Lecture 5: Data Wrangling: Dates and Times

Economía Laboral

Junghanss, Juan Cruz

Universidad del CEMA

2nd Semester 2023

Contents

Today's lecture content:

- More on Data Wrangling: Dates and Times.
- Labor supply problem algorithm.

Motivation: Dates and Times

It might seem trivial for you to study dates and times, but believe me, they're a big headache in your first programming experiences so it's better to know them well.

Some important things for you to learn are related with these questions:

- What's the difference between dates, times and datetimes?
- What happens if we have data with different timehours?
- Are we sure that every year has 365 days? And every day has 24 hours?
- How are arithmetic operations with time represented?

Dates and Times

Some definitions you need to have always very clear:

- A *date* is only a date format that can have day/month/year as components (Example: 20/12 or "20-Dec")
- A *time* is the hourtime within a day that can have hour/minutes/seconds as components (Example: 11:42am)
- A *date-time* is a date plus a time. (Example: 27/8/1998 22:17)

Always use the simplest possible data type that fits your model. That means if you can use date instead of date-time, you should do it!

Dates and Times with lubridate

The package related to dates and times is called "**lubridate**", which also has to be explicitly loaded since it's not part of the core tidyverse.

The most important functions are:

- today() to get the actual date or now() to get the actual date-time.
- ymd() / mdy() / dmy() to parse strings as dates in a specific format, i.e. year-month-day, for example.
- make_date() or make_datetime() to create dates from columns in tibbles.
- as_datetime() to switch from date to datetime.
- as_date() to switch from datetime to date.

Dates and Times with lubridate

- To extract individual components of the date or datetime format:
  - year(), month()
  - mday() (day of the month), yday() (day of the year), wday() (day of the week)
  - hour(), minute(), sec()
- round_date() or floor_date() to round the date to a nearby unit of time.
- update() to set the components of the date/time.

Time with hms

We're only going to focus on dates and date-times as R doesn't have a native class for storing **times**. If you need one, you can use the **hms** package.

Creating date/time variables

You can create date/time variables to work with:

1. From scratch: for example getting the current date/time with today() or now()
2. **From other variables**: mostly you will receive a dataset's variable with "dates" that are **strings** or **characters**.
3. **From individual components**: Instead of a single string, sometimes you'll have the individual components of the date-time spread across multiple columns.

Creating date/time variables from string variables

When you have a string, you can use

1. **readr parsers**
2. **lubridate parsers**

You pick between**3 (three) parsers** of readr package depending on whether you want a date, a date-time or a time. When called without any additional arguments:

- parse_datetime() expects an ISO8601 date-time.
- parse_date() expects a four-digit year, a - or /, the month, a - or /, then the day
- parse_time() expects the hour, :, minutes, optionally : and seconds, and an optional a.m./p.m. specifier (you have to load hms package).

Creating date/time variables from string variables

One of the best hacks is to get in touch with the formats documentation. If the defaults don't work for your data you can supply your own date-time format, built up of the following pieces:

- Year
  - %Y (4 digits).
  - %y (2 digits; 00-69 is 2000-2069, 70-99 is 1970-1999).
- Month
  - %m (2 digits).
  - %b (abbreviated name, like Jan).
  - %B (full name, January).
- Day
  - %d (2 digits).
  - %e (optional leading space).
- Nondigits
  - %. (skips one nondigit character).
  - %* (skips any number of nondigits).

Creating date/time variables from string variables

The best way to figure out the correct format is to create a few examples in a character vector, and test with one of the parsing functions. For example:

```
parse_date("06/09/22", "%m/%d/%y")
#> [1] "2022-09-06"

parse_date("06/09/22", "%d/%m/%y")
#> [1] "2022-06-09"

parse_date("06/09/22", "%y/%m/%d")
#> [1] "2006-09-22"
```

Creating date/time variables from string variables

If you're using %b or %B with non-English month names, you'll need to set the lang argument to locale(). See the list of built-in languages in date_names_langs()

```
parse_date(
"6 Septiembre 2022", "%d %B %Y",
locale = locale("es")
)
#> [1] "2022-06-09"
```

Creating date/time variables from string variables

Alternatively, you can use other "parsers" from lubridate (instead of readr). These ones are more user-friendly, since they figure out the complete format automatically. You just have to specify the order of year, day, month and arrange the letters in that order:

```
ymd("2022-09-06")
#> [1] "2022-09-06"

mdy("September 6, 2022")
#> [1] "2022-09-06"

dmy("06-Sep-2022")
#> [1] "2022-09-06"
```

Creating date/time variables from individual components

Sometimes you'll have the individual components of the date-time spread across multiple columns. In such case you can use function make_datetime() with the date/time variables as arguments:

```
dataset %>%
select(year, month, day, hour, minute) %>%
mutate(
date_time = make_datetime(year, month, day, hour, minute)
)
```

Switch between date and datetime types

You may want to switch between a date-time and a date. That's he job of as_datetime() and as_date():

```
as_datetime(today())
#> [1] "2016-10-10 UTC"

as_date(now())
#> [1] "2016-10-10"
```

Pulling out components of dates

You can pull out individual parts of the date with the accessor functions year(), month(), mday() (day of the month), yday() (day of the year), wday() (day of the week), hour(), minute(), and second()

Pulling out components of dates

Problema: supongan que en un reporte se nos pide presentar la distribución de los días de nacimiento de los individuos de la EPH expresados como **días de la semana** (Lunes, Martes, etc.). Por ejemplo: hay 10504 nacimientos reportados el día lunes, 9785 el día martes, etc. Los datos relativamente útiles para esto que tenemos son las variables CH05 y CH06 (fecha de nacimiento y edad respectivamente).

¿Cómo podemos resolver esta tarea?

Time Spans

Consider the case where you want to calculate the (time) difference between two dates. In such scenario, the class *difftime* appears as a result.

Formats are:

- *Duration*: it represents an exact number of seconds.
- *Period*: a period represents human units like weeks and months.
- *Interval*: an interval represents a starting and ending point.

Which one choose? Again, pick the simplest data structure that solves your problem!

Arithmetic Operations with Time

Here are the arithmetic operations allowed in each time class summarized:

	date				date time				duration				period				interval				number			
date	-								-	+			-	+							-	+		
date time					-				-	+			-	+							-	+		
duration	-	+			-	+			-	+		/									-	+	×	/
period	-	+			-	+							-	+							-	+	×	/
interval												/				/								
number	-	+			-	+			-	+	×		-	+	×		-	+	×		-	+	×	/

Duration class

A difftime class object records a time span of seconds, minutes, hours, days, or weeks. This ambiguity can make difftimes a little painful to work with, so lubridate provides an alternative that always uses seconds—the **duration**:

```
dseconds(15)
#> [1] "15s"

dminutes(10)
#> [1] "600s (10 minutes)"

dweeks(3)
#> [1] "1814400s (3 weeks)"
```

Duration class

Durations always record the time span in seconds. Larger units are created by converting minutes, hours, days, weeks, and years to seconds at the standard rate (60 seconds in a minute, 60 minutes in an hour, 24 hours in day, 7 days in a week, 365 days in a year).

```
dyears(1) + dweeks(12) + dhours(15)
#> [1] "38847600s (1.23 years)"
```

```
tomorrow <- today() + ddays(1)
```

```
last_year <- today() - dyears(1)
```

Period class

Periods are time spans but don't have a fixed length in seconds; instead they work with "human" times, like days and months. That allows them to work in a more intuitive way:

```
minutes(10)
#> [1] "10M 0S"
```

```
weeks(3)
#> [1] "21d 0H 0M 0S"
```

```
ymd("2022-01-01") + dyears(1)
#> [1] "2022-12-31"
```

Interval class

It's obvious what dyears(1) / ddays(365) should return: one, because durations are always represented by a number of seconds, and a duration of a year is defined as 365 days' worth of seconds.

What should years(1) / days(1) return? Well, if the year was 2015 it should return 365, but if it was 2016, it should return 366! There's not quite enough information for lubridate to give a single clear answer. What it does instead is give an estimate, with a warning:

```
years(1) / days(1)
#> estimate only: convert to intervals for accuracy
#> [1] 365
```

If you want a more accurate measurement, you'll have to use an *interval*.

Labor Supply Problem

Para entender el mecanismo resolutivo del problema de la oferta laboral, veamos cómo se plantea su algoritmo de resolución en pseudocodigo:

Algorithm 1 Algorithm for Labor Supply Problem

Require: Utility function $U(l, c)$ with leisure l and consumption c
Require: Budget constraint equation $w \cdot h + A = c$
Require: Time constraint equation $T = h + l$

1: compute Lagrangian from $\max U(l, c)$ s.t. to $wT + A - wl - c = 0$
2: compute FOCs $\mathcal{L}_l$; $\mathcal{L}_c$; $\mathcal{L}_\lambda$
3: *leisure demand* $\leftarrow$ Set FOCs equations equal and solve for $l(w, A, T)$
4: *consumption demand* $\leftarrow$ Substitute *leisure demand* $l(w, A, T)$ in $TMS = w$ and solve for $c(w, T, A)$
5: *labor supply* $\leftarrow$ Substitute *leisure demand* $l(w, A, T)$ in time supply $T = h + l$ and solve for $h(w, A, T)$

Labor Supply Problem

Para despejar la elasticidad de la oferta laboral:

$$\eta = \frac{\Delta h/h}{\Delta w/w} = \frac{\Delta h}{\Delta w} \cdot \frac{w}{h}$$

Algorithm 2 Algorithm for Labor Supply Problem

Require: Labor supply function $h(w, A, T)$

1: $\eta \leftarrow$ compute first derivative h_w with respect to wage w

2: multiply h_w with $\frac{w}{h(w,A,T)}$

Labor Supply Problem

Para despejar el salario de reserva:

$$h^*(w, T, A) = 0 \Longrightarrow w_r$$

Algorithm 3 Algorithm for Labor Supply Problem

Require: Labor supply function $h(w, A, T)$

1: $w_r \leftarrow$ set labor supply $h(w, A, T) = 0$

2: solve for w