Applied R Programming 2021

-- Nikhil Vidhani
PhD Student,
Finance and Accounting

Course Objective

Course Objective

- At the end of 10 weeks:
  - You should be able to perform basic programming tasks in R
    - Irrespective whether they are related to data analysis or your work
    - Appreciate programming as a means to accomplish huge no. of smaller tasks
    - Build complex logic and work-flow through small and concise functions
  - Have a good understanding of how to approach an empirical project
    - Data sources, merging, cleaning etc
  - Perform data analysis
    - Summaries, plots, regressions
    - Scale up your project

Course Objective

- At the end of 10 weeks:
  - You should be able to perform basic programming tasks in R
    - Irrespective whether they are related to data analysis or your work
    - Appreciate programming as a means to accomplish huge no. of smaller tasks
    - Build complex logic and work-flow through small and concise functions
  - Have a good understanding of how to approach an empirical project
    - Data sources, merging, cleaning etc
  - Perform data analysis
    - Summaries, plots, regressions
    - Scale up your project

- What should you do?
  - Practice, practice and practice. There is no other way to learn programming.
    - Programming nuances
    - Experiment

Course Outline

- Module I (Lectures 1, 2 and 3)
  - Basics of Computer Architecture and Programming
  - Intro to Programming through R
  - Introduction to data.frame
  - Functions
  - useful R methods
  - Loop Functions

Course Outline

- Module I (Lectures 1, 2 and 3)
  - Basics of Computer Architecture and Programming
  - Intro to Programming through R
  - Introduction to data.frame
  - Functions
  - useful R methods
  - Loop Functions

- Module II (Lectures 4 and 5)
  - Introduction to *data.table* R package: syntax, usage and benefits
  - Merging datasets
  - Long form and wide form

- Module III (Lectures 6, 7 and 8)
  - Mini Project
    - Study of NASA climate data
      - data.table one-liners
  - Introduction to Data Analysis
    - Steps in a Data analysis project
    - Nuances: missing values, repeating data and extremes
  - Plotting in R
    - legends, colors, line types, …
    - Multiple lines, multiple axes, multiple plots
  - Regression Basics
    - meaning of significance and R2
    - Introduction to *felm* package
      - Fixed effects, error clustering

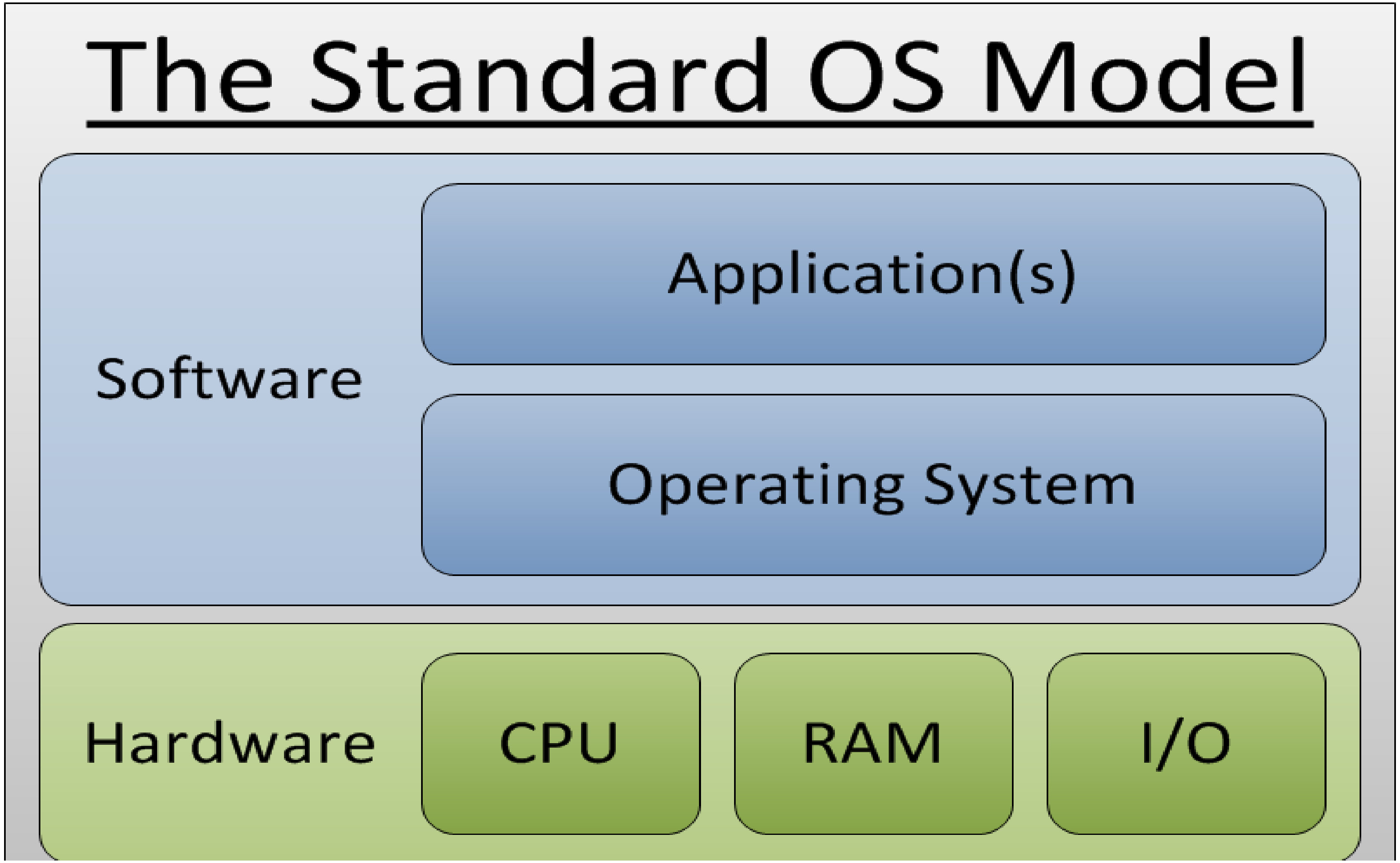
- Module IV (Lectures 9 and 10)
  - Mini Project
    - A country-wide panel of CO_2 emissions
  - Miscellaneous
    - Running different types of regressions in R
      - OLS, GLS, IV, logit, GMM
    - Rmarkdown/latex for documenting your work
    - Other useful stuff!

Session-1

Module - I

- Basics of Computer Architecture and Programming
- Intro to Programming through R
- Introduction to data.frame
- Functions
- useful R methods
- Loop Functions

What’s a computer look like?



What does a Computer do?

What does a Computer do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed

What does a Computer do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed
- Stores data
  - Cache vs RAM vs HDD
    - SSD
  - Speed vs storage cost

What does a Computer do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed
- Stores data
  - Cache vs RAM vs HDD
    - SSD
  - Speed vs storage cost
- Runs Software
  - System (OS): Linux, Windows and Mac-OS
  - Application: R, RStudio, Excel

What is a program?

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish
- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!
- Complexity of a Program
  - Time and space!
  - E.g. Fibonacci series! (next slide)

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!

- Complexity of a Program
  - Time and space!
  - E.g. Fibonacci series! (next slide)

- Programming Paradigms
  - Iterative vs Recursive
  - Procedural vs Object Oriented
    - No need to understand OOP concepts unless you wish to build a software (like an R package)

Fibonacci Time Complexity Example

Fibonacci Time Complexity Example

- Fibonacci numbers are defined as:

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.

Fibonacci Time Complexity Example

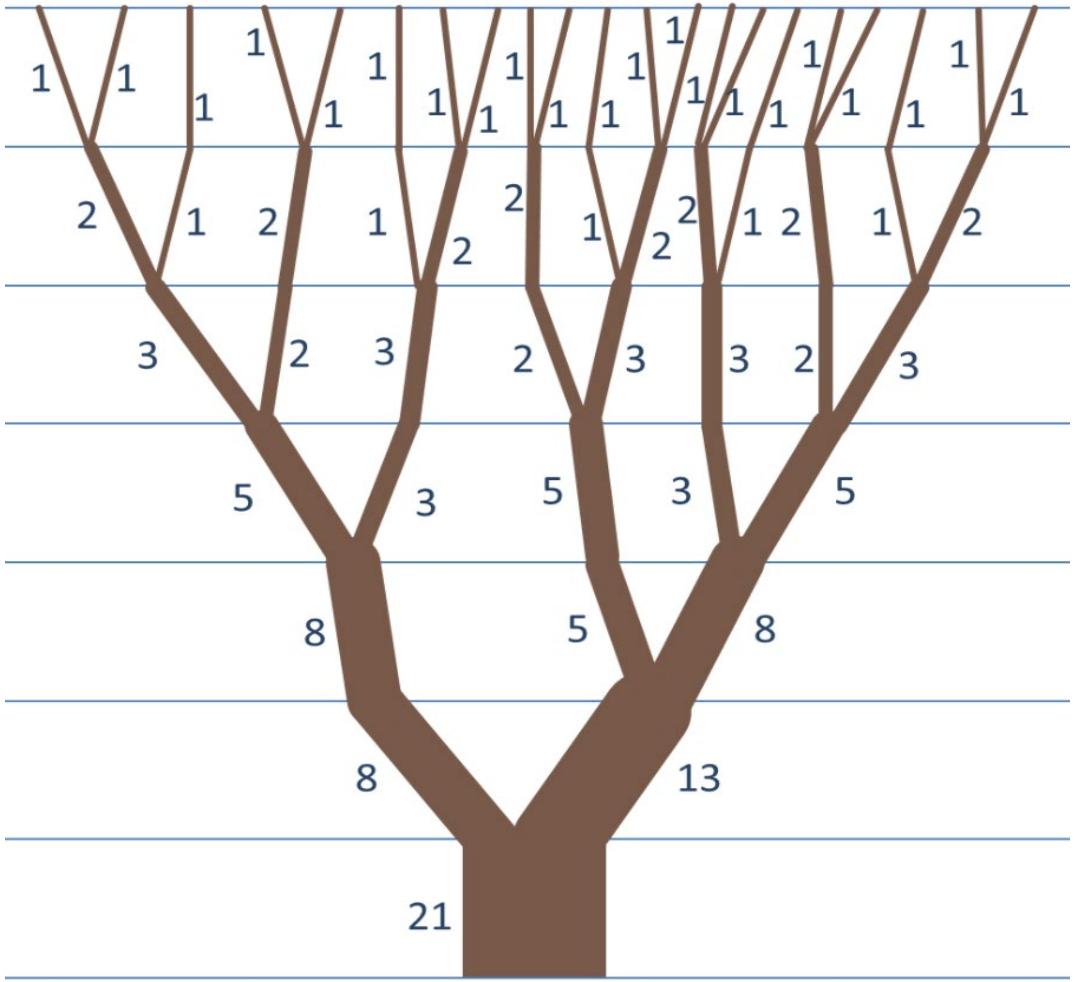
- Fibonacci numbers are defined as:

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.



Fibonacci Time Complexity Example

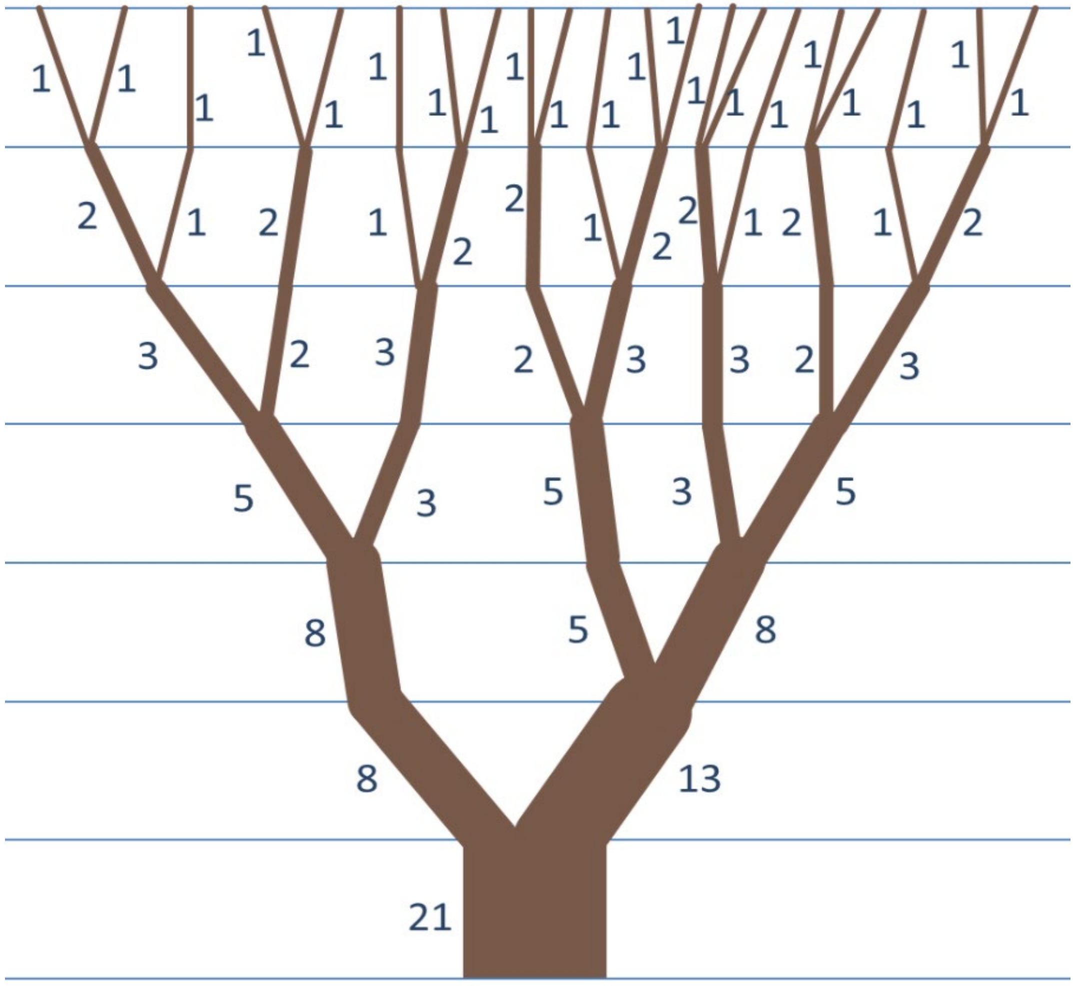
- Fibonacci numbers are defined as:

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.



- Thus, finding a Fibonacci number is pretty straight-forward.

3 ways to find Fib(n)

3 ways to find Fib(n)

- Method-1,
  return 1
  - Fib(n-1) + Fib(n-2)
  - Recursion
  - Base case:
    - If $n \leq 2$,
      return 1

3 ways to find Fib(n)

- Method-1,
  return 1
  - Fib(n-1) + Fib(n-2)
  - Recursion
  - Base case:
    - If $n \leq 2$,
      return 1

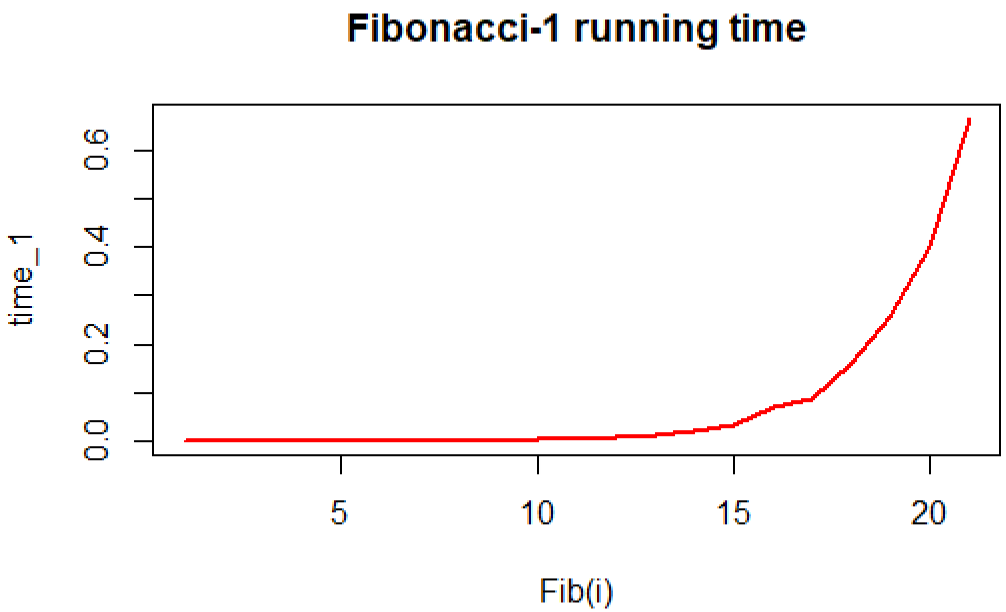
- Method-2
  - a = b = 1
  - for $i \in 3 \ldots n$
    - f = a + b
    - a = b
    - b = f
  - Iterative
  - Base Case:
    - If $n \leq 2$,
      return 1

3 ways to find Fib(n)

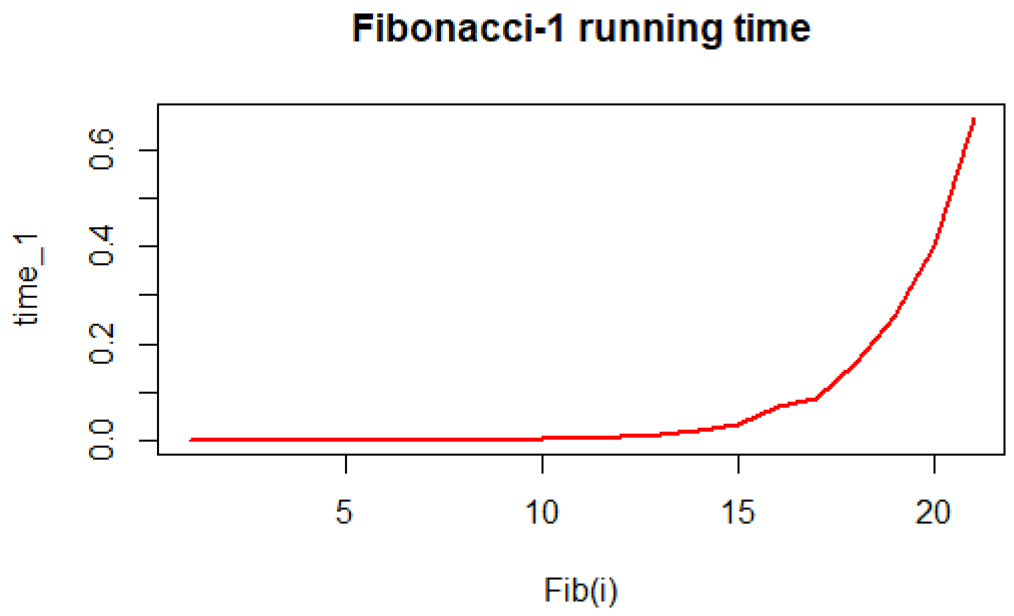
- Method-1,
  return 1
  - Fib(n-1) + Fib(n-2)
  - Recursion
  - Base case:
    - If $n \leq 2$,
      return 1

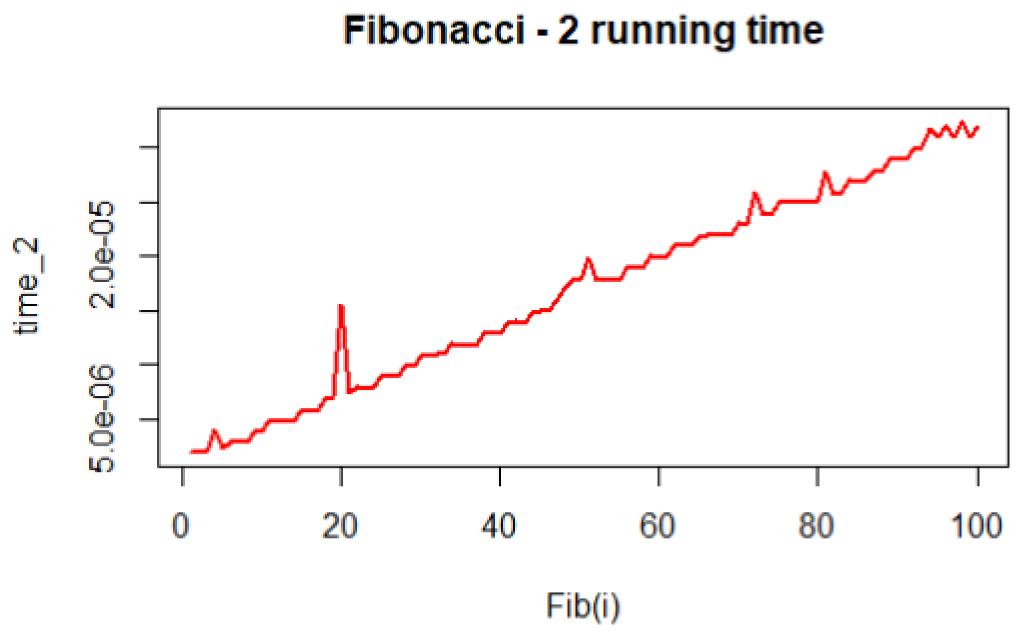
- Method-2
  - a = b = 1
  - for $i \in 3 \dots n$
    - f = a + b
    - a = b
    - b = f
  - Iterative
  - Base Case:
    - If $n \leq 2$,
      return 1

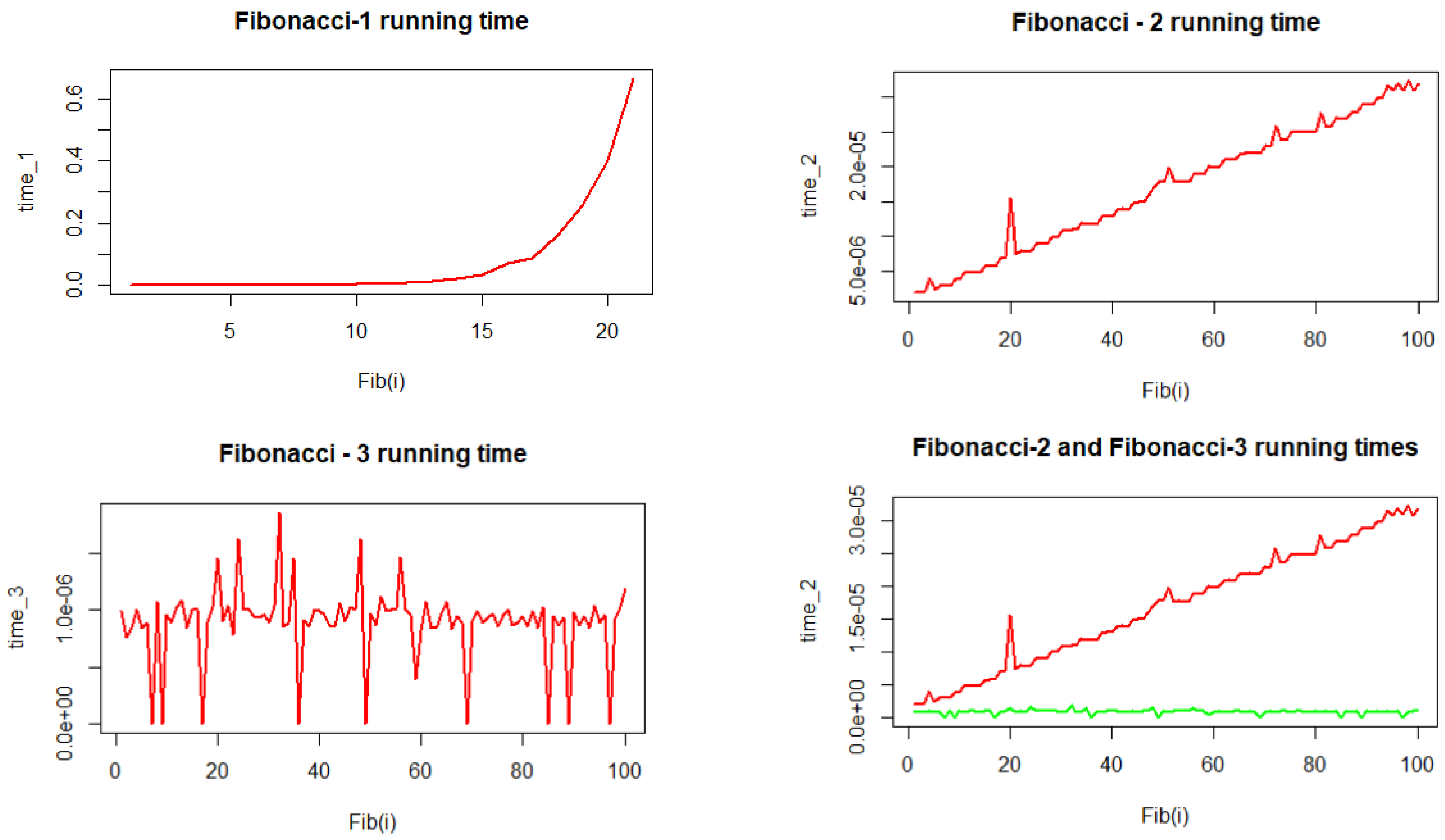
- Method-3
  - $\phi = \frac{1+\sqrt{5}}{2}$
  - $\hat{\phi} = \frac{1-\sqrt{5}}{2}$
  - $Fib(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$
  - Constant time
    - Really?



Fibonacci-1 running time
time_1
0.0
0.2
0.4
0.6
5
10
15
20
Fib(i)


Fibonacci-1 running time
time_1
0.0
0.2
0.4
0.6
5
10
15
20
Fib(i)


Fibonacci - 2 running time
time_2
5.0e-06
2.0e-05
0
20
40
60
80
100
Fib(i)


Fibonacci-1 running time
time_1
0.0
0.2
0.4
0.6
5
10
15
20
Fib(i)
Fibonacci - 2 running time
time_2
5.0e-06
2.0e-05
0
20
40
60
80
100
Fib(i)
Fibonacci - 3 running time
time_3
0.0e+00
1.0e-06
0
20
40
60
80
100
Fib(i)
Fibonacci-2 and Fibonacci-3 running times
time_2
0.0e+00
1.5e-05
3.0e-05
0
20
40
60
80
100
Fib(i)

What makes a good program?

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn’t worth pursuing

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn’t worth pursuing
- Readability
  - Make your code succinct and comments verbose
    - You will forget your code within weeks
    - The key to writing good code is the ability to quickly recap previously written code

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn't worth pursuing
- Readability
  - Make your code succinct and comments verbose
    - You will forget your code within weeks
    - The key to writing good code is the ability to quickly recap previously written code
- Organized
  - Writing code is an art. Much like a story!
  - Make small modules which can be reused later. Make it like a building: Bricks → Wall → floor → Building

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn’t worth pursuing
- Readability
  - Make your code succinct and comments verbose
    - You will forget your code within weeks
    - The key to writing good code is the ability to quickly recap previously written code
- Organized
  - Writing code is an art. Much like a story!
  - Make small modules which can be reused later. Make it like a building: Bricks → Wall → floor → Building
- Error Handling
  - Identify potential errors and print messages so that you know what and where problem occurred
  - For time consuming code, print regular messages (logs) in a file
    - You don’t want to run 4 hours of code just to find a small bug! Instead look at the log file.

Session-2

Reflection on Session-1

Reflection on Session-1

- Why are you interested in data analysis? What you hope to achieve?

Reflection on Session-1

- Why are you interested in data analysis? What you hope to achieve?

- Why do you need to learn R?

Reflection on Session-1

- Why are you interested in data analysis? What you hope to achieve?
- Why do you need to learn R?
- Why can’t you use Excel, Stata or SAS?

Reflection on Session-1

- Why are you interested in data analysis? What you hope to achieve?

- Why do you need to learn R?

- Why can’t you use Excel, Stata or SAS?

- Is it okay to write inefficient (time) programs?

Typical Programing Errors

Typical Programing Errors

- Syntactical (spelling mistake)
  - Will get caught very easily! Just run the program.
    - `a = c("hi"; "there")`

Typical Programing Errors

- Syntactical (spelling mistake)
  - Will get caught very easily! Just run the program.
    - `a = c("hi"; "there")`
- Semantic Errors (meaningless operations)
    - For e.g. `"iimb" + 32`
  - Exceptions: like divide by 0.
  - May get caught. A warning will be thrown nonetheless.

Typical Programing Errors

- Syntactical (spelling mistake)
  - Will get caught very easily! Just run the program.
    - `a = c("hi"; "there")`

- Semantic Errors (meaningless operations)
    - For e.g. `"iimb" + 32`
  - Exceptions: like divide by 0.
  - May get caught. A warning will be thrown nonetheless.

- Logical Errors (Unintentional)
  - Program will crash, run forever or give a wrong answer!
    - Like using `i` in place of `j`, adding in place of multiplying, …
    - These will happen when you first learn programming.
    - You'll get better with skill and experience.

What is R

What is R

- Implementation of S Programming language
  - Started as statistical environment
  - Explains the deep rootedness of R in statistics
  - Mostly written in C (earlier FORTRAN)
  - More info on Wikipedia!

What is R

- Implementation of S Programming language
  - Started as statistical environment
  - Explains the deep rootedness of R in statistics
  - Mostly written in C (earlier FORTRAN)
  - More info on Wikipedia!

- Philosophy behind R (or S, S+)
  - Interactive environment
  - Transition from users to Programmers as per need!
  - You don’t need to be a programmer to learn (and) use basic R
  - More info at http://ect.bell-labs.com/sl/S/history.html

What is R (cont.)

What is R (cont.)

- Features
  - Very easy to follow and understand
    - Require understanding of vector and matrix indexing!
    - Interactive
  - Runs on all platforms.
    - Small software to download and load. Use packages as per need.
  - Free of cost. Open source software (GNU GPL). More info at www.fsf.org
  - Wide availability of user developed packages!
  - Very active development
    - Frequent updates and releases
    - Very active and responsive user community – Stackoverflow!

What is R (cont.)

- Features
  - Very easy to follow and understand
    - Require understanding of vector and matrix indexing!
    - Interactive
  - Runs on all platforms.
    - Small software to download and load. Use packages as per need.
  - Free of cost. Open source software (GNU GPL). More info at [www.fsf.or](http://www.fsf.org)g
  - Wide availability of user developed packages!
  - Very active development
    - Frequent updates and releases
    - Very active and responsive user community – Stackoverflow!

- Drawbacks
  - Limited 3-D graphics capability
    - Rarely needed in management research or applications
  - Everything must be in RAM – big data?
    - Buy more RAM or use AWS!
  - If a functionality is missing you got to code it yourself!
    - Very rare! Opens new avenues for research!

Alternatives to R

Alternatives to R

- There are several high-level and interpreted languages around
  - Most common are Python and MATLAB
  - MATLAB is used much more in engineering than in statistics
    - It may not support the great variety of linear/non-linear/regression models
    - Syntax is similar to R (Read: http://mathesaurus.sourceforge.net/octave-r.html)
  - Python is also very popular although its more used in data science
    - Computation heavy research (like text analysis and ML) also employ Python routinely
  - Nobody stops you from using multiple languages for your research
    - Several researchers also perform regressions in Stata as well

Alternatives to R

- There are several high-level and interpreted languages around
  - Most common are Python and MATLAB
  - MATLAB is used much more in engineering than in statistics
    - It may not support the great variety of linear/non-linear/regression models
    - Syntax is similar to R (Read: http://mathesaurus.sourceforge.net/octave-r.html)
  - Python is also very popular although its more used in data science
    - Computation heavy research (like text analysis and ML) also employ Python routinely
  - Nobody stops you from using multiple languages for your research
    - Several researchers also perform regressions in Stata as well

- Statistical Alternatives?
  - SAS and Stata
    - Paid and closed software
    - If Stata implements an algorithm, I can't see their code. Only source is their documentation.
  - Very different than R in syntax!
    - Non-interactive
    - Limited user community support (huge deal-breaker)
  - Despite the differences Stata is very popular in management research.
  - SAS has a lot of legacy code and hence it is still used a lot in Finance research

Downloading and Installing R

Downloading and Installing R

- Download R: https://cran.r-project.org/
  - Choose base package for your OS
    - Windows: https://cran.r-project.org/bin/windows/base/R-4.0.3-win.exe
    - Linux: Use apt-get (Debian based) OR yum install (RPM based) from terminal.
    - Mac: https://cran.r-project.org/bin/macosx/R-4.0.3.pkg
  - Install R

Downloading and Installing R

- Download R: https://cran.r-project.org/
  - Choose base package for your OS
    - Windows: https://cran.r-project.org/bin/windows/base/R-4.0.3-win.exe
    - Linux: Use apt-get (Debian based) OR yum install (RPM based) from terminal.
    - Mac: https://cran.r-project.org/bin/macosx/R-4.0.3.pkg
  - Install R

- Download RStudio IDE
  - Choose the free RStudio Desktop edition
  - https://www.rstudio.com/products/rstudio/download/#download
  - Choose the appropriate one according to your OS
  - Install RStudio

Getting Help in R

Getting Help in R

- From Console
  - Just type: ? followed by function name without parenthesis
  - E.g. `?mean; ?sum; ?length;`
  - Clarify:
    - **?mean** - help for the function “mean”
    - **??mean** - will perform the search over the internet (CRAN database)
      - Look for base::mean!
    - **mean()** - call the function mean
    - **mean** - print the definition of the function “mean”

Getting Help in R

- From Console
  - Just type: ? followed by function name without parenthesis
  - E.g. `?mean; ?sum; ?length;`
  - Clarify:
    - **?mean** - help for the function "mean"
    - **??mean** - will perform the search over the internet (CRAN database)
      - Look for base::mean!
    - **mean()** - call the function mean
    - **mean** - print the definition of the function "mean"

- From Web sources
  - Most reliable and easy to incorporate is www.stackoverflow.com.
  - www.r-bloggers.com is also quite helpful.
  - You can use https://cran.r-project.org for any resource on R
    - Read the package vignette and manuals
    - e.g. data.table CRAN; data.table vignette; data.table manual
  - Even typing your question in google will get you good results!
    - 99% of your questions are already answered! You just need to find them!

R Input and Output

R Input and Output

- Simple assignment
  - `X = 1;` (*or* `X <- 1;`)
  - Assignment is always right to left
    - Read 1 goes into X
    - We aren't comparing X with 1 here
    - Learn this by heart! (for first-time programmers)
  - The semi-colon isn't necessary in R, but it's a good practice to use it
    - semi-colon is an instruction demarcation. Meaning you can write `a = 1; b = 2` in one line.
    - `X = ;` is incomplete
  - `#` (prefix) is used as a comment. Use it for helpful comments.
    - Use Ctrl-Shift-C for multi-line comments

R Input and Output

- Simple assignment
  - `X = 1;` (*or* `X <- 1;`)
  - Assignment is always right to left
    - Read 1 goes into X
    - We aren’t comparing X with 1 here
    - Learn this by heart! (for first-time programmers)
  - The semi-colon isn’t necessary in R, but it’s a good practice to use it
    - semi-colon is an instruction demarcation. Meaning you can write `a = 1; b = 2` in one line.
    - `X = ;` is incomplete
  - `#` (prefix) is used as a comment. Use it for helpful comments.
    - Use Ctrl-Shift-C for multi-line comments

- Value of X can be seen by writing `X` and hitting enter

Vectors

Vectors

- A sequence of numbers. Many ways to input!
  - `Y = c(1,7,-3,41); # concatenate arbitrary numbers`
  - `Y = 1:10; # natural numbers`
  - `Y = seq(1,100,9); # skip by 9`
  - `Y = rep(2, 3); # repeat 2 3-times`
  - `Y = rep(1:2, 3); # repeat the vector c(1,2) 3-times`
  - `Y = rep(1:2, each = 3); # repeat each element of c(1,2) 3-times`
  - `Y = c(); # empty vector`
  - Execute this: `Y = c(1:3, rep(c(5,7), each = 2), rep(9, 4), 7);`

Vectors

- A sequence of numbers. Many ways to input!
  - `Y = c(1,7,-3,41); # concatenate arbitrary numbers`
  - `Y = 1:10; # natural numbers`
  - `Y = seq(1,100,9); # skip by 9`
  - `Y = rep(2, 3); # repeat 2 3-times`
  - `Y = rep(1:2, 3); # repeat the vector c(1,2) 3-times`
  - `Y = rep(1:2, each = 3); # repeat each element of c(1,2) 3-times`
  - `Y = c(); # empty vector`
  - Execute this: `Y = c(1:3, rep(c(5,7), each = 2), rep(9, 4), 7);`

- Length of vector: `length(Y);`

Vectors

- A sequence of numbers. Many ways to input!
  - `Y = c(1,7,-3,41); # concatenate arbitrary numbers`
  - `Y = 1:10; # natural numbers`
  - `Y = seq(1,100,9); # skip by 9`
  - `Y = rep(2, 3); # repeat 2 3-times`
  - `Y = rep(1:2, 3); # repeat the vector c(1,2) 3-times`
  - `Y = rep(1:2, each = 3); # repeat each element of c(1,2) 3-times`
  - `Y = c(); # empty vector`
  - Execute this: `Y = c(1:3, rep(c(5,7), each = 2), rep(9, 4), 7);`

- Length of vector: `length(Y);`

- Accessing i^{th} element of vector: `Y[i];`
  - square (not curly or parenthesis) brackets
  - `i` should be between `1` and `length(Y)`
  - Printing the entire vector is as before: `Y`

Objects in R

Objects in R

- 5 basic (atomic) types of objects
  - character – strings
  - numeric – real numbers. Also called double.
  - integer – natural numbers. Default data type for numeric vectors.
    - `typeof(1:10)`
    - Execute: `as.integer(2^31 - 1)` and then `as.integer(2^31)`
    - There is raw data type as well. It represents hexadecimal numbers. Try: `as.raw(255)`
  - complex – complex numbers. We won't use them now!
    - `2 + 3i`
  - logical – True/False (binary)
    - `T, F, TRUE, FALSE`

Objects in R

- 5 basic (atomic) types of objects
  - character – strings
  - numeric – real numbers. Also called double.
  - integer – natural numbers. Default data type for numeric vectors.
    - `typeof(1:10)`
    - Execute: `as.integer(2^31 - 1)` and then `as.integer(2^31)`
    - There is raw data type as well. It represents hexadecimal numbers. Try: `as.raw(255)`
  - complex – complex numbers. We won't use them now!
    - `2 + 3i`
  - logical – True/False (binary)
    - `T, F, TRUE, FALSE`

- Most basic collection of objects is a vector (also called an array)
  - Can only contain objects of same class (i.e. character or integer; not both)
  - "list" is a general object type and can contain heterogeneous objects as its members
    - Any Combination of vector, matrix, atomic types etc.
    - It can even contain another list as an object. E.g. linked-lists!
    - Due to its generality its very slow and hence rarely used with large datasets unless situation demands it

Numbers

Numbers

- Default type of any number is numeric (i.e. real). `typeof(1)`

Numbers

- Default type of any number is numeric (i.e. real). `typeof(1)`
- R can differentiate between corner cases:
  - `1/0` is `Inf` -- `is.infinite();`
  - `0/0` is `NaN` -- `is.nan();`
  - Missing data is `NA` -- `is.na();`
  - Check what’s `Inf-Inf` ?

Numbers

- Default type of any number is numeric (i.e. real). `typeof(1)`
- R can differentiate between corner cases:
  - `1/0` is `Inf` -- `is.infinite();`
  - `0/0` is `NaN` -- `is.nan();`
  - Missing data is `NA` -- `is.na();`
  - Check what's `Inf-Inf` ?

- Arithmetic Operations
  - `*` multiplies
  - `/` divides
  - `^` takes exponent
  - `%%` is the modulo (remainder) operator. Try: `7 %% 2;`

Coercion

Coercion

- Mixing Objects
  - Automatically coerced to the same class.
  - Try: `c(1:7, "a"); c(T, 2); c("a", FALSE);`
  - Implicit coercion!
  - Never use unless you know what you’re doing!
    - Even then its better to explicitly coerce objects

Coercion

- Mixing Objects
  - Automatically coerced to the same class.
  - Try: `c(1:7, "a"); c(T, 2); c("a", FALSE);`
  - Implicit coercion!
  - Never use unless you know what you’re doing!
    - Even then its better to explicitly coerce objects

- Explicit Coercion
  - `as.character(1:5);`
  - `as.numeric("iimb"); # warning!`
  - `as.logical(seq(-2,2,1));`

List

List

- Can carry different types of data together
  - `L = list(1, FALSE, 3.14, "iimb", "c", 4-3i, list("2nd-list"));`
  - Print list: `L;`
  - Check: `typeof(L); typeof(L[4]); typeof(L[[4]]); typeof(L[[7]]);`
  - Single square brackets `[i]` access the i^{th} list embedded in the list `L`
    - It's a pointer to the element (don't bother if you don't know what a pointer is!)
  - Double square brackets `[[i]]` access the i^{th} element
  - Can append elements in list: `L = append(L, "8-th");`
  - `unlist(L);` will coerce all elements into a single type and return a vector
  - Delete an element from a list:
    - I don't know how to do that!
    - Let's google: "delete element from list in R"
    - Open the answer on [www.stackoverflow.co](http://www.stackoverflow.com)m

Matrices

Matrices

- Generalization of vectors
  - 2 dimensions instead of one!

Matrices

- Generalization of vectors
  - 2 dimensions instead of one!
  - N x K matrix means a matrix having N rows and K columns. Total of NK elements.
  - `M = matrix(nrow = 2, ncol = 3);`
  - Dimensions: `dim(M);`
  - Can think of M as
    - 3 columns vectors each of length 2, *or*
    - 2 row vectors each of length 3
  - Populate matrix:
    - `M = rbind(1:3, 4:6);`
    - `M = matrix(1:6, nrow = 2, byrow = T); # default is by column`
    - `M = cbind(1:2, 3:4, 5:6);`

Matrices (cont.)

- Indexing a matrix
  - `M[i,j]` gives the element at i^{th} row and j^{th} column
  - `M[i,]` gives the entire i^{th} row (a vector)
  - `M[,j]` gives the entire j^{th} column (a vector)

Matrices (cont.)

- Indexing a matrix
  - `M[i,j]` gives the element at i^{th} row and j^{th} column
  - `M[i,]` gives the entire i^{th} row (a vector)
  - `M[,j]` gives the entire j^{th} column (a vector)

- Matrix multiplication
  - `%*%` performs the usual matrix multiplication. Try: `M %*% M`
    - Dimensions must match
    - Try `t(M) %*% M;`
    - `t(M)` takes transpose of a matrix!
  - `M*N` perform element-wise multiplication

Matrices (cont.)

- Identity matrix: `diag(3)`
- Diagonal Matrix: `diag(c(1,5,7)); diag(1:7);`
- Diagonal of a matrix: `diag(M)`
- Trace of a matrix: `sum(diag(M))`
- Inverse of a matrix:
  - Must be a square matrix: `M = matrix(1:9, nrow = 3, ncol = 3);`
    - Another way to create a matrix. Data is entered column-wise.
  - Determinant must be non-zero: `det(M); M[3,3] = 19; det(M);`
  - Inverse: `solve(M);`

Higher (> 2) dimension Arrays

Higher (> 2) dimension Arrays

- ```
  A = array(1:24, dim = c(2,3,4),
            dimnames = list(paste0("ROW-", 1:2),
                            paste0("COL-", 1:3),
                            paste0("MATRIX-", 1:4)));
  ```
- `A[,,i]` will be a 2x3 matrix, `A[,i,j]` will be a 2x1 vector, `A[i,j,k]` is a scaler.
  - What the dimension of `A[i,,]`, `A[,j,]`, `A[i,,k]` and `A[i,j,]` ?
  ```

Higher (> 2) dimension Arrays

-
```
A = array(1:24, dim = c(2,3,4),
          dimnames = list(paste0("ROW-", 1:2),
                          paste0("COL-", 1:3),
                          paste0("MATRIX-", 1:4)));
```
- `A[,,i]` will be a 2x3 matrix, `A[,i,j]` will be a 2x1 vector, `A[i,j,k]` is a scaler.
  - What the dimension of `A[i,,]`, `A[,j,]`, `A[i,,k]` and `A[i,j,]` ?
- They have a limited use
  - For instance, if you need to compute 10,000 matrices (of dimension NxN) and then add them, then it’s better to define an array of dimension `c(N,N,10000)` and after computation do `apply(A, c(1,2), sum)`
  - data.frames/data.tables are almost always easier to build, interpret and summarize!

Factors

Factors

- For categorical data
  - Male, female
  - Cities in a dataset
  - Typically useful when the dataset is large but the no. of categories is small
  - Using factors is more descriptive than integer values
    - Rather than using 1 for PGP, 2 for FPM and 3 for Others; its more intuitive to use factors.

Factors

- For categorical data
  - Male, female
  - Cities in a dataset
  - Typically useful when the dataset is large but the no. of categories is small
  - Using factors is more descriptive than integer values
    - Rather than using 1 for PGP, 2 for FPM and 3 for Others; its more intuitive to use factors.
- Example:
  - `sex = rep(c("male", "female"), 5);`
  - `sex_f = as.factor(sex);`
  - `Check: typeof(sex_f); as.integer(sex_f);`

Factors

- For categorical data
  - Male, female
  - Cities in a dataset
  - Typically useful when the dataset is large but the no. of categories is small
  - Using factors is more descriptive than integer values
    - Rather than using 1 for PGP, 2 for FPM and 3 for Others; its more intuitive to use factors.
- Example:
  - `sex = rep(c("male", "female"), 5);`
  - `sex_f = as.factor(sex);`
  - `Check: typeof(sex_f); as.integer(sex_f);`
- Useful in the regression framework using `lm();`
  - Automatically creates dummy for all but one categories.
  - Conversion between integers (like year) and factor can corrupt your data!
  - Try: `as.integer(as.factor(2000:2020))`

Session - 3

Reflection on Session-2

Data Frame

Data Frame

- Probably the most important data type you'll use.
  - All external data (from excel, csv, tables, webpages etc) is read as data frame
  - It's a list where each element of list must have the same length.
  - Think of it like a matrix but with the flexibility that each column can have different data type. E.g. set of Names, weights and heights
  - Example:
    - `d = data.frame(name = c("a", "b"), weight = c(70, 75), height = c(1.78, 1.82));`
    - `d; d$name; d[1,]; d$weight; d[,3];`
    - `d$bmi = d$weight / (d$height^2); # new row`
    - `nrow(d); ncol(d); dim(d);`
    - `colnames(d)[1] = "names";`
    - `rownames(d) = c("I", "II");`

Reading Data

Reading Data

- Download some stock data from NSE
  - https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm
  - Save the CSV file as data.csv

Reading Data

- Download some stock data from NSE
  - https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm
  - Save the CSV file as data.csv
- From CSV (most common)
  - `setwd("C:/Users/nikhi/Downloads/");`

    `nifty = read.csv("data.csv");`
  - Alternatively: `nifty = read.csv("C:/Users/nikhi/Downloads/data.csv");`

Reading Data

- Download some stock data from NSE
  - https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm
  - Save the CSV file as data.csv
- From CSV (most common)
  - ```
    setwd("C:/Users/nikhi/Downloads/");
    nifty = read.csv("data.csv");
    ```
  - Alternatively: `nifty = read.csv("C:/Users/nikhi/Downloads/data.csv");`
- From Excel
  - Search it yourself! It is not recommended btw.
    ```

Reading Data

- Download some stock data from NSE
  - https://www.nseindia.com/products/content/equities/indices/historical_index_data.htm
  - Save the CSV file as data.csv
- From CSV (most common)
  - `setwd("C:/Users/nikhi/Downloads/");`
    `nifty = read.csv("data.csv");`
  - Alternatively: `nifty = read.csv("C:/Users/nikhi/Downloads/data.csv");`
- From Excel
  - Search it yourself! It is not recommended btw.
- From clipboard
  - `read.table("clipboard");`
  - This is quick fix for small data transfer between R and excel. Use `read.csv()` as your primary method for data reading!

Reading Data (cont.)

Reading Data (cont.)

- Viewing data
  - `View(nifty);`

Reading Data (cont.)

- Viewing data
  - `View(nifty);`
- Date

```
nifty$Date = as.Date(nifty$Date, format = "%d-%b-%Y");
n = nrow(nifty);
d = nifty$Date[1];
format(d, format = "%D");             #  10/01/20
format(d, format = "%d-%m-%y");       #  01-10-20
format(d, format = "%d.%b.%Y");       #  01.Oct.2020
format(d, format = "%A, %B %d, %Y") #  Thursday, October 01, 2020
```

Reading Data (cont.)

- Viewing data
  - `View(nifty);`
- Date

```
nifty$Date = as.Date(nifty$Date, format = "%d-%b-%Y");
n = nrow(nifty);
d = nifty$Date[1];
format(d, format = "%D");            #  10/01/20
format(d, format = "%d-%m-%y");      #  01-10-20
format(d, format = "%d.%b.%Y");      #  01.Oct.2020
format(d, format = "%A, %B %d, %Y") #  Thursday, October 01, 2020
```

- Alternatively,
  - `read.table("data.csv", header = T, sep = ",", nrows = 5);`

if-else

if-else

- ```
  if(<COND_1>) {
    # do something!
  }
  ```

- ```
  if(<COND_1>) {
    # do something!
  } else {
    # ...
  }
  ```

- ```
  if(<COND_1>) {
    # do something!
  } else if(<COND_2>) {
    # ...
  } else {
    # ...
  }
  ```
  ```

if-else

- ```
  if(<COND_1>) {
    # do something!
  }
  ```
- ```
  if(<COND_1>) {
    # do something!
  } else {
    # ...
  }
  ```
- ```
  if(<COND_1>) {
    # do something!
  } else if(<COND_2>) {
    # ...
  } else {
    # ...
  }
  ```

```
if(nifty$Close[2] > nifty$Close[1]) {
  str = paste("Stock market closed green on", nifty$Date[2]);
} else if(nifty$Close[2] > nifty$Open[2]) {
  str = paste("Stock market closed above opening on", nifty$Date[2]);
} else {
  str = paste("Stock market was red and closed below opening on", nifty$Date[2]);
}
print(str);
```
```

for loop

for loop

- Looping is used to perform similar set of tasks repetitively
  - ```
    for(i in n:1) {
      print(nifty$Date[i]);
    }
    ```
  - `n:1;` is same as `seq(n,1,1);` i.e. backwards counting!
  - Alternatively, you can execute: `rev(nifty$Date);` or `nifty$Date[n:1];`
    ```

for loop

- Looping is used to perform similar set of tasks repetitively
  - ```
    for(i in n:1) {
      print(nifty$Date[i]);
    }
    ```
  - `n:1;` is same as `seq(n,1,1);` i.e. backwards counting!
  - Alternatively, you can execute: `rev(nifty$Date);` or `nifty$Date[n:1];`

- Try avoiding loops if you can!
  - Increasing all dates by a week: `nifty$Date + 7`
  - Finding Daily growth: `nifty$Close[-1] / nifty$Close[-n]`
  - Daily diff. b/w high and low prices: `nifty$High - nifty$Low`
  - Question: find % growth in daily volatility
    - $G = \frac{(Value_{t+1} - Value_t)}{Value_t} * 100$
    ```

Nested if-else and for loop

Nested if-else and for loop

```
for(i in 2:n) {
  if(nifty$Close[i] > 1.01 * nifty$Close[i-1]) {
    # market gained more than 1%
    for(j in 1:ncol(nifty)) {
      print( paste("Gain", i, colnames(nifty)[j], nifty[i,j], sep =":") );
    } # end for(j)
  } else if(nifty$Close[i] < 0.99 * nifty$Close[i-1]) {
    # market lost more than 1%
    for(j in 1:ncol(nifty)) {
      print( paste("Loss", i, colnames(nifty)[j], nifty[i,j], sep =":") );
    }
  } else {
    print(paste("Market movement was within 1% for i =", i));
  } # end if()
} # end for(i)
```

Jumping

Jumping

- Till now all our commands executed sequentially
  - There may be circumstances when we need to jump

Jumping

- Till now all our commands executed sequentially
  - There may be circumstances when we need to jump
- Next and Break
  - `next` is used to skip an iteration, while `break` exits the loop entirely.
  - ```
    for(i in 1:10) {
      if(i <= 3) {
        next;
      }
      if(i > 6) {
        break;
      }
      print(i);
    }
    i;
    ```
    ```

Jumping

- Till now all our commands executed sequentially
  - There may be circumstances when we need to jump
- Next and Break
  - `next` is used to skip an iteration, while `break` exits the loop entirely.
  - ```
    for(i in 1:10) {
      if(i <= 3) {
        next;
      }
      if(i > 6) {
        break;
      }
      print(i);
    }
    i;
    ```
- `return()` is used to exit a function with a value
    ```

Function

Function

- Organize often-used set of instructions separately in a “function”
- Calling a function will execute all the commands in the body of function

Function

- Organize often-used set of instructions separately in a “function”
- Calling a function will execute all the commands in the body of function
- We have used many functions till now
    - They end with parenthesis: `()`
        - Not square or curly braces
    - E.g. `sum(); rbind(); vector(); format(); read.csv();` etc
    - Note that curly braces `{}` are used for if-else, for and function body, square braces `[]` for vector/matrix indexing and parenthesis `()` for grouping, if-else condition, for condition and functions arguments.

Function

- Organize often-used set of instructions separately in a “function”
- Calling a function will execute all the commands in the body of function
- We have used many functions till now
  - They end with parenthesis: `()`
    - Not square or curly braces
  - E.g. `sum(); rbind(); vector(); format(); read.csv();` etc
  - Note that curly braces `{}` are used for if-else, for and function body, square braces `[]` for vector/matrix indexing and parenthesis `()` for grouping, if-else condition, for condition and functions arguments.
- A function has
  - A name by which we call them, e.g. `sum`
  - A set of inputs to be put within parenthesis like numbers `1:10` in `sum()`
    - A function can have no input: `getwd()`
  - Return value which is the output of the function like the sum of numbers in `sum()`

Function Example

```
my_mean = function(x) {
  n = length(x);
  mean = sum(x) / n;
  return(mean);
}
```

Function Example

```
my_mean = function(x) {
  n = length(x);
  mean = sum(x) / n;
  return(mean);
}
```

- Name of the function is: `my_mean`
- Input is: `x`
- Output is: `mean`
  - Note that the mean here is just a name, we could well have used any other name without changing anything about our function

Function (Example) Cont.

- Alternate ways to write the same function
  - ```
    my_mean = function(x) {
      return( sum(x) / length(x) );
    }
    ```
    - No need to store sum and length. We can directly divide them!
  - ```
    my_mean = function(x) {
      sum(x) / length(x);
    }
    ```
    - No need for an explicit return. The last statement is returned by default.

Function (Example) Cont.

- Alternate ways to write the same function
  - ```
    my_mean = function(x) {
      return( sum(x) / length(x) );
    }
    ```
    - No need to store sum and length. We can directly divide them!
  - ```
    my_mean = function(x) {
      sum(x) / length(x);
    }
    ```
    - No need for an explicit return. The last statement is returned by default.
- Try various value with `my_mean()` and the inbuilt `mean()`. See that the answers are exactly the same.

Function (Example) Cont.

- Alternate ways to write the same function
  - ```
    my_mean = function(x) {
      return( sum(x) / length(x) );
    }
    ```
    - No need to store sum and length. We can directly divide them!
  - ```
    my_mean = function(x) {
      sum(x) / length(x);
    }
    ```
    - No need for an explicit return. The last statement is returned by default.
- Try various value with `my_mean()` and the inbuilt `mean()`. See that the answers are exactly the same.
- Exercise: Write your own version of variance function
  - $Var(x) = mean([x - mean(x)]^2)$
  - Compare it with the inbuilt `var()` function in R

Multiple conditions & which() function

Multiple conditions & which() function

- The arguments to `if()` and `which()` and the output of `is.xx()` family of functions is a logical object, i.e. either `TRUE` or `FALSE`.

Multiple conditions & which() function

- The arguments to `if()` and `which()` and the output of `is.xx()` family of functions is a logical object, i.e. either `TRUE` or `FALSE`.

- A valid combination of logical objects is also a logical object. E.g.
  - Logical AND: `TRUE & FALSE` is `FALSE`
  - Logical OR: `TRUE | FALSE` is `TRUE`
  - Logical NOT: `! FALSE` is `TRUE`

Multiple conditions & which() function

- The arguments to `if()` and `which()` and the output of `is.xx()` family of functions is a logical object, i.e. either `TRUE` or `FALSE`.

- A valid combination of logical objects is also a logical object. E.g.
  - Logical AND: `TRUE & FALSE` is `FALSE`
  - Logical OR: `TRUE | FALSE` is `TRUE`
  - Logical NOT: `! FALSE` is `TRUE`

- De Morgan's Law
  - $!\,(A \,\&\, B) = (!\,A) \mid (!\,B)$

- The below two indexes are one and same (by De Morgan Law),
  - `day = as.numeric( substr(nifty$Date, 9, 10) );`
    - OR `day = as.numeric( format(df[,1], format = "%d") );`

```
idx_1 = which( nifty$Close > nifty$Open    & (day < 5)    );
idx_2 = which(!(nifty$Close <= nifty$Open  | (day >= 5)) );
```

- The below two indexes are one and same (by De Morgan Law),
  - `day = as.numeric( substr(nifty$Date, 9, 10) );`
    - OR `day = as.numeric( format(df[,1], format = "%d") );`

```
idx_1 = which( nifty$Close > nifty$Open    & (day < 5)    );
idx_2 = which(!(nifty$Close <= nifty$Open  | (day >= 5)) );
```

- `which()` gives the indexes matching the criterion. E.g. out of `101:200` which numbers are multiples of 2,3 and 5 ?
  - `count = 101:200;`
  - `which( count %% 2 == 0 & count %% 3 == 0 & count %% 5 == 0 );`
  - `count[count %% 2 == 0 & count %% 3 == 0 & count %% 5 == 0];`

- The below two indexes are one and same (by De Morgan Law),
  - `day = as.numeric( substr(nifty$Date, 9, 10) );`
    - OR `day = as.numeric( format(df[,1], format = "%d") );`

```
idx_1 = which( nifty$Close > nifty$Open    & (day < 5)    );
idx_2 = which(!(nifty$Close <= nifty$Open  | (day >= 5)) );
```

- `which()` gives the indexes matching the criterion. E.g. out of `101:200` which numbers are multiples of 2,3 and 5 ?
  - `count = 101:200;`
  - `which( count %% 2 == 0 & count %% 3 == 0 & count %% 5 == 0 );`
  - `count[count %% 2 == 0 & count %% 3 == 0 & count %% 5 == 0];`

- We can do multi-way match using `%in%`
  - `mult_17 = seq(17,300,17);`
  - `which(count %in% mult_17);`
  - `which(mult_17 %in% count);`
  - `which(!(count %in% mult_17));`
  - `which(!(mult_17 %in% count));`

Some Useful functions

Some Useful functions

- `unique(), duplicated()`

Some Useful functions

- `unique(), duplicated()`
- `list.files()`
  - Pattern matching using regex
  - All files starting from “s”: `"^s"`
  - All files starting with “b” or “d”: `"^(b|d)“`
  - All CSV files: `".*.csv"`

Some Useful functions

- `unique(), duplicated()`
- `list.files()`
  - Pattern matching using regex
  - All files starting from “s”: `"^s"`
  - All files starting with “b” or “d”: `"^(b|d)“`
  - All CSV files: `".*.csv"`
- `order()`
  - Sort data/dataframes
  - It gives the sequence of ordered indexes NOT the ordered numbers
  - Can do 2-way and 3-way sorts

Some Useful functions

- `unique(), duplicated()`
- `list.files()`
  - Pattern matching using regex
  - All files starting from “s”: `"^s"`
  - All files starting with “b” or “d”: `"^(b|d)“`
  - All CSV files: `".*.csv"`
- `order()`
  - Sort data/dataframes
  - It gives the sequence of ordered indexes NOT the ordered numbers
  - Can do 2-way and 3-way sorts
- `union(), intersect()`

Some Useful functions

- `unique(), duplicated()`
- `list.files()`
  - Pattern matching using regex
  - All files starting from “s”: `"^s"`
  - All files starting with “b” or “d”: `"^(b|d)“`
  - All CSV files: `".*.csv"`
- `order()`
  - Sort data/dataframes
  - It gives the sequence of ordered indexes NOT the ordered numbers
  - Can do 2-way and 3-way sorts
- `union(), intersect()`
- `cumsum(), cumprod()`
  - Can you write your own version of `cumprod()` using only `cumsum()` ?

Loop Functions

Loop Functions

- Writing loops in a single command. Can come very handy and compact. The function name ends with “apply”.
  - `lapply(), apply(), sapply(), tapply(), mapply()`

Loop Functions

- Writing loops in a single command. Can come very handy and compact. The function name ends with “apply”.
  - `lapply(), apply(), sapply(), tapply(), mapply()`
- These functions work on a list of inputs, not just one input!
  - A data.frame is also a list.
  - Loop functions (esp. `lapply`) will be heavily used later!

Loop Functions

- Writing loops in a single command. Can come very handy and compact. The function name ends with “apply”.
  - `lapply(), apply(), sapply(), tapply(), mapply()`
- These functions work on a list of inputs, not just one input!
  - A data.frame is also a list.
  - Loop functions (esp. `lapply`) will be heavily used later!
- E.g.
  - `X = list(a = 1:10, b = rnorm(100, 0, 1), c = runif(1e3, 9, 91));`
  - `lappy(X, mean); # returns a list`
  - `sapply(X, mean); # returns a vector`

Loop Functions

- Writing loops in a single command. Can come very handy and compact. The function name ends with “apply”.
  - `lapply(), apply(), sapply(), tapply(), mapply()`
- These functions work on a list of inputs, not just one input!
  - A data.frame is also a list.
  - Loop functions (esp. `lapply`) will be heavily used later!
- E.g.
  - `X = list(a = 1:10, b = rnorm(100, 0, 1), c = runif(1e3, 9, 91));`
  - `lappy(X, mean); # returns a list`
  - `sapply(X, mean); # returns a vector`
- Let’s say we want to find `cor(X,X^i)` for `i in 1:10` w/o writing a loop?
  - `X = rnorm(1000, 0, 1);`
  - `sapply(1:10, function(i) cor(X,X^i));`
    - Here we have used anonymous function, i.e. a function w/o a name.

- `apply()` is mostly used for applying functions on rows or cols of a matrix
  - Like taking means by rows
  - `M = matrix(1:50, nrow = 10, ncol = 5); # data filled by column (default)`
  - `apply(M, 1, mean); # mean of each row (1st dimension)`
  - `apply(M, 2, mean); # mean of each col (2nd dimension)`
  - You can do the above using a loop also, but `apply()` is more compact.
  - The faster version of above are also available:
    - `rowSums(x)` is equivalent to `apply(x, 1, sum)`
    - `colMeans(x)` is equivalent to `apply(x, 2, mean)`
  - `apply()` works with multi-dimensional (> 2) arrays as well

- `apply()` is mostly used for applying functions on rows or cols of a matrix
  - Like taking means by rows
  - `M = matrix(1:50, nrow = 10, ncol = 5); # data filled by column (default)`
  - `apply(M, 1, mean); # mean of each row (1st dimension)`
  - `apply(M, 2, mean); # mean of each col (2nd dimension)`
  - You can do the above using a loop also, but `apply()` is more compact.
  - The faster version of above are also available:
    - `rowSums(x)` is equivalent to `apply(x, 1, sum)`
    - `colMeans(x)` is equivalent to `apply(x, 2, mean)`
  - `apply()` works with multi-dimensional (> 2) arrays as well

- `mapply()` is a multi-variate version of lapply/sapply:
  - Let's say `set.seed(1); u = rnorm(1000, 0, 1); v = rnorm(1000, 0, 1);`
    `X = u + v; Y = u - v;`
  - Suppose we need to find $cor(X^p, Y^q)$ for different values of p and q
    - We can't do this with `lapply()` since it only accepts one argument for looping
  - `mapply(p = 1:5, q = 5:1, function(p,q) cor(X^p, Y^q))`

Session - 4

Reflection on Session-3

Module - II

- Introduction to ***data.table*** R package: syntax, usage and benefits
- Merging datasets
- Long form and wide form

data.table

data.table

- It’s a (very) fast, memory efficient and flexible package to analyse data
  - I haven’t used data.frame since discovering data.table
- Extends data.frame
  - Most of the data.frame code will work

data.table

- It’s a (very) fast, memory efficient and flexible package to analyse data
  - I haven’t used data.frame since discovering data.table
- Extends data.frame
  - Most of the data.frame code will work
- Has a “different” and succinct syntax
  - May take some time to learn. But the effort is worth the benefits!
- Have a look:
  - https://cran.r-project.org/web/packages/data.table/index.html
  - github: https://github.com/Rdatatable/data.table

data.table

- It’s a (very) fast, memory efficient and flexible package to analyse data
  - I haven’t used data.frame since discovering data.table
- Extends data.frame
  - Most of the data.frame code will work
- Has a “different” and succinct syntax
  - May take some time to learn. But the effort is worth the benefits!
- Have a look:
  - https://cran.r-project.org/web/packages/data.table/index.html
  - github: https://github.com/Rdatatable/data.table
- Parallelized read and write
  - Very useful while reading GBs of raw data
    - Upto 5x – 10x speedup
    - Speed becomes a big issue when working with huge datasets

Syntax

- Column names can be used as variables
  - `dt[Date > “2020-01-01”]` is valid.
  - Remember in data.frame you need `dt[dt$Date > “2020-01-01”]`

  - Column names are infact variables inside `[]`
  - You can do `dt[, min(Date)]` to get the first date
  - In data.frame you must supply `dt$Date` to min externally.
    - `setDT()` and `setDF()`

- Grouping within `[]` syntax
  - Most of data analysis requires some form of grouping

Syntax: general form

DT[i, j, by]

Syntax: general form

DT[i, j, by]

Which rows?

- vector of rownumbers
- logical condition
- another data.table

What to do?

- summarizing
- updating variable(s)
- adding variable(s)

Grouped by what?

- one or more columns
- on the fly grouping var(s)

Analogy with SQL (queries)

DT[i, j, by]

data.table:	i	j	by
SQL:	where	select | update	group by

Examples

```
subset rows        :        airquality[Day <= 10]

select columns     :        airquality[, .(Month)]
```

- `airquality` dataset is present in package `datasets`

Examples

```
subset rows                :       airquality[Day <= 10]

select columns             :       airquality[, .(Month)]

subset rows & select columns :
                              airquality[Day %between% c(5, 10), .(Wind, Temp)]
```

- `airquality` dataset is present in package `datasets`

Counts

syntax: DT[**i**, **j**, **by**]

special symbol: **.N**

count

```
iris[Species == "setosa", .N]
```

count distinct

```
iris[, uniqueN(Species)]
iris[Petal.Width < 0.9, uniqueN(Species)]
```

Aggregating

syntax: DT[**i**, **j**, **by**]

Simple aggregation: iris[, .(count = .N, average = mean(Petal.Width))]

Including filtering: iris[Petal.Width < 0.9, .(count = .N, average = mean(Petal.Width))]

Group by

syntax: DT[**i**, **j**, **by**]

```r
iris[, .N, by = Species]

iris[, .(avg = mean(Petal.Width)), by = Species]

iris[Sepal.Length < 5.3, .(avg = mean(Petal.Width)), by = Species]

iris[, .(avg = mean(Petal.Width)), by = .(Species, logi = Sepal.Length < 5.3)]
```

Group by

special symbol: **.SD**

SD = **S**ubset of **D**ata

- a data.table by itself
- holds data of current goup as defined in by
- when no by, .SD applies to whole data.table
- allows for calculations on multiple columns

Group by

special symbol: **.SD**

```
iris[, lapply(.SD, mean), by = Species]
```

```
iris[Sepal.Length < 5.3, lapply(.SD, mean), by = Species]
```

Group by

special symbol: **.SD**

special symbol: **.SDcols**

```
iris[, lapply(.SD, mean), by = Species, .SDcols = 1:2]

iris[, lapply(.SD, mean), by = Species, .SDcols = grep("Length", names(irisDT))]
```

Examples

syntax: DT[**i**, **j**, **by**]

Count the number of days per month

airquality[, .N, by = Month]

Examples

syntax: DT[**i**, **j**, **by**]

Count the number of days per month

```
airquality[, .N, by = Month]
```

Calculate the average Wind speed by month for only those days that have an ozone value

```
airquality[!is.na(Ozone), mean(Wind), by = Month]
```

Examples

syntax: DT[i, j, **by**]

Count the number of days per month	`airquality[, .N, by = Month]`
Calculate the average Wind speed by month for only those days that have an ozone value	`airquality[!is.na(Ozone), mean(Wind), by = Month]`
Calculate the mean temperature for the odd and even days for each month	`airquality[, mean(Temp), by = .(Month, odd = Day %% 2)]`

Updating, adding & deleting variables

special operator: **:=**

- updates a data.table in place (by reference)
- can be used to:
  - update existing column(s)
  - add new column(s)
  - delete column(s)

- you don't need **<-** **OR** **=**

Updating variables

special operator: **:=**

```
iris[, Sepal.Length := Sepal.Length *  2]

iris[, `:=` (Sepal.Length = Sepal.Length * 2,
             Petal.Width = Petal.Width /  2)]
```

Updating variables by group

special operator: **:=**

```
iris[, Sepal.Length := Sepal.Length * uniqueN(Sepal.Width) / .N, by = Species]
```

```
iris[, `:=` (Sepal.Length = Sepal.Length * uniqueN(Sepal.Width),
                  Petal.Width = Petal.Width /  .N)
          , by =  Species]
```

Adding variables

special operator: **:=**

special symbol: **.I**

```
iris[, rownumber := .I]

iris[, Sepal.Area := Sepal.Length *  Sepal.Width]

iris[, `:=` (Sepal.Area = Sepal.Length * Sepal.Width,
             Petal.Area = Petal.Length *  Petal.Width)]
```

Adding variables by group

special operator: **:=**

```
iris[, Total.Sepal.Area := sum(Sepal.Area), by = Species]
```

```
iris[, `:=` (Total.Sepal.Area = sum(Sepal.Area),
             Total.Petal.Area = sum(Petal.Area))
     , by = Species]
```

Deleting variables

special operator: **:=**

```
iris[, Sepal.Length := NULL]

iris[, (1:4) := NULL]

iris[, grep("Length", names(irisDT)) := NULL]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Calculate a new **chill** variable (Wind * Temperature)

```
airquality[, chill := Wind * Temp]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind * 1.6]
```

Calculate a new **chill** variable (Wind * Temperature)

```
airquality[, chill := Wind * Temp]
```

Calculate the average chill by month and add that as a new variable

```
airquality[, mean.chill := mean(chill) ,
        by = Month]
```

Examples

Change the Wind column from miles per hour to kilometers per hour (1 mph = 1.6 kmh)

```
airquality[, Wind := Wind *  1.6]
```

Calculate a new **chill** variable (Wind * Temperature)

```
airquality[, chill := Wind *  Temp]
```

Calculate the average chill by month and add that as a new variable

```
airquality[, mean.chill := mean(chill) ,
             by =  Month]
```

Remove the **Ozone** and **Solar.R** columns

```
airquality[, c("Ozone ", "Solar.R ") := NULL]
airquality[, (1:2) := NULL]
```

Merging

Merging

- Combining two datasets is a very routine and important task
- Merging is akin to Joining (in relational database, SQL etc)

Merging

- Combining two datasets is a very routine and important task
- Merging is akin to Joining (in relational database, SQL etc)

- Summary:

$r =$

attr1	attr2
a	r1
b	r2
c	r3

$s =$

attr1	attr3
b	s2
c	s3
d	s4

$r \bowtie s$

attr1	attr2	attr3
b	r2	s2
c	r3	s3

$r ⟕ s$

attr1	attr2	attr3
a	r1	*null*
b	r2	s2
c	r3	s3

$r ⟖ s$

attr1	attr2	attr3
b	r2	s2
c	r3	s3
d	*null*	s4

$r ⟗ s$

attr1	attr2	attr3
a	r1	*null*
b	r2	s2
c	r3	s3
d	*null*	s4

Merging Example

Merging Example

- Taken from [Stackoverflow webpage]
- Create two datasets:
  - `df1 = data.frame(cust_id = c(1:6), Product = c(rep("Toaster", 3), rep("Radio", 3)));`
  - `df2 = data.frame(cust_id = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)));`
  - `dt1 = as.data.table(df1); dt2 = as.data.table(df2);`

Merging Example

- Taken from [Stackoverflow webpage]
- Create two datasets:
  - `df1 = data.frame(cust_id = c(1:6), Product = c(rep("Toaster", 3), rep("Radio", 3)));`
  - `df2 = data.frame(cust_id = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)));`
  - `dt1 = as.data.table(df1); dt2 = as.data.table(df2);`

df1		df2	
cust_id	Product	cust_id	State
1	Toaster	2	Alabama
2	Toaster	4	Alabama
3	Toaster	6	Ohio
4	Radio		
5	Radio		
6	Radio		

Merging Example

- Taken from Stackoverflow webpage
- Create two datasets:
  - `df1 = data.frame(cust_id = c(1:6), Product = c(rep("Toaster", 3), rep("Radio", 3)));`
  - `df2 = data.frame(cust_id = c(2, 4, 6), State = c(rep("Alabama", 2), rep("Ohio", 1)));`
  - `dt1 = as.data.table(df1); dt2 = as.data.table(df2);`

df1		df2	
cust_id	Product	cust_id	State
1	Toaster	2	Alabama
2	Toaster	4	Alabama
3	Toaster	6	Ohio
4	Radio		
5	Radio		
6	Radio		

- We can merge using `merge()` command on data.table OR we can use a new package `dplyr`
  - The function `merge()` works differently for data.frames and data.tables. It's very slow on DFs and extremely fast on DTs
    - data.table class overrides its own implementation of merge for DTs

Inner Join

- `merge(dt1, dt2, by = "cust_id");`
- `dplyr::inner_join(df1, df2);`

cust_id	Product	State
2	Toaster	Alabama
4	Radio	Alabama
6	Radio	Ohio

Inner Join

- `merge(dt1, dt2, by = "cust_id");`
- `dplyr::inner_join(df1, df2);`

cust_id	Product	State
2	Toaster	Alabama
4	Radio	Alabama
6	Radio	Ohio

Full Join

- `merge(dt1, dt2, by = "cust_id", all = T);`
- `dplyr::full_join(df1, df2);`

cust_id	Product	State
1	Toaster	NA
2	Toaster	Alabama
3	Toaster	NA
4	Radio	Alabama
5	Radio	NA
6	Radio	Ohio

Left Join

- `merge(dt1, dt2, by = "cust_id", all.x = T);`
- `dplyr::left_join(df1, df2);`

cust_id	Product	State
1	Toaster	NA
2	Toaster	Alabama
3	Toaster	NA
4	Radio	Alabama
5	Radio	NA
6	Radio	Ohio

Left Join

- `merge(dt1, dt2, by = "cust_id", all.x = T);`
- `dplyr::left_join(df1, df2);`

cust_id	Product	State
1	Toaster	NA
2	Toaster	Alabama
3	Toaster	NA
4	Radio	Alabama
5	Radio	NA
6	Radio	Ohio

Right Join

- `merge(dt1, dt2, by = "cust_id", all.y = T);`
- `dplyr::right_join(df1, df2);`

cust_id	Product	State
2	Toaster	Alabama
4	Radio	Alabama
6	Radio	Ohio

Cartesian Product

- Every row of `df1` multiplied with every row of `df2`

Cartesian Product

- Every row of `df1` multiplied with every row of `df2`
- `cart_prod = dt1[, as.list(dt2), by = "cust_id"];`
- `cart_prd = merge(df1, df2, by = NULL);`

S.No.	cust_id.x	Product	cust_id.y	State
1	1	Toaster	2	Alabama
2	2	Toaster	2	Alabama
3	3	Toaster	2	Alabama
4	4	Radio	2	Alabama
5	5	Radio	2	Alabama
6	6	Radio	2	Alabama
7	1	Toaster	4	Alabama
8	2	Toaster	4	Alabama
9	3	Toaster	4	Alabama
10	4	Radio	4	Alabama
11	5	Radio	4	Alabama
12	6	Radio	4	Alabama
13	1	Toaster	6	Ohio
14	2	Toaster	6	Ohio
15	3	Toaster	6	Ohio
16	4	Radio	6	Ohio
17	5	Radio	6	Ohio
18	6	Radio	6	Ohio

- Cartesian products are extremely slow. Never do that even on a decent sized (> 1e4 rows) dataset. Your computer will probably hang.
- Although there is no use of Cartesian products, it encapsulates all types of merges. Meaning we can extract any type of merge from a Cartesian product.
- Inner join can be extracted via
  - `idx = which(cart_prd$cust_id.x == cart_prd$cust_id.y);`
  - `cart_prd[idx,];`

Session - 5

Reflection on Session-4

Merging with more than one variable

- Most of the merging usage is with two variables: date and company name. Generate data using below:

```
date = seq.Date(as.Date("2018-01-01"), by = 1, length.out = 5);
comp = c("A", "B", "C", "D", "E");
all_pairs = merge(comp, date, by = NULL);
# sales data - 15 points
set.seed(1);
idx = sample(1:nrow(all_pairs), 15, replace = F);
df1 = data.frame(comp = all_pairs$x[idx], date = all_pairs$y[idx],
                 sales = round(runif(15, min = 1e3, max = 1e5)));
# advertising data - 12 points
idx = sample(1:nrow(all_pairs), 12, replace = F);
df2 = data.frame(comp = all_pairs$x[idx], date = all_pairs$y[idx],
                 adv = round(runif(12, min = 1e2, max = 1e4)));
```

df1

comp	date	sales
B	05-04-2018	53429
A	03-04-2018	70750
B	03-04-2018	5426
B	06-04-2018	88504
E	06-04-2018	78449
A	04-04-2018	98954
D	04-04-2018	57618
E	04-04-2018	22011
C	02-04-2018	85976
D	02-04-2018	42842
E	02-04-2018	94057
A	05-04-2018	25063
E	05-04-2018	51320
D	06-04-2018	99720
E	03-04-2018	16845

df2

comp	date	adv
C	03-04-2018	980
C	06-04-2018	6183
B	06-04-2018	8077
B	02-04-2018	241
D	02-04-2018	3642
B	04-04-2018	3878
E	02-04-2018	1501
B	03-04-2018	6727
D	06-04-2018	4259
D	03-04-2018	4434
A	06-04-2018	6745
E	03-04-2018	2208

Inner Join

- `merge(df1, df2, by = c(" comp", "date"))`

comp	date	sales	adv
B	03-04-2018	5426	6727
B	06-04-2018	88504	8077
D	02-04-2018	42842	3642
E	02-04-2018	94057	1501
D	06-04-2018	99720	4259
E	03-04-2018	16845	2208

Full Join

- `merge(df1, df2, by = c(" comp", "date"), all = T)`

comp	date	sales	adv
B	05-04-2018	53429	NA
A	03-04-2018	70750	NA
B	03-04-2018	5426	6727
B	06-04-2018	88504	8077
E	06-04-2018	78449	NA
A	04-04-2018	98954	NA
D	04-04-2018	57618	NA
E	04-04-2018	22011	NA
C	02-04-2018	85976	NA
D	02-04-2018	42842	3642
E	02-04-2018	94057	1501
A	05-04-2018	25063	NA
E	05-04-2018	51320	NA
D	06-04-2018	99720	4259
E	03-04-2018	16845	2208
C	03-04-2018	NA	980
C	06-04-2018	NA	6183
B	02-04-2018	NA	241
B	04-04-2018	NA	3878
D	03-04-2018	NA	4434
A	06-04-2018	NA	6745

Left Join

- `merge(df1, df2, by = c(" comp", "date"), all.x = T)`

comp	date	sales	adv
B	05-04-2018	53429	NA
A	03-04-2018	70750	NA
B	03-04-2018	5426	6727
B	06-04-2018	88504	8077
E	06-04-2018	78449	NA
A	04-04-2018	98954	NA
D	04-04-2018	57618	NA
E	04-04-2018	22011	NA
C	02-04-2018	85976	NA
D	02-04-2018	42842	3642
E	02-04-2018	94057	1501
A	05-04-2018	25063	NA
E	05-04-2018	51320	NA
D	06-04-2018	99720	4259
E	03-04-2018	16845	2208

Right Join

- `merge(df1, df2, by = c(" comp", "date"), all.y = T)`

comp	date	sales	adv
C	03-04-2018	NA	980
C	06-04-2018	NA	6183
B	06-04-2018	88504	8077
B	02-04-2018	NA	241
D	02-04-2018	42842	3642
B	04-04-2018	NA	3878
E	02-04-2018	94057	1501
B	03-04-2018	5426	6727
D	06-04-2018	99720	4259
D	03-04-2018	NA	4434
A	06-04-2018	NA	6745
E	03-04-2018	16845	2208

Anti Join

- Sometimes, we wish to find observations in `df1` that are not in `df2`
- `setkey(df1, comp, date); setkey(df2, comp, date);`
- `df1[!df2]` is anti-join.
  - `df2[!df1]` is also anti-join (but the other way round)
- `df1[df2]` is the observations in `df1` that are also in `df2`. This is same as `merge(df1, df2, all.y = T); # right-join`
- `df2[df1]` is left-join. `df1[df2, nomatch = 0]` is inner-join.
- There is no short-hand for full outer join and Cartesian product.
- We can't do anti-join using `merge()` commands.

Rolling Join

- What if the sales data in `df1` arrives on or after the advertising data.
  - We would then like to match sales date to any advertising date that happened on or before sales date
  - Such kind of tasks could be very complicated to using loops and if-else
  - data.table provides rolling joins
- `names(df1)[2] = "sales_date"; names(df2)[2] = "adv_date";`
- `setkey(df1, comp, sales_date); setkey(df2, comp, adv_date);`
- You can still do `df1[df2], df1[!df2],...` to get left, right, anti and inner joins
  - While joining, `sales_date` of `df1` will be merged with `adv_date` of `df2`

- `df1[df2]` looks up `df1` using `df2`, i.e. lookup `df1[, .(comp, sales_date)]` using `df2[, .(comp, adv_date)]`.
  - This is similar to `merge(df1, df2, by.x = c("comp", "sales_date"), by.y = c("comp", "adv_date"), all.y = T)`
- `df1[df2, roll = Inf]` will still lookup `df1` using `df2`, but with a caveat. The last join column (2nd key) of `df2` (`adv_date`) is rolled back to infinity until a match with the last join column (2nd key) of `df1` (`sales_date`).
  - All other keys (except the last key) are matched exactly
- `df2[df1, roll = Inf]` gives the desired match, i.e. matching advertising (`df2`) today with sales (`df1`) in future
  - `df1[df2, roll = -Inf]` will match sales (`df1`) today with advertising (`df2`) in past
  - `df2[df1, roll = 1]` will only look for 1 future sales date and stop looking
  - `df2[df1, roll = "nearest“]` matches the nearest match
    - Match advertising today with the most near sales data. Near can be past or future!
- Be cautious and verify your results when using rolling joins

```
> df1
    comp sales_date sales
 1:    A 2018-01-01 13430
 2:    A 2018-01-02 49861
 3:    A 2018-01-03 39225
 4:    A 2018-01-04 48726
 5:    B 2018-01-01 27455
 6:    B 2018-01-02 65516
 7:    B 2018-01-05 87099
 8:    C 2018-01-04 38856
 9:    C 2018-01-05 82910
10:    D 2018-01-01 22002
11:    D 2018-01-03  2326
12:    D 2018-01-04 19436
13:    E 2018-01-01 34695
14:    E 2018-01-02 60357
15:    E 2018-01-05 93536
```

```
> df2
    comp   adv_date  adv
 1:    A 2018-01-01 8626
 2:    A 2018-01-02  800
 3:    A 2018-01-05 3231
 4:    B 2018-01-02 3007
 5:    B 2018-01-03 4128
 6:    B 2018-01-05 5234
 7:    C 2018-01-01 2523
 8:    D 2018-01-02 9137
 9:    D 2018-01-04 4645
10:    E 2018-01-02 1085
11:    E 2018-01-03 6654
12:    E 2018-01-04 4437
```

← Input tables

Output tables →

```
> df1[df2, roll = -Inf]
    comp sales_date sales  adv
 1:    A 2018-01-01 13430 8626
 2:    A 2018-01-02 49861  800
 3:    A 2018-01-05    NA 3231
 4:    B 2018-01-02 65516 3007
 5:    B 2018-01-03 87099 4128
 6:    B 2018-01-05 87099 5234
 7:    C 2018-01-01 38856 2523
 8:    D 2018-01-02  2326 9137
 9:    D 2018-01-04 19436 4645
10:    E 2018-01-02 60357 1085
11:    E 2018-01-03 93536 6654
12:    E 2018-01-04 93536 4437
```

```
> df2[df1, roll = Inf]
    comp   adv_date  adv sales
 1:    A 2018-01-01 8626 13430
 2:    A 2018-01-02  800 49861
 3:    A 2018-01-03  800 39225
 4:    A 2018-01-04  800 48726
 5:    B 2018-01-01   NA 27455
 6:    B 2018-01-02 3007 65516
 7:    B 2018-01-05 5234 87099
 8:    C 2018-01-04 2523 38856
 9:    C 2018-01-05 2523 82910
10:    D 2018-01-01   NA 22002
11:    D 2018-01-03 9137  2326
12:    D 2018-01-04 4645 19436
13:    E 2018-01-01   NA 34695
14:    E 2018-01-02 1085 60357
15:    E 2018-01-05 4437 93536
```

```
> df2[df1, roll = 1]
    comp   adv_date  adv sales
 1:    A 2018-01-01 8626 13430
 2:    A 2018-01-02  800 49861
 3:    A 2018-01-03  800 39225
 4:    A 2018-01-04   NA 48726
 5:    B 2018-01-01   NA 27455
 6:    B 2018-01-02 3007 65516
 7:    B 2018-01-05 5234 87099
 8:    C 2018-01-04   NA 38856
 9:    C 2018-01-05   NA 82910
10:    D 2018-01-01   NA 22002
11:    D 2018-01-03 9137  2326
12:    D 2018-01-04 4645 19436
13:    E 2018-01-01   NA 34695
14:    E 2018-01-02 1085 60357
15:    E 2018-01-05 4437 93536
```

Reshaping (long to wide and vice-versa)

- Example data (long form):
  - `dt = fread("long_form_returns.csv", na.strings = "");`
  - Four columns: `cusip, Date, retadj, industry`
- Suppose, we want each firm (`cusip`) in a separate column
  - `wide.ret = dcast(dt, Date + industry ~ cusip, value.var = "retadj");`
    - The wide-form variable comes after ~
    - Long-form variables comes before ~
    - Finally, we want "returns" as the value of wide-from format
- How do we get the long form back from `wide.ret`?
  - ```
    dt.org = melt(wide.ret,
                  id.vars = c("Date", "industry"),
                  measure.vars = 3:ncol(wide.ret),
                  variable.name = "cusip", value.name = "retadj",
                  variable.factor = F, na.rm = T);
    ```
    ```

- Wide-form data has only one “value” variable:
  - Rows are dates and columns are company names and the matrix inside is returns. It’s very difficult to get more than one variable.
    - Try: `dt[, ret2 := retadj];`
      `dcast(dt, Date + industry ~ cusip, value.var = c("retadj", “ret2"));`
    - The column names take the form: `retadj_<cusip>` and `ret2_<cusip>`
    - N cusips, D dates and M other variables (like `retadj`, `ret2` etc)
      - long-form: N*D rows of M variables
      - wide-form: D rows of N*M variables

- If there are multiple matches in converting to wide-form
  - We can aggregate data using any function like `sum`, `mean`, …
  - `wide.ret = dcast(dt, Date ~ industry, value.var = "retadj", fun.aggregate = mean, na.rm = T);`
  - This step is irreversible in the sense that we can’t get back original data

- Most of the data will be in long-form but you should know how to quickly convert wide-form to long-form.
  - World bank provides data in wide-form

Session - 6

Reflection on Session-5

Module – III

- Mini Project
  - Study of NASA climate data
    - data.table one-liners
- Introduction to Data Analysis
  - Steps in a Data analysis project
  - Nuances: missing values, repeating data and extremes
- Plotting in R
  - legends, colors, line types, …
  - Multiple lines, multiple axes, multiple plots
- Regression Basics
  - meaning of significance and R2
  - Introduction to <u>*felm*</u> package
    - Fixed effects, error clustering

Session - 7

Reflection on Session-6

Data Analysis

Data Analysis

- Wikipedia:
  - Data analysis is a process of **inspecting**, **cleansing**, **transforming**, and **modeling** data with the goal of **discovering useful information**, informing conclusions, and supporting **decision-making**.

Data Analysis

- Wikipedia:
  - Data analysis is a process of **<u>inspecting</u>**, **<u>cleansing</u>**, **<u>transforming</u>**, and **<u>modeling</u>** data with the goal of **<u>discovering useful information</u>**, informing conclusions, and supporting **<u>decision-making</u>**.

- Key Distinctions:
  - We are not supremely interested in data collection, organization and storage tasks. Although these are important in any project!
  - Data analysis is also NOT equivalent to data science. We are chiefly interested in testing our hypothesis from the dataset.

Key Steps in a Data Analysis Project

Key Steps in a Data Analysis Project

- Collecting data
  - Tabular, qualitative, unstructured
  - Derive quantitative measures from qualitative/unstructured data

Key Steps in a Data Analysis Project

- Collecting data
  - Tabular, qualitative, unstructured
  - Derive quantitative measures from qualitative/unstructured data
- Cleaning and filtering
  - Missing data / extra data / extreme values
  - Different frequencies

Key Steps in a Data Analysis Project

- Collecting data
  - Tabular, qualitative, unstructured
  - Derive quantitative measures from qualitative/unstructured data
- Cleaning and filtering
  - Missing data / extra data / extreme values
  - Different frequencies
- Combining Data
  - Merging from different sources
  - Deriving variables from multiple sources

Key Steps in a Data Analysis Project

- Collecting data
  - Tabular, qualitative, unstructured
  - Derive quantitative measures from qualitative/unstructured data
- Cleaning and filtering
  - Missing data / extra data / extreme values
  - Different frequencies
- Combining Data
  - Merging from different sources
  - Deriving variables from multiple sources
- Exploration
  - Plots, trends, summaries and correlations

Key Steps in a Data Analysis Project

- Collecting data
  - Tabular, qualitative, unstructured
  - Derive quantitative measures from qualitative/unstructured data
- Cleaning and filtering
  - Missing data / extra data / extreme values
  - Different frequencies
- Combining Data
  - Merging from different sources
  - Deriving variables from multiple sources
- Exploration
  - Plots, trends, summaries and correlations
- Model Validation
  - Regression, out of sample tests, robustness analysis

Data Collection

- Different types of data

Data Collection

- Different types of data
  - Tabular data
    - Conventional and most common format
    - Usually fetched from popular data sources (prowess/world-bank), regulatory bodies (SEC/RBI), government websites or proprietary sources (firm's internal data)

Data Collection

- Different types of data
  - Tabular data
    - Conventional and most common format
    - Usually fetched from popular data sources (prowess/world-bank), regulatory bodies (SEC/RBI), government websites or proprietary sources (firm’s internal data)
  - Textual data
    - Getting increasingly important in Social sciences research
    - Tweets/News/Blogs/10-K reports

Data Collection

- Different types of data
  - Tabular data
    - Conventional and most common format
    - Usually fetched from popular data sources (prowess/world-bank), regulatory bodies (SEC/RBI), government websites or proprietary sources (firm’s internal data)
  - Textual data
    - Getting increasingly important in Social sciences research
    - Tweets/News/Blogs/10-K reports
  - Graphical data
    - User network (facebook)
    - Map of suppliers and customers (Samsung → apple → facebook)

Textual Data

Textual Data

- Not all text is same
    - Annual reports, tweets and blogs all use different lingo and can’t be compared or assessed uniformly
        - Each source of text will have it’s own dictionary

Textual Data

- Not all text is same
  - Annual reports, tweets and blogs all use different lingo and can’t be compared or assessed uniformly
    - Each source of text will have it’s own dictionary
- Bag of words
  - Count the number of positive, negative, hateful, pessimistic, … words
  - May wanna look at more than one word at a time
    - Better efficiency (‘good’ vs ‘so good’ vs ‘not so good’)
    - Dictionary will explode (most n-words will have 0 frequency)

Textual Data

- Not all text is same
  - Annual reports, tweets and blogs all use different lingo and can’t be compared or assessed uniformly
    - Each source of text will have it’s own dictionary
- Bag of words
  - Count the number of positive, negative, hateful, pessimistic, … words
  - May wanna look at more than one word at a time
    - Better efficiency (‘good’ vs ‘so good’ vs ‘not so good’)
    - Dictionary will explode (most n-words will have 0 frequency)
- Natural Language Processing (realm of data science)
  - Checkout the Stanford YouTube course if interested

Tabular Data

Tabular Data

- The focus of data analysis is on deriving value from data. Tabular data requires minimal efforts in deriving variables of interest.

Tabular Data

- The focus of data analysis is on deriving value from data. Tabular data requires minimal efforts in deriving variables of interest.
- Dimensions
  - Cross-section
    - Gold-standard for econometric analysis and causal inference
    - Cross-correlation, endogeneity.
  - Time-series
    - Auto-correlation, confounding effects
  - Panel (both time and firm variation)
    - Best of both worlds, difference-in-difference
  - Multi-dimensional (year, company, analyst)
    - Latest research is increasingly using bigger datasets

Data getting bigger!

Data getting bigger!

- Finance datasets tend to be notoriously huge
  - Other fields are catching up
    - Mostly with non-tabular data
  - Stock market data is almost always used in all research fields
    - No other setting provides a dynamic, efficient and fast (informationally) source of data

Data getting bigger!

- Finance datasets tend to be notoriously huge
    - Other fields are catching up
        - Mostly with non-tabular data
    - Stock market data is almost always used in all research fields
        - No other setting provides a dynamic, efficient and fast (informationally) source of data
- Things to consider
    - Can you store your dataset in RAM?
    - Will your program run in “reasonable” amount of time?
        - Parallel computing?

Data getting bigger!

- Finance datasets tend to be notoriously huge
  - Other fields are catching up
    - Mostly with non-tabular data
  - Stock market data is almost always used in all research fields
    - No other setting provides a dynamic, efficient and fast (informationally) source of data
- Things to consider
  - Can you store your dataset in RAM?
  - Will your program run in “reasonable” amount of time?
    - Parallel computing?
- Good programming practices and knowledge of space/time complexity will help to overcome issues with big data
  - Although at some point you may need to invest in hardware/cloud-computing

Cleaning Data

Cleaning Data

- Real-world data is full of holes
  - Simply deleting all missing observations will leave your analysis craving power
  - Some data is only updated very infrequently (like ratings)
    - The sensible thing is to carry forward the ratings (indefinitely or up to some period)

Cleaning Data

- Real-world data is full of holes
  - Simply deleting all missing observations will leave your analysis craving power
  - Some data is only updated very infrequently (like ratings)
    - The sensible thing is to carry forward the ratings (indefinitely or up to some period)

- Extra Data
  - What if there are multiple observations for a firm and year pair?
    - Knowledge of the subject helps to understand and take a decision
      - For restated earnings, take the most recent number
    - In case of analyst recommendations, take the average/median
    - In case of bad news (rating downgrade), take the first item as most meaningful

- Data with different frequencies?
  - Data from multiple sources rarely comes in same time durations
  - For instance how would you make sense of inflation (monthly) and GDP (quarterly/annual)?
    - use the most recent inflation OR carry-forward the GDP OR assume some process of interpolating GDP to monthly series
  - Quarterly earnings and daily stock trading?
    - Are you trying to learn about earnings quality OR are you trying to understand effect of earnings on returns?
    - The research question should guide you in choosing your method of mis-matching frequencies.
  - Twitter reaction to a new product launch and annual sales numbers?
    - Lots of tweets during launch (also maybe during sales). No need to match frequencies if the goal is to understand whether twitter reaction predicts sales.

Merging Datasets

Merging Datasets

- Each datasets has their own key (or id) variables
  - CRSP (US stock prices) uses company id (permno) and date
  - Compustat (US company financials) uses (gvkey) and fiscal/announcement date
  - IBES (analyst forecasts) uses (ticker) and forecast date
  - Macroeconomic data (like GDP, inflation, employment etc) will only have year and quarter/month information
  - Twitter data will give userd_id (twitter handle) and time of tweet

Merging Datasets

- Each datasets has their own key (or id) variables
  - CRSP (US stock prices) uses company id (permno) and date
  - Compustat (US company financials) uses (gvkey) and fiscal/announcement date
  - IBES (analyst forecasts) uses (ticker) and forecast date
  - Macroeconomic data (like GDP, inflation, employment etc) will only have year and quarter/month information
  - Twitter data will give userd_id (twitter handle) and time of tweet

- In most cases there will be a cross-sectional identifier (like company name, ticker, key) and a time-series identifier (like quarter, date or timestamp)

- There may be some rules (research practice) of how to merge
  - For instance, a company operating in 2014 will (hopefully) release its results by March 2015.
  - After the announcement of results, stock prices would reflect the new information
  - Hence, it makes sense to use the 2014 fiscal year data in stock prices from April 2015 till the next year’s (2015) announcement (again hopefully in March 2016)
  - Finance journals typically merge 2014 data from July 2015 onwards!

- There may be some rules (research practice) of how to merge
  - For instance, a company operating in 2014 will (hopefully) release its results by March 2015.
  - After the announcement of results, stock prices would reflect the new information
  - Hence, it makes sense to use the 2014 fiscal year data in stock prices from April 2015 till the next year's (2015) announcement (again hopefully in March 2016)
  - Finance journals typically merge 2014 data from July 2015 onwards!

- Tweets and product launch
  - Twitter will be most active in a short window around product launch (a new iPhone!)
  - Capture tweets around a [-2,7] day window of each launch
  - What of iPhone-12 and S-12 are launched in the same week?
    - This will complicate things and create possibilities for asking better questions!

Case in Point: Book-to-Market Ratio

- Very popular way to identify undervalued stocks (Warren Buffet). Also used to identify tech firms (TSLA has a BTM of 0.025)

Case in Point: Book-to-Market Ratio

- Very popular way to identify undervalued stocks (Warren Buffet). Also used to identify tech firms (TSLA has a BTM of 0.025)

The book value of equity is computed as follows. First, we set the book value of equity equal to stockholders' equity (SEQ) if this data item exists. This is also the data item collected by Davis et al. (2000) for the pre-1963 data. Second, if SEQ is missing but both common equity (CEQ) and the par value of preferred stock (PSTK) exist, then we set the book value of equity equal to PSTK + CEQ. Third, if the above definitions cannot be used, but the book values of total assets (AT) and total liabilities (LT) exist, then we set the book value of equity equal to AT − LT. If the book value of equity is now nonmissing, we adjust it by subtracting the redemption, liquidation, or par value of preferred stock—in that order, depending on data availability. Lastly, we add deferred taxes (TXDITC) and subtract postretirement benefits (PRBA) when these items exist.

Case in Point: Book-to-Market Ratio

- Very popular way to identify undervalued stocks (Warren Buffet). Also used to identify tech firms (TSLA has a BTM of 0.025)

The book value of equity is computed as follows. First, we set the book value of equity equal to stockholders' equity (SEQ) if this data item exists. This is also the data item collected by Davis et al. (2000) for the pre-1963 data. Second, if SEQ is missing but both common equity (CEQ) and the par value of preferred stock (PSTK) exist, then we set the book value of equity equal to PSTK + CEQ. Third, if the above definitions cannot be used, but the book values of total assets (AT) and total liabilities (LT) exist, then we set the book value of equity equal to AT − LT. If the book value of equity is now nonmissing, we adjust it by subtracting the redemption, liquidation, or par value of preferred stock—in that order, depending on data availability. Lastly, we add deferred taxes (TXDITC) and subtract postretirement benefits (PRBA) when these items exist.

```
book_val = coalesce(seq, ceq + pstk, at - lt) -
           coalesce(pstkrv, pstkl, pstk, 0) +
           coalesce(txditc, 0) -
           coalesce(prba, 0);
```

To ensure that the accounting variables are known before the returns they are used to explain, we match the accounting data for all fiscal yearends in calendar year $t-1$ (1962–1989) with the returns for July of year t to June of $t+1$. The 6-month (minimum) gap between fiscal yearend and the return

We use a firm's market equity at the end of December of year $t-1$ to compute its book-to-market, leverage, and earnings-price ratios for $t-1$, and

To ensure that the accounting variables are known before the returns they are used to explain, we match the accounting data for all fiscal yearends in calendar year $t - 1$ (1962–1989) with the returns for July of year t to June of $t + 1$. The 6-month (minimum) gap between fiscal yearend and the return

We use a firm's market equity at the end of December of year $t - 1$ to compute its book-to-market, leverage, and earnings-price ratios for $t - 1$, and

```
dt[, new_date := date + (18 - month(date))/12];

setorder(dt, cusip, new_date);
dt[, new_date := na.locf(new_date, 11), by = cusip];
```

To ensure that the accounting variables are known before the returns they are used to explain, we match the accounting data for all fiscal yearends in calendar year $t - 1$ (1962–1989) with the returns for July of year t to June of $t + 1$. The 6-month (minimum) gap between fiscal yearend and the return

We use a firm's market equity at the end of December of year $t - 1$ to compute its book-to-market, leverage, and earnings-price ratios for $t - 1$, and

```
dt[, new_date := date + (18 - month(date))/12];

setorder(dt, cusip, new_date);
dt[, new_date := na.locf(new_date, 11), by = cusip];
```

- Deleting all missing entries to book_val components would give very few data points.
  - But you may want to delete negative book values (what does that even mean?)
- Accounting data from past is merged onto stock data of future
- Lower frequency accounting data is carried forward 11 months

Exploratory Analysis

Exploratory Analysis

- Once your dataset is ready, the first step is to describe it
  - Your audience (readers of paper) should get a feel for the data before jumping into regression results
  - Does your variable has a time trend?
  - What is the cross-sectional variation (range, SD, IQR) of the derived variable?
  - Does any variable exhibit skewness?
    - Maybe necessary to scale it or take logs
    - Researchers rarely use stock price in regressions. The more common choice is returns ($\Delta P_t/P_{t-1}$) or log price.
    - book_val by itself will have a fat tail
      - Small number of firms have high book value while a large majority has very small value
      - Hence book_val is usually scaled by market equity and BTM is used

Correlations

Correlations

- Every paper has a table of correlations.
  - This is usually overlooked (by readers) but contains wealth of information
  - Captures the degree of co-movement between variables
    - Invariant to scaling

Correlations

- Every paper has a table of correlations.
  - This is usually overlooked (by readers) but contains wealth of information
  - Captures the degree of co-movement between variables
    - Invariant to scaling
- How does different variables relate to each other?
  - Before running a regression, its important to ask whether the variables of interest are even related?
  - Are there pairs of variables which are heavily correlated?
    - First evidence of multi-collinearity. If present regression coefficients will be unstable.

- How would you find correlations in panel data (with variables having time-trend)?
  - Like stock price and book value of companies
  - Both will grow over time
  - One-shot correlation will pick the time-trend rather than economic relation

- How would you find correlations in panel data (with variables having time-trend)?
  - Like stock price and book value of companies
  - Both will grow over time
  - One-shot correlation will pick the time-trend rather than economic relation
- Solution
  - De-trend variables
  - Report a series of cross-section correlation (one for each period)
    - This will give a time-series of correlation
  - But what about outliers?
    - Some firms have very small book value and huge stock price (TSLA). Others have opposite (Boeing?)
      - This is due to the fact that book value is a snapshot of past while price is an expectation of future.
  - Report correlations of ranks rather than variables (spearman rank correlation)
    - More and more papers nowadays report their analysis using ranks

- How about correlating tweets? How would you find two closest tweets from a set of million tweets?
  - Correlation is only defined for quantitative data. Thus, we can only capture correlation between some quantitative measure of tweets (like sentiment, word count, number of emojis etc)
  - Some NLP tools can find distance between words (wrod2vec)
    - NLP is exciting but out of the scope of data analysis
  - Possible approach:
    - Assign a vector to each tweet where the vector comprises of:
      - Tweets with same hashtag
      - Tweets with same company tag ($AAPL)
      - Tweet’s timestamp, country of origin, …
      - Number of words in tweets, number of characters of longest word, …
    - Then compare tweets using distance between vectors
    - More features will improve the accuracy
      - One more: number of misspelled words

Regression

Regression

- Core of any empirical social sciences paper

Regression

- Core of any empirical social sciences paper
- The best case is to find causal effects
  - Identification is very hard. Endogeneity spoils the party!
    - Omitted variable, simultaneous relations, reverse causality
    - Domain knowledge, institutional details and natural shocks can save the day.
  - Read “Mostly Harmless Econometrics” for more insight!
    - “Mastering Metrics” for a less technical approach

Regression

- Core of any empirical social sciences paper
- The best case is to find causal effects
  - Identification is very hard. Endogeneity spoils the party!
    - Omitted variable, simultaneous relations, reverse causality
    - Domain knowledge, institutional details and natural shocks can save the day.
  - Read “Mostly Harmless Econometrics” for more insight!
    - “Mastering Metrics” for a less technical approach
- Need to challenge every OLS assumption in your paper
  - And provide supporting arguments, tests, corrections, robustness checks etc to counter that
    - Are errors homoscedastic? Correlated? Clustered?
    - Are regressors exogenous? Linearly independent?

Different types of regressions

Different types of regressions

- Time-series and cross-sectional regressions
  - We do not do a lot of time-series regressions because of several problems
    - Co-integration (confounding effects)
    - Autocorrelation, non-stationarities

Different types of regressions

- Time-series and cross-sectional regressions
  - We do not do a lot of time-series regressions because of several problems
    - Co-integration (confounding effects)
    - Autocorrelation, non-stationarities
- OLS vs GLS
  - If residuals (errors) are correlated then we can do better than OLS
  - GLS imposes some structural form on errors to counter heteroscedasticity

Different types of regressions

- Time-series and cross-sectional regressions
  - We do not do a lot of time-series regressions because of several problems
    - Co-integration (confounding effects)
    - Autocorrelation, non-stationarities
- OLS vs GLS
  - If residuals (errors) are correlated then we can do better than OLS
  - GLS imposes some structural form on errors to counter heteroscedasticity
- Fixed Effects
  - Very common in literature
  - A panel regression usually includes both firm level and time level FE
    - So that firm level (and time level) idiosyncrasies do not drive main results
    - Irrespective of operational performance, almost all firms' stocks did poorly in 2008 (and early 2020).

Session - 8

Reflection on Session-7

Plotting in R

Plotting in R

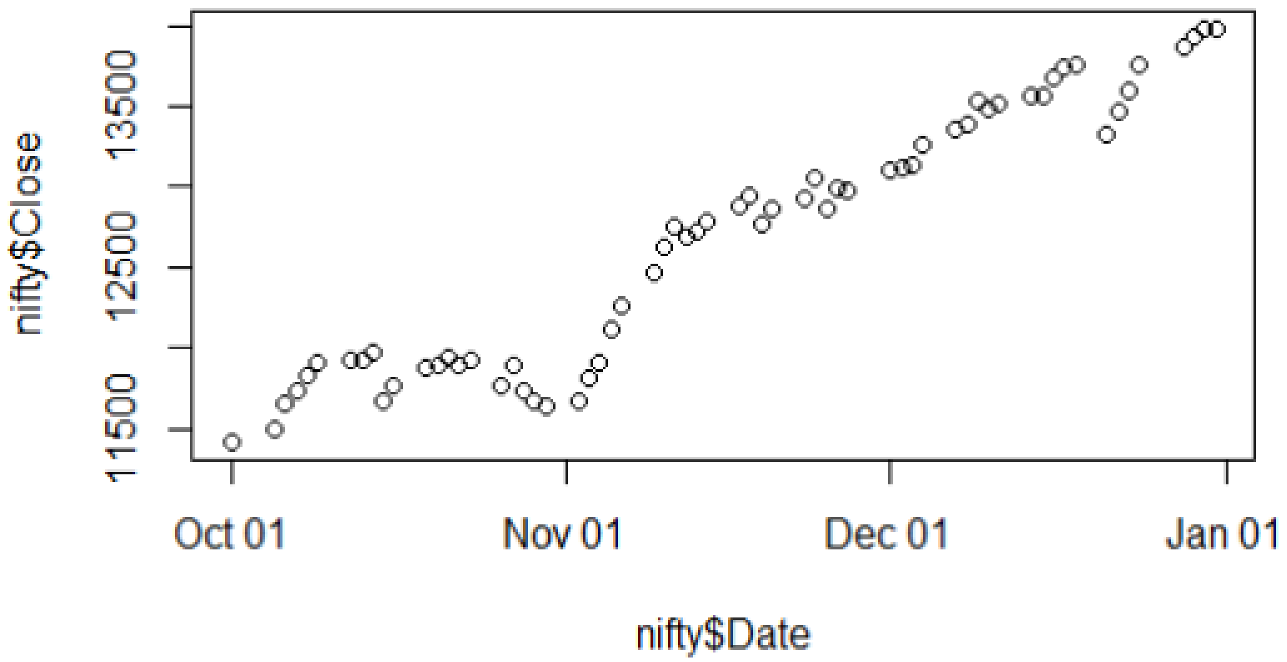
- `plot(x, y, <OPTIONS>);`
- `plot(nifty$Date, nifty$Close);`

Plotting in R

- `plot(x, y, <OPTIONS>);`
- `plot(nifty$Date, nifty$Close);`
- The below are equivalent to the plot command above:
  - `plot(Date, Close, data = nifty);`
  - `nifty[, plot(Date, Close)]; # only if nifty is a DT`

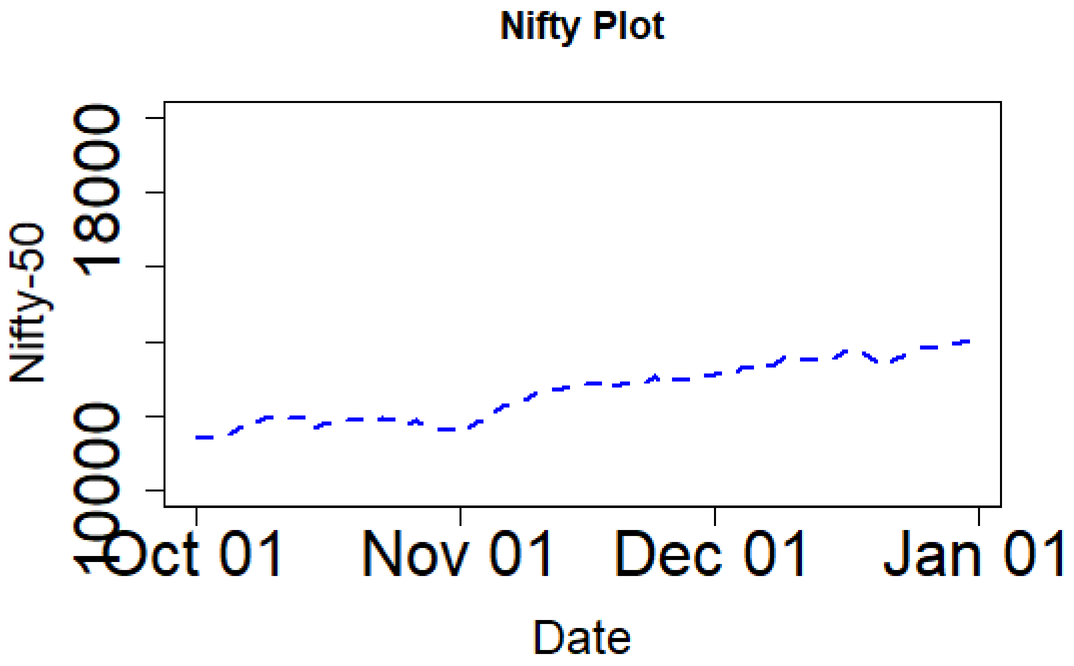
Plotting in R

- `plot(x, y, <OPTIONS>);`
- `plot(nifty$Date, nifty$Close);`
- The below are equivalent to the plot command above:
  - `plot(Date, Close, data = nifty);`
  - `nifty[, plot(Date, Close)]; # only if nifty is a DT`



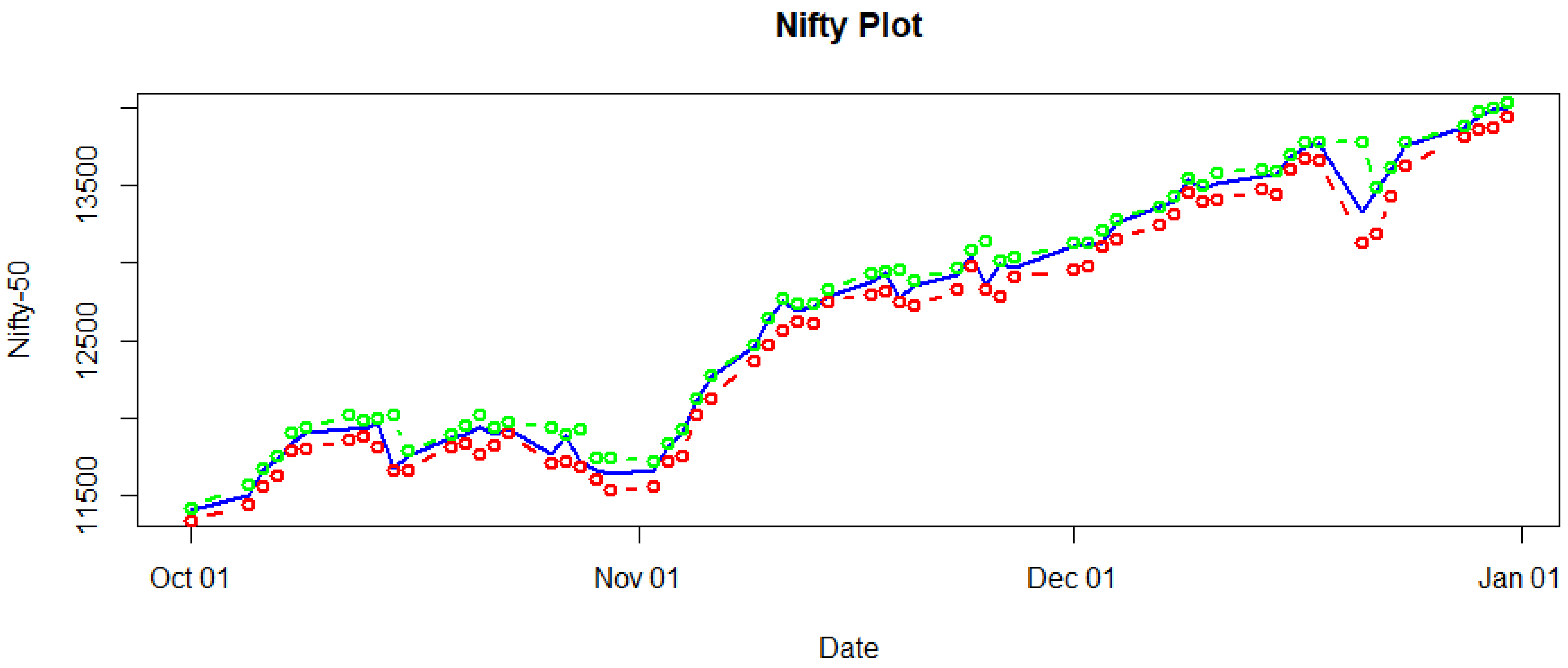
Plotting Options

```
plot(nifty$Date, nifty$Close,
     type = 'l',
     col = "blue",
     lty = 2,
     lwd = 2,
     xlab = "Date",
     ylab = "Nifty-50",
     main = "Nifty Plot",
     cex.axis = 2,
     cex.lab = 1.5,
     ylim = c(1e4, 2e4));
```



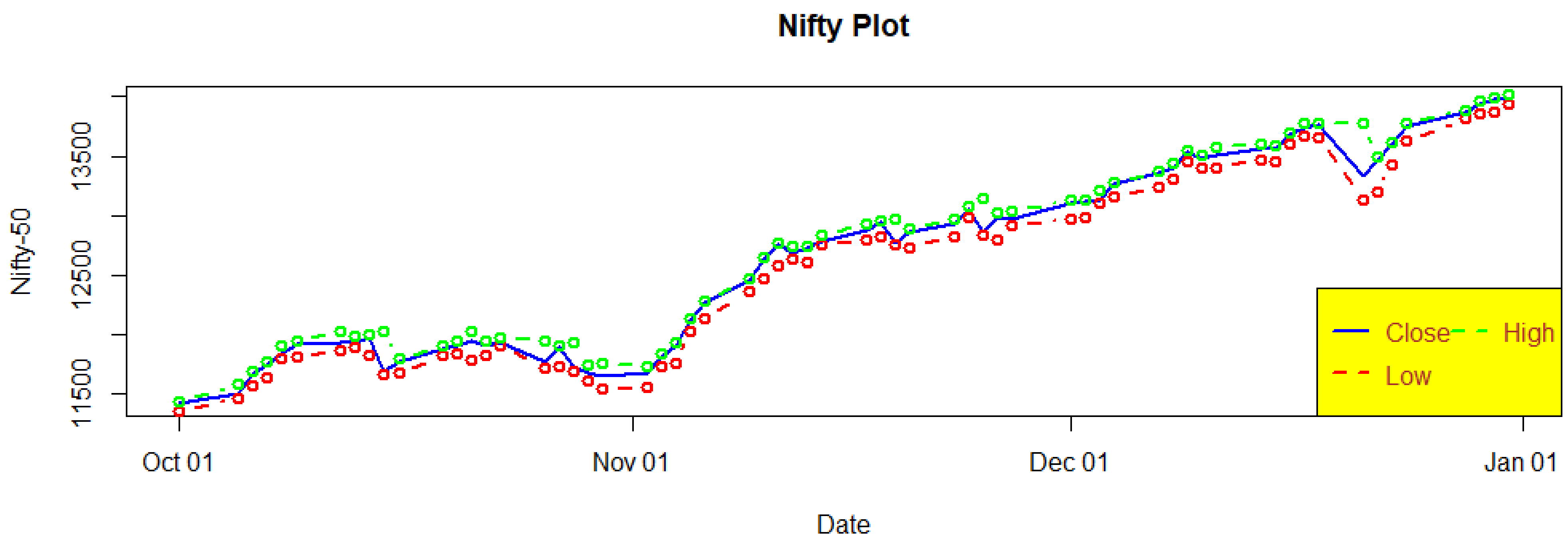
Multiple Lines

- `plot(nifty$Date, nifty$Close, type = 'l', col = "blue", lty = 1, lwd = 2, xlab = "Date", ylab = "Nifty-50", main = "Nifty Plot");`
- `lines(nifty$Date, nifty$Low, type = 'b', col = "red", lty = 2, lwd = 2);`
- `lines(nifty$Date, nifty$High, type = 'b', col = "green", lty = 2, lwd = 2);`



Legend

-
```
legend("bottomright", legend = c("Close", "Low", "High"),
            col = c("blue", "red", "green"),
            lty = c(1,2,2), lwd = 2,
            bg = "yellow", text.col = "brown", ncol = 2)
```



Multiple Axes

```
x = seq(1, 10, length.out = 1e3);
y = sin(x);
z = sqrt(x);

org_mar = par("mar");
par(mar = c(5,5,2,5)) # for extra margin on right y-axis

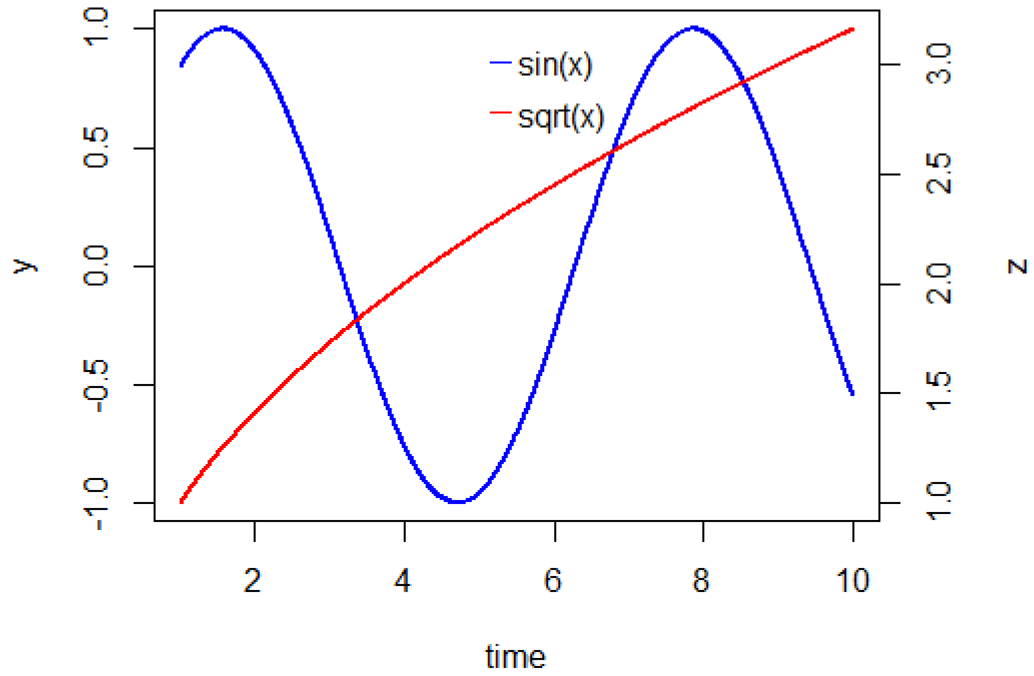
plot(x, y, type = "l", col = "blue", lwd = 2, xlab = "time", ylab = "y");
par(new = TRUE);

plot(x, z, type = "l", col = "red", lwd = 2, xlab = NA, ylab = NA, axes = F);

axis(4); # makes axis on RHS
mtext(side = 4, line = 3, "z"); # RHS text

legend("top", legend = c("sin(x)", "sqrt(x)"), lty = 1, col = c("blue", "red"), lwd = 2, bg = NULL);

par(mar = org_mar);
```



Multiple Plots

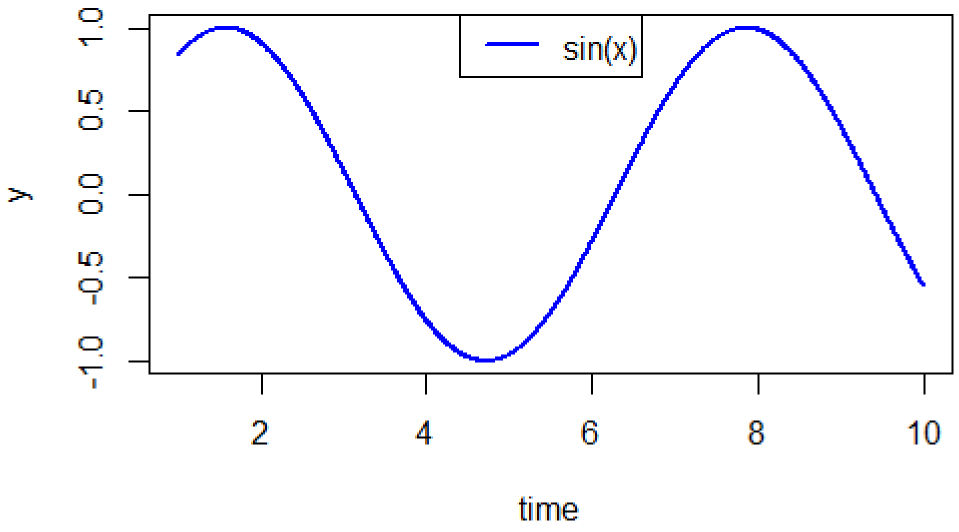
```
x = seq(1, 10, length.out = 1e3);
y = sin(x);
z = sqrt(x);

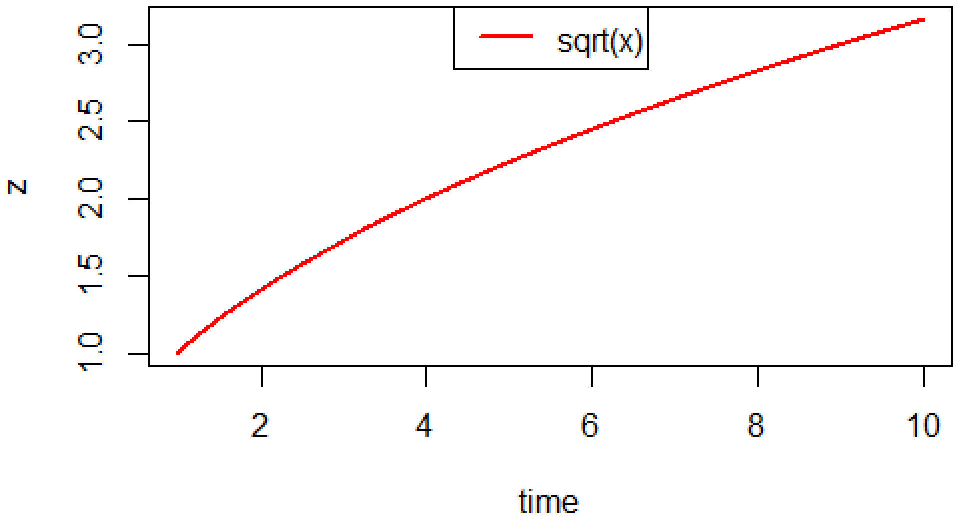
org_mfrow = par("mfrow");
par(mfrow = c(2,1));

plot(x, y, type = "l", col = "blue", lwd = 2, xlab = "time",
ylab = "y");
legend("top", legend = c("sin(x)"), lty = 1, col = c("blue"),
lwd = 2);

plot(x, z, type = "l", col = "red", lwd = 2, xlab = "time",
ylab = "z");
legend("top", legend = c("sqrt(x)"), lty = 1, col = c("red"),
lwd = 2);

par(mfrow = org_mfrow);
```





Regression Basics

Regression Basics

- Let, $Y = \beta_0 + \beta_1 \cdot X + u$ be the true model. By regressing Y on X, we hope to recover an unbiased estimate of β_1 and see how much of the variation in Y is explained by variation in X unrelated to variation in u.

Regression Basics

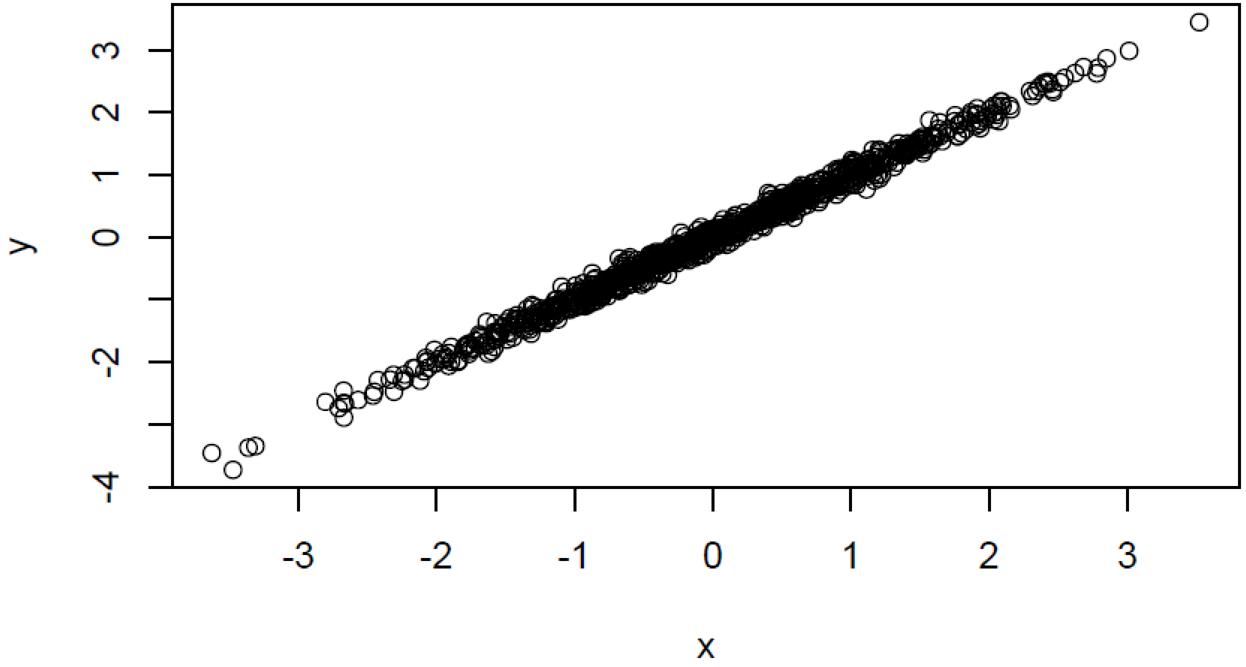
- Let, $Y = \beta_0 + \beta_1 \cdot X + u$ be the true model. By regressing Y on X, we hope to recover an unbiased estimate of β_1 and see how much of the variation in Y is explained by variation in X unrelated to variation in u.

```
n = 1000;
x = rnorm(n, 0, 1);
u = rnorm(n, 0, 0.1);
y = u + x;
plot(x,y);
```

Regression Basics

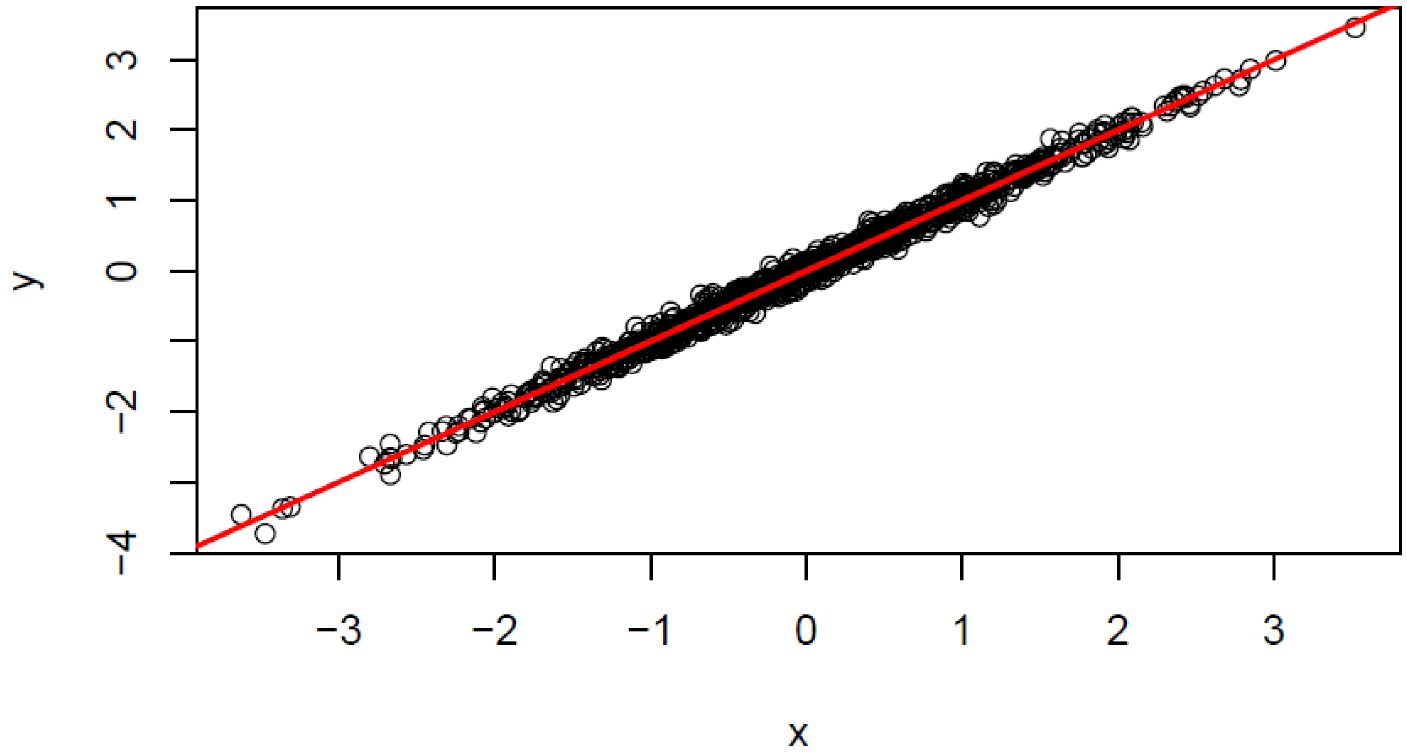
- Let, $Y = \beta_0 + \beta_1 \cdot X + u$ be the true model. By regressing Y on X, we hope to recover an unbiased estimate of β_1 and see how much of the variation in Y is explained by variation in X unrelated to variation in u.

```
n = 1000;
x = rnorm(n, 0, 1);
u = rnorm(n, 0, 0.1);
y = u + x;
plot(x,y);
```

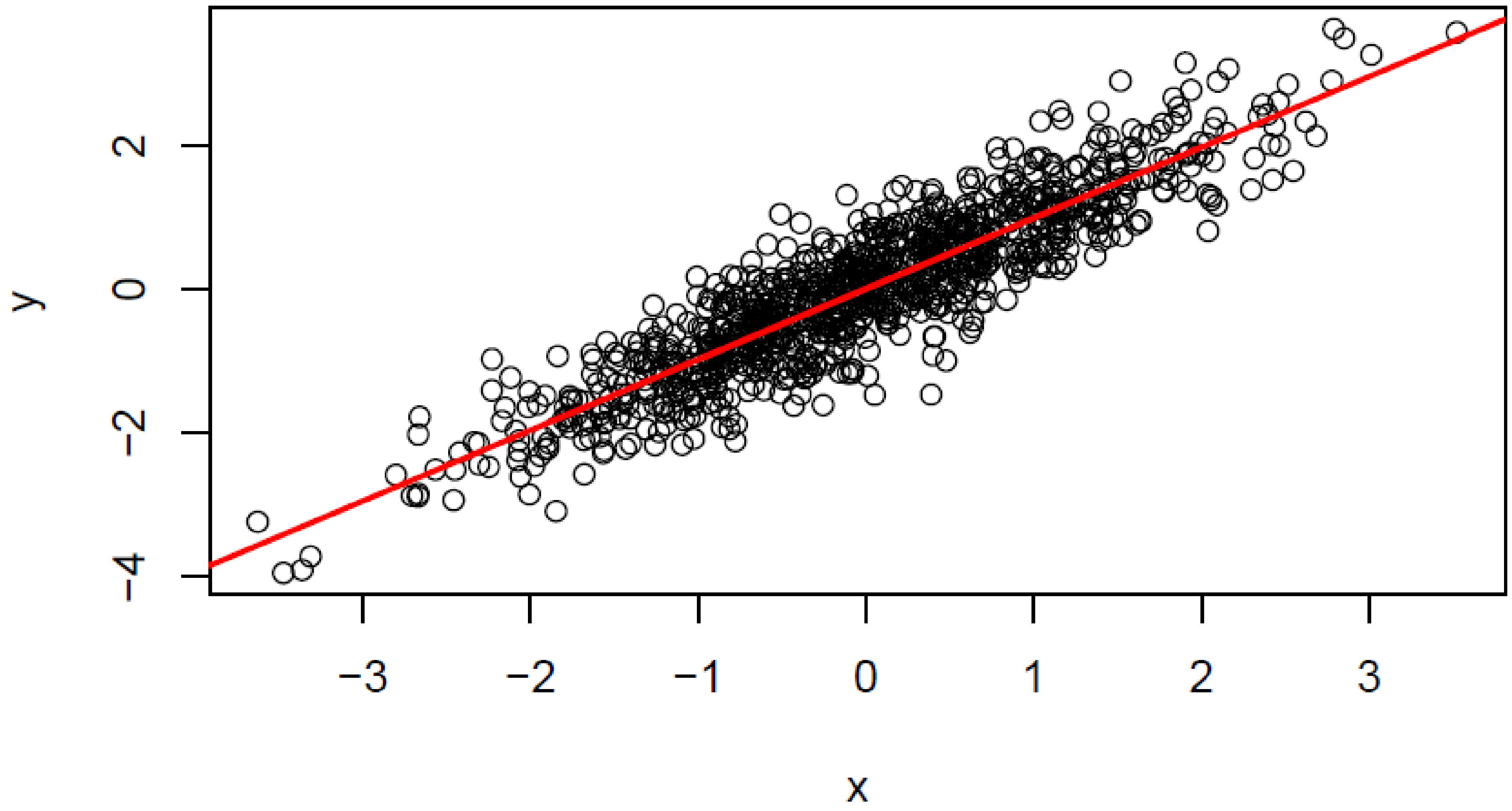


```
fit = lm(y ~ x);
summary(fit);
stargazer(fit, type = "html", out = "fit.html");
# Regression Line
abline(fit$coefficients, col = "red", lwd = 2);
```

```
fit = lm(y ~ x);
summary(fit);
stargazer(fit, type = "html", out = "fit.html");
# Regression Line
abline(fit$coefficients, col = "red", lwd = 2);
```

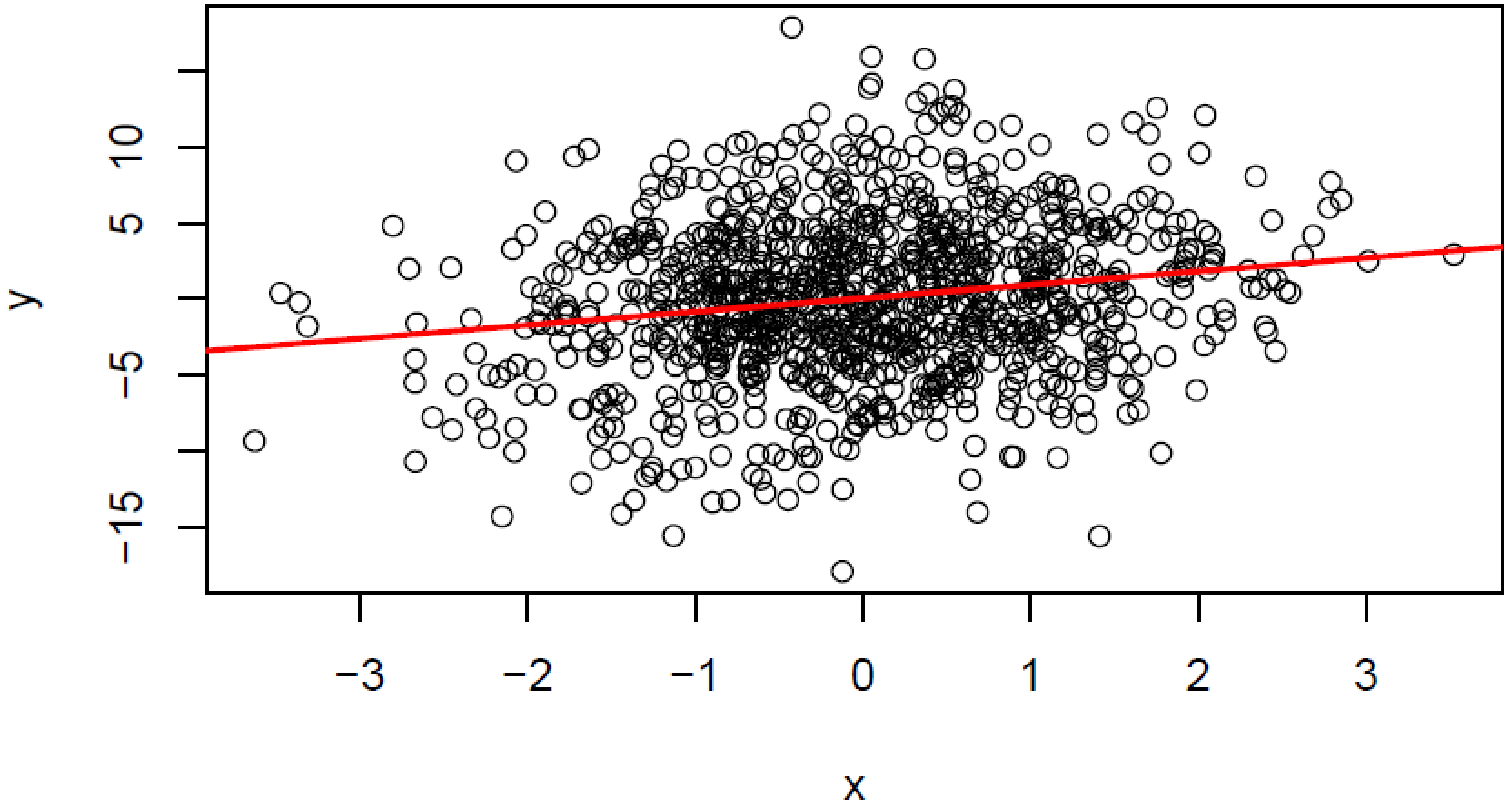


```r
u = rnorm(n, 0, 0.5);
y = u + x;
plot(x,y);
fit = lm(y ~ x);
summary(fit);
# Regression Line
abline(fit$coefficients, col = "red", lwd = 2);
```




```
u = rnorm(n, 0, 0.5);
y = u + x;
plot(x,y);
fit = lm(y ~ x);
summary(fit);
# Regression Line
abline(fit$coefficients, col = "red", lwd = 2);
```

```
u = rnorm(n, 0, 5);
y = u + x;
plot(x,y);
fit = lm(y ~ x);
summary(fit);
# Regression Line
abline(fit$coefficients, col = "red", lwd = 2);
```




```
u = rnorm(n, 0, 5);
y = u + x;
plot(x,y);
fit = lm(y ~ x);
summary(fit);
# Regression Line
abline(fit$coefficients, col = "red", lwd = 2);
```

Fixed Effects and Clustering (lfe package)

- This package is currently under repair. So we need to download an earlier version
  - rtools provides a set of tools to build and install earlier version of softwares
  - `remotes::install_version("lfe");`
    - OR try: `remotes::install_github("cran/lfe");`
  - The above will take some time

Fixed Effects and Clustering (lfe package)

- This package is currently under repair. So we need to download an earlier version
  - rtools provides a set of tools to build and install earlier version of softwares
  - `remotes::install_version("lfe");`
    - OR try: `remotes::install_github("cran/lfe");`
  - The above will take some time
- We ran basic regression as: `lm(y ~ x)`
- Suppose we wish to add fixed effects, we can do
  - `lm(y ~ x + factor(fe_1) + factor(fe_2) + ...)`
  - The above works fine for a small number of fixed effects (FE)
  - `lm()` is prohibitively slow for large number of FE and FE with large num of levels

- There are two ways to take care of FE
  - Include them in regression
  - De-mean the variables (both x and y) with respect to those FE levels
    - `lfe:felm()` is very efficient at this

- There are two ways to take care of FE
  - Include them in regression
  - De-mean the variables (both x and y) with respect to those FE levels
    - `lfe:felm()` is very efficient at this
- `lm()` offers no support for clustering of standard errors
  - Most recent research includes some form of clustering in the results
    - In panel data, you often need multi-way clustering
    - Earlier approach was to correct for heteroscedasticity and autocorrelation (HAC)
      - You might recognize terms like White (Robust) standard errors, Newey-West adjustment, etc while reading papers
  - `lfe::felm()` provides inbuilt cluster robust standard errors

- There are two ways to take care of FE
  - Include them in regression
  - De-mean the variables (both x and y) with respect to those FE levels
    - `lfe:felm()` is very efficient at this
- `lm()` offers no support for clustering of standard errors
  - Most recent research includes some form of clustering in the results
    - In panel data, you often need multi-way clustering
    - Earlier approach was to correct for heteroscedasticity and autocorrelation (HAC)
      - You might recognize terms like White (Robust) standard errors, Newey-West adjustment, etc while reading papers
  - `lfe::felm()` provides inbuilt cluster robust standard errors
- Syntax: `felm(formula, data)`
  - formula: <MODEL> | <FE> | <INSTR> | <CLUSTERS>

```
felm(   y ~ x1 + x2   | f1 + f2 |   0   |   c1 + c2     )
```

?felm (Help page of felm command)

?felm (Help page of felm command)

- The formula specification is a response variable followed by a four part formula.
  - The first part consists of ordinary covariates, the second part consists of factors to be projected out. The third part is an IV-specification. The fourth part is a cluster specification for the standard errors.
  - I.e. something like `y ~ x1 + x2 | f1 + f2 | (Q|W ~ x3+x4) | clu1 + clu2` where `y` is the response, `x1,x2` are ordinary covariates, `f1,f2` are factors to be projected out, `Q` and `W` are covariates which are instrumented by `x3` and `x4`, and `clu1,clu2` are factors to be used for computing cluster robust standard errors.
  - Parts that are not used should be specified as `0`, except if it's at the end of the formula, where they can be omitted.
    - The parentheses are needed in the third part since | has higher precedence than ~.
  - Multiple left hand sides like `y|w|x ~ x1 + x2 |f1+f2|...` are allowed.

Session - 9

Reflection on Session-8

Module - IV

- Mini Project
  - A country-wide panel of CO_2 emissions
- Miscellaneous
  - Running different types of regressions in R
    - OLS, GLS, IV, logit, GMM
  - Rmarkdown/latex for documenting your work
  - Other useful stuff!

Session - 10

Reflection on Session-9

Different Regressions in R

Different Regressions in R

- OLS is very simple
  - `fit = lm(y ~ x)`
    - You can check the summary using `summary(fit)`. This gives std errors, t-stats and R2
  - `lfe::felm(y ~ x)`

Different Regressions in R

- OLS is very simple
  - `fit = lm(y ~ x)`
    - You can check the summary using `summary(fit)`. This gives std errors, t-stats and R2
  - `lfe::felm(y ~ x)`
- GLS
  - `nlme::gls(y ~ x, correlation = C, weights = w)`
    - C is the group correlation mx, while w are heteroscedasticity weights

Different Regressions in R

- OLS is very simple
  - `fit = lm(y ~ x)`
    - You can check the summary using `summary(fit)`. This gives std errors, t-stats and R2
  - `lfe::felm(y ~ x)`
- GLS
  - `nlme::gls(y ~ x, correlation = C, weights = w)`
    - C is the group correlation mx, while w are heteroscedasticity weights
- IV
  - We can use `lfe::felm` for IV regression
    - `lfe::felm(y ~ x1 + x2 | 0 | (x3|x4 ~ w3 + w4) | 0)`
    - `AER::ivreg(y ~ x1 + x2 + x3 + x4 | x1 + x2 + w3 + w4)`
  - Requirements:
    - Instruments (w3, w4) must be correlated with endogenous variables (x3, x4)
    - Instruments (w3, w4) are unrelated to the error term, i.e. instruments affect y only through endogenous variables x3 and x4

- Logit and Probit Models
  - `glm(y ~ x1 + x2, family = binomial("logit"))`
  - `glm(y ~ x1 + x2, family = binomial("probit"))`
  - Try `?family` for more insights into how to use other models

- Logit and Probit Models
  - `glm(y ~ x1 + x2, family = binomial("logit"))`
  - `glm(y ~ x1 + x2, family = binomial("probit"))`
  - Try `?family` for more insights into how to use other models
- For more details check out `VGAM::vglm` which implements vectorised glm for several other models
- The need to perform a regression other than plain vanilla OLS rarely arises
  - Important exceptions are IV (2-SLS), FE and clustering.
  - Refer “Introductory Econometrics” by Woolridge for more theory and details on different regression models.

- Logit and Probit Models
  - `glm(y ~ x1 + x2, family = binomial("logit"))`
  - `glm(y ~ x1 + x2, family = binomial("probit"))`
  - Try `?family` for more insights into how to use other models
- For more details check out `VGAM::vglm` which implements vectorised glm for several other models
- The need to perform a regression other than plain vanilla OLS rarely arises
  - Important exceptions are IV (2-SLS), FE and clustering.
  - Refer “Introductory Econometrics” by Woolridge for more theory and details on different regression models.
- Bootstrapping
  - In some models, its impossible to accurately judge the structure of standard errors. There we can employ boot-strapping to get standard errors.
    - Sample N data points (with repetition) from your dataset and estimate the model
      - Do this 1000 (or more) times
      - The standard error of coef. errors is then your standard errors

Time-series Regressions

- The time-series regression is specified the same way as a cross-sectional regression

Time-series Regressions

- The time-series regression is specified the same way as a cross-sectional regression
- But we need to be careful about some issues
  - Non-stationarity in data
    - $1^{st}/2^{nd}$ order integration?
  - Persistent time-series
    - AR (autoregressive) and moving average (MA) parameters
  - Fit `arima(x, order)` model where `order = c(p, d, q)`
    - `p`: AR order, `d`: integration order, `q`: MA order
  - Spurious Regression
    - Will return of SBI predict return of HDFC? Reliance?
      - Nifty (or broader market) return predicts both SBI and Reliance!
  - Co-integration
    - Does India's GDP forecasts predict Nifty levels?
- Learn more at https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model

Rmarkdown/Latex for Documentation

Rmarkdown/Latex for Documentation

- Benefits of Rmarkdown/Latex over Word
  - Separation of formatting and content
    - Adding/deleting content doesn’t affect formatting
    - Changing formatting (fonts, size, columns, …) happens seamlessly
  - Managing citations, indexing (figure/table numbers)
  - Integration of your work (R) and writing (latex)
    - Reproducible research
    - Of utmost importance for an evolving research project

Rmarkdown/Latex for Documentation

- Benefits of Rmarkdown/Latex over Word
  - Separation of formatting and content
    - Adding/deleting content doesn’t affect formatting
    - Changing formatting (fonts, size, columns, …) happens seamlessly
  - Managing citations, indexing (figure/table numbers)
  - Integration of your work (R) and writing (latex)
    - Reproducible research
    - Of utmost importance for an evolving research project

- Drawbacks
  - No immediate feedback
  - Not many people use Latex outside of academia
    - Few people use latex at IIMB
    - Word has support for math symbols
  - Comments/Review/external feedback

Other Useful Stuff!

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)
- coalesce and CL0

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)
- coalesce and CL0
- time-series avg. of correlations

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)
- coalesce and CL0
- time-series avg. of correlations
- Plotting 10s/100s of variables (PLT)

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)
- coalesce and CL0
- time-series avg. of correlations
- Plotting 10s/100s of variables (PLT)
- Winsorization

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)
- coalesce and CL0
- time-series avg. of correlations
- Plotting 10s/100s of variables (PLT)
- Winsorization
- Working with ranks/quintiles/deciles

Other Useful Stuff!

- Multiple tries for a command (TRY_CATCH)
- coalesce and CL0
- time-series avg. of correlations
- Plotting 10s/100s of variables (PLT)
- Winsorization
- Working with ranks/quintiles/deciles
- Filling missing dates