Introduction to the tidyverse

David García Callejas

The `tidyverse` is not an R package - it is, rather, as set of tightly integrated packages via similar syntax, a unified concept of data management, and consistency in argument types and function outputs. The main premise of the packages that make up the `tidyverse` is to facilitate data analysis through concatenating operations on datasets. Many `tidyverse` functions are useful by themselves, of course, but we will see how chaining operations together provides a different way of looking at data analysis. This chaining relies in the use of the pipe operator, `%>%`, which redirects the output of the function (or data) to its left as the input of the function to its right.

```r
# load the package bundle
library(tidyverse)

# read the earthquake dataset
eq <- read.csv2(file = "../data/earthquakes.csv")

# these two notations are equivalent
summary(eq)
eq %>% summary()
```

With this toolset, we can concatenate as many operations as we need

```r
eq %>%
  drop_na() %>%
  nrow()

eq %>%
  filter(!is.na(Dep)) %>%
  summary()
```

This syntax can be very convenient to apply sequential operations of filtering, ordenation, grouping, etc. As a first example, we will use the `iris` dataset to gather some species-level information.

```r
avg.values <- iris %>%
  group_by(Species) %>%
  summarise(mean.sepal.length = mean(Sepal.Length),
            mean.petal.length = mean(Petal.Length))
```

We may also create new columns in the original dataset

```r
iris2 <- iris %>%
  mutate(new_col = Sepal.Length * Sepal.Width)
```

Of course, all these operations can be coded in the style of base R. However, the `tidyverse` syntax increases flexibility, compactness, and for some of us, it also makes the core more readable.

Let's now see a full-fledged example, in which we want to obtain the average magnitude and depth of earthquakes in each country. For doing so, we need to assign the earthquake observations to country names. So, let's define a function that returns a country name given a pair of latitude and longitude numbers.

https://stackoverflow.com/questions/14334970/convert-latitude-and-longitude-coordinates-to-country-name-in-r

```
# we need these two packages
library(sp)
library(rworldmap)

# points is a dataframe with values longitude (column 1) y latitude (c2)
coords2country <- function(points){
  countriesSP <- rworldmap::getMap(resolution='low')

  #setting CRS directly to that from rworldmap
  pointsSP = sp::SpatialPoints(points,
                               proj4string=sp::CRS(sp::proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = sp::over(pointsSP, countriesSP)

  # return the ADMIN names of each country
  indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  #indices$continent   # returns the continent (6 continent model)
  #indices$REGION   # returns the continent (7 continent model)
}
```

We can also create an equivalent function to return the continent of a given point

```
coords2continent <- function(points){
  countriesSP <- rworldmap::getMap(resolution='low')

  #setting CRS directly to that from rworldmap
  pointsSP = sp::SpatialPoints(points,
                               proj4string=sp::CRS(sp::proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = sp::over(pointsSP, countriesSP)

  # return the ADMIN names of each country
  # indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  indices$continent   # returns the continent (6 continent model)
  #indices$REGION   # returns the continent (7 continent model)
}
```

Now let's apply these functions

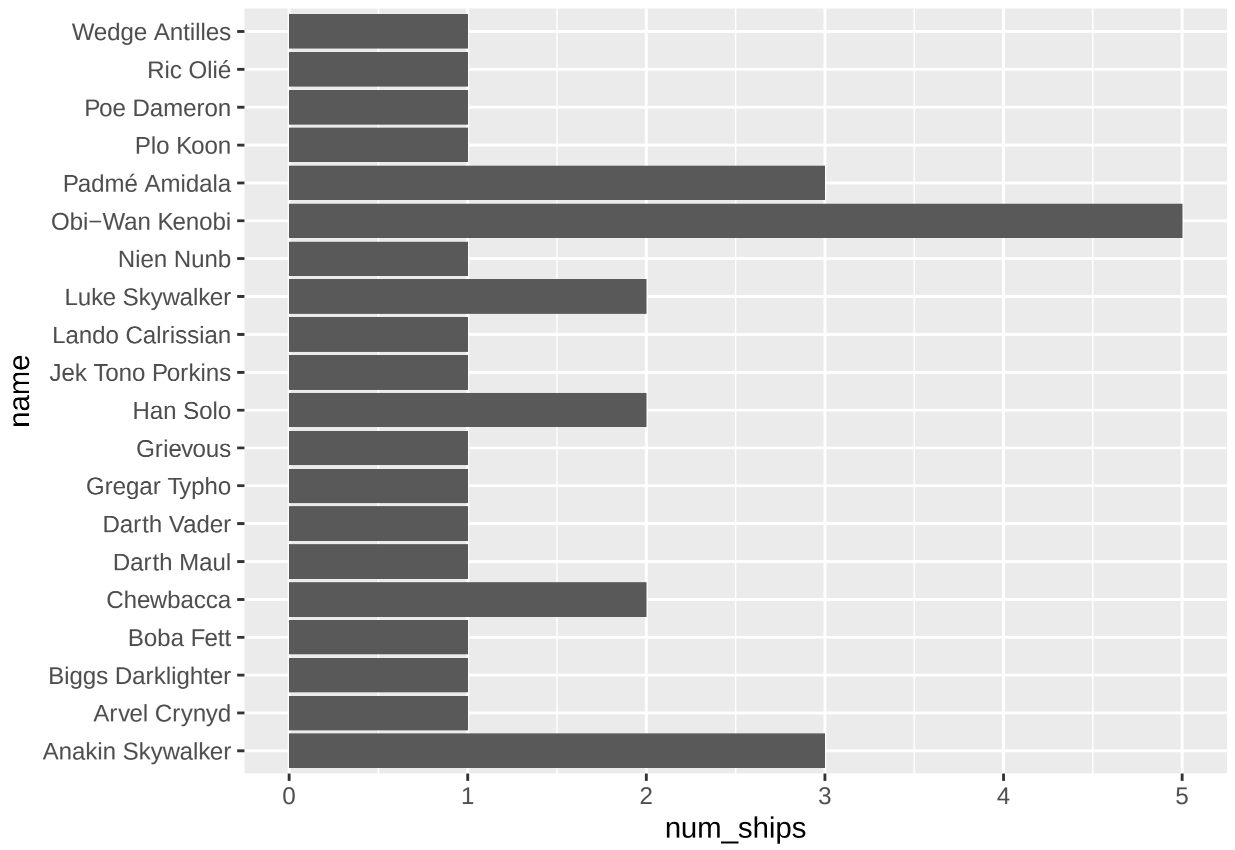
```
eq.country <- eq %>%
  mutate(country = coords2country(data.frame(Lon,Lat)),
         continent = coords2continent(data.frame(Lon,Lat)))

table(eq.country$continent)
table(eq.country$country)
```

and generate a dataframe with the relevant columns

```
magnitude.eurasia <- eq.country %>%
  filter(continent == "Eurasia") %>%
  summarise(mean.magnitude = mean(M),
            sd.magnitude = sd(M),
            min.year = min(Year))

# this can be automated
magnitude.continents <- eq.country %>%
  group_by(continent )%>%
  summarise(mean.magnitude = mean(M),
            sd.magnitude = sd(M),
            num.eq = n(),
            min.year = min(Year))
```

Other important functions:

```
# the join family

sw1 <- read.csv2("../data/starwars_info_characters.csv")
sw2 <- read.csv2("../data/starwars_spaceships.csv")

sw.full.1 <- left_join(sw1,sw2)
sw.full.2 <- left_join(sw2,sw1)

# pivot - longer and wider

sw.ships <- sw2 %>%
  pivot_longer(X.wing:H.type.Nubian.yacht,
               names_to = "spaceship_name",
               values_to = "can_pilot")

# now the data is easier to work with, e.g. in ggplot
sw.ships %>%
  group_by(name) %>%
  summarise(num_ships = sum(can_pilot)) %>%
  filter(num_ships > 0) %>%
  ggplot(aes(x = name, y = num_ships)) +
  geom_col() +
  coord_flip()
```



```
sw.char.wide <- sw1 %>%
  select(name,species_group) %>%
  mutate(tt = TRUE) %>%
  pivot_wider(names_from = species_group,
              values_from = tt,
              values_fill = NA)

# across
sw.char.wide.clean <- sw.char.wide %>%
  mutate(across(-name, ~ replace_na(.x,FALSE)))
```