Introduction to R for Libraries

Part 3: Data Analysis and Visualization

Written by Clarke Iakovakis. Webinar hosted by ALCTS, Association for Library Collections & Technical Services

May, 2018

Contents

1 Introduction

This guide coves Part 3 of the *Introduction to R for Libraries* webinar, hosted by ALCTS, Association for Library Collections & Technical Services. It is distributed under a https://creativecommons.org/licenses/by/4.0/.

2 Cleaning up strings with stringr

There is no end to the number of problems you will run into when cleaning your data. It is a truism that data cleanup and preparation generally takes 80% of the time in a data project, leaving the remaining 20% for analysis and visualization.

Character strings are sequences of characters. For libraries, common strings include titles, authors, subject headings, ISBNs (even though they are digits, they are treated as characters because they are identifiers), publishers, vendors, and more. Strings can be one of the most problematic parts of the data cleanup process, which makes `stringr` one of the most useful packages in R. `stringr` contains a number of functions to manipulate characters, trim whitespace from the beginning and ending of a string, and execute pattern matching operations (for example, recognize all 13 digit numbers beginning with 978 as ISBNs).

Below we will explore one small example of a string cleanup issue in the **ebooks.csv** file, included in the course documents.

2.1 Replace characters with str_replace()

First read the data in. Notice we have to use the `colClasses` argument on the ISBN fields, otherwise they're read in incorrectly as integers. Use `View()` to examine the data, and call `str()` on it to see a glimpse of it in the console.

```
ebooks <- read.csv("./ebooks.csv"
                   , stringsAsFactors = F
                   , colClasses = c(ISBN.Print = "character"
                                    , ISBN.Electronic = "character"))
View(ebooks)
str(ebooks)
```

There are several fields here we can do some analysis with, including **Pages.Viewed** and **User.Sessions.** Let's say we want to see the mean number of user sessions:

```
mean(ebooks$User.Sessions)
## Warning message: In mean.default(ebooks$User.Sessions) : argument is
## not numeric or logical: returning NA
```

Calling `class(ebooks$User.Sessions)` we see that R read this variable in as **character,** as it did with Total.Pages, Pages.Viewed, and Pages.Printed. You cannot take the mean of a character, only a **numeric** or **integer**. The reason R read these variables in as characters is because the vendor put a comma in numbers 1,000 and higher. R detected the comma, and determined that these vectors are **characters**, not **numeric.**

Now, we could coerce it to integer by calling `ebooks$User.Sessions <- as.integer(ebooks$User.Sessio` however this would be a serious error. If we coerced this field to integer, it would replace any value including a comma with `NA`, which means we would have no data for our highest usage items!

So we need to do two things:

1. Remove the comma
2. Coerce the vectors from characters to integers using `as.integer()`

The `str_replace` function is similar to **Find/Replace** in Microsoft Excel. Call `help(str_replace)` to see it has three arguments: `string`, `pattern`, and `replacement`. Remember you cannot feed an entire **data frame** to this function. You must work with one vector at a time.

```
vec <- c("800", "900", "1,000")

# Remember you have to load stringr first.  When you load tidyverse,
# stringr is automatically loaded
library(tidyverse)

# scan through each item in vec and replace the pattern (a comma: ',')
# with nothing (an empty set of quotes '')
vec2 <- str_replace(vec, pattern = ",", replacement = "")
vec2
## [1] '800' '900' '1000'

# notice that this is still a character vector you can also tell by the
# quotation marks around the numbers.
class(vec2)
## [1] 'character'

# so we must coerce it to an integer
vec2 <- as.integer(vec2)
```

```
# now we can perform mathematical operations on it
mean(vec2)
## [1] 900
```

We can do exactly the same thing on the ebooks data, except on multiple vectors at once using the `mutate()` function from `dplyr`. Note that you do not have to include the `pattern =` and `replacement =` in your argument. This will overwrite the existing variables Total.Pages, Pages.Viewed, Pages.Printed, and User.Sessions and remove the commas in all these fields.

Of course, that means those vectors will still be in character class, which means we can't do mathematical operations on them. So in sum, the easiest way to handle this is to do it all at once, by nesting the `str_replace` function within `as.integer`.

```
ebooks <- mutate(ebooks
          , Total.Pages = as.integer(str_replace(ebooks$Total.Pages, ",", ""))
          , Pages.Viewed = as.integer(str_replace(ebooks$Pages.Viewed, ",", ""))
          , Pages.Printed = as.integer(str_replace(ebooks$Pages.Printed, ",", ""))
          , User.Sessions = as.integer(str_replace(ebooks$User.Sessions, ",", "")))
```

2.2 Extract characters with str_sub()

Another data manipulation task we can do is to create a new variable **LC.Class** using the `str_sub()` function. This function takes a vector, a starting character, and an ending character, and grabs the characters in between those points (i.e. a **substring**). For example:

```
vec3 <- "Macbeth"

# Start at character 1 and end at character 3
str_sub(vec3, 1, 3)
## [1] "Mac"
```

With our LC class, since we only want the first character, our starting point and ending point are both 1. We can put this into `mutate()` from the `dplyr` package to create the new variable all at once.

```
ebooks <- mutate(ebooks
                 , LC.Class = str_sub(LC.Call
                                      , start = 1
                                      , end = 1))
```

2.3 Remove whitespace with str_trim()

Sometimes blank spaces can end up before or after a string. You can remove these automatically with the `str_trim()` function. The opposite of `str_trim()` is `str_pad()`, which will add white space.

```
vec4 <- c(" a", "b ", " c ")

str_trim(vec4)
## [1] "a" "b" "c"
```

2.4 Detect patterns with str_detect()

The `str_detect()` function takes a character vector and a pattern, and looks for that pattern in each element in the vector. If the pattern is found, a TRUE value is returned. Use `str_detect()` in combination with the `filter()` function from `dplyr` to creat custom subsets

```
titles <- c("Macbeth", "Dracula", "1984")

str_detect(titles, "Dracula")
## [1] FALSE  TRUE FALSE

# use in combination with which()
which(str_detect(titles, "Dracula"))
## [1] 2

# or use in combination with filter to create a subset
# this will scan through the Title column in the ebooks dataset
# and return all values that contain the word "Chemistry"
ebooksScience <- filter(ebooks
                        , str_detect(Title, "Chemistry"))

head(ebooksScience$Title)
## [1] "Specialist Periodical Reports, Volume 29 : Organometallic Chemistry"
## [2] "Advanced Organic Chemistry, Part A:  Structure and Mechanisms (4th Edition)"
## [3] "Advanced Organic Chemistry, Part B : Reactions and Synthesis (4th Edition)"
## [4] "Guide to Chalogen-Nitrogen Chemistry"
## [5] "Bioanalytical Chemistry"
## [6] "Environmental Laboratory Exercises for Instrumental Analysis and Environmental C

# the ignore_case argument will get the word regardless of upper or lower case
ebooksScience <- filter(ebooks
                        , str_detect(Title, fixed("Chemistry", ignore_case = T)))
```

```r
# Perhaps it will be more effective to also get books where the word chemistry
# appears in the Sub.Category as well.
# Here I use the | symbol to represent a Boolean OR
ebooksScience2 <- ebooks %>%
  filter(str_detect(Title, fixed("Chemistry", ignore_case = T))
         | str_detect(Sub.Category, fixed("Chemistry", ignore_case = T)))
```

2.5 Help with `stringr`

- Read the chapter on strings in *R for Data Science*
- Visit http://stringr.tidyverse.org/ to read uses of stingr
- Run `vignette("stringr")` in your console
- You can leverage `stringr` by learning **Regular Expressions** (aka regex). Take a regex tutorial at https://regexone.com/ and test your own regular expressions at https://regexr.com/.

TRY IT YOURSELF

1. Execute the following operations and see what happens

```r
titles <- c("Dracula", "Macbeth", "1984", "Jane Eyre", "The Great Gatsby",
    "The Iliad", "Moby Dick")
str_detect(titles, "The")
titles[str_detect(titles, "The")]
str_subset(titles, "The")
str_to_lower(titles)
str_to_upper(titles)
str_c(titles, collapse = ", ")
str_length(titles)
which(str_length(titles) >= 9)
titles[str_length(titles) >= 9]
titles[str_length(titles) < 9]

x <- c("a ", " b", " c ")
str_trim(x)
```

2. Click on the Packages tab in the navigation pane in R Studio, then click on stringr. Read some of the help pages for these functions.
3. With the ebooks data, use `str_detect()` with `filter()` to get all books with the word "Britain" in the title. Write another expression to get all books with Britain in the title or in the sub category.

---

3 Group and summarize data

Last session we reviewed several `dplyr` functions including `filter()`, `select()`, `mutate()` and `arrange()`.

`group_by()` and `summarize()` are two more powerful `dplyr` functions. Use `group_by()` to cluster the dataset together in the variables you specify. For example, if you group by LC Call number class, then you can sum up the total number of ebook user sessions per call number class using `summarize()`. `summarize()` will collapse many values down into a single summary statistics.

Here is an example using the **LC.Class** variable created in Section 1.2 above.

```
# Make sure you have created the LC.Class variable
ebooks <- mutate(ebooks
                 , LC.Class = str_sub(LC.Call
                                      , start = 1
                                      , end = 1))

# First, create the grouped variable
byLC <- group_by(ebooks, LC.Class)

# Then, summarize by User.Sessions
UserSessions_byLC <- summarize(byLC, totalSessions = sum(User.Sessions))
View(UserSessions_byLC)

# Arrange it
UserSessions_byLC <- arrange(UserSessions_byLC, desc(totalSessions))
View(UserSessions_byLC)
```

We can use the %>% operator to do both these operations at the same time. Doing it this way means we do not have to create an intermediate `byLC` data frame, but can instead operate directly on `ebooks`.

```
UserSessions_byLC <- ebooks %>%
  group_by(LC.Class) %>%
  summarize(totalSessions = sum(User.Sessions)) %>%
  arrange(desc(totalSessions))
```

The top 5 are H, Q, B, T, and L. If we were ambitious, we could use the `recode()` function to assign each call number class with its classification name:

```
recode(UserSessions_byLC$LC.Class
       , H = "Social Sciences"
       , Q = "Science"
       , B = "Philosophy"
       , T = "Technology"
       , L = "Education")
```

We may also want to see the total number of books in each call number class, and also calculate the mean user sessions per class:

- use `n()` to sum up the total number of ebooks per call number class. Add it as a column called `count`.
- use `sum(User.Sessions)` to add up the cumulative total user sessions per call number class. Add it as a column called `totalSessions`.
- use `mean(User.Sessions)` to calculate the mean user sessions per call number class. Add it as a column called `avgSessions`.

```
byLCSummary <- byLC %>%
  summarize(
    count = n()
    , totalSessions = sum(User.Sessions)
    , avgSessions = mean(User.Sessions)) %>%
  arrange(desc(avgSessions))
```

Additional summary statistics include:

- `median()`
- `sd()` : standard deviation
- `IQR()` : interquartile range
- `min()` and `max()` : minimum and maximum
- `quantile()`
- `distinct_n()` : `dplyr` function for number of unique values

So we can construct an extensive summary data frame. Notice that we are sorting by `avgSessions` in descending order.

```
byLCSummary <- ebooks %>%
  group_by(LC.Class) %>%
  summarize(
    count = n()
    , totalSessions = sum(User.Sessions)
    , avgSessions = mean(User.Sessions)
    , medianSessions = median(User.Sessions)
    , sdSessions = sd(User.Sessions)
    , highestSession = max(User.Sessions)
    , lowestSession = min(User.Sessions)
    , varianceSessions = var(User.Sessions)) %>%
  arrange(desc(avgSessions))
```

Calling `View(byLCSummary)` to have a look at this, a few items of interest jump out; for example:

- Call # Q has a very high standard deviation, max value, and variance.
- Call # P has a relatively low average, standard deviation, and highest session, but a high count and high number of total sessions.

There is, as we will see in a below visualization, an extreme outlier in the Q category. What you do with outliers is dependent on the parameters of your own analysis. We can call `filter()` if we want to get rid of it, but we have to go back to the raw data, since `byLCSummary` is already summarized.

```
byLCSummary2 <- ebooks %>%
  filter(User.Sessions < 1123) %>%
  group_by(LC.Class) %>%
  summarize(
    count = n()
    , totalSessions = sum(User.Sessions)
    , avgSessions = mean(User.Sessions)
    , medianSessions = median(User.Sessions)
    , sdSessions = sd(User.Sessions)
    , highestSession = max(User.Sessions)
    , lowestSession = min(User.Sessions)
    , varianceSessions = var(User.Sessions)) %>%
  arrange(desc(avgSessions))
```

Now we see that Q drops down to 6th highest, and its standard deviation goes from over 1100 to 107.

In sum, `group_by()` and `summary()` can produce insightful information into patterns on a variable level. See Section 5.6 of *R For Data Science* for more information on this.

TRY IT YOURSELF

1. Using the pipe `%>%` operator, `group_by` the **Category** variable and `summarize` by the **Pages.Viewed** variable. Create columns for count, totalPagesViewed, avgPagesViewed, medianPagesViewed, sdPagesViewed, maxPagesViewed, minPagesViewed, and variancePagesViewed. `arrange` by avgPagesViewed. You can assign it to `CategorySummary`. Which categories are at the top and bottom?
2. Looking in the `count` variable, we can see that some of these categories don't have very many ebooks. Use `filter()` on **CategorySummary** where count is more than 20. You can assign it over `CategorySummary` or create a new object.
3. `arrange` by totalPagesViewed. Do you notice any differences?

4 Visualizing data with ggplot2

Base R contains a number of functions for quick data visualization such as `plot()`, `barplot()`, `hist()`, and `boxplot()`. However, just as data manipulation is easier with `dplyr` than Base R, so data visualization is easier with `ggplot2` than Base R. `ggplot2` is a "grammar" for data visualization also created by Hadley Wickham, as an implementation of Leland Wilkinson's [Grammar of Graphics](http://www.worldcat.org/oclc/934513040.

When you run the `ggplot` function, it plots directly to the Plots tab in the Navigation Pane (lower right). Alternatively, you can assign a plot to an R object, then call `print()` to view the plot in the Navigation Pane.

The `ggplot()` function creates the initial coordinate system, which you can consider a "blank canvas".

```
# first load the package
library(ggplot2)

# print a blank canvas
ggplot(data = ebooks)
```

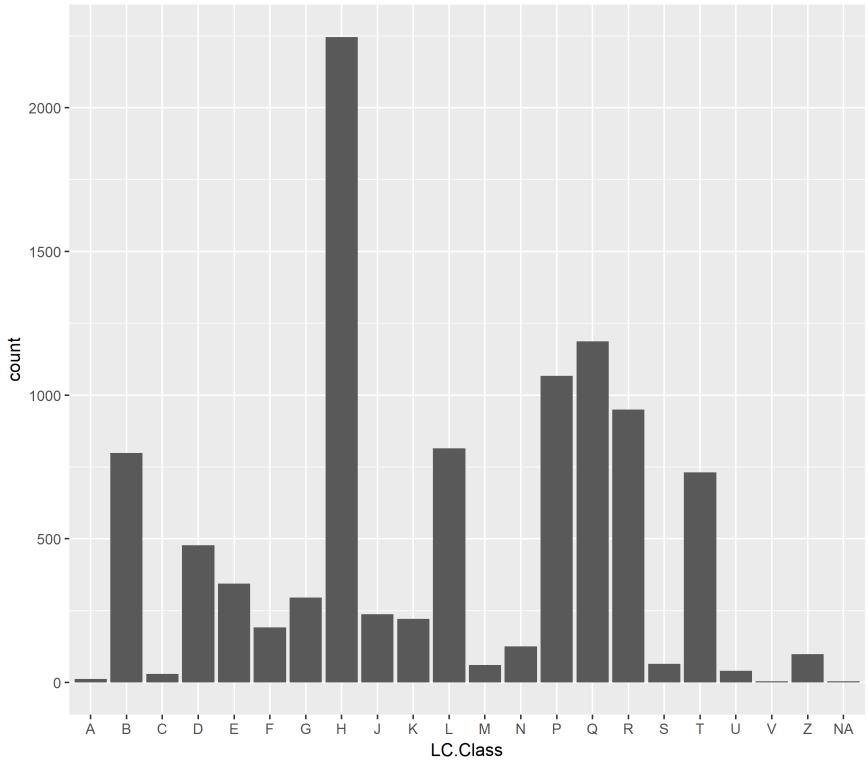

Not very interesting. We need to add layers to it. We start with `geoms`.

4.1 Display data with `geoms`

Data is visualized in the canvas with "geometric shapes" such as bars and lines; what are called **geoms**. In your console, type `geom_` and press the tab key to see the geoms included–there are over 30. For example, bar plots use bar geoms (`geom_bar()`), line plots use `geom_line()`, scatter plots use `geom_point()`.

Add layers to your plot by adding a `geom` with the plus symbol `+`. Below, we use `geom_bar()` to add a bar plot to the canvas:

```
ggplot(data = ebooks) +
  geom_bar(mapping = aes(x = LC.Class))
```



As we will see, if you add different geoms to `ggplot()` your visualization will change accordingle.

4.2 Map aesthetics with aes()

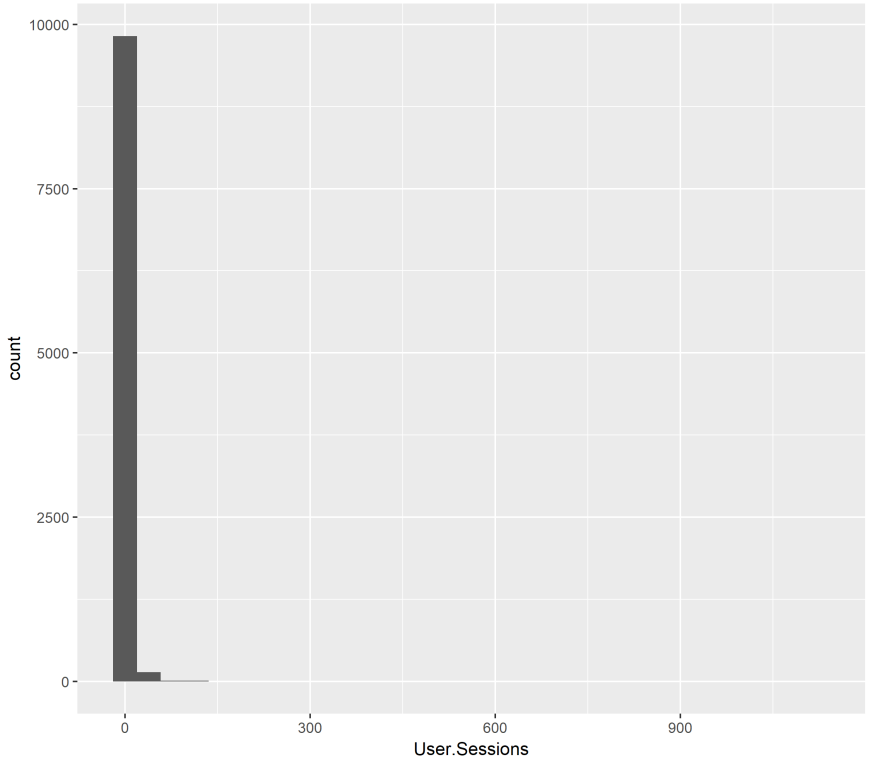
Each geoms takes a `mapping` argument within the `aes()` call. This is called the **aesthetic mapping argument.** In other words, inside the geom function is an aes function, and inside aes is a mapping argument specifying how to map the variables inside the visualization. `ggplot` then looks for that variable inside the `data` argument, and plots it accordingly.

For example, in the above expression, the **LC.Class** variable is being mapped to the x axis in the geometric shape of a bar. In this example, the y axis (count) is not a variable in the original dataset, but is the result of `geom_bar()` **binning** your data inside each LC.Class and plotting the bin counts (the number of items falling into each bin).

4.2.1 Univariate geoms

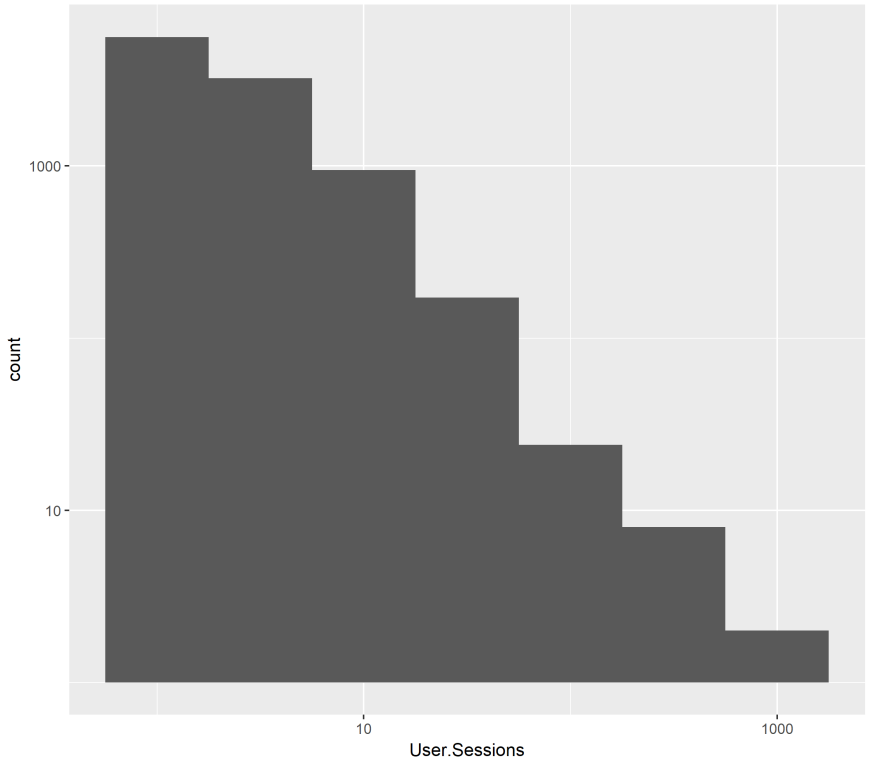
"Univariate" refers to a single variable. A histogram is a univariate plot: it shows the frequency counts of each value inside a single variable. Let's say we want to visualize a frequency distibution of User Sessions in the ebooks data. In other words, how many items have 1 session? How many have 2 sessions? And so on. Note that this data has **no items with zero user sessions.**

```
# Note that we can omit the "data =" argument in ggplot()
# and the "mapping = " argument in aes()
ggplot(ebooks) +
   geom_histogram(aes(x = User.Sessions))
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The problem here is that the overwhelming majority of books have a small amount of usage, so the plot is heavily skewed. I thought about using a different dataset for this reason, but chose not to because this is a very common problem in usage analysis. It can be addressed either by adding a `binwidth` argument to `aes()`, and changing the scales of the x and y axes by adding more arguments to ggplot:

```
# I am also changing the binwidth. Read ?geom_histogram for more information
# Map a histogram of User.Sessions to the y axis
ggplot(data = ebooks) +
  geom_histogram(aes(x = User.Sessions)
                     , binwidth = .5) +
  scale_x_log10() +
  scale_y_log10()
```



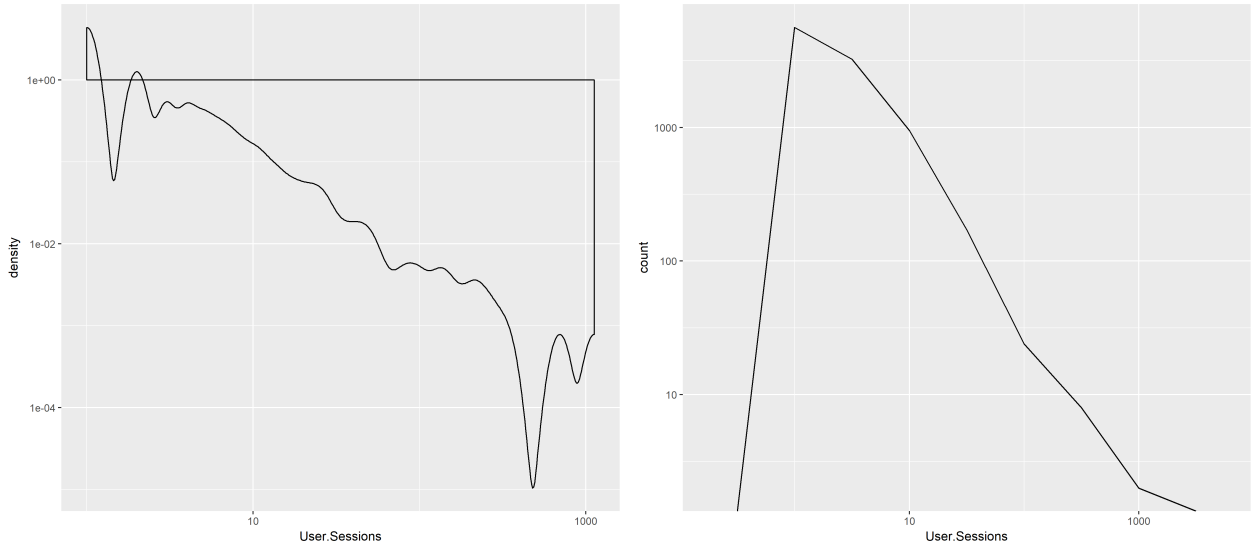
Notice the scale on both the y and x axes go from 0-10, 10-100, and 100-1000. This is called "logarithmic scale" and is based on orders of magnitude. We can therefore see that over 1,000 ebooks (on the y axis) have between 0-10 user sessions (on the x axis), and fewer than 10 ebooks (on the y axis) have over 1,000 user sessions (on the y axis). ggplot has thus given us an easy way to visualize the distribution of user sessions. I think you will find if you test this on your own print and ebook usage data, you will find something similar.

4.2.1.1 Changing the geom

This same exact data can be visualized in a couple different ways by replacing the `geom_histogram()` function with either `geom_density()` or `geom_freqpoly()`:

```
ggplot(data = ebooks) +
  geom_density(aes(x = User.Sessions)
                   , binwidth = .5) +
  scale_x_log10() +
  scale_y_log10()

ggplot(data = ebooks) +
  geom_freqpoly(aes(x = User.Sessions)
                   , binwidth = .5) +
  scale_x_log10() +
  scale_y_log10()
```

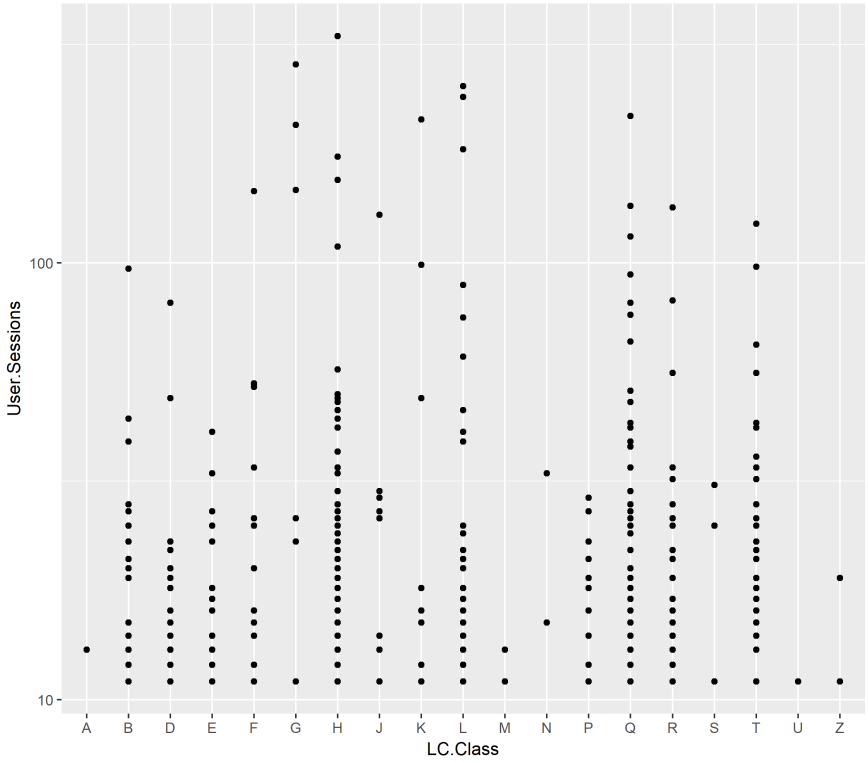


4.2.2 Bivariate geoms

Bivariate plots visualize two variables. I want to take a look at some higher usage items, but first eliminate the extreme outliers with high usage, which I will do with `filter()` from the `dplyr` package. I will then visualize user sessions by call number with a scatter plot. There is still so much skew that I retain the logarithmic scale on the y axis with `scale_y_log10()`.

```
# first create a new variable ebooksPlot with filter()
ebooksPlot <- filter(ebooks
                     , User.Sessions < 500 & User.Sessions > 10)

# map the User.Sessions variable to the y axis,
# and the LC.Class category to the x axis,
# and plot it on a logarithmic scale
ggplot(data = ebooksPlot) +
  geom_point(aes(x = LC.Class, y = User.Sessions)) +
  scale_y_log10()
```



Again, notice the scale on the y axis. We can obseve a few items of interest here: The P Class has no items with over 10 user sessions, and the G Class has very few items, but three of them have over 100 user sessions. You can then dig into these higher usage items using `filter()`:

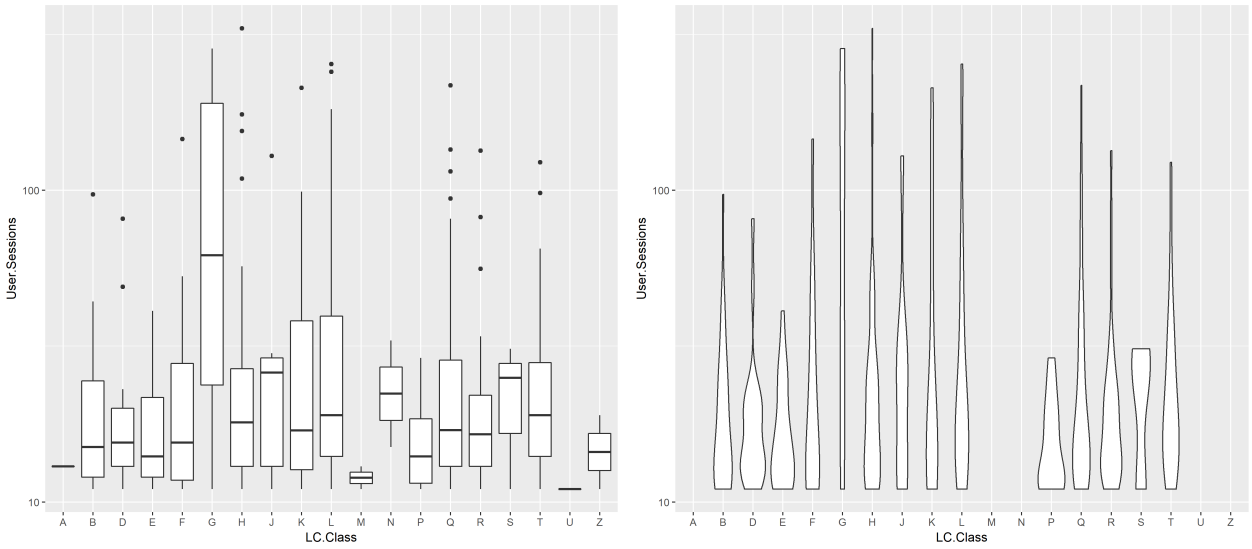
```
ebooks %>%
  filter(LC.Class == "G" & User.Sessions > 100) %>%
  select(Title)
```

```
##                                                                                 Title
## 1 Around the Tuscan Table : Food, Family, and Gender in Twentieth Century Florence
## 2                                Fundamentals of Environmental Sampling and Analysis
## 3            Life and Words : Exploring Violence and the Descent into the Ordinary
```

Just as with univariate plots, we can use different geoms to view various aspects of the data, which in turn reveal different patterns:

```
ggplot(data = ebooksPlot) +
  geom_boxplot(aes(x = LC.Class, y = User.Sessions)) +
  scale_y_log10()

ggplot(data = ebooksPlot) +
  geom_violin(aes(x = LC.Class, y = User.Sessions)) +
  scale_y_log10()
```



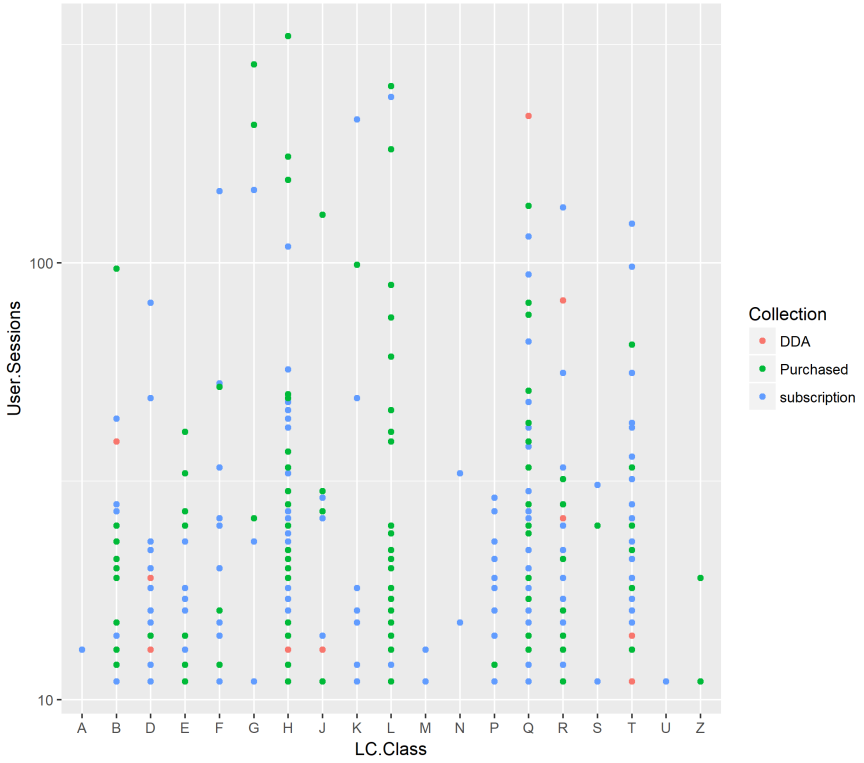
4.3 Add a third variable

You can convey even more information in your visualization by adding a third variable, in addition to the first two on the x and y scales.

4.3.1 Add a third variable with aes()

We can use arguments in `aes()` to map a visual aesthetic in the plot to a variable in the dataset. Specifically, we will map `color` to the Collection variable. In other words, associate the name of the aesthetic (`color`) to the name of the variable (`Collection`) inside `aes()`

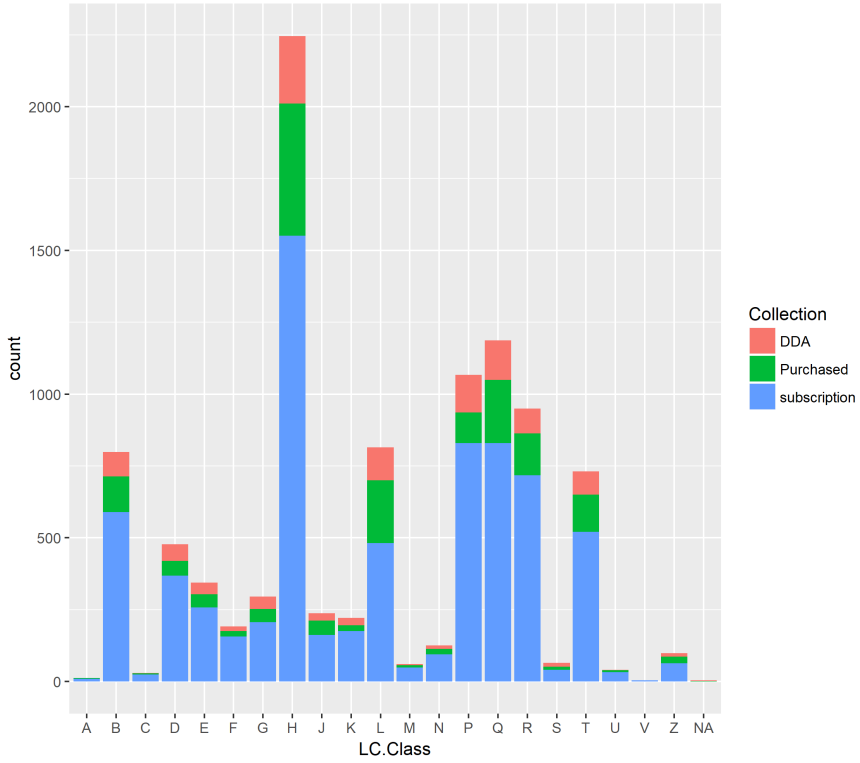
```
ggplot(data = ebooksPlot) +
  geom_point(aes(x = LC.Class
                 , y = User.Sessions
                 , color = Collection)) +
  scale_y_log10()
```



`ggplot()` automatically assigns a unique level of the aesthetic to each unique value of the variable (this is called **scaling**). Most of the items in this dataset are either librarian-purchased or part of our subscription, but there are some DDA ("demand driven acquisition") items in there.

Use `fill()` with `geom_bar()` to create a stacked bar plot:

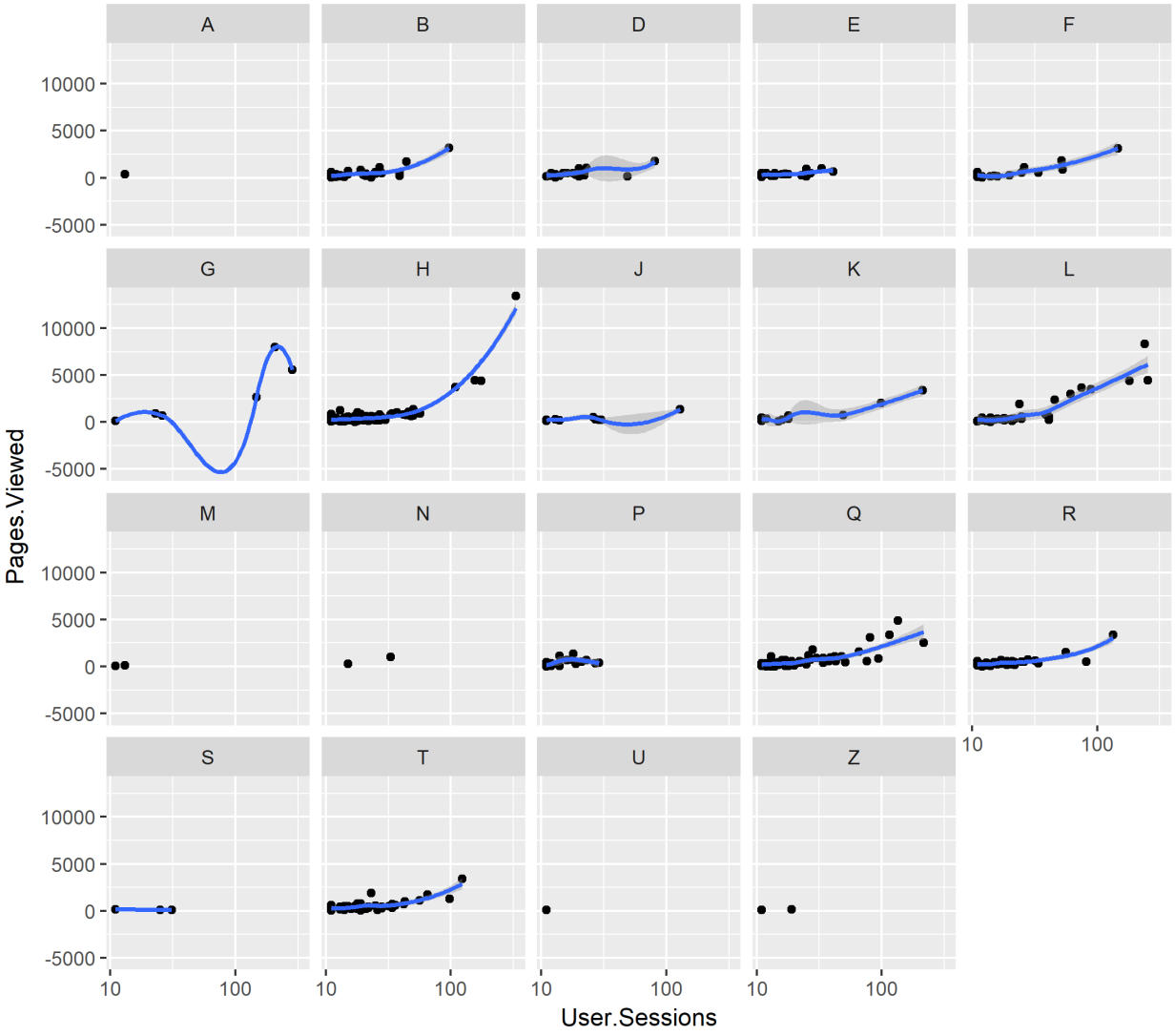
```
ggplot(data = ebooks) +
    geom_bar(aes(x = LC.Class
         , fill = Collection))
```



4.3.2 Add a third variable with facets

Another way to add a third variable is to use facet functions. These create subplots of subsets of the data. Below I use `facet_wrap()` to break up the main plot by LC.Class. Read more at `help(facet_wrap)` and `help(facet_grid)`.

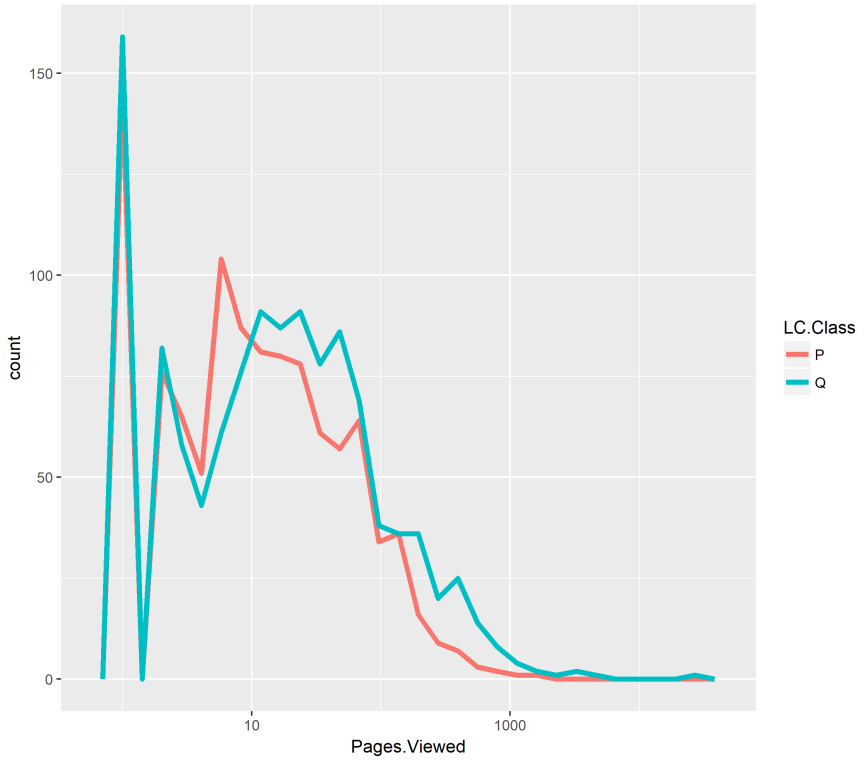
```
ggplot(data = ebooksPlot) +
  geom_point(aes(x = User.Sessions
                 , y = Pages.Viewed)) +
  facet_wrap(~ LC.Class) +
  geom_smooth(aes(x = User.Sessions
                  , y = Pages.Viewed)) +
  scale_x_log10()
```


A
B
D
E
F
G
H
J
K
L
M
N
P
Q
R
S
T
U
Z
10000
5000
0
-5000
10
100
Pages.Viewed
User.Sessions

4.4 Putting it all together with %>%

We saw at the end of Session 2 that multiple `dplyr` functions can be added to each other with the pipe symbols `%>%`. This symbol also works with `ggplot2` functions. Below, I use `filter()` to create a subset of `ebooks` where LC.Class is equal to either Q or P, then, using a frequency polgon (`freqpoly`), I map the x axis to Pages.Viewed (i.e. how many books have 1 page viewed, how many books have 2 pages viewed, and so on), and I map the color aesthetic to LC.Class, changing the size (thickness) of the line to 1.5.

```
filter(ebooks
       , LC.Class == "Q" | LC.Class == "P") %>%
  ggplot() +
  geom_freqpoly(aes(x = Pages.Viewed
                    , color = LC.Class
                    , binwidth = .5)
                , size = 1.5) +
  scale_x_log10()
```



We see on the whole that more pages per item tend to be viewed in items with call number Q.

4.5 Theme your plot

Of course we want to add titles, change axis coordinate displays, background color, tick marks, and other matters of style. There are some preexisting themes, which you can read about on the tidyverse ggtheme page. Call `help(theme)` to read about all the different components you can change manually to create your own custom theme.

4.6 Save your plot

Call `ggsave()` to save an image of your plot. As you can see if you read `help(ggsave)`, arguments include the filename, path, size, file type (`device`), dimensions, and dpi. For example:

```
myPlot <- ggplot(data = ebooks) +
  geom_histogram(aes(x = User.Sessions)
                     , binwidth = .5) +
  scale_x_log10() +
  scale_y_log10()
ggsave(filename = "myPlot.png"
      , plot = myPlot
      , path = "./doc/images/"
      , device = "png"
      , height = 4
      , width = 4
      , units = "in")
```

4.7 Help with ggplot2

- *R For Data Science* Chapter 3: http://r4ds.had.co.nz/data-visualisation.html
- Data Visualization with ggplot2 interactive course by Rick Scavetta
- The R Graphics Cookbook by Winston Change