Loops, Maps, and Table One

Abhijit Dasgupta

Fall, 2019

Where we are

- Got a start on plotting and creating panelled graphs with ggplot2
- Can modify a data set somewhat
    - `dplyr` verbs (mutate, filter, select, separate, unite)
    - joins
    - gather/spread

Repetitive copying

For HW 6 (sorry about the mess), you had to copy and paste multiple times to get things done

- Had to do same processing on multiple data sets
- Had to do same graphs from multiple data sets

For loops and Maps

For loops

For-loops are a computational structure that allows you to do the same thing repeatedly over a loop with some index.

The basic structure is

```
for (variable in vector) {
  <code to execute for each iteration>
}
```

For loops

Using numeric indices

```
library(fs)
sites <- c('Brain','Colon','Esophagus','Lung','Oral')
dats <- list() # Initialize empty list
for(i in 1:length(sites)){
  dats[[i]] <-
    read_csv(path('data',
                          paste0(sites[i], '.csv'),
                     skip=4)
}
```

Using names

```
library(fs)
sites <- c('Brain','Colon','Esophagus','Lung','Oral')
dats <- list() # Initialize empty list
for(n in sites){
  dats[[n]] <-
    read_csv(path('data',
                  paste0(n, '.csv')),
             skip=4)
}
```

Lists

Directly using lists has efficiency advantages. `rio` can load all the datasets into a list, for example.

```
dats <- rio::import_list(path('data', paste0(sites,'.csv')))
names(dats)
```

```
#>  [1] "Brain"     "Colon"     "Esophagus" "Lung"      "Oral"
```

```
str(dats[['Brain']])
```

```
#>  'data.frame':    43 obs. of  10 variables:
#>   $ Year of Diagnosis   : chr  "1975-2016" "1975" "1976" "1977" ...
#>   $ All Races,Both Sexes: num  6.59 5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 ...
#>   $ All Races,Males     : num  7.88 6.84 7.14 7.76 6.79 7.42 7.58 8.07 7.93 7.6 ...
#>   $ All Races,Females   : num  5.51 5.01 4.68 4.89 4.91 5.01 5.24 5.2 5.24 5.19 ...
#>   $ Whites,Both Sexes   : num  7.22 6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 ...
#>   $ Whites,Males        : num  8.61 7.31 7.51 8.26 7.19 8.03 8.2 8.44 8.57 8.2 ...
#>   $ Whites,Females      : num  6.04 5.28 5.03 5.27 5.19 5.37 5.65 5.63 5.64 5.74 ...
#>   $ Blacks,Both Sexes   : num  4.08 4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 ...
#>   $ Blacks,Males        : num  4.79 4.31 5.37 5.17 4.34 4.19 3.35 7.24 4.4 3.79 ...
#>   $ Blacks,Females      : chr  "3.51" "3.88" "-" "2.47" ...
```

Recall, lists are the most generic buckets in R. Elements of lists can be anything. To use `map` it's best that each element of the input list be of the same type

Maps

`map` is like a for-loop, but strictly for lists. It is more efficient than for-loops. The basic template is:

```
map(<list>, <function>, <function arguments>)
```

For example, if we want to take out the first row of each dataset and make sure all the variables are numeric, we could do:

```
dats <- map(dats, function(d){
  d %>% slice(-1) %>%  # remove first row
    mutate_all( as.numeric)
})
str(dats[['Brain']])
```

```
#>  'data.frame':    42 obs. of  10 variables:
#>   $ Year of Diagnosis   : num  1975 1976 1977 1978 1979 ...
#>   $ All Races,Both Sexes: num  5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 6.12 ...
#>   $ All Races,Males     : num  6.84 7.14 7.76 6.79 7.42 7.58 8.07 7.93 7.6 7.18 ...
#>   $ All Races,Females   : num  5.01 4.68 4.89 4.91 5.01 5.24 5.2 5.24 5.19 5.2 ...
#>   $ Whites,Both Sexes   : num  6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 6.49 ...
#>   $ Whites,Males        : num  7.31 7.51 8.26 7.19 8.03 8.2 8.44 8.57 8.2 7.64 ...
#>   $ Whites,Females      : num  5.28 5.03 5.27 5.19 5.37 5.65 5.63 5.64 5.74 5.49 ...
#>   $ Blacks,Both Sexes   : num  4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 4.53 ...
#>   $ Blacks,Males        : num  4.31 5.37 5.17 4.34 4.19 3.35 7.24 4.4 3.79 5.34 ...
#>   $ Blacks,Females      : num  3.88 NA 2.47 3.51 3.23 2.92 3.16 3.05 1.84 3.88 ...
```

Maps

`map` is like a for-loop, but strictly for lists. It is more efficient than for-loops. The basic template is:

```
map(<list>, <function>, <function arguments>)
```

For example, if we want to take out the first row of each dataset and make sure all the variables are numeric, we could do:

```
dats <- map(dats, function(d){
  d %>% slice(-1) %>%  # remove first row
    mutate_all( as.numeric)
})
str(dats[['Brain']])
```

The argument for the function inside the `map` function is an element of the list. In this case, it is a data frame.

The output of `map` is a list the same length as the input list.

Maps

I don't like the names with spaces, so I can just apply a function to each data set to fix that.

```
dats <- map(dats, janitor::clean_names)
str(dats[['Oral']])
```

```
#>  'data.frame':    42 obs. of  10 variables:
#>   $ year_of_diagnosis   : num  1975 1976 1977 1978 1979 ...
#>   $ all_races_both_sexes: num  13.2 13.3 12.7 13.4 14 ...
#>   $ all_races_males     : num  21.2 21 20.1 20.9 21.9 ...
#>   $ all_races_females   : num  7.09 7.39 6.94 7.71 7.98 7.91 7.91 7.93 7.24 7.86 ...
#>   $ whites_both_sexes   : num  13.3 13.2 12.6 13.2 13.7 ...
#>   $ whites_males        : num  21.7 21.1 19.9 20.7 21.6 ...
#>   $ whites_females      : num  6.94 7.38 7 7.57 7.72 7.62 7.95 7.85 7.28 7.64 ...
#>   $ blacks_both_sexes   : num  13.4 15.2 14.5 15.9 18.5 ...
#>   $ blacks_males        : num  20.2 23.8 23.9 26 28.2 ...
#>   $ blacks_females      : num  8.23 8.37 6.77 8.18 10.77 ...
```

Maps

Now let's split up by sexes

```
dats_all <- map(dats, select, year_of_diagnosis, ends_with('sexes'))
dats_male <- map(dats, select, year_of_diagnosis, ends_with('_males'))
dats_female <- map(dats, select, year_of_diagnosis, ends_with('females'))
str(dats_all[['Esophagus']])
```

```
#>  'data.frame':    42 obs. of  4 variables:
#>   $ year_of_diagnosis   : num  1975 1976 1977 1978 1979 ...
#>   $ all_races_both_sexes: num  4.14 4.3 4.06 4.12 4.42 4.27 4.14 4.26 4.29 4.18 ...
#>   $ whites_both_sexes   : num  3.55 3.72 3.33 3.41 3.73 3.54 3.31 3.46 3.57 3.52 ...
#>   $ blacks_both_sexes   : num  10.9 10.7 12 13.1 12.9 ...
```

Here I used the form `map(<list>, <function>, <function arguments>)`.

Earlier I had used `map(<list>,<function definition>)` and `map(<list>, <function>)` with no (i.e., default) arguments.

Maps

Let's make the column headers of each dataset reflect the site, so that when we join we can keep the sites separate

```
for(n in sites){
  names(dats_all[[n]]) <- str_replace(names(dats_all[[n]]), 'both_sexes',n)
  names(dats_male[[n]]) <- str_replace(names(dats_male[[n]]), 'male',n)
  names(dats_female[[n]]) <- str_replace(names(dats_female[[n]]), 'female',n)
}
names(dats_all[['Esophagus']])
```

```
#>  [1] "year_of_diagnosis"   "all_races_Esophagus" "whites_Esophagus"
#>  [4] "blacks_Esophagus"
```

Higher order maps

When we joined these data sets, we had to repeatedly use `left_join` to create the final data set. There is a shortcut to this repeated operation of a function with two inputs as applied to a list successively.

```
dats2_all <- Reduce(left_join, dats_all)
dats2_male <- Reduce(left_join, dats_male)
dats2_female <- Reduce(left_join, dats_female)
```

> Could we have used a for loop or map here? Sure, but it makes it harder to read IMO.

```
str(dats2_all)
```

```
#>  'data.frame':    42 obs. of  16 variables:
#>   $ year_of_diagnosis  : num  1975 1976 1977 1978 1979 ...
#>   $ all_races_Brain    : num  5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 6.12 ...
#>   $ whites_Brain       : num  6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 6.49 ...
#>   $ blacks_Brain       : num  4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 4.53 ...
#>   $ all_races_Colon    : num  59.5 61.3 62.4 62 62.4 ...
#>   $ whites_Colon       : num  60.2 62.2 63.2 62.8 63 ...
#>   $ blacks_Colon       : num  56.9 55 60.8 62.2 58.6 ...
#>   $ all_races_Esophagus: num  4.14 4.3 4.06 4.12 4.42 4.27 4.14 4.26 4.29 4.18 ...
#>   $ whites_Esophagus   : num  3.55 3.72 3.33 3.41 3.73 3.54 3.31 3.46 3.57 3.52 ...
#>   $ blacks_Esophagus   : num  10.9 10.7 12 13.1 12.9 ...
#>   $ all_races_Lung     : num  52.2 55.4 56.7 57.8 58.6 ...
#>   $ whites_Lung        : num  51.9 54.6 55.9 57.2 58 ...
#>   $ blacks_Lung        : num  64.5 72.3 73.6 74.4 74.5 ...
```

Maps

Next, we want to separate the races from the sites, after a `gather`. The `all_races` will pose a problem if we split on _. Let's fix that.

```
names(dats2_all) <- str_replace(names(dats2_all), 'all_races','allraces')
names(dats2_male) <- str_replace(names(dats2_male), 'all_races','allraces')
names(dats2_female) <- str_replace(names(dats2_female), 'all_races','allraces')
```

Maps

Now, for each of these , we need to gather then separate. We'll put the data sets in a list first

```
dats2 <- list('both'=dats2_all, 'male'=dats2_male, 'female'=dats2_female)
dats2 <- map(dats2,
             function(d){
               d %>% tidyr::gather(variable, rate, -year_of_diagnosis) %>%
                 separate(variable, c('race','site'), sep='_')
             })
str(dats2[['both']])
```

```
#>  'data.frame':    630 obs. of  4 variables:
#>   $ year_of_diagnosis: num  1975 1976 1977 1978 1979 ...
#>   $ race             : chr  "allraces" "allraces" "allraces" "allraces" ...
#>   $ site             : chr  "Brain" "Brain" "Brain" "Brain" ...
#>   $ rate             : num  5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 6.12 ...
```

Final graphing

Now we're in a position to do the graphing.

```
pltlist <- dats2[['both']] %>% group_split(race) %>%
  map(function(d) {ggplot(d,
                          aes(x = year_of_diagnosis,
                              y = rate,
                              color=site))+
  geom_point(show.legend = F) })
cowplot::plot_grid(plotlist=pltlist, ncol=1,
                   labels=c('Both','Males','Females'))
```

I'm using quite advanced R here, but hopefully you'll learn by example.

`group_split` splits the dataset by the values of the grouping variable into a list

(Yes, your homework asked for a different panel placement)

Table One

When we start a manuscript, we start with Table 1, which gives the description of the data. We can do this manually, but there is a really nice `tableone` package that does this well.

	1	2	p	test
n	158	154		
time (mean (SD))	2015.62 (1094.12)	1996.86 (1155.93)	0.883	
status (%)			0.894	
0	83 (52.5)	85 (55.2)		
1	10 (6.3)	9 (5.8)		
2	65 (41.1)	60 (39.0)		
trt = 2 (%)	0 (0.0)	154 (100.0)	<0.001	
age (mean (SD))	51.42 (11.01)	48.58 (9.96)	0.018	
sex = f (%)	137 (86.7)	139 (90.3)	0.421	
ascites = 1 (%)	14 (8.9)	10 (6.5)	0.567	
hepato = 1 (%)	73 (46.2)	87 (56.5)	0.088	
spiders = 1 (%)	45 (28.5)	45 (29.2)	0.985	
edema (%)			0.877	
0	132 (83.5)	131 (85.1)		
0.5	16 (10.1)	13 (8.4)		

We'll use the `pbc` dataset in the `survival` package

```
library(tableone)
kableone(CreateTableOne(data=pbc), format='html')
```

	Overall
n	418
id (mean (SD))	209.50 (120.81)
time (mean (SD))	1917.78 (1104.67)
status (mean (SD))	0.83 (0.96)
trt (mean (SD))	1.49 (0.50)
age (mean (SD))	50.74 (10.45)
sex = f (%)	374 (89.5)
ascites (mean (SD))	0.08 (0.27)
hepato (mean (SD))	0.51 (0.50)
spiders (mean (SD))	0.29 (0.45)
edema (mean (SD))	0.10 (0.25)
bili (mean (SD))	3.22 (4.41)
chol (mean (SD))	369.51 (231.94)
albumin (mean (SD))	3.50 (0.42)

Some of these may be categorical, and some numeric

```
myVars <- c("time", "status", "trt", "age",
            "sex", "ascites", "hepato",
            "spiders", "edema", "bili", "chol",
            "albumin", "copper", "alk.phos",
            "ast", "trig", "platelet", "protime", "st
## Vector of categorical variables that need transfor
catVars <- c("status", "trt", "ascites", "hepato",
             "spiders", "edema", "stage")
tab2 <- CreateTableOne(data=pbc,
                       vars=myVars,
                       factorVars=catVars)
kableone(tab2, format='html')
```

	Overall
n	418
time (mean (SD))	1917.78 (1104.67)
status (%)	
0	232 (55.5)
1	25 (6.0)
2	161 (38.5)
trt = 2 (%)	154 (49.4)
age (mean (SD))	50.74 (10.45)
sex = f (%)	374 (89.5)
ascites = 1 (%)	24 (7.7)
hepato = 1 (%)	160 (51.3)
spiders = 1 (%)	90 (28.8)
edema (%)	
0	354 (84.7)
0.5	44 (10.5)
1	20 (4.8)

You can also get missingness information from these summary objects

```
summary(tab2)
```

```
#>
#>       ### Summary of continuous variables ###
#>
#>  strata: Overall
#>              n miss p.miss mean    sd median   p25  p75   min   max  skew
#>  time      418    0    0.0 1918 1e+03   1730 1e+03 2614  41.0  4795  0.47
#>  age       418    0    0.0   51 1e+01     51 4e+01   58  26.3    78  0.09
#>  bili      418    0    0.0    3 4e+00      1 8e-01    3   0.3    28  2.72
#>  chol      418  134   32.1  370 2e+02    310 2e+02  400 120.0  1775  3.41
#>  albumin   418    0    0.0    3 4e-01      4 3e+00    4   2.0     5 -0.47
#>  copper    418  108   25.8   98 9e+01     73 4e+01  123   4.0   588  2.30
#>  alk.phos 418  106   25.4 1983 2e+03   1259 9e+02 1980 289.0 13862  2.99
#>  ast       418  106   25.4  123 6e+01    115 8e+01  152  26.4   457  1.45
#>  trig      418  136   32.5  125 7e+01    108 8e+01  151  33.0   598  2.52
#>  platelet 418   11    2.6  257 1e+02    251 2e+02  318  62.0   721  0.63
#>  protime   418    2    0.5   11 1e+00     11 1e+01   11   9.0    18  2.22
#>            kurt
#>  time      -0.5
#>  age       -0.6
#>  bili       8.1
#>  chol      14.3
#>  albumin    0.6
#>  copper     7.6
#>  alk.phos   9.7
#>  ast        4.3
#>  trig      11.8
#>  platelet   0.9
#>  protime   10.0
#>
```

Often times lab values or biomarker levels are rather skew, and aren't summarized well by the mean. You can specify them as non-normal.

```
biomarkers <- c("bili","chol","copper","alk.phos","as
tab3 <- CreateTableOne(data=pbc,
                       vars=biomarkers)
kableone(print(tab3, nonnormal = biomarkers), format=
```

	Overall
n	418
bili (median [IQR])	1.40 [0.80, 3.40]
chol (median [IQR])	309.50 [249.50, 400.00]
copper (median [IQR])	73.00 [41.25, 123.00]
alk.phos (median [IQR])	1259.00 [871.50, 1980.00]
ast (median [IQR])	114.70 [80.60, 151.90]
trig (median [IQR])	108.00 [84.25, 151.00]
protime (median [IQR])	10.60 [10.00, 11.10]

We can test differences by strata too

```
myvars=c('age', biomarkers)
tab4 <- CreateTableOne(data=pbc,
                       vars=myvars,
                       strata='trt')
kableone(
  print(tab4, nonnormal=biomarkers),
  format='html'
)
```

	1	2	p	test
n	158	154		
age (mean (SD))	51.42 (11.01)	48.58 (9.96)	0.018	
bili (median [IQR])	1.40 [0.80, 3.20]	1.30 [0.72, 3.60]	0.842	nonnorm
chol (median [IQR])	315.50 [247.75, 417.00]	303.50 [254.25, 377.00]	0.544	nonnorm
copper (median [IQR])	73.00 [40.00, 121.00]	73.00 [43.00, 139.00]	0.717	nonnorm
alk.phos (median [IQR])	1214.50 [840.75, 2028.00]	1283.00 [922.50, 1949.75]	0.812	nonnorm