R Tutorial

R is an *open source* software for statistical analysis, computing, and graphics http://cran.r-project.org/. It provides a software environment where data, results, and functions are treated as objects. The users can interact directly with these objects at the command prompt. A comprehensive online introduction to R is available at http://cran.r-project.org/doc/manuals/R-intro.html. Sections 2 to 9 are particularly useful if you are not familiar with R. R is a great tool to conduct analyses in a reproducible manner.

R studio is a very helpful application that provides an easy interface to R, where you can create text files that conduct and document your analysis http://www.rstudio.com/.

To compile code and text into PDF, you may need to install latex. When you try to compile this code, look for errors and follow their instructions.

We will illustrate some basic operations using a dataset of daily ambient ozone concentrations and temperature in 3 Georgia counties between March and November 2000. Please download the `ozoneATL_2000.csv` and `OzoneATL_2000.RData`. You will also need to change the path names to get the code to run (as indicated below).

Installing R Packages

R packages are bundles of code creating functions that do not exist in basic R (to analyze or plot data). There are thousands of them, but a few main ones will meet many of your needs. Additional ones can be found online.

(Option 1) Identify the package name & download it.

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

(Option 2) Have RStudio do everything for you: Search for the package & click Install. Once it is installed, check the box in the package list.

How to read in a dataset?

(Option 1) To read in a comma-separated (.csv) file, use the read.csv function. If you launched this tutorial by clicking on M0_R_Overview.Rproj, this should work:

```
data <- read.csv ("ozoneATL_2000.csv")
```

Here we read in a csv file (`ozoneATL_2000.csv`) and assign it to an object called `data` using the right arrow (`<-`) operator. The equal sign (`=`) also works for assigning objects. If we are using R projects, then the working directory will be set to the directory containing the .Rproj file. Let's check which working directory we are using:

```
getwd()
```

```
## [1] "/home/benjaminrisk/Dropbox/EmoryCourses/BIOS_526/Modern-Regression-Analysis/M0_R_Overview"
```

Working with R Data Frame

Here are some functions that describe a dataset.

```
dim (data) #Print the dimension (row by column) of a dataset
```

```
## [1] 724   3
```

```
names (data) #Print the variable names
```

```
## [1] "TEMP"   "Ozone"  "COUNTY"
```

Here we see that the data frame has 3 variables and 724 records. The function `str( )` prints the structure of an object, here the variable names, their classes, and the first few records.

```
str (data) #Print the variable's classes and give examples
```

```
## 'data.frame':    724 obs. of  3 variables:
##  $ TEMP  : num  58.7 57.7 55.9 54.5 53.7 59.4 63.1 64.5 65.8 62.2 ...
##  $ Ozone : num  0.043 0.04 0.022 0.02 0.039 0.045 0.057 0.035 0.022 0.025 ...
##  $ COUNTY: chr  "Cobb" "Cobb" "Cobb" "Cobb" ...
```

The above shows that the first two variables (TEMP) and (Ozone) are numerical values, while the third variable (COUNTY) is a *factor*. COUNTY is a categorical variable for the three counties (Cobb, DeKalb, and Fulton).

We can quickly view the data in R studio by clicking on the dataset's name in the "Environment" window.

We can also use the function `summary ( )` to calculate some summary statistics for each variable. Note that for COUNTY, we see the frequency count for each categorical group. This function is also very useful to see whether R read-in the variables as continuous or categorical.

```
summary (data)
```

```
##       TEMP           Ozone             COUNTY
##  Min.   :43.40   Min.   :0.00700   Length:724
##  1st Qu.:63.40   1st Qu.:0.03800   Class :character
##  Median :71.70   Median :0.05100   Mode  :character
##  Mean   :70.06   Mean   :0.05366
##  3rd Qu.:77.90   3rd Qu.:0.06700
##  Max.   :89.60   Max.   :0.13500
```

There are several ways to extract elements from a data frame. One approach is to use the square brackets, *[row numbers, column numbers]*. For example, to get the first row or the first three rows:

```
data[1, ]   # First row of the dataset
```

```
##   TEMP Ozone COUNTY
## 1 58.7 0.043   Cobb
```

```
data[1:3, ] # First three rows of the data set. Here the colon indicates "to"
```

```
##   TEMP Ozone COUNTY
## 1 58.7 0.043   Cobb
## 2 57.7 0.040   Cobb
## 3 55.9 0.022   Cobb
```

```
data[ ,2]   # Second column (too much to print)
```

```
##   [1] 0.043 0.040 0.022 0.020 0.039 0.045 0.057 0.035 0.022 0.025 0.038 0.034
##  [13] 0.034 0.044 0.029 0.027 0.034 0.036 0.038 0.040 0.050 0.046 0.037 0.051
##  [25] 0.053 0.033 0.053 0.029 0.031 0.049 0.061 0.032 0.033 0.040 0.050 0.062
```

```
##  [37] 0.041 0.034 0.044 0.034 0.046 0.035 0.023 0.017 0.027 0.046 0.045 0.043
##  [49] 0.051 0.054 0.037 0.054 0.047 0.030 0.032 0.057 0.067 0.050 0.071 0.083
##  [61] 0.068 0.064 0.060 0.052 0.068 0.066 0.060 0.060 0.054 0.049 0.058 0.043
##  [73] 0.052 0.062 0.066 0.074 0.072 0.067 0.047 0.047 0.043 0.055 0.059 0.052
##  [85] 0.042 0.051 0.035 0.045 0.048 0.069 0.080 0.104 0.087 0.081 0.042 0.034
##  [97] 0.050 0.073 0.093 0.108 0.107 0.105 0.088 0.048 0.046 0.039 0.026 0.042
## [109] 0.031 0.039 0.035 0.048 0.050 0.068 0.097 0.059 0.051 0.066 0.035 0.034
## [121] 0.073 0.067 0.099 0.083 0.072 0.059 0.069 0.065 0.074 0.073 0.066 0.074
## [133] 0.077 0.083 0.084 0.066 0.068 0.073 0.092 0.073 0.079 0.082 0.078 0.057
## [145] 0.065 0.041 0.064 0.110 0.093 0.064 0.052 0.038 0.035 0.027 0.057 0.066
## [157] 0.076 0.069 0.040 0.042 0.065 0.083 0.069 0.063 0.065 0.071 0.072 0.087
## [169] 0.107 0.082 0.098 0.069 0.041 0.077 0.087 0.079 0.067 0.090 0.053 0.074
## [181] 0.079 0.062 0.026 0.036 0.046 0.041 0.045 0.048 0.025 0.023 0.025 0.044
## [193] 0.056 0.045 0.066 0.061 0.058 0.056 0.051 0.050 0.041 0.071 0.036 0.014
## [205] 0.020 0.023 0.029 0.025 0.031 0.035 0.041 0.055 0.047 0.045 0.041 0.062
## [217] 0.071 0.050 0.028 0.032 0.028 0.028 0.032 0.033 0.045 0.044 0.063 0.061
## [229] 0.054 0.054 0.047 0.040 0.040 0.046 0.050 0.065 0.058 0.052 0.055 0.063
## [241] 0.061 0.044 0.037 0.047 0.034 0.029 0.034 0.018 0.035 0.036 0.063 0.072
## [253] 0.033 0.018 0.022 0.035 0.025 0.045 0.032 0.020 0.029 0.030 0.032 0.023
## [265] 0.038 0.041 0.044 0.047 0.046 0.012 0.048 0.015 0.032 0.032 0.039 0.048
## [277] 0.056 0.047 0.036 0.041 0.050 0.056 0.026 0.019 0.015 0.025 0.047 0.046
## [289] 0.035 0.045 0.063 0.037 0.047 0.043 0.030 0.023 0.041 0.065 0.044 0.066
## [301] 0.078 0.071 0.065 0.060 0.055 0.058 0.054 0.061 0.056 0.055 0.044 0.058
## [313] 0.043 0.054 0.069 0.066 0.075 0.066 0.067 0.052 0.052 0.041 0.055 0.051
## [325] 0.045 0.040 0.039 0.036 0.048 0.045 0.064 0.076 0.106 0.081 0.085 0.036
## [337] 0.019 0.045 0.067 0.071 0.090 0.074 0.068 0.064 0.048 0.050 0.038 0.028
## [349] 0.031 0.047 0.043 0.046 0.061 0.040 0.075 0.074 0.046 0.061 0.053 0.026
## [361] 0.017 0.073 0.075 0.062 0.077 0.071 0.055 0.085 0.079 0.060 0.062 0.075
## [373] 0.072 0.065 0.089 0.079 0.065 0.081 0.078 0.111 0.076 0.078 0.085 0.091
## [385] 0.051 0.069 0.032 0.069 0.078 0.107 0.068 0.057 0.037 0.036 0.029 0.043
## [397] 0.055 0.085 0.066 0.045 0.057 0.061 0.110 0.063 0.066 0.071 0.087 0.081
## [409] 0.091 0.135 0.080 0.103 0.061 0.046 0.067 0.087 0.079 0.062 0.083 0.038
## [421] 0.073 0.072 0.061 0.026 0.034 0.038 0.035 0.052 0.036 0.018 0.019 0.022
## [433] 0.036 0.039 0.043 0.069 0.055 0.056 0.050 0.051 0.044 0.024 0.057 0.033
## [445] 0.007 0.008 0.023 0.033 0.021 0.023 0.031 0.043 0.050 0.041 0.037 0.037
## [457] 0.065 0.059 0.041 0.014 0.023 0.022 0.022 0.018 0.022 0.043 0.040 0.039
## [469] 0.052 0.048 0.047 0.038 0.045 0.052 0.053 0.051 0.067 0.046 0.067 0.054
## [481] 0.052 0.058 0.046 0.030 0.025 0.044 0.038 0.037 0.026 0.041 0.041 0.061
## [493] 0.074 0.040 0.019 0.024 0.037 0.037 0.046 0.054 0.036 0.030 0.033 0.035
## [505] 0.043 0.037 0.044 0.044 0.055 0.055 0.026 0.057 0.026 0.029 0.052 0.059
## [517] 0.033 0.030 0.042 0.057 0.045 0.032 0.044 0.050 0.055 0.028 0.026 0.018
## [529] 0.031 0.048 0.049 0.041 0.045 0.064 0.041 0.053 0.045 0.026 0.029 0.054
## [541] 0.072 0.050 0.073 0.082 0.073 0.068 0.063 0.056 0.060 0.056 0.064 0.057
## [553] 0.055 0.050 0.058 0.045 0.056 0.070 0.074 0.074 0.065 0.068 0.052 0.052
## [565] 0.042 0.060 0.062 0.050 0.047 0.041 0.039 0.051 0.050 0.065 0.080 0.107
## [577] 0.086 0.089 0.040 0.031 0.050 0.071 0.077 0.101 0.080 0.076 0.073 0.050
## [589] 0.053 0.039 0.031 0.033 0.042 0.042 0.047 0.067 0.048 0.084 0.079 0.057
## [601] 0.066 0.057 0.034 0.029 0.092 0.089 0.078 0.099 0.083 0.067 0.085 0.079
## [613] 0.063 0.068 0.076 0.076 0.065 0.092 0.084 0.070 0.081 0.077 0.109 0.081
## [625] 0.080 0.083 0.091 0.056 0.077 0.038 0.075 0.092 0.113 0.063 0.057 0.038
## [637] 0.035 0.028 0.046 0.067 0.090 0.081 0.052 0.063 0.067 0.120 0.074 0.070
## [649] 0.078 0.092 0.080 0.094 0.132 0.085 0.115 0.066 0.047 0.070 0.090 0.086
## [661] 0.068 0.084 0.044 0.084 0.083 0.065 0.021 0.036 0.046 0.043 0.054 0.045
## [673] 0.023 0.025 0.025 0.039 0.044 0.047 0.072 0.051 0.063 0.057 0.054 0.047
```

```
## [685] 0.033 0.061 0.032 0.015 0.028 0.035 0.048 0.054 0.044 0.042 0.035 0.068
## [697] 0.060 0.049 0.022 0.028 0.025 0.026 0.024 0.024 0.047 0.042 0.041 0.058
## [709] 0.044 0.039 0.049 0.043 0.051 0.053 0.056 0.068 0.054 0.059 0.057 0.061
## [721] 0.061 0.047 0.041 0.034
```

```
data[2,2]   # Second row, second column
```

```
## [1] 0.04
```

Because a data frame contains variable names, we can also directly extract a variable (all records) by using the dollar ($) sign.

```
ozone = data$Ozone
class (ozone)
```

```
## [1] "numeric"
```

```
dim(ozone)
```

```
## NULL
```

```
length(ozone)
```

```
## [1] 724
```

```
ozone[1:4]
```

```
## [1] 0.043 0.040 0.022 0.020
```

In the above operations, a new object *ozone* is defined and takes the values of the variable "Ozone" in the data frame. Note that ozone is a *vector* of class numeric. It has NULL dimension but a vector of length 724.

Here are some example functions to calculate simple statistical summaries of a variable:

```
median (data$Ozone)
```

```
## [1] 0.051
```

```
quantile (data$Ozone, .50)
```

```
##   50%
## 0.051
```

```
sd (data$Ozone)
```

```
## [1] 0.02094673
```

```
max (data$Ozone)
```

```
## [1] 0.135
```

Creating New Variables

It is easy to define new variables in a data frame. For example, we can create a new variable called "LogOzone" as the log of daily ozone concentrations using

```
data$LogOzone = log (data$Ozone)
data[1:2,]
```

```
##   TEMP Ozone COUNTY  LogOzone
## 1 58.7 0.043   Cobb -3.146555
## 2 57.7 0.040   Cobb -3.218876
```

To remove a variable, we can use a "minus" operation in the square bracket

```
data = data[,-4]
data[1:2,]
```

```
##   TEMP Ozone COUNTY
## 1 58.7 0.043   Cobb
## 2 57.7 0.040   Cobb
```

Vectors

In R, a sequence of numbers and characters can be stored in a vector. The function `c( )`, for combining, is used to create a vector. For example,

```
x = c(1, 3, 5, 6)
y = c("a", "b", "c")
class (x)
```

```
## [1] "numeric"
```

```
class (y)
```

```
## [1] "character"
```

R has several functions to create structured vectors: `seq( )` for consecutive elements; `rep( )` for repeating elements. Element-by-element arithmetic operations are available to the vectors.

```
x + x
```

```
## [1]  2  6 10 12
```

```
x*x
```

```
## [1]  1  9 25 36
```

```
x^2
```

```
## [1]  1  9 25 36
```

Matrix

R can also store data in blocks as a matrix.

```
x = matrix (c(1,2,3,4, 5, 6), nrow = 2, ncol = 3)
x
```

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

```
class(x)
```

```
## [1] "matrix" "array"
```

Note that the default is to fill a matrix by column. You can use the option "`by.row = TRUE`"

```
x = matrix (c(1,2,3,4, 5, 6), nrow = 2, ncol = 3, byrow = T)
x
```

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

Functions such as `dim( )`, `nrow( )`, `ncol( )` can be used to obtain dimensions of a matrix. Elements of a matrix can also be extracted using the square bracket notations. Common matrix operations include transpose `t( )`, multiplication `%*%`, inverse `solve( )`, diagonal elements `diag( )`. You can also perform element-by-element arithmetic operations.

```
t(x)
```

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

```
x%*%t(x)
```

```
##      [,1] [,2]
## [1,]   14   32
## [2,]   32   77
```

```
solve (x%*%t(x))
```

```
##            [,1]       [,2]
## [1,]  1.4259259 -0.5925926
## [2,] -0.5925926  0.2592593
```

```
x-1
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    3    4    5
```

Note how the last operation below subtracts 1 from *every* element. This is an example of R's "recycling" behavior, which makes it fast to program, but the results are not always intuitive. For example:

```
x - c(1,2)
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    2    3    4
```

Whereas

```
x - c(1,2,3)
```

```
##      [,1] [,2] [,3]
## [1,]    0   -1    1
## [2,]    2    4    3
```

subtracts c(1,2,3) from the vectorized "x", which here is equivalent to 1 from x[1,1], 2 from x[1,2], and 3 from x[2,1], then R "recycles" c(1,2,3), i.e., 1 from x[2,2], 2 from x[1,3], and 3 from x[2,3]. See what happens with the following code:

```
x - c(1,2,3,4,5)
```

```
## Warning in x - c(1, 2, 3, 4, 5): longer object length is not a multiple of
## shorter object length
```

```
##      [,1] [,2] [,3]
## [1,]    0   -1   -2
## [2,]    2    1    5
```

Factors

R stores categorical variables as factors and represent each group as a *level*. A factor is stored as a numeric integer with a character label, which is more memory efficient. When R reads in a CSV file, it will automatically convert variables with character values into a factor.

```
county = data$COUNTY
class(county)
```

```
## [1] "character"
```

```
levels (county)
```

```
## NULL
```

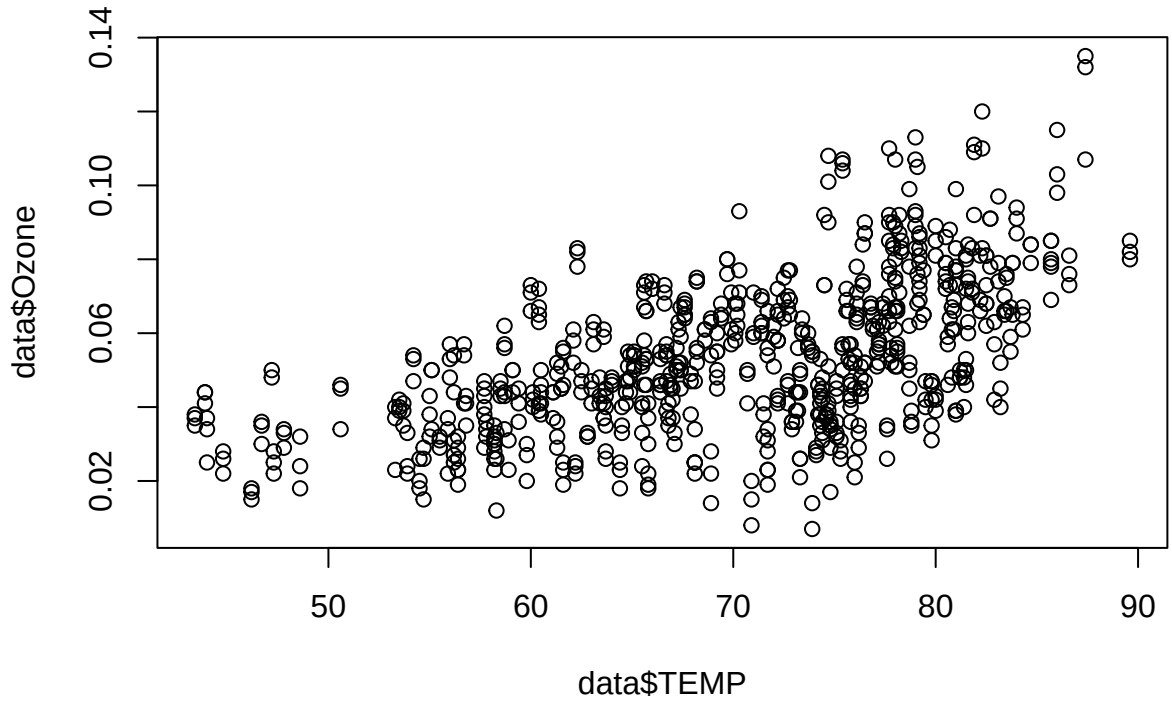
```
table (county)
```

```
## county
##   Cobb DeKalb Fulton
##    244    241    239
```

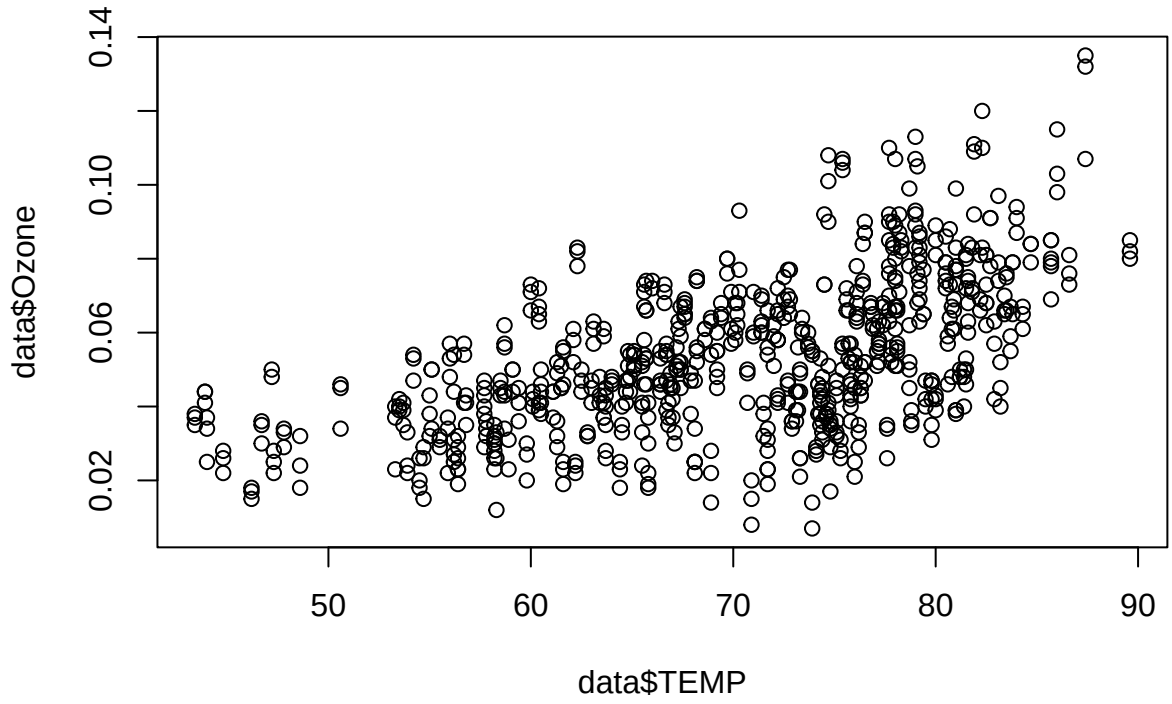
Plotting in R

The function `plot` will produce a scatter plot of two variables. The following two commands all produce the same plot of daily ozone levels versus daily temperature.

```
plot (data$Ozone~data$TEMP)
```

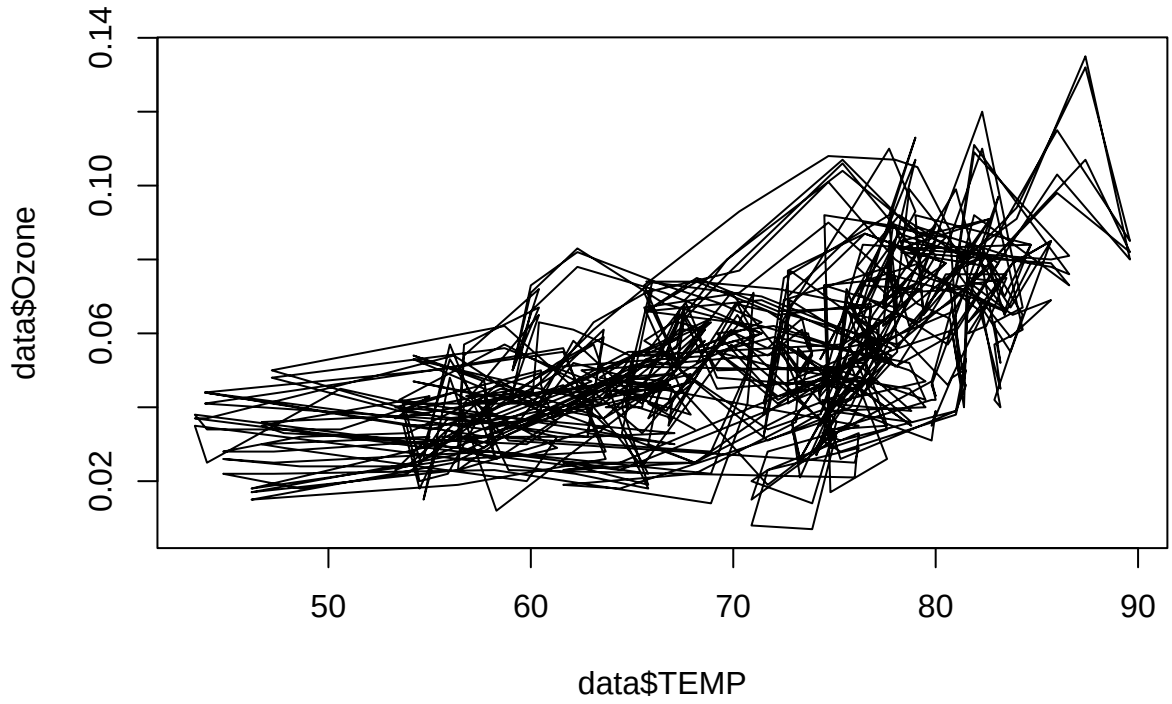


```
plot (data$TEMP, data$Ozone)
```



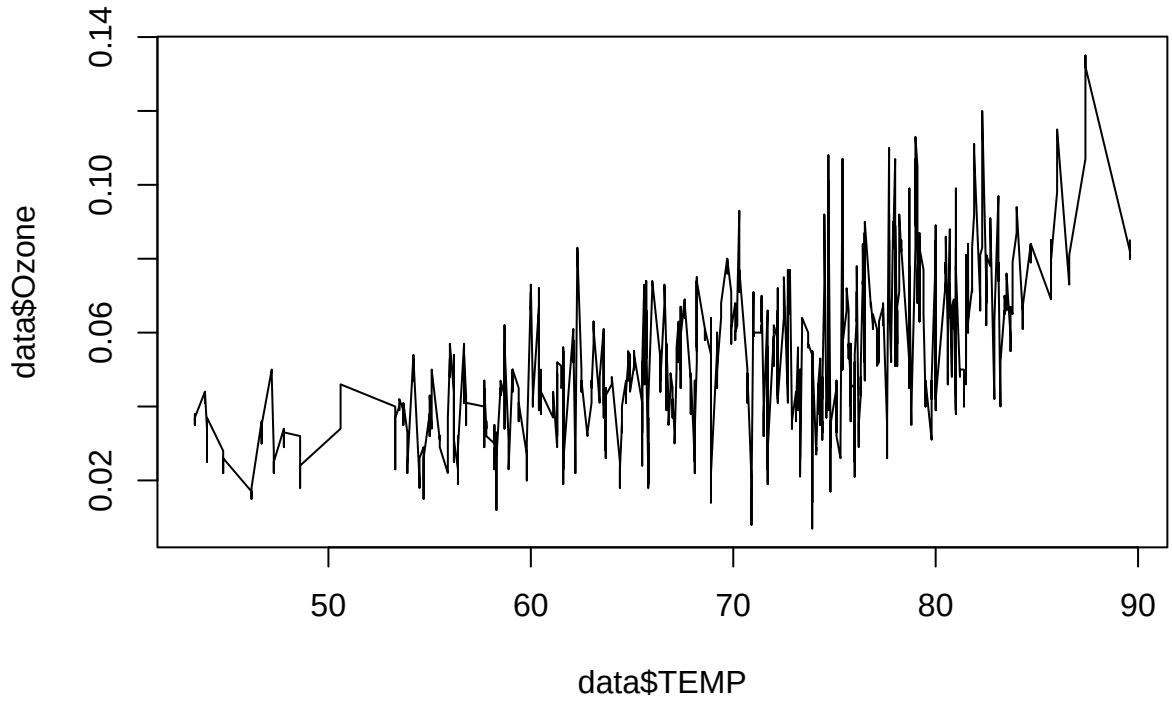
In R, the tilde notation (~) denotes relationship between variables. The response variable (y-axis) is given on the left of ~. If the tilde is not use, R typically interprets the first variable as the predictor (x-axis). Using the option type, the `plot` function can also be used to graph different 2-D plot. For example

```
plot (data$Ozone~data$TEMP, type = "l") #Connect the points
```

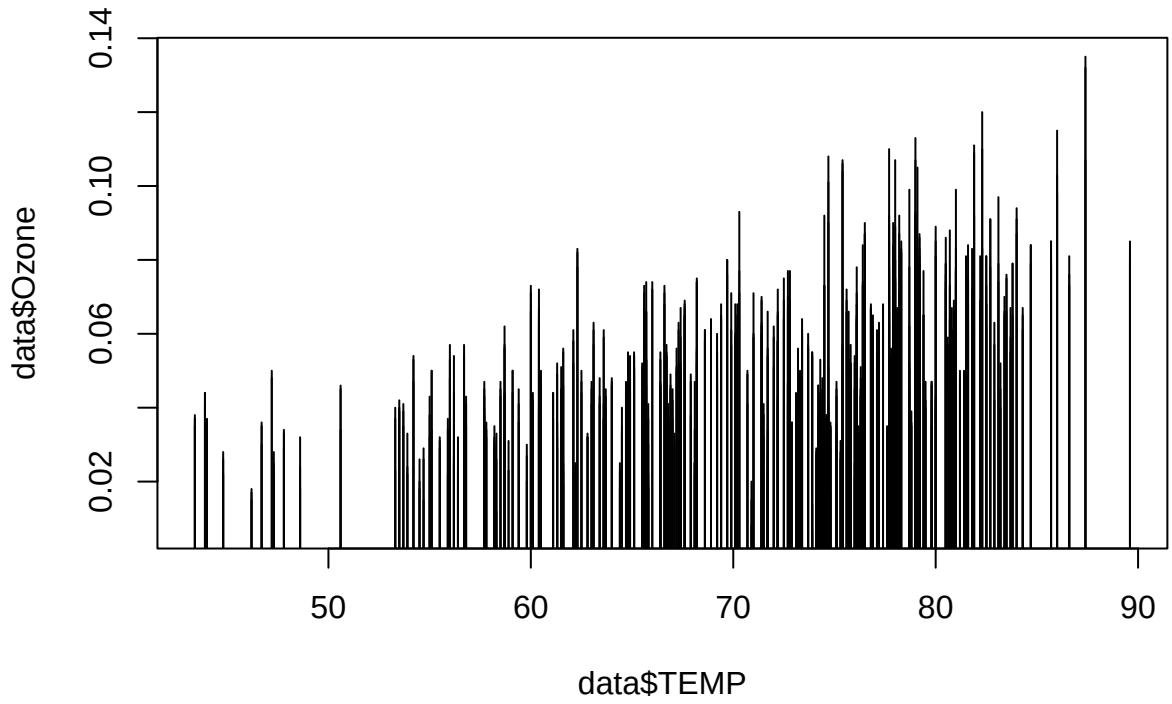



```
# sort data for more reasonable plot:
data = data[order(data$TEMP),]
```

```
plot (data$Ozone~data$TEMP, type = "l") #Connect the points
```

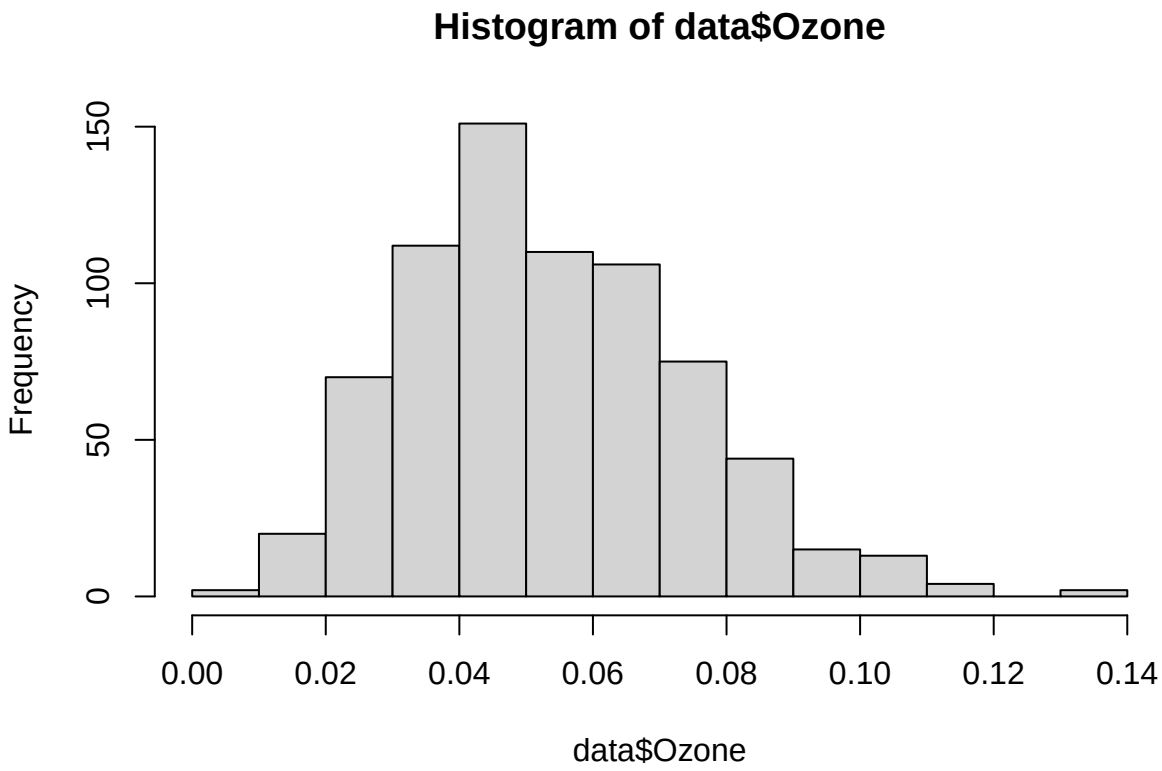



```
plot (data$Ozone~data$TEMP, type = "h") #Show vertical lengths
```

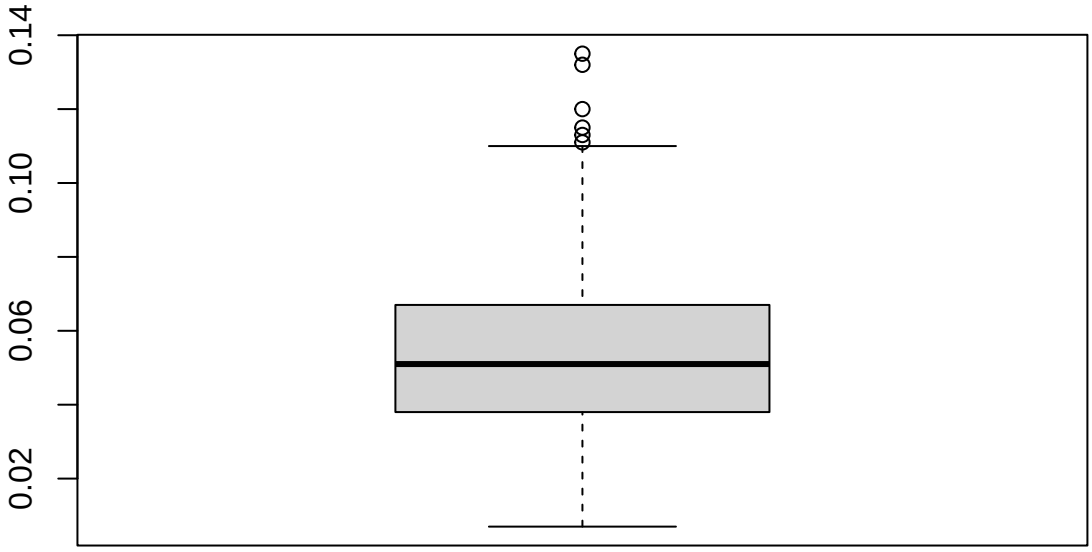


Other useful plots include:

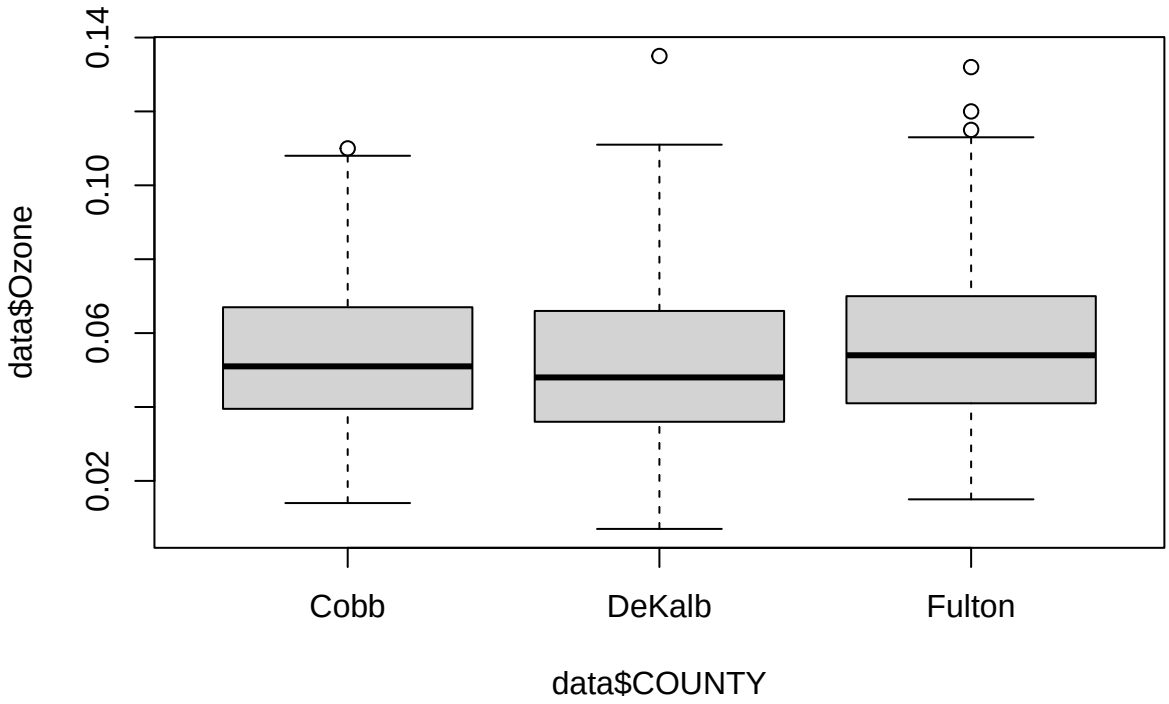
```
hist (data$Ozone)       #Plot histogram
```




```
boxplot (data$Ozone)    #Plot boxplot
```



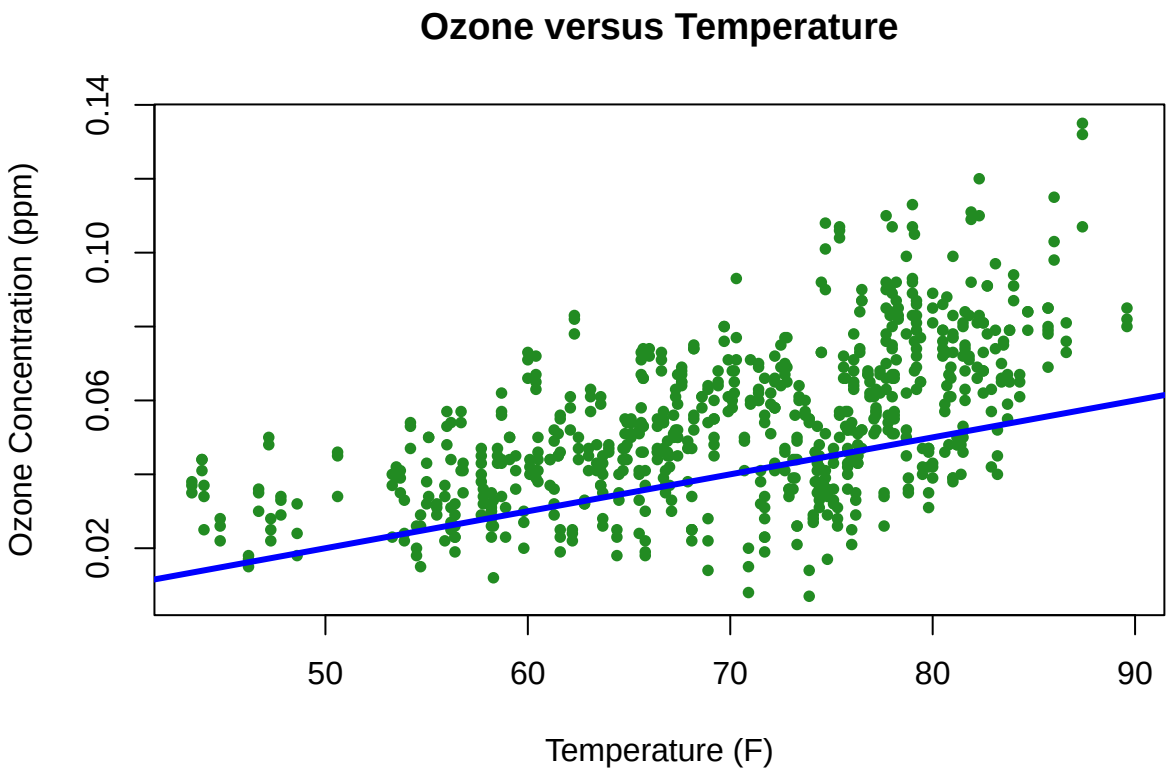

```
boxplot (data$Ozone~data$COUNTY)#Plot boxplot by county
```



One major advantage of R is the ability to easily customize graphics. Here is an example

```
plot (data$Ozone~data$TEMP,
      cex = 0.8,   ##Size of the plot symbole
      pch = 16,     ##Plot symbol
      col = "forestgreen", ##plot color
      main = "Ozone versus Temperature", ##Title text
      xlab = "Temperature (F)",     ##X-axis text
      ylab = "Ozone Concentration (ppm)"    ##Y-axis text
      )

abline (-0.030, 0.001, ## Intercept and slope of the ozone~temp best fit line
            lwd = 3, ## line width
            col = "blue") ##line color
```

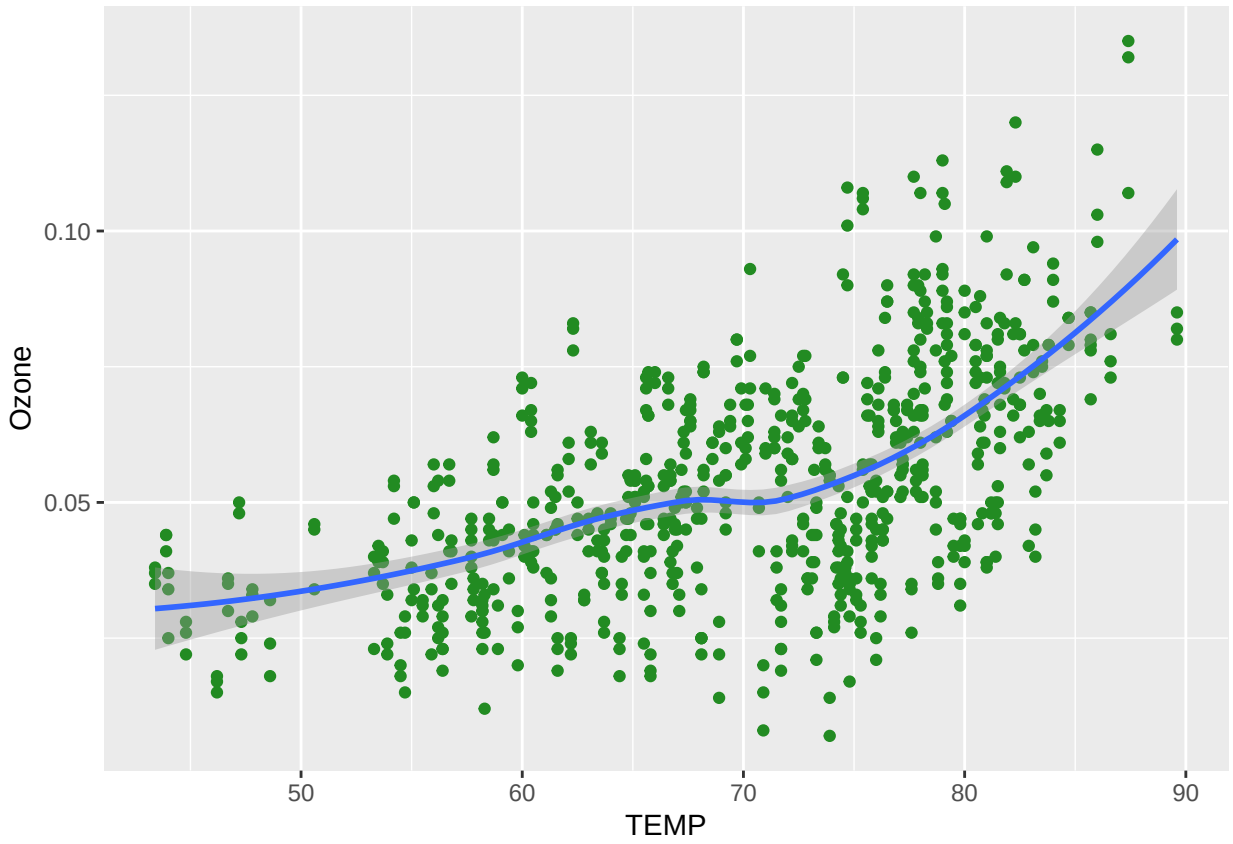


Alternative plotting functions

We used base R plotting functions above. Now, let's make a plot using ggplot. GGplot is a set of functions that can produce beautiful plots, although in my opinion, the syntax is a bit more difficult than the base R plots.

```
library(ggplot2)
p<-ggplot(data, aes(x=TEMP, y=Ozone)) + geom_point(color='forestgreen') + geom_smooth()
p
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Getting Help in R

For any function in R, you can access the help file by putting a question mark (`?`) in front of the function name. For example,

```
?hist
```

RStudio contains a help area on the right side with a search function.

The R help contains the following sections:

1. Description
2. Usage

- Show all arguments (input variables) for the function and their default values.
- These are the options that users can change

3. Arguments

- Describes each argument and how they change the model or outputs

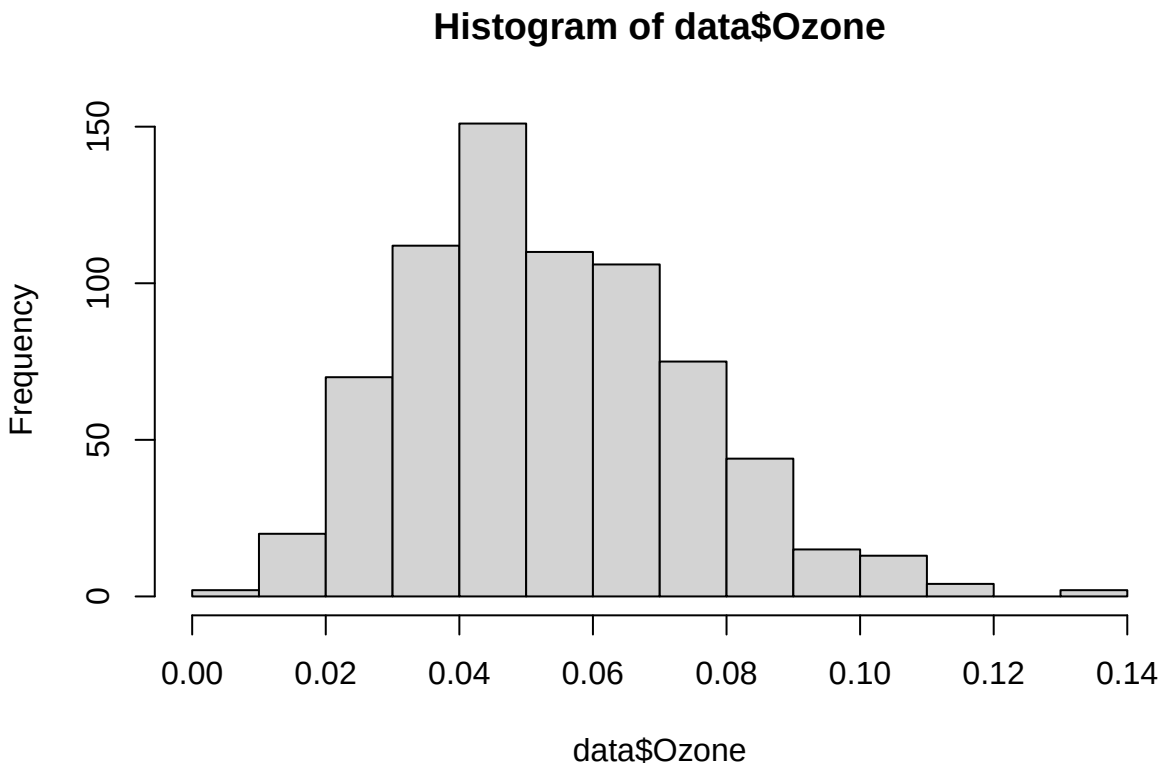
4. Details

- Give details on model specification, computation, and assumptions.

5. Value

- Give available variable names that can be called via the `$`
- For example we can assign object *junk* to be the outputs from the histogram

```
junk = hist(data$Ozone)
```



- We can then access various statistics/properties of the histogram.

```
junk$breaks ## the bin break points for the histogram
```

```
##  [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14
```

6. See Also

- Give other related functions

7. Examples

Additional information on working directories

Now suppose we want to change our working directory. In this example, I have a second copy of the same ozone dataset in a different location; the point of this exercise is to provide an example of changing the working directory.

You will have to modify the code below to get this to work on your computer.

To change the working directory in an .R script, you use the following:

```
# When you are running code in a .R file, you can use setwd:
setwd("~/Dropbox/EmoryCourses/BIOS_526/Materials_BRisk_2020/Data/")
```

In R markdown, you need to do this a special way. Details are below.

```
require(knitr)
```

```
## Loading required package: knitr
```

```
opts_knit$set(root.dir = "~/Dropbox/EmoryCourses/BIOS_526/Materials_BRisk_2020/Data")
data = read.csv("ozoneATL_2000.csv")
```

Then R remembers this working directory for all the code chunks that follow.

We can also use the full path name to get the dataset

```
data = read.csv("~/Dropbox/EmoryCourses/BIOS_526/Materials_BRisk_2020/Data/ozoneATL_2000.csv")
```

In R, everything is an object and every object has a *class*. We have specialized functions to manipulate, analyze, and plot different classes of objects. Some objects of data we will encounter often include data.frame, numeric, logical, matrix, and factor. We can check the class of our object using

```
class (data)
```

```
## [1] "data.frame"
```

We can view what objects are in the environment with

```
ls()
```

```
## [1] "county" "data"   "junk"   "ozone"  "p"      "x"      "y"
```

Here we only have an object called `data`.

(Option 2) If we have previously saved the R environment (in an .RData file), it can be loaded as

```
#load("mydrive/mydirectory/OzoneATL_2000.RData")
# since we set the directory with the special code for knit (opts_knit$set), we can load this file from
load("OzoneATL_2000.RData")
ls()
```

```
## [1] "county" "data"   "junk"   "ozone"  "p"      "x"      "y"
```

Alternatively, you can click on the *File* tab, select *Load workspace*, and navigate to your file.

Useful Resources

- http://www.statmethods.net
- https://science.nature.nps.gov/im/datamgmt/statistics/R/documents/R_for_SAS_SPSS_users.pdf
- Short Reference Card http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- http://www.ats.ucla.edu/stat/r/
- http://stackoverflow.com/
- Google