

# Chapter 2 R ggplot2 Examples

Bret Larget

February 5, 2014

## Abstract

This document introduces many examples of R code using the `ggplot2` library to accompany Chapter 2 of the Lock 5 textbook. The primary data set used is from the student survey of this course, but some plots are shown that use textbook data sets.

## 1 Getting Started

### 1.1 Installing R, the Lock5Data package, and ggplot2

Install R onto your computer from the CRAN website ([cran.r-project.org](http://cran.r-project.org)). CRAN is a repository for all things R. Follow links for your appropriate operating system and install in the normal way.

After installing R, download the `Lock5Data` and `ggplot2` packages. Do this by starting R and then using `install.packages()`. You will need to select a mirror site from which to download these packages. The code will automatically install the packages in the appropriate place on your computer. You only need to do these steps once.

```
install.packages("Lock5Data")
install.packages("ggplot2")
```

### 1.2 Loading Data into an R Session

To load the `SleepStudy` data set from the textbook, first load the `Lock5Data` library and then use `data()` to load the specific data set.

```
library(Lock5Data)
data(SleepStudy)
```

To load data from files, use either `read.table()` for files where variables are separated by space or `read.csv()` where variables are separated by commas. The former requires a second command to indicate that the file contains a header row. The latter assumes a header row by default.

```
heart = read.table("heart-rate", header = TRUE)
students = read.csv("students.csv")
```

The preferred procedure to avoid typing in long expressions for a path to where the file is is to change the working directory of R to a folder where the data is (or to move the data to where the R session is running). An alternative is to use `file.choose()` in place of the file name. This will open up a window in your operating system you can use to navigate to the desired file.

```
heart = read.table(file.choose(), header = TRUE)
```

Each of the loaded data sets is an object in R called a *data frame*, which is like a matrix where each row is a case and each column is a variable. Unlike matrices, however, columns can contain *categorical variables*, where the values are labels (typically words or *strings*) and are not numerical. In R a categorical variable is called a *factor* and its possible values are *levels*.

### 1.3 Using the Loaded Data

There are a number of useful things you can do to examine a loaded data set to verify that it loaded correctly and to find useful things like the names and types of variables and the size of the data set.

The `str()` function shows the structure of an object. For a data frame, it gives the number of cases and variables, the name and type of each variable, and the first several values of each. The dimensions are returned by `dim()`. The number of cases (rows) and variables (columns) can be found with `nrow()` and `ncol()`.

```
str(students)

## 'data.frame': 48 obs. of 13 variables:
## $ Sex      : Factor w/ 2 levels "female","male": 1 2 2 2 2 1 2 2 1 1 ...
## $ Major    : Factor w/ 12 levels "actuarial science",..: 1 11 11 9 11 11 5 10 12 12 ...
## $ Major2   : Factor w/ 11 levels "", "business",..: 1 4 8 11 1 1 11 7 1 1 ...
## $ Major3   : Factor w/ 4 levels "", "african studies",..: 1 1 3 1 1 1 1 2 1 1 ...
## $ Level    : Factor w/ 6 levels "freshman","graduate",..: 1 3 1 4 5 2 5 4 1 1 ...
## $ Brothers : int 0 3 0 0 0 0 1 1 0 2 ...
## $ Sisters  : int 1 2 1 1 0 0 1 1 1 2 ...
## $ BirthOrder: int 1 3 1 1 1 1 1 1 1 5 ...
## $ MilesHome : num 107.5 64.7 155.8 83.9 269.7 ...
## $ MilesLocal: num 0.43 0.2 0.7 1.2 0.3 0.9 0.1 0.8 0.52 0.3 ...
## $ Sleep     : num 7 6 9 8 7 7.5 7.5 7 9.5 7.5 ...
## $ BloodType : Factor w/ 5 levels "", "A", "AB", "B",..: 1 2 4 1 1 5 3 4 4 1 ...
## $ Height    : num 62 72 72 71.5 71 ...

dim(students)

## [1] 48 13

nrow(students)

## [1] 48

ncol(students)

## [1] 13
```

Single variables may be extracted using `$`. For example, here is the variable for *Brothers*. Note that when R writes a single variable, it writes the values in a row, even when we think of the variables as a column.

```
students$Brothers
## [1] 0 3 0 0 0 0 1 1 0 2 2 0 0 1 0 0 2 0 3 0 0 2 1 0 1 2 1 2 0 1 1 1 0 4 1
## [36] 0 2 0 7 0 1 1 0 0 1 0 1 0
```

The `with()` command is useful when we want to refer to variables multiple times in the same command. Here is an example that finds the number of siblings (brothers plus sisters) for each student in two ways.

```
with(students, Brothers + Sisters)
## [1] 1 5 1 1 0 0 2 2 1 4 2 0 1 1 0 0 2 1 6 2 1 2 1
## [24] 0 2 4 1 2 1 1 1 1 1 4 1 0 4 0 10 1 1 1 1 0 1 1
## [47] 1 0

students$Brothers + students$Sisters
## [1] 1 5 1 1 0 0 2 2 1 4 2 0 1 1 0 0 2 1 6 2 1 2 1
## [24] 0 2 4 1 2 1 1 1 1 1 4 1 0 4 0 10 1 1 1 1 0 1 1
## [47] 1 0
```

## 1.4 Square Bracket Operator

The square brackets or *subset* operator are one of the most powerful parts of the R language. Here is a way to extract the 1st, 6th, and 7th, columns and the first five rows. Note the code also shows the use of the colon operator for a sequence of numbers and the `c` function for combining a number of like items together. For a data frame, there are two arguments separated by a comma between the square brackets: which rows and which columns do we want? If left blank, all rows (or columns) are included. For a single array like `students$Brothers`, there is only a single argument.

```
1:6
## [1] 1 2 3 4 5 6

c(1, 6, 7)
## [1] 1 6 7

students[1:6, c(1, 6, 7)]
##      Sex Brothers Sisters
## 1 female      0      1
## 2  male      3      2
## 3  male      0      1
## 4  male      0      1
## 5  male      0      0
## 6 female      0      0

students[1, ]
```

```
##      Sex          Major Major2 Major3    Level Brothers Sisters
## 1 female actuarial science                freshman      0      1
##   BirthOrder MilesHome MilesLocal Sleep BloodType Height
## 1          1     107.5      0.43     7          62

students$Brothers[1:6]

## [1] 0 3 0 0 0 0
```

## 2 Categorical Variables

Categorical variables place cases into groups. Each group has a label called a *level*. By default, R orders the levels alphabetically. We will see later how to change this.

### 2.1 table()

The `table()` function is useful for summarizing one or more categorical variables. Here is an example using `Sex` and then both `Sex` and `BloodType`. In addition, the function `summary()` is useful for many purposes. We can use it on a single variable or an entire data frame.

```
with(students, table(Sex))

## Sex
## female  male
##      19     29

with(students, table(Sex, BloodType))

##      BloodType
## Sex      A AB  B  O
## female  2  6  1  5  5
## male    6  3  3  6 11

with(students, summary(Sex))

## female  male
##      19     29
```

### 2.2 Missing Data and read.csv()

Notice that students that did not report a blood type have this information stored as an empty string. We want this to be given the code `NA` which is the missing values code for this and any variables (like second and third majors) where the information was left blank. To do this correctly when reading in with `read.csv()`, we should add an argument to say empty fields are missing. The following example tells R to treat the string `NA` and an empty field between commas as missing data. By default, `table()` skips cases with missing values. We can change this by letting `useNA = "always"`.

```

students = read.csv("students.csv", na.strings = c("", "NA"))
with(students, table(Sex, BloodType))

##           BloodType
## Sex           A AB  B  O
##  female    6  1  5  5
##   male     3  3  6 11

with(students, table(BloodType))

## BloodType
##  A AB  B  O
##  9 4 11 16

with(students, table(BloodType, useNA = "always"))

## BloodType
##   A  AB   B   O <NA>
##   9   4  11  16   8

```

## 2.3 Proportions

We find proportions by dividing each count by the sum. Here is an example where `sum()` is used to sum the entries in the table. In the second case, proportions are rounded to 3 decimal places.

```

tab = with(students, table(BloodType))
tab/sum(tab)

## BloodType
##   A  AB   B   O
## 0.225 0.100 0.275 0.400

tab = with(students, table(Sex, BloodType))
round(tab/sum(tab), 3)

##           BloodType
## Sex           A  AB   B   O
##  female 0.150 0.025 0.125 0.125
##   male  0.075 0.075 0.150 0.275

```

## 2.4 Bar Graphs

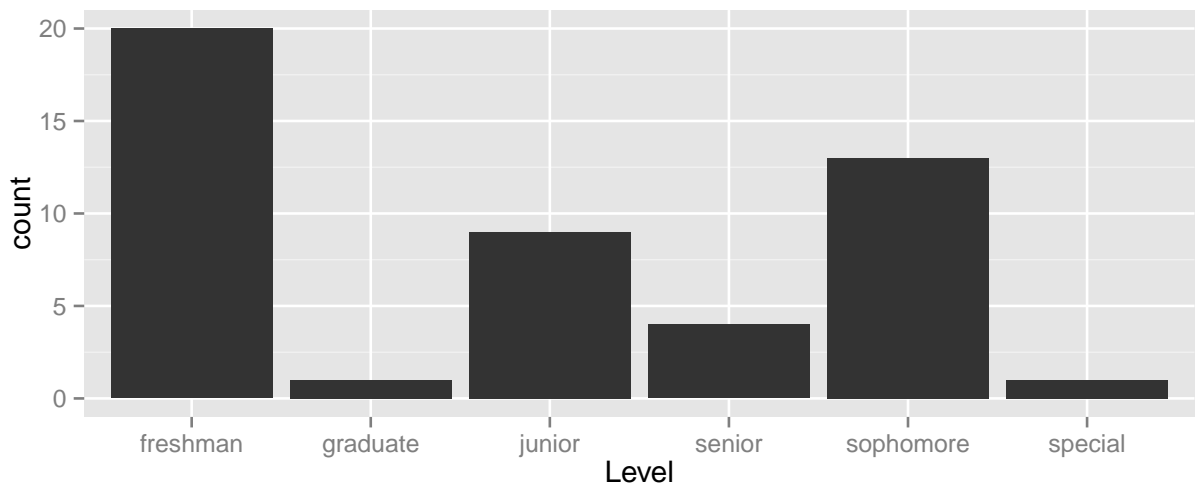
Bar graphs (or bar charts) are the best way to display categorical variables. Here is how to display *Level* using `ggplot2`. Note, this requires having typed `library(ggplot2)` earlier in the session. The syntax of a plotting command in `ggplot2` is to use `ggplot()` to define the data frame where variables are defined and to set aesthetics using `aes()` and then to add to this one or more layers with other commands. Aesthetics are characteristics that each plotted object can have, such as an

x-coordinate, a y-coordinate, a color, a shape, and so on. The layers we will use are all geometric representations of the data and have function names that have the form `geom_XXX()` where `XXX` is the name of the type of plot. This first example will use `geom_bar()` for a bar graph. With a bar graph, we set `x` to be the name of the categorical variable and `y` is automatically chosen to be the count.

```
require(ggplot2)

## Loading required package: ggplot2

ggplot(students, aes(x = Level)) + geom_bar()
```



## 2.5 reorder()

The preceding graph would be improved if the order `Level` was not alphabetical. `Level` has a natural order (at least in part) from freshman to senior, followed by special and graduate (the naturalness of the order breaks down at the end). The function `reorder()` can be used to change the order of the levels of a factor. The way it works is to include a second argument which is a quantitative variable of the same length: the levels are ordered so that the mean value for each group goes from smallest to largest. So, if we were to use `reorder(Level, Brothers)`, for example, the groups would be reordered based on the average number of brothers for each level. As there is no variable in the data set that we can be sure will put `Level` in the desired order, we will create one with 1 for *freshman*, 2 for *sophomore*, and so on. Watch the use of `with()` and square brackets to select parts of objects. A single `=` is for assignment or setting the value of an argument to a function. The double `==` is to check equality. Here is the R code to do it for a temporary variable named `foo` that we create and then discard.

```
# create an empty variable foo by repeating 0 for the number of cases in
# students
foo = rep(0, nrow(students))
# set the positions where Level == 'freshman' to be 1
foo[with(students, Level == "freshman")] = 1
# now do the others
```

```

foo[with(students, Level == "sophomore")] = 2
foo[with(students, Level == "junior")] = 3
foo[with(students, Level == "senior")] = 4
foo[with(students, Level == "special")] = 5
foo[with(students, Level == "graduate")] = 6
# look at foo to see if it looks right
foo

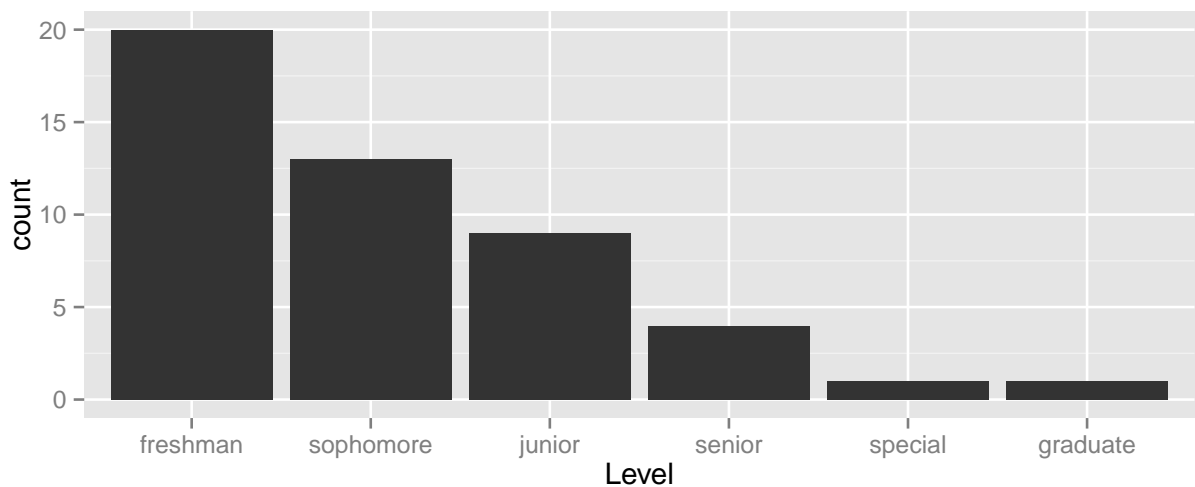
## [1] 1 3 1 4 2 6 2 4 1 1 1 3 2 1 3 1 2 1 2 1 1 2 4 2 3 2 1 3 2 5 3 4 2 1 2
## [36] 1 3 1 1 3 1 1 3 2 1 1 1 2

# change students$Level to the reordered version and discard foo
students$Level = with(students, reorder(Level, foo))
rm(foo)

```

Now, redo the plot. We see that the class has many first and second year students. The graduate student is your TA Katherine.

```
ggplot(students, aes(x = Level)) + geom_bar()
```



## 2.6 Bar plots for 2 Categorical Variables

If we want to examine the sex distribution by level in school, we can tabulate it and then make a stacked bar plot. I want to see the distribution of sexes by level, but the alternative plot is also legitimate. Interestingly, most freshmen in the course are female, but all other levels have more males (excluding the TA). What might this mean? Females are smart enough to start taking statistics courses as freshmen and the guys are slower to get with the program?

```

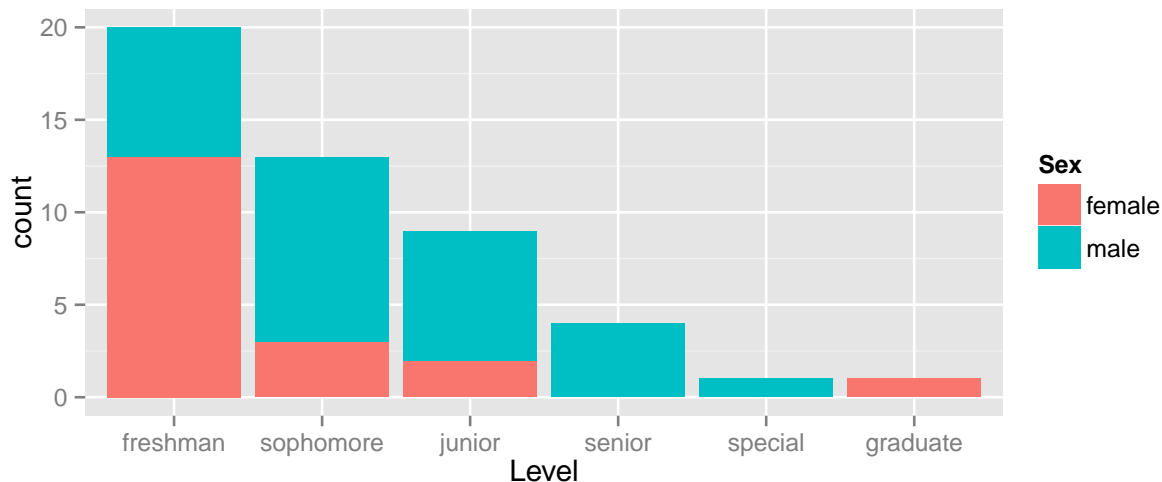
with(students, table(Sex, Level))

##           Level
## Sex      freshman sophomore junior senior special graduate

```

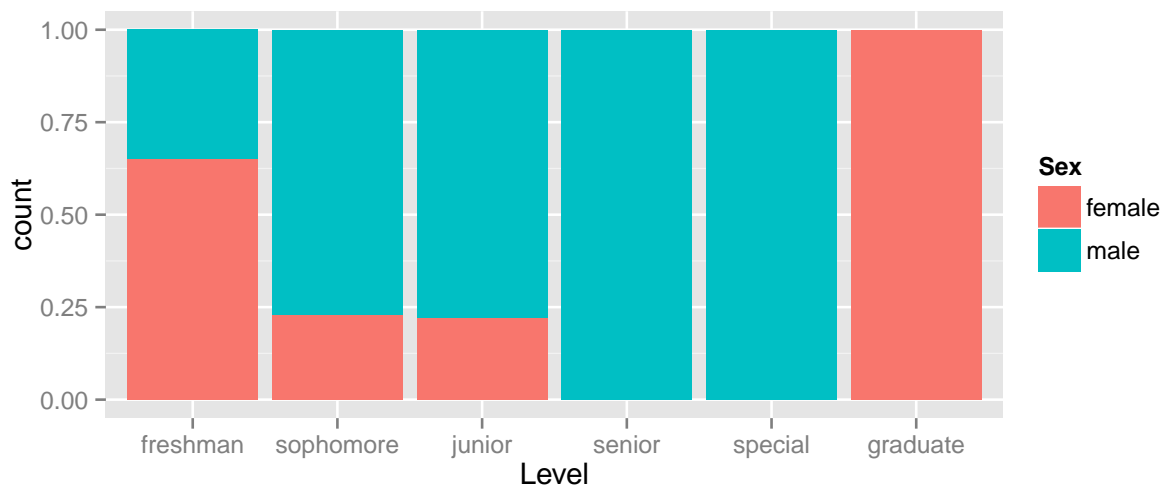
```
## female      13      3      2      0      0      1
## male        7      10     7      4      1      0

ggplot(students, aes(x = Level, fill = Sex)) + geom_bar()
```



If we want to see the proportion of females and males in each level, we change the *position* attribute of the bar plot to "fill".

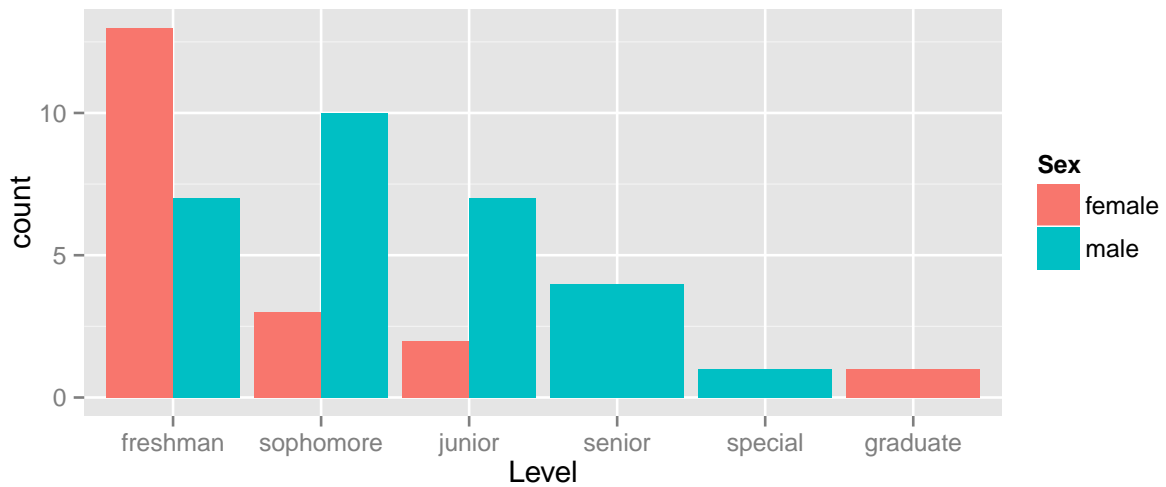
```
ggplot(students, aes(x = Level, fill = Sex)) + geom_bar(position = "fill")
```



Change the *position* to "dodge" if we want to have separate bars for each sex within each level. (The bars *dodge* each other to avoid overlapping.) This is very unsatisfactory when some combinations of levels have zero counts as the widths of the bars are not what we expect. There is a complicated workaround that involves adding fake students with count zero, but it should not be so hard to do the right thing.



```
ggplot(students, aes(x = Level, fill = Sex)) + geom_bar(position = "dodge")
```



### 3 One Quantitative Variable

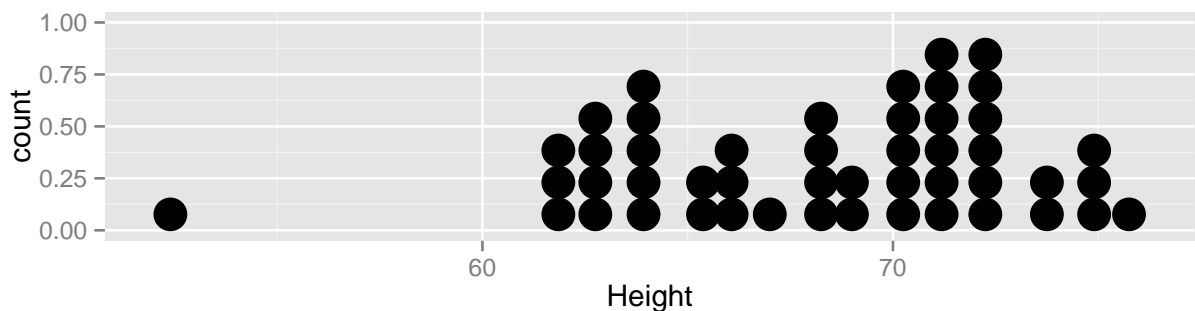
There are multiple ways to graphically display the distribution of a single quantitative variable. They differ in how clearly they show various features of the distribution.

#### 3.1 Dot plots

For small data sets, a dot plot is an effective way to visualize all of the data. A dot is placed at the appropriate value on the x axis for each case, and dots stack up. Here is how to make a dotplot with `ggplot2` for the *Height* variable from *students*.

```
ggplot(students, aes(x = Height)) + geom_dotplot()
```

*## stat\_bindot: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.*



The interesting feature of this variable is the single observation far lower than the others. We can find its exact value by a number of different ways. Look around class some day and try to discern if this is an error or if we have a very short person in class with us.

```
with(students, min(Height))

## [1] 52.4

with(students, sort(Height)[1])

## [1] 52.4

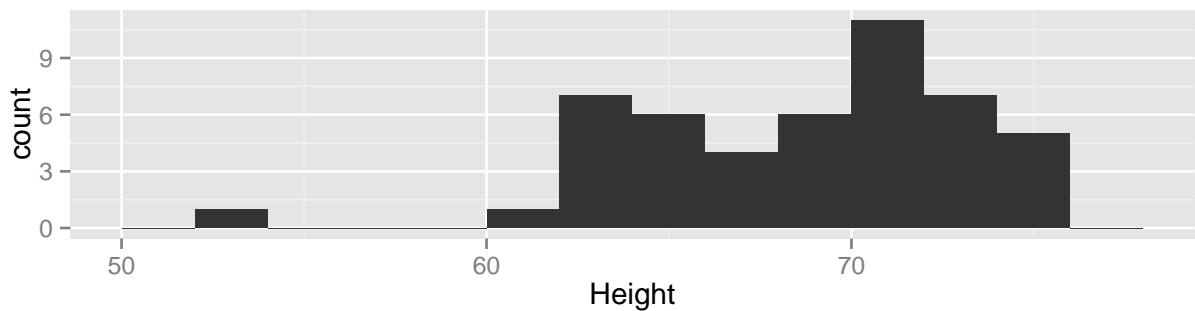
with(students, fivenum(Height))

## [1] 52.40 64.21 69.00 71.75 75.75
```

### 3.2 Histograms

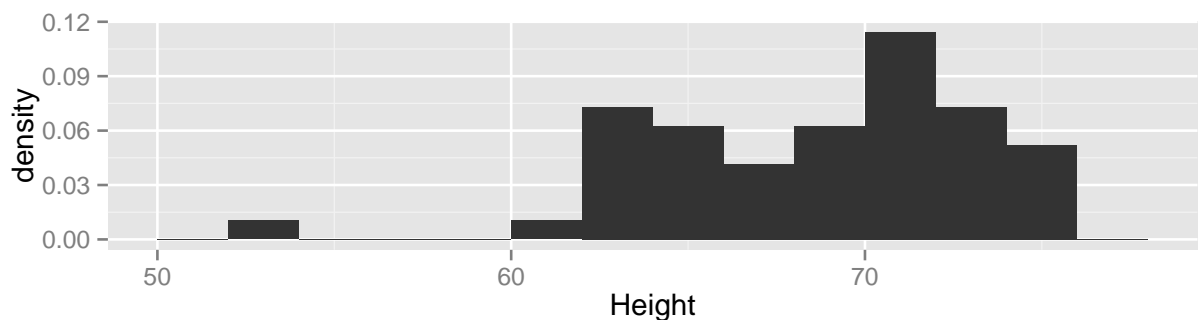
Another popular way to graph a single quantitative variable is with a histogram. Just use `geom_histogram()` instead. There are ways to specify the breakpoints. It is often useful to specify the `binwidth` and/or `origin` manually.

```
ggplot(students, aes(x = Height)) + geom_histogram(binwidth = 2)
```



In the previous plot, the y-axis corresponds to the count of observations in each bin. Another common way to present histograms is so that the total area is one, so that the area of a bin represents the proportion of data in the corresponding interval. Adding an aesthetic `y=..density..` makes this change.

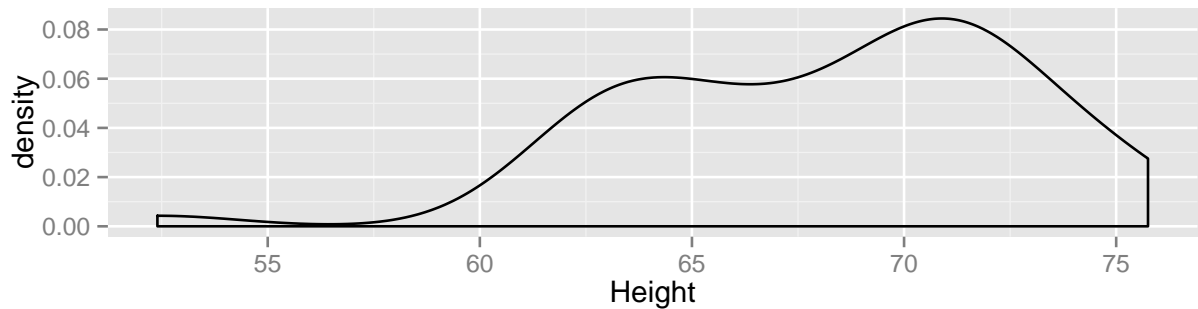
```
ggplot(students, aes(x = Height)) + geom_histogram(binwidth = 2, aes(y = ..density..))
```



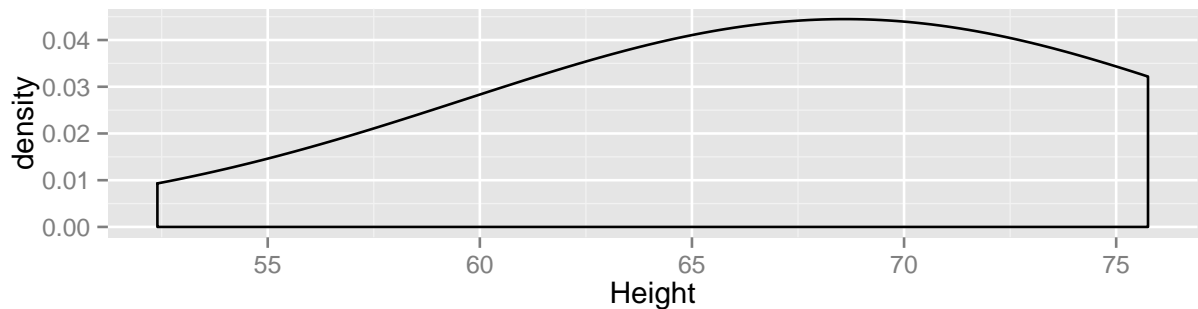
### 3.3 Density Plots

Density plots are similar to histograms on a density scale, but instead of fixed bins or intervals with jumps at the boundaries, are smooth. The argument `adjust` to `geom_density` regulates how smooth the density estimate is, with larger values resulting in smoother graphs.

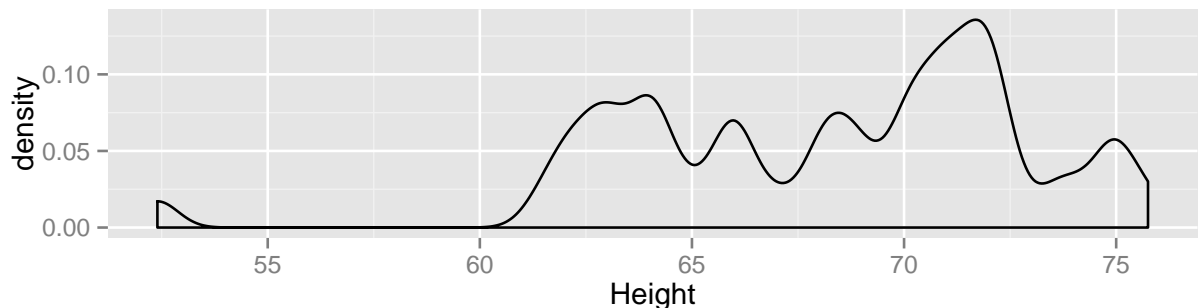
```
ggplot(students, aes(x = Height)) + geom_density()
```



```
ggplot(students, aes(x = Height)) + geom_density(adjust = 4)
```



```
ggplot(students, aes(x = Height)) + geom_density(adjust = 0.25)
```



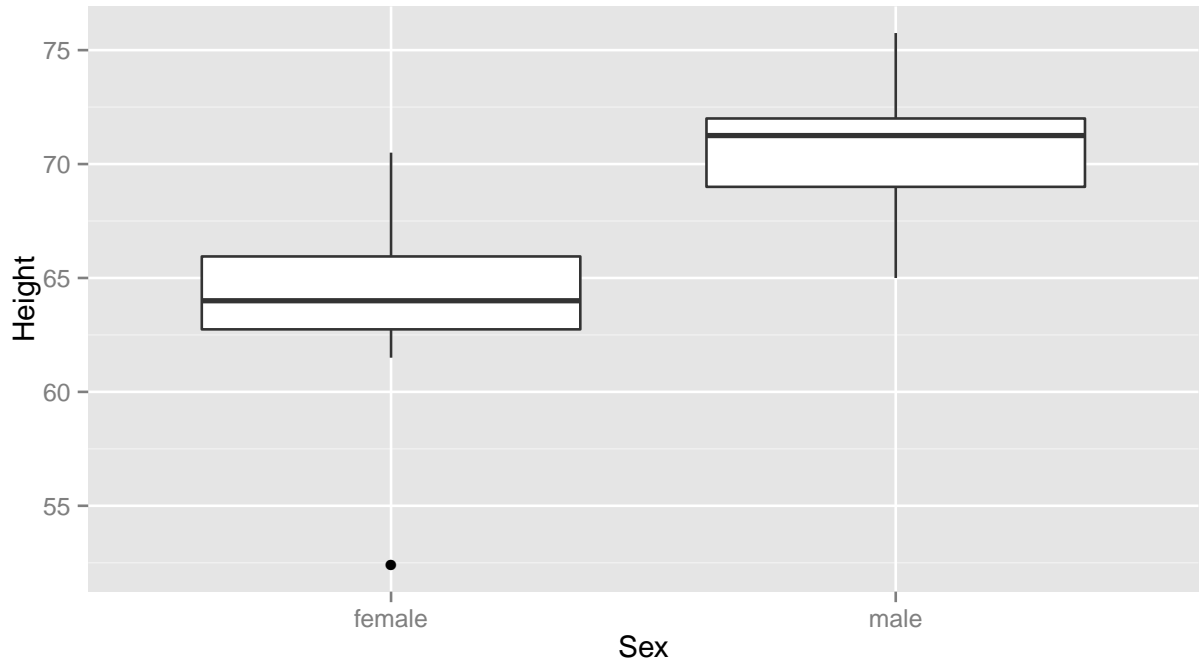
### 3.4 Box and Whisker Plots

Box and whisker plots are highly summarized representations of the data, essentially condensing the entire sample to a five number summary, plus locations of outliers as defined by extreme distance

outside the range of the middle half of the data. While the textbook (and many other statistical software packages) use boxplots to summarize single variables, `ggplot2` by construction only uses box plots for comparing two or more distributions. It is, however, possible to force the functions to handle a single variable by using an artificial grouping variable.

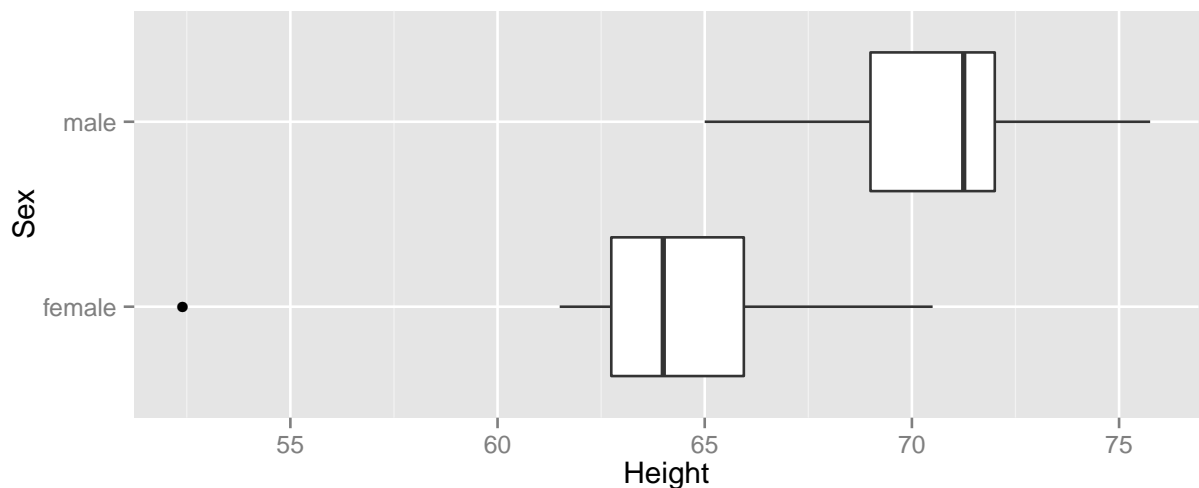
First, here is an example of side-by-side boxplots with the grouping variable on the x-axis and the quantitative variable on the y-axis.

```
ggplot(students, aes(x = Sex, y = Height)) + geom_boxplot()
```



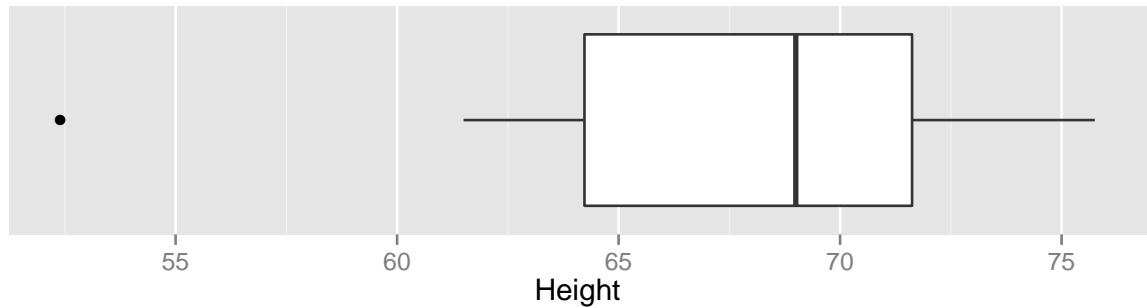
One additional layer command flips the coordinates so that the quantitative variable is horizontal (which often makes comparisons about center and spread easier to perceive).

```
ggplot(students, aes(x = Sex, y = Height)) + geom_boxplot() + coord_flip()
```



To make a boxplot for a single variable, we need to create a fake grouping variable. Other commands change the axis label and tick marks from the false axis. Here is an example.

```
ggplot(students, aes(x = factor(0), y = Height)) + geom_boxplot() + xlab("") +  
  scale_x_discrete(breaks = NULL) + coord_flip()
```



### 3.5 Quantitative Summaries

The function to compute the mean in R is called `mean()`. The mean height of all students and computed separately for females and males are computed like this.

```
with(students, mean(Height))  
  
## [1] 68.23  
  
with(students, by(Height, Sex, mean))  
  
## Sex: female  
## [1] 64.16  
## -----  
## Sex: male  
## [1] 70.9
```

Similarly, the median is computed with `median()` and the standard deviation with `sd()`.

```
with(students, median(Height))  
  
## [1] 69  
  
with(students, by(Height, Sex, median))  
  
## Sex: female  
## [1] 64  
## -----  
## Sex: male  
## [1] 71.25  
  
with(students, sd(Height))
```

```
## [1] 4.667

with(students, by(Height, Sex, sd))

## Sex: female
## [1] 3.944
## -----
## Sex: male
## [1] 2.816
```

The five number summary (minimum, lower quartile, median, upper quartile, maximum) is computed with `fivenum()`. Quantiles in general can be found with `quantile()`. Here are examples. Notice that `fivenum()` and `quantile()` have different algorithms for finding quartiles. In fact, there are 9 different ways to define quantiles implemented in `quantile()`. Switching from the default `type=7` to `type=2` will match `fivenum()`.

```
with(students, fivenum(Height))

## [1] 52.40 64.21 69.00 71.75 75.75

with(students, quantile(Height))

## 0% 25% 50% 75% 100%
## 52.40 64.23 69.00 71.62 75.75

with(students, quantile(Height, type = 2))

## 0% 25% 50% 75% 100%
## 52.40 64.21 69.00 71.75 75.75

with(students, quantile(Height, c(0.1, 0.25, 0.5, 0.75, 0.9)))

## 10% 25% 50% 75% 90%
## 62.84 64.23 69.00 71.62 73.65

with(students, summary(Height))

## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 52.4 64.2 69.0 68.2 71.6 75.8
```

### 3.6 Effects of an Outlier

The unusual observation, in fact, is an error caused by a mistaken conversion from metric measurements to inches. The true height of the female student we have recorded as 52.4 inches should have been 62.99 inches. We should correct this in the data file, but here is how to fix it for the `students` data frame. First, find which observation it is.

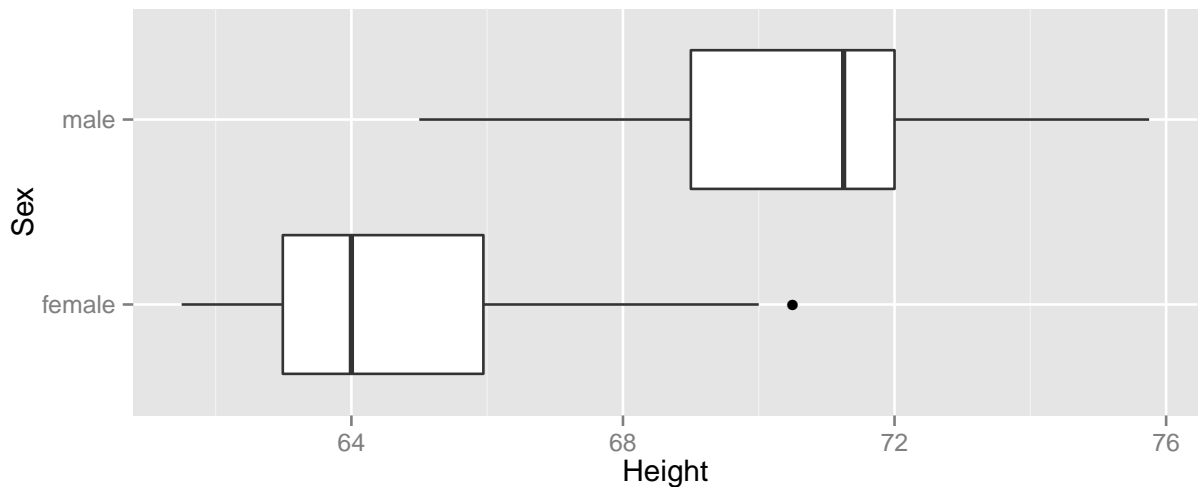
```
with(students, which(Height < 55))
## [1] 39
with(students, Height[39])
## [1] 52.4
```

Now make the correction.

```
students$Height[39] = 62.99
```

The side-by-side boxplots now look much different.

```
ggplot(students, aes(x = Sex, y = Height)) + geom_boxplot() + coord_flip()
```



Let's see how the mean, median, and standard deviation of the heights of female students compare now that we have corrected the data. With the outlier, the mean height was 64.16 inches, the median was 64 inches, and the standard deviation was 3.94 inches.

```
with(students, mean(Height[Sex == "female"]))
## [1] 64.72
with(students, median(Height[Sex == "female"]))
## [1] 64
with(students, sd(Height[Sex == "female"]))
## [1] 2.761
```

Notice that the median is unchanged, but the numerical values of the mean and the standard deviation do change substantially. The incorrect mean was 0.6 inches too low and the incorrect standard deviation was over 1.1 inches too large.

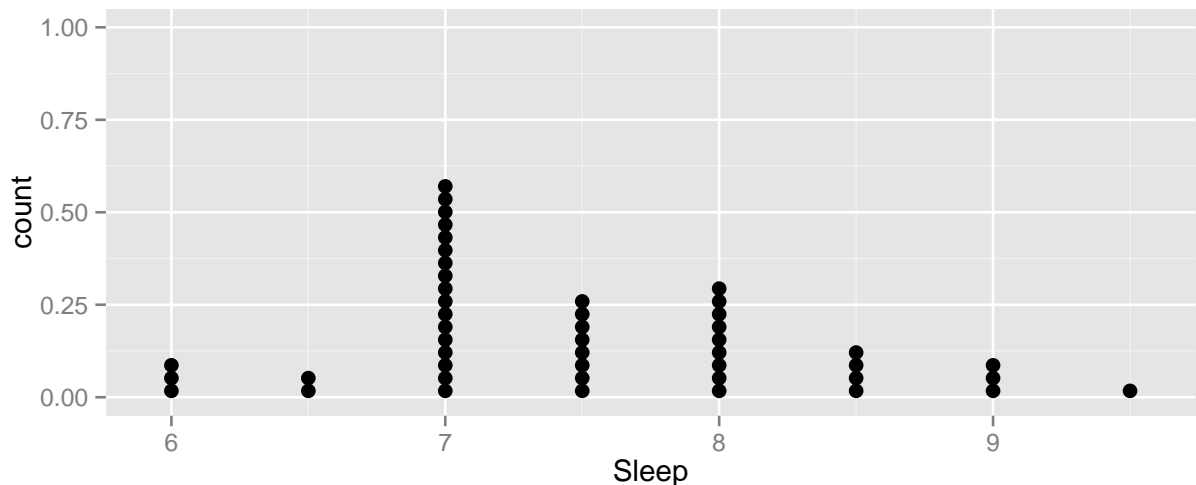
## 4 Relationships Between a Categorical and Quantitative Variable

It is very common in statistical settings to be interested in how a quantitative variable differs among groups. We have seen height versus sex in the previous section. In this section, we will review graphs and summary calculations for this situation.

### 4.1 Faceting

One thing `ggplot2` does very well is to make it easy to split a variable among the groups of a quantitative variable to show univariate displays for each group separately using common scaling to allow easy visual comparisons. For something different, let's look at the distributions of the number of hours of sleep students get each night versus their level in school. Let me first show this as a dotplot. Here is the command that shows hours of sleep for the entire sample. I have adjusted the dot size.

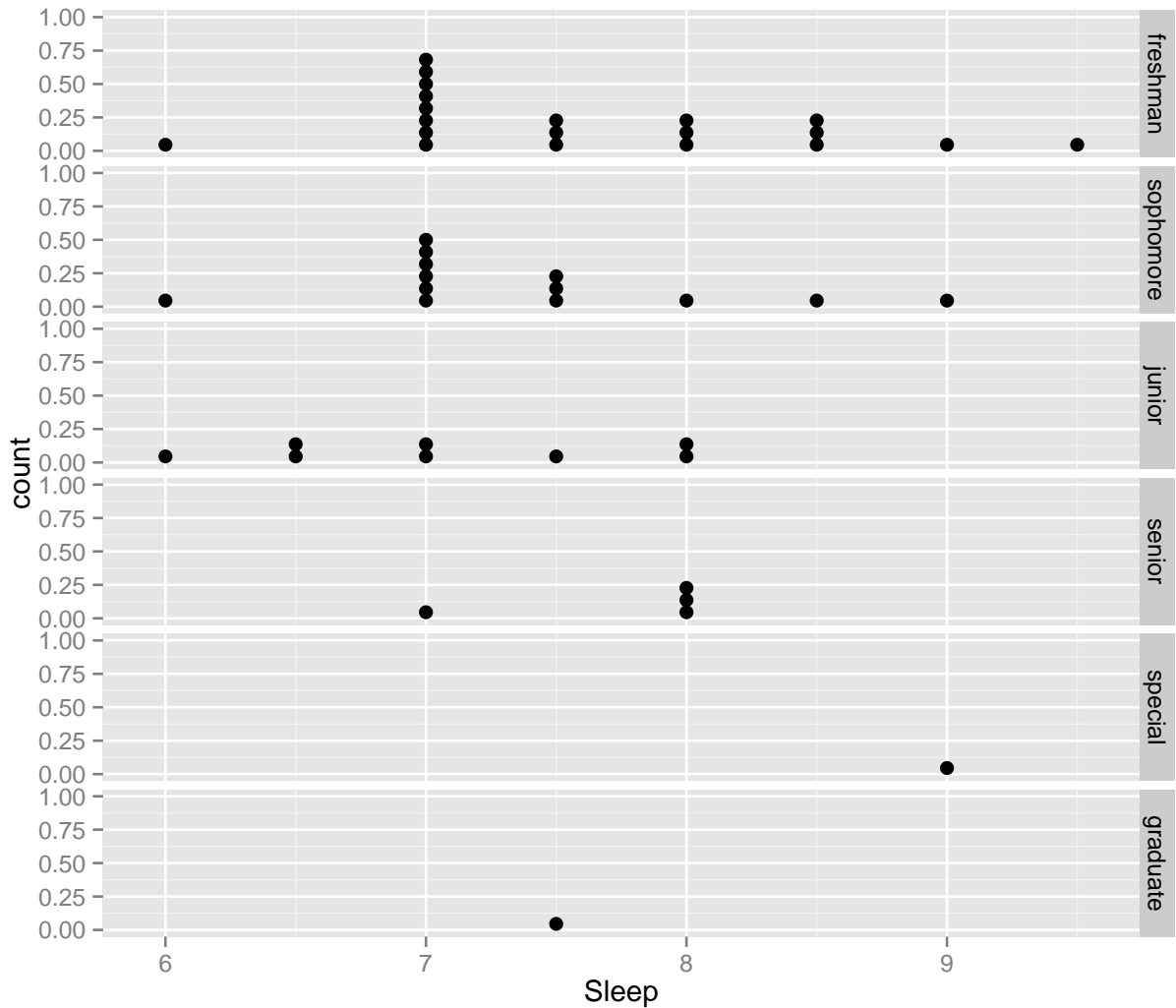
```
ggplot(students, aes(x = Sleep)) + geom_dotplot(dotsize = 0.4)
```



Here is a plot that first separates the students by level and makes a separate dotplot for each level. I will have a different row for each plot using `facet_grid()`, where the argument specifies which categorical variables to use to split the data by row and or column. A period means do not split in that dimension. In the example, each level of Level gets its own row, but there is only a single column of display.

```
ggplot(students, aes(x = Sleep)) + geom_dotplot(dotsize = 0.4) + facet_grid(Level ~  
.)
```





Visually, it looks like freshmen and sophomores get more sleep on average than juniors and seniors. Recall the use of `by()` to find means within groups. The extra argument `na.rm=TRUE` instructs `mean()` to ignore missing values when computing means.

```
with(students, by(Sleep, Level, mean, na.rm = TRUE))
```

```
## Level: freshman
## [1] 7.625
## -----
## Level: sophomore
## [1] 7.385
## -----
## Level: junior
## [1] 7.062
## -----
## Level: senior
## [1] 7.75
## -----
```

```

## Level: special
## [1] 9
## -----
## Level: graduate
## [1] 7.5

with(students, by(Sleep, Level, median, na.rm = TRUE))

## Level: freshman
## [1] 7.5
## -----
## Level: sophomore
## [1] 7
## -----
## Level: junior
## [1] 7
## -----
## Level: senior
## [1] 8
## -----
## Level: special
## [1] 9
## -----
## Level: graduate
## [1] 7.5

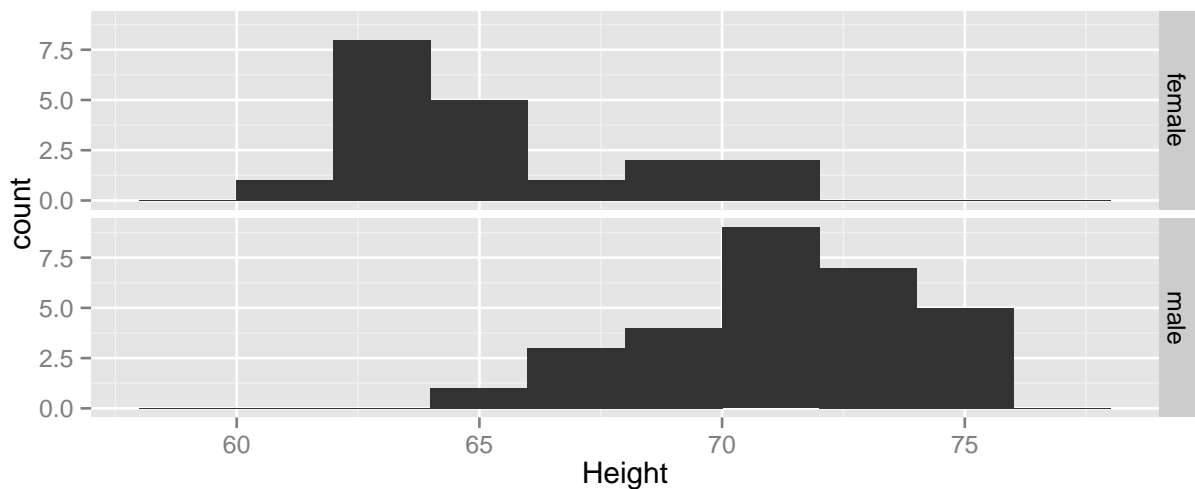
```

Stacking histograms is another way to compare distributions of a quantitative variable for different groups. Here is code to compare heights of females and males.

```

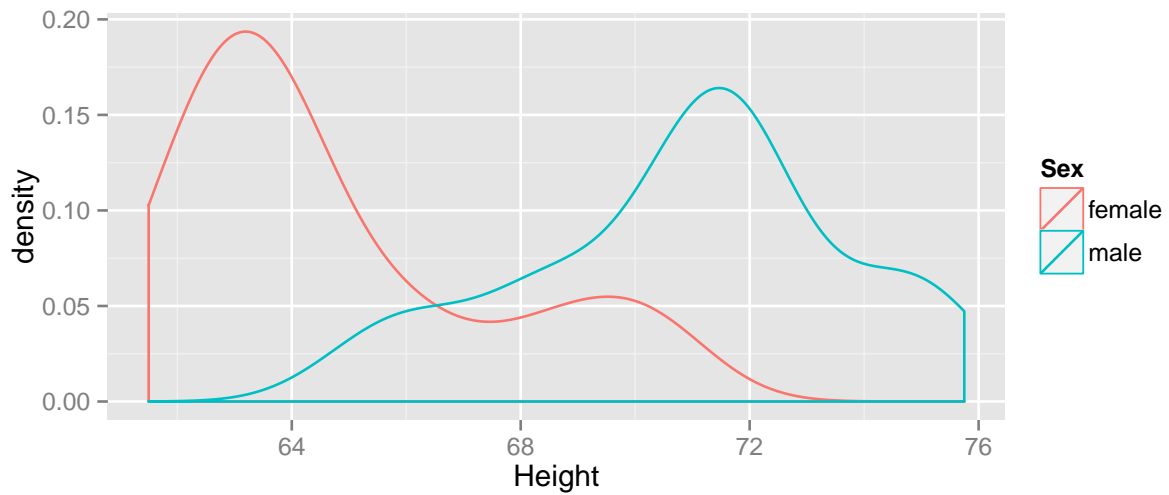
ggplot(students, aes(x = Height)) + geom_histogram(binwidth = 2) + facet_grid(Sex ~
.)

```



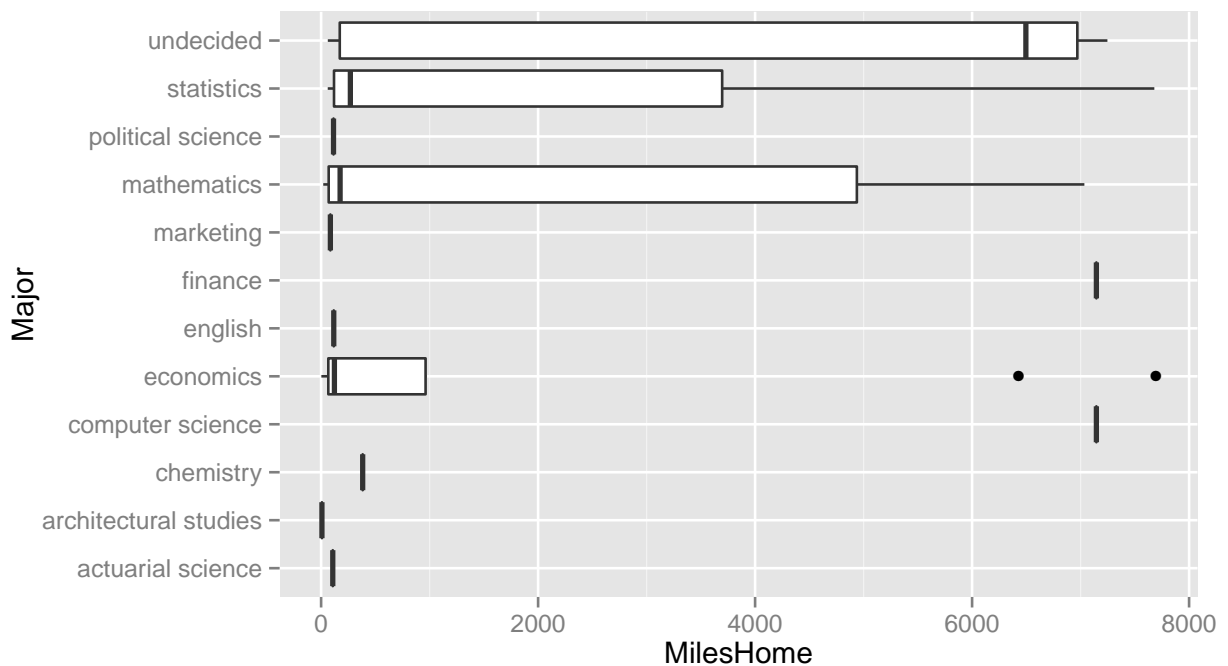
Because density plots are lines, a few of them can be drawn together on the same plot, making them easy to compare. Here is an example that uses the `color` aesthetic to distinguish the sexes.

```
ggplot(students, aes(x = Height, color = Sex)) + geom_density()
```



Density plots and histograms can show many features of a distribution, but can be too messy if there are many groups. From the data on students in the course, if we wanted to examine miles from home for each different major (first major as selected by student), side-by-side boxplots will be best because there are too many groups to look at many features at the same time.

```
ggplot(students, aes(x = Major, y = MilesHome)) + geom_boxplot() + coord_flip()
```



## 5 Two Quantitative Variables

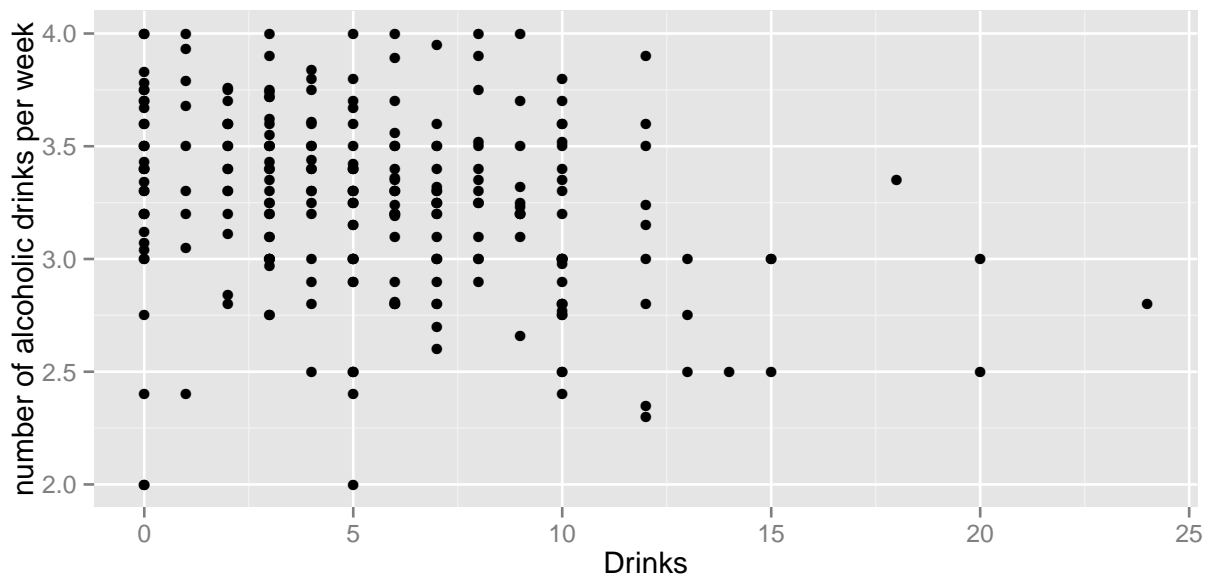
A scatterplot is the most useful way to display two quantitative variables. More information can be added to plots by using different colors or symbols for different groups. Overlapping points can be handled by *jittering* (moving the positions a small random amount) or by using partially opaque points.

For numerical summaries of pairs of quantitative variables, we will look at the correlation coefficient as a measure of the strength of the linear relationship between two variables. Here are some examples.

### 5.1 Scatterplots

In the *SleepStudy* data set, we can examine the number of drinks a student consumes per week as an explanatory variable for their GPA.

```
ggplot(SleepStudy, aes(x=Drinks,y=GPA)) + geom_point() +  
  ylab('number of alcoholic drinks per week')
```

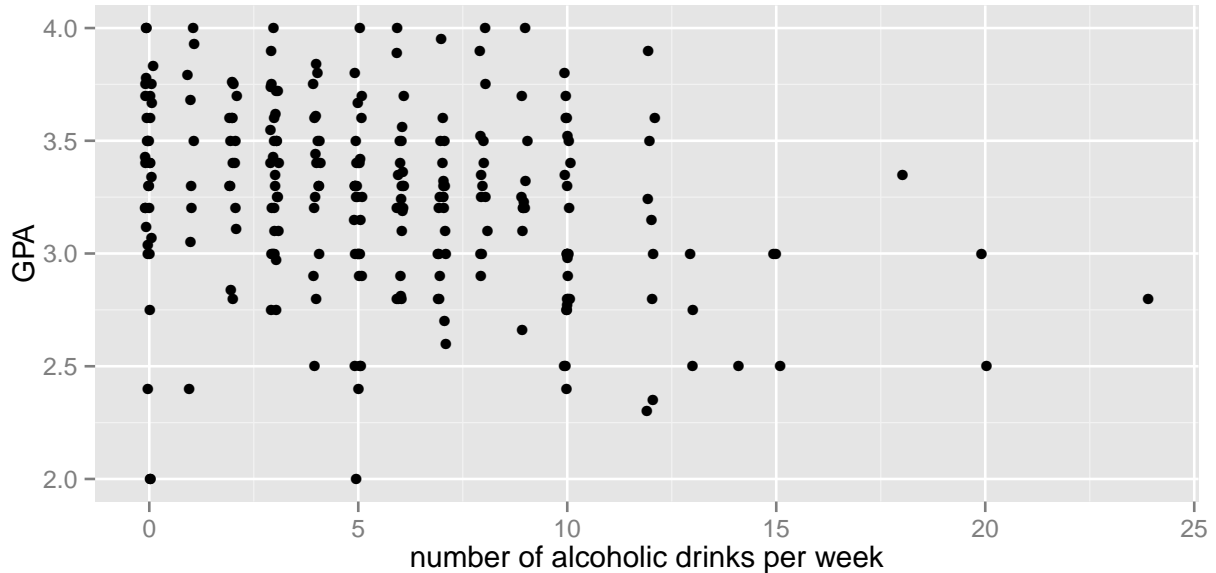


We see from this graph a weak negative association between the variables. We can measure this with the correlation coefficient.

```
with(SleepStudy, cor(Drinks, GPA))  
## [1] -0.2693
```

The previous graph also has a fair amount of overplotting because drinks is a small integer and some students have equal GPAs. The following command will jitter all of the points in a controlled way. We choose to jitter only the width by a small amount and not the height so the GPA remains accurate and we can still be clear the true number of drinks.

```
ggplot(SleepStudy, aes(x=Drinks,y=GPA)) +
  geom_point(position=position_jitter(w=0.1,h=0)) +
  xlab('number of alcoholic drinks per week')
```



## 5.2 Least-squares Lines

Later this semester, we will examine linear regression and summarize pairs of quantitative variables with a line that predicts the response variable  $y$  as a function of an explanatory variable  $x$ . Regression is a special case of a *linear model*. The R function for fitting a linear model is `lm()` and we can use it like this. The function `coef()` returns the estimated coefficients of the fitted linear model. Below, `fit` is an unimaginative name for the object that contains the fitted model.

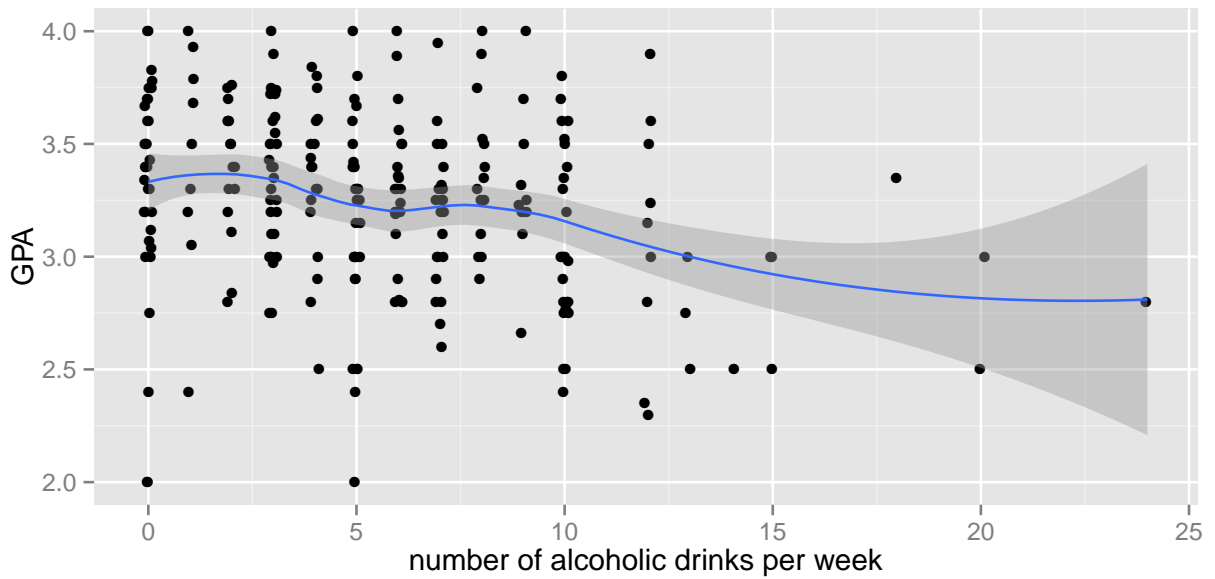
```
fit = lm(GPA ~ Drinks, data = SleepStudy)
coef(fit)

## (Intercept)      Drinks
##      3.39186     -0.02659
```

We can add the line to the plot using `geom_smooth()`. The default behavior is to add a locally smooth regression line with a shaded gray area to represent pointwise 95 percent confidence regions for the true mean GPA as a function of drinks. We first show this plot and then the options to override the default and just plot the simple regression line.

```
ggplot(SleepStudy, aes(x=Drinks,y=GPA)) +
  geom_point(position=position_jitter(w=0.1,h=0)) +
  geom_smooth() +
  xlab('number of alcoholic drinks per week')
```

```
## geom_smooth: method="auto" and size of largest group is <1000, so using loess.
Use 'method = x' to change the smoothing method.
```



```
ggplot(SleepStudy, aes(x=Drinks,y=GPA)) +
  geom_point(position=position_jitter(w=0.1,h=0)) +
  geom_smooth(method="lm", se=FALSE) +
  xlab('number of alcoholic drinks per week')
```

