How to analyze a new dataset (or, analyzing 'supercar' data, part 1)

December 16, 2014 by Sharp Sight Labs — 4 Comments

I love cars.

The way they sound. The engineering. The craftsmanship. And let's be honest: fast cars are just *fun*.

Given my love of cars, I frequently watch Top Gear clips on YouTube.

A couple of weeks ago, I stumbled across this:

Watching the video, I'm thinking, "253 miles per hour? *You've got to be kidding me.*"

And the next thing you know, I'm scraping 'high-performance' car data from a website.

Data analysis example: 'supercar' data

After gathering the data together, I realized it would be a great dataset to use for a data analysis example.

When I initially saw the data on the website, I was already asking questions: how much horsepower does that Bugatti have? How much torque? How much compared to other cars?

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**.

Moreover, the data was scattered across several smaller datasets, making it perfect for demonstrating the **process of merging data together**. Note that I haven't published any tutorials on "joins" yet, so just follow along during that section. I'll publish a tutorial on joins later. In the meantime, this will give you a preview.

And finally, this data gives us the opportunity to **put some of our visualization skills to use: creating scatterplots, histograms, bar charts, and small multiples**. Moreover, we'll be able to use these tools *to practice data exploration*. Remember: finding insight in data is an art, and that art must be practiced.

The following data analysis example will show you the rough process for analyzing data, end-to-end.

When I say "rough process," what I mean is that this isn't comprehensive. At every step, there might be more to do (e.g., get more data, do more visualizations, "polish" the charts for presentation). Having said that, even though it doesn't show everything we might do, it does take you through the overall process at a high level.

Ok, let's get started.

Get the data

First, we're just going to get our data from 5 separate .txt files. (These files are available here on the Sharp Sight Labs blog; you should be able to access directly them via the following code.)

Keep in mind that .txt files are sort of a simple case. As you progress, your data might be in a database (requiring you to write some SQL code) or it might be on a website that you need to scrape.

We'll keep this simple though: these datasets are already in comma-delimited text files.

(Also note: I scraped these from a website. I did some data wrangling to reshape the original data into the following .txt files. That process was a bit more complicated though, so I've left it out of the tutorial.)

In the following code, we're just going to import the files into dataframes using `read.csv()`.

```
library(dplyr)
library(ggplot2)

###################
# IMPORT datasets #
###################


df.car_torque <- read.csv(url("http://www.sharpsightlabs.com/wp-content/uploads/2014/11/auto-snou
df.car_0_60_times  <- read.csv(url("http://www.sharpsightlabs.com/wp-content/uploads/2014/11/autc
df.car_engine_size <- read.csv(url("http://www.sharpsightlabs.com/wp-content/uploads/2014/11/autc
df.car_horsepower  <- read.csv(url("http://www.sharpsightlabs.com/wp-content/uploads/2014/11/autc
df.car_top_speed   <- read.csv(url("http://www.sharpsightlabs.com/wp-content/uploads/2014/11/autc
df.car_power_to_weight <- read.csv(url("http://www.sharpsightlabs.com/wp-content/uploads/2014/11/


#-------------------------
# Inspect data with head()
#-------------------------
head(df.car_torque)
head(df.car_0_60_times)
head(df.car_engine_size)
head(df.car_horsepower)
head(df.car_top_speed)
head(df.car_power_to_weight)
```

Examine the data

Next, examine the data.

Specifically, you're going to look for duplicate records. The reason is that we're going to join these together into one dataset. Duplicate records will cause a faulty join.

```
##############################
# LOOK FOR DUPLICATE RECORDS #
##############################
df.car_torque      %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)
df.car_0_60_times  %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)
df.car_engine_size %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)
df.car_horsepower  %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)
df.car_top_speed   %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)
df.car_power_to_weight %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)


# NOTE: duplicate records!
# several datasets have duplicate rows for the following cars (car_full_nm)

# Chevrolet Chevy II Nova SS 283 V8 Turbo Fire - [1964]     2
#           Koenigsegg CCX 4.7 V8 Supercharged - [2006]     2
#                   Pontiac Bonneville 6.4L V8 - [1960]     2
```

Here, to look for duplicate records, we've chained together several dplyr verbs. In this process, we're aggregating each dataframe by "car_full_nm," counting the number of records for each car, and then filtering the resulting data, leaving only cars that have more than one record.

If you don't understand this, that's OK. The dplyr tutorial explains chaining, and all of these verbs. So if you don't understand this code, read that tutorial.

Remove duplicate records to prep for 'joins' (i.e., dedupe)

Since we found some duplicate records, we want to remove those.

We'll use the `distinct()` function to remove records that have the same value for car_full_nm.

```
###############
# DEDUPE DATA #
###############

df.car_0_60_times  <- distinct(df.car_0_60_times ,car_full_nm)
df.car_engine_size <- distinct(df.car_engine_size ,car_full_nm)
df.car_horsepower  <- distinct(df.car_horsepower ,car_full_nm)
df.car_top_speed   <- distinct(df.car_top_speed ,car_full_nm)
df.car_torque      <- distinct(df.car_torque ,car_full_nm)
df.car_power_to_weight <- distinct(df.car_power_to_weight, car_full_nm)
```

Join datasets together into one data frame

Now that our dataframes are deduped, we'll join them together.

Joins are an an entirely separate tutorial all by themselves, so I won't discuss this code in detail.

Having said that, you'll notice that I'm:
1. Starting with df.car_horsepower and joining df.car_torque to that. I'm joining these two datasets together on the car_full_nm variable. This joined dataframe is called df.car_spec_data.

2. Then, one by one, I'm joining all of the datasets to df.car_spec_data to create a "master" dataset.

Note also that after each line of code, I've added a comment (using the '#' character), indicating the number of records. This is a simple check to make sure that the join worked properly. Because of the nature of `left_join()`, we're expecting the same number of records at every step. If at any step, we saw *a larger number of records*, that would be an indication that something went wrong. We'd have to recheck our work.

```
#############
# JOIN DATA #
#############

df.car_spec_data <- left_join(df.car_horsepower, df.car_torque, by="car_full_nm")      # count af
df.car_spec_data <- left_join(df.car_spec_data, df.car_0_60_times, by="car_full_nm")   # count af
df.car_spec_data <- left_join(df.car_spec_data, df.car_engine_size, by="car_full_nm")  # count af
df.car_spec_data <- left_join(df.car_spec_data, df.car_top_speed, by="car_full_nm")    # count af
df.car_spec_data <- left_join(df.car_spec_data, df.car_power_to_weight, by="car_full_nm") # count


# Test duplicates
df.car_spec_data      %>% group_by(car_full_nm) %>% summarise(count=n()) %>% filter(count!=1)

# local data frame [0 x 2]
# i.e., 0 duplicate records


# Re-inspect data
str(df.car_spec_data)
head(df.car_spec_data)
```

Add new variables

Next, you'll add new variables using the `mutate()` function.

You can learn more about `mutate()` in the dplyr tutorial.

Additionally, you'll notice some arcane syntax inside of the `sub()` function within the `mutate()` call.

`sub()` uses regular expressions to parse a character variable, and replace part of a string with something else.

The topic of regular expressions is a little more advanced, so don't worry about it right now. Really, if you don't know regular expressions, you might be tempted to spend a lot of time trying to figure it out. Learning regular expressions isn't a great use of your time right now. Just let it work it's magic and we'll come back to the topic another time. For the time being, try to focus on the overall process.

```
####################
# ADD NEW VARIABLES
####################

#--------------
# NEW VAR: year
#--------------

df.car_spec_data <- mutate(df.car_spec_data, year=sub(".*\\[([0-9]{4})\\]","\\1",car_full_nm))

str(df.car_spec_data$year)


#----------------
# NEW VAR: decade
#----------------

df.car_spec_data <- mutate(df.car_spec_data,
                           decade = as.factor(
                                    ifelse(substring(df.car_spec_data$year,1,3)=='193','1930s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='194','1940s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='195','1950s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='196','1960s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='197','1970s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='198','1980s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='199','1990s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='200','2000s',
                                    ifelse(substring(df.car_spec_data$year,1,3)=='201','2010s',"E
                                    )))))))))
                                    )
                            )

head(df.car_spec_data)
str(df.car_spec_data)


#--------------------------------
# NEW VAR: make_nm
#  (i.e., the "make" of the car;
#  the "brand name" of the car)
#--------------------------------

df.car_spec_data <- mutate(df.car_spec_data, make_nm = gsub(" .*$","", df.car_spec_data$car_full_


#--------------------------
# NEW VAR: car_weight_tons
#--------------------------

df.car_spec_data <- mutate(df.car_spec_data, car_weight_tons = horsepower_bhp / horsepower_per_to


#--------------------------
# NEW VAR: torque_per_ton
#--------------------------

df.car_spec_data <- mutate(df.car_spec_data, torque_per_ton = torque_lb_ft / car_weight_tons)
```

Inspect data again

Now that we've added our variables, we'll just check our data again.

Notice that we're *doing that a lot*.

At every step, you want to look and see if the data set turned out the way you expected.

Specifically, we're going to check that we created the new variables properly. We're going to accomplish that by using `dplyr` verbs again, chaining together several verbs with the `%>%` operator. We'll use these chained verbs to generate a frequency table (a list of categorical variable values, and the counts for those values).

```
#########################################
# INSPECT DATA
#  - quick checks to make sure that
#    the variables were created properly
#########################################

head(df.car_spec_data)

#-----------------------------
# Frequency table (AGGREGATE)
#  - decade
#-----------------------------

# CHECK 'decade' variable
df.car_spec_data %>%
  group_by(decade) %>%
  summarize(count=n())

# decade count
# 1  1930s     2
# 2  1940s     7
# 3  1950s    57
# 4  1960s   143
# 5  1970s   125
# 6  1980s   154
# 7  1990s   262
# 8  2000s   526
# 9  2010s   302


#-----------------------------
# Frequency table (AGGREGATE)
#  - make
#-----------------------------

df.car_spec_data %>%
  group_by(make_nm) %>%
  summarise(make_count = length(make_nm)) %>%
  arrange(desc(make_count))

# note: the list of car makes is too long so the output hasn't
#       been added here
```

Using several dplyr verbs chained together, we've taken the df.car_spec_data data frame, grouped it by decade, then counted the number of records by decade. We're doing this to check that we created our 'decade' variable correctly; if we found an incorrect value for 'decade,' or a count that looked way off, we'd have to go back and check our work.

We're doing the same thing with the 'make' variable: chaining together several dplyr verbs to aggregate, count, and create a frequency table.

(I've omitted the output of the car-make frequency table. The output is a long list. Just run the code and you'll see the output yourself.)

Recap: creating a master dataset

Now we have a dataset we can work with.

Let's recap what we just did:
1. Imported multiple .txt files with `read.csv()`
2. Inspected the data
3. Removed duplicate records from each data set (deduped)
4. Merged the datasets together into a single dataframe
5. Created new variables
6. Inspected again

Keep this process in mind. When you're initiating a new analysis and creating your dataset, this is a rough outline of the steps you'll probably need to execute.

Want to see part 2?

In part 2 of this data analysis example, I'm going to show how to explore this data using the core data visualization techniques.

If you want to see part two as soon as it's published, sign up for our email list and we'll send the link directly to you, so you don't miss it.

Filed Under: dplyr, r-bloggers

Comments

Daryle says
December 19, 2014 at 1:17 pm

Nice tutorial. It's really cool how most of the ddplyr functions work very much like SQL statements.

Reply

Sharpsight_Admin says
December 19, 2014 at 2:07 pm

Yeah, the dplyr verbs are really very close to SQL.

The benefit over SQL though comes from being able to chain them together in increasingly complex ways. You can do some clever aggregations and reshapes on your data and then pipe the output directly into ggplot2. When you use this combination of dplyr+ggplot2, you can explore your data very quickly (more on that in part 2).

Reply

Trackbacks

How to start learning data science – Sharp Sight Labs says:
December 18, 2014 at 4:51 pm

[...] As you advance though, the “shape” of your data will be a problem: you’ll have multiple data files that you need to join together; you’ll need to subset and change variables; you’ll need to do lots of aggregations. When you reach this point – where you the shape of your data is a bottleneck – then put more time into learning data manipulation. An example of this is the recent tutorial analyzing ‘supercar’ data, where the data were found in five separate files. [...]

Data exploration with ggplot2 and dplyr (code and tutorial) says:
December 23, 2014 at 6:29 pm
[...] post is a continuation of analyzing ‘supercar’ data part 1, where we create a dataset using R’s dplyr package. To learn how we created our dataset, [...]

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: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

POST COMMENT

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.

Your first name

Your best email address

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.

First Name

E-Mail Address

GET STARTED!