Lecture 2: Exploratory Data Analysis with R

Last Time:
1. Introduction: "Why use R?" / Syllabus
2. R as calculator
3. Data structures
4. Getting help, adding packages

Questions?

Trivia: When/where was the oldest surviving population census?
http://www.jstor.org/discover/10.2307/2172247?uid=3739576&uid=2129&uid=2&uid=70&uid=4&uid=373925 6&sid=21104175689307

Outline for today:
1. Summary statistics
2. Graphics in R (*very* powerful feature)

Objectives:
By the end of this session students will be able to:
1. Review basic descriptive statistics in R
2. Create summary statistics for a single group and for subgroups
3. Generate graphical displays of data: histograms, empirical cumulative distribution, QQ-plots, box plots, bar plots, and dot charts

2.1 Summary statistics

Quick Recap: what are common types of data? Categorical (Binary as a special case), Ordinal, Continuous, Time-to-Event. Examples?

Types of summary statistics?

We have already seen how to compute simple summary statistics in R—let's try out some of these procedures on the airquality dataset inbuilt in R. To recall the structure of the dataset, print out the first 5 rows:

```
>  airquality[1:5,]
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
```

(alternatively, use the `head()` statement)

```
> mean(airquality$Temp)
[1] 77.88235

> mean(airquality[,4])
[1] 77.88235

> mean(airquality[,"Temp"])
[1] 77.88235
```

```
> median(airquality$Temp)
[1] 79

> var(airquality$Wind)
[1] 12.41154
```

It sometimes becomes onerous to keep using the "$" sign to point to variables in a dataset. If we are dealing with a single dataset, we can use the `attach()` command to keep the dataset variables in R working memory. We can then just invoke the variables by name without specifying the dataset. For example:

```
> attach(airquality)

> var(Wind)
[1] 12.41154
```

Once we are finished working with these data, we can use the `detach()` command to remove this data set from the working memory.

Caution: Make sure you never attach two data sets that share the same variable names- this could lead to confusion and errors! A good idea is to detach ***a data set as soon as you have finished working with it.***

For now, let us keep this data set attached, while we test out some other plotting functions.

If you want to numerically look at how your data is falling, you can use the `quantile()` function to get a layout of the data.

By default you get the minimum, the maximum, and the three *quar*tiles — the 0.25, 0.50, and 0.75 quantiles. The difference between the first and third quartiles is called the *interquartile range* (IQR) and is sometimes used as an alternative to the standard deviation.

```
> quantile(airquality$Temp)

  0%  25%  50%  75% 100% 
  56   72   79   85   97
```

It is also possible to obtain other quantiles; this is done by adding an argument to the function containing the desired percentage cut points. To get the *deciles*, do

```
> pvec <- seq(0,1,0.1) #sequence of digits from 0 to 1, by 0.1

> pvec
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0


> quantile(airquality$Temp, probs=pvec)
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100% 
56.0 64.2 69.0 74.0 76.8 79.0 81.0 83.0 86.0 90.0 97.0
```

How would we do this for *quintiles*?

We can also get summary statistics for multiple columns at once, using the `apply()` command. `apply()` is **EXTREMELY** useful, as are its cousins `tapply()` and `lapply()` (more on these functions later).

Let us first try to find the column means using the `apply` command:

```
> apply(airquality, 2, mean) #do for multiple columns at once

Error in FUN(newX[, i], ...) : missing observations in cov/cor
```

We (may or may not, depending on the version of R) get an error, as the data contains missing observations! R will not skip missing values unless explicitly requested to do so. You can give the `na.rm=TRUE` argument (***n***ot ***a***vailable, ***r***e***m***ove) to request that missing values be removed:

```
> apply(airquality, 2, mean, na.rm=T)
     Ozone    Solar.R       Wind       Temp      Month        Day
 42.129310 185.931507   9.957516  77.882353   6.993464  15.803922
```

The `na.rm=TRUE` option can be used even when calculating a statistic for a single column:

```
> mean(airquality$Ozone, na.rm=T)
[1] 42.12931
```

There is also a `summary` function that gives a number of statistics on a numeric variable (or even the whole data frame!) in a nice vector format:

```
> summary(airquality$Ozone)  #note we don't need "na.rm" here
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00   18.00   31.50   42.13   63.25  168.00   37.00
```

```
> summary(airquality)
     Ozone           Solar.R           Wind             Temp           Month            Day
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000   Min.   : 1.00
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000   1st Qu.: 8.00
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000   Median :16.00
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993   Mean   :15.80
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000   3rd Qu.:23.00
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000   Max.   :31.00
 NA's   : 37.00   NA's   :  7.0
```

Notice that "`Month`" and "`Day`" are coded as numeric variables even though they are clearly categorical. This can be mended as follows, e.g.:

```
> airquality$Month = factor(airquality$Month)
```

```
> summary(airquality)
     Ozone             Solar.R            Wind              Temp         Month      Day       
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   5:31   Min.   : 1.00  
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   6:30   1st Qu.: 8.00  
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   7:31   Median :16.00  
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   8:31   Mean   :15.80  
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   9:30   3rd Qu.:23.00  
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00          Max.   :31.00  
 NA's   : 37.00   NA's   :  7.0                                                          
```

Notice how the display changes for the factor variable Month.

Next lecture we'll work on programming so that we can make the *Month* variable be called "May", "June", etc. instead of a number.

Exercise 1. Find the standard deviations (SDs) of all the numeric variables in the *airquality* dataset using the `apply()` function.

(Don't know which command does this? Guess! Or look at the "See Also" section at the bottom of the help page for the `var()` function.)

By the way, how are variance and standard deviation related??? How else can you find the standard deviation of a vector?

For more on the `apply()` function and its variants, see:
http://www.r-bloggers.com/using-apply-sapply-lapply-in-r/
https://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/

Brief Aside

A couple of notes on the mean: one way to think of the mean is as a sum of weighted values. The probability theory definition of a mean is the sum of the value times the probability of that value.

$$\mu = \sum [\, x * p(x) \,]$$

If we assume that each value is equally likely, we get that the sample mean is

$$\mu = \sum [\, x * 1/n] \;=\; 1/n * \sum x$$

An example of the "trimmed" mean:

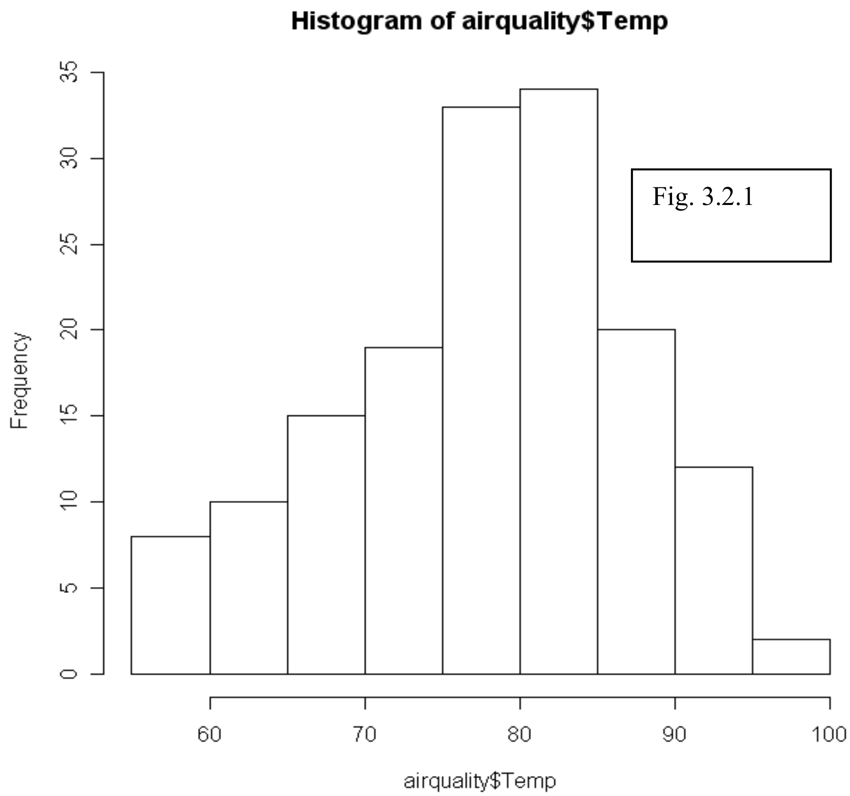
```
> sample.values<-c(-10:10,1000)
> sample.values
[1]  -10   -9   -8   -7   -6   -5   -4   -3    -2    -1    0    1    2   3
[15]     4     5     6     7     8     9    10   1000
> summary(sample.values)
   Min. 1st Qu.   Median     Mean 3rd Qu.     Max.
 -10.00   -4.75     0.50    45.45     5.75 1000.00

> mean(sample.values, trim=0.05)
[1] 0.5
```

2.2 Graphical displays of data

2.2.1 Histograms.

Good reference website for generating plots in R:
http://www.statmethods.net/graphs/creating.html



The simplest display for the shape of a distribution of data can be done using a histogram—a stacked *count* of how many observations fall within specified divisions ("bins") of the x-axis.

```
> hist(airquality$Temp)
```

R usually chooses a sensible number of classes (bins) by default, but a recommendation can be given with the `nclass` (number of classes) or `breaks` argument.

```
> hist(airquality$Temp, breaks = 20)
```

By choosing breaks as a vector rather than a number, you can have control over the interval divisions.

By default, R plots the *frequencies* (counts) in the histogram; if you would rather plot the *relative frequencies*, you need to use the argument `prob=TRUE`.   What happens to the vertical axis?

```
> hist(airquality$Temp, prob=T)
```

There are a LOT of options to spruce this up. Here is code for a much nicer histogram :

```
> hist(airquality$Temp, prob=T, main="Temperature")

> points(density(airquality$Temp),type="l",col="blue")

> rug(airquality$Temp,col="red")  #what do you think this does ? …
```

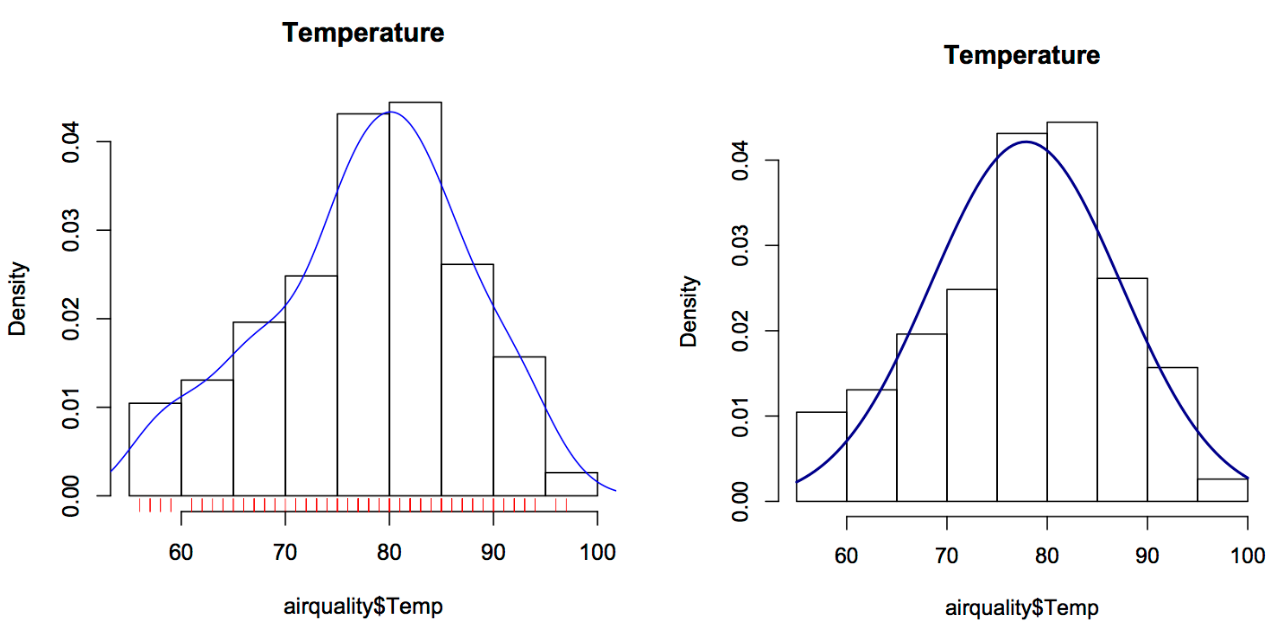
If we want to fit a **normal** curve over the data, instead of the command `density()` we can use `dnorm()` and `curve()` like so:

```
> m <- mean(airquality$Temp); std <- sqrt(var(airquality$Temp))

> hist(airquality$Temp, prob=T, main="Temperature")

> curve(dnorm(x, mean=m, sd=std), col="darkblue", lwd=2, add=TRUE)
```

**Note : If you use this option you need to make sure that you have `prob=TRUE` as an argument in your histogram !



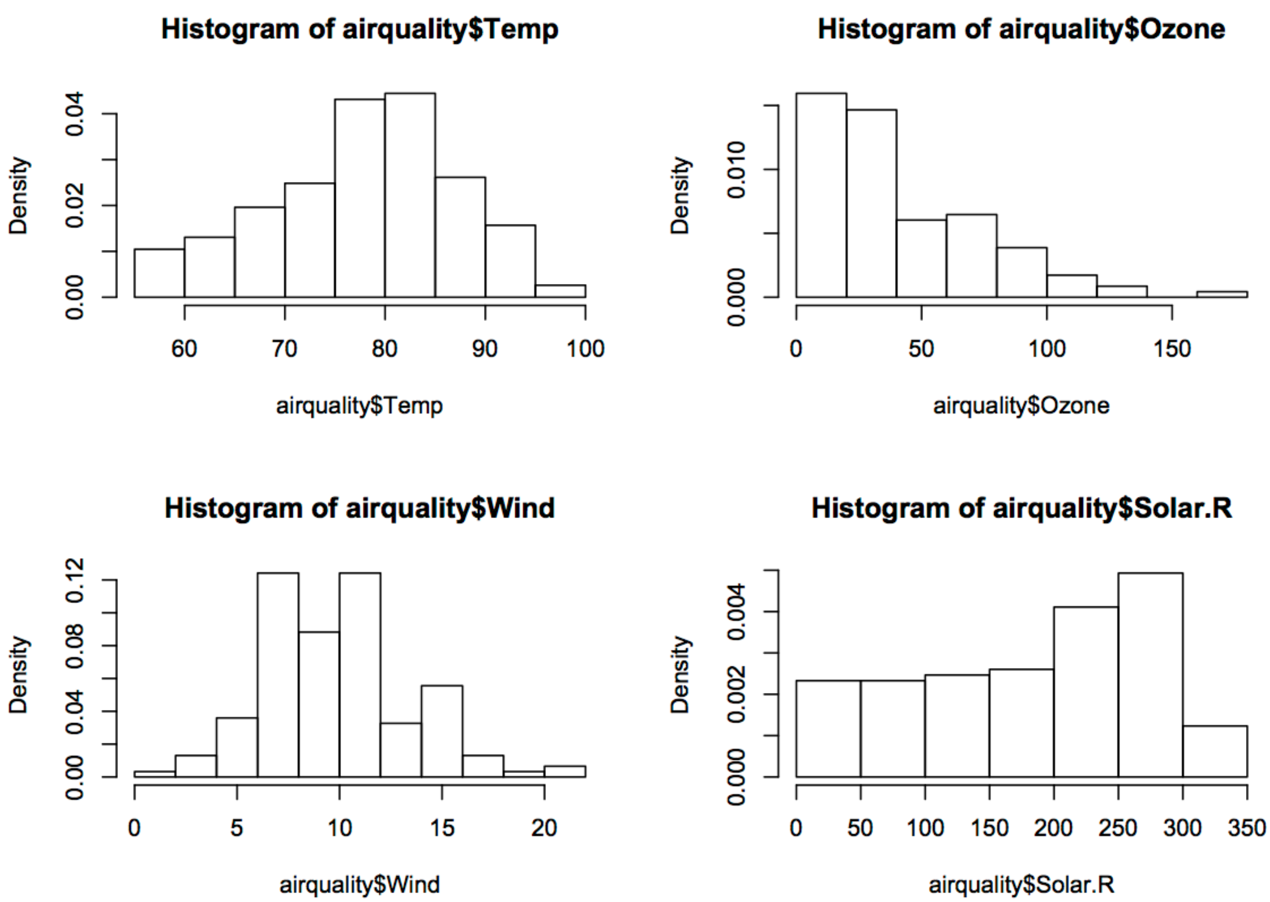
If you type `help(hist)` into the console, the R help page will show all the possible parameters you can add to a histogram. There are a lot of options.

If you want two or more plots in the same window, you can use the command

```
> par(mfrow=c( #rows, #columns ) )
```

With the airquality dataset, we can do this :

```
> par(mfrow=c(2,2))
> hist(airquality$Temp, prob=T)
> hist(airquality$Ozone, prob=T)
> hist(airquality$Wind, prob=T)
> hist(airquality$Solar.R, prob=T)
```



And we can make the bars colored by manipulating the `col=` argument in the `hist()` function.

Stem and Leaf Plots:
A neat way to summarize data that could just as easily be put in a histogram is to use a stem-and-leaf plot (compare to a histogram):

```
> stem(rnorm(40,100))
  The decimal point is 1 digit(s) to the right of the |

   97 | 7
   98 |
   98 | 5669
   99 | 00224
   99 | 58
  100 | 00111111122233
  100 | 55567889
  101 | 014
  101 | 6
  102 | 01
```

2.2.2 QQ plots

To see whether data can be assumed to be normally distributed, it is useful to make a *qq-plot*. We plot the kth smallest observation against the **expected value** of the kth smallest observation out of *n* in a standard normal distribution. We expect to obtain a straight line ***if*** the data come from a normal distribution.

```
> qqnorm(airquality$Temp)
```

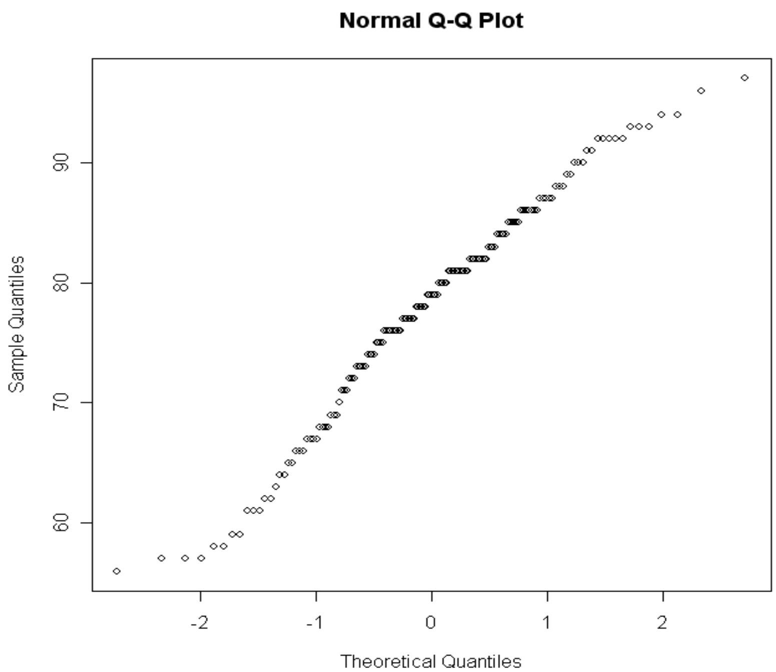


Fig. 3.2.2

The observed (empirical) quantiles are drawn along the vertical axis, while the theoretical quantiles are along the horizontal axis. The distribution is normal if the slope follows a diagonal line. Deviations towards the end indicate a heavy tail. This will come in handy when validating the appropriateness of linear regression.

After you generate the plot, use the function `qqline()` to fit a line going through the first and third quartile. This can be used to judge (non-statistically) the goodness-of-fit of the QQ-plot to a straight line.

Exercise 2. Use a histogram and qq-plot to determine whether the Ozone measurements in the *airquality* data can be considered normally distributed.

2.2.3 Box Plots

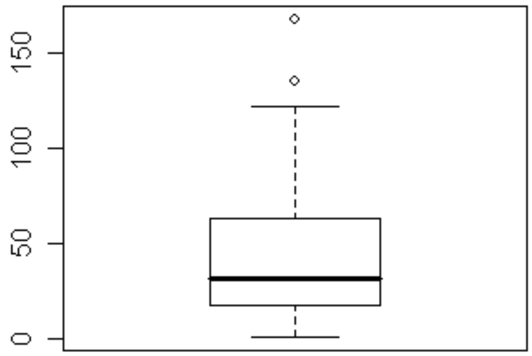
A "boxplot", or "box-and-whiskers plot" is a graphical summary of a distribution; the box in the middle indicates "hinges" (close to the first and third quartiles) and median. The lines ("whiskers") show the largest or smallest observation that falls within a distance of 1.5 times the box size from the nearest hinge. If any observations fall farther away, the additional points are considered "extreme" values and are shown separately. A boxplot can often give a good idea of the data distribution.

It is useful to compare distributions side-by-side, as it is more compact than a histogram.

```
> boxplot(airquality$Ozone)
```

We can use the boxplot function to calculate quick summaries for all the variables in our data set—by default, R computes boxplots column by column. Notice that missing data causes no problems to the boxplot function (similar to summary).

```
> boxplot(airquality[,1:4])
# only for the numerical variables
```



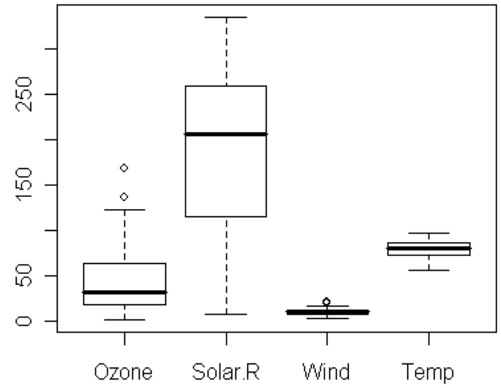


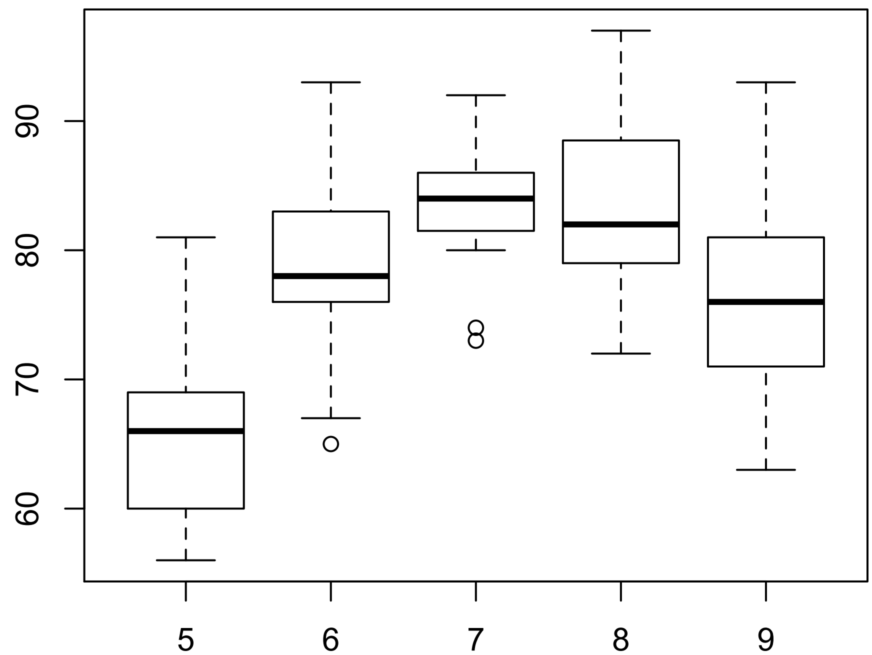
Fig. 2.2.3

Figure (b) is not always meaningful, as the variables may not be on comparable scales. The real power of box plots is to do comparisons of variables by **sub-grouping**. For example, we may be interested in comparing the fluctuations in temperature across months. To create boxplots of temperature data grouped by the factor "month", we use the command:

```
> boxplot(airquality$Temp ~ airquality$Month)
```

We can also write the same command using a slightly more readable language:

```
> boxplot(Temp ~ Month, data = airquality)
```



The tilde symbol "~" indicates which factor to group by. We will come back to more discussion on plotting grouped data later on.

For those interested in more on this topic, see http://www.r-bloggers.com/box-plot-with-r-tutorial/

2.2.4 Scatter Plots

One very commonly used tool in exploratory analysis is the scatterplot. We will look at this in more detail later when we discuss regression and correlation. The R command for drawing a scatterplot of two variables is a simple command of the form `plot(x,y)`.

```
> plot(airquality$Temp, airquality$Ozone)
```

```
# How do Ozone and temperature measurements relate?
```

With more than two variables, the `pairs()` command draws a scatterplot matrix.

Exercise 3. Write the following command in R and describe what you see in terms of relationships between the variables.

```
> pairs(airquality[,1:4])
```

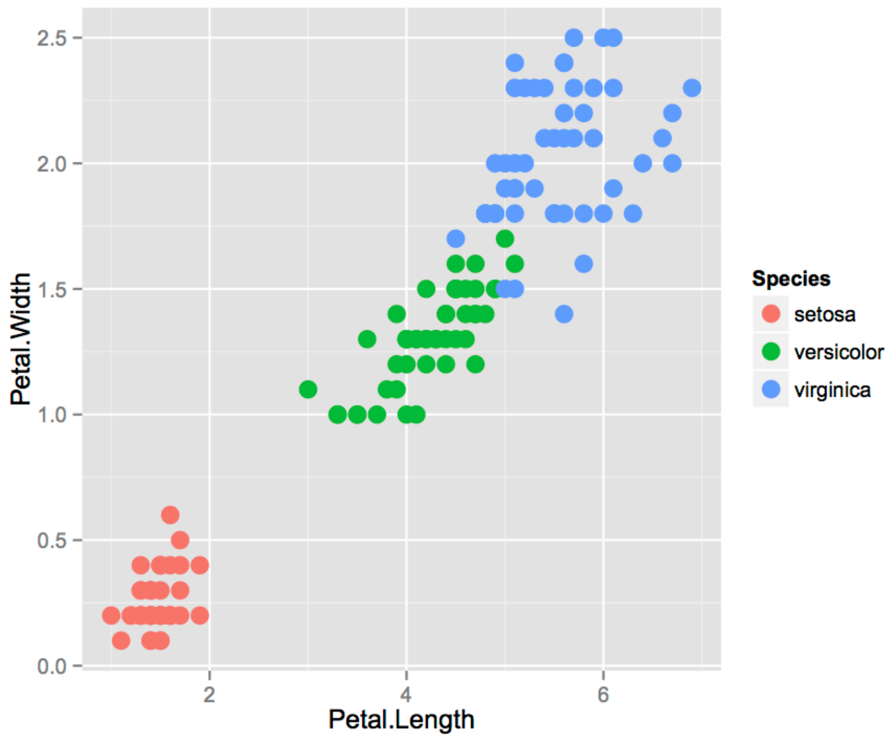
The default plotting symbols in R are not always pretty! You can change the plotting symbols, or colors to something nicer. For example, the following command

```
> plot(airquality$Temp, airquality$Ozone, col="red", pch =19)
```

repeats the scatterplot, this time with red filled circles that are nicer to look at. "`col`" refers to the color of symbols plotted. The full range of point plotting symbols used in R are given by "`pch`" in the range 1 to 25; see the help on "`points`" to see what each of these represent.

Aside:
R has a *very* nice package for graphics called `ggplot2`. We won't do much with it in this course, but here is an example using the in-built `iris` R dataset:

```
> #install.packages("ggplot2")
> library(ggplot2)
> ?iris
>
> head(iris)
> ggplot(iris, aes(x = Petal.Length, y = Petal.Width, color = Species)) + geom_point(size = 4)
```



See http://stackoverflow.com/questions/12675147/how-can-we-make-xkcd-style-graphs-in-r for how to make XKCD-type graphics in the ggplot2 package.

2.3 Summary statistics for groups

When dealing with grouped data, you may want to have various summary statistics computed *within* each subgroup. This can be done using the `tapply()` command. For example, we might want to compute the mean temperature in each month:

```
> tapply(airquality$Temp, airquality$Month, mean)
       5        6        7        8        9
65.54839 79.10000 83.90323 83.96774 76.90000
```

The first argument is the variable of interest, while the second is the stratifying variable. If there are any missing values, these can be excluded by adding the extra argument `na.rm=T`.

Exercise 4. Compute the range and mean of Ozone levels for each month, using the `tapply()` command.

2.3.2 Tables

Categorical data are often described in the form of tables. We now discuss how you can create tables from your data and calculate relative frequencies. The simple "table" command in R can be used to create one-, two- and multi-way tables from categorical data. For the next few examples we will be using the dataset airquality.new.csv. Here is a preview of the dataset:

```
  Ozone Solar.R Wind Temp Month Day goodtemp badozone
1    41     190  7.4   67     5   1      low      low
2    36     118  8.0   72     5   2      low      low
3    12     149 12.6   74     5   3      low      low
4    18     313 11.5   62     5   4      low      low
7    23     299  8.6   65     5   7      low      low
```

The columns *goodtemp* and *badozone* represent days when temperatures were greater than or equal to 80 (good) or not (low) and if the ozone was greater than or equal to 60 (high) or not (low), respectively.

Exercise 7. Read in the airquality.new.csv file and print out rows 50 to 60 of the new data set `airquality.new`.

Now, let us construct some simple tables. Make sure you first detach the airquality data set and attach `airquality.new`.

```
> table(goodtemp)
goodtemp
high  low
  54   57

> table(badozone)

> table(goodtemp, badozone)

> table(goodtemp, Month)
        Month
goodtemp  5  6  7  8  9
    high  1  4 24 15 10
    low  23  5  2  8 19
```

We can also construct tables with more than two sides in R. For example, what do you see when you do the following?

```
> table(goodtemp, Month, badozone)

, , badozone = high

        Month
goodtemp  5  6  7  8  9
    high  0  1 13 10  4
    low   1  0  0  0  0

, , badozone = low

        Month
goodtemp  5  6  7  8  9
    high  1  3 11  5  6
    low  22  5  2  8 19
```

As you add dimensions, you get more of these two-sided sub-tables and it becomes easy to lose track. This is where the `ftable` command is useful. This function creates "flat" tables; e.g., like this:

```
> ftable (goodtemp + badozone ~ Month)
      goodtemp high      low
      badozone high low high low
Month
5                 0   1    1  22
6                 1   3    0   5
7                13  11    0   2
8                10   5    0   8
9                 4   6    0  19
```

It may sometimes be of interest to compute marginal tables; that is, the sums of the counts along one or the other dimension of a table, or relative frequencies, generally expressed as proportions of the row or column totals.

These can be done by means of the `margin.table()` and `prop.table()` functions respectively. For either of these, we first have to construct the original cross-table.

```
> Temp.month = table(goodtemp, Month)
> margin.table(Temp.month,1)
## what does the second index (1 or 2) mean?

> margin.table(Temp.month,2)
> prop.table(Temp.month,2)
```

**Note: for those interested in exporting an R table to a nice LaTeX formatted table, see documentation for the package `xtable`: http://cran.r-project.org/web/packages/xtable/xtable.pdf

2.4 Graphical display of tables

For presentation purposes, it may be desirable to display a graph rather than a table of counts or percentages, with categorical data. One common way to do this is through a bar plot or bar chart, using the R command `barplot`. With a vector (or 1-way table), a bar plot can be simply constructed as:

```
> total.temp = margin.table(Temp.month,2)
> barplot(total.temp)          ## what does this show?
```

If the argument is a matrix, then `barplot` creates by default a "stacked barplot", where the columns are partitioned according to the contributions from different rows of the table. If you want to place the row contributions beside each other instead, you can use the argument `beside=T`.

Exercise 8. Construct a table of the "`badozone`" variable by month from the airquality.new data. Then create and interpret the bar plot you get using the following commands:

```
> ozone.month = table(badozone, Month)
> barplot(ozone.month)
> barplot(ozone.month, beside=T)
```

The bar plot by default appears in color; if you want a black-and-white illustration, you just need to add the argument `col="white"`.

2.4.1 Dot charts

Dot charts can be employed to study a table from both sides at the same time. They contain the same information as barplots with `beside=T` but give a different visual impression.

```
> dotchart(ozone.month)
```

2.5 Saving and exporting figures in R

Click on the figure created in R so that the window is selected. Now click on the dropdown menu bar labeled “File” and click “Save as”- you will see a number of options for file formats to save the file in.

Common ones are PDF, png, jpeg, etc. The “png” (portable network graphic) format is often the most compact, and is readable on different platforms and can be easily inserted into Word or PowerPoint documents.

There is also a direct “command-line” option to save figures as a file from R. The command varies according to file format, but the basic syntax is to open the file to save in, then create the plot, and finally close the file. For example,

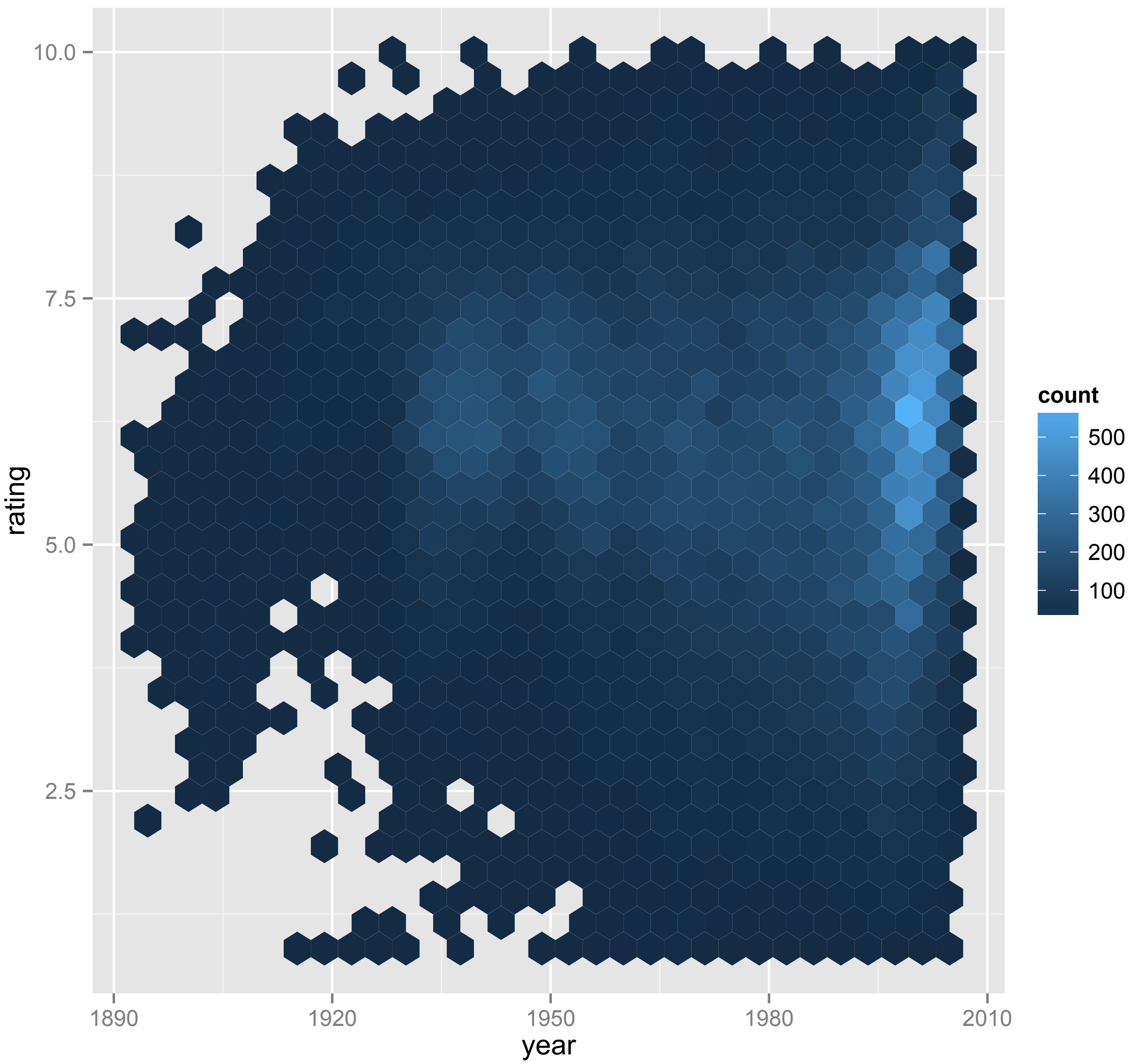
```
> png("barplotozone.png")      ## name the file to save the figure in
> barplot(ozone.month, beside=T)
> dev.off()                            ## close the plotting device
```

This option is often useful when you need to plot a large number of figures at once, and doing them one-by-one becomes cumbersome. There are also a number of ways to control the shape and size of the figure. Look up the help on the figure plotting functions (e.g. "`help(png)`") for more details.

Appendix:

3-D plots
manipulating the graph axes
(see class R code)

There is also a very nice cheat sheet for the package ggplot2 up on the websites for those interested in *very* nice graphics. For example:



Class overview:

- Summary Statistics (`mean`, `var`, `apply` commands, `summary`)
- Graphical Displays of Data: Histograms, Box-and-Whiskers, Stem-and-Leaf, Scatterplots, options
- Displaying Tabular Data
- Saving Plots