CSI 2300: Introduction to Data Science

Lecture 08: Exploratory Data Analysis

Today's Topics

Line Plots

- Simple line plots
- Adding lines

Time Series

- Syntax
- Features
- Handling dates

Basic Legends

- Syntax

These are the packages that we need for today's lecture:

```
# first download the mowateR_0.2.tar.gz file then uncomment and run the following:
#install.packages("mowateR_0.2.tar.gz", repos = NULL, type = "source")
library(mowateR)
library(lubridate)
```

Line Plots

You have already seen how to create a plot of values with the `plot(x,y)` command. With `plot()`, you can create scatterplots, plots of lines and functions, and even time series (which you will see in the next section). Here, we will cover

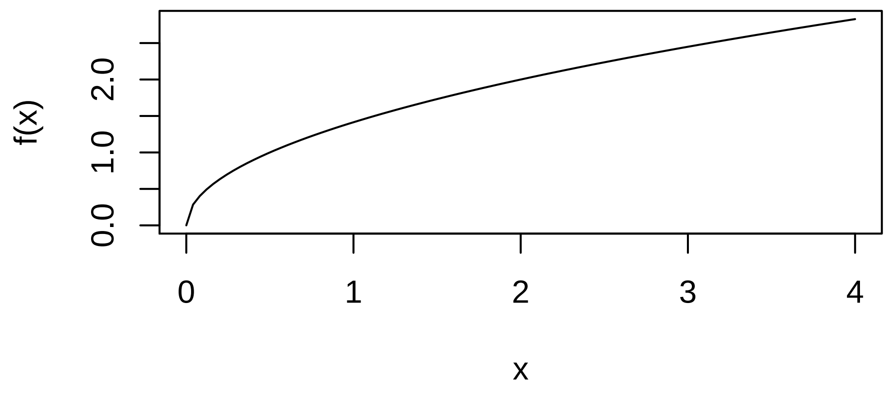
- some of the fundamentals of how this function works,
- introduce the `lines()` function to overlay a line on an existing plot and
- the `points()` function to add points to an existing plot, and
- show how to add a legend to a plot.

R is a big calculator, and like a graphing calculator, it can plot any type of function. It works by creating a fine grid of numbers for the horizontal values, and then supplying those numbers to the function that you wish to plot for the vertical values.

Example, Plotting a Basic Function: Plot the function

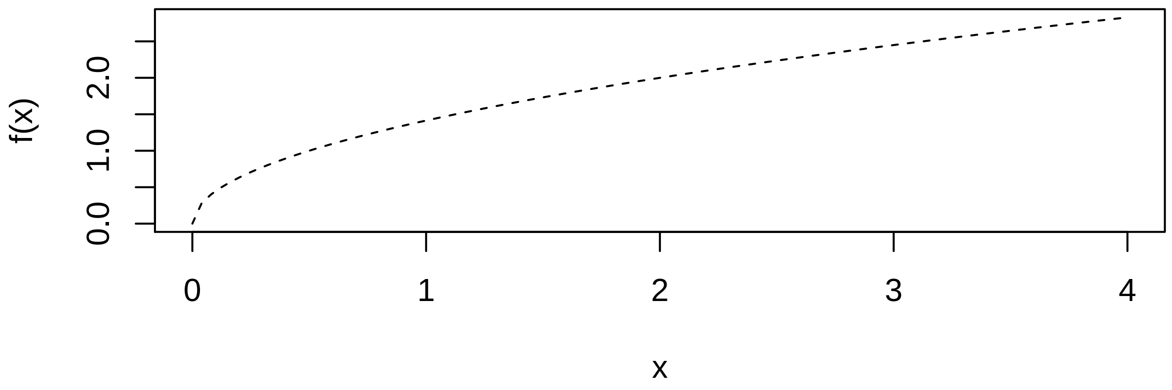
$$f(x) = \sqrt{2x}.$$

```r
xx <- seq(0, 4, len = 100)
yy <- sqrt(2*xx)
plot(xx, yy,
     type = "l",
     xlab = "x",
     ylab = "f(x)")
```



- What happens if you change the "grid" for the x values to be only of length 10? Of length 1000?
- The `type` argument in the `plot()` function allows you to change the type of plot to be drawn. Test some of these common choices, and see how it affects the plot:
  - "p" for points
  - "l" for lines
  - "b" for both
  - "n" for no plotting. This is useful when you want to just set up the plotting window and then overlay lines or points on it afterwards.

- The `lty` argument allows you to change the type of line from solid to dashed to dotted, etc. Try putting this argument in as follows:



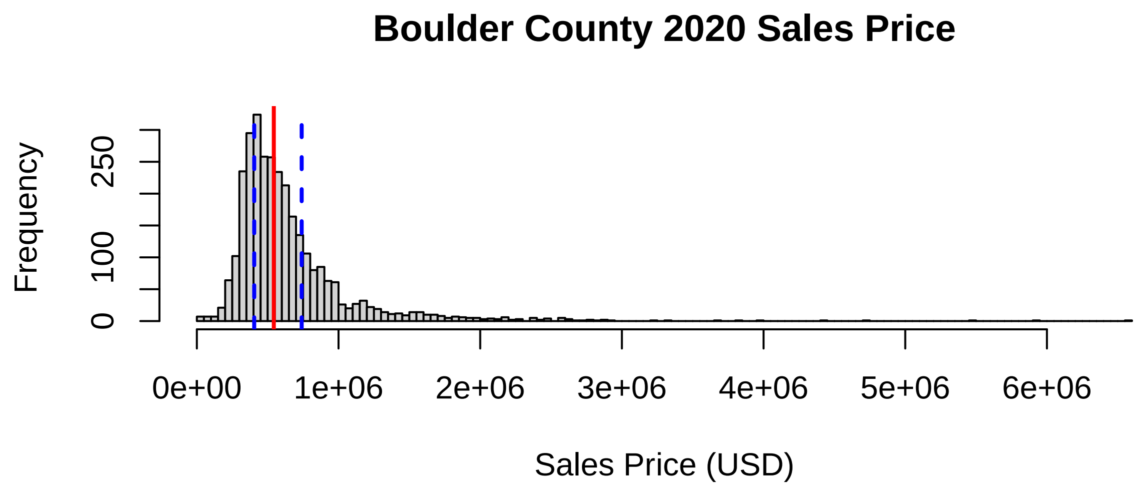
You can add a line to an existing plot with the `lines()` or `abline()` command.

- The `lines()` command requires both an x and a y argument, each of the same length.
- The `abline()` command can accept several different inputs, as follows:
  - `abline(v = some_number)` will create a vertical line at `some_number`
  - `abline(h = some_number)` will create a horizontal lines at `some_number`
  - `abline(intercept, slope)` will draw a line with that y-intercept and slope.

Example, Add a Line to a Plot: Plot the histogram of sales prices of homes in Boulder County in 2020. Overlay a vertical line at the 1st quartile, median, and 75th quartile.

```
hist(sales2020$SALE_PRICE, breaks="FD",
       main = "Boulder County 2020 Sales Price",
       xlab = "Sales Price (USD)")

abline(v = median(sales2020$SALE_PRICE),
       col = "red")
```

```r
#To make the line a little thicker so that it stands out
abline(v = median(sales2020$SALE_PRICE),
       col = "red",
       lwd = 2)

abline(v = quantile(sales2020$SALE_PRICE, 0.25),
       col = "blue",
       lwd = 2,
       lty = 2)
abline(v = quantile(sales2020$SALE_PRICE, 0.75),
       col = "blue",
       lwd = 2,
       lty = 2)
```

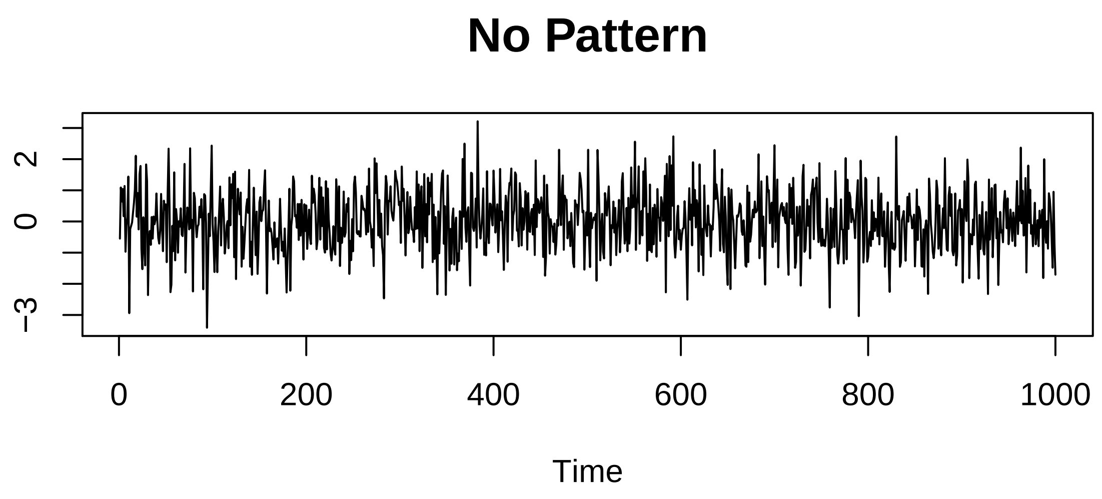


Time Series

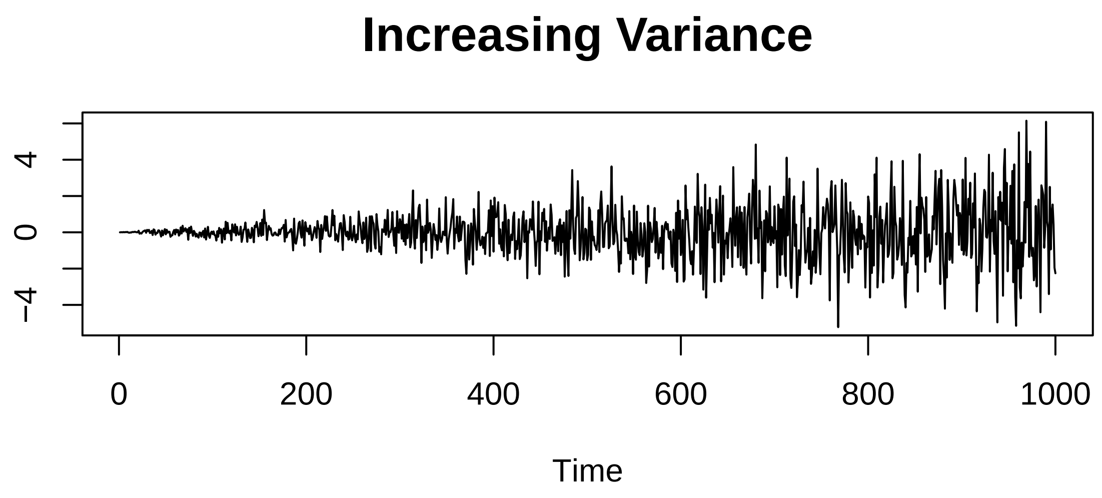
These plots are typically used as a way to process chronological information and visualize it in such a way as to make observations about the state of data over a specific time interval. The visual information typically is represented as a 2D plot with the observed value on the y-axis and time on the x-axis. The values of the variable plotted in the order in which they were observed, often connected by a line. Other plot forms (i.e. ways of connecting the individual data points) can be useful in conveying important information dependent on the situation constraints. Once constructed, some common strategies when analyzing the visual information is to look for repeating patterns or shifts in center or spread.

There are many features within 'R' for time series data, but most of these features are outside the scope of a beginner course (we might dive into them later on in the semester if time avails). For now, we focus on plotting the time series data and simple pattern analysis.

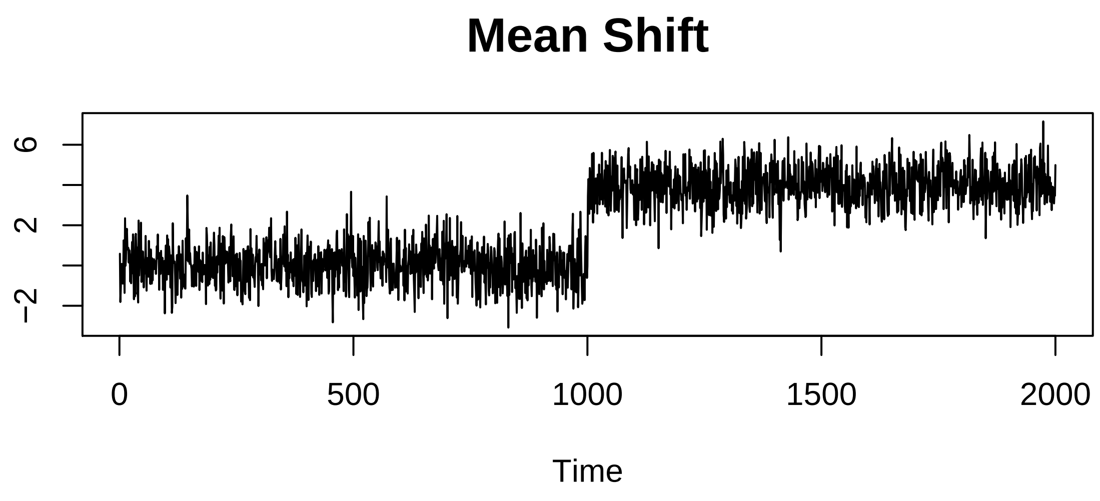
```r
load(file="dat/boulder_ammonia.rda")
set.seed(77)
nn <- 1000
ts.plot(rnorm(nn), ylab="")
title("No Pattern", cex.main = 1.5)
```



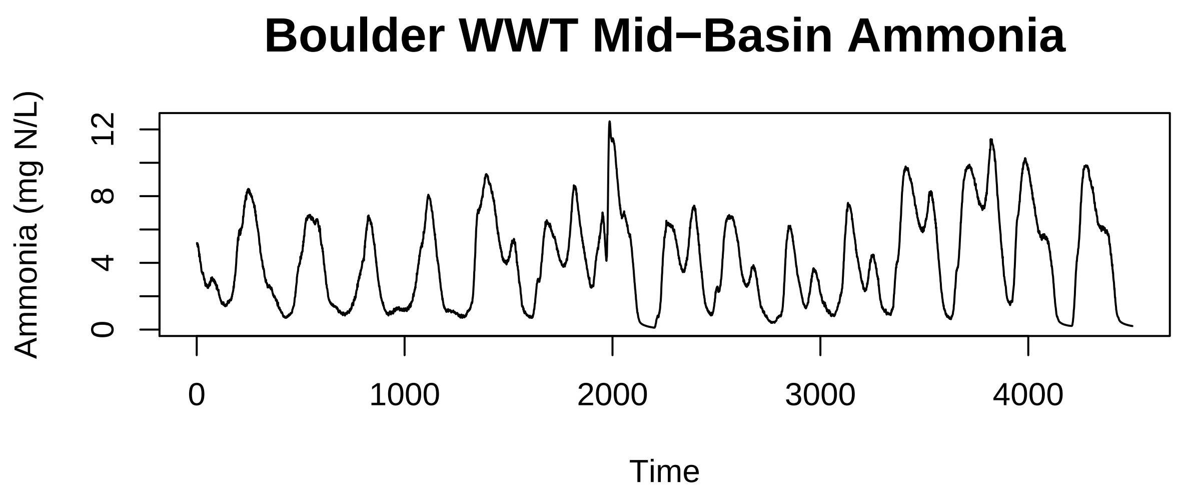
```r
xx<-seq(0, 2, len=nn)
increase.variance <- 1.25*xx
ts.plot(rnorm(nn, 0, increase.variance), ylab="")
title("Increasing Variance", cex.main = 1.5)
```



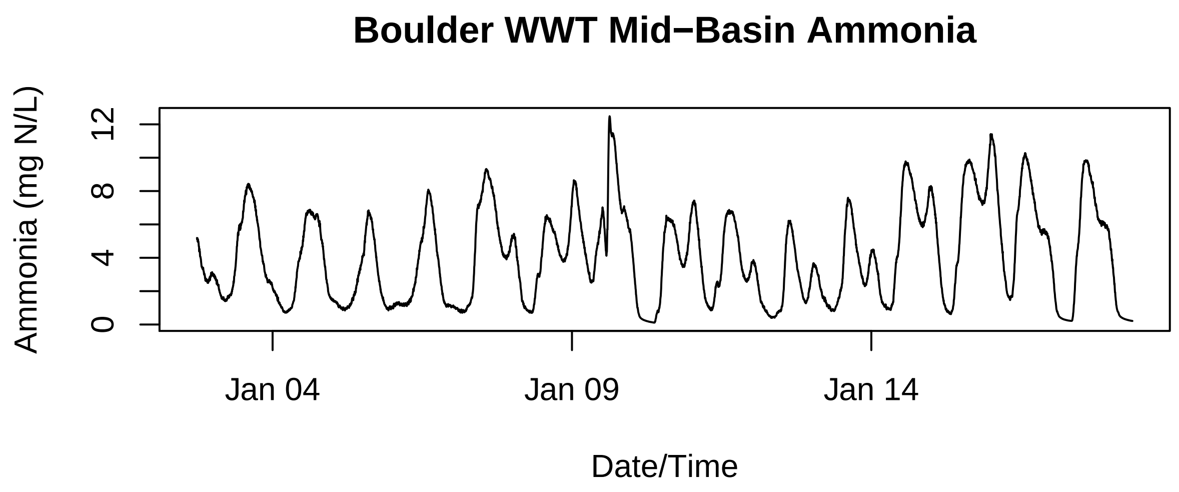
```r
data1<-rnorm(nn, 0, 1)
data2<-rnorm(nn, 4, 1)
ts.plot(c(data1, data2), ylab="")
title("Mean Shift", cex.main = 1.5)
```



```r
sub <- 500:5000
ammonia_sub <- boulder_ammonia$AB3.Z7.Ammonia.mg.N.L[sub]
date_time <- boulder_ammonia$Date.Time[sub]

ts.plot(ammonia_sub, ylab = "Ammonia (mg N/L)")
title("Boulder WWT Mid-Basin Ammonia", cex.main = 1.5)
```



```r
#This is another approach to create the same plot of ammonia
#but with the dates clearly labeled on the x-axis.
plot(date_time, ammonia_sub,
     xlab = "Date/Time",
     ylab =  "Ammonia (mg N/L)",
     type = "l",
     main = "Boulder WWT Mid-Basin Ammonia")
```

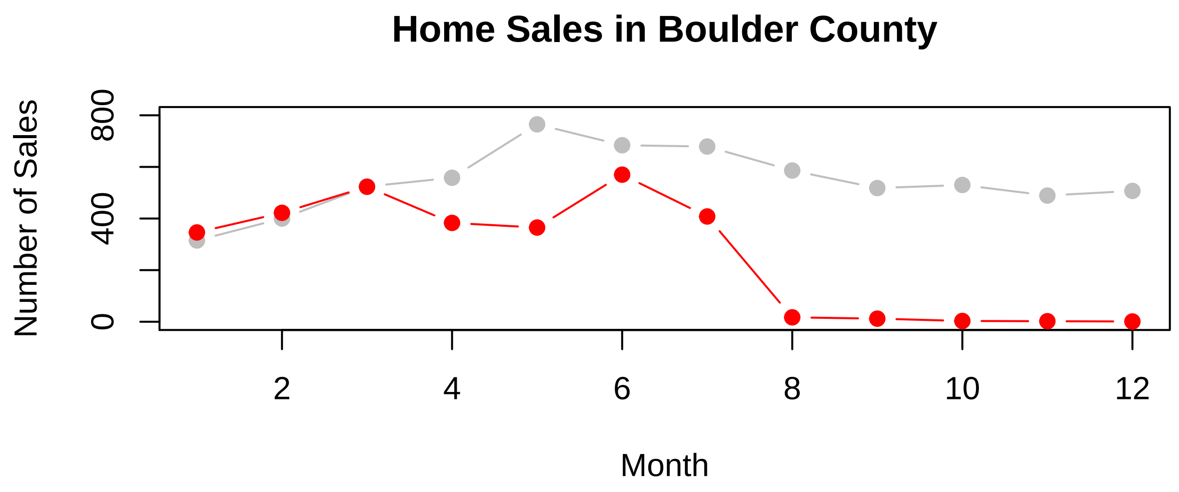


The `lubridate` package in R has many functions that help with making dates and times easier to work with, like the `mdy()` function that you've already seen. Here's a link to a cheatsheet with descriptions of many of the commonly used functions: https://rawgit.com/rstudio/cheatsheets/master/lubridate.pdf. You can use it to do these commonly needed actions:

- **Parse:** read dates and/or times into R
- **Get Components:** get just the month, day, or day of week of a date. Or get the hour of a time stamp
- **Time Zones:** convert from one time zone to another
- **Compute:** find the length of time between two dates, etc.

Example, Boulder County Housing Data: Create a plot with `month` on the x-axis and the number of sales on the y-axis in 2019. Then, overlay the same type of line for the 2020 home sales. Compare the two.

```r
monthly_sales_count_2019 <- mdy(sales2019$SALE_DATE)
# Warning: 1 failed to parse.

#What happened with the "1 that failed to parse"?
summary(monthly_sales_count_2019)

#There is one NA.  Let's find it.
which(is.na(monthly_sales_count_2019) == TRUE)
sales2019$SALE_DATE[5840]

#It has a time associated with it as well. So, let's
#overwrite that value with only a date and remove the time.
sales2019$SALE_DATE[5840] <- "11/15/19"

#And then reapply the `mdy` command to the sales dates.
monthly_sales_count_2019 <- mdy(sales2019$SALE_DATE)

#Success!

#Now, count the number of sales in each month by
#extracting the month from the date variable.
month_2019 <- month(monthly_sales_count_2019)

sold_counts_2019 <- as.numeric(table(month_2019))
plot(1:12, sold_counts_2019,
     type = "b",
     pch = 19,
     ylim = c(0,800),
     xlab = "Month",
     ylab = "Number of Sales",
     main = "Home Sales in Boulder County",
     col="grey")

#Now, we want to do the same and add the 2020 sales.
monthly_sales_count_2020 <- mdy(sales2020$SALE_DATE)
month_2020 <- month(monthly_sales_count_2020)
sold_counts_2020 <- as.numeric(table(month_2020))

#We make this one a different color.
lines(1:12, sold_counts_2020,
      type = "b",
      pch = 19,
      col = "red")
```



Adding Legends and fine tuning the figure.

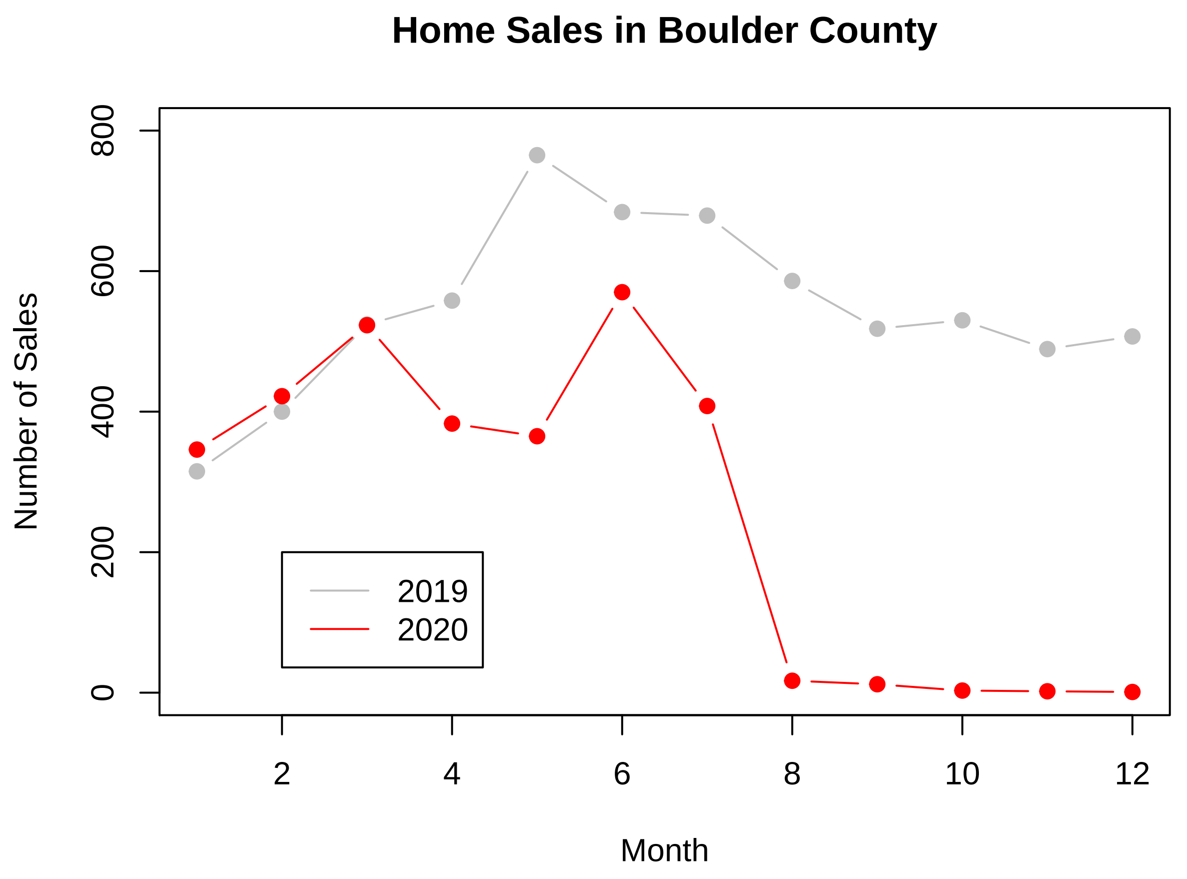
The basic syntax for adding legends, using the legend( ) function, to a plot is as follows:

```
legend(x, y=NULL, legend, fill, col, bg)
```

where the parameters within the call are: * **x and y**: position of the legend using the x and y coordinates * **legend**: the text of the legend to be displayed * **fill**: colors to use for filling the boxes beside the legend text * **col**: refers to the colors of lines and points beside the legend text * **bg**: background color of the legend box

Lets add a basic legend to our previous example to better explain to the viewer what they are reading. Now we redraw this *same* plot with a legend and use this as an example of the **matplot** function to add more than one line. Also we have given this a figure height of "5" (5 inches) to help Rmarkdown format this with a better aspect ratio of height to width. To get the legend placement just right I played around with the **2, 200** coordinates to locate the upper right corner and also considered adding **cex=.75** to make the text smaller. Placing the legend usually involves a little bit of trial and error so do it at the very end when you have the rest of the figure the way you like it.

```
matplot(1:12, cbind( sold_counts_2019,sold_counts_2020),
     type = "b",
     pch = 19,
     ylim = c(0,800),
     xlab = "Month",
     ylab = "Number of Sales",
     main = "Home Sales in Boulder County",
     lty=1,
     col=c("grey", "red")
        )

legend(2, 200,
       legend=c("2019", "2020"),
       lty = c(1, 1),
       col = c("grey", "red")
       )
```


Home Sales in Boulder County
Number of Sales
Month