Lecture 4: Programming in R

Programming in R

- Pipes
- Functions
- Vectors
- Iteration

Pipes

- How I used to code (and sometimes still do)

```
x1=flights[flights$arr_time<1200,]
x2=aggregate(distance~tailnum,data=x1,FUN=sum)
x3=x2[order(-x2$distance),]
```

- Using pipes

```
flights %>% filter(arr_time<1200) %>%
        group_by(tailnum) %>%
        summarise(sum_dist=sum(distance)) %>%
        arrange(desc(sum_dist))
```

Pipes

- Hadley Wickham advice on when not to use them
  - Your pipes are longer than (say) ten steps. In that case, create intermediate objects with meaningful names. That will make debugging easier
  - You have multiple inputs or outputs. If there isn’t one primary object being transformed, but two or more objects being combined together, don’t use the pipe.

Functions

```
summ<-function(x,y){
  sm=x+y
  return(sm)
}
```

```
> summ(c(1,2))
> 3
```

Functions

- Hadley Wickham advice on functions advantage over copy and paste
  - You can give a function an evocative name that makes your code easier to understand.
  - As requirements change, you only need to update code in one place, instead of many.
  - You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

Naming functions

```
# Too short
f()

# Not a verb, or descriptive
my_awesome_function()

# Long, but clear
impute_missing()
collapse_years()
```

See R4DS for more discussion on R style conventions & recommendations

Environment

```
f <- function(x) {
 x + y
}
```

This will not return error, but instead will look for y globally in the environment

```
y=10
f(3)
>13
```

Can use functions in pipes

```
row_count<-function(df){
  nrow(df)
}
```

```
> planes %>% row_count()
[1] 3322
```

R functions written in C++

```
library(Rcpp)

cppFunction('int add(int x, int y, int z) {
  int sum = x + y + z;
  return sum;
}')

> add(1,2,3)
[1] 6
```

Vectors

- **Atomic** vectors, of which there are six types: **logical**, **integer**, **double**, **character**, **complex**, and **raw**. Integer and double vectors are collectively known as **numeric** vectors.

- **Lists**, which are sometimes called recursive vectors because lists can contain other lists.

```
> y=c(4,2,3)
> y[1]
[1] 4

> z=c("a","b","c")
> z[2]
[1] "b”

> ls=list(y,z)
> ls[[1]]
[1] 4 2 3
```

Iteration

```
summ<-function(x){
  sm=0

  for(i in 1:length(x)){
    sm=sm+x[i]
  }

  return(sm)
}
```

```
> summ(c(1,2,3,4))
> 10
```

purrr package (or alternatively apply in base R)

First figure out how to solve a problem for a single element of a list

Once you've solved that problem, purrr takes care of generalising your solution to every element in the list.

purrr package

```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
```

```
map_dbl(df, mean)
#>          a          b          c          d
#> -0.3260369  0.1356639  0.4291403 -0.2498034
map_dbl(df, median)
#>          a          b          c          d
#> -0.51850298  0.02779864  0.17295591 -0.61163819
map_dbl(df, sd)
#>        a        b        c        d
#> 0.9214834 0.4848945 0.9816016 1.1563324
```

apply (older, available in base R)

```
> df <- tibble(
    a = rnorm(10),
    b = rnorm(10),
    c = rnorm(10),
    d = rnorm(10)
+ )
```

```
> lapply(df,mean)
$a
[1] 0.2940624

$b
[1] 0.3139556

$c
[1] -0.2170487

$d
[1] -0.1184357
```

Summary

- First 3 weeks just to get you started on a data science programming language
- Optionally read units on wrangle and programming in r4ds to better your understanding (there are also advanced R books)
- You can python if you prefer for the modeling competition, but I will keep using R in class

In class exercise

- Write a function “name_split” that takes a string and splits it at the spaces and returns the first sub string (e.g. “Airbus Industry” -> “Airbus”)

- Next use map_chr to apply “name_split” to the manufacturer column of the planes data frame

- Write a function “square_sum” that takes a vector and sums the square of the entries. Try to write it in both R and C++ using Rcpp