hw04-activity5-data-manipulation-cheat-sheet-thibodeau-mylinh

My Linh Thibodeau
2017-10-07

```
suppressPackageStartupMessages(library(tidyverse))
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
knitr::opts_chunk$set(fig.width=12, fig.height=9)
library(knitr)
library(kableExtra)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
options(knitr.table.format = "html")
```

Essentials of data manipulations in R

Matching tasks and functions

Task	tidyr/dplyr function	reshape2 function	base R
1. Group	group_by	"cast"	"aggregate(~ group)"

Task	tidyr/dplyr function	reshape2 function	base R
2. Split	spread, ?separate	dcast	table
3. Stack	gather	melt	"stack"
4. Join	_join functions	reshape::merge_all	merge, cbind, rbind
5. Subset	filter, extract	NA	df[which()] or subset
6. Transpose	gather/spread	dcast/melt	data.frame(t())
7. Sort	arrange		

- *Note1. I have put in quotation marks the functions that I do not recommend for the specific task, because although there are some ways around it, they are no the simplest/fastest way to get the solution in my opinion.*
- *Note2. In reshape2, acast() is for a array/vector/matrix output, and dcast for a data.frame output. I have put dcast for simplicity in the table, but acast can also be used.*
- *Note3. When a backslash separate 2 functions in the table (e.g. gather/spread), it mean that they may need to be combined to complete the Task.*

Let's make a small table to illustrate the different funcitons.

```
d1 <- read.table("scratch-space/small_dataframe_example.txt", sep = "\t", header = TR
d1 %>%
  kable("html") %>% kable_styling()
```

gene.type	gene	expression	copy.number
TS	TP53	very.low	0
TS	FLCN	average	4
TS	SBHD	very.low	3
TS	RB1	low	1
ONC	KRAS	very.high	5
ONC	EGFR	high	0

(1) GROUP

Objective: group gene.type and copy.state, when count the number of rows (pairs) and the sum of the copy.number for each pair.

Tidyr

```
gr_tidy <- select(d1, gene.type, expression, copy.number) %>%
  group_by(gene.type, expression) %>%
  summarize(Nrows = n(), SumCol.copy.number = sum(copy.number))
gr_tidy %>% kable("html") %>% kable_styling()
```

gene.type	expression	Nrows	SumCol.copy.number
ONC	high	1	0
ONC	very.high	1	5
TS	average	1	4
TS	low	1	1
TS	very.low	2	3

Reshape

```
gr_resh <- d1 %>%
  select(gene.type, expression, copy.number) %>%
  dcast(gene.type+expression ~ expression) %>%
  melt(varnames = c("gene.type", "expression"), value.name = "Nrows", na.rm =TRUE) %>
  filter(Nrows !=0) %>%
  select(gene.type, expression, Nrows)


## Using copy.number as value column: use value.var to override.

## Aggregation function missing: defaulting to length

## Using gene.type, expression as id variables


gr_resh %>% kable("html") %>% kable_styling()
```

gene.type	expression	Nrows
TS	average	1
ONC	high	1
TS	low	1
ONC	very.high	1
TS	very.low	2

Note. Using reshape is a very convoluted way to do group data, Tidyr is better for these data manipulations.

```
aggregate(gene.type~expression, d1, length)  %>% kable("html") %>% kable_styling()
```

expression	gene.type
average	1
high	1
low	1
very.high	1
very.low	2

```
aggregate(. ~expression, d1, length)  %>% kable("html") %>% kable_styling()
```

expression	gene.type	gene	copy.number
average	1	1	1
high	1	1	1
low	1	1	1
very.high	1	1	1
very.low	2	2	2

```
tapply(d1$gene.type, d1$expression, FUN=length)
```

```
##   average      high       low very.high  very.low
##         1         1         1         1         2
```

Note. Again, base R is not ideal to pursue grouping. I was not able to obtain the wanted table, and since Tidyr offers better alternative, I won't pursue further base R for grouping.

(2) SPLIT

Objective: transform the Nrows column (count of gene.type-expression pairs) of gr_tidy so that each group (TS, ONC) is represented by one row, each expression category by one column, and the numbers (Nrows of gr_tidy) represent the number of occurence in each pair (gene.type-expression).

Tidyr

```
split_tidy <- gr_tidy %>%
  group_by(gene.type, expression) %>%
  select(gene.type, Nrows) %>%
  spread(key=expression, value = Nrows)
```

```
## Adding missing grouping variables: `expression`
```

```
split_tidy %>% kable("html") %>% kable_styling()
```

gene.type	average	high	low	very.high	very.low
ONC	NA	1	NA	1	NA
TS	1	NA	1	NA	2

Reshape

```
split_resh <- gr_tidy %>%
  dcast(gene.type~expression, value.var = "Nrows")
split_resh %>% kable("html") %>% kable_styling()
```

gene.type	average	high	low	very.high	very.low
ONC	NA	1	NA	1	NA
TS	1	NA	1	NA	2

Base R

```
split_R <- with(d1, table(gene.type, expression))
split_R %>% kable("html") %>% kable_styling()
```

	average	high	low	very.high	very.low
ONC	0	1	0	1	0
TS	1	0	1	0	2

(3) STACK

Objective: display split_resh in a way such that all possible gene.type-expression pairs are displayed in col1 and col2, which col3 returns the count of such pairs (Nrows)

Tidyr

```
stack_tidy <- split_resh %>%
  gather(key = variable, value = expression, very.low, low,  average, very.high, high
stack_tidy %>% kable("html") %>% kable_styling()
```

gene.type	variable	expression
ONC	very.low	NA
TS	very.low	2
ONC	low	NA
TS	low	1
ONC	average	NA
TS	average	1

gene.type	variable	expression
ONC	very.high	1
TS	very.high	NA
ONC	high	1
TS	high	NA

Reshape

```
stack_resh <- melt(split_resh, id="gene.type") %>%
  arrange(gene.type)
stack_resh %>% kable("html") %>% kable_styling()
```

gene.type	variable	value
ONC	average	NA
ONC	high	1
ONC	low	NA
ONC	very.high	1
ONC	very.low	NA
TS	average	1
TS	high	NA
TS	low	1
TS	very.high	NA
TS	very.low	2

Or also:

```
stack_resh2 <- d1 %>% dcast(gene.type~expression) %>% melt()
```

```
## Using copy.number as value column: use value.var to override.
```

```
## Aggregation function missing: defaulting to length

## Using gene.type as id variables
```

```
stack_resh2 %>% kable("html") %>% kable_styling()
```

gene.type	variable	value
ONC	average	0
TS	average	1
ONC	high	1
TS	high	0
ONC	low	0
TS	low	1
ONC	very.high	1
TS	very.high	0
ONC	very.low	0
TS	very.low	2

```
stack(split_resh)
```

```
## Warning in stack.data.frame(split_resh): non-vector columns will be ignored

##    values       ind
## 1      NA   average
## 2       1   average
## 3       1      high
## 4      NA      high
## 5      NA       low
## 6       1       low
## 7       1 very.high
## 8      NA very.high
## 9      NA  very.low
## 10      2  very.low
```

Note. Base R can provide the number of possible gene.type-expression pairs, but will only return two columns: values (concatenation of vectors in d1) and ind (factor from which the vector originated in d1), so it's not a very intuitive method.

Let's prepare another dataset (d2) for the following exercises.

```
d2 <- read.table("scratch-space/small_dataframe_example_2.txt",sep = "\t", header = T
d2 %>% kable("html") %>% kable_styling()
```

gene	IHC
TP53	absent
FLCN	normal
SBHD	absent
RB1	absent
KRAS	strong
EGFR	normal
PALB2	normal

(4) JOIN

Objective: join d1 and d2 according to the common column gene.

Tidyr

Only one joining example below, but for the complete dplyr join functions cheatsheet, please go HERE !

left_join()

```
join_tidy <- left_join(d1, d2, by = "gene")
```

```
## Warning: Column `gene` joining factors with different levels, coercing to
## character vector
```

```
join_tidy %>% kable("html") %>% kable_styling()
```

gene.type	gene	expression	copy.number	IHC
TS	TP53	very.low	0	absent
TS	FLCN	average	4	normal
TS	SBHD	very.low	3	absent
TS	RB1	low	1	absent
ONC	KRAS	very.high	5	strong
ONC	EGFR	high	0	normal

Base R

```
join_R <- merge(d1, d2)
join_R %>% kable("html") %>% kable_styling()
```

gene	gene.type	expression	copy.number	IHC
EGFR	ONC	high	0	normal
FLCN	TS	average	4	normal
KRAS	ONC	very.high	5	strong
RB1	TS	low	1	absent
SBHD	TS	very.low	3	absent
TP53	TS	very.low	0	absent

(5) SUBSET

Objective: only take the data for the ONC gene.type.

Tidyr

```
subset_tidy <- d1 %>%
  filter(gene.type == "ONC")
subset_tidy %>% kable("html") %>% kable_styling()
```

gene.type	gene	expression	copy.number
ONC	KRAS	very.high	5
ONC	EGFR	high	0

```
subset_resh <- d1 %>% dcast(gene.type~expression) %>% melt(subset = .(gene.type=="ONC
```

```
## Using copy.number as value column: use value.var to override.

## Aggregation function missing: defaulting to length

## Using gene.type as id variables
```

```
View(subset_resh)
```

Base R

```
subset_R <- d1[which(d1$gene.type== 'ONC'),]
subset_R %>% kable("html") %>% kable_styling()
```

	gene.type	gene	expression	copy.number
5	ONC	KRAS	very.high	5
6	ONC	EGFR	high	0

```
subset_R2 <- subset(d1, gene.type=='ONC')
subset_R2 %>% kable("html") %>% kable_styling()
```

	gene.type	gene	expression	copy.number

	gene.type	gene	expression	copy.number
5	ONC	KRAS	very.high	5
6	ONC	EGFR	high	0

(5) TRANSPOSE

Objective: take the subset_tidy horizontal table and transpose it to a vertical position such that the rows become columns and vice versa.

Base R

```
transpose_R <- data.frame(t(subset_tidy))
transpose_R %>% kable("html") %>% kable_styling()
```

	X1	X2
gene.type	ONC	ONC
gene	KRAS	EGFR
expression	very.high	high
copy.number	5	0

I note that the columns are called X1 and X2 instead of ONC and ONC. Oh well, we get the general idea anyway !

(7) SORT

Objective: sort the table according to ascending copy.number

Tidyr

```
join_tidy %>%
  arrange(copy.number) %>% kable("html") %>% kable_styling()
```

gene.type	gene	expression	copy.number	IHC

gene.type	gene	expression	copy.number	IHC
TS	TP53	very.low	0	absent
ONC	EGFR	high	0	normal
TS	RB1	low	1	absent
TS	SBHD	very.low	3	absent
TS	FLCN	average	4	normal
ONC	KRAS	very.high	5	strong

REFERENCES/RESOURCES

- Reshape2 manual here
- Reshape2 intro here
- Reshape reference here, here
- Blog on data manipulation here on r-statistics, and here on Oregon University.
- Panda can also offer options, as exemplified here, but I ran out of time to look into this further.
- Aggregate in base R here or here or here Base R stack resource here and here
- Base R subset