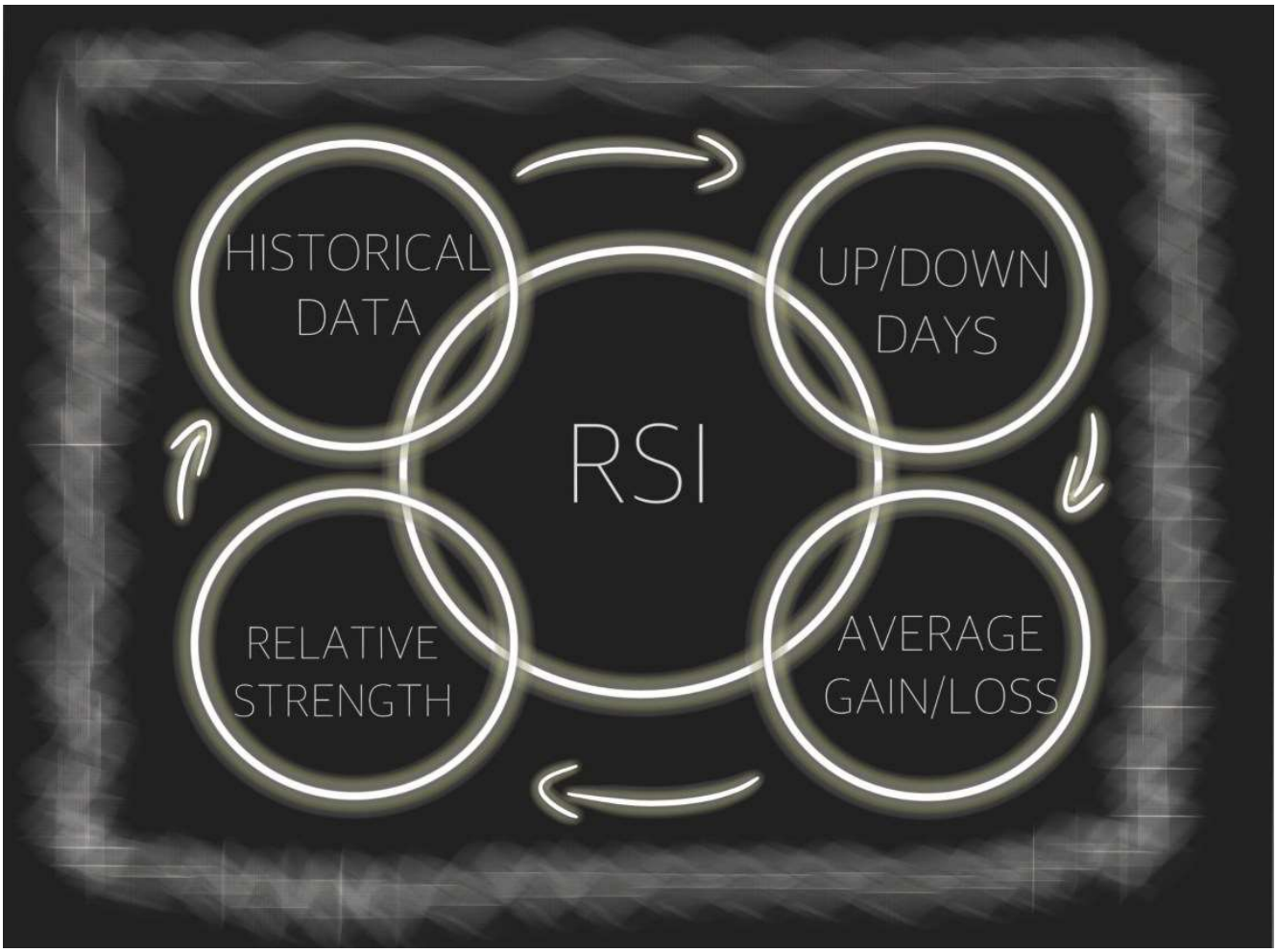
TECHNICAL ANALYSIS

Calculate and Analyze RSI Using Python

by Cameron | Published August 23, 2020

Automate the calculation of RSI for a list of stocks, and then analyze its accuracy at predicting future price movements.



An outline of the process to calculate RSI and its historical accuracy for a stock. (Image by Author)

The relative strength index is a momentum oscillator commonly used to predict when a company is oversold or overbought. The calculation process is straightforward:

1. Observe the last 14 closing prices of a stock.
2. Determine whether the current day's closing price is higher or lower than the previous day.
3. Calculate the average gain and loss over the last 14 days.
4. Compute the relative strength (RS): (AvgGain/AvgLoss)
5. Compute the relative strength index (RSI): (100–100 / (1 + RS))

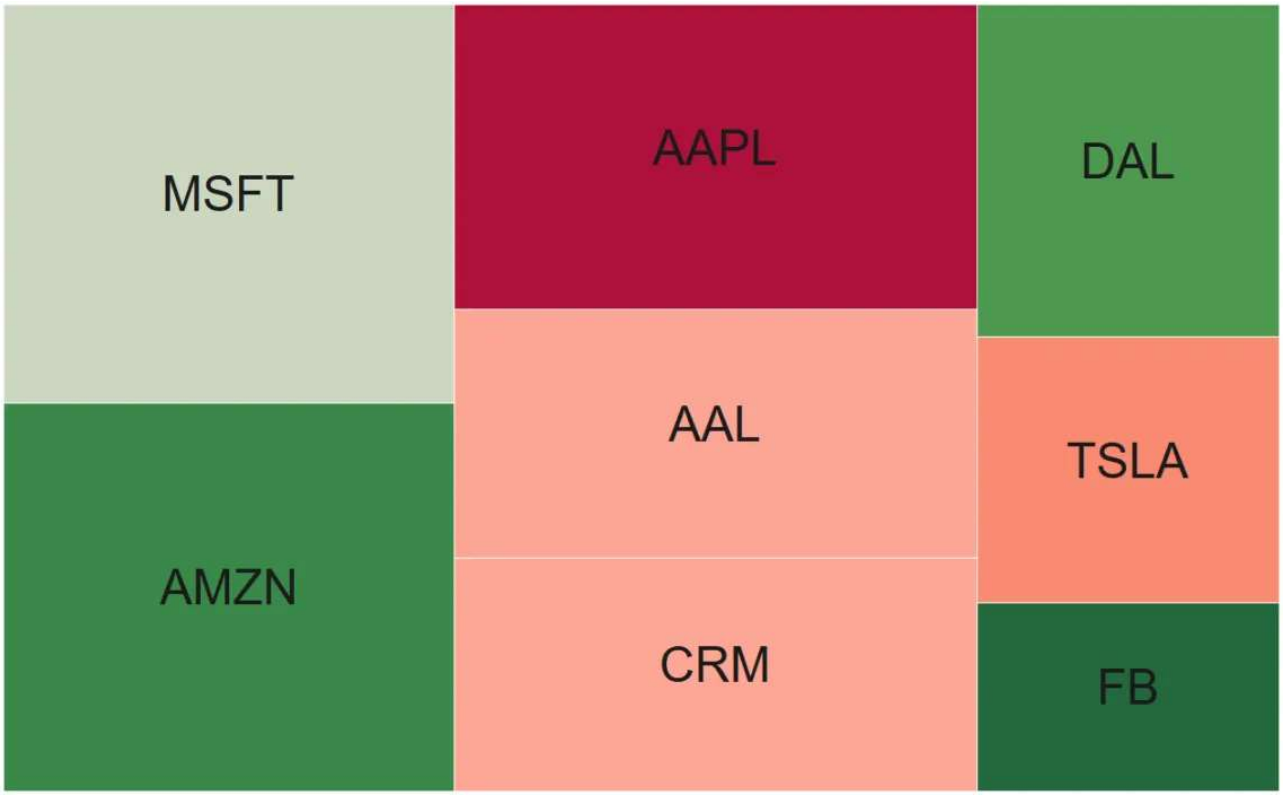
The RSI will then be a value between 0 and 100. It is widely accepted that when the RSI is 30 or below, the stock is undervalued and when it is 70 or above, the stock is overvalued.

The code in this walk-through will calculate the RSI for each stock in a user-defined list of tickers. It will then highlight every crossover in each stock's historical data and using this information, determine the accuracy of the RSI at predicting future price movements.

I will also touch on the analysis of this technical indicator. Using the algorithm below, we can extract a lot of information about each stock and their likelihood of being accurately predicted by the RSI. For example, we can quantify and visualize which stocks were most successfully predicted using the relative strength index.

Company	Days_Observed	Crosses	True_Positive	False_Positive	True_Negative	False_Negative	Sensitivity	Specificity	Accuracy	TPR	FPR
AAL	3,716	1,084	245	288	273	278	47%	49%	48%	47%	51%
AAPL	4,297	1,330	167	638	407	118	59%	39%	43%	59%	61%
AMZN	5,797	1,463	256	520	509	178	59%	49%	52%	59%	51%
CRM	4,047	1,019	166	404	321	128	56%	44%	48%	56%	56%
DAL	3,327	841	186	240	248	167	53%	51%	52%	53%	49%
FB	2,055	477	83	171	173	50	62%	50%	54%	62%	50%
MSFT	6,572	1,499	245	562	481	211	54%	46%	48%	54%	54%
TSLA	2,532	671	116	272	199	84	58%	42%	47%	58%	58%

The final output of the analysis section in the code below.



A visualization of the observed stocks where color signifies RSI's predictive accuracy and size is the volume of momentum crossovers. (Image by Author)

In the first picture above, you can get an idea of what the final output of this code's analysis section will look like. The variables we will be using to judge the RSI's predictive power can be used to create an ROC curve for further visualization.

In the second picture, we can see the list of observed stocks graphed by their respective accuracy and volume of crosses. I will dive further into the insights derived from this code later.

In the next section of this article, I will give a breakdown of the code I created to automate RSI calculation for a list of stocks. The

- Import python libraries.
- Initialize necessary variables.
- Import historical stock data from yahoo finance.
- Calculate the RSI for each stock's historical data.
- Analyze and compare the RSI's predictive power for each stock.

Prepare the Algorithm

In order to automate the calculation of RSI on the historical data of a list of stocks, we must first complete a couple of preparation steps.

Import Necessary Libraries

The first step to obtaining the relative strength index (RSI) is to bring in the libraries needed to compute it. Below are the required libraries:

Libraries:

- Yfinance: Gather the historical data of each stock.
- Pandas: Work with large datasets.
- Shutil, Glob, and OS: Access folders/files on your computer.
- Time: Forcing the program to pause for a period of time.
- Get_All_Tickers: Filter through all stocks to get the list you desire.
- Numpy: Work with arrays.
- Requests: Avoid connection errors.
- Statistics: Calculate exponentially weighted means.

https://gist.github.com/cameronShadmehry/36cf5bbc7809777e2dd618cc1714cc77#file-rsi_lib-py

```
# Necessary Libraries
import yfinance as yf, pandas as pd, shutil, os, time, glob
import numpy as np
import requests
from get_all_tickers import get_tickers as gt
from statistics import mean
```

Initialize Variables

Next we need to set some variables that will be important in the next section. You can add as many tickers as you would like to this list. In the past, I have used the get_all_tickers library to automate the collection of tickers by filtering through a large data-set. Unfortunately, this library has a bug right now and isn't working, so for the time being the tickers list will have to be updated manually.

Other than setting the list of tickers we want to observe, we also need to remove and create two folders on our local machine. One folder will be used to hold historical stock data and the other will contain the stock data but with RSI values joined to the CSV file.

Note that before this step runs for the first time, you will need to manually create these two folders so that the program can remove the folders and recreate them. You will get an error if you run this code for the first time without first manually creating those folders. This step is important because it will get rid of the data from past iterations and allow the program to start new.

https://gist.github.com/cameronShadmehry/59308488cf13abe89ccc1d3774864d91#file-rsi_init-py

```
# If you have a list of your own you would like to use just create a new list instead of using this, for example: tickers = ["FB", "AMZN", ...]
tickers = ["AMZN", "CRM", "AAL", "DAL", "TSLA", "MSFT", "FB", "AAPL"]
# Check that the amount of tickers isn't more than 2000
print("The amount of stocks chosen to observe: " + str(len(tickers)))
# These two lines remove the Stocks folder and then recreate it in order to remove old stocks. Make sure you have created a Stocks Folder the first time you run this.
shutil.rmtree("<Your Path>")
os.mkdir("<Your Path>")
#  These will do the same thing but for the folder jolding the RSI values for each stock.
shutil.rmtree("<Your Path>")
os.mkdir("<Your Path>")
```

Import Historical Stock Data

We will now obtain the historical price data of each stock in our tickers list by making independent calls to Yahoo Finance. After receiving the data, the program will save each company's information in a new CSV file that will be located in the folder you created beforehand.

In the code below you will notice two things:

- There are exception handling sections to ensure that the program does not stop if it runs into a value error or request error.
- I have my historical data going back to the first day the stock was publicly traded. If you do not need to go this far back, you can change your observation time period by altering this statement:

> ”
>
> Hist_data = temp.history(period=”max”)

```
# Do not make more than 2,000 calls per hour or 48,000 calls per day or Yahoo Finance may block your IP. The clause "(Amount_of_API_Calls < 1800)" below will stop the loop from making
# too many calls to the yfinance API.
Stock_Failure = 0
Stocks_Not_Imported = 0
# Used to iterate through our list of tickers
i=0
while (i < len(tickers)) and (Amount_of_API_Calls < 1800):
    try:
        print("Iteration = " + str(i))
        stock = tickers[i]  # Gets the current stock ticker
        temp = yf.Ticker(str(stock))
        Hist_data = temp.history(period="max")  # Tells yfinance what kind of data we want about this stock (In this example, all of the historical data)
        Hist_data.to_csv("<Your Path>")  # Saves the historical data in csv format for further processing later
        time.sleep(2)  # Pauses the loop for two seconds so we don't cause issues with Yahoo Finance's backend operations
        Amount_of_API_Calls += 1
        Stock_Failure = 0
        i += 1  # Iteration to the next ticker
    except ValueError:
        print("Yahoo Finance Backend Error, Attempting to Fix")  # An error occured on Yahoo Finance's backend. We will attempt to retreive the data again
        if Stock_Failure > 5:  # Move on to the next ticker if the current ticker fails more than 5 times
            i+=1
            Stocks_Not_Imported += 1
        Amount_of_API_Calls += 1
        Stock_Failure += 1
    # Handle SSL error
    except requests.exceptions.SSLError as e:
        print("Yahoo Finance Backend Error, Attempting to Fix SSL")  # An error occured on Yahoo Finance's backend. We will attempt to retreive the data again
        if Stock_Failure > 5:  # Move on to the next ticker if the current ticker fails more than 5 times
            i+=1
            Stocks_Not_Imported += 1
        Amount_of_API_Calls += 1
        Stock_Failure += 1
print("The amount of stocks we successfully imported: " + str(i - Stocks_Not_Imported))
```

Calculate the Historical RSI for Each Stock

I outlined the process for calculating RSI in the introduction. Now, I will show you how to calculate RSI for every day in a stocks historical price data. The code below will also loop through each stock in the list of tickers, therefore completely automating the calculation of historical RSI for a list of stocks.

RSI Computation Loop

A high level overview of the code below:

1. Loop through each stock's historical price data.
2. Compute the price movement every day (up/down).
3. Gather the average gain and loss over the last 14 days.
4. Calculate the Relative Strength (RS) and Relative Strength Index (RSI).
5. Save the RSI and price data to a new CSV file for later use.

```
# Get the path for each stock file in a list
list_files = (glob.glob("<Your Path>"))
# You can use this line to limit the analysis to a portion of the stocks in the "stocks folder"
# list_files = list_files[:1]
# Create the dataframe that we will be adding the final analysis of each stock to
Compare_Stocks = pd.DataFrame(columns=["Company", "Days_Observed", "Crosses", "True_Positive", "False_Positive",
"True_Negative", "False_Negative", "Sensitivity",
"Specificity", "Accuracy", "TPR", "FPR"])
# While loop to cycle through the stock paths
for stock in list_files:
    # Dataframe to hold the historical data of the stock we are interested in.
    Hist_data = pd.read_csv(stock)
    Company = ((os.path.basename(stock)).split(".csv")[0])  # Name of the company
    # This list holds the closing prices of a stock
    prices = []
    c = 0
    # Add the closing prices to the prices list and make sure we start at greater than 2 dollars to reduce
outlier calculations.
    while c < len(Hist_data):
        if Hist_data.iloc[c,4] > float(2.00):  # Check that the closing price for this day is greater than $2.00
            prices.append(Hist_data.iloc[c,4])
        c += 1
    # prices_df = pd.DataFrame(prices)  # Make a dataframe from the prices list
    i = 0
    upPrices=[]
    downPrices=[]
    #  Loop to hold up and down price movements
    while i < len(prices):
        if i == 0:
            upPrices.append(0)
            downPrices.append(0)
        else:
            if (prices[i]-prices[i-1])>0:
                upPrices.append(prices[i]-prices[i-1])
                downPrices.append(0)
            else:
                downPrices.append(prices[i]-prices[i-1])
                upPrices.append(0)
        i += 1
    x = 0
    avg_gain = []
    avg_loss = []
    #  Loop to calculate the average gain and loss
    while x < len(upPrices):
        if x <15:
            avg_gain.append(0)
            avg_loss.append(0)
        else:
            sumGain = 0
            sumLoss = 0
            y = x-14
            while y<=x:
                sumGain += upPrices[y]
                sumLoss += downPrices[y]
                y += 1
            avg_gain.append(sumGain/14)
            avg_loss.append(abs(sumLoss/14))
        x += 1
    p = 0
    RS = []
    RSI = []
    #  Loop to calculate RSI and RS
    while p < len(prices):
        if p <15:
            RS.append(0)
            RSI.append(0)
        else:
            RSvalue = (avg_gain[p]/avg_loss[p])
```

```
        p+=1
    #  Creates the csv for each stock's RSI and price movements
    df_dict = {
        'Prices' : prices,
        'upPrices' : upPrices,
        'downPrices' : downPrices,
        'AvgGain' : avg_gain,
        'AvgLoss' : avg_loss,
        'RS' : RS,
        'RSI' : RSI
    }
    df = pd.DataFrame(df_dict, columns = ['Prices', 'upPrices', 'downPrices', 'AvgGain','AvgLoss', 'RS', "RSI"])
    df.to_csv("<Your Path>"+Company+"_RSI.csv", index = False)
```

RSI Algorithm Output

Here is an example output from the code above. The picture below is an excerpt from the RSI calculation on Amazon's historical price data.

Prices	upPrices	downPrices	AvgGain	AvgLoss	RS	RSI
1670.43	76.55	0	14.05571429	13.06642857	1.075712021	51.82376024
1718.73	48.3	0	17.30285714	13.06642857	1.32422238	56.97485712
1626.23	0	-92.5	17.30285714	19.445	0.889835801	47.08534997
1633.31	7.08	0	17.80857143	18.32642857	0.971742604	49.28344106
1658.81	25.5	0	19.63	16.65857143	1.178372352	54.09416581
1640.26	0	-18.55	15.53357143	17.98357143	0.863764547	46.34515387
1614.37	0	-25.89	14.875	19.83285714	0.750018008	42.85773086
1588.22	0	-26.15	14.20071429	21.70071429	0.654389256	39.5547332
1591	2.78	0	14.18642857	21.70071429	0.65373095	39.53067155

RSI calculation on Amazon's historical stock data.

Analyze and Compare the Predictive Power of RSI

Potentially the most important part of this code, is its ability to measure the RSI's accuracy at predicting an individual stock's future price movement. This is only something we can achieve after first calculating the RSI, which we have done in the code above.

Now, we will observe every RSI crossover the 30% and 70% mark to see if this technical indicator truly is a good signal of future price movement.

Compute the Accuracy of RSI at Predicting Future Price Movements

A high level overview of the code below:

1. Initialize the variables we are using to measure accuracy.

3. If a crossover did occur, check if the stock's future price moved as expected.

4. Compute the measurement variables.
5. Save the accuracy measurements for each stock to the same CSV file for easy comparison.

https://gist.github.com/cameronShadmehry/e1909d0ff720b51d12636738c88e7f17#file-rsi_get_accuracy-py

```
#  Code to test the accuracy of the RSI at predicting stock prices
    Days_Observed = 15
    Crosses = 0
    nothing = 0
    True_Positive = 0
    False_Positive = 0
    True_Negative = 0
    False_Negative = 0
    Sensitivity = 0
    Specificity = 0
    Accuracy = 0
    while Days_Observed < len(prices)-5:
        if RSI[Days_Observed] <= 30:
            if ((prices[Days_Observed + 1] + prices[Days_Observed + 2] + prices[Days_Observed + 3] + prices[Days_Observed + 4] + prices[Days_Observed + 5])/5) > prices[Days_Observed]:
                True_Positive += 1
            else:
                False_Negative += 1
            Crosses += 1
        elif RSI[Days_Observed] >= 70:
            if ((prices[Days_Observed + 1] + prices[Days_Observed + 2] + prices[Days_Observed + 3] + prices[Days_Observed + 4] + prices[Days_Observed + 5])/5) <= prices[Days_Observed]:
                True_Negative += 1
            else:
                False_Positive += 1
            Crosses += 1
        else:
            #Do nothing
            nothing+=1
        Days_Observed += 1
    # while Days_Observed<len(prices)-5:

    #     Days_Observed += 1
    try:
        Sensitivity = (True_Positive / (True_Positive + False_Negative)) # Calculate sensitivity
    except ZeroDivisionError:  # Catch the divide by zero error
        Sensitivity = 0
    try:
        Specificity = (True_Negative / (True_Negative + False_Positive)) # Calculate specificity
    except ZeroDivisionError:
        Specificity = 0
    try:
        Accuracy = (True_Positive + True_Negative) / (True_Negative + True_Positive + False_Positive + False_Negative) # Calculate accuracy
    except ZeroDivisionError:
        Accuracy = 0
    TPR = Sensitivity  # Calculate the true positive rate
    FPR = 1 - Specificity  # Calculate the false positive rate
    # Create a row to add to the compare_stocks
    add_row = {'Company' : Company, 'Days_Observed' : Days_Observed, 'Crosses' : Crosses, 'True_Positive' : True_Positive, 'False_Positive' : False_Positive,
    'True_Negative' : True_Negative, 'False_Negative' : False_Negative, 'Sensitivity' : Sensitivity,
'Specificity' : Specificity, 'Accuracy' : Accuracy, 'TPR' : TPR, 'FPR' : FPR}
    Compare_Stocks = Compare_Stocks.append(add_row, ignore_index = True) # Add the analysis on the stock to the existing Compare_Stocks dataframe
Compare_Stocks.to_csv("<Your Path>", index = False)  # Save the compiled data on each stock to a csv
```

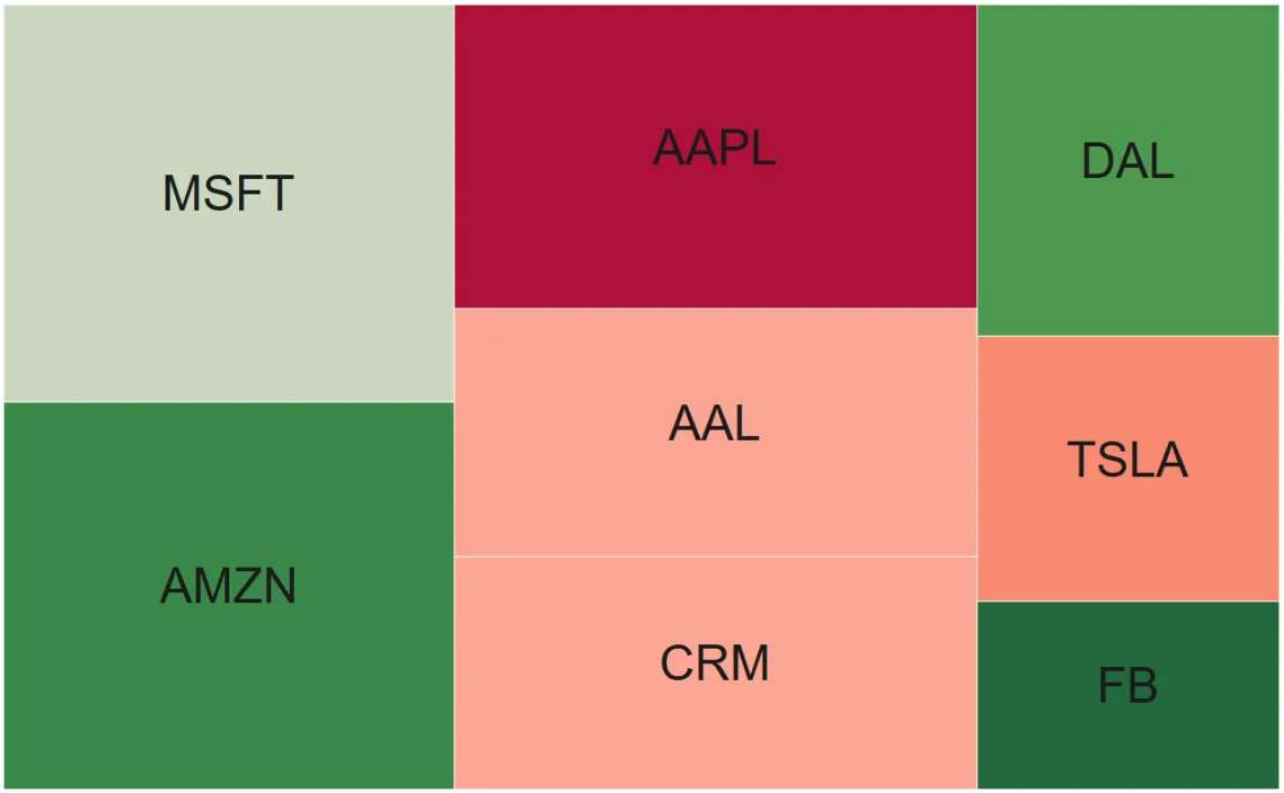
I briefly touched on the results of this analysis in the introduction, but I will reiterate here. After running this code we get a final document displaying the accuracy of the RSI at predicting each stock's future price movements. This is an excerpt from the CSV file that the above code creates:

Company	Days_Observed	Crosses	True_Positive	False_Positive	True_Negative	False_Negative	Sensitivity	Specificity	Accuracy	TPR	FPR
AAL	3,716	1,084	245	288	273	278	47%	49%	48%	47%	51%
AAPL	4,297	1,330	167	638	407	118	59%	39%	43%	59%	61%
AMZN	5,797	1,463	256	520	509	178	59%	49%	52%	59%	51%
CRM	4,047	1,019	166	404	321	128	56%	44%	48%	56%	56%
DAL	3,327	841	186	240	248	167	53%	51%	52%	53%	49%
FB	2,055	477	83	171	173	50	62%	50%	54%	62%	50%
MSFT	6,572	1,499	245	562	481	211	54%	46%	48%	54%	54%
TSLA	2,532	671	116	272	199	84	58%	42%	47%	58%	58%

Analysis output.

Along with the accuracy measurement, we also obtain a lot of other useful measurement tools. Tools like sensitivity and specificity that can be used to create an ROC curve, and themselves tell whether the RSI is better at predicting upward or downward trends at a stock-level granularity.

Using the data in this CSV you can create many different kinds of visualizations, the one I chose to display shows each stock by its accuracy (color) and volume of crosses (size).



A visualization of the observed stocks where color signifies RSI's predictive accuracy and size is the volume of momentum crossovers. (Image by Author)

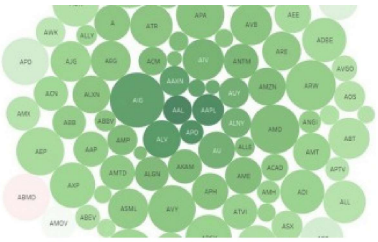
Takeaways

The RSI momentum oscillator is one of the most widely used technical indicators. But like any technical indicator, it does not predict as well for some stocks as it does others.

Using this code will allow you to do this as well as derive other significant insights from this technical indicator.

If you're interested in the automation of technical indicators like this one, check out this article I wrote on the MACD. Also, connect with me on LinkedIn here. I'm always happy to make some new connections!

Related

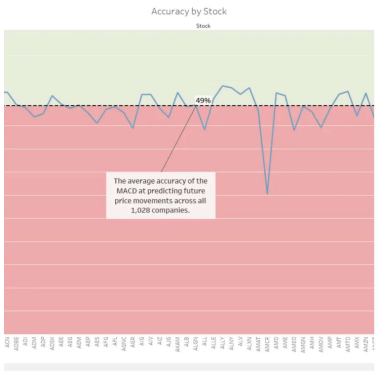


How to Calculate the MACD Using Python
August 23, 2020
In "Technical Analysis"

Calculate On-Balance Volume (OBV) Using Python
August 23, 2020
In "Technical Analysis"

Automating Your Stock Analysis With Python
August 16, 2020
In "Algorithmic Investing"

YOU MAY ALSO LIKE



Leave a comment

Your email address will not be published. Required fields are marked *

COMMENT

*NAME

WEBSITE

SAVE MY NAME, EMAIL, AND WEBSITE IN THIS BROWSER FOR THE NEXT TIME I COMMENT.

NOTIFY ME OF FOLLOW-UP COMMENTS BY EMAIL.

NOTIFY ME OF NEW POSTS BY EMAIL.

POST COMMENT

‹ THE NEXT WARREN BUFFET WILL BE A DATA S...

HOW I LOST ALL OF MY MONEY IN THE STOCK... ›

Subscribe to Our Blog via Email

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 23 other subscribers

Email Address

SUBSCRIBE

Disclaimer

The content on this website is not financial advice. We encourage all visitors to do their own research before making any financial decisions. We do not take any responsibility for the actions taken by users before, during, or after visiting this site.

Follow Us