Tuesday Class Notes

Tim Riffe

July 2, 2019

Agenda

1. Understand what a `tidy` data processing pipline looks like.
2. Learn the main `tidy` data processing verbs. For example `group_by()` or `mutate()`.
3. Learn to read a `tidy` data processing pipeline.
4. Learn to string these pieces together into sentences using pipes. This is a pipe: `%>%`.

These two ways of getting the standard deviation are identical calculations. The first reads from the inside out, while the second reads from the top down.

```
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()
```

```
set.seed(1) # little tip in case u want the
            # same random numbers (internally)
a <- rnorm(10)

sqrt(mean((a - mean(a)) ^ 2))
```

```
## [1] 0.7405289
```

```
asd <-  a %>%
    mean() %>%
    '-'(a) %>% # operators are functions too!
    '^'(2) %>%
    mean() %>%
    sqrt()
```

Let's read in Joan G.'s data. This we spent all morning working through.

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

# This are the activity codes, redundantly
# kept as metadata about each time column
# in the original source.
activities <- attr(JG$time_1000_max, "label")
```

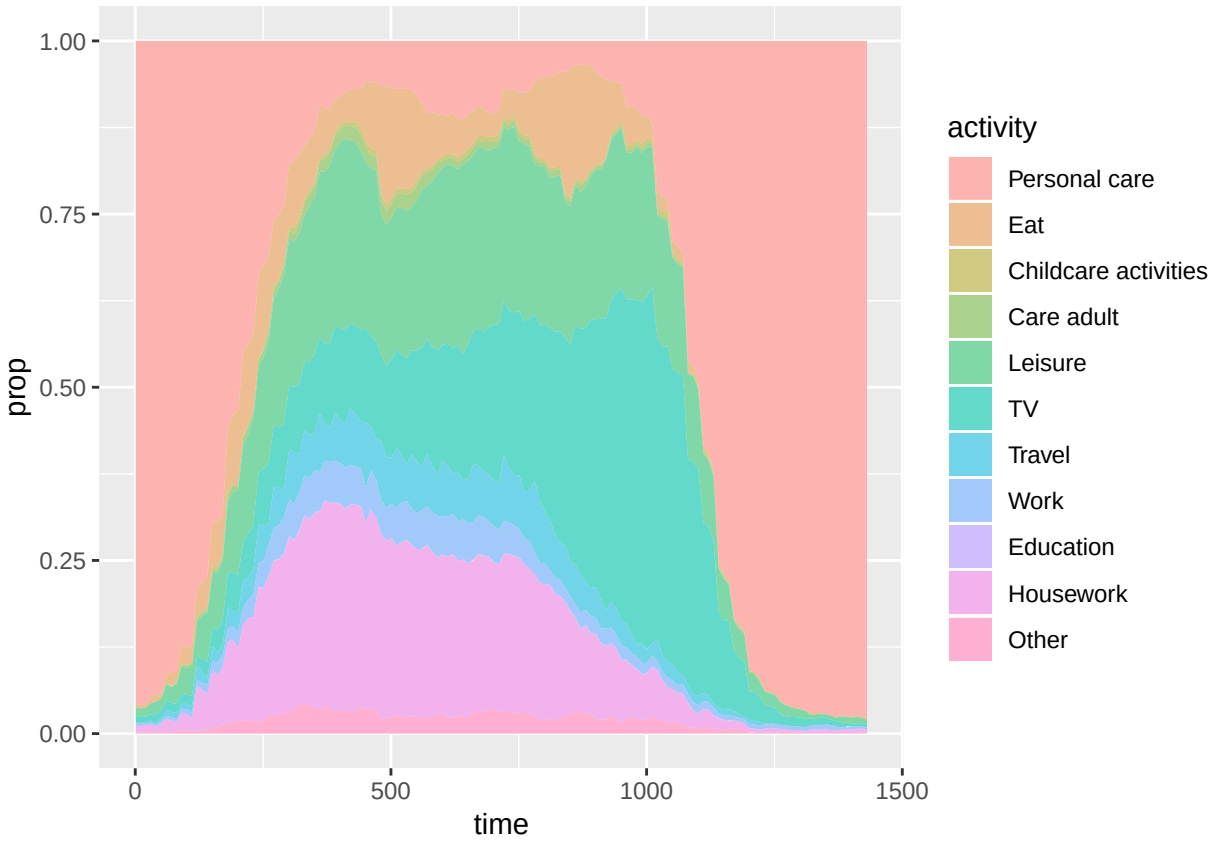
```
JG <- sample_n(JG, 1000)

# pipeline starts here!
JG <- JG %>%
    gather(key = time,
           value = activity,
           time_1_max:time_1440_max) %>%
# now split the concatenated column called time
# keeping the middle bit and converting it to integer
    separate(col = time,
             into = c(NA, "time", NA),
             sep = "_",
             remove = TRUE,
             convert = TRUE) %>%
    filter(time %% 10 == 1) %>%    # select every 10th minute
    group_by(time, activity) %>%
    summarize(n = n()) %>%         # get frequencies
    mutate(prop = n / sum(n)) %>%
    ungroup() %>%
    mutate(activity = factor(
        activity,
        levels = activities,
        labels = names(activities)
    ))
```

now visualize it using ggplot2

The original colors were way too saturated, so we tweaked them a little.

```
#head(JG)
ggplot(JG, mapping = aes(x = time,
                         y = prop,
                         fill = activity)) +
    geom_area() +
    scale_fill_discrete(c = 50, l = 80)
```



Mini example of modulo

This is an example of modulo logic in practice, which we used above. The modulo operator is like when you were a little kid right before learning about decimals or fractions when you divided numbers there was a so-called *remainder*.

```
A <- data.frame(id = c(1,1,1,1,2,2,2,2),
                time = c(1,2,3,4,1,2,3,4))
mutate(A,
       mod = time %% 2)
```

```
##   id time mod
## 1  1    1   1
## 2  1    2   0
## 3  1    3   1
## 4  1    4   0
## 5  2    1   1
## 6  2    2   0
## 7  2    3   1
## 8  2    4   0
```

This is being used in the time-use example above on a much larger scale.

Pipeline difference with traditional researcher coding practices

```
JG         <- read_spss("caseid_aggr.sav")
activities <- attr(JG$time_1000_max, "label")

JG <- sample_n(JG, 1000)

# pipeline starts here!
JG2 <- gather(JG,
              key = time,
               value = activity,
               time_1_max:time_1440_max)

JG3 <-  separate(JG2,
                 col = time,
                into = c(NA, "time", NA),
                sep = "_",
                remove = TRUE,
                convert = TRUE)

JG4 <-  filter(JG3, time %% 10 == 1)

JG5 <-  group_by(JG4, time, activity)

JG6 <-  summarize(JG5,n = n())

JG7 <-  mutate(JG6, prop = n / sum(n))

JG8 <-  ungroup(JG7)

JG9 <-  mutate(JG8, activity = factor(
        activity,
        levels = activities,
        labels = names(activities)
    ))

rm(JG1, JG2)
gc()
```

Now for a new worked example: HFC

```
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.
```

```
HFC <- HFC %>%
    mutate(YearInt = Year2 - Year1 + 1)
HFC$sub_id <- group_indices(HFC,
                            Country,
                            Year1,
                            AgeDef,
                            Vitality,
                            Collection,
                            SourceType,
                            RefCode,
                            YearInt)
```

Exercise: Plot ALLLLLL the fertility curves overlapped in a single plot. Tip: `geom_line()` will do the drawing.

More tips: 1. `x` is the same `x` used in the picture I drew on the board. 2. `y` is the same `y` used in the picture I drew on the board. 3. If you want each line to be draw independently then you need to use `group=` to tell it that (assuming we want them all drawn the same, which we do!). If you want the lines separated and with different colors (you don't!) then `color=sub_id` would also do the same feat.

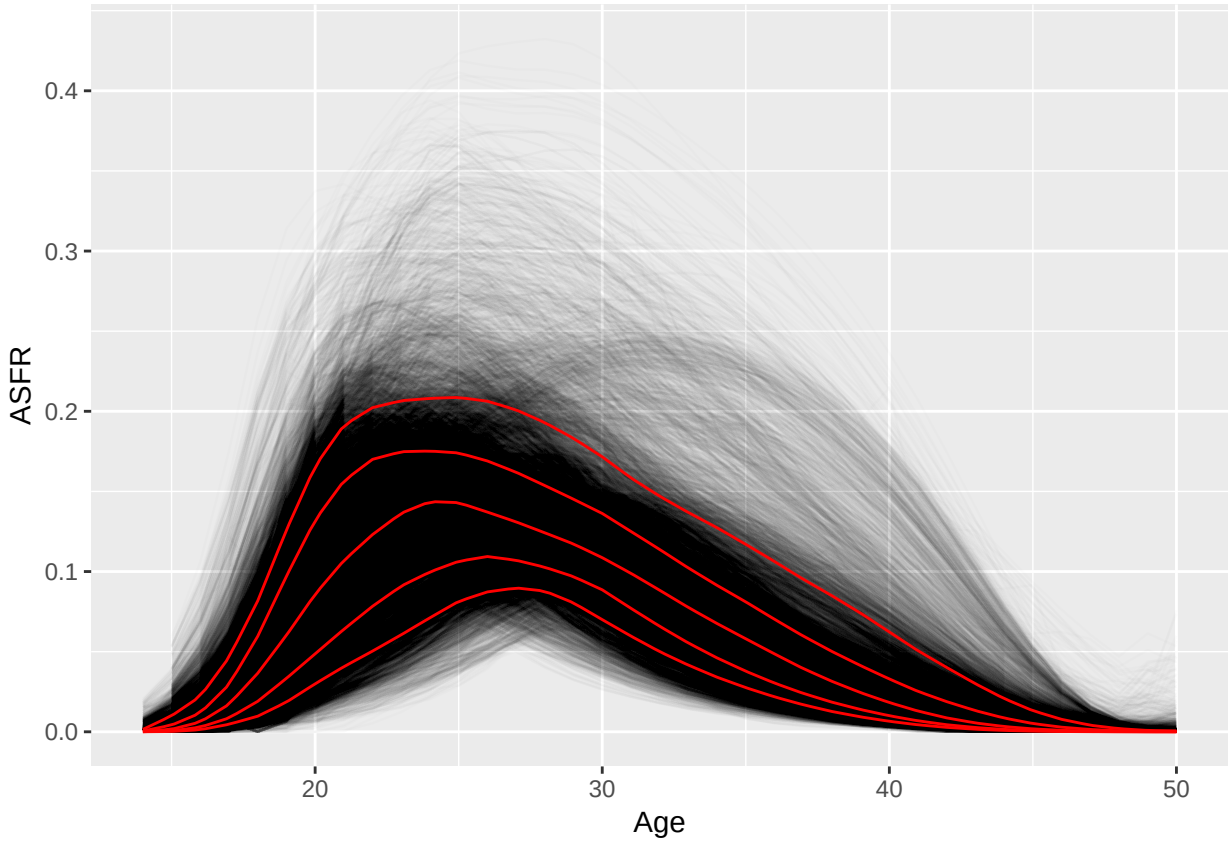
It was key here to remember to use the `group` mapping inside the `geom_line`, otherwise the quantile regression is done to each curve!

```
p <- ggplot(HFC, mapping = aes(x = Age,
                               y = ASFR))
p + geom_line(mapping=aes(group = sub_id), alpha = .01) +
    # enhance the overplotted area by finding quantiles
    geom_quantile(method = "rqss",
                  lambda = 1,
                  quantiles = c(.1,.25,.5,.75,.9),
                  color = "red")
```

```
## Warning: Removed 10893 rows containing non-finite values (stat_quantile).
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
## The following object is masked from 'package:base':
##
##     backsolve
## Smoothing formula not specified. Using: y ~ qss(x, lambda = 1)
## Warning: Removed 10893 rows containing missing values (geom_path).
```



Exercise

Calculate TFR and MAB for each subset.

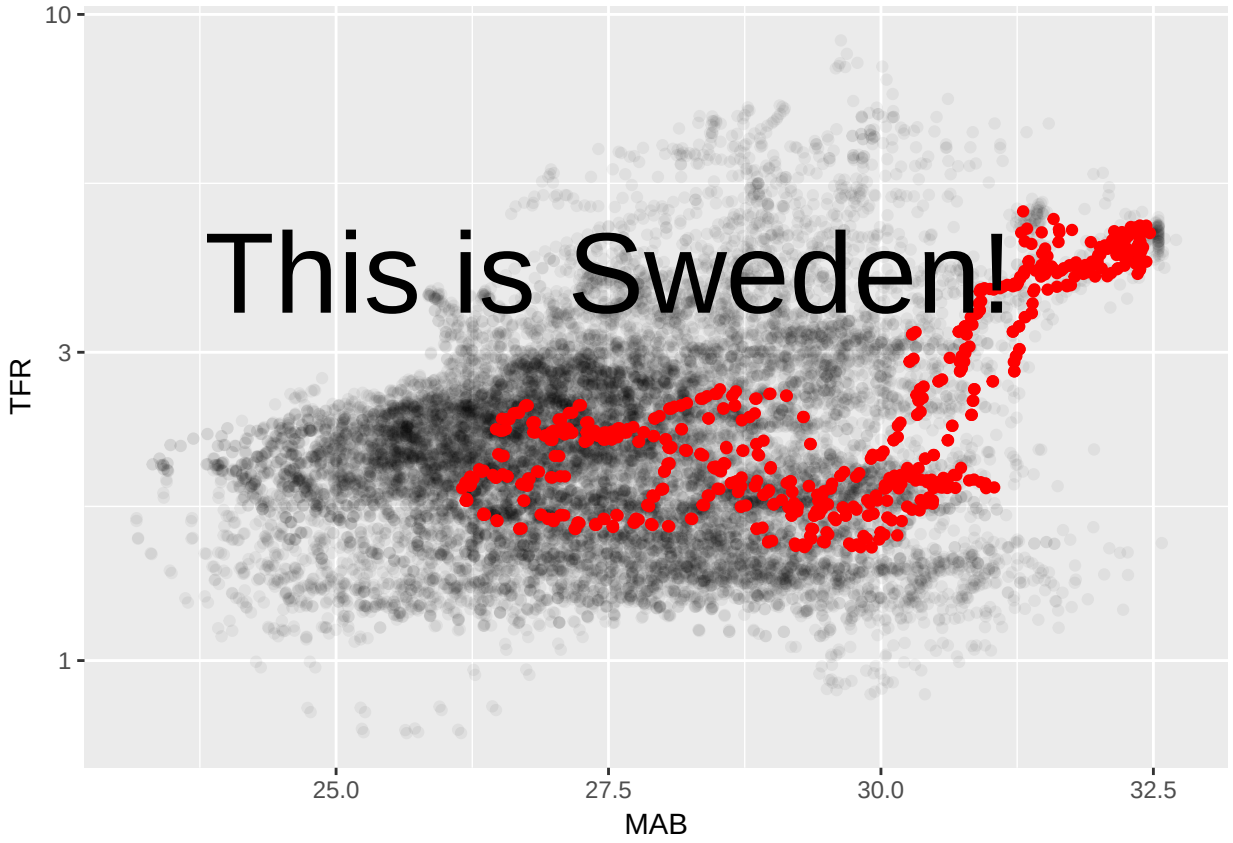
$$TFR = \sum ASFR$$

$$MAB = \frac{\sum ASFR * Age}{TFR}$$

```
TFRMAB <- HFC %>%
    group_by(sub_id) %>%
    summarize(TFR = sum(ASFR, na.rm = TRUE),
              MAB = sum(ASFR * Age, na.rm = TRUE) / TFR,
              Country = first(Country),
              Year = first(Year1))
```

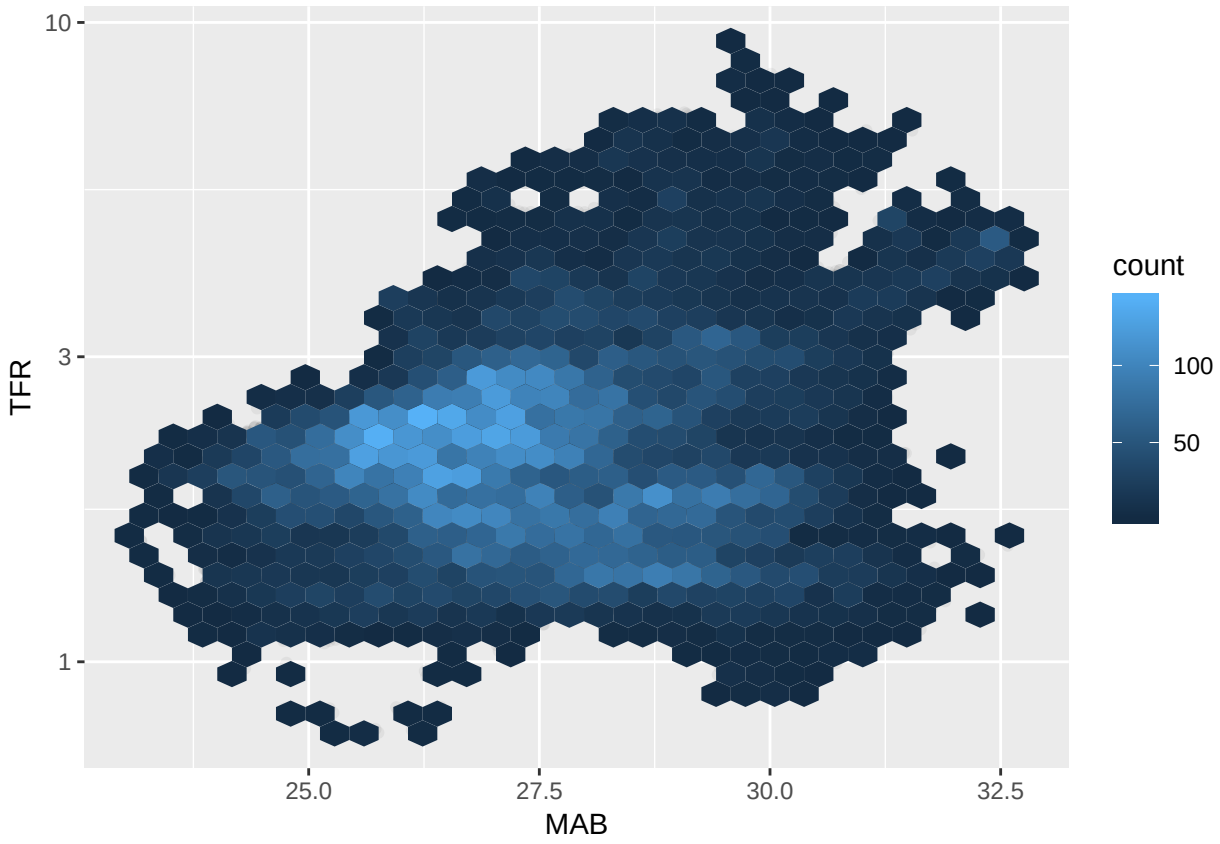
Then make a scatterplot of TFR vs MAB. What's that look like?

```
ggplot(TFRMAB, mapping = aes(x = MAB,
                             y = TFR)) +
    geom_point(alpha = .05) +
    scale_y_log10() +
    geom_point(data = filter(TFRMAB, Country == "SWE"),
               color = "red") +
    annotate(geom = "text",
             x = 27.5,
             y = 4,
             size = 15,
             label = "This is Sweden!")
```



Quick sponanteous check for alternative way to show a dense point cloud (only works if it's dense!). Sorry for wasting your time.

```
#install.packages("hexbin")
library(hexbin)
ggplot(TFRMAB, mapping = aes(x = MAB,
                             y = TFR)) +
    geom_point(alpha = .05) +
    scale_y_log10() +
    geom_hex()
```



If you wanted to add TFR to the top level rather than aggregating down, use mutate:

```
HFC %>%
    group_by(sub_id) %>%
    mutate(TFR = sum(ASFR, na.rm = TRUE),
              MAB = sum(ASFR * Age, na.rm = TRUE) / TFR) %>%
    head()
```

```
## # A tibble: 6 x 21
## # Groups:   sub_id [1]
##   Country Region Urban Origin Year1 Year2   Age AgeInt AgeDef Vitality
##   <chr>   <lgl>  <lgl> <lgl>  <dbl> <dbl> <dbl>  <dbl> <chr>     <dbl>
## 1 ABW     NA     NA    NA      1992  1992    14    -99 ACY           1
## 2 ABW     NA     NA    NA      1992  1992    15      1 ACY           1
## 3 ABW     NA     NA    NA      1992  1992    16      1 ACY           1
## 4 ABW     NA     NA    NA      1992  1992    17      1 ACY           1
## 5 ABW     NA     NA    NA      1992  1992    18      1 ACY           1
## 6 ABW     NA     NA    NA      1992  1992    19      1 ACY           1
## # ... with 11 more variables: ASFR <dbl>, CPFR <dbl>, Collection <chr>,
## #   SourceType <chr>, RefCode <chr>, Note <lgl>, Split <dbl>,
## #   YearInt <dbl>, sub_id <int>, TFR <dbl>, MAB <dbl>
```

Super tricky challenge

Solution next day, found in separate markdown file. 1. start with HFC (use those groups!) 2. Get TFR and MAB, but keep the age detail (`mutate()`) 3. Make TFR intervals, for example rounded to .25. 4. Make a column that identifies the highest and lowest MAB (like, maybe it uses NA for the others). 5. Within

each interval pick out the lowest observed and highest observed MAB, and throw out the rest (`filter()`. 6. install the `ggridges` package. 7. make a ridge plot where TFR intervals are the levels, and on each plot two distributions: the distribution (ASFR) belonging to the highest and lowest observed MAB.

```
x <- runif(10)
round(x,1)
```

```
##  [1] 0.0 0.8 0.2 0.8 0.2 0.8 0.5 0.3 0.3 0.7
```

```
x - x %% .25
```

```
##  [1] 0.00 0.75 0.00 0.75 0.00 0.75 0.50 0.25 0.25 0.50
```