Stat 260, Lecture 9: Factors, Dates and Times

David Stenning

Load packages and datasets

```r
library(tidyverse)
library(forcats)
library(lubridate)
library(nycflights13)
```

Reading

Required Reading:

- Factors, Dates and Times: Chapters 15 and 16 of online text.

Useful References:

- Factors (forcats) cheat sheet [https://github.com/rstudio/cheatsheets/raw/master/factors.pdf]
- Dates (lubridate) cheat sheet: [https://github.com/rstudio/cheatsheets/raw/master/lubridate.pdf]

Factors in R

- Factors are used to represent categorical variables in R
- The name is from experimental design.
- Factors are very useful when fitting models, but can be a bit of a pain to work with during data wrangling.
- The `forcats` package from the tidyverse aims to make working with factors more sane than base R.

Creating a Factor

- You can coerce any vector to a factor.

```r
x1 <- c("dog","cat","mouse"); x2 <- factor(x1); x2
```

```
## [1] dog   cat   mouse
## Levels: cat dog mouse
```

```r
x1 <- c(3,4,1); x2 <- factor(x1); x2
```

```
## [1] 3 4 1
## Levels: 1 3 4
```

```r
x1 <- c(TRUE,FALSE,TRUE); x2 <- factor(x1); x2
```

```
## [1] TRUE  FALSE TRUE
## Levels: FALSE TRUE
```

Creating a Factor

- The internal representation of a factor with K levels is a vector of integers in $\{1, 2, \ldots, K\}$, with an “attribute” that maps these integers to the levels.
- By default, the levels are ordered alphabetically, but you can specify the ordering when you coerce.

```r
x1 <- c("dog","cat","mouse"); x2 <- factor(x1)
str(x2)
```

```
##  Factor w/ 3 levels "cat","dog","mouse": 2 1 3
```

```r
x2 <- factor(x1,levels=c("mouse","cat","dog"))
str(x2)
```

```
##  Factor w/ 3 levels "mouse","cat",..: 3 2 1
```

```r
x2 <- factor(x1,levels=unique(x1))
str(x2)
```

```
##  Factor w/ 3 levels "dog","cat","mouse": 1 2 3
```

More or Fewer levels than Values

- There can be more or fewer levels than values in `x1`, but fewer creates missing values.

```
x1
```

```
## [1] "dog"   "cat"   "mouse"
```

```
ll <- c("cat","dog","horse","mouse")
factor(x1,levels=ll)
```

```
## [1] dog   cat   mouse
## Levels: cat dog horse mouse
```

```
ll <- ll[1:2]
factor(x1,levels=ll)
```

```
## [1] dog  cat  <NA>
## Levels: cat dog
```

Accessing the Levels

- Use `levels()` to get or set the levels.
    - We'll shy away from using `levels()` to set levels though.

```r
levels(x2)
```

```
## [1] "dog"   "cat"   "mouse"
```

```r
levels(x2) <- c("Mouse","Cat","Dog")
x2
```

```
## [1] Mouse Cat   Dog
## Levels: Mouse Cat Dog
```

Example Data: Canadian Communities Health Survey

- Survey data on older Canadians from the 10 provinces.
- Complex survey design, in which sampled individuals represent different numbers of Canadians.
    - E.G., people from rural BC are less likely to be included in the survey, they represent more Canadians than people from urban BC.

CCHS Summary

"The Canadian Community Health Survey (CCHS) - Healthy Aging is a cross-sectional survey that collected information about the factors, influences and processes that contribute to healthy aging through a multidisciplinary approach focusing on health, social and economic determinants."

"The CCHS - Healthy Aging collected responses from persons aged 45 and over living in private dwellings in the ten provinces. Excluded from the sampling frame were residents of the three territories, persons living on Indian reserves or Crown lands, persons living in institutions, full-time members of the Canadian Forces and residents of some remote regions. Data were collected between December 2008 and November 2009."

Health Utilities Index (HUI) Variables

```r
hui <- read_csv("HUI.csv.gz",col_types=cols(
  GEO_PRV = col_factor(), # province
  GEOGCMA2 = col_factor(), # urban/rural
  DHHGAGE = col_factor(), # age categories
  DHH_SEX = col_factor(), # sex
  HUIDCOG = col_factor(), # cognitive score
  HUIGDEX = col_factor(), # dexterity score
  HUIDEMO = col_factor(), # emotional
  HUIGHER = col_factor(), # hearing
  HUIDHSI = col_double(), # index variable
  HUIGMOB = col_factor(), # mobility
  HUIGSPE = col_factor(), # speech trouble
  HUIGVIS = col_factor(), # vision
  WTS_M = col_double() # sampling weights
))
```

HUIDCOG

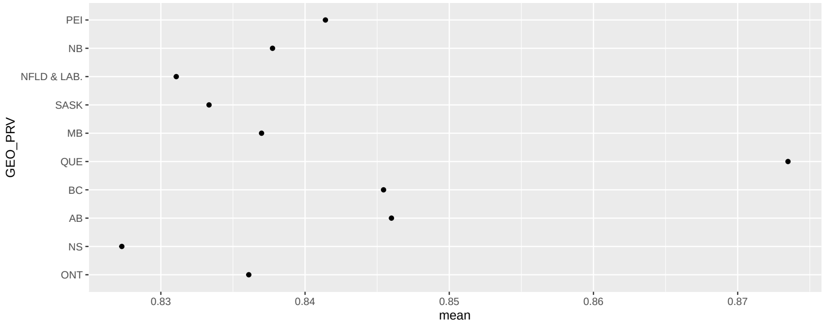
- Cognitive function has levels:

1. Able to remember most things, think clearly and solve day to day problems
2. Able to remember most things, but have a little difficulty when trying to think and solve day to day problems
3. Somewhat forgetful, but able to think clearly and solve day to day problems
4. Somewhat forgetful, and have a little difficulty when trying to think or solve day to day problems
5. Very forgetful, and have great difficulty when trying to think or solve day to day problems
6. Unable to remember anything at all, and unable to think or solve day to day problems

HUIDHSI

- An index variable derived from HUIDCOG and the seven other HUI variables that measure vision, hearing, speech, ambulation, dexterity, emotion, and pain. Index near 1 is normal, near 0 is disabled.
- We will look at how HUIDHSI varies by other factors, such as province

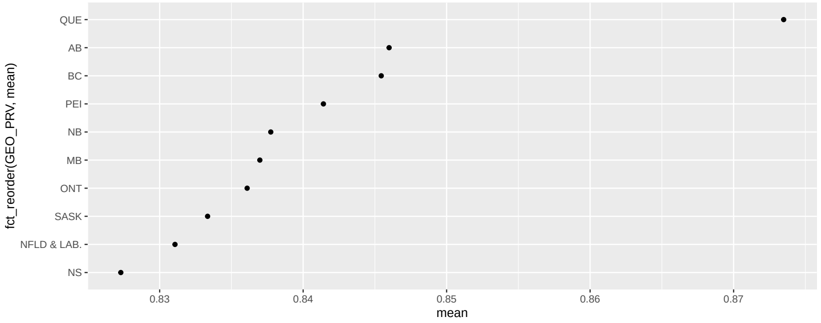
```
hui %>% group_by(GEO_PRV) %>%
  summarize(mean = weighted.mean(HUIDHSI,WTS_M,na.rm=TRUE)) %>%
  ggplot(aes(x=mean,y=GEO_PRV)) + geom_point()
```



Reordering Factor Levels

- The plot may look better with provinces sorted by the index.
- Use `fct_reorder()` to reorder levels by a second variable.

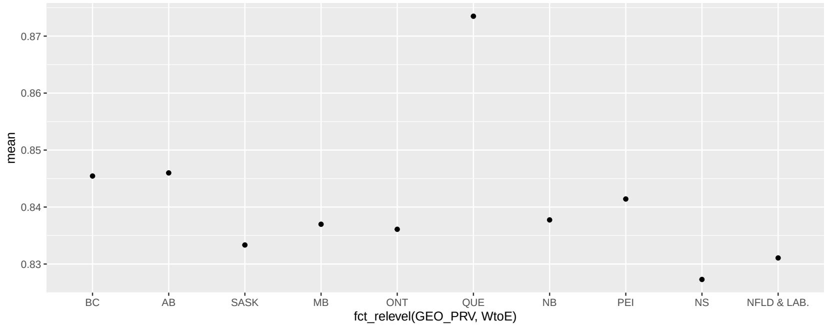
```r
hui %>% group_by(GEO_PRV) %>%
  summarize(mean = weighted.mean(HUIDHSI,WTS_M,na.rm=TRUE)) %>%
  ggplot(aes(x=mean,y=fct_reorder(GEO_PRV,mean))) + geom_point()
```



Manual Re-order

- fct_relevel() lets you move levels to the front of the ordering, to partially or completely re-order a factor's levels.

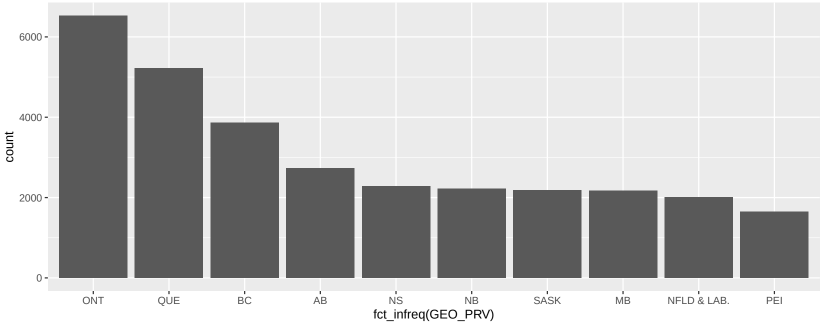
```
WtoE <- c("BC","AB","SASK","MB","ONT","QUE",
          "NB","PEI","NS","NFLD & LAB.")
hui %>% group_by(GEO_PRV) %>%
  summarize(mean = weighted.mean(HUIDHSI,WTS_M,na.rm=TRUE)) %>%
  ggplot(aes(x=fct_relevel(GEO_PRV,WtoE),y=mean)) + geom_point()
```



Ordering Levels by Frequency

- You can also order levels by their frequency in the dataset.
- This can be useful for ordering bars of a barplot.

```r
ggplot(hui,aes(x=fct_infreq(GEO_PRV))) + geom_bar()
```

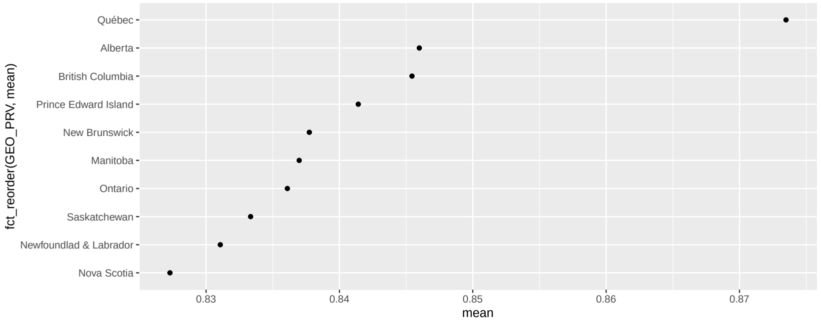


Modifying Factor Levels

- For labelling, change factor levels to complete province names.

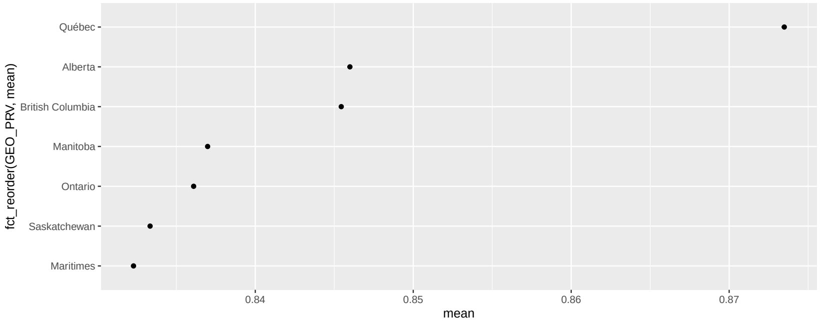
```
hui <- hui %>%
  mutate(GEO_PRV = fct_recode(GEO_PRV,
          "British Columbia" = "BC",
          "Alberta" = "AB",
          "Saskatchewan" = "SASK",
          "Manitoba" = "MB",
          "Ontario" = "ONT",
          "Québec" = "QUE",
          "New Brunswick" = "NB",
          "Prince Edward Island" = "PEI",
          "Nova Scotia" = "NS",
          "Newfoundlad & Labrador" = "NFLD & LAB."))
```

```r
hui %>% group_by(GEO_PRV) %>%
  summarize(mean = weighted.mean(HUIDHSI,WTS_M,na.rm=TRUE)) %>%
  ggplot(aes(x=mean,y=fct_reorder(GEO_PRV,mean))) + geom_point()
```



- We can also use `fct_recode()` to combine levels. (The following illustrates this feature, but is not good data analysis.)

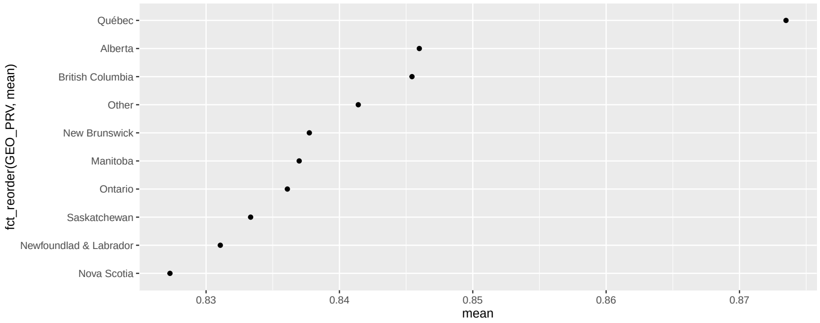
```
hui %>%
  mutate(GEO_PRV = fct_recode(GEO_PRV,
         "Maritimes" = "New Brunswick",
         "Maritimes" = "Prince Edward Island",
         "Maritimes" = "Nova Scotia",
         "Maritimes" = "Newfoundlad & Labrador")) %>%
  group_by(GEO_PRV) %>%
  summarize(mean = weighted.mean(HUIDHSI,WTS_M,na.rm=TRUE)) %>%
  ggplot(aes(x=mean,y=fct_reorder(GEO_PRV,mean))) + geom_point()
```



Grouping, or Lumping Infrequent Levels

- `fct_lump()` will group rare levels into one. (The following is not a great illustration. You'll see a better one in your exercises.)

```
hui %>% mutate(GEO_PRV = fct_lump(GEO_PRV,w=WTS_M)) %>%
  group_by(GEO_PRV) %>%
  summarize(mean = weighted.mean(HUIDHSI,WTS_M,na.rm=TRUE)) %>%
  ggplot(aes(x=mean,y=fct_reorder(GEO_PRV,mean))) + geom_point()
```



Exercise 1

- The levels of `HUIDCOG` are `COG. ATT. LEVE 1` through `COG. ATT. LEVE 6`. Recode these in terms of memory and thinking abilities, as follows:

1. good memory, clear thinking
2. good memory, some difficulty thinking
3. somewhat forgetful, clear thinking
4. somewhat forgetful, some difficulty thinking
5. very forgetful, great difficulty thinking
6. unable to remember, unable to think

- Save your recoded variable for use in Exercises 2 to 4.

Exercises 2-4

- **Exercise 2** For each level of `HUIDCOG` calculate `wtd.n = sum(WTS_M)`. Plot the levels of `HUIDCOG` *versus* the log-base10 of `wtd.n` with `geom_point()`.
- **Exercise 3** Repeat exercise 2, but with the levels of `HUIDCOG` ordered by `wtd.n`.
- **Exercise 4** Repeat Exercise 3, but with the levels of `HUIDCOG` recoded according to memory as “good memory”, “somewhat forgetful”, “very forgetful” or “unable to remember”.

Dates and Times

- Moments in time can be dates, times, or a combination called a date-time.
- We have seen dates and times before when parsing input files, and in the `flights` tibble from `nycflights13`.
- Focus on dates and date-times.
- We will create dates and date-times from strings, and as combinations of date and time objects.

Dates and Times From Strings

- The `lubridate` package contains functions to coerce strings to date objects:
  - `ymd()` to coerce data in year-month-date, `mdy()` to coerce data in month-day-year, `ymd_hm()` to coerce data in year-month-date-hour-minute, etc.

```r
ymd("19-09-01"); ymd("20190901"); ymd("2019September01")
```

```
## [1] "2019-09-01"

## [1] "2019-09-01"

## [1] "2019-09-01"
```

```r
mdy("09-01-2019"); mdy("09012019");mdy("Sep 1, 2019")
```

```
## [1] "2019-09-01"

## [1] "2019-09-01"

## [1] "2019-09-01"
```

```r
ymd_hm("19-09-01 2:00")
```

```
## [1] "2019-09-01 02:00:00 UTC"
```

Time Zones

- Time data includes a time zone. For some summaries the time zone doesn't matter and you can leave it at the default, UTC (aka Grenwich Mean Time).
- To set a time zone with the lubridate time functions, use the `tz` argument.
- Time zones names are from a time-zone database called Olson/IANA, and are not the abbreviations (like PDT) that you might expect.
- Vancouver is in the "America/Vancouver" time zone.

```r
Sys.timezone()
```

```
## [1] "America/Vancouver"
```

Example data

- Hourly weather data from YVR airport in October, 2019.
  - `read_csv()` figures out the `Time` column, but not the Date/Time column.

```
yvr <- read_csv("weatherYVROct2019.csv") %>%
  select("Date/Time",Year,Month,Day,Time,"Temp (C)")
yvr
```

```
## # A tibble: 744 x 6
##    `Date/Time`     Year Month   Day Time   `Temp (C)`
##    <chr>          <dbl> <dbl> <dbl> <time>      <dbl>
##  1 19-10-01 0:00   2019    10     1 00:00         7.9
##  2 19-10-01 1:00   2019    10     1 01:00         6.1
##  3 19-10-01 2:00   2019    10     1 02:00         5.8
##  4 19-10-01 3:00   2019    10     1 03:00         5.3
##  5 19-10-01 4:00   2019    10     1 04:00         6  
##  6 19-10-01 5:00   2019    10     1 05:00         7  
##  7 19-10-01 6:00   2019    10     1 06:00         6.3
##  8 19-10-01 7:00   2019    10     1 07:00         7.1
##  9 19-10-01 8:00   2019    10     1 08:00         8.8
## 10 19-10-01 9:00   2019    10     1 09:00        10.3
## # ... with 734 more rows
```

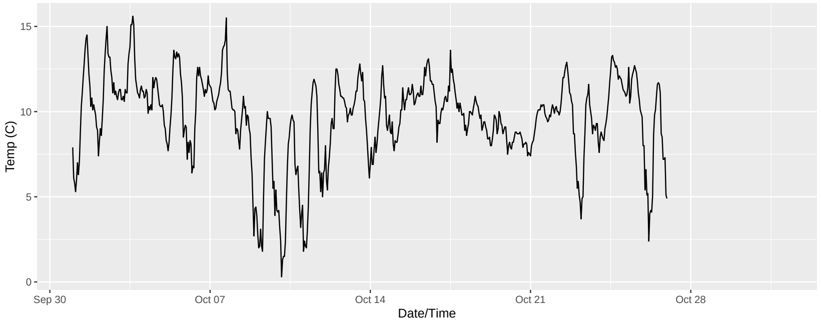
Coerce Date/Time

```r
yvr <- yvr %>% mutate(`Date/Time` =
                        ymd_hm(`Date/Time`,tz="America/Vancouver"))
yvr
```

```
## # A tibble: 744 x 6
##    `Date/Time`          Year Month   Day Time   `Temp (C)`
##    <dttm>              <dbl> <dbl> <dbl> <time>      <dbl>
##  1 2019-10-01 00:00:00  2019    10     1 00:00         7.9
##  2 2019-10-01 01:00:00  2019    10     1 01:00         6.1
##  3 2019-10-01 02:00:00  2019    10     1 02:00         5.8
##  4 2019-10-01 03:00:00  2019    10     1 03:00         5.3
##  5 2019-10-01 04:00:00  2019    10     1 04:00         6
##  6 2019-10-01 05:00:00  2019    10     1 05:00         7
##  7 2019-10-01 06:00:00  2019    10     1 06:00         6.3
##  8 2019-10-01 07:00:00  2019    10     1 07:00         7.1
##  9 2019-10-01 08:00:00  2019    10     1 08:00         8.8
## 10 2019-10-01 09:00:00  2019    10     1 09:00        10.3
## # ... with 734 more rows
```

Time Series Plots with Date-times

```r
ggplot(yvr,aes(x=`Date/Time`,y=`Temp (C)`)) + geom_line()
```



Date-time from Components

- `make_datetime()` makes a date-time object from components.
- `hour()`, `minute()`, etc. extract components.

```
yvrdt <- yvr %>%
  mutate(datetime =
     make_datetime(Year,Month,Day,hour(Time),minute(Time),
     tz = "America/Vancouver")) %>%
  select(`Date/Time`,datetime)
yvrdt
```

```
## # A tibble: 744 x 2
##    `Date/Time`         datetime
##    <dttm>              <dttm>
##  1 2019-10-01 00:00:00 2019-10-01 00:00:00
##  2 2019-10-01 01:00:00 2019-10-01 01:00:00
##  3 2019-10-01 02:00:00 2019-10-01 02:00:00
##  4 2019-10-01 03:00:00 2019-10-01 03:00:00
##  5 2019-10-01 04:00:00 2019-10-01 04:00:00
##  6 2019-10-01 05:00:00 2019-10-01 05:00:00
##  7 2019-10-01 06:00:00 2019-10-01 06:00:00
##  8 2019-10-01 07:00:00 2019-10-01 07:00:00
##  9 2019-10-01 08:00:00 2019-10-01 08:00:00
## 10 2019-10-01 09:00:00 2019-10-01 09:00:00
## #     with 734 more rows
```

Durations

- Differences in date/time objects are reported in days, hours and minutes.
- Durations are always in seconds.

```
(ymd_hms("20191029 01:01:00") - ymd_hms("20191028 01:00:00"))
```

```
## Time difference of 1.000694 days
```

```
(ymd_hms("20191028 01:01:00") - ymd_hms("20191028 01:00:00"))
```

```
## Time difference of 1 mins
```

```
as.duration(ymd_hms("20191028 01:01:00") - ymd_hms("20191028 01:00:00"))
```

```
## [1] "60s (~1 minutes)"
```

```r
yvrdt %>%
  mutate(diff = datetime - mean(datetime),
         diffsec = as.duration(datetime - mean(datetime))) %>%
  summarize(sd=sd(diff), sd2 = sd(diffsec))
```

```
## # A tibble: 1 x 2
##        sd     sd2
##     <dbl>   <dbl>
## 1 12895. 773707.
```

Exercise 5

- Verify that the `datetime` object created on about slide 30 is equal to the `Date/Time` object created earlier with `ymd_hm()`.
- Use the `sd()` function to calculate the SD of the `datetime` object. What are the units?