STAT 545 HW 04

Elijah Willie

October 8, 2018

Introduction

Data Reshaping Prompts (and relationship to aggregation)

Activity - Make your own cheatsheet

I will be making a cheatsheet about tidying, splitting and applying functions to data. For this I will be using two separate datasets. First I will demonstrate tidying using the billboards top 100 dataset, and second I will demonstrate splitting and applying using a dataset from my previous university

Data Description

The billboards dataset contains data sets regarding songs on the Billboard Hot 100 list in the year 2000. The fields include, the artist, the time it entered, the time it peaked, and many other interesting features. However, the data is quite messy, so I will be using tidyr to mak the data cleaner.

Load the required libraries and data

```
library(tidyverse) # loads dplyr, ggplot2, tidyr, etc
```

```
## Warning: package 'tidyverse' was built under R version 3.4.3
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
## Warning: package 'tibble' was built under R version 3.4.3
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```
## Warning: package 'purrr' was built under R version 3.4.4
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
## Warning: package 'stringr' was built under R version 3.4.4
```

```
## Warning: package 'forcats' was built under R version 3.4.4
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(dplyr)
```

load in the required datasets

```
#load in the billboard dataset
bill_board.data <- read.csv("billboard.csv",stringsAsFactors = FALSE)

#load in the sfu statistics dataset
load("scilong.RData")
```

Get a feel for the datasets

```
knitr::kable(head(bill_board.data))
```

year	artist.inverted	track	time	genre	date.entered	date.peaked	x1st.v
2000	Destiny's Child	Independent Women Part I	3:38	Rock	2000-09-23	2000-11-18	
2000	Santana	Maria, Maria	4:18	Rock	2000-02-12	2000-04-08	
2000	Savage Garden	I Knew I Loved You	4:07	Rock	1999-10-23	2000-01-29	
2000	Madonna	Music	3:45	Rock	2000-08-12	2000-09-16	
2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	3:38	Rock	2000-08-05	2000-10-14	
2000	Janet	Doesn't Really Matter	4:17	Rock	2000-06-17	2000-08-26	

```
knitr::kable(head(scilong))
```

Subject	CrsNum	CreditHrs	semester	enrollment	FTEs	year
ACMA	210	3	1077	51	5.10000	2008
ACMA	335	3	1077	20	2.00000	2008
ACMA	425	3	1077	22	2.20000	2008
ACMA	465	3	1077	16	1.60000	2008
ACMA	490	3	1077	4	0.40000	2008
BISC	100	4	1077	176	23.46667	2008

Structure of the billboard dataset(taken from online)

- Columns `year` through `date.peaked` describe the song, then `x1st.week` through `x76th.week` are the chart positions for the first through 76th weeks.
  - If a song is on the chart for less than 76 weeks, its position is `NA` for any missing weeks.
- Weeks are not variables, they are the time data for the time series.

Looking at the billboard dataset, it seems pretty messy. So lets tidy it up a bit.

- Main idea is gather the rankings in the different weeks into a `rank` variable.
- Before gathering, may need to select/rename some of the variables.
- After gathering, will create some new variables and sort the data frame.

I will select and rename some of the variables

- I don't think I will need `time` or `genre`, as it does not seem relevant for my cause.
  - Use the `select()` function from `dplyr`. You can us `-` to de-select a column
- Rename `artist.inverted` as it does not seem too appropriate.
  - `rename()` from `dplyr` takes arguments of the form `newname = oldname`

```
bill_board.data <-
  bill_board.data %>% select(-time,-genre) %>%
  rename(artist = artist.inverted)

knitr::kable(head(bill_board.data))
```

year	artist	track	date.entered	date.peaked	x1st.week	x2nd.wee
2000	Destiny's Child	Independent Women Part I	2000-09-23	2000-11-18	78	6
2000	Santana	Maria, Maria	2000-02-12	2000-04-08	15	
2000	Savage Garden	I Knew I Loved You	1999-10-23	2000-01-29	71	
2000	Madonna	Music	2000-08-12	2000-09-16	41	
2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	2000-08-05	2000-10-14	57	
2000	Janet	Doesn't Really Matter	2000-06-17	2000-08-26	59	

Now that looks somewhat better.

use the gather function tidying up the weeks column in the dataset

```
# gather (data, key, value, ... ) where ... are the columns to collapse
bill_board.datalong <- gather(bill_board.data,week,rank,x1st.week:x76th.week,na.rm=TRUE)
knitr::kable(head(bill_board.datalong))
```

year	artist	track	date.entered	date.peaked	week	rank
2000	Destiny's Child	Independent Women Part I	2000-09-23	2000-11-18	x1st.week	78
2000	Santana	Maria, Maria	2000-02-12	2000-04-08	x1st.week	15
2000	Savage Garden	I Knew I Loved You	1999-10-23	2000-01-29	x1st.week	71
2000	Madonna	Music	2000-08-12	2000-09-16	x1st.week	41
2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	2000-08-05	2000-10-14	x1st.week	57
2000	Janet	Doesn't Really Matter	2000-06-17	2000-08-26	x1st.week	59

Now that is looking much better.

Finally I will be doing some final touch ups that will incorporate the use of fuctions such as `parse_number()`, `mutate()`, `as.date()`, and `arrange()`.

- Extract week numbers from `week` variable using the `parse_number()` function.
- Coerce `date.entered` to a `Date` object using the `as.date()` function.
- Calculate the date of each ranking based on the date it entered the charts and the week.
- Remove the date each ranking was entered since it is not relevant anymore.
- Sort the resulting dataset on artist, track and week using the `arrange()` function.

```
bill_board.data <-
  bill_board.datalong %>% mutate(week = parse_number(week), #parse the week column
              date = as.Date(date.entered) + 7*(week-1)) %>% #compute the date object
  select(-date.entered) %>% # don't need date.entered anymore
  arrange(artist,track,week) #sort the resulting dataset
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
knitr::kable(head(bill_board.data))
```

year	artist	track	date.peaked	week	rank	date
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	2000-03-11	1	87	2000-02-26
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	2000-03-11	2	82	2000-03-04
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	2000-03-11	3	72	2000-03-11
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	2000-03-11	4	77	2000-03-18
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	2000-03-11	5	87	2000-03-25
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	2000-03-11	6	94	2000-04-01

Now that just looks perfect. We started with a super messy dataset, and now we have tidied it up to something that looks a lot cleaner and less cluttered while retaining all of the relevant informations.

Join Prompts (join, merge, look up)

- In this section I will be exploring the various ways to merge two set of data. I will be comparing functions provided by base R and others provided in the tidyverse library. Particularly, I will be exploring inner joins, and outer joins. I will be using a synthetic dataset. I will also be analyzing the match function provided in base R.

Explore the base R function merge(), which also does joins. Compare and contrast with dplyr joins.

Generate synthetic datasets for illustrations

```
#define two dataframes I will be using for my analyses
#define the station dataframe
STATION <- data.frame(ID=c(13,44,66),
    City = c("Phoenix","Denver","Caribou"),
    State = c("AZ","CO","ME"),
    Lat_N = c(33,40,47),
    Long_W = c(112,105,68))
#define the stats dataframe
STATS <- data.frame(row = 1:6,
    ID = c(13,13,44,44,66,66),
    Month = c(1,7,1,7,1,7),
    Temp_F = c(57.4,91.7,27.3,74.8,6.7,65.8),
    Rain_I = c(0.31,5.15,0.18,2.11,2.1,4.52))

#create a singe dataframes for downstream illustrations
miami <- data.frame(ID=77,City="Miami",State="FL",Lat_N=26,Long_W=80)
temp <- data.frame(row=7, ID=89, Month=4, Temp_F=100, Rain_I=3.5)


#view the airlines dataframe
knitr::kable(head(STATION))
```

ID	City	State	Lat_N	Long_W
13	Phoenix	AZ	33	112
44	Denver	CO	40	105
66	Caribou	ME	47	68

```
#view the flights dataframe
knitr::kable(head(STATS))
```

row	ID	Month	Temp_F	Rain_I
1	13	1	57.4	0.31
2	13	7	91.7	5.15
3	44	1	27.3	0.18
4	44	7	74.8	2.11
5	66	1	6.7	2.10
6	66	7	65.8	4.52

- Add miami to the station dataframe

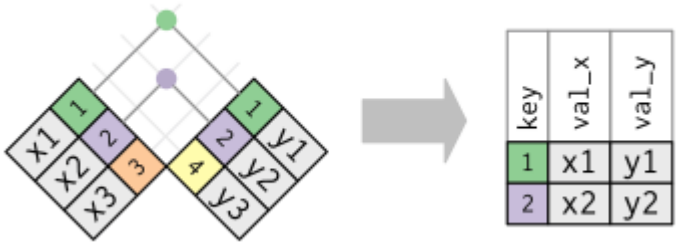
```
STATION <- rbind(STATION,miami)
```

- Add temp to STATs dataframe

```
STATS <- rbind(STATS,temp)
```

Inner joins

- An inner join is quite simple in that it matches pairs of observations whenever their keys are equal.



Refer to figure below

- I will be demonstrating inner joins using dplyr and base R

```
#join the tables using dplyr functions
my.data <- inner_join(STATION, STATS, by = "ID")

knitr::kable(my.data)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52

- Now do the same as above, but instead use base r functions

```
my.data.new <- merge(STATION, STATS, by = "ID")
knitr::kable(head(my.data.new))
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52

- Comparing the tables returned from using dplyr vs base R, we see that they are the same. The `merge()` of base R is equilvalent to the `inner_join()` function of dplyr.
- Also note that inner join is only done on fields that match in both datasets, so we do not see miami in the resulting table.

Outer joins

- Outer joins is comprised of a set of three types of joins. Particularly left_join, right_join and full_join.
- Left join keeps all the observations in x
- Right join keeps all the observations in y
- Full join all the observations in both x and y
- Refer to figure below

- Use dplyr functions to perform an left join between the station table and the stats table on ID.

```
#use dplyr to perform left join
data.left <- left_join(STATION, STATS, by = "ID")
knitr::kable(data.left)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52
77	Miami	FL	26	80	NA	NA	NA	NA

- Do the left join using base R `merge()` function

```
#perform a left merge using base R
data.left.new <- merge(STATION,STATS,by="ID",all.x=TRUE)
knitr::kable(data.left.new)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52
77	Miami	FL	26	80	NA	NA	NA	NA

- Again we see that both methods returns the same result.

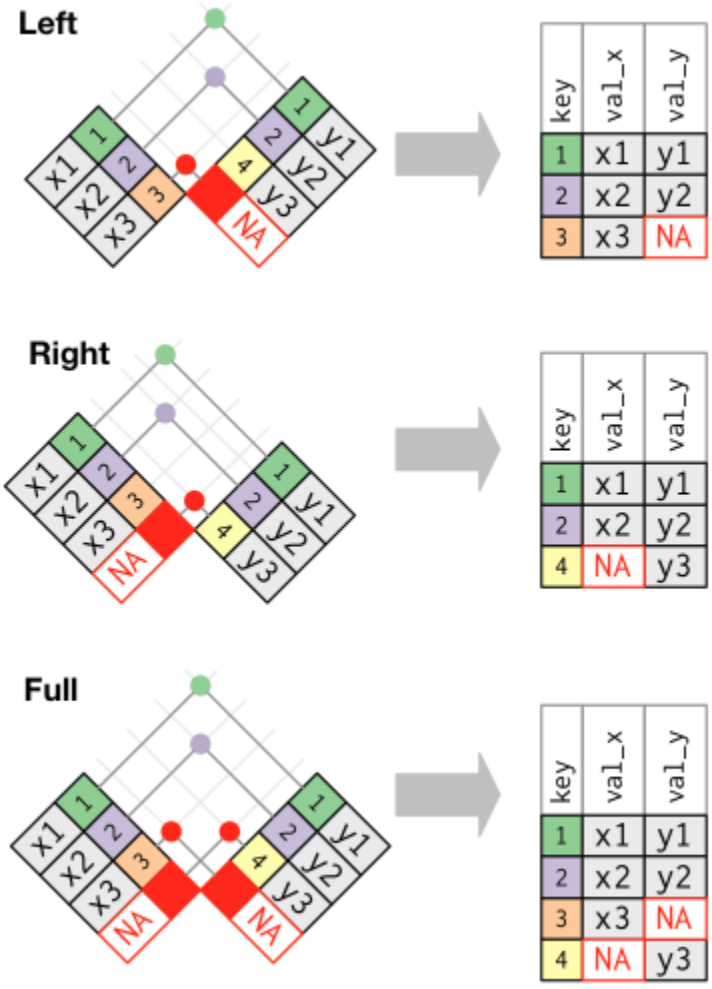

Left
x1
x2
x3
y1
y2
y3
NA
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA
Right
x1
x2
x3
NA
y1
y2
y3
key
val_x
val_y
1 x1 y1
2 x2 y2
4 NA y3
Full
x1
x2
x3
NA
NA
y1
y2
y3
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA
4 NA y3

Figure 1:

- The only difference between the two is that dpylr has an explicit left merge function where as with base R, it is an extra parameter passed in.
- Use dplyr functions to perform an right join between the station table and the stats table on ID.

```
#use dplyr to peform a right join
data.right <- right_join(STATION,STATS,by="ID")
knitr::kable(data.right)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52
89	NA	NA	NA	NA	7	4	100.0	3.50

- Now do the same thing using the `merge()` in base R.

```
data.right.new <- merge(STATION,STATS,by="ID",all.y=TRUE)
knitr::kable(data.right.new)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52
89	NA	NA	NA	NA	7	4	100.0	3.50

- Again we see that both methods returns the same result.
- The only difference between the two is that dpylr has an explicit left merge function where as with base R, it is an extra parameter passed in.
- Use dplyr functions to perform an full join between the station table and the stats table on ID.

```
data.full <- full_join(STATION,STATS,by="ID")
knitr::kable(data.full)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52
77	Miami	FL	26	80	NA	NA	NA	NA
89	NA	NA	NA	NA	7	4	100.0	3.50

- Now do the same thing using the `merge()` in base R.

```
data.full.new <- merge(STATION,STATS,by="ID", all = TRUE)
knitr::kable(data.full.new)
```

ID	City	State	Lat_N	Long_W	row	Month	Temp_F	Rain_I
13	Phoenix	AZ	33	112	1	1	57.4	0.31
13	Phoenix	AZ	33	112	2	7	91.7	5.15
44	Denver	CO	40	105	3	1	27.3	0.18
44	Denver	CO	40	105	4	7	74.8	2.11
66	Caribou	ME	47	68	5	1	6.7	2.10
66	Caribou	ME	47	68	6	7	65.8	4.52
77	Miami	FL	26	80	NA	NA	NA	NA
89	NA	NA	NA	NA	7	4	100.0	3.50

- In summary, both dplyr and base R provides functions that enables us to perform various sorts of joins on our datasets. The only difference is that, with dplyr we have eplicit functions for each type of joins and with base R, it is all packaged into a single `merge()` function. The packages may be different, but the functionalities remains the same. Also, using different types of joins enables us some flexibility with how we would like to restructure our dataset

Explore the base R function match(), which is related to joins and merges, but is really more of a "table lookup". Compare and contrast with a true join/merge.

- The match function in base R essentially checks for the first occurence of a dataframe in another dataframe. If the condition is satisfied, it returns the index of the value, or else it returns NA.
- I will demonstrate with an example

```
#match the occurence of ID STATION vs STATS
result <- match(STATION$ID, STATS$ID)

knitr::kable(result)
```

x
1
3
5
NA

- The above results show us that STATION and STATS ID match on fields 1, 3, and 5 Respectively. We also see that there is one non matching which is returned as `NA`.
- Another thing to note is that the length of the result vector will be equal to the length of the query vector. In our case, the length of the result vector is equal to the length of the STATION dataframe.
- Finally, to compare the match function with the merge/join function. I think the match function is a subroutine of the merge/join function. It finds all the row or indices where the join condition is true and the merge/join function then combines both datframes based on these indices.
- Essentially, one could implement their own merge/join function by calling the match function on a field bewteen two dataframes and then concatinating the rows where the fields match.