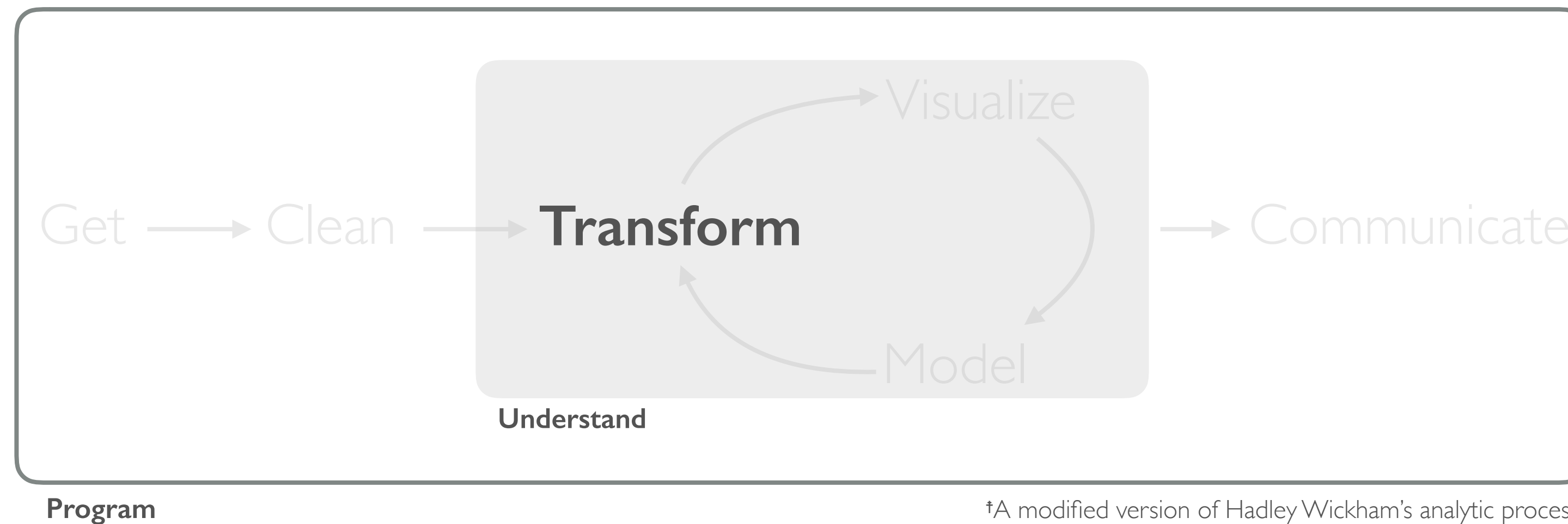


DATA TRANSFORMATION



†A modified version of Hadley Wickham's analytic process

“Data sets contain more information than they display.”

– Garrett Grolemund

dplyr

You are going to learn the five key **dplyr** functions that allow you to solve the vast majority of your data manipulation challenges:

- **filter**: pick observations based on values
- **arrange**: reorder data
- **select**: pick variables
- **mutate**: create new variables
- **summarise**: summarize data by functions of choice

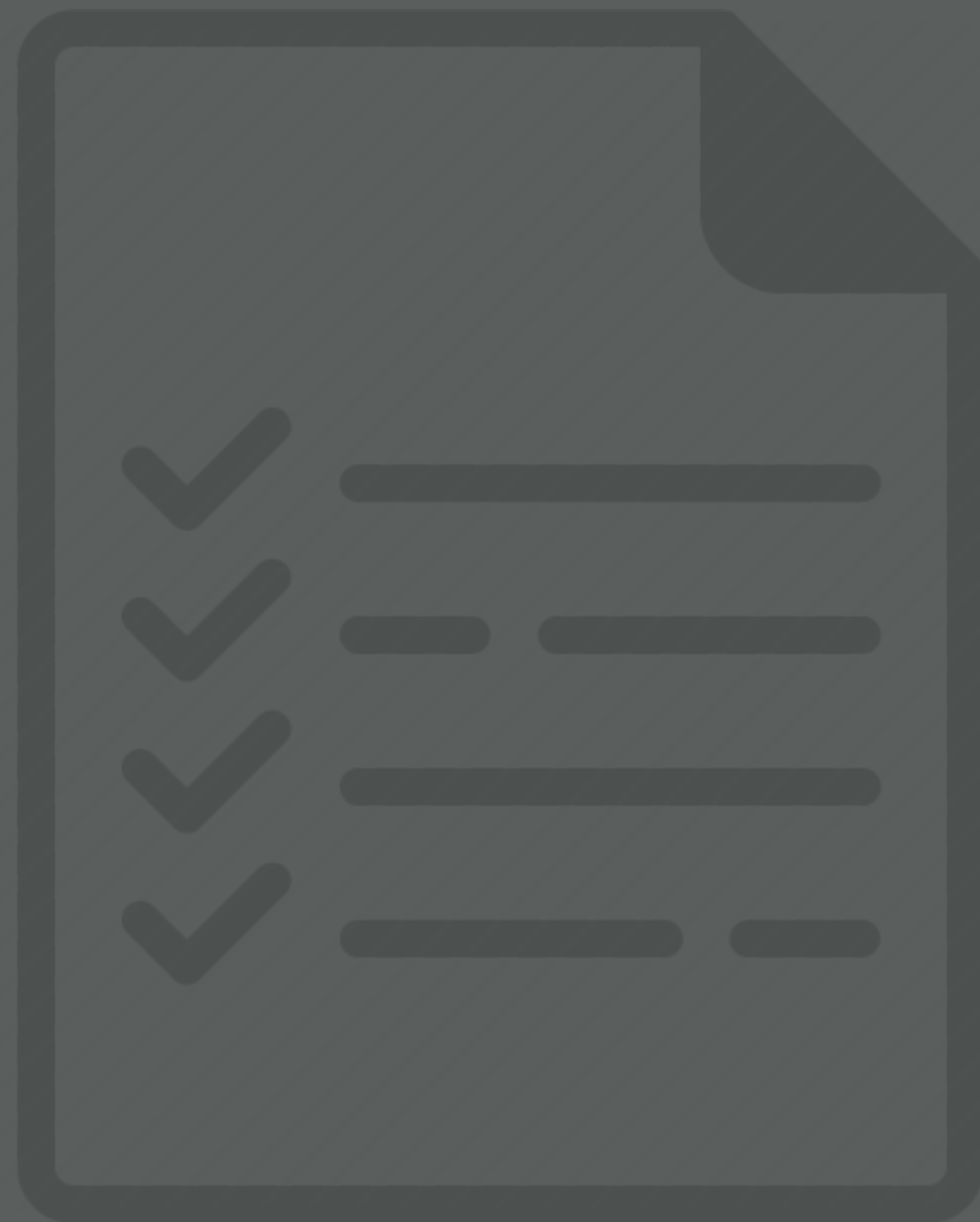


BASICS

All functions work similarly:

- The first argument is a data frame
- Subsequent arguments describe what to do
- Output is a new data frame

PREREQUISITES



PACKAGE PREREQUISITE

```
library(nycflights13)
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages -----
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

DATA PREREQUISITE

flights

A tibble: 336,776 × 19

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>	<int>
1	2013	1	1	517	515	2	830	819	11	UA	1545
2	2013	1	1	533	529	4	850	830	20	UA	1714
3	2013	1	1	542	540	2	923	850	33	AA	1141
4	2013	1	1	544	545	-1	1004	1022	-18	B6	725
5	2013	1	1	554	600	-6	812	837	-25	DL	461
6	2013	1	1	554	558	-4	740	728	12	UA	1696
7	2013	1	1	555	600	-5	913	854	19	B6	507
8	2013	1	1	557	600	-3	709	723	-14	EV	5708
9	2013	1	1	557	600	-3	838	846	-8	B6	79
10	2013	1	1	558	600	-2	753	745	8	AA	301

... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>,

air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>

YOUR TURN!

*Are there vignettes for the **dp1yr** package?*

*Can you find additional documentation explaining the
flights data set?*

SOLUTION

```
# are there vignettes for the dplyr package -> yes, 8 of them  
vignette(package = "dplyr")
```

```
# additional documentation for the mpg data set  
?flights
```

filter

Filter values based on defined conditions



BASIC FILTERING

Filter based on one or more variables

```
filter(flights, month == 1)
```

```
# A tibble: 27,004 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	2013	1	1	517	515	2	830	819	11
2	2013	1	1	533	529	4	850	830	20
3	2013	1	1	542	540	2	923	850	33
4	2013	1	1	544	545	-1	1004	1022	-18
5	2013	1	1	554	600	-6	812	837	-25
6	2013	1	1	554	558	-4	740	728	12
7	2013	1	1	555	600	-5	913	854	19
8	2013	1	1	557	600	-3	709	723	-14
9	2013	1	1	557	600	3	838	846	8

BASIC FILTERING

Filter based on one or more variables

```
filter(flights, month == 1, day == 1)
```

```
# A tibble: 842 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	2013	1	1	517	515	2	830	819	11
2	2013	1	1	533	529	4	850	830	20
3	2013	1	1	542	540	2	923	850	33
4	2013	1	1	544	545	-1	1004	1022	-18
5	2013	1	1	554	600	-6	812	837	-25
6	2013	1	1	554	558	-4	740	728	12
7	2013	1	1	555	600	-5	913	854	19
8	2013	1	1	557	600	-3	709	723	-14
9	2013	1	1	557	600	3	838	846	8

BASIC FILTERING

Filter based on one or more variables

```
filter(flights, month == 1, day == 1, dep_delay > 0)
```

```
# A tibble: 352 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	2013	1	1	517	515	2	830	819	11
2	2013	1	1	533	529	4	850	830	20
3	2013	1	1	542	540	2	923	850	33
4	2013	1	1	601	600	1	844	850	-6
5	2013	1	1	608	600	8	807	735	32
6	2013	1	1	611	600	11	945	931	14
7	2013	1	1	613	610	3	925	921	4
8	2013	1	1	623	610	13	920	915	5
9	2013	1	1	632	608	24	740	728	12

SAVE NEW DATA FRAME

- Save filter data frame using assignment operator (<-)
- Save and view filtered data frame by wrapping entire function with ()

```
jan1 <- filter(flights, month == 1, day == 1)
```

```
(dec25 <- filter(flights, month == 12, day == 25))
```

```
# A tibble: 719 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	2013	12	25	456	500	-4	649	651	-2
2	2013	12	25	524	515	9	805	814	-9
3	2013	12	25	542	540	2	832	850	-18
4	2013	12	25	546	550	-4	1022	1027	-5
5	2013	12	25	556	600	-4	730	745	-15
6	2013	12	25	557	600	-3	743	752	-9

LOGICAL TESTS

```
12 == 12
```

```
[1] TRUE
```

```
12 <= c(12, 11)
```

```
[1] TRUE FALSE
```

```
12 %in% c(12, 11, 8)
```

```
[1] TRUE
```

```
x <- c(12, NA, 11, NA, 8)
```

```
is.na(x)
```

```
[1] FALSE TRUE FALSE TRUE FALSE
```

?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

COMPARISON

What will these operations produce?

```
filter(flights, month == 12)
filter(flights, month != 12)
filter(flights, month %in% c(11, 12))
filter(flights, arr_delay <= 120)
filter(flights, !(arr_delay <= 120))
filter(flights, is.na(tailnum))
```


MULTIPLE LOGICAL TESTS

```
12 == 12 & 12 < 14
```

```
[1] TRUE
```

```
12 == 12 & 12 < 10
```

```
[1] FALSE
```

```
12 == 12 | 12 < 10
```

```
[1] TRUE
```

```
any(12 == 12, 12 < 10)
```

```
[1] TRUE
```

```
all(12 == 12, 12 < 10)
```

```
[1] FALSE
```

?base::Logic

&	boolean and
 	boolean or
xor	exclusively x or y
!	not
any	any true
all	all true

MULTIPLE COMPARISONS

Using comma is same as using &

```
filter(flights, month == 12, day == 25)
filter(flights, month == 12 & day == 25)
```

Use `%in%` as a shortcut for `|`

```
filter(flights, month == 11 | month == 12)
filter(flights, month %in% c(11, 12))
```

Are these the same????

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

YOUR TURN!

- I. Find the number of flights that
 - (a) Had an arrival delay of two or more hours
 - (b) Flew to Houston (IAH or HOU)
 - (c) Were operated by United, American, or Delta
 - (d) Departed in summer (July, August, and September) and have missing departure time values
 - (e) Arrived more than two hours late, but didn't leave late
 - (f) Were delayed by at least an hour, but made up over 30 minutes in flight

SOLUTION

```
# Had an arrival delay of two or more hours
```

```
filter(flights, arr_delay >= 120)
```

```
# Flew to Houston (IAH or HOU)
```

```
filter(flights, dest %in% c("IAH", "HOU"))
```

```
# Were operated by United, American, or Delta
```

```
filter(flights, carrier %in% c("UA", "AA", "DL"))
```

```
# Departed in summer (July, August, and September) and have missing departure time values
```

```
filter(flights, month %in% c(7, 8, 9) & is.na(dep_time))
```

```
# Arrived more than two hours late, but didn't leave late
```

```
filter(flights, arr_delay > 120, dep_delay <= 0)
```

```
# Were delayed by at least an hour, but made up over 30 minutes in flight
```

```
filter(flights, dep_delay >= 60, arr_delay <= (dep_delay - 30))
```

arrange

Reorder data



ORDERING YOUR DATA

Order data based on one or more variables

```
arrange(flights, dep_delay)
```

A tibble: 336,776 × 19

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>	<int>
1	2013	12	7	2040	2123	-43	40	2352	48	B6	97
2	2013	2	3	2022	2055	-33	2240	2338	-58	DL	1715
3	2013	11	10	1408	1440	-32	1549	1559	-10	EV	5713
4	2013	1	11	1900	1930	-30	2233	2243	-10	DL	1435
5	2013	1	29	1703	1730	-27	1947	1957	-10	F9	837
6	2013	8	9	729	755	-26	1002	955	7	MQ	3478
7	2013	10	23	1907	1932	-25	2143	2143	0	EV	4361
8	2013	3	30	2030	2055	-25	2213	2250	-37	MQ	4573
9	2013	3	2	1431	1455	-24	1601	1631	-30	9E	3318
10	2013	5	5	934	958	-24	1225	1309	-44	B6	375

with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>

ORDERING YOUR DATA

Order data based on **one or more variables**

```
arrange(flights, dep_delay, arr_delay)
```

```
# A tibble: 336,776 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>	<int>
1	2013	12	7	2040	2123	-43	40	2352	48	B6	97
2	2013	2	3	2022	2055	-33	2240	2338	-58	DL	1715
3	2013	11	10	1408	1440	-32	1549	1559	-10	EV	5713
4	2013	1	11	1900	1930	-30	2233	2243	-10	DL	1435
5	2013	1	29	1703	1730	-27	1947	1957	-10	F9	837
6	2013	8	9	729	755	-26	1002	955	7	MQ	3478
7	2013	3	30	2030	2055	-25	2213	2250	-37	MQ	4573
8	2013	10	23	1907	1932	-25	2143	2143	0	EV	4361
9	2013	5	5	934	958	-24	1225	1309	-44	B6	375
10	2013	9	18	1631	1655	-24	1812	1845	-33	AA	2223

with 336,766 more rows and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>

ORDERING YOUR DATA

Reverse the order by using `desc()`

```
arrange(flights, desc(dep_delay))
```

```
# A tibble: 336,776 × 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>	<int>
1	2013	1	9	641	900	1301	1242	1530	1272	HA	51
2	2013	6	15	1432	1935	1137	1607	2120	1127	MQ	3535
3	2013	1	10	1121	1635	1126	1239	1810	1109	MQ	3695
4	2013	9	20	1139	1845	1014	1457	2210	1007	AA	177
5	2013	7	22	845	1600	1005	1044	1815	989	MQ	3075
6	2013	4	10	1100	1900	960	1342	2211	931	DL	2391
7	2013	3	17	2321	810	911	135	1020	915	DL	2119
8	2013	6	27	959	1900	899	1236	2226	850	DL	2007
9	2013	7	22	2257	759	898	121	1026	895	DL	2047
10	2013	12	5	756	1700	896	1058	2020	878	AA	172

```
# with 336,766 more rows and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>
```


ORDERING YOUR DATA

Note that [missing values](#) are always sorted at the end:

```
df <- tibble(x = c(5, 2, 5, NA))  
# A tibble: 4 × 1  
      x  
  <dbl>  
1     5  
2     2  
3     5  
4    NA
```

```
arrange(df, x)  
# A tibble: 4 × 1  
      x  
  <dbl>  
1     2  
2     5  
3     5  
4    NA
```

```
arrange(df, desc(x))  
# A tibble: 4 × 1  
      x  
  <dbl>  
1     5  
2     5  
3     2  
4    NA
```

YOUR TURN!

1. Sort flights to find the most delayed flights
2. Find the flights that left earliest
3. Which flights travelled the longest?
4. Which travelled the shortest?

SOLUTION

```
# Sort flights to find the most delayed flights  
arrange(flights, desc(dep_delay))
```

```
# Find the flights that left earliest  
arrange(flights, dep_time)
```

```
# Which flights travelled the longest?  
arrange(flights, desc(distance))
```

```
# Which travelled the shortest?  
arrange(flights, distance)
```

select

Select variables of concern



SELECTING VARIABLES

Select one or more variables

```
select(flights, year, month, day)
```

```
# A tibble: 336,776 × 3
```

	year	month	day
	<int>	<int>	<int>
1	2013	1	1
2	2013	1	1
3	2013	1	1
4	2013	1	1
5	2013	1	1
6	2013	1	1
7	2013	1	1
8	2013	1	1
9	2013	1	1

Same

Results

```
select(flights, year:day)
```

```
# A tibble: 336,776 × 3
```

	year	month	day
	<int>	<int>	<int>
1	2013	1	1
2	2013	1	1
3	2013	1	1
4	2013	1	1
5	2013	1	1
6	2013	1	1
7	2013	1	1
8	2013	1	1
9	2013	1	1

SELECTING VARIABLES

Deselect one or more variables

```
select(flights, -(year:day))
```

```
# A tibble: 336,776 × 16
```

	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight
	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>	<int>
1	517	515	2	830	819	11	UA	1545
2	533	529	4	850	830	20	UA	1714
3	542	540	2	923	850	33	AA	1141
4	544	545	-1	1004	1022	-18	B6	725
5	554	600	-6	812	837	-25	DL	461
6	554	558	-4	740	728	12	UA	1696
7	555	600	-5	913	854	19	B6	507
8	557	600	-3	709	723	-14	EV	5708
9	557	600	-3	838	846	-8	B6	79
10	558	600	-2	753	745	8	AA	301

```
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>.
```

USEFUL `select` FUNCTIONS

* Blue functions come in `dplyr`

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

SELECTING VARIABLES

Select variables based on [name patterns](#)

```
select(flights, ends_with("time"))
```

```
# A tibble: 336,776 × 5
```

	dep_time	sched_dep_time	arr_time	sched_arr_time	air_time
	<int>	<int>	<int>	<int>	<dbl>
1	517	515	830	819	227
2	533	529	850	830	227
3	542	540	923	850	160
4	544	545	1004	1022	183
5	554	600	812	837	116
6	554	558	740	728	150
7	555	600	913	854	158
8	557	600	709	723	53
9	557	600	838	846	140
10	558	600	753	745	138

SELECTING VARIABLES

Select variables based on multiple name patterns

```
select(flights, c(carrier, ends_with("time"), contains("delay")))
```

```
# A tibble: 336,776 × 8
```

	carrier	dep_time	sched_dep_time	arr_time	sched_arr_time	air_time	dep_delay	arr_delay
	<chr>	<int>	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>
1	UA	517	515	830	819	227	2	11
2	UA	533	529	850	830	227	4	20
3	AA	542	540	923	850	160	2	33
4	B6	544	545	1004	1022	183	-1	-18
5	DL	554	600	812	837	116	-6	-25
6	UA	554	558	740	728	150	-4	12
7	B6	555	600	913	854	158	-5	19
8	EV	557	600	709	723	53	-3	-14
9	B6	557	600	838	846	140	-3	-8
10	AA	558	600	753	745	138	-2	8

VARIABLE PLACEMENT

Sometimes we just want to change the [order of variables](#)

```
select(flights, time_hour, air_time, everything())
```

```
# A tibble: 336,776 × 19
```

	time_hour	air_time	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<dtm>	<dbl>	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013-01-01 05:00:00	227	2013	1	1	517	515	2	830
2	2013-01-01 05:00:00	227	2013	1	1	533	529	4	850
3	2013-01-01 05:00:00	160	2013	1	1	542	540	2	923
4	2013-01-01 05:00:00	183	2013	1	1	544	545	-1	1004
5	2013-01-01 06:00:00	116	2013	1	1	554	600	-6	812
6	2013-01-01 05:00:00	150	2013	1	1	554	558	-4	740
7	2013-01-01 06:00:00	158	2013	1	1	555	600	-5	913
8	2013-01-01 06:00:00	53	2013	1	1	557	600	-3	709
9	2013-01-01 06:00:00	140	2013	1	1	557	600	-3	838
10	2013-01-01 06:00:00	138	2013	1	1	558	600	-2	753

RENAMING VARIABLES

Other times we just want to [rename](#) our variables:

```
rename(flights, ANNOYING = dep_delay)
```

```
# A tibble: 336,776 × 19
```

	year	month	day	dep_time	sched_dep_time	ANNOYING	arr_time	sched_arr_time	arr_delay
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>	<dbl>
1	2013	1	1	517	515	2	830	819	11
2	2013	1	1	533	529	4	850	830	20
3	2013	1	1	542	540	2	923	850	33
4	2013	1	1	544	545	-1	1004	1022	-18
5	2013	1	1	554	600	-6	812	837	-25
6	2013	1	1	554	558	-4	740	728	12
7	2013	1	1	555	600	-5	913	854	19
8	2013	1	1	557	600	-3	709	723	-14
9	2013	1	1	557	600	-3	838	846	-8
10	2013	1	1	558	600	-2	753	745	8

YOUR TURN!

1. What happens if you include the name of a variable multiple times in a **select()** call?
2. What does the **one_of()** function do? Why might it be helpful in conjunction with this vector?

```
vars <- c("MONTH", "month", "day", "dep_delay", "arr_delay")
```

3. Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

```
select(flights, contains("TIME"))
```

SOLUTION

```
# 1. what happens if you call a variable multiple times in select()
```

```
select(flights, month, month)
```

```
# A tibble: 336,776 × 1
```

```
  month
```

```
  <int>
```

```
1     1
```

```
2     1
```

```
3     1
```

```
4     1
```

```
5     1
```

```
6     1
```

```
7     1
```

```
8     1
```

```
9     1
```

SOLUTION

```
# 2. what does one_of() do?
```

```
vars <- c("MONTH", "month", "day", "dep_delay", "arr_delay")
```

```
select(flights, one_of(vars))
```

```
# A tibble: 336,776 × 4
```

	month	day	dep_delay	arr_delay
	<int>	<int>	<dbl>	<dbl>
1	1	1	2	11
2	1	1	4	20
3	1	1	2	33
4	1	1	-1	-18
5	1	1	-6	-25
6	1	1	-4	12
7	1	1	-5	19

SOLUTION

3. Default for select helpers?

?select

```
starts_with(match, ignore.case = TRUE, vars = current_vars())  
ends_with(match, ignore.case = TRUE, vars = current_vars())  
contains(match, ignore.case = TRUE, vars = current_vars())  
matches(match, ignore.case = TRUE, vars = current_vars())
```

mutate

Create new variables with functions of existing variables



REDUCE OUR DATA

Lets work with a smaller data set

```
flights_sml <- select(flights,  
  year:day,  
  ends_with("delay"),  
  distance,  
  air_time  
)
```

```
flights_sml
```

```
# A tibble: 336,776 × 7
```

	year	month	day	dep_delay	arr_delay	distance	air_time
	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>
1	2013	1	1	2	11	1400	227
2	2013	1	1	4	20	1416	227
3	2013	1	1	2	33	1089	160
4	2013	1	1	-1	-18	1576	183
5	2013	1	1	-6	-25	762	116
6	2013	1	1	-4	12	719	150

CREATE NEW VARIABLES

`mutate()` creates new variables with functions of existing variables:

```
mutate(flights_sml,  
  gain = arr_delay - dep_delay,  
  speed = distance / air_time * 60  
)
```

```
# A tibble: 336,776 × 9
```

	year	month	day	dep_delay	arr_delay	distance	air_time	gain	speed
	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2013	1	1	2	11	1400	227	9	370.0441
2	2013	1	1	4	20	1416	227	16	374.2731
3	2013	1	1	2	33	1089	160	31	408.3750
4	2013	1	1	-1	-18	1576	183	-17	516.7213
5	2013	1	1	-6	-25	762	116	-19	394.1379
6	2013	1	1	-4	12	719	150	16	287.6000
7	2013	1	1	-5	19	1065	158	24	404.4304
8	2013	1	1	-3	-14	229	53	-11	259.2453
9	2013	1	1	-3	-8	944	140	-5	404.5714
10	2013	1	1	3	8	733	138	10	318.6053

CREATE NEW VARIABLES

Note: you can create **variables** based on **columns** that you've just created:

```
mutate(flights_sml,  
  gain = arr_delay - dep_delay,  
  hours = air_time / 60,  
  gain_per_hour = gain / hours  
)
```

```
# A tibble: 336,776 × 10
```

	year	month	day	dep_delay	arr_delay	distance	air_time	gain	hours	gain_per_hour
	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2013	1	1	2	11	1400	227	9	3.7833333	2.378855
2	2013	1	1	4	20	1416	227	16	3.7833333	4.229075
3	2013	1	1	2	33	1089	160	31	2.6666667	11.625000
4	2013	1	1	-1	-18	1576	183	-17	3.0500000	-5.573770
5	2013	1	1	-6	-25	762	116	-19	1.9333333	-9.827586
6	2013	1	1	-4	12	719	150	16	2.5000000	6.400000
7	2013	1	1	-5	19	1065	158	24	2.6333333	9.113924
8	2013	1	1	-3	-14	229	53	-11	0.8833333	-12.452830
9	2013	1	1	2	8	944	140	5	2.2222222	2.142857

CREATE NEW VARIABLES

If you only want to keep the **new variables** use `transmute()`:

```
transmute(flights,  
  gain = arr_delay - dep_delay,  
  hours = air_time / 60,  
  gain_per_hour = gain / hours  
)
```

```
# A tibble: 336,776 × 3
```

	gain	hours	gain_per_hour
	<dbl>	<dbl>	<dbl>
1	9	3.78333333	2.378855
2	16	3.78333333	4.229075
3	31	2.66666667	11.625000
4	-17	3.05000000	-5.573770
5	-19	1.93333333	-9.827586
6	16	2.50000000	6.400000
7	24	2.63333333	9.113924
8	-11	0.88333333	-12.452830
9	5	2.22222222	2.250000

MANY USEFUL CREATION FUNCTIONS

There are a wide variety of functions you can use with `mutate()`

Functions	Description
<code>+, -, *, /, ^</code>	arithmetic
<code>x / sum(x)</code>	arithmetic w/aggregate functions
<code>%/%, %%</code>	modular arithmetic
<code>log, exp, sqrt</code>	transformations
<code>lag, lead</code>	offsets
<code>cumsum, cumprod, cum...</code>	cum/rolling aggregates
<code>>, >=, <, <=, !=, ==</code>	logical comparisons
<code>min_rank, dense_rank,</code>	ranking
<code>between</code>	are values between a and b?
<code>ntile</code>	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,
  normalized_delay = dep_delay / (mean(dep_delay, na.rm = TRUE)))
# A tibble: 336,776 × 1
  normalized_delay
      <dbl>
1      0.15823949
2      0.31647898
3      0.15823949
4     -0.07911974
5     -0.47471846
6     -0.31647898
7     -0.39559872
8     -0.23735923
9     -0.23735923
10     -0.15823949
# ... with 336,766 more rows
```

Functions	Description
<code>+, -, *, /, ^</code>	arithmetic
<code>x / sum(x)</code>	arithmetic w/aggregate functions
<code>%/%, %%</code>	modular arithmetic
<code>log, exp, sqrt</code>	transformations
<code>lag, lead</code>	offsets
<code>cumsum, cumprod, cum...</code>	cum/rolling aggregates
<code>>, >=, <, <=, !=, ==</code>	logical comparisons
<code>min_rank, dense_rank,</code>	ranking
<code>between</code>	are values between a and b?
<code>ntile</code>	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,
  log_air_time = log2(air_time),
  exp_delay = exp(dep_delay))
# A tibble: 336,776 × 2
  log_air_time      exp_delay
    <dbl>          <dbl>
1    7.826548    7.389056099
2    7.826548  54.598150033
3    7.321928    7.389056099
4    7.515700    0.367879441
5    6.857981    0.002478752
6    7.228819    0.018315639
7    7.303781    0.006737947
8    5.727920    0.049787068
9    7.129283    0.049787068
10   7.108524    0.135335283
# with 336,766 more rows
```

Functions	Description
<code>+, -, *, /, ^</code>	arithmetic
<code>x / sum(x)</code>	arithmetic w/aggregate functions
<code>%/%, %%</code>	modular arithmetic
<code>log, exp, sqrt</code>	transformations
<code>lag, lead</code>	offsets
<code>cumsum, cumprod, cum...</code>	cum/rolling aggregates
<code>>, >=, <, <=, !=, ==</code>	logical comparisons
<code>min_rank, dense_rank,</code>	ranking
<code>between</code>	are values between a and b?
<code>ntile</code>	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,
  dep_delay = dep_delay,
  lag_delay = lag(dep_delay),
  sum_delay = cumsum(dep_delay))
# A tibble: 336,776 × 3
  dep_delay lag_delay sum_delay
    <dbl>      <dbl>      <dbl>
1         2         NA         2
2         4         2         6
3         2         4         8
4        -1         2         7
5        -6        -1         1
6        -4        -6        -3
7        -5        -4        -8
8        -3        -5       -11
9        -3        -3       -14
10       -2        -3       -16
```

Functions	Description
<code>+, -, *, /, ^</code>	arithmetic
<code>x / sum(x)</code>	arithmetic w/aggregate functions
<code>%/%, %%</code>	modular arithmetic
<code>log, exp, sqrt</code>	transformations
<code>lag, lead</code>	offsets
<code>cumsum, cumprod, cum...</code>	cum/rolling aggregates
<code>>, >=, <, <=, !=, ==</code>	logical comparisons
<code>min_rank, dense_rank,</code>	ranking
<code>between</code>	are values between a and b?
<code>ntile</code>	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,  
  arr_delay = arr_delay,  
  bucket = ntile(arr_delay, 10))
```

A tibble: 336,776 × 2

	arr_delay	bucket
	<dbl>	<int>
1	11	8
2	20	8
3	33	9
4	-18	3
5	-25	2
6	12	8
7	19	8
8	-14	3
9	-8	5
10	8	7

Functions	Description
+, -, *, /, ^	arithmetic
x / sum(x)	arithmetic w/aggregate functions
%/%, %%	modular arithmetic
log, exp, sqrt	transformations
lag, lead	offsets
cumsum, cumprod, cum...	cum/rolling aggregates
>, >=, <, <=, !=, ==	logical comparisons
min_rank, dense_rank,	ranking
between	are values between a and b?
ntile	bin values into buckets

YOUR TURN!

1. Create a new variable **distance_km** that converts distance in miles to kilometers
2. Create a **time_per_km** variable based on **air_time** and **distance_km**.

SOLUTION

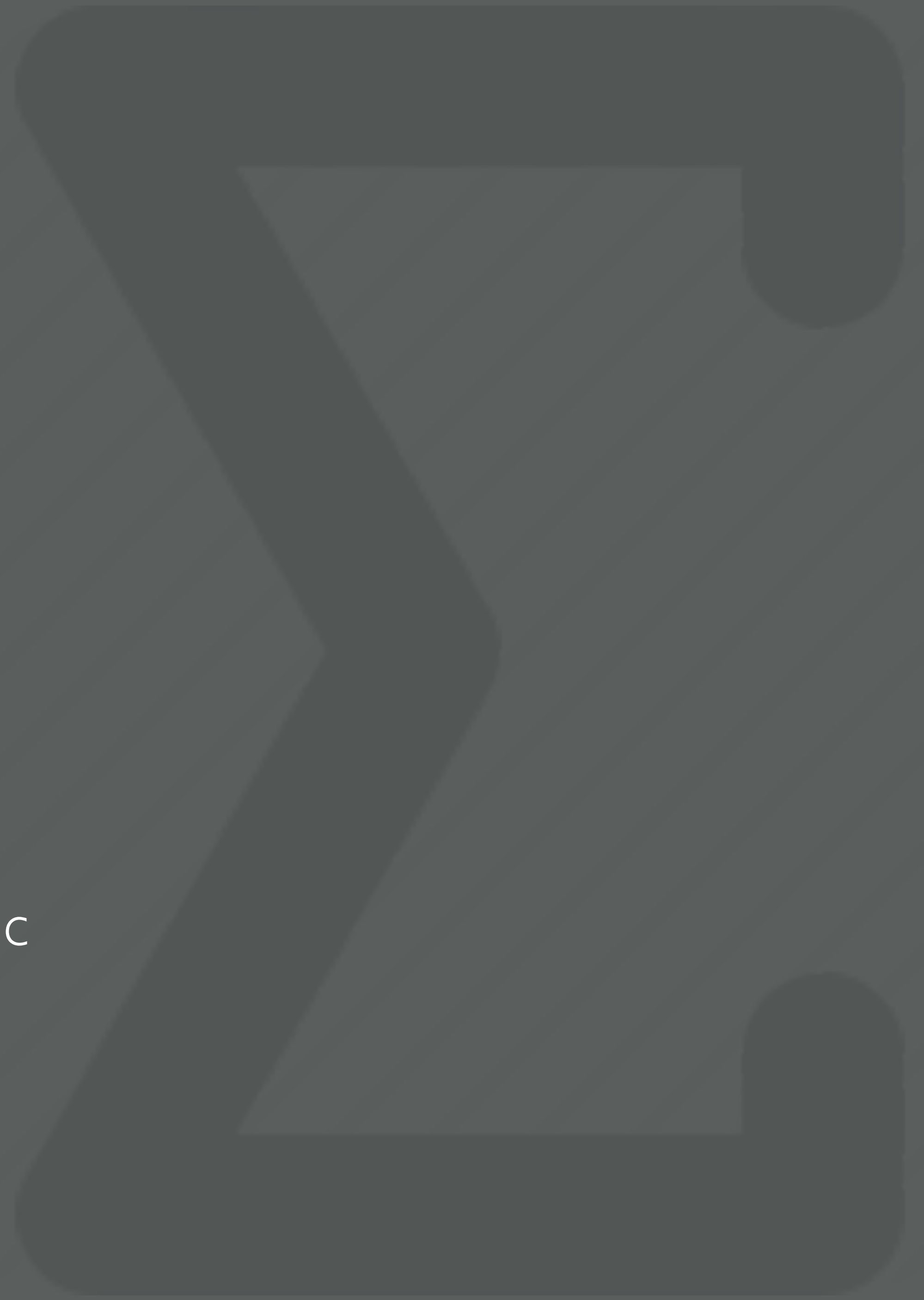
```
transmute(flights,  
          distance_km = distance * 1.60934,  
          time_per_km = air_time / distance_km)
```

```
# A tibble: 336,776 × 2
```

	distance_km	time_per_km
	<dbl>	<dbl>
1	2253.0760	0.10075115
2	2278.8254	0.09961272
3	1752.5713	0.09129443
4	2536.3198	0.07215178
5	1226.3171	0.09459218
6	1157.1155	0.12963270
7	1713.9471	0.09218488
8	368.5389	0.14381116
9	1510.3170	0.09315274

summarise

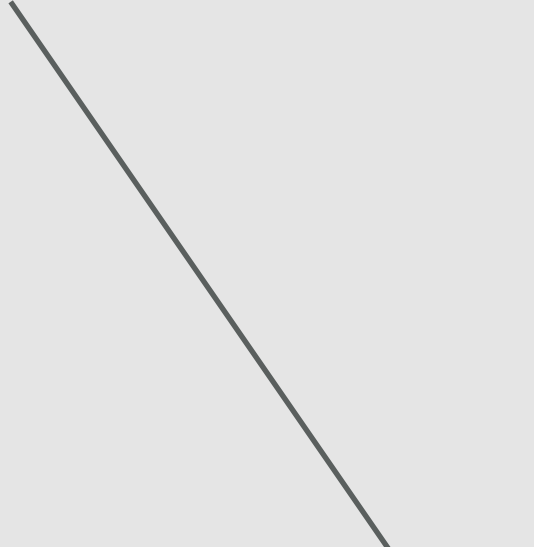
Collapse many values down to a single summary statistic



SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarise(flights, dep_delay_mean = mean(dep_delay, na.rm = TRUE))  
# A tibble: 1 × 1  
  dep_delay_mean  
      <dbl>  
1      12.63907
```



Important, try this without `na.rm = TRUE` and see what happens. Why does this happen?

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarise(flights,  
          dep_delay_mean = mean(dep_delay, na.rm = TRUE),  
          dep_delay_sd = sd(dep_delay, na.rm = TRUE))  
  
# A tibble: 1 × 2  
  dep_delay_mean dep_delay_sd  
    <dbl>         <dbl>  
1    12.63907     40.21006
```

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarise(flights,  
  dep_delay_mean = mean(dep_delay, na.rm = TRUE),  
  dep_delay_sd = sd(dep_delay, na.rm = TRUE),  
  n = n())
```

```
# A tibble: 1 × 3
```

	dep_delay_mean	dep_delay_sd	n
	<dbl>	<dbl>	<int>
1	12.63907	40.21006	336776

SUMMARY FUNCTIONS

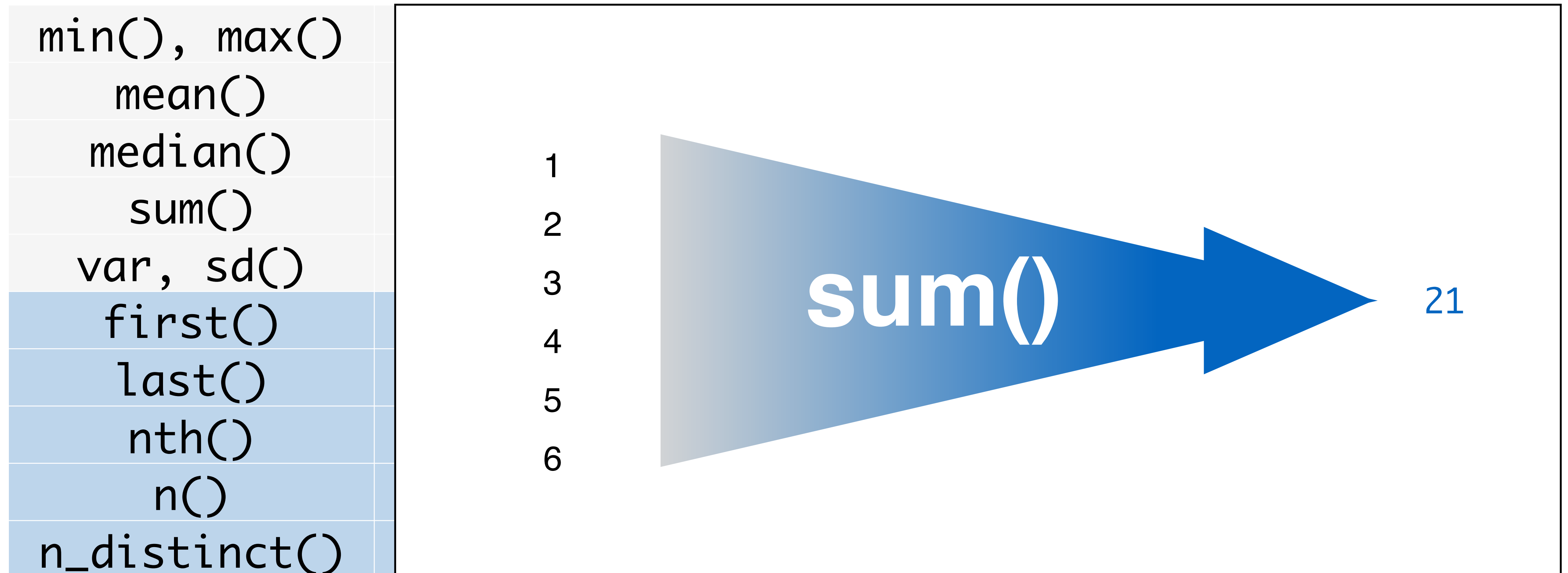
* All take a vector of values and return a single value

** Blue functions come in **dplyr**

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

SUMMARY FUNCTIONS

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



SUMMARIZING GROUPED DATA

Summary statistics become more powerful when we can compare [groups](#):

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

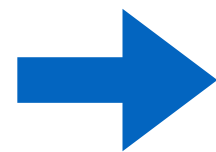
Source: local data frame [365 x 4]

Groups: year, month [?]

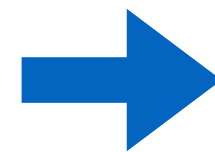
	year	month	day	delay
	<int>	<int>	<int>	<dbl>
1	2013	1	1	11.548926
2	2013	1	2	13.858824
3	2013	1	3	10.987832
4	2013	1	4	8.951595
5	2013	1	5	5.732218
6	2013	1	6	7.148014

SUMMARIZING GROUPED DATA

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3



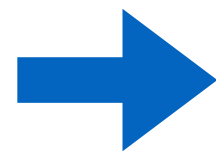
country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3



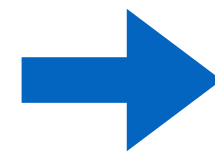
```
group_by(data, country)
```

SUMMARIZING GROUPED DATA

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3



country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

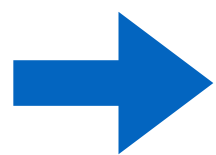


country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

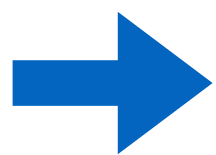
```
group_by(data, country, year)
```

SUMMARIZING GROUPED DATA

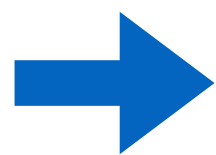
country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3



country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3



country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3



country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

ungroup(data)

YOUR TURN!

1. Which carrier had the largest mean departure delay?
Smallest?
2. Which carrier had the largest difference between their
max and min departure delay?
3. Which month has the largest variance for arrival delays?

SOLUTION

```
# Which carrier had the largest mean departure delay?  Smallest?
```

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

```
summarise(by_carrier, delay = mean(dep_delay, na.rm = TRUE))
```

```
# Which carrier had the largest difference between their max and min departure delay?
```

```
summarise(by_carrier,
```

```
  max = max(dep_delay, na.rm = TRUE),
```

```
  min = min(dep_delay, na.rm = TRUE),
```

```
  delta = max - min)
```

```
# Which month has the largest variance for arrival delays?
```

```
by_month <- group_by(flights, month)
```

```
summarise(by_month, delay = sd(arr_delay, na.rm = TRUE))
```

pipe operator

Chaining functions together with the pipe operator



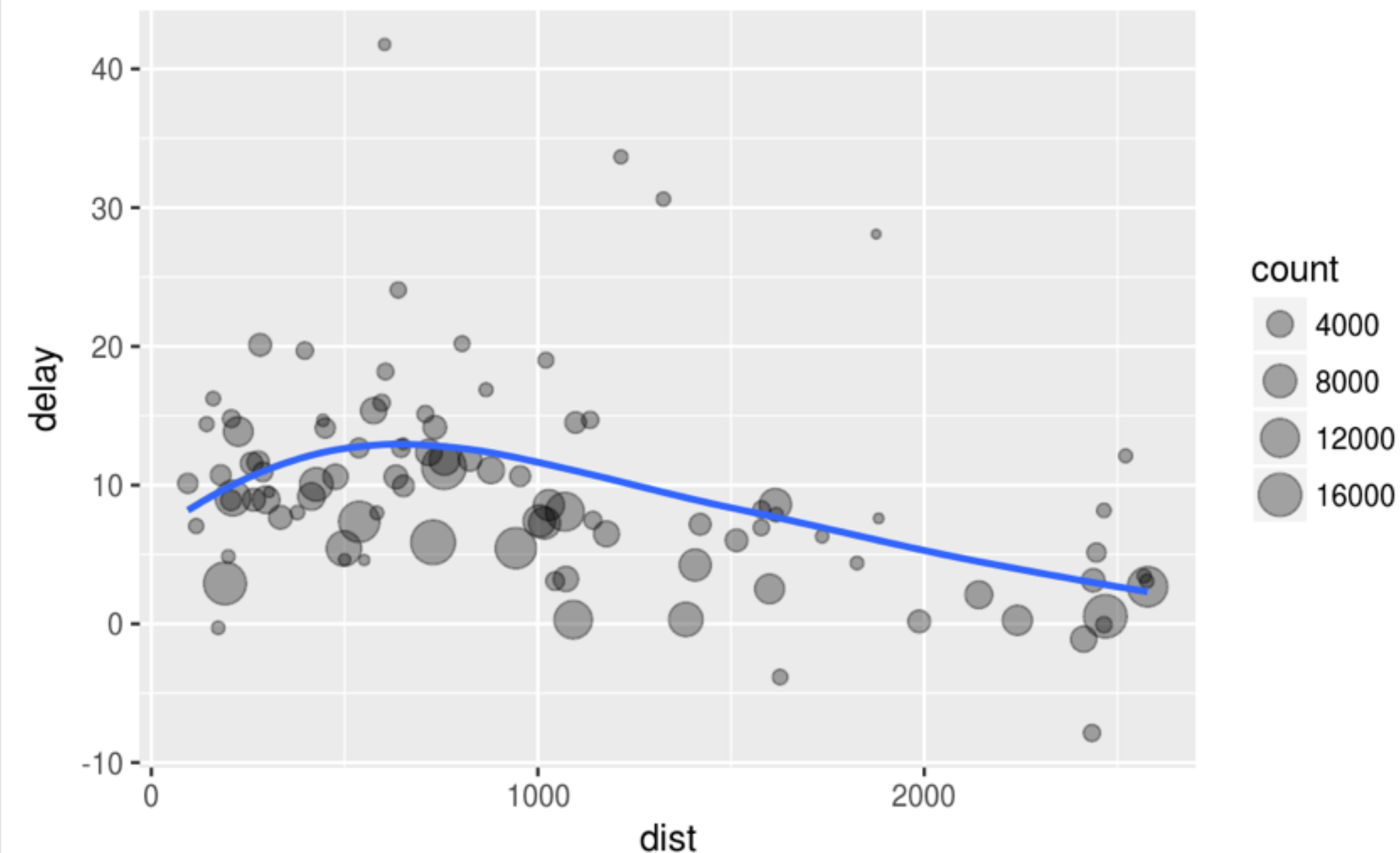
STREAMLINING OUR ANALYSIS

Imagine that we want to explore the relationship between the distance and average delay for each location.

Using what you know about dplyr and ggplot, you might write code like this:

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")

ggplot(data = delay, mapping = aes(x = dist, y =
delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```



STREAMLINING OUR ANALYSIS

Imagine that we want to explore the relationship between the distance and average delay for each location.

Using what you know about dplyr and ggplot, you might write code like this:

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")

ggplot(data = delay, mapping = aes(x = dist, y =
delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

This code does four things:

1. ?
2. ?
3. ?
4. ?

STREAMLINING OUR ANALYSIS

Imagine that we want to explore the relationship between the distance and average delay for each location.

Using what you know about dplyr and ggplot, you might write code like this:

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")

ggplot(data = delay, mapping = aes(x = dist, y =
  delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

This code does four things:

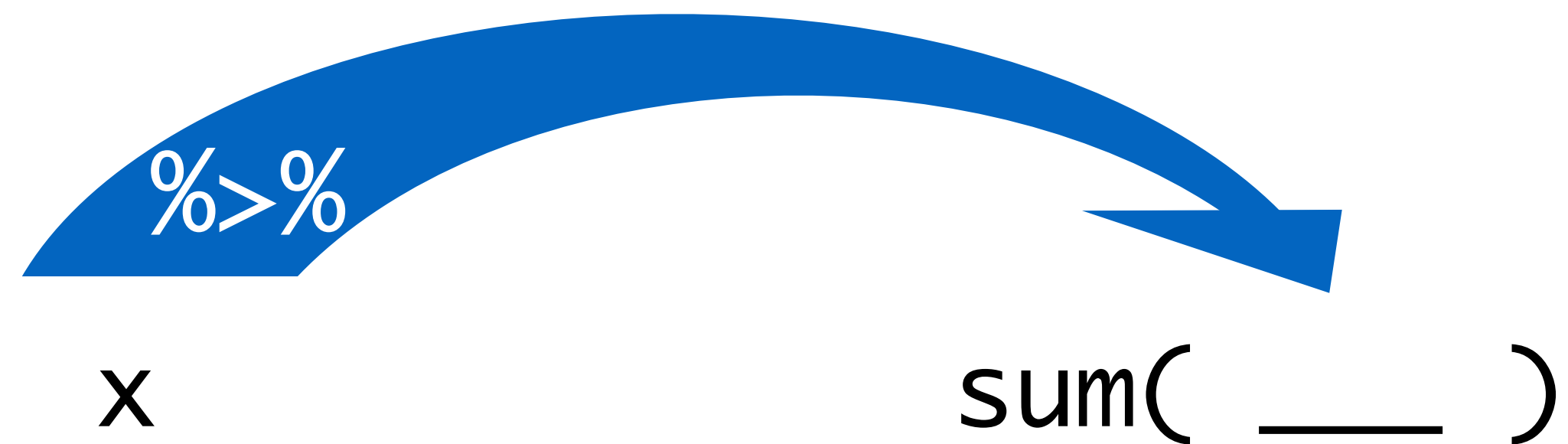
1. grouping by destination
2. summarizing count, distance, and delay
3. filtering out low counts and Honolulu
4. creating a plot

STREAMLINING OUR ANALYSIS

We can streamline our code to make it more *efficient* and *legible*

```
library(magrittr)
x <- 1:15
sum(x)
x %>% sum()
```

These do the
same thing
Try it!



STREAMLINING OUR ANALYSIS

Lets re-write our code using the pipe (`%>%`) operator:

```
flights %>%  
  group_by(dest) %>%  
  summarise(count = n(),  
            dist = mean(distance, na.rm = TRUE),  
            delay = mean(arr_delay, na.rm = TRUE)) %>%  
  filter(count > 20, dest != "HNL") %>%  
  ggplot(aes(x = dist, y = delay)) +  
    geom_point(aes(size = count), alpha = 1/3) +  
    geom_smooth(se = FALSE)
```

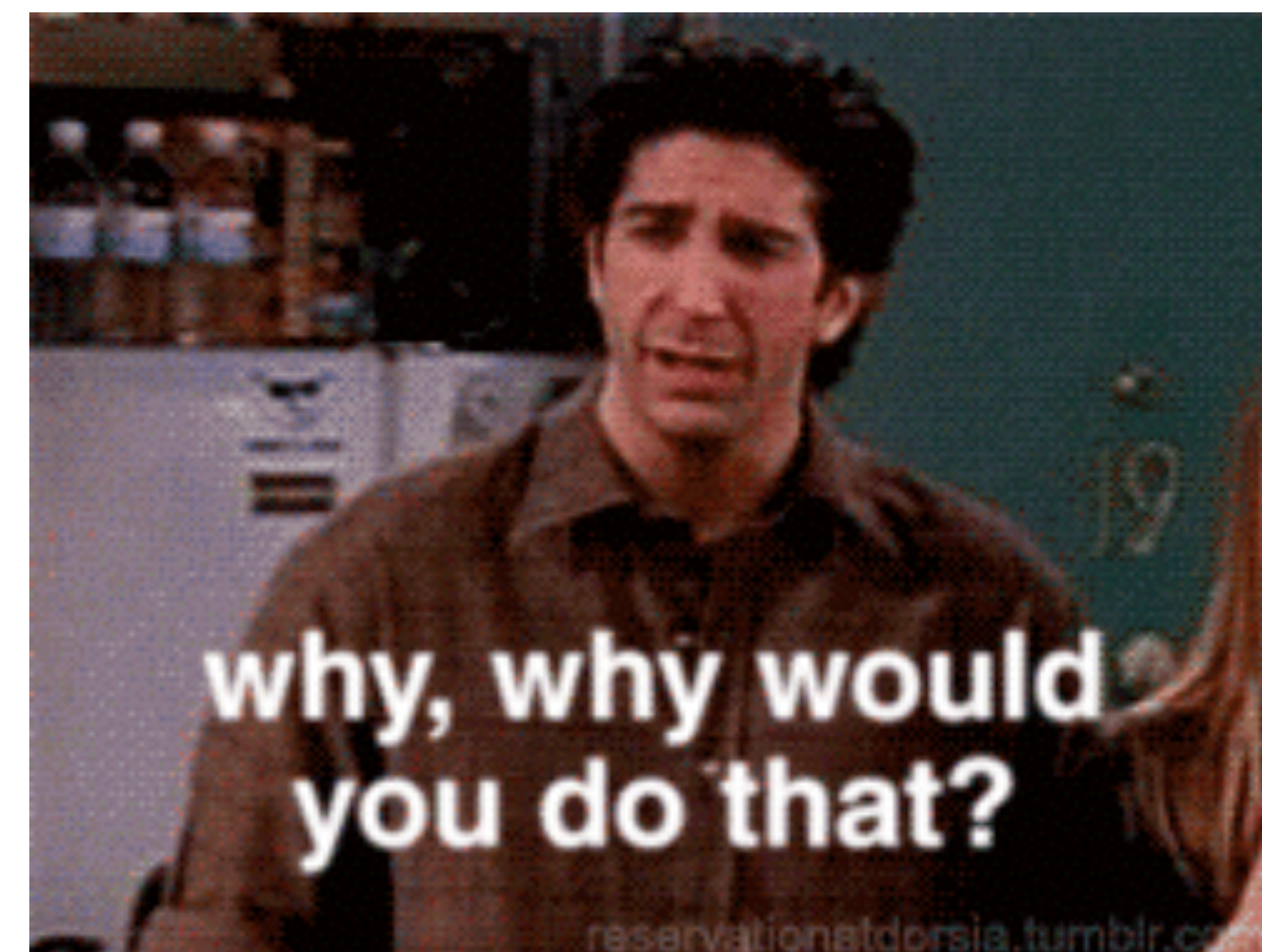
This code does four things in a very *efficient* & *readable* manner:

1. grouping by destination
2. summarizing count, distance, and delay
3. filtering out low counts and Honolulu
4. creating a plot

STREAMLINING OUR ANALYSIS

Lets re-write our code using the pipe (`%>%`) operator:

```
flights %>%  
  group_by(dest) %>%  
  summarise(count = n(),  
            dist = mean(distance, na.rm = TRUE),  
            delay = mean(arr_delay, na.rm = TRUE)) %>%  
  filter(count > 20, dest != "HNL") %>%  
  ggplot(aes(x = dist, y = delay)) +  
    geom_point(aes(size = count), alpha = 1/3) +  
    geom_smooth(se = FALSE)
```



YOUR TURN!

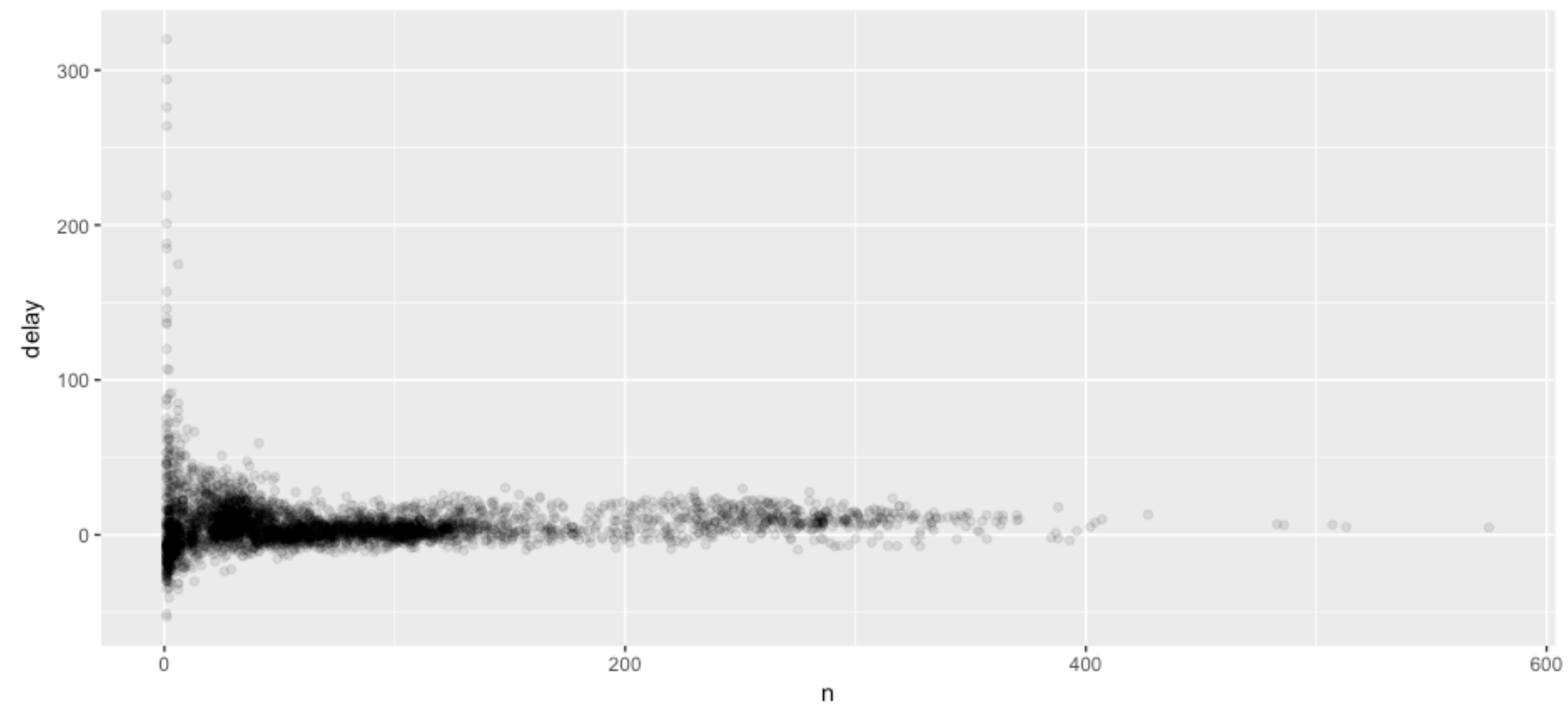
Using the pipe operator follow these steps with the flight data:

1. filter out missing values that exist in the tail number variable
2. group the data by tail number
3. calculate the average arrival delay and the number of observations by tail number
4. plot average arrival delay vs. number of observations

What do these findings tell you?

SOLUTION

```
flights %>%  
  filter(!is.na(tailnum)) %>%  
  group_by(tailnum) %>%  
  summarise(delay = mean(arr_delay, na.rm = TRUE),  
            n = n()) %>%  
  ggplot(aes(n, delay)) +  
    geom_point(alpha = .1)
```



YOUR TURN!

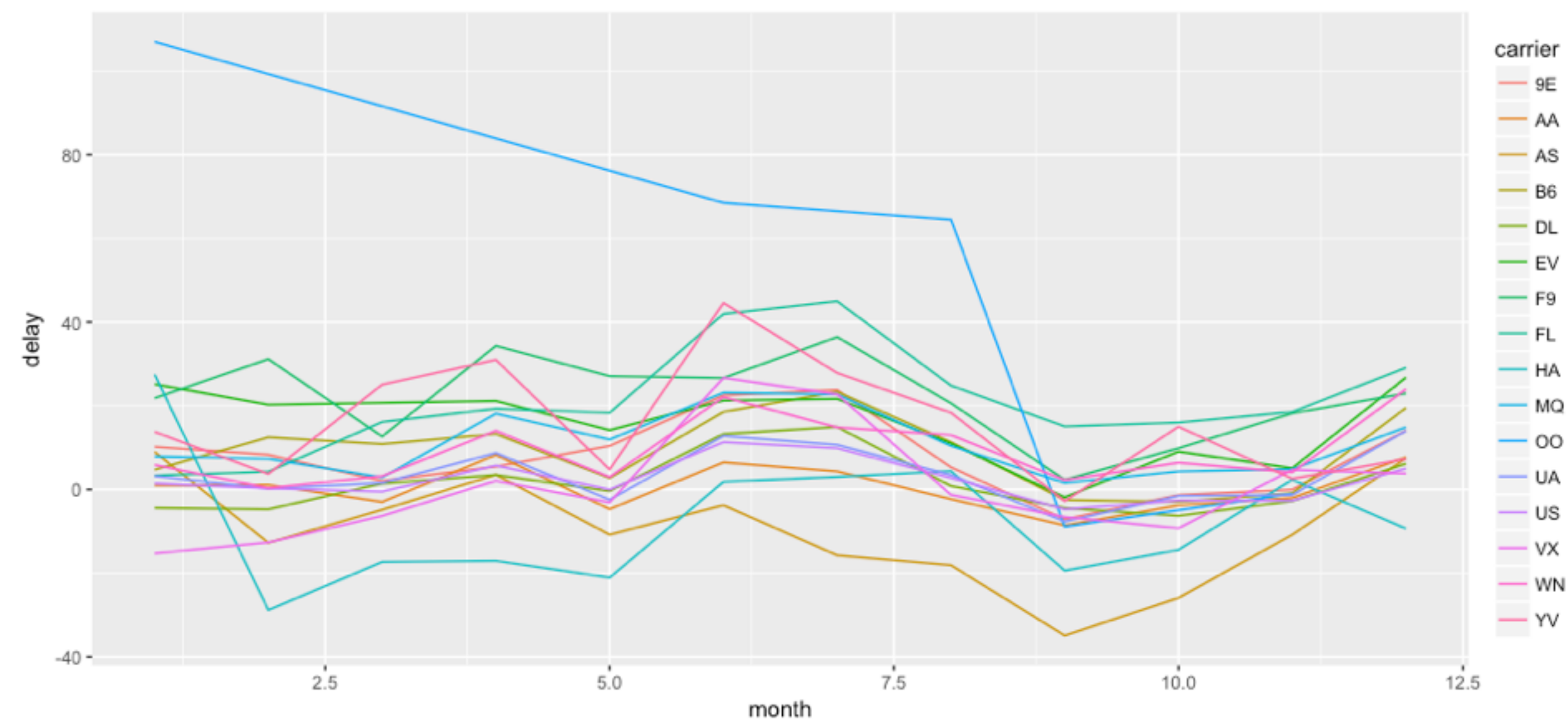
Using the pipe operator follow these steps with the flight data:

1. group by month and carrier
2. compute the average arrival delay
3. plot a line chart with month on the x-axis, average delay on the y axis, and each line colored coded by carrier (or you could facet by carrier)

What do these findings tell you?

SOLUTION

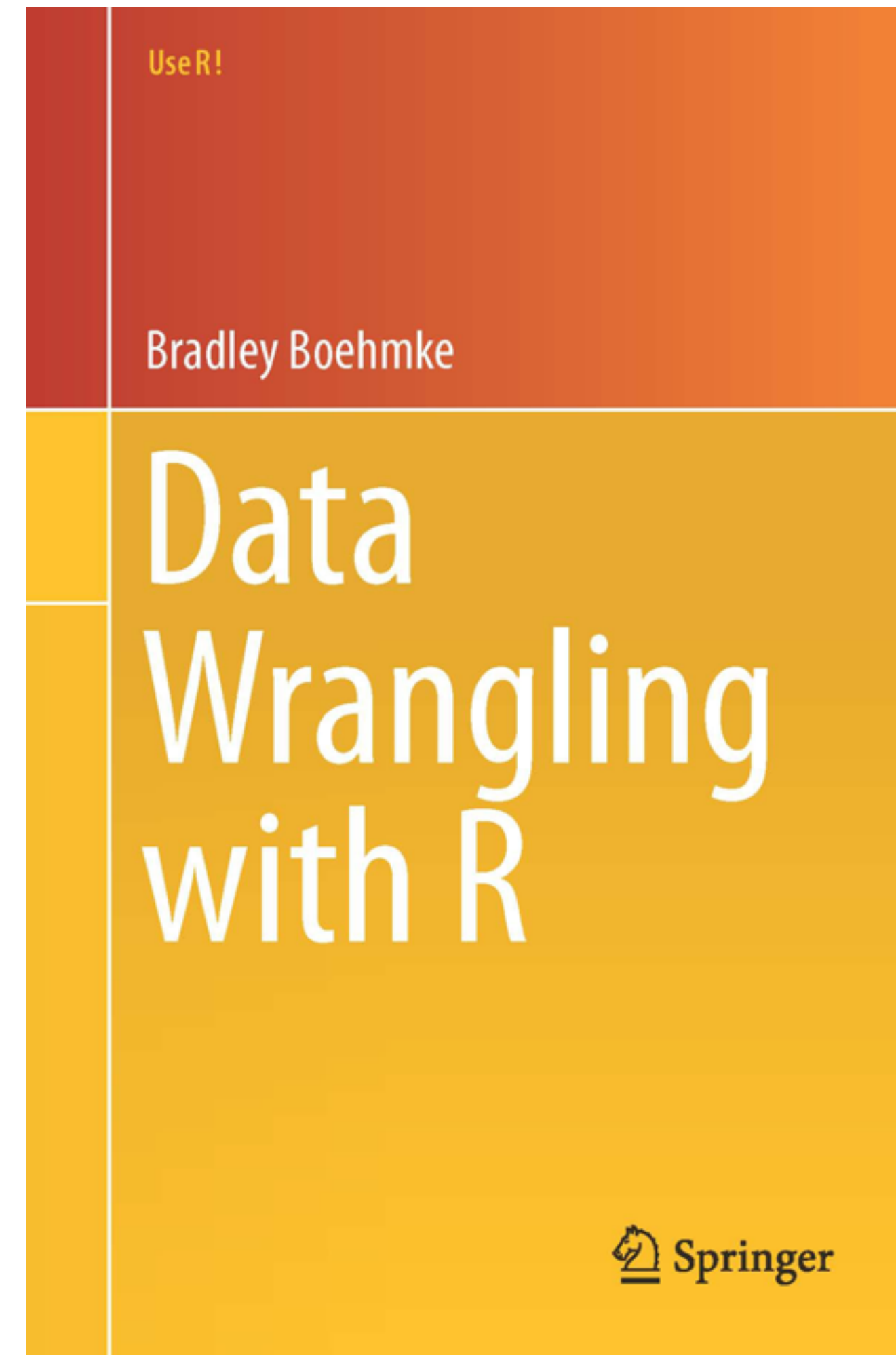
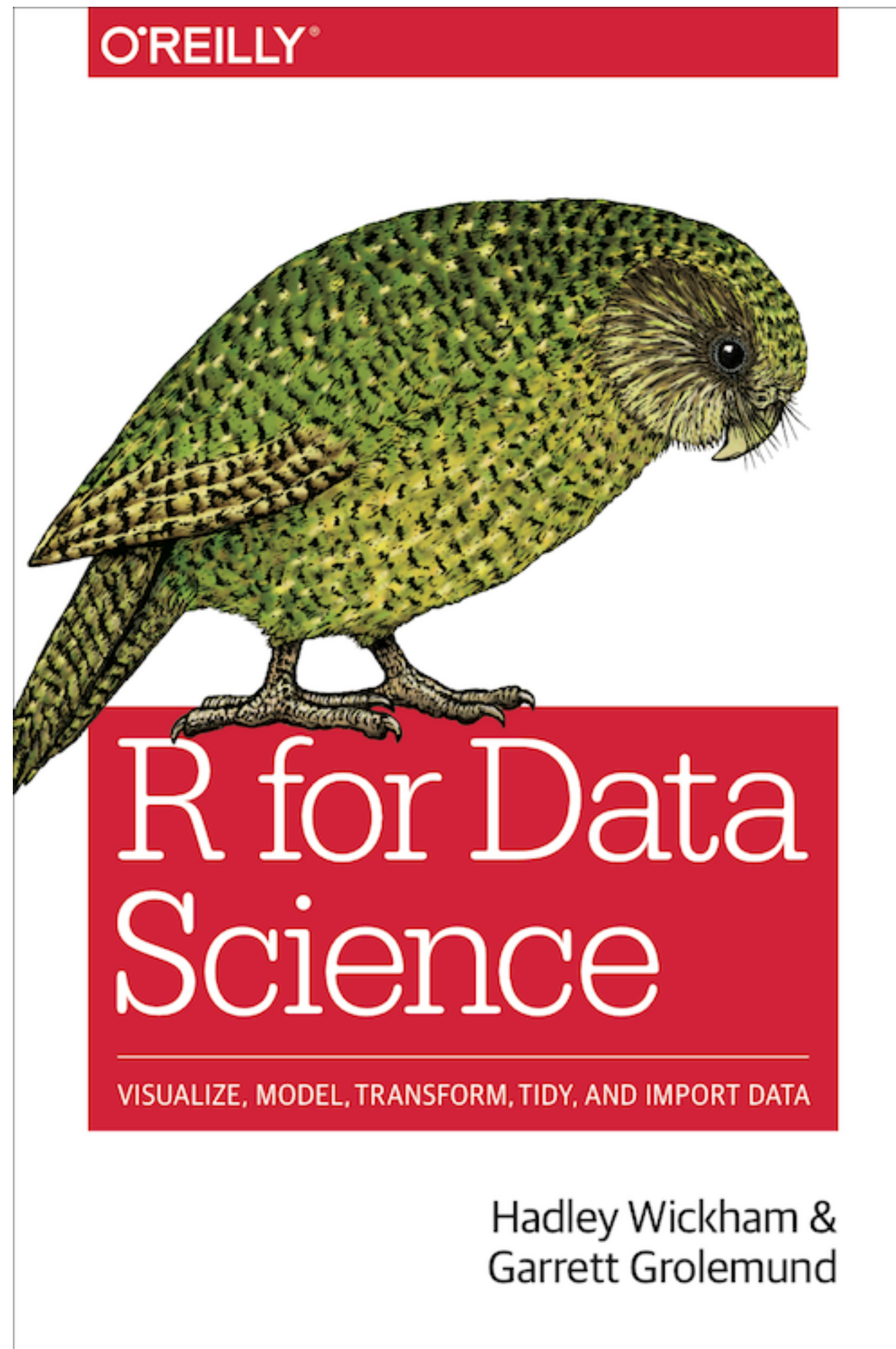
```
flights %>%  
  group_by(month, carrier) %>%  
  summarise(delay = mean(arr_delay, na.rm = TRUE)) %>%  
  ggplot(aes(month, delay, color = carrier)) +  
    geom_line()
```



SO LITTLE TIME!



LEARN MORE



WHAT TO REMEMBER



FUNCTIONS TO REMEMBER

Operator/Function	Description
<code>filter</code>	pick observations based on their values
<code>>, >=, <, <=, !=, ==</code>	comparison operators
<code>is.na</code>	identify missing values
<code>arrange</code>	re-order rows
<code>desc</code>	order in descending order
<code>select</code>	select variables
<code>starts_with, ends_with, contains, etc.</code>	select variables based on patterns
<code>rename</code>	rename select variables
<code>mutate, transmute</code>	create new variables
<code>summarise</code>	summarize data
<code>group_by, ungroup</code>	group/ungroup based on categorical variables
<code>%>%</code>	pipe operator to chain together functions