Manipulating Data with dplyr

Code

In this lesson, you'll learn how to manipulate data using dplyr. dplyr is a fast and powerful R package written by Hadley Wickham and Romain Francois that provides a consistent and concise grammar for manipulating tabular data.

One unique aspect of dplyr is that the same set of tools allow you to work with tabular data from a variety of sources, including data frames, data tables, databases and multidimensional arrays. In this lesson, we'll focus on data frames, but everything you learn will apply equally to other formats.

As you may know, "CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R" (http://cran.rstudio.com/ (http://cran.rstudio.com/)). RStudio maintains one of these so-called 'CRAN mirrors' and they generously make their download logs publicly available (http://cran-logs.rstudio.com/ (http://cran-logs.rstudio.com/)).

We'll be working with the log from July 8, 2014, which contains information on roughly 225,000 package downloads.

I've created a variable called path2csv, which contains the full file path to the dataset. Call read.csv() with two arguments, path2csv and stringsAsFactors = FALSE, and save the result in a new variable called mydf. Check ?read.csv if you need help.

Hide

```
path2csv <- "/home/cabunic/R/x86_64-pc-linux-gnu-library/3.4/swirl/Courses/Getting_and_Cleaning_Data/Manipulating_Data_with_
dplyr/2014-07-08.csv"
mydf <- read.csv(path2csv, stringsAsFactors = FALSE)
```

Use dim() to look at the dimensions of mydf.

Hide

```
dim(mydf)
```

```
[1] 225468     11
```

Now use head() to preview the data.

Hide

```
head(mydf)
```

	X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>
1	1	2014-07-08	00:54:41	80589	3.1.0	x86_64	mingw32	htmltools	0.2.4
2	2	2014-07-08	00:59:53	321767	3.1.0	x86_64	mingw32	tseries	0.10-32
3	3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party	1.0-15
4	4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4
5	5	2014-07-08	00:46:50	79825	3.0.2	x86_64	linux-gnu	digest	0.6.4
6	6	2014-07-08	00:48:04	77681	3.1.0	x86_64	linux-gnu	randomForest	4.6-7

6 rows | 1-10 of 11 columns

The dplyr package was automatically installed (if necessary) and loaded at the beginning of this lesson. Normally, this is something you would have to do on your own. Just to build the habit, type library(dplyr) now to load the package again.

Hide

```
library(dplyr)
```

It's important that you have dplyr version 0.4.0 or later. To confirm this, type packageVersion("dplyr").

If your dplyr version is not at least 0.4.0, then you should hit the Esc key now, reinstall dplyr, then resume this lesson where you left off.

Hide

```
packageVersion("dplyr")
```

```
[1] '0.7.4'
```

The first step of working with data in dplyr is to load the data into what the package authors call a 'data frame tbl' or 'tbl_df'. Use the following code to create a new tbl_df called cran: cran <- tbl_df(mydf).

Hide

```
cran <- tbl_df(mydf)
```

To avoid confusion and keep things running smoothly, let's remove the original data frame from your workspace with rm("mydf").

Hide

```
rm("mydf")
```

From ?tbl_df, "The main advantage to using a tbl_df over a regular data frame is the printing." Let's see what is meant by this. Type cran to print our tbl_df to the console.

Hide

```
cran
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>
1	2014-07-08	00:54:41	80589	3.1.0	x86_64	mingw32	htmltools	0.2.4
2	2014-07-08	00:59:53	321767	3.1.0	x86_64	mingw32	tseries	0.10-32
3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party	1.0-15
4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4
5	2014-07-08	00:46:50	79825	3.0.2	x86_64	linux-gnu	digest	0.6.4
6	2014-07-08	00:48:04	77681	3.1.0	x86_64	linux-gnu	randomForest	4.6-7
7	2014-07-08	00:48:35	393754	3.1.0	x86_64	linux-gnu	plyr	1.8.1
8	2014-07-08	00:47:30	28216	3.0.2	x86_64	linux-gnu	whisker	0.3-2
9	2014-07-08	00:54:58	5928	*NA*	*NA*	*NA*	Rcpp	0.10.4
10	2014-07-08	00:15:35	2206029	3.0.2	x86_64	linux-gnu	hflights	0.1

1-10 of 225,468 rows | 1-9 of 11 columns Previous **1** 2 3 4 5 6 … 100 Next

This output is much more informative and compact than what we would get if we printed the original data frame (mydf) to the console.

First, we are shown the class and dimensions of the dataset. Just below that, we get a preview of the data. Instead of attempting to print the entire dataset, dplyr just shows us the first 10 rows of data and only as many columns as fit neatly in our console. At the bottom, we see the names and classes for any variables that didn't fit on our screen.

According to the "Introduction to dplyr" vignette written by the package authors, "The dplyr philosophy is to have small functions that each do one thing well."

Specifically, dplyr supplies five 'verbs' that cover most fundamental data manipulation tasks:

1. select()
2. filter()
3. arrange()
4. mutate()
5. summarize()

select()

Use ?select to pull up the documentation for the first of these core functions.

Hide

```
?select
```

Help files for the other functions are accessible in the same way.

As may often be the case, particularly with larger datasets, we are only interested in some of the variables. Use select(cran, ip_id, package, country) to select only the ip_id, package, and country variables from the cran dataset.

Hide

```
select(cran, ip_id, package, country)
```

```
```

ip_id <int>	package <chr>	country <chr>
1	htmltools	US
2	tseries	US
3	party	US
3	Hmisc	US
4	digest	CA
3	randomForest	US
3	plyr	US
5	whisker	US
6	Rcpp	CN
7	hflights	US

The first thing to notice is that we don't have to type cranip_id, $cran$package, and cran$country, as we normally would when referring to columns of a data frame. The select() function knows we are referring to columns of the cran dataset.

Also, note that the columns are returned to us in the order we specified, even though ip_id is the rightmost column in the original dataset.

Recall that in R, the ： operator provides a compact notation for creating a sequence of numbers. For example, try 5:20.

Hide

```
5:20
```

```
 [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Normally, this notation is reserved for numbers, but select() allows you to specify a sequence of columns this way, which can save a bunch of typing. Use select(cran, r_arch:country) to select all columns starting from r_arch and ending with country.

Hide

```
select(cran, r_arch:country)
```

r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>
x86_64	mingw32	htmltools	0.2.4	US
x86_64	mingw32	tseries	0.10-32	US
x86_64	linux-gnu	party	1.0-15	US
x86_64	linux-gnu	Hmisc	3.14-4	US
x86_64	linux-gnu	digest	0.6.4	CA
x86_64	linux-gnu	randomForest	4.6-7	US
x86_64	linux-gnu	plyr	1.8.1	US
x86_64	linux-gnu	whisker	0.3-2	US
NA	*NA*	Rcpp	0.10.4	CN
x86_64	linux-gnu	hflights	0.1	US

We can also select the same columns in reverse order. Give it a try.

Hide

```
select(cran, country:r_arch)
```

country <chr>	version <chr>	package <chr>	r_os <chr>	r_arch <chr>
US	0.44-8	LPCM	darwin13.1.0	x86_64
US	1.2-4	colorspace	linux-gnu	x86_64
US	0.11.2	Rcpp	darwin13.1.0	x86_64
US	2.0-0	dichromat	darwin13.1.0	x86_64

country <chr>	version <chr>	package <chr>	r_os <chr>	r_arch <chr>
US	0.44-8	LPCM	darwin13.1.0	x86_64
US	0.2	labeling	darwin10.8.0	x86_64
US	0.1.2	gtable	linux-gnu	x86_64
US	0.2.4	scales	linux-gnu	x86_64
KR	0.4-12	proxy	linux-gnu	x86_64
US	1.7-11	zoo	linux-gnu	x86_64

61-70 of 225,468 rows | Previous 1 … 5 6 **7** 8 9 … 100 Next

Print the entire dataset again, just to remind yourself of what it looks like. You can do this at anytime during the lesson.

Hide

```
cran
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>
1	2014-07-08	00:54:41	80589	3.1.0	x86_64	mingw32	htmltools	0.2.4
2	2014-07-08	00:59:53	321767	3.1.0	x86_64	mingw32	tseries	0.10-32
3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party	1.0-15
4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4
5	2014-07-08	00:46:50	79825	3.0.2	x86_64	linux-gnu	digest	0.6.4
6	2014-07-08	00:48:04	77681	3.1.0	x86_64	linux-gnu	randomForest	4.6-7
7	2014-07-08	00:48:35	393754	3.1.0	x86_64	linux-gnu	plyr	1.8.1
8	2014-07-08	00:47:30	28216	3.0.2	x86_64	linux-gnu	whisker	0.3-2
9	2014-07-08	00:54:58	5928	*NA*	*NA*	*NA*	Rcpp	0.10.4
10	2014-07-08	00:15:35	2206029	3.0.2	x86_64	linux-gnu	hflights	0.1

1-10 of 225,468 rows | 1-9 of 11 columns | Previous **1** 2 3 4 5 6 … 100 Next

Instead of specifying the columns we want to keep, we can also specify the columns we want to throw away. To see how this works, do select(cran, -time) to omit the time column.

Hide

```
select(cran, -time)
```

X <int>	date <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>	ip_id <int>
1	2014-07-08	80589	3.1.0	x86_64	mingw32	htmltools	0.2.4	US	1
2	2014-07-08	321767	3.1.0	x86_64	mingw32	tseries	0.10-32	US	2
3	2014-07-08	748063	3.1.0	x86_64	linux-gnu	party	1.0-15	US	3
4	2014-07-08	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4	US	3
5	2014-07-08	79825	3.0.2	x86_64	linux-gnu	digest	0.6.4	CA	4
6	2014-07-08	77681	3.1.0	x86_64	linux-gnu	randomForest	4.6-7	US	3
7	2014-07-08	393754	3.1.0	x86_64	linux-gnu	plyr	1.8.1	US	3
8	2014-07-08	28216	3.0.2	x86_64	linux-gnu	whisker	0.3-2	US	5
9	2014-07-08	5928	*NA*	*NA*	*NA*	Rcpp	0.10.4	CN	6
10	2014-07-08	2206029	3.0.2	x86_64	linux-gnu	hflights	0.1	US	7

1-10 of 225,468 rows | Previous **1** 2 3 4 5 6 … 100 Next

The negative sign in front of time tells select() that we DON'T want the time column. Now, let's combine strategies to omit all columns from X through size (X:size). To see how this might work, let's look at a numerical example with -5:20.

Hide

```
-5:20
```

```
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Oops! That gaves us a vector of numbers from -5 through 20, which is not what we want. Instead, we want to negate the entire sequence of numbers from 5 through 20, so that we get -5, -6, -7, … , -18, -19, -20. Try the same thing, except surround 5:20 with parentheses so that R knows we want it to first come up with the sequence of numbers, then apply the negative sign to the whole thing.

Hide

```
-(5:20)
```

```
 [1]  -5  -6  -7  -8  -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20
```

Use this knowledge to omit all columns X:size using select().

Hide

```
select(cran, -(X:size))
```

r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>	ip_id <int>
3.1.0	x86_64	mingw32	htmltools	0.2.4	US	1
3.1.0	x86_64	mingw32	tseries	0.10-32	US	2
3.1.0	x86_64	linux-gnu	party	1.0-15	US	3
3.1.0	x86_64	linux-gnu	Hmisc	3.14-4	US	3
3.0.2	x86_64	linux-gnu	digest	0.6.4	CA	4
3.1.0	x86_64	linux-gnu	randomForest	4.6-7	US	3
3.1.0	x86_64	linux-gnu	plyr	1.8.1	US	3
3.0.2	x86_64	linux-gnu	whisker	0.3-2	US	5
NA	*NA*	*NA*	Rcpp	0.10.4	CN	6
3.0.2	x86_64	linux-gnu	hflights	0.1	US	7

1-10 of 225,468 rows | Previous **1** 2 3 4 5 6 … 100 Next

filter()

Now that you know how to select a subset of columns using select(), a natural next question is “How do I select a subset of rows?” That’s where the filter() function comes in.

Use filter(cran, package == “swirl”) to select all rows for which the package variable is equal to “swirl”. Be sure to use two equals signs side-by-side!

Hide

```
filter(cran, package == "swirl")
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>
27	2014-07-08	00:17:16	105350	3.0.2	x86_64	mingw32	swirl	2.2.9	US
156	2014-07-08	00:22:53	41261	3.1.0	x86_64	linux-gnu	swirl	2.2.9	US
358	2014-07-08	00:13:42	105335	2.15.2	x86_64	mingw32	swirl	2.2.9	CA
593	2014-07-08	00:59:45	105465	3.1.0	x86_64	darwin13.1.0	swirl	2.2.9	MX
831	2014-07-08	00:55:27	105335	3.0.3	x86_64	mingw32	swirl	2.2.9	US
997	2014-07-08	00:33:06	41261	3.1.0	x86_64	mingw32	swirl	2.2.9	US
1023	2014-07-08	00:35:36	106393	3.1.0	x86_64	mingw32	swirl	2.2.9	BR
1144	2014-07-08	00:00:39	106534	3.0.2	x86_64	linux-gnu	swirl	2.2.9	US
1402	2014-07-08	00:41:41	41261	3.1.0	i386	mingw32	swirl	2.2.9	US
1424	2014-07-08	00:44:49	106393	3.1.0	x86_64	linux-gnu	swirl	2.2.9	US

1-10 of 820 rows | 1-10 of 11 columns Previous **1** 2 3 4 5 6 … 82 Next

Again, note that filter() recognizes 'package' as a column of cran, without you having to explicitly specify cran$package.

The == operator asks whether the thing on the left is equal to the thing on the right. If yes, then it returns TRUE. If no, then FALSE. In this case, package is an entire vector (column) of values, so package == "swirl" returns a vector of TRUEs and FALSEs. filter() then returns only the rows of cran corresponding to the TRUEs.

You can specify as many conditions as you want, separated by commas. For example filter(cran, r_version == "3.1.1", country == "US") will return all rows of cran corresponding to downloads from users in the US running R version 3.1.1. Try it out.

Hide

```
filter(cran, r_version == "3.1.1", country == "US")
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>
2216	2014-07-08	00:48:58	385112	3.1.1	x86_64	darwin13.1.0	colorspace	1.2-4	US
17332	2014-07-08	03:39:57	197459	3.1.1	x86_64	darwin13.1.0	httr	0.3	US
17465	2014-07-08	03:25:38	23259	3.1.1	x86_64	darwin13.1.0	snow	0.3-13	US
18844	2014-07-08	03:59:17	190594	3.1.1	x86_64	darwin13.1.0	maxLik	1.2-0	US
30182	2014-07-08	04:13:15	77683	3.1.1	i386	mingw32	randomForest	4.6-7	US
30193	2014-07-08	04:06:26	2351969	3.1.1	i386	mingw32	ggplot2	1.0.0	US
30195	2014-07-08	04:07:09	299080	3.1.1	i386	mingw32	fExtremes	3010.81	US
30217	2014-07-08	04:32:04	568036	3.1.1	i386	mingw32	rJava	0.9-6	US
30245	2014-07-08	04:10:41	526858	3.1.1	i386	mingw32	LPCM	0.44-8	US
30354	2014-07-08	04:32:51	1763717	3.1.1	i386	mingw32	mgcv	1.8-1	US

1-10 of 1,588 rows | 1-10 of 11 columns Previous **1** 2 3 4 5 6 … 100 Next

The conditions passed to filter() can make use of any of the standard comparison operators. Pull up the relevant documentation with ?Comparison (that's an uppercase C).

Hide

```
?Comparison
```

Edit your previous call to filter() to instead return rows corresponding to users in "IN" (India) running an R version that is less than or equal to "3.0.2". The up arrow on your keyboard may come in handy here. Don't forget your double quotes!

Hide

```
filter(cran, r_version <= "3.0.2", country == "IN")
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>
348	2014-07-08	00:44:04	10218907	3.0.0	x86_64	mingw32	BH	1.54.0-2
9990	2014-07-08	02:11:32	397497	3.0.2	x86_64	linux-gnu	equateIRT	1.1
9991	2014-07-08	02:11:32	119199	3.0.2	x86_64	linux-gnu	ggdendro	0.1-14
9992	2014-07-08	02:11:33	81779	3.0.2	x86_64	linux-gnu	dfcrm	0.2-2
10022	2014-07-08	02:19:45	1557078	2.15.0	x86_64	mingw32	RcppArmadillo	0.4.320.0
10023	2014-07-08	02:19:46	1184285	2.15.1	i686	linux-gnu	forecast	5.4
10189	2014-07-08	02:38:06	908854	3.0.2	x86_64	linux-gnu	editrules	2.7.2
10199	2014-07-08	02:38:28	178436	3.0.2	x86_64	linux-gnu	energy	1.6.1
10200	2014-07-08	02:38:29	51811	3.0.2	x86_64	linux-gnu	ENmisc	1.2-7
10201	2014-07-08	02:38:29	65245	3.0.2	x86_64	linux-gnu	entropy	1.2.0

1-10 of 4,139 rows | 1-9 of 11 columns Previous **1** 2 3 4 5 6 … 100 Next

Our last two calls to filter() requested all rows for which some condition AND another condition were TRUE. We can also request rows for which EITHER one condition OR another condition are TRUE. For example, filter(cran, country == "US" | country == "IN") will gives us all rows for which the country variable equals either "US" or "IN". Give it a go.

Hide

```
filter(cran, country == "US" | country == "IN")
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>
1	2014-07-08	00:54:41	80589	3.1.0	x86_64	mingw32	htmltools	0.2.4
2	2014-07-08	00:59:53	321767	3.1.0	x86_64	mingw32	tseries	0.10-32
3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party	1.0-15
4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4
6	2014-07-08	00:48:04	77681	3.1.0	x86_64	linux-gnu	randomForest	4.6-7
7	2014-07-08	00:48:35	393754	3.1.0	x86_64	linux-gnu	plyr	1.8.1
8	2014-07-08	00:47:30	28216	3.0.2	x86_64	linux-gnu	whisker	0.3-2
10	2014-07-08	00:15:35	2206029	3.0.2	x86_64	linux-gnu	hflights	0.1
11	2014-07-08	00:15:25	526858	3.0.2	x86_64	linux-gnu	LPCM	0.44-8
12	2014-07-08	00:14:45	2351969	2.14.1	x86_64	linux-gnu	ggplot2	1.0.0

1-10 of 95,283 rows | 1-9 of 11 columns

Previous **1** 2 3 4 5 6 … 100 Next

Now, use filter() to fetch all rows for which size is strictly greater than (>) 100500 (no quotes, since size is numeric) AND r_os equals “linux-gnu”. Hint: You are passing three arguments to filter(): the name of the dataset, the first condition, and the second condition.

Hide

```
filter(cran, size > 100500, r_os == "linux-gnu")
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>
3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party	1.0-15	US
4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4	US
7	2014-07-08	00:48:35	393754	3.1.0	x86_64	linux-gnu	plyr	1.8.1	US
10	2014-07-08	00:15:35	2206029	3.0.2	x86_64	linux-gnu	hflights	0.1	US
11	2014-07-08	00:15:25	526858	3.0.2	x86_64	linux-gnu	LPCM	0.44-8	US
12	2014-07-08	00:14:45	2351969	2.14.1	x86_64	linux-gnu	ggplot2	1.0.0	US
14	2014-07-08	00:15:35	3097729	3.0.2	x86_64	linux-gnu	Rcpp	0.9.7	VE
15	2014-07-08	00:14:37	568036	3.1.0	x86_64	linux-gnu	rJava	0.9-6	US
16	2014-07-08	00:15:50	1600441	3.1.0	x86_64	linux-gnu	RSQLite	0.11.4	US
18	2014-07-08	00:26:59	186685	3.1.0	x86_64	linux-gnu	ipred	0.9-3	DE

1-10 of 33,683 rows | 1-10 of 11 columns

Previous **1** 2 3 4 5 6 … 100 Next

Finally, we want to get only the rows for which the r_version is not missing. R represents missing values with NA and these missing values can be detected using the is.na() function.

To see how this works, try is.na(c(3, 5, NA, 10)).

Hide

```
is.na(c(3, 5, NA, 10))
```

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

Now, put an exclamation point (!) before is.na() to change all of the TRUEs to FALSEs and all of the FALSEs to TRUEs, thus telling us what is NOT NA: !is.na(c(3, 5, NA, 10)).

Hide

```
!is.na(c(3, 5, NA, 10))
```

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

Okay, ready to put all of this together? Use filter() to return all rows of cran for which r_version is NOT NA. Hint: You will need to use !is.na() as part of your second argument to filter().

Hide

```
filter(cran, !is.na(r_version))
```

X <int>	date <chr>	time <chr>	size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>
1	2014-07-08	00:54:41	80589	3.1.0	x86_64	mingw32	htmltools	0.2.4	US
2	2014-07-08	00:59:53	321767	3.1.0	x86_64	mingw32	tseries	0.10-32	US
3	2014-07-08	00:47:13	748063	3.1.0	x86_64	linux-gnu	party	1.0-15	US
4	2014-07-08	00:48:05	606104	3.1.0	x86_64	linux-gnu	Hmisc	3.14-4	US
5	2014-07-08	00:46:50	79825	3.0.2	x86_64	linux-gnu	digest	0.6.4	CA
6	2014-07-08	00:48:04	77681	3.1.0	x86_64	linux-gnu	randomForest	4.6-7	US
7	2014-07-08	00:48:35	393754	3.1.0	x86_64	linux-gnu	plyr	1.8.1	US
8	2014-07-08	00:47:30	28216	3.0.2	x86_64	linux-gnu	whisker	0.3-2	US
10	2014-07-08	00:15:35	2206029	3.0.2	x86_64	linux-gnu	hflights	0.1	US
11	2014-07-08	00:15:25	526858	3.0.2	x86_64	linux-gnu	LPCM	0.44-8	US

1-10 of 207,205 rows | 1-10 of 11 columns

Previous **1** 2 3 4 5 6 … 100 Next

We've seen how to select a subset of columns and rows from our dataset using select() and filter(), respectively. Inherent in select() was also the ability to arrange our selected columns in any order we please.

arrange()

Sometimes we want to order the rows of a dataset according to the values of a particular variable. This is the job of arrange().

To see how arrange() works, let's first take a subset of cran. select() all columns from size through ip_id and store the result in cran2.

Hide

```
cran2 <- select(cran, size:ip_id)
```

Now, to order the ROWS of cran2 so that ip_id is in ascending order (from small to large), type arrange(cran2, ip_id). You may want to make your console wide enough so that you can see ip_id, which is the last column.

Hide

```
arrange(cran2, ip_id)
```

size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>	ip_id <int>
80589	3.1.0	x86_64	mingw32	htmltools	0.2.4	US	1
180562	3.0.2	x86_64	mingw32	yaml	2.1.13	US	1
190120	3.1.0	i386	mingw32	babel	0.2-6	US	1
321767	3.1.0	x86_64	mingw32	tseries	0.10-32	US	2
52281	3.0.3	x86_64	darwin10.8.0	quadprog	1.5-5	US	2
876702	3.1.0	x86_64	linux-gnu	zoo	1.7-11	US	2
321764	3.0.2	x86_64	linux-gnu	tseries	0.10-32	US	2
876702	3.1.0	x86_64	linux-gnu	zoo	1.7-11	US	2
321768	3.1.0	x86_64	mingw32	tseries	0.10-32	US	2
784093	3.1.0	x86_64	linux-gnu	strucchange	1.5-0	US	2

1-10 of 225,468 rows

Previous **1** 2 3 4 5 6 … 100 Next

To do the same, but in descending order, change the second argument to desc(ip_id), where desc() stands for 'descending'. Go ahead.

Hide

```
arrange(cran2, desc(ip_id))
```

size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>	ip_id <int>
5933	*NA*	*NA*	*NA*	CPE	1.4.2	CN	13859

size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>	ip_id <int>
569241	3.1.0	x86_64	mingw32	multcompView	0.1-5	US	13858
228444	3.1.0	x86_64	mingw32	tourr	0.5.3	NZ	13857
308962	3.1.0	x86_64	darwin13.1.0	ctv	0.7-9	CN	13856
950964	3.0.3	i386	mingw32	knitr	1.6	CA	13855
80185	3.0.3	i386	mingw32	htmltools	0.2.4	CA	13855
1431750	3.0.3	i386	mingw32	shiny	0.10.0	CA	13855
2189695	3.1.0	x86_64	mingw32	RMySQL	0.9-3	US	13854
4818024	3.1.0	i386	mingw32	igraph	0.7.1	US	13853
197495	3.1.0	x86_64	mingw32	coda	0.16-1	US	13852

1-10 of 225,468 rows Previous **1** 2 3 4 5 6 … 100 Next

We can also arrange the data according to the values of multiple variables. For example, arrange(cran2, package, ip_id) will first arrange by package names (ascending alphabetically), then by ip_id. This means that if there are multiple rows with the same value for package, they will be sorted by ip_id (ascending numerically). Try arrange(cran2, package, ip_id) now.

Hide

```
arrange(cran2, package, ip_id)
```

size <int>	r_version <chr>	r_arch <chr>	r_os <chr>	package <chr>	version <chr>	country <chr>	ip_id <int>
71677	3.0.3	x86_64	darwin10.8.0	A3	0.9.2	CN	1003
71672	3.1.0	x86_64	linux-gnu	A3	0.9.2	US	1015
71677	3.1.0	x86_64	mingw32	A3	0.9.2	IN	1054
70438	3.0.1	x86_64	darwin10.8.0	A3	0.9.2	CN	1513
71677	*NA*	*NA*	*NA*	A3	0.9.2	BR	1526
71892	3.0.2	x86_64	linux-gnu	A3	0.9.2	IN	1542
71677	3.1.0	x86_64	linux-gnu	A3	0.9.2	ZA	2925
71672	3.1.0	x86_64	mingw32	A3	0.9.2	IL	3889
71677	3.0.3	x86_64	mingw32	A3	0.9.2	DE	3917
71672	3.1.0	x86_64	mingw32	A3	0.9.2	US	4219

1-10 of 225,468 rows Previous **1** 2 3 4 5 6 … 100 Next

Hide

```
arrange(cran2, country, desc(r_version), ip_id)
```

mutate()

To illustrate the next major function in dplyr, let's take another subset of our original data. Use select() to grab 3 columns from cran – ip_id, package, and size (in that order) – and store the result in a new variable called cran3.

Hide

```
cran3 <- select(cran, ip_id, package, size)
```

Take a look at cran3 now.

Hide

```
cran3
```

ip_id <int>	package <chr>	size <int>
1	htmltools	80589
2	tseries	321767

ip_id <int>	package <chr>	size <int>
3	party	748063
3	Hmisc	606104
4	digest	79825
3	randomForest	77681
3	plyr	393754
5	whisker	28216
6	Rcpp	5928
7	hflights	2206029

1-10 of 225,468 rows Previous **1** 2 3 4 5 6 … 100 Next

It’s common to create a new variable based on the value of one or more variables already in a dataset. The mutate() function does exactly this.

The size variable represents the download size in bytes, which are units of computer memory. These days, megabytes (MB) are a more common unit of measurement. One megabyte is equal to 2^20 bytes. That’s 2 to the power of 20, which is approximately one million bytes!

We want to add a column called size_mb that contains the download size in megabytes. Here’s the code to do it: mutate(cran3, size_mb = size / 2^20)

Hide

```
mutate(cran3, size_mb = size / 2^20)
```

ip_id <int>	package <chr>	size <int>	size_mb <dbl>
1	htmltools	80589	0.076855659
2	tseries	321767	0.306860924
3	party	748063	0.713408470
3	Hmisc	606104	0.578025818
4	digest	79825	0.076127052
3	randomForest	77681	0.074082375
3	plyr	393754	0.375513077
5	whisker	28216	0.026908875
6	Rcpp	5928	0.005653381
7	hflights	2206029	2.103833199

1-10 of 225,468 rows Previous **1** 2 3 4 5 6 … 100 Next

An even larger unit of memory is a gigabyte (GB), which equals 2^10 megabytes. We might as well add another column for download size in gigabytes!

One very nice feature of mutate() is that you can use the value computed for your second column (size_mb) to create a third column, all in the same line of code. To see this in action, repeat the exact same command as above, except add a third argument creating a column that is named size_gb and equal to size_mb / 2^10.

Hide

```
mutate(cran3, size_mb = size/2^20, size_gb = size_mb/2^10)
```

ip_id <int>	package <chr>	size <int>	size_mb <dbl>	size_gb <dbl>
1	htmltools	80589	0.076855659	7.505435e-05
2	tseries	321767	0.306860924	2.996689e-04
3	party	748063	0.713408470	6.966880e-04
3	Hmisc	606104	0.578025818	5.644783e-04
4	digest	79825	0.076127052	7.434282e-05
3	randomForest	77681	0.074082375	7.234607e-05
3	plyr	393754	0.375513077	3.667120e-04
5	whisker	28216	0.026908875	2.627820e-05

ip_id <int>	package <chr>	size <int>	size_mb <dbl>	size_gb <dbl>
6	Rcpp	5928	0.005653381	5.520880e-06
7	hflights	2206029	2.103833199	2.054525e-03

1-10 of 225,468 rows Previous **1** 2 3 4 5 6 … 100 Next

Let's try one more for practice. Pretend we discovered a glitch in the system that provided the original values for the size variable. All of the values in cran3 are 1000 bytes less than they should be. Using cran3, create just one new column called correct_size that contains the correct size.

Hide

```
mutate(cran3, correct_size = size + 1000)
```

ip_id <int>	package <chr>	size <int>	correct_size <dbl>
1	htmltools	80589	81589
2	tseries	321767	322767
3	party	748063	749063
3	Hmisc	606104	607104
4	digest	79825	80825
3	randomForest	77681	78681
3	plyr	393754	394754
5	whisker	28216	29216
6	Rcpp	5928	6928
7	hflights	2206029	2207029

1-10 of 225,468 rows Previous **1** 2 3 4 5 6 … 100 Next

summarize()

The last of the five core dplyr verbs, summarize(), collapses the dataset to a single row. Let's say we're interested in knowing the average download size. summarize(cran, avg_bytes = mean(size)) will yield the mean value of the size variable. Here we've chosen to label the result 'avg_bytes', but we could have named it anything. Give it a try.

Hide

```
summarize(cran, avg_bytes = mean(size))
```

avg_bytes <dbl>
844086.5

1 row

That's not particularly interesting. summarize() is most useful when working with data that has been grouped by the values of a particular variable.

We'll look at grouped data in the next lesson, but the idea is that summarize() can give you the requested value FOR EACH group in your dataset.

In this lesson, you learned how to manipulate data using dplyr's five main functions. In the next lesson, we'll look at how to take advantage of some other useful features of dplyr to make your life as a data analyst much easier.

END