Chapter 2: Exploring Data

Shujia Wong

Department of Statistics, College of Economics
Shenzhen University

Introduction

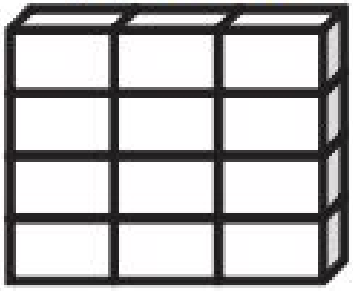
- *Visualisation* is a great place to start with R programming.
- *Data transformation* allows you to select important variables, filter out key observations, create new variables, and compute summaries.
- *Exploratory data analysis* combine visualisation and transformation with your curiosity and scepticism to ask and answer interesting questions about data.

Reference:
Garrett Grolemund, Hadley Wickham (2018): *R for Data Science*, http://r4ds.had.co.nz/.

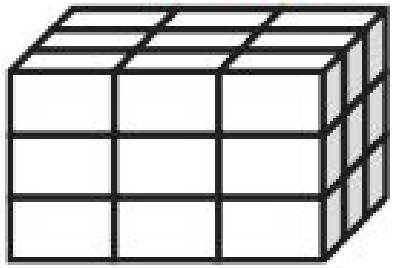
Objects for holding data

(a) Vector

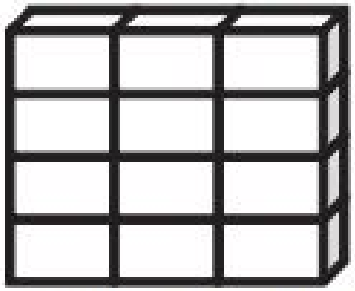
(b) Matrix



(c) Array



(d) Data frame



Columns can be different modes

(e) List

- Vectors
- Arrays
- Data frames
- Lists

Vectors

Vectors are one-dimensional arrays that can hold numeric data, character data, or logical data

- The combine function c() is used to form the vector

```
> a<-c(1, 2, 5, 3, 4) # numeric vector
> b<-c("one","two","three") # character vector
> c<-c(TRUE,TRUE,FALSE) # logical vector
> a

[1] 1 2 5 3 4

> b[2]

[1] "two"
```

Matrices

A **matrix** is a two-dimensional array in which each element has the *same mode* (numeric, character, or logical).

```
mymatrix <- matrix(vector, nrow=m, ncol=n,
                   byrow=logical_value, dimnames=list(
                   char_vector_rownames, char_vector_colna
```

```
> y<-matrix(0:9, nrow=2, ncol=5);y

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    2    4    6    8
[2,]    1    3    5    7    9

> y[,1]; y[2,]

[1] 0 1
[1] 1 3 5 7 9
```

Matrices

```
> cells<-c(1,26,24,68)
> rnames<-c("R1","R2")
> cnames<-c("C1","C2")
> mymatrix<-matrix(cells,nrow=2,ncol=2,byrow=TRUE,
+           dimnames=list(rnames, cnames))
> mymatrix

   C1 C2
R1  1 26
R2 24 68
```

Arrays

Arrays are similar to matrices but can have more than two dimensions

```
myarray <- array(vector, dimensions, dimnames)
```

```
> dim1<-c("A1","A2")
> dim2<-c("B1","B2","B3")
> dim3<-c("C1","C2","C3","C4")
> z<-array(1:24,c(2,3,4),dimnames=list(dim1,dim2,dim3))
```

Arrays

```
> z

, , C1

   B1 B2 B3
A1  1  3  5
A2  2  4  6

, , C2

   B1 B2 B3
A1  7  9 11
A2  8 10 12

, , C3

   B1 B2 B3
A1 13 15 17
A2 14 16 18

, , C4
```

Data Frame: Most used structure in Statistics

A **data frame** is more general than a matrix in that different columns can contain *different modes* of data (numeric, character, and so on)

```
mydata <- data.frame(col1, col2, col3,...)
```

where col1, col2, col3, and so on are column vectors of any type .

Example

```
> patientID<-c(1, 2, 3, 4)
> age<-c(25,34,28,52)
> diabetes<-c("Type1","Type2","Type1","Type1")
> status<-c("Poor","Improved","Excellent","Poor")
> patientdata<-data.frame(patientID,age,diabetes,status)
> patientdata

  patientID age diabetes    status
1         1  25    Type1      Poor
2         2  34    Type2  Improved
3         3  28    Type1 Excellent
4         4  52    Type1      Poor
```

Frequently used: str() and summary()

str(object) gives the structure of an object

```
> str(patientdata)

'data.frame':^^I4 obs. of  4 variables:
 $ patientID: num  1 2 3 4
 $ age      : num  25 34 28 52
 $ diabetes : Factor w/ 2 levels "Type1","Type2": 1 2 1 1
 $ status   : Factor w/ 3 levels "Excellent","Improved",..: 3 2 1 3

> summary(patientdata)

   patientID          age          diabetes       status
 Min.   :1.00   Min.   :25.00   Type1:3   Excellent:1
 1st Qu.:1.75   1st Qu.:27.25   Type2:1   Improved :1
 Median :2.50   Median :31.00             Poor     :2
 Mean   :2.50   Mean   :34.75
 3rd Qu.:3.25   3rd Qu.:38.50
 Max.   :4.00   Max.   :52.00
```

Frequently used: head() and tail()

head(object) lists the first part of an object. tail(object) lists the last part of an object. They are useful for quickly scanning large datasets.

```
> head(patientdata)

  patientID age diabetes    status
1         1  25    Type1      Poor
2         2  34    Type2  Improved
3         3  28    Type1 Excellent
4         4  52    Type1      Poor
```

Specifying elements of a data frame

```
> patientdata$age #variable age from patientdata

[1] 25 34 28 52

> patientdata[1:2]

  patientID age
1         1  25
2         2  34
3         3  28
4         4  52

> patientdata[c("diabetes", "status")]

  diabetes    status
1    Type1      Poor
2    Type2  Improved
3    Type1 Excellent
4    Type1      Poor
```

Types of variables

- Nominal variables
  - are categorical, without an implied order. e.g. Diabetes (Type1, Type2)
- Ordinal variables
  - categorical, imply order but not amount. e.g. Status (poor, improved, excellent)
- Continuous variables
  - can take on any value within some range, and both order and amount are implied

Definition

Categorical (nominal) and ordered categorical (ordinal) variables in R are called factors

The use of factor()

```
> diabetes<-c("Type1","Type2","Type1","Type1")
> diabetes

[1] "Type1" "Type2" "Type1" "Type1"

> diabetes<-factor(diabetes)
> diabetes

[1] Type1 Type2 Type1 Type1
Levels: Type1 Type2

> levels(diabetes)

[1] "Type1" "Type2"

> class(diabetes)

[1] "factor"
```

Ordered factor

```
> status<-c("Poor","Improved","Excellent","Poor")
> status1<-factor(status,order=TRUE)
> status1

[1] Poor      Improved  Excellent Poor
Levels: Excellent < Improved < Poor

> status2<-factor(status,order=TRUE,levels=c("Poor","Improved","Excellent"))
> status2

[1] Poor      Improved  Excellent Poor
Levels: Poor < Improved < Excellent

> status3<-ordered(status)
> status3

[1] Poor      Improved  Excellent Poor
Levels: Excellent < Improved < Poor
```

List: the most flexible and richest structure in R

Basically, a **list** is an ordered collection of objects (components).
A **list** allows you to gather a variety of (possibly unrelated) objects under one name.

list()

```
mylist<-list(object1,object2,...)
```

or

```
mylist<-list(name1=object1,name2=object2,...)
```

Example of a list

```
> g<-"My First List"
> h<-c(25, 26, 18, 39)
> j<-matrix(1:10,nrow=2)
> k<-c("one", "two", "three")
> mylist<-list(title=g,ages=h,j,k)
> mylist

$title
[1] "My First List"

$ages
[1] 25 26 18 39

[[3]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

[[4]]
[1] "one"   "two"   "three"
```

Data types summary

Data structure	Instruction in R	Description
vector	`c()`	Sequence of elements of the **same nature**.
matrix	`matrix()`	Two-dimensional table of elements of the **same nature**.
multidimensional table	`array()`	More general than a matrix; table with several dimensions.
list	`list()`	Sequence of R structures of any (and possibly different) nature.
individual$\times$variable table	`data.frame()`	Two-dimensional table. The columns can be of different natures, but must have the same length.
factor	`factor(), ordered()`	Vector of character strings associated with a modality table.
dates	`as.Date()`	Vector of dates.
time series	`ts()`	Values of a variable observed at several time points.

What type is your data?

Type	Description
`class()`	Class from which object inherits (vector, matrix, function, logical, list, $\cdots$)
`mode()`	Numeric, character, logical, $\cdots$
`storage.mode()`	Mode used by R to store object (double, integer, character, logical, $\cdots$)
`is.function()`	Logical (TRUE if function)
`is.na()`	Logical (TRUE if missing)
`names()`	Names associated with object
`dimnames()`	Names for each dim of array
`attributes()`	Names, class, etc.

Introduction to ggplot2

ggplot2 is a powerful and a flexible R package, implemented by *Hadley Wickham*, for producing elegant graphics.
The *gg* means *Grammar of Graphics*.
Plot = data + Aesthetics + Geometry

data	is a data frame
Aesthetics	is used to indicate x and y variables. It can be also used to control the color, the size or the shape of points, the height of bars, etc.....
Geometry	corresponds to the *type of graphics* (histogram, box plot, line plot, density plot, dot plot, ...)

Data: mpg

contains observations collected by the US Environment Protection Agency on 38 models of cars

```
> library(tidyverse)
> head(mpg)

# A tibble: 6 x 11
  manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
1 audi         a4      1.8  1999     4 auto~ f        18    29 p     comp~
2 audi         a4      1.8  1999     4 manu~ f        21    29 p     comp~
3 audi         a4      2    2008     4 manu~ f        20    31 p     comp~
4 audi         a4      2    2008     4 auto~ f        21    30 p     comp~
5 audi         a4      2.8  1999     6 auto~ f        16    26 p     comp~
6 audi         a4      2.8  1999     6 manu~ f        18    26 p     comp~
```

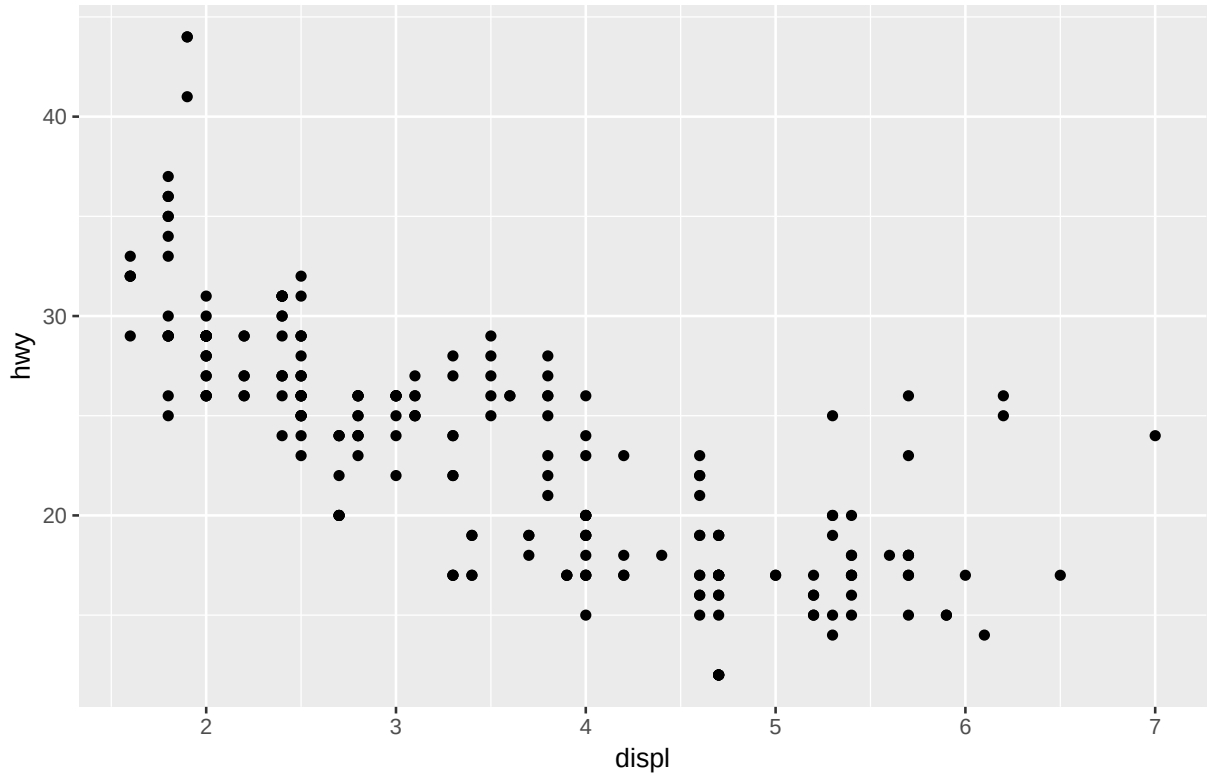
Data: mpg

Variables:

hwy	Fuel efficiency on the highway, in miles per gallon
year	year of manufacture
displ	Engine size, in liters
model	model name
drv	f = front-wheel drive, r = rear wheel drive, 4 = 4wd
trans	type of transmission
cyl	number of cylinders
cty	city miles per gallon
class	"type" of car

Creating a ggplot

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

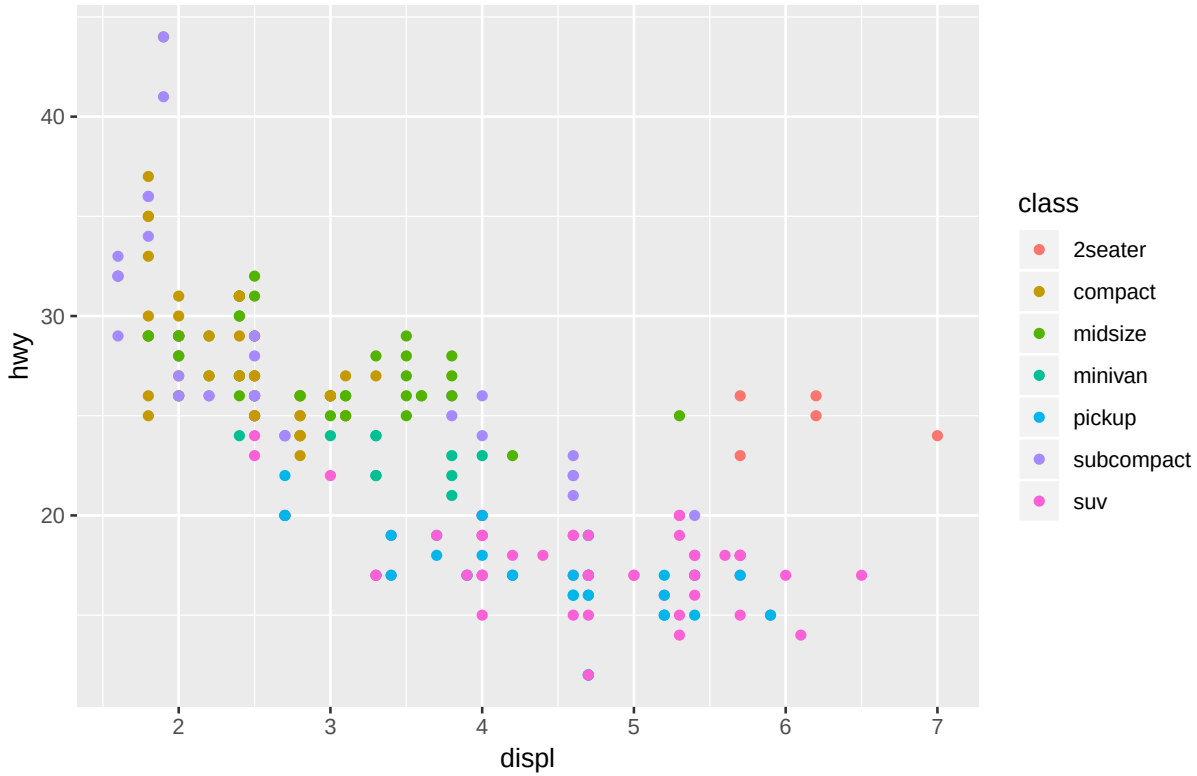


Save ggplots

```
# Print the plot to a pdf file

    pdf("myplot.pdf")
    myplot <- ggplot(...)
    print(myplot)
    dev.off()

# Print the plot to a png file

    png("myplot.png")
    print(myplot)
    dev.off()

# Save the plot to a pdf

    ggsave("myplot.pdf")

# OR save it to png file

    ggsave("myplot.png")
```

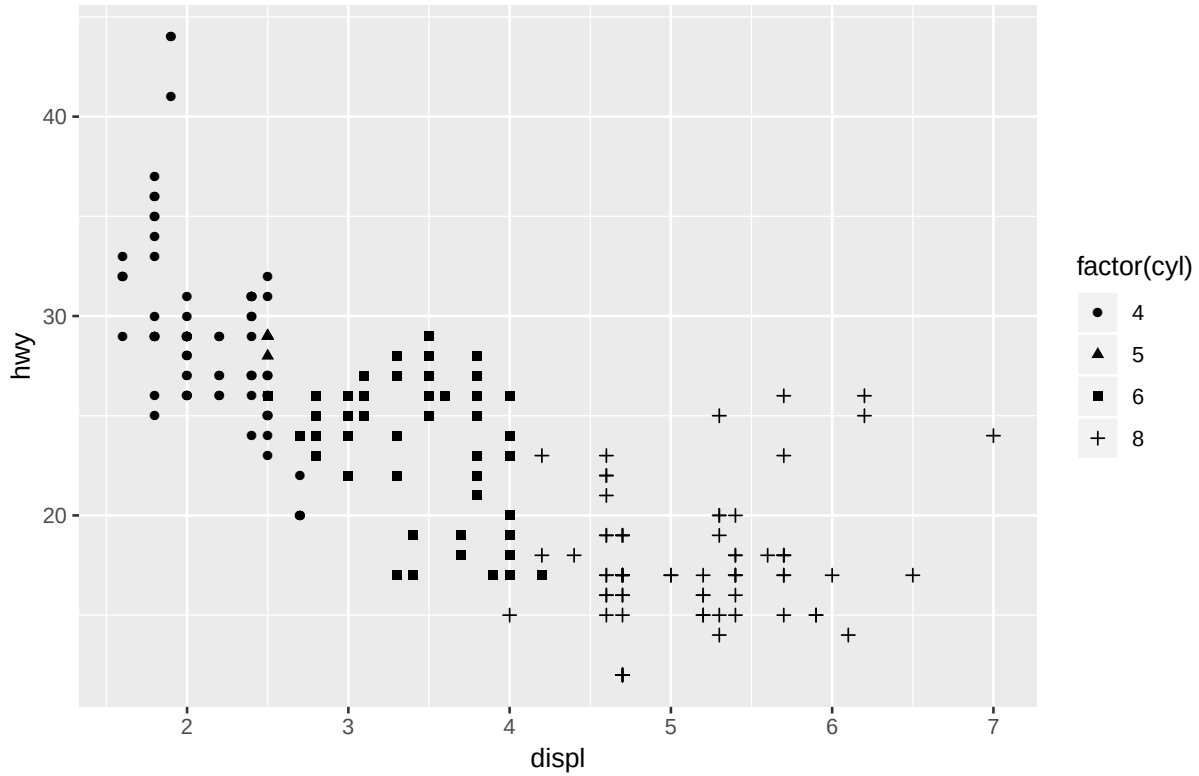
with colors

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



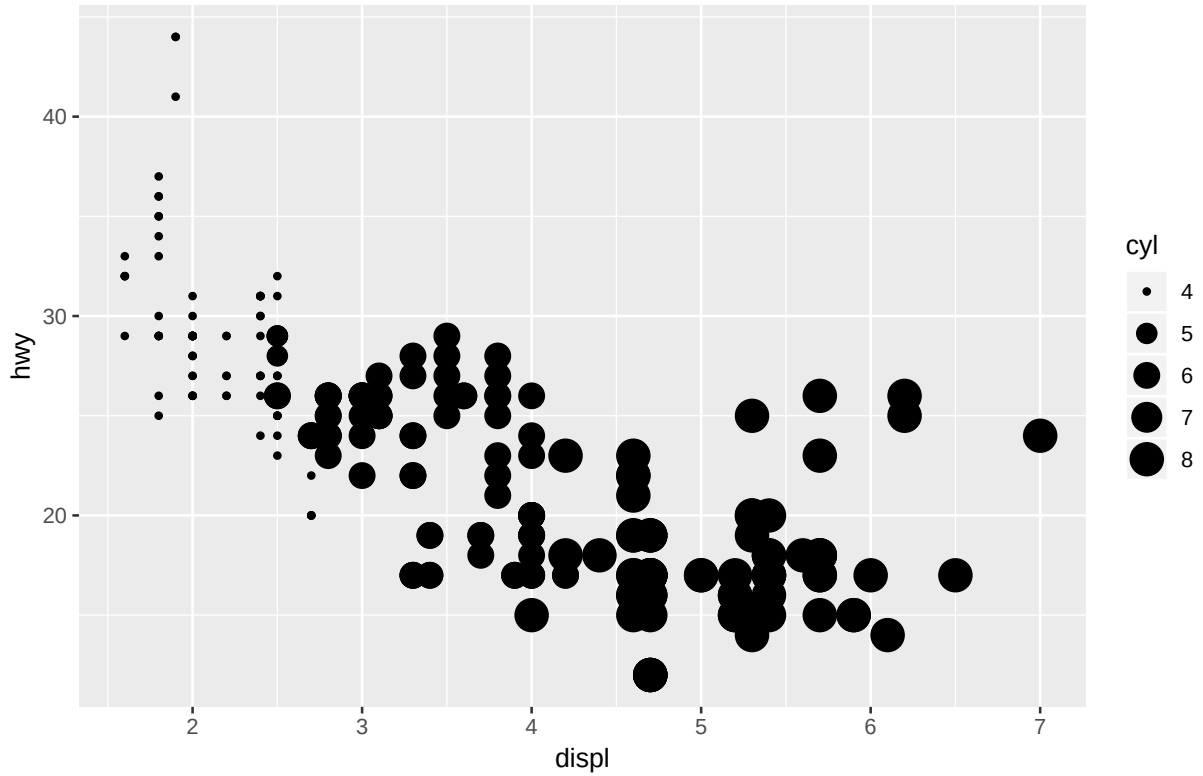
shape of the points

```
> ggplot(data = mpg) +
+  geom_point(mapping = aes(x = displ, y = hwy, shape = factor(cyl)))
```



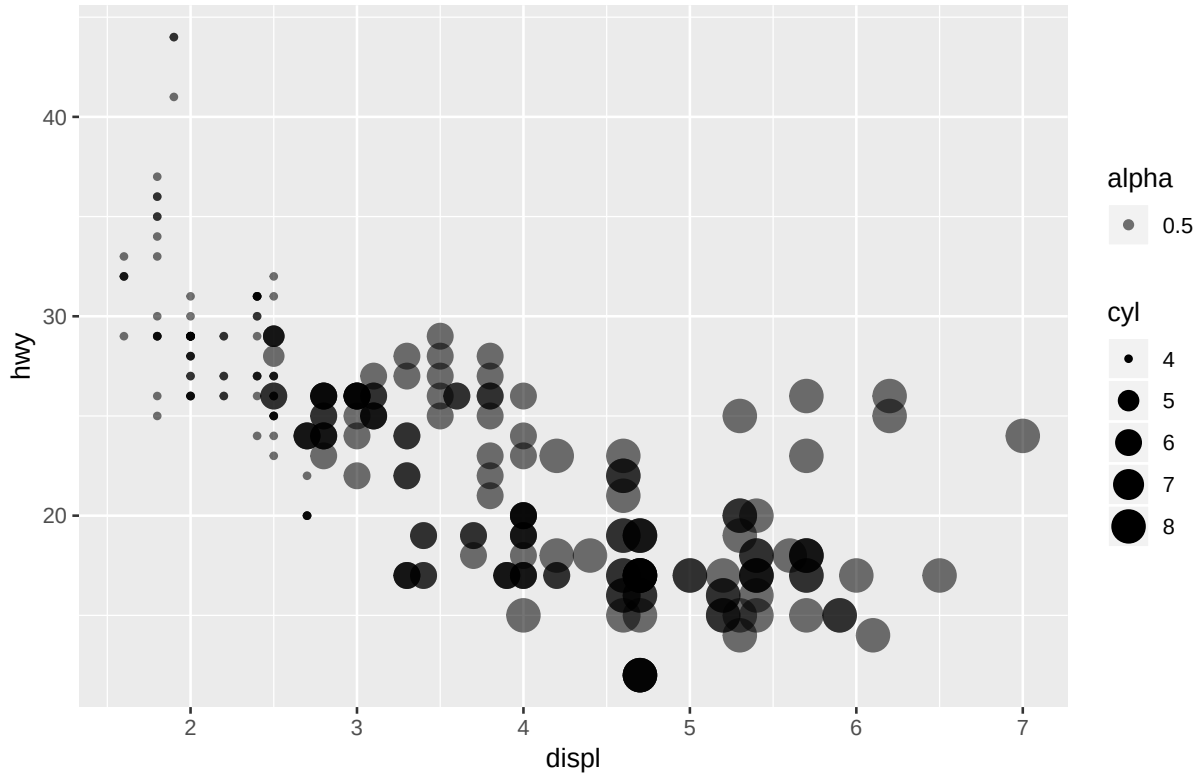
Variable size of points

```
> ggplot(data = mpg) +
+  geom_point(mapping = aes(x = displ, y = hwy, size = cyl))
```



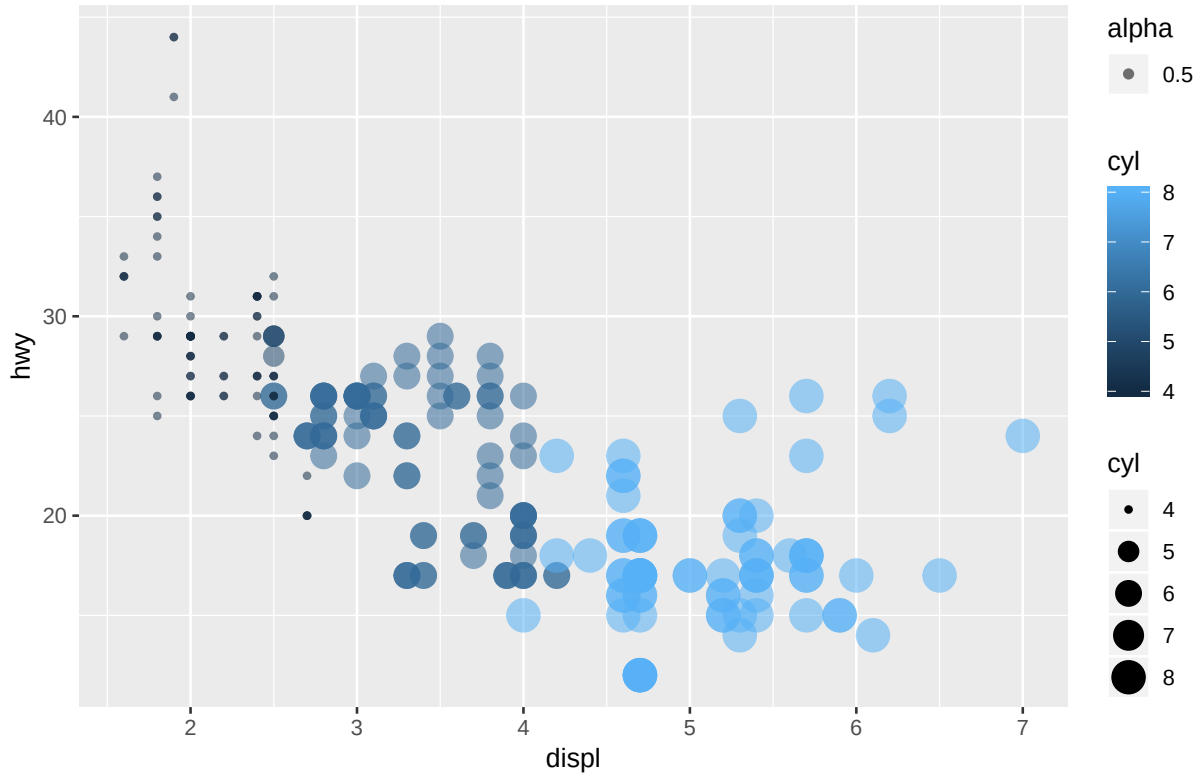
Variable points: size and transparency

```
> ggplot(data = mpg) +
+    geom_point(mapping = aes(x = displ, y = hwy,
+              size = cyl, alpha = 0.5))
```



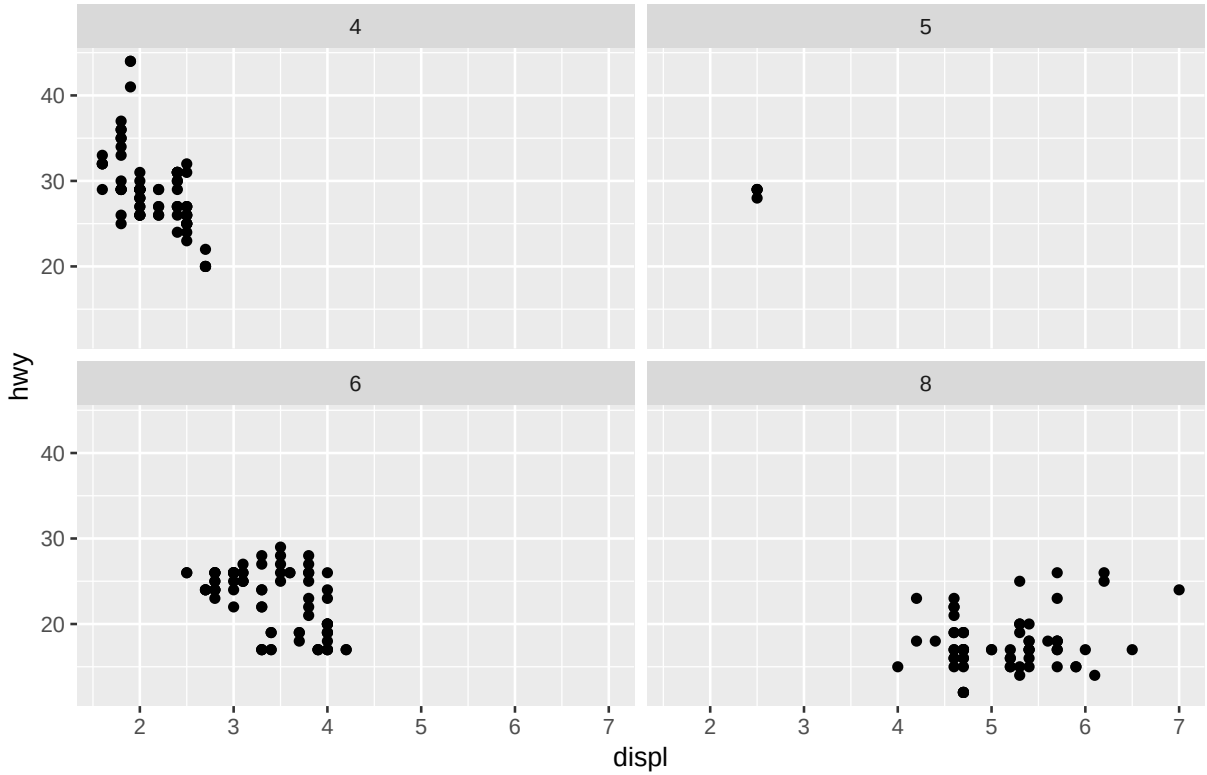
Variable points: size, colors and transparency

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy,
+     size = cyl, color = cyl, alpha = 0.5))
```



Facets

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ cyl, nrow = 2)
```



Plot One Variable

- For one continuous variable:

  geom_area() for area plot
  geom_density() for density plot
  geom_dotplot() for dot plot
  geom_freqpoly() for frequency polygon
  geom_histogram() for histogram plot
  stat_ecdf() for empirical cumulative density function
  stat_qq() for quantile - quantile plotting

- For one discrete variable:

  geom_bar() for bar plot

Plot Two Variables

geom_point() for scatter plot

geom_smooth() for adding smoothed line such as regression line

geom_quantile() for adding quantile lines

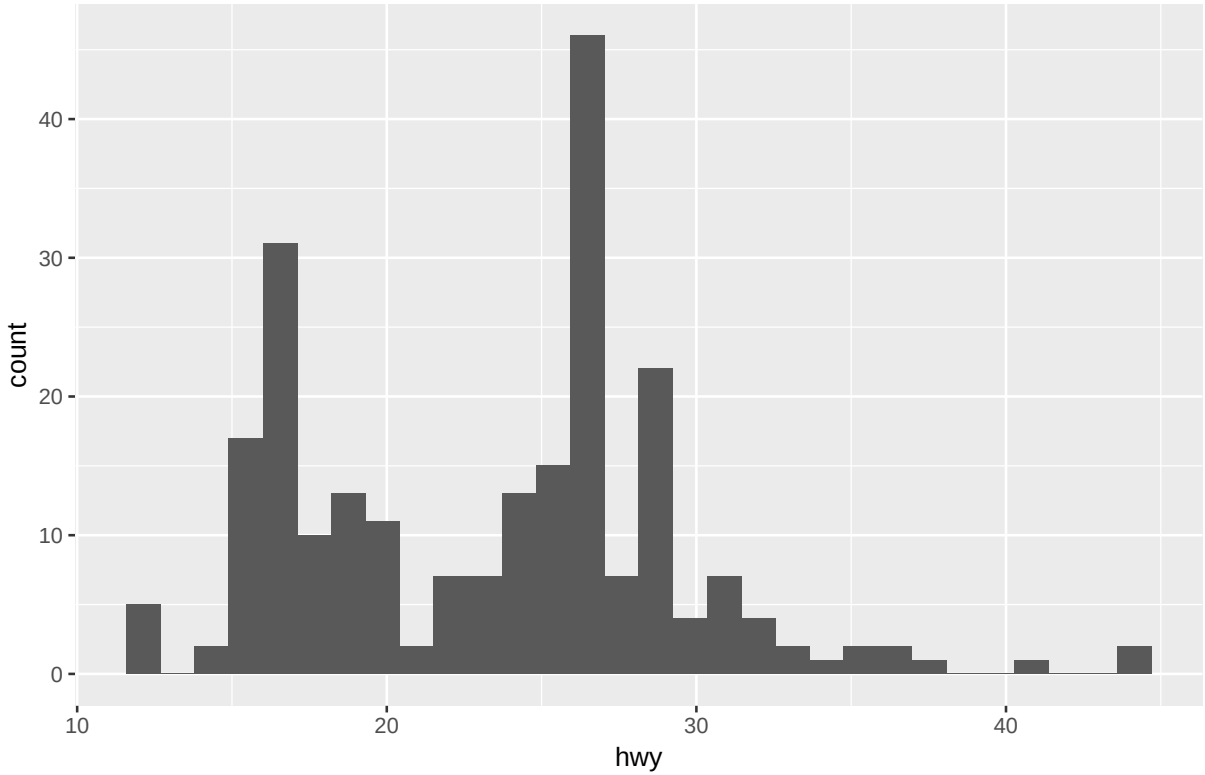
geom_rug() for adding a marginal rug

geom_jitter() for avoiding overplotting

geom_text() for adding textual annotations

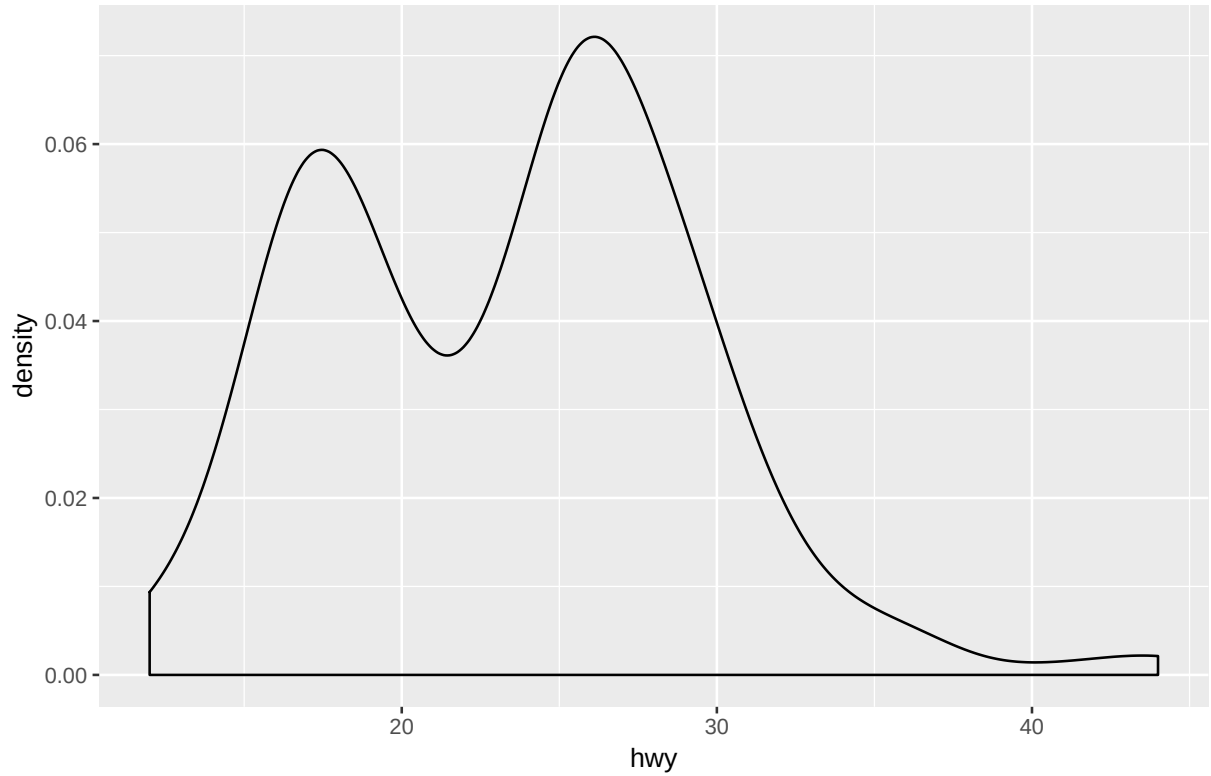
Plot of hwy: histogram

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



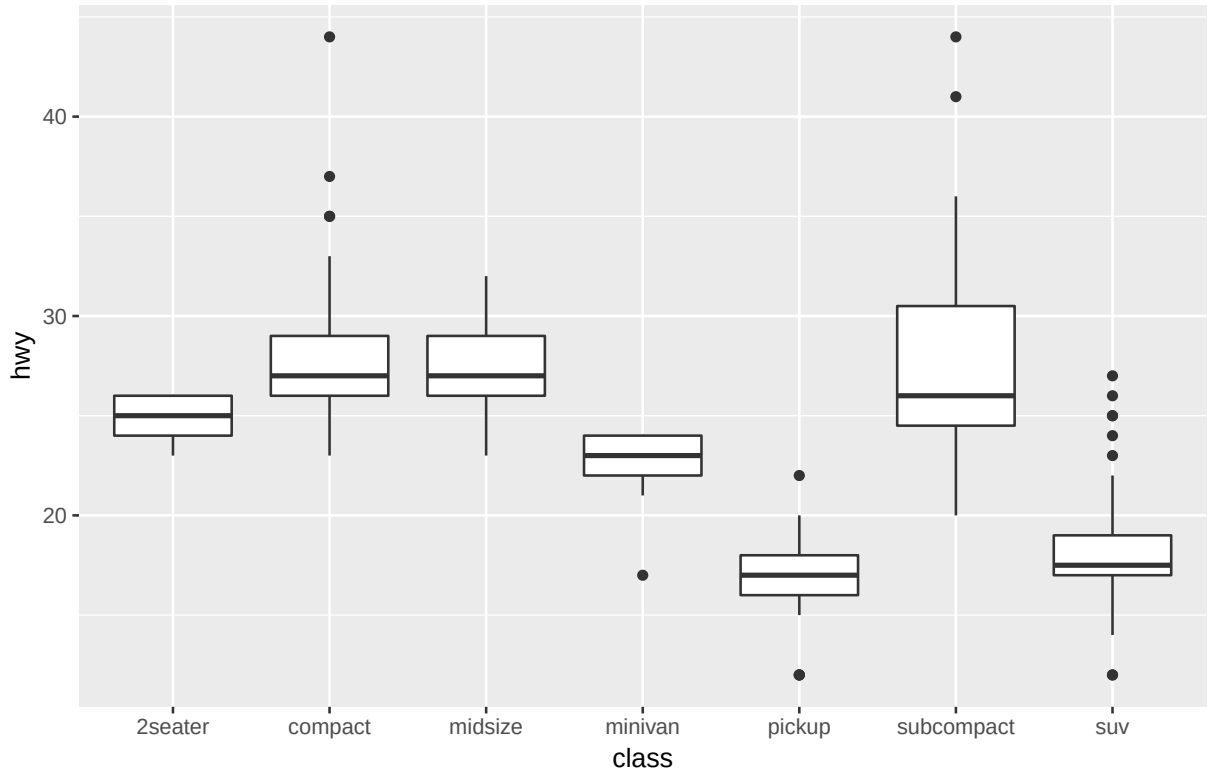
Plot of hwy: density

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



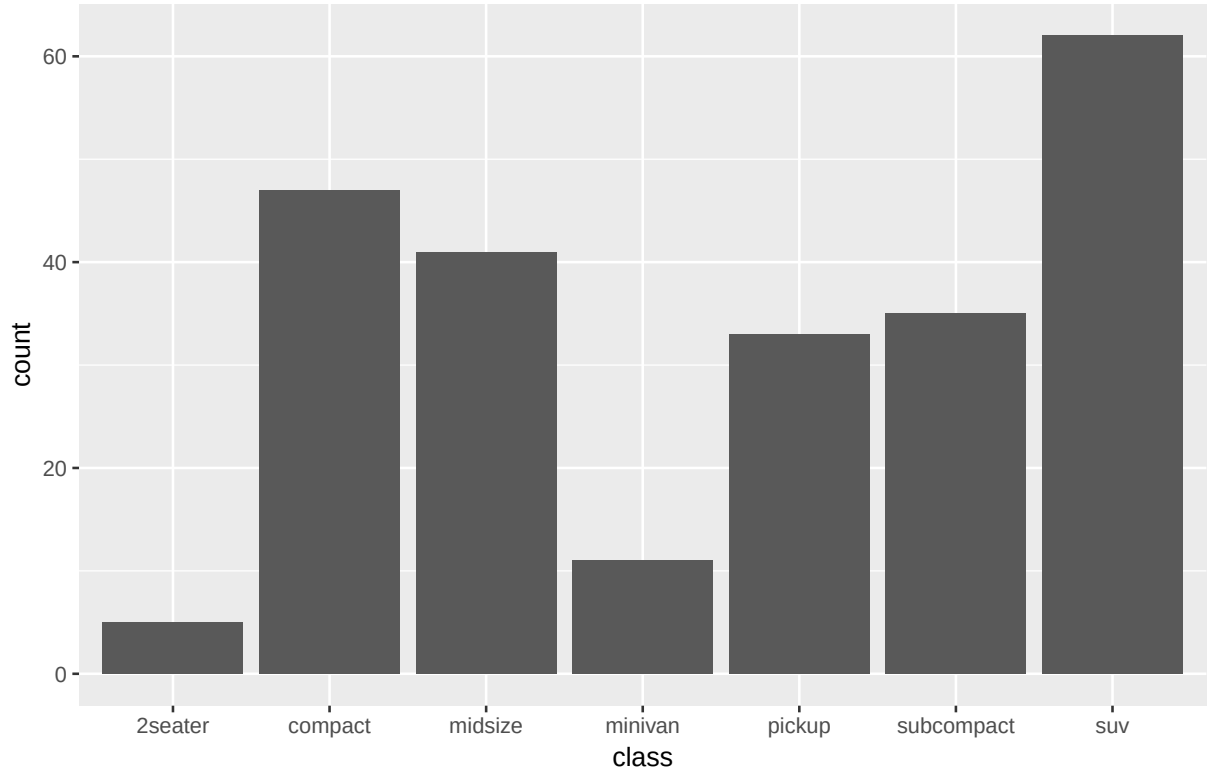
Plot of hwy: Box plot for comparison

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



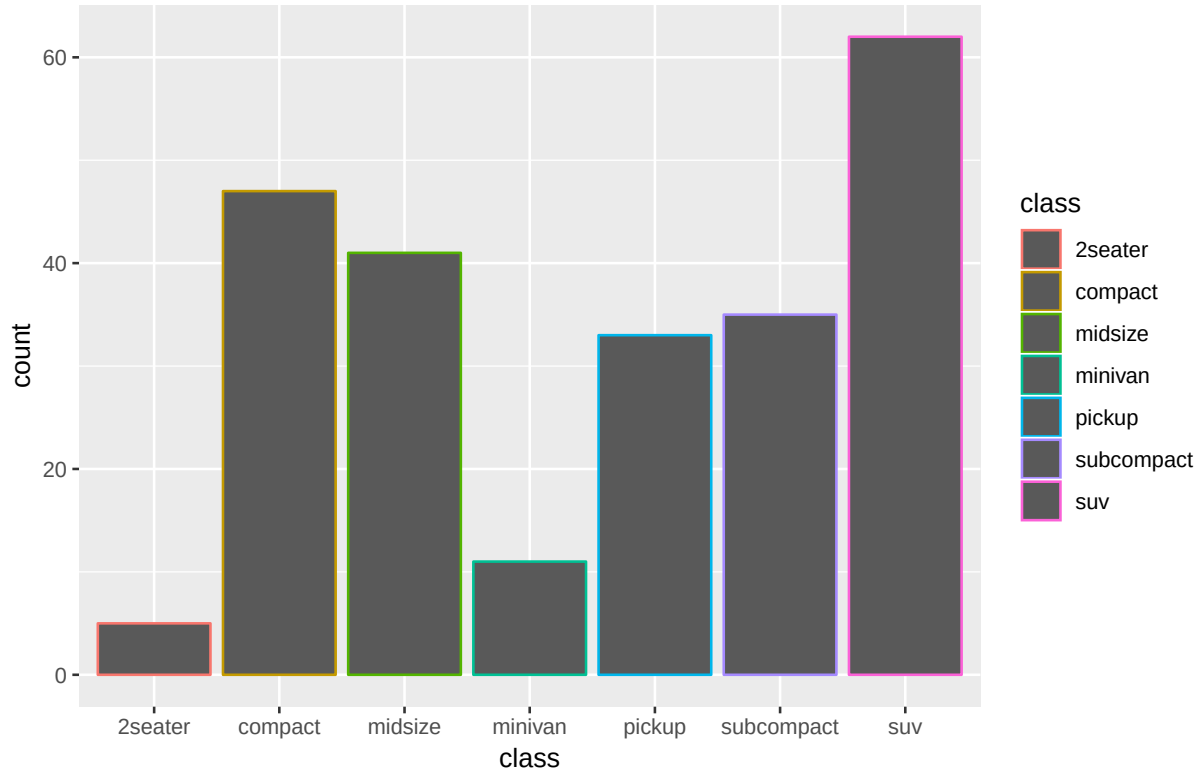
Plot of class: bar chart

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



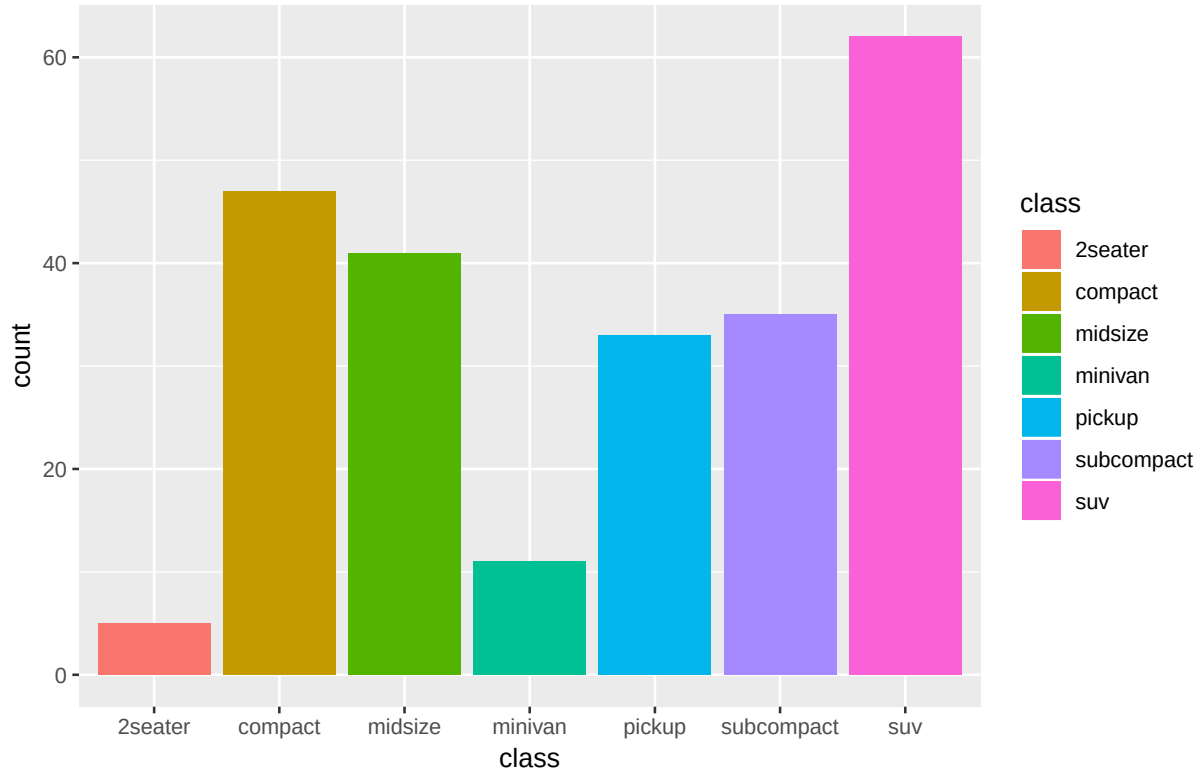
Plot of class: bar chart with colors

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



Plot of class: fill in colors into bars

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



Plot Two Variables: Scatter Plot

Scatter Plots:

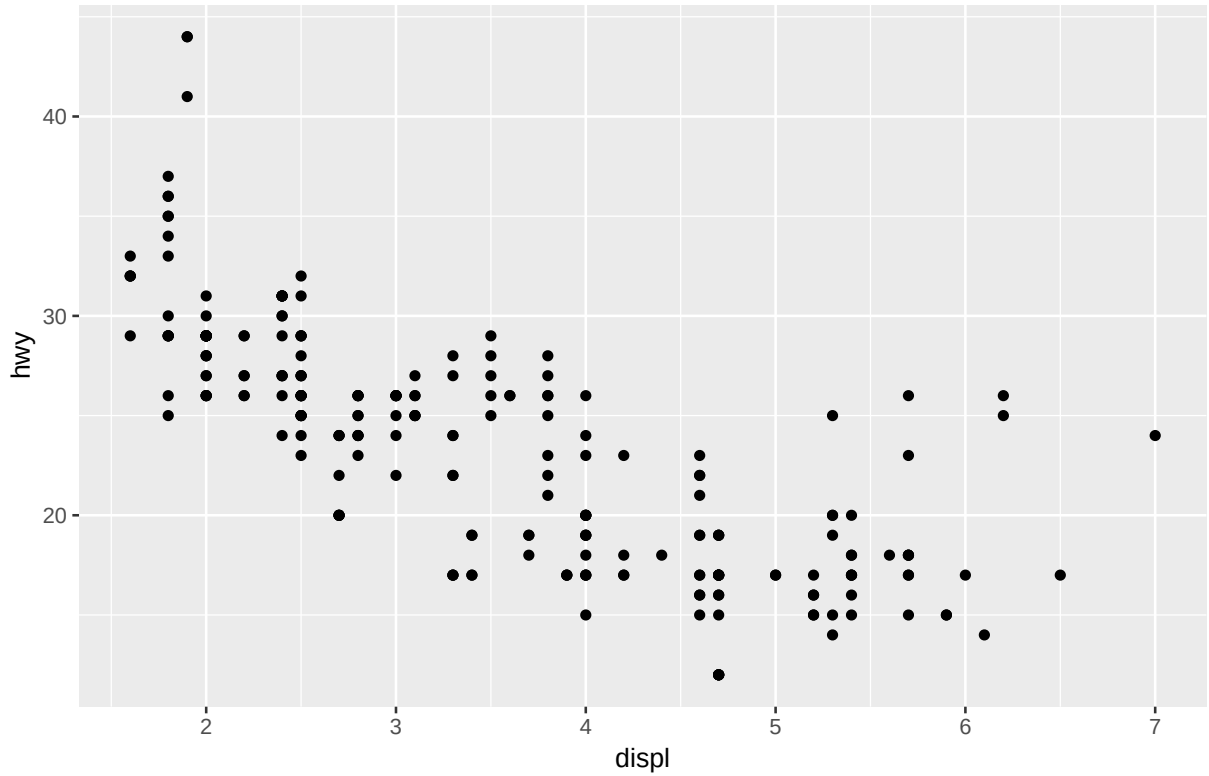
- Key function: geom_point()
- Key arguments to customize the plot: alpha, color, fill, shape and size

Add regression line or smoothed conditional mean:

- Key functions: geom_smooth() and geom_abline()
- Key arguments to customize the plot: alpha, color, fill, shape, linetype and size

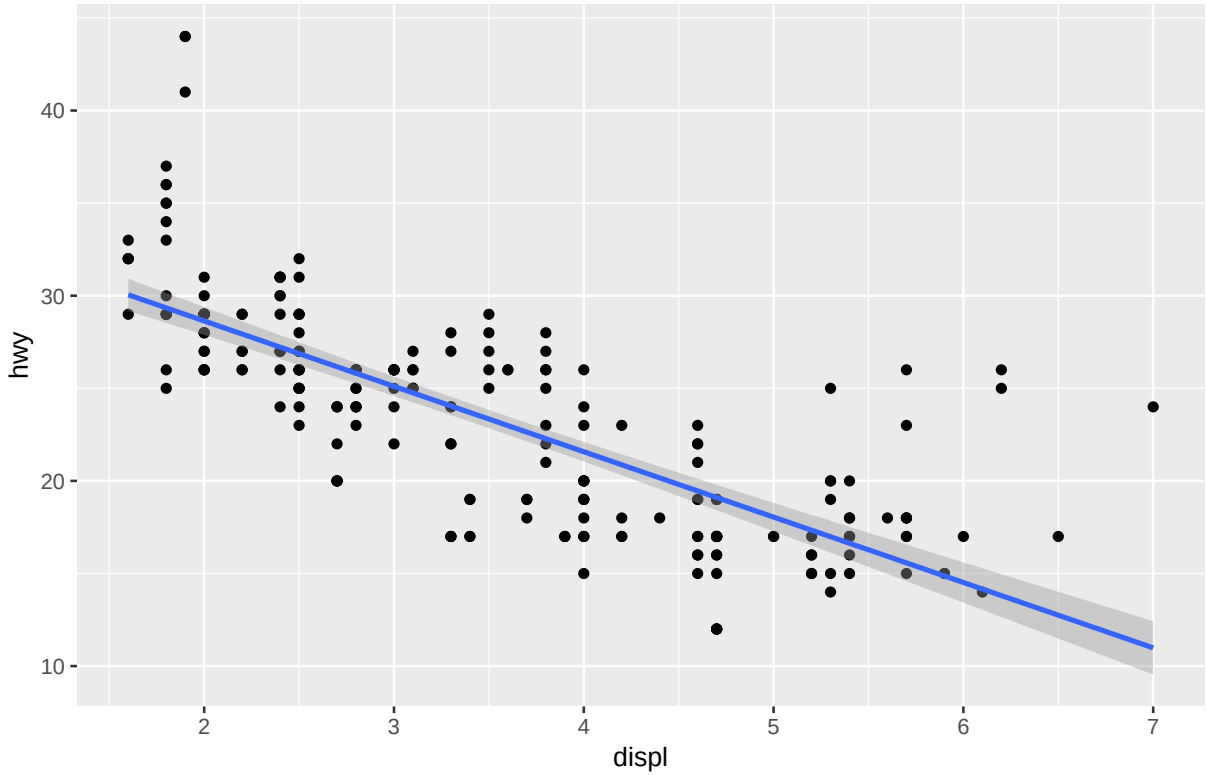
Scatter Plot

```
> b <- ggplot(mpg, aes(x = displ, y = hwy))
> b + geom_point()
```



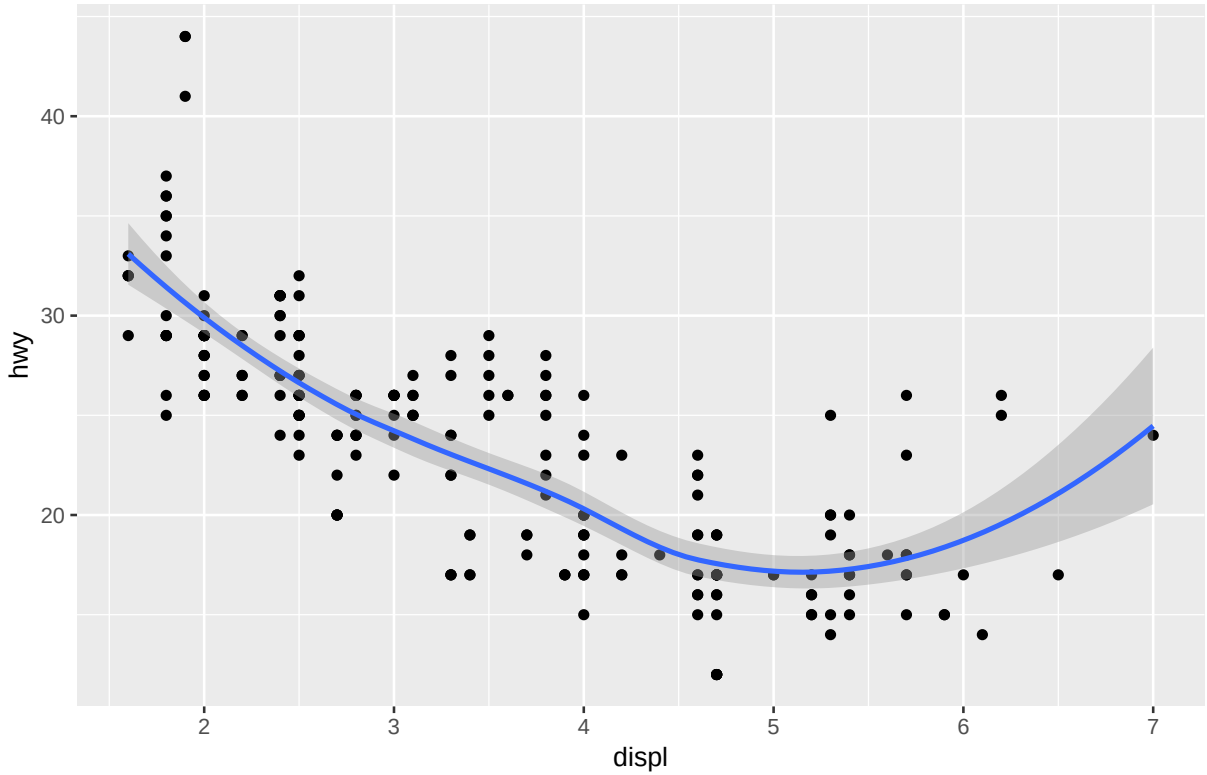
Scatter Plot with Regression Line

```
> b + geom_point() + geom_smooth(method = lm)
```



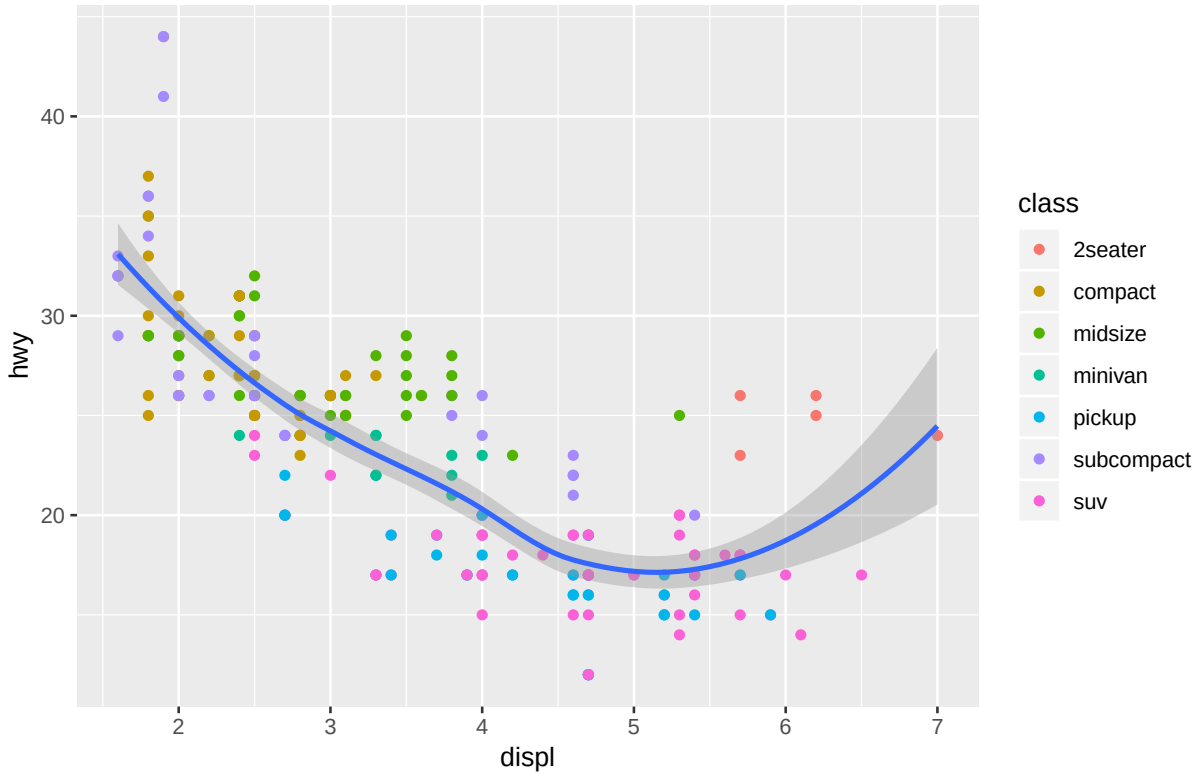
Loess method: local regression fitting

```
> b + geom_point() + geom_smooth()
```



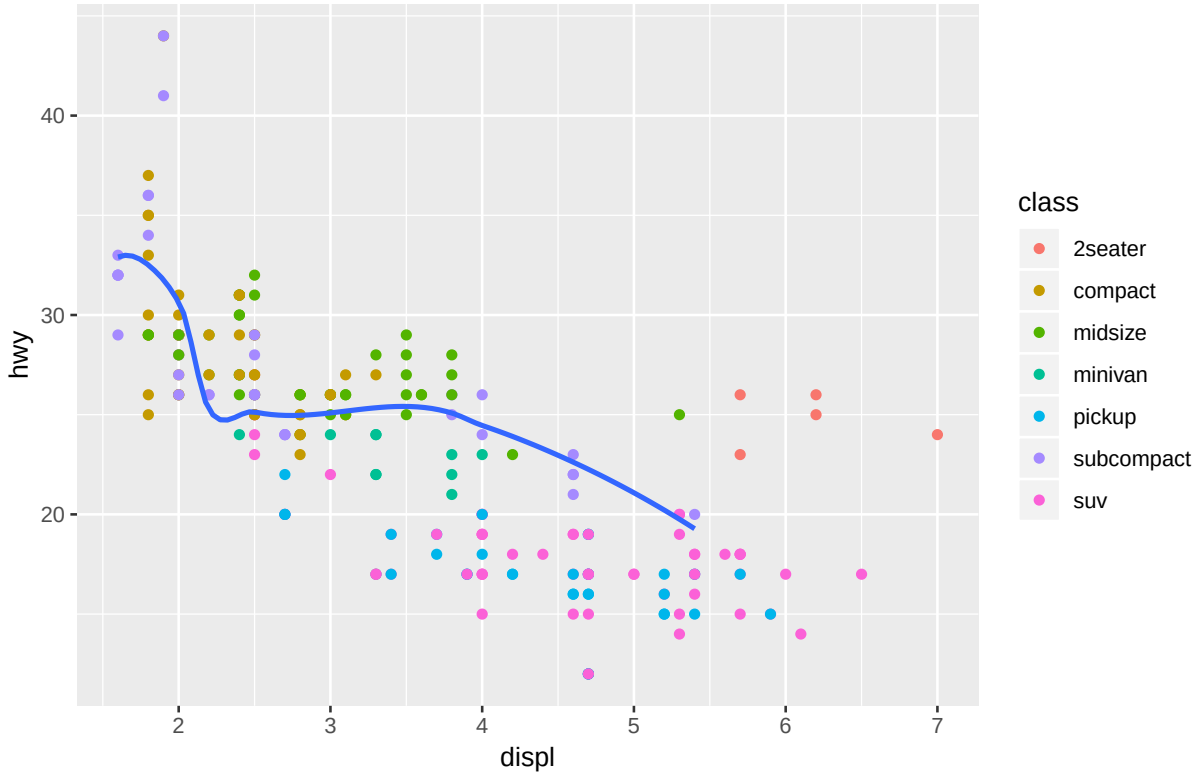
Local mappings for a layer

```
> b + geom_point(mapping = aes(color = class)) + geom_smooth()
```



Displays just a subset of the dataset

```
>  b + geom_point(mapping = aes(color = class)) +
+  geom_smooth(data = filter(mpg, class == "subcompact"),
+  se = FALSE)
```

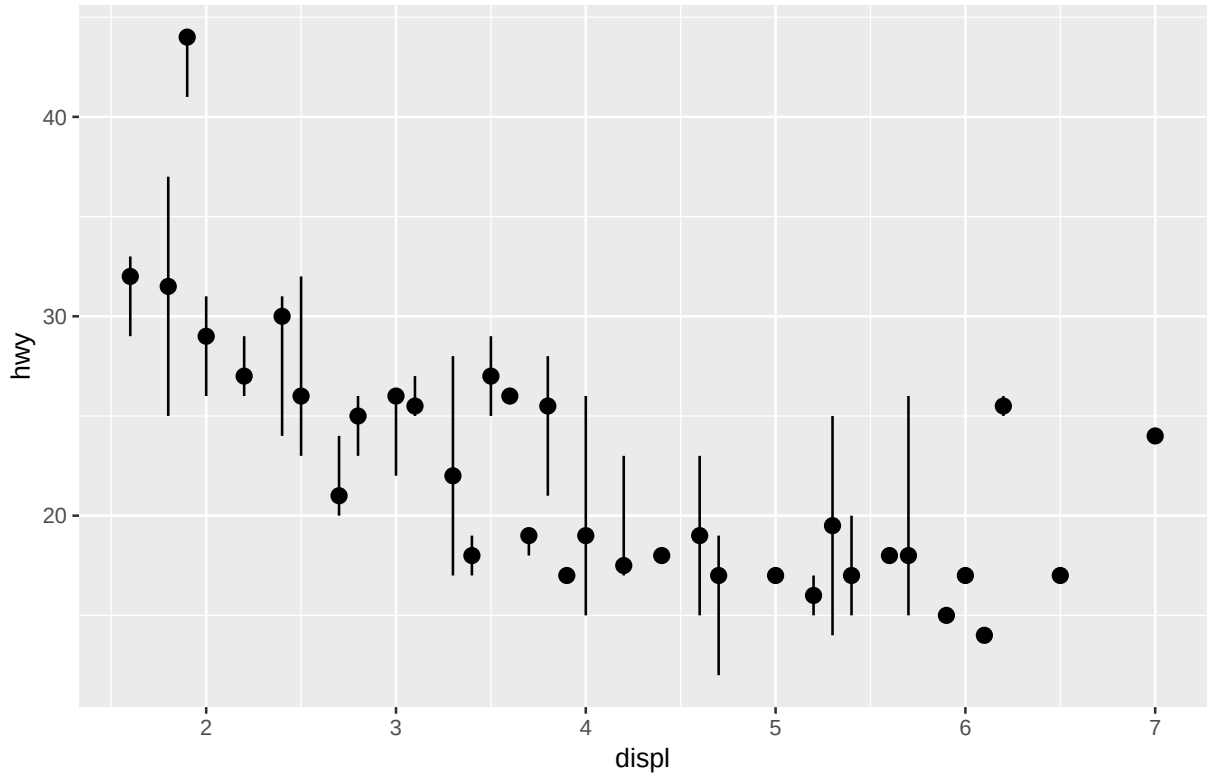


stat

- The algorithm used to calculate new values for a graph is called a *stat*
- Some plots visualize a transformation of the original data set. In this case, an alternative way to build a layer is to use *stat_*()* functions.
- geom_bar() = stat_count(): *?geom_bar* shows the default value for *stat* is "*count*"

Statistical transformation in your code

```
> ggplot(data = mpg) +
+ stat_summary( mapping = aes(x = displ, y = hwy),
+     fun.ymin = min, fun.ymax = max, fun.y = median )
```



Aims

In this section, you will learn

- how to transform your data using the *dplyr* package
- and a new dataset on flights departing New York City in 2013.

nycights13

This data frame contains all 336,776 flights that departed from New York City in 2013.
The data comes from the *US Bureau of Transportation Statistics*

```
> library(nycflights13)
> flights

# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
#     with 336,766 more rows, and 12 more variables: sched_arr_time <int>
```

tibble is a data frame

- Tibbles are a modern take on data frames.
- They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors).

int	stands for integers.
dbl	stands for doubles, or real numbers.
chr	stands for character vectors, or strings.
dttm	stands for date-times (a date + a time)

dplyr Basics

filter()	Pick observations by their values.
arrange()	Reorder the rows.
select()	Pick variables by their names.
mutate()	Create new variables with functions of existing variables.
summarize()	Collapse many values down to a single summary.

filter() allows you to subset observations

All dplyr work similarly:

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

Display filtered results

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

# A tibble: 842 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 832 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

Display and save results

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

# A tibble: 719 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013    12    25      456            500        -4      649
 2  2013    12    25      524            515         9      805
 3  2013    12    25      542            540         2      832
 4  2013    12    25      546            550        -4     1022
 5  2013    12    25      556            600        -4      730
 6  2013    12    25      557            600        -3      743
 7  2013    12    25      557            600        -3      818
 8  2013    12    25      559            600        -1      855
 9  2013    12    25      559            600        -1      849
10  2013    12    25      600            600         0      850
# ... with 709 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

Logical Operators

Boolean operators:

- "&" is "and,"
- "|" is "or,"
- "!" is "not,"
- "!=" is "not equal,"
- "==" is "equal,"
- "x %in% y" select every row where x is one of the values in y.

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

Missing Values

One important feature of R that can make comparison tricky is missing values, or NAs ("not availables").

```
> df <- tibble(x = c(1, NA, 3))
> filter(df, x > 1)

# A tibble: 1 x 1
      x
  <dbl>
1     3

> filter(df, is.na(x) | x > 1)

# A tibble: 2 x 1
      x
  <dbl>
1    NA
2     3
```

arrange()

- *arrange()* works similarly to *filter()* except that instead of selecting rows, it *changes their order*.
- It takes a data frame and a set of column names (or more complicated expressions) to order by.

Arrange by a set of column names

```
> arrange(flights, year, month, day)

# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

Use desc() to reorder by a column in descending order

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

# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     9      641            900      1301     1242
 2  2013     6    15     1432           1935      1137     1607
 3  2013     1    10     1121           1635      1126     1239
 4  2013     9    20     1139           1845      1014     1457
 5  2013     7    22      845           1600      1005     1044
 6  2013     4    10     1100           1900       960     1342
 7  2013     3    17     2321            810       911      135
 8  2013     6    27      959           1900       899     1236
 9  2013     7    22     2257            759       898      121
10  2013    12     5      756           1700       896     1058
# ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

select()

select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

```
> # Select columns by name
> select(flights, year, month, day)

# A tibble: 336,776 x 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
10  2013     1     1
# ... with 336,766 more rows
```

Select all columns between year and day

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

# A tibble: 336,776 x 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
10  2013     1     1
# ... with 336,766 more rows
```

Select all columns except those from year to day

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

# A tibble: 336,776 x 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
      <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1      517            515         2      830            819        11
 2      533            529         4      850            830        20
 3      542            540         2      923            850        33
 4      544            545        -1     1004           1022       -18
 5      554            600        -6      812            837       -25
 6      554            558        -4      740            728        12
 7      555            600        -5      913            854        19
 8      557            600        -3      709            723       -14
 9      557            600        -3      838            846        -8
10      558            600        -2      753            745         8
# ... with 336,766 more rows, and 10 more variables: carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

mutate() always adds new columns at the end of your dataset

```
#adds variables gain and speed
flights_sml <- select(flights, year:day,
                ends_with("delay"), distance, air_time )
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       speed = distance / air_time * 60)
```

mutate() adds gain and speed

```
> flights_sml <- select(flights, year:day, ends_with("delay"), distance, ai
> mutate(flights_sml,
+     gain = arr_delay - dep_delay,
+     speed = distance / air_time * 60)

# A tibble: 336,776 x 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.
 2  2013     1     1         4        20     1416      227    16  374.
 3  2013     1     1         2        33     1089      160    31  408.
 4  2013     1     1        -1       -18     1576      183   -17  517.
 5  2013     1     1        -6       -25      762      116   -19  394.
 6  2013     1     1        -4        12      719      150    16  288.
 7  2013     1     1        -5        19     1065      158    24  404.
 8  2013     1     1        -3       -14      229       53   -11  259.
 9  2013     1     1        -3        -8      944      140    -5  405.
10  2013     1     1        -2         8      733      138    10  319.
# ... with 336,766 more rows
```

new added variable can be used

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

# A tibble: 336,776 x 10
    year month   day dep_delay arr_delay distance air_time  gain hours
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
 1  2013     1     1         2        11     1400      227     9 3.78
 2  2013     1     1         4        20     1416      227    16 3.78
 3  2013     1     1         2        33     1089      160    31 2.67
 4  2013     1     1        -1       -18     1576      183   -17 3.05
 5  2013     1     1        -6       -25      762      116   -19 1.93
 6  2013     1     1        -4        12      719      150    16 2.5
 7  2013     1     1        -5        19     1065      158    24 2.63
 8  2013     1     1        -3       -14      229       53   -11 0.883
 9  2013     1     1        -3        -8      944      140    -5 2.33
10  2013     1     1        -2         8      733      138    10 2.3
# ... with 336,766 more rows, and 1 more variable: gain_per_hour <dbl>
```

summarize() collapses a data frame to a single row

```
> summarize(flights, delay = mean(dep_delay, na.rm = TRUE))

# A tibble: 1 x 1
  delay
  <dbl>
1  12.6
```

summarize() togethar with group_by()

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

# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day delay
   <int> <int> <int> <dbl>
 1  2013     1     1 11.5
 2  2013     1     2 13.9
 3  2013     1     3 11.0
 4  2013     1     4  8.95
 5  2013     1     5  5.73
 6  2013     1     6  7.15
 7  2013     1     7  5.42
 8  2013     1     8  2.55
 9  2013     1     9  2.28
10  2013     1    10  2.84
# ... with 355 more rows
```

Motivation

Imagine that we want to explore the relationship between the *distance* and *average delay* for each location.
Using what you know about *dplyr*, you might write code like this:

```
by_dest <- group_by(flights, dest)

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

There are three steps to prepare this data:

1. Group flights by destination.
2. Summarize to compute distance, average delay, and number of flights.
3. Filter to remove noisy points and Honolulu airport, which is almost twice as far away as the next closest airport.

This code is a little frustrating to write because we have to give each intermediate data frame a name, even though we don't care about it. Naming things is hard, so this slows down our analysis.

Useful pipe operator: %>%

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

This focuses on the transformations, not what's being transformed.
A good way to pronounce %>% when reading code is "then."

Missing Values via %>%

You may have wondered about the na.rm argument we used earlier. What happens if we don't set it?

```
> flights %>%
+ group_by(year, month, day) %>%
+ summarize(mean = mean(dep_delay))

# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day  mean
   <int> <int> <int> <dbl>
 1  2013     1     1    NA
 2  2013     1     2    NA
 3  2013     1     3    NA
 4  2013     1     4    NA
 5  2013     1     5    NA
 6  2013     1     6    NA
 7  2013     1     7    NA
 8  2013     1     8    NA
 9  2013     1     9    NA
10  2013     1    10    NA
```

Missing Values via %>%

```
> flights %>%
+ group_by(year, month, day) %>%
+ summarize(mean = mean(dep_delay, na.rm = TRUE))

# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day  mean
   <int> <int> <int> <dbl>
 1  2013     1     1 11.5
 2  2013     1     2 13.9
 3  2013     1     3 11.0
 4  2013     1     4  8.95
 5  2013     1     5  5.73
 6  2013     1     6  7.15
 7  2013     1     7  5.42
 8  2013     1     8  2.55
 9  2013     1     9  2.28
10  2013     1    10  2.84
# ... with 355 more rows
```

Descriptive Statistics

Numerical summaries of the *population* are called **parameters**, while numerical summaries of the *sample* are called **statistics**.

- Summary Measures of Location
  - mean, median, mode, quantiles,
- Summary Measures of Spread
  - range, interquartile-range(IQR), variance, standard-deviation(sd),The Median Absolute Deviation (MAD)
- Summary Measures of Shape
  - skewness, kurtosis

R functions for location

- Population mean: μ
- Sample mean: $\bar{x}$
- R functions: mean(x), median(x), mode(x)
- Quantiles: the x_p is called a p-**quantile** of a distribution, if $\mathrm{P}(X \leq x_p) \geq p$ and $\mathrm{P}(X \geq x_p) \leq 1 - p$
  - for continuous r.v., $\mathrm{P}(X \leq x_p) = p$
- quantile(x, probs=c(0.25, 0.5, 0.75)): Q_1, Q_2, Q_3

Mtcar data

```
> attach(mtcars)
> mean(mpg)

[1] 20.09062

> median(mpg)

[1] 19.2

> quantile(mpg,probs=c(0.25,0.5,0.75))

   25%    50%    75%
15.425 19.200 22.800

> detach(mtcars)
```

functions

- range(x): returns the smallest and largest values in x
- IQR(x): Interquartile Range, IQR= Q3 - Q1
- var(x): $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
- sd(x): $s = \sqrt{s^2}$
- Sample Coeffcient of Variation: $CV = S/\bar{X}$
- Relative Standard Deviation: $RSD = |S/\bar{X}| \times 100$
- The Median Absolute Deviation (MAD): is a robust measure of spread, often used when the median is reported to describe the center of a *skewed data set*.

$$MAD = \text{median}\{|x_i - m|\}$$

where m is the median of x.

Skewness and Kurtosis

The base installation of R doesn't provide functions for skew and kurtosis

- **Skewness** is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean.
- **Kurtosis** is a measure of the "tailedness" of the probability distribution of a real-valued random variable.

$$\text{Skew} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^3$$

$$\text{Kurt} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^4$$

- Negative skew: The left tail is longer; the mass of the distribution is concentrated on the right of the figure.The distribution is said to be *left-skewed, left-tailed*, or *skewed to the left*
- Positive skew: *right-skewed, right-tailed*, or *skewed to the right*
- The ***excess kurtosis*** = *Kurt - 3* (Kurt=3 for Normal)

Own-written function for descriptive statistics

```
> mystats<-function(x,na.omit=FALSE){
+   if(na.omit)
+      x<-x[!is.na(x)]
+   m<-mean(x)
+   n<-length(x)
+   s<-sd(x)
+   skew<-sum((x-m)^3/s^3)/n
+   kurt<-sum((x-m)^4/s^4)/n - 3
+ return(c(n=n,mean=m,stdev=s,skew=skew,kurtosis=kurt))
+ }
> round(mystats(mtcars$mpg),3)

      n      mean     stdev      skew kurtosis
 32.000   20.091    6.027    0.611   -0.373
```