An annotated R script for Homework 6 using lists and maps

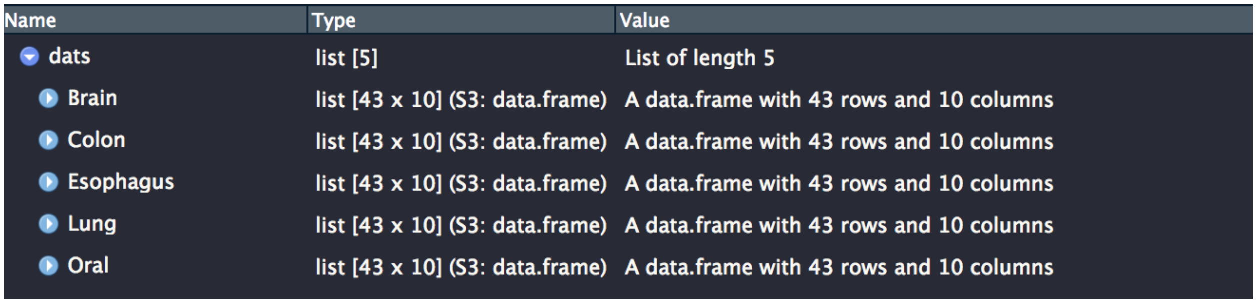
First, read the data in. We've stored just the data files in the folder `data/HW6`.

```
library(tidyverse)
library(rio)

fnames <-  dir('data/HW6', pattern='csv', full.names=T)
# full.names=T ensures that the full relative path to the files is returned

# Now pull the data from multiple files into R in one go
dats <- import_list(fnames, skip=4) # from the `rio` package
```

This command creates a **list** of data frames. If you look at `dats` in the Environment tab, you'll see that is is a list of 5 objects. If you click on the magnifying glass to the right of that listing, you'll see



Name	Type	Value
dats	list [5]	List of length 5
Brain	list [43 x 10] (S3: data.frame)	A data.frame with 43 rows and 10 columns
Colon	list [43 x 10] (S3: data.frame)	A data.frame with 43 rows and 10 columns
Esophagus	list [43 x 10] (S3: data.frame)	A data.frame with 43 rows and 10 columns
Lung	list [43 x 10] (S3: data.frame)	A data.frame with 43 rows and 10 columns
Oral	list [43 x 10] (S3: data.frame)	A data.frame with 43 rows and 10 columns

i.e., it is a list of 5 data frames of the same size.

Why lists?

Well it has to do with two issues:

1. If we want to do the same operations over and over again, we use functions
2. Applying the same functions to many datasets is most efficient using lists and `purrr::map`. `map` inputs a list, applies a function to each member of the list, and then outputs the results in a list.

   Just like the `dplyr` pipelines start with a data frame and ends with a data frame, you can create a list-based pipeline using `map`, since it starts with a list and outputs a list.

To demonstrate:

```
dats <- map(dats, janitor::clean_names) # clean the names of the columns into snake_case
```

Here we use the function `clean_names` from the `janitor` package to clean the names of the columns to snake_case, without spaces and punctuation. This is an example of using a function with default options; the first argument of the function is fed each element of the list in turn, and the results are output into corresponding elements of a new list.

Next, we want to do the same munging operation on all 5 data sets. This operation will remove the first row of a dataset, and then ensure that all the columns are of type `numeric`. This can be implemented on the fly using an anonymous function (i.e. a function with no name) within the `map` function.

```
dats <- map(dats,
            function(d){
              d %>%
                slice(-1) %>% # Take away first column
                mutate_all(as.numeric) # Transform all columns to numeric
            })
```

```
## Warning: NAs introduced by coercion
```

The above warning happens since for black females, one of the data points is missing and is coded as `-` in the data; coercing this value to numeric results in a `NA`.

Next, for each site, we want to create separate data sets for men, women and both sexes. Once again, this is an instance of the same operation being performed on each component of the list `dats`, so `map` would be appropriate.

```
dats_both <- map(dats, select, year_of_diagnosis,
                 ends_with('sexes')) # Create both sexes
dats_male <- map(dats, select, year_of_diagnosis,
                 ends_with('_males')) # Create male datasets
dats_female <- map(dats, select, year_of_diagnosis,
                 ends_with('females')) # Create female datasets
```

Join the datasets into 1, by gender

Let's focus on the "both sexes" dataset first; we'll do the same operation on the male and female datasets (using a `for`-loop).

There are two ways to put these datasets together. One is to keep the data as is, and stack them, with an additional column denoting the site of the cancer associated with each row. This can be easily done from a list of data frames, provided that you have a **named list**.

```
names(dats_both)
```

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

Yes, we have names.

Stacking data sets, with a new column denoting the site

If we now want to stack all the data sets together, with a column denoting site of each record, we can do

```
dats_both_joined <- bind_rows(dats_both, .id = 'Site')
```

Here, the `.id` option creates "a new column of identifiers is created to link each row to its original data frame. The labels are taken from the named arguments to bind_rows(). When a list of data frames is supplied, the labels are taken from the names of the list. If no names are found a numeric sequence is used instead."

```
str(dats_both_joined)
```

```
## 'data.frame':    210 obs. of  5 variables:
##  $ Site                 : chr  "Brain" "Brain" "Brain" "Brain" ...
##  $ 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 ...
##  $ whites_both_sexes   : num  6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 6.49 ...
##  $ blacks_both_sexes   : num  4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 4.53 ...
```

Joining data sets into wide dataset, then use `gather`

An alternative to get to the same point is to use `left_join` successively to create a wide dataset. For this we would first need to distinguish the column names for each site. This is achieved by the following for loop:

```
for (nm in names(dats_both)){
  names(dats_both[[nm]]) <- str_replace(names(dats_both[[nm]]),
                                       'both_sexes', nm)
  names(dats_male[[nm]]) <- str_replace(names(dats_both[[nm]]),
                                            'males', nm)
  names(dats_female[[nm]]) <- str_replace(names(dats_both[[nm]]),
                                            'females', nm)
}
```

In this for loop, first `nm` takes the value `"Brain"`, and so for the Brain data, it replaces, for example, the string `"both_sexes"` in the names of the data with the string `"Brain"`. So that data set would have column names `year_of_diagnosis`, `all_races_Brain`, `whites_Brain` and `black_Brain`. The for loop then moves on to the next element, which is `"Colon"`, and replaces, in the recipe, the string `"Colon"` everywhere it seens `nm`. and so on.

Conceptually, for the homework, you could now have done the following pipe to join all the datasets together:

`brain %>% left_join(colon) %>% left_join(esophagus) %>% left_join(lung) %>% left_join(oral)`

This is great, if you can type out all the data sets. For more generality, lists and the function `Reduce` come in handy. `Reduce` uses a binary function (i.e., a function with two inputs) to successively combine the elements of a given vector or list. So it does the operation described above for arbitrary sizes of the list. Here our binary function will be `left_join` which takes two inputs, namely a left dataset and a right dataset.

```
dats2_both <- Reduce(left_join, dats_both)
dats2_male <- Reduce(left_join, dats_male)
dats2_female <- Reduce(left_join, dats_female)
```

Now we have a bit of data procesisng to do. We need to gather the data into long form, and split the gathered key column into site and race. These needs three steps:

1. gather the dataset into 3 columns (year and the other two)
2. Fix `all_races` to `allraces` so that `separate` can separate based on `_`.
3. Separate the variable into race and site variables

Since we'll be doing the same operation on all three datasets, we can write a function instead of copying-and-pasting.

```
f1 <- function(d){
  d %>%
  gather(variable, rate, -year_of_diagnosis) %>%
  mutate(variable = str_replace(variable,
                                'all_races',
                                'allraces')) %>%
  separate(variable, c('race','site'), sep='_')
}
```

Now we can do one of two things: either just do this separately for the three datasets:

```
dats3_both <- f1(dats2_both)
dats3_male <- f1(dats2_male)
dats3_female <- f1(dats2_female)
```

or use lists and maps

```
dats2 <- list('both' = dats2_both,
              'male' = dats2_male,
              'female' = dats2_female)

dats3 <- map(dats2, f1)
```

or, create a pipeline of lists using `map`:

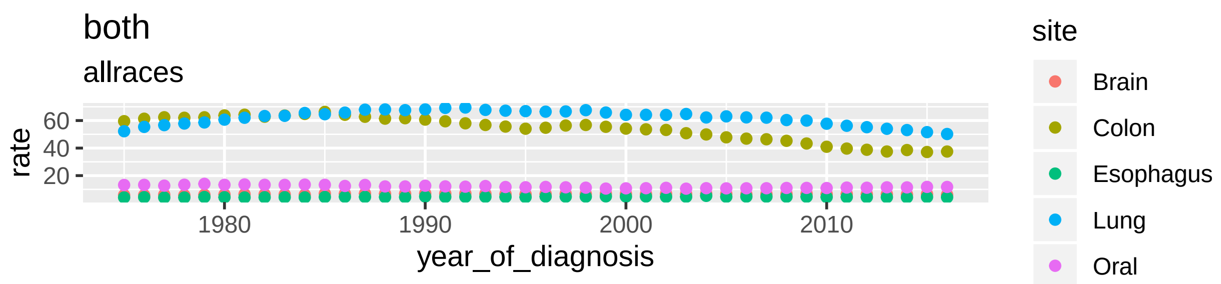
```
dats2_both %>%
  map(gather, variable, rate, -year_of_diagnosis) %>%
  map(mutate, variable = str_replace(variable, 'all_races','allraces')) %>%
  map(separate, variable, c('race','site'), sep='_')
```

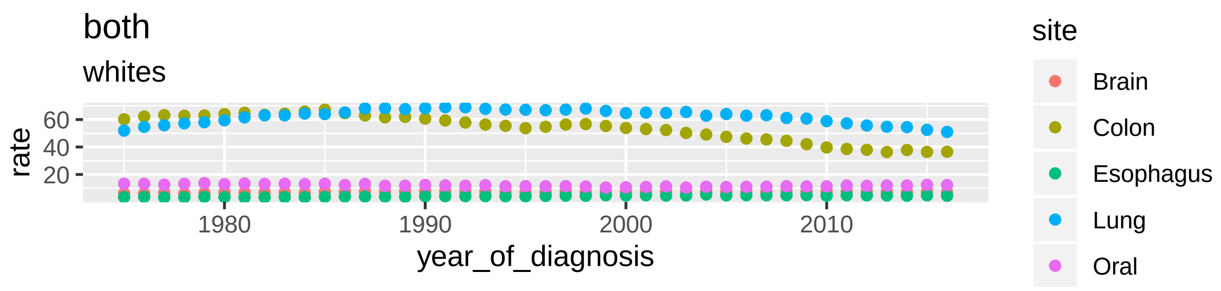
```
## Error in UseMethod("gather_"): no applicable method for 'gather_' applied to an object of class "c('
```

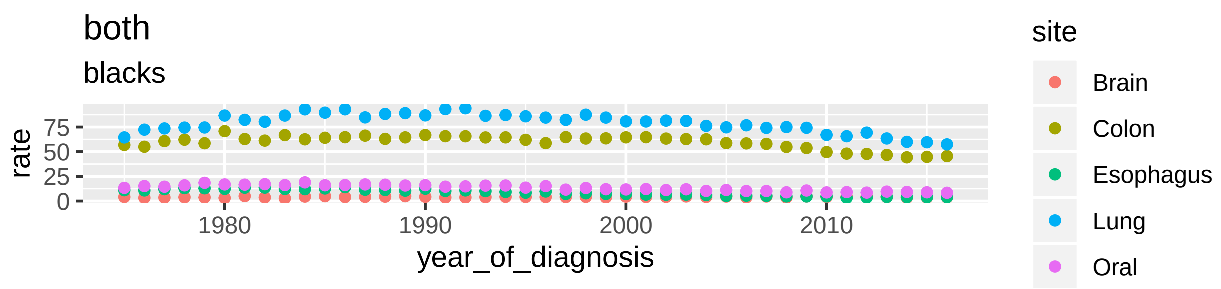
Note that the map pipeline is very similar to the pipeline in `f1`. Whereever we had `function(...)` in `f1`, we now have `map(function, ...)` where `function` is the function name and `...` are the function arguments.

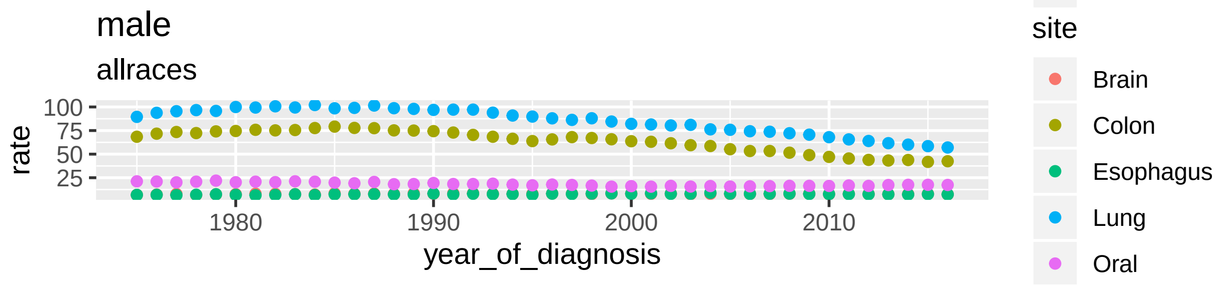
The last step is to create the graphs. I'm doing a simpler version where I just use a for-loop to create all the plots

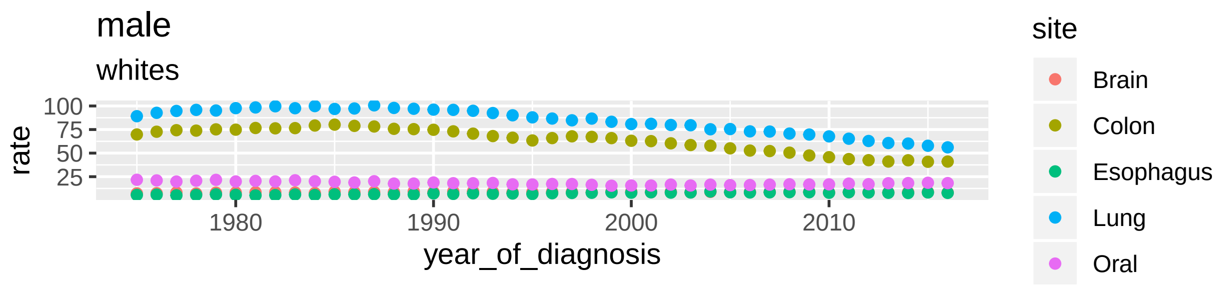
```
for(nm in names(dats3)){
  plt1 <- dats3[[nm]] %>%
    filter(race=='allraces') %>%
    ggplot(aes(x = year_of_diagnosis,
               y = rate,
               color = site))+
    geom_point() +
    labs(title = nm,
         subtitle='allraces')
  plt2 <- dats3[[nm]] %>%
    filter(race=='whites') %>%
    ggplot(aes(x = year_of_diagnosis,
               y = rate,
               color = site))+
    geom_point()+
    labs(title=nm, subtitle='whites')
  plt3 <- dats3[[nm]] %>%
    filter(race=='blacks') %>%
    ggplot(aes(x = year_of_diagnosis,
               y = rate,
               color = site))+
    geom_point()+
    labs(title=nm, subtitle='blacks')
  print(cowplot::plot_grid(plt1,plt2, plt3, ncol=1))
}
```

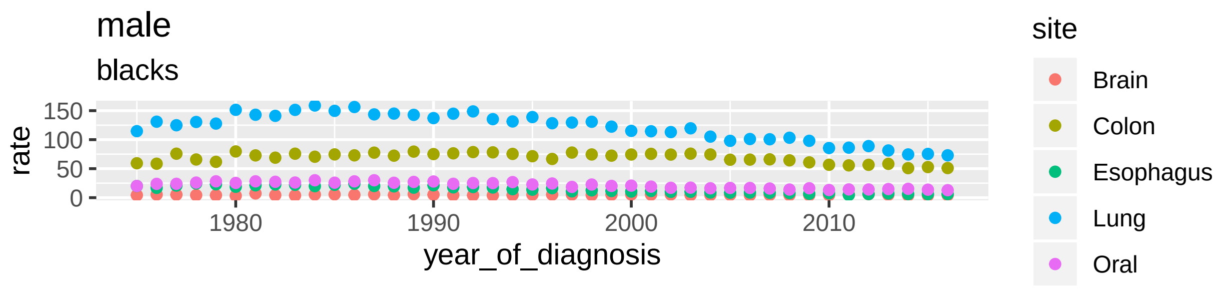

both
allraces
rate
60
40
20
1980
1990
2000
2010
year_of_diagnosis
site
Brain
Colon
Esophagus
Lung
Oral

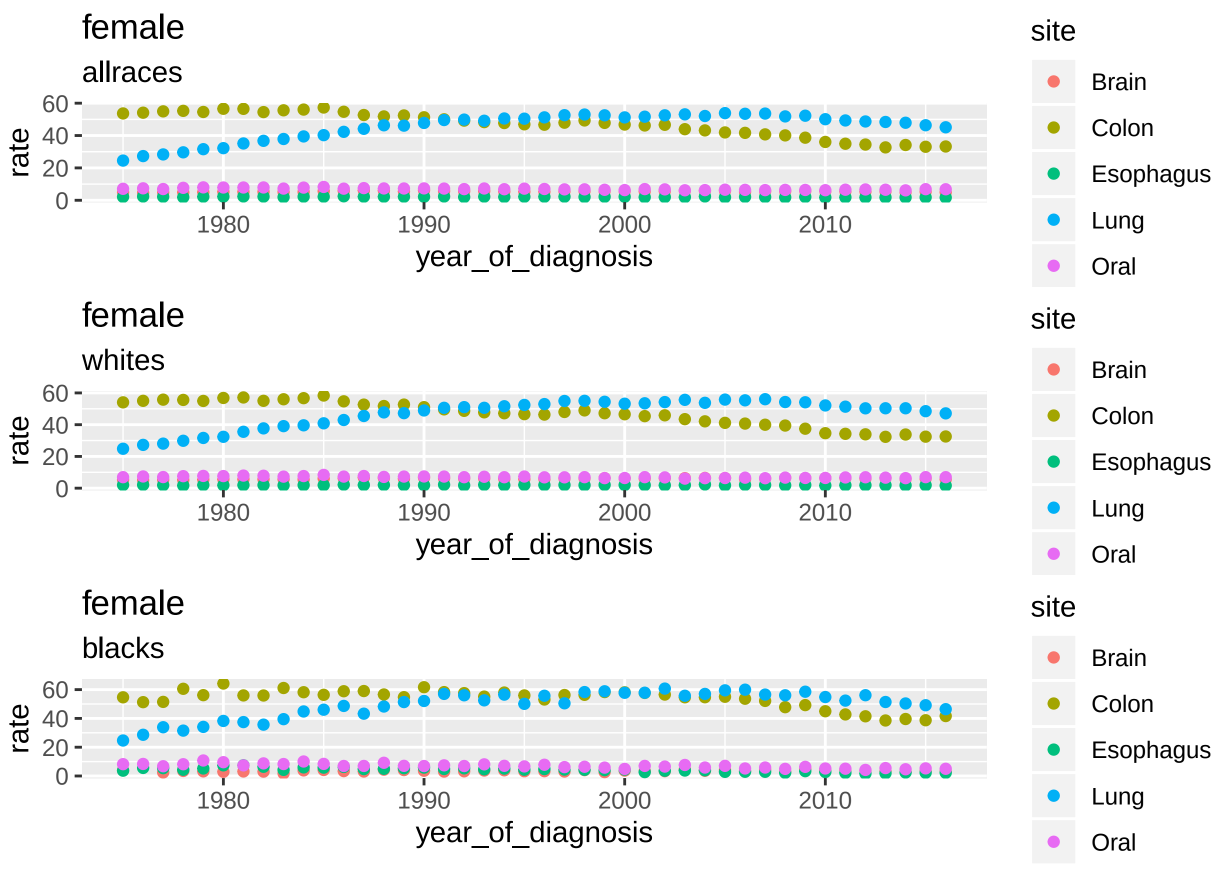

both
whites
rate
60
40
20
1980
1990
2000
2010
year_of_diagnosis
site
Brain
Colon
Esophagus
Lung
Oral


both
blacks
rate
75
50
25
0
1980
1990
2000
2010
year_of_diagnosis
site
Brain
Colon
Esophagus
Lung
Oral


male
allraces
rate
100
75
50
25
1980
1990
2000
2010
year_of_diagnosis
site
Brain
Colon
Esophagus
Lung
Oral

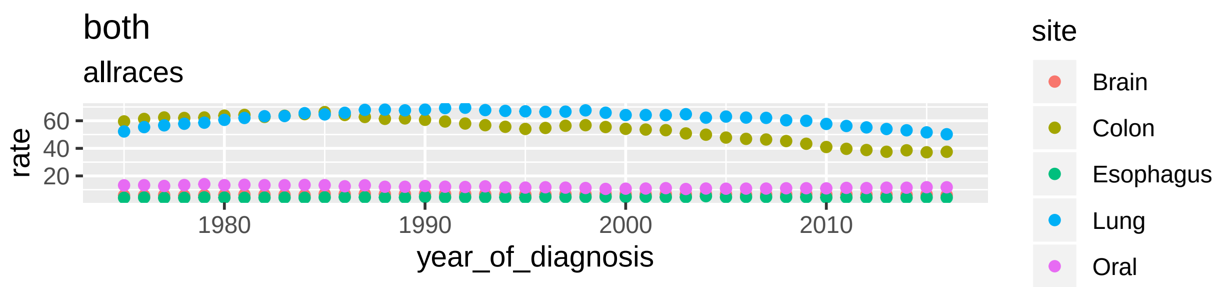

male
whites
rate
100
75
50
25
1980
1990
2000
2010
year_of_diagnosis
site
Brain
Colon
Esophagus
Lung
Oral

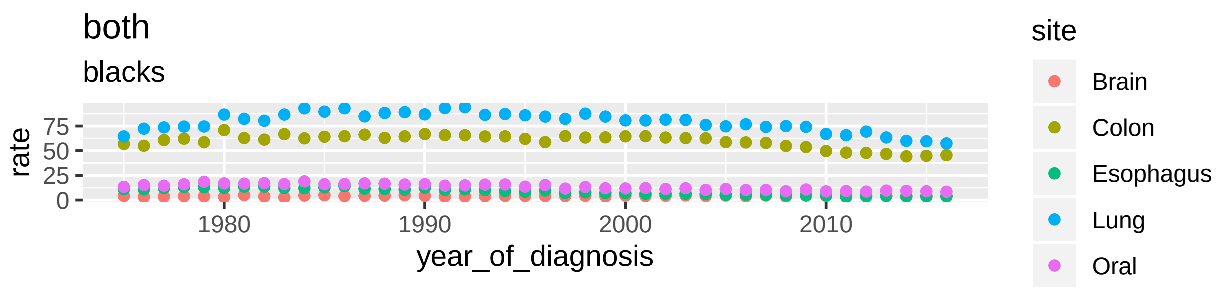

male
blacks
rate
150
100
50
0
1980
1990
2000
2010
year_of_diagnosis
site
Brain
Colon
Esophagus
Lung
Oral

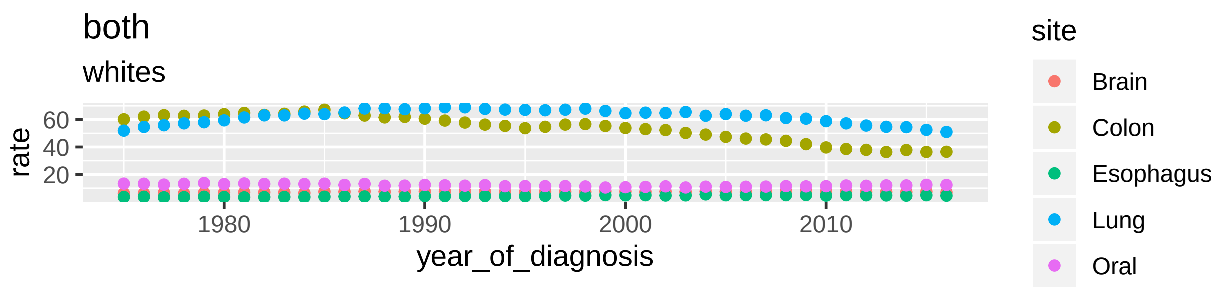


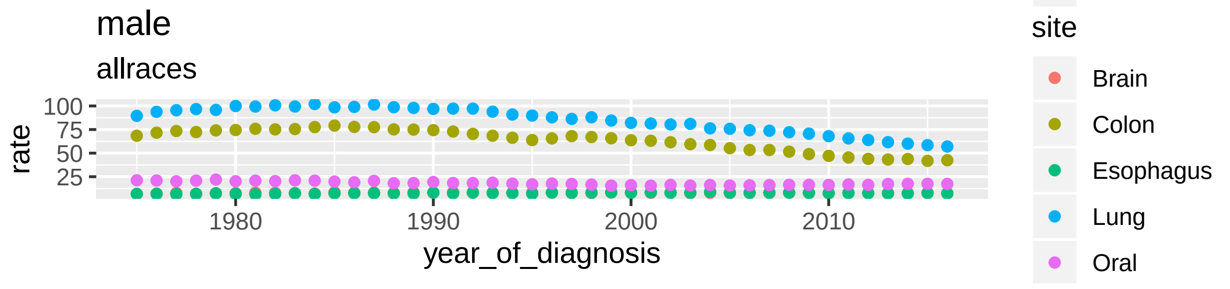
Using lists and plots, we might do this perhaps a bit more succintly. However it is up to you to decide which one is easier for you to understand (code should be human-readable)

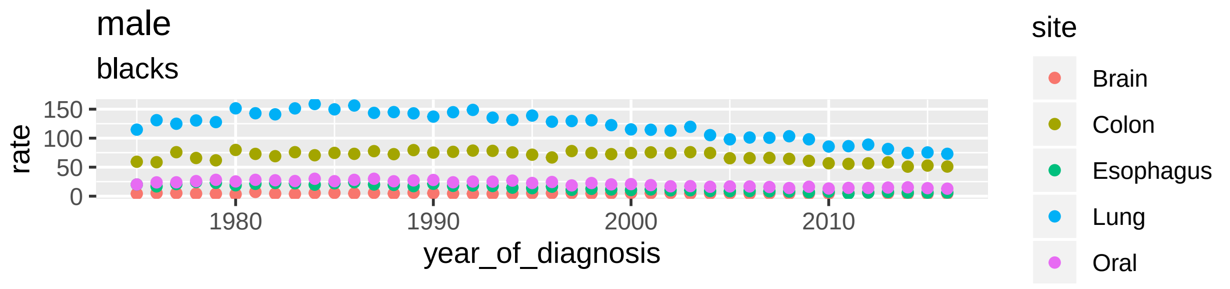
```r
for(nm in names(dats3)){
  plts <- dats3[[nm]] %>%
    group_split(race) %>% # Splits data into list of datasets based on values of race
    map(~ggplot(., aes(x = year_of_diagnosis,
                       y = rate,
                       color=site))+
          geom_point() +
          labs(title=nm, subtitle=unique(.$race)))
  print(cowplot::plot_grid(plotlist=plts, ncol=1))
}
```

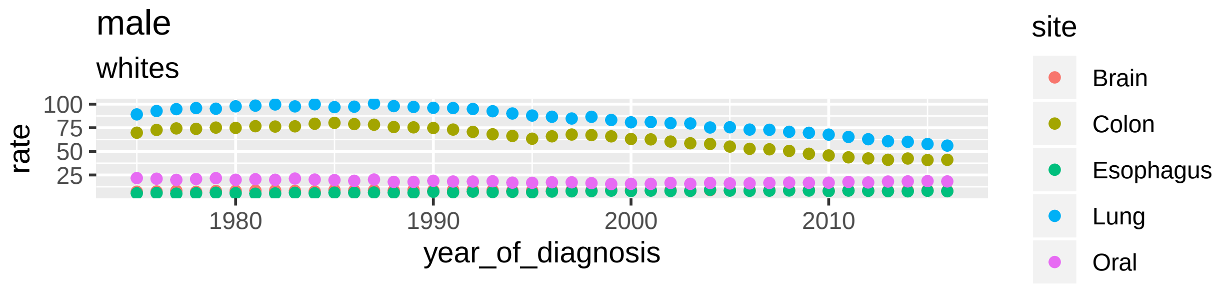

both
allraces
site
Brain
Colon
Esophagus
Lung
Oral
rate
60
40
20
1980
1990
2000
2010
year_of_diagnosis

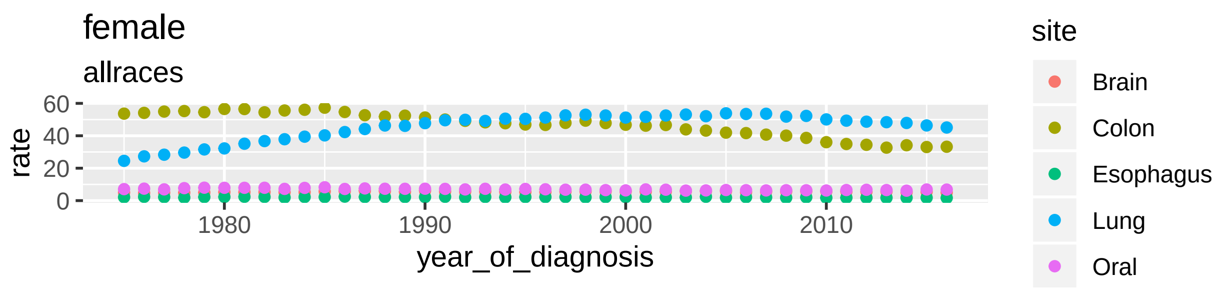

both
blacks
site
Brain
Colon
Esophagus
Lung
Oral
rate
75
50
25
0
1980
1990
2000
2010
year_of_diagnosis

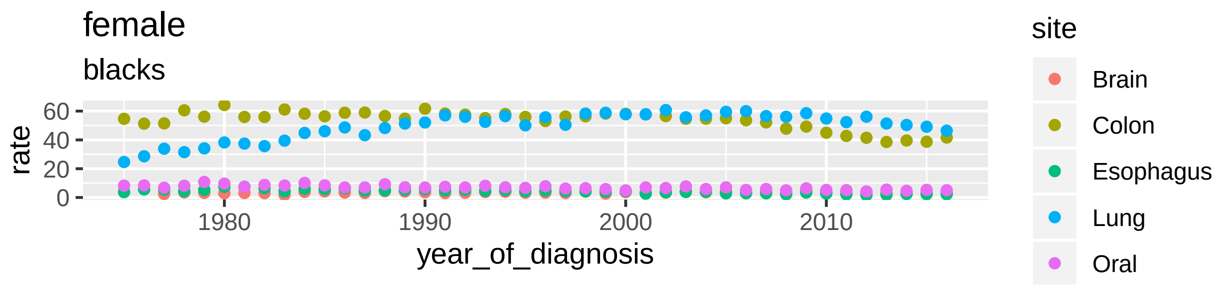

both
whites
site
Brain
Colon
Esophagus
Lung
Oral
rate
60
40
20
1980
1990
2000
2010
year_of_diagnosis

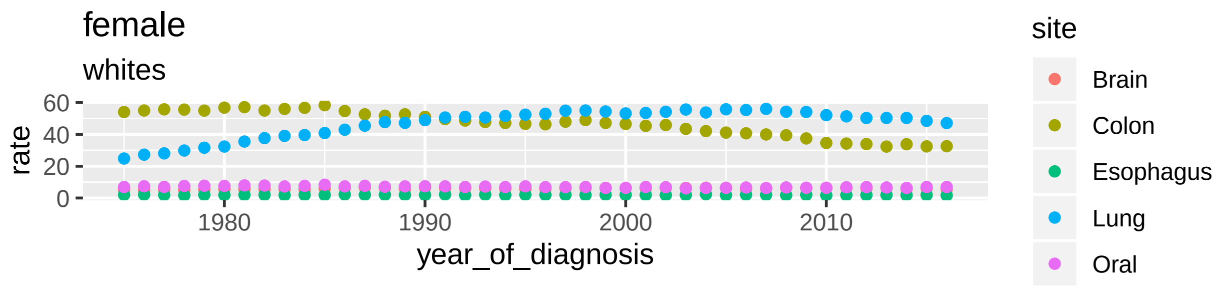

male
allraces
site
Brain
Colon
Esophagus
Lung
Oral
rate
100
75
50
25
1980
1990
2000
2010
year_of_diagnosis


male
blacks
site
Brain
Colon
Esophagus
Lung
Oral
rate
150
100
50
0
1980
1990
2000
2010
year_of_diagnosis


male
whites
site
Brain
Colon
Esophagus
Lung
Oral
rate
100
75
50
25
1980
1990
2000
2010
year_of_diagnosis







Summary

The coding strategy we've described here works well when you have a bunch of standardized datasets (formatted similarly) and you want to do the same operations to all of them to achieve some high-throughput computing. It requires both

1. Data sets in identical standard formats
2. The same common operations to be done to all of them