Basic Time Series Analysis with a Focus on Finance

Jiwon Kim
October 30, 2017

Introduction

This is an introduction to basic time series analysis, with a focus on Technical Analysis techniques. More specifically, this post will go over moving averages (exponential weighted MA, simple MA, weighted MAs), the RSI oscillator, and their uses in financial analysis. First and foremost, we will explore these indicators, how they are used and analyze how useful they are. To analyze these techniques, we will use S&P 500 stock prices from the past 5 years.

These methods (exponential MAs, RSI oscillator, etc.) are known as technical indicators, which falls under the large scope of technical analysis. In finance, two schools of thoughts exist: fundamental analysis and technical analysis. Fundamental analysis analyzes the state of the company by looking at their profit margins, the CEO, etc. However, technical analysis uses the market information to see in which direction the stock price might move in. Today, we will explore how some basic statistical time series analysis can help us understand the movement of stocks in the world of technical analysis and not fundamental.

Data Cleaning

We get our S&P 500 stock prices from a user on kaggle, who scraped the data from Google finance via the Python library pandas_datareader.

Here, we simply read in the stock information using the `readr` library.

```
library(readr)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggpubr)
```

```
## Loading required package: magrittr
```

```
sp500 = read_csv("sandp500/all_stocks_5yr.csv", col_types = list(
    Date = col_date(format = ""),
    Open = col_double(),
    High = col_double(),
    Low = col_double(),
    Close = col_double(),
    Volume = col_double(),
    Name = col_character()
))
```

To make analyzing the data easier, we have to convert the Date data into a numeric type, as to make graphing easier. We do this by converting to a POSIXct object and then converting to an integer.

```
# This converts the date to ISO Unix Time, which is a number
sp500$datetime = as.numeric(as.POSIXct(sp500$Date, format = "%m-%d-%Y"))
head(sp500, n = 5)
```

```
## # A tibble: 5 x 8
##         Date  Open  High   Low Close  Volume  Name   datetime
##       <date> <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <dbl>
## 1 2012-08-13 92.29 92.59 91.74 92.40 2075391   MMM 1344816000
## 2 2012-08-14 92.36 92.50 92.01 92.30 1843476   MMM 1344902400
## 3 2012-08-15 92.00 92.74 91.94 92.54 1983395   MMM 1344988800
## 4 2012-08-16 92.75 93.87 92.21 93.74 3395145   MMM 1345075200
## 5 2012-08-17 93.93 94.30 93.59 94.24 3069513   MMM 1345161600
```

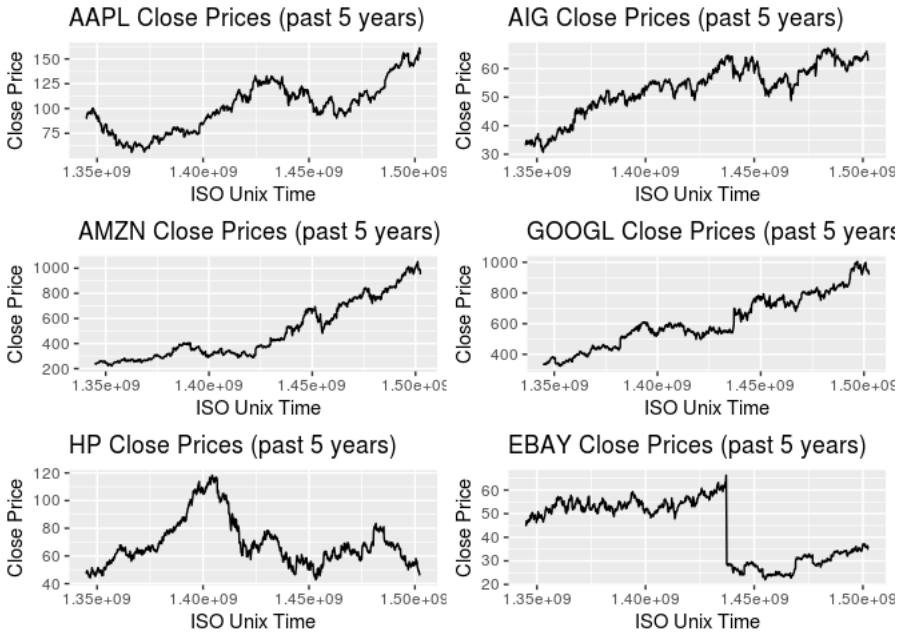
Introductory Visualizations

To start, we will start visualizing the time series data without any analysis using the library `ggplot2`. To make graphing easier, we create the following function that graphs the time series data given the name of the stock.

```
graph = function(name) {
  filtered = filter(sp500, Name == name)
  # plots the price points as a line
  fin = ggplot() + geom_line(aes(x = filtered$datetime, y = filtered$Close))
  # Adding labels
  fin = fin + labs(x = "ISO Unix Time", y = "Close Price", title = paste0(name, " Close Prices (past 5 year
};
```

Here, we graph some famous companies.

```
names = c("AAPL", "AIG", "AMZN", "GOOGL", "HP", "EBAY")
tempList = rep(list(ggplot()), 6)
for (x in 1:6) {
  tempList[[x]] = graph(names[x])
}
# Graphing the famous companies
ggarrange(plotlist = tempList, nrow = 3, ncol = 2)
```



Simple Moving Average

The simple moving average is calculated by adding the closing price for a certain number of time periods and then dividing the total by the number of time periods (source: Investopedia).

![Simple Moving Average Graph](photos/sma.png)

The mathematical description for the simple moving average is as below (Source: Wikipedia):

$$
\begin{align}
sma_{t} = \frac{x_{t} + x_{t - 1} + ... + x_{t - n + 1}}{n}
\end{align}
$$

To plot the simple moving average on top of the regular prices, we create the following function:

```
graphSMA = function(name, n = 5) {
  plt = graph(name)
  filtered = filter(sp500, Name == name)
  # this will be the x-values (i.e, the ISO Unix Time)
  datetimes = filtered$datetime
  # create a vector of the values of the simple moving
  # average, which we will plot with the datetimes
  sma = vector(mode = "numeric", length = length(datetimes))
  sum = 0
  for (ind in 1: length(datetimes)) {
    if (ind <= n) {
      # Before we have n data points, we cannot
      # calculate SMA, so we simply set it equal
      # to the price for the first n values

      # the 5th column is the closing price
      curr = as.numeric(filtered[ind, 5])
      sum = sum + curr
      sma[ind] = curr
    } else {
      # calculations for simple moving averages
      sma[ind] = sum * 1.0 / n
      sum = sum + as.numeric(filtered[ind, 5]) - sma[ind - n]
    }
  }
  # get rid of first n data points
  datetimes = datetimes[n + 1: length(datetimes)]
  sma = sma[n + 1: length(sma)]
  # we create a temporary dataframe to plot
  tempDF = data.frame(datetimes = datetimes, sma = sma)
  # now we create the graph using ggplot2
  fin = plt + geom_line(data = tempDF , aes(x = datetimes, y = sma), color = "red")
  fin = fin + labs(title = paste0(name, " MA (n = ", n, ")"))
  return(fin)
}
```

Now that we have a function that graphs the moving average with a particular `n` value, we can play around with values of n. For example, in the following, we've graphed AAPL stock with varying ns.

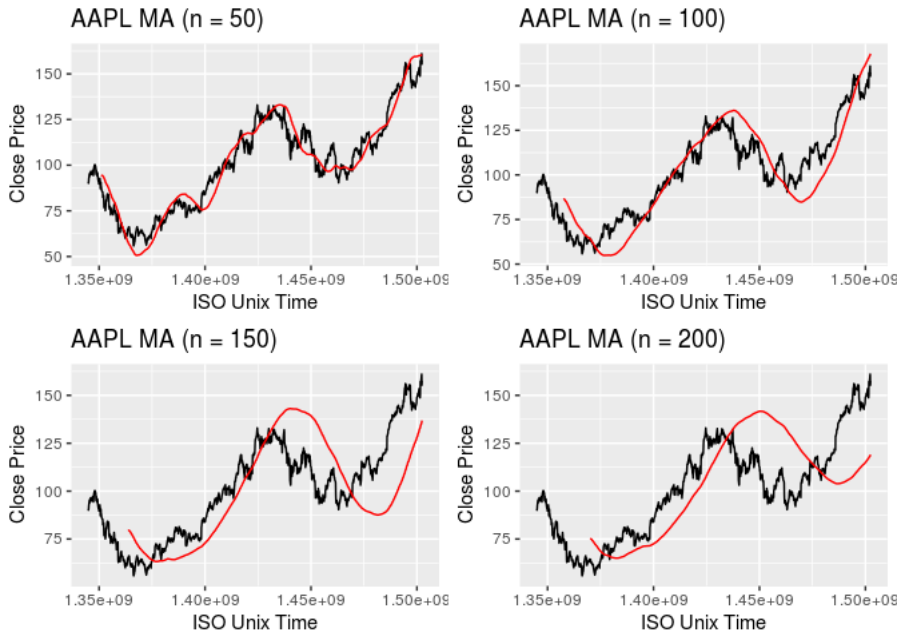
```
tempList = rep(list(ggplot()), 4)
count = 1
for (x in seq(from = 50, to = 200, by = 50)) {
  tempList[[count]] = graphSMA("AAPL", x)
  count = count + 1
}
# plotting for values n = 50 -> 200 in increments of 50
ggarrange(plotlist = tempList, nrow = 2, ncol = 2)
```

```
## Warning: Removed 50 rows containing missing values (geom_path).

## Warning: Removed 100 rows containing missing values (geom_path).

## Warning: Removed 150 rows containing missing values (geom_path).

## Warning: Removed 200 rows containing missing values (geom_path).
```



Notice how the larger n is, the larger the lag is between the moving average and the price itself. This is something that we will deal with in the exponential moving average section.

One important use of the moving average in technical analysis is the indication of trend. For example, consider the graph where $n = 200$. When the moving average line is below the price, the stock is said to be on an up-trend while it is considered to be in a down-trend if the MA line is above the price. Intuitively, this is because if the price goes up, that means the MA line will be below because it will be brought down by the early lower prices.

Exponential Moving Averages

One problem with simple moving averages was that the lag from the line to current prices could be too slow. One way to solve this problem is to use the exponential moving average. Instead of weighting each data point equally, EMAs weight the data points later more than the data points earlier (Source: Investopedia). Mathematically, the equation for EMA is as follows (Source: Wikipedia):

\begin{align*}
S_{t} =
\begin{cases}
p_{1} & \text{if $ x = 1$} \\
\alpha p_{t} + (1 - \alpha) S_{t - 1} & \text{if $ x > 1 $}
\end{cases}
\end{align*}
Looking at the previous image, we can see that the EMA reacts more quickly to stock prices:

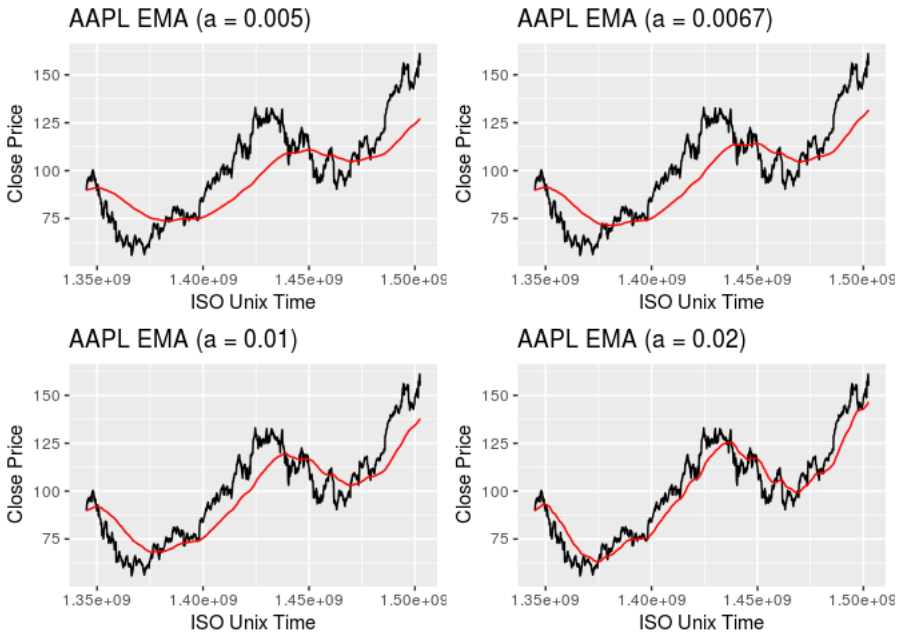
![Simple Moving Average Graph](photos/sma.png)
For convenience's sake, we define a function that graphs EMA.

```
graphEMA = function(name, alpha = 0.9) {
  plt = graph(name)
  filtered = filter(sp500, Name == name)
  datetimes = filtered$datetime

  # we calculate the y values and save them in a vector
  emas = vector(mode = "numeric", length = length(datetimes))
  ema = as.numeric(filtered[1, 5])
  # the first exponential moving average is equal to the price
  # point, so that's what we do here
  emas[1] = ema
  for (ind in 2: length(datetimes)) {
    # calculation
    ema = as.numeric(filtered[ind, 5]) * alpha + (1 - alpha) * ema
    emas[ind] = ema
  }
  # we create temporary dataframe to plot
  tempDF = data.frame(datetimes = datetimes, emas = emas)
  # we now create the final plot
  fin = plt + geom_line(data = tempDF, aes(x = datetimes, y = emas), color = "red")
  fin = fin + labs(title = paste0(name, " EMA (a = ", alpha, ")"))
  return(fin)
}
```

Here, we graph the Exponential Moving Average curves for the AAPL stock in the past 5 years.

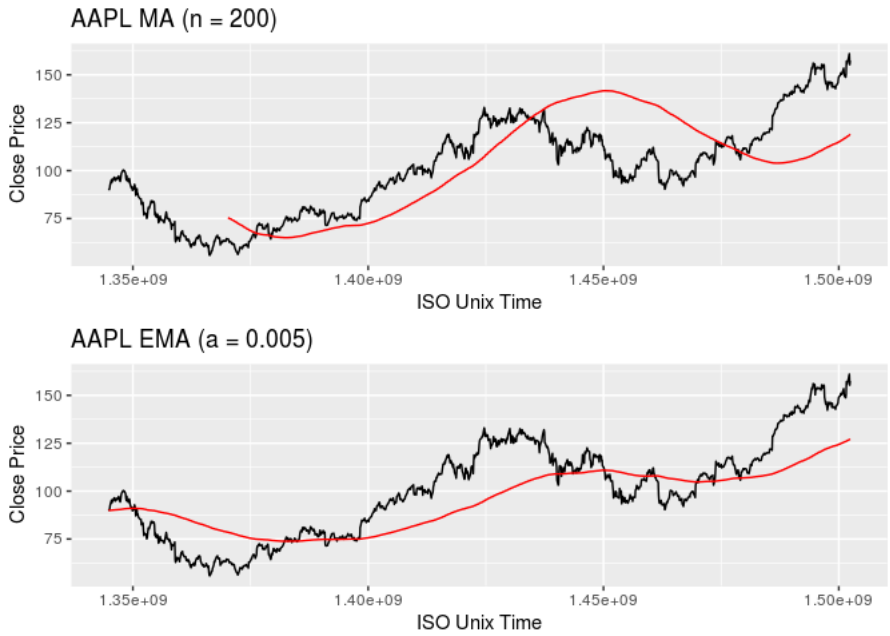
```
tempList = rep(list(ggplot()), 4)
count = 1
for (x in seq(from = 200, to = 50, by = -50)) {
  tempList[[count]] = graphEMA("AAPL", round(1.0 / x, digits = 4))
  count = count + 1
}
# plot for differing values of alpha
ggarrange(plotlist = tempList, nrow = 2, ncol = 2)
```



To see how much quicker EMA is at responding to price changes than SMA is, consider the following two graphs:

```
ggarrange(graphSMA("AAPL", 200), graphEMA("AAPL", 1.0 / 200), nrow = 2, ncol = 1)
```

```
## Warning: Removed 200 rows containing missing values (geom_path).
```

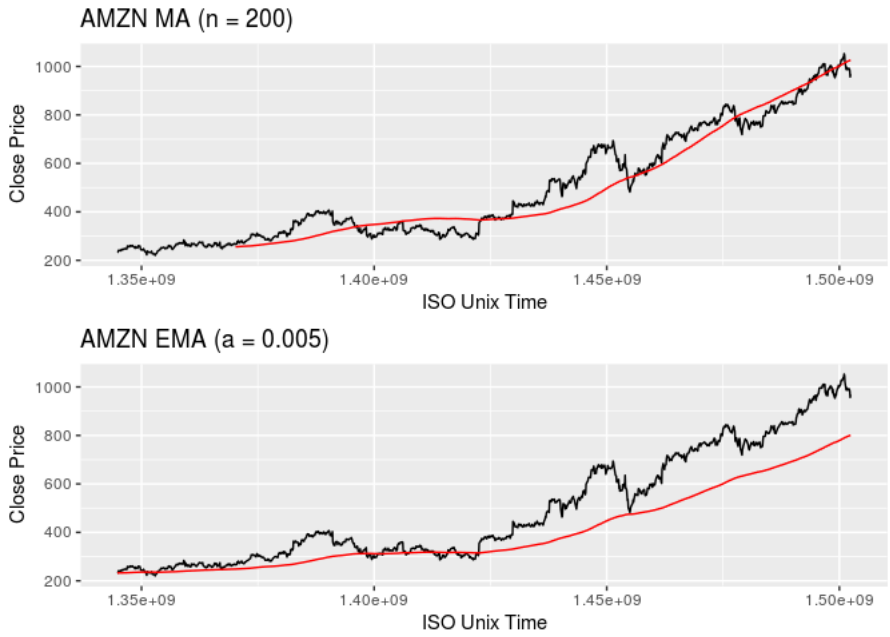


Using a α = 0.005 value means that the last data point is given a weight of $\frac{1}{200}$, which is the equivalent of all the data points in a simple moving average. Thus, these two graphs are comparable. Note how more reactive the EMA is to the price change around the 1.45e09 point. The SMA continues to increase to the 140s, but the EMA stays low at around price 115 because it gives more weight to the latest data points.

We can also note that the EMA is better at trend analysis than a SMA. For example consider the following two graphs

```
ggarrange(graphSMA("AMZN", 200), graphEMA("AMZN", 1.0 / 200), nrow = 2, ncol = 1)
```

```
## Warning: Removed 200 rows containing missing values (geom_path).
```



As we noted before, if the MA is below the prices, then the stock price is in an uptrend. Using a SMA provides a lot of false positives as there are quite a number of places where the price goes below the MA (despite being in an uptrend); however in the EMA, the EMA is almost always below the price level, which more accurately represents the up-trend in the data.

RSI Oscilator

Developed by J. W Elles Wilder, the Relative Strength Index (RSI) is a technical indicator that measures the speed and change of price movements (Source: Stockcharts). The mathematical description of the RSI is as follows:
\begin{align}
RSI &= 100 - \frac{100}{1 + RS}

RS &= \frac{\text{Average Gain} }{\text{Average Loss}}
\end{align}
As seen in the above description, the number of periods used is usually 14. Furthermore, an up period is one time frame where the close is higher than the open. Average Gain or (AG) simply adds the differences between the close and the open for each time period where the close is higher than the open and divides by 14. Average Loss or (AL) is the other way (i.e, the close is lower than the open).

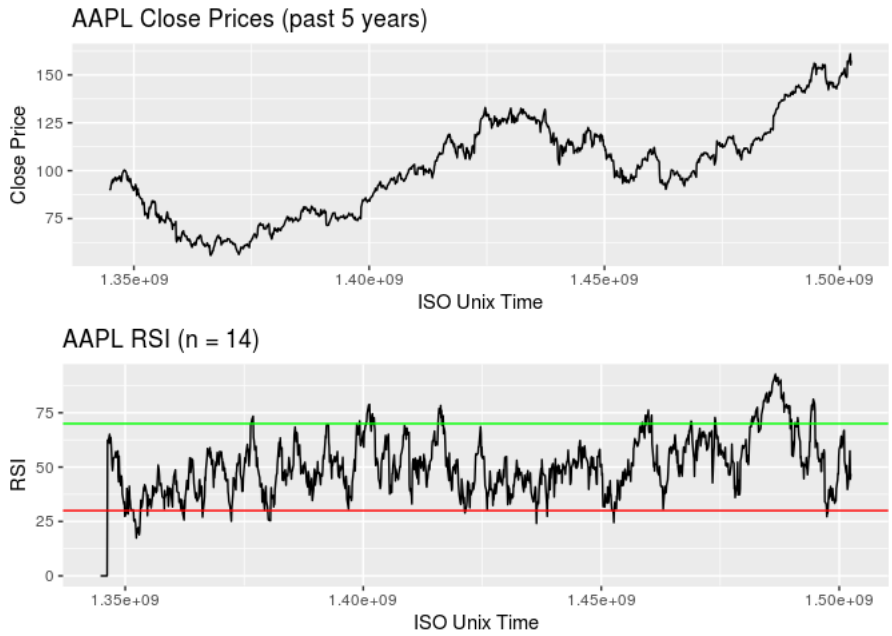
This mathematical description is only true for the first RSI value. Every RSI value after uses exponential smoothing (i.e, similar to EMAs from the previous section) (Source: Wikipedia). For example,
\begin{align*}
AG_{t} &= \frac{AG_{t- 1} 13 + g}{14} \
AL_{t} &= \frac{AL_{t- 1} * 13 + l}{14}
\end{align*}
Where A**Gt *is the average gain at time t,* AL**t** *is the average loss at time t,* g* is the current gain, and *l* is the current loss.

Now, we write a function that graphs the RSI.

```
graphRSI = function(name, n = 14) {
  filtered = filter(sp500, Name == name)
  datetimes = filtered$datetime
  # To save the y-values, we save the values in a vector
  rsi = vector(mode = "numeric", length = length(datetimes))
  ag = 0
  al = 0
  for (ind in 1: length(datetimes)) {
    diff = as.numeric(filtered[ind, 5]) - as.numeric(filtered[ind, 2])
    # calculating the first 14
    if (ind <= 14) {
      # gain or loss
      if (diff > 0) {
        ag = ag + diff
      } else {
        al = al + diff * (-1)
      }
      # save value
      if (ind == 14) {
        ag = ag / 14.0
        al = al / 14.0
        rsi[ind] = 100 - (100) / (1 + ag * 1.0/al)
      } else {
        rsi[ind] = 0
      }
    } else {
      # Exponential smoothing
      if (diff > 0) {
        ag = (ag * 13 + diff) / 14.0
        al = (al * 13) / 14.0
      } else {
        al = (al * 13 + diff * (-1) ) / 14.0
        ag = (ag * 13) / 14.0
      }
      # Saving the RSI value
      rsi[ind] = 100 - 100 / (1 + (ag * 1.0 / al))
    }
  }
  # rsig = RSI Graph
  rsig = ggplot(data = data.frame(x = datetimes, y = rsi))
  # Data
  rsig = rsig + geom_line(aes(x = x, y = y))
  # Labels
  currtitle = paste0(name, " RSI (n = ", n, ")")
  rsig = rsig + labs(title = currtitle, x = "ISO Unix Time", y = "RSI")
  # Horizontal Lines
  rsig = rsig + geom_hline(yintercept = 30, color = "red")
  rsig = rsig + geom_hline(yintercept = 70, color = "green")
  return(rsig)
}
```

The RSI indicator ranges from 0 to 100 and if the RSI goes above 70, it is said to be over bought, meaning the RSI indicator is going to go down (and most likely the price with it). If the RSI goes below 30, it is said to be over sold, meaning the RSI indicator is likely to go up (and most likely the price with it). Thus, a common strategy is to buy when the RSI goes below 30 and sell when the RSI goes above 70. **However**, this is a very dangerous strategy. For example, see below:

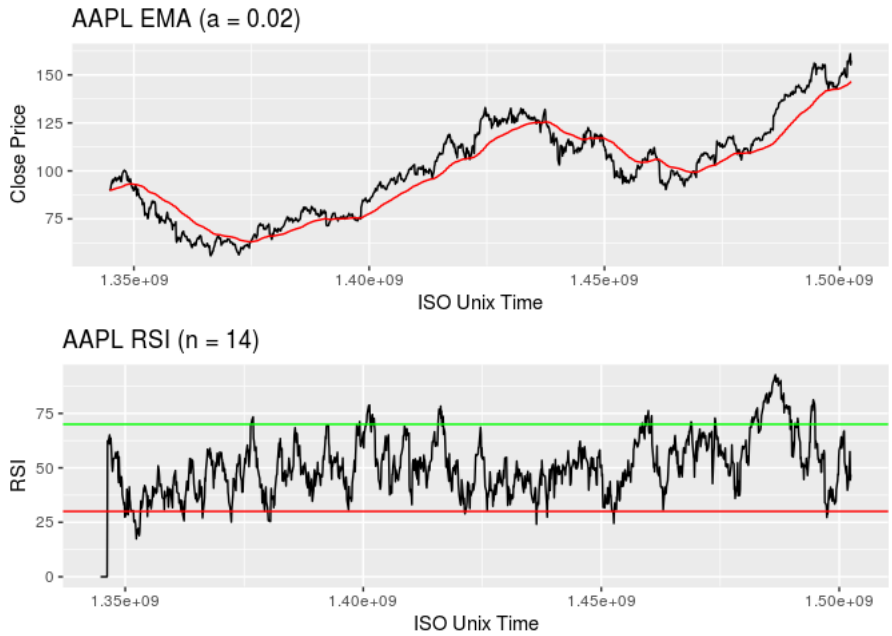
```
ggarrange(graph("AAPL"), graphRSI("AAPL"), nrow = 2, ncol = 1)
```



The problem with that buying/selling strategy is that if the stock price is trending (i.e, going up or going down), betting against the market makes it much more likely that you will lose your bet. For example, the AAPL stock has been trending upwards in the past 5 years, so selling when the RSI indicator is overbought (i.e, above 70), expecting the price to go down, rarely works.

More specifically, if you look at 1.45e+09 ISO Unix Time, AAPL is oversold; however, the stock price continues to increase. Thus, it is good to use the RSI indicator with a trend indicator. In our case, we will use the EMA to see what trend the stock is currently in.

```
ggarrange(graphEMA("AAPL", 0.02), graphRSI("AAPL"), nrow = 2, ncol = 1)
```



Thus, we use the EMA to figure out whether or not it is an uptrend or a downtrend, and we use the RSI graph to figure out when we enter the market. One problem with the RSI is that it does not produce that many signals, which means there are less false positives, but there can be many instances where it is never used. One way to solve this problem is to use the stochastic RSI, which we will not cover here.

Conclusion

Technical Analysis is filled with many technical indicators that have uses in some scenarios, but are not useful in others.

Today, we saw how moving averages can be used to identify trends in a stock price and we compared and contrasted exponential moving averages to simple moving averages. Furthermore, we explored the RSI oscillator, which we combined with the exponential moving averages to create a buy/sell strategy. Thus, in conclusion, we learned how to apply some statistical methods to analyze trends and price momentum in stock prices.

References

1. S&P500 Data - Kaggle
2. Simple Moving Average - Investopedia
3. Simple Moving Average Formula - Wikipedia
4. Exponential Moving Average Description - Investopedia
5. Exponential Moving Average Formula - Wikipedia
6. RSI Oscillator Description - Stockcharts
7. RSI Oscillator Formula - Wikipedia