# Session 3: Data Processing

R for Stata Users

Rony Rodrigo Maximiliano Rodriguez-Ramirez
The World Bank – DIME | WB Github

16 November 2020

## Table of contents

## Introduction

## Introduction

### Goals of this session

- To organize data in a way that it will be easier to analyze it and communicate it.
- We'll use a set of packages that are bundled into something called the `tidyverse`.

## Introduction

### Goals of this session

- To organize data in a way that it will be easier to analyze it and communicate it.
- We'll use a set of packages that are bundled into something called the `tidyverse`.

### Things to keep in mind

- We'll take you through the same steps we've taken when we were preparing the datasets.
- In most cases, your datasets won't be `tidy`.

> **Tidy data**: A dataset is said to be tidy if it satisfies the following conditions:
>
> 1. observations are in rows
> 2. variables are in columns
> 3. contained in a single dataset.

Takeaway: long format > wide format

## Introduction

- In this session, you'll be introduced to some basic conceptos of data cleaning in R. We will cover:

1. Exploring a dataset;
2. Creating new variables;
3. Filtering and subsetting datasets;
4. Merging datasets;
5. Dealing with factor variables;
6. Saving data.

## Introduction

- In this session, you'll be introduced to some basic conceptos of data cleaning in R. We will cover:

1. Exploring a dataset;
2. Creating new variables;
3. Filtering and subsetting datasets;
4. Merging datasets;
5. Dealing with factor variables;
6. Saving data.

> There are many other tasks that we usually perform as part of data cleaning that are beyond the scope of this session.

## Introduction

- Before we start, let's makue sure we are all set:

1. Start a fresh session.
2. Load the tidyverse package.
3. Set your file paths.

# Tidyverse packages

Let's load the tidyverse meta-package:

```r
# If you haven't installed the package uncomment the next line
# install.package("tidyverse", dependencies = TRUE)

# Load packages
library(tidyverse)
library(janitor)
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

> Remember that you should always load your packages before your start coding.

## File paths

For this session, my file paths are as follows. We will use them to load and export datasets.

```r
# Example of my filepaths for this presentation
project           ← "D:/Documents/RA Jobs/DIME/trainings/dime-r-training"
dataWorkFolder    ← file.path(projectFolder,"DataWork")
Data              ← file.path(dataWorkFolder,"DataSets")
finalData         ← file.path(Data,"Final")
rawData           ← file.path(Data,"Raw")
rawOutput         ← file.path(dataWorkFolder,"Output","Raw")
```

## Just in case: Check your R and RStudio versions

☑ R version:

```
version$version.string
```

```
## [1] "R version 4.0.3 (2020-10-10)"
```

☑ RStudio version:

```
# Use the following function to get the version: RStudio.Version()$version
# [1] ‘1.3.1073’
```

☑ Packages:

```
update.packages(ask = FALSE, checkBuilt = TRUE)
```

## Loading a dataset in R

Before we start wrangling our data, let's read ourdataset. In R, we can use the `read.csv` function from Base R, or `read_csv` from the `readr` package if we want to load a CSV file. For this exercise, we are going to the World Happiness Report (2015-2018)

> *Background of the data*: This data comes from the US Census's archives

```
whr15 <- read_csv(file.path(rawData, "Un WHR" , "WHR2015.csv")) %>% clean_names()
whr16 <- read_csv(file.path(rawData, "Un WHR" , "WHR2016.csv")) %>% clean_names()
whr17 <- read_csv(file.path(rawData, "Un WHR" , "WHR2017.csv")) %>% clean_names()
```

# Load and show a dataset

We can just show our dataset using the name of the object; in this case, `census`.

```
whr15
```

```
## # A tibble: 158 x 12
##    country region happiness_rank happiness_score standard_error economy_gdp_per~
##    <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
##  1 Switze~ Weste~              1            7.59         0.0341             1.40
##  2 Iceland Weste~              2            7.56         0.0488             1.30
##  3 Denmark Weste~              3            7.53         0.0333             1.33
##  4 Norway  Weste~              4            7.52         0.0388             1.46
##  5 Canada  North~              5            7.43         0.0355             1.33
##  6 Finland Weste~              6            7.41         0.0314             1.29
##  7 Nether~ Weste~              7            7.38         0.0280             1.33
##  8 Sweden  Weste~              8            7.36         0.0316             1.33
##  9 New Ze~ Austr~              9            7.29         0.0337             1.25
## 10 Austra~ Austr~             10            7.28         0.0408             1.33
## # ... with 148 more rows, and 6 more variables: family <dbl>,
## #   health_life_expectancy <dbl>, freedom <dbl>,
## #   trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>
```

## Data wrangling

## Exploring a data set

Some useful functions from base R:

- `View()`: open the data set
- `class()`: reports object type of type of data stored.
- `dim()`: reports the size of each one of an object's dimension.
- `names()`: returns the variable names of a dataset.
- `str()`: general information on an R object.
- `summary()`: summary information about the variables in a data frame.
- `head()`: shows the first few observations in the dataset.
- `tail()`: shows the last few observations in the dataset.

Some other useful functions from the tidyverse:

- `glimpse()`: get a glimpse of your data

# Glimpse your data

This functions give your information about your variables (e.g., type, row, columns,)

```
whr15 %>%
  glimpse()
```

```
## Rows: 158
## Columns: 12
## $ country                      <chr> "Switzerland", "Iceland", "Denmark", "N…
## $ region                       <chr> "Western Europe", "Western Europe", "We…
## $ happiness_rank               <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,  …
## $ happiness_score              <dbl> 7.587, 7.561, 7.527, 7.522, 7.427, 7.40…
## $ standard_error               <dbl> 0.03411, 0.04884, 0.03328, 0.03880, 0.0…
## $ economy_gdp_per_capita       <dbl> 1.39651, 1.30232, 1.32548, 1.45900, 1.3…
## $ family                       <dbl> 1.34951, 1.40223, 1.36058, 1.33095, 1.3…
## $ health_life_expectancy       <dbl> 0.94143, 0.94784, 0.87464, 0.88521, 0.9…
## $ freedom                      <dbl> 0.66557, 0.62877, 0.64938, 0.66973, 0.6…
## $ trust_government_corruption  <dbl> 0.41978, 0.14145, 0.48357, 0.36503, 0.3…
## $ generosity                   <dbl> 0.29678, 0.43630, 0.34139, 0.34699, 0.4…
## $ dystopia_residual            <dbl> 2.51738, 2.70201, 2.49204, 2.46531, 2.4…
```

# dplyr:filter

Filter or subsetting a dataset.

```r
whr15 %>%
  filter(region == "Western Europe",
         happiness_rank <= 10)
```

```
## # A tibble: 7 x 12
##   country region happiness_rank happiness_score standard_error economy_gdp_per~
##   <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
## 1 Switze~ Weste~              1            7.59         0.0341             1.40
## 2 Iceland Weste~              2            7.56         0.0488             1.30
## 3 Denmark Weste~              3            7.53         0.0333             1.33
## 4 Norway  Weste~              4            7.52         0.0388             1.46
## 5 Finland Weste~              6            7.41         0.0314             1.29
## 6 Nether~ Weste~              7            7.38         0.0280             1.33
## 7 Sweden  Weste~              8            7.36         0.0316             1.33
## # ... with 6 more variables: family <dbl>, health_life_expectancy <dbl>,
## #   freedom <dbl>, trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>
```

# dplyr:filter regular expressions

One advantage of the filter command over Stata is that you can also integrate regular expressions. Let's say that we want to subset all regions' divisions that have `East` in their names. We can use the following:

```r
whr15 %>%
  filter(grepl("America", region)) %>%
  head(5)
```

```
## # A tibble: 5 x 12
##   country region happiness_rank happiness_score standard_error economy_gdp_per~
##   <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
## 1 Canada  North~              5            7.43         0.0355            1.33 
## 2 Costa ~ Latin~             12            7.23         0.0445            0.956
## 3 Mexico  Latin~             14            7.19         0.0418            1.02 
## 4 United~ North~             15            7.12         0.0384            1.39 
## 5 Brazil  Latin~             16            6.98         0.0408            0.981
## # ... with 6 more variables: family <dbl>, health_life_expectancy <dbl>,
## #   freedom <dbl>, trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>
```

> Notice that I have used `head()` to show just the first 8 observations of the subset. If you want to save this subset you can assign it to an objet. For example `census_east ← + and the code above`.

# dplyr:filter missing cases

If case you want to remove the missing cases for a specif variable, you can use `!is.na()`. Now we have a dataset that contains information per region and division without missing values.

```
whr15 %>%
  filter(!is.na(region)) %>%
  head(5)
```

```
## # A tibble: 5 x 12
##   country region happiness_rank happiness_score standard_error economy_gdp_per~
##   <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
## 1 Switze~ Weste~              1            7.59         0.0341             1.40
## 2 Iceland Weste~              2            7.56         0.0488             1.30
## 3 Denmark Weste~              3            7.53         0.0333             1.33
## 4 Norway  Weste~              4            7.52         0.0388             1.46
## 5 Canada  North~              5            7.43         0.0355             1.33
## # ... with 6 more variables: family <dbl>, health_life_expectancy <dbl>,
## #   freedom <dbl>, trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>
```

> Notice that we are negating a function, i.e., !
> In case we want to keep the observations that contains missing information we will only use `is.na()`.

# Other relevant functions: slice, subset, select

Arrange Slice Select Combining functions

`Arrange`: allows you to order by a specific column.

```
whr15 %>%
  arrange(region, country) %>%
  head(5)
```

```
## # A tibble: 5 x 12
##   country region happiness_rank happiness_score standard_error economy_gdp_per~
##   <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
## 1 Austra~ Austr~             10            7.28         0.0408           1.33
## 2 New Ze~ Austr~              9            7.29         0.0337           1.25
## 3 Albania Centr~             95            4.96         0.0501           0.879
## 4 Armenia Centr~            127            4.35         0.0476           0.768
## 5 Azerba~ Centr~             80            5.21         0.0336           1.02
## # ... with 6 more variables: family <dbl>, health_life_expectancy <dbl>,
## #   freedom <dbl>, trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>
```

## ID variables

## ID variables

Dimensions of your data:

```r
dim(whr15)
```

```
## [1] 158  12
```

```r
dim_desc(whr15)
```

```
## [1] "[158 x 12]"
```

The number of distinct values of a particular variable:

```r
n_distinct(whr15$region, na.rm = TRUE)
```

```
## [1] 10
```

```r
n_distinct(whr15$country, na.rm = TRUE)
```

```
## [1] 158
```

## ID variables

We can also test whether the number of rows is equal to the number of distinct values in a specific variable as follows :

```
n_distinct(whr15$country, na.rm = TRUE) == nrow(whr15)
```

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

```
n_distinct(whr16$country, na.rm = TRUE) == nrow(whr16)
```

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

```
n_distinct(whr17$country, na.rm = TRUE) == nrow(whr17)
```

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

## Comparing two dataframes

**setdiff()**: Prints all the elements of the first object that are not in the second object (ignores duplicates).

We can use this function to see which countries are coming in and out of the WHR dataset set between 2015 and 2016.

```r
# Any countries in 2015 that are not in 2016?
setdiff(whr15$country, whr16$country)
```

```
## [1] "Oman"                     "Somaliland region"
## [3] "Mozambique"               "Lesotho"
## [5] "Swaziland"                "Djibouti"
## [7] "Central African Republic"
```

```r
# And vice-versa
setdiff(whr16$country, whr15$country)
```

```
## [1] "Puerto Rico"       "Belize"            "Somalia"
## [4] "Somaliland Region" "Namibia"           "South Sudan"
```

## Replacing values

You might notice in the last slide, that *Somaliland region* and *Somaliland Region* are not considered the same. We can fix this as follows using base R:

```r
whr15$country[whr15$country == "Somaliland region"] <- "Somaliland Region"
```

Now, if run again `setdiff` again, we get:

```r
# Any countries in 2015 that are not in 2016?
setdiff(whr15$country, whr16$country)
```

```
## [1] "Oman"                     "Mozambique"
## [3] "Lesotho"                  "Swaziland"
## [5] "Djibouti"                 "Central African Republic"
```

```r
# And vice-versa
setdiff(whr16$country, whr15$country)
```

```
## [1] "Puerto Rico" "Belize"      "Somalia"     "Namibia"     "South Sudan"
```

## Replacing values

You might notice in the last slide, that *Somaliland region* and *Somaliland Region* are not considered the same. We can fix this as follows using base R:

```
whr15$country[whr15$country == "Somaliland region"] <- "Somaliland Region"
```

Now, if run again `setdiff` again, we get:

```
# Any countries in 2015 that are not in 2016?
setdiff(whr15$country, whr16$country)
```

```
## [1] "Oman"                        "Mozambique"
## [3] "Lesotho"                     "Swaziland"
## [5] "Djibouti"                    "Central African Republic"
```

```
# And vice-versa
setdiff(whr16$country, whr15$country)
```

```
## [1] "Puerto Rico" "Belize"      "Somalia"     "Namibia"     "South Sudan"
```

> *Notes*: We are going to see different ways of handling this replacement.

## Creating new variables

# Creating new variables

## In the tidyverse, we refer to creating variables as mutating

So, instead of **gen**erate, we use mutate(). Let's say we want to have ratios:

```
whr15 %>%
  arrange(region, country, -happiness_rank) %>%
  mutate(
    hap_hle = happiness_score * health_life_expectancy,
  ) %>%
  select(country:happiness_score, health_life_expectancy, hap_hle) %>%
  head(5)
```

```
## # A tibble: 5 x 6
##   country   region      happiness_rank happiness_score health_life_expe~ hap_hle
##   <chr>     <chr>                <dbl>           <dbl>             <dbl>   <dbl>
## 1 Australia Australia ~             10            7.28             0.932    6.79
## 2 New Zeal~ Australia ~              9            7.29             0.908    6.62
## 3 Albania   Central an~             95            4.96             0.813    4.03
## 4 Armenia   Central an~            127            4.35             0.730    3.18
## 5 Azerbaij~ Central an~             80            5.21             0.640    3.34
```

## Creating new variables: Dummy variables

```r
whr15 %>%
  mutate(
    happiness_score_6 = (happiness_score > 6)
  )
```

Q Well, what do you think it is happening to this variable?

## Creating new variables: Dummy variables

```
whr15 %>%
  mutate(
    happiness_score_6 = (happiness_score > 6)
  )
```

Q Well, what do you think it is happening to this variable?

A The variable we created contains either TRUE or FALSE.
If we want to have it as a numeric (1 or 0), we could include `as.numeric()`

## Creating new variables: Dummy variables

```
whr15 %>%
  mutate(
    happiness_score_6 = (happiness_score > 6)
  )
```

Q Well, what do you think it is happening to this variable?

A The variable we created contains either TRUE or FALSE.
If we want to have it as a numeric (1 or 0), we could include `as.numeric()`

```
whr15 %>%
  mutate(
    happiness_score_6 = as.numeric((happiness_score > 6))
  )
```

## Creating new variables: Dummy variables

```
whr15 %>%
  mutate(
    happiness_score_6 = (happiness_score > 6)
  )
```

Q Well, what do you think it is happening to this variable?

A The variable we created contains either TRUE or FALSE.
If we want to have it as a numeric (1 or 0), we could include `as.numeric()`

```
whr15 %>%
  mutate(
    happiness_score_6 = as.numeric((happiness_score > 6))
  )
```

Finally, instead of using a random number such as 6, we can do the following:

```
whr15 %>%
  mutate(
    happiness_high_mean = as.numeric((happiness_score > mean(happiness_score)))
  )
```

# Using ifelse when creating a variable

We can also create a dummy variable with `ifelse` as follows:

```
whr15 %>%
  mutate(
    latin_america_car = ifelse(region == "Latin America and Caribbean", 1, 0)
  ) %>%
  arrange(-latin_america_car) %>%
  head(5)
```

```
## # A tibble: 5 x 13
##   country region happiness_rank happiness_score standard_error economy_gdp_per~
##   <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
## 1 Costa ~ Latin~             12            7.23         0.0445            0.956
## 2 Mexico  Latin~             14            7.19         0.0418            1.02
## 3 Brazil  Latin~             16            6.98         0.0408            0.981
## 4 Venezu~ Latin~             23            6.81         0.0648            1.04
## 5 Panama  Latin~             25            6.79         0.0491            1.06
## # ... with 7 more variables: family <dbl>, health_life_expectancy <dbl>,
## #   freedom <dbl>, trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>, latin_america_car <dbl>
```

The way we use this function is as: `ifelse(test, yes, no)`. We can also use the `case_when()` function.

## Some notes: mutate() vs transmute()

`mutate()` *vs* `transmute()`

Similar in nature but:

1. `mutate()` returns original and new columns (variables).
2. `transmute()` returns only the new columns (variables).

# Creating variables by groups

Let's imagine now that we want to create a variable at the region level -- recal `bys gen` in Stata. In R, we can `group_by()` before we mutate. For example:

```
whr15 %>%
  arrange(country, region, happiness_score) %>%
  group_by(region) %>%
  mutate(
    mean_hap = mean(happiness_score)
  ) %>%
  select(country:happiness_score, mean_hap) %>%
  head(5)
```

```
## # A tibble: 5 x 5
## # Groups:   region [5]
##   country     region                      happiness_rank happiness_score mean_hap
##   <chr>       <chr>                                <dbl>           <dbl>    <dbl>
## 1 Afghanistan Southern Asia                          153            3.58     4.58
## 2 Albania     Central and Eastern Europe              95            4.96     5.33
## 3 Algeria     Middle East and Northern ~              68            5.60     5.41
## 4 Angola      Sub-Saharan Africa                     137            4.03     4.20
## 5 Argentina   Latin America and Caribbe~              30            6.57     6.14
```

## Creating multiple variables at the same type

With the new version of `dplyr`, we now can create multiple variables in an easier way. So, let's imagine that we want to estimate the mean value for the variables: white, black, black_free, black_slaves.

Across Output

```
vars ← c("happiness_score", "health_life_expectancy", "trust_government_corruption")

whr15 %>%
  group_by(region) %>%
  summarize(
    across(all_of(vars), mean)
  )
```

## Creating variables

Before we merge our dataframes, we should add a year variable to identify each period:

```
whr15 <- whr15 %>%
  mutate(
    year = 2015
  )

whr16 <- whr16 %>%
  mutate(
    year = 2016
  )

whr17 <- whr17 %>%
  mutate(
    year = 2017
  )
```

## Appending and merging data sets

# Appending and merging data sets

Let's create a panel with the three dataframes. We can use the `bind_rows` function:

```
bind_rows(whr15, whr16, whr17)
```

```
## # A tibble: 470 x 17
##    country region happiness_rank happiness_score standard_error economy_gdp_per~
##    <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
##  1 Switze~ Weste~              1            7.59         0.0341             1.40
##  2 Iceland Weste~              2            7.56         0.0488             1.30
##  3 Denmark Weste~              3            7.53         0.0333             1.33
##  4 Norway  Weste~              4            7.52         0.0388             1.46
##  5 Canada  North~              5            7.43         0.0355             1.33
##  6 Finland Weste~              6            7.41         0.0314             1.29
##  7 Nether~ Weste~              7            7.38         0.0280             1.33
##  8 Sweden  Weste~              8            7.36         0.0316             1.33
##  9 New Ze~ Austr~              9            7.29         0.0337             1.25
## 10 Austra~ Austr~             10            7.28         0.0408             1.33
## # … with 460 more rows, and 11 more variables: family <dbl>,
## #   health_life_expectancy <dbl>, freedom <dbl>,
## #   trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>, year <dbl>, lower_confidence_interval <dbl>,
## #   upper_confidence_interval <dbl>, whisker_high <dbl>, whisker_low <dbl>
```

# Appending and merging data sets

Let's create a panel with the three dataframes. We can use the `bind_rows` function:

```
bind_rows(whr15, whr16, whr17)
```

```
## # A tibble: 470 x 17
##    country region happiness_rank happiness_score standard_error economy_gdp_per~
##    <chr>   <chr>           <dbl>           <dbl>          <dbl>            <dbl>
##  1 Switze~ Weste~              1            7.59         0.0341             1.40
##  2 Iceland Weste~              2            7.56         0.0488             1.30
##  3 Denmark Weste~              3            7.53         0.0333             1.33
##  4 Norway  Weste~              4            7.52         0.0388             1.46
##  5 Canada  North~              5            7.43         0.0355             1.33
##  6 Finland Weste~              6            7.41         0.0314             1.29
##  7 Nether~ Weste~              7            7.38         0.0280             1.33
##  8 Sweden  Weste~              8            7.36         0.0316             1.33
##  9 New Ze~ Austr~              9            7.29         0.0337             1.25
## 10 Austra~ Austr~             10            7.28         0.0408             1.33
## # … with 460 more rows, and 11 more variables: family <dbl>,
## #   health_life_expectancy <dbl>, freedom <dbl>,
## #   trust_government_corruption <dbl>, generosity <dbl>,
## #   dystopia_residual <dbl>, year <dbl>, lower_confidence_interval <dbl>,
## #   upper_confidence_interval <dbl>, whisker_high <dbl>, whisker_low <dbl>
```

# Appending and merging data sets

To be honest, the most important variable that we will need to include in the 2017 dataset is the region variable. We can do the following:

- Create a region vector from the `whr2015` dataframe.

```
regions ← whr15 %>%
  select(country, region)
```

- Now, we join the `regions` dataframe with the `whr17` dataframe.

```
whr17 ← whr17 %>%
  left_join(regions) %>%
  select(country, region, everything())
```

```
## Joining, by = "country"
```

## Appending and merging data sets

But unfortunately, some countries were not in the whr15 data.

```
whr17 %>%
  filter(is.na(region))
```

```
## # A tibble: 6 x 14
##   country region happiness_rank happiness_score whisker_high whisker_low
##   <chr>   <chr>           <dbl>           <dbl>        <dbl>       <dbl>
## 1 Taiwan~ <NA>               33            6.42         6.49        6.35
## 2 Belize  <NA>               50            5.96         6.20        5.71
## 3 Hong K~ <NA>               71            5.47         5.55        5.39
## 4 Somalia <NA>               93            5.15         5.24        5.06
## 5 Namibia <NA>              111            4.57         4.77        4.38
## 6 South ~ <NA>              147            3.59         3.73        3.46
## # ... with 8 more variables: economy_gdp_per_capita <dbl>, family <dbl>,
## #   health_life_expectancy <dbl>, freedom <dbl>, generosity <dbl>,
## #   trust_government_corruption <dbl>, dystopia_residual <dbl>, year <dbl>
```

## Appending and merging data sets

Let's fix these six countries.

- Taiwan and Hong Kong have different names. Let's use `case_when`:

```
whr17 <- whr17 %>%
  mutate(
    country = case_when(country == "Hong Kong S.A.R., China" ~ "Hong Kong",
                        country == "Taiwan Province of China" ~ "Taiwan",
                        TRUE ~ country)
  )
```

- Now, let's joing again the regions dataframe.

```
whr17 <- whr17 %>%
  select(-region) %>%
  left_join(regions) %>%
  select(country, region, everything())
```

```
## Joining, by = "country"
```

## Appending and merging data sets

Which countries still don't have region information?

```
whr17 %>%
  filter(is.na(region))
```

```
## # A tibble: 4 x 14
##   country region happiness_rank happiness_score whisker_high whisker_low
##   <chr>   <chr>           <dbl>           <dbl>        <dbl>       <dbl>
## 1 Belize  <NA>               50            5.96         6.20        5.71
## 2 Somalia <NA>               93            5.15         5.24        5.06
## 3 Namibia <NA>              111            4.57         4.77        4.38
## 4 South ~ <NA>              147            3.59         3.73        3.46
## # … with 8 more variables: economy_gdp_per_capita <dbl>, family <dbl>,
## #   health_life_expectancy <dbl>, freedom <dbl>, generosity <dbl>,
## #   trust_government_corruption <dbl>, dystopia_residual <dbl>, year <dbl>
```

## Appending and merging data sets

We can get their info from the whr16 dataset as follows

```r
whr17 <- whr17 %>%
  left_join(
    whr16 %>%
      select(country, new_region = region),
    by = "country"
  ) %>%
  mutate(
    region = ifelse(is.na(region), as.character(new_region), region)
  ) %>%
  select(-new_region)
```

# Appending and merging data sets

Any other country that still don't have region information?

```
whr17 %>%
  filter(is.na(region))
```

```
## # A tibble: 0 x 14
## # ... with 14 variables: country <chr>, region <chr>, happiness_rank <dbl>,
## #   happiness_score <dbl>, whisker_high <dbl>, whisker_low <dbl>,
## #   economy_gdp_per_capita <dbl>, family <dbl>, health_life_expectancy <dbl>,
## #   freedom <dbl>, generosity <dbl>, trust_government_corruption <dbl>,
## #   dystopia_residual <dbl>, year <dbl>
```

## Appending and merging data sets

Finally, let's keep those relevant variables first and bind those baby rows.

```
keepvars ← c("country", "region", "year", "happiness_rank",
              "happiness_score", "economy_gdp_per_capita",
              "health_life_expectancy", "freedom")

whr15 ← select(whr15, all_of(keepvars))
whr16 ← select(whr16, all_of(keepvars))
whr17 ← select(whr17, all_of(keepvars))

whr_panel ← rbind(whr15, whr16, whr17)    # or bind_rows
```

## Saving a dataset

## Saving a dataset

To save a dataset we can use the `write_csv` function from the tidyverse, or `write.csv` from base R.

```r
# Save the whr data set

write.csv(whr_panel,
          file.path(finalData,"whr_panel.csv"),
          row.names = FALSE)
```

## Saving a dataset

To save a dataset we can use the `write_csv` function from the tidyverse, or `write.csv` from base R.

```
# Save the whr data set

write.csv(whr_panel,
          file.path(finalData,"whr_panel.csv"),
          row.names = FALSE)
```

- The problem with CSVs is that they cannot differentiate between strings and factors
- They also don't save factor orders
- Data attributes (which are beyong the scope of this training, but also useful to document data sets) are also lost in csv data

## Saving a dataset

The R equivalent of a `.dta` file is a `.Rds` file. It can be saved and loaded using the following commands:

- `saveRDS(object, file = "")`: Writes a single R object to a file.
- `readRDS(file)`: Load a single R object from a file.

```r
# Save the data set

saveRDS(whr_panel, file = file.path(finalData, "whr_panel.Rds"))
```

## Reshaping a dataset

## Reshaping a dataset

Finally, let's try to reshape our dataset using the tidyverse functions. No more `reshape` from Stata. We can use `pivot_wider` or `pivot_longer`

Long to Wide    Wide to Long

```
whr_panel %>%
  select(country, region, year, happiness_score) %>%
  pivot_wider(
    names_from = year,
    values_from = happiness_score
  ) %>%
  head(5)
```

```
## # A tibble: 5 x 5
##   country     region          `2015` `2016` `2017`
##   <chr>       <chr>            <dbl>  <dbl>  <dbl>
## 1 Switzerland Western Europe    7.59   7.51   7.49
## 2 Iceland     Western Europe    7.56   7.50   7.50
## 3 Denmark     Western Europe    7.53   7.53   7.52
## 4 Norway      Western Europe    7.52   7.50   7.54
## 5 Canada      North America     7.43   7.40   7.32
```

Thank you~~