Statistical Methods for Insurance: Compiling data for problem solving

Di Cook & Souhaib Ben Taieb, Econometrics and Business Statistics, Monash University

W8.C2

Overview of this class

- String operations, working with `text`
- Reading different `data formats`
- Handling missing data

Working with text

```
tb <- read_csv("../data/tb.csv")
tb[7:10,1:10]
#> # A tibble: 4 x 10
#>    iso2  year  m_04 m_514 m_014 m_1524 m_2534 m_3544 m_4554 m_5564
#>   <chr> <int> <int> <int> <int>  <int>  <int>  <int>  <int>  <int>
#> 1    AD  1996    NA    NA     0      0      0      4      1      0
#> 2    AD  1997    NA    NA     0      0      1      2      2      1
#> 3    AD  1998    NA    NA     0      0      0      1      0      0
#> 4    AD  1999    NA    NA     0      0      0      1      1      0
```

Convert to long form

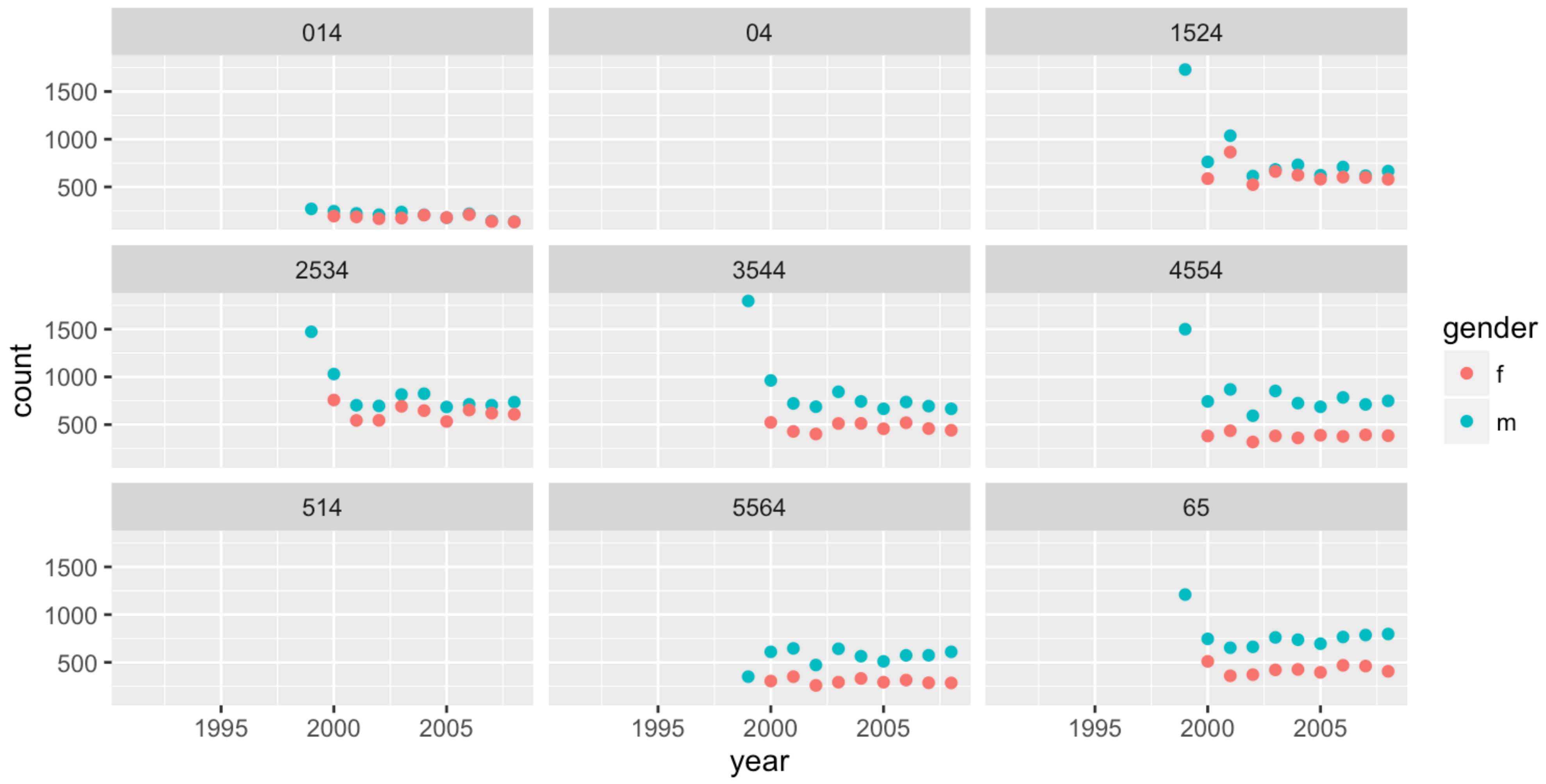
```
tb_long <- tb %>% gather(variable, count, m_04:f_u)
head(tb_long)
#> # A tibble: 6 x 4
#>    iso2  year variable count
#>   <chr> <int>    <chr> <int>
#> 1    AD  1989     m_04    NA
#> 2    AD  1990     m_04    NA
#> 3    AD  1991     m_04    NA
#> 4    AD  1992     m_04    NA
#> 5    AD  1993     m_04    NA
#> 6    AD  1994     m_04    NA
```

String split

```
tb_long <- tb_long %>% separate(variable, c("gender", "age"), "_")
head(tb_long)
#> # A tibble: 6 x 5
#>    iso2  year gender   age count
#>   <chr> <int>  <chr> <chr> <int>
#> 1    AD  1989      m    04    NA
#> 2    AD  1990      m    04    NA
#> 3    AD  1991      m    04    NA
#> 4    AD  1992      m    04    NA
#> 5    AD  1993      m    04    NA
#> 6    AD  1994      m    04    NA
```

Take a look

```
tb_long %>% filter(iso2 == "CO", age!="u", year>1990) %>%
  ggplot(aes(x=year, y=count, colour=gender)) + geom_point() + facet_wrap(~age, ncol=3)
```



Reading different file formats: shapefiles

The Australian Electorate Commission publishes the boundaries of the electorates on their website at http://www.aec.gov.au/Electorates/gis/gis_datadownload.htm.

Once the files (preferably the national files) are downloaded, unzip the file (it will build a folder with a set of files). We want to read the shapes contained in the `shp` file into R.

```r
library(maptools)

# shapeFile contains the path to the shp file:
shapeFile <- "../data/vic-esri-24122010/vic 24122010.shp"
sF <- readShapeSpatial(shapeFile)
class(sF)
#> [1] "SpatialPolygonsDataFrame"
#> attr(,"package")
#> [1] "sp"
```

`sF` is a spatial data frame containing all of the polygons. We use the `rmapshaper` package available from ateucher's github page to thin the polygons while preserving the geography:

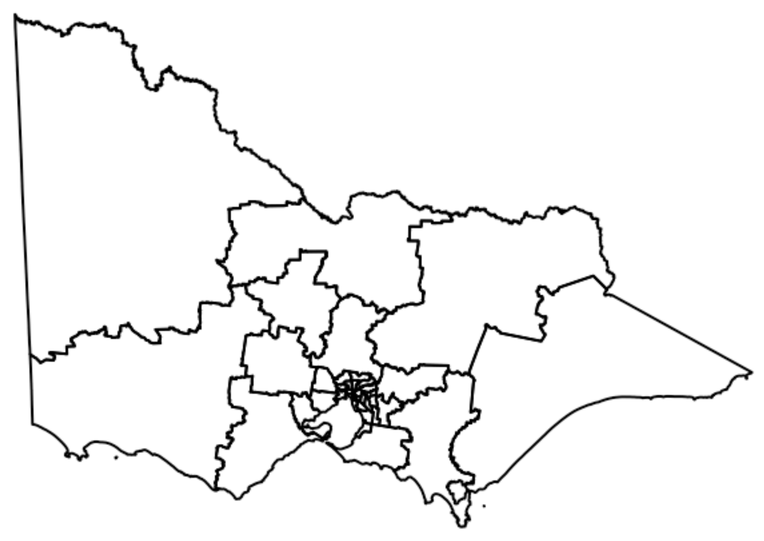
```
library(rmapshaper)
```

```
sFsmall <- ms_simplify(sF, keep=0.05) # use instead of thinnedSpatialPoly
```

`keep` indicates the percentage of points we want to keep in the polygons. 5% makes the electorate boundary still quite recognizable, but reduce the overall size of the map considerably, making it faster to plot.

We can use base graphics to plot this map:

```
plot(sFsmall)
```



Extracting the electorate information

A spatial polygons data frame consists of both a data set with information on each of the entities (in this case, electorates), and a set of polygons for each electorate (sometimes multiple polygons are needed, e.g. if the electorate has islands). We want to extract both of these parts.

```
nat_data <- sF@data
head(nat_data)
#>   GEODB_OID OBJECTID DIV_NUMBER ELECT_DIV NUMCCDS ACTUAL PROJECTED
#> 0         1        1          1     Aston     190  92370     98469
#> 1         2        2          2  Ballarat     274  95003    100786
#> 2         3        3          3    Batman     265  96909    104258
#> 3         4        4          4   Bendigo     284  95729    102582
#> 4         5        5          5     Bruce     226  95472     99904
#> 5         6        6          6   Calwell     214  99031    104734
#>   POPULATION OVER_18 AREA_SQKM SORTNAME                MAP_SYMBOL MAP_TYPE
#> 0          0       0   98.9337    Aston Final Divisional Boundary    Metro
#> 1          0       0 4651.6400 Ballarat Final Divisional Boundary    Rural
#> 2          0       0   65.6887   Batman Final Divisional Boundary    Metro
#> 3          0       0 6255.0000  Bendigo Final Divisional Boundary    Rural
#> 4          0       0   72.6900    Bruce Final Divisional Boundary    Metro
#> 5          0       0  174.7130  Calwell Final Divisional Boundary    Metro
#>      LENGTH SHAPE_AREA
#> 0  58422.89   99056594
#> 1 417555.12 4652896627
#> 2  58226.02   65717035
#> 3 622901.96 6255411672
#> 4  48696.38   72629548
#> 5  81925.61  174767685
```

The row names of the data file are identifiers corresponding to the polygons - we want to make them a separate variable:

```
nat_data$id <- row.names(nat_data)
```

Extracting the polygon information

The `fortify` function in the `ggplot2` package extracts the polygons into a data frame.

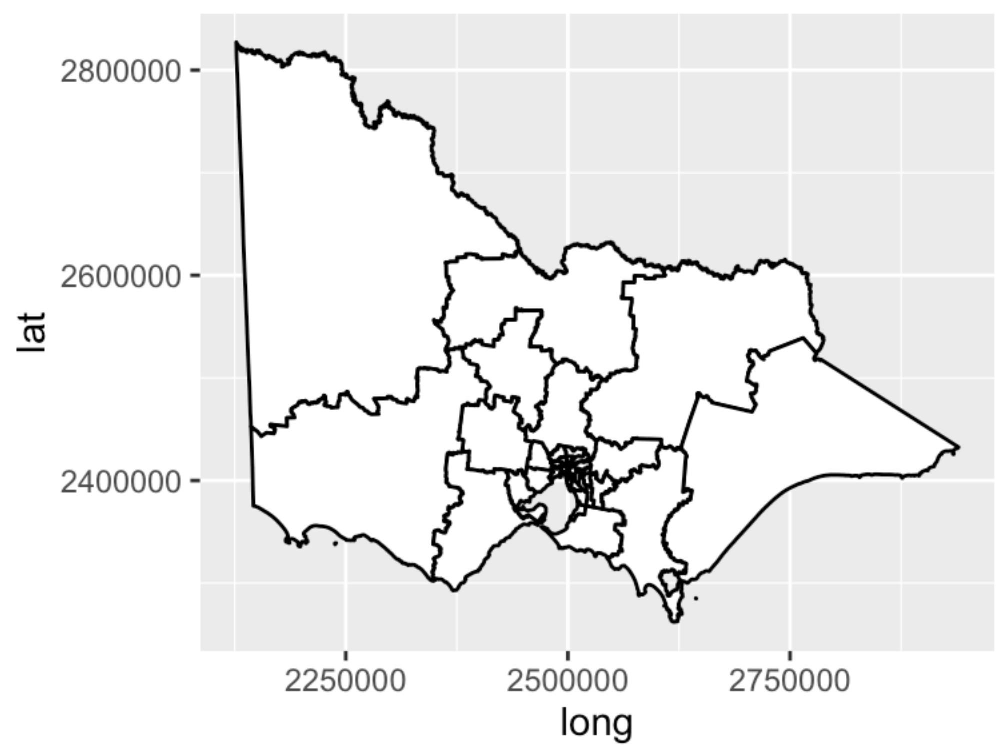
```
nat_map <- ggplot2::fortify(sFsmall)
head(nat_map)
#>      long     lat order  hole piece id group
#> 1 2525287 2407463     1 FALSE     1  0   0.1
#> 2 2525840 2407413     2 FALSE     1  0   0.1
#> 3 2527187 2406561     3 FALSE     1  0   0.1
#> 4 2527167 2406442     4 FALSE     1  0   0.1
#> 5 2527987 2406306     5 FALSE     1  0   0.1
#> 6 2527967 2406186     6 FALSE     1  0   0.1
```

We need to make sure that `group` and `piece` are kept as factor variables - if they are allowed to be converted to numeric values, it messes things up, because as factor levels `9` and `9.0` are distinct, whereas they are not when interpreted as numbers …

```
nat_map$group <- paste("g",nat_map$group,sep=".")
nat_map$piece <- paste("p",nat_map$piece,sep=".")
head(nat_map)
#>      long     lat order  hole piece id group
#> 1 2525287 2407463     1 FALSE   p.1  0 g.0.1
#> 2 2525840 2407413     2 FALSE   p.1  0 g.0.1
#> 3 2527187 2406561     3 FALSE   p.1  0 g.0.1
#> 4 2527167 2406442     4 FALSE   p.1  0 g.0.1
#> 5 2527987 2406306     5 FALSE   p.1  0 g.0.1
#> 6 2527967 2406186     6 FALSE   p.1  0 g.0.1
```

Plot it

```r
ggplot(nat_map, aes(x=long, y=lat, group=group)) +
  geom_polygon(fill="white", colour="black")
```



Handling missing values

- Need to know how the missings are coded, hopefully clearly missing, treated as NA in R, not 0, or -9, or -9999, or . Recode as need be.
- Study the distribution of missing vs not missing, which will help determine how to handle them.

What ways can these affect analysis?

- If missings happen when conditions are special, eg sensor tends to stop when temperature drops below 3 degrees Celsius, estimation of model parameters may not reflect the population parameters
- Some techniques, particularly multivariate methods like many used in data mining require complete records over many variables. Just a few missing numbers can mean a lot of cases that cannot be used.

Making it Easy - MissingDataGUI

- Methods for summarising missings in a data set
- Ways to plot to examine dependence between missing vs not missing
- Imputation methods to substitute missings

```
library(MissingDataGUI)
data(tao)
MissingDataGUI(tao)
```

Resources

- eechidna: Exploring Election and Census Highly Informative Data Nationally for Australia
- AEC electorate polygons
- Paper on the MissingDataGUI

Share and share alike