

Lecture 4: Data manipulation and visualization in R

Modeling Social Data, Spring 2018

Columbia University

Mia Fryer

February 15, 2018

1 Overview

- Intro To R
 - Why R?
 - An Overview of the basics
- tidyverse
 - Overview
 - dplyr
 - * filter
 - * arrange
 - * select
 - * mutate
 - * group_by
 - * summarize
 - * %>% (Pipe)
 - * gather
 - * spread
- Data Visualization
 - Why Visualize
 - Methods of Visualization
 - ggplot2

2 Intro to R

2.1 Why R?

While R is by no means a perfect language, it happens to be a wonderful tool for data analysis. Once learned it facilitates the transfer of ideas from brain to computer screen with as little friction as is currently possible.

Some examples of how R can be a bit unruly are as follows

- There is not standard case convention so seeing camelCase, this.that, and snake case conventions mixed and matched is not uncommon.

- Dots (.) (mostly) don't mean anything special
- \$ gets used in funny ways
- R is very flexible in nature, and does many things for you behind the scenes which can lead to unexpected results if not careful.

2.2 An Overview of the basics

Where R shines is its ability to go from raw data to results quickly and efficiently, allowing you to "ask more questions"

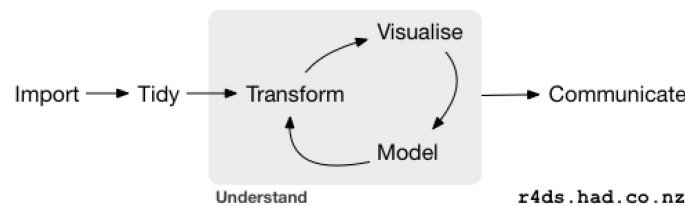


Figure 1: The data analysis cycle

In R there are basic data types which behave in different ways

- int, double: for numbers
- character: for strings
- factor: for categorical variables (similar to struct or ENUM a factor is categorical variable where strings or values are categorized)

As well as the type of individual data, the containers for this data have their own characteristics

- vector: for multiple values of the same type (array)
- list: for multiple values of different types (dictionary)
- data.frame: for tables of rectangular data of mixed types (matrix)

We'll mostly work with data frames, which themselves are lists of vectors

3 tidyverse

3.1 Overview

tidyverse is the brain child of Hadley Wickham, and while he would most definitely scold us for giving him the credit for the creation and upkeep of tidyverse, his work on this package and many others has helped propel the work of data scientists around the world.

The tidyverse itself is a collection of packages including

- dplyr for split / apply / combine type counting
- ggplot2 for making plots
- tidyr for reshaping and tidying data

- readr for reading and writing files

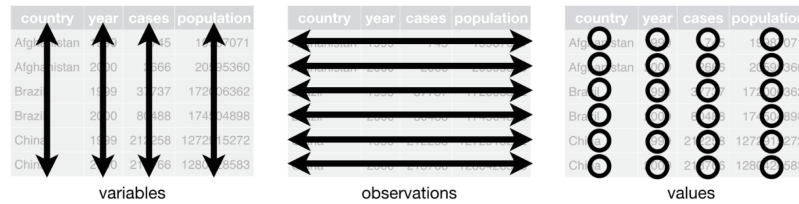


Figure 2: A representation of a "tidy table"

The core philosophy of tidyverse is that all data should be in a tidy table (As see in Figure 2) which means:

- One variable per column
- One observation per row
- One measured value per cell

3.2 dplyr

On the data manipulation side of "tidyverse" we have the library **dplyr**. The functions in **dplyr** we are going to go over today are as follows: filter, arrange, select, mutate, group_by, summarize, %>% (Pipe), gather, spread

3.2.1 filter

filter is a function which allows you to filter through a data frame with specific criteria. Utilizing the structure **filter(Data Frame, Criteria 1, Criteria 2, ...)**

An example of how we can filter in the data set we are utilizing for our homework (citibike) is as follows

```
filter(trips, start_station_name == "Broadway & E 14 St")
```

	tripduration	starttime	stoptime	start_station_id	start_station_name	start_station_latitude	start_station_longitude
1	372	2014-02-01 00:00:03	2014-02-01 00:06:15	285	Broadway & E 14 St	40.73455	-73.99074
2	439	2014-02-01 00:02:14	2014-02-01 00:09:33	285	Broadway & E 14 St	40.73455	-73.99074
3	636	2014-02-01 00:08:25	2014-02-01 00:19:01	285	Broadway & E 14 St	40.73455	-73.99074
4	914	2014-02-01 00:43:21	2014-02-01 00:58:35	285	Broadway & E 14 St	40.73455	-73.99074
5	906	2014-02-01 00:43:36	2014-02-01 00:58:42	285	Broadway & E 14 St	40.73455	-73.99074
6	468	2014-02-01 00:57:12	2014-02-01 01:05:00	285	Broadway & E 14 St	40.73455	-73.99074

Figure 3: Example: Trips Filtered for trips that start on Broadway

Whats weird about this? if you've looked at the data set for homework one there is no object named start_station_name, instead start_station_name is a column. This means that the command filter is taking care of the scope for you, meaning you don't have to index to the specific column like you would in other languages. This is very convenient because it allows you to work naturally.

3.2.2 arrange

The function arrange is a nifty function that allows you to sort columns in your data set. following the structure of **arrange(Data Frame, column to sort, ...)** adding of course any sort modifiers you might want to include

3.2.3 select

The select function allows you to choose which rows you want to display or pass on to another function (more on that later). The format of the select function is as follows **select(Data Frame, Column1, Column 2, ...)** Lets look at an example.

```
select(trips, starttime, stoptime, start_station_name, end_station_name)
```

	starttime	stoptime	start_station_name	end_station_name
1	2014-02-01 00:00:00	2014-02-01 00:06:22	Washington Square E	Stanton St & Chrystie St
2	2014-02-01 00:00:03	2014-02-01 00:06:15	Broadway & E 14 St	E 4 St & 2 Ave
3	2014-02-01 00:00:09	2014-02-01 00:10:00	Perry St & Bleecker St	Mott St & Prince St
4	2014-02-01 00:00:32	2014-02-01 00:10:15	E 11 St & Broadway	Greenwich Ave & 8 Ave
5	2014-02-01 00:00:41	2014-02-01 00:04:24	Allen St & Rivington St	E 4 St & 2 Ave
6	2014-02-01 00:00:46	2014-02-01 00:09:47	Warren St & Church St	Pike St & Monroe St

Figure 4: Example: Start Time, Stop Time, Start Station, and End Station are the columns displayed

3.2.4 mutate

The mutate function allows you to create new data columns by doing an operation on one or more data columns. The resulting calculation will be put into a new column which you can either name or have auto named.

```
mutate(trips, time_in_min = tripduration \60)
```

	tripduration	starttime	stoptime	time_in_min
1	382	2014-02-01 00:00:00	2014-02-01 00:06:22	6.366667
2	372	2014-02-01 00:00:03	2014-02-01 00:06:15	6.200000
3	591	2014-02-01 00:00:09	2014-02-01 00:10:00	9.850000
4	583	2014-02-01 00:00:32	2014-02-01 00:10:15	9.716667
5	223	2014-02-01 00:00:41	2014-02-01 00:04:24	3.716667
6	541	2014-02-01 00:00:46	2014-02-01 00:09:47	9.016667

Figure 5: Example: setting a new column time in minutes equal to the trip duration divided by 60 (since the trip duration is in seconds)

3.2.5 group_by

The group_by command allows you to keep track of groups within a data frame. This is useful especially when combined with the summarize function which will be covered shortly. An example of grouping can be seen below

```
trips.by_gender %>% group_by(trips, gender)
```

```
Source: local data frame [224,736 x 4]
Groups: gender [3]
```

	tripduration <int>	starttime <dtm>	stoptime <dtm>	gender <int>
1	382	2014-02-01 00:00:00	2014-02-01 00:06:22	1
2	372	2014-02-01 00:00:03	2014-02-01 00:06:15	2
3	591	2014-02-01 00:00:09	2014-02-01 00:10:00	2
4	583	2014-02-01 00:00:32	2014-02-01 00:10:15	1
5	223	2014-02-01 00:00:41	2014-02-01 00:04:24	1
6	541	2014-02-01 00:00:46	2014-02-01 00:09:47	1
7	354	2014-02-01 00:01:01	2014-02-01 00:06:55	1
8	916	2014-02-01 00:01:11	2014-02-01 00:16:27	1
9	277	2014-02-01 00:01:33	2014-02-01 00:06:10	1
10	439	2014-02-01 00:02:14	2014-02-01 00:09:33	2
# ... with 224,726 more rows				

Figure 6: Example: In this example, we are grouping by gender and storing the values in a new data frame called `trips_by_gender` noticed the red circle. We have three groups 1 = male, 2 = female, 0 = unknown

A word of caution when using the `group_by` function, after you're done with it you will want to un-group. This is due to the fact that when you group you're adding on a separate index which holds what "groups" are in the data set. So when you run `summarize`, it will take these groups and give you statistics for each one. However if you are done analyzing those groups, it could make for some problematic results when trying to get statistics for other matters of interest. So in short after you're done using a group for analysis **Always un-group**

3.2.6 group_by + summarize

The `summarize` command allows input queries such as mean and standard deviation and computes these queries for the data you input into it. This can be truly powerful when used in conjunction with the `group_by` function as you can then look for specific statistics for specific groups you are interested in. See example below

```
summarize(trips_by_gender, count = n(), mean_duration = mean(tripduration) \60, sd_duration =
          sd(tripduration) \60)
```

	gender	count	mean_duration	sd_duration
1	Unknown	6731	29.01385	92.76851
2	Male	176526	13.56721	83.67627
3	Female	41479	16.52268	118.57922

Figure 7: Example: Here we are looking for the count, mean, and standard deviation of trip duration by gender


3.2.7 gather

Using the `gather` command allows you too merge data sets in a way that combines columns. This might be useful in different situations where a transformation like this would help with the analysis of data. For our example below we wanted to see the sequential happenings of each trip instead of just when each trip starts and stops. This might help us calculate something like how many bikes are on the road at a given time.

Lets take the example of calculating how many bikes are on the road at a given time.

```
trips %>% gather("variable", "value", starttime, stoptime) %>% arrange(value) %>% mutate(delta = ifelse(variable == "starttime", 1, -1))
```

trip_id	starttime	stoptime
1	2014-02-01 00:00:00	2014-02-01 00:06:22
2	2014-02-01 00:04:02	2014-02-01 00:08:54
3	2014-02-01 00:06:53	2014-02-01 00:18:28
4	2014-02-01 00:09:03	2014-02-01 00:23:41



trip_id	variable	value	delta
1	starttime	2014-02-01 00:00:00	1
2	starttime	2014-02-01 00:04:02	1
1	stoptime	2014-02-01 00:06:22	-1
3	starttime	2014-02-01 00:06:53	1
2	stoptime	2014-02-01 00:08:54	-1
4	starttime	2014-02-01 00:09:03	1
3	stoptime	2014-02-01 00:18:28	-1
4	stoptime	2014-02-01 00:23:41	-1

Figure 8: Example: By "gathering" the stop and start time in one column we can see the order of sequential events, by arranging these times it allows us to calculate how many bikes are on the road via the mutate function

3.2.8 spread

The spread function operates much like the gather function, but in reverse. Taking groups of data and making them separate columns. Looking at figure 8 the spread(variable, value) command would revert the columns on the right to the columns on the left.

4 Data Visualization

4.1 Why Visualize

Data visualization serves two main purposes. First it allows you to see data and get a better overall view of what is going on. Second it allows us to communicate data to others in the most time effective way possible.

A great example of why data visualization is important can be seen in Anscombes quartet. As the name quartet implies, it is 4 data sets that share the same mean, variance, linear regression line, as well as other statistical measures as seen in the figure below.

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Property	Value
Mean of x	9
Sample variance of x	11
Mean of y	7.50
Sample variance of y	4.125
Correlation between x and y	0.816
Linear regression line	$y = 3.00 + 0.500x$
Coefficient of determination of the linear regression	0.67

Figure 9: Example: Anscombe's quartet

When you look at each of these different data sets plotted out however, the story looks much different.

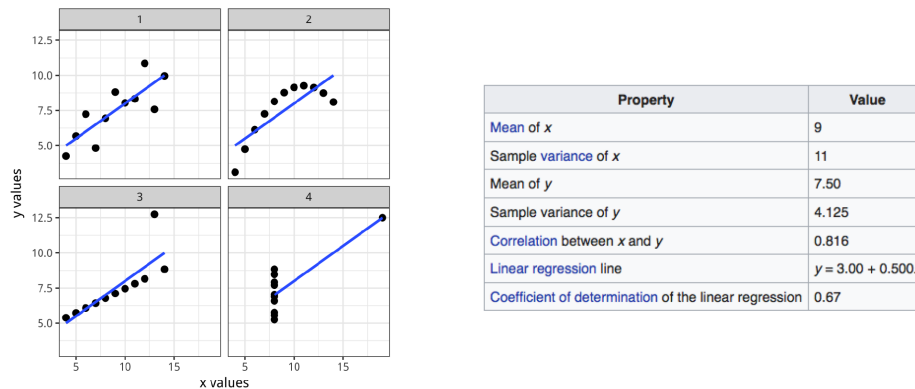


Figure 10: Example: Anscombe's quartet plotted

4.2 Methods of Visualization

As demonstrated in figure 10 plotting out data can reveal traits more quickly than reading of some statistical analysis of the data. However when it comes to generating information graphics for others to consume what is the best method of displaying said data?

Obviously the answer to that question is a solid "It depends" but instead of trying to reinvent the wheel lets take a look at some of the research that has been done on the topic starting with a paper by **Jock Mackinlay** back in 1986

In his paper entitled *Automating the Design of Graphical Presentations of Relational Information* he espoused some basic principles of a good plot

- Good plots should express the facts effectively as possible
- Tell the truth and nothing but the truth
- Use encodings that people can easily decode
- Make a clear and concise point
- Have a one sentence take-away

He also broke down how we as humans perceive data, in terms of how accurately we can decipher different representations.

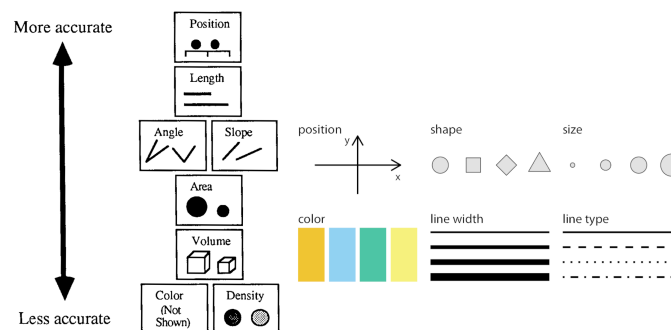


Figure 11: Example: According to Mackinlay, the arrow on the left indicates from easiest to understand on top to hardest to understand on bottom how accurate humans are at deciphering information from graphical representation

Mackinlay further delved into the human readability of plots and data as he looked at the strokes for different data types. where Quantitative: numerical values in a range (e.g., height) Ordinal: categories with natural ordering (e.g., day of week) Nominal: categories with no natural ordering (e.g., gender) this can be seen in both figure 12 and figure 13

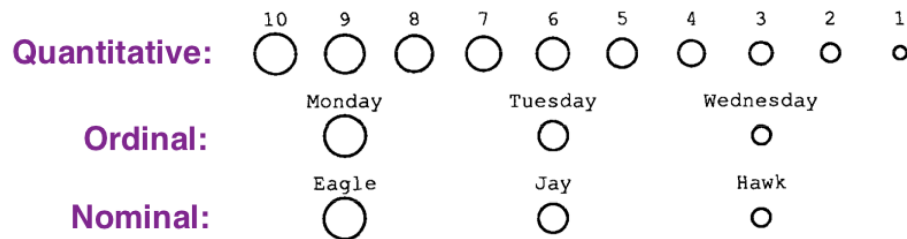


Figure 12: Analysis of the area task. The top case shoes that area is moderately effective for encoding quantitative information. The middle case shoes that is it possible to encode ordinal information as long as the step size between areas is large enough so that the values are not confused. the bottom case shows that it is possible to encode nominal information, but people may perceive an ordinal meaning.

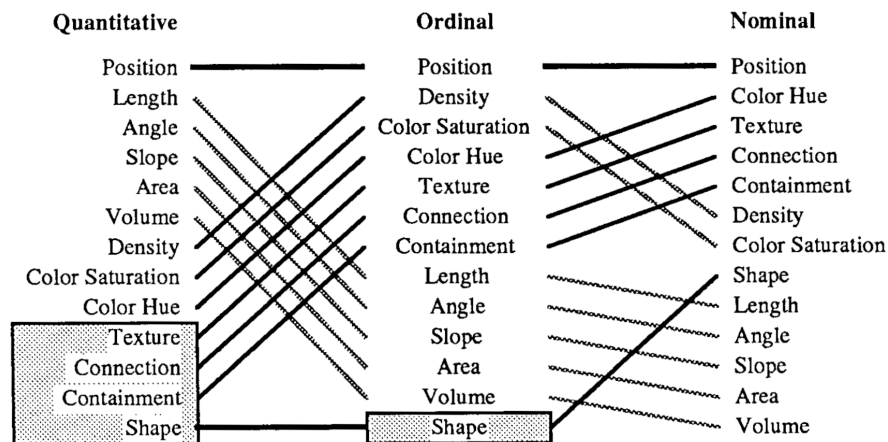


Figure 13: Ranking of perceptual tasks. The task shown in the gray boxes are not relevant to these types of data.

4.3 ggplot2

In the actual implementation of data visualization in R we are going to be using a package called ggplot2.

First off lets take a look at the grammer structure of how to write ggplot2 code.

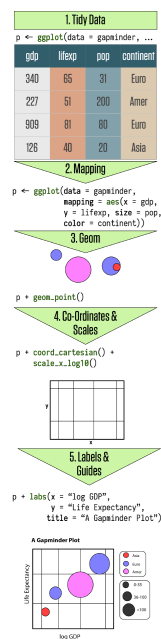

```

ggplot(data = <DATA>) +
  <GEOM_FUNCTION> (
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>

```

Figure 14: An overview of the grammatical structure of how to program a ggplot visualization

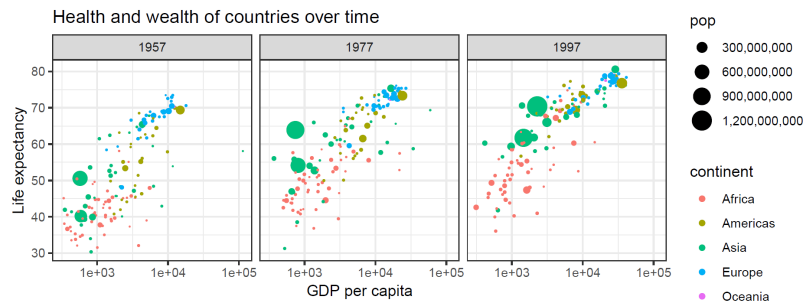
As with any type of work, its good to maintain a consistent work flow, in figure 15 below is a good work flow to follow when using ggplot



- ① Get your **data** into the right format
- ② Map variables to **aesthetics**
- ③ Choose a **geometry** for your plot
- ④ Set **co-ordinate system** and **scales**
- ⑤ Add **annotations, legends, and labels**

Figure 15: ggplot work flow

Finally let's dive into an example of how to use ggplot to plot the health and wealth of countries over time.



```
ggplot(data = gapminder,
       aes(x = gdpPercap, y = lifeExp,
           size = pop, color = continent)) +
  geom_point() + scale_x_log10() +
  scale_size_area(label = comma) +
  labs(x = 'GDP per capita', y = 'Life expectancy',
       title = 'Health and wealth of countries over time')
  facet_wrap(~ year)
```

Figure 16: Plot and code necessary to build plot of the health and wealth of different countries over time.

If you haven't been able to understand the benefits of using ggplot and in general data visualization yet then I feel a bit bad for you, but let me recap here just in case.

- Data visualization lowers the barrier to asking questions of your data
- Lets you explore more, and faster
- Allows you to easily produces publication-ready plots
- Large and active user base for support