A Quick Introduction to Multilevel Bayesian Models for Linguistic Researchers

Santiago Bareda

2020-12-31

Introduction

This book will focus on a conceptual introduction to multilevel Bayesian models. I am going to talk about math as little as possible, so I am going to avoid giving details unless necessary. My goal is to convey some of the more important and useful concepts necessary to understand these models and to use them in your own work. I hope to set you up in a position to 'fill in the blanks' as necessary later.

I'm going to try to provide intuitive explanations for statistical models that rely on understanding the figures we use represent and interpret our data and models. The book assumes some understanding of R (only to understand the code), but none of statistics. I don't really 'explain' many basic things, for example, how to calculate a standard deviation? Why is it calculated like it is? The reason for this is that first, there are many better resources for that. And second, I am trying to make this quick, to go from 0 to understanding and interpreting Bayesian multilevel models in 10 weeks.

The book is specifically aimed for linguists only because it focuses on the sorts of research designs frequently used by linguists. However, the models and principles outlined in this book are used and useful in many domains.

If you have any comments or suggestions please let me know, and if you find any errors definitely let me know!

Inspecting a single sample of values

In this chapter I am going to present an introduction to some fundamental statistical concepts (i.e, probability, likelihood). We will discuss how to use these concepts to make inferences about our observations.

Data and research questions

Voice fundamental frequency (f0, related to pitch) is a very important cue in speech communication. It relates to phoneme identification, prosody, and to the communication of social and indexical information (speaker gender, age, . . . etc.).

We are going use a well-known data set, the Hillenbrand et al. (1995) data of Michigan English. We are going to focus on a single vector (f0), representing the f0 produced by a set of female speakers.

```
url1 = "https://raw.githubusercontent.com/santiagobarreda"
url2 = "/stats-class/master/data/h95_vowel_data.csv"
## read data from my Github page
h95 = read.csv (url(paste0 (url1, url2)))
# select the 'f0' vector, for women only (speaker type = 'w')
f0s = h95[['f0']][h95$type == 'w']
```

These speakers represent a sample from a larger population. The sample is a finite set of observations that you actually have. The population is the larger group of possible observations that you are *actually* interested in. For example, Hillenbrand et al. collected this data not to study these speakers in particular, but instead to make inferences about Michigan speakers more generally.

Similarly, we want to answer a few basic questions about the population of female speakers from Michigan, not about the sample itself:

1) What is the average f0 of the whole *population* likely to be?

2) Can we set bounds on likely mean f0 values based on the data we collected?

The second point is crucial. First, our sample will never exactly match the population. But it should be *representative* of it, meaning it should not be *too* far off from the real mean (otherwise, it is likely not a good sample from that population). For example, the mean f0 in the data we will discuss below is 220 Hz. This seems to suggest that, for example, a *true* mean of 100 Hz is unlikely. Is a true mean of 150 Hz also unlikely? What about 190 Hz?

Below we will discuss how to measure the spread and location of a set of numbers, and how to establish likely and credible values for the variable.

Inspecting the central location and spread of values

On the left, values are plotted according to the order they appear in the vector. This is not very useful. On the right, the same values have been ordered from low to high.

```
par (mfrow = c(1,2), mar = c(1,4,1,1))
plot (f0s, xaxt='n',xlab='', ylab = 'f0', pch = 16, col = skyblue)
plot (sort (f0s), xaxt='n',xlab='',ylab='f0 (sorted)', pch=16, col = deepgreen)
```

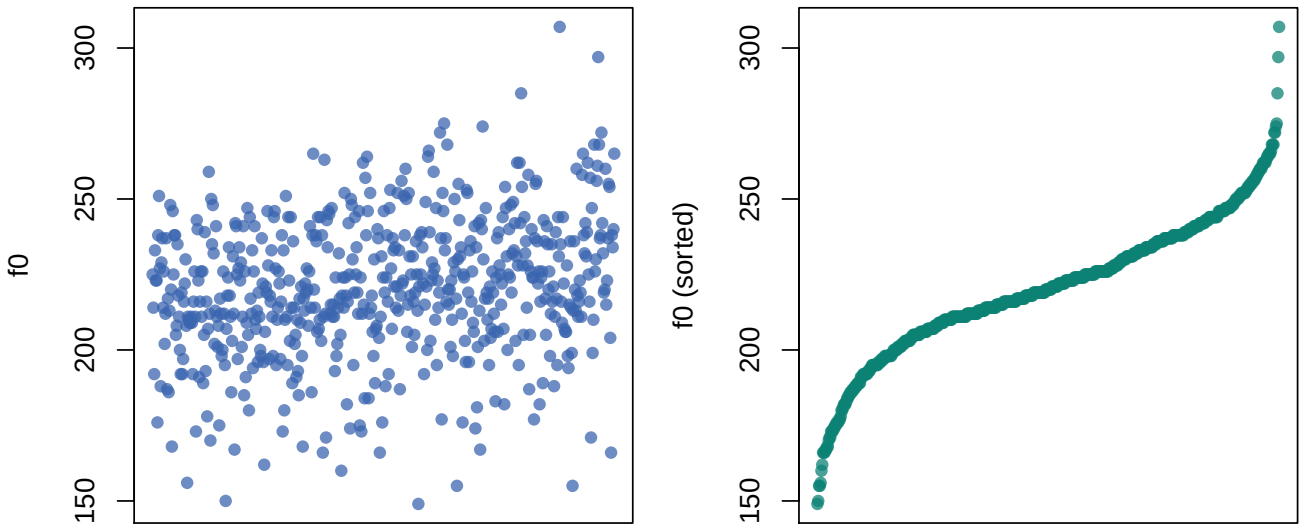


Figure 1: Initial plot of values.

We can easily find descriptive statistics like the sample mean ($\bar{x}$), the sample standard deviation (s_x), and important quantiles for this sample of values. The quantiles below correspond to the values of ordered observations, like in the right plot above. The 0% quantile is the smallest observation, while 100% is the highest. Any other quantile is found by ordering the observations and selecting the observation that is higher than x% of the sample values. For example, the 50% quantile (the median) is higher than 50% of values, and the 25% quantile is higher than 1/4 of the values in the sample.

```
## calculate basic descriptive statistics
mean (f0s)
```

```
## [1] 220.401
```

```
sd (f0s)
```

```
## [1] 23.22069
```

```
quantile (f0s)
```

```
##      0%     25%     50%     75%    100%
## 149.00 207.00 220.00 236.25 307.00
```

We can look at the distribution of speakers. In the top row, points indicate individual productions, and are jittered along the y axis to make them easier to see. In the middle row we see a histogram of the same data. The histogram gives you the count of observations in each bin. In the bottom row we see a box plot of the same data. The edges of the box correspond to the 25 and 75% quantiles of the distribution, and the line in the middle of it corresponds to the median. As a result, 50% of observations are contained in the box.

```
par (mfrow = c(3,1), mar = c(1,4,1,1), oma = c(4,0,0,0))
plot (f0s, jitter (rep(1,length(f0s))), xlim = c(140, 320), ylim = c(.95,1.05),
      yaxt='n',ylab='', pch = 16, col = yellow)
hist (f0s,main="", col = teal)
boxplot (f0s, horizontal = TRUE, ylim = c(140, 320), col = coral)
mtext (side =1, outer = TRUE, text = "f0", line = 2.5)
```

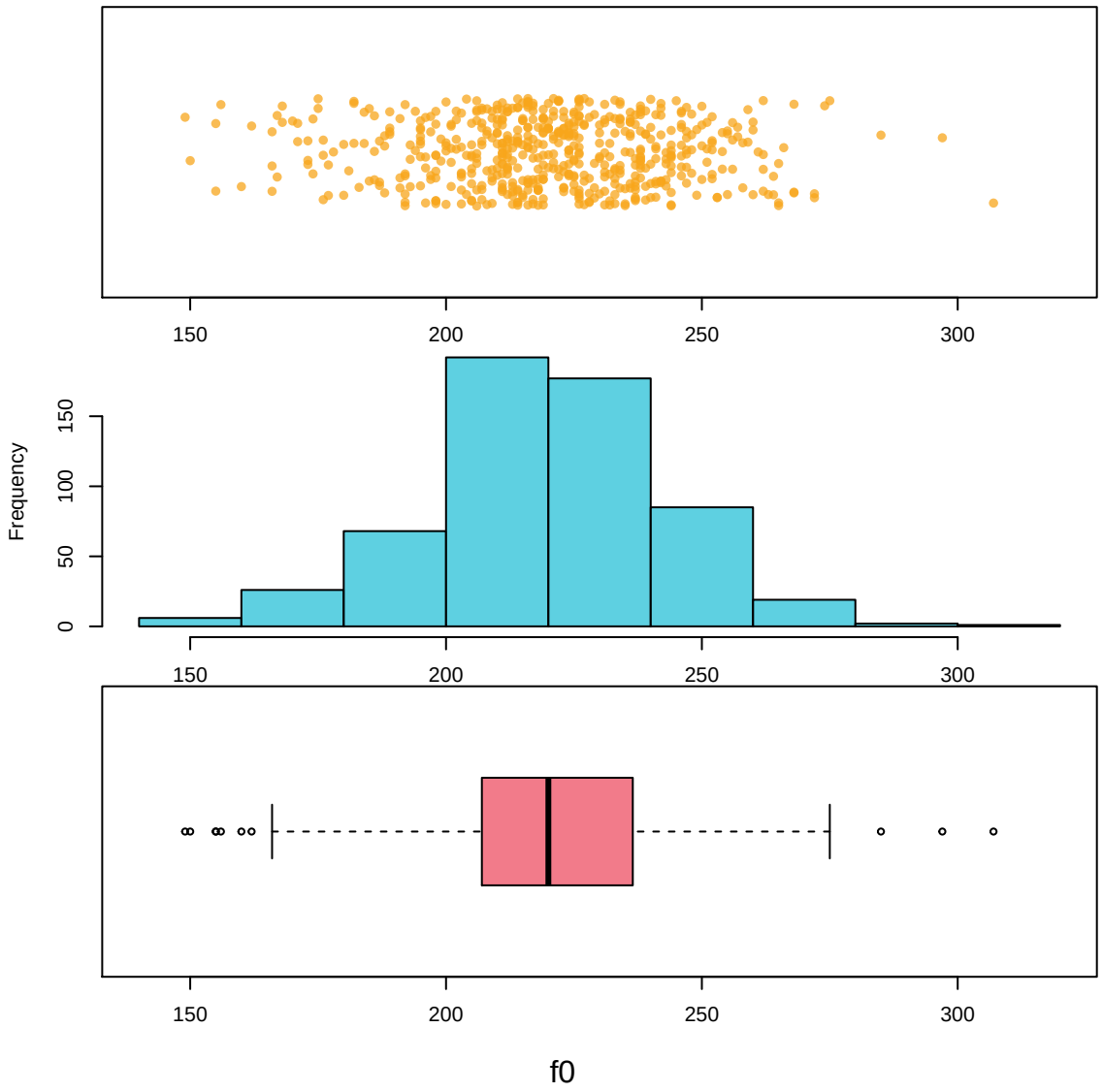


Figure 2: Different ways to consider a distribution

Probability Distributions

Histograms are particularly useful to understand because of how they relate to probability distributions. For our purposes, the probability is the number of times an event is expected to occur, out of all the other observed events and outcomes. This can also be thought of as the *percent* of times an event is expected to occur.

The total probability of all events is always equal to 1. This is like using 100 to communicate percent, its just easier that way. As a result of this convention, you know that a probability of 0.5 means something is expected to

occur half the time (i.e., on 50% of trials). For example, suppose we want to know the probability of being an adult female in our sample who produces an f0 under 175 Hz. Finding the probability of observing this event is easy:

```
# the evaluation in the parenthesis will return 1 if true, 0 if false
sum (f0s < 175)  ## number of observations the fall below threshold
```

```
## [1] 22
```

```
sum (f0s < 175) / length (f0s)  ## divided by total number of events
```

```
## [1] 0.03819444
```

```
mean (f0s < 175) ## a shortcut to calculate probability, mean = total/length
```

```
## [1] 0.03819444
```

The top value is the frequency of the occurrence. This is not so useful because this number can mean different things given different sample sizes (e.g., 22/23, 22/10000). The middle and bottom values have been divided by the total number of observations. As a result, these now represent a proportion, or probability.

Histograms can also show this difference between total counts and probabilities. Below, the histogram on the left shows the number of observations in each bin. The histogram on the right shows *density* on the y axis. When you see *density* on the y axis, that means that y axis values have been scaled to make the area under the curve equal to 1. This has two benefits:

1) It lets you compare the distribution of values across different sample sizes.
2) It makes the histogram more comparable to a probability distribution.

```
par (mfrow = c(1,2), mar = c(4,4,1,1))

hist (f0s, main="", col = lavender)
hist (f0s, freq = FALSE, main = "", col = deepgreen)
```

The density is just the thickness of the distribution at a certain location. In probability theory, the sum of the probabilities of all possible outcomes is 1, by definition. So, the fact that the area under the curve of a density is equal to 1 means that the density contains all your *stuff*, all the possible outcomes of the variable we are discussing.

Imagine a circle like in a Venn diagram that contains all possible productions of female f0. This circle has an area of 1 since it contains all possible instances of the variable. Imagine we spread out this circle along the x axis so that its shape reflected the relative frequencies of different values of the variable. For example, if some outcomes were 5 times more probable than others, the shape should be 5 times taller there, and so on. If we managed to do this, the height (or 'density') of this shape would exactly correspond to a probability distribution like that seen in the right plot above.

Below I've repeated the data, doubling the counts. Notice that the y axis in the right panel does not change. This is because increasing the number of observations changes your counts but not the relative frequencies of observations. For instance, increasing the number of coin flips will not change the fact that 50% will be heads, but it will change the number of heads observed.

```
par (mfrow = c(1,2), mar = c(4,4,1,1))
hist (c(f0s,f0s), breaks = 10, main = "", col = lavender)
hist (c(f0s,f0s), freq = FALSE, breaks = 10, main = "", col = deepgreen)
```

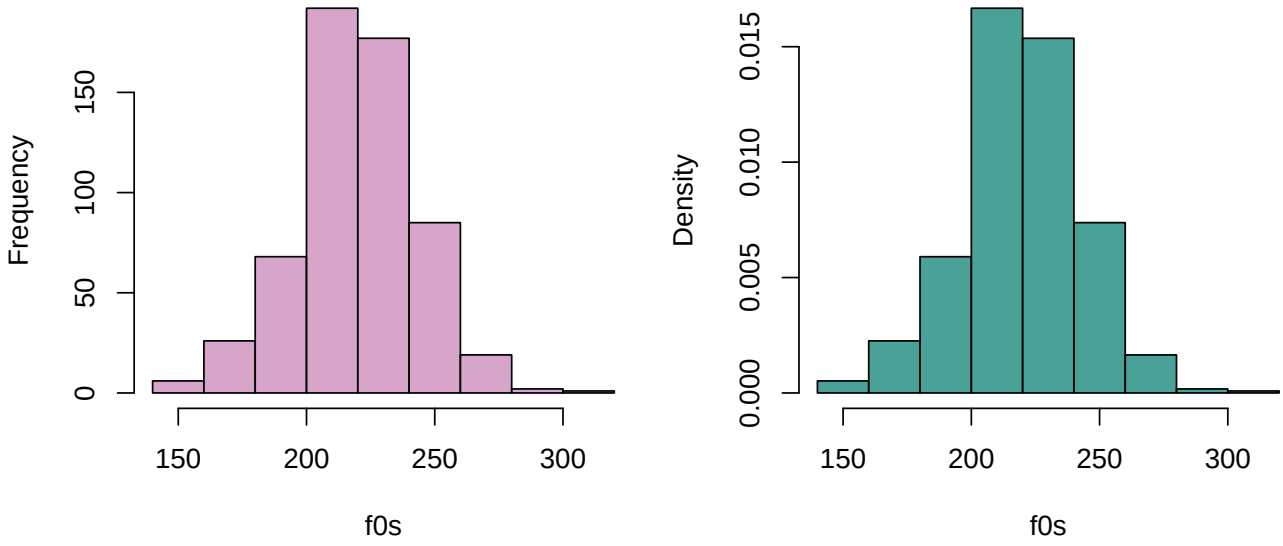


Figure 3: A comarison of two histograms

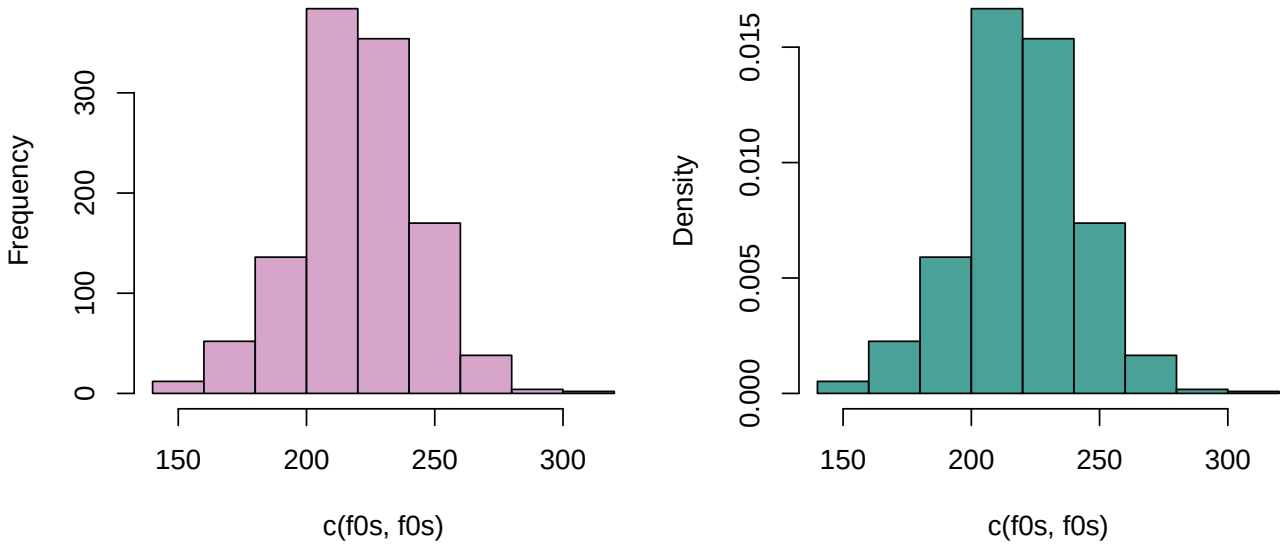


Figure 4: The counts have been doubled relative to above.

The normal distribution

The distribution of many variables (including f0) follows what is called a normal distribution. This means if you take a random sample of a variable and arrange observations into bins, they will tend to resemble the shape of a normal distribution. This distribution is also called a Gaussian distribution and has a familiar, bell-shaped curve.

The normal distribution has the following important characteristics.

1. The distribution is approximately symmetrical - i.e., producing a higher or lower than average f0 is about equally likely.

2. The probability of observing a given value decreases as you get further from the mean.

3. It is easy to work with, very well understood, and naturally arises in basically all domains.

Normal distributions have two parameters. This means they vary from each other in only two ways. These parameters are:

1. A mean, μ, which determines where the distribution is located along the x axis. The mean is the 50% halfway point of the 'mass' of the distribution. If the distribution were an physical object, its mean would be its center of gravity.

2. A standard deviation, σ, that determines its *spread* along the x axis. Since every distribution has an area under the curve equal to one (they all have the same 'volume'), the smaller the variance the higher the peak of the density along the y axis must be.

Below, I compare the histogram of f0 values to the density of a normal distribution with a mean equal to our sample mean ($\mu = \bar{x}$) and a standard deviation equal to our sample standard deviation ($\sigma = s_x$). The density was drawn using the `dnorm` function. This function will help draw a curve representing the shape of a theoretical normal distribution with a given mean and standard deviation.

```
par (mfrow = c(1,1), mar = c(4,4,1,1))
hist (f0s, freq = FALSE, main = "", breaks = 20, col = deepgreen)
abline (v = 63.8, lwd = 2, col = 2, lty=3)
## plots the normal density (red line) using stats calculated form our sample.
curve (dnorm (x, mean(f0s), sd(f0s)), from = 100, to = 300,
       lwd=3, col = coral, add = TRUE)
```

When you are dealing with normally-distributed data, summary statistics can tell you a lot about the shape of your distribution, and about where you can expect the bulk of the density/distribution to lie. The left panel shows the locations of quantiles (0%, 25%, 50%, 75%, 100%), the right panel shows you the mean and standard deviations from the mean (-3, -2, 0, +1, +2, +3). Notice that ±2 standard deviations enclose most of the distribution (around 95%), and ±3 standard deviations enclose almost all of it (99%).

```
par (mfrow = c(1,2), mar = c(4,4,1,1))

hist (f0s, main = "", col = skyblue)
abline (v = quantile (f0s), lwd = 2, col = deepgreen)
hist (c(f0s,f0s), freq = FALSE, breaks = 10, main = "", col = yellow)
abline (v = seq (mean(f0s)-3*sd(f0s),mean(f0s)+3*sd(f0s),sd(f0s)), lwd = 2,
        col = coral)
```

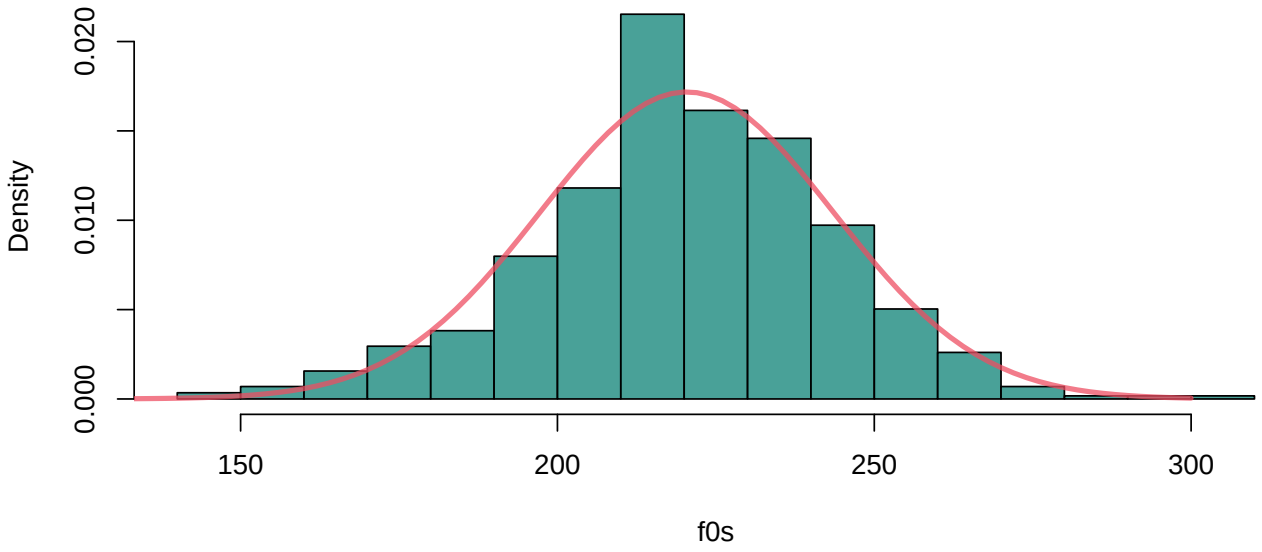


Figure 5: A comparison of the data distribution with a theoretical normal distribution.

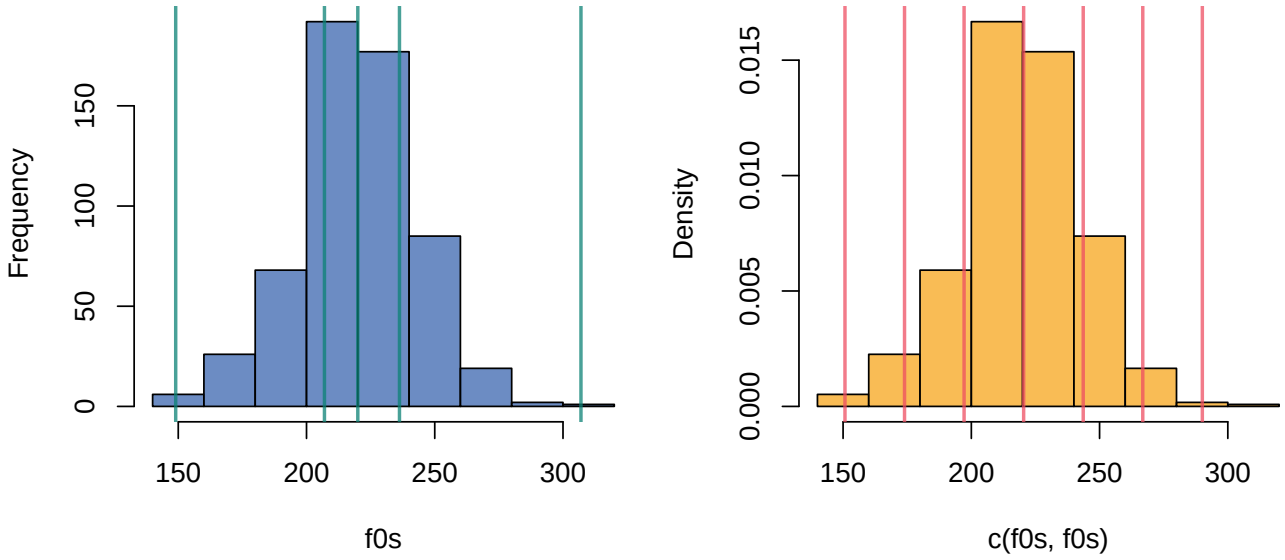


Figure 6: Quantiles and standard deviations help understand the shape of a distribution.

Referring to the normal distribution to make inferences

In general, it is impossible to know what the 'true' data distribution is, so that *perfect* inference is not possible. As a result, scientists often use theoretical probability distributions to make inferences about real-life populations and observations. Notice that our real life measurements follow the 'shape' predicted by the theoretical normal distribution. This suggests that we may be able to use the characteristics of an appropriate normal distribution to make inferences about female f0 (and other variables).

Using a normal distribution to make inferences about your data is like using a mathematical model for spheres to understand the behavior of billiard balls. In reality the balls are not perfect spheres. However, their shapes will be spherical enough to allow us to make useful predictions based on the simplified model. In general, it is useful to keep in mind that reality will never exactly conform to our model. This can result in unpredictable errors in our conclusions, which can cause errors to occur. In general, the things you don't know you don't know are the things that will cause the most problems.

So, since we expect the distribution of f0 values to have the shape of the normal distribution, we can use the shape of the normal distribution to make inferences about the distribution of f0 values, even the ones we did not observe. For example, we can use the theoretical normal density to estimate the probability of observing a female production with an f0 of under 175 Hz, from among *all* possible observable productions of f0 in this *population*.

We do this by referring to the proportion of values expected to be less 175 Hz in the normal distribution. This can be found by finding the area under the curve of the probability density to the left of that point (the red area below). Since the *total* area is always equal to 1, the area of the red portion below corresponds to a percentage/probability.

Below, I use the function `pnorm` to find the proportion of values that are expected to be greater/less than 175 Hz. I use the parameters estimated form our sample to run the `pnorm`function, as these are our best guesses of the population parameters. As we can see, this value is reasonably close to our empirical proportion, which was 0.038 (3.8%).

```
par (mfrow = c(1,1), mar = c(4,4,1,1))
hist (f0s, freq = FALSE, main = "", breaks = 20, col = deepgreen)
abline (v = 175, lwd = 2, col = 2, lty=3)
## plots the normal density (red line) using stats calculated form our sample.
curve (dnorm (x, mean(f0s), sd(f0s)),from=100, to=300, lwd=2, col = 2, add=TRUE)

x = c(140,seq(140,175,length.out = 100),175)
y = c(0,dnorm(seq(140,175,length.out = 100), mean (f0s), sd (f0s)),0)
polygon(x, y, col='2')
abline (v = 63.8, lwd = 2, col = 2, lty=3); abline (v = 70, lwd = 2,col=1,lty=3)
```



```
## probability of observing a production below 175 Hz
pnorm (175, mean (f0s), sd(f0s))
```

```
## [1] 0.02527988
```

```
## probability of observing a production greater than 175 Hz
1 - pnorm (175, mean (f0s), sd(f0s))
```

```
## [1] 0.9747201
```

Imagine you had 1 pound of clay and I asked you to make a shape **exactly** like the normal density (red curve) above with a constant depth. The 'area under the curve' would just correspond to the amount of clay in a certain area. So, if you made the density just right and I took a knife and cut the shape left of 175 Hz (the red part) and we weighed it, it should weigh 2.5% of a pound. So, the area under the curve, the probability, is just the amount of the *stuff* in the density that falls below/above a certain point, or between two points.

Since the total number of observations is always one. This helps us compare across many different actual numbers of observations. The probability above suggests the following:

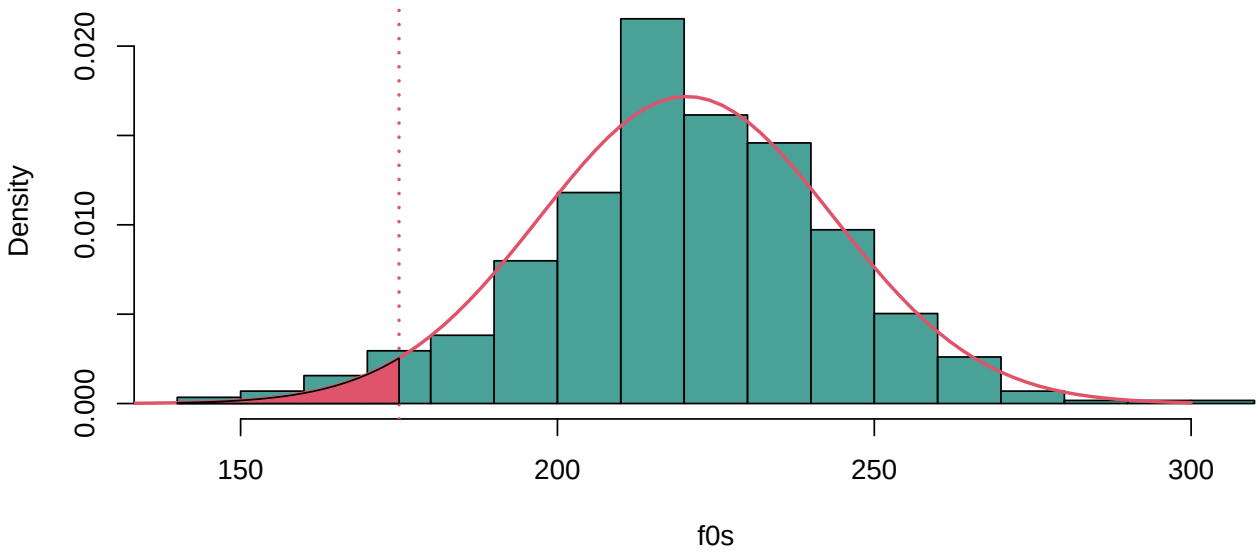


Figure 7: The read area relects the distribution of outcomes that satisfy f0 < 175 Hz.

```
## probability of observing a production with an f0 under 175 Hz
pnorm (175, mean (f0s), sd(f0s))
```

```
## [1] 0.02527988
```

```
## expected count
pnorm (175, mean (f0s), sd(f0s)) * length (f0s)
```

```
## [1] 14.56121
```

```
## actual count
sum (f0s < 175)
```

```
## [1] 22
```

We can also use this theoretical distribution to think about other possible outcomes:

```
min (f0s)
```

```
## [1] 149
```

```
pnorm (149, mean (f0s), sd(f0s)) # probability of observing our smallest value
```

```
## [1] 0.001052907
```

```
pnorm (140, mean (f0s), sd(f0s)) # probability of observing a smaller value
```

```
## [1] 0.0002676171
```

```
## predicted number of tokens below 175 Hz if we was had 5500 observations
pnorm (175, mean (f0s), sd(f0s)) * 5500
```

```
## [1] 139.0394
```

Probabilities of events and likelihoods of parameters

We are going to switch from talking about *probabilities* to talking about *likelihoods*. A probability is the odds of observing some event/outcome, given some parameter(s). A likelihood is the odds of observing a parameter given some observed data. In both cases, these terms assume that you are referring to some probability distribution.

Every parameter for every probability distribution has a likelihood function, given some data. I am only going to talk about the likelihood of the normal mean parameter, μ, in detail.

The *likelihood function* is a curve showing the relative likelihoods of different parameter values, given a fixed set of data. The likelihood function tells you what values are *believable* given your data. If a value is very unlikely, that means that it is not supported by your data. In other words, unlikely parameter estimates represent conclusions that your data is rejecting as not viable.

Here are three useful properties of the likelihood functions of μ, the mean parameter of the normal distribution:

1. The likelihood function of μ will tend to be a normal distribution.
2. The mean (and peak) of the likelihood function of μ given some sample x is equal to the arithmetic mean of the sample ($\bar{x}$).
3. The standard deviation of the likelihood of μ is equal to the standard deviation of the data (s_x), divided by the square root of N (the sample size).

The first point tells us that we can use the normal distribution to make inferences about likely, and unlikely values for means, given some data.

The second point says that if you are wondering what the best (most likely) estimate of μ is given your sample, the answer is the arithmetic mean of your sample ($\bar{x}$).

The third point means that the likelihood function for μ will tend to be *much* narrower than the distribution of our original data. This is because a mean based on, for example, 50 samples will contains many positive and negative deviations from the average that will tend to cancel out. As a result, the more data you have the more *precise* your estimates are, and the less *uncertainty* is associated with any estimate.

The left panel below shows the likelihood function for μ based on the first 10 observations of our f0 vector, shown by the blue points at the bottom of the plot. I chose this small sample just to make this example clearer. Notice that the most likely mean values for these points like over the bulk of the sampled values. The vertical dotted lines show three possible mean values that will be highlighted.

The likelihood of any parameter estimate (e.g., $\mu = 175$ Hz in the right panel below) is equal to the product of the density of each observation in the sample, if we assume that the estimate were true. For example, to calculate the likelihood that μ, we:

1) assume that the data is generated by a normal distribution with a μ equal to x, and σ equal to the sample standard deviation (s_x).
2) Find the the height of the curve of the probability distribution (the density) over each point (indicated by lines in the right panel below).
3) The likelihood is the product of all of these densities. In practice, the logarithms of the individual probabilities are added together, yielding the *log-likelihood*. This is because multiplying together too many fractions can lead to numbers so small computers have a hard time representing them.

Imagine I follow the steps above for each position along the x axis, recording the likelihood values I calculate. I then plot the product of the densities for each x value: I have just plotted a likelihood function for μ given our data.

```
x = f0s[1:10]   ## tiny sub sample for example
mean(x)  ## sample mean
```

```
## [1] 220.2
```

```
sd (x)   ## sample standard deviation
```

```
## [1] 21.86219
```

```
par (mfrow = c(1,2), mar = c(4,4,1,1))
plot (x,rep(0,10), ylim = c(0,.08), pch=16,col=4, xlim = c(150,300),
      ylab='Density', main = 'Likelihood of mean',xlab='f0',cex.main=.8)
## here the likelihood sd is divided by the sample size
curve (dnorm (x, mean(x), 21.9 / sqrt (10)), from = c(150,300),
       add=TRUE, col = 2, lwd = 2)
abline (v = c(175, 200, 225), lwd=2,lty=3)

plot (x,rep(0,10), ylim = c(0,.022), pch=16,col=4,xlim = c(150,300),cex.main=.8,
      ylab='Density', main = "mean = 175",xlab='f0')
## now it is centered at mean = 175 Hz
curve (dnorm (x, 175, 21.9), from = c(150,300),
       add=TRUE, col = 2, lwd = 2)
segments (x,rep(0,10),x,dnorm (x, 175, sd (x)))
```

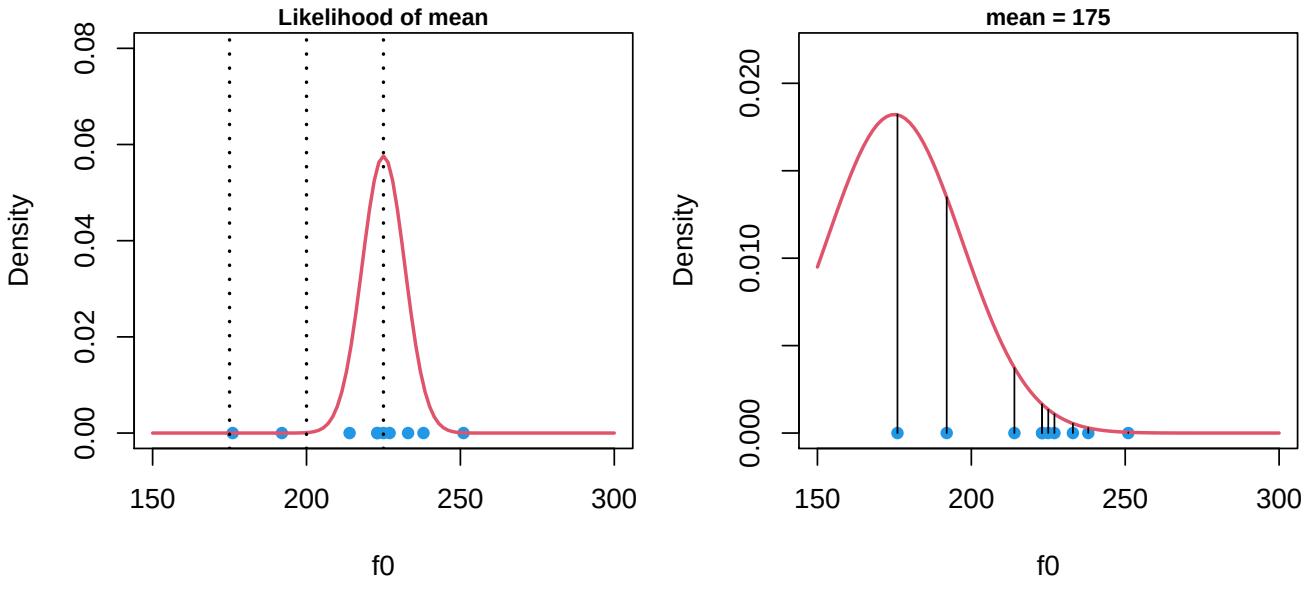


Figure 8: (Left) The likelihood of the population mean given the blue points in the figure. (right) The probability of the points given an assumed mean of 175 Hz.

In the right panel above we see that a normal distribution with a μ of 175 Hz is very unlikely to generate this data. Many points are extremely improbable and have densities close to zero. As a result, the product of these values (the heights of the lines) will be a very small number. This is reflected in the extremely small values in the likelihood function at 175 Hz in the left panel above.

In the left panel below, we see that a normal distribution with a μ of 200 Hz is more likely to generate this data, and the probability distribution is clearly a much better fit. However a distribution with a mean of 200 Hz is still not very likely to have generated this data.

Finally, in the right panel below we see the the maximum likelihood estimate of 225 Hz, the value representing the peak of the likelihood function (in the left panel above). When we say that 225 Hz is the most likely mean for this data, we are saying that this data is most probably the outcome of a normal distribution centered at 225 Hz, relative to the alternatives.

```
x = f0s[1:10]   ## tiny sub sample for exampls
par (mfrow = c(1,2), mar = c(4,4,1,1))
plot (x,rep(0,10), ylim = c(0,.022), pch=16,col=4,xlim= c(150,300),cex.main=.8,
      ylab='Density', main = "mean = 200",xlab='f0')
## distribution centered ar 200
curve (dnorm (x, 200, 21.9), from = c(150,300),
       add=TRUE, col = 2, lwd = 2)
segments (x,rep(0,10),x,dnorm (x, 200, sd (x)))

plot (x,rep(0,10), ylim = c(0,.022), pch=16,col=4,xlim =c(150,300),cex.main=.8,
      ylab='Density', main = "mean = 225",xlab='f0')
## and now at 220
curve (dnorm (x, 225, 21.9), from = c(150,300),
       add=TRUE, col = 2, lwd = 2)
segments (x,rep(0,10),x,dnorm (x, 225, sd (x)))
```

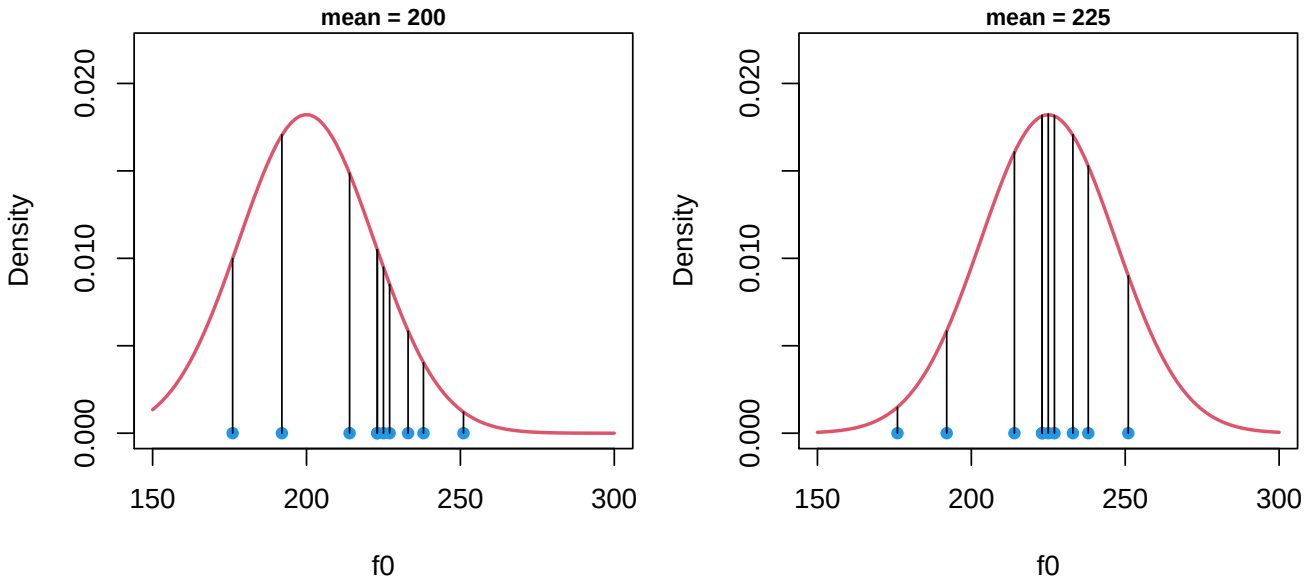


Figure 9: (Left) The probability of the points given an assumed mean of 200 Hz. (right) The probability of the points given an assumed mean of 225 Hz.

Making inferences using likelihoods

Previously, I mentioned using the normal distribution to make inferences. When variables are normally distributed we can use the theoretical normal distribution and functions such as `pnorm` to answer questions about values we expect, and don't expect, to see.

We can take this same approach to make inferences about *parameters* when their likelihood functions follow a normal distribution. For example, we can use the results of the calculations below:

```
mean (f0s)   ## sample mean
```

```
## [1] 220.401
```

```
sd (f0s)     ## sample standard deviation
```

```
## [1] 23.22069
```

```
length (f0s)  ## sample size
```

```
## [1] 576
```

```
sd (f0s) / sqrt ( length (f0s) ) ## the standard deviation of the likelihood function
```

```
## [1] 0.9675289
```

To draw the expected likelihood function for μ given our data and our model. You may be thinking, what model? It may seem too simple to be a model, but by assuming that our data can be understood as coming from a normal distribution with some given μ and σ, we have already created a simple model for our data. I'll return to this below.

We can take our model and our parameter estimates and draw the likelihood function for μ. We can then use the `qnorm` function to calculate quantiles for our likelihood, presented below. I added vertical lines at the 2.5% and 97.5% quantiles of our distribution. These vertical lines enclose 95% of the likelihood density, and so represent the range of values representing the 95% most likely values of μ. I chose an interval enclosing 95% of the likelihood because this is used by convention. This is a commonly-used interval but otherwise has no special significance.

```
par (mfrow =c(1,1), mar = c(4,4,1,1))
curve (dnorm (x, mean(f0s), sd(f0s)/sqrt(length(f0s))), xlim = c(216,225),
       ylab = 'Density', xlab = 'f0', col = lavender, lwd = 4)
quantiles = qnorm (c(0.025, 0.975), mean (f0s), sd (f0s) / sqrt (length (f0s) ) )
quantiles
```

```
## [1] 218.5047 222.2974
```

```
abline (v = quantiles, lwd=2,col=deepgreen)
```

The likelihood tells you about the most believable/credible parameter values, given your model and data. Given the information presented in the figure above, we may conclude that the most likely parameter values fall between 218 and 222 Hz. This means that it is reasonable that the true value might be 221 Hz, as this value is very likely given our sample. Basically, maybe our sample mean is wrong and arose by accident, and 221 Hz is the true μ. This outcome is compatible with our data.

However, a value of 216 Hz is extremely *unlikely* to fit our data. It is just too far from our sample mean relative to the amount of variation in our sample. This is like if you measured the heights of 100 women in a small town (pop. 1500) and found the average was 5'4". You might accept that the actual population average is 5'5", but may find it difficult to accept that it was actually 6'0". It would mean you happened to measure all of the shortest women in the town, an extremely unlikely event.

So, since we think that 216 Hz is not a plausible mean f0 given our sample, this also means that it is very unlikely that the real μ is 216 Hz. This is because a distribution centered at 216 would be extremely unlikely to generate a sample mean of 220 Hz. Using this aproach, we can rule out implausible values of μ based on the characteristics of our data.

At this point we can offer conventional responses to the research questions posed at the start of the Chapter:

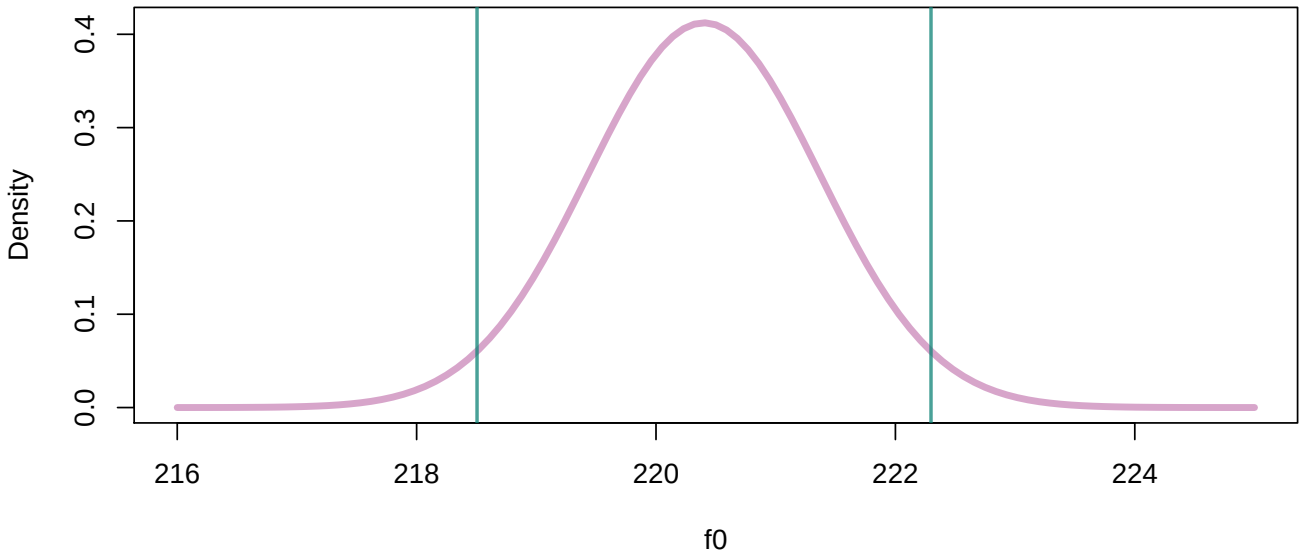


Figure 10: Likelihood of population mean given our data. Horizontal lines indicate intervals enclosing 95% of the distribution.

Q1) What is the average f0 of the whole *population* likely to be?

A1) The most likely value for the population mean is our sample mean, 220.4 Hz.

Q2) Can we set bounds on likely mean f0 values based on the data we collected?

A2) Yes, there is a 95% probability that the population mean is between 218.5 222.3 Hz, given our data and model structure.

Traditional approaches to statistics (sometimes generally referred to as 'frequentist') estimate parameters by trying to find the most likely values for parameters (i.e., 'maximum likelihood estimation'). They do this by referring to the theoretical likelihood functions such as what we plotted above. Although this works very well for simple data, it is difficult if not impossible for some of the more complicated datasets that often arise for even the simplest research questions in linguistics.

Bayesian models

In this class we are going to learn about *multilevel Bayesian models*. These models have many advantages over 'traditional' approaches. They provide researchers with more information, are more robust, and at **worst**, they are as good as traditional models. I may sound biased, but the main reason for all of these advantages is that traditional models were developed over 100 years ago. On the other hand, mathematical and technological advances have only made Bayesian multilevel models possible in the last 10+ years. It is only reasonable that the newer approaches should offer some advantages over methods developed before calculators existed.

Here, I am going to address what is meant by two aspects of the term 'Bayesian multilevel models': 'Bayesian' and 'models'.

What are regression models?

I have been referring somewhat obliquely to 'models' without really explaining what I mean by this. It's difficult to offer a precise definition because the term is so broad, but 'regression' modeling can be thought of as trying to understand variation the mean parameter (μ) of a normal distributions. Actually, you can use many other probability distributions, but for now we will focus on models based on the normal distribution.

Basically it goes like this:

- you assume that your data is well described by a normal probability distribution. This is a mathematical function ($\mathcal{N}(\mu, \sigma)$) that described what is and is not probable based on two parameters.
- the mean of this distribution is either fixed, or varies in a logical manner.
- the variation in the mean of this distribution can be understood using some other variables.

We can write this model more formally like:

$$y \sim \mathcal{N}(\mu, \sigma) \tag{1}$$

This says that we expect that the variable we are interested in is distributed according to ($\sim$) a normal distribution with those parameters. Basically, this just formalizes the fact that we think the *shape* of our data will be like that of a normal distribution with a mean equal to μ and a standard deviation equal to σ.

When you see this, $\mathcal{N}(\mu, \sigma)$, just picture in your mind the shape of a normal distribution, like if you see this $y = x^2$ you may imagine a parabola. $\mathcal{N}(\mu, \sigma)$ Really just represents that shape of the normal distribution, and the associated expectation about more and less probable outcomes.

The above relationship can also be presented like this:

$$y = \mu + \mathcal{N}(0, \sigma)(\#eq : 2)$$

Notice that we got rid of the $\sim$ symbol, moved μ out of the distribution function ($\mathcal{N}()$), and that the mean of the distribution function is now 0. This breaks up our variable into two components:

1) A systematic component, μ, that contributes the same value to all instances of a variable.
2) A random component, $\mathcal{N}(0, \sigma)$, that causes unpredictable variation around μ.

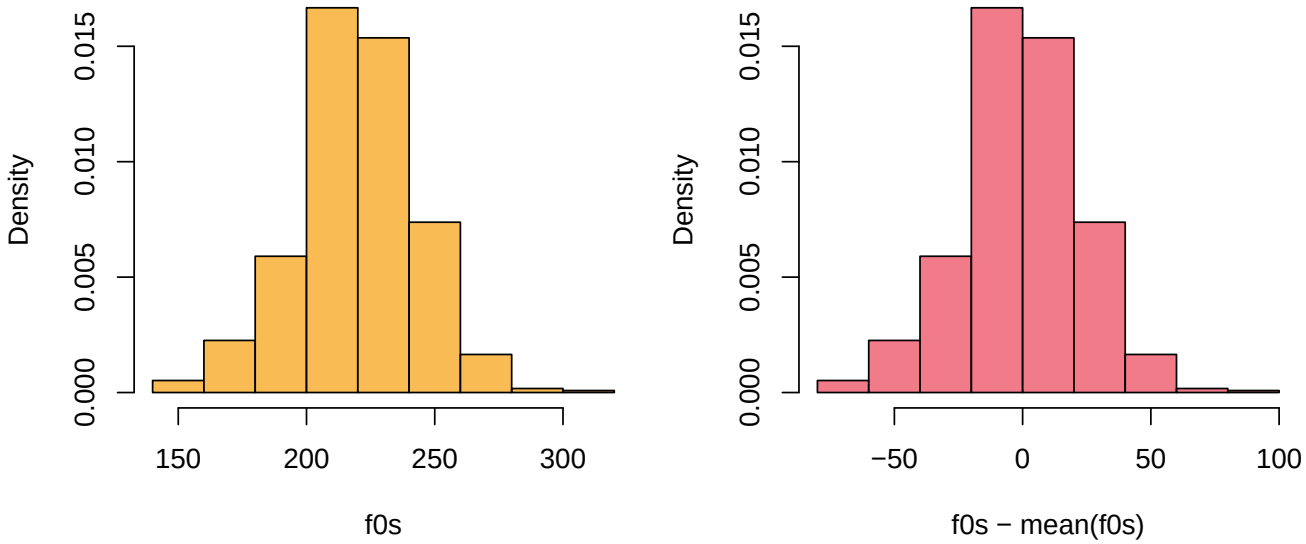
In terms of our data, I might express the distribution in either of the following ways:

$$f0 = \mathcal{N}(220.4, 23.2)(\#eq : 3)$$

$$f0 = 220.4 + \mathcal{N}(0, 23.2)(\#eq : 4)$$

The distribution on the left below is the original data, centered at 220.4 Hz and with a standard deviation of 23.2 Hz. On the right, the mean has been subtracted from each value. The sample now represents random variation around the sample mean, variation that our model can't explain. From the perspective of our model, this is *noise*, or *error*. This doesn't mean that it's unexplainable, it only means that we've structured our model in a way that doesn't let us explain it.

```
par (mfrow =c(1,2), mar = c(4,4,1,1))
hist (f0s, main="", freq=FALSE, col = yellow)
hist (f0s - mean (f0s), main="", freq=FALSE, col = coral)
```



In regression models, we can decompose systematic variation in μ into component parts, based on i predictor variables. The x_i. These x variables the co-vary (vary with) our y variable, and that we think help explain the variation in y. Below, I am saying that I think μ is actually equal to some combination sum of x_1 x_2 and x_3. For example, I could think that f0 could be affected by vowel category (x_1), the height of the speaker (x_2), and whether the utterance is a sentence or a question (x_3).

$$\mu = x_1 + x_2 + x_3 (\#eq : 5)$$

Actually, the mean is very unlikely to just be an equal combination of the predictors, so that a *weighting* of the predictors will be necessary. For example, maybe x_1 is twice as important as the other two predictors and so a is 2, while b and c are 1.

$$\mu = a * x_1 + b * x_2 + c * x_3 (\#eq : 6)$$

Decomposition of μ into sub-components makes our model something more like:

$$y = \mu + \mathcal{N}(0, \sigma) (\#eq : 7)$$

$$y = (a * x_1 + b * x_2 + c * x_3) + \mathcal{N}(0, \sigma) (\#eq : 8)$$

Often, ε is used to represent the random component, as in:

$$y = a * x_1 + b * x_2 + c * x_3 + \varepsilon (\#eq : 9)$$

When expressed in this manner, this is now a 'regression equation' or a 'regression model'. 'Fitting' a regression model basically consists of trying to guess the most likely values of a, b, and c given our data.

Notice that the above formulation means that regression models do not require that our *data* be normally distributed, but only that the *random variation* in our data (ε) be normally distributed. For example, in the left panel below I plot the distribution of f0 from among the entire Hillenbrand et al. data, including boys, girls, men and women. The data is not normally distributed, however, we can still use a regression based on normally-distributed data to model this as long as we expect that:

1) There is systematic variation in the μ of f0 across different groups, speakers, conditions, etc.
2) The *random variation* around these predicted values of μ more or less follows a normal distribution.

In the right panel I plot the individual densities for different speaker classes. We see that although the data is not normally distributed, the within-group variation is. This suggests a regression model is appropriate for this data.

```
par (mfrow =c(1,2), mar = c(4,4,1,1))
hist (h95$f0, main="", freq=FALSE, xlim = c(80,320), col = yellow)
plot (density (h95$f0[h95$type=='b']),col=2,lwd=4, main='',
      xlim = c(80,320),ylim=c(0,0.025))
lines (density (h95$f0[h95$type=='g']),col=3,lwd=3)
lines (density (h95$f0[h95$type=='m']),col=4,lwd=3)
lines (density (h95$f0[h95$type=='w']),col=5,lwd=3)
```

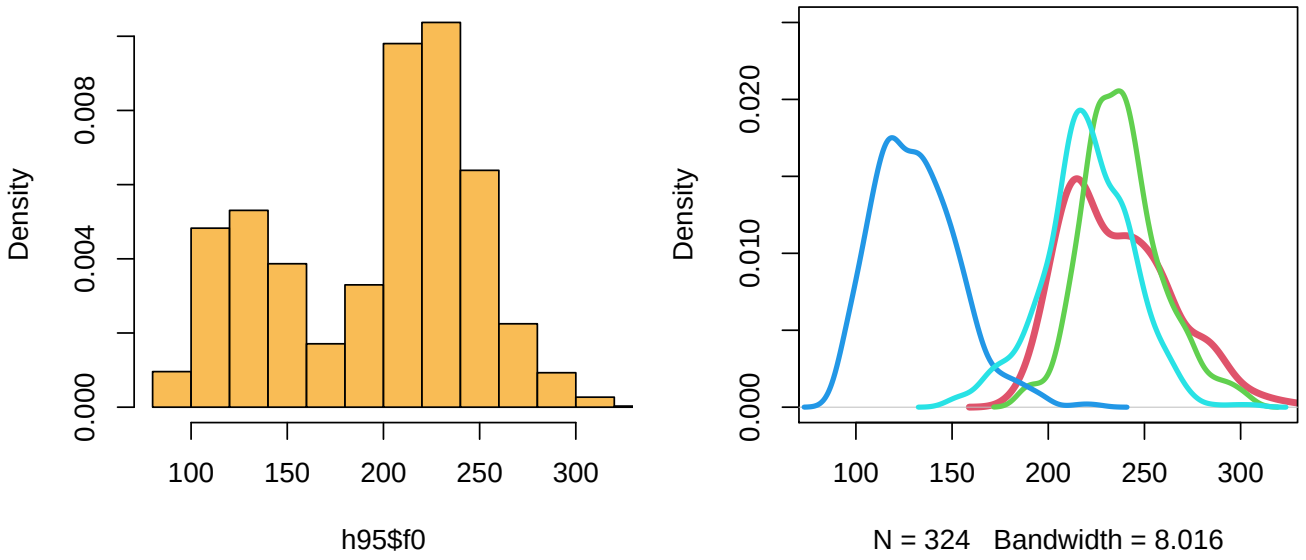


Figure 11: (left) Distribution of f0 across all speakers. (right) Densities of distributions of f0 for different speaker classes: boys (red), girls (green), men (blue) and women (cyan).

What's 'Bayesian' about these models?

The major difference between Bayesian and traditional models is that Bayesian models rely on *posterior distributions* rather than likelihood functions. I am going to define some terms:

- prior probability distribution: the distribution of possible/believable parameter values **prior** to the *current* experiment. This *a priori* expectation can come from world knowledge, previous experiments, or some combination of the two.
- the likelihood: this is the distribution of possible/credible parameter values given the **current** data and probability model, and nothing else.
- posterior probability distribution: the distribution of possible/believable parameter values you have **after** your current experiment. You get this by combining the prior distribution and the likelihood.

Traditional models make inferences based on the likelihood functions of parameters. Bayesian models make inferences based on the posterior distributions of parameters. To do this, they have to have to actually combine information about likelihood with information about the prior probabilities of parameters.

Posterior distributions

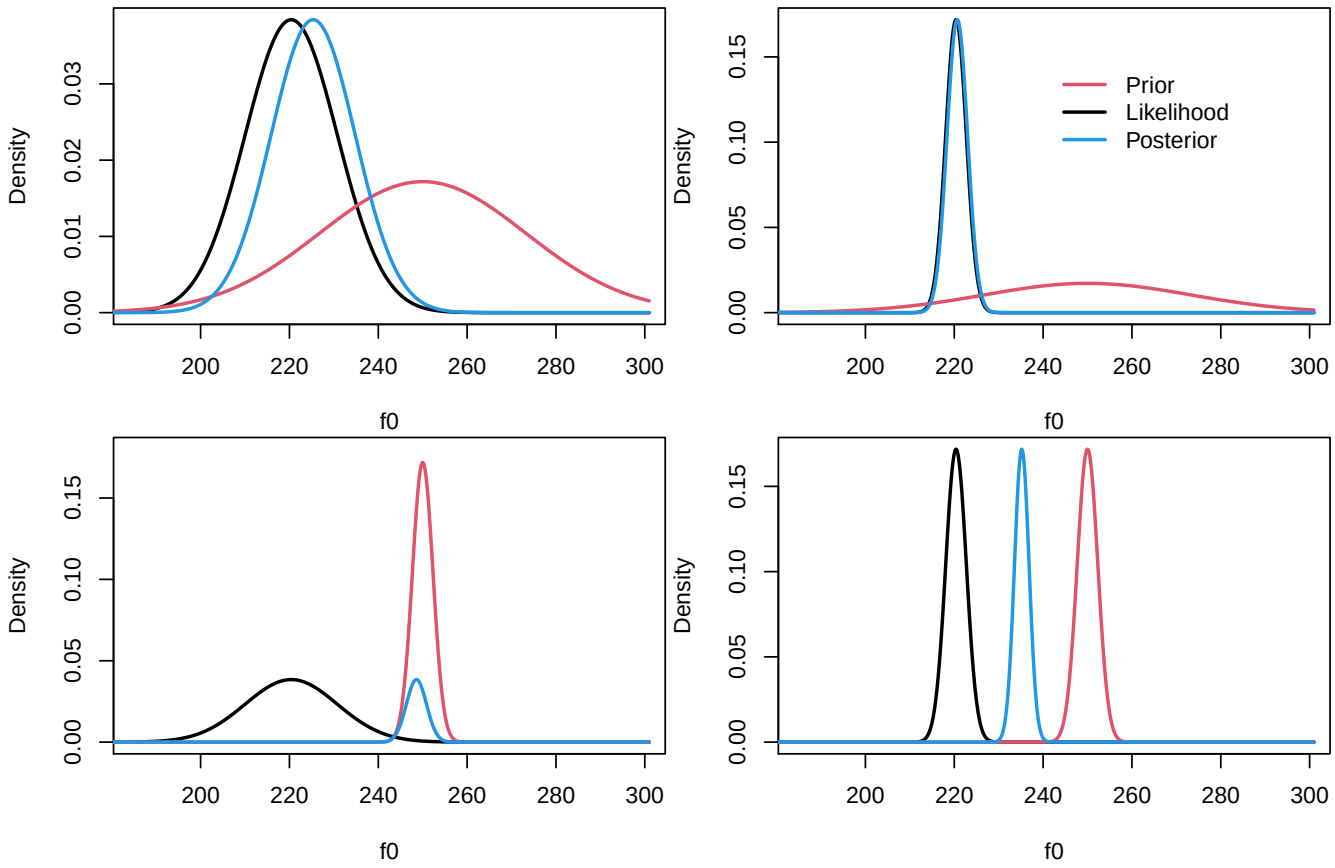
The combination of probability distributions is straightforward conceptually: you just multiply the values of the distributions at each x-axis location, and the result is the new curve. In the figure below (code at end of chapter), I combine several sets of probability distributions, showing the effects of variations in priors and likelihoods. In each plot, I scale the posterior density so that it is the same height as the likelihood. This is only to make the figures interpretable but does not affect any of the points I make below.

In the top-left panel, I plot the likelihood function for μ given a sample of size 5 with a mean of 220 Hz. I show what happens when I combine this with a relatively weak but very different prior: the standard deviation is the same as our f0 data, however the mean is much higher (250 Hz). With only 5 data points the likelihood already dominates the posterior, though the prior distribution is exerting a pull.

In the top-right panel, the posterior is almost identical to the likelihood. The likelihood represents a sample of size 100, which is actually a tiny sample in experimental linguistics work where you may have 200+ samples from each of 50+ subjects. As you might imagine, when the sample size is that large the prior exerts almost no influence on results.

In the bottom-left panel we see a situation where the prior dominates the estimate. Consider a situation where we actually have really good reasons to think that the mean is 250 Hz. If we really *know* this, why would we accept and estimate of 220 Hz based on only 5 samples? In this case, the posterior distribution is basically saying: your estimate is great, but come back when you have more evidence and I might believe you.

In the bottom-right panel we see a situation where the likelihood and the prior are equal. In this case the posterior represents compromise between new and prior knowledge.



The use of prior probabilities is often said to make Bayesian models 'subjective' but its not really a big deal. First, every model involves arbitrary decisions which can substantially affect our results. Second, a researcher will always use common sense to interpret a model. For example, before collecting my sample I can say that I expect my female average f0 to be 200 Hz or so, but think its reasonable to expect anything from 100 to 300 Hz. Based on everything

we know about human speech, even these bounds are too wide, and anything outside would suggest something is very wrong. So, even if I did not use a prior, I would use my expectations to 'screen' my results, and be very wary of anything that did not meet my expectations.

A Bayesian model simply requires that you build your expectation into your model. It formalizes it, makes it definable and replicable. Also, being 'objective' does not quite make sense in many cases. Is it really being objective to ignore common sense and act as if a mean f0 of 250 is exactly as likely a priori as one of 20,000 Hz? Because not using a prior is equivalent to using a 'flat' prior and acting like almost any value is equally likely a priori, when this is hardly the case.

Sampling from the posterior

We want to understand the posterior distribution of parameters. How do we get this information? It is difficult to get this 'analytically', that is, using exact methods and solving a bunch of equations. Many traditional methods can actually be solved in this way, and that is a big part of their popularity.

Understanding the characteristics of posterior probabilities is not possible analytically for many Bayesian models. As a result, these questions are answered 'numerically', basically by using a bunch of 'guesses'. To understand the properties of posterior distributions, we use 'sampling' software that knows how to investigate these distributions.

The way these samplers work is you specify a set of data and some relationships you think are represented in your data (i.e., a model). The sampler then 'walks around' the parameter space, which is the range of possible values a parameter (or set of parameters) can take. For example, for a single parameter the parameter space is a line (like the x axis in the plots above) along which the parameter varies.

The sampler then does some variant of the following algorithm:

1) Pick a random value for the parameter (i.e., $\mu_{tmp} = 221$ Hz).
2) Calculate the posterior probability for the current estimate of μ_{tmp}.
3) If the posterior estimate meets some criteria (e.g., it is better than the last one, it is not too low, etc.), then the value of μ_{tmp} is recorded, and becomes $\mu_{estimate}$. If not it is just discarded.
4) Go back to step 1.

As incredible as it may seem, under a very reasonable set of conditions if you do the above enough times, the distribution of $\mu_{estimate}$ that results from the above process will converge on the posterior distribution of μ given your data and model structure (including prior probabilities).

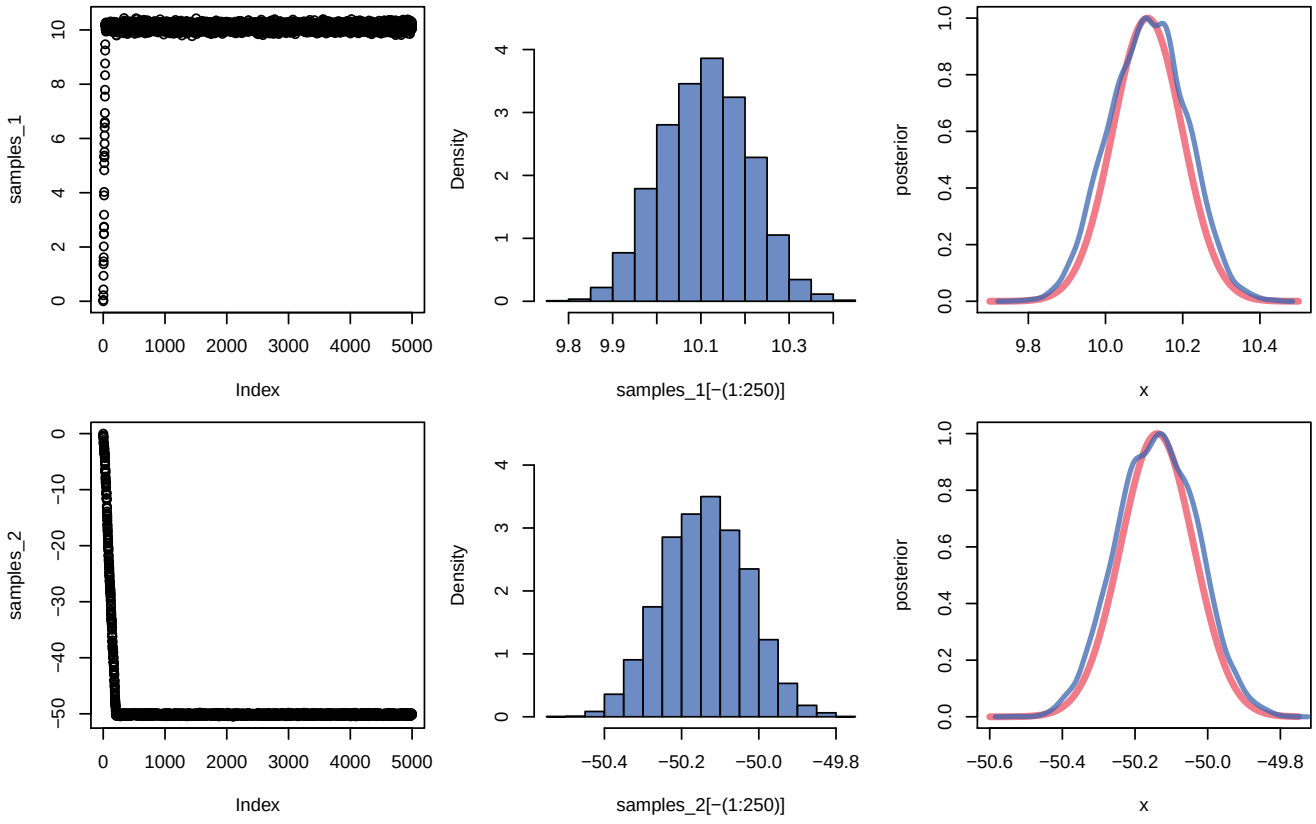
Below I have made a small example of this process. I use the Metropolis-Hastings algorithm, which is an algorithm to sample from probability distributions. The small example below assumes the standard deviation of the population is known, and just tries to investigate the posterior distribution of μ. It uses a very broad prior distribution ($\mu = 0$, $\sigma = 5000$) so that it will have a very weak effect on the outcomes.

```
# the function below takes a random sample, an initial mean estimate and a fixed
# standard deviation. It then takes a certain amount of samples from the
# posterior distribution of the parameter, assuming a broad prior centered at 0
sampler_example = function (sample, mu_estimate = 0, stdev = 1, nsamples = 1000){
  # initial posterior calculation. This is the sum of the log likelihood and
  # the logarithm of the prior probability.
  prior = log (dnorm (mu_estimate[1],0, 500))
  loglik = sum (dnorm (sample, mu_estimate[1],stdev,log=TRUE))
  old_posterior = loglik + prior

  for (i in 2:nsamples){
    accept = FALSE
    ## this loop will keep proposing new steps until one gets accepted.
    while (!accept){
```

```
      ## (step 1 above)
      ## draw new proposal by randomly changing the previous mu_estimate
      mu_tmp = mu_estimate[i-1] + rnorm (1, 0, .3)
      ## (step 2 above)
      ## find prior probability for new mu_tmp proposal
      prior = log (dnorm (mu_tmp,0, 500))
      ## find log likelihood for new mu_tmp proposal
      loglik = sum (dnorm (sample, mu_tmp,stdev,log=TRUE))
      ## calculate the new posterior probability
      new_posterior = prior + loglik
      ## (step 3 above)
      ## if better accept always. If worse, accept sometimes
      if ( ( new_posterior - old_posterior ) >= log ( runif (1,0,1) ) ){
        mu_estimate[i] = mu_tmp
        ## if you accept, the new estimate becomes the current estimate
        old_posterior = new_posterior
        accept = TRUE
      }
    }
  }
  return (mu_estimate)
}
```

In the plots below (code at end of chapter), you can see the algorithm begins at 0 (the initial guess) but is quickly able to find the most likely sample mean given the data (left column). In the middle, I show the distribution of the samples on the left, minus the burn-in phase (arbitrarily chosen by me). On the right, I compare our samples (blue) to the theoretical posterior distribution for the mean given the data and prior (red). I toss out the samples during the 'burn in' phase, as there are used up in trying to 'find' the correct location in the parameter space.



The results clearly coincide, but aren't perfect. But this sampler isn't very sophisticated! The samplers we will

be using in this class *do* provide an excellent match to the posterior distribution. As a result, we can inspect the distribution of collected $\mu_{estimate}$ to understand the posterior of our parameter. We can use these distributions in the same way that we used the theoretical likelihood functions above, by using them to make statements about likely parameter values and ranges of values.

Plot Code

Plot showing combinations of different priors and likelihoods into posteriors::

```
x = seq (150, 301, .1)
par (mfrow = c(4,1))
par(mar =c(4,4,.1,.1), oma = c(1,0,1,0))

## likelihood is a bit stronger than the prior
likelihood = dnorm (x, mean (f0s), sd (f0s) / sqrt ( 5 ) )
prior = dnorm (x, 250, sd (f0s)) ; posterior = likelihood * prior
plot (x, likelihood / max (likelihood), type = 'l', ylab='Density',lwd=2,
      xlim = c(175, 300), ylim = c(0,1.1))
lines (x, prior / max (prior),lwd=2,col=2)
lines (x, posterior / max (posterior),lwd=2,col=4)
legend (178, 1, legend = c('Prior','Likelihood','Posterior'), col = c(2,1,4),
        lwd = 2, bty = 'n')
## likelihood is much stronger than the prior
likelihood = dnorm (x, mean (f0s), sd (f0s) / sqrt ( 100 ) )
prior = dnorm (x, 250, sd (f0s)) ; posterior = likelihood * prior
plot (x, likelihood / max (likelihood), type = 'l', ylab='Density',lwd=2,
      xlim = c(175, 300), ylim = c(0,1.1))
lines (x, prior / max (prior),lwd=2,col=2)
lines (x, posterior / max (posterior),lwd=2,col=4)
## prior overwhelms the likelihood
likelihood = dnorm (x, mean (f0s), sd (f0s) / sqrt ( 5 ) )
prior = dnorm (x, 250, sd (f0s)/10) ; posterior = likelihood * prior
plot (x, likelihood / max (likelihood), type = 'l', ylab='Density',lwd=2,
      xlim = c(175, 300), ylim = c(0,1.1))
lines (x, prior / max (prior),lwd=2,col=2)
lines (x, posterior / max (posterior),lwd=2,col=4)
## prior and likelihood have about equal influence
likelihood = dnorm (x, mean (f0s), sd (f0s) / sqrt ( 100 ) )
prior = dnorm (x, 250, sd (f0s)/10) ; posterior = likelihood * prior
plot (x, likelihood / max (likelihood), type = 'l', ylab='Density',lwd=2,
      xlim = c(175, 300), ylim = c(0,1.1))
lines (x, prior / max (prior),lwd=2,col=2)
lines (x, posterior / max (posterior),lwd=2,col=4)
```

Plot showing the output of the example sampler included above:

```
set.seed(1)
## make a random sample of data
data_1 = rnorm (100,10,1)
## collect samples from the likelihood of the mean
samples_1 = sampler_example (data_1,0,sd(data_1), 5000)

## do a second one to show its not a fluke
data_2 = rnorm (100,-50,1)
```

```
## collect samples from the likelihood of the mean
samples_2 = sampler_example (data_2,0,sd(data_2), 5000)

# the left column shows the path the sampler took. the middle column shows the
# distribution of these samples, minus the burn in phase. the right column shows
# a comparison of theoretical and observed posterior distributions
par (mfrow = c(2,3), mar = c(4,4,1,1))
plot (samples_1)
hist (samples_1[-(1:250)], freq = FALSE, ylim = c(0,4.5), main="",col=4)
x = seq (9.7,10.5,.001)
likelihood = dnorm (x, mean (data_1), sd(data_1) / sqrt (100) )
prior = dnorm (x, 0, 500 )
posterior = likelihood * prior
posterior = posterior / max (posterior)
plot (x, posterior, lwd = 4, col = 2, type = 'l')
density_1 = density (samples_1[-(1:250)])
density_1$y = density_1$y / max (density_1$y)
lines (density_1, lwd = 3, col = 4)

plot (samples_2)
hist (samples_2[-(1:250)], freq = FALSE, ylim = c(0,4.5), main="",col=4)
x = seq (-50.6, -49.75,.001)
likelihood = dnorm (x, mean (data_2), sd(data_2) / sqrt (100) )
prior = dnorm (x, 0, 500 )
posterior = likelihood * prior
posterior = posterior / max (posterior)
plot (x, posterior, lwd = 4, col = 2, type = 'l')
density_2 = density (samples_2[-(1:250)])
density_2$y = density_2$y / max (density_2$y)
lines (density_2, lwd = 3, col = 4)
```

Inference for a 'single group' of observations using a Bayesian multilevel model

In this chapter I am going to discuss how to use the `brms` package to estimate a population mean given a sample of data. For these models the data:

- can come from one speaker/subject or many speakers/subjects.
- each speaker/subject can contribute multiple data points.
- does not need to be 'balanced' or 'complete' across all subjects.

The 'traditional' designs equivalent to these models are: one-sample t-test, and repeated-measures one-way ANOVA with only two groups. However, these models won't be discussed in the chapter below.

Data and research questions

We are going to keep analyzing the female f0 data from the Hillenbrand et al. (1995) dataset.

```
url1 = "https://raw.githubusercontent.com/santiagobarreda"
url2 = "/stats-class/master/data/h95_vowel_data.csv"
h95 = read.csv (url(paste0 (url1, url2)))

# select women only
w = h95[h95$type == 'w',]
# this is unique subject numbers across all groups
w$uspeaker = factor (w$uspeaker)
# select only the vector of interest
f0s = w$f0
```

We are going to try to address the same questions we talked about last week:

1) What is the average f0 of the whole *population* likely to be?

2) Can we set bounds on likely mean f0 values based on the data we collected?

However, this time we are going to do this with a Bayesian multilevel model.

Estimating a single mean with the `brms` package

The `brms` Bayesian regression models package in R lets you fit Bayesian models using the STAN probabilistic programming language using R. The package is really amazing and makes Bayesian multilevel modeling easy and accessible for anyone. It also includes a lot of helper functions that making with these models very convenient.

`brms` should be installed in R so that the models described below will work. Make sure you have the latest version of R (and Rstudio) and the latest version of the 'brms' package installed. Sometimes using older versions can cause R to crash when fitting models.

The model

Model structures are expressed in R using a very specific syntax. Think of writing a model formula as writing a language within R. The good thing about learning to write models is then you can use this knowledge to describe your models in your work, and to interpret other people's models.

The model formulas resemble regression equations to some extent, but there are some differences. Remember that regression models can be thought of like this:

$$y = \mu + \varepsilon (\#eq : 21)$$

Which means that your observed variable y is the sum of some of some average value (μ) and some random error μ. The random error is expected to be normally distributed with some unknown standard deviation ($\varepsilon \sim \mathcal{N}(0, \sigma)$).

What we would really like is to understand orderly variation in μ by breaking it up into parts $(\mathrm{x}_1, \mathrm{x}_2, ...)$ when combined using some weights $(a, b, ...)$.

$$y = a * \mathrm{x}_1 + b * \mathrm{x}_2 + ... + \varepsilon (\#eq : 22)$$

'Fitting' a regression model consists of trying to 'guess' the values of the weighing factors (a and b above), called the *model coefficients*. When we are only trying to estimate a single average, we don't have any predictors to explain variation in μ. In fact, our model structure suggests we expect no variation in μ.

Mathematically, we can't just say 'we have no predictor' since everything needs to be represented by a number. As a result, we use a 'predictor' with a value of 1 so that our regression equation is:

$$y = a * 1 + \varepsilon (\#eq : 23)$$

Now our model is trying to guess the value of a single parameter (a), and we expect this parameter to be equal to μ since it is being multiplied by a 'predictor' with a constant value of 1.

This kind of model is called an 'Intercept only' model. Regression models are really about representing *differences*, differences between groups and across conditions. When you are encoding differences, you need an overall reference point. For example, saying that something is 5 miles north is only interpretable given some reference point. The 'reference point' used by your model is called your 'Intercept'. Basically, our model consists *only* of a single reference point, the intercept. Also, when a predictor is just being multiplied by 1, we can just omit it from the regression model (but its still secretly there).

Our complete model is now:

$$\begin{aligned} f0 &= Intercept + \varepsilon \\ \varepsilon &\sim \mathcal{N}(0, \sigma) \end{aligned}$$

Put in English, each line in the model says the following:

- f0 is equal to the sum the intercept and error.

- the error is also drawn from a normal distribution with a mean of 0 and a standard deviation of σ. This distribution represents all deviation in f0 around the mean f0 for the sample.

Generally, model formulas in R have the form:

```
y ~ predictor
```

The variable we are interested in understanding (y) goes on the left hand side, and on our predictors go on the right hand side, separated by a $\sim$. Notice that the random term (ε) is not included. The formula above can be read as 'y is distributed according to some predictor', which really means "we think there is systematic variation in our y variable that can be understood by considering its joint variation with our predictor variable(s).

For intercept only models, the number `1` is included in the model formula to indicate that a single constant value is being estimated. As a result, our model formula will have the form `f0 ~ 1`. This model could be said out loud like "we are trying to estimate the mean of f0" or "we are predicting mean f0 given only an intercept".

Calling the `brm` function

Below, I load the `brms` package, which contains the `brm` function. The `brm` function takes a model specification, data and some other information, and fits a model that estimates all the model parameters. Unless otherwise specified, `brm` assumes that the error component (ε) of my model is normally distributed. The first argument in the function call is the model formula, and the second argument tells the function where to find the data. The other argument tell the function to estimates a single set of samples (chains = 1) using a single processor on your CPU (cores = 1). These arguments will be discussed more later.

```
# To ensure predictable results in examples, I will be using the same random
# seed throughout, and resetting it before running any 'random' process.
set.seed (1)
model = brms::brm (f0 ~ 1, data = w, chains = 1, cores = 1)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL '98dae0f1caaef07c210aac2156c73749' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.06 seconds (Warm-up)
```

```
## Chain 1:                0.04 seconds (Sampling)
## Chain 1:                0.1 seconds (Total)
## Chain 1:
```

By default, `brms` takes 2000 samples, throwing out the first 1000 and returning the last 1000. The output above shows you that the sampler is working, and tells you about the progress as it works.

This is the last time I will be actually fitting a model in the code chunks. I am going to be relying on pre-fit models that you can load after downloading from the book GitHub. Models can be found in the folder corresponding to each chapter.

```
## load pre fit model
model = readRDS ('2_model.RDS')
```

Interpreting the model print statement

We can evaluate the model name to show the default `brms` model print statement:

```
model
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ 1
##    Data: w (Number of observations: 576)
## Samples: 1 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 1000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.40      0.97   218.33   222.30 1.00      851      557
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    23.24      0.69    21.99    24.61 1.00      653      550
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Typing the model name into the console and hitting enter prints the information seen above. The first part just tells you technical details that we don't have to worry about for now (though some are obvious).

```
Family: gaussian
  Links: mu = identity; sigma = identity
Formula: f0 ~ 1
   Data: w (Number of observations: 576)
Samples: 1 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 1000
```

Next we see estimated effects for out predictors, in this case only an intercept. This is a 'population' level effect because is is shared by all observations in our sample, and not specific to any one observation.

```
Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   220.40      0.97   218.33   222.30 1.00      851      557
```

The information above provides the mean (Estimate) and standard deviation (Est. Error) of the posterior distribution of μ (Intercept). The values of `l-95% CI` and `u-95% CI` represent the upper and lower '95% credible intervals' for the posterior distribution of this parameter.

The *x% credible interval* for a parameter is the smallest interval that encloses x% of the distribution. This parameter has an x% chance (0.x probability) of falling inside the x% credible interval. So, this means that there is a 95% probability that μ is between 218 and 222 Hz given our data and model structure.

Notice that the parameter estimate and intervals almost exactly match the estimate and intervals we obtain by referencing the theoretical likelihood function (discussed in Chapter 1):

```
## sample mean
mean (f0s)
```

```
## [1] 220.401
```

```
## theoretical quantiles for likelihood of mean
qnorm (c(0.025, 0.975), mean (f0s), sd (f0s) / sqrt (length (f0s) ) )
```

```
## [1] 218.5047 222.2974
```

Our model also provides us an estimate of the error (ε), under 'Family Specific Parameters: sigma'. This estimate closely matches our sample standard deviation (s_x) estimate of 23.2. In addition, we also get a 95% credible interval for this parameter (2.5% = 21.99, 97.5% = 24.61).

```
Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    23.24      0.69    21.99    24.61 1.00      653      550
```

This last section is just boilerplate and contains some basic reminders. This text will look the same after all models.

```
Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

Seeing the samples

In Chapter 1 I discussed that samplers (like `brm`, or STAN) take samples of the posterior distributions of parameters given the data and model structure. It's helpful to see that this is quite literally what is happening, and that the print statement above just summarizes the information contained in the posterior samples.

Below I get the posterior samples from the model. We have 1000 samples, as indicated in the model output above. The first column represents the model intercept, the middle column is the error, and the third column is a statistic related to model fit.

```
## get posterior samples from model
samples = brms::posterior_samples (model)
str (samples)
```

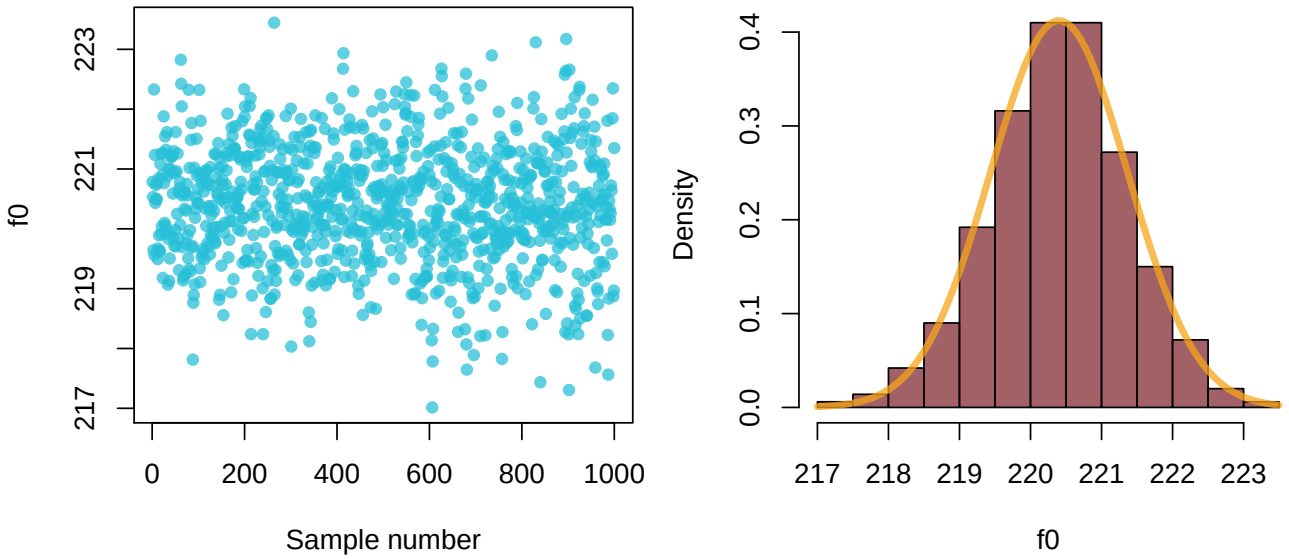
```
## 'data.frame':    1000 obs. of  3 variables:
##  $ b_Intercept: num  221 221 220 222 220 ...
##  $ sigma      : num  22.1 21.9 23.2 21.9 22.9 ...
##  $ lp__       : num  -2635 -2635 -2634 -2637 -2634 ...
```

```
## inspect values
head (samples)
```

```
##   b_Intercept    sigma      lp__
## 1    220.7860 22.06961 -2634.851
## 2    220.5312 21.93719 -2635.160
## 3    219.6445 23.22390 -2633.588
## 4    222.3286 21.93414 -2637.392
## 5    219.5626 22.87259 -2633.785
## 6    221.2339 23.08532 -2633.672
```

I can plot the individual samples for the mean parameter on the left below. On the right I plot a histogram of the same samples, superimposed with the theoretical distribution of the likelihood. Although this is not the posterior, with so many data points we expect our posterior to be dominated by the likelihood anyways.

```
par (mfrow = c(1,2), mar = c(4,4,1,1))
plot (samples[,1], xlab = 'Sample number',ylab = 'f0',col=teal,pch=16)
hist (samples[,1], freq = FALSE, breaks = 20,main='',xlab='f0',col=maroon)
curve (dnorm (x, mean (f0s), sd (f0s) / sqrt (length (f0s) )), add = TRUE,
       lwd = 4, col = yellow)
```



Recall that our model output provides information about expected values for the mean parameter:

```
Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   220.40      0.97   218.33   222.30 1.00      851      557
```

These simply correspond to the quantiles of the posterior samples!

```
quantile (samples[,1], c(.025, .5, .975))
```

```
##     2.5%      50%    97.5%
## 218.3288 220.3997 222.3001
```

There is no special status for these quantiles. We can check the values of other ones:

```
quantile (samples[,1], c(.25, .75))
```

```
##      25%      75%
## 219.8043 221.0296
```

Or even use the posterior distribution to find the probability that the mean parameter is over/under any arbitrary value:

```
mean (samples[,1] < 221)
```

```
## [1] 0.74
```

For example, given the calculation above we can say that there is a 0.74 probability (a 74% chance) that the mean f0 for female speakers in this population is under 221 Hz, given our data and model structure. We come to this conclusion by finding that 74% of the posterior samples of the parameter of interest are below 221 Hz.

Repeated measures data

The model we fit above is a reasonable starting point, but it has many weaknesses. For example, it does not consider the fact that our data was produced by a fixed number of speakers, sampled from a population. It does not consider the variation in f0 inherent between speakers, treating this as 'noise'.

Importantly, our data consists of 12 productions from each speaker in our sample, meaning we have 'repeated measures' data. Treating repeated measures data as if it were *not* repeated measures data can cause problems for our inferences. This is because it can give us a warped perspective of how much variability there really is in the sample.

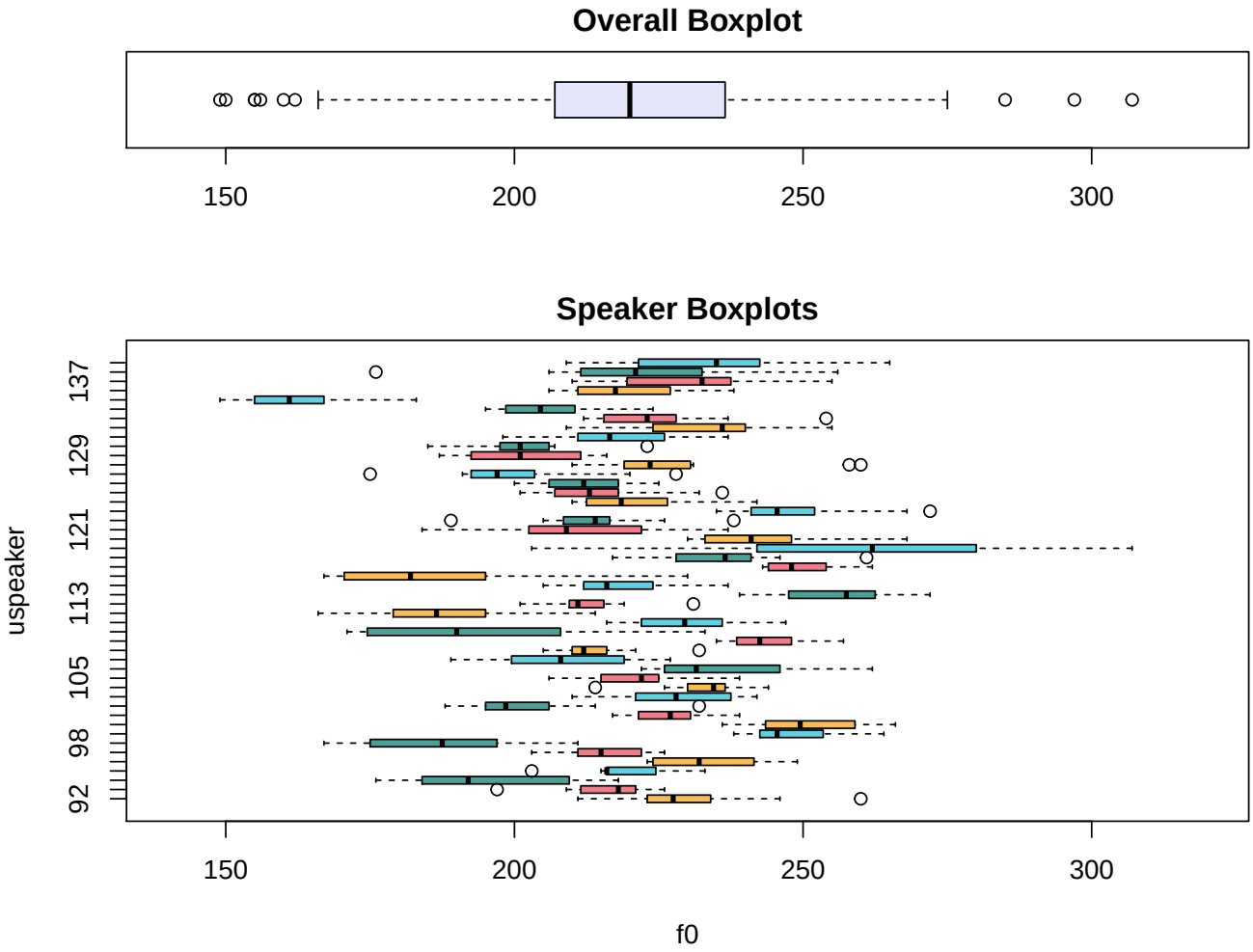
For example, if I told you I had 1,000,000 samples of speech from male speakers from Los Angeles, you may be confident that I can estimate the average f0 male speakers from Los Angeles very accurately. But what if I told you that all these samples were from only three different people? You know instinctively that this makes my data less reliable.

The reason repeated-measures data can cause problems is because the measurements are correlated: multiple measurements from the same person are obviously going to be related to each other. If you measure the height of a tall person today, they will still be tall tomorrow. Because of this general principle, although we have 12 productions from each of 48 female speakers, we do not actually have 576=48*12 totally independent observations in our data.

This can be seen quite clearly below. The top panel shows the distribution of all our f0 measurements. The bottom panel shows speaker boxplots (one for each speaker's data). If we were to 'push down' on the bottom panel and collapse all our boxplots into a single distribution, we would end up with the boxplot in the top panel.

These boxplots shows that each speaker has their own average f0, and that their productions tend to vary around their 'natural average. As a result, we might have closer to 46 observations (one average value per speaker) than 576. For example, the 'outliers' around 150 Hz may seem like huge 'errors' in the top plot. In the bottom plot we see that these productions all come from one speaker, and actually reflect her average f0. These are not errors but systematic between-speaker *variation*.

```
par (mar = c(4,4,2,1)); layout (mat = c(1,2), heights = c(.3,.7))
boxplot (f0s, main = "Overall Boxplot", col="lavender",
         horizontal = TRUE, ylim = c(140,320))
boxplot (f0 ~ uspeaker, data = w, main = "Speaker Boxplots", col=c(yellow,coral,
         deepgreen,teal), horizontal = TRUE, ylim = c(140,320))
abline (h = 220.4,lty=3,col='grey',lwd=2)
```



Multilevel models

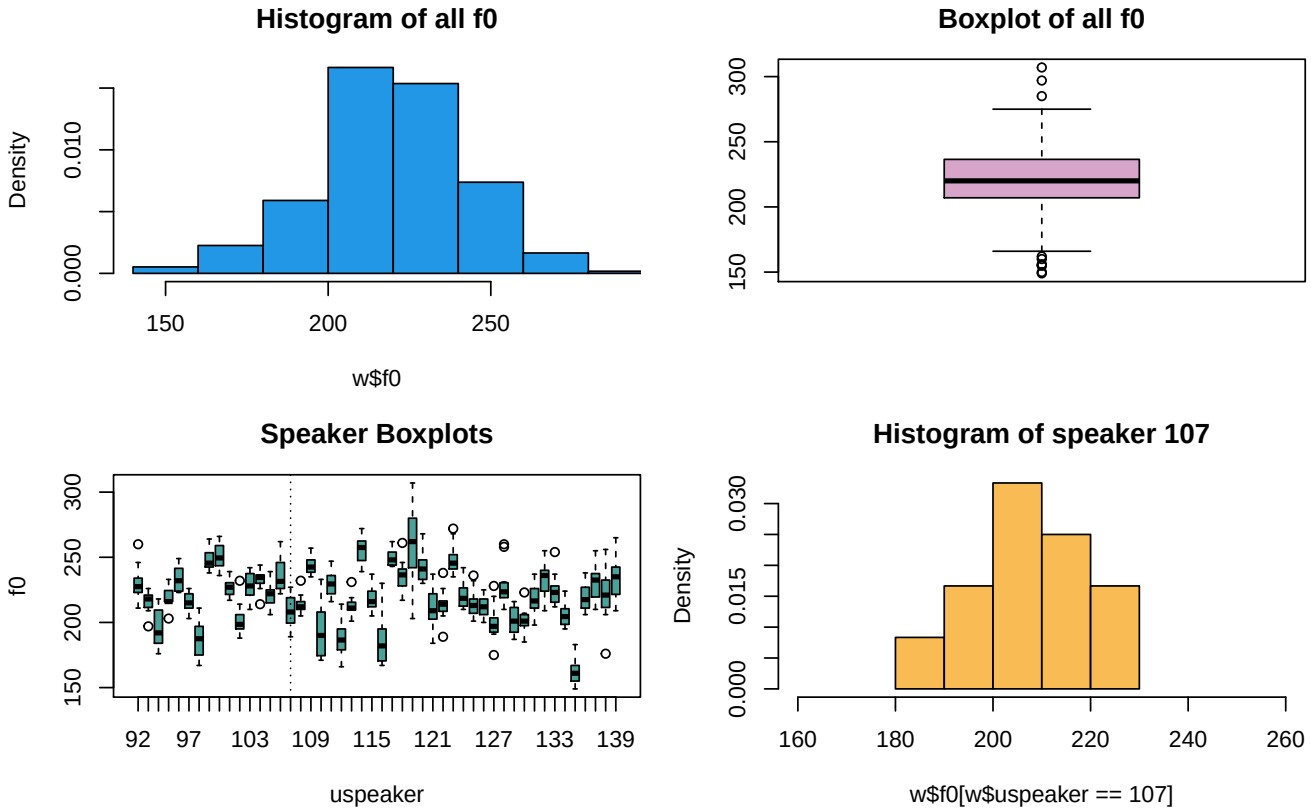
In linguistics, and many other similar fields, almost all of our data is repeated measures data. The methods most commonly-used by linguists (e.g., experiments, interviews, corpora, ... etc.) yield many observations per person, and typically all involve data from multiple people/sources. As a result, the analysis of this data requires that models be able to account for within *and* between speaker variation in our data. Multilevel models address the correlated nature of repeated measures data by estimating multiple sources of variation simultaneously.

Repeated-measures data leads to random variation in parameters that is *indistinguishable* from that of our 'data'. To a large extent, whether something is a parameter or a data point depends somewhat on your perspective. For example, consider the figure below. The top left presents a histogram of all f0 measurements, while the top right presents a boxplot of the same. The bottom left presents the speaker boxplots (one per speaker), each of which resembles the overall boxplot in the top right. We can then zoom in on a single speaker's productions (bottom right) and produce a histogram that suggests a normal distribution reminiscent in shape to the overall aggregated data (top left).

If you are trying to estimate a speaker's mean f0, then the individual productions might be 'data' and the mean can be thought of as a 'parameter'. If you were instead only interested in the population average, maybe now your subject mean is actually just a single data point, and the *population* mean is actually your 'parameter'.

```
par (mfrow = c(2,2), mar = c(4,4,3,1))
hist (w$f0, main = "Histogram of all f0",xlim = c(140, 290), freq = FALSE,
      col=4)
boxplot (w$f0, main = "Boxplot of all f0",col=lavender)
```

```
boxplot (f0 ~ uspeaker, data = w, main = "Speaker Boxplots",col=deepgreen)
abline (v = 16,lty=3)
hist (w$f0[w$uspeaker == 107], main = "Histogram of speaker 107",
      xlim = c(160, 260), freq = FALSE,col=yellow)
```



A multilevel model is able to simultaneously model independent variation at multiple 'levels'. For our f0 data, these are:

- The 'upper' level: Between-speaker variation in mean f0. This can be thought of like variation in $\mu_{speaker}$. Speaker's have an average f0 ($\mu_{speaker}$) that they produce over time. However, speakers are chosen randomly from a larger population, and so any given speaker's $\mu_{speaker}$ is unpredictable a priori.
- The 'lower' level: Within-speaker variation, analogous to ε. When an individual speaker produces speech, their productions will vary around their average from token to token. Our model cannot explain this and so this is 'error'

As seen in the figure above, the variation at the lower and upper levels are analogous. Just like individual speakers will rarely have an average f0 exactly like the population average, individual speakers will rarely produce f0 values exactly at their speaker average. Importantly, variation at the two levels is independent and logically distinct: within-speaker variation can be small or large independently of whether between-speaker variation is large or small.

Basically, each subject's productions form a little normal distribution around their average, and the mix of these little distributions results in the overall 'big' distribution of data across all subjects. By using multilevel models, we can estimate the effects of multiple sources of variation at the same time.

Estimating a multilevel model with `brms`

We are now going to fit the same model we fit above, but with a structure that reflects the repeated-measures nature of the data.

The model

To specify a multilevel model, you need to write a slightly more complicated model formula. This explanation assumes that you have a dataframe or matrix where one column contains the variable you are interested and predicting (in this case `f0`), and another column contains a vector containing unique labels for each speaker or source of data (in this case a unique speaker label `uspeaker`).

To indicate that your model contains an 'upper' level where you have clusters of data coming from different individuals, you have to put another model inside your main model!

Before, the model formula looked like this:

```
f0 ~ 1
```

which meant 'predict f0 using only an intercept'. Now the model formula will look like this:

```
f0 ~ 1 + ( 1 | uspeaker)
```

When you place a predictor in the formula in parenthesis, on the right-hand-side of a pipe, like this `( | predictor )`, you tell `brm` that you expect data to be clustered according to each category represented in the grouping vector. In this case, we are telling `brm` that each unique speaker is a cluster of data. Whatever you put in the left-hand-side of the parentheses `( in here | predictor )` is the model for each subcluster!

So what does this model formula mean: `f0 ~ 1 + ( 1 | uspeaker)`? It tells `brm`: predict f0 based on only an intercept, and also calculate a separate intercept for each speaker. Effectively, this model formula is telling `brm` to figure out all the information presented in the figures above.

This regression model is now something like this:

$$y = \mu_{overall} + \mu_{speaker} + \varepsilon(\#eq : 24)$$

Recall that when we predict an intercept in a regression model, we actually perform the following decomposition $\mu = parameter * 1$. This means that we are actually just fitting a parameter that we expect to equal the value of the μ we are interested in.

Below, where I have decomposed each μ into its constituent 'predictors' (1) and parameters (a, b).

$$y = a * 1 + b_{speaker} * 1 + \varepsilon(\#eq : 25)$$

In addition to the coefficient estimating the overall intercept (a), we know have another term $b_{speaker}$. This coefficient is actually a set of coefficients since it has a different value for each speaker (its a vector). It has a different value for each speaker because it will reflect variation in $\mu_{speaker}$. However, $\mu_{speaker}$ is a random variable since it reflects the random average f0 of each person drawn from the population. If $\mu_{speaker}$ behaves like a random variable, then the coefficients that reflect this value in our model $(b_{speaker})$ will behave in the same way.

This means that actually our model has *two* random variables. The first one is the error term $\varepsilon \sim \mathcal{N}(0, \sigma_{error})$, which has a mean of 0 and a standard deviation which we can refer to as σ_{error}. The second is the random, speaker-specific intercepts, or by-speaker intercepts, that can also be thought of as random draw from a normal distribution.

A careful consideration of the model in equation 2.4 suggests that this model wouldn't really work. If the overall mean is 220 Hz and a speaker's average is 230, this would suggest a predicted average of 450 $(\mu_{overall} + \mu_{speaker})$ for this speaker. Clearly that is not how the model should be working.

Recall that I previously said that regression models encode *differences*. Our model already encodes the overall data average in the $\mu_{overall}$ parameter. Thus, the speaker-specific averages only need to contain information about *differences* to this overall average. As a result, the model parameters for mean f0 across all speakers will be centered at 0 (i.e., the average), and will tend to be normally distributed with a population-specific standard deviation.

Since our model parameters represent speaker-specific deviations rather than their actual mean f0s, people often use this symbol, γ, for them instead of μ, where $\gamma = \mu_{speaker} - \mu_{overall}$. We can show the expected distribution of this variable below, where $\sigma_{speakers}$ is a population-specific standard deviation term.

$$\gamma_{uspeaker} \sim \mathcal{N}(0, \sigma_{speakers})(\#eq:26)$$

Our overall model is now as shown below, made specific for the data we have, and using expected parameter names.

$$\begin{aligned} f0 &= Intercept + \gamma_{uspeaker} + \varepsilon \\ \gamma_{uspeaker} &\sim \mathcal{N}(0, \sigma_{speakers}) \\ \varepsilon &\sim \mathcal{N}(0, \sigma_{error}) \end{aligned}$$

Each line in the model says the following:

- f0 is equal to the sum the intercept, a speaker-specific deflection from the intercept, and error.
- the speaker intercepts are drawn from a normal distribution with a mean of 0 and a standard deviation of $\sigma_{speakers}$. This distribution represents the random variation of speakers around the average f0 for the population.
- the error is also drawn from a normal distribution with a mean of 0 and a standard deviation of σ_{error}. This distribution represents the random within-speaker variation of productions around the average f0 for the speaker.

There is a very important difference in how the initial and final models we fit view and partition the variation in our model. The initial model we fit viewed the variation in the model like this:

$$\sigma_{total} = \sigma_{error}(\#eq:28)$$

In other words, all variation was error. We don't know why values vary from the mean. Our multilevel model views the variation in our data like this:

$$\sigma_{total} = \sigma_{speaker} + \sigma_{error}(\#eq:29)$$

It sees only *some* of the variation in data as error. Basically, from the perspective of this multilevel model, the variation in the data is a combination of random (but systematic) between-speaker variation, and random within-speaker variation.

Fitting the model

We can fit a model with a formula that appropriately specifies the clustering we expect in our data. As a result, this model can estimate both between- and within-speaker variability.

```
set.seed (1)
multilevel_model =
  brm (f0 ~ (1|uspeaker), data = w, chains = 1, cores = 1)
```

```
## loand and inspect pre-fit model
multilevel_model = readRDS ("2_multilevel_model.RDS")
multilevel_model
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ (1 | uspeaker)
##    Data: w (Number of observations: 576)
```

```
## Samples: 1 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 1000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 48)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    20.19      2.10    16.65    25.06 1.00      135      192
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.39      3.10   214.49   226.13 1.03       51      108
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.54      0.39    11.83    13.34 1.01      400      700
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

This new model contains one new chunk its print statement:

```
Group-Level Effects:
~uspeaker (Number of levels: 48)
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)    20.19      2.10    16.65    25.06 1.00      135      192
```

This sections contains information about the standard deviation of between-speaker averages ($\mu_{speaker}$) in the sample. We can see that the information provided by `brms` is quite similar to what we can estimate directly using our data. However, *brms* does this all for us, in addition to giving us a lot more information.

```
## find mean f0 for each speaker
speaker_means = aggregate (f0 ~ uspeaker, data = w, FUN = mean)
## find the within speaker variance. This is the within-talker 'error'.
speaker_vars = aggregate (f0 ~ uspeaker, data = w, FUN = var)

## the mean of the speaker means corresponds to our overall mean estimate
mean (speaker_means$f0)
```

```
## [1] 220.401
```

```
## sd(Intercept) in the model reflects the amount of variation in talker
## intercepts. This is the between speaker variation in our model. See how it is
## similar to the sd of the actual speaker means.
sd (speaker_means$f0)
```

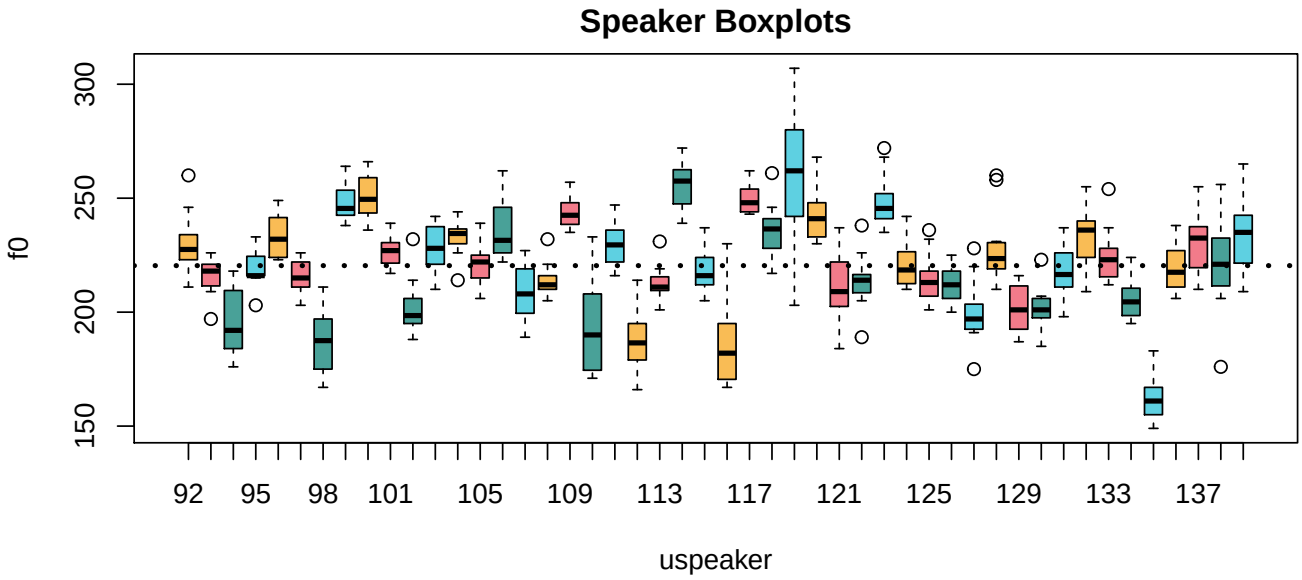
```
## [1] 20.07397
```

```
## sigma in the model reflects the amount of variation in talker intercepts.
## This is the between speaker variation in our model.
sqrt (sd (speaker_vars$f0))
```

```
## [1] 12.42719
```

The overall mean f0 in our data (220.4) corresponds quite well to our model estimate of 220.4. This reflects the central location of the overall distribution below (the horizontal line in the figure below). The standard deviation of the speaker means (Intercept = 20.1) is again very similar to our model estimate (sd(Intercept) = 20.1). This reflects the average distance from each speaker's average, and the overall average. Finally, the average of the within speaker standard deviation in our data (12.4) corresponds closely to our model's error estimate (sigma = 12.5). This reflects the average spread of each speaker's data relative to their own mean, within their own little boxplot.

```
par (mfrow = c(1,1), mar = c(4,4,2,1))
boxplot (f0 ~ uspeaker, data = w, main = "Speaker Boxplots",col=c(yellow,coral,
         deepgreen,teal))
abline (h = 220.4, lwd=3,lty=3)
```



Checking model convergence

Remember that our model parameter estimates consist of a set of samples from the posterior distribution of a parameter. If we don't take enough of these samples, our parameter estimates will be unreliable.

For this reason, it's important to look at the ESS values (the 'expected sample size'), and the 'Rhat' values provided by `brm`. ESS tells you about how many independent samples you have taken from the likelihood. Bulk ESS is how many samples the sampler took in the thick part of the density, and Tail ESS reflects how much time the sampler spent in the thin part, the 'tails'. Rhat tells you about whether your 'chains' have converged (more on this later). As noted above, values of Rhat near 1 are good, and values higher than around 1.1 are a bad sign.

We haven't really taken many samples here, so we can't be confident in our parameter estimates. Ideally we would like several hundred samples (at least) for mean estimates, and thousands to be confident in the 95% confidence intervals.

To get more samples we can run the model longer, or we can use more *chains*. A chain is basically a separate set of samples for your parameter values. Just imagine you had estimated the model 4 times in a row and mixed your estimations. A model can be fit in parallel across several chains, and then the estimates can be merged across chains. When you do this across multiple cores, you can get N times as many samples when you use N cores. Since many computers these days have 4-8 (or more) cores, we can take advantage of parallel processing to fit models faster.

Below, I refit the same model from above but run it on 4 chains, and on 4 cores at once. This doesn't take any longer but it does give us a higher ESS. Just make sure you leave a couple of cores free on your computer when you fit a model!

```
set.seed (1)
multilevel_multicore =
  brm (f0 ~ 1 + (1|uspeaker), data = w, chains = 4, cores = 4)
```

```
## load and inspect pre-fit model
multilevel_multicore = readRDS ('2_multilevel_multicore.RDS')
multilevel_multicore
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ 1 + (1 | uspeaker)
##    Data: w (Number of observations: 576)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 48)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    20.22      2.19    16.54    25.19 1.02      345      615
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.47      2.91   214.67   226.14 1.02      228      543
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.54      0.39    11.81    13.34 1.00     3427     2943
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

If we compare the ESS for this new model to the previous model, we see that using 4 chains has substantially increased our ESS, without taking up any more computing time.

One final tweak that I will be using going forward is to 'thin' samples. Notice that we have collected 'total post-warmup samples = 4000'. This means our model has 4000 samples for every parameter in the model. However, we have only about 400 'effective samples' to show for it for some parameters of interest. This means that a lot of our samples are basically dead weight, taking up space and slowing down computations for no good reason.

Sometimes consecutive samples can be too similar and so don't given you that much independent information. A way to fix this is to run longer chains and keep only every nth one. This lets your models be smaller while containing approximately the same information.

To do this you have to set the `iter`, `warmup` and `thin` parameters. You will keep every sample after the warmup is done, up to the `iter` maximum. So if `iter=3000` and `warmup=1000` you will end up with 2000 samples. After this, you keep only one every `thin` samples. Basically, you will end up with $(iter - warmup)/thin$ samples per chain.

Below, I ask for 11,000 sample per chain, 10,000 post warm-up. However, since I plan to keep only 1/10, I will have 1000 per core, so 4000 samples in total. However, despite having the same number of samples as the `multilevel_multicore`, the ESS for this model is much higher for important parameters such as the model intercept.

```
set.seed (1)
multilevel_thinned =
  brm (f0 ~ 1 + (1|uspeaker), data = w, chains = 4, cores = 4,
       warmup = 1000, iter = 11000, thin = 10)
```

```
## load and inspect pre-fit model
multilevel_thinned = readRDS ('2_multilevel_thinned.RDS')
multilevel_thinned
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ 1 + (1 | uspeaker)
##    Data: w (Number of observations: 576)
## Samples: 4 chains, each with iter = 11000; warmup = 1000; thin = 10;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 48)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    20.06      2.23    16.24    24.98 1.00     2775     3392
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.40      2.95   214.57   226.19 1.00     1987     2625
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.54      0.38    11.81    13.31 1.00     3908     3891
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Specifying prior probabilities

In Chapter 1 we discussed that Bayesian models require that prior probabilities be specified for all parameters. You may have noticed that to this point I haven't discussed priors at all.

If you don't specify prior probabilities for your parameters, `brm` will use a 'flat' prior for all parameters. When you do this, you are basically relying only on the likelihood for your analysis. You are also telling your model that, a priori, *any* value of average f0 is equally believable. Empirically, this is false. As a practical matter this can cause problems for samplers like `brm` (STAN actually). Basically, the sampler has a harder time figuring out the most likely values when you tell it to look anywhere from positive to negative infinity. Even a bit of guidance can help.

`brms` makes it really easy to specify prior probabilities for specific parameters or whole groups of parameters. First we figure out the overall mean and the standard deviation of the data.

```
mean(f0s)
```

```
## [1] 220.401
```

```
sd(f0s)
```

```
## [1] 23.22069
```

In the example below, I use this information to set reasonable bounds on the parameters in the model. I do this by class of parameter:

- Intercept: this is a unique class, only for intercepts.

- b: this is for all the non-intercept predictors. There are none in this model.
- sd: this is for all standard deviation parameters. In our example this is `sd(Intercept)` for `uspeaker` ($\sigma_{speaker}$), and `sigma` (σ_{error}).

Both priors below use a 't' distribution, which is just like a normal distribution but it is more pointy, and has more density in the outer parts of the distribution. I use this because it has good properties, but you can use normal priors, or any other priors that you think work for your model. Rather than focusing on the mathematical properties of priors, the most important thing is that their *shape* reflect the distribution of credible parameter values a priori (before you conducted your experiment).

The format for the priors looks like this `student_t(nu, mean, sd)`, where `nu` is a parameter that determines how pointy the distribution is, and `mean` and `sd` are the same mean and standard deviation parameters from the normal distribution. The nu parameter ranges from 1 to infinity, and large numbers result in a more normal-like distribution.

So, for the overall intercept ($\mu_{overall}$) I am using a prior with the same mean as the data mean, and a standard deviation that is twice as large as the data standard deviation. For both the model standard deviation terms ($\sigma_{error}, \sigma_{speaker}$) I am using a t distribution centered at 0 (explained below), with a standard deviation twice as large as the data standard deviation.

```
set.seed (1)
multilevel_priors =
  brm (f0 ~ 1 + (1|uspeaker), data = w, chains = 4, cores = 4,
       warmup = 1000, iter = 11000, thin = 10,
       prior = c(set_prior("student_t(3, 220.4, 46.4)", class = "Intercept"),
                 set_prior("student_t(3, 0, 46.4)", class = "sd")))
```

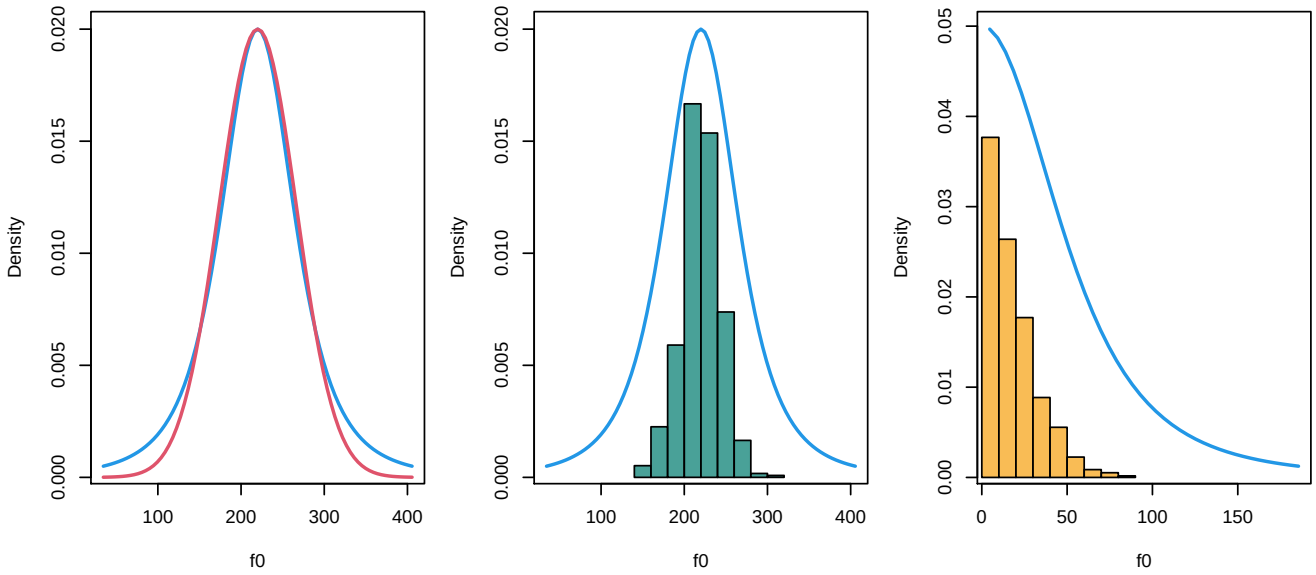
```
## load and inspect pr-fit model
multilevel_priors = readRDS ('2_multilevel_priors.RDS')
multilevel_priors
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ 1 + (1 | uspeaker)
##    Data: w (Number of observations: 576)
## Samples: 4 chains, each with iter = 11000; warmup = 1000; thin = 10;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 48)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    20.19      2.24    16.29    25.06 1.00     3114     3663
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.42      2.97   214.44   226.38 1.00     1796     2841
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.54      0.39    11.81    13.34 1.00     3968     3972
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

In the left panel below (plot code at end of chapter) I compare the t distribution we used (blue) to the equivalent normal distribution (red). It is clear that the t distribution can tolerate more extreme values because of its 'fatter'

tails. As we will discuss later, using t distributions can make our models more robust to outliers. This is really important in linguistics where subjects/speakers sometimes do weird stuff!

In the middle panel we compare this prior to the data, and see that the prior distribution is much broader (more vague) than the data distribution. The right panel compares the prior for the standard deviation parameters to the absolute value of the centered f0 data. This presentation shows how far each observation is from the mean f0 (at 220 Hz). Again, the prior distribution we have assigned for these parameters is much larger than the variation in the data. As a result, neither of these priors is going to have much of an effect on our parameter estimates.



If we compare the output of this model to `multilevel_thinned`, we see that specifying a prior has has no noticeable effect on our results. This is because the prior matters less and less when you have a lot of data, and because we have set wide priors that are appropriate (but vague) given our data. Although the priors may not matter much for models as simple as these, they can be very important when working with more complex data, and are a necessary component of Bayesian modeling.

Answering our research questions

Let's return again to the research questions I posed initially:

1) What is the average f0 of the whole *population* likely to be?
2) Can we set bounds on likely mean f0 values based on the data we collected?

And we can compare the answers provided to this question by our initial and final models.

```
model
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ 1
##    Data: w (Number of observations: 576)
## Samples: 1 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 1000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.40      0.97   218.33   222.30 1.00      851      557
```

```
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    23.24      0.69    21.99    24.61 1.00      653      550
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
multilevel_priors
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ 1 + (1 | uspeaker)
##    Data: w (Number of observations: 576)
## Samples: 4 chains, each with iter = 11000; warmup = 1000; thin = 10;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 48)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    20.19      2.24    16.29    25.06 1.00     3114     3663
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   220.42      2.97   214.44   226.38 1.00     1796     2841
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.54      0.39    11.81    13.34 1.00     3968     3972
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Our initial model (`model`) and our final model (`multilevel_priors`) agree on what the average f0 is. However, they disagree on a credible interval for that parameter. Our initial model did not specify information about repeated measures. This causes our model to think that it has more independent observations than it does, and so it returns an overly-precise estimate.

Another difference is that the final model has a much smaller `sigma` parameter (12.5 vs 23.2). This indicates that the error (ε) is much smaller in the final model than in the initial model. Keep in mind that 'error' is just what your model can't explain. Our final model explains much more and so there is less error. The reduced error is a direct result of the fact that the final model splits random variation up into between-speaker and within-speaker components, estimating the between-speaker variation ($sigma_{speaker}$) to be about 20 Hz.

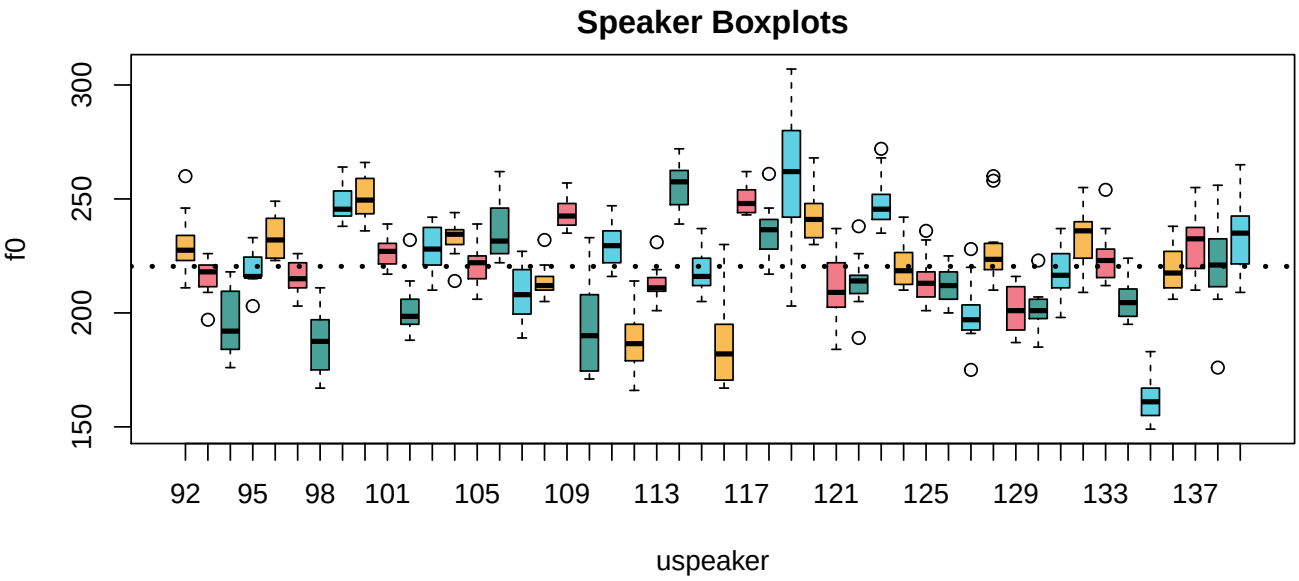
Usually, when I report parameters I provide the mean and standard deviations of the posterior distribution, in addition to the bounds of the 95% credible interval of the parameter. Based on the result of our final model, I think a thorough description of the general properties of our data might go something like:

> "Based on our model the average f0 produced by adult females in Michigan is likely to be 220 Hz (s.d. = 2.97, 95% CI = 214.4, 226.4). However, consistent between-speaker variation averages about 20 Hz (s.d. = 2.24, 95% CI = 16.29, 25.06), meaning that we can expect the average f0 produced by individuals to deviate substantially from 220 Hz. Finally, the standard deviation of production error was about 12.5 Hz (s.d. = 0.39, 95% CI = 11.81, 13.34) indicating that the amount of random between speaker variation in production is about half the magnitude of the stable, between-speaker differences in f0.

I am including the speaker boxplots below because I think this image basically presents the same information as the paragraph above, but in visual form. In general, any data relationship or result can be presented in a figure, and the relationships presented in a figure can also be expressed as a mathematical model.

When you are thinking about the relationships in your data, or that you expect in your data, its a good idea to think: what kind of picture could illustrate this relationship? Conversely, if you see a figure of your results that you feel really expresses something interesting about your data you should think, how can these relationships be represented in a model?

```
par (mfrow = c(1,1), mar = c(4,4,2,1))
boxplot (f0 ~ uspeaker, data = w, main = "Speaker Boxplots",col=c(yellow,coral,
         deepgreen,teal))
abline (h = 220.4, lwd=3,lty=3)
```



Plot Code

Plot for comparison of prior distributions with f0 data

```
# get t density
x1 = seq (-4,4,.1)
y1 = dt (x1, 3); y1 = y1 / max (y1) / 50
x2 = 220+(x1*46.4)
y2 = dnorm (x2, 220, 46.4); y2 = y2 / max (y2) / 50

par (mfrow = c(1,3), mar = c(4,4,1,1))
## plot t
plot (x2, y1, type = 'l', lwd = 2, ylab = 'Density', xlab = 'f0', col = 4)
## compare to equivalent normal
lines (x2, y2, lwd=2,col=2)

## plot t
plot (x2, y1, type = 'l', lwd = 2, ylab = 'Density', xlab = 'f0', col = 4)
## compare to equivalent normal
hist (f0s, add = TRUE, freq = FALSE)

## plot prior for standard deviations
```

```
x3 = x2 - mean (x2)
y3 = dt (x1, 3); y3 = y3 / max (y3) / 20
plot (x3[x3>0], y3[x3>0], type = 'l', lwd = 2, ylab = 'Density',
      xlab = 'f0', col = 4)
hist (abs (f0s - mean (f0s)), add = TRUE, freq = FALSE)
```

Comparing two groups

In the previous chapter, I focused on investigating values from a single group, which is the simplest data you can deal with. In this chapter we are going to compare data from two groups to see how similar/different they really are. The models we will discuss are similar to two-sample t-tests (and some ANOVA designs) in the types of research questions they can help investigate.

Data and research questions

We are going to be using the same Hillenbrand et al. f0 data we used in the other chapters, but this time we are going to compare the f0 measurements for adult females to those of girls between 10-12 years of age.

Below I load in the Hillenbrand et al. data, and include a new variable that indicates whether the talker is an adult or a child. I also split the data up by gender. The variable `uspeaker` is a speaker number that is unique across all speaker groups, and `type` indicates speaker group from among `b` (boys), `g` (girls), `m` (men), and `w` (women).

```
url1 = "https://raw.githubusercontent.com/santiagobarreda"
url2 = "/stats-class/master/data/h95_vowel_data.csv"
h95 = read.csv (url(paste0 (url1, url2)))
## make variable that indicates if the talker is an adult
h95$adult = ""
h95$adult[h95$type %in% c('w','m')] = "adult"
h95$adult[h95$type %in% c('g','b')] = "child"

## split up data by into male and female groups, only some columns
males = h95[h95$type %in% c('m','b'),c('f0', 'adult', 'type', 'uspeaker')]
females = h95[h95$type %in% c('w','g'),c('f0', 'adult', 'type', 'uspeaker')]
f0s = females[['f0']]

# re-factor to remove excluded subjects
males$uspeaker = factor (males$uspeaker)
# re-factor to remove excluded subjects
females$uspeaker = factor (females$uspeaker)
```

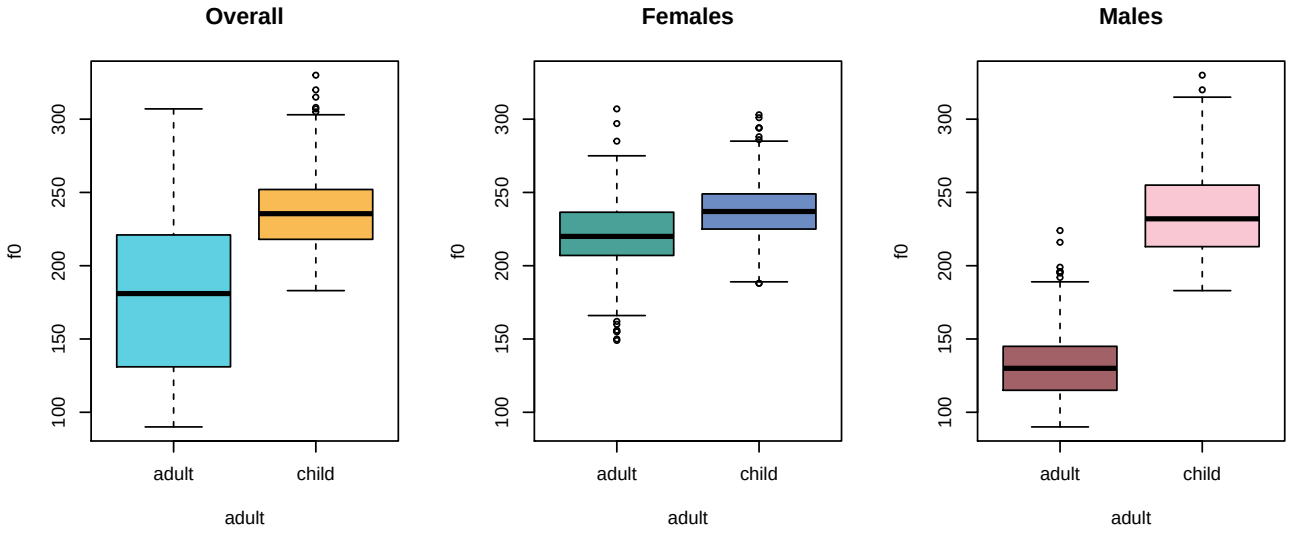
One of the simplest questions a researcher can ask (from a modeling perspective) is: Are two groups of observations different or are they the same? For example in phonetics, researchers ask questions like, have these vowels merged in this dialect (are these two things different)? Does visual information speed up speech perception or not (are two sets of reaction times the same)?

A good way to isolate a single difference is to create groups that differ primarily according to the characteristic you are interested in testing, but are otherwise the same 'on average'. For the reaction-time example above, I might present the same listeners with very similar words in the same conditions, but half with and half without visual information. The difference in reaction times between the groups should reflect the advantage provided by visual information. The question would then be, are reaction times with visual information *the same thing* as reaction times without visual information? Are these two sets of observations samples from the same population? Or is

the set of observations "reaction times for speech recognized *with* visual information" a different set of things than "reaction times for speech recognized *without* visual information"?

In our data we have male and female talkers of the same dialect, who differ primarily in terms of age. Basically, these groups differ mostly in terms of 'adultness', and so we can use the characteristics of these groups to investigate the effect of 'adultness' on average f0. Below I present boxplots of the effect of adultness on f0, first overall and then divided by speaker gender.

```
par (mfrow = c(1,3))
boxplot (f0 ~ adult, data = h95, main = "Overall", ylim = c(90,330),
         col = c(teal, yellow))
boxplot (f0 ~ adult, data = females, main = "Females", ylim = c(90,330),
         col = c(deepgreen,skyblue))
boxplot (f0 ~ adult, data = males, main = "Males", ylim = c(90,330),
         col = c(maroon,lightpink))
```



Clearly there is a difference in f0 between children and adults, but the difference also seems to be gender-dependent: there is more of a difference between men and boys than between women and girls. We are going to focus on the female data for now, and I leave the male data so that the reader can modify the analysis presented here to analyze that data.

Below, the figure on the left highlights both between- and within-speaker variation in f0 by women (cyan) and girls (red). On the right, we see the overall distribution (black) compared to the distribution of f0 for women (cyan) and girls (red).

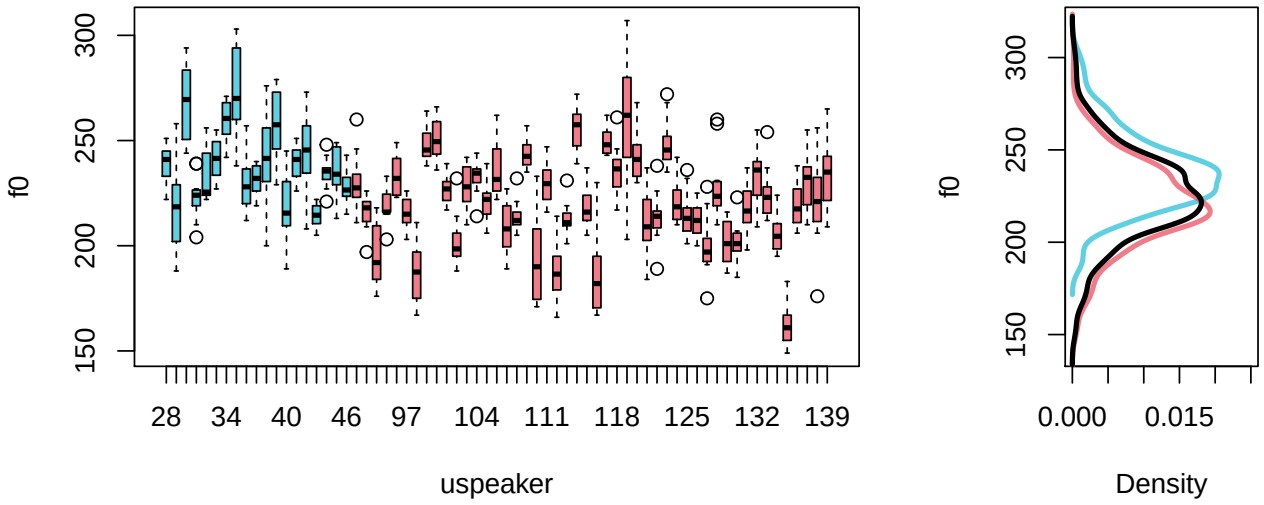
Obviously the distributions seem a *bit* different, but they are also not *that* different. We can also clearly see that there is substantial overlap between individual girls and women (in the figure on the left). As a result, a cautious analysis of this data would recognize this overlap even if somehow 'differentess' were found. We are going to fit a model that can help us quantify all of the variability shown in the figures below.

```
colors =
  c(coral,teal)[ apply (table(females$uspeaker, females$adult),1,which.max) ]

par (mfrow = c(1,2)); layout (mat = t(c(1,2)), widths = c(.7,.3))
boxplot (f0 ~ uspeaker, data=females, col = colors)

## I rotate the density figures so that you can see how these correspond to
## the boxplots on the left.
tmp = density (females$f0[females$adult=="child"])
```

```
plot (tmp$y, tmp$x, lwd = 3, col = teal, ylab = "f0",xlab="Density",
      ylim = c(140,320), xlim = c(0,0.025), type = 'l')
tmp = density (females$f0[females$adult=="adult"])
lines (tmp$y, tmp$x, lwd = 3, col = coral)
tmp = density (females$f0)
lines (tmp$y, tmp$x, lwd = 3, col = 1)
```



Estimating the difference two means with 'brms'

In chapter 2 the model we fit had the simplest possible formula, just an intercept. Here, we need to extend this to include an actual predictor: a vector indicating 'adultness'. Remember that formulas look like this `variable ~ predictor`. So, if we want to predict f0 based on whether the talker is an adult or not, our model is going to have a formula that looks basically like this `f0 ~ adult + (1|uspeaker)`. You can think of this meaning something like 'We expect the distribution to vary based on whether the talker is an adult or not'.

Note that the model calculates an intercept (mean value) for each speaker, but does **not** include an effect for `adult` for each speaker (i.e., the model does *not* include a term like `(adult|uspeaker)`). This is because we are estimating a mean for each speaker, but not the effect for 'adultness' for each speaker. Each speaker is only either an adult or a child, and so we cannot estimate the effect for 'adultness' for each speaker. Doing things that don't 'make sense' from the model's perspective will cause it to crash or return strange values.

I'm not going to go into so much detail about the structure of the regression model right now because an explanation involves some of the less intuitive concepts relating to regression. We are going to to fit the model first and discuss its structure, and then get to the details of the model later.

Fitting the model

First we find the mean and the standard deviation of our data to use in the priors. As before, I intend to use the data mean for the intercept, and use a standard deviation twice as wide as the data standard deviation.

```
## establish data statistics for priors
mean (f0s)
```

```
## [1] 225.4913
```

```
sd (f0s)
```

```
## [1] 23.98959
```

We load the `brms` package and fit the model. Notice that I now include a prior for `class = "b"`, which is the class for all predictors that are not the intercept. Our model now includes a non-intercept term, and so I need to include this prior:

```
library (brms)
```

```
set.seed (1)
model =
  brms::brm (f0 ~ adult + (1|uspeaker), data = females, chains = 4, cores = 4,
       warmup = 1000, iter = 11000, thin = 10,
       prior = c(set_prior("student_t(3, 225.5, 48)", class = "Intercept"),
                 set_prior("student_t(3, 0, 48)", class = "b"),
                 set_prior("student_t(3, 0, 48)", class = "sd")))
```

```
## load and inspect pre-fit model
model = readRDS ('3_model.RDS')
model
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ adult + (1 | uspeaker)
##    Data: females (Number of observations: 804)
## Samples: 4 chains, each with iter = 11000; warmup = 1000; thin = 10;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 67)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    19.10      1.77    16.00    22.88 1.00     2781     3089
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept    220.47      2.80   215.10   226.16 1.00     1827     2813
## adultchild    17.63      5.22     7.34    27.88 1.00     1987     2068
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.93      0.34    12.28    13.59 1.00     3537     3773
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

The output is mostly familiar, but there is a new predictor in the section on `Population-Level Effects`:

```
           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    220.47      2.80   215.10   226.16 1.00     1827     2813
adultchild    17.63      5.22     7.34    27.88 1.00     1987     2068
```

In addition to the 'Intercept' term, we now get estimates for a term called `adultchild`. Admittedly, this is a strange name, but its how R handles predictors that are words (called 'factors' in R). What this tells us is that this is the estimate for the 'child' level of the 'adult' factor. '

A couple of questions arise. First, the 'Intercept' term in the model above seems to correspond to the mean f0 for adult females. We can confirm this:

```
## calculate means of f0 based on values of adult vector
tapply (females$f0, females$adult, mean)
```

```
##    adult    child
## 220.4010 238.3509
```

However, the estimate for children is 17.6 Hz, which is odd. This is obviously not the mean f0 for the girls in our sample. Why not? To understand model coefficients we are going to have to talk about contrasts

Contrasts

Contrasts are the numerical implementation of factors in your model. Factors are variables like adult vs. child that are not inherently numerical. You may initially think that we can separately estimate the women's average, the girl's average, and the overall mean. However, our models can't actually do this. The general problem is this: if you have two groups you can't independently calculate:

1) group 1 mean.
2) group 2 mean.
3) the overall mean.

Why not? Because once you know 2 of those things you know the 3rd. For example, if the group 1 mean is 5 and the overall mean is 6, obviously the group 2 mean **must** be 7. Why does this matter? Because when things are entirely predictable based on each other, they are not actually separate things, even though they may seem that way to us. When things are entirely predictable in this way we say they are linearly dependent, and regression models don't like this. Here's three perspectives on why this is a problem:

1) Imagine you were trying to predict a person's weight from their height. You want to include height in centimeters *and* height in meters in your model, and you want to independently estimate effects for both predictors. Since height in centimeters = height in meters / 100, that is obviously not going to be possible. The effect of one must be 100 times the effect of the other! Even though it may be less transparent, this is the same reason why we can't estimate all the group means *and* the overall mean.

2) With two groups, or any two points in a space, you can estimate one distance, not two. If each group could really be a different distance from the mean, you would need to estimate *two* distances. Instead, we are really only in a position to estimate *one* difference, that between our two group averages.

3) When we had one group we obviously couldn't get the overall mean independently from the sample mean. All we had was the sample mean, and that was our best estimate of the population mean too. Adding 1 more group allows us to calculate 1 more mean. Why would it let us calculate 2 more means? That would mean adding a second group (with 1 mean) somehow contributed twice as much information as the first group did. Instead, adding a second mean changes our best guess for the population mean: it is now between the two groups. However, this information is in no way independent from the two group means in our design.

Treatment coding

The coding scheme you use determines how your model represents the differences it encodes. In the model above we used 'treatment' coding (the default in R). In treatment coding, a 'reference' level is chosen to be the intercept, and all group effects reflect the difference between the group mean and the mean for the reference level.

By default, R chooses the alphabetically-lowest level to be the reference level. That is why the Intercept in our model is equal to the mean of the 'adult' group, the average for adult females. The effect for 'child' (`adultchild`) represents the difference between the child mean and the adult mean. This means that our credible intervals also represent the *difference* in the means and not the means themselves. So, we expect the *difference* between girls and women in this sample to be about 17 Hz, and we think there is a 95% chance that the *difference* between the means is between 7.3 and 27.9 Hz in magnitude.

We can see how the effects estimates in our model resemble the means, or differences between means, in our sample.

```
# calculate group means
tapply (females$f0, females$adult, mean)
```

```
##    adult    child
## 220.4010 238.3509
```

```
# find the difference between them
diff (tapply (females$f0, females$adult, mean))
```

```
##    child
## 17.94984
```

To interpret treatment coded coefficients remember:

- The reference category mean is the 'Intercept' in the model.
- The value of the coefficients of any other group mean will be equal to `group mean - reference group mean`.
- To recover the mean estimate for any other group, we add `group mean + reference group mean`.

Sum coding

There are multiple options for coding schemes, and the best one for you depends on what you want to get out of the model. The only other alternative to treatment coding that we will talk about is what is called 'sum coding'. It is a very useful coding scheme for many of the designs most commonly-used by linguists, and it is the coding shceme we will be using going forward.

In sum coding, there is no reference level. Instead, the model Intercept represents the overall mean of all your groups, and group differences are represented with respect to the mean of your data. Just like for treatment coding, you can't estimate all of your group means. For treatment coding, R selects the *alphabetically last* level of your factor, and does not estimate it. Under sum coding, the missing factor level will always be equal to the **negative sum** of the other factors. This means you add up the values of the levels that *are* present, and flip the sign. The outcome is the value of your missing level.

As discussed earlier, with only two groups if you know the overall mean and the distance of one group to the mean, you also know the distance of the other group to the mean. This can be seen quite clearly below where the difference of each group to the overall mean is exactly -8.97. So, if our model tells us that the mean is 229.4 and the adult group is -8.97 Hz below this, then the child group must be the negative sum of the other coefficients. In this case there is only one so we just flip the sign on it (i.e., - (-8.97)).

```
# calculate group means
means = tapply (females$f0, females$adult, mean)
mean (means)
```

```
## [1] 229.376
```

```
# find the distances to the overall mean
means - mean (means)
```

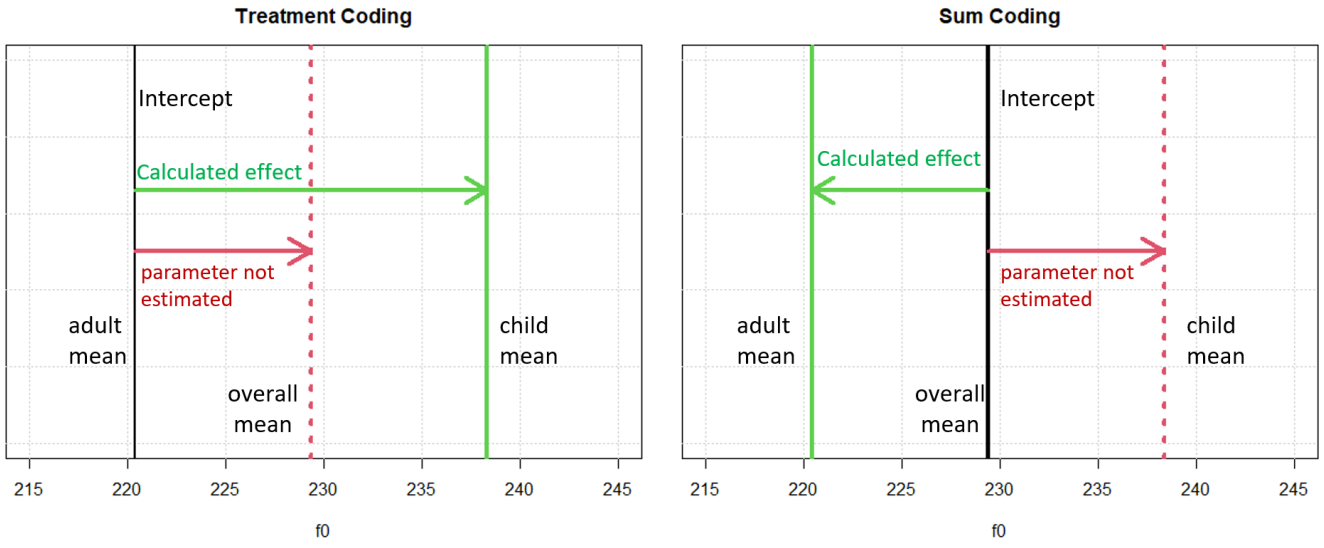
```
##     adult     child
## -8.974918  8.974918
```

To interpret treatment coded coefficients remember:

- The overall mean of all your groups is the 'Intercept' in the model.
- The value of the coefficients of any other group mean will be equal to `group mean - overall mean`.
- To recover the mean estimate for any other group, we add `group mean + overall mean`.

Comparison of sum and treatment coding

The image below shows a comparison of the way the two coding schemes treat our data. In each case they estimate 1 intercept and one effect, letting you recreate 1 other mean (i.e., they each omit one parameter). In treatment coding this is the overall mean, which in the 2-group case will always be `Intercept + Effect/2`. In the case of sum coding this is the effect for the second group, which will always be the same magnitude but have the opposite sign as the effect for the first group.



Refitting the model with sum coding

We are going to re-fit the model we fit above using sum coding, and see how the coefficients change.

Fitting the model

To fit a model with sum coding, we change the global contrast options in R. These options will be in effect until we restart R or change the contrasts to something else.

```
## to change to sum coding
options (contrasts = c('contr.sum','contr.sum'))
## to change back to treatment coding
#options (contrasts = c('contr.treatment','contr.treatment'))
```

We can fit the model with sum coding using the exact same code since the options have changed. Please keep in mind you will need to set this every time you start R, as it will reset to treatment coding each time it restarts.

```
set.seed (1)
model_sum_coding =
  brms::brm (f0 ~ adult + (1|uspeaker), data = females, chains = 4, cores = 4,
       warmup = 1000, iter = 11000, thin = 10,
       prior = c(set_prior("student_t(3, 225.5, 48)", class = "Intercept"),
                 set_prior("student_t(3, 0, 48)", class = "b"),
                 set_prior("student_t(3, 0, 48)", class = "sd")))
```

```
## load and inspect pre-fit model
model_sum_coding = readRDS ('3_model_sum_coding.RDS')
model_sum_coding
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: f0 ~ adult + (1 | uspeaker)
##    Data: females (Number of observations: 804)
## Samples: 4 chains, each with iter = 11000; warmup = 1000; thin = 10;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~uspeaker (Number of levels: 67)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)    19.13      1.80    16.01    22.96 1.00     2892     3144
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   229.33      2.71   224.07   234.81 1.00     2169     2758
## adult1       -8.89      2.65   -14.19    -3.55 1.00     2085     2825
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    12.95      0.34    12.30    13.62 1.00     3922     3710
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

A comparison of these results to that of our first model shows that the main difference is that, as expected, the Intercept now reflects the overall mean and the single parameter reflects the distance of the adult mean to the overall mean.

Note that the parameter is now called `adult1`. This is just how `brm` handles factors. Predictors representing factors will be named `factornameN`, where `factorname` is the predictor name and `N` is the level number. Levels are ordered alphabetically starting at one, and the alphabetically-last level will not be estimated.

```
sort ( unique (females$adult))
```

```
## [1] "adult" "child"
```

So, `adult1` in our model corresponds to the "adult" level of our predictor, and `adult2` *would* be "child", but it is not separately estimated by our model (since `adult2 = -adult12`).

The model

Regression models try to break up values into their components. This is why effects are expressed in terms of differences to some reference value. For example, if we say that that a person's average f0 is usually 220 Hz and under some condition their average f0 is also 220 Hz, then we could say that this condition has no *effect* on their f0. To say that this condition has no effect on mean f0 is to say that it causes no difference in mean f0.

On the other hand something that *does* cause a difference in mean f0 *does* have an effect on mean f0. We can express the effect of something in terms of the difference it causes, for example, saying that under so and so conditions a person will tend to raise their f0 by 20 Hz, relative to some reference value (the intercept).

More generally, we can think of any variable as the sum of a bunch of independent *effects*. This is just a way to think about variables, to break up observed values into their component parts. It should not be confused with the *reality* of these values and the processes that underlie them (whatever that is!).

So far we have covered the fact that after picking a value to use as a reference point (the model intercept), our models:

- represent group means as deviations from the intercept.
- represent the speaker-specific parameters $\gamma_{uspeaker}$ as being centered at 0, with a standard deviation of $\sigma_{speaker}$).
- represent the random error (ε) as having a mean of 0 and a standard deviation of σ_{error}.

In each case, these reflect *deviations* from some reference point. As a result, when the parameters associated with different effects equal 0, this means that no effect is present.

- when group coefficients are 0 the group lies exactly at the intercept. In sum coding this is the overall mean.
- when a speaker-effect is 0 this speaker is exactly average with respect to their group.
- when an error is 0 this production is exactly as expected for a given speaker.

Based on thinking of our predictors as representing deviations from some reference value, We can 'break up' any observed value into its component parts. For example, suppose that the the overall mean is 229 Hz, the adult female mean is 220 Hz, and that a particuar speaker has a mean f0 of 240 Hz. If we observe a value of 256 Hz for this speaker, that suggests the following decomposition:

256 = 229 (Intercept) - 9 (adult female mean) + 20 (speaker effect) + 16 (error)

This reflects the following considerations:

- the average f0 across the groups is 229 Hz.
- the average for adult females is 9 Hz below the overall mean.
- this speaker's average f0 is 20 Hz above the average for adult females.
- this particular production is 16 Hz higher than expected for this particular speaker.

nother observation from this talker might be:

237 = 229 (Intercept) - 9 (adult female mean) + 20 (speaker effect) - 3 (error)

In this case, the error is -3 since the production is now 3 Hz *below* the speaker average. Regressions models basically carry out these decompositions for us, and present information regarding these decompositions in the model summaries.

The full model specification, including prior probabilities is below. I used the same ordering format for the t-distributions that `brm` uses (nu, mean, sd).

$$y = Intercept + adult1 + \gamma_{uspeaker} + \varepsilon$$

Random Variables:

$$\gamma_{speaker} \sim \mathcal{N}(0, \sigma_{speaker})$$
$$\varepsilon \sim \mathcal{N}(0, \sigma_{error})$$

Priors:

$$Intercept \sim \sqcup(3, 225.5, 48)$$
$$adult1 \sim \sqcup(3, 225.5, 48)$$
$$\sigma_{error} \sim \sqcup(3, 0, 48)$$
$$\sigma_{speaker} \sim \sqcup(3, 0, 48)$$

Since our models are getting more complicated, I am omitting the (1) predictor from the nominal variables (Intercept, adult1), and will keep doing so going forward.

Interpreting the two-group model

The `brms` package has several functions that make getting information from these models simple. The `fixef` function gets you means and 95% credible intervals for all your 'population level' parameters (sometimes called 'fixed' effects).

```
brms::fixef (model_sum_coding)
```

```
##             Estimate Est.Error      Q2.5      Q97.5
## Intercept 229.327715  2.711261 224.07166 234.810010
## adult1     -8.888301  2.651641 -14.18979  -3.550734
```

To recover the group means for women and girls, we need to combine the Intercept and the group parameters. We're not actually allowed to just add the values you see above (for good, but technical reasons).

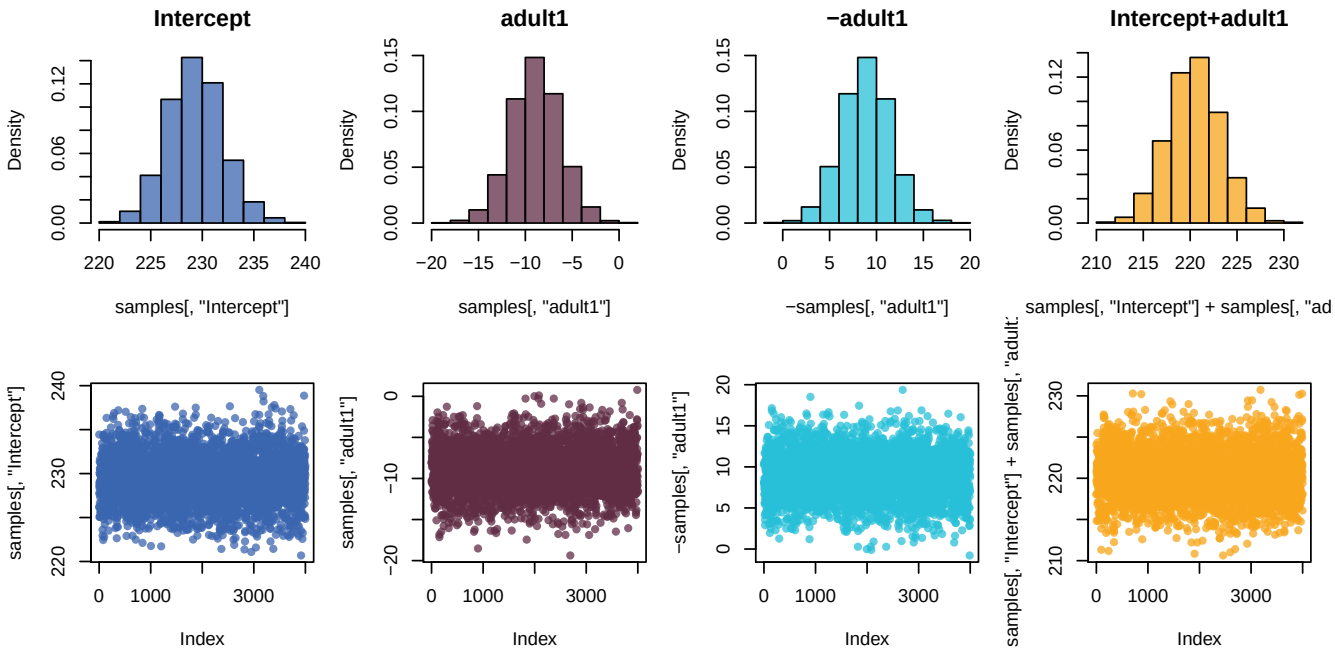
What you actually need to do is add *the individual samples for parameters*, and then inspect the output. You can see the individual samples by calling the `fixef` function and setting `summary` to `FALSE`. Below I plot the individual samples and histograms for the `Intercept` (the overall mean), the `adult1` (the effect for adults) parameter, and the negative of the `adult` parameter (the effect for girls). I also present the combination of `Intercept+adult1`, which yields an estimate of the adult female mean.

```
samples = brms::fixef (model_sum_coding, summary = FALSE)
head(samples, 10)
```

```
##           parameters
## iterations Intercept     adult1
```

```
##       [1,]  226.1704  -8.121943
##       [2,]  234.4365 -10.380060
##       [3,]  228.6168  -7.901476
##       [4,]  226.2287  -8.130842
##       [5,]  225.0607 -10.228533
##       [6,]  227.6880  -6.691685
##       [7,]  227.8307  -8.078614
##       [8,]  230.0513 -10.569158
##       [9,]  228.3428  -8.612124
##      [10,]  228.9045  -5.617644
```

```
par (mfrow = c(2,4), mar = c(4,4,3,1))
hist (samples[,'Intercept'],freq=FALSE, col = skyblue,main='Intercept')
hist (samples[,'adult1'], freq=FALSE, col = deeppurple,main='adult1')
hist (-samples[,'adult1'], freq=FALSE, col = teal,main='-adult1')
hist (samples[,'Intercept']+samples[,'adult1'], freq=FALSE,
      col = yellow,main='Intercept+adult1')
plot (samples[,'Intercept'], col = skyblue,pch=16)
plot (samples[,'adult1'], col = deeppurple, pch=16)
plot (-samples[,'adult1'], col = teal, pch=16)
plot (samples[,'Intercept']+samples[,'adult1'], col = yellow,pch=16)
```



We can then summarize the sum of the parameters using the `posterior_summary` function, resulting in a mean, standard deviation, and credible interval for the new parameter:

```
## calculate child mean
adult_mean = samples[,'Intercept'] + samples[,'adult1']
## report mean and spread of samples
brms::posterior_summary (adult_mean)
```

```
##      Estimate Est.Error     Q2.5    Q97.5
## [1,] 220.4394  2.871496 214.8694 226.2171
```

Luckily, there is a function in `brms` called `hypothesis` that helps us add terms very easily, without having to do any of the above steps. You can ask the `hypothesis` function to add terms in your model (spelled just as they are

in the print statement), and to compare the result to some number. If you compare the result to 0, it just tells you about the result of the terms you added.

```
brms::hypothesis(model_sum_coding, "Intercept + adult1 = 0")
```

```
## Hypothesis Tests for class b:
##                Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio
## 1 (Intercept+adult1) = 0   220.44      2.87   214.87   226.22         NA
##   Post.Prob Star
## 1        NA    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

The output above tells us what our estimate is for `Intercept + adult1`, which we know to be the expected mean f0 for adult females. Whereas the credible interval for the original effect reflected uncertainty in difference between means, the credible interval provided is now for the actual girl's mean, not for the difference between the means.

We can use the hypothesis function to confirm similar results for the model fit using treatment coding. In that model, the reference category was the adult mean. So, if we call the `hypothesis` function on the intercept of the treatment-coding model, we can see that it will present similar values to those seen above.

```
brms::hypothesis(model, "Intercept = 0")
```

```
## Hypothesis Tests for class b:
##        Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio Post.Prob
## 1 (Intercept) = 0   220.47       2.8    215.1   226.16         NA        NA
##   Star
## 1    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

We can check several parameter combinations simultaneously. Below I recreate all our mean estimates of interest, first for the sum coding model, and then for the treatment coding model. Notice that these models contain the same information, just represented in different ways.

```
brms::hypothesis(model_sum_coding,
                 c("Intercept = 0",   # overall mean
                   "Intercept + adult1 = 0",  # adult mean
                   "Intercept - adult1 = 0")) # child mean
```

```
## Hypothesis Tests for class b:
##                Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio
## 1        (Intercept) = 0   229.33      2.71   224.07   234.81         NA
## 2 (Intercept+adult1) = 0   220.44      2.87   214.87   226.22         NA
## 3 (Intercept-adult1) = 0   238.22      4.53   229.34   247.13         NA
##   Post.Prob Star
## 1        NA    *
## 2        NA    *
## 3        NA    *
```

```
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

```
brms::hypothesis(model,
                 c("Intercept + adultchild/2 = 0",   # overall mean
                   "Intercept = 0",  # adult mean
                   "Intercept + adultchild = 0")) # child mean
```

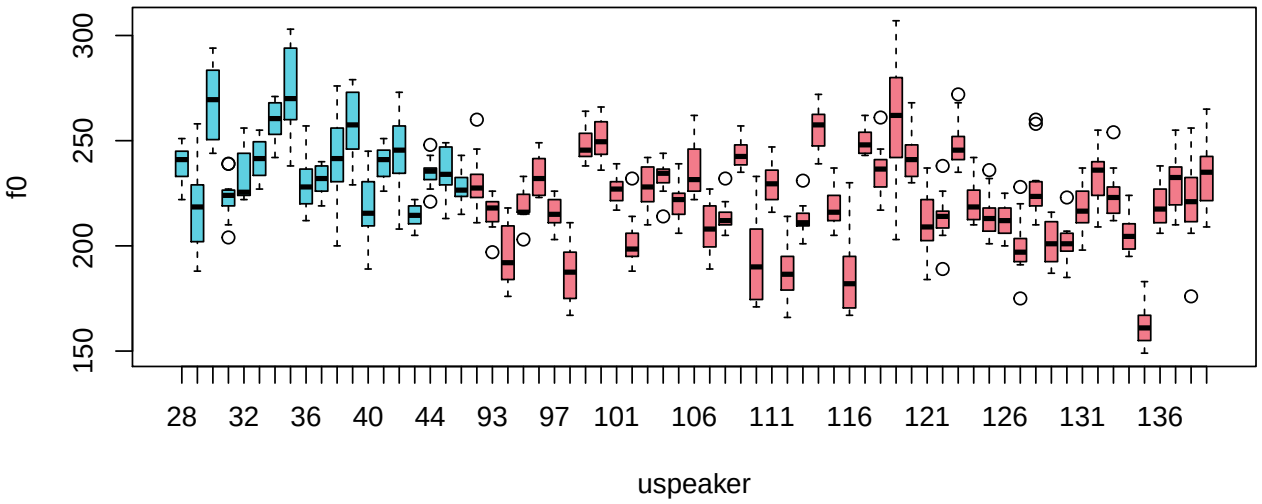
```
## Hypothesis Tests for class b:
##                 Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio
## 1 (Intercept+adultc... = 0   229.28      2.69   224.19   234.74         NA
## 2          (Intercept) = 0   220.47      2.80   215.10   226.16         NA
## 3 (Intercept+adultc... = 0   238.10      4.51   229.41   247.26         NA
##   Post.Prob Star
## 1        NA    *
## 2        NA    *
## 3        NA    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

If I were writing this is in a paper, at this point I could present this information in a paragraph. Based on the sum coded model, I would say something like:

"The overall mean f0 across all speakers was 229 Hz (sd = 2.7, 95% CI = 224, 234). Adult female mean f0 was 220 Hz (sd = 2.9, 95% CI = [215, 226]), while the mean f0 for girls was 228 Hz (sd = 4.28, 95% CI = [230, 246]). The difference between the group means was 18 Hz on average (sd = 5.3, 95% CI = [7, 28]), suggesting a small but noisy difference between groups, on average."

'Random' Effects

Below I present the speaker boxplot, with different colors for each group. So far we have discussed the speaker-dependent deviations from averages, but we haven't actually done anything with them. One of the nice things about multilevel models is that we can actually estimate these parameters, and use them to answer our research questions.



Multilevel models are sometimes also called 'random effects' or 'mixed effects' models. What is 'mixed' about them? Researchers often talk about whether effects are 'fixed' or 'random'. The general idea is that 'fixed' effects are specifically chosen from a small set of possibilities, and we are not necessarily interested in the 'other' levels.

For example, in this experiment we include children 10-12 and adults. Speaker age-groups don't really come from an infinite set of independent age-groups, not ones that will meaningfully affect our research questions anyways. Instead, categories like 'adult' and 'child' are chosen from a finite set of meaningful categories, and we are *specifically* interested in the categories we've chosen. In this case, I really do care about mean f0 for adults and not some other category I didn't sample.

In contrast, 'random' effects are not chosen arbitrarily but at random. The speakers in our sample *do* form part of a practically infinite sample. I am actually *not* interested in the speakers I have but in what this says about the categories I did not include. In fact, most researchers would gladly trade perfect knowledge about any one speaker for even a small amount knowledge about a large set of speakers from the population.

Despite this primarily philosophical definition, in practice the terms 'fixed' and 'random' effects have several inconsistent and sometimes contradictory definitions.

Luckily, when thinking about these concepts in terms of multilevel models we can be very specific: 'random' effects are those for which we estimate a standard deviation term. So, we treat differences in speaker average f0 ($\gamma_{uspeaker}$) as a variable, and estimate $\sigma_{speaker}$ for the distribution of variable.

Random effects, priors and pooling

In Chapter 1 I mentioned that every parameter in a Bayesian model needs a prior probability. If you look at our latest model definition (presented below), you will see four priors being specified: one for the Intercept, one for the effect for adult females, one for the error standard deviation (σ_{error}) and one for the expected amount of variation in speaker averages ($\sigma_{speaker}$).

$$y = Intercept + adult1 + \gamma_{uspeaker} + \varepsilon$$

Random Variables:

$$\gamma_{speaker} \sim \mathcal{N}(0, \sigma_{speaker})$$
$$\varepsilon \sim \mathcal{N}(0, \sigma_{error})$$

Priors:

$$Intercept \sim \sqcup(3, 225.5, 48)$$
$$adult1 \sim \sqcup(3, 225.5, 48)$$
$$\sigma_{error} \sim \sqcup(3, 0, 48)$$
$$\sigma_{speaker} \sim \sqcup(3, 0, 48)$$

Notably missing from the list of priors are any priors for the speaker-average parameters, $\gamma_{uspeaker}$. There should be one for each speaker in our sample. It turns out that the prior for these parameters is the $\sigma_{speaker}$ parameter, and this is estimated form the data! For this reason, the prior you set on $\sigma_{speaker}$ is sometimes called a 'hyperprior', because it is the prior for your prior!

Using $\sigma_{speaker}$ to determine the prior distribution of $\gamma_{uspeaker}$ actually makes perfect sense if you think about it. By definition, the speaker effects are centered around 0 (the average). The only question is, how widely are they distributed? Well, what better way to answer this question than using the distribution evident in the data itself?

By estimating the prior for some parameters from the data itself, multilevel models can help protect against problems that naturally arise when researchers compare many things. This is because in a Bayesian analysis, the prior influences the estimates of your individual parameters. As a result, the variation in the *other* parameters in your sample can influence any given parameter.

This process is sometimes referred to as 'partial pooling', since it refers to the partial pooling of information across subjects. Using partial pooling means the subject estimates are neither completely independent nor totally merged. They actually end up influencing each other in a logical manner. Broadly speaking, individual observations that deviate from 'typical' values of the population are maintained when there is good enough evidence for them. When there is wak evidence for them relative to the other observations in the sample, estimates may be brought closer to the group averages. As a result, partial pooling results in what is sometimes called 'shrinkage', because extreme values get 'shrunk' towards the overall mean.

In the context of a Bayesian multilevel model 'random effects' are those you estimate with partial pooling, where the prior is estimated from the data (e.g., $\sigma_{speaker}$). In contrast, 'fixed effects' are those for which you set arbitrary priors before fitting the model.

Although the terms terms 'fixed' and 'random' effects are useful (and I continue to use them to describe my models), it is important to keep in mind that the philosophical distinction between 'fixed' and 'random' variables is not relevant for the models we are discussing here. The real distinction is: do I want to fit every level totally independently as if there were all unrelated? Or do I want to use partial pooling in my estimates, thereby using all of the information present in my sample to protect against spurious findings?

In general, when you have many levels of a factor, it may be a good idea to include these as 'random' effects. There is not much downside to it, you get more information from the model (e.g., information about $\sigma_{predictor}$), and you can always fit multiple models to see what, if any differences, result form the different approaches.

Some useful things to consider are also: Do you believe that treating a predictor as a 'random' effect offers a modeling advantage? Does it better reflect the process/relationships you are trying to understand? Does it provide you with information we find useful? Is it realistic to fit this kind of model given the amount of data I have, and the nature of the relationships in the data? Right now the last point is not something we have talked about very much, but it is something we will need to worry about more as our models become more complicated.

Inspecting the random effects

The `brms` package has several functions to help understand our 'random' effects. There is a function called `ranef` which returns random effects estimates in the same way that `fixef` provides fixed effects estimates. The leftmost column of the output below represents the estimated effects associated with each speaker average value.

```
## I am telling it to give me the 'uspeaker' Intercepts, but only the first
## 10 rows. This is just so it doesn't take up the whole page.
brms::ranef (model_sum_coding)$uspeaker[1:10,,"Intercept"]
```

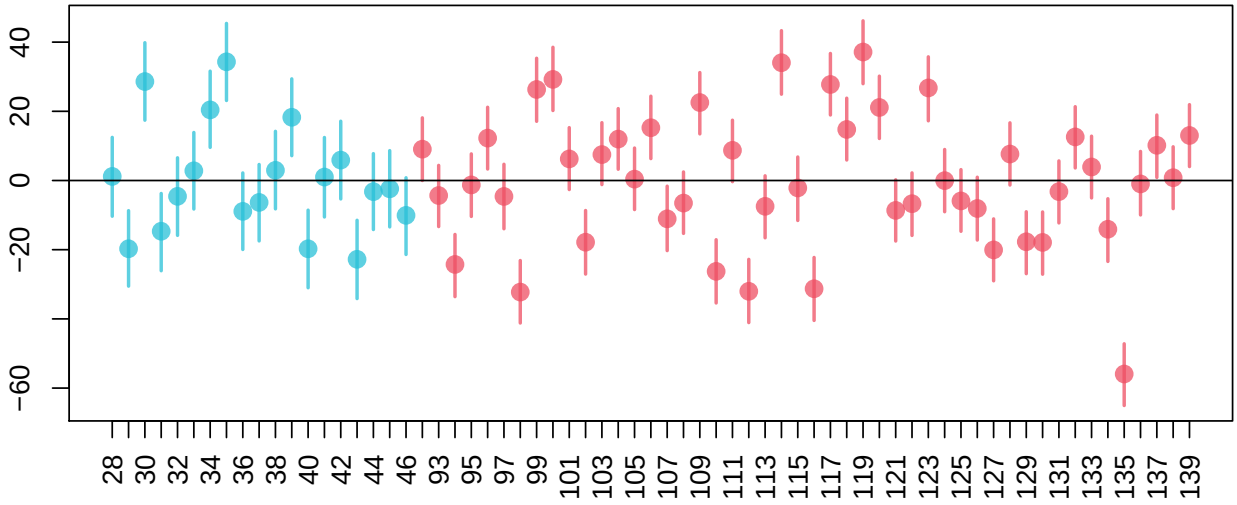
```
##      Estimate Est.Error       Q2.5     Q97.5
## 28   1.178409  5.790598 -10.331626 12.469034
## 29 -19.727367  5.642356 -30.539406 -8.716086
## 30  28.603411  5.701636  17.425326 39.832925
## 31 -14.676172  5.708269 -26.092560 -3.744481
## 32  -4.580821  5.671900 -15.861710  6.565227
## 33   2.798833  5.655844  -8.245980 13.886910
## 34  20.406371  5.604936   9.573044 31.603909
## 35  34.305413  5.788705  23.119722 45.409219
## 36  -8.921223  5.662577 -19.945362  2.183889
## 37  -6.332217  5.665746 -17.468755  4.633425
```

Notice that the speaker averages vary around 0, and some are even negative. That is because the speaker effects (and all random effects) are coded using sum-coding. So, what the speaker-specific mean terms tell us is: what is the average value for this speaker, relative to the overall average, and the group specific adjustment? Keep in mind, since our model encodes both the overall average *and* the group means, the speaker-effects encode differences to the group means and not to the overall means.

For example, we see that the mean for the second speaker above is -19.8 Hz. This means that they have a lower f0 than 'expected', and their mean f0 is 19.8 Hz lower than their group mean f0. In contrast, the first speaker has a speaker mean effect that is nearly zero (1.1). That tells you that this speaker's mean f0 was nearly the same as that of the average for their group

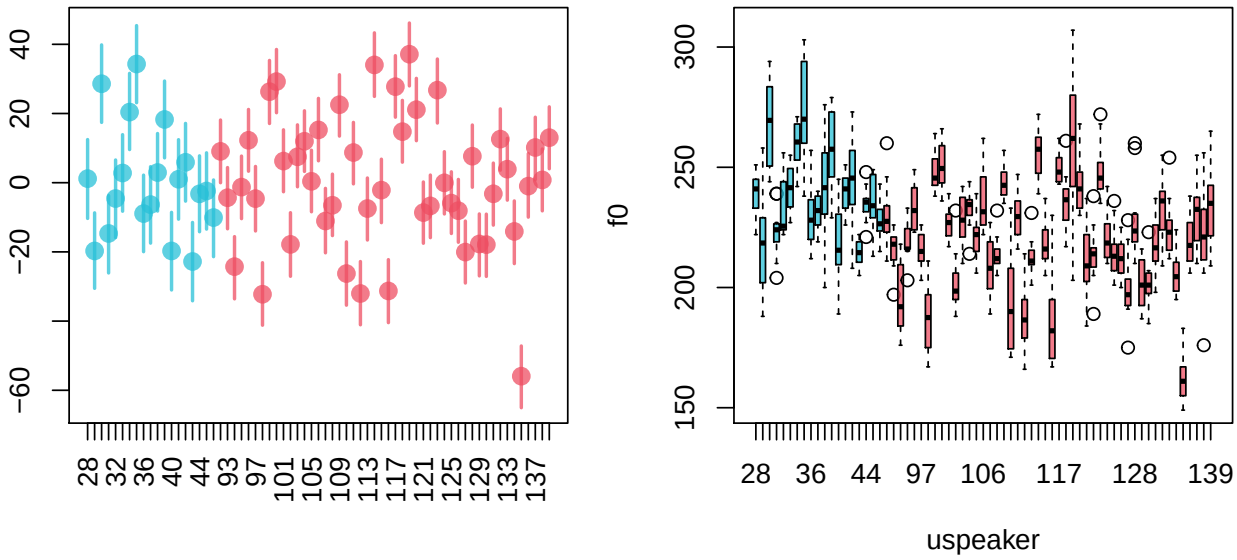
We can use a simple plotting function I wrote to look at the distribution of speaker effects terms. The function takes in the summaries made by brms and plots a point for each parameter mean, and lines indicating the credible intervals calculated by `brm` (usually 95% credible intervals).

```
par (mfrow = c(1,1), mar = c(4,4,1,1))
brmplot( brms::ranef (model_sum_coding)[["uspeaker"]][,,"Intercept"], col = colors)
abline (h=0)
```



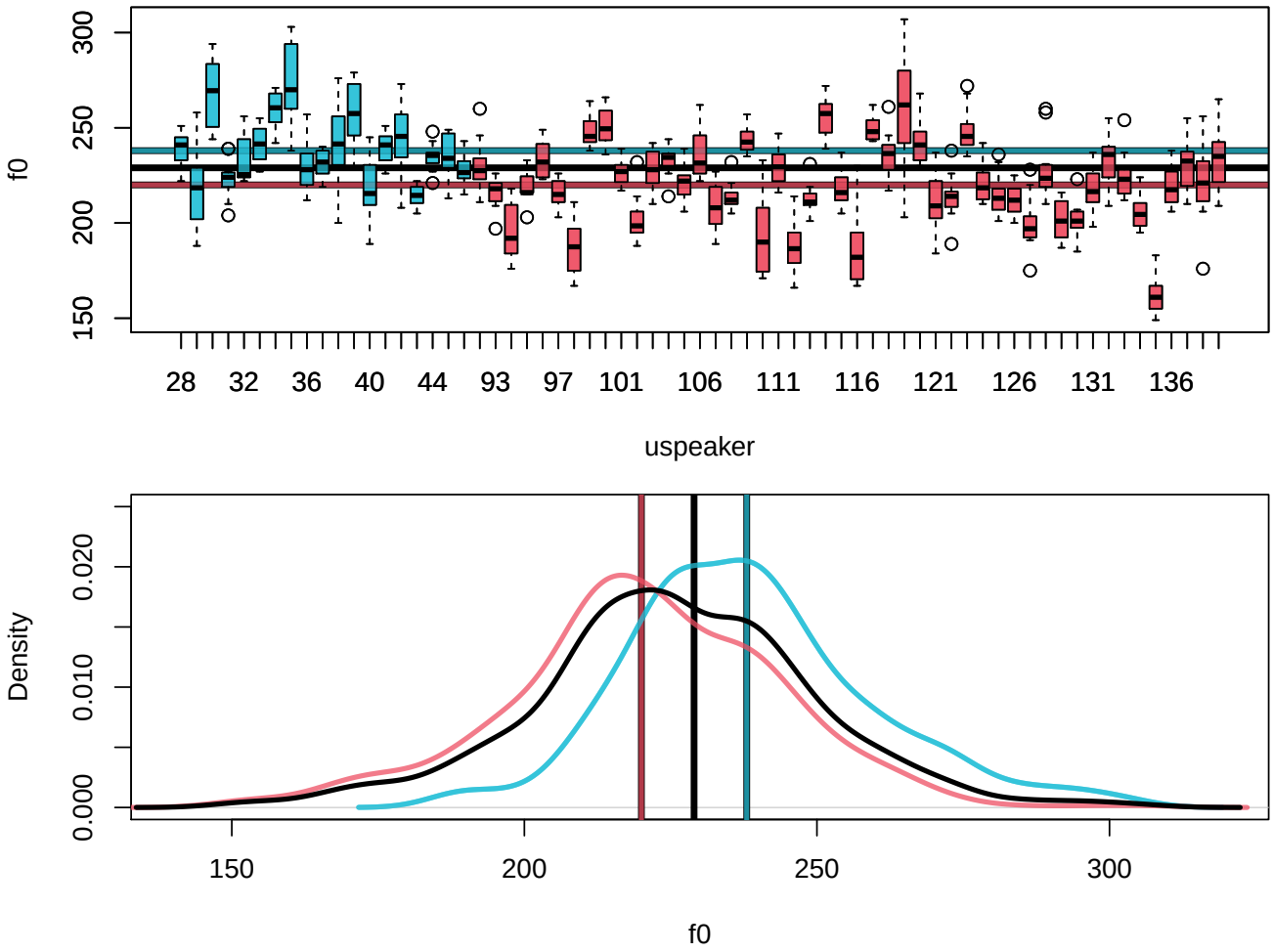
We can compare the estimates of by-speaker intercepts to the distribution of actual data arranged by subject. There is clearly a close correspondence!

```
## plot comparison of estimates of speaker means to actual data
par (mfrow = c(1,2), mar = c(4,4,1,1))
brmplot( brms::ranef (model_sum_coding)[["uspeaker"]][,,"Intercept"],
         col = colors)
boxplot (f0 ~ uspeaker, data=females, col = colors, ylim = c(150,310))
```



But what does it all mean?

Ok, so are the f0s produced by women and girls different? Consider the distribution of productions between and within speakers and groups, as shown in the figure below (lines represent the group means). An unbiased look at our results (and figures) so far suggests that:



- the magnitude of between speaker variation is **larger** than the difference between girls and women (19 Hz vs 18/17 Hz). This means that random people drawn from the two groups largely overlap.
- the magnitude of within-speaker variation (12 Hz) is almost as large as the group difference and the between-speaker difference! This means that two random productions from two different people might be the same, even when the speakers average f0s are quite different.

And yet:

- the mean f0 **is** reliably different between women and girls, and there are well-known anatomical reasons for this that are expected a priori (i.e., adult females are larger than younger women, larger speakers often produce lower f0s).
- the between-speaker differences are 'random' from person to person, but systematic for a given person.
- even the within-speaker variation may be systematic given a more-complicated model.

So are they different yes or no? Statistics aside, a fair assessment of our data suggests that neither binary conclusion is fully supported: they are distinguishable but overlap substantially. So, our results can basically mean whatever we want them to mean, as long as reviewers (and readers in general) will believe us. The meaning is as much in our heads as it is in the model. If you want to use this model to highlight the differences between girls and women, I think it is valid. I also think it would be valid to use this data to talk about between and within-speaker variation, highlighting the overlap that exists between speakers. Both are true! The model is simply a reflection of the relationships in our data, and the interpretation is up to us.

This may sound like I am advocating that people should feel free to draw any conclusions given their data, but keep in mind that the "as long as reviewers (and readers in general) will believe us" component is crucial. The results of the model will need to be interpreted in the larger context of the work it is presented in, and in terms of scientific and general knowledge that readers have. The results of any model will need to 'make sense' given this, and a statistical result on its own will not be enough to make people (including us) believe outlandish, or even weakly supported claims.

The model is not reality and should not be confused with reality. This is a very important point! A statistical finding does not *prove* that something is *true*. This kind of thinking has caused many problems for researchers in the social sciences recently. In general, we can imagine that 10 people might approach any given research question in 10 different ways, a concept known as researcher degrees of freedom. This would cause slight differences in their results, resulting in a sort of 'distribution' for any given result. How can a fixed underlying reality result in a distribution or results? When they are all slightly wrong!

For example, we know for a fact that f0 varies, weakly but systematically, across vowel categories, a concept known as (intrinsic f0)[https://www.sciencedirect.com/science/article/abs/pii/S0095447095801650]. A model that included vowel category as a within-speaker predictor would reduce the apparent error in our model (making $sigma_{error}$ smaller), and might affect the precision of our other estimates. Would this new model invalidate our current model? Answering yes to this question is generally problematic because there is *always* a better model out there, and so every model is automatically invalid.

The solution is to think of your model not as a mathematical implementation of *reality* but instead as a mathematical implementation of your research questions. Your model should include the information and structure that you think are necessary to represent and investigate your questions.

Using a different model can result in different results given the same data, but asking a different question can also lead to different results given then same data! One of my favorite phrases to use is "given our data and model structure". This phrase is helpful because it highlights the fact that your results are contingent on:

1) the data you collected. Given other data you may have come to other conclusions.

2) the model you chose. Given another model you may have come to other conclusions.