

dplyr: How to do data manipulation with R

December 11, 2014 by [Sharp Sight Labs](#) — [13 Comments](#)

Ok. Here's an ugly secret of that data world: *lots* of your work will be prep work.

Of course, any maker, artist, or craftsman has the same issue: chefs have their mise en place. Carpenters spend a heck of a lot of time measuring vs. cutting. Etcetera.

So, you just need to be prepared that once you become a data scientist, 80% of your work will be data manipulation.

Getting data, aggregating data, subsetting data, cleaning it, and merging various datasets together: this constitutes a large percent of the day-to-day work of an analyst.

When you're just starting out with analytics and data science, you can get away with doing only minimal data manipulation. In the beginning, your datasets are likely to be txt, csv, or simple Excel files.

And if you do need to do some basic data formatting, for simple datasets you can do your data manipulation in Excel. (actually, Excel is a good tool in your workflow for basic data manipulation tasks.)

As you progress though, you'll eventually reach a bottleneck. You'll start doing more sophisticated data visualizations or machine learning techniques, and you will *need* to put your data in the right format. And you'll need a better toolset.



By most accounts, the best toolset for data manipulation with R is dplyr.

dplyr: the essential data manipulation toolset

In data wrangling, what are the main tasks?

- Filtering rows (to create a subset)
- Selecting *columns* of data (i.e., selecting variables)
- Adding new variables
- Sorting
- Aggregating

`dplyr` gives you tools to do these tasks, and it does so in a way that streamlines the analytics workflow. It's not an exaggeration to say that `dplyr` is almost perfectly suited to real analytics work, as it is actually performed.

To be clear, these aren't just the "basics." They are the *essentials*. These are tasks that you'll be doing every. single. day. You really need to master these.

Again though, `dplyr` makes them extremely easy. It's the toolset that I wish I had years ago.

Moreover, once you combine `dplyr` verbs with "chaining" (covered below) it becomes even more streamlined and more powerful. Not to mention, chaining together the data wrangling tools of `dplyr` with the data visualization tools of `ggplot`. Once you start combining these together, you will have a powerful toolset for rapid data exploration and analysis.

dplyr verbs

`dplyr` has 5 main "verbs" (think of "verbs" as commands).

filter()

Row selection from your data.

`filter()` subsets your data by keeping rows that meet specified conditions.

```
library(dplyr)
library(ggplot2)

head(diamonds)

#-----
# FILTER
#-----

df.diamonds_ideal <- filter(diamonds, cut=="Ideal")
```

Here, we're subsetting (i.e., filtering) the diamonds dataset and keeping only the rows where `cut=="Ideal"`.

select()

Select allows you to select specific *columns* from your data.

In the following code, we're going to first inspect the `df.diamonds_ideal` dataframe to see what's inside of it. Then, we'll modify our dataframe, selecting only the columns we want.

```
#-----
# SELECT
# - select specific columns from your data
#-----

# Examine the data first
head(df.diamonds_ideal)
#   carat    cut      color clarity depth table price     x     y     z
#   0.23   Ideal     E    SI2   61.5   55   326   3.95  3.98  2.43
#   0.23   Ideal     J    VS1   62.8   56   340   3.93  3.90  2.46
#   0.31   Ideal     J    SI2   62.2   54   344   4.35  4.37  2.71
#   0.30   Ideal     I    SI2   62.0   54   348   4.31  4.34  2.68
#   0.33   Ideal     I    SI2   61.8   55   403   4.49  4.51  2.78
#   0.33   Ideal     I    SI2   61.2   56   403   4.49  4.50  2.75

df.diamonds_ideal <- select(df.diamonds_ideal, carat, cut, color, price, clarity)

head(df.diamonds_ideal)
#   carat    cut      color price clarity
#   0.23   Ideal     E    326    SI2
#   0.23   Ideal     J    340    VS1
#   0.31   Ideal     J    344    SI2
#   0.30   Ideal     I    348    SI2
#   0.33   Ideal     I    403    SI2
#   0.33   Ideal     I    403    SI2
```

Notice that now `df.diamonds_ideal` only has our selected columns: `carat`, `cut`, `color`, `price`, and `clarity`.

mutate()

Mutate allows you to add variables to your dataset. Obviously, this is a really common task in any programming environment.

```
#-----
# MUTATE:
# - Add variables to your dataset
#-----

df.diamonds_ideal <- mutate(df.diamonds_ideal, price_per_carat = price/carat)

head(df.diamonds_ideal)
#   carat    cut      color price clarity price_per_carat
#   0.23   Ideal     E    326    SI2        1417.391
#   0.23   Ideal     J    340    VS1        1478.261
#   0.31   Ideal     J    344    SI2        1109.677
#   0.30   Ideal     I    348    SI2        1160.000
#   0.33   Ideal     I    403    SI2        1221.212
#   0.33   Ideal     I    403    SI2        1221.212
```

Using `mutate()`, we've successfully added a new variable: `price_per_carat`.

arrange()

Arrange sorts your data. In base R, this is commonly done with `order`, and the syntax is a bit of a pain.

Here, we're not going to use the diamonds dataset, because it's too large to properly see `order` at work.

So, we'll create a simple dataframe with a numeric variable. This numeric variable has the numbers out of order and we'll use `arrange` to reorder the numbers.

```

#-----
# ARRANGE: sort your data
#-----

# create simple data frame
# containing disordered numeric variable
df.disordered_data <- data.frame(num_var = c(2,3,5,1,4))

head(df.disordered_data)

# these are out of order

# 2
# 3
# 5
# 1
# 4

# now we'll order them with arrange()
arrange(df.disordered_data, num_var)

# 1
# 2
# 3
# 4
# 5

# we can also put them in descending order
arrange(df.disordered_data, desc(num_var))

# 5
# 4
# 3
# 2
# 1

```

This might not seem useful, but quite a bit of data-inspection involves sorting and ordering your data. This is more useful than it might seem at first blush.

summarize()

Summarize allows you to compute summary statistics.

Again: the following (simple) example might not seem useful, but `summarize()` becomes extremely useful when combined with `group_by()`.

```

#-----
# SUMMARIZE:
# aggregate your data
#-----

summarize(df.diamonds_ideal, avg_price = mean(price, na.rm = TRUE) )

# avg_price
# 3457.542

```

“Chaining” in dplyr

Moving beyond the simple examples above, the real power of `dplyr` begins to show itself when you start to “chain” different verbs together (or, chain different `dplyr` verbs together with commands and functions from other packages).

The %>% operator

In the `dplyr` syntax, we “chain” commands together using the `%>%` operator. This is very similar to the “pipe” operator in Unix.

We use the `%>%` operator to *connect* one command to another. The output of one command

becomes the input for the next command.

Here's an example:

```
# Subset the diamonds dataset to 'Ideal' cut diamonds
# THEN, keep (select) the variables: carat, cut, color, price, clarity
# THEN, add new variable called price_per_carat (mutate)

df.diamonds_ideal_chained <- diamonds %>%
  filter(cut=="Ideal") %>%
  select(carat, cut, color, price, clarity) %>%
  mutate(price_per_carat = price/carat)

head(df.diamonds_ideal_chained)
# carat cut color price clarity price_per_carat
# 0.23 Ideal E 326 SI2 1417.391
# 0.23 Ideal J 340 VS1 1478.261
# 0.31 Ideal J 344 SI2 1109.677
# 0.30 Ideal I 348 SI2 1160.000
# 0.33 Ideal I 403 SI2 1221.212
# 0.33 Ideal I 403 SI2 1221.212
```

Here we've created a new, reshaped dataset in 4 streamlined lines of code.

To be more specific, we've "chained" together multiple commands and directed the output of that set of commands into a new dataframe called `df.diamonds_ideal_chained`.

Let's also review how we read this code.

We read the `%>%` operator as "then." So, you can read the code out-loud to yourself, saying:

- "Take the `diamonds` dataset"
- "Then filter it, keeping only the rows where 'cut' equals 'Ideal' "
- "Then select specific variables, 'carat', 'cut', 'color', 'price', 'clarity' "
- "Then create a new variable, 'price_per_carat' using 'mutate()' "

Rapid Data Exploration with dplyr and ggplot

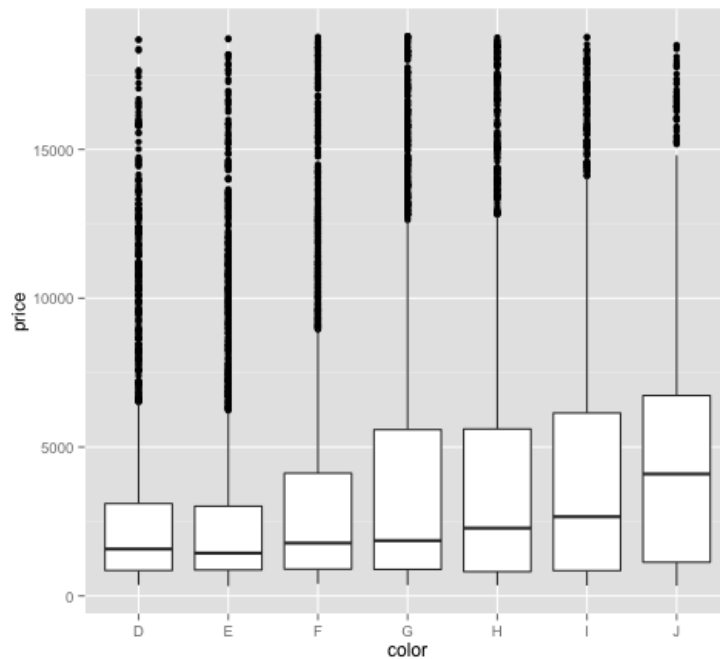
To be honest, the above example is somewhat simple. Where this begins to be more useful is creating much longer chains where you filter, aggregate, select, add variables, and visualize, all in one fell swoop.

Chaining commands from `dplyr` and `ggplot2` creates a powerful tool for rapid data exploration.

Example 1: Boxplot, dplyr + ggplot

```
# dplyr + ggplot
# PRICE DISTRIBUTION, Ideal-cut diamonds
diamonds %>%
  filter(cut == "Ideal") %>%
  ggplot(aes(x=color,y=price)) +
  geom_boxplot()

# Start with the 'diamonds' dataset
# Then, filter down to rows where cut == Idea
# Then, plot using ggplot
# with and create a boxplot
```



This should give you a sample of what's possible: using dplyr "chained" together with ggplot to perform rapid data exploration.

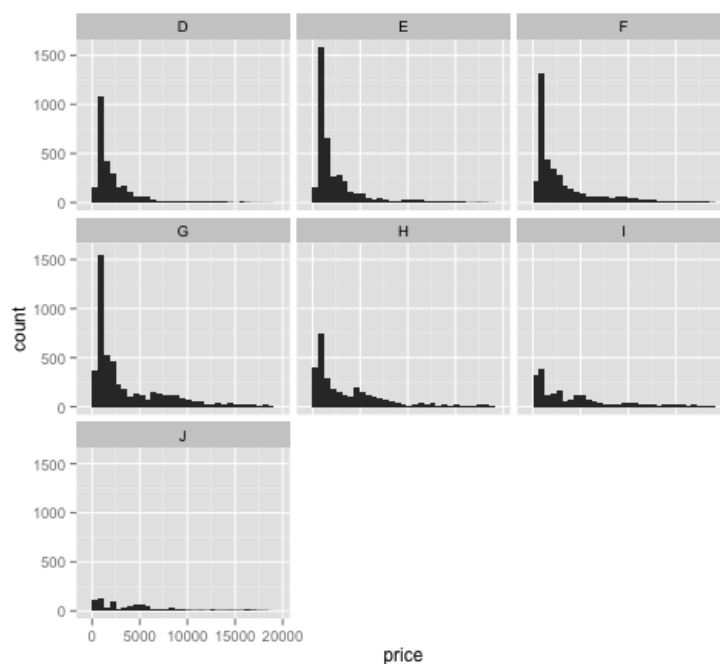
Example 2: Small Multiple Histogram, dplyr + ggplot

Next, we'll create a **histogram** of 'ideal cut' diamonds, in a **'small multiple' layout**.

(Small multiples are called 'facets' in ggplot. They are also sometimes called a 'trellis charts'.)

```
# dplyr + ggplot
# HISTOGRAM of price, ideal cut diamonds
diamonds %>%
  filter(cut == "Ideal") %>%
  ggplot(aes(price)) +
    geom_histogram() +
    facet_wrap(~ color)

# Start with the 'diamonds' dataset
# Then, filter down to rows where cut == Ideal
# Then, plot using ggplot
# and plot histograms
# in a 'small multiple' plot, broken out by 'color'
```



Here, in only 5 lines of code, we've subsetting our data and created a fairly decent looking

'small multiples' chart. If we wanted, we could add a few more lines of code to format this and make it look more presentable (but, we won't do that here).

This should give you a minor sense of how powerful dplyr and ggplot are when used together. I wrote this code in well under 5 minutes. It could have taken hours to do in Excel.

dplyr: part of 'foundational' data science


If you're just getting started with data science, I recommend that you master the following (assuming you're starting with R. Which you should.):

- the 5 dplyr verbs
- the "big 3" visualizations: [the line chart](#), [the bar chart](#), and [the scatterplot](#)
- [The small multiple design](#) (i.e., faceting)
- chaining, using the `%>%` operator

These skills form the foundation for more sophisticated work. And, they're really easy to learn in R.

For another take on dplyr, I'd highly recommend the following tutorial as well, at [Getting Genetics Done](#).

Want to become a data scientist?

 Enter Your Email

SIGN ME UP!

Data science is one of the hottest careers in tech. Enter your email now and we'll send you free data science tutorials, every week.

We hate spam. We'll only send you useful stuff, and won't share your email with anyone.

Filed Under: [dplyr](#), [ggplot2](#)

Comments



Sid says
[January 27, 2015 at 10:48 pm](#)

Extremely useful!!

[Reply](#)



Joe says
[February 18, 2015 at 9:45 pm](#)

This is probably the best introduction to dplyr that I have found! Thank you for your work!

[Reply](#)



Neha Gupta says

March 17, 2015 at 7:06 am

Thank you....It is very useful and helped me to understand the use of dplyr package.

[Reply](#)

Trackbacks

Data analysis example - supercar data (code and tutorial) says:

December 16, 2014 at 10:01 am

[...] The data is rich enough to answer those questions, but it needs some data manipulation to extract the exact variables from it; which means, we'll be able to use some dplyr verbs to manipulate and reshape our data. [...]

How to start learning data science - Sharp Sight Labs says:

December 18, 2014 at 4:50 pm

[...] dplyr for data manipulation [...]

Data exploration with ggplot2 and dplyr (code and tutorial) says:

December 23, 2014 at 6:29 pm

[...] Created a summarized variable using summarize() [...]

Why you should learn R for data science - Sharp Sight Labs says:

January 27, 2015 at 10:47 am

[...] dplyr package in R makes data manipulation easy. It is the tool I wish I had years ago. When you "chain" the basic dplyr together, you [...]

How data visualizations are tools (and what you're building with them) - SHARP SIGHT LABS says:

February 3, 2015 at 3:36 pm

[...] (within weeks, for many students) you can add dplyr to start manipulating your data and putting it into the right format. The best part is that when you begin to chain dplyr verbs together with ggplot2, you can find [...]

Learn data visualization and manipulation first - Sharp Sight Labs says:

February 10, 2015 at 1:38 pm

[...] will allow you to subset your data, aggregate it, and otherwise transform your data to help you find more insights (you can also use data manipulation techniques to merge in new data, [...]

Why You Should Learn R First for Data Science - Dataconomy says:

February 16, 2015 at 3:04 pm

[...] dplyr package in R makes data manipulation easy. It is the tool I wish I had years ago. When you "chain" the basic dplyr together, you [...]

Information = Data + R says:

February 23, 2015 at 9:46 pm

[...] dplyr package in R makes data manipulation easy. It is the tool I wish I had years ago. When you "chain" the basic dplyr together, you can [...]

Summer Internships for BTech CSE, IT Branch students 2015 says:

March 7, 2015 at 3:34 am

[...] dplyr package in R makes data manipulation easy. It is the tool I wish I had years ago. When you "chain" the basic dplyr together, you can [...]

dplyr: How to do data manipulation with R | Drawing the Data says:

March 7, 2015 at 5:11 pm

[...] <http://www.sharpsightlabs.com/dplyr-intro-data-manipulation-with-r/> [...]

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: ` <abbr title=""> <acronym title=""> <blockquote cite=""> <cite> <code> <del datetime=""> <i> <q cite=""> <strike> `

POST COMMENT

Want to become a data scientist?

Demand for data scientists is growing rapidly with the explosion of big data.

Learn the skills for the hottest career in tech.

Enter your email address now and we'll send you free, step-by-step tutorials every week.

SIGN ME UP!

You'll get ...

- ☐ A free "Getting Started with Analytics and Data Science" pdf.
- ☐ Free, data science tutorials (weekly)
- ☐ Updates on data-industry job trends

Recommended Reading

R Bloggers

Flowingdata

StatsBlogs

Subscribe to receive our free "Getting Started with Analytics and Data Science" pdf.

GET STARTED!

Copyright © 2015 · [Genesis Framework](#) · [WordPress](#) · [Log in](#)

