Code

R Notebook

Mike Kozlowski

Dplyr and sqldf Examples

These are both alternative ways of filtering or searching r dataframes

Dplyr is a unique set of tools

sqldf is SQL for data frames, so you can use SQL commands on an r dataframe

#Data Pliers

Data Pliers (dplyr) is a R package for manipulating data.

It was largely written by Hadly Wickham, who was head of the organization that produced RStudio and GGPlot2

GGplot2 was based on a “grammar of graphics”, which is to say, a form of expressing ideas about how graphic visualizations of data are stated and formed.

dplyr is based on a “grammar of data manipulation”

There are “native” ways in R to do all the manipulation tasks that are carried out in dplyr. The “Native R” approaches are very similar to “Native Python” approaches. The syntax is a bit different, but the general ideas are the same and they work pretty much the same way. The python Pandas package works much like Native R.

I think there are some advantages to dplyr, it does some things very well. In other cases, I find “Native R” to work better.

Personally, I do not use dplyr. It takes time to learn, and it is R only. The native R and Python approaches work just fine.

But, in some subcultures, dplyr is very popular, so you need to have some exposure to it.

If you like it, go ahead and use it.

Tidyverse

Hadley Wickham has had a bunch of ideas about processing data, a lot of them are very interesting (as is dplyrs). They are packaged up into a library called tidyverse, so you can just download them all at once.

See

https://dplyr.tidyverse.org/ (https://dplyr.tidyverse.org/)

Basic Operations in dplyr

mutate() adds new variables that are functions of existing variables select() picks variables based on their names. filter() picks cases based on their values. summarise() reduces multiple values down to a single summary. arrange() changes the ordering of the rows.

Hide

```
require("dplyr")
```

```
Loading required package: dplyr

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union
```

Hide

```
data(starwars)
head(starwars)
```

name <chr>	height <int>	m… <dbl>	hair_color <chr>	skin_color <chr>	eye_color <chr>	birth_year <dbl>	sex <chr>
Luke Skywalker	172	77	blond	fair	blue	19.0	male
C-3PO	167	75	*NA*	gold	yellow	112.0	none
R2-D2	96	32	*NA*	white, blue	red	33.0	none
Darth Vader	202	136	none	white	yellow	41.9	male
Leia Organa	150	49	brown	light	brown	19.0	female
Owen Lars	178	120	brown, grey	light	blue	52.0	male

6 rows | 1-8 of 14 columns

This example data set is in the form of a tibble. A tibble is like a data frame except that it is stricter in how it handles data, so it sort of a more reliable dataframe

-you cannot change variable names or data types of columns -it creates errors if you search on a variable that does not exist -it does not allow recycling of variables

dplyr filtering

Hide

```
starwars %>% filter(species=="Droid")
```

name <chr>	height <int>	m… <dbl>	hair_color <chr>	skin_color <chr>	eye_color <chr>	birth_year <dbl>	sex <chr>	gender <chr>
C-3PO	167	75	*NA*	gold	yellow	112	none	masculine
R2-D2	96	32	*NA*	white, blue	red	33	none	masculine
R5-D4	97	32	*NA*	white, red	red	*NA*	none	masculine
IG-88	200	140	none	metal	red	15	none	masculine
R4-P17	96	*NA*	none	silver, red	red, blue	*NA*	none	feminine

name <chr>	height <int>	m… <dbl>	hair_color <chr>	skin_color <chr>	eye_color <chr>	birth_year <dbl>	sex <chr>	gender <chr>
BB8	*NA*	*NA*	none	none	black	*NA*	none	masculine

6 rows | 1-9 of 14 columns

The tibble starwars was pipelined into a filter operation, that selected for species equal to Droid. The pipeline symbol is %>% and it passes a tibble through a series of operations in dplyr

The pipeline operator idea is really a big part of unix and linux, and you may see it later in python for some more advanced operations. You can link togehter a bunch of operations with a pipeline

The native R version of this filter operation, just for comparison

Hide

```
starwars[starwars$species=="Droid",]
```

name <chr>	height <int>	m… <dbl>	hair_color <chr>	skin_color <chr>	eye_color <chr>	birth_year <dbl>	sex <chr>	gender <chr>
C-3PO	167	75	*NA*	gold	yellow	112	none	masculine
R2-D2	96	32	*NA*	white, blue	red	33	none	masculine
R5-D4	97	32	*NA*	white, red	red	*NA*	none	masculine
IG-88	200	140	none	metal	red	15	none	masculine
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
R4-P17	96	*NA*	none	silver, red	red, blue	*NA*	none	feminine
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
BB8	*NA*	*NA*	none	none	black	*NA*	none	masculine
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*

1-10 of 10 rows | 1-9 of 14 columns

We can pipeline two filters

Hide

```
starwars%>%filter(species=="Droid")%>%filter(height>100)
```

name <chr>	height <int>	ma… <dbl>	hair_color <chr>	skin_color <chr>	eye_color <chr>	birth_year <dbl>	sex <chr>	gender <chr>
C-3PO	167	75	*NA*	gold	yellow	112	none	masculine
IG-88	200	140	none	metal	red	15	none	masculine

2 rows | 1-9 of 14 columns

In base R

Hide

```
starwars[(starwars$species=="Droid")&(starwars$height>100),]
```

name <chr>	height <int>	ma… <dbl>	hair_color <chr>	skin_color <chr>	eye_color <chr>	birth_year <dbl>	sex <chr>	gender <chr>
C-3PO	167	75	*NA*	gold	yellow	112	none	masculine
IG-88	200	140	none	metal	red	15	none	masculine
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*

7 rows | 1-9 of 14 columns

#dplyr select

filters variables by name

This allows some interesting combinations

Hide

```
starwars%>%select(name,ends_with("color"))
```

name <chr>	hair_color <chr>	skin_color <chr>	eye_color <chr>
Luke Skywalker	blond	fair	blue
C-3PO	*NA*	gold	yellow
R2-D2	*NA*	white, blue	red
Darth Vader	none	white	yellow
Leia Organa	brown	light	brown
Owen Lars	brown, grey	light	blue
Beru Whitesun lars	brown	light	blue
R5-D4	*NA*	white, red	red
Biggs Darklighter	black	light	brown
Obi-Wan Kenobi	auburn, white	fair	blue-gray

1-10 of 87 rows Previous **1** 2 3 4 5 6 … 9 Next

In base R we would do this

Hide

```
ctargets=c("name","hair_color","skin_color","eye_color")
starwars[,ctargets]
```

name <chr>	hair_color <chr>	skin_color <chr>	eye_color <chr>
Luke Skywalker	blond	fair	blue
C-3PO	*NA*	gold	yellow
R2-D2	*NA*	white, blue	red
Darth Vader	none	white	yellow
Leia Organa	brown	light	brown
Owen Lars	brown, grey	light	blue
Beru Whitesun lars	brown	light	blue
R5-D4	*NA*	white, red	red
Biggs Darklighter	black	light	brown
Obi-Wan Kenobi	auburn, white	fair	blue-gray

1-10 of 87 rows Previous **1** 2 3 4 5 6 … 9 Next

Hide

```
NA
```

#dplyr mutate

This adds a new, computed column to the data

Note this set of commands creates bmi and then selects a few columns

Hide

```
starwars %>%
  mutate(name, bmi = mass / ((height / 100)  ^ 2)) %>%
  select(name:mass, bmi)
```

name <chr>	height <int>	mass <dbl>	bmi <dbl>
Luke Skywalker	172	77.0	26.02758
C-3PO	167	75.0	26.89232
R2-D2	96	32.0	34.72222
Darth Vader	202	136.0	33.33007
Leia Organa	150	49.0	21.77778
Owen Lars	178	120.0	37.87401
Beru Whitesun lars	165	75.0	27.54821
R5-D4	97	32.0	34.00999
Biggs Darklighter	183	84.0	25.08286
Obi-Wan Kenobi	182	77.0	23.24598

1-10 of 87 rows Previous 1 2 3 4 5 6 ... 9 Next

We cannot simply add a column to a tibble, we have use mutate

We can add columns to a dataframe

Hide

```
df_starwars<-as.data.frame(starwars)
df_starwars["bmi"]<-df_starwars$mass/(df_starwars$height/100)^2
df_starwars[,c("name","height","mass","bmi"),]
```

name <chr>	**height** <int>	**mass** <dbl>	**bmi** <dbl>
Luke Skywalker	172	77.0	26.02758
C-3PO	167	75.0	26.89232
R2-D2	96	32.0	34.72222
Darth Vader	202	136.0	33.33007
Leia Organa	150	49.0	21.77778
Owen Lars	178	120.0	37.87401
Beru Whitesun lars	165	75.0	27.54821
R5-D4	97	32.0	34.00999
Biggs Darklighter	183	84.0	25.08286
Obi-Wan Kenobi	182	77.0	23.24598

1-10 of 87 rows Previous 1 2 3 4 5 6 ... 9 Next

Hide

```
NA
```

#arrange()

sort or order data

This has the ability to use complicated orderings that are hard to do in base R

This is a simple calculation

Hide

```
starwars %>%
  arrange(desc(mass))
```

name <chr>	**height** <int>	**mass** <dbl>	**hair_color** <chr>	**skin_color** <chr>
Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown
Grievous	216	159.0	none	brown, white
IG-88	200	140.0	none	metal

name <chr>	height <int>	mass <dbl>	hair_color <chr>	skin_color <chr>
Darth Vader	202	136.0	none	white
Tarfful	234	136.0	brown	brown
Owen Lars	178	120.0	brown, grey	light
Bossk	190	113.0	none	green
Chewbacca	228	112.0	brown	unknown
Jek Tono Porkins	180	110.0	brown	fair
Dexter Jettster	198	102.0	none	brown

1-10 of 87 rows | 1-5 of 14 columns Previous **1** 2 3 4 5 6 … 9 Next

Hide

```
# create an ordering index based on mass
temp=order(df_starwars$mass,decreasing=TRUE)

df_starwars[temp,]
```

	name <chr>	height <int>	mass <dbl>	hair_color <chr>	skin_color <chr>
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown
77	Grievous	216	159.0	none	brown, white
22	IG-88	200	140.0	none	metal
4	Darth Vader	202	136.0	none	white
78	Tarfful	234	136.0	brown	brown
6	Owen Lars	178	120.0	brown, grey	light
23	Bossk	190	113.0	none	green
13	Chewbacca	228	112.0	brown	unknown
18	Jek Tono Porkins	180	110.0	brown	fair
68	Dexter Jettster	198	102.0	none	brown

1-10 of 87 rows | 1-6 of 15 columns Previous **1** 2 3 4 5 6 … 9 Next

One thing that dplyr can do that is tough in base R is a compound ordering, such as we see in Excel

Remember this if you need it!

Hide

```
starwars %>%
  arrange(species,desc(mass))
```

name <chr>	height <int>	mass <dbl>	hair_color <chr>	skin_color <chr>

name <chr>	height <int>	mass <dbl>	hair_color <chr>	skin_color <chr>
Ratts Tyerell	79	15.0	none	grey, blue
Dexter Jettster	198	102.0	none	brown
Ki-Adi-Mundi	198	82.0	white	pale
Mas Amedda	196	*NA*	none	blue
Zam Wesell	168	55.0	blonde	fair, green, yellow
IG-88	200	140.0	none	metal
C-3PO	167	75.0	*NA*	gold
R2-D2	96	32.0	*NA*	white, blue
R5-D4	97	32.0	*NA*	white, red
R4-P17	96	*NA*	none	silver, red

1-10 of 87 rows | 1-5 of 14 columns

Previous **1** 2 3 4 5 6 … 9 Next

summarise and group_by

Here are two operations working together

This is much like a tapply operation in base R

using group_by to group by a factor

then summarise calculates a couple of variables for us n() is a count operation within summarise, mean is a the standard mean function

We can set up a number of different operations within summarise

This would be a bit tough to do in base r.

Tapply can group data, but it only allows one function, we could call tapply several times and merge the data

Hide

```
starwars %>%
  group_by(species) %>%
  summarise(
    n = n(),
    mass = mean(mass, na.rm = TRUE)
  ) %>%
  filter(
    n > 1,
    mass > 50
  )
```

species <chr>	n <int>	mass <dbl>
Droid	6	69.75000
Gungan	3	74.00000

species <chr>	n <int>	mass <dbl>
Human	35	82.78182
Kaminoan	2	88.00000
Mirialan	2	53.10000
Twi'lek	2	55.00000
Wookiee	2	124.00000
Zabrak	2	80.00000

8 rows

Here this is in base R

Tapply can group data, but it only allows one function, we could call tapply several times and merge the data

Hide

```
x=tapply(df_starwars$species,df_starwars$species,"length")
y=tapply(df_starwars$mass,df_starwars$species,"mean")
data.frame("n"=x,"mean mass"=y)
```

	n <int>	mean.mass <dbl>
Aleena	1	15.0
Besalisk	1	102.0
Cerean	1	82.0
Chagrian	1	*NA*
Clawdite	1	55.0
Droid	6	*NA*
Dug	1	40.0
Ewok	1	20.0
Geonosian	1	80.0
Gungan	3	*NA*

1-10 of 37 rows Previous **1** 2 3 4 Next

sqldf

This package allows you to use SQL queries on an R dataframe.

See

https://cran.r-project.org/web/packages/sqldf/sqldf.pdf (https://cran.r-project.org/web/packages/sqldf/sqldf.pdf)

If you are really good at SQL, this allows you to use your SQL chops in r

The installation was a bit cranky, use this sequence of installs

install.packages(“gsubfn”) install.packages(“RSQLite”) install.packages(“sqldf”,type=“binary”)

Hide

```
require('sqldf')
```

```
Loading required package: sqldf
Loading required package: gsubfn
Loading required package: proto
Loading required package: RSQLite
```

With this in place, we can run SQL commands on an R dataframe

sqldf is not tolerant of NAs, if the df has an NA, it will crash

the df_starwars used above has NAs, we will use mtcars as an example

Hide

```
sqldf("Select * from mtcars ")
```

mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>	am <dbl>	gear <dbl>
21.0	6	160.0	110	3.90	2.620	16.46	0	1	4
21.0	6	160.0	110	3.90	2.875	17.02	0	1	4
22.8	4	108.0	93	3.85	2.320	18.61	1	1	4
21.4	6	258.0	110	3.08	3.215	19.44	1	0	3
18.7	8	360.0	175	3.15	3.440	17.02	0	0	3
18.1	6	225.0	105	2.76	3.460	20.22	1	0	3
14.3	8	360.0	245	3.21	3.570	15.84	0	0	3
24.4	4	146.7	62	3.69	3.190	20.00	1	0	4
22.8	4	140.8	95	3.92	3.150	22.90	1	0	4
19.2	6	167.6	123	3.92	3.440	18.30	1	0	4

1-10 of 32 rows | 1-10 of 11 columns

Previous **1** 2 3 4 Next

Hide

```
sqldf("Select *  from mtcars where mpg>=30")
```

mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>	am <dbl>	gear <dbl>
32.4	4	78.7	66	4.08	2.200	19.47	1	1	4
30.4	4	75.7	52	4.93	1.615	18.52	1	1	4
33.9	4	71.1	65	4.22	1.835	19.90	1	1	4
30.4	4	95.1	113	3.77	1.513	16.90	1	1	5

4 rows | 1-10 of 11 columns