Welcome to DATA 151

I'm so glad you're here!

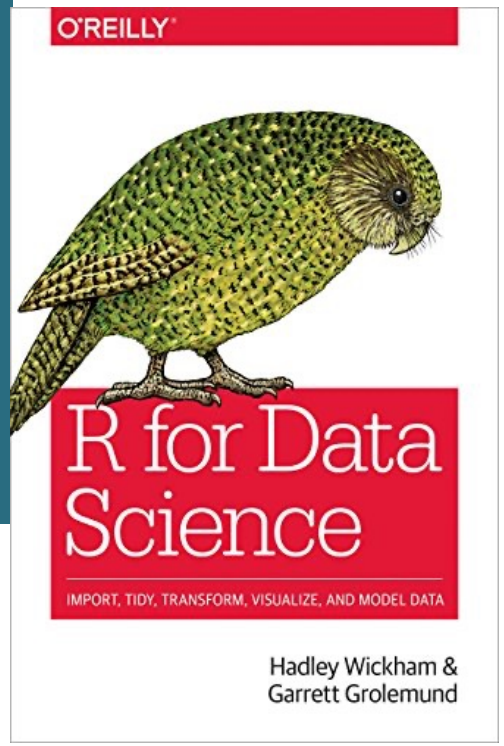
DATA 151: CLASS 6A
INTRODUCTION TO DATA SCIENCE (WITH R)

TIDY DATA AND GRAPHICS

NOTES PREPARED BY PROF. KITADA SMALLEY (FALL 2022)

ANNOUNCEMENTS

RELEVANT READING



Introduction to Data Science:

- Tuesday:
  - Introduction to Data Science
  - Ch 7: Introduction to data viz
  - Ch 8: ggplot2
- Thursday:
  - R for Data Science
  - Ch 7: Exploratory Data Analysis

HOMEWORK REMINDER

Due this week: (DUE 10/6)

- *HW #5: DC Importing Data in R*
  - *Only one chapter (not the whole course)*
  - ***No submission on WISE necessary, do on DataCamp***
- *Project Milestone #2: Project Meetings*
  - Each group must have their data set approved by class on Thursday

HOMEWORK REMINDER

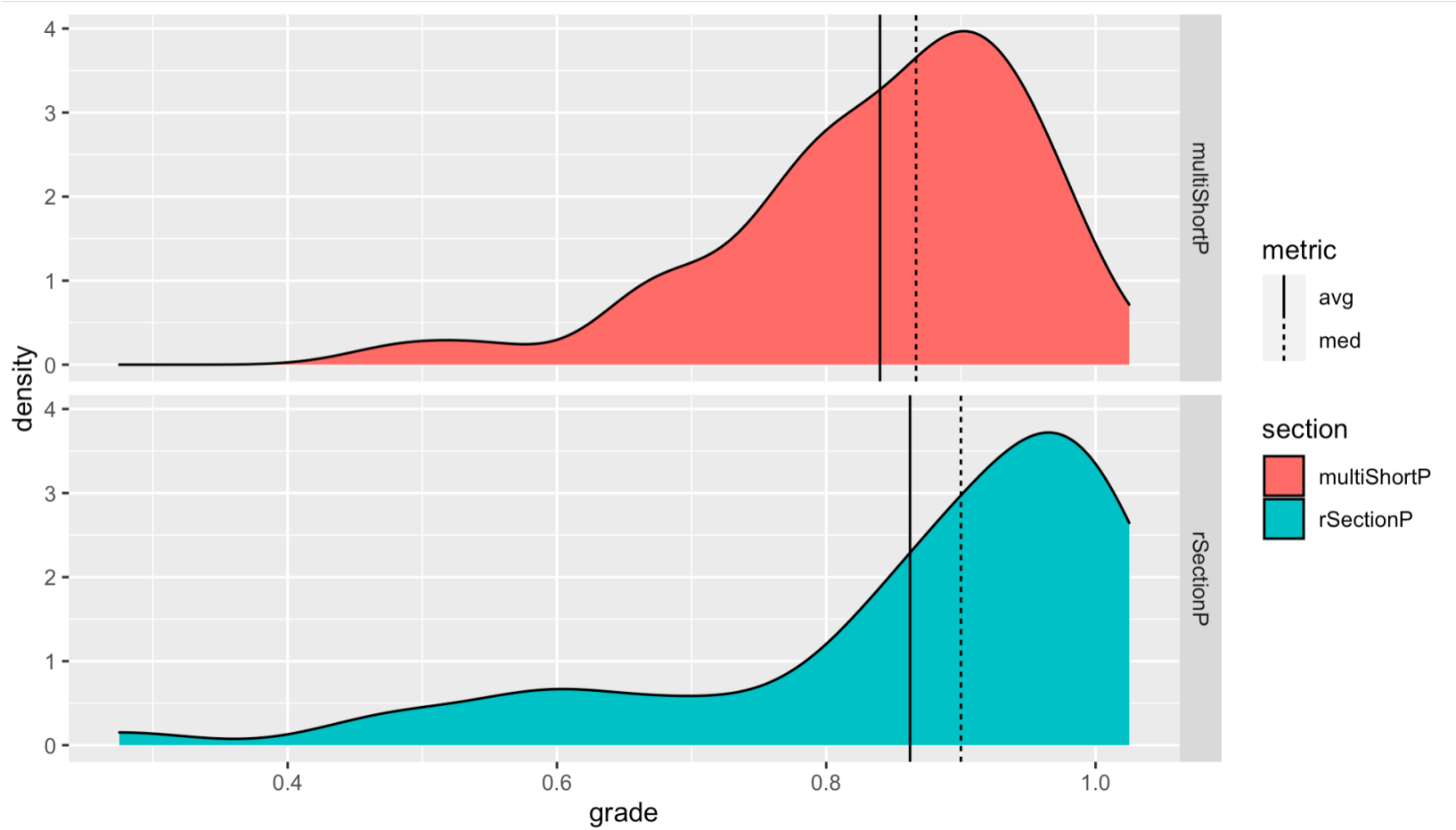
Due next week: (DUE 10/13)

- *HW #6: DC Introduction to Data Visualization in ggplot2*
  - ***No submission on WISE necessary, do on DataCamp***
- *Project Milestone #3: EDA Step 1*
  - Ask questions and form hypotheses

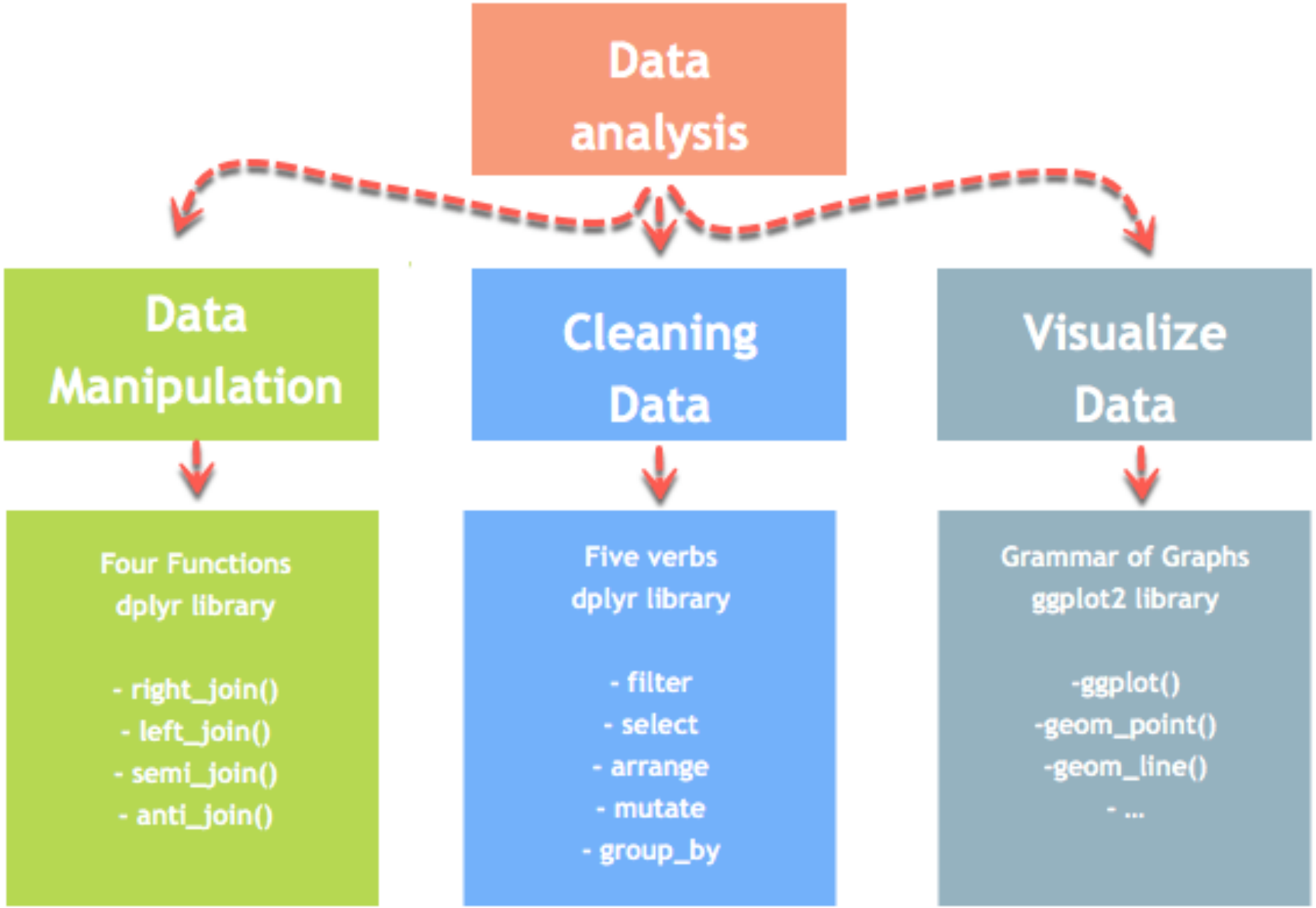
MIDTERM SCORES

Multiple Choice and Short Answer
avg = 0.8399
med = 0.8667

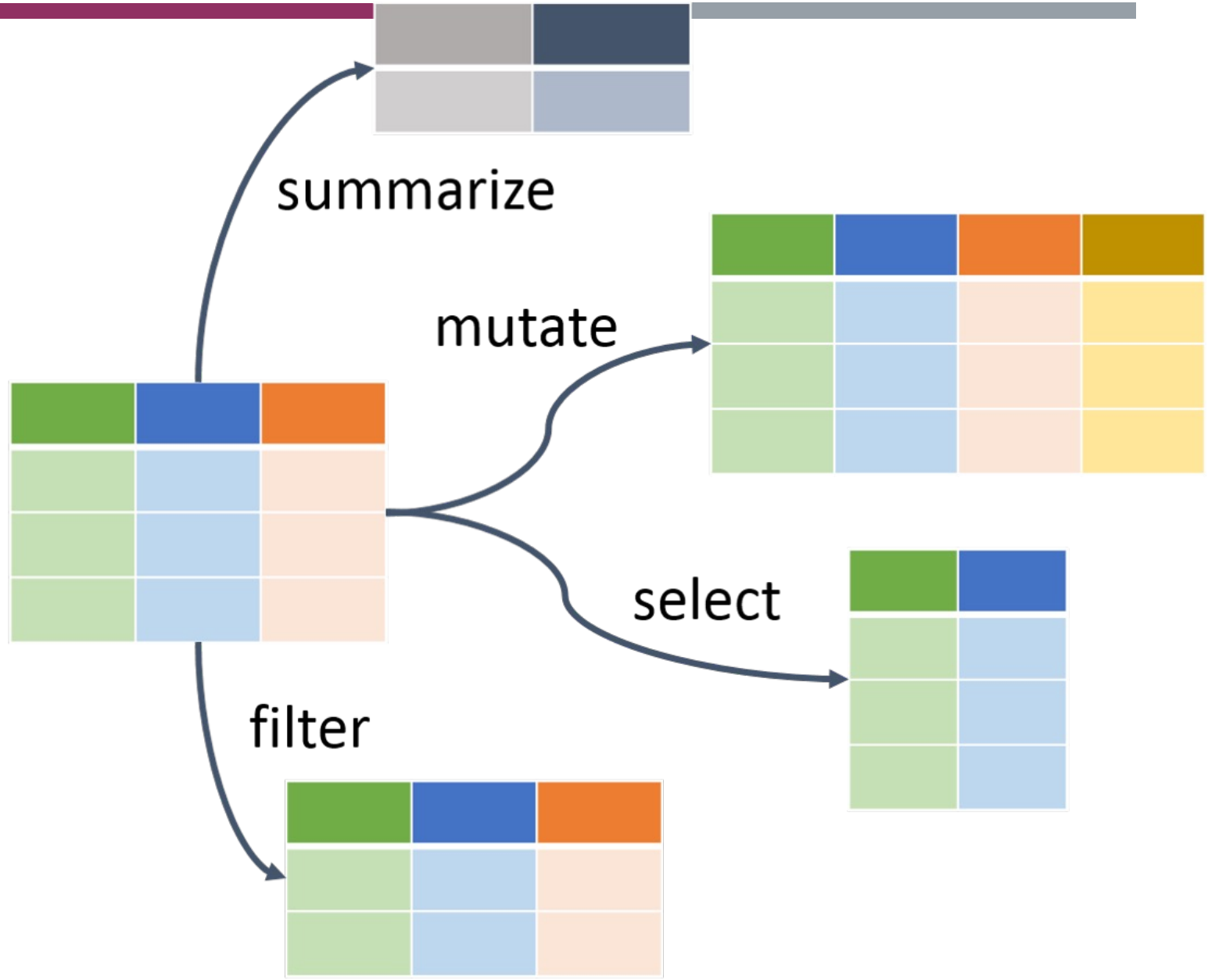
R Section
avg = 0.8622
med = 0.9000



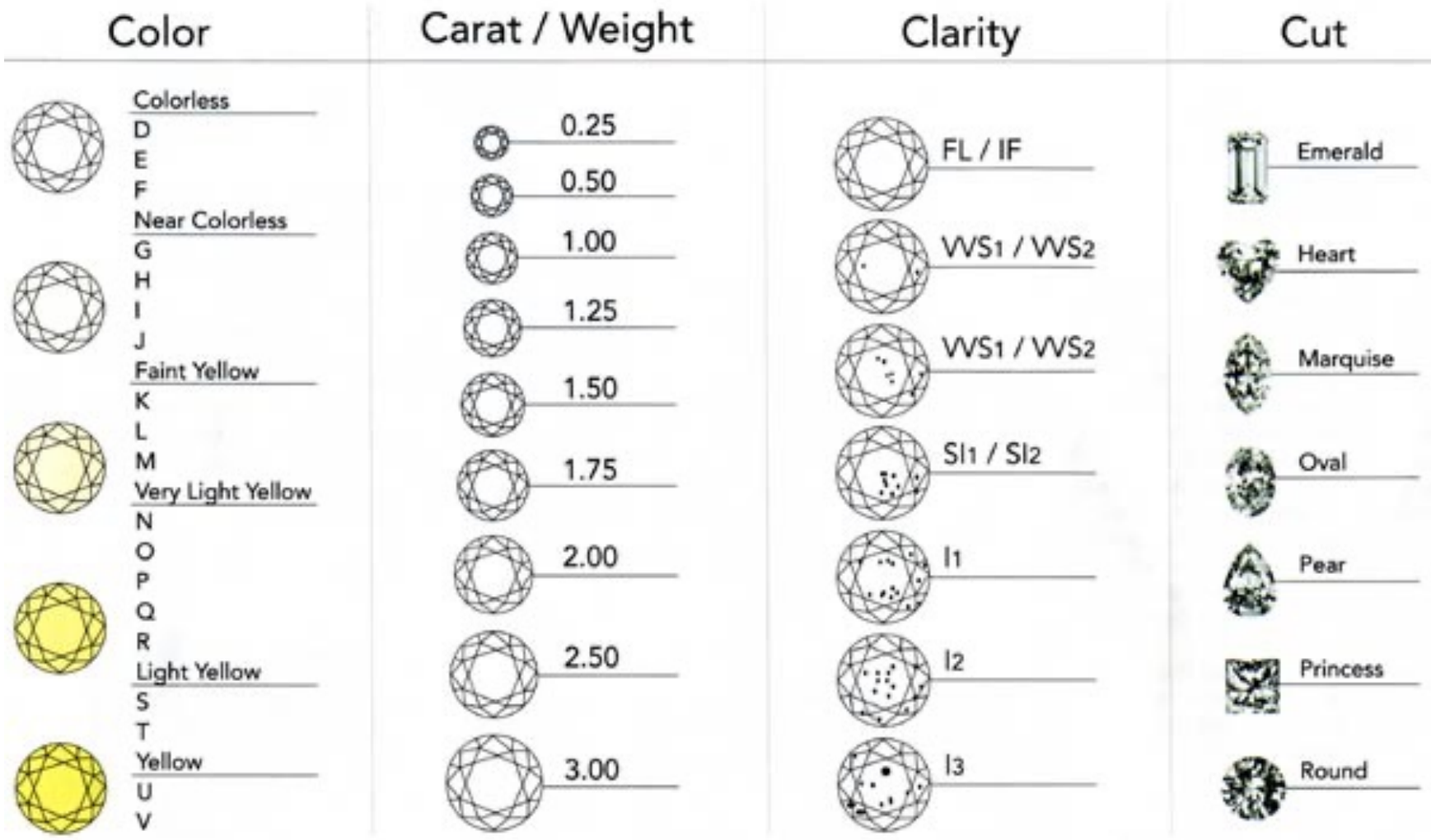
WARM-UP / REVIEW


Data analysis
Data Manipulation
Cleaning Data
Visualize Data
Four Functions
dplyr library
- right_join()
- left_join()
- semi_join()
- anti_join()
Five verbs
dplyr library
- filter
- select
- arrange
- mutate
- group_by
Grammar of Graphs
ggplot2 library
-ggplot()
-geom_point()
-geom_line()
- ...


dplyr
www.rstudio.com


summarize
mutate
select
filter

DIAMONDS: THE 4 C'S



DIAMONDS

diamonds {ggplot2} R Documentation

Prices of over 50,000 round cut diamonds

Description

A dataset containing the prices and other attributes of almost 54,000 diamonds. The variables are as follows:

Usage

```
diamonds
```

Format

A data frame with 53940 rows and 10 variables:

price

price in US dollars (\$326–\$18,823)

carat

LOAD THE DATA AND CHECK IT OUT!

```
> data("diamonds")
> str(diamonds)
tibble [53,940 × 10] (S3: tbl_df/tbl/data.frame)
 $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
 $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

DIRECTIONS

Directions/Introduction: We have used the diamonds dataset contained in the tidyverse package for several visualization examples in class. Now we will use it to have some fun with data transformations and wrangling.

- You will practice your knowledge of the "verbs" filter, mutate, group_by, summarise, and the piping operator %>%

QUESTIONS

- **Question 1:** Make a new data set that has the average depth and price of the diamonds in the data set.
- **Question 2:** Add a new column to the data set that records each diamond's price per carat.
- **Question 3:** Create a new data set that groups diamonds by their cut and displays the average price of each group.

QUESTIONS

- **Question 4:** Create a new data set that groups diamonds by color and displays the average depth and average table for each group.

- **Question 5:** Which color diamonds seem to be largest on average (in terms of carats)?

- **Question 6:** What color of diamonds occurs the most frequently among diamonds with ideal cuts?

QUESTIONS

- **Question 7:** Which clarity of diamonds has the largest average table per carats?
- **Question 8:** What is the average price per carat of diamonds that cost more than $10,000?
- **Question 9:** Of the diamonds that cost more than $10,000 what is the most common clarity?

PRINCIPLES OF TIDY DATA


DOES IT SPARK JOY?
NETFLIX | NEW YEAR'S DAY
TIDYING UP
WITH MARIE KONDO

BEFORE

AFTER

PRINCIPLES OF TIDY DATA

- "Tidy" data needed to import data files (.csv, .txt, …) into statistical software packages such as R, SAS, Stata, SPSS, …

- **(Wickham 2014)**
  - 1) Each variable forms a column
  - 2) Each observation forms a row
  - 3) Each type of observation unit forms a table
    - (ie don't mix levels of aggregation or different observation types)

PRINCIPLES OF TIDY DATA

- **Examples of Messy Data:**
  - Column headers are values, not variable names
  - Multiple variables are stored in one column
  - Variables are stored in both rows and columns
  - Multiple types of observational units are stored in the same table
  - A single observational unit is stored in multiple tables

TOY EXAMPLE
STEP 1: GENERATE DATA

- Imagine that we have stock market data!

```
set.seed(1)
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 20, 1),
  Y = rnorm(10, 20, 2),
  Z = rnorm(10, 20, 4)
)
stocks
```

```
         time        X        Y        Z
1  2009-01-01 19.37355 23.02356 23.67591
2  2009-01-02 20.18364 20.77969 23.12855
3  2009-01-03 19.16437 18.75752 20.29826
4  2009-01-04 21.59528 15.57060 12.04259
5  2009-01-05 20.32951 22.24986 22.47930
6  2009-01-06 19.17953 19.91013 19.77549
7  2009-01-07 20.48743 19.96762 19.37682
8  2009-01-08 20.73832 21.88767 14.11699
9  2009-01-09 20.57578 21.64244 18.08740
10 2009-01-10 19.69461 21.18780 21.67177
```

STEP 2: GATHER

```
# gather
stocksG<-stocks%>%
  gather(key=stock, value=price, -time )

head(stocksG)
```

```
> head(stocksG)
        time stock    price
1 2009-01-01     X 19.37355
2 2009-01-02     X 20.18364
3 2009-01-03     X 19.16437
4 2009-01-04     X 21.59528
5 2009-01-05     X 20.32951
6 2009-01-06     X 19.17953
```

Why is it important to have data in this format?

GGPLOT2 HAS 7 LAYERS OF CONTROL!

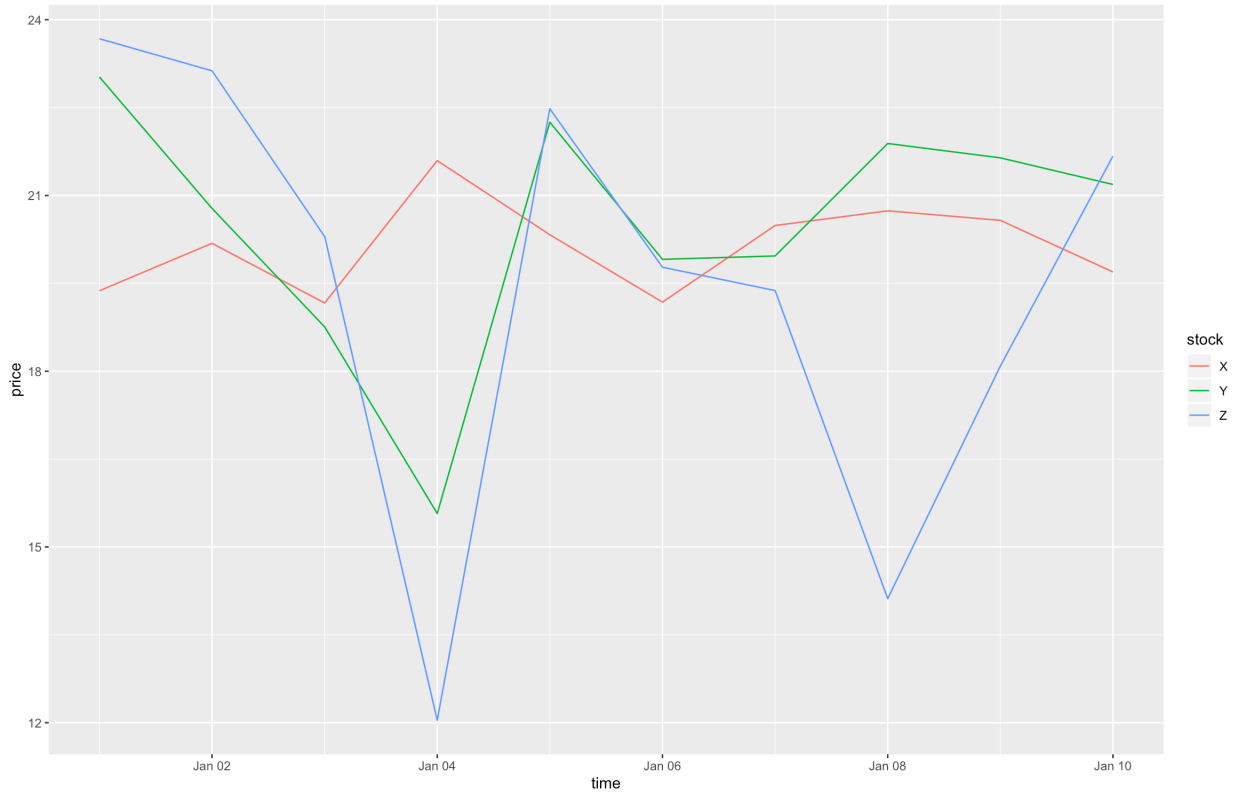
Element	Description
Data	The dataset being plotted.
Aesthetics	The scales onto which we *map* our data.
Geometries	The visual elements used for our data.
Facets	Plotting small multiples.
Statistics	Representations of our data to aid understanding.
Coordinates	The space on which the data will be plotted.
Themes	All non-data ink.

3 ESSENTIAL LAYERS

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

STEP 3:VISUALIZE

```
ggplot(stocksG, aes(time, price, color=stock))+
  geom_line()
```



TIDYR: GATHER

- We can combine tidyr and dplyr together into a pipeline

```
stocks %>%
   gather(stock,price,X:Z)%>%
   group_by(stock) %>%
   summarise(min = min(price), max = max(price))
```

```
  stock       min       max
1     X 19.16437 21.59528
2     Y 15.57060 23.02356
3     Z 12.04259 23.67591
```

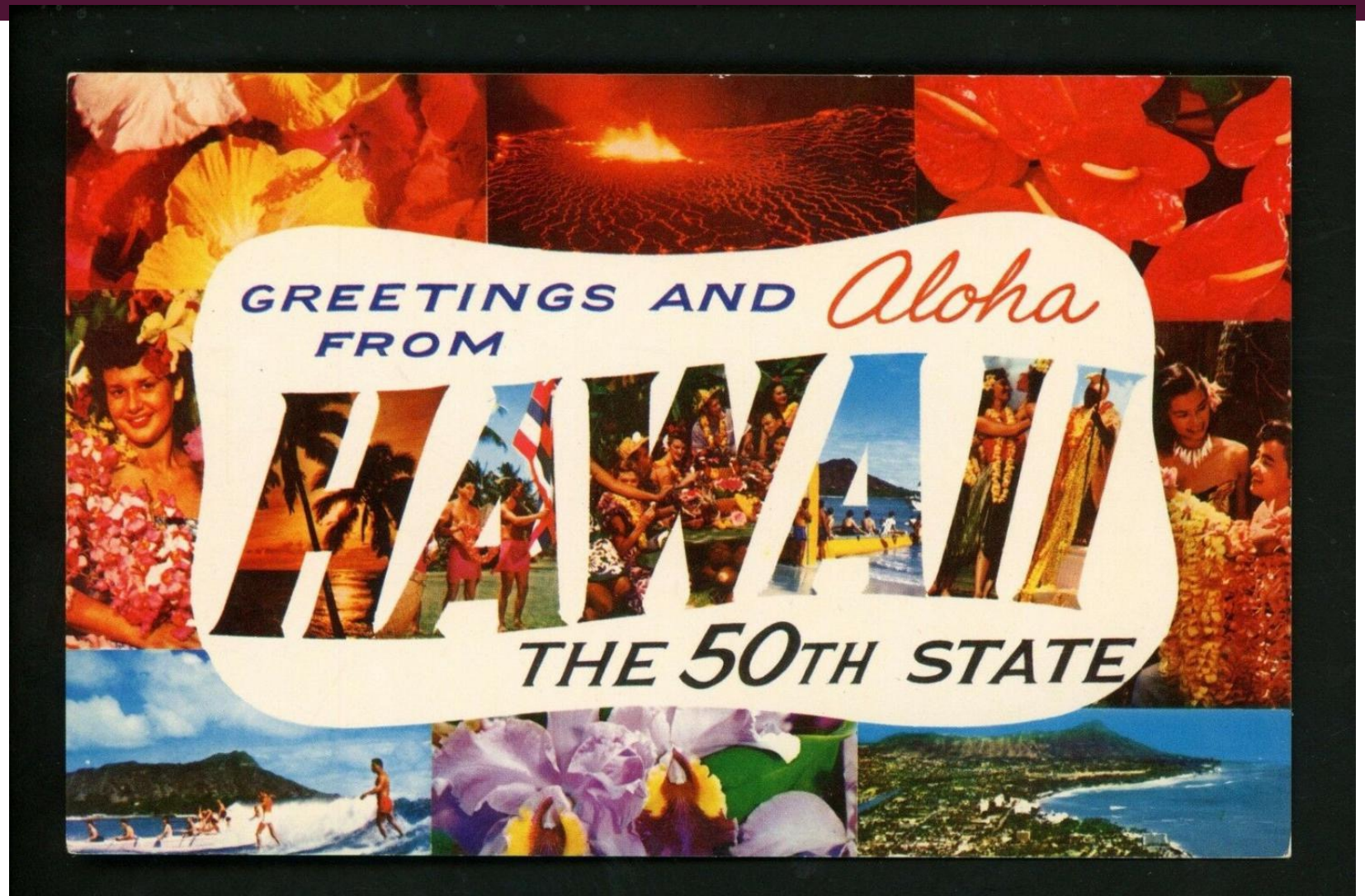
STEP 4: SPREAD

- The opposite of gather is spread

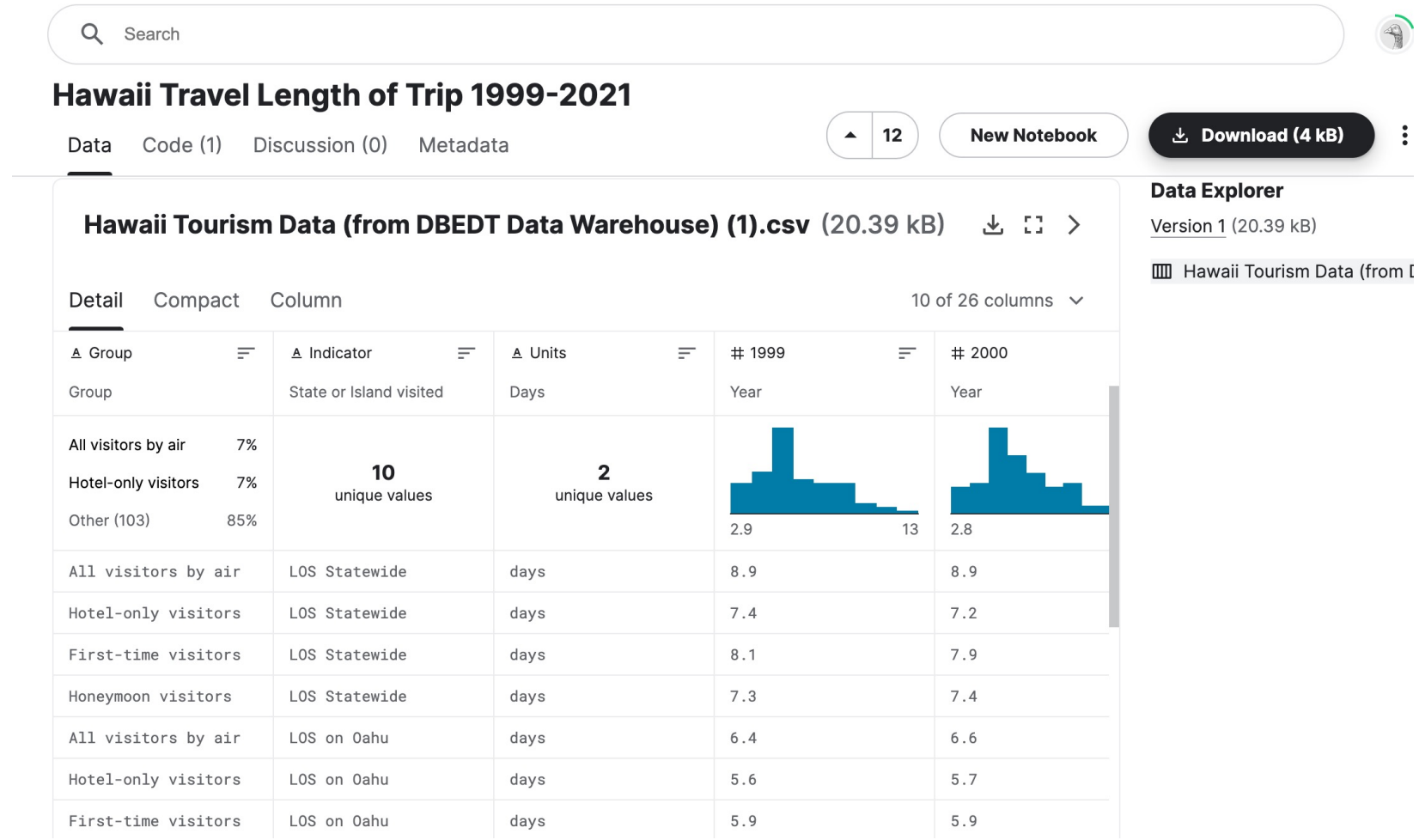
```
# spread
stocksS<-stocksG%>%
  spread(key=stock, value=price)

head(stocksS)
```

```
> head(stocksS)
        time         X         Y         Z
1 2009-01-01 19.37355 23.02356 23.67591
2 2009-01-02 20.18364 20.77969 23.12855
3 2009-01-03 19.16437 18.75752 20.29826
4 2009-01-04 21.59528 15.57060 12.04259
5 2009-01-05 20.32951 22.24986 22.47930
6 2009-01-06 19.17953 19.91013 19.77549
```

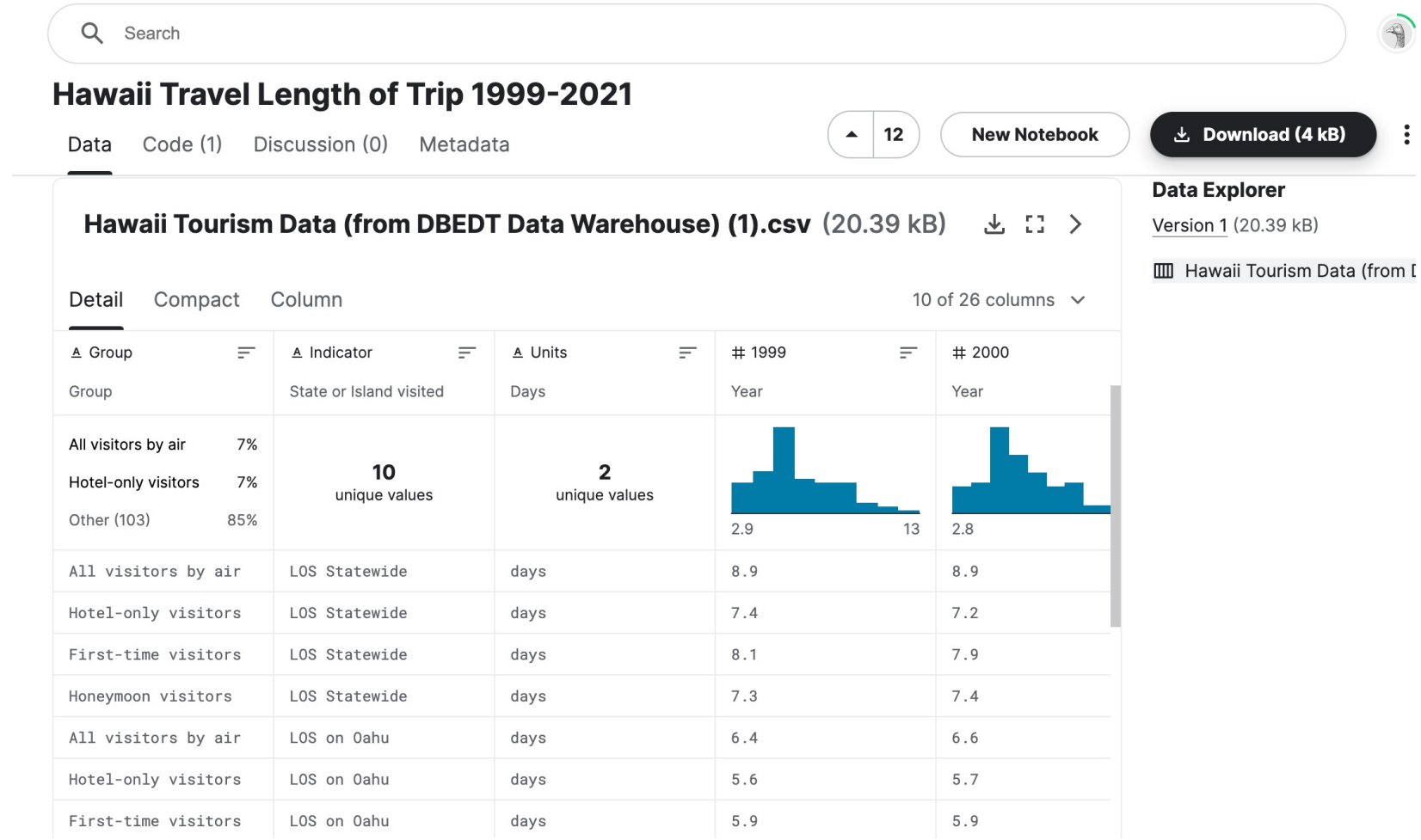
REAL WORLD EXAMPLE



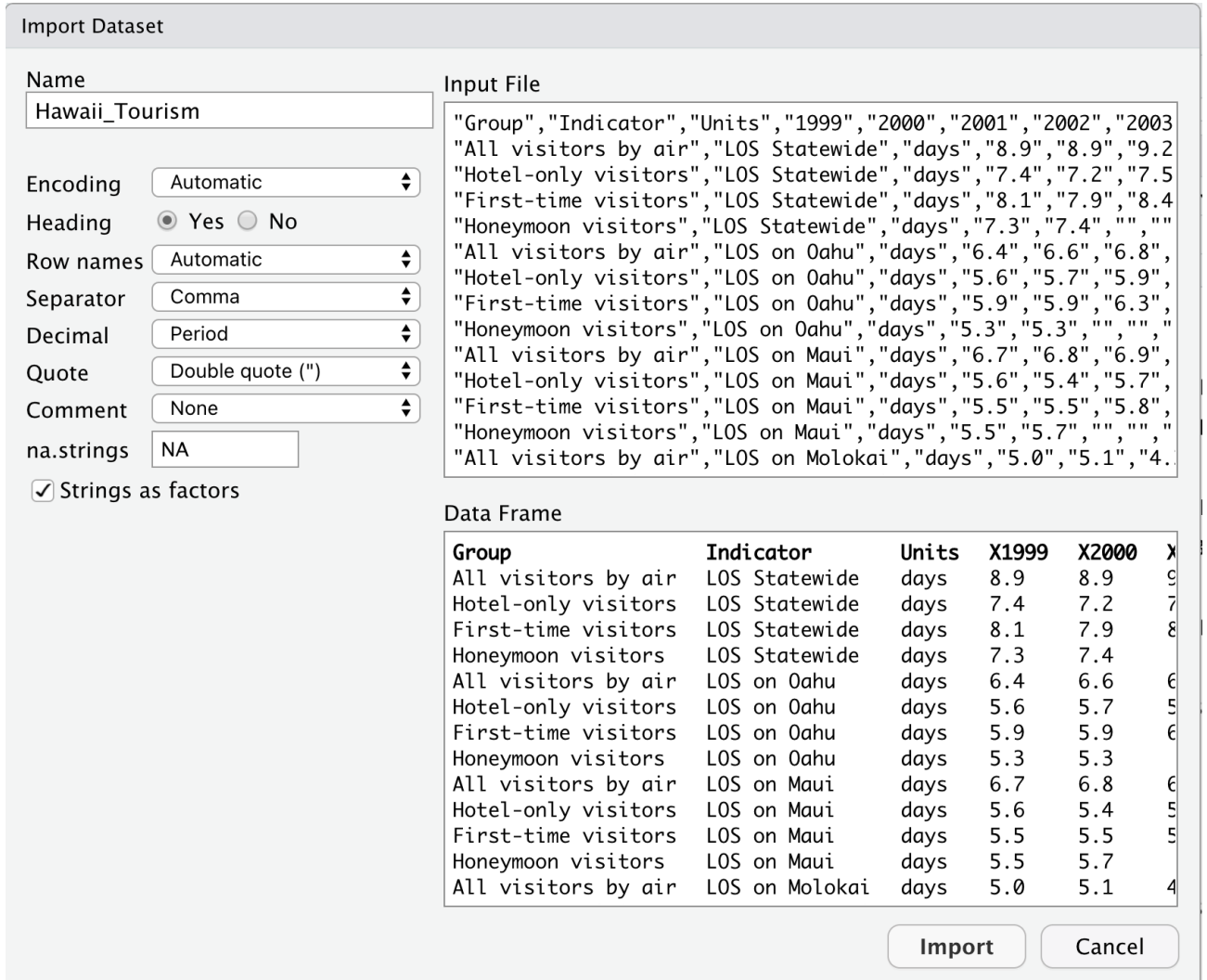
REAL WORLD EXAMPLE



STEP 0: DOWNLOAD THE DATA



STEP 1: IMPORT THE DATA



STEP 2: LOOK AT THE STRUCTURE

Step 2: Look at the structure

```
## LOOK AT THE STRUCTURE
str(hi)
```

```
## 'data.frame':    121 obs. of  26 variables:
##  $ Group    : Factor w/ 17 levels "All visitors by air",..: 1 10 6
8 1 10 6 8 1 10 ...
##  $ Indicator: Factor w/ 10 levels "","LOS in Hilo",..: 10 10 10 10
9 9 9 9 7 7 ...
##  $ Units    : Factor w/ 2 levels "","days": 2 2 2 2 2 2 2 2 2 2
...
##  $ X1999    : num  8.9 7.4 8.1 7.3 6.4 5.6 5.9 5.3 6.7 5.6 ...
##  $ X2000    : num  8.9 7.2 7.9 7.4 6.6 5.7 5.9 5.3 6.8 5.4 ...
##  $ X2001    : num  9.2 7.5 8.4 NA 6.8 5.9 6.3 NA 6.9 5.7 ...
##  $ X2002    : num  9.4 7.7 8.5 NA 7 6.2 6.4 NA 7.2 5.9 ...
##  $ X2003    : num  9.2 7.7 8.5 NA 6.9 6.1 6.3 NA 7.3 6.3 ...
##  $ X2004    : num  9.1 7.3 8.2 8 6.9 5.9 6.2 5.5 7.5 6.2 ...
##  $ X2005    : num  9.1 7.3 8.3 7.8 6.9 6 6.3 5.5 7.5 6.3 ...
##  $ X2006    : num  9.2 7.3 8.3 7.9 6.9 6 6.2 5.5 7.4 6.3 ...
```

STEP 3: RENAME THE COLUMNS

Step 3: Rename the columns

```
## RENAME THE COLUMNS
colnames(hi)[4:26]<-1999:2021
```

STEP 4: GATHER

Step 4: Tidy Data with `gather()`

```
## TIDY YOUR DATA
hiG<-hi%>%
  filter(Units=="days")%>%
  select(-Units)%>%
  gather(key=year, value=days, -c(Group, Indicator))

## LOOK AT THE NEW STRUCTURE
str(hiG)
```

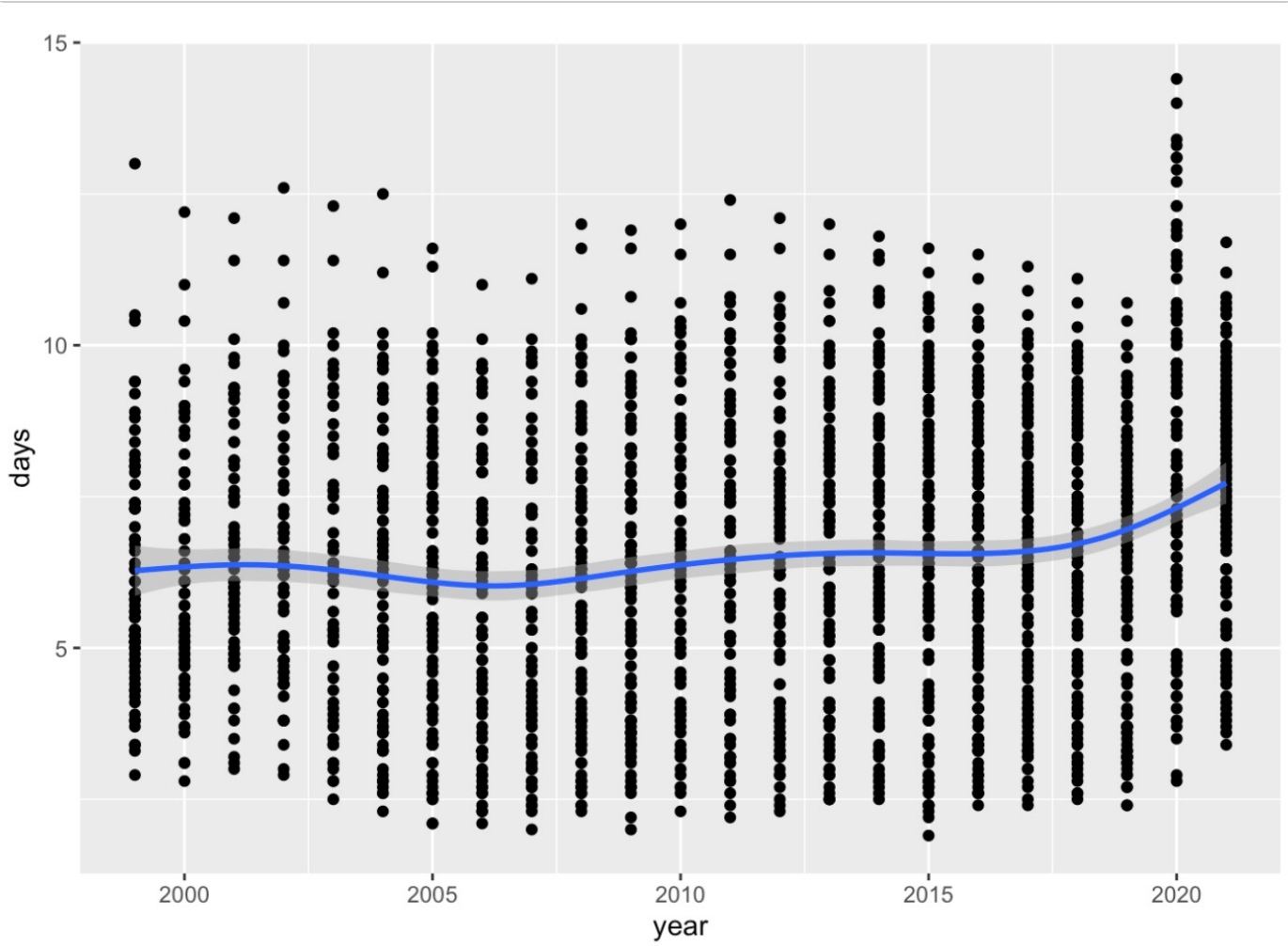
STEP 5:VISUALIZE TRENDS

Step 5: Visualize Trends

```
## YEAR NEEDS TO BE NUMERIC TO PLOT
hiG$year<-as.numeric(hiG$year)

## SCATTERPLOT WITH TREND
ggplot(hiG, aes(x=year, y=days))+
  geom_point()+
  geom_smooth()
```

STEP 5: VISUALIZE TRENDS

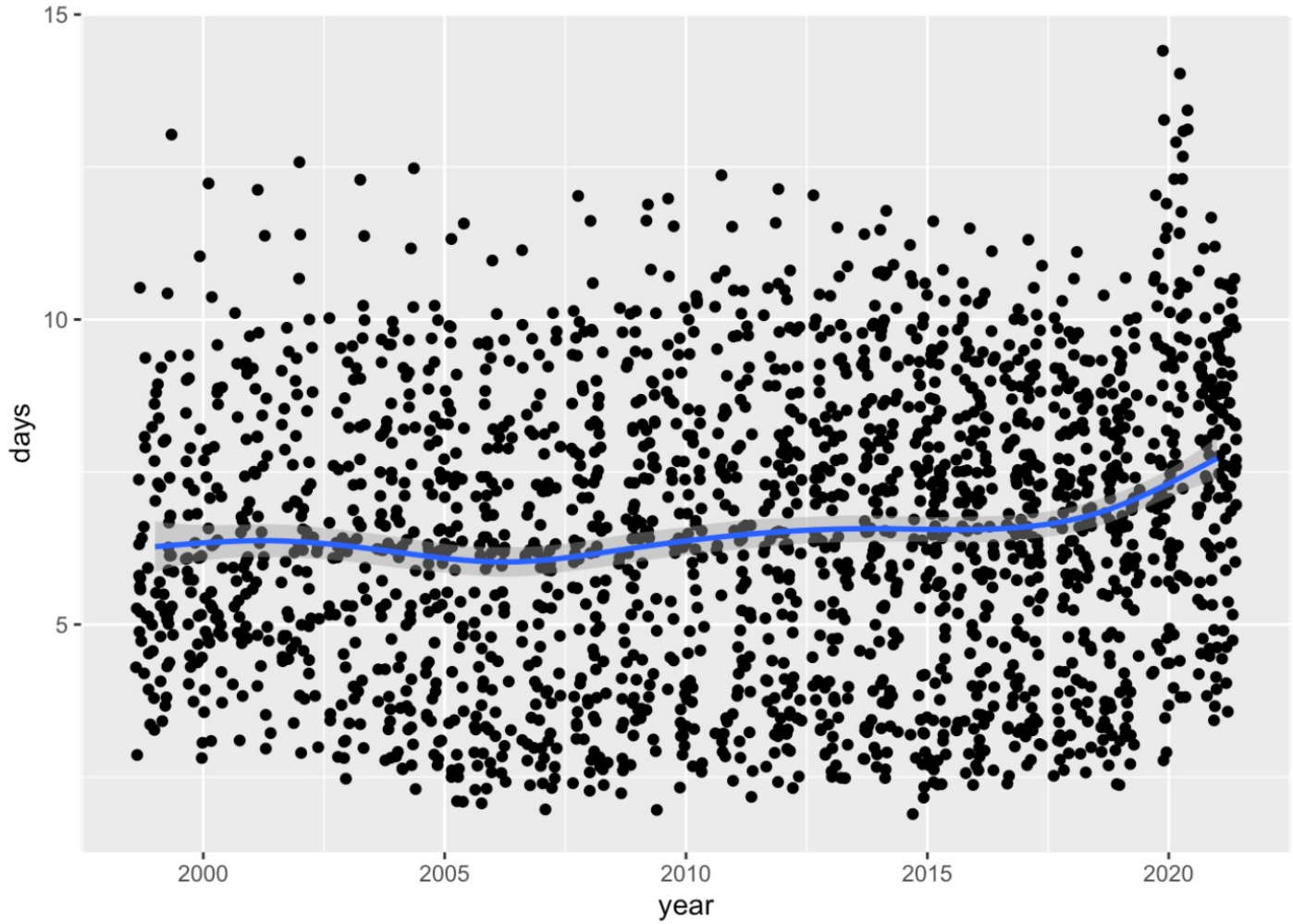
geom_point()



STEP 5: VISUALIZE TRENDS

geom_jitter()

```
## TRY GEOM_JITTER
ggplot(hiG, aes(x=year, y=days))+
  geom_jitter()+
  geom_smooth()
```



MOTIVATING QUESTION

Which island do visitors stay at the longest?

STEP 6: STUDYING SUBGROUPS

Step 6: Studying Subgroups

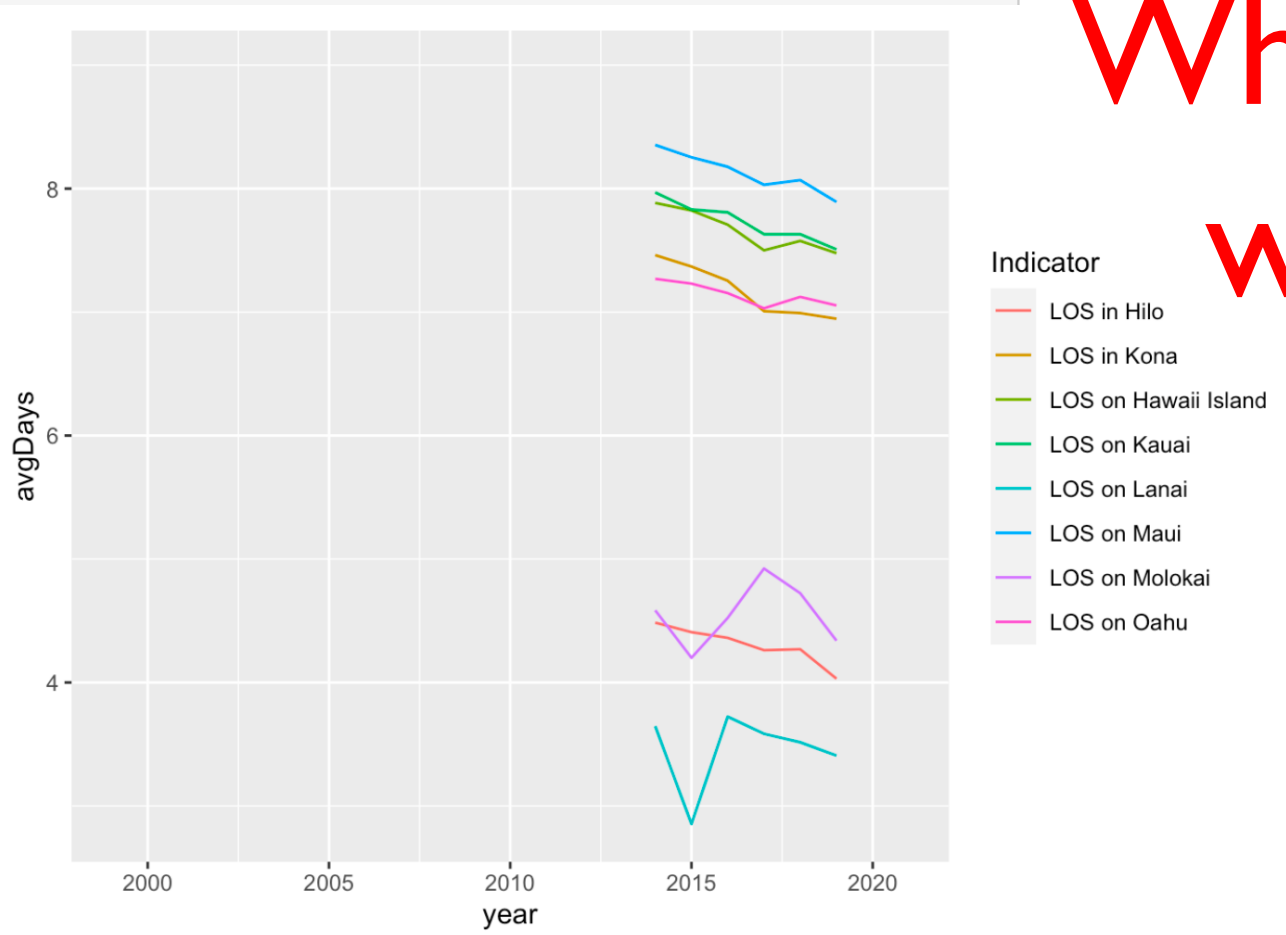
Which island is the most popular to visit?

Use `dplyr` to explore subgroups.

```
## WHAT ISLAND IS MOST POPULAR?
islands<-hiG%>%
  filter(Indicator!="LOS Statewide")%>%
  group_by(Indicator, year)%>%
  summarise(avgDays=mean(days))
```

STEP 6: STUDYING SUBGROUPS

```
ggplot(islands, aes(x=year, y=avgDays, color=Indicator))+
  geom_line()
```



What went wrong?

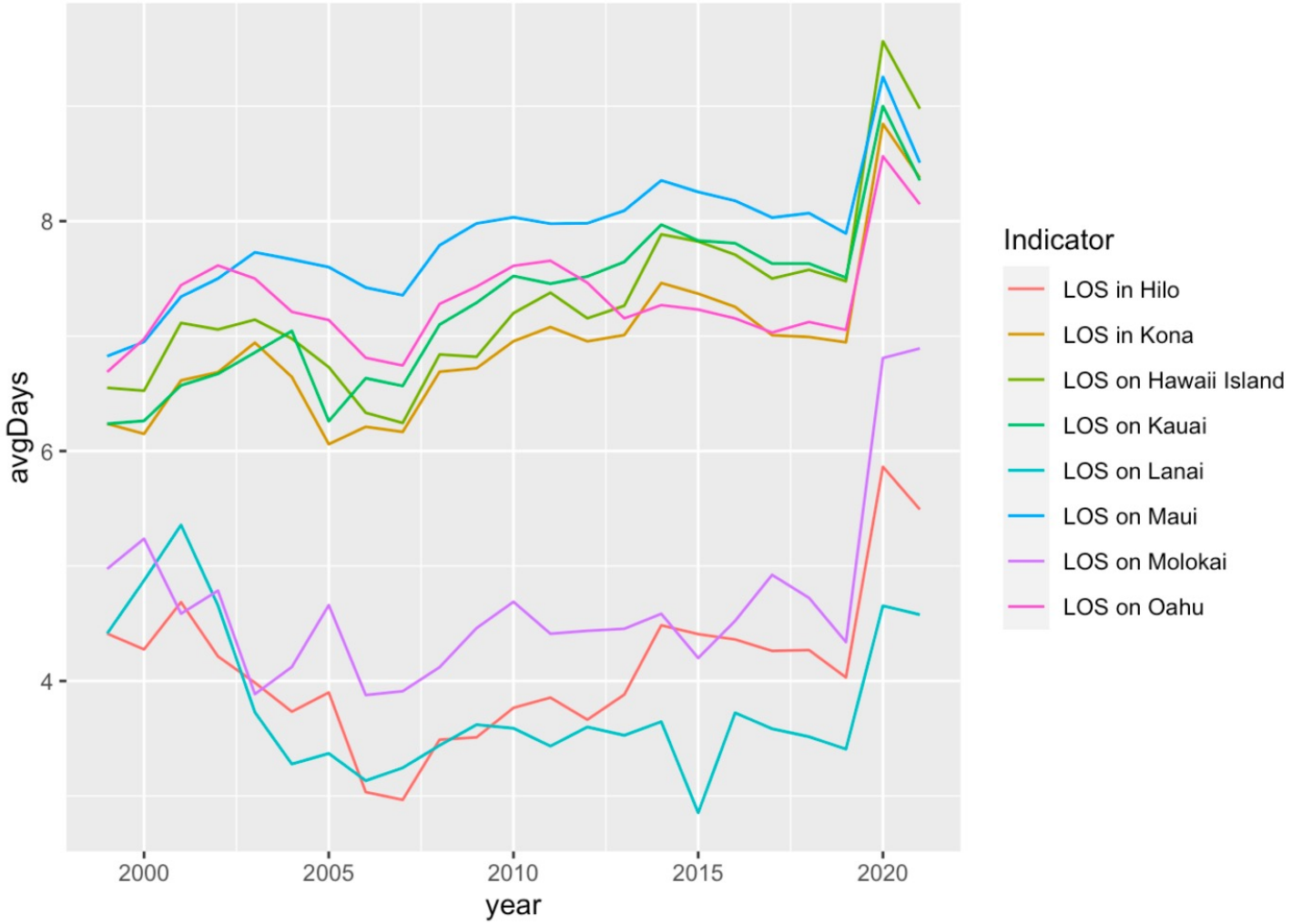
STEP 6: STUDYING SUBGROUPS

Oh no! What happened?!

We forgot about NAs!

```
islands<-hiG%>%
  group_by(Indicator, year)%>%
  filter(Indicator!="LOS Statewide")%>%
  summarise(avgDays=mean(days, na.rm=TRUE))
```

STEP 6: STUDYING SUBGROUPS



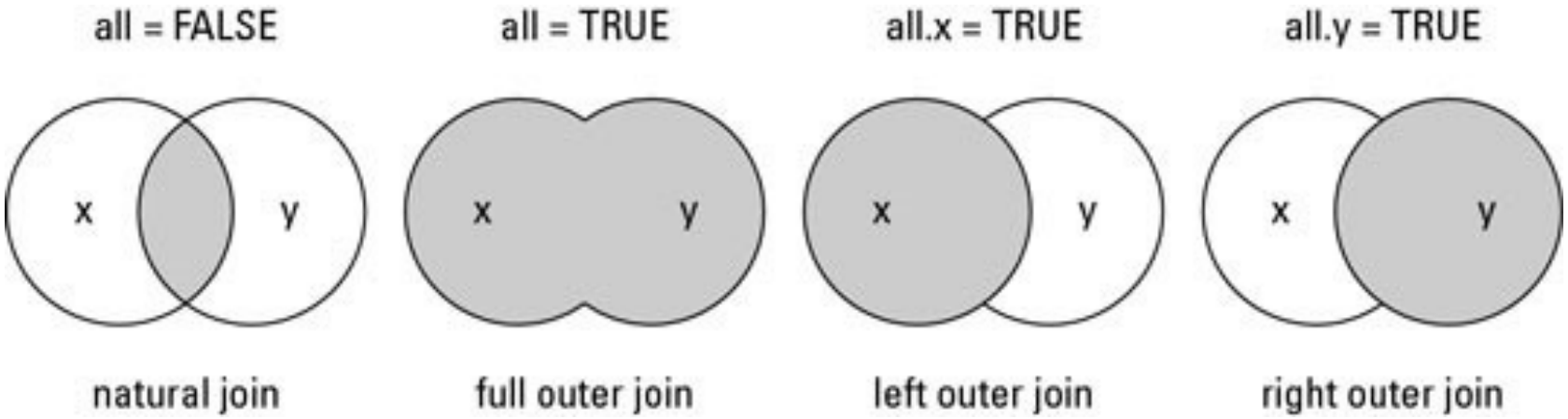
TRY ANOTHER QUESTION

Which group of travelers have the longest trips?

JOINS

JOINS

- There are four types of join methods that can be used in R:
  - Left, right, inner, and full



  - Note: the natural join is called "inner" join in R

JOINS

- **Natural join**: To keep only rows that match from the data frames, specify the argument all=FALSE.

- **Full outer join:** To keep all rows from both data frames, specify all=TRUE.

- **Left outer join:** To include all the rows of your data frame x and only those from y that match, specify x=TRUE.

- **Right outer join:** To include all the rows of your data frame y and only those from x that match, specify y=TRUE.

TOY EXAMPLE FOR JOINS

JOINS

```
superheroes <- tibble::tribble(
  ~name, ~alignment,  ~gender,          ~publisher,
  "Magneto",       "bad",   "male",            "Marvel",
  "Storm",      "good", "female",           "Marvel",
  "Mystique",       "bad", "female",            "Marvel",
  "Batman",      "good",   "male",              "DC",
  "Joker",       "bad",   "male",              "DC",
  "Catwoman",       "bad", "female",                "DC",
  "Hellboy",      "good",   "male", "Dark Horse Comics"
)

publishers <- tibble::tribble(
  ~publisher, ~yr_founded,
  "DC",        1934L,
  "Marvel",        1939L,
  "Image",        1992L
)
```

JOINS

```
# inner join super hero and publisher
insp<-inner_join(superheroes, publishers)
insp
```

superheroes			
name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers	
publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

inner_join(x = superheroes, y = publishers)				
name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934

JOINS

```
# left join super hero and publisher
ljsp<-left_join(superheroes, publishers)
ljsp
```

superheroes			
name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers	
publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

left_join(x = superheroes, y = publishers)				
name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934
Hellboy	good	male	Dark Horse Comics	NA

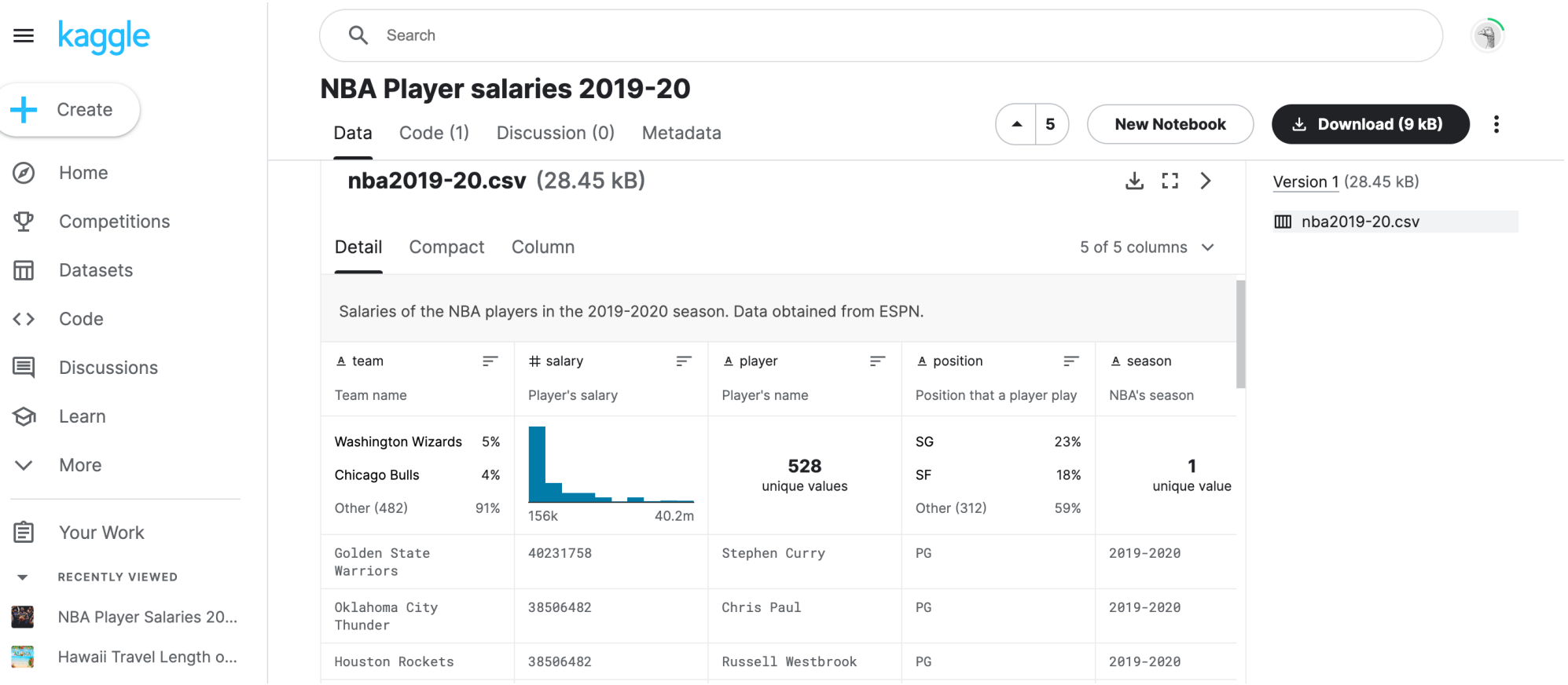
REAL WORLD EXAMPLE: JOINS

REAL WORLD EXAMPLE: JOINS

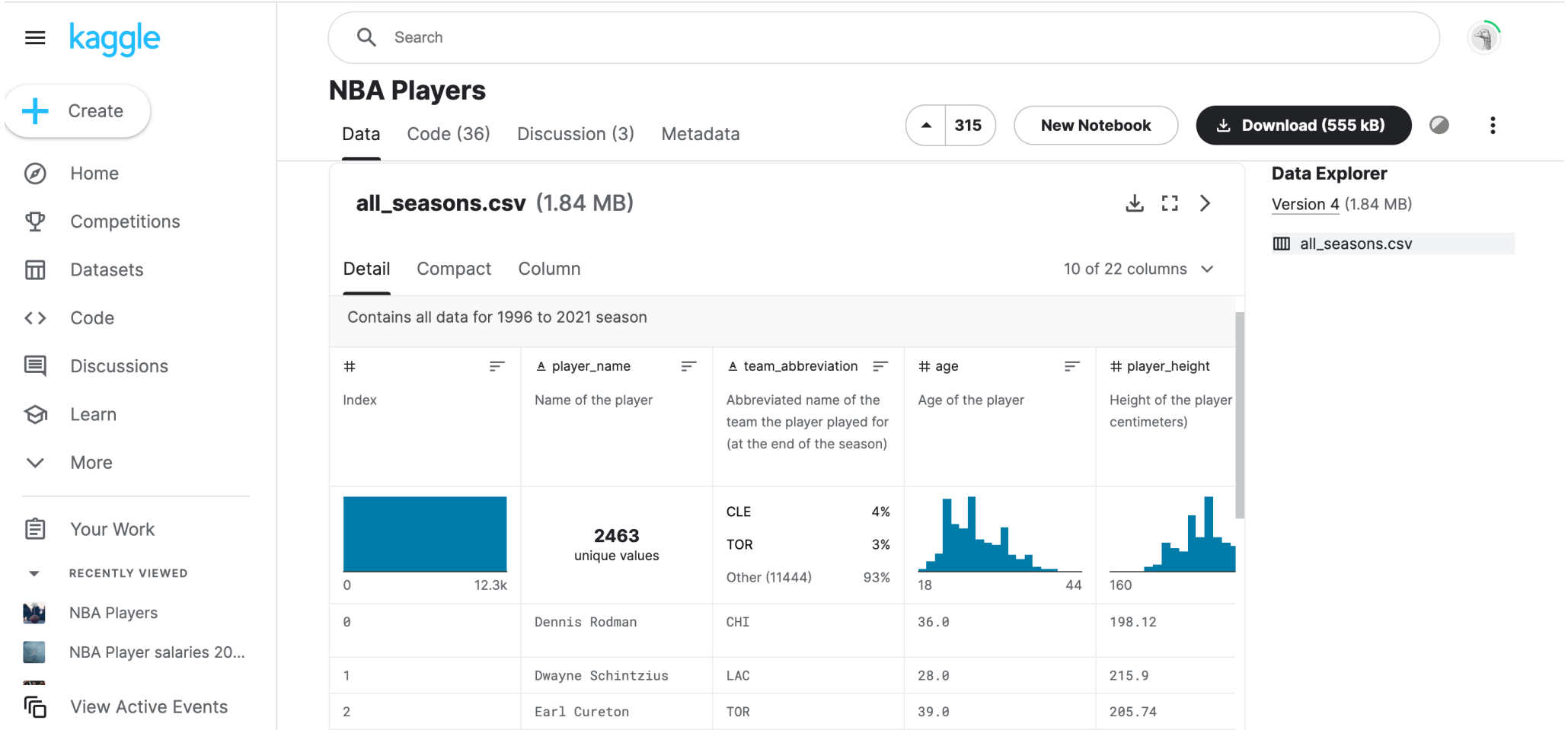
MOTIVATING QUESTION

Is player salary related to player performance?

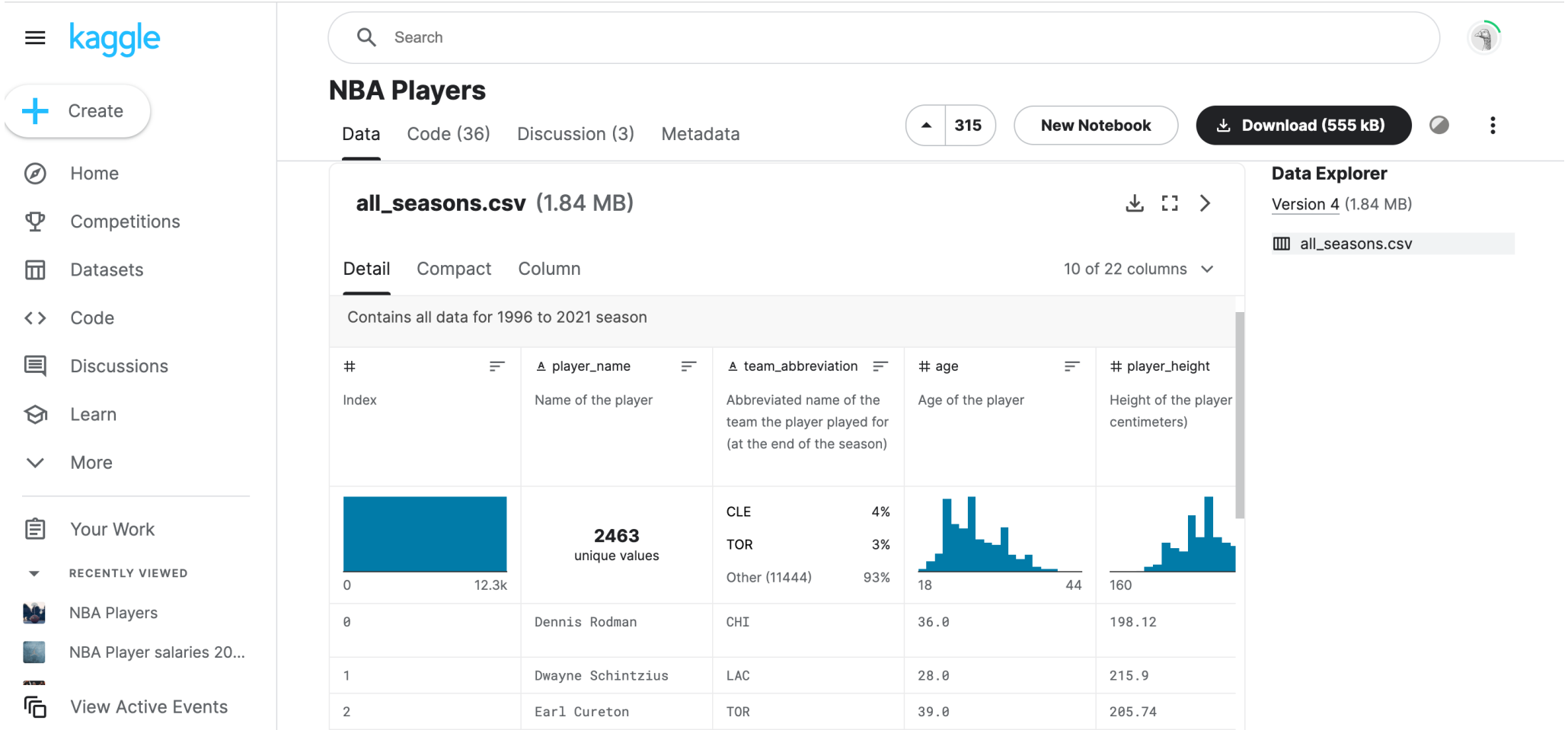
STEP 0: DOWNLOAD THE DATA



STEP 0: DOWNLOAD THE DATA



STEP 0: DOWNLOAD THE DATA



STEP 1: LOAD THE DATA

Step 1: Load Data

```
## SALARY DATA for 2019-2020 season
salaries1920 <- read.csv("~/Downloads/nba2019-20.csv")

## METRICS ON PLAYER PERFORMANCE
## 1996 to 2022
all_seasons <- read.csv("~/Downloads/all_seasons.csv")
```

STEP 2: LOOK AT THE DATA STRUCTURE

Step 2: Learn about your data

```
# SALARIES
str(salaries1920)
```

```
## 'data.frame':    528 obs. of  5 variables:
##  $ team    : Factor w/ 30 levels "Atlanta Hawks",..: 10 21 1
1 30 3 11 14 28 9 23 ...
##  $ salary  : int  40231758 38506482 38506482 38199000 381990
00 38199000 37436858 34996296 34449964 32742000 ...
##  $ player  : Factor w/ 528 levels "Aaron Gordon",..: 457 73
439 255 295 221 323 312 37 483 ...
##  $ position: Factor w/ 7 levels " C"," F"," G",..: 5 5 5 5 6
7 6 5 4 6 ...
##  $ season  : Factor w/ 1 level "2019-2020": 1 1 1 1 1 1 1 1
1 1 ...
```

STEP 2: LOOK AT THE DATA STRUCTURE

```
# METRICS
str(all_seasons)
```

```
## 'data.frame':    12305 obs. of  22 variables:
##  $ X                : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ player_name      : Factor w/ 2463 levels "A.C. Green","A.J. Bramlett",..: 585 705 716 720 721 727
728 737 738 745 ...
##  $ team_abbreviation: Factor w/ 36 levels "ATL","BKN","BOS",..: 6 14 33 8 17 12 15 15 1 18 ...
##  $ age              : num  36 28 39 24 34 38 25 28 29 28 ...
##  $ player_height    : num  198 216 206 203 206 ...
##  $ player_weight    : num  99.8 117.9 95.3 100.7 108.9 ...
##  $ college          : Factor w/ 347 levels " ","                                    ",..: 255 85 75 2
99 315 110 275 58 324 155 ...
##  $ country          : Factor w/ 82 levels "Angola","Argentina",..: 79 79 79 79 79 79 79 79 79 79 ...
##  $ draft_year       : Factor w/ 47 levels "1963","1976",..: 11 15 4 20 10 6 19 15 17 16 ...
##  $ draft_round      : Factor w/ 9 levels "0","1","2","3",..: 3 2 4 2 2 3 2 2 9 3 ...
##  $ draft_number     : Factor w/ 76 levels "0","1","10","11",..: 27 24 61 75 3 29 3 27 76 38 ...
##  $ gp               : int  55 15 9 64 27 52 80 77 71 82 ...
##  $ pts              : num  5.7 2.3 0.8 3.7 2.4 8.2 17.2 14.9 5.7 6.9 ...
##  $ reb              : num  16.1 1.5 1 2.3 2.4 2.7 4.1 8 1.6 1.5 ...
##  $ ast              : num  3.1 0.3 0.4 0.6 0.2 1 3.4 1.6 1.3 3 ...
##  $ net_rating       : num  16.1 12.3 -2.1 -8.7 -11.2 4.1 4.1 3.3 -0.3 -1.2 ...
##  $ oreb_pct         : num  0.186 0.078 0.105 0.06 0.109 0.034 0.035 0.095 0.036 0.018 ...
##  $ dreb_pct         : num  0.323 0.151 0.102 0.149 0.179 0.126 0.091 0.183 0.076 0.081 ...
##  $ usg_pct          : num  0.1 0.175 0.103 0.167 0.127 0.22 0.209 0.222 0.172 0.177 ...
##  $ ts_pct           : num  0.479 0.43 0.376 0.399 0.611 0.541 0.559 0.52 0.539 0.557 ...
##  $ ast_pct          : num  0.113 0.048 0.148 0.077 0.04 0.102 0.149 0.087 0.141 0.262 ...
##  $ season           : Factor w/ 26 levels "1996-97","1997-98",..: 1 1 1 1 1 1 1 1 1 1 ...
```

THESE DATA SETS ARE APPLES AND ORANGES!

STEP 3: WRANGLE YOUR DATA

Step 3: Wrangle your data

We need to make an apples to apples comparison.

- Filter the season data by 2019-2020 season.
- We also need to have the same name for the variable we wish to match.

```
season1920<-all_seasons%>%
  filter(season=="2019-20")%>%
  select(-season)%>%
  mutate(player=player_name)
```

NOW WE HAVE APPLES TO APPLES

STEP 4: JOIN THE DATA

Step 4: Join the data

```
joinNBA<-salaries1920%>%
  left_join(season1920)
```

```
## Joining, by = "player"
```

```
str(joinNBA)
```

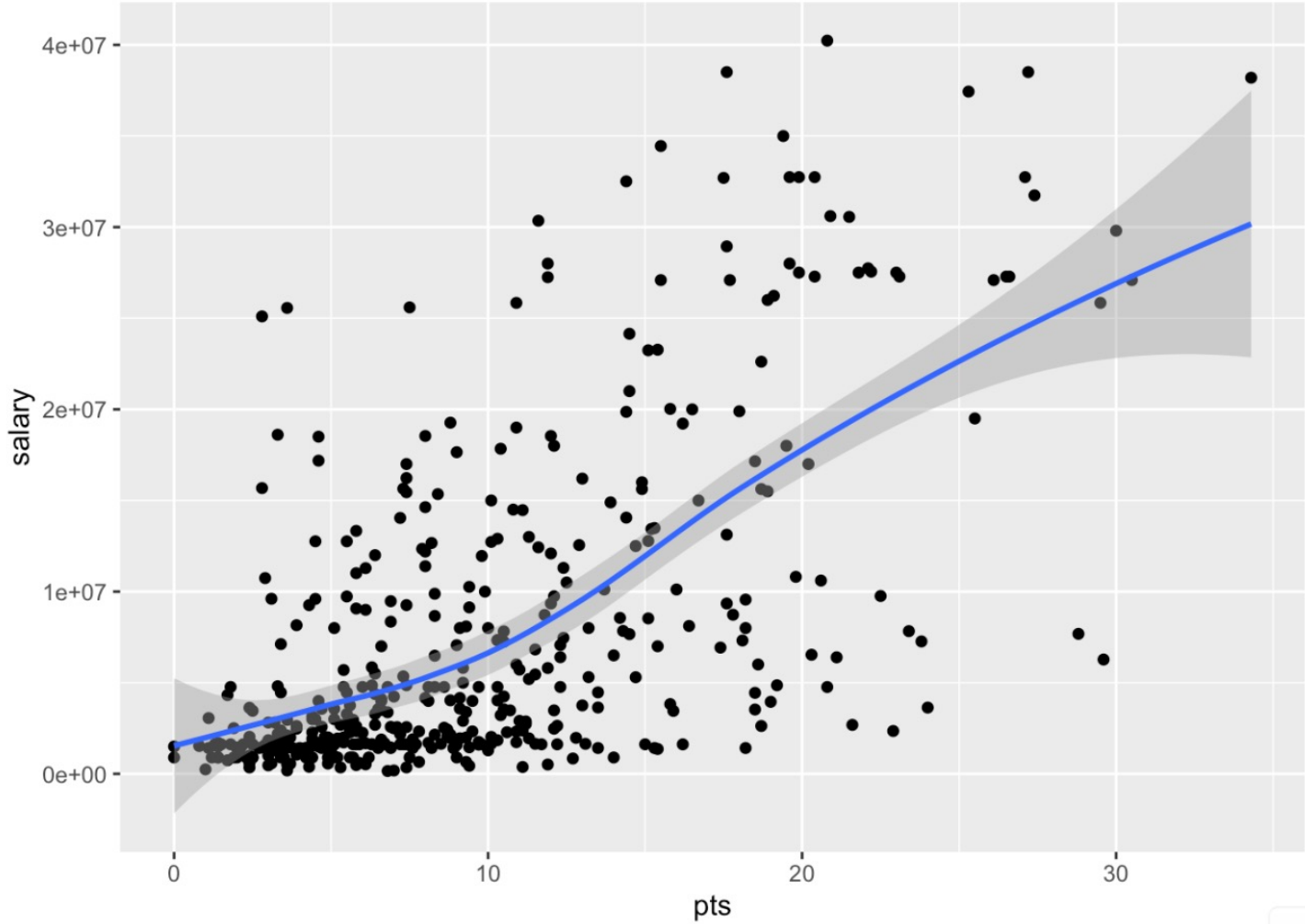
MOTIVATING QUESTION

Is player salary related to player performance?

STEP 5:VISUALIZE

Step 5: Visualize

```
ggplot(joinNBA, aes(x=pts, y=salary))+
  geom_point()+
  geom_smooth()
```



BONUS/EXTRA CONTENT (IF TIME ALLOWS)

TIDYR: UNITE

- Sometimes we want to combine several columns into one string
  - For example with dates

```
set.seed(1)
date <- as.Date('2016-01-01') + 0:14
hour <- sample(1:24, 15)
min <- sample(1:60, 15)
second <- sample(1:60, 15)
event <- sample(letters, 15)
data <- data.frame(date, hour, min, second, event)
data
```

	date	hour	min	second	event
1	2016-01-01	7	30	29	u
2	2016-01-02	9	43	36	a
3	2016-01-03	13	58	60	l
4	2016-01-04	20	22	11	q
5	2016-01-05	5	44	47	p
6	2016-01-06	18	52	37	k
7	2016-01-07	19	12	43	r
8	2016-01-08	12	35	6	i
9	2016-01-09	11	7	38	e
10	2016-01-10	1	14	21	b
11	2016-01-11	3	20	42	w
12	2016-01-12	14	1	32	t
13	2016-01-13	23	19	52	h
14	2016-01-14	21	41	26	s
15	2016-01-15	8	16	25	o

TIDYR: UNITE

- We can use unit to do this and define what character separate the components

```
dataUnite <- data %>%
  unite(datehour, date, hour, sep = ' ') %>%
  unite(datetime, datehour, min, second, sep = ':')

str(dataUnite)
```

```
> dataUnite
              datetime event
1   2016-01-01 7:30:29     u
2   2016-01-02 9:43:36     a
3  2016-01-03 13:58:60     l
4  2016-01-04 20:22:11     q
5   2016-01-05 5:44:47     p
6  2016-01-06 18:52:37     k
7  2016-01-07 19:12:43     r
8   2016-01-08 12:35:6     i
9   2016-01-09 11:7:38     e
```

TIDYR: SEPARATE

- The opposite of unite is separate

```
# seperate
dataSep <- dataUnite %>%
  separate(datetime, c('date', 'time'), sep = ' ') %>%
  separate(time, c('hour', 'min', 'second'), sep = ':')

dataSep
```

```
> dataSep
         date hour min second event
1  2016-01-01    7  30     29     u
2  2016-01-02    9  43     36     a
3  2016-01-03   13  58     60     l
4  2016-01-04   20  22     11     q
5  2016-01-05    5  44     47     p
6  2016-01-06   18  52     37     k
7  2016-01-07   19  12     43     r
8  2016-01-08   12  35      6     i
9  2016-01-09   11   7     38     e
10 2016-01-10    1  14     21     b
11 2016-01-11    3  20     42     w
12 2016-01-12   14   1     32     t
13 2016-01-13   23  19     52     h
14 2016-01-14   21  41     26     s
15 2016-01-15    8  16     25     o
```