Programming with Data

Session 2: R Programming (I)

Dr. Wang Jiwei
Master of Professional Accounting

Introduction to R

What is R?

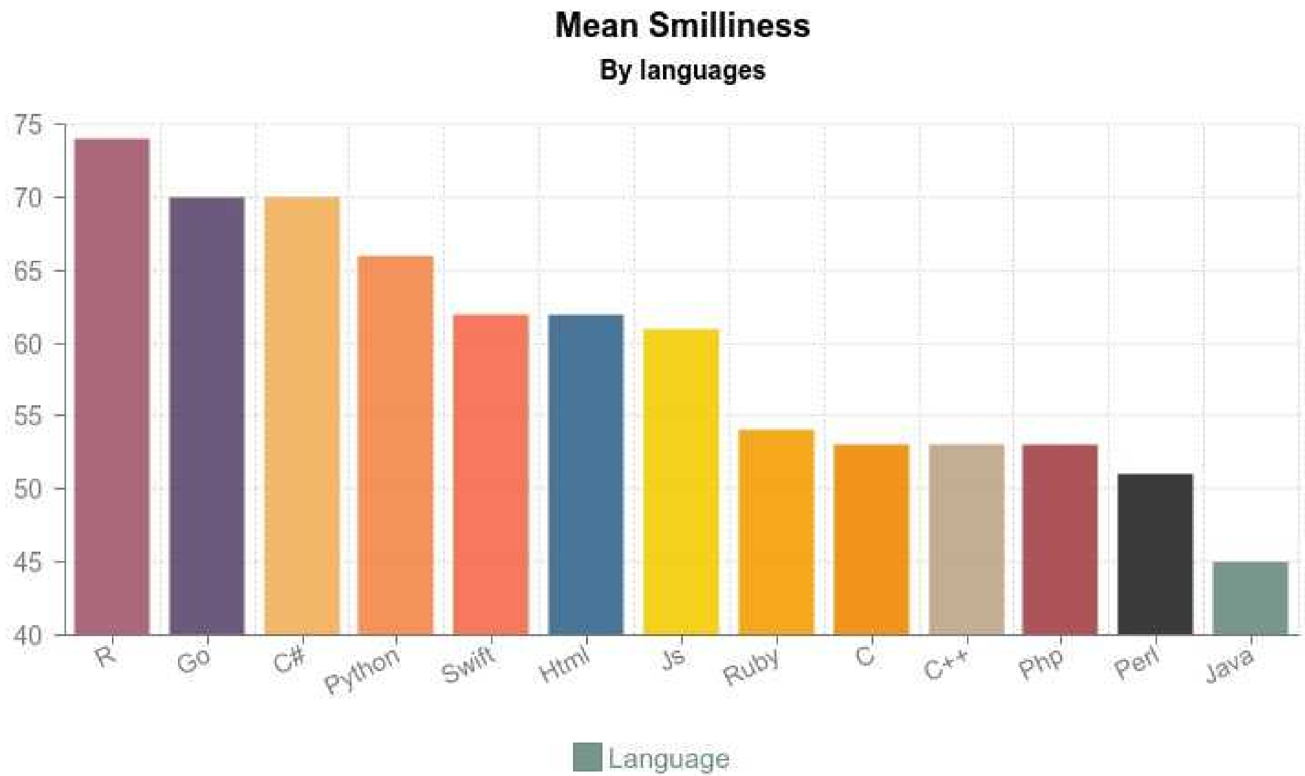
- R is free and open source
- R is a “statistical programming language”
    - Focussed on data handling, calculation, data analysis, and visualization
- R is *not* a general programming language (wikipedia)
- We will use R for all work in this course

The History of R

- 1993: developed by Ross Ihaka and Robert Gentleman at University of Auckland
- Why R? "R & R"
- R is written in C and is developed from Bell Laboratory's S language
- 2000.2.29: R 1.0.0 official release

The Happiest R

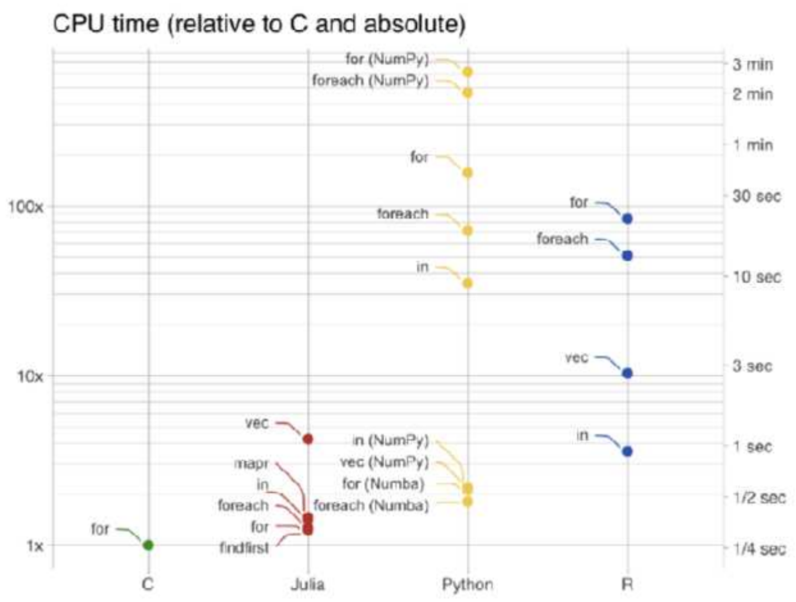
- based on programmers' pictures on GitHub



R vs Python

- Each has its own merits

R	Python
Statistical analysis with smaller dataset	Machine/Deep learning with large dataset
Data visualization	General purpose which is great for automation



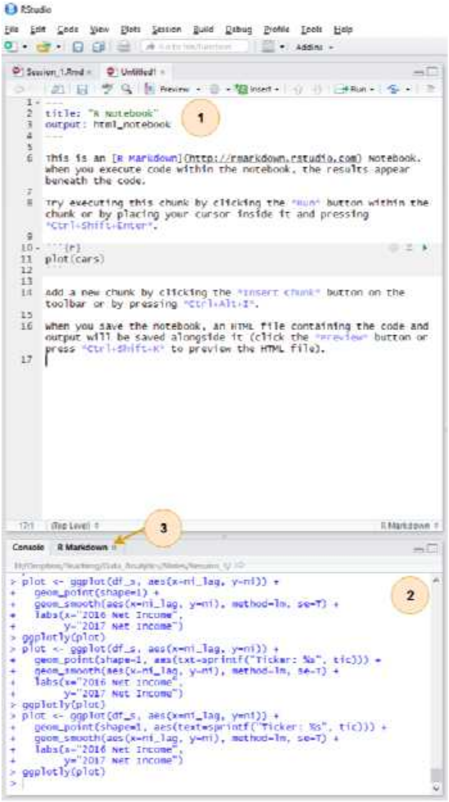
Setup

- For this class, I will assume you are using RStudio for R programming. You will need to first install R and then RStudio.
  - R Installation
  - RStudio downloads
- You will need a laptop or desktop for this
- For the most part, everything will work the same across all computer types
- Everything in these slides was tested using R version 4.1.1 (2021-08-10) Kick Things on Windows 10 x64 build 18362 😄

> R and RStudio installation path should be in English. Any non-English path may result in installation failure.

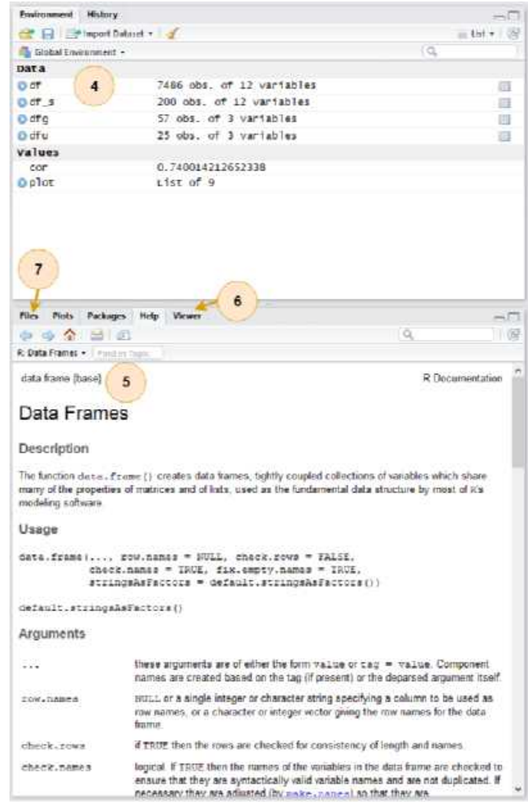
How to use RStudio

1. R markdown file
   - integrate code into reports
   - more interactive reports with analytics
   - this slides written with R Markdown using the **xaringan** package
2. Console
   - Useful for testing code and exploring your data
   - Enter your code one line at a time
3. R Markdown console
   - Shows if there are any errors when preparing your report



How to use RStudio

4. Environment - Shows all the values you have stored
5. Help - Can search documentation for instructions on how to use a function
6. Viewer - Shows any output you have at the moment.
7. Files - Shows files on your computer



Basic R commands

Arithmetic

- Anything in boxes like those on the right are R code
- The slides themselves are made in R, so you could copy and paste any code in the slides right into R to use it yourself
- Grey boxes: R Code
    - Lines starting with hash # are comments
        - They only explain what the code does
- Boxes with ##: Output

```r
# Addition uses '+'
1 + 1
```

```
## [1] 2
```

```r
# Subtraction uses '-'
2 - 1
```

```
## [1] 1
```

```r
# Multiplication uses '*'
3 * 3
```

```
## [1] 9
```

```r
# Division uses '/'
4 / 2
```

```
## [1] 2
```

Arithmetic

- Exponentiation ^
  - Write x^y as `x ^ y`
- Modulus %%
  - The remainder after division
  - Ex.: $46 \bmod 6 = 4$
    1. $6 \times 7 = 42$
    2. $46 - 42 = 4$
    3. $4 < 6$, so 4 is the remainder
- Integer division %/% (not used often)
  - Like division, but it drops any decimal

```r
# Exponentiation uses '^'
5 ^ 5
```

```
## [1] 3125
```

```r
25 ^ (1/2)
```

```
## [1] 5
```

```r
# Modulus (remainder) uses '%%'
46 %% 6
```

```
## [1] 4
```

```r
# Integer division uses '%/%'
46 %/% 6
```

```
## [1] 7
```

Variable assignment

- Variable assignment lets you give something a name
  - This lets you easily reuse it
- In R, we can name almost anything that we create
  - Values
  - Data
  - Functions, etc...
- We will name things using the `<-` or = command, with the first being preferred

```
# Store 2 in 'x' and 'x1'
x <- 2
x1 <- 2
# Check the value of x and x1
x; x1
```

```
## [1] 2
```

```
## [1] 2
```

```
# Store arithmetic in y
y <- x * 2

# Check the value of y
y
```

```
## [1] 4
```

Variable assignment

- Note that values are calculated at the time of assignment
- We previously set `y <- 2 * x`
- If we change the values of `x` and `y` remain unchanged!
- Variables: combinations of alphanumeric characters along with periods (`.`) and underscores (_), cannot start with a number or an underscore though
- Best practice: use actual names for variables instead of single letters.

```r
# Previous value of x and y
x
```

```
## [1] 2
```

```r
y
```

```
## [1] 4
```

```r
# Change x, how about y?
x <- 200

x
```

```
## [1] 200
```

```r
y
```

```
## [1] 4
```

Variable assignment

- To remove a variable, use function `rm()`
  - free up memory
- Variable names are case sensitive

```r
# Assign value to x
x <- 1

# remove variable x
rm(x)

# Check the value of x
x
```

```r
# Store 2 in 'x'
x <- 2

# Check the value of X
X
```

Application: Singtel

> Set a variable `growth` to the amount of Singtel's earnings growth percent in 2018

```r
# Data from Singtel's earnings reports, in Millions of SGD
singtel_2017 <- 3831.0
singtel_2018 <- 5430.3

# Compute growth
growth <- singtel_2018 / singtel_2017 - 1

# Check the value of growth
growth
```

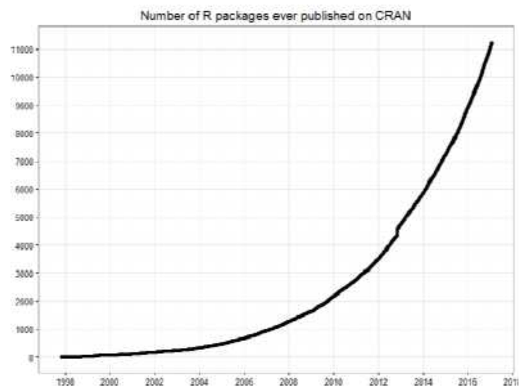
```
## [1] 0.4174628
```

Recap

- So far, we are using R as a glorified calculator
- The key to using R is that we can scale this up with little effort
  - Calculating *all* public companies' earnings growth isn't much harder than calculating Singtel's!

> Scaling this up will give use a lot more value

- We can also leverage **functions** to automate more complex operations
  - There are many functions built in, and many more freely available
- We'll also need ways to read **data files** and work with collections of numbers



Working with data in R

Data types in R

- The four main types of data in R:
- **Numeric:** Any number
  - Positive or negative
  - With or without decimals
- **Boolean/Logical:** `TRUE` or `FALSE`
  - Capitalization matters!
  - Shorthand is `T` and `F`
- **Character:** "text in quotes"
  - More difficult to work with
  - Either single or double quotes although double is recommended
- **Factor:** Converts text into numeric data
  - Categorical data for statistical analysis
  - eg, convert Male/Female into numbers to be included in statistical analysis

Data types in R

```
tech_firm <- TRUE  # boolean data
earnings <- 12662  # numeric data

class(tech_firm)
```

```
## [1] "logical"
```

```
is.logical(tech_firm)
```

```
## [1] TRUE
```

```
is.numeric(earnings)
```

```
## [1] TRUE
```

Data types in R

```
company_name <- "Google"  # character data
company_name <- 'Google' # also character data
company_name
```

```
## [1] "Google"
```

```
class(company_name)
```

```
## [1] "character"
```

```
is.character(company_name)
```

```
## [1] TRUE
```

```
nchar(company_name)
```

```
## [1] 6
```

Practice: Data types

- This practice is to make sure you understand main data types
- Do Exercise 1 on the following R practice file:
  - R Practice

Scaling up......

- We already have some data entered, but it's only a small amount
- We need to scale this up...
  - **Vectors** using `c()`!
  - **Matrices** using `matrix()`!
  - **Lists** using `list()`!
  - **Data frames** using `data.frame()`!

> Each of these is covered in the coming slides

Vectors

Vectors: What are they?

- Remember back to linear algebra...
  - Examples:

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$$

> Vector is a row (or column) of data

Vector creation

- Vectors are entered using the `c()` command
- Any data type is fine, but all elements must be the *same type*

```
company <- c("Google", "Microsoft", "Goldman")
company
```

```
## [1] "Google"    "Microsoft" "Goldman"
```

```
tech_firm <- c(TRUE, TRUE, FALSE)
tech_firm
```

```
## [1]  TRUE  TRUE FALSE
```

```
earnings <- c(12662, 21204, 4286)
earnings
```

```
## [1] 12662 21204  4286
```

Vector has no dimension

> A vector in R can be seen as a "concatenation" (in fact *c* stands for concatenate) of elements of 1 or more of the *same* data type, indexed by their positions and so no dimensions (in a spatial sense), but just a continuous index that goes from 1 to the length of the object itself.

- A vector is neither a row vector nor a column vector.
- So R will interpret a vector in whichever way makes the *matrix* product sensible.

Vector has no dimension

```
dim(earnings) = c(1, 3)   # add dimmensions
earnings
```

```
##       [,1]  [,2] [,3]
## [1,] 12662 21204 4286
```

```
dim(earnings) = c(3, 1)
earnings
```

```
##       [,1]
## [1,] 12662
## [2,] 21204
## [3,]  4286
```

```
class(earnings)
```

```
## [1] "matrix" "array"
```

```
dim(earnings) = NULL   # remove dimensions
class(earnings)
```

```
## [1] "numeric"
```

Special cases for vectors

- Counting between integers using colon and seq()
- `:`, e.g. `1:5` or `22:500`
- `seq()`, e.g. `seq(from=0, to=100, by=5)`

```
1:5
```

```
## [1] 1 2 3 4 5
```

```
seq(from=0, to=100, by=5)
```

```
##  [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
## [20]  95 100
```

↑ note that [20] means the 20th output

- Repeating something
  - `rep()`, e.g. `rep(1,times=10)` or `rep("hi",times=5)`

```
rep(1, times=10)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
rep("hi", times=5)
```

```
## [1] "hi" "hi" "hi" "hi" "hi"
```

Vector math

> Works the same as scalars (real numbers), but applies *element-wise*

- First element with first element,
- Second element with second element,
- ......

```
earnings  # previously defined
```

```
## [1] 12662 21204  4286
```

```
earnings + earnings  # Add element-wise
```

```
## [1] 25324 42408  8572
```

```
earnings * earnings  # multiply element-wise
```

```
## [1] 160326244 449609616  18369796
```

Vector math

> Can also use 1 vector and 1 scalar

- Scalar is applied to all vector elements

```
earnings + 10000  # Adding a scalar to a vector
```

```
## [1] 22662 31204 14286
```

```
10000 + earnings  # Order doesn't matter
```

```
## [1] 22662 31204 14286
```

```
earnings / 1000  # Dividing a vector by a scalar
```

```
## [1] 12.662 21.204  4.286
```

Vector math

- From linear algebra, you might remember multiplication being a bit different, as a dot product. That can be done with `%*%`

```
# Dot product: sum of product of elements
earnings %*% earnings  # returns a matrix though...
```

```
##           [,1]
## [1,] 628305656
```

```
drop(earnings %*% earnings)  # drops excess dimensions
```

```
## [1] 628305656
```

Vector math

- Other useful functions, `length()` and `sum()`:

```
length(earnings)  # returns the number of elements
```

```
## [1] 3
```

```
sum(earnings)  # returns the sum of all elements
```

```
## [1] 38152
```

Naming vectors

- Vectors allow us to include a lot of information in one object
  - It isn't easy to read though
- We can make things more readable by assigning `names()`
  - Names provide a way to easily work with and understand the data

Hard to read:

```
earnings
```

```
## [1] 12662 21204  4286
```

Easy to read:

```
names(earnings) <- c("Google",
                     "Microsoft",
                     "Goldman")
earnings
```

```
##    Google Microsoft   Goldman
##     12662     21204      4286
```

Selecting vectors

- Selecting can be done a few ways.
  - By index, such as `[1]`
  - By name, such as `["Google"]`

```
earnings[1]
```

```
## Google
##  12662
```

```
earnings["Google"]
```

```
## Google
##  12662
```

- Multiple selection:
  - `earnings[c(1,2)]`
  - `earnings[1:2]`
  - `earnings[c("Google", "Microsoft")]`

```
# Each of the above 3 is equivalent
earnings[1:2]
```

```
##    Google Microsoft
##     12662     21204
```

Combining vectors

- Combining is done using `c()`

```
c1 <- c(1, 2, 3)
c2 <- c(4, 5, 6)
c3 <- c(c1, c2)
c3
```

```
## [1] 1 2 3 4 5 6
```

Factor vectors

- *Factors* in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.
  - convert character values into numerical values
  - categorical variables in statistical modeling
- *Levels* of a factor are the unique values of that factor variable
  - R is giving each unique value of a factor a unique integer, tying it back to the character representation
  - Levels can be ordered

Factor vectors

```
x <- factor(c("High School", "College", "Masters", "PhD"))
x
```

```
## [1] High School College     Masters     PhD
## Levels: College High School Masters PhD
```

```
x <- factor(c("College", "High School", "PhD", "PhD", "Masters"),
            levels = c("High School", "College", "Masters", "PhD"),
            ordered = TRUE)
x
```

```
## [1] College     High School PhD         PhD         Masters
## Levels: High School < College < Masters < PhD
```

```
as.numeric(x)
```

```
## [1] 2 1 4 4 3
```

Missing data

- Missing data is represented by *NA* in R.
  - an element of a vector
- `is.na` tests each element of a vector for missingness
- *NULL* is the absence of anyting, ie, nothingness
  - atomical and cannot exist within a vector

```
z <- c(1, NA, 8, 3, 5)
z
```

```
## [1]  1 NA  8  3  5
```

```
is.na(z)
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

```
mean(z)
```

```
## [1] NA
```

```
mean(z, na.rm = TRUE)
```

```
## [1] 4.25
```

```
y <- c(1, NULL, 2)
y
```

```
## [1] 1 2
```

```
is.null(y)
```

```
## [1] FALSE
```

Vector example

```
# Calculating profit margin for all public US tech firms
# 715 tech firms with >1M sales in 2017
summary(earnings_2017)  # Cleaned data from Compustat, in $M USD
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4307.49   -15.98     1.84   296.84    91.36 48351.00
```

```
summary(revenue_2017)  # Cleaned data from Compustat, in $M USD
```

```
##      Min.    1st Qu.    Median      Mean   3rd Qu.      Max.
##      1.06     102.62    397.57   3023.78   1531.59 229234.00
```

```
profit_margin <- earnings_2017 / revenue_2017
summary(profit_margin)
```

```
##      Min.    1st Qu.    Median      Mean   3rd Qu.      Max.
## -13.97960  -0.10253   0.01353  -0.10967   0.09295   1.02655
```

Vector example

```r
# order() to sort and return the index for each element
# head() to output the first few elements
head(order(profit_margin))
```

```
## [1] 424 477 612 305 317 625
```

```r
# These are the worst and best profit margin firms in 2017.
profit_margin[order(profit_margin)][c(1, length(profit_margin))]
```

```
## HELIOS AND MATHESON ANALYTIC            CCUR HOLDINGS INC
##                   -13.979602                     1.026549
```

Practice: Vectors

- This practice explores the ROA of Goldman Sachs, JPMorgan, and Citigroup in 2017
- Do Exercise 2 on the following R practice file:
  - R Practice

Matrices

Matrices: what are they?

- Remember back to linear algebra...
  - Example:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

> Matrix is a rows *and* columns of data

Matrix creation

- Matrices are entered using the `matrix()` command
- Any data type is fine, but all elements must be the *same type*

```
columns <- c("Google", "Microsoft", "Goldman")
rows <- c("Earnings","Revenue")

# same: matrix(data=c(12662, 21204, 4286, 110855, 89950, 42254),ncol=3)
firm_data <- matrix(data=c(12662, 21204, 4286, 110855, 89950, 42254),
                    nrow=2)
firm_data
```

```
##       [,1]   [,2]  [,3]
## [1,] 12662   4286 89950
## [2,] 21204 110855 42254
```

Math with matrices

> Everything with matrices works just like vectors

```r
firm_data + firm_data
```

```
##        [,1]   [,2]   [,3]
## [1,] 25324   8572 179900
## [2,] 42408 221710  84508
```

```r
firm_data / 1000
```

```
##         [,1]    [,2]   [,3]
## [1,] 12.662   4.286 89.950
## [2,] 21.204 110.855 42.254
```

Math with matrices

- Matrix transposing, A^T, uses `t()`

```
firm_data_T <- t(firm_data)
firm_data_T
```

```
##       [,1]   [,2]
## [1,] 12662  21204
## [2,]  4286 110855
## [3,] 89950  42254
```

- Matrix multiplication, $A\ B$, uses `%*%`

```
firm_data %*% firm_data_T
```

```
##            [,1]        [,2]
## [1,] 8269698540  4544356878
## [2,] 4544356878 14523841157
```

> Matrix is the cornerstone of machine learning, although we don't use it much for this course

Matrix naming

- We can name matrix rows and columns, much like we named vector elements
- Use `rownames()` for rows
- Use `colnames()` for columns

```
rownames(firm_data) <- rows
colnames(firm_data) <- columns
firm_data
```

```
##          Google Microsoft Goldman
## Earnings  12662      4286   89950
## Revenue   21204    110855   42254
```

Selecting from matrices

- Select using 2 indexes instead of 1:
  - `matrix_name[rows, columns]`
  - To select all rows or columns, leave that index blanks

```r
firm_data[2, 3]
```

```
## [1] 42254
```

```r
firm_data[, c("Google","Microsoft")]
```

```
##          Google Microsoft
## Earnings  12662      4286
## Revenue   21204    110855
```

```r
firm_data[1, ]
```

```
##    Google Microsoft   Goldman
##     12662      4286     89950
```

Combining matrices

- Matrices are combined top to bottom as rows with `rbind()`

```r
# Preloaded: industry codes as indcode (vector)
# - GICS codes: 40 = Financials, 45 = Information Technology
# - https://en.wikipedia.org/wiki/Global_Industry_Classification_Standard

mat <- rbind(firm_data, indcode)  # Add a row
rownames(mat)[3] <- "Industry"  # Name the new row
mat
```

```
##          Google Microsoft Goldman
## Earnings  12662      4286   89950
## Revenue   21204    110855   42254
## Industry     45        45      40
```

Combining matrices

- Matrices are combined side-by-side as columns with `cbind()`

```
# Preloaded: JPMorgan data as jpdata (vector)

mat <- cbind(firm_data, jpdata)  # Add a column
colnames(mat)[4] <- "JPMorgan"  # Name the new column
mat
```

```
##          Google Microsoft Goldman JPMorgan
## Earnings  12662      4286   89950    17370
## Revenue   21204    110855   42254   115475
```

Lists

Lists: what are they?

- Like vectors, but with mixed types
- Generally not something we will create, often returned by analysis functions in R
  - Such as the linear regression models `lm()`

```
model <- summary(lm(earnings ~ revenue, data=tech_df))
model
```

```
##
## Call:
## lm(formula = earnings ~ revenue, data = tech_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16045.0     20.0    141.6    177.1  12104.6
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.837e+02  4.491e+01  -4.091 4.79e-05 ***
## revenue      1.589e-01  3.564e-03  44.585  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1166 on 713 degrees of freedom
## Multiple R-squared:  0.736,    Adjusted R-squared:  0.7356
## F-statistic:  1988 on 1 and 713 DF,  p-value: < 2.2e-16
```

- `str()` will tell us what's in this list

```
str(model)
```

```
## List of 11
##  $ call         : language lm(formula = earnings ~ revenue, data = tech_df)
##  $ terms        :Classes 'terms', 'formula'  language earnings ~ revenue
##   .. ..- attr(*, "variables")= language list(earnings, revenue)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "earnings" "revenue"
##   .. .. .. ..$ : chr "revenue"
##   .. ..- attr(*, "term.labels")= chr "revenue"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(earnings, revenue)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "earnings" "revenue"
##  $ residuals    : Named num [1:715] -59.7 173.8 -620.2 586.7 613.6 ...
##   ..- attr(*, "names")= chr [1:715] "40" "103" "127" "135" ...
##  $ coefficients : num [1:2, 1:4] -1.84e+02 1.59e-01 4.49e+01 3.56e-03 -4.09 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "(Intercept)" "revenue"
##   .. ..$ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
##  $ aliased      : Named logi [1:2] FALSE FALSE
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "revenue"
##  $ sigma        : num 1166
##  $ df           : int [1:3] 2 713 2
##  $ r.squared    : num 0.736
##  $ adj.r.squared: num 0.736
##  $ fstatistic   : Named num [1:3] 1988 1 713
##   ..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
```

Looking into lists

- Lists generally use double square brackets, `[[index]]`
  - Used for pulling individual elements out of a list
- `[[c()]]` will drill through lists, as opposed to pulling multiple values
- Single square brackets pull out elements as it is
- Double square brackets extract just the element
- For 1 level, we can also use `$`

```r
model["r.squared"]
```

```
## $r.squared
## [1] 0.7360059
```

```r
model[["r.squared"]]
```

```
## [1] 0.7360059
```

```r
model$r.squared
```

```
## [1] 0.7360059
```

```r
earnings["Google"]
```

```
## Google
##  12662
```

```r
earnings[["Google"]]
```

```
## [1] 12662
```

```r
#Can't use $ with vectors
```

Practice: Lists

- In this practice, we will explore lists and how to parse them
- Do Exercise 3 on the following R practice file:
  - R Practice

Data frames

Data frames: what?

- Data frames are like a hybrid between lists and matrices

Like a matrix:

- 2 dimensional like matrices
- Can access data with `[]`
- All elements in a column must be the same data type

Like a list:

- Can have different data types for different columns
- Can access data with `$`

> Think of columns as variables, rows as observations, and data frames as the Excel spreadsheet

Example of a data frame

```
library(DT) # The library is for including larger collections of data in output
datatable(tech_df[1:20, c("conm","tic","margin")],
          options = list(pageLength = 5), rownames=FALSE)
```

Show 5 entries Search:

conm	tic	margin
AVX CORP	AVX	0.00314245229040611
BK TECHNOLOGIES	BKTI	-0.0920421373270719
ADVANCED MICRO DEVICES	AMD	0.00806905610808782
ASM INTERNATIONAL NV	ASMIY	0.613509486149511
SKYWORKS SOLUTIONS INC	SWKS	0.276661006737142

Showing 1 to 5 of 20 entries Previous 1 2 3 4 Next

How to create a df?

1. On import of data, usually you will get a data frame
2. Using the `data.frame()` function

```
df <- data.frame(companyName = company,
                 earnings = earnings,
                 tech_firm = tech_firm)
df
```

```
##           companyName earnings tech_firm
## Google         Google    12662      TRUE
## Microsoft   Microsoft    21204      TRUE
## Goldman       Goldman     4286     FALSE
```

Selecting from df

- Access like a matrix

```
df[, 1]
```

```
## [1] "Google"    "Microsoft" "Goldman"
```

- Access like a list

```
df$companyName
```

```
## [1] "Google"    "Microsoft" "Goldman"
```

```
df[[1]]
```

```
## [1] "Google"    "Microsoft" "Goldman"
```

> All are relatively equivalent. Using `$` is generally most natural. Using `[,]` is good for complex references.

Making new columns

- Suggested method: use `$`

```r
df$all_zero <- 0
df$revenue <- c(110855, 89950, 42254)
df$margin <- df$earnings / df$revenue
# html_df() is a custom function for small tables
html_df(df)
```

	companyName	earnings	tech_firm	all_zero	revenue	margin
Google	Google	12662	TRUE	0	110855	0.1142213
Microsoft	Microsoft	21204	TRUE	0	89950	0.2357310
Goldman	Goldman	4286	FALSE	0	42254	0.1014342

- Alternative method: use `cbind()` just like with matrices

Sorting data frames

- To sort a *vector*, we could use the `sort()`

```
sort(df$earnings)
```

```
## [1]  4286 12662 21204
```

> THIS CAN'T SORT DATA FRAMES

- A column of a data frame is fine, but it can't sort the whole thing!

Sorting data frames

- To sort a data frame, we use the `order()` function
  - It returns the order of each element in increasing value
    - 1 is the lowest value
  - Then we pass the new order like we are selecting elements

```
ordering <- order(df$earnings)
ordering
```

```
## [1] 3 1 2
```

```
df <- df[ordering, ]
df
```

```
##           companyName earnings tech_firm all_zero revenue    margin
## Goldman       Goldman     4286     FALSE        0   42254 0.1014342
## Google         Google    12662      TRUE        0  110855 0.1142213
## Microsoft   Microsoft    21204      TRUE        0   89950 0.2357310
```

Sorting data frames

- Order can sort by multiple levels
  - `order(level1, level2, ...)`, where `level_` are vectors or df columns

```
example <- data.frame(firm=c("Google","Microsoft","Google","Microsoft"),
                      year=c(2017, 2017, 2016, 2016))
example
```

```
##        firm year
## 1    Google 2017
## 2 Microsoft 2017
## 3    Google 2016
## 4 Microsoft 2016
```

```
ordering <- order(example$firm, example$year)
example <- example[ordering, ]
example
```

```
##        firm year
## 3    Google 2016
## 1    Google 2017
## 4 Microsoft 2016
## 2 Microsoft 2017
```

Subsetting data frames

1. We can use the selecting methods from before
2. We can pass a vector of logical values telling R what to keep
   - This is pretty useful!
3. We can also use `subset()` function

```
df[df$tech_firm, ]  # Remember the comma!
```

```
##            companyName earnings tech_firm all_zero revenue    margin
## Google          Google    12662      TRUE        0  110855 0.1142213
## Microsoft    Microsoft    21204      TRUE        0   89950 0.2357310
```

```
subset(df, earnings < 20000)
```

```
##          companyName earnings tech_firm all_zero revenue    margin
## Goldman      Goldman     4286     FALSE        0   42254 0.1014342
## Google        Google    12662      TRUE        0  110855 0.1142213
```

Practice: Data frames

- This exercise explores the nature of banks' deposits
  - We will see which of Goldman, JPMorgan, and Citigroup have (since 2010):
    - The least of their assets in deposits
    - The most of their assets in deposits
- Do Exercise 4 on the following R practice file:
  - R Practice

Summary of Session 2

For next week

- continue with your Datacamp and textbook
- review today's code and pre-read next week's seminar notes
- start the **Assignment 1** which is due in two weeks.

> Tentatively, there will be the following progress assessment (30%):

1. Individual Assignment 1, on R Programming Basics
2. Individual Assignment 2, on Regressions
3. Two pop up quizzes

- Individual assignments will be in R Markdown (.rmd) file format

> All sumbissions and feedback are on eLearn. Please pay attention to academic integrity.

R Markdown: A quick guide

- Headers and subheaders start with #, ##, ..., ######
- Code blocks starts with ```` ```{r} ```` and end with ```` ``` ```` (backticks or grave accent)
  - By default, all code and figures will show up in the output
  - `echo=FALSE`: don't display code in output document
  - `results="hide"`: don't display results in output
- Inline code goes in a block starting with `` `r `` and ending with `` ` ``
- Italic font can be used by putting * or _ around *text*
- Bold font can be used by putting ** around text
  - E.g.: `**bold text**` becomes **bold text**
- To render the document, click 

- Math can be placed between `$` to use LaTeX notation
  - E.g. `$\frac{revt}{at}$` becomes $\frac{revt}{at}$
- Full equations (on their own line) can be placed between `$$`
- A block quote is prefixed with `>`
- For a complete guide, see R Studio's R Markdown::Cheat Sheet
- My slides are prepared using the xaringan template
  - The assignment is prepared using the tufte style

R Coding Style Guide

Style is subjective and arbitrary but it is important to follow a generally accepted style if you want to share code with others. I suggest the The tidyverse style guide which is also adopted by Google with some modification

- Highlights of **the tidyverse style guide**:
  - *File names*: end with .R
  - *Identifiers*: variable_name, function_name, try not to use "." as it is reserved by Base R's S3 objects
  - *Line length*: 80 characters
  - *Indentation*: two spaces, no tabs (RStudio by default converts tabs to spaces and you may change under global options)
  - *Spacing*: x = 0, not x=0, no space before a comma, but always place one after a comma
  - *Curly braces {}*: first on same line, last on own line
  - *Assignment*: use `<-`, not = nor `->`
  - *Semicolon(;)*: don't use, I used once for the interest of space
  - *return()*: Use explicit returns in functions: default function return is the last evaluated expression
  - *File paths*: use relative file path "../../filename.csv" rather than absolute path "C:/mydata/filename.csv". Backslash needs `\\`

R packages used in this slide

This slide was prepared on 2021-09-03 from Session_2s.Rmd with R version 4.1.1 (2021-08-10) Kick Things on Windows 10 x64 build 18362 😃.

The attached packages used in this slide are:

```
##         DT kableExtra      knitr 
##     "0.18"    "1.3.4"     "1.33" 
```