# Stock Market Analysis

***Qiuhua Liu(11258799) Mingliang Wei(11274244)***

## 1 Introduction

Stock market analysis is an essential part for investors in stock market. Even though it hardly predicts trading halted and stocks plummet in March in U.S., it is still worth learning more about it. When we come to stock market analysis, it includes two approaches: fundamental Analysis and technical Analysis from macroeconomic and microeconomic perspectives. In this tutorial, we mainly focus on the technical analysis which explores historical price and volume, as well as related factors to predict the future price movement and further to decide investment actions.

The tutorial will be organized into 4 sections as follows. The next section gives a brief review of the literature. Section 3 discusses some widely used methods with R in technical analysis and makes a very basic introduction of indicator moving average for the future code example presentation. Section 4 will list some specifically designed packages for stock analysis, along with a few code examples. Section 5 presents two analysis examples using Apple stock data.

## 2 Literature review

Stock market price have been treated to be analyzed as a time series data for a long time in financial industry, several traditional time series forecasting methods can be used to predict the future price and characteristics of the data.Kapila et al. (2015)[6] developed a new hybrid forecasting approach based on ARIMA-ANN for estimating price indices in CSE and has proved it is the most suitable forecasting price method for time series data under high volatility. Najjar[7] measured and modeled volatility of stock market return in Jordan’s capital market by employing both ARCH/GARCH models, characteristics of the volatility includes volatility clustering, Leptokurtic distribution and leverage effect and proved their effectiveness.

Recently, there are more and more machine learning techniques have been impleted in prediction the long-term and short term trend of the stock price. Chatzis et al. (2018)[1] has selected the most significant financial market indicators which can be used to predict stock market tail events and explores the efficacy of ensemble techniques to improve the accuracy of prediction. Khaidem et al. (2016)[2] has proved the robustness of using random forest in predicting the direction of the stock market.

A large number of technical indicators is exhibited in stock market data. And many researches on them showed positive results. Wong et al.(2003)[4] studied the returns of buy-sell trading rules based on two most familar indicators-Moving Averages (MA) and Relative strength index (RSI)- in Singapore stock market. Even though some rules outperform the others, great returns were obtained. Generally "technical indicators can play a useful role in the timing of stock market entry and exits" and practitioners had substantial profits by applying technical indicators. It is not the unique case which demonstrate the effect of technical indicators. Metghalchi et al.(2012)[5] evaluated the effect of trading rule based only on the moving average, and the results indicated that moving average is helpful for predictive power and "technical trading rules can outperform the buy and hold strategy after accounting for transaction costs". Other researches conducted in London stock exchange, in Indian and etc. also showed positive results.

## 3 Brief description of the methods(Brief description of the method)

### *3.1 Time series forecasting*

Time series analysis is a major branch in statistics that mainly focuses on analyzing data set to study the characteristics of the data and extract meaningful statistics in order to predict future values of the series. We will take advantage of using the Garch and Arima models which belongs to time-domain investigating the autocorrelation of the series and is of great use of Box-Jenkins to perform forecast of the series. We will explain how to manipulate the time seires data and how to deal with the trend and seasonality with the time seires, then using the function under ugarch to fit the data and make prediction.

### *3.2 Technical Indicators and Trading Rules*

As we know, multiple analysis could be done due to various technical indicators and therefore bunch of trading rules can be created. In our short example, we mainly conduct those analysis and rules based on moving average, the most frequently used one. More specifically, we use dual moving average, 20-day and 50-day and apply golden cross and death cross to identify the trading signals (buy or sell). First we make a very brife explanation about golden cross and death cross. A golden cross is identified when the short-term moving average crosses over the long-term moving average to the upside, conversely the downside short-term moving average crosses the long-term one.A golden cross suggests a bull market while a death cross suggests a bear market. The crossovers introduce trading signals.A buy signal is triggered by a golden cross and a sell signal is triggered by a death cross.

## 4 Resources available in R

In this part, we split all the resources into various sections:

1. Data resources:where is the stock market data and how to get data from various sources?
2. Data manipulation:data pre-processing
3. Data analysis:visualization, statistical analysis and financial analysis

### *4.1 Data Resources*

#### *Where is the stock market data?*

There are some resources on hand.

Yahoo: Free stock quotes, up to date news, portfolio management resources, international market data, message boards, and mortgage rates that help you manage your financial life.

FRED: Download, graph, and track 149,000 economic time series from 59 sources.

Oanda: Currency information, tools, and resources for investors, businesses, and travelers.

Google: Stock market quotes, news, currency conversions & more.

Quandl: Futures prices, daily. Quandl is a search engine for numerical data. The site offers access to several million financial, economic and social datasets.

TrueFX: Tick-By-Tick Real-Time And Historical Market Rates, Clean, Aggregated, Dealer Prices.

Bloomberg: Financial news, business news, economic news, stock quotes, markets quotes, finance stocks, financial markets, stock futures, personal finance, personal finance advice, mutual funds, financial calculators, world business, small business, financial trends, forex trading, technology news, bloomberg financial news.

Interactive Broker: Interactive Brokers Group, Inc. is an online discount brokerage firm in the United States.

Datastream: Datastream Professional is a powerful tool that integrates economic research and strategy with cross asset analysis to seamlessly bring together top down and bottom up in one single, integrated application.

pwt: The Penn World Table provides purchasing power parity and national income accounts converted to international prices for 189 countries/territories for some or all of the years 1950-2010.

Thinknum: Thinknum brings financial data from a variety of useful sources together on one platform. We use this data to develop applications.

DataMarket: DataMarket brings complex and diverse data together so you can search, visualize and share data in one place and one format.

World bank:offer open access to global development data in various fields over several decades.

***How to get the data?***

For the resources mentioned above, there are a few packages in R to obtain data from them.

1.Package: quantmod with function getSymbols()

The quantmod package may be the most commonly used one right now and it provides easy access to Yahoo! Finance and Google Finance data, along with FRED and Oanda. Use "getSymbols" to get data (default is yahoo). Ex: getSymbols("AAPL", src = "yahoo", from = startdate("2014-01-01"), to =enddate("2015-01-01"))

There also is a wrapper around the getSymbols() function called fidlr, an RSutio addin designed to simplify the financial data downloading process from various providers.

2.Package： BatchGetSymbols with function BatchGetSymbols()

However,GetSymbols() does not aggregate or clean the financial data for several tickers. In the usage of GetSymbols(), each stock will have its own "xts" object with different column names and this makes it harder to store data from several tickers in a single dataframe.

Package BatchGetSymbols could solve this problem. Based on a list of tickers and a time period, BatchGetSymbols will download price data from yahoo finance and organize it so that we don't need to worry about cleaning it ourself. Ex: BatchGetSymbols(tickers = c('FB','MMM','PETR4.SA','abcdef'), first.date = first date,last.date = last date, freq.data = freq data, cache.folder = file.path(tempdir(), 'BGS_Cache') ) If we want to download all stocks in the current composition of the SP500 stock index, this package offers a convenient way to do so. Ex: df.SP500 <- GetSP500Stocks() tickers <- df.SP500$Tickers l.out <- BatchGetSymbols(tickers = tickers, first.date = firs tdate, last.date = last date)

3.Package: Quandl with function Quandl() or Quandl.datatable()

Quandl helps us get millions of financial and economic datasets from hundreds of publishers directly into R.Quandl supports two data formats: time-series (Quandl()) and "datatables" (Quandl.datatable(),used for non-time-series data).

(1)Time-Series Format

```
Make a Basic Data Call:
mydata = Quandl("FRED/GDP")
Note: we need to know the "Quandl code" of each dataset which we want to download.

Change Formats: dataframe, ts, xts or zoo
mydata = Quandl("FRED/GDP", type="raw")  #data frame
mydata = Quandl("FRED/GDP", type="ts")
mydata = Quandl("FRED/GDP", type="xts")
mydata = Quandl("FRED/GDP", type="zoo")

Slice and Dice the Data:
A.set start and end dates:
mydata = Quandl("FRED/GDP", start_date="2001-12-31", end_date="2005-12-31")
B.Request specific columns:
mydata = Quandl(c("FRED/GDP.1", "WIKI/AAPL.4"))

Preprocess the Data:
A.To change the sampling frequency:
mydata = Quandl("FRED/GDP", collapse="annual")
B.To perform elementary calculations on the data:
mydata = Quandl("FRED/GDP", transform="rdiff")
```

(2)Datatables

```
Make a Basic Data Call
mydata = Quandl.datatable("ZACKS/FC", ticker="AAPL")

Set Pagination :Datatables can return large volumes of data due to their format. To reques
t more than one page of data, please turn on pagination.
To turn on pagination:
mydata = Quandl.datatable("ZACKS/FC", ticker="AAPL", paginate=T)

Slice and Dice the Data
A.To request specific columns:
mydata = Quandl.datatable("ZACKS/FC", ticker=c("AAPL", "MSFT"), qopts.columns=c("ticker",
 "per_end_date"))
This returns the ticker and per_end_date columns for the tickers AAPL and MSFT from the ZA
CKS/FC datatable.
B.To filter based on column:
mydata = Quandl.datatable("ZACKS/FC", ticker=c("AAPL", "MSFT"), per_end_date.gt="2015-01-0
1",qopts.columns=c("m_ticker", "per_end_date", "tot_revnu"))
This returns the m_ticker, per_end-date, and tot_revnu columns for the tickers AAPL and MS
FT for every per_end_date greater than 2015-01-01 from the   ZACKS/FC datatable.
```

One point needed to be noted that if we would like to make more than 50 calls a day, we will need to create a free Quandl account and set your API key: Quandl.api_key("YOUR_API_KEY_HERE")

4.Packages TFX with function QueryTrueFX() and ConnectTrueFX()

It connects R to TrueFX(tm) for free streaming real-time and historical tick-by-tick market data for dealable interbank foreign exchange rates with millisecond detail.Results can be returned in different formats (html or csv).In order to make customized requests, you must create an authenticated session which requires a username and password. Ex: yen <- ConnectTrueFX("USD/JPY,EUR/JPY,GBP/JPY,AUD/JPY,CAD/JPY,CHF/JPY", username=username, password=password) QueryTrueFX(yen)

5.Package Rblpapi

Rblpapi provides R with access to data and calculations from Bloomberg Finance L.P. via the API libraries provided by Bloomberg. There are multiple ways to fetch data from the Bloomberg financial data application,such as function bdh(),bdp(),bds(),beqs().

6.Package IBrokers

This package provides native R access to Interactive Brokers Trader Workstation API.Some commonly used funtions are below. twsConnect():used to establish, check or terminate a connection to TWS. The function returns a twsConnection object for use in subsequent TWS API calls. twsConnectionTime():provides the time when the connection to the TWS was made. reqAccountUpdates():used to request and view account details from Interactive Brokers. wsContract():to create, test or coerce a twsContract for use in API calls. The value returned by the function is a "twsContract" object. twsCurrency():is similar to wsContract() and a wrapper to twsContract to make ‘currency/FX’ contracts easier to specify. reqMktData():allows for streaming market data to be handled in R. reqHistoricalData():used to request historical data from TWS. placeOrder():used to place or cancel an order to the TWS. Ex: tws = twsConnect(port=7497) symbol = twsSTK("AAPL") data_AAPL = reqHistoricalData(tws, symbol)

7.Package rdatastream with function ds()

RDatastream is a R interface to the Thomson Dataworks Entreprise SOAP API (non free), with some convenience functions for retrieving Datastream data specifically. This package requires valid credentials for this API.Funtion ds() request data from Thomson Reuters Datastream SOAP API. It is similar to ConnectTrueFX() in package TFX which also needs username and password.

8.Package pwt

The Penn World Table provides purchasing power parity and national income accounts converted to international prices for 189 countries/territories for some or all of the years 1950-2010. There are 6 versions:pwt5.6, pwt6.1,pwt6.2,pwt6.3,pwt7.0 and pwt7.1. Different version has different base year and time horizon. The latest version 7.1 of the package was developed in July 2013. Ex: data("pwt7.1") to obtain purchasing power parity and national income accounts in international prices for 189 countries over 1950–2010 (2005 as base year).

9.Package fImport

fImport offers access to Yahoo, FRED and Oanda.Funtions fredImport() and fredSeries() import Market Data from the Fred, yahooSeries() and yahooImport() from Yahoo, and oandaSeries() and oandaImport() from Oanda. All of them can return data of timeseries.

10.Package Thinknum

This package interacts directly with the Thinknum API to offer data in a number of formats usable in R. Function Thinknum() pulls data from the Thinknum API.Once we find the expression associated with the data we'd like to load into R on ThinkNum, copy the ThinkNum expression and past it into the function. Ex: Thinknum("total_revenue(goog)")

### *4.2 Data manipulation*

#### *How to pre-process the time series data in R?*

Same as the weather data, industry forecast, stock price is also a typical time series data. We will introduce the methods we use to process this time series data in R within this chapter, but before this, we would like to have a quick introduction of the time series data to help you better understand the following analysis.

Any metric that is measured over regular time intervals makes a Time Series. The **ts()** function in R will help us to convert a numeric vector or matrix into an R time series object. These are vectors or matrices with class of "ts" (and additional attributes) which represent data which has been sampled at equispaced points in time. We can specity the start time and end time in ts() fucntion. Also by giving different values of **frequency()** in ts() function, we can decide the which format our time series data will be displayed, e.g., we could use a value of 7 for **frequency()** when the data are sampled daily, 4 for quarterly and 12 for yearly.

Since each point of a time series data has a time stamp, if we shift time base by a given number of periods, a Lag of time series is created. Lags of a time series are often used as explanatory variables to model the actual time series itself. The underlying reasoning is that the state of the time series few periods back may still has an influence on the series current state. Under ts class, **lag()** creates a lagged version of a time series, shifting the time base **forward** by a given number of observations. On the other hand, **Lag()** shifting the time base backwards by the given number of observations. **lag()** and **Lag()** create a single lagged series, while **lags()** and **Lags()** can create a multivariate series with several lags at once.

Missing values and outliers are also very common in time series data, under ts class, we can use **tsclean()** to estimate and replace missing values and outliers, linear interpolation is used on the (possibly seasonally adjusted) series, we can also choose Box-Cox as our transformation parameter.

Time series data is a sequential of datasets which can be measured over regular time intervals. Each data point $Y_t$ at time $t$ in a time series can be expressed as a decomposition of 3 components, namely, Seasonality($S_t$), Trend($T_t$) and Error($\epsilon_t$, also known as Whit Noise).

- For Additive Time Series: $Y_t = S_t + T_t + \epsilon_t$
- For Multiplicative Time Series: $Y_t = S_t \text{x} T_t \text{x} \epsilon_t$

So a multiplicative time series can be converted to additive by taking a log of the time series. After our getting the time series data, we can use the **decompose()** function to extract the information of time series and splits it into seasonality, trend and error components. When your extract the info from time seires, you can also choose which kind of time series data you have, additive or multiplicative.

An alternative to decomposition for removing trends is differencing(**diff()**), it can also help us to remove trends or seasonal effects. there are three arguments in diff() function which are: x(the data), lag(the lag at which to difference), and differences(the order of differencing). If we set differences=1, the function will operate a first-differencing to remove a linear trend of time series, twice-differencing will remove a quadratic trend. In addition, first-differencing a time series at a lag equal to the period will remove a seasonal trend (e.g., set lag=12 for monthly data)

***Stationary of time series data***

Stationary is a very important feature of the time series data, not only in analysis but also have a great effect on prediction. To define the stationary of the time series data, it need to hold the following conditions true:

- The mean value of time-series is constant over time, which implies, the trend($T_t$) component is nullified.
- The variance does not increase over time.
- Seasonality($S_t$) effect is minimal.

This means it is devoid of trend or seasonal patterns, which makes it looks like a random white noise irrespective of the observed time interval.

***How to test stationary of time series data***

To test the stationarity of the time series dataset, several functions in R can be used: the **Augmented Dickey Fuller Test (ADF)** is unit root test for stationarity under **tseries** package, different from Dickey-Fuller test, ADF test allows for higher-order autoregressive processes by including $\Delta y_{t-p}$ in the model, our test will still be if $\gamma = 0$ as Dickey Fuller Test:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \delta_2 \Delta y_{t-2} + \dots$$

The null hypothesis for the test is that the data are non-stationary. We want to reject the null hypothesis for this test, so we want a p-value of less that 0.05 (or smaller).

There is also a wrapper of different stationary test methods in function **stationary.test** under **aTSA** package, it contains ADF, Phillips-Perron test(PP.test) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test.

There are also some functions which are available in R for us to use as a graphical method to test stationary. Firstly, let's dig into the definition of **Autocorrelation** and **Partial-Autocorrelation**. Autocorrelation is the correlation of a Time Series with lags of itself. This is a significant metric because:

- It shows if the previous states (lagged observations) of the time series has an influence on the current state. In the autocorrelation chart, if the autocorrelation crosses the dashed blue line (an approximate 95% confidence interval), it means that specific lag is significantly correlated with current series.
- It is used commonly to determine if the time series is stationary or not. A stationary time series will have the autocorrelation fall to zero fairly quickly but for a non-stationary series it drops gradually.

Partial Autocorrelation is the correlation of the time series with a lag of itself, with the **linear dependence** of all the lags between them removed. In R, we can use **acf()** to generate autocorrelation plot and using **pacf()** to generate partial autocorrelation plot.

***How to make a time series stationary?***

In real life, time series data are already exist with Seasonality($S_t$) and Trend($T_t$) which will make it more difficult to model and make the prediction, so we always use the decompose() and diff() methods to remove the trend and seasonality to make the time series data stationary. For example, if a time series data is seasonal, a better approach is to difference with respective season's data points to remove seasonal effect. We can use **nsdiffs(data)** function from **forecast** package to get the number for seasonal differencing needed. By using the function **frequency()**, we can get the lag at which we will difference and put these data we get into the **diff()** function. After that, if needed, difference it again with successive data points. For most time series patterns, 1 or 2 differencing is necessary to make it a stationary series.

***4.3 Data Analysis***

***4.3.1 Visualization***

***Static Plot: ggplot2***

As we know, ggplot2 is a very popular package for graphic plot in R. Two main plotting functions are qplot() and ggplot(). qplot() is useful for plotting quickly, but ggplot() is good for systemic plotting. Various kinds of plots are included in qplot() by specifiying value of the parameter "geom" in the arguments of qplot() and in ggplot() by adding different arguments geom_XXX().

Scatter plot:is useful to depict relationship between "x" and "y". qplot(...,geom="point") VS ggplot(...)+geom_point() Box plot:fit for two variables one of which is continuous and the other is categorical. qplot(...,geom="boxplot") VS ggplot(...)+geom_boxplot() Line plot:is useful to show time trend and how two variables are related. If the horizontal variable ("x") does not have any duplicated values, it is better to use line plot instead of scatter plot. qplot(...,geom="line") VS ggplot(...)+geom_line() Histogram: is useful to visualize the distribution of single continuous variable.qplot(...,geom="histogram") VS ggplot(...)+geom_histogram() Density plot: is similar to histogram but there is no grouping as in histogram but the plot is smothed. qplot(...,geom="density") VS ggplot(...)+geom_freqpoly() Bar chart: is similar to histogram but it is for discrete data. qplot(...,geom="bar") VS ggplot(...)+geom_bar()

Multiple other arguments in the functions also help to improve the appearance of plot, like size, color, themes and etc.. Additionally, there is a package named ggthemes to offer more options about the themes in ggplot().

***Dynamic Plot: dygraphs, plotly***

The package dygraphs can produce dynamic graphs for time series data. Those graphs are fully interactive: as our mouse moves over the series individual values are displayed. We can also select regions of the graph to zoom into (double-click zooms out). There are four kinds dynamic plots:standard dynamic(dygraph()), shading(dyShading(dygraph(),...)),event line(dyEvent(dygraph(),...)) and candle chart(dyCandlestick(dygraph())). We can utilise multiple-nested dyEvent() to produce multiple event lines in the plot. Ex:dyEvent(dyEvent(dygraph(),...),...) corresponds two event lines in the plot. Detailed illustrations are at "dygraphs for R" (https://rstudio.github.io/dygraphs/index.html (https://rstudio.github.io/dygraphs/index.html)).

Another package plotly with function plot_ly() also could produce nice interactive plots. Some examples are at "Plotly R Open Source Graphing Library"(https://plotly.com/r/ (https://plotly.com/r/)).

***Financial Plot: quantmod, plotly, candlesticks/CandleStickPattern, stocks***

1.Package quantmod

Package quantmod means "quantitative financial modelling framework", which specifies, builds, trades, and analyses quantitative financial trading strategies. We have see quantmod when we try to download data from websits with function getSymbols(). Here we will see another main role quantmod plays:charting. It produce nice plots of line, bars and candlestick in a convenient way:lineChart(),barChart(),and candleChart. Ex:candleChart(AAPL, up.col = "black", dn.col = "red", theme = "white") Those plots can be obtained in quantmod by chartSeries() in which we can specify the style of chart to draw in parameter "type". The possible values for "type" are "auto", "candlesticks", "matchsticks", "bars" and "line". Another nice feature is that it offers charting with various technical indicators (we will come to those indicators in the next setion.) We can directly use all the indicator charts, such as addBBands(), after chartSeries() or candleChart(), or we can set the value of argument "TA" in chartSeries() to addBBands(). Ex: chartSeries(AAPL,subset='2007-05::2009-01',theme=chartTheme('white')) addBBands(n=20,sd=2) or chartSeries(AAPL,subset='2007-05::2009-01',theme=chartTheme('white'),TA = 'addBBands(...)')

2.Package plotly

Package plotly could be an alternative for financial charts with function plot_ly(). Ex: fig <- df %>% plot_ly(x = ~Date, type="candlestick", open = ~AAPL.Open, close = ~AAPL.Close, high = ~AAPL.High, low = ~AAPL.Low) For details, visit https://plotly.com/r/financial-charts/ (https://plotly.com/r/financial-charts/)

3.Package candlesticks/CandleStickPattern

Some consecutive candlesticks form patterns that we use for analysis or trading purposes. There are bunch of candlestick patterns. Two packages candlesticks and CandleStickPattern have been developing to recognize those patterns. Package candlesticks is at R-Forge(https://r-forge.r-project.org/projects/candlesticks/ (https://r-forge.r-project.org/projects/candlesticks/)), whereas CandleStickPattern is at GitHub and details is at http://htmlpreview.github.io/? https://github.com/kochiuyu/CandleStickPattern/blob/master/vignettes/CandleStick.html (http://htmlpreview.github.io/? https://github.com/kochiuyu/CandleStickPattern/blob/master/vignettes/CandleStick.html)

4.Package stocks

Package stocks also has a few functions to produce graphs, such as growth_graph(), gains_graph() and etc. For more details https://cran.r-project.org/web/packages/stocks/stocks.pdf (https://cran.r-project.org/web/packages/stocks/stocks.pdf)

#### *4.3.2 Statistical Analysis*

As usual, we can do basic statistical analysis, such as central tendency and dispersion, and we also can do regression analysis with generalized linear model and neural network to identify the underlying relationship between price and various indicators. Besides that, we have numerous packages targeting time series data for stock market data because of the nature of stock data - time series. Some examples of applications with those packages in time series could be visited at https://bookdown.org/singh_pratap_tejendra/intro_time_series_r/may(Introduction (https://bookdown.org/singh_pratap_tejendra/intro_time_series_r/may(Introduction) to Time Series Analysis and Forecasting in R), and some applications in stock market data is at https://rpubs.com/kapage/523169(Stock (https://rpubs.com/kapage/523169(Stock) Price Forecasting Using Time Series Analysis, Machine Learning and single layer neural network Models). Here we just list a few frequently used packages about ARIMA and GARCH methods.

1.Package forecast

Package forecast has multiple methods and tools for displaying and analysing univariate time series forecasts including exponential smoothing via state space models and automatic ARIMA modelling.For example, can use Arima(), auto.arima() and etc. Under Arima(), we can train AR, MA, ARMA model. In addition to predict(), the future price will be obtained. For detailed introduction about package forecast visit https://cran.r-project.org/web/packages/forecast/forecast.pdf (https://cran.r-project.org/web/packages/forecast/forecast.pdf)

2.Package vars

Stock price is not just related to the historical price, but also related to other factors. So mulitvariate analysis is needed. The package vars could do so by implementing vector autoregressive,structural vector autoregressive and structural vector error correction models with functions VAR(), SVAR() and SVEC() respectively. A more comprehensive illustration is in the artice "VAR, SVAR and SVEC Models: Implementation Within R Package vars"(Pfaff)(https://cran.r-project.org/web/packages/vars/vignettes/vars.pdf (https://cran.r-project.org/web/packages/vars/vignettes/vars.pdf)). It includes mathmatical models and code examples.

3.Package rugarch

Variance in time series may be time dependent, and methods like ARIMA can't handle this problem, whereas ARCH (Autoregressive Conditional Heteroskedasticity) and its extension GARCH (Generalized Autoregressive Conditional Heteroskedasticity) provide a way to model the change. The package rugarch is usually used to train ARCH and GARCH model. It is for univariate GARCH modelling and contains a number of GARCH models including IGARCH, EGARCH, GJR, APARCH, FGARCH, Component-GARCH, multiplicative Component-GARCH for high frequency returns and the realized-GARCH model, as well as a very large number of conditional distributions including (Skew)-Normal, (Skew)-GED, (Skew)-Student (Fernandez/Steel), (Skew)-Student (GH), Normal Inverse Gaussian (NIG), Generalized Hyperbolic (GH) and Johnson?s SU (JSU). The conditional mean equation includes ARFIMA and ARCH-in-mean, and is estimated in a joint step with the GARCH model. Both the conditional mean and variance parts allow for external regressors to be used. A comprehensive set of methods to work with these models are implemented, and include estimation, filtering, forecasting, simulation, inference tests and plots, with additional functionality in the form of the GARCH bootstrap, parameter uncertainty via the GARCH distribution function, misspecification tests (Hansen's GMM and Hong & Li Portmanteau type test), predictive accuracy tests (Pesaran & Timmermann, Anatolyev & Gerko), and Value at Risk tests (VaR Exceedances and Expected Shortfall tests). For more details about package rugarch, visit https://www.rdocumentation.org/packages/rugarch/versions/1.4-2 (https://www.rdocumentation.org/packages/rugarch/versions/1.4-2)

There is also a book named "GARCH Models: Structure, Statistical Inference and Financial Applications" to do a step-by-step explanation about those models.

Finally, We tak a GARCH(1,1) with ARMA(1,1) model training as an example: a<-runif(1000,min=0,max=100) Spec <- ugarchspec(variance.model = list(model = "sGARCH",garchOrder = c(1, 1)), mean.model = list(armaOrder = c(1, 1), distribution.model = "std") Garch <- ugarchfit(spec = Spec, data = a)

Additionally, a package called ‘fGarch’ could provide a collection of functions to analyze and model heteroskedastic behavior for financial time series.

#### *4.3.3 Financial Analysis*

***Technical Analysis: TTR (quantmod), stocks, PerformanceAnalytics***

1. Package TTR

Technical analysis is more related various of technical indicators. Package TTR (Technical trading rule) can do this for us, which could be loaded separately or loaded when we load quantmod package. There are bunch of indicators in TTR, such as Simple Moving Average (SMA()),Bollinger band (BBands()) and etc. EX: SMA(Cl(AAPL),n=20) For more details about possible indicators, visit https://cran.r-project.org/web/packages/TTR/TTR.pdf (https://cran.r-project.org/web/packages/TTR/TTR.pdf) We also care about the return from certain stock. Package quantmod has periodReturn() to calculate periodic returns given a set of prices. We can set the parameter "period" with "daily", "weekly", "monthly", "quarterly" and "yearly", or directly use dailyReturn(),weeklyReturn() and etc.

1. Package stocks and PerformanceAnalytics

Package stocks contains multiple functions for analyzing performance of stocks or other investments, whereas package PerformanceAnalytics provides a collection of econometric functions for performance and risk analysis. For more details, stocks is at https://cran.r-project.org/web/packages/stocks/stocks.pdf (https://cran.r-project.org/web/packages/stocks/stocks.pdf), and PerformanceAnalytics at https://cran.r-project.org/web/packages/PerformanceAnalytics/PerformanceAnalytics.pdf (https://cran.r-project.org/web/packages/PerformanceAnalytics/PerformanceAnalytics.pdf).

***Trading: quantmod, quantstrat,blotter, PerformanceAnalytics, IKTrading, FinancialInstruments***

Technical indicators is not just a way to evaluate the performance of a stock, but also a tool to identify trading signals and create trading rules and further trading strategy. Trading signals and various rules can be set based on combination of indicators no matter they are existing common indicators or customized ones. During this whole process, multiple packages will be put together. Even though it is impossible to list all the popular examples here, we will go to the main used packages in this part.

Package quantstrat (Quantitative Strategy Model Framework) is to specify, build, and back-test quantitative financial trading and portfolio strategies. With the following packages, it provides a flexible framework for backtesting that should almost allow for any trading strategy to be backtested. Main functions and some examples are at https://www.rdocumentation.org/packages/quantstrat/versions/0.16.7 (https://www.rdocumentation.org/packages/quantstrat/versions/0.16.7).

Package blotter is to provide tools for transaction-oriented trading systems development. It helps simplify tasks of recording tranasction of stocks, keeping track of cash and stock holding, and evaluating return. This package is still under development. Some details are here https://rdrr.io/rforge/blotter/man/ (https://rdrr.io/rforge/blotter/man/).

Package IKTrading provides a collection of various trading tools of which some help customize indicators, signals, and rules. List of functions is at https://www.rdocumentation.org/packages/IKTrading/versions/1.0 (https://www.rdocumentation.org/packages/IKTrading/versions/1.0).

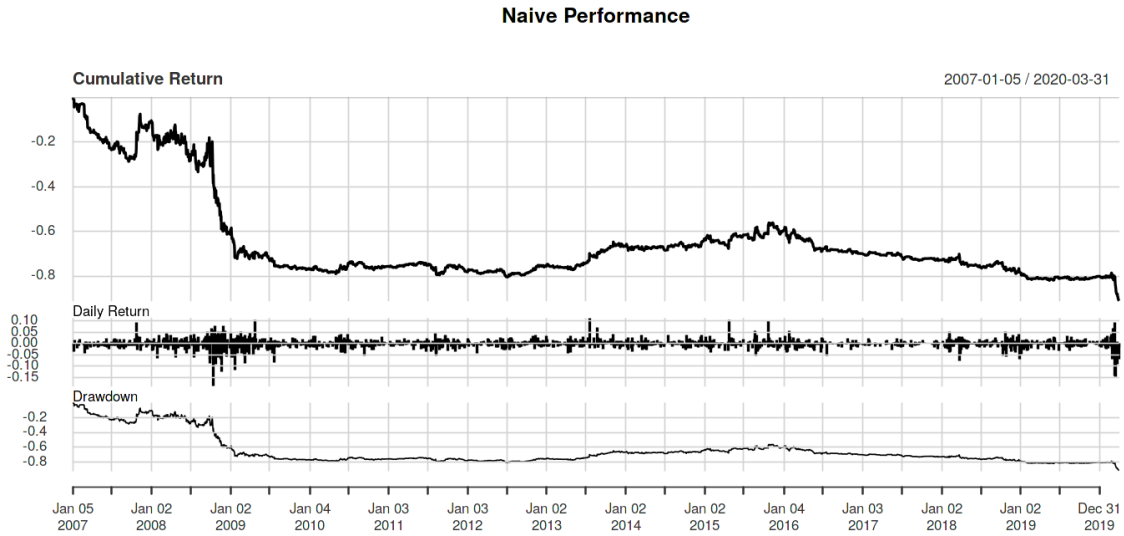
Packge FinancialInstruments is a companion package to blotter and defining meta-data and relationships for different financial instruments. List of multiple functions is at https://www.rdocumentation.org/packages/FinancialInstrument/versions/1.3.1 (https://www.rdocumentation.org/packages/FinancialInstrument/versions/1.3.1).

Here we create a simple sell-buy trading strtegy based on simple filter rule (trading signal) as an example to illustrate the process.

In [45]:
```
library(quantmod)
library(PerformanceAnalytics)
getSymbols("MSFT")
price <- Cl(MSFT) # close price
r <- price/Lag(price) - 1 # % price change
delta<-0.005 #threshold
signal <-c(NA) # first signal is NA
# sell-buy rule, using close price here
for (i in 2: length(Cl(MSFT))){
  if (r[i] > delta){
    signal[i]<- 1
  } else if (r[i]< -delta){
    signal[i]<- -1
  } else
    signal[i]<- 0
}
signal<-reclass(signal,Cl(MSFT)) # Assign time to action variable using reclass;

trade1 <- Lag(signal) # trade based on yesterday signal
ret1<-dailyReturn(MSFT)*trade1 # calculat returns based on signal
names(ret1) <- 'Naive'
#Performance Summary
charts.PerformanceSummary(ret1)
```

Out[45]: 'MSFT'

Out[45]:




One thing need to be mentioned as supplementary. Packages curl,FinancialInstruments,blotter and quantstrat are dependent and need to be installed in order. Package devtools is also required to install blotter and quantstrat. Also noted that installation of package curl and quantstrat may meet the problem under 3.6.1 version of R. The problem can be solved after upgrading R to 3.6.2.

As an end, we would like to mention another Package tidyquant (Tidy Quantitative Financial Analysis). It provides a convenient wrapper to various 'xts', 'zoo', 'quantmod', 'TTR' and 'PerformanceAnalytics' package functions and returns the objects in the tidy 'tibble' format. The main advantage is being able to use quantitative functions with the 'tidyverse' functions including 'purrr', 'dplyr', 'tidyr', 'ggplot2', 'lubridate', etc. For more information of documentation and examples, see https://www.rdocumentation.org/packages/tidyquant/versions/1.0.0 (https://www.rdocumentation.org/packages/tidyquant/versions/1.0.0).

**5 Data analysis example(Examples of analyzes with data.(core). Should be written as an example. https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html (https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html))**

In [47]:
```
library(xts)
library(zoo)
library(magrittr)
library(ggplot2)
library(gridExtra)
library(grid)
library(tseries)
library(forecast)
library(rugarch)
```

### *5.1 Datasets*

The dataset we used throughout this chapter are the stock prices of **Apple**, including the highest and lowest price, also the open price and close price, adjusted price on a daily basis. The time period of the stock market data are from 2014-01-01 to 2018-10-01.

In [3]:
```
# Preprocessing of the data
start <- as.Date("2014-01-01")
end <- as.Date("2019-10-01")
#get the data for Apple/Microsoft/Google
getSymbols(c("AAPL"), src = "yahoo", from = start, to = end,getSymbols.warning4.0=FALS
E)
```

Out[3]: 'AAPL'

Let's take a quick look of the dataset we get by using getSymbols(). The features of the dataset containing open and close prices, highest and lowest prices, adjusted price, also the volume of transactions during one day, the dates which have a format "$yyyy-mm-dd$" will be set as the index automatically.

In [4]:
```
head(AAPL)
```

Out[4]:

| | AAPL.Open | AAPL.High | AAPL.Low | AAPL.Close | AAPL.Volume | AAPL.Adjusted |
|---|---|---|---|---|---|---|
| 2014-01-02 | 79.38286 | 79.57571 | 78.86000 | 79.01857 | 58671200 | 70.93883 |
| 2014-01-03 | 78.98000 | 79.10000 | 77.20428 | 77.28286 | 98116900 | 69.38062 |
| 2014-01-06 | 76.77857 | 78.11429 | 76.22857 | 77.70428 | 103152700 | 69.75897 |
| 2014-01-07 | 77.76000 | 77.99429 | 76.84571 | 77.14857 | 79302300 | 69.26006 |
| 2014-01-08 | 76.97285 | 77.93714 | 76.95571 | 77.63715 | 64632400 | 69.69869 |
| 2014-01-09 | 78.11429 | 78.12286 | 76.47857 | 76.64571 | 69787200 | 68.80863 |

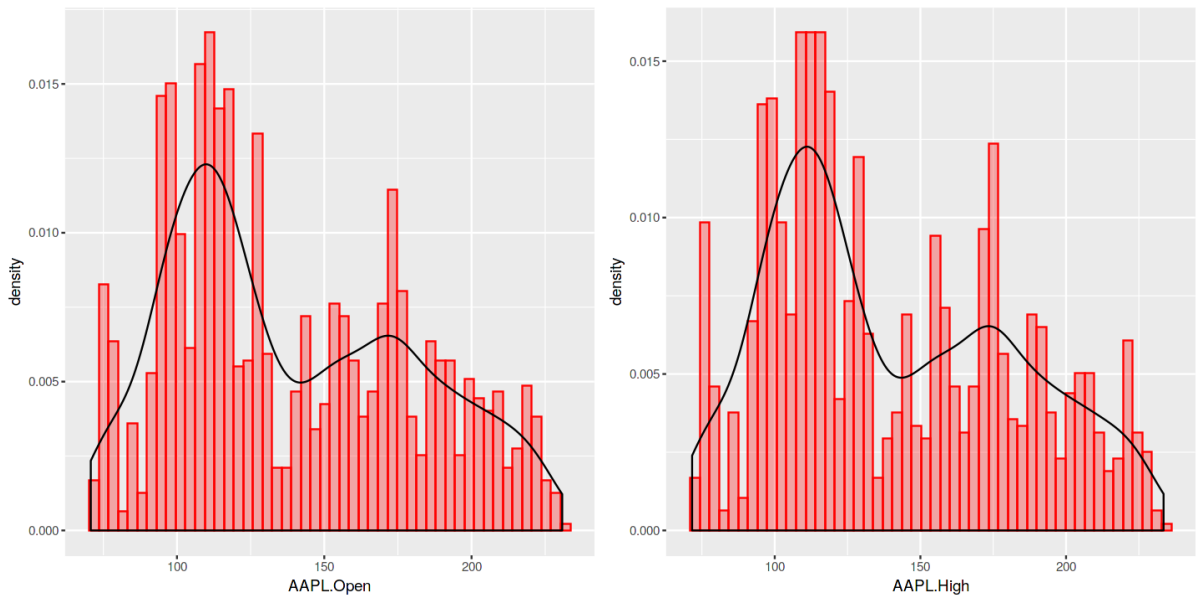
Transform the dataset we get from xts to dataframe in order to be used in the time series forecasting packages we have.

In [5]:
```
AAPL=data.frame(date=index(AAPL), coredata(AAPL))
i_stock=AAPL
```

Before going deeper into the analysis of the data, we need to check the distributions of different types of the prices in order to find some hidden regulations of the prices.

In [6]:
```
# Set the size of the plots
options(repr.plot.width=12, repr.plot.height=6)
# Get the distribution of the open/close/high low prices of Apple
p1 = ggplot(i_stock, aes(AAPL.Open)) + geom_histogram(bins = 50, aes(y = ..density..),
col = "red", fill = "red", alpha = 0.3) + geom_density()# + xlim(c(0, 1000))
p2 = ggplot(i_stock, aes(AAPL.High)) + geom_histogram(bins = 50, aes(y = ..density..),
col = "red", fill = "red", alpha = 0.3) + geom_density()# + xlim(c(0, 1000))
p3 = ggplot(i_stock, aes(AAPL.Low)) + geom_histogram(bins = 50, aes(y = ..density..),
col = "red", fill = "red", alpha = 0.3) + geom_density()# + xlim(c(0, 1000))
p4 = ggplot(i_stock, aes(AAPL.Close)) + geom_histogram(bins = 50, aes(y = ..density..
), col = "red", fill = "red", alpha = 0.3) + geom_density()# + xlim(c(0, 1000))
grid.arrange(p1,p2, nrow=1,ncol=2)
```

Out[6]:




### *5.3 Time serires Forecasting*

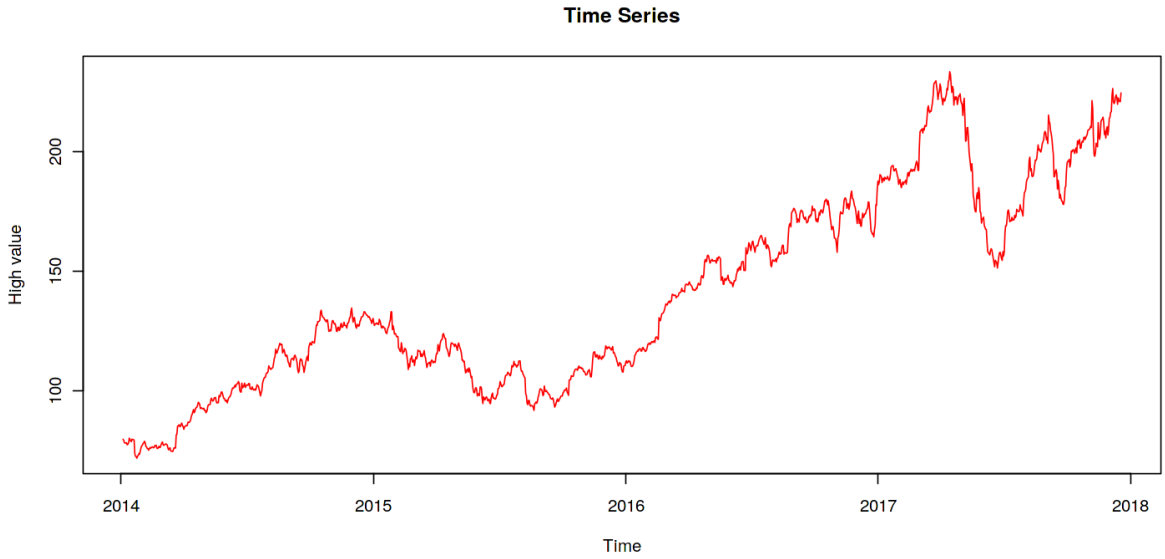
We will create a function creat_ts() here to choose the type of price we want to explore.

In [7]:
```
## Create a daily Date object
inds <- seq(as.Date("2014-01-04"), as.Date("2018-10-01"), by = "day")

# For ts function, freq: 365.5=daily1=annual, 4=quartly, 12=monthly
create_ts <- function(col_idx){
  ## Create a time series object
  i_ts <- as.numeric(i_stock[,col_idx]) %>%
    tsclean(replace.missing = TRUE, lambda = NULL) %>%
    ts(start = c(2014, as.numeric(format(inds[1], "%j"))),
       frequency = 365.5)
  return(i_ts)
}
```

We choose the highest price during a day as the price we want to explore, and by using plot() function to draw the price changing graph throughout the time period we are interested in.

In [8]:
```
# Check the High price of Apple stock at a daily basis
i_ts = create_ts(which(colnames(i_stock) == "AAPL.High"))
plot.ts(i_ts, xlab = "Time", ylab = "High value", main = "Time Series", col = "red")
```

Out[8]:



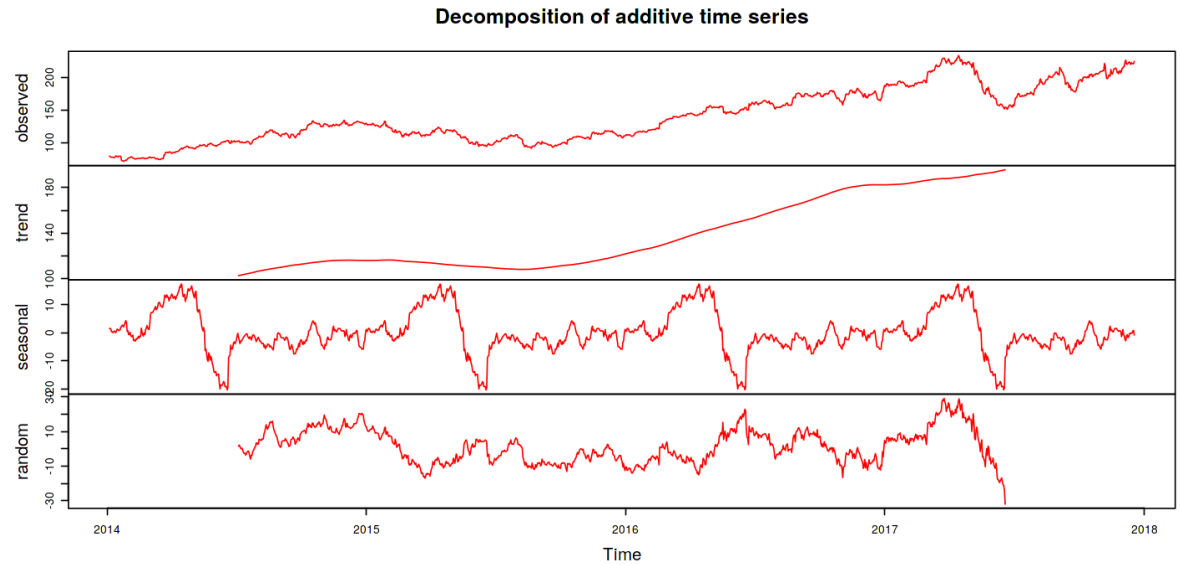

Using the ADF to test stationary of the time series dataset, since we get a p-value 0.6009 which is higher than 0.5, so we have to accept the null hypothesis that the dataset is non-stationary.

In [9]:
```
# Augmented Dickey-Fuller test to check if the high price curve is stationary
adf.test(i_stock[,which(colnames(i_stock) == "AAPL.High")], alternative = "stationary"
, k = 0)
```

Out[9]:
```
        Augmented Dickey-Fuller Test

data:  i_stock[, which(colnames(i_stock) == "AAPL.High")]
Dickey-Fuller = -1.9466, Lag order = 0, p-value = 0.6009
alternative hypothesis: stationary
```

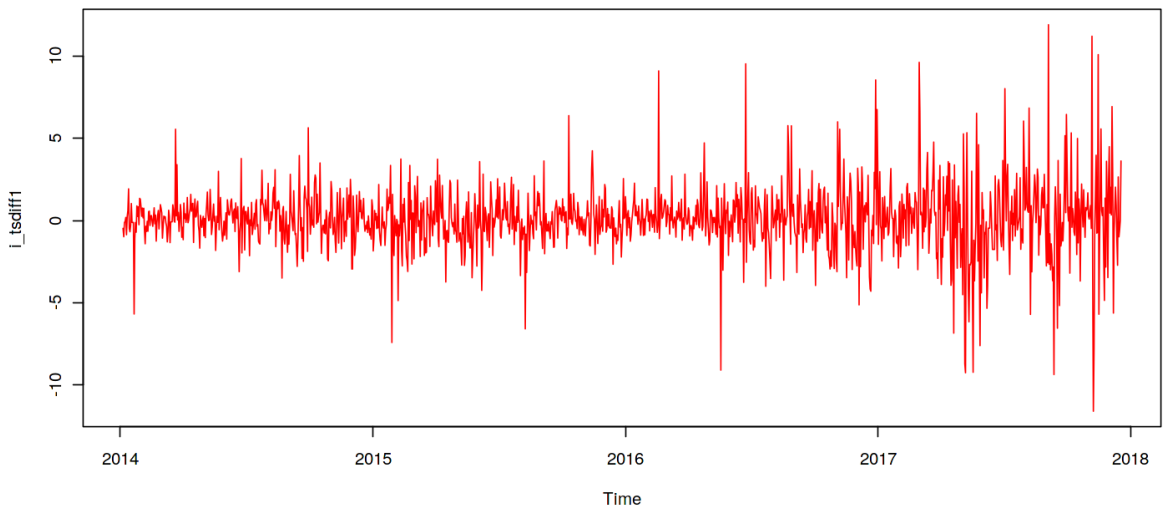
In [10]:
```
# Decomposing of the time series
i_tscomponents <- decompose(i_ts)
plot(i_tscomponents, col = "red")
```

Out[10]:



In [11]:
```
i_tsdiff1 <- diff(i_ts, differences=1)
plot.ts(i_tsdiff1, col = "red")
# The time series (above) appears to be stationary.
```

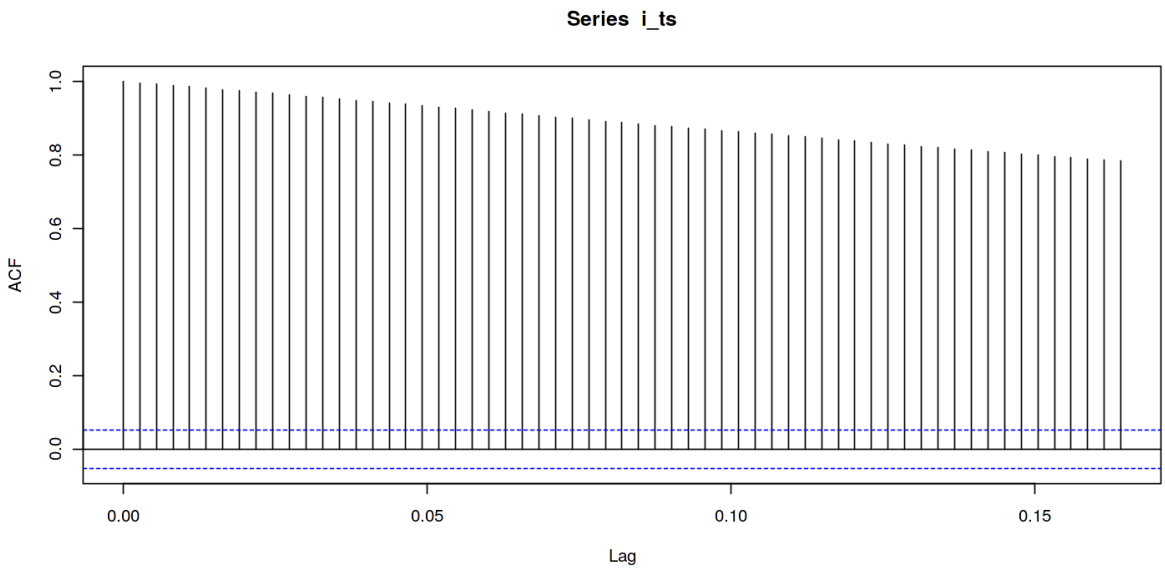
Out[11]:




This is the data before using diff() function, we can see clearly that the autocorrelation drops down gradually, which can prove that the data is still non-stationary.

In [12]:
```
acf(i_ts, lag.max=60)              # plot a correlogram
```

Out[12]:

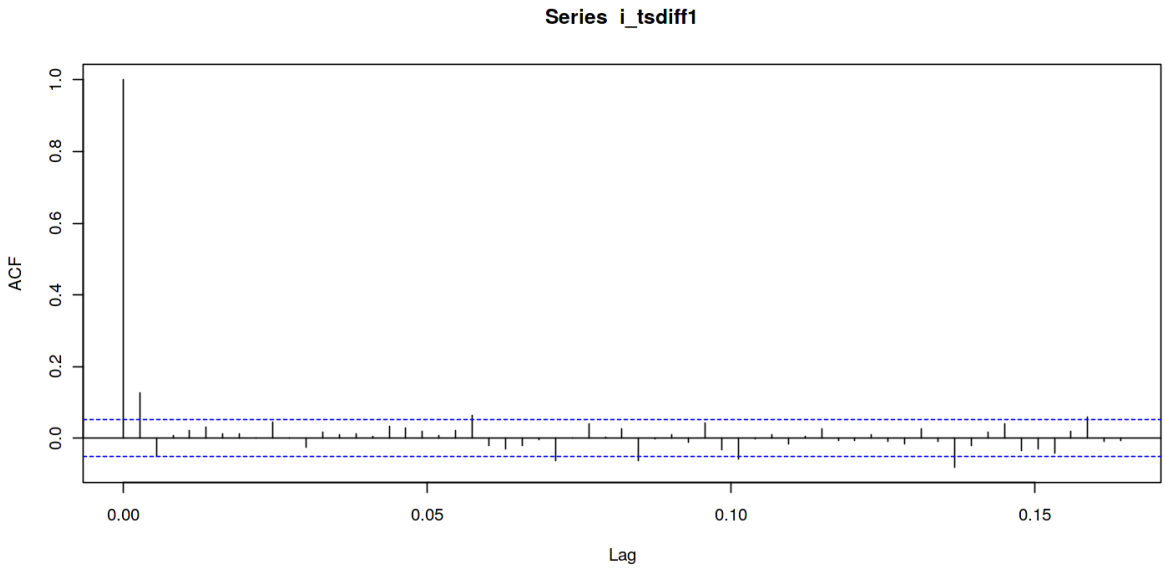



We can see that the autocorrelation drops down quickly after lag=0, which means that the time series data is stationary.

In [13]:
```
acf(i_tsdiff1, lag.max=60)             # plot a correlogram
acf(i_tsdiff1, lag.max=60, plot=FALSE) # get the autocorrelation values
```

Out[13]:
```
Autocorrelations of series 'i_tsdiff1', by lag

0.00000 0.00274 0.00547 0.00821 0.01094 0.01368 0.01642 0.01915 0.02189 0.02462
  1.000   0.127  -0.050   0.009   0.021   0.031   0.013   0.012   0.001   0.044
0.02736 0.03010 0.03283 0.03557 0.03830 0.04104 0.04378 0.04651 0.04925 0.05198
  0.000  -0.024   0.017   0.009   0.013   0.006   0.034   0.028   0.019   0.008
0.05472 0.05746 0.06019 0.06293 0.06566 0.06840 0.07114 0.07387 0.07661 0.07934
  0.022   0.063  -0.020  -0.030  -0.020  -0.005  -0.062   0.001   0.040   0.002
0.08208 0.08482 0.08755 0.09029 0.09302 0.09576 0.09850 0.10123 0.10397 0.10670
  0.026  -0.062  -0.002   0.010  -0.010   0.042  -0.032  -0.059  -0.003   0.010
0.10944 0.11218 0.11491 0.11765 0.12038 0.12312 0.12585 0.12859 0.13133 0.13406
 -0.015   0.005   0.025  -0.006  -0.007   0.009  -0.008  -0.017   0.027  -0.009
0.13680 0.13953 0.14227 0.14501 0.14774 0.15048 0.15321 0.15595 0.15869 0.16142
 -0.080  -0.020   0.016   0.039  -0.034  -0.031  -0.042   0.019   0.058  -0.009
0.16416
 -0.008
```

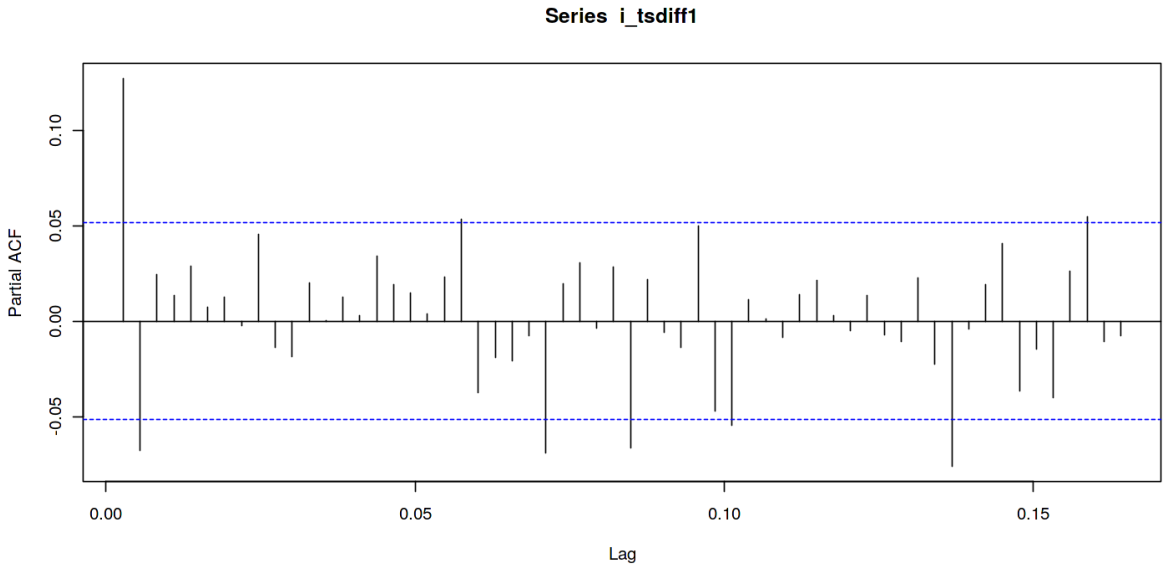
Out[13]:



In [14]:
```
pacf(i_tsdiff1, lag.max=60)             # plot a partial correlogram
pacf(i_tsdiff1, lag.max=60, plot=FALSE) # get the partial autocorrelation values
```

Out[14]:
```
Partial autocorrelations of series 'i_tsdiff1', by lag

0.00274 0.00547 0.00821 0.01094 0.01368 0.01642 0.01915 0.02189 0.02462 0.02736
  0.127  -0.068   0.024   0.014   0.029   0.007   0.013  -0.002   0.046  -0.014
0.03010 0.03283 0.03557 0.03830 0.04104 0.04378 0.04651 0.04925 0.05198 0.05472
 -0.018   0.020   0.000   0.013   0.003   0.034   0.019   0.015   0.004   0.023
0.05746 0.06019 0.06293 0.06566 0.06840 0.07114 0.07387 0.07661 0.07934 0.08208
  0.054  -0.038  -0.019  -0.021  -0.008  -0.069   0.020   0.031  -0.003   0.028
0.08482 0.08755 0.09029 0.09302 0.09576 0.09850 0.10123 0.10397 0.10670 0.10944
 -0.066   0.022  -0.006  -0.014   0.050  -0.047  -0.055   0.011   0.001  -0.008
0.11218 0.11491 0.11765 0.12038 0.12312 0.12585 0.12859 0.13133 0.13406 0.13680
  0.014   0.022   0.003  -0.005   0.014  -0.007  -0.011   0.023  -0.023  -0.076
0.13953 0.14227 0.14501 0.14774 0.15048 0.15321 0.15595 0.15869 0.16142 0.16416
 -0.004   0.019   0.041  -0.036  -0.015  -0.040   0.026   0.055  -0.010  -0.008
```

Out[14]:




Using the auto.arima() function to fit the dataset, we set the maximum of p,q,d value=3, the function will search for a range of the parameters and get the model with the best performence.

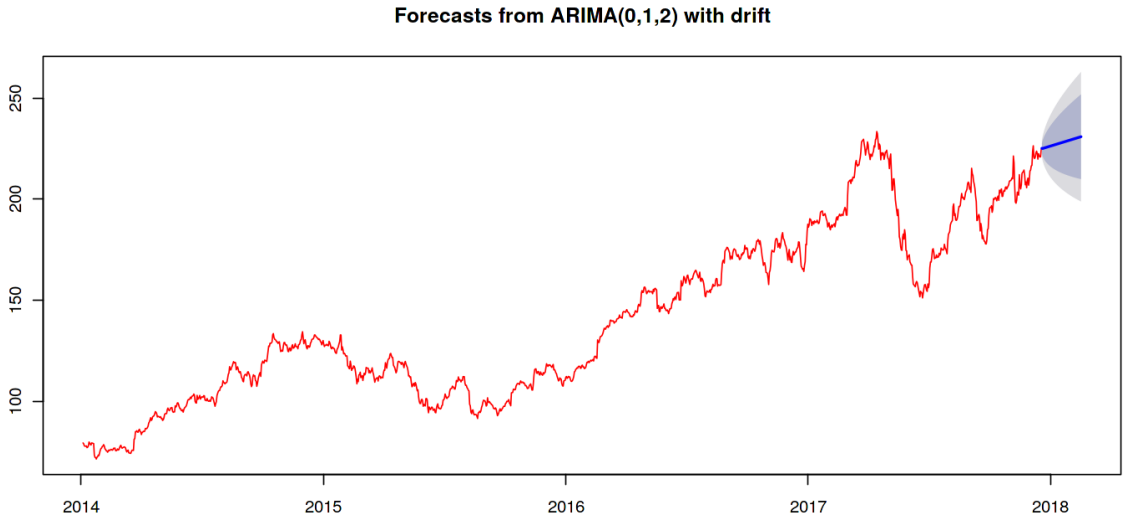
In [15]:
```
i_tsarima <- auto.arima(i_ts, max.p = 3, max.q = 3, max.d = 3)
i_tsarima
```

Out[15]:
```
Series: i_ts
ARIMA(0,1,2) with drift

Coefficients:
         ma1      ma2   drift
      0.1380  -0.0519  0.1004
s.e.  0.0263   0.0260  0.0555

sigma^2 estimated as 3.778:  log likelihood=-3009.21
AIC=6026.42   AICc=6026.45   BIC=6047.53
```

Then we can get the prediction for next 60 days, the 80% confidence intervals are encompassed in the shallow blue bands, and 95% confidence intervals are encompassed in the dark blue bands.

In [16]:
```
i_tsforecasts <- forecast(i_tsarima, h = 60)
plot(i_tsforecasts, col = "red")
```

Out[16]:




Fit ARMA(0,0)-eGARCH(1,1) model with t-distribution

In [17]:
```
garch11.spec=ugarchspec(variance.model=list(garchOrder=c(1,1)), mean.model=list(armaOr
der=c(0,0)))
#estimate model
garch11.fit=ugarchfit(spec=garch11.spec, data=i_ts)
```

Rolling 20 days of the forecast by the fitted model.

In [18]:
```
f=ugarchforecast(garch11.fit, n.ahead=20)
f
```

Out[18]:
```
*----------------------------------*
*       GARCH Model Forecast       *
*----------------------------------*
Model: sGARCH
Horizon: 20
Roll Steps: 0
Out of Sample: 0

0-roll forecast [T0=1973-12-17]:
     Series Sigma
T+1   113.9 110.6
T+2   113.9 110.6
T+3   113.9 110.5
T+4   113.9 110.5
T+5   113.9 110.4
T+6   113.9 110.4
T+7   113.9 110.3
T+8   113.9 110.3
T+9   113.9 110.2
T+10  113.9 110.2
T+11  113.9 110.1
T+12  113.9 110.1
T+13  113.9 110.0
T+14  113.9 110.0
T+15  113.9 109.9
T+16  113.9 109.9
T+17  113.9 109.8
T+18  113.9 109.7
T+19  113.9 109.7
T+20  113.9 109.6
```

### *5.4 Trade strategy analysis*

In [19]:
```
# remove all the history from the previous part
rm(list=ls())
```

In [46]:
```
# reload packages
#install.packages("quantmod")
library(quantmod)
library(blotter)
library(quantstrat)
library(TTR)
#install.packages("remotes")
#remotes::install_github("pdrano/IKTrading")
library(IKTrading)
library(dplyr)
#library(gridExtra)
#library(grid)
#library(ggplot2)
#library(lattice)
#library(ggplotify)
```

In [21]:
```
start <- as.Date("2016-01-01")
end <- as.Date("2016-10-01")
#get the data for Apple
getSymbols("AAPL", src = "yahoo", from = start, to = end, getSymbols.warning4.0=FALSE)
# Let's get data for Microsoft (MSFT) and Google (GOOG)
getSymbols(c("MSFT", "GOOG"), src = "yahoo", from = start, to = end,getSymbols.warning
4.0=FALSE)
head(AAPL)
```

Out[21]: 'AAPL'

Out[21]: 'MSFT' 'GOOG'

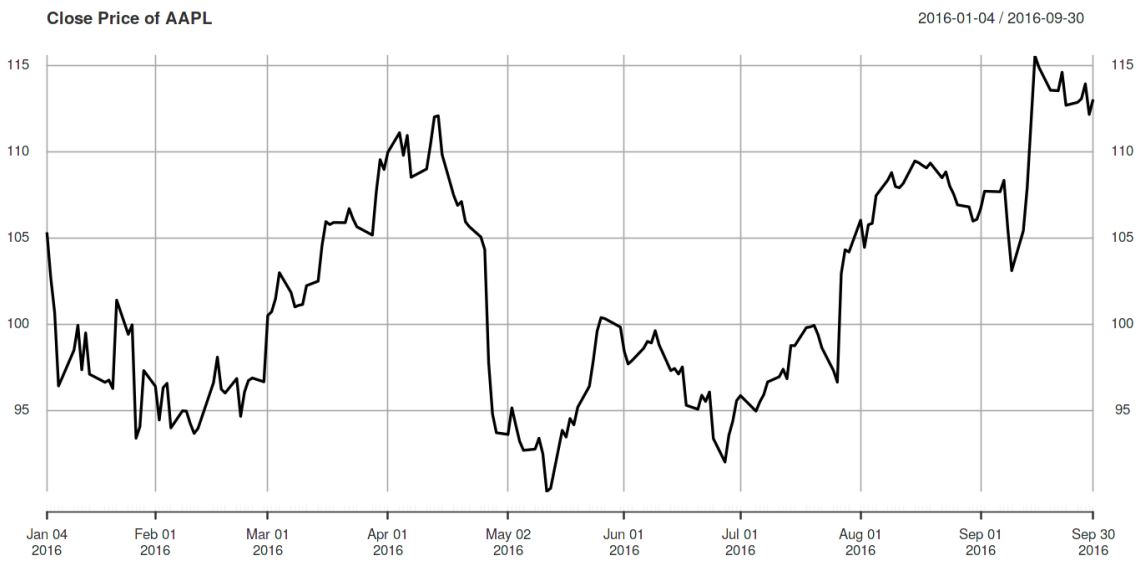
Out[21]:

| | AAPL.Open | AAPL.High | AAPL.Low | AAPL.Close | AAPL.Volume | AAPL.Adjusted |
|---|---|---|---|---|---|---|
| 2016-01-04 | 102.61 | 105.37 | 102.00 | 105.35 | 67649400 | 98.21358 |
| 2016-01-05 | 105.75 | 105.85 | 102.41 | 102.71 | 55791000 | 95.75242 |
| 2016-01-06 | 100.56 | 102.37 | 99.87 | 100.70 | 68457400 | 93.87859 |
| 2016-01-07 | 98.68 | 100.13 | 96.43 | 96.45 | 81094400 | 89.91647 |
| 2016-01-08 | 98.55 | 99.11 | 96.76 | 96.96 | 70798000 | 90.39191 |
| 2016-01-11 | 98.97 | 99.06 | 97.34 | 98.53 | 49739400 | 91.85557 |

Visualize the data of Apple

In [22]:
```
# lineplot for close price
plot(AAPL[, "AAPL.Close"], main = "Close Price of AAPL")
```

Out[22]:




In [23]:
```
#Bollinger Band chart, % Bollinger change,Volume Traded and Moving Average Convergence Diverence
chartSeries(AAPL,TA='addBBands();addVo();addMACD()')
```

Out[23]:


In [24]:
```
#candlestick plot
#candleChart(AAPL, up.col = "black", dn.col = "red", theme = "white")
```

In [25]:
```
#We not just need to look into the Apple, but also need to compare it with other stocks.
# Create an xts object (xts is loaded with quantmod) that contains closing
# prices for AAPL, MSFT, and GOOG
stocks <- as.xts(data.frame(AAPL = AAPL[, "AAPL.Close"], MSFT = MSFT[, "MSFT.Close"],
                            GOOG = GOOG[, "GOOG.Close"]))
head(stocks)
```

Out[25]:

| | AAPL.Close | MSFT.Close | GOOG.Close |
|---|---|---|---|
| 2016-01-04 | 105.35 | 54.80 | 741.84 |
| 2016-01-05 | 102.71 | 55.05 | 742.58 |
| 2016-01-06 | 100.70 | 54.05 | 743.62 |
| 2016-01-07 | 96.45 | 52.17 | 726.39 |
| 2016-01-08 | 96.96 | 52.33 | 714.47 |
| 2016-01-11 | 98.53 | 52.30 | 716.03 |

There is obvious big differences in their prices, especially between google and the other two. However, we don't care too much about the absolute price of a stock,but its changes in price or returns. Here, two simple transformations are designed to achieve that.

One transformation would be to consider the stock's return since the beginning of the period of interest with magrittr package.

$$\text{return}_{t,0} = \frac{\text{price}_t}{\text{price}_0}$$

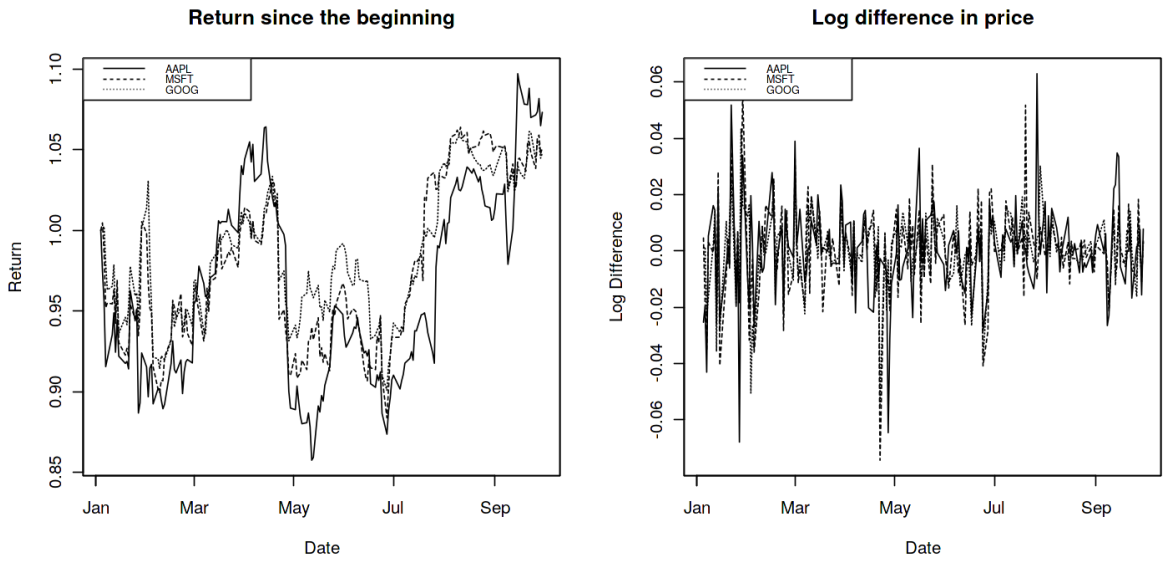
Alternatively, we could plot the change of each stock per day. In order to avoid confusion, we use log difference to model the growth of a stock.

$$\text{change}_t = \log(\text{price}_t) - \log(\text{price}_{t-1}) = \log(\frac{\text{price}_t}{\text{price}_{t-1}})$$

The advantage of using log differences is that this difference can be interpreted as the percentage change in a stock but does not depend on the denominator of a fraction. When the price in time t is greater than that in time t-1, the value of change in time t is greater than 0 and the return is positive.

In [26]:
```
par(mfrow=c(1,2))
# transformation 1: return since the beginning of period
stock_return = apply(stocks, 1, function(x) {x / stocks[1,]}) %>% t %>% as.xts
plot(as.zoo(stock_return), screens=1,lty = 1:3, xlab = "Date", ylab = "Return",main="R
eturn since the beginning")
legend("topleft", c("AAPL", "MSFT", "GOOG"), lty = 1:3, cex = 0.6)

# transformation 2 :log difference
stock_change = stocks %>% log %>% diff
plot(as.zoo(stock_change), screens=1, lty = 1:3, xlab = "Date", ylab = "Log Differenc
e", main="Log difference in price")
legend("topleft", c("AAPL", "MSFT", "GOOG"), lty = 1:3, cex = 0.6)
```

Out[26]:




From the left graph, we can now see how profitable each stock was since the beginning of the period. From the right graph, we see which periods are profitable. Furthermore, we see that these stocks generally move in the same direction, which means they are highly correlated.

From a technical analysis perspective, trends or patterns are what we want to figure out when investgating the prices of stocks.There are many indicators to do so. Here we just exam the moving average with various time windows. Traders usually define differents pricinples to utilise those found trends. Take moving average as an example, if the fast moving average(ex:20-day moving average) is above the slow moving average(ex:50-day moving average), this is a bullish market (the bulls rule), and a bearish market (the bears rule) holds when the fast moving average is below the slow moving average. Now we want to identify them in the whole time zone.

In [27]:
```
# in order to have the complete moving average in a year, we plan to extend the data
start = as.Date("2010-01-01")
getSymbols(c("AAPL", "MSFT", "GOOG"), src = "yahoo", from = start, to = end)
#candleChart(AAPL, up.col = "black", dn.col = "red", theme = "white", subset = "2016-0
1-04/")
#the 20-day, 50-day, and 200-day moving averages
#addSMA(n = c(20, 50, 200))
# using addSMA() after candleChart() will produces two charts
# to produce one chart with the two indicators, we use chartSeries()
chartSeries(AAPL, type="candlesticks",up.col = "black", dn.col = "red", theme = "whit
e", subset = "2016-01-04/",TA='addSMA(n = c(20, 50, 200));addVo()')
```

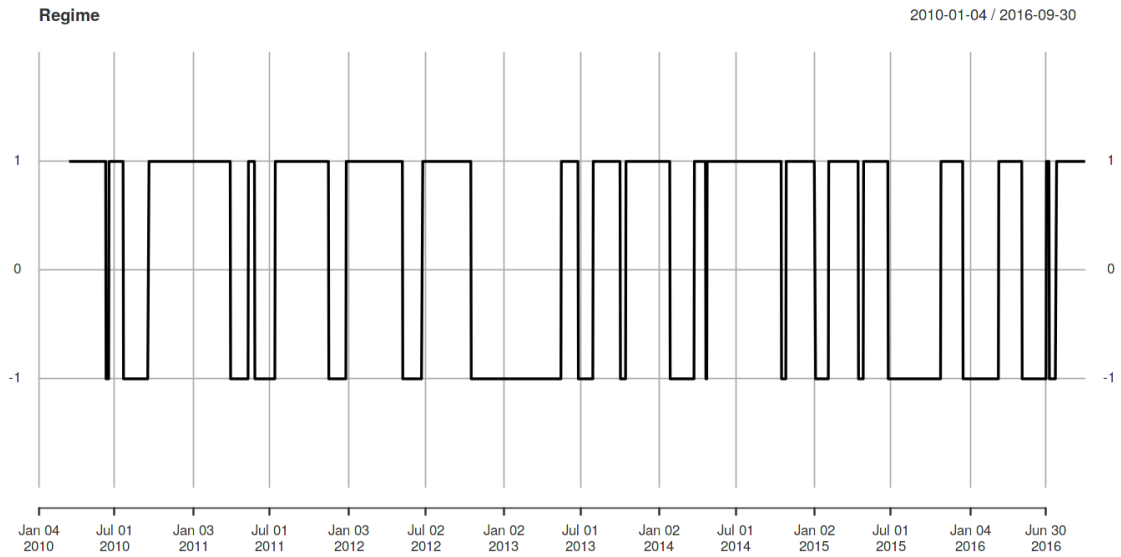
Out[27]: 'AAPL' 'MSFT' 'GOOG'

Out[27]:

In [28]:
```
# get more data
start <- as.Date("2010-01-01")
end <- as.Date("2016-10-01")
getSymbols("AAPL", src="yahoo", from = start, to = end)
AAPL_sma_20 <- SMA(Cl(AAPL),  # The closing price of AAPL, obtained by quantmod's Cl()
function
  n = 20)     # The number of days in the moving average window

AAPL_sma_50 <- SMA(Cl(AAPL),n = 50)
AAPL_sma_200 <- SMA(Cl(AAPL),n = 200)
AAPL_trade <- AAPL
AAPL_trade$`20d` <- AAPL_sma_20
AAPL_trade$`50d` <- AAPL_sma_50

regime_val <- sigComparison("", data = AAPL_trade,
                            columns = c("20d", "50d"), relationship = "gt") -
              sigComparison("", data = AAPL_trade,
                            columns = c("20d", "50d"), relationship = "lt")

plot(regime_val, main = "Regime", ylim = c(-2, 2))
```

Out[28]: 'AAPL'

Out[28]:



In [29]:
```
# put the identified regimes with the other indicators together

# again, we can use the command step by step to add each indicator
#candleChart(AAPL, up.col = "black", dn.col = "red", theme = "white", subset = "2016-0
1-04/")
#addSMA(n = c(20, 50), on = 1, col = c("red", "blue"))
#addLines(h = 0, col = "black", on = 3)
#addTA(regime_val, col = "blue", yrange = c(-2, 2))

# we also can add them together with one long command
chartSeries(AAPL, type="candlesticks",up.col = "black", dn.col = "red", theme = "whit
e", subset = "2016-01-04/",
            TA='addVo();addSMA(n = c(20, 50), on = 1, col = c("red", "blue"));
                addLines(h = 0, col = "black", on = 3);
                addTA(regime_val, col = "blue", yrange = c(-2, 2))')
```

Out[29]:


In [30]:
```
# see bullish on Apple stock for 1034 days, and bearish for 616 days
table(as.vector(regime_val))
```

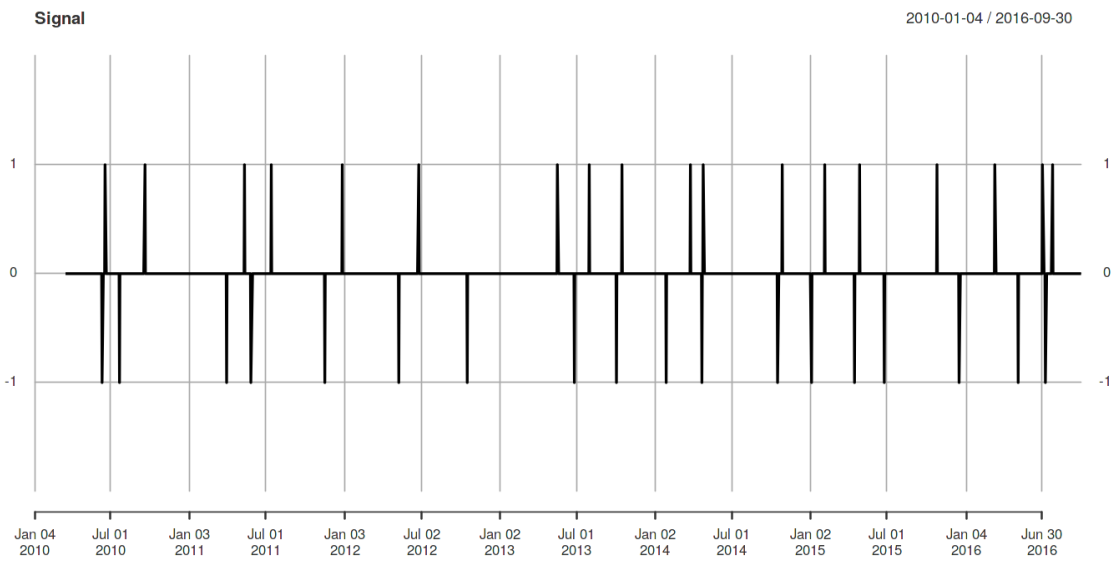
Out[30]:
```
 -1    1
616 1034
```

Trading signals appear at regime changes. For example,when a bullish regime begins, a buy signal is triggered, and when it ends, a sell signal is triggered. Let $r_t$ indicate the regime at time t, and $s_t$ the signal at time t. Then: $s_t = \text{sign}(r_t - r_{t-1})$

$s_t \in \{-1, 0, 1\}$, with -1 indicating “sell”, 1 indicating “buy”, and 0 no action.

In [31]:
```
sig <- diff(regime_val) / 2
plot(sig, main = "Signal", ylim = c(-2, 2))
```

Out[31]:




In [32]:
```
table(sig)
```

Out[32]:
```
sig
  -1    0    1
  18 1613   18
```

We would buy Apple stock 18 times and sell Apple stock 18 times. If we only go long on Apple stock, only 18 trades will be engaged in over the 6-year period, while if we pivot from a long to a short position every time a long position is terminated, we would engage in 18 trades total. Additionally, a long position is one in which a profit is made if the stock increases in value, and a short position is on in which a profit is made if stock decreases in value. When trading stocks directly, all long positions are bullish and all short position are bearish.

Now it is time to identify what the prices of the stock is at every buy and every sell. We would take the closing price as buying price or selling price when the trade signals is activated.

In [33]:
```
## The Cl function from quantmod pulls the closing price from the object
Cl(AAPL)[which(sig == 1)][1:5,]#1 indicating “buy”
```

Out[33]:
```
           AAPL.Close
2010-06-18   39.15286
2010-09-20   40.46143
2011-05-12   49.51000
2011-07-14   51.11000
2011-12-28   57.52000
```

In [34]:
```
Cl(AAPL)[which(sig == -1)][1:5,] #-1 indicating “sell”
```

Out[34]:
```
           AAPL.Close
2010-06-11   36.21571
2010-07-22   37.00286
2011-03-31   49.78714
2011-05-27   48.20143
2011-11-17   53.91571
```

In [35]:
```
# the profit is calculate as
as.vector(Cl(AAPL)[sig == 1])[-1] - Cl(AAPL)[sig == -1][-table(sig)[["1"]]]
```

Out[35]:
```
           AAPL.Close
2010-06-11   4.245716
2010-07-22  12.507141
2011-03-31   1.322857
2011-05-27   9.318573
2011-11-17  27.622860
2012-05-09 -19.417145
2012-10-17 -27.440002
2013-06-26  14.720001
2013-10-03   8.052856
2014-01-28   9.348571
2014-04-22  30.782852
2014-10-17  22.270004
2015-01-05  24.309998
2015-04-16 -11.619995
2015-06-25 -25.239998
2015-12-18 -10.140000
2016-05-05   4.099998
```

Let's now create a simulated portfolio of $1,000,000, and see how it would behave, according to the rules we have established. This includes:

Investing only 10% of the portfolio in any trade

Exiting the position if losses exceed 20% of the value of the trade.

When simulating, we are going to not force orders to be in batches of 100 shares, nor will we enforce the stop-loss rule and the commission to the broker.

In [36]:
```
rm(list = ls(.blotter), envir = .blotter)  # Clear blotter environment
currency("USD")  # Currency being used
Sys.setenv(TZ = "MDT")  # Allows quantstrat to use timestamps
AAPL_adj <- adjustOHLC(AAPL,use.Adjusted = TRUE)
stock("AAPL_adj", currency = "USD", multiplier = 1)
initDate <- "1990-01-01"  # A date prior to first close price, used in function initAc
ct()
```

Out[36]: 'USD'

Out[36]: 'AAPL_adj'

Now we create a portfolio and account object that will be used in the simulation.

In [37]:
```
strategy_st <- portfolio_st <- account_st <- "SMAC-20-50"  # Names of objects
rm.strat(portfolio_st)  # Need to remove portfolio from blotter env
rm.strat(strategy_st)   # Ensure no strategy by this name exists either
initPortf(portfolio_st, symbols = "AAPL_adj",  # This is a simple portfolio
                                               # trading AAPL only
          initDate = initDate, currency = "USD")
```

Out[37]: 'SMAC-20-50'

In [38]:
```
initAcct(account_st, portfolios = portfolio_st,  # Uses only one portfolio
         initDate = initDate, currency = "USD",
         initEq = 1000000)  # Start with a million dollars
initOrders(portfolio_st, store = TRUE)  # Initialize the order container; will
                                        # contain all orders made by strategy
```

Out[38]: 'SMAC-20-50'

In [39]:
```
# define the strategy.
strategy(strategy_st, store = TRUE)  # store = TRUE tells function to store in
                                     # the .strategy environment

# Now define trading rules
# Indicators are used to construct signals
add.indicator(strategy = strategy_st, name = "SMA",      # SMA is a function
              arguments = list(x = quote(Cl(mktdata)),  # args of SMA
                               n = 20),
              label = "fastMA")
add.indicator(strategy = strategy_st, name = "SMA",
              arguments = list(x = quote(Cl(mktdata)),
                               n = 50),
              label = "slowMA")
```

Out[39]: 'SMAC-20-50'

Out[39]: 'SMAC-20-50'

In [40]:
```
# Next comes trading signals
add.signal(strategy = strategy_st, name = "sigComparison",
           arguments = list(columns = c("fastMA", "slowMA"),
                            relationship = "gt"),
           label = "bull")
add.signal(strategy = strategy_st, name = "sigComparison",
           arguments = list(columns = c("fastMA", "slowMA"),
                            relationship = "lt"),
           label = "bear")
```

Out[40]: 'SMAC-20-50'

Out[40]: 'SMAC-20-50'

In [41]:
```
#rules that generate trades of buy or sell
add.rule(strategy = strategy_st, name = "ruleSignal",  # Almost always this one
         arguments = list(sigcol = "bull",  # Signal (see above) that triggers
                          sigval = TRUE,
                          ordertype = "market",
                          orderside = "long",
                          replace = FALSE,
                          prefer = "Open",
                          osFUN = osMaxDollar,
                          # The next parameter, which is a parameter passed to
                          # osMaxDollar, will ensure that trades are about 10%
                          # of portfolio equity
                          maxSize = quote(floor(getEndEq(account_st,
                                                         Date = timestamp) * .1)),
                          tradeSize = quote(floor(getEndEq(account_st,
                                                           Date = timestamp) * .1))),
         type = "enter", path.dep = TRUE, label = "buy")

add.rule(strategy = strategy_st, name = "ruleSignal",
         arguments = list(sigcol = "bear",
                          sigval = TRUE,
                          orderqty = "all",
                          ordertype = "market",
                          orderside = "long",
                          replace = FALSE,
                          prefer = "Open"),
         type = "exit", path.dep = TRUE, label = "sell")
```

Out[41]: 'SMAC-20-50'

Out[41]: 'SMAC-20-50'

In [0]:
```
# apply trade strategy now
applyStrategy(strategy_st, portfolios = portfolio_st)
```

In [43]:
```
# analyze trade strategy result
updatePortf(portfolio_st)
dateRange <- time(getPortfolio(portfolio_st)$summary)[-1]
updateAcct(portfolio_st, dateRange)
updateEndEq(account_st)
tStats <- tradeStats(Portfolios = portfolio_st, use="trades",
                     inclZeroDays = FALSE)
tStats[, 4:ncol(tStats)] <- round(tStats[, 4:ncol(tStats)], 2)
print(data.frame(t(tStats[, -c(1,2)])))
```

Out[43]: 'SMAC-20-50'

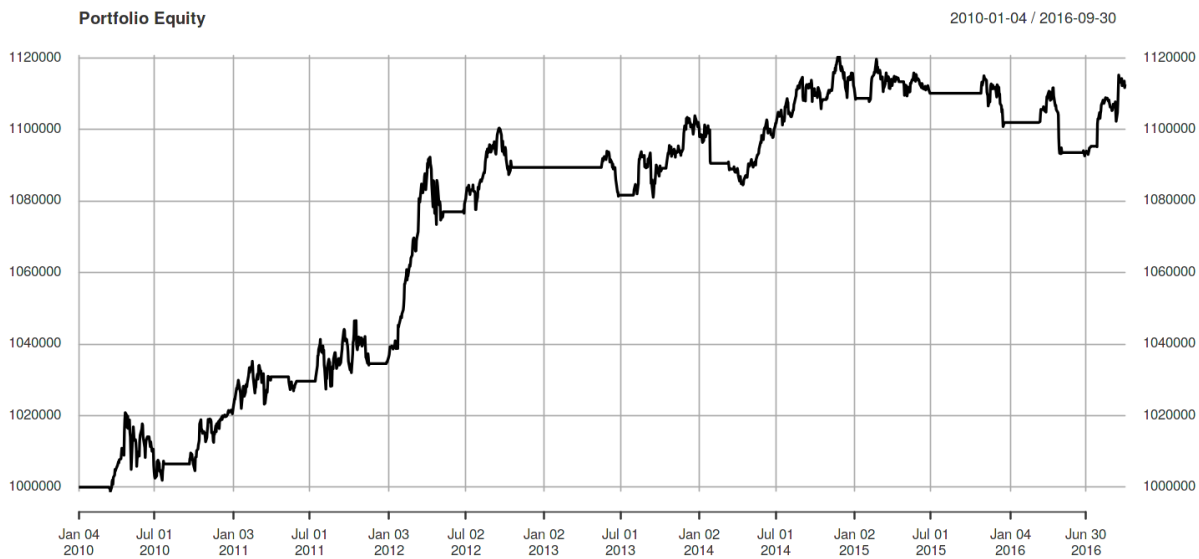
Out[43]: 'SMAC-20-50'

Out[43]: 'SMAC-20-50'

| | AAPL_adj |
|---|---|
| Num.Txns | 88.00 |
| Num.Trades | 21.00 |
| Net.Trading.PL | 112580.16 |
| Avg.Trade.PL | 5360.96 |
| Med.Trade.PL | 1380.43 |
| Largest.Winner | 42433.15 |
| Largest.Loser | -8389.52 |
| Gross.Profits | 152877.05 |
| Gross.Losses | -40296.89 |
| Std.Dev.Trade.PL | 12836.74 |
| Std.Err.Trade.PL | 2801.21 |
| Percent.Positive | 57.14 |
| Percent.Negative | 42.86 |
| Profit.Factor | 3.79 |
| Avg.Win.Trade | 12739.75 |
| Med.Win.Trade | 9969.59 |
| Avg.Losing.Trade | -4477.43 |
| Med.Losing.Trade | -3231.50 |
| Avg.Daily.PL | 4765.04 |
| Med.Daily.PL | 857.11 |
| Std.Dev.Daily.PL | 12868.75 |
| Std.Err.Daily.PL | 2877.54 |
| Ann.Sharpe | 5.88 |
| Max.Drawdown | -27949.54 |
| Profit.To.Max.Draw | 4.03 |
| Avg.WinLoss.Ratio | 2.85 |
| Med.WinLoss.Ratio | 3.09 |
| Max.Equity | 120547.45 |
| Min.Equity | -1182.33 |
| End.Equity | 112580.16 |

In [44]:
```
final_acct <- getAccount(account_st)
plot(final_acct$summary$End.Eq["2010/2016"], main = "Portfolio Equity")
```

Out[44]:



Our portfolio's value grew by 10% in about six years.

**6 Reference**


[1]S. P. Chatzis, V. Siakoulis, A. Petropoulos, E. Stavroulakis, and N. Vlachogiannakis, "Forecasting stock market crisis events using deep and statistical machine learning techniques," Expert Systems with Applications, vol. 112, pp. 353–371, 2018, doi: https://doi.org/10.1016/j.eswa.2018.06.032 (https://doi.org/10.1016/j.eswa.2018.06.032).

[2]L. Khaidem, S. Saha, and S. R. Dey, "Predicting the direction of stock market prices using random forest," arXiv:1605.00003 [cs], Apr. 2016, Accessed: 27-Mar-2020. [Online]. Available: http://arxiv.org/abs/1605.00003 (http://arxiv.org/abs/1605.00003).

[3]D. M. Q. Nelson, A. C. M. Pereira, and R. A. de Oliveira, "Stock market's price movement prediction with LSTM neural networks," in 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 1419–1426, doi: 10.1109/IJCNN.2017.7966019.

[4]Wing-Keung Wong , Meher Manzur & Boon-Kiat Chew, How rewarding is technical analysis? Evidence from Singapore stock market, Applied Financial Economics, 2003,13:7, 543-551, DOI: 10.1080/0960310022000020906

[5] Massoud Metghalchi , Juri Marcucci & Yung-Ho Chang, Are moving average trading rules profitable? Evidence from the European stock markets, Applied Economics, 2012, 44:12, 1539-1559, DOI: 10.1080/00036846.2010.543084

[6] R. M. Kapila Tharanga Rathnayaka, D. M. K. N. Seneviratna, W. Jianguo and H. I. Arumawadu, "A hybrid statistical approach for stock market forecasting based on Artificial Neural Network and ARIMA time series models," 2015 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC), Nanjing, 2015, pp. 54-60.

[7] Al-Najjar, D. (2016). Modelling and Estimation of Volatility Using ARCH/GARCH Models in Jordan's Stock Market.

[8]An Introduction to Stock Market Data Analysis with R(2017):https://ntguardian.wordpress.com/2017/03/27/introduction-stock-market-data-r-1/,https://ntguardian.wordpress.com/2017/04/03/introduction-stock-market-data-r-2/ (https://ntguardian.wordpress.com/2017/03/27/introduction-stock-market-data-r-1/,https://ntguardian.wordpress.com/2017/04/03/introduction-stock-market-data-r-2/)

[9]Using BatchGetSymbols to download financial data for several tickers https://cran.r-project.org/web/packages/BatchGetSymbols/vignettes/BatchGetSymbols-vignette.html (https://cran.r-project.org/web/packages/BatchGetSymbols/vignettes/BatchGetSymbols-vignette.html)

[10]Quandl R Package https://www.quandl.com/tools/r (https://www.quandl.com/tools/r)

[11]Interactive Brokers With R: Ibrokers Trading API https://blog.quantinsti.com/using-ibrokers-package-implement-r-interactive-brokers-api/ (https://blog.quantinsti.com/using-ibrokers-package-implement-r-interactive-brokers-api/)

[12]Financial Data Accessible from R – part IV https://www.thertrader.com/2013/12/13/financial-data-accessible-from-r-part-iv/ (https://www.thertrader.com/2013/12/13/financial-data-accessible-from-r-part-iv/)

[13]Jesse Piburn,Introduction to the wbstats R-package,2018,https://cran.r-project.org/web/packages/wbstats/vignettes/Using_the_wbstats_package.html#fn1 (https://cran.r-project.org/web/packages/wbstats/vignettes/Using_the_wbstats_package.html#fn1)

[14]Ko Chiu Yu,Techincal Analysis with R,2019,https://bookdown.org/kochiuyu/Technical-Analysis-with-R/ (https://bookdown.org/kochiuyu/Technical-Analysis-with-R/)

[15]Ko Chiu Yu,Applied Financial Economics -- Programming: Excel, VBA and R,2019