Base plotting system

Raphael Carvalho

25/03/2020

In another lesson, we gave you an overview of the three plotting systems in R. In this lesson we'll focus on the base plotting system and talk more about how you can exploit all its many parameters to get the plot you want. We'll focus on using the base plotting system to create graphics on the screen device rather than another graphics device.

The core plotting and graphics engine in R is encapsulated in two packages. The first is the graphics package which contains plotting functions for the "base" system. The functions in this package include plot, hist, boxplot, barplot, etc. The second package is grDevices which contains all the code implementing the various graphics devices, including X11, PDF, PostScript, PNG, etc.

Base graphics are often constructed piecemeal, with each aspect of the plot handled separately through a particular function call. Usually you start with a plot function (such as plot, hist, or boxplot), then you use annotation functions (text, abline, points) to add to or modify your plot.

Before making a plot you have to determine where the plot will appear and what it will be used for. Is there a large amount of data going into the plot? Or is it just a few points? Do you need to be able to dynamically resize the graphic?

What do you think is a disadvantage of the Base Plotting System?

[x] You can't go back once a plot has started

[ ] It's intuitive and exploratory

[ ] It mirrors how we think of building plots and analyzing data

[ ] A complicated plot is a series of simple R commands

Yes! The base system is very intuitive and easy to use. You can't go backwards, though, say, if you need to readjust margins or have misspelled a caption. A finished plot will be a series of R commands, so it's difficult to translate a finished plot into a different system.

Calling a basic routine such as plot(x, y) or hist(x) launches a graphics device (if one is not already open) and draws a new plot on the device. If the arguments to plot or hist are not of some special class, then the default method is called.

As you'll see, most of the base plotting functions have many arguments, for example, setting the title, labels of axes, plot character, etc. Some of the parameters can be set when you call the function or they can be added later in a separate function call.

Now we'll go through some quick examples of basic plotting before we delve into gory details. We'll use the dataset airquality (part of the library datasets) which we've loaded for you. This shows ozone and other air measurements for New York City for 5 months in 1973.

Use the R command head with airquality as an argument to see what the data looks like.

```r
head(airquality)
```

```
##   X Ozone Solar.R Wind Temp Month Day
## 1 1    41     190  7.4   67     5   1
## 2 2    36     118  8.0   72     5   2
## 3 3    12     149 12.6   74     5   3
## 4 4    18     313 11.5   62     5   4
## 5 5    NA      NA 14.3   56     5   5
## 6 6    28      NA 14.9   66     5   6
```

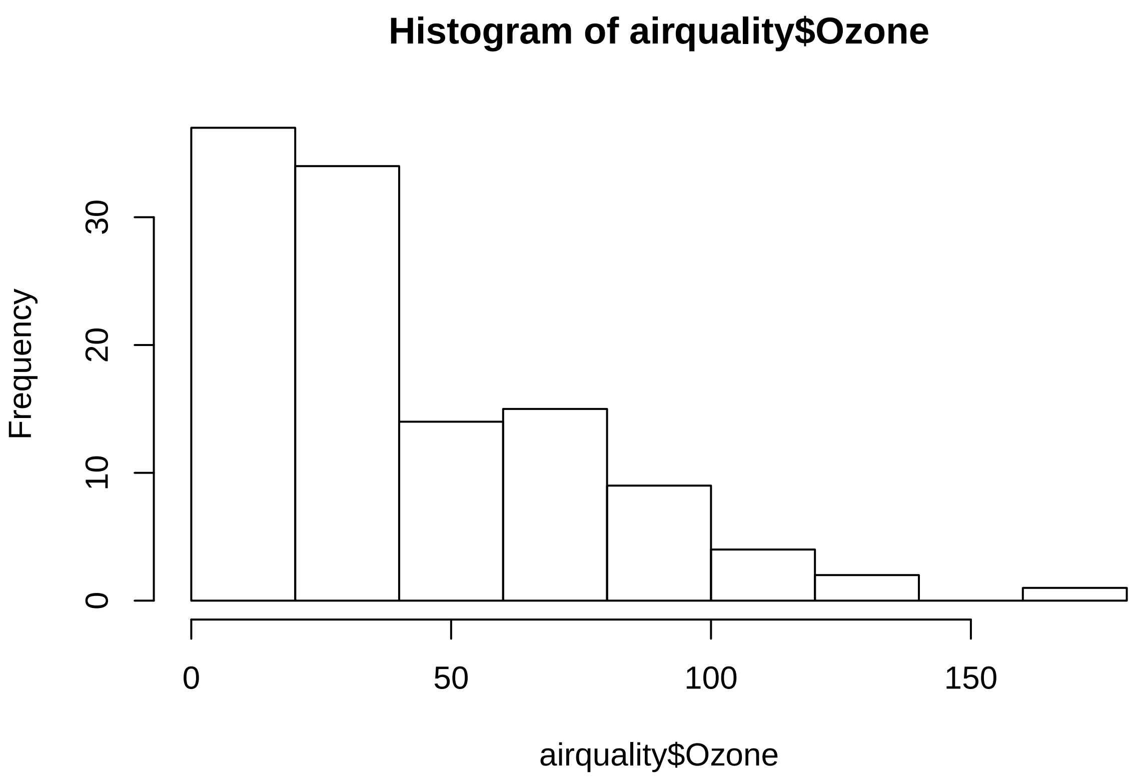
We see the dataset contains 6 columns of data. Run the command range with two arguments. The first is the ozone column of airquality, specified by airquality$Ozone, and the second is the boolean na.rm set equal to TRUE. If you don't specify this second argument, you won't get a meaningful result.

```r
range(airquality$Ozone, na.rm = TRUE)
```

```
## [1]   1 168
```

So the measurements range from 1 to 168. First we'll do a simple histogram of this ozone column to show the distribution of measurements. Use the R command hist with the argument airquality$Ozone.

```r
hist(airquality$Ozone)
```



Simple, right? R put a title on the histogram and labeled both axes for you. What is the most frequent count?

[x] **Under 25**

[ ] Over 150

[ ] Over 100

[ ] Between 60 and 75

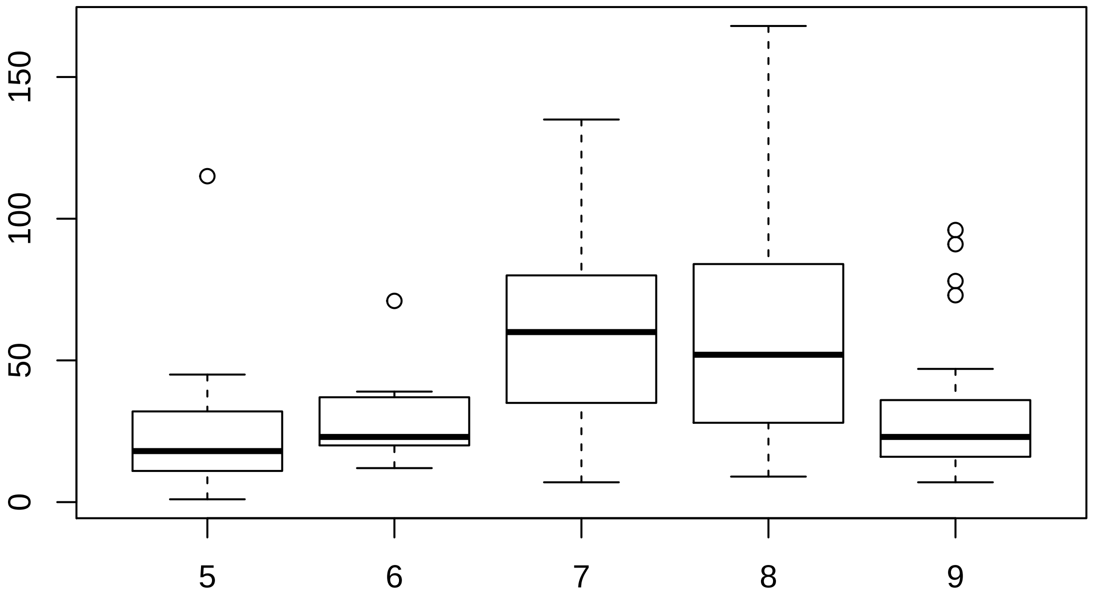
Next we'll do a boxplot. First, though, run the R command table with the argument airquality$Month.

```r
table(airquality$Month)
```

```
##
##  5  6  7  8  9
## 31 30 31 31 30
```

We see that the data covers 5 months, May through September. We'll want a boxplot of ozone as a function of the month in which the measurements were taken so we'll use the R formula Ozone~Month as the first argument of boxplot. Our second argument will be airquality, the dataset from which the variables of the first argument are taken. Try this now.

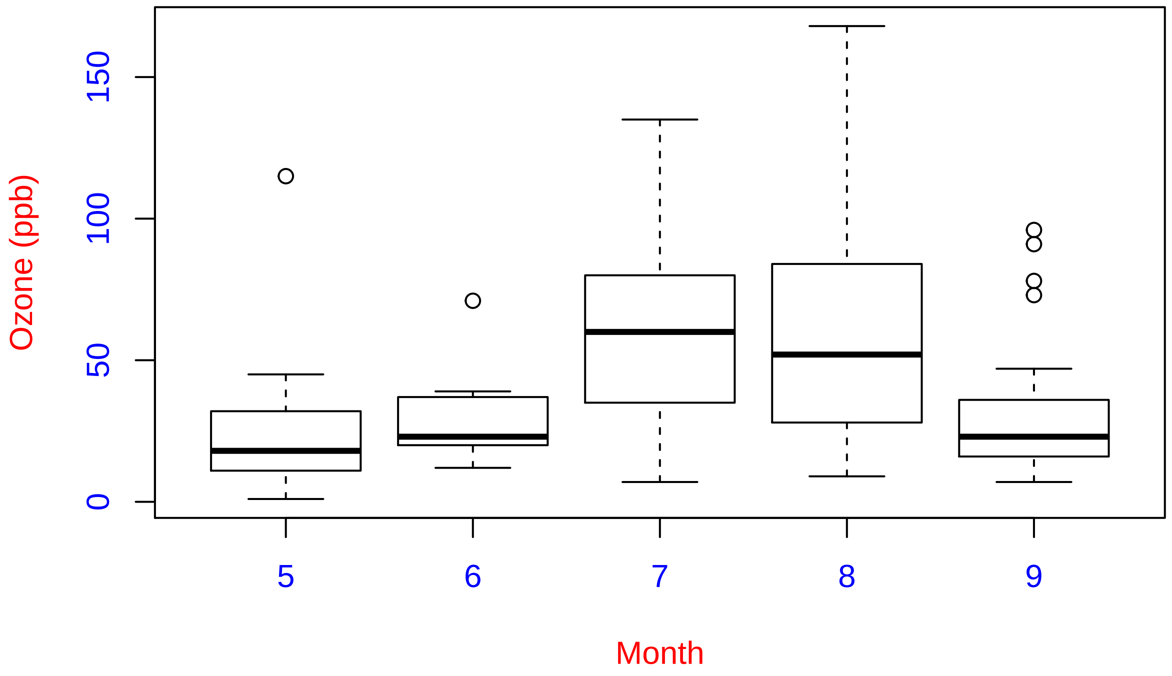
```
boxplot(Ozone~Month, airquality)
```



Note that boxplot, unlike hist, did NOT specify a title and axis labels for you automatically.

Let's call boxplot again to specify labels. (Use the up arrow to recover the previous command and save yourself some typing.) We'll add more arguments to the call to specify labels for the 2 axes. Set xlab equal to "Month" and ylab equal to "Ozone (ppb)". Specify col.axis equal to "blue" and col.lab equal to "red". Try this now.

```
boxplot(Ozone~Month, airquality, xlab = "Month", ylab = "Ozone (ppb)", col.axis = "blue", col.lab = "re
```



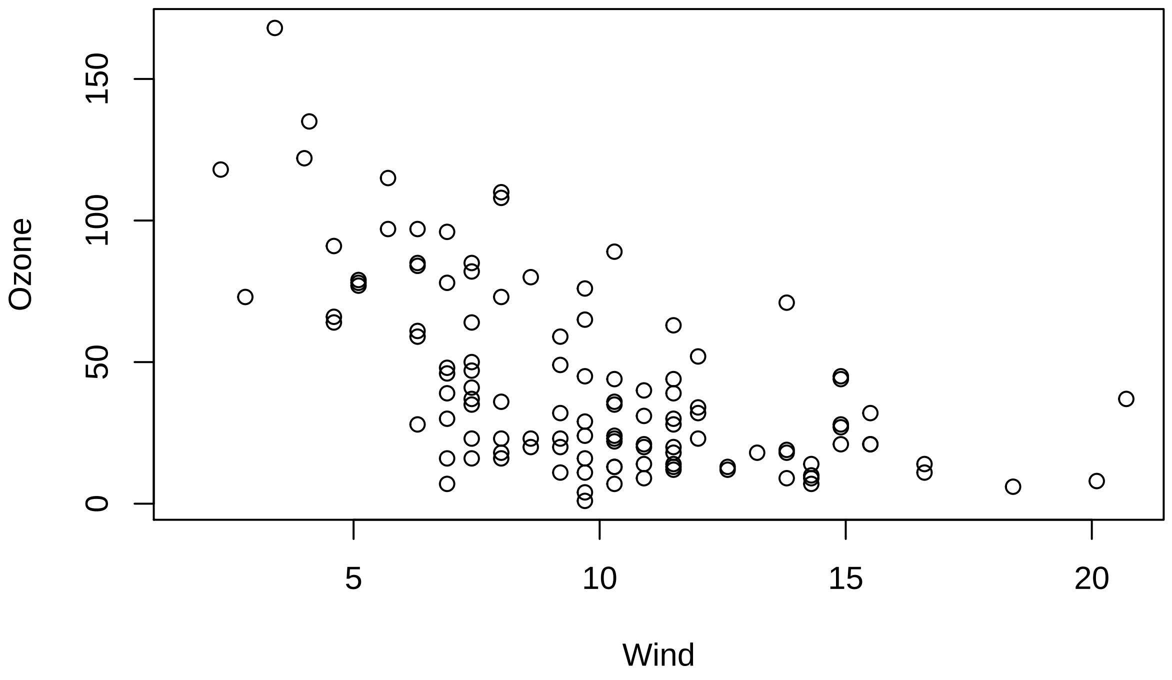
Nice colors, but still no title. Let's add one with the R command title. Use the argument main set equal to the string "Ozone and Wind in New York City".

```
title(main = "Ozone and Wind in New York City")
```

```
## Error in title(main = "Ozone and Wind in New York City"): plot.new has not been called yet
```

Now we'll show you how to plot a simple two-dimensional scatterplot using the R function plot. We'll show the relationship between Wind (x-axis) and Ozone (y-axis). We'll use the function plot with those two arguments (Wind and Ozone, in that order). To save some typing, though, we'll call the R command with using 2 arguments. The first argument of with will be airquality, the dataset containing Wind and Ozone; the second argument will be the call to plot. Doing this allows us to avoid using the longer notation, e.g., airquality$Wind. Try this now.

```
with(airquality, plot(Wind, Ozone))
```



Note that plot generated labels for the x and y axes but no title.

Add one now with the R command title. Use the argument main set equal to the string "Ozone and Wind in New York City". (You can use the up arrow to recover the command if you don't want to type it.).

```
title(main="Ozone and Wind in New York City")
```

```
## Error in title(main = "Ozone and Wind in New York City"): plot.new has not been called yet
```

The basic plotting parameters are documented in the R help page for the function par. You can use par to set parameters OR to find out what values are already set. To see just how much flexibility you have, run the R command length with the argument par() now.

```
length(par())
```

```
## [1] 72
```

So there are a boatload (72) of parameters that par() gives you access to. Run the R function names with par() as its argument to see what these parameters are.

```
names(par())
```

```
##  [1] "xlog"      "ylog"      "adj"       "ann"       "ask"
##  [6] "bg"        "bty"       "cex"       "cex.axis"  "cex.lab"
## [11] "cex.main"  "cex.sub"   "cin"       "col"       "col.axis"
## [16] "col.lab"   "col.main"  "col.sub"   "cra"       "crt"
## [21] "csi"       "cxy"       "din"       "err"       "family"
## [26] "fg"        "fig"       "fin"       "font"      "font.axis"
## [31] "font.lab"  "font.main" "font.sub"  "lab"       "las"
## [36] "lend"      "lheight"   "ljoin"     "lmitre"    "lty"
## [41] "lwd"       "mai"       "mar"       "mex"       "mfcol"
## [46] "mfg"       "mfrow"     "mgp"       "mkh"       "new"
## [51] "oma"       "omd"       "omi"       "page"      "pch"
## [56] "pin"       "plt"       "ps"        "pty"       "smo"
## [61] "srt"       "tck"       "tcl"       "usr"       "xaxp"
## [66] "xaxs"      "xaxt"      "xpd"       "yaxp"      "yaxs"
## [71] "yaxt"      "ylbias"
```

Variety is the spice of life. You might recognize some of these such as col and lwd from previous swirl lessons. You can always run ?par to see what they do. For now, run the command par()$pin and see what you get.

```
par()$pin
```

```
## [1] 5.26 2.66
```

Alternatively, you could have gotten the same result by running par("pin") or par('pin). What do you think these two numbers represent?

[x] **Plot dimensions in inches**

[ ] Random numbers

[ ] A confidence interval

[ ] Coordinates of the center of the plot window

Now, run the command par("fg") or par('fg') or par()$fg and see what you get.

```
par()$fg
```

```
## [1] "black"
```

It gave you a color, right? Since par()$fgspecifiesforegroundcolor, whatdoyouthinkpar()$bg specifies?

```
par()$bg
```

```
## [1] "transparent"
```

[ ] Better color

[ ] blue-green

[ ] Beautiful color

[x] **Background color**

Many base plotting functions share a set of parameters. We'll go through some of the more commonly used ones now. See if you can tell what they do from their names.

What do you think the graphical parameter pch controls?

[ ] point control height

[ ] pc help

[ ] picture characteristics

[x] **plot character**

The plot character default is the open circle, but it "can either be a single character or an integer code for one of a set of graphics symbols." Run the command par("pch") to see the integer value of the default. When you need to, you can use R's Documentation (?pch) to find what the other values mean.

```r
par()$pch
```

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

[ ] line length and width

[x] line type and width

[ ] line slope and intercept

[ ] line width and type

Run the command par("lty") to see the default line type

```r
par()$lty
```

```
## [1] "solid"
```

So the default line type is solid, but it can be dashed, dotted, etc. Once again, R's ?par documentation will tell you what other line types are available. The line width is a positive integer; the default value is 1.

We've seen a lot of examples of col, the plotting color, specified as a number, string, or hex code; the colors() function gives you a vector of colors by name.

What do you think the graphical parameters xlab and ylab control respectively?

[x] labels for the x- and y- axes

[ ] labels for the y- and x- axes

The par() function is used to specify global graphics parameters that affect all plots in an R session. (Use dev.off or plot.new to reset to the defaults.) These parameters can be overridden when specified as arguments to specific plotting functions. These include las (the orientation of the axis labels on the plot), bg (background color), mar (margin size), oma (outer margin size), mfrow and mfcol (number of plots per row, column).

The last two, mfrow and mfcol, both deal with multiple plots in that they specify the number of plots per row and column. The difference between them is the order in which they fill the plot matrix. The call mfrow will fill the rows first while mfcol fills the columns first.

So to reiterate, first call a basic plotting routine. For instance, plot makes a scatterplot or other type of plot depending on the class of the object being plotted.

As we've seen, R provides several annotating functions. Which of the following is NOT one of them?

[ ] points

[ ] lines

[ ] text

[x] hist

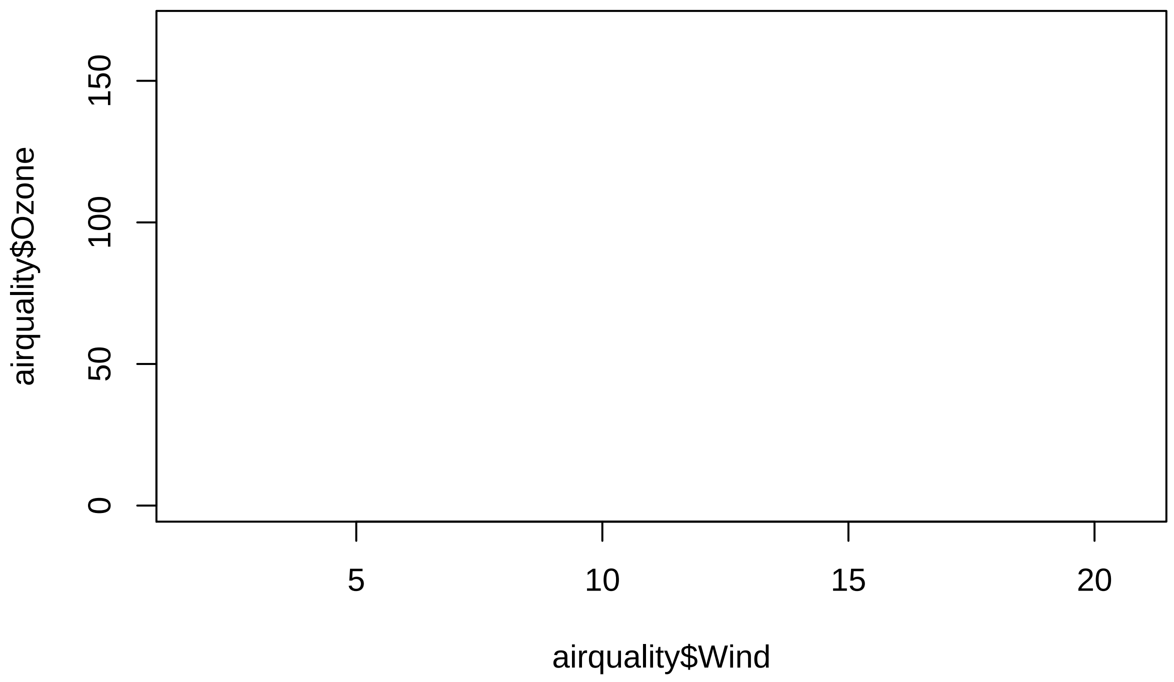
[ ] title

So you can add text, title, points, and lines to an existing plot. To add lines, you give a vector of x values and a corresponding vector of y values (or a 2-column matrix); the function lines just connects the dots. The function text adds text labels to a plot using specified x, y coordinates.

The function title adds annotations. These include x- and y- axis labels, title, subtitle, and outer margin. Two other annotating functions are mtext which adds arbitrary text to either the outer or inner margins of the plot and axis which adds axis ticks and labels. Another useful function is legend which explains to the reader what the symbols your plot uses mean.

Before we close, let's test your ability to make a somewhat complicated scatterplot. First run plot with 3 arguments. airquality$Wind, airquality$Ozone, and type set equal to "n". This tells R to set up the plot but not to put the data in it.

```
plot(airquality$Wind, airquality$Ozone, type="n")
```



Now for the test. (You might need to check R's documentation for some of these) Add a title with the argument main set equal to the string "Wind and Ozone in NYC"

```
title(main = "Wind and Ozone in NYC")
```

```
## Error in title(main = "Wind and Ozone in NYC"): plot.new has not been called yet
```

Now create a variable called may by subsetting airquality appropriately. (Recall that the data specifies months by number and May is the fifth month of the year.)

```
may <- subset(airquality, Month == 5)
```

Now use the R command points to plot May's wind and ozone (in that order) as solid blue triangles. You have to set the color and plot character with two separate arguments. Note we use points because we're adding to an existing plot.

```
points(may$Wind,may$Ozone,col="blue",pch=17)
```

```
## Error in plot.xy(xy.coords(x, y), type = type, ...): plot.new has not been called yet
```

Now create the variable notmay by subsetting airquality appropriately.

```
notmay <- subset(airquality, Month != 5)
```

Now use the R command points to plot these notmay's wind and ozone (in that order) as red snowflakes.

```
points(notmay$Wind,notmay$Ozone,col="red",pch=8)
```

```
## Error in plot.xy(xy.coords(x, y), type = type, ...): plot.new has not been called yet
```

Now we'll use the R command legend to clarify the plot and explain what it means. The function has a lot of arguments, but we'll only use 4. The first will be the string "topright" to tell R where to put the legend.

The remaining 3 arguments will each be 2-long vectors created by R's concatenate function, e.g., c(). These arguments are pch, col, and legend. The first is the vector (17,8), the second ("blue","red"), and the third ("May","Other Months"). Try it now.

```r
legend("topright", pch=c(17,8), col=c("blue", "red"), legend=c("May", "Other Months"))
```

```
## Error in strwidth(legend, units = "user", cex = cex, font = text.font): plot.new has not been called
```

Now add a vertical line at the median of airquality$Wind. Make it dashed (lty=2) with a width of 2.

```r
abline(v = median(airquality$Wind), lty=2, lwd = 2)
```

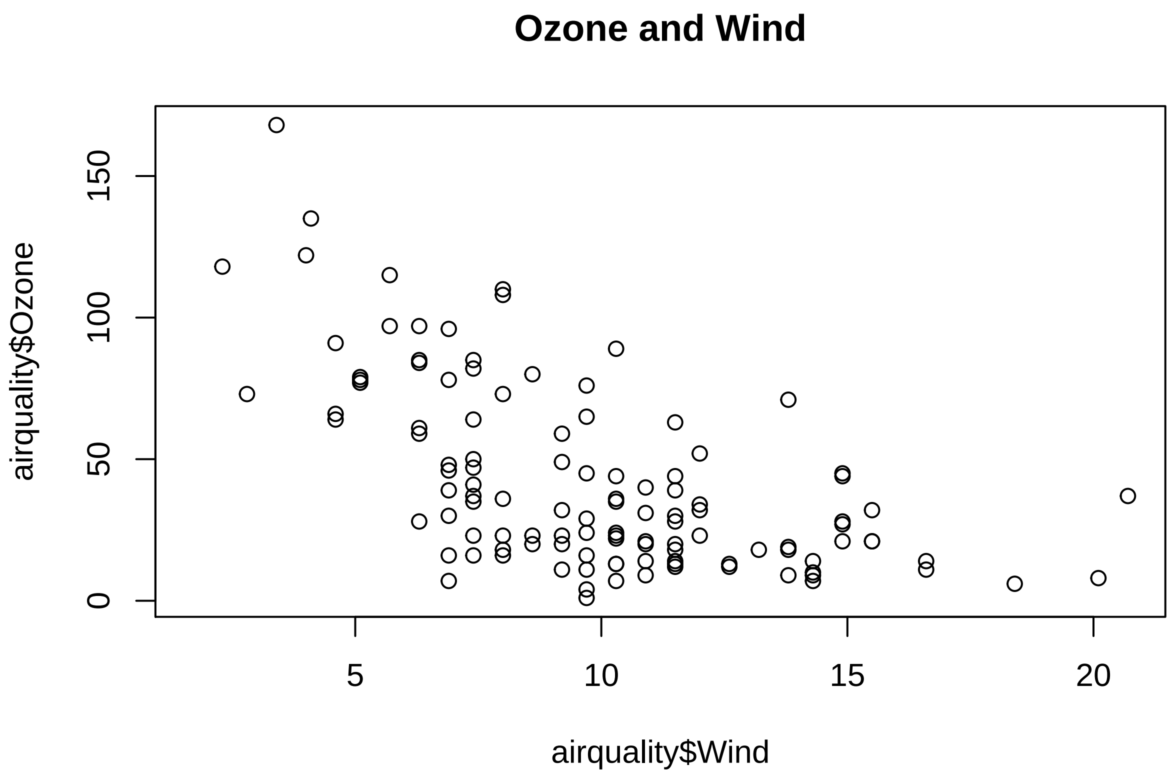
```
## Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): plot.new has not been called yet
```

Use par with the parameter mfrow set equal to the vector (1,2) to set up the plot window for two plots side by side. You won't see a result.

```r
par(mfrow=c(1, 2))
```

Now plot airquality$Windandairquality$Ozone and use main to specify the title "Ozone and Wind".

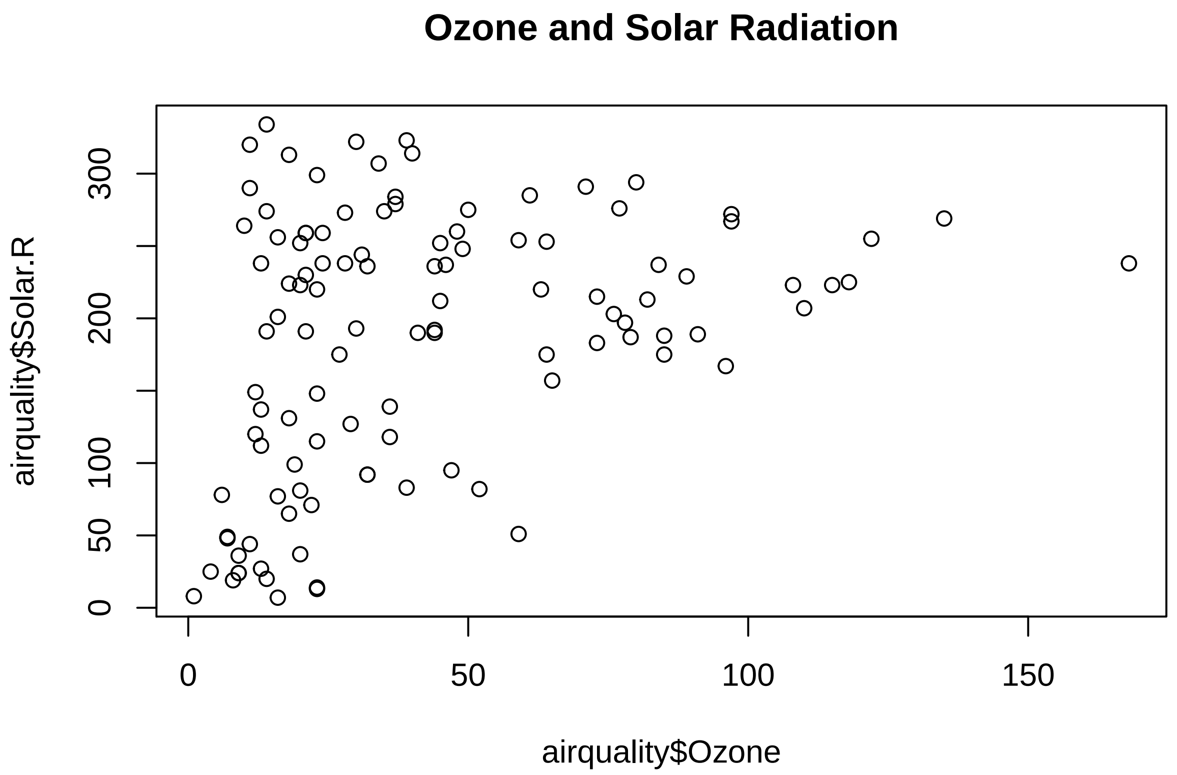
```r
plot(airquality$Wind, airquality$Ozone, main = "Ozone and Wind")
```



Now for the second plot.

Plot airquality$Ozoneandairquality$Solar.R and use main to specify the title "Ozone and Solar Radiation".

```r
plot(airquality$Ozone, airquality$Solar.R, main = "Ozone and Solar Radiation")
```



Now for something more challenging.

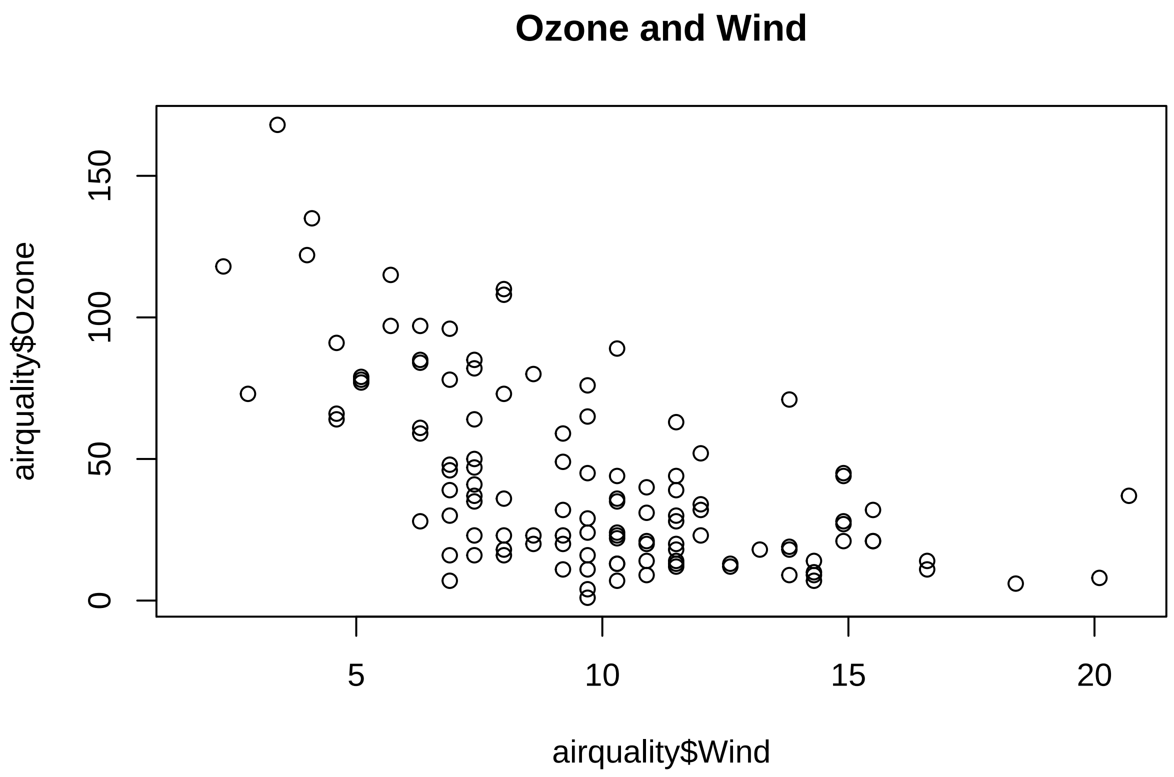
This one with 3 plots, to illustrate inner and outer margins. First, set up the plot window by typing par(mfrow = c(1, 3), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))

```r
par(mfrow = c(1, 3), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))
```

Margins are specified as 4-long vectors of integers. Each number tells how many lines of text to leave at each side. The numbers are assigned clockwise starting at the bottom. The default for the inner margin is c(5.1, 4.1, 4.1, 2.1) so you can see we reduced each of these so we'll have room for some outer text.

The first plot should be familiar. Plot airquality$Wind and airquality$Ozone with the title (argument main) as "Ozone and Wind".

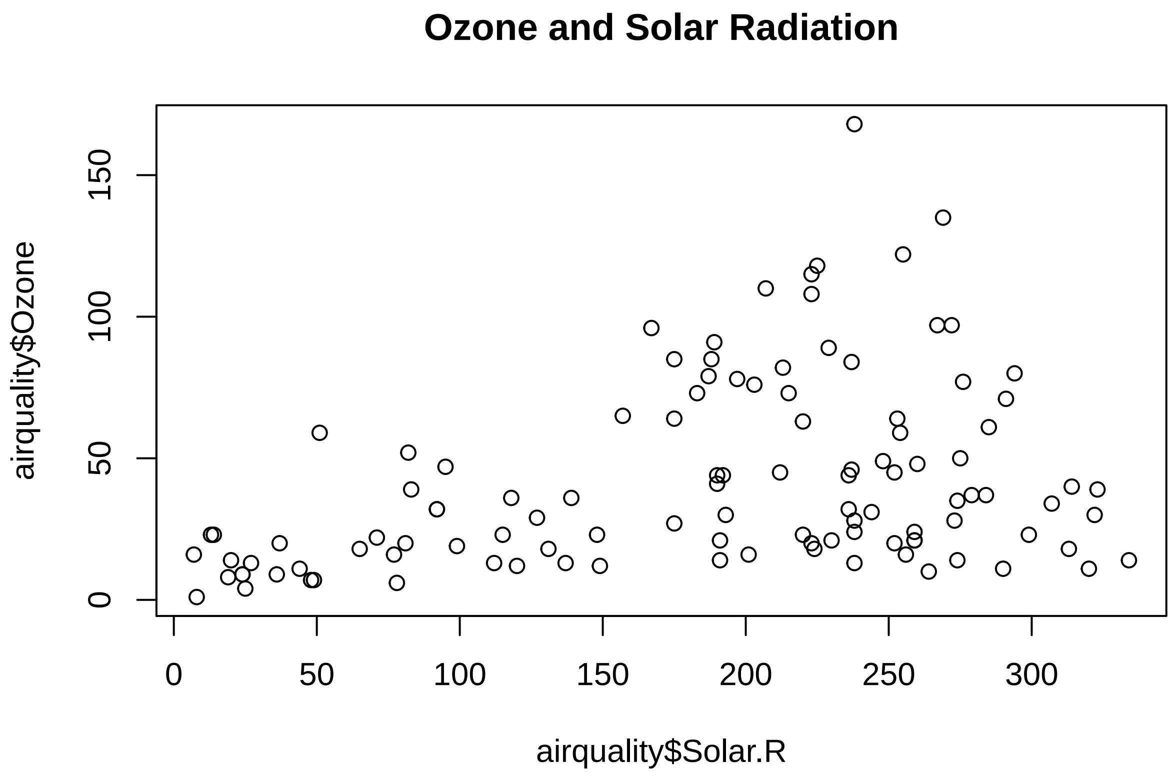
```r
plot(airquality$Wind, airquality$Ozone, main = "Ozone and Wind")
```



The second plot is similar

Plot airquality$Solar.R and airquality$Ozone with the title (argument main) as "Ozone and Solar Radiation"

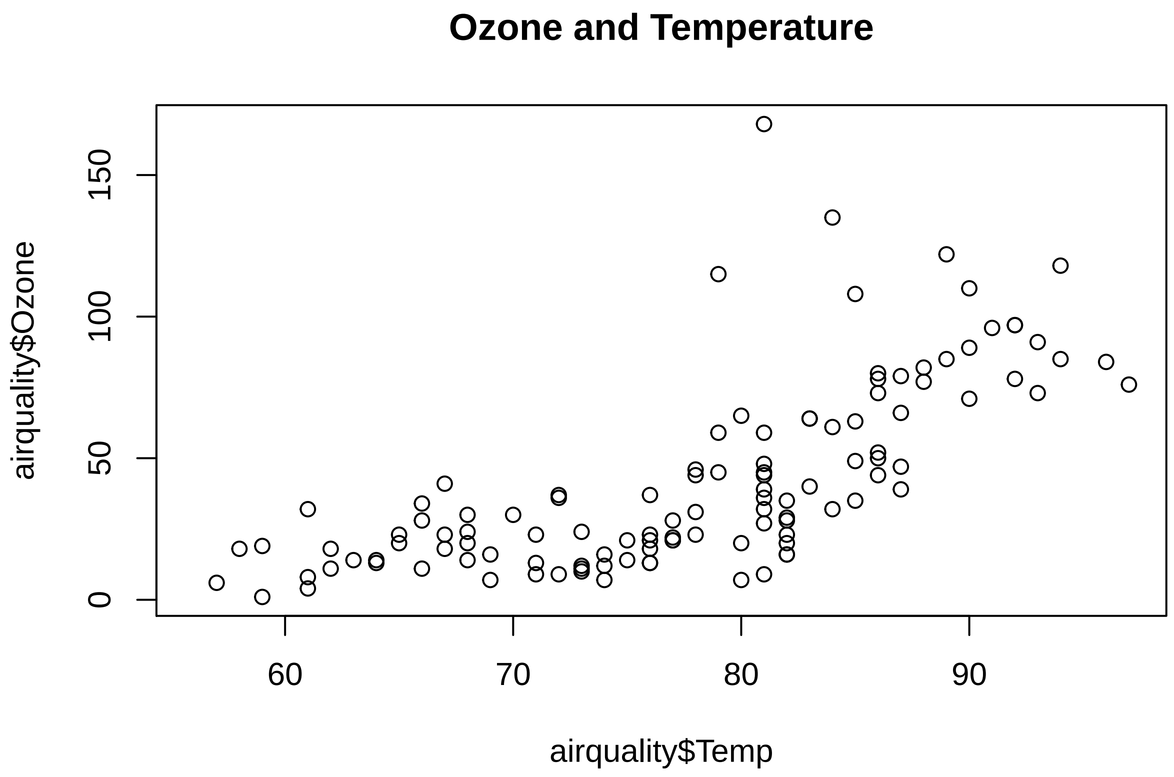
```
plot(airquality$Solar.R, airquality$Ozone, main = "Ozone and Solar Radiation")
```



Now for the final panel.

Plot airquality*Tempandairquality*Ozone with the title (argument main) as "Ozone and Temperature".

```r
plot(airquality$Temp, airquality$Ozone, main = "Ozone and Temperature")
```



Now we'll put in a title.

Since this is the main title, we specity it with the R command mtext. Call mtext with the string "Ozone and Weather in New York City" and the argument outer set equal to TRUE.

```r
mtext("Ozone and Weather in New York City", outer = TRUE)
```

```
## Error in mtext("Ozone and Weather in New York City", outer = TRUE): plot.new has not been called yet
```