DSA2101

Essential Data Analytics Tools: Data Visualization

Yuting Huang

Weeks 12-13 Exploring data through visualization

The process of making plots

In this week, we shall put the techniques and principles that we have covered so far into practice.

- Making a plot should not be a matter of dumping your data into a software and then pasting the plot into PowerPoint.
- As Join W. Tukey said,

  *The **greatest value** of a picture is when it forces us to notice what **we never expected to see**.*

Exploratory data analysis

Visualization is an integral part of exploratory data analysis (EDA).

- It is a highly iterative process. We should expect to:
  - Generate questions about our data.
  - Search for answers by visualizing, transforming, and modeling out data.
  - Use what we learn to refine our questions and/or generate new questions.
- The final plot we produce should be carefully design.
  - We should not expect to have the final plot ready in an instant. Even if we know exactly what kind of plot we want, we should expect to plot it several times over.
  - Each time we plot it, we shall vary the colors, titles, labels, and so on. Until we are satisfied that it highlights the data and the message it carries with.

Questions

When we inspect our data, the questions that we generate almost always falls into the umbrella questions:

1. What type of variation occurs within my variables?
2. What type of covariation occurs between my variables?

Making comparisons

From Join W. Tukey:

> *Two kinds of comparisons come up in the simplest of common language:*
>
> *"Bill is a head taller than Jim."*
>
> *"George weighs twice as much as his brother Jack."*

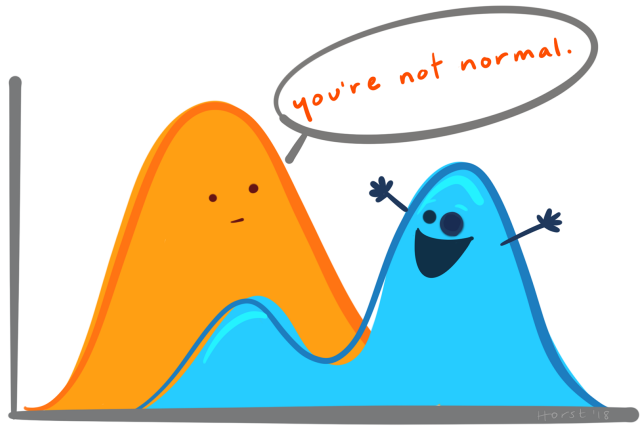
- The first type of comparison is based on addition. It is easy to understand.
- The second type is based on multiplication.
  - We can use logarithms to convert multiplication to addition.

Definitions

Let us recall some terms we first encountered when tidying our data:

- A **variable** is a quantity that we measure. In tidy format, it should be stored in its own column.
  - A variable is **categorical** if it can only take on a small set of values. In `R`, these are typically stored as a factor.
  - A variable is **continuous** if it can take on an infinite set of ordered values. In `R`, these are typically stored as numeric or integer.
- A **value** is a state of variable when we measure it. It should be in a single cell in our table.
- An **observation** is a set of measurements made under similar conditions. For data to be tidy, each observation should be stored in its own row.

General guidelines



When we display a chart, here are a few general guidelines regarding what we should discuss:

1. Which values are the most common? Can we say why?
2. Which values are rare? Why?
3. Are there any unusual patterns?

diamonds data

Let us work with a diamonds data set from the tidyverse package.

```
library(tidyverse)
str(diamonds)
```

```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 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 [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39
```

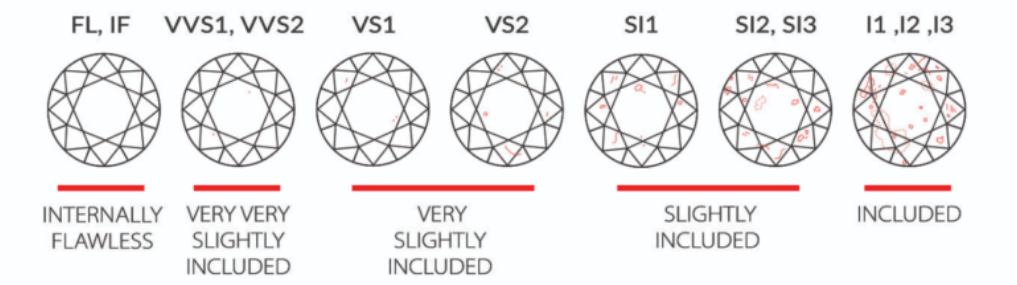
Data description

The data contain information on the prices and other attributes for almost 54,000 diamonds.

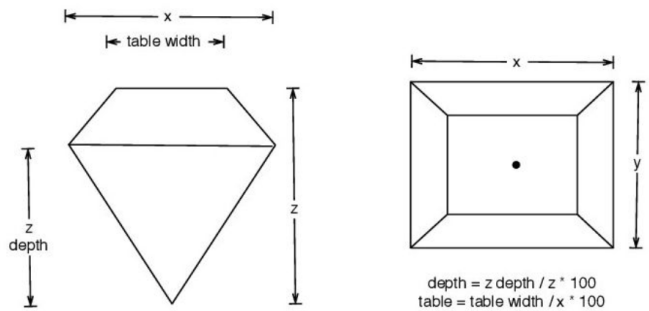
- `carat` is a unit of mass equal to 200 mg and is used for measuring gemstones and pearls.
- `cut` grade is an objective measure of a diamond's light performance.
- `color` is another dimension to evaluating the a diamond. The less color, the higher the value.

Data description

- `clarity` is a measure on how clear the diamond is.



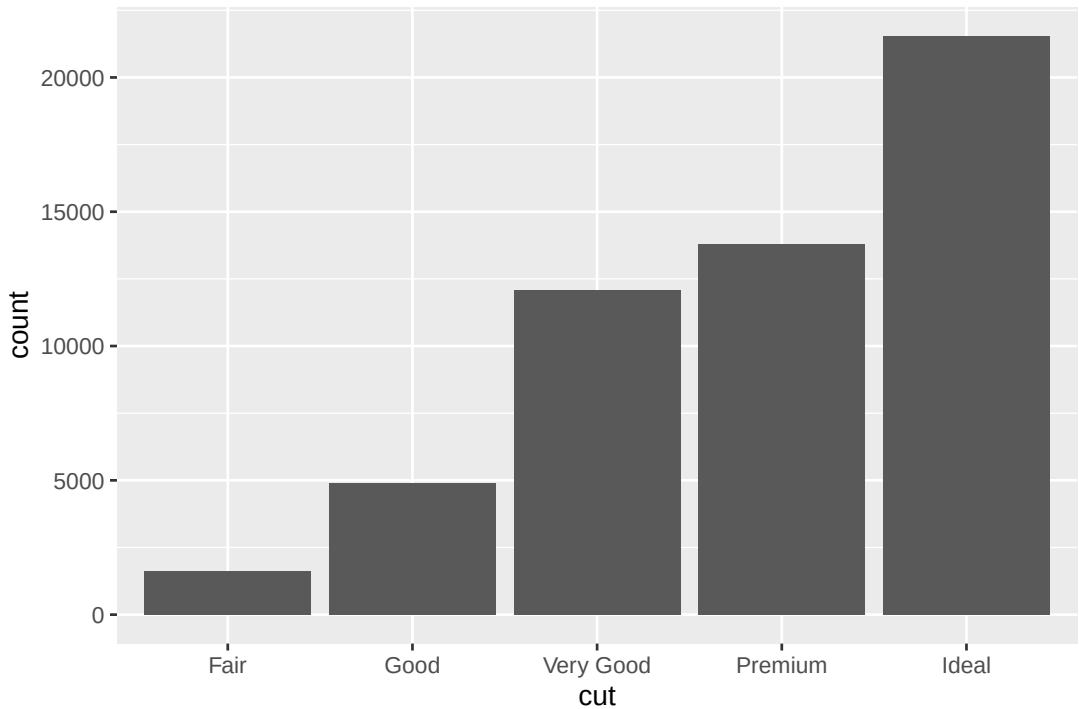
- There are five physical measurements, `depth`, `table`, `x`, `y`, and `z`:



Diamond quality, bar chart

A **bar chart** is a display for one categorical variable.

```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut))
```



What can we say about the bar chart?

- The **modal category** is Ideal – the highest-quality cut possible.
- The **count** for Ideal is approximately 30% higher than the next most frequent category.
- The category with the lowest count is Fair.
- The counts for Very Good and Premium are close to each other.

Contingency Table

A **contingency table** is a display for two categorical variables.

- Its rows list the categories for one variable and its columns list the categories of the other variable.
- Each entry is the number of observations at a particular combination of the categories of the two variables.

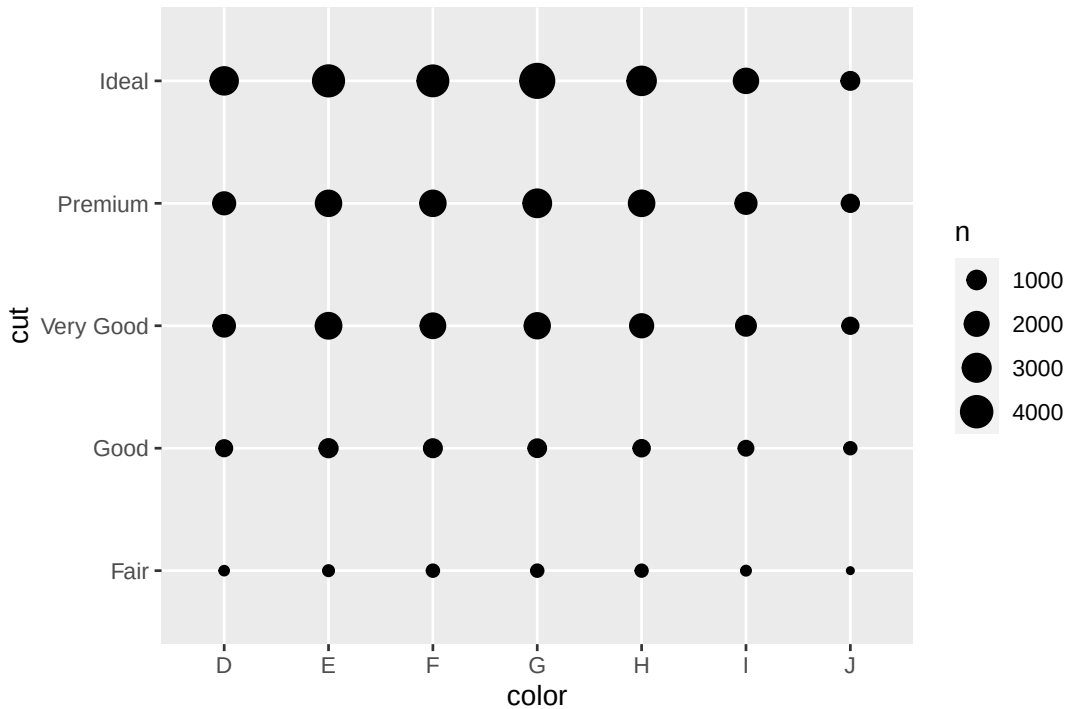
```
diamonds %>% count(cut, color) %>% spread(key = cut, value = n)
```

```
## # A tibble: 7 x 6
##   color  Fair  Good `Very Good` Premium Ideal
##   <ord> <int> <int>       <int>   <int> <int>
## 1 D       163   662        1513    1603  2834
## 2 E       224   933        2400    2337  3903
## 3 F       312   909        2164    2331  3826
## 4 G       314   871        2299    2924  4884
## 5 H       303   702        1824    2360  3115
## 6 I       175   522        1204    1428  2093
## 7 J       119   307         678     808   896
```

Visualizing a contingency table

`geom_count()` maps the count to the area of points. It is useful for visualizing a contingency table.

```
ggplot(data = diamonds) +
  geom_count(aes(x = color, y = cut))
```



Visualizing a contingency table

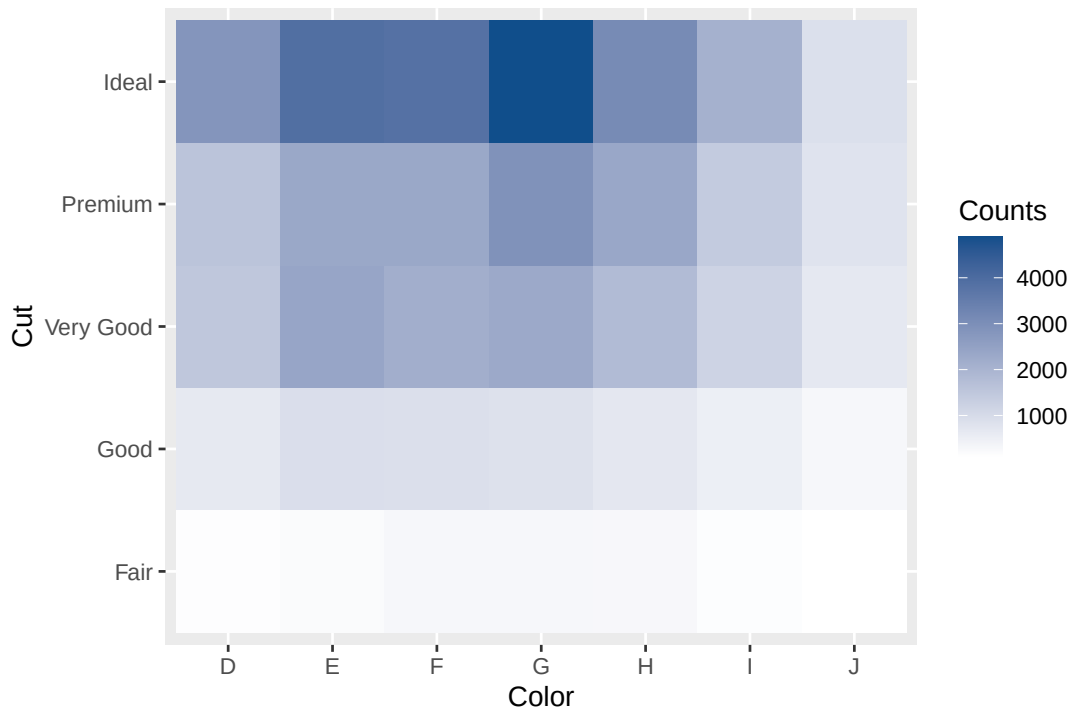
By default, the size of the points is mapped to the count at each combination of the `x` and `y`-categories.

- This is not ideal, as it is difficult to compare the sizes of circle by eye.
- Moreover, the guide on the right is not useful for comparisons.

An alternative is to reflect count size using color, with `geom_tile()`.

Visualizing a contingency table

```
diamonds %>%
  count(color, cut) %>%
  ggplot(aes(x = color, y = cut)) +
  geom_tile(aes(fill = n)) +
  scale_fill_gradient(low = "white", high = "dodgerblue4") +
  labs(x = "Color", y = "Cut", fill = "Counts")
```



Regarding counts, circles, and colors

In the previous two plots, we used size of the circles, and then color of the tiles, to represent counts.

- When we used circles, the row corresponding to `Ideal cut` had large circles throughout, and all the circles in the `Fair` column are pretty much indistinguishable.
- When using different shades of blue, a **continuous color scale** is used.

Let us focus on the use of colors first. Then we shall address the visualization of **correlation** between two variables.

Colors

Colors can be thought of as a three-dimensional concept consisting of

- **Hue**: Red, green, blue, ...
- **Saturation**: The purity of light, e.g., dull versus vivid.
- **Brightness**: The amount of light present, e.g., light versus dark.

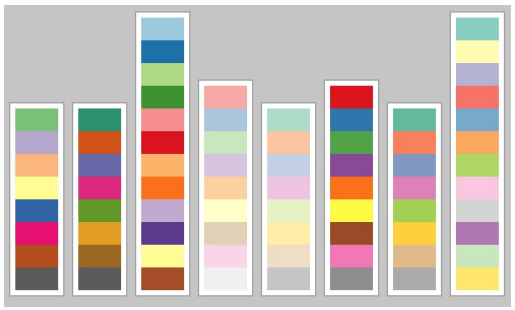
Apart from making graphs prettier and more pleasant to look at, colors add solid **functionality** to visual representations.

1. Distinguish between different **categorical groups**.
2. Distinguish between the magnitude of **continuous values**.

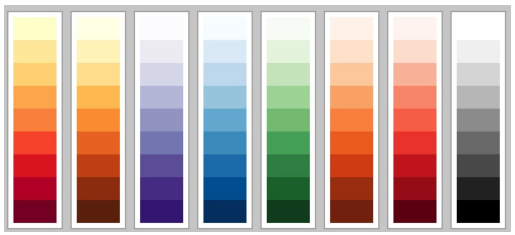
Types of color palettes

These functions roughly correspond to different types of color palettes.

- **Qualitative** color palettes for categorical data
  - To highlight distinction across groups



- **Sequential** color palettes for continuous data
  - Use increasing intensity or saturation of color to indicate larger values



Base R colors

To gain control over colors, we first need to define colors or color palettes.

- Base `R` comes with 657 predefined colors.
  - We can call them by names, e.g., `col = "dodgerblue4"`
- The default color palette in `R` (using version 4.2.2):

```
palette()
```

```
## [1] "black"   "#DF536B" "#61D04F" "#2297E6" "#28E2E5" "#CD0BBC" "#F5C710"
## [8] "gray62"
```

HEX color codes

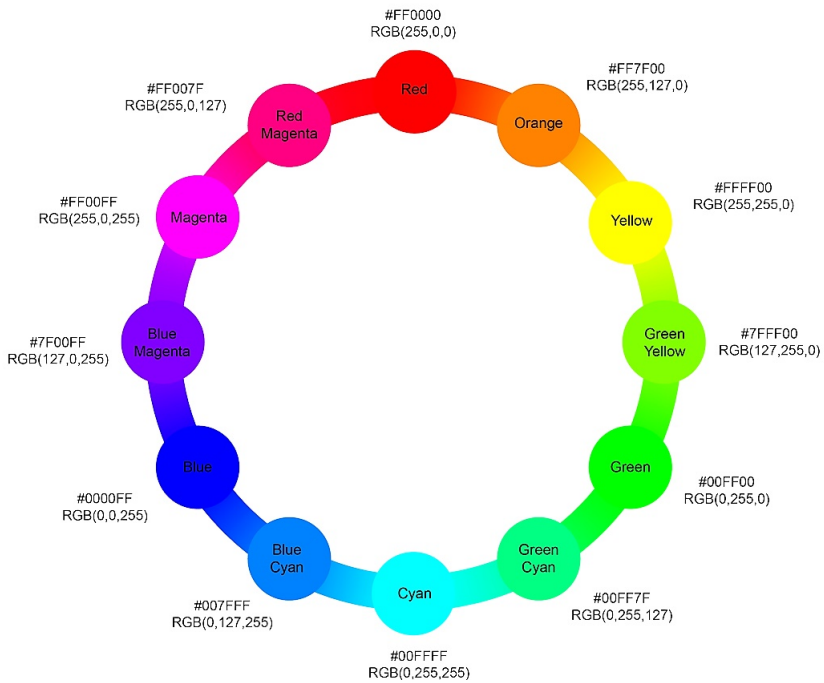
In the previous slide, colors are defined by 6-digit **HEX** (hexadecimal) codes.

- The six digits indicate the color.
  - Black is #000000 and white is #FFFFFF.
- We can add two digits at the end, called alpha, to encode the degree of transparency or opacity.

RGB color codes

Colors can also be represented using **RGB** (red-green-blue) codes.

- An additive color model used for screens.
- Each code is specified with three parameters, defining the intensity of the color as an integer between 0 and 255.
- `rgb(0, 0, 0)` is black.
- `rgb(255, 255, 255)` is white.

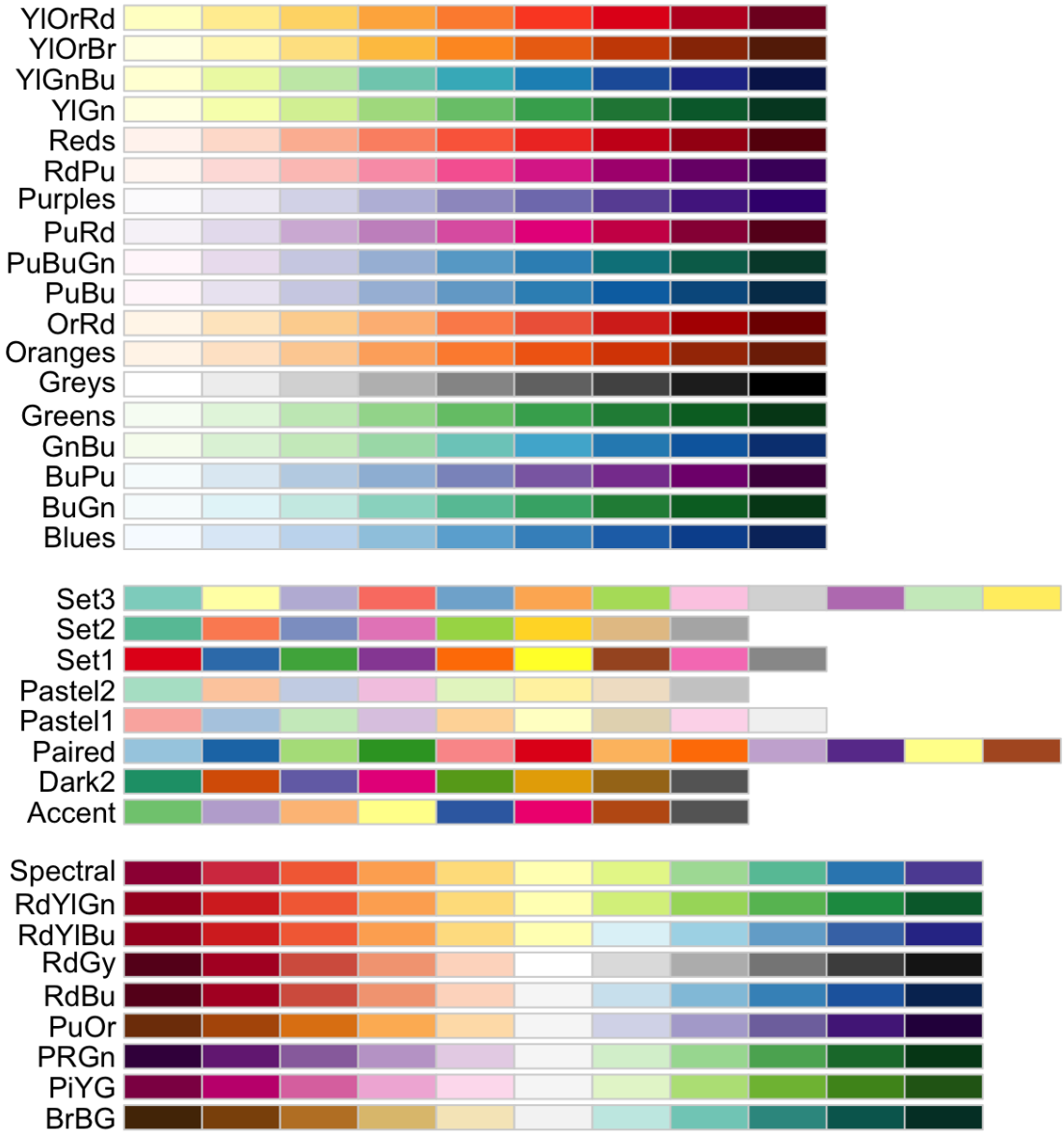


Using color packages

Most visualization packages, like `ggplot2`, provide their own color palettes. There is also a large number of `R` packages that supply additional color support.

- `viridis` can be perceived by readers with the most common form of color blindness.
- `RColorBrewer` provides a vibrant color palettes that are also widely used in the `R` community.

RColorBrewer palettes



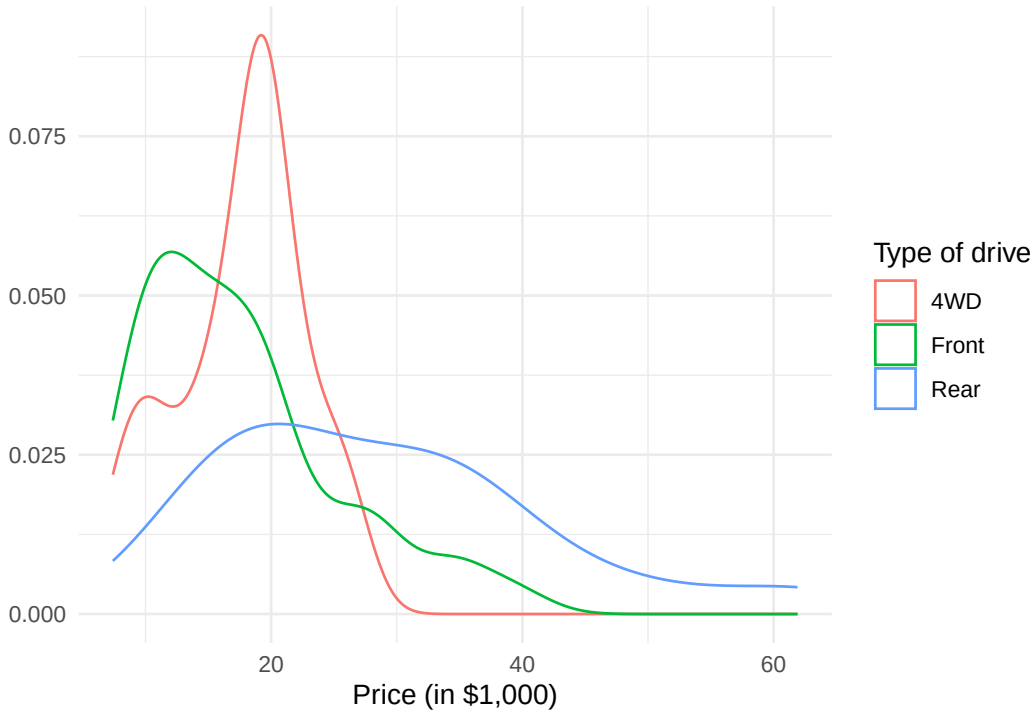
Using color with `ggplot2`

- Specify a single color to be applied to a `geom`:
  - Use `color` or `fill` to a specific color.
- Assign colors by categories:
  - Map the argument `color` or `fill` to the variable of interest.
- Set custom color palettes:
  - `scale_*_manual()`
- Set a gradient color for continuous variable(s):
  - Use `scale_*_gradient()` for sequential gradients between two colors.
- Use color packages such as `viridis` and `RColorBrewer`:
  - Use `scale_*_viridis()` and `scale_*_brewer()`.

Example

Let us use the `Cars93` data set from the `MASS` package. The data contain 27 variables on 93 cars. Here is a scatter plot between the weight of the car and city MPG.

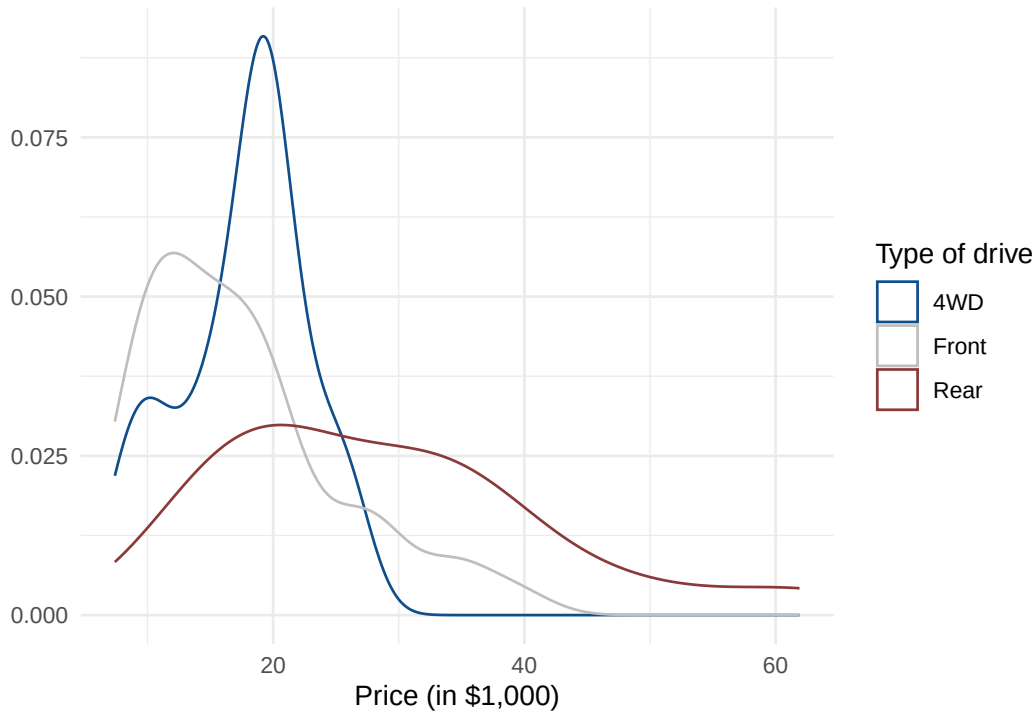
```
# install.packages("MASS")
data(Cars93, package = "MASS")
p1 = ggplot(data = Cars93, aes(x = Price)) +
  geom_density(aes(color = DriveTrain)) +
  labs(x = "Price (in $1,000)", y = "", color = "Type of drive") +
  theme_minimal()
p1
```



Manually set colors

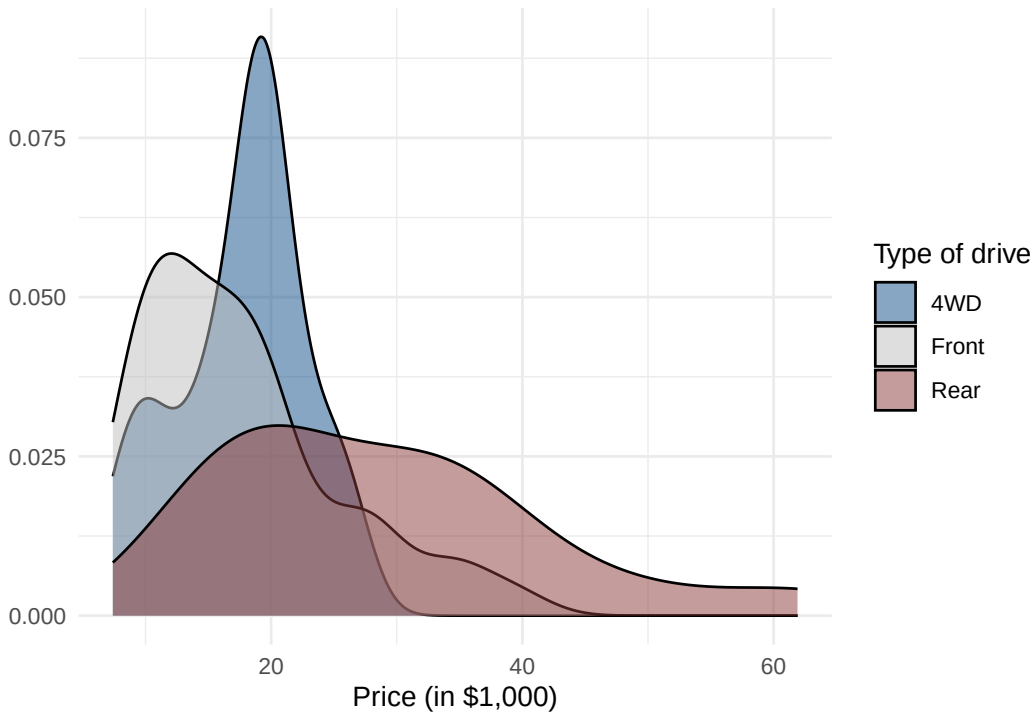
- Here we have specified a `color` aesthetic. If we want to custom colors, we can use the `scale_color_manual()` function:

```
p1 + scale_color_manual(values = c("4WD" = "dodgerblue4", "Front" = "grey",
                                   "Rear" = "indianred4"))
```



- For this density plot, specifying `fill` is probably nicer than specifying `color`. Let us try that.

```
p2 = ggplot(data = Cars93, aes(x = Price)) +
  geom_density(aes(fill = DriveTrain), alpha = 0.5) +
  scale_fill_manual(values = c("4WD" = "dodgerblue4", "Front" = "grey",
                               "Rear" = "indianred4")) +
  labs(x = "Price (in $1,000)", y = "", fill = "Type of drive") +
  theme_minimal()
p2
```



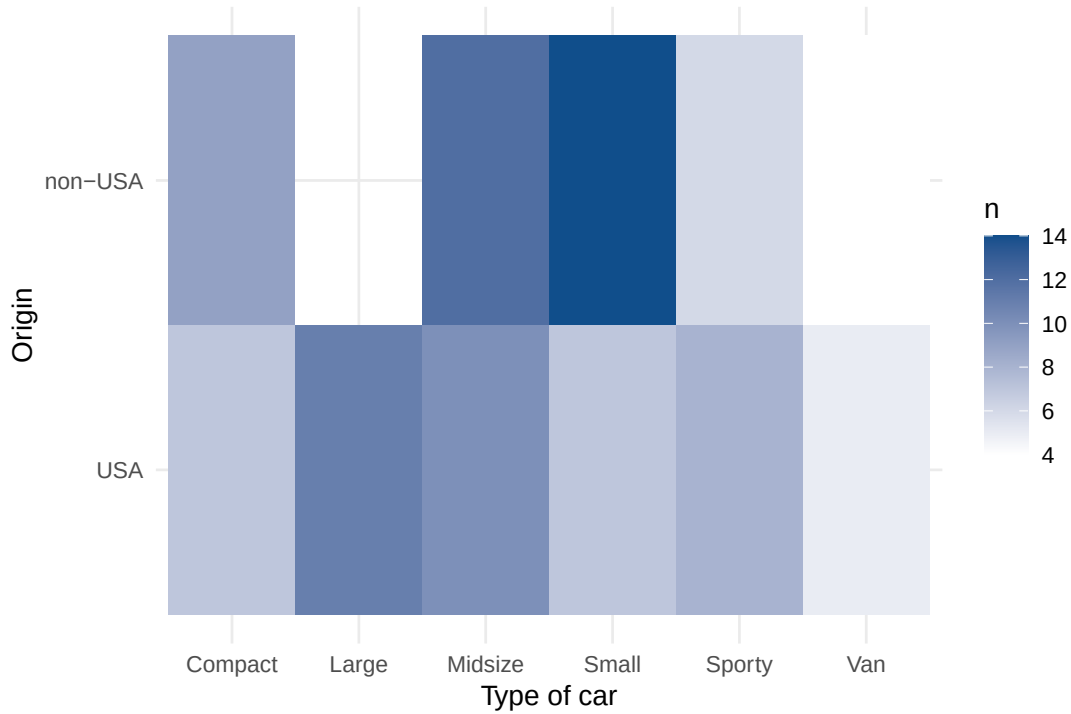
If we want to use the `RColorBrewer` palettes, use a `scale_color_brewer` function:

- The `type` argument specifies the type of color palettes; the `palette` argument specifies the name of the palette.

```
library(RColorBrewer)
p2 + scale_fill_brewer(type = "qual", palette = "Set3")
```

- The previous examples are useful when mapping `fill` to a categorical variable.
- We can also map `fill` to a numeric variable, using **gradient** in coloring.

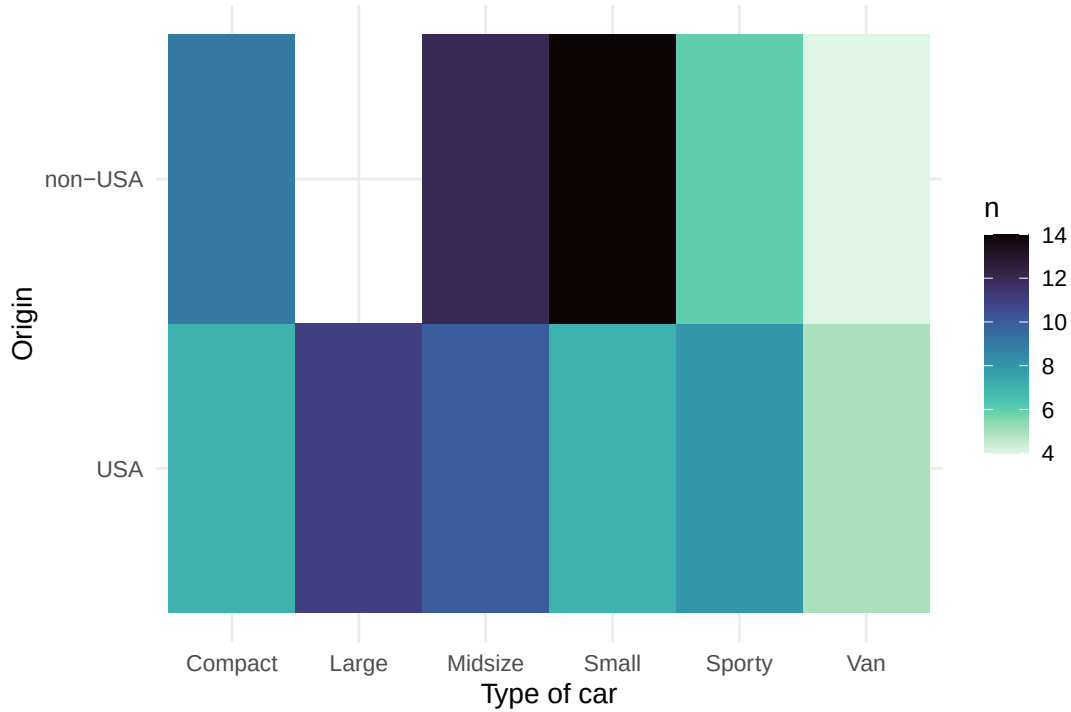
```
p3 = Cars93 %>% count(Type, Origin) %>%
  ggplot(aes(x = Type, y = Origin)) +
  geom_tile(aes(fill = n)) +
  scale_fill_gradient(low = "white", high = "dodgerblue4", na.value = NA) +
  labs(x = "Type of car", y = "Origin") +
  theme_minimal()
p3
```



Use the `viridis` color package

- Lastly, let us use a `viridis` palette.

```
library(viridis)
p3 + scale_fill_viridis(discrete = FALSE, option = "mako", direction = -1)
```



Representing correlations

Correlation matrix is a matrix that represent pair-wise correlation between variables.

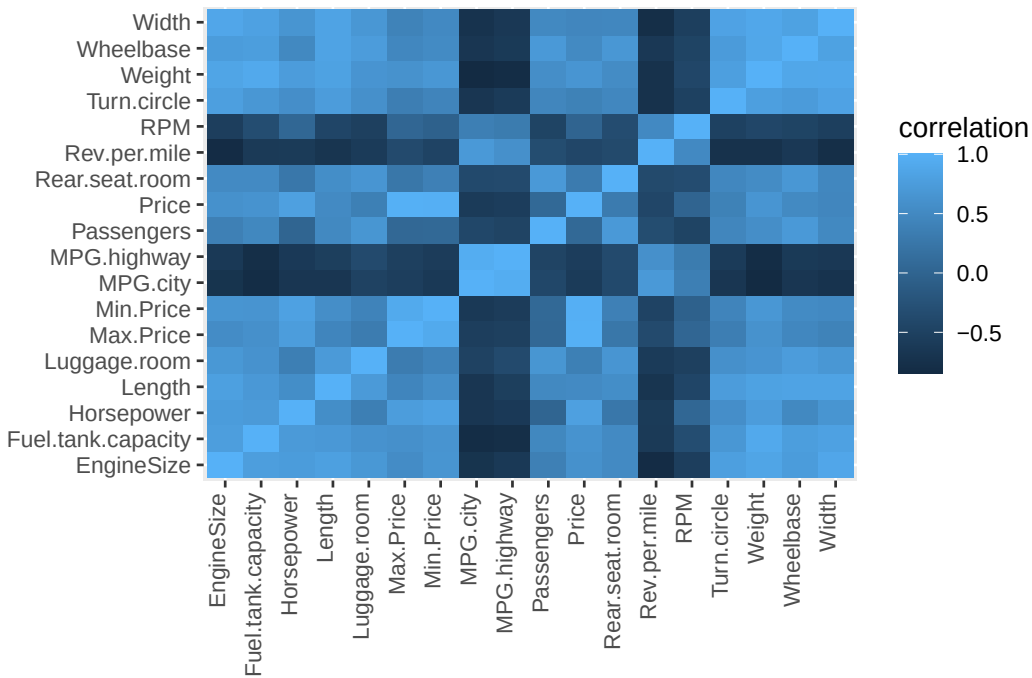
- We can use a **heat map** to visualize pair-wise correlation. In technical terms, the correlation is mapped to the `fill` aesthetic.
- Let us first compute the pair-wise correlation between continuous variables in the `Cars93` data set.

```
# Correlation matrix
corr_cars93 = select(Cars93, which(!sapply(Cars93, is.factor))) %>%
  cor(use = "pair")

# Prepare data for plotting
corr_df = data.frame(corr_cars93, row.names = NULL) %>%
  mutate(var1 = row.names(corr_cars93)) %>%
  gather(Min.Price:Weight, key = "var2", value = "correlation")
```

- Then let us map the pair-wise correlation to the `fill` aesthetics.

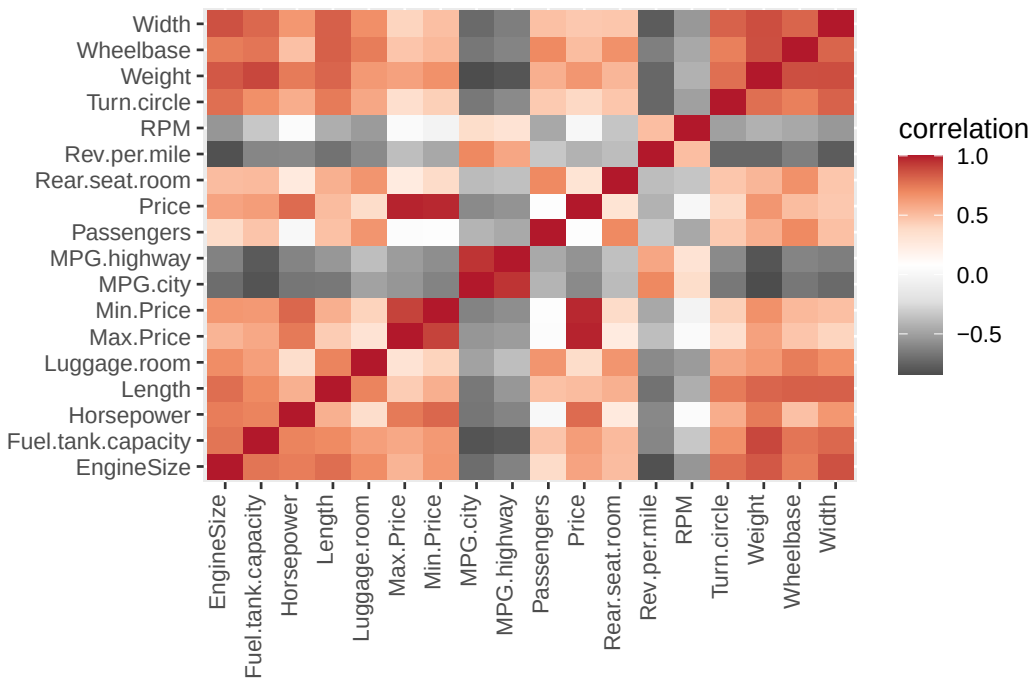
```
p4 = ggplot(corr_df) +
  geom_tile(aes(x = var1, y = var2, fill = correlation)) +
  labs(x = "", y = "") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0, hjust = 1))
p4
```



Use a `RColorBrewer` palette

- Since we are mapping a continuous variable (`correleation`) to the `fill` aesthetics, we need to use the `distiller` variant to use a `RColorBrewer` palette.

```
p4 + scale_fill_distiller(palette = "RdGy")
```

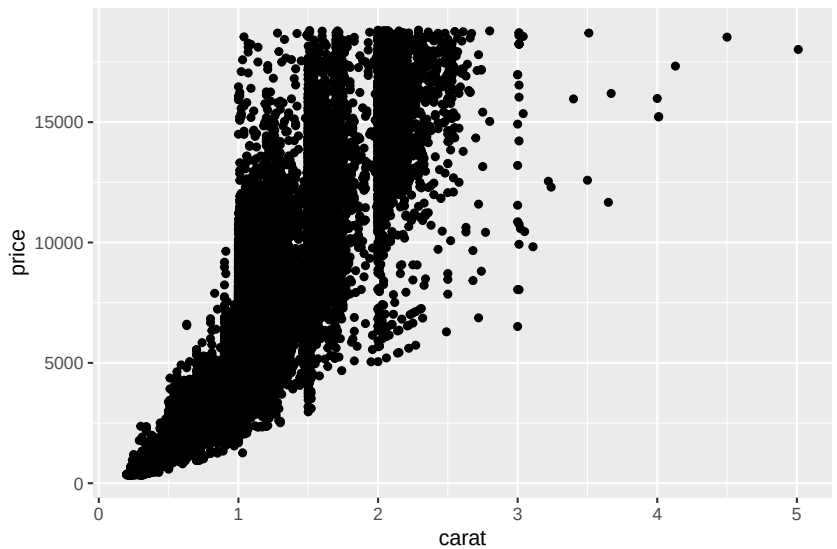


Scatter plot

Scatter plot, or a point `geom`, are used to visualize the relationship between two continuous variables.

- Consider the variables `carat` and `price` from the `diamonds` data set:

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

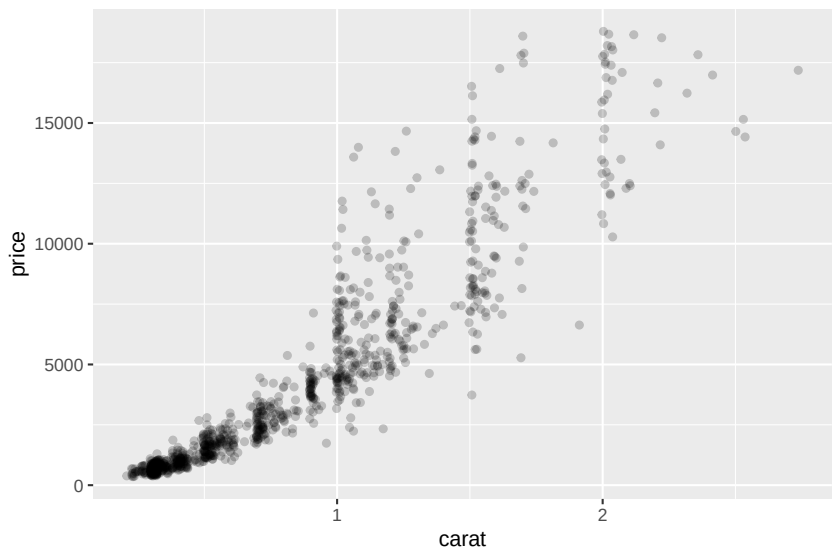


Too many points?

When we run the codes earlier, we find that there are too many points on the chart ⇒ Overplotting.

- To deal with this issue, we learned about using transparent colors and jittering the location of the points.
- Another solution is to randomly sample a set of points and then plot them.

```
set.seed(100)
sub_diamonds = sample_n(diamonds, size = 1000)
ggplot(data = sub_diamonds, aes(x = carat, y = price)) +
  geom_point(alpha = 0.2, position = "jitter")
```



Transformation of data

The relationship between `carat` and `price` does not appear to be linear.

- It is our job as an analyst to find a succinct relationship for the data.
- There are several transformations we can make in order to make the relationship easier to visualize, describe, and compare.
- For the `x` and/or `y`-variable, consider transformations of the form

$$-x^{-2}, -x^{-1}, log(x), x, x^2, x^3$$

Tukey's ladder of transformations

When we begin, we are at the identity transformation.

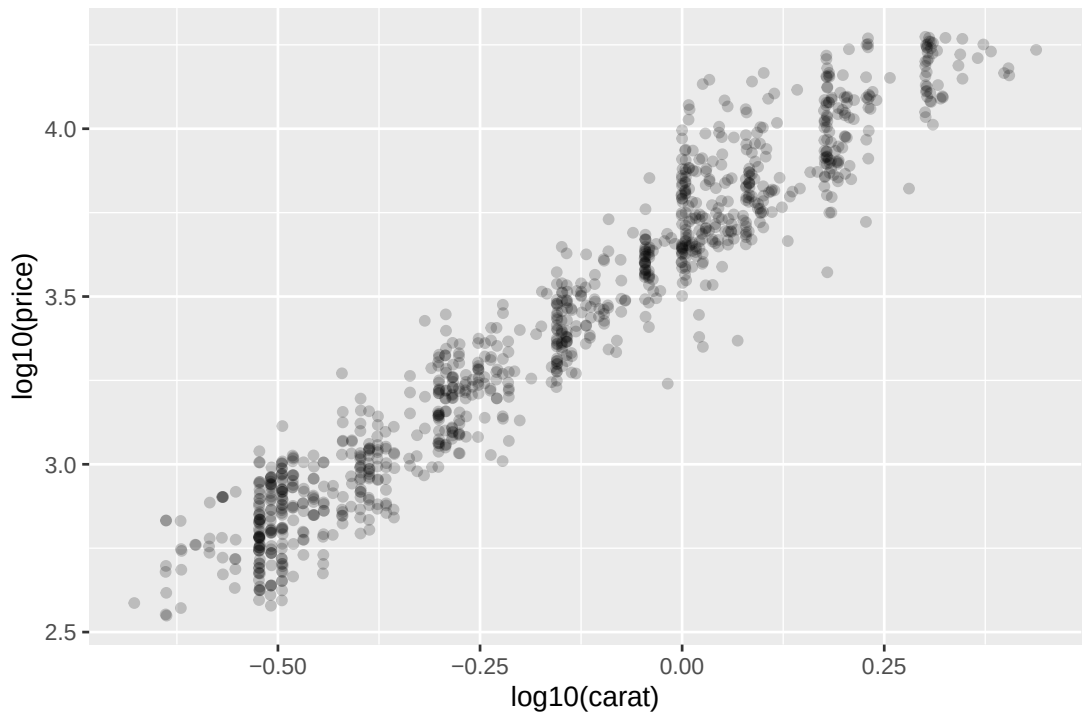
- We shall try different transformations for each variable, and refer to going to the right as "going up the ladder", and the left as "going down the ladder".

λ	-2	-1	-1/2	0	1/2	1	2
y	$\frac{1}{x^2}$	$\frac{1}{x}$	$\frac{1}{\sqrt{x}}$	$logx$	$\sqrt{x}$	x	x^2
Suggested functional form[18]	Inverse quadratic	Inverse	Inverse Square Root	Cobb-Douglas/ Logarithmic	Square Root	linear	quadratic

Source: Tukey 1977, pp. 171-197.

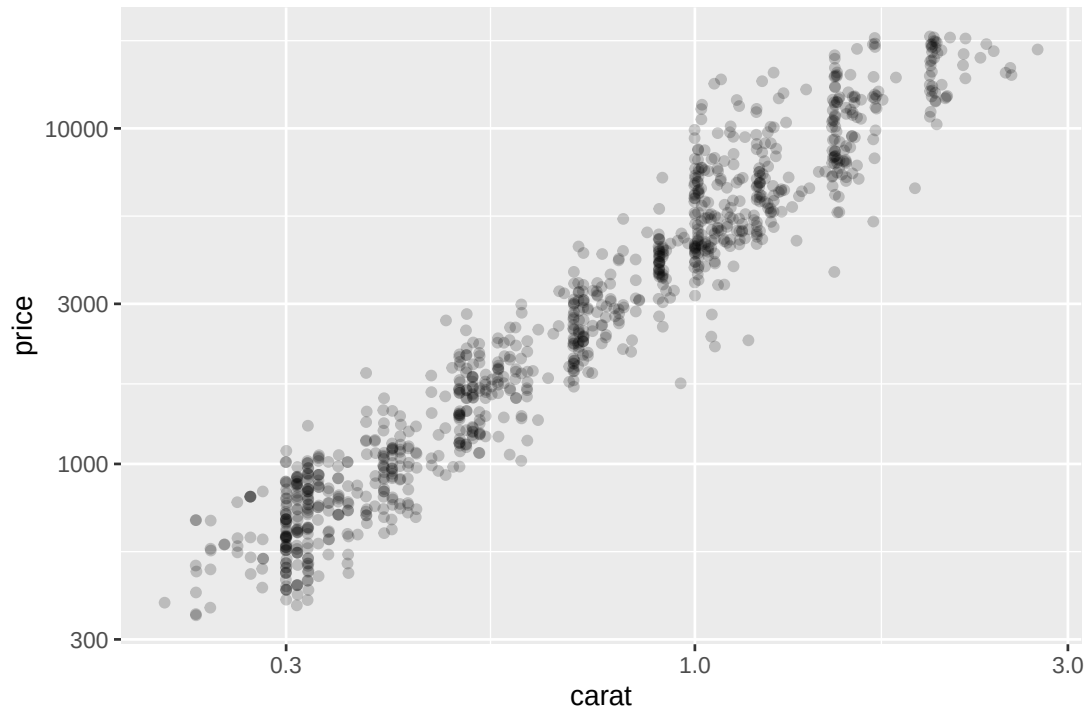
Transformation of data

```
ggplot(data = sub_diamonds, aes(x = log10(carat), y = log10(price))) +
  geom_point(alpha = 0.2, position = "jitter")
```



Better scales

```
ggplot(data = sub_diamonds, aes(x = carat, y = price)) +
  geom_point(alpha = 0.2, position = "jitter") +
  scale_x_log10() +
  scale_y_log10()
```

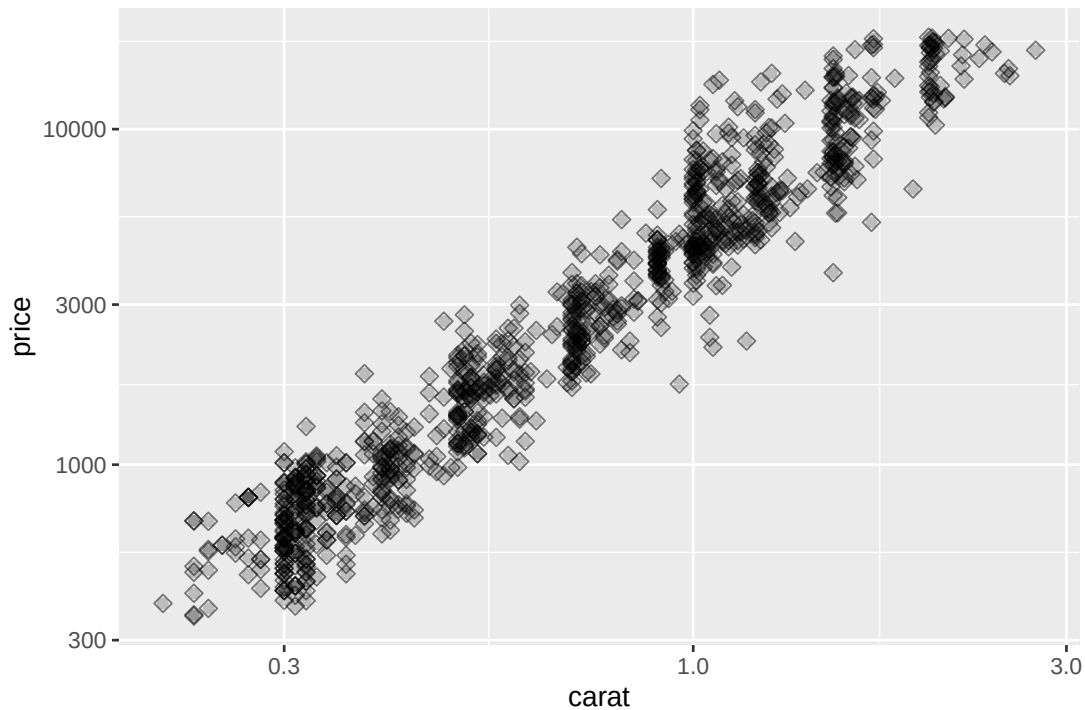


Useful functions in ggplot2

Let us continue using **diamonds** data to review **ggplot2**, building on the scatter plot we just created.

1. Changing size, shape, and alpha.

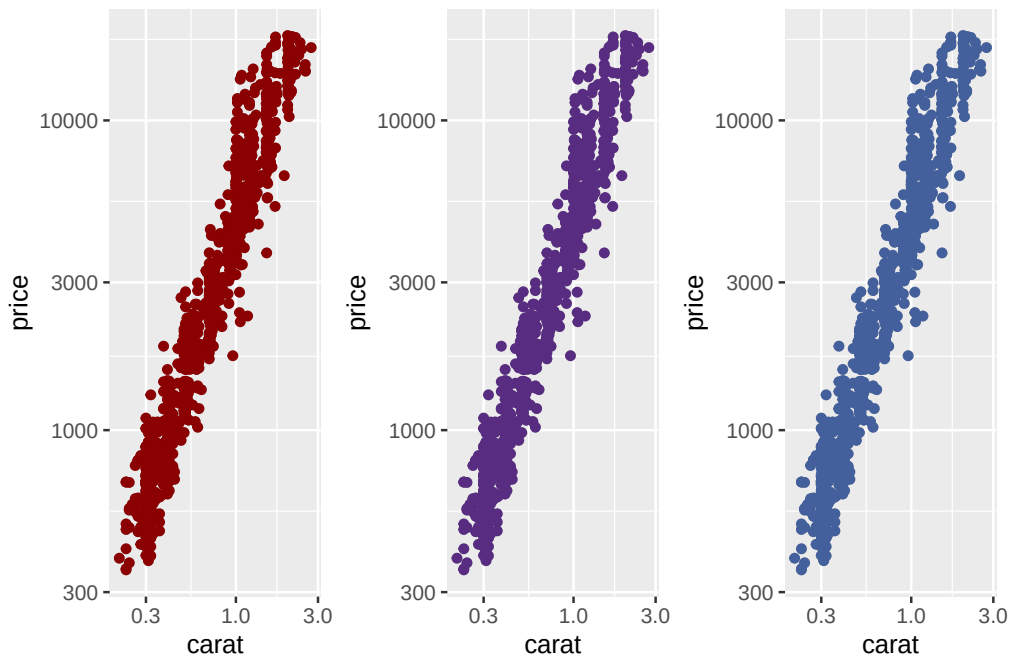
```
p1 = ggplot(data = sub_diamonds, aes(x = carat, y = price)) +
  geom_point(alpha = 0.2, position = "jitter") +
  scale_x_log10() + scale_y_log10()
p1 + geom_point(size = 2, shape = 5, alpha = 0.5)
```



Scatter plot

2. Changing color.

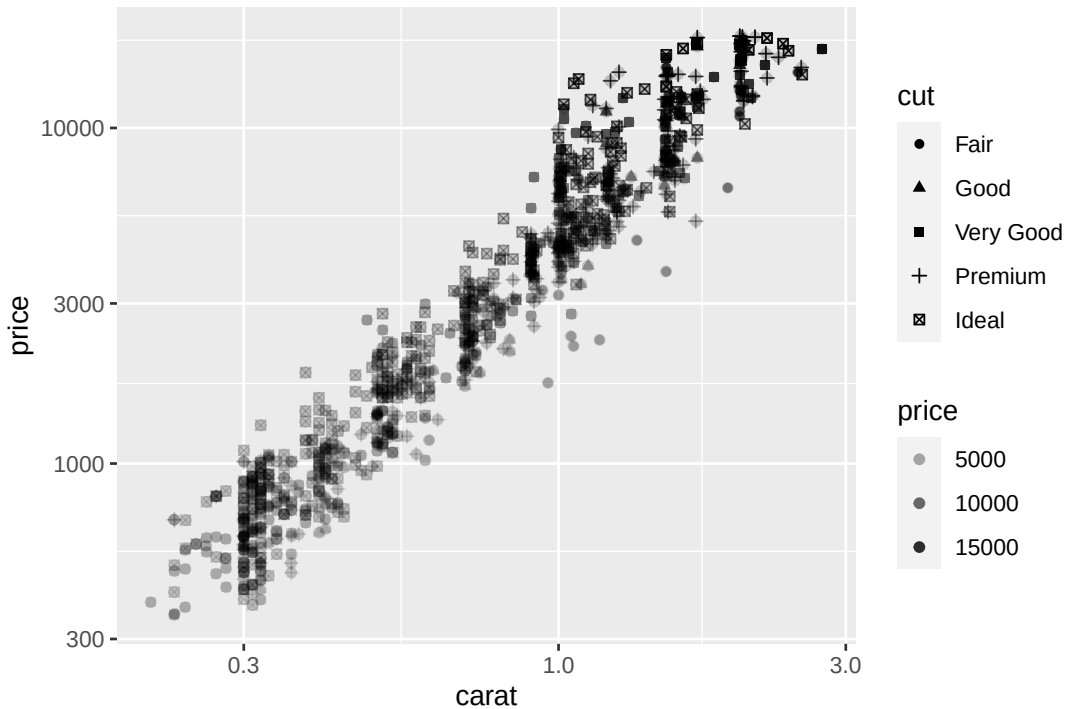
```
p2 = p1 + geom_point(color = "red4")
p3 = p1 + geom_point(color = "#582C81FF")
p4 = p1 + geom_point(color = rgb(67, 96, 156, max = 255))
library(gridExtra)
grid.arrange(p2, p3, p4, nrow = 1)
```



Scatter plot

3. Assigning variables to graphical parameters.

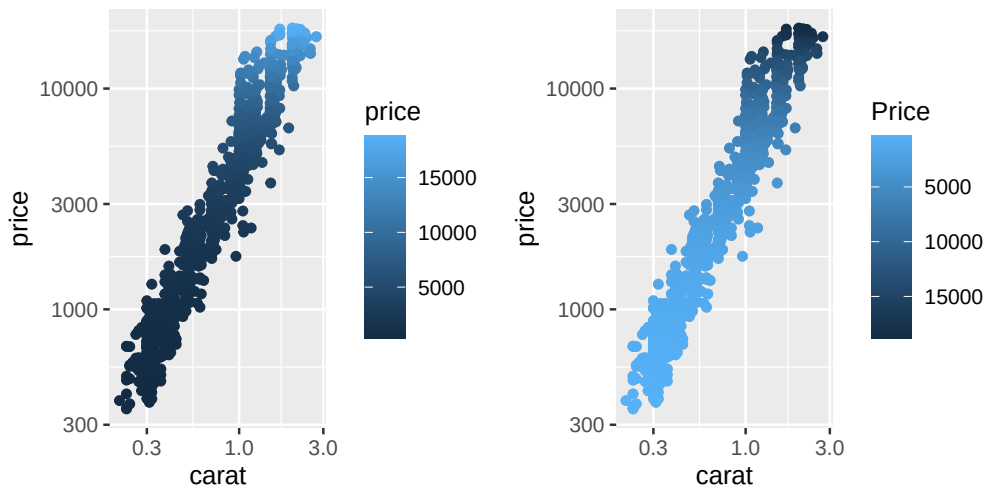
```
p1 + geom_point(aes(shape = cut, alpha = price))
```



Scatter plot

4. Expressing a **continuous** variable using color.

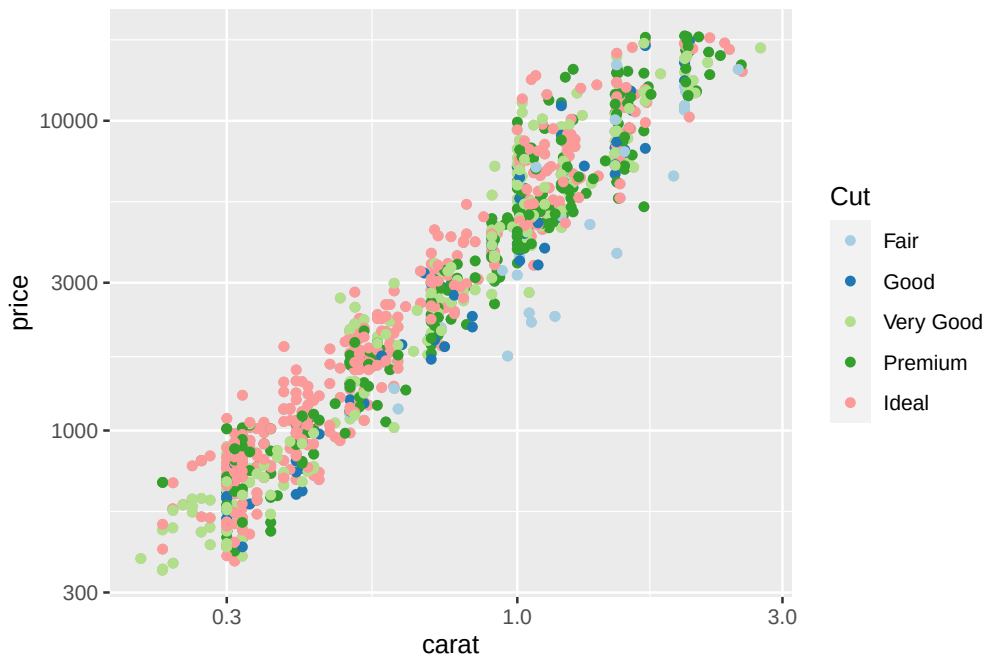
```
p2 = p1 + geom_point(aes(color = price))
p3 = p1 + geom_point(aes(color = price)) +
  scale_color_gradient(name = "Price", trans = "reverse")
grid.arrange(p2, p3, nrow = 1)
```



Scatter plot

5. Expressing a **categorical** variable using color.

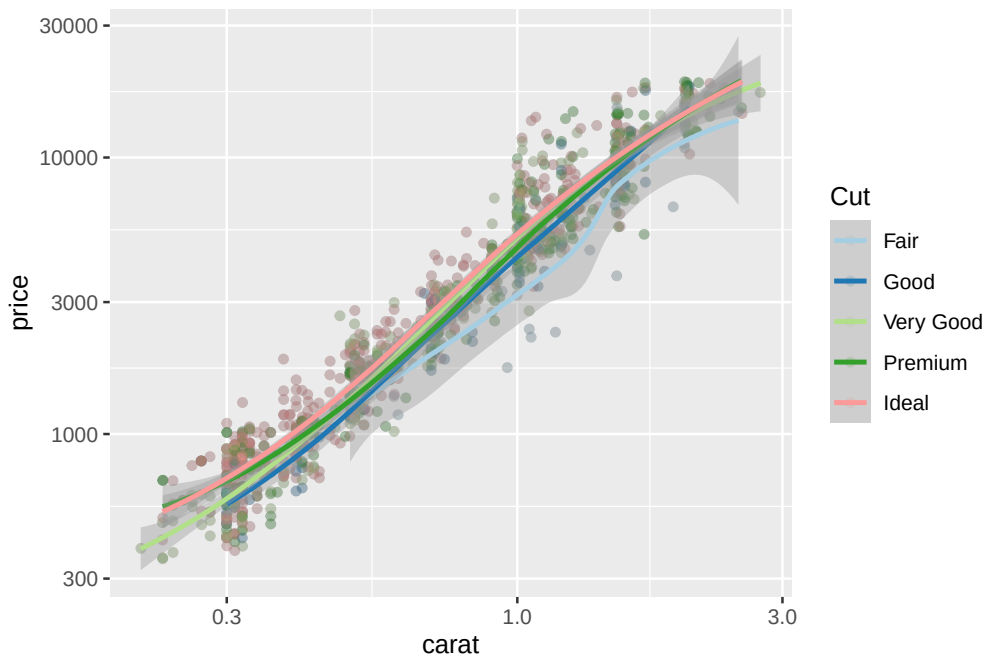
```
library(RColorBrewer)
p1 + geom_point(aes(color = cut)) +
  scale_color_brewer(name = "Cut", palette = "Paired")
```



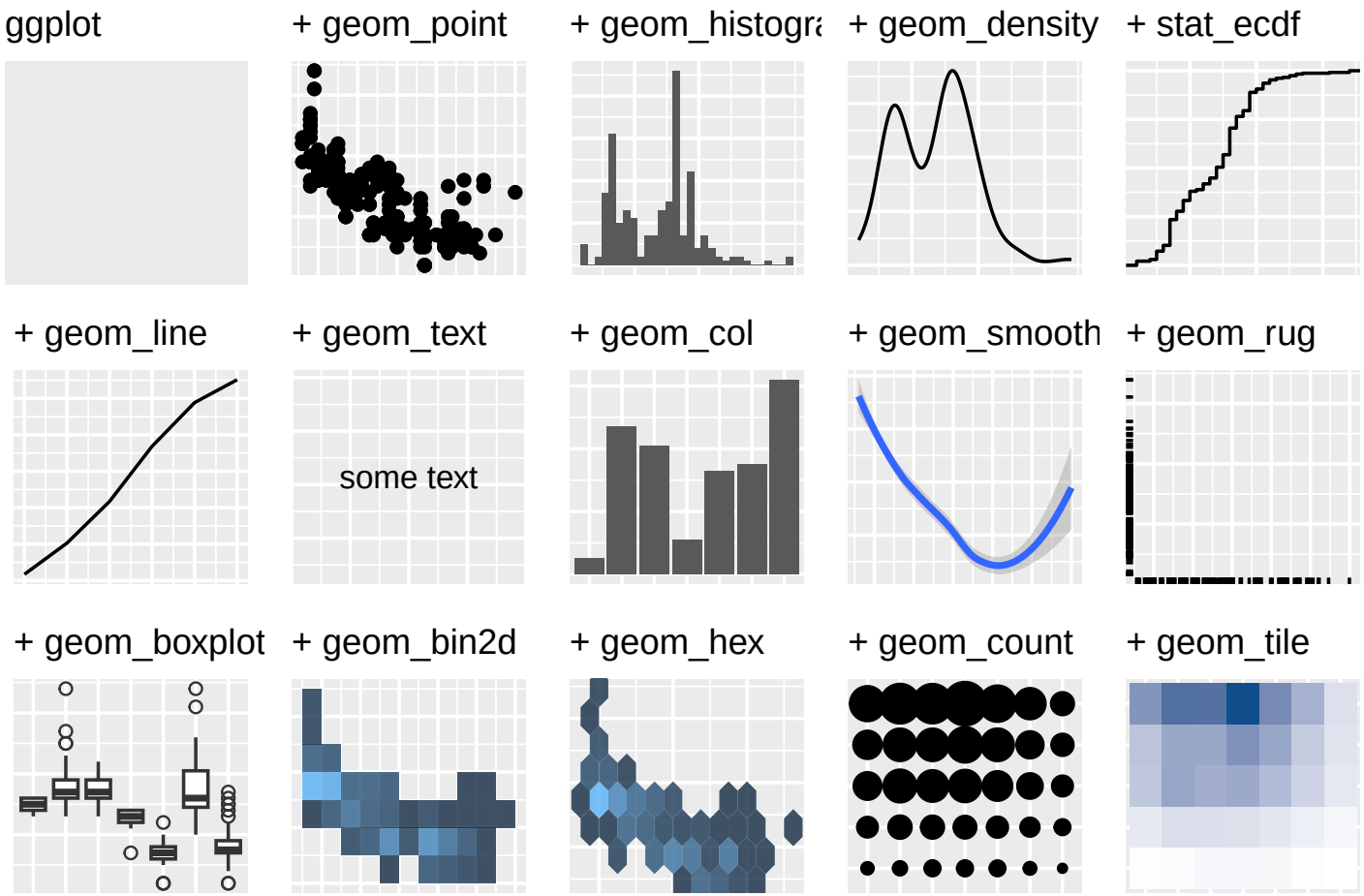
Scatter plot

6. Combining point and smoother geoms.

```
p1 + geom_point(alpha = 0.2, aes(color = cut)) +
  geom_smooth(method = "loess", aes(color = cut)) +
  scale_color_brewer(name = "Cut", palette = "Paired")
```

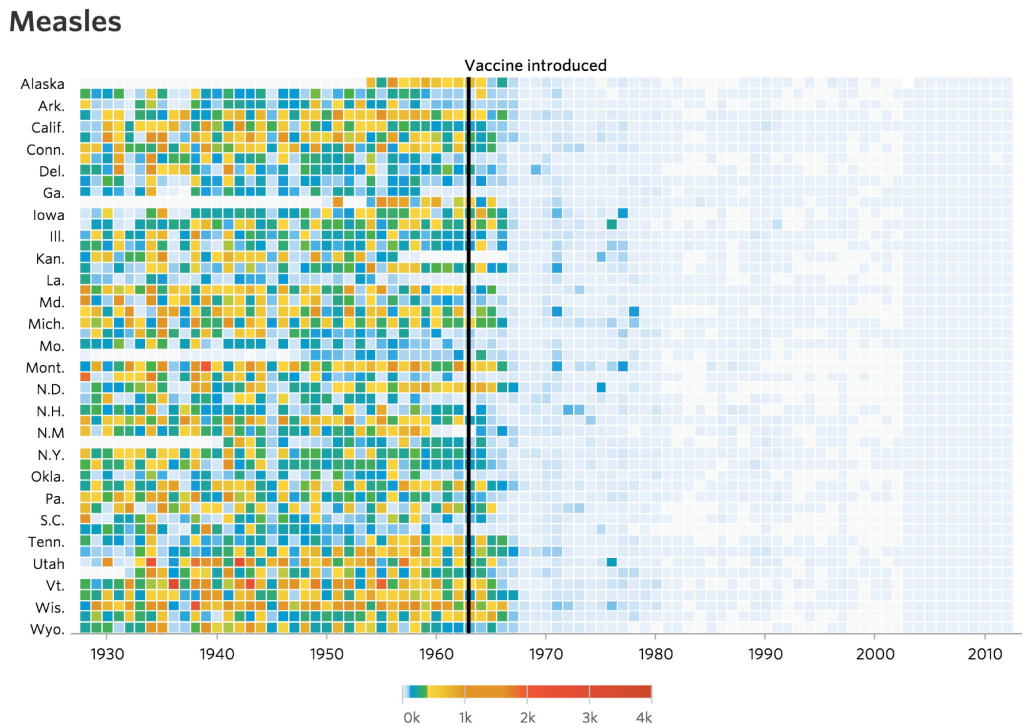


Summary on ggplot2 functions



Vaccines and infecious disease

In this last case study, we reconstruct the graph on measles vaccination in Week 9.



Source: Wall Street Journal, February 11, 2015

Vaccines and infectious disease

The data used for this plot were collected, organized, and distributed by the Tycho Project.

- We can obtain the data from the `dslabs` package.
- Weekly counts of seven diseases from 1928 to 2011 across 50 states and the District of Columbia.

```
library(dslabs)
df2 = us_contagious_diseases
str(df2)
```

```
## 'data.frame':    16065 obs. of  6 variables:
##  $ disease        : Factor w/ 7 levels "Hepatitis A",..: 1 1 1 1 1 1 1 1 1 1
##  $ state          : Factor w/ 51 levels "Alabama","Alaska",..: 1 1 1 1 1 1 1
##  $ year           : num  1966 1967 1968 1969 1970 ...
##  $ weeks_reporting: num  50 49 52 49 51 51 45 45 45 46 ...
##  $ count          : num  321 291 314 380 413 378 342 467 244 286 ...
##  $ population     : num  3345787 3364130 3386068 3412450 3444165 ...
```

Summary of the data

```
summary(df2)
```

```
##         disease             state             year      weeks_reporting
##  Hepatitis A:2346   Alabama   :  315   Min.   :1928   Min.   : 0.00
##  Measles    :3825   Alaska    :  315   1st Qu.:1950   1st Qu.:31.00
##  Mumps      :1785   Arizona   :  315   Median :1975   Median :46.00
##  Pertussis  :2856   Arkansas  :  315   Mean   :1971   Mean   :37.38
##  Polio      :2091   California:  315   3rd Qu.:1990   3rd Qu.:50.00
##  Rubella    :1887   Colorado  :  315   Max.   :2011   Max.   :52.00
##  Smallpox   :1275   (Other)   :14175
##      count              population
##  Min.   :     0   Min.   :   86853
##  1st Qu.:     7   1st Qu.: 1018755
##  Median :    69   Median : 2749249
##  Mean   :  1492   Mean   : 4107584
##  3rd Qu.:   525   3rd Qu.: 4996229
##  Max.   :132342   Max.   :37607525
##                   NA's   :214
```

Data manipulation

Some manipulations before visualization

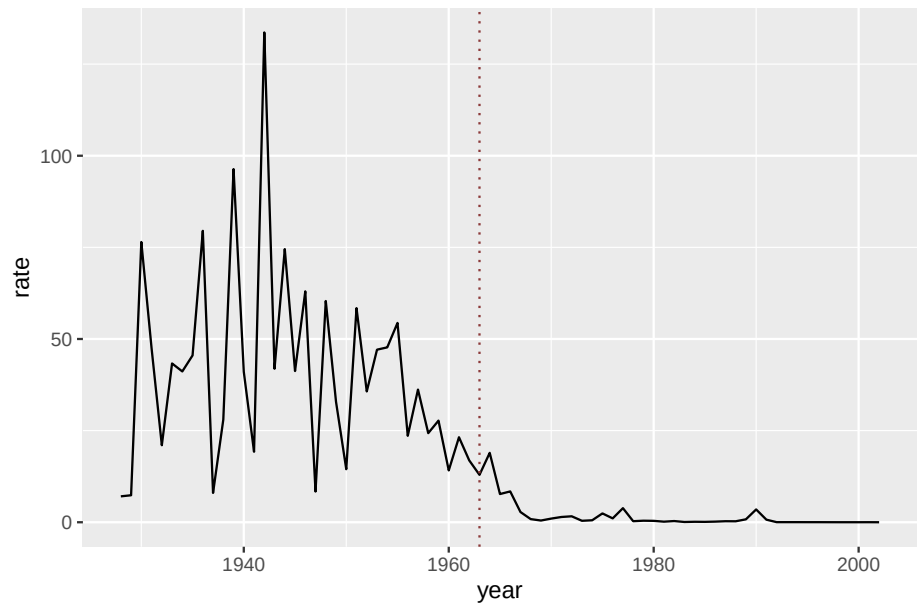
- Missing value on population for Alaska and Hawaii – they only became states in the 1950s ⇒ Remove the two states from our data.
- Restrict our attention to the measles data.
- Compute the case count per 10,000 population.
- Take into account `weeks_reporting` when calculating infection rate.

```
df2 = df2 %>%
  filter(!state %in% c("Hawaii", "Alaska"), disease == "Measles") %>%
  mutate(rate = count / (population / 10000) / weeks_reporting * 52)
```

Time-series line chart

1. Time-series disease rates per year in California
   - A verticle line at 1963 was added – the time when the vaccine was first introduced.

```
df2 %>% filter(state == "California" & !is.na(rate)) %>%
  ggplot(aes(x = year, y = rate)) +
  geom_line() +
  geom_vline(xintercept = 1963, lty = 3, col = "indianred4")
```



Time-series line chart

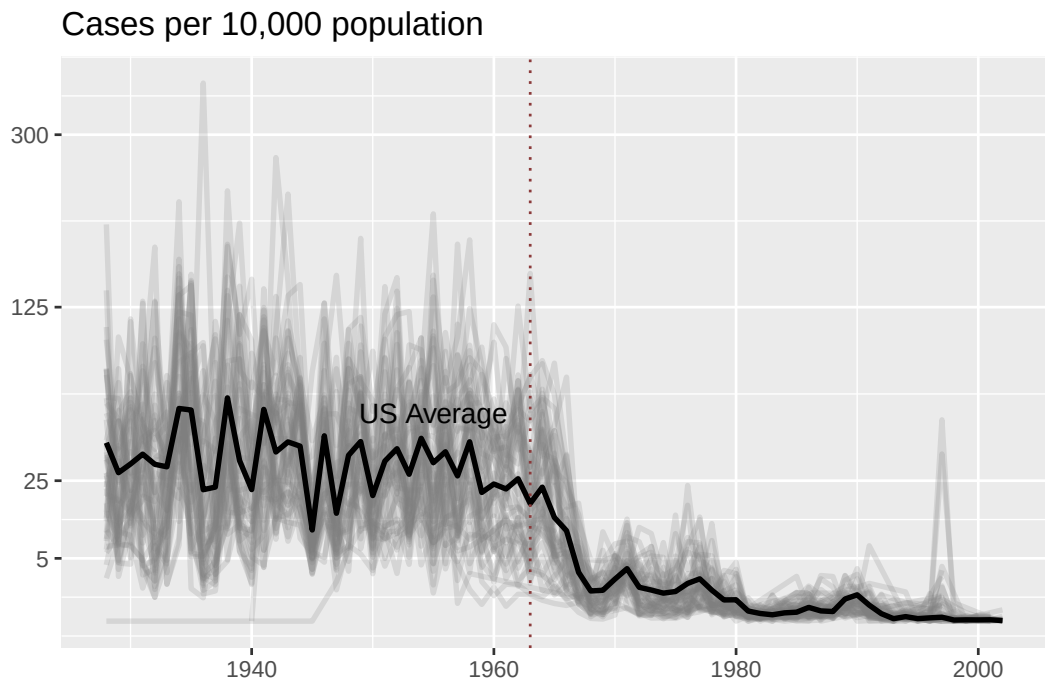
2. Time-series diseases rates per year for all states

```
avg = df2 %>%
  group_by(year) %>%
  summarize(us_avg = sum(count, na.rm = TRUE)/sum(population, na.rm = TRUE)*100

p0 = df2 %>%
  filter(!is.na(rate)) %>%
  ggplot() +
  geom_line(aes(x = year, y = rate, group = state),
            color = "grey50", show.legend = FALSE, alpha = 0.2, size = 1) +
  geom_line(data = avg, mapping = aes(x = year, y = us_avg), size = 1) +
  geom_vline(xintercept = 1963, lty = 3, col = "indianred4") +
  annotate("text", label = "US Average", x = 1955, y = 55, color = "black") +
  scale_y_continuous(breaks = c(5, 25, 125, 300), trans = "sqrt") +
  labs(title = "Cases per 10,000 population", x = "", y = "")
```

Time-series line chart

```
p0
```



Bar chart

3. Bar chart of infection rate across all states in 1967

- Compare the following three graphs:

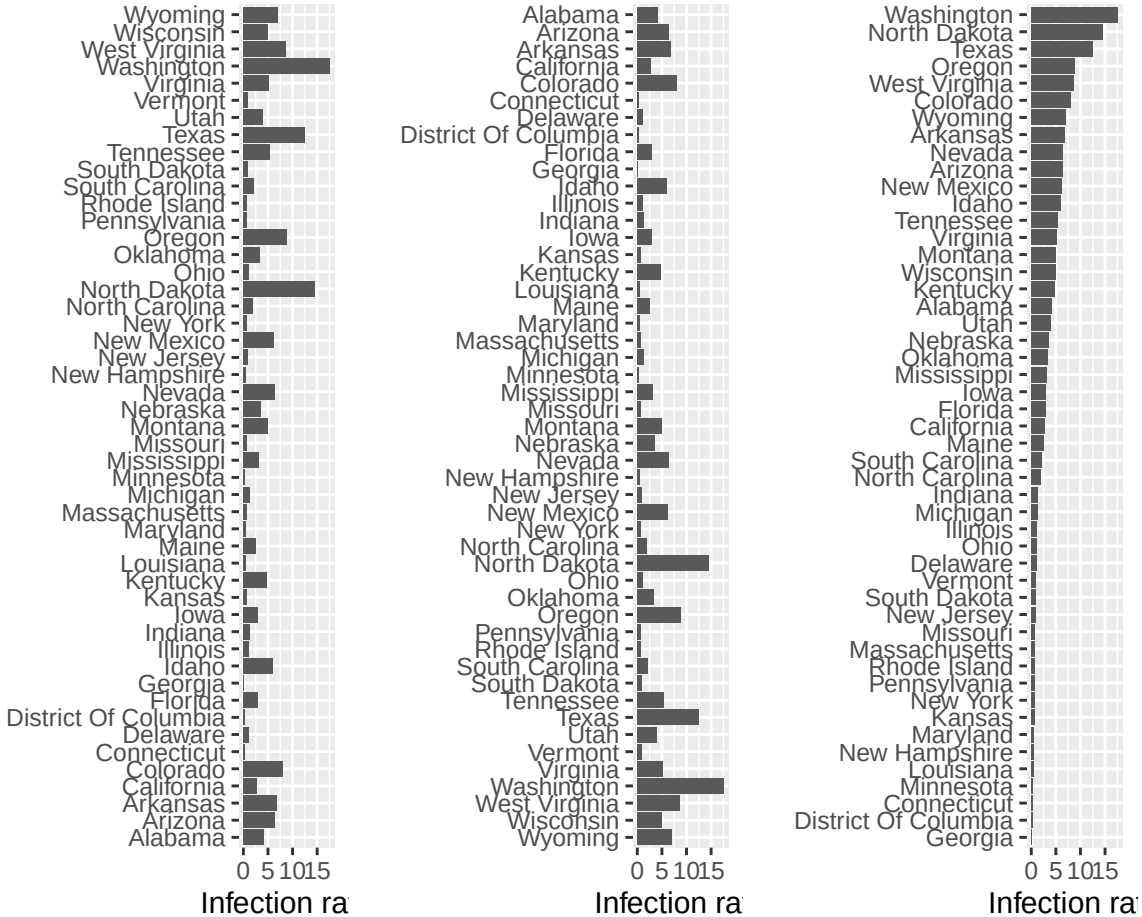
```
df3 = df2 %>%
  filter(year == 1967, !is.na(rate))

p1 = ggplot(data = df3, aes(x = state, y = rate)) +
  geom_col() +
  coord_flip() +
  labs(y = "Infection rate", x = "")

p2 = ggplot(data = df3, aes(x = fct_rev(state), y = rate)) +
  geom_col() +
  coord_flip() +
  labs(y = "Infection rate", x = "")

p3 = df3 %>%
  mutate(state = reorder(state, rate)) %>%
  ggplot(aes(x = state, y = rate)) +
  geom_col() +
  coord_flip() +
  labs(y = "Infection rate", x = "")
```

```
grid.arrange(p1, p2, p3, nrow = 1)
```



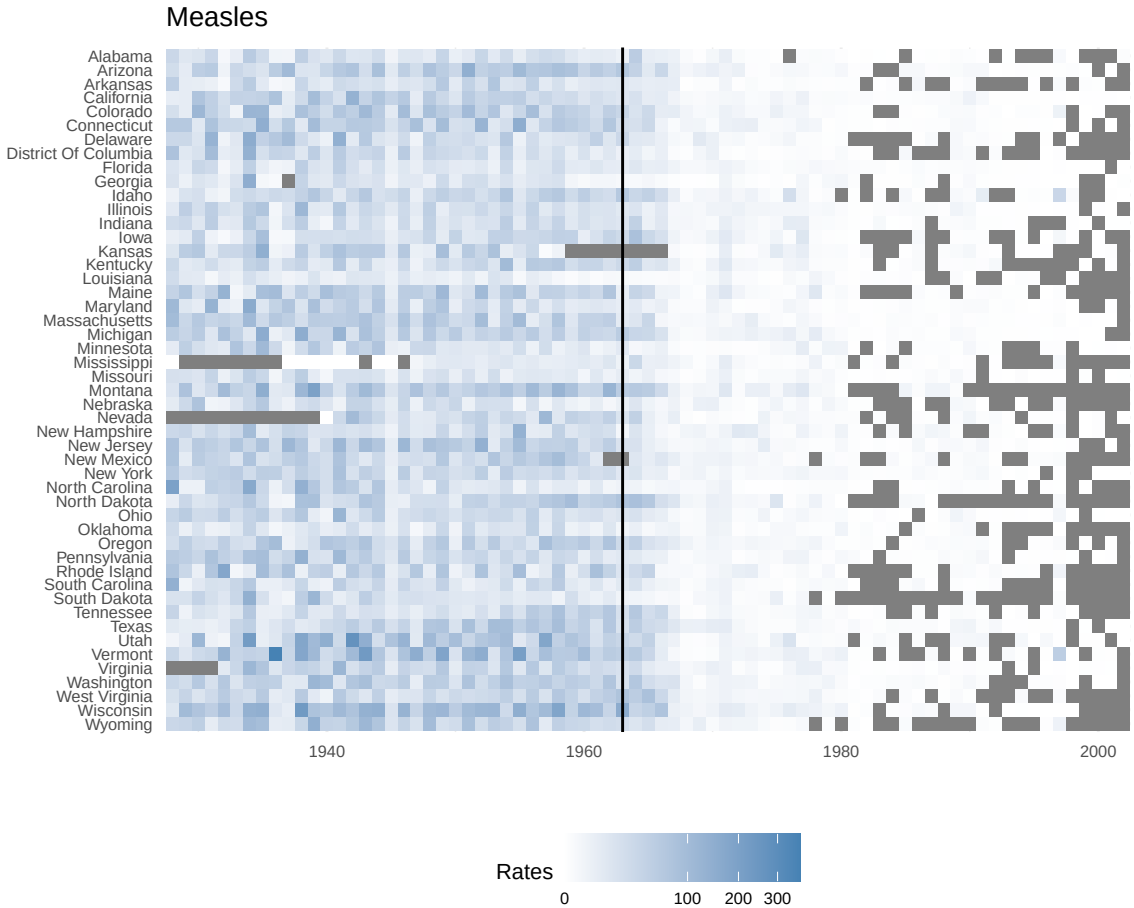
Heatmap

4. A Heatmap to show data for all states in one plot

```
p4 = df2 %>%
  ggplot(aes(x = year, y = fct_rev(state), fill = rate)) +
  geom_tile() +
  geom_vline(xintercept = 1963, col = "black") +
  scale_x_continuous(expand = c(0,0)) +
  scale_fill_gradient(low = "white", high = "steelblue",
                      trans = "sqrt", name = "Rates") +
  labs(title = "Measles", x = "", y = "") +
  theme_minimal() +
  theme(legend.position = "bottom", text = element_text(size = 8))
```

Heatmap

```
p4
```


Measles
Alabama
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District Of Columbia
Florida
Georgia
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
1940
1960
1980
2000
Rates
0
100
200
300