MTH 3240 R Notes 8

8 Graphics

- R's built-in graphics functions can be split into two categories:
  - Plotting functions are used to ***create*** plots.
  - Plotting functions are used to ***add*** text, lines, points etc. to *existing* plots.

8.1 Creating Plots

- Here are some of the most commonly used *high-level* functions for creating plots:

```
plot()              # Scatterplot, time-series plot
hist()              # Histogram
boxplot()           # Boxplot(s)
stripchart()        # Dot plot, individual value plot
curve()             # Graph of a function
pairs()             # Scatterplot matrix
qqnorm()            # Normal probability plot (quantile-quantile plot)
barplot()           # Bar chart of specified bar heights
pie()               # Pie chart of specified pie areas
```

8.1.1 Scatterplots

- `plot()` takes arguments `x` and `y` and plots them in a ***scatterplot***. Some other arguments that can be passed to `plot()` are:

```
main            # Main title (in quotation marks)
sub             # Subtitle (in quotation marks)
xlab, ylab      # Labels for the x and y axes (in quotation marks)
xlim, ylim      # Limits for the x and y axes in the plot (in the form
                # c(lower, upper) )
type            # Type of plot that should be drawn (e.g. points, lines,
                # etc.)
...             # Other arguments such as the graphical parameters that
                # can be controlled by par()
```

- Here's an example using these vectors `my.x` and `my.y`:

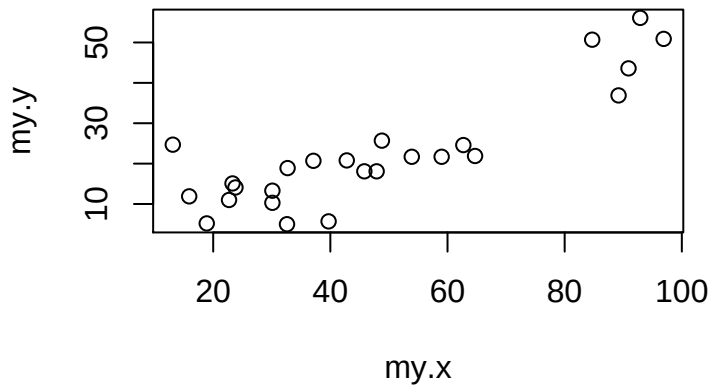
```
my.x
```

```
##  [1] 18.9 53.9 42.8 47.9 37.1 23.3 90.9 30.1 96.9 22.7
## [11] 15.9 45.8 59.0 89.2 30.1 92.9 32.6 84.7 64.7 48.8
## [21] 23.8 62.7 13.1 39.7 32.7
```

```
my.y
```

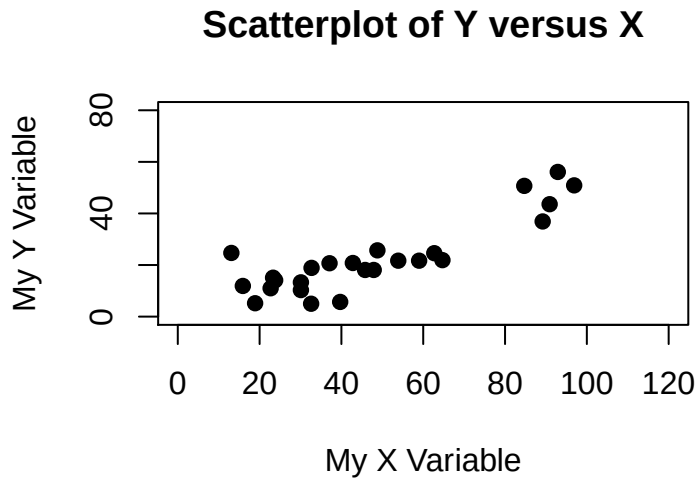
```
##  [1]  5.2 21.7 20.8 18.1 20.7 15.1 43.6 10.3 50.9 11.0
## [11] 11.9 18.1 21.7 36.9 13.3 56.1  5.0 50.7 21.9 25.7
## [21] 14.1 24.6 24.7  5.7 18.9
```

```
plot(x = my.x, y = my.y)
```



- Here's a nicer version of the plot:

```
plot(x = my.x, y = my.y,
     main = "Scatterplot of Y versus X",
     xlab = "My X Variable",
     ylab = "My Y Variable",
     xlim = c(0, 120),
     ylim = c(0, 80),
     pch = 19)
```

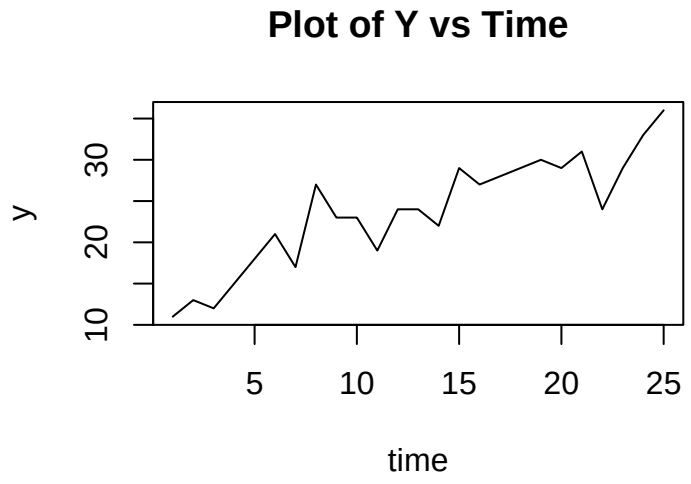


(The argument `pch`, for "plot character", is one of the '`...`' arguments that can be passed to `plot()` and that can also be set by the `par()` function. Specifying `pch = 19` indicates solid circles for the points. To see a list of the available point types, look for `pch` in the help file for `par()`.)

- In a ***time-series plot***, x represents time and the points are connected by lines. To make one, we specify `type = "l"` (the *letter* `"l"` for "line") in `plot()`. For example:

```
y

##  [1] 11 13 12 15 18 21 17 27 23 23 19 24 24 22 29 27 28 29
## [19] 30 29 31 24 29 33 36

time

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
## [19] 19 20 21 22 23 24 25
```

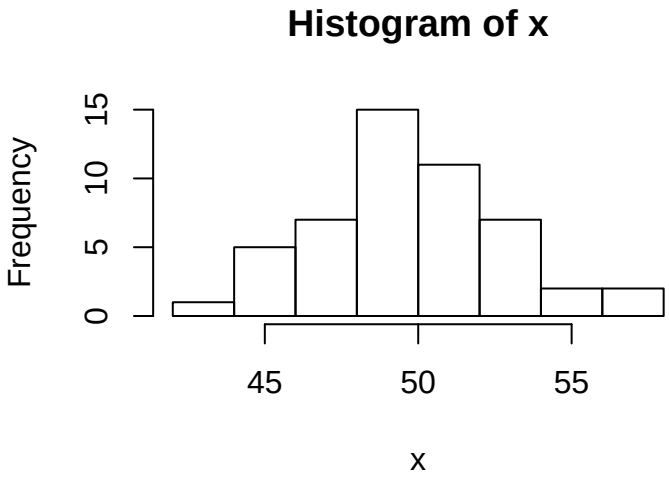
```
plot(x = time, y = y, type = "l", main = "Plot of Y vs Time")
```



8.1.2 Histograms and Boxplots

- `hist()` takes a vector argument `x` and produces a histogram of the data. For example:

```
# Simulate a random sample of size n = 50 from a N(50, 4) population:
x <- rnorm(n = 50, mean = 50, sd = 4)

hist(x)
```

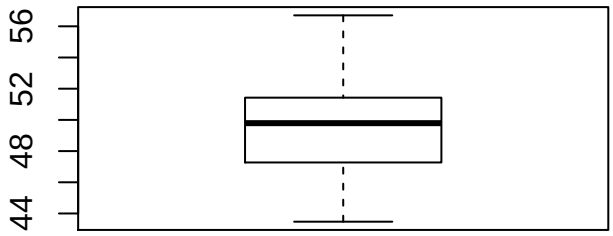


(The `rnorm()` function generates a random sample of size `n` from a *normal* distribution with a specified `mean` and standard deviation `sd`.)

- **Interpretation of a Histogram**:
  - The range of the data (smallest value to largest) is partitioned into intervals, or ***bins***, which correspond to the bases of the bars in the histogram.
  - The number of data values falling into a bin (i.e. the ***frequency*** for that bin) determines the height of the bar over the bin.

- `boxplot()` takes one or more vector arguments and produces the boxplot(s). For example:

```
boxplot(x)
```

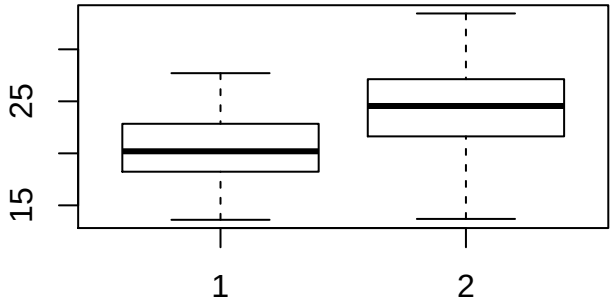


- **Interpretation of a Boxplot**:
  - Data values are represented along the vertical axis.
  - The top of the box is at the ***third quartile***, so 75% of the data values lie below the top of the box and 25% lie above it.
  - The bottom of the box is at the ***first quartile***, so 25% of the data values lie below the bottom of the box and 75% lie above it.
  - The horizontal line through the box is at the ***median***, so half of the data values lie below that line and half lie above it.
  - The "whiskers" (vertical lines) extend above and below the box to the largest and smallest data values, unless those values are *outliers*, in which case the whiskers only extend to the largest and smallest values that aren't *outliers*.

- To produce **side-by-side boxplots**, we pass multiple vectors to `boxplot()`. For example:

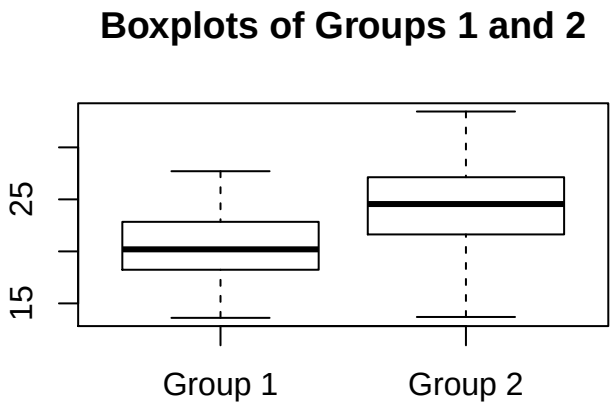
```
x1 <- rnorm(n = 50, mean = 20, sd = 3)
x2 <- rnorm(n = 40, mean = 25, sd = 4)
```

```
boxplot(x1, x2)
```



- We can include labels below the boxes via the `names` argument (and add a title via `main`):

```
boxplot(x1, x2,
        names = c("Group 1", "Group 2"),
        main = "Boxplots of Groups 1 and 2")
```

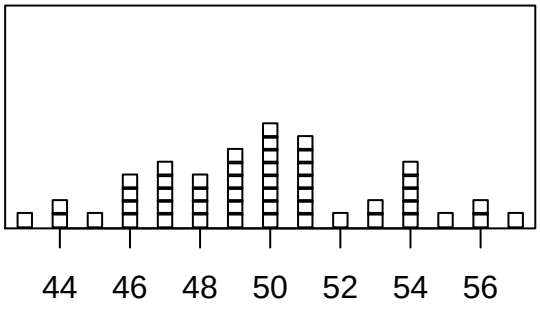


8.1.3 Dot Plots

- The function `stripchart()` will produce a dot plot of a data set if we specify `method = "stack"`. It's sometimes necessary to round the data values to get them to stack on top of

each other:

```
x <- round(x)
stripchart(x, method = "stack", at = 0)
```



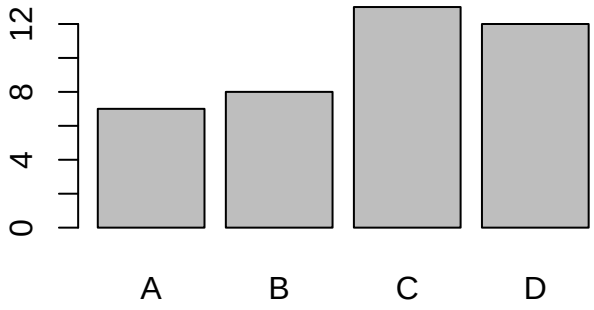
Specifying `at = 0` indicates that we want base of the stacks of dots to be "at" the horizontal axis ($y = 0$).

8.1.4 Bar Plots and Pie Charts

- ***Categorical*** (or ***qualitative***) data are usually displayed in bar plots or pie charts.
- `barplot()` takes a vector argument `height` containing bar heights and produces the bar plot. For example:

```
bar.hts <- c(7, 8, 13, 12)
```

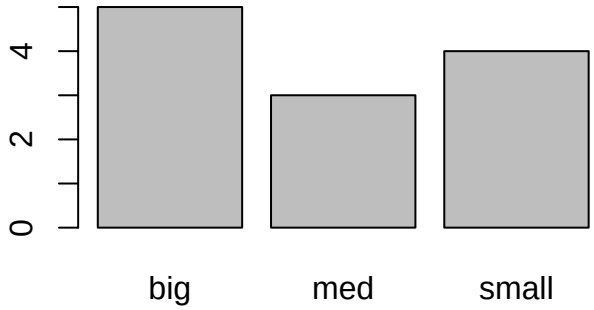
```
barplot(height = bar.hts, names.arg = c("A", "B", "C", "D"))
```



(The `names.arg` argument was used to the add labels below the bars.)

- The bar heights can be obtained using category **counts** from a `"character"` vector (or factor) using `table()`, and a *table* object can be passed directly to `barplot()`:

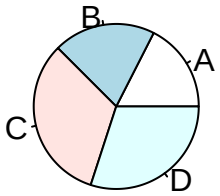
```
char.vec <- c("big", "big", "med", "small", "med", "big", "big", "small",
              "small",  "med", "big", "small")
my.tab <- table(char.vec)
my.tab

## char.vec
##   big   med small
##     5     3     4

barplot(my.tab)
```



- `pie()` takes a vector argument `x` indicating the *relative* sizes of the pie slices, and produces a pie chart. For example:

```
slice.sizes <- c(7, 8, 13, 12)
```

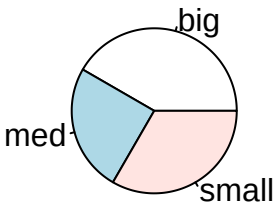
```
pie(x = slice.sizes, labels = c("A", "B", "C", "D"))
```



(The `labels` argument was used to the add labels to the slices.)

- The pie areas can be obtained using category **counts** from a `"character"` vector (or factor) using `table()`, and a *table* object can be passed directly to `pie()`. For example (using `char.vec` from above):

```
my.tab <- table(char.vec)
pie(my.tab)
```

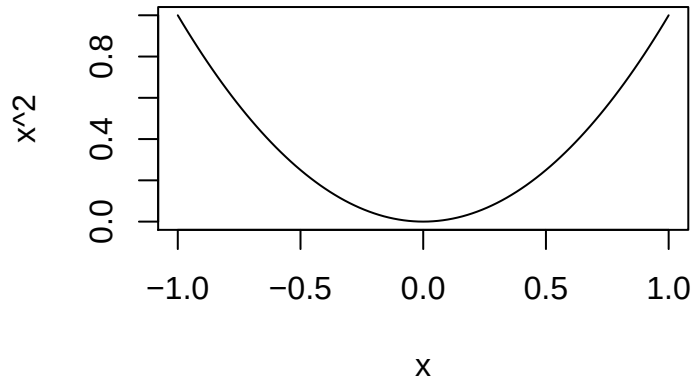


8.1.5 Graphing a Curve

- The best way to graph a curve in R is using `curve()`.
- `curve()` takes as its main argument either:
  - An expression involving a variable `x` and representing a mathematical function, (e.g. `x^2` or `1/x`),
    or
  - The name of an existing function in R (e.g. `log` or `sqrt`).

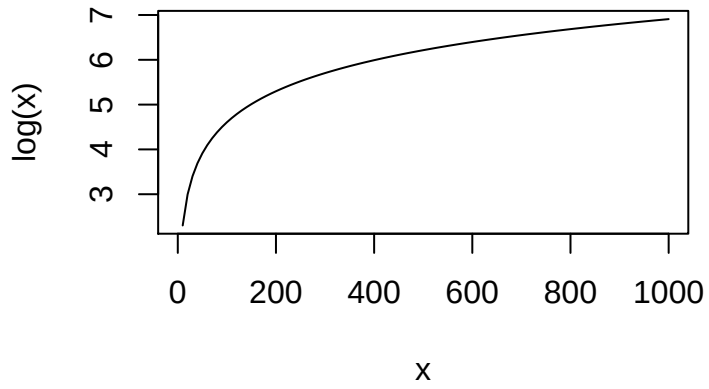
  It graphs the curve over an interval specified by the arguments `from` and `to`.
- For example, to graph $f(x) = x^2$ over the interval from -1 to 1, type:

```
curve(x^2, from = -1, to = 1)
```



and to graph the natural logarithmic function (`log()` in R), type:

```
curve(log, from = 0, to = 1000)
```



Section 8.1 Exercises

Exercise 1 The data set `state.x77` comes built-in with R. The vectors `illit` and `murder`, created below (from the 3rd and 5th columns of `state.x77`), contain illiteracy and murder rates for each of the 50 states in the U.S.:

```
illit <- state.x77[ , 3]
murder <- state.x77[ , 5]
```

a) Use `plot()` to make a scatterplot of the murder rates (y-axis) versus illiteracy rates (x-axis). Report your R command.

b) Modify your `plot()` command from part a using the arguments `main`, `xlab`, `ylab`, `xlim`, and `ylim` to add a main title and x and y axis labels, and to change the x and y axis limits. Report your R command.

Exercise 2 Recall that `table()` takes a `"character"` vector (or factor) argument and returns a *table* of category counts.

Both `barplot()` and `pie()` accept *tables* as arguments, and produce plots from the tabled counts.

A recent Gallup poll asked people if they smiled or laughed "a lot" on a given day. Here's a representative sampling of the responses:

```
laughed <- c("Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes",
             "Yes", "No", "Yes", "Yes", "Yes", "No", "No", "Yes", "Yes",
             "Yes", "No", "Yes")
```

a) Use `table()` to create a *table* from the `"character"` vector `laughed`:

```
my.tab <- table(laughed)
```

Then pass the table to `barplot()` to make a bar plot of the table counts. Report your R command(s).

b) Now pass the *table* to `pie()` to make a pie chart of the counts. Report your R command(s).

Exercise 3 Use `curve()`, with `from = -2` and `to = 2`, to graph the polynomial function

$$f(x) = 1 - 2x + x^2 + 3x^3$$

over the interval from -2 to 2 by typing:

```
curve(1 - 2*x + x^2 + 3*x^3,  from = -2, to = 2)
```

Describe the result.

8.2 Customizing Plots

8.2.1 Setting Graphical Parameters Using `par()`

- A number of plot features (***graphical parameters***) can be controlled using the function:

```
par()               # Get or set graphical parameters
```

- Here are just some of the graphical parameters that can be set by `par()`:

```
pch          # Plot character, or symbol type (an integer from 0 to 25)
cex          # Character expansion factor, i.e. size of plot characters
             # and/or text (values greater than 1 increase the size)
lty, lwd     # Line type (e.g. "dashed" or "solid") and line width
             # (values greater than 1 increase the width)
col          # Color of the objects being plotted (in quotation marks)
bg, fg       # Background and foreground colors (in quotation marks)
mfrow, mfcol # Multiple-figure plot arrangement (as a numerical vector
             # of the form c(nrow, ncol))
xaxt, yaxt   # x and y axis types (specify "n" for no axis)
tcl          # Length of axis tick marks (as a fraction of a line of
             # margin text)
bty          # Type of box drawn around the plot (specify "n" for none)
mar, mai     # Margin size in number of lines of margin text or inches
             # (a numerical vector of the form c(bottom, left, top,
             # right))
cex.main,
cex.axis,
cex.lab      # Character expansion factor (size of text) for main
             # title, axis annotations, and axis labels (values greater
             # than 1 increase their sizes)
```

For the full list, look at the help file for `par()`:

```
? par
```

- Setting a graphical parameter using `par()` affects *all subsequent plots* made during the current R session.
- As an example, to change plot symbols to triangles (`pch = 25`) that are blue (`col = "blue"`) and eliminate the box around our plots (`bty = "n"`), we can type:

```
par(pch = 25, col = "blue", bty = "n")
```

Now *all subsequent plots* will have these features (until we either reset them using `par()` again or end the R session). This example uses `my.x` and `my.y` from above:

```
plot(x = my.x, y = my.y)
```

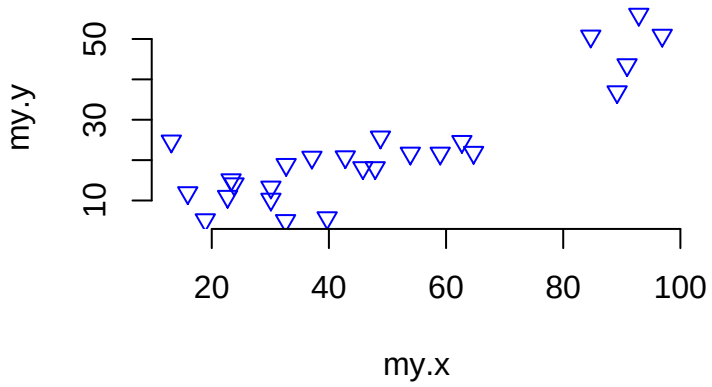


Figure 1

To reset the graphical parameters back to their original settings, we type:

```
par(pch = 1, col = "black", bty = "o")     # Returns the graphical parameters
                                           # to their original settings.
```

(We could also reset them by ending the current R session.)

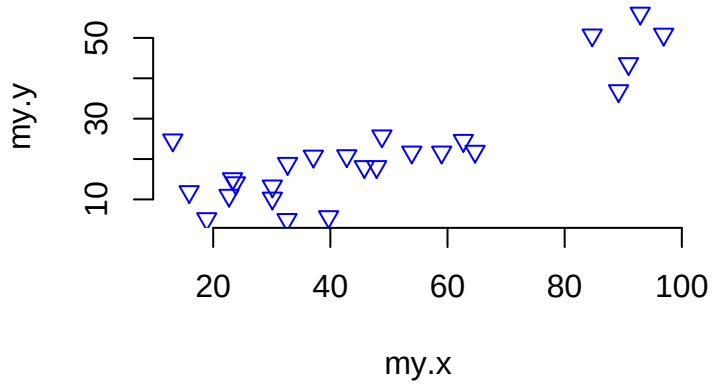
- To see the current settings for all of the graphical parameters, type:

```
par()       # Check the current graphical parameter settings.
```

8.2.2 Setting Graphical Parameters in `plot()` (and Other Plotting Functions)

- Most of `par()`'s graphical parameters can also be passed as arguments to `plot()` (and to `hist()`, `barplot()`, etc.). In this case it *only* affects the *current plot*.
- For example, we can duplicate the scatterplot of Fig. 1 by specifying `pch = 25`, `col = "blue"`, and `bty = "n"` in `plot()` (instead of in `par()`):

```
plot(x = my.x, y = my.y,
     pch = 25,
     col = "blue",
     bty = "n")
```



(In this case, *only* the *current plot* is affected, so we don't need to reset `pch`, `col`, and `bty` back to their original settings.)

Section 8.2 Exercises

Exercise 4 This exercise illustrates the difference between setting a graphical parameter using `par()` versus setting it in a `plot()` command. Here are two sets of x and y data:

```
my.x <- c(2, 4, 7, 6, 9)
my.y <- c(19, 18, 21, 24, 25)
```

a) Setting a graphical parameter using `par()` affects *all subsequent plots* until either the graphical parameter is reset (using `par()` again) or the R session is terminated.

Upon running the following commands in order, what color will the points be in the *second* plot below?

```
par(col = "red")
plot(x = my.x, y = my.y)
plot(x = my.x, y = my.y)              # What color will this plot be?
par(col = "black")
```

(Make sure you run the last of the four commands above before proceeding.)

b) Most of the graphical parameters that can be set by `par()` can also be passed as arguments to `plot()` (and `hist()`, and `boxplot()`, etc.). In this case, it *only* affects the *current plot*.

What color will the points be in the *second* plot below?

```
plot(x = my.x, y = my.y, col = "red")
plot(x = my.x, y = my.y)              # What color will this plot be?
```

8.3 Adding to an Existing Plot

- Here are some of the functions for *adding* features to an *existing* plot:

```
points()     # Add points to the plot at specified coordinates
lines()      # Add a line to the plot connecting specified coordinates
polygon()    # Draw a polygon in the plot with a given set of vertices
abline()     # Add a line to the plot with given intercept a and
             # slope b
segments()   # Add line segments to the plot between pairs of points
arrows()     # Draw an arrow in the plot (with specified start and
             # end points)
text()       # Add text to the plot at a specified set of coordinates
mtext()      # Add text in a margin of the plot
legend()     # Add a legend to the plot
title()      # Add a main title to the plot (if it doesn't already
             # have one).  Can also be used to add x and y axis labels.
axis()       # Add an axis to the plot on a given side
curve()      # Add a curve to the plot (specify add = TRUE)
qqline()     # Add a line to a normal probability plot
box()        # Add a box around the plot (if one doesn't already exist)
rect()       # Draw a rectangle in the plot at a given set of
             # coordinates
symbols()    # Add various symbols to the plot (circles, squares,
             # etc.)
```

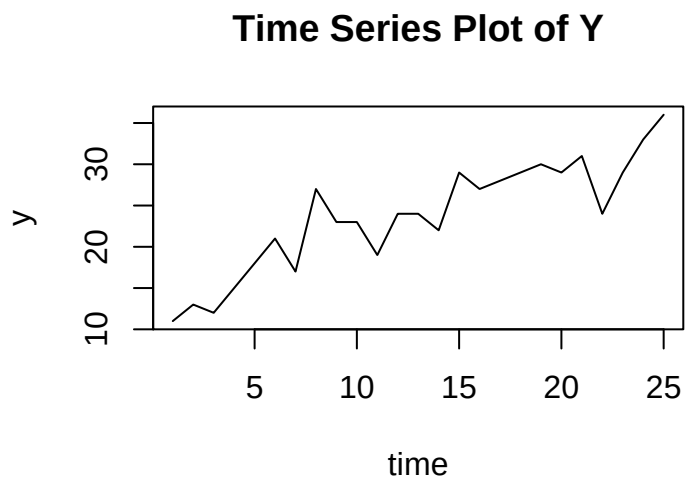
- In addition to their main arguments (see their help files), these functions also accept arguments representing graphical parameters that can be set by `par()` (e.g. `col`, `pch`, `cex`, `lwd`, etc.).
- As an example, here are two vectors:

```
time <- 1:25
```

```
y <- c(11, 13, 12, 15, 18, 21, 17, 27, 23, 23, 19, 24, 24, 22, 29,
       27, 28, 29, 30, 29, 31, 24, 29, 33, 36)
```

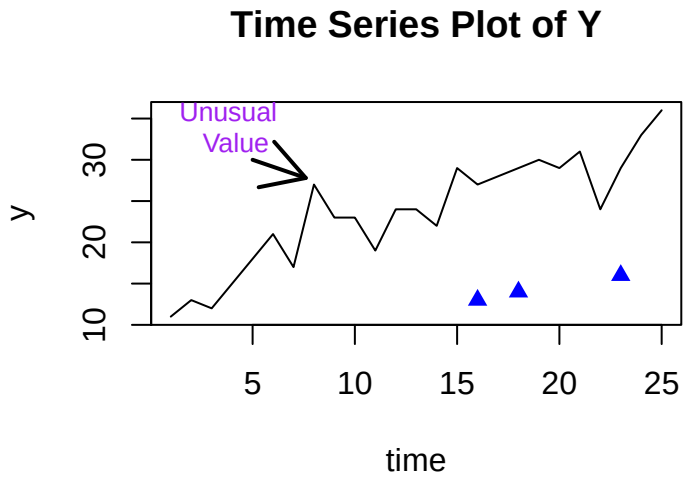
Consider the time-series plot:

```
plot(x = time, y = y,
     pch = 19,
     type = "l",
     main = "Time Series Plot of Y")
```



Below, we use `text()`, `arrows()`, and `points()` to add to the plot we created above:

```
text(x = 4, y = 34, labels = "Unusual \n Point", cex = 0.8, col = "purple")
arrows(x0 = 5, y0 = 30, x1 = 7.6, y1 = 27.8, lwd = 2)

new.x <- c(16, 18, 23)
new.y <- c(13, 14, 16)

points(x = new.x, y = new.y, pch = 17, col = "blue")
```



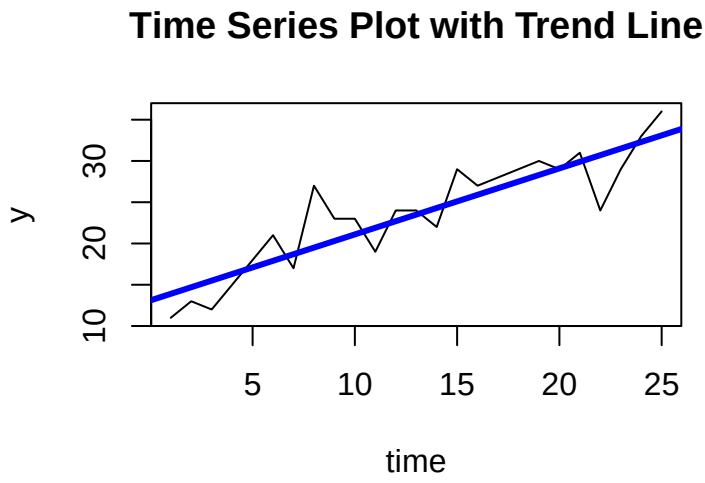
(The symbol `\n` in the call to `text()` above is a "new line" character.)

- As another example, consider again the time-series plot:

```
plot(x = time, y = y,
     type = "l",
     main = "Time Series Plot with Trend Line")
```

Below, we use `abline()` to add a trend line (with intercept $a = 13.1$ and slope $b = 0.8$):

```
abline(a = 13.1, b = 0.8, lwd = 3, col = "blue")
```



- Here's another example that uses `lines()` to add a new time-series (of `z`) to an *existing* time-series plot (of `y` and `time` from above) and uses `legend()` to add a legend:

```
z <- c(10, 12, 11, 14, 16, 15, 12, 16, 17, 15, 15, 11, 13, 12, 15,
17, 18, 19, 18, 15, 17, 19, 19, 14, 16)
```

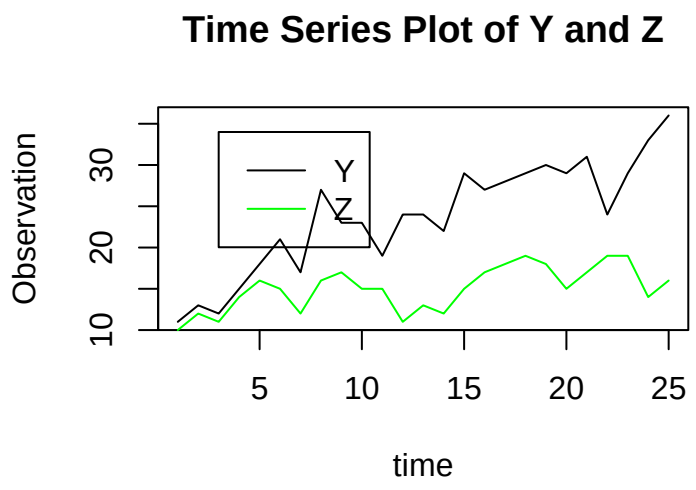
Here's the original time-series plot of `y` versus `time`:

```
plot(x = time, y = y,
     type = "l",
     ylab = "Observation",
     main = "Time Series Plot of Y and Z")
```

Now we add a time-series line for `z` and a legend by typing:

```
lines(x = time, y = z, col = "green")
```

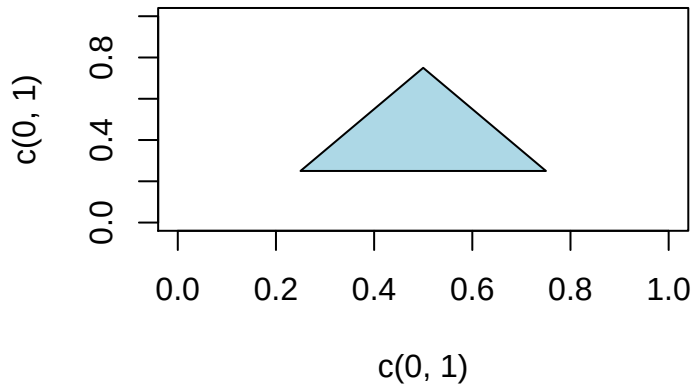
```
legend(x = 3, y = 34,
       legend = c("Y", "Z"),
       col = c("black", "green"),
       lty = c("solid", "solid"))
```



- `polygon()` takes vector arguments `x` and `y` representing coordinates of a polygon's vertices, and `col` giving a fill color, and adds the polygon to an existing plot.

Below, a blank plot is created first, and the polygon (triangle) is added to the blank plot:

```r
plot(x = c(0, 1), y = c(0, 1),       # Just set up a blank coordinate system
     col = "transparent")

polygon(x = c(0.25, 0.5, 0.75),      # Add the polygon (a triangle)
        y = c(0.25, 0.75, 0.25),
        col = "lightblue")
```



(Above, the `x` and `y` coordinates passed to `polygon()` are in clockwise order.)

Section 8.3 Exercises

Exercise 5 Suppose two variables, x and y, were measured on males and females:

```r
x.m <- c(1, 3, 2, 6, 5, 5, 3, 4)          # Males
y.m <- c(7, 7, 9, 6, 6, 8, 3, 5)


x.f <- c(4, 4, 3, 7, 6, 8, 7, 9)          # Females
y.f <- c(9, 11, 8, 8, 7, 8, 4, 5)
```

a) `points()` is useful for plotting different groups together in the same scatterplot using different plot characters or colors. After creating the vectors `x.m`, `y.m`, `x.f`, and `y.f` above, run the following commands and describe the result:

```
plot(x = x.m, y = y.m,
     xlim = c(1, 10), ylim = c(4, 12),
     pch = 19,
     col = "blue")
points(x = x.f, y = y.f,
     pch = 17,
     col = "red")
```

b) Now (after creating the plot of part *a*) execute the following command and describe the result:

```
legend(x = 8, y = 11,
       legend = c("Male", "Female"),
       pch = c(19, 17),
       col = c("blue", "red"))
```