Book of Exploratory Data Analysis (EDA)

Berk Orbay

2022-08-01

Table of contents

Welcome

This "book", Book of Exploratory Data Analysis (EDA), is actually the organized lecture notes of the course MEF BDA 503.

Materials will progressively emerge...

Latest update in 2023-11-23

Part I

Fundamentals

This chapter covers the fundamental topics of Exploratory Data Analysis. These are

- Data Manipulation: Taking raw data and creating analyses.
- Data Visualization: Visualizing findings from the data.
- Reporting: Combining all the findings into a single cohesive report.
- Interactive Reporting (Dashboards): Self-exploring analyses and outcomes.

To achieve these skills following fundamental R packages are used.

- `dplyr` for data manipulation
- `ggplot2` for data visualization
- `quarto` for reporting (alternatively use `rmarkdown`)
- `shiny` for interactive dashboarding

Following sections provide guidance and examples on these packages in the context of Exploratory Data Analysis. There are many extensions and further packages to consider. However, in order for Fundamentals chapter to stay "fundamental" these topics are moved to Advanced chapter.

1 Base R

1.1 Brief History of R

- Created by Ross Ithaka and Robert Gentlemen of University of Auckland in 1993. It was derived from commercial **S** programming language (no kidding) which was created in 1976.
- Version 1.0.0 is released in 2000. Current version is 4.1.1.
- In 2017, CRAN (official package manager) had more than 10,000 packages. Today it has 18,214 packages on CRAN.
- Ranked as the 9th most popular language in TIOBE index as of September 2021.

1.1.1 Sources

- https://blog.revolutionanalytics.com/2020/07/the-history-of-r-updated-for-2020.html
- https://en.wikipedia.org/wiki/R_(programming_language)
- https://bookdown.org/rdpeng/rprogdatascience/history-and-overview-of-r.html
- https://cran.r-project.org/web/packages/
- https://www.tiobe.com/tiobe-index/

1.2 What is there to like R?

(Personal opinions)

- One of the two most powerful scripting languages in data analysis with Python. (Julia, first released in 2012, is an emerging third.)
- Syntax and style focused on more non-computer scientists. (Especially tidyverse)
- Excellently curated and managed package manager (CRAN).
- A powerhouse focused on data analytics. Many packages include implementations of novel research papers which cannot be found elsewhere.
- Supported by a powerful IDE (RStudio).

- Low learning curve for data analysis, visualization, publishing and interactive analysis.

Note: Python and R are not competitors. In many cases they complement each other. It is highly recommended to learn both.

1.3 What are the disadvantages of R?

- Not quite popular as Python in CS community. Support is lagging behind in some areas (especially in cloud computing) compared to Python.
- Despite a very convenient web framework (shiny), not greatly suited for scalable web applications without heavy modifications. (Still a great start)
- Parallel computing is not native in R. So, speed can be an issue.
- R keeps data in-memory.

Each disadvantage can be alleviated using a package or a solution. Its benefits far outweigh its disadvantages.

class: inverse, center, middle

1.4 Basic Features

- R is a vector based language. When you call a function or do an operation, it is usually done for every member of the vector. (It is a powerful feature which requires some time to learn.)
- Main data types are `numeric`, `character` and `logical`. But `factor`, `integer`, `date`, `dttm` (date-time) and some other types are also very common.
- Main object types are `vector`, `matrix`, `data.frame` and `list`.
- Assignment operators are "`<-`" and "`=`". Aside from rare exceptions, they are the same (`x <- 5` is the same as `x = 5`). Please be consistent in its use.

```r
x <- 5
x
```

```
[1] 5
```

- R console is completely interactive. You can run anything line by line.

1.5 Data Types

- Numeric (`double`): 1.33, 5422.22…
  - There is also `integer`: 3, 5, 6…
- Character (`character`): “a”, “course”, “pizza”…
- Boolean (`logical`): Either `TRUE` or `FALSE`.
- Date (`date`) and date-time (`dttm`): “`2020-07-28`”, “`2020-07-29 14:00:05.12 UTC+3`”
  - This part is a bit complicated with POSIXct and POSIXt types.
- Factor (`factor`): Numeric levels with labels of any kind.
  - Encountered rarely in this course.

1.6 Object Types

1.6.1 Vector

Vector is the foundation stone of R object types. A variable with a single value is called “atomic” vector.

Vectors with multiple values can be defined using `c()` (“combine”) function.

```
x <- c("a","b","c")
x
```

```
[1] "a" "b" "c"
```

A vector can have only a single data type. R conveniently converts vectors to the most appropriate data type.

```
x <- c(1,"hi",FALSE) ## Vector of numeric, character and logical values
x ## converted to all character
```

```
[1] "1"     "hi"    "FALSE"
```

1.6.2 Matrix

Matrix is simply a two dimensional special vector.

```
mat1<-matrix(1:9, ncol=3, nrow=3)
mat1
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

We can get a value from a matrix by providing its location as row/column coordinates or by simply by treating it as a vector.

```
mat1[2,2]
```

```
[1] 5
```

```
mat1[5]
```

```
[1] 5
```

1.6.3 Data Frame

Data frame object type is still two dimensional but each column can be of a different data type.

```
df1 <- data.frame(some_numbers=1:3,
                  some_names=c("Blood","Sweat","Tears"),
                  some_logical=c(TRUE,FALSE,TRUE))
df1
```

```
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            2      Sweat        FALSE
3            3      Tears         TRUE
```

Data frames are extremely powerful structures. Most of our work will be on data frames.

Note: In `dplyr` package we will see a special version of data frames: `tibble`.

1.6.4 List

Lists are like vectors but they can hold any object (including lists). You can also add names to lists.

```r
list1 <- list(data_frame = df1,matrix = mat1,vector= x)
list1
```

```
$data_frame
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            2      Sweat        FALSE
3            3      Tears         TRUE

$matrix
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$vector
[1] "1"     "hi"    "FALSE"
```

1.6.5 Functions

Functions are very useful types as they allow to run reusable code with dynamic inputs. For example, let's write a function to calculate the area of a triangle.

```r
area_of_triangle <- function(height,base_length){
  area <- height*base_length/2
  return(area) ## Return value using return command
}
## You can assign the result of a function to a variable
x <- area_of_triangle(height = 3, base_length = 4)
x
```

```
[1] 6
```

- Rule of thumb is “If you need to copy paste the same code three times, write a function instead.”
- R has thousands of predefined functions to make life easier.
- If you want to return multiple values return a list.

class: center, middle, inverse

1.7 Exercises

Complete base R document before attempting to solve these.

1.7.1 Temperature Conversion

Write a function to convert Fahrenheit to Celsius and Celsius to Fahrenheit.

(X°C $\times$ 9/5) + 32 = Y°F

```
convert_temperature <- function(x, F_to_C = TRUE){
  if(F_to_C){
    return((x-32)*5/9)
  }else{
    return(x*9/5 + 32)
  }
}
```

```
convert_temperature(30,F_to_C = FALSE)
```

```
[1] 86
```

```
convert_temperature(86,F_to_C = TRUE)
```

```
[1] 30
```

1.7.1.1 Future Value

Write a function to calculate the future value of an investment given annually compounding interest over an amount of years.

$$FV = X * (1 + i)^T$$

```r
calculate_future_value <-
function(investment, interest, duration_in_years){
  return(investment * ((1 + interest) ^ duration_in_years))
}
```

```r
## 100 units of investments 7% interest rate over 5 years
calculate_future_value(
  investment = 100, interest = 0.07, duration_in_years = 5)
```

```
[1] 140.2552
```

1.7.2 Color Hex Code

Write a function to randomly generate n color hex codes. You can use `letters` predefined vector.

```r
generate_hex_code <- function(n=1){
  hex_vec <- c(0:9,letters[1:6])
  colors <- c()
  for(i in 1:n){
    colors <- c(colors,
      paste0("#",
      paste0(sample(hex_vec,6,replace=TRUE),collapse="")))
  }
  return(colors)
}
```

```r
generate_hex_code(n=3)
```

```
[1] "#896f26" "#aea09d" "#8ba178"
```

1.7.3 Calculate Probability of Dice

Write a function which calculates the probability of getting **k** sixes in **n** throws of a die. Hint: Use binomial distribution.

```r
get_prob_dice <- function(k,n){
  combination <- factorial(n)/(factorial(k) * factorial(n-k))
  probability <- (1/6)^k * (5/6)^(n-k)
  return(combination*probability)
}
```

```r
get_prob_dice(3,5)
```

```
[1] 0.03215021
```

```r
dbinom(3,5,prob=1/6) ## or simply use dbinom
```

```
[1] 0.03215021
```

1.7.4 Rock, Scissors, Paper

Write a rock scissors paper game which computer randomly chooses

```r
rsp_game <- function(user,choices=c("rock","scissors","paper")){
  if(!(user %in% choices))
    return("Choose only rock, scissors or paper as input.")
  response <- sample(choices,1)
  if(user == response)
    return("I chose the same. Tie!")
  if((user == "rock" & response == "scissors") |
     (user == "scissors" & response == "paper") |
     (user == "paper" & response == "rock")){
    return(paste0("I chose ", response, ". You win!"))
  }else{
    return(paste0("I chose ", response, ". You lose!"))
  }
}
```

```r
rsp_game("rock")
```

```
[1] "I chose scissors. You win!"
```

Check course webpage for more exercises!

1.8 Other Links

- Base R Cheat Sheet

2 Base R in Detail

This document is more step by step. It was created in 2018, therefore there is no information about Base R pipes ("|>").

3 Briefly About R

R is a scripting language[1] with the main purpose of conducting tasks related to statistics, mainly by academics for the academics. Though, these days things got slightly out of hand and R became one of the most popular languages especially in the field of "data science". The biggest advantage of R is the huge package (R equivalent of "There is an app for that") and developer support.

Other main points to know about R are as follows.

- R is mainly based on vector operations.[2]
- R inherently does not support parallel processing. (There are solutions but still frustrating.)
- R does all the computations in the memory. A bit problematic for bigger data (>10M rows) applications, but there are solutions as well.
- Even though R packages have magnificent reporting tools (e.g. This document is prepared using R Markdown.), it is not suitable for all purpose use such as Python (especially web development).
- Package management is mainly done by CRAN repository. Though, these days there are many packages in other sources as well For instance, putting R packages on GitHub before CRAN for testing is quite popular.
- There is a Microsoft version of R with additional abilities (It supports Mac and Linux, too).

A list of resources with links and explanations will be given at the end of this document.

[1]Read as not compiled like C or C++. Line by line, you can run the code.
[2]Currently, take it as a warning of "Do not use unnecessary loops (i.e. `for`) in your code."

4 Getting Ready

- Download R from this link https://cran.r-project.org/ (latest release is 3.3.1 as of Oct 6, 2016) and install. Make sure it is working.
- Recommended to choose an editor (see R 101 below for alternatives).

5 R 101

This part lays out the very basics of R. Content is mainly about data types (numeric, character and logical), object types (vectors, matrices, lists and) and basic operations. Before starting check the following tips that can be useful.

- Commenting the code is done with `#` for each line. There is no block comment like `/* */` used in C, but there are block commment keyboard shortcuts for most editors.
- If you need information about any object just put `?` before the object. For example, try `?mean` to get information about the function `mean`.
- You will need a good code editor or an IDE (Integrated Development Environment). Most popular choice of an IDE for R is RStudio (Remember to install R first). Recommended alternatives are Atom and Sublime Text with proper add-ons.

##Value Assignment, Vectors and Data Types

Values can be defined on variables with the assignment operator `<-` or `=`.[1] For example let's assign a numeric value to the variable `x`.[2] You don't need to define a variable, assigning a value is enough.

```
x <- 522
x
```

```
[1] 522
```

Your can also assign character strings,

```
x <- "BDA503"
x
```

```
[1] "BDA503"
```

and logical. (There is also a factor type, but it is skipped for now.)

[1]You can use either or both; there is only a small difference between those two.
[2]You can specifically define integer, double or complex numeric types. For the sake of simplicity let's use only numeric, for now.

```r
x <- FALSE
x
```

```
[1] FALSE
```

5.1 Object Types

In this part, object types such as `vector`, `matrix`, `data.frame` and `list` are explained. Although this is not a complete list (e.g. `array` is another object type) and object is a more general concept, these object types are mostly sufficient at beginner and intermediate levels.

5.1.1 Vector

Most basic data structure is a vector. You can create a simple vector with `c()` (**c**ombine).

```r
x <- c(5,2,2)
x
```

```
[1] 5 2 2
```

You can change any value in a vector by defining its index. Index starts with 1.

```r
x[2] <- 7
x
```

```
[1] 5 7 2
```

You can omit a value by putting a negative index.

```r
x[-2] <- 0
x
```

```
[1] 0 7 0
```

R handles out of bounds index values and returns `NA`.

```
x[5] <- 10
x
```

```
[1]  0  7  0 NA 10
```

You can define multiple index values and define rules to choose the index.

```
x2 <- 10:19 #This is a special representation that generates a vector from a (10) to b (19
x2[c(1,3,7)] #Return 1st, 3rd and 7th values.
```

```
[1] 10 12 16
```

```
x2[(1:3)] #Return 1st to 3rd values.
```

```
[1] 10 11 12
```

```
x2[x2>15] #Return the index values where x2 > 15
```

```
[1] 16 17 18 19
```

```
x2>15
```

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

You can give names instead of index values.

```
x3<-c(1,2,3)
names(x3)<-c("a1","b2","c3")
x3
```

```
a1 b2 c3
 1  2  3
```

```r
x3["b2"]
```

```
b2
 2
```

If you try to combine different data types, R will transform them to characters or numeric.

```r
c(5,FALSE)
```

```
[1] 5 0
```

```r
c(5,FALSE,"BDA503")
```

```
[1] "5"      "FALSE"  "BDA503"
```

Mathematical operations can be easily done with vectors.

```r
vec1 <- 1:5 # This is a special representation of consecutive numbers.
vec1
```

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

```r
vec2 <- vec1 * 2
vec2
```

```
[1]  2  4  6  8 10
```

```r
vec1 + vec2
```

```
[1]  3  6  9 12 15
```

Vectors need not to be of equal size (though recommended).

```
vec1 <- 1:6
vec2 <- 3:5
vec1 + vec2
```

```
[1]  4  6  8  7  9 11
```

5.1.2 Matrix

Matrix is more like a stylized vector in a rectangular (matrix) format with some special functions.

```
mat1<-matrix(1:9, ncol=3, nrow=3)
mat1
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

You can manipulate a value of a matrix by giving its index value.

```
mat1[2,2] <- -10
mat1
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2  -10    8
[3,]    3    6    9
```

Here are some basic matrix operations.

```
mat2 <- matrix(c(0,4,1,2,0,0,0,0,1),ncol=3)
mat2
```

```
     [,1] [,2] [,3]
[1,]    0    2    0
[2,]    4    0    0
[3,]    1    0    1
```

```r
t(mat2) # Transpose of a matrix
```

```
     [,1] [,2] [,3]
[1,]    0    4    1
[2,]    2    0    0
[3,]    0    0    1
```

```r
solve(mat2) # Inverse of a matrix
```

```
     [,1]  [,2] [,3]
[1,]  0.0  0.25    0
[2,]  0.5  0.00    0
[3,]  0.0 -0.25    1
```

```r
det(mat2) # Determinant value of a matrix
```

```
[1] -8
```

```r
dim(mat2) # Dimensions of a matrix
```

```
[1] 3 3
```

```r
nrow(mat2) # Number of rows of a matrix
```

```
[1] 3
```

```r
ncol(mat2) # Number of columns of a matrix
```

```
[1] 3
```

```r
diag(mat2) # Diagonal values of a matrix
```

```
[1] 0 0 1
```

```r
eigen(mat2) # Eigenvalues and eigenvectors of a matrix
```

```
eigen() decomposition
$values
[1]  2.828427 -2.828427  1.000000

$vectors
          [,1]       [,2] [,3]
[1,] 0.5505553  0.5708950    0
[2,] 0.7786028 -0.8073674    0
[3,] 0.3011087 -0.1491200    1
```

```r
mat1 %*% mat2 # Matrix multiplication
```

```
     [,1] [,2] [,3]
[1,]   23    2    7
[2,]  -32    4    8
[3,]   33    6    9
```

You can also do vector operations with matrices.

```r
mat1 + mat2
```

```
     [,1] [,2] [,3]
[1,]    1    6    7
[2,]    6  -10    8
[3,]    4    6   10
```

```r
mat1 - mat2
```

```
     [,1] [,2] [,3]
[1,]    1    2    7
[2,]   -2  -10    8
[3,]    2    6    8
```

```r
mat1 / mat2
```

```
     [,1] [,2] [,3]
[1,]  Inf    2  Inf
[2,]  0.5 -Inf  Inf
[3,]  3.0  Inf    9
```

```r
mat1 * mat2
```

```
     [,1] [,2] [,3]
[1,]    0    8    0
[2,]    8    0    0
[3,]    3    0    9
```

You can do operations with matrices and vectors together. Then matrix is treated like a vector with the index column order (i.e. starts from top to bottom, then goes to next column).

```r
mat3 <- matrix(1:9,ncol=3)
mat3
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```r
vec <- c(0,1,0)
mat3 + vec
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    3    6    9
[3,]    3    6    9
```

```r
mat3 * vec
```

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    2    5    8
[3,]    0    0    0
```

You can name rows and columns of a matrix.

```
rownames(mat3) <- c("a","b","c")
colnames(mat3) <- c("y1","y2","y3")
mat3
```

```
  y1 y2 y3
a  1  4  7
b  2  5  8
c  3  6  9
```

5.1.3 Data Frame

Data frame is the most useful object type. Unlike matrix and vector you can define different data types for different columns.

```
df1 <- data.frame(some_numbers=1:3,some_names=c("Blood","Sweat","Tears"),some_logical=c(TR
df1
```

```
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            2      Sweat        FALSE
3            3      Tears         TRUE
```

You can see the details of an object (in this case the data frame) using `str()` function.

```
str(df1)
```

```
'data.frame':	3 obs. of  3 variables:
 $ some_numbers: int  1 2 3
 $ some_names  : chr  "Blood" "Sweat" "Tears"
 $ some_logical: logi  TRUE FALSE TRUE
```

You easily can do operations on a single column using `$`.

```
df1$some_numbers
```

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

```r
df1$some_names
```

```
[1] "Blood" "Sweat" "Tears"
```

```r
df1$some_logical
```

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

```r
df1$some_numbers <- df1$some_numbers^2
df1
```

```
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            4      Sweat        FALSE
3            9      Tears         TRUE
```

There are many example data sets in base R and packages in `data.frame` format. For instance, `EuStockMarkets` contains the closing prices of DAX (Germany), SMI (Switzerland), CAC (French), FTSE (UK) stock market indices.

```r
head(EuStockMarkets) #head() function shows the first rows of a data frame.
```

```
         DAX    SMI    CAC   FTSE
[1,] 1628.75 1678.1 1772.8 2443.6
[2,] 1613.63 1688.5 1750.5 2460.2
[3,] 1606.51 1678.6 1718.0 2448.2
[4,] 1621.04 1684.1 1708.1 2470.4
[5,] 1618.16 1686.6 1723.1 2484.7
[6,] 1610.61 1671.6 1714.3 2466.8
```

5.1.4 List

Lists can hold many objects (including lists).

```
list1 <- list(df1,mat3,vec2)
list1
```

```
[[1]]
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            4      Sweat        FALSE
3            9      Tears         TRUE

[[2]]
  y1 y2 y3
a  1  4  7
b  2  5  8
c  3  6  9

[[3]]
[1] 3 4 5
```

```
list1[[1]]
```

```
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            4      Sweat        FALSE
3            9      Tears         TRUE
```

You can name the objects and call them with the names if you like.

```
list1 <- list(some_df=df1,some_mat=mat3,vec2)
list1
```

```
$some_df
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            4      Sweat        FALSE
3            9      Tears         TRUE
```

```
$some_mat
  y1 y2 y3
a  1  4  7
b  2  5  8
c  3  6  9

[[3]]
[1] 3 4 5
```

```
list1$some_df
```

```
  some_numbers some_names some_logical
1            1      Blood         TRUE
2            4      Sweat        FALSE
3            9      Tears         TRUE
```

Lists are frequently used in functions as parameter set holders and for other purposes.

5.2 Useful Base R Functions

Remember, you can always look for help for a function using `?function_name` or `help(function_name)`. This is not an exhaustive list, there are many other fantastic functions in base R.

```
rep(x=5,times=10) #Repeat a value or a vector
```

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

```
seq(from=5,to=10,length.out=11) #Create a sequence with the given number of equidistant el
```

```
 [1]  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

```
seq(from=5,to=10,by=0.25) #Create a sequence with the given increment value
```

```
 [1]  5.00  5.25  5.50  5.75  6.00  6.25  6.50  6.75  7.00  7.25  7.50  7.75
[13]  8.00  8.25  8.50  8.75  9.00  9.25  9.50  9.75 10.00
```

```r
vec1 <- sample(x=1:10,size=10,replace=FALSE) #Pick 10 numbers randomly without replacement
vec1
```

```
 [1]  8 10  9  1  3  7  5  4  2  6
```

```r
print(vec1/2) #Print the outputs of an object. Useful for later.
```

```
 [1] 4.0 5.0 4.5 0.5 1.5 3.5 2.5 2.0 1.0 3.0
```

```r
rev(vec1) #Reverse of a vector
```

```
 [1]  6  2  4  5  7  3  1  9 10  8
```

```r
length(vec1) #Number of elements of a vector
```

```
[1] 10
```

```r
vec1 %% 2 #Mod 2 of the elements in the vector
```

```
 [1] 0 0 1 1 1 1 1 0 0 0
```

```r
min(vec1) #Minimum value of the vector
```

```
[1] 1
```

```r
max(vec1) #Maximum value of the vector
```

```
[1] 10
```

```r
factorial(vec1) #Factorial value of all elements of a vector (You can use a single value a
```

```
 [1]   40320 3628800  362880       1       6    5040     120      24       2
[10]     720
```

```r
sum(vec1) #Sum of all the values in the vector
```

```
[1] 55
```

```r
cumsum(vec1) #Cumulative sum of all the values in the vector
```

```
 [1]  8 18 27 28 31 38 43 47 49 55
```

```r
prod(vec1) #Product (multiplication) of all the values in the vector
```

```
[1] 3628800
```

```r
cumprod(vec1) #Cumulative product of all the values in the vector
```

```
 [1]       8      80     720     720    2160   15120   75600  302400  604800
[10] 3628800
```

```r
log(vec1) #Natural logarithm of the values in the vector
```

```
 [1] 2.0794415 2.3025851 2.1972246 0.0000000 1.0986123 1.9459101 1.6094379
 [8] 1.3862944 0.6931472 1.7917595
```

```r
log(vec1,base=2) #Logarithm of base 2.
```

```
 [1] 3.000000 3.321928 3.169925 0.000000 1.584963 2.807355 2.321928 2.000000
 [9] 1.000000 2.584963
```

```r
exp(vec1) #Exponential values of a vector (e=2.71...)
```

```
 [1]  2980.957987 22026.465795  8103.083928     2.718282    20.085537
 [6]  1096.633158   148.413159    54.598150     7.389056   403.428793
```

```r
vec1^2 #Power of 2
```

```
 [1]  64 100  81   1   9  49  25  16   4  36
```

```r
sqrt(vec1) #Square root
```

```
 [1] 2.828427 3.162278 3.000000 1.000000 1.732051 2.645751 2.236068 2.000000
 [9] 1.414214 2.449490
```

```r
vecx <- c(1,3,5,7) #Define another vector
vecy <- c(8,6,4,2) #Define another vector
pmax(vecx,vecy) #Maximum of each corresponding element of two (or more) vectors
```

```
[1] 8 6 5 7
```

```r
pmin(vecx,vecy) #Minimum of each corresponding element of two (or more) vectors
```

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

```r
max(vecx,vecy) #Difference between max and pmax
```

```
[1] 8
```

```r
vec1 <- c(-1,0.5,-1.2,4/3)
vec1
```

```
[1] -1.000000  0.500000 -1.200000  1.333333
```

```r
abs(vec1) #Absolute value
```

```
[1] 1.000000 0.500000 1.200000 1.333333
```

```r
round(vec1,digits = 1) #Round a value to a number of digits
```

```
[1] -1.0  0.5 -1.2  1.3
```

```r
floor(vec1) #Round down value of vector
```

```
[1] -1  0 -2  1
```

```r
ceiling(vec1) #Round up value of vector
```

```
[1] -1  1 -1  2
```

```r
round(0.5) #Interesting case about rounding. Compare with below.
```

```
[1] 0
```

```r
round(1.5) #Interesting case about rounding. Compare with above.
```

```
[1] 2
```

```r
vec_table<-sample(letters[1:5],20,replace=TRUE) #Another vector for frequency tables. lett
vec_table
```

```
 [1] "a" "d" "b" "b" "c" "e" "a" "e" "d" "e" "a" "e" "c" "a" "b" "a" "d" "a" "b"
[20] "d"
```

```r
table(vec_table) #Easily do a frequency table.
```

```
vec_table
a b c d e
6 4 2 4 4
```

5.2.1 Sorting, Ranking and Ordering

```
vec2 <- sample(x=11:20,size=10,replace=FALSE)
vec2
```

```
[1] 20 17 19 18 15 16 13 11 12 14
```

```
sort(vec2) #Sort the values in the vector
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

```
rank(vec2) #Rank of the values in the vector
```

```
[1] 10  7  9  8  5  6  3  1  2  4
```

```
order(vec2) #Returns the index values (ascending) of the sorted vector.
```

```
[1]  8  9  7 10  5  6  2  4  3  1
```

```
order(vec2,decreasing=TRUE) #Returns the index values (descending) of the sorted vector.
```

```
[1]  1  3  4  2  6  5 10  7  9  8
```

5.2.2 Logical operators

These operators return TRUE or FALSE values. They are especially useful to

```
vec1 <- 1:10
vec1
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

```
vec1 > 5 #Logical (TRUE/FALSE) result of elements greater than 5.
```

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

```
vec1[vec1 > 5]
```

```
[1]  6  7  8  9 10
```

```
vec1 >= 5 #Logical result of elements greater than or equal to 5.
```

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

```
vec1[vec1 >= 5]
```

```
[1]  5  6  7  8  9 10
```

```
vec1 < 5 #Logical result of elements less than 5.
```

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

```
vec1 <= 5 #Logical result of elements less than or equal to 5.
```

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

```
vec1 > 5 & vec1 < 9 #and (&) operator
```

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

```
vec1[vec1 > 5 & vec1 < 9]
```

```
[1] 6 7 8
```

```
vec1 < 5 | vec1 > 9 #or (|) operator
```

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

```
vec1[vec1 < 5 | vec1 > 9]
```

```
[1]  1  2  3  4 10
```

You can also do element by element comparisons of two vectors.

```
eu_df<- data.frame(EuStockMarkets[1:20,]) #Take the first 20 rows of the stock market inde
eu_df_returns <- data.frame(DAX=100*(round(eu_df$DAX[-1]/eu_df$DAX[-20],4)-1),
                            CAC=100*(round(eu_df$CAC[-1]/eu_df$CAC[-20],4)-1)) #Calculate
eu_df_returns$DAX_or_CAC <- eu_df_returns$DAX >= eu_df_returns$CAC #If the return of DAX i
eu_df_returns
```

```
     DAX   CAC DAX_or_CAC
1  -0.93 -1.26       TRUE
2  -0.44 -1.86       TRUE
3   0.90 -0.58       TRUE
4  -0.18  0.88      FALSE
5  -0.47 -0.51       TRUE
6   1.25  1.18       TRUE
7   0.58  1.32      FALSE
8  -0.29 -0.19      FALSE
9   0.64  0.02       TRUE
10  0.12  0.31      FALSE
11 -0.58 -0.24      FALSE
12 -0.51  0.15      FALSE
13 -0.52 -0.03      FALSE
14  0.20  0.34      FALSE
15  0.18 -0.04       TRUE
16  0.27  0.35      FALSE
17 -0.66  0.52      FALSE
18 -0.48  0.11      FALSE
19 -0.52 -0.70       TRUE
```

5.2.3 Statistics Functions

Some functions are predefined to facilitate statistics calculations.

```
vec1 <- sample(1:20,50,replace=TRUE) #Sample 50 numbers from values between 1 to 20
vec1
```

```
 [1]  5  2 10  8  6 13  1 15 19 18 19 11  1 20 13 20 16  3 19 14 10 10  1 11  7
[26]  8  4 10  5 10 11 18 10  8  3 16  9  5 14 18 19  9 10  4  8  4 20  7 12 17
```

```
mean(vec1) #Mean
```

```
[1] 10.62
```

```
median(vec1) #Median
```

```
[1] 10
```

```
var(vec1) #Variance
```

```
[1] 33.4649
```

```
sd(vec1) #Standard deviation
```

```
[1] 5.784885
```

```
quantile(vec1) #Quantile values
```

```
   0%   25%   50%   75%  100% 
 1.00  6.25 10.00 15.75 20.00 
```

```
quantile(vec1,0.65) #Quantile value of a specific percentage
```

```
  65% 
12.85 
```

```
summary(vec1) #An aggregate summary
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00    6.25   10.00   10.62   15.75   20.00 
```

```
cor(matrix(sample(1:20,50,replace=TRUE),ncol=5)) #Correlation matrix
```

```
             [,1]       [,2]         [,3]        [,4]        [,5]
[1,]  1.000000000  0.1083833 -0.002557377 -0.04967734  0.24495237
[2,]  0.108383315  1.0000000 -0.292261315  0.14144954 -0.10656271
[3,] -0.002557377 -0.2922613  1.000000000  0.06279868  0.04206186
[4,] -0.049677337  0.1414495  0.062798679  1.00000000 -0.32640864
[5,]  0.244952367 -0.1065627  0.042061858 -0.32640864  1.00000000
```

```
cov(matrix(sample(1:20,50,replace=TRUE),ncol=5)) #Covariance matrix
```

```
          [,1]        [,2]      [,3]       [,4]        [,5]
[1,] 49.155556  16.2888889 14.955556  -5.933333   4.7111111
[2,] 16.288889  39.6000000 -4.622222 -18.288889  -0.3777778
[3,] 14.955556  -4.6222222 18.044444   7.600000  -2.6000000
[4,] -5.933333 -18.2888889  7.600000  39.655556 -17.1555556
[5,]  4.711111  -0.3777778 -2.600000 -17.155556  27.3777778
```

There are also random number generators and functions related with densities and cdf's of different distributions. Here are the functions for normal distribution.

```
rnorm(5,mean=0,sd=1) #Generate 5 normally distributed random numbers with mean 0 and sd 1
```

```
[1]  0.5038281  0.2076145 -3.6264609 -0.2233007 -1.3242220
```

```
dnorm(x=0,mean=0,sd=1) #Density value of a point in a normal distribution with mean 0 and
```

```
[1] 0.3989423
```

```
pnorm(q=1.96,mean=0,sd=1) #Cumulative distribution value of a point in a normal distributi
```

```
[1] 0.9750021
```

```
qnorm(p=0.975,mean=0,sd=1) #Quantile value of a point in a normal distribution with mean 0
```

```
[1] 1.959964
```

Other distributions include `dpois` (poisson), `dbinom` (binomial), `dgeom` (geometric), `dunif` (uniform), `dgamma` (gamma), `dexp` (exponential), `dchisq` (chi-squared), `dt` (t distribution), `df` (F distribution), `dcauchy` (cauchy),`dnbinom` (negative binomial), `dhyper` (hypergeometric), `dlnorm` (lognormal), `dbeta` (beta), `dlogis` (logistic) and `dweibull` (weibull) with the same format (e.g. `rpois` generates random poisson numbers).

5.2.3.1 Random Number Generation

Tip: For reproducibility use `set.seed`. It will set the randomness seed to a value and random number generation will be the same for (almost) everyone.

```
set.seed(522)
rnorm(10)
```

```
 [1]  0.52028245  0.75354770 -0.80932517 -0.42112173  0.08458416  1.80153605
 [7]  1.25071091 -0.31097287  1.16377544 -0.67728655
```

Let's run it a second time by resetting the seed. The output will be the same.

```
set.seed(522)
rnorm(10)
```

```
 [1]  0.52028245  0.75354770 -0.80932517 -0.42112173  0.08458416  1.80153605
 [7]  1.25071091 -0.31097287  1.16377544 -0.67728655
```

See, the same output happens when randomness seed is restarted at the same value.

5.2.4 Conversion between data and object types

You can convert numeric to character, logical to numeric using functions starting with `as.` and check the type of the object with `is.` or `typeof()`.

```r
vec1<-c(1,2,3,4)
is.numeric(vec1) #Is the vector numeric?
```

```
[1] TRUE
```

```r
as.character(vec1) #Make the vector character?
```

```
[1] "1" "2" "3" "4"
```

```r
typeof(vec1) #What is the type?
```

```
[1] "double"
```

```r
vec2<-c("a","b","c","d")
typeof(vec2)
```

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

```r
as.numeric(vec2) # oops
```

```
Warning: NAs introduced by coercion
```

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

```r
vec3<-c(TRUE,FALSE,TRUE,FALSE)
is.logical(vec3)
```

```
[1] TRUE
```

```r
as.numeric(vec3)
```

```
[1] 1 0 1 0
```

```
as.character(vec3)
```

```
[1] "TRUE"  "FALSE" "TRUE"  "FALSE"
```

```
vec3*1 #Convert to numeric with multiplication
```

```
[1] 1 0 1 0
```

```
df1<-data.frame(a=c(1,2,3),b=c(4,5,6),c=c(7,8,9))
as.matrix(df1) #Convert to matrix
```

```
     a b c
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

```
mat1 <- matrix(1:9,ncol=3)
as.data.frame(mat1)
```

```
  V1 V2 V3
1  1  4  7
2  2  5  8
3  3  6  9
```

5.2.5 String Manipulation

```
strvec1<-c("BDA503","FE511","IE422")
grep("ETM",strvec1) #Index values of character strings including FE
```

```
integer(0)
```

```
grepl("ETM",strvec1) #TRUE FALSE statements of character strings including FE
```

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

```r
gsub("ETM","IE",strvec1) #Replacing strings
```

```
[1] "BDA503" "FE511"  "IE422"
```

```r
nchar(strvec1) #Return number of characters in string
```

```
[1] 6 5 5
```

```r
substr(strvec1,start=1,stop=2) #Trim the string from start to stop
```

```
[1] "BD" "FE" "IE"
```

```r
paste("ETM","522",sep="-") #Concatenate two strings with a separator.
```

```
[1] "ETM-522"
```

```r
paste0("ETM","522") #Concatenate two strings without a separator, equivalent of paste(.,se
```

```
[1] "ETM522"
```

```r
paste(strvec1,collapse="+") #Concatenate elements of a vector with a collapse character.
```

```
[1] "BDA503+FE511+IE422"
```

5.3 Conditionals (If-Else)

Conditionals are straightforward. If a statement returns `TRUE`, then the code chunk defined by the brackets are executed.

```r
course_name <- "BDA503" #Define the course name.

if(course_name=="BDA503"){ #If the course name is FE522.
  print("Correct course.")
}
```

```
[1] "Correct course."
```

It is possible to execute some other code chunk if the statement is `FALSE` with `else` and add other conditionals using `else if`.

```r
course_name <- "FE511" #Define the course name.

if(course_name=="BDA503"){ #If the course name is FE522.
  print("Correct course.")
}else if(grepl("ETM",course_name)){ #If the course name include FE but it is not FE522.
  print("Wrong course but close.")
}else{ #If none of the above
  print("Wrong course.")
}
```

```
[1] "Wrong course."
```

`if` conditional statements accept only one value. If you want to check for all elements in a vector use `ifelse()`.

```r
course_name<-c("BDA503","FE511","IE422")
ifelse(course_name=="BDA503","Correct Course","Wrong Course")
```

```
[1] "Correct Course" "Wrong Course"   "Wrong Course"
```

5.4 Loops

Although you are warned that R works slowly with loops (especially loops within loops), it is usually inevitable to use the loops.

5.4.1 For

For loops consist of a loop variable and a scope.

```
val<-2
for(i in 1:3){ #Define the loop variable and scope
  print(val^i)
}
```

```
[1] 2
[1] 4
[1] 8
```

Scope does not need to be numbers. For returns whatever in the scope in index order

```
for(i in c("BDA503","FE511","IE422")){
  print(i)
}
```

```
[1] "BDA503"
[1] "FE511"
[1] "IE422"
```

5.4.2 While

While is a less frequently used loop type. It repeats the code while a condition is met. It first checks the condition. When it is not satisfied, it skips the code chunk.

```
x <- 0
while(x < 3){
  x <- x+1
  print(paste0("x is ",x," x is not at the desired level. Desired level is above 3."))
}
```

```
[1] "x is 1 x is not at the desired level. Desired level is above 3."
[1] "x is 2 x is not at the desired level. Desired level is above 3."
[1] "x is 3 x is not at the desired level. Desired level is above 3."
```

5.5 Functions

R lets you to define functions easily, with a flexible format. Here are some examples.

```
fun1<-function(par1="This is a default value"){
  print(par1)
}
```

If there is a default value defined on the function you do not need to enter any value if you are comfortable with.

```
fun1()
```

```
[1] "This is a default value"
```

You can change the parameters when you call the function.

```
fun1(par1="Congratulations, you changed the parameter.")
```

```
[1] "Congratulations, you changed the parameter."
```

If you are careful about the order of your entered parameters, you do not need to write the parameter name.

```
fun1("Wow you do it like a pro without parameter names!")
```

```
[1] "Wow you do it like a pro without parameter names!"
```

Here is another simple example. Let's calculate the future value of an initial investment compounded interest.

```
calc_future_value<-function(present_value,interest_rate,years){
  return(present_value*(1+interest_rate)^years)
}
calc_future_value(100,0.05,5)
```

```
[1] 127.6282
```

Put a technical analysis.

5.6 Input Output (I/O) Operations

Reading from and writing to data files will be unavoidable at some point. While it is useful to know the fundamental functions, I/O operations usually require experience. In other words, you will face many challenges to read a table from an excel file or writing outputs to txt files. Though, it gets easier

Frequently use the help of these functions to understand their inner workings. For `xlsx` files and other data types (e.g. JSON, SQL) there are packages.

```
setwd("~/some_path") #Set working directory path.
getwd() #Get the working directory path.
scan(file="some_data_file.txt") #Read data from file.
read.table(file="some_data_file.csv") #Read xls or csv files but not xlsx files. You will
source("path_to_some_r_file/some_r_file.r")
write("writing_something",file="some_document_file.txt")
write.table() #Writing to csv or xls. Similar logic to to read.table with opposite functio
file.choose() #Manually choosing a file from computer. You can use it like read.table(file
dir(path="some_path") #Files in the path directory.
```

Important: Defining paths in R can be different in Windows and Mac. See this link for more detail.

```
dir("C:/Desktop/") #Windows style 1
dir("C:\\Desktop\\") #Windows style 2
dir("~/Documents/") #Mac and Linux style. Might work for Windows too.
```

Tip: Sometimes, R reads columns containing characters as `factor` data type. It is not covered in this tutorial and it is tough to handle and convert. Therefore using the following code will prevent R to read character strings as factors.

```
options(stringsAsFactors=FALSE)
```

If your character vector is read as a factor, use `as.character()` function. If your numeric vector is read as a factor, use `as.numeric(as.character())` function. Examples are given below.

```
factvec<-factor(c("a","b","c","a")) #Factor data vector
factvec
as.character(factvec) #Convert to character
factvec2<-factor(c(10,20,30,40,10)) #Factor data vector with numbers only
factvec2
```

```
as.numeric(factvec2) #If you want to convert directly to numeric, output will not be desir
as.numeric(as.character(factvec2))
```

5.6.1 RData

RData is a special data file type used by R. It is quite useful and efficient to store (better than csv). One disadvantage is it is not as common as csv, so reading RData outside R is a challenge.

```
load(path="some_RData")
save(some_data_frame,file="some_file.RData")
```

5.7 Packages

Packages are the most important asset class of R. These last years have seen a rapid expansion of R packages for almost any topic of interest that need computation. There are two steps to use a package; to install and to load.

```
install.packages("package_name") #Install command
library(package_name) #Load the package require() also works. No quotes!
```

Remember: You need to install a package only once. It is downloaded and ready to use whenever you load the package with `library()`. Packages are updated from time to time. To update your installed packages, use `update.packages()` command.

Below displays an example of a package use from the start. You will see how it is done in base R and how it can be enhanced with the packages.

5.7.1 Plotting

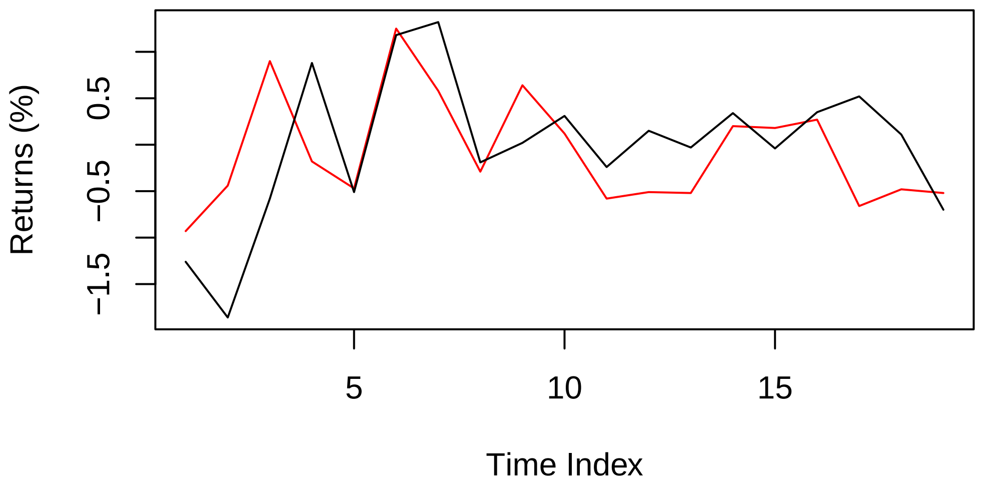
Plotting in R can be a bit problematic and hard. Let's plot the returns of stock indexes of the previous `EuStockMarkets` data.

```
#Let's redo what we did previously.
eu_df<- data.frame(EuStockMarkets[1:20,]) #Take the first 20 rows of the stock market inde
eu_df_returns <- data.frame(DAX=100*(round(eu_df$DAX[-1]/eu_df$DAX[-20],4)-1),
                            CAC=100*(round(eu_df$CAC[-1]/eu_df$CAC[-20],4)-1)) #Calculate
eu_df_returns
```

```
     DAX   CAC
1  -0.93 -1.26
2  -0.44 -1.86
3   0.90 -0.58
4  -0.18  0.88
5  -0.47 -0.51
6   1.25  1.18
7   0.58  1.32
8  -0.29 -0.19
9   0.64  0.02
10  0.12  0.31
11 -0.58 -0.24
12 -0.51  0.15
13 -0.52 -0.03
14  0.20  0.34
15  0.18 -0.04
16  0.27  0.35
17 -0.66  0.52
18 -0.48  0.11
19 -0.52 -0.70
```

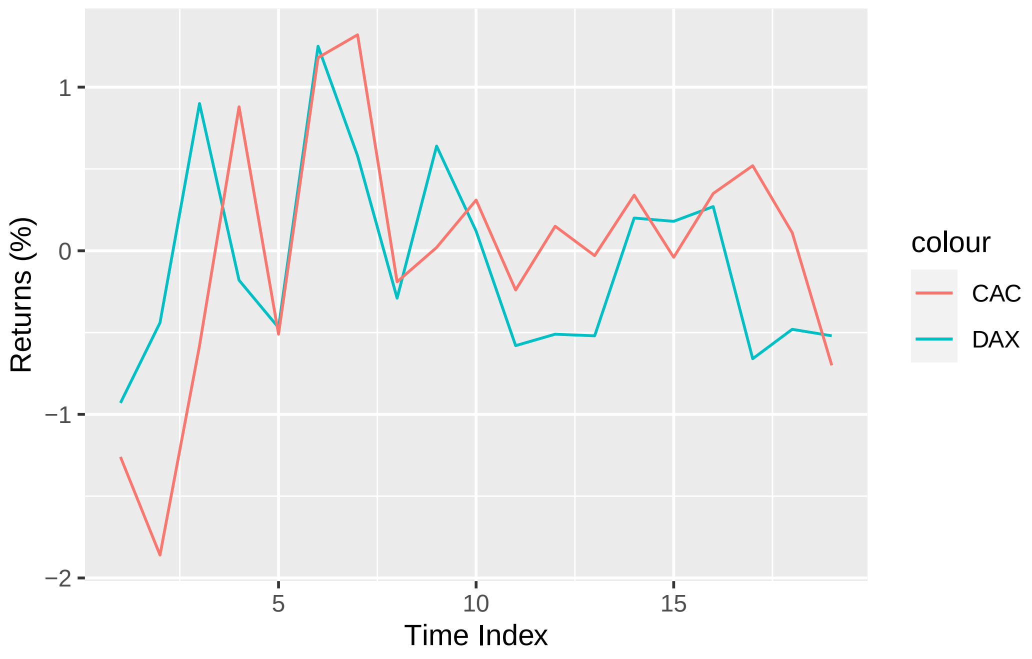
Base R plotting is as following.

```
plot(x=1:nrow(eu_df_returns),
     y=eu_df_returns$DAX,
     type="l",col="red",
     ylim=c(min(unlist(eu_df_returns)),max(unlist(eu_df_returns))),
     ylab="Returns (%)",
     xlab="Time Index")
lines(eu_df_returns$CAC)
```



You can probably do better with `ggplot2` package. It has more beautiful aesthetics, more readable code and better options. Even with the default values your plots will look better. Here is a simple implementation of the previous example.

```
if(!("ggplot2" %in% rownames(installed.packages()))){
  install.packages("ggplot2") #Install the package (you can skip it if it is already insta
}
library(ggplot2)
ggplot(data=eu_df_returns,aes(x=1:nrow(eu_df_returns))) +
geom_line(aes(y=DAX,color="DAX")) +
geom_line(aes(y=CAC,color="CAC")) +
labs(x="Time Index",y="Returns (%)")
```



5.8 Recommendations

R is quite extensive and the best ways to quickly learn are to write as much code as possible (this is the boring advice) and expose yourself to information by subscribing to newsletters, following related Twitter accounts and Facebook pages. Some prominently beneficial sources are given below.

R CRAN Task View: Curated lists of packages categorized on the purpose of use. They have categories such as Finance, Time Series and Econometrics. It is a good way to start searching for packages.

Stackexchange: Programmer's best friend. It is the umbrella site for highly technical Q&A sites such as Stack Overflow (for general programming), Cross Validated (statistics and data science) and Quantitative Finance. You can ask your programming problems here by providing an MWE (minimal working example)

Kaggle: There are many data science tasks, data sets and codes in here. Known for data competitions.

Quandl: Vast collection of data sets mainly on economics and finance. Great R support (even has a package).

GitHub: Most popular online code repository for git[3] based projects. Plus, putting R packages on GitHub prior to release on CRAN is a popular practice (advanced topic).

Coursera: Online learning at its best. There are many good quality quantitative finance, R and data science courses in here.

R-bloggers: Very useful site about R. I personally recommend subscribing to their newsletter and following their Twitter and Facebook accounts.

R-SIG-FIN: A mail group about R and Finance. A bit outdated but you can still search the archives.

ROpenSci: An organization promoting reproducible research with R. They have many good packages also.

5.8.1 R Cheat Sheets

There are many code cheat sheets on the internet. Here are some. I will update the list with new additions.

RStudio Cheat Sheets: Cheat sheets on base R, plotting and some very useful packages (i.e. `dplyr`, `ggplot2`, `shiny`, `rmarkdown`).

Data Management

Quandl Cheat Sheet: A cheat sheet by Quandl. There are also tips to use the `quandl` package.

R Reference Card: This one is from official R site.

Google's R Style Guide: This is more about styling your code. Best practices for readability.

[3]Git is a version controlling system mainly for software development but basically for any project. It mainly replaces the practice of code_finalfinalfinal.r and finalreport_lastlastlastthisisthelastipromise.docx

6 dplyr

6.1 Introduction to dplyr 1.0.0+

Main purpose of this document is to introduce a major data manipulation package, `dplyr`, with a contemporary subject. There are seven fundamental `dplyr` functions: `select`/`rename`, `filter`, `distinct`, `arrange`, `mutate`/`transmute`, `group_by` and `summarise` with a number of assisting functions used either in these functions or separately. In this document, we will cover every one of them and there will be supplementary functions to carry out data operations. Also, pipe operator (`%>%`) will be briefly introduced. This document is updated for `dplyr 1.0.0` and `R 4.0+` to show you new stuff as well. We will also use `lubridate` package for date time operations but we will not cover it except for quick explanations (check appendix at the end of this document for a mini tutorial). You can use advanced features in the “Advance Usage” subsections.

6.1.1 Preparations

There are two prerequisites to start: Install `tidyverse` package and putting the relevant data set into the working directory (write `getwd()` in the console to locate your working directory). In this document, topic of the data set is the hourly licensed and unlicensed renewable energy production data between January 1, 2018 and May 31, 2020.

To install the package run `install.packages("tidyverse")` in the console and select a mirror (first one is quite ok). Once you install the library you can always call it with `library(tidyverse)` command (**no need to reinstall**). You can download the data set from its **GitHub Repository**.

```r
library(tidyverse) #tidyverse is a package group which includes dplyr as well
library(lubridate)
raw_df <- readRDS("rp_201801_202005_df.rds")
```

First of those above commands calls the package (different from installing) The second command assigns the data to `raw_df` variable. There are two types of assignment operators in R: `<-` and `=`. No operation in R is permanent unless you assign it to somewhere (There are exceptions, though. See `data.table` package for instance.). We will benefit from this property in this document as well. No matter how many operations we do on each example we will always start from the original data frame.

Let's do a simple sanity check. The output of the following command reads "21,168 x 17" in the first line, which means there are 21,168 rows and 17 columns in the tibble. Tibble is the name of the data.frame type of dplyr. It usually is data frame plus some advanced features. There are abbreviations of data types under each column name. These are usually character/string (), numeric (, if integer), logical (TRUE/FALSE, logical) (), factor (), date () and datetime (). In the introduction phase we will only use character, numeric and logical data types.

```
print(raw_df,n=3)
```

```
# A tibble: 21,168 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-31 23:00:00    1434.           913.       75.8         2585.
2 2020-05-31 22:00:00    1577.           908.       75.6         2631.
3 2020-05-31 21:00:00    1858.           901.       75.4         2585.
# i 21,165 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

Also we can use `glimpse` function to inspect. Using `glimpse` each column is represented in a row with its data type and first few entries. Our data consists of hourly renewable electricity production of YEKDEM plants from different origins and license types. YEKDEM is a type of feed-in-tariff incentive framework. Suffixes with "_lic" represents licensed (larger scale) plants and "_ul" represents unlicensed (smaller scale) production. canalType, riverType and reservoir columns represent hydro power.

```
raw_df %>% glimpse()
```

```
Rows: 21,168
Columns: 17
$ dt              <dttm> 2020-05-31 23:00:00, 2020-05-31 22:00:00, 2020-05-31 ~
$ wind_lic        <dbl> 1433.8132, 1577.1419, 1857.5492, 1933.0142, 2031.7862,~
$ geothermal_lic  <dbl> 912.7882, 907.9303, 900.5844, 888.4561, 864.5402, 847.~
$ biogas_lic      <dbl> 75.8047, 75.6163, 75.3906, 76.7873, 76.9707, 77.5750, ~
$ canalType_lic   <dbl> 2584.930, 2630.602, 2585.038, 2542.381, 2594.459, 2622~
$ riverType_lic   <dbl> 316.5538, 316.6800, 356.7637, 350.1544, 377.5312, 379.~
$ biomass_lic     <dbl> 262.4994, 253.0814, 246.9268, 249.9152, 248.2336, 246.~
$ landfillGas_lic <dbl> 100.3971, 101.1378, 100.4442, 100.7307, 102.2474, 102.~
$ sun_lic         <dbl> 0.0000, 0.0000, 0.0000, 0.0000, 2.0594, 14.2800, 48.09~
$ reservoir_lic   <dbl> 2306.303, 2296.045, 2279.266, 2308.918, 2792.313, 3180~
$ others_lic      <dbl> 48.3833, 48.4011, 48.4041, 48.4199, 48.4653, 48.5485, ~
$ wind_ul         <dbl> 3.7751, 4.8375, 7.6659, 11.8121, 13.1070, 13.1830, 10.~
$ biogas_ul       <dbl> 16.9293, 16.9227, 16.9052, 16.7517, 16.2928, 16.5989, ~
$ canalType_ul    <dbl> 4.1749, 4.4221, 4.4658, 4.6020, 4.6195, 4.5146, 4.6616~
$ biomass_ul      <dbl> 15.4698, 15.3609, 16.0483, 15.2271, 15.5563, 15.5007, ~
$ sun_ul          <dbl> 0.0582, 0.0320, 0.0335, 1.3121, 103.3267, 555.5787, 14~
$ others_ul       <dbl> 0.0610, 0.0395, 0.4136, 0.5508, 0.7106, 1.3775, 2.7468~
```

Did you notice the expression we used this time? Pipe operator makes data analysis and transformation very easy and civilized. We will use pipes frequently in this document and in the future.

We can connect many functions without calling the variable multiple times with the help of the pipe operator.

6.2 Fundamentals

In this chapter fundamental functions of `dplyr` are introduced. Every function will be used in the following examples after it has been introduced. To limit the number of displayed rows, we use the following global option. You can ignore this part in your exercises.

```r
options(tibble.print_max = 3, tibble.print_min = 3)
```

6.2.1 `select/rename`

Select, as the name suggests, is used to select columns. For instance, suppose we only want licensed wind production (wind_lic) and date-time (dt) columns.

```r
raw_df %>% select(dt,wind_lic)
```

```
# A tibble: 21,168 x 2
  dt                  wind_lic
  <dttm>                 <dbl>
1 2020-05-31 23:00:00    1434.
```

```
2 2020-05-31 22:00:00    1577.
3 2020-05-31 21:00:00    1858.
# i 21,165 more rows
```

If we wanted to write the above expression without the pipe operator, we could go with the sad expression below. You can extrapolate how complicated things can be without the pipe.

```
select(raw_df,dt,wind_lic)
```

We can use **rename** to rename columns (again as the name suggests). Let's change dt to date_time.

```
raw_df %>% rename(date_time = dt)
```

```
# A tibble: 21,168 x 17
  date_time           wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-31 23:00:00    1434.           913.       75.8         2585.
2 2020-05-31 22:00:00    1577.           908.       75.6         2631.
3 2020-05-31 21:00:00    1858.           901.       75.4         2585.
# i 21,165 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

p.s. We can rename columns inside select function.

Select has many convenient sub operators and special expressions. If we know the order of columns, we can use the scope (:) expression to get all the columns determined by the scope. Suppose, we want date-time (dt) and licensed production.

```
raw_df %>% select(date_time=dt,wind_lic:others_lic)
```

```
# A tibble: 21,168 x 11
  date_time           wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-31 23:00:00    1434.           913.       75.8         2585.
2 2020-05-31 22:00:00    1577.           908.       75.6         2631.
3 2020-05-31 21:00:00    1858.           901.       75.4         2585.
```

```
# i 21,165 more rows
# i 6 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>, others_lic <dbl>
```

We can eliminate unwanted columns by putting `-` before the names. Suppose I am not interested in wind values, want to remove all other related columns from the data set, and all other related column names start with "wind_". We can do it using `-` and `starts_with`.

```
raw_df %>% select(-starts_with("wind_"))
```

```
# A tibble: 21,168 x 15
  dt                  geothermal_lic biogas_lic canalType_lic riverType_lic
  <dttm>                       <dbl>      <dbl>         <dbl>         <dbl>
1 2020-05-31 23:00:00           913.       75.8         2585.          317.
2 2020-05-31 22:00:00           908.       75.6         2631.          317.
3 2020-05-31 21:00:00           901.       75.4         2585.          357.
# i 21,165 more rows
# i 10 more variables: biomass_lic <dbl>, landfillGas_lic <dbl>, sun_lic <dbl>,
#   reservoir_lic <dbl>, others_lic <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

There are similar expressions for other purposes, such as `starts_with`, `everything` and `contains`. You can see all the expressions in the Cheat Sheet link given at the end of this document.

`dplyr 1.0.0` Feature: Sometimes you just want to change the order of the columns. Then use `relocate`. Suppose we want to show solar and wind production with date-time. But we want to get licensed wind together with licensed solar.

```
raw_df %>% select(dt,starts_with("sun_"),starts_with("wind")) %>% relocate(wind_lic,.befor
```

```
# A tibble: 21,168 x 5
  dt                  sun_lic wind_lic sun_ul wind_ul
  <dttm>                <dbl>    <dbl>  <dbl>   <dbl>
1 2020-05-31 23:00:00       0    1434. 0.0582    3.78
2 2020-05-31 22:00:00       0    1577. 0.032     4.84
3 2020-05-31 21:00:00       0    1858. 0.0335    7.67
# i 21,165 more rows
```

If we specify nothing, it will be in the first place.

```
raw_df %>% select(dt,starts_with("sun_"),starts_with("wind")) %>% relocate(wind_lic)
```

```
# A tibble: 21,168 x 5
  wind_lic dt                  sun_lic sun_ul wind_ul
     <dbl> <dttm>                <dbl>  <dbl>   <dbl>
1    1434. 2020-05-31 23:00:00       0 0.0582    3.78
2    1577. 2020-05-31 22:00:00       0 0.032     4.84
3    1858. 2020-05-31 21:00:00       0 0.0335    7.67
# i 21,165 more rows
```

We use `last_col()` if we want to take a column to the end.

```
raw_df %>% select(dt,starts_with("sun_"),starts_with("wind")) %>% relocate(dt,.after=last_
```

```
# A tibble: 21,168 x 5
  sun_lic sun_ul wind_lic wind_ul dt
    <dbl>  <dbl>    <dbl>   <dbl> <dttm>
1       0 0.0582    1434.    3.78 2020-05-31 23:00:00
2       0 0.032     1577.    4.84 2020-05-31 22:00:00
3       0 0.0335    1858.    7.67 2020-05-31 21:00:00
# i 21,165 more rows
```

Honestly, `relocate` is a very convenient function.

6.2.1.1 `select/rename` advanced usage

Advanced usage subsection introduces extra functionality which can be a bit confusing at the first phase. But, once you get a grasp on the fundamentals check back here as reference. There are several features not available for versions before `dplyr 1.0.0`.

We can use `rename_with` function to rename columns with given criteria. In pipe version of the function, first parameter is the function and the second parameter is the criterion. Let's replace all "Type" with "_type". For instance it should change "canalType" to "canal_type".

```
raw_df %>% rename_with(~gsub("Type","_type",.),contains("Type")) %>% glimpse()
```

```
Rows: 21,168
Columns: 17
$ dt               <dttm> 2020-05-31 23:00:00, 2020-05-31 22:00:00, 2020-05-31 ~
$ wind_lic         <dbl> 1433.8132, 1577.1419, 1857.5492, 1933.0142, 2031.7862,~
$ geothermal_lic   <dbl> 912.7882, 907.9303, 900.5844, 888.4561, 864.5402, 847.~
$ biogas_lic       <dbl> 75.8047, 75.6163, 75.3906, 76.7873, 76.9707, 77.5750, ~
$ canal_type_lic   <dbl> 2584.930, 2630.602, 2585.038, 2542.381, 2594.459, 2622~
$ river_type_lic   <dbl> 316.5538, 316.6800, 356.7637, 350.1544, 377.5312, 379.~
$ biomass_lic      <dbl> 262.4994, 253.0814, 246.9268, 249.9152, 248.2336, 246.~
$ landfillGas_lic  <dbl> 100.3971, 101.1378, 100.4442, 100.7307, 102.2474, 102.~
$ sun_lic          <dbl> 0.0000, 0.0000, 0.0000, 0.0000, 2.0594, 14.2800, 48.09~
$ reservoir_lic    <dbl> 2306.303, 2296.045, 2279.266, 2308.918, 2792.313, 3180~
$ others_lic       <dbl> 48.3833, 48.4011, 48.4041, 48.4199, 48.4653, 48.5485, ~
$ wind_ul          <dbl> 3.7751, 4.8375, 7.6659, 11.8121, 13.1070, 13.1830, 10.~
$ biogas_ul        <dbl> 16.9293, 16.9227, 16.9052, 16.7517, 16.2928, 16.5989, ~
$ canal_type_ul    <dbl> 4.1749, 4.4221, 4.4658, 4.6020, 4.6195, 4.5146, 4.6616~
$ biomass_ul       <dbl> 15.4698, 15.3609, 16.0483, 15.2271, 15.5563, 15.5007, ~
$ sun_ul           <dbl> 0.0582, 0.0320, 0.0335, 1.3121, 103.3267, 555.5787, 14~
$ others_ul        <dbl> 0.0610, 0.0395, 0.4136, 0.5508, 0.7106, 1.3775, 2.7468~
```

Did you notice `~` and `.` in the function? Dot (`.`) is a representation of the entity. Depending on the situation it can be the latest version of the tibble in the pipe chain, a specific column or something else. `~` is a special character notifying that function evaluation will be done using the dot notation. We will see more examples of that.

Let's introduce `where`. If is a function from `tidyselect` package to select variables with a function where it returns TRUE. It is quite handy.

```r
raw_df %>% select(dt,starts_with("sun_"),starts_with("wind")) %>% relocate(where(is.numeri
```

```
# A tibble: 21,168 x 5
  sun_lic sun_ul wind_lic wind_ul dt
    <dbl>  <dbl>    <dbl>   <dbl> <dttm>
1       0 0.0582    1434.    3.78 2020-05-31 23:00:00
2       0 0.032     1577.    4.84 2020-05-31 22:00:00
3       0 0.0335    1858.    7.67 2020-05-31 21:00:00
# i 21,165 more rows
```

We can also use `any_of` or `all_of` functions in `select`. The main difference is while the former returns as much as it can, the latter will throw an error if any one of the criteria is not fulfilled. Let's try to select "dt", "others_lic" and "nuclear_lic". Since this data does not include nuclear power production we should not see it.

```r
raw_df %>% select(any_of(c("dt","others_lic","nuclear_lic")))
```

```
# A tibble: 21,168 x 2
  dt                  others_lic
  <dttm>                   <dbl>
1 2020-05-31 23:00:00       48.4
2 2020-05-31 22:00:00       48.4
3 2020-05-31 21:00:00       48.4
# i 21,165 more rows
```

In order not to break our notebook, we wrap it around `try` (error handling is another topic).

```r
try(raw_df %>% select(all_of(c("dt","others_lic","nuclear_lic"))))
```

```
Error in all_of(c("dt", "others_lic", "nuclear_lic")) :
  Can't subset columns that don't exist.
x Column `nuclear_lic` doesn't exist.
```

6.2.2 `filter/distinct`

Filter (no more "as the name suggests", as you already figured it out) helps filter rows according to given criteria. It is highly similar with Excel's filter functionality (but much much more flexible and reliable).

Let's see the production at `2020-05-08 16:00:00`.

```r
raw_df %>% filter(dt == "2020-05-08 16:00:00")
```

```
# A tibble: 1 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-08 16:00:00    2618.           856.       79.8         3896.
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

By using `==` operator, we bring the values in `dt` column which are equal to the hour we desired. There are other expressions such as not equal to (`!=`), greater than (or equal to) (`>`,`>=`), smaller than (or equal to) (`<`,`<=`), in (`%in%`) and some more.

At the same time we can make comparisons between columns and combine multiple criteria to create more complex filters. Here we use AND (`&`) and OR (`|`) operators to combine criteria.

Suppose we want to find our the times when licensed wind production is greater than all of hydro type licensed production.

```
raw_df %>% filter(wind_lic > canalType_lic & wind_lic > reservoir_lic & wind_lic > riverTy
```

```
# A tibble: 11,287 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-27 19:00:00    3303.           930.       74.7         2969.
2 2020-05-27 18:00:00    3596.           914.       75.0         2953.
3 2020-05-27 17:00:00    3551.           900.       76.3         2954.
# i 11,284 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

We can add numeric operations as well. Suppose we want to find the total solar production is greater than total wind production.

```
raw_df %>% filter(wind_lic + wind_ul < sun_lic + sun_ul)
```

```
# A tibble: 4,949 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-31 16:00:00    2036.           843.       76.8         2616.
2 2020-05-31 15:00:00    1875.           845.       77.4         2685.
3 2020-05-31 14:00:00    1755.           853.       77.2         2715.
# i 4,946 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

Suppose we want to filter only the unique values. Then we simply use `distinct` command. Let's get unique rounded licensed wind production values.

```r
raw_df %>% distinct(round(wind_lic))
```

```
# A tibble: 4,893 x 1
  `round(wind_lic)`
              <dbl>
1              1434
2              1577
3              1858
# i 4,890 more rows
```

If we want to keep all columns we simply make the parameter `.keep=TRUE`.

```r
raw_df %>% distinct(round(wind_lic),.keep=TRUE)
```

```
# A tibble: 4,893 x 2
  `round(wind_lic)` .keep
              <dbl> <lgl>
1              1434 TRUE
2              1577 TRUE
3              1858 TRUE
# i 4,890 more rows
```

6.2.2.1 `filter/distinct` advanced usage

Let's introduce `slice`. This function helps return rows by its row number. Suppose we want the top 5 rows.

```r
raw_df %>% slice(1:5) %>% print(n=5)
```

```
# A tibble: 5 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-31 23:00:00    1434.           913.       75.8         2585.
2 2020-05-31 22:00:00    1577.           908.       75.6         2631.
3 2020-05-31 21:00:00    1858.           901.       75.4         2585.
```

```
4 2020-05-31 20:00:00    1933.            888.       76.8          2542.
5 2020-05-31 19:00:00    2032.            865.       77.0          2594.
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

If we want to return random rows we have `slice_sample`. Let's bring 5 random rows.

```r
raw_df %>% slice_sample(n=5)
```

```
# A tibble: 5 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2018-05-19 06:00:00     657.           603.       61.7          2012.
2 2018-02-14 11:00:00    4399.           700.       67.4          1864.
3 2020-04-04 14:00:00    1851.           882.       79.6          3293.
# i 2 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

If we want to do it proportionately, we have the `prop` parameter. Let's say we want 0.1% of the data frame.

```r
raw_df %>% slice_sample(prop=0.001)
```

```
# A tibble: 21 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2019-02-16 03:00:00    4364.           886.       83.9          2165.
2 2018-02-01 12:00:00    3076.           703.       68.2          1349.
3 2018-02-10 18:00:00    2827.           687.       65.9          1672.
# i 18 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

There are other `slice_*` type functions. These are `slice_head`/`slice_tail` for first/last n or percentage of rows. `slice_min`/`slice_max` for the top/bottom n rows according to an ordering criteria.

6.2.3 `arrange`

Arrange sorts rows from A to Z or smallest to largest. It has great similarity with Excel's Sort functionality.

Let's sort licensed reservoir production from largest to smallest.

```
raw_df %>% select(dt,reservoir_lic) %>% arrange(desc(reservoir_lic))
```

```
# A tibble: 21,168 x 2
  dt                  reservoir_lic
  <dttm>                      <dbl>
1 2019-05-10 23:00:00         5058.
2 2019-05-10 21:00:00         5035.
3 2019-05-15 02:00:00         5019.
# i 21,165 more rows
```

Do you see `desc()` function inside `arrange`? By default `arrange` sorts a column by first to last or A-Z. `desc` reverses this.

You can also use multiple sorting criteria and use operations inside `arrange`. Let's arrange by licensed wind production rounded down (`floor`) in 100s range (e.g. 5634 = 5600 and 5693 = 5600 as well). Then we sort by date time to see the first time the production entered a 100-range in the data time period.

```
raw_df %>% arrange(desc(floor(wind_lic/100)*100),dt)
```

```
# A tibble: 21,168 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2019-09-15 17:00:00    5767.           691.       83.0          922.
2 2018-09-26 19:00:00    5622.           672.       67.8          951.
3 2019-09-15 15:00:00    5628.           692.       81.4          901.
# i 21,165 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

6.2.4 `mutate/transmute`

Mutate is the function when we do operations and calculations using other columns.

For instance let's calculate wind power's share in total renewables production at each hour.

```
raw_df %>% mutate(wind_lic_perc = wind_lic / (wind_lic + geothermal_lic + biogas_lic + can
```

```
# A tibble: 21,168 x 2
  dt                  wind_lic_perc
  <dttm>                      <dbl>
1 2020-05-31 23:00:00         0.177
2 2020-05-31 22:00:00         0.191
3 2020-05-31 21:00:00         0.219
# i 21,165 more rows
```

You can use many R functions (from both base functions and other packages). For instance to calculate "competition" wind and solar we can use the following expression.

```
raw_df %>% mutate(wind_or_solar = ifelse(wind_lic + wind_ul > sun_lic + sun_ul, "wind", "s
```

```
# A tibble: 21,168 x 2
  dt                  wind_or_solar
  <dttm>              <chr>
1 2020-05-31 23:00:00 wind
2 2020-05-31 22:00:00 wind
3 2020-05-31 21:00:00 wind
# i 21,165 more rows
```

Transmute has the same functionality as mutate with the additional property similar to `select`. Transmute returns only the columns included in the function. Suppose we also want to calculate the difference between total wind and total solar.

```
raw_df %>% transmute(dt, wind_or_solar = ifelse(wind_lic + wind_ul > sun_lic + sun_ul, "wi
```

```
# A tibble: 21,168 x 3
  dt                  wind_or_solar absdiff
  <dttm>              <chr>           <dbl>
1 2020-05-31 23:00:00 wind            1438.
```

```
2 2020-05-31 22:00:00 wind            1582.
3 2020-05-31 21:00:00 wind            1865.
# i 21,165 more rows
```

6.2.4.1 `mutate/transmute` advanced usage

Suppose we want to see the difference between the previous and next hour's production. We offset rows using `lead` and `lag` functions. But remember lead and lag does not actually give you "next/previous hour's" values, just the rows. You may need to arrange your data.

```r
raw_df %>% transmute(dt, wind_lic, wind_lic_prev_h = lead(wind_lic,1), wind_lic_next_h = l
```

```
# A tibble: 21,168 x 4
  dt                  wind_lic wind_lic_prev_h wind_lic_next_h
  <dttm>                 <dbl>           <dbl>           <dbl>
1 2020-05-31 23:00:00    1434.           1577.             NA
2 2020-05-31 22:00:00    1577.           1858.           1434.
3 2020-05-31 21:00:00    1858.           1933.           1577.
# i 21,165 more rows
```

If you want to use the same function over several columns, you can use the new `across` function. Let's round every numeric column to one significant digit.

```r
raw_df %>% mutate(across(where(is.numeric),~round(.,1)))
```

```
# A tibble: 21,168 x 17
  dt                  wind_lic geothermal_lic biogas_lic canalType_lic
  <dttm>                 <dbl>          <dbl>      <dbl>         <dbl>
1 2020-05-31 23:00:00    1434.           913.       75.8         2585.
2 2020-05-31 22:00:00    1577.           908.       75.6         2631.
3 2020-05-31 21:00:00    1858.           901.       75.4         2585
# i 21,165 more rows
# i 12 more variables: riverType_lic <dbl>, biomass_lic <dbl>,
#   landfillGas_lic <dbl>, sun_lic <dbl>, reservoir_lic <dbl>,
#   others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>, canalType_ul <dbl>,
#   biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

We can also specify columns. Let's see the comparative production of wind and reservoir hydro against the unlicensed solar production. We just increment solar production by 1 to prevent any infinity values.

```
raw_df %>% mutate(sun_ul = sun_ul + 1) %>% transmute(dt,across(c(wind_lic,reservoir_lic),~
```

```
# A tibble: 21,168 x 3
  dt                  wind_lic reservoir_lic
  <dttm>                 <dbl>         <dbl>
1 2020-05-31 23:00:00    1355.         2179.
2 2020-05-31 22:00:00    1528.         2225.
3 2020-05-31 21:00:00    1797.         2205.
# i 21,165 more rows
```

If there are multiple conditions `ifelse` is not enough. It is possible to use `case_when` to specify multiple conditions and outcomes.

```
raw_df %>% transmute(dt, solar_production_level = case_when(sun_ul > quantile(sun_ul,0.9)
```

```
# A tibble: 3 x 2
  dt                  solar_production_level
  <dttm>              <chr>
1 2020-05-31 17:00:00 above median
2 2020-05-31 16:00:00 high
3 2020-05-31 15:00:00 very high
```

`rowwise` is actually a type of `group_by/summarise` function but as the name suggests it allows us to do row-wise operations. Let's calculate row sums by using `c_across` function and `rowwise`. Oh and also now, experimentally, you can use relocate functionality in `mutate/transmute`. So, conveniently we can place it after date time.

```
raw_df %>% slice_head(n=5) %>% rowwise() %>% mutate(total_prod = sum(c_across(where(is.num
```

```
# A tibble: 5 x 18
# Rowwise:
  dt                  total_prod wind_lic geothermal_lic biogas_lic
  <dttm>                   <dbl>    <dbl>          <dbl>      <dbl>
1 2020-05-31 23:00:00      8082.    1434.           913.       75.8
2 2020-05-31 22:00:00      8248.    1577.           908.       75.6
3 2020-05-31 21:00:00      8496.    1858.           901.       75.4
# i 2 more rows
# i 13 more variables: canalType_lic <dbl>, riverType_lic <dbl>,
#   biomass_lic <dbl>, landfillGas_lic <dbl>, sun_lic <dbl>,
#   reservoir_lic <dbl>, others_lic <dbl>, wind_ul <dbl>, biogas_ul <dbl>,
#   canalType_ul <dbl>, biomass_ul <dbl>, sun_ul <dbl>, others_ul <dbl>
```

6.2.5 group_by/summarise

Finally we will learn how to calculate summary tables. It is similar to Pivot Tables in Excel. `group_by` is the grouping function, `summarise` is the summarising function.

For instance let's calculate number of hours where wind production is above 3000 MWh. We will use a special function `n()` to calculate number of rows. We can define groupings just like `mutate`.

```
raw_df %>% group_by(production_group = cut(wind_lic + wind_ul,breaks=c(0,1000,2000,3000,40
```

```
# A tibble: 6 x 2
  production_group count
  <fct>            <int>
1 [0,1e+03]         4294
2 (1e+03,2e+03]     5733
3 (2e+03,3e+03]     4725
# i 3 more rows
```

Normally, we get one result for each group and summary function. From `dplyr 1.0.0` we can have multiple row summarise for each group. Let's say we want to find the minimum and maximum licensed wind production ranges for each year. But, be warned, it can be a little confusing.

```
raw_df %>% group_by(year = lubridate::year(dt)) %>% summarise(wind_lic_range = range(wind_
```

```
Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
dplyr 1.1.0.
i Please use `reframe()` instead.
i When switching from `summarise()` to `reframe()`, remember that `reframe()`
  always returns an ungrouped data frame and adjust accordingly.
```

```
`summarise()` has grouped output by 'year'. You can override using the
`.groups` argument.
```

```
# A tibble: 6 x 2
# Groups:   year [3]
   year wind_lic_range
  <dbl>          <dbl>
1  2018           46.2
2  2018         5622.
3  2019           32.8
# i 3 more rows
```

6.2.5.1 `group_by/summarise` advanced usage

Just like `mutate/transmute` you can use `across` in `summarise` as well. Let's see median production of each year and each source. In this example we can also use the named list version of the functions and additional names structure.

```
raw_df %>% group_by(year = lubridate::year(dt)) %>% summarise(across(where(is.numeric),lis
```

```
# A tibble: 3 x 17
   year med_wind_lic med_geothermal_lic med_biogas_lic med_canalType_lic
  <dbl>        <dbl>              <dbl>          <dbl>             <dbl>
1  2018        1989.               694.           66.4             1480.
2  2019        2136.               814.           80.9             1575.
3  2020        2297.               951.           78.5             2779.
# i 12 more variables: med_riverType_lic <dbl>, med_biomass_lic <dbl>,
#   med_landfillGas_lic <dbl>, med_sun_lic <dbl>, med_reservoir_lic <dbl>,
#   med_others_lic <dbl>, med_wind_ul <dbl>, med_biogas_ul <dbl>,
#   med_canalType_ul <dbl>, med_biomass_ul <dbl>, med_sun_ul <dbl>,
#   med_others_ul <dbl>
```

6.3 Exercises

Solve the following exercises. Outputs are given below, you are expected write code to match the outputs.

1. Find the mean and standard deviation of licensed geothermal productions in all years. (Tip: Use `lubridate::year` to get years from date data.)

```
# A tibble: 3 x 3
   year mean_geo sd_geo
  <dbl>    <dbl>  <dbl>
```

```
1  2018     681.   65.2
2  2019     799.   74.2
3  2020     935.   59.0
```

2. Find the hourly average unlicensed solar (sun_ul) production levels for May 2020.

```
# A tibble: 24 x 2
   hour avg_prod
  <int>    <dbl>
1     0     0.17
2     1     0.37
3     2     0.7
# i 21 more rows
```

3. Find the average daily percentage change of licensed biomass (biomass_lic) in 2019. (e.g. Suppose daily production is 50 in day 1 and 53 in day 2, then the change should be (53-50)/50 -1 = 0.06) (Tip: Use `lubridate::as_date` to convert date time to date. Use `lag` and `lead` functions to offset values.)

```
# A tibble: 1 x 1
  average_change
           <dbl>
1        0.00282
```

4. Find the yearly total production levels in TWh (Current values are in MWh. 1 GWh is 1000 MWh and 1 TWh is 1000 GWh). (Tip: In order to avoid a lengthy summation you can use `tidyr::pivot_longer` to get a long format.)

```
# A tibble: 3 x 2
   year total_production
  <dbl>            <dbl>
1  2018             62.6
2  2019             76.7
3  2020             37.3
```

6.4 Conclusion

Fundamental `dplyr` functions provide very convenient tools for data analysis. It can also be used to generate the features required for modelling. You can process few million rows of data without breaking a sweat (for larger data sets you can use `data.table`), you can prepare functions instead of manual Excel operations. With R Markdown system, which this tutorial is prepared in, you can create reproducible documents and automatize the reports. You can use `ggplot2` for visualizations, which is also part of the tidyverse package ecosystem.

6.5 Appendix

6.5.1 Mini lubridate tutorial

In this tutorial we use a small portion of a very powerful package, `lubridate`. You can see the official website here.

Let's take just 3 dates at random from our data set.

```
set.seed(5)
lub_df <-
raw_df %>%
  select(dt) %>%
  sample_n(3)

print(lub_df)
```

```
# A tibble: 3 x 1
  dt
  <dttm>
1 2018-12-02 06:00:00
2 2019-01-12 05:00:00
3 2018-04-16 09:00:00
```

Since we called lubridate at the beginning of this tutorial we do not need to call by package reference (`lubridate::`) but it is generally good practice.

```
lub_df %>%
  mutate(
    year = lubridate::year(dt),
    month = lubridate::month(dt),
    day = lubridate::day(dt),
    week_day = lubridate::wday(dt),
    wday_label = lubridate::wday(dt,label=TRUE),
    hour = lubridate::hour(dt),
    minute = lubridate::minute(dt),
    second = lubridate::second(dt)
  )
```

```
# A tibble: 3 x 9
  dt                   year month   day week_day wday_label  hour minute second
  <dttm>              <dbl> <dbl> <int>    <dbl> <ord>      <int>  <int>  <dbl>
1 2018-12-02 06:00:00  2018    12     2        1 Sun            6      0      0
2 2019-01-12 05:00:00  2019     1    12        7 Sat            5      0      0
3 2018-04-16 09:00:00  2018     4    16        2 Mon            9      0      0
```

6.6 References

- Data Set: EPIAS/EXIST Transparency Platform
- Tidyverse: https://www.tidyverse.org/
- R for Data Science: https://r4ds.had.co.nz/
- Cheatsheet (Data Transformation) (Turkish Version)

7 ggplot2

7.1 Preparation

```
raw_df <- readRDS("rp_201801_202005_df.rds")
```

7.2 Scatterplot

First let's preprocess our data to get production in May 2020 and hours between 10:00 AM and 5:00 PM.

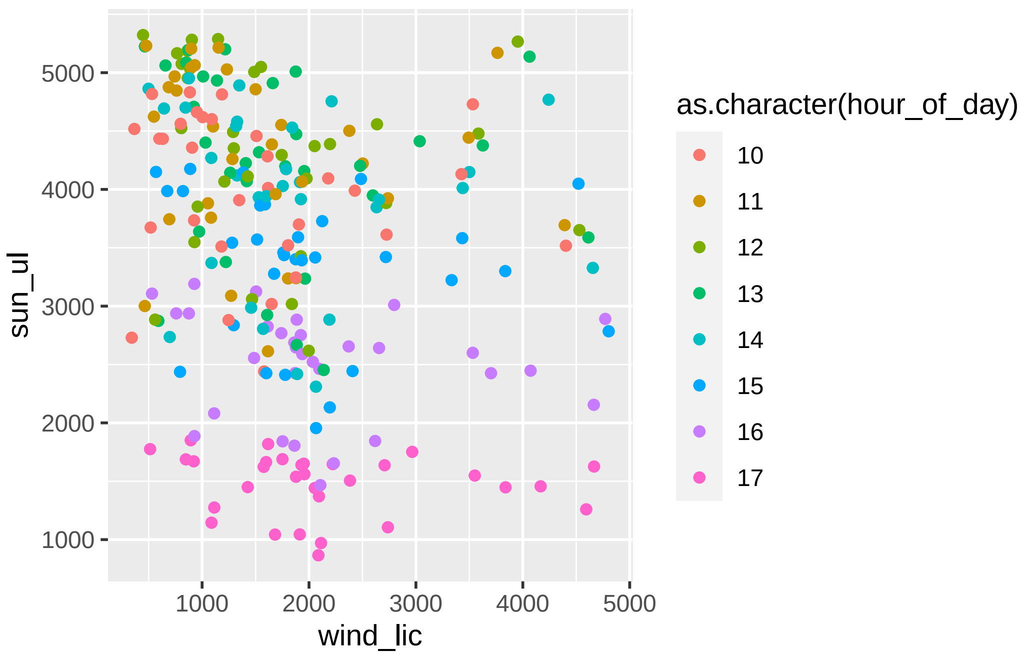
```
plot_df1 <-
  raw_df %>%
    filter(dt >= "2020-05-01" & dt < "2020-06-01" & lubridate::hour(dt) >= 10 & lubridate:

print(plot_df1,n=3)
```

```
# A tibble: 248 x 3
  hour_of_day wind_lic sun_ul
        <int>    <dbl>  <dbl>
1          17    2055.  1442.
2          16    2036.  2523.
3          15    1875.  3402.
# i 245 more rows
```

Let's plot licensed wind production against unlicensed solar production for May 2020 for hours between 10-17. We can say that

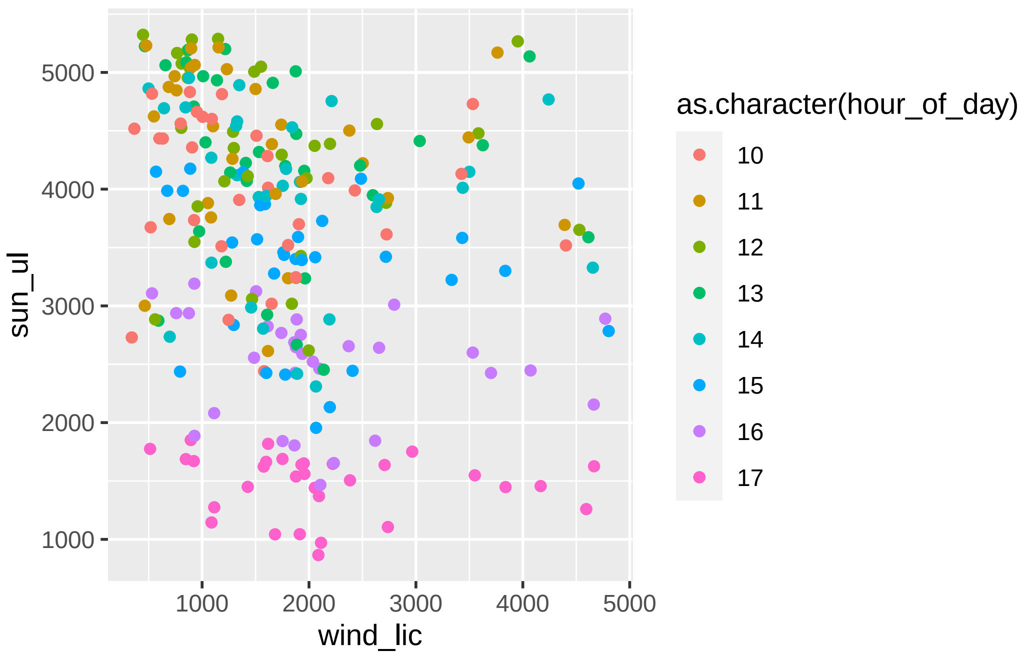
- Usually during morning time solar production is high and wind production is kind of low.
- Expectedly solar production is decreasing in the afternoon

```
ggplot(plot_df1, aes(x = wind_lic, y = sun_ul, color = as.character(hour_of_day))) + geom_
```



ggplot2 is quite flexible. We can move `aes` from ggplot object to geom_point object.

```
ggplot(plot_df1) + geom_point(aes(x = wind_lic, y = sun_ul, color = as.character(hour_of_d
```



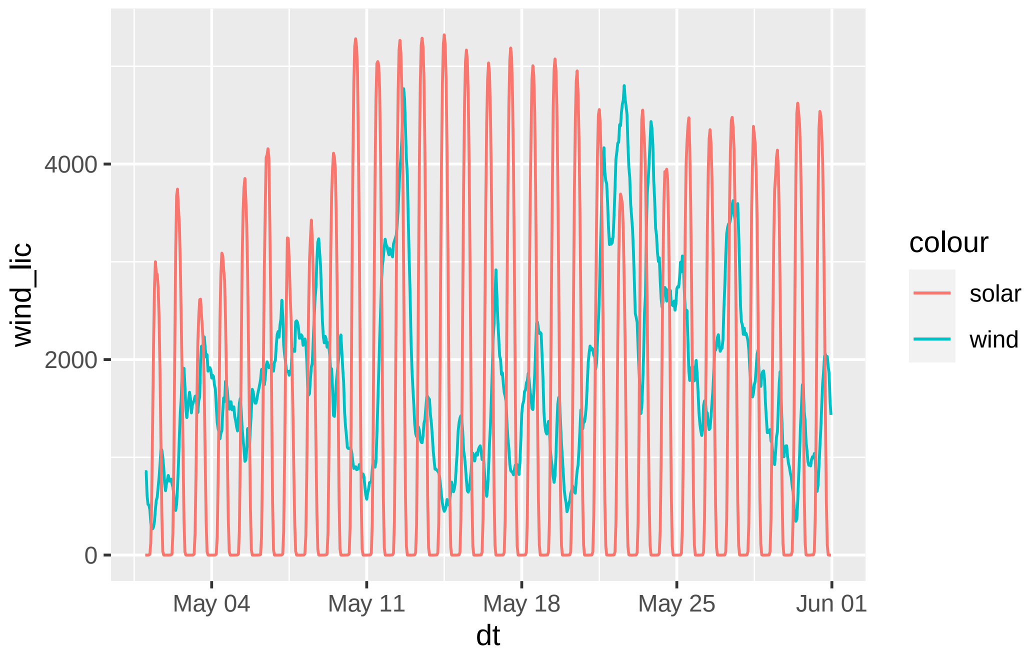
7.3 Line Chart

Here I want to compare wind and solar production in a time series plot.

```
plot_df2 <- raw_df %>% filter(dt >= "2020-05-01" & dt < "2020-06-01") %>% select(dt,wind_l

print(plot_df2,n=3)
```

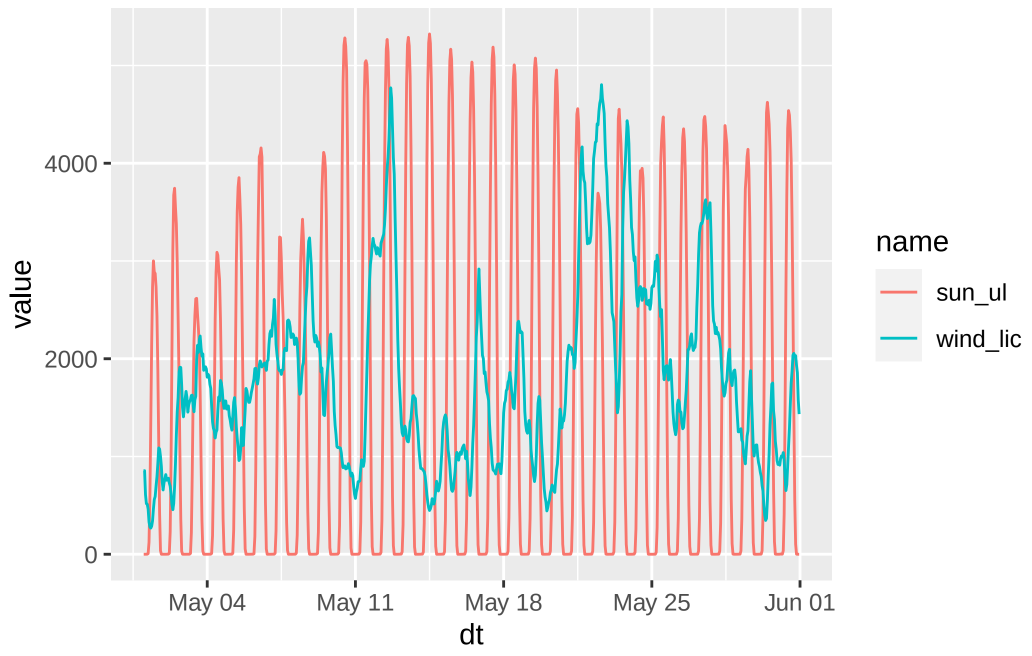
```
# A tibble: 744 x 3
  dt                    wind_lic sun_ul
  <dttm>                   <dbl>  <dbl>
1 2020-05-31 23:00:00    1434. 0.0582
2 2020-05-31 22:00:00    1577. 0.032
3 2020-05-31 21:00:00    1858. 0.0335
# i 741 more rows
```

```
ggplot(plot_df2) +
  geom_line(aes(x=dt,y=wind_lic,color="wind")) +
  geom_line(aes(x=dt,y=sun_ul,color="solar"))
```



What if I don't want to repeat geom_line functions? I can use `pivot_longer` function to get all in a single geom_line.

```
ggplot(plot_df2 %>% pivot_longer(.,cols=-dt),aes(x=dt,y=value,color=name)) + geom_line()
```



7.4 Bar Chart

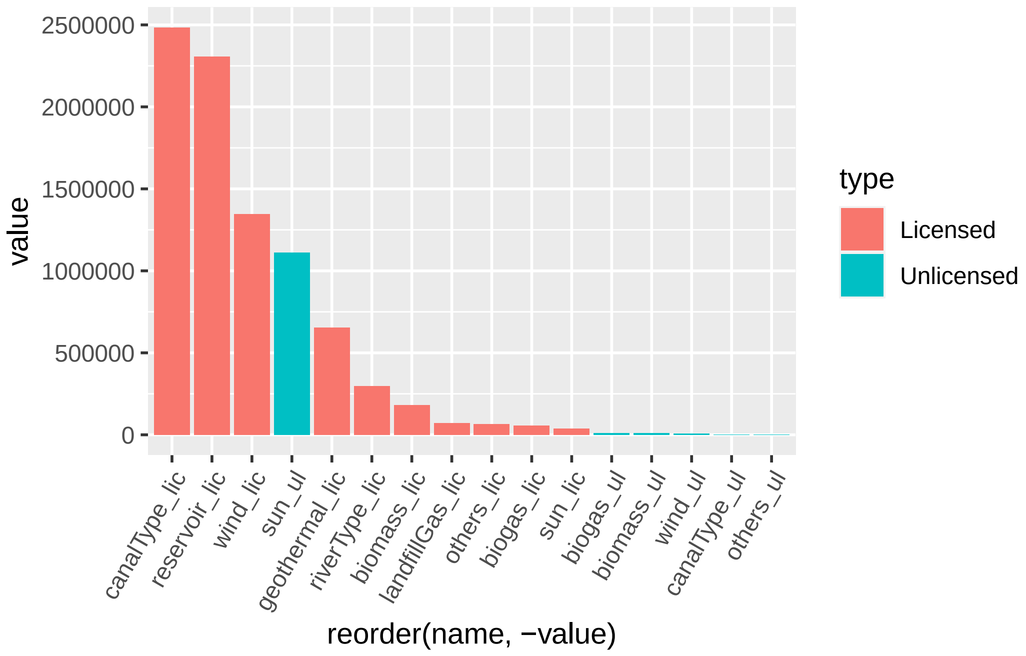
Now I'd like to get the May 2020 production and I'd like to differentiate Licensed and Unlicensed.

```
plot_df3 <- raw_df %>% filter(dt >= "2020-05-01" & dt < "2020-06-01") %>% summarise(across

print(plot_df3,n=3)
```

```
# A tibble: 16 x 3
  name                value type
  <chr>               <dbl> <chr>
1 wind_lic         1346260. Licensed
2 geothermal_lic    654089. Licensed
3 biogas_lic         56839. Licensed
# i 13 more rows
```

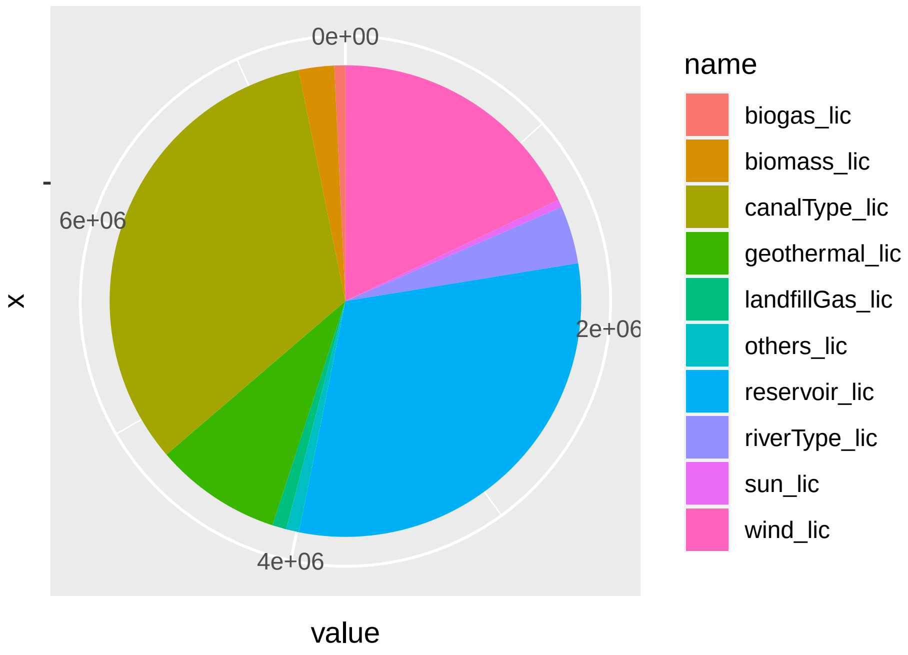
Let's plot total productions using a bar chart. To improve readability, we reorder by production and differentiate Licensed/Unlicensed using color.

```
ggplot(plot_df3,aes(x=reorder(name,-value),y=value,fill=type)) + geom_bar(stat = "identity
```



7.5 Pie Chart

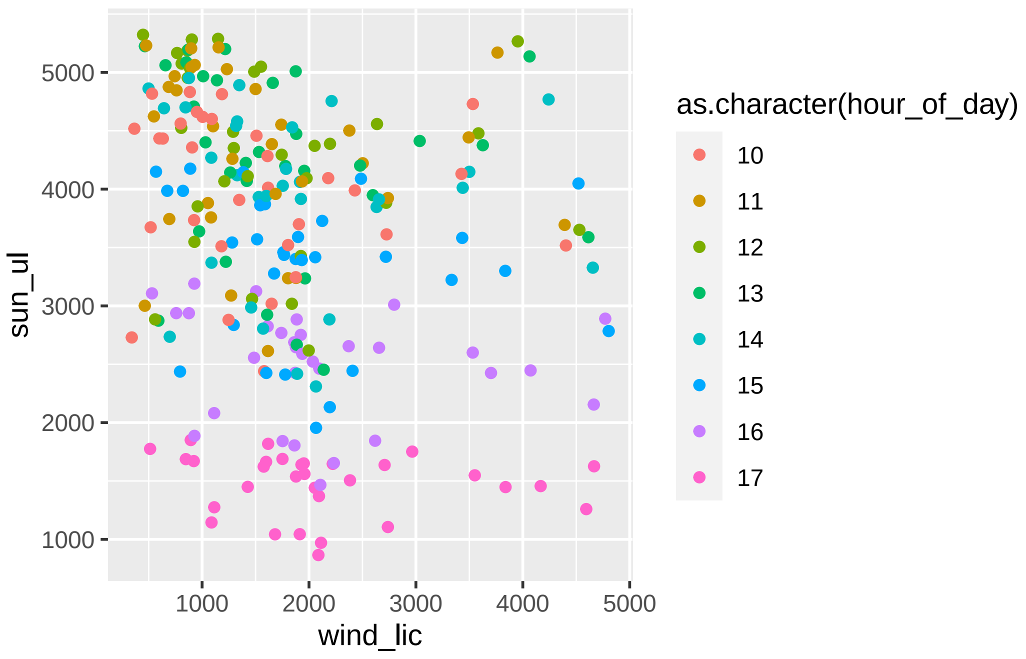
```
ggplot(plot_df3 %>% filter(type=="Licensed"),aes(x="",y=value,fill=name)) + geom_bar(stat=
```



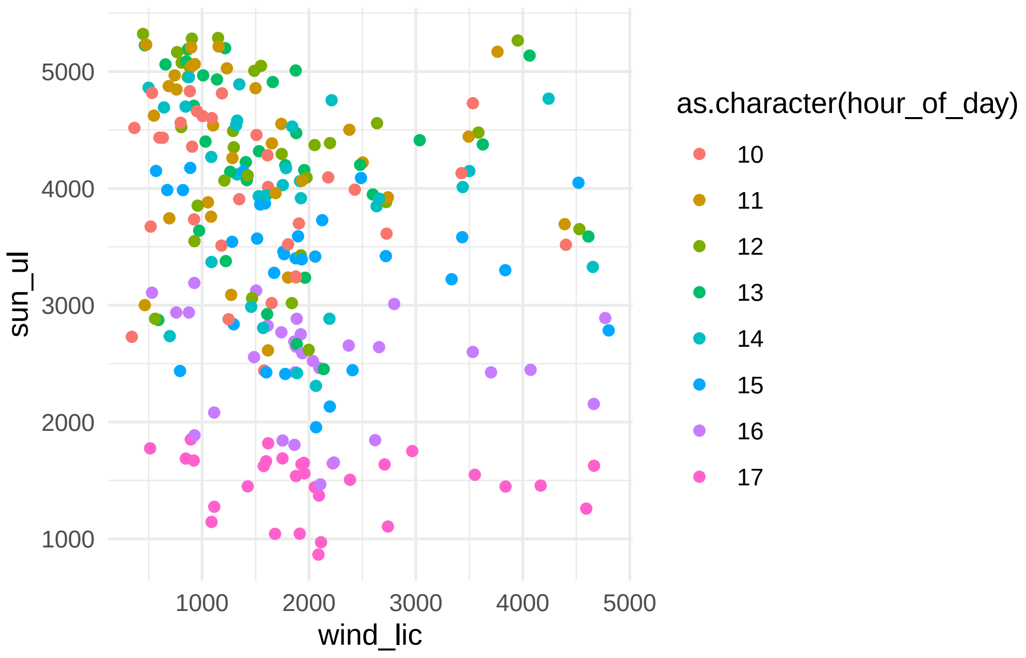
7.6 Theming and Customization

Let's get our charts better looks!

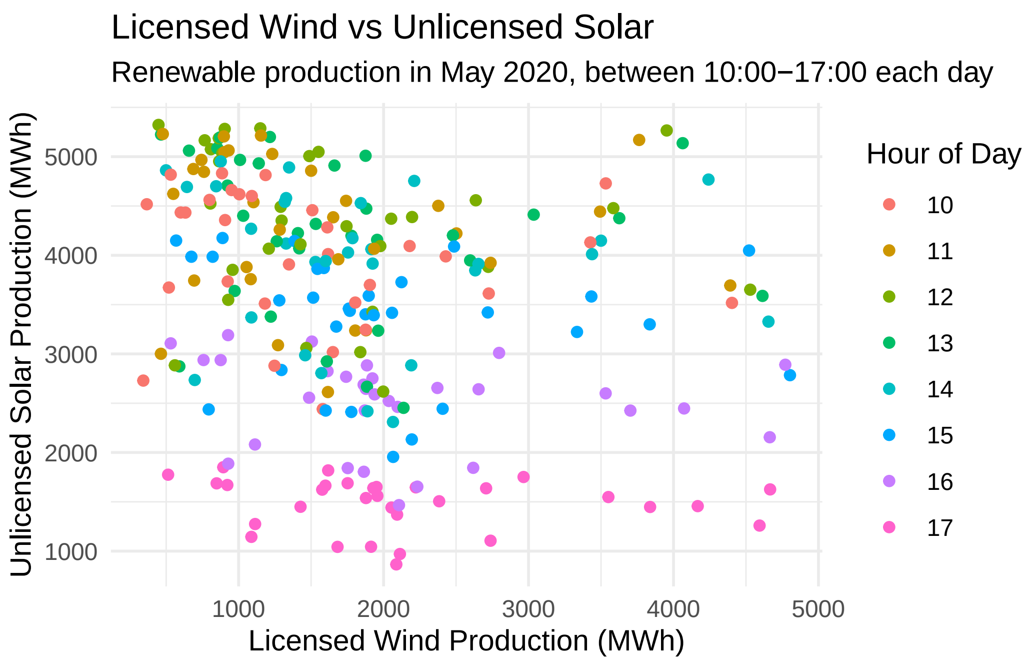
```
sc_plot <- ggplot(plot_df1) + geom_point(aes(x = wind_lic, y = sun_ul, color=as.character(

sc_plot
```



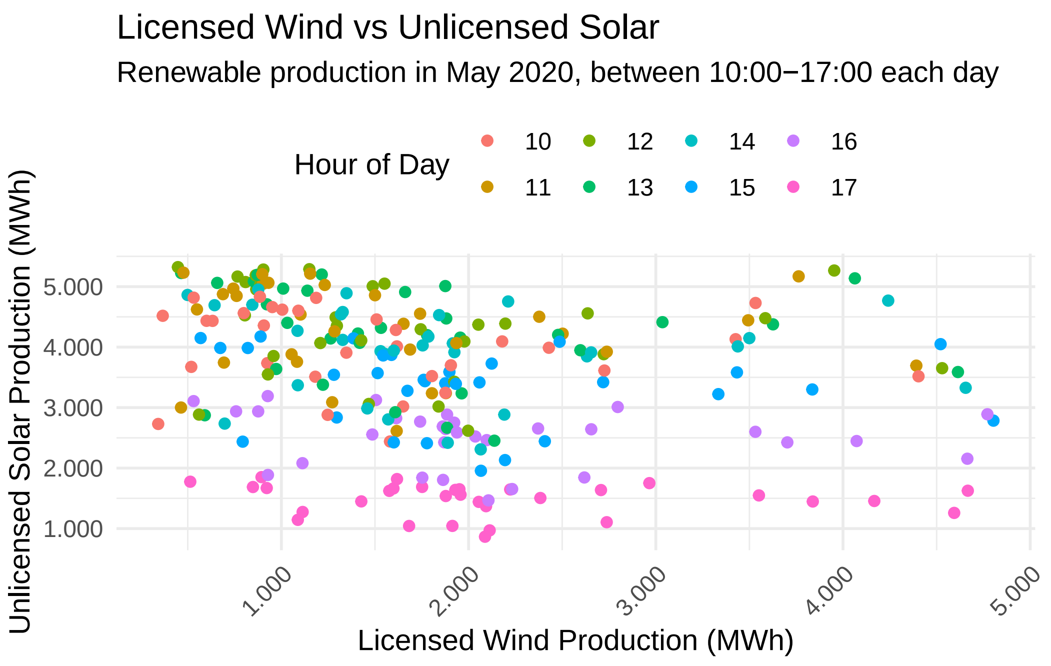
```
sc_plot2 <- sc_plot + theme_minimal()
sc_plot2
```



```r
sc_plot3 <-
sc_plot2 + labs(x="Licensed Wind Production (MWh)", y="Unlicensed Solar Production (MWh)",

sc_plot3
```



```r
sc_plot3 + theme(legend.position = "top",axis.text.x = element_text(angle=45,hjust=1,vjust
```


Licensed Wind vs Unlicensed Solar
Renewable production in May 2020, between 10:00−17:00 each day
Hour of Day
10
11
12
13
14
15
16
17
Unlicensed Solar Production (MWh)
5.000
4.000
3.000
2.000
1.000
1.000
2.000
3.000
4.000
5.000
Licensed Wind Production (MWh)

8 Quarto

Quarto is the most recent solution released by RStudio/Posit. It is very similar to RMarkdown but it has two major advantages. First advantage is Quarto is multilingual (currently R, Python, Julia and Observable). Second advantage is while RMarkdown inspired other packages such as `pagedown` for webpages, `bookdown` for books, `pkgdown` for packages, `blogdown` for blogs and `xaringan` for presentations Quarto provides those capabilities (mostly) and more under a single umbrella.

According to official announcement Quarto will not replace RMarkdown but it will not go in the same direction either. For more details you can read Yihui Xie's (author of RMarkdown) blog post.

You can start with the official page's Get Started and then Quarto Guide. You can also use Quarto Pub to publish your Quarto documents.

Also see Thomas Mock's Quarto Webinar (beware! 2hrs) and its materials.

8.1 Resources

- A Quarto tip a day
- Quarto Gallery
- Awesome Quarto
- #quartopub hashtag on Twitter

9 shiny

Coming soon...

10 Chapter End

Fundamentals chapter ends here. You now built the foundations for data analysis skills to take it to the next level. Congratulations!

Part II

Advanced

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

Using Python and R Together

In the ever-increasing requirements era of data science, Python is one of the fundamental tools that a data scientist should know about. Even though R is quite elegant and enough in many data related applications, a mix of Python might also be required to get the job done. One of the reasons is to benefit from API connections because Python enjoys first class support (e.g. native SDK) in many services. For instance, using `boto3` package for AWS was quite useful (before the paws R package).

RStudio especially positions itself as "A Single Home for R and Python Data Science" in their blog post. It is also the main reason why they created and support the `reticulate` R package. Before `reticulate`, Python integration was still possible but much more difficult in many aspects. It still might have quirks but `reticulate` provides much better integration.

In this tutorial, we are going to do simple examples using `reticulate` package. This tutorial is not exhaustive. For better coverage, check the official package page.

Initialization

It always starts with the loading of the package.

```r
library(reticulate)
```

Python has multiple versions (thankfully they phased out Python 2 but still a pain in many operating systems) You can learn the current python version PATH using `Sys.which` function. Output will differ in different systems and installations.

```r
Sys.which("python")
```

```
                              python
"/Users/rocket/.pyenv/shims/python"
```

For better detail, `py_config` is a good function. Output will differ in different systems and installations.

```
py_config()
```

```
python:          /Users/rocket/.virtualenvs/r-reticulate/bin/python
libpython:       /Users/rocket/.pyenv/versions/3.11.6/lib/libpython3.11.dylib
pythonhome:      /Users/rocket/.virtualenvs/r-reticulate:/Users/rocket/.virtualenvs/r-reticula
version:         3.11.6 (main, Nov 23 2023, 13:24:35) [Clang 14.0.0 (clang-1400.0.29.202)]
numpy:           /Users/rocket/.virtualenvs/r-reticulate/lib/python3.11/site-packages/numpy
numpy_version:   1.26.2
```

For available configurations `py_discover_config` can be used.

```
py_discover_config()
```

`reticulate` can use other python installations, virtual environments and conda versions. Although it is a great convenience, intricacies and delicateness of Python versions might still hurt your workflows.

```
use_python() ## python path
use_virtualenv() ## virtual environment name
use_condaenv() ## conda environment
```

Methods and Examples

In this section we will see some methodologies to use with `reticulate`.

Call Python functions R style

This is the fundamental and (in my opinion) worst way to benefit from Python in R. Because translation of style is imperfect and might not work in every case.

Here is an example with `pandas`. Install `pandas` if not installed.

```
py_install("pandas")
```

```
## similar to import pandas as pd
pd <- import("pandas")
## create a simple dataframe
df_pandas <- pd$DataFrame(data = list(col1 = c(2, 1, 3), col2 = c("a", "b", "c")))
```

```r
df_pandas
```

```
  col1 col2
1    2    a
2    1    b
3    3    c
```

Let’s try a simple example.

```r
os <- import("os")
os$path$join("a", "b", "c")
```

```
[1] "a/b/c"
```

Caveats: Some functions working on console might not work on RMarkdown

```r
try(df_pandas$sort_values("col1"))
```

```
Error in try(df_pandas$sort_values("col1")) : 
  attempt to apply non-function
```

```r
## filter using query fails both on console and rmarkdown
try(df_pandas$query("col1 > 1.0"))
```

```
Error in try(df_pandas$query("col1 > 1.0")) : 
  attempt to apply non-function
```

Call Python in RMarkdown chunk

The format is similar to an R chunk but instead of `r` write `python`.

~~~
```{python}
#your python code here
```
~~~

Here is an example

```python
import os
os.path.join("a","b","c")
```

```
'a/b/c'
```

Our pandas examples also works in this case.

```python
import pandas as pd

pandas_df_py = pd.DataFrame(data={'col1':[2,1,3],'col2':['a','b','c']})

pandas_df_py
```

```
   col1 col2
0     2    a
1     1    b
2     3    c
```

```python
pandas_df_py.sort_values('col1')
```

```
   col1 col2
1     1    b
0     2    a
2     3    c
```

```python
## filter using query fails both on console and rmarkdown
pandas_df_py.query('col1 > 1.0')
```

```
   col1 col2
0     2    a
2     3    c
```

Source Python Script

Personally, the most convenient way to use Python code is to source a `.py` file. Even though interoperability is a great idea, keeping Python and R codes as separate as possible will save you from a lot of headache in the future (at least with the current implementation).

We can source any python file using `source_python()` function easily. Let's name our python file `triangle.py` and write the following.

```
def area_of_triangle(h,x):
  return h*x/2
```

```
area_of_triangle(3,5)
```

```
7.5
```

Follow

```
reticulate::source_python("triangle.py") ### Remember to provide proper relative or absolu
```

Note: For RMarkdown demonstration purposes below we;

- created a temporary file
- wrote our python function to that file using `cat`
- then executed `source_python`
- then called the function as R function

```
pyfile <- tempfile(fileext = ".py")
cat("def area_of_triangle(h,x):
  return h*x/2", file = pyfile)
source_python(pyfile)

area_of_triangle(3, 5)
```

```
[1] 7.5
```

But deep down it is a Python function.

```
print(area_of_triangle)
```

```
<function area_of_triangle at 0x14a014f40>
```

Conclusion

To be honest Python versioning is a mess. Lots of parallel versions (even python2 version issues are still looming) and conflicts in different layers of computing might be troublesome especially in Docker settings. Therefore it is highly recommended to add Python code to your R code if it is totally necessary.

Though it is always a good thing to have an exquisite tool if you need to use Python and cannot separate processes. `reticulate` is currently that tool.

Joins with dplyr

Data Preparation

This part follows the notes of STAT 545 of Jenny Bryan

```r
library(tidyverse) ## dplyr provides the join functions
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.3     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
superheroes <- tibble::tribble(
    ~name, ~alignment, ~gender, ~publisher,
    "Magneto", "bad", "male", "Marvel",
    "Storm", "good", "female", "Marvel",
    "Mystique", "bad", "female", "Marvel",
    "Batman", "good", "male", "DC",
    "Joker", "bad", "male", "DC",
    "Catwoman", "bad", "female", "DC",
    "Hellboy", "good", "male", "Dark Horse Comics"
)

publishers <- tibble::tribble(
    ~publisher, ~yr_founded,
    "DC", 1934L,
    "Marvel", 1939L,
    "Image", 1992L
)
```

We have two data sets. Our first data set (superheroes) contains information about fictional characters from graphic novels. There are 7 characters from different publishers. Information about those characters include name, alignment (good or bad), gender (male or female) and the publishers where they belong to.

Our second data set is about publishers and the year they are founded. These two data sets are the perfect minimal example to introduce data table joins because they have a common (key) column and they have both similar and disparate values.

```
print(superheroes)
```

```
# A tibble: 7 x 4
  name     alignment gender publisher
  <chr>    <chr>     <chr>  <chr>
1 Magneto  bad       male   Marvel
2 Storm    good      female Marvel
3 Mystique bad       female Marvel
4 Batman   good      male   DC
5 Joker    bad       male   DC
6 Catwoman bad       female DC
7 Hellboy  good      male   Dark Horse Comics
```

```
print(publishers)
```

```
# A tibble: 3 x 2
  publisher yr_founded
  <chr>          <int>
1 DC              1934
2 Marvel          1939
3 Image           1992
```

Joins

We call data sets left and right (hence left join, right join), or X and Y (usually X denotes left and Y denotes right). Here we will use superheroes and publishers in different positions but usually superheroes will be on the left.

Left Join

In left join, all rows of X are preserved, only relevant rows Y and multiply rows if there are multiple matchings.

```r
left_join(superheroes, publishers, by = "publisher")
```

```
# A tibble: 7 x 5
  name     alignment gender publisher         yr_founded
  <chr>    <chr>     <chr>  <chr>                  <int>
1 Magneto  bad       male   Marvel                  1939
2 Storm    good      female Marvel                  1939
3 Mystique bad       female Marvel                  1939
4 Batman   good      male   DC                      1934
5 Joker    bad       male   DC                      1934
6 Catwoman bad       female DC                      1934
7 Hellboy  good      male   Dark Horse Comics         NA
```

Right join is the same as left join but the main data set is Y.

```r
right_join(superheroes, publishers, by = "publisher") %>% arrange(name)
```

```
# A tibble: 7 x 5
  name     alignment gender publisher yr_founded
  <chr>    <chr>     <chr>  <chr>          <int>
1 Batman   good      male   DC              1934
2 Catwoman bad       female DC              1934
```

```
3 Joker    bad       male   DC                1934
4 Magneto  bad       male   Marvel            1939
5 Mystique bad       female Marvel            1939
6 Storm    good      female Marvel            1939
7 <NA>     <NA>      <NA>   Image             1992
```

```
left_join(publishers, superheroes, by = "publisher") %>%
    arrange(name) %>%
    relocate(publisher, yr_founded, .after = gender)
```

```
# A tibble: 7 x 5
  name     alignment gender publisher yr_founded
  <chr>    <chr>     <chr>  <chr>          <int>
1 Batman   good      male   DC              1934
2 Catwoman bad       female DC              1934
3 Joker    bad       male   DC              1934
4 Magneto  bad       male   Marvel          1939
5 Mystique bad       female Marvel          1939
6 Storm    good      female Marvel          1939
7 <NA>     <NA>      <NA>   Image           1992
```

Inner Join

In inner join, only rows with common values are returned and rows are multiplied if there are multiple matchings.

```
inner_join(superheroes, publishers, by = "publisher")
```

```
# A tibble: 6 x 5
  name     alignment gender publisher yr_founded
  <chr>    <chr>     <chr>  <chr>          <int>
1 Magneto  bad       male   Marvel          1939
2 Storm    good      female Marvel          1939
3 Mystique bad       female Marvel          1939
4 Batman   good      male   DC              1934
5 Joker    bad       male   DC              1934
6 Catwoman bad       female DC              1934
```

Semi Join

Semi Join is very similar to inner join but without columns from Y.

```r
semi_join(superheroes, publishers, by = "publisher")
```

```
# A tibble: 6 x 4
  name     alignment gender publisher
  <chr>    <chr>     <chr>  <chr>
1 Magneto  bad       male   Marvel
2 Storm    good      female Marvel
3 Mystique bad       female Marvel
4 Batman   good      male   DC
5 Joker    bad       male   DC
6 Catwoman bad       female DC
```

Full Join

Full join returns all rows and columns from both X and Y and both multiplies multiple matchings and compensates for missing matches.

```r
full_join(superheroes, publishers, by = "publisher")
```

```
# A tibble: 8 x 5
  name     alignment gender publisher         yr_founded
  <chr>    <chr>     <chr>  <chr>                  <int>
1 Magneto  bad       male   Marvel                  1939
2 Storm    good      female Marvel                  1939
3 Mystique bad       female Marvel                  1939
4 Batman   good      male   DC                      1934
5 Joker    bad       male   DC                      1934
6 Catwoman bad       female DC                      1934
7 Hellboy  good      male   Dark Horse Comics         NA
8 <NA>     <NA>      <NA>   Image                   1992
```

Anti Join

Anti join returns all rows from X which do not have information (based on key column) in Y and returns only columns from X.

```r
anti_join(superheroes, publishers, by = "publisher")
```

```
# A tibble: 1 x 4
  name    alignment gender publisher
  <chr>   <chr>     <chr>  <chr>
1 Hellboy good      male   Dark Horse Comics
```

Website parsing with rvest

What is `rvest`?

`rvest` is a very useful R package to parse HTML files of web pages. Usually information is not directly provided in Excel files but embedded in one or more pages in a web site.

Main sources

- `rvest` main development page: https://rvest.tidyverse.org/
- Tutorial on rvest, httr and RSelenium

`rvest` Usage Examples

Since `rvest` is part of the tidyverse package we can easily use pipes with it.

```
library(tidyverse)
library(rvest)
```

In the first example, we will extract tables from BKM's (Interbank Card Center) sectoral development reports (see the website). Let's start with getting the web page.

```
the_url <- "https://bkm.com.tr/secilen-aya-ait-sektorel-gelisim/?filter_year=2020&filter_m
html_obj <- read_html(the_url)
```

When we check the object itself, we see it is a bunch of html code. We are on a good path.

```
html_obj
```

We can examine the html structure by using the powerful `html_structure()` function. Since it is a bit of verbose see for yourself.

```
html_obj %>% html_structure()
```

We can simply extract the table using `html_table()`. If you get an error about inconsistent fields, use parameter `fill=TRUE`.

```
[[1]]
# A tibble: 2 x 3
  X1                                X2                                X3
  <chr>                             <chr>                             <chr>
1 SEÇİLEN AYA AİT SEKTÖREL GELİŞİM Seçilen Aya Ait Sektörel Gelişim Seçilen Aya~
2 Seçilen Aya Ait Sektörel Gelişim <NA>                              <NA>

[[2]]
# A tibble: 1 x 1
  X1
  <chr>
1 Seçilen Aya Ait Sektörel Gelişim

[[3]]
# A tibble: 1 x 1
  X1
  <chr>
1 "Yıl:\n                    2023202220212020201920182017201620152014Ay:\n     ~

[[4]]
# A tibble: 29 x 5
   X1                                      X2                     X3    X4    X5
   <chr>                                   <chr>                  <chr> <chr> <chr>
 1 İşyeri Grubu                            İşlem Adedi            İşle~ "İşl~ "İşl~
 2 İşyeri Grubu                            İşlem Adedi(Kredi K~ İşle~ "İşl~ "İşl~
 3 ARABA KİRALAMA                          341.271                70.9~ "212~ "23,~
 4 ARAÇ KİRALAMA-SATIŞ/SERVİS/YEDEK PARÇA 3.497.791              969.~ "2.6~ "217~
 5 BENZİN VE YAKIT İSTASYONLARI            27.488.661             11.6~ "5.6~ "1.0~
 6 BIREYSEL EMEKLILIK                      2.078.609              2.043 "771~ "1,0~
 7 ÇEŞİTLİ GIDA                            33.777.181             21.5~ "5.2~ "1.1~
 8 DOĞRUDAN PAZARLAMA                      627.712                60.3~ "728~ "11,~
 9 EĞİTİM / KIRTASİYE / OFİS MALZEMELERİ  6.274.208              3.56~ "2.0~ "337~
10 ELEKTRİK-ELEKTRONİK EŞYA, BİLGİSAYAR   8.809.733              2.60~ "4.1~ "522~
# i 19 more rows
```

```
html_obj %>% html_table(fill = TRUE)
```

Now we are getting somewhere on the fourth item but it is not up to our quality standard. Let's deploy `dplyr` functions to make it better.

```
html_df <- read_html(the_url) %>%
    html_table(fill = TRUE) %>%
    `[[`(4)

html_df %>%
    # Since we do not have too many columns let's rename them manually
    # number (num) or value (val) of transactions (txn)
    # by credit card (cc) or debit card (dc)
    rename(category = 1, num_txn_cc = 2, num_txn_dc = 3, val_txn_cc = 4, val_txn_dc = 5) %
    # remove the first two rows because they are actually titles
    slice(-(1:2)) %>%
    # then convert every numeric value by using parse_number function from readr
    mutate(
        across(
            -category,
            ~ readr::parse_number(.,
                locale = readr::locale(decimal_mark = ",", grouping_mark = ".")
            )
        )
    )
```

```
# A tibble: 27 x 5
   category                              num_txn_cc num_txn_dc val_txn_cc val_txn_dc
   <chr>                                      <dbl>      <dbl>      <dbl>      <dbl>
 1 ARABA KİRALAMA                            341271      70947       212.      23.2
 2 ARAÇ KİRALAMA-SATIŞ/SERVİS/YEDEK~        3497791     969202      2676.     217.
 3 BENZİN VE YAKIT İSTASYONLARI            27488661   11642138      5608.    1060.
 4 BIREYSEL EMEKLILIK                       2078609       2043       772.       1.03
 5 ÇEŞİTLİ GIDA                            33777181   21569718      5284.    1124.
 6 DOĞRUDAN PAZARLAMA                        627712      60381       728.      11.2
 7 EĞİTİM / KIRTASİYE / OFİS MALZEM~        6274208    3565787      2044.     337.
 8 ELEKTRİK-ELEKTRONİK EŞYA, BİLGİS~        8809733    2609689      4185.     523.
 9 GİYİM VE AKSESUAR                       26750805   12487879      5670.    1957.
10 HAVAYOLLARI                              2013912     589812      2033.     755.
# i 17 more rows
```

In the second example we will harvest the links from Istanbul's Şehir Hatları (ferry line) domestic trips.

```
html_obj <- read_html("https://sehirhatlari.istanbul/en/timetables/domestic-trips")
```

Let's get all the links using "a" nodes and "href" attributes. We are looking for "domestic trips" links.

```
links_vec <- html_obj %>%
    html_nodes("a") %>%
    html_attr("href")
links_vec
```

```
 [1] "/en/corporate/privacy-and-cookie-policy-744"
 [2] "javascript:__doPostBack('ctl00$Cookie$btnCerezler','')"
 [3] NA
 [4] "/tr"
 [5] "/en"
 [6] "#"
 [7] "/en/corporate/sehir-hatlari-616"
 [8] "/en/corporate/presidents-message-720"
 [9] "/en/corporate/general-manager-721"
[10] "/en/corporate/vision-and-values-217"
[11] "/en/corporate/management-systems-policy-364"
[12] "#"
[13] "/en/timetables/domestic-trips"
[14] "/en/timetables/ferry-vehicle"
[15] "/en/timetables/bosphorus-tours"
[16] "/en/price-list"
[17] "/uploads/pdf/Sefer-Haritasi-Line-Map-2022.pdf"
[18] "#"
[19] "/en/information/disabled-guests-186"
[20] "/en/information/corporate-film-618"
[21] "/en/frequently-asked-questions"
[22] "#"
[23] "/en/corporate/rental-services-208"
[24] "/en/corporate/shipyard-services-617"
[25] "/en/corporate/advertising-areas-209"
[26] "/en/corporate/filming-210"
[27] "#"
[28] "/en/contact"
[29] "/en/applies"
[30] "https://twitter.com/sehir_hatlari"
[31] "https://www.facebook.com/sehirhatlari"
[32] "https://www.instagram.com/sehir_hatlari/"
[33] "https://www.youtube.com/user/sehirhatlari1"
[34] NA
```

```
[35] "/tr"
[36] "#"
[37] "#"
[38] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines-26"
[39] "/en/timetables/domestic-trips/bosphorus-lines-52"
[40] "/en/timetables/domestic-trips/adalar-princes-islands-lines-176"
[41] "/en"
[42] "/en/timetables"
[43] "#ichatlartarifeleri"
[44] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines-26"
[45] "/en/timetables/domestic-trips/bosphorus-lines-52"
[46] "/en/timetables/domestic-trips/adalar-princes-islands-lines-176"
[47] "/en/timetables/ferry-vehicle"
[48] "/en/timetables/bosphorus-tours"
[49] "#ucrettarifeleri"
[50] "/en/price-list/ferry-lines-79"
[51] "/en/price-list/bosphorus-tours-78"
[52] "/en/price-list/ferry-vehicle-line-77"
[53] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/kadikoy-kabatas-766"
[54] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/kadikoy-karakoy-eminonu-163"
[55] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/uskudar-karakoy-eminonu-164"
[56] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/kadikoy-besiktas-165"
[57] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/halic-hatti-37"
[58] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/kadikoy-karakoy-besiktas-768
[59] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/bostanci-karakoy-kabatas-166
[60] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/asiyan-anadolu-hisari-kucuksu
[61] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/uskudar-asiyan-2015"
[62] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/kadikoy-kasimpasa-fener-hask
[63] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/besiktas-kabatas-karakoy-kas
[64] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/cengelkoy-kabatas-line-2021"
[65] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/bostanci-moda-kadikoy-kabata
[66] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/avcilar-bakirkoy-yenikapi-ka
[67] "/en/timetables/domestic-trips/inner-istanbul-ferry-lines/avcilar-bostanci-line-2028"
[68] "/en/timetables/domestic-trips/bosphorus-lines/to-bosphorus-from-bosphorus-167"
[69] "/en/timetables/domestic-trips/bosphorus-lines/anadolu-kavagi-rumeli-kavagi-sariyer-16
[70] "/en/timetables/domestic-trips/bosphorus-lines/kucuksu-besiktas-kabatas-169"
[71] "/en/timetables/domestic-trips/bosphorus-lines/cengelkoy-istinye-170"
[72] "/en/timetables/domestic-trips/bosphorus-lines/kadikoy-sariyer-171"
[73] "/en/timetables/domestic-trips/bosphorus-lines/anadolu-kavagi-uskudar-172"
[74] "/en/timetables/domestic-trips/bosphorus-lines/ortakoy-uskudar-kadikoy-173"
[75] "/en/timetables/domestic-trips/bosphorus-lines/rumeli-kavagi-eminonu-174"
[76] "/en/timetables/domestic-trips/bosphorus-lines/kucuksu-istinye-175"
[77] "/en/timetables/domestic-trips/bosphorus-lines/bebek-emirgan-595"
```

```
 [78] "/en/timetables/domestic-trips/bosphorus-lines/beykoz-sariyer-767"
 [79] "/en/timetables/domestic-trips/bosphorus-lines/ortakoy-besiktas-eminonu-2078"
 [80] "/en/timetables/domestic-trips/adalar-princes-islands-lines/kabatas-adalar-177"
 [81] "/en/timetables/domestic-trips/adalar-princes-islands-lines/adalar-besiktas-769"
 [82] "/en/timetables/domestic-trips/adalar-princes-islands-lines/bostanci-adalar-circuit-li
 [83] "/en/timetables/domestic-trips/adalar-princes-islands-lines/maltepe-buyukada-heybeliad
 [84] "/uploads/dokuman/46-2/tarife.pdf"
 [85] "https://www.ibb.istanbul"
 [86] "tel:+153"
 [87] "http://ibb.tv/yayin"
 [88] "/en/corporate/sehir-hatlari-616"
 [89] "/en/corporate/presidents-message-720"
 [90] "/en/corporate/general-manager-721"
 [91] "/en/corporate/vision-and-values-217"
 [92] "/en/corporate/management-systems-policy-364"
 [93] "/en/ferries"
 [94] "/en/piers"
 [95] "/en/journeys/domestic-trips"
 [96] "/en/journeys/bosphorus-tours"
 [97] "/en/journeys/ferry-vehicle"
 [98] "/en/price-list"
 [99] "/en/information/disabled-guests-186"
[100] "en/information/corporate-film-618"
[101] "/en/frequently-asked-questions"
[102] "/en/corporate/filming-210"
[103] "/en/corporate/advertising-areas-209"
[104] "/en/corporate/rental-services-208"
[105] "/en/corporate/shipyard-services-617"
[106] "/en/contact"
[107] "/en/applies"
[108] "http://www.sehirhatlari.com.tr/sanaltur/index.htm"
[109] "https://itunes.apple.com/tr/app/%C5%9Fehir-hatlar%C4%B1/id783673371?l=tr&mt=8"
[110] "https://play.google.com/store/apps/details?id=com.spexco.flexcoder2.sehirhatlari.acti
[111] "https://www.facebook.com/sehirhatlari"
[112] "https://www.instagram.com/sehir_hatlari/"
[113] "https://twitter.com/sehir_hatlari"
[114] "https://www.youtube.com/user/sehirhatlari1"
[115] "https://www.trios.com.tr"
```

With a simple regex and adding the root url we can get all the relevant links from the web page.

```r
domestic_trips_links <- paste0("https://sehirhatlari.istanbul", links_vec[grepl("/domestic
domestic_trips_links
```

```
 [1] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines-
 [2] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines-52"
 [3] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines
 [4] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines-
 [5] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines-52"
 [6] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines
 [7] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
 [8] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
 [9] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[10] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[11] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[12] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[13] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[14] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[15] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[16] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[17] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[18] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[19] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[20] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[21] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/
[22] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/to-bosphorus
[23] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/anadolu-kava
[24] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/kucuksu-besi
[25] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/cengelkoy-is
[26] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/kadikoy-sari
[27] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/anadolu-kava
[28] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/ortakoy-usku
[29] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/rumeli-kavag
[30] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/kucuksu-isti
[31] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/bebek-emirga
[32] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/beykoz-sariy
[33] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines/ortakoy-besi
[34] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines
[35] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines
[36] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines
[37] "https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines
```

You can click all the links below.

```
paste0("+ ", domestic_trips_links, " <br>", collapse = " ")
```

[1] “+ https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-
lines-26 + https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines-52 +
https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines-176
+ https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines-
26 + https://sehirhatlari.istanbul/en/timetables/domestic-trips/bosphorus-lines-52 +
https://sehirhatlari.istanbul/en/timetables/domestic-trips/adalar-princes-islands-lines-176 +
https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-lines/kadikoy-
kabatas-766 + https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-
ferry-lines/kadikoy-karakoy-eminonu-163 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/inner-istanbul-ferry-lines/uskudar-karakoy-eminonu-164 + https://sehirhatlari.istanbul/en/timetables/do
trips/inner-istanbul-ferry-lines/kadikoy-besiktas-165 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/inner-istanbul-ferry-lines/halic-hatti-37 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/inner-istanbul-ferry-lines/kadikoy-karakoy-besiktas-768 + https://sehirhatlari.istanbul/en/timetables/dom
trips/inner-istanbul-ferry-lines/bostanci-karakoy-kabatas-166 + https://sehirhatlari.istanbul/en/timetables/dom
trips/inner-istanbul-ferry-lines/asiyan-anadolu-hisari-kucuksu-circuit-line-2014 + https://sehirhatlari.istanbul/e
trips/inner-istanbul-ferry-lines/uskudar-asiyan-2015 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/inner-istanbul-ferry-lines/kadikoy-kasimpasa-fener-haskoy-sutluce-eyupsultan-line-
2017 + https://sehirhatlari.istanbul/en/timetables/domestic-trips/inner-istanbul-ferry-
lines/besiktas-kabatas-karakoy-kasimpasa-sutluce-eyupsultan-line-2019 + https://sehirhatlari.istanbul/en/time
trips/inner-istanbul-ferry-lines/cengelkoy-kabatas-line-2021 + https://sehirhatlari.istanbul/en/timetables/dome
trips/inner-istanbul-ferry-lines/bostanci-moda-kadikoy-kabatas-line-2024 + https://sehirhatlari.istanbul/en/tim
trips/inner-istanbul-ferry-lines/avcilar-bakirkoy-yenikapi-kadikoy-line-2027 + https://sehirhatlari.istanbul/en/t
trips/inner-istanbul-ferry-lines/avcilar-bostanci-line-2028 + https://sehirhatlari.istanbul/en/timetables/domest
trips/bosphorus-lines/to-bosphorus-from-bosphorus-167 + https://sehirhatlari.istanbul/en/timetables/domestic
trips/bosphorus-lines/anadolu-kavagi-rumeli-kavagi-sariyer-168 + https://sehirhatlari.istanbul/en/timetables/d
trips/bosphorus-lines/kucuksu-besiktas-kabatas-169 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/cengelkoy-istinye-170 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/kadikoy-sariyer-171 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/anadolu-kavagi-uskudar-172 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/ortakoy-uskudar-kadikoy-173 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/rumeli-kavagi-eminonu-174 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/kucuksu-istinye-175 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/bebek-emirgan-595 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/beykoz-sariyer-767 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/bosphorus-lines/ortakoy-besiktas-eminonu-2078 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/adalar-princes-islands-lines/kabatas-adalar-177 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/adalar-princes-islands-lines/adalar-besiktas-769 + https://sehirhatlari.istanbul/en/timetables/domestic-
trips/adalar-princes-islands-lines/bostanci-adalar-circuit-line-770 + https://sehirhatlari.istanbul/en/timetables/

trips/adalar-princes-islands-lines/maltepe-buyukada-heybeliada-burgazada-kinaliada-lines-2020 ”

Exercises

- Collect multiple periods from BKM page and create an analysis.
- Collect all timetables from Şehir Hatları domestic lines and create a timetable Shiny app.

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.