Exploratory Data Analysis

Data Visualization

Often, the first way that we examine a set of data is visually. There are many different ways that data could be graphed, and in R there are many ways to create those graphs. We will look at several of the most common ways to graph data, and we will look at how to do so using a function called ggplot(), based around a concept called the "grammar of graphics".

The ggplot() function is included in the ggplot2 package.

In RStudio, let's create a new R Markdown document for this section.

Create a code chunk to load the packages we will need: "tidyverse" and "nycflights13".

We will use two subsets of the "flights" data as examples here. We can create a new data frame looking only at flights from Alaska Airlines running the following in a code chunk:

```
alaska_flights <- flights %>%
  filter(carrier=="AS")
```

And we can similarly create a new data frame looking only at flights from Delta Air Lines running the following in a code chunk:

```
delta_flights <- flights %>%
filter(carrier=="DL")
```

Now let's examine the relationship between departure delays and arrival delays for each flight on Alaska Airlines. Use a code chunk to run the following code:

```
ggplot(data = alaska_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point()
```

This has created a scatterplot. We can see the departure delay values on the x axis and the arrival delay values on the y axis.

We can observe a number of things from this graph:

- There is a positive relationship between departure delays and arrival delays. That is, flights that had a longer departure delay tended to also have a longer arrival delay.
- Most of the values for both departure delays and arrival delays are near zero. That is, most flights were relatively close to being on time.
- Some values are both variables are negative. In this case, a negative value must mean that, rather than departing or arriving late, a flight departed or arrived early.
- We received a warning message that 5 rows were removed because of missing values. That means that for five of the flights in our Alaska Airlines dataset, at least one of the two variables we are interested in was missing in the dataset, and therefore couldn't be graphed.

The Grammar of Graphics

We can also see a number of things in this code to illustrate the idea of the "grammar of graphics."

We can think of three fundamental components of a graph:

1. The data being plotted
2. The geometric object being used to plot the data (points, lines, bars, etc)
3. The aesthetic attributes of the geometric object (color, shape, size, position, etc)

We see each of these components referred to separately in the code that we ran.

The data:

```
ggplot(data = alaska_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point()
```

The geometric object:

```
ggplot(data = alaska_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point()
```

The aesthetic attributes:

```
ggplot(data = alaska_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point()
```

In this case, we are creating a graph where the dataset being used is the data on flights from Alaska Airlines, where the geometric objects being used to display the data are points, and where the aesthetics being used are to show departure delays on the x axis and arrival delays on the y axis.

Let's examine ways in which we might modify each of these components.

First, consider the data. Instead of flights from Alaska Airlines, suppose we wanted to look at flights from Delta Air Lines. We could change the data attribute:

```
ggplot(data = delta_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point()
```

Or we could consider using a different geometric object:

```
ggplot(data = delta_flights, aes(x = dep_delay, y = arr_delay)+
  geom_line()
```

Note that this change of geometric objects can change the message that the data appears to convey. Using points highlights each observation as a distinct thing, while using lines suggests some sort of underlying continuous curve. In this case, points would make more sense for the data we are considering.

But if we were in a situation where the x variable represents some sort of continuum that we could be interested in, lines could be more appropriate: for example, if we wanted to look at whether there was a pattern in arrival delays

over the course of the year, we could plot the date along the x axis and use lines to show the changes in arrival delays.

Another change we might make to the geometry would be to make the points partially transparent. When there are many overlapping points, this can often make graphs easier to view. We can do this by adding an argument inside the geom_point() function. The argument "alpha" is used by geom_point() to determine the level of transparency of points:

```
ggplot(data = delta_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point(alpha = 0.2)
```

Finally, we could consider a number of changes to the aesthetic attributes. This could involve mapping different variables than the ones originally selected. Perhaps, rather than looking for a pattern between arrival delays and departure delays, we want to look for a pattern between arrival delays and month:

```
ggplot(data = delta_flights, aes(x = month, y = arr_delay))+
  geom_point(alpha = 0.2)
```

Or we could add additional aesthetic attributes. Maybe we want to also consider how the distance traveled is related to both arrival and departure delays, and want to have the size of each dot represent the "distance" variable:

```
ggplot(data = delta_flights, aes(x = dep_delay, y = arr_delay, size = distance))+
  geom_point(alpha = 0.2)
```

We could also include aesthetic attributes to represent categorical data. In our data set, the flights all originate in one of three airports. We could have these airports represented by different colors in our graph:

```
ggplot(data = delta_flights, aes(x = dep_delay, y = arr_delay, color = origin))+
  geom_point(alpha = 0.2)
```

Alternately, if we wanted a graph that showed these differing origin airports without relying on colors, we could use different shapes:

```
ggplot(data = delta_flights, aes(x = dep_delay, y = arr_delay, shape = origin))+
  geom_point(alpha = 0.2)
```

Creating a Scatterplot

Scatterplots are a type of graph that lets us look at the relationship between two numerical variables.

When we are looking at the relationship between two variables, it is often the case that we are interested in thinking about how we could predict what to expect for one variable based on knowing the other variable. For example, we might be interested in predicting the amount of growth we could see in a plant based on the quantity of fertilizer it is given. Or we might be interested in predicting someone's first year college GPA based on their high school GPA.

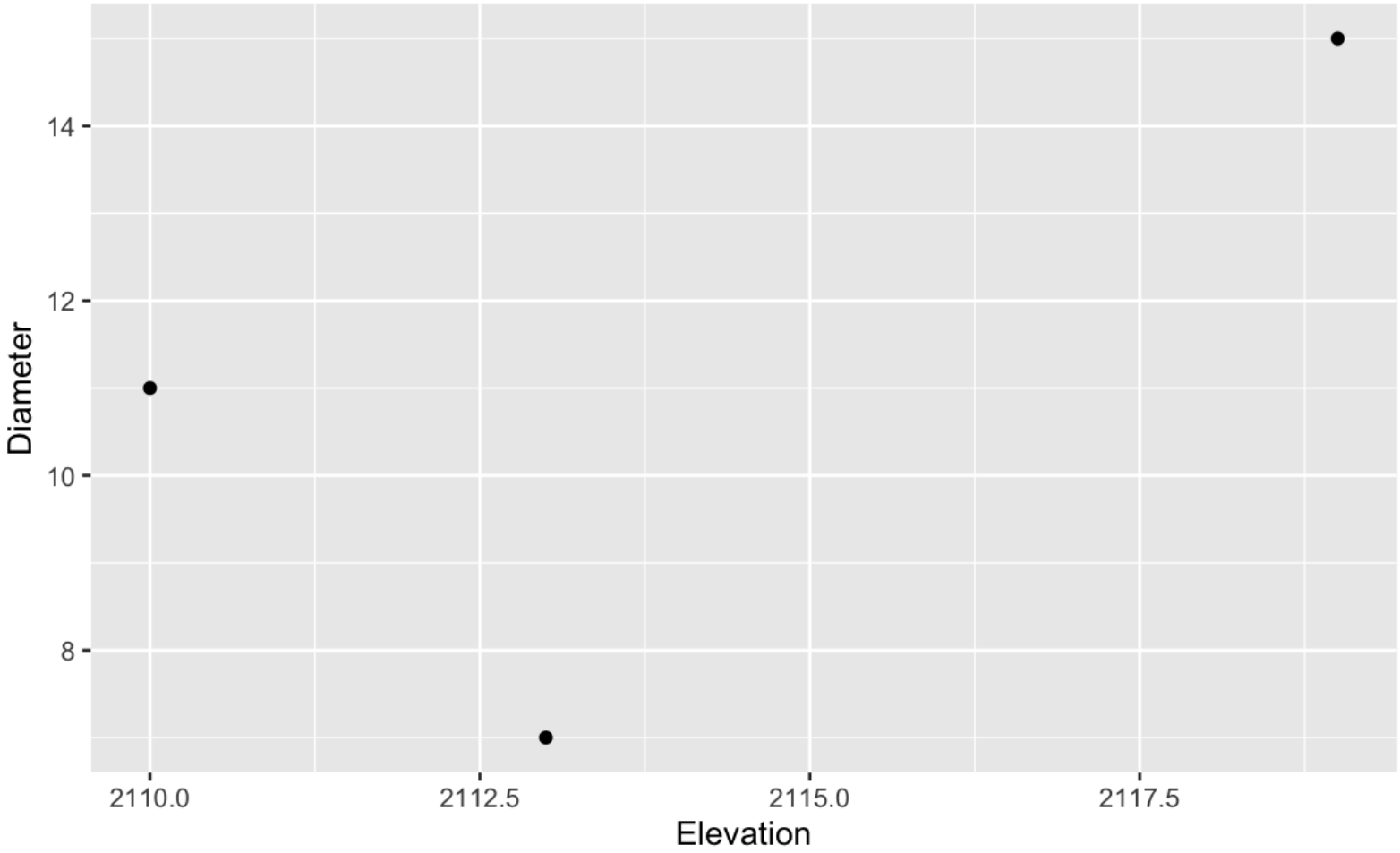
In these situations, we refer to the variable we are interested in predicting as the **response variable**. We may also see if referred to as the **dependent variable**. The variable we are using the make the prediction is referred to as the **explanatory variable** or the **independent variable**.

The response variable will usually be shown on the y axis. We can think of the x axis as representing an input, and the y axis as representing an output.

If we are not thinking of one variable as a function of the other, and instead are purely interested in examining the relationship between two variables, then it doesn't matter which variable is represented on which axis.

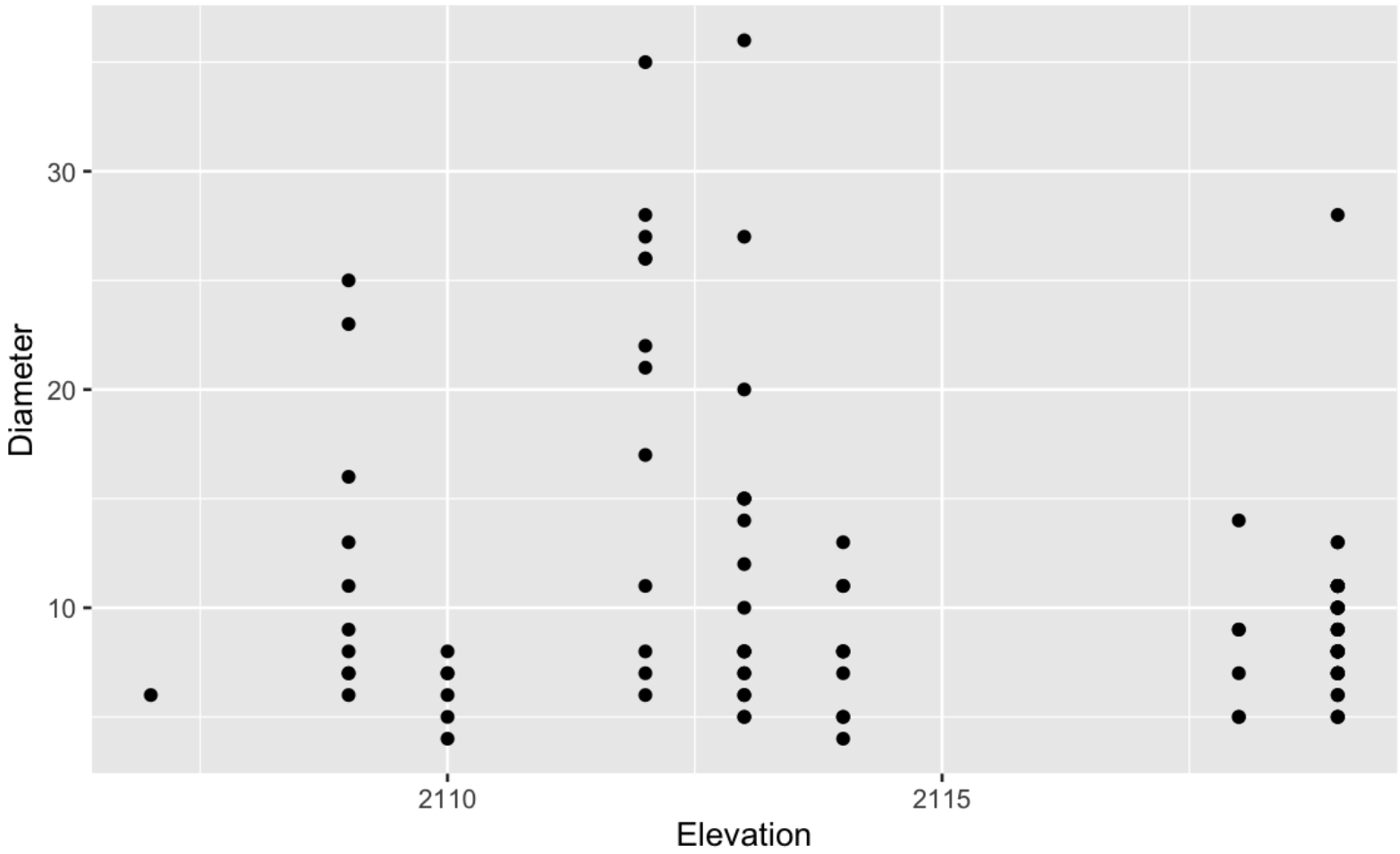
We can visually examine the relationship between two numerical variables via a scatterplot. Each individual is represented as a point in the plot. Values of the response variable are represented along the y axis, and values of the explanatory variable are represented along the x axis.

We could create a graph for trees, looking at diameter as a function of elevation:



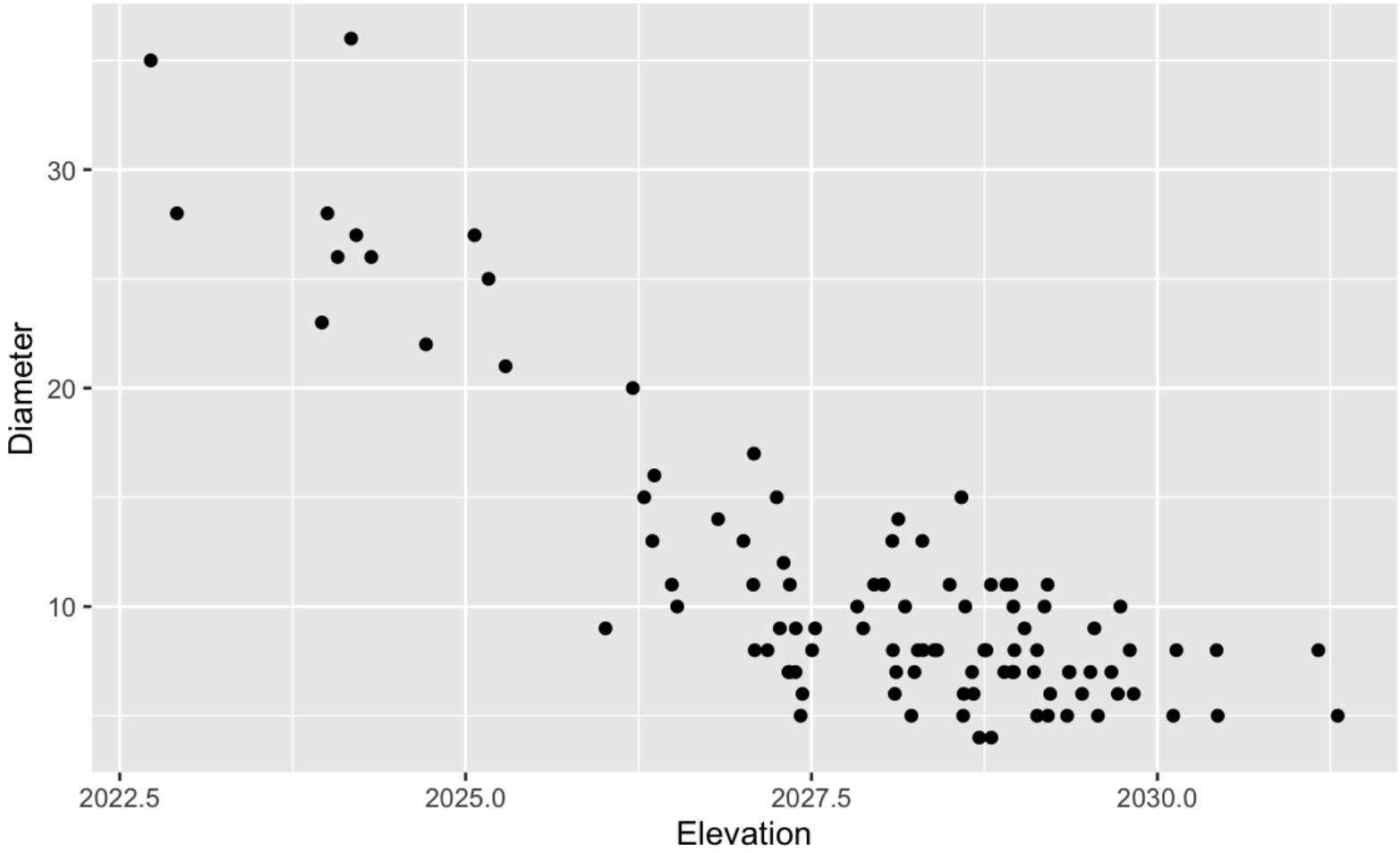
We see that we have three trees represented here.

More often, we are interested in looking at much larger numbers of data points:



In this particular case, we don't see much of a pattern between elevation and diameter.

Suppose we had instead seen the following graph:



This would have shown that there was a relationship - at higher elevations, we tend to see smaller trees.

The key elements in creating a scatterplot a scatterplot are that the geometric objects will be points, and the aesthetic attributes will be the two variables represented on the x and y axes.

When looking at flight departure delays and arrival delays, it might make sense to think about how a departure delay impacts an arrival delay. So we are thinking of arrival delays as a function of departure delays, meaning we will put arrival delays on the y axis and departure delays on the x axis.

We showed above how to create such a scatterplot using the following code:

```
ggplot(data = alaska_flights, aes(x = dep_delay, y = arr_delay))+
  geom_point()
```

We also looked at how we could modify the geometric objects, the points, to be partially transparent using the following code:

```
ggplot(data = delta_flights, aes(x = month, y = arr_delay))+
  geom_point(alpha = 0.2)
```

This allows us to address a problem known as *overplotting* - when points in a graph overlap with one another, making it difficult to fully see what's going on.

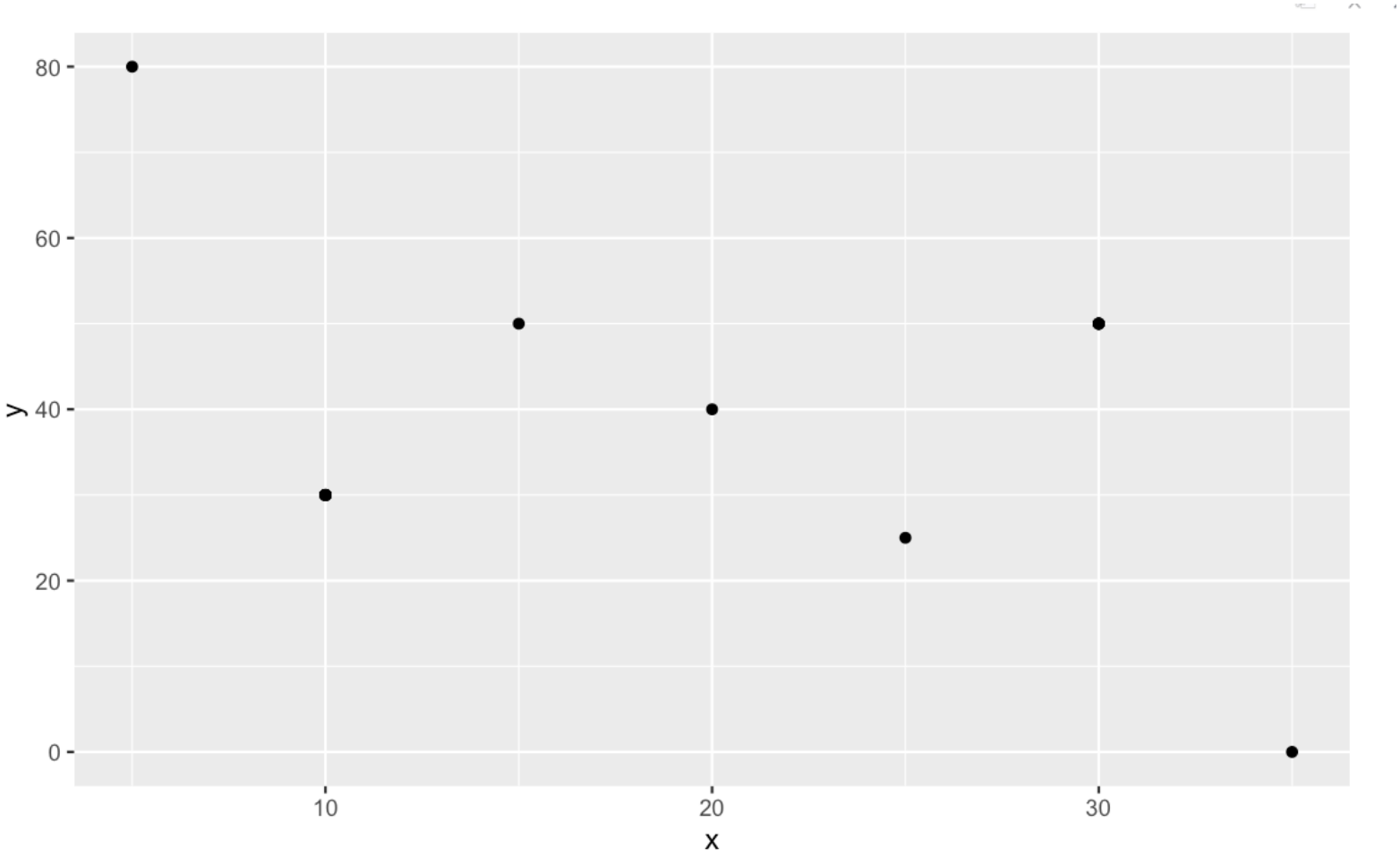
In our data, this occurs because points that are near to one another partially overlap, creating a large black blob. Another way that this could occur would be if we had multiple instances of exactly the same value.

Use the following code chunk to create two vectors, x and y, and then place them into a tibble named "plot.data":

```
x <- c(10,10,10,10,10,10,10,10,10,10,30,30,30,30,30,5,15,20,25,35)
y <- c(30,30,30,30,30,30,30,30,30,30,50,50,50,50,50,80,50,40,25,0)
plot.data <- tibble(x, y)
```

Next, use the following code to create a scatterplot with this data:

```
ggplot(data=plot.data, aes(x=x, y=y))+
geom_point()
```



In this plot, we might decide that there appears to be a bit of a negative relationship between x and y.

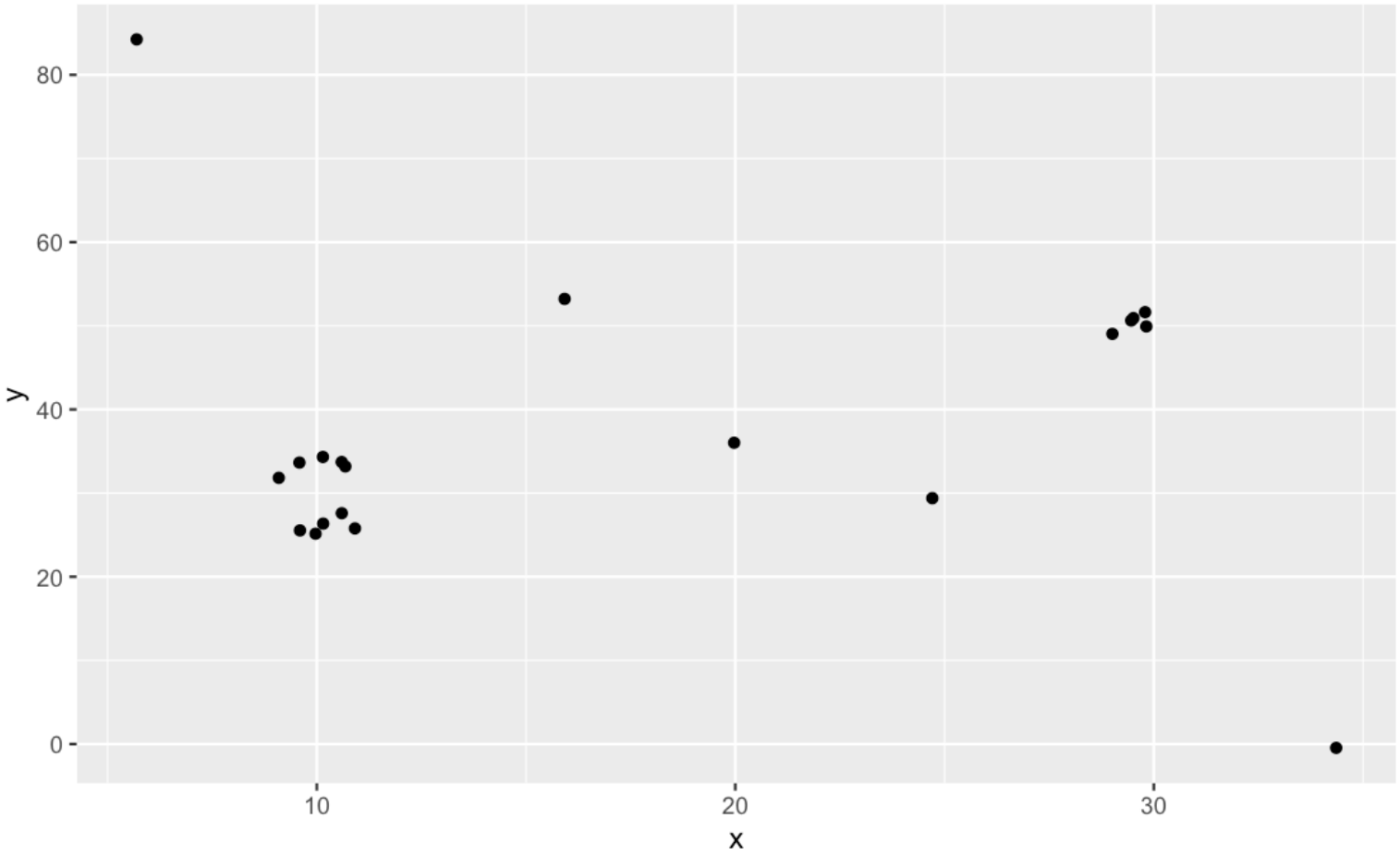
But if we look at the vectors we created, we can see that we are missing an important piece of information - some of these points in our graph represent many data points, and some represent individual data points.

We can address this by adding "jitter" to our graphs - by adding small amounts of noise to our data points to make them no longer perfectly overlap, so that each point will show up near to where it ought to be, but as a distinct point.

In our code, we achieve this by changing the geometric objects from points to jittered points. Use the following code:

```
ggplot(data=plot.data, aes(x=x, y=y))+
geom_jitter(width = 1, height = 5)
```

This should generate a graph that looks something like this:



(Note that the jitter is added at random, so each time you run it you will get a slightly different result.)

In this graph, we can now clearly see that there are many points around x = 10 and around x = 30.

When adding jitter, we have to specify how much randomness to add to our points. These values are controlled by the "width" and "height" options. Play around with different values for them and see how it changes your graph. In general, we want to add enough jitter that we can see all of our points separately, but not so much that our points start looking like they have drastically different values than they actually do.

Note that when you include jitter in a graph, no changes are being made to the underlying data stored in R. The random noise is only being added to the graph, not to the actual dataset.

Creating Linegraphs

Linegraphs are also used to look at relationships between two numerical variables, but in a special case. They are used when the explanatory variable, on the x-axis, is of a sequential nature. Most often this occurs when our explanatory variable is time. To visually indicate that the data should be viewed as a sequence of values, rather than individual distinct data points, we use connected lines rather than points.

In a time plot, the two features we tend to look for are periodic behavior and overall trends.

Much like a scatterplot, a linegraph has two key aesthetic attributes, the two variables represented on the x axis and the y axis. Functionally, the difference between a scatterplot and a linegraph is in the choice of geometric object: rather than using points, a linegraph uses lines.

To illustrate a situation where a linegraph could be a useful visualization tool, we will consider the weather dataset included in the nycflights13 package. To simplify our graphs, we will look at weather data only for Newark, between January 1st and 15th. We can use the following code to create this subset of the weather data:

```
early_january_weather <- weather %>%
  filter(origin == "EWR" & month == 1 & day <= 15)
```

We can then create a linegraph in which the x axis represents time and the y axis represents the recorded temperature:

```
ggplot(data = early_january_weather, mapping = aes(x = time_hour, y = temp)) +
  geom_line()
```

In this graph, we see periodic behavior, where temperature rises and falls corresponding to daytime and nighttime. We also see an overall trend that, as the month progresses, it is tending to become warmer.

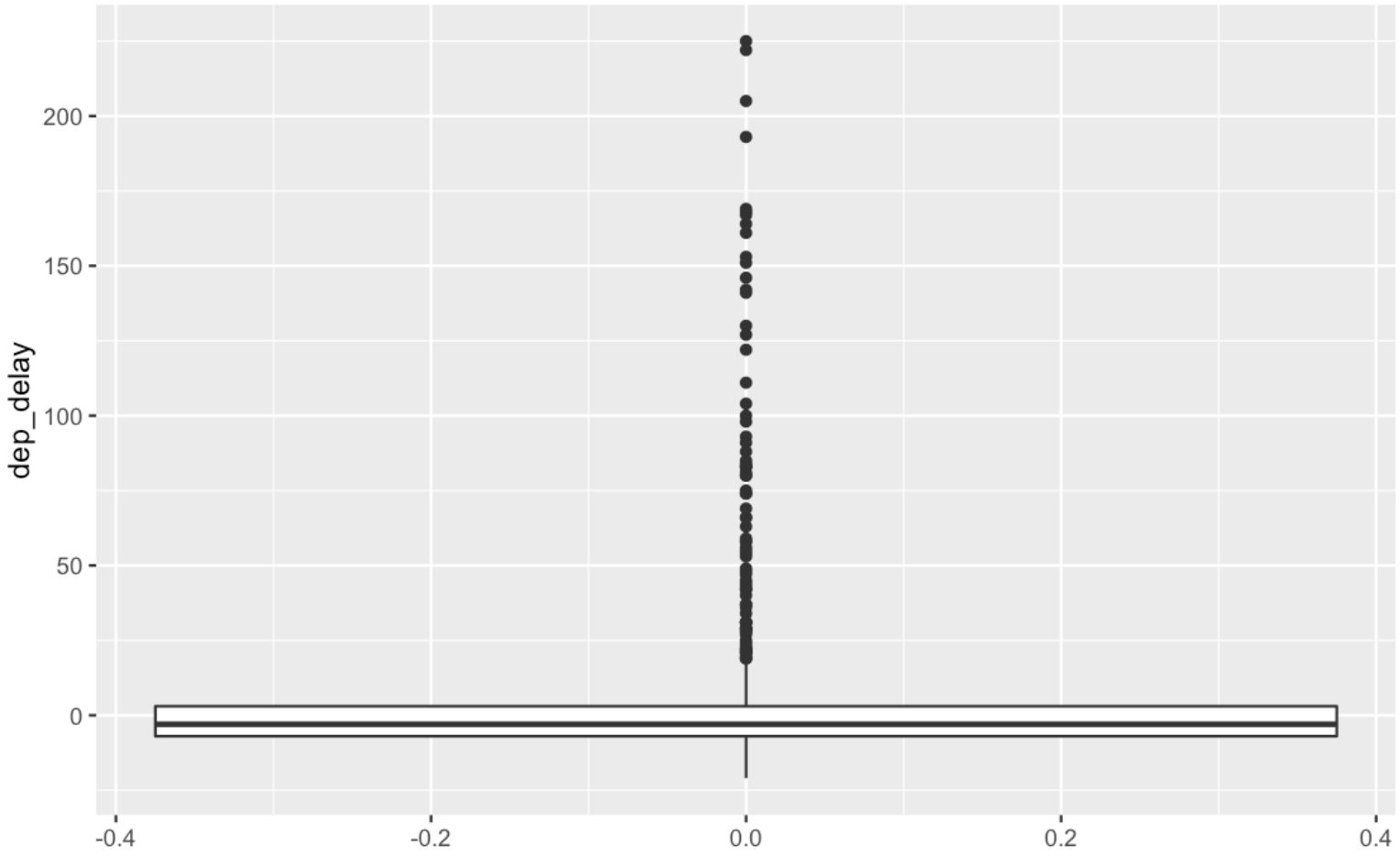
Creating Boxplots

Boxplots are very useful ways to give a quick visual summary of a set of numerical data. This makes them ideal for comparing a numerical variable across several groups.

In a boxplot, the geometric object is the boxplot itself. The primary aesthetic attribute is the numerical variable being summarized, which is represented on the y axis. If we are creating several boxplots to represent different groups, then a secondary aesthetic attribute is the categorical variable being used to define these groups, which will be represented on the x axis.

Suppose we wanted to create a boxplot summarizing the departure delays for all Alaskan Air flights. We could use the following code:

```
ggplot(data = alaska_flights, aes(y = dep_delay)) +
  geom_boxplot()
```



Note that in this graph, there are a very large number of individual dots above the box. Dots represent outliers, observations that are "unusually" far away from the majority of the data. The rules used by R (and other statistical software) to determine which points are outliers are based on what we would expect if we had data coming from a normal distribution - a distribution that is bell-shaped and symmetric. When our data have distributions with other shapes, we can expect to see more observations flagged as outliers. In particular, the more skewed a distribution of data is, the more points we can expect to see flagged as outliers. In this case, from this graph, we can see that most observations involve delays of no more than maybe 20 minutes, but that there are a number of observations involving much longer delays.

A boxplot is based on something called the **five number summary**. The five number summary is a list of values that divide our data into chunks that each represent about 25% of the data. The five numbers are the minimum, the lower quartile, the median, the upper quartile, and the maximum. The lower quartile is defined as a cutoff such that 25% of the data is below it, and 75% is above it. The median is defined as a cutoff such that 50% of the data is below it, and 50% is above it. The upper quartile is defined as a cutoff such that 75% of the data is below it, and 25% is above it.

The box in the middle of the graph goes from the lower quartile to the upper quartile. The line through the box is at the median. So the box contains the middle 50% of the data, with 25% in the area below the median line and 25% in the area above the median line.

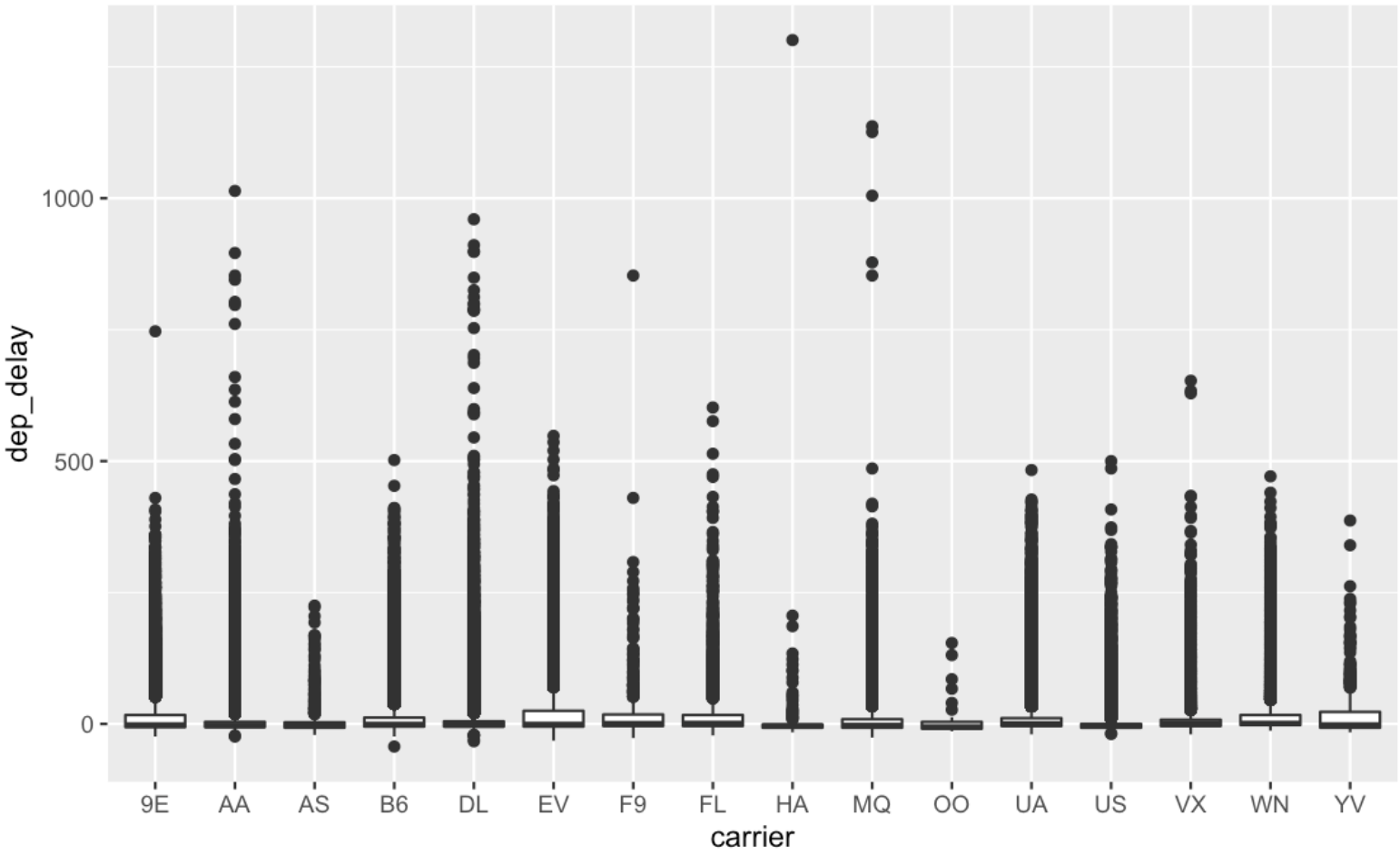
Then there are lines going out from the box. These extend out to the minimum and maximum, with one exception: if there are any outliers in the data, they are displayed as individual points, and the lines will only go out to the largest and smallest non-outlier values.

So the lower line represents the bottom 25% of the data, and the upper line represents the top 25% of the data.

While we can create figures with only a single box, as above, more often we use boxplots to compare between groups. Rather than looking only at Alaskan Air flights, what if we wanted to quickly see how departure delays differ

across carriers? We can run the following code:

```
ggplot(data = flights, aes(x = carrier, y = dep_delay)) +
  geom_boxplot()
```



It is hard to see clearly how these compare, as the outliers occur over such a wide range that the lower end, where most observations occur, is compressed together to the point that we can barely see where the edges of the boxes fall.

There are a number of ways that this sort of issue could be addressed. One approach might be to limit the graph to only show y values within a specified range. This can be done by adding an additional element to our code:

```
ggplot(data = flights, aes(x = carrier, y = dep_delay)) +
  geom_boxplot() +
  ylim(-25,100)
```

This shows how the idea of the grammar of graphics can be extended. In English language grammar, the basic structure of a sentence consists of a noun and a verb. This grammar can then be extended through adjectives, adverbs, object pronouns, clauses, and so on. Similarly, the grammar of graphics can be extended beyond the basic structure through the inclusion of other attributes, in this case a limit on the y values displayed.

In the resulting graph, we can no longer see all of the outlier points. We have zoomed in to focus on the boxes themselves. We can see here, for example, that carriers HA and OO are both tend to have fewer delays than many of the other carriers.

Let's consider another example of boxplots in a situation without so many outliers. Let's return to the weather data that we examined previously with linegraphs. Rather than restricting ourselves to a narrow window of time, let's look at how temperatures varied over the year.

We will construct a boxplot for temperature, with each box representing the temperature values for one month.

Remember that we said at the start of this page that, when we create boxplots, we can separate our data into groups based on a categorical variable. Are the months stored in the weather data as categorical or numerical data?

If we use glimpse() to look at the weather data, we can see that the months are stored as "int" type, which is one way of storing numerical data. Before we can use months to separate our boxplot into groups, we will have to convert it into a categorical, or "factor" variable. We can use the following code to create these boxplots:

```
ggplot(data = weather, mapping = aes(x = factor(month), y = temp)) +
  geom_boxplot()
```

Here, rather than just setting x = month, we have used the factor() function to convert the month variable to a factor variable. Note that this does not change how the month variable is stored in our dataset - it is only creating a temporary version of month in factor form to pass into the ggplot() function.

Creating Histograms

Perhaps the most commonly used tool for visually exploring a numerical variable is a histogram. Possible values of our variable are represented along the x axis, and the heights of bars indicate how often values appeared in our data set.

We construct histograms slightly differently if we are dealing with discrete versus continuous data.

First, we will consider a discrete data set. Suppose I randomly sampled 30 students at Seattle U, and asked each of them how many siblings they have.

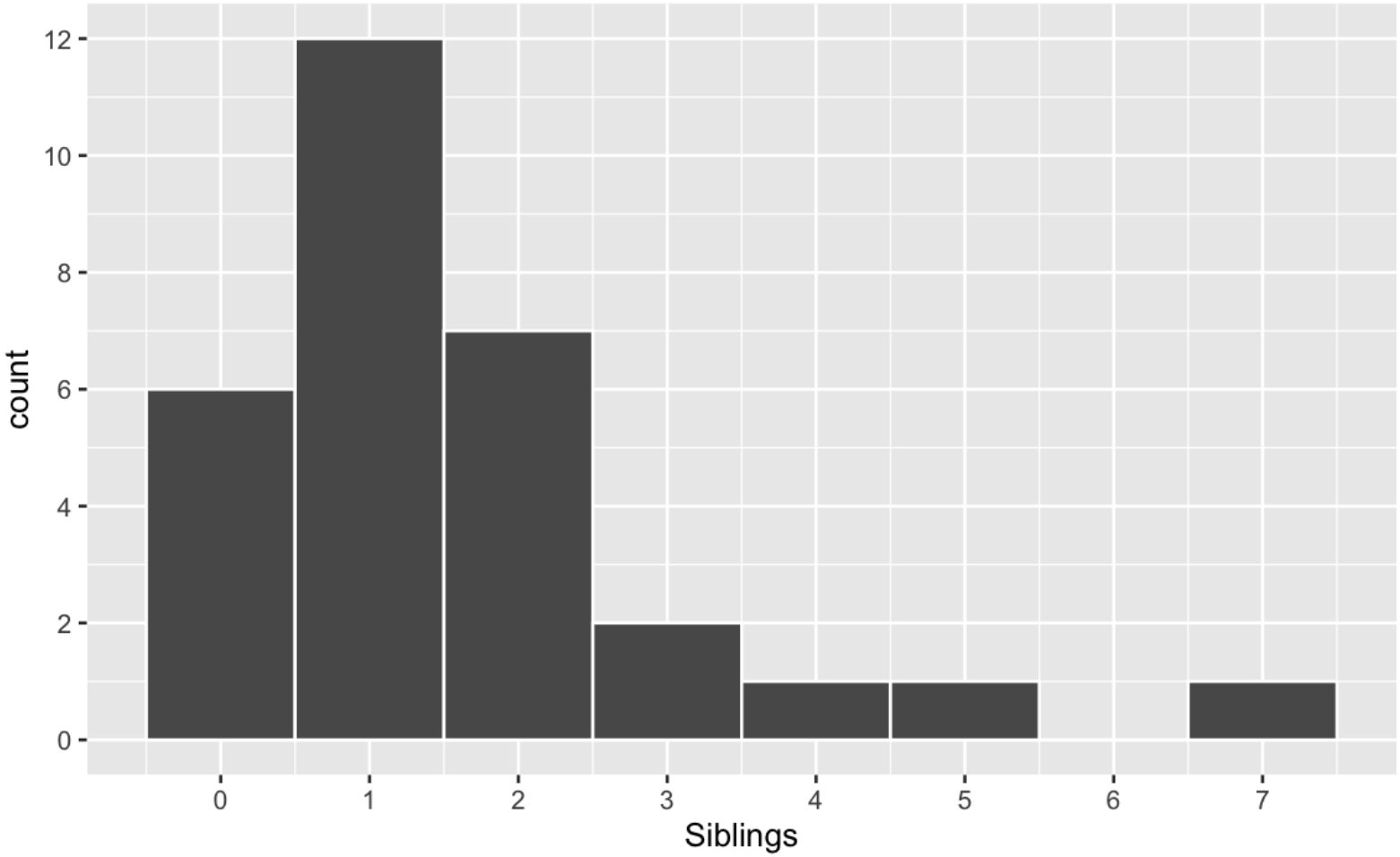
I obtain the following 30 response:

1 0 3 1 1 0 1 1 2 1 2 0 0 1 1 1 2 0 4 5 2 2 3 2 1 2 1 0 7 1

We start by summarizing these responses in a table, counting how often we observed each one:

# of siblings	frequency
0	6
1	12
2	7
3	2
4	1
5	1
6	0
7	1

We then construct a graph with a bar for each possible value of our variable, with the heights of the bars corresponding to these frequencies:



We can quickly see from this graph that 1 was the most common value, with most people having between 0 and 2 siblings.

What changes if we have a continuous random variable instead of a discrete one? We no longer have an individual list of possible values. Instead, we need to create intervals of values, and count how many observations fall into each interval. We will examine how to use ggplot() to create a graph in this case.

The geometric object used will be a histogram, and the primary aesthetic attribute will be the numerical variable represented on the x axis.

If we again consider the temperature data, we can create a graph showing the overall distribution of recorded temperatures using the following code:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram()
```

The default histogram created here might be a bit difficult to read - there are no lines around the bars, making it difficult to see where the edges are. We can add borders to the bars:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(color="white")
```

Note that this did not make the entire bars white, only the outlines around them. The "color" attribute of the histogram object only describes the color of the outline. If we wanted to change the color of the bars themselves, that is referred to as the "fill" attribute.

There are a number of ways to refer to colors in R. The most common way is what we saw in the code above, where we used a word to describe a color. R has a large number of color names programmed into it. We can also specify colors by giving a six-digit hexadecimal number to indicate the levels of red, green, and blue, or through a number of other standard ways of describing colors. **You can find more detail on ways to refer to colors in R here (http://sape.inf.usi.ch/quick-reference/ggplot2/colour)** , as well as finding a list of all of the predefined color names in R.

Here is an example of code in which we use the hexadecimal approach to add a fill color to our histogram as well:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(color="white", fill="#440079")
```

When we create our histogram, we see that R defaulted to using bins of width 30. We have a number of ways that we can change this. We can specify the number of bins to use:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(bins=40, color="white", fill="#440079")
```

Or we can specify the width to be used for each bin:

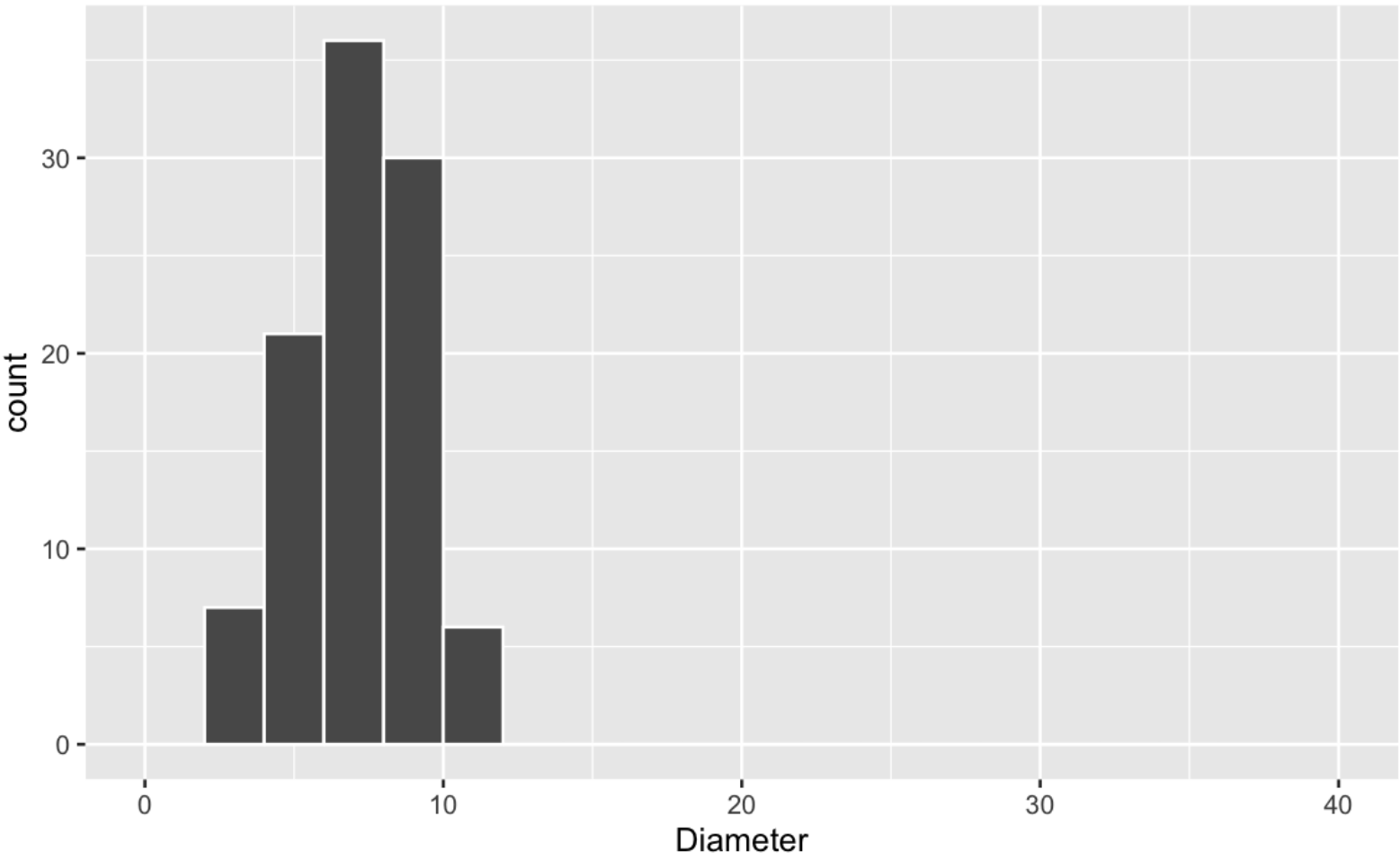
```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(binwidth=10, color="white", fill="#440079")
```

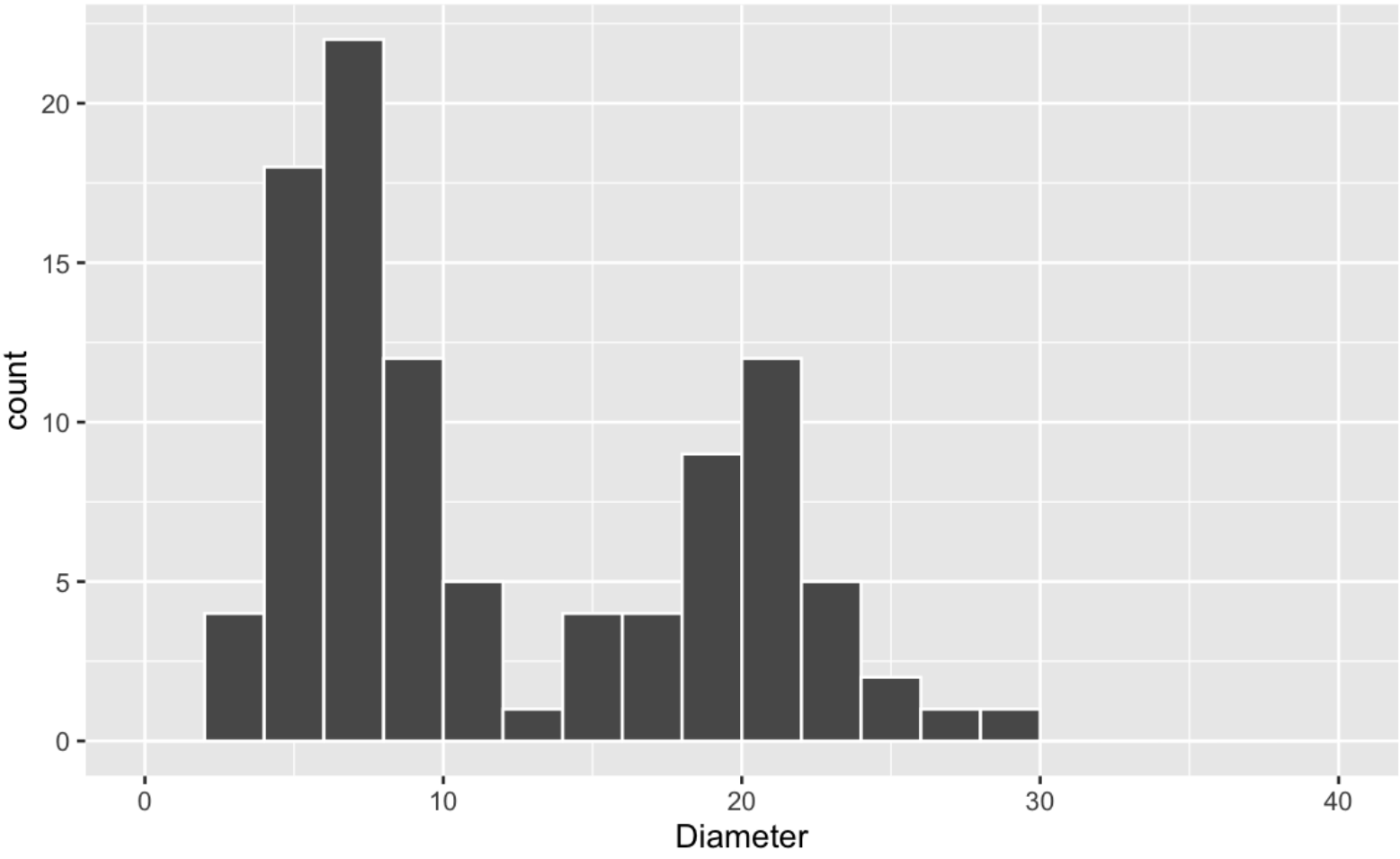
Or we can provide a list of exactly where we want the breaks between bins to be:

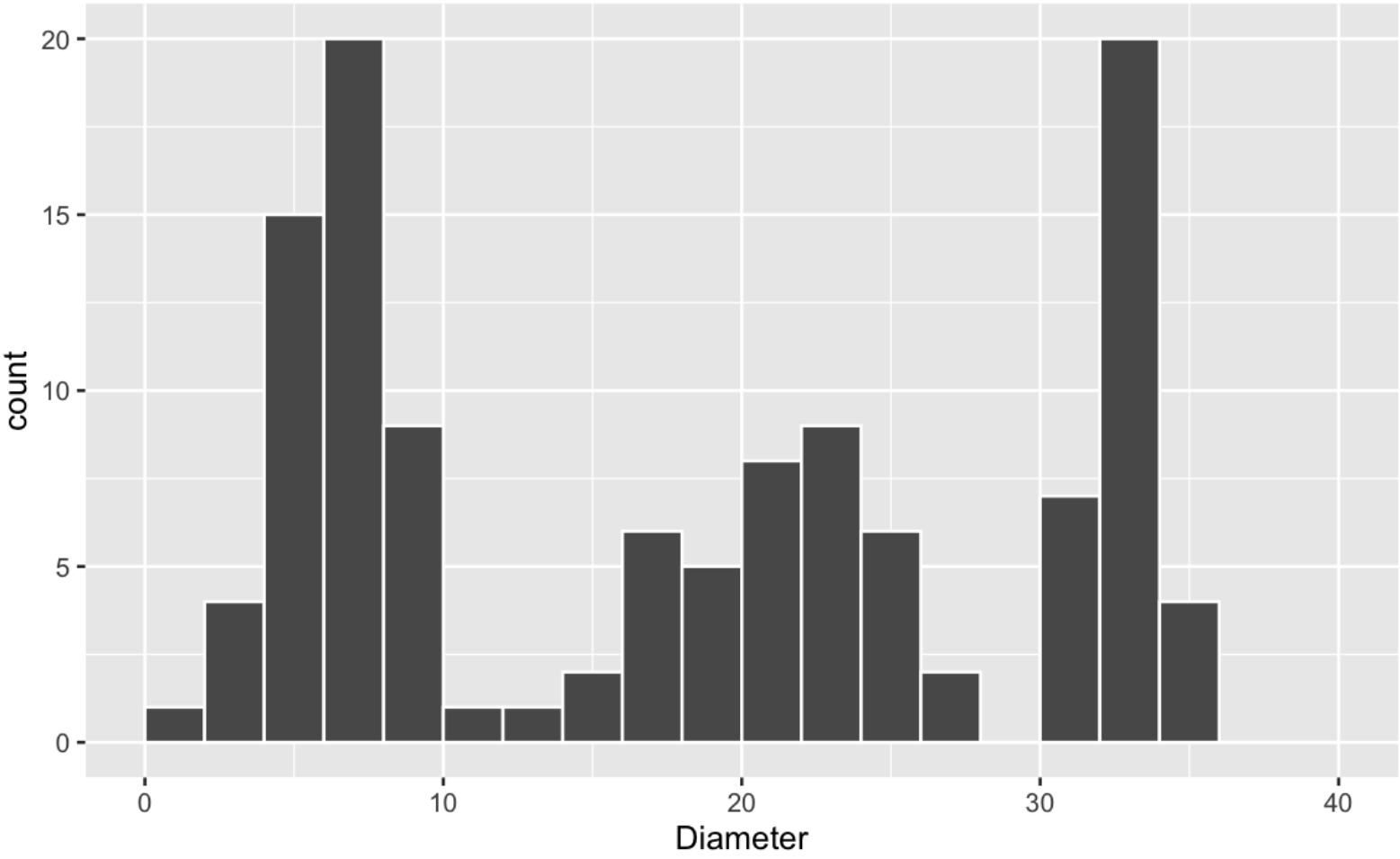
```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(breaks=seq(0, 105, by=5), color="white", fill="#440079")
```

There are two key features that we want to be aware of when we examine a histogram. The first is known as modality. This refers to the number of distinct peaks in the histogram.

Consider the following three histograms:


count
30
20
10
0
0
10
20
30
40
Diameter


count
20
15
10
5
0
0
10
20
30
40
Diameter

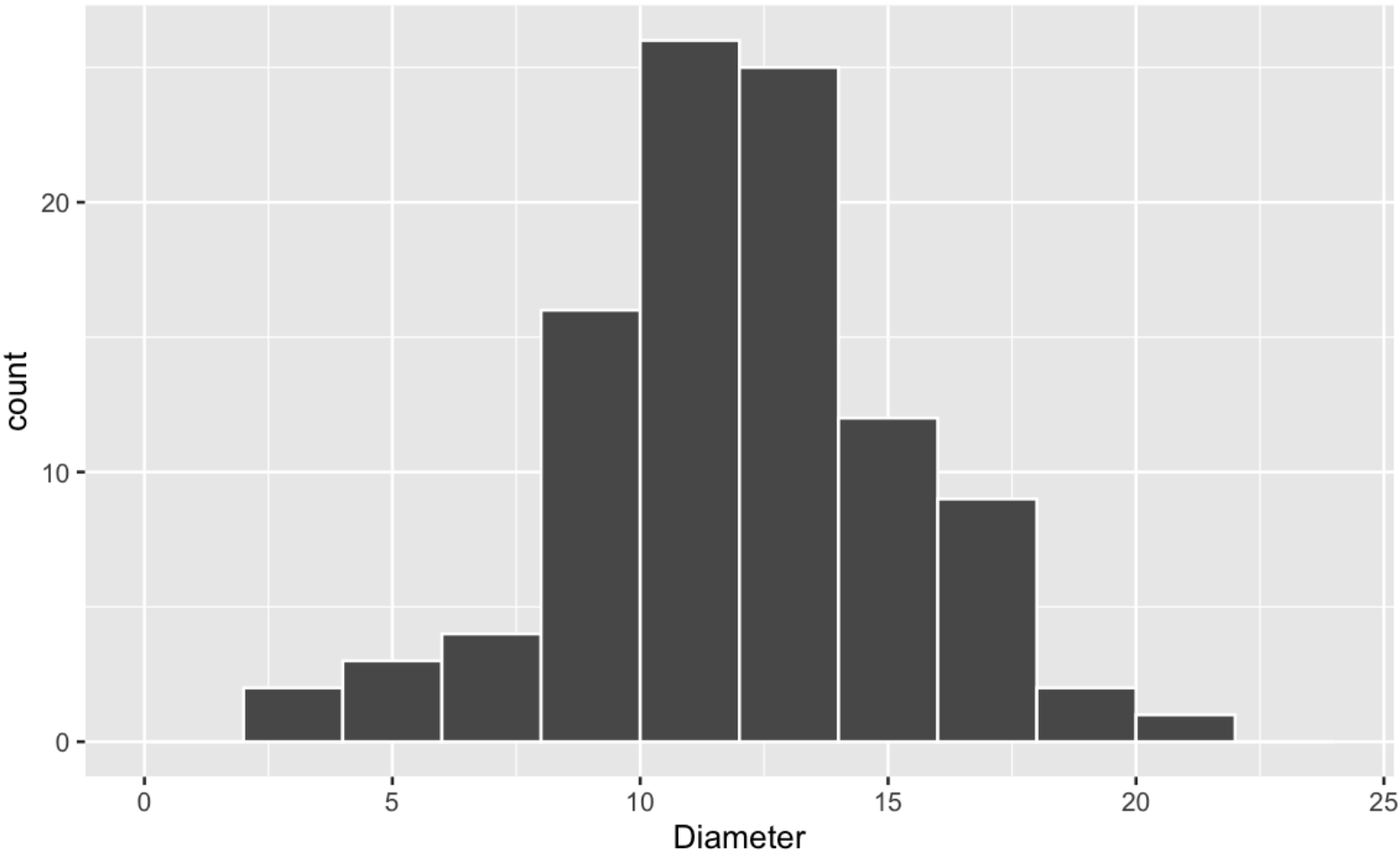


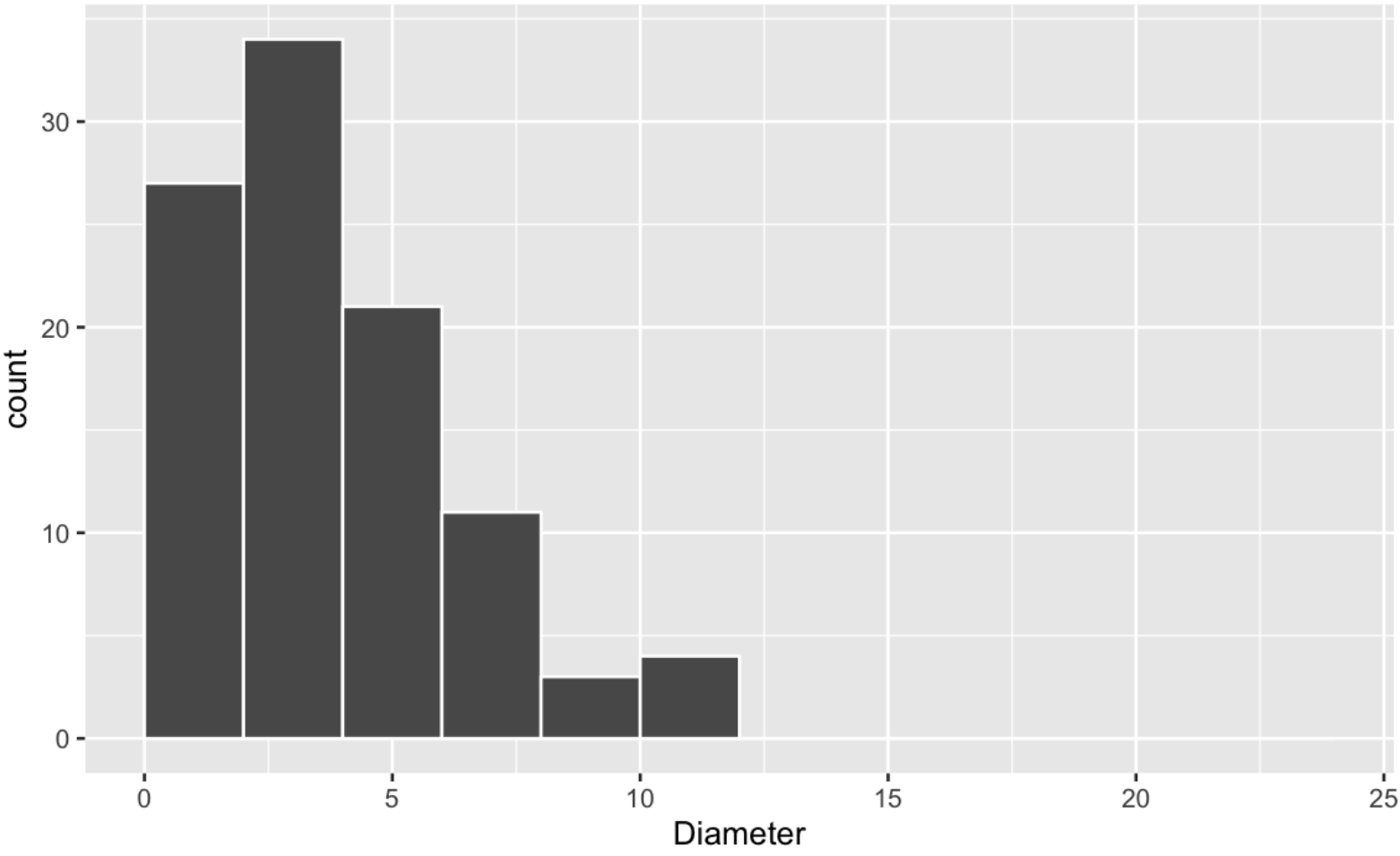
In the first case, there is a single peak. We refer to this sort of graph as **unimodal**. In the second case, there are two distinct peaks. We refer to this sort of graph as **bimodal**. And in the third case, there are more than two distinct peaks. We refer to this sort of graph as **multimodal**.

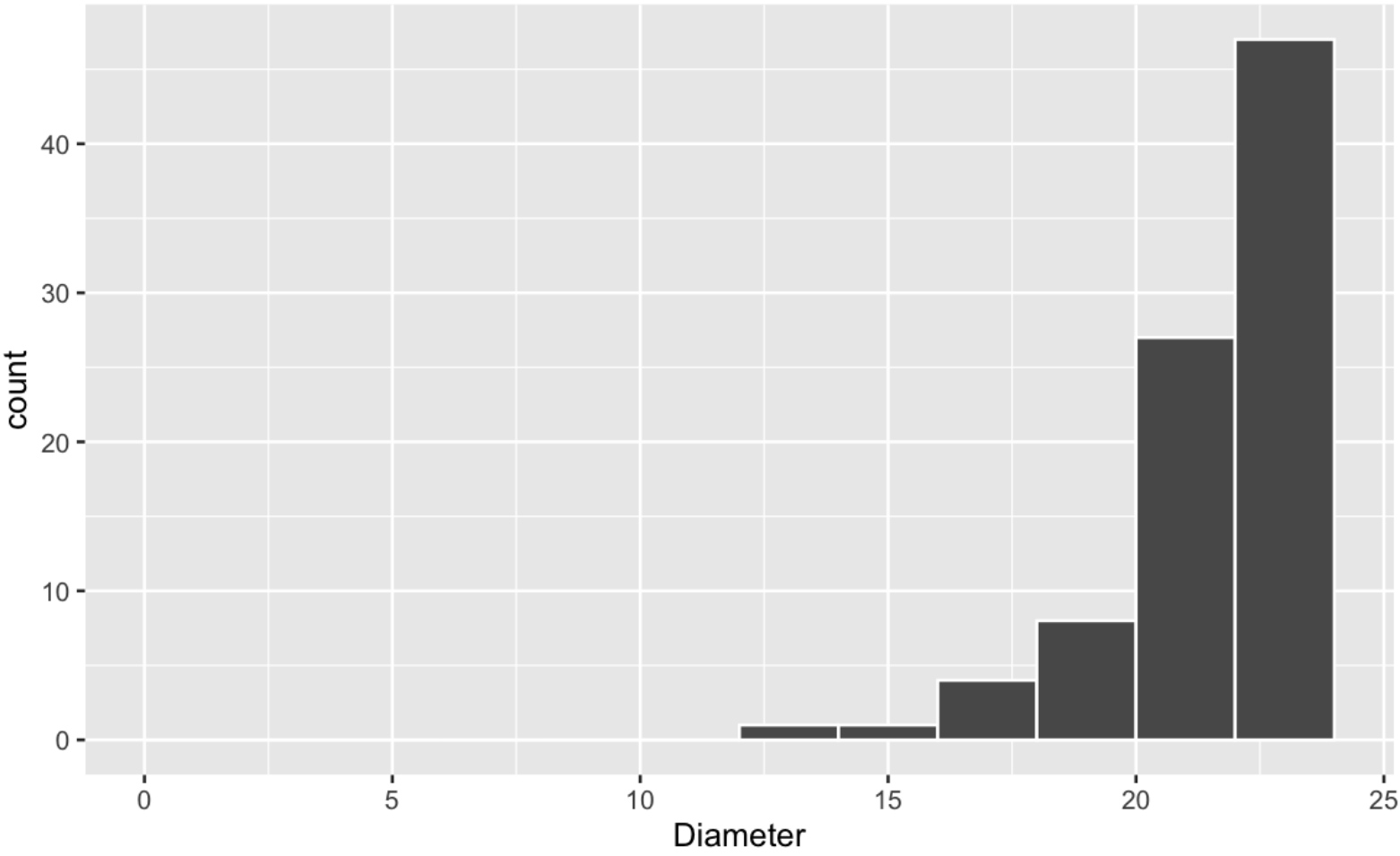
This is an important feature to look for in histograms because, if we see more than one peak in our graph, that can often indicate that there are really two or more distinct subgroups within our data, and that perhaps we should be analyzing them separately. If we saw a histogram like the second or third one above, we would want to look through our data to try to understand why there might be distinct groups. Suppose this was a graph for tree diameters. Do the groups correspond with different tree species? With different elevations, or regions?

We may not always have enough information in our data set to answer this sort of question, but we should always see if we can find an explanation.

The other feature that we look for in our histograms is skewness. This has to do with how symmetrical or asymmetrical our distribution is. Again, consider three possible histograms:


count
20
10
0
0
5
10
15
20
25
Diameter


count
30
20
10
0
0
5
10
15
20
25
Diameter



In the first graph, the distribution is approximately **symmetric** - if we were to cut it near its peak, one half looks about like a mirror image of the other.

In the second graph, if we were to cut it in half near its peak, we would see that the right side stretches out much further than the left side. We refer to this sort of graph as **right-skewed** or **positively skewed**.

In the third graph, we see just the opposite. If we were to cut it near its peak, the left side would stretch out much further than the right. We refer to this sort of graph as **left-skewed** or **negatively skewed**.

Unlike modality, skewness doesn't necessarily tell us much of immediate interest about our data. However, many common statistical techniques only work for symmetric data. We need to check our data early on to determine whether we will be able to use these techniques.

When we have data that is skewed, we sometimes apply some sort of transformation to our data so that the result looks approximately symmetric. Maybe our original data is skewed, but if we were to take the log of our data, the result would look symmetric. Or we could square our data, or take a square root.

In general, when data is positively skewed, it can be useful to either take a log of the data, or some root of the data (square root, cubed root, etc). When data is negatively skewed, it can be useful to transform the data by raising to some power greater than one (squaring the data, cubing the data, etc).

Facets

Any time that we are constructing graphs, we may find it useful to create a set of graphs, rather than a single graph, by dividing our data into groups. This comes up especially often with histograms. We can once again extend the

grammar of graphics to now specify multiple graphs divided up based on some variable. For example, if we wanted to divide our temperature histograms based on months, we could use the following code:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(breaks=seq(0,105, by=5), color="white", fill="#440079") +
  facet_wrap(~ month)
```

We might want to specify more exactly how to break our graphs over multiple lines. Within the facet_wrap() function, we can specify how many rows or columns we want by using the nrow or ncol attributes:

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(breaks=seq(0,105, by=5), color="white", fill="#440079") +
  facet_wrap(~ month, nrow=4)
```

Creating Barplots

A barplot is the most common tool used for visualizing categorical data. In a barplot, each category is represented by a bar, and the height of each bar represents either the count or proportion of observations falling into each category.

Visually, there is one key difference in how histograms tend to be presented relative to barplots: in a histogram, we tend to draw the bars so that they are touching one another, whereas in a barplot, the bars are kept separate from one another. This is to help visually emphasize, in a barplot, that we are not looking at any numerical scale along the x axis, but rather are looking at a set of distinct categories.

Barplots can be created in one of two ways, depending on how our data is formatted. If we have data such as the flights dataset, where each row represents a single observation, and we have one or more columns representing categorical variables, we can generate a barplot using the geometric object bar, and the aesthetic attribute of the categorical variable on the x axis:

```
ggplot(data = flights, mapping = aes(x = carrier)) +
  geom_bar()
```

Here, we have created a bar graph showing how many flights there were from each carrier.

In other circumstances, our data may be formatted differently. We might have data where one column lists the possible categories, and another column lists counts of how many observations were in each of those categories. We can create an example tibble formatted in this manner:

```
fruits_counted <- tibble(
  fruit = c("apple", "orange"),
  number = c(3, 2)
)
```

If we want to create a barplot for data formatted in this manner, we will use columns as our geometric objects, and our aesthetic attributes will be the categories on the x axis and our counts on the y axis:

```
ggplot(data = fruits_counted, mapping = aes(x = fruit, y = number)) +
geom_col()
```

We could modify our graphs to examine more than one categorical variable at once. Perhaps, in addition to carriers, we want to examine the airport of origin. One option would be to create a stacked barplot where each bar is divided into different colors:

```
ggplot(data = flights, mapping = aes(x = carrier, fill = origin)) +
  geom_bar()
```

We previously looked at the "fill" option when discussing histograms. There, we included it as an attribute of our geometric object. Here, instead, it is part of the aesthetic attributes, because it is now being used to encode information about a variable.

Rather than stacking the different origin airports in a single bar per carrier, we might find it useful to arrange them side by side as distinct bars. We can achieve that by modifying our geometric object to now specify that the bars will have their positions arranged next to one another:

```
ggplot(data = flights, mapping = aes(x = carrier, fill = origin)) +
  geom_bar(position = position_dodge(preserve = "single"))
```

Numerical Summaries

In addition to exploring data visually, it is common to look at numerical summaries.

There are two common ways of describing how a numerical variable is centered: the **mean** and the **median**.

The **mean** is also often referred to as the average. The **median** is often referred to as a "typical" value.

With sample data, we find the mean by summing up all of the values, and dividing by how many data points we had. We can write this mathematically:

$$\bar{x} = \frac{1}{n} \sum x_i$$

Suppose we had recorded the diameters of thirteen Jeffrey Pine trees:

7, 12, 15, 3, 8, 11, 8, 35, 12, 9, 13, 8, 11

We can calculate the sample mean:

$$\bar{x} = \frac{1}{13} \cdot (7 + 12 + 15 + 3 + 8 + 11 + 8 + 35 + 12 + 9 + 13 + 8 + 11) = 11.7$$

To find the median, we first have to arrange our values in ascending order:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

Then we find the middle value in the list:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

We can use $\tilde{x}$ to denote the median (notation for medians is not consistent, you will see different people or textbooks use different notation). So we would say $\tilde{x}$ = 11.

What if we had had an even number of data points? Suppose we also collected data on 12 Ponderosa Pines:

25, 28, 33, 33, 34, 36, 37, 40, 44, 45, 50, 53

We would take the two middle values, and average them:

25, 28, 33, 33, 34, 36, 37, 40, 44, 45, 50, 53

So for this set of data, $\tilde{x}$ = 36.5.

Why do we have two different ways of measuring the center of a data set? In fact, there are more than two, but the mean and median are most common. When we are picking a measure of center, we want to pick the value that is, on average, closest to our data points. What value is closest depends on how we define, or measure, closeness. If we use squared distance as our measure of closeness, then the mean turns out to be the best fit. If we use absolute distance, then the median turns out to be the best fit. Using other measures for closeness could result in other quantities being the best fit to describe the center.

On a more practical level, these measures behave differently from one another. Let's look at an example to understand this. Suppose I am in a room with four other people. Our heights are as follows (arranged from shortest to tallest):

5'3", 5'7", 5'8", 6'0", 6'2"

Before we can do calculations with these numbers, we need to convert them into inches:

63", 67", 68", 72", 74"

The mean for these values is 68.8", and the median is 68".

Now, suppose Mechagodzilla walks into the room.

Mechagodzilla is 180 feet tall. That's 2160".

So now our data set is as follows:

63", 67", 68", 72", 74", 2160"

How will that change our numbers?

The median is now 70". The mean is 417.3".

The mean is more impacted by extreme values than the median. When we have skewed data, the mean will be pulled out more in the direction of the long tail of the distribution. It will not give a value that is representative of what we typically see. This is why, when we see reports about skewed data, we tend to see it summarized in terms of the median (for example, when discussing incomes, folks tend to talk about median incomes, because the distribution of incomes is positively skewed).

On the other hand, if we have data that is approximately symmetric, the mean and the median will tend to be very similar to one another. In this situation, another property becomes important. Suppose, rather than just looking at a single group of five people, I took one hundred samples of five people each. Each time, I calculated both the mean height and the median height for the group. What we would find is that the mean is more consistent from sample to sample than the median is. This makes it more useful as an estimate, because there's less uncertainty in our estimate if we use the mean. For this reason, if we have data that is approximately symmetric, we will tend to describe it in terms of the mean.

After we describe how our data is centered, we next might want to describe how spread out, or variable, our data is.

We have seen how we use two different measures of center, depending on our situation:

- If our data is approximately symmetric, we use the mean to measure center
- If our data is skewed, we use the median to measure center

Consequently, we will also have two ways to measure spread. We will first consider how to measure spread around a mean. We will call this spread the **standard deviation**.

If we have approximately symmetric data, we use the mean to measure the center. To talk about spread, we can look at how far away from the mean values tend to be. Recall that we said that the mean arises as a measure of center if we measure distance in terms of squared distance. To stay consistent with that, we will look at the typical squared distance around the mean. For any individual observation, its distance away from the mean would be $(x_i - \bar{x})^2$.

It might feel like the obvious thing to do would be to take the average of these distances. That is almost, but not quite, what we will end up doing.

We calculate the **variance** of a sample as follows:

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

Rather than dividing by n, we divide by n-1. There are a few ways to understand why this is. The essential issue is that what we are working with here is sample data. That is, we don't actually know the values for the entire population we are interested in, we only have data from a subset, which we are using to estimate some larger population. Rather than measuring distance from the sample mean, $\bar{x}$, what we really want to be measuring is distance from the population mean, μ. But we don't know μ, so we estimate it with $\bar{x}$.

However, since $\bar{x}$ was based on only our specific sample data points, it will, on average, tend to be a little bit closer to those points than μ would have been. Using some probability theory concepts (beyond the scope of what we want to focus on in this course), it can be shown that if we just took the average of the $(x_i - \bar{x})^2$ values, our estimate would be off by a factor of $\frac{n-1}{n}$. Dividing by n-1 instead of n corrects for this.

This connects to a slightly more general idea that will come up in some of our later sections in this class - something called **degrees of freedom**. This is an issue that arises when estimating variability. In general, the degrees of freedom refers to what we need to divide by instead of n in order to adjust for bias in our estimated variance. In many situations, the degrees of freedom will be equal to the number of data points we have minus the number of other things we first had to estimate before we could estimate the variance.

In this situation, we had n data points, and we first had to estimate 1 other thing, the mean. So our degrees of freedom is n-1. In an upcoming chapter, we will look at a situation where, rather than describing variability around a mean, we want to describe variability around a line. In order to describe a line, we need to estimate 2 things: a slope and an intercept. In this situation, our degrees of freedom will be n-2.

Let's return to one of our example data sets from before. We can calculate the variance for the heights of five people as follows:

$$s^2 = \frac{1}{5-1} \cdot ((63 - 68.8)^2 + (67 - 68.8)^2 + (68 - 68.8)^2 + (72 - 68.8)^2 + (74 - 68.8)^2) = 18.7$$

What are the units on this? All of our observations were in inches. So the units here would be inches squared. We will often take the square root of the variance to get a measure that is in our original units. We refer to this quantity as the **standard deviation**:

$$s = \sqrt{s^2}$$

For our height data, s = 4.3"

The standard deviation gives us a measure of how far away from the mean individual observations will tend to be.

What if we have skewed data? In that case, we measure the center via the median. If our data is skewed, it is spread out further on one side of the median than on the other. Consequently, a single measure of how far values tend to be spread out doesn't really make sense. Instead, we will describe spread through something known as the **five number summary**.

We will use five numbers to divide our data into intervals that each contain approximately one quarter of the data points. To do this, we will need to calculate two new values: the **lower quartile** and the **upper quartile**.

The lower quartile, denoted Q_1, is a cutoff with one quarter of the observations below it. The upper quartile, denoted Q_3, is a cutoff with one quarter of the observations above it. We find these values by taking the median of the upper and lower halves of the data.

Let's consider the Jeffrey Pine data again:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

We found the median to be 11:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

To find the lower quartile, we take all of the values up to the median :

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

And then we find the median of those numbers:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

In this case, Q_1 = 8

Similarly, to find the upper quartile, we take all of the values from the median and up:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

And then we find the median of those numbers:

3, 7, 8, 8, 8, 9, 11, 11, 12, 12, 13, 15, 35

In this case, $Q_3 = 12$.

The total five number summary is the minimum, Q_1, $\tilde{x}$, Q_3, the maximum. For the Jeffrey Pine data, the five number summary would be: 3, 8, 11, 12, 35. Note that when we split the data up here, the median itself in included in both the upper and lower halves.

We could go through a similar process with the Ponderosa Pine data:

25, 28, 33, 33, 34, 36, 37, 40, 44, 45, 50, 53

In this case, the median is in between two values, which means that every value will be included in either the upper or lower half. We might visually look at it this way:

25, 28, 33, 33, 34, 36 | 37, 40, 44, 45, 50, 53

We look at the lower half to find $Q_1 = 33$:

25, 28, 33, 33, 34, 36 | 37, 40, 44, 45, 50, 53

And the upper half shows us that $Q_3 = 44.5$:

25, 28, 33, 33, 34, 36 | 37, 40, 44, 45, 50, 53

So for the Ponderosa Pines, our five number summary would be 25, 33, 36.5, 44.5, 53.

We saw in our introduction to R that we can use the summarize function to create a table of numerical summaries. We can also directly use functions like mean(), median(), sd(), percentile(), min(), max(), etc, to calculate a single numerical summary at a time:

```
x <- c(63, 67, 68, 72, 74)
mean(x)
median(x)
sd(x)
```

Normal Quantile Plots

We noted earlier that many common statistical tools are only applicable for a set of data that follows a normal distribution. It can be useful, then, to be able to analyze whether a set of data appears to follow a normal distribution. We can construct something known as a normal quantile plot (also often called a quantile-quantile plot or a q-q plot). This will be a type of scatterplot. The observed data values will be represented on the y-axis, and the expected values if we had data coming from a normal distribution will be represented on the x-axis.

We will illustrate how to construct such a plot in R, using simulated data from various distributions:

First, let's generate a random set of observations from a normal distribution. We will use the function rnorm() to do this.

```
normal_simulation <- rnorm(20, 50, 3)
```

This creates a simulated set of 20 values from a normal distribution with a mean of 50 and a standard deviation of 3.

Now we will create the normal quantile plot, and draw a reference line. If our data follows a normal distribution, we would expect the observed values to line up with the expected values.

```
qqnorm(normal_simulation)
qqline(normal_simulation)
```

How far off from the line might we expect points to be? Try repeatedly generating simulated samples and plotting the normal quantile plot to see what variety of results you could see.

What if we have data that does not come from a normal distribution?

Let's consider a common skewed distribution, the exponential distribution. We will generate a sample of 20 observations from an exponential distribution with parameter $\lambda = .02$:

```
exponential_simulation <- rexp(20, .02)
qqnorm(exponential_simulation)
qqline(exponential_simulation)
```

We see here that, at both the lower end and the upper end, our observed values are larger than what we would expect from a normal distribution. This is consistent with the positive skew of the exponential distribution.

Let's consider one additional distribution, the t distribution. This distribution is symmetrical, like the normal distribution, but with heavier tails. We will simulate a set of 20 observations from a t distribution with 4 degrees of freedom.

t_simulation <- rt(20, 4)
qqnorm(t_simulation)
qqline(t_simulation)

We can see that, due to the heavier tails in the t distribution, observed values at the lower end are smaller than we would expect from a normal distribution, while observed values at the upper end are larger than we would expect from a normal distribution.