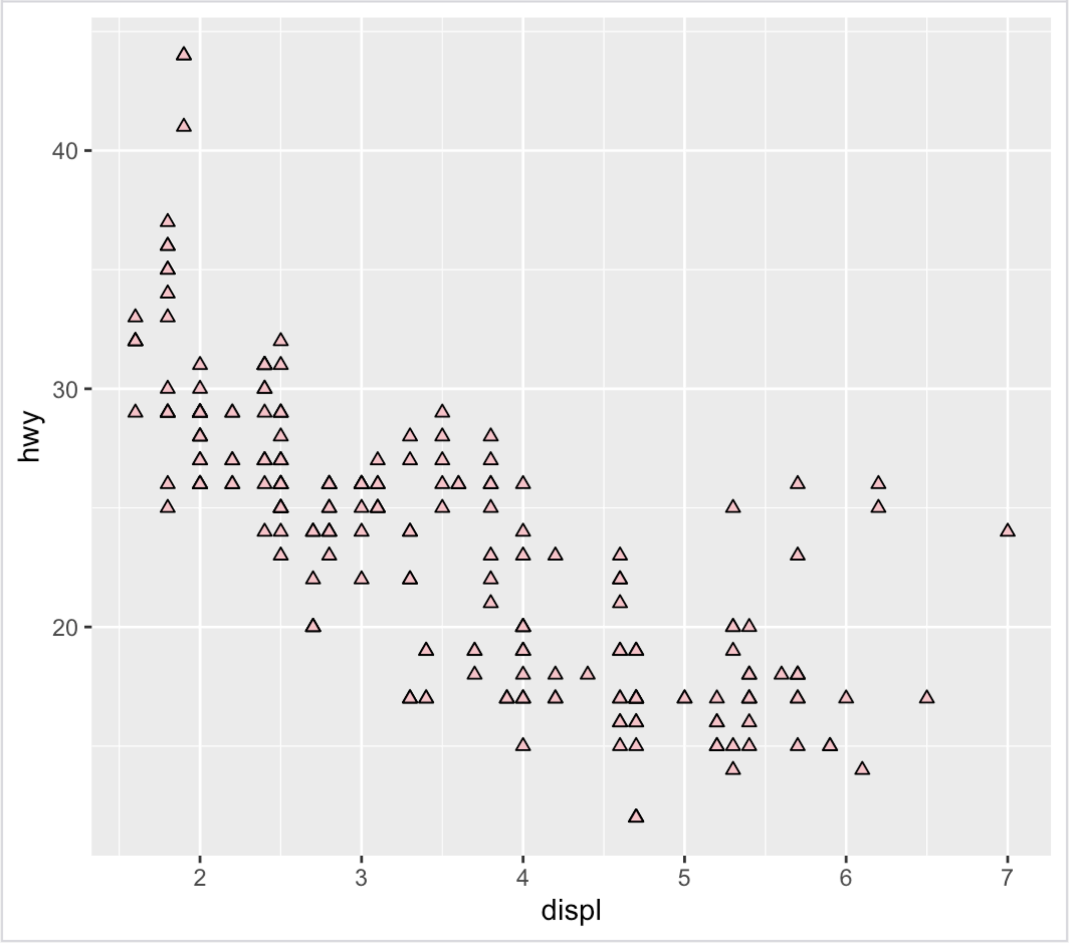
Chapter 9 & 10: Exercises and Executables

9.2.1 Exercises

1. Create a scatterplot of hwy vs. displ where the points are pink filled in triangles.

- ggplot(mpg, aes(x = displ, y = hwy)) +
-   geom_point(shape = 24, fill = "pink", color = "black")



-

2. Why did the following code not result in a plot with blue points?

```
ggplot(mpg) +

  geom_point(aes(x = displ, y = hwy, color = "blue"))
```

- The code sets color = "blue" inside aes() which tells ggplot2 to treat "blue" as a data value rather than a color specification. To directly set the color to blue, you need to place color = "blue" outside aes() so that it's treated as a fixed aesthetic property rather than a data mapping.

3. What does the stroke aesthetic do? What shapes does it work with? (Hint: use ?geom_point)

- The stroke aesthetic in geom_point() is used to control the width of the border around shapes, like changing the border size from 5-10. The shapes can be filled and are outlines hence stroke adjusts the thickness of the outline.

4. What happens if you map an aesthetic to something other than a variable name, like aes(color = displ < 5)? Note, you'll also need to specify x and y.

- If you map an aesthetic to an expression like aes(color = displ < 5), ggplot2 will treat the result (TRUE or FALSE) as a categorical variable. It will then assign different colors to TRUE and FALSE values.

9.3.1 Exercises

1. What geom would you use to draw a line chart? A boxplot? A histogram? An area chart?

- Line Chart: Use geom_line()
- Boxplot: Use geom_boxplot()
- Histogram: Use geom_histogram()
- Area Chart: Use geom_area()

Each geom is designed to visualize data in a specific way and can be customized with aesthetics like color, fill, and size.

2. Earlier in this chapter we used show.legend without explaining it:

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

          geom_smooth(aes(color = drv), show.legend = FALSE)
```

- The show.legend = FALSE option removes the legend for that specific layer in the plot. This helps make the plot less cluttered and clearer if the legend isn't necessary.

What does show.legend = FALSE do here? What happens if you remove it? Why do you think we used it earlier?

- show.legend = FALSE removes the legend for the geom_smooth() layer. If you remove it, the legend will appear, showing how different groups are represented by color. It's useful to reduce clutter or when the legend isn't necessary for understanding the plot.

3. What does the se argument to geom_smooth() do?

- The se argument in geom_smooth() controls whether the shaded confidence interval around the smooth line is shown. If se = TRUE (default), the confidence interval is displayed.

4. Recreate the R code necessary to generate the following graphs. Note that wherever a categorical variable is used in the plot, it's drv.

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

 geom_point(color = "black") +

 geom_smooth(se = FALSE)
```

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point()
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(color = "black") +
  geom_smooth(se = FALSE, show.legend = FALSE)

ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)

ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)

ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE, show.legend = FALSE)
```

9.4.1 Exercises

1. What happens if you facet on a continuous variable?

- If you facet on a continuous variable, ggplot2 will treat it like a categorical variable, creating a separate panel for each value.

2. What do the empty cells in the plot above with facet_grid(drv ~ cyl) mean? Run the following code. How do they relate to the resulting plot?

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

- Empty cells in the plot mean that there are no data points for those combinations of drv and cyl.

3. What plots does the following code make? What does . do?

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(drv ~ .)
```

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```

The code creates faceted plots:

1. facet_grid(drv ~ .) creates a row for each value of drv. The . on the right means no faceting by columns.
2. facet_grid(. ~ cyl) creates a column for each value of cyl. The . on the left means no faceting by rows.

The . acts as a placeholder, telling ggplot2 not to split the plot in that direction.

4. Take the first faceted plot in this section:

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

What are the advantages to using faceting instead of the color aesthetic? What are the disadvantages? How might the balance change if you had a larger dataset?

Advantages:

- Faceting makes it easier to compare patterns across groups.
- It reduces clutter, especially when groups overlap.
- It allows for clearer visualization when there are many categories.

Disadvantages:

- Faceting reduces the size of each individual plot.
- It becomes harder to see fine details when the dataset is large.

With Larger Datasets:

- Faceting is helpful to avoid overplotting but may make individual plots too small to interpret clearly.

5. Read ?facet_wrap. What does nrow do? What does ncol do? What other options control the layout of the individual panels? Why doesn't facet_grid() have nrow and ncol arguments?

- nrow sets the number of rows, and ncol sets the number of columns in the faceted plot. Other options include scales (to control whether axes are fixed or free) and dir (to control the order of panels).
- facet_grid() doesn't have nrow and ncol because it arranges panels based on the levels of the row and column variables.

6. Which of the following plots makes it easier to compare engine size (displ) across cars with different drive trains? What does this say about when to place a faceting variable across rows or columns?

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(drv ~ .)
```

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(. ~ drv)
```

- The plot with facet_grid(. ~ drv) makes it easier to compare engine size (displ) across drive trains because it places the faceting variable (drv) along columns, allowing for easier side-by-side comparison. This shows that using columns is often better for comparing distributions across categories, while rows may work better for highlighting individual patterns.

7. Recreate the following plot using facet_wrap() instead of facet_grid(). How do the positions of the facet labels change?

```
ggplot(mpg) +
 geom_point(aes(x = displ, y = hwy)) +
         facet_grid(drv ~ .)
```

- In facet_grid(), the facet labels appear along the side of the plot (aligned with the rows).
- In facet_wrap(), the facet labels are positioned at the top of each individual panel.
- This makes facet_wrap() more flexible for arranging multiple panels.

9.5.1 Exercises

1. What is the default geom associated with stat_summary()? How could you rewrite the previous plot to use that geom function instead of the stat function?

- The default geom for stat_summary() is geom_pointrange(). To rewrite a plot using the geom instead of the statfunction, you would replace stat_summary() with geom_pointrange() and manually calculate the summary statistics (e.g., mean and confidence interval).

2. What does geom_col() do? How is it different from geom_bar()?

- geom_col() and geom_bar() both create bar charts, but they differ in how they handle data:

- geom_col() requires you to provide both the x and y values, and it directly represents the y values as the height of the bars.
- geom_bar() only requires an x value and calculates the y value (height) by counting the number of observations in each category.

3. Most geoms and stats come in pairs that are almost always used in concert. Make a list of all the pairs. What do they have in common? (Hint: Read through the documentation.)

- Common geom-stat pairs in ggplot2:

1. geom_bar() ↔ stat_count()
2. geom_col() ↔ stat_identity()
3. geom_smooth() ↔ stat_smooth()
4. geom_bin2d() ↔ stat_bin_2d()
5. geom_density_2d() ↔ stat_density_2d()
6. geom_histogram() ↔ stat_bin()
7. geom_pointrange() ↔ stat_summary()

4. What variables does stat_smooth() compute? What arguments control its behavior?

- stat_smooth() computes the following variables:
- y – predicted value
- ymin and ymax – confidence interval bounds
- se – standard error
- The method argument controls the smoothing method (e.g., "lm" for linear model). The se argument decides whether to display the confidence interval. The level argument sets the confidence level (e.g., 0.95 for 95%). The span argument controls the smoothness of loess curves — a larger value creates a smoother curve.

5. In our proportion bar chart, we needed to set group = 1. Why? In other words, what is the problem with these two graphs?

```
ggplot(diamonds, aes(x = cut, y = after_stat(prop))) +

  geom_bar()

ggplot(diamonds, aes(x = cut, fill = color, y = after_stat(prop))) +

          geom_bar()
```

- Setting group = 1 ensures that ggplot2 treats the whole bar as a single group when calculating proportions. Without it, ggplot2 tries to compute proportions separately for each level of the fill variable, leading to incorrect bar heights.

9.7.1 Exercises

1. Turn a stacked bar chart into a pie chart using coord_polar().

- To convert a stacked bar chart into a pie chart, you need to use coord_polar(). First, create a bar chart with geom_bar(). Then, apply coord_polar(theta = "y") to map the y-values to the radius, creating a circular pie chart. The x = factor(1) ensures that the bars stack into a single column, which coord_polar() then transforms into a circle.

2. What's the difference between coord_quickmap() and coord_map()?

- coord_quickmap() quickly sets up a map projection that keeps shapes close to their true proportions but sacrifices some accuracy. coord_map() is more accurate for mapping but takes longer to process because it applies a more complex projection.

3. What does the following plot tell you about the relationship between city and highway mpg? Why is coord_fixed() important? What does geom_abline() do?

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_fixed()
```

- The plot shows a positive relationship between city (cty) and highway (hwy) mpg — cars that perform well in the city also tend to perform well on highways.

- **coord_fixed()** keeps the aspect ratio 1:1, so equal units on both axes are the same length, making comparisons easier.
- **geom_abline()** adds a reference line (usually a 45-degree line) to help see if points align with an equal relationship between city and highway mpg.

10.3.3 Exercises

1. Explore the distribution of each of the x, y, and z variables in diamonds. What do you learn? Think about a diamond and how you might decide which dimension is the length, width, and depth.

- In the distribution of the x, y, and z variables in the diamonds dataset, you can create separate histograms for each variable. This helps reveal typical values and any unusual patterns.

2. Explore the distribution of price. Do you discover anything unusual or surprising? (Hint: Carefully think about the binwidth and make sure you try a wide range of values.)

- To explore the price distribution in the diamonds dataset, you can create a histogram. Adjusting the binwidth helps uncover patterns like spikes or gaps, which might suggest market pricing strategies or data issues. A small binwidth shows finer details, while a larger binwidth reveals overall trends. Unusual patterns may reflect popular price points or rounding in the data.

3. How many diamonds are 0.99 carat? How many are 1 carat? What do you think is the cause of the difference?

- 23 diamonds are 0.99 carat and 1558 diamonds weigh 1 carat. The difference is likely due to market pricing strategies. A 1 carat diamond is more desirable and valuable, so manufacturers may cut diamonds to reach the 1 carat mark, making them more common than 0.99-carat diamonds.

4. Compare and contrast coord_cartesian() vs. xlim() or ylim() when zooming in on a histogram. What happens if you leave binwidth unset? What happens if you try and zoom so only half a bar shows?

- coord_cartesian() zooms into the plot without changing the data, so bins remain the same size. xlim() and ylim()remove data outside the limits, which can alter the shape of the histogram. If binwidth is unset, ggplot2 picks a default value. If you zoom in so only half a bar shows, coord_cartesian() displays the partial bar, but xlim() and ylim() may remove it entirely.

10.4.1 Exercises

1. What happens to missing values in a histogram? What happens to missing values in a bar chart? Why is there a difference in how missing values are handled in histograms and bar charts?

- A histogram, missing values are removed and not shown. In a bar chart, missing values are displayed as an "NA" category if the variable is categorical. The difference is because histograms work with continuous data, where missing values don't have a logical position, while bar charts handle categorical data, so "NA" can be treated as a valid category.

2. What does na.rm = TRUE do in mean() and sum()?

- The na.rm = TRUE argument in mean() and sum() tells R to ignore missing values (NA) when calculating the result.

3. Recreate the frequency plot of scheduled_dep_time colored by whether the flight was cancelled or not. Also facet by the cancelled variable. Experiment with different values of the scales variable in the faceting function to mitigate the effect of more non-cancelled flights than cancelled flights.

- ggplot(flights, aes(x = scheduled_dep_time, fill = cancelled)) +
- geom_histogram(binwidth = 10) +
- facet_wrap(~ cancelled, scales = "free_y")

10.5.1.1 Exercises

1. Use what you've learned to improve the visualization of the departure times of cancelled vs. non-cancelled flights.

- To improve the visualization of departure times for cancelled vs. non-cancelled flights, you can:

- Adjust the binwidth to control the level of detail.
- Use facet_wrap(~ cancelled) to separate cancelled and non-cancelled flights.
- Set scales = "free_y" to allow different y-axis scales for better comparison.
- Adjust transparency (alpha) to handle overlapping data points.

2. Based on EDA, what variable in the diamonds dataset appears to be most important for predicting the price of a diamond? How is that variable correlated with cut? Why does the combination of those two relationships lead to lower quality diamonds being more expensive?

- The carat variable is the most important predictor of diamond price — larger diamonds are generally more expensive. Carat and cut are negatively correlated because larger diamonds tend to have lower cut quality. This happens because large, lower-quality diamonds are often priced higher due to their size, even if the cut is poor.

3. Instead of exchanging the x and y variables, add coord_flip() as a new layer to the vertical boxplot to create a horizontal one. How does this compare to exchanging the variables?

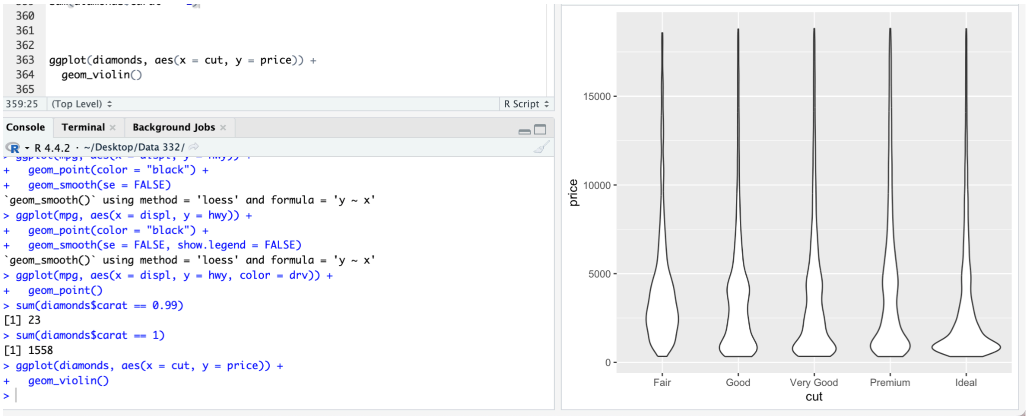
- ggplot(mpg, aes(x = drv, y = hwy)) +
- geom_boxplot() +
- coord_flip()
- coord_flip() keeps the original variable mapping but rotates the plot.
- Switching x and y in aes() changes the actual variable mapping, which might affect other layers.

4. One problem with boxplots is that they were developed in an era of much smaller datasets and tend to display a prohibitively large number of "outlying values". One approach to remedy this problem is the letter value plot. Install the lvplot package, and try using geom_lv() to display the distribution of price vs. cut. What do you learn? How do you interpret the plots?

- Letter value plots are better for large datasets because they summarize extreme values more effectively.
- Unlike boxplots, they display deeper quantiles, helping to reveal the full distribution of price vs. cut.

5. Create a visualization of diamond prices vs. a categorical variable from the diamonds dataset using geom_violin(), then a faceted geom_histogram(), then a colored geom_freqpoly(), and then a colored geom_density(). Compare and contrast the four plots. What are the pros and cons of each method of visualizing the distribution of a numerical variable based on the levels of a categorical variable?

- geom_violin()



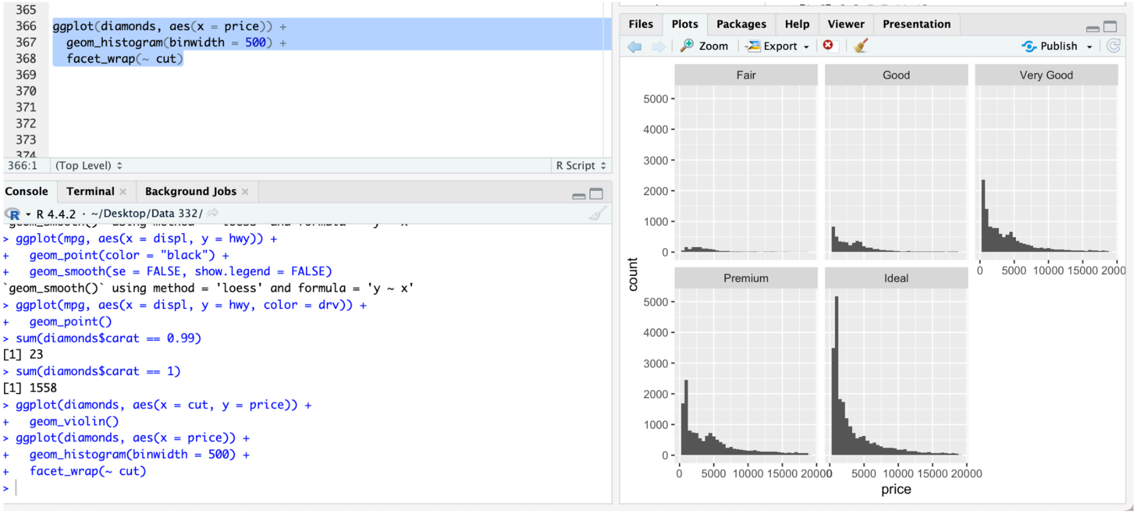
```
ggplot(diamonds, aes(x = cut, y = price)) +
  geom_violin()
```

```
359:25   (Top Level) ÷                                R Script ÷
```

```
+   geom_point(color = "black") +
+   geom_smooth(se = FALSE)
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
> ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(color = "black") +
+   geom_smooth(se = FALSE, show.legend = FALSE)
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
> ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
+   geom_point()
> sum(diamonds$carat == 0.99)
[1] 23
> sum(diamonds$carat == 1)
[1] 1558
> ggplot(diamonds, aes(x = cut, y = price)) +
+   geom_violin()
>
```

-

- geom_histogram()



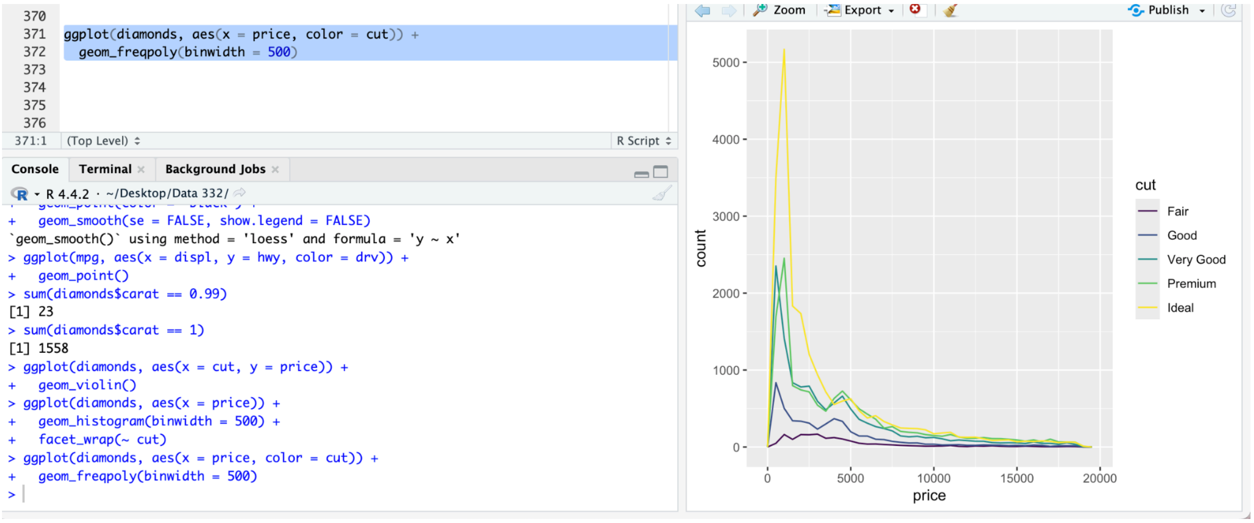
```
ggplot(diamonds, aes(x = price)) +
  geom_histogram(binwidth = 500) +
  facet_wrap(~ cut)
```

```
366:1   (Top Level) ÷                                R Script ÷
```

```
> ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(color = "black") +
+   geom_smooth(se = FALSE, show.legend = FALSE)
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
> ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
+   geom_point()
> sum(diamonds$carat == 0.99)
[1] 23
> sum(diamonds$carat == 1)
[1] 1558
> ggplot(diamonds, aes(x = cut, y = price)) +
+   geom_violin()
> ggplot(diamonds, aes(x = price)) +
+   geom_histogram(binwidth = 500) +
+   facet_wrap(~ cut)
>
```

-

- geom_freqpoly()



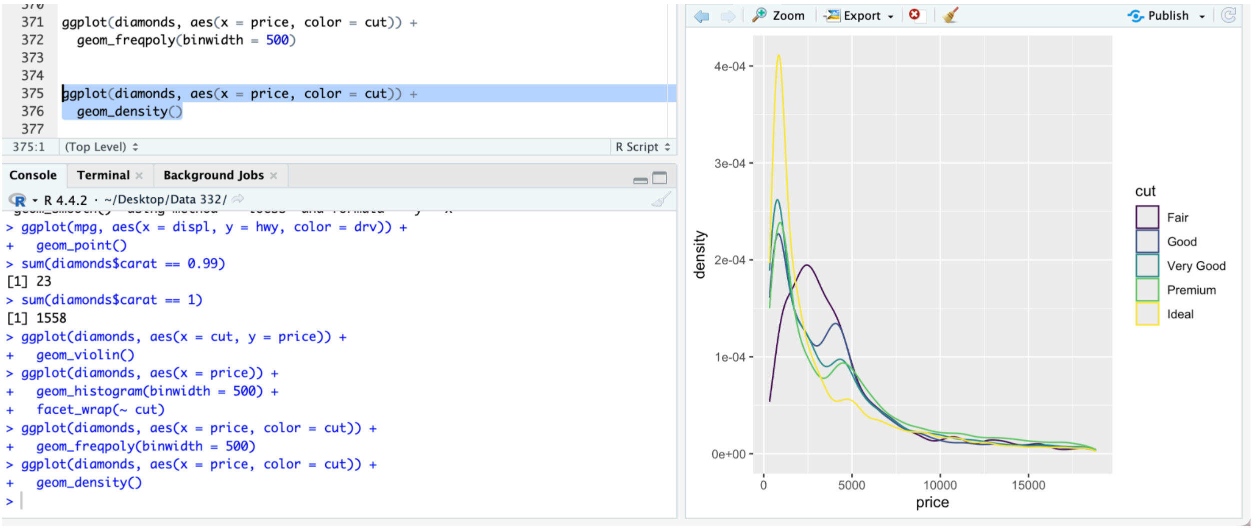
```
ggplot(diamonds, aes(x = price, color = cut)) +
  geom_freqpoly(binwidth = 500)
```

```
371:1   (Top Level) ÷                                R Script ÷
```

```
+   geom_smooth(se = FALSE, show.legend = FALSE)
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
> ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
+   geom_point()
> sum(diamonds$carat == 0.99)
[1] 23
> sum(diamonds$carat == 1)
[1] 1558
> ggplot(diamonds, aes(x = cut, y = price)) +
+   geom_violin()
> ggplot(diamonds, aes(x = price)) +
+   geom_histogram(binwidth = 500) +
+   facet_wrap(~ cut)
> ggplot(diamonds, aes(x = price, color = cut)) +
+   geom_freqpoly(binwidth = 500)
>
```

-

- geom_density()

- 

```
ggplot(diamonds, aes(x = price, color = cut)) +
  geom_freqpoly(binwidth = 500)


ggplot(diamonds, aes(x = price, color = cut)) +
  geom_density()
```

```
> ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
+   geom_point()
> sum(diamonds$carat == 0.99)
[1] 23
> sum(diamonds$carat == 1)
[1] 1558
> ggplot(diamonds, aes(x = cut, y = price)) +
+   geom_violin()
> ggplot(diamonds, aes(x = price)) +
+   geom_histogram(binwidth = 500) +
+   facet_wrap(~ cut)
> ggplot(diamonds, aes(x = price, color = cut)) +
+   geom_freqpoly(binwidth = 500)
> ggplot(diamonds, aes(x = price, color = cut)) +
+   geom_density()
> 
```

6. If you have a small dataset, it's sometimes useful to use geom_jitter() to avoid overplotting to more easily see the relationship between a continuous and categorical variable. The ggbeeswarm package provides a number of methods similar to geom_jitter(). List them and briefly describe what each one does.
   - geom_jitter() adds random noise to the points, making overlapping points easier to see.
   - Adjusting width and height controls the amount of jittering applied.

10.5.2.1 Exercises

1. How could you rescale the count dataset above to more clearly show the distribution of cut within color, or color within cut?

- To better show the distribution of cut within color (or color within cut), you can rescale the counts to proportions. This makes it easier to compare the relative sizes of each group rather than focusing on absolute counts.
- You can do this by setting position = "fill" in geom_bar(). This rescales the y-axis to a range of 0 to 1, so the bars show the proportion of each category instead of the raw count.

2. What different data insights do you get with a segmented bar chart if color is mapped to the x aesthetic and cut is mapped to the fill aesthetic? Calculate the counts that fall into each of the segments.

- When you map color to the x-axis and cut to the fill, you create a segmented bar chart that shows how the distribution of cut varies across different color categories.
- This helps reveal patterns such as which cuts are more common for each color. You can calculate the counts using:

- diamonds %>%
-   count(color, cut)

3. Use geom_tile() together with dplyr to explore how average flight departure delays vary by destination and month of year. What makes the plot difficult to read? How could you improve it?

- If there are too many destinations, the labels may overlap, making the plot hard to read. Small differences in color may also be hard to distinguish.
- Use coord_fixed() to adjust the aspect ratio.
- Use scale_fill_gradient() to adjust the color scale for better contrast.
- Filter to show only the most frequent destinations to reduce clutter.

10.5.3.1 Exercises

1. Instead of summarizing the conditional distribution with a boxplot, you could use a frequency polygon. What do you need to consider when using cut_width()vs.

cut_number()? How does that impact a visualization of the 2d distribution of caratand price?

- **cut_width()** – Divides data into equal-sized intervals. Good for evenly spread data.
- **cut_number()** – Divides data into bins with an equal number of points. Good for uneven data.

Impact:

- cut_width() shows consistent intervals but may hide patterns in uneven data.
- cut_number() adjusts for clustering but may distort natural grouping.

2. Visualize the distribution of carat, partitioned by price.

- ggplot(diamonds, aes(x = carat)) +
-   geom_histogram(binwidth = 0.1) +
-   facet_wrap(~ cut_number(price, n = 5))

3. How does the price distribution of very large diamonds compare to small diamonds? Is it as you expect, or does it surprise you?

- The price distribution for large diamonds is more spread out and tends to be higher, while smaller diamonds have a more concentrated price range. This makes sense since larger diamonds are rarer and their price is influenced more by factors like cut and clarity. However, it's surprising that some large diamonds are priced lower, which could be due to lower quality in cut or clarity.

4. Combine two of the techniques you've learned to visualize the combined distribution of cut, carat, and price.

- ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
-   geom_point(alpha = 0.5) +

- facet_wrap(~ cut)

5. Two dimensional plots reveal outliers that are not visible in one dimensional plots. For example, some points in the following plot have an unusual combination of x and yvalues, which makes the points outliers even though their x and y values appear normal when examined separately. Why is a scatterplot a better display than a binned plot for this case?

```
diamonds |>

 filter(x >= 4) |>

 ggplot(aes(x = x, y = y)) +

 geom_point() +

coord_cartesian(xlim = c(4, 11), ylim = c(4, 11))
```

- A scatterplot is better than a binned plot in this case because it shows the relationship between x and y directly. Even if xand y values seem normal individually, unusual combinations (like very high x but low y) are easier to spot in a scatterplot.

6. Instead of creating boxes of equal width with cut_width(), we could create boxes that contain roughly equal number of points with cut_number(). What are the advantages and disadvantages of this approach?

```
ggplot(smaller, aes(x = carat, y = price)) +

      geom_boxplot(aes(group = cut_number(carat, 20)))
```

Advantages:

- cut_number() creates bins with an equal number of points, making it easier to compare distributions across different groups.
- It ensures that each boxplot has enough data points for a meaningful comparison.

Disadvantages:

- Since the bins have unequal widths, it can distort the true distribution of carat values.
- It might hide patterns in the data if the variable is naturally clustered or unevenly distributed.

Executables

```
#Chapter 9 Layers .....................................

#9.1.1 Prerequisites

library(tidyverse)

#9.2 Aesthetic mappings

mpg

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

ggplot(mpg, aes(x = displ, y = hwy, shape = class)) +
  geom_point()

#Similarly, we can map class to size or alpha aesthetics as well, which control the size and the transparency of the points, respectively.

ggplot(mpg, aes(x = displ, y = hwy, size = class)) +
  geom_point()

ggplot(mpg, aes(x = displ, y = hwy, alpha = class)) +
  geom_point()

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

#9.3 Geometric objects

#To change the geom in your plot, change the geom function that you add to ggplot(). For instance, to make the plots above, you can use the following code:

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

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

ggplot(mpg, aes(x = displ, y = hwy, shape = drv)) +
  geom_smooth()

ggplot(mpg, aes(x = displ, y = hwy, linetype = drv)) +
  geom_smooth()

#If this sounds strange, we can make it clearer by overlaying the lines on top of the raw data and then coloring everything according to drv.

ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(aes(linetype = drv))

#left
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()

# Middle
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(group = drv))

# Right
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(color = drv), show.legend = FALSE)

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

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_point(
    data = mpg |> filter(class == "2seater"),
    color = "red"
  ) +
  geom_point(
    data = mpg |> filter(class == "2seater"),
    shape = "circle open", size = 3, color = "red"
  )

# Left
ggplot(mpg, aes(x = hwy)) +
  geom_histogram(binwidth = 2)

# Middle
ggplot(mpg, aes(x = hwy)) +
  geom_density()

# Right
ggplot(mpg, aes(x = hwy)) +
  geom_boxplot()

library(ggridges)

ggplot(mpg, aes(x = hwy, y = drv, fill = drv, color = drv)) +
  geom_density_ridges(alpha = 0.5, show.legend = FALSE)
#> Picking joint bandwidth of 1.28


#9.4 Facets

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~cyl)

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(drv ~ cyl)

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(drv ~ cyl, scales = "free")

#9.5 Statistical transformations

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

```
#You might want to override the default stat. In the code below, we change the stat of geom_bar() from count (the default) to identity. This lets us map the height of the bars to the raw values of a y variable.

diamonds |>
  count(cut) |>
  ggplot(aes(x = cut, y = n)) +
  geom_bar(stat = "identity")

#You might want to override the default mapping from transformed variables to aesthetics. For example, you might want to display a bar chart of proportions, rather than counts:

ggplot(diamonds, aes(x = cut, y = after_stat(prop), group = 1)) +
  geom_bar()

ggplot(diamonds) +
  stat_summary(
    aes(x = cut, y = depth),
    fun.min = min,
    fun.max = max,
    fun = median
  )

#9.6 Position adjustments

# Left
ggplot(mpg, aes(x = drv, color = drv)) +
  geom_bar()

# Right
ggplot(mpg, aes(x = drv, fill = drv)) +
  geom_bar()

ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar()

# Left
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(alpha = 1/5, position = "identity")
```

```
# Right
ggplot(mpg, aes(x = drv, color = class)) +
  geom_bar(fill = NA, position = "identity")

# Left
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(position = "fill")

# Right
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(position = "dodge")

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(position = "jitter")

#9.7 Coordinate systems

nz <- map_data("nz")

ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black")

ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black") +
  coord_quickmap()

bar <- ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = clarity, fill = clarity),
    show.legend = FALSE,
    width = 1
  ) +
  theme(aspect.ratio = 1)

bar + coord_flip()
bar + coord_polar()

#Chapter 10 Exploratory Data analysis

#10.1.1 Prerequisites
```

```
library(tidyverse)

#10.3 Variation

ggplot(diamonds, aes(x = carat)) +
  geom_histogram(binwidth = 0.5)

#10.3.1 Typical values

smaller <- diamonds |>
  filter(carat < 3)

ggplot(smaller, aes(x = carat)) +
  geom_histogram(binwidth = 0.01)

#10.3.2 Unusual values

ggplot(diamonds, aes(x = y)) +
  geom_histogram(binwidth = 0.5)

ggplot(diamonds, aes(x = y)) +
  geom_histogram(binwidth = 0.5) +
  coord_cartesian(ylim = c(0, 50))

unusual <- diamonds |>
  filter(y < 3 | y > 20) |>
  select(price, x, y, z) |>
  arrange(y)
unusual

#10.4 Unusual values

#Drop the entire row with the strange values:

diamonds2 <- diamonds |>
  filter(between(y, 3, 20))

diamonds2 <- diamonds |>
  mutate(y = if_else(y < 3 | y > 20, NA, y))
```

```
ggplot(diamonds2, aes(x = x, y = y)) +
  geom_point()

#To suppress that warning, set na.rm = TRUE:
ggplot(diamonds2, aes(x = x, y = y)) +
  geom_point(na.rm = TRUE)

nycflights13::flights |>
  mutate(
    cancelled = is.na(dep_time),
    sched_hour = sched_dep_time %/% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + (sched_min / 60)
  ) |>
  ggplot(aes(x = sched_dep_time)) +
  geom_freqpoly(aes(color = cancelled), binwidth = 1/4)

#10.5 Covariation

#10.5.1 A categorical and a numerical variable

ggplot(diamonds, aes(x = price)) +
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)

ggplot(diamonds, aes(x = price, y = after_stat(density))) +
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)

#A visually simpler plot for exploring this relationship is using side-by-side boxplots.

ggplot(diamonds, aes(x = cut, y = price)) +
  geom_boxplot()

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

#To make the trend easier to see, we can reorder class based on the median value of hwy:

ggplot(mpg, aes(x = fct_reorder(class, hwy, median), y = hwy)) +
  geom_boxplot()
```

#If you have long variable names, geom_boxplot() will work better if you flip it 90°. You can do that by exchanging the x and y aesthetic mappings.

```
ggplot(mpg, aes(x = hwy, y = fct_reorder(class, hwy, median))) +
  geom_boxplot()
```

#10.5.2 Two categorical variables

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

#Another approach for exploring the relationship between these variables is computing the counts with dplyr:

```
diamonds |>
  count(color, cut)
```

#Then visualize with geom_tile() and the fill aesthetic:

```
diamonds |>
  count(color, cut) |>
  ggplot(aes(x = color, y = cut)) +
  geom_tile(aes(fill = n))
```

#10.5.3 Two numerical variables

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

ggplot(smaller, aes(x = carat, y = price)) +
  geom_point(alpha = 1 / 100)

ggplot(smaller, aes(x = carat, y = price)) +
  geom_bin2d()

# install.packages("hexbin")
ggplot(smaller, aes(x = carat, y = price)) +
  geom_hex()
```

```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_boxplot(aes(group = cut_width(carat, 0.1)))

#10.6 Patterns and models

library(tidymodels)

diamonds <- diamonds |>
  mutate(
    log_price = log(price),
    log_carat = log(carat)
  )

diamonds_fit <- linear_reg() |>
  fit(log_price ~ log_carat, data = diamonds)

diamonds_aug <- augment(diamonds_fit, new_data = diamonds) |>
  mutate(.resid = exp(.resid))

ggplot(diamonds_aug, aes(x = carat, y = .resid)) +
  geom_point()

ggplot(diamonds_aug, aes(x = cut, y = .resid)) +
  geom_boxplot()
```