Data Preparation

Dan Lizotte

2017-01-12

Project

- I have a secret… …your project might not work.
- That is okay. Prove to me and to your classmates that:
  - You thoroughly understand the substantive area and problem
  - You thoroughly understand the data
  - You know what methods are reasonable to try and why
  - You tried several ***and evaluated them rigorously***, but your predictions are just not that good.
- You can't get blood from a turnip. (But demonstrate that as best you can.)

Figure 1:

Data Cleaning

Data cleansing, data cleaning, or data scrubbing is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Data cleansing may be performed interactively with data wrangling tools, or as batch processing through scripting.

(Wikipedia)

Tools

- Your scripting/programming language of choice. (Will look at R today.)
  - Ensures reproducibility.
  - R cheat sheets under Other Resources on the course wiki
- OpenRefine http://openrefine.org
  - Formerly Google Refine. Open source.
- Tableau http://www.tableau.com
  - Industrial-strength. Free student license available.

Data Exploration in R

Reasons to use R

- Easy-ish input from relational database, `.csv`, `.xlsx`
- Nice syntax for data table manipulation
- Elegant plotting
- There's a package for everything. (The python of the statistics world.)

New York City Flights 2013

```
library(nycflights13)
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  19 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num  1400 1416 1089 1576 762 ...
##  $ hour          : num  5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num  15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

```
str(planes)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    3322 obs. of  9 variables:
##  $ tailnum     : chr  "N10156" "N102UW" "N103US" "N104UW" ...
##  $ year        : int  2004 1998 1999 1999 2002 1999 1999 1999 1999 1999 ...
##  $ type        : chr  "Fixed wing multi engine" "Fixed wing multi engine" "Fixed wing multi engine" "
##  $ manufacturer: chr  "EMBRAER" "AIRBUS INDUSTRIE" "AIRBUS INDUSTRIE" "AIRBUS INDUSTRIE" ...
##  $ model       : chr  "EMB-145XR" "A320-214" "A320-214" "A320-214" ...
##  $ engines     : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ seats       : int  55 182 182 182 55 182 182 182 182 182 ...
##  $ speed       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ engine      : chr  "Turbo-fan" "Turbo-fan" "Turbo-fan" "Turbo-fan" ...
```

```
str(weather)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    26130 obs. of  15 variables:
##  $ origin    : chr  "EWR" "EWR" "EWR" "EWR" ...
##  $ year      : num  2013 2013 2013 2013 2013 ...
##  $ month     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ day       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ hour      : int  0 1 2 3 4 6 7 8 9 10 ...
##  $ temp      : num  37 37 37.9 37.9 37.9 ...
##  $ dewp      : num  21.9 21.9 21.9 23 24.1 ...
##  $ humid     : num  54 54 52.1 54.5 57 ...
##  $ wind_dir  : num  230 230 230 230 240 270 250 240 250 260 ...
##  $ wind_speed: num  10.4 13.8 12.7 13.8 15 ...
##  $ wind_gust : num  11.9 15.9 14.6 15.9 17.2 ...
##  $ precip    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pressure  : num  1014 1013 1013 1013 1013 ...
##  $ visib     : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ time_hour : POSIXct, format: "2012-12-31 19:00:00" "2012-12-31 20:00:00" ...
```

```
str(airlines)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    16 obs. of  2 variables:
##  $ carrier: chr  "9E" "AA" "AS" "B6" ...
##  $ name   : chr  "Endeavor Air Inc." "American Airlines Inc." "Alaska Airlines Inc." "JetBlue Airways
```

```
str(airports)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1396 obs. of  7 variables:
##  $ faa : chr  "04G" "06A" "06C" "06N" ...
##  $ name: chr  "Lansdowne Airport" "Moton Field Municipal Airport" "Schaumburg Regional" "Randall Airp
##  $ lat : num  41.1 32.5 42 41.4 31.1 ...
##  $ lon : num  -80.6 -85.7 -88.1 -74.4 -81.4 ...
##  $ alt : int  1044 264 801 523 11 1593 730 492 1000 108 ...
##  $ tz  : num  -5 -5 -6 -5 -4 -4 -5 -5 -5 -8 ...
##  $ dst : chr  "A" "A" "A" "A" ...
```

dplyr

The `dplyr` package in R provides simple functions that correspond to the most common data manipulation operations (or *verbs*) and uses efficient storage approaches.

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

```
verb          meaning
-------------------------------------------
select()      select variables (or columns)
filter()      subset observations (or rows)
mutate()      add new variables (or columns)
arrange()     re-order the observations
summarise()   reduce to a single row
group_by()    aggregate
left_join()   merge two data objects
distinct()    remove duplicate entries
collect()     force computation and bring data back into R
```

Filtering observations

- `filter()` extracts a subset of rows of interest
- Suppose we wanted to find out which airports certain codes belong to?

```
library(dplyr)
airports %>% filter(faa %in% c('ALB', 'BDL', 'BTV'))
```

```
## # A tibble: 3 × 7
##     faa            name      lat       lon   alt    tz   dst
##   <chr>           <chr>    <dbl>     <dbl> <int> <dbl> <chr>
## 1   ALB     Albany Intl 42.74827 -73.80169   285    -5     A
## 2   BDL    Bradley Intl 41.93889 -72.68322   173    -5     A
## 3   BTV Burlington Intl 44.47186 -73.15328   335    -5     A
```

Grouping

```
flights
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1   2013     1     1      517            515         2      830
## 2   2013     1     1      533            529         4      850
## 3   2013     1     1      542            540         2      923
## 4   2013     1     1      544            545        -1     1004
## 5   2013     1     1      554            600        -6      812
## 6   2013     1     1      554            558        -4      740
## 7   2013     1     1      555            600        -5      913
## 8   2013     1     1      557            600        -3      709
## 9   2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
```

```
## #   minute <dbl>, time_hour <dttm>
```

```
bycarrier <- group_by(flights, carrier)
bycarrier
```

```
## Source: local data frame [336,776 x 19]
## Groups: carrier [16]
##
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1   2013     1     1      517            515         2      830
## 2   2013     1     1      533            529         4      850
## 3   2013     1     1      542            540         2      923
## 4   2013     1     1      544            545        -1     1004
## 5   2013     1     1      554            600        -6      812
## 6   2013     1     1      554            558        -4      740
## 7   2013     1     1      555            600        -5      913
## 8   2013     1     1      557            600        -3      709
## 9   2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Grouping and Summaries

```
flights %>% summarise(numflights = n()) #n() counts rows
```

```
## # A tibble: 1 × 1
##   numflights
##        <int>
## 1     336776
```

```
bycarrier %>% summarise(numflights = n()) #n() counts rows
```

```
## # A tibble: 16 × 2
##    carrier numflights
##      <chr>      <int>
## 1       9E      18460
## 2       AA      32729
## 3       AS        714
## 4       B6      54635
## 5       DL      48110
## 6       EV      54173
## 7       F9        685
## 8       FL       3260
## 9       HA        342
## 10      MQ      26397
## 11      OO         32
## 12      UA      58665
## 13      US      20536
## 14      VX       5162
## 15      WN      12275
```

```
## 16      YV        601
```

Aggregating observations

Aggregate the counts of flights at the monthly level.

```
monthlycounts <- flights %>%
   filter(dest %in% c('ALB', 'BDL', 'BTV')) %>%
   group_by(year, month) %>%
   summarise(numflights = n())
monthlycounts
```

```
## Source: local data frame [12 x 3]
## Groups: year [?]
##
##     year month numflights
##    <int> <int>      <int>
## 1   2013     1        324
## 2   2013     2        293
## 3   2013     3        376
## 4   2013     4        297
## 5   2013     5        372
## 6   2013     6        346
## 7   2013     7        253
## 8   2013     8        284
## 9   2013     9        204
## 10  2013    10        248
## 11  2013    11        214
## 12  2013    12        260
```

Aggregating observations

Aggregate the counts of flights at three airports at the monthly level.

```
airportmonthlycounts <- flights %>%
   filter(dest %in% c('ALB', 'BDL', 'BTV')) %>%
   group_by(year, month, dest) %>%
   summarise(numflights = n())
airportmonthlycounts
```

```
## Source: local data frame [36 x 4]
## Groups: year, month [?]
##
##     year month  dest numflights
##    <int> <int> <chr>      <int>
## 1   2013     1   ALB         64
## 2   2013     1   BDL         37
## 3   2013     1   BTV        223
## 4   2013     2   ALB         58
## 5   2013     2   BDL         46
## 6   2013     2   BTV        189
## 7   2013     3   ALB         57
## 8   2013     3   BDL         62
## 9   2013     3   BTV        257
```

```
## 10  2013     4   ALB         13
## # ... with 26 more rows
```

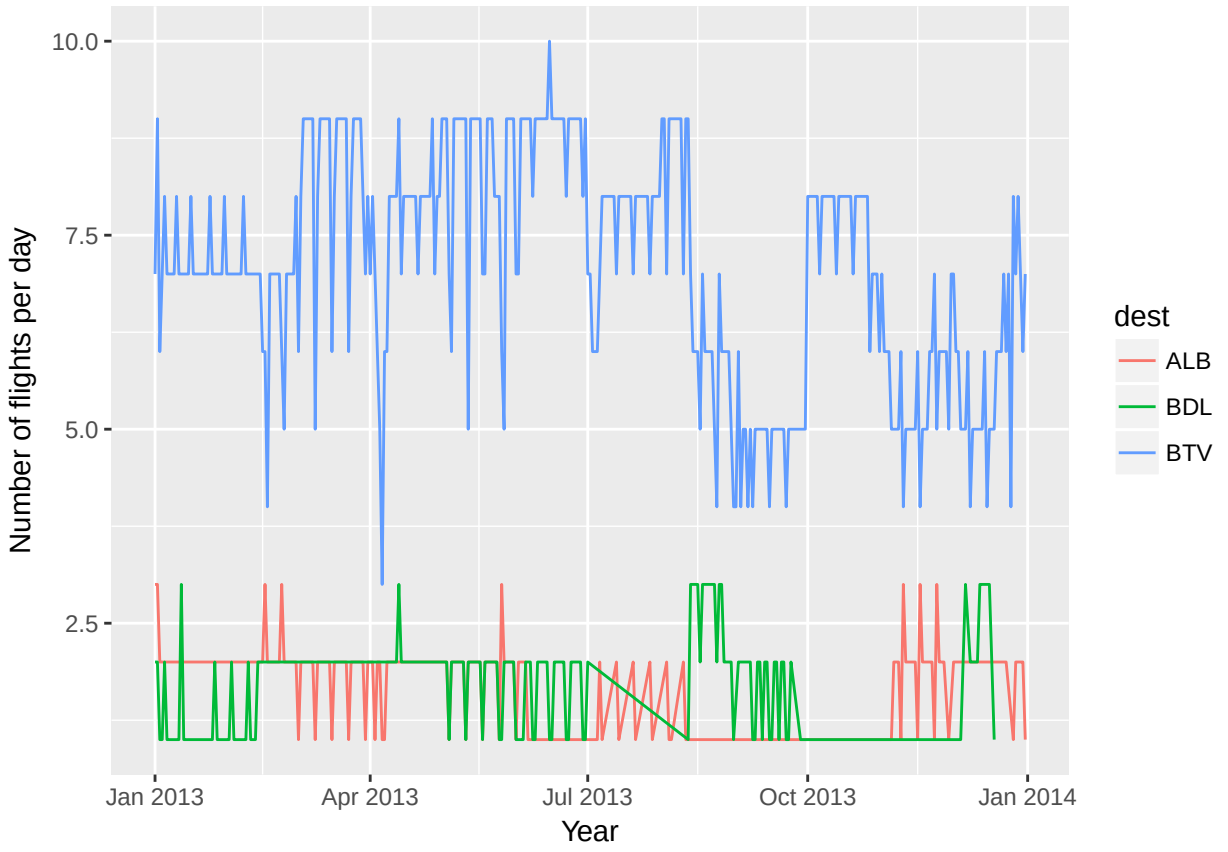
Creating new derived variables

Add a new column by constructing a date variable (using `mutate()` and helper functions from the `lubridate` package), then generate time series plots.

```
library(lubridate)
airportdailycounts <- flights %>%
   filter(dest %in% c('ALB', 'BDL', 'BTV')) %>% group_by(year, month, day, dest) %>%
   summarise(numflights = n()) %>% mutate(date = ymd(paste(year, month, day, sep="-")))
airportdailycounts
```

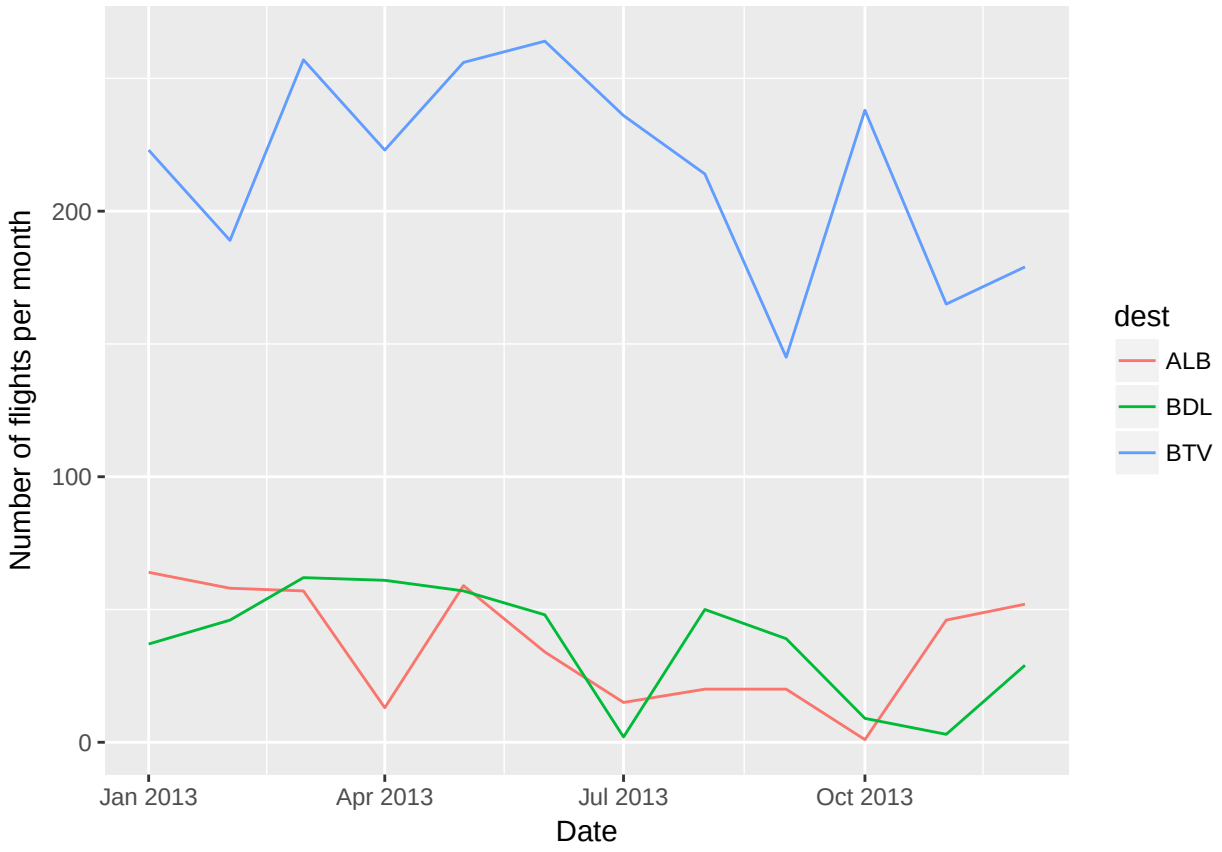
```
## Source: local data frame [876 x 6]
## Groups: year, month, day [365]
##
##     year month   day  dest numflights       date
##    <int> <int> <int> <chr>      <int>     <date>
## 1   2013     1     1   ALB          3 2013-01-01
## 2   2013     1     1   BDL          2 2013-01-01
## 3   2013     1     1   BTV          7 2013-01-01
## 4   2013     1     2   ALB          3 2013-01-02
## 5   2013     1     2   BDL          2 2013-01-02
## 6   2013     1     2   BTV          9 2013-01-02
## 7   2013     1     3   ALB          2 2013-01-03
## 8   2013     1     3   BDL          1 2013-01-03
## 9   2013     1     3   BTV          6 2013-01-03
## 10  2013     1     4   ALB          2 2013-01-04
## # ... with 866 more rows
```

```
library(ggplot2)
qplot(date, numflights, colour=dest, geom = "line",
xlab="Year", ylab="Number of flights per day", data=airportdailycounts)
```



Plot by month instead

```
airportmonthlycounts <- airportmonthlycounts %>%
    mutate(FirstOfMonth = ymd(paste(year, "-", month, "-01", sep="")))
qplot(FirstOfMonth, numflights, colour=dest, geom = "line",
xlab="Date", ylab="Number of flights per month", data=airportmonthlycounts)
```



Sorting and selecting

`arrange()` lets us display the months with the largest number of flights.

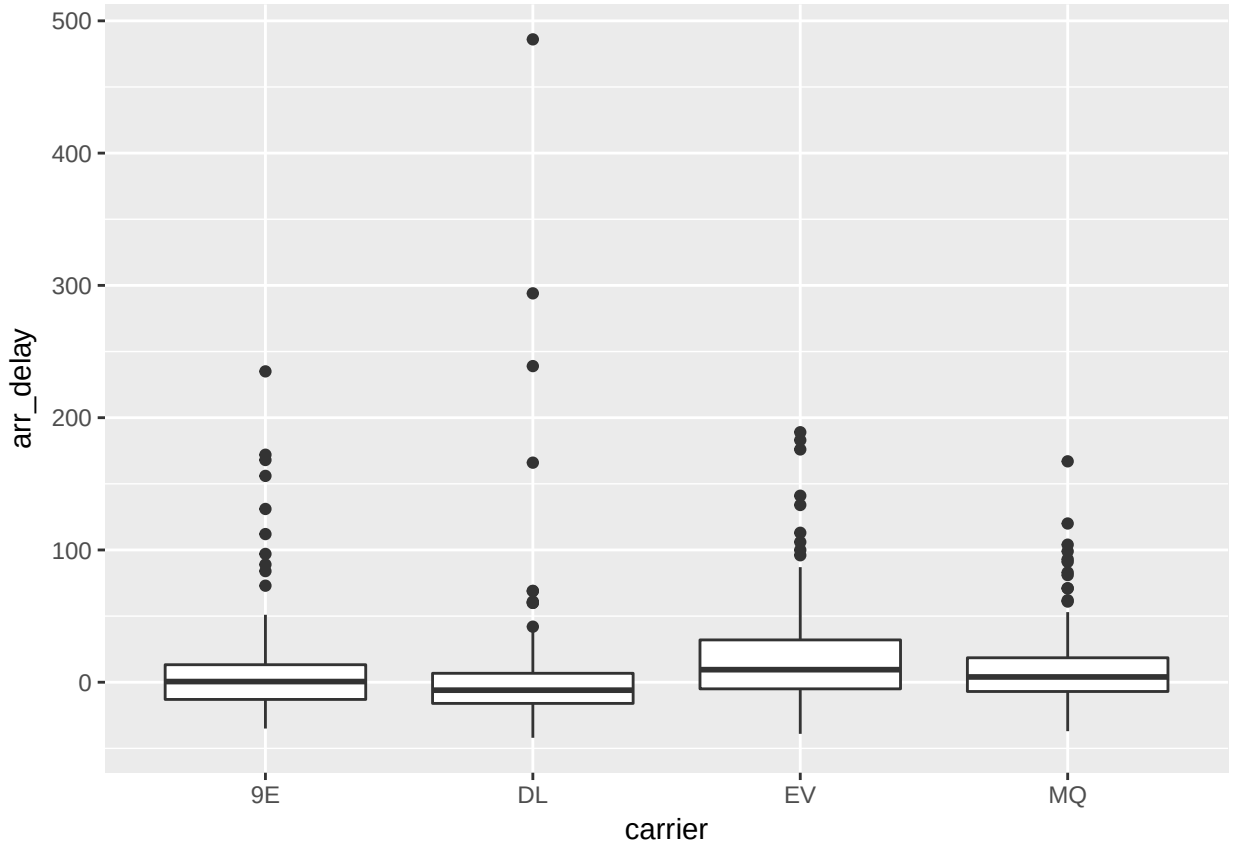
```
airportmonthlycounts %>% arrange(desc(numflights))
```

```
## Source: local data frame [36 x 5]
## Groups: year, month [12]
##
##     year month  dest numflights FirstOfMonth
##    <int> <int> <chr>      <int>       <date>
## 1   2013     6   BTV        264   2013-06-01
## 2   2013     3   BTV        257   2013-03-01
## 3   2013     5   BTV        256   2013-05-01
## 4   2013    10   BTV        238   2013-10-01
## 5   2013     7   BTV        236   2013-07-01
## 6   2013     1   BTV        223   2013-01-01
## 7   2013     4   BTV        223   2013-04-01
## 8   2013     8   BTV        214   2013-08-01
## 9   2013     2   BTV        189   2013-02-01
## 10  2013    12   BTV        179   2013-12-01
## # ... with 26 more rows
```

Comparing airlines

Which airline was most reliable flying from New York to Minneapolis/St. Paul (MSP) in January, 2013?

```
jandelays <- flights %>%
  select(origin, dest, year, month, day, carrier, arr_delay) %>%
  filter(dest == 'MSP' & month == 1)
qplot(carrier, arr_delay, data = jandelays, geom = "boxplot")
```



Merging or "Joining"

Here, the full carrier names are merged (or joined, in database speak) using the `left_join()`

```
merged <- left_join(jandelays, airlines, by=c("carrier" = "carrier"))
merged
```

```
## # A tibble: 546 × 8
##    origin  dest  year month   day carrier arr_delay
##     <chr> <chr> <int> <int> <int>   <chr>     <dbl>
## 1     LGA   MSP  2013     1     1      DL        -8
## 2     EWR   MSP  2013     1     1      EV        29
## 3     LGA   MSP  2013     1     1      MQ        10
## 4     LGA   MSP  2013     1     1      DL         8
## 5     JFK   MSP  2013     1     1      9E        11
## 6     LGA   MSP  2013     1     1      DL       -10
## 7     LGA   MSP  2013     1     1      DL         3
## 8     LGA   MSP  2013     1     1      MQ        93
## 9     LGA   MSP  2013     1     1      DL        -8
## 10    LGA   MSP  2013     1     1      MQ        91
## # ... with 536 more rows, and 1 more variables: name <chr>
```

Truth in Advertising

`?flights` gives description: "On-time data for **all** flights that departed NYC (i.e. JFK, LGA or EWR) in 2013."

```
flights %>% filter(dest == 'ORD') %>% summarize(count = n())
```

```
## # A tibble: 1 × 1
##   count
##   <int>
## 1 17283
```

```
flights %>% filter(dest == 'YYZ') %>% summarize(count = n())
```

```
## # A tibble: 1 × 1
##   count
##   <int>
## 1     0
```

Big Databases

`nycflights13` is just a fraction of the available flight information. See http://www.amherst.edu/~nhorton/precursors for example code using SQLite.

Relational databases, first popularized in the 1970's, provide fast and efficient access to terabyte-sized files. These systems use a structured query language (SQL) to specify data operations.

Database systems have been highly optimized and tuned since they were first invented. Connections between general purpose statistics packages such as R and database systems can be facilitated through use of SQL.

Key operators in SQL

```
verb          meaning
-------------------------------------------
SELECT      create a new result set from a table
FROM        specify table
WHERE       subset observations
GROUP BY    aggregate
ORDER       re-order the observations
DISTINCT    remove duplicate values
JOIN        merge two data objects
```