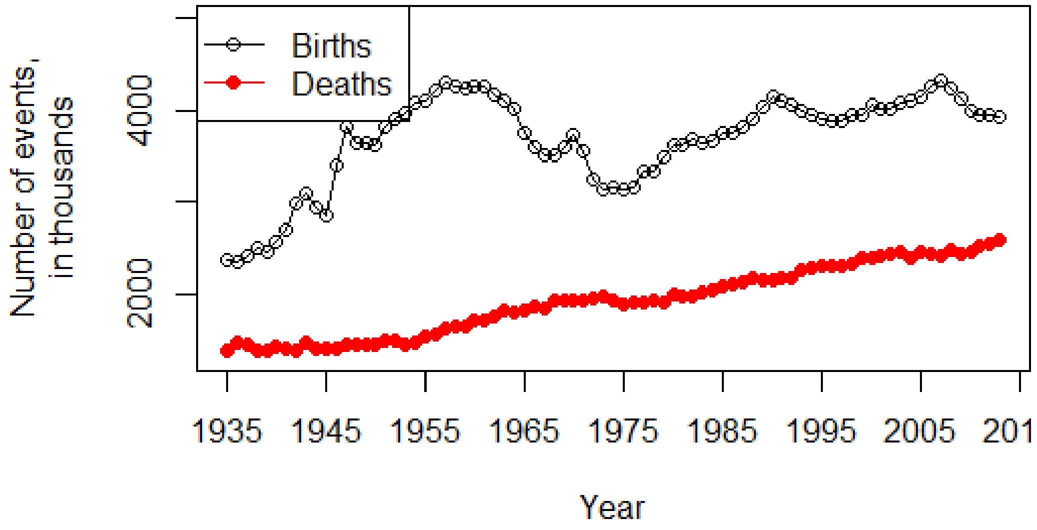
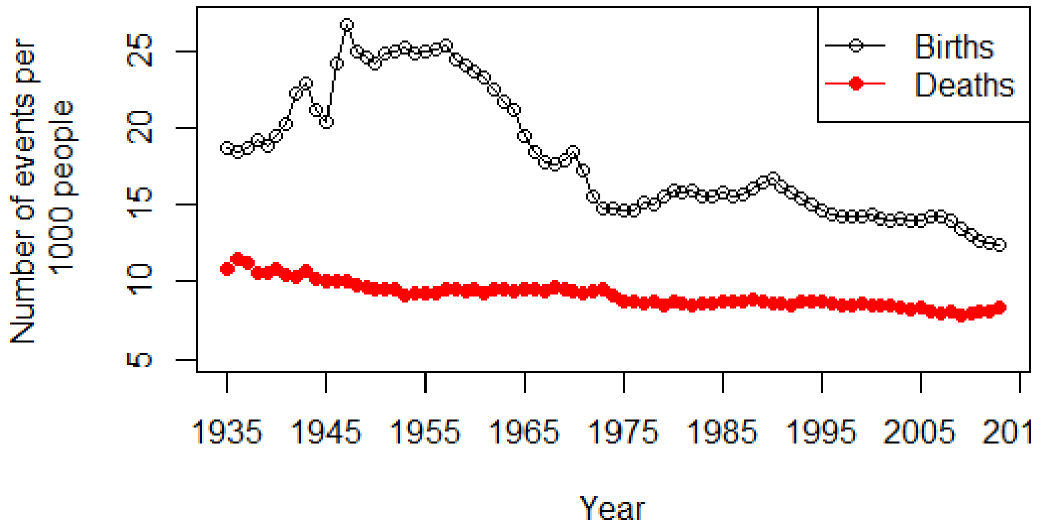
Web-scraping with R

In this handout, we will explore web-scraping with R. For motivation, we will consider several plots. These plots could be used towards the beginning of an introductory statistics course to discuss different ways of thinking about data, and how interpretation of data can change depending on the context in which they are displayed. Not all ways of thinking about data are created equal!

As an example, consider these two plots:

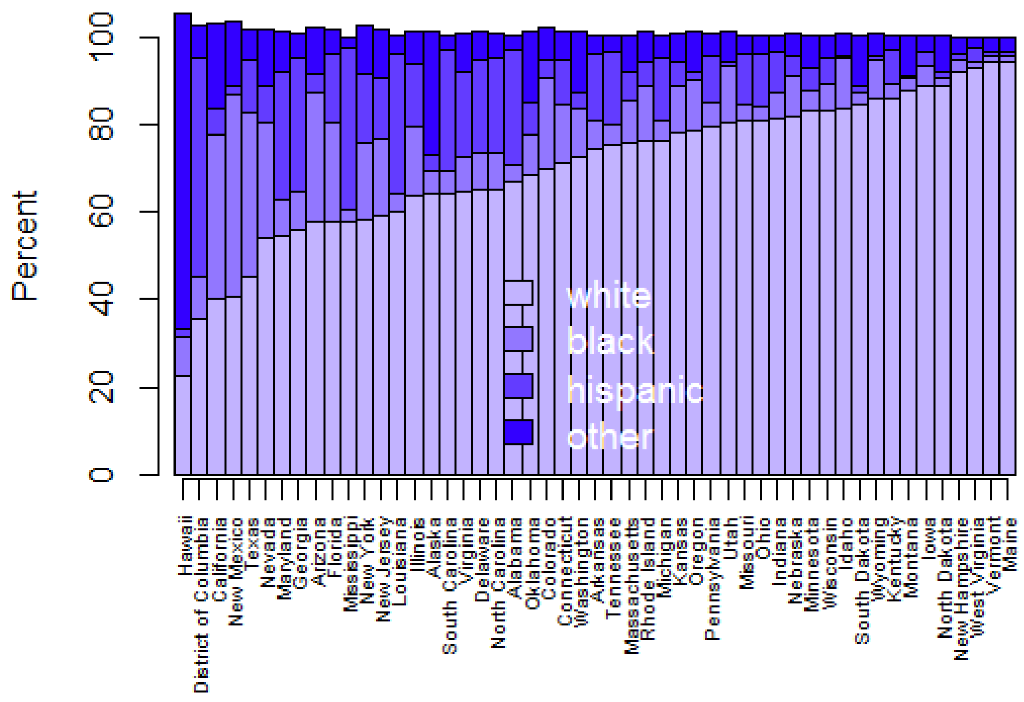




The first plot show births and deaths in the U.S, from 1935 until 2013. The second plot shows birth and death *rates* per 1000 people, over the same time period. There are a number of interesting discussion questions to ask about these plots in an introductory statistics class:

1. Both plots show a peak in birth (or birth rate) between about 1950 and 1965. What do you think this peak represents?
2. One plot shows a positive trend, while the other shows a negative trend. Why is this, and does this imply that these plots contradict each other?
3. Which plot is better, and why?
4. Given your answer to #3, what do we conclude about overall trend in birth and death rates in the U.S.?

A third plot shows a barchart of racial trends across the 50 states and the District of Columbia.



This plot can be used to ask several questions about which states are the most diverse. This would likely raise interesting questions about how to define diversity: is is the state with the lowest non-white population? The state with the most even race distribution? One would also definitely want to ask students if they can identify any strange things about this plot! (The percentages add to more than 100%; we will investigate this later.)

While these plots themselves might seem relatively simple to interpret, they are far from simple to create! These data were taken from the following Wikipedia page: http://en.wikipedia.org/wiki/Demographics_of_the_United_States. Scroll down to find the table called "Vital Statistics" and the race "Breakdown by State." These are the data tables we will be using to create the above plots. To scrape these tables, we will need to:

1. Scrape the data into R;
2. Find the relevant data tables;
3. Clean the tables so they are in a format for plotting;
4. Plot the data.

Along the way, we will cover some intermediate `R` topics such as:

1. Object classes;
2. String manipulation;
3. Intermediate graphics;
4. The three dimensions of color.

Web-scraping HTML tables is best done in R using the `XML` package. If you haven't installed this package before, install it and load it using the following commands:

```
install.packages('XML')
library(XML)
```

Once the `XML` package is installed, we will read in the Wikipedia page with the following command, and investigate a few elements:

```
all <- readHTMLTable('http://en.wikipedia.org/wiki/Demographics_of_the_United_States')
class(all)

## [1] "list"

length(all)

## [1] 39
```

Note that `all` is a list, where each element of the list is one of the HTML tables on the Wikipedia page. Not all of them are pretty, and only some are named. Run the following commands to investigate the `all` object:

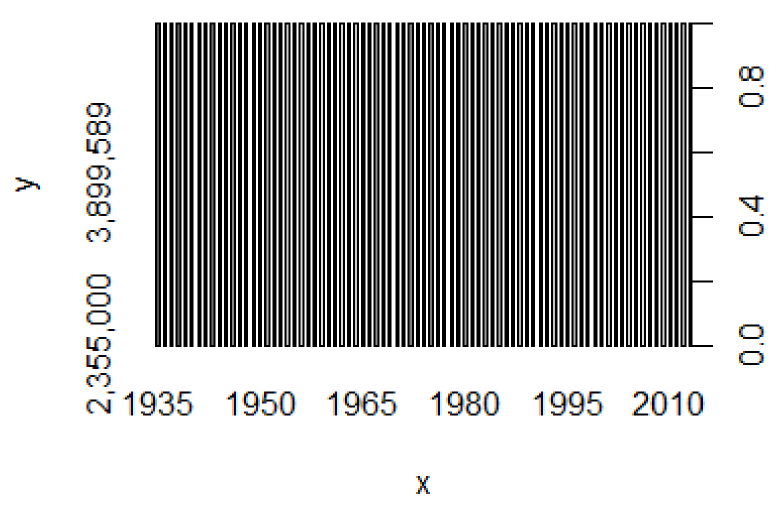
```
names(all)
all[[1]]
all[[5]]
all[[6]]
all[[7]]
```

Note that the first element of `all` is the table titled "Demographics of the United States"; it doesn't look like your typical $n \times p$ data format! The data that we want are in the 5th and 7th elements of `all`. It's best to rename the elements we want, so we can refer to them more easily, and investigate what we've created. We'll first start with the vital statistics data. Run the following commands:

```
bd <- all[[5]]
head(bd)
names(bd)
#Note that the "year" column doesn't have a name: let's change that
names(bd)[1] <- 'Year'
```

Now that we have the data, let's try plotting birth live births by year. This seems like it should be simple enough, but:

```
plot(bd$Year,bd$Live)
```



Yikes! This plot is meaningless. Although R has successfully scraped the relevant HTML table, there are several issues with the resulting data frame, `bd`. Run the following lines of code, and see if you can identify some of the issues.

```
class(bd[,1])

## [1] "factor"

for (i in 1:ncol(bd)) print(class(bd[,i]))

## [1] "factor"
## [1] "factor"
## [1] "factor"
## [1] "factor"
## [1] "factor"
## [1] "factor"
## [1] "factor"
## [1] "factor"
## [1] "factor"
```

Note that `R` is thinking of each of these variables as a "factor", which as we've seen is `R`-speak for "categorical". We need to change these variables to numeric. We can do this easily enough for Year. Run these lines of code and discover what happens.

```
as.vector(bd$Year)
as.numeric(as.vector(bd$Year))
class(as.numeric(as.vector(bd$Year)))
```

But what if we try this on "Live Births"? Run this line of code. What is the reason for the resulting output?

```
as.numeric(as.vector(bd$Live))
```

`R` is not a fan of the commas in the "Live births" variable. It would be nice if we had a customized function that eliminated commas (if present), and changed the resulting variable to numeric. The following function is one way to accomplish this:

```
#Get rid of commas (if present), change to numeric
cleanup <- function(vec) {
  new <- gsub(',','',vec) #This searches for and eliminates any commas
  new <- as.numeric(as.vector(new)) #Change the vector to class "numeric"
    return(new)
}
```

Aside: function writing

After running the above code, R now knows about a new function that takes a vector as its single argument, removes any commas, and changes the class to numeric. This is the first time we've written our own function, so let's take some time to examine it. The basic form of function writing is as follows:

```
function.name <- function(arg1, arg2, ...){
  Operations on arguments
  output <- result of operations
  return(output)
}
```

Our `cleanup()` function only has one argument, called `vec`. In our case, we expect to feed `cleanup` a vector and perform operations on that vector, specifically, remove commas and change the class to numeric. The cleaned up vector we call `new`, and we need to make sure to return `new` otherwise the function will carry out the operations, but not give us the result!

For example, suppose we had written a function `cleanup2`, that is identical to `cleanup` except for the `return()` command. Run the following code and see what happens.

```
#Get rid of commas (if present), change to numeric
cleanup2 <- function(vec) {
  new <- gsub(',','',vec) #This searches for and eliminates any commas
  new <- as.numeric(as.vector(new)) #Change the vector to class "numeric"
}

#Create a toy vector to feed to our function:
test.vec <- c('1,000','2,000','3,000')
cleanup2(test.vec) #No output!
cleanup(test.vec)
```

Mini exercise 1:

Write a function called `area()` that computes the area of a circle. The `area()` function takes a radius as its sole argument, and returns the area. Use this function to calculate the areas of circles of radius 3, 5, and 9.

Mini exercise 2:

Write a function called `fullname()` that takes 2 arguments, called `firstname` and `lastname`, each of which is a vector. The output should give the full name of the people fed to the function. Hint: use the `paste()` function within your function.

Exercise

Now back on task. Use a combination of `apply()` and `cleanup()` to clean the vital statistics data. Assign the cleaned data to a new object, `bd2`.

Key to exercise

```
bd2 <- apply(bd,2,cleanup)
bd2 <- data.frame(bd2)
head(bd2$Year)
class(bd2$Year) #Success!
head(bd2$Live)
class(bd2$Live) #Success again!
```

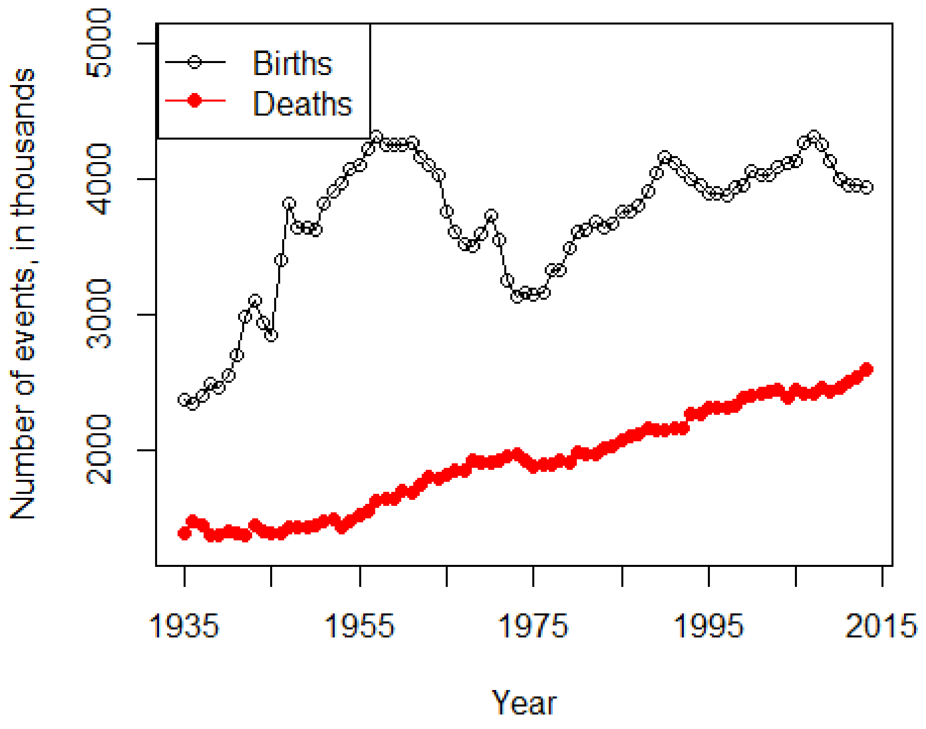
Now we are in business! The data is clean and ready for plotting. So, let's plot!

Exercise

Create the two scatter plots shown at the beginning of this handout.

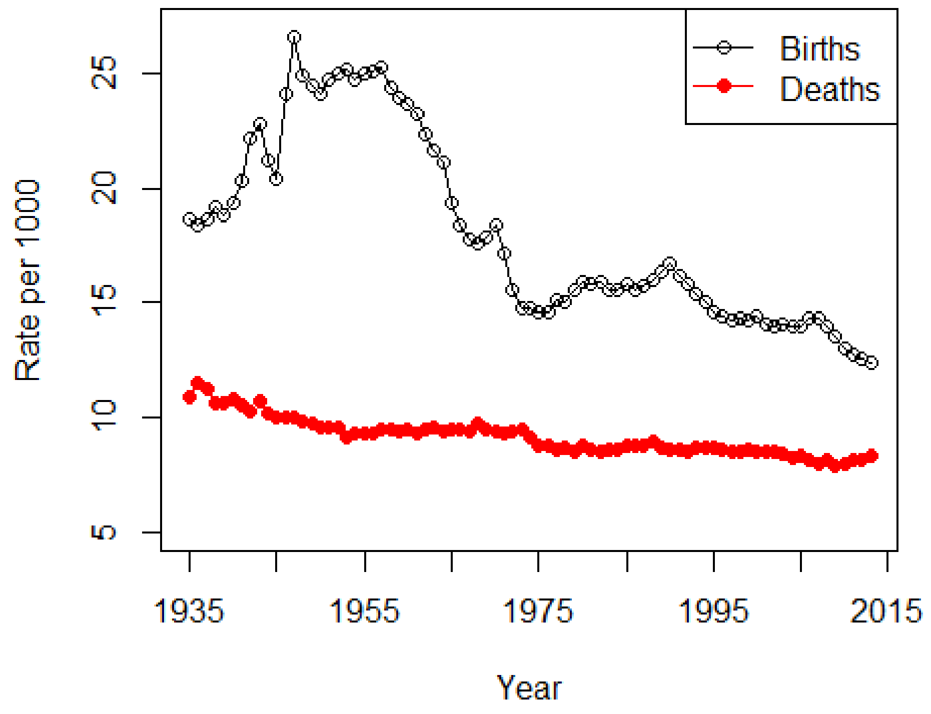
Key to exercise

```
#Divide births and deaths by 1000 for cleaner output:
bd2[,3:4] <- bd2[,3:4]/1000
par(mar=c(5,5,1,1)) #Set the margins of the figure
plot(bd2$Year,bd2$Live,type='o',
                ylab='Number of events, in thousands',
                    xlab='Year',
                    ylim=c(1300,5000),
                    xaxt='n')
        #type: try 'n', 'l', 'h', 'p'
        #xaxt: try NULL
lines(bd2$Year,bd2$Deaths,type='o',col='red',pch=19)
      #pch: type ?pch for more
year.axis <- seq(1935,2015,by=10)
axis(1,at=year.axis,labels=year.axis)


legend('topleft',c('Births','Deaths'),
            col=c('black','red'),
            pch=c(1,19),
            lty=1)
```



The birth rates per 1000 are plotted as follows:

```
par(mar=c(5,5,1,1)) #Set the margins of the figure

##Now plot birth rate
plot(bd2$Year,bd2$Crude.birth,type='o',
                ylab='Rate per 1000',
                    xlab='Year',
                    ylim=c(5,27),
                    xaxt='n')
lines(bd2$Year,bd2$Crude.death,type='o',col='red',pch=19)
axis(1,at=year.axis,labels=year.axis)

legend('topright',c('Births','Deaths'),
            col=c('black','red'),
            pch=c(1,19),
            lty=1)
```



Now for the the racial rates by state. Recall that the racial data was in the 7th element of the list `all`. We'll call it `races`:

```
races <- all[[7]]
head(races)
for (i in 1:ncol(races)) print(class(races[,i]))
```

Exercise

Write necessary code to clean the `races` data so it is ready for plotting. Assign the cleaned data to an object called `newrace`.

Key to exercise

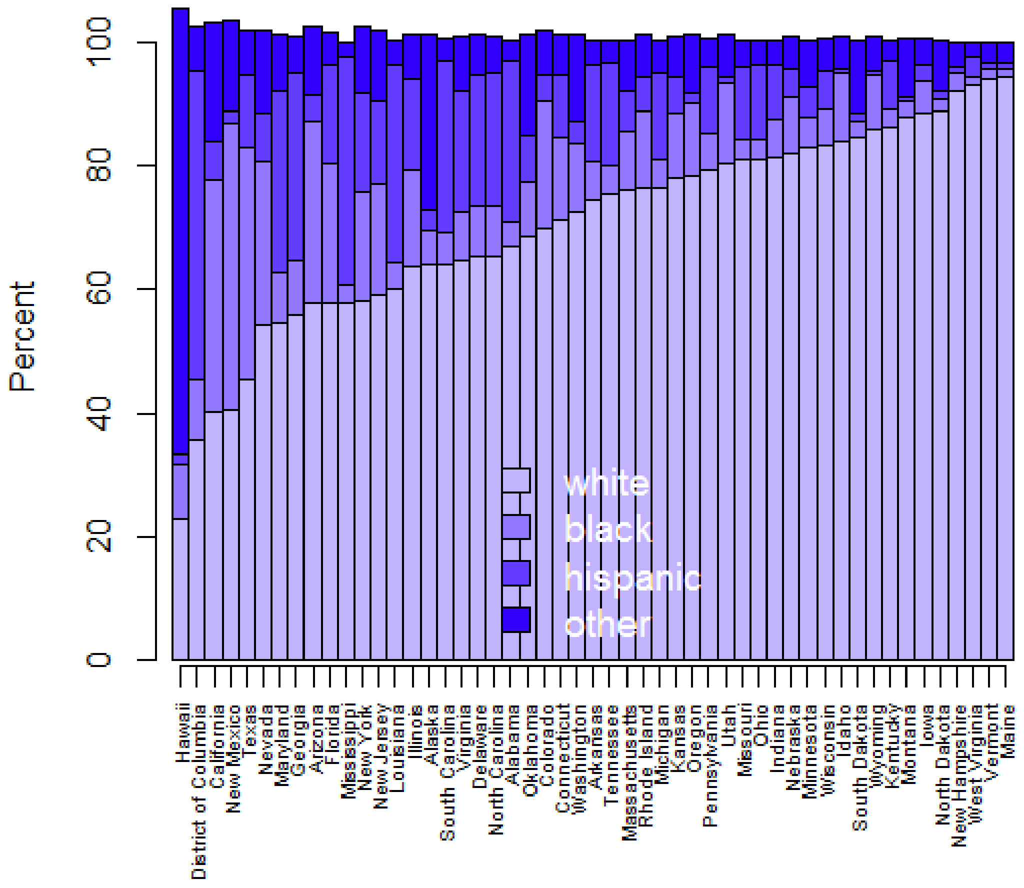
```
newrace<- data.frame(apply(races,2,cleanup))
head(newrace)
#Why do we need this line?
newrace[,1] <- races[,1]

## Warning in FUN(newX[, i], ...): NAs introduced by coercion
```

Exercise

Create the bar graph shown at the beginning of the handout.

Key to exercise

```
newrace<- newrace[order(newrace$Non.Hispanic),]
newrace$Other <- apply(newrace[,6:9],1,sum)
par(mar=c(7,4,1,1))
cols <- hsv(h=0.7,s=seq(0.3,1,l=4),v=1)
barplot(t(as.matrix(newrace[,c(3,4,5,10)])),
  space=0,
  xlim=c(1,50),
    col=cols,
    names.arg=rep('',51),
    ylab='Percent')
axis(rep(1,51),seq(.5,50.5,by=1),newrace$State,cex.axis=.65,tick=T,las=2)
legend('bottom',c('white','black','hispanic','other'),fill=cols,
            bty='n',text.col='white',cex=1.2)
```



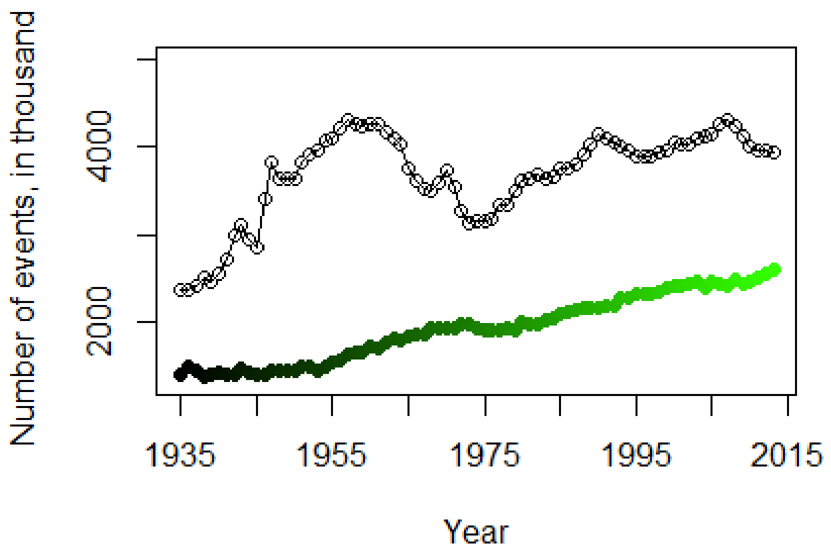
Tasks/Questions:
1. Change the values in the `par(mar=c())` setting, and see what changes.
2. Play around with the `space` argument. What settings do you think look best?
3. What does the `names.arg` argument control? Why do we set it to `rep('',51)`?
4.How did we know where to put the horizontal axis tick marks? (Hint: try re-running the `barplot()` command, with the argument `plot=FALSE`.)
5. What does the `las` argument control? Run `?par` and search for `las`. Try `las = 0`, `las = 1`, etc.

6. Note that the colors are set by way of the very powerful, but simple, `hsv()` function. HSV stands for the three dimensions of color: hue, saturation, and volume. These three dimensions are on a scale of 0 to 1; the `hsv()` function returns the appropriate color in hexadecimal form. To investigate the three-dimensional color space, run the following function and play around with setting `h` while keeping `s` and `v` fixed; setting `s` while keeping `h` and `v` fixed; and setting `v` while keeping `h` and `s` fixed.

```
plot.hsv <- function(h=NULL,s=NULL,v=NULL) {
  x <- seq(0,1,l=100)
    y <- seq(0,1,l=100)
    if(!is.null(h)) {
        plot(x,y,type='n',xlab='Saturation',ylab='Value',main=paste('Hue = ',h))
        for(i in 1:100) points(rep(x[i],100),y,col=hsv(h,x[i],y),pch=15)
    }
    if(!is.null(s)) {
        plot(x,y,type='n',xlab='Hue',ylab='Value',main=paste('Saturation = ',s))
        for(i in 1:100) points(rep(x[i],100),y,col=hsv(x[i],s,y),pch=15)
    }
    if(!is.null(v)) {
        plot(x,y,type='n',xlab='Hue',ylab='Saturation',main=paste('Value = ',v))
        for(i in 1:100) points(rep(x[i],100),y,col=hsv(x[i],y,v),pch=15)
    }
}

plot.hsv(v=1)
plot.hsv(h=0.8)
plot.hsv(s=1)
```

`hsv()` is especially nice for creating color gradients. For example, here's a redo of the vital statistics plot:

```
par(mar=c(5,5,1,1))
plot(bd2$Year,bd2$Live,type='o',
                ylab='Number of events, in thousands',
                    xlab='Year',
                    ylim=c(1300,5000),
                    xaxt='n')
lines(bd2$Year,bd2$Deaths,type='o',col=hsv(h=0.3,s=1,v=seq(0,1,length=nrow(bd))),pch=19)
year.axis <- seq(1935,2015,by=10)
axis(1,at=year.axis,labels=year.axis)
```



Finally, recall that the percents in the bar chart do not add up to 100%, but instead exceed 100%. The reason for this can be uncovered by noticing the footnote of the racial information table takes us to the U.S. Census Bureau's "Quick Facts" page. Doing some investigation on the Quick Facts website, we find the note that "Hispanics may be of any race, so also are included in applicable race categories." This seems to imply that people in the Hispanic/Latino category are probably being double-counted, and may belong to another racial category as well.