Function, Pipe, and Map

Programming with Tidyverse

Strongway

23 May 2018

Tidyverse

- We all know powerful of `tidyverse`
- A typical process

```
mtcars %>% group_by(gear, carb) %>%
  summarise(wt = mean(wt)) -> mt

fig1 = ggplot(mt, aes(x = gear, y = wt,
        group = carb, color = carb)) +
      geom_point() + geom_line()
```

Pros and Cons of Basic approach

- Pros
  - Grammar-like language
  - Easy to change and add layers
  - nice plots, quick and faster
- cons
  - For each analysis you need similar codes
  - Many redundant codes, prone to error

We need some adavanced tricks!

Replace table in ggplot

Some times we have multiple experiments, similar settings and plotting. The only difference is initial input table.

- %+% operator to replace the table in ggplot

```
mtcars %>% group_by(gear, carb, cyl) %>%
  summarise(wt = mean(wt)) -> mt2
# you have fig1 already, use it
fig2 = fig1 %+% mt2 + facet_wrap(~cyl)
```

Flexibility in dplyr functions

- variables in `dplyr` are usually non-standard evaluation.

```r
filter(df, x==1, y == 2)
# this means
df[df$x==1 & df$y == 2, ]
```

- This makes it difficult to use variables in `dplyr`

```r
# you have two groups that you want to summarize
df %>% group_by(g1) %>% summarise(a = mean(a))
df %>% group_by(g2) %>% summarise(a = mean(a))

# You want to use a variable, but it won't work
my_var <- g1  # or my_var = "g1"
df %>% group_by(my_var) %>% summarise(a = mean(a))
```

Make inputs working

1. We need to **quote** the input ourselves, using `quo()` or `quos()`
2. Tell `dplyr` function we have already quote, using !!

```
quo(g1)
```

```
## <quosure>
##   expr: ^g1
##   env:  global
```

```
quos(a1,a2)
```

```
## [[1]]
## <quosure>
##   expr: ^a1
##   env:  global
##
```

define a function

- Now let's create a 'flexible' function

```
# define a function, inputs that we quote ourselves
mySummary = function(df, my_var, mean_var) {
  df %>% group_by(!! my_var) %>%
    summarise(m = mean(!!mean_var))
}
# use this function
df1 = mySummary(mtcars, quo(gear),quo(wt))
df2 = mySummary(mtcars, quo(cyl), quo(hp))
```

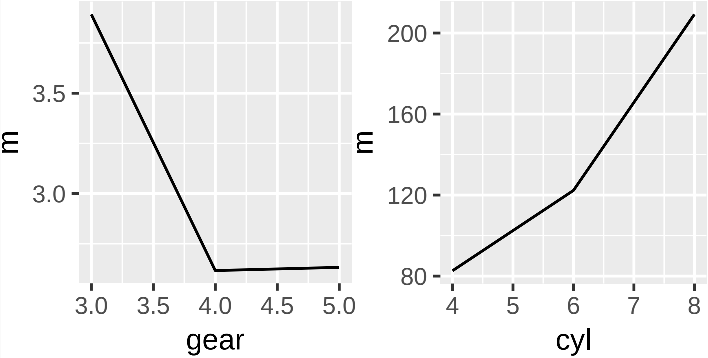
Improving the function

- quo is literally quote the variable
- enquo uses some dark magic to look at the argument, see what the user typed, and return that value as a quosure.

```
# define a function
mySummary = function(df, grp, mvar) {
  quo_var1 = enquo(grp)
  quo_var2 = enquo(mvar)
  df %>% group_by(!! quo_var1) %>%
    summarise(m = mean(!!quo_var2))
}
# Now you can remove the quo()
df1 = mySummary(mtcars, gear,wt)
df2 = mySummary(mtcars, cyl, hp)
```

Now use your function in pipes

```
mySummary(mtcars,gear,wt) %>% ggplot(aes(gear, m)) + geom_l
mySummary(mtcars,cyl,hp) %>% ggplot(aes(cyl, m)) + geom_lin
```



further improvement on the function

- We make the inputs flexible, but what about output variables?
- This needs two new tricks
  - `quo_name()` to convert the input expression to a string
  - := to help to define the new name output

refine the function

```
mySummary = function(df, grp, mvar) {
  quo_var1 = enquo(grp)
  quo_var2 = enquo(mvar)
  quo_out = quo_name(enquo(mvar))
  df %>% group_by(!! quo_var1) %>%
    summarise(!!quo_out := mean(!!quo_var2))
}
mySummary(mtcars,cyl, hp)
```

```
## # A tibble: 3 x 2
##     cyl    hp
##   <dbl> <dbl>
## 1    4.  82.6
## 2    6. 122.
## 3    8. 209.
```

Nested tibble

- Unlike standard data.frame, tibble table can have complex structure, such as list nested in one column
  - nest()
  - unnest()

```
library(gapminder)
gapminder %>% group_by(continent, country) %>% nest() -> gr
head(gn,3)
```

```
## # A tibble: 3 x 3
##   continent country     data
##   <fct>     <fct>       <list>
## 1 Asia      Afghanistan <tibble [12 x 4]>
## 2 Europe    Albania     <tibble [12 x 4]>
## 3 Africa    Algeria     <tibble [12 x 4]>
```

Why do we need nested table?

- calculate multiple values

```
probs = c(0.1,0.25, 0.5, 0.75, 0.9)
gapminder %>% group_by(continent) %>%
  summarise(p = list(probs),
            q = list(quantile(lifeExp))) %>%
  unnest()-> gn

head(gn,3)
```

```
## # A tibble: 3 x 3
##   continent     p     q
##   <fct>     <dbl> <dbl>
## 1 Africa    0.100  23.6
## 2 Africa    0.250  42.4
```

Why do we need nested table?

- individual modelling

```
country_model <- function(df) {
  lm(lifeExp ~ year, data = df)
}
gapminder %>% nest(-country) %>%
  mutate(model = map(data, country_model)) -> gm
head(gm,3)
```

```
## # A tibble: 3 x 3
##   country     data              model
##   <fct>       <list>            <list>
## 1 Afghanistan <tibble [12 x 5]> <S3: lm>
## 2 Albania     <tibble [12 x 5]> <S3: lm>
## 3 Algeria     <tibble [12 x 5]> <S3: lm>
```

purrr map functions

- map(your_list, your_function)

```
map(c(9,16,25),sqrt) %>% unlist()
```

```
## [1] 3 4 5
```

- map_df(list, function) return data.frame structure

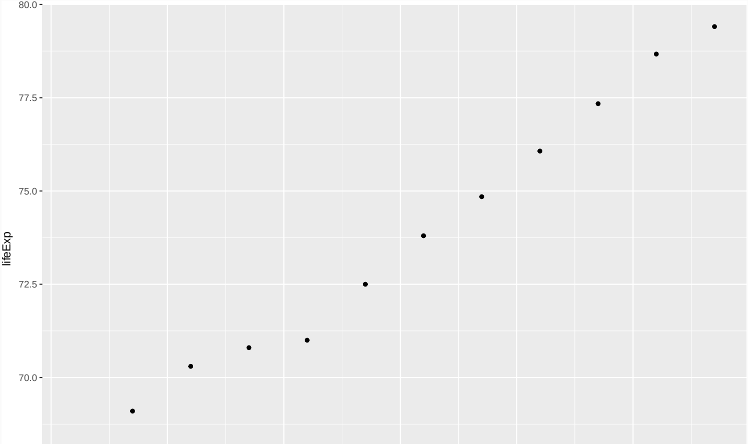
```
# suppose you have readData(filename) function
files = list.files('data') # list all raw files
data = map_df(files, readData) # read each file, and combi
```

- map2 can have two input lists

Back to modeling

- First to build a model on a typical data set

```
df = gapminder %>% filter(country == 'Germany')
ggplot(df, aes(year, lifeExp)) + geom_point()
```



Extracting model parameters

- broom::glance() retrieve key information

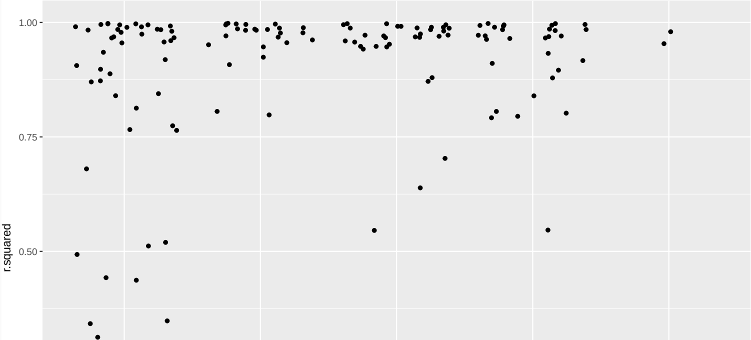
```
broom::glance(model)
```

```
##   r.squared adj.r.squared     sigma statistic     p.valu
## 1 0.9895057     0.9884563 0.4160796  942.8966 3.14615e-1
##         AIC     BIC deviance df.residual
## 1 16.82158 18.2763 1.731222          10
```

Putting together

```
gapminder %>% group_by(continent, country) %>% nest() %>%
  mutate(model = map(data, country_model)) %>%
  mutate(glance = map(model, broom::glance)) %>%
  unnest(glance) -> gm

ggplot(gm, aes(continent, r.squared)) + geom_jitter()
```


r.squared