Lecture 3: Data Visualization
Modeling Social Data, Spring 2019
Columbia University

February 15, 2019

Notes from mzf2106

1 Overview

- Intro To R
  - Why R?
  - An Overview of the basics
- tidyverse
  - Overview
  - dplyr
    - filter
    - arrange
    - select
    - mutate
    - group_by
    - summerize
    - %>% (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) dont 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.

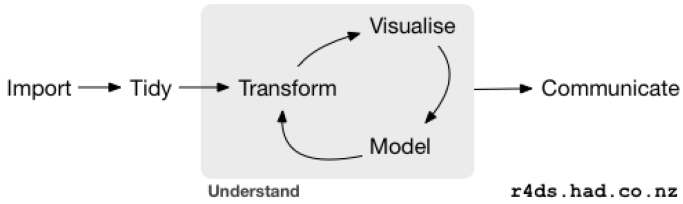


Figure 1: The data analysis cycle

2.2 An Overview of the basics

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

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

- int, double: for numbers
- character: for strings
- factor: for categorical variables (similart 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 diferent 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 tiderverse, 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

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

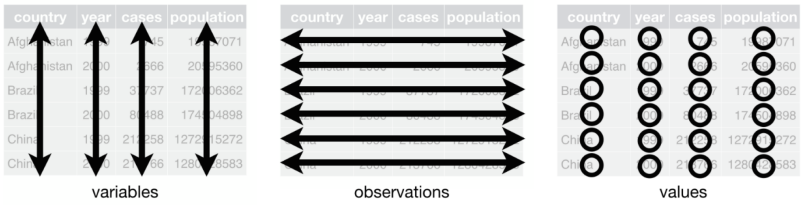


Figure 2: A representation of a "tidy table"

3.2 dplyr

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

A 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, intead 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)

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)

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

   tripduration           starttime            stoptime gender
          <int>              <dttm>              <dttm>  <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

	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))

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.

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

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

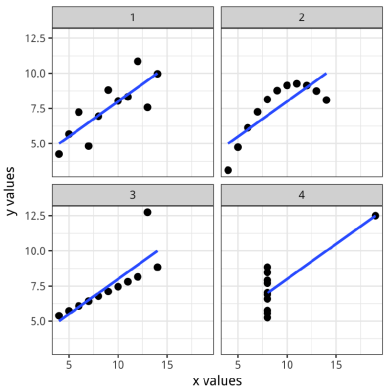
Figure 9: Example: Anscombe's quartet

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

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



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 10: Example: Anscombe's quartet plotted

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 eectively 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.

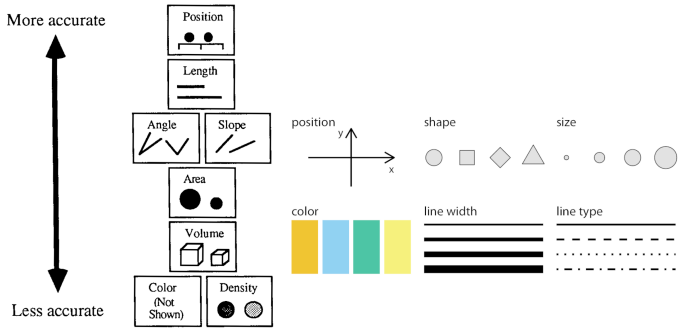


Figure 11: Example: According toe Mackinlay, the arrow on the left indicates from easiest to understand on top to hardest to understand on bottom how accurate humans are at deciphering information from graphical representation

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

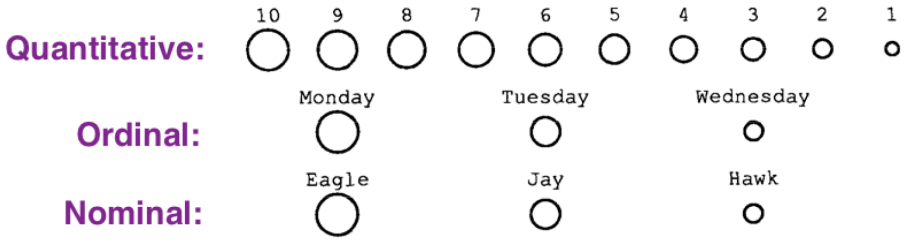


Figure 12: Analysis of the area task. The top case shoes that area is moderately effective for encoding quantitative information. The middle case shoes that is it possible to encode ordinal information as long as the step size between areas is large enough so that the values are not confused. the bottom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

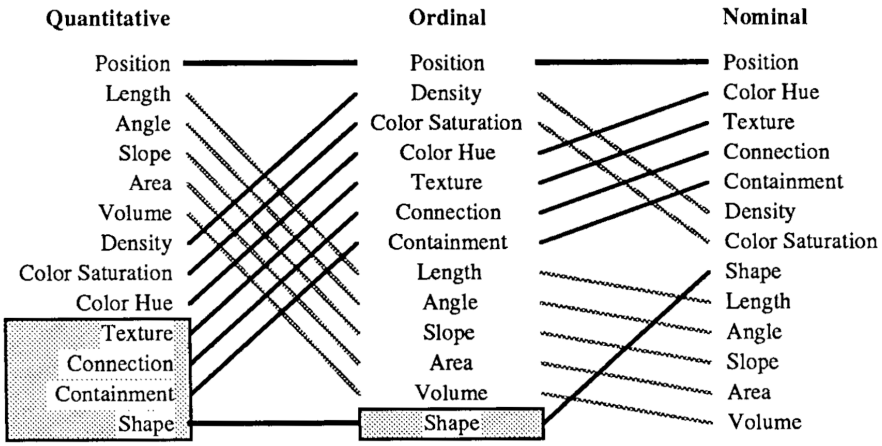


Figure 13: Ranking of perceptual tasks. The task shown in the gray boxes are not relevant to these types of data.

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.

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

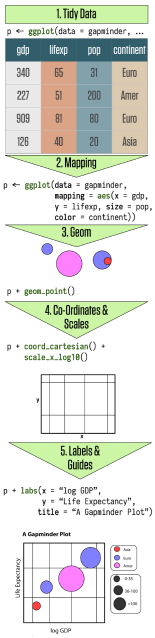
Finally lest dive into an example of how to use ggplot to plot the health and wealth of countries over time.

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

```
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



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

Notes from oum2000

1 Introduction

This lecture, we reviewed basics and data operations in R, discussed why data visualization is important and how can we benefit from it, and lastly learned how to utilize the ggplot2, plotting library in R.

2 Review of Basics and Data Operations

In some cases, we may want to convert our data to a new format, and this is often denoted as 'tidying data' . The R functions we will have to know in particular for this purpose are 'gather()' and 'spread()'.

2.1 gather() and spread()

- Gather takes wide data and makes it long, by converting column names into actual values in the table. It is not guaranteed to be in order.

- We have gone over the CitiBike example that is asked on the homework, where we convert start and stop time columns under a new variable column, and the respective times as Date objects under a value column.
- We can then sort according to the dates and end up with a periodic time line. Hence, as start times and stop times are processed, we can keep a running count on active bikes and update accordingly to get the maximum number of active bikes on a given time.
- With 'gather()' and piped operations, we can use the 'ifelse' block of base R which acts as a vectorized form of the conditional blocks seen in other languages.
- One way is to use this block inside mutate() function to assign certain values to rows of your data.
- With 'spread()' on the other hand, we go from long to wide. This time, we take a row value and convert it to a column.

2.2 Question: Are gather() and spread() inverses of each other?

Although they are pretty close to being called 'inverses', formally **NO**, they are not. This is because

1. There is no guarantee in order, the data frame doesn't necessarily match going back and forth between these two operations.
2. There is no guarantee that all columns will contain values, there may be missing values.

2.3 Basics of R and RStudio

- In RStudio, we were introduced to the terminal, the environment which holds all your data and variables, the help section, and the file system. In overall, RStudio is a great development environment with many benefits.
- We saw how to load up a library with the simple library(libraryname) conventionally placed in the beginning of the file.
- We saw that we can use concatenation to easily create vectors in R. To create a vector over a range or interval, the seq() and c() functions come in handy.
- Vectors can have named columns. Hence, a value at an index may be accessed through vector[columnname]
- We can look at the structure of any object by calling str() function upon it.
- If you cast a vector to a factor with the as.factor() function, the str() function will now give us the number of unique levels in the data. The plotting tools will know to treat these as categorical variables.
- We will not be using a lot of R 'lists'. Instead, data frames that hold tabular data are often much useful.
- **Quick R Tip**: If you are opening up a file and you are not finding something you are trying to load, it is probably because the working directory is not set to the source file location. You can do this from Session, Set Working Directory, To Source File Location.

2.4 Data Operations

- The summary() function called upon a data frame will give minimum, maximum, median, mean, 1st Quartile, and 3rd Quartile values for numerical columns, and in the case of a factor column it gives a summary of the categorical levels. The adeptness of the summary() function comes in very handy while exploring and analyzing data. Below is only what is discussed in this class. Definitely go over this file (...) to learn about more nifty operations and functions in R.

- Old way is logical indexing to select rows through dollar sign column name and the logical operation inside the brackets, whereas the new way is to use the filter(logic) function. The new way is definitely more readable. The grepl() function returns a boolean value depending on whether the pattern is contained within the value or not. Used within filter(), it will act in vectorized form.
- In the new way, use arrange() to reorder rows, select() to select rows, mutate() to modify columns, and summarize() to summarize columns and extract information such as mean and standard deviation.
- To create a grouped data frame, use groupby() function which simulates the split/apply/combine procedure discussed in earlier lectures. Within this function, there are a bunch of helper methods that can be used like n() which gives you the number of rows, rownumber() which gives you 1,2,3,4,5,... (incremental indexing) for all the rows.
- Ungrouping shouldn't be forgotten when you want to revert back to the original data frame format. Not remembering this step might cause us to get some unintended results and computations in our data analysis.
- Another reason to ungroup might be based on performance issues. When the data frame is grouped, any vectorized operation is going through each group and within each group, it is iterating which proves grouping to be a costly operation.
- A filter() operation was demonstrated on a grouped data frame, where the operation is distributed to each group as result of the split/apply/combine process, and hence it takes a longer computational time than expected. The expected value in this case was the maximum value of the whole data frame, rather than the maximum value within each group as outputted. This often leads to confusion.
- The pipe operator, that is showcased with a data frame being chained into a series of vectorized operations, is a nice convention to use in R. It enhances readability, and decreases the complexity of the code.

3 Introduction to Data Visualization

3.1 Question: Why do you want to visualize data?

- To make full meaning and utility of data.
- Convey any results in an easily understandable fashion to the readers.
- The idea is to lower the cost of going from a idea on your mind to an actual, interpretive plot.

3.2 Real Life Applications and Usage

- Professor Jake mentioned that often he has spent multiple hours on deciding figures and the types of plots he wanted to display. On projects he has worked, figures had often multiple revisions by many different people. All this is to say that, good data visualization takes a lot of practice and conveying information to readers through plots is an important tool.
- We have discussed about Anscombe's Quartet (1973) to understand why data visualization is important. The difference between the four data sets are really hard to tell just by looking at a data table. The spread, range, and the correlation between variables are especially hard to understand. When plotted, on the other hand, the difference is immediately a lot easier to interpret.
- Now that we know about the existence of a powerful plotting library in R, ggplot2, we noted that we have so many options to choose from (histogram, density plot, cumulative density, boxplots, etc.) with regard to our display of the data. We even have many options about styling (shape, size, color, and line width properties of data). There are nice rules of thumb about how to do these things also things we can learn from practice.

- Good plots should accurately and effectively express the facts. Professor Jake has mentioned that he decides whether a plot is useful or not by attempting to construct a one sentence take-away or conclusion just by looking at the plot. Looking back to the slides from the first lecture, every figure and plot provided had a one sentence takeaway written beneath it.
- Cleveland and McGill (1984) experiments reveal that position and length are often more easily understood than color and density differences in plots. Heer and Bostock (2010) has reproduced a similar ranking based of on data collected from Mechanical Turk. This is important, because it shows that visualization research can be done online instead of just depending of physical lab participants.
- In a seperate research by Jock Mackinlay, it has been shown that the ranking also depends on the type of data at hand. There are three possibilities:
    1. Quantitative: numerical values in a range
    2. Ordinal: categories with natural ordering
    3. Nominal: categories with no natural ordering
- For example, using area to encode quantitative information makes more sense than with ordinal and nominal data.
- We have seen that with annual median income (quantitative data), a plot on the map of the U.S. with different colors (darker blue indicates a higher income) was preferred. On the other hand, population growth (nominal data) was chosen to be represented reversed and sorted histogram where color now indicates the region of the state (West, South, Midwest, or Northeast).

4 Data Visualization in R with ggplot2

4.1 Grammar of Graphics

- Grammar of graphics is the language to describe the components of a plot or a visual. This grammar usually follows a general guideline template with 5 steps:
    1. Get your data into the right format by tidying.
    2. Map variables to aesthetics (color, shape, size)
    3. Choose a geometry for your plot. (different plot types)
    4. Set co-ordinate system and scales (linear or log)
    5. Add annotations, legends, and labels. (x-axis, y-axis, title)
- Then you get a plot at the end of the day. The aes() helper function takes care of the mappings for your plot inside the ggplot() function call.

4.2 Benefits of ggplot2

The benefits of ggplot2 comes directly with its conciseness and doing massive amounts of work in less lines. Some other benefits mentioned are:

- Lowers the barrier to asking questions of your data, lets you interpret easily using plots
- Lets you explore and analyze more, and faster
- Easily produces beautiful, publication-ready plots
- Large and activate user base for support

4.3 Diving into ggplot2

- Usage is simple: ggplot(data, aes=(x = data, y = data)) where "aes" stands for aesthetic mapping.
- **Keep in mind**: The plus (+) sort of adds on stuff to the plot. It is not equivalent to pipe operator.
- geomhistogram() is the function to create a histogram. You should always set the bins of the histogram yourself through geomhistorgram(bins = number of bins)
- geompoint() gives a scatter plot. ggplot() takes care of spacing for you, which is very useful.
- The piping operator is something we better get used to it because of it convenience. We can summarize the data and pipe the result immediately into ggplot.
- geomline() creates line plots. Always include xlab(title) and ylab(title) to name your axes and enhance readibility.
- geomline() + scaleylog10() gives a logged y-axis, which may make your data easier to understand from a plot.
- To see an overall trend, we can use geomsmooth(method = "lm") to fit a linear model to the data and display it on top. We can combine geompoint() which will give regular points for actual data points, with geomsmooth(method = "lm") with the + operator which gives the best fit line for the model.
- We can filter out unwanted data points by
    1. Simply excluding them from the visualization with coordcartesian(xlim = vector, ylim = vector). This will leave the best fit line and hence the representative model unchanged; only the display is changed.
    2. Use xlim(vector)or scalexcontinuous(lim = vector) to first filter the data, then change the fit accordingly, and finally display the new model.
    3. Extract the data yourself before plotting with data frame piped to filter(logic) function. This method may be used to prevent any kind of errors and confusions going forward.
- Pipe data frame to leftjoin(¡dataframe¿) function call to extend your data with more descriptive columns.
- Plotting a histogram of number of ratings by movie with the MovieLens data that is used in Homework 1, we observed that this data has a "long tailed" distribution. This means that whereas the bulk of the available movies draw little attention, a few movies get a lot of attention.
- Use geomdensity(fill = color) to have a smoothed version of the histogram. You can also include the mean in your plot as a dashed line with geomvline(aes(xintercept = mean(numratings)), linetype = "dashed")
- Use the cumsum(¡data¿) function combined with ggplot() to get a cumulative distribution (CDF).

Notes from pmb2164

1 Introduction

Lecture 4 was covered mostly hands on material paired with a little bit of lectures. He will go into detail about R, and certain programs within R. You might ask why even use R? According to master Hofman "R isn't the best programming language out there, but it happens to be great for data analysis... and when you first start to learn it, the first couple weeks will be you banging your head against the wall, only then you'll be able to show your skills." R provides the user an extremely fast exploratory data analysis, that easily generates high-quality data visualizations, and also fits pretty much any statically model you can think off, as you can see in figure one. The first half of class will talk about how data is manipulated in R, and then the second half we will discuss data visualisation.

2 First Half of Class - Data Manipulations

2.1 Lecture Slides - Lecture 3 Slides

During this subsection we will talk about the basic components of R. in R, there are basic types of variables integer, double (for numbers), character(for strings), factor (for categorical variables). Can also do containers; vector (for multiple values of the same type i.e. array), list (for multiple values of different types i.e. dictionary), and, what we will mostly use, data.frame (for tables of rectangular data of mixed types i.e. a matrix).

Installing tidyverse is a must!! Every part of this class will use the package, and all of its components, so download it. Tidyverse is a unified interface that is a collection of packages that works together to make data analysis easier such as dply (for split/apply/ and combine type counting), ggplot (for those bomb graphs we will make, like in Figure 1), tidyr for reshaping data, readr for reading and writing files, and many more. However the core philosophy is that your data should be in a "tidy" table with : one variable per column, one observation per row, one measured value per cell, as seen in Figure 2. One imporant function function in R is dplyr. For dplyr, it is mostly for a grammar of data manipulation. dplyr implements the split/apply/combine framework discussed in the last lecture (see slide for more detail). The filter function, can separate you data into nice column space. Notice that in the filter function the column names are in scope. We can also use the arrange function to then arrange your data in a certain order, you can also yous the assignment operator ¡- in R. The Select function is for certain columns of your data that you want, it can also rename your data if so chooses. The mutate function, literally mutates your data such that you can change the data to something you want (EX. change trip time in hours to trip time to minutes). The summarize function takes the data and makes it a one row summary of your input into your summarize function (such as mean and std). The Group-by function takes the data a groups it to the parameters you specify (trips-by-gender ¡- group-by(trips, gender), note that it is recommended to always ungroup a the end. Note that the group-by and summarize functions can be used together, such that you can summarize the grouped data by more parameters giving a more than one row product. Tee pipe operator takes your data, trips and is the more fluent way to do the summarize and group-by function.

For untidy data, data is...well isn't pretty and needs to be restructure we follow these commands. The function gather takes your data from "wide to long", it takes the multiple columns of data and combines them such that they're in one column, which will help with when you want to count (hint hint this is on the homework). You can also use the gather function and then mutate functions to give us additional data to how we want it (see slide 22/25). Notice that you can also undo the data set, the spread function will take your from long to wide. They are not inverses ("pretty darn close to being the inverse, but it is not in industry" - Hoffman), as the order is not guaranteed as it will failed if we do not have a complete data set, if we mutate our data (slide 24/25). All of this is in the Hadley book, there is also a tidyverse stlye guide website that has a walk through for functions and is recommended to look at.

2.2 In R Studio - Lecture 3 - Intro to R

Notice the R is nice enough to use your mouse in. There is a four panel view: top left is the Script, the bottom left is the console and your terminal, bottom right is the plot view/packages installer, and top right is the environments

for your data. In class we talked about vectors, factors, how to make lists, and most importantly data frames. We also went into have we used the functions mentioned in the lecture slides above and what they did for an example data frame. For exact examples and more thorough investigation please visit Professors Hofman's Github. DISCLAIMER: I am new to R, and it is highly highly recommended you look at lecture 3's Intro to R.

3 Second Half of Class

3.1 Lecture 4 Slides: Visualizing Data

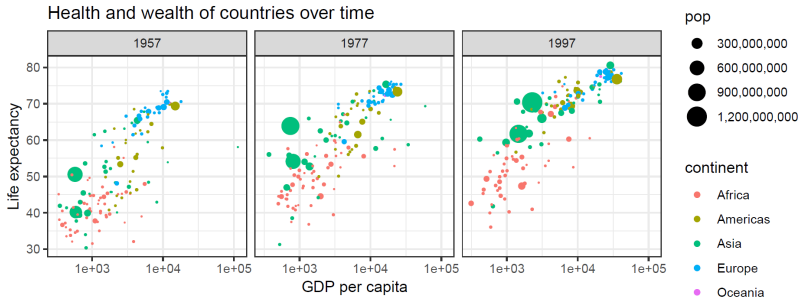
Data Visualization is a key aspect in the modern world, as it shows findings to the public. For example in Figure 3, Ansombe's quartet is a good representation of why choosing visualization is extremely important as summary data can be misleading the data has all of the same mean, sample variance, regression line, etc.

One important question that should be answered when presenting data visualizations is which type of plot is best. How do we know which type of graph is best for specific data as there are so many options(as seen in Figure 1), there are even more options how to specialise and spice up the data. Good plots are the goal, as they express the facts effectively as possible. Plots should be able to use encoding that people can easily decode, its has a clear and concise point. (see Mackinlay 1996 slide 9). Position of the plot, shape, size. color, line width, and even line type are very crucial when choosing the correct data visualization. The wrong visual will lead to improper interpretation, tread carefully reader. "Automating the Design of Graphical Presentations of Relational Information" b Jock University in 1986 was one of the first scientific paper that showed which visualizations are better than others. It showed that position in very influential, while color is not so much. It also went on to show that different strokes for different data types were and which were better. For example, Shape is not so influential for quantitative and ordinal data sets, but Shape is much more influential for nominal data sets. If the reader would like to learn more, a good book to read is "The Grammar of Graphics" by Springer or "ggplot2" by Hadley Wickham.

3.2 R Studio: Working with ggplot

During the last part of lecture, Professor Hofman quickly went over how to setup ggplot, and different types of plots and examples of such. First make sure you are running the Tidyverse, lubridate and scales libraries to start. Note that the ggplot histogram doesn't require a y, it associates the y as the count of the x function. Please, when you're reading this, open Professor Hofman's Github hub for visuals. We talked about how make bar charts, a normal histogram, a scatter plot histogram, line plots, and many more. While going through the examples keep in mind what is trying to be shown, and how each graph has its perks and drawbacks. We also go into to how to show question 3 on the homework, hint hint hint. And also how to label the charts, how to add color on the charts, and how to add error bars and confidence intervals. Please, Please go through the Github page, please it will be very beneficial

El Fin




```
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.

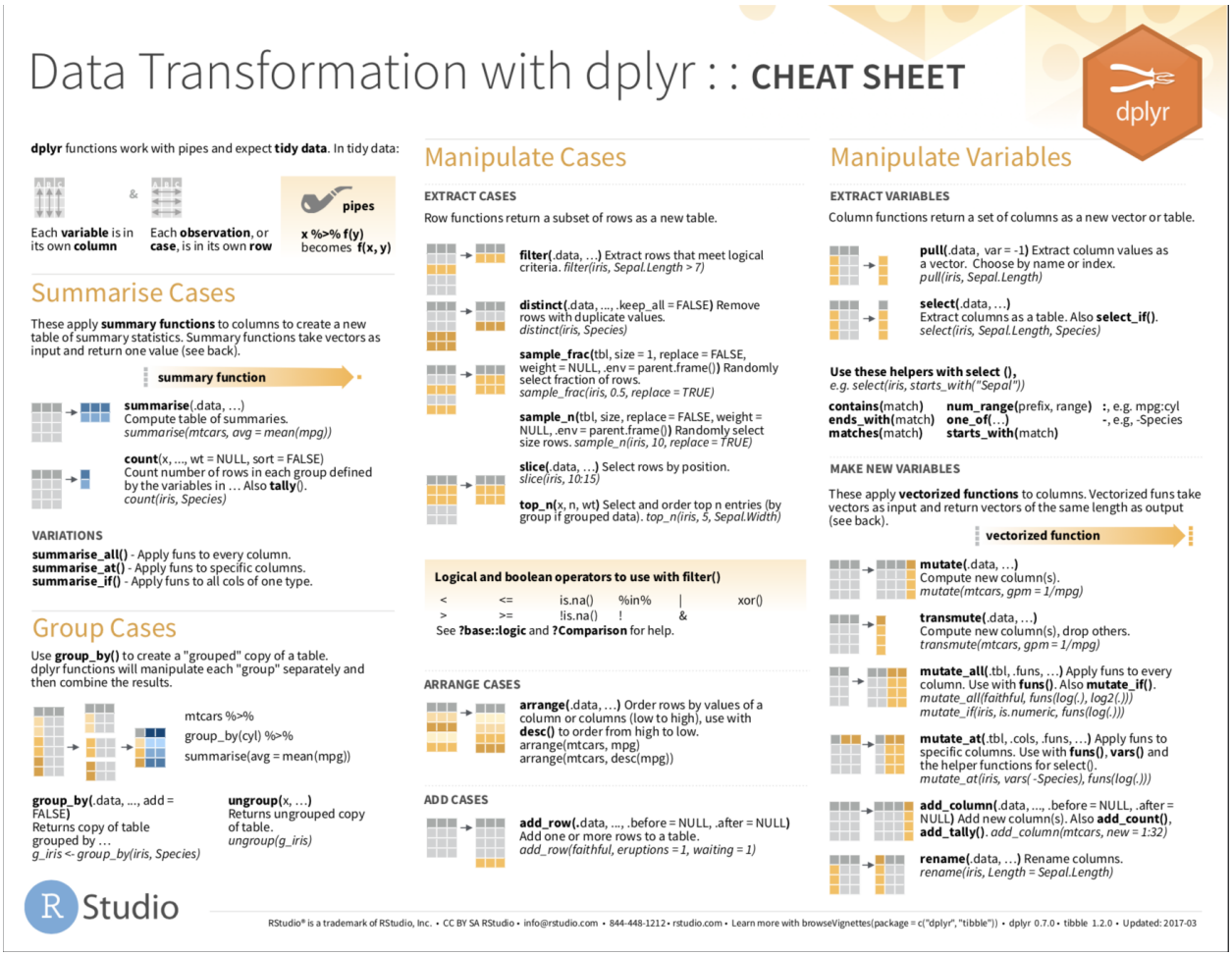


Figure 17: Official RStudio Cheat Sheet for Data Transformation with dplyr, https://www.rstudio.com/resources/cheatsheets/dplyr

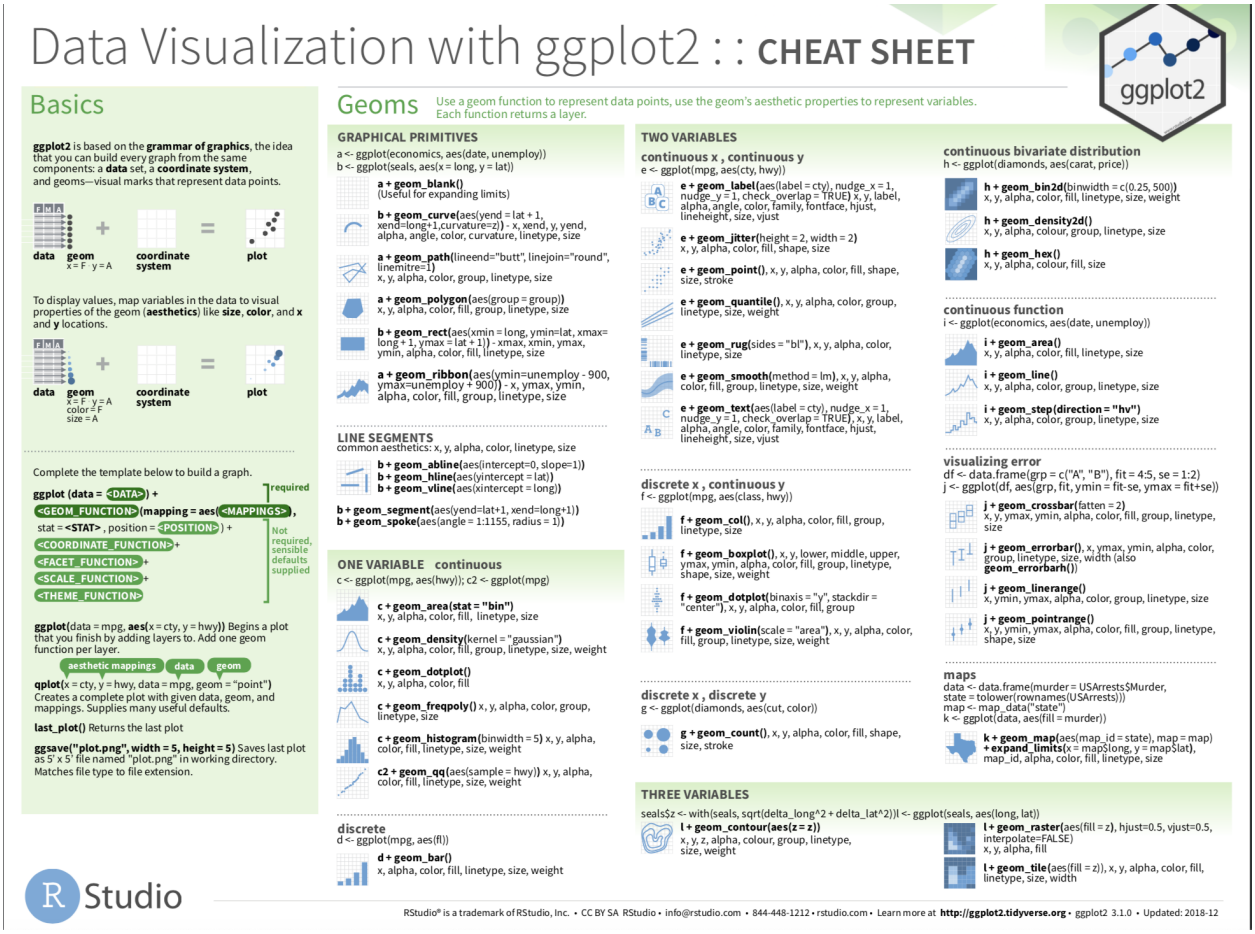


Figure 18: Official RStudio Cheat Sheet for Data Visualization with ggplot2, "https://www.rstudio.com/resources/cheatsheets/ggplot2"

Figure 19: Shows varity of statisical models in R (Lecture 4 slides)

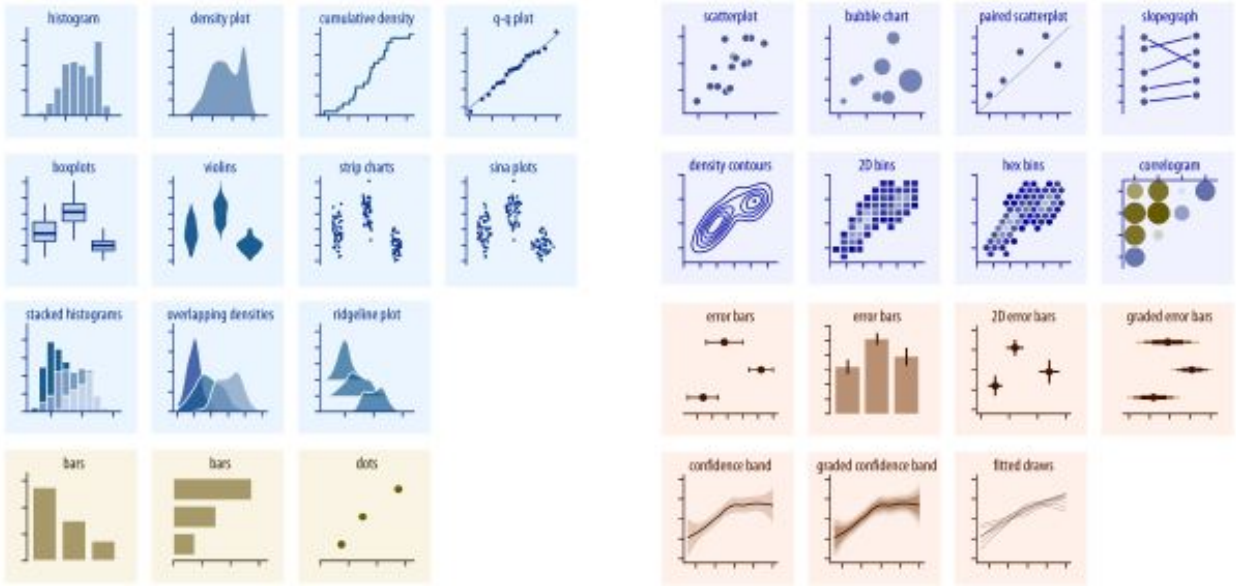


Figure 20: Tidy Data (Lecture 3 slides)

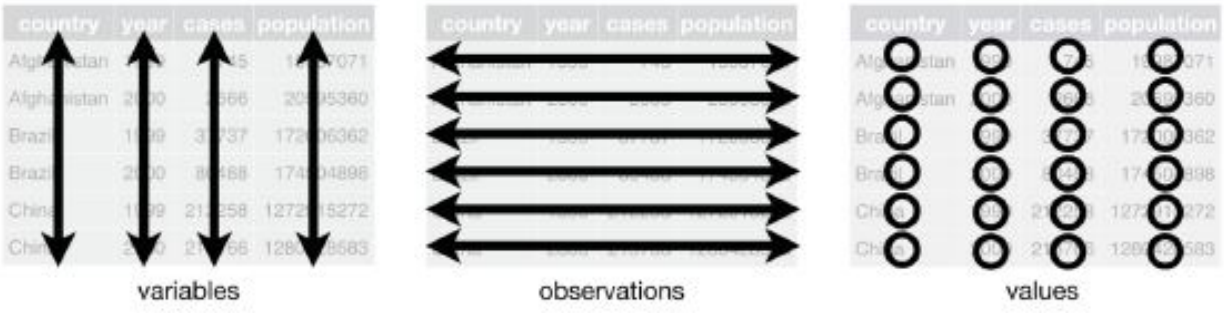
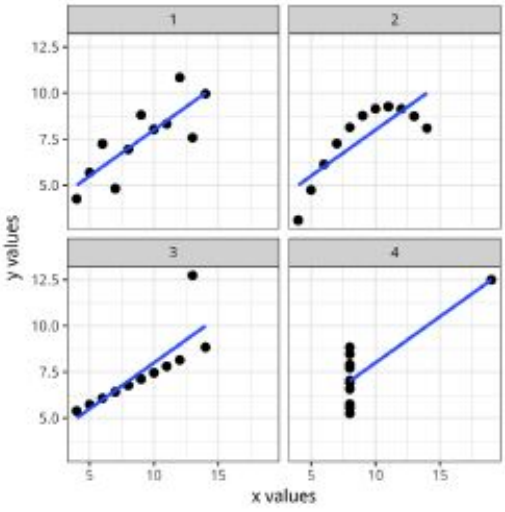


Figure 21: Anscombe's (Lecture 4 slides)



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