Hadley Wickham EDA using base R graphics

A version of the eBook Chapter "7 Exploratory Data Analysis", in *R For Data Science* by Hadley Wickham, at https://r4ds.had.co.nz/exploratory-data-analysis.html.

7 Exploratory Data Analysis

7.1 Introduction

This chapter will show you how to use visualisation and transformation to explore your data in a systematic way, a task that statisticians call exploratory data analysis, or EDA for short. EDA is an iterative cycle. You:

- Generate questions about your data
- Search for answers by visualising, transforming, and modelling your data
- Use what you learn to refine your questions and/or generate new questions

EDA is not a formal process with a strict set of rules. More than anything, *EDA is a state of mind.* During the initial phases of EDA you should feel free to investigate every idea that occurs to you. Some of these ideas will pan out, and some will be dead ends. As your exploration continues, you will home in on a few particularly productive areas that you'll eventually write up and communicate to others.

EDA is an important part of any data analysis, even if the questions are handed to you on a platter, because you always need to investigate the quality of your data. Data cleaning is just one application of EDA: you ask questions about whether your data meets your expectations or not. To do data cleaning, you'll need to deploy all the tools of EDA: visualisation, transformation, and modelling.

7.1.1 Prerequisites

In this chapter we'll interactively ask questions, answer them with data, and then ask new questions.

7.2 Questions

> "There are no routine statistical questions, only questionable statistical routines." — Sir David Cox

> "Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise." — John Tukey

Your goal during EDA is to develop an understanding of your data. The easiest way to do this is to use questions as tools to guide your investigation. When you ask a question, the question focuses your attention on a specific part of your dataset and helps you decide which graphs, models, or transformations to make.

EDA is fundamentally a creative process. And like most creative processes, the key to asking quality questions is to generate a large quantity of questions. It is difficult to ask revealing questions at the start of your analysis because you do not know what insights are contained in your dataset. On the other hand, each new question that you ask will expose you to a new aspect of your data and increase your chance of making a discovery. You can quickly drill down into the most interesting parts of your data—and develop a set of thought-provoking questions—if you follow up each question with a new question based on what you find.

There is no rule about which questions you should ask to guide your research. However, two types of questions will always be useful for making discoveries within your data. You can loosely word these questions as:

- What type of variation occurs within my variables?
- What type of covariation occurs between my variables?

The rest of this chapter will look at these two questions. I'll explain what variation and covariation are, and I'll show you several ways to answer each question. To make the discussion easier, let's define some terms:

A **variable** is a quantity, quality, or property that you can measure.

A **value** is the state of a variable when you measure it. The value of a variable may change from measurement to measurement.

An **observation** is a set of measurements made under similar conditions (you usually make all of the measurements in an observation at the same time and on the same object). An observation will contain several values, each associated with a different variable. I'll sometimes refer to an observation as a data point.

Tabular data is a set of values, each associated with a variable and an observation. Tabular data is tidy if each value is placed in its own "cell", each variable in its own column, and each observation in its own row.

So far, most of the data that you've seen has been tidy. In real-life, most data isn't tidy.

7.3 Variation

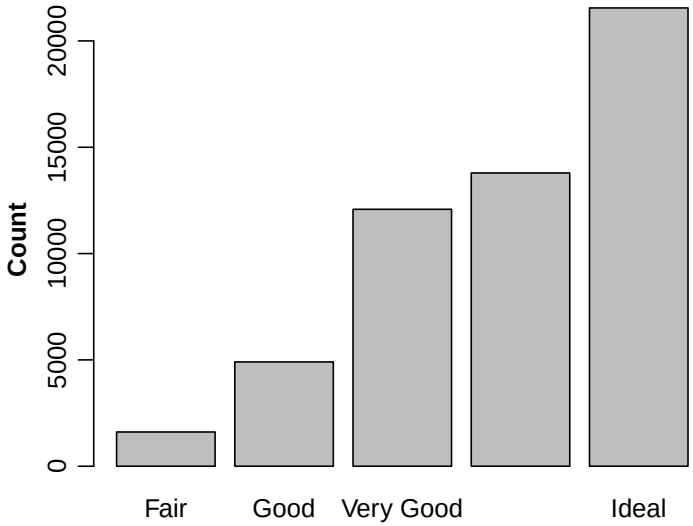
Variation is the tendency of the values of a variable to change from measurement to measurement. You can see variation easily in real life; if you measure any continuous variable twice, you will get two different results. This is true even if you measure quantities that are constant, like the speed of light. Each of your measurements will include a small amount of error that varies from measurement to measurement. Categorical variables can also vary if you measure across different subjects (e.g. the eye colors of different people), or different times (*e.g.* the energy levels of an electron at different moments). Every variable has its own pattern of variation, which can reveal interesting information. The best way to understand that pattern is to visualise the distribution of the variable's values.

7.3.1 Visualising distributions

How you visualise the distribution of a variable will depend on whether the variable is categorical or continuous. A variable is **categorical** if it can only take one of a small set of values. In R, categorical variables are usually saved as factors or character vectors. To examine the distribution of a categorical variable, use a bar chart:

```
par(mar = c(4,4,1,1), mgp = c(2,0.7,0), font.lab=2)
```

```
barplot(table(diamonds$cut), ylab = "Count")
```



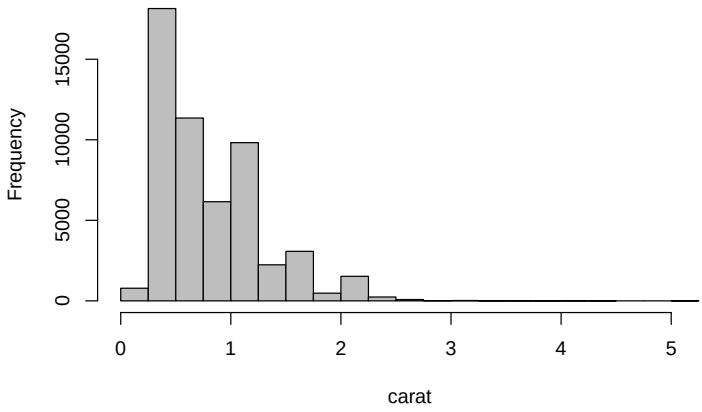
The height of the bars displays how many observations occurred with each x value. You can compute these values manually with `table()`:

```
table(diamonds$cut)
```

```
##
##      Fair      Good Very Good   Premium     Ideal
##      1610      4906     12082     13791     21551
```

A variable is **continuous** if it can take any of an infinite set of ordered values. Numbers and date-times are two examples of continuous variables. To examine the distribution of a continuous variable, use a histogram:

```
with(diamonds, hist(carat,
     main = "",
     col = "grey",
     breaks = seq(0,5.25,0.25))
)
```



You can compute this by hand by combining `table()` and `cut()`:

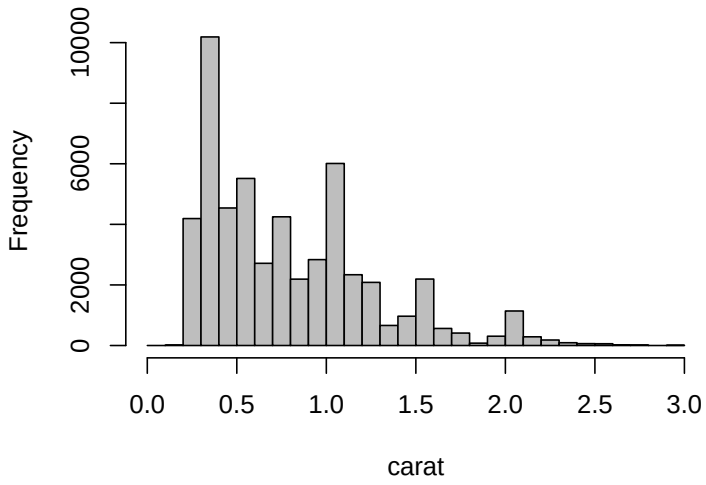
```
table(cut(diamonds$carat,
    breaks = seq(0,5.25,0.25)))
```

```
##
##   (0,0.25] (0.25,0.5] (0.5,0.75]   (0.75,1]   (1,1.25] (1.25,1.5]
##        785      18147      11351       6155       9822       2238
## (1.5,1.75]   (1.75,2]   (2,2.25] (2.25,2.5] (2.5,2.75]   (2.75,3]
##       3075        478       1524        239         83         11
##   (3,3.25] (3.25,3.5] (3.5,3.75]   (3.75,4]   (4,4.25] (4.25,4.5]
##         21          2          3          1          3          1
## (4.5,4.75]   (4.75,5]   (5,5.25]
##          0          0          1
```

A histogram divides the x-axis into equally spaced bins and then uses the height of a bar to display the number of observations that fall in each bin. In the graph above, the tallest bar shows that over 18,000 observations have a carat value between 0.25 and 0.5, which are the left and right edges of the bar.

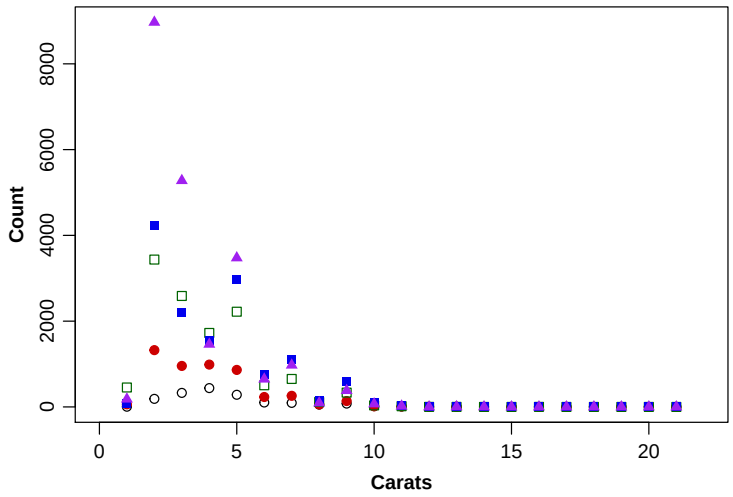
You can set the width of the intervals in a histogram with the breaks argument, which is measured in the units of the x variable. You should always explore a variety of binwidths when working with histograms, as different binwidths can reveal different patterns. For example, here is how the graph above looks when we zoom into just the diamonds with a size of less than three carats and choose a smaller binwidth.

```
smaller <- subset(diamonds, subset = diamonds$carat<=3)
with(smaller, hist(carat,
     main = "",
     col = "grey",
     breaks = seq(0,3,0.1))
)
```



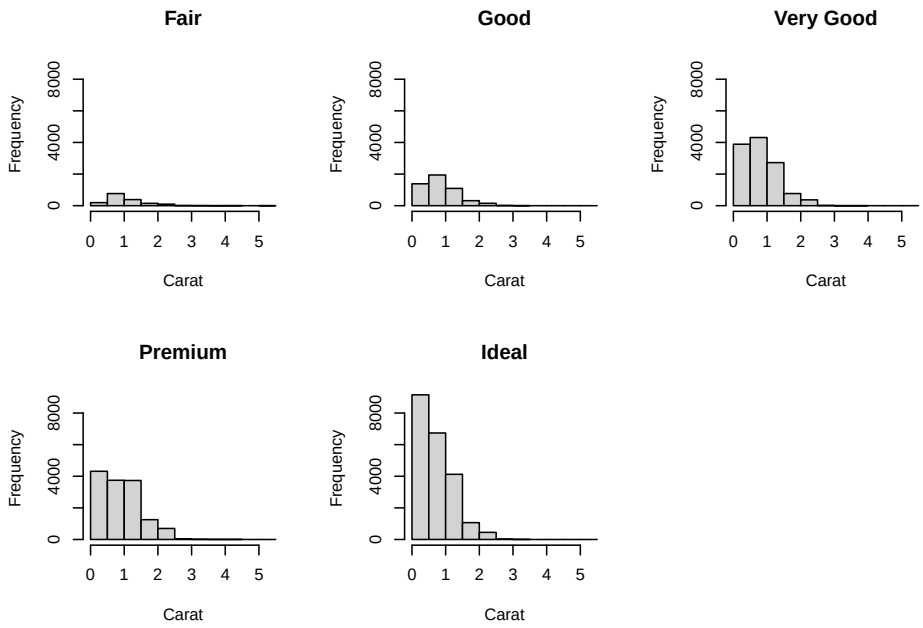
If you wish to overlay multiple histograms in the same plot, I recommend using geom_freqpoly() instead of geom_histogram(). geom_freqpoly() performs the same calculation as geom_histogram(), but instead of displaying the counts with bars, uses lines instead. It's much easier to understand overlapping lines than bars.

```
table2way <- as.data.frame(table(diamonds$cut, cut(diamonds$carat,
    breaks = seq(0,5.25,0.25))))
colnames(table2way) <- c("Cut","Carats","Count")
table2way$Carats <- as.numeric(table2way$Carats)
par(mar = c(4,4,1,1), mgp = c(2,0.7,0), font.lab=2)
palette(c("black","red3","darkgreen","blue2",
          "purple","darkcyan","sienna","gray65"))
with(table2way, plot(Carats, Count,
     col = seq(1, 5)[table2way$Cut],
     pch = c(1, 19, 0, 15, 17)[table2way$Cut],
     xlim = c(0, 22),
     type = "p"))
```



Alternatively

```
par(mfrow = c(2, 3))
for (i in 1:nlevels(diamonds$cut)) {
  subdata <- subset(diamonds,
                    subset = diamonds$cut==levels(diamonds$cut)[i])
  hist(subdata$carat, breaks = seq(0,5.5,0.5),
       main = levels(diamonds$cut)[i],
       xlab = "Carat",
       ylim = c(0, 9000),
       col = "lightgray")
}
par(mfrow = c(1, 1))
```



There are a few challenges with these type of plot, which we will come back to in visualising a categorical and a continuous variable.

Now that you can visualise variation, what should you look for in your plots? And what type of follow-up questions should you ask? I've put together a list below of the most useful types of information that you will find in your graphs, along with some follow-up questions for each type of information. The key to asking good follow-up questions will be to rely on your curiosity (What do you want to learn more about?) as well as your skepticism (How could this be misleading?).

7.3.2 Typical values

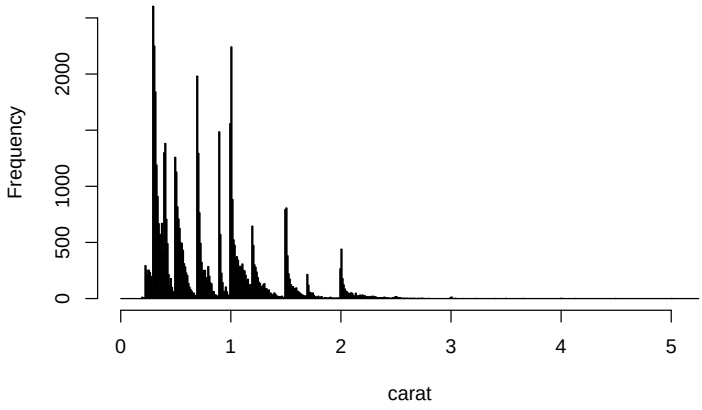
In both bar charts and histograms, tall bars show the common values of a variable, and shorter bars show less-common values. Places that do not have bars reveal values that were not seen in your data. To turn this information into useful questions, look for anything unexpected:

- Which values are the most common? Why?
- Which values are rare? Why? Does that match your expectations?
- Can you see any unusual patterns? What might explain them?

As an example, the histogram below suggests several interesting questions:

- Why are there more diamonds at whole carats and common fractions of carats?
- Why are there more diamonds slightly to the right of each peak than there are slightly to the left of each peak?
- Why are there no diamonds bigger than 3 carats?

```
with(diamonds,
     hist(carat,
     main = "",
     breaks = seq(0,5.25,0.01))
     )
```

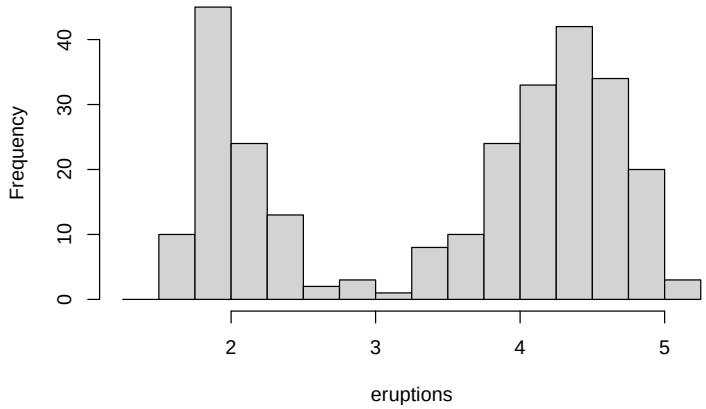


Clusters of similar values suggest that subgroups exist in your data. To understand the subgroups, ask:

- How are the observations within each cluster similar to each other?
- How are the observations in separate clusters different from each other?
- How can you explain or describe the clusters?
- Why might the appearance of clusters be misleading?

The histogram below shows the length (in minutes) of 272 eruptions of the Old Faithful Geyser in Yellowstone National Park. Eruption times appear to be clustered into two groups: there are short eruptions (of around 2 minutes) and long eruptions (4-5 minutes), but little in between.

```
data("faithful")
with(faithful,
     hist(eruptions, breaks = seq(1.25, 5.25, 0.25),
          col = "lightgray",
          main="")
     )
```

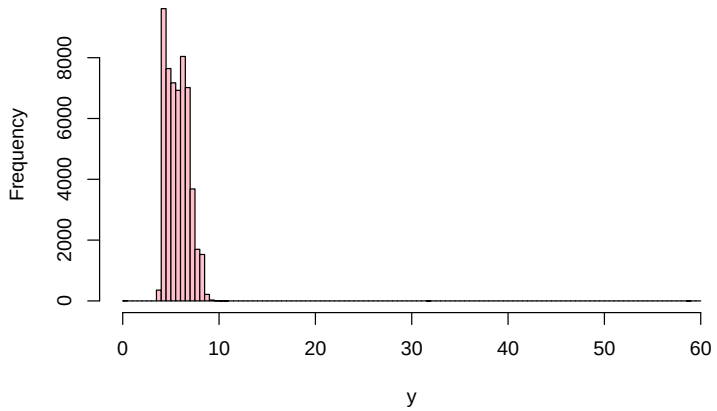


Many of the questions above will prompt you to explore a relationship between variables, for example, to see if the values of one variable can explain the behavior of another variable. We'll get to that shortly.

7.3.3 Unusual values

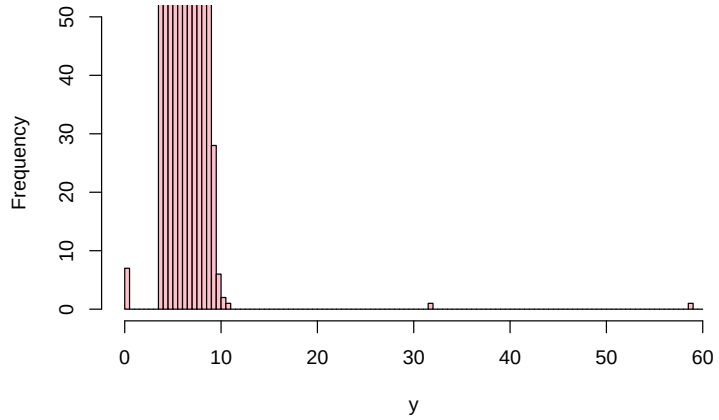
Outliers are observations that are unusual; data points that don't seem to fit the pattern. Sometimes outliers are data entry errors; other times outliers suggest important new science. When you have a lot of data, outliers are sometimes difficult to see in a histogram. For example, take the distribution of the y variable from the diamonds dataset. The only evidence of outliers is the unusually wide limits on the x-axis.

```
with(diamonds,
     hist(y,
     breaks = seq(0, 60, 0.5),
     main="", col = "pink"))
```



There are so many observations in the common bins that the rare bins are so short that you can't see them (although maybe if you stare intently at 0 you'll spot something). To make it easy to see the unusual values, we need to zoom to small values of the y-axis with the `ylim()` option:

```
with(diamonds,
     hist(y,
     breaks = seq(0, 60, 0.5),
     main="", col = "pink",
     ylim = c(0, 50)))
```



(Many plot functions in R also have an `xlim()` argument for when you need to zoom into the x-axis.)

This allows us to see that there are three unusual values: 0, ~30, and ~60. We pluck them out with `subset()`:

```
unusual <- subset(diamonds, subset = diamonds$y<3 | diamonds$y>20)
print(unusual)
```

```
##       carat       cut color clarity depth table price    x    y    z
## 11964  1.00 Very Good     H     VS2  63.3    53  5139 0.00  0.0 0.00
## 15952  1.14      Fair     G     VS1  57.5    67  6381 0.00  0.0 0.00
## 24068  2.00   Premium     H     SI2  58.9    57 12210 8.09 58.9 8.06
## 24521  1.56     Ideal     G     VS2  62.2    54 12800 0.00  0.0 0.00
## 26244  1.20   Premium     D    VVS1  62.1    59 15686 0.00  0.0 0.00
## 27430  2.25   Premium     H     SI2  62.8    59 18034 0.00  0.0 0.00
## 49190  0.51     Ideal     E     VS1  61.8    55  2075 5.15 31.8 5.12
## 49557  0.71      Good     F     SI2  64.1    60  2130 0.00  0.0 0.00
## 49558  0.71      Good     F     SI2  64.1    60  2130 0.00  0.0 0.00
```

The y variable measures one of the three dimensions of these diamonds, in mm. We know that diamonds can't have a width of 0 mm, so these values must be incorrect. We might also suspect that measurements of 32 mm and 59 mm are implausible: those diamonds are over an inch long, but don't cost hundreds of thousands of dollars!

It's good practice to repeat your analysis with and without the outliers. If they have minimal effect on the results, and you can't figure out why they're there, it's reasonable to replace them with missing values, and move on. **However, if they have a substantial effect on your results, you shouldn't drop them without justification**. You'll need to figure out what caused them (*e.g.* a data entry error) and disclose that you removed them in your write-up.

7.3.4 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.

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.)

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

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?

7.4 Missing values

If you've encountered unusual values in your dataset, and simply want to move on to the rest of your analysis, you have two options.

Drop the entire row with the strange values (row numbers from table above):

```
diamonds2 <- diamonds[-as.numeric(row.names(unusual)),]
cat("\nRemoved",NROW(diamonds)-NROW(diamonds2),"rows from data frame.\n")
```

```
##
## Removed 9 rows from data frame.
```

I don't recommend this option because just because one measurement is invalid, doesn't mean all the measurements are. Additionally, if you have low quality data, by time that you've applied this approach to every variable you might find that you don't have any data left!

Instead, I recommend replacing the unusual values with missing values. The easiest way to do this is to use mutate() to replace the variable with a modified copy. You can use the ifelse() function to replace unusual values with NA:

```
require(dplyr)
```

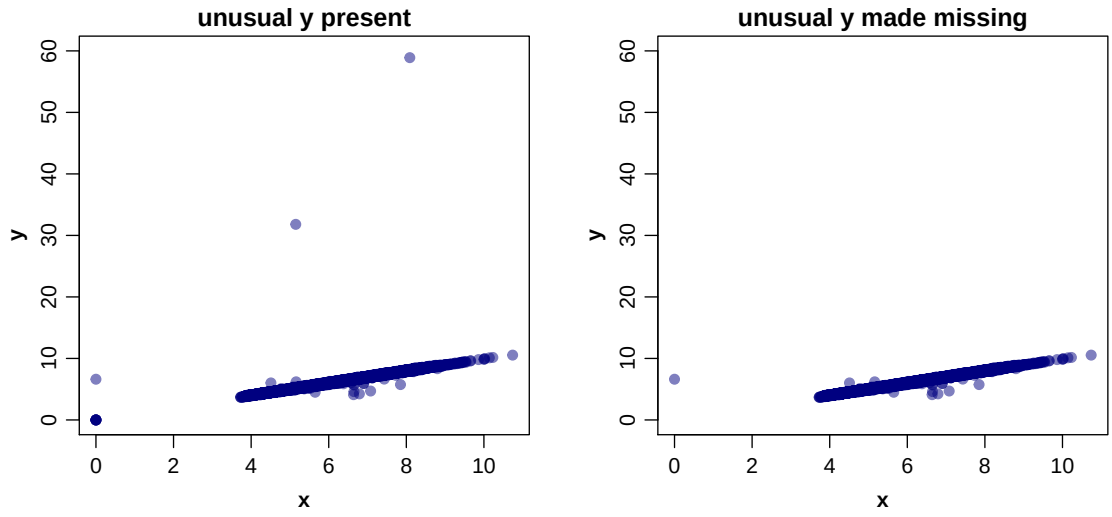
```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

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

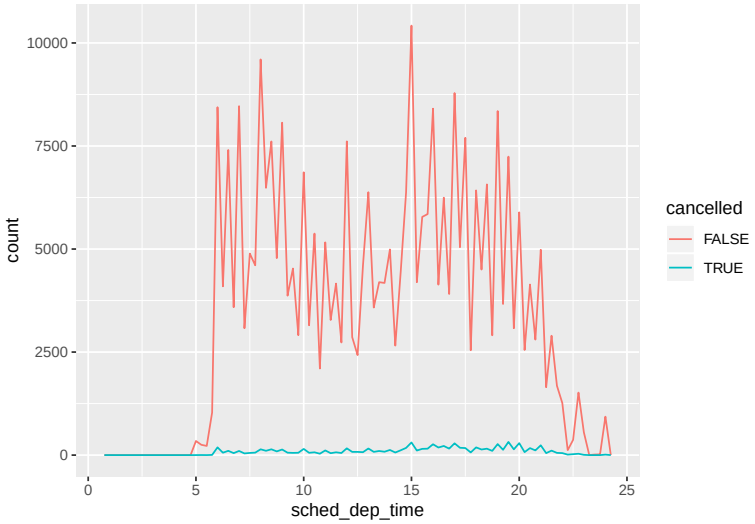
ifelse() has three arguments. The first argument test should be a logical vector. The result will contain the value of the second argument, yes, when test is TRUE, and the value of the third argument, no, when it is false. Alternatively to ifelse, use dplyr::case_when(). case_when() is particularly useful inside mutate when you want to create a new variable that relies on a complex combination of existing variables.

Like R, ggplot2 subscribes to the philosophy that missing values should never silently go missing. It's not obvious where you should plot missing values, so ggplot2 doesn't include them in the plot, but it does warn that they've been removed:

```
par(mfrow = c(1, 2), mar = c(3,4,1.5,1), mgp = c(2,0.7,0),
    font.lab=2)
with(diamonds,
     plot(y ~ x, ylim = c(0, 60),
          col = "#00008080", pch = 19, main = "unusual y present"))
with(diamonds2,
     plot(y ~ x, ylim = c(0, 60),
          col = "#00008080", pch = 19, main = "unusual y made missing"))
```



At other times you might want to understand what makes observations with missing values different to observations with recorded values.For example, in the New York flights data plotted below, missing values in the \texttt{sched_dep_time} variable indicate that the flight was cancelled. So you might want to compare the scheduled departure times for cancelled and non-cancelled times. You can do this by making a new variable with is.na().



However this plot isn't great because there are many more non-cancelled flights than cancelled flights. In the next section we'll explore some techniques for improving this comparison.

7.4.1 Exercises

1. What happens to missing values in a histogram? What happens to missing values in a bar chart? Why is there a difference?

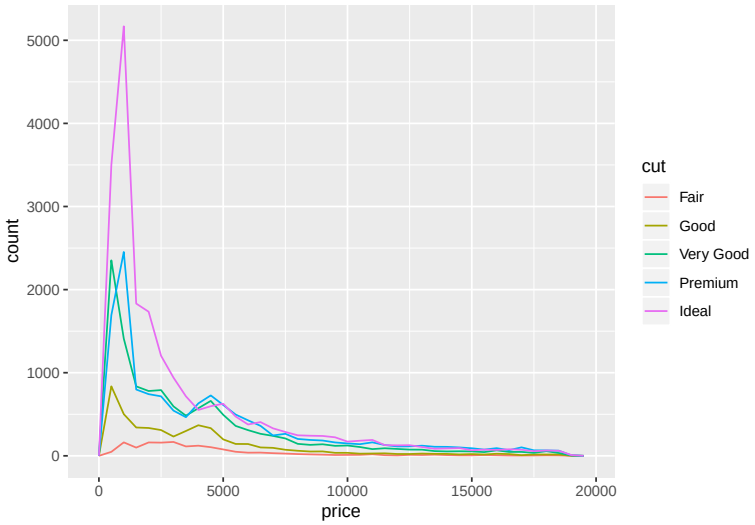
2. What does na.rm = TRUE do in mean() and sum()?

7.5 Covariation

If variation describes the behavior within a variable, covariation describes the behavior between variables. Covariation is the tendency for the values of two or more variables to vary together in a related way. The best way to spot covariation is to visualise the relationship between two or more variables. How you do that should again depend on the type of variables involved.

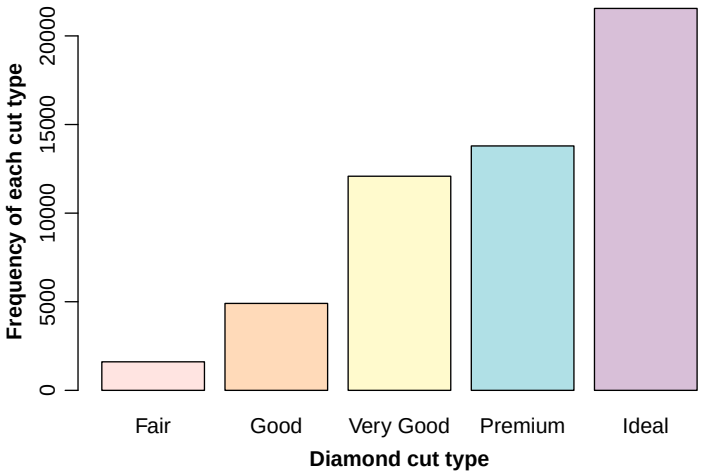
7.5.1 A categorical and continuous variable

It's common to want to explore the distribution of a continuous variable broken down by a categorical variable, as in the previous frequency polygon. The default appearance of geom_freqpoly() is not that useful for that sort of comparison because the height is given by the count. That means if one of the groups is much smaller than the others, it's hard to see the differences in shape. For example, let's explore how the price of a diamond varies with its quality:

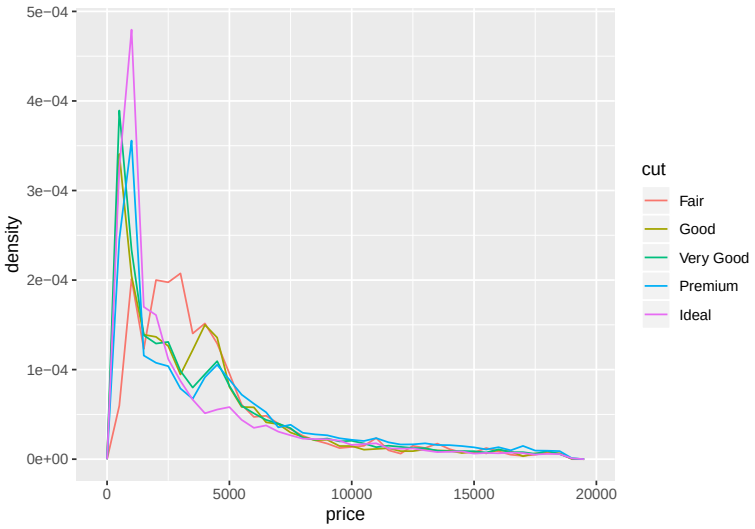


It's hard to see the difference in distribution because the overall counts differ so much:

```
par(mar = c(4,4,1,1), mgp = c(2,0.7,0), font.lab=2)
barplot(table(diamonds$cut),
     xlab = "Diamond cut type",
     ylab = "Frequency of each cut type",
     col = c("mistyrose", "peachpuff", "lemonchiffon", "powderblue", "thistle"))
```



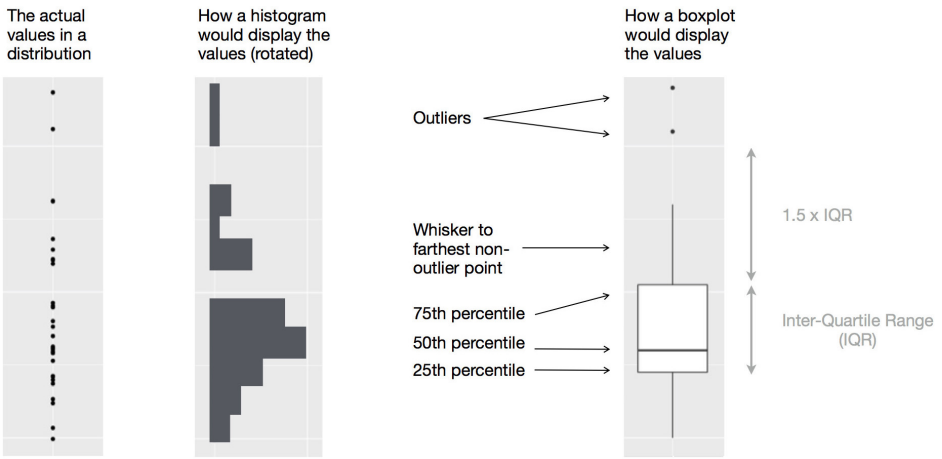
To make the comparison easier we need to swap what is displayed on the y-axis. Instead of displaying count, we'll display density, which is the count standardised so that the area under each frequency polygon is one.



There's something rather surprising about this plot - it appears that fair diamonds (the lowest quality) have the highest average price! But maybe that's because frequency polygons are a little hard to interpret - there's a lot going on in this plot.

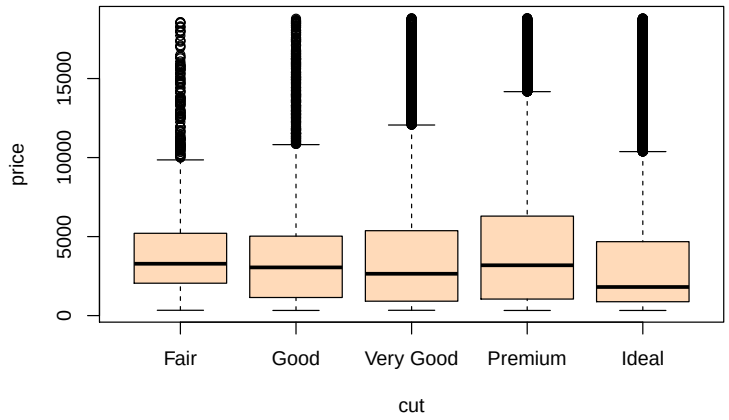
Another alternative to display the distribution of a continuous variable broken down by a categorical variable is the **boxplot**. A boxplot is a type of visual shorthand for a distribution of values that is popular among statisticians. Each boxplot consists of:

- A box that stretches from the 25th percentile of the distribution to the 75th percentile, a distance known as the **interquartile range (IQR)**. In the middle of the box is a line that displays the median, i.e. 50th percentile, of the distribution. These three lines give you a sense of the spread of the distribution and whether or not the distribution is symmetric about the median or skewed to one side.
- Visual points that display observations that fall more than 1.5 times the IQR from either edge of the box. These outlying points are unusual so are plotted individually.
- A line (or whisker) that extends from each end of the box and goes to the farthest non-outlier point in the distribution.



Let's take a look at the distribution of price by cut using geom_boxplot():

```
with(diamonds,
     boxplot(price ~ cut,
             col = "peachpuff")
)
```



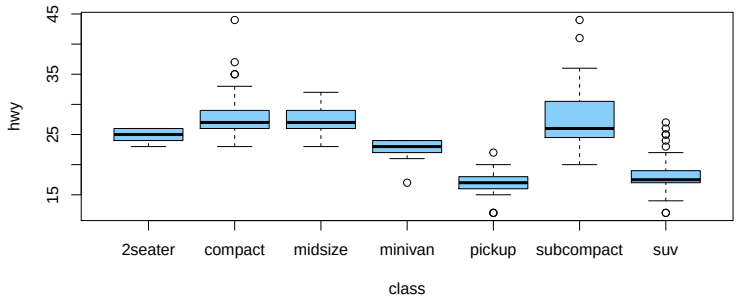
ggplot(data = diamonds, mapping = aes(x = cut, y = price)) + geom_boxplot()

We see much less information about the distribution, but the boxplots are much more compact so we can more easily compare them (and fit more on one plot). It supports the counterintuitive finding that better quality diamonds are cheaper on average! In the exercises, you'll be challenged to figure out why.

`cut` is an ordered factor: fair is worse than good, which is worse than very good and so on. Many categorical variables don't have such an intrinsic order, so you might want to reorder them to make a more informative display. One way to do that is with the reorder() function.

For example, take the class variable in the mpg dataset. You might be interested to know how highway mileage varies across classes:

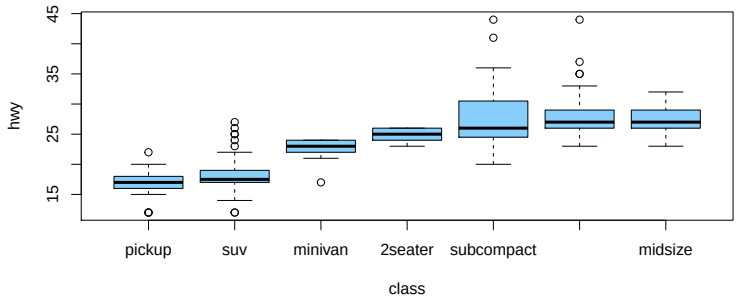
```
with(mpg,
     boxplot(hwy ~ class,
             col = "lightskyblue"))
```



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

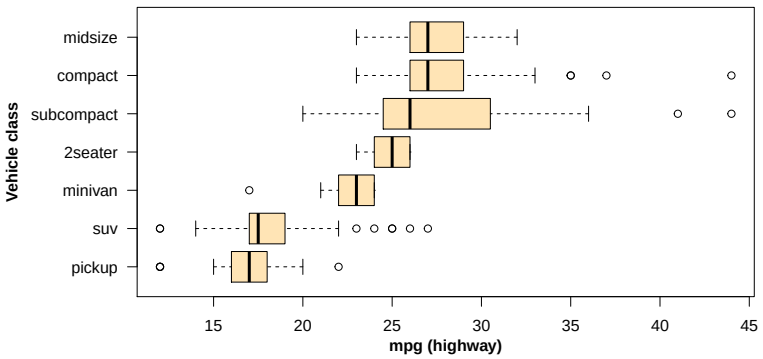
```
mpg2 <- mpg
mpg2$class <- factor(mpg2$class,
                     levels = c("pickup", "suv", "minivan", "2seater",
                                "subcompact", "compact", "midsize"))
with(mpg2,
```

```
      boxplot(hwy ~ class,
              col = "lightskyblue"))
```



If you have long variable names, geom_boxplot() will work better if you flip it 90?. You can do that with coord_flip().

```
par(las=1, mar = c(4,7,1,1), mgp = c(6,1,0), font.lab = 2)
with(mpg2,
     boxplot(hwy ~ class,
             horizontal = TRUE,
             col = "moccasin",
             ylab = "Vehicle class"))
mtext("mpg (highway)",1,2, font=2)
```



NOTE: the graphics, data analyses, and exercises in the following sections rely heavily on use of the ggplot2 package (and also the package dplyr)

7.5.1.1 Exercises

1. Use what you've learned to improve the visualisation of the departure times of cancelled vs. non-cancelled flights.
2. What variable in the diamonds dataset is 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?
3. Install the ggstance package, and create a horizontal boxplot. How does this compare to using coord_flip()?
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?

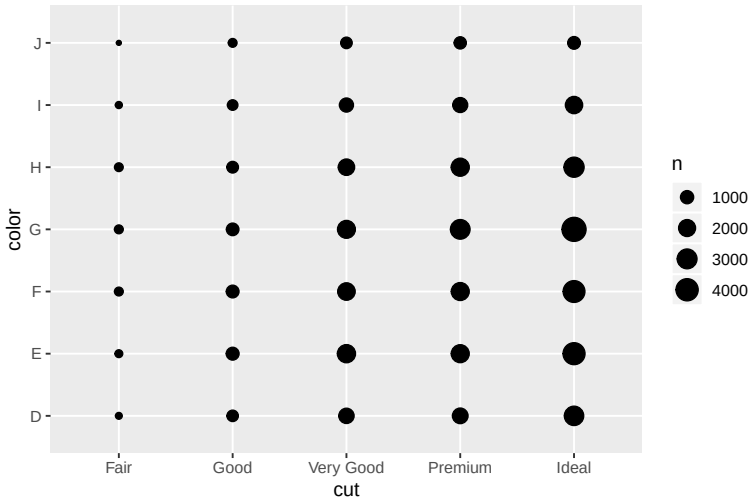
5. Compare and contrast geom_violin() with a facetted geom_histogram(), or a coloured geom_freqpoly(). What are the pros and cons of each method?

6. If you have a small dataset, it's sometimes useful to use geom_jitter() to 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.

7.5.2 Two categorical variables

To visualise the covariation between categorical variables, you'll need to count the number of observations for each combination. One way to do that is to rely on the built-in geom_count():

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



The size of each circle in the plot displays how many observations occurred at each combination of values. Covariation will appear as a strong correlation between specific x values and specific y values.

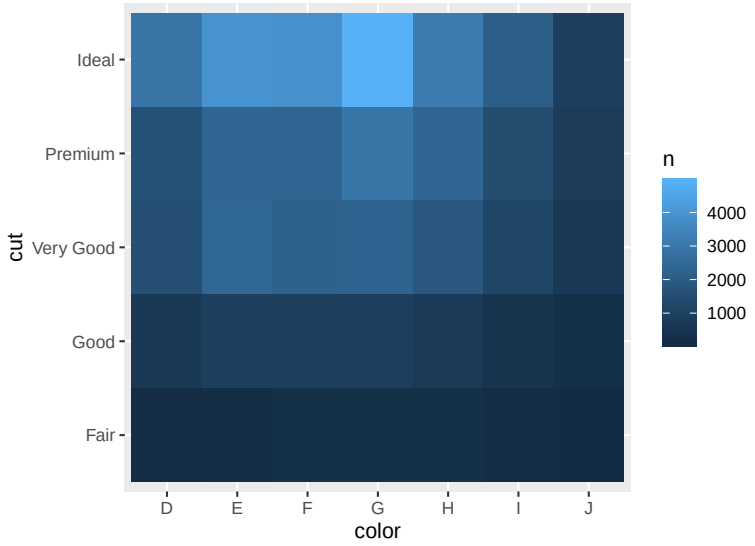
Another approach is to compute the count with dplyr:

```
require(dplyr)
diamonds %>%
  count(color, cut)
```

```
## # A tibble: 35 x 3
##    color cut           n
##    <fct> <fct>     <int>
##  1 D     Fair        163
##  2 D     Good        662
##  3 D     Very Good  1513
##  4 D     Premium    1603
##  5 D     Ideal      2834
##  6 E     Fair        224
##  7 E     Good        933
##  8 E     Very Good  2400
##  9 E     Premium    2337
## 10 E     Ideal      3903
## # ... with 25 more rows
```

Then visualise with geom_tile() and the fill aesthetic:

```
require(ggplot2)
diamonds %>%
  count(color, cut) %>%
  ggplot(mapping = aes(x = color, y = cut)) +
    geom_tile(mapping = aes(fill = n))
```



If the categorical variables are unordered, you might want to use the seriation package to simultaneously reorder the rows and columns in order to more clearly reveal interesting patterns. For larger plots, you might want to try the d3heatmap or heatmaply packages, which create interactive plots.

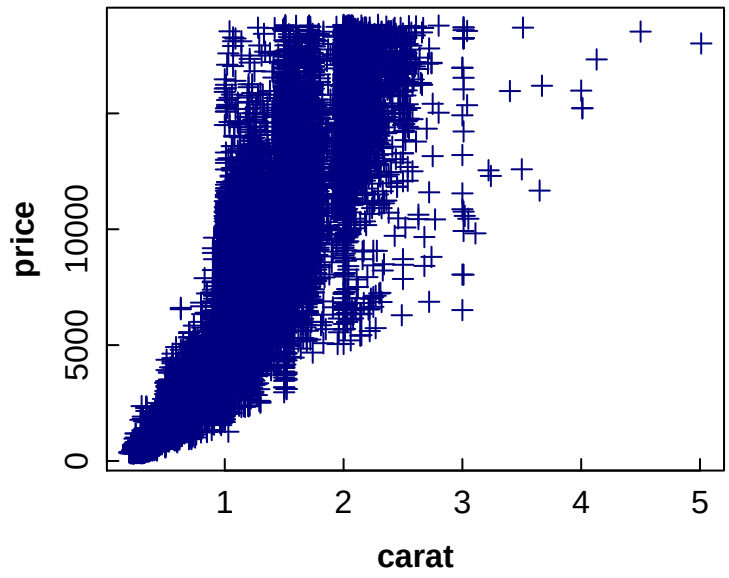
7.5.2.1 Exercises

1. How could you rescale the count dataset above to more clearly show the distribution of cut within colour, or colour within cut?
2. Use geom_tile() together with dplyr to explore how average flight delays vary by destination and month of year. What makes the plot difficult to read? How could you improve it?
3. Why is it slightly better to use aes(x = color, y = cut) rather than aes(x = cut, y = color) in the example above?

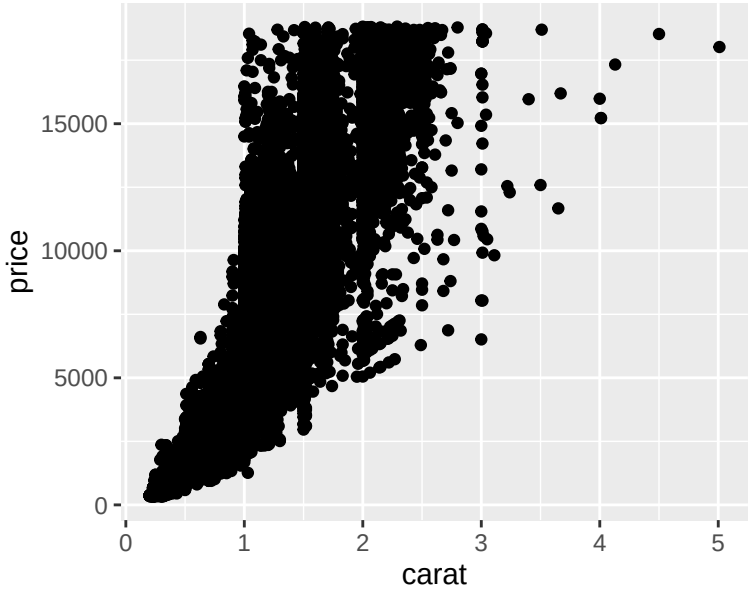
7.5.3 Two continuous variables

You've already seen one great way to visualise the covariation between two continuous variables: draw a scatterplot with geom_point(). You can see covariation as a pattern in the points. For example, you can see an exponential relationship between the carat size and price of a diamond.

```
par(mar = c(3,3,1,1), mgp = c(1.7,0.3,0), tcl = 0.3, font.lab = 2)
with(diamonds,
     plot(price ~ carat, col = "navy", pch = 3))
```

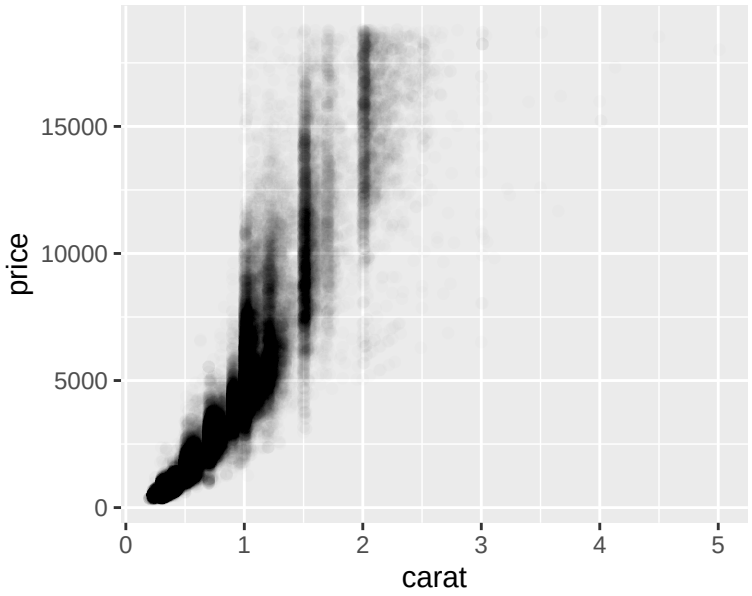



```
require(ggplot2)
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = carat, y = price))
```



Scatterplots become less useful as the size of your dataset grows, because points begin to overplot, and pile up into areas of uniform black (as above). You've already seen one way to fix the problem: using the alpha aesthetic to add transparency.

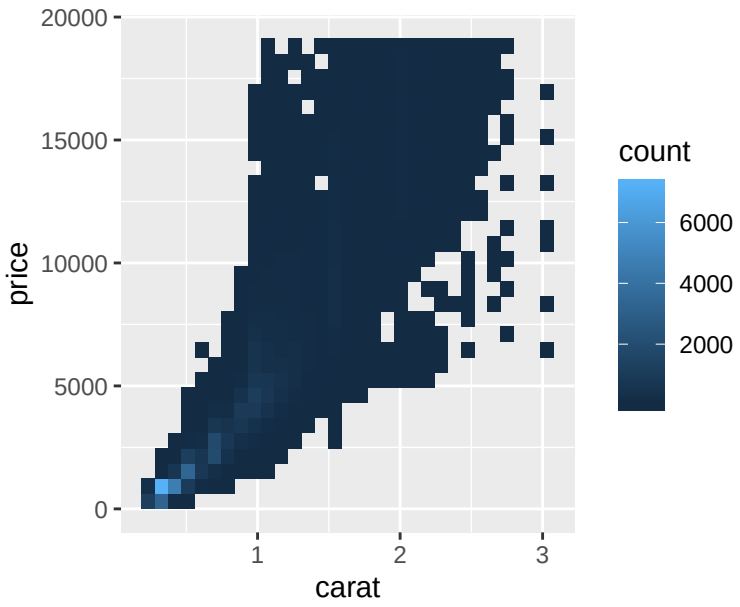
```
require(ggplot2)
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = carat, y = price), alpha = 1 / 100)
```



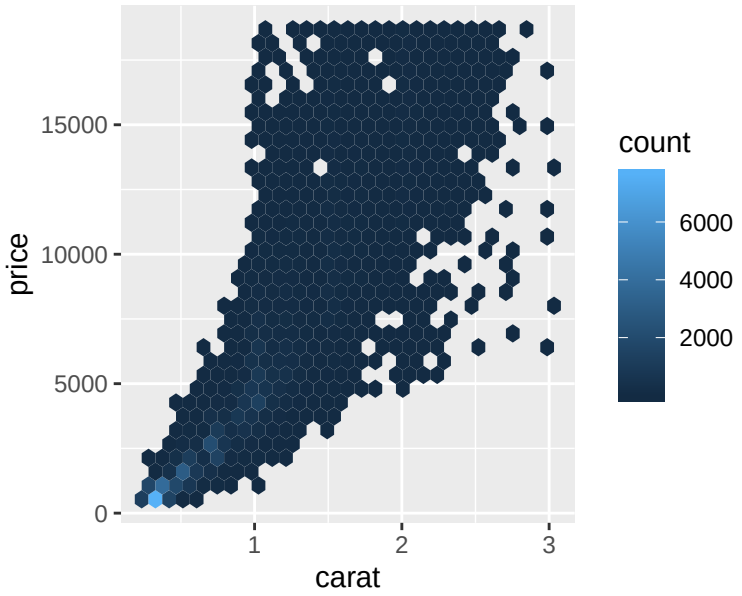
But using transparency can be challenging for very large datasets. Another solution is to use bin. Previously you used geom_histogram() and geom_freqpoly() to bin in one dimension. Now you'll learn how to use geom_bin2d() and geom_hex() to bin in two dimensions.

geom_bin2d() and geom_hex() divide the coordinate plane into 2d bins and then use a fill color to display how many points fall into each bin. geom_bin2d() creates rectangular bins. geom_hex() creates hexagonal bins. You will need to install the hexbin package to use geom_hex().

```
require(ggplot2)
ggplot(data = smaller) +
  geom_bin2d(mapping = aes(x = carat, y = price))
```

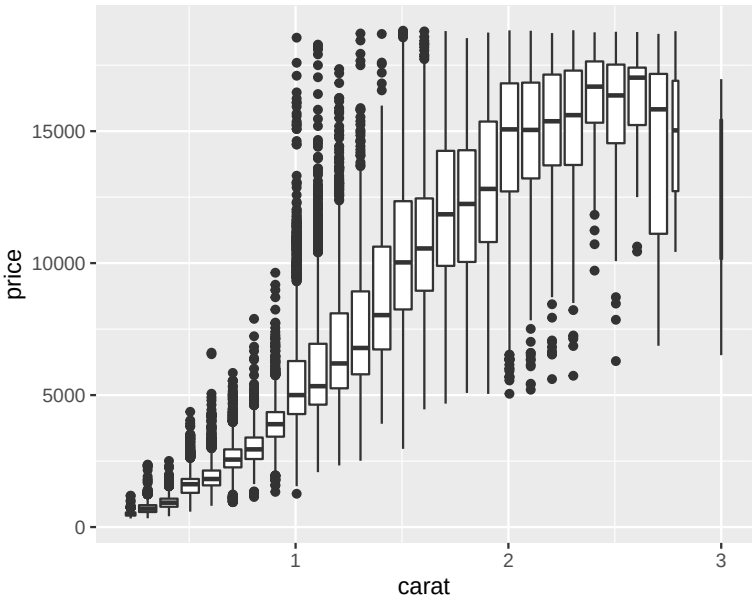



```
require(ggplot2)
# install.packages("hexbin")
ggplot(data = smaller) +
  geom_hex(mapping = aes(x = carat, y = price))
```



Another option is to bin one continuous variable so it acts like a categorical variable. Then you can use one of the techniques for visualising the combination of a categorical and a continuous variable that you learned about. For example, you could bin carat and then for each group, display a boxplot:

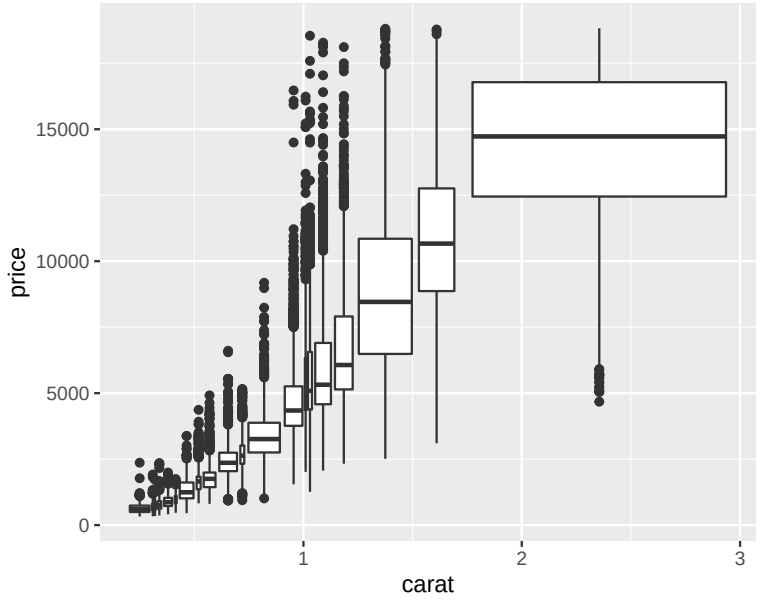
```
require(ggplot2)
ggplot(data = smaller, mapping = aes(x = carat, y = price)) +
  geom_boxplot(mapping = aes(group = cut_width(carat, 0.1)))
```



\texttt{cut_width(x, width)}, as used above, divides x into bins of width width. By default, boxplots look roughly the same (apart from number of outliers) regardless of how many observations there are, so it's difficult to tell that each boxplot summarises a different number of points. One way to show that is to make the width of the boxplot proportional to the number of points with varwidth = TRUE.

Another approach is to display approximately the same number of points in each bin. That's the job of cut_number():

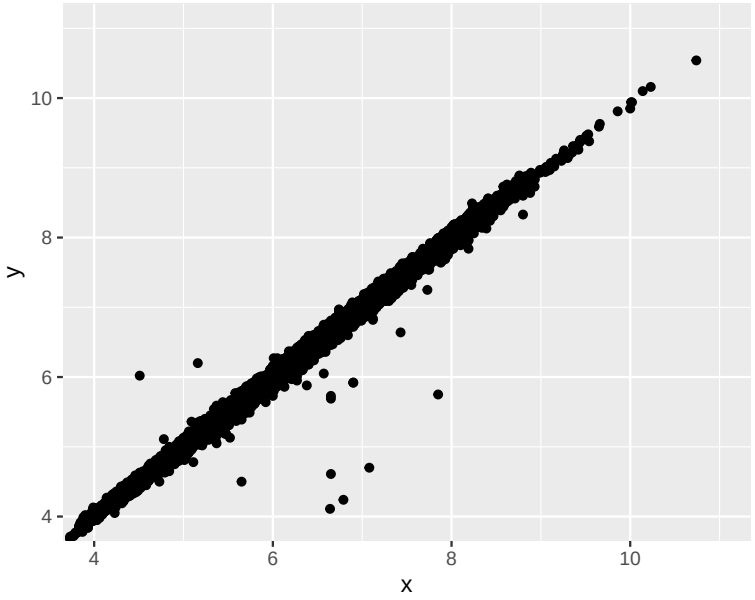
```
require(ggplot2)
ggplot(data = smaller, mapping = aes(x = carat, y = price)) +
  geom_boxplot(mapping = aes(group = cut_number(carat, 20)))
```



7.5.3.1 Exercises

1. Instead of summarising 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 visualisation of the 2d distribution of carat and price?
2. Visualise the distribution of carat, partitioned by price.
3. How does the price distribution of very large diamonds compare to small diamonds? Is it as you expect, or does it surprise you?
4. Combine two of the techniques you've learned to visualise the combined distribution of cut, carat, and price.
5. Two dimensional plots reveal outliers that are not visible in one dimensional plots. For example, some points in the plot below have an unusual combination of x and y values, which makes the points outliers even though their x and y values appear normal when examined separately.

```
require(ggplot2)
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = x, y = y)) +
  coord_cartesian(xlim = c(4, 11), ylim = c(4, 11))
```



Why is a scatterplot a better display than a binned plot for this case?

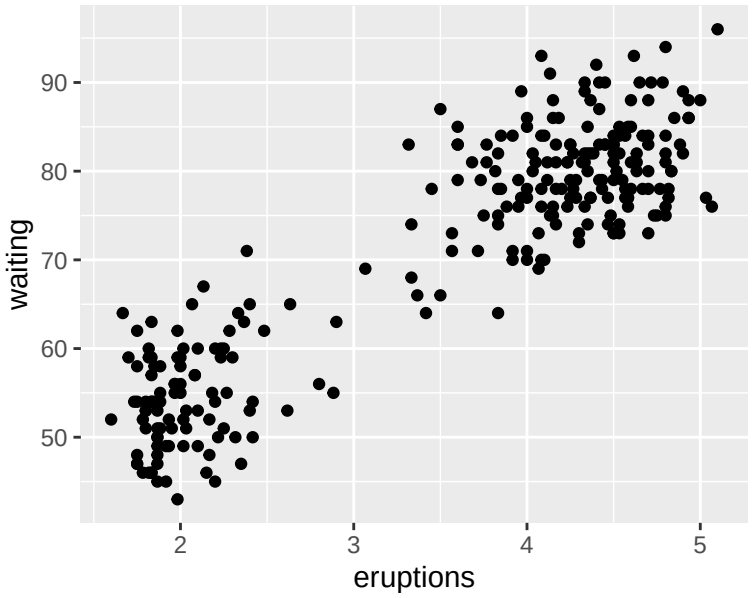
7.6 Patterns and models

Patterns in your data provide clues about relationships. If a systematic relationship exists between two variables it will appear as a pattern in the data. If you spot a pattern, ask yourself:

- Could this pattern be due to coincidence (i.e. random chance)?
- How can you describe the relationship implied by the pattern?
- How strong is the relationship implied by the pattern?
- What other variables might affect the relationship?
- Does the relationship change if you look at individual subgroups of the data?

A scatterplot of Old Faithful eruption lengths versus the wait time between eruptions shows a pattern: longer wait times are associated with longer eruptions. The scatterplot also displays the two clusters that we noticed above.

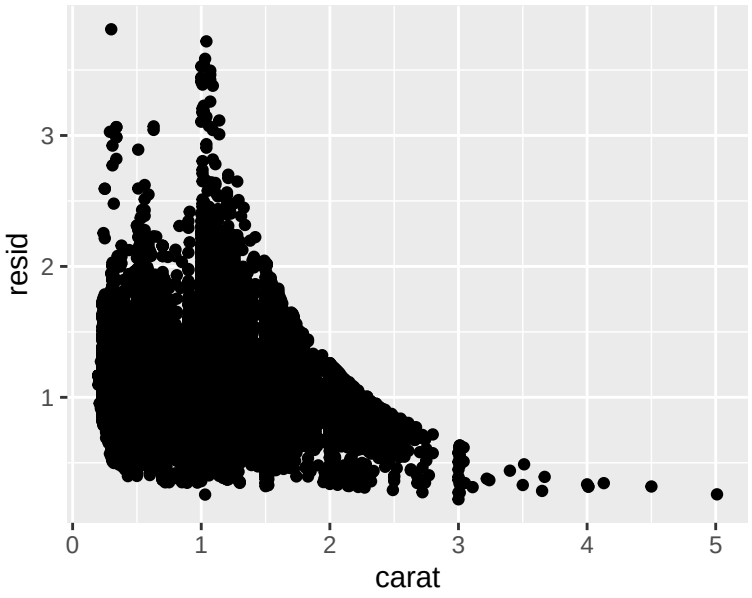
```
require(ggplot2)
ggplot(data = faithful) +
  geom_point(mapping = aes(x = eruptions, y = waiting))
```



Patterns provide one of the most useful tools for data scientists because they reveal covariation. If you think of variation as a phenomenon that creates uncertainty, covariation is a phenomenon that reduces it. If two variables covary, you can use the values of one variable to make better predictions about the values of the second. If the covariation is due to a causal relationship (a special case), then you can use the value of one variable to control the value of the second.

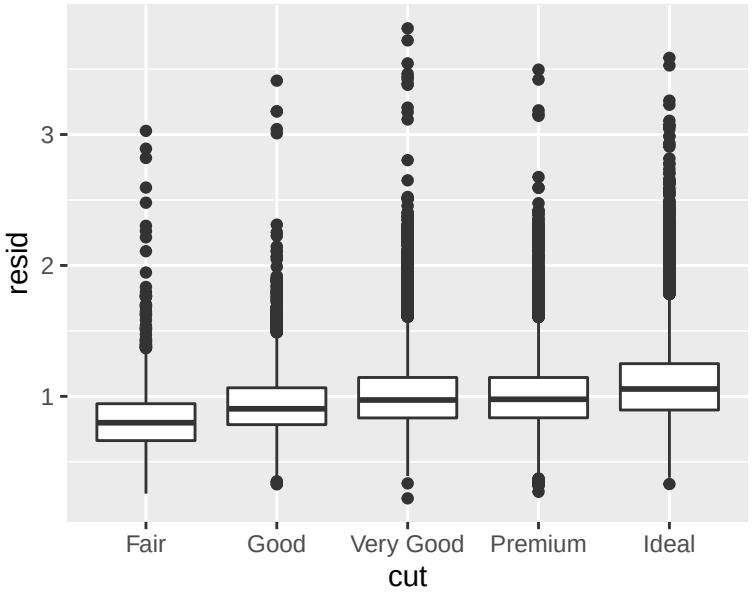
Models are a tool for extracting patterns out of data. For example, consider the diamonds data. It's hard to understand the relationship between cut and price, because cut and carat, and carat and price are tightly related. It's possible to use a model to remove the very strong relationship between price and carat so we can explore the subtleties that remain. The following code fits a model that predicts price from carat and then computes the residuals (the difference between the predicted value and the actual value). The residuals give us a view of the price of the diamond, once the effect of carat has been removed.

```
require(ggplot2)
library(modelr)

mod <- lm(log(price) ~ log(carat), data = diamonds)

diamonds2 <- diamonds %>%
  add_residuals(mod) %>%
  mutate(resid = exp(resid))

ggplot(data = diamonds2) +
  geom_point(mapping = aes(x = carat, y = resid))
```



Once you've removed the strong relationship between carat and price, you can see what you expect in the relationship between cut and price: relative to their size, better quality diamonds are more expensive.

```
require(ggplot2)
ggplot(data = diamonds2) +
  geom_boxplot(mapping = aes(x = cut, y = resid))
```



You'll learn how models, and the modelr package, work in the final part of the book, model. We're saving modelling for later because understanding what models are and how they work is easiest once you have tools of data wrangling and programming in hand.

Final Section 7.7 not included in this version