

# R Lab #3b - Making better graphs

## Contents

<b>Default graphs are not necessarily the best graphs</b>	<b>1</b>
<b>Graphing human hemoglobin data</b>	<b>1</b>
Strip chart . . . . .	1
Changing point shape . . . . .	2
Changing point color . . . . .	4
Changing spacing between groups . . . . .	5
Changing axis labels . . . . .	6
Changing graph “theme” . . . . .	7
<b>Saving a graph</b>	<b>8</b>
Save with “Export” button . . . . .	8
Save with ggsave() function . . . . .	8
<b>Other plot types for human hemoglobin data</b>	<b>9</b>
Box plot . . . . .	9
Violin plot . . . . .	10
<b>Combining graph types</b>	<b>14</b>
<b>Annotating a graph with text</b>	<b>16</b>
<b>Summary</b>	<b>18</b>
<b>R commands summary</b>	<b>18</b>

## Default graphs are not necessarily the best graphs

The default graphs that ggplot2 may not be the best format for effectively presenting your data. In fact, it usually isn’t, and this is true for most programs. Fortunately, just about any aspect of a graph can be customized in R.

## Graphing human hemoglobin data

### Strip chart

Chapter 2 presents a strip chart and a violine plot of samples of hemoglobin concentrations of male humans from three populations living in high altitude (Andes, Ethiopia, Tibet) and one lowland population (USA) (Figure 2.3-4).

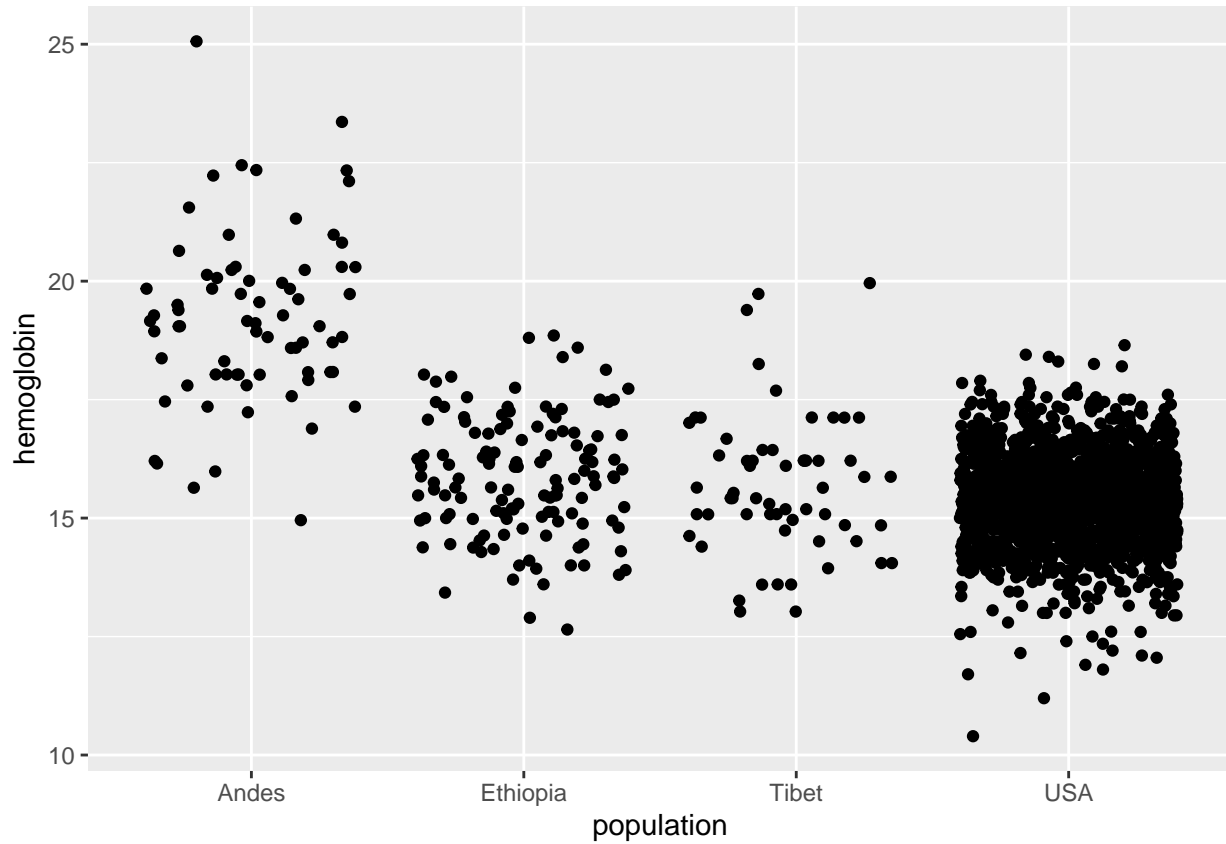
Let’s practice customizing figures with this data set.

First, download the data file chap02e3bHumanHemoglobinElevation.csv from Canvas (Modules > Data sets for examples/questions in textbook > Chapter 2 data sets) and save it to your R working directory. Once complete, import the data into R and load the ggplot2 library:

```
hemData <- read.csv("chap02e3bHumanHemoglobinElevation.csv",
  stringsAsFactors = T)
library(ggplot2)
```

Now make the default strip chart with `geom_jitter()`.

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_jitter()
```



### Changing point shape

First let's learn how to change the points to open circles like the book figure. This is better in this case because it is a bit easier to see overlapping points. This is done by adding the argument `pch` to the graphing command. R has a bunch of different symbol options that are coded with numbers and designated with `pch`. Here is a `pch` cheat sheet:

**0**  


**1**  


**2**  



**3**  


**4**  


**5**  


**6**  


**7**  


**8**  


**9**  


**10**  


**11**  


**12**  


**13**  


**14**  


**15**  


**16**  


**17**  


**18**  


**19**  


**20**  


**21**  


**22**  

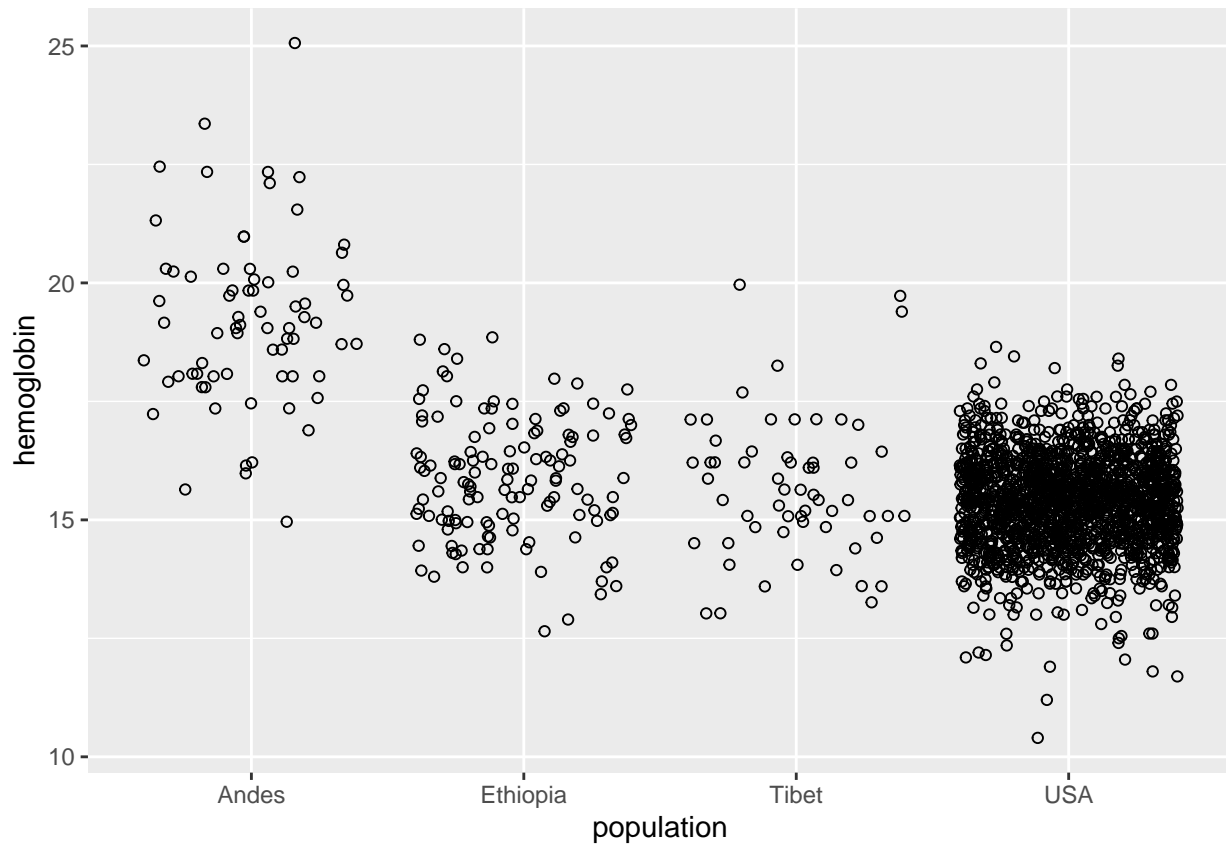

**23**  


**24**  


**25**  


So for an open circle we want to add `pch=1` to `geom_jitter`:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_jitter(pch = 1)
```

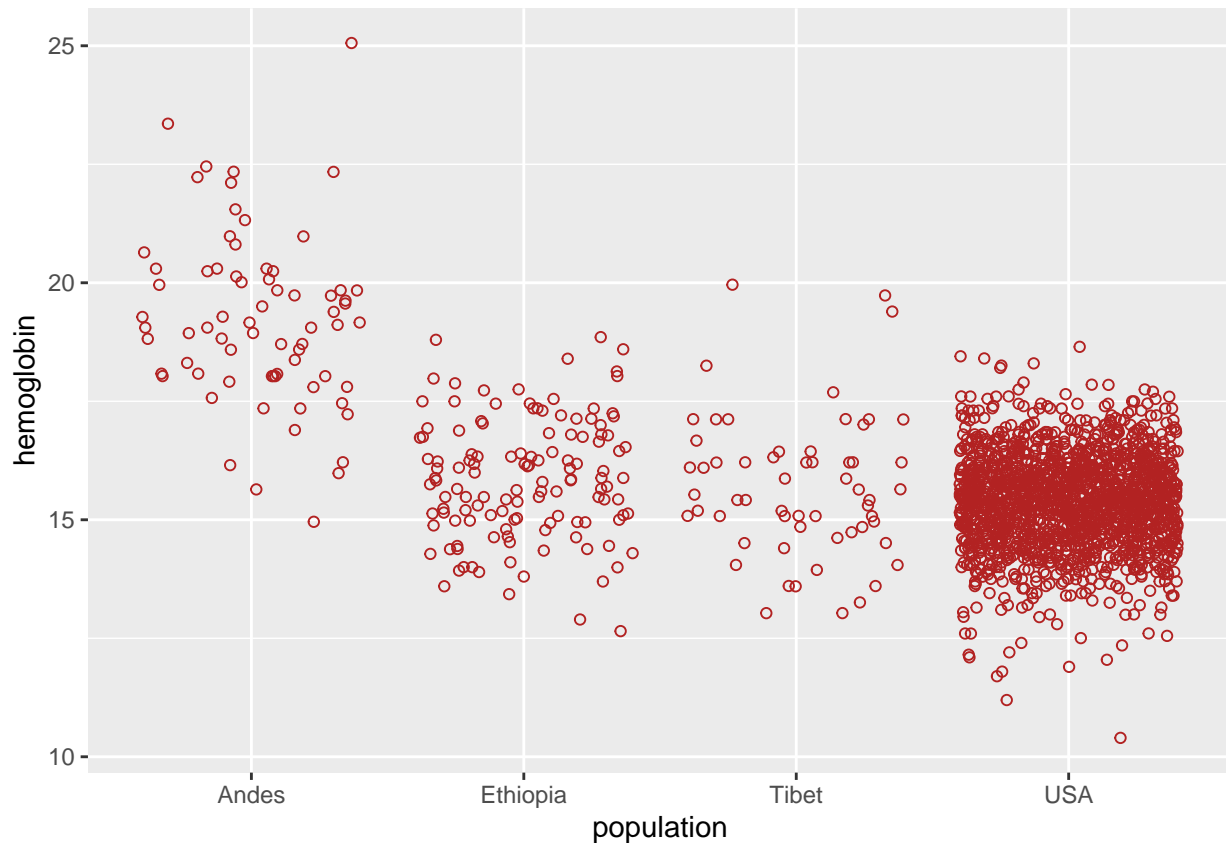


### Changing point color

Next let's change the color of the point. This is done by adding the argument `color` to the graphing command. R has a wide variety of colors. See this one guide on the web: <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

I believe that the shade of red that they use is "firebrick" and in the syntax it needs to be in quotes:

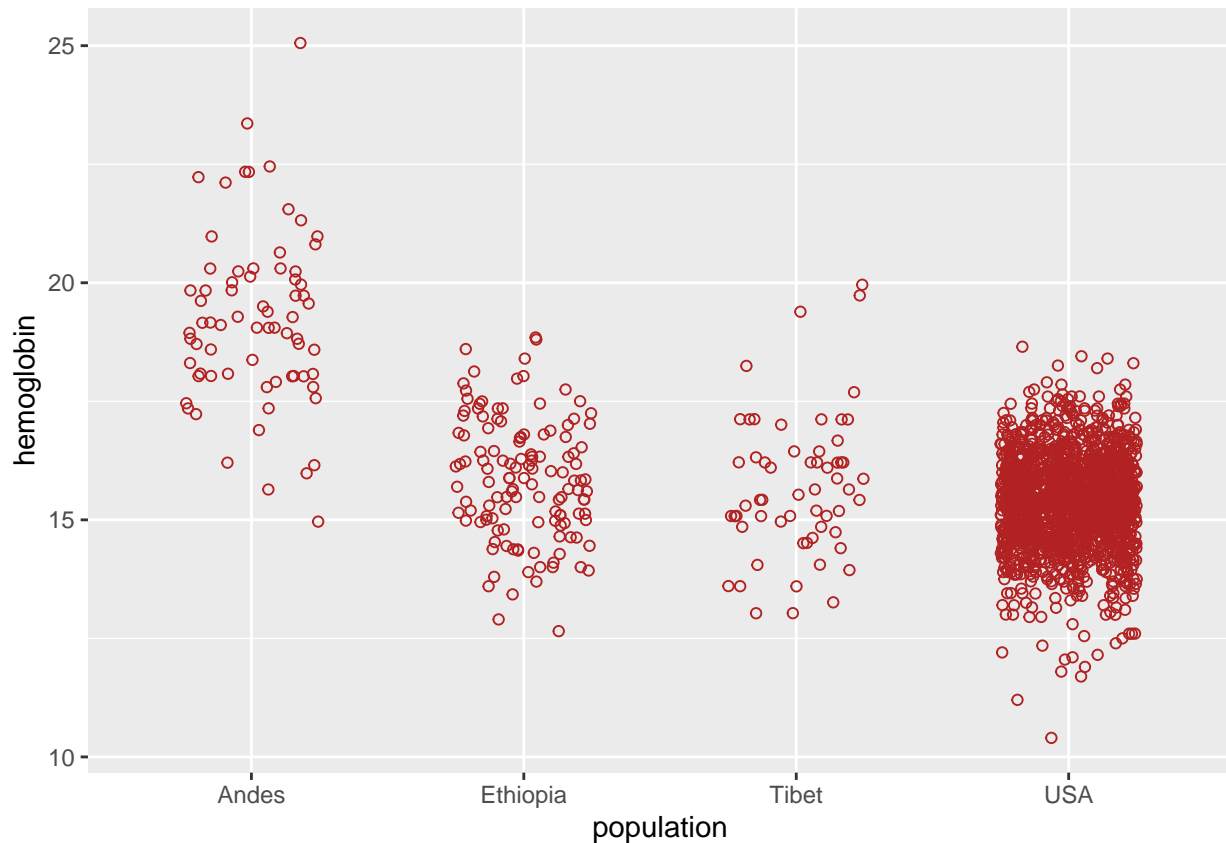
```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_jitter(pch = 1,  
  color = "firebrick")
```



### Changing spacing between groups

Next thing I notice is that there is less space between groups than the plot in the book. Too little space makes it harder to differentiate groups, and too little space causes too much overlap in points. This can be tweaked by adding the argument `width` to the graphing command. The default width is 0.4 (spread 40% of distance each way). Let's decrease the value to 0.25:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_jitter(pch = 1,  
  color = "firebrick", width = 0.25)
```



### Changing axis labels

Note that the axis labels are the variable names in the data file. It's good to use simple, meaningful names in data files, but the graph typically needs more detail. In particular, whenever you have a numerical variable you want to add the units. Axis labels can be modified simply by adding `xlab()` and `ylab()` commands to the line:

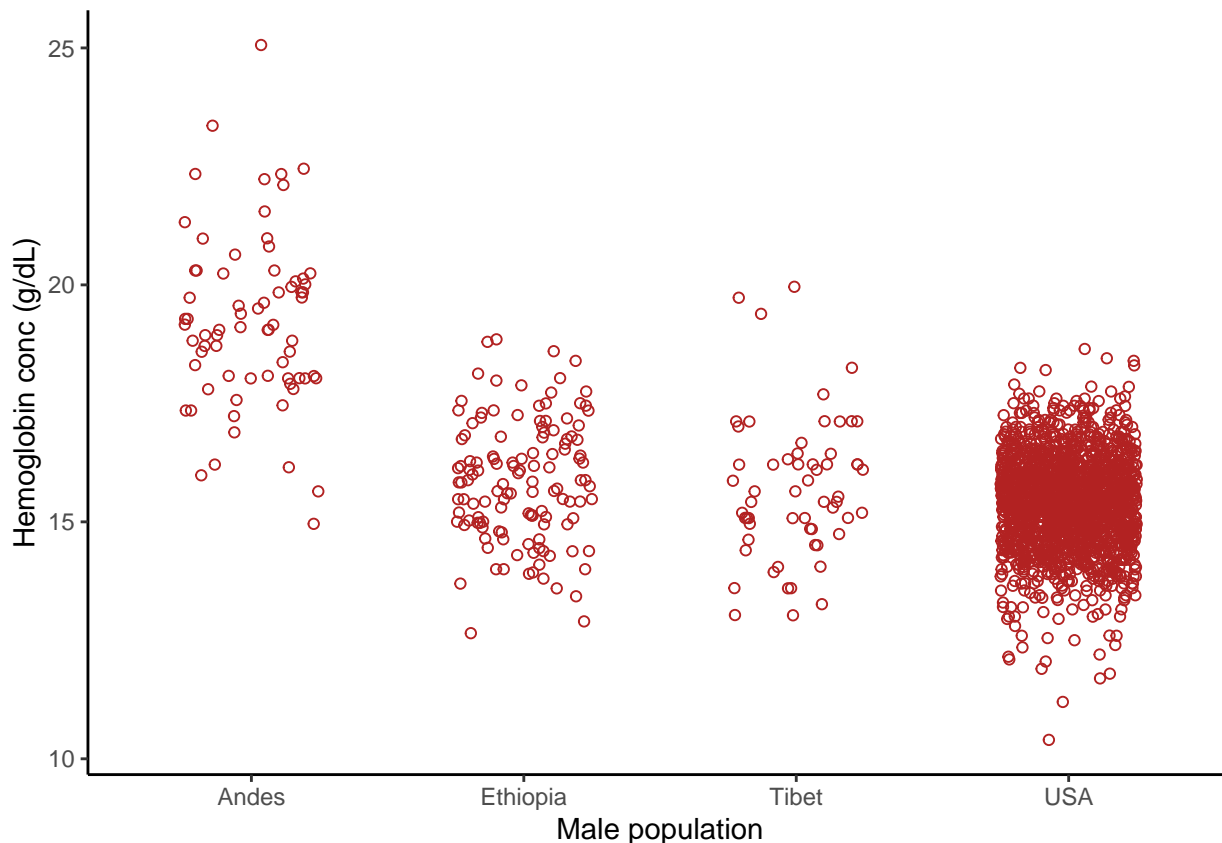
```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_jitter(pch = 1,
  color = "firebrick", width = 0.25) + xlab("Male population") +
  ylab("Hemoglobin conc (g/dL)")
```



### Changing graph “theme”

Now let’s make the format more simple by adding the `theme_classic()` command. This changes the background to white, removes the grid lines, and generates a cleaner aesthetic similar to figures shown in the textbook.

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_jitter(pch = 1,
  color = "firebrick", width = 0.25) + xlab("Male population") +
  ylab("Hemoglobin conc (g/dL)") + theme_classic()
```



## Saving a graph

Now we have a nice graph! If you wrote a script then you have the code used to generate the graph saved, but you probably want to save the graph yourself. There are multiple options available for saving a graph in R.

### Save with “Export” button

Notice that above your plot in RStudio there is a button labeled “Export” with a down arrow. Clicking here gives the options “Save as Image...”, “Save as PDF...”, and “Copy to Clipboard.”

Both “Save as PDF” and “Save as Image” (allows a variety of formats) open pop-up windows. There you can set the file name, change the location (working directory by default), and change the height & width. Note that the height and width shown is the size of the plot in RStudio. So it’s easiest to close the pop-up, re-size the plot window in RStudio to exactly how you want it to look, and then click Export again.

### Save with `ggsave()` function

It is also possible to save the plot with a command, which gives you a record of exactly what you did and provides more options. One function used for this is `ggsave()`, which will save the last graph made by default. The function can take several possible arguments, including:

filename: Name of file device: Type of file. Options include “png”, “pdf”, and “jpeg”. path: Path to folder for file (working directory by default) dpi: Resolution (dots per inch)

Again, by default the graph size will match what you have in the RStudio window, but you can change it with the following arguments.



width: Plot width height: Plot height units: Units for height and width. Options include “in”, “cm”, “mm”, and “px”.

So to simply save the file to the working directory with the current size in a pdf format you would run:

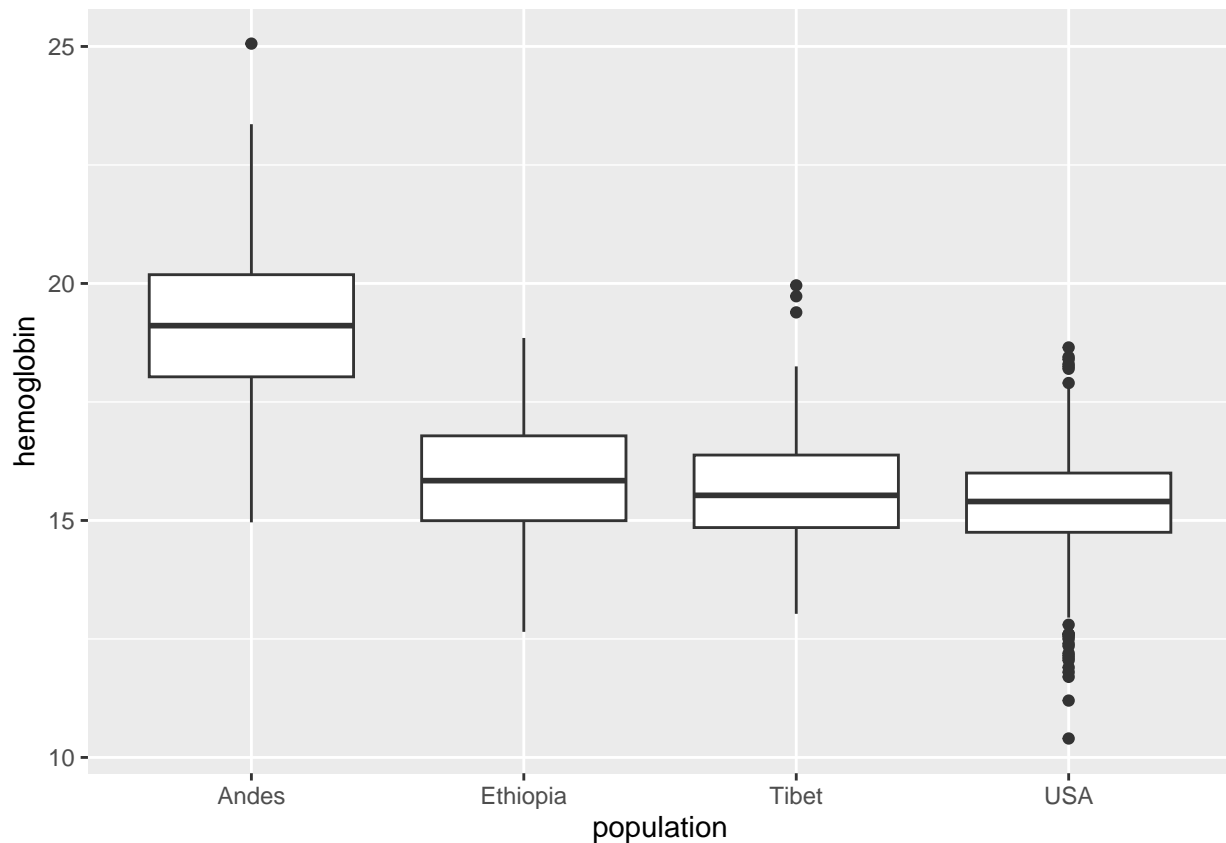
```
ggsave(filename = "HumanHemoglobin.pdf", device = "pdf")
```

## Other plot types for human hemoglobin data

### Box plot

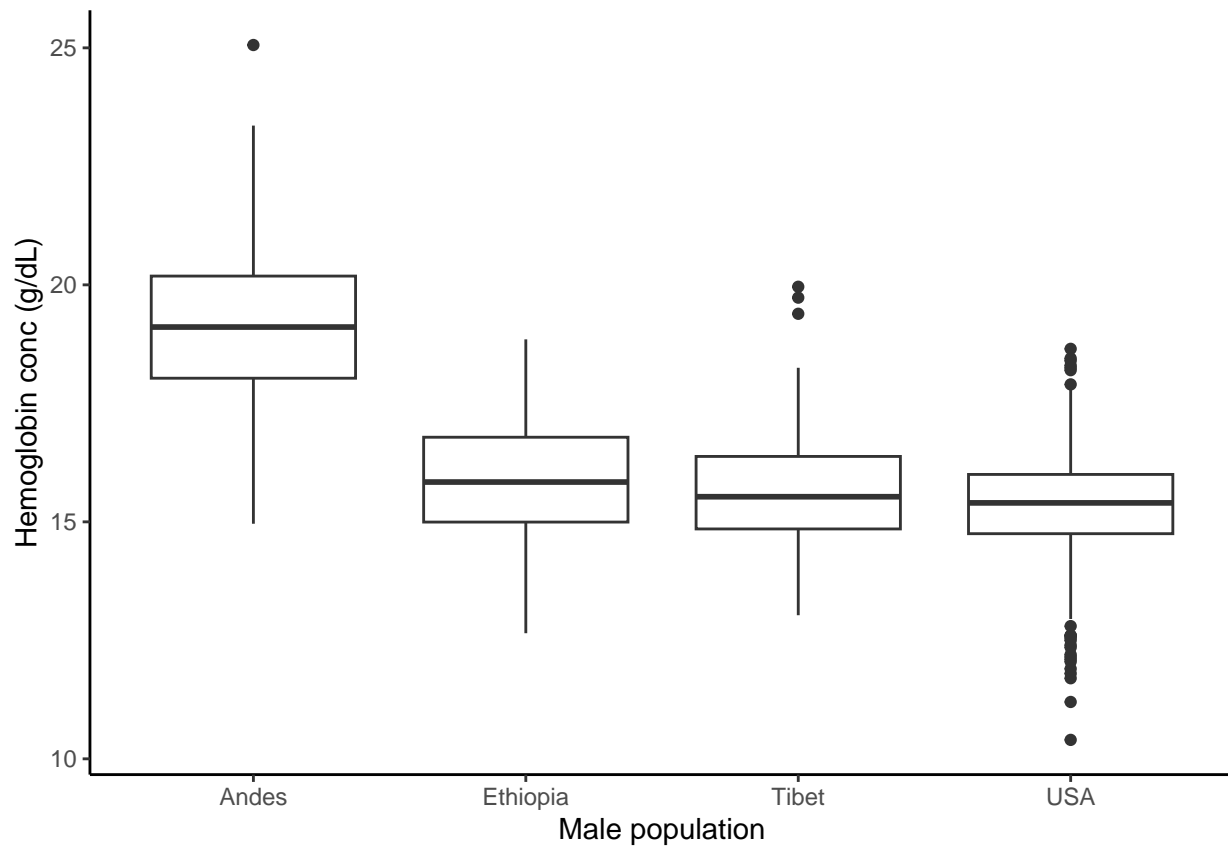
Now let's make a box plot of the same data. Note that box plots are described in chapter 3. First the default plot with `geom_boxplot()`:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_boxplot()
```



Now fix the axis labels with `xlab()` and `ylab()`, scale of the y-axis with `ylim()`, and simplify the look with `theme_classic()`:

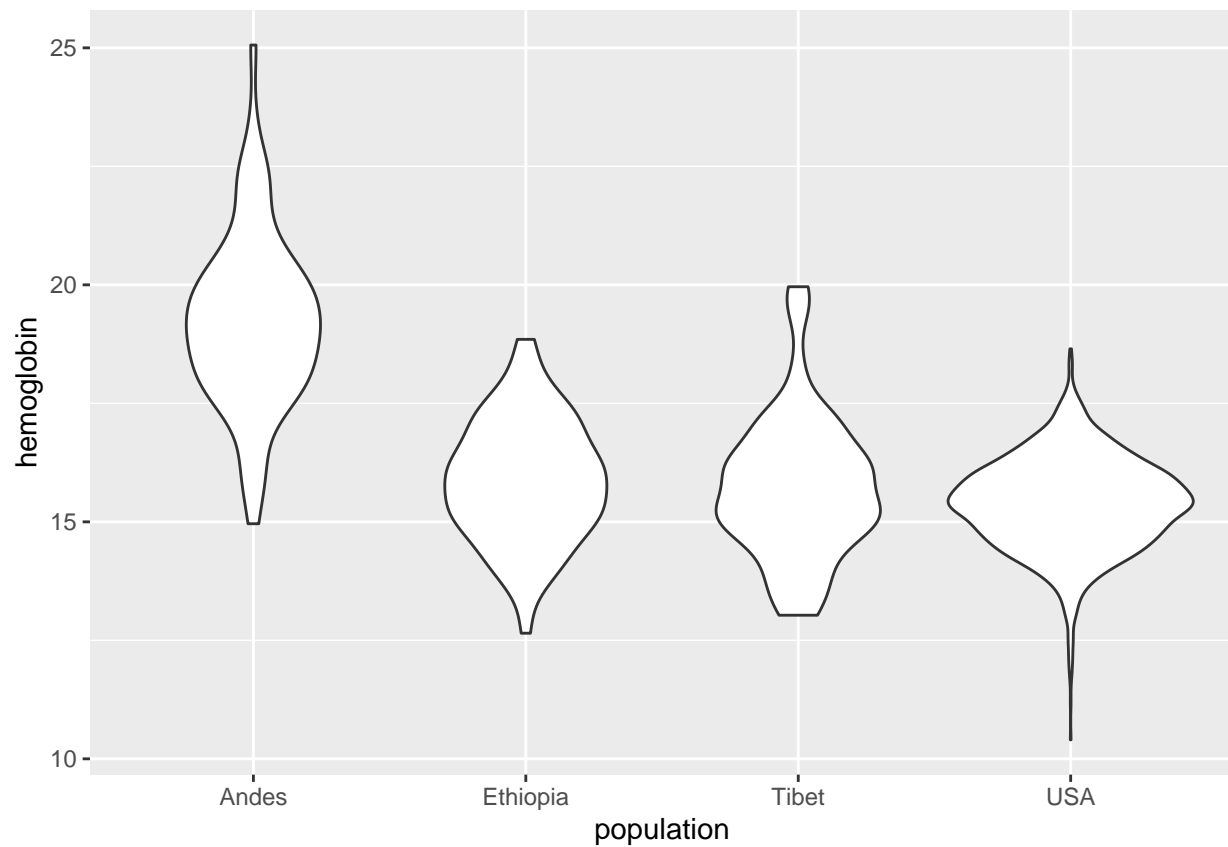
```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_boxplot() +  
  xlab("Male population") + ylab("Hemoglobin conc (g/dL)") +  
  theme_classic()
```



## Violin plot

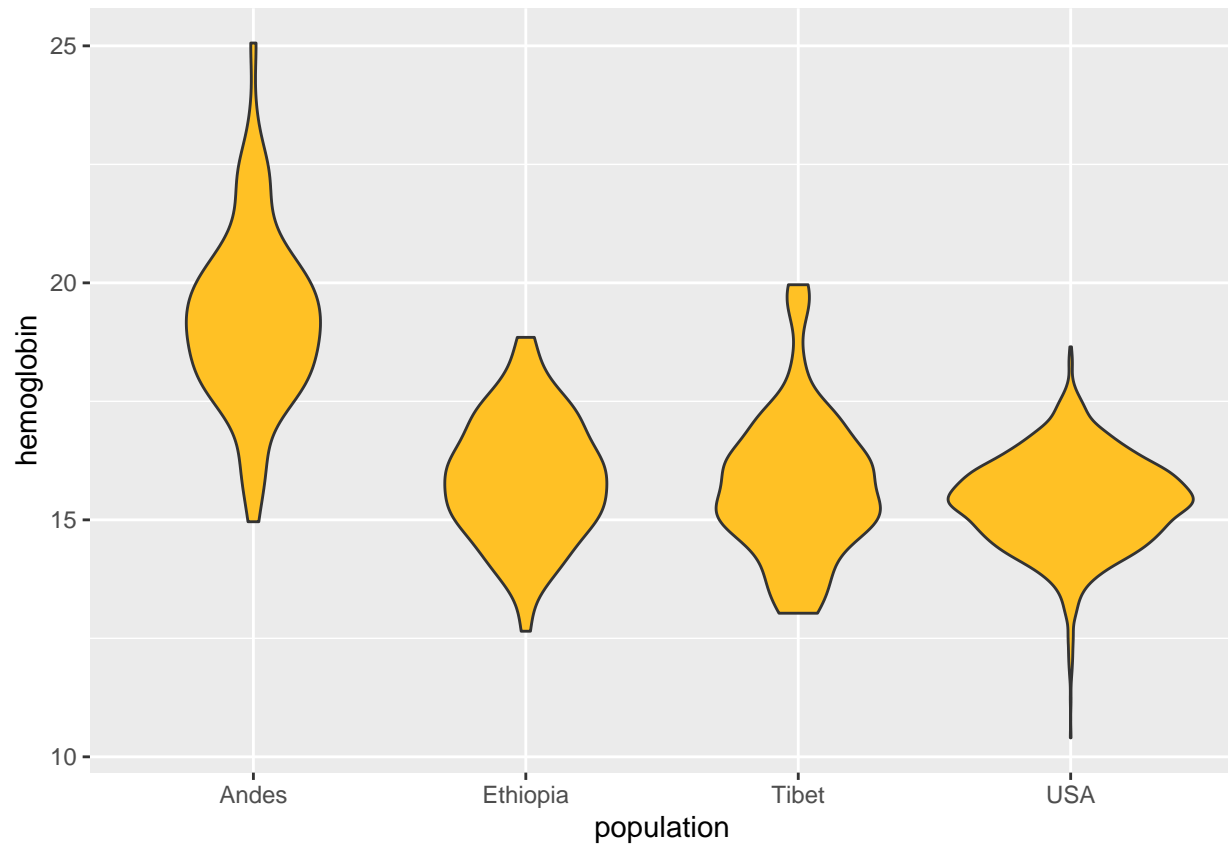
Next let's make a violin plot with `geom_violin()`. First the default:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin()
```



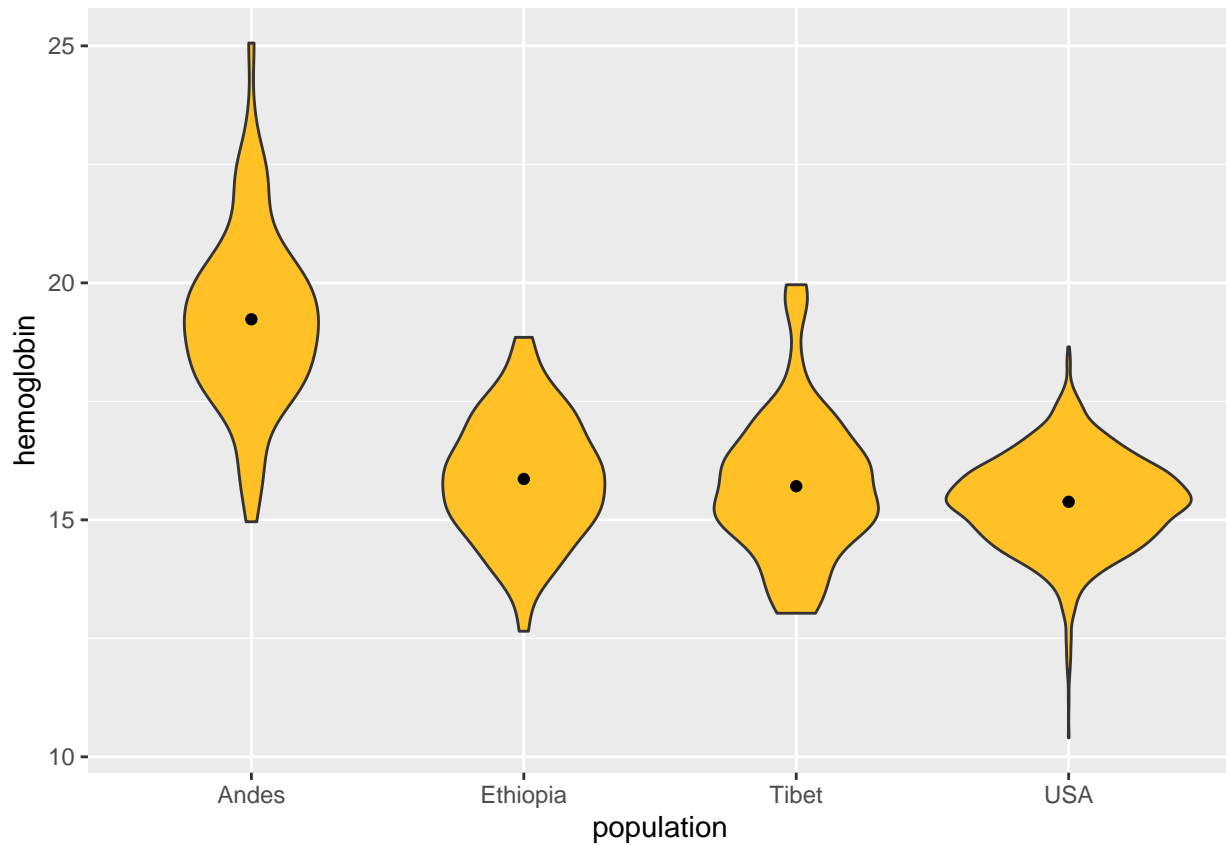
Notice that in Figure 2.3-4 that the polygons are filled with a yellow color (which is “goldenrod1”). This can be done by adding the argument `fill` to the `geom_violin()`:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1")
```



The black point in the Figure 2.3-4 is the mean estimate for each group. This can be added with the command `stat_summary()`. Here is the syntax:

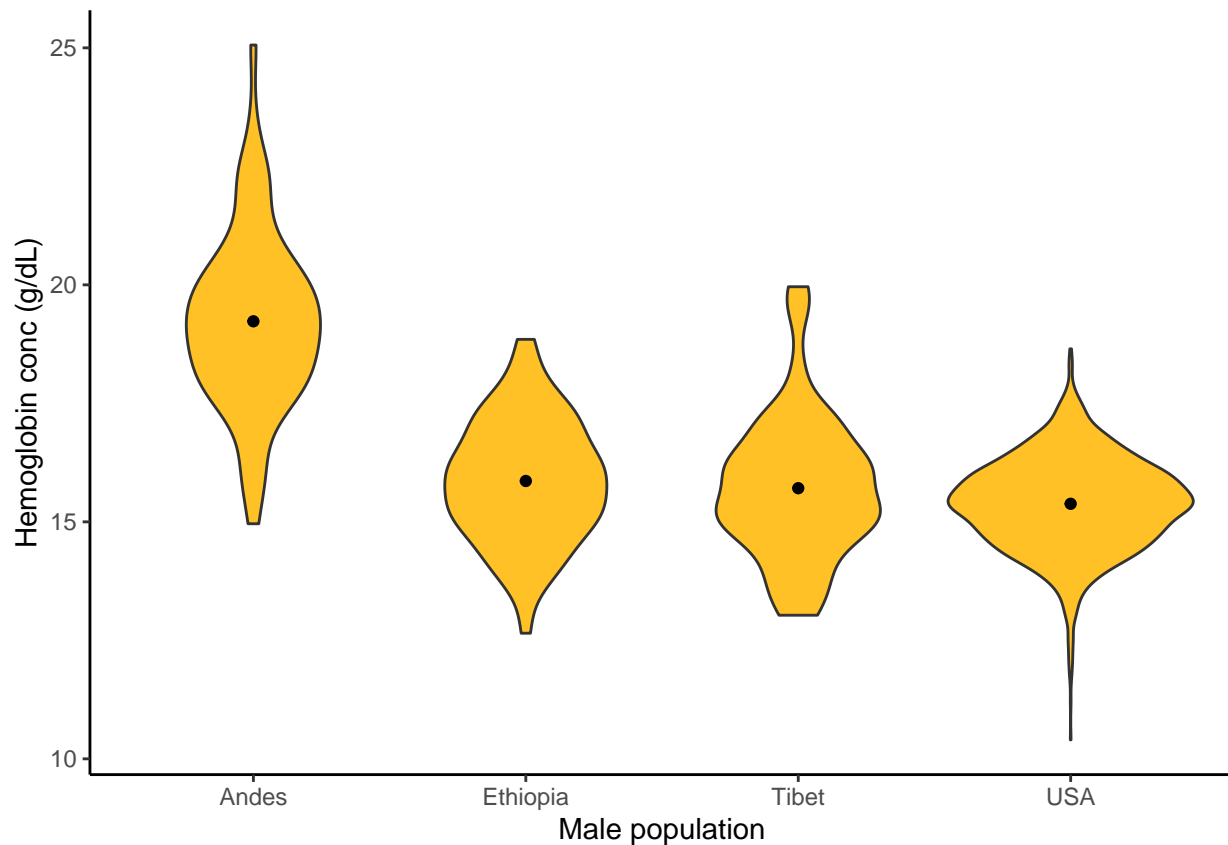
```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1") +  
  stat_summary(fun = mean, geom = "point")
```



In `stat_summary()`, `fun` means function and tells R to calculate the mean, and `geom` means geometry and tells R to add the stat as a point.

Now improve the axis labels/limits and simplify the look:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1") +  
  stat_summary(fun = mean, geom = "point") + xlab("Male population") +  
  ylab("Hemoglobin conc (g/dL)") + theme_classic()
```

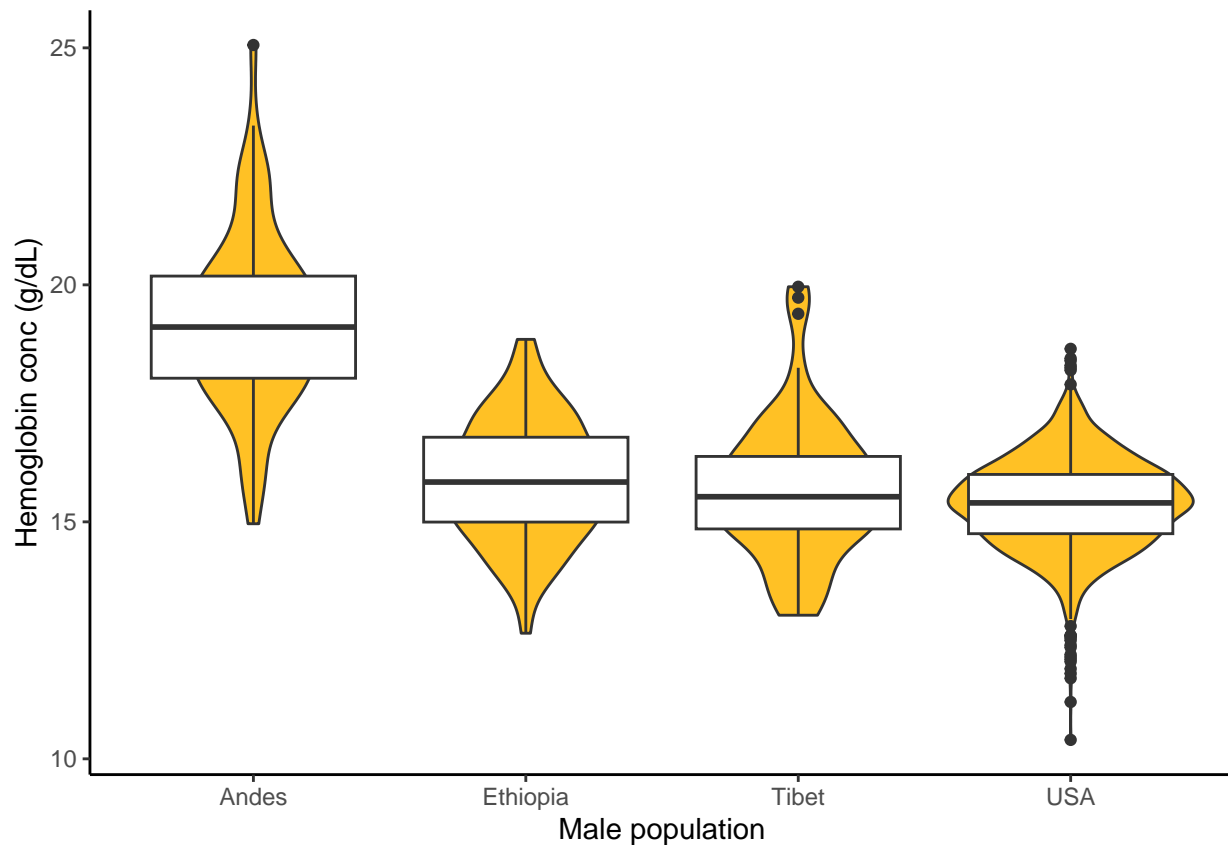


## Combining graph types

I've seen an increasing amount of figures in journal articles that show a box plot overlaid on a violin plot. This allows the reader to see the variation in groups with two different styles in the same plot.

You can combine two (or more) plot types in a single ggplot command. The first plot type in the command will be on the "bottom" and the second type will be overlaid on the "top". So this command makes a graph with a box plot layered on top of a violin plot.

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1") +  
  xlab("Male population") + ylab("Hemoglobin conc (g/dL)") +  
  theme_classic() + geom_boxplot()
```



This doesn't look that great because the box plot covers a lot of the violin plot, particularly because of the wide size and white fill. This can be changed with width and matching the color of the fill, respectively:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1") +
  xlab("Male population") + ylab("Hemoglobin conc (g/dL)") +
  theme_classic() + geom_boxplot(width = 0.25, fill = "goldenrod1")
```



## Annotating a graph with text

Sometimes you might want to add text to a graph to show the reader additional information. For example, you might want to show the sample size for each group above the respective plot. First we need the sample sizes. There are multiple ways to do this. One is to do this with the command `summary()`.

```
summary(hemData)
##           id           hemoglobin           population
## Andean.male1 :    1   Min.   :10.40   Andes      : 71
## Andean.male10:    1  1st Qu.:14.80   Ethiopia: 128
## Andean.male11:    1  Median :15.45   Tibet     : 59
## Andean.male12:    1   Mean   :15.56   USA      :1704
## Andean.male13:    1  3rd Qu.:16.15
## Andean.male14:    1   Max.    :25.06
## (Other)         :1956
```

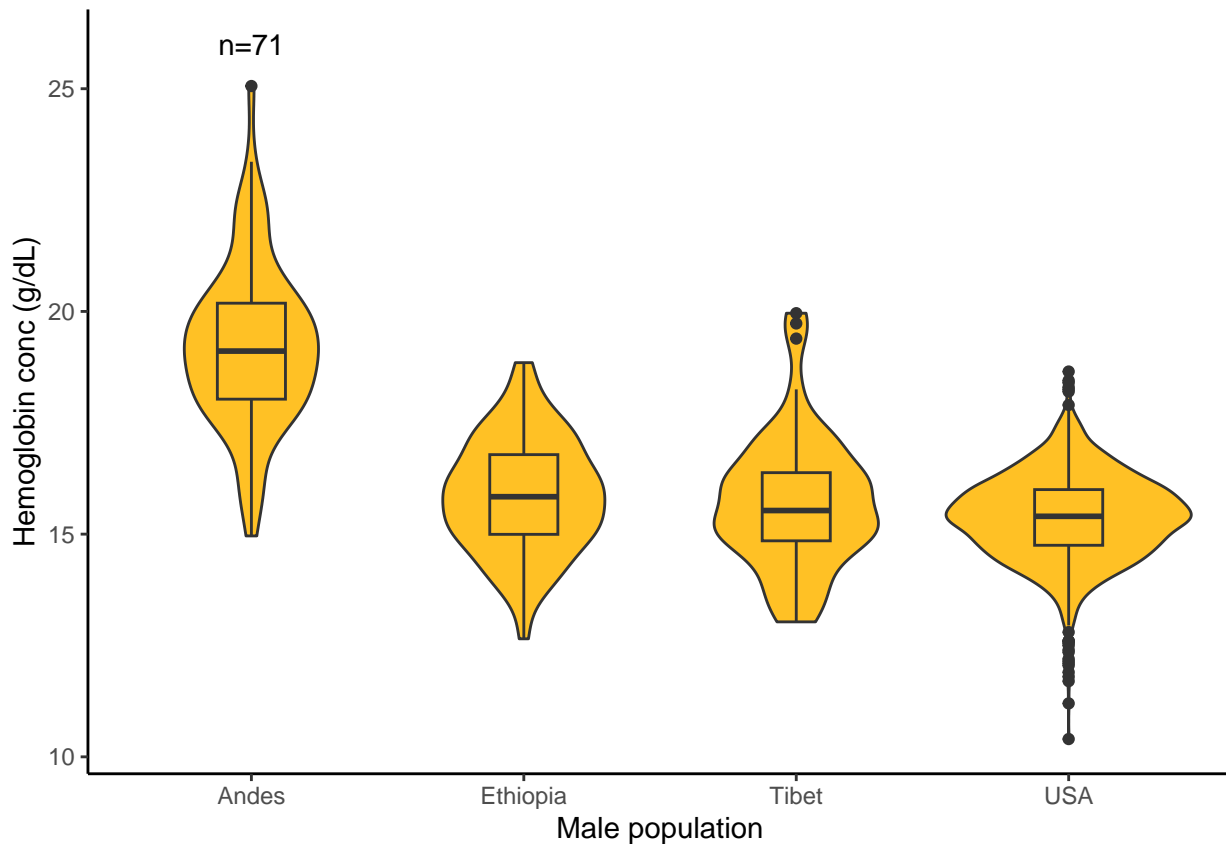
Provided that you used `stringsAsFactors = T` when reading in your csv file, this should show the sample size for each group. In R terminology, population is treated as a “factor” (categorical variable) with different “levels” (groups).

To add text to a plot you add the `annotate()` function to the command. For arguments, you add the geometry of annotation to make (`geom="text"`), the x/y coordinates, and the text (label). Note that for the x-axis the groups are at positions 1, 2, 3, etc. From the summary command above, we can see that the highest hemoglobin value is 25.06. So let's put our annotations at `y=26`. For the text/label, we'll use `n=sample size` (from summary output).

Here is how we add the sample size of 71 above the Andes part of the plot:

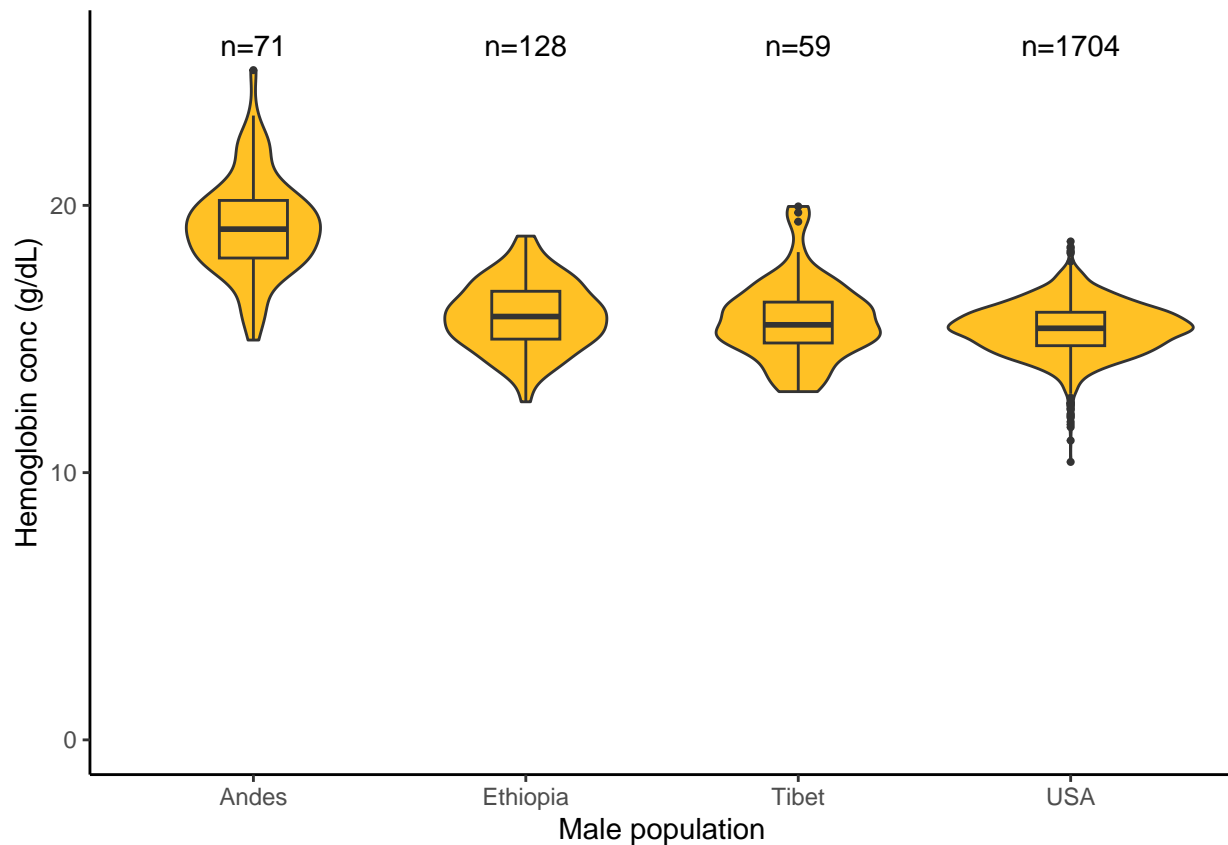


```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1") +
  xlab("Male population") + ylab("Hemoglobin conc (g/dL)") +
  theme_classic() + geom_boxplot(width = 0.25, fill = "goldenrod1") +
  annotate(geom = "text", x = 1, y = 26, label = "n=71")
```



Now we can add the other three with three additional `annotate()` commands:

```
ggplot(hemData, aes(x = population, y = hemoglobin)) + geom_violin(fill = "goldenrod1") +
  stat_summary(fun = mean, geom = "point") + xlab("Male population") +
  ylab("Hemoglobin conc (g/dL)") + theme_classic() + ylim(0,
26) + stat_summary(fun.data = mean_se, geom = "errorbar",
fun.args = list(mult = 2), width = 0.25) + geom_boxplot(width = 0.25,
fill = "goldenrod1", outlier.size = 0.75) + annotate(geom = "text",
x = 1, y = 26, label = "n=71") + annotate(geom = "text",
x = 2, y = 26, label = "n=128") + annotate(geom = "text",
x = 3, y = 26, label = "n=59") + annotate(geom = "text",
x = 4, y = 26, label = "n=1704")
```



## Summary

Here we learned various functions for changing and improving graphs with ggplot2. These are just a small percentage of the world of possible options. Basically, anything that you could want to tweak in a graph can probably be done.

While some of the options covered here are not necessary for our quizzes and exams, there are a few things that I expect. Namely:

- Plot type is appropriate for the data
- Axis labels are readable (not always the case with variable names) and include units as appropriate
- Use `theme_classic()` so we are all making plots with a similar aesthetic

## R commands summary

- **Jittered strip plot**
  - `geom_jitter()`
- **Point shape**
  - `pch()`
- **Point color**
  - `col()`
- **Space between groups**
  - `width()`
- **Axis labels**
  - `xlab()`
  - `ylab()`

- **“Classic” theme aesthetic**
  - `theme_classic()`
- **Save plot**
  - `ggsave()`
- **Box plot**
  - `geom_boxplot()`
- **Violin plot**
  - `geom_violin()`
- **Fill color in box or violin plot**
  - `fill`
- **Text annotation**
  - `annotate()`

There are LOTS of additional options for customizing a graph with ggplot2. For a full reference see this site:  
<https://ggplot2.tidyverse.org/reference/>