| Please choose a lesson, or type 0 to return to course menu.

1: Manipulating Data with dplyr

2: Grouping and Chaining with dplyr

3: Tidying Data with tidyr

4: Dates and Times with lubridate

Selection: 1

| Attemping to load lesson dependencies...

| This lesson requires the ‘dplyr’ package. Would you like me to install it for you now?

1: Yes

2: No

Selection: 1

| Trying to install package ‘dplyr’ now...

also installing the dependencies ‘assertthat’, ‘lazyeval’, ‘DBI’

package ‘assertthat’ successfully unpacked and MD5 sums checked

package ‘lazyeval’ successfully unpacked and MD5 sums checked

package ‘DBI’ successfully unpacked and MD5 sums checked

package ‘dplyr’ successfully unpacked and MD5 sums checked

| Package ‘dplyr’ loaded correctly!

| | 0%

| 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.

...

|== | 2%

| 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.

...

|=== | 3%

| 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/). RStudio maintains one

| of these so-called 'CRAN mirrors' and they generously make their download logs publicly available

| (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.

...

|===== | 5%

| 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.

> mydf <- read.csv(path2csv, stringsAsFactors = FALSE)

| You got it right!

|======= | 7%

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

> dim(mydf)

[1] 225468 11

| You are amazing!

|======== | 8%

| Now use head() to preview the data.

> head(mydf)

X date time size r\_version r\_arch r\_os package version country ip\_id

1 1 2014-07-08 00:54:41 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-08 00:59:53 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-08 00:47:13 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 4 2014-07-08 00:48:05 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 5 2014-07-08 00:46:50 79825 3.0.2 x86\_64 linux-gnu digest 0.6.4 CA 4

6 6 2014-07-08 00:48:04 77681 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

| Keep up the great work!

|========== | 10%

| 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.

> library(dplyr)

| That's the answer I was looking for.

|=========== | 12%

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

> packageVersion("dplyr")

[1] ‘0.4.3’

| You nailed it! Good job!

|============= | 13%

| 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.

...

|=============== | 15%

| 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).

> cran <- tbl\_df(mydf)

| You are doing so well!

|================ | 17%

| To avoid confusion and keep things running smoothly, let's remove the original data frame from your

| workspace with rm("mydf").

> rm("mydf")

| You are quite good my friend!

|================== | 18%

| 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.

> cran

Source: local data frame [225,468 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 1 2014-07-08 00:54:41 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-08 00:59:53 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-08 00:47:13 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 4 2014-07-08 00:48:05 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 5 2014-07-08 00:46:50 79825 3.0.2 x86\_64 linux-gnu digest 0.6.4 CA 4

6 6 2014-07-08 00:48:04 77681 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

7 7 2014-07-08 00:48:35 393754 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

8 8 2014-07-08 00:47:30 28216 3.0.2 x86\_64 linux-gnu whisker 0.3-2 US 5

9 9 2014-07-08 00:54:58 5928 NA NA NA Rcpp 0.10.4 CN 6

10 10 2014-07-08 00:15:35 2206029 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

.. ... ... ... ... ... ... ... ... ... ... ...

| That's correct!

|==================== | 20%

| This output is much more informative and compact than what we would get if we printed the original data

| frame (mydf) to the console.

...

|===================== | 22%

| 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.

...

|======================= | 23%

| 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: select(), filter(), arrange(), mutate(), and summarize().

...

|======================== | 25%

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

> ?select

| You got it right!

|========================== | 27%

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

...

|============================ | 28%

| 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.

> select(cran, ip\_id, package, country)

Source: local data frame [225,468 x 3]

ip\_id package country

(int) (chr) (chr)

1 1 htmltools US

2 2 tseries US

3 3 party US

4 3 Hmisc US

5 4 digest CA

6 3 randomForest US

7 3 plyr US

8 5 whisker US

9 6 Rcpp CN

10 7 hflights US

.. ... ... ...

| Keep working like that and you'll get there!

|============================= | 30%

| The first thing to notice is that we don't have to type cran$ip\_id, cran$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.

...

|=============================== | 32%

| 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.

...

|================================= | 33%

| Recall that in R, the `:` operator provides a compact notation for creating a sequence of numbers. For

| example, try 5:20.

> 5:20

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

| You're the best!

|================================== | 35%

| 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.

> select(cran, r\_arch:country)

Source: local data frame [225,468 x 5]

r\_arch r\_os package version country

(chr) (chr) (chr) (chr) (chr)

1 x86\_64 mingw32 htmltools 0.2.4 US

2 x86\_64 mingw32 tseries 0.10-32 US

3 x86\_64 linux-gnu party 1.0-15 US

4 x86\_64 linux-gnu Hmisc 3.14-4 US

5 x86\_64 linux-gnu digest 0.6.4 CA

6 x86\_64 linux-gnu randomForest 4.6-7 US

7 x86\_64 linux-gnu plyr 1.8.1 US

8 x86\_64 linux-gnu whisker 0.3-2 US

9 NA NA Rcpp 0.10.4 CN

10 x86\_64 linux-gnu hflights 0.1 US

.. ... ... ... ... ...

| You are amazing!

|==================================== | 37%

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

> select(cran, country:r\_arch)

Source: local data frame [225,468 x 5]

country version package r\_os r\_arch

(chr) (chr) (chr) (chr) (chr)

1 US 0.2.4 htmltools mingw32 x86\_64

2 US 0.10-32 tseries mingw32 x86\_64

3 US 1.0-15 party linux-gnu x86\_64

4 US 3.14-4 Hmisc linux-gnu x86\_64

5 CA 0.6.4 digest linux-gnu x86\_64

6 US 4.6-7 randomForest linux-gnu x86\_64

7 US 1.8.1 plyr linux-gnu x86\_64

8 US 0.3-2 whisker linux-gnu x86\_64

9 CN 0.10.4 Rcpp NA NA

10 US 0.1 hflights linux-gnu x86\_64

.. ... ... ... ... ...

| Your dedication is inspiring!

|====================================== | 38%

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

| during the lesson.

> cran

Source: local data frame [225,468 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 1 2014-07-08 00:54:41 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-08 00:59:53 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-08 00:47:13 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 4 2014-07-08 00:48:05 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 5 2014-07-08 00:46:50 79825 3.0.2 x86\_64 linux-gnu digest 0.6.4 CA 4

6 6 2014-07-08 00:48:04 77681 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

7 7 2014-07-08 00:48:35 393754 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

8 8 2014-07-08 00:47:30 28216 3.0.2 x86\_64 linux-gnu whisker 0.3-2 US 5

9 9 2014-07-08 00:54:58 5928 NA NA NA Rcpp 0.10.4 CN 6

10 10 2014-07-08 00:15:35 2206029 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

.. ... ... ... ... ... ... ... ... ... ... ...

| You are really on a roll!

|======================================= | 40%

| 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.

> select(cran, -time)

Source: local data frame [225,468 x 10]

X date size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 1 2014-07-08 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-08 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-08 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 4 2014-07-08 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 5 2014-07-08 79825 3.0.2 x86\_64 linux-gnu digest 0.6.4 CA 4

6 6 2014-07-08 77681 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

7 7 2014-07-08 393754 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

8 8 2014-07-08 28216 3.0.2 x86\_64 linux-gnu whisker 0.3-2 US 5

9 9 2014-07-08 5928 NA NA NA Rcpp 0.10.4 CN 6

10 10 2014-07-08 2206029 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

.. ... ... ... ... ... ... ... ... ... ...

| You are amazing!

|========================================= | 42%

| 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.

> -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

| You are amazing!

|========================================== | 43%

| 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.

> -(5:20)

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

| That's a job well done!

|============================================ | 45%

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

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

Source: local data frame [225,468 x 7]

r\_version r\_arch r\_os package version country ip\_id

(chr) (chr) (chr) (chr) (chr) (chr) (int)

1 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 3.0.2 x86\_64 linux-gnu digest 0.6.4 CA 4

6 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

7 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

8 3.0.2 x86\_64 linux-gnu whisker 0.3-2 US 5

9 NA NA NA Rcpp 0.10.4 CN 6

10 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

.. ... ... ... ... ... ... ...

| You got it!

|============================================== | 47%

| 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.

...

|=============================================== | 48%

| 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!

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

Source: local data frame [820 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 27 2014-07-08 00:17:16 105350 3.0.2 x86\_64 mingw32 swirl 2.2.9 US 20

2 156 2014-07-08 00:22:53 41261 3.1.0 x86\_64 linux-gnu swirl 2.2.9 US 66

3 358 2014-07-08 00:13:42 105335 2.15.2 x86\_64 mingw32 swirl 2.2.9 CA 115

4 593 2014-07-08 00:59:45 105465 3.1.0 x86\_64 darwin13.1.0 swirl 2.2.9 MX 162

5 831 2014-07-08 00:55:27 105335 3.0.3 x86\_64 mingw32 swirl 2.2.9 US 57

6 997 2014-07-08 00:33:06 41261 3.1.0 x86\_64 mingw32 swirl 2.2.9 US 70

7 1023 2014-07-08 00:35:36 106393 3.1.0 x86\_64 mingw32 swirl 2.2.9 BR 248

8 1144 2014-07-08 00:00:39 106534 3.0.2 x86\_64 linux-gnu swirl 2.2.9 US 261

9 1402 2014-07-08 00:41:41 41261 3.1.0 i386 mingw32 swirl 2.2.9 US 234

10 1424 2014-07-08 00:44:49 106393 3.1.0 x86\_64 linux-gnu swirl 2.2.9 US 301

.. ... ... ... ... ... ... ... ... ... ... ...

| Your dedication is inspiring!

|================================================= | 50%

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

| specify cran$package.

...

|=================================================== | 52%

| 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.

...

|==================================================== | 53%

| 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.

> filter(cran, r\_version == "3.1.1", country == "US")

Source: local data frame [1,588 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 2216 2014-07-08 00:48:58 385112 3.1.1 x86\_64 darwin13.1.0 colorspace 1.2-4 US 191

2 17332 2014-07-08 03:39:57 197459 3.1.1 x86\_64 darwin13.1.0 httr 0.3 US 1704

3 17465 2014-07-08 03:25:38 23259 3.1.1 x86\_64 darwin13.1.0 snow 0.3-13 US 62

4 18844 2014-07-08 03:59:17 190594 3.1.1 x86\_64 darwin13.1.0 maxLik 1.2-0 US 1533

5 30182 2014-07-08 04:13:15 77683 3.1.1 i386 mingw32 randomForest 4.6-7 US 646

6 30193 2014-07-08 04:06:26 2351969 3.1.1 i386 mingw32 ggplot2 1.0.0 US 8

7 30195 2014-07-08 04:07:09 299080 3.1.1 i386 mingw32 fExtremes 3010.81 US 2010

8 30217 2014-07-08 04:32:04 568036 3.1.1 i386 mingw32 rJava 0.9-6 US 98

9 30245 2014-07-08 04:10:41 526858 3.1.1 i386 mingw32 LPCM 0.44-8 US 8

10 30354 2014-07-08 04:32:51 1763717 3.1.1 i386 mingw32 mgcv 1.8-1 US 2122

.. ... ... ... ... ... ... ... ... ... ... ...

| Excellent job!

|====================================================== | 55%

| 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).

> ?Comparison

| You're the best!

|======================================================== | 57%

| 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!

> filter(cran, r\_version <= "3.0.2", country == "IN")

Source: local data frame [4,139 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 348 2014-07-08 00:44:04 10218907 3.0.0 x86\_64 mingw32 BH 1.54.0-2 IN 112

2 9990 2014-07-08 02:11:32 397497 3.0.2 x86\_64 linux-gnu equateIRT 1.1 IN 1054

3 9991 2014-07-08 02:11:32 119199 3.0.2 x86\_64 linux-gnu ggdendro 0.1-14 IN 1054

4 9992 2014-07-08 02:11:33 81779 3.0.2 x86\_64 linux-gnu dfcrm 0.2-2 IN 1054

5 10022 2014-07-08 02:19:45 1557078 2.15.0 x86\_64 mingw32 RcppArmadillo 0.4.320.0 IN 1060

6 10023 2014-07-08 02:19:46 1184285 2.15.1 i686 linux-gnu forecast 5.4 IN 1060

7 10189 2014-07-08 02:38:06 908854 3.0.2 x86\_64 linux-gnu editrules 2.7.2 IN 1054

8 10199 2014-07-08 02:38:28 178436 3.0.2 x86\_64 linux-gnu energy 1.6.1 IN 1054

9 10200 2014-07-08 02:38:29 51811 3.0.2 x86\_64 linux-gnu ENmisc 1.2-7 IN 1054

10 10201 2014-07-08 02:38:29 65245 3.0.2 x86\_64 linux-gnu entropy 1.2.0 IN 1054

.. ... ... ... ... ... ... ... ... ... ... ...

| Nice work!

|========================================================= | 58%

| 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.

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

Source: local data frame [95,283 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 1 2014-07-08 00:54:41 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-08 00:59:53 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-08 00:47:13 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 4 2014-07-08 00:48:05 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 6 2014-07-08 00:48:04 77681 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

6 7 2014-07-08 00:48:35 393754 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

7 8 2014-07-08 00:47:30 28216 3.0.2 x86\_64 linux-gnu whisker 0.3-2 US 5

8 10 2014-07-08 00:15:35 2206029 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

9 11 2014-07-08 00:15:25 526858 3.0.2 x86\_64 linux-gnu LPCM 0.44-8 US 8

10 12 2014-07-08 00:14:45 2351969 2.14.1 x86\_64 linux-gnu ggplot2 1.0.0 US 8

.. ... ... ... ... ... ... ... ... ... ... ...

| Excellent job!

|=========================================================== | 60%

| 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.

> filter(cran, size > 100500, r\_os == "linux-gnu")

Source: local data frame [33,683 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 3 2014-07-08 00:47:13 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

2 4 2014-07-08 00:48:05 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

3 7 2014-07-08 00:48:35 393754 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

4 10 2014-07-08 00:15:35 2206029 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

5 11 2014-07-08 00:15:25 526858 3.0.2 x86\_64 linux-gnu LPCM 0.44-8 US 8

6 12 2014-07-08 00:14:45 2351969 2.14.1 x86\_64 linux-gnu ggplot2 1.0.0 US 8

7 14 2014-07-08 00:15:35 3097729 3.0.2 x86\_64 linux-gnu Rcpp 0.9.7 VE 10

8 15 2014-07-08 00:14:37 568036 3.1.0 x86\_64 linux-gnu rJava 0.9-6 US 11

9 16 2014-07-08 00:15:50 1600441 3.1.0 x86\_64 linux-gnu RSQLite 0.11.4 US 7

10 18 2014-07-08 00:26:59 186685 3.1.0 x86\_64 linux-gnu ipred 0.9-3 DE 13

.. ... ... ... ... ... ... ... ... ... ... ...

| All that practice is paying off!

|============================================================ | 62%

| 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.

...

|============================================================== | 63%

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

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

[1] FALSE FALSE TRUE FALSE

| Keep working like that and you'll get there!

|================================================================ | 65%

| 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)).

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

[1] TRUE TRUE FALSE TRUE

| That's the answer I was looking for.

|================================================================= | 67%

| 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().

> filter(cran, !is.na(r\_version))

Source: local data frame [207,205 x 11]

X date time size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 1 2014-07-08 00:54:41 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-08 00:59:53 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-08 00:47:13 748063 3.1.0 x86\_64 linux-gnu party 1.0-15 US 3

4 4 2014-07-08 00:48:05 606104 3.1.0 x86\_64 linux-gnu Hmisc 3.14-4 US 3

5 5 2014-07-08 00:46:50 79825 3.0.2 x86\_64 linux-gnu digest 0.6.4 CA 4

6 6 2014-07-08 00:48:04 77681 3.1.0 x86\_64 linux-gnu randomForest 4.6-7 US 3

7 7 2014-07-08 00:48:35 393754 3.1.0 x86\_64 linux-gnu plyr 1.8.1 US 3

8 8 2014-07-08 00:47:30 28216 3.0.2 x86\_64 linux-gnu whisker 0.3-2 US 5

9 10 2014-07-08 00:15:35 2206029 3.0.2 x86\_64 linux-gnu hflights 0.1 US 7

10 11 2014-07-08 00:15:25 526858 3.0.2 x86\_64 linux-gnu LPCM 0.44-8 US 8

.. ... ... ... ... ... ... ... ... ... ... ...

| You are doing so well!

|=================================================================== | 68%

| 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.

...

|===================================================================== | 70%

| Sometimes we want to order the rows of a dataset according to the values of a particular variable. This

| is the job of arrange().

...

|====================================================================== | 72%

| 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.

> cran2 <- select(cran, size:ip\_id)

| You are quite good my friend!

|======================================================================== | 73%

| 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.

> arrange(cran2, ip\_id)

Source: local data frame [225,468 x 8]

size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 80589 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 180562 3.0.2 x86\_64 mingw32 yaml 2.1.13 US 1

3 190120 3.1.0 i386 mingw32 babel 0.2-6 US 1

4 321767 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

5 52281 3.0.3 x86\_64 darwin10.8.0 quadprog 1.5-5 US 2

6 876702 3.1.0 x86\_64 linux-gnu zoo 1.7-11 US 2

7 321764 3.0.2 x86\_64 linux-gnu tseries 0.10-32 US 2

8 876702 3.1.0 x86\_64 linux-gnu zoo 1.7-11 US 2

9 321768 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

10 784093 3.1.0 x86\_64 linux-gnu strucchange 1.5-0 US 2

.. ... ... ... ... ... ... ... ...

| Excellent work!

|========================================================================== | 75%

| To do the same, but in descending order, change the second argument to desc(ip\_id), where desc() stands

| for 'descending'. Go ahead.

> arrange(cran2, desc(ip\_id))

Source: local data frame [225,468 x 8]

size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 5933 NA NA NA CPE 1.4.2 CN 13859

2 569241 3.1.0 x86\_64 mingw32 multcompView 0.1-5 US 13858

3 228444 3.1.0 x86\_64 mingw32 tourr 0.5.3 NZ 13857

4 308962 3.1.0 x86\_64 darwin13.1.0 ctv 0.7-9 CN 13856

5 950964 3.0.3 i386 mingw32 knitr 1.6 CA 13855

6 80185 3.0.3 i386 mingw32 htmltools 0.2.4 CA 13855

7 1431750 3.0.3 i386 mingw32 shiny 0.10.0 CA 13855

8 2189695 3.1.0 x86\_64 mingw32 RMySQL 0.9-3 US 13854

9 4818024 3.1.0 i386 mingw32 igraph 0.7.1 US 13853

10 197495 3.1.0 x86\_64 mingw32 coda 0.16-1 US 13852

.. ... ... ... ... ... ... ... ...

| You are really on a roll!

|=========================================================================== | 77%

| 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.

> arrange(cran2, package, ip\_id)

Source: local data frame [225,468 x 8]

size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 71677 3.0.3 x86\_64 darwin10.8.0 A3 0.9.2 CN 1003

2 71672 3.1.0 x86\_64 linux-gnu A3 0.9.2 US 1015

3 71677 3.1.0 x86\_64 mingw32 A3 0.9.2 IN 1054

4 70438 3.0.1 x86\_64 darwin10.8.0 A3 0.9.2 CN 1513

5 71677 NA NA NA A3 0.9.2 BR 1526

6 71892 3.0.2 x86\_64 linux-gnu A3 0.9.2 IN 1542

7 71677 3.1.0 x86\_64 linux-gnu A3 0.9.2 ZA 2925

8 71672 3.1.0 x86\_64 mingw32 A3 0.9.2 IL 3889

9 71677 3.0.3 x86\_64 mingw32 A3 0.9.2 DE 3917

10 71672 3.1.0 x86\_64 mingw32 A3 0.9.2 US 4219

.. ... ... ... ... ... ... ... ...

| You nailed it! Good job!

|============================================================================= | 78%

| Arrange cran2 by the following three variables, in this order: country (ascending), r\_version

| (descending), and ip\_id (ascending).

> arrange(cran2, country, desc(r\_version), ip\_id)

Source: local data frame [225,468 x 8]

size r\_version r\_arch r\_os package version country ip\_id

(int) (chr) (chr) (chr) (chr) (chr) (chr) (int)

1 1556858 3.1.1 i386 mingw32 RcppArmadillo 0.4.320.0 A1 2843

2 1823512 3.1.0 x86\_64 linux-gnu mgcv 1.8-1 A1 2843

3 15732 3.1.0 i686 linux-gnu grnn 0.1.0 A1 3146

4 3014840 3.1.0 x86\_64 mingw32 Rcpp 0.11.2 A1 3146

5 660087 3.1.0 i386 mingw32 xts 0.9-7 A1 3146

6 522261 3.1.0 i386 mingw32 FNN 1.1 A1 3146

7 522263 3.1.0 i386 mingw32 FNN 1.1 A1 3146

8 1676627 3.1.0 x86\_64 linux-gnu rgeos 0.3-5 A1 3146

9 2118530 3.1.0 x86\_64 linux-gnu spacetime 1.1-0 A1 3146

10 2217180 3.1.0 x86\_64 mingw32 gstat 1.0-19 A1 3146

.. ... ... ... ... ... ... ... ...

| All that practice is paying off!

|============================================================================== | 80%

| 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.

> cran3 <- select(cran, ip\_id, package, size)

| You are really on a roll!

|================================================================================ | 82%

| Take a look at cran3 now.

> cran3

Source: local data frame [225,468 x 3]

ip\_id package size

(int) (chr) (int)

1 1 htmltools 80589

2 2 tseries 321767

3 3 party 748063

4 3 Hmisc 606104

5 4 digest 79825

6 3 randomForest 77681

7 3 plyr 393754

8 5 whisker 28216

9 6 Rcpp 5928

10 7 hflights 2206029

.. ... ... ...

| Keep working like that and you'll get there!

|================================================================================== | 83%

| 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.

...

|=================================================================================== | 85%

| 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!

...

|===================================================================================== | 87%

| 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)

> mutate(cran3, size\_mb = size / 2^20)

Source: local data frame [225,468 x 4]

ip\_id package size size\_mb

(int) (chr) (int) (dbl)

1 1 htmltools 80589 0.076855659

2 2 tseries 321767 0.306860924

3 3 party 748063 0.713408470

4 3 Hmisc 606104 0.578025818

5 4 digest 79825 0.076127052

6 3 randomForest 77681 0.074082375

7 3 plyr 393754 0.375513077

8 5 whisker 28216 0.026908875

9 6 Rcpp 5928 0.005653381

10 7 hflights 2206029 2.103833199

.. ... ... ... ...

| That's correct!

|======================================================================================= | 88%

| 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!

...

|======================================================================================== | 90%

| 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.

> mutate(cran3, size\_mb = size / 2^20, size\_gb = size\_mb / 2^10)

Source: local data frame [225,468 x 5]

ip\_id package size size\_mb size\_gb

(int) (chr) (int) (dbl) (dbl)

1 1 htmltools 80589 0.076855659 7.505435e-05

2 2 tseries 321767 0.306860924 2.996689e-04

3 3 party 748063 0.713408470 6.966880e-04

4 3 Hmisc 606104 0.578025818 5.644783e-04

5 4 digest 79825 0.076127052 7.434282e-05

6 3 randomForest 77681 0.074082375 7.234607e-05

7 3 plyr 393754 0.375513077 3.667120e-04

8 5 whisker 28216 0.026908875 2.627820e-05

9 6 Rcpp 5928 0.005653381 5.520880e-06

10 7 hflights 2206029 2.103833199 2.054525e-03

.. ... ... ... ... ...

| Great job!

|========================================================================================== | 92%

| 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.

> mutate(cran3, correct\_size = size + 1000)

Source: local data frame [225,468 x 4]

ip\_id package size correct\_size

(int) (chr) (int) (dbl)

1 1 htmltools 80589 81589

2 2 tseries 321767 322767

3 3 party 748063 749063

4 3 Hmisc 606104 607104

5 4 digest 79825 80825

6 3 randomForest 77681 78681

7 3 plyr 393754 394754

8 5 whisker 28216 29216

9 6 Rcpp 5928 6928

10 7 hflights 2206029 2207029

.. ... ... ... ...

| You are quite good my friend!

|=========================================================================================== | 93%

| 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.

> summarize(cran, avg\_bytes = mean(size))

Source: local data frame [1 x 1]

avg\_bytes

(dbl)

1 844086.5

| Excellent job!

|============================================================================================= | 95%

| That's not particularly interesting. summarize() is most useful when working with data that has been

| grouped by the values of a particular variable.

...

|=============================================================================================== | 97%

| 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.

...

|================================================================================================ | 98%

| 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.

...

|==================================================================================================| 100%