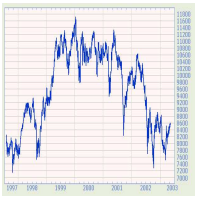

Playing with the Stock Market

Stock markets are a matter of international attention. Massive amounts of money are lost and gained each day, as stock traders decide the value of a stock, and players of the stock market make gambles about these values. Our goal here isn't to get rich quick by making bets on the market, rather we have academic desires—to learn some R commands using data from the stock market.

1 Starting in with R

The R software can perform in a convenient way most of the calculations in statistics. Think of R as a calculator for statistics where the many dedicated buttons are replaced by a keyboard where you type the commands for what you want to do.

In Windows you start R by clicking on the icon for it. This varies with other types of installations. Starting R in Windows opens up a large window that will contain various subwindows: a command console for typing commands, windows for displaying graphs, data-editing windows, and help page windows.

Interacting with R is done in a **question-and-answer** manner: you ask questions and R answers. You ask these questions by **typing** them in after the prompt:

```
>
```

For example, to see that R can be a calculator, type the following commands (not the prompt) and hit the `Enter` key:

```
> 2 + 2

[1] 4

> 5 * 6

[1] 30

> (3 + 2)^2

[1] 25
```

After a leading `[1]`, R returns the correct answer. (The leading `[1]` will be explained later.) As you see, R uses `+`, `-`, `*`, `/`, and `^` for the usual math notations; and parentheses to group expressions.

Question 1: Use R to find values for each:

$$52 \cdot 17.75, \quad 365/4, \quad 12 \cdot 5^2, \quad 125 \cdot (10.61 - 7.27).$$

2 Working with data

Statistics is about analyzing data sets which likely will have more than one data point. Unlike most calculators, R works naturally with data sets.

The price of a share of stock fluctuates on a daily basis. Some stocks more so than most. In January of 2004, The AT&T wireless stock (symbol AWE) for AT&T's cellphone services had been having a big decline. In late January though, word of a possible merger was released changing how investor's viewed the stock. (AT&T merged with Cingular in 2004.)

Data for the closing price of AT&T wireless stock for a few different Fridays are in Table 1. What can we say about this data?

23-Jan-04	10.61	12-Dec-03	7.13
16-Jan-04	9.99	5-Dec-03	7.27
9-Jan-04	8.15	28-Nov-03	7.50
2-Jan-04	8.08	21-Nov-03	7.00
26-Dec-03	7.63	14-Nov-03	6.81
19-Dec-03	7.35	7-Nov-03	7.02

Table 1: Closing price of AT&T wireless stock

2.1 Storing data

Before doing anything, let's store the data into the computer for January and December.

We use the function `c()` to combine numbers into a data set. Simply separate the values with commas.

```
> c(10.61, 9.99, 8.15, 8.08, 7.63, 7.35, 7.13, 7.27)

[1] 10.61  9.99  8.15  8.08  7.63  7.35  7.13  7.27
```

The numbers were combined and then printed – then they were forgotten! Again, the `[1]` appears. This helps keep track of how many numbers are in the **data vector** (we call a variable that stores data a data vector). When there are several rows of numbers output, the number in square brackets indicates the position of the first number in that row.

Functions in R are **called** using the function name, an opening parentheses, any arguments, and then a closing parentheses. Don't forget the parentheses. The **output** of a function is the name for what is returned.

We need to store the data so we can reuse it. To do this, we **assign** the output to a **variable** using an equals sign. The following will store the values into the variable called `awe`.

```
> awe = c(10.61, 9.99, 8.15, 8.08, 7.63, 7.35, 7.13, 7.27)
```

R is quiet after an assignment; only the prompt is returned. However, R was busy. Wherever the variable `awe` is used, R will refer to this dataset. For example, to see the values of a variable simply type its name:

```
> awe

[1] 10.61  9.99  8.15  8.08  7.63  7.35  7.13  7.27
```

Question 2: Type in the following data sets.

1. The numbers 7.97, 7.51, 5.94, 5.62 into the variable `pcs`
2. The numbers 24.40, 24.34, 24.16, 23.98, 24.76, 24.45 into the variable `sbux`

3 Manipulating data using functions

In `R` data sets are explored, summarized, and analyzed by applying functions to the data sets. A basic usage looks like

```
functionname( datasetname )
```

Though, many functions will have extra arguments to change their default behavior.

Many things can be done with the output of a function. It may simply answer your question. Or you may want to store it for later usage, or you may compose it directly with another function.

For the stock market, where there is so much data available, people are interested in summaries of the data. For example, maximum price, minimum price, and average price. `R` has functions `max()` and `min()` to find the maximum and minimum values in a data vector.

```
> max(awe)

[1] 10.61

> min(awe)

[1] 7.13
```

These are returned together with the `range()` function.

```
> range(awe)

[1]  7.13 10.61
```

Question 3: Use `R` to find the maximum and minimum of the variables `sbux` and `pcs`.

Question 4: The difference between the maximum and minimum values in a data set is sometimes referred to as the range of the data sets. There are several ways to find this. We can subtract the minimum from the maximum, or use the `diff()` function on the output of `range()`. For example,

```
> max(awe) - min(awe)

[1] 3.48
```

```
> diff(range(awe))

[1] 3.48
```

Find the difference between the maximum and minimum values of the variables `sbux` and `pcs`.

The average value of a data set can be found several ways, as illustrated next. For the data in `awe` we can do it all by hand:

```
> (10.61 + 9.99 + 8.15 + 8.08 + 7.63 + 7.35 + 7.13 + 7.27)/8

[1] 8.276
```

But, why should we type the data values in when they are already stored into `awe`. We can let the computer do the addition using the `sum()` function:

```
> sum(awe)/8

[1] 8.276
```

As well, rather than counting the eight numbers we added, we can let the computer find the length using `length(awe)`):

```
> sum(awe)/length(awe)

[1] 8.276
```

This works fine, but as find the average is a common task in statistics there is a built-in function, `mean()`, for this (the sample mean is the name of the average of a data set in statistics)

```
> mean(awe)

[1] 8.276
```

Question 5: Find the average of the data sets `sbux` and `pcs`

3.1 graphical views

`R` has several functions that produce graphics for viewing a data set. For example, we can make a simple plot of the data in a time-ordered manner using `plot()` (Figure 1):

```
> plot(awe)
```

After typing this command, a plot window should open up showing an admittedly boring plot. By default, this plots the numbers in the order they are typed in. The x-axis label, `Index`, refers to the position in the data vector of the data point.

Seems like the stock price is dropping doesn't it? Well not really, that's because the stock numbers were typed in reverse chronological order. How can we reverse the numbers without retyping the data? `R` has a built in function `rev()` to do so:

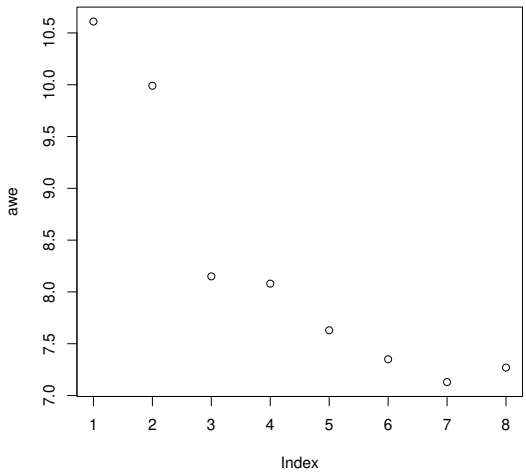

Figure 1: Plot of `awe` data

```
> rev(awe)

[1]  7.27  7.13  7.35  7.63  8.08  8.15  9.99 10.61
```

Okay, it works, we could store it for later use, but instead we plot (Figure 2) the output directly, as follows:

```
> plot(rev(awe))
```

The plot now shows an increasing trend. We all should have bought some shares.

Question 6: Make a plot of the `sbux` data set. Describe any trends.

Question 7: Another plotting function is a `barplot()`. Make a barplot of `awe`. Explain what the `barplot()` function does for this data.

Question 8: The variable `pcs`, from above, is stock data for the Sprint PCS company. It too is reversed in time. First use `rev()` and then plot the data.

4 Real data sets

All of the previous computer work could have been done by hand or with a calculator. To illustrate why a computer is a much better tool for statistics than a calculator, let's use bigger datasets. So big, you wouldn't even want to find the largest number by hand, let alone the average value. Rather than type the data in, we are going to let the computer do the work for us. However, you need to teach the computer how by typing the following exactly as shown (there are four capital letters):

```
> source("http://www.math.csi.cuny.edu/st/R/downloadStockData.R")
```

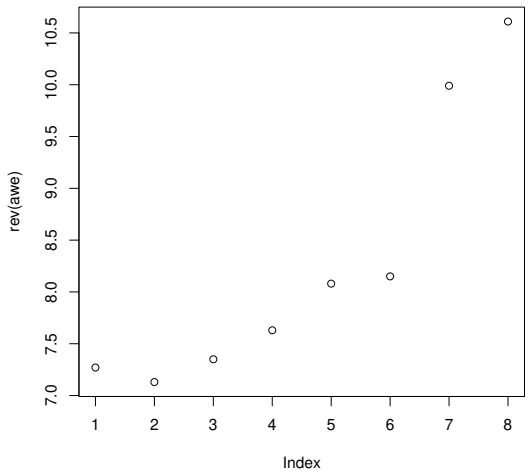

Figure 2: Plot of `awe` data after reversing order

This command downloads a file from the Stem and Tendril website. The file defines a new function, `downloadStockData()`, that will fetch the previous years worth of data on a stock courtesy of `http://finance.yahoo.com`. It only requires the user to provide the stock symbol. To illustrate, a years worth of stock data for for Yahoo! for can be retrieved by using its symbol, "YHOO."

```
> yahoo = downloadStockData("YHOO")
> max(yahoo)

[1] 57.59
```

This shows the maximum closing value of the stock for the previous year at the time this project was made (January 26, 2005).

A plot (Figure 3) of the year's activities is produced as before:

```
> plot(yahoo)
```

From this graph we can see a lot about the history of the stock. For example, We can look at this graph and see that the minimum value occurred near 130 and the maximum value occurred near index 60.

Question 9: Download *current* stock data for Yahoo!. Answer the following:

1. What was the maximum price?
2. What was the minimum price?
3. What was the average price?
4. Make a time series plot of the stock data. Identify when the maximum and minimum took place.

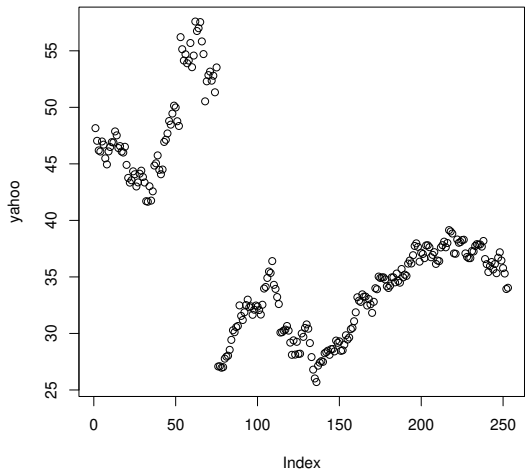

Figure 3: Yahoo! closing stock prices for previous year

Question 10: Download *current* stock data for Google (symbol "GOOG") into the variable `google`. Answer the following:

1. What was the maximum price?
2. What was the minimum price?
3. What was the average price?
4. Make a time series plot of the stock data. Identify when the maximum and minimum took place.

Question 11: The day-to-day differences in the stock price can be looked at by using the function `diff()`. This will form a new data vector containing the differences between successive days values. For example, the command

```
> yahoo.diffs = diff(yahoo)
```

forms the differences and stores them into the data vector `yahoo.diffs`.

For `yahoo.diffs` do the following:

1. What was the largest increase in a given day?
2. What was the largest decrease in a given day?
3. Make a plot of the data, can you identify periods when the stock was increasing most of the period? What about decreasing?

Question 12: Are the changes in price for the Google stock and the Yahoo! stock consistent? We can look at two plots at once, but a little bit of extra work is involved to make the y-axis large enough. The following commands will do so:

```
> ylim = c(0, max(yahoo, google))
> plot(yahoo, ylim = ylim)
> points(google)
```

(The `points()` function is similar to `plot()` except it does not make a new figure.)

Make the plot. Do the two graphs tend to track each other? Why might you expect them to?

Question 13: Do the above exercise comparing the stock price of Starbucks Coffee (symbol "SBUX") with Yahoo!. Do these two graphs, of data from different sectors, seem to track each other?

5 Using indices

The entries in a data vector come with a natural order: the first, second, ..., nth. Being able to access the values by their index can extend the ways we can look at a data vector.

To access a single value in a data set can be done using square brackets, `[]`. For example, if the closing value of the Dow Jones Industrial Average for a week was

10196 10243 10391 10433 10368

We can use indexing to subtract the week's first value from the last

```
> dow = c(10196, 10243, 10391, 10433, 10368)
> dow[5] - dow[1]

[1] 172
```

This says the market went up 172 points during this week.

This could also be done using `length()` to retrieve the last number:

```
> dow[length(dow)] - dow[1]

[1] 172
```

(Note that you use square brackets for data extraction, and parentheses for functions.)

More than one index can be referred to at once. To pull out the first and fifth days is done with:

```
> dow[c(1, 5)]

[1] 10196 10368
```

Question 14: Place a years worth of data for the Dow Jones Industrial average into the variable `dow.yr` as follows:

```
> dow.yr = downloadStockData("^DJI")
```

1. What was the overall change for the last year?
2. How many days of stock data are there for the last year?
3. What were the values of the 50th, 100th and 150th trading days?

5.1 Logical expressions

Indices can also be logical expressions allowing one to question the data.

We use this data for `dow`.

```
> dow

[1] 10196 10243 10391 10433 10368
```

We can ask what days were more than 10,200 as follows

```
> dow > 10200

[1] FALSE  TRUE  TRUE  TRUE  TRUE
```

The answer is `TRUE` or `FALSE` for each value in the data vector `dow`. When using such answers as indices, the values corresponding to `TRUE` are returned.

```
> dow[dow > 10200]

[1] 10243 10391 10433 10368
```

Logical expressions used for indices must be the same length as the data vector. Other logical questions are possible using `>`, `>=`, `<`, `<=`, `==` (double equals signs), and `!` for the negative. Expressions can be combined using `&` (and) and `|` (or).

For example, values less than or equal to 10,400 are

```
> dow[dow <= 10400]

[1] 10196 10243 10391 10368
```

Both conditions are found with

```
> dow[dow <= 10400 & dow > 10200]

[1] 10243 10391 10368
```

Question 15: For the variable `dow.yr` answer the following:

1. Find all the values more than the last closing value of the DOW.
2. What does

   ```
   > sum(dow.yr > dow.yr[length(dow.yr)] )
   ```

   return? Use this to find the proportion of days that the DOW was more than the last value.

Question 16: Compare `dow.yr` to its average value. What percent of the time was `dow.yr` more than its previous years average? Should this always be 50%?

5.2 What index was that?

A natural question to ask is what index has a value that does something special. For example, when is something at its maximum, or minimum? The `which()` command can answer in terms of the index.

```
> which(dow == max(dow))

[1] 4
```

The answer are the *indices* where the data set `dow` is at a maximum value. Similarly, the indices of when `dow` is at its minimum would be found with:

```
> which(dow == min(dow))

[1] 1
```

Question 17: For `dow.yr` find the indices of the times when the DOW was at it high for the year. If there is more than one, explain why `max()` can be used to find the largest index.

Question 18: How many trading days has it been since the DOW was at its last maximum?

Question 19: Make this plot:

```
> plot(cummax(dow.yr))
```

What do you think the function `cummax()` does?

5.3 Fancy indexing

There are a few conventions with indexing that can be used for great convenience.

For example, to find the percentage change for a stock per day would be done with a formula like

$$\text{percentage change} = \frac{\text{Today's value} - \text{Yesterday's value}}{\text{Yesterday's value}} \cdot 100\%$$

To find this for the second day for the variable `dow` might be done with this command:

```
> (dow[2] - dow[1])/dow[1] * 100

[1] 0.461
```

Okay, it works, but you wouldn't want to find 250 such values with this method. Instead we can use the appropriate functions.

The numerator is simply the difference of successive days. To find this, is done with `diff(dow)`. If we divide this by the proper value of `dow` we will be done. However, if `dow` has n entries, `diff(dow)` will have only $n-1$. Simply dividing by `dow` will not be right. However, dividing by all but the last one will be. This is referenced by using negative indices.

If negative indices in the range 1 to n are used, then **all but** those values are returned.

Using this indexing notation, to find the percentage difference for the daily fluctuations may be done with:

```
> diff(dow)/dow[-length(dow)] * 100

[1]  0.4610  1.4449  0.4042 -0.6230
```

Question 20: Find the percentage difference for the daily fluctuations for the variable `dow.yr`. Store the values in `diffs`.

- What was the maximum percentage difference?
- What was the minimum percentage difference?
- How many days was the change positive? What percentage is this?

5.4 Moving averages, medians

A moving average for stock market data, is an average taken over a window of time. For example, the stocks average for the last 50 days. This is relatively easy to find using fancy indexing. For example, to find the average of the last 50 days for the variable `dow.yr` is done with

```
> n = length(dow.yr)
> mean(dow.yr[(n - 49):n])

[1] 10596
```

The key is the notation a:b which finds a sequence of number from `a` to `b` by 1s. In this case, the 50 numbers from $n - 49$ to n.

Stock traders find useful a plot of this moving average for different indices. This can be done in `R`, but it is easier to find the moving median in `R` using the function `runmed()`.

The median of a data set is the middle value of the data once it is sorted. When there is an odd number of data points, the median can be found by repeatedly removing the smallest and largest values until only one remains. The `runmed()` function requires the data vector and a size of the window. For example, to find the running median with size 3 of the variable `dow` is done with

```
> runmed(dow, k = 3)

[1] 10196 10243 10391 10391 10391
attr(,"k")
[1] 3

> dow

[1] 10196 10243 10391 10433 10368
```

Different sizes for the window are specified to the argument `k=`.

For each index i, the median of the k values straddling the ith value are used. For example, as $k = 3$, for index 4 the median of values 3, 4, and 5 are found and returned. These values are `10391`, `10433`, and `10368`. The middle value, once sorted, is 10,391, so this is the fourth entry of the output of `runmed(dow, k=3)`.

Stock traders use the running median to smooth out the day-to-day fluctuations of a stock. To see how this happens for an 11-day window for the DOW data you can run these commands:

```
> plot(dow.yr, type = "l")
> lines(runmed(dow.yr, k = 11), col = "blue")
```

(The argument `type="l"` will plot a connected plot. The `lines()` function will add a connected plot with a different line color, specified with `col="blue"`.)

Question 21: Repeat the above graph with $k = 31$, and $k = 91$. Does the running median smooth out the data in a noticeable way?

Question 22: The data for a stock is an example of a time series. `R` can make nicer plots of time series, if asked. The key is to convert the dates into dates that `R` understands. This is done with the function `strptime()`, which needs to be told the date format.

For example, a time-series plot of the last year of Starbucks Coffee can be produced as follows (Figure 4).

```
> sbux = downloadStockData("SBUX")
> dates = strptime(names(sbux),"%d-%b-%y")
> plot(dates, sbux)
```

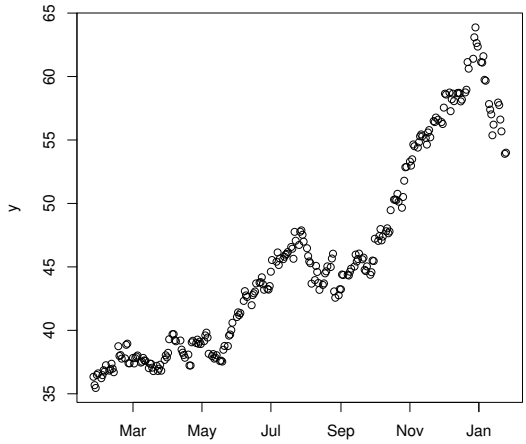

Figure 4: Time series plot of Starbucks Coffee stock

Produce this plot. What is different about this plot from one generated with `plot(sbux)`?

Make time-series plots of the stock values for Microsoft (MSFT) and Sun Microsystems (SUN). Do they show the same trend?