Introduction to Financial Data Analysis

Week1-2: FINANCIAL DATA AND THEIR PROPERTIES

Course Structure

- This course aims at providing basic knowledge of financial data analysis, introducing the useful statistical tools for analyzing financial data—R programming language, and gaining experience in financial applications.

Part I: Basics in Programming in R and Financial Asset Properties

- 1. Basic programming knowledge in understanding stocks and portfolio performance by visualizing financial data
- Two keys for investing: Return and Risk, trade-off
- Return:
  - Single asset returns, multiple asset returns
  - Factor method in evaluate whether your investment beats the market. CAPM and Fama-French model
- Risk:
  - Variance, VaR Method, Expected Shortfall
- Balance between risk and return:
  - Mean-variance efficient portfolio optimization.

Part II: Understand Financial Data Time Series Properties in Depth

- 1. Financial Data are usually time series, eg. Price, volume, returns all have a timestamp.
- 2. Predict time series by understand the classic properties of timeseries. Random walk, trend, AR, etc.
- 3. Statistical methods to judge whether the financial data are predictable?
- 3. High frequency data properties

Part III: Cross-sectional Portfolio Construction

- 1. How to choose the right targets of investing in the huge universe of financial assets?
- Cross-sectional analysis rather than timeseries analysis.
- Timeseries analysis aims to predict single asset’ s future returns/price/volatilities.
- Cross-sectional analysis aims to select out the financial assets that perform better than other financial assets.
- A case study of size factor investing.

Introduction to R Programming

What is R?

- R is a programming language specially designed to resolve statistical problems and allow the graphical display of data.
- R is rooted from S, a programming language originally created in Bell Laboratories (formerly AT&T, now Lucent Technologies). The base code of R was developed by two academics, Ross Ihaka and Robert Gentleman, resulting in the programming platform we have today. For anyone curious about the name, the letter R was chosen due to the common first letter of the name of their creators.
- In the business side, large and established companies, such as Google and Microsoft, already adopted R as the internal language for data analysis.

Why R Programming?

- 1. R is available for most operating systems.
- 2. R is a mature, stable platform, continuously supported and intensively used in the industry.
- 3. Many researchers have developed nice packages for analyzing financial data. E.g. tidyr, dplyr, TTR, timeseries, quantmod, etc.
- 4. Learning R is easy. Friendly to users without professional programming background
- 5. Powerful statistical analysis compared to other programming language, eg. Python
- 6. The interface of R called RStudio is very friendly and productive.

Why R Programming?

- 7. The graphical interface provided by RStudio facilitates the use of R and increases productivity. E.g. Base R plot, ggplot, ploty, shiny,
  - https://plot.ly/r/ohlc-charts/,
  - https://shiny.rstudio.com/gallery/see-more.html
- 8. R is compatible with different operating systems and it can interface with different programming languages. If you need to use a code in other programming language, such as C++, Python, Latex, it is easy to integrate it with R. Therefore, the user is not restricted to a single language and can use features and functions from other platforms.
- 9. R is free!!!!

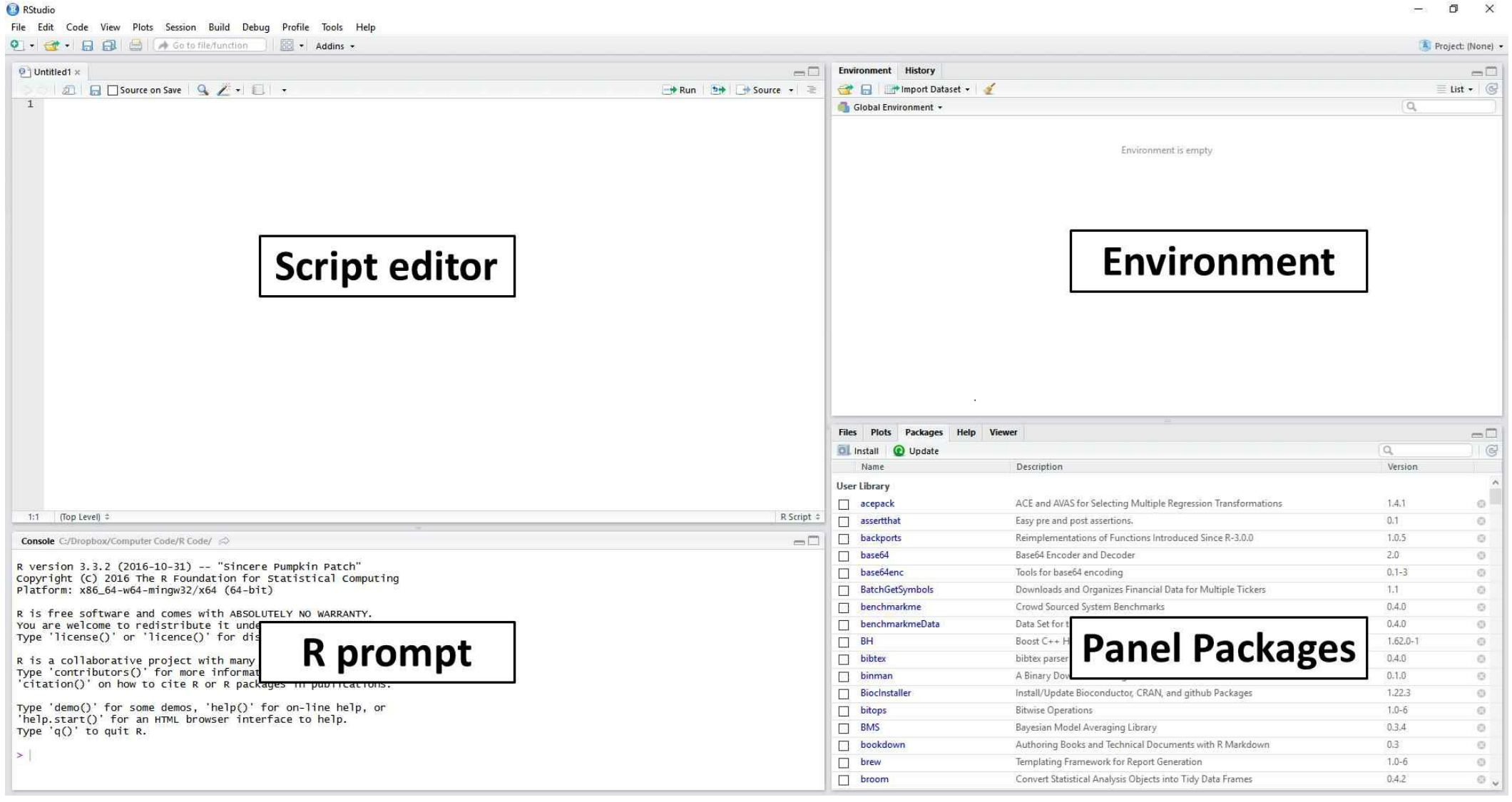
What Can You Do With R and RStudio?

- Import, export, process, and store financial data based on local files or the internet
- Substitute and improve data intensive tasks from spreadsheet like software
- Develop routines for managing and controlling investment portfolios and executing financial orders
- Implementation of various possibilities of empirical research through statistical tools, such as econometric models and hypothesis testing
- Create dynamic websites with the Shiny package, allowing anyone in the world to use a financial tool created by you
- Create an automated process of developing technical financial reports with package knitr
- Write a technical book with bookdown
- Write and publish a blog about finance with blogdown

R Resources

- The CRAN website (https://cran.r-project.org/web/views/Finance.html) offers a *Task Views* panel for the topic of Finance. On this page, you can find the main packages available to perform specific operations in Finance. This includes importing financial data from the internet, estimating econometric model, calculation of different risk estimates, among many other possibilities. Reading this page and the knowledge of these packages is essential for those who intend to work in Finance. It is worth noting, however, this list contains only the main items. The complete list of packages related to Finance is much larger than shown in *Task Views*.
- R-Bloggers (https://www.r-bloggers.com/) is a website that aggregates these blogs in a single place, making it easier for anyone to access and participate..
- Stack Overflow (https://stackoverflow.com/) is a question and answer site for professional and enthusiast programmers.

Explaining the RStudio Screen



Explaining the RStudio Screen

```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

Panels of RStudio

- **Script Editor:** located on the left side and above the screen. This panel is used to write scripts and functions;
- **R prompt:** located on the left side and below the script editor. It displays the prompt of R, which can also be used to give commands to R. The main function of the prompt is to test code and display the results of the commands entered in the script editor;
- **Environment:** located on the top-right of the screen.
  - Shows all objects, including variables and functions currently available to the user. Also note a History panel, which shows the history of the commands previously executed by the user;
- **Panel Packages:** shows the packages installed and loaded by R. Four tabs:
  - Files, to load and view system files;
  - Plots, to view pictures;
  - Help to access the help system;
  - Viewer to display dynamic and interactive results, such as a web page.
- You can customize your R studio by using Tools->Global Options

Write Code in Prompt

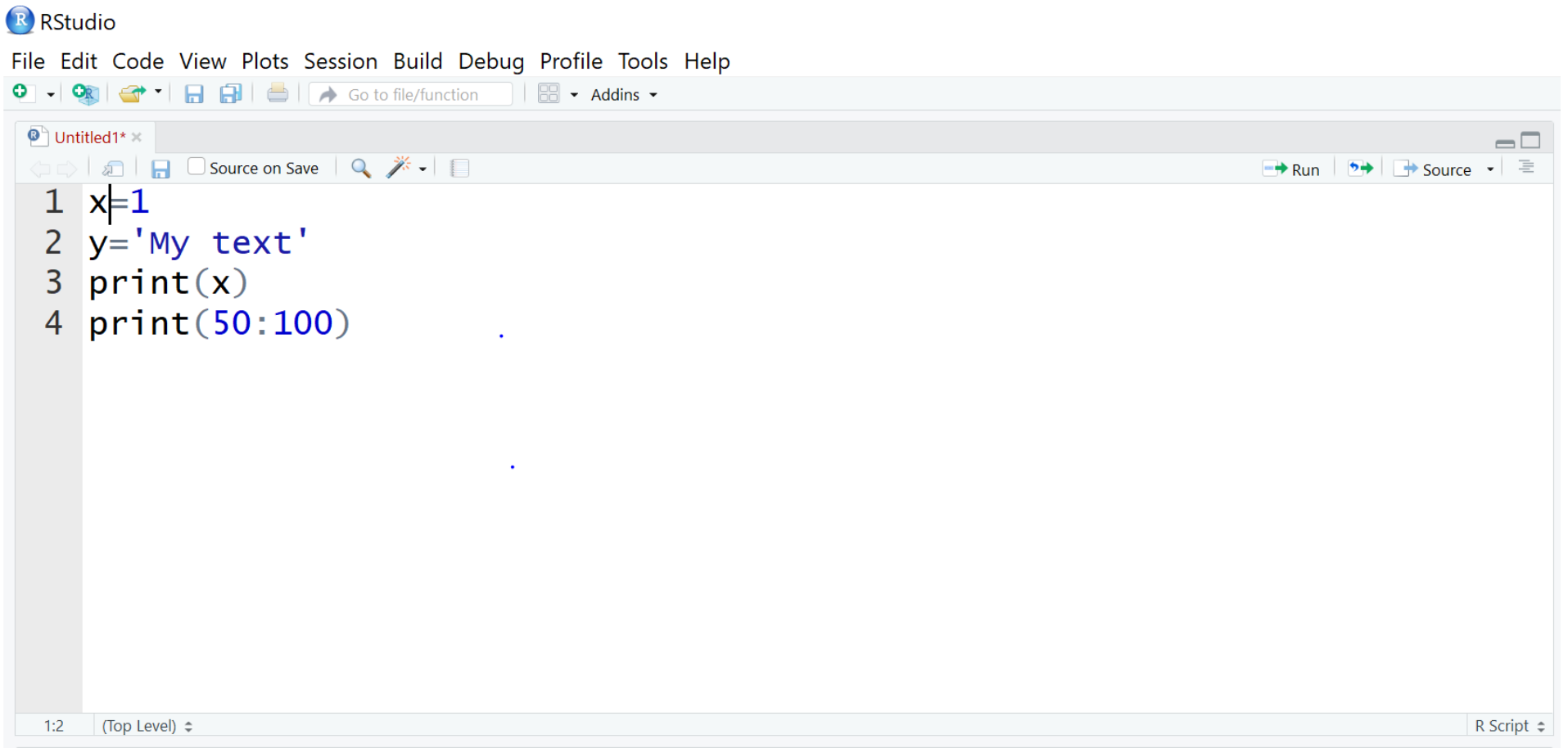
```
Console ~/trade/
> # set x
> x <- 1
> # set y
> y <- 'My text'
> # print contents of x
> print(x)
[1] 1
> ## [1] 1
> # print a sequence
> print(50:100)
 [1]  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65
[17]  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
[33]  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
[49]  98  99 100
>
```

- The use of double quotes (" ") or single quotes (' ') defines objects of the class character.
- Numbers are defined by the value itself.
- Each object in R has a class and each class has a different behavior.
- After sending the previous commands to R, the history tab has been updated.
- The print function is one of the main functions for displaying values in the prompt of R. The text displayed as [1] indicates the index of the first line number.
- # marks comments in R

Write your first R script

- As a first exercise, click *file*, *New File*, and *R Script*.
- A text editor should appear on the left side of the screen.
- It is there that we will enter our commands, which are executed from top to bottom, in the same direction that we normally read text.
- A side note, all *.R* files created in RStudio are just text files and can be edited in other editors as well. It is not uncommon for experienced programmers to use a specific software to write code and another to run it.

Write Your Code in Script



Save Code

- Save the .R file to a personal folder where you have read and write permissions. One possibility is to save it in the My Documents folder with a name like 'MyFirstRScript.R' .
- Shortcuts for executing codes:

1. control + shift + s: executes (source) the current RStudio file;
2. control + shift + enter: executes the current file with echo, showing the commands on the prompt;
3. control + enter: executes the selected line, showing on-screen commands;
4. control + shift + b: executes the codes from the beginning of the file to the current line where the cursor is;
5. control + shift + e: executes the codes of the lines where the cursor is until the end of the file.

Get Financial Asset Data

- WRDS (Wharton Research Data Services)
- https://wrds-web.wharton.upenn.edu/wrds/
- WRDS provides a direct interface for R, enabling you to query WRDS data from within your R program. WRDS data is stored in a PostgreSQL database and accessed using an R Postgres driver.
- Once you have everything set up (as covered in the below sections), querying WRDS data within your code is as simple as the following example, which queries the Dow Jones Averages & Total Return Indexes using the R Postgres driver:

```
res = dbSendQuery(wrds, "select date,dji from djones.djdaily")
data = dbFetch(res, n=-1)
dbClearResult(res)
```

Example: Get Close Level of Dow Jones Index

- There are huge amount of data within the WRDS database. The WRDS data are stored in SQL database format.

- SQL (Structured Query Language) is a programming language used in programming and designed for managing data held in a relational database management system. It is particularly useful in handling structured data where there are relations between different entities/variables of the data.

Example: Get one single stock’ s High Low Open Close Price: AAPL

- CRSP database in WRDS: The CRSP U.S. Stock database contains end-of-day and month-end prices on primary listings for the NYSE, NYSE MKT, NASDAQ, and Arca exchanges, along with basic market indices.
- Coverage of CRSP: All securities listed in CRSP U.S. Stock databases are equity securities, not bonds.
- CRSP U.S. Stock database contains the following information:
  - ** Price and quote data (e.g. Open, close, bid/low, ask/high, trade-only).
  - ** Holding period returns with and without dividends.
  - ** Excess returns and other derived data items.
  - ** Market capitalization.
  - ** Shares outstanding.
  - ** Trading volume.
  - ** Security delisting information.
  - ** Corporate actions.
  - ** Identifiers, descriptors, and supplemental data items.
- PERMNO represents permanent number. Stocks can reuse cusip, ticker, sedol when they are merged/delisted/change name
- From ticker to the unique identifier(permno) in CRSP CRSP has a table called CRSP.DSENAMES, linking common stock identifiers: such as cusip, ticker, sedol, name to a unique id CRSP use to identify stocks.

Example: Get one single stock’ s High Low Open Close Price: AAPL

- SQL quote
- “**select** field(s) **from** datatable **where** condition1 **and** condition2 ”
- fields are separated by “,”
- Example of conditions:
  - date>=’ 2015-01-01’
  - permno= ‘14593’
  - ticker in (‘AAPL’ ,’ MSFT’)
- the filter condition’ s fields like date and permno must exist in the datatable you’ re selecting from

Quote from WRDS

- q = “select fields(s) from datatable where condition1 and condition 2”
- res = dbSendQuery(wrds, q)
- myvar = dbFetch(res, n=-1)
- dbClearResult(res)

- q is simply a long character

Write your quote flexible using paste0()

- paste0() function in R allows you to concatenate characters together
- Example: paste0(“text1” ,” text2”) gives you “text1text2”
- text1= “test2”
- test2= “test1”
- What is the print of paste0(text1,text2, “text2”)?
- Create a long quote using paste0()
- text 1 = “select fields(s) from database where”
- text 2 = “ticker = ‘AAPL’ ”
- paste0(text1,text2)

A Case Study of Long-Short Trading Using MSFT and AAPL

- An investor believe MSFT is a better stock than AAPL
  - She could either long MSFT, or short AAPL or do both at the same time.
  - The benefit of long-short strategy is that she could achieve market neutral returns.
- Market Neutral: A market-neutral strategy is a type of investment strategy undertaken by an investor or an investment manager that seeks to profit from both increasing and decreasing prices in one or more markets, while attempting to completely avoid some specific form of market risk.
  - AAPL and MSFT are both highly correlated with the entire market.
  - When market is bearish, MSFT as a better stock, drop less in value compared to AAPL, making the investor a profit.
  - When market is bullish, MSFT as a better stock, gets higher returns compared to AAPL, making the investor a profit.

A Case Study of Relative Performance of IBM and MSFT

- Step 1. Grab IBM and MSFT price data from WRDS CRSP database
- Step 2: check data quality
- Step 3: calculate returns of two stocks
- Step 4: calculate the relative performance of two stocks

Introduction to R Programming with a Case Study of MSFT and AAPL

- Refer to
- “L1_A_CASE_STUDY_MSFT_AAPL.html”

Introduction to R Programming

- Refer to “L1_R_Programming.html”

A Case Study of Technical Analysis

- Technical analysis is the use of charts to study stock price and volume data for the purpose of forecasting future trends.
- Those who follow technical analysis are using stock price and volume data as an indication of the supply and demand for the stock.
  - For example, a rising stock price may indicate that demand exceeds supply while a falling stock price may indicate that supply exceeds demand.
  - As a trading strategy, profiting from technical analysis relies on being able to identify trends and the ability to catch on during the early stages of the trend.
  - Moreover, the same technical indicators may be interpreted differently by different chartists depending on their investment philosophies (e.g., trend follower or contrarian).

A Case Study of Technical Analysis

- There are three broad groups of technical indicators:
- (i) trend indicator,
- (ii) volatility indicator
- (iii) momentum indicator.
- In this chapter, we go through an example each type starting with simple moving average crossover (trend), Bollinger Bands (volatility), and relative strength index (momentum).
- In constructing trading strategies, combining indicators is often used
  - For example, we can use the relative strength index to confirm signals identified by the simple moving average. Therefore,

Trend: Simple Moving Average Crossover

- A common technical analysis trend indicator is the Simple Moving Average (SMA) crossover.
- The moving average is calculated by taking the average of a firm’ s stock price over a certain number of days.
- The term “simple” comes from the fact that this type of average treats all days equally, regardless of how near or far from those days are from the present.
- SMA “crossover” uses two SMA lines, a shorter-term and a longer-term, and make trading decisions when the lines cross.

Example of AAPL

- Classic SMA use an average over 50 (200) days for the shorter (longer) term.
- Implement an SMA crossover for AAPL stock from 2016-2019.
- If the 50-day moving average cross above the 200-day moving average, which is called a bullish crossover, this may be taken as an indicator to buy the stock.
- Conversely, if the 50-day moving average crosses below the 200-day moving average, which is known as a bearish crossover, this may be taken as an indication to sell the stock.

Steps of SMA

- Step 1: Obtain Closing Prices for AAPL
- Step 2: Calculate the Rolling 50-Day and 200-Day Average Price
- Step 3: Subset to Only Show 2018 and 2019 Data
- Step 4: Plot the SMA

Volatility: Bollinger Bands

- The Bollinger Bands have three components:
  - 1. The first component is a 20-day simple moving average (SMA).
  - 2. The second component is an upper band, which is two standard deviations above the 20-day SMA.
  - 3. The third component is a lower band, which is two standard deviations below the 20-day SMA.
- Bollinger Bands are considered as volatility indicators because the Bollinger Bands widen (narrow) with more (less) volatility in the stock.
- When the bands narrow, it may be used as an indication that volatility is about to rise.

Steps of Bollinger Bands

- Step 1: Obtain Closing Prices of AAPL
- Step 2: Calculate Rolling 20-Day Mean and Standard Deviation
  - We use the rollmean command with k=20 to calculate the 20-day moving average.
  - For the standard deviation, we use the rollapply command, which allows us to apply a function on a rolling basis.
  - The FUN=sd tells R that the function we want to apply is the standard deviation and the width=20 tells R that it is a 20-day standard deviation that we want to calculate.
- Step 3: Subset to 2016
- Step 4: Calculate the Bollinger Bands
  - The bands that are two standard deviations around the average. Using January 4, 2016 as an example, this means that the upper Bollinger Band is equal to 119.8524 (=110.726+2*4.563188), and the lower Bollinger Band is equal to 101.5996 (= 110.726-2*4.563188)
- Step 5: Plot the Bollinger Bands

Interpreting the Bollinger Bands

- Assuming a normal distribution, two standard deviations in either direction from the mean should cover pretty much the majority of the data.
- Most of the closing prices fell within the Bollinger Bands.
- For a trend follower, when the stock price was right around the upper band, this may be taken as an indication that the stock is *overbought*.
- Conversely, when the stock price moved right around the lower band, this may be taken as an indication that the stock is *oversold*.

Momentum: Relative Strength Index

- A common technical analysis momentum indicator is the Relative Strength Index (RSI).
- The typical calculation is to use a 14-day period.
- The RSI is calculated as

$$RSI = 100 - \frac{100}{1 + RS},$$

- where *RS* is equal to the up average divided by the down average with the averages calculated using the Wilder Exponential Moving Average described below.

Momentum: RSI

- The RSI is used in conjunction with an overbought line and an oversold line. The overbought line is typically set at a level of 70 and the oversold line is typically set at a level of 30.
- A buy signal is created when the RSI rises from below the oversold line and crosses the oversold line.
- Conversely, a sell signal is created when the RSI falls from above the overbought line and crosses the overbought line.

Steps of Constructing RSI

- Step 1: Obtain Closing Prices
  - We also calculate the difference in AAPL’ s price using the lag command. The difference between the closing price today and yesterday’ s closing price is reported in the column labeled delta.
- Step 2: Create Dummy Variables to Indicate Whether Price Went Up or Price Went Down
  - We construct the up variable, which takes on a value of 1 if the price of AAPL went up and zero otherwise.
  - We also construct the down variable, which takes on a value of 1 if the price of AAPL went down and zero otherwise.
  - To create these dummy variables we use the ifelse command.

Steps of Constructing RSI

- Step 3: Calculate Prices for Up Days and Prices for Down Days
  - To construct a series of prices on up days, we multiply close with up.
  - If it is an up day, up will equal one, so up.val will equal the closing price. On a down day, up will equal zero, so up.val will equal zero. The down day prices are calculated in a similar way.
- Step 4: Calculate Initial Up and Down 14-Day Averages
  - We use the command rollapply on the average function FUN=mean.
  - This is similar to using the rollmean command above, but I just thought of switching it up for a little variety.

Steps of Constructing RSI

- Step 5: Calculate the Wilder Exponential Moving Average to Calculate Final Up and Down 14-Day Averages
  - To make the calculation easier, we extract the up.val and down.val columns in aapl.rsi into separate independent vectors.
  - We then need to calculate the Wilder Exponential Moving Average for the up and down values.
  - This average calculation assumes that the initial average the day before would have a weight of 13 out of 14 days and the current average will have a weight of one out of 14 days.
  - For t>=15: WEMV(t) = WEMV(t-1) * 13/14 + Value(t)* 1/14
  - For t<15, WEMV(t) = SMA(t)
  - We use for loop to calculate this WEMA
  - We apply this same logic for the up average and down average.
  - The calculated values under the Wilder Exponential Moving Average are reported under the up.avg and down.avg columns

Steps of Constructing RSI

- Step 6: Calculate the RSI using the formula as before.
- Step 7: Subset to Show 2016 Data
- Step 8: Plot the RSI

Homework template

- Refer to “L1_HW_RMarkdown_Template.html”
- Knit your work with command line code
- rmarkdown::render(“filename.Rmd”)