Barcelona Summer School of Demography

Module 1. Introduction to R

2. Tidy Pipelines

Tim Riffe
riffe@demogr.mpg.de

2 July 2019

Contents

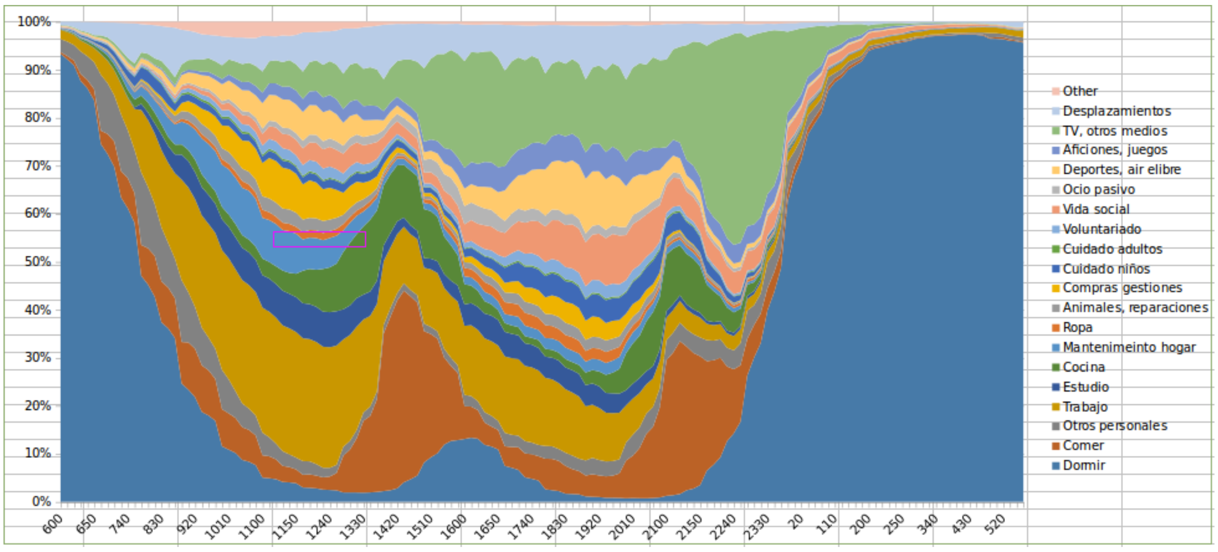
1 Data processing verbalized

We know we'll need tidyverse functions today, so let's just get this loaded:

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------ tidyverse 1.2.1 --

## v ggplot2 3.1.0       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- Conflicts ---------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

So coincidentally a friend Joan G. asked me how to make this plot in `R` the other day:



It looks like we have time of day in x, population percentile in y, and filled areas indicate what fraction of people are doing different activities. The table underlying this was derived from survey microdata.

Remember we can't really go this high with qualitative colors very easy, and we may or may not avoid this issue. I've asked him for the raw data that came before the final table that went into this plot so that we can see those steps too in `R` (Thanks JG!).

This by the way turns out to be a super advanced example. We will build it up incrementally together, and it features many of the `dplyr` heroes:

1. `gather()` make a wide range of columns tidy
2. `mutate()` make new columns using other columns, no loss of rows
3. `select()` selects columns
4. `filter()` selects rows (subsets)
5. `group_by()` allows operations on subgroups
6. `ungroup()` removes the former
7. `summarize()` aggregates over rows. Usually reduces nr of rows

and some useful but less important functions that do specific things:

7. `separate()` splits a column into two
8. `remove_na()` replaces `NA`s with some value
9. `n()` counts cases, usually in grouped data

There are a couple seemingly silly but important things to note about this data processing pipline: i) most of the operations are verbs, ii) it's all a single flow, strung together with `%>%` which you can read as "and then do this". You can find a nice overview of `dplyr` functions here: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

First let's read in Joan's data. It's in an SPSS format, so we'll use the `haven` package, which knows what to do with it. Let's take a look at this data.

```r
library(haven)
JG <- read_spss("caseid_aggr.sav")
```

The columns are similar to factors: but really they mimick the value-label style of SPSS. The actual values are just integers, and each integer stands for an activity. When we reshape to a tidy style we'll lose that information, just for the sake of being able to label activities in the plot later, we save in an object them now.

```
activities <- attr(JG$time_1000_max,"labels")
```

And now begins the pipeline:

1. We pull the 1440 minute-stamped columns into a single column `activity`, and the column labels are pulled into the column `Time`. We give the range of columns to gather with `time_1_max:time_1440_max`.

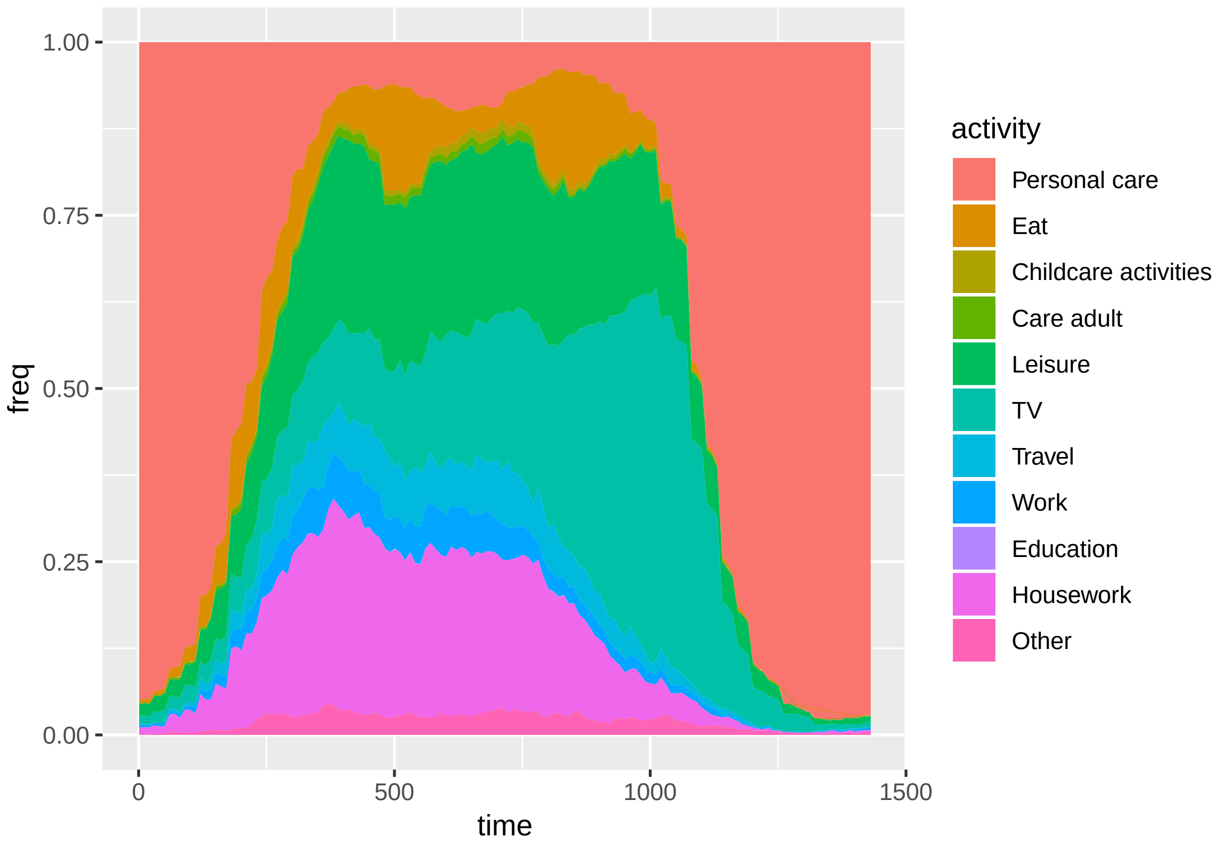
```
JG <- JG %>%
    sample_n(1000) %>%          # reduce size for testing purposes
    gather(key = "Time",        # new column for header names
           value = "activity", # new column for value collected
           time_1_max:time_1440_max) %>%
    separate(col = Time,
             into = c(NA,"time",NA),
             sep ="_",
             remove = TRUE,
             convert = TRUE) %>%
    select(CASEID, time, activity) %>% # only cols we need
    filter(time %% 10 == 1) %>%      # cut rows to every 10th
    group_by(time, activity) %>%
    summarise(n = n()) %>%
    mutate(freq = n / sum(n, na.rm = TRUE)) %>%
    ungroup() %>%
    mutate(activity = factor(activity,
                             levels = activities,
                             labels = names(activities))) %>%
    replace_na(list(freq = 0))

     # tip for largish data:
gc() # free up memory no longer needed
```

```
##           used (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  796598 42.6    3438902 183.7  4298628 229.6
## Vcells 1384052 10.6   57234115 436.7 57285892 437.1
```

Now we have the final dataset that we want to plot: it literally contains the fraction of time (`freq`) spent doing different activities (`activity`) in 10 minute steps through the day (`time`). Nothing less and nothing more.

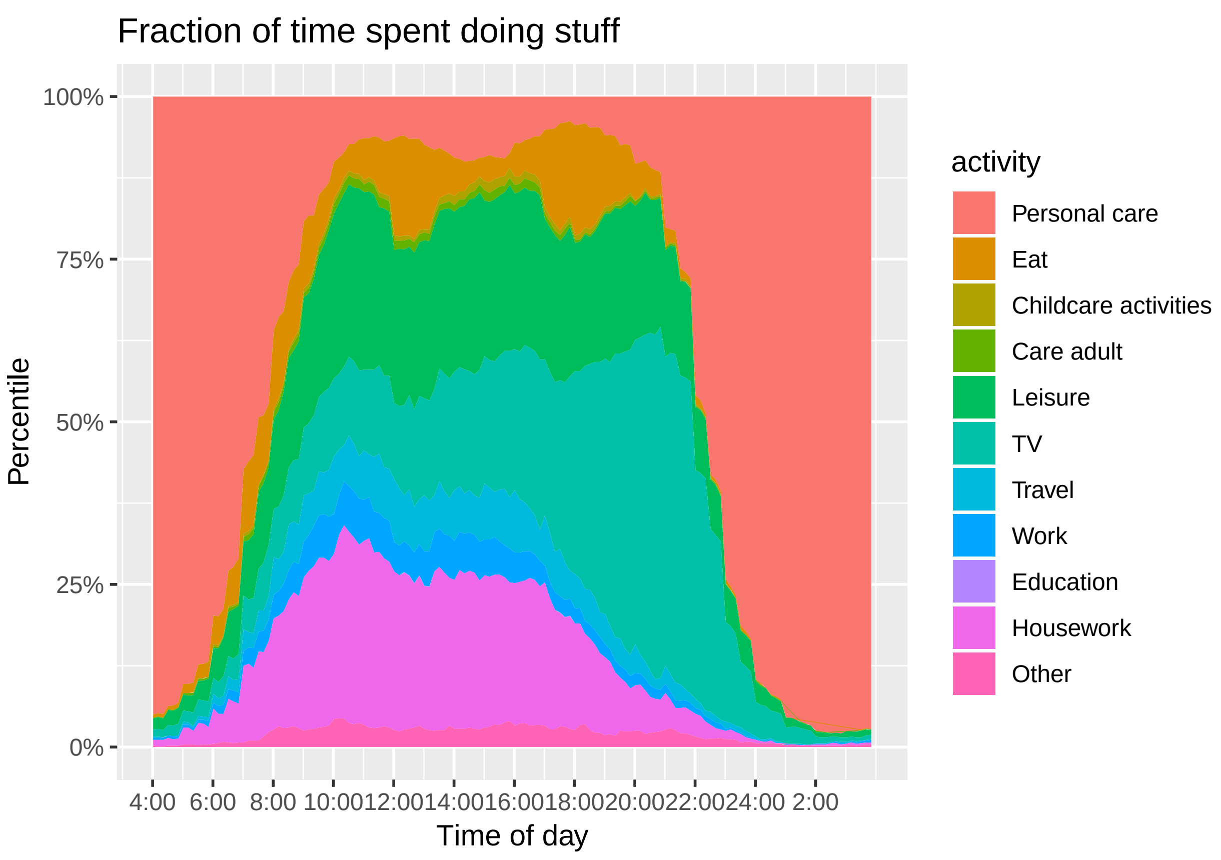
In plotting this we will introduce a new `geom`: `geom_area()` to make our filled areas. Here's a first rough stab at it. You see we have the basic structure, and everything from here on out is going to be plot detail management.

```
ggplot(JG, mapping = aes(x = time, y = freq, fill = activity)) +
    geom_area()
```



For the sake of a semi complete solution, here's a way to get more meaningful x and y breaks and labels.

```
# Figure out time labels...
# we have minutes since 4:00AM,
# so hour = (time - 1) / 60
mins    <- seq(0,1440,by=10)
hrs     <- seq(0, 1430, by = 120)
hrslabs <- paste(c(seq(4, 24, by = 2),2), "00", sep = ":")

ggplot(JG, mapping = aes(x = time, y = freq, fill = activity)) +
    geom_area() +
    scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
    scale_x_continuous(breaks = hrs,
                       labels = hrslabs) +
    labs(x = "Time of day",
         y = "Percentile",
         title = "Fraction of time spent doing stuff")
```



Joan G. told me that time is expressed in minutes since 4:00 AM, so that's my reference point. I'd like a label every two hours, so 120 minutes `seq(0, 1440, by = 120)` gives me a vector of hour breaks exrpressed in minutes. Now I produce the labels using similar tricks, `paste()` concatenates things together with an optional separator (`:`) in our case. We just need to remember that we have 24 hours of data, so at the end we add a break for 2:00 AM. These are then set using `scale_x_continuous()` (because `time` is continuous here). To get percentile labels on the y axis I used a function from the `scales` package (solution found on Stack Overflow, that's right).

2 Worked example number 2

Here is a file I downloaded from the Human Fertility Collection https://www.fertilitydata.org/cgi-bin/index.php, it's somewhat tidy already, but we'll still need to do things to it to .

```
HFC <- read_csv("HFC_ASFRstand_TOT.txt",
                na = ".")
```

```
## Parsed with column specification:
## cols(
##   Country = col_character(),
##   Region = col_logical(),
##   Urban = col_logical(),
##   Origin = col_logical(),
##   Year1 = col_double(),
##   Year2 = col_double(),
##   Age = col_double(),
##   AgeInt = col_double(),
##   AgeDef = col_character(),
##   Vitality = col_double(),
##   ASFR = col_double(),
##   CPFR = col_double(),
##   Collection = col_character(),
##   SourceType = col_character(),
##   RefCode = col_character(),
##   Note = col_logical(),
##   Split = col_double()
## )
```

```
## Warning: 20573 parsing failures.
##   row  col           expected actual                    file
## 46954 Note 1/0/T/F/TRUE/FALSE      3 'HFC_ASFRstand_TOT.txt'
## 46955 Note 1/0/T/F/TRUE/FALSE      3 'HFC_ASFRstand_TOT.txt'
## 46956 Note 1/0/T/F/TRUE/FALSE      3 'HFC_ASFRstand_TOT.txt'
## 46957 Note 1/0/T/F/TRUE/FALSE      3 'HFC_ASFRstand_TOT.txt'
## 46958 Note 1/0/T/F/TRUE/FALSE      3 'HFC_ASFRstand_TOT.txt'
## ..... .... .................. ...... .......................
## See problems(...) for more details.
```

```
#str(HFC)
```

This is a rather large and underexploited data source. We have age-specific fertility rates for many countries and years. There's one inconsistency in the age definition that probably only demographers care about, can explain if you want. Also, sometimes the same year and place has more than one estimate, so we'll want to be careful picking out all the potential grouping variables if using `group_by()`.

Let's use the `HFC` dataset to plot some different things.

3 Exercises

Exercise 1.1:

How about we get a glimpse of the full variety of fertility curves in there? Let's make a single line graph with all ASFR patterns plotted. To do this, you'll need to identify subsets, which is a major drawback with this dataset, so here's a tip for that:

```
HFC$YrInt  <- HFC$Year2 - HFC$Year1 + 1
HFC$sub_id <- group_indices(HFC,
                            Country,
                            Year1,
                            Region,
                            Urban,
                            Origin,
                            AgeDef,
                            Vitality,
                            Collection,
                            SourceType,
                            RefCode,
                            YrInt)
```

Now `sub_id` can be your grouping variable. Give it a try:

```
# ggplot(HFC ...) +
#   geom_ ...
```

It will be tricky to get this plot to work due to overplotting. Try reducing `alpha` to something small.

Exercise 1.2: Plot the time series of the total fertility rate (TFR) for 5 or so selected countries. You choose which (note they're not all there). If you're not sure about country codes used, you can check here: https://www.fertilitydata.org/cgi-bin/country_codes.php

Exercise 1.3: To be determined ad hoc