# Visualizing Data and Assessing Relationships

This exercise uses data on women's hockey from Hockey Abstract collected by Rob Vollman http://www.hockeyabstract.com/testimonials/womenshockeydata. The data have been lightly cleaned by me. A summary of the variables in the data is available below.

| Name | Description |
|---|---|
| `Season` | Year of season |
| `League` | League |
| `First.Name` | Player's first name |
| `Last.Name` | Player's last name |
| `Pos` | Player's position |
| `Team` | Player's team |
| `GP` | Games played |
| `G` | Goals scored |
| `A` | Assists |
| `PTS` | Total points |
| `PIM` | Penalty minutes |
| `PM` | Plus/minus |

This exercise is due on Sakai **at the end of class today**. Don't worry if you don't finish it; you'll receive full credit for submitting it, regardless of how far you make it. This is a group assignment, so **you may work with your neighbors**.

## Question 1

Load the data into R by using the `read.csv()` function. Remember, you can either change your working directory to where the `hockey.csv` file is located with the `setwd()` function, or you can write the full filepath to it in the `read.csv()` function. On my computer that looks like this:

```
setwd('~/Dropbox/UNC/Teaching/281 Spring 2019/In Class Materials/')
hockey <- read.csv('Data/hockey.csv')
```

## Question 2

First off, let's check to make sure there aren't any glaring errors in our data. In hockey, players receive one point each for both goals and assists. Check that every player's point total is equal to their goals plus assists. Remember that a conditional will evaluate to `TRUE` or `FALSE` and R treats them as 1 and 0, so you can use either the `mean()` or `min()` functions to check for any incorrect point totals.

## Question 3

Subset the data to just the 2014 Olympics using the `subset()` function and assign the result to a new object called `oly2014`. To do this, you want to get only rows where `Season` is equal to `2014` and

`League` is equal to `Oly`. Once you've done this, find the top goal scorer(s). You could accomplish this with square bracket indexing by doing the following:

```
oly2014 <- hockey[hockey$Season == 2014 & hockey$League == 'Oly', ]
oly2014[oly2014$G == max(oly2014$G), ]
```

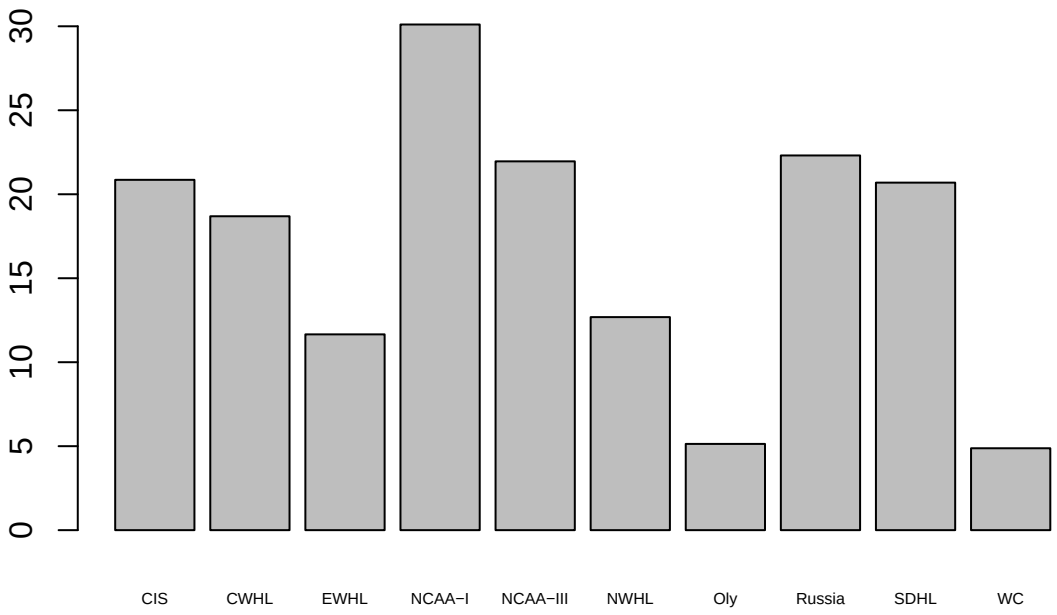
# Question 4

Return to our main `hockey` object. A good way to visualize discrete data is a **bar plot**. Bar plots are a natural fit when the different values in our data don't have a natural ordering to them. Our data contain information on players in the following leagues:

```
##  [1] "CIS"      "CWHL"     "EWHL"     "NCAA-I"   "NCAA-III" "NWHL"
##  [7] "Oly"      "Russia"   "SDHL"     "WC"
```

There's no inherent way to sort these leagues. You *could* try and rank them based on the overall level of play (but that's complicated and there are lots of ways you define it...), or the average salary (but that doesn't work for the World Championships or Olympics...), so instead we treat them as **categorical data**. If you have to think about how to sort your data for more than five seconds, they're probably categorical. This is how to create a bar plot of average games played by league, and what it looks like

```
games_mean <- tapply(X = hockey$GP, INDEX = hockey$League, FUN = mean)
barplot(height = games_mean, cex.names = .5)
```




Notice that I'm using the `tapply()` function here. `tapply()` take three arguments: 1. `X` – a vector of numeric values we want to perform some mathetmatical operation on. 2. `INDEX` – a vector that tells us how we want to split up `X`. You can think of this as an ID variable that tells R which group each element of `X` belongs to. 3. `FUN` – the mathematical operation we want to perform on each group of `X`

Using `tapply()` and `barplot()`, create a bar plot of average goals by league. Hint: the `cex.names = .5` argument above will make sure that all league names are drawn on the bottom of the plot. If

you don't use, some will be omitted because otherwise they would overlap.

## Question 5

We can use `tapply()` to group by any variable we want. Create another bar plot of average goals, but this time group the data by season instead.

## Question 6

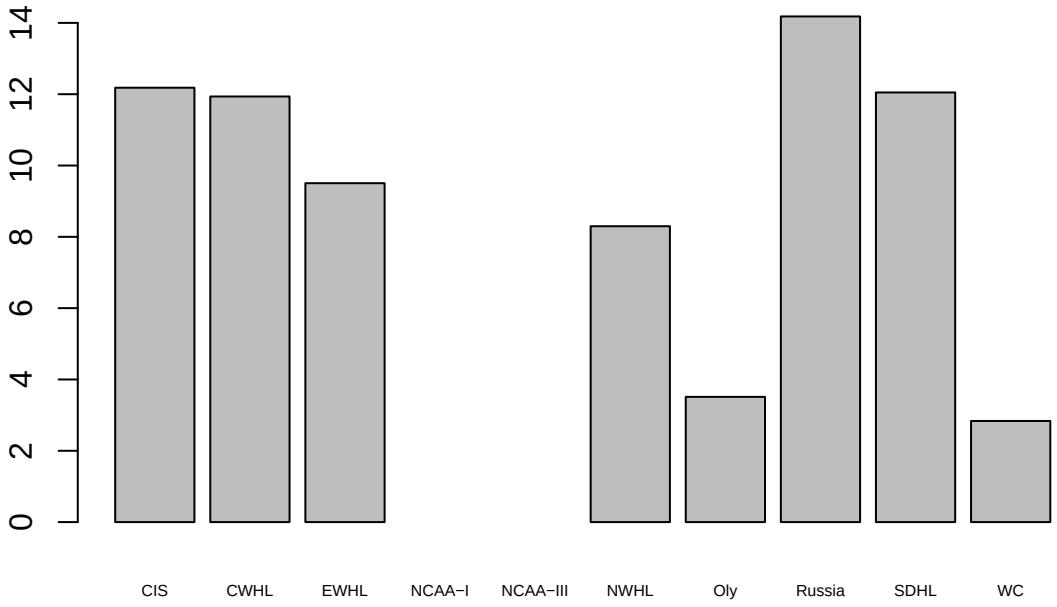
There's a huge jump in average goals from 2000 to 2001, and then a giant dip from 2016 to 2017. Maybe something interesting happened at these points like a rule change? If that were the case, we'd expect everything else to stay pretty much the same. One obvious culprit here could be the fact that number of goals scored is an increasing function of number of games played. Use `tapply()` and `barplot()` to see the average games played in each year.

## Question 7

One last question we should ask is do the 1998-2000 and 2017 seasons have fewer games in the data, or do they have far more players playing fewer games each? Let's create a histogram of the total number of players in each season to compare to our previous histograms. Remember that `tapply()` splits `X` into separate vectors, so use a function that will tell you how many elements are in a vector for `FUN`.

## Question 8

This is what a bar plot of average penalty minutes for each league looks like:




It looks like no players got any penalty minutes in Division I and III college hockey; that can't be right. Run this code to figure out what's going on:

We have **missing data** in our penalty minutes! Missing data occur when some of the data are, unsurprisingly, missing. R uses the special value `NA` to represent missing data. It's important to note that an `NA` for an observation *does not mean that it is zero*. Instead, it just means that it's not in the data. There are many potential reasons for this, and we'll talk more about them and how to deal with missing data later in the course.

For now, we're just going to tell R to *ignore* the missing data. We can do this with the `na.rm = T` argument to the mean function.

```
mean(hockey$PIM)
```

```
## [1] NA
```

```
mean(hockey$PIM, na.rm = T)
```

```
## [1] 12.40496
```

If you type `?tapply` you'll see that there's a fourth argument called `...` we can use. The `...` argument allows you to pass optional arguments to `FUN`, so we can use it to sneak `na.rm = T` into the evaluation of `mean()`. Note, however, that these optional arguments need *their names* and not `...`, so add `na.rm = T` as the last argument in your call to `taply()`

# Question 9

Now let's move onto a **continuous** variable. While `G` is technically a count of goals, one skilled player managed to score 59 goals in a single season, so we're going to treat the goals variable as continuous. Recall from last week that the standard deviation of a variable is root mean square of its deviation from its average:

$$\text{sd}(\mathbf{x}) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Use the `sd()` function to calculate the standard deviation of goals and games played.

# Question 10

The standard deviation of the games played is way higher than goals scored (9.6594568 vs. 4.8936736). Let's figure out why using a **histogram**, which is a convenient way to plot continuous data. This is what a histogram of plus-minus looks like (sidenote for non hockey fans: plus-minus is terrible statistic that tells you very little about a player's skill and you should avoid it all costs if you're making predictions for fantasy hockey):

A histogram puts all the elements of a variable in order, just like quantiles, splits them into equal-width groups, or bins, and then counts up how many data points fall within each bin. The y axis on a histogram tells you how much of your data falls within each bin. We can see here that most players fall relatively close to zero, and about twice as many players have negative plus-minus ratings as have positive ones.

Use the `hist()` function to create histograms of goals scored and games played. What is the relationship between the standard deviation of each variable and its histogram?

# Question 11

Every plot we've made so far has been a **univariate** and every statistic we've calculated has been a univariate statistic. This just means that each one only deals with one random variable. Let's move onto a **bivariate** relationship. Use the `plot()` function to plot games played on the x axis and goals scored on the y axis. This is called a scatterplot. What relationship do you observe between the two variables?

# Bonus

The standard deviation is a numerical counterpart to the histogram. Do you know what the numerical counterpart is to the scatterplot? If you don't know it, take another look at section 6 of chapter 3 in Imai because we'll be talking about it in class on Thursday.