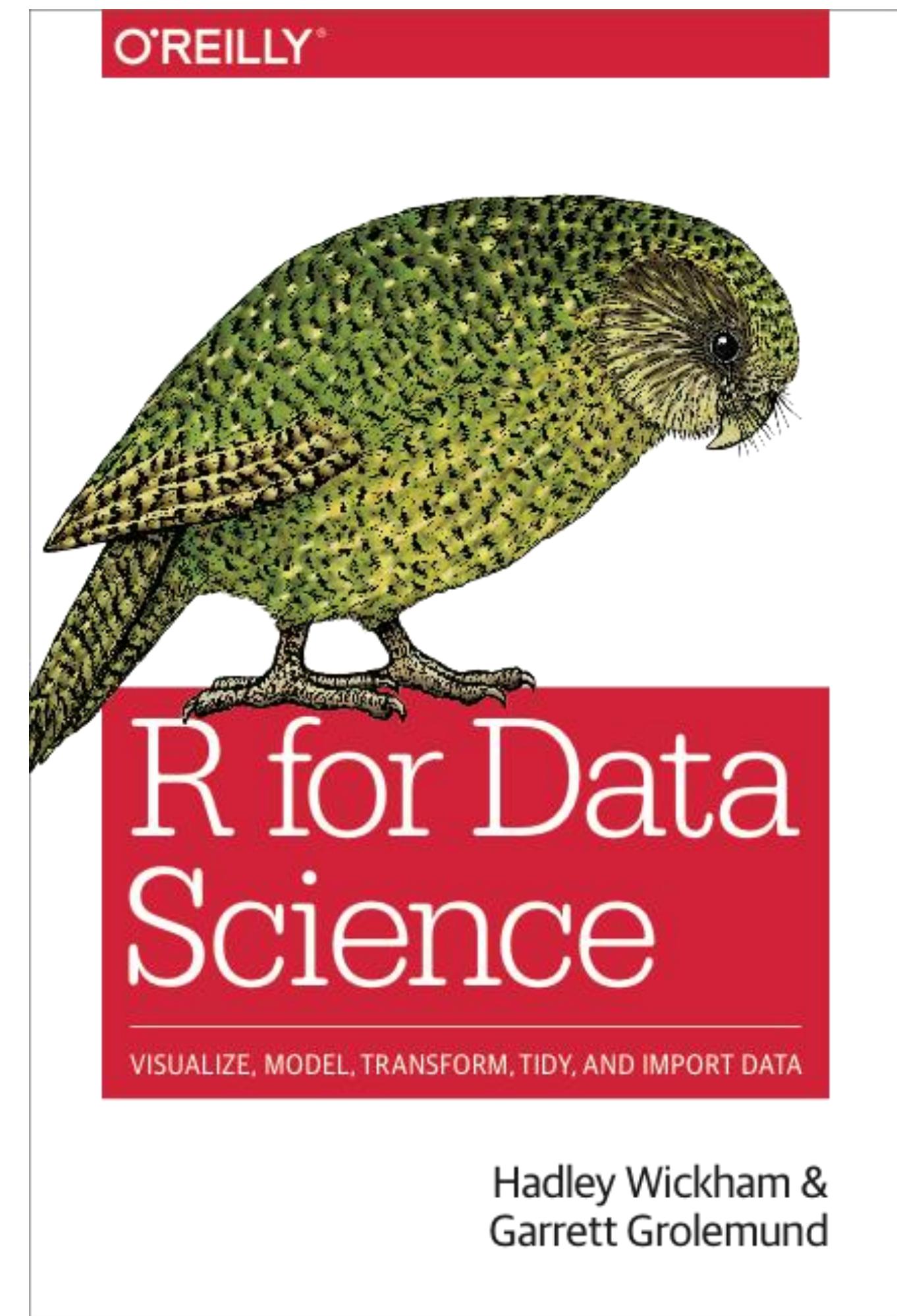


Transforming Data & Data Visualization

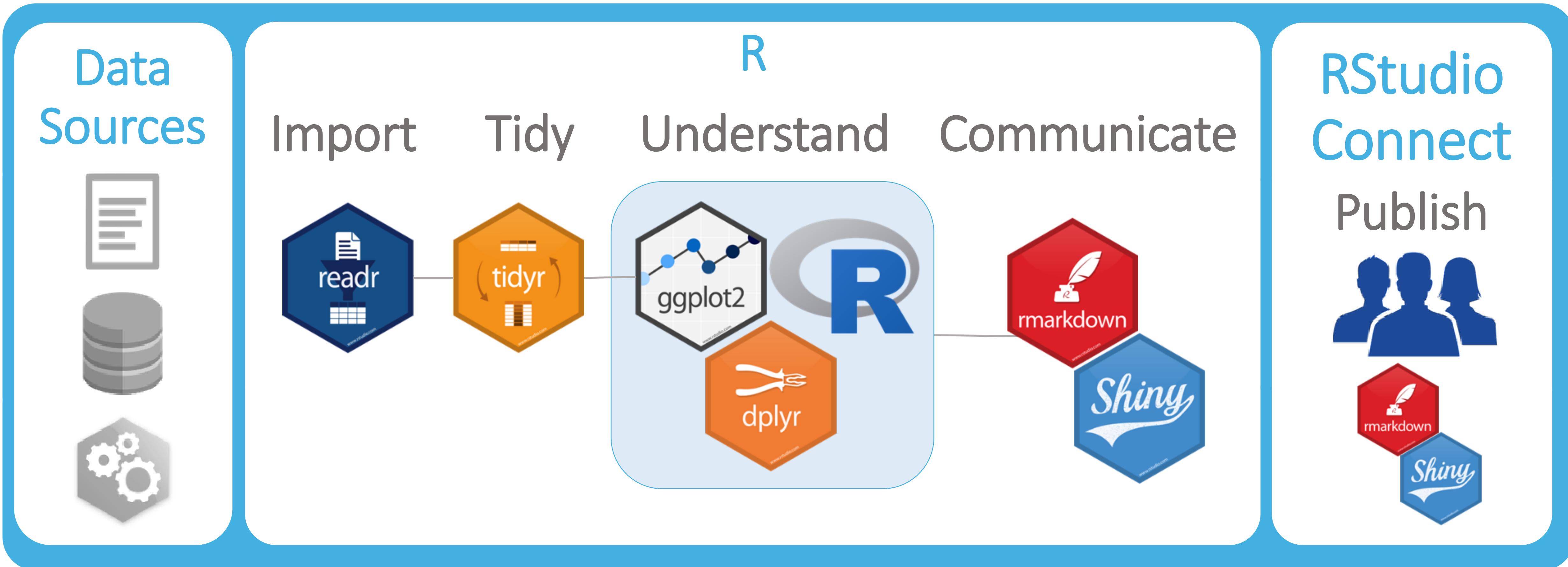


Art by Lou Pimentel



<https://github.com/rstudio/RStartHere>

R for Data Science Toolchain



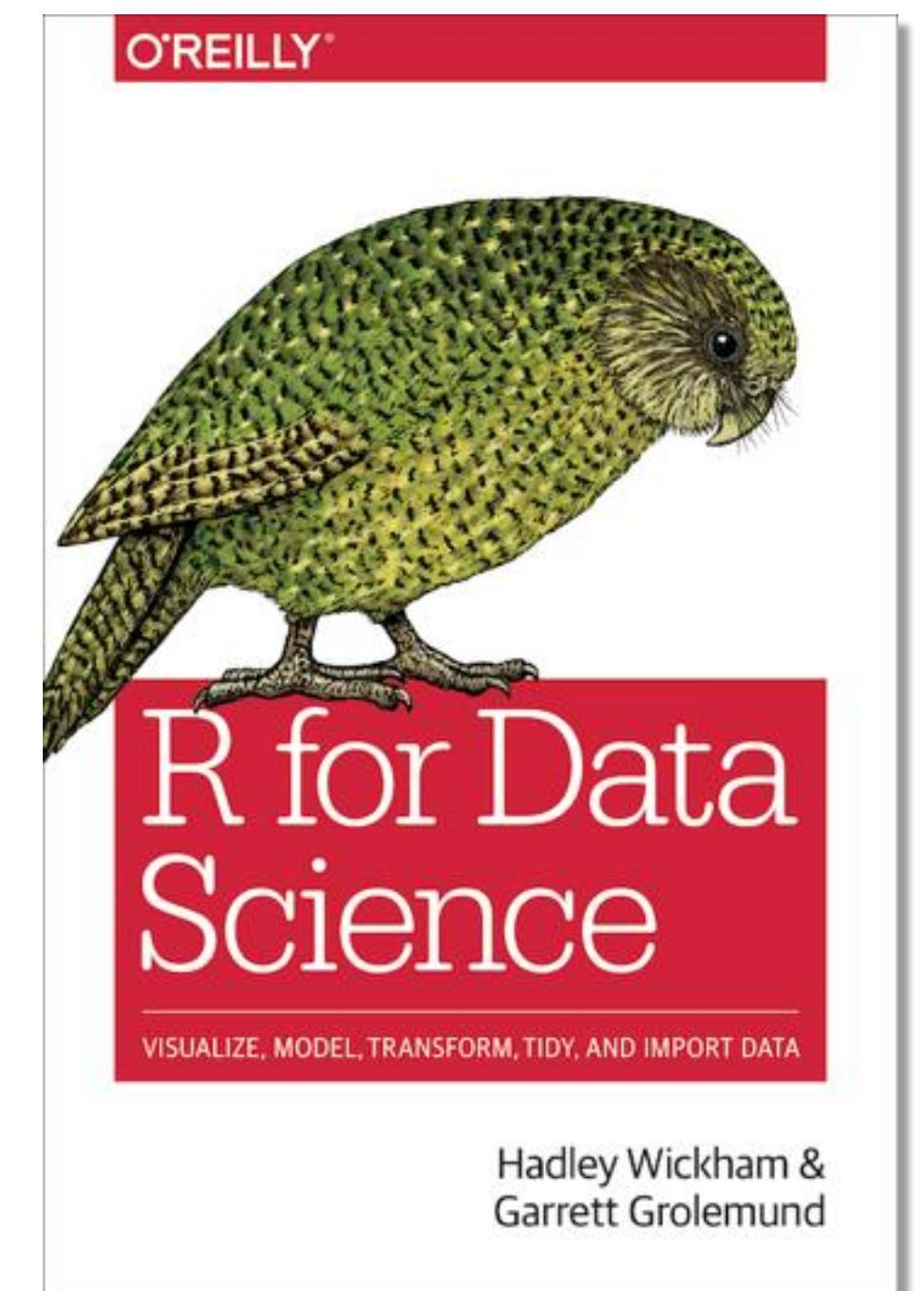
Tidyverse

A collection of R packages that share common philosophies and are designed to work together.

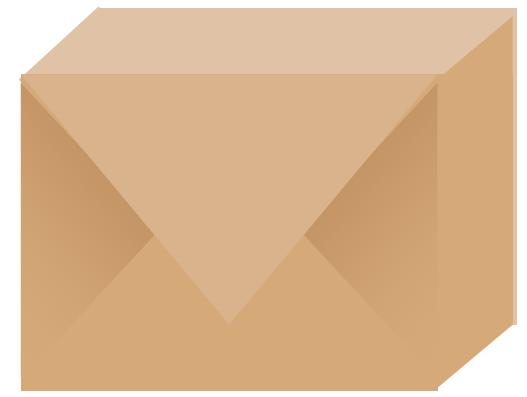


tidyverse.org

```
install.packages("tidyverse")
library(tidyverse)
```



tidyverse



An R package that serves as a short cut for installing and loading the components of the tidyverse.

```
install.packages("tidyverse")
```

does the equivalent of

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
install.packages("purrr")
install.packages("tibble")
install.packages("hms")
install.packages("stringr")
install.packages("lubridate")
install.packages("forcats")
install.packages("DBI")
install.packages("haven")
install.packages("httr")
install.packages("jsonlite")
install.packages("readxl")
install.packages("rvest")
install.packages("xml2")
install.packages("modelr")
install.packages("broom")
```

```
install.packages("tidyverse")
```

does the equivalent of

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
install.packages("purrr")
install.packages("tibble")
install.packages("hms")
install.packages("stringr")
install.packages("lubridate")
install.packages("forcats")
install.packages("DBI")
install.packages("haven")
install.packages("httr")
install.packages("jsonlite")
install.packages("readxl")
install.packages("rvest")
install.packages("xml2")
install.packages("modelr")
install.packages("broom")
```

```
library("tidyverse")
```

does the equivalent of

```
library("ggplot2")
library("dplyr")
library("tidyr")
library("readr")
library("purrr")
library("tibble")
```

Notebooks

The screenshot shows the RStudio Source Editor with an R Markdown file named "nb-demo.Rmd". The code chunk at line 10 contains the command `summary(iris)`, which is executed and its output is displayed in a code evaluation panel below the code. The output shows statistical summaries for Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species. The code chunk at line 16 contains the command `qplot(Sepal.Length, Petal.Length, data = iris, color = Species, size = Petal.Width)`, which is run and its output is displayed in a plot evaluation panel below the code. The plot is a scatter plot of Sepal.Length (x-axis) vs Petal.Length (y-axis). Data points are colored by Species (setosa in red, versicolor in green, virginica in blue) and sized by Petal.Width. A legend on the right side of the plot panel shows five sizes corresponding to Petal.Width values of 0.5, 1.0, 1.5, 2.0, and 2.5.

```
9
10 ````{r}
11 summary(iris)
12 ````

Sepal.Length Sepal.Width Petal.Length Petal.Width Species
Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100 setosa :50
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 versicolor:50
Median :5.800 Median :3.000 Median :4.350 Median :1.300 virginica :50
Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500

13
14 ````{r}
15 library(ggplot2)
16 qplot(Sepal.Length, Petal.Length, data = iris, color = Species, size =
Petal.Width)
17 ````
```

Combine in a single document:

- Narrative
- Code
- Output

Then Render to HTML

tibbles



Tidy data

country	year	cases	population
Afghanistan	2009	745	3071
Afghanistan	2009	766	30560
Bolivia	2009	2727	17262
Brazil	2009	30100	171518
China	2009	214258	127232372
China	2009	21500	12802633

A data set is tidy iff:

1. It is a data frame (e.g. tibble)
2. Each **variable** is in its own **column**
3. Each **case** is in its own **row**

Standardized framework allows your data to fit nicely into the APIs provided by base R and other key packages

Examining data frames in base R can be a challenge...

```
dim(iris)  
str(iris)  
iris
```

Coerce a data frame to a tibble, now it's pretty!...

```
tibble_iris <- tbl_df(iris)  
tibble_iris
```



Or

```
tibble(treatment = sample(letters, 10),  
       expression_value = rnorm(10, 100, 10))
```

tibbles

A type of data frame common throughout tidyverse packages.

Tibbles enhance data frames in three ways:

1. Subsetting - [always returns a new tibble, [[and \$ always return a new vector
2. No partial matching - You must use full column names when subsetting
3. Display - When you print a tibble, R provides a concise view of the data that fits on one screen





tibble

A package with several helper functions for tibbles:

- `as_tibble()` - convert a data frame to a tibble
- `as.data.frame()` - convert a tibble to a data frame
- `tribble()` - make a tibble (transversed)

```
tribble(  
  ~x, ~y,  
  1, "a",  
  2, "b",  
  3, "c")
```

x	y
1	a
2	b
3	c



babynames



R package

Names of male and female babies born in the US from 1880 to 2008. 1.8M rows.

```
# install.packages("babynames")
library(babynames)
```



View(babynames)

	year	sex	name	n	prop
1	1880	F	Mary	7065	0.0723835869
2	1880	F	Anna	2604	0.0266789611
3	1880	F	Emma	2003	0.0205214897
4	1880	F	Elizabeth	1939	0.0198657856
5	1880	F	Minnie	1746	0.0178884278
6	1880	F	Margaret	1578	0.0161672045
7	1880	F	Ida	1472	0.0150811946
8	1880	F	Alice	1414	0.0144869628
9	1880	F	Bertha	1320	0.0135238973
10	1880	F	Sarah	1288	0.0131960453
11	1880	F	Annie	1258	0.0128886840

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the **tibble**. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a **tibble**, R provides a concise view of the data that fits on one screen.
- **Subsetting** - [always returns a new **tibble**, [[and \$ always return a vector.
- **No partial matching** - You must use full column names when subsetting

```
# A tibble: 234 x 6
  manufacturer model dispel
  <chr>        <chr> <dbl>
1 audi         a4     1.8
2 audi         a4     1.8
3 audi         a4     2.0
4 audi         a4     2.0
5 audi         a4     2.0
6 audi         a4     2.0
7 audi         a4     2.0
8 audi         a4 quattro 1.8
9 audi         a4 quattro 1.8
10 audi        a4 quattro 1.8
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

tibble display

```
156 1999 6 audi(4)
157 1999 6 audi(4)
158 1999 6 audi(4)
159 2000 8 audi(4)
160 1999 4 manual(5)
161 1999 4 audi(4)
162 2000 4 manual(5)
163 2000 4 manual(5)
164 2000 4 audi(4)
165 2000 4 audi(4)
166 1999 4 audi(4)
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

A large table to display

data frame display

- Control the default appearance with options:
`options(tibble.print_max = n,
tibble.print_min = m, tibble.width = Inf)`
- View entire data set with `View(x, title)` or
`glimpse(x, width = NULL, ...)`
- Revert to data frame with `as.data.frame()`
(required for some older packages)

Construct a tibble in two ways

`tibble(...)`

Construct by columns.
`tibble(x = 1:3,
y = c("a", "b", "c"))`

Both make this tibble

`tribble(...)`

Construct by rows.
`tribble(~x, ~y,
1, "a",
2, "b",
3, "c")`

A tibble: 3 x 2
x y
<int> <dbl>
1 1 a
2 2 b
3 3 c

`as_tibble(x, ...)` Convert data frame to tibble.

`enframe(x, name = "name", value = "value")`
Converts named vector to a tibble with a names column and a values column.

`is_tibble(x)` Test whether x is a tibble.

tibbles

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the **tibble**. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a **tibble**, R provides a concise view of the data that fits on one screen.
- **Subsetting** - [always returns a new **tibble**, [[and \$ always return a vector.
- **No partial matching** - You must use full column names when subsetting

```
# A tibble: 234 x 6
  manufacturer model dispel
  <chr>        <chr> <dbl>
1 audi         a4     1.8
2 audi         a4     1.8
3 audi         a4     2.0
4 audi         a4     2.0
5 audi         a4     2.0
6 audi         a4     2.0
7 audi         a4     2.0
8 audi         a4 quattro 1.8
9 audi         a4 quattro 1.8
10 audi        a4 quattro 1.8
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

A large table to display

```
156 1999 6 audi(4)
157 1999 6 audi(4)
158 1999 6 audi(4)
159 2000 8 audi(4)
160 1999 4 manual(5)
161 1999 4 audi(4)
162 2000 4 manual(5)
163 2000 4 manual(5)
164 2000 4 audi(4)
165 2000 4 audi(4)
166 1999 4 audi(4)
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

data frame display

• Control the default appearance with options:
`options(tibble.print_max = n,
tibble.print_min = m, tibble.width = Inf)`

• View entire data set with `View(x, title)` or
`glimpse(x, width = NULL, ...)`

• Revert to data frame with `as.data.frame()`
(required for some older packages)

Construct a tibble in two ways

`tibble(...)`
Construct by columns.
`tibble(x = 1:3,
y = c("a", "b", "c"))`

Both make this tibble

`tribble(...)`
Construct by rows.
`tribble(~x, ~y,
1, "a",
2, "b",
3, "c")`

A tibble: 3 x 2
x y
<int> <dbl>
1 1 a
2 2 b
3 3 c

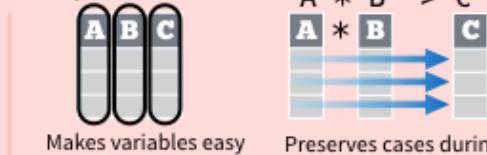
`as_tibble(x, ...)` Convert data frame to tibble.

`enframe(x, name = "name", value = "value")`
Converts named vector to a tibble with a names column and a values column.

`is_tibble(x)` Test whether x is a tibble.

Tidy Data with tidyverse

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages. A table is tidy if:



Each variable is in its own column

Each observation, or case, is in its own row

Makes variables easy to access as vectors

Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use `gather()` and `spread()` to reorganize the values of a table into a new layout. Each uses the idea of a key column: value column pair.

`gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)`

Gather moves column names into a key column, gathering the column values into a single value column.

`table4a`

country	1999	2000	rate
A	0.7K	2K	19M
B	37K	80K	20M
B	2000	80K	174M
C	212K	172M	1T
C	2000	213K	1T

`gather(table4a, `1999`, `2000`, key = "year", value = "cases")`

`spread(table2, type, count)`

`table2`

country	1999	type	cases	pop
A	1999	0.7K	19M	
A	2000	2K	20M	
B	1999	37K	172M	
B	2000	80K	174M	
C	1999	212K	1T	
C	2000	213K	1T	

`key` `value`

`gather(table4a, `1999`, `2000`, key = "year", value = "cases")`

`spread(table2, type, count)`

`table3`

`country` `year` `rate`

A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

`separate_rows(table3, rate, into = c("cases", "pop"))`

`separate_rows(data, ..., sep = "[[:alnum:]]+", convert = FALSE)`

Separate each cell in a column to make several rows. Also `separate_rows_()`.

`table4`

`country` `year` `rate`

A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

`key` `value`

`separate_rows(table3, rate)`

`unite(data, col, ..., sep = "_", remove = TRUE)`

Collapse cells across several columns to make a single column.

`table5`

`country century year`

Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

`country century year`

`col = "year", sep = ""`

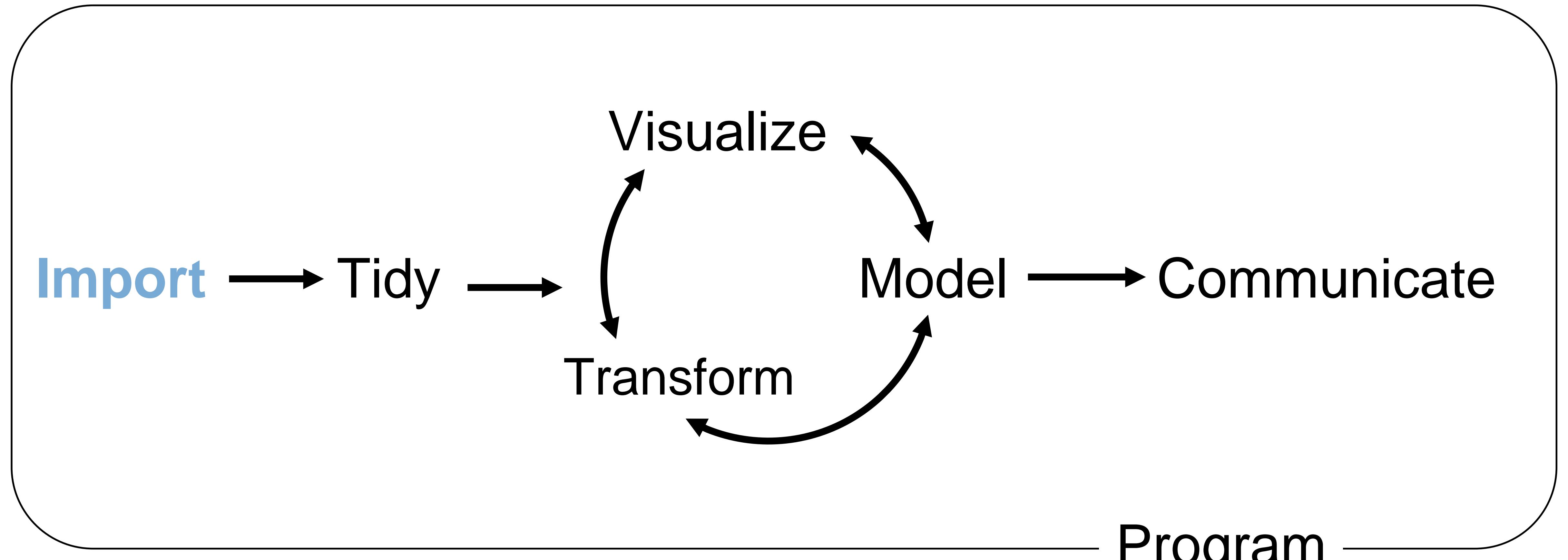
`unite(table5, century, year, col = "year", sep = "")`



Import Data with



(Applied) Data Science



Importing Data

R

readr



Simple, consistent functions for working with strings.

```
# install.packages("tidyverse")
library(tidyverse)
```



Compared to `read.table` and its derivatives, `readr` functions are:

1. ~ 10 times faster
2. Return tibbles
3. Have more intuitive defaults. No row names, no strings as factors.



readr functions

function	reads
read_csv()	Comma separated values
read_csv2()	Semi-comma separated values
read_delim()	General delimited files
read_fwf()	Fixed width files
read_log()	Apache log files
read_table()	Space separated
read_tsv()	Tab delimited values



`read_csv()`

`readr` functions share a common syntax

```
df <- read_csv("path/to/file.csv", ...)
```

object to save
output into

path from working
directory to file

dplyr



A package that transforms data.
dplyr implements a *grammar* for
transforming tabular data.

Data transformation toolbox



Power of dplyr - sparklyr

<http://spark.rstudio.com/dplyr.html>

<http://spark.rstudio.com/>

<http://spark.rstudio.com/examples-emr.html>

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data.
(Dasu and Johnson, 2003)



single table verbs

`filter()` - extract **cases**

`arrange()` - reorder **cases**

`group_by()` - group **cases**

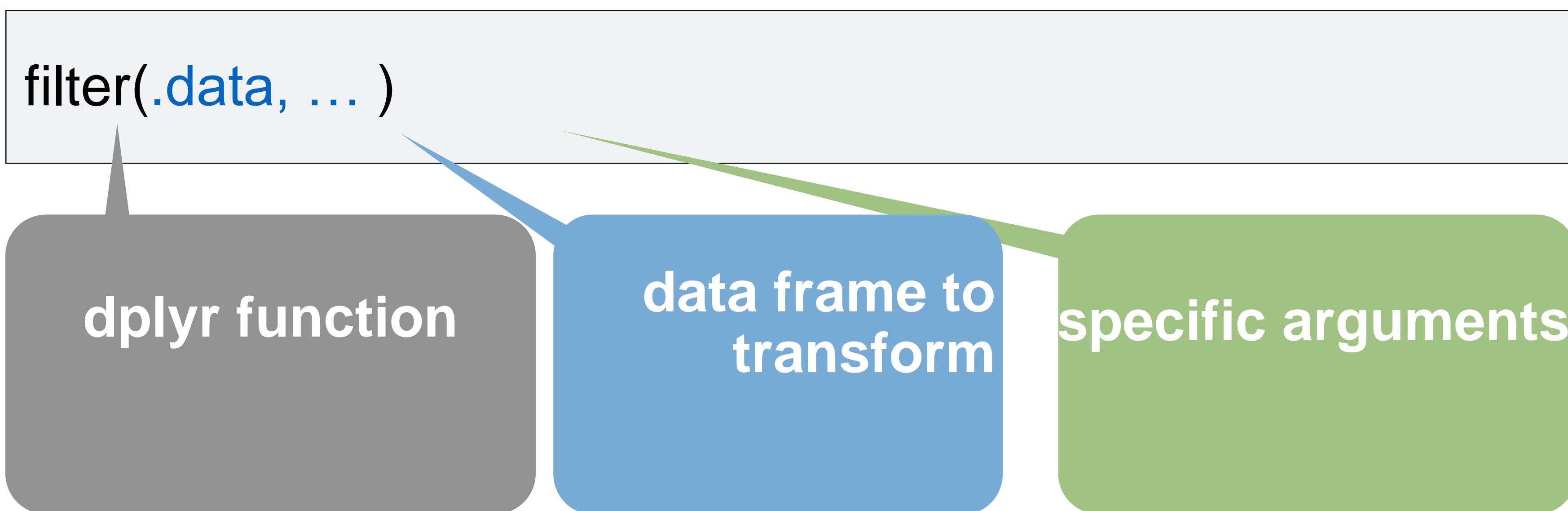
`select()` - extract **variables**

`mutate()` - create new **variables**

`summarise()` - summarise **variables** / create
cases

common syntax

Each function take a data frame / tibble as its first argument and returns a data frame / tibble.



two table verbs

`bind_rows()` - adds one table to another as new `cases`

`union()`, `intersect()`, `setdiff()` - set operations for `cases`

`semi_join()`, `anti_join()` - filters `cases` in one table against another

`bind_cols()` - adds one table to another as new `variables`

`left_join()`, `right_join()`, `full_join()`, `inner_join()` - mutates one table by matching values from another as new `variables`



Toy data

```
storms <- tribble(  
  ~storm, ~wind, ~pressure, ~date,  
  "Alberto", 110, 1007, "2000-08-12",  
  "Alex", 45, 1009, "1998-07-30",  
  "Allison", 65, 1005, "1995-06-04",  
  "Ana", 40, 1013, "1997-07-01",  
  "Arlene", 50, 1010, "1999-06-13",  
  "Arthur", 45, 1010, "1996-06-21"  
)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

filter()



filter()

Extract rows that meet logical criteria.

```
filter(.data, ...)
```

data frame to
transform

one or more logical tests (filter
returns each row for which the
test is TRUE)

filter()

Extract rows that meet logical criteria.

```
filter(storms, wind == 45)
```

storms			
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21

= sets
(returns nothing)
== tests if equal
(returns TRUE or FALSE)



filter()

Extract rows that meet logical criteria.

```
filter(storms, wind == 45)
```

storms			
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

storm	wind	pressure	date
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21



filter()

Extract rows that meet logical criteria.

```
filter(storms, wind == 45, pressure == 1009)
```

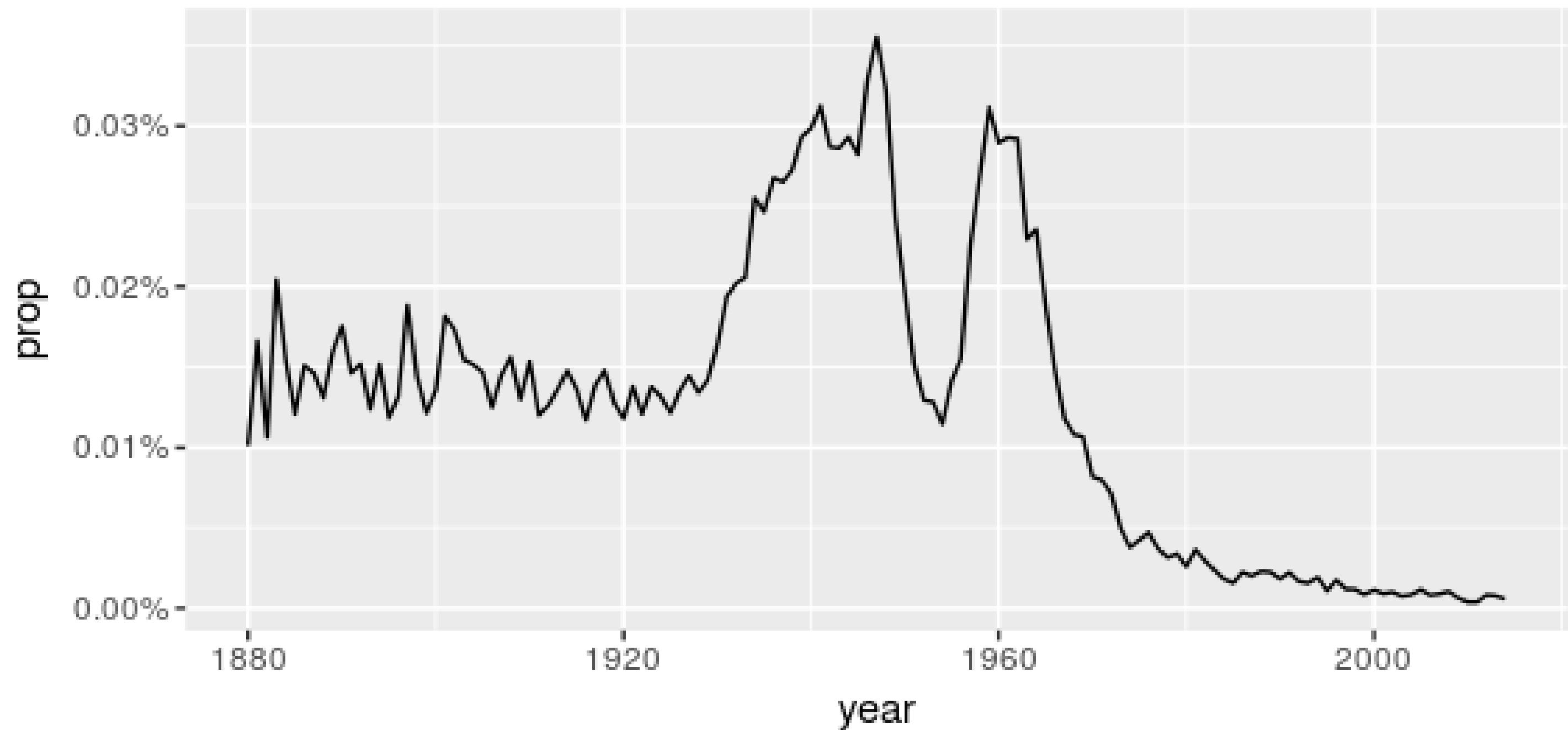
storms				
storm	wind	pressure	date	
Alberto	110	1007	2000-08-12	
Alex	45	1009	1998-07-30	
Allison	65	1005	1995-06-04	
Ana	40	1013	1997-07-01	
Arlene	50	1010	1999-06-13	
Arthur	45	1010	1996-06-21	

→

storm	wind	pressure	date
Alex	45	1009	1998-07-30



```
Phil <- filter(babynames, name == "Phil", sex == "M")
P <- ggplot(data = Phil, mapping = aes(year, prop)) +
  geom_line()
require(scales)
p + scale_y_continuous(labels = percent)
```



Logical tests

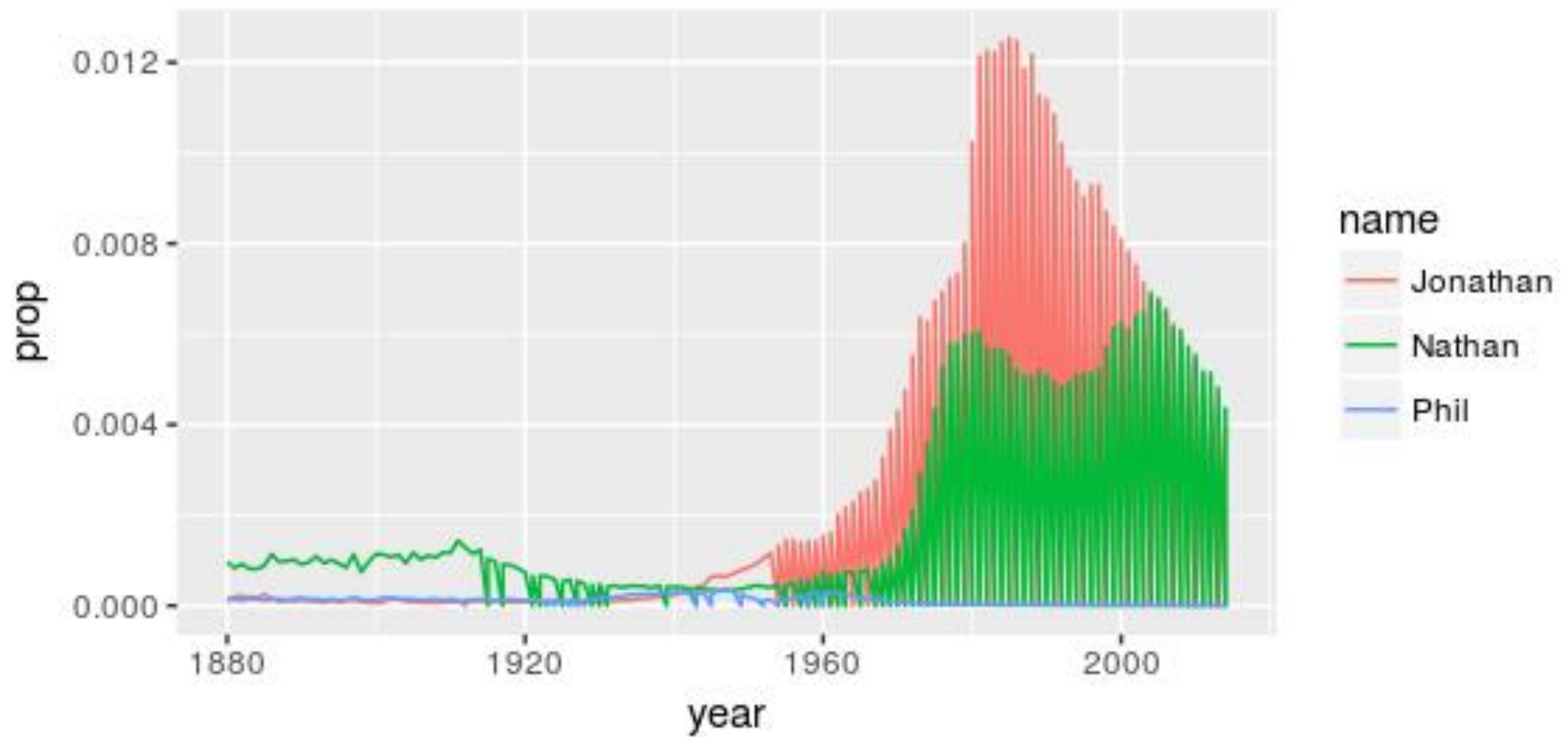
?Comparison

<	Less than
>	Greater than
==	Equal to
<=	Less than or equal to
=	Greater than or equal to
!=	Not equal to
%in%	Group membership
is.na()	Is NA
!is.na()	Is not NA

?base::Logic

&	and
	or
xor()	exactly or
!	not
any()	any true
all()	all true

```
gnames <- c("Nathan", "Jonathan", "Phil")
group_names <- filter(babynames, name %in% gnames)
ggplot(group_names, aes(year, prop, color = name)) +
  geom_line()
```



arrange()

R

arrange()

Order rows from smallest to largest values.

```
arrange(.data, ...)
```

data frame to transform

one or more columns to order by
(additional columns will be used as tie breakers)

arrange()

Order rows from smallest to largest values.

```
arrange(storms, wind)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12



desc()

Changes ordering to largest to smallest.

```
arrange(storms, desc(wind))
```

storms				
storm	wind	pressure	date	
Alberto	110	1007	2000-08-12	
Alex	45	1009	1998-07-30	
Allison	65	1005	1995-06-04	
Ana	40	1013	1997-07-01	
Arlene	50	1010	1999-06-13	
Arthur	45	1010	1996-06-21	

→

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21
Alex	45	1009	1998-07-30
Ana	40	1013	1997-07-01



select()

R

select()

Extract columns by name.

```
select(.data, ...)
```

data frame to transform

**name(s) of columns to extract
(or a select helper function)**

select()

Extract columns by name.

```
select(storms, storm, pressure)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

select() helpers

: - Select range of columns

```
select(storms, storm:pressure)
```

- - Select every column but

```
select(storms, -c(storm, pressure))
```

starts_with() - Select columns that start with...

```
select(storms, starts_with("w"))
```

ends_with() - Select columns that end with...

```
select(storms, ends_with("e"))
```



select() helpers

contains() - Select columns whose names contain...

```
select(storms, contains("d"))
```

matches() - Select columns whose names match regular expression

```
select(storms, matches("^.{4}$"))
```

one_of() - Select columns whose names are one of a set

```
select(storms, one_of(c("storm", "storms", "Storm")))
```

num_range() - Select columns named in prefix, number style

```
select(storms, num_range("x", 1:5))
```



mutate()

R

mutate()

Create new columns.

```
mutate(.data, ...)
```

data frame to transform

named argument
(that consists of the name of column to create set equal to an expression that creates it)

mutate()

Create new columns.

```
mutate(storm, ratio = pressure / wind)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date	ratio
Alberto	110	1007	2000-08-12	9.15
Alex	45	1009	1998-07-30	22.42
Allison	65	1005	1995-06-04	15.46
Ana	40	1013	1997-07-01	25.32
Arlene	50	1010	1999-06-13	20.20
Arthur	45	1010	1996-06-21	22.44



summarise()

R

summarise()

Compute table of summaries.

```
summarise(.data, ...)
```

**data frame to
transform**

named argument
that consists of the name of column to
create set equal to an expression that
creates it)



summarise()

Compute table of summaries.

```
summarise(storms, avg_wind = mean(wind), n = n())
```

storms					avg_wind	n
storm	wind	pressure	date			
Alberto	110	1007	2000-08-12		59.17	6
Alex	45	1009	1999-07-01			
Allison	65	1005	1999-07-01			
Ana	40	1013	1999-07-01			
Arlene	50	1010	1999-07-01			
Arthur	45	1010	1996-06-21			

Returns the number of cases
(rows) Very useful!



n() and n_distinct()

Two helper functions for summarise.

```
summarise(storms,  
  n = n(), # Number of cases / rows  
  n_wind = n_distinct(wind) # number of unique values  
)
```

storm	wind	pressure	date	n	n_wind
Alberto	110	1007	2000-08-12	6	5
Alex	45	1009	1998-07-30		
Allison	65	1005	1995-06-04		
Ana	40	1013	1997-07-01		
Arlene	50	1010	1999-06-13		
Arthur	45	1010	1996-06-21		



%>%



Little bunny Foo Foo
Went hopping through the forest
Scooping up the field mice
And bopping them on the head

```
# Every intermediate as a new object
foo_foo <- little_bunny()
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

```
# Overwrite the original object
foo_foo <- hop(foo_foo, through = forest)
foo_foo <- scoop(foo_foo, up = field_mice)
foo_foo <- bop(foo_foo, on = head)
```

```
# Nested function calls
bop(
  scoop(
    hop(foo_foo, through = forest),
    up = field_mice
  ),
  on = head
)
```

```
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mouse) %>%
  bop(on = head)
```

```
head(filter(select(iris, Species, Petal.Length),  
Species == "setosa"))
```

is clearer when written:

```
iris %>% select(Species, Petal.Length) %>%  
filter(Species == "setosa") %>% head()
```

```
mutate(babynames, percent = prop * 100)
```

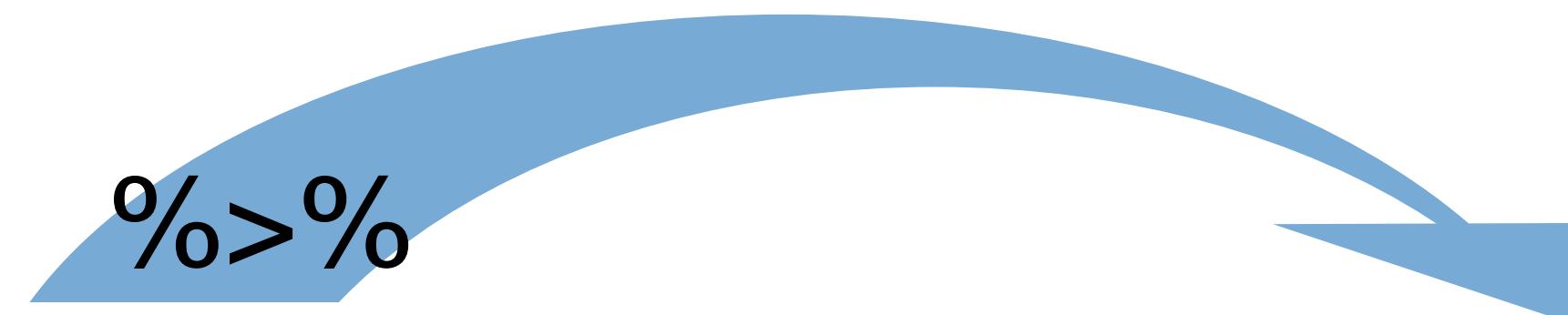
```
summarize(babynames, n = n_distinct(name))
```

```
phil <- filter(babynames, name == "Phil", sex == "M")
summarise(phil, min = min(prop), mean = mean(prop),
max = max(prop))
```

```
babynames2 <- mutate(babynames, percent = prop * 100)
babynames3 <- filter(babynames2, percent > 1)
summarise(babynames3, nn = n())
```

The pipe operator %>%

%>%



```
babynames     mutate(___, percent = prop * 100)
```

Passes result on left into first argument of function on right.
So, for example, these do the same thing. Try it.

```
mutate(babynames, percent = prop * 100)
```

```
babynames %>% mutate(percent = prop * 100)
```



```
Phil <- filter(babynames, name == "Phil", sex == "M")
summarise(Phil, min = min(prop), mean = mean(prop),
max = max(prop))
```

```
filter(babynames, name == "Phil", sex == "M") %>%
summarise(min = min(prop), mean = mean(prop),
max = max(prop))
```

```
phil <- filter(babynames, name == "Phil", sex == "M")
summarise(phil, min = min(prop), mean = mean(prop),
max = max(prop))
```

```
babynames %>%
filter(name == "Phil", sex == "M") %>%
summarise(min = min(prop), mean = mean(prop),
max = max(prop))
```

Shortcut to type %>%

Cmd + **Shift** + **M** (Mac)

Ctrl + **Shift** + **M** (Windows)



Grouping cases

group_by()

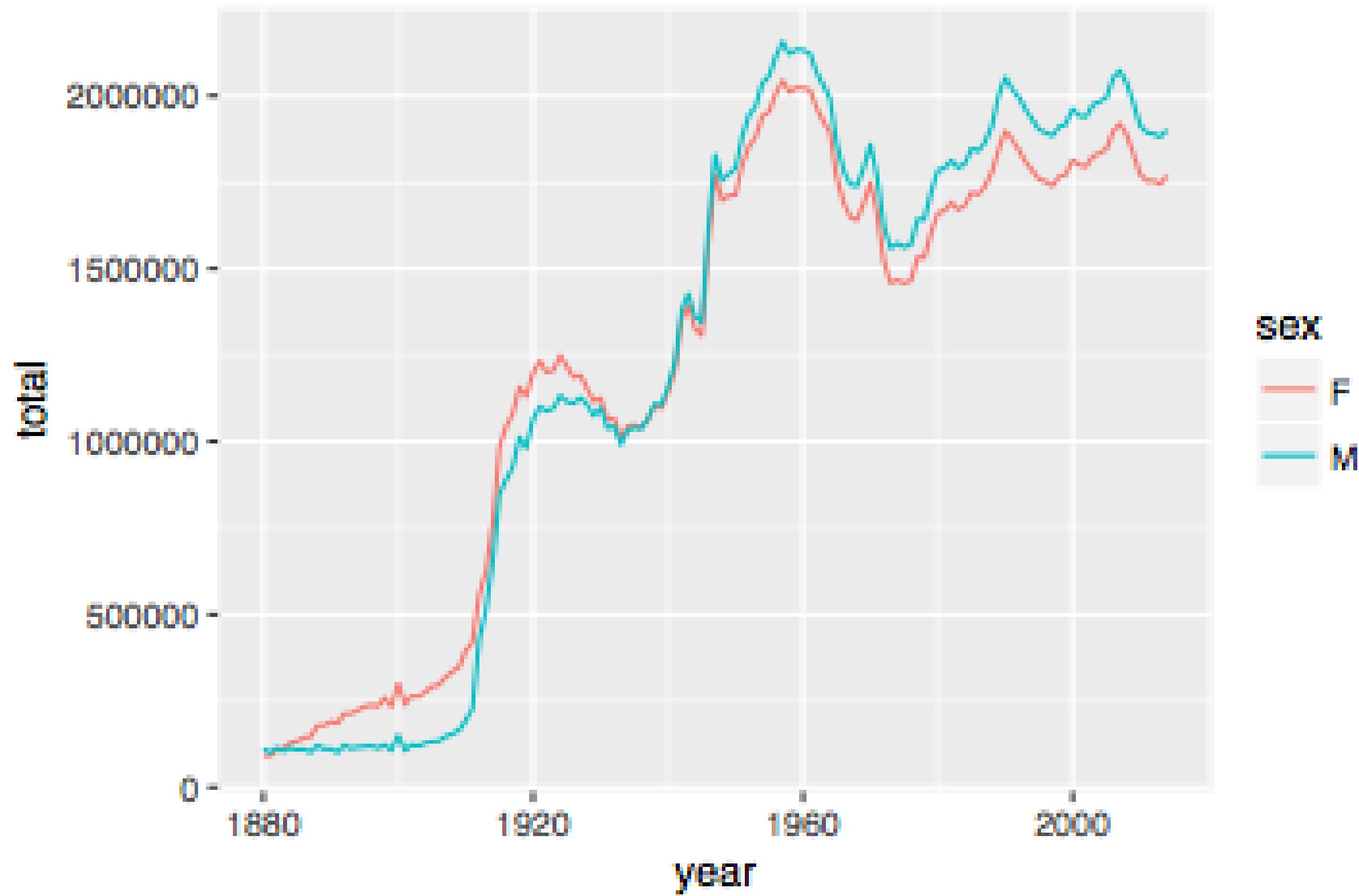
Groups cases by common values.

```
group_by(.data, ...)
```

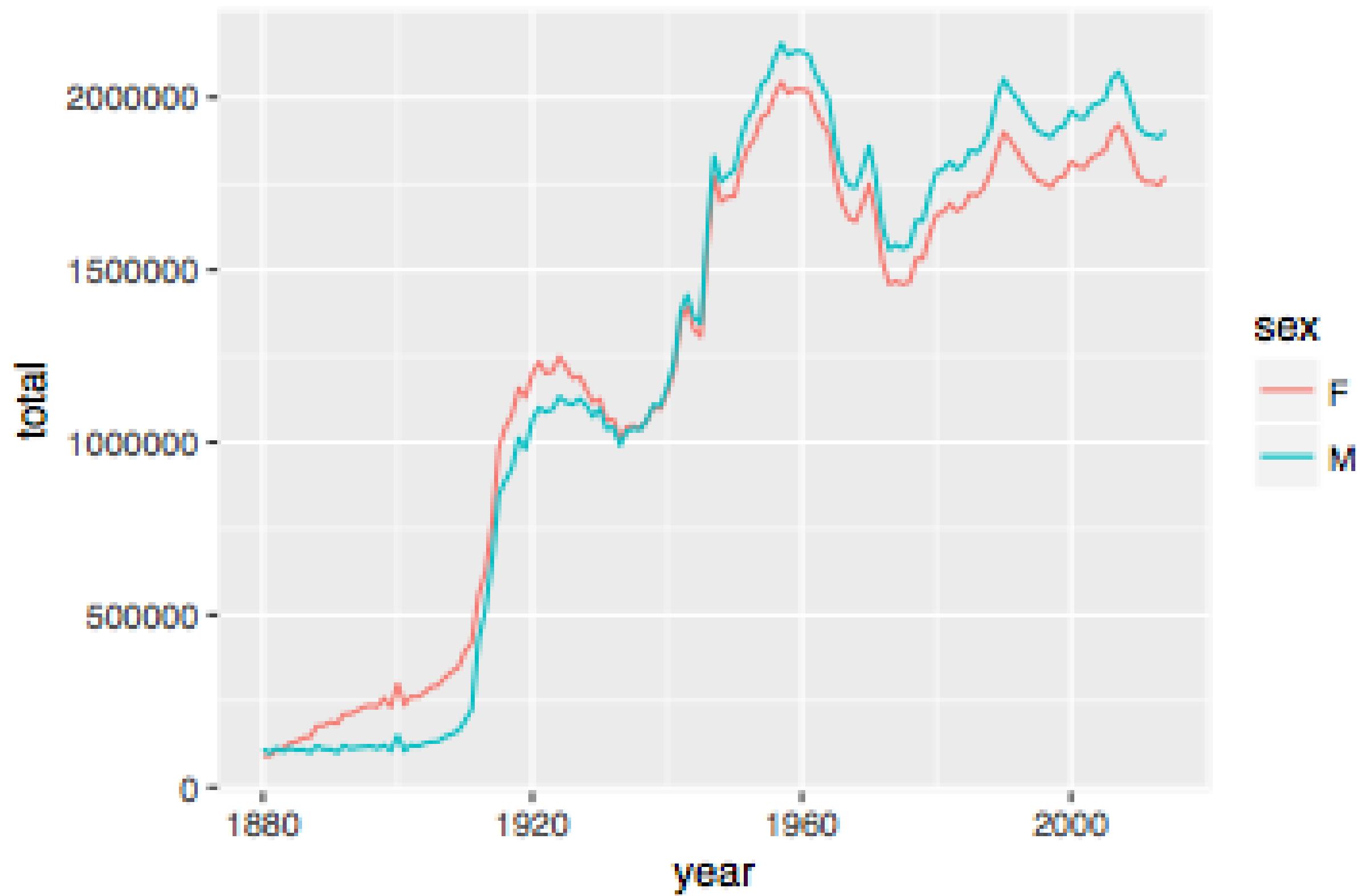
**data frame to
transform**

column(s) to group by
(will group by combinations of values if
more than one column is specified)

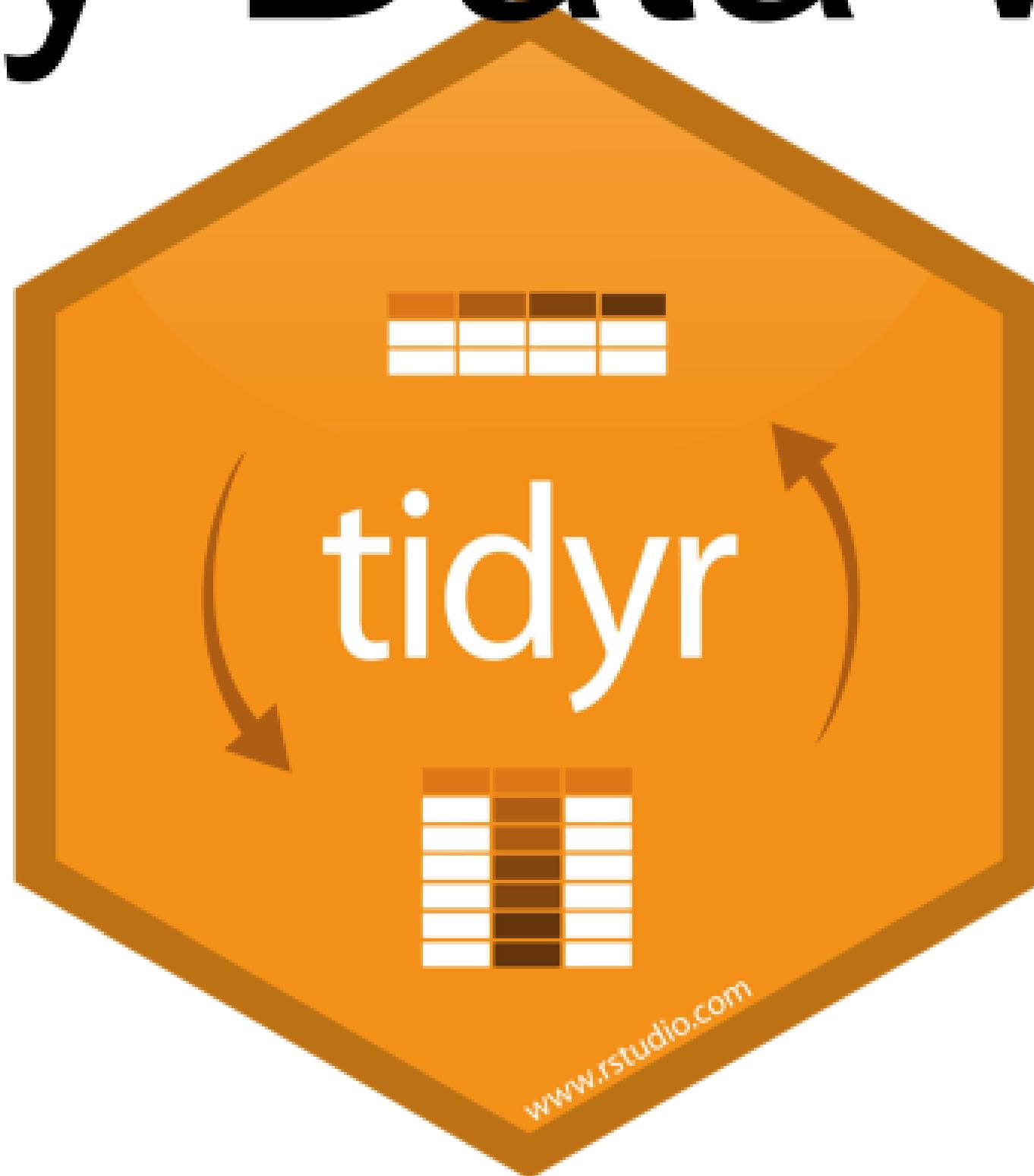
```
babynames %>%  
  group_by(year, sex) %>%  
  summarise(total = sum(n)) %>%  
  ggplot(aes(x = year, y = total, color = sex)) +  
  geom_line()
```



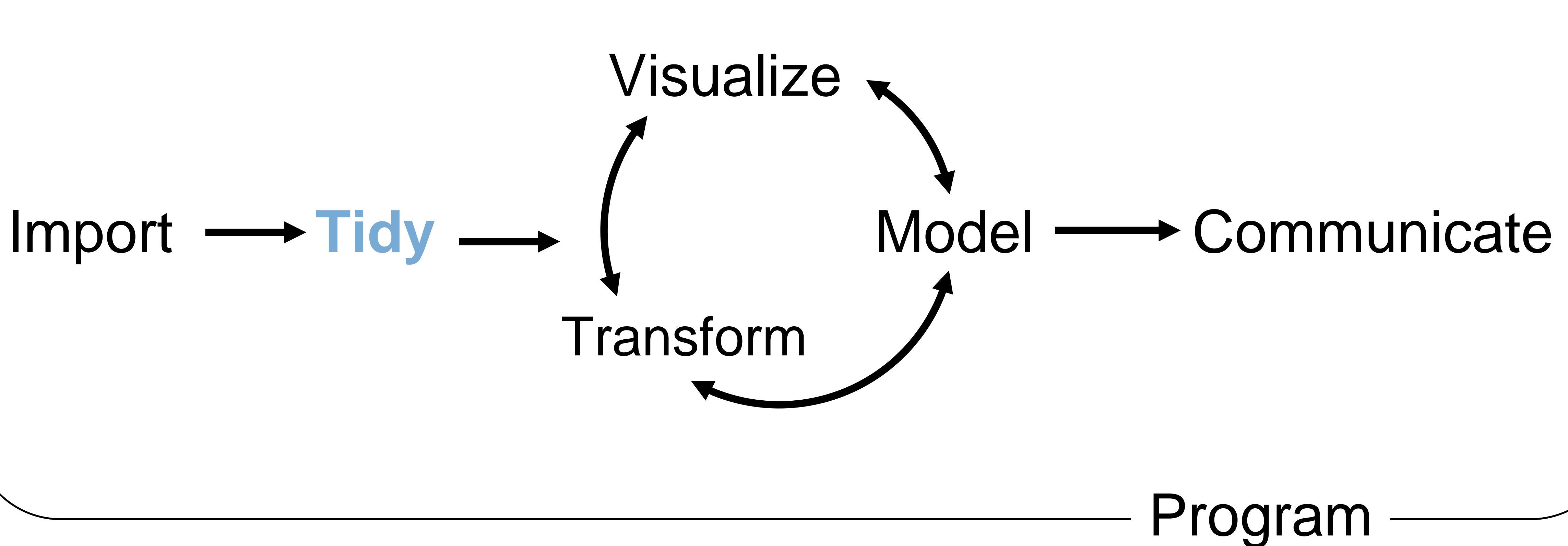
```
babynames %>%  
  group_by(year, sex) %>%  
  summarise(total = sum(n)) %>%  
  ggplot(aes(x = year, y = total, color = sex)) +  
  geom_line()
```



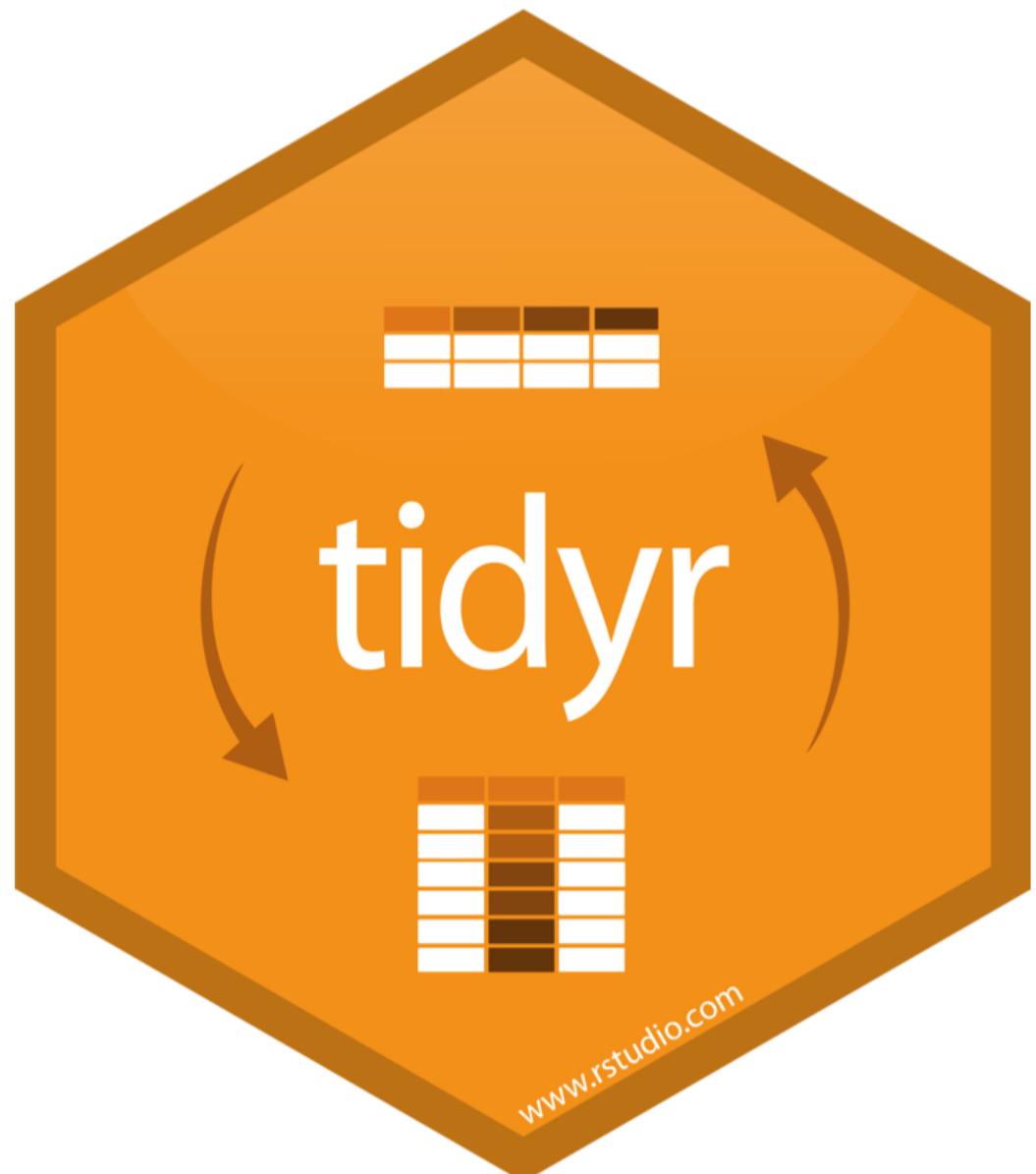
Tidy Data with



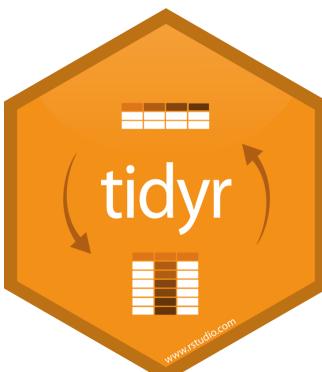
(Applied) Data Science



tidyr



A package that reshapes the layout
of tabular data.

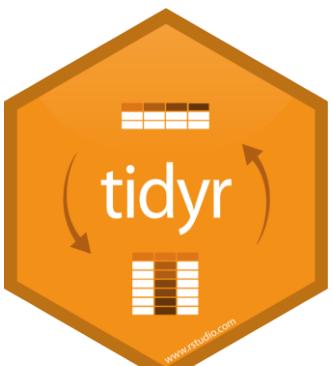


Reshaping tables

`spread()` is one of a family of functions for reshaping tables.

- `spread()` - move values into column names
- `gather()` - move column names into values
- `separate()` - separate variables that share a column
- `unite()` - unite a variable that is split across several columns

Most data comes to you untidy

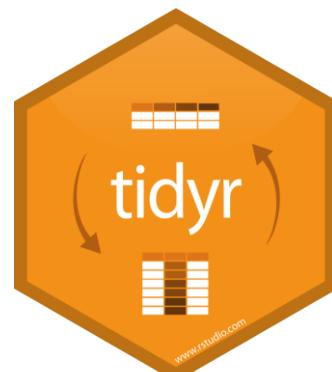


spread()
and gather()



You can represent any group of values as **key:value pairs**

key	value
2011	: 7000
2011	: 5800
2011	: 15000
2012	: 6900
2012	: 6000
2012	: 14000
2013	: 7000
2013	: 6200
2013	: 13000



You can represent any group of values as **key:value pairs** or a **group of columns** with column names.

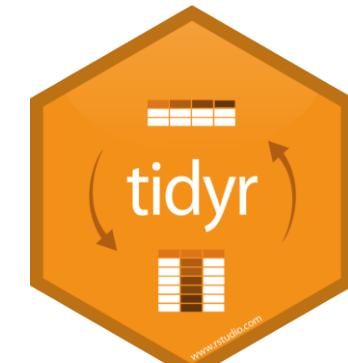
The diagram illustrates the transformation between two data representations using the `tidyverse` package's `spread()` and `gather()` functions. A large blue arrow points from the left table to the right table, labeled `spread()`. A green arrow points from the right table back to the left table, labeled `gather()`.

Key:Value Pairs (Left):

key	value
2011	7000
2011	5800
2011	15000
2012	6900
2012	6000
2012	14000
2013	7000
2013	6200
2013	13000

Group of Columns (Right):

	2011	2012	2013
7000	7000	6900	7000
5800	5800	6000	6200
15000	15000	14000	13000



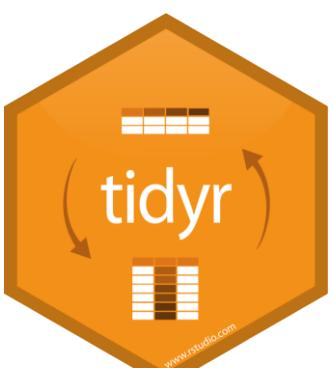
spread()

```
pollution %>% spread(key = size, value = amount)
```

data frame to
reshape

column to use for keys
(becomes new
column names)

column to use for
values
(becomes new
column cells)



gather()

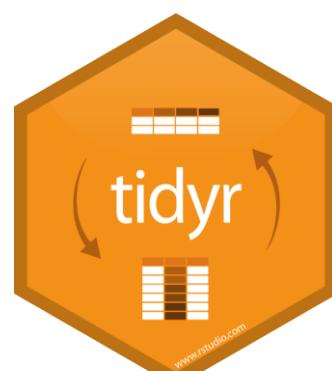
```
cases %>% gather(key = "year", value = "n", 2:4)
```

data frame to
reshape

name of the
new key
column
(a character
string)

name of the
new value
column
(a character
string)

numeric
indexes of
columns to
collapse
(or names)



separate()

R

separate()

Splits a column by dividing values at a specific character.

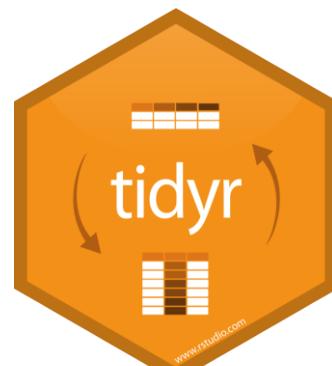
```
separate(storms, date, c("year","month","day"), sep = "-")
```

**data frame to
reshape**

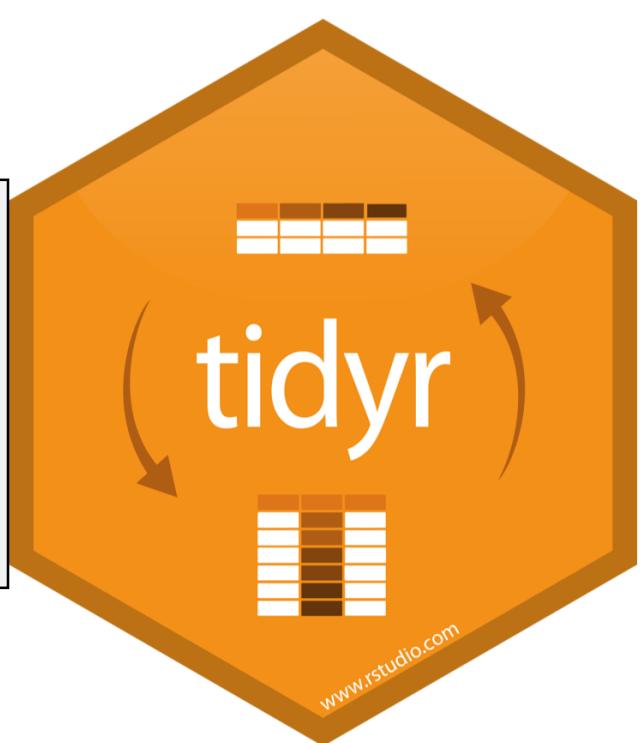
**a column to
split**

**names of new
columns to
make**

string to split on
(Defaults to any non_alpha-
numeric character)



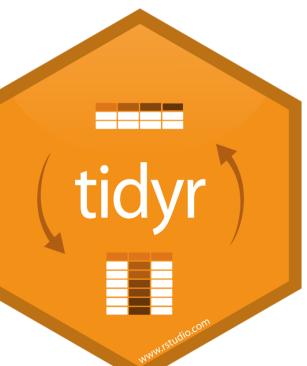
```
separate(storms, date, c("year","month","day"), sep = "-",
convert = TRUE)
```



	storm	wind	pressure	date
1	Alberto	110	1007	2000-08-03
2	Alex	45	1009	1998-07-27
3	Allison	65	1005	1995-06-03
4	Ana	40	1013	1997-06-30
5	Arlene	50	1010	1999-06-11
6	Arthur	45	1010	1996-06-17



	storm	wind	pressure	year	month	day
1	Alberto	110	1007	2000	08	03
2	Alex	45	1009	1998	07	27
3	Allison	65	1005	1995	06	03
4	Ana	40	1013	1997	06	30
5	Arlene	50	1010	1999	06	11
6	Arthur	45	1010	1996	06	17



unite()

R

unite()

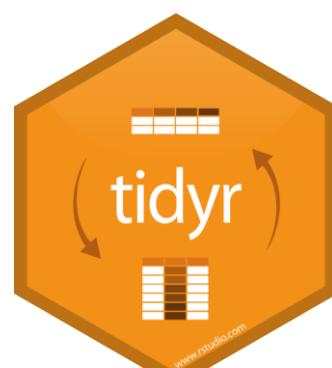
Unites columns into single column by combining cells.

```
unite(data, col, ..., sep = "")
```

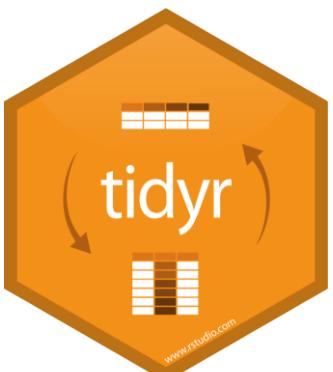
name of new
column to make
(in quotes)

two or more
columns to combine

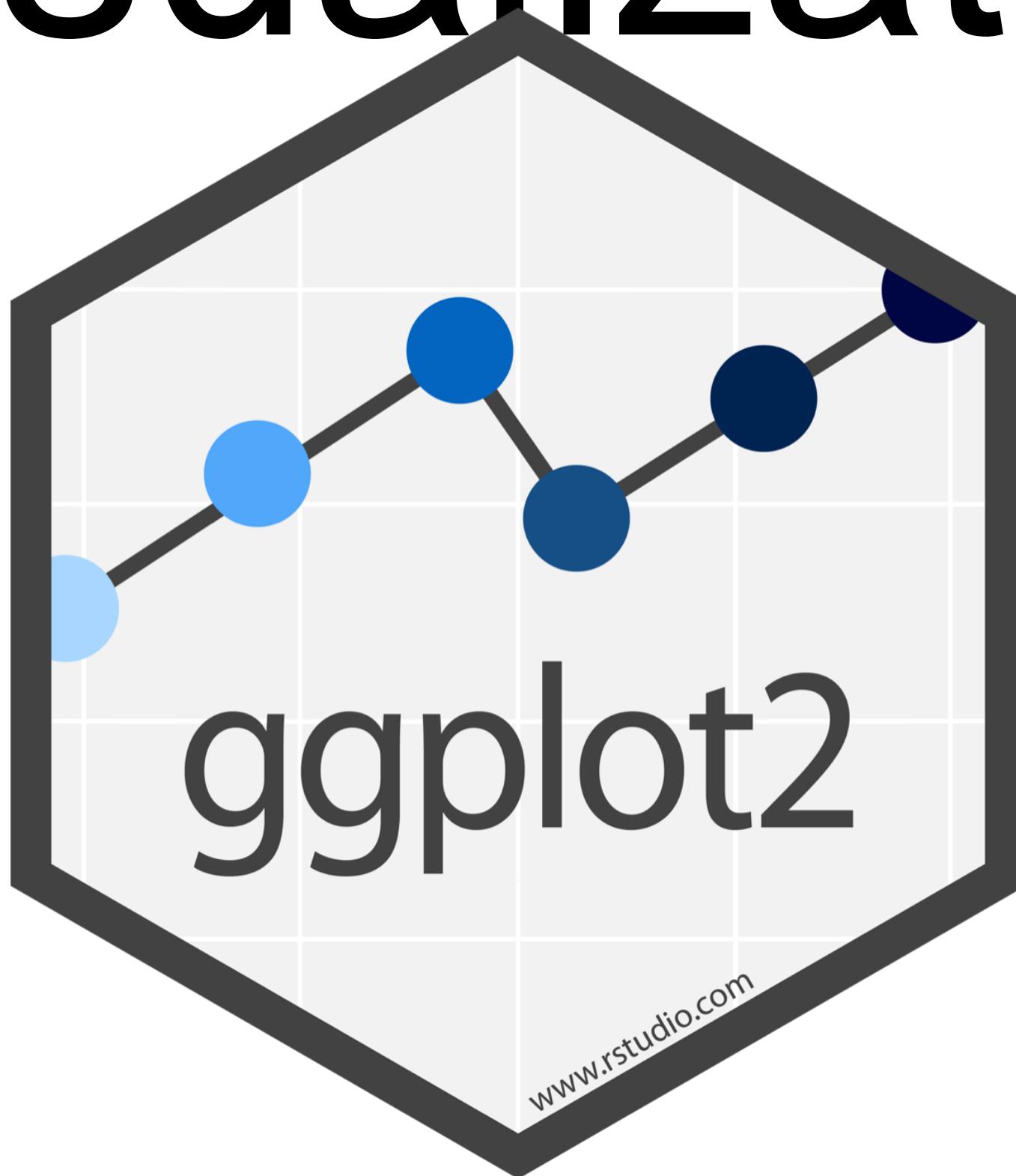
separator to place between elements in
new column (Defaults to an underscore)



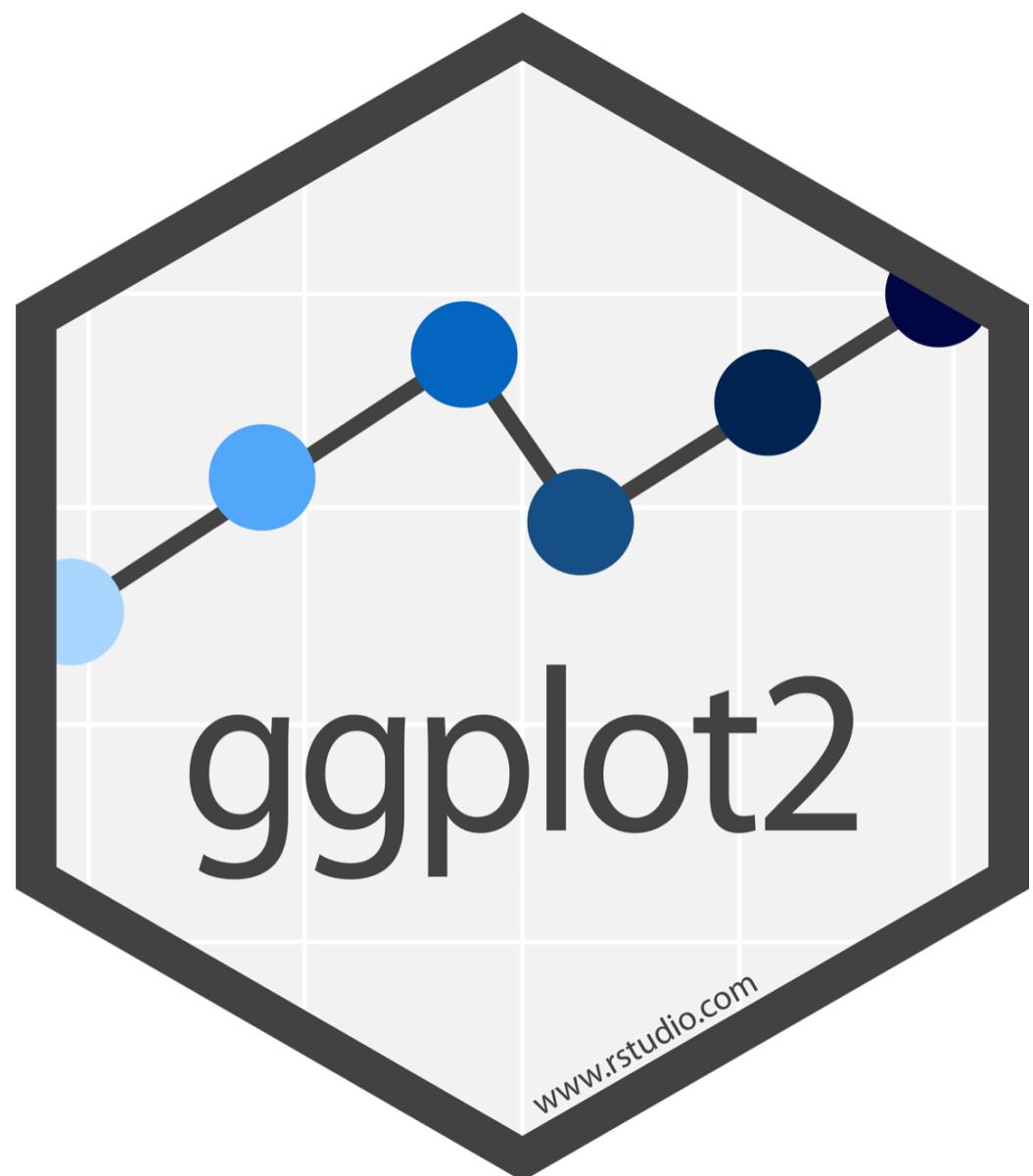
```
storms %>%  
  separate(date, c("year", "month", "day"), sep = "-") %>%  
  unite("date", month, day, year, sep = "/")
```



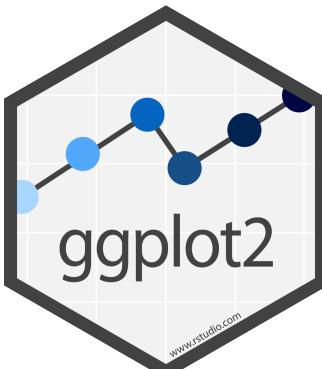
Data Visualization with



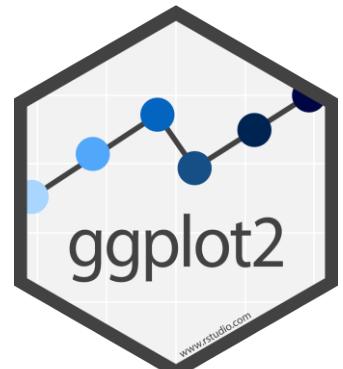
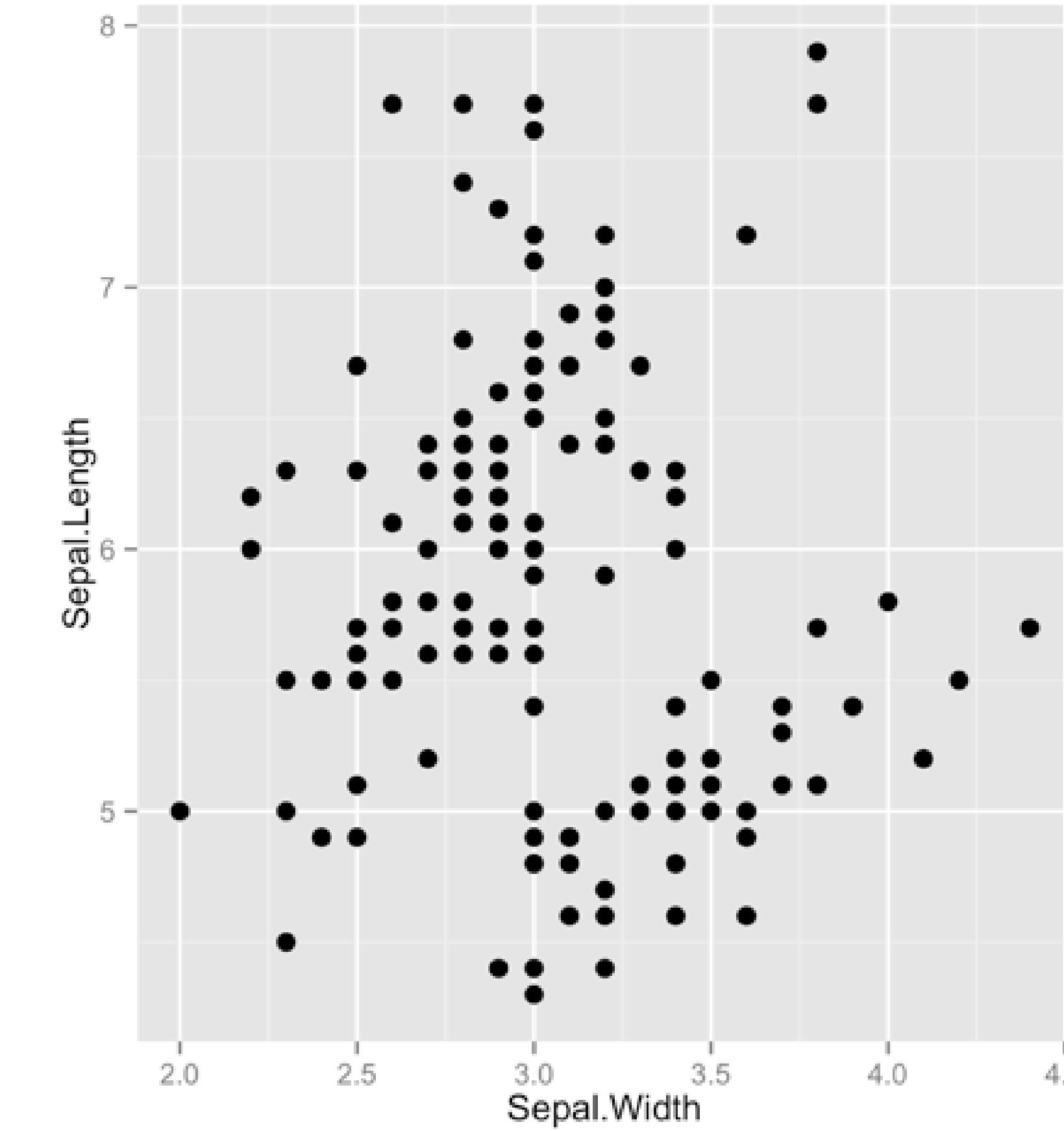
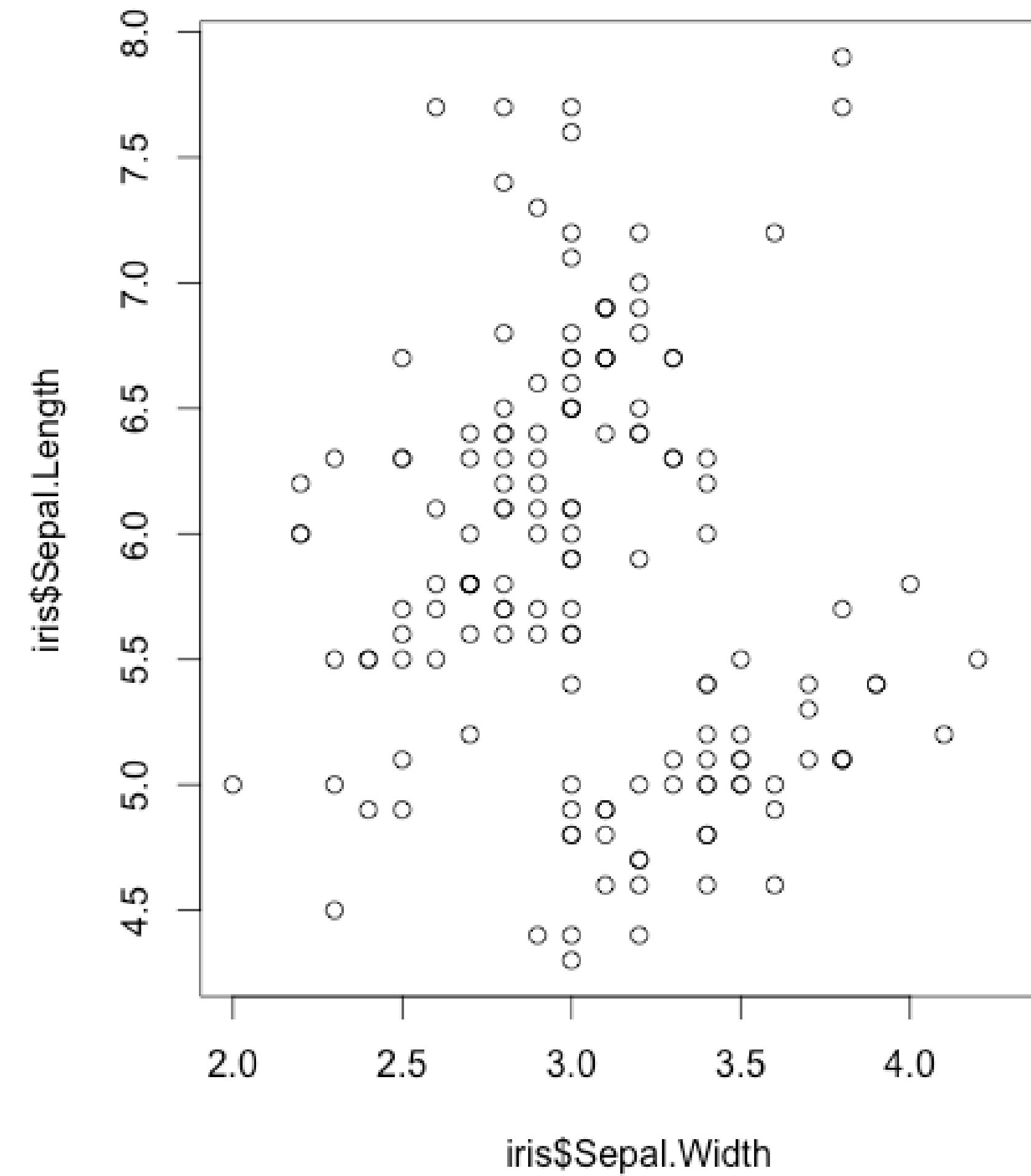
ggplot2



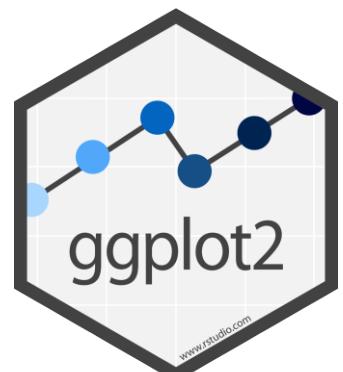
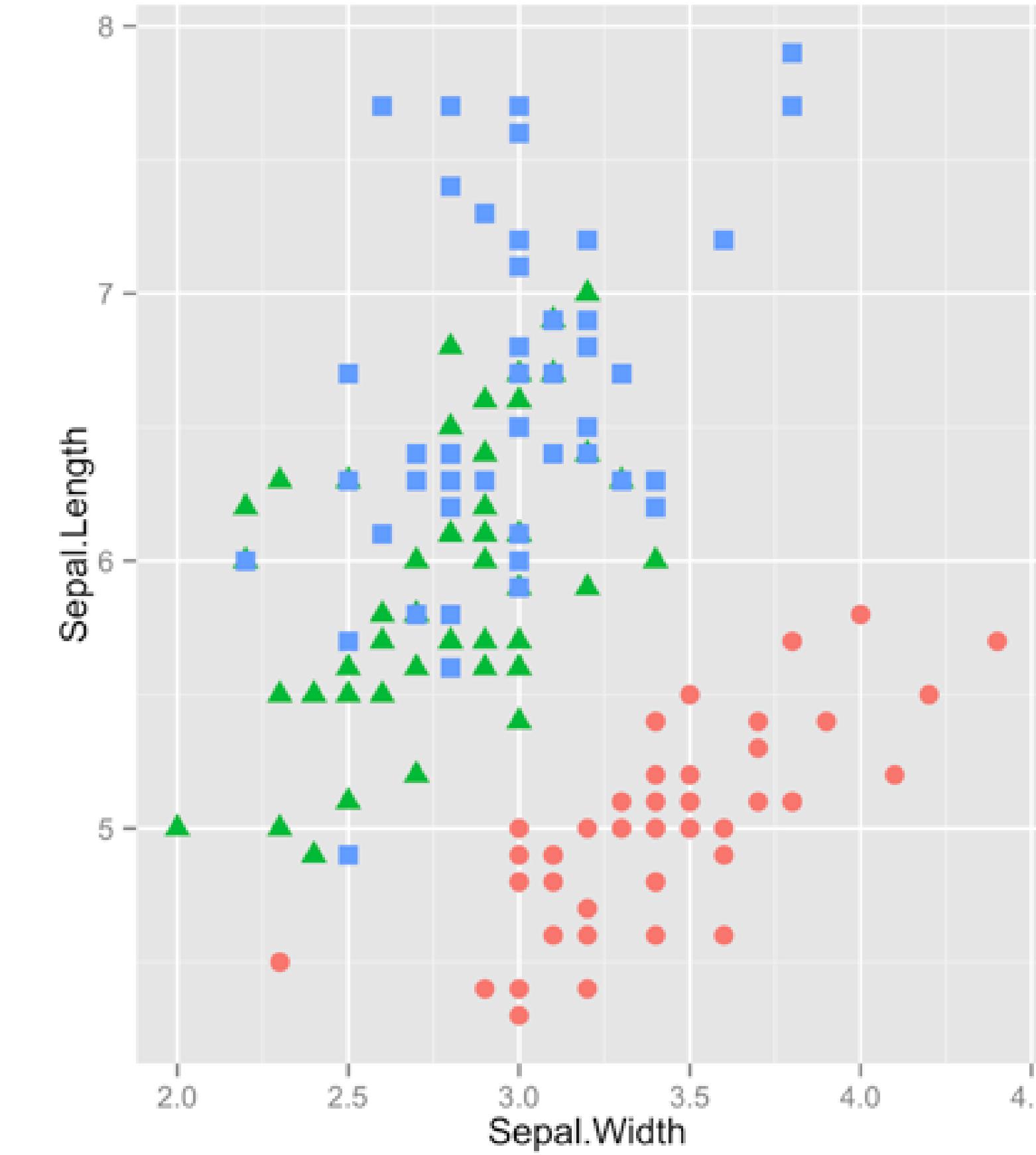
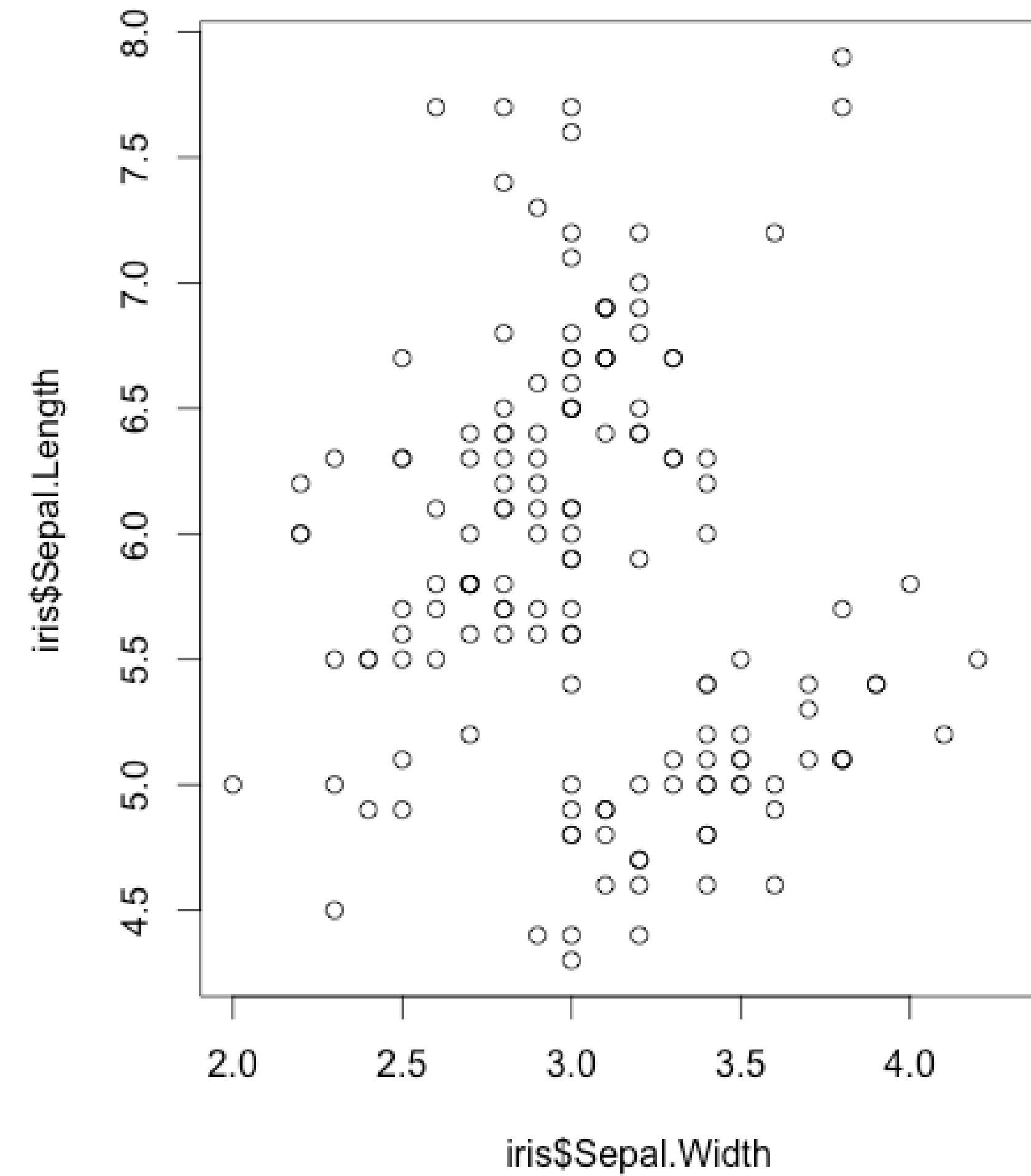
A package that visualizes data.
ggplot2 implements the *grammar of graphics*, a system for building visualizations that is built around **cases** and **variables**.



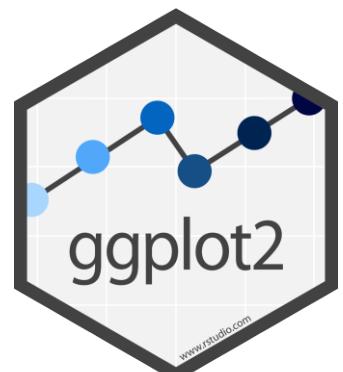
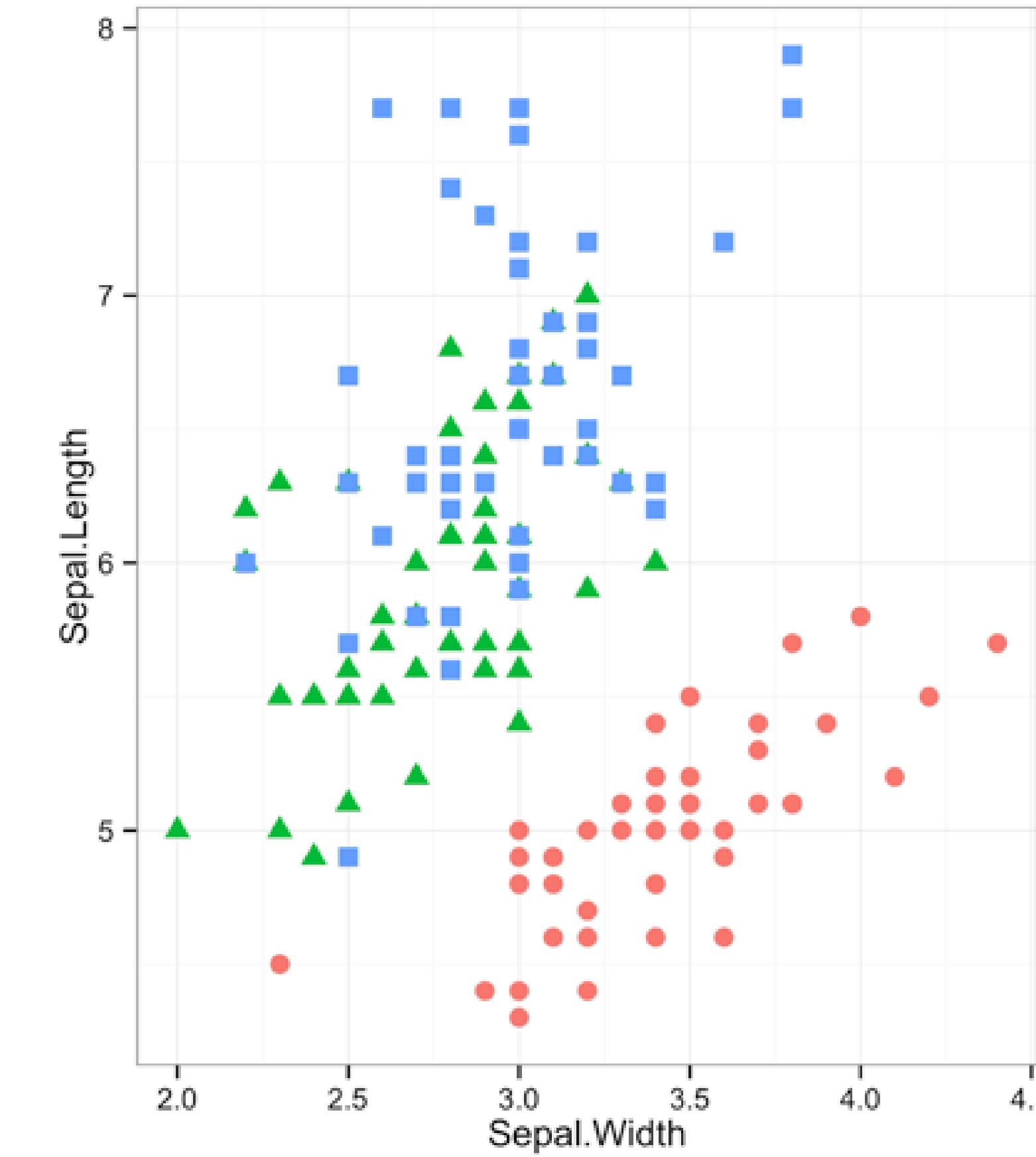
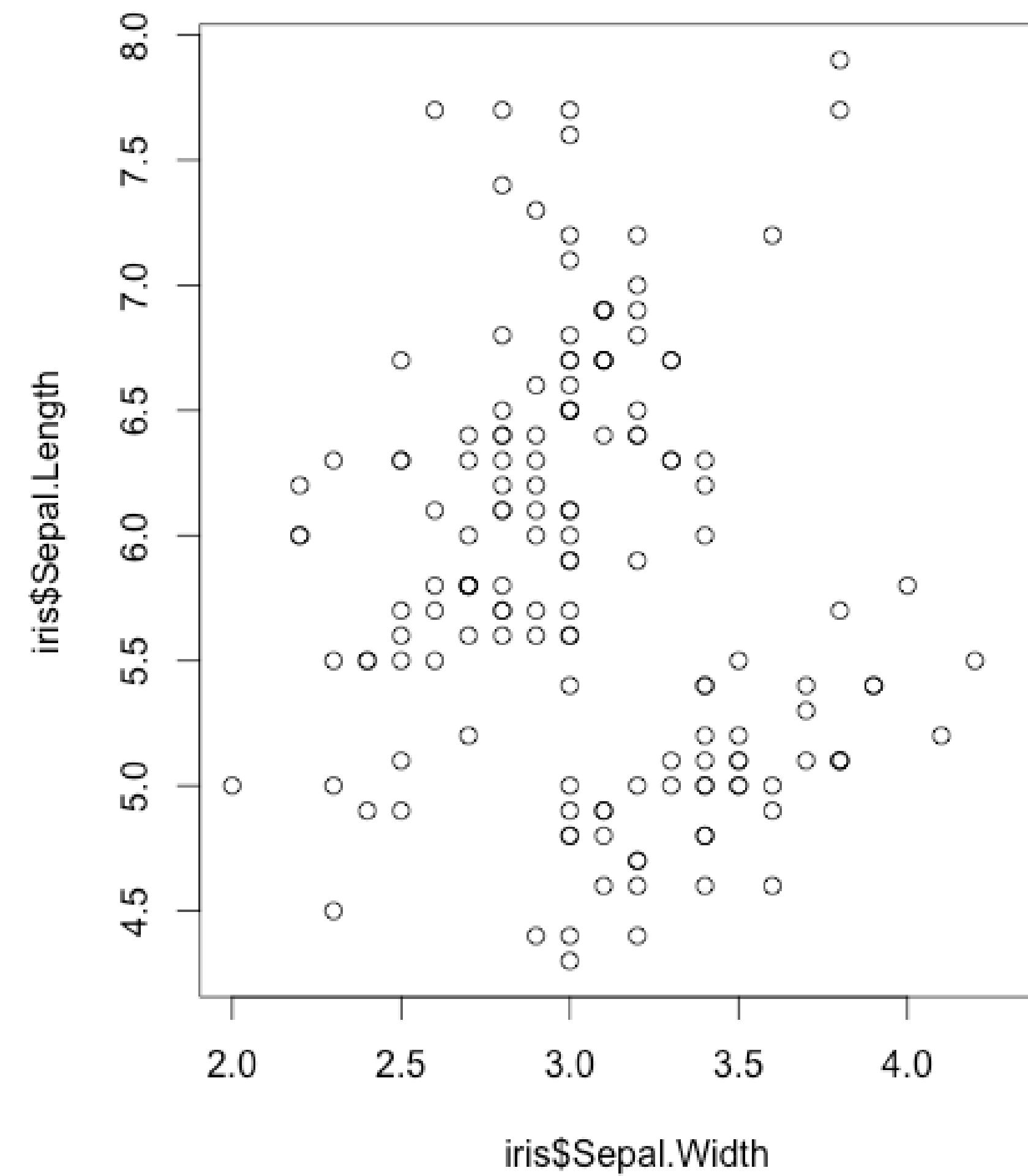
ggplot2



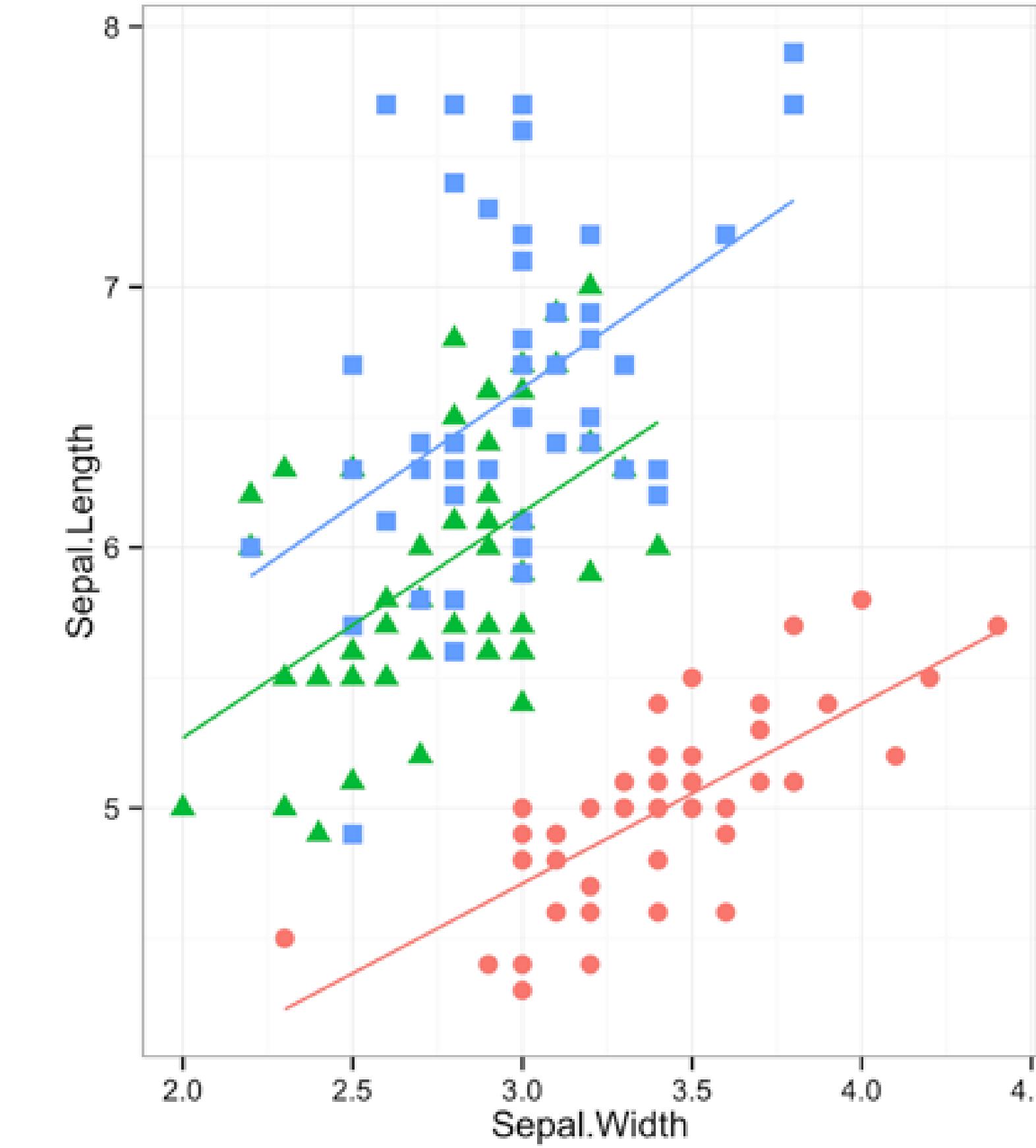
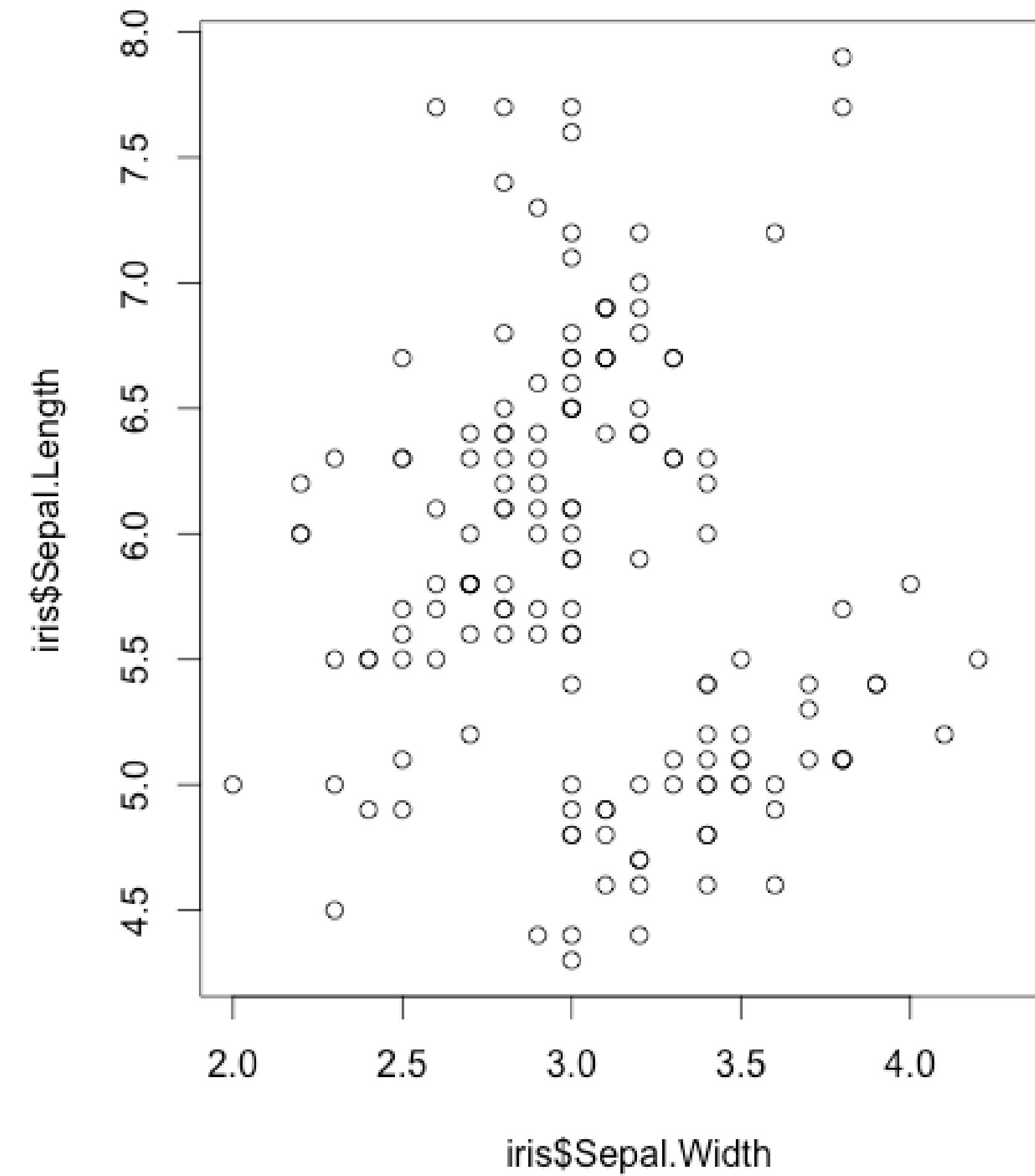
ggplot2



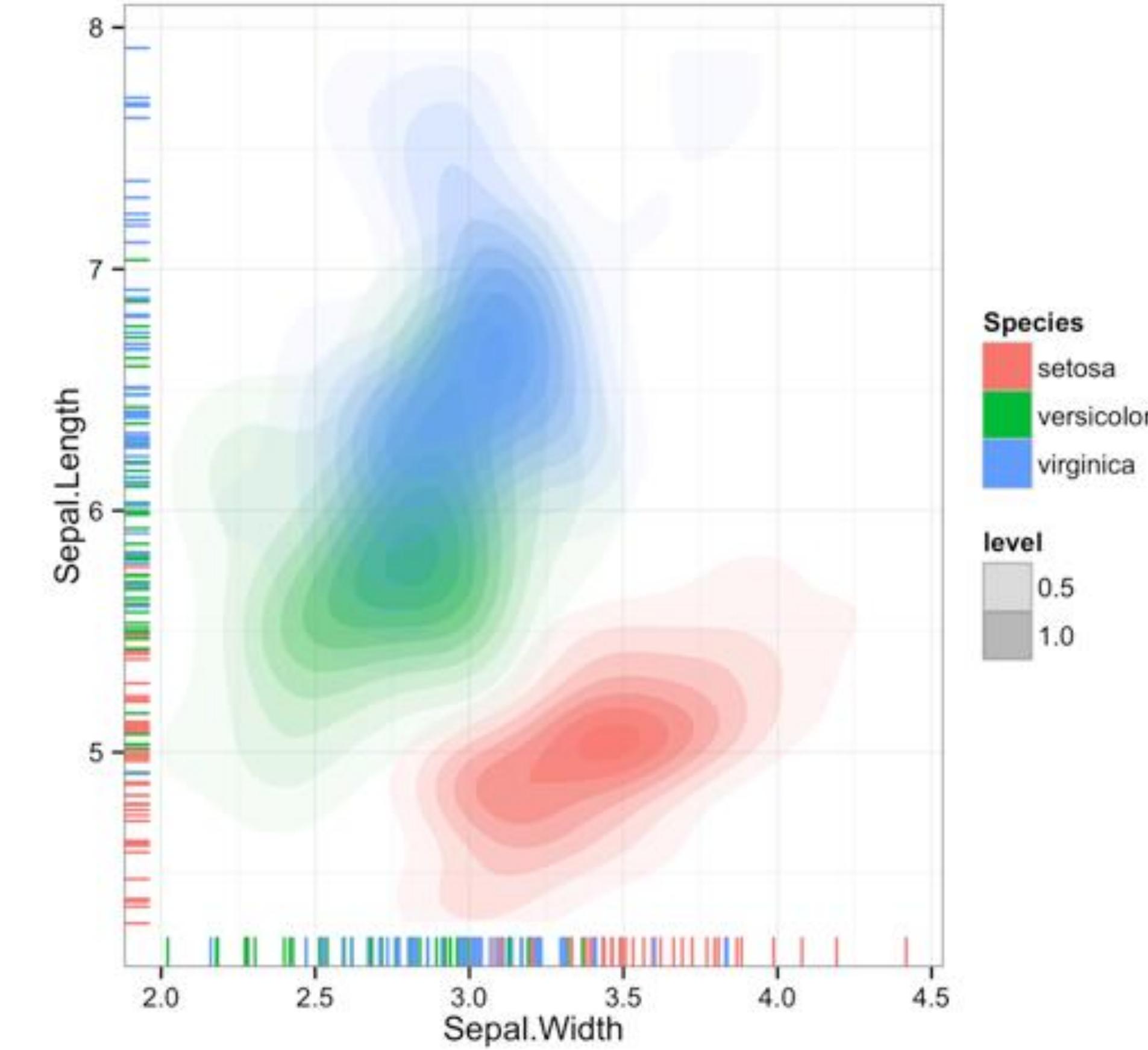
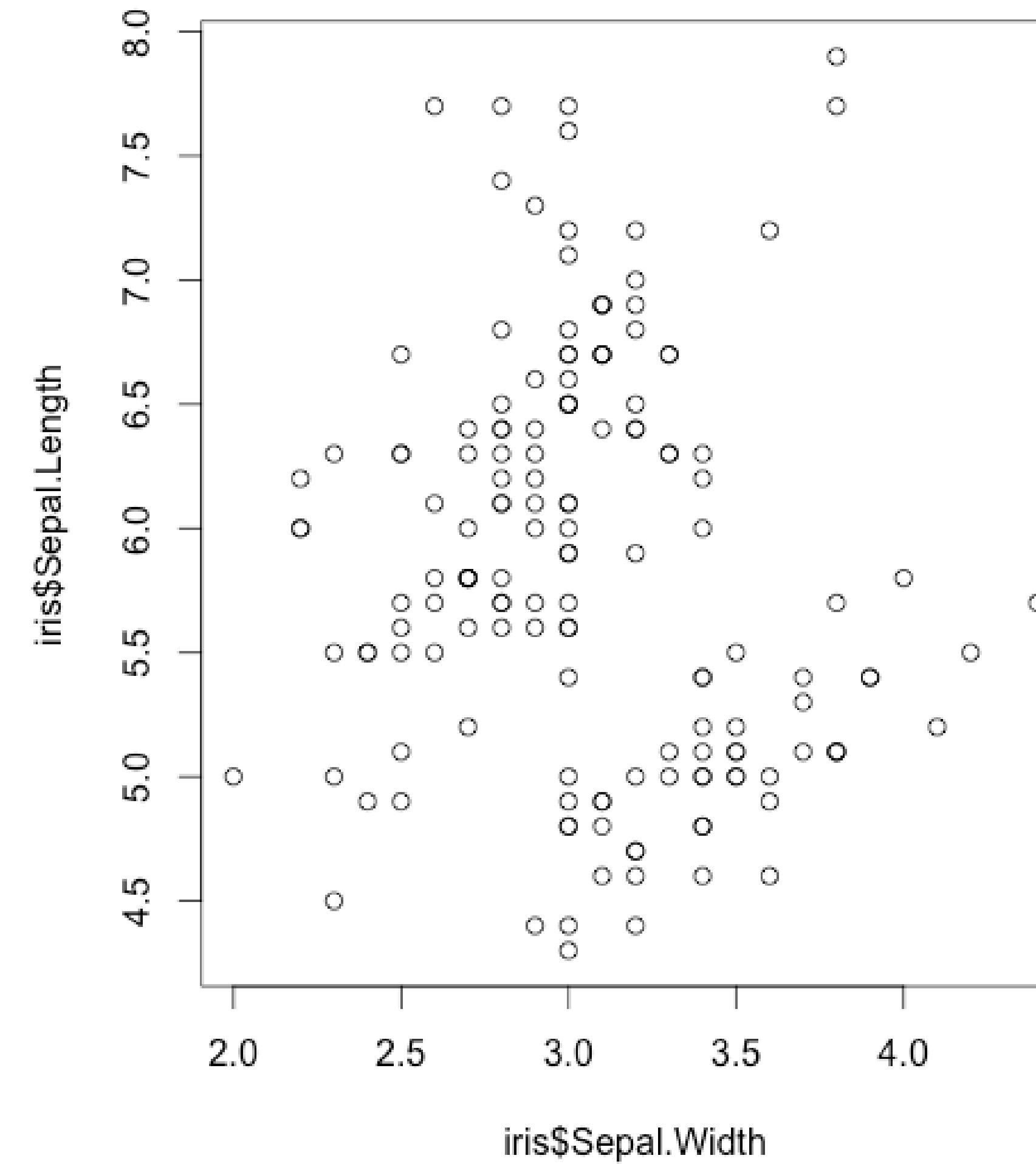
ggplot2



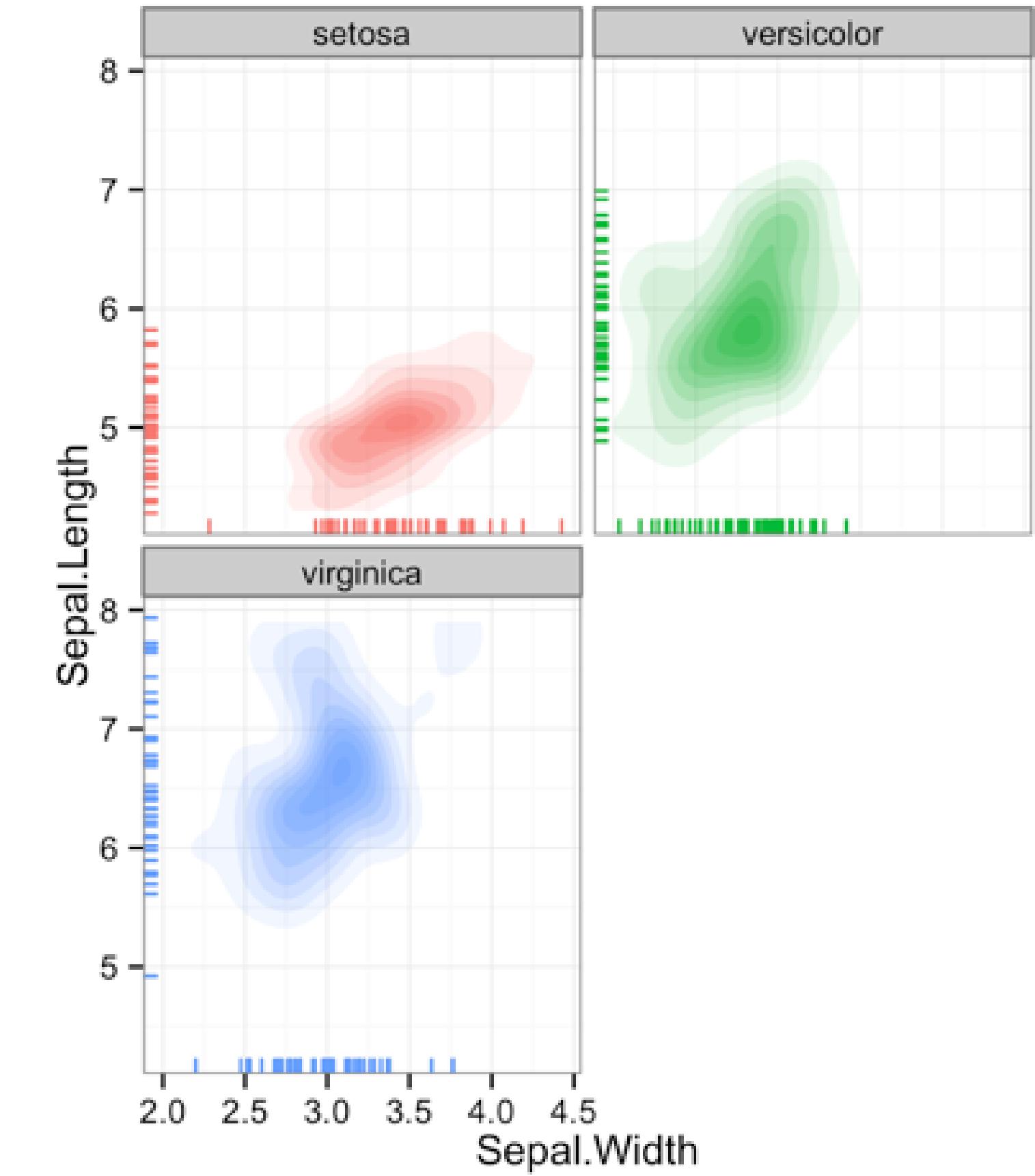
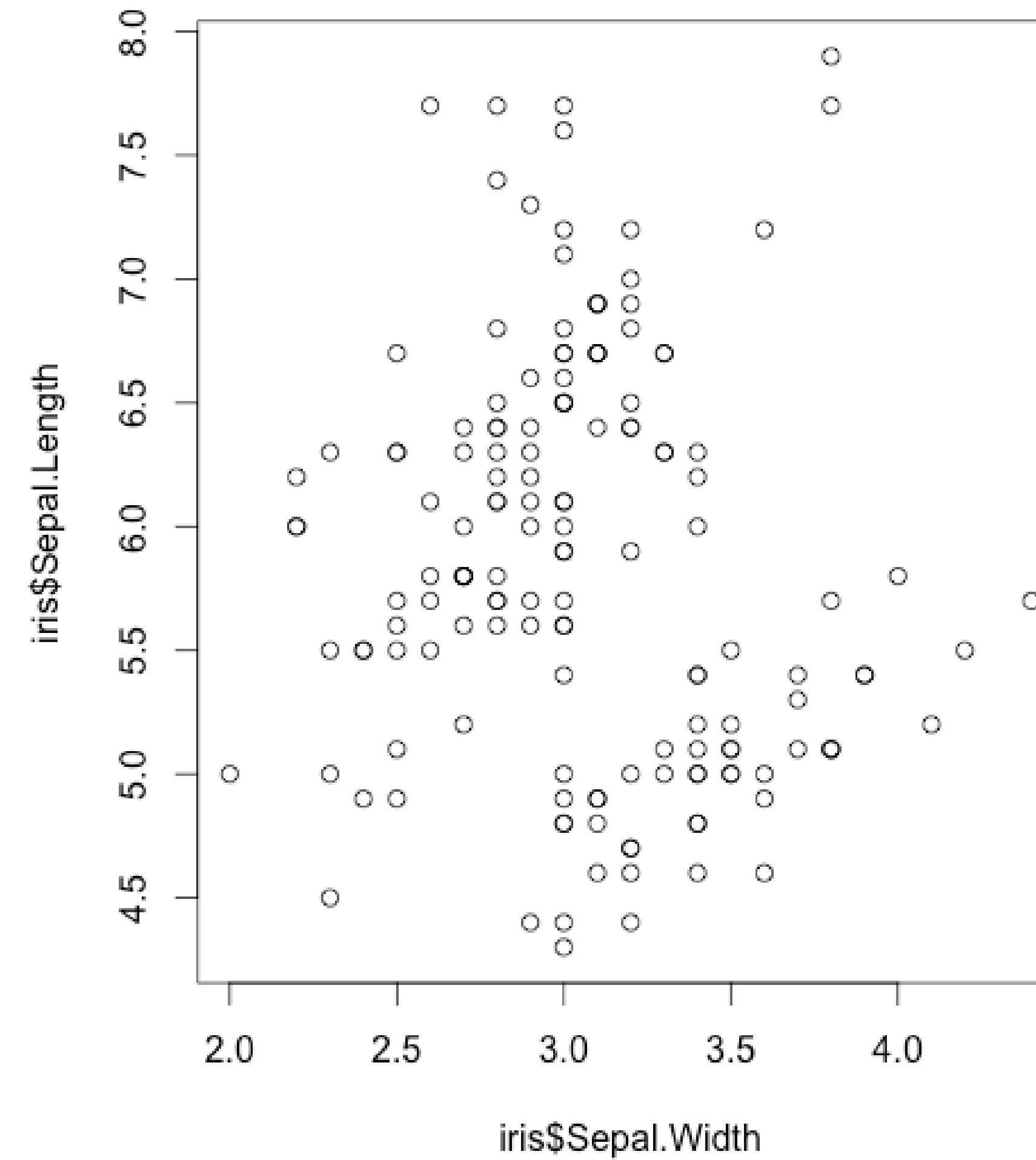
ggplot2



ggplot2



ggplot2



To plot

1. "Initialize" a plot with `ggplot()`
2. Add layers with `geom_` functions

Pro tip: Always put the `+` at the end of a line,
Never at the beginning

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



To plot

1. "Initialize" a plot with `ggplot()`
2. Add layers with `geom_` functions

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

data + before new

type of layer

aes()

x variable

y variable

A ggplot2 template

Make any plot by filling in the parameters of this template

```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

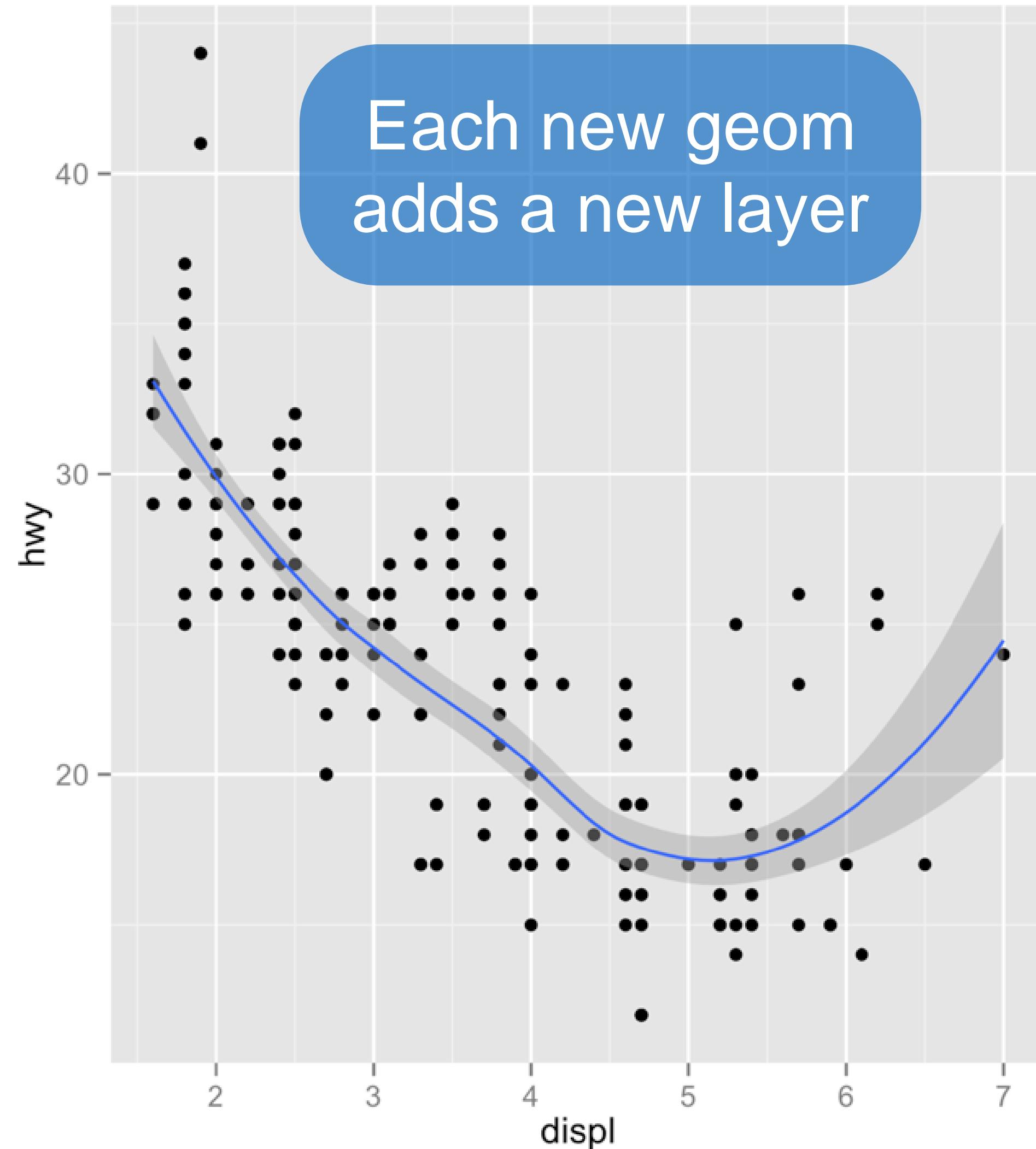
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

geom

The visual objects that represents the cases. geom functions add layers to the graph.

```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

The geom



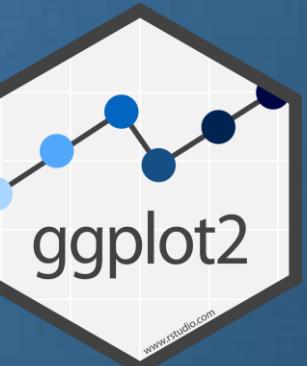
```
ggplot(mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

A ggplot2 template

Make any plot by filling in the parameters of this template

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

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut), stat = "count")
```



Data types with



Iteration with



Modeling with

