Fun with functions and dplyr

Brian Wright

1/24/2020

Overview of Functions (Advanced R)

- Functions are at the core of R language, it's really a function based language

  "R, at its heart, is a"functional" language. This means that it has certain technical properties, but more importantly that it lends itself to a style of problem solving centred on functions." Hadley Wickham

What is a functional based language?

- Recently functions have grown in popularity because they can produce efficient and simple solutions to lots of problems. Many of the problems with performance have been solved.
- Functional programming compliments object oriented programming

What makes a programming approach "functional"?

- Functions can behave like any other data structure
  - Assign them to variables, store to lists, pass them as aurguments to other functions, create them inside functions and even produce a function as a result of a funcion
- Functions need to be "pure" meaning that if you call it again with the same inputs you get the same results. sys.time() not a "pure" function
- The execution of the function shouldn't change global variables, have no side effects.

Functions

- Function don't have to be "pure" but it can help to ensure your code is doing what you intend it to do.
- Functional programming helps to break a problem down into it's pieces. When working to solve a problem it helps to divide the code into individually operating functions that solve parts of the problem.

Types of Functions

In \ Out	Vector	Function
Vector	Regular function	Function factory
Function	Functional	Function operator

Figure 1: Function Types

Let's Build a Function

- Basically recipes composed of series of R statements

```
name <- funtion(variables){
  #In here goes the series of R statements
}
```

Example, talk out the steps

```
my_mean <- function(x){
  Sum <- sum(x)#Here we are using a function
  #inside a function!
  N <- length(x)
  return(Sum/N)#return is optional but helps with
  #clarity on some level.
}
```

Create a little list and pass it to the function and see if it works. Also call the Sum and N variables. . . does this work?

Functional - Will show later, Function Factory (Advanced R)

```
power1 <- function(exp) {
  function(x) {
    x ^ exp
  }
}

#Assigning the exponentials
square <- power1(2)
cube <- power1(3)
```

Run the Created Functions

```
square(3)
```

```
>   [1] 9
```

```
cube(3)
```

```
>   [1] 27
```

Quick Exercise

Create a function that computes the range of a variable and then for no good reason adds 100 and divides by 10. Write out the steps you would need first in Pseudocode, then develop the function.

dplyr verbs in the tidyverse

The `dplyr` package gives us a few verbs for data manipulation

Function	Purpose
select	Select columns based on name or position
mutate	Create or change a column
filter	Extract rows based on some criteria
arrange	Re-order rows based on values of variable(s)
group_by	Split a dataset by unique values of a variable
summarize	Create summary statistics based on columns

`select`

You can select columns by name or position, of course.

You can also select columns based on some criteria, which are encapsulated in functions.

- starts_with("***"), ends_with("***"), contains("_____")
- one_of("_____","_____","_____")

There are others; see `help(starts_with)`.

Example

Load the `weather.csv`. This contains daily temperature data in 2010 for some location.

```
>  [1] "C:/Users/Brian Wright/Documents/git_3001/DS-4001/2_R_f
```

```
head(weather, 2)
```

```
>  # A tibble: 2 x 35
>    id     year month element    d1    d2    d3    d4    d5 
>    <chr> <int> <int> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <
>  1 MX17~  2010     1 tmax       NA    NA    NA    NA    NA 
>  2 MX17~  2010     1 tmin       NA    NA    NA    NA    NA 
>  # ... with 23 more variables: d9 <lgl>, d10 <dbl>, d11 <dbl
>  #   d13 <dbl>, d14 <dbl>, d15 <dbl>, d16 <dbl>, d17 <dbl>,
>  #   d19 <lgl>, d20 <lgl>, d21 <lgl>, d22 <lgl>, d23 <dbl>,
>  #   d25 <dbl>, d26 <dbl>, d27 <dbl>, d28 <dbl>, d29 <dbl>,
```

How would you just select the columns with the daily data?

```
select(weather, starts_with("d"))
```

mutate

`mutate` can either transform a column in place or create a new column in a dataset

We'll use the in-built `mpg` dataset for this example, We'll select only the city and highway mileages. To use this selection later, we will need to assign it to a new name

```
mpg1 <- select(mpg, cty, hwy)
```

mutate

We'll change the city and highway mileage to km/l from mpg. This will involve multiplying it by 1.6 and dividing by 3.8

```
head(mutate(mpg1, cty = cty * 1.6 / 3.8,
                  hwy = hwy * 1.6/3.8), 5)
```

```
>  # A tibble: 5 x 2
>      cty   hwy
>    <dbl> <dbl>
>  1  7.58  12.2
>  2  8.84  12.2
>  3  8.42  13.1
>  4  8.84  12.6
>  5  6.74  10.9
```

This is in-place replacement

New Variable Defined

```
mutate(mpg1, cty1 = cty * 1.6/3.8, hwy1 = hwy * 1.6/3.8)
```

```
>  # A tibble: 234 x 4
>       cty   hwy  cty1  hwy1
>     <int> <int> <dbl> <dbl>
>   1    18    29  7.58  12.2
>   2    21    29  8.84  12.2
>   3    20    31  8.42  13.1
>   4    21    30  8.84  12.6
>   5    16    26  6.74  10.9
>   6    18    26  7.58  10.9
>   7    18    27  7.58  11.4
>   8    18    26  7.58  10.9
>   9    16    25  6.74  10.5
>  10    20    28  8.42  11.8
>  # ... with 224 more rows
```

This creates new variables

filter

`filter` extracts rows based on criteria

```
filter(mpg, cyl == 4)
```

```
>  # A tibble: 81 x 11
>     manufacturer model     displ  year   cyl trans   drv
>     <chr>        <chr>     <dbl> <int> <int> <chr>   <chr> <i
>   1 audi         a4          1.8  1999     4 auto(l~ f
>   2 audi         a4          1.8  1999     4 manual~ f
>   3 audi         a4          2    2008     4 manual~ f
>   4 audi         a4          2    2008     4 auto(a~ f
>   5 audi         a4 quat~    1.8  1999     4 manual~ 4
>   6 audi         a4 quat~    1.8  1999     4 auto(l~ 4
>   7 audi         a4 quat~    2    2008     4 manual~ 4
>   8 audi         a4 quat~    2    2008     4 auto(s~ 4
>   9 chevrolet    malibu      2.4  1999     4 auto(l~ f
>  10 chevrolet    malibu      2.4  2008     4 auto(l~ f
>  # ... with 71 more rows
```

Practice Piping

```
admit_df <- read_csv("~/git_3001/DS-4001/data/LogReg.csv")
str(admit_df)
```

```
>  tibble [400 x 4] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
>   $ admit: num [1:400] 0 1 1 1 0 1 1 0 1 0 ...
>   $ gre  : num [1:400] 380 660 800 640 520 760 560 400 540 7
>   $ gpa  : num [1:400] 3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.3
>   $ rank : num [1:400] 3 3 1 4 4 2 1 2 3 2 ...
>   - attr(*, "spec")=
>     .. cols(
>     ..   admit = col_double(),
>     ..   gre = col_double(),
>     ..   gpa = col_double(),
>     ..   rank = col_double()
>     .. )
```

```
#Do we notice anything that seems a bit off.
```

Coercion num to factor

```
admit_df$rank <- as.factor(admit_df$rank)
#changes rank to a factor
```

Five Basic Classes in R

- character
- numeric (double precision floating point numbers, default)
- integer (subset of numeric)
- complex (j = 10 + 5i)
- logical (True/False)

All have coercion calls (example from: R Nuts and Bolts)

```
x <- 0:6
class(x)#why
```

```
>  [1] "integer"
```

```
as.numeric(x)
```

```
>  [1] 0 1 2 3 4 5 6
```

```
as.logical(x)
```

```
>  [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
as.character(x)
```

```
>  [1] "0" "1" "2" "3" "4" "5" "6"
```

Functional Example: Pass a function get a vector

We can also convert multiple columns using lapply(), great example of functional orientation of R.

```
names <- c("admit","rank")
#using names as a index on admit_df,
admit_df[,names] <- lapply(admit_df[,names], factor)

#Check class of those two variables
(as.character(meta_fun <- lapply(subset(admit_df,
                                        select = names),
                                 class)))
```

```
>  [1] "factor" "factor"
```

#using a functional with two functions inside that creates a object coerced to a character list. . . what fun.

Using the code chunk below to "group_by" rank

Using the code chunk below to filter by 1 in the admit column

Ok now summarise by average GPA

Now Pipe everything together