



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



Module 3 Overview

- Data wrangling with R
- Data visualization with R





BIG DATA
EXPERIENCE

Data Wrangling

Part 1



BIG DATA
EXPERIENCE

Wrangling
Munging
Janitor Work
Manipulation
Transformation

50-80%
of your time?



Two goals

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



Required Packages

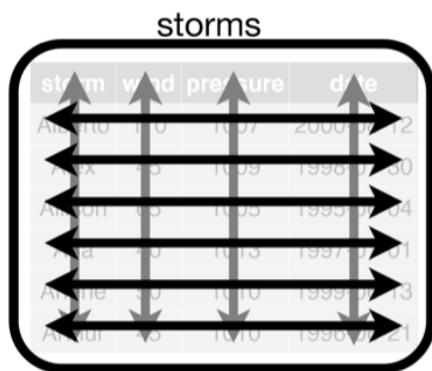
Data Wrangling

- tidyverse
- dplyr

Data Visualization

- ggplot2

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).

Load data

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

```
load("2-DataPrep.RData")
```

storms				cases			pollution			
storm	wind	pressure	date	Country	2011	2012	2013	city	particle size	amount ($\mu\text{g}/\text{m}^3$)
Alberto	110	1007	2000-08-12	FR	7000	6900	7000	New York	large	23
Alex	45	1009	1998-07-30	DE	5800	6000	6200	New York	small	14
Allison	65	1005	1995-06-04	US	15000	14000	13000	London	large	22
Ana	40	1013	1997-07-01					London	small	16
Arlene	50	1010	1999-06-13					Beijing	large	121
Arthur	45	1010	1996-06-21					Beijing	small	56



Tidy data



BIG DATA
EXPERIENCE



Data Wrangling with dplyr and tidyverse

Cheat Sheet

R Studio

Syntax - Helpful conventions for wrangling

dplyr: tbl_df(iris)
Converts data totbl class.tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <dbl> <dbl> <dbl> <dbl> <fct>
  1 5.1 3.5 1.4 0.2 setosa
  2 4.9 3.0 1.4 0.2 setosa
  3 4.7 3.2 1.3 0.2 setosa
  4 4.6 3.1 1.5 0.2 setosa
  5 4.8 3.4 1.6 0.2 setosa
  ... ... ... ... ...
  Variables not shown: Petal.Width (dbl), Species (fctr)
```

dplyr: glimpse(iris)
Information dense summary oftbl data.

utils: View(iris)
View data set in spreadsheet-like display (note capital V).

dplyr: %>%
Passes object on left hand side as first argument (or . argument) of function on right hand side.

```
x %>% f(y) is the same as f(x, y)
y %>% f(x, .. z) is the same as f(x, y, z)
```

"Piping" with %>% makes code more readable, e.g.

```
iris %>
  group_by(Species) %>
  summarise(avg = mean(Sepal.Width)) %>
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

In a tidy data set:

Each variable is saved in its own column & Each observation is saved in its own row

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

M * A

Reshaping Data - Change the layout of a data set

tidy: gather(cases, "year", "n", 2:4)
Gather columns into rows.

tidy: spread(pollution, size, amount)
Spread rows into columns.

tidy: separate(storms, date, c("y", "m", "d"))
Separate one column into several.

tidy: unite(data, col, ..., sep)
Unite several columns into one.

Subset Observations (Rows)

filter(iris, Sepal.Length > 7)
Extract rows that meet logical criteria.

dplyr: distinct(iris)
Remove duplicate rows.

dplyr: sample_n(iris, 0.5, replace = TRUE)
Randomly select fraction of rows.

dplyr: sample_n(iris, 10, replace = TRUE)
Randomly select n rows.

dplyr: slice(iris, 10:15)
Select rows by position.

dplyr: top_n(storms, 2, date)
Select and order top n entries (by group if grouped data).

Subset Variables (Columns)

dplyr: select(iris, Sepal.Width, Petal.Length, Species)
Select columns by name or helper function.

Helper functions for select - ?select

- selections, contains()**: Select columns whose name contains a character string.
- select(iris, ends_with("Length"))**: Select columns whose name ends with a character string.
- select(iris, everything())**: Select every column.
- select(iris, matches("x"))**: Select columns whose name matches a regular expression.
- select(iris, num_range("v", "z"))**: Select columns named x1, x2, x3, x4, x5.
- select(iris, one_of("Species", "Genus"))**: Select columns whose names are in a group of names.
- select(iris, starts_with("Sepa"))**: Select columns whose name starts with a character string.
- select(iris, Select_if(is.na, is.na))**: Select all columns between Sepal.Length and Petal.Width (inclusive).
- select(iris, -Species)**: Select all columns except Species.

Learn more with <http://www.rstudio.com/resources/cheatsheets/>

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



BIG DATA
EXPERIENCE



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



Load library

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

KM g·able



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)
```

KM g·able



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)`



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



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

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()**



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)`



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





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

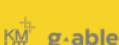


Useful select functions

* 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

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

filter()



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

storms

filter()



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"))

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

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)
```

Useful mutate functions

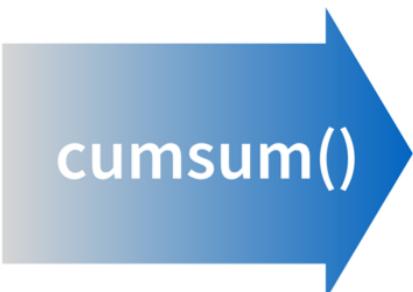
- * All take a vector of values and return a vector of values
- ** Blue functions come in dplyr

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

"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()	



1	1
2	3
3	6
4	10
5	15
6	21



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)



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)



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))`

The pipe operator %>%

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



tb %>% select(_____, child:elderly)



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)`



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)
```



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)
```





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)
```



summarise()

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
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))
```





summarise()

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
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())
```



Useful summary functions

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

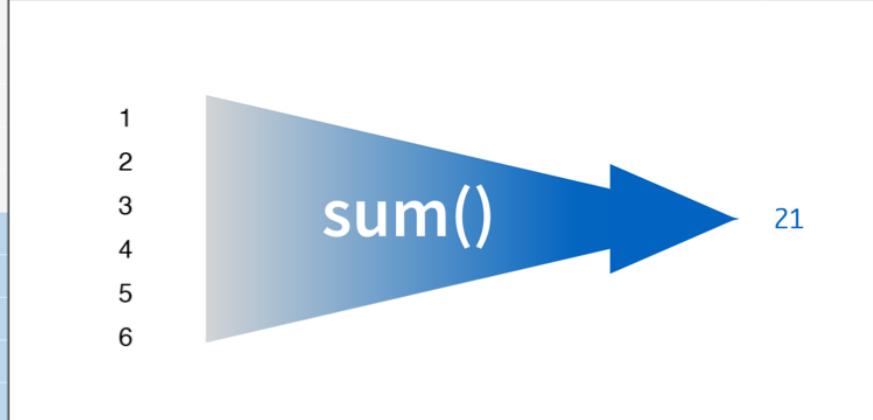
min(), max()	Minimum and maximum values
mean()	Mean value
median()	Median value
sum()	Sum of values
var, sd()	Variance and standard deviation of a vector
first()	First value in a vector
last()	Last value in a vector
nth()	Nth value in a vector
n()	The number of values in a vector
n_distinct()	The number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



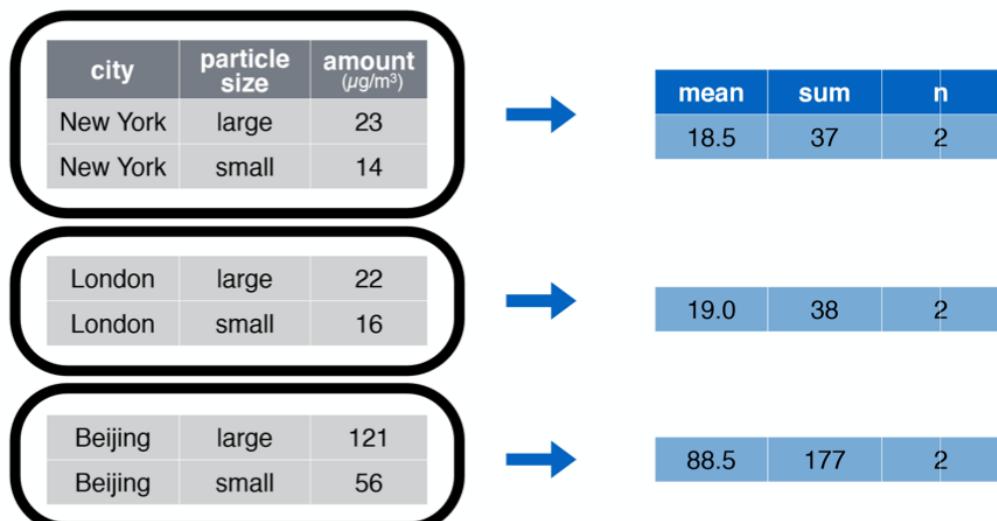
"Summary" functions

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

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



21



The diagram illustrates three examples of summarizing data using "group_by()" and "summarise()".

- New York:** The input data frame has rows for "New York" with "particle size" "large" and "amount" 23, and "New York" with "particle size" "small" and "amount" 14. The output summary shows a mean of 18.5, a sum of 37, and an n of 2.
- London:** The input data frame has rows for "London" with "particle size" "large" and "amount" 22, and "London" with "particle size" "small" and "amount" 16. The output summary shows a mean of 19.0, a sum of 38, and an n of 2.
- Beijing:** The input data frame has rows for "Beijing" with "particle size" "large" and "amount" 121, and "Beijing" with "particle size" "small" and "amount" 56. The output summary shows a mean of 88.5, a sum of 177, and an n of 2.

`group_by() + summarise()`



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



dplyr::bind_cols()

The diagram shows the result of binding two data frames, *y* and *z*, side-by-side using the `bind_cols` function.

Table *y*:

x1	x2
A	1
B	2
C	3

Table *z*:

x1	x2
B	2
C	3
D	4

The resulting table after binding:

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

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





dplyr::bind_rows()

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

+ = bind_rows(y, z)



dplyr::left_join()

songs		artists		song	name	plays
song	name	name	plays	Across the Universe	John	guitar
Across the Universe	John	George	sitar	Come Together	John	guitar
Come Together	John	John	guitar	Hello, Goodbye	Paul	bass
Hello, Goodbye	Paul	Paul	bass	Peggy Sue	Buddy	<NA>
Peggy Sue	Buddy	Ringo	drums			

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





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"))`



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

semi_join()

songs		artists			
song	name	name	plays	song	name
Across the Universe	John	George	sitar	Across the Universe	John
Come Together	John	John	guitar	Come Together	John
Hello, Goodbye	Paul	Paul	bass	Hello, Goodbye	Paul
Peggy Sue	Buddy	Ringo	drums		

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



g·able

anti_join()

songs		artists			
song	name	name	plays	song	name
Across the Universe	John	George	sitar	Peggy Sue	Buddy
Come Together	John	John	guitar		
Hello, Goodbye	Paul	Paul	bass		
Peggy Sue	Buddy	Ringo	drums		

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



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
```



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





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)



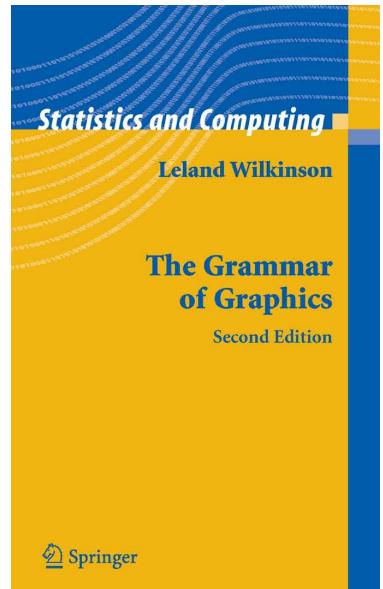
What Is The Grammar Of Graphics?

- 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



Origin of ggplot2

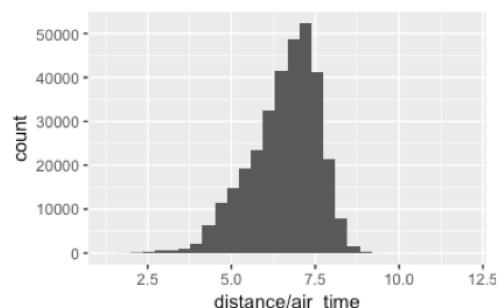
- Foundation for producing almost every quantitative graphic found
- Designed for a distributed computing environment
- Make graphics easier



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.



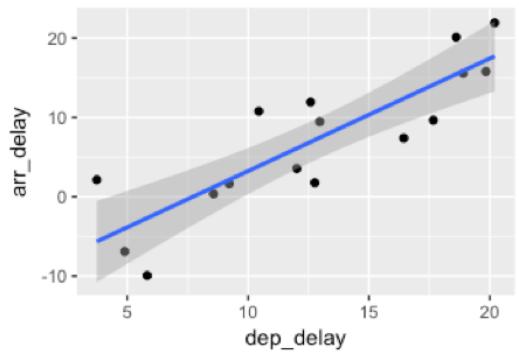
Geometric Objects (geom_)

- 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



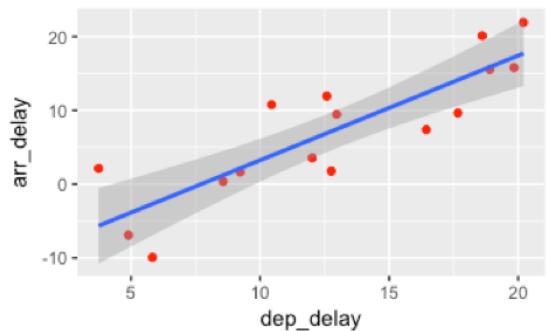
Adding more than 1 geometric objects

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



Set aesthetic parameter directly

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





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`:

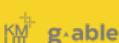
```
args(geom_histogram)  
args(stat_bin)
```



Setting Statistical Transformation Arguments

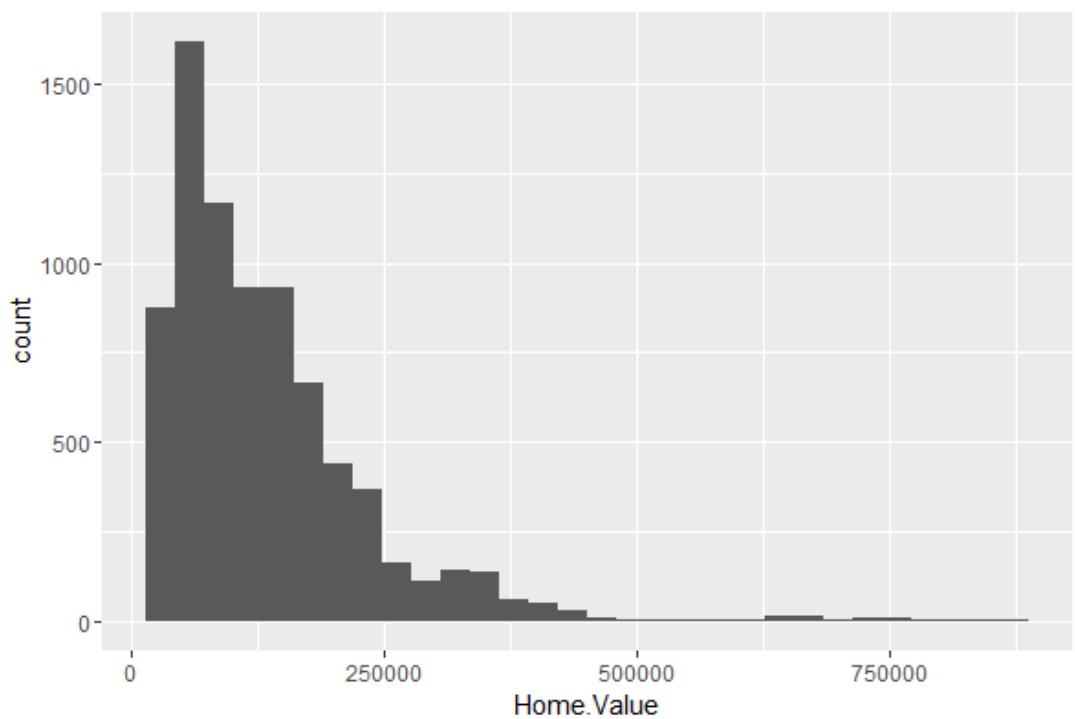
- Arguments to `stat_` functions can be passed through `geom_` functions.
- This can be slightly annoying because in order to change it you have to first determine which stat the geom uses, then determine the arguments to that stat.
- For example, here is the default histogram of Home.Value:

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





KM g·able



KM g·able

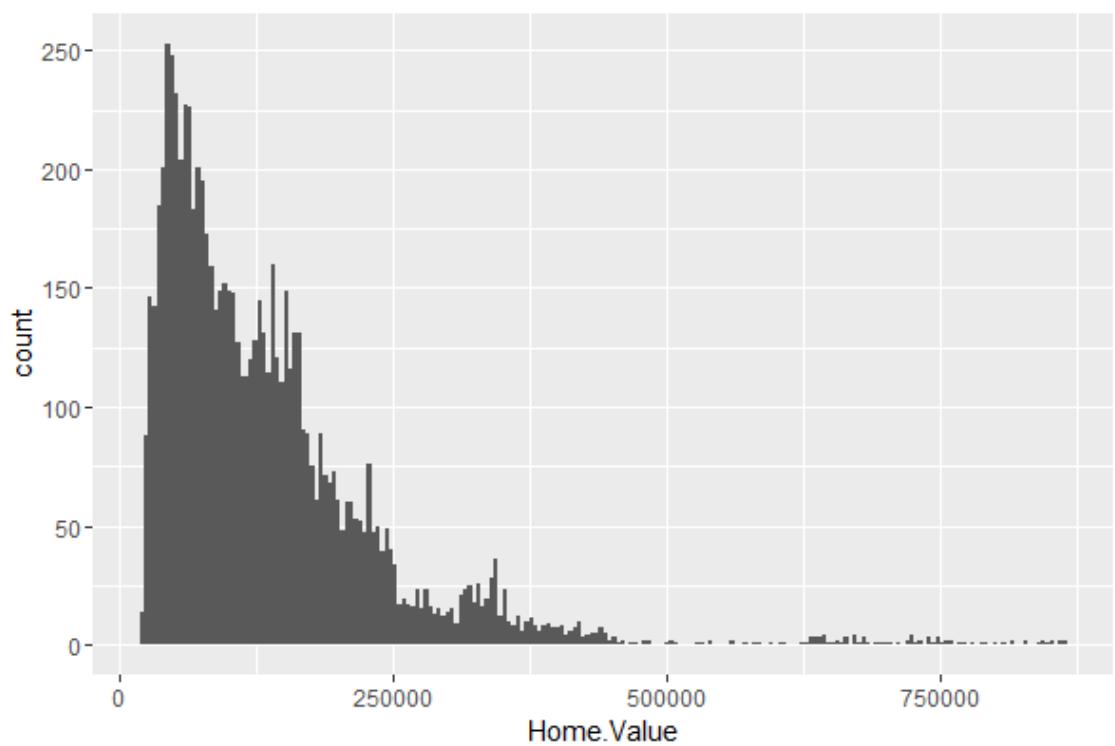
Changing binwidth

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

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



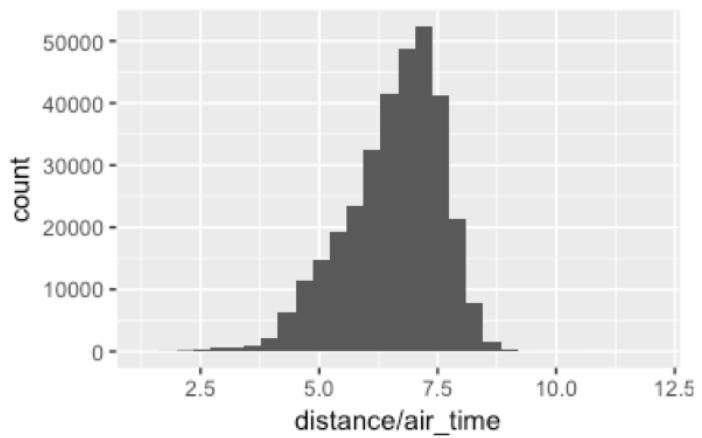
g·able



g·able

Univariate Histogram

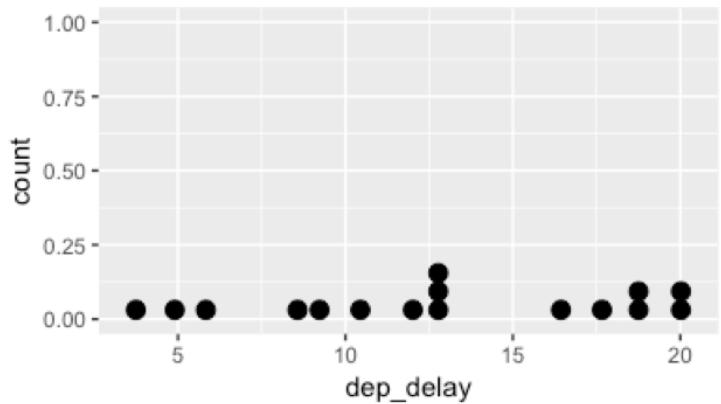
```
ggplot(data = flightData,  
       mapping = aes(x = distance/air_time)) +  
       geom_histogram()
```





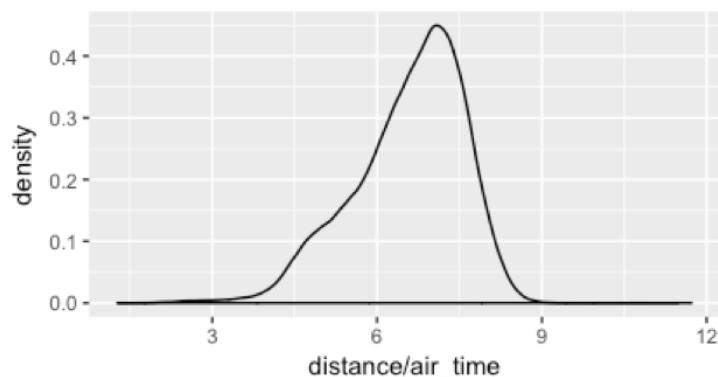
Univariate Dotplot

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



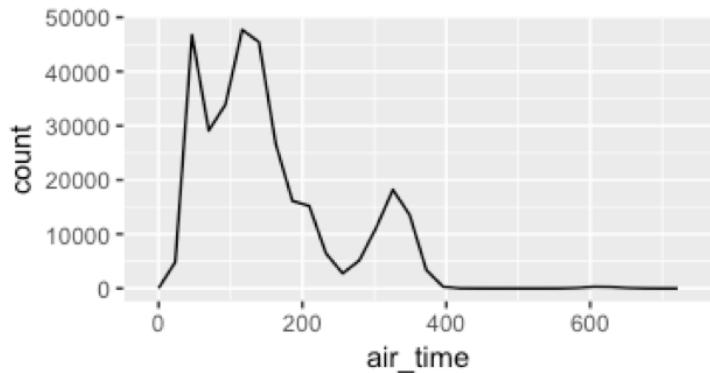
Univariate Density plot

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



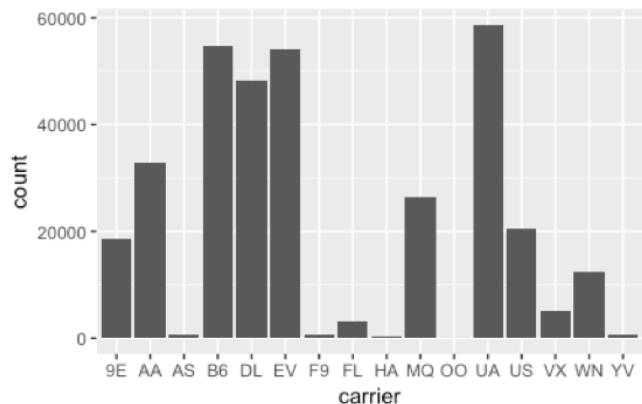
Univariate Frequency Polygon

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



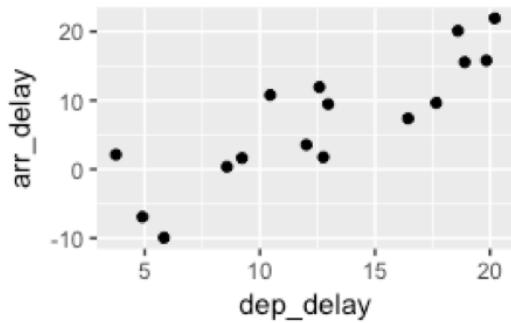
Univariate (discrete) Bar

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



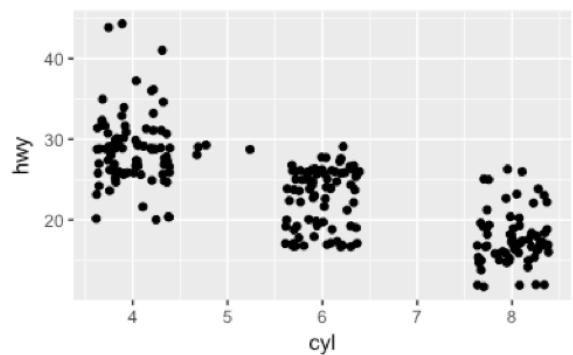
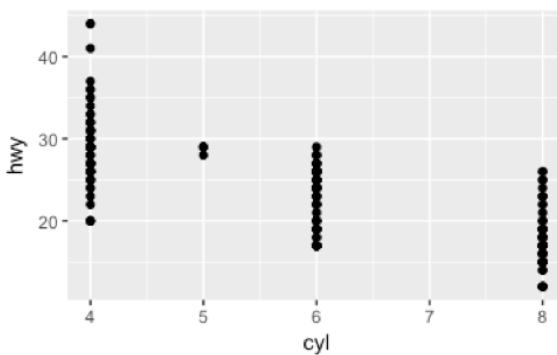
Bivariate (Continuous X, Continuous Y) Scatter

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



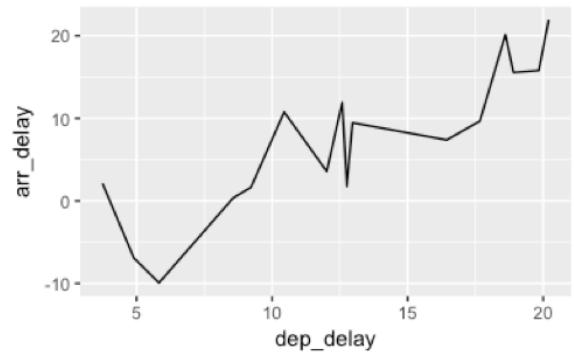
Bivariate (Continuous X, Continuous Y) Jitter (repeated points)

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



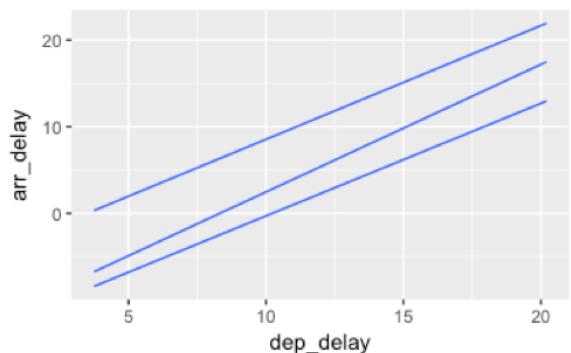
Bivariate (Continuous X, Continuous Y) Line

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



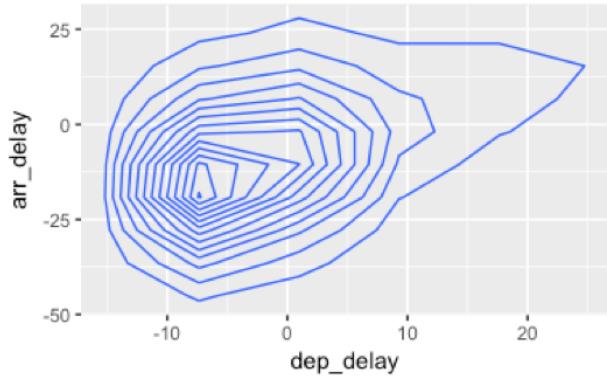
Bivariate (Continuous X, Continuous Y) Quantile

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



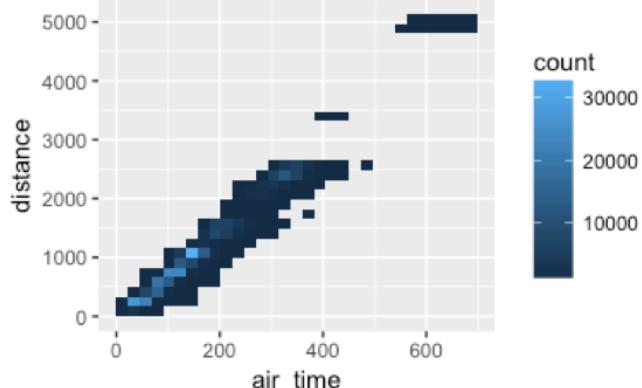
Bivariate (Continuous X, Continuous Y) Contour - 2D Distribution

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



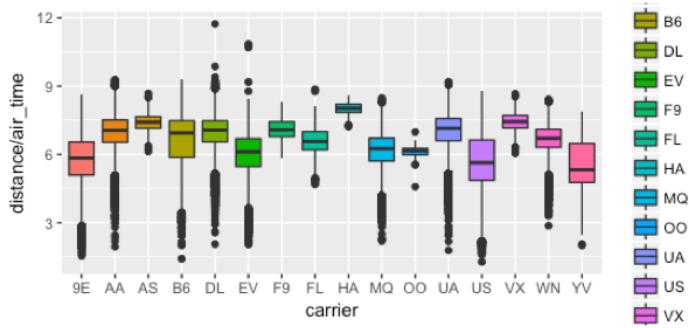
Bivariate (Continuous X, Continuous Y) Density - 2D Distribution

```
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?

g·able