Introduction to Data Science Topic-2

- Instructor: Professor Henry Horng-Shing Lu,

  Institute of Statistics, National Yang Ming Chiao Tung University, Taiwan

  Email: henryhslu@nycu.edu.tw
- WWW: http://misg.stat.nctu.edu.tw/hslu/course/DataScience.htm
- Classroom: ED B27 (新竹市大學路1001號工程四館B27教室)
- References:

  M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.

  K.-T. Tsai, Machine Learning for Knowledge Discovery with R: Methodologies for Modeling, Inference, and Prediction, 2021, Chapman and Hall/CRC.
- Evaluation: Homework: 70%, Term Project: 30%
- Office hours: By appointment

Grading Policy

- Evaluation will be based on the grade of 10 homework (70%) and the Final project (30%).
- The score of every assignment is graded by TAs first and then confirmed by the Tutor.
- The range of scores will be in the range of 70-100 (based on individual performance).
- We will not do any normalization of grades.
- We will not grade any homework that is submitted after the deadline unless there are any valid reason for late submission.

Course Outline

10 Topics and 10 Homeworks:

- Introduction of Data Science
- Introduction of R and Python
- Getting Data into R and Python
- Data Visualization
- Exploratory Data Analysis
- Regression (Supervised Learning)
- Classification (Supervised Learning)
- Text Mining
- Clustering (Unsupervised Learning)
- Neural Network and Deep Learning

Topic 2: Introduction to R

- **References:**

Ch. 2, M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.

https://www.r-project.org/

https://www.statmethods.net/input/datatypes.html

https://www.kaggle.com/learn

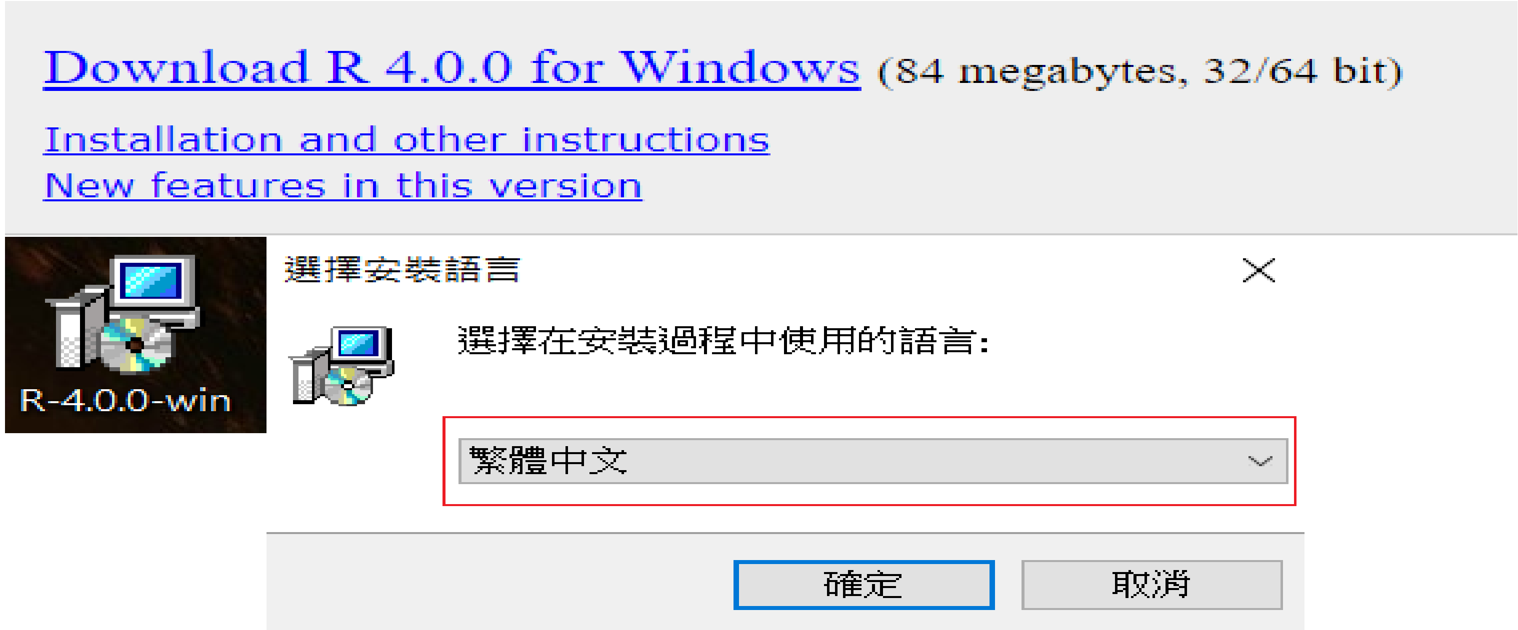
Demo:

R demo: https://www.kaggle.com/code/dongdongxzoez/r-topic-2

Python demo: https://www.kaggle.com/code/dongdongxzoez/python-topic-2

R Download

https://cran.r-project.org/bin/windows/base/ or https://rstudio-education.github.io/hopr/starting.html
And https://cran.r-project.org/bin/macosx/



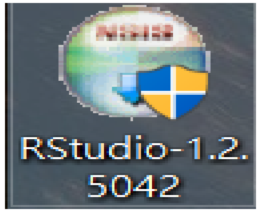
R Window

RGui - [R Console]
檔案 編輯 看 其它 程式套件 視窗 輔助
R version 2.5.1 (2007-06-27)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
R 是免費軟體，不提供任何擔保。
在某些條件下您可以將其自由散布。
用 'license()' 或 'licence()' 來獲得散布的詳細條件。
R 是個合作計劃，有許多人為之做出了貢獻。
用 'contributors()' 來看詳細的情況並且
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。
用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。
用 'q()' 離開 R。
>
Interactive command window

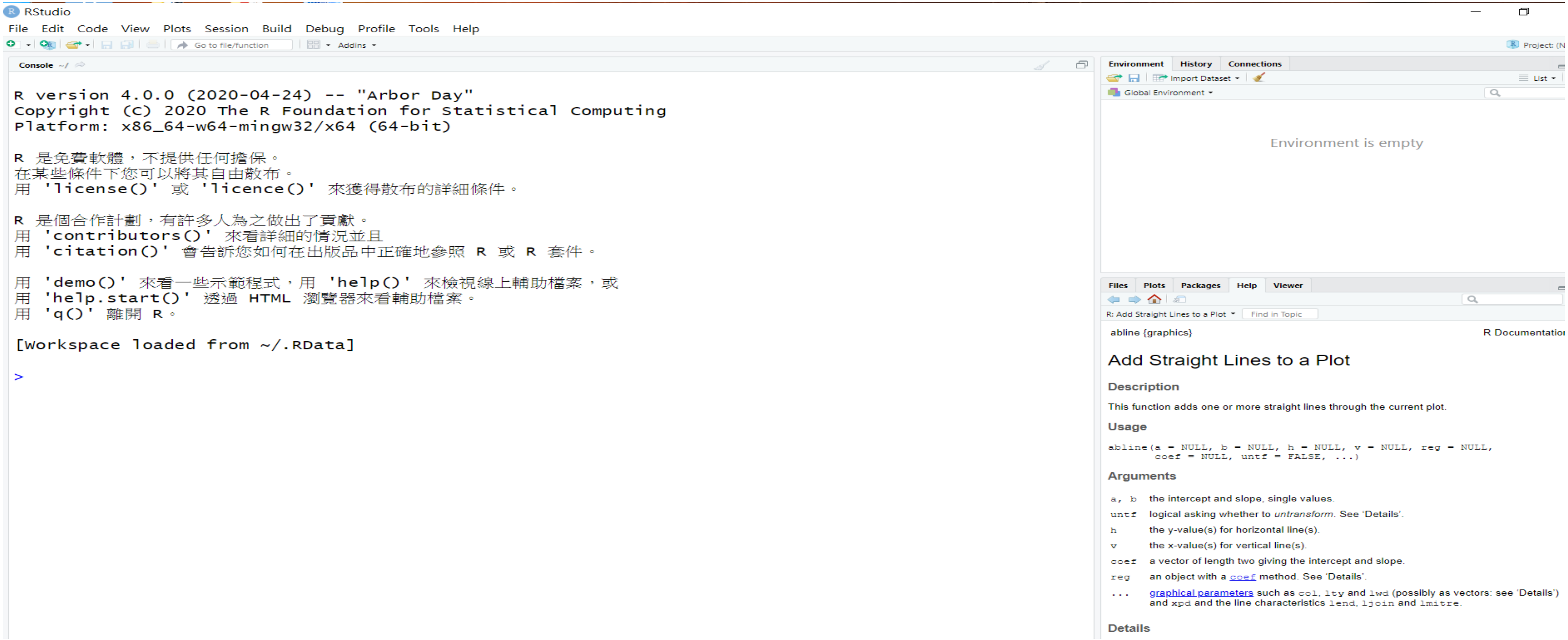
RStudio Download

https://rstudio.com/products/rstudio/download/

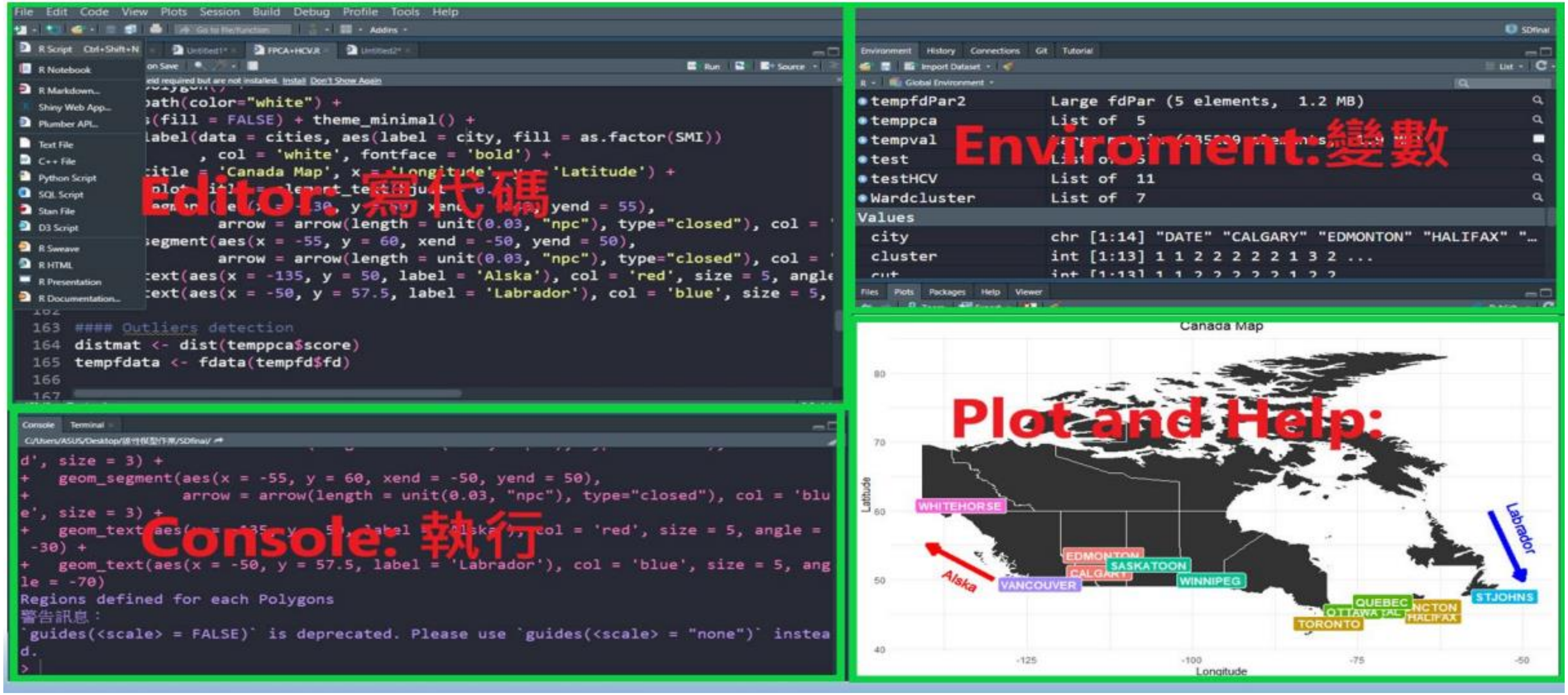
OS	Download	Size	SHA-256
Windows 10/8/7	RStudio-1.2.5042.exe	149.84 MB	5d4cd644
macOS 10.13+	RStudio-1.2.5042.dmg	126.89 MB	74ea68eb
Ubuntu 14/Debian 8	rstudio-1.2.5042-amd64.deb	96.41 MB	485e2757
Ubuntu 16	rstudio-1.2.5042-amd64.deb	104.07 MB	e2f15cc2
Ubuntu 18/Debian 10	rstudio-1.2.5042-amd64.deb	104.93 MB	99e0f57b
Fedora 19/Red Hat 7	rstudio-1.2.5042-x86_64.rpm	119.75 MB	5ab559e2
Fedora 28/Red Hat 8	rstudio-1.2.5042-x86_64.rpm	120.39 MB	cb962044
Debian 9	rstudio-1.2.5042-amd64.deb	105.40 MB	92684c84
SLES/OpenSUSE 12	rstudio-1.2.5042-x86_64.rpm	98.88 MB	a419cef8
OpenSUSE 15	rstudio-1.2.5042-x86_64.rpm	106.56 MB	c050eb25



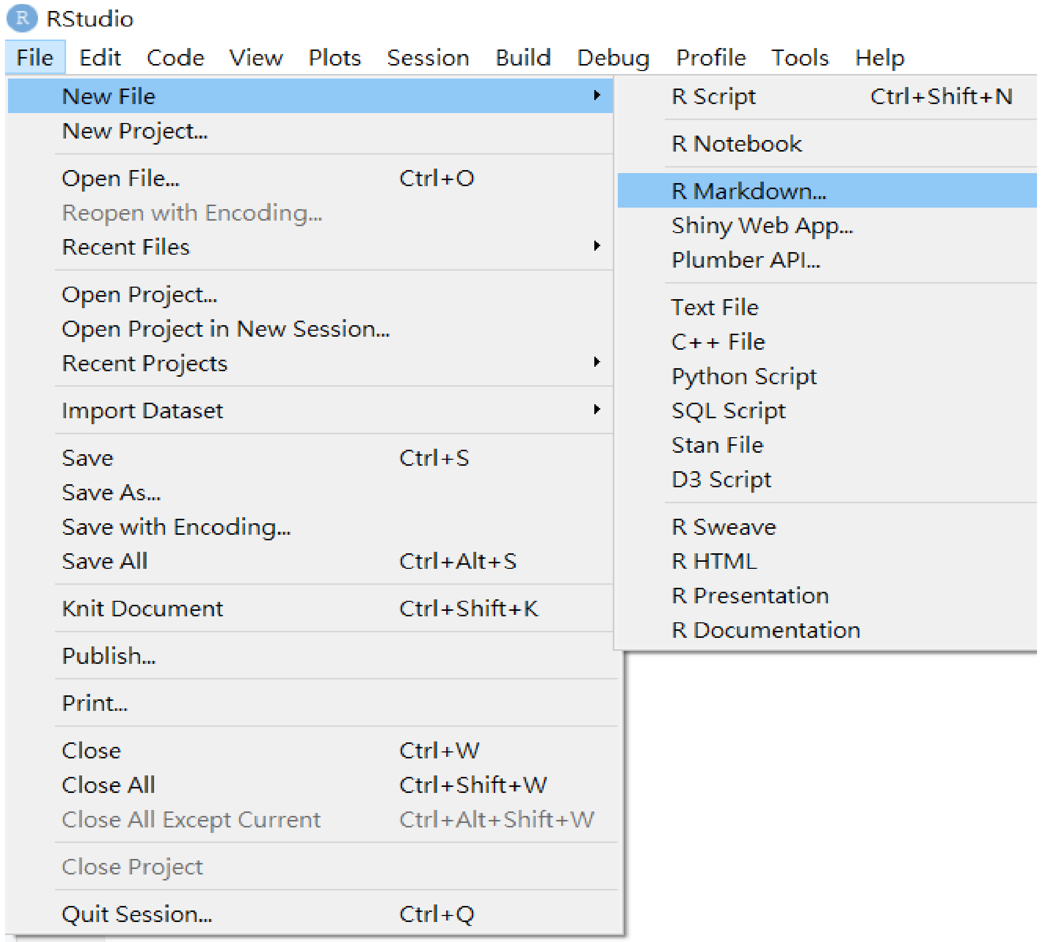
RStudio


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (N
Console ~/
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R 是免費軟體，不提供任何擔保。
在某些條件下您可以將其自由散布。
用 'license()' 或 'licence()' 來獲得散布的詳細條件。
R 是個合作計劃，有許多人為之做出了貢獻。
用 'contributors()' 來看詳細的情況並且
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。
用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。
用 'q()' 離開 R。
[Workspace loaded from ~/.RData]
>
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
R: Add Straight Lines to a Plot
Find in Topic
abline {graphics}
R Documentatio
Add Straight Lines to a Plot
Description
This function adds one or more straight lines through the current plot.
Usage
abline(a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL, coef = NULL, untf = FALSE, ...)
Arguments
a, b the intercept and slope, single values.
untf logical asking whether to untransform. See 'Details'.
h the y-value(s) for horizontal line(s).
v the x-value(s) for vertical line(s).
coef a vector of length two giving the intercept and slope.
reg an object with a coef method. See 'Details'.
... graphical parameters such as col, lty and lwd (possibly as vectors: see 'Details') and xpd and the line characteristics lend, ljoin and lmitre.
Details

Interface


Editor: 寫代碼
Enviroment:變數
Plot and Help:
Console: 執行
Canada Map
Latitude
Longitude
WHITEHORSE
EDMONTON
CALGARY
SASKATOON
VANCOUVER
WINNIPEG
QUEBEC
TORONTO
STJOHNS
Alska
Labrador

R Markdown


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
New File
New Project...
Open File... Ctrl+O
Reopen with Encoding...
Recent Files
Open Project...
Open Project in New Session...
Recent Projects
Import Dataset
Save Ctrl+S
Save As...
Save with Encoding...
Save All Ctrl+Alt+S
Knit Document Ctrl+Shift+K
Publish...
Print...
Close Ctrl+W
Close All Ctrl+Shift+W
Close All Except Current Ctrl+Alt+Shift+W
Close Project
Quit Session... Ctrl+Q
R Script Ctrl+Shift+N
R Notebook
R Markdown...
Shiny Web App...
Plumber API...
Text File
C++ File
Python Script
SQL Script
Stan File
D3 Script
R Sweave
R HTML
R Presentation
R Documentation


Knit
Insert
Run
---
title: "Untitled"
output: html_document

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
R Markdown
This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:
```{r cars}
summary(cars)
```

Hot Keys

- Source: compile R file
- New File:
  - R script: .R file , R code
  - R markdown: .Rmd file , R + Markdown
- Editor:
  - Ctrl + F → replace
  - Ctrl + Enter → compile the row you specify
  - Ctrl + S → save file

Set your working directory

```
---
title: "Topic-2 Introduction to Data Science"
author: "TA Kiran"
date: "2023-09-11"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

Set your working directory

```{r}
setwd("D:/TA Introducing the data Science/2023 notes/Topic-2")
getwd() # will print out the current directory
cat("Current Working Directory:", getwd())

```
```

```
[1] "D:/TA Introducing the data Science/2023 notes/Topic-2"
Current Working Directory: D:/TA Introducing the data Science/2023 notes/Topic-2
```

Read or Write File

```r
#Importing the .txt file
```{r}
iris = read.table(file = 'iris.txt', sep = '', header = TRUE)
head(iris, 6)
```
```

	Species <chr>	SepalLength <int>	SepalWidth <int>	PetalLength <int>	PetalWidth <int>
1	Setosa	50	33	14	2
2	Setosa	46	34	14	3
3	Setosa	46	36	10	2
4	Setosa	51	33	17	5
5	Setosa	55	35	13	2
6	Setosa	48	31	16	2

6 rows

we can also import by csv file

- `read.csv(file, header, fileEncoding = "UTF-8")`
  - file: file path
  - header: the first line of csv file is title or not
  - fileEncoding: file encoding format

```{r}
iris = read.csv(file = 'iris.csv', sep=',')
head(iris, 6)
```

	Species <chr>	SepalLength <int>	SepalWidth <int>	PetalLength <int>	PetalWidth <int>
1	Setosa	50	33	14	2
2	Setosa	46	34	14	3
3	Setosa	46	36	10	2
4	Setosa	51	33	17	5
5	Setosa	55	35	13	2
6	Setosa	48	31	16	2

6 rows

Read or Write File

```
**We can use write.csv or write.table to write data into .csv file**

```{r}
write.csv(iris, 'iris_from_csv.csv')
write.table(iris, 'iris_from_table.csv')
```
```

Check Type and Structure of the Data

```{r}
class(iris)
str(iris)
```

```
[1] "data.frame"
'data.frame':	150 obs. of  5 variables:
 $ Species    : chr  "Setosa" "Setosa" "Setosa" "Setosa" ...
 $ SepalLength: int  50 46 46 51 55 48 52 49 44 50 ...
 $ SepalWidth : int  33 34 36 33 35 31 34 36 32 35 ...
 $ PetalLength: int  14 14 10 17 13 16 14 14 13 16 ...
 $ PetalWidth : int  2 3 2 5 2 2 2 1 2 6 ...
```

R - Operators

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Arithmetic Operator

In [14]:

```
x <- c(1,5,3)
y <- c(6,5,2)
x + y
x - y
x / y
x * y # elementwise multiply
x %% y # remainder
x %/% y # quotient
x %*% y # inner product
matrix(1:4, nrow=2) %*% matrix(1:4, nrow=2)
x^y # power
x < y # > 、= 、!= 、>= 、<=
```

Statistics Operation

In [17]:

```
x <- c(1,2,3)
y <- c(2,4,5)
sqrt(x)
exp(x)
log(x)
max(x)
min(x)
which.max(x)
which.min(x)
sort(x)
sd(x)
var(x)
unique(x)
quantile(x)
rank(x)
rev(x)
cor(x, y)
```

Other commands

- NA : Non avaliable
- Inf : Infinity
- NaN : Not a number (0/0)
- Null : Null

In [18]:

```
x <- c(1,4,NA,Inf,NaN)
y <- c(1,4,5,9,7,5,6,8)
x %in% y
is.na(x)
complete.cases(x)
```

TRUE · TRUE · FALSE · FALSE · FALSE

Data Type and Structure

- Before we go further into the coding, Let's give a brief intro about R data structure.

- R is an Object Oriented Language. R use vector as the basis of any object in R, the following show the six most commonly used object.

1) Vectors
2) Matrices
3) Arrays
4) Data Frames
5) Lists
6) Factors

Vectors

````
**vector**

```{r}
a <- c(1, 2, 5, 6, -2.4) #numeric vector
a
b <- c('Student', 'TA'); b #character vector
c <- c(TRUE, TRUE, FALSE, TRUE); c #logical vector
vec1 <- c('TRUE', 'FALSE') # character vector
vec2 <- factor(b, levels = c("TA", "Student")) # factor
vec2
```
````

```
[1]  1.0  2.0  5.0  6.0 -2.4
[1] "Student" "TA"
[1]  TRUE  TRUE FALSE  TRUE
[1] Student TA
Levels: TA Student
```

````
- We can named a vector
```{r}
names(a) <- c('x', 'y', 'z')
length(a)
a
```
````

```
[1] 3
   x    y    z
 1.0  2.0 -2.4
```

Commonly used function

- `seq(from, to, by)`
  - from: head of a sequence
  - to: tail of a sequence
  - by: step of a sequence

- `rep(x, each)`
  - x: enter vector
  - each: times for repeat

```
```{r}
seq(from=1, to=29, by=2) # creating the sequence
seq(from=1, to=29, by=0.5)
1:29 #same as seq(1, 29, 1)
rev(1:29) #In the R programming language, the rev() function is used to reverse the order of elements in a vector.
rep(c(0, 1), 5) #the rep() function is used to replicate values or sequences a specified number of times.
```
```

```
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0
[26] 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5
[51] 26.0 26.5 27.0 27.5 28.0 28.5 29.0
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
 [1] 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
 [1] 0 1 0 1 0 1 0 1 0 1
```

Matrix

$$
\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & \cdots & n \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} &
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
a_{31} & a_{32} & \cdots & a_{3n} \\
\vdots & \vdots & \vdots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix}
\end{array}
$$

```
##matrix

```{r}
x <- matrix(1:20, nrow = 5, ncol = 4,byrow=F) ; x #create 5*4 matrix
y <- matrix(1:20, nrow = 5, ncol = 4, byrow = T) ; y #fill the matrix by row
#finding the number of elements in the matrix
length(x)
#finding the dimension of the matrix
dim(x)
```
```

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

```
**Identify rows, columns or elements using subscripts**

```{r}
x[1,] #first row
x[,1] #first column
x[5,2]
x[2:4, 1:3]  # creating a sub-matrix from matrix x

# combine two matrix two matrix
#In R, the cbind() function is used to combine vectors or matrices column-wise (i.e., binding them by columns).
## Create two vectors
matrix1 <- c(1,2,3,4)
matrix2 <- c(5,6,7,8)
# Combine them using cbind
cbind(matrix1, matrix2)
```
```

```
[1]  1  6 11 16
[1] 1 2 3 4 5
[1] 10
     [,1] [,2] [,3]
[1,]    2    7   12
[2,]    3    8   13
[3,]    4    9   14
     matrix1 matrix2
[1,]       1       5
[2,]       2       6
[3,]       3       7
[4,]       4       8
```

Commonly used function

- t : transpose
- %*% : multiply
- diag : diagonal matrix
- det : determinant of a matrix
- solve : inverse of a matrix
- eigen : eigenvectors and eignevalues for a matrix

```{r}
mat1 <- matrix(1:4, nrow = 2, ncol = 2, byrow = T)
mat1
mat2 <-  matrix(5:8, nrow = 2, ncol = 2, byrow = T)
mat2
#Matrix Addition
mat1 + mat2
#transpose
t(mat1)
#matrix multiplication
mat1 %*% mat2

diag(c(2,-5))
det(mat1)
solve(diag(c(2,-5)))
eigen(mat1)
```

Arrays

Arrays:

1. **Homogeneous Data Types**: Arrays are designed to store data of the same data type. This means that all elements within an array must be of the same type, typically numeric.
2. **Multi-Dimensional**: Arrays can be multi-dimensional, meaning they can have more than one dimension, such as rows and columns (2D arrays) or rows, columns, and layers (3D arrays).
3. **Fixed Dimensions**: Once you create an array with a specific number of dimensions, you cannot change those dimensions. You must recreate the array if you want to change its shape.
4. **Optimized for Numeric Data**: Arrays are optimized for numerical computations and are commonly used for tasks like matrix algebra, statistical analysis, and working with data frames.
5. **Created with the `array()` Function**: You typically create arrays using the `array()` function and specify the dimensions and data.

- Arrays are similar to matrices but can have more than two dimensions.
- Ex.

```
> # Arrays
> x = array(1:24, c(2,3,4));x
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

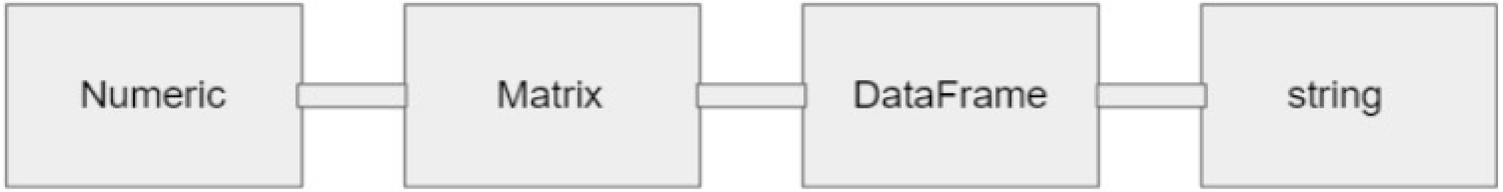
Lists

Lists:

1. **Heterogeneous Data Types**: Lists can store elements of different data types, including numbers, characters, vectors, other lists, and more. This makes lists more versatile for handling diverse data.
2. **One-Dimensional**: Lists are one-dimensional structures, meaning they do not have multiple dimensions like arrays.
3. **Dynamic Structure**: Lists can grow or shrink dynamically. You can add or remove elements from a list without needing to recreate it, which makes them flexible for various data manipulation tasks.
4. **Created with the `list()` Function**: You create lists using the `list()` function. You can specify the elements you want to include in the list.

Lists

- Lists are a fundamental data structure in R.
- They can store elements of different data types, including numbers, characters, vectors, other lists, and more.
- Lists are created using the list() function.



List can contain all the data type, including `List` , we use $ operator to extract the attribute

```
###array or Lists
```{r}
x <- array(1:24, dim = c(2, 3, 4));  #create 2 * 3 * 4 array
y = array(1:24, dim = c(2, 3, 4));

list1 <- list(
  numeric = c(0,1,2),
  bool = c(TRUE, FALSE),
  mtx = matrix(1:16, nrow=4)
)
list1$mtx
list1$bool
```
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
[1]  TRUE FALSE
```

Lists - Identify elements using [[]]

```{r}
###List
w <- list(name = "kiran", cost = c(5, 10, 15), age = 20)
w[[2]] #select component by index
w[["cost"]] #select component by name

```

```
[1]  5 10 15
[1]  5 10 15
```

A simple comparison between Arrays and Lists

Here's a simple comparison:

- Use arrays when you have homogeneous numerical data and need to perform numerical computations efficiently.
- Use lists when you have heterogeneous data types, need flexibility in the structure, or when you want to store different types of objects in a single container.

In summary, the choice between arrays and lists in R depends on your specific data and analysis requirements. Arrays are more suitable for numeric data and computations, while lists are more versatile and can accommodate a wider range of data types and structures.

Data Frames

KS Explain Data Frame In R?

In R, a data frame is a fundamental data structure used for storing and organizing data in a tabular format, similar to a spreadsheet or a database table. It is a two-dimensional data structure where data is organized into rows and columns. Each column can contain data of different types, such as numeric, character, factor, or logical, making data frames a versatile way to work with heterogeneous data.

Here are some key characteristics and concepts related to data frames in R:

1. **Tabular Structure**: A data frame is essentially a table with rows and columns, where each column represents a variable and each row represents an observation or record.
2. **Column Names**: Data frames have column names, which are used to label and identify each variable (column) in the data frame. You can access and manipulate columns using these names.
3. **Data Types**: Unlike matrices, data frames can hold columns with different data types. This flexibility allows you to work with mixed data, such as a combination of numbers, text, and factors.
4. **Homogeneous Columns**: Each column within a data frame must be of the same length. This means that all columns should have the same number of elements.
5. **Indexing**: Data frames support both row and column indexing. You can access specific rows and columns using row and column indices or by using column names.
6. **Importing Data**: Data frames are commonly used to import and work with external data sources like CSV files, Excel spreadsheets, or database tables. R provides various functions, such as `read.csv()`, `read.table()`, and others, to read data into data frames.

Data Frames

The "frame" to store data, and the attribute should be equal in length, and the data type between attributes can differ, however, the data type within attributes should be equal.

- Ex.

```
> # Data Frames
> d <- c(1,2,3,4)
> e <- c("red", "white", "red", NA)
> f <- c(TRUE, TRUE, TRUE, FALSE)
> myData <- data.frame(d,e,f)
> names(myData) <- c("ID", "Color", "Passed") # variable names
> myData
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Data Frames - Identify the elements

- Ex.

```
> # identify the elements
> myData[2:3] # columns 2,3 of data frame
  Color Passed
1   red   TRUE
2 white   TRUE
3   red   TRUE
4  <NA>  FALSE
> myData[c("ID", "Passed")] # columns ID and Passed from data frame
  ID Passed
1  1   TRUE
2  2   TRUE
3  3   TRUE
4  4  FALSE
> myData$ID # Variable ID in the data frame
[1] 1 2 3 4
```

Example of Data Frame using ChatGPT

KS

create an example of Data Frame in R?

Certainly! Here's an example of creating a data frame in R with some sample data:

R   Copy code

```r
# Creating a data frame with sample data
data <- data.frame(
  StudentID = c(1, 2, 3, 4, 5),
  FirstName = c("Alice", "Bob", "Charlie", "David", "Eve"),
  LastName = c("Smith", "Johnson", "Brown", "Lee", "White"),
  Age = c(22, 25, 23, 24, 21),
  GPA = c(3.8, 3.6, 3.9, 3.5, 3.7)
)

# Printing the data frame
print(data)
```

In this example, we create a data frame named `data` with five columns: StudentID, FirstName, LastName, Age, and GPA. Each column contains sample data for five students. The `c()` function is used to create vectors for each column, and `data.frame()` is used to combine these vectors into a data frame.

When you run this code, you'll see the following output:

mathematica   Copy code

```
  StudentID FirstName LastName Age GPA
1         1     Alice    Smith  22 3.8
2         2       Bob  Johnson  25 3.6
3         3   Charlie    Brown  23 3.9
4         4     David      Lee  24 3.5
5         5       Eve    White  21 3.7
```

This is a simple data frame containing information about five students, including their student IDs, first names, last names, ages, and GPAs. You can perform various operations and analyses on this data frame in R.

Factors

KS what is Factors in R/

In R, a factor is a data type used to represent categorical or nominal data. Categorical data consists of distinct categories or levels, often representing attributes that do not have a natural order or ranking. Factors are used to store and work with such categorical data efficiently. Factors are an essential part of data analysis and statistical modeling in R.

Here are some key characteristics and concepts related to factors in R:

1. **Levels**: Factors are built on top of an integer vector, where each integer corresponds to a level or category. The levels represent the distinct categories of the categorical variable.
2. **Ordered and Unordered Factors**: Factors can be either ordered or unordered. Ordered factors are used when the levels have a natural order or ranking, such as "low," "medium," and "high." Unordered factors are used when there is no inherent order among the levels, such as "red," "green," and "blue."
3. **Creating Factors**: You can create factors using the `factor()` function in R. For example:

R Copy code

```r
# Creating an unordered factor
color <- factor(c("red", "green", "blue", "red", "green"))

# Creating an ordered factor
size <- factor(c("small", "medium", "large"), ordered = TRUE)
```

4. **Viewing Levels**: You can view the levels of a factor using the `levels()` function:

Factors

Example

```
gender <- c(rep("male", 20), rep("female", 30)); gender
gender <- factor(gender)
summary(gender)
```

'male' · 'female' · 'female'

female: 30 **male:** 20

Check Data Type: Class Function

class(): We identify the data type of a variable using the class() function.

- Ex.

```
> x = 5
> class(x)
[1] "numeric"
```

Fancy Indexing

There are some tips for you to get familiar with R index system

1. R has a different index system, which start from 1, not 0.
2. If you want to select multiple indices, you should turn index into sequence
3. If you want to remove element, you should use negative index

Some interesting fact:

1. If the language is based on Fourtan, the index start from 1
2. If the language is based on C, the index start from 0

Data Type Transformation

Transform	isinstance
`as.numeric()`	`is.numeric()`
`as.matrix()`	`is.matrix()`
`as.data.frame()`	`is.data.frame()`
`as.array()`	`is.array()`
`as.table()`	`is.table()`
`as.function()`	`is.function()`
`as.na()`	`is.na()`
`as.null()`	`is.null()`
`as.list()`	`is.list()`
`as.logical()`	`is.logical()`
`as.factor()`	`is.factor()`

Example of Data Type Transformation

`as.numeric()` and `is.numeric()` are two distinct functions in R that serve different purposes and return different results:

1. `as.numeric()`:
   - Purpose: `as.numeric()` is used to explicitly convert an R object to a numeric (numeric vector) data type. It attempts to coerce the input object into a numeric type. If successful, it returns a numeric vector; otherwise, it returns `NA` for elements that cannot be converted to numeric.
   - Example:

```R
x <- c("1", "2", "3", "abc")
numeric_x <- as.numeric(x)
print(numeric_x)
```

Output:

```csharp
[1]  1  2  3 NA
```

   - Note: In the example above, the `as.numeric()` function converted the character elements "1", "2", and "3" to numeric values, but it couldn't convert "abc," so it assigned `NA` to that element.

2. `is.numeric()`:
   - Purpose: `is.numeric()` is used to test whether an object is of numeric data type. It returns a logical value (either `TRUE` or `FALSE`) indicating whether the given object is numeric.
   - Example:

```R
x <- c(1, 2, 3, 4)
is_x_numeric <- is.numeric(x)
print(is_x_numeric)
```

Output:

```csharp
[1] TRUE
```

   - Note: In this example, `is.numeric(x)` returns `TRUE` because all elements in the vector `x` are numeric.

In summary, `as.numeric()` is used for type conversion, attempting to change an object into a numeric type, while `is.numeric()` is used for type checking, determining whether an object is already of numeric type or not.

Install and import Packages In R

Package

So, what is exactly a "package"? Package is just a collection of useful functions, we wrap all these functions into a bag, then the bag is called a "package"

Install and Import

We use `library()` to open (import) the package, and use `install.packages()` to install the package

Q: If we are not necessary to import all the package, we just interested on a few functions is some packages, how can we do?

A: Use `Package::Function`

Here, we provide 3 ways to install R package

1. Bioconductor
2. R Cran
3. Github

In [29]:

```
#install.packages("ggplot2")
library(ggplot2)
library("devtools")
install_github('https://github.com/DongDong-Zoez/HCV')
```

KS

How to install packages in R?

In R, you can install packages using the `install.packages()` function. Packages are collections of R functions, data sets, and documentation bundled together for specific purposes. Here's how to install packages in R:

1. **Using `install.packages()`:**
   To install a package from CRAN (Comprehensive R Archive Network), follow these steps:

   ```R
   # Install a package from CRAN
   install.packages("package_name")
   ```

   Replace `"package_name"` with the name of the package you want to install. For example, if you want to install the "ggplot2" package:

   ```R
   install.packages("ggplot2")
   ```

   This command will download and install the package and its dependencies.

2. **Using `install.packages()` with a Repository URL:**
   If you need to install a package from a specific repository or source other than CRAN, you can specify the repository using the `repos` argument:

   ```R
   # Install a package from a specific repository
   install.packages("package_name", repos = "repository_url")
   ```

   Replace `"package_name"` with the package name and `"repository_url"` with t
   of the repository you want to use.

3. **Install Multiple Packages at Once**:

   You can install multiple packages at once by providing a vector of package names:

   R Copy code

```r
# Install multiple packages
install.packages(c("package1", "package2", "package3"))
```

4. **Loading Installed Packages**:

   Once you've installed a package, you need to load it into your R session to use its functions and features. You can do this using the `library()` function:

   R Copy code

```r
# Load a package
library(package_name)
```

   Replace `"package_name"` with the name of the package you want to load. After loading a package, you can use its functions and data sets in your R scripts or sessions.

5. **Check Installed Packages**:

   To see which packages are currently installed in your R environment, you can use the `installed.packages()` function:

   R Copy code

```r
# List installed packages
installed_packages <- installed.packages()
print(installed_packages)
```

   This will display a list of installed packages along with their details.

It's worth noting that packages in R are typically hosted on CRAN, but there are other repositories and sources where you can find and install packages as well. Additionally, some packages may require specific dependencies or additional setup, so it's a good practice to refer to the package documentation for any specific installation instructions or requirements.

Rege te

R Help System

- help() or ?: we can look up the documentation.
- example(): we can also see the example

- Ex.

```
> example(cat)

cat> iter <- stats::rpois(1, lambda = 10)
cat> ## print an informative message
cat> cat("iteration = ", iter <- iter + 1, "\n")
iteration =  21
cat> ## 'fill' and label lines:
cat> cat(paste(letters, 100* 1:26), fill = TRUE, labels = paste0("{", 1:10, "}:"))
{1}: a 100 b 200 c 300 d 400 e 500 f 600 g 700 h 800 i 900 j 1000 k 1100
{2}: l 1200 m 1300 n 1400 o 1500 p 1600 q 1700 r 1800 s 1900 t 2000 u 2100
{3}: v 2200 w 2300 x 2400 y 2500 z 2600
```

Homework 2 (submitted to e3.nycu.edu.tw before Oct 4, 2023)

- Use R, Python, and suitable computer packages to analyze the data set that you select.
- Explain the results you obtain.
- Discuss possible problems you plan to investigate for future studies.

Possible sources of open datasets:

- UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php)
- Kaggle Datasets (https://www.kaggle.com/datasets)

Topic 2.0.1: Black and White Histogram with R

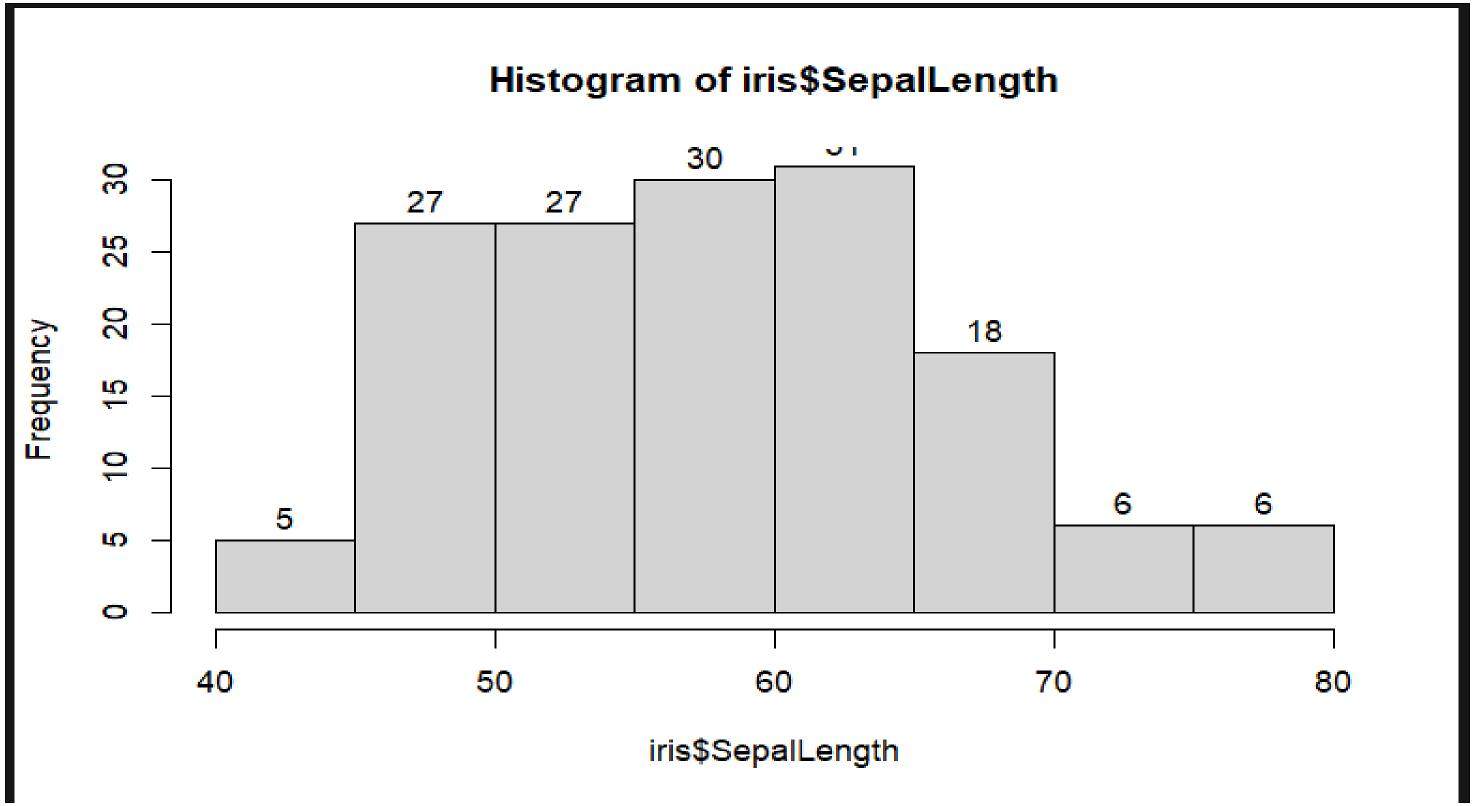
References:

http://stat.ethz.ch/R-manual/R-devel/library/graphics/html/hist.html

Black and White Histogram With R

Code:-

```
iris = read.table(file = 'iris.txt', sep = '', header = TRUE)
hist(iris$SepalLength, labels = TRUE)
```



KS

plot histogram in R

In R, you can create a histogram to visualize the distribution of a numeric variable using the `hist()` function. Here's a step-by-step guide on how to plot a histogram:

1. **Prepare Your Data**: Ensure you have a numeric vector or dataset that you want to create a histogram for. If you don't have data readily available, you can create a simple example like this:

R Copy code

```R
# Create a numeric vector (example data)
data <- c(23, 45, 56, 34, 67, 89, 45, 34, 78, 90, 56, 34, 67, 56)
```

1. **Create the Histogram**:
Use the `hist()` function to generate the histogram. You need to provide the data you want to plot as an argument to the function. Optionally, you can specify parameters like the number of bins, axis labels, and more.

R Copy code

```R
# Create a histogram
hist(data,
     main = "Histogram of Data",        # Title of the plot
     xlab = "Values",                   # Label for the x-axis
     ylab = "Frequency",                # Label for the y-axis
     col = "skyblue",                   # Color of the bars
     border = "black",                 # Border color of the bars
     breaks = 5)                        # Number of bins (intervals)
```

In this example, we create a histogram of the `data` vector with a title, axis labels, and some visual customization.

Rege[...]te

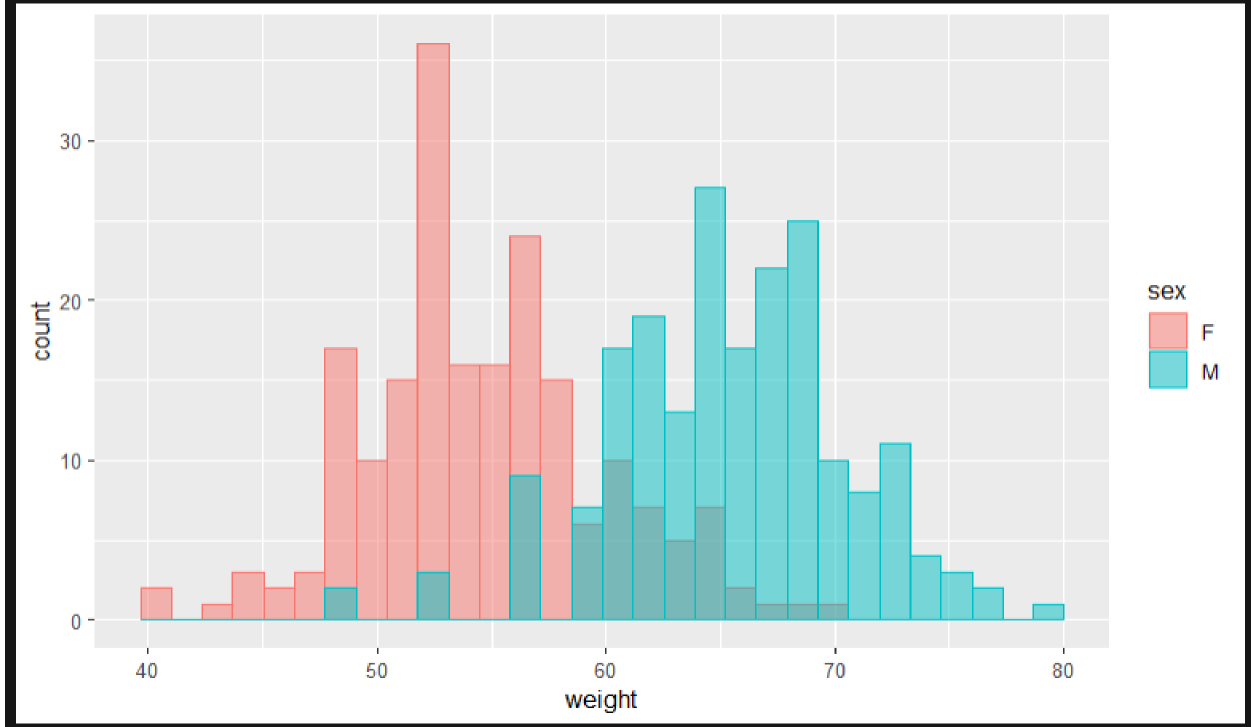
Topic 2.0.2: Color Histogram

References:-
http://www.r-tutor.com/elementary-statistics/quantitative-data/histogram
http://www.sthda.com/english/wiki/ggplot2-histogram-plot-quick-start-guide-r-software-and-data-visualization

Color Histogram With R

````
```{r}
set.seed(1234)
df <- data.frame(
 sex=factor(rep(c("F", "M"), each=200)),
 weight=round(c(rnorm(200, mean=55, sd=5), rnorm(200,
mean=65, sd=5)))
)
library(ggplot2)
# Use semi-transparent fill
p<-ggplot(df, aes(x=weight, fill=sex, color=sex)) +
 geom_histogram(position="identity", alpha=0.5)
p

```
````

```
[38;5;252m`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.[39m
```



KS

Color Histogram With R

To create a colored histogram in R, you can use the `hist()` function along with the `col` argument to specify the colors of the histogram bars. Additionally, you can add custom colors to the bars based on certain conditions or data values. Here's an example of how to create a colored histogram with R:

R — Copy code

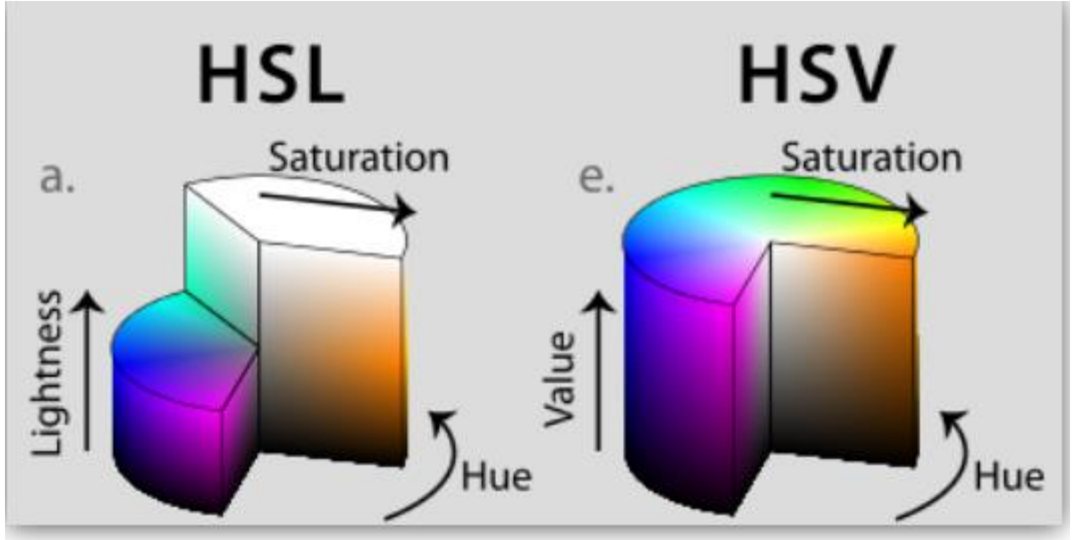
```r
# Create example data with three different colors
data <- data.frame(
  values = c(15, 25, 35, 45, 55, 65, 75, 85, 95),
  colors = c("red", "green", "blue", "red", "green", "blue", "red", "green",
)

# Create a colored histogram
hist(data$values,
     col = data$colors,           # Use the 'colors' column for bar colors
     main = "Colored Histogram",  # Title of the plot
     xlab = "Values",            # Label for the x-axis
     ylab = "Frequency",         # Label for the y-axis
     border = "black",           # Border color of the bars
     breaks = 5)                 # Number of bins (intervals)
```

Topic 2.0.2: Different Color Models

References:-
https://en.wikipedia.org/wiki/Color_model
https://en.wikipedia.org/wiki/HSL_and_HSV
https://cran.r-project.org/src/contrib/Archive/ColorPalette/

Color model

- HSL (Hue, Saturation, Lightness)
- HSV (Hue, Saturation, Value)



KS Different Color Models

There are several color models used in computer graphics and image processing to represent and manipulate colors. These color models provide different ways to express and work with colors. Here are some of the most common color models:

1. **RGB (Red, Green, Blue)**:
   - RGB is an additive color model used for representing colors on electronic displays, such as computer monitors and TVs.
   - It defines colors by specifying the intensity of red, green, and blue components. Each component can range from 0 to 255 or 0 to 1, depending on the scale used.
   - Different combinations of these three primary colors create a wide range of colors.
2. **CMY (Cyan, Magenta, Yellow)**:
   - CMY is a subtractive color model used in color printing and mixing pigments.
   - It represents colors by subtracting the intensity of cyan, magenta, and yellow from white. When all three colors are combined at full intensity, they produce black.
   - CMY is often extended to CMYK (with Key or Black) for color printing, where black is added to enhance color depth.
3. **HSV (Hue, Saturation, Value)**:
   - HSV is a color model that represents colors in terms of their perceived attributes: hue, saturation, and value (brightness).
   - Hue corresponds to the type of color (e.g., red, blue, green).
   - Saturation measures the vividness or purity of a color (0% is grayscale, 100% is fully saturated).
   - Value represents the brightness of the color (0 is black, 100 is fully bright).
4. **HSL (Hue, Saturation, Lightness)**:
   - HSL is similar to HSV but uses lightness instead of value to represent brightness.
   - Lightness represents the perceived brightness of the color and ranges from 0 (black) to 100 (white).
   - HSL is often used in computer graphics and design applications.

Regenerate

Different Color Models

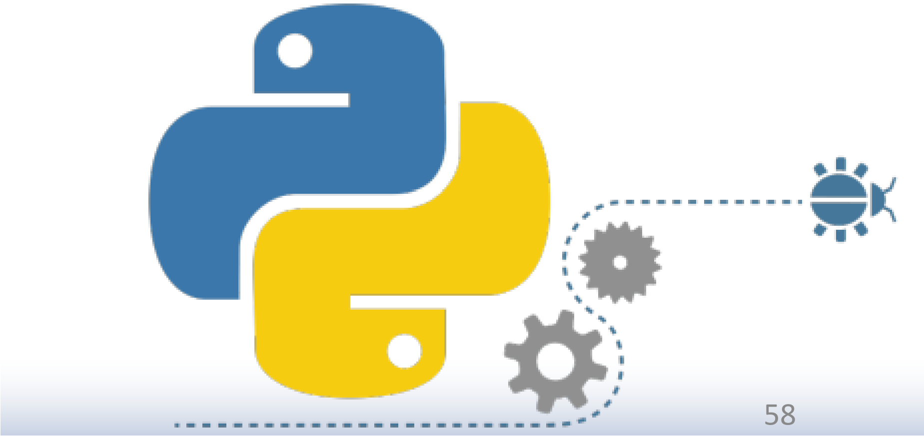
```
> library(ColorPalette)
> library(grDevices)
> library(stringi)
> library(imager)
> library(ImaginR)
```

```
> (rc <- col2rgb("red"))
      [,1]
red    255
green    0
blue     0

> (hc <- rgb2hsv(rc))
  [,1]
h    0
s    1
v    1

>  hex2rgb(hsv2rgb(hc[1], hc[2], hc[3]))
       R G B
[1,] 255 0 0
```

Topic 2.2: Introduction of Python

References:-

https://www.kaggle.com/code/dgawlik/house-prices-eda
https://phonchi.github.io/nsysu-math524-2021/



Topic 2.2: Introduction of Python

- Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, and a syntax that allows programmers to express concepts in fewer lines of code, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.
- Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

KS

Introduction of Python

Python is a high-level, versatile, and interpreted programming language known for its simplicity, readability, and wide range of applications. Developed by Guido van Rossum and first released in 1991, Python has gained immense popularity in both the programming community and industry. Here's an introduction to Python:

1. General-Purpose Language:

- Python is a general-purpose programming language, meaning it can be used for a wide variety of tasks, including web development, data analysis, machine learning, scientific computing, automation, and more.

2. Readability and Syntax:

- Python is famous for its clean and readable syntax. It uses indentation (whitespace) to define code blocks instead of braces or keywords, making the code more human-friendly and reducing syntax errors.

3. High-Level and Interpreted:

- Python is a high-level language, which means it provides abstraction from low-level details, making it easier to write and understand code.
- It is an interpreted language, which means you can run Python code directly without the need for a separate compilation step. Python's interpreter executes code line by line.

4. Dynamically Typed:

- Python is dynamically typed, meaning you don't need to declare the data type of a variable explicitly. The interpreter determines the data type at runtime.

5. Strong Community and Libraries:

5. Strong Community and Libraries:

- Python has a large and active community of developers, making it easy to find solutions to problems, get help, and share code.
- It boasts a rich ecosystem of libraries and frameworks, such as NumPy, pandas, TensorFlow, Django, and Flask, which facilitate various tasks and domains.

6. Cross-Platform Compatibility:

- Python is available on multiple platforms, including Windows, macOS, and various Unix-like systems. This cross-platform compatibility allows you to write code that works on different operating systems without modification.

7. Open Source:

- Python is open-source software, meaning it is freely available, and you can modify and distribute it as per the terms of the Python Software Foundation License.

8. Versatile Applications:

- Python is widely used in various fields and industries, such as web development (with frameworks like Django and Flask), data analysis (with libraries like pandas and matplotlib), machine learning and AI (with libraries like TensorFlow and scikit-learn), scientific computing, and more.

9. Interactive Shell:

- Python provides an interactive shell (REPL - Read-Eval-Print Loop), which allows you to experiment with code, test small snippets, and get immediate feedback.

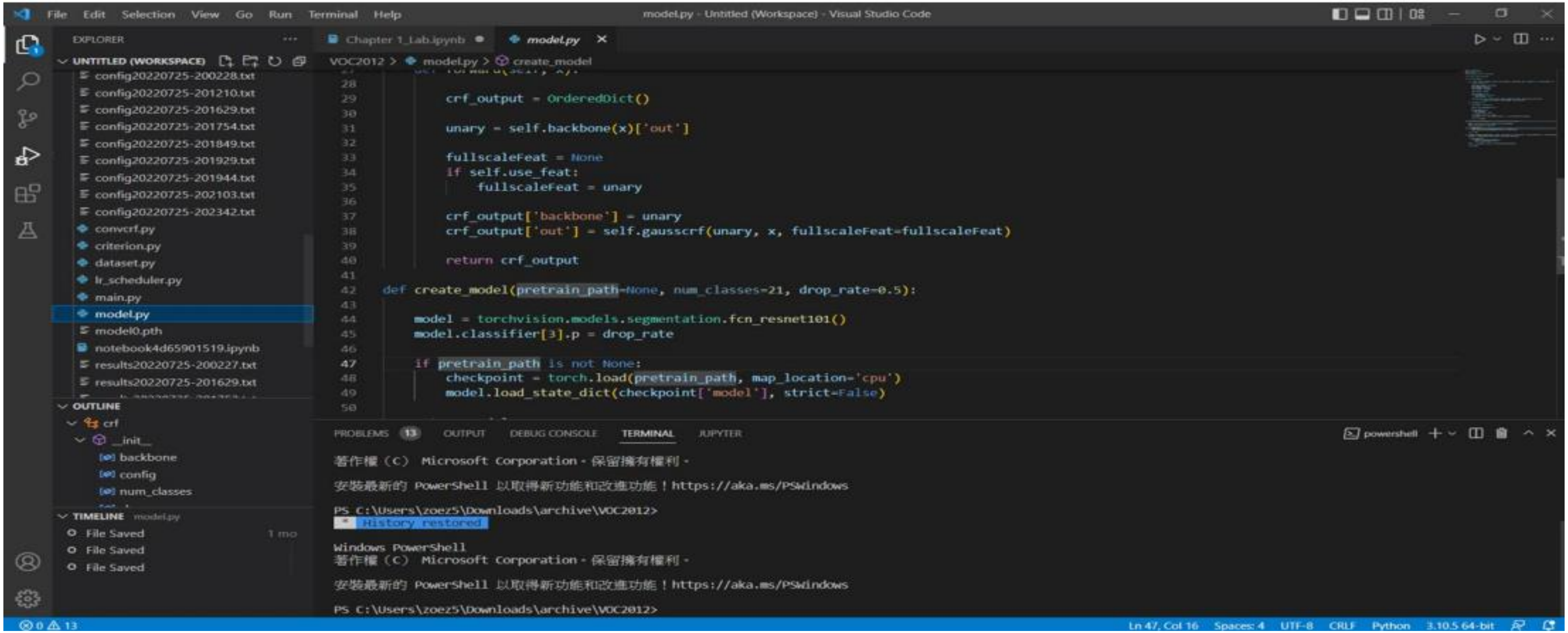
10. Learning and Teaching:
- Python is often recommended as a first programming language for beginners due to its simple and clear syntax. It is also a popular choice for educational purposes.

Regenerate

Which Integrated Development Environment (IDE) to Use?

1. Jupyter (local recommend)
2. Spyder
3. Pycharm
4. VS code (local recommend)
5. Vim
6. Google Colab (online recommend)
7. Atom
8. Kaggle (online recommend)

VS code Interface



Quick Start to Python

Note:- If you are not familiar with Python, check this tutorial

- In this lecture, we are going to use pandas and seaborns to analyze data.
- First, Let's import some packages and some dependencies

Read CSV to Data Frame:

D: > TA Introducing the data Science > 2023 notes > Topic-2 > Topic_2 Introduction to Data Science.py > ...

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_profiling as pp
import seaborn as sns
import warnings
import os

iris = pd.read_csv("D:/TA Introducing the data Science/2023 notes/Topic-2/iris.csv")
print(iris.head(10))
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#csv-text-files

KS

read and write csv file in python using the pandas library

You can read and write CSV files in Python using the pandas library, which provides convenient functions for data manipulation and analysis. Here's how to do it:

Reading a CSV File with Pandas:

To read data from a CSV file using pandas, you can use the `pd.read_csv()` function. Here's an example:

python Copy code

```python
import pandas as pd

# Read data from a CSV file
df = pd.read_csv('data.csv')

# Display the DataFrame
print(df)
```

In this code:

- We import the pandas library as `pd`.
- We use the `pd.read_csv()` function to read the data from the 'data.csv' file and store it in a pandas DataFrame `df`.
- Finally, we print the contents of the DataFrame.

Writing to a CSV File with Pandas:

To write data to a CSV file using pandas, you can use the `to_csv()` method of a pandas DataFrame. Here's an example:

python Copy code

```python
import pandas as pd

# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 22]}
df = pd.DataFrame(data)

# Write the DataFrame to a CSV file
df.to_csv('output.csv', index=False)
```

In this code:

- We create a pandas DataFrame `df` using a dictionary of data.
- We use the `to_csv()` method to write the contents of the DataFrame to an 'output.csv' file. The `index=False` argument ensures that the DataFrame index is not written to the file.

After running this code, a CSV file named 'output.csv' will be created with the data from the DataFrame.

Pandas offers a wide range of options and parameters for customizing the reading and writing of CSV files, such as specifying delimiters, handling missing values, and more. You can refer to the pandas documentation for more details: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#csv-text-files

Pandas (Data Frame from matrix)

In [6]:

```
# from matrix
nrow = 5
n_feateures = 4
dates = pd.date_range("20220101", periods=nrow, freq='s')
df = pd.DataFrame(np.random.randn(nrow, n_feateures), index=dates, columns=list("ABCD"))
df
```

Out[6]:

	A	B	C	D
2022-01-01 00:00:00	0.705401	-0.152891	0.912559	0.619826
2022-01-01 00:00:01	0.355068	-0.292160	-0.364981	-0.406755
2022-01-01 00:00:02	0.719457	0.131548	-0.277148	0.555377
2022-01-01 00:00:03	-1.890779	1.164003	-0.609252	-0.617723
2022-01-01 00:00:04	-0.635492	-0.827052	2.193252	0.483568

In pandas, you can create a DataFrame from a matrix (2D array) using the `pd.DataFrame()` constructor. Here's how you can do it:

python Copy code

```python
import pandas as pd

# Create a 2D matrix (list of lists)
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Create a DataFrame from the matrix
df = pd.DataFrame(matrix)

# Display the DataFrame
print(df)
```

In this example:

1. We import the pandas library as `pd`.
2. We create a 2D matrix `matrix` as a list of lists. Each inner list represents a row of the matrix.
3. We use the `pd.DataFrame()` constructor to create a DataFrame `df` from the `matrix`.
4. Finally, we print the DataFrame, which will display the matrix in tabular form:

Copy code

```
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
```

The DataFrame is created with default column names (0, 1, 2) and an automatically generated index (0, 1, 2). You can customize the column names and index labels if needed by providing additional arguments to the `pd.DataFrame()` constructor. For example:

Pandas (Data Frame from series)

In [7]:

```
# from series
t = np.arange(5) #start from 0
sin_t = np.sin(t)
cos_t = np.cos(t)
exp_t = np.exp(t)
df2 = pd.DataFrame({'t': t, 'sin': sin_t, 'cos': cos_t, 'exp': exp_t})
df2
```

Out[7]:

	t	sin	cos	exp
0	0	0.000000	1.000000	1.000000
1	1	0.841471	0.540302	2.718282
2	2	0.909297	-0.416147	7.389056
3	3	0.141120	-0.989992	20.085537
4	4	-0.756802	-0.653644	54.598150

Creating a DataFrame from Multiple Series:

You can also create a DataFrame from multiple Series by passing them as a dictionary to the `pd.DataFrame()` constructor. Each Series becomes a column in the resulting DataFrame. Here's an example:

python Copy code

```python
import pandas as pd

# Create multiple Series
series1 = pd.Series([1, 2, 3], name='A')
series2 = pd.Series([4, 5, 6], name='B')

# Create a DataFrame from the Series
df = pd.DataFrame({'Column1': series1, 'Column2': series2})

# Display the DataFrame
print(df)
```

In this example, we create two Series ('A' and 'B') and use them to create a DataFrame with two columns ('Column1' and 'Column2'):

Copy code

```
   Column1  Column2
0        1        4
1        2        5
2        3        6
```

You can create more complex DataFrames by combining multiple Series with different names and lengths.

The ability to create DataFrames from Series is useful when you have data organized as individual Series, and you want to combine them into a structured tabular format for analysis and manipulation.

Regenerate

Basis Application Programming Interface (API)

We now can check the contains by following API:

1. head: view the top of the df.
2. tail: view bot of the df.
3. columns: view df column names.
4. index: view df row names.
5. shape: df shape.

In [8]:

```
df.head(3), df.tail(2), df.columns, df.index, df.shape
```

For the overall information, we can use following API:

1. `info:` view df information
2. `describe:` view df statistics

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5 entries, 2022-01-01 00:00:00 to 2022-01-01 00:00:04
Freq: S
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       5 non-null      float64
 1   B       5 non-null      float64
 2   C       5 non-null      float64
 3   D       5 non-null      float64
dtypes: float64(4)
memory usage: 200.0 bytes
```

Basis API

In [10]:

```
df.describe()
```

Out[10]:

	A	B	C	D
count	5.000000	5.000000	5.000000	5.000000
mean	-0.149269	0.004690	0.370886	0.126859
std	1.119014	0.735671	1.176494	0.590134
min	-1.890779	-0.827052	-0.609252	-0.617723
25%	-0.635492	-0.292160	-0.364981	-0.406755
50%	0.355068	-0.152891	-0.277148	0.483568
75%	0.705401	0.131548	0.912559	0.555377
max	0.719457	1.164003	2.193252	0.619826

Data Selection

There are four ways to select the sub data

1. `[x:y]:` slice df
2. `loc:` select by row or column
3. `iloc:` select by position
4. `[boolean mask]:` select by mask

```
df.loc[:,'A']
```

Out[12]:

```
2022-01-01 00:00:00    0.705401
2022-01-01 00:00:01    0.355068
2022-01-01 00:00:02    0.719457
2022-01-01 00:00:03   -1.890779
2022-01-01 00:00:04   -0.635492
Freq: S, Name: A, dtype: float64
```

In [11]:

```
df[0:3]
```

Out[11]:

	A	B	C	D
2022-01-01 00:00:00	0.705401	-0.152891	0.912559	0.619826
2022-01-01 00:00:01	0.355068	-0.292160	-0.364981	-0.406755
2022-01-01 00:00:02	0.719457	0.131548	-0.277148	0.555377

Data Selection

In [13]:

```
df.iloc[0:2,3:]
```

Out[13]:

	D
2022-01-01 00:00:00	0.619826
2022-01-01 00:00:01	-0.406755

In [14]:

```
df[df.A < 0]
```

Out[14]:

	A	B	C	D
2022-01-01 00:00:03	-1.890779	1.164003	-0.609252	-0.617723
2022-01-01 00:00:04	-0.635492	-0.827052	2.193252	0.483568

KS how to select sub data in pandas python

In pandas, you can select sub-data or subsets of a DataFrame using various techniques and methods. Here are some common methods for selecting data in pandas:

1. **Selection by Column Name**:
   - To select one or more columns by name, you can use square brackets and specify the column names as a list.

```python
# Select a single column by name
single_column = df['ColumnName']

# Select multiple columns by name
multiple_columns = df[['Column1', 'Column2']]
```

2. **Selection by Row Index**:
   - You can select rows by their index using the `.loc[]` indexer. Pass the row labels as arguments.

```python
# Select a single row by label
single_row = df.loc['Label']

# Select multiple rows by labels
multiple_rows = df.loc[['Label1', 'Label2']]
```

3. **Selection by Row and Column Index**:
   - You can select specific rows and columns by index using the `.loc[]` indexer with both row and column labels.

```python
# Select a specific cell by label
cell_value = df.loc['RowLabel', 'ColumnLabel']

# Select a subset of rows and columns by labels
subset = df.loc[['Row1', 'Row2'], ['Column1', 'Column2']]
```

4. **Selection by Numeric Position**:
   - You can select rows and columns by their numeric position using the `.iloc[]` indexer.

```python
# Select a single row by position
single_row = df.iloc[0]

# Select a single cell by position
cell_value = df.iloc[0, 1]

# Select a subset of rows and columns by positions
subset = df.iloc[1:4, 2:5]
```

5. **Conditional Selection**:
   - You can use boolean indexing to select rows that meet certain conditions.

```python
# Select rows where a condition is met
selected_rows = df[df['Column'] > 10]
```

6. **Selection by `.query()` Method**:
   - The `.query()` method allows you to select rows using a query expression.

```python
# Select rows using a query expression
selected_rows = df.query('Column > 10')
```

Merge and Group Operation

- Like R, we can merge two *df* with same shape at some axis, also, we are interesting in grouping data to evaluate the statistics of each group, this can be done via following API:

In [15]:

```
df1 = pd.DataFrame(np.random.rand(2,4))
df2 = pd.DataFrame(np.random.rand(1,4))
df3 = pd.DataFrame(np.random.rand(3,5))
df_list = [df1, df2, df3]
```

In [16]:

```
df4 = pd.concat(df_list, axis=0)
df4
```

	0	1	2	3	4
0	0.290076	0.377931	0.588759	0.591100	NaN
1	0.064946	0.292407	0.426675	0.452083	NaN
0	0.944321	0.380966	0.293576	0.728972	NaN
0	0.276507	0.985832	0.952853	0.807235	0.200123
1	0.462103	0.714112	0.720310	0.658994	0.575129
2	0.314486	0.069164	0.324812	0.436557	0.089504

In [18]:

```
df5.fillna(0)
```

Out[18]:

	0	1	2	3	0	1	2	3	0	1
0	0.290076	0.377931	0.588759	0.591100	0.944321	0.380966	0.293576	0.728972	0.276507	0.985832
1	0.064946	0.292407	0.426675	0.452083	0.000000	0.000000	0.000000	0.000000	0.462103	0.714112
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.314486	0.069164

In [19]:

```
iris.groupby('Species').apply(np.mean)
```

Out[19]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
Species				
Setosa	50.06	34.28	14.62	2.46
Versicolor	59.36	27.70	42.60	13.26
Virginica	65.88	29.74	55.52	20.26

Merge and Group Operation in pandas python

In pandas, you can perform merging and grouping operations to manipulate and analyze data in DataFrames. Here's an overview of these operations:

Merging Operations:

Merging operations are used to combine two or more DataFrames into a single DataFrame. You can perform different types of merges, such as inner join, left join, right join, and outer join, to control how data is combined.

1. `pd.concat()`:
   - Concatenation is used to combine DataFrames vertically (along rows) or horizontally (along columns).
   - `pd.concat([df1, df2])` combines DataFrames `df1` and `df2` vertically.
   - `pd.concat([df1, df2], axis=1)` combines DataFrames `df1` and `df2` horizontally.
2. `pd.merge()`:
   - Merging is used to combine DataFrames based on common columns or keys.
   - `pd.merge(df1, df2, on='key')` performs an inner join on the 'key' column.
   - You can also specify the type of join using the `how` parameter (`'inner'`, `'left'`, `'right'`, `'outer'`).

Grouping Operations:

Grouping operations involve splitting data into groups based on a criterion and then applying an operation (e.g., aggregation) to each group.

1. `groupby()`:
   - The `groupby()` method is used to group data based on one or more columns.
   - After grouping, you can apply aggregation functions like `sum()`, `mean()`, `count()`, etc., to calculate summary statistics for each group.

python   Copy code

```python
grouped = df.groupby('column_name')
result = grouped['value_column'].sum()
```

2. **Aggregation Functions:**
   - You can apply various aggregation functions to grouped data, such as `sum()`, `mean()`, `count()`, `min()`, `max()`, and custom functions using `agg()`.

python   Copy code

```python
result = grouped['value_column'].agg(['sum', 'mean', 'count'])
```

3. **Filtering and Transformation:**
   - You can use the `filter()` and `transform()` methods to filter groups based on conditions or apply transformations to each group, respectively.

python   Copy code

```python
filtered_groups = grouped.filter(lambda x: x['value_column'].sum() > 100)
transformed_values = grouped['value_column'].transform(lambda x: x - x.mean
```

Seaborn

- In R, we have ggplot2 to visualize our data, the alternative choice in Python is seaborn.

- Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- It is a powerful tool to visualize our data and perform EDA, Let's have a quick start on Seaborn.

API

You can find all the API from here, Let's take house price dataset to demonstrate.

KS

Seaborn in python

Seaborn is a popular Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn simplifies the process of creating complex visualizations by providing easy-to-use functions and themes. Here are some key features and common tasks you can perform with Seaborn:

Features and Capabilities:

1. **High-Level Interface:** Seaborn provides a high-level interface for creating various statistical plots with concise and expressive syntax.
2. **Beautiful Aesthetics:** It comes with attractive themes and color palettes, making it easy to create visually appealing plots.
3. **Statistical Estimation:** Seaborn includes functions for statistical estimation and data aggregation within plots, such as kernel density estimation (KDE) and confidence intervals.
4. **Categorical Plots:** Seaborn excels at creating categorical plots like bar plots, count plots, and box plots, making it suitable for visualizing data with categorical variables.
5. **Regression Plots:** It offers functions to create regression plots with automatic linear model fits and uncertainty visualization.
6. **Matrix Plots:** Seaborn provides heatmap and cluster map functions for visualizing matrices of data.
7. **Distribution Plots:** You can create distribution plots like histograms, KDE plots, and violin plots to explore data distributions.
8. **Time Series Visualization:** Seaborn supports time series data visualization, including line plots and time series heatmaps.
9. **FacetGrids:** FacetGrids allow you to create multiple plots for subsets of your data, which is useful for exploring relationships between variables.

Input

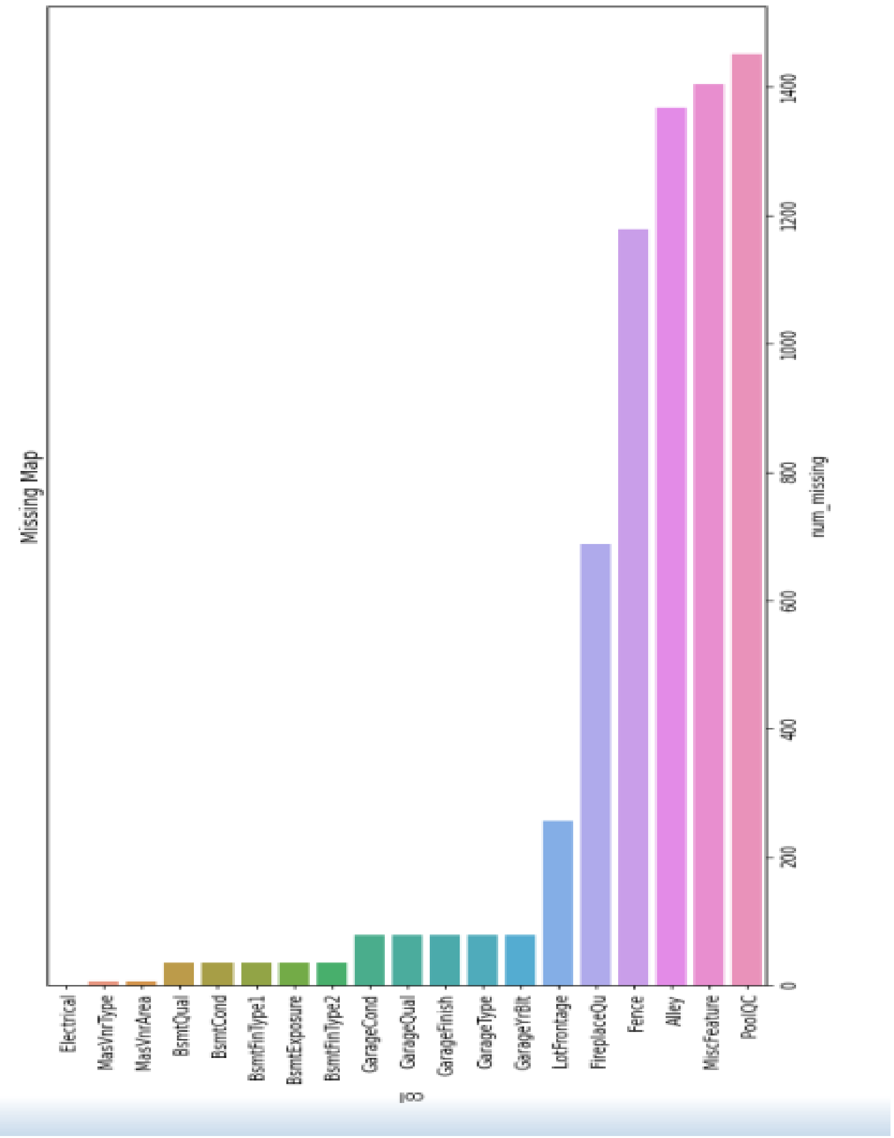
```python
##################   Seaborn      ##########################
import seaborn as sns
df = pd.read_csv('D:/TA Introducing the data Science/2023 notes/Topic-2/train.csv')
print(df.head(10))
```

Output

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape LandContour Utilities LotConfig LandSlope  ... 3SsnPorch ScreenPorch PoolArea PoolQC Fence MiscFeature MiscVal  MoSold  YrSold  SaleType  SaleCondition  SalePrice
0   1          60       RL         65.0     8450   Pave   NaN      Reg         Lvl    AllPub    Inside       Gtl  ...         0           0        0    NaN   NaN         NaN       0       2    2008        WD         Normal     208500
1   2          20       RL         80.0     9600   Pave   NaN      Reg         Lvl    AllPub       FR2       Gtl  ...         0           0        0    NaN   NaN         NaN       0       5    2007        WD         Normal     181500
2   3          60       RL         68.0    11250   Pave   NaN      IR1         Lvl    AllPub    Inside       Gtl  ...         0           0        0    NaN   NaN         NaN       0       9    2008        WD         Normal     223500
3   4          70       RL         60.0     9550   Pave   NaN      IR1         Lvl    AllPub    Corner       Gtl  ...         0           0        0    NaN   NaN         NaN       0       2    2006        WD        Abnorml     140000
4   5          60       RL         84.0    14260   Pave   NaN      IR1         Lvl    AllPub       FR2       Gtl  ...         0           0        0    NaN   NaN         NaN       0      12    2008        WD         Normal     250000
5   6          50       RL         85.0    14115   Pave   NaN      IR1         Lvl    AllPub    Inside       Gtl  ...       320           0        0    NaN MnPrv        Shed     700      10    2009        WD         Normal     143000
6   7          20       RL         75.0    10084   Pave   NaN      Reg         Lvl    AllPub    Inside       Gtl  ...         0           0        0    NaN   NaN         NaN       0       8    2007        WD         Normal     307000
7   8          60       RL          NaN    10382   Pave   NaN      IR1         Lvl    AllPub    Corner       Gtl  ...         0           0        0    NaN   NaN        Shed     350      11    2009        WD         Normal     200000
8   9          50       RM         51.0     6120   Pave   NaN      Reg         Lvl    AllPub    Inside       Gtl  ...         0           0        0    NaN   NaN         NaN       0       4    2008        WD        Abnorml     129900
9  10         190       RL         50.0     7420   Pave   NaN      Reg         Lvl    AllPub    Corner       Gtl  ...         0           0        0    NaN   NaN         NaN       0       1    2008        WD         Normal     118000

[10 rows x 81 columns]
```

Missing Value Map

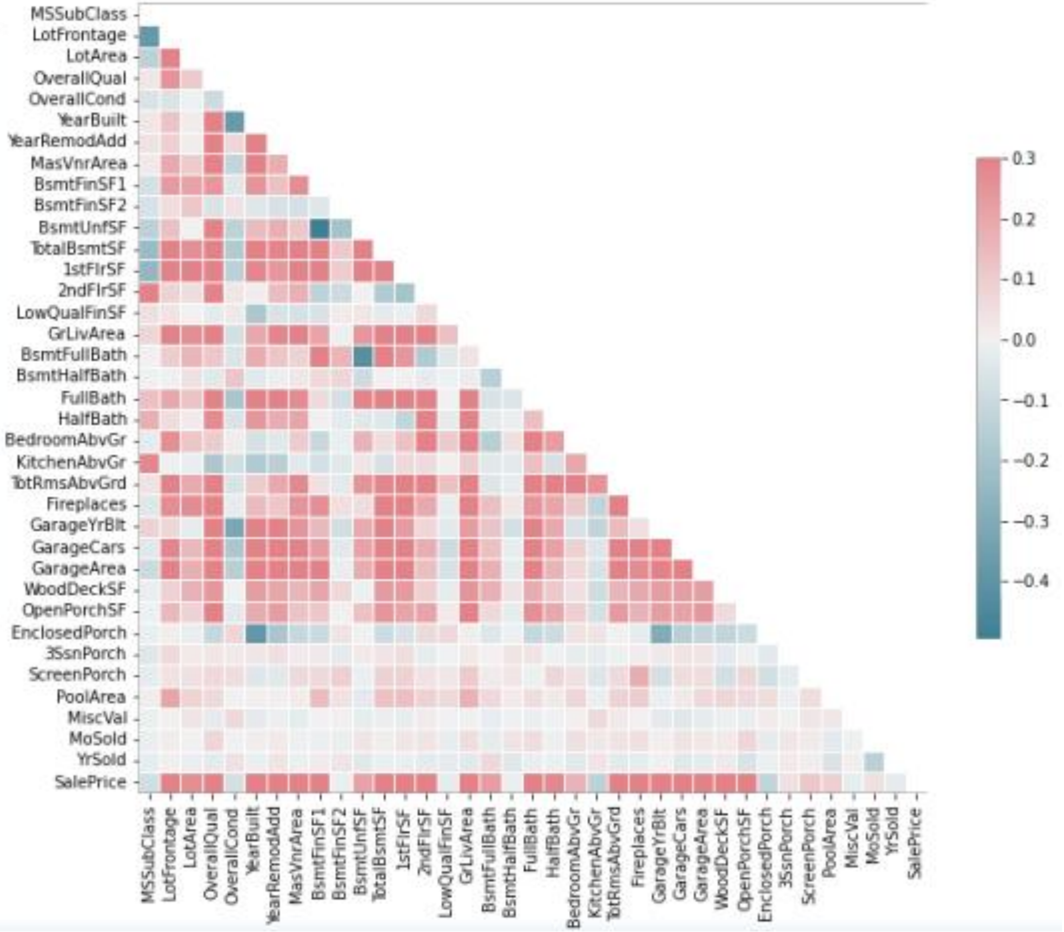
```
#################  Seaborn      ########################
import seaborn as sns
df = pd.read_csv('D:/TA Introducing the data Science/2023 notes/Topic-2/train.csv')
print(df.head(10))

#Missing Map
missing = df.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing_df = pd.DataFrame({'col': missing.index, 'num_missing': missing.values})

plt.figure(figsize=(14,7))
plt.title('Missing Map')
f = sns.barplot(y='col', x='num_missing', data=missing_df)
f.figure
plt.show()
```



Heat Map

[ ]:

```
corr = df.drop(['Id'], axis=1).corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11,11))
cmap = sns.diverging_palette(220, 10, as_cmap=True)

f = sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
                center=0, square=True, linewidths=.5,
                cbar_kws={"shrink": .5})
f.figure
```



Common Tasks with Seaborn:

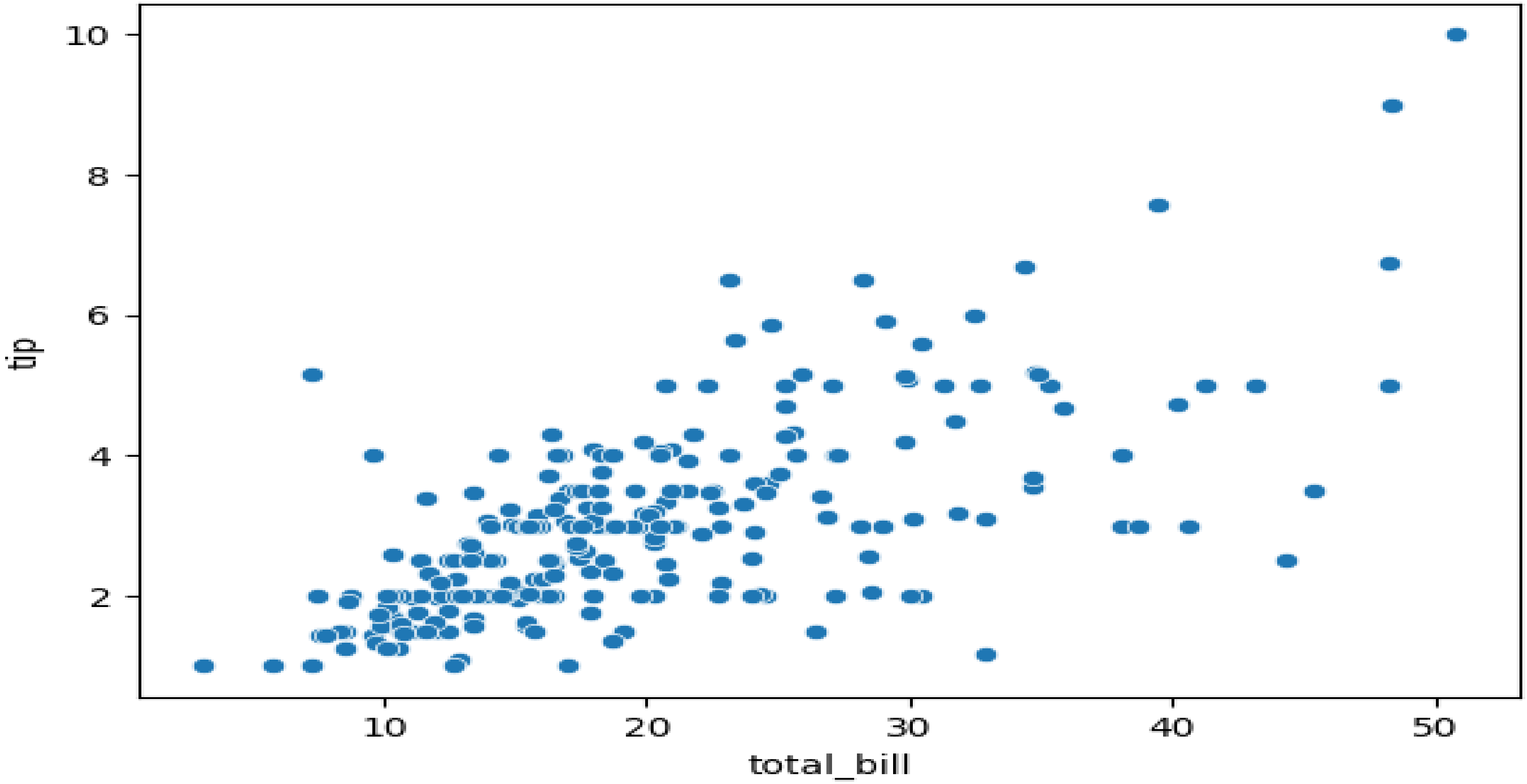
Here are some common tasks and examples of using Seaborn:

python Copy code

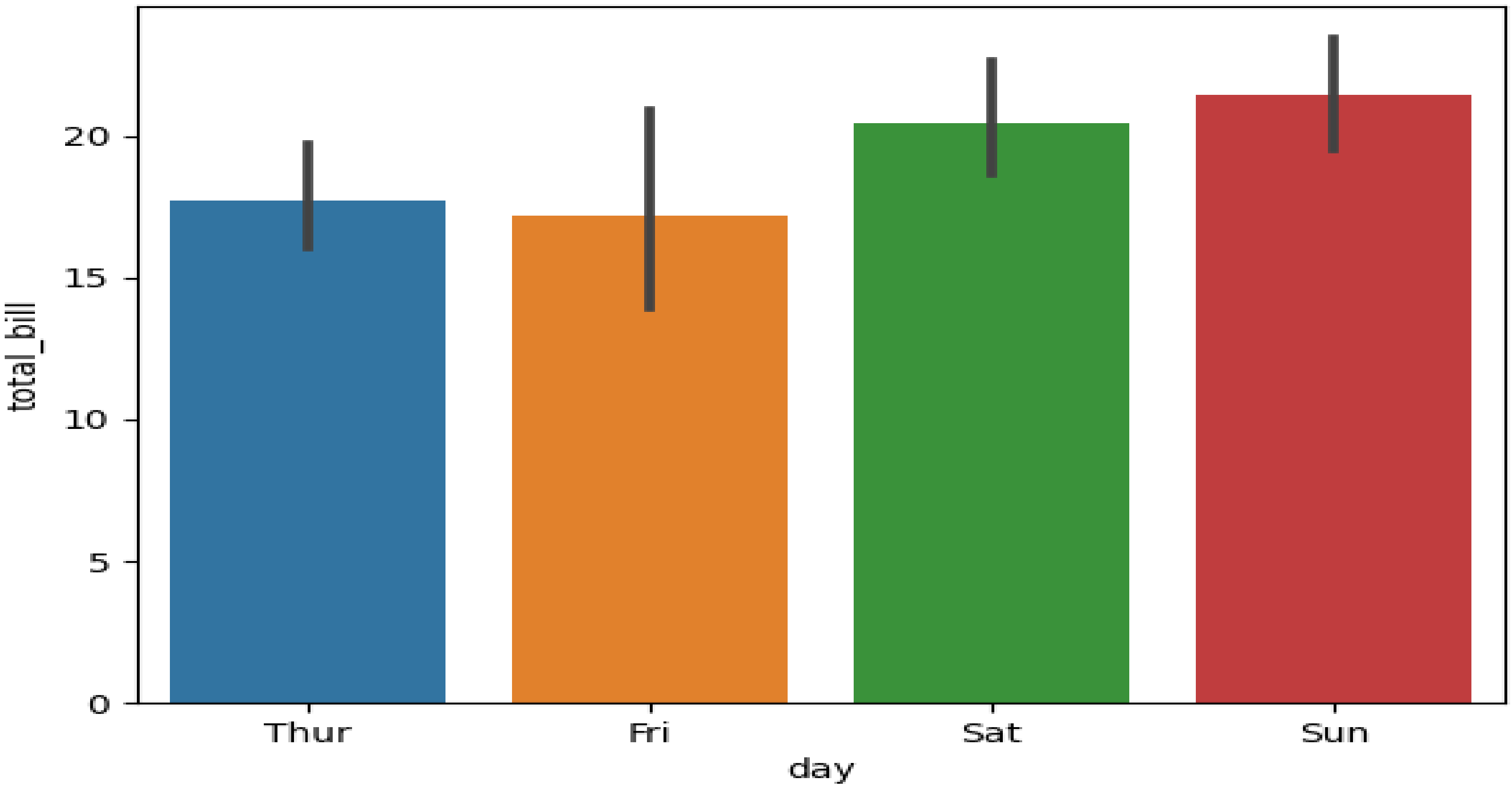
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Load a sample dataset (e.g., tips dataset)
tips = sns.load_dataset("tips")

# Create a scatter plot with regression line
sns.scatterplot(x="total_bill", y="tip", data=tips)

# Create a bar plot of categorical data
sns.barplot(x="day", y="total_bill", data=tips)

# Create a box plot to show data distribution
sns.boxplot(x="day", y="total_bill", data=tips)

# Create a pair plot to visualize relationships among multiple variables
sns.pairplot(tips, hue="sex")

# Create a heatmap to visualize a correlation matrix
correlation_matrix = tips.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")

# Customize plot aesthetics and themes
sns.set_style("whitegrid")
plt.title("Customized Plot")
```

Regen

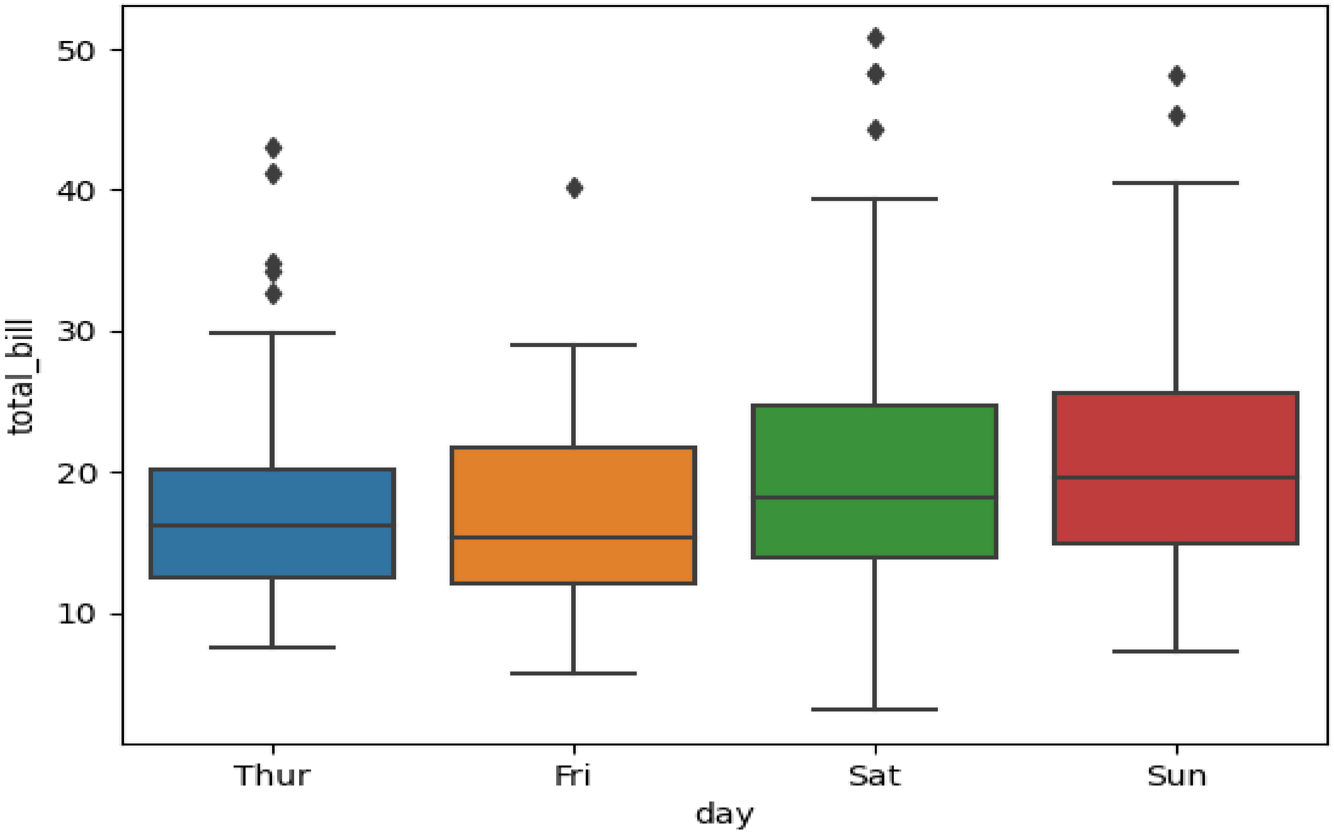
scatter plot



Bar Plot of Categorical Data



Box Plot



pair plot to visualize relationships among multiple variables