Machine Learning for Business

Module 3: Data wrangling and visualization

Day 2, 9.00 – 12.00

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

BIG DATA EXPERIENCE CENTER

Module 3 Overview

- Data wrangling with R
- Data visualization with R

BIG DATA EXPERIENCE

Data Wrangling

Part 1

g·able

BIG DATA EXPERIENCE

Wrangling
Munging
Janitor Work
Manipulation
Transformation

50-80%
of your time?

g·able

BIG DATA EXPERIENCE

g·able

Two goals

1. Make data suitable to use with a particular piece of software
2. Reveal information

BIG DATA EXPERIENCE

g·able

Required Packages

Data Wrangling

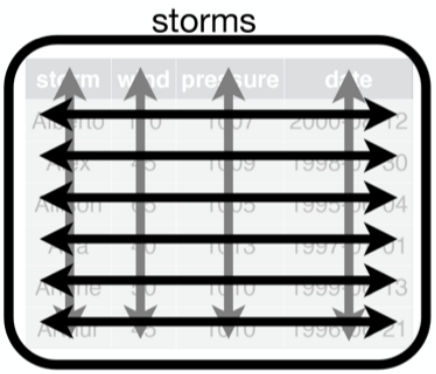
- tidyr
- dplyr

Data Visualization

- ggplot2

BIG DATA EXPERIENCE

Tidy data



1. Each **variable** is saved in its own **column**.
2. Each **observation** is saved in its own **row**.
3. Each "type" of observation stored in a **single table** (here, storms).

g·able

BIG DATA EXPERIENCE

Load data

load("2-DataPrep.RData")

Data	
cases	3 obs. of 4 variables
pollution	6 obs. of 3 variables
storms	6 obs. of 4 variables
tb	3800 obs. of 6 variables

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

cases

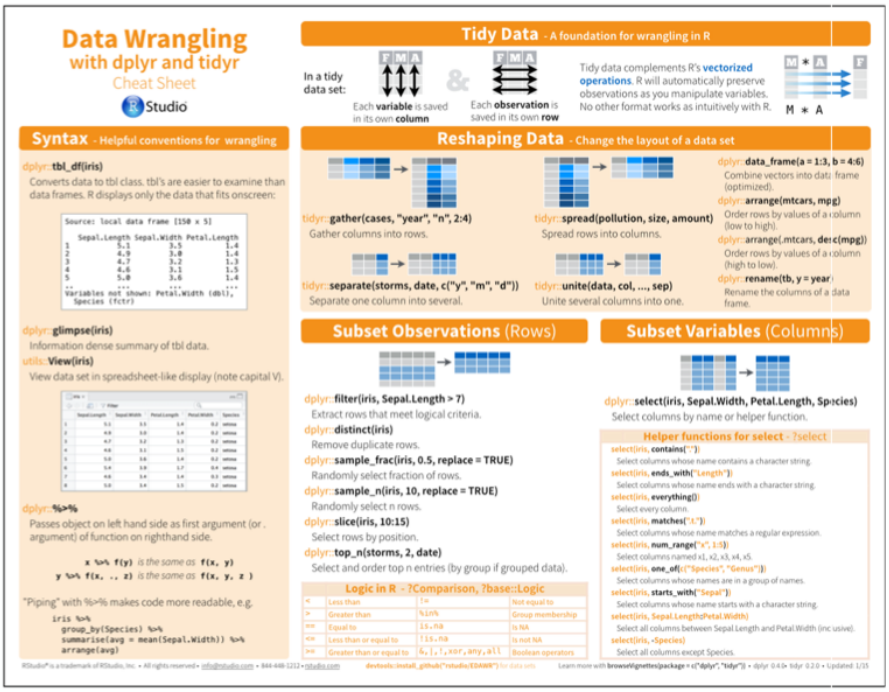
Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

pollution

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

Tidy data

g·able



http://www.rstudio.com/resources/cheatsheets/

Data Preparation Tasks

- Reshaping table: gather, spread
- Reshaping column: unite, separate
- Subsetting rows: filter, distinct, sample_n, top_n
- Subsetting columns: select
- Summarise data: summarise
- Group data: group_by
- Make new variables: mutate
- Integration: left_join, bind_cols

BIG DATA EXPERIENCE

Load library

```
library(tidyr)
library(dplyr)
```

BIG DATA EXPERIENCE

tidyr
Gather columns

```
##   country  2011  2012  2013
## 1      FR  7000  6900  7000
## 2      DE  5800  6000  6200
## 3      US 15000 14000 13000
```

```
##   country year     n
## 1      FR 2011  7000
## 2      DE 2011  5800
## 3      US 2011 15000
## 4      FR 2012  6900
## 5      DE 2012  6000
## 6      US 2012 14000
## 7      FR 2013  7000
## 8      DE 2013  6200
## 9      US 2013 13000
```

```
gather(cases, "year", "n", 2:4)
```

BIG DATA EXPERIENCE

tidyr
Spread columns

```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
## 6  Beijing small     56
```

```
##       city large small
## 1  Beijing   121    56
## 2   London    22    16
## 3 New York    23    14
```

```
spread(pollution, size, amount)
```

g·able

BIG DATA EXPERIENCE

separate()

Separate splits a column by a character string separator.

```
storms2 <- separate(storms, date, c("year", "month", "day"), sep = "-")
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

g·able

BIG DATA EXPERIENCE

unite()

Unite unites columns into a single column.

```
unite(storms2, "date", year, month, day, sep = "-")
```

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

g·able

BIG DATA EXPERIENCE

Ways to access information

1. **Extract** existing variables. — **select()**
2. **Extract** existing observations. — **filter()**
3. **Derive** new variables (from existing variables) — **mutate()**
4. **Change** the unit of analysis — **summarise()**

g·able

BIG DATA EXPERIENCE

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
select(storms, storm, pressure)
```

g·able

BIG DATA EXPERIENCE

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

wind	pressure	date
110	1007	2000-08-12
45	1009	1998-07-30
65	1005	1995-06-04
40	1013	1997-07-01
50	1010	1999-06-13
45	1010	1996-06-21

```
select(storms, -storm)
# see ?select for more
```

g·able

BIG DATA EXPERIENCE

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

wind	pressure	date
110	1007	2000-08-12
45	1009	1998-07-30
65	1005	1995-06-04
40	1013	1997-07-01
50	1010	1999-06-13
45	1010	1996-06-21

```
select(storms, wind:date)
# see ?select for more
```

g·able

BIG DATA EXPERIENCE

Useful select functions

* Blue functions come in dplyr

-	Select everything but
:	Select range
contains()	Select columns whose name contains a character string
ends_with()	Select columns whose name ends with a string
everything()	Select every column
matches()	Select columns whose name matches a regular expression
num_range()	Select columns named x1, x2, x3, x4, x5
one_of()	Select columns whose names are in a group of names
starts_with()	Select columns whose name starts with a character string

g·able

BIG DATA EXPERIENCE

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13

```
filter(storms, wind >= 50)
```

g·able

BIG DATA EXPERIENCE

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04

```
filter(storms, wind >= 50,
  storm %in% c("Alberto", "Alex", "Allison"))
```

g·able

BIG DATA EXPERIENCE

logical tests in R

`?Comparison`

<	Less than
>	Greater than
==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to
%in%	Group membership
is.na	Is NA
!is.na	Is not NA

`?base::Logic`

&	boolean and
|	boolean or
xor	exactly or
!	not
any	any true
all	all true

g·able

BIG DATA EXPERIENCE

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date	ratio	inverse
Alberto	110	1007	2000-08-12	9.15	0.11
Alex	45	1009	1998-07-30	22.42	0.04
Allison	65	1005	1995-06-04	15.46	0.06
Ana	40	1013	1997-07-01	25.32	0.04
Arlene	50	1010	1999-06-13	20.20	0.05
Arthur	45	1010	1996-06-21	22.44	0.04

```
mutate(storms, ratio = pressure / wind, inverse = ratio^-1)
```

g·able

BIG DATA EXPERIENCE

Useful mutate functions

* All take a vector of values and return a vector of values
** Blue functions come in dplyr

pmin(), pmax()	Element-wise min and max
cummin(), cummax()	Cumulative min and max
cumsum(), cumprod()	Cumulative sum and product
between()	Are values between a and b?
cume_dist()	Cumulative distribution of values
cumall(), cumany()	Cumulative all and any
cummean()	Cumulative mean
lead(), lag()	Copy with values one position
ntile()	Bin vector into n buckets
dense_rank(), min_rank(), percent_rank(), row_number()	Various ranking methods

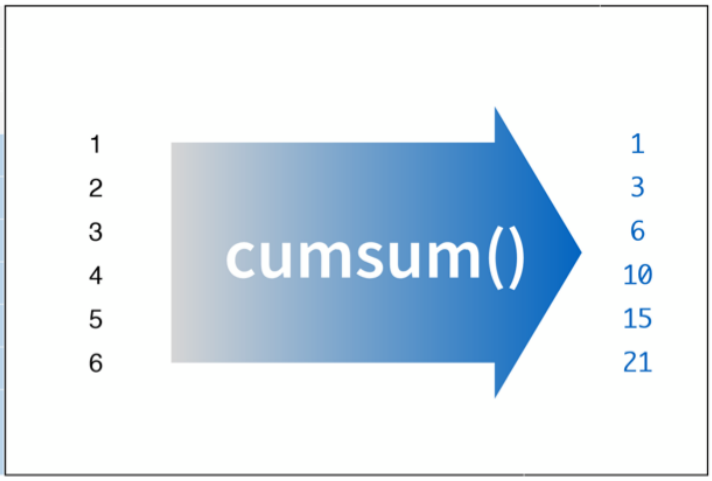
g·able

BIG DATA EXPERIENCE

"Window" functions

* All take a vector of values and return a vector of values

pmin(), pmax()
cummin(), cummax()
cumsum(), cumprod()
between()
cume_dist()
cumall(), cumany()
cummean()
lead(), lag()
ntile()
dense_rank(), min_rank(), percent_rank(), row_number()



g·able

BIG DATA EXPERIENCE

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

```
arrange(storms, wind)
```

g·able

BIG DATA EXPERIENCE

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

```
arrange(storms, wind)
```

g·able

BIG DATA EXPERIENCE

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21
Alex	45	1009	1998-07-30
Ana	40	1013	1997-07-01

```
arrange(storms, desc(wind))
```

g·able

BIG DATA EXPERIENCE

The pipe operator %>%

```
library(dplyr)
select(tb, child:elderly)
tb %>% select(child:elderly)
```


```
tb          select( _____, child:elderly)
```

g·able

BIG DATA EXPERIENCE

g·able

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
select(storms, storm, pressure)
```

BIG DATA EXPERIENCE

g·able

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
storms %>% select(storm, pressure)
```

BIG DATA EXPERIENCE

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Allison	1005
Arlene	1010

```
storms %>%
  filter(wind >= 50) %>%
  select(storm, pressure)
```

g·able

BIG DATA EXPERIENCE

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21


```
storms %>%
  mutate(ratio = pressure / wind) %>%
  select(storm, ratio)
```

g·able

BIG DATA EXPERIENCE

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	ratio
Alberto	9.15
Alex	22.42
Allison	15.46
Ana	25.32
Arlene	20.20
Arthur	22.44

```
storms %>%
  mutate(ratio = pressure / wind) %>%
  select(storm, ratio)
```

g·able

BIG DATA EXPERIENCE

summarise()

city	particle size	amount (μg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

median	variance
22.5	1731.6

```
pollution %>% summarise(median = median(amount), variance = var(amount))
```

g·able

BIG DATA EXPERIENCE

summarise()

city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

```
pollution %>% summarise(mean = mean(amount), sum = sum(amount), n = n())
```

g·able

BIG DATA EXPERIENCE

Useful summary functions

* All take a vector of values and return a single value
** Blue functions come in dplyr

min(), max()	Minimum and maximum values
mean()	Mean value
median()	Median value
sum()	Sum of values
var, sd()	Variance and standard deviation of a vector
first()	First value in a vector
last()	Last value in a vector
nth()	Nth value in a vector
n()	The number of values in a vector
n_distinct()	The number of distinct values in a vector

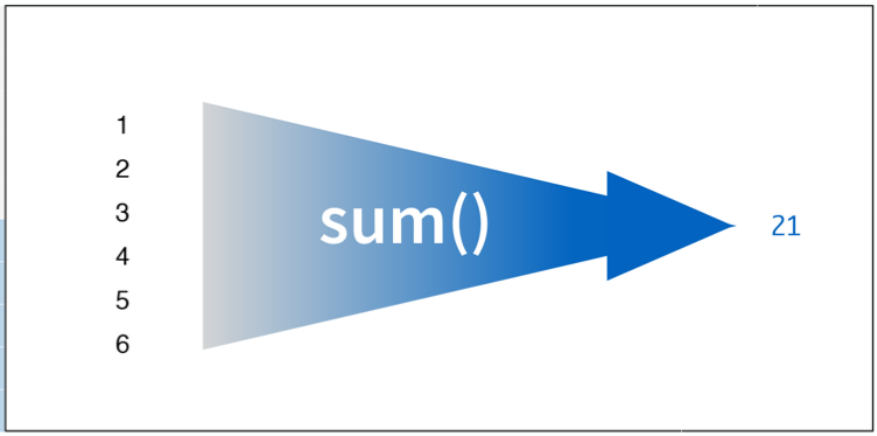
g·able

BIG DATA EXPERIENCE

"Summary" functions

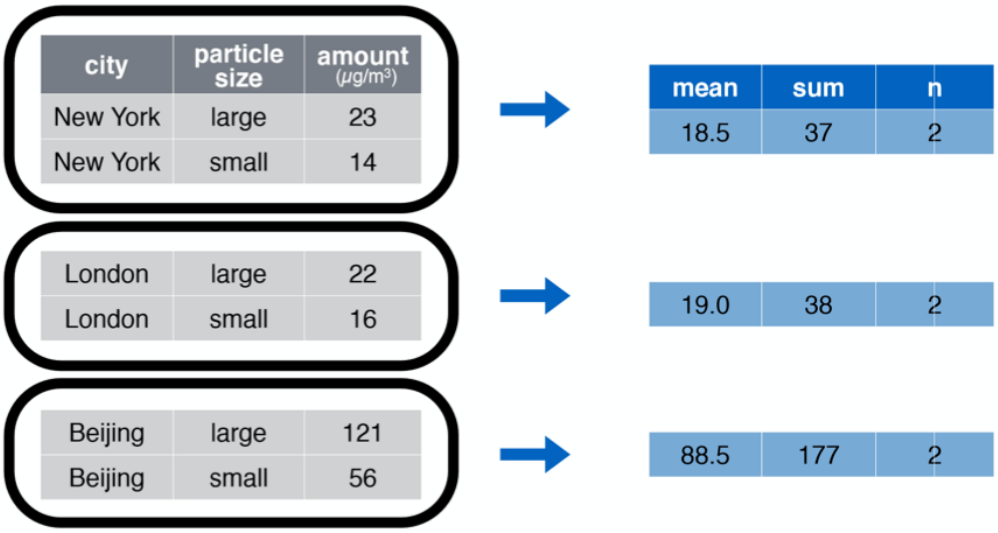
* All take a vector of values and return a single value

min(), max()
mean()
median()
sum()
var, sd()
first()
last()
nth()
n()
n_distinct()



g·able

BIG DATA EXPERIENCE



group_by() + summarise()

g·able

BIG DATA EXPERIENCE

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

→

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

→

city	mean
New York	18.5
London	19.0
Beijing	88.5

```
pollution %>% group_by(city) %>%  summarise(mean = mean(amount))
```

g·able

BIG DATA EXPERIENCE

dplyr::**bind_cols()**

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

```
bind_cols(y, z)
```

g·able

BIG DATA EXPERIENCE

dplyr::**bind_rows()**

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

```
bind_rows(y, z)
```

g·able

BIG DATA EXPERIENCE

dplyr::**left_join()**

songs

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul
Peggy Sue	Buddy

+

artists

name	plays
George	sitar
John	guitar
Paul	bass
Ringo	drums

=

song	name	plays
Across the Universe	John	guitar
Come Together	John	guitar
Hello, Goodbye	Paul	bass
Peggy Sue	Buddy	<NA>

```
left_join(songs, artists, by = "name")
```

g·able

BIG DATA EXPERIENCE

songs2

song	first	last
Across the Universe	John	Lennon
Come Together	John	Lennon
Hello, Goodbye	Paul	McCartney
Peggy Sue	Buddy	Holly

+

artists2

first	last	plays
George	Harrison	sitar
John	Lennon	guitar
Paul	McCartney	bass
Ringo	Starr	drums
Paul	Simon	guitar
John	Coltrane	sax

=

song	first	last	plays
Across the Universe	John	Lennon	guitar
Come Together	John	Lennon	guitar
Hello, Goodbye	Paul	McCartney	bass
Peggy Sue	Buddy	Holly	<NA>

```
left_join(songs2, artists2, by = c("first", "last"))
```

g·able

BIG DATA EXPERIENCE

inner_join()

songs

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul
Peggy Sue	Buddy

+

artists

name	plays
George	sitar
John	guitar
Paul	bass
Ringo	drums

=

song	name	plays
Across the Universe	John	guitar
Come Together	John	guitar
Hello, Goodbye	Paul	bass

```
inner_join(songs, artists, by = "name")
```

g·able

BIG DATA EXPERIENCE

semi_join()

songs

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul
Peggy Sue	Buddy

+

artists

name	plays
George	sitar
John	guitar
Paul	bass
Ringo	drums

=

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul

```
semi_join(songs, artists, by = "name")
```

g·able

BIG DATA EXPERIENCE

anti_join()

songs

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul
Peggy Sue	Buddy

+

artists

name	plays
George	sitar
John	guitar
Paul	bass
Ringo	drums

=

song	name
Peggy Sue	Buddy

```
anti_join(songs, artists, by = "name")
```

g·able

BIG DATA EXPERIENCE

g·able

Source-Sink

```
> storms %>%
+   select(storm, wind) %>%
+   filter(wind >= 50) -> storm1
> storm1
  # A tibble: 3 x 2
      storm  wind
      <chr> <int>
  1 Alberto   110
  2 Allison    65
  3  Arlene    50
```

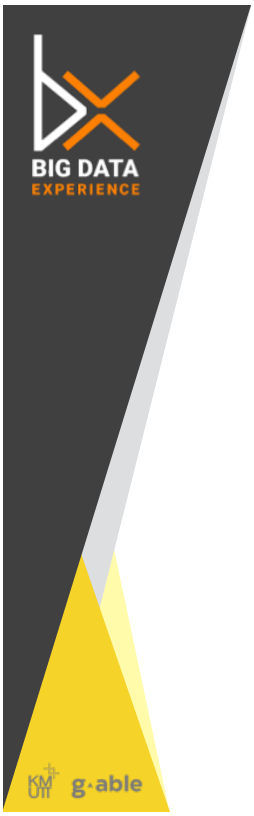
BIG DATA EXPERIENCE

g·able

Exercise 1

- Load two data frames “flights.csv”

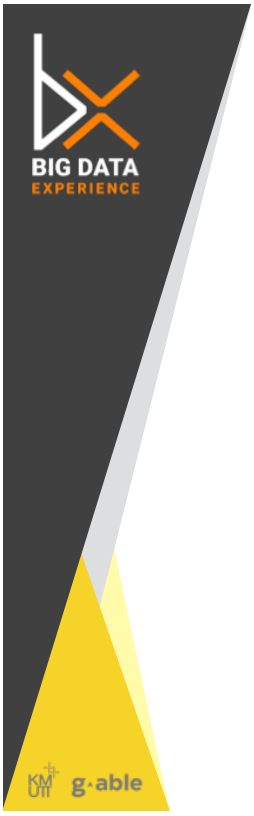
  Hint: `read.csv`
- Select two columns (carrier, dep_delay, arr_delay)
- Filter NA out of dep_delay and arr_delay
- Use carrier as a group and calculate mean of dep_delay and arr_delay
- Sort the worst carrier by departure delay (dep_delay)



Exercise: Solution

```
flightData %>%
   select(carrier,dep_delay,arr_delay) %>%
   filter(!is.na(dep_delay)) %>%
   filter(!is.na(arr_delay)) %>%
   group_by(carrier) %>%
   summarise(dep_delay = mean(dep_delay),
             arr_delay = mean(arr_delay)) -> data2
```

```
# A tibble: 16 x 3
   carrier dep_delay  arr_delay
    <fctr>     <dbl>      <dbl>
 1      9E 16.439574  7.3796692
 2      AA  8.569130  0.3642909
 3      AS  5.830748 -9.9308886
 4      B6 12.967548  9.4579733
 5      DL  9.223950  1.6443409
```



Data Visualization

Part 2

BIG DATA EXPERIENCE

g·able

Why ggplot2?

Advantages of ggplot2

- consistent underlying grammar of graphics (Wilkinson, 2005)
- plot specification at a high level of abstraction
- very flexible
- theme system for polishing plot appearance
- mature and complete graphics system
- many users, active community

When to not using ggplot2:

- 3-dimensional graphics (see the rgl package)
- Graph-theory type graphs (see the igraph package)
- Interactive graphics (see the ggvis package)

BIG DATA EXPERIENCE

g·able

What Is The Grammar Of Graphics?

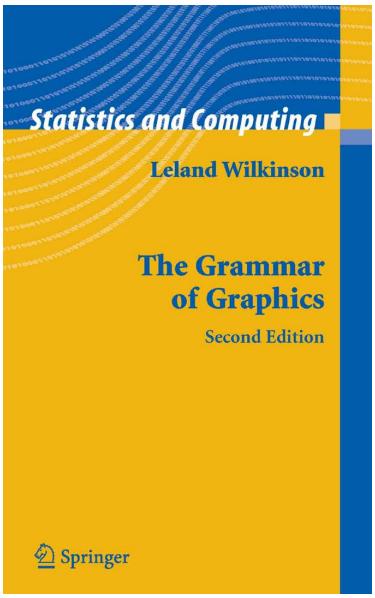
- The basic idea: independently specify plot building blocks and combine them to create just about any kind of graphical display you want.
- Building blocks of a graph include:
  - data
  - aesthetic mapping
  - geometric object
  - statistical transformations
  - scales
  - coordinate system
  - position adjustments
  - faceting

BIG DATA EXPERIENCE

g·able

Origin of ggplot2

- Foundation for producing almost every quantitative graphic found
- Designed for a distributed computing environment
- Make graphics easier



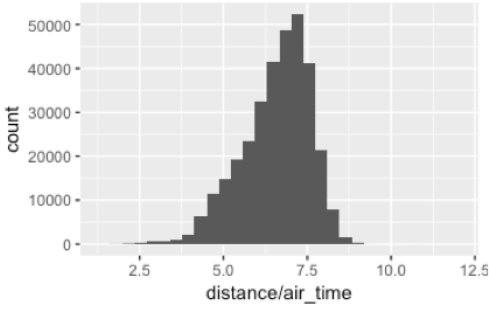
BIG DATA EXPERIENCE

g·able

Data + Mapping + Geometrics

```
library(ggplot2)

ggplot(data = flightData,
       mapping = aes(x = distance/air_time)) +
  geom_histogram()
```



Geometric Objects And Aesthetics Aesthetic Mapping

- In ggplot land, aesthetic means "something you can see".
- Examples include:
  - position (i.e., on the x and y axes)
  - color ("outside" color)
  - fill ("inside" color)
  - shape (of points)
  - linetype
  - size
- Each type of geom accepts only a subset of all aesthetics.
- Aesthetic mappings are set with the aes() function.

Geometic Objects (geom_)

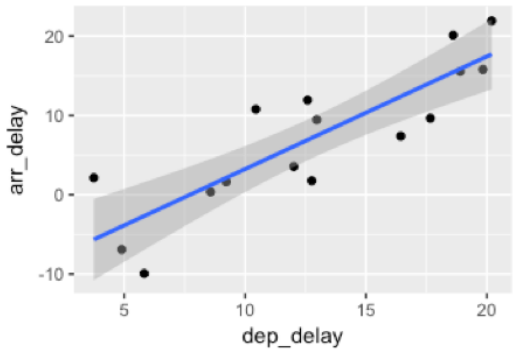
- Geometric objects are the actual marks we put on a plot. Examples include:
  - points (geom_point, for scatter plots, dot plots, etc)
  - lines (geom_line, for time series, trend lines, etc)
  - boxplot (geom_boxplot, for, well, boxplots!)
- A plot must have at least one geom; there is no upper limit.
- You can add a geom to a plot using the + operator

BIG DATA EXPERIENCE

g·able

Adding more than 1 geometric objects

```
ggplot(data = data2,
       mapping = aes(x = dep_delay,
                     y = arr_delay)) +
  geom_point() +
  geom_smooth(method = "lm")
```

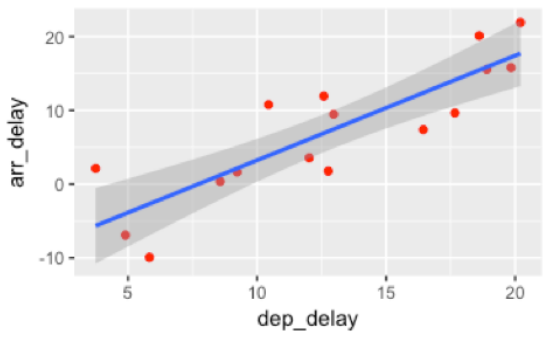


BIG DATA EXPERIENCE

g·able

Set aesthetic parameter directly

```
ggplot(data = data2,
       mapping = aes(x = dep_delay,
                     y = arr_delay)) +
  geom_point(color = "red") +
  geom_smooth(method = "lm")
```



BIG DATA EXPERIENCE

Statistical Transformations

- Some plots, such as boxplots, histograms, prediction lines etc. require statistical transformations:
  - for a boxplot the y values must be transformed to the median and 1.5(IQR)
  - for a smoother smother the y values must be transformed into predicted values
- Each geom has a default statistic, but these can be changed. For example, the default statistic for geom_bar is stat_count:

```
args(geom_histogram)
args(stat_bin)
```

g·able

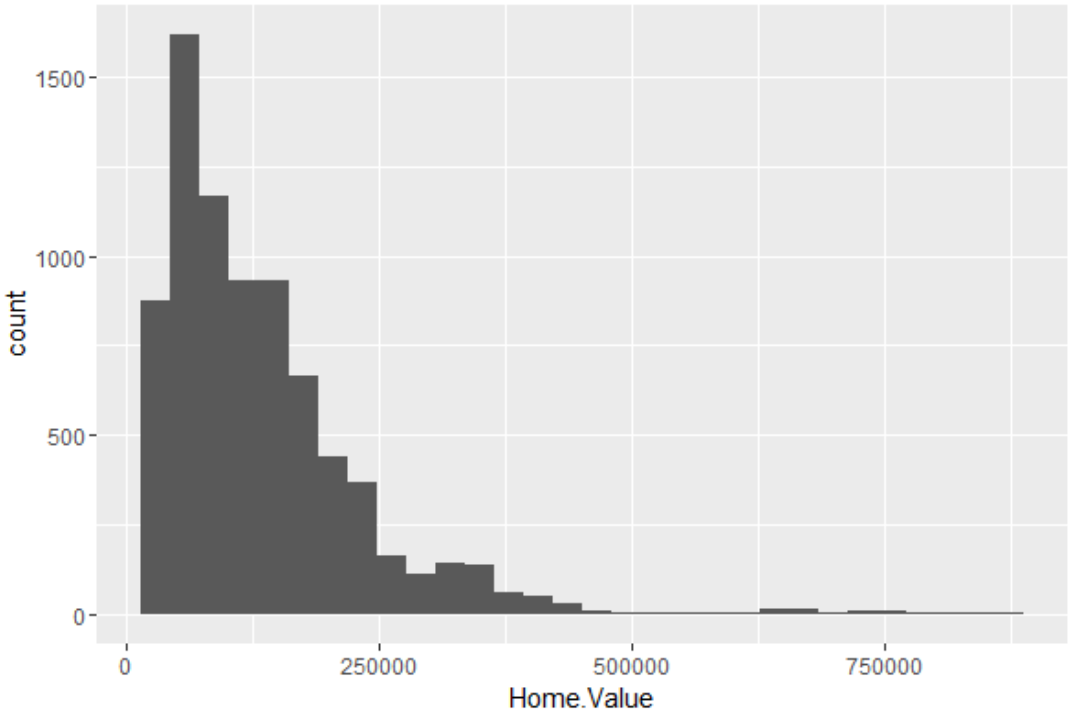
BIG DATA EXPERIENCE

Setting Statistical Transformation Arguments

- Arguments to stat_ functions can be passed through geom_ functions.
- This can be slightly annoying because in order to change it you have to first determine which stat the geom uses, then determine the arguments to that stat.
- For example, here is the default histogram of Home.Value:

```
p2 <- ggplot(housing, aes(x = Home.Value))
p2 + geom_histogram()
```

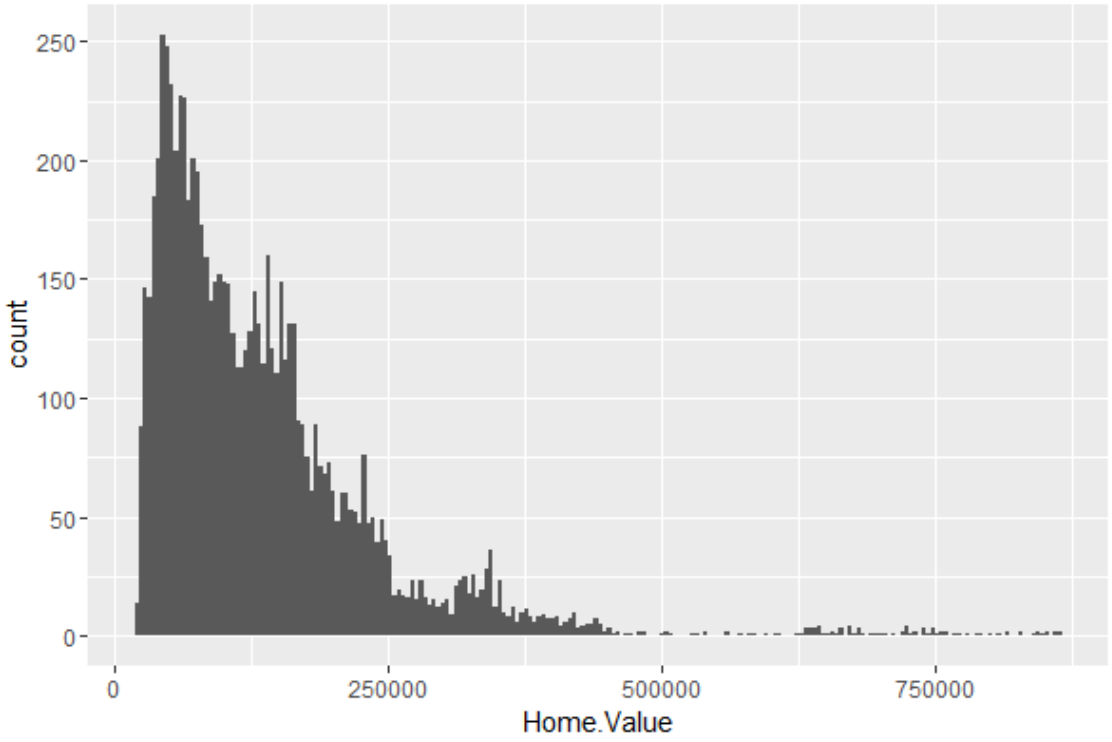
g·able



Changing binwidth

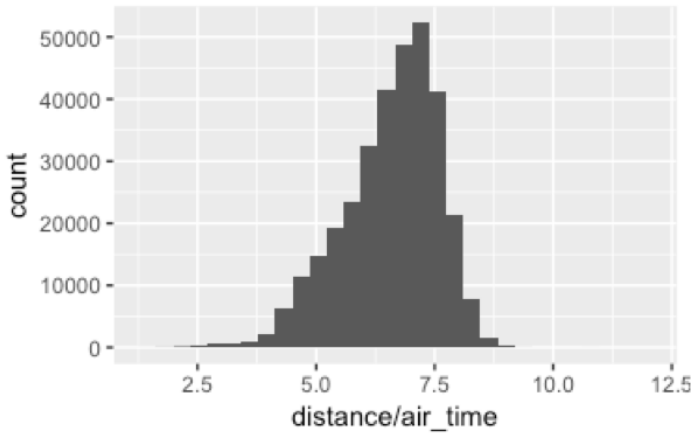
- we can change it by passing the binwidth argument to the stat_bin function:

```
p2 + geom_histogram(stat = "bin", binwidth=4000)
```



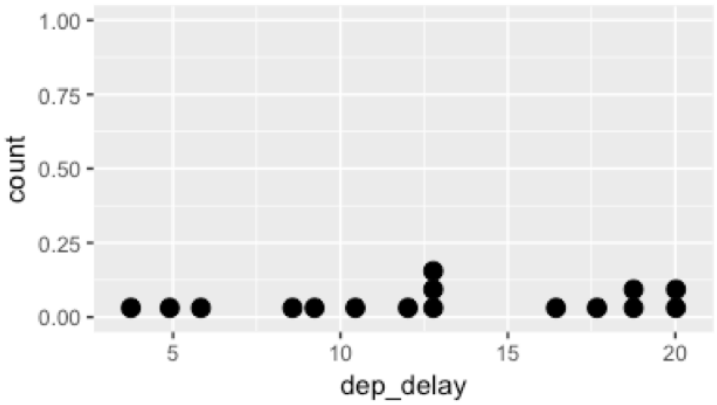
Univariate Histogram

```
ggplot(data = flightData,
       mapping = aes(x = distance/air_time)) +
  geom_histogram()
```



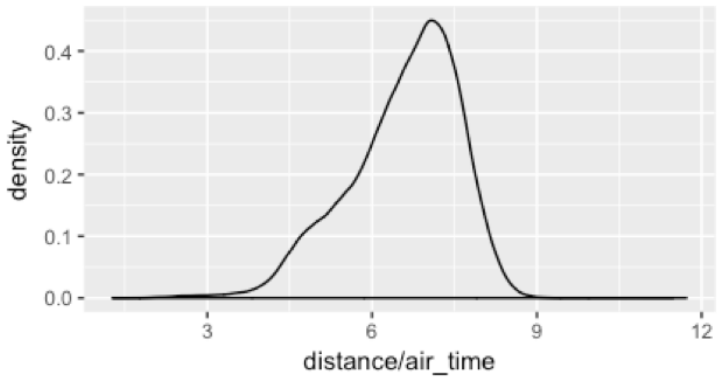
Univariate Dotplot

```
ggplot(data2, aes(x = dep_delay)) +
  geom_dotplot()
```



Univariate Density plot

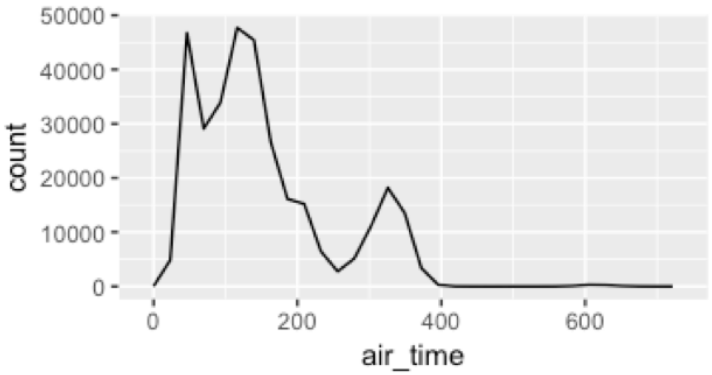
```
ggplot(flightData, aes(x = distance/air_time)) +
  geom_density()
```



BIG DATA EXPERIENCE

Univariate Frequency Polygon

```
ggplot(flightData, aes(x = air_time)) +
   geom_freqpoly()
```

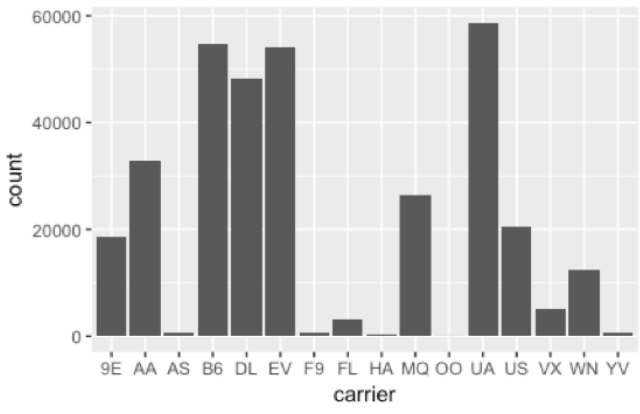


g·able

BIG DATA EXPERIENCE

Univariate (discrete) Bar

```
ggplot(flightData, aes(x = carrier)) +
   geom_bar()
```



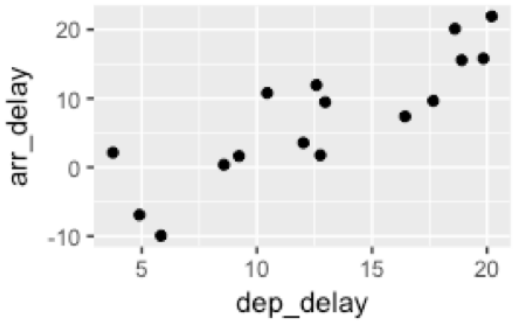
g·able

BIG DATA EXPERIENCE

g·able

Bivariate (Continuous X, Continuous Y) Scatter

```
ggplot(data = data2,
       mapping = aes(x = dep_delay,
                     y = arr_delay)) +
  geom_point()
```

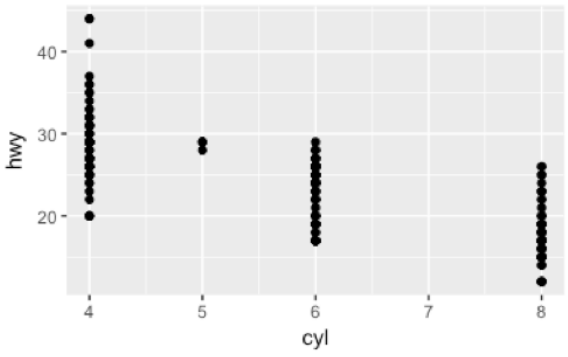


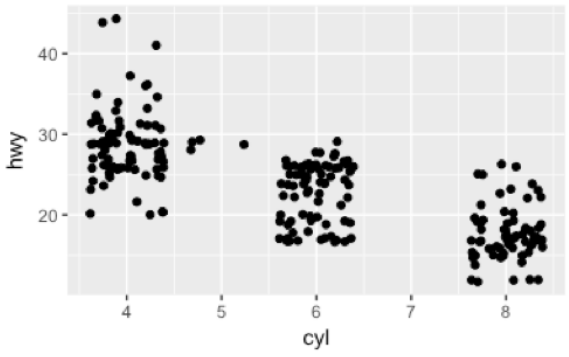
BIG DATA EXPERIENCE

g·able

Bivariate (Continuous X, Continuous Y) Jitter (repeated points)

```
p <- ggplot(mpg, aes(cyl, hwy))
p + geom_point()
p + geom_jitter()
```



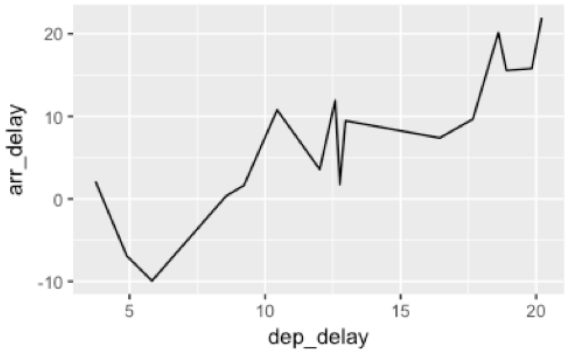


BIG DATA EXPERIENCE

g·able

Bivariate (Continuous X, Continuous Y) Line

```
ggplot(data = data2,
       mapping = aes(x = dep_delay,
                     y = arr_delay)) +
  geom_line()
```

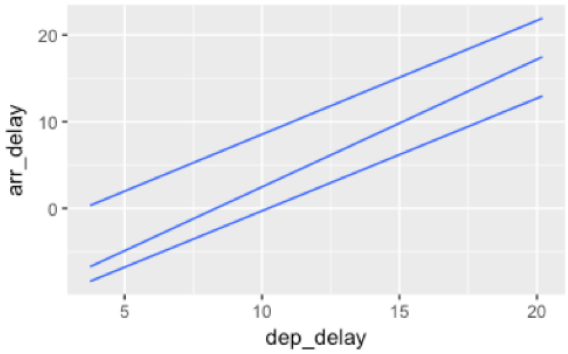


BIG DATA EXPERIENCE

g·able

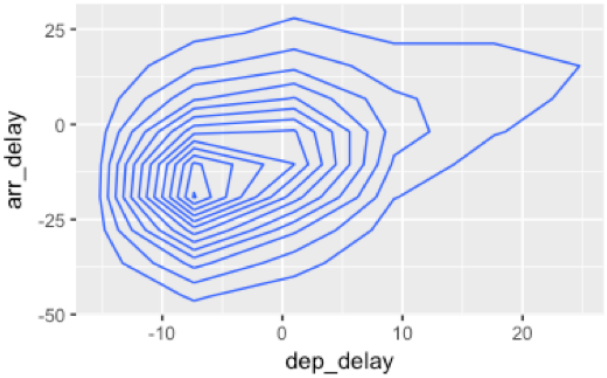
Bivariate (Continuous X, Continuous Y) Quantile

```
ggplot(data = data2,
       mapping = aes(x = dep_delay,
                     y = arr_delay)) +
geom_quantile()
```



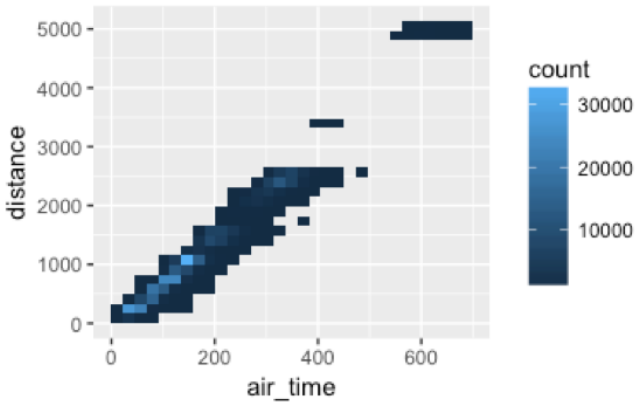
Bivariate (Continuous X, Continuous Y) Contour - 2D Distribution

```
ggplot(data = sample_frac(flightData,0.1),
       mapping = aes(x = dep_delay,
                     y = arr_delay)) +
  geom_density_2d()
```



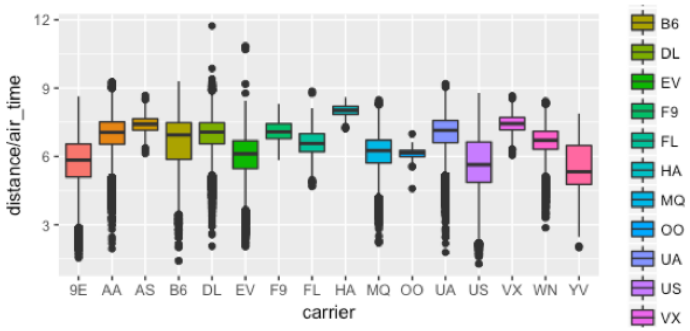
Bivariate (Continuous X, Continuous Y) Density - 2D Distribution

```
ggplot(data = flightData,
       mapping = aes(x = air_time,
                     y = distance)) +
  geom_bin2d()
```



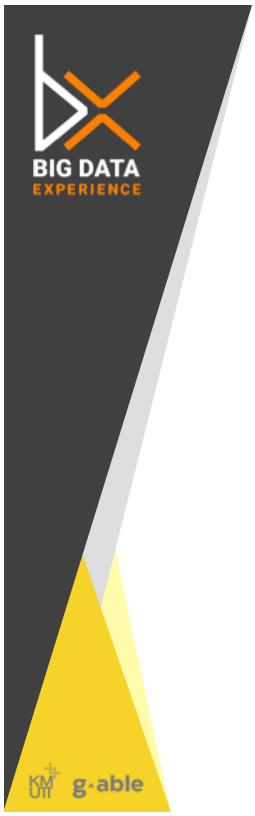
Bivariate (Discrete X, Continuous Y) Boxplot

```
ggplot(data = flightData,
       mapping = aes(x = carrier,
                     y = distance/air_time,
                     fill = carrier)) +
  geom_boxplot()
```



Activity 2

- Use the variable “data2”
- Plot the bar chart using geom_bar
  - x: carrier, y: mean departure delay
- Change the stat to “identity”
- Color the bar with different carrier



Thank you

Question?