|== | 2%

| In the last lesson, you learned about the five main data manipulation 'verbs' in dplyr:

| select(), filter(), arrange(), mutate(), and summarize(). The last of these,

| summarize(), is most powerful when applied to grouped data.

...

|=== | 4%

| The main idea behind grouping data is that you want to break up your dataset into groups

| of rows based on the values of one or more variables. The group\_by() function is

| reponsible for doing this.

...

|===== | 6%

| We'll continue where we left off with RStudio's CRAN download log from July 8, 2014,

| which contains information on roughly 225,000 R package downloads

| (http://cran-logs.rstudio.com/).

...

|====== | 8%

| As with the last lesson, 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)

| Great job!

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

| I've made the dataset available to you in a data frame called mydf. Put it in a 'data

| frame tbl' using the tbl\_df() function and store the result in a object called cran. If

| you're not sure what I'm talking about, you should start with the previous lesson.

| Otherwise, practice makes perfect!

> cran <- tbl\_df(mydf)

| Great job!

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

| To avoid confusion and keep things running smoothly, let's remove the original dataframe

| from your workspace with rm("mydf").

> rm("mydf")

| You are doing so well!

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

| Print cran to the console.

> cran

# A tibble: 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-… 00:54:… 8.06e4 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-… 00:59:… 3.22e5 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-… 00:47:… 7.48e5 3.1.0 x86\_64 linux-… party 1.0-15 US 3

4 4 2014-07-… 00:48:… 6.06e5 3.1.0 x86\_64 linux-… Hmisc 3.14-4 US 3

5 5 2014-07-… 00:46:… 7.98e4 3.0.2 x86\_64 linux-… digest 0.6.4 CA 4

6 6 2014-07-… 00:48:… 7.77e4 3.1.0 x86\_64 linux-… randomFor… 4.6-7 US 3

7 7 2014-07-… 00:48:… 3.94e5 3.1.0 x86\_64 linux-… plyr 1.8.1 US 3

8 8 2014-07-… 00:47:… 2.82e4 3.0.2 x86\_64 linux-… whisker 0.3-2 US 5

9 9 2014-07-… 00:54:… 5.93e3 NA NA NA Rcpp 0.10.4 CN 6

10 10 2014-07-… 00:15:… 2.21e6 3.0.2 x86\_64 linux-… hflights 0.1 US 7

# … with 225,458 more rows

| Nice work!

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

| Our first goal is to group the data by package name. Bring up the help file for

| group\_by().

> ?group\_by

| You got it right!

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

| Group cran by the package variable and store the result in a new object called

| by\_package.

> by\_package <- group\_by(cran)

| That's not exactly what I'm looking for. Try again. Or, type info() for more options.

| Store the result of group\_by(cran, package) in a new object called by\_package.

> by\_package <- group\_by(cran , package)

| You got it!

|================ | 19%

| Let's take a look at by\_package. Print it to the console.

> by\_package

# A tibble: 225,468 x 11

# Groups: package [6,023]

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-… 00:54:… 8.06e4 3.1.0 x86\_64 mingw32 htmltools 0.2.4 US 1

2 2 2014-07-… 00:59:… 3.22e5 3.1.0 x86\_64 mingw32 tseries 0.10-32 US 2

3 3 2014-07-… 00:47:… 7.48e5 3.1.0 x86\_64 linux-… party 1.0-15 US 3

4 4 2014-07-… 00:48:… 6.06e5 3.1.0 x86\_64 linux-… Hmisc 3.14-4 US 3

5 5 2014-07-… 00:46:… 7.98e4 3.0.2 x86\_64 linux-… digest 0.6.4 CA 4

6 6 2014-07-… 00:48:… 7.77e4 3.1.0 x86\_64 linux-… randomFor… 4.6-7 US 3

7 7 2014-07-… 00:48:… 3.94e5 3.1.0 x86\_64 linux-… plyr 1.8.1 US 3

8 8 2014-07-… 00:47:… 2.82e4 3.0.2 x86\_64 linux-… whisker 0.3-2 US 5

9 9 2014-07-… 00:54:… 5.93e3 NA NA NA Rcpp 0.10.4 CN 6

10 10 2014-07-… 00:15:… 2.21e6 3.0.2 x86\_64 linux-… hflights 0.1 US 7

# … with 225,458 more rows

| Nice work!

|================= | 21%

| At the top of the output above, you'll see 'Groups: package', which tells us that this

| tbl has been grouped by the package variable. Everything else looks the same, but now

| any operation we apply to the grouped data will take place on a per package basis.

...

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

| Recall that when we applied mean(size) to the original tbl\_df via summarize(), it

| returned a single number -- the mean of all values in the size column. We may care about

| what that number is, but wouldn't it be so much more interesting to look at the mean

| download size for each unique package?

...

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

| That's exactly what you'll get if you use summarize() to apply mean(size) to the grouped

| data in by\_package. Give it a shot.

> summarize(by\_package, mean)

Error: Column `mean` is of unsupported type function

> summarize(by\_package, mean())

Error in mean.default() : argument "x" is missing, with no default

> ?summarize

> summarize(by\_package

+ )

# A tibble: 6,023 x 1

package

<chr>

1 NA

2 A3

3 abc

4 abcdeFBA

5 ABCExtremes

6 ABCoptim

7 ABCp2

8 abctools

9 abd

10 abf2

# … with 6,013 more rows

| One more time. You can do it! Or, type info() for more options.

| Call summarize() with two arguments: by\_package and mean(size).

> summarize(by\_package, mean(size))

# A tibble: 6,023 x 2

package `mean(size)`

<chr> <dbl>

1 NA 822376.

2 A3 62195.

3 abc 4826665

4 abcdeFBA 455980.

5 ABCExtremes 22904.

6 ABCoptim 17807.

7 ABCp2 30473.

8 abctools 2589394

9 abd 453631.

10 abf2 35693.

# … with 6,013 more rows

| You are really on a roll!

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

| Instead of returning a single value, summarize() now returns the mean size for EACH

| package in our dataset.

...

|======================= | 29%

| Let's take it a step further. I just opened an R script for you that contains a

| partially constructed call to summarize(). Follow the instructions in the script

| comments.

|

| When you are ready to move on, save the script and type submit(), or type reset() to

| reset the script to its original state.

> submit()

| Sourcing your script...

Error in n\_countries(country) : could not find function "n\_countries"

| Almost! Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

| You are really on a roll!

|========================= | 31%

| Print the resulting tbl, pack\_sum, to the console to examine its contents.

> tbl

function (src, ...)

{

UseMethod("tbl")

}

<bytecode: 0x55e7a7e7c3d8>

<environment: namespace:dplyr>

| Almost! Try again. Or, type info() for more options.

| Type pack\_sum to view its contents.

> pack\_sum(tbl)

Error in pack\_sum(tbl) : could not find function "pack\_sum"

> pack\_sum()

Error in pack\_sum() : could not find function "pack\_sum"

> pack\_sum

# A tibble: 6,023 x 5

package count unique countries avg\_bytes

<chr> <int> <int> <int> <dbl>

1 NA 13 8 7 822376.

2 A3 25 24 10 62195.

3 abc 29 25 16 4826665

4 abcdeFBA 15 15 9 455980.

5 ABCExtremes 18 17 9 22904.

6 ABCoptim 16 15 9 17807.

7 ABCp2 18 17 10 30473.

8 abctools 19 19 11 2589394

9 abd 17 16 10 453631.

10 abf2 13 13 9 35693.

# … with 6,013 more rows

| Your dedication is inspiring!

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

| The 'count' column, created with n(), contains the total number of rows (i.e. downloads)

| for each package. The 'unique' column, created with n\_distinct(ip\_id), gives the total

| number of unique downloads for each package, as measured by the number of distinct

| ip\_id's. The 'countries' column, created with n\_distinct(country), provides the number

| of countries in which each package was downloaded. And finally, the 'avg\_bytes' column,

| created with mean(size), contains the mean download size (in bytes) for each package.

...

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

| It's important that you understand how each column of pack\_sum was created and what it

| means. Now that we've summarized the data by individual packages, let's play around with

| it some more to see what we can learn.

...

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

| Naturally, we'd like to know which packages were most popular on the day these data were

| collected (July 8, 2014). Let's start by isolating the top 1% of packages, based on the

| total number of downloads as measured by the 'count' column.

...

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

| We need to know the value of 'count' that splits the data into the top 1% and bottom 99%

| of packages based on total downloads. In statistics, this is called the 0.99, or 99%,

| sample quantile. Use quantile(pack\_sum$count, probs = 0.99) to determine this number.

> quantile(pack\_sum$count, probs = 0.99)

99%

679.56

| All that hard work is paying off!

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

| Now we can isolate only those packages which had more than 679 total downloads. Use

| filter() to select all rows from pack\_sum for which 'count' is strictly greater (>) than

| 679. Store the result in a new object called top\_counts.

> ?filter

> filter(pack\_sum, count > 679)

# A tibble: 61 x 5

package count unique countries avg\_bytes

<chr> <int> <int> <int> <dbl>

1 bitops 1549 1408 76 28715.

2 car 1008 837 64 1229122.

3 caTools 812 699 64 176589.

4 colorspace 1683 1433 80 357411.

5 data.table 680 564 59 1252721.

6 DBI 2599 492 48 206933.

7 devtools 769 560 55 212933.

8 dichromat 1486 1257 74 134732.

9 digest 2210 1894 83 120549.

10 doSNOW 740 75 24 8364.

# … with 51 more rows

| That's not the answer I was looking for, but try again. Or, type info() for more

| options.

| Store the result of filter(pack\_sum, count > 679) in a new object called top\_counts.

> top\_counts <- filter(pack\_sum, count > 679)

| You are quite good my friend!

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

| Let's take a look at top\_counts. Print it to the console.

> top\_counts

# A tibble: 61 x 5

package count unique countries avg\_bytes

<chr> <int> <int> <int> <dbl>

1 bitops 1549 1408 76 28715.

2 car 1008 837 64 1229122.

3 caTools 812 699 64 176589.

4 colorspace 1683 1433 80 357411.

5 data.table 680 564 59 1252721.

6 DBI 2599 492 48 206933.

7 devtools 769 560 55 212933.

8 dichromat 1486 1257 74 134732.

9 digest 2210 1894 83 120549.

10 doSNOW 740 75 24 8364.

# … with 51 more rows

| Great job!

|==================================== | 44%

| There are only 61 packages in our top 1%, so we'd like to see all of them. Since dplyr

| only shows us the first 10 rows, we can use the View() function to see more.

...

|===================================== | 46%

| View all 61 rows with View(top\_counts). Note that the 'V' in View() is capitalized.

> View(top\_counts)

| You nailed it! Good job!

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

| arrange() the rows of top\_counts based on the 'count' column and assign the result to a

| new object called top\_counts\_sorted. We want the packages with the highest number of

| downloads at the top, which means we want 'count' to be in descending order. If you need

| help, check out ?arrange and/or ?desc.

> ?arrange

> ?desc

> top\_counts\_sorted <- arrange(top\_counts, desc(count))

| You are amazing!

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

| Now use View() again to see all 61 rows of top\_counts\_sorted.

> View(top\_counts\_sorted)

| You're the best!

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

| If we use total number of downloads as our metric for popularity, then the above output

| shows us the most popular packages downloaded from the RStudio CRAN mirror on July 8,

| 2014. Not surprisingly, ggplot2 leads the pack with 4602 downloads, followed by Rcpp,

| plyr, rJava, ....

...

|============================================ | 54%

| ...And if you keep on going, you'll see swirl at number 43, with 820 total downloads.

| Sweet!

...

|============================================= | 56%

| Perhaps we're more interested in the number of \*unique\* downloads on this particular

| day. In other words, if a package is downloaded ten times in one day from the same

| computer, we may wish to count that as only one download. That's what the 'unique'

| column will tell us.

...

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

| Like we did with 'count', let's find the 0.99, or 99%, quantile for the 'unique'

| variable with quantile(pack\_sum$unique, probs = 0.99).

> quantile(pack\_sum$unique, probs = 0.99)

99%

465

| You are doing so well!

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

| Apply filter() to pack\_sum to select all rows corresponding to values of 'unique' that

| are strictly greater than 465. Assign the result to a object called top\_unique.

> top\_unique <- filter(pack\_sum, unique > 465)

| You got it right!

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

| Let's View() our top contenders!

> View(top\_unique)

| You got it!

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

| Now arrange() top\_unique by the 'unique' column, in descending order, to see which

| packages were downloaded from the greatest number of unique IP addresses. Assign the

| result to top\_unique\_sorted.

> top\_unique\_sorted <- arrange(top\_unique, desc(unique))

| You are doing so well!

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

| View() the sorted data.

> View(top\_unique\_sorted)

| Great job!

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

| Now Rcpp is in the lead, followed by stringr, digest, plyr, and ggplot2. swirl moved up

| a few spaces to number 40, with 698 unique downloads. Nice!

...

|======================================================== | 69%

| Our final metric of popularity is the number of distinct countries from which each

| package was downloaded. We'll approach this one a little differently to introduce you to

| a method called 'chaining' (or 'piping').

...

|========================================================== | 71%

| Chaining allows you to string together multiple function calls in a way that is compact

| and readable, while still accomplishing the desired result. To make it more concrete,

| let's compute our last popularity metric from scratch, starting with our original data.

...

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

| I've opened up a script that contains code similar to what you've seen so far. Don't

| change anything. Just study it for a minute, make sure you understand everything that's

| there, then submit() when you are ready to move on.

> submit()

| Sourcing your script...

# A tibble: 46 x 5

package count unique countries avg\_bytes

<chr> <int> <int> <int> <dbl>

1 Rcpp 3195 2044 84 2512100.

2 digest 2210 1894 83 120549.

3 stringr 2267 1948 82 65277.

4 plyr 2908 1754 81 799123.

5 ggplot2 4602 1680 81 2427716.

6 colorspace 1683 1433 80 357411.

7 RColorBrewer 1890 1584 79 22764.

8 scales 1726 1408 77 126819.

9 bitops 1549 1408 76 28715.

10 reshape2 2032 1652 76 330128.

# … with 36 more rows

| Perseverance, that's the answer.

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

| It's worth noting that we sorted primarily by country, but used avg\_bytes (in ascending

| order) as a tie breaker. This means that if two packages were downloaded from the same

| number of countries, the package with a smaller average download size received a higher

| ranking.

...

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

| We'd like to accomplish the same result as the last script, but avoid saving our

| intermediate results. This requires embedding function calls within one another.

...

|================================================================ | 79%

| That's exactly what we've done in this script. The result is equivalent, but the code is

| much less readable and some of the arguments are far away from the function to which

| they belong. Again, just try to understand what is going on here, then submit() when you

| are ready to see a better solution.

> submit()

| Sourcing your script...

# A tibble: 46 x 5

package count unique countries avg\_bytes

<chr> <int> <int> <int> <dbl>

1 Rcpp 3195 2044 84 2512100.

2 digest 2210 1894 83 120549.

3 stringr 2267 1948 82 65277.

4 plyr 2908 1754 81 799123.

5 ggplot2 4602 1680 81 2427716.

6 colorspace 1683 1433 80 357411.

7 RColorBrewer 1890 1584 79 22764.

8 scales 1726 1408 77 126819.

9 bitops 1549 1408 76 28715.

10 reshape2 2032 1652 76 330128.

# … with 36 more rows

| Great job!

|================================================================= | 81%

| In this script, we've used a special chaining operator, %>%, which was originally

| introduced in the magrittr R package and has now become a key component of dplyr. You

| can pull up the related documentation with ?chain. The benefit of %>% is that it allows

| us to chain the function calls in a linear fashion. The code to the right of %>%

| operates on the result from the code to the left of %>%.

|

| Once again, just try to understand the code, then type submit() to continue.

> ?chain

No documentation for ‘chain’ in specified packages and libraries:

you could try ‘??chain’

> ??chain

> submit()

| Sourcing your script...

# A tibble: 46 x 5

package count unique countries avg\_bytes

<chr> <int> <int> <int> <dbl>

1 Rcpp 3195 2044 84 2512100.

2 digest 2210 1894 83 120549.

3 stringr 2267 1948 82 65277.

4 plyr 2908 1754 81 799123.

5 ggplot2 4602 1680 81 2427716.

6 colorspace 1683 1433 80 357411.

7 RColorBrewer 1890 1584 79 22764.

8 scales 1726 1408 77 126819.

9 bitops 1549 1408 76 28715.

10 reshape2 2032 1652 76 330128.

# … with 36 more rows

| That's correct!

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

| So, the results of the last three scripts are all identical. But, the third script

| provides a convenient and concise alternative to the more traditional method that we've

| taken previously, which involves saving results as we go along.

...source('/tmp/RtmpdaoAKS/summarize4.R')

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

| Once again, let's View() the full data, which has been stored in result3.

> View(result3)

| That's the answer I was looking for.

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

| It looks like Rcpp is on top with downloads from 84 different countries, followed by

| digest, stringr, plyr, and ggplot2. swirl jumped up the rankings again, this time to

| 27th.

...

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

| To help drive the point home, let's work through a few more examples of chaining.

...

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

| Let's build a chain of dplyr commands one step at a time, starting with the script I

| just opened for you.

> submit()

| Sourcing your script...

Error in function\_list[[k]](value) : object 'ip\_id' not found

| Try again. Getting it right on the first try is boring anyway!

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

| Try again. Getting it right on the first try is boring anyway!

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> ?select

> select(cran, ip\_id)

# A tibble: 225,468 x 1

ip\_id

<int>

1 1

2 2

3 3

4 3

5 4

6 3

7 3

8 5

9 6

10 7

# … with 225,458 more rows

> select(cran, ip\_id,country)

# A tibble: 225,468 x 2

ip\_id country

<int> <chr>

1 1 US

2 2 US

3 3 US

4 3 US

5 4 CA

6 3 US

7 3 US

8 5 US

9 6 CN

10 7 US

# … with 225,458 more rows

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

# A tibble: 225,468 x 4

ip\_id country package size

<int> <chr> <chr> <int>

1 1 US htmltools 80589

2 2 US tseries 321767

3 3 US party 748063

4 3 US Hmisc 606104

5 4 CA digest 79825

6 3 US randomForest 77681

7 3 US plyr 393754

8 5 US whisker 28216

9 6 CN Rcpp 5928

10 7 US hflights 2206029

# … with 225,458 more rows

> ?select

> submit()

| Sourcing your script...

Error in ip\_id(.) : could not find function "ip\_id"

| Not quite right, but keep trying.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in function\_list[[i]](value) : attempt to apply non-function

| You're close...I can feel it! Try it again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

# A tibble: 225,468 x 4

ip\_id country package size

<int> <chr> <chr> <int>

1 1 US htmltools 80589

2 2 US tseries 321767

3 3 US party 748063

4 3 US Hmisc 606104

5 4 CA digest 79825

6 3 US randomForest 77681

7 3 US plyr 393754

8 5 US whisker 28216

9 6 CN Rcpp 5928

10 7 US hflights 2206029

# … with 225,458 more rows

| That's the answer I was looking for.

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

| Let's add to the chain.

> submit()

| Sourcing your script...

| Give it another try.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

# A tibble: 225,468 x 5

ip\_id country package size `size/2^20`

<int> <chr> <chr> <int> <dbl>

1 1 US htmltools 80589 0.0769

2 2 US tseries 321767 0.307

3 3 US party 748063 0.713

4 3 US Hmisc 606104 0.578

5 4 CA digest 79825 0.0761

6 3 US randomForest 77681 0.0741

7 3 US plyr 393754 0.376

8 5 US whisker 28216 0.0269

9 6 CN Rcpp 5928 0.00565

10 7 US hflights 2206029 2.10

# … with 225,458 more rows

| Not quite right, but keep trying.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in size\_mb(.) : could not find function "size\_mb"

| You almost had it, but not quite. Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in .f(.x[[i]], ...) : object 'size\_mb' not found

| Nice try, but that's not exactly what I was hoping for. Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in mutate(size/2^20) : object 'size' not found

| You're close...I can feel it! Try it again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> reset()

Error in UseMethod("do\_rst") :

no applicable method for 'do\_rst' applied to an object of class "c('environment', 'default')"

| Leaving swirl now. Type swirl() to resume.

> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same name as you

| did then. If you are new, call yourself something unique.

What shall I call you? blacktrojan

| Would you like to continue with one of these lessons?

1: Getting and Cleaning Data Grouping and Chaining with dplyr

2: No. Let me start something new.

Selection: 1

| Attempting to load lesson dependencies...

| Package ‘dplyr’ loaded correctly!

| Let's add to the chain.

> submit()

| Sourcing your script...

Error : object 'size\_mb' not found

| Almost! Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in source(e$script\_temp\_path, encoding = "UTF-8") :

/tmp/RtmpdaoAKS/chain2.R:10:0: unexpected end of input

8: select(ip\_id, country, package, size) %>%

9: mutate(size\_mb <- size/2^20) %>%

^

| Give it another try.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

| You almost had it, but not quite. Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in eval(lhs, parent, parent) : object 'size\_mb' not found

| One more time. You can do it!

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> ?mutate

> submit()

| Sourcing your script...

| That's not exactly what I'm looking for. Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in mutate(size/2^20) : object 'size' not found

| That's not the answer I was looking for, but try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in cran %>% select(ip\_id, country, package, size) %>% size\_mb <- mutate(cran, :

could not find function "%>%<-"

| You're close...I can feel it! Try it again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

Error in eval(lhs, parent, parent) : object 'size\_mb' not found

| Not quite! Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> submit()

| Sourcing your script...

# A tibble: 225,468 x 5

ip\_id country package size `size\_mb <- size/2^20`

<int> <chr> <chr> <int> <dbl>

1 1 US htmltools 80589 0.0769

2 2 US tseries 321767 0.307

3 3 US party 748063 0.713

4 3 US Hmisc 606104 0.578

5 4 CA digest 79825 0.0761

6 3 US randomForest 77681 0.0741

7 3 US plyr 393754 0.376

8 5 US whisker 28216 0.0269

9 6 CN Rcpp 5928 0.00565

10 7 US hflights 2206029 2.10

# … with 225,458 more rows

| That's not exactly what I'm looking for. Try again.

| Follow the directions in the script comments very carefully. If R gave you an error

| above, try to understand what it is telling you. If you get stuck, type reset() to start

| with a fresh script, then save the script and type submit() when you are ready.

> select(cran, ip\_id, country, package, size , size\_mb= size/2^20)

Error in .f(.x[[i]], ...) : object 'size' not found

> submit()

| Sourcing your script...

# A tibble: 225,468 x 5

ip\_id country package size size\_mb

<int> <chr> <chr> <int> <dbl>

1 1 US htmltools 80589 0.0769

2 2 US tseries 321767 0.307

3 3 US party 748063 0.713

4 3 US Hmisc 606104 0.578

5 4 CA digest 79825 0.0761

6 3 US randomForest 77681 0.0741

7 3 US plyr 393754 0.376

8 5 US whisker 28216 0.0269

9 6 CN Rcpp 5928 0.00565

10 7 US hflights 2206029 2.10

# … with 225,458 more rows

| You are amazing!

|============================================================================ | 94%

| A little bit more now.

> submit()

| Sourcing your script...

| That's correct!

|============================================================================== | 96%

| And finish it off.

> submit()

| Sourcing your script...

| Your dedication is inspiring!

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

| In this lesson, you learned about grouping and chaining using dplyr. You combined some

| of the things you learned in the previous lesson with these more advanced ideas to

| produce concise, readable, and highly effective code. Welcome to the wonderful world of

| dplyr!