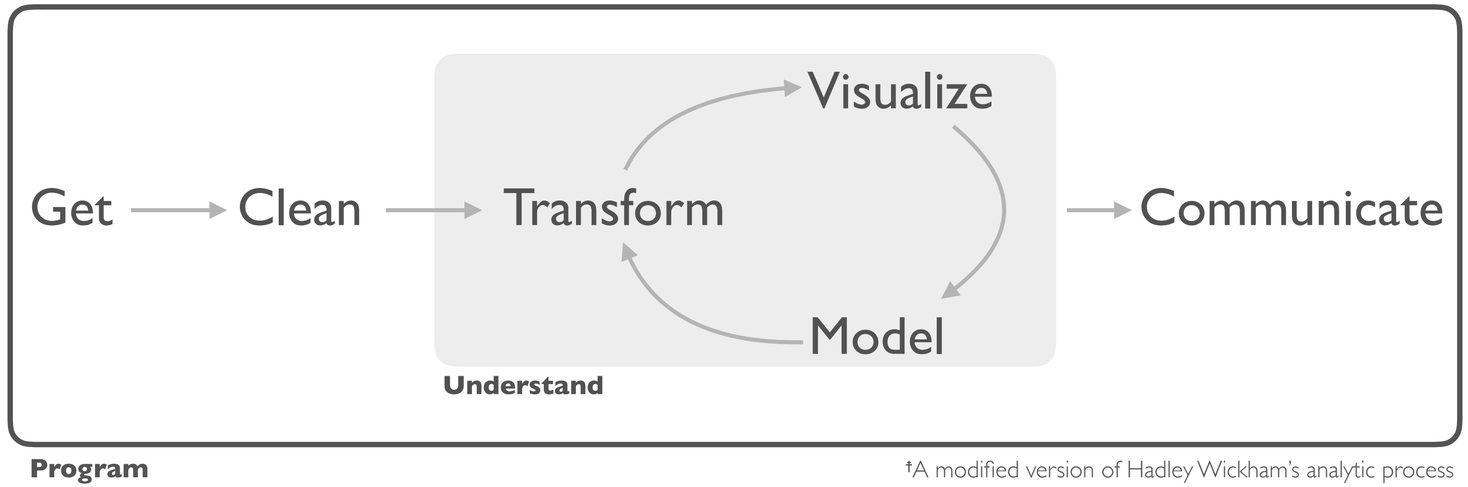
TIDYVERSE



†A modified version of Hadley Wickham's analytic process

WHAT IS TIDYVERSE?

PACKAGE PREREQUISITE

```
library(nycflights13)
library(tidyverse)
── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 ──
✔ ggplot2 2.2.1          ✔ purrr   0.2.4
✔ tibble  1.4.1          ✔ dplyr   0.7.4
✔ tidyr   0.7.2.9000     ✔ stringr 1.2.0
✔ readr   1.1.1          ✔ forcats 0.2.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

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 <dttm>
```

DATA PREREQUISITE

```
mpg
#> # A tibble: 234 × 11
#>   manufacturer model displ  year   cyl      trans   drv   cty   hwy    fl
#>          <chr> <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>
#> 1         audi    a4   1.8  1999     4   auto(l5)     f    18    29     p
#> 2         audi    a4   1.8  1999     4 manual(m5)     f    21    29     p
#> 3         audi    a4   2.0  2008     4 manual(m6)     f    20    31     p
#> 4         audi    a4   2.0  2008     4   auto(av)     f    21    30     p
#> 5         audi    a4   2.8  1999     6   auto(l5)     f    16    26     p
#> 6         audi    a4   2.8  1999     6 manual(m5)     f    18    26     p
#> # ... with 228 more rows, and 1 more variables: class <chr>
```

dplyr

Making data manipulation & transformation easy

`dplyr`

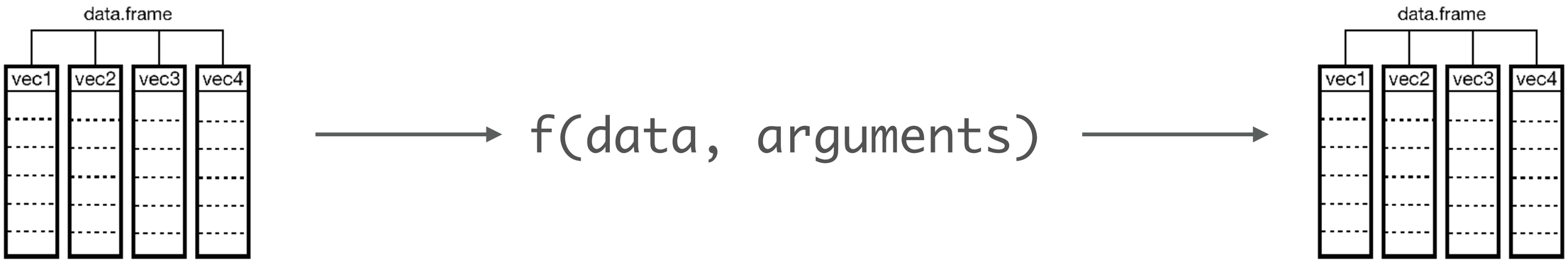
You are going to learn the five key **`dplyr`** functions that allow you to solve the vast majority of your data manipulation challenges:

- `filter:` pick observations based on values
- `arrange:` reorder data
- `select:` pick variables
- `mutate:` create new variables
- `summarize:` 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



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
```

COMPARISON

Try these operations

```
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))
```

`?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

YOUR TURN!

1. Import the CustomerData.csv file.
2. Filter `Gender` for female customers only.
3. Filter `Gender` for female customers whose `Age` are greater than 45 years old ***and*** live in `Region` 3.
4. Filter for female customers that are greater than 45 years old ***or*** live in region 3.

SOLUTION

```
# 1: import the data
customer <- read_csv("data/CustomerData.csv")

# 2: filter for female customers only
filter(customer, Gender == "Female")

# 3: filter for female customers that are greater than 45 years old and live in region 3
filter(customer, Gender == "Female", Age > 45, Region == 3)

# 4: filter for female customers that are greater than 45 years old or live in region 3
filter(customer, Gender == "Female", Age > 45 | Region == 3)
```

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
```

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

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
```

YOUR TURN!

1. Using the customer data, select all columns between `CustomerID` and `Gender`.

2. Now select all columns other than those between columns between `CustomerID` and `Gender`.

3. Select `CustomerID` and all variables that contain the word "`Card`".

SOLUTION

```
# 1. select all variables between CustomerID and Gender
select(customer, CustomerID:Gender)

# 2. select all variables except for those between CustomerID and Gender
select(customer, -(CustomerID:Gender))

#3. select CustomerID and all variables that contain the word “Card”
select(customer, CustomerID, contains(“Card”))
```

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

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>
```

YOUR TURN!

1. Select the variables `CustomerID`, `Region`, `Gender`, `Age`, `HHIncome`, `CardspendMonth` and save this as `sub_cust`.

2. Order `sub_cust` data by `Age` and `CardSpendMonth` (ascending order)

3. Order `sub_cust` data by `Age` (oldest to youngest) and `CardSpendMonth` (least to most)

SOLUTION

```
# 1: select variables
sub_cust <- select(customer, CustomerID, Region, Gender, Age, HHIncome, CardSpendMonth)

# 2: Order sub_cust data by Age and CardSpendMonth (ascending order)
arrange(customer, Age, CardSpendMonth)

# 3: Order sub_cust data by Age (oldest to youngest) and CardSpendMonth (least to most)
arrange(customer, desc(Age), CardSpendMonth)
```

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

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
```

YOUR TURN!

1. With **`sub_cust`**, create a *ratio* variable that computes the ratio of *CardSpendMonth* to *HHIncome*

2. Create two variables:
   i. *ratio1* = *CardSpendMonth / HHIncome*
   ii. *ratio2* = *CardSpendMonth / Age*

SOLUTION

```
#1: create a ratio variable that computes the ratio of CardSpendMonth to HHIncome
mutate(sub_cust, ratio = CardSpendMonth / HHIncome)

#2: create 2 variables:
#        i) ratio1 = CardSpendMonth / HHIncome
#       ii) ratio2 = CardSpendMonth / Age
mutate(sub_cust,
  ratio1 = CardSpendMonth / HHIncome,
  ratio2 = CardSpendMonth / Age)
```

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

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

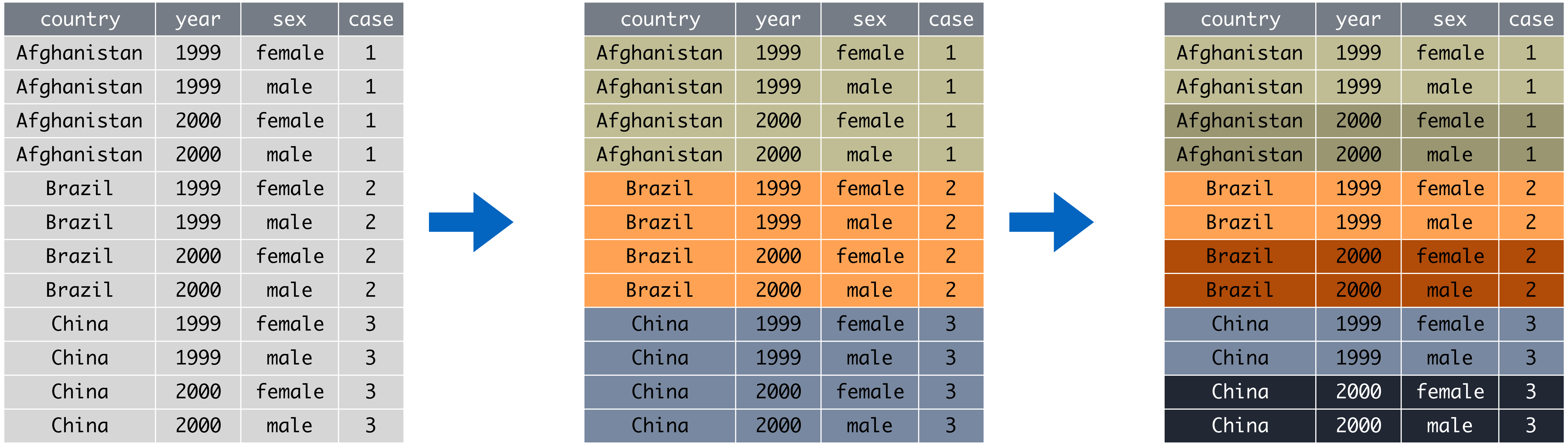
Important, try this without `na.rm = TRUE` and see what happens. Why does this happen?

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarize(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
```

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

Summary statistics become more powerful when we can compare groups:

```
by_day <- group_by(flights, month)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
Source: local data frame [365 x 4]
Groups: year, month [?]

# A tibble: 12 x 2
   month delay
   <int> <dbl>
 1     1 10.0
 2     2 10.8
 3     3 13.2
 4     4 13.9
 5     5 13.0
 6     6 20.8
 7     7 21.7
```

YOUR TURN!

1. In our **`sub_cust`** data, compute the average CardSpendMonth across all customers.

2. Now compute the average CardSpendMonth for each gender.

3. Now compute the average CardSpendMonth for each gender and region. Which gender and region have the highest average spend?

SOLUTION

```
#1: Avg spend across all customers
summarize(sub_cust, Avg_spend = mean(CardSpendMonth, na.rm = TRUE)

#2: Now compute the average CardSpendMonth for each gender.
by_gender <- group_by(sub_cust, Gender)
summarize(by_gender, Avg_spend = mean(CardSpendMonth, na.rm = TRUE))

#3: Now compute the average CardSpendMonth for each gender and region.
#   Which gender and region have the highest average spend?
by_gdr_rgn <- group_by(sub_cust, Gender, Region)
avg_gdr_rgn <- summarize(by_gdr_rgn, Avg_spend = mean(CardSpendMonth, na.rm = TRUE))
arrange(avg_gdr_rgn, desc(Avg_spend))

   Gender Region Avg_spend
    <chr>  <int>     <dbl>
1    Male      3  3692.818
2    Male      5  3617.054
```

STREAMLINING OUR ANALYSIS

Going back to our last problem, our code was doing three things:

1. grouping by gender and region
2. summarizing average spend
3. sorting spend by greatest to least

```
by_gdr_rgn <- group_by(sub_cust, Gender, Region)

avg_gdr_rgn <- summarize(by_gdr_rgn, Avg_spend = mean(CardSpendMonth, na.rm = TRUE))

arrange(avg_gdr_rgn, desc(Avg_spend))
```

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

```
sub_cust %>%
  group_by(Gender, Region) %>%
  summarize(Avg_spend = mean(CardSpendMonth, na.rm = TRUE)) %>%
  arrange(desc(Avg_spend))

  Gender Region Avg_spend
      <chr>  <int>     <dbl>
  1    Male      3  3692.818
  2    Male      5  3617.054
  3    Male      4  3535.671
```

YOUR TURN!

Using the pipe operator follow these steps with the **`customer`** data:

1. filter for *male* customers only
2. create a new variable: *ratio* = *CardSpendMonth / HHIncome*
3. group this data by *age*
4. compute the mean of the new *ratio* variable by *age*
5. sort this output to find the *age* with the highest *ratio* of expenditures to income.

SOLUTION

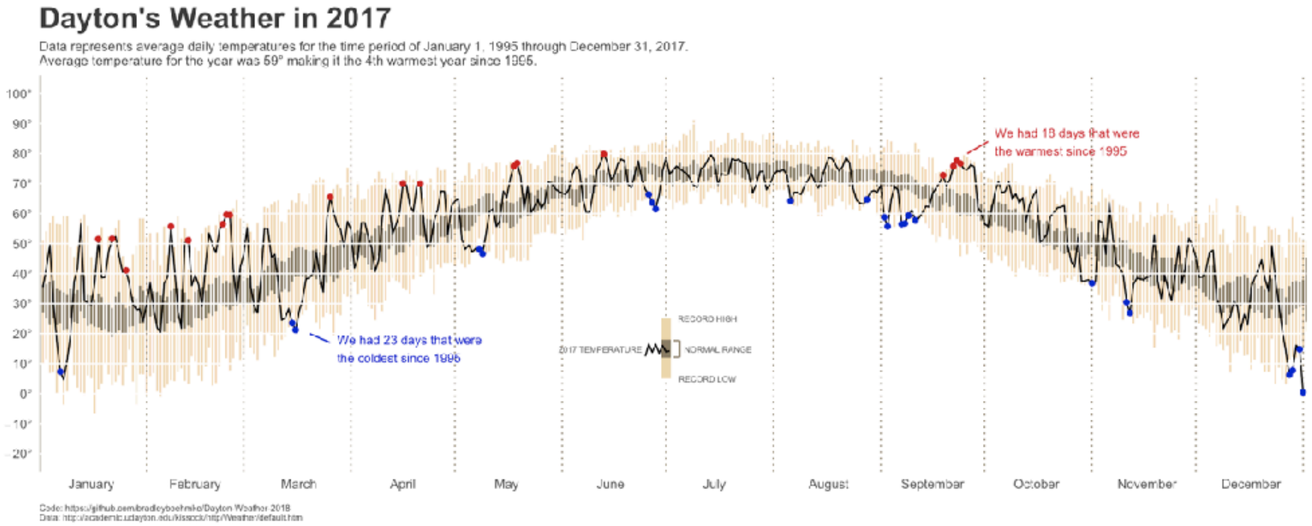
```
customer %>%
  filter(Gender == “Male") %>%
  mutate(ratio = CardSpendMonth / HHIncome) %>%
  group_by(Age) %>%
  summarize(Avg_ratio = mean(ratio, na.rm = TRUE)) %>%
  arrange(desc(Avg_ratio))

     Age Avg_ratio
   <int>     <dbl>
1     20 0.1470240
2     18 0.1452089
3     79 0.1440063
4     19 0.1425964
5     24 0.1363957
6     75 0.1296193
```

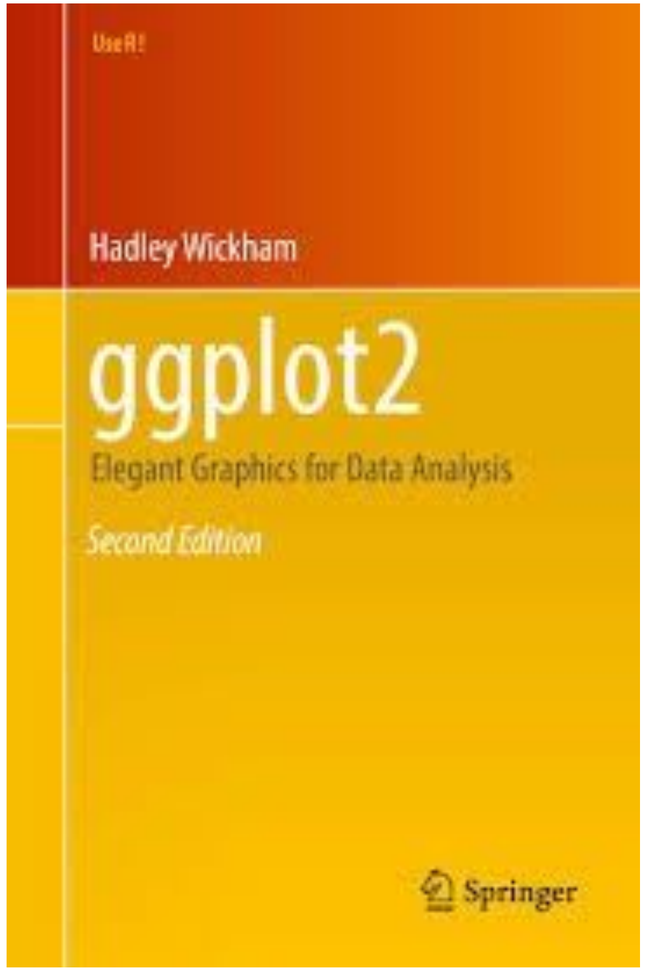
ggplot

A grammar of graphics

ggplot2

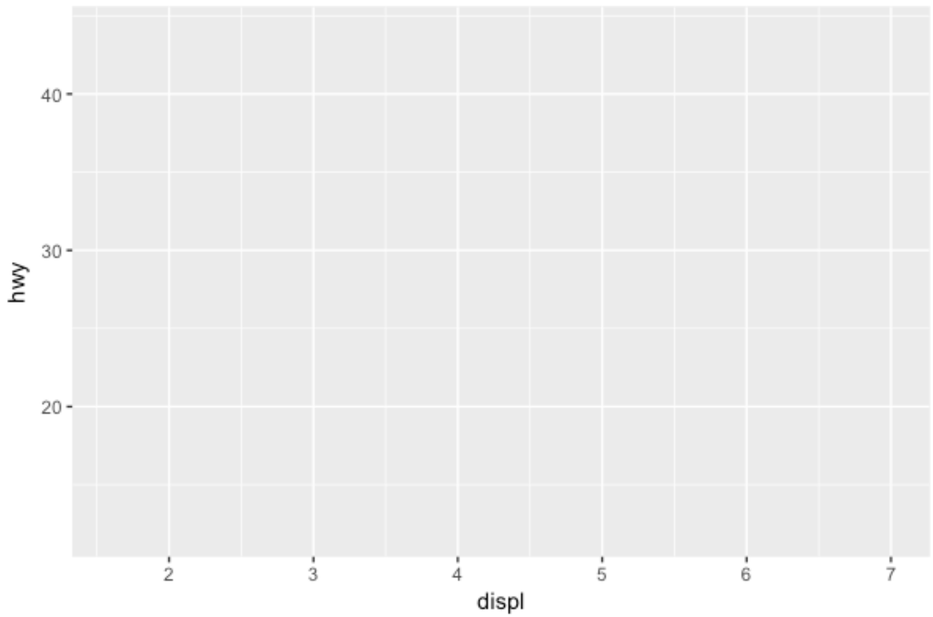
- R has several systems for making graphs
- `ggplot2` is the most elegant and versatile
- Implements the grammar of graphics theory behind data visualization





LET'S CREATE OUR "CANVAS"

```
# left
ggplot(data = mpg)

# right
ggplot(data = mpg, aes(x = displ, y = hwy))
```

LETS ADD "GEOMS"

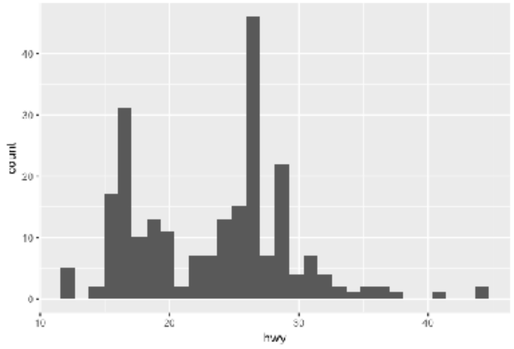
- We display data with geometric shapes
- ~ 30 built-in geoms (with many more offered by other pkgs)

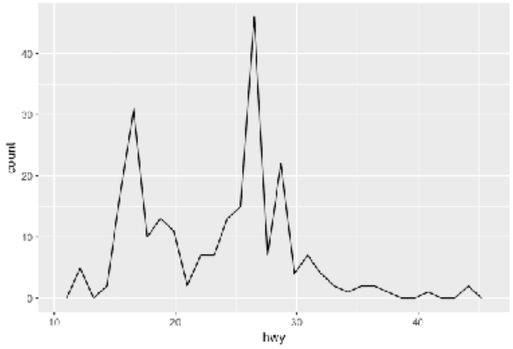
Type **`geom_`** **+** *tab in the console*

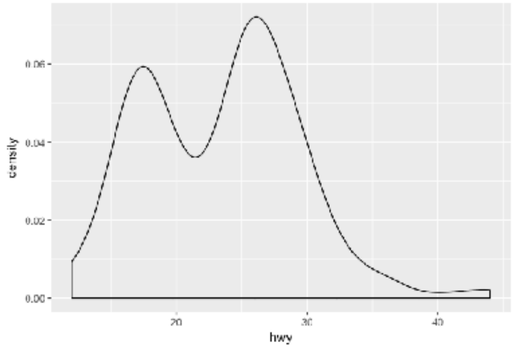
`geom_abline`	`geom_histogram`
`geom_bar`	`geom_jitter`
`geom_bin2d`	`geom_label`
`geom_blank`	`geom_map`
`geom_boxplot`	`geom_path`
`geom_contour`	`geom_point`
`geom_count`	`geom_polygon`
`geom_hex`	`geom_quantile`
`geom_crossbar`	`geom_raster`
`geom_density`	`geom_ribbon`
`geom_density_2d`	`geom_rug`
`geom_dotplot`	`geom_segment`
`geom_errorharh`	`geom_smooth`
`geom_freqpoly`	`geom_violin`

UNIVARIATE GEOMS

```
ggplot(data = mpg, aes(x = hwy)) +
  geom_histogram()

ggplot(data = mpg, aes(x = hwy)) +
  geom_freqpoly()

ggplot(data = mpg, aes(x = hwy)) +
  geom_density()
```



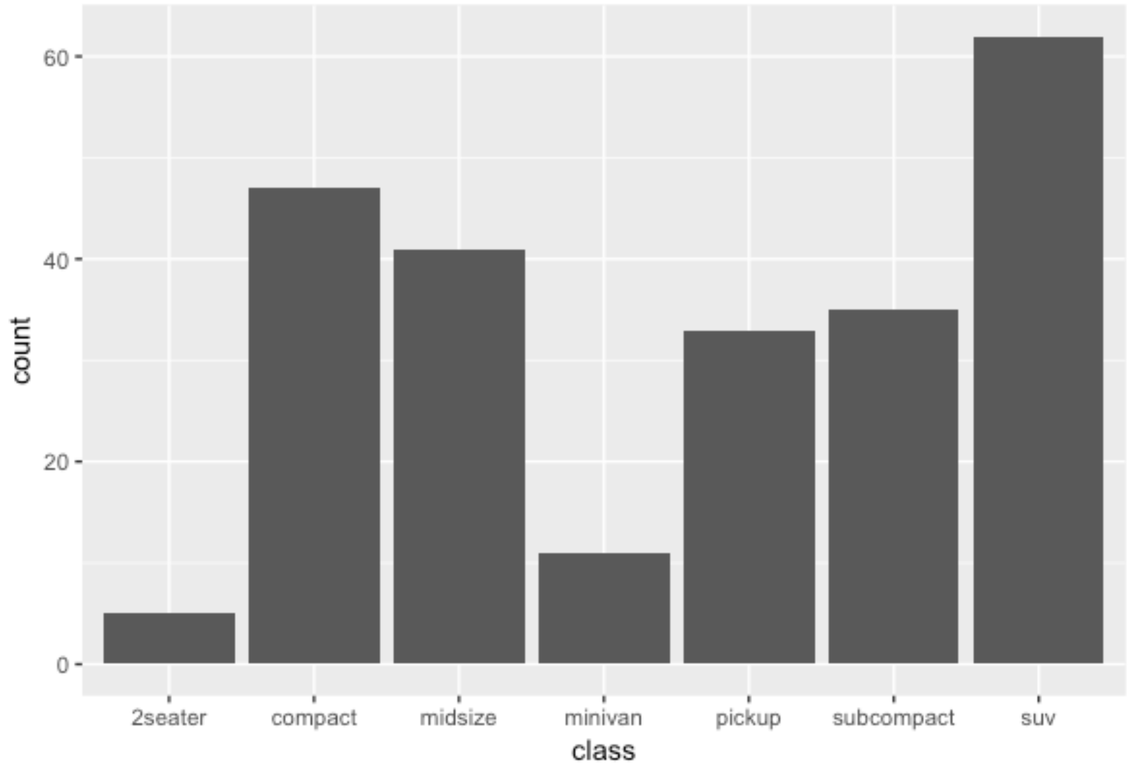




geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorharh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

UNIVARIATE GEOMS

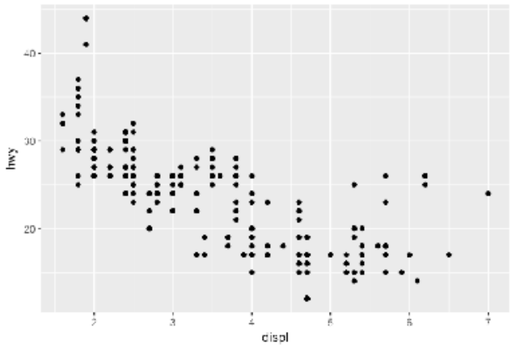
```
ggplot(data = mpg, aes(x = class)) +
  geom_bar()
```

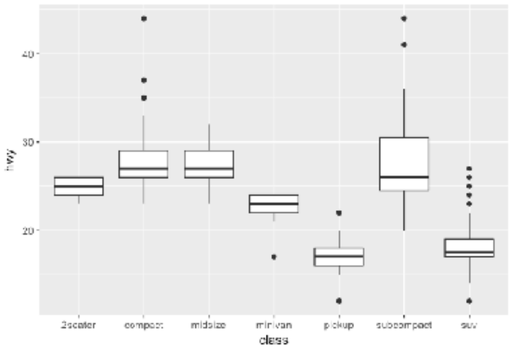


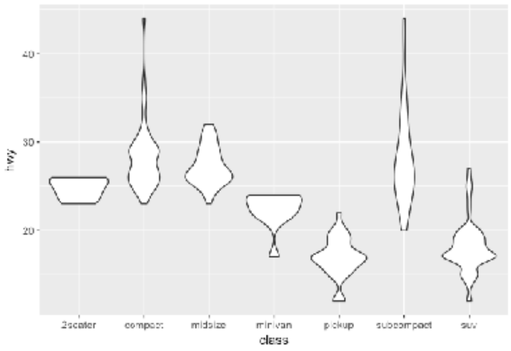
geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorharh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

BIVARIATE GEOMS

```
ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point()

ggplot(data = mpg, aes(x = class, y = hwy)) +
  geom_boxplot()

ggplot(data = mpg, aes(x = class, y = hwy)) +
  geom_violin()
```







geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorharh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

YOUR TURN!

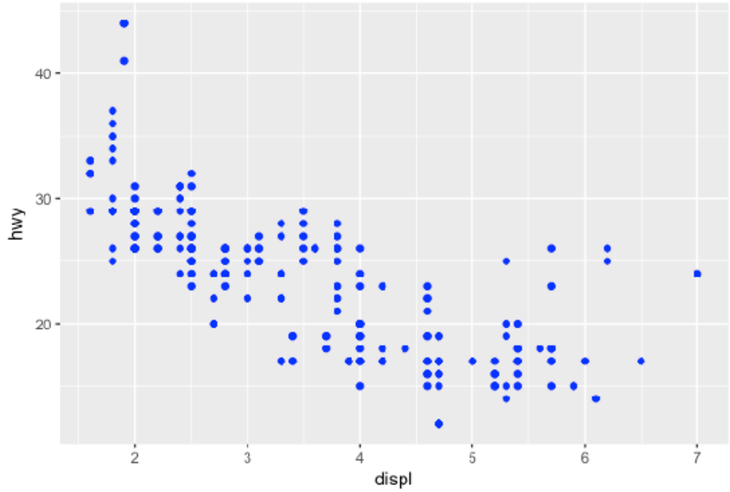
1. Import the CustomerData.csv file
2. Create a chart that illustrates the distribution of the `DebtToIncomeRatio` variable.
3. Create a chart that shows the counts for each `JobCategory`
4. Create a scatter plot of `HHIncome` vs `CardSpendMonth`

SOLUTION

```
#1: import data
customer <- read_csv("data/CustomerData.csv")

#2: distribution of DebtToIncomeRatio variable
ggplot(data = customer, aes(x = DebtToIncomeRatio)) +
  geom_histogram()

#3: distribution of JobCategory variable
ggplot(data = customer, aes(x = JobCategory)) +
  geom_bar()

#4: scatter plot for HHIncome vs CardSpendMonth
ggplot(data = customer, aes(x = HHIncome, y = CardSpendMonth)) +
  geom_point()
```

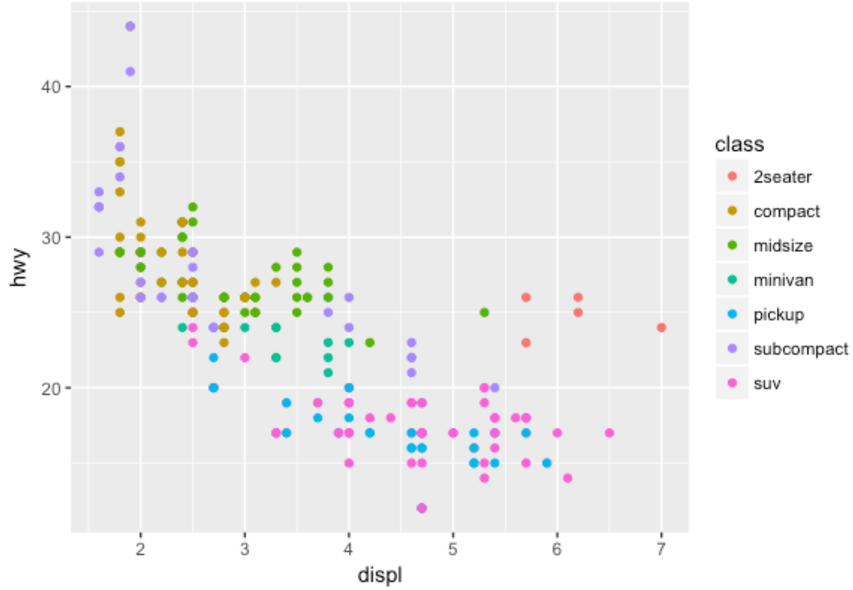
ADDING A 3RD DIMENSION

```
ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point(color = "blue")
```



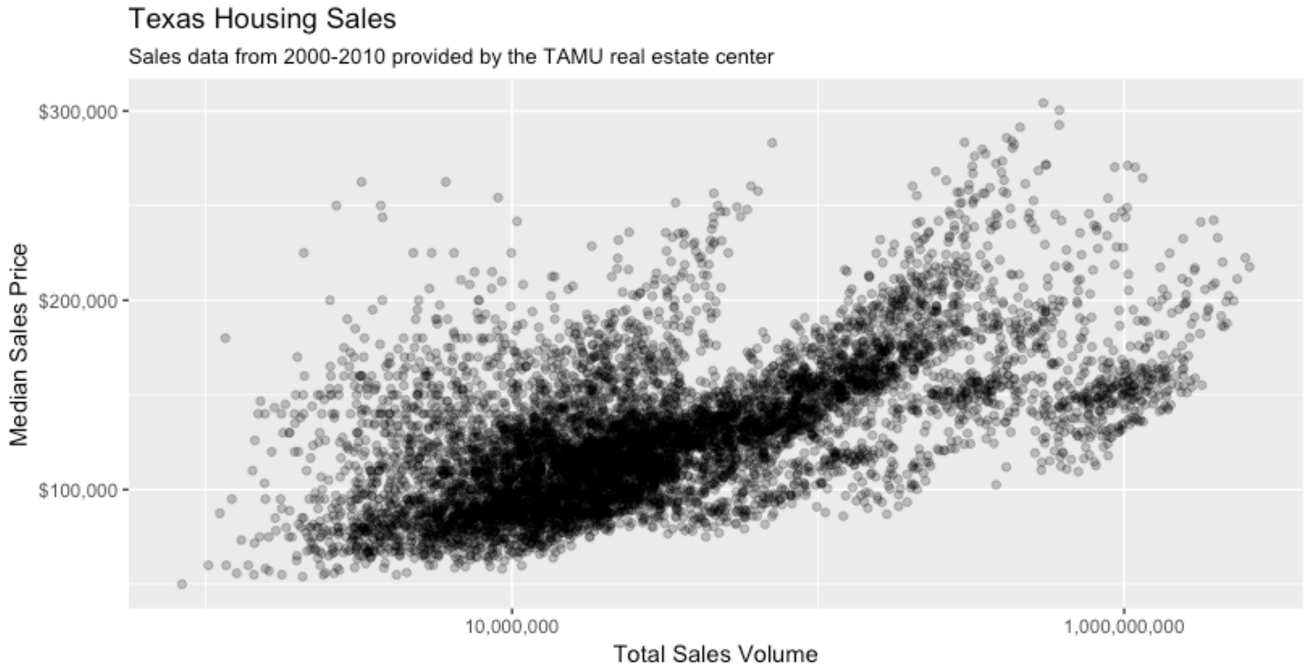
By moving the **`color`** argument to within `aes()`, we can map a 3rd variable to our plot

```
ggplot(data = mpg, aes(x = displ, y = hwy, color = class)) +
  geom_point()
```



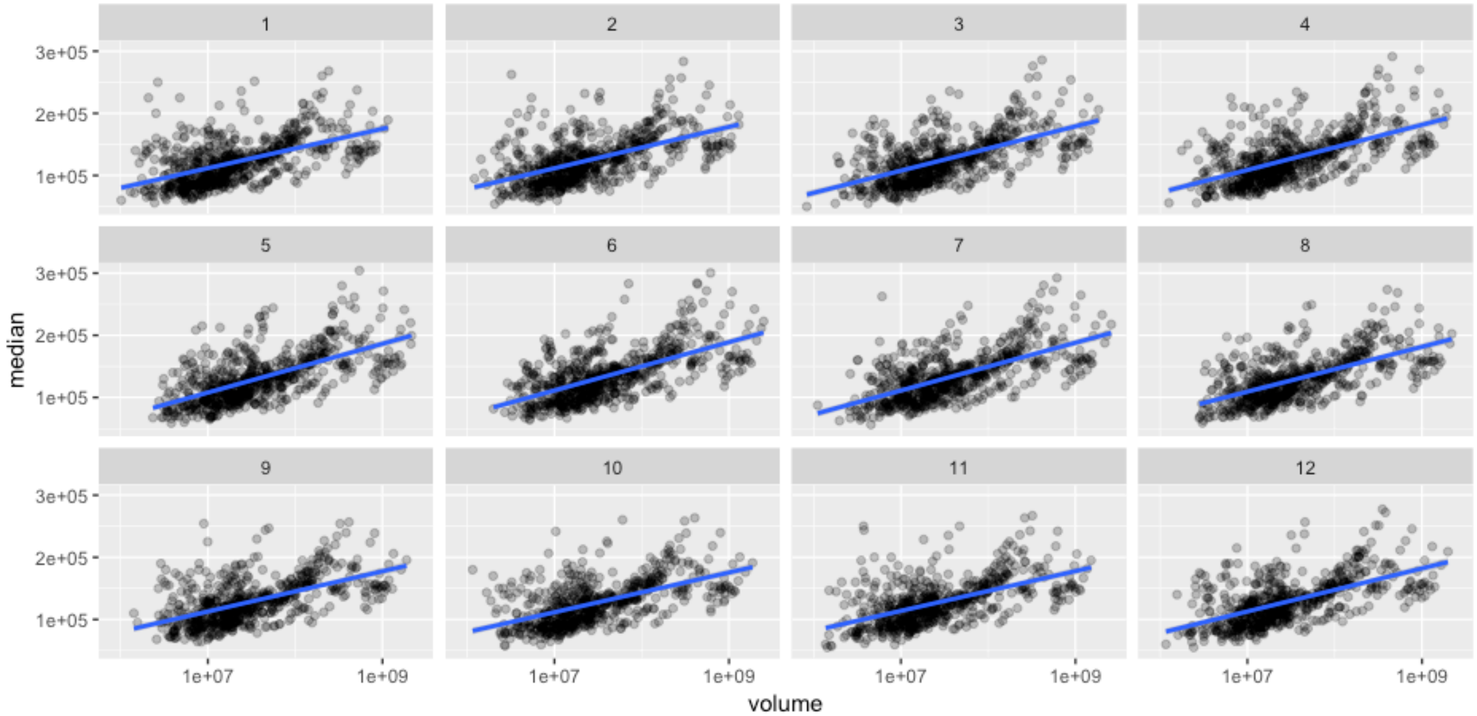
ADDING OTHER ATTRIBUTES

```
ggplot(data = txhousing, aes(x = volume, y = median)) +
  geom_point(alpha = .25) +
  scale_y_continuous(name = "Median Sales Price", labels = scales::dollar) +
  scale_x_log10(name = "Total Sales Volume", labels = scales::comma) +
  ggtitle("Texas Housing Sales",
          subtitle = "Sales data from 2000-2010 provided by the TAMU real estate center")
```



LAYERING & SMALL MULTIPLES HELP DISPLAY PATTERNS

```
ggplot(data = txhousing, aes(x = volume, y = median)) +
  geom_point(alpha = .25)  +
  scale_x_log10() +
  geom_smooth(method = "lm") +
  facet_wrap(~ month)
```



YOUR TURN!

Practice plotting different variables from the **`customer`** data and adding/adjusting multiple attributes