarto document(Rmarkdown or Quarto): generate analysis code+results+text report.

- In most cases, we should **first** code in the R script(R Script).

Introduction to RStudio: Packages

Packages(包)

- 包是基于R的基本功能，用来完成某些高级功能的专属模块。可以通过全世界的镜像地址，在线下载和安装。

```
.packages(TRUE) # show all packages
```

- Packages(包) 基本分成三类：
    - 基础包
    - 附加包
    - 个人包
- 因为开源，提交包的门槛相对较低，所以导致包的质量良莠不齐。因此在使用一个包之前，最好对这个包的使用情况有一定了解。

Packages(包)

- Basic commands
  - Install: use `install.packages("package name")`
  - Load, after installation, you need to load it: use `library(package name)`
  - Unload, sometimes you need to unload it: `detach("package:package name", unload=TRUE)`
  - Remove, completely remove: `remove.packages("package name")`
- In RStudio, you can also use the window menu to select Install, Load, Unload, Remove. in the Packages tab.
- eg. install `AER` package, you can use the following code:

```
install.packages("AER",repos = "https://mirrors.nju.edu.cn/CRAN/") # use the NJU mirror
```

Packages(包)

- 选择合适的镜像
- Tools → Global Options → Packages → Primary CRAN repository
- 在Primary CRAN repository中选择国内的相关镜像。

Packages(包)

- 查看某个Packages的内容

```
help(package = "AER")
```

- 加载Packages

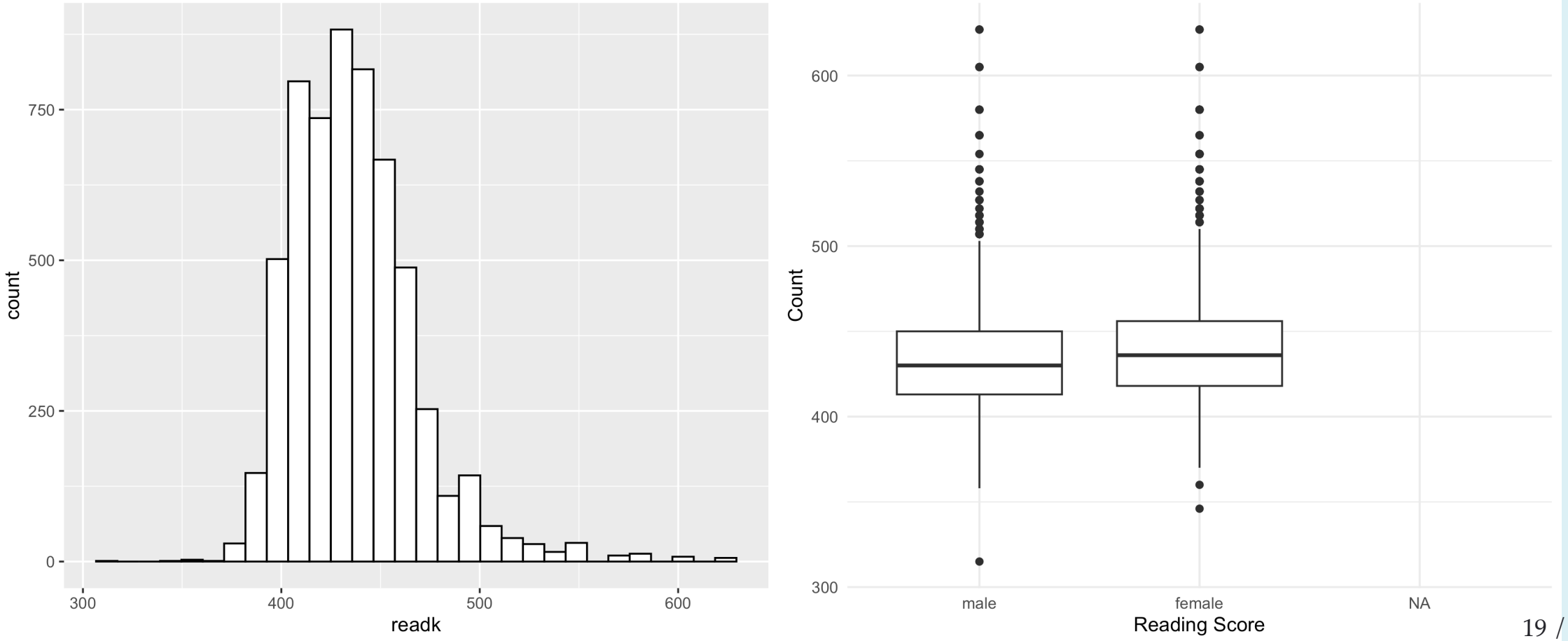
```
library(AER)
data(STAR)
```

RStudio: Plot

```
library(ggplot2)
p1<-ggplot(STAR, aes(readk)) +
  geom_histogram(bins=30,colour="black",fill="white")
p2<-ggplot(STAR, aes(x=gender,y=readk)) +
  geom_boxplot() +
  theme_minimal() +
  theme(legend.position = "none") +
  xlab("Reading Score") +
  ylab("Count")
```

RStudio: Plot

```
library(gridExtra)
grid.arrange(p1,p2,ncol = 2, nrow = 1)
```



Where are packages installed and `.libPath`

- Normally,packages are installed in the directory where R was installed.
- If you want to know the exactly place, just type

```
.libPaths()
```

```
## [1] "/Users/freeman/Library/CloudStorage/Dropbox/R/R_Packages/arm"
## [2] "/Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library"
```

- You can also add a directory that you created as the "store" of packages.Like as

```
.libPaths(c(.libPaths(),"~/Library/CloudStorage/Dropbox/R/R_Packages/"))
.libPaths()
```

```
## [1] "/Users/freeman/Library/CloudStorage/Dropbox/R/R_Packages/arm"
## [2] "/Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library"
## [3] "/Users/freeman/Library/CloudStorage/Dropbox/R/R_Packages"
```

Adding `.libPath`

- But it is just a temporary setting. If you restart your R and type `.libPath()` , you will find that the new directory disappear.

```
.rs.restartR()
.libPaths()
```

Editing `.Renviron` files for MacOS

- If you want to set it permanently, you have to set it in `.Renviron` file, which is a hidden file in your home directory.Every time when R starts, it will be executed first.
- Open it by the following code:

```
file.edit("~/.Renviron")
```

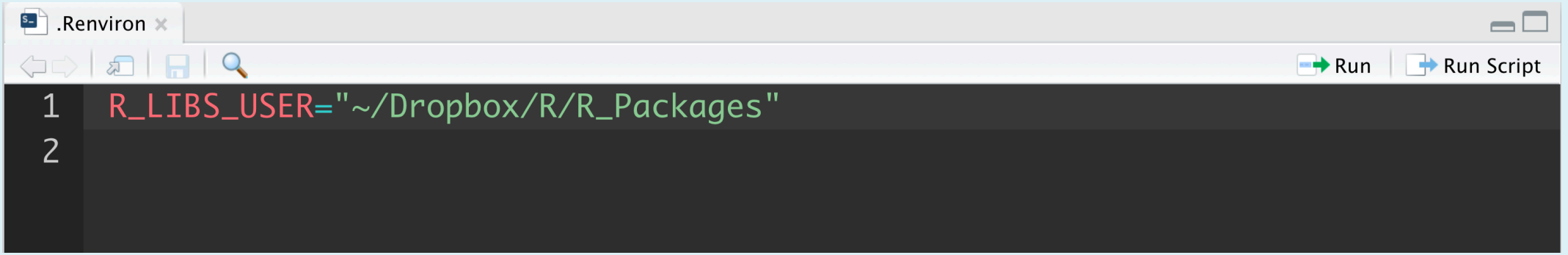
- or use the function `usethis::edit_r_environ()`

```
usethis::edit_r_environ()
```

Introduction to RStudio

`.Renviron` files for MacOS

- Just add a line `R_LIBS_USER=*your directory*` in the file.



- Restart R session, you will find the added directory is permanently changed(or added).

Editing `.Renviron` files for **Windows**

- For Windows users, the `.Renviron` file location and path format are different:

```
# Open .Renviron file on Windows
file.edit("~/.Renviron")
# or use the usethis package
usethis::edit_r_environ()
```

- Add a line with Windows path format (use forward slashes or double backslashes):

```
R_LIBS_USER=C:/Users/YourUsername/Documents/R/Packages
# or
R_LIBS_USER=C:\\Users\\YourUsername\\Documents\\R\\Packages
```

Editing `.Renviron` files for **Windows**

- Important Notes for Windows:
  - Avoid paths with spaces or special characters
  - Create a simple path like `D:/R_Packages`
  - Use forward slashes `/` or double backslashes `\\`
  - Restart R session to make changes permanent

Understanding `.Renviron` vs `.Rprofile`

`.Rprofile` - R Code (executed **second**)

- Contains R code that runs at startup
- Can use variables set in `.Renviron`
- Used for R-specific customizations
- Can load packages, set options, define functions

```
# locate the .Rprofile file
file.path(R.home('etc'),'Rprofile')
edit(file.path(R.home('etc'),'Rprofile'))
```

.Renviron and .Rprofile

.Renviron

- `.Renviron`文件则是一个环境变量文件，在R启动时，如果这个文件存在，它会被首先执行。
- 使用这个文件可以设置一些环境变量，比如包的安装路径等。

```
file.edit('~/.Renviron') # open the file and edit
```

- Restart R(Cmd+Shift+F10), then try to install a package
- 自定义包的安装路径的好处：当重装系统、更换电脑等情况下，不需要重新reinstall这些包。

.Rprofile

- 而`.Rprofile`文件则是一个R代码文件，在R启动时，如果这个文件存在，它会被首先执行。

```
file.edit('~/.Rprofile') # open the file and edit
```

`.Renviron` vs `.Rprofile` Examples

`.Renviron` Example:

```
R_LIBS_USER=D:/R_Packages # set the package inst
R_MAX_NUM_DLLS=150 # set the maximum number of D
LANGUAGE=en # set the language
system.file() # show the system file path
```

Purpose:

- Package installation paths
- Memory settings
- Language settings
- System paths

`.Rprofile` Example:

```
# R code executed at startup
options(repos = "https://mirrors.nju.edu.cn/CRAN
library(tidyverse) # load the tidyverse package
#Custom function
hello <- function() {
  cat("Welcome to R!\n")
}
```

Purpose:

- Set R options
- Load frequently-used packages
- Define custom functions
- Customize R behavior

English Language Setting

Windows 下的`Rconsole`文件

```
file.path(R.home('etc'),'Rconsole') # find the path
```

```
language = en # set the language
```

MacOS 下的系统环境

```
Sys.setenv(LANGUAGE = 'en')
Sys.unsetenv() # unset the language
```

养成一个好习惯

- 在可能的情况下，**尽量使用英文而不是中文**，避免一些不必要的麻烦（比如编码问题，路径问题，文件名问题等等）。

Data Management in R

Introduction

Common Types Of Variable in Statistics

- Quantitative(定量)
  - Income,Testscore,Weight,Height
- Qualitative(定性)
  - Gender, Nationality, Identity, Profession...
- Date(时间)
  - day,month,year,lunar calendar

Types in Qualitative(定性)

- Qualitative（定性）
  - categoried(nominal) variable 无序分类变量 分类变量
    - 性别、民族、行业...
  - ordered(ordinal) variable 有序分类变量
    - 教育水平、自评健康、幸福感
  - 逻辑变量
    - 是与否

Data Types in R

- Numeric Data
  - integer: only store whole numbers: `2L`
  - double: floating point numbers (i.e. with a decimal part)
- Character Data: store strings of characters.
  - character: 字符串
  - factor: 分类标签 (如性别、民族、行业等)
- Dates
  - Date: the number of days
  - POSIXct: the number of seconds
- Logical: store TRUE or FALSE 用于逻辑判断

Data forms(object types) in R

Scalars(标量)

Vectors(向量)

Matrix(矩阵)

List(列表)

Array(数组)

Data.frames(数据框)

Data forms(object types) in R

Vectors(向量)

- All elements in a vector are the same type.
  - c(1,2,3,4)
  - c("R","Stata","Python")
- Factor Vectors(因子向量)

```
q<-c("soccer","badminton","soccer",
     "tennis","soccer","tennis")
q
```

```
## [1] "soccer"    "badminton" "soccer"    "tennis"    "soccer"    "tennis"
```

- transform into a factor vector

```
q.factor <- as.factor(q)
q.factor
```

```
## [1] soccer    badminton soccer    tennis    soccer
## Levels: badminton soccer tennis
```

Data forms(object types) in R

Data.frames(数据框)

- `data.frame` is just like an Excel spreadsheet in that it has columns and rows.
- Each column is a *variable* and each row is an *observation*.
- A `data.frame` can be seen as a collection of vectors.

- generate a data.frame

```
x  <-  6:1
y  <- -4:1
df <- data.frame(x,y,q)
df
```

```
##   x  y         q
## 1 6 -4    soccer
## 2 5 -3 badminton
## 3 4 -2    soccer
## 4 3 -1    tennis
```

- naming variables

```
df <- data.frame(Frist=x,Second=y,Sport=q)
df
```

```
##   Frist Second     Sport
## 1     6     -4    soccer
## 2     5     -3 badminton
## 3     4     -2    soccer
## 4     3     -1    tennis
## 5     2      0    soccer
## 6     1      1    tennis
```

Data forms(object types) in R

List(列表)

- A list is a container which can contain all *numerics* or *characters* elements or a mix of the two or `date.frames` or other `lists`.

```
list(1,2,3)
```

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

```
list(c(1,2,3))
```

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

Data forms(object types) in R

List(列表)

- combination of a `data.frame` and a `vector`

```
    list(df,1:6)
```

```
## [[1]]
##   Frist Second     Sport
## 1     6     -4    soccer
## 2     5     -3 badminton
## 3     4     -2    soccer
## 4     3     -1    tennis
## 5     2      0    soccer
## 6     1      1    tennis
##
## [[2]]
## [1] 1 2 3 4 5 6
```

Data forms(object types) in R

Conversion functions

- `is.datatype()` return `TRUE` or `FALSE`
- `as.datatype()` converts the argument to that type

Test	Convert
`is.numeric()`	`as.numeric()`
`is.character()`	`as.character()`
`is.vector()`	`as.vector()`
`is.matrix()`	`as.matrix()`
`is.data.frame()`	`as.data.frame()`
`is.factor()`	`as.factor()`
`is.logical()`	`as.logical()`

Preparation for Data Analysis

Working directory

- Working in an appropriate directory.

```
getwd()
setwd("~/Dropbox/R/R_Class/DSAI/Labs/Lab2")
```

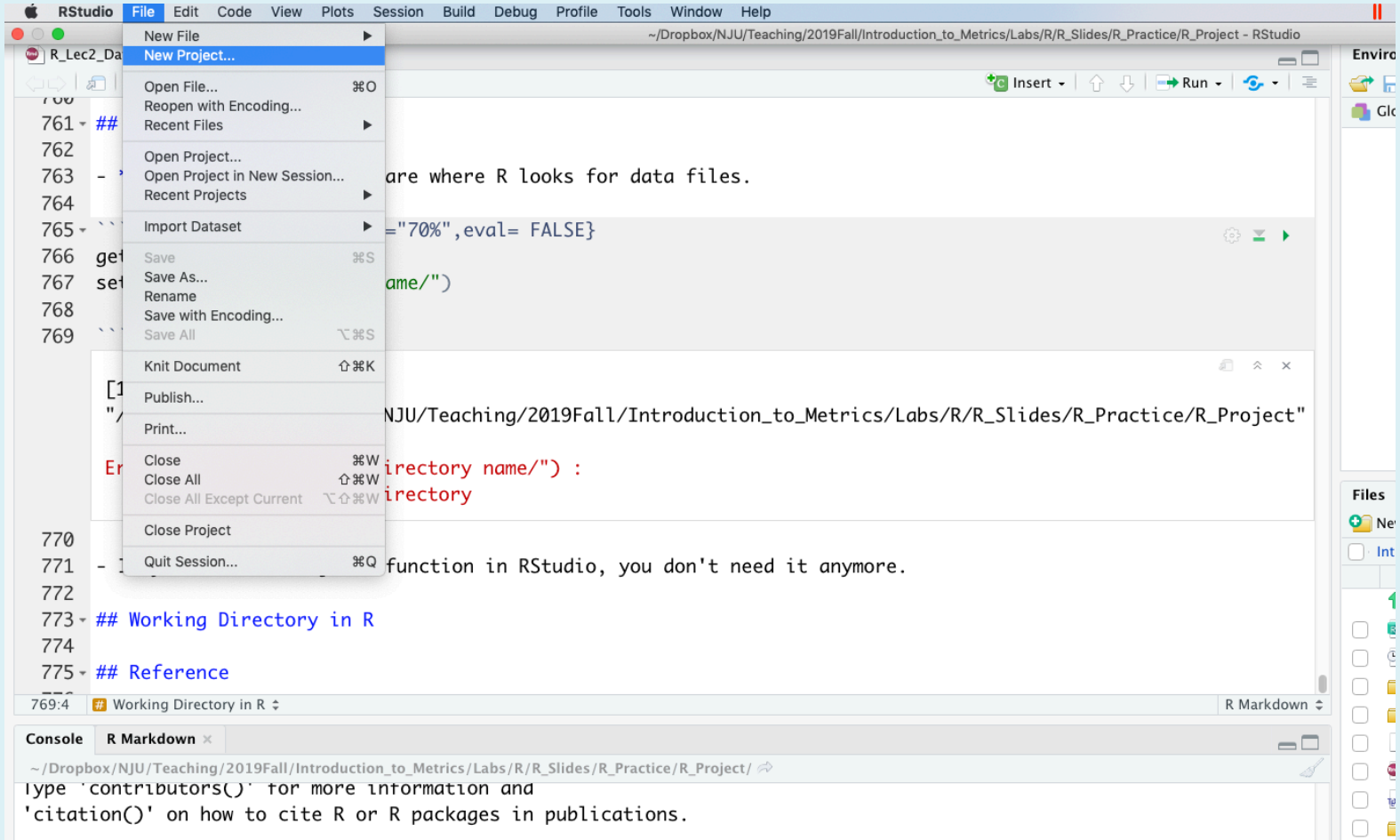
- Clean out your workspace
    - `Session -> Clear`

```
rm(list=ls())
```

- `Restart R` and open a new `RScript` file.
- Note: If you use the `Projects` function in RStudio, you don't need to set the working directory anymore.

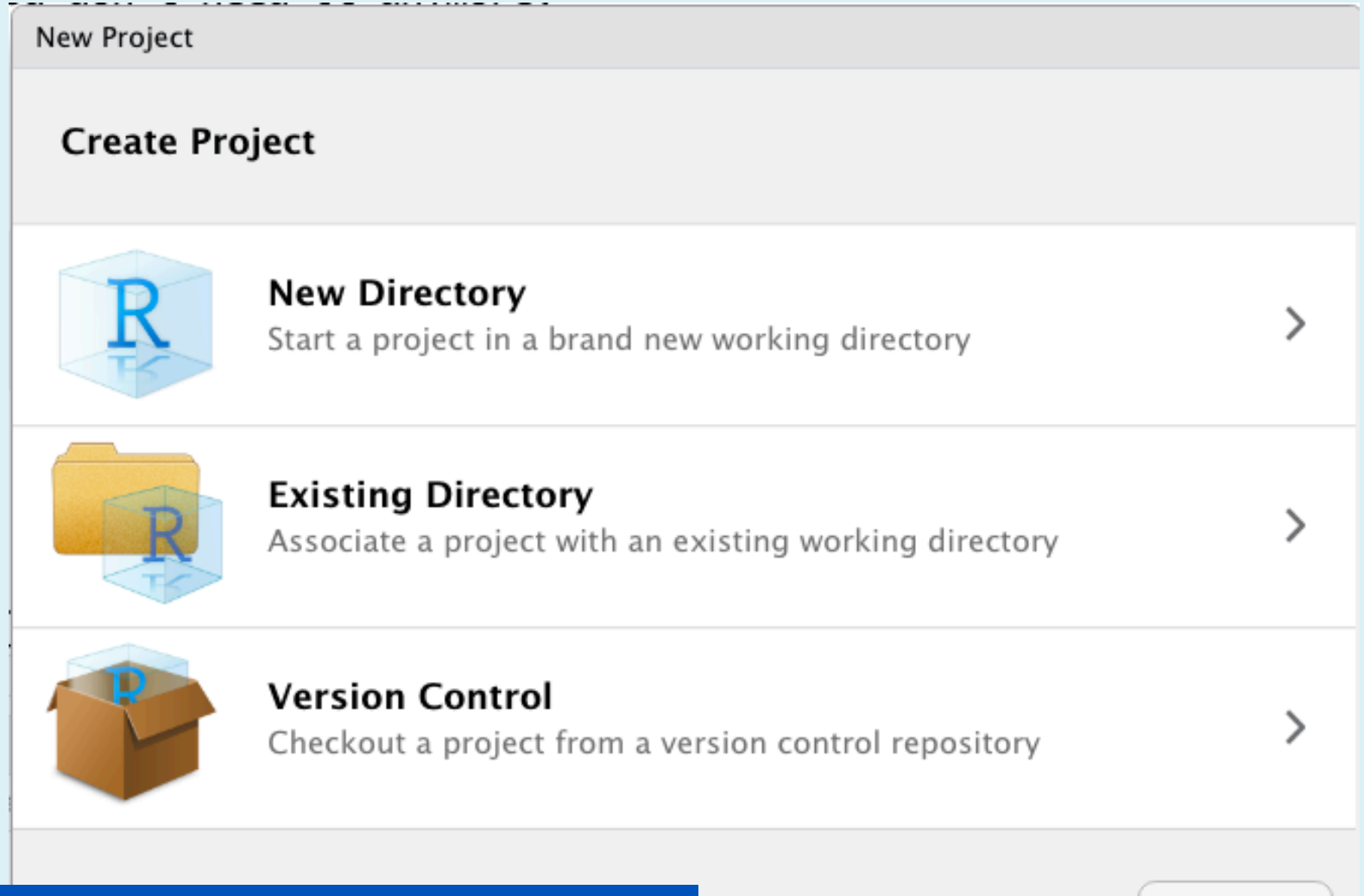
Projects function in RStudio

Projects function in RStudio



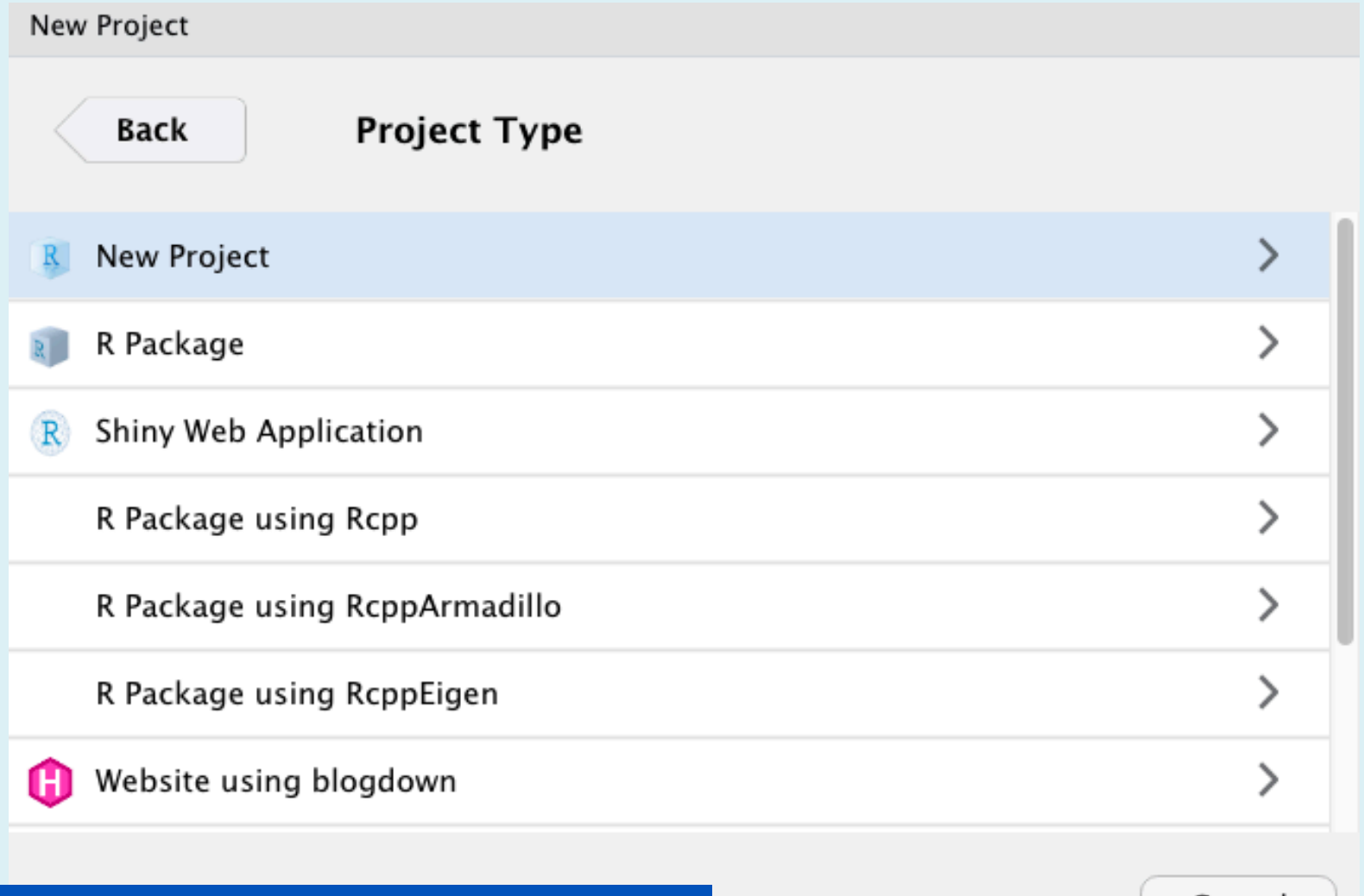
Projects function in RStudio

Projects function in RStudio

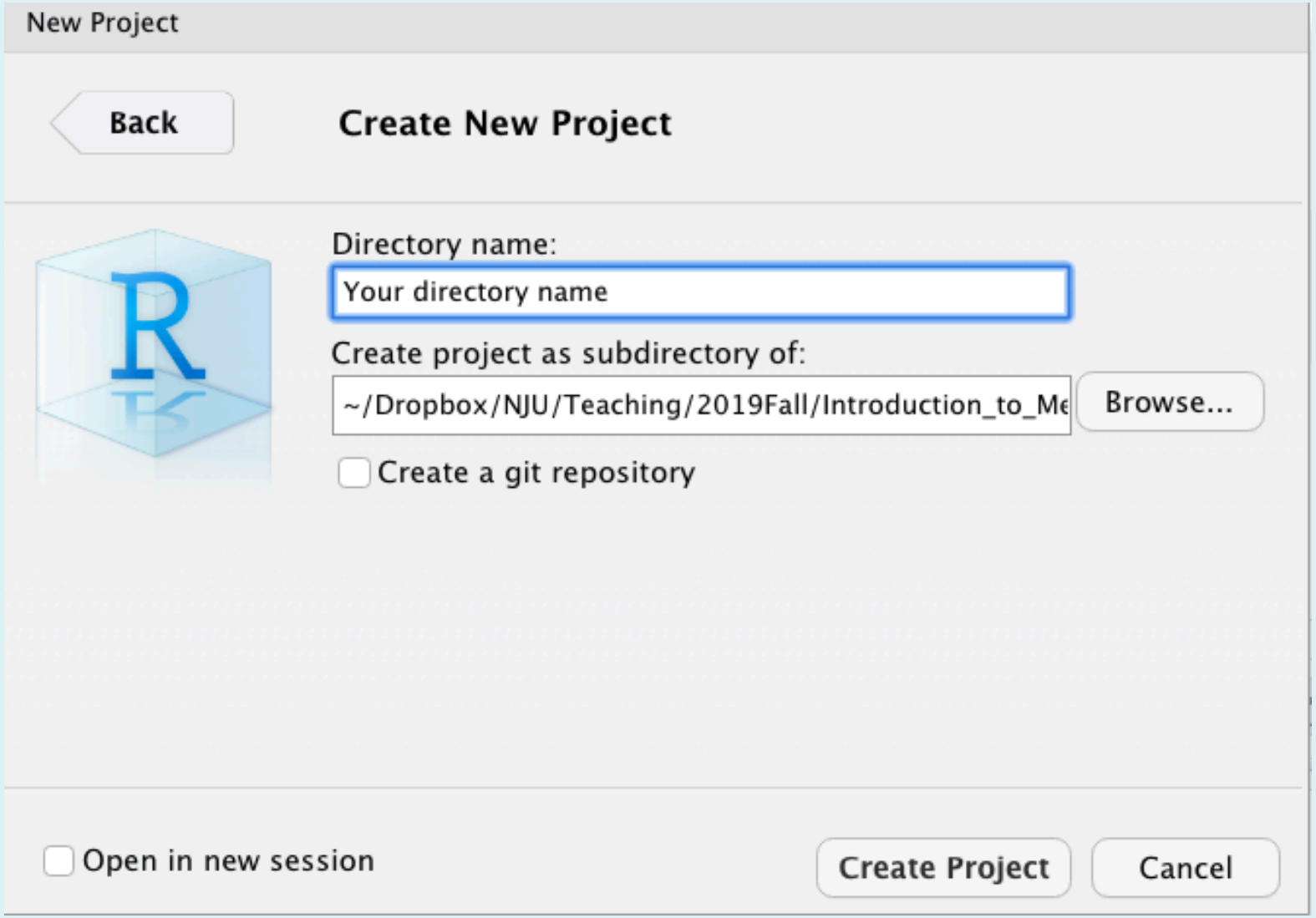


Projects function in RStudio

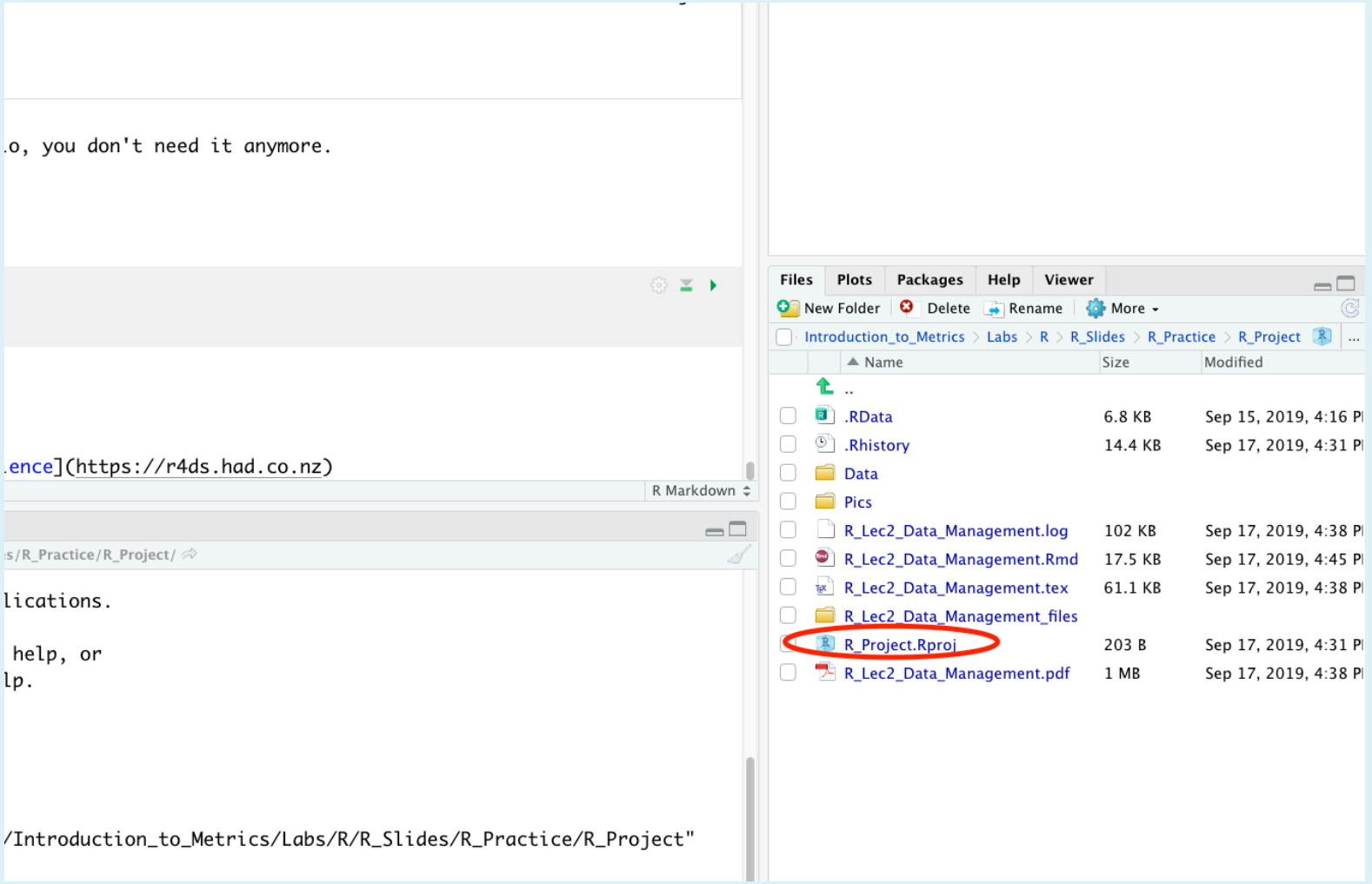
Projects function in RStudio



Projects function in RStudio



Projects function in RStudio



Project function in RStudio: Practice

you have done this in the previous lecture

1. Creat a new project on your github
2. Clone the project to your local computer
3. Copy all the files in the project to your local computer
4. Make a commit to the project on your local computer
5. Push the project to your github

you are good to go

1. solve the conflict in the project
2. creat a new branch to solve the conflict
3. merge the branch to the main branch

Data Management in Practice: [redacted] data

Install and Load `gapminder` data

```
install.packages("gapminder")
library(gapminder)
```

Look at the `gapminder` data frame

```
class(gapminder)
```

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

Meet the `gapminder` data frame or "tibble"

```
str(gapminder)
```

```
## tibble [1,704 × 6] (S3: tbl_df/tbl/data.frame)
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num [1:1704] 28.8 30.3 32 34 36.1 ...
##  $ pop      : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921
##  $ gdpPercap: num [1:1704] 779 821 853 836 740 ...
```

info of `gapminder` data

- obtain information about the columns:

variable	meaning
country	
continent	
year	
lifeExp	life expectancy at birth
pop	total population
gdpPercap	per-capita GDP

Basic R

```
head(gapminder)
```

```
## # A tibble: 6 × 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Basic R

```
tail(gapminder)
```

```
## # A tibble: 6 × 6
##   country  continent  year lifeExp      pop gdpPercap
##   <fct>    <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Zimbabwe Africa     1982    60.4  7636524      789.
## 2 Zimbabwe Africa     1987    62.4  9216418      706.
## 3 Zimbabwe Africa     1992    60.4 10704340      693.
## 4 Zimbabwe Africa     1997    46.8 11404948      792.
## 5 Zimbabwe Africa     2002    40.0 11926563      672.
## 6 Zimbabwe Africa     2007    43.5 12311143      470.
```

Basic R

```
names(gapminder)
```

```
## [1] "country"   "continent" "year"      "lifeExp"   "pop"       "gdpPercap"
```

```
ncol(gapminder)
```

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

```
length(gapminder)
```

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

```
dim(gapminder)
```

```
## [1] 1704    6
```

```
nrow(gapminder)
```

```
## [1] 1704
```

Basic R: Summary

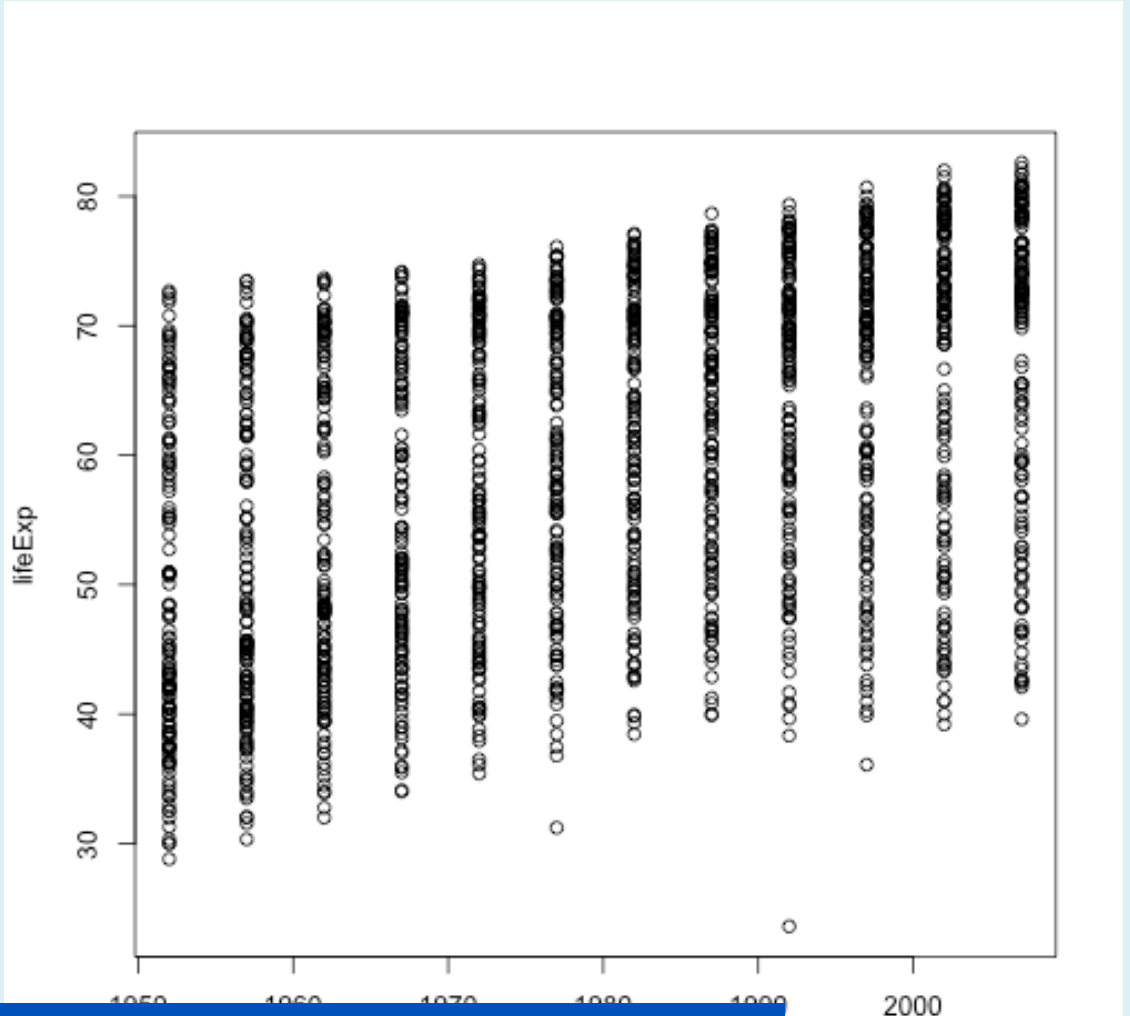
- A statistical overview can be obtained with `summary()`

```
summary(gapminder)
```

```
##         country        continent        year         lifeExp
##  Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60
##  Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20
##  Algeria    :  12   Asia    :396   Median :1980   Median :60.71
##  Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47
##  Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
##  Australia  :  12                  Max.   :2007   Max.   :82.60
##  (Other)    :1632
##       pop              gdpPercap
##  Min.   :6.001e+04   Min.   :   241.2
##  1st Qu.:2.794e+06   1st Qu.:  1202.1
##  Median :7.024e+06   Median :  3531.8
##  Mean   :2.960e+07   Mean   :  7215.3
##  3rd Qu.:1.959e+07   3rd Qu.:  9325.5
##  Max.   :1.319e+09   Max.   :113523.1
##
```

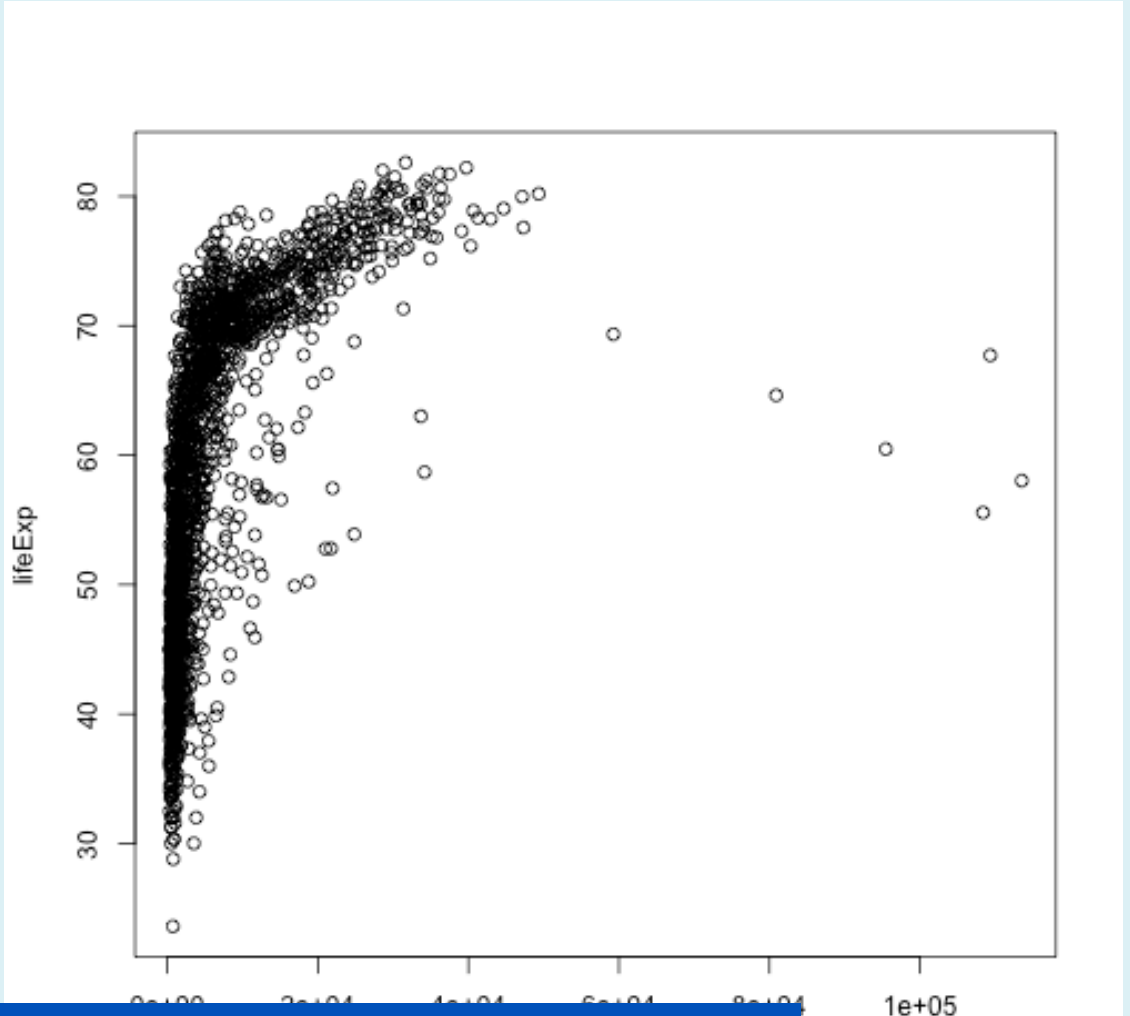
Basic R: Plot

```
plot(lifeExp ~ year, gapminder)
```



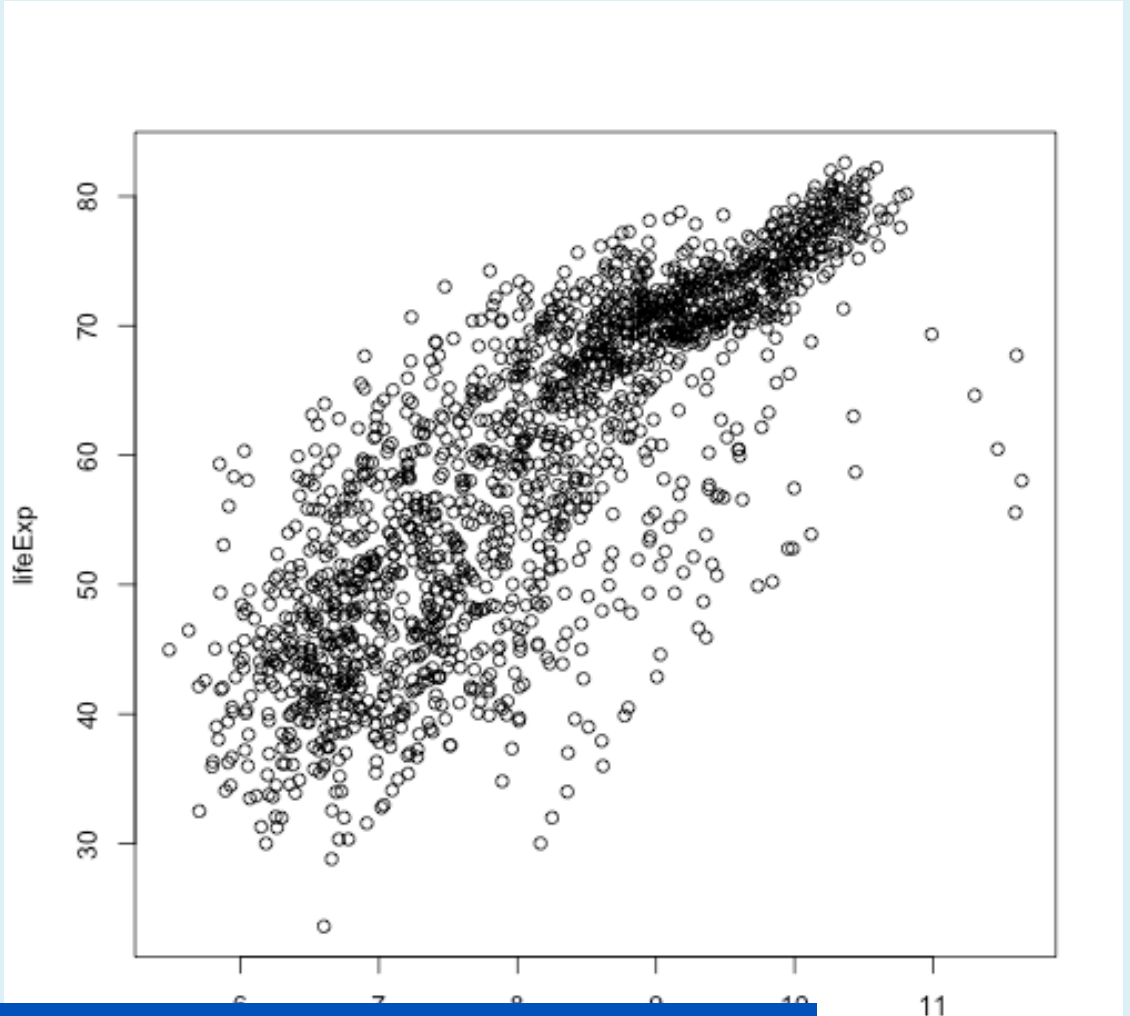
Basic R: Plot

```
plot(lifeExp ~ gdpPercap, gapminder)
```



Basic R: Plot

```
plot(lifeExp ~ log(gdpPercap), gapminder)
```



Basic R: Look at the variable inside a data

```
head(gapminder$lifeExp)
```

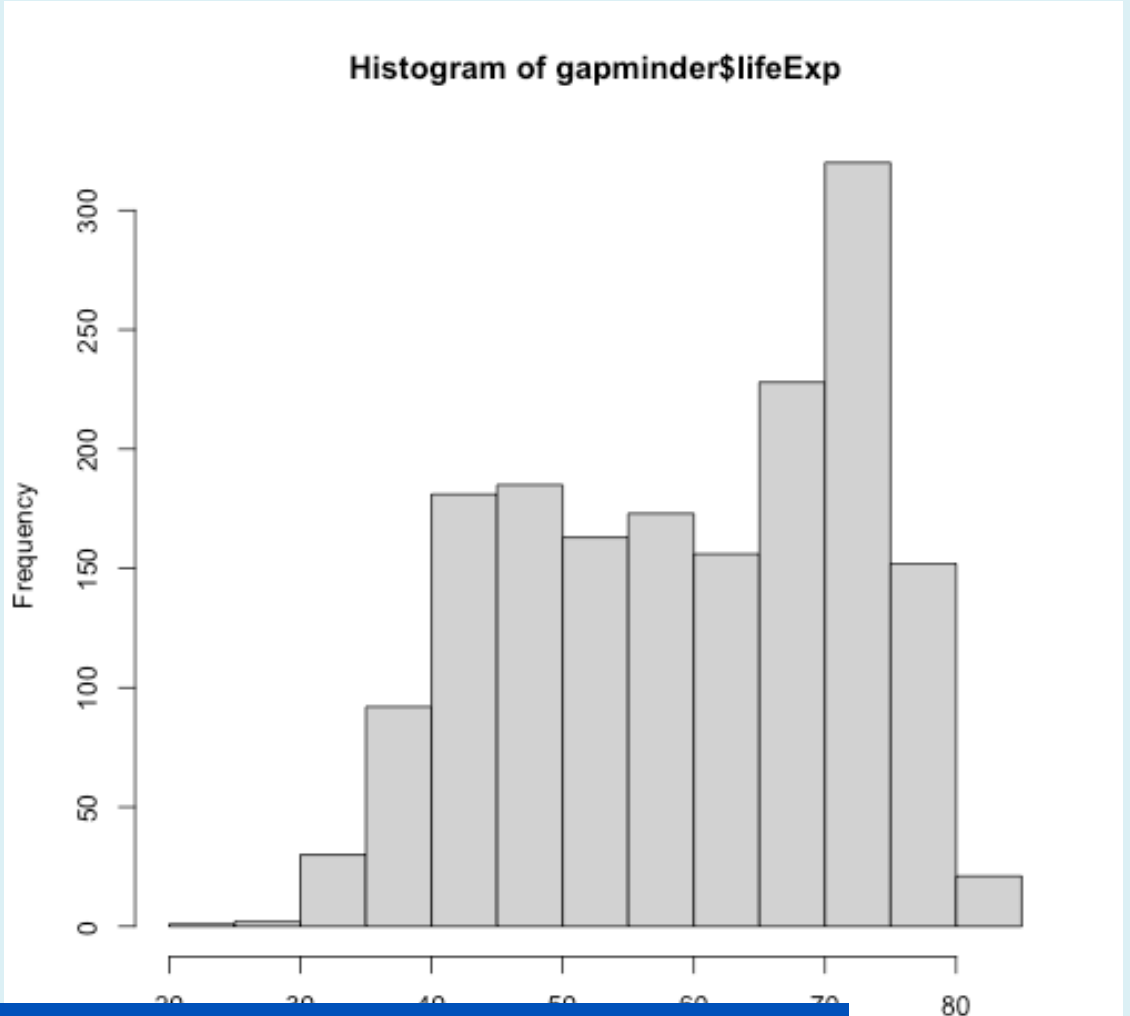
```
## [1] 28.801 30.332 31.997 34.020 36.088 38.438
```

```
summary(gapminder$lifeExp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   23.60   48.20   60.71   59.47   70.85   82.60
```

Basic R: Look at the variable inside a data

```
hist(gapminder$lifeExp)
```



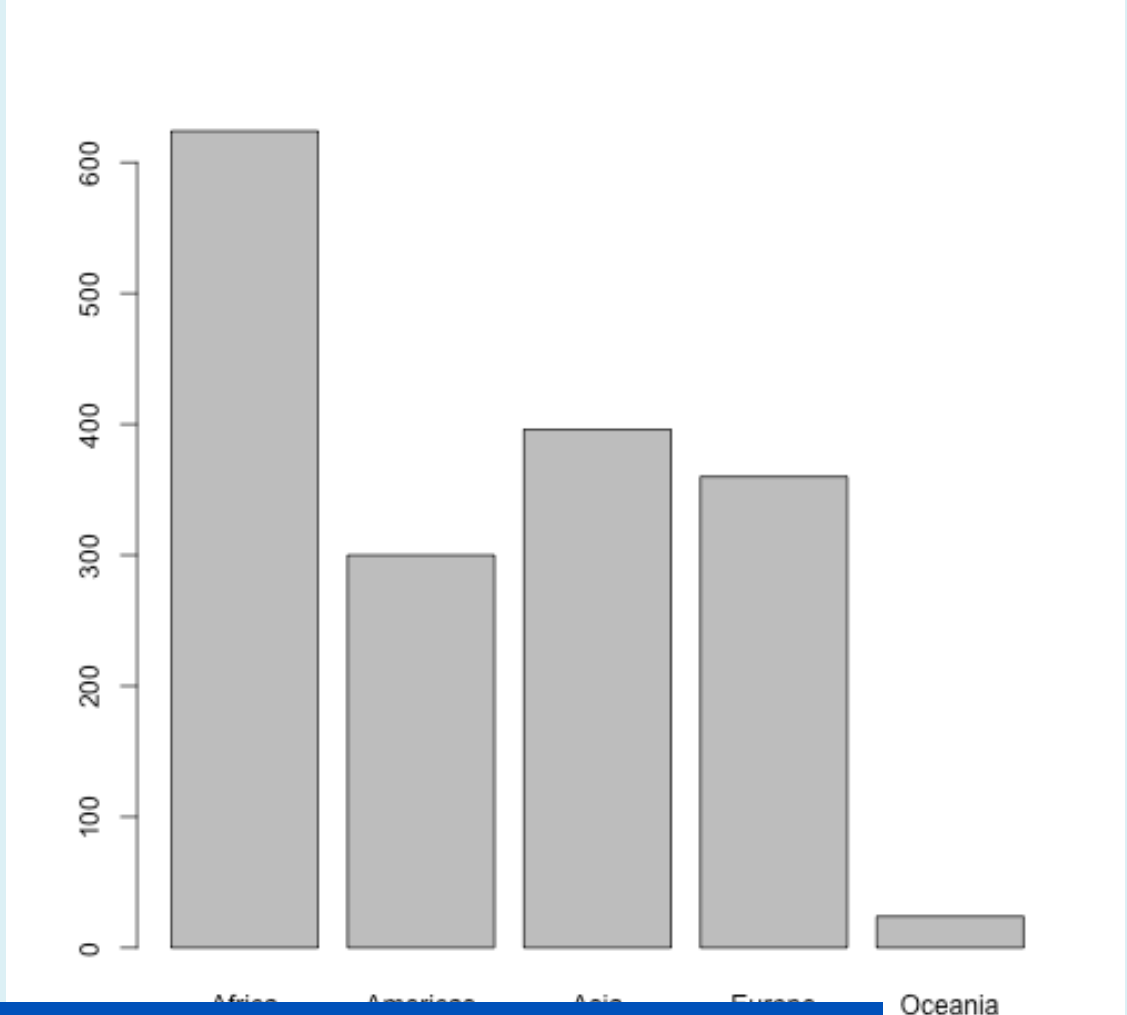
Basic R: Look at the variable inside a data

```
table(gapminder$continent)
```

```
##
##   Africa Americas     Asia   Europe  Oceania
##      624      300      396      360       24
```

Basic R: Look at the variable inside a data

```
barplot(table(gapminder$continent))
```

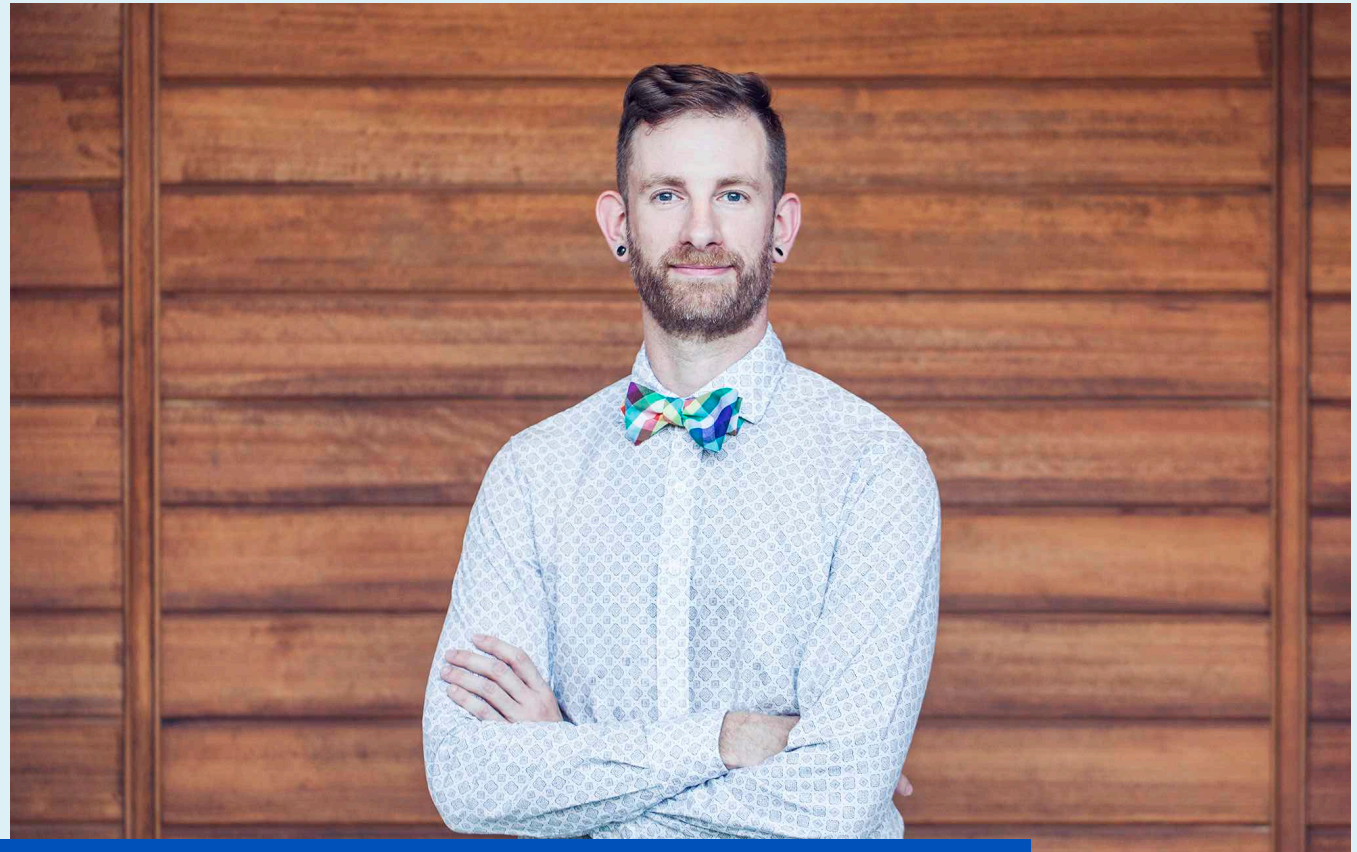


"Fancy" Data Management using Tidyverse

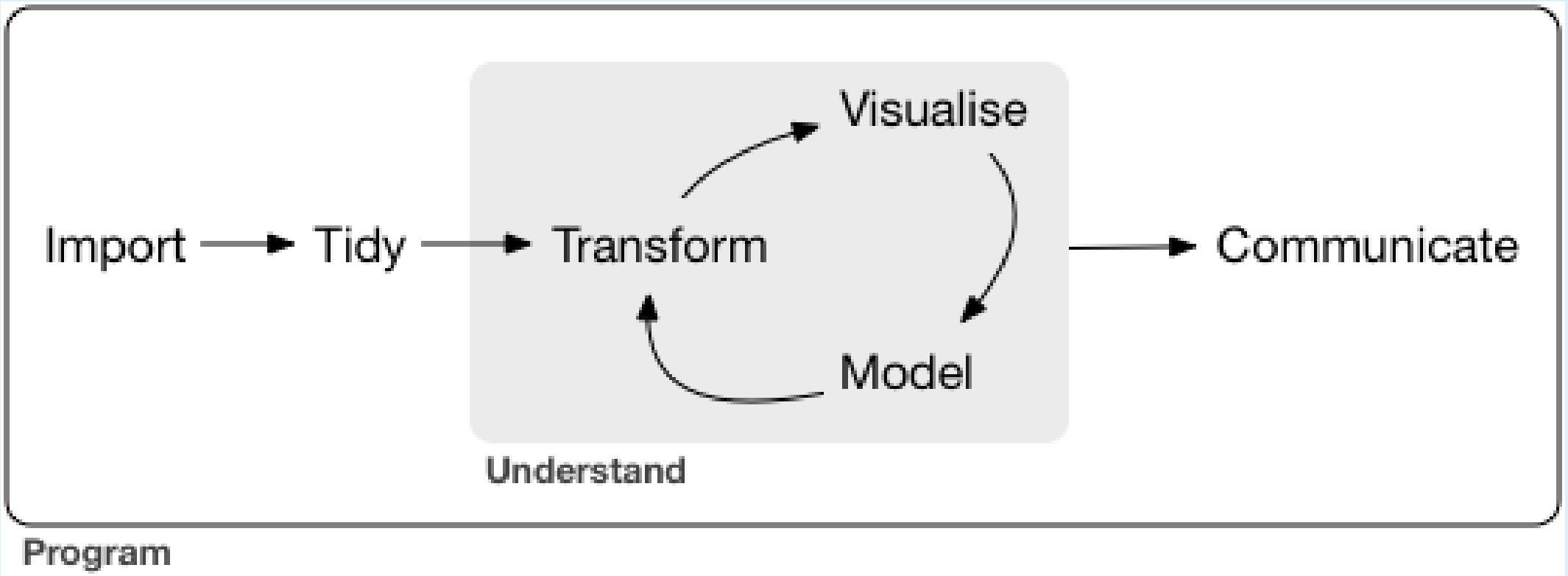
Introduction to `tidydata`

- A set of packages that are useful specifically for *data manipulation*, *exploration* and *visualization* with a common philosophy: *"tidy"* data.

- Developed by a leading figure in R field: Hadley Wickham



The Data Analysis Workflow



Installing and Loading tidyverse

```
library(tidyverse)
```

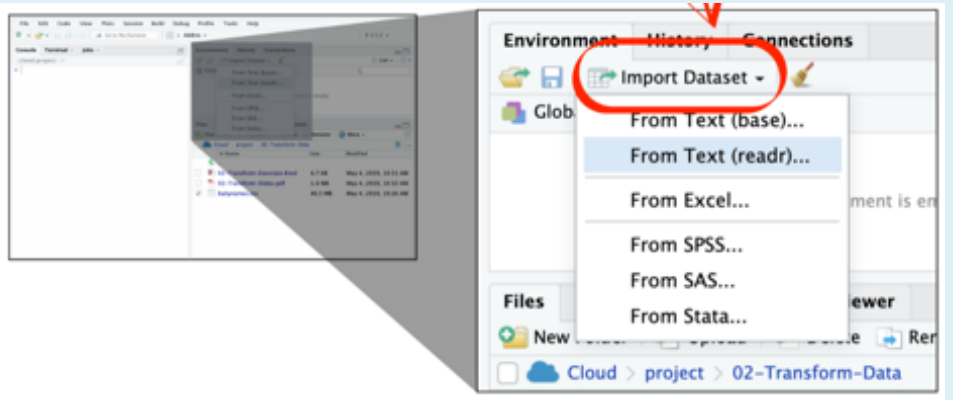
When we load the tidyverse package, six core R packages load:

- readr and haven - data import
- tidyr - data tidying
- dplyr - data wrangling
- ggplot2 - data visualization
- purrr - functional programming
- tibble - modern data frames

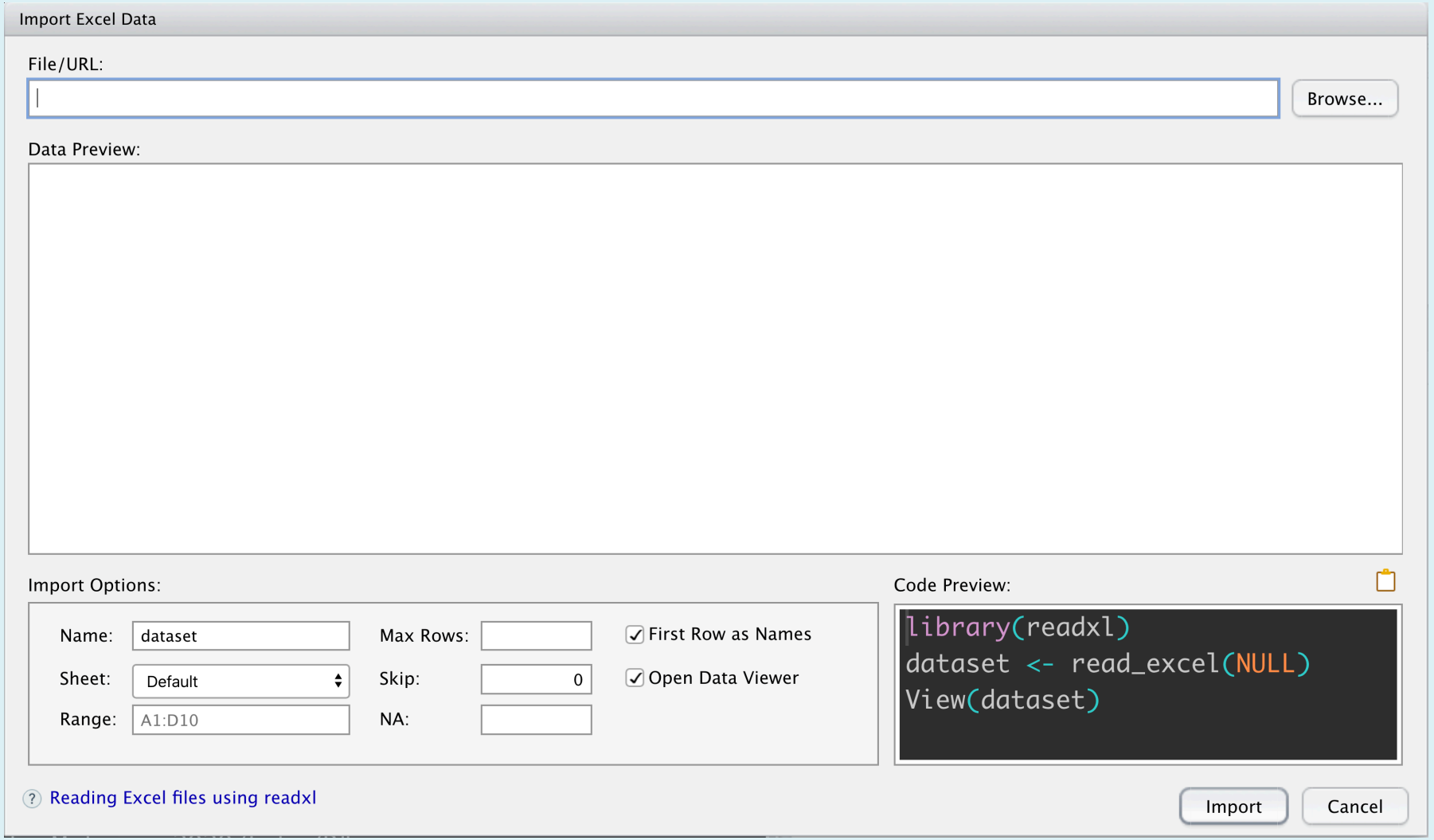
Data Import with `tidyverse`

Import Functions by `readr` and `haven`

- `read_csv()` - comma separated (CSV) files
- `read_tsv()` - tab separated files
- `read_delim()` - delimited files
- `read_table()` - tabular files with white-space separation
- `read_sas()` - reads .sas7bdat + .sas7bcat files
- `read_spss()` - reads .sav files of SPSS
- `read_stata()` - reads .dta files (up to version 15)



Data Import with `readxl`



"Tidy" Data Principles

What does `tidy` data mean?

- Structure: transform, combine and separate variables to make it easier to analyze.
- Quality and Observations: Handle missing values, duplicates, outliers
- Variables: Select and generate appropriate variables, label the variables

Which data is `tidy`?

```
table1
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table2
```

```
## # A tibble: 12 × 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
```

Which data is `tidy`?

```
table3
```

```
## # A tibble: 6 × 3
##   country      year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```
table4a
```

```
## # A tibble: 3 × 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
```

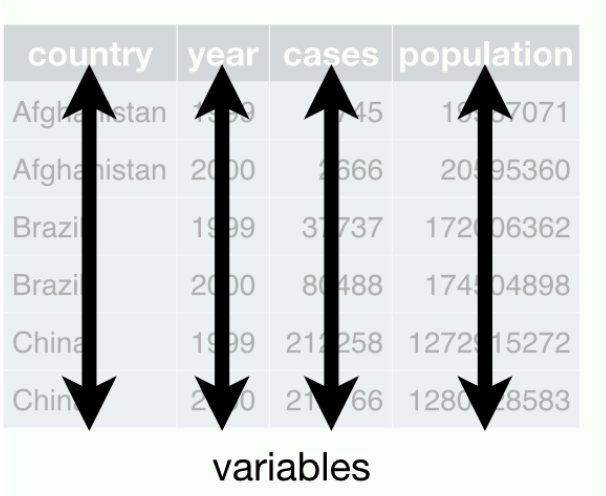
```
table4b
```

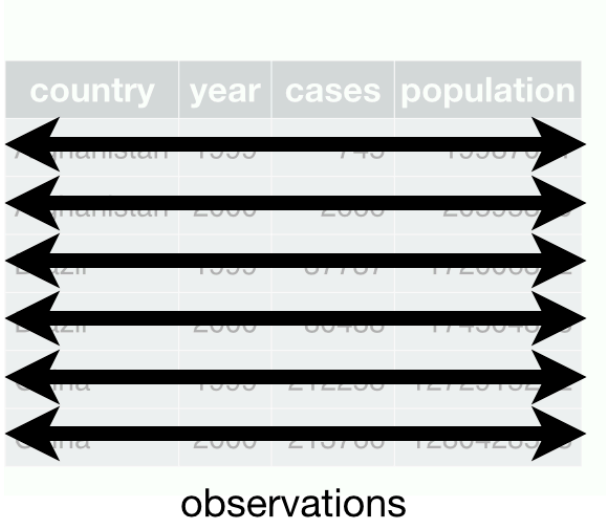
```
## # A tibble: 3 × 3
##   country         `1999`     `2000`
##   <chr>            <dbl>      <dbl>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

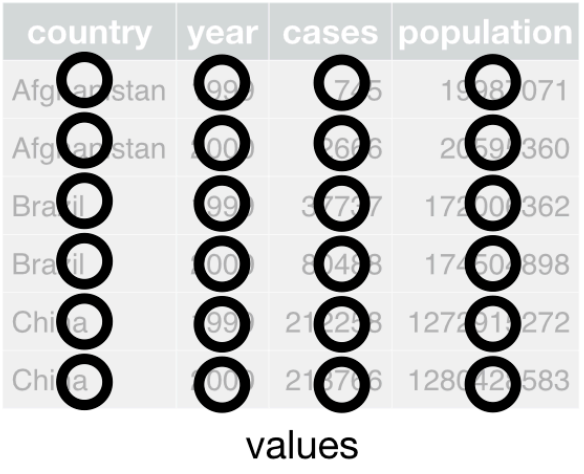
Tidy Data Structure

Three Key Principles:

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table







Core tidyr Functions

tidyr simplifies the process of creating tidy data with four key functions:

- gather() - makes "wide" data longer
- spread() - makes "long" data wider
- separate() - splits a character column into multiple columns
- unite() - combines multiple columns into a single column

Tidy data using `spread()`

```
table1
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table2
```

```
## # A tibble: 12 × 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases            37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases            80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases           212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases           213766
## 12 China        2000 population 1280428583
```

Tidy data using `spread()`

```
library(tidyr)
spread(table2, type, count)
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

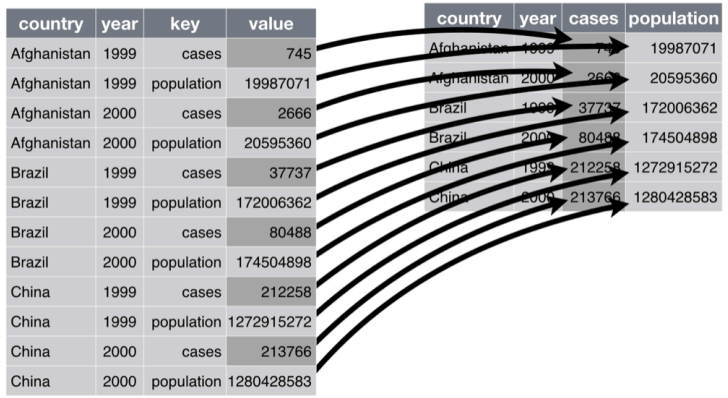


table2

Tidy data using `gather()`

```
table1
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table4a
```

```
## # A tibble: 3 × 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

Tidy data using `gather()`

```
gather(table4a, "year", "cases", 2:3)
```

```
## # A tibble: 6 × 3
##   country     year   cases
##   <chr>       <chr>  <dbl>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Tidy data using `separate()`

```
table1
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table3
```

```
## # A tibble: 6 × 3
##   country      year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

Tidy data using `separate()`

```
separate(table3, rate, into = c("cases", "population"), sep = "/")
```

```
## # A tibble: 6 × 4
##   country      year cases  population
##   <chr>       <dbl> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Tidy data using `unite()`

```
table5 <- separate(table3, rate, into = c("cases", "population"), sep = "/")
```

```
unite(table5, "rate", cases, population, sep = "/")
```

```
## # A tibble: 6 × 3
##   country      year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

Data Wrangling Overview

Review: `tidy` data workflow

- Data Structure Transforming: Reshape data format
- Data Cleaning: Deal with missing values and duplicates
- Data Relating: Merge or join multiple datasets
- Data Wrangling: Select and create variables, filter observations

Data Wrangling with `dplyr`

Key `dplyr` Verbs

The most important `dplyr` functions:

`dplyr` verbs	Description
`select()`	select columns
`filter()`	filter rows
`arrange()`	re-order or arrange rows
`mutate()`	create new columns
`rename()`	rename columns
`summarize()`	summarize values
`group_by()`	allows for group operations

The Pipe Operator: `%>%` and `|>`

How Pipes Work

- Pipes allow you to chain functions together by passing the output from one function as the first argument to the next function
- Instead of nesting functions (reading from inside to outside), pipes let you read functions from left to right

Two Types of Pipes in R:

- `|>` (Native pipe): Built into R 4.1+ (2021), no packages needed
- `%>%` (magrittr pipe): From `magrittr` package, widely used in tidyverse

```
library(dplyr)
library(magrittr)  # for %>%
```

Pipe Operator Examples

Traditional Nested Functions vs Pipes

Nested (hard to read):

```
summarize(
  group_by(
    filter(gapminder, year == 2007),
    continent
  ),
  avg_life = mean(lifeExp)
)
```

With Pipes (easy to read):

```
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarize(avg_life = mean(lifeExp))
```

Both `%>%` and `|>` work the same way for basic operations!

Selecting Columns with `select()`

Drop Columns

To select all columns except a specific column, use the "-" operator:

```
gapminder %>%
  select(-lifeExp)
```

```
## # A tibble: 1,704 × 5
##    country     continent  year      pop gdpPercap
##    <fct>       <fct>     <int>    <int>     <dbl>
##  1 Afghanistan Asia       1952  8425333      779.
##  2 Afghanistan Asia       1957  9240934      821.
##  3 Afghanistan Asia       1962 10267083      853.
##  4 Afghanistan Asia       1967 11537966      836.
##  5 Afghanistan Asia       1972 13079460      740.
##  6 Afghanistan Asia       1977 14880372      786.
##  7 Afghanistan Asia       1982 12881816      978.
##  8 Afghanistan Asia       1987 13867957      852.
##  9 Afghanistan Asia       1992 16317921      649.
## 10 Afghanistan Asia       1997 22227415      635.
```

Selecting Columns with `select()`

Select Range of Columns

To select a range of columns by name, use the ":" operator:

```
gapminder %>%
  select(lifeExp:gdpPercap)
```

```
## # A tibble: 1,704 × 3
##    lifeExp      pop gdpPercap
##      <dbl>    <int>     <dbl>
##  1    28.8  8425333      779.
##  2    30.3  9240934      821.
##  3    32.0 10267083      853.
##  4    34.0 11537966      836.
##  5    36.1 13079460      740.
##  6    38.4 14880372      786.
##  7    39.9 12881816      978.
##  8    40.8 13867957      852.
##  9    41.7 16317921      649.
## 10    41.8 22227415      635.
```

Additional `select()` Options

Selection Function	Meaning
`starts_with("c")`	Select columns starting with "c"
`ends_with()`	Select columns ending with string
`contains()`	Select columns containing string
`matches()`	Select columns matching regex
`one_of()`	Select columns from group of names

Selecting Rows with `filter()`

Filter by Year

Only want data from 2007:

```
gapminder %>%
  filter(year == 2007)
```

```
## # A tibble: 142 × 6
##    country     continent  year lifeExp       pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Afghanistan Asia       2007    43.8  31889923      975.
##  2 Albania     Europe     2007    76.4   3600523     5937.
##  3 Algeria     Africa     2007    72.3  33333216     6223.
##  4 Angola      Africa     2007    42.7  12420476     4797.
##  5 Argentina   Americas   2007    75.3  40301927    12779.
##  6 Australia   Oceania    2007    81.2  20434176    34435.
##  7 Austria     Europe     2007    79.8   8199783    36126.
##  8 Bahrain     Asia       2007    75.6    708573    29796.
##  9 Bangladesh  Asia       2007    64.1 150448339     1391.
## 10 Belgium     Europe     2007    79.4  10392226    33693.
```

Selecting Rows with `filter()`

Filter by Country

Only want data from China:

```
gapminder %>%
  filter(country == "China")
```

```
## # A tibble: 12 × 6
##    country continent  year lifeExp        pop gdpPercap
##    <fct>   <fct>     <int>   <dbl>      <int>     <dbl>
##  1 China   Asia       1952    44    556263527      400.
##  2 China   Asia       1957    50.5  637408000      576.
##  3 China   Asia       1962    44.5  665770000      488.
##  4 China   Asia       1967    58.4  754550000      613.
##  5 China   Asia       1972    63.1  862030000      677.
##  6 China   Asia       1977    64.0  943455000      741.
##  7 China   Asia       1982    65.5 1000281000      962.
##  8 China   Asia       1987    67.3 1084035000     1379.
##  9 China   Asia       1992    68.7 1164970000     1656.
## 10 China   Asia       1997    70.4 1230075000     2289.
```

Advanced `filter()` with Boolean Operators

Multiple Conditions

Use boolean operators (`>`, `<`, `>=`, `<=`, `!=`, `%in%`) for logical tests:

```
gapminder %>%
  filter(country == "China", year > 1977)
```

```
## # A tibble: 6 × 6
##   country continent  year lifeExp        pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>      <int>     <dbl>
## 1 China   Asia       1982    65.5 1000281000      962.
## 2 China   Asia       1987    67.3 1084035000     1379.
## 3 China   Asia       1992    68.7 1164970000     1656.
## 4 China   Asia       1997    70.4 1230075000     2289.
## 5 China   Asia       2002    72.0 1280400000     3119.
## 6 China   Asia       2007    73.0 1318683096     4959.
```

Creating New Variables with `mutate()`

Basic `mutate()`

`mutate()` defines and inserts new variables into a tibble:

```
gapminder %>%
  mutate(gdp = pop * gdpPercap)
```

```
## # A tibble: 1,704 × 7
##    country     continent  year lifeExp      pop gdpPercap          gdp
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>        <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.  6567086330.
##  2 Afghanistan Asia       1957    30.3  9240934      821.  7585448670.
##  3 Afghanistan Asia       1962    32.0 10267083      853.  8758855797.
##  4 Afghanistan Asia       1967    34.0 11537966      836.  9648014150.
##  5 Afghanistan Asia       1972    36.1 13079460      740.  9678553274.
##  6 Afghanistan Asia       1977    38.4 14880372      786. 11697659231.
##  7 Afghanistan Asia       1982    39.9 12881816      978. 12598563401.
##  8 Afghanistan Asia       1987    40.8 13867957      852. 11820990309.
##  9 Afghanistan Asia       1992    41.7 16317921      649. 10595901589.
## 10 Afghanistan Asia       1997    41.8 22227415      635. 14121995875.
```

Creating New Variables with `mutate()`

Chaining Operations

```
gapminder %>%
  filter(year == 2007) %>%          # only data in 2007
  select(-continent) %>%            # drop continent variable
  mutate(gdp = pop * gdpPercap)     # generate new variable "gdp"
```

```
## # A tibble: 142 × 6
##    country      year lifeExp       pop gdpPercap           gdp
##    <fct>       <int>   <dbl>     <int>     <dbl>         <dbl>
##  1 Afghanistan  2007    43.8  31889923      975.  31079291949.
##  2 Albania      2007    76.4   3600523     5937.  21376411360.
##  3 Algeria      2007    72.3  33333216     6223. 207444851958.
##  4 Angola       2007    42.7  12420476     4797.  59583895818.
##  5 Argentina    2007    75.3  40301927    12779. 515033625357.
##  6 Australia    2007    81.2  20434176    34435. 703658358894.
##  7 Austria      2007    79.8   8199783    36126. 296229400691.
##  8 Bahrain      2007    75.6    708573    29796.  21112675360.
##  9 Bangladesh   2007    64.1 150448339     1391. 209311822134.
## 10 Belgium      2007    79.4  10392226    33693. 350141166520.
```

Renaming Variables with `rename()`

```
gapminder %>%
  rename(life_exp = lifeExp,
         gdp_percap = gdpPercap)
```

```
## # A tibble: 1,704 × 6
##    country     continent  year life_exp      pop gdp_percap
##    <fct>       <fct>     <int>    <dbl>    <int>      <dbl>
##  1 Afghanistan Asia       1952     28.8  8425333       779.
##  2 Afghanistan Asia       1957     30.3  9240934       821.
##  3 Afghanistan Asia       1962     32.0 10267083       853.
##  4 Afghanistan Asia       1967     34.0 11537966       836.
##  5 Afghanistan Asia       1972     36.1 13079460       740.
##  6 Afghanistan Asia       1977     38.4 14880372       786.
##  7 Afghanistan Asia       1982     39.9 12881816       978.
##  8 Afghanistan Asia       1987     40.8 13867957       852.
##  9 Afghanistan Asia       1992     41.7 16317921       649.
## 10 Afghanistan Asia       1997     41.8 22227415       635.
## # ℹ 1,694 more rows
```

Sorting Data with `arrange()`

Ascending Order

```
gapminder %>%
  filter(year == 2007, continent == "Asia") %>%
  select(-c(continent, pop, gdpPercap)) %>%
  arrange(lifeExp)
```

```
## # A tibble: 33 × 3
##    country      year lifeExp
##    <fct>       <int>   <dbl>
##  1 Afghanistan  2007    43.8
##  2 Iraq         2007    59.5
##  3 Cambodia     2007    59.7
##  4 Myanmar      2007    62.1
##  5 Yemen, Rep.  2007    62.7
##  6 Nepal        2007    63.8
##  7 Bangladesh   2007    64.1
##  8 India        2007    64.7
##  9 Pakistan     2007    65.5
## 10 Mongolia     2007    66.8
```

Sorting Data with `arrange()`

Descending Order

```
gapminder %>%
  filter(year == 2007, continent == "Asia") %>%
  select(-c(continent, gdpPercap)) %>%
  arrange(desc(lifeExp))  # descending order
```

```
## # A tibble: 33 × 4
##    country           year lifeExp      pop
##    <fct>            <int>   <dbl>    <int>
##  1 Japan             2007    82.6 127467972
##  2 Hong Kong, China  2007    82.2   6980412
##  3 Israel            2007    80.7   6426679
##  4 Singapore         2007    80.0   4553009
##  5 Korea, Rep.       2007    78.6  49044790
##  6 Taiwan            2007    78.4  23174294
##  7 Kuwait            2007    77.6   2505559
##  8 Oman              2007    75.6   3204897
##  9 Bahrain           2007    75.6    708573
## 10 Vietnam           2007    74.2  85262356
```

Summary Statistics

Group by Continent

```
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent, year) %>%
  summarize(gdpPercap_con = mean(gdpPercap)) %>%
  select(c(continent, gdpPercap_con, year)) %>%
  arrange(desc(gdpPercap_con))  # descending order
```

```
## `summarise()` has grouped output by 'continent'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 5 × 3
## # Groups:   continent [5]
##   continent gdpPercap_con  year
##   <fct>             <dbl> <int>
## 1 Oceania          29810.  2007
## 2 Europe           25054.  2007
## 3 Asia             12473.  2007
## 4 Americas         11003.  2007
```

Summary Statistics

Group by Country

```
gapminder %>%
  filter(continent == "Asia") %>%
  group_by(country) %>%
  summarize(gdpPercap_coun = mean(gdpPercap)) %>%
  arrange(desc(gdpPercap_coun))  # descending order
```

```
## # A tibble: 33 × 2
##    country          gdpPercap_coun
##    <fct>                     <dbl>
##  1 Kuwait                   65333.
##  2 Saudi Arabia             20262.
##  3 Bahrain                  18078.
##  4 Japan                    17751.
##  5 Singapore                17425.
##  6 Hong Kong, China         16229.
##  7 Israel                   14161.
##  8 Oman                     12139.
##  9 Taiwan                   10225.
```

Summary Statistics with `count()`

```
gapminder %>%
  filter(year == 2002) %>%
  count(continent, sort = TRUE)
```

```
## # A tibble: 5 × 2
##   continent     n
##   <fct>     <int>
## 1 Africa       52
## 2 Asia         33
## 3 Europe       30
## 4 Americas     25
## 5 Oceania       2
```

Practice Exercise

Question

Which country in which continent experienced the sharpest 5-year drop in life expectancy?

```
gapminder %>%
  select(country, year, continent, lifeExp) %>%
  group_by(continent, country) %>%
  # within country, calculate year-over-year change
  mutate(le_delta = lifeExp - lag(lifeExp)) %>%
  # retain the worst life expectancy change (most negative)
  summarize(worst_le_delta = min(le_delta, na.rm = TRUE)) %>%
  # within continent, find the lowest worst_le_delta
  top_n(-1, wt = worst_le_delta) %>%
  arrange(worst_le_delta)
```

Practice Exercise

Answer

```
## `summarise()` has grouped output by 'continent'. You can override using the
## `.groups` argument.

## # A tibble: 5 × 3
## # Groups:   continent [5]
##   continent country     worst_le_delta
##   <fct>     <fct>                <dbl>
## 1 Africa    Rwanda             -20.4
## 2 Asia      Cambodia            -9.10
## 3 Americas  El Salvador         -1.51
## 4 Europe    Montenegro          -1.46
## 5 Oceania   Australia            0.170
```

References

1. Grolemund & Wickham (2017), R for Data Science
2. DataCamp (2018), Introduction to Tidyverse