Data Analytics and Visualizations in R - Exercises

Contents

```
library(ggplot2)
library(data.table)
library(magrittr)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:magrittr':
##
##     extract
```

1 Basic R Data Structures

1.1 Types

What are the scalar types in R?

```
# Scalars are just vectors of length one
```

1.2 Weirdness of R

What is the major difference between atomic vectors and lists? How can you turn a list into an atomic vector?

In order to check if an object is of a certain type you can use `is.[type](object)` ,e.g. `is.integer(object)`

Can you use the `is.vector()` function to understand whether a data structure is a vector? If not, what are the functions that you can use for this purpose?

```
# Atomic vectors can contain elements of the same type. The elements of a list
# can have different types.

# Yes, we can use the `is.vector()` function to understand is the structure is a
# vector. It will return TRUE or FALSE.

a <- list(c(1,2,3), "bla", TRUE)
print(a)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "bla"
##
## [[3]]
## [1] TRUE
```

```
is.vector(a)
```

```
## [1] TRUE
```

```r
# Best response: If possible you should use type specific coercions like
# `as.numeric()` or `as.character()`. But since lists are heterogenous, this
# might not work. A more general function is `unlist()`,  which returns the list
# into a vector of the most general type. Notice this difference:

a <- list(c(1,2,3), "bla", TRUE)
unlist(a)
```

```
## [1] "1"    "2"    "3"    "bla"  "TRUE"
```

```r
as.character(a)
```

```
## [1] "c(1, 2, 3)" "bla"        "TRUE"
```

```r
# Generally you can, but here comes the weird part: `is.vector()` will only return `TRUE` if the vector
# no attributes `names`. Therefore more specific functions like `is.atomic()` or `is.list()` functions
# be used to test if an object is actually atomic vectoror a list
```

1.3 Atomic Vector Concatenation

What happens when you try to generate an atomic vector with `c()`which is composed of different types of elements? What is the `mean()` of a logical vector?

```r
# When we attempt to combine different types they will be coerced to the most
# flexible type. Types from least to most flexible are: logical, integer, double
#  and character.

str(c("a", 1)) # 1 corced to char
```

```
##  chr [1:2] "a" "1"
```

```r
# As TRUE is encoded as 1 and FALSE as 0, the mean is the number of TRUEs
# devided by the vector length.
mean(c(TRUE, FALSE,FALSE))
```

```
## [1] 0.3333333
```

1.4 Vector Concatenation

Compare X and Y where X and Y are defined as follows. What is the difference?

```r
x <- list(list(1,2), c(3,4))
y <- c(list(1,2), c(3,4))
```

```r
# Answer
# X will combine seveal lists into one. Given a combination of atomic vectors
# and lists, y will coerce the vectors to lists before combining them.
str(x)
```

```
## List of 2
##  $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
##  $ : num [1:2] 3 4
```

```r
str(y)
```

```
## List of 4
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : num 4
```

1.5 From Vectors to `data.frames`

First, create three named numeric vectors of size 10, 11 and 12 respectively in the following manner:

- One vector with the "colon" approach: *from:to*
- One vector with the `seq()` function: *seq(from, to)*
- And one vector with the `seq()` function and the by argument: *seq(from, to, by)*

For easier naming you can use the vector `letters` or `LETTERS` which contain the latin alphabet in small and capital, respectively. In order to select specific letters just use e.g. `letters[1:4]` to get the first four letters. Check their types. What is the outcome? Where do you think does the difference come from?

Then combine all three vectors in a list. Check the attributes of the vectors and the list. What is the difference and why?

Finally coerce the list to a `data.frame` with `as.data.frame()`. Why does it fail and how can we fix it? What happend to the names?

Hint: If list elements have no names, we can access them with the double brackets and an index, e.g. `my_list[[1]]`

```
# Answer

# A. create vectors
aa <- 1:10
names(aa) <- letters[aa]
aa
```

```
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
```

```
bb <- seq(1, 11)
names(bb) <- letters[bb]
bb
```

```
##  a  b  c  d  e  f  g  h  i  j  k
##  1  2  3  4  5  6  7  8  9 10 11
```

```
cc <- seq(1, 12, by=1)
names(cc) <- letters[cc]

typeof(aa)
```

```
## [1] "integer"
```

```
typeof(bb)
```

```
## [1] "integer"
```

```
typeof(cc)
```

```
## [1] "double"
```

```r
# B. Combine all three vectors in a list

my_list <- list(aa, bb, cc)
my_list
```

```
## [[1]]
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##  a  b  c  d  e  f  g  h  i  j  k
##  1  2  3  4  5  6  7  8  9 10 11
##
## [[3]]
##  a  b  c  d  e  f  g  h  i  j  k  l
##  1  2  3  4  5  6  7  8  9 10 11 12
```

```r
attributes(aa)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```r
attributes(bb)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k"
```

```r
attributes(cc)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```r
attributes(my_list)
```

```
## NULL
```

```r
# C. Coerce to data.frames

# my_df <- as.data.frame(my_list)# fails

# Fixing the length
my_list[[1]] <- c(my_list[[1]], NA, NA)
my_list[[2]] <- c(my_list[[2]], NA)

my_df <- as.data.frame(my_list)
names(my_df) <- LETTERS[1:3]
my_df
```

```
##    A  B  C
## a  1  1  1
## b  2  2  2
## c  3  3  3
## d  4  4  4
## e  5  5  5
## f  6  6  6
## g  7  7  7
## h  8  8  8
## i  9  9  9
```

```
## j 10 10 10
## k NA 11 11
##   NA NA 12
```

1.6 Attributes

Take again our `data.frame` from Question 5.

- Change the row names and the column names of the `data.frame` to capital letters (or small letters, if they are already capital.
- Change the `class` attribute to *list*. What happens?
- Change it now to any name you like. What happens now? What happens if you remove the class attribute

```
# Answer
# A. One possible way through attributes

attributes(my_df)
```

```
## $names
## [1] "A" "B" "C"
##
## $row.names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" ""
##
## $class
## [1] "data.frame"
```

```
attr(my_df, "names") <- letters[1:3]
attr(my_df, "row.names") <- LETTERS[1:12]
my_df
```

```
##    a  b  c
## A  1  1  1
## B  2  2  2
## C  3  3  3
## D  4  4  4
## E  5  5  5
## F  6  6  6
## G  7  7  7
## H  8  8  8
## I  9  9  9
## J 10 10 10
## K NA 11 11
## L NA NA 12
```

```
# Or through accessor functions

names(my_df) <- LETTERS[1:3]
row.names(my_df) <- letters[1:12]
my_df
```

```
##    A  B  C
## a  1  1  1
## b  2  2  2
## c  3  3  3
```

```
## d  4  4  4
## e  5  5  5
## f  6  6  6
## g  7  7  7
## h  8  8  8
## i  9  9  9
## j 10 10 10
## k NA 11 11
## l NA NA 12
```

```
# B.

attr(my_df, "class") <- "list"
my_df
```

```
## $A
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA
##
## $B
##  [1]  1  2  3  4  5  6  7  8  9 10 11 NA
##
## $C
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## attr(,"row.names")
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
## attr(,"class")
## [1] "list"
```

```
# Answer - the data.frame coerced to a list

# C
attr(my_df, "class") <- "Batman"
my_df
```

```
## $A
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA
##
## $B
##  [1]  1  2  3  4  5  6  7  8  9 10 11 NA
##
## $C
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## attr(,"row.names")
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
## attr(,"class")
## [1] "Batman"
```

```
# Answer - Nothing changes
```

1.7 Factors

- What is the difference between a Factor and a Vector?

- Create a vector of length 30 with three levels *Rita Repulsa, Lord Zedd* and *Rito Revolto* and equal length for each level
- What happens if you replace the second element of the vector with *Shredder*

```
# Answer
# A. A factor is a vector that can contain only predefined values, and is used
# to store categorical data. It is stored as an integer with a character string
# associated with each integer value

# B.

x <- gl(n=3, k=10, length=30, labels=c("Rita Repulsa", "Lord Zedd", "Rito Revolto"))
str(x)
```

```
##  Factor w/ 3 levels "Rita Repulsa",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
levels(x)
```

```
## [1] "Rita Repulsa" "Lord Zedd"    "Rito Revolto"
```

```
attributes(x)
```

```
## $levels
## [1] "Rita Repulsa" "Lord Zedd"    "Rito Revolto"
##
## $class
## [1] "factor"
```

```
# C
x[2] <- "Shredder"
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "Shredder"): invalid factor
## level, NA generated
```

```
# It doesn't work. We get the error 'NA generated'
```

1.8 More fun with factors

```
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))
f2 <- rev(factor(letters))
f3 <- factor(letters, levels = rev(letters))
```

The function `rev` reverses the order of an orderable object. What is the difference between f1, f2 and f3? Why?

```
# Answer
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))

# f1 goes from a to z and when we apply the levels(f1), z will become 1 and a=26

f2 <- rev(factor(letters))

# f2 goes from z to a. but the levels are not changed.

f3 <- factor(letters, levels = rev(letters))
```

```
# f3 goes from a - z, but the underlying encoding goes from z = 1 to a = 26.
# We create the vector with the letters a to z BUT the mapped integer structure
# 26 to 1. Hence the levels but not the vector are reversed

f3
```

```
##  [1] a b c d e f g h i j k l m n o p q r s t u v w x y z
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

```
# Reversing f3 will give f1

rev(f3)
```

```
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

1.9 Creating data.frames

Create a data.frame with 26 rows like this: Only the first and the last six rows are shown. Hint: Instead of the workaround with list you can also use simply `data.frame(column_name = column_vector, ...)`

```
aa <- seq(1:26)
bb <- seq(from=4, to=4*26, by=4)
cc <- rep(seq(1, 26, 2), each=2)
df <- data.frame(V1 = aa, V2 = bb, V3 = letters[cc])
head(df)
```

```
##   V1 V2 V3
## 1  1  4  a
## 2  2  8  a
## 3  3 12  c
## 4  4 16  c
## 5  5 20  e
## 6  6 24  e
```

```
tail(df)
```

```
##    V1  V2 V3
## 21 21  84  u
## 22 22  88  u
## 23 23  92  w
## 24 24  96  w
## 25 25 100  y
## 26 26 104  y
```

1.10 Combining `data.frames`

Now take the previous data.frame from Question 10 and reproduce the following `data.frame`. Only the first and the last six rows are shown **Hint:** In order to combine to data.frames by column you can use `cbind(df1, df2, ...)`

help(cbind)

```
df[,1] <- NULL
dd <- rev(rep(seq(1, 26, 2), each = 2))
```

```r
ee <- seq(0, 1.6, length.out = 26)
df2 <- data.frame(V4 = dd, V5 = ee)
binded_df <- cbind(df2, df)
head(binded_df)
```

```
##   V4    V5 V2 V3
## 1 25 0.000  4  a
## 2 25 0.064  8  a
## 3 23 0.128 12  c
## 4 23 0.192 16  c
## 5 21 0.256 20  e
## 6 21 0.320 24  e
```

```r
tail(binded_df)
```

```
##    V4    V5  V2 V3
## 21  5 1.280  84  u
## 22  5 1.344  88  u
## 23  3 1.408  92  w
## 24  3 1.472  96  w
## 25  1 1.536 100  y
## 26  1 1.600 104  y
```

1.11 Computation on data.frames

Create the data.frame *df* with `df <- as.data.frame(matrix(runif(9e6), 3e3, 3e3))` This will create a data.frame with 3000 columns and rows and a total of 9mil values.

Now compute the sum of any row, then compute the sum of any column. Measure the time for both operations. Why are the times different, although the size is the same?

- **Hint1:** The time is measured with the function `system.time(my_function_call())`, e.g: `system.time(mean(my_vector))`
- **Hint2:** The sum can be computed with the sum function `sum(my_vector)`
- **Hint2:** Columns and rows are selected by single brackets. Rows: `df[row_number,]`, Columns: `df[,col_number]`

```r
# Answer

df <- as.data.frame(matrix(runif(9e6), 3e3, 3e3))

# rows
system.time(res <- sum(df[1,]))
```

```
##    user  system elapsed
##   0.024   0.000   0.023
```

```r
res
```

```
## [1] 1489.909
```

```r
# columngs
system.time(res2 <- sum(df[,1]))
```

```
##    user  system elapsed
##       0       0       0
```

```
res2
```

```
## [1] 1490.652
```

```
# Look at the structure of the objects over which we are computing the sum
# Column
str(df[1,])
```

```
## 'data.frame':    1 obs. of  3000 variables:
##  $ V1   : num 0.602
##  $ V2   : num 0.993
##  $ V3   : num 0.795
##  $ V4   : num 0.542
##  $ V5   : num 0.84
##  $ V6   : num 0.433
##  $ V7   : num 0.182
##  $ V8   : num 0.121
##  $ V9   : num 0.432
##  $ V10  : num 0.296
##  $ V11  : num 0.285
##  $ V12  : num 0.843
##  $ V13  : num 0.134
##  $ V14  : num 0.991
##  $ V15  : num 0.0529
##  $ V16  : num 0.201
##  $ V17  : num 0.603
##  $ V18  : num 0.336
##  $ V19  : num 0.805
##  $ V20  : num 0.858
##  $ V21  : num 0.668
##  $ V22  : num 0.87
##  $ V23  : num 0.912
##  $ V24  : num 0.118
##  $ V25  : num 0.808
##  $ V26  : num 0.868
##  $ V27  : num 0.199
##  $ V28  : num 0.612
##  $ V29  : num 0.641
##  $ V30  : num 0.827
##  $ V31  : num 0.69
##  $ V32  : num 0.989
##  $ V33  : num 0.76
##  $ V34  : num 0.751
##  $ V35  : num 0.549
##  $ V36  : num 0.661
##  $ V37  : num 0.542
##  $ V38  : num 0.142
##  $ V39  : num 0.959
##  $ V40  : num 0.386
##  $ V41  : num 0.836
##  $ V42  : num 0.324
##  $ V43  : num 0.664
##  $ V44  : num 0.765
##  $ V45  : num 0.0155
##  $ V46  : num 0.311
```

```
##  $ V47  : num 0.231
##  $ V48  : num 0.865
##  $ V49  : num 0.54
##  $ V50  : num 0.299
##  $ V51  : num 0.7
##  $ V52  : num 0.679
##  $ V53  : num 0.924
##  $ V54  : num 0.0106
##  $ V55  : num 0.9
##  $ V56  : num 0.972
##  $ V57  : num 0.27
##  $ V58  : num 0.661
##  $ V59  : num 0.204
##  $ V60  : num 0.951
##  $ V61  : num 0.822
##  $ V62  : num 0.208
##  $ V63  : num 0.359
##  $ V64  : num 0.997
##  $ V65  : num 0.628
##  $ V66  : num 0.757
##  $ V67  : num 0.272
##  $ V68  : num 0.571
##  $ V69  : num 0.668
##  $ V70  : num 0.867
##  $ V71  : num 0.641
##  $ V72  : num 0.272
##  $ V73  : num 0.59
##  $ V74  : num 0.368
##  $ V75  : num 0.529
##  $ V76  : num 0.0851
##  $ V77  : num 0.352
##  $ V78  : num 0.135
##  $ V79  : num 0.708
##  $ V80  : num 0.668
##  $ V81  : num 0.0696
##  $ V82  : num 0.0209
##  $ V83  : num 0.705
##  $ V84  : num 0.523
##  $ V85  : num 0.984
##  $ V86  : num 0.777
##  $ V87  : num 0.806
##  $ V88  : num 0.0852
##  $ V89  : num 0.763
##  $ V90  : num 0.558
##  $ V91  : num 0.632
##  $ V92  : num 0.659
##  $ V93  : num 0.918
##  $ V94  : num 0.547
##  $ V95  : num 0.613
##  $ V96  : num 0.00087
##  $ V97  : num 0.614
##  $ V98  : num 0.254
##  $ V99  : num 0.507
##    [list output truncated]
```

```r
# Row
str(df[,1])
```

```
##  num [1:3000] 0.6025 0.8252 0.2185 0.1491 0.0572 ...
```

```r
# As we can see the extracted column is a numeric vector. But the extracted
# row is a list. Under the hood the sum function is iterating in C/Fortran
# over the specific structure. Iterating over a native array of doubles is
# faster, than iterating over a structure, where at each position, the value
# has to be retrieved from an object possibly strored somewhere further away
# in memory.
```

1.12 Missing Values

- If NA is just a placeholder for a missing value of the same type and Infinity is of type double, why is Infinity plus NA not Infinity?

Hint:

```r
paste(paste(rep((Inf - Inf), 20), collapse = ""), "Batman!")
```

```
## [1] "NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!"
```

```r
# Answer:
# Because infinity is not a well defined number, but a concept with
# the type 'double' in R. Adding or subtracting any number from infinity
# will give infinity. However NA is placeholder for any value of the same
# type and therefore also for infinity. As infinity plus/minus infinity
# is not defined, adding NA to infinity can theoretically lead to
# Nan (not a number). Therefore Inf + NA will produce NA.

Inf + NA
```

```
## [1] NA
```

```r
Inf + 1
```

```
## [1] Inf
```

```r
Inf - Inf
```

```
## [1] NaN
```

---

2 Advanced R Data Structures and Mathematical Operations

2.1 Basic data structures

How would you create a 3 by 4 matrix that contains the numbers 1 to 12 and then convert it into a data frame?

```r
# Answer

x <- matrix(1:12, 3,4)
x <- as.data.frame(x)
x
```

```
##   V1 V2 V3 V4
## 1  1  4  7 10
## 2  2  5  8 11
## 3  3  6  9 12
```

2.2

Please use the data frame you created in the first question for the next 5 questions. How would you select the second row elements at second and fourth column ?

```
x <- data.frame(matrix(1:12, 3, 4))
x[2, c(2,4)]
```

```
##   X2 X4
## 2  5 11
```

2.3

How would you assign zero to the elements at row 2 which are greater than 4?

```
x <- data.frame(matrix(1:12, 3, 4))
x[2, x[2,]>4] <- 0
x
```

```
##   X1 X2 X3 X4
## 1  1  4  7 10
## 2  2  0  0  0
## 3  3  6  9 12
```

2.4

How do you set the rownames to “row1”, “row2”, “row3” and column names to “col1”, “col2”, “col3” and “col4” ? (hint:use function “paste0”)

```
x <- data.frame(matrix(1:12, 3, 4))
rownames(x) <- paste0("row", 1:3)
colnames(x) <- paste0("col", 1:4)
x
```

```
##      col1 col2 col3 col4
## row1    1    4    7   10
## row2    2    5    8   11
## row3    3    6    9   12
```

2.5

How do you assign 0 to all elements in columns “col3” and “col4” by using paste0 function ?

```
x <- data.frame(matrix(1:12, 3, 4))
colnames(x) <- paste0("col", 1:4)
x[, paste0(0, 3:4)] <- 0
x
```

```
##   col1 col2 col3 col4 03 04
## 1    1    4    7   10  0  0
## 2    2    5    8   11  0  0
## 3    3    6    9   12  0  0
```

2.6 lapply()

How do you get the numbers whose mod 2 is 0

- by using lapply() function
- by subsetting the data frame directly?

```
x <- data.frame(matrix(1:12), 3, 4)

lapply(x, function(a) a[(a %% 2) == 0])
```

```
## $matrix.1.12.
## [1]  2  4  6  8 10 12
##
## $X3
## numeric(0)
##
## $X4
##  [1] 4 4 4 4 4 4 4 4 4 4 4 4
```

```
x[x %% 2 == 0]
```

```
##  [1]  2  4  6  8 10 12  4  4  4  4  4  4  4  4  4  4  4  4
```

2.7

Considering `x <- c("a"=1, "b"=2, "c"=3, "d"=4, "e"=5)`, show how to select the third and fifth elements of x by using positive integers, negative integers, a logical vector, and a character vector.

```
x <- c("a"=1, "b"=2, "c"=3, "d"=4, "e"=5)
x[c(3,5)]
```

```
## c e
## 3 5
```

```
x[-c(1,2,4)]
```

```
## c e
## 3 5
```

```
x[c(F,F,T,F,T)]
```

```
## c e
## 3 5
```

```
x[c("c", "e")]
```

```
## c e
## 3 5
```

2.8

Why are `vals[c(2, 5)]` and `vals[2, 5]` different where `vals <- outer(1:5, 1:5, FUN = "/")`? How would you select fifth and nineth elements of vals by the use of a matrix?

```
vals <- outer(1:5, 1:5, FUN = "/")

# Because when you subset matrix with a vector, the 2d matrix behaves
# like a vector and vals[c(2, 5)] returns the elements at indices 2
# and 5 in column-major order. vals[2, 5] returns the element at row 2, column 5.

select <- matrix(ncol=2, byrow=TRUE, c(5,1,4,2))

vals[select]
```

```
## [1] 5 2
```

2.9

Consider df <- data.frame(a=paste("Point_", 1:20), b=rep(1:4, each = 2, len = 20), c= seq(1,40,length.out = 20), stringsAsFactors = F). Assign "Point_undefined" to column a of all rows of df where column b > 1 and column c > 21 ? What is the reason of the different result that you get if you do the same operation with df being created with option stringsAsFactors = T ?

2.10

Assume x <- matrix(1:20, ncol=2). What is the difference between x[1, , drop = T] and x[1, , drop = F]? Now let y <- as.data.frame(x). What is the difference between y[,1] , y[[1]] and y[1]

2.11

What is the difference between x["b"] <- list(NULL) and x["b"] <- NULL where x <- list(a = c(1:5), b = c(12:15))?

2.12

Assume you have a lookup table as lookup <- c(a = "sun", b = "rain", c="wind", u = NA). How would you generate the weekly weather predictions c("sun","sun","rain", NA, "rain", "rain", "wind") out of this lookup table?

2.13

Now assume the weather in winter lookup table is a data frame as below and we have the predictions for the next week as stored in weeklyCast. How would you create "weeklyTable" by the use of rownames function? How would you create it by the use of match function? How would you order the rows of lookup table by desc column?

2.14

Consider the bigDF data frame which has 1500 columns and rows. How would you select the even numbered columns named such as "Column_2", "Column_4", etc.? How would you select all the columns other than column 76? How would you assign 1 to 500 randomly selected diagonal indices? How can you retrieve the row and column indices of the elements which has been assigned 1? How would you select rows where columns Column_1 or Column_2 are 1 by using the subset() function?

2.15

Assume x <- 1:20 %% 2 == 0 and y <- 1:20 %% 5 == 0 . What are the indices of the elements that are True for both x and y? What are the indices of the elements that are True for either x or y, or both?

3 Data Import

3.1 Flat File - Q1

A csv file has numbers as column names in the first row, i.e. IDs to randomize persons. Which parameter of read.table() needs to be adjusted to read the column names as they are in the csv?

```
tmp_tidy_table <- "1_colname, 2_colname, 3_colname
3,4,5
a,b,c"
tmp_tidy_table
```

```
## [1] "1_colname, 2_colname, 3_colname\n3,4,5\na,b,c"
```

```
read.csv(text=tmp_tidy_table)
```

```
##   X1_colname X2_colname X3_colname
## 1          3          4          5
## 2          a          b          c
```

```
# Parameter `check.names`: a logical, tests for synatactically valid varaible
# names

tidy_text_df <- read.csv(text=tmp_tidy_table, check.names = FALSE)
tidy_text_df
```

```
##   1_colname 2_colname 3_colname
## 1         3         4         5
## 2         a         b         c
```

3.2 Flat File - Q2

How to read the following table to have the `identical()` information as in `tidy_txt_df` from question above?

```
tmp_messy_table <- "# This line is just useless info

1_colname,2_colname,3_colname
3,4,5
```

```
a,b,c"
```

```
# To have the identical information as in the previous table we have to check
# which lines are comments. We can do this with `comment.char` parameter.

messy_text_df <- read.csv(text = tmp_messy_table, comment.char = '#', check.names = F)
identical(messy_text_df, tidy_text_df)
```

```
## [1] TRUE
```

3.3 Flat File - Q3

Read the `hollywood.tsv` (not *.csv) file into a `data.table` R object. What is the problem with `fread()`?

```
file_holly_tab <- "extdata/hollywood.tsv"
holly <- as.data.table(read.delim(file_holly_tab), keep_row_names=T)
head(holly)
```

```
##                    Film   Genre Lead.Studio Audience..score..
## 1:           27 Dresses  Comedy         Fox                71
## 2: (500) Days of Summer  Comedy         Fox                81
## 3:   A Dangerous Method   Drama Independent                89
## 4:        A Serious Man   Drama   Universal                64
## 5:  Across the Universe Romance Independent                84
## 6:            Beginners  Comedy Independent                80
##    Profitability Rotten.Tomatoes.. Worldwide.Gross Year
## 1:     5.3436218                40      160.308654 2008
## 2:     8.0960000                87       60.720000 2009
## 3:     0.4486447                79        8.972895 2011
## 4:     4.3828571                89       30.680000 2009
## 5:     0.6526032                54       29.367143 2007
## 6:     4.4718750                84       14.310000 2011
```

```
class(holly)
```

```
## [1] "data.table" "data.frame"
```

```
holly_data_table <- fread(file_holly_tab, skip=1)
class(holly_data_table)
```

```
## [1] "data.table" "data.frame"
```

```
head(holly_data_table)
```

```
##    V1                   V2      V3          V4 V5        V6 V7         V8
## 1:  1           27 Dresses  Comedy         Fox 71 5.3436218 40 160.308654
## 2:  2 (500) Days of Summer  Comedy         Fox 81 8.0960000 87  60.720000
## 3:  3   A Dangerous Method   Drama Independent 89 0.4486447 79   8.972895
## 4:  4        A Serious Man   Drama   Universal 64 4.3828571 89  30.680000
## 5:  5  Across the Universe Romance Independent 84 0.6526032 54  29.367143
## 6:  6            Beginners  Comedy Independent 80 4.4718750 84  14.310000
##      V9
## 1: 2008
## 2: 2009
## 3: 2011
## 4: 2009
```

```
## 5: 2007
## 6: 2011
```

```
holly_cn <- c("ID", colnames(read.delim(file_holly_tab, nrows= 1)))
holly_cn
```

```
## [1] "ID"                "Film"              "Genre"
## [4] "Lead.Studio"       "Audience..score.." "Profitability"
## [7] "Rotten.Tomatoes.." "Worldwide.Gross"   "Year"
```

```
# setnames(holly, holly_cn)
# head(holly)
# Difference between them: We can keep row_names. data.frame has no row_names.
```

3.4 Flat File - Q4

Who was the oldest surviving passenger of the titanic accident (titanic.csv)? Tipp: `?subset`

```
tit_df <- read.csv("extdata/titanic.csv")
head(tit_df)
```

```
##   pclass survived                                            name    sex
## 1      1        1                   Allen, Miss. Elisabeth Walton female
## 2      1        1                  Allison, Master. Hudson Trevor   male
## 3      1        0                    Allison, Miss. Helen Loraine female
## 4      1        0            Allison, Mr. Hudson Joshua Creighton   male
## 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6      1        1                         Anderson, Mr. Harry   male
##     age sibsp parch ticket     fare   cabin embarked boat body
## 1 29.00     0     0  24160 211.3375      B5        S    2   NA
## 2  0.92     1     2 113781 151.5500 C22 C26        S   11   NA
## 3  2.00     1     2 113781 151.5500 C22 C26        S        NA
## 4 30.00     1     2 113781 151.5500 C22 C26        S       135
## 5 25.00     1     2 113781 151.5500 C22 C26        S        NA
## 6 48.00     0     0  19952  26.5500     E12        S    3   NA
##                         home.dest
## 1                     St Louis, MO
## 2 Montreal, PQ / Chesterville, ON
## 3 Montreal, PQ / Chesterville, ON
## 4 Montreal, PQ / Chesterville, ON
## 5 Montreal, PQ / Chesterville, ON
## 6                     New York, NY
```

```
survivor_name <- subset(tit_df, survived==1 & age==max(age, na.rm = T), name)
survivor_age <- subset(tit_df, survived==1 & age==max(age, na.rm = T), age)

survivor_name
```

```
##                                    name
## 15 Barkworth, Mr. Algernon Henry Wilson
```

```
survivor_age
```

```
##    age
## 15  80
```

3.5 Excel Questions - Q1

Read only Name, Type and Total columns for only the first 10 pokemons of the pokemon.xlsx file.

3.6 Excel Question - Q2

Which athlete won most bronze medals?

3.7 Excel Question - Q3

Are the columns Gender and Event_gender consistent?

3.8 Excel Question - Q4

Which country won most medals? Which country has the highest ratio of silver medals? Use the data in the country summary sheet starting at row 147.

3.9 Excel Questions

Which countries did participate, but without winning medals?

3.10 XML - Q1

Load the XML document plant_catalog.xml. Use XPath and DOM functions to find out all unique element names in the document.

Get all plants of zone 4 and transform the data into an R list.

3.11 XML - Q2

Read the tables HTML tables from the TUM website of dates for the winter term https://www.tum.de/en/studies/application-and-acceptance/dates-and-deadlines/ dates-and-deadlines-17/ into your workspace. When are the Christmas holidays?

3.12 JSON - Q1

Read the countries.json file. Which countries have common border with Jordan? Which country has the most neighbors?

3.13 JSON - Q2

Read this JSON file about projects funded by the world bank: world_bank.json.zip. Be aware, you might need to add syntax elements like "[" and "," to convert the file into textbook JSON format, i.e. readable by R. What was the most expensive project?

3.14 SQL - Q1

Use the extdata/Northwind.sl3 SQLite data base and retrieve a table that lists for all customers (name of the company, name of the contact person and city) all the products (name of the product) that they ordered. How many rows does this table have? Display the first 5 rows.

4 Grammar of graphics and plotting I

4.1 Q1

Match each chart type with the relationship it shows best.

1. shows distribution and quantiles, especially useful when comparing distributions.
2. highlights individual values, supports comparison and can show rankings or deviations categories and totals
3. shows overall changes and patterns, usually over intervals of time
4. shows relationship between two continues variables.

Options: bar chart, line chart, scatterplot, boxplot

```
# Answer
#
# 1 -> boxplot
# 2 -> bar chart
# 3 -> line chart
# 4. -> scatterplot
```

4.2 Q2

`Iris` is a classical dataset in machine learning literature, was first introduced by R.A. Fisher in his 1936 paper. Load the iris data into your R environment. What is the dimension of the dataset and what kind of data type does each column has?

```
dim(iris)
```

```
## [1] 150   5
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

4.3 Q3

How are the lengths and widths of sepals and petals distributed? How would you visualize them. Describe what you see. Hint: `facet_wrap(~variable)`.

```
iris_melt <- melt(iris, id.var=c("Species"))
iris_melt
```

```
##        Species     variable value
## 1       setosa Sepal.Length   5.1
## 2       setosa Sepal.Length   4.9
## 3       setosa Sepal.Length   4.7
## 4       setosa Sepal.Length   4.6
## 5       setosa Sepal.Length   5.0
## 6       setosa Sepal.Length   5.4
## 7       setosa Sepal.Length   4.6
## 8       setosa Sepal.Length   5.0
## 9       setosa Sepal.Length   4.4
## 10      setosa Sepal.Length   4.9
## 11      setosa Sepal.Length   5.4
## 12      setosa Sepal.Length   4.8
## 13      setosa Sepal.Length   4.8
## 14      setosa Sepal.Length   4.3
## 15      setosa Sepal.Length   5.8
## 16      setosa Sepal.Length   5.7
## 17      setosa Sepal.Length   5.4
## 18      setosa Sepal.Length   5.1
## 19      setosa Sepal.Length   5.7
## 20      setosa Sepal.Length   5.1
## 21      setosa Sepal.Length   5.4
## 22      setosa Sepal.Length   5.1
## 23      setosa Sepal.Length   4.6
## 24      setosa Sepal.Length   5.1
## 25      setosa Sepal.Length   4.8
## 26      setosa Sepal.Length   5.0
## 27      setosa Sepal.Length   5.0
## 28      setosa Sepal.Length   5.2
## 29      setosa Sepal.Length   5.2
## 30      setosa Sepal.Length   4.7
## 31      setosa Sepal.Length   4.8
## 32      setosa Sepal.Length   5.4
## 33      setosa Sepal.Length   5.2
## 34      setosa Sepal.Length   5.5
## 35      setosa Sepal.Length   4.9
## 36      setosa Sepal.Length   5.0
## 37      setosa Sepal.Length   5.5
## 38      setosa Sepal.Length   4.9
## 39      setosa Sepal.Length   4.4
## 40      setosa Sepal.Length   5.1
## 41      setosa Sepal.Length   5.0
## 42      setosa Sepal.Length   4.5
## 43      setosa Sepal.Length   4.4
## 44      setosa Sepal.Length   5.0
## 45      setosa Sepal.Length   5.1
## 46      setosa Sepal.Length   4.8
## 47      setosa Sepal.Length   5.1
## 48      setosa Sepal.Length   4.6
## 49      setosa Sepal.Length   5.3
## 50      setosa Sepal.Length   5.0
## 51  versicolor Sepal.Length   7.0
## 52  versicolor Sepal.Length   6.4
## 53  versicolor Sepal.Length   6.9
```

```
## 54  versicolor Sepal.Length   5.5
## 55  versicolor Sepal.Length   6.5
## 56  versicolor Sepal.Length   5.7
## 57  versicolor Sepal.Length   6.3
## 58  versicolor Sepal.Length   4.9
## 59  versicolor Sepal.Length   6.6
## 60  versicolor Sepal.Length   5.2
## 61  versicolor Sepal.Length   5.0
## 62  versicolor Sepal.Length   5.9
## 63  versicolor Sepal.Length   6.0
## 64  versicolor Sepal.Length   6.1
## 65  versicolor Sepal.Length   5.6
## 66  versicolor Sepal.Length   6.7
## 67  versicolor Sepal.Length   5.6
## 68  versicolor Sepal.Length   5.8
## 69  versicolor Sepal.Length   6.2
## 70  versicolor Sepal.Length   5.6
## 71  versicolor Sepal.Length   5.9
## 72  versicolor Sepal.Length   6.1
## 73  versicolor Sepal.Length   6.3
## 74  versicolor Sepal.Length   6.1
## 75  versicolor Sepal.Length   6.4
## 76  versicolor Sepal.Length   6.6
## 77  versicolor Sepal.Length   6.8
## 78  versicolor Sepal.Length   6.7
## 79  versicolor Sepal.Length   6.0
## 80  versicolor Sepal.Length   5.7
## 81  versicolor Sepal.Length   5.5
## 82  versicolor Sepal.Length   5.5
## 83  versicolor Sepal.Length   5.8
## 84  versicolor Sepal.Length   6.0
## 85  versicolor Sepal.Length   5.4
## 86  versicolor Sepal.Length   6.0
## 87  versicolor Sepal.Length   6.7
## 88  versicolor Sepal.Length   6.3
## 89  versicolor Sepal.Length   5.6
## 90  versicolor Sepal.Length   5.5
## 91  versicolor Sepal.Length   5.5
## 92  versicolor Sepal.Length   6.1
## 93  versicolor Sepal.Length   5.8
## 94  versicolor Sepal.Length   5.0
## 95  versicolor Sepal.Length   5.6
## 96  versicolor Sepal.Length   5.7
## 97  versicolor Sepal.Length   5.7
## 98  versicolor Sepal.Length   6.2
## 99  versicolor Sepal.Length   5.1
## 100 versicolor Sepal.Length   5.7
## 101  virginica Sepal.Length   6.3
## 102  virginica Sepal.Length   5.8
## 103  virginica Sepal.Length   7.1
## 104  virginica Sepal.Length   6.3
## 105  virginica Sepal.Length   6.5
## 106  virginica Sepal.Length   7.6
## 107  virginica Sepal.Length   4.9
```

```
## 108  virginica Sepal.Length   7.3
## 109  virginica Sepal.Length   6.7
## 110  virginica Sepal.Length   7.2
## 111  virginica Sepal.Length   6.5
## 112  virginica Sepal.Length   6.4
## 113  virginica Sepal.Length   6.8
## 114  virginica Sepal.Length   5.7
## 115  virginica Sepal.Length   5.8
## 116  virginica Sepal.Length   6.4
## 117  virginica Sepal.Length   6.5
## 118  virginica Sepal.Length   7.7
## 119  virginica Sepal.Length   7.7
## 120  virginica Sepal.Length   6.0
## 121  virginica Sepal.Length   6.9
## 122  virginica Sepal.Length   5.6
## 123  virginica Sepal.Length   7.7
## 124  virginica Sepal.Length   6.3
## 125  virginica Sepal.Length   6.7
## 126  virginica Sepal.Length   7.2
## 127  virginica Sepal.Length   6.2
## 128  virginica Sepal.Length   6.1
## 129  virginica Sepal.Length   6.4
## 130  virginica Sepal.Length   7.2
## 131  virginica Sepal.Length   7.4
## 132  virginica Sepal.Length   7.9
## 133  virginica Sepal.Length   6.4
## 134  virginica Sepal.Length   6.3
## 135  virginica Sepal.Length   6.1
## 136  virginica Sepal.Length   7.7
## 137  virginica Sepal.Length   6.3
## 138  virginica Sepal.Length   6.4
## 139  virginica Sepal.Length   6.0
## 140  virginica Sepal.Length   6.9
## 141  virginica Sepal.Length   6.7
## 142  virginica Sepal.Length   6.9
## 143  virginica Sepal.Length   5.8
## 144  virginica Sepal.Length   6.8
## 145  virginica Sepal.Length   6.7
## 146  virginica Sepal.Length   6.7
## 147  virginica Sepal.Length   6.3
## 148  virginica Sepal.Length   6.5
## 149  virginica Sepal.Length   6.2
## 150  virginica Sepal.Length   5.9
## 151     setosa  Sepal.Width   3.5
## 152     setosa  Sepal.Width   3.0
## 153     setosa  Sepal.Width   3.2
## 154     setosa  Sepal.Width   3.1
## 155     setosa  Sepal.Width   3.6
## 156     setosa  Sepal.Width   3.9
## 157     setosa  Sepal.Width   3.4
## 158     setosa  Sepal.Width   3.4
## 159     setosa  Sepal.Width   2.9
## 160     setosa  Sepal.Width   3.1
## 161     setosa  Sepal.Width   3.7
```

```
## 162     setosa  Sepal.Width   3.4
## 163     setosa  Sepal.Width   3.0
## 164     setosa  Sepal.Width   3.0
## 165     setosa  Sepal.Width   4.0
## 166     setosa  Sepal.Width   4.4
## 167     setosa  Sepal.Width   3.9
## 168     setosa  Sepal.Width   3.5
## 169     setosa  Sepal.Width   3.8
## 170     setosa  Sepal.Width   3.8
## 171     setosa  Sepal.Width   3.4
## 172     setosa  Sepal.Width   3.7
## 173     setosa  Sepal.Width   3.6
## 174     setosa  Sepal.Width   3.3
## 175     setosa  Sepal.Width   3.4
## 176     setosa  Sepal.Width   3.0
## 177     setosa  Sepal.Width   3.4
## 178     setosa  Sepal.Width   3.5
## 179     setosa  Sepal.Width   3.4
## 180     setosa  Sepal.Width   3.2
## 181     setosa  Sepal.Width   3.1
## 182     setosa  Sepal.Width   3.4
## 183     setosa  Sepal.Width   4.1
## 184     setosa  Sepal.Width   4.2
## 185     setosa  Sepal.Width   3.1
## 186     setosa  Sepal.Width   3.2
## 187     setosa  Sepal.Width   3.5
## 188     setosa  Sepal.Width   3.6
## 189     setosa  Sepal.Width   3.0
## 190     setosa  Sepal.Width   3.4
## 191     setosa  Sepal.Width   3.5
## 192     setosa  Sepal.Width   2.3
## 193     setosa  Sepal.Width   3.2
## 194     setosa  Sepal.Width   3.5
## 195     setosa  Sepal.Width   3.8
## 196     setosa  Sepal.Width   3.0
## 197     setosa  Sepal.Width   3.8
## 198     setosa  Sepal.Width   3.2
## 199     setosa  Sepal.Width   3.7
## 200     setosa  Sepal.Width   3.3
## 201 versicolor  Sepal.Width   3.2
## 202 versicolor  Sepal.Width   3.2
## 203 versicolor  Sepal.Width   3.1
## 204 versicolor  Sepal.Width   2.3
## 205 versicolor  Sepal.Width   2.8
## 206 versicolor  Sepal.Width   2.8
## 207 versicolor  Sepal.Width   3.3
## 208 versicolor  Sepal.Width   2.4
## 209 versicolor  Sepal.Width   2.9
## 210 versicolor  Sepal.Width   2.7
## 211 versicolor  Sepal.Width   2.0
## 212 versicolor  Sepal.Width   3.0
## 213 versicolor  Sepal.Width   2.2
## 214 versicolor  Sepal.Width   2.9
## 215 versicolor  Sepal.Width   2.9
```

```
## 216 versicolor  Sepal.Width   3.1
## 217 versicolor  Sepal.Width   3.0
## 218 versicolor  Sepal.Width   2.7
## 219 versicolor  Sepal.Width   2.2
## 220 versicolor  Sepal.Width   2.5
## 221 versicolor  Sepal.Width   3.2
## 222 versicolor  Sepal.Width   2.8
## 223 versicolor  Sepal.Width   2.5
## 224 versicolor  Sepal.Width   2.8
## 225 versicolor  Sepal.Width   2.9
## 226 versicolor  Sepal.Width   3.0
## 227 versicolor  Sepal.Width   2.8
## 228 versicolor  Sepal.Width   3.0
## 229 versicolor  Sepal.Width   2.9
## 230 versicolor  Sepal.Width   2.6
## 231 versicolor  Sepal.Width   2.4
## 232 versicolor  Sepal.Width   2.4
## 233 versicolor  Sepal.Width   2.7
## 234 versicolor  Sepal.Width   2.7
## 235 versicolor  Sepal.Width   3.0
## 236 versicolor  Sepal.Width   3.4
## 237 versicolor  Sepal.Width   3.1
## 238 versicolor  Sepal.Width   2.3
## 239 versicolor  Sepal.Width   3.0
## 240 versicolor  Sepal.Width   2.5
## 241 versicolor  Sepal.Width   2.6
## 242 versicolor  Sepal.Width   3.0
## 243 versicolor  Sepal.Width   2.6
## 244 versicolor  Sepal.Width   2.3
## 245 versicolor  Sepal.Width   2.7
## 246 versicolor  Sepal.Width   3.0
## 247 versicolor  Sepal.Width   2.9
## 248 versicolor  Sepal.Width   2.9
## 249 versicolor  Sepal.Width   2.5
## 250 versicolor  Sepal.Width   2.8
## 251  virginica  Sepal.Width   3.3
## 252  virginica  Sepal.Width   2.7
## 253  virginica  Sepal.Width   3.0
## 254  virginica  Sepal.Width   2.9
## 255  virginica  Sepal.Width   3.0
## 256  virginica  Sepal.Width   3.0
## 257  virginica  Sepal.Width   2.5
## 258  virginica  Sepal.Width   2.9
## 259  virginica  Sepal.Width   2.5
## 260  virginica  Sepal.Width   3.6
## 261  virginica  Sepal.Width   3.2
## 262  virginica  Sepal.Width   2.7
## 263  virginica  Sepal.Width   3.0
## 264  virginica  Sepal.Width   2.5
## 265  virginica  Sepal.Width   2.8
## 266  virginica  Sepal.Width   3.2
## 267  virginica  Sepal.Width   3.0
## 268  virginica  Sepal.Width   3.8
## 269  virginica  Sepal.Width   2.6
```

```
## 270  virginica  Sepal.Width   2.2
## 271  virginica  Sepal.Width   3.2
## 272  virginica  Sepal.Width   2.8
## 273  virginica  Sepal.Width   2.8
## 274  virginica  Sepal.Width   2.7
## 275  virginica  Sepal.Width   3.3
## 276  virginica  Sepal.Width   3.2
## 277  virginica  Sepal.Width   2.8
## 278  virginica  Sepal.Width   3.0
## 279  virginica  Sepal.Width   2.8
## 280  virginica  Sepal.Width   3.0
## 281  virginica  Sepal.Width   2.8
## 282  virginica  Sepal.Width   3.8
## 283  virginica  Sepal.Width   2.8
## 284  virginica  Sepal.Width   2.8
## 285  virginica  Sepal.Width   2.6
## 286  virginica  Sepal.Width   3.0
## 287  virginica  Sepal.Width   3.4
## 288  virginica  Sepal.Width   3.1
## 289  virginica  Sepal.Width   3.0
## 290  virginica  Sepal.Width   3.1
## 291  virginica  Sepal.Width   3.1
## 292  virginica  Sepal.Width   3.1
## 293  virginica  Sepal.Width   2.7
## 294  virginica  Sepal.Width   3.2
## 295  virginica  Sepal.Width   3.3
## 296  virginica  Sepal.Width   3.0
## 297  virginica  Sepal.Width   2.5
## 298  virginica  Sepal.Width   3.0
## 299  virginica  Sepal.Width   3.4
## 300  virginica  Sepal.Width   3.0
## 301     setosa Petal.Length   1.4
## 302     setosa Petal.Length   1.4
## 303     setosa Petal.Length   1.3
## 304     setosa Petal.Length   1.5
## 305     setosa Petal.Length   1.4
## 306     setosa Petal.Length   1.7
## 307     setosa Petal.Length   1.4
## 308     setosa Petal.Length   1.5
## 309     setosa Petal.Length   1.4
## 310     setosa Petal.Length   1.5
## 311     setosa Petal.Length   1.5
## 312     setosa Petal.Length   1.6
## 313     setosa Petal.Length   1.4
## 314     setosa Petal.Length   1.1
## 315     setosa Petal.Length   1.2
## 316     setosa Petal.Length   1.5
## 317     setosa Petal.Length   1.3
## 318     setosa Petal.Length   1.4
## 319     setosa Petal.Length   1.7
## 320     setosa Petal.Length   1.5
## 321     setosa Petal.Length   1.7
## 322     setosa Petal.Length   1.5
## 323     setosa Petal.Length   1.0
```

```
## 324     setosa Petal.Length   1.7
## 325     setosa Petal.Length   1.9
## 326     setosa Petal.Length   1.6
## 327     setosa Petal.Length   1.6
## 328     setosa Petal.Length   1.5
## 329     setosa Petal.Length   1.4
## 330     setosa Petal.Length   1.6
## 331     setosa Petal.Length   1.6
## 332     setosa Petal.Length   1.5
## 333     setosa Petal.Length   1.5
## 334     setosa Petal.Length   1.4
## 335     setosa Petal.Length   1.5
## 336     setosa Petal.Length   1.2
## 337     setosa Petal.Length   1.3
## 338     setosa Petal.Length   1.4
## 339     setosa Petal.Length   1.3
## 340     setosa Petal.Length   1.5
## 341     setosa Petal.Length   1.3
## 342     setosa Petal.Length   1.3
## 343     setosa Petal.Length   1.3
## 344     setosa Petal.Length   1.6
## 345     setosa Petal.Length   1.9
## 346     setosa Petal.Length   1.4
## 347     setosa Petal.Length   1.6
## 348     setosa Petal.Length   1.4
## 349     setosa Petal.Length   1.5
## 350     setosa Petal.Length   1.4
## 351 versicolor Petal.Length   4.7
## 352 versicolor Petal.Length   4.5
## 353 versicolor Petal.Length   4.9
## 354 versicolor Petal.Length   4.0
## 355 versicolor Petal.Length   4.6
## 356 versicolor Petal.Length   4.5
## 357 versicolor Petal.Length   4.7
## 358 versicolor Petal.Length   3.3
## 359 versicolor Petal.Length   4.6
## 360 versicolor Petal.Length   3.9
## 361 versicolor Petal.Length   3.5
## 362 versicolor Petal.Length   4.2
## 363 versicolor Petal.Length   4.0
## 364 versicolor Petal.Length   4.7
## 365 versicolor Petal.Length   3.6
## 366 versicolor Petal.Length   4.4
## 367 versicolor Petal.Length   4.5
## 368 versicolor Petal.Length   4.1
## 369 versicolor Petal.Length   4.5
## 370 versicolor Petal.Length   3.9
## 371 versicolor Petal.Length   4.8
## 372 versicolor Petal.Length   4.0
## 373 versicolor Petal.Length   4.9
## 374 versicolor Petal.Length   4.7
## 375 versicolor Petal.Length   4.3
## 376 versicolor Petal.Length   4.4
## 377 versicolor Petal.Length   4.8
```

```
## 378 versicolor Petal.Length   5.0
## 379 versicolor Petal.Length   4.5
## 380 versicolor Petal.Length   3.5
## 381 versicolor Petal.Length   3.8
## 382 versicolor Petal.Length   3.7
## 383 versicolor Petal.Length   3.9
## 384 versicolor Petal.Length   5.1
## 385 versicolor Petal.Length   4.5
## 386 versicolor Petal.Length   4.5
## 387 versicolor Petal.Length   4.7
## 388 versicolor Petal.Length   4.4
## 389 versicolor Petal.Length   4.1
## 390 versicolor Petal.Length   4.0
## 391 versicolor Petal.Length   4.4
## 392 versicolor Petal.Length   4.6
## 393 versicolor Petal.Length   4.0
## 394 versicolor Petal.Length   3.3
## 395 versicolor Petal.Length   4.2
## 396 versicolor Petal.Length   4.2
## 397 versicolor Petal.Length   4.2
## 398 versicolor Petal.Length   4.3
## 399 versicolor Petal.Length   3.0
## 400 versicolor Petal.Length   4.1
## 401  virginica Petal.Length   6.0
## 402  virginica Petal.Length   5.1
## 403  virginica Petal.Length   5.9
## 404  virginica Petal.Length   5.6
## 405  virginica Petal.Length   5.8
## 406  virginica Petal.Length   6.6
## 407  virginica Petal.Length   4.5
## 408  virginica Petal.Length   6.3
## 409  virginica Petal.Length   5.8
## 410  virginica Petal.Length   6.1
## 411  virginica Petal.Length   5.1
## 412  virginica Petal.Length   5.3
## 413  virginica Petal.Length   5.5
## 414  virginica Petal.Length   5.0
## 415  virginica Petal.Length   5.1
## 416  virginica Petal.Length   5.3
## 417  virginica Petal.Length   5.5
## 418  virginica Petal.Length   6.7
## 419  virginica Petal.Length   6.9
## 420  virginica Petal.Length   5.0
## 421  virginica Petal.Length   5.7
## 422  virginica Petal.Length   4.9
## 423  virginica Petal.Length   6.7
## 424  virginica Petal.Length   4.9
## 425  virginica Petal.Length   5.7
## 426  virginica Petal.Length   6.0
## 427  virginica Petal.Length   4.8
## 428  virginica Petal.Length   4.9
## 429  virginica Petal.Length   5.6
## 430  virginica Petal.Length   5.8
## 431  virginica Petal.Length   6.1
```

```
## 432  virginica Petal.Length   6.4
## 433  virginica Petal.Length   5.6
## 434  virginica Petal.Length   5.1
## 435  virginica Petal.Length   5.6
## 436  virginica Petal.Length   6.1
## 437  virginica Petal.Length   5.6
## 438  virginica Petal.Length   5.5
## 439  virginica Petal.Length   4.8
## 440  virginica Petal.Length   5.4
## 441  virginica Petal.Length   5.6
## 442  virginica Petal.Length   5.1
## 443  virginica Petal.Length   5.1
## 444  virginica Petal.Length   5.9
## 445  virginica Petal.Length   5.7
## 446  virginica Petal.Length   5.2
## 447  virginica Petal.Length   5.0
## 448  virginica Petal.Length   5.2
## 449  virginica Petal.Length   5.4
## 450  virginica Petal.Length   5.1
## 451     setosa  Petal.Width   0.2
## 452     setosa  Petal.Width   0.2
## 453     setosa  Petal.Width   0.2
## 454     setosa  Petal.Width   0.2
## 455     setosa  Petal.Width   0.2
## 456     setosa  Petal.Width   0.4
## 457     setosa  Petal.Width   0.3
## 458     setosa  Petal.Width   0.2
## 459     setosa  Petal.Width   0.2
## 460     setosa  Petal.Width   0.1
## 461     setosa  Petal.Width   0.2
## 462     setosa  Petal.Width   0.2
## 463     setosa  Petal.Width   0.1
## 464     setosa  Petal.Width   0.1
## 465     setosa  Petal.Width   0.2
## 466     setosa  Petal.Width   0.4
## 467     setosa  Petal.Width   0.4
## 468     setosa  Petal.Width   0.3
## 469     setosa  Petal.Width   0.3
## 470     setosa  Petal.Width   0.3
## 471     setosa  Petal.Width   0.2
## 472     setosa  Petal.Width   0.4
## 473     setosa  Petal.Width   0.2
## 474     setosa  Petal.Width   0.5
## 475     setosa  Petal.Width   0.2
## 476     setosa  Petal.Width   0.2
## 477     setosa  Petal.Width   0.4
## 478     setosa  Petal.Width   0.2
## 479     setosa  Petal.Width   0.2
## 480     setosa  Petal.Width   0.2
## 481     setosa  Petal.Width   0.2
## 482     setosa  Petal.Width   0.4
## 483     setosa  Petal.Width   0.1
## 484     setosa  Petal.Width   0.2
## 485     setosa  Petal.Width   0.2
```

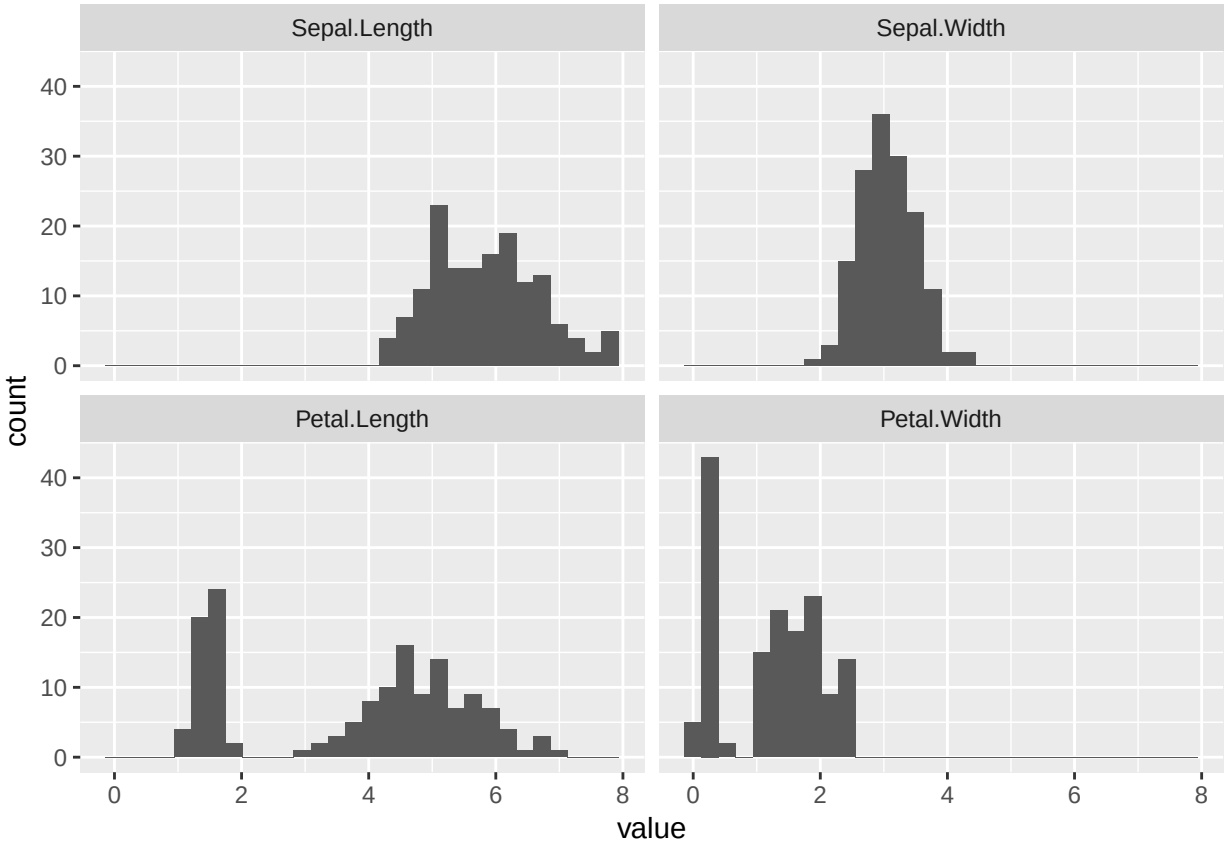
```
## 486     setosa  Petal.Width   0.2
## 487     setosa  Petal.Width   0.2
## 488     setosa  Petal.Width   0.1
## 489     setosa  Petal.Width   0.2
## 490     setosa  Petal.Width   0.2
## 491     setosa  Petal.Width   0.3
## 492     setosa  Petal.Width   0.3
## 493     setosa  Petal.Width   0.2
## 494     setosa  Petal.Width   0.6
## 495     setosa  Petal.Width   0.4
## 496     setosa  Petal.Width   0.3
## 497     setosa  Petal.Width   0.2
## 498     setosa  Petal.Width   0.2
## 499     setosa  Petal.Width   0.2
## 500     setosa  Petal.Width   0.2
## 501 versicolor  Petal.Width   1.4
## 502 versicolor  Petal.Width   1.5
## 503 versicolor  Petal.Width   1.5
## 504 versicolor  Petal.Width   1.3
## 505 versicolor  Petal.Width   1.5
## 506 versicolor  Petal.Width   1.3
## 507 versicolor  Petal.Width   1.6
## 508 versicolor  Petal.Width   1.0
## 509 versicolor  Petal.Width   1.3
## 510 versicolor  Petal.Width   1.4
## 511 versicolor  Petal.Width   1.0
## 512 versicolor  Petal.Width   1.5
## 513 versicolor  Petal.Width   1.0
## 514 versicolor  Petal.Width   1.4
## 515 versicolor  Petal.Width   1.3
## 516 versicolor  Petal.Width   1.4
## 517 versicolor  Petal.Width   1.5
## 518 versicolor  Petal.Width   1.0
## 519 versicolor  Petal.Width   1.5
## 520 versicolor  Petal.Width   1.1
## 521 versicolor  Petal.Width   1.8
## 522 versicolor  Petal.Width   1.3
## 523 versicolor  Petal.Width   1.5
## 524 versicolor  Petal.Width   1.2
## 525 versicolor  Petal.Width   1.3
## 526 versicolor  Petal.Width   1.4
## 527 versicolor  Petal.Width   1.4
## 528 versicolor  Petal.Width   1.7
## 529 versicolor  Petal.Width   1.5
## 530 versicolor  Petal.Width   1.0
## 531 versicolor  Petal.Width   1.1
## 532 versicolor  Petal.Width   1.0
## 533 versicolor  Petal.Width   1.2
## 534 versicolor  Petal.Width   1.6
## 535 versicolor  Petal.Width   1.5
## 536 versicolor  Petal.Width   1.6
## 537 versicolor  Petal.Width   1.5
## 538 versicolor  Petal.Width   1.3
## 539 versicolor  Petal.Width   1.3
```

```
## 540 versicolor  Petal.Width   1.3
## 541 versicolor  Petal.Width   1.2
## 542 versicolor  Petal.Width   1.4
## 543 versicolor  Petal.Width   1.2
## 544 versicolor  Petal.Width   1.0
## 545 versicolor  Petal.Width   1.3
## 546 versicolor  Petal.Width   1.2
## 547 versicolor  Petal.Width   1.3
## 548 versicolor  Petal.Width   1.3
## 549 versicolor  Petal.Width   1.1
## 550 versicolor  Petal.Width   1.3
## 551  virginica  Petal.Width   2.5
## 552  virginica  Petal.Width   1.9
## 553  virginica  Petal.Width   2.1
## 554  virginica  Petal.Width   1.8
## 555  virginica  Petal.Width   2.2
## 556  virginica  Petal.Width   2.1
## 557  virginica  Petal.Width   1.7
## 558  virginica  Petal.Width   1.8
## 559  virginica  Petal.Width   1.8
## 560  virginica  Petal.Width   2.5
## 561  virginica  Petal.Width   2.0
## 562  virginica  Petal.Width   1.9
## 563  virginica  Petal.Width   2.1
## 564  virginica  Petal.Width   2.0
## 565  virginica  Petal.Width   2.4
## 566  virginica  Petal.Width   2.3
## 567  virginica  Petal.Width   1.8
## 568  virginica  Petal.Width   2.2
## 569  virginica  Petal.Width   2.3
## 570  virginica  Petal.Width   1.5
## 571  virginica  Petal.Width   2.3
## 572  virginica  Petal.Width   2.0
## 573  virginica  Petal.Width   2.0
## 574  virginica  Petal.Width   1.8
## 575  virginica  Petal.Width   2.1
## 576  virginica  Petal.Width   1.8
## 577  virginica  Petal.Width   1.8
## 578  virginica  Petal.Width   1.8
## 579  virginica  Petal.Width   2.1
## 580  virginica  Petal.Width   1.6
## 581  virginica  Petal.Width   1.9
## 582  virginica  Petal.Width   2.0
## 583  virginica  Petal.Width   2.2
## 584  virginica  Petal.Width   1.5
## 585  virginica  Petal.Width   1.4
## 586  virginica  Petal.Width   2.3
## 587  virginica  Petal.Width   2.4
## 588  virginica  Petal.Width   1.8
## 589  virginica  Petal.Width   1.8
## 590  virginica  Petal.Width   2.1
## 591  virginica  Petal.Width   2.4
## 592  virginica  Petal.Width   2.3
## 593  virginica  Petal.Width   1.9
```

```
## 594  virginica  Petal.Width   2.3
## 595  virginica  Petal.Width   2.5
## 596  virginica  Petal.Width   2.3
## 597  virginica  Petal.Width   1.9
## 598  virginica  Petal.Width   2.0
## 599  virginica  Petal.Width   2.3
## 600  virginica  Petal.Width   1.8
```

```
iris_melt %>%
  ggplot(aes(value)) +
  geom_histogram() +
  facet_wrap(~variable)
```

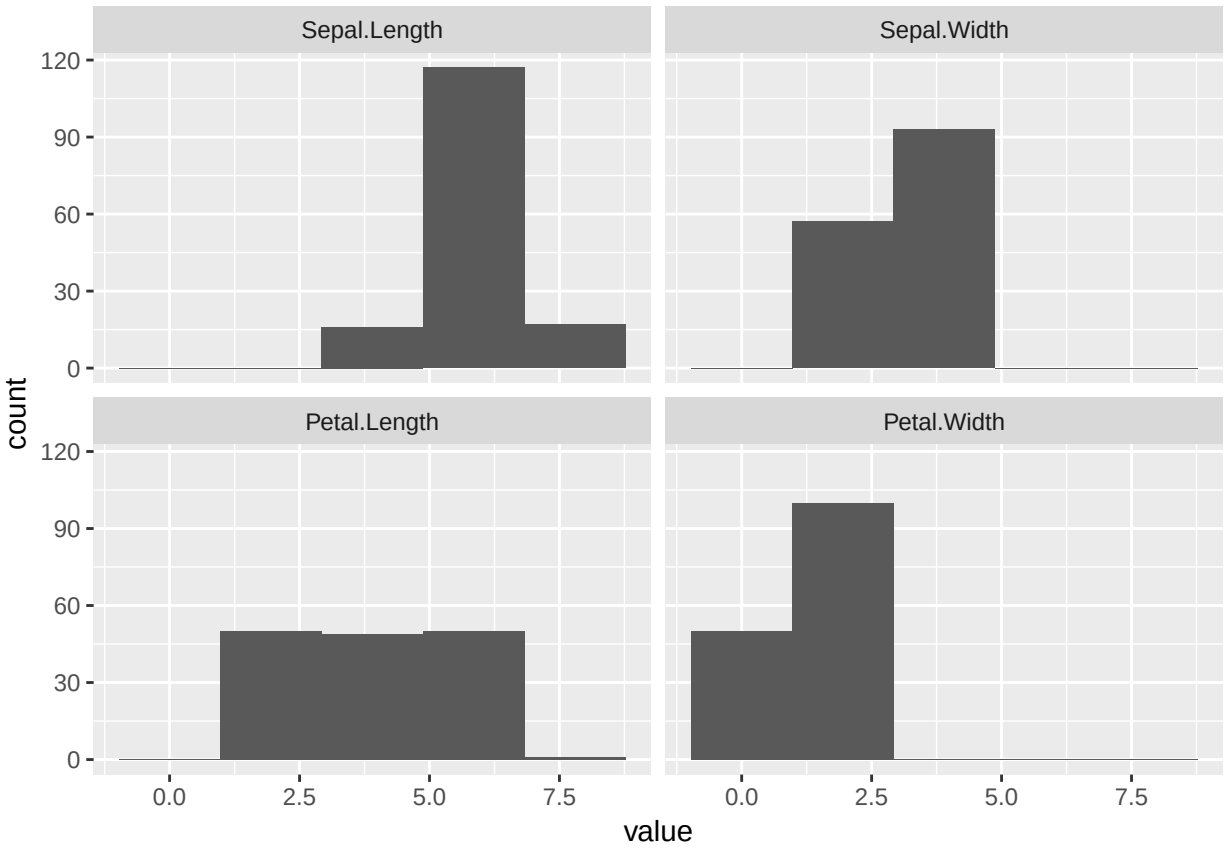
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



4.4 Q4

Vary the number of bins in the above histogram. Describe what you see

```
#Answer: With very few bins, we cannot show the bimodal distribution correctly

iris_melt %>%
  ggplot(aes(value)) +
  geom_histogram(bins=5) +
  facet_wrap(~variable)
```



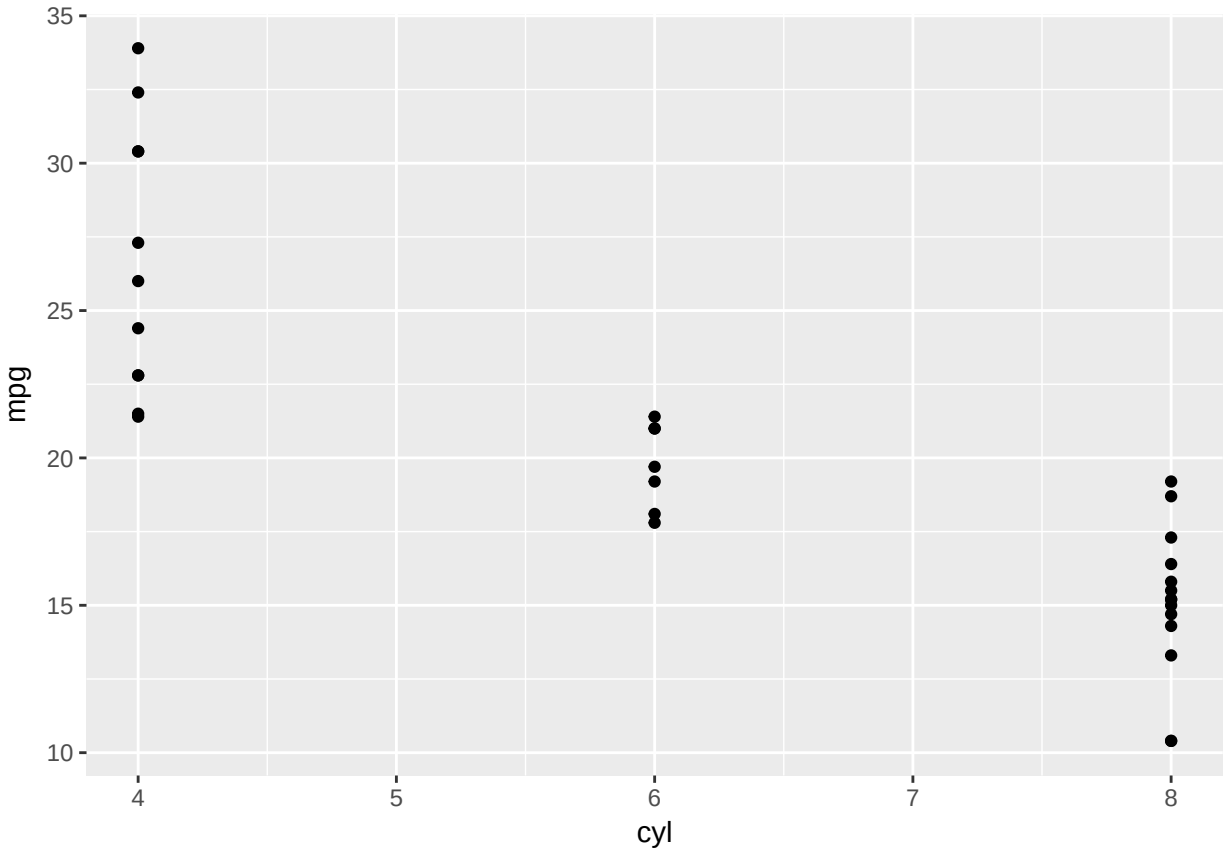
5 Datacamp

5.1 ggplot2 on Datacamp

```
str(mtcars)
```

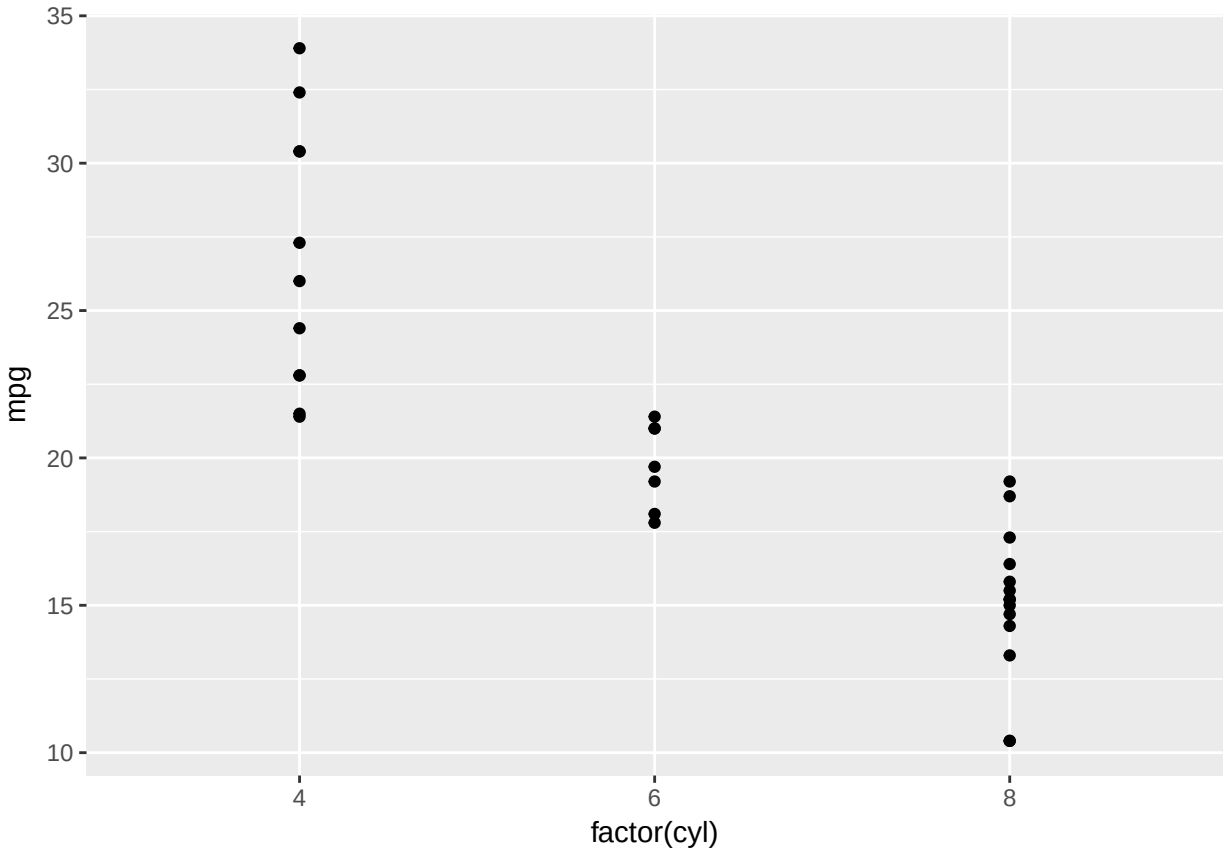
```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
ggplot(mtcars, aes(x = cyl, y = mpg)) +
  geom_point()
```




```
#Stellar scatterplotting! Notice that ggplot2 treats cyl as a factor.
#This time the x-axis does not contain variables like 5 or 7, only the values
#that are present in the dataset.
```

```
# Change the command below so that cyl is treated as factor
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_point()
```



5.2 Exploring ggplot2, part 3

We'll use several datasets throughout the courses to showcase the concepts discussed in the videos. In the previous exercises, you already got to know mtcars. Let's dive a little deeper to explore the three main topics in this course: The data, aesthetics, and geom layers.

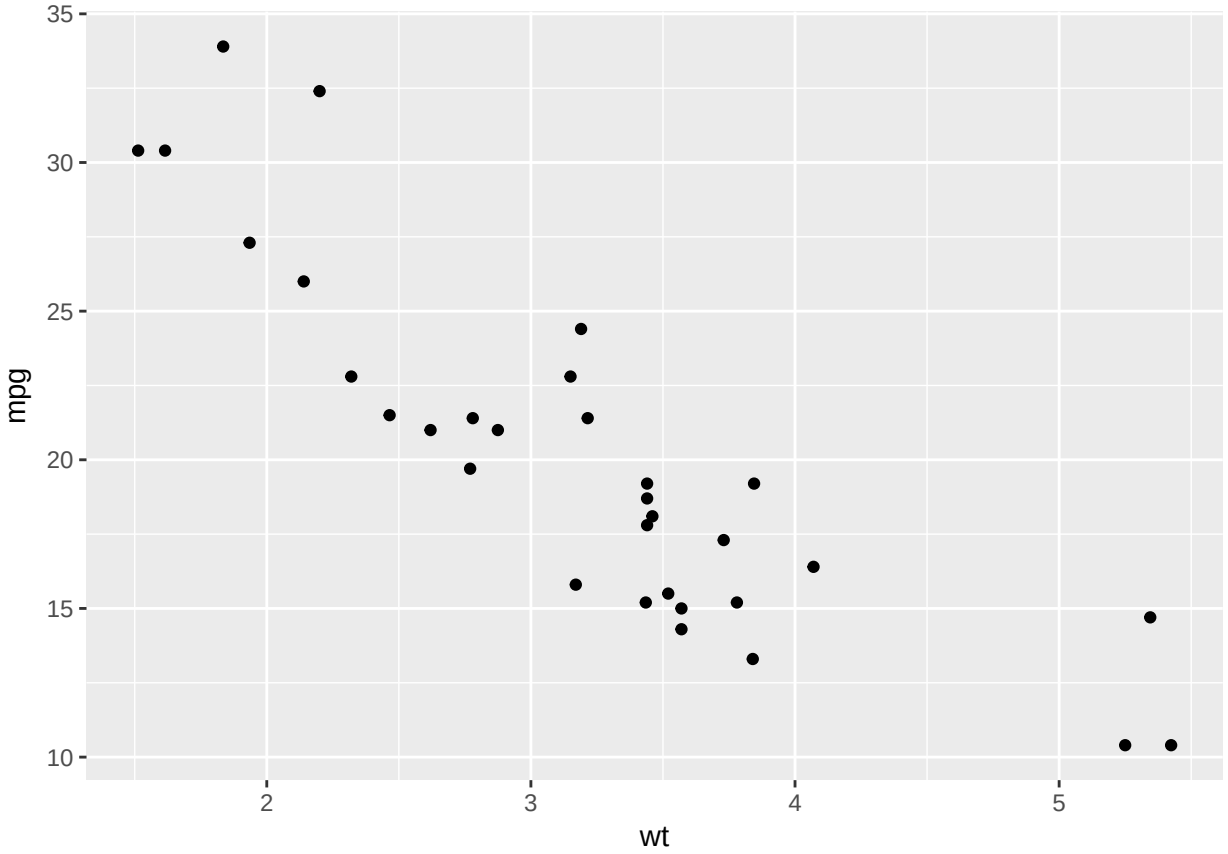
The mtcars dataset contains information about 32 cars from 1973 Motor Trend magazine. This dataset is small, intuitive, and contains a variety of continuous and categorical variables.

You're encouraged to think about how the examples and concepts we discuss throughout these data viz courses apply to your own data-sets!

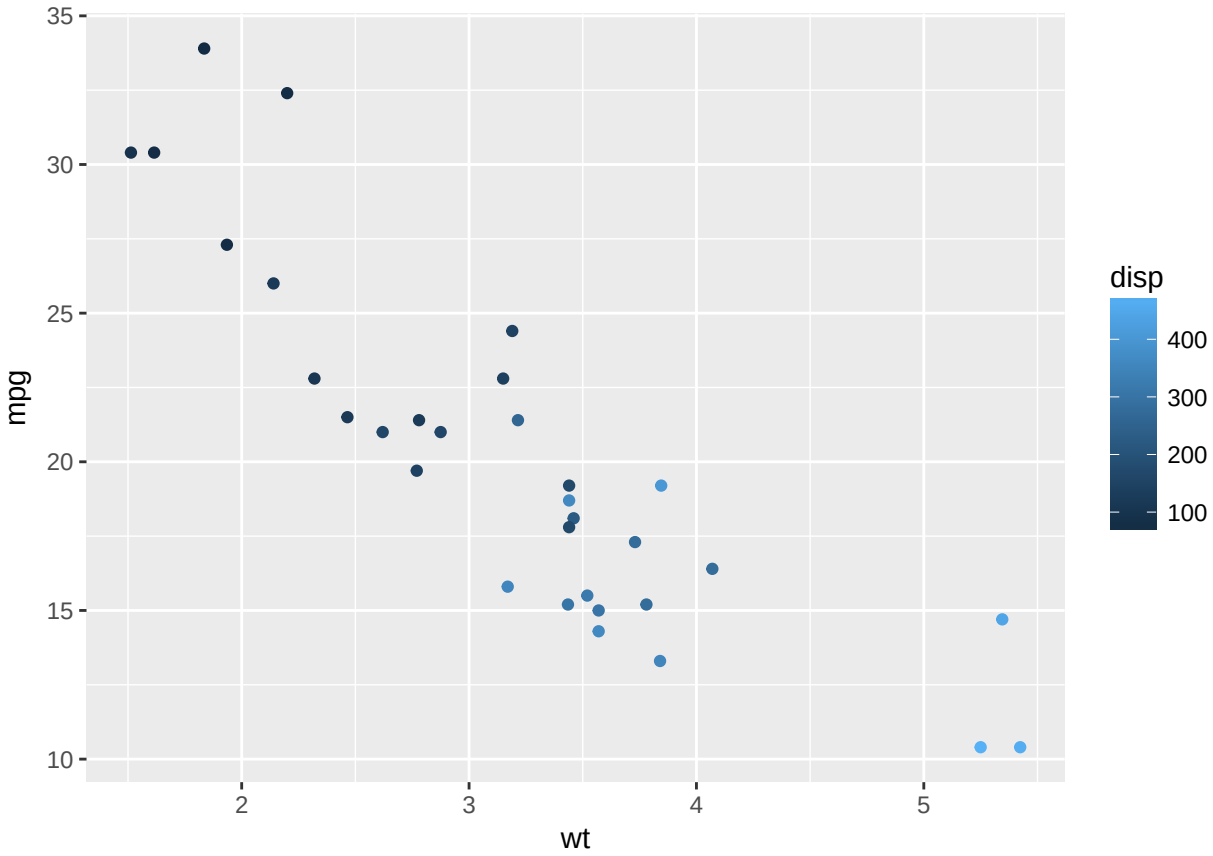
5.2.1 Instructions

- `ggplot2` has already been loaded for you. Take a look at the first command. It plots the mpg (miles per galon) against the weight (in thousands of pounds). You don't have to change anything about this command.
- In the second call of ggplot() change the color argument in aes() (which stands for aesthetics). The color should be dependent on the displacement of the car engine, found in disp.
- In the third call of ggplot() change the size argument in aes() (which stands for aesthetics). The size should be dependent on the displacement of the car engine, found in disp.

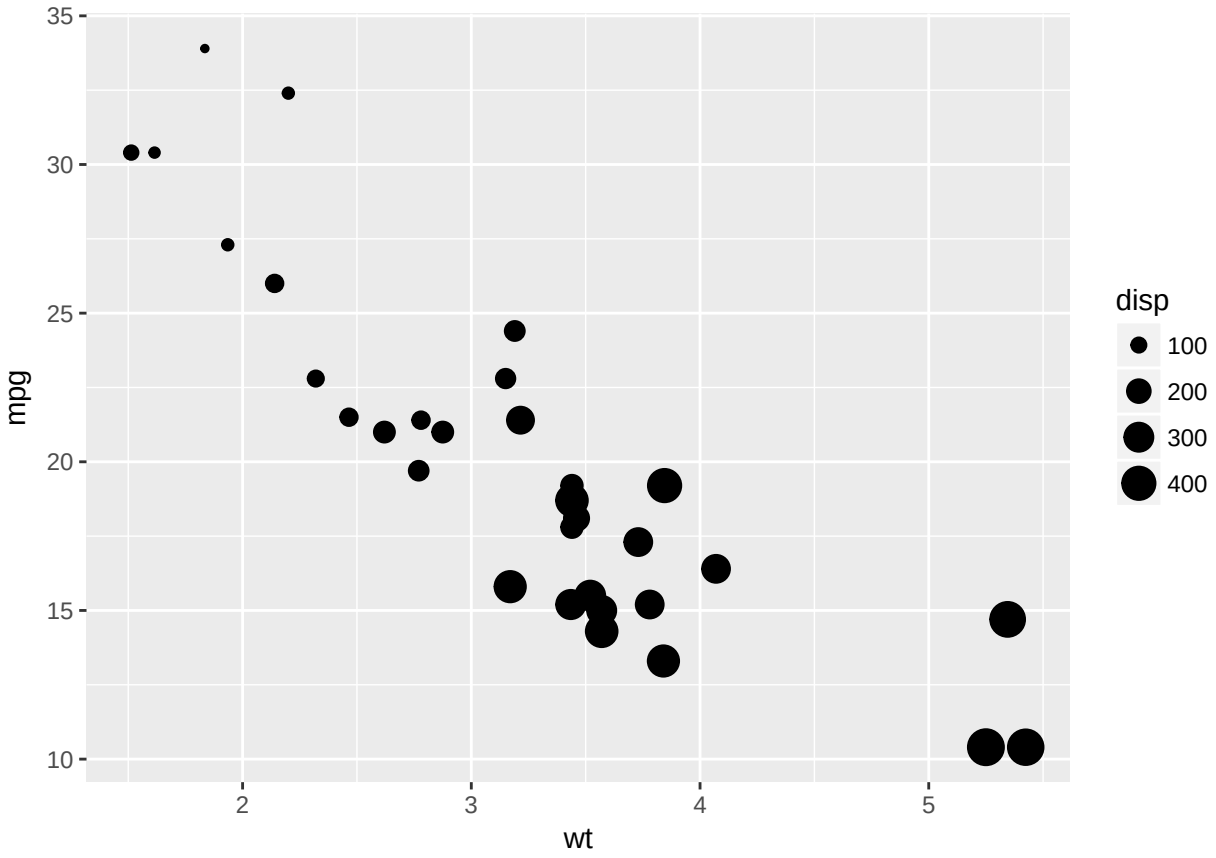
```
# A scatter plot has been made for you
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()
```




```
# Replace ___ with the correct column
ggplot(mtcars, aes(x = wt, y = mpg, color = disp)) +
  geom_point()
```




```
# Replace ___ with the correct column
ggplot(mtcars, aes(x = wt, y = mpg, size = disp)) +
  geom_point()
```



5.3 Understanding Variables

In the previous exercise you saw that `disp` can be mapped onto a color gradient or onto a continuous size scale.

Another argument of `aes()` is the shape of the points. There are a finite number of shapes which ggplot() can automatically assign to the points. However, if you try this command in the console to the right:

```
#ggplot(mtcars, aes(x = wt, y = mpg, shape = disp)) +
#  geom_point()
```

It gives an error. What does this mean?

```
# Error: A continuous variable can not be mapped to shape
#
# Correct. The error message 'A continuous variable can not be mapped to shape',
# means that shape doesn't exist on a continuous scale here.
```

5.4 Exploring ggplot2, part 4

The `diamonds` data frame contains information on the prices and various metrics of 50,000 diamonds. Among the variables included are `carat` (a measurement of the size of the diamond) and price. For the next exercises, you'll be using a subset of 1,000 diamonds.

Here you'll use two common geom layer functions: `geom_point()` and `geom_smooth()`. We already saw in the earlier exercises how these are added using the + operator.

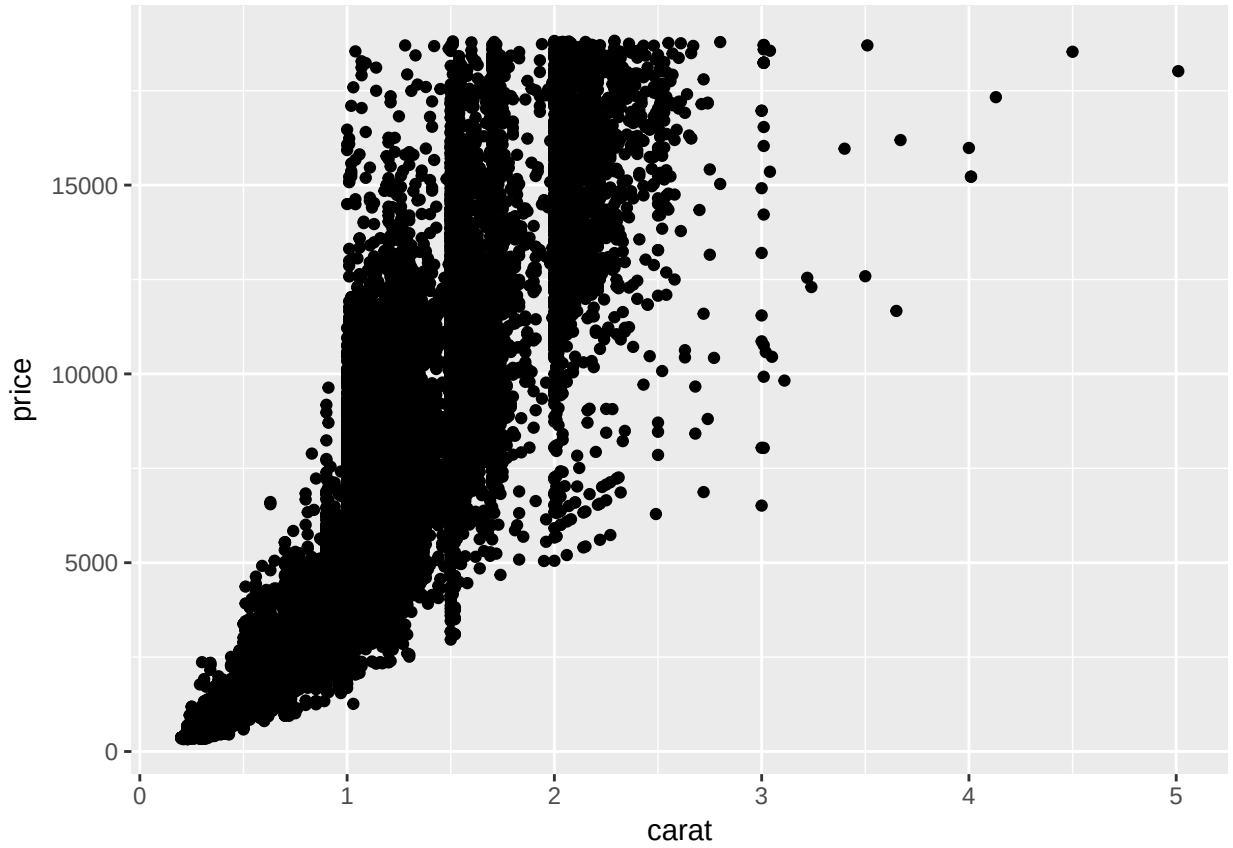
5.4.1 Instructions

- Explore the diamonds data frame with the `str()` function.
- Use the `+` operator to add `geom_point()` to the first `ggplot()` command. This will tell `ggplot2` to draw points on the plot.
- Use the `+` operator to add `geom_point()` and `geom_smooth()`. These just stack on each other! `geom_smooth()` will draw a smoothed line over the points.

```
# Explore the diamonds data frame with str()
str(diamonds)
```

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

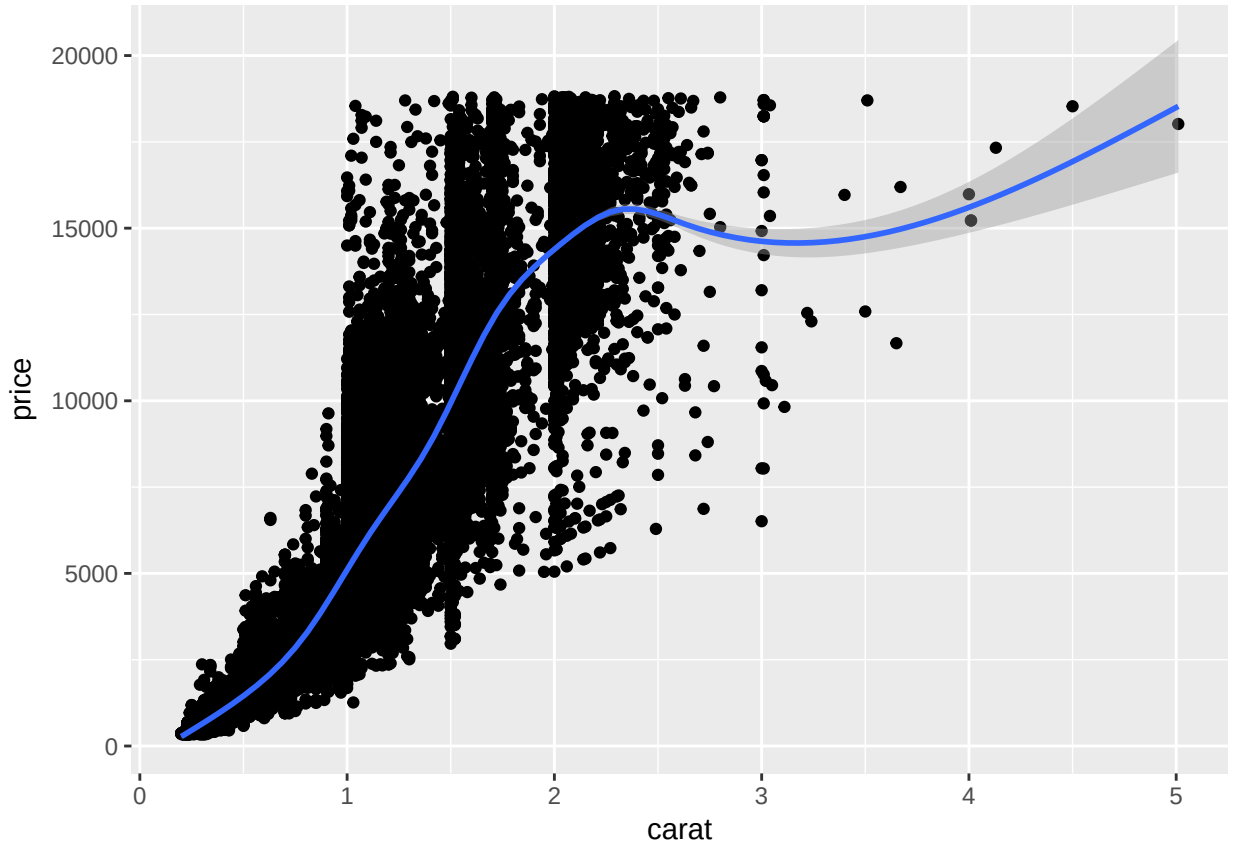
```
# Add geom_point() with +
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point()
```




```
# Add geom_point() and geom_smooth() with +
```

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```




```
# Lovely layering! If you had executed the command without adding a +, it would
# produce an error message 'No layers in plot' because you are missing the third
# essential layer - the geom layer.
```

5.5 Exploring ggplot2, part 5

The code for last plot of the previous exercise is available in the script on the right. It builds a scatter plot of the diamonds dataset, with carat on the x-axis and price on the y-axis. geom_smooth() is used to add a smooth line.

With this plot as a starting point, let's explore some more possibilities of combining geoms.

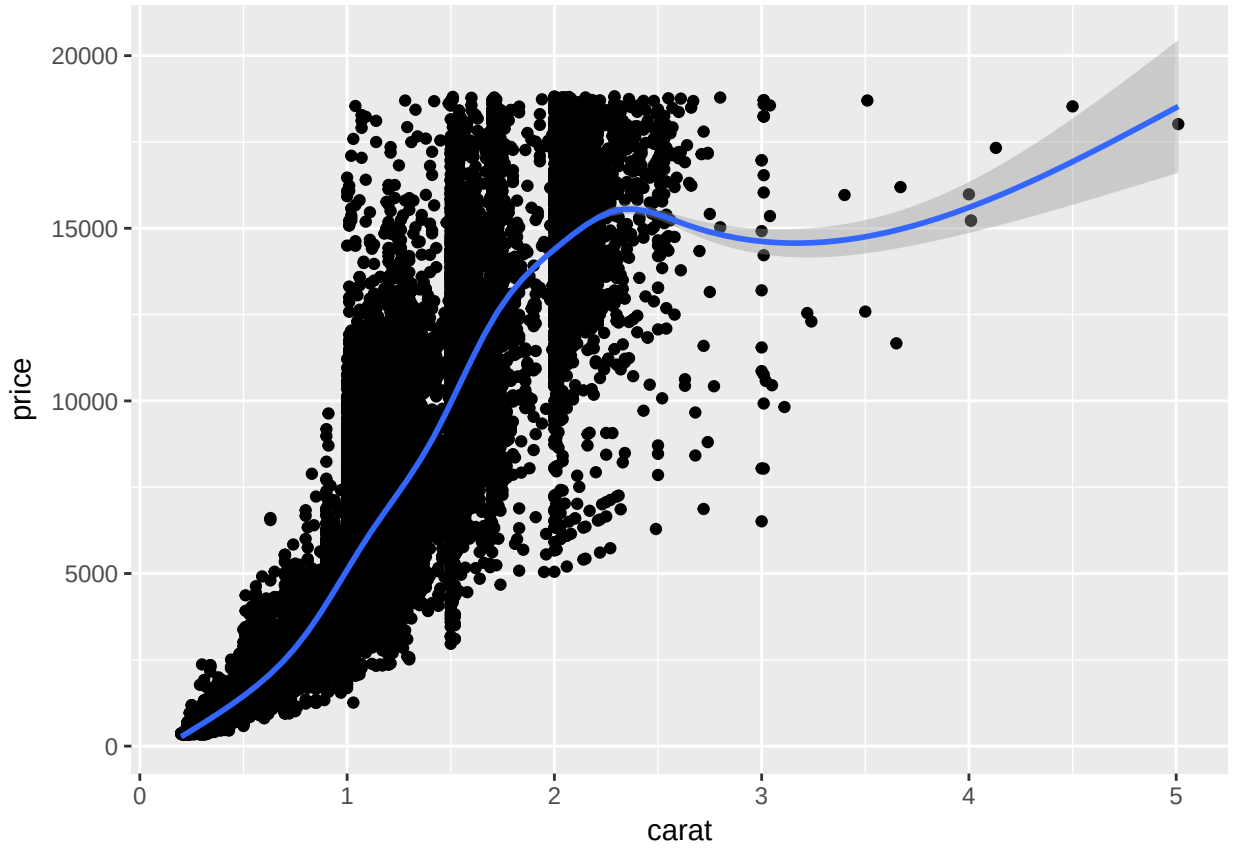
5.5.1 Instructions

- Plot 2 - Copy and paste plot 1, but show only the smooth line, no points.
- Plot 3 - Show only the smooth line, but color according to clarity by placing the argument color = clarity in the aes() function of your ggplot() call.
- Plot 4 - Draw translucent colored points.
  - Copy the ggplot() command from plot 3 (with clarity mapped to color).
  - Remove the smooth layer.

- Add the points layer back in.
- Set alpha = 0.4 inside geom_point(). This will make the points 40% transparent.

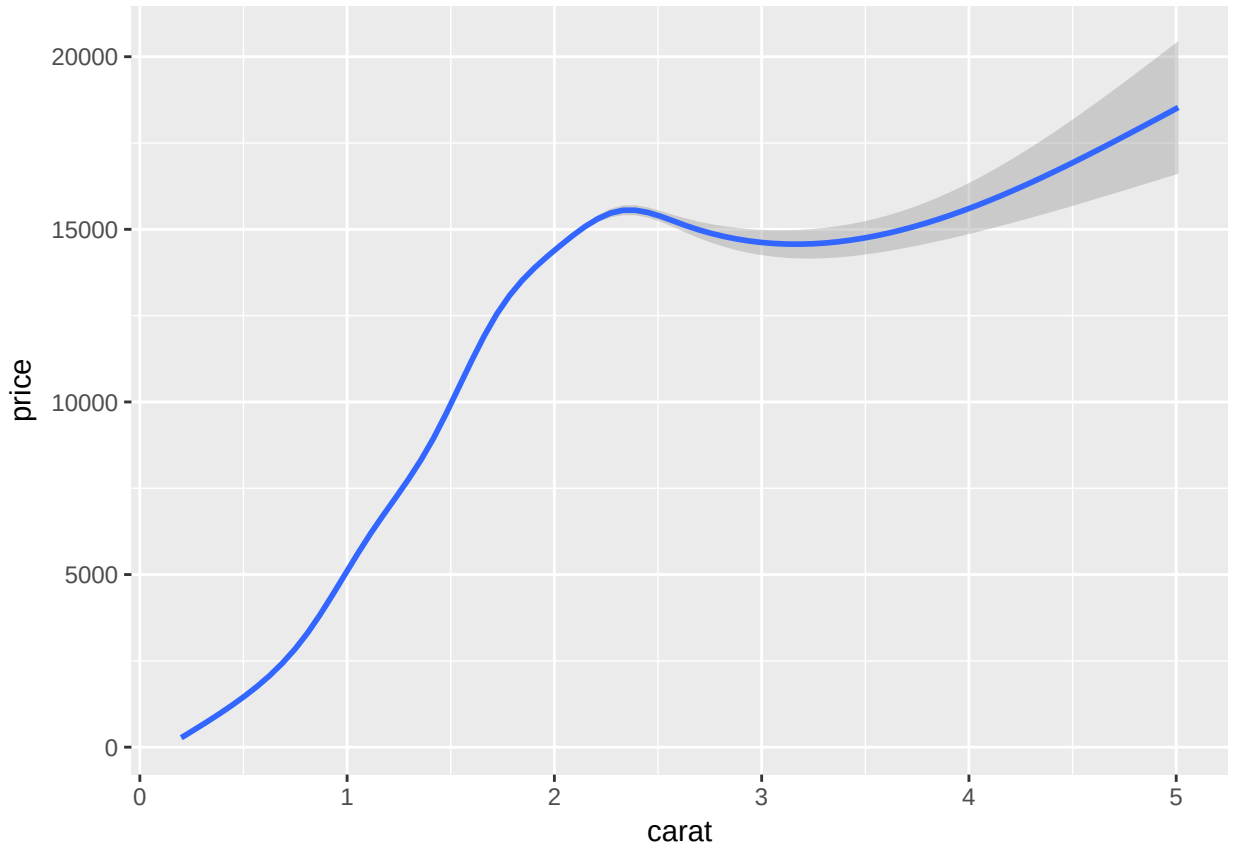
```
# 1 - The plot you created in the previous exercise
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```



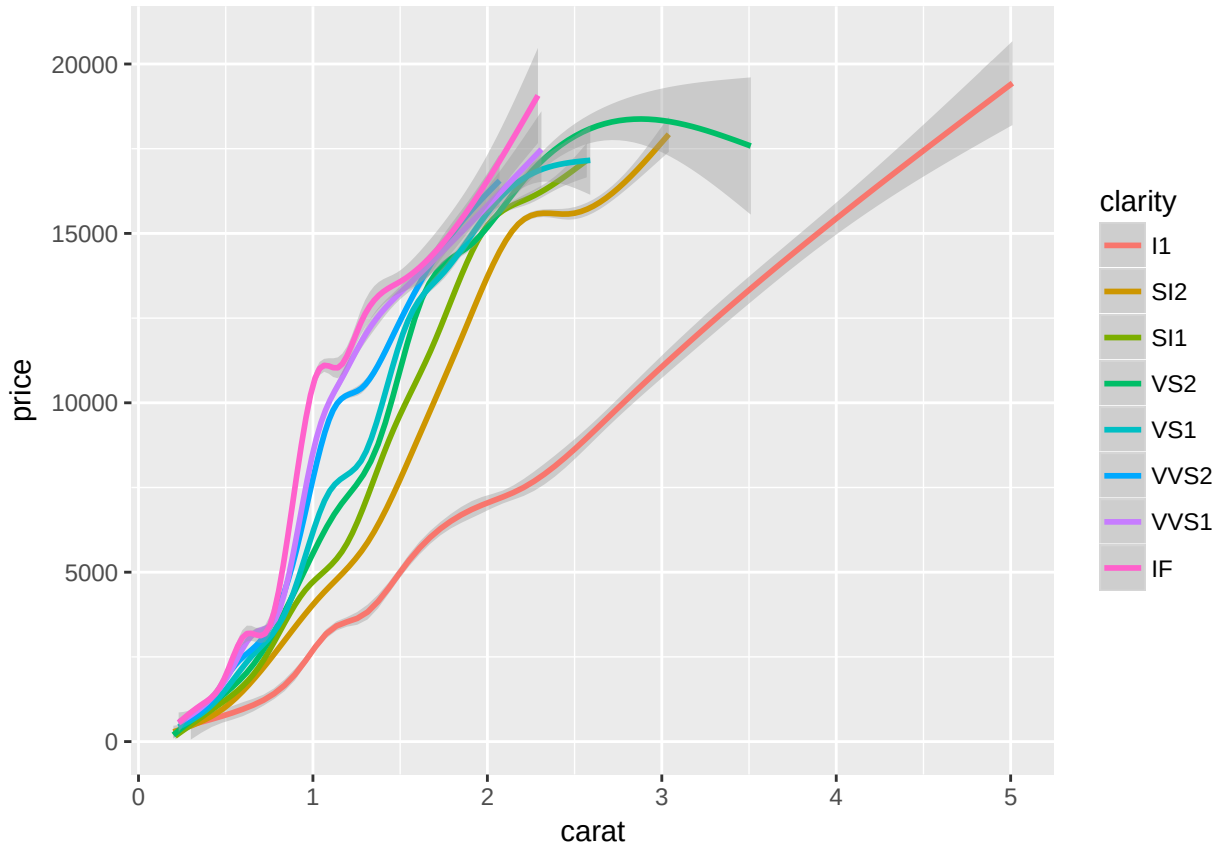

```
# 2 - Copy the above command but show only the smooth line
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```

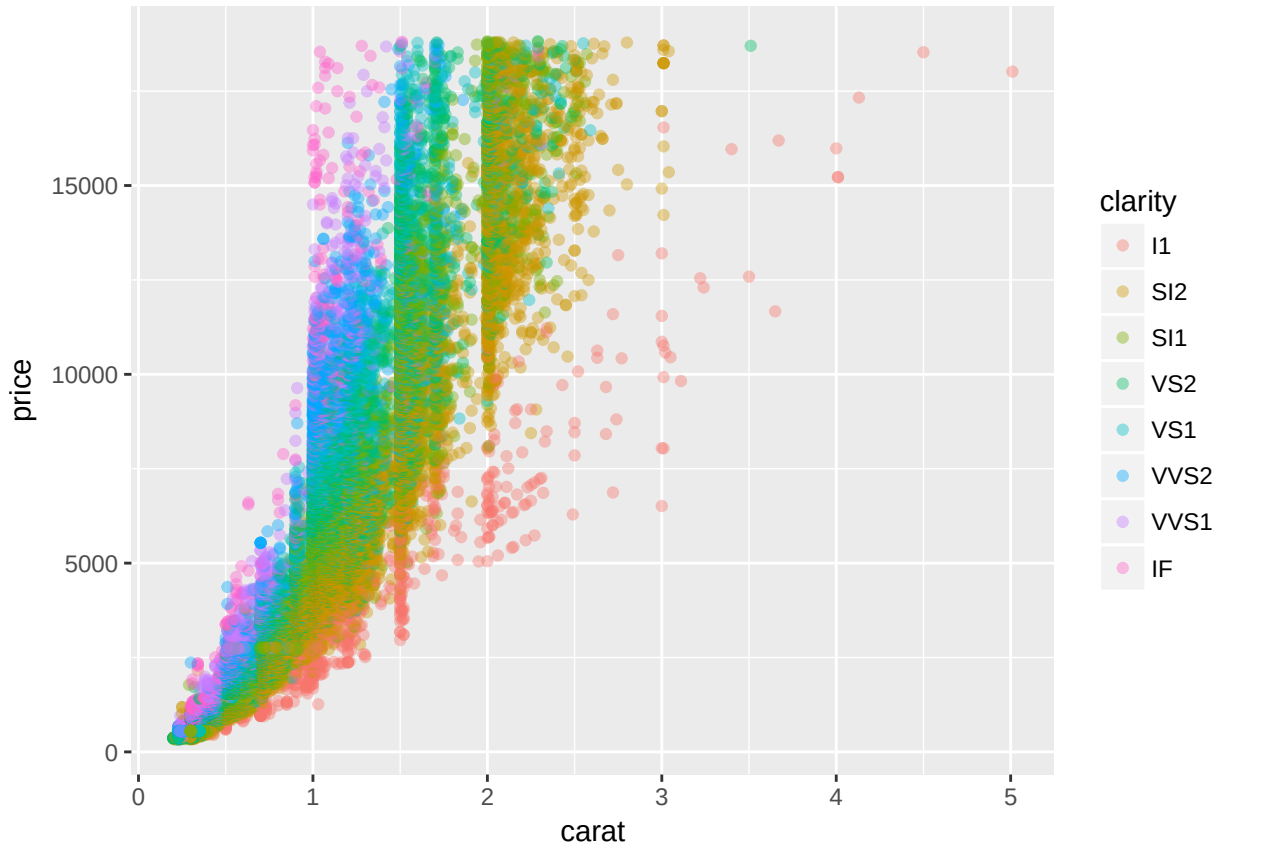



```
# 3 - Copy the above command and assign the correct value to col in aes()
ggplot(diamonds, aes(x = carat, y = price, color=clarity)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```




```
# 4 - Keep the color settings from previous command. Plot only the points with argument alpha.
ggplot(diamonds, aes(x = carat, y = price, color=clarity)) +
  geom_point(alpha=0.4)
```




```
## Smooth work! `geom_point() + geom_smooth()` is a common combination.
```

5.6 Understanding the grammar, part 1

Here you'll explore some of the different grammatical elements. Throughout this course, you'll discover how they can be combined in all sorts of ways to develop unique plots.

In the following instructions, you'll start by creating a ggplot object from the diamonds dataset. Next, you'll add layers onto this object to build beautiful & informative plots.

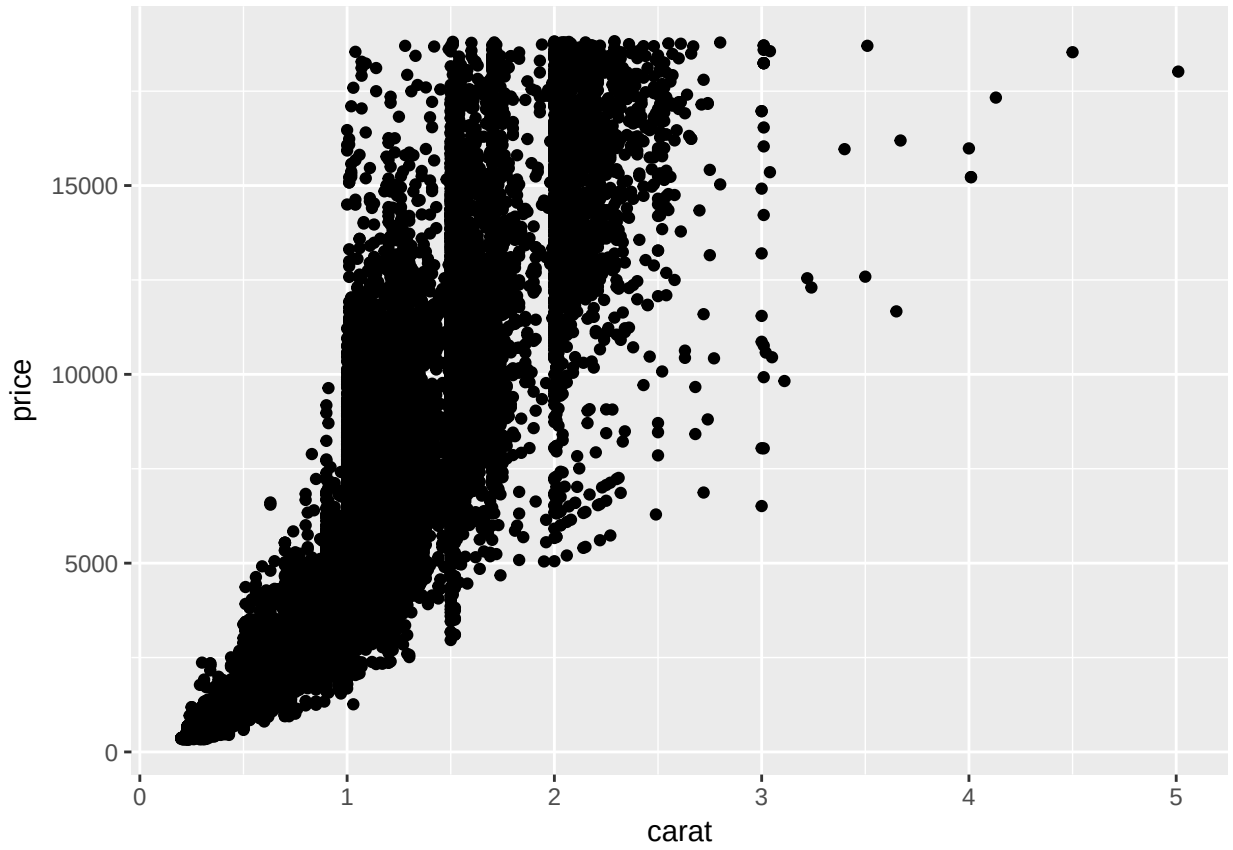
5.6.1 Instructions

Define the data (diamonds) and aesthetics layers. Map carat on the x axis and price on the y axis. Assign it to an object: dia_plot.

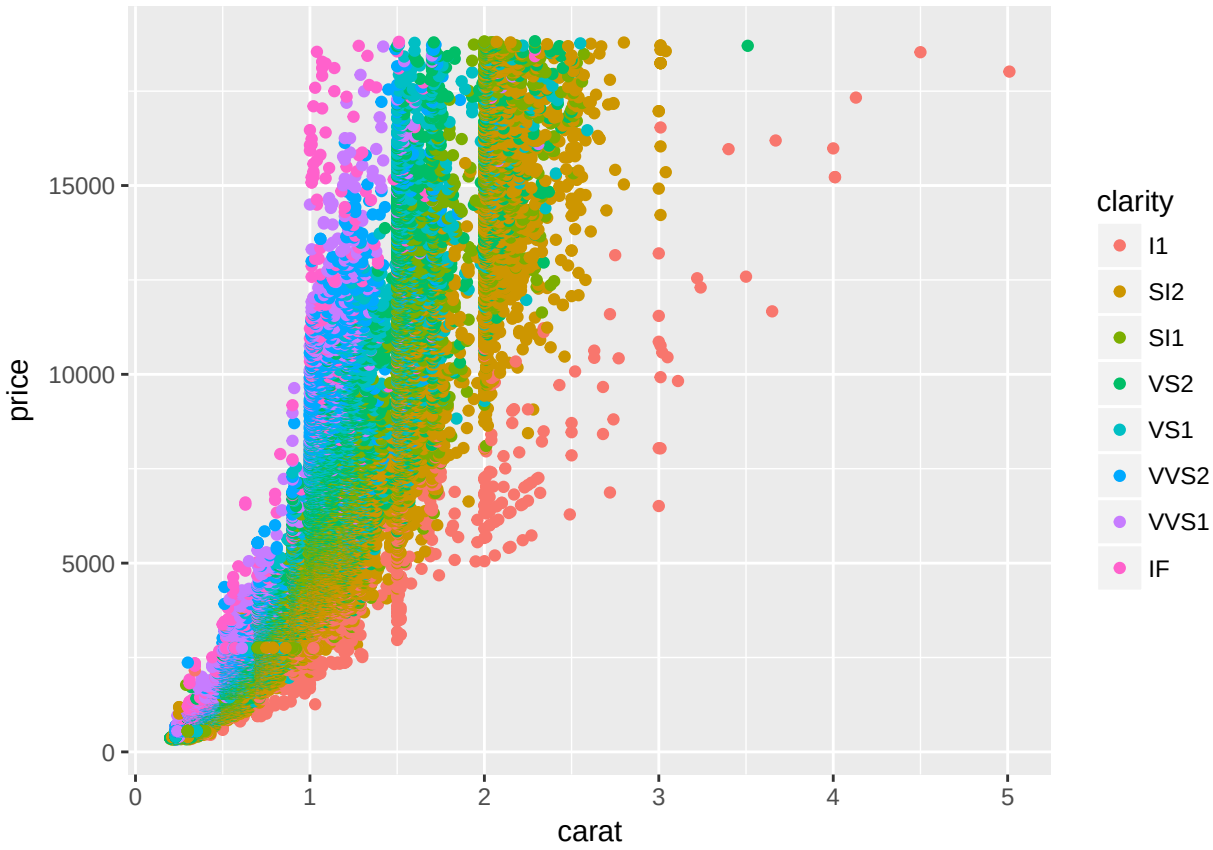
Using +, add a geom_point() layer (with no arguments), to the dia_plot object. This can be in a single or multiple lines.

Note that you can also call aes() within the geom_point() function. Map clarity to the color argument in this way.

```
# Create the object containing the data and aes layers: dia_plot
dia_plot <- ggplot(diamonds, aes(x = carat, y = price))

# Add a geom layer with + and geom_point()
dia_plot + geom_point()
```




```
# Add the same geom layer, but with aes() inside
dia_plot + geom_point(aes(color = clarity))
```




```
# Remarkable plot recyling! Notice how you can store the plot as a ggplot object
# that you can use later on to add other layers; that's pretty convenient!
```

5.7 Understanding the grammar, part 2

Continuing with the previous exercise, here you'll explore mixing arguments and aesthetics in a single geometry.

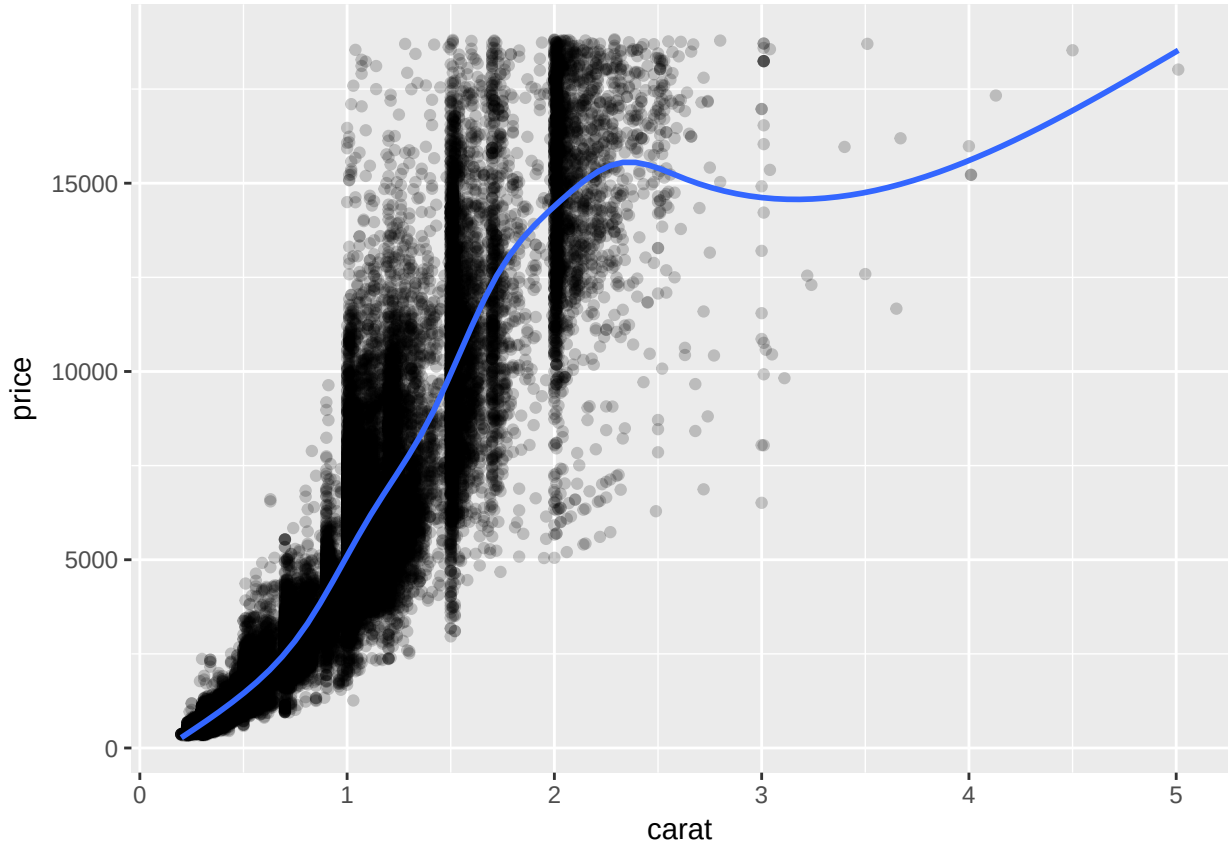
You're still working on the diamonds dataset.

5.7.1 Instructions

1 - The dia_plot object has been created for you. 2 - Update dia_plot so that it contains all the functions to make a scatter plot by using geom_point() for the geom layer. Set alpha = 0.2. 3 - Using +, plot the dia_plot object with a geom_smooth() layer on top. You don't want any error shading, which can be achieved by setting the se = FALSE in geom_smooth(). 4 - Modify the geom_smooth() function from the previous instruction so that it contains aes() and map clarity to the col argument.

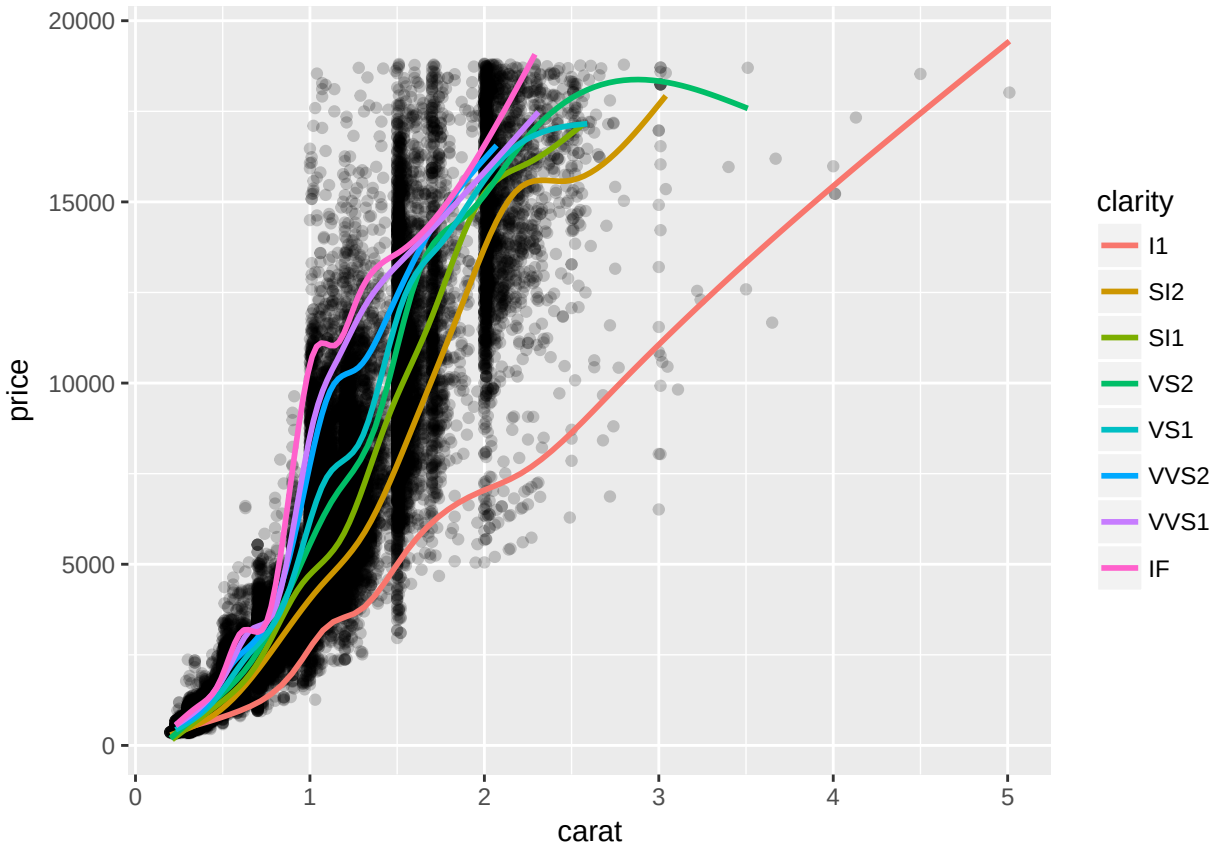
```
# 1 - The dia_plot object has been created for you
dia_plot <- ggplot(diamonds, aes(x = carat, y = price))

# 2 - Expand dia_plot by adding geom_point() with alpha set to 0.2
dia_plot <- dia_plot +geom_point(alpha=0.2)

# 3 - Plot dia_plot with additional geom_smooth() with se set to FALSE
dia_plot + geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'gam'
```




```
# 4 - Copy the command from above and add aes() with the correct mapping to
# geom_smooth()
dia_plot + geom_smooth(aes(col = clarity), se = FALSE)
```

```
## `geom_smooth()` using method = 'gam'
```




```
# Bravo! To set a property of a geom to a single value, pass it as an argument.
# To give the property different values for each row of data, pass it as an
# aesthetic.
```

5.8 Chapter 2

5.9 base package and ggplot2, part 1 - plot

These courses are about understanding data visualization in the context of the grammar of graphics. To gain a better appreciation of ggplot2 and to understand how it operates differently from base package, it's useful to make some comparisons.

In the video, you already saw one example of how to make a (poor) multivariate plot in base package. In this series of exercises you'll take a look at a better way using the equivalent version in ggplot2.

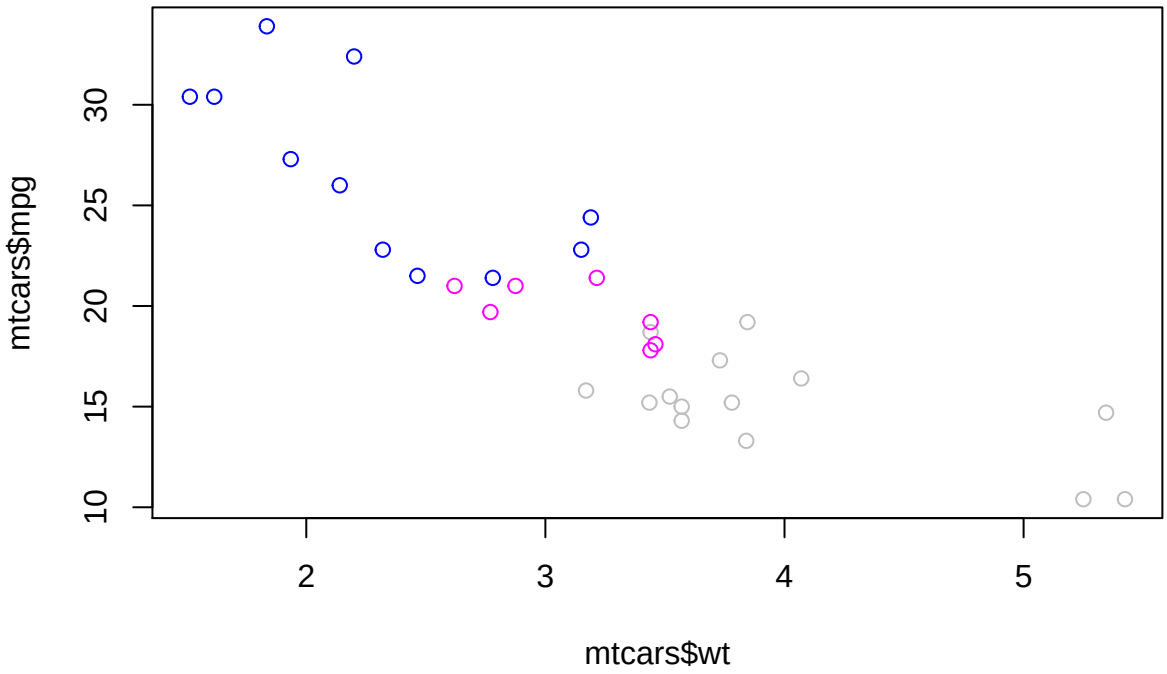
First, let's focus on base package. You want to make a plot of mpg (miles per gallon) against wt (weight in thousands of pounds) in the mtcars data frame, but this time you want the dots colored according to the number of cylinders, cyl. How would you do that in base package? You can use a little trick to color the dots by specifying a factor variable as a color. This works because factors are just a special class of the integer type.

5.9.1 Instructions

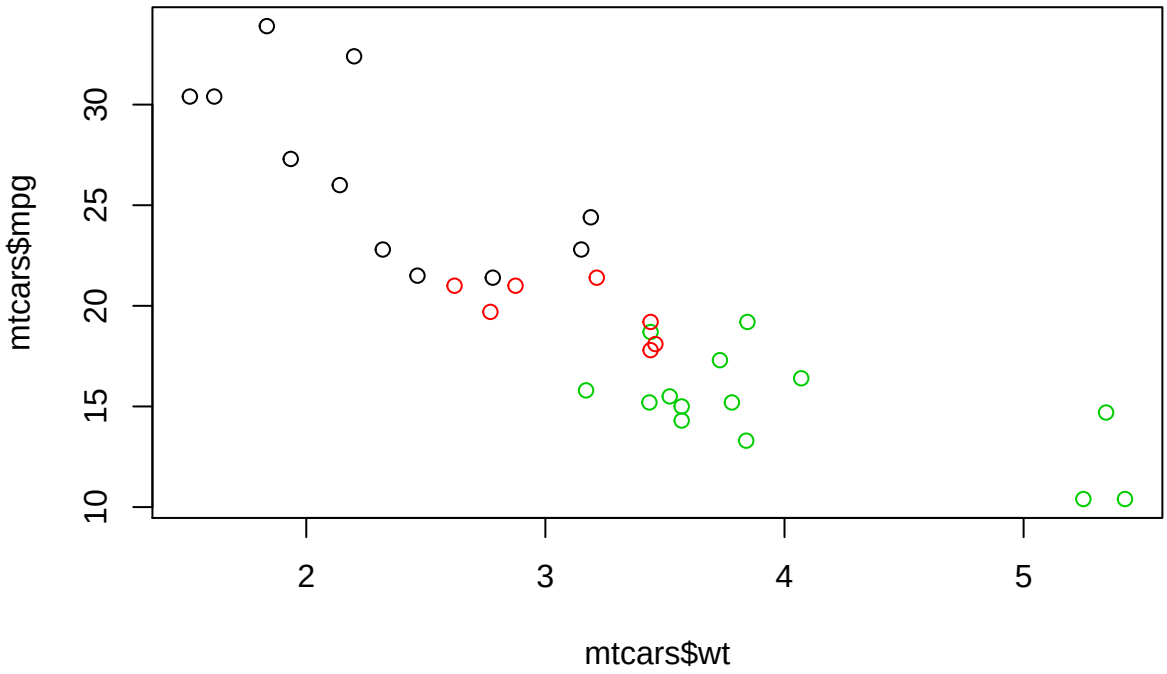
Using the base package plot(), make a scatter plot with `mtcars$wt` on the x-axis and `mtcars$mpg` on the y-axis, colored according to `mtcars$cyl` (use the col argument). You can specify data = but you'll just do it

the long way here. Add a new column, fcyl, to the mtcars data frame. This should be cyl converted to a factor. Create a similar plot to instruction 1, but this time, use fcyl (which is cyl as a factor) to set the col.

```
# Plot the correct variables of mtcars
plot(mtcars$wt, mtcars$mpg, col = mtcars$cyl)
```




```
# Change cyl inside mtcars to a factor
mtcars$fcyl <- as.factor(mtcars$cyl)

# Make the same plot as in the first instruction
plot(mtcars$wt, mtcars$mpg, col=mtcars$fcyl)
```



```
# It's all about that base! Recall that under-the-hood, factors are simply
# integer type vectors, so the colors in the second plot are 1, 2, and 3.
# In the first plot the colors were 4, 6, and 8.
```

5.10 base package and ggplot2, part 2 - lm

If you want to add a linear model to your plot, shown right, you can define it with lm() and then plot the resulting linear model with abline(). However, if you want a model for each subgroup, according to cylinders, then you have a couple of options.

You can subset your data, and then calculate the lm() and plot each subset separately. Alternatively, you can vectorize over the cyl variable using lapply() and combine this all in one step. This option is already prepared for you.

The code to the right contains a call to the function lapply(), which you might not have seen before. This function takes as input a vector and a function. Then lapply() applies the function it was given to each element of the vector and returns the results in a list. In this case, lapply() takes each element of `mtcars$cyl` and calls the function defined in the second argument. This function takes a value of mtcars$cyl and then subsets the data so that only rows with cyl == x are used. Then it fits a linear model to the filtered dataset and uses that model to add a line to the plot with the abline() function.

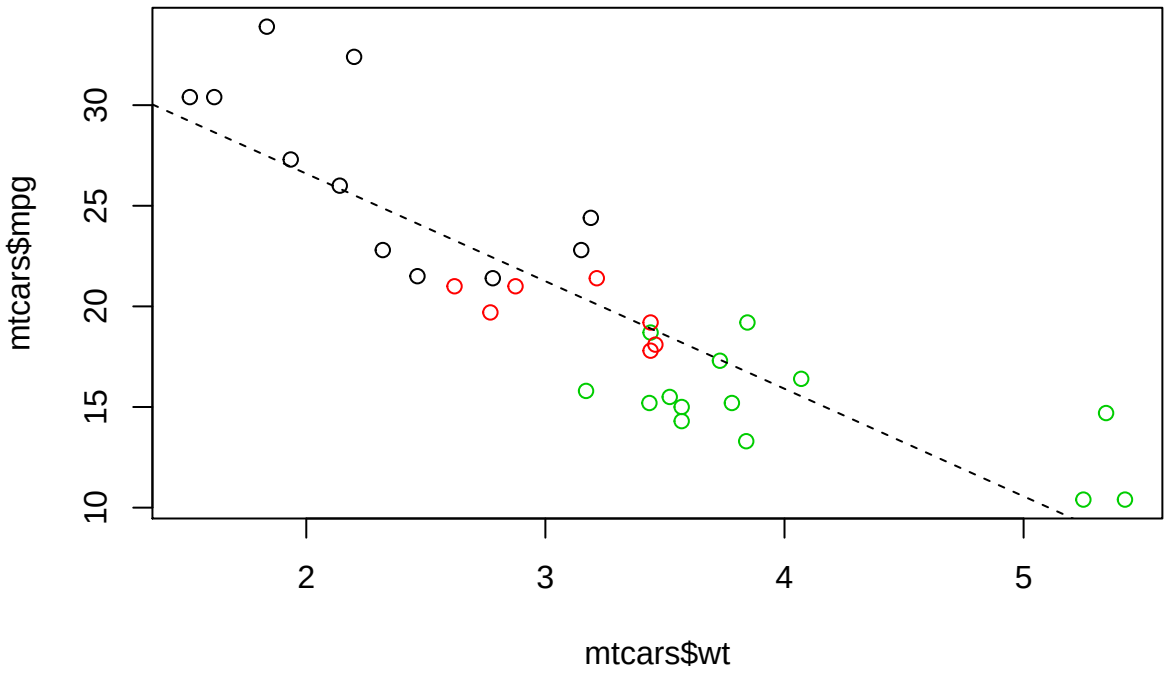
Now that you have an interesting plot, there is a very important aspect missing - the legend!

In base package you have to take care of this using the legend() function. This has been done for you in the predefined code.

5.10.1 Instructions

Fill in the lm() function to calculate a linear model of mpg described by wt and save it as an object called carModel. Draw the linear model on the scatterplot. Write code that calls abline() with carModel as the first argument. Set the line type by passing the argument lty = 2. Run the code that generates the basic plot and the call to abline() all at once by highlighting both parts of the script and hitting control/command + enter on your keyboard. These lines must all be run together in the DataCamp R console so that R will be able to find the plot you want to add a line to. Run the code already given to generate the plot with a different model for each group. You don't need to modify any of this.

```
# Use lm() to calculate a linear model and save it as carModel
carModel <- lm(mpg ~ wt, data = mtcars)

# Basic plot
mtcars$cyl <- as.factor(mtcars$cyl)
plot(mtcars$wt, mtcars$mpg, col = mtcars$cyl)

# Call abline() with carModel as first argument and set lty to 2
abline(carModel, lty = 2)
```



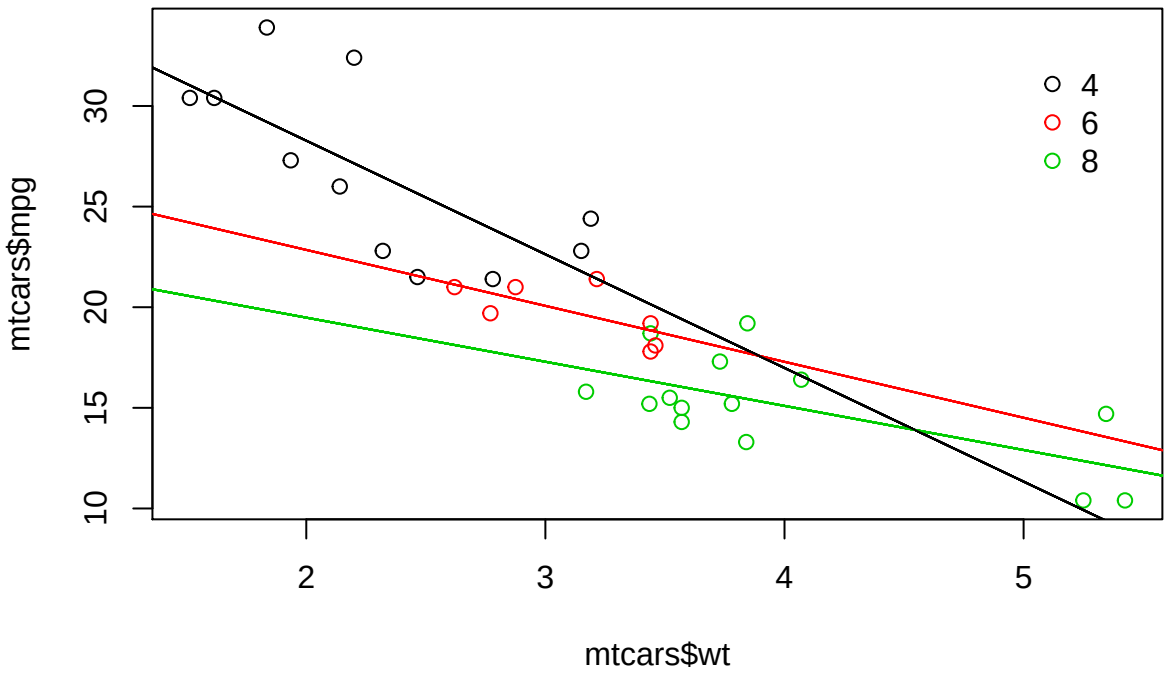

```
# Plot each subset efficiently with lapply
# You don't have to edit this code
plot(mtcars$wt, mtcars$mpg, col = mtcars$cyl)
lapply(mtcars$cyl, function(x) {
  abline(lm(mpg ~ wt, mtcars, subset = (cyl == x)), col = x)
  })
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
##
## [[9]]
## NULL
```

```
##
## [[10]]
## NULL
##
## [[11]]
## NULL
##
## [[12]]
## NULL
##
## [[13]]
## NULL
##
## [[14]]
## NULL
##
## [[15]]
## NULL
##
## [[16]]
## NULL
##
## [[17]]
## NULL
##
## [[18]]
## NULL
##
## [[19]]
## NULL
##
## [[20]]
## NULL
##
## [[21]]
## NULL
##
## [[22]]
## NULL
##
## [[23]]
## NULL
##
## [[24]]
## NULL
##
## [[25]]
## NULL
##
## [[26]]
## NULL
##
## [[27]]
## NULL
```

```
##
## [[28]]
## NULL
##
## [[29]]
## NULL
##
## [[30]]
## NULL
##
## [[31]]
## NULL
##
## [[32]]
## NULL
```

```
# This code will draw the legend of the plot
# You don't have to edit this code
legend(x = 5, y = 33, legend = levels(mtcars$cyl),
       col = 1:3, pch = 1, bty = "n")
```




```
# Phew! Notice how the legend had to be set manually. In general, ggplot2 makes
# it easier to polish plots compared to base
```

5.11 base package and ggplot2, part 3

In this exercise you'll recreate the base package plot in ggplot2.

The code for base R plotting is given at the top. The first line of code already converts the cyl variable of mtcars to a factor.

5.11.1 Instructions

Plot 1: add geom_point() in order to make a scatter plot.

Plot 2: copy and paste Plot 1

Add a linear model for each subset according to cyl by adding a geom_smooth() layer

Inside this geom_smooth(), set method to "lm" and se to FALSE.

Note: geom_smooth() will automatically draw a line per cyl subset. It recognizes the groups you want to identify by color in the aes() call within the ggplot() command.

Plot 3: copy and paste Plot 2

Plot a linear model for the entire dataset, do this by adding another geom_smooth() layer

Set the group aesthetic inside this geom_smooth() layer to 1. This has to be set within the aes() function.

Set method to "lm", se to FALSE and linetype to 2. These have to be set outside aes() of the geom_smooth().

Note: the group aesthetic will tell ggplot() to draw a single linear model through all the points.

5.11.2 HINT

For Plot 1, you have to add geom_point() to the ggplot() command without any arguments. Use +. For Plot 2, you should expand the previous ggplot() command with a geom_smooth() layer. The arguments you have to set are described in the instructions. You don't have to add anything to draw a line per cyl group, ggplot() will do this automatically, as described in the instructions. For Plot 3, expand the previous ggplot() command with another geom_smooth() layer. This time the first argument should be an aes() mapping with group = 1. Don't forget to set the linetype as defined in the instructions. This is an argument of geom_smooth(), not of aes(). method and se should be set as in the previous command.

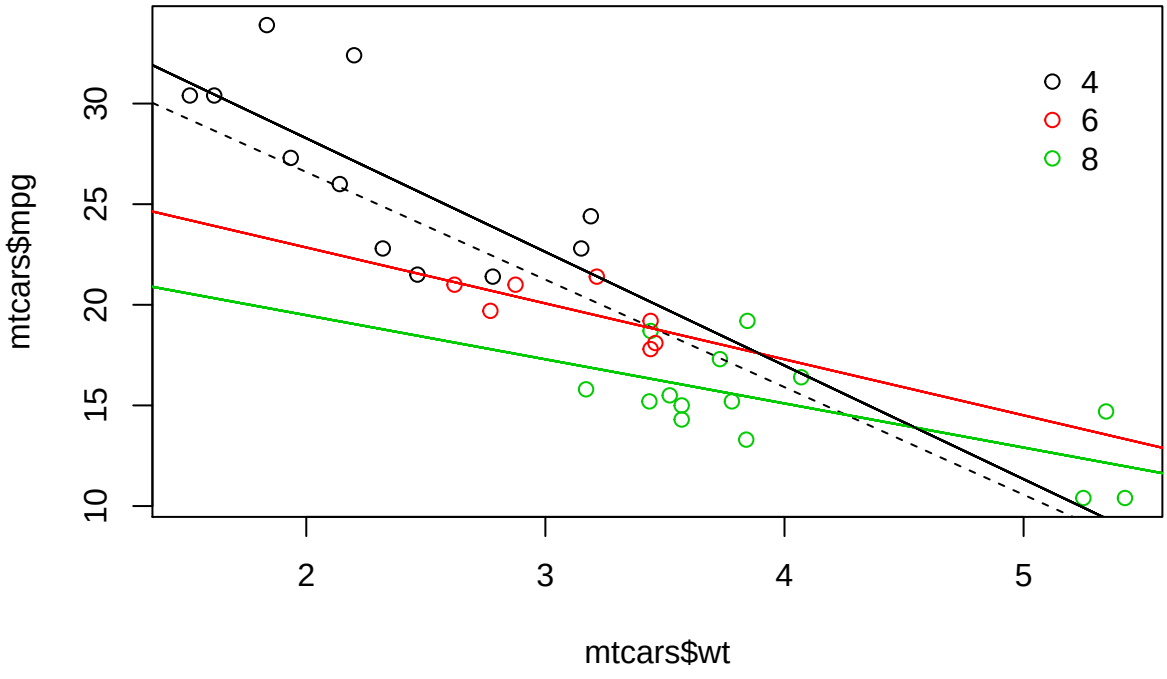
```
# Convert cyl to factor (don't need to change)
mtcars$cyl <- as.factor(mtcars$cyl)

# Example from base R (don't need to change)
plot(mtcars$wt, mtcars$mpg, col = mtcars$cyl)
abline(lm(mpg ~ wt, data = mtcars), lty = 2)
lapply(mtcars$cyl, function(x) {
  abline(lm(mpg ~ wt, mtcars, subset = (cyl == x)), col = x)
  })
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
```

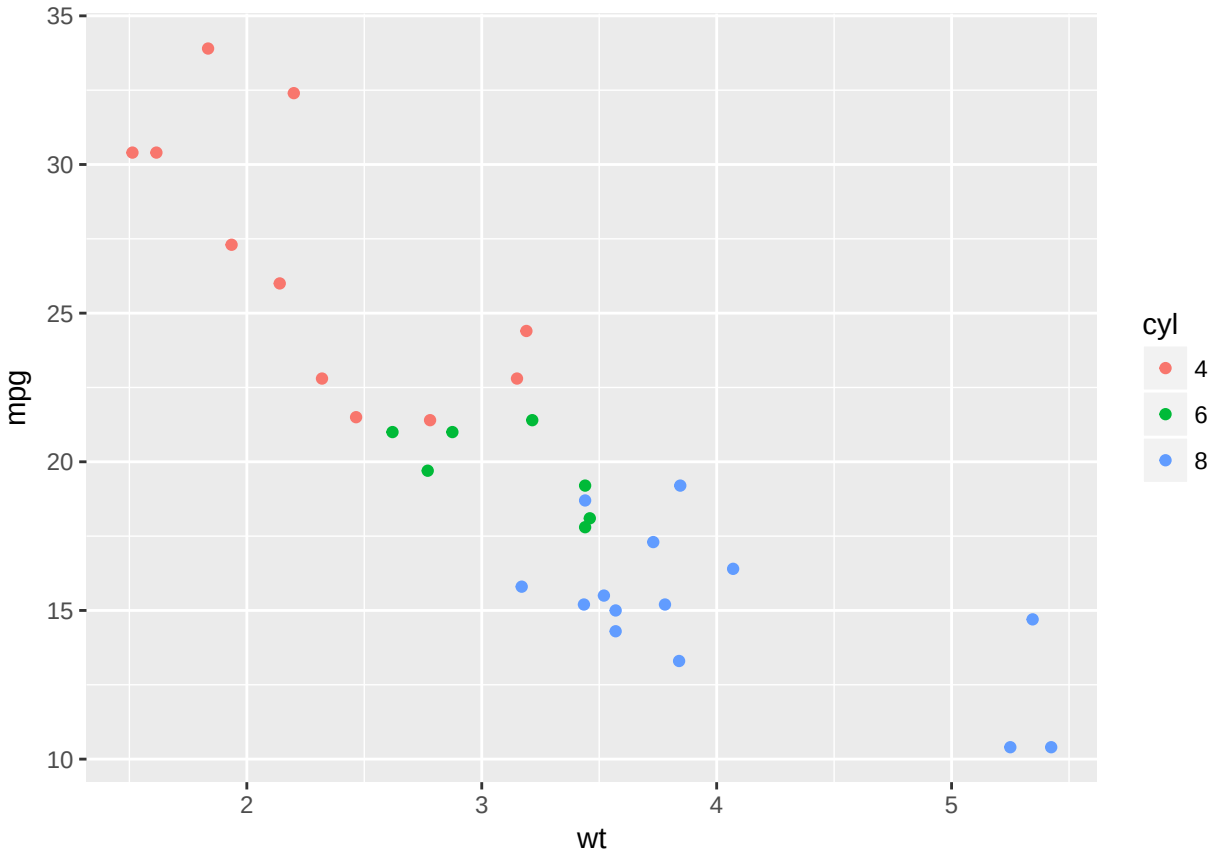
```
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
##
## [[9]]
## NULL
##
## [[10]]
## NULL
##
## [[11]]
## NULL
##
## [[12]]
## NULL
##
## [[13]]
## NULL
##
## [[14]]
## NULL
##
## [[15]]
## NULL
##
## [[16]]
## NULL
##
## [[17]]
## NULL
##
## [[18]]
## NULL
##
## [[19]]
## NULL
##
## [[20]]
## NULL
##
## [[21]]
## NULL
##
## [[22]]
## NULL
##
## [[23]]
## NULL
##
## [[24]]
```

```
## NULL
##
## [[25]]
## NULL
##
## [[26]]
## NULL
##
## [[27]]
## NULL
##
## [[28]]
## NULL
##
## [[29]]
## NULL
##
## [[30]]
## NULL
##
## [[31]]
## NULL
##
## [[32]]
## NULL
```

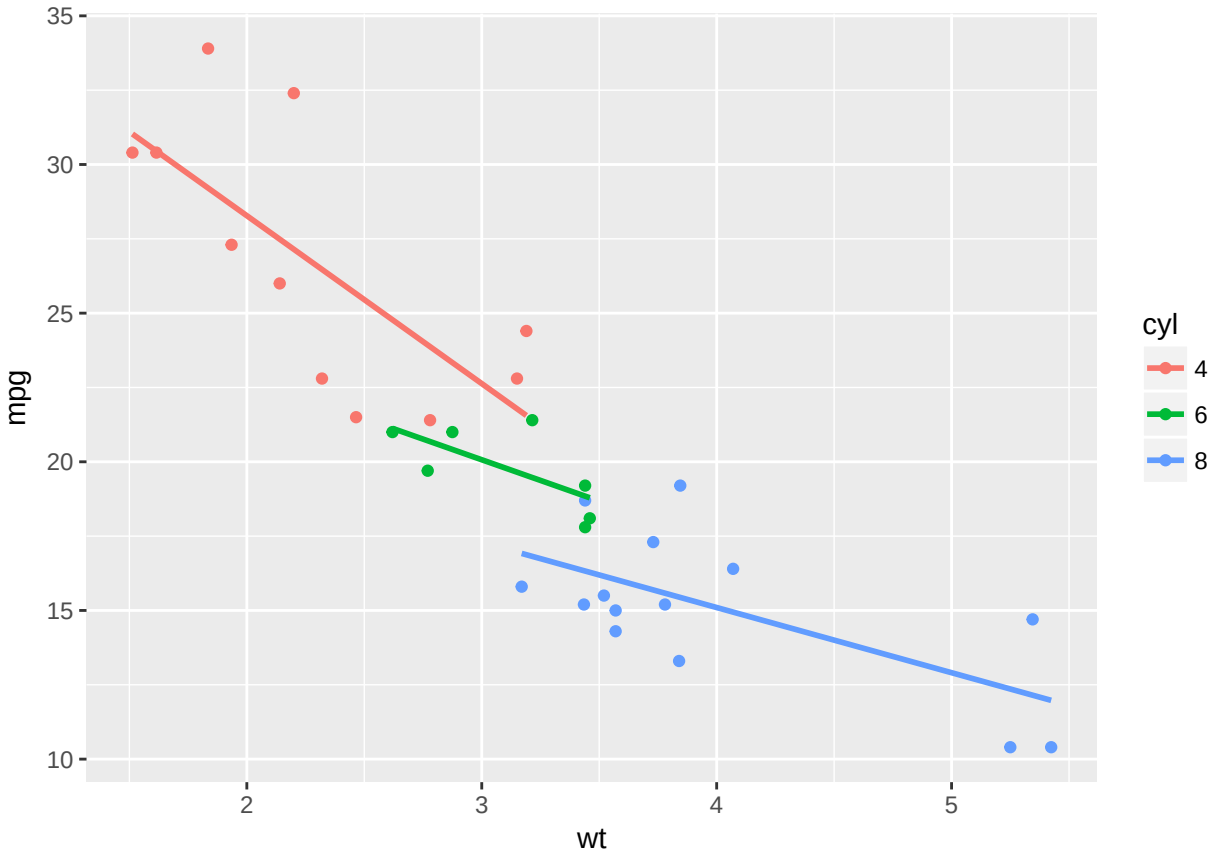
```
legend(x = 5, y = 33, legend = levels(mtcars$cyl),
       col = 1:3, pch = 1, bty = "n")
```



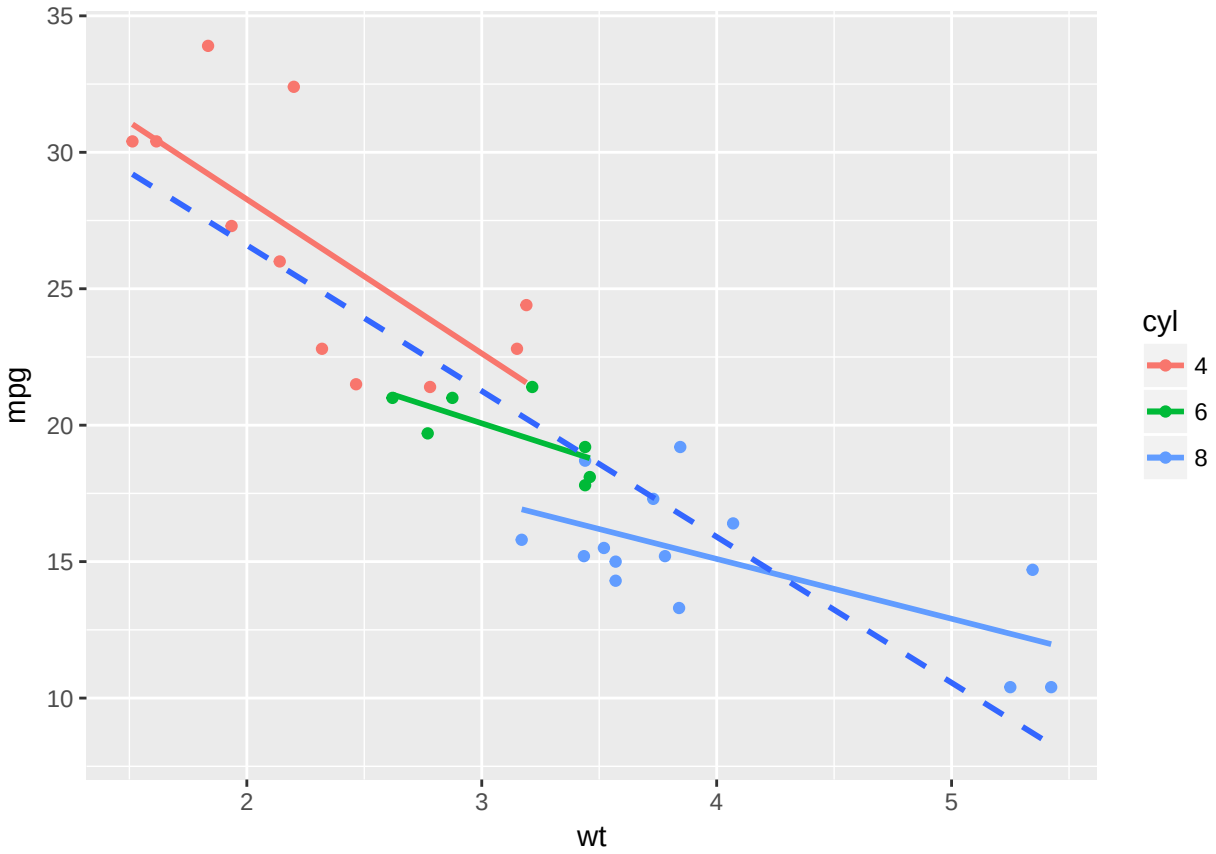

```
# Plot 1: add geom_point() to this command to create a scatter plot
ggplot(mtcars, aes(x = wt, y = mpg, col = cyl)) +
  geom_point()
```




```
# Plot 2: include the lines of the linear models, per cyl
ggplot(mtcars, aes(x = wt, y = mpg, col = cyl)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```




```
# Plot 3: include a lm for the entire dataset in its whole
ggplot(mtcars, aes(x = wt, y = mpg, col = cyl)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_smooth(aes(group = 1), method = "lm", se = FALSE, linetype = 2)
```



5.12 Plotting the ggplot2 way

In the video, Rick showed you different ggplot2 calls to plot two groups of data onto the same plot:

- Option 1 ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) + geom_point() + geom_point(aes(x = Petal.Length, y = Petal.Width), col = "red")
- Option 2 ggplot(iris.wide, aes(x = Length, y = Width, col = Part)) + geom_point() Which one is preferable? Both iris and iris.wide are available in the workspace, so you can experiment in the R Console straight away!

```
# Correct! You're starting to grasp the ggplot2 philosophy; that's great!
```

5.13 Variables to visuals, part 1

So far you've seen four different forms of the iris dataset: iris, iris.wide, iris.wide2 and iris.tidy. Don't let all these different forms confuse you! It's exactly the same data, just rearranged so that your plotting functions become easier.

To see this in action, consider the plot in the graphics device at right. Which form of the dataset would be the most appropriate to use here?

5.13.1 Instructions

Look at the structures of iris, iris.wide and iris.tidy using str(). Fill in the ggplot function with the appropriate data frame and variable names. The variable names of the aesthetics of the plot will match the ones you

found using the str() command in the previous step.

```
# Iris library

iris$Flower <- 1:nrow(iris)

iris.wide <- iris %>%
  gather(key, value, -Species, -Flower) %>%
  separate(key, c("Part", "Measure"), "\\.") %>%
  spread(Measure, value)

iris.tidy <- iris %>%
  gather(key, Value, -Species) %>%
  separate(key, c("Part", "Measure"), "\\.")
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 150 rows
## [601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615,
## 616, 617, 618, 619, 620, ...].
```

```
# Consider the structure of iris, iris.wide and iris.tidy (in that order)
str(iris)
```

```
## 'data.frame':    150 obs. of  6 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Flower      : int  1 2 3 4 5 6 7 8 9 10 ...
```

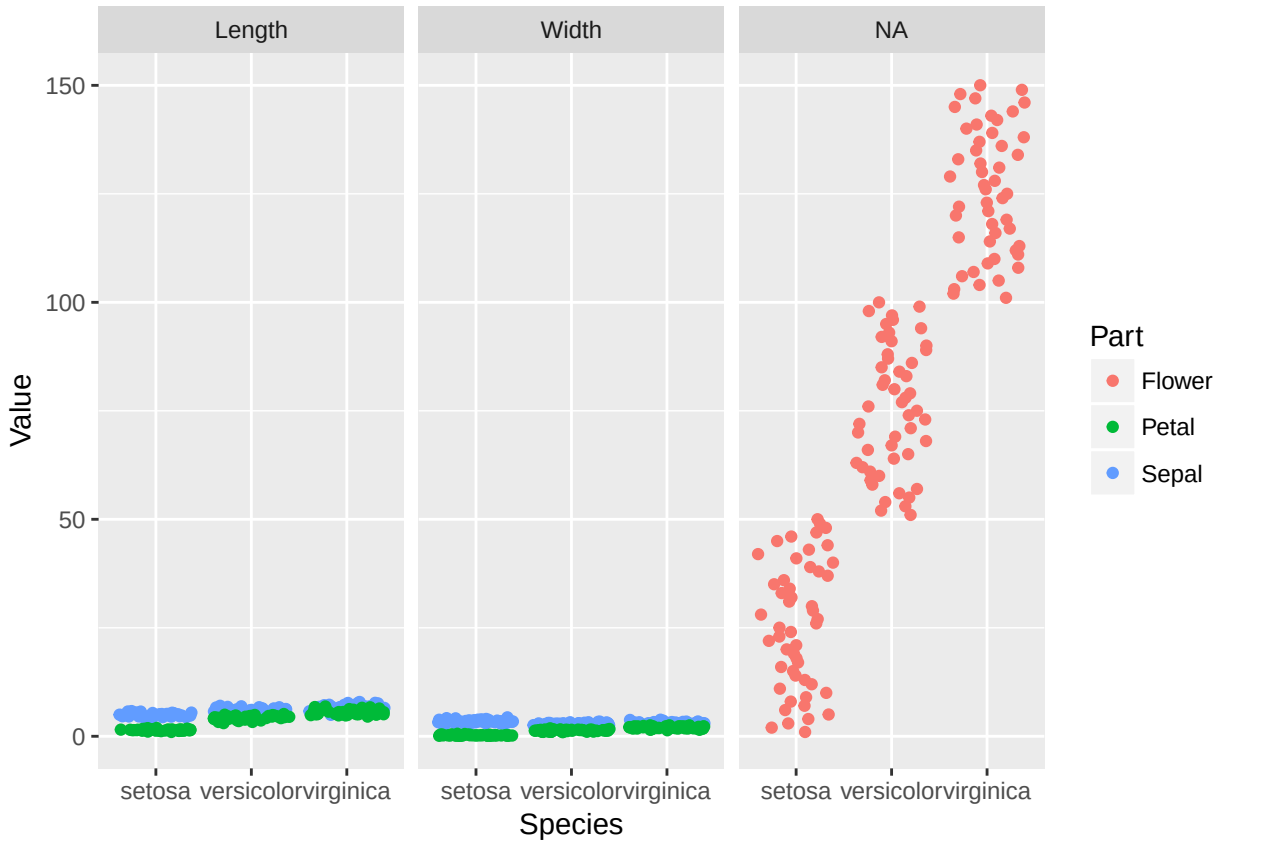
```
str(iris.wide)
```

```
## 'data.frame':    300 obs. of  5 variables:
##  $ Species: Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Flower : int  1 1 2 2 3 3 4 4 5 5 ...
##  $ Part   : chr  "Petal" "Sepal" "Petal" "Sepal" ...
##  $ Length : num  1.4 5.1 1.4 4.9 1.3 4.7 1.5 4.6 1.4 5 ...
##  $ Width  : num  0.2 3.5 0.2 3 0.2 3.2 0.2 3.1 0.2 3.6 ...
```

```
str(iris.tidy)
```

```
## 'data.frame':    750 obs. of  4 variables:
##  $ Species: Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Part   : chr  "Sepal" "Sepal" "Sepal" "Sepal" ...
##  $ Measure: chr  "Length" "Length" "Length" "Length" ...
##  $ Value  : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

```
# Think about which dataset you would use to get the plot shown right
# Fill in the ___ to produce the plot given to the right
ggplot(iris.tidy, aes(x = Species, y = Value, col = Part)) +
  geom_jitter() +
  facet_grid(. ~ Measure)
```




```
# Tidy work! Ggplots always want one measurement per row of the data frame.
```

5.14 Variables to visuals, part 1b

In the last exercise you saw how iris.tidy was used to make a specific plot. It's important to know how to rearrange your data in this way so that your plotting functions become easier. In this exercise you'll use functions from the tidyr package to convert iris to iris.tidy.

The resulting iris.tidy data should look as follows:

```
#      Species  Part Measure Value
#    1  setosa Sepal  Length   5.1
#    2  setosa Sepal  Length   4.9
#    3  setosa Sepal  Length   4.7
#    4  setosa Sepal  Length   4.6
#    5  setosa Sepal  Length   5.0
#    6  setosa Sepal  Length   5.4
#    ...
```

You can have a look at the iris dataset by typing head(iris) in the console.

Note: If you're not familiar with %>%, gather() and separate(), you may want to take the Cleaning Data in R course. In a nutshell, a dataset is called tidy when every row is an observation and every column is a variable. The gather() function moves information from the columns to the rows. It takes multiple columns and gathers them into a single column by adding rows. The separate() function splits one column into two or more columns according to a pattern you define. Lastly, the %>% (or "pipe") operator passes the result of the left-hand side as the first argument of the function on the right-hand side.

5.14.1 Instructions

You'll use two functions from the tidyr package:

gather() rearranges the data frame by specifying the columns that are categorical variables with a - notation. Complete the command. Notice that only one variable is categorical in iris. separate() splits up the new key column, which contains the former headers, according to .. The new column names "Part" and "Measure" are given in a character vector. Don't forget the quotes.

```
# Load the tidyr package

head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Flower
## 1          5.1         3.5          1.4         0.2  setosa      1
## 2          4.9         3.0          1.4         0.2  setosa      2
## 3          4.7         3.2          1.3         0.2  setosa      3
## 4          4.6         3.1          1.5         0.2  setosa      4
## 5          5.0         3.6          1.4         0.2  setosa      5
## 6          5.4         3.9          1.7         0.4  setosa      6
```

```
# Fill in the ___ to produce to the correct iris.tidy dataset
iris.tidy <- iris %>%
  gather(key, Value, -Species) %>%
  separate(key, c("Part", "Measure"), "\\.")
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 150 rows
## [601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615,
## 616, 617, 618, 619, 620, ...].
```

5.15 Variables to visuals, part 2

Here you'll take a look at another plot variant, shown at right. Which of your data frames would be used to produce this plot?

5.15.1 Instructions

Look at the heads of iris, iris.wide and iris.tidy using head(). Fill in the ggplot function with the appropriate data frame and variable names. The names of the aesthetics of the plot will match with variable names in your dataset. The previous instruction will help you match variable names in datasets with the ones in the plot.

```
# The 3 data frames (iris, iris.wide and iris.tidy) are available in your environment
# Execute head() on iris, iris.wide and iris.tidy (in that order)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Flower
## 1          5.1         3.5          1.4         0.2  setosa      1
## 2          4.9         3.0          1.4         0.2  setosa      2
## 3          4.7         3.2          1.3         0.2  setosa      3
## 4          4.6         3.1          1.5         0.2  setosa      4
## 5          5.0         3.6          1.4         0.2  setosa      5
## 6          5.4         3.9          1.7         0.4  setosa      6
```

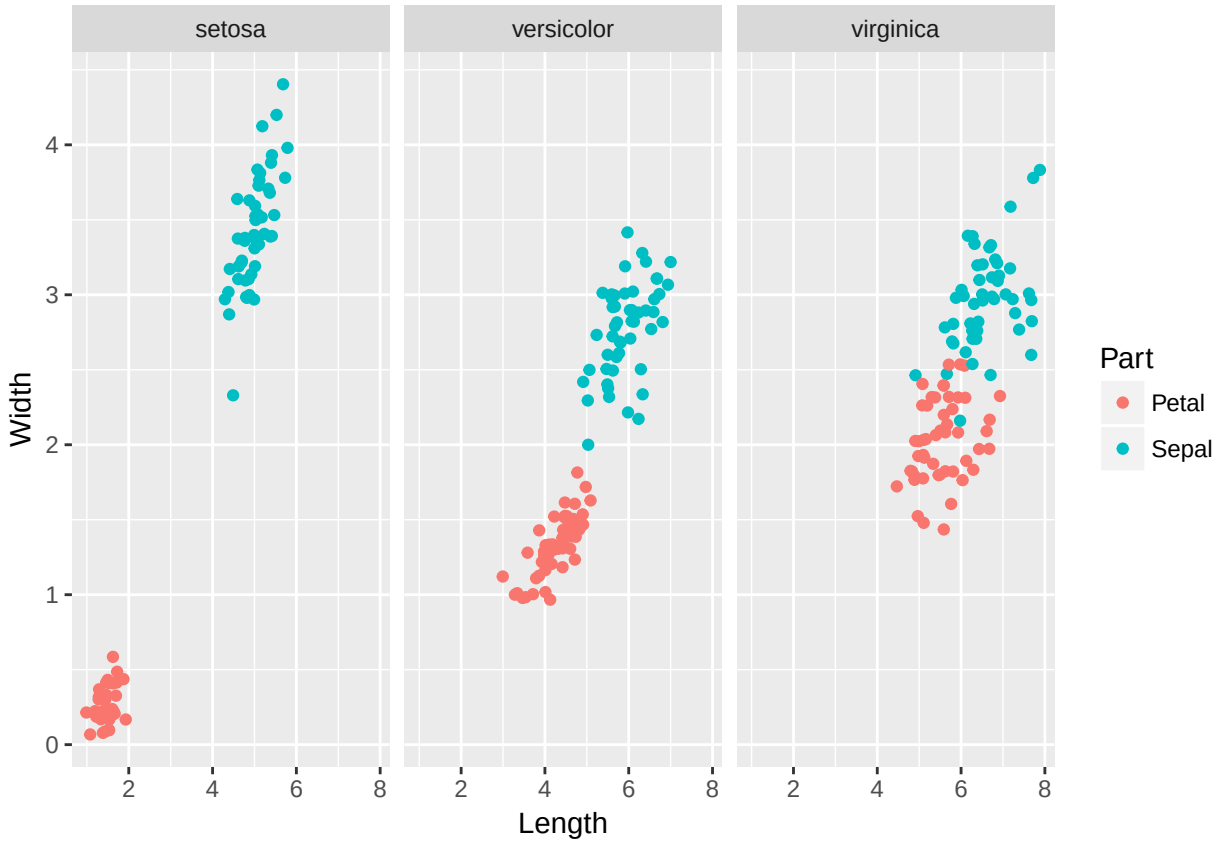
```
head(iris.wide)
```

```
##   Species Flower  Part Length Width
## 1  setosa      1 Petal    1.4   0.2
## 2  setosa      1 Sepal    5.1   3.5
## 3  setosa      2 Petal    1.4   0.2
## 4  setosa      2 Sepal    4.9   3.0
## 5  setosa      3 Petal    1.3   0.2
## 6  setosa      3 Sepal    4.7   3.2
```

```
head(iris.tidy)
```

```
##   Species  Part Measure Value
## 1  setosa Sepal  Length   5.1
## 2  setosa Sepal  Length   4.9
## 3  setosa Sepal  Length   4.7
## 4  setosa Sepal  Length   4.6
## 5  setosa Sepal  Length   5.0
## 6  setosa Sepal  Length   5.4
```

```
# Think about which dataset you would use to get the plot shown right
# Fill in the ___ to produce the plot given to the right
ggplot(iris.wide, aes(x = Length, y = Width, color = Part)) +
  geom_jitter() +
  facet_grid(. ~ Species)
```



5.16 Variables to visuals, part 2b

In the last exercise you saw how iris.wide was used to make a specific plot. You also saw previously how you can derive iris.tidy from iris. Now you'll move on to produce iris.wide.

The head of the iris.wide should look like this in the end:

```
#   Species  Part Length Width
#1  setosa Petal    1.4   0.2
#2  setosa Petal    1.4   0.2
#3  setosa Petal    1.3   0.2
#4  setosa Petal    1.5   0.2
#5  setosa Petal    1.4   0.2
#6  setosa Petal    1.7   0.4
#...
```

You can have a look at the iris dataset by typing head(iris) in the console.

5.16.1 Instructions

Before you begin, you need to add a new column called Flower that contains a unique identifier for each row in the data frame. This is because you'll rearrange the data frame afterwards and you need to keep track of which row, or which specific flower, each value came from. It's done for you, no need to add anything yourself. gather() rearranges the data frame by specifying the columns that are categorical variables with a - notation. In this case, Species and Flower are categorical. Complete the command. separate() splits up the new key column, which contains the former headers, according to .. The new column names "Part" and "Measure" are given in a character vector. The last step is to use spread() to distribute the new Measure column and associated value column into two columns.

```
# Add column with unique ids (don't need to change)
iris$Flower <- 1:nrow(iris)

# Fill in the ___ to produce to the correct iris.wide dataset
iris.wide <- iris %>%
  gather(key, value, -Species, -Flower) %>%
  separate(key, c("Part", "Measure"), "\\.") %>%
  spread(Measure, value)
```

5.17 All about aesthetics, part 1

In the video you saw 9 visible aesthetics. Let's apply them to a categorical variable - the cylinders in mtcars, cyl.

(You'll consider line type when you encounter line plots in the next chapter).

These are the aesthetics you can consider within aes() in this chapter: x, y, color, fill, size, alpha, labels and shape.

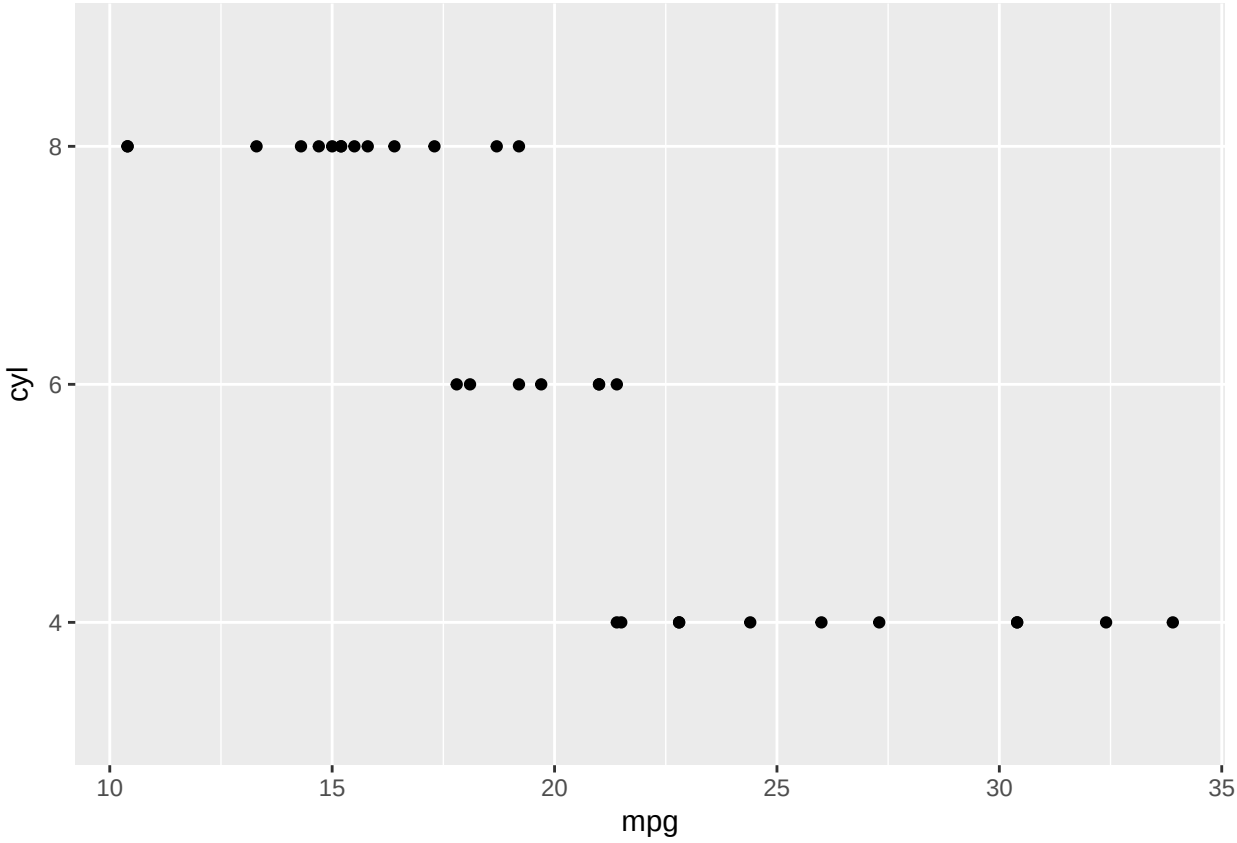
In the following exercise you can assume that the cyl column is categorical. It has already been transformed into a factor for you.

5.17.1 Instructions

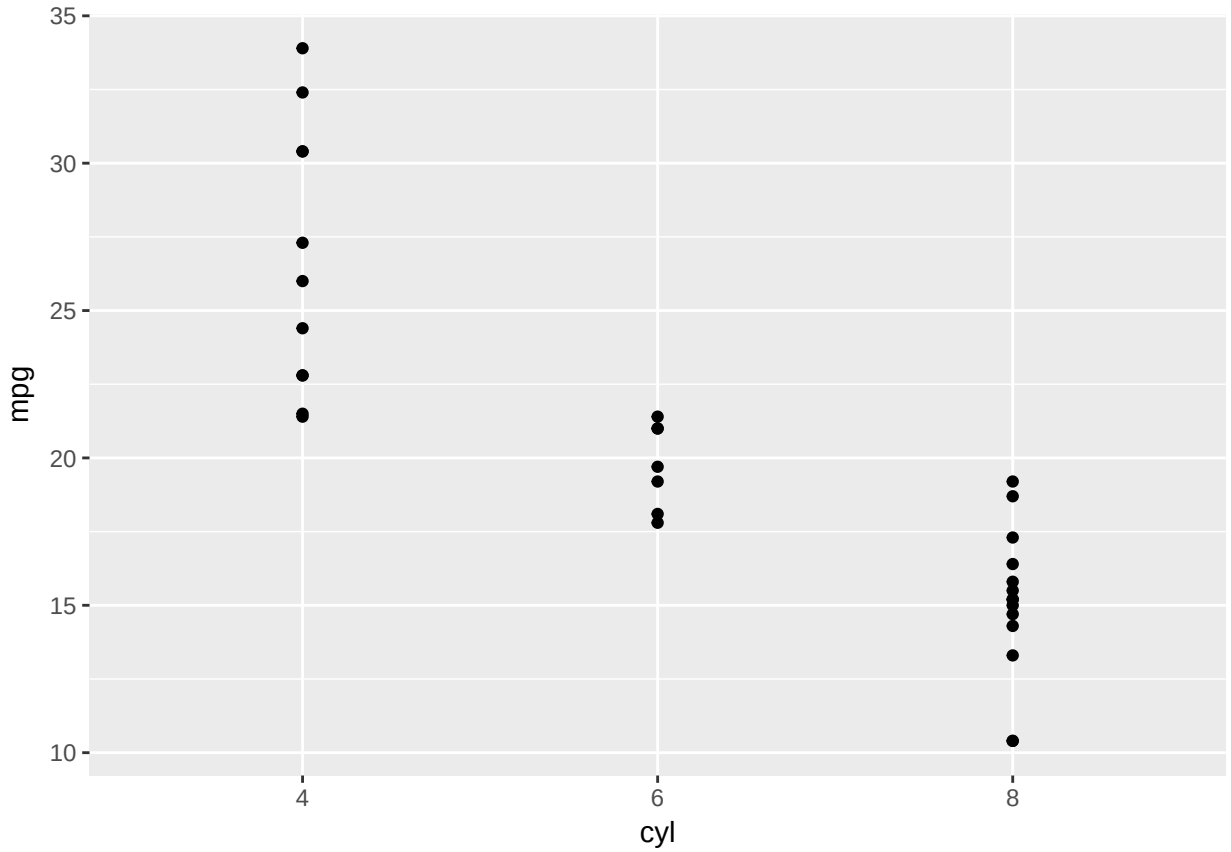
The mtcars data frame is available in your workspace. For each of the following four plots, use geom_point():

1 - Map mpg onto the x aesthetic, and cyl onto the y. 2 - Reverse the mappings of the first plot. 3 - Map wt onto x, mpg onto y, and cyl onto color. Modify the previous plot by changing the shape argument of the geom to 1 and increase the size to 4. These are attributes that you should specify inside geom_point().

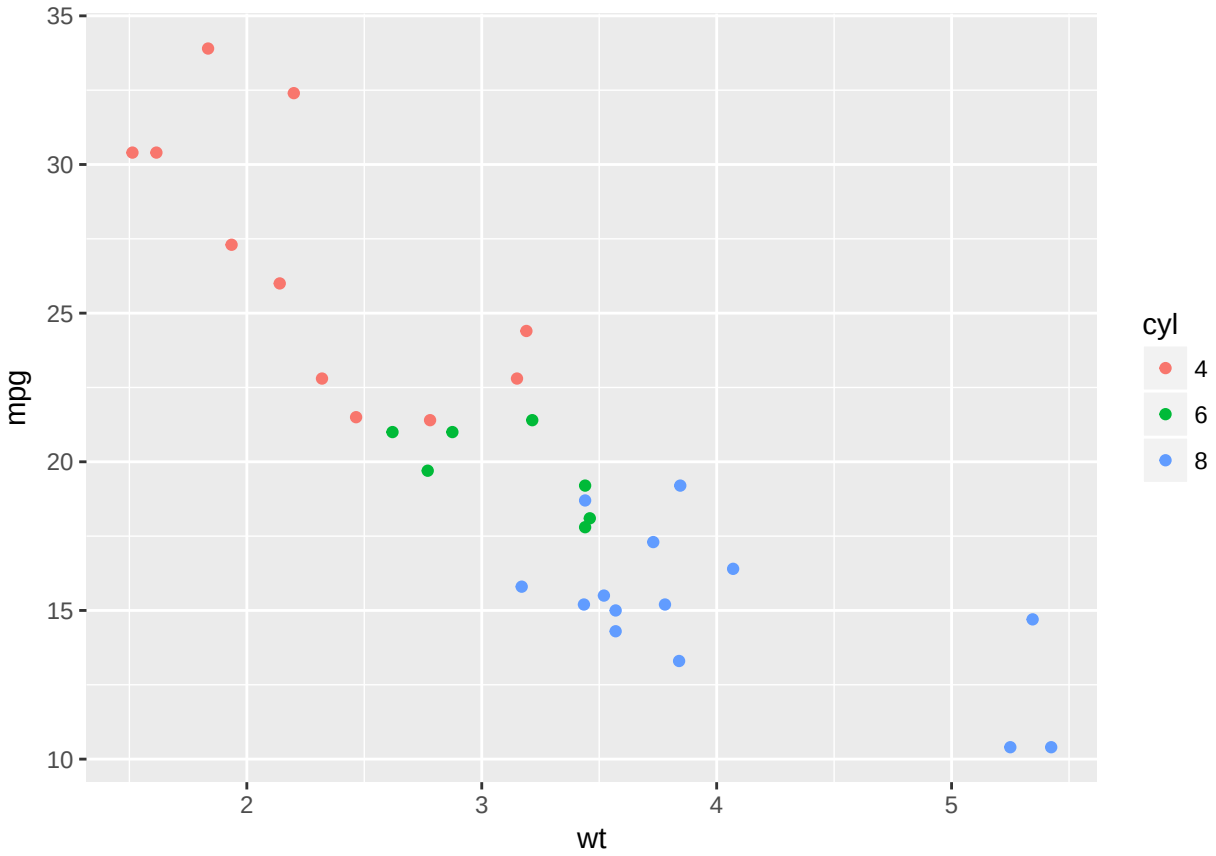
```
# 1 - Map mpg to x and cyl to y
ggplot(mtcars, aes(x = mpg, y = cyl)) +
  geom_point()
```



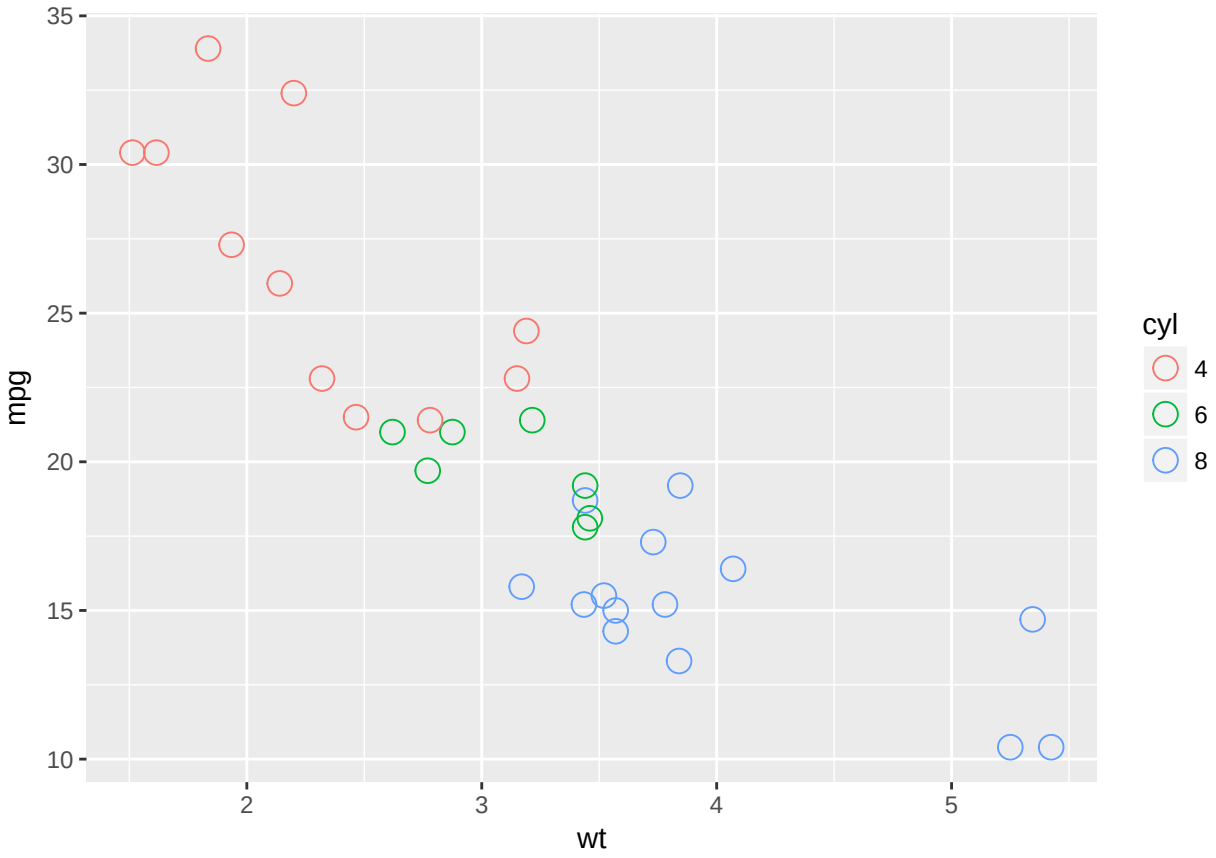

```
# 2 - Reverse: Map cyl to x and mpg to y
ggplot(mtcars, aes(x = cyl, y = mpg)) +
  geom_point()
```




```
# 3 - Map wt to x, mpg to y and cyl to col
ggplot(mtcars, aes(x = wt, y = mpg, col=cyl)) +
  geom_point()
```




```
# 4 - Change shape and size of the points in the above plot
ggplot(mtcars, aes(x = wt, y = mpg, col = cyl)) +
  geom_point(shape=1, size=4)
```



5.18 All about aesthetics, part 2

The color aesthetic typically changes the outside outline of an object and the fill aesthetic is typically the inside shading. However, as you saw in the last exercise, geom_point() is an exception. Here you use color, instead of fill for the inside of the point. But it's a bit subtler than that.

Which shape to use? The default geom_point() uses shape = 19 (a solid circle with an outline the same colour as the inside). Good alternatives are shape = 1 (hollow) and shape = 16 (solid, no outline). These all use the col aesthetic (don't forget to set alpha for solid points).

A really nice alternative is shape = 21 which allows you to use both fill for the inside and col for the outline! This is a great little trick for when you want to map two aesthetics to a dot.

What happens when you use the wrong aesthetic mapping? This is a very common mistake! The code from the previous exercise is in the editor. Using this as your starting point complete the instructions.

5.18.1 Instructions

Note: In the mtcars dataset, cyl and am have been converted to factor for you.

1 - Copy & paste the first plot's code. Change the aesthetics so that cyl maps to fill rather than col. 2 - Copy & paste the second plot's code. In geom_point() change the shape argument to 21 and add an alpha argument set to 0.6. 3 - Copy & paste the third plot's code. In the ggplot() aesthetics, map am to col.

```
# am and cyl are factors, wt is numeric
class(mtcars$am)
```

```
## [1] "numeric"
```

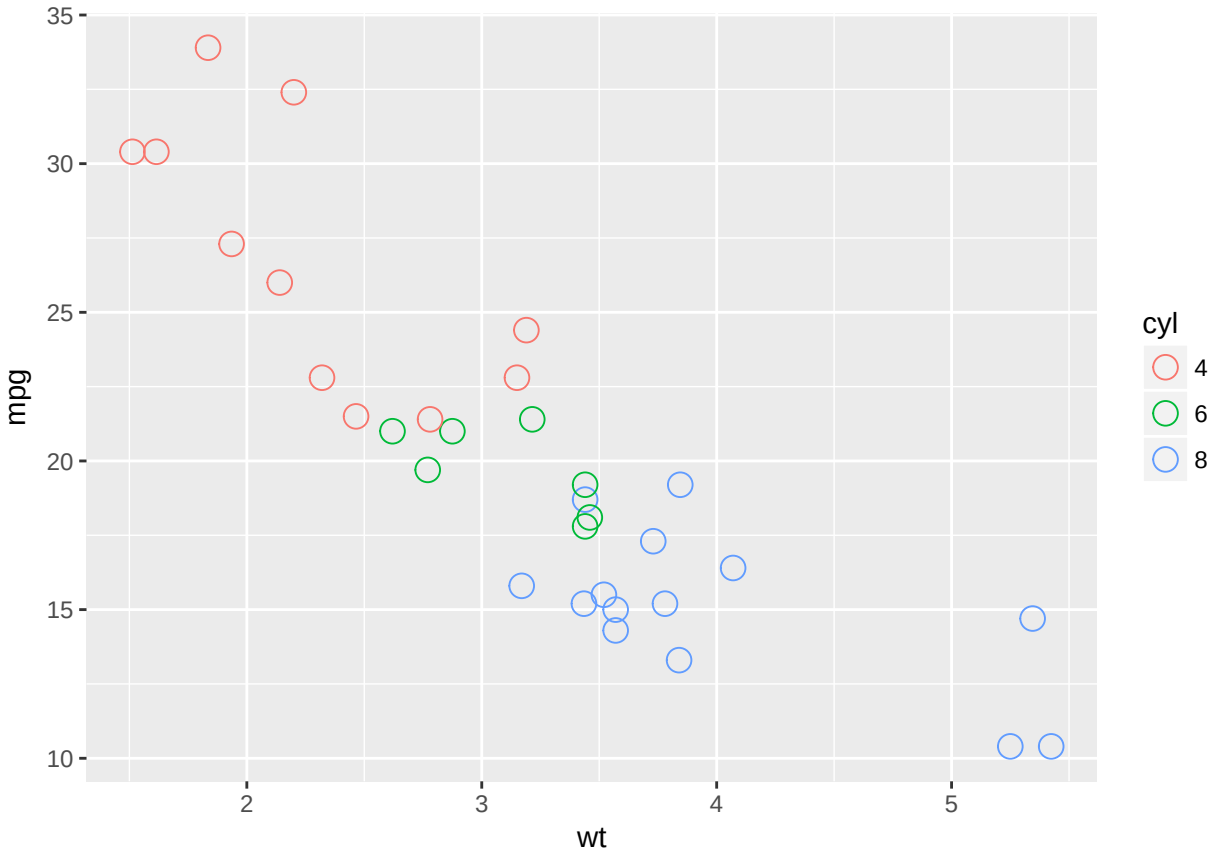
```
class(mtcars$cyl)
```

```
## [1] "factor"
```

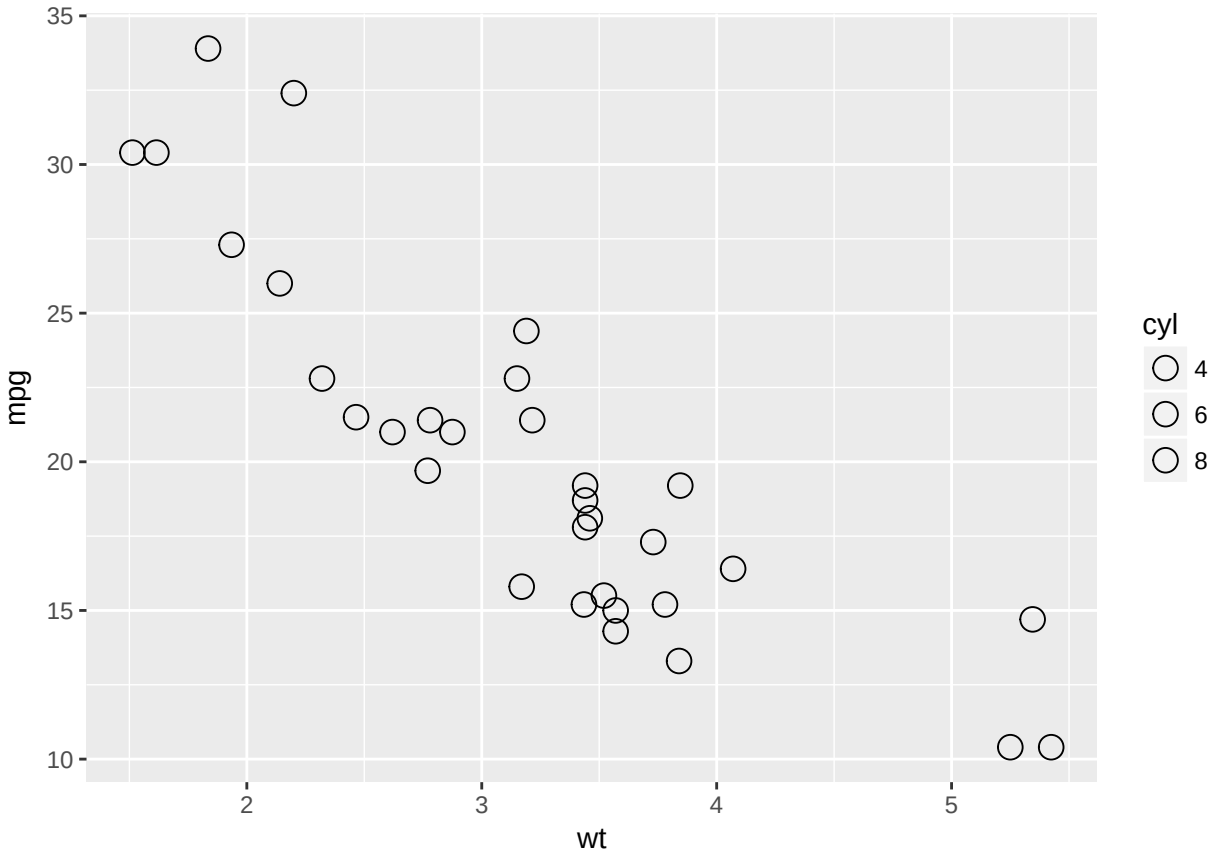
```
class(mtcars$wt)
```

```
## [1] "numeric"
```

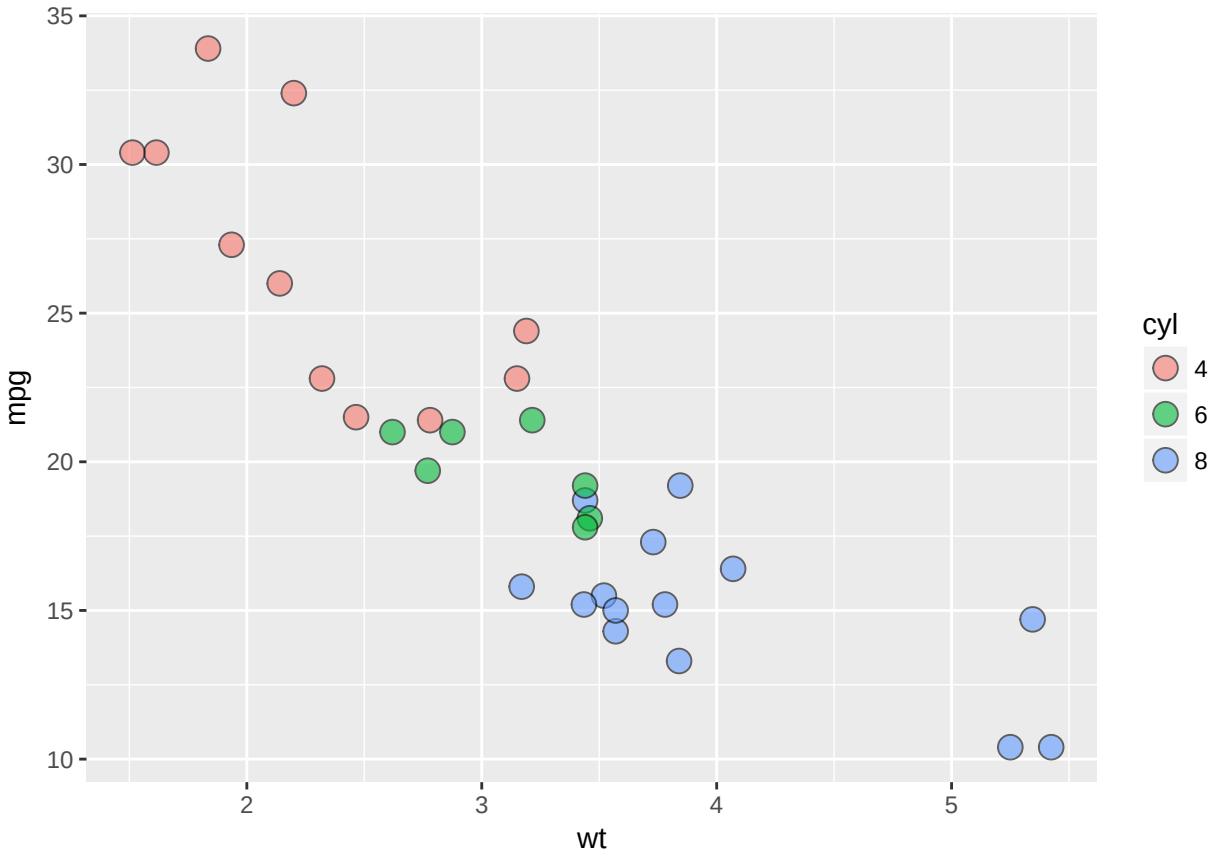
```
# From the previous exercise
ggplot(mtcars, aes(x = wt, y = mpg, col = cyl)) +
  geom_point(shape = 1, size = 4)
```



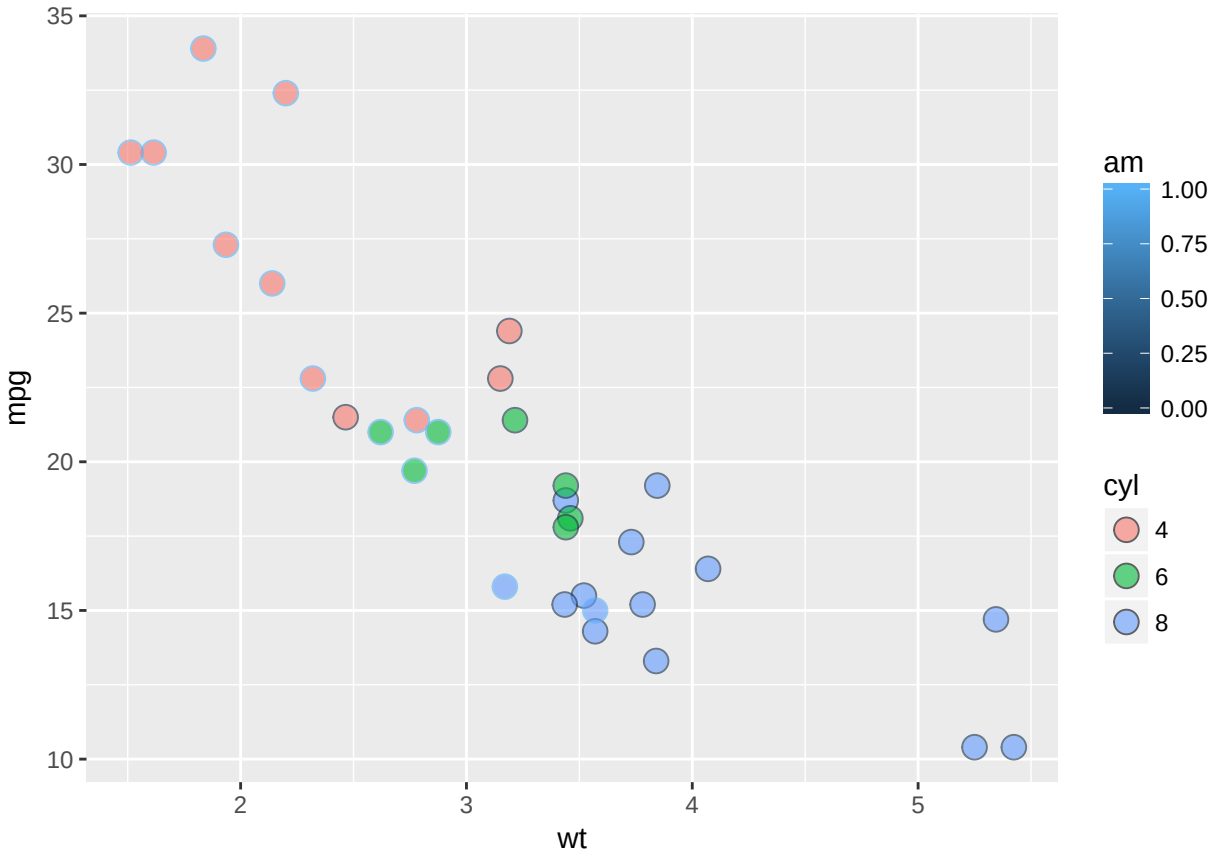

```
# 1 - Map cyl to fill
ggplot(mtcars, aes(x = wt, y = mpg, fill = cyl)) +
  geom_point(shape = 1, size = 4)
```




```
# 2 - Change shape and alpha of the points in the above plot
ggplot(mtcars, aes(x = wt, y = mpg, fill = cyl)) +
  geom_point(shape = 21, size = 4, alpha=0.6)
```




```
# 3 - Map am to col in the above plot
ggplot(mtcars, aes(x = wt, y = mpg, fill = cyl, col = am)) +
  geom_point(shape = 21, size = 4, alpha=0.6)
```




```
# Shapely coding! Notice that mapping a categorical variable onto fill doesn't
# change the colors, although a legend is generated! This is because the default
# shape for points only has a color attribute and not a fill attribute! Use fill
# when you have another shape (such as a bar), or when using a point that does
# have a fill and a color attribute, such as shape = 21, which is a circle with
# an outline. Any time you use a solid color, make sure to use alpha blending to
# account for over plotting.
```

5.19 All about aesthetics, part 3

Now that you've got some practice with incrementally building up plots, you can try to do it from scratch! The mtcars dataset is pre-loaded in the workspace.

5.19.1 Instructions

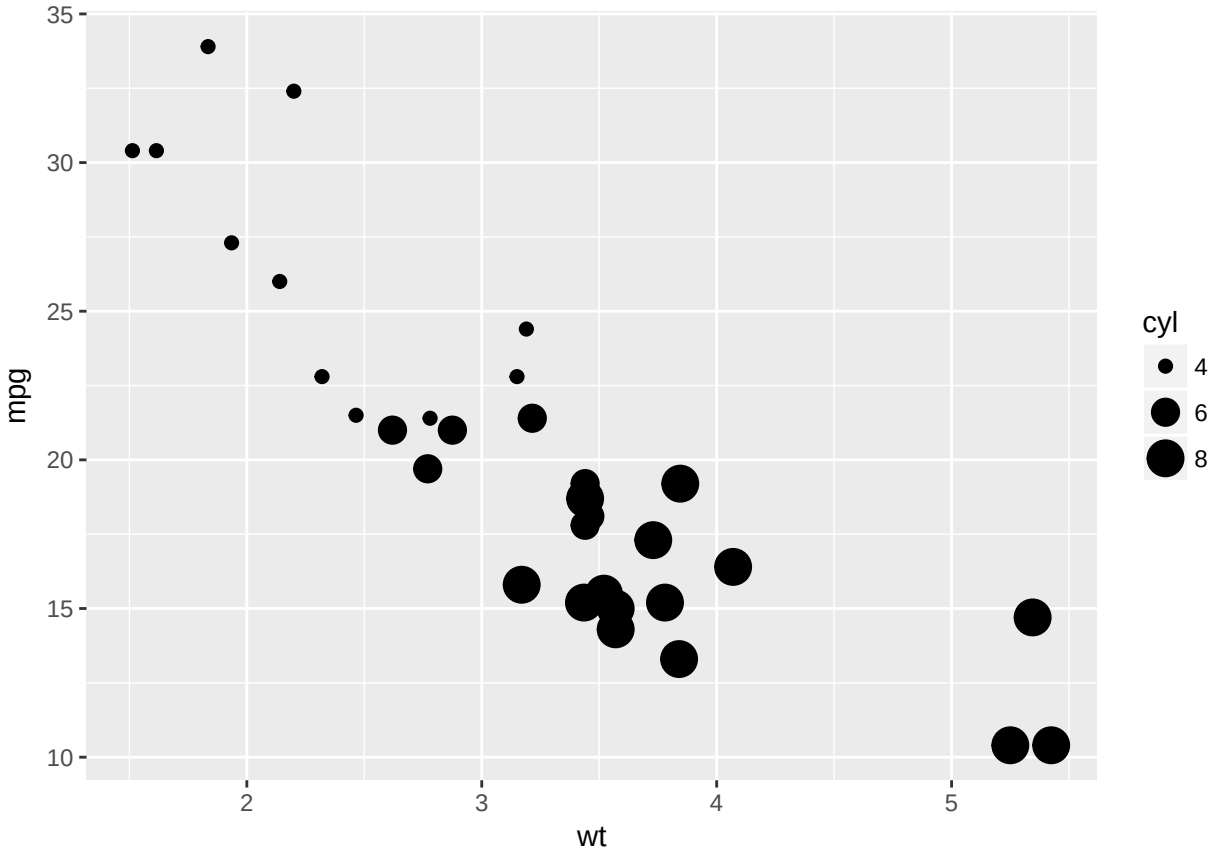
Use ggplot() to create a basic scatter plot. Inside aes(), map wt onto x and mpg onto y. Typically, you would say "mpg described by wt" or "mpg vs wt", but in aes(), it's x first, y second. Use geom_point() to make three scatter plots:

cyl on size cyl on alpha cyl on shape Try this last variant:

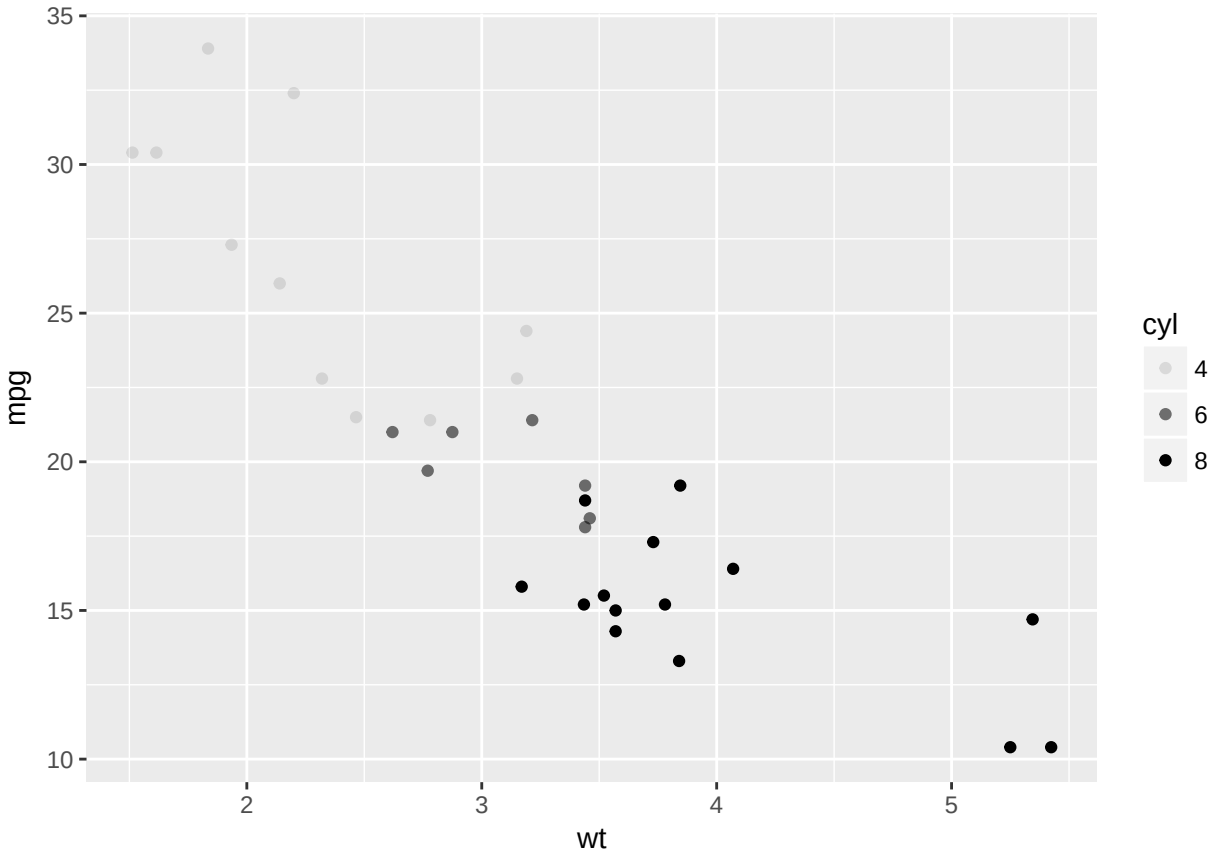
cyl on label. In order to correctly show the test (i.e. label), use geom_text().

```
# Map cyl to size
ggplot(mtcars, aes(x = wt, y = mpg, size=cyl)) +
  geom_point()
```

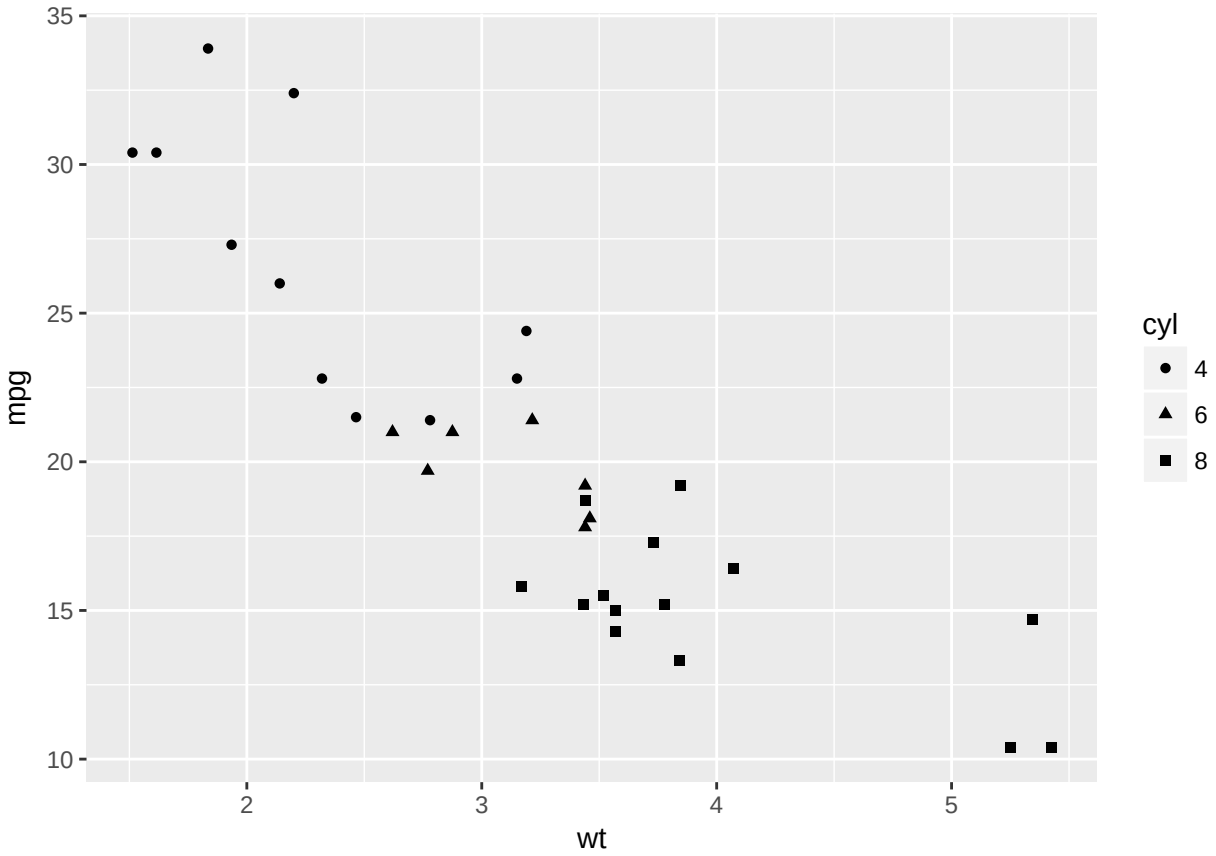
```
## Warning: Using size for a discrete variable is not advised.
```



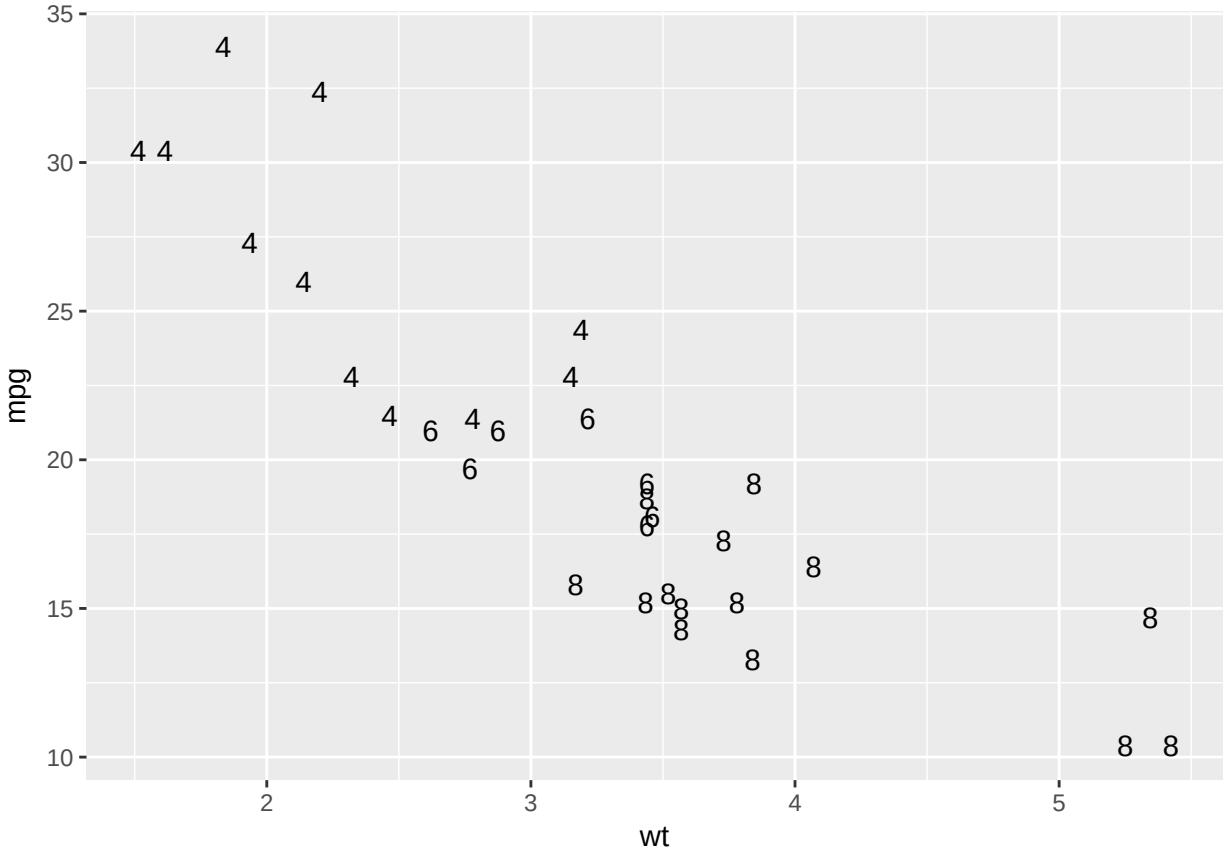

```
# Map cyl to alpha
ggplot(mtcars, aes(x = wt, y = mpg, alpha=cyl)) +
  geom_point()
```




```
# Map cyl to shape
ggplot(mtcars, aes(x = wt, y = mpg, shape=cyl)) +
  geom_point()
```




```
# Map cyl to label
ggplot(mtcars, aes(x = wt, y = mpg, label=cyl)) +
  geom_text()
```




```
# Nice! Which aesthetic do you think is the clearest for categorical data?
```

5.20 All about attributes, part 1

In the video you saw that you can use all the aesthetics as attributes. Let's see how this works with the aesthetics you used in the previous exercises: x, y, color, fill, size, alpha, label and shape.

This time you'll use these arguments to set attributes of the plot, not aesthetics. However, there are some pitfalls you'll have to watch out for: these attributes can overwrite the aesthetics of your plot!

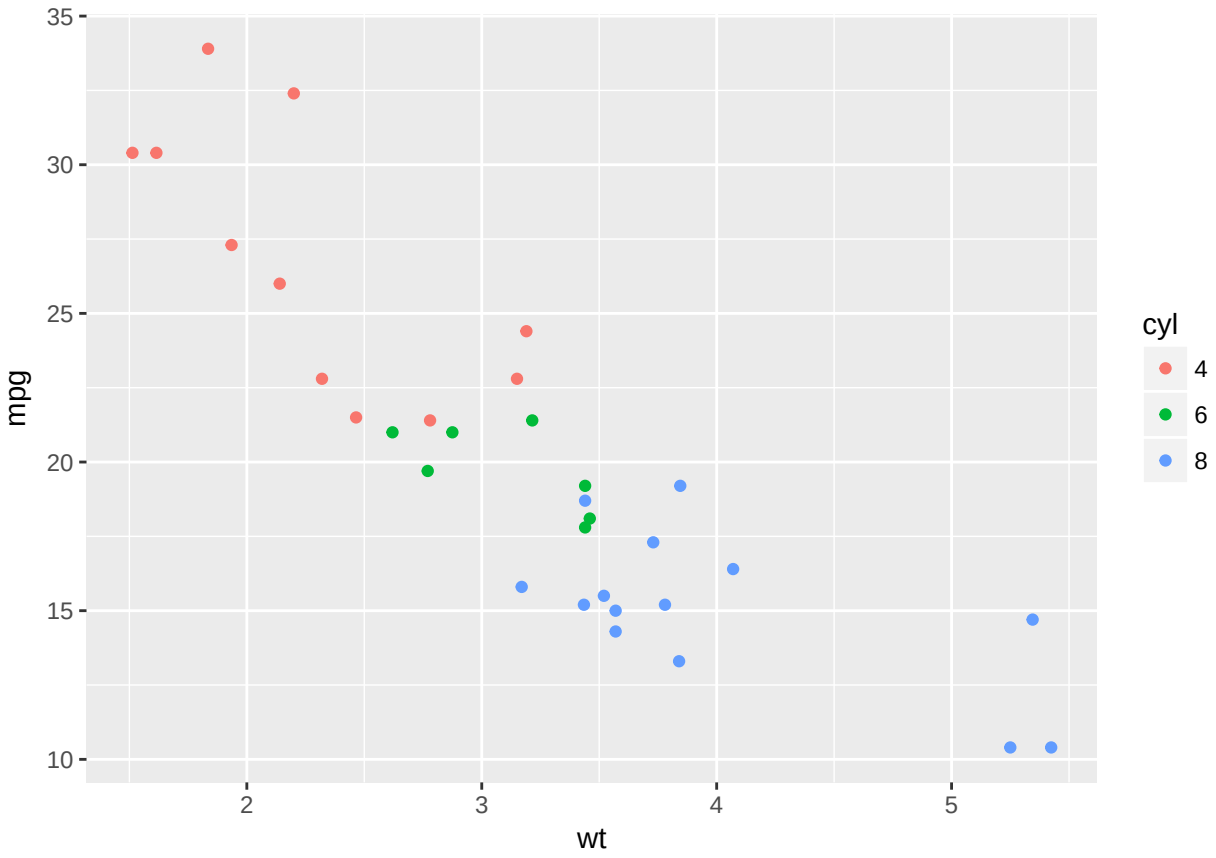
A word about shapes: In the exercise "All about aesthetics, part 2", you saw that shape = 21 results in a point that has a fill and an outline. Shapes in R can have a value from 1-25. Shapes 1-20 can only accept a color aesthetic, but shapes 21-25 have both a color and a fill aesthetic. See the pch argument in par() for further discussion.

A word about hexadecimal colours: Hexadecimal, literally "related to 16", is a base-16 alphanumeric counting system. Individual values come from the ranges 0-9 and A-F. This means there are 256 possible two-digit values (i.e. 00 - FF). Hexadecimal colours use this system to specify a six-digit code for Red, Green and Blue values ("#RRGGBB") of a colour (i.e. Pure blue: "#0000FF", black: "#000000", white: "#FFFFFF"). R can accept hex codes as valid colours.

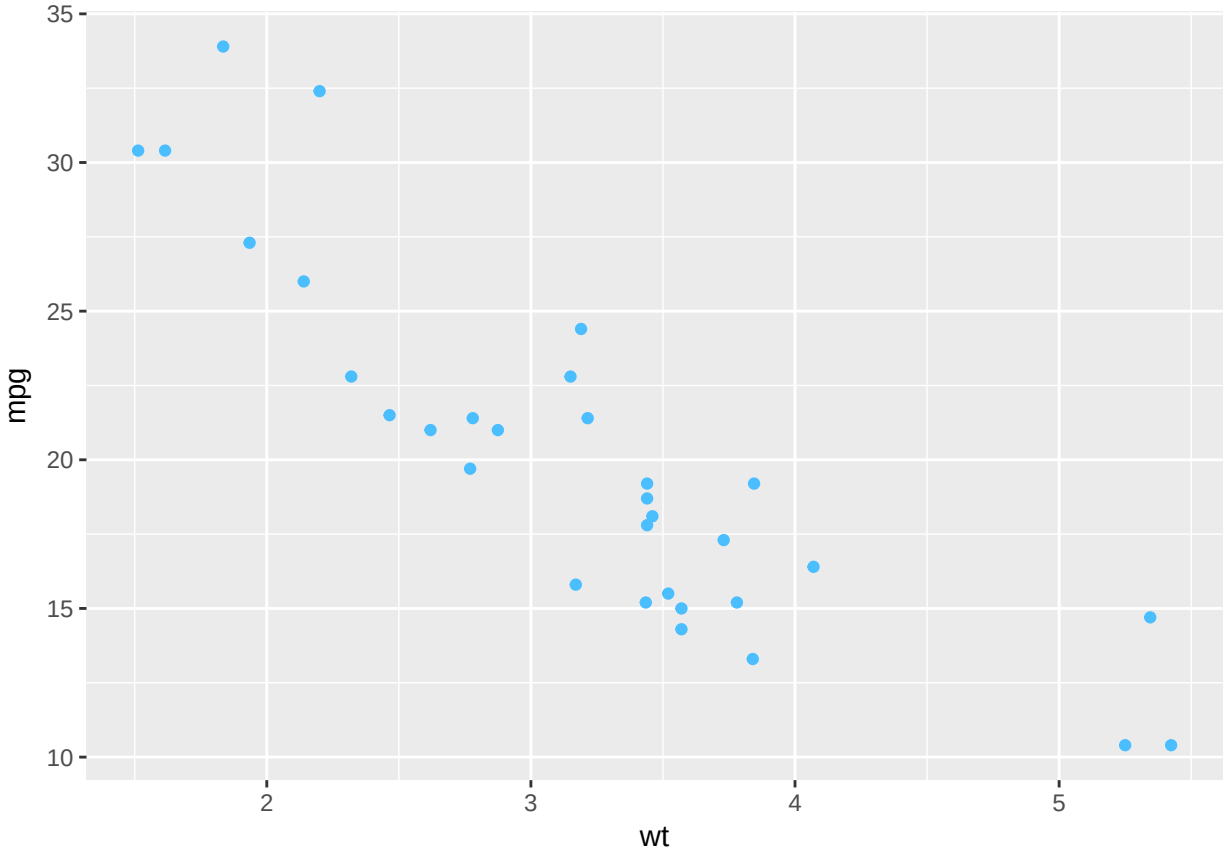
5.20.1 Instructions

1 - You will continue to work with mtcars. Use ggplot() to create a basic scatter plot: map wt onto x, mpg onto y and cyl onto color. 2 - Overwrite the color of the points inside geom_point() to my_color. Notice how this cancels out the colors given to the points by the number of cylinders! 3 - Starting with plot 2, map cyl to fill instead of col and set the attributes size to 10, shape to 23 and color to my_color inside geom_point().

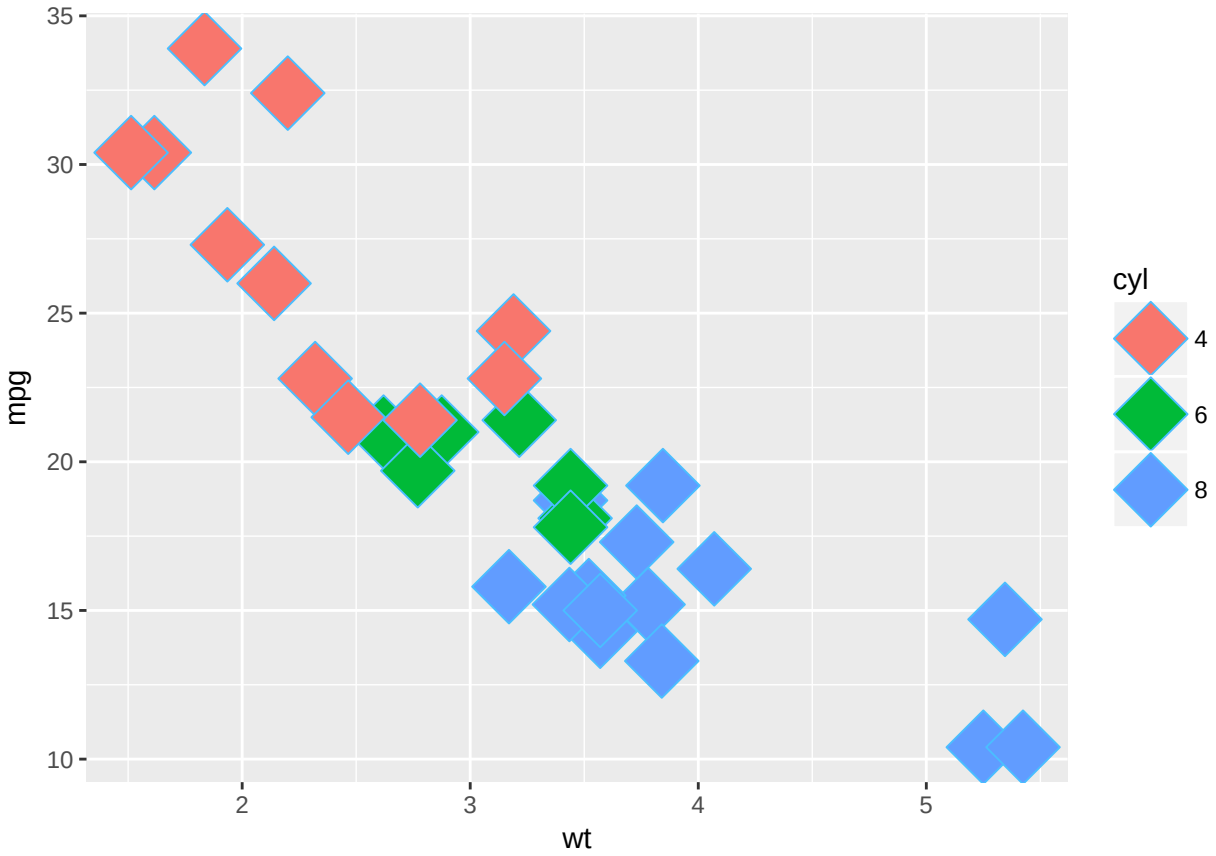
```r
# Define a hexadecimal color
my_color <- "#4ABEFF"

# 1 - First scatter plot, with col aesthetic:
ggplot(mtcars, aes(x = wt, y=mpg, col=cyl)) +
  geom_point()
```




```r
# 2 - Plot 1, but set col attributes in geom layer:
ggplot(mtcars, aes(x=wt, y=mpg, col=cyl)) +
  geom_point(color=my_color)
```




```
# 3 - Plot 2, with fill instead of col aesthetic, plut shape and size attributes in geom layer.
ggplot(mtcars, aes(x=wt, y=mpg, fill=cyl)) +
  geom_point(size=10, shape=23, color=my_color)
```




```
# Hunky-dory hex specs! Notice that if an aesthetic and an attribute are set
# with the same argument, the attribute takes precedence. Once again, you see
# that the attribute needs to match the shape and geom, the fill aesthetic
# (or attribute) will only work with certain shapes.
```

5.21 All about attributes, part 2

In the videos you saw that you can use all the aesthetics as attributes. Let's see how this works with the aesthetics you used in the previous exercises: x, y, color, fill, size, alpha, label and shape.

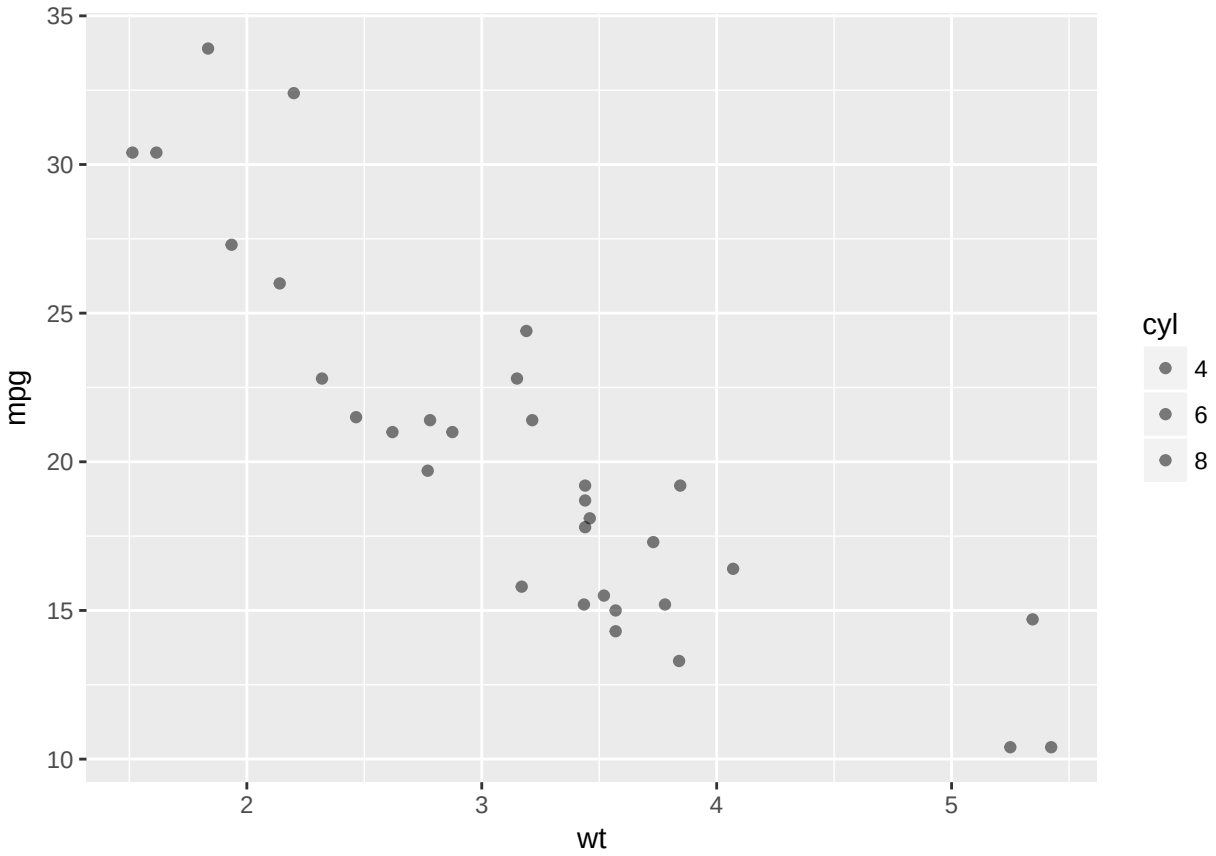
In this exercise you will set all kinds of attributes of the points!

You will continue to work with mtcars.

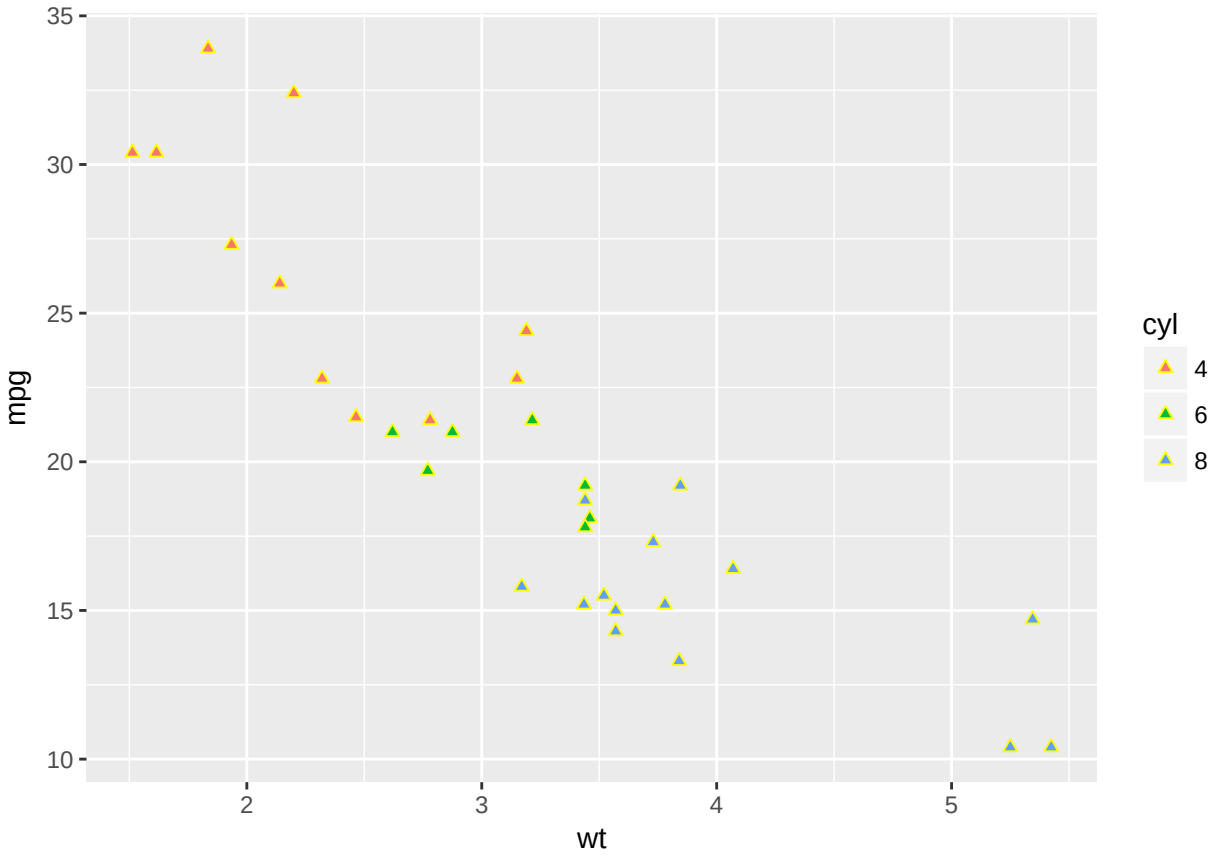
5.21.1 Instructions

Add to the first command: draw points with alpha set to 0.5. Add to the second command: draw points of shape 24 in the color yellow. Add to the third command: draw text with label rownames(mtcars) in the color red. Don't use geom_point() here! You should get a scatter plot with the names of the cars instead of points. Note: Remember to specify characters with quotation marks ("yellow", not yellow).

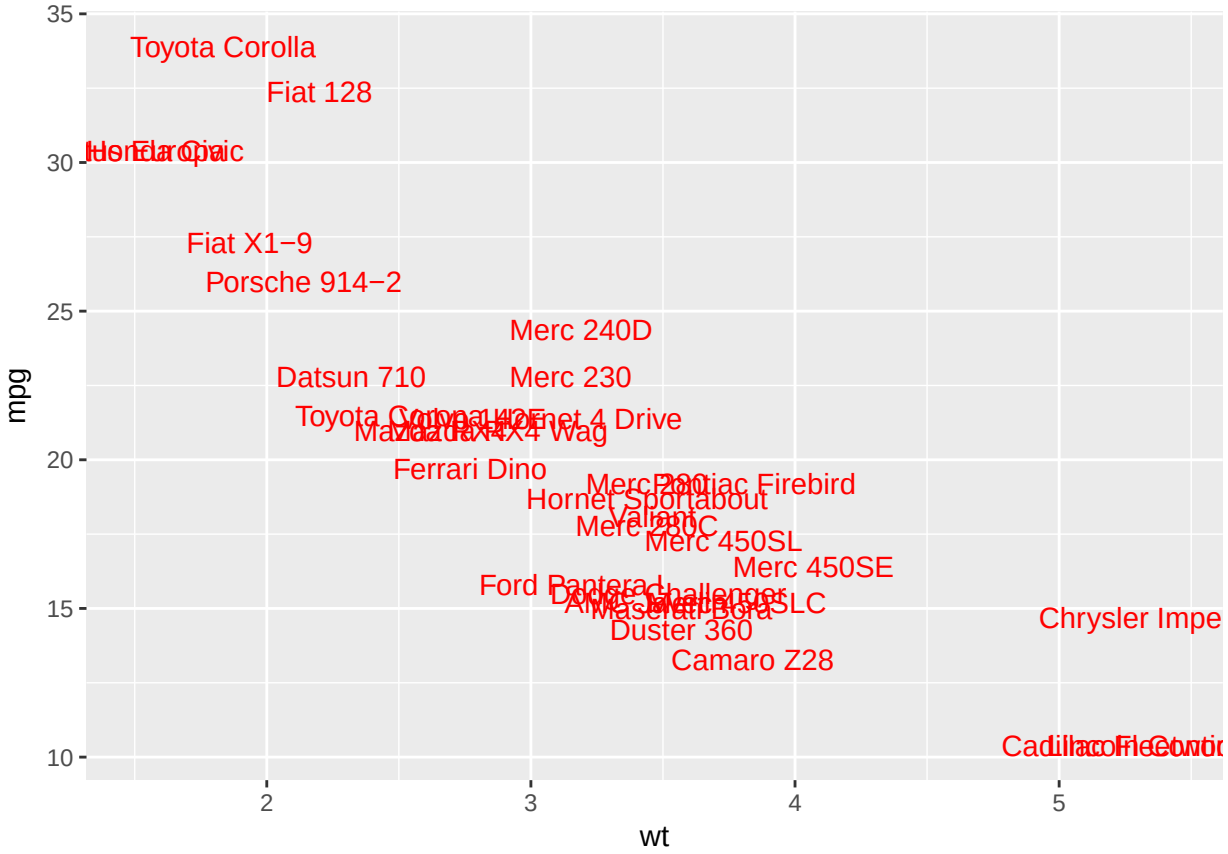
```
# Expand to draw points with alpha 0.5
ggplot(mtcars, aes(x = wt, y = mpg, fill = cyl)) +
  geom_point(alpha=0.5)
```




```
# Expand to draw points with shape 24 and color yellow
ggplot(mtcars, aes(x = wt, y = mpg, fill = cyl)) +
  geom_point(shape=24, color="yellow")
```




```
# Expand to draw text with label rownames(mtcars) and color red
ggplot(mtcars, aes(x = wt, y = mpg, fill = cyl)) +
  geom_text(label=rownames(mtcars), color="red")
```




```
# Awesome attribute assembling! Notice how ggplot2 lets you control these
# different attributes.
```

5.22 Going all out

In this exercise, you will gradually add more aesthetics layers to the plot. You're still working with the mtcars dataset, but this time you're using more features of the cars. For completeness, here is a list of all the features of the observations in mtcars:

mpg – Miles/(US) gallon cyl – Number of cylinders disp – Displacement (cu.in.) hp – Gross horsepower drat – Rear axle ratio wt – Weight (lb/1000) qsec – 1/4 mile time vs – V/S engine. am – Transmission (0 = automatic, 1 = manual) gear – Number of forward gears carb – Number of carburetors

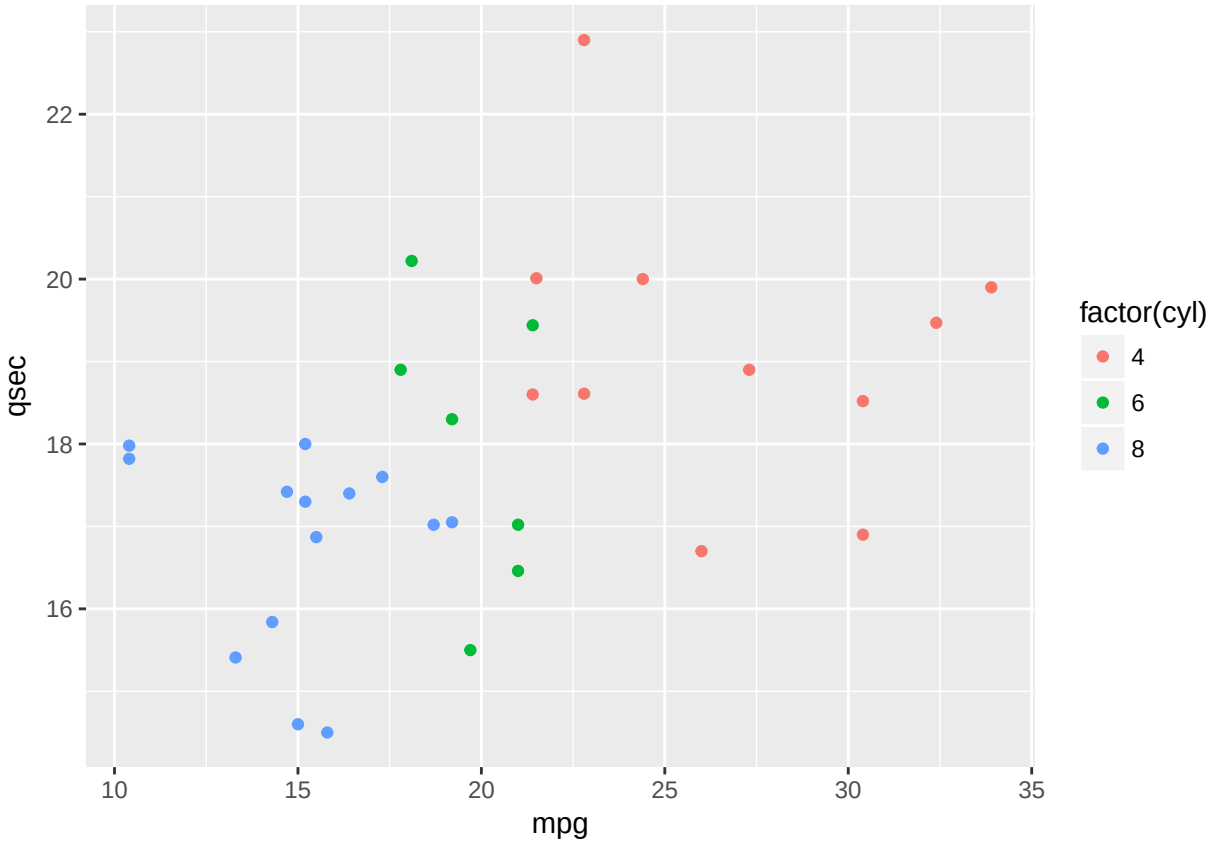
Notice that adding more aesthetics to your plot is not always a good idea. Adding aesthetic mappings to a plot will increase its complexity, and thus decrease its readability.

5.22.1 Instructions

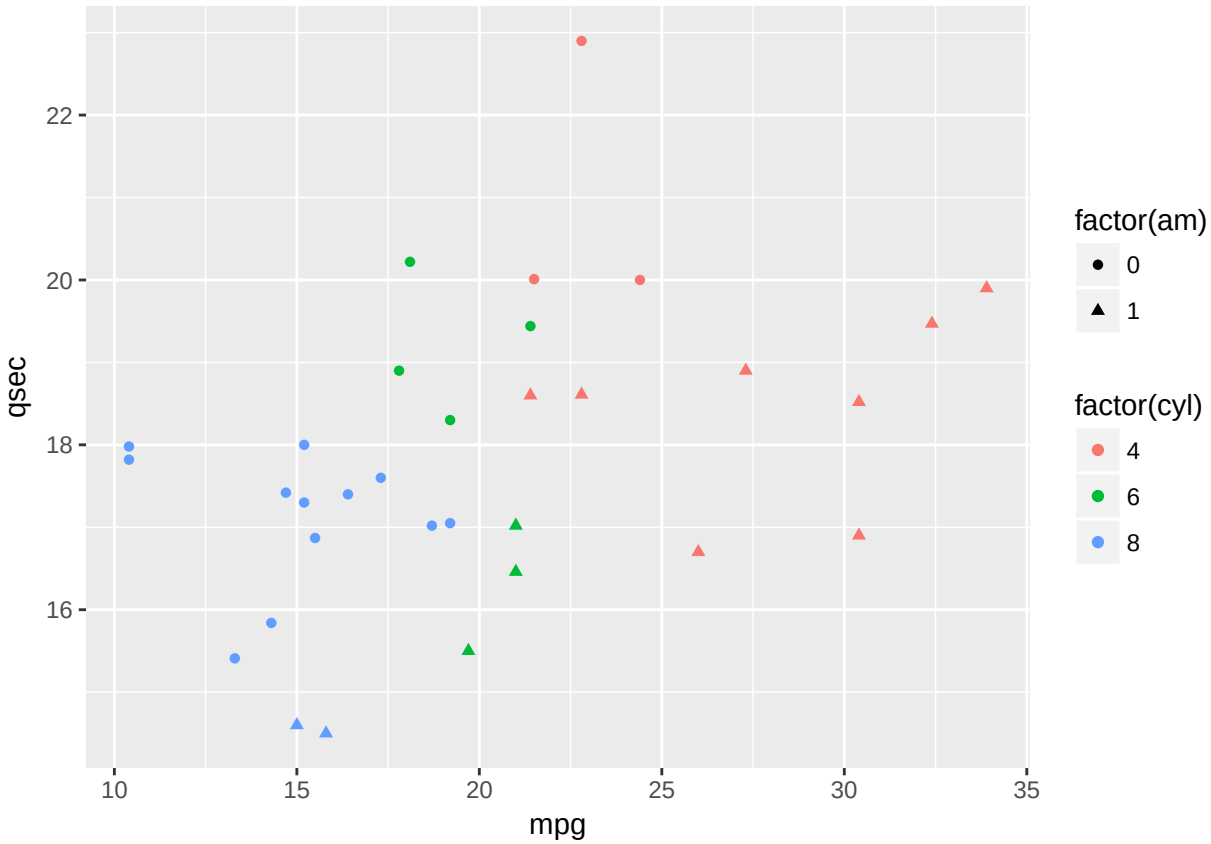
Note: In this chapter you saw aesthetics and attributes. Variables in a data frame are mapped to aesthetics in aes(). (e.g. aes(col = cyl)) within ggplot(). Visual elements are set by attributes in specific geom layers (geom_point(col = "red")). Don't confuse these two things - here you're focusing on aesthetic mappings.

Draw a scatter plot of mtcars with mpg on the x-axis, qsec on the y-axis and factor(cyl) as colors. Expand the previous plot to include factor(am) as the shape of the points. Expand the previous plot to include the ratio of horsepower to weight (i.e. (hp/wt)) as the size of the points.

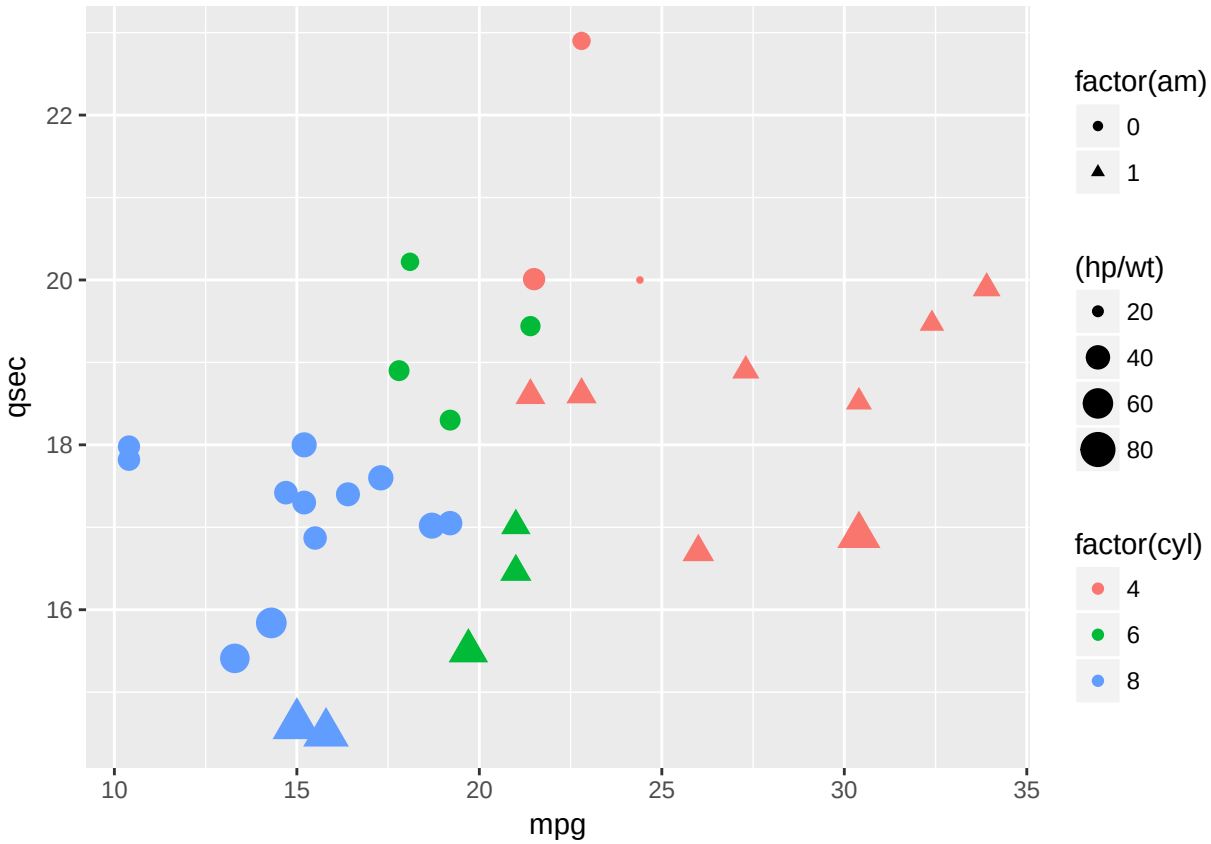
```
# Map mpg onto x, qsec onto y and factor(cyl) onto col
ggplot(mtcars, aes(x=mpg, y=qsec, col=factor(cyl))) +
  geom_point()
```




```
# Add mapping: factor(am) onto shape
ggplot(mtcars, aes(x=mpg, y=qsec, col=factor(cyl), shape=factor(am))) +
  geom_point()
```




```
# Add mapping: (hp/wt) onto size
ggplot(mtcars, aes(x=mpg, y=qsec, col=factor(cyl), shape=factor(am), size=(hp/wt))) +
  geom_point()
```




```
# That's a pretty slick plot! Between the x and y dimensions, the color, shape,
# and size of the points, your plot displays five dimensions of the dataset!
```

5.23 Position

You saw how jittering worked in the video, but bar plots suffer from their own issues of overplotting, as you'll see here. Use the "stack", "fill" and "dodge" positions to reproduce the plot in the viewer.

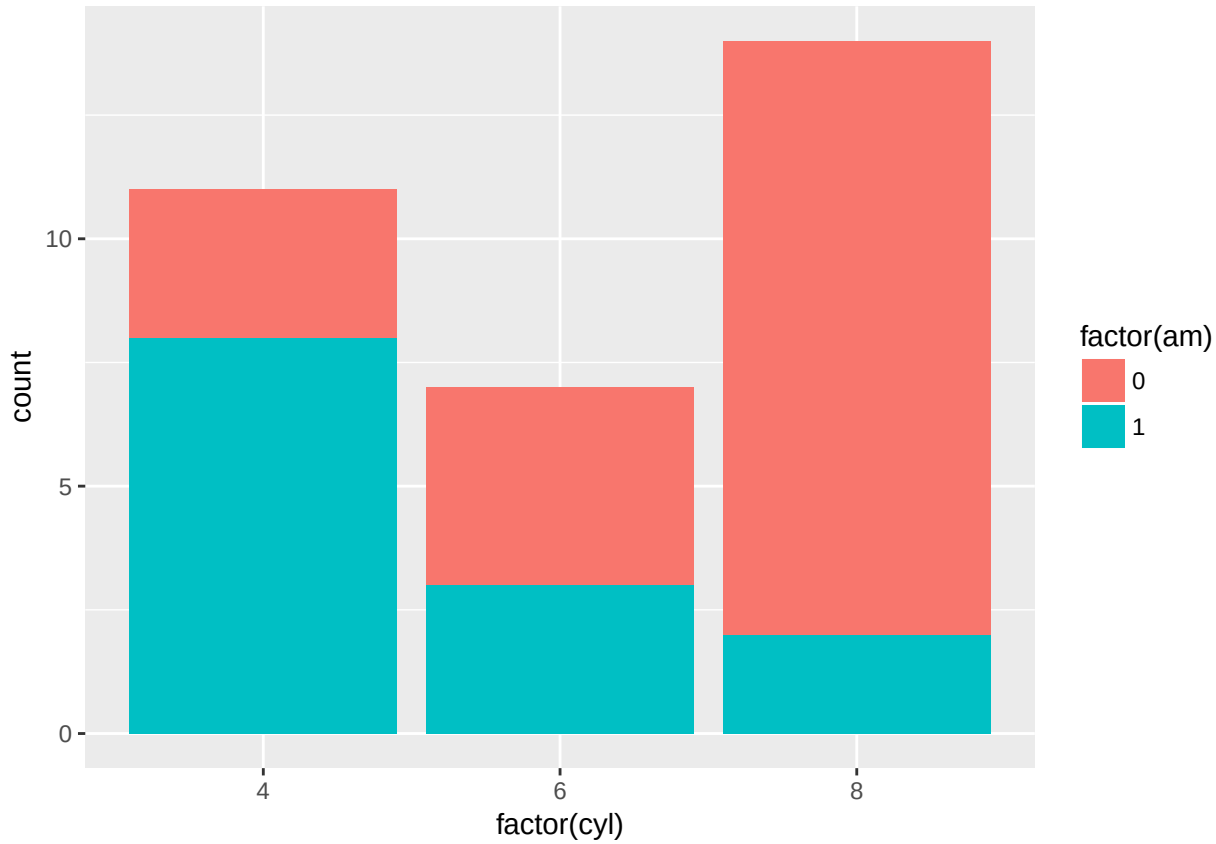
The ggplot2 base layers (data and aesthetics) have already been coded; they're stored in a variable cyl.am. It looks like this:

```
cyl.am <- ggplot(mtcars, aes(x = factor(cyl), fill = factor(am)))
```

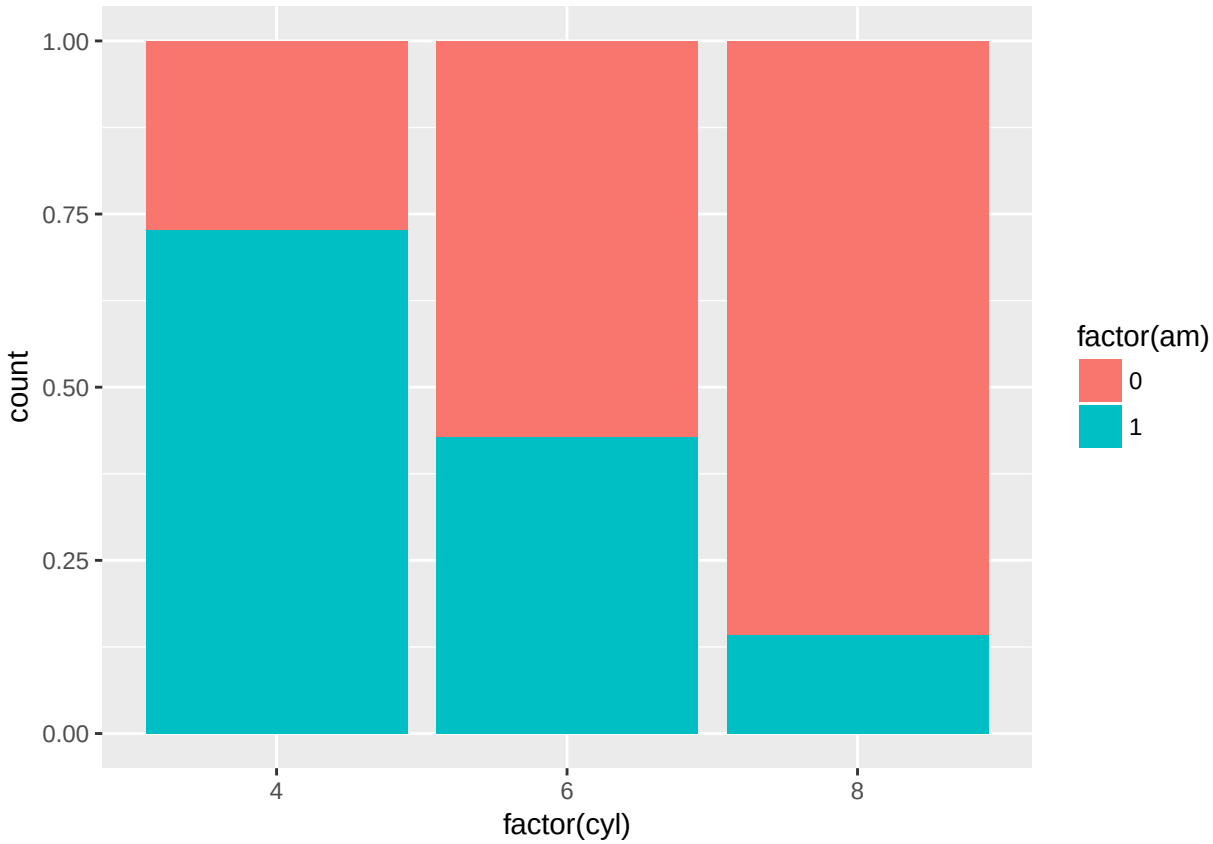
5.23.1 Instructions

Add a geom_bar() call to cyl.am. By default, the position will be set to "stack". Fill in the second ggplot command. Explicitly set position to "fill" inside geom_bar(). Fill in the third ggplot command. Set position to "dodge". The position = "dodge" version seems most appropriate. Finish off the fourth ggplot command by completing the three scale_ functions: scale_x_discrete() takes as its only argument the x-axis label: "Cylinders". scale_y_continuous() takes as its only argument the y-axis label: "Number". scale_fill_manual() fixes the legend. The first argument is the title of the legend: "Transmission". Next, values and labels are set to predefined values for you. These are the colors and the labels in the legend.

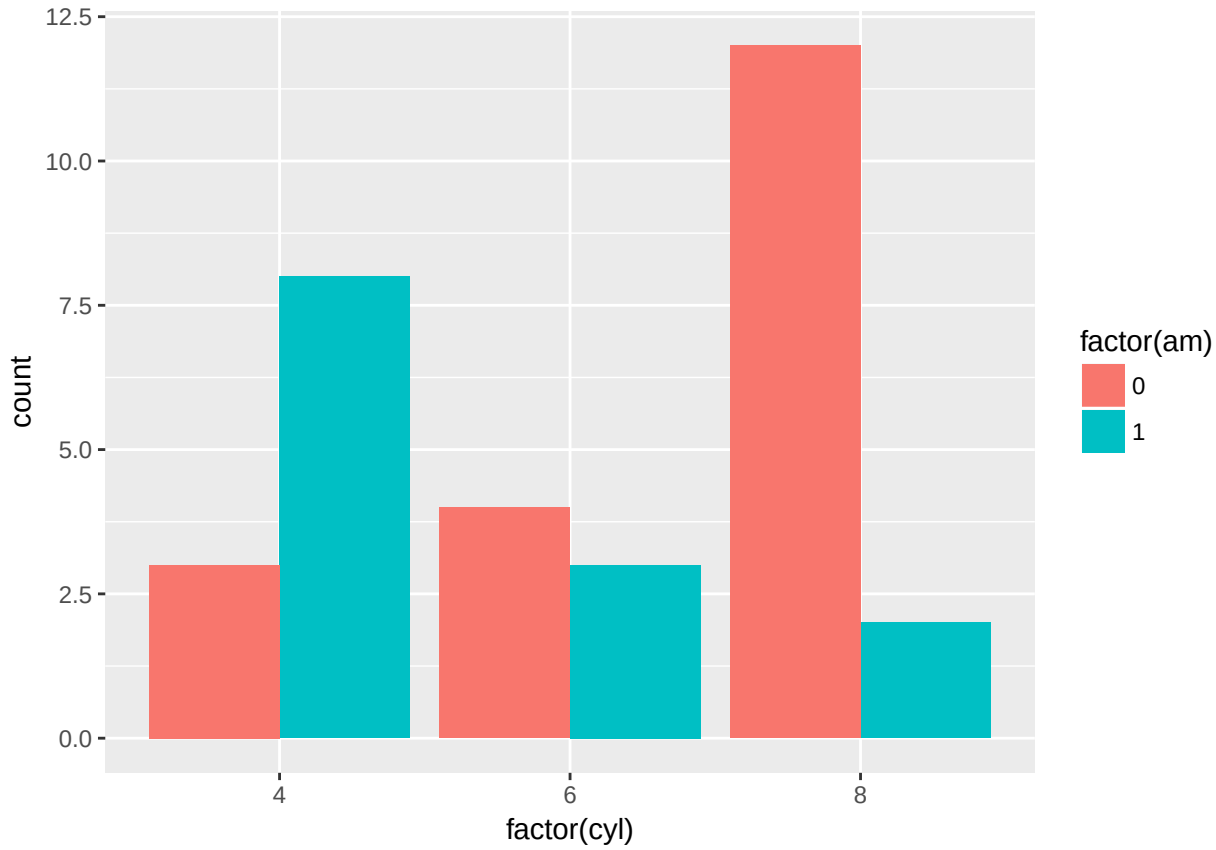
```
cyl.am <- ggplot(mtcars, aes(x = factor(cyl), fill = factor(am)))
# The base layer, cyl.am, is available for you
# Add geom (position = "stack" by default)
cyl.am + geom_bar(position="stack")
```



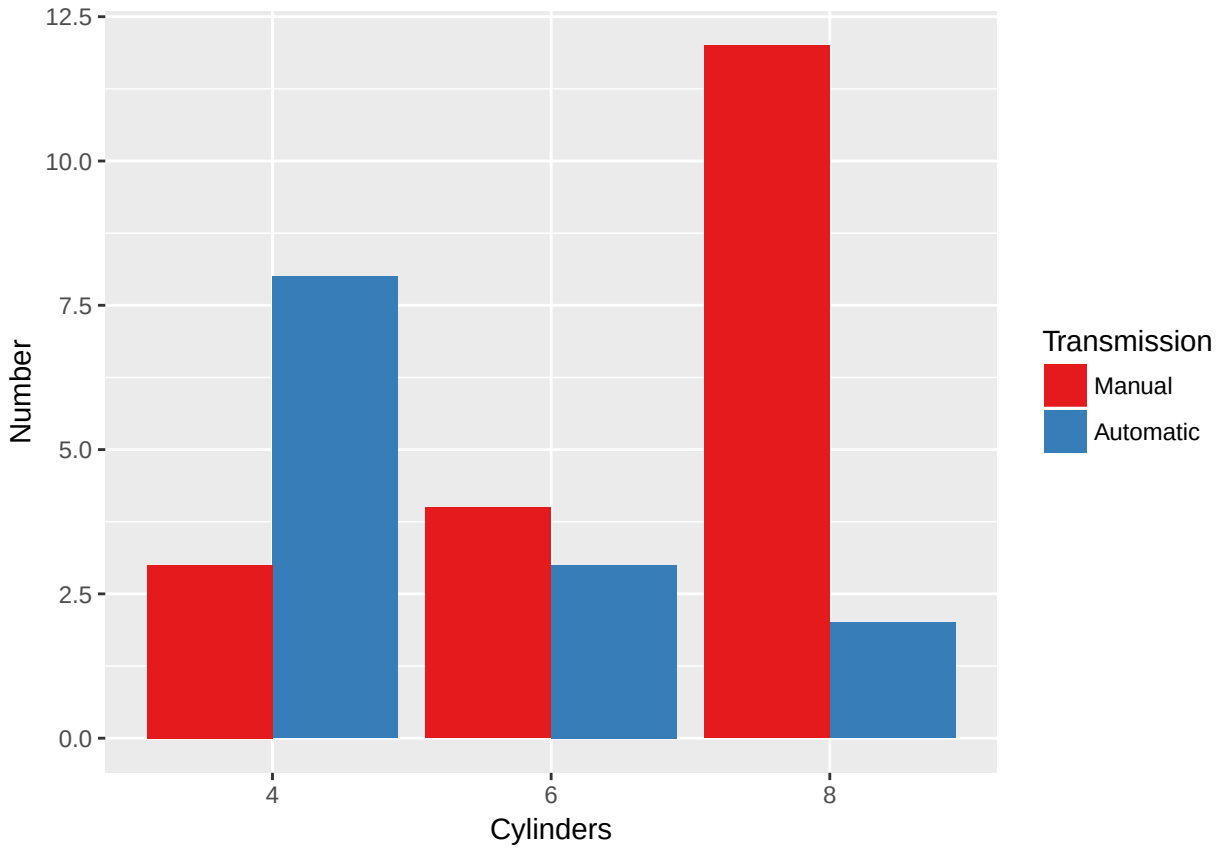

```
# Fill - show proportion
cyl.am +
  geom_bar(position = "fill")
```




```
# Dodging - principles of similarity and proximity
cyl.am +
  geom_bar(position = "dodge")
```




```
# Clean up the axes with scale_ functions
val = c("#E41A1C", "#377EB8")
lab = c("Manual", "Automatic")
cyl.am +
  geom_bar(position = "dodge") +
  scale_x_discrete("Cylinders") +
  scale_y_continuous("Number") +
  scale_fill_manual("Transmission",
                    values = val,
                    labels = lab)
```




```
# Premier positioning! Choosing the right position argument is an important part
# of making a good plot.
```

5.24 Setting a dummy aesthetic

In the last chapter you saw that all the visible aesthetics can serve as attributes and aesthetics, but I very conveniently left out x and y. That's because although you can make univariate plots (such as histograms, which you'll get to in the next chapter), a y-axis will always be provided, even if you didn't ask for it.

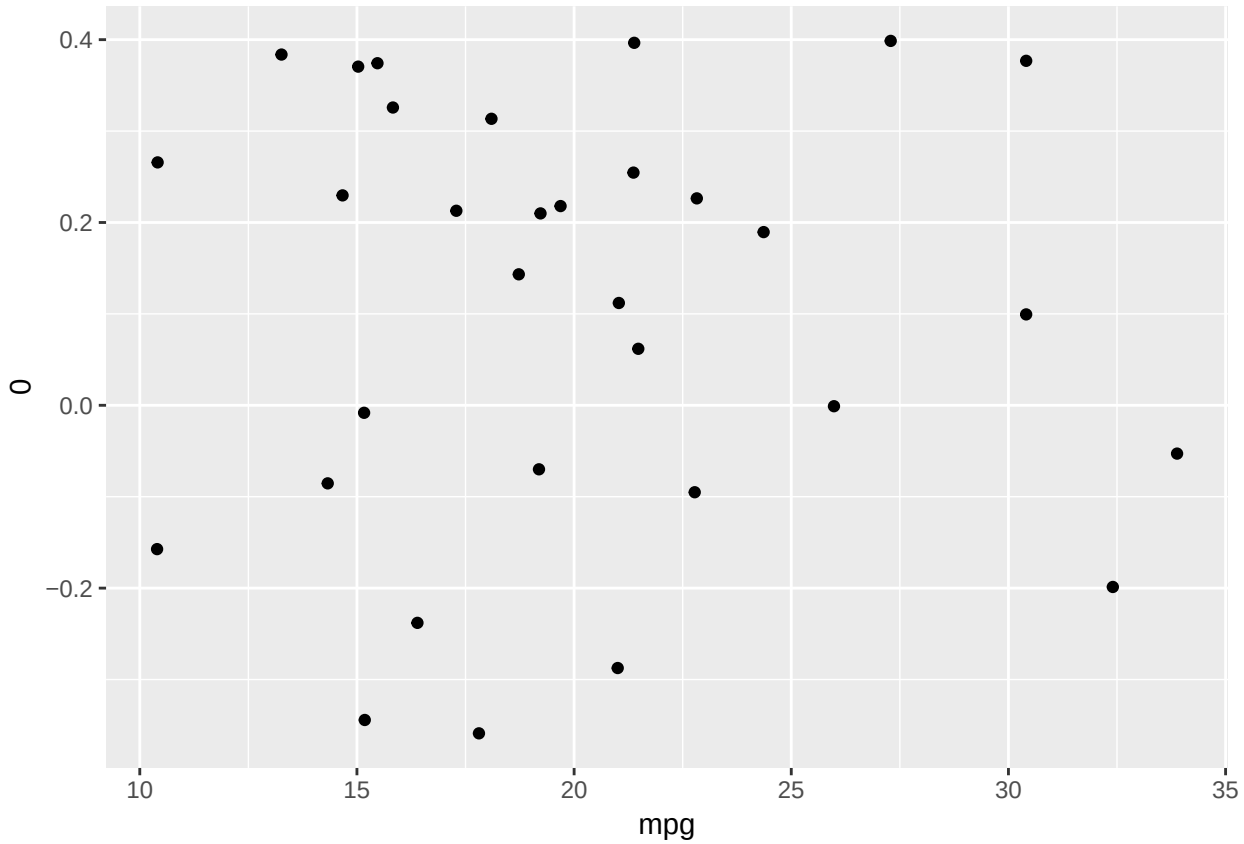
In the base package you can make univariate plots with stripchart() (shown in the viewer) directly and it will take care of a fake y axis for us. Since this is univariate data, there is no real y axis.

You can get the same thing in ggplot2, but it's a bit more cumbersome. The only reason you'd really want to do this is if you were making many plots and you wanted them to be in the same style, or you wanted to take advantage of an aesthetic mapping (e.g. colour).

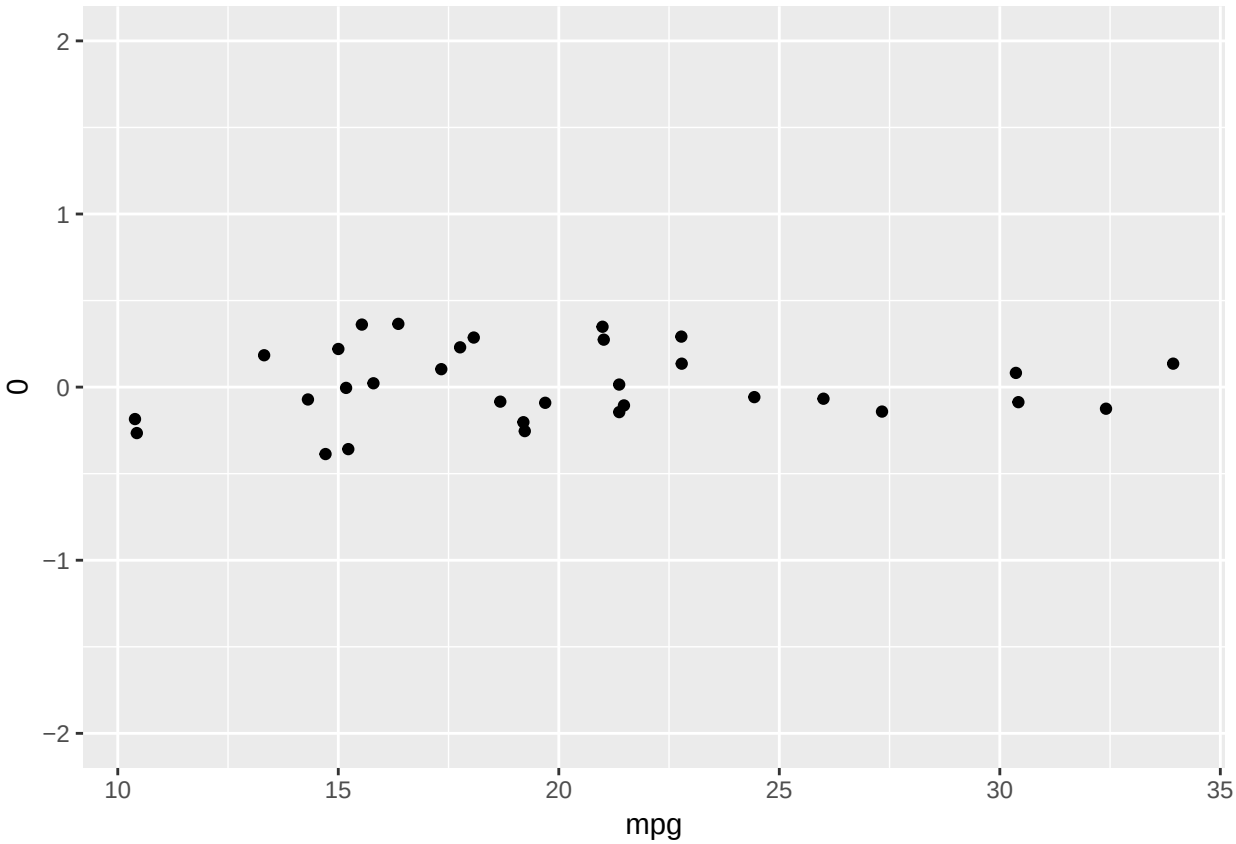
5.24.1 Instructions

Try to run ggplot(mtcars, aes(x = mpg)) + geom_point() in the console. x is only one of the two essential aesthetics for geom_point(), which is why you get an error message. 1 - To fix this, map a value, e.g. 0, instead of a variable, onto y. Use geom_jitter() to avoid having all the points on a horizontal line. 2 - To make everything look nicer, copy & paste the code for plot 1 and change the limits of the y axis using the appropriate scale_y_...() function. Set the limits argument to c(-2, 2).

```
# 1 - Create jittered plot of mtcars, mpg onto x, 0 onto y
ggplot(mtcars, aes(x = mpg, y = 0)) +
  geom_jitter()
```




```
# 2 - Add function to change y axis limits
ggplot(mtcars, aes(x = mpg, y = 0)) +
  geom_jitter() +
  scale_y_continuous(limits = c(-2,2))
```




```
# Great work! The best way to make your plot depends on a lot of different
# factors and sometimes ggplot2 might not be the best choice.
```

5.25 Overplotting 1 - Point shape and transparency

In the previous section you saw that there are lots of ways to use aesthetics. Perhaps too many, because although they are possible, they are not all recommended. Let's take a look at what works and what doesn't.

So far you've focused on scatter plots since they are intuitive, easily understood and very common. A major consideration in any scatter plot is dealing with overplotting. You'll encounter this topic again in the geometries layer, but you can already make some adjustments here.

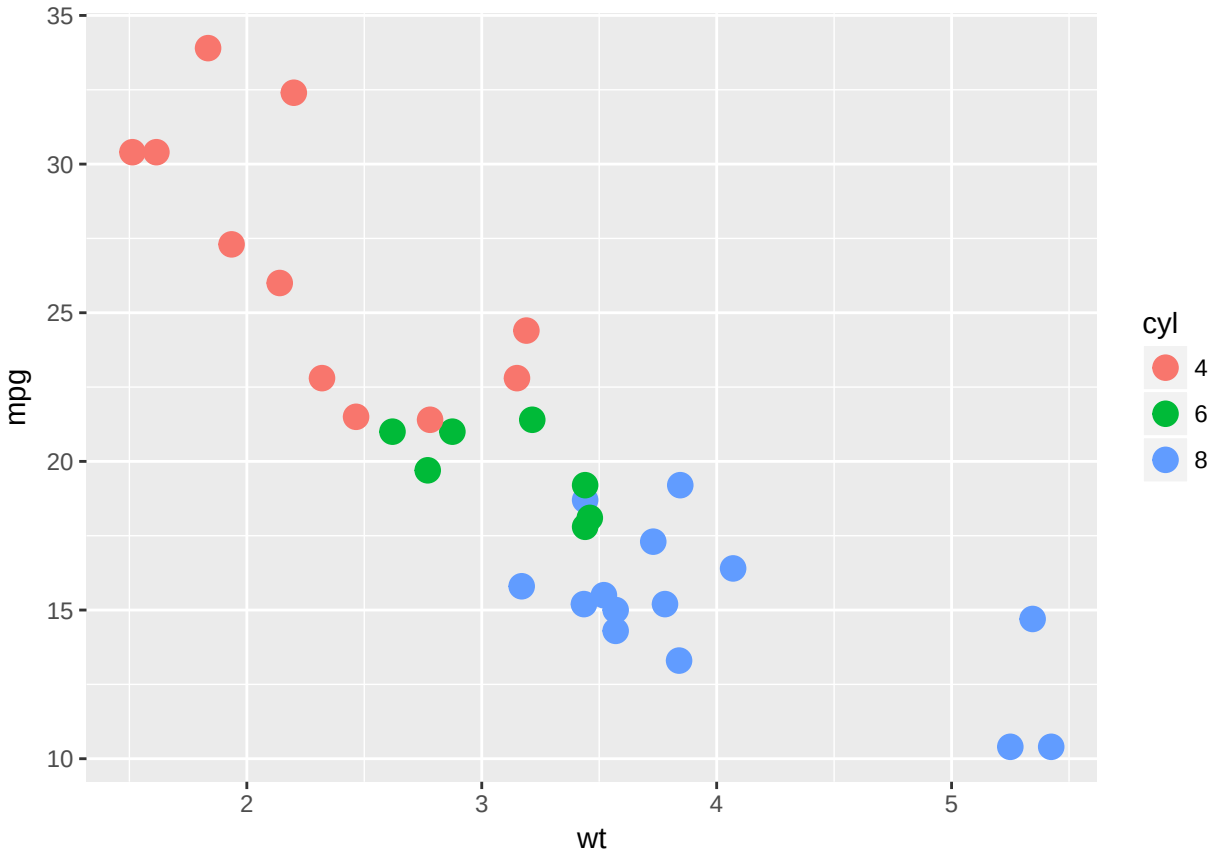
You'll have to deal with overplotting when you have:

Large datasets, Imprecise data and so points are not clearly separated on your plot (you saw this in the video with the iris dataset), Interval data (i.e. data appears at fixed values), or Aligned data values on a single axis. One very common technique that I'd recommend to always use when you have solid shapes it to use alpha blending (i.e. adding transparency). An alternative is to use hollow shapes. These are adjustments to make before even worrying about positioning. This addresses the first point as above, which you'll see again in the next exercise.

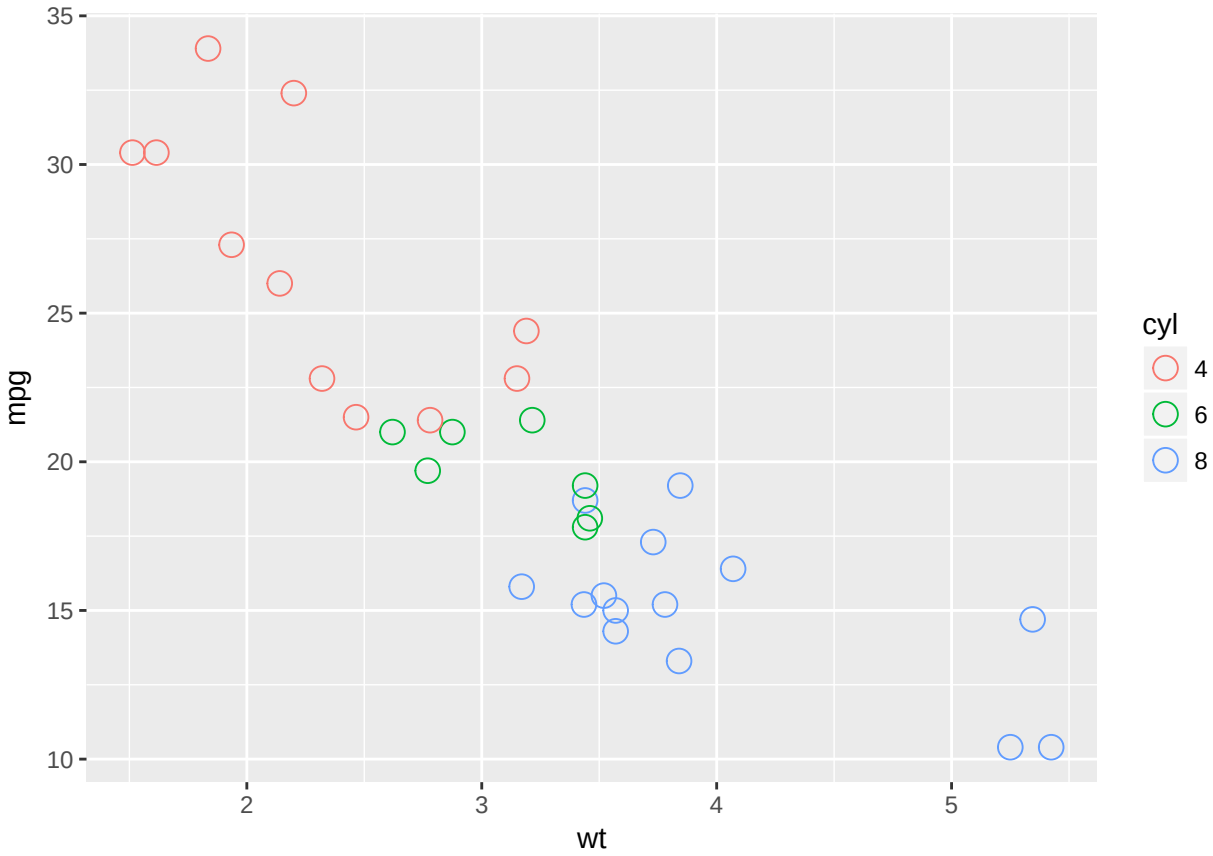
5.25.1 Instructions

Begin by making a basic scatter plot of mpg (y) vs. wt (x), map cyl to color and make the size = 4. cyl has already been converted to a factor variable for you. Modify the above plot to set shape to 1. This allows for hollow circles. Modify the first plot to set alpha to 0.6.

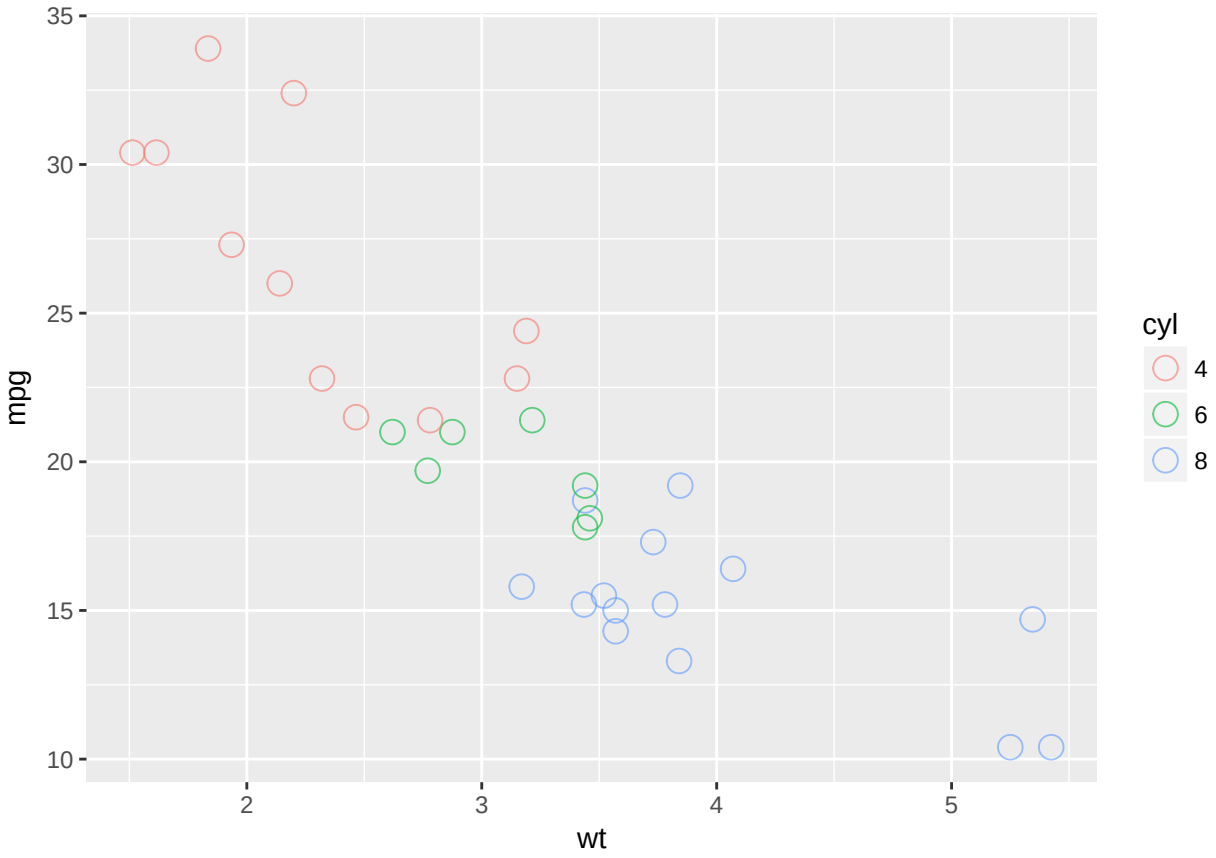
```
# Basic scatter plot: wt on x-axis and mpg on y-axis; map cyl to col
ggplot(mtcars, aes(x=wt, y=mpg, col=cyl)) +
  geom_point(size=4)
```




```
# Hollow circles - an improvement
ggplot(mtcars, aes(x=wt, y=mpg, col=cyl)) +
  geom_point(size=4, shape=1)
```




```
# Add transparency - very nice
ggplot(mtcars, aes(x=wt, y=mpg, col=cyl)) +
  geom_point(size=4, shape=1, alpha=0.6)
```




```
# Good job! By now you should understand why solid shapes are not really useful.
```

5.26 Overplotting 2 - alpha with large datasets

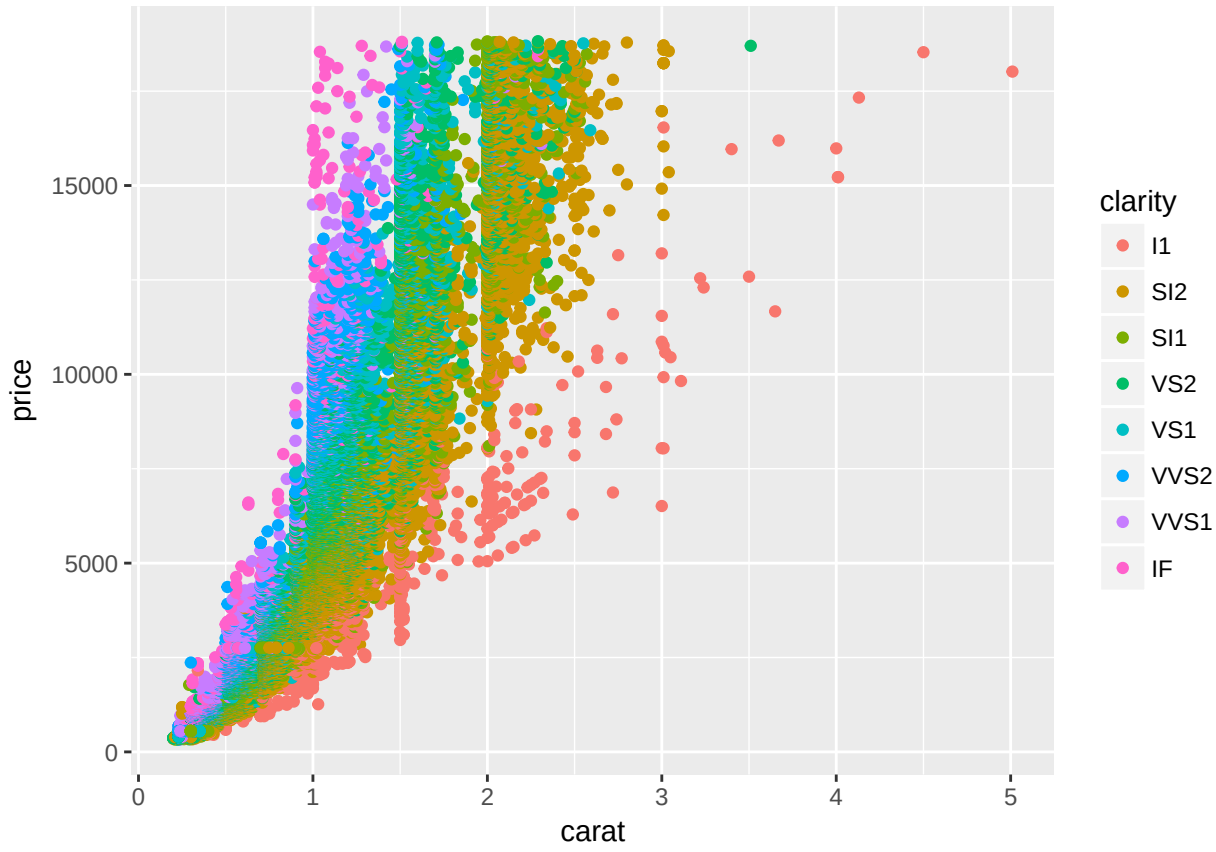
In a previous exercise we defined four situations in which you'd have to adjust for overplotting. You'll consider the last two here with the diamonds dataset:

1. Large datasets.
2. Aligned data values on a single axis

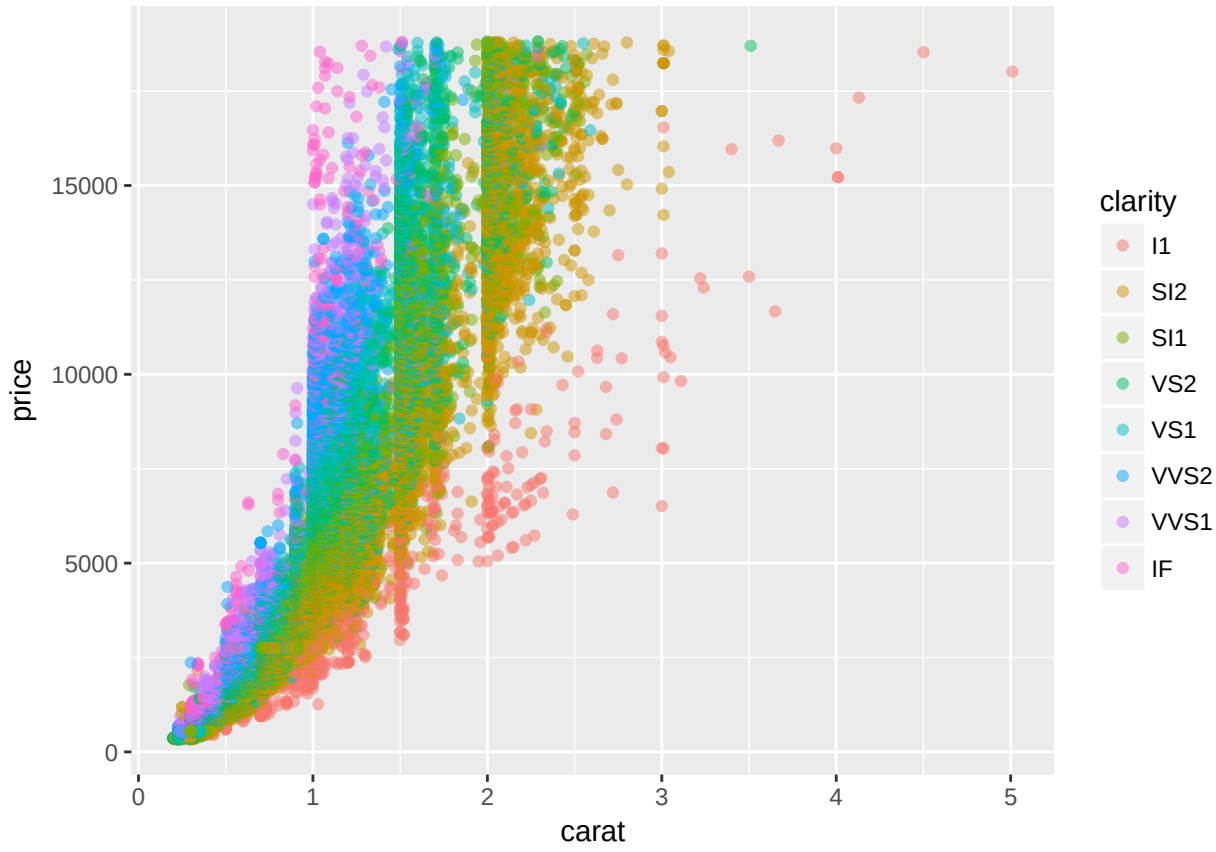
5.26.1 Instructions

The diamonds data frame is available in the ggplot2() package. Begin by making a basic scatter plot of price (y) vs. carat (x) and map clarity onto color. Copy the above functions and set the alpha to 0.5. This is a good start to dealing with the large dataset. Align all the diamonds within a clarity class, by plotting carat (y) vs. clarity (x). Map price onto color. alpha should still be 0.5. In the previous plot, all the individual values line up on a single axis within each clarity category, so you have not overcome overplotting. Modify the above plot to use the position = "jitter" inside geom_point().

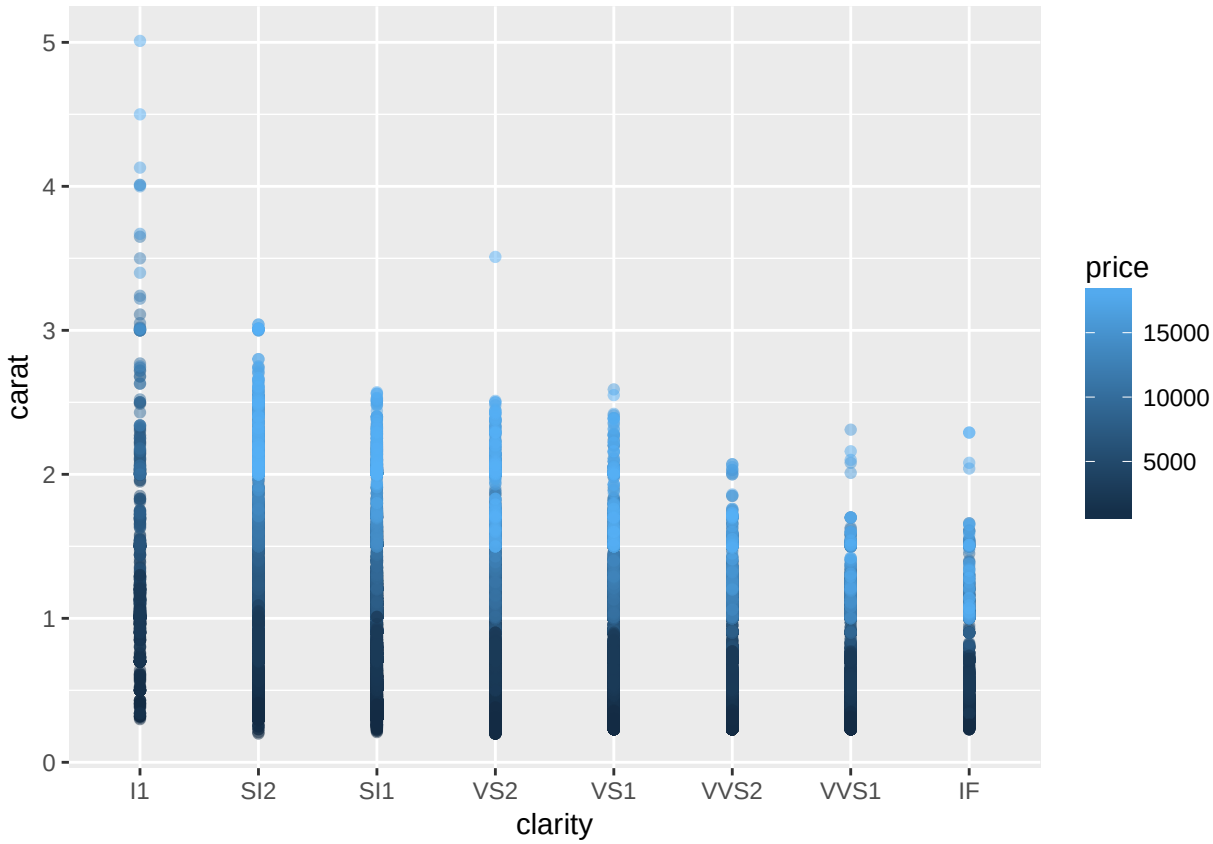
```
# Scatter plot: carat (x), price (y), clarity (color)
ggplot(diamonds, aes(x=carat, y=price, col=clarity)) +
  geom_point()
```



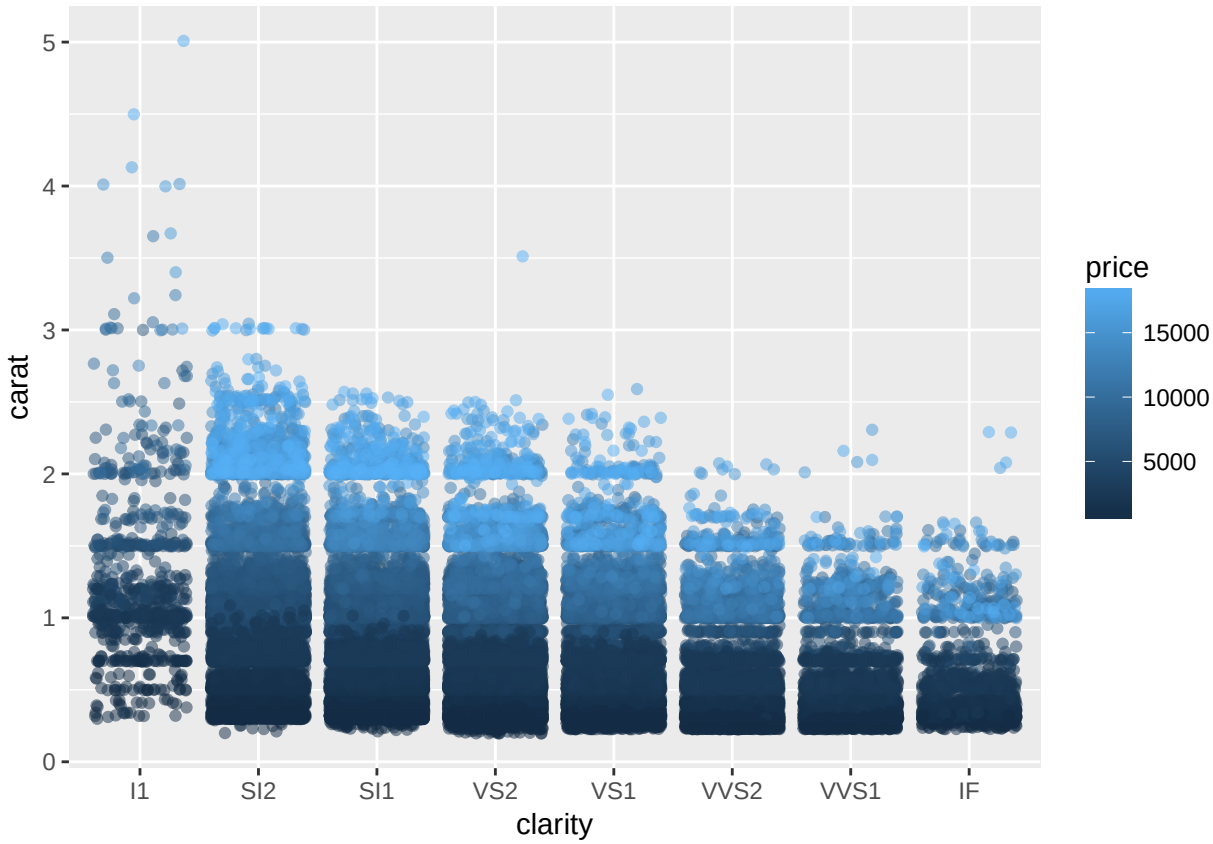

```
# Adjust for overplotting
ggplot(diamonds, aes(x=carat, y=price, col=clarity)) +
  geom_point(alpha=0.5)
```




```
# Scatter plot: clarity (x), carat (y), price (color)
ggplot(diamonds, aes(x=clarity, y=carat, col=price)) +
  geom_point(alpha=0.5)
```




```
# Dot plot with jittering
ggplot(diamonds, aes(x=clarity, y=carat, col=price)) +
  geom_point(alpha=0.5, position="jitter")
```




```
# Tantalizing transparency! These are some simple ways of dealing with large
# datasets, but you'll encounter more ideas in the geom chapter and in the
# advanced course on ggplot2 when you look at atypical geoms.

# You have finished the chapter "Aesthetics"!
```