Basic Plots in R

Dr. Sarah Hunter

3/20/2020

Introduction to R Plots

One of R's strengths as a statistical software is its versatility when it comes to making graphs. In this lesson, we will explore R's base plot functions. This requires no user-created packages in order to work. I will also walk you through the options that you can use to create functional and attractive plots.

To start our plots, we need to first set our `working directory` and load our data. For this example, I will be using the American National Election Study data from the last lesson. After I set my `working directory` and load the data, I use the `summary` command to make sure the data are loaded correctly.

```r
#Setting the working directory
setwd("/Users/sarahhunter/Desktop/Data")

#Loading CSV data
mydata<-read.csv("nes2004subset3.csv")

#Getting summary statistics
summary(mydata)
```

```
##        X             religion          bush           female      
##  Min.   :   1.0   Min.   :1.000   Min.   :0.000   Min.   :0.0000  
##  1st Qu.: 302.5   1st Qu.:1.000   1st Qu.:0.000   1st Qu.:0.0000  
##  Median : 607.0   Median :1.000   Median :1.000   Median :1.0000  
##  Mean   : 606.1   Mean   :2.311   Mean   :0.508   Mean   :0.5319  
##  3rd Qu.: 908.5   3rd Qu.:2.000   3rd Qu.:1.000   3rd Qu.:1.0000  
##  Max.   :1212.0   Max.   :7.000   Max.   :1.000   Max.   :1.0000  
##                   NA's   :15      NA's   :396                     
##    unionhouse        partyid         eval_WoT         eval_HoE      
##  Min.   :0.0000   Min.   :0.000   Min.   :-2.000   Min.   :-2.0000  
##  1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:-2.000   1st Qu.:-2.0000  
##  Median :0.0000   Median :3.000   Median : 1.000   Median :-1.0000  
##  Mean   :0.1724   Mean   :2.872   Mean   : 0.152   Mean   :-0.3467  
##  3rd Qu.:0.0000   3rd Qu.:5.000   3rd Qu.: 2.000   3rd Qu.: 2.0000  
##  Max.   :1.0000   Max.   :6.000   Max.   : 2.000   Max.   : 2.0000  
##  NA's   :6        NA's   :17      NA's   :29       NA's   :36       
##     ideology       bush_therm       education         income     
##  Min.   :1.000   Min.   :  0.00   Min.   :0.000   Min.   : 1.00  
##  1st Qu.:3.000   1st Qu.: 30.00   1st Qu.:3.000   1st Qu.:11.00  
##  Median :4.000   Median : 60.00   Median :4.000   Median :16.00  
##  Mean   :4.268   Mean   : 54.94   Mean   :4.305   Mean   :14.97  
##  3rd Qu.:6.000   3rd Qu.: 85.00   3rd Qu.:6.000   3rd Qu.:20.00  
##  Max.   :7.000   Max.   :100.00   Max.   :7.000   Max.   :23.00  
##  NA's   :288                                      NA's   :141    
```

While this is not a large dataset, sometimes you will be working with datasets that contain thousands of rows and 100 variables. Therefore, sometimes asking for all of the summary statistics can take up a lot of room. There are a few handy commands to make sure your data are loaded correctly without using the `summary` command. The first of these is the `head` command. This command shows only the first 5 rows of a dataset. It is used by:

```
head(mydata)
```

```
##   X religion bush female unionhouse partyid eval_WoT eval_HoE ideology
## 1 1        7    0      0          1       3       NA        0        4
## 2 2        1    0      0          0       2       -1       -1        4
## 3 3        1    1      1          0       6        2        2        6
## 4 4        1   NA      0          0       3       -2       -1        4
## 5 5        1    1      1          0       6        2        2        6
## 6 6        1   NA      0          0       3       -2       -2        6
##   bush_therm education income
## 1         70         7     17
## 2         40         4     19
## 3        100         6     23
## 4         50         2      3
## 5        100         3     12
## 6         60         7     12
```

However, this can also be a bit cumbersome. In this case, I can also use the `dim` command, which tells the dimensions of the dataset. The `dim` command returns the number of rows and columns (number of observations and number of variables respectively). It is used like:

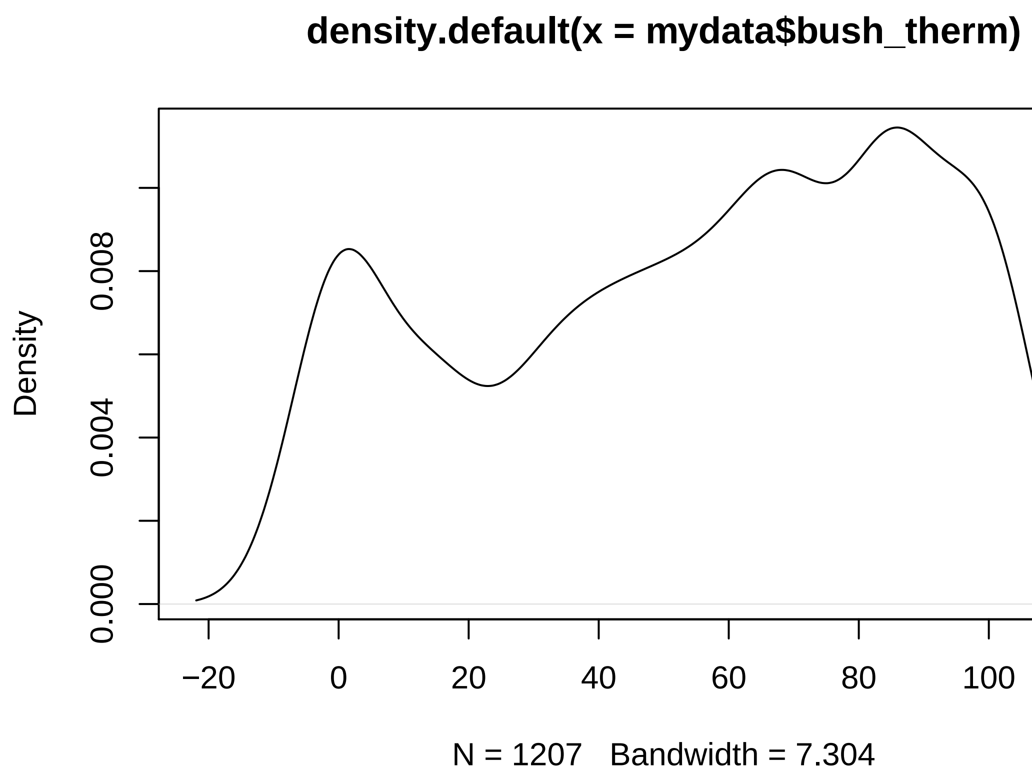
```
dim(mydata)
```

```
## [1] 1207   12
```

Univariate Displays

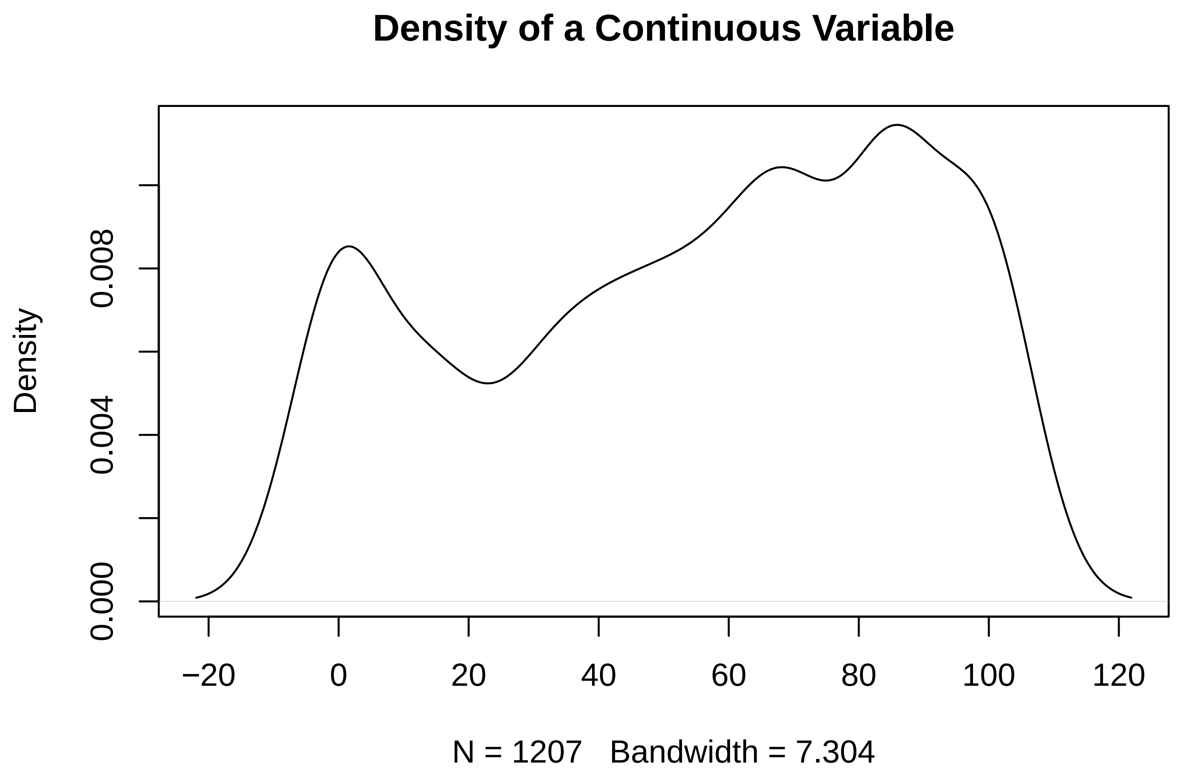
Now we can see that are data are loaded correctly and we can begin making plots. The first group of plots we will cover are univariate plots. These are plots that show the shape of a variable. In other words, univariate displays are visual depictions of descriptive statistics. The first plot we will look at is the density plot. This is a plot that shows the frequency of observations at certain values of the variable. Essentially, the density plot shows how many observations of that variable are at each possible value of the variable. To make a density plot in R, you start with the basic plot command: `plot()`. However, R needs to know what kind of plot to use, therefore, we have to include the `density()` function inside the `plot()` command. Density plots are created in R by:

```
plot(density(mydata$bush_therm))
```



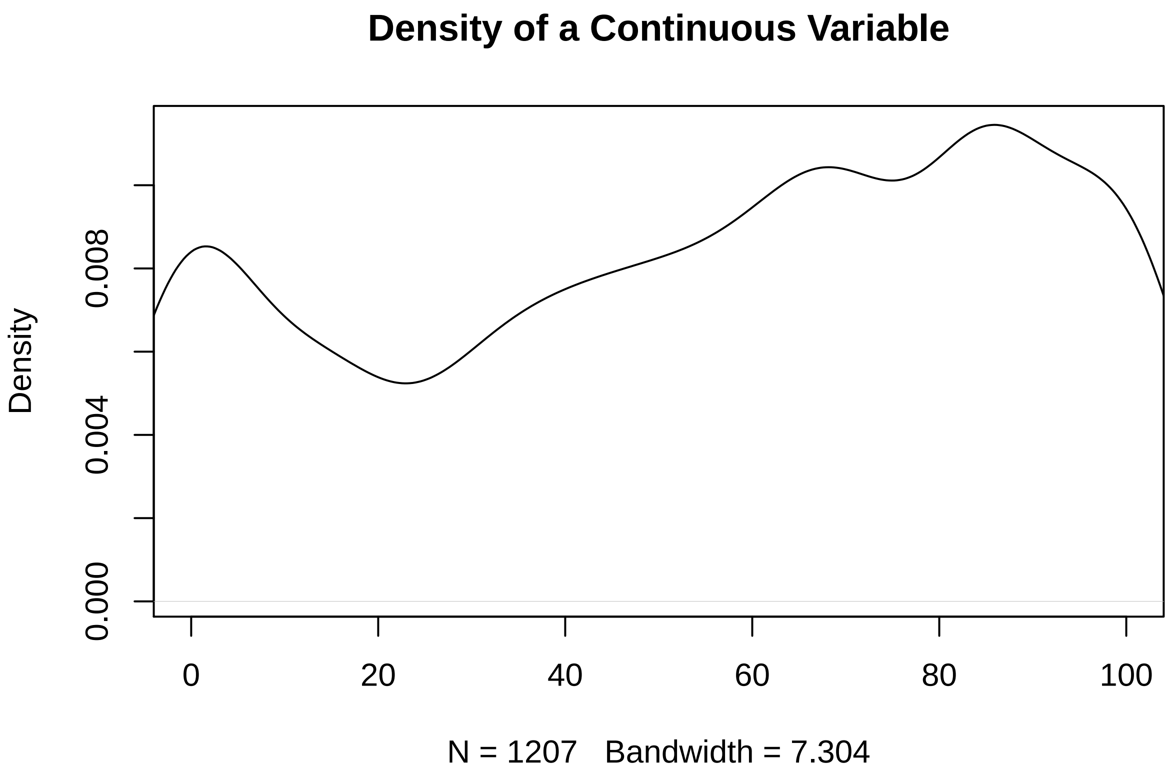
This plot could be more effective. We can add a title by using options in the base plot function. Adding a title uses the option `main=` and titles should always be in quotation marks. Also notice the placement of the option. It is always after the main object to be plotted (the density of the variable) and separated by a comma.

```
plot(density(mydata$bush_therm), main = "Density of a Continuous Variable")
```



This plot is still not as useful as it could be due to the impossible nature of negative values and those over 100. This variable (bush_therm) is the rating the survey respondents gave to President George W. Bush about how warm they felt for him. The higher the number, the more warmth the respondent felt toward President Bush. This is a variable that is bounded between 0 and 100. We can adjust the axes of the plot to get rid of the impossible values and get a more accurate picture. We can make the X-axis (horizontal axis) of the plot by limiting the span of the axis, using the `xlim=c()` command. The `c()` part of the command is used to denote a vector, or a list of values. You will need to specify 2 values for the `xlim` command: the minimum and the maximum. To see it used in a plot, look at the following example:

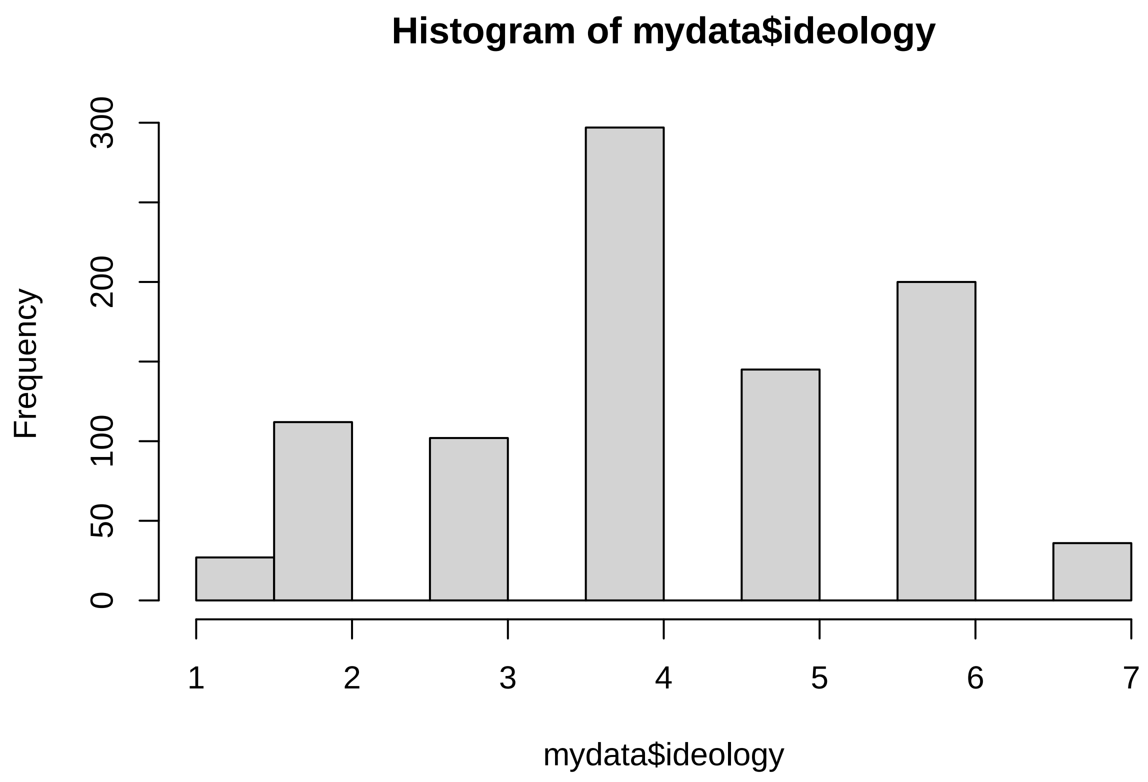
```
plot(density(mydata$bush_therm), main="Density of a Continuous Variable", xlim=c(0, 100))
```



Histograms

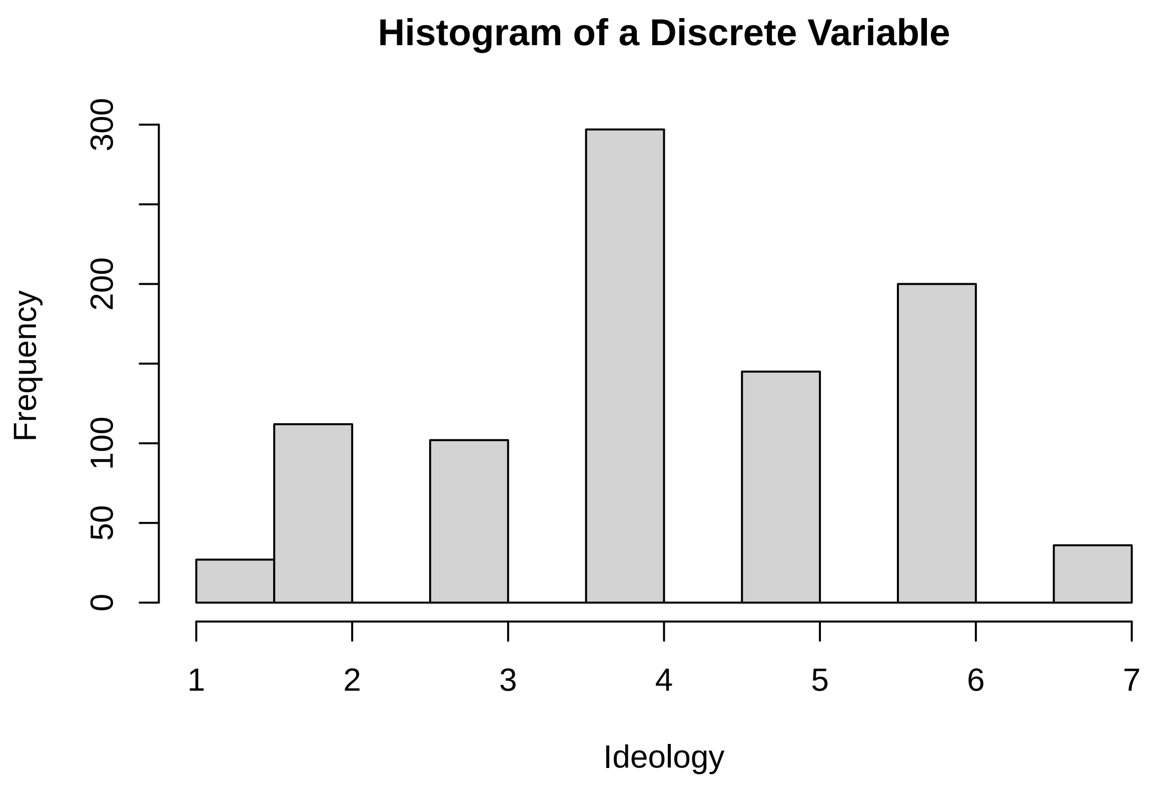
Density plots are great for continuous variables, however, they do not provide any information for a variable with more discrete values (i.e only a finite number of values). In this case, we would use a histogram. A histogram, like the density, shows the frequency of the value of a variable in your observations. Unlike a density plot, a histogram is not a continuous line. As the variable itself is a categorical variable, it makes sense to have a plot that has set "bins" or sections. The following code shows how to make a histogram using R's base plot functions:

```r
hist(mydata$ideology)
```



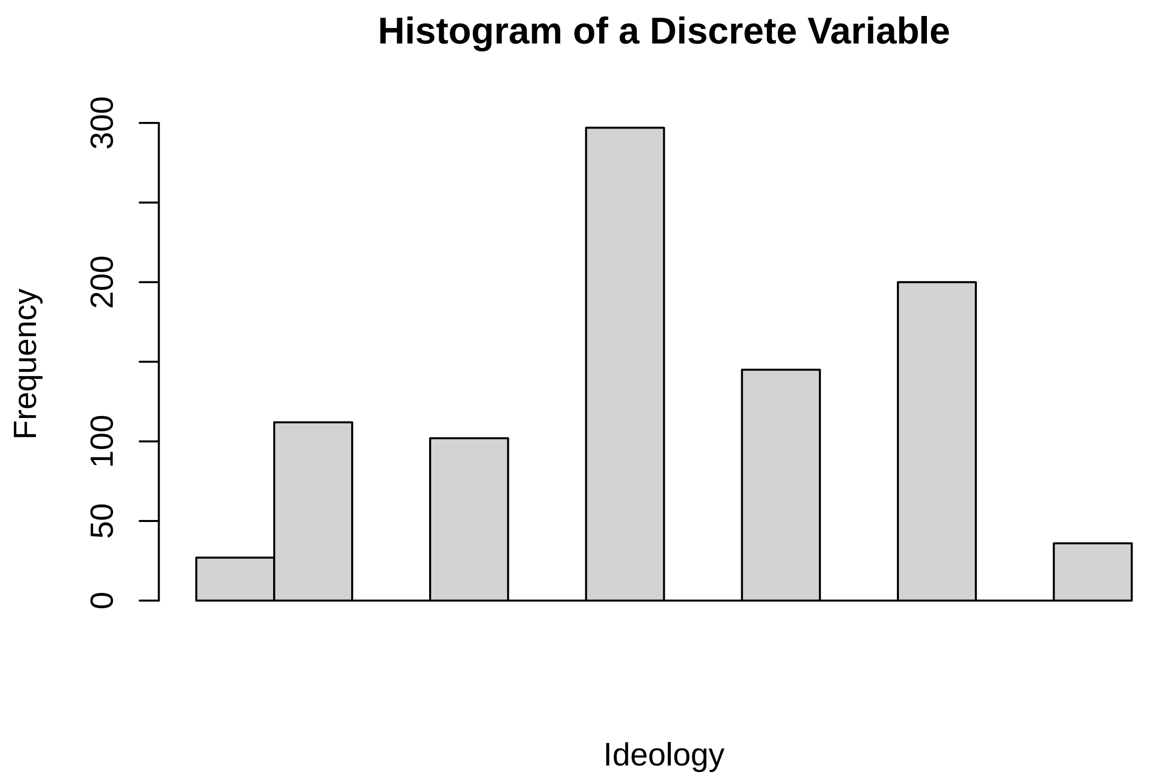
This plot is ugly. Remember that this is a categorical variable referring to the self-reported ideology score of the survey respondent. We can make this plot nicer using both the options discussed before, and a few histogram-specific tricks. The first step is to change the title of the plot using the `main=` command. Then we should also change the label of the x-axis using the `xlab=` command. Notice that `xlab` uses the same structure as `main`:

```r
hist(mydata$ideology, main="Histogram of a Discrete Variable", xlab="Ideology")
```



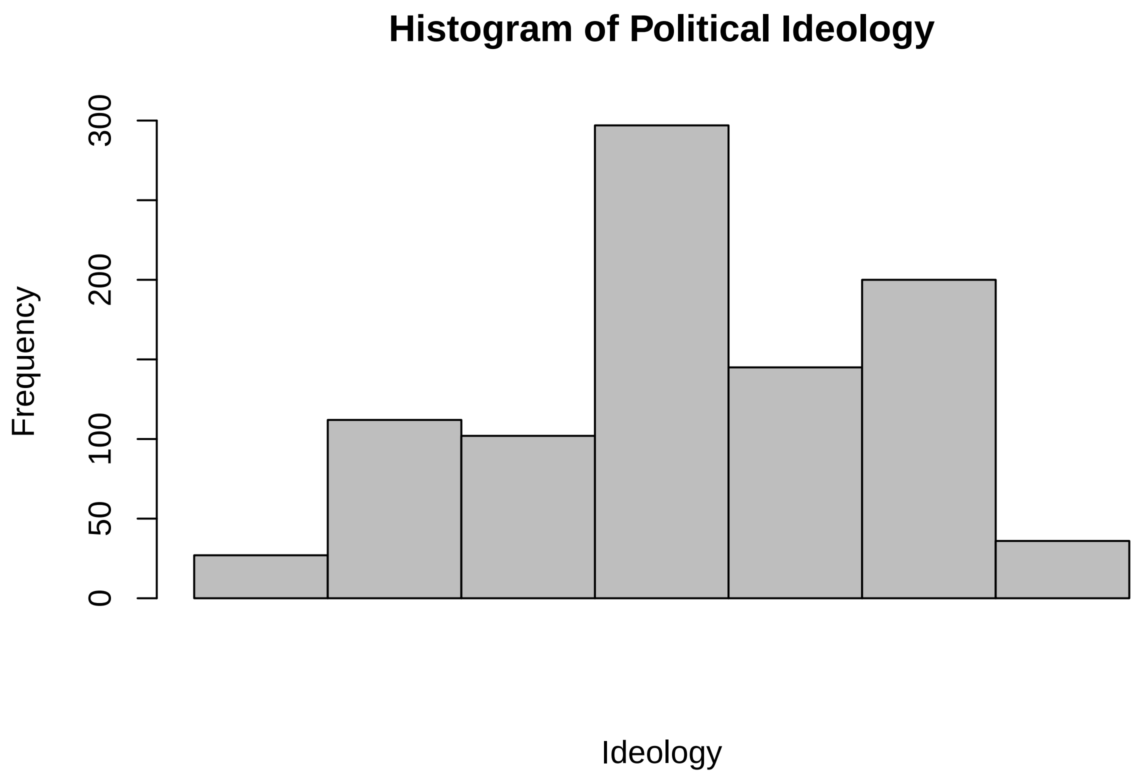
Now, we will need to tackle the messy x-axis tick marks and gaps. We can do this by first turning the x-axis "off", or making it blank using the `xaxt="n"` option:

```
hist(mydata$ideology, main="Histogram of a Discrete Variable", xlab="Ideology", xaxt="n")
```



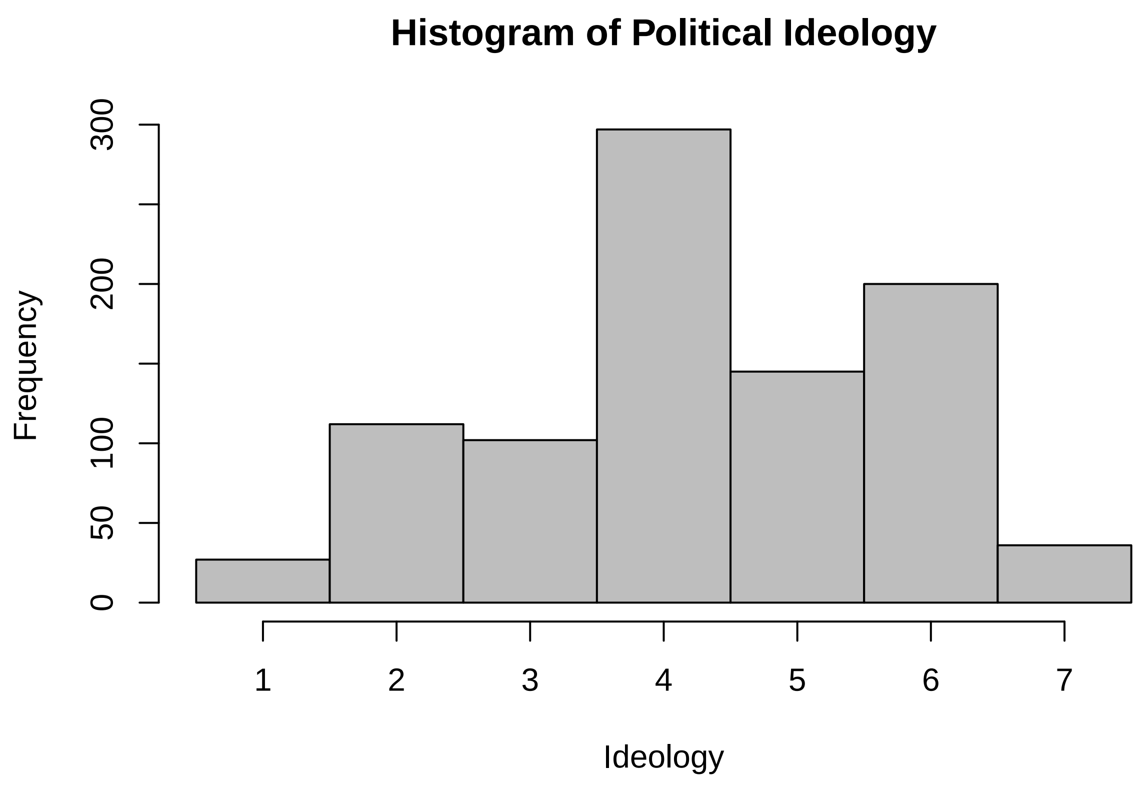
Now we can customize the x-axis to what we want. First, we need to tell R where to put the bars. We do this with the `breaks` option, which tells R exactly where to place each bar. I also changed the color of the bars with the `col=` option.

```r
hist(mydata$ideology, col="grey", main="Histogram of Political Ideology", xlab="Ideology",
     breaks=seq(0,7), xaxt="n")
```



Now we need to relabel the x-axis with the tick marks we removed earlier. This requires a second line of code that starts with the command `axis`. With this command you need to first start by specifying which axis you want to change. In this case we want to change the x axis (`side=1`). Then we need to tell R where to but the tick marks (`at=c()`). Finally, we have to decide which labels to put at each tick mark (`labels=c()`). The following code is an example of this:

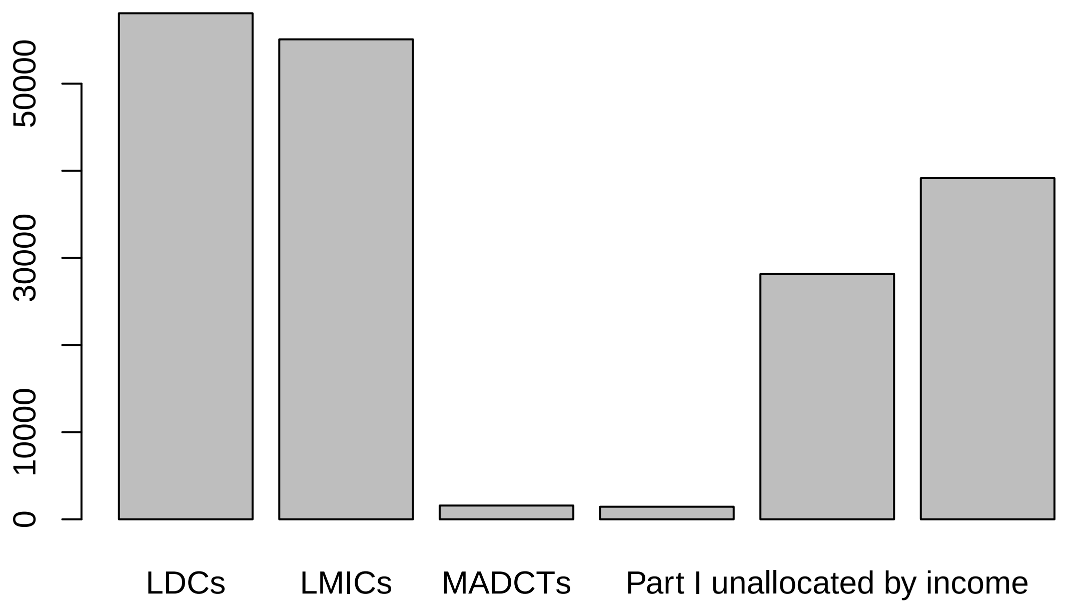
```r
hist(mydata$ideology, col="grey", main="Histogram of Political Ideology",
     xlab="Ideology", breaks=seq(0,7), xaxt="n")

axis(side=1, at=c(.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5),
     labels=c("1", "2", "3", "4", "5", "6", "7"))
```



Barplots

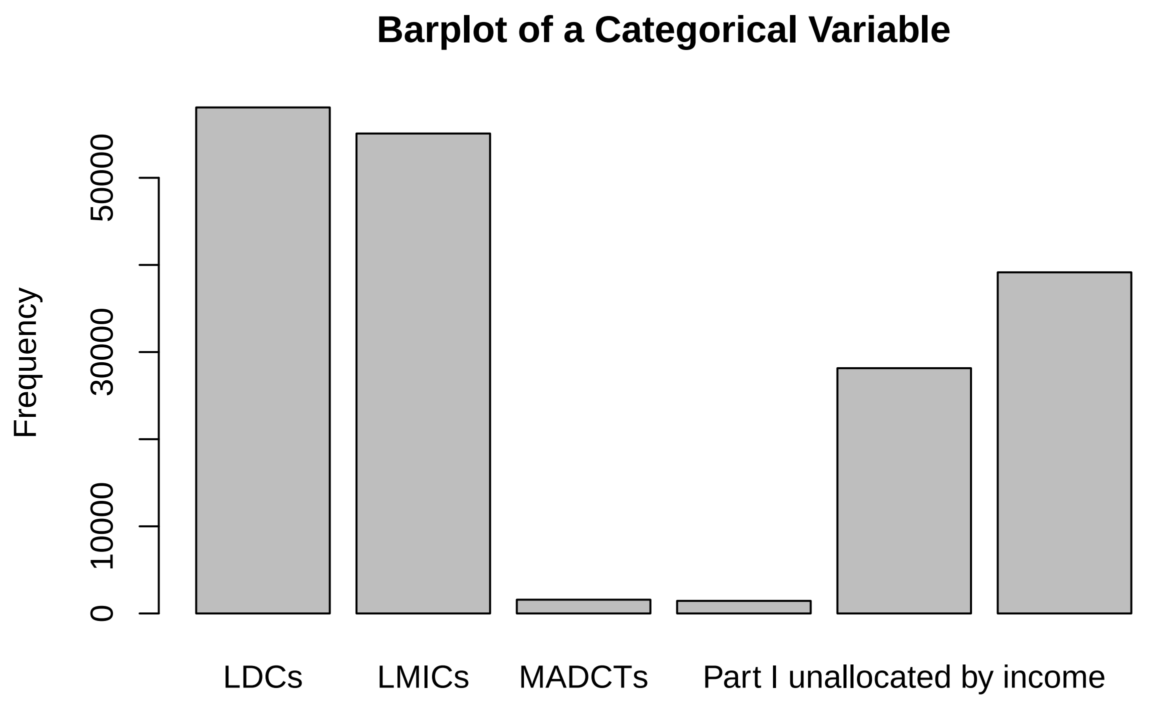
Histograms are only useful when R recognizes that object as "numeric". What happens for variables that are labeled categorical variables, without numeric labels? In this case, you can use the `barplot` command. While the current dataset with which we are working does not contain any of these such variables, they are everywhere in the OECD foreign aid data that we used last time. So we will switch gears and load this data into R:

```
barplot(table(oecd$IncomegroupName))
```



The key to using a barplot effectively is to include the `table()` command, which tallies up the number of observations that have that value of the variable. In the above example, the barplot tells you how many observations in the foreign aid dataset fall under difference income categories (for example, LMICs means Lower Middle Income Countries, etc.). We can make this plot look nicer by adding a title like we have earlier and add a label to the y-axis using the `ylab=` command:

```
barplot(table(oecd$IncomegroupName), main="Barplot of a Categorical Variable", ylab="Frequency")
```



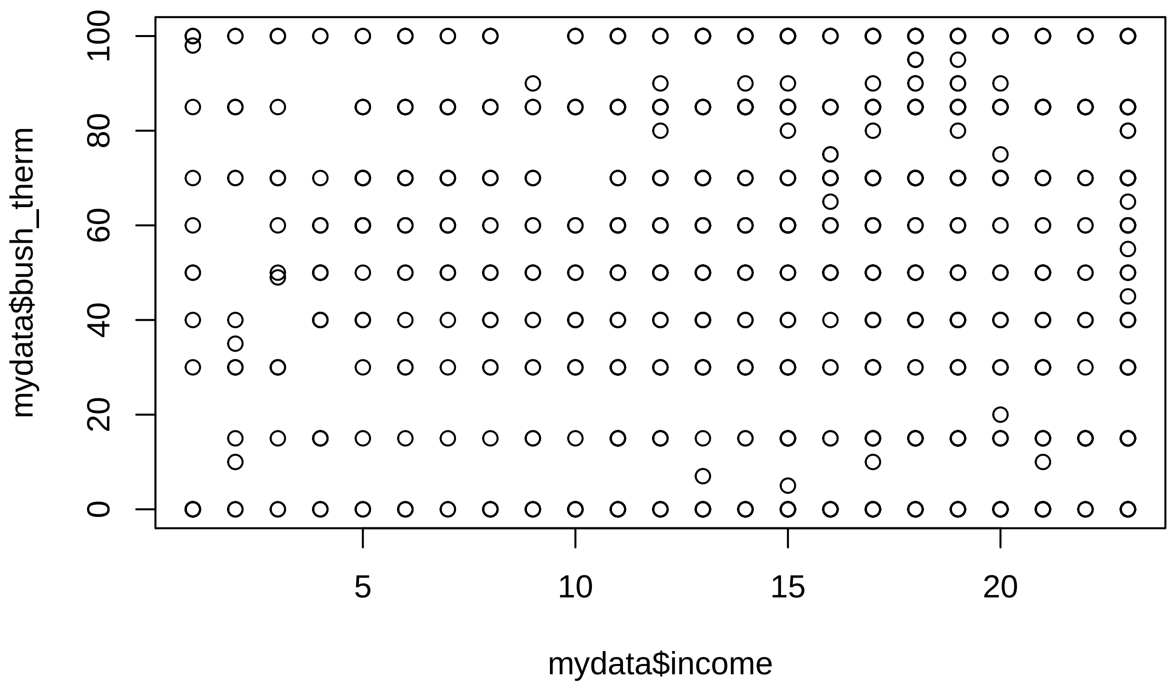
Bivariate Displays

Most of the time, we want to know about the relationship between two variables, not simple the distribution of a single variable. In this case, we would use bivariate plots. R has many functions for these as well. To demonstrate, we are going back to using the American National Election Study survey results.

Scatterplots

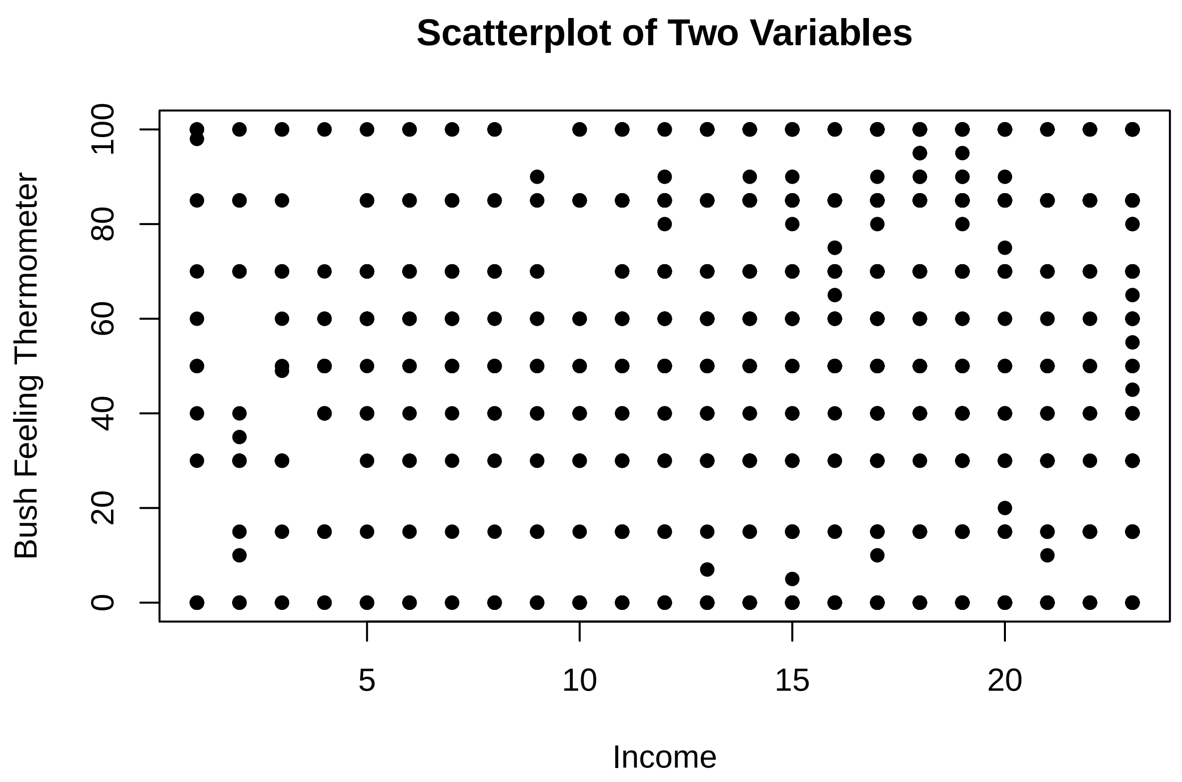
The most basic plot for a bivariate display is a scatterplot. Scatterplots are used to shoe relationships between two continuous variables. The best part about scatterplots is that they are R's default plot. The syntax for scatterplots start with deciding your x and y variable assignments. You need to ask yourself: What am I trying to explain? That is the variable that will be your y variable, or the one that will go on the vertical axis. In our example, the y variable will be the bush_therm variable, or how warm each respondent felt toward President Bush. In this example, my x variable will be the respondent's income.

```r
plot(x=mydata$income, y=mydata$bush_therm)
```



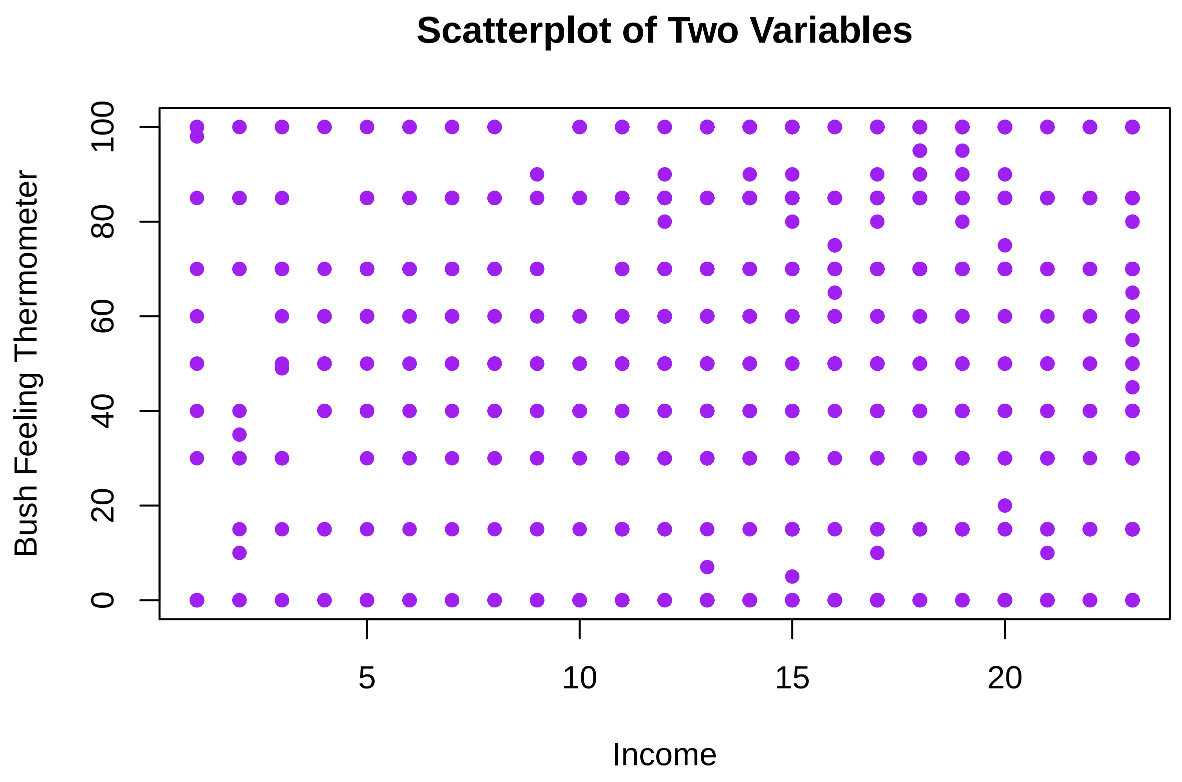
There are many options to make the scatterplots look nicer, the first among these is the same commands we used earlier: `main`, `xlab`, and `ylab`. However, we can also change the shape and fill of the dots by using the `pch` option. The numbers refer to what shape/fill you want for your scatterplot. Feel free to play with the numbers.

```r
plot(x=mydata$income, y=mydata$bush_therm, xlab = "Income", ylab="Bush Feeling Thermometer",
     main="Scatterplot of Two Variables", pch=16)
```



You can even change the color of the dots with the same `col` option discussed earlier:

```r
plot(x=mydata$income, y=mydata$bush_therm, xlab = "Income", ylab="Bush Feeling Thermometer",
     main="Scatterplot of Two Variables", pch=16, col='purple')
```

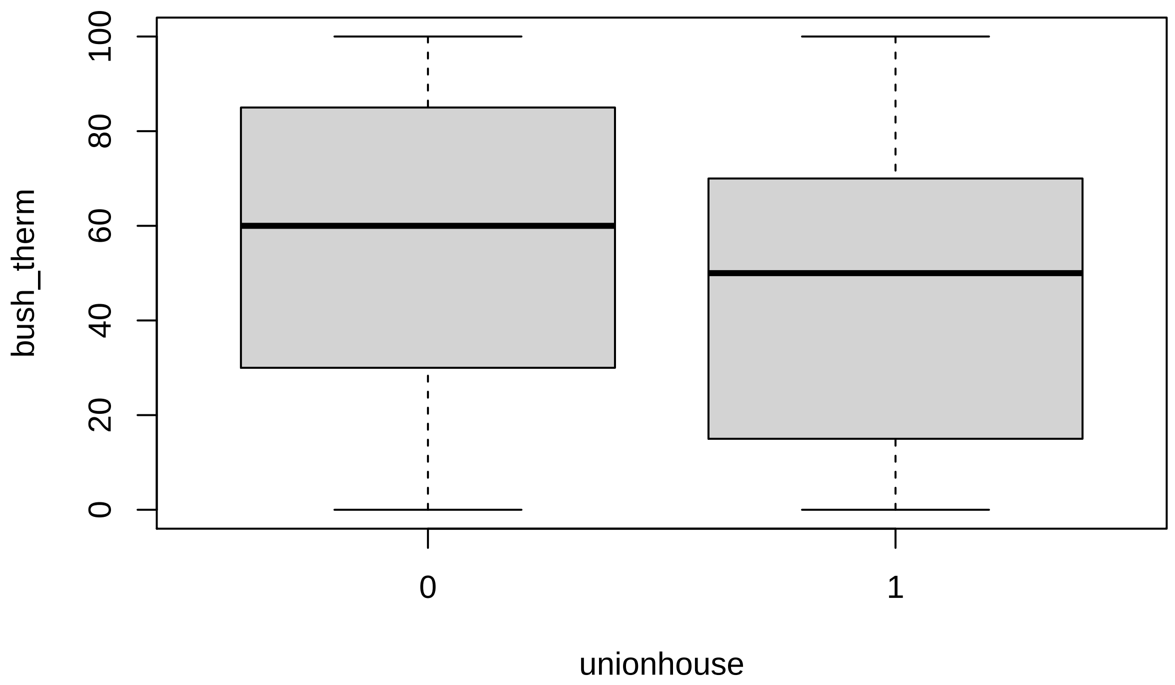


Box and Whisker Plots

What happens if our x variable is not a continuous variable? Well, scatterplots start to look a bit strange. In this case, we would use a Box-and-whisker plot to show a relationship between a continuous dependent variable (y variable) and a categorical x variable (independent variable).

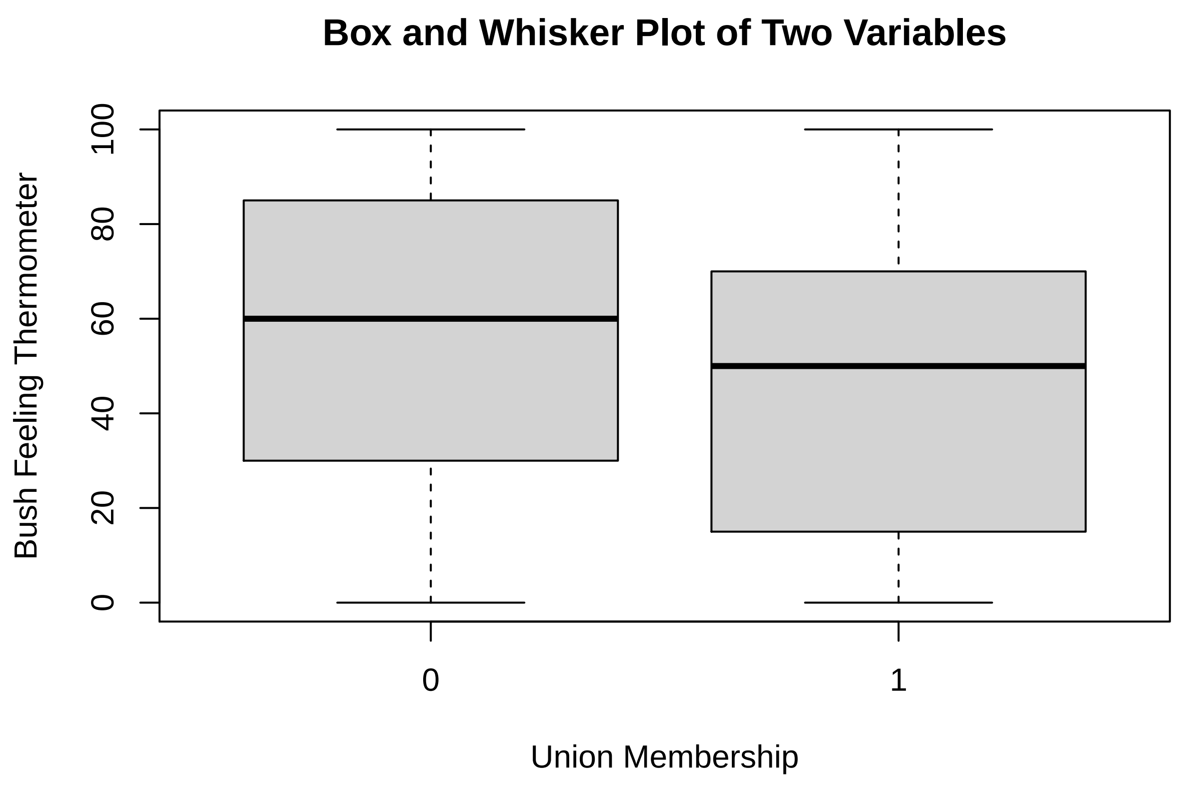
The Box and whisker plots are a bit different in their plot syntax. For these, you need to use the formula notation, mainly the ~ symbol. This is the symbol that lets you know that you want to complete a function on one variable (the one before the ~), conditional on the value of the variable that follows. In this following example, I use the value of the Bush feeling thermometer as my dependent variable and union membership as my independent variable:

```
boxplot(bush_therm~unionhouse, data=mydata)
```



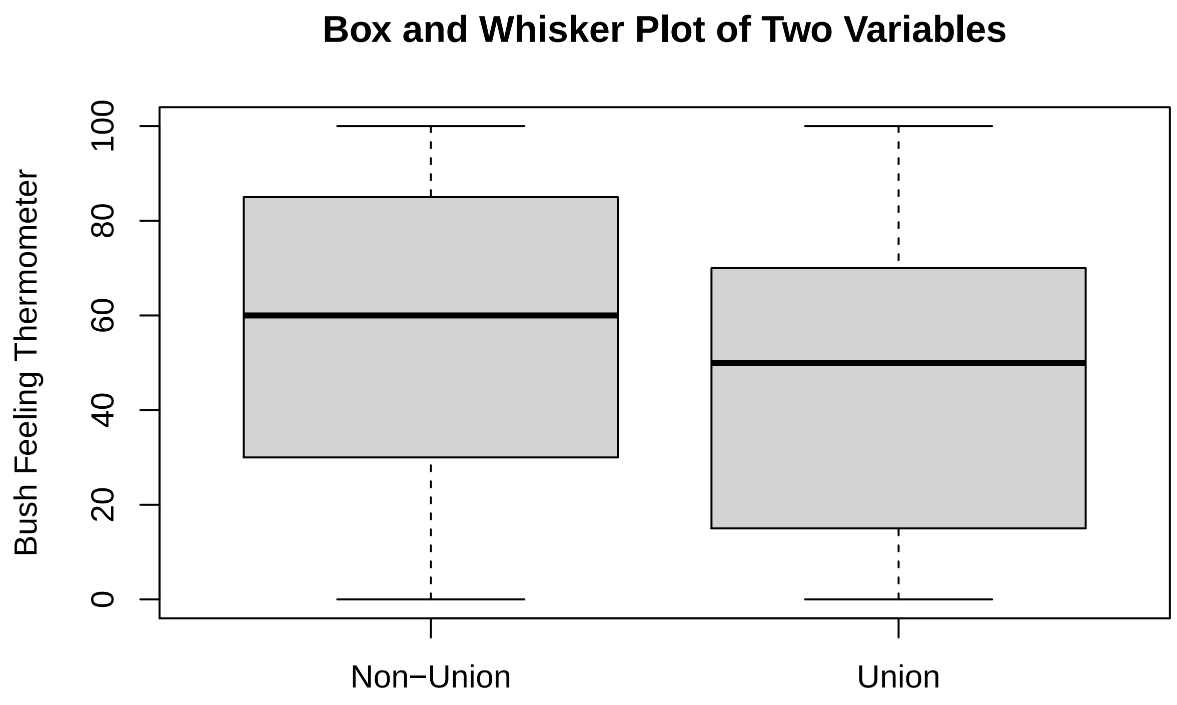
From here, you can use the same options as before in making this plot more attractive:

```
boxplot(bush_therm~unionhouse, data=mydata, xlab = "Union Membership",
        ylab="Bush Feeling Thermometer", main="Box and Whisker Plot of Two Variables")
```



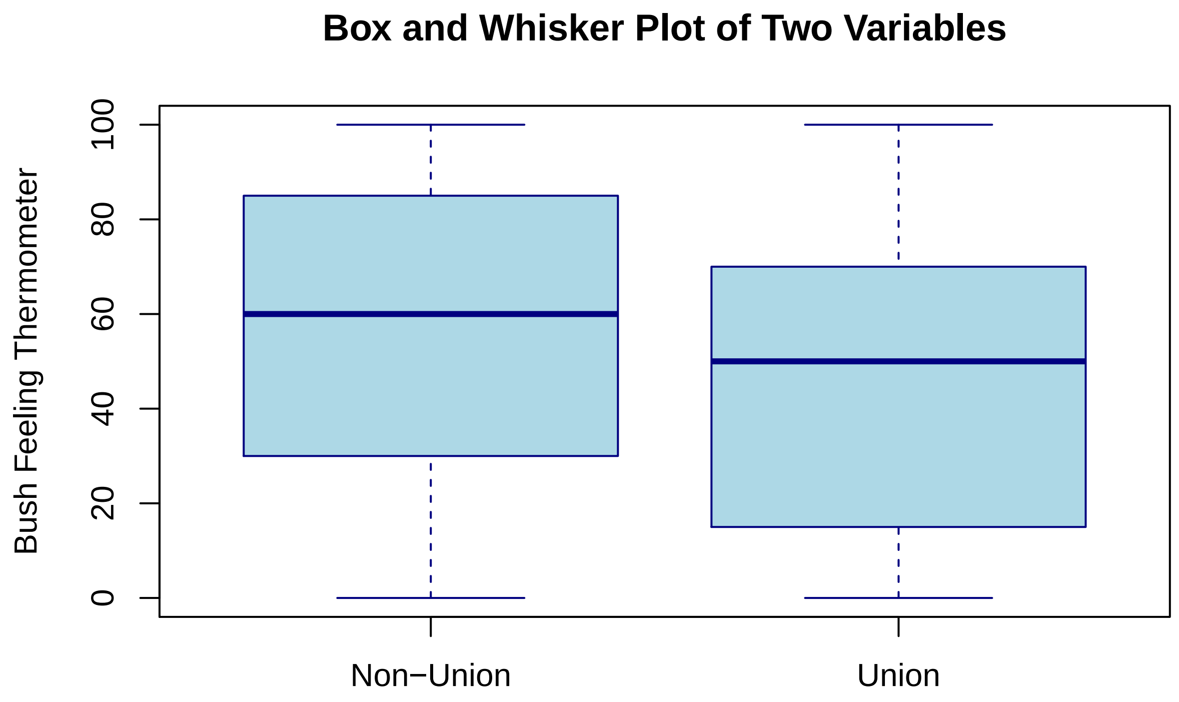
You can additionally change the labels of the tick marks with the `names=c()` option and drop the x-axis label altogether.

```r
boxplot(bush_therm~unionhouse, data=mydata, xlab = "", ylab="Bush Feeling Thermometer",
        main="Box and Whisker Plot of Two Variables", names=c("Non-Union", "Union"))
```



And, for fun, you can change the colors of the plot using the `col` and `border` options:

```r
boxplot(bush_therm~unionhouse, data=mydata, xlab = "", ylab="Bush Feeling Thermometer",
        main="Box and Whisker Plot of Two Variables", names=c("Non-Union", "Union"),
        col="lightblue", border="navy")
```


Box and Whisker Plot of Two Variables
Bush Feeling Thermometer
Non−Union
Union