THE OHIO STATE UNIVERSITY
FISHER COLLEGE OF BUSINESS

BUSMGT 7331: Descriptive Analytics and Visualization
Week 2

More Data Wrangling, Covariation, and Customizing Visualization

Hyunwoo Park
Fisher College of Business
The Ohio State University

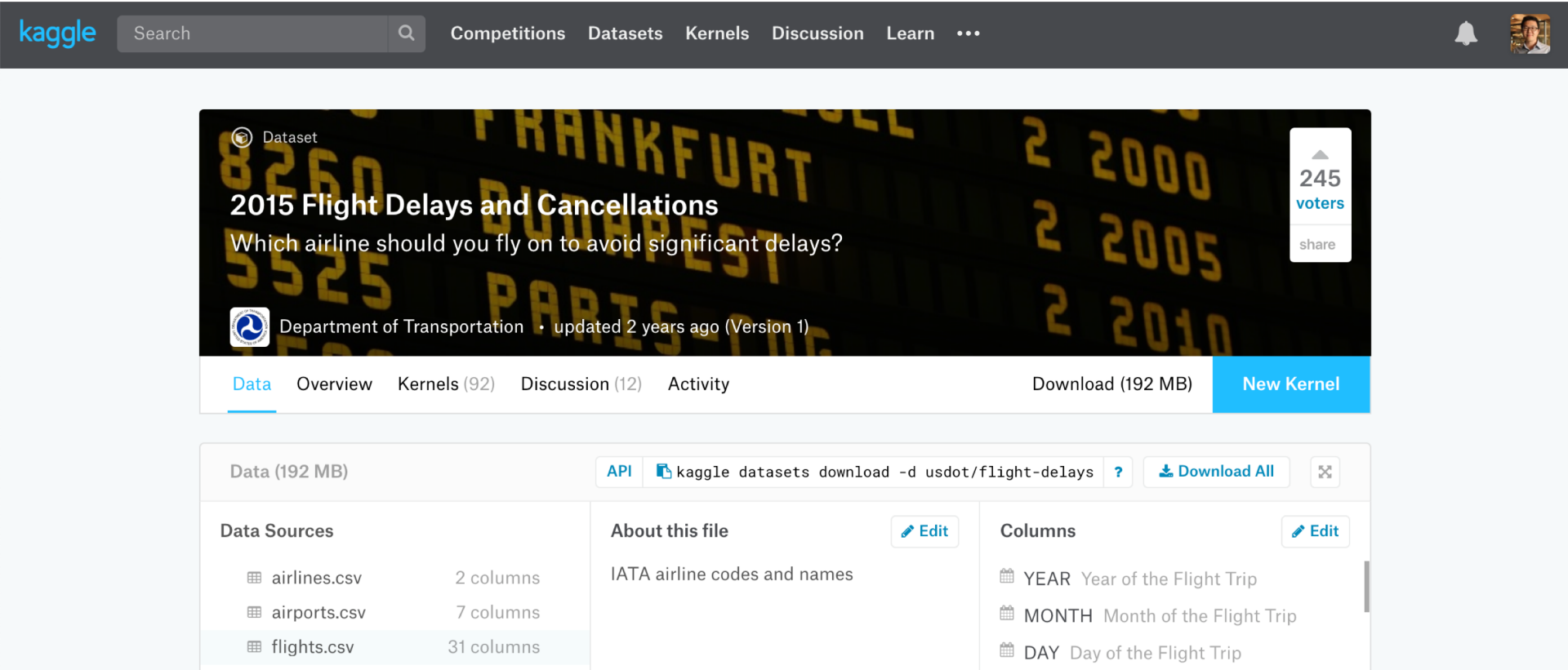
Merge, String, Factor, Date

Reading

- R4DS Chapter 10. Relational Data with dplyr
- R4DS Chapter 11. Strings with stringr
- R4DS Chapter 12. Factors with forcats
- R4DS Chapter 13. Dates and Times with lubridate

Dataset for this week

- https://www.kaggle.com/usdot/flight-delays



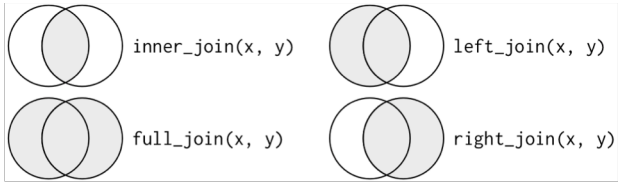
Data import

```
1  airlines <- read_csv("data/airlines.csv")
2  glimpse(airlines)
3  airports <- read_csv("data/airports.csv")
4  glimpse(airports)
5  flights <- read_csv("data/flights.csv.zip")
6  glimpse(flights)
```

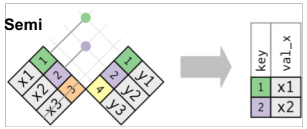
```
> glimpse(airlines)
Observations: 14
Variables: 2
$ IATA_CODE <chr> "UA", "AA", "US", "F9", "B6", "OO", "AS", "NK", "WN", "DL", "EV", "HA", "MQ"...
$ AIRLINE   <chr> "United Air Lines Inc.", "American Airlines Inc.", "US Airways Inc.", "Front...
> glimpse(airports)
Observations: 322
Variables: 7
$ IATA_CODE <chr> "ABE", "ABI", "ABQ", "ABR", "ABY", "ACK", "ACT", "ACV", "ACY", "ADK", "ADQ",...
$ AIRPORT   <chr> "Lehigh Valley International Airport", "Abilene Regional Airport", "Albuquer...
$ CITY      <chr> "Allentown", "Abilene", "Albuquerque", "Aberdeen", "Albany", "Nantucket", "W...
$ STATE     <chr> "PA", "TX", "NM", "SD", "GA", "MA", "TX", "CA", "NJ", "AK", "AK", "LA", "GA"...
$ COUNTRY   <chr> "USA", "USA", "USA", "USA", "USA", "USA", "USA", "USA", "USA", "USA", "USA",...
$ LATITUDE  <dbl> 40.65236, 32.41132, 35.04022, 45.44906, 31.53552, 41.25305, 31.61129, 40.978...
$ LONGITUDE <dbl> -75.44040, -99.68190, -106.60919, -98.42183, -84.19447, -70.06018, -97.23052...
> glimpse(flights)
Observations: 5,819,079
Variables: 31
$ YEAR                <int> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, ...
$ MONTH               <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ DAY                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ DAY_OF_WEEK         <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, ...
$ AIRLINE             <chr> "AS", "AA", "US", "AA", "AS", "DL", "NK", "US", "AA", "DL", "DL", ...
$ FLIGHT_NUMBER       <int> 98, 2336, 840, 258, 135, 806, 612, 2013, 1112, 1173, 2336, 1674, 1...
$ TAIL_NUMBER         <chr> "N407AS", "N3KUAA", "N171US", "N3HYAA", "N527AS", "N3730B", "N635N...
$ ORIGIN_AIRPORT      <chr> "ANC", "LAX", "SFO", "LAX", "SEA", "SFO", "LAS", "LAX", "SFO", "LA...
$ DESTINATION_AIRPORT <chr> "SEA", "PBI", "CLT", "MIA", "ANC", "MSP", "MSP", "CLT", "DFW", "AT...
$ SCHEDULED_DEPARTURE <time> 00:05:00, 00:10:00, 00:20:00, 00:20:00, 00:25:00, 00:25:00, 00:25...
$ DEPARTURE_TIME      <time> 23:54:00, 00:02:00, 00:18:00, 00:15:00, 00:24:00, 00:20:00, 00:19...
$ DEPARTURE_DELAY     <int> -11, -8, -2, -5, -1, -5, -6, 14, -11, 3, -6, -8, 0, -6, -1, -4, -1...
$ TAXI_OUT            <int> 21, 12, 16, 15, 11, 18, 11, 13, 17, 12, 12, 21, 18, 18, 28, 17, 25...
$ WHEELS_OFF          <time> 00:15:00, 00:14:00, 00:34:00, 00:30:00, 00:35:00, 00:38:00, 00:30...
```

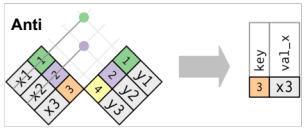
Join two tables

- "join" means "merge" in tidyverse. The join command family merges two tibbles.
- Suppose we are joining two tables, x and y.
- Types of join
  - *Mutating join* combines the variables from two tables.
    - [1] inner_join(x, y)
    - Outer joins
      - [2] left_join(x, y)
      - [3] right_join(x, y)
      - [4] full_join(x, y)



  - *Filtering join* only affects the observations based on the match.
    - [1] semi_join(x, y)
    - [2] anti_join(x, y)





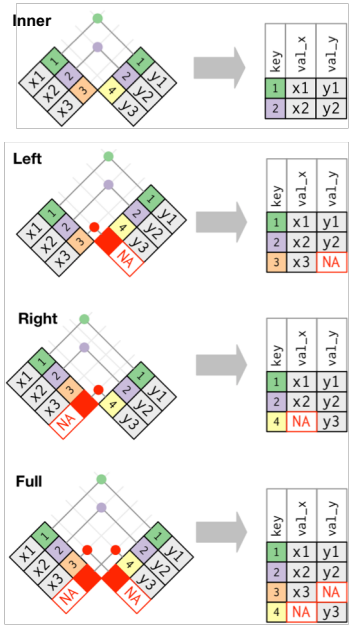


Figure from R4DS, pp. 181-182 (https://r4ds.had.co.nz/relational-data.html)

Mutating joins and SQL syntax

dplyr join syntax	SQL syntax
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING (z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`

- `by` argument
  - If you don't provide any `by` argument, dplyr will use the combination of all common variables (i.e., having the same variable name) as key.
  - If you supply a set of strings (e.g., `by="AIRLINE"`), dplyr use only those variables to merge.
  - If you want to match based on variables with different names in two data, use `by=c("COL_IN_X"="COL_IN_Y")`.

Keys

- Key is a variable that uniquely identifies an observation.
- You can create a key by combining multiple variables when they collectively identifies a unique observation.

- *Primary key* is a key in its own table.
- *Foreign key* is a key in another table.
- *Surrogate key* is an artificially generated key with `mutate()` and `row_number()`.

Let's join flights with airlines and airports.

- To save time, first count flights by airline and airport.
- We can do this multiple ways.

```
1  flights %>% count(AIRLINE, ORIGIN_AIRPORT)
2  flights %>% count(AIRLINE, ORIGIN_AIRPORT, sort=T)
3  flights %>% group_by(AIRLINE, ORIGIN_AIRPORT) %>% tally() %>% ungroup()
4  flights %>% group_by(AIRLINE, ORIGIN_AIRPORT) %>% tally(sort=T) %>% ungroup()
5  crosstab <- flights %>% count(AIRLINE, ORIGIN_AIRPORT, sort=T)
```

- The dplyr function `count` is equivalent to `group_by %>% tally %>% ungroup`.

```
# A tibble: 2,386 x 3
   AIRLINE ORIGIN_AIRPORT     n
   <chr>   <chr>          <int>
 1 DL      ATL           221705
 2 AA      DFW           134270
 3 WN      MDW            76350
 4 WN      LAS            68520
 5 WN      BWI            64063
 6 MQ      ORD            63130
 7 UA      ORD            59538
 8 EV      IAH            58330
 9 EV      ATL            56959
10 DL      MSP            55929
# ... with 2,376 more rows
```

Mutating join: inner_join

```
crosstab %>%
  inner_join(airlines, by=c("AIRLINE"="IATA_CODE")) %>%
  inner_join(airports, by=c("ORIGIN_AIRPORT"="IATA_CODE"))
```

```
# A tibble: 2,386 x 4
   AIRLINE ORIGIN_AIRPORT     n AIRLINE.y
   <chr>   <chr>          <int> <chr>
 1 DL      ATL           221705 Delta Air Lines Inc.
 2 AA      DFW           134270 American Airlines Inc.
 3 WN      MDW            76350 Southwest Airlines Co.
 4 WN      LAS            68520 Southwest Airlines Co.
 5 WN      BWI            64063 Southwest Airlines Co.
 6 MQ      ORD            63130 American Eagle Airlines Inc.
 7 UA      ORD            59538 United Air Lines Inc.
 8 EV      IAH            58330 Atlantic Southeast Airlines
 9 EV      ATL            56959 Atlantic Southeast Airlines
10 DL      MSP            55929 Delta Air Lines Inc.
# ... with 2,376 more rows
```

```
# A tibble: 1,304 x 10
   AIRLINE ORIGIN_AIRPORT     n AIRLINE.y    AIRPORT      CITY    STATE COUNTRY LATITUDE LONGITUDE
   <chr>   <chr>          <int> <chr>        <chr>        <chr>   <chr> <chr>      <dbl>     <dbl>
 1 DL      ATL           221705 Delta Air L… Hartsfield-… Atlan… GA    USA         33.6     -84.4
 2 AA      DFW           134270 American Ai… Dallas/Fort… Dalla… TX    USA         32.9     -97.0
 3 WN      MDW            76350 Southwest A… Chicago Mid… Chica… IL    USA         41.8     -87.8
 4 WN      LAS            68520 Southwest A… McCarran In… Las V… NV    USA         36.1    -115.
 5 WN      BWI            64063 Southwest A… Baltimore-W… Balti… MD    USA         39.2     -76.7
 6 MQ      ORD            63130 American Ea… Chicago O'H… Chica… IL    USA         42.0     -87.9
 7 UA      ORD            59538 United Air … Chicago O'H… Chica… IL    USA         42.0     -87.9
 8 EV      IAH            58330 Atlantic So… George Bush… Houst… TX    USA         30.0     -95.3
 9 EV      ATL            56959 Atlantic So… Hartsfield-… Atlan… GA    USA         33.6     -84.4
10 DL      MSP            55929 Delta Air L… Minneapolis… Minne… MN    USA         44.9     -93.2
# ... with 1,294 more rows
```

- Try other mutating joins (left_join, right_join, full_join) and see if you get what you expect to get. How are they different? Why?

Filtering join: semi_join

```
1  crosstab %>%
2    semi_join(airlines, by=c("AIRLINE"="IATA_CODE")) %>%
3    semi_join(airports, by=c("ORIGIN_AIRPORT"="IATA_CODE"))
```

```
# A tibble: 2,386 x 3
   AIRLINE ORIGIN_AIRPORT     n
   <chr>   <chr>          <int>
 1 DL      ATL           221705
 2 AA      DFW           134270
 3 WN      MDW            76350
 4 WN      LAS            68520
 5 WN      BWI            64063
 6 MQ      ORD            63130
 7 UA      ORD            59538
 8 EV      IAH            58330
 9 EV      ATL            56959
10 DL      MSP            55929
# ... with 2,376 more rows
```

```
# A tibble: 1,304 x 3
   AIRLINE ORIGIN_AIRPORT     n
   <chr>   <chr>          <int>
 1 DL      ATL           221705
 2 AA      DFW           134270
 3 WN      MDW            76350
 4 WN      LAS            68520
 5 WN      BWI            64063
 6 MQ      ORD            63130
 7 UA      ORD            59538
 8 EV      IAH            58330
 9 EV      ATL            56959
10 DL      MSP            55929
# ... with 1,294 more rows
```

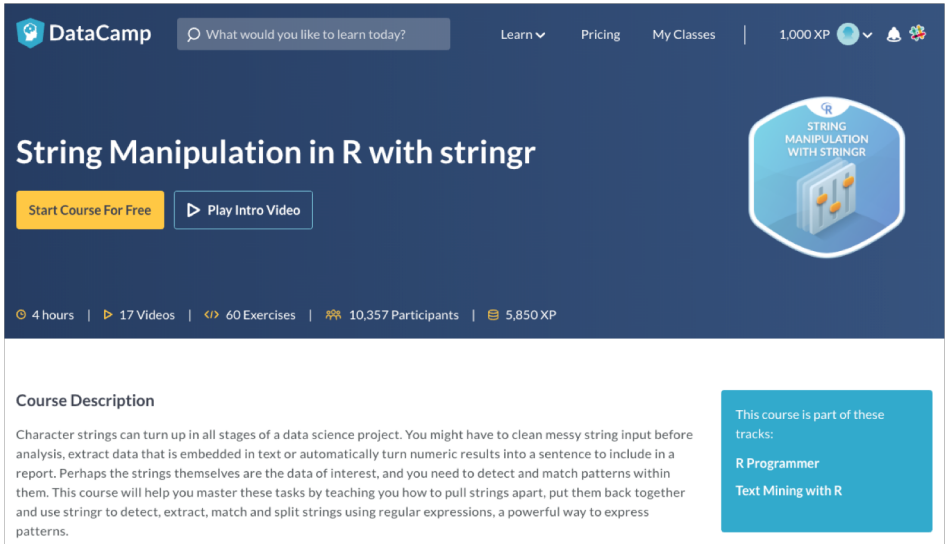
- Try the other filtering join (anti_join) and see if you get what you expect to get. How are they different? Why?

Basic string manipulation with stringr

- Make sure to load stringr package by `library(stringr)`.
  It's not part of core tidyverse because not all data analysis involves textual data.
- All stringr functions start with `str_`.
- Common (easier) functions
  - *str_length* counts the number of characters in a string.
  - *str_c* combines multiple strings into one.
  - *str_sub* extracts a substring from a string.
  - *str_trim* removes whitespaces from start and end of a string.
  - *str_squish* replaces two or more consecutive whitespaces into one.
  - *str_to_lower* lowercases a string. (This is cool => this is cool)
  - *str_to_upper* uppercases a string. (This is cool => THIS IS COOL)
  - *str_to_title* titlecases a string. (This is cool => This Is Cool)

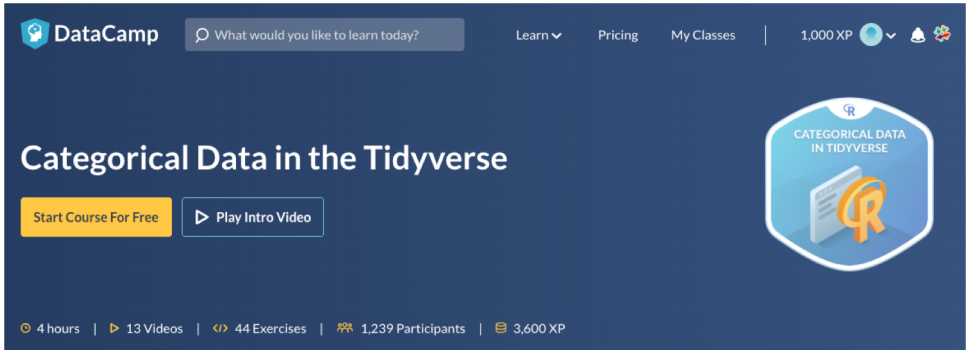
More string manipulation with stringr

- To learn more advanced use of stringr functions, if involving regular expression in particular,
  - Read R4DS Chapter 11: Strings with stringr. https://r4ds.had.co.nz/strings.html
  - Complete DataCamp course: String Manipulation in R with stringr. https://www.datacamp.com/courses/string-manipulation-in-r-with-stringr



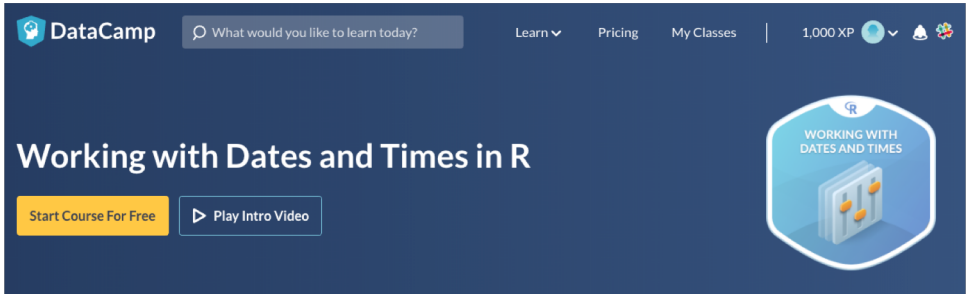
Factor

- Make sure to load forcats package by `library(forcats)`.
- Factor can be considered implicitly ordered categorical variable.
- It's essential to understand factor for customizing a visualization involving a categorical variable.
- To learn more,
  - Read R4DS Chapter 12: Factors with forcats. https://r4ds.had.co.nz/factors.html
  - Complete DataCamp course: Categorical Data in the Tidyverse https://www.datacamp.com/courses/categorical-data-in-the-tidyverse



Date, time, datetime

- Make sure to load lubridate package by `library(lubridate)`.
- Date and time are numerical variables in nature,
  but their notations are rather unique and different from usual numerical variables.
- Understanding how to handle date and time is important for
  analyzing data involving time dimension.
- To learn more,
  - Read R4DS Chapter 13. Dates and Times with lubridate https://r4ds.had.co.nz/dates-and-times.html
  - Complete DataCamp course: Working with Dates and Times in R
    https://www.datacamp.com/courses/working-with-dates-and-times-in-r



Covariation

Relationship between two variables

- Last week we learned how to describe the distribution of a single categorical/discrete/continuous variable.
- A natural next question is about relationship between two variables.

- *Variation* is how much measurements of a single variable vary across observations.
- *Covariation* is how the variation of one variable is associated with the variation of another variable.

Measures of variation and covariation

	Variation (single variable)	Covariation (two variables)
Unscaled	Variance $s_x^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$	Covariance $q_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n-1}$
Normalized		Correlation $r_{x,y} = \frac{q_{x,y}}{s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$

Hand calculation of variance, covariance, and correlation

```r
a <- tibble(x=1:3, y=3:1)
a
c(var(a$x), var(a$y), cov(a$x, a$y), cor(a$x, a$y))
```

```
> a <- tibble(x=1:3, y=3:1)
> a
# A tibble: 3 x 2
      x     y
  <int> <int>
1     1     3
2     2     2
3     3     1
> c(var(a$x), var(a$y), cov(a$x, a$y), cor(a$x, a$y))
[1]  1  1 -1 -1
```

$$s_x^2 = s_y^2 = \frac{(1-2)^2 + (2-2)^2 + (3-1)^2}{3-1} = 1$$

$$q_{x,y} = \frac{(1-2)(3-2) + (2-2)(2-2) + (3-2)(2-3)}{3-1} = -1$$

$$r_{x,y} = \frac{q_{x,y}}{s_x s_y} = \frac{-1}{1 \times 1} = -1$$

Different levels of correlation

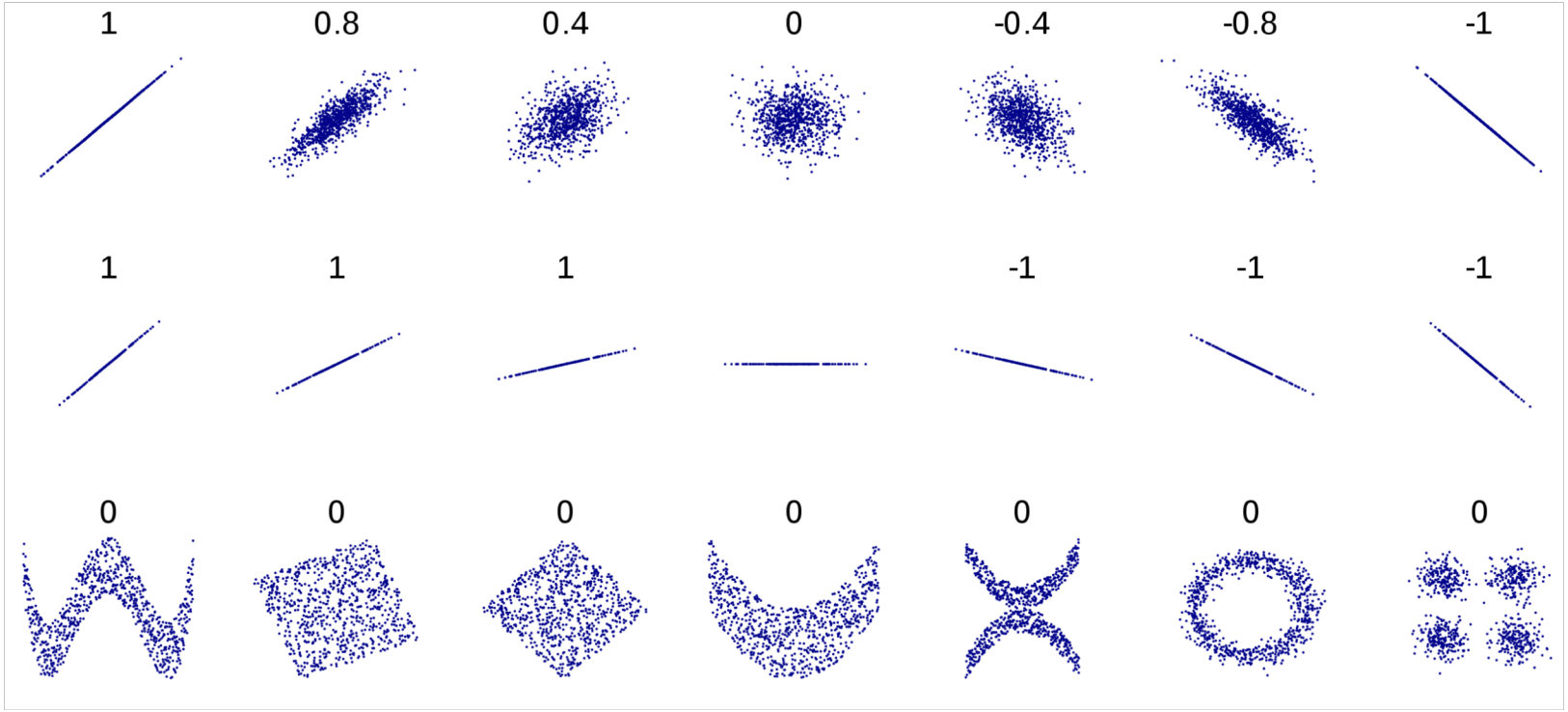


Figure from Wikipedia (https://en.wikipedia.org/wiki/Correlation_and_dependence)

Anscombe's quartet

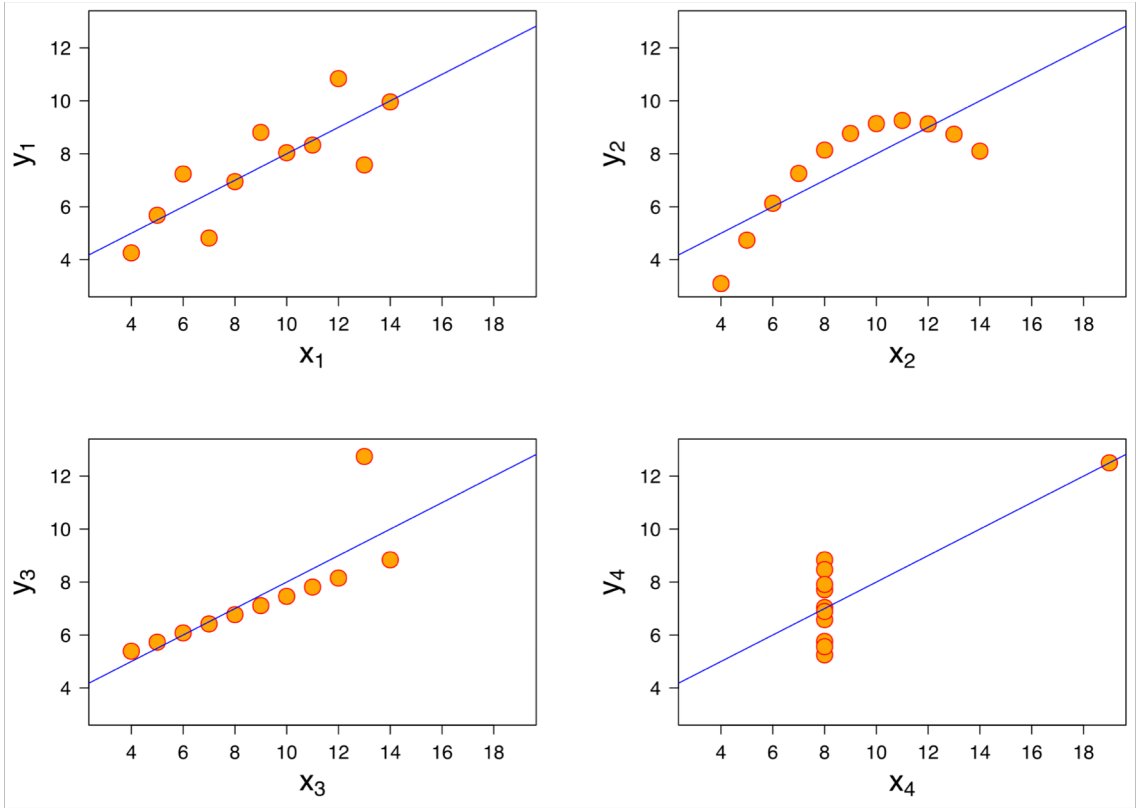


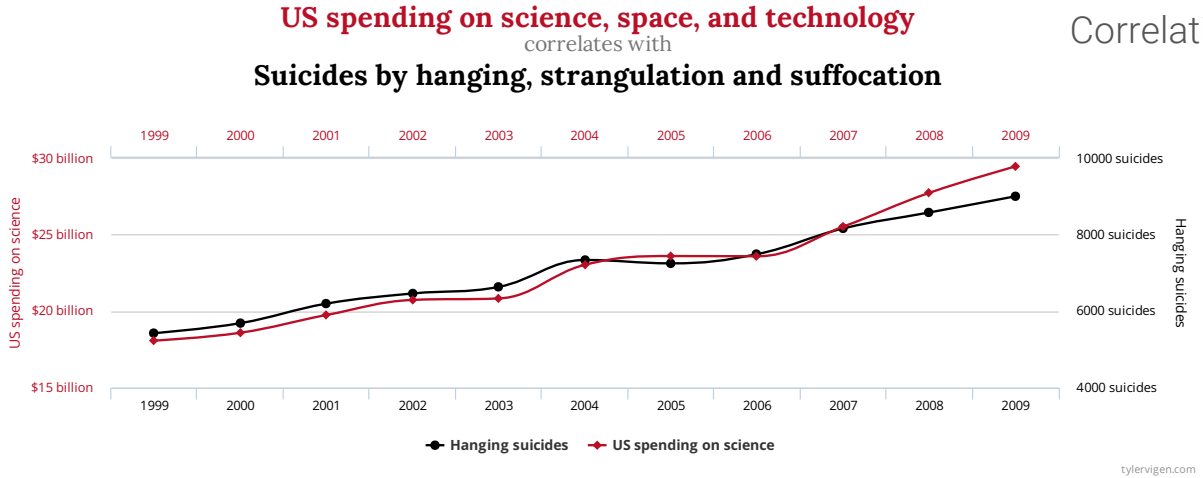
Figure from Wikipedia (https://en.wikipedia.org/wiki/Anscombe%27s_quartet)

Correlation vs. causation

- Short reading on Wikipedia: Correlation does not imply causation.
- When one observes a strong correlation between A and B, one is strongly tempted to conclude “[1] A causes B”. It’s certainly a possibility. But, there are a few alternative scenarios that may give rise to a strong correlation between A and B.
  - [2] B causes A. (Reverse causality)
  - [3] C causes both A and B. (Common causal variable; omitted variable bias)
  - [4] A and B both cause C, and observations were only collected conditioned on C. (Selection bias)
  - [5] A causes B, and B also causes A. (Bidirectional / cyclic causation; imprecise magnitude of effect)
  - [6] A causes C which cases B. (Indirect causation; unknown mechanism)
  - [7] A and B have no connection. (Coincidence)

An example of misled causal conclusion from correlation

- A fun collection of spurious correlations
  http://tylervigen.com/spurious-correlations



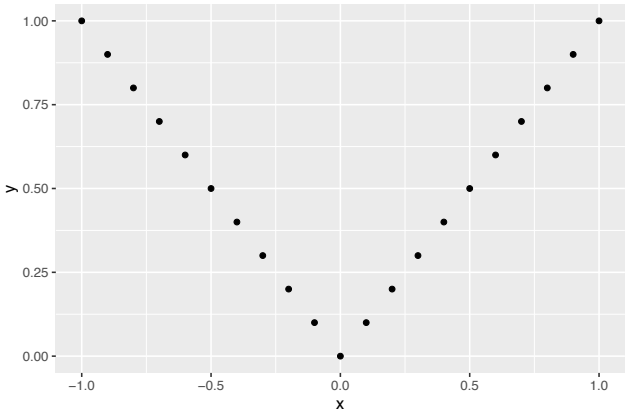
- Is US spending on SST “make” people suicide by hanging? Hell no! Probably not…

Figure from http://tylervigen.com/spurious-correlations

Uncorrelated doesn't mean independence, either.

```
1  a <- tibble(x=seq(-1,1,.1)) %>% mutate(y=abs(x))
2  ggplot(a) + geom_point(aes(x,y))
3  a %>% summarize(cor(x,y))
4  cor(a$x,a$y)
```

Correlation: 0.00000000000000001081586



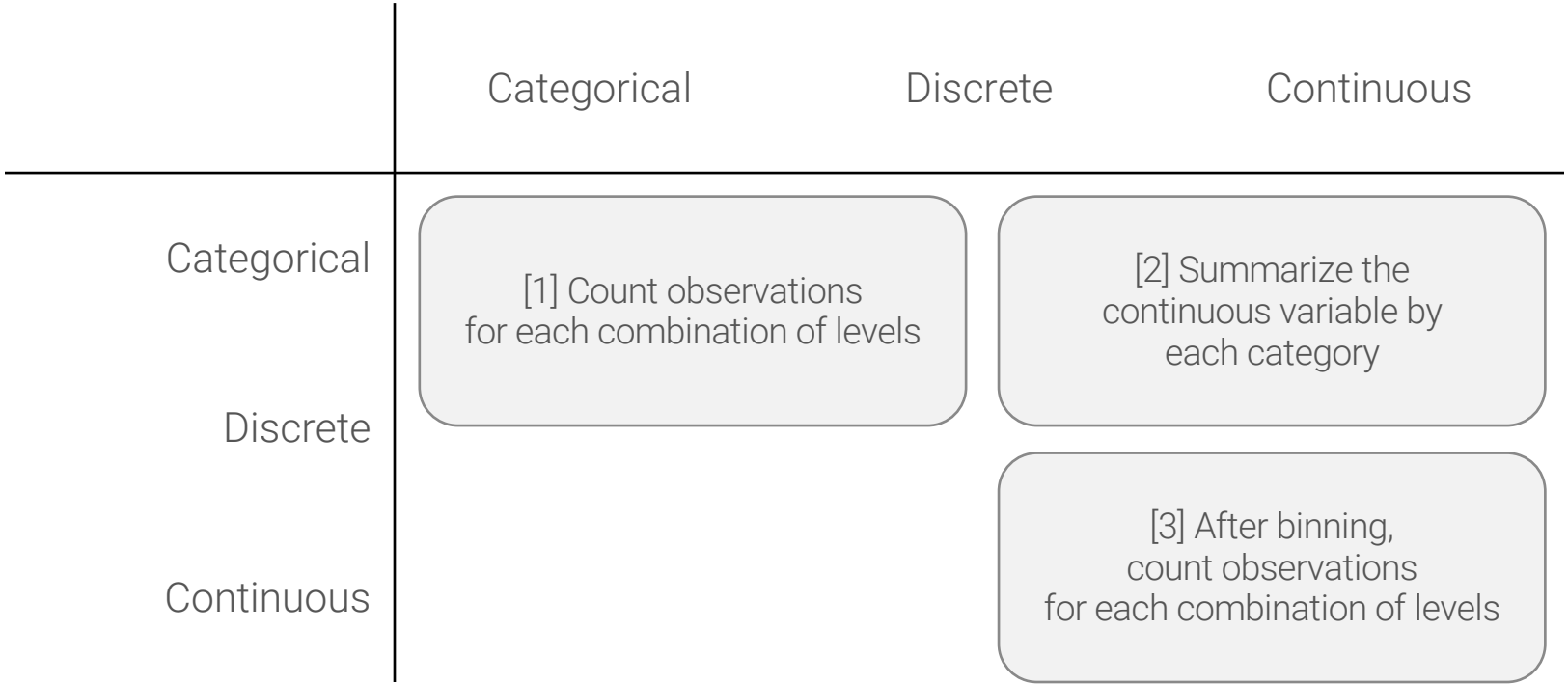

```
> a %>% summarize(cor(x,y))
# A tibble: 1 x 1
  `cor(x, y)`
        <dbl>
1    1.08e-16
> cor(a$x,a$y)
[1] 1.081586e-16
```

True relationship between x and y: $y = |x|$
In this case, x causes y.

How to establish a causal relationship?

- It's really a deep (philosophical, really) problem, especially for social sciences, fundamentally because we cannot rewind history and relive the identically same world with only changes in variables of our interest.
- In order to establish causality, we need counterfactual (i.e., what if A were B and everything else remained the same).
  - Well-designed scientific experiments can achieve this counterfactual with randomization by randomly selecting groups from a population and giving a treatment to only one group.
  - In the business world, A/B testing is such an experiment.
  - Many interesting and important questions in social sciences do not lend themselves to such randomized experiments due to ethical and practical issues.
- Then, what to do with observational data?
  - To me, establishing causality from observational data analysis seems a matter of argumentation with sophisticated slicing-and-dicing of the data.
  - Also, I believe, establishing causality is not the only good goal of data analysis.
  - Descriptive analytics is important at least because it allows us to better understand what's actually happening in the world we live in.
  - Visualization is a great tool for communicating our understanding with each other.

Combination of types of two variables



[1] Cross tabulation (two categorical variables)

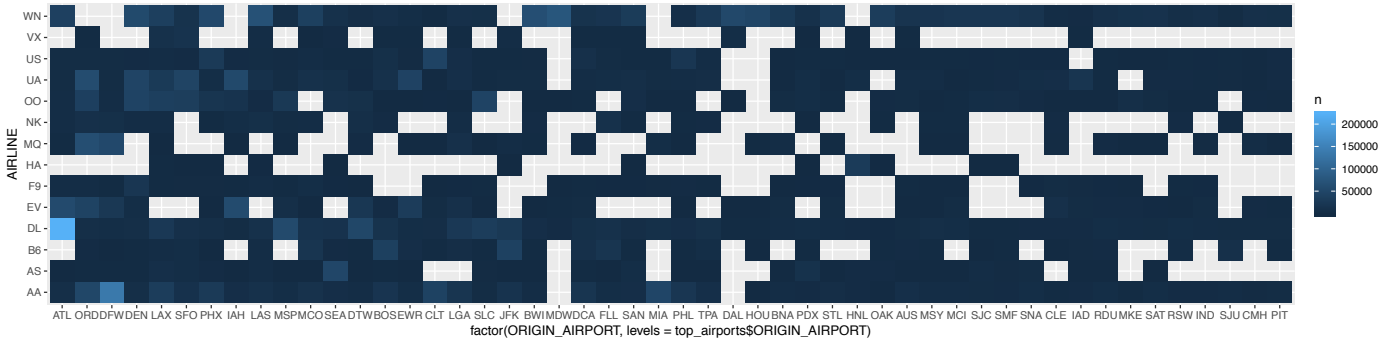
- Count observations for each combination of levels in two categorical variables.
- If you put one variable in rows and the other variable in columns, it's called contingency table.

```
crosstab %>%
  spread(AIRLINE, n) %>%
  replace(is.na(.), 0) %>%
  semi_join(airports, by=c("ORIGIN_AIRPORT"="IATA_CODE")) %>%
  mutate(n=rowSums(.[2:ncol(.)])) %>%
  arrange(-n)
```

```
# A tibble: 322 x 16
   ORIGIN_AIRPORT     AA    AS    B6     DL    EV    F9    HA    MQ    NK    OO    UA    US    VX    WN      n
   <chr>           <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
 1 ATL              7498   429     0 221705 56959  4494     0  1295  4369  4230  3405  2939     0 39513 346836
 2 ORD             50741  1748  2256   6962 44243  4611     0 63130  9965 37858 59538  3452  1380     0 285884
 3 DFW            134270  1348   598   5095 24112  1288     0 53603  8519  4131  3505  3082     0     0 239551
 4 DEN              7107  1589   958   7080  5507 21175     0     0  3691 44633 46218  2329     0 55768 196055
 5 LAX             32738  8071  4108  25171     0  1382  1555   184  4344 36691 27429  3688 11801 37511 194673
 6 SFO             12216  5177  4928   9701     0  1946   668     0     0 34994 45587  2615 15940 14236 148008
 7 PHX             28214  2275   663   6918     8  1870   334     0   966 18552  5469 27781     0 53765 146815
 8 IAH              6501   381     0   2927 58330  1529     0    27  5096 15381 53985  2465     0     0 146622
 9 LAS             11359  4073  4064  10626     0  4327   811     0  9037  2421 11153  3024  3766 68520 133181
10 MSP              5487   587     0  55929  6840  1513     0   187  3495 25513  3208  2086     0  7272 112117
# ... with 312 more rows
```

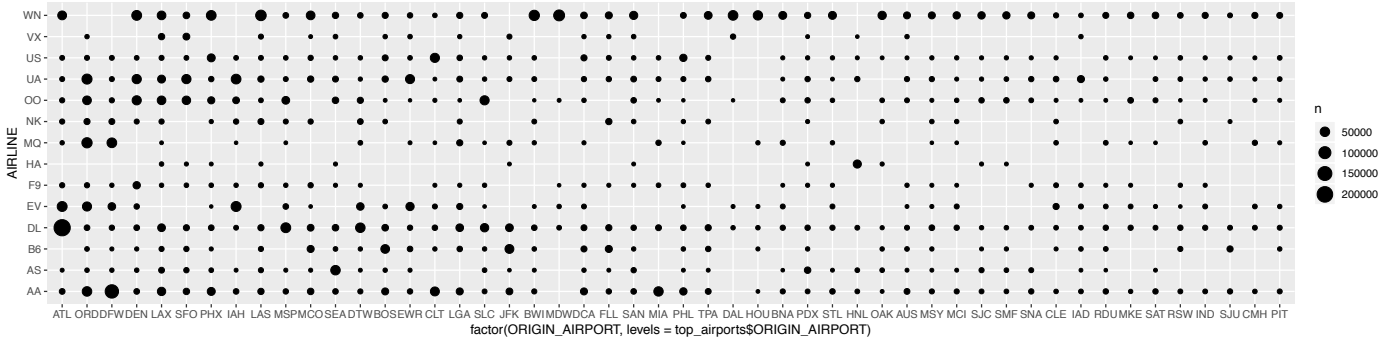
[1] Visualizing crosstab with tile chart (heatmap)

```
1   top_airports <- crosstab %>%
2     semi_join(airports, by=c("ORIGIN_AIRPORT"="IATA_CODE")) %>%
3     group_by(ORIGIN_AIRPORT) %>%
4     summarize(n=sum(n)) %>% arrange(-n) %>% head(50)
5
6   filtered_crosstab <- crosstab %>%
7     semi_join(top_airports, by="ORIGIN_AIRPORT")
8
9   ggplot(filtered_crosstab) +
10    geom_tile(aes(x=AIRLINE, y=factor(ORIGIN_AIRPORT,
    levels=top_airports$ORIGIN_AIRPORT), fill=n)) +
11    coord_flip()
```



[1] Visualizing crosstab with bubble chart

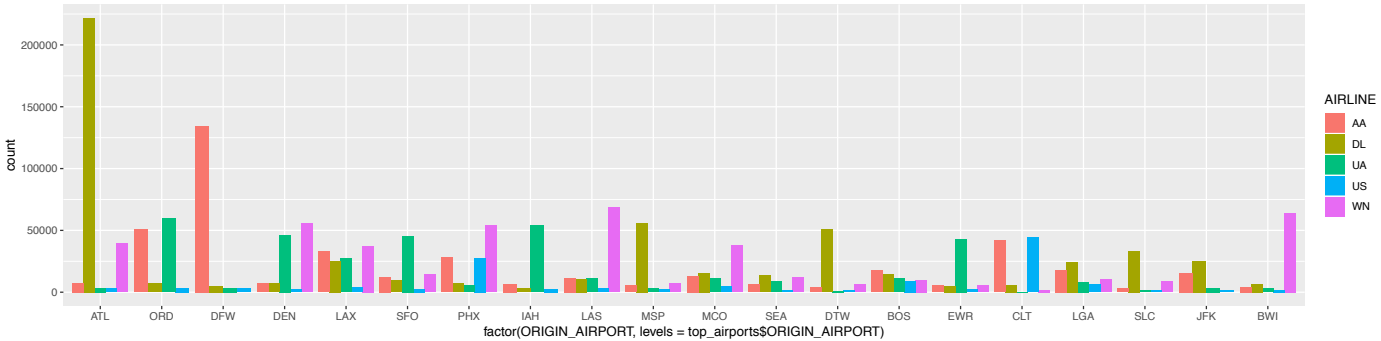
```
1  ggplot(filtered_crosstab) +
2    geom_point(aes(x=AIRLINE, y=factor(ORIGIN_AIRPORT,
   levels=top_airports$ORIGIN_AIRPORT), size=n)) +
3    coord_flip()
```



- Just like bar chart shows distribution of a categorical variable, these charts are like a 3-D bar chart looked down from above.

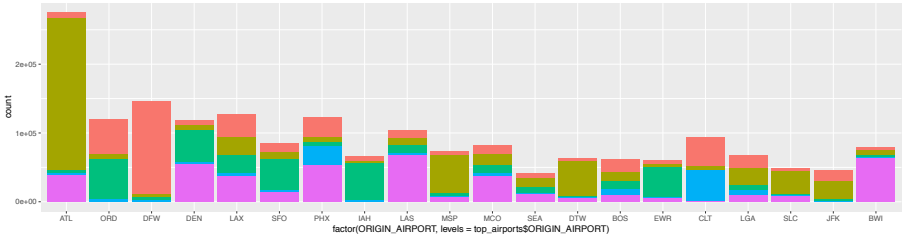
[1] Grouped bar chart

```
top_airports <- crosstab %>%
  semi_join(airports, by=c("ORIGIN_AIRPORT"="IATA_CODE")) %>%
  group_by(ORIGIN_AIRPORT) %>%
  summarize(n=sum(n)) %>% arrange(-n) %>% head(20)
filtered_flights <- flights %>%
  semi_join(top_airports, by="ORIGIN_AIRPORT") %>%
  filter(grepl("AA|DL|WN|UA|US", AIRLINE))
ggplot(filtered_flights) +
  geom_bar(aes(
    x=factor(ORIGIN_AIRPORT, levels=top_airports$ORIGIN_AIRPORT),
    fill=AIRLINE),
  position="dodge")
```



[1] Stacked bar chart & proportion stacked bar chart

```
1  ggplot(filtered_flights) +
2    geom_bar(aes(
3      x=factor(ORIGIN_AIRPORT, levels=top_airports$ORIGIN_AIRPORT),
4      fill=AIRLINE), position="stack")
5
6  ggplot(filtered_flights) +
7    geom_bar(aes(
8      x=factor(ORIGIN_AIRPORT, levels=top_airports$ORIGIN_AIRPORT),
9      fill=AIRLINE), position="fill")
```



[2] Summarize by group (categorical + continuous variables)

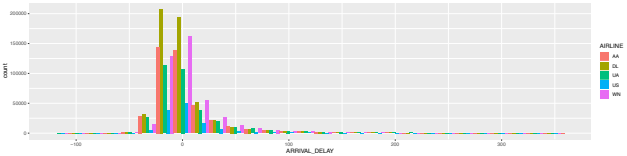
- The true power of `summarize` is realized when used with the `group_by`.
- The typical syntax is:
  - ```
    df %>%
      group_by(<VARIABLES>) %>%
      summarize(<NEW_SUMMARY_VARIABLE> = <STAT_FUNCTION>(), count = n())
    ```

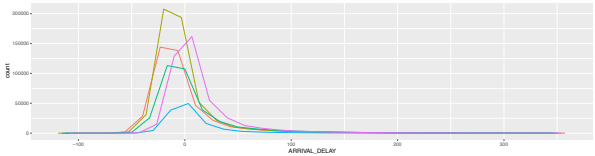
```
1  top_airports <- crosstab %>%
2    semi_join(airports, by=c("ORIGIN_AIRPORT"="IATA_CODE")) %>%
3    group_by(ORIGIN_AIRPORT) %>%
4    summarize(n=sum(n)) %>% arrange(-n) %>% head(50)
```

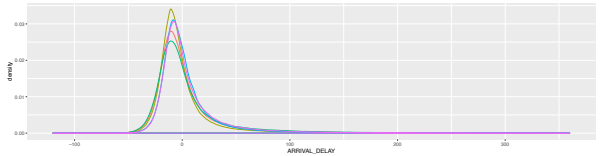
```
> top_airports
# A tibble: 50 x 2
   ORIGIN_AIRPORT      n
   <chr>           <int>
 1 ATL            346836
 2 ORD            285884
 3 DFW            239551
 4 DEN            196055
 5 LAX            194673
 6 SFO            148008
 7 PHX            146815
 8 IAH            146622
 9 LAS            133181
10 MSP            112117
# ... with 40 more rows
```
```

[2] Grouped histogram & line plot & density plot

```r
ggplot(filtered_flights) +
  geom_histogram(aes(x=ARRIVAL_DELAY, fill=AIRLINE), position="dodge") +
  xlim(c(-120,360))
ggplot(filtered_flights) +
  geom_freqpoly(aes(x=ARRIVAL_DELAY, color=AIRLINE), position="dodge") +
  xlim(c(-120,360))
ggplot(filtered_flights) +
  geom_density(aes(x=ARRIVAL_DELAY, color=AIRLINE), position="dodge") +
  xlim(c(-120,360))
```

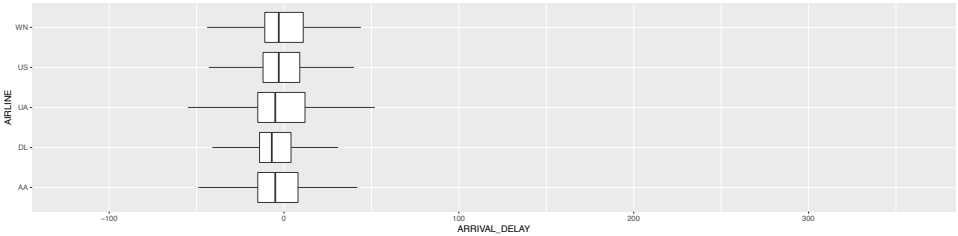






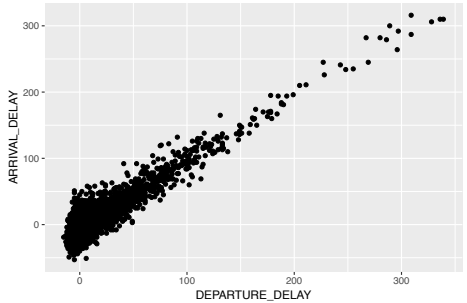
[2] Grouped box plot & violin plot

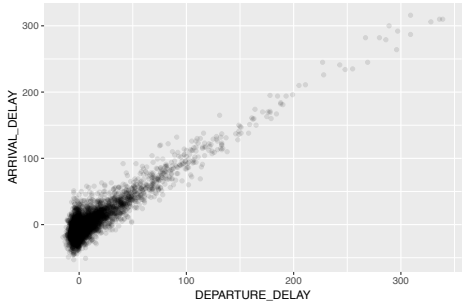
```r
ggplot(filtered_flights) +
  geom_boxplot(aes(x=AIRLINE, y=ARRIVAL_DELAY), outlier.shape=NA) +
  ylim(c(-120,360)) + coord_flip()

ggplot(filtered_flights) +
  geom_violin(aes(x=AIRLINE, y=ARRIVAL_DELAY)) +
  ylim(c(-120,360)) + coord_flip()
```

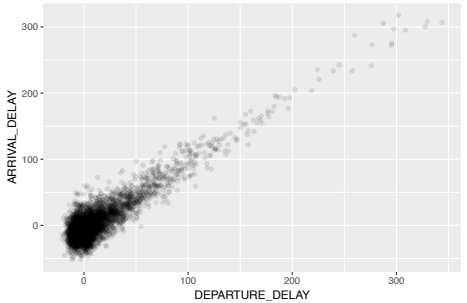


[3] Scatter plot & jittering

```
more_filtered_flights <- filtered_flights %>%
  filter(MONTH==1, DAY==1, !is.na(DEPARTURE_DELAY), !is.na(ARRIVAL_DELAY),
  DEPARTURE_DELAY<400, ARRIVAL_DELAY<400)
ggplot(more_filtered_flights) +
  geom_point(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY))
ggplot(more_filtered_flights) +
  geom_point(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY), alpha=.1)
ggplot(more_filtered_flights) +
  geom_jitter(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY), alpha=.1, width=10,
height=10)
```

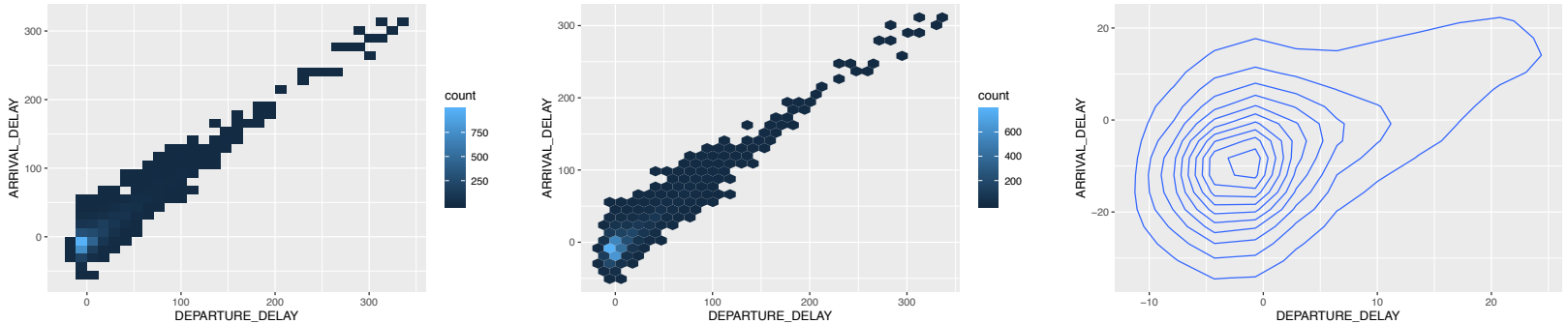






[3] Counting after binning: bin2d, hex, density_2d

```
ggplot(more_filtered_flights) +
  geom_bin2d(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY))
ggplot(more_filtered_flights) +
  geom_hex(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY))
ggplot(more_filtered_flights) +
  geom_density_2d(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY))
```

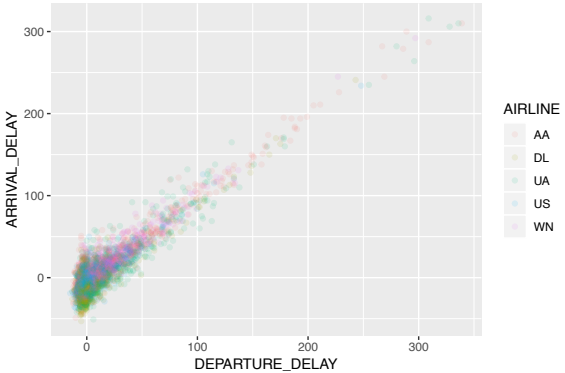


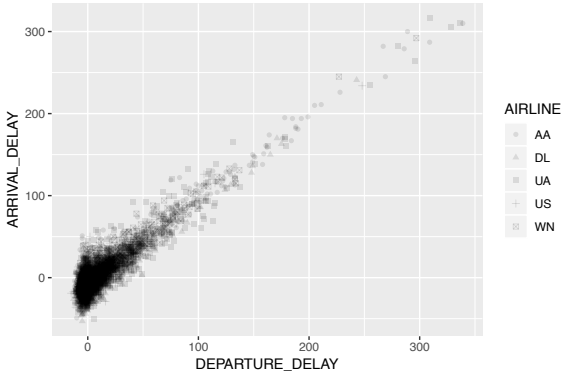
Relationship among three or more variables

- It’s all about visually encoding (=mapping) variables onto aesthetic elements.
- What do we have?
  - Size
  - Color
  - Fill
  - Shape
  - Alpha
  - Line type
  - Weight
  - Facets (small multiples)

Examples encoding more than two variables

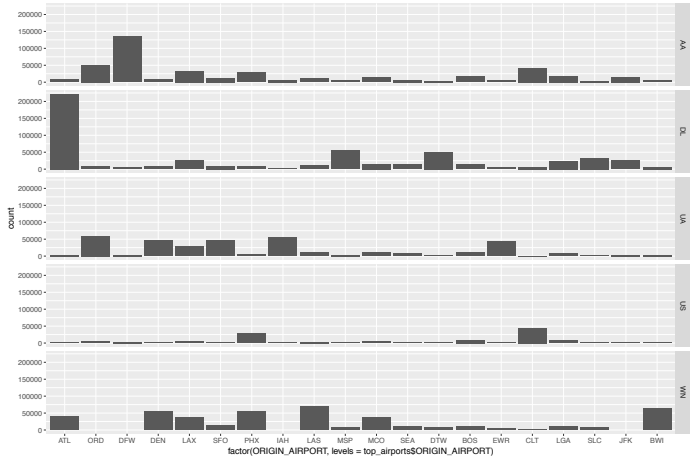
```
1  ggplot(more_filtered_flights) +
2    geom_point(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY, color=AIRLINE),
   alpha=.1)
3  ggplot(more_filtered_flights) +
4    geom_point(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY, shape=AIRLINE),
   alpha=.1)
```

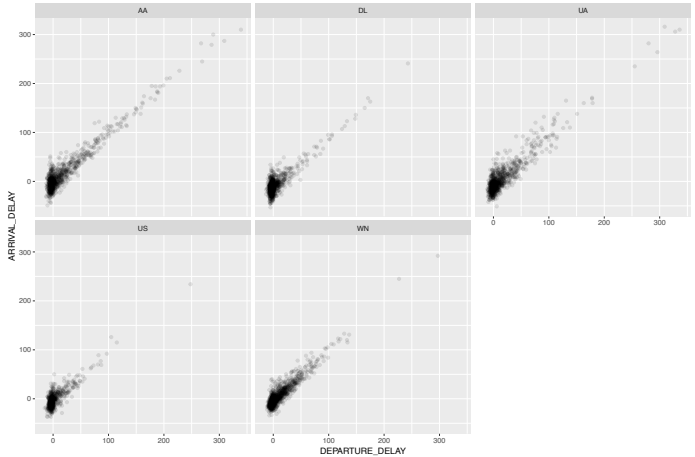




Facets (small multiples)

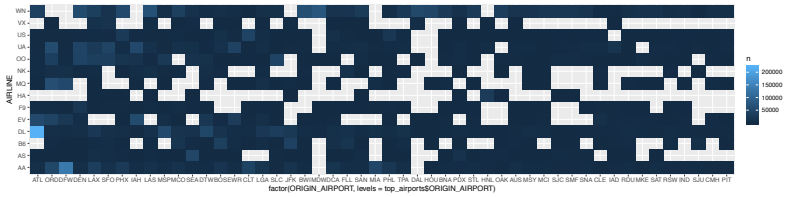
```
1  ggplot(filtered_flights) +
2    geom_bar(aes(x=factor(ORIGIN_AIRPORT,levels=top_airports$ORIGIN_AIRPORT)))+
3    facet_grid(rows=vars(AIRLINE))
4  ggplot(more_filtered_flights) +
5    geom_point(aes(x=DEPARTURE_DELAY, y=ARRIVAL_DELAY), alpha=.1) +
6    facet_wrap(~AIRLINE)
```

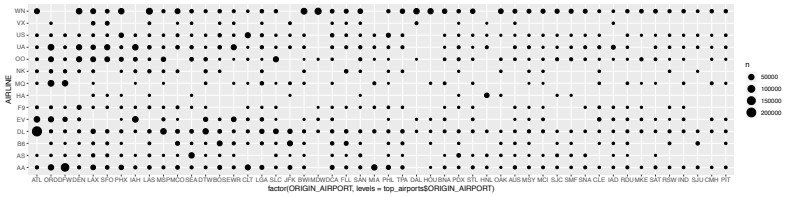




Visualizing grouped summary stats

- So far, we learned how to visualize distribution of a single variable, two variables, and three variables.
- To do that, we just “counted” the number of observations.
- We can plot other summary statistics such as mean instead of count.
- Imagine that color or size in the visualizations below represents average departure delay instead of the number of observations.





Shortcut to create a plot: qplot

- qplot is a convenient wrapper for ggplot. The following example is from `?qplot`.

```
# Use data from data.frame
qplot(mpg, wt, data = mtcars)
qplot(mpg, wt, data = mtcars, colour = cyl)
qplot(mpg, wt, data = mtcars, size = cyl)
qplot(mpg, wt, data = mtcars, facets = vs ~ am)

# qplot will attempt to guess what geom you want depending on the input
# both x and y supplied = scatterplot
qplot(mpg, wt, data = mtcars)
# just x supplied = histogram
qplot(mpg, data = mtcars)
# just y supplied = scatterplot, with x = seq_along(y)
qplot(y = mpg, data = mtcars)

# Use different geoms
qplot(mpg, wt, data = mtcars, geom = "path")
qplot(factor(cyl), wt, data = mtcars, geom = c("boxplot", "jitter"))
qplot(mpg, data = mtcars, geom = "dotplot")
```

Visualization Typology

Dr. Abela's chart chooser

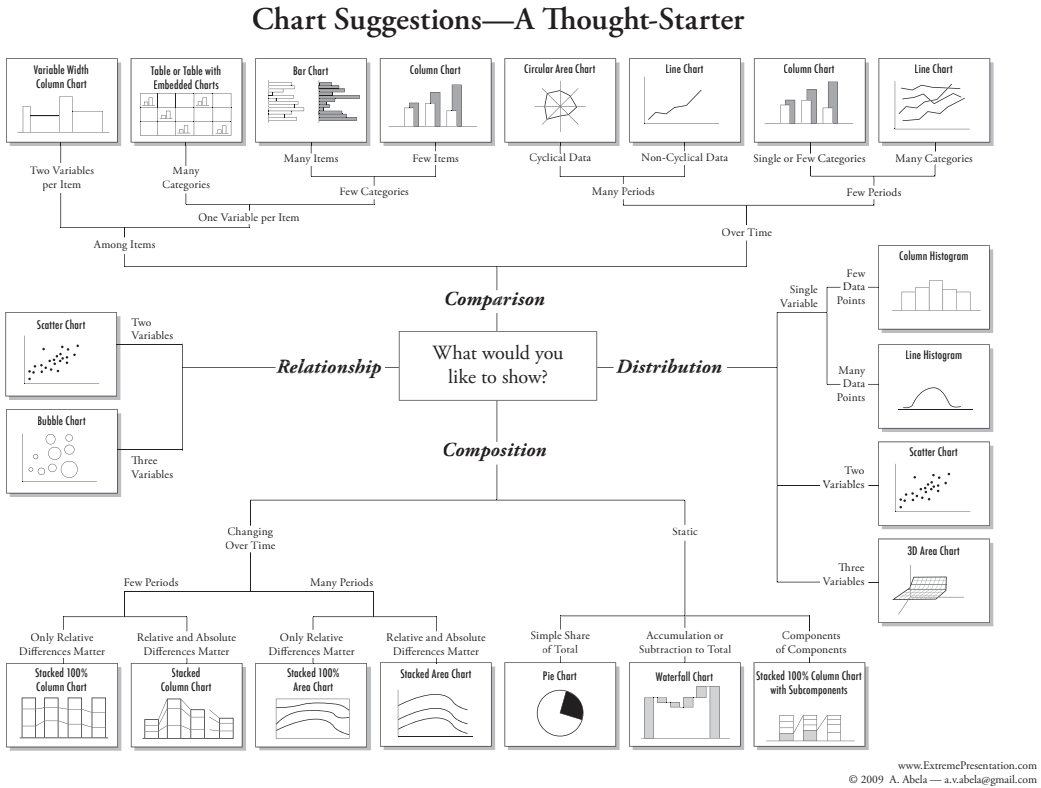


Figure from https://extremepresentation.com/design/7-charts/

A periodic table of visualization methods

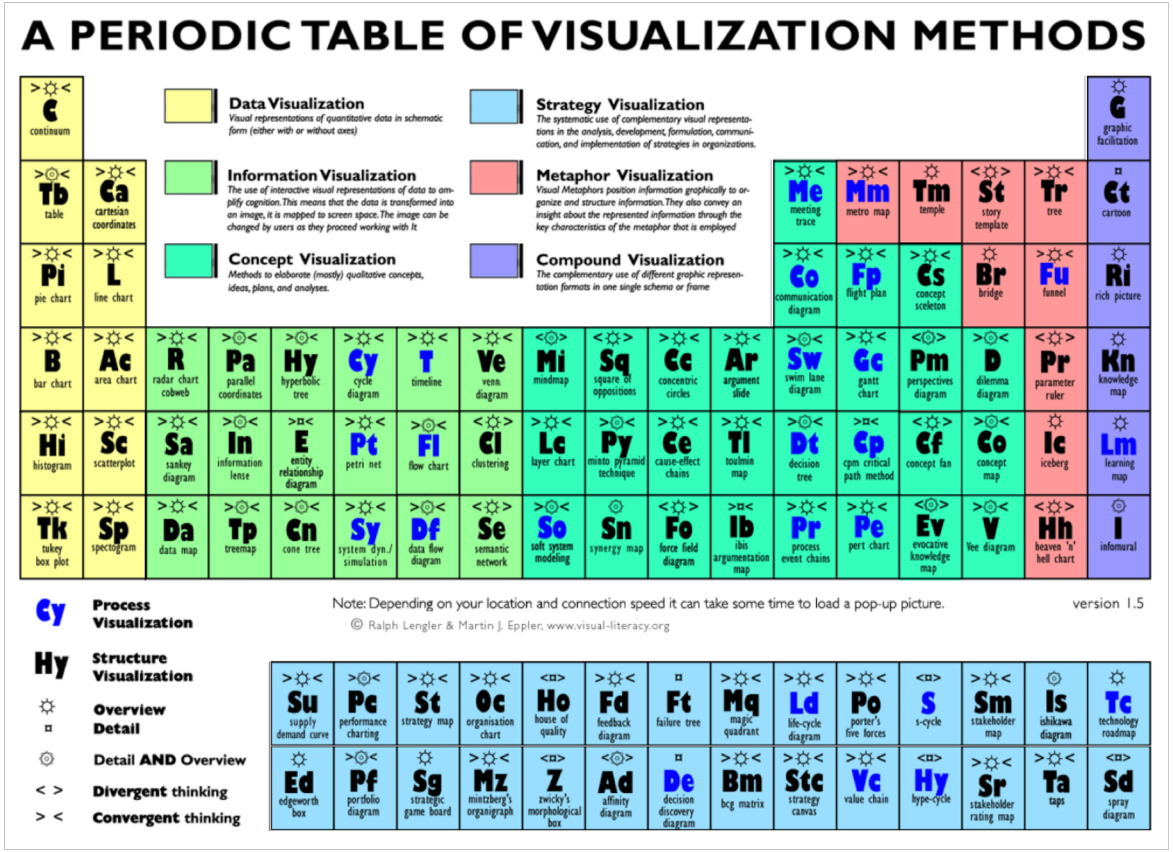


Figure from http://www.visual-literacy.org/periodic_table/periodic_table.html

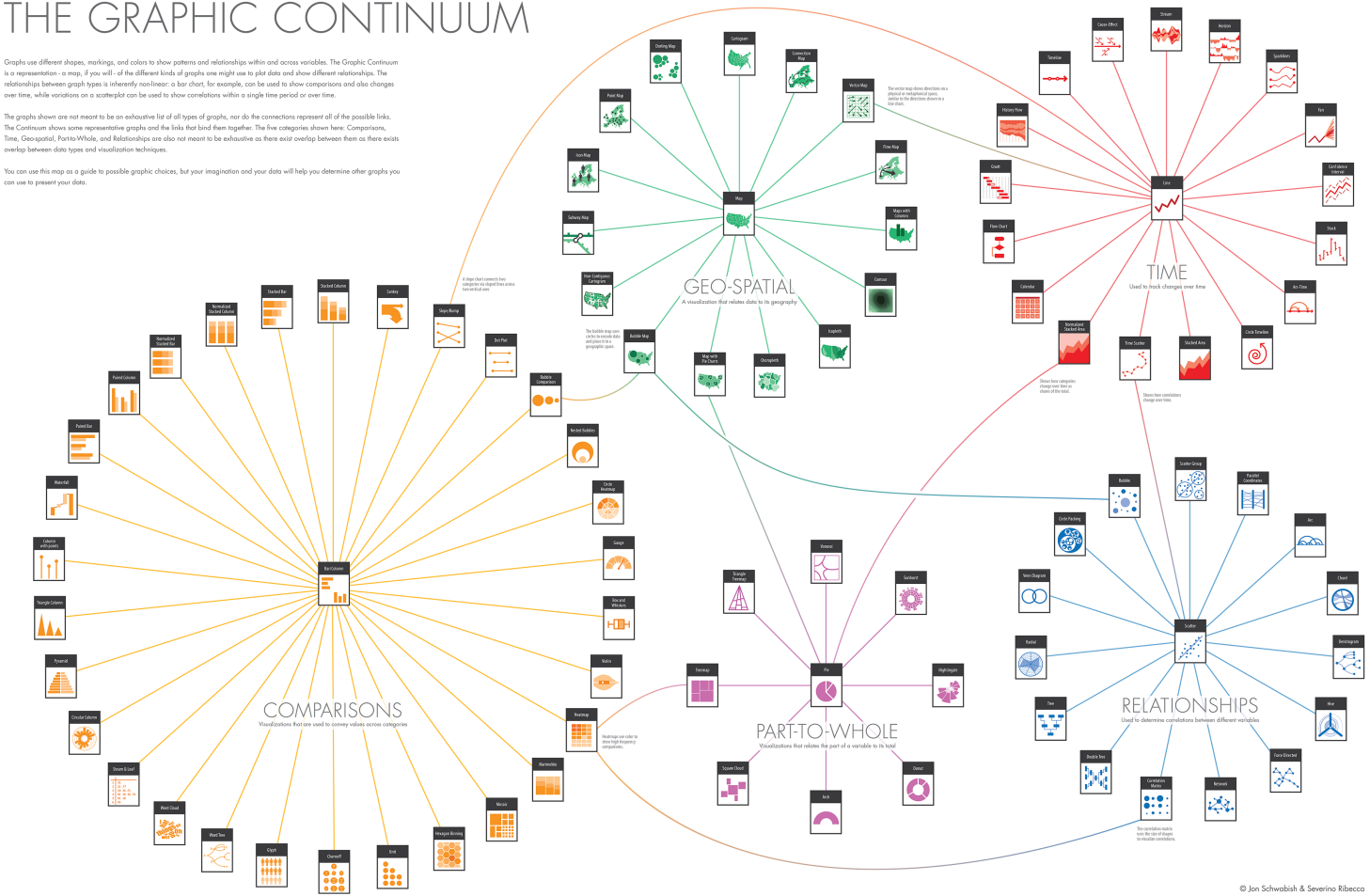


Figure from https://visual.ly/blog/graphic-continuum/

The Data Visualization Catalog

Figure from https://datavizcatalogue.com/

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

required

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

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map) **+ expand_limits**(x = map$long, y = map$lat), map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).

Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | **stat function** | **geommappings**

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() **x, y, size** | ..n.., ..prop..

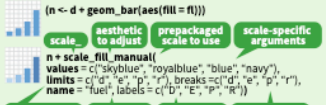
e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



(n <- d + geom_bar(aes(fill = fl)))

scale_ | **aesthetic to adjust** | **prepackaged scale to use** | **scale-specific arguments**

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | **title to use in legend/axis** | **labels to use in legend/axis** | **breaks to use in legend/axis**

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() **+ scale_size**()
p + scale_shape_manual(values = c(3:7))

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(xtrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes
r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

t + facet_wrap(vars(fl))
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

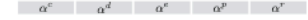
t + facet_grid(rows = vars(drv), cols = vars(fl), scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl), labeller = label_bquote(alpha ^ .(fl)))



Labels

t + labs(x = "New x axis label", **y** = "New y axis label", **title** ="Add a title above the plot", **subtitle** = "Add a subtitle below title", **caption** = "Add a caption below plot", <AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | **manual values for geom's aesthetics**

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) **+ ylim**(10, 20)

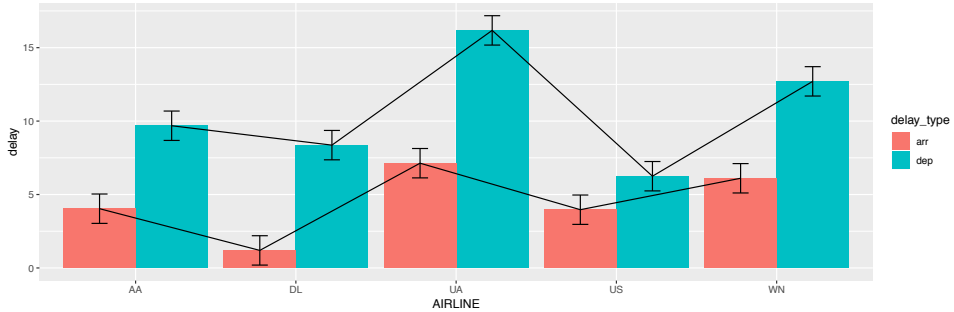
t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous**(limits = c(0, 100))

Customizing Visualization

Layering multiple geoms

- The power of the grammar of graphics is that you can stack up visualizations as long as they share the same core mappings.

```
1  delay_summary <- filtered_flights %>% group_by(AIRLINE) %>%
2    summarize(dep=mean(DEPARTURE_DELAY, na.rm=T), arr=mean(ARRIVAL_DELAY,
   na.rm=T)) %>%
3    gather(delay_type, delay, -AIRLINE)
4
5  ggplot(delay_summary, aes(x=AIRLINE, y=delay, group=delay_type)) +
6    geom_col(aes(fill=delay_type), position="dodge") +
7    geom_line(position=position_dodge(width=0.9)) +
8    geom_errorbar(aes(ymin=delay-1, ymax=delay+1), width=0.2,
   position=position_dodge(width=0.9))
```



Annotation

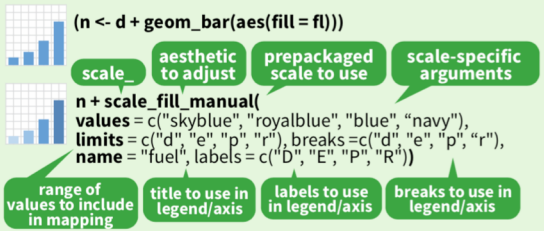
- For plotting data-driven labels, use geom_label or geom_text.
- You can also annotate your plot with text, line, etc.
- Review the following two URLs.
  - geom_text: https://ggplot2.tidyverse.org/reference/geom_text.html
  - annotate: https://ggplot2.tidyverse.org/reference/annotate.html

Legends and labels

- Legend customization
  - http://www.cookbook-r.com/Graphs/Legends_(ggplot2)/
- Title, subtitle, axis labels: ggtitle(), xlab(), ylab()
  - https://ggplot2.tidyverse.org/reference/labs.html
- Tick label style: theme(axis.text.x = …, axis.text.y = …)
  - http://www.sthda.com/english/wiki/ggplot2-axis-ticks-a-guide-to-customize-tick-marks-and-labels

Axis scaling & reordering

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()**)** - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks"**)** - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl)**)**

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red"**)**

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues"**)**

o + scale_fill_gradient(low="red", high="yellow"**)**

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

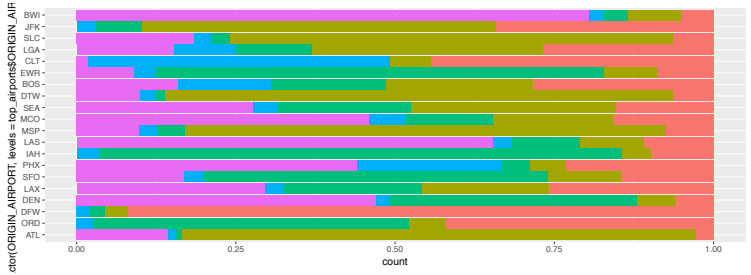
SHAPE AND SIZE SCALES

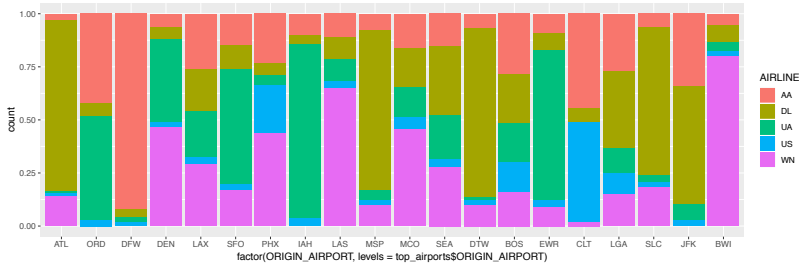
p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)**

p + scale_radius(range = c(1,6)**)**
p + scale_size_area(max_size = 6**)**

Coordinate

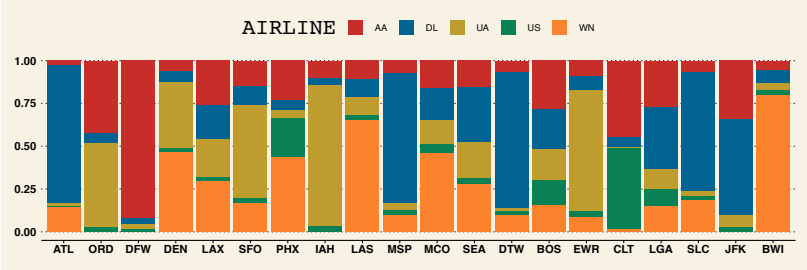
- Mainly used functions
  - coord_cartesian(): default coordinate system
  - coord_polar(): one we used for creating pie chart
  - coord_map(): for geospatial visualization
  - coord_flip(): to interchange x and y axes

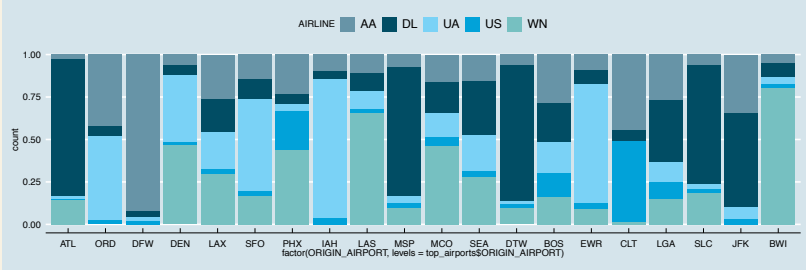


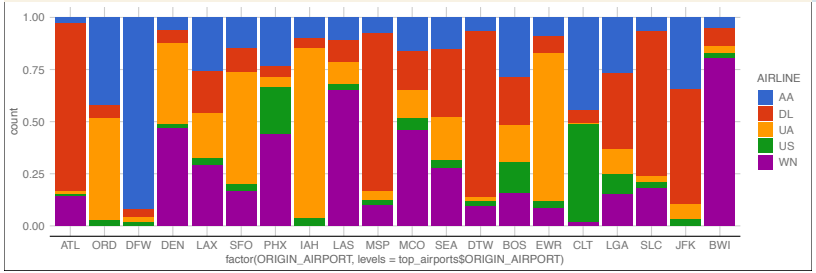


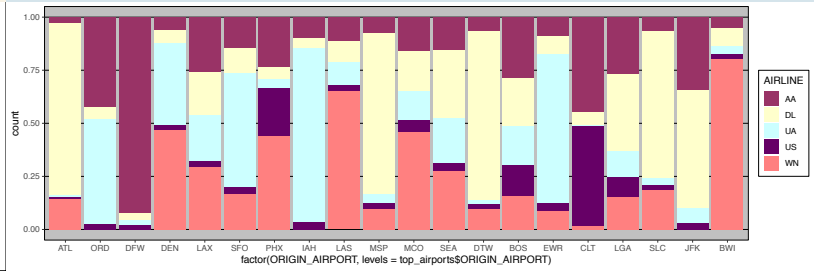
Themes

- Default themes: theme_bw(), theme_linedraw(), theme_light(), theme_dark(), theme_minimal(), theme_classic(), theme_void()
- Additional themes via ggthemes: theme_tufte(), theme_solarized(), theme_excel(), …









Weekly Recap

Things we covered this week

- More data wrangling
  - Join (merge)
  - String
  - Factor
  - Datetime
- Covariation
  - Covariance / Correlation
  - Visualizing two variables
  - Visualizing three or more variables
- Visualization typology
- Customizing visualization
  - Layering multiple geometries
  - Annotation
  - Labels
  - Legends
  - Axes scaling & reordering
  - Coordinate
  - Theme

Things to do this week

- 4 Concept Checker Quizzes (Due: Thursday, January 17, 11:59pm)
  - Week 2.1. Merge, String, Factor, Date
  - Week 2.2. Covariation
  - Week 2.3. Visualization Topology
  - Week 2.4. Customizing Visualization
- 1 Weekly Problem (Due: Friday, January 18, 11:59pm)
- 3 DataCamp Courses (Due: Friday, January 18, 11:59pm)
  - Text Mining with R
    - [1] String Manipulation in R with stringr
  - Individual Courses
    - Categorical Data in the Tidyverse
    - Working with Dates and Times in R
- 1 In-class Activity (Due: Saturday, January 19, 11:59pm)