MASTER OF STATISTICS

STATISTICS IN BRIEF

A short handout for the MaSTAT students

GSEM, University of Geneva

CONTENTS

Acknowledgements

This guide is designed to help new graduate students in statistics to quickly review and familiarize themselves with the basic content of statistics.

The first version was jointly completed by Anna Venancio (anna.venancio@etu.unige.ch), Alice Scattolin (alice.scattolin@unige.ch), Chen Jiahao (Jiahao.Chen@etu.unige.ch), Eloi Favre (eloi.favre@etu.unige.ch), Laura Winkler (laura.winkler@etu.unige.ch), and Francisco Acosta (francisco.acosta@etu.unige.ch) in February 2023, under the supervision of Prof. Davide La Vecchia (davide.lavecchia@unige.ch). As time goes by, some of these items may no longer valid, please feel free to update and correct, and add useful topics.

Introduction

This book provides a short and to the point summary of the needed background knowledge and topics. If you feel like you are lacking some backgrorund knowledge, feel free to use Moodle, the online student platform used at the University of Geneva. There, you can enroll into courses, even if you're not officially enrolled. You can download lecture materials and work on the practicals, to make sure you are up to speed and filled the blanks you've experienced.

Navigation

The exact locations of the topics that are being referenced will be inside these boxes.

Enjoy your statistics learning journey!

Part I.
Python guidance

1 BASIC PART

Python is a high-level scripting language that combines interpretability, compilation, interactivity, and object-oriented.

- **Python Download and Run**
  Python3 can be downloaded from the official website of Python https://www.python.org/

  You also can download Python documentation at https://www.python.org/doc/link

  If you can successfully download and install python, then we have three methods to run python.

  1.REPL (Read-Eval-Print Loop)
  You can enter Python through the command line window and start writing code in the interactive interpreter. Generally, you can do this on Unix, DOS, or any other system that provides a command line or shell.

  2.Command-line scripts
  Python scripts can be executed on the command line by including the interpreter in your application, as follows:

```
$ python  script.py         # Unix/Linux

#or

C:>python script.py         # Windows/DOS
```

Listing 1: Python example

3.IDE: Integrated Development Environment: PyCharm or Anaconda, Jupyter Notebook
Integrated Development Environment (IDE) is an application program used to provide a program development environment, generally including tools such as code editors, compilers, debuggers, and graphical user interfaces. An integrated development software service suite that integrates code writing functions, analysis functions, compilation functions, and debugging functions. All software or software suites (groups) with this feature can be called integrated development environments. Such as Microsoft's Visual Studio series, Borland's C++ Builder, Delphi series, etc. The program can run independently,

or it can be used together with other programs. The integrated environment of python mainly includes PyCharm or Anaconda, Jupyter Notebook.

- **Identifier**
  With Python, you name your own variables. Just follow these rules:
  1. Characters must be letters of the alphabet or underscore
  2. The rest of the identifier consists of letters, numbers and underscores
  3. Identifiers are case sensitive
  4. Words such as "False" and "None" have been used by python(Called "Reserved word"), so they cannot be used as identifiers. Python's standard library provides a keyword module that outputs all Reserved words of the current version:

```
import keyword

print(keyword.kwlist)

>>> ['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', '
    del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in'
    , 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', '
    with', 'yield']
```

Listing 2: Reserved words

- **Note**
  Single-line comments in Python start with #, and multi-line comments are implemented using '''.

```
# first comment

# Second comment

'''
Third comment 1

Third comment 2
'''

print ("Hello, Python!")
>>> Hello, Python!
```

Listing 3: Notes and Comments

- **Operator**
  Python Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations.

```
# Examples of Arithmetic Operator
a = 9
```

```
b = 4

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b

# Multiplication of number
mul = a * b

# Division(float) of number
div1 = a / b

# Division(floor) of number
div2 = a // b

# Modulo of both number
mod = a % b

# Power
p = a ** b

# print results
print(add)
>>>13
print(sub)
>>>5
print(mul)
>>>36
print(div1)
>>>2.25
print(div2)
>>>2
print(mod)
>>>1
print(p)
>>>656
```

Listing 4: Example of Arithmetic Operator

- **Data type**
  Variables in Python do not need to be declared. Each variable must be assigned a value before use, and the variable will not be created until the variable is assigned. In Python, a variable is a variable, it doesn't have a type, what we mean by "type" is the type of object in memory that the variable refers to. The equal sign (=) is used to assign values to variables.

  The left side of the equal sign (=) operator is a variable name, and the right side of the equal sign (=) operator is the value stored in the variable. E.g:

```
counter = 100          # integer
miles   = 1000.0       # floate
name    = "unige"       # string

print (counter)
>>>100
print (miles)
>>>1000.0
print (name)
>>>unige
```

Listing 5: Examples of Basic Data Types

Python allows you to assign values to multiple variables at the same time. E.g.:

```
a = b = c = 1

aaa,bbb,ccc = 1, 2, "unige"
```

Listing 6: Multiple Variable Assignment

There are six standard data types in Python3: Number, String (string), List (list), Tuple, Set (collection), Dictionary, Among the six standard data types of Python3: Immutable data (3): Number (number), String (string), Tuple (tuple);Variable data (3): List (list), Dictionary (dictionary), Set (collection). We will discuss them later.

- **Number**
  Python3 supports int, float, bool, complex (complex numbers). Like most languages, assignment and evaluation of numeric types is straightforward. The built-in type() function can be used to query what type of object a variable refers to.

```
a, b, c, d = 20, 5.5, True, 4+3j

print(type(a), type(b), type(c), type(d))

>>> <class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

Listing 7: Function type() Example

- **String**
  Strings in Python are enclosed in single quotes or double quotes while special characters are escaped with a backslash. The grammatical format of string interception is as follows: variable [head subscript: tail subscript]

```
str = 'unigestat'

print (str)
# output is string
>>> unigestat
print (str[0:-1])
# Output is all characters from the first to the second last
>>> unigesta
```

```
print (str[0])
# output is the first character of the string
>>> u
print (str[2:5])
# Output is the characters from the third to the fifth
>>> ige
print (str[2:])
# Output is all characters starting from the third
>>> igestat
print (str * 2)
# output is the string twice
>>> unigestatunigestat
print (str + "TEST")
# merge
>>> unigestatTEST
```

Listing 8: String Example 1

Python uses the backslash to escape special characters. If you don't want the backslash to be escaped, you can add an r in front of the string to represent the original string.

```
 print('unige\stats')
>>>unige
>>>stats

print(r'unige\stats')
>>> unige\stats
```

Listing 9: String Example 2

- **List**
  List is the most frequently used data type in Python. Lists can complete the data structure implementation of most collection classes. The types of elements in the list can be different, it supports numbers, strings can even contain lists (so-called nesting).A list is a comma-separated list of elements written between square brackets [].

  Like strings, lists can also be indexed and truncated. After the list is truncated, a new list containing the required elements is returned. The syntax format of list interception is as follows:

  variable [head subscript: tail subscript]

```
list1 = ['abcd', 786, 2.23, 'unige', 70.2]
tinylist = [123, 'unige']

print (list1)
# output is the complete list
>>> ['abcd', 786, 2.23, 'unige', 70.2]
print (list1[0])
# output is the first element of the list
```

```
>>> abcd
print (list1[1:3])
# output is from the second to the third element
>>> [786, 2.23]
print (list1[2:])
# output is all elements starting from the third element
>>> [2.23, 'unige', 70.2]
print (tinylist * 2)
# output is the list twice
>>> [123, 'unige', 123, 'unige']
print (list1 + tinylist)
# merge
>>> ['abcd', 786, 2.23, 'unige', 70.2, 123, 'unige']
```

Listing 10: List Example

The elements of a list can be changed:

```
a = [1, 2, 3, 4, 5, 6]
a[0] = 9
a[2:5] = [13, 14, 15]
print(a)
>>> [9, 2, 13, 14, 15, 6]
a[2:5] = [] # Set the corresponding element value to []
print(a)
>>> [9, 2, 6]
```

Listing 11: List Example 2

List has many built-in methods, such as append(), pop(), etc. Interested students can refer to relevant information.

- **Tuple**
  A tuple is similar to list, except that the elements cannot be modified. The element types in the tuple can also be different:

```
tuple1 = ('abcd', 786, 2.23, 'unige', 70.2)
tinytuple = (123, 'unige')

print (tuple1)
# output is the complete tuple
>>> ('abcd', 786, 2.23, 'unige', 70.2)
print (tuple1[0])
# output is the first element of the tuple
>>>  abcd
print (tuple1[1:3])
# output starts from the second element to the third element
>>> (786, 2.23)
print (tuple1[2:])
# output all elements starting from the third element
>>> (2.23, 'unige', 70.2)
print (tinytuple * 2)
```

```
# output is tuple twice
>>> (123, 'unige', 123, 'unige')
print (tuple1 + tinytuple)
# merge
>>> ('abcd', 786, 2.23, 'unige', 70.2, 123, 'unige')
```

Listing 12: Tuple Example 1

Similar to strings, tuples can be indexed and the subscript index starts from 0, and -1 is the position from the end. In fact, strings can be thought of as a special kind of tuple.

```
tup = (1, 2, 3, 4, 5, 6)
print(tup[0])
>>> 1
print(tup[1:5])
>>> (2, 3, 4, 5)
tup[0] = 11 # Operations that modify tuple elements are illegal
>>> Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Listing 13: Tuple Example 2

- **Set**
  A set is composed of one or several wholes of various shapes and sizes, and the things or objects that constitute a set are called elements or members. The basic functionality is to perform membership tests and remove duplicate elements. Sets can be created using curly braces or the set() function. Note: To create an empty set, you must use set() instead of , because is used to create an empty dictionary.

```
parame = {value01,value02,...}
# or
set(value)
```

Listing 14: Set Example 1

Here are two examples using set.

```
sites = {'Google', 'Taobao', 'Runoob', 'Facebook', 'Zhihu', 'Baidu'}

print(sites)
# Output collection, duplicate elements are automatically removed

>>> {'Zhihu', 'Baidu', 'Taobao', 'Runoob', 'Google', 'Facebook'}

# membership test
if 'Runoob' in sites:
     print('Runoob is in the collection')
else:
     print('Runoob is not in the set')
```

```

>>> 'Runoob is in the set'

# set can perform set operations
a = set('abracadabra')
b = set('alacazam')

print(a)
>>> {'b', 'c', 'a', 'r', 'd'}
print(a - b) # difference between a and b
>>> {'r', 'b', 'd'}
print(a | b) # union of a and b
>>> {'b', 'c', 'a', 'z', 'm', 'r', 'l', 'd'}
print(a & b) # intersection of a and b
>>> {'c', 'a'}
print(a ^ b) # elements that do not exist in a and b
>>> {'z', 'b', 'm', 'r', 'l', 'd'}
```

Listing 15: Set Example 2

- **Dictionary**
  Dictionaries are another very useful built-in data type in Python. A list is an ordered collection of objects, and a dictionary is an unordered collection of objects. The difference between the two is that the elements in the dictionary are accessed by key, not by offset. A dictionary is a mapping type, and a dictionary is marked with , which is an unordered collection of key (key): value (value). Keys must use immutable types. Within the same dictionary, keys must be unique.

```
dict1 = {}
dict1['one'] = "1 - unige"
dict1[2] = "2 - stats"

tinydict = {'name': 'gsem', 'code':1, 'site': 'https://www.unige.ch/gsem'}


print (dict1['one']) # output the value whose key is 'one'
>>> 1 - unige
print (dict1[2]) # output the value whose key is 2
>>> 2 - stats
print (tinydict) # output the complete dictionary
>>> {'name': 'gsem', 'code': 1, 'site': 'https://www.unige.ch/gsem'}
print (tinydict.keys()) # output all keys
>>> dict_keys(['name', 'code', 'site'])
print (tinydict.values()) # output all values
>>> dict_values(['gsem', 1, 'https://www.unige.ch/gsem'])
```

Listing 16: Dictionary Example

- **Data Type Conversions**
  Sometimes, we need to convert the built-in data type, data type conversion, in general, you only need to use the data type as the function name.Python data type conversion can

be divided into two types: Implicit type conversions - auto-completion and Explicit type conversion - need to use type function to convert.

1.Implicit type conversions
In implicit type conversion, Python automatically converts one data type to another without our intervention. In the following example, we operate on two different types of data, and the lower data type (integer) is converted to the higher data type (float) to avoid data loss.

```
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
>>> datatype of num_int : <class 'int'>

print("datatype of num_flo:",type(num_flo))
>>> datatype of num_flo : <class 'float'>

print("value of num_new:",num_new)
>>> value of num_new: 124.23

print("datatype of num_new:",type(num_new))
>>> datatype of num_new : <class 'float'>
```

Listing 17: Implicit Type Conversions 1

Code analysis:
In this example, we add two variables num-int and num-flo of different data types and store them in the variable num-new. Then we check the data types of the three variables. In the output, we see that num-int is an integer and num-flo is a float. Also, the new variable num-new is float because Python converts smaller data types to larger data types to avoid data loss. Let's look at another example, adding integer data and string data:

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))
print(num_int+num_str)
```

Listing 18: Implicit Type Conversions 2

output as follows:

```
>>> Data type of num_int: <class 'int'>
>>> Data type of num_str: <class 'str'>
>>> Traceback (most recent call last):
  File "/Python-test/test.py", line 7, in <module>
```

```
    print(num_int+num_str)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Listing 19: Implicit Type Conversions 2 Output

It can be seen from the output that the operation results of integer and string types will report an error, and TypeError will be output. Python cannot use implicit conversions in this case. However, Python provides a solution for these types of situations called explicit conversions.

2.Explicit type conversion
In explicit type conversion, the user converts the data type of the object to the desired data type. We use predefined functions like int(), float(), str() etc. to perform explicit type conversions.

int() converts to an integer:

```
x = int(1)
# x output is 1

y = int(2.8)
# y output is 2

z = int("3")
# z output is 3
```

Listing 20: Explicit Type Conversion 1

float() converts to a float:

```
x = float(1)
# x output is 1.0

y = float(2.8)
# y output is 2.8

z = float("3")
# z output is 3.0

w = float("4.2")
# w output is 4.2
```

Listing 21: Explicit Type Conversion 2

str() converts to a string type:

```
x = str("s1")
# The output of x is 's1'

y = str(2)
# y output is '2'
```

```

z = str(3.0)
# z output is '3.0'
```

Listing 22: Explicit Type Conversion 3

Operations on integer and string types can be done by those function.

```
num_int = 123
num_str = "456"

print("num_int data type is:",type(num_int))
print("Before type conversion, the data type of num_str is:",type(num_str))

num_str = int(num_str) # cast to integer
print("After type conversion, the data type of num_str is:",type(num_str))

num_sum = num_int + num_str

print("The result of adding num_int and num_str is:",num_sum)
print("sum data type is:",type(num_sum))
```

Listing 23: Explicit Type Conversion 4

output as follows:

```
The num_int data type is: <class 'int'>
Before type conversion, the data type of num_str is: <class 'str'>
After type conversion, the data type of num_str is: <class 'int'>
The result of adding num_int and num_str is: 579
The sum data type is: <class 'int'>
```

Listing 24: Explicit Type Conversion 4 Output

- **If-else Statement**
  The general form of an if statement in Python is as follows:

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

Listing 25: If-else Statement Example 1

If "condition 1" is True the "statement block 1" block statement will be executed. If "condition 1" is False, "condition 2" will be evaluated. If "condition 2" is True the "statement block 2" block statement will be executed. If "condition 2" is False, the "statement block 3" block statement will be executed.

```
dog_age = 15

if age <= 0:
     print("You are kidding me!")
elif age == 1:
     print("Equivalent to a 14-year-old.")
elif age == 2:
     print("Equivalent to a 22-year-old.")
elif age > 2:
     human = 22 + (age -2)*5
     print("corresponding to human age: ", human)

>>> corresponding to human age: 87
```

Listing 26: If-else Statement Example 2

In nested if statements, you can place an if...elif...else structure inside another if...elif...else structure. The following is an example of nested if-else statements.

```
num = 6

if num%2==0:
     if num%3==0:
         print ("The number you entered is divisible by 2 and 3")
     else:
         print ("The number you entered is divisible by 2, but not by 3")
else:
     if num%3==0:
         print ("The number you entered is divisible by 3, but not by 2")
     else:
         print ("The number you entered is not divisible by 2 and 3")

>>> "The number you entered is divisible by 2 and 3"
```

Listing 27: If-else Statement Example 3

Python 3.10 adds the conditional judgment of match...case, no need to use a series of if-else to judge. The object after the match will be matched with the content after the case in turn. If the match is successful, the matched expression will be executed, otherwise it will be skipped directly. The underscore can match everything.The syntax format is as follows:

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

Listing 28: Match Case Statement Example 1

The following is an example of Match Case statements.

```
mystatus = 400
print(http_error(400))

def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"

>>> Bad request
```

Listing 29: Match Case Statement Example 2

- **For-loop Statement**
  A Python for loop can iterate over any iterable object, such as a list or a string.The general format of a for loop is as follows:

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>
```

Listing 30: For-loop Statement Example 1

Python for loop example:

```
sites = ["Baidu", "Google","Runoob","Facebook"]
for site in sites:
     print(site)

>>> Baidu
    Google
    Runoob
    Taobao
```

Listing 31: For-loop Statement Example 2

Integer range values can be used with the range() function:

```
for number in range(1, 6):
    print(number)

>>> 1
```

```
    2
    3
    4
    5
```

Listing 32: For-loop Statement Example 3

In Python, the for...else statement is used to execute a block of code after the loop ends. When the loop is executed (that is, all elements in the iterable are traversed), the code in the else clause will be executed. If a break statement is encountered during the loop, the loop will be interrupted, and the else clause will not be executed at this time.

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")

>>> 0
    1
    2
    3
    4
    5
    Finally finished!
```

Listing 33: For-loop Statement Example 4

The break statement is used in the following for example, the break statement is used to jump out of the current loop body, and the else clause will not be executed:

```
sites = ["Baidu", "Google","Taobao","Runoob"]
for site in sites:
    if site == "Taobao":
        print("Taobao!")
        break
    print("loop" + site)
else:
    print("no loop!")
print("finish loop!")

>>> loop Baidu
    loop Google
    Taobao!
    finish loop!
```

Listing 34: For-loop Statement Example 5

- **While-loop Statement**
  The general form of a while statement in Python:

```
while Judgment condition (condition):
```

```
    Execute statements...
```

Listing 35: While-loop Statement Example 1

The following example uses while to calculate the sum from 1 to 100:

```
n = 100

sum = 0
counter = 1
while counter <= n:
    sum = sum + counter
    counter += 1

print("The sum of 1 to %d is: %d" % (n,sum))

>>> The sum of 1 to 100 is: 5050
```

Listing 36: While-loop Statement Example 2

Use break in while-loop Statement:

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('The end of the loop.')
>>> 4
    3
    The end of the loop.
```

Listing 37: While-loop Statement Example 3

Use continue in while-loop Statement:

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        continue
    print(n)
print('The end of the loop.')
>>> 4
    3
    1
    0
    The end of the loop.
```

Listing 38: While-loop Statement Example 4

- **Function**
  Python defines functions using the def keyword, and the general format is as follows:

```
def function name (parameters):
     body
```

Listing 39: Function Example 1

Let's use a function to output "Hello World!":

```
def hello() :
    print("Hello World!")
hello()
>>> "Hello World!"
```

Listing 40: Function Example 2

More complex applications, with parameter variables in the function. Task: compare two numbers and return the larger number:

```
def max(a, b):
    if a > b:
        return a
    else:
        return b

a = 4
b = 5
print(max(a, b))
>>> 5
```

Listing 41: Function Example 3

Now you have mastered the basic python programming ability. Next, let me try some more advanced applications.

2 ADVANCED PART

- **Data Structure**
  In this chapter, we mainly introduce the Python data structure based on the knowledge points learned earlier.

  1.Advanced part of list

  Lists in Python are mutable, which is the most important feature that distinguishes them from strings and tuples. In one sentence, lists can be modified, but strings and tuples cannot.

Here are the methods for lists in Python:

Methods	Description
list.append(x)	Adds an element, equivalent to a[len(a):] = [x]
list.extend(L)	Extends the list, equivalent to a[len(a):] = L
list.insert(i, x)	Inserts an element at the specified position.
list.remove(x)	Removes the first element in the list with value x.
list.pop([i])	Removes the element from the specified position.
list.clear()	Removes all items from the list, equals del a[:]
list.index(x)	Returns the index of the first element in the list whose value is x
list.count(x)	Returns the number of times x occurs in the list
list.sort()	Sorts the elements in the list
list.reverse()	Reverses the elements in the list
list.copy()	Returns a shallow copy of the list, equal to a[:]

The following example demonstrates most of the list methods:

```
a = [66.25, 333, 333, 1, 1234.5]
print(a.count(333), a.count(66.25), a.count('x'))
>>> 2 1 0
a.insert(2, -1)
a.append(333)
print(a)
>>> [66.25, 333, -1, 333, 1, 1234.5, 333]
print(a.index(333))
>>>1
a.remove(333)
print(a)
>>>[66.25, -1, 333, 1, 1234.5, 333]
a.reverse()
print(a)
>>>[333, 1234.5, 1, 333, -1, 66.25]
a.sort()
print(a)
>>>[-1, 1, 66.25, 333, 333, 1234.5]
```

Listing 42: Data Structure 1

The list method makes it convenient to use the list as a stack. The stack is a specific data structure, and the first element entered is the last one released (last in, first out). An element can be added to the top of the stack with the append() method.

Use the pop() method without specifying an index to release an element from the top of the stack. E.g:

```
stack = [3, 4, 5]
stack.append(6)
stack.append(7)
stack
```

```
>>>[3, 4, 5, 6, 7]
stack.pop()
>>>7
stack
>>>[3, 4, 5, 6]
stack.pop()
>>>6
stack.pop()
>>>5
stack
>>>[3, 4]
```

Listing 43: Data Structure 2

2.Advanced part of tuples and sequences

As follows, tuples are always output with parentheses to facilitate the correct expression of nested structures. They may or may not be entered with or without parentheses, but they are usually required (if the tuple is part of a larger expression).

```
t = 12345, 54321, 'hello!'
t[0]
>>> 12345
t
>>> (12345, 54321, 'hello!')
# Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
>>> ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

Listing 44: Data Structure 3

3.Advanced part of Set

A set is an unordered collection of unique elements. Basic functionality includes relationship testing and elimination of duplicate elements.

Collections can be created with braces (). Note: If you want to create an empty set, you must use set() instead of ; the latter creates an empty dictionary.

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

print(basket) # delete duplicates
>>> {'orange', 'banana', 'pear', 'apple'}

'orange' in basket # detect members
>>> True

'crabgrass' in basket
>>> False

# The following demonstrates the operation of two collections
 a = set('abracadabra')
b = set('alacazam')
```

```
a # unique letter in a
>>> {'a', 'r', 'b', 'c', 'd'}

a - b # letter in a but not in b
>>> {'r', 'd', 'b'}

a | b # letter in a or b
>>> {'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

a & b # letters in both a and b
>>> {'a', 'c'}

a ^ b # letter in a or b, but not both a and b
>>> {'r', 'd', 'b', 'm', 'z', 'l'}
```

Listing 45: Data Structure 4

4.traversal skills traversal skills

When traversing through a dictionary, the key and the corresponding value can be interpreted at the same time using the items() method:

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.items():
    print(k, v)

>>> gallahad the pure
    robin the brave
```

Listing 46: Data Structure 5

When traversing in a sequence, the index position and corresponding value can be obtained at the same time using the enumerate() function:

```
for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)

>>> 0 tic
    1 tac
    2 toe
```

Listing 47: Data Structure 6

Iterating over two or more sequences simultaneously can be combined using zip():

```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print('What is your {0}?  It is {1}.'.format(q, a))

>>> What is your name?  It is lancelot.
```

```
    What is your quest?  It is the holy grail.
    What is your favorite color?  It is blue.
```

Listing 48: Data Structure 7

To traverse a sequence in reverse, first specify the sequence, then call the reversed() function:

```
for i in reversed(range(1, 10, 2)):
    print(i)
>>> 9
    7
    5
    3
    1
```

Listing 49: Data Structure 8

To iterate over a sequence in order, use the sorted() function to return a sorted sequence without modifying the original value:

```
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
for f in sorted(set(basket)):
    print(f)

>>> apple
    banana
    orange
    pear
```

Listing 50: Data Structure 9

- **Errors and Exceptions**
  As a Python beginner, when you first learn Python programming, you often see some error messages, which we did not mention before, and we will introduce them in this chapter.There are two types of errors in Python that are easily recognizable: syntax errors and exceptions.Python assert (assertion) is used to judge an expression and trigger an exception when the expression condition is false.

  Python grammatical errors or parsing errors are often encountered by beginners, as shown in the following example:

```
while True print('Hello world')

>>> File "<stdin>", line 1, in ?
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

Listing 51: Errors and Exceptions 1

In this example, the function print() is checked for an error because it is missing a colon : in front of it.The parser points out the wrong line and marks a small arrow at the location of the first error found.

Even if the syntax of a Python program is correct, errors may occur when running it. Errors detected at runtime are called exceptions.Most exceptions will not be handled by the program, and are displayed here in the form of error messages:

```
10 * (1/0)
# 0 cannot be used as a divisor, triggering an exception
>>> Traceback (most recent call last):
   File "<stdin>", line 1, in ?
ZeroDivisionError: division by zero

4 + spam*3
# spam is undefined, trigger exception
>>> Traceback (most recent call last):
   File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined

'2' + 2
# int cannot be added to str, an exception is triggered
>>> Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Listing 52: Errors and Exceptions 2

Exceptions occur in different types, which are all printed as part of the message: the types in the example are ZeroDivisionError, NameError and TypeError.

The front part of the error message shows the context where the exception occurred, and the specific information is displayed in the form of a call stack.We can be try to show exception. But I won't discuss too much here, and those who are interested can refer to relevant information.

- **Random**
  The Python random module is mainly used to generate random numbers. The random module implements pseudorandom number generators for various distributions. To use the random function you must first import.

  Next we use the random() method to return a random number in the half-open interval [0,1), including 0 but excluding 1.

```
# import the random package
import random

# Generate random numbers
print(random. random())

```

```
>>> 0.4784904215869241
```

Listing 53: Random 1

The seed() method changes the seed of the random number generator, and this function can be called before calling other random module functions.

```
import random

random. seed()
print ("Generate a random number using the default seed:", random.random())
>>> Generate random numbers with default seed: 0.7908102856355441

print ("Generate a random number using the default seed:", random.random())
>>> Generate random numbers with default seed: 0.81038961519195

random.seed(10)
print("Generate a random number using the integer 10 seed:", random.random())
>>> Generate random numbers with default seed: 0.81038961519195

random.seed(10)
print("Generate a random number using the integer 10 seed:", random.random())
>>> Generate random numbers with default seed: 0.81038961519195

random.seed("hello",2)
print ("Generate a random number using a string seed: ", random.random())
>>> Generate random numbers with default seed: 0.81038961519195
```

Listing 54: Random 2

We have no way to provide all the random variables that random generators want, which obey a specific distribution. But we've provided an explanation of the method below so you can find it when you need it.

Methods	Description
seed()	Initialize the random number generator
getrandbits(k)	Return a non-negative Python integer with k random bits
randrange()	Return a randomly selected element from range(a, b, c)
randint(a, b)	Return a random integer N such that a <= N <= b.
choice(seq)	Returns a element from the non-empty sequence seq.
shuffle(x[, random])	Randomly shuffle the sequence x
sample(N, k)	Returns k-length list of unique elements from the N
random()	Returns the random number in the range [0.0, 1.0).
uniform()	Returns the random number in the range [0.0, 1.0).
betavariate(alpha, beta)	Returns Beta distribution
expovariate(lambd)	Returns Exponent distribution
gammavariate()	Returns Gamma distribution
gauss(u, sigma)	Returns normal distribution(faster than the below one)
lognormvariate(u, sigma)	Returns Lognormal distribution.
normalvariate(u, sigma)	Returns normal distribution.
vonmisesvariate(u, sigma)	Returns von Mises distribution.
paretovariate(u, sigma)	Returns Pareto distribution
weibullvariate(u, sigma)	Returns Weibull distribution.

- **Pip**
  We know that there are many libraries available in python. We can use pipi to install and manage these libraries. In a word, pip is a Python package management tool that provides functions for finding, downloading, installing, and uninstalling Python packages.

  To check if pip is installed you can use the following command on the terminal:

```
pip --version
```

Listing 55: Pip Example 1

Download the installation package using the following command: pip install some-package-name For example, we install the numpy package:

```
pip install numpy
```

Listing 56: Pip Example 2

We can also easily remove packages with the following command: pip uninstall some-package-name

For example we remove the numpy package:

```
pip uninstall numpy
```

Listing 57: Pip Example 3

If we want to see the packages we have installed, we can use the following command:

```
pip list
```

Listing 58: Pip Example 4

The above is the content of all python guides. We only list some of the frequently used content. As time goes by, some of them may no longer apply, welcome to update and correct.

Part II.
R guidance

3 BASIC PART

R language is a mathematical programming language designed for mathematical researchers, mainly used for statistical analysis, graphics, and data mining.
Both the R language and the C language are the research results of Bell Labs, but they have different focus areas. The R language is an interpreted language for mathematical researchers, while the C language is designed for computer software engineers.

The R language is an interpreted language (different from the compiled language of the C language), and its execution speed is much slower than that of the C language, which is not conducive to optimization. But it provides a richer data structure operation at the grammatical level and can output text and graphic information very conveniently, so it is widely used in mathematics, especially in the field of statistics.
R language official website: https://cran.r-project.org/.

List of official mirror sites: https://cran.r-project.org/mirrors.html

If you are slow to download files from r's official website, you can try to download from a mirror website.

- **R installation**

  The development environment of R language itself has a graphical development environment, which is different from many other engineering languages, so the development environment is best installed on an operating system designed for desktop personal computers (such as Windows, macOS or Ubuntu desktop version, etc.).

  First, we need to download the installation package for the R locale:

  Windows:
  Official address: https://cloud.r-project.org/bin/windows/base/

  Linux:
  Official address: https://cloud.r-project.org/bin/linux/

macOS:
Official address: https://cloud.r-project.org/bin/macosx/

- **Grammar**
  The "Hello, World!" program code in R language is as follows:

```
myString <- "Hello, World!"

print ( myString )
```

Listing 59: R Example

The above example assigns the string "Hello, World!" to the myString variable, and then uses the print() function to output.

Note: R language assignments use the left arrow <- symbol, but some newer versions also support the equals sign =.

- **Note**
  Comments are mainly used to analyze a piece of code, which can make it easier for readers to understand. Comments in programming languages will be ignored by the compiler and will not affect the execution of the code.

  Comments in general programming languages are divided into single-line comments and multi-line comments, but R language only supports single-line comments, and the comment symbol is #.

  In fact, if there are multiple lines of comments, we only need to add the sign to each line.

```
# This is my first programming code
myString <- "Hello, World!"

print( myString )
```

Listing 60: Note Example

- **Operations**
  The assignment symbol is a formal advantage and an operational disadvantage of the R language: the form is more suitable for mathematicians. After all, not all mathematicians are accustomed to using = as an assignment symbol.

  Operationally speaking, neither the < symbol nor the - symbol are very easy to type characters, which will make many programmers uncomfortable. Therefore, newer versions of the R language also support = as an assignment operator: The following examples demonstrate simple mathematical operations:

```
> 1 + 2 * 3
[1] 7
```

```
> (1 + 2) * 3
[1] 9
> 3 / 4
[1] 0.75
> 3.4 - 1.2
[1] 2.2
> 1 - 4 * 0.5^3
[1] 0.5
> 8 / 3 %% 2
[1] 8
> 8 / 4 %% 2
[1] Inf
> 3 %% 2^2
[1] 3
> 10 / 3 %/% 2
[1] 10
```

Listing 61: Operations Example

- **Data types**
  Data types refer to a broad system for declaring variables or functions of different types. The type of a variable determines how much space the variable is stored in, and how the stored bit pattern is interpreted. There are three basic data types in R language: numeric, logical, and text.

  There are two main types of numeric constants: scientific notation and general form.

  The logical type is often called Boolean in many other programming languages, and the only constant values are TRUE and FALSE. Logical vectors are mainly used for logical operations on vectors, for example:

```
> c(11, 12, 13) > 12
[1] FALSE FALSE  TRUE
```

Listing 62: Data types Example 1

The which function is a very common logical vector processing function, which can be used to filter the subscripts of the data we need:

```
> c(11, 12, 13) > 12
[1] FALSE FALSE  TRUE
```

Listing 63: Data types Example 2

For example, we need to filter data greater than or equal to 60 and less than 70 from a linear table:

```
> vector = c(10, 40, 78, 64, 53, 62, 69, 70)
> print(vector[which(vector >= 60 & vector < 70)])
```

```
[1] 64 62 69
```

Listing 64: Data types Example 3

Similar functions are all and any:

```
> all(c(TRUE, TRUE, TRUE))
[1] TRUE
> all(c(TRUE, TRUE, FALSE))
[1] FALSE
> any(c(TRUE, FALSE, FALSE))
[1] TRUE
> any(c(FALSE, FALSE, FALSE))
[1] FALSE
```

Listing 65: Data types Example 4

all() is used to check whether a logical vector is all TRUE, and any() is used to check whether a vector contains TRUE.

The most intuitive data type is the text type. Text is a string that often appears in other languages, and constants are enclosed in double quotes. In R, literal constants can be enclosed in single or double quotes, for example:

```
> 'UNIGE' == "UNIGE"
[1] TRUE
```

Listing 66: Data types Example 5

The string data type itself is not complicated. Here we focus on introducing the operation functions of strings:

```
> toupper("unige") # convert to uppercase
[1] "UNIGE"
> tolower("UNIGE") # convert to lowercase
[1] "unige"
> substr("123456789", 1, 5) # Intercept the string, from 1 to 5
[1] "12345"
> substring("1234567890", 5) # Intercept the string, from 5 to the end
[1] "567890"
> as.numeric("12") # convert string to number
[1] 12
> as.character(12.34) # convert a number to a string
[1] "12.34"
> strsplit("2019;10;1", ";") # delimiter to split the string
[[1]]
[1] "2019" "10" "1"
> gsub("/", "-", "2019/10/1") # replace string
[1] "2019-10-1"
```

Listing 67: Data types Example 6

If we divide by object type, there are the following 6 types: vector, list, matrix, array, factor, data.frame.

1.Vector

Vectors are an indispensable tool in mathematical operations - our most common vector is a two-dimensional vector, which is bound to be used in a planar coordinate system. From the perspective of data structure, a vector is a linear table, which can be regarded as an array. The existence of vectors as a type in R language makes it easier to manipulate vectors:

```
> a = c(3, 4)
> b = c(5, 0)
> a + b
[1] 8 4
```

Listing 68: Data types Example 7

c() is a function that creates vectors. Here two 2D vectors are added to get a new 2D vector (8, 4). If a two-dimensional vector and a three-dimensional vector are operated, the mathematical meaning will be lost. Although the operation will not stop, it will be warned. I suggest that you get used to putting an end to this kind of situation.

Each element in the vector can be taken out individually by subscript:

```
> a = c(10, 20, 30, 40, 50)
> a[2]
[1] 20
```

Listing 69: Data types Example 8

Note: The "subscript" in the R language does not represent the offset, but represents the number, that is to say, it starts from 1.

R can also easily take out a part of the vector:

```
> a[1:4] # Take items 1 to 4, including items 1 and 4
[1] 10 20 30 40
> a[c(1, 3, 5)] # get items 1, 3, 5
[1] 10 30 50
> a[c(-1, -5)] # remove items 1 and 5
[1] 20 30 40
```

Listing 70: Data types Example 9

Vectors support scalar calculations:

```
> c(1.1, 1.2, 1.3) - 0.5
[1] 0.6 0.7 0.8
> a = c(1,2)
> a^2
[1] 1 4
```

Listing 71: Data types Example 10

"Vector" has some commonly used linear table processing functions: vector sorting/vector statistics/vector generation

Example of vector sorting:

```
> a = c(1, 3, 5, 2, 4, 6)
> sort(a)
[1] 1 2 3 4 5 6
> rev(a)
[1] 6 4 2 5 3 1
> order(a)
[1] 1 4 2 5 3 6
> a[order(a)]
[1] 1 2 3 4 5 6
```

Listing 72: Data types Example 11

Note: The order() function returns a vector of subscripts after the vector is sorted.

Examples of vector statistics:

```
> sum(1:5)
[1] 15
> sd(1:5)
[1] 1.581139
> range(1:5)
[1] 1 5
```

Listing 73: Data types Example 12

Vectors can be generated using the c() function, or a continuous sequence using the min:max operator. If you want to generate an arithmetic sequence with gaps, you can use the seq function. Examples of vector generation:

```
> seq(1, 9, 2)
[1] 1 3 5 7 9

> seq(0, 1, length.out=3)
[1] 0.0 0.5 1.0

> rep(0, 5)
[1] 0 0 0 0 0
```

Listing 74: Data types Example 13

2.Matrix

The R language provides a matrix type for the study of linear algebra. This data structure is very similar to a two-dimensional array in other languages, but R provides language-level matrix operation support. Elements in a matrix can be numbers, symbols, or mathematical expressions. A matrix in R language can be created using the matrix() function.

```
# byrow is TRUE elements are arranged by row
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)

# Ebyrow is FALSE elements are arranged in columns
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)

# define row and column names
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

Listing 75: Data types Example 14

The output is:

```
[,1] [,2] [,3]
[1,] 3 4 5
[2,] 6 7 8
[3,] 9 10 11
[4,] 12 13 14
      [,1] [,2] [,3]
[1,] 3 7 11
[2,] 4 8 12
[3,] 5 9 13
[4,] 6 10 14
      col1 col2 col3
row1 3 4 5
row2 6 7 8
row3 9 10 11
row4 12 13 14
```

Listing 76: Output of Data types Example 14

The R language matrix provides the t() function, which can exchange the rows and columns of the matrix. For example, if there is a matrix with m rows and n columns, use the t() function to convert it into a matrix with n rows and m columns.

```
# Create a matrix with 2 rows and 3 columns
```

```
M = matrix( c(2,6,5,1,10,4), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
      [,1] [,2] [,3]
[1,] 2 6 5
[2,] 1 10 4
# convert to a matrix with 3 rows and 2 columns
print(t(M))
```

Listing 77: Data types Example 15

The output is:

```
[,1] [,2] [,3]
[1,] 2 6 5
[2,] 1 10 4
[1] "-----Conversion-----"
      [,1] [,2]
[1,] 2 1
[2,] 6 10
[3,] 5 4
```

Listing 78: Output of Data types Example 15

If you want to get the matrix elements, you can use the column index and row index of the element, similar to the coordinate form.

```
# define row and column names
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# create matrix
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
# Get the elements of the first row and third column
print(P[1,3])

# Get the element of the fourth row and second column
print(P[4,2])

# get the second row
print(P[2,])

# Get the third column
print(P[,3])
```

Listing 79: Data types Example 16

The output is:

```
col1 col2 col3
row1 3 4 5
row2 6 7 8
row3 9 10 11
row4 12 13 14
[1] 5
[1] 13
col1 col2 col3
     6 7 8
row1 row2 row3 row4
     5 8 11 14
```

Listing 80: Output of Data types Example 16

Example of Matrix Calculations

```r
# Create a matrix with 2 rows and 3 columns
matrix1 <- matrix(c(7, 9, -1, 4, 2, 3), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(6, 1, 0, 9, 3, 2), nrow = 2)
print(matrix2)

# add two matrices
result <- matrix1 + matrix2
cat("Addition result: ","\n")
print(result)

# Subtract two matrices
result <- matrix1 - matrix2
cat("Subtraction result: ","\n")
print(result)

# Multiply two matrices
result <- matrix1 * matrix2
cat("multiplication result: ","\n")
print(result)

# Divide two matrices
result <- matrix1 / matrix2
cat("Division result: ","\n")
print(result)
```

Listing 81: Data types Example 17

The output is:

```
[,1] [,2] [,3]
[1,] 7 -1 2
[2,] 9 4 3
     [,1] [,2] [,3]
[1,] 6 0 3
[2,] 1 9 2
Addition result:
     [,1] [,2] [,3]
[1,] 13 -1 5
[2,] 10 13 5
Subtract result:
     [,1] [,2] [,3]
[1,] 1 -1 -1
[2,] 8 -5 1

Multiplication result:
     [,1] [,2] [,3]
[1,] 42 0 6
[2,] 9 36 6
Divide result:
         [,1] [,2] [,3]
[1,] 1.166667-Inf 0.6666667
[2,] 9.000000 0.4444444 1.5000000
```

Listing 82: Output of Data types Example 17

3.List

A list is a collection of objects in the R language that can be used to hold different types of data, which can be numbers, strings, vectors, another list, etc., and of course can also contain matrices and functions. The R language creates lists using the list() function.

In the following example, we create a list containing strings, vectors and numbers:

```
list_data <- list("facebook", "google", c(11,22,33), 123, 51.23, 119.1)
print(list_data)
```

Listing 83: Data types Example 18

The output is:

```
[[1]]
[1] "facebook"

[[2]]
[1] "google"

[[3]]
[1] 11 22 33

[[4]]
[1] 123

[[5]]
[1] 51.23

[[6]]
[1] 119.1
```

Listing 84: Output of Data types Example 18

We can use the names() function to name the elements of the list:

```
# list contains vectors, matrices, lists
list_data <- list(c("Google","facebook","Tesla"), matrix(c(1,2,3,4,5,6), nrow = 2),
    list("facebook",12.3))

# Set names for list elements
names(list_data) <- c("Sites", "Numbers", "Lists")

# show list
print(list_data[1])

# access the third element of the list
print(list_data[3])

# access the first vector element
print(list_data$Numbers)
```

Listing 85: Data types Example 19

The output is:

```
$Sites
[1] "Google","facebook","Tesla"

$Numbers
     [,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

$Lists
$Lists[[1]]
[1] "facebook"

$Lists[[2]]
[1] 12.3
```

Listing 86: Output of Data types Example 19

The elements in the list can be accessed using the index. If we use the names() function to name, we can also use the corresponding name to access:

```
# list contains vectors, matrices, lists
list_data <- list(c("Google","facebook","Tesla"), matrix(c(1,2,3,4,5,6), nrow = 2),
    list("facebook",12.3))

# Set names for list elements
names(list_data) <- c("Sites", "Numbers", "Lists")

# show list
print(list_data[1])

# access the third element of the list
print(list_data[3])

# access the first vector element
print(list_data$Numbers)
```

Listing 87: Data types Example 20

The output is:

```
$Sites
[1] "Google","facebook","Tesla"

$Lists
$Lists[[1]]
[1] "facebook"

$Lists[[2]]
[1] 12.3


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

Listing 88: Output of Data types Example 20

We can add, delete, and update operations on the list, as shown in the following example:

```
# list contains vectors, matrices, lists
list_data <- list(c("Google","facebook","Tesla"), matrix(c(1,2,3,4,5,6), nrow = 2),
    list("facebook",12.3))

# Set names for list elements
names(list_data) <- c("Sites", "Numbers", "Lists")

# add elements
list_data[4] <- "new element"
print(list_data[4])

# delete element
list_data[4] <- NULL

# Output is NULL after deletion
print(list_data[4])

# update element
list_data[3] <- "I replaced the third element"
print(list_data[3])
```

Listing 89: Data types Example 21

The output is:

```
[[1]]
[1] "New element"

$<NA>
NULL

$Lists
[1] "I replaced the third element"
```

Listing 90: Output of Data types Example 21

We can combine multiple lists into one list using the c() function:

```
# Create two lists
list1 <- list(1,2,3)
list2 <- list("Google","facebook","Tesla")

# Merge list
merged.list <- c(list1,list2)

# display the merged list
print(merged.list)
```

Listing 91: Data types Example 22

The output is:

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

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Google"

[[5]]
[1] "facebook"

[[6]]
[1] "Tesla"
```

Listing 92: Output of Data types Example 22

To convert a list into a vector, you can use the unlist() function to convert the list into a vector, which is convenient for us to perform arithmetic operations::

```r
# create list
list1 <- list(1:5)
print(list1)

list2 <- list(10:14)
print(list2)

# convert to vector
v1 <- unlist(list1)
v2 <- unlist(list2)

print(v1)
print(v2)

# add two vectors
result <- v1+v2
print(result)
```

Listing 93: Data types Example 23

The output is:

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

[[1]]
[1] 10 11 12 13 14

[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 19
```

Listing 94: Output of Data types Example 23

4.Array

Arrays are also objects of the R language, and the R language can create one-dimensional or multi-dimensional arrays. The R language array is a collection of the same type. The matrix matrix we learned earlier is actually a two-dimensional array. The syntax of the array() function is as follows:

```r
array(data = NA, dim = length(data), dimnames = NULL)
```

Listing 95: Data types Example 24

In the following example, we create a two-dimensional array with 3 rows and 3 columns:

```
# Create two vectors of different lengths
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# create array
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

Listing 96: Data types Example 25

The output is:

```
, , 1

     [,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15

, , 2

     [,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15
```

Listing 97: Output of Data types Example 25

Use the dimnames parameter to set the names of the individual dimensions:

```
# Create two vectors of different lengths
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Create an array and set the names of each dimension
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
    matrix.names))
print(result)
```

Listing 98: Data types Example 26

The output is:

```
, , Matrix1

     COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15

, , Matrix2

     COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15
```

Listing 99: Output of Data types Example 26

If you want to get an array element, you can use the column index and row index of the element, similar to the coordinate form.

```
# Create two vectors of different lengths
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# create array
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names, column.names,
    matrix.names))

# Display the elements of the third row in the second matrix of the array
print(result[3,,2])

# Display the elements in the first row and third column of the first matrix of the array
print(result[1,3,1])

# output the second matrix
print(result[,,2])
```

Listing 100: Data types Example 27

The output is:

```
COL1 COL2 COL3
    3 12 15
[1] 13
     COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15
```

Listing 101: Output of Data types Example 27

Since arrays are composed of matrices of multiple dimensions, we can access array elements by accessing the elements of the matrix.

```
# Create two vectors of different lengths
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# create array
array1 <- array(c(vector1,vector2),dim = c(3,3,2))

# Create two vectors of different lengths
vector3 <- c(9,1,0)
vector4 <- c(6,0,11,3,14,1,2,6,9)
array2 <- array(c(vector1,vector2),dim = c(3,3,2))

# create matrix from array
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]

# matrix addition
result <- matrix1+matrix2
print(result)
```

Listing 102: Data types Example 28

The output is:

```
     [,1] [,2] [,3]
[1,] 10 20 26
[2,] 18 22 28
[3,] 6 24 30
```

Listing 103: Output of Data types Example 28

In addition, we can use the apply() element to perform cross-dimensional calculations on array elements

```
# Create two vectors of different lengths
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# create array
new.array <- array(c(vector1,vector2),dim = c(3,3,2))
print(new.array)

# Calculate the sum of the numbers in the first row of all matrices in the array
result <- apply(new. array, c(1), sum)
print(result)
```

Listing 104: Data types Example 29

The output is:

```
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

[1] 56 68 60
```

Listing 105: Output of Data types Example 29

5.Factor

Factors are used to store different types of data. For example, the gender of a person can be divided into two categories: male and female, and age can be divided into minors and adults. The R language creates factors using the factor() function, with a vector as an input parameter. factor() function syntax format:

```
factor(x = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)
```

Listing 106: Data types Example 30

The following example converts a character vector to a factor:

```
x <- c("Male", "Female", "Male", "Male", "Female")
sex <- factor(x)
print(sex)
print(is. factor(sex))
```

Listing 107: Data types Example 31

The output is:

```
[1] Male Female Male Male Female
Levels: Male Female
[1] TRUE
```

Listing 108: Output of Data types Example 31

The following example sets the factor level to c('male','female'):

```
x <- c("Male", "Female", "Male", "Male", "Female",levels=c('Male','Female'))
sex <- factor(x)
print(sex)
print(is. factor(sex))
```

Listing 109: Data types Example 32

The output is:

```
levels1 levels2
Male Female Male Male Female Male Female
Levels: Male Female
[1] TRUE
```

Listing 110: Output of Data types Example 32

Next, we use the labels parameter to add labels to each factor level. The character order of the labels parameter must be consistent with the character order of the levels parameter, for example:

```
sex=factor(c('f','m','f','f','m'),levels=c('f','m'),labels=c('female','male '),ordered=TRUE)
print(sex)
```

Listing 111: Data types Example 33

The output is:

```
[1] female male female female female male
Levels: female < male
```

Listing 112: Output of Data types Example 33

6.Data.frame

The R language data frame is created using the data.frame() function, and the syntax format is as follows:

```
data.frame(..., row.names = NULL, check.rows = FALSE,
            check.names = TRUE, fix.empty.names = TRUE,
            stringsAsFactors = default. stringsAsFactors())
```

Listing 113: Data types Example 34

The following creates a simple data frame, including name, job number, monthly salary:

```
table = data. frame(
     name = c("alex", "venti"),
     Job number = c("001","002"),
     Monthly salary = c(1000, 2000)

)
print(table) # view table data
```

Listing 114: Output of Data types Example 34

The output is:

```
Name Job Number Monthly Salary
1 alex 001 1000
2 venti 002 2000
```

Listing 115: Data types Example 35

The data structure of the data frame can be displayed through the str() function:

```
table = data. frame(
     name = c("alex", "venti"),
     Job number = c("001","002"),
     Monthly salary = c(1000, 2000)
)
# Get the data structure
str(table)
```

Listing 116: Data types Example 35

The output is:

```
'data.frame': 2 obs. of 3 variables:
  $ Name: chr "alex", "venti"
  $ Job number: chr "001" "002"
  $ Monthly salary: num 1000 2000
```

Listing 117: Output of Data types Example 35

summary() can display the summary information of the data frame:

```
table = data. frame(
     name = c("alex", "venti"),
     Job number = c("001","002"),
     Monthly salary = c(1000, 2000)

)
# show summary
print(summary(table))
```

Listing 118: Data types Example 36

The output is:

```
Name Job Number Monthly Salary
Length: 2 Length: 2 Min. : 1000
Class :character Class :character 1st Qu.:1250
Mode :character Mode :character Median :1500
                                    Mean :1500
                                    3rd Qu.:1750
                                    Max. :2000
```

Listing 119: Output of Data types Example 36

We can also extract specified columns:

```
table = data. frame(
     name = c("alex", "venti"),
     Job number = c("001","002"),
     Monthly salary = c(1000, 2000)
)
# extract the specified column
result <- data.frame(table$name,table$monthly salary)
print(result)
```

Listing 120: Data types Example 37

The output is:

```
table.name table.monthly salary
1 alex 1000
2 venti 2000
```

Listing 121: Output of Data types Example 37

We can read the data of a certain column of a specified row in a form similar to coordinates. Below we read the data of columns 1 and 2 of rows 2 and 3:

```
table = data. frame(
     Name = c("alex", "venti","huh"),
     Job number = c("001","002","003"),
     Monthly Salary = c(1000, 2000, 3000)
)
# Read the data in columns 1 and 2 of rows 2 and 3:
result <- table[c(2,3),c(1,2)]
print(result)
```

Listing 122: Data types Example 38

The output is:

```
name job number
2 venti 002
3 huh 003
```

Listing 123: Output of Data types Example 38

We can extend the existing data frame. In the following example, we add department columns:

```
table = data. frame(
     Name = c("alex", "venti","huh"),
     Job number = c("001","002","003"),
     Monthly Salary = c(1000, 2000, 3000)
)
# Add department column
table$Department <- c("Operation", "Technology", "Editor")

print(table)
```

Listing 124: Data types Example 39

The output is:

```
Name Job Number Monthly Salary Department
1 alex 001 1000 Operations
2 venti 002 2000 technology
3 huh 003 3000 edit
```

Listing 125: Output of Data types Example 39

We can use the rbind() function to combine multiple vectors into a data frame:

```
table = data. frame(
     Name = c("alex", "venti","huh"),
     Job number = c("001","002","003"),
     Monthly Salary = c(1000, 2000, 3000)
)
newtable = data.frame(
     name = c("s1", "s2"),
     Job number = c("101","102"),
     Monthly salary = c(5000, 7000)
)
# merge two dataframes
result <- rbind(table,newtable)
print(result)
```

Listing 126: Data types Example 40

The output is:

```
Name Job Number Monthly Salary
1 alex 001 1000
2 venti 002 2000
3 huh 003 3000
4 s1 101 5000
5 s2 102 7000
```

Listing 127: Output of Data types Example 40

We can use the cbind() function to combine multiple vectors into a data frame:

```
# create vector
sites <- c("Google","facebook","Tesla")
likes <- c(222,111,123)
url <- c("www.google.com","www.facebook.com","www.Tesla.com")

# Combine the vectors into a data frame
addresses <- cbind(sites,likes,url)

# view the data frame
print(addresses)
```

Listing 128: Data types Example 41

The output is:

```
      sites like url
[1,] "Google" "222" "www.google.com"
[2,] "facebook" "111" "www.facebook.com"
[3,] "Tesla" "123" "www.Tesla.com"
```

Listing 129: Output of Data types Example 41

- **If-else Statement**

  An if statement consists of a Boolean expression followed by one or more statements. The syntax format is as follows:

```
if(boolean_expression) {
     // statements that will be executed if the boolean expression is true
}
```

Listing 130: If-else Statement 1

Execute the code inside if the Boolean expression is true, and do not execute it if it is false.

```
x <- 50L
if(is. integer(x)) {
    print("X is an integer")
}

[1] "X is an integer"
```

Listing 131: If-else Statement 2

An if statement can be followed by an optional else statement, which executes if the Boolean expression is false. The syntax format is as follows:

```
if(boolean_expression) {
     // statement that will be executed if the boolean expression is true
} else {
     // statement that will be executed if the boolean expression is false
}
```

Listing 132: If-else Statement 3

The code inside the if block is executed if the Boolean expression boolean expression is true. If the Boolean expression is false, the code inside the else block is executed.

```
x <- c("Google","facebook","Tesla")

if("facebook" %in% x) {
    print("contains facebook")
} else {
    print("does not contain facebook")
}

[1] "contains facebook"
```

Listing 133: If-else Statement 4

summary() can display the summary information of the data frame:

```
x <- c("google","facebook","taobao")

if("weibo" %in% x) {
    print("The first if contains weibo")
} else if ("facebook" %in% x) {
    print("The second if contains facebook")
} else {
    print("not found")
}
# show summary
print(summary(table))

[1] "The second if contains facebook"
```

Listing 134: If-else Statement 5

- **Loop Statement**

The loop types provided by R language are: repeat loop, while loop, for loop The loop control statements provided by R language are: break statement, Next statement.

The repeat loop will execute the code until the conditional statement is true before exiting the loop, and the break statement is used to exit. The syntax format is as follows:

```
repeat {
     // code
     if(condition) {
        break
     }
}
```

Listing 135: Loop Statement 1

The following example exits the loop when the variable cnt, which is the count variable, is 5:

```
v <- c("Google","facebook")
cnt <- 2

repeat {
    print(v)
    cnt <- cnt+1

    if(cnt > 5) {
       break
    }
}
```

Listing 136: Loop Statement 2

The output is:

```
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
```

Listing 137: Loop Statement 3

A while loop statement in R repeatedly executes a target statement as long as a given condition is true. The syntax format is as follows:

```
while(condition)
{
    statement(s);
}
```

Listing 138: Loop Statement 4

The following example outputs the content in the while statement block when the variable cnt is less than 7, and cnt is the counting variable:

```
v <- c("Google","Runoob")
cnt <- 2

while (cnt < 7) {
    print(v)
    cnt = cnt + 1
}
```

Listing 139: Loop Statement 5

The output is:

```
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
```

Listing 140: Loop Statement 6

The for loop statement in the R programming language can repeatedly execute the specified statement, and the number of repetitions can be controlled in the for statement. The syntax format is as follows:

```
for (value in vector) {
     statements
}
```

Listing 141: Loop Statement 7

The following example outputs 26 letters for the first four letters:

```r
v <- LETTERS[1:4]
for (i in v) {
    print(i)
}
```

Listing 142: Loop Statement 8

The output is:

```
[1] "A"
[1] "B"
[1] "C"
[1] "D"
```

Listing 143: Loop Statement 9

The break statement in R language is inserted in the loop body to exit the current loop or statement and start the script to execute the following statement. If you use nested loops, the break statement will stop the execution of the innermost loop and start the execution of the outer loop statement.

The following example uses break to exit the loop when the variable cnt is 5, where cnt is the count variable:

```r
v <- c("Google","facebook")
cnt <- 2

repeat {
    print(v)
    cnt <- cnt+1

    if(cnt > 5) {
       break
    }
}
```

Listing 144: Loop Statement 10

The output is:

```
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
[1] "Google" "facebook"
```

Listing 145: Loop Statement 11

The next statement is used to skip the current loop and start the next loop (similar to Python's continue). The following example outputs the first 6 letters of the 26 letters, and skips the current cycle when the letter is D, and proceeds to the next cycle:

```r
v <- LETTERS[1:6]
for (i in v) {

    if (i == "D") { # D will not output, skip this loop and enter the next one
       next
    }
    print(i)
}
```

Listing 146: Loop Statement 12

The output is:

```
[1] "A"
[1] "B"
[1] "C"
[1] "E"
[1] "F"
```

Listing 147: Loop Statement 13

- **Function**

  A function is a group of statements that together perform a task. The R language itself provides many built-in functions, and of course we can also create our own functions. The function definition in R language uses the function keyword, and the general form is as follows:

```r
function_name <- function(arg_1, arg_2, ...) {
     // function body
}
```

Listing 148: Function 1

The following demonstrates how to define two functions:

```r
# Define a function for counting a series to the squared value
new. function <- function(a) {
    for(i in 1:a) {
       b <- i^2
       print(b)
    }
}
```

Listing 149: Function 2

Next we can call the function:

```r
new. function <- function(a) {
     for(i in 1:a) {
```

```
        b <- i^2
        print(b)
    }
  }

# Call the function and pass parameters
new. function(6)
```

Listing 150: Function 3

The output is:

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

Listing 151: Function 4

Now you have mastered the basic R programming ability. Next, let me try some more advanced applications.

4 ADVANCED PART

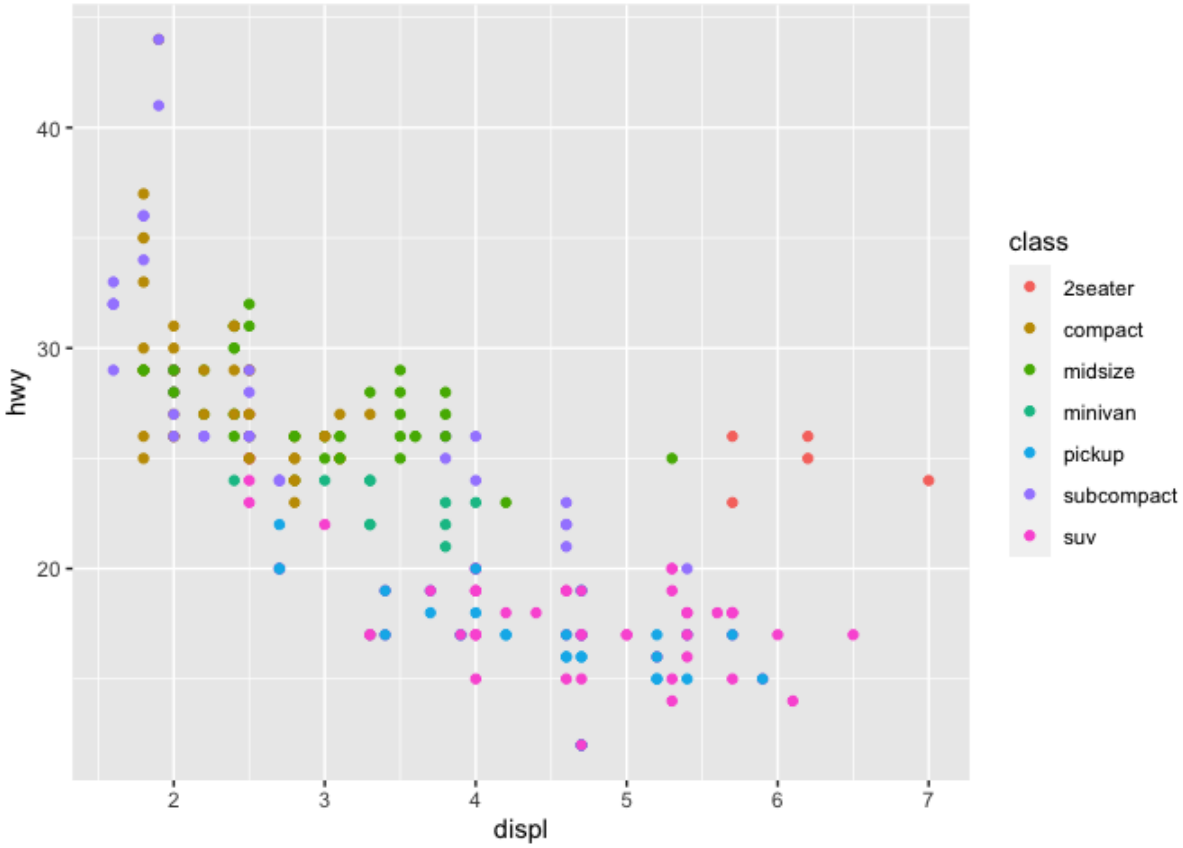
- **ggplot2**

  ggplot2 is a system for declaratively creating graphics.

```
library(ggplot2)

ggplot(mpg, aes(displ, hwy, color = class)) +
   geom_point()
```

Listing 152: ggplot2 example 1



ggplot2 also has more visualization capabilities. Here we do not introduce them one by one. However, we have provided a cheat sheet to help you look it up if you need to.

Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                  required
<GEOM_FUNCTION>(mapping = aes <MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +                 Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

Geoms Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank() and **a + expand_limits()** Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1) x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)) - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin") x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian") x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot() x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar() x, alpha, color, fill, linetype, size, weight

TWO VARIABLES both continuous

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point() x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile() x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl") x, y, alpha, color, linetype, size

e + geom_smooth(method = lm) x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col() x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot() x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center") x, y, alpha, color, fill, group

f + geom_violin(scale = "area") x, y, alpha, color, fill, group, linetype, size, weight

both discrete

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count() x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2) x, y, alpha, color, fill, shape, size

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)) x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d() x, y, alpha, color, group, linetype, size

h + geom_hex() x, y, alpha, color, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area() x, y, alpha, color, fill, linetype, size

i + geom_line() x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv") x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

j + geom_crossbar(fatten = 2) - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width
Also **geom_errorbarh()**.

j + geom_linerange() x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
                   state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map) **+ expand_limits**(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z)) x, y, z, alpha, color, group, linetype, size, weight

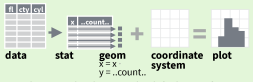
l + geom_contour_filled(aes(fill = z)) x, y, alpha, color, fill, group, linetype, size, subgroup

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE) x, y, alpha, fill

l + geom_tile(aes(fill = z)) x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

i + stat_density_2d(aes(fill = ..level..), geom = "polygon")
(geom to use; stat function; geommappings; variable created by stat)

c + stat_bin(binwidth = 1, boundary = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins = 30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se = T, level = 0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + xlim(-5, 5) **+ stat_function**(fun = dnorm, n = 20, geom = "point") **x** | ..x.., ..y..

ggplot() + stat_qq(aes(sample = 1:100))
x, y, sample | ..sample.., ..theoretical..

e + stat_sum() **x, y, size** | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun = "mean", geom = "bar")

e + stat_identity()

e + stat_unique()

Scales Override defaults with **scales** package.

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

(scale_; aesthetic to adjust; prepackaged scale to use; scale-specific arguments; range of values to include in mapping; title to use in legend/axis; labels to use in legend/axis; breaks to use in legend/axis)

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()) - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - Treat data values as dates.
scale_*_datetime() - Treat data values as date times. Same as scale_*_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.

COLOR AND FILL SCALES (DISCRETE)

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

```
o <- c + geom_dotplot(aes(fill = ..x..))
```

o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colors = topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

```
p <- e + geom_point(aes(shape = fl, size = cyl))
```

p + scale_shape() **+ scale_size()**
p + scale_shape_manual(values = c(3:7))

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

```
r <- d + geom_bar()
```

r + coord_cartesian(xlim = c(0, 5)) - xlim, ylim
The default cartesian coordinate system.

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim - Cartesian coordinates with fixed aspect ratio between x and y units.

ggplot(mpg, aes(y = fl)) **+ geom_bar()**
Flip cartesian coordinates by switching x and y aesthetic mappings.

r + coord_polar(theta = "x", direction=1)
theta, start, direction - Polar coordinates.

r + coord_trans(y = "sqrt") - x, y, xlim, ylim
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation = c(41, -74, 0)) - projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.).

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

```
s <- ggplot(mpg, aes(fl, fill = drv))
```

s + geom_bar(position = "dodge")
Arrange elements side by side.

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height.

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting.

e + geom_label(position = "nudge")
Nudge labels away from points.

s + geom_bar(position = "stack")
Stack elements on top of one another.

Each position adjustment can be recast as a function with manual **width** and **height** arguments:
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines.

r + theme_gray()
Grey background (default theme).

r + theme_dark()
Dark for contrast.

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal theme.

r + theme_void()
Empty theme.

r + theme() Customize aspects of the theme such as axis, legend, panel, and facet properties.
r + ggtitle("Title") + theme(plot.title.position = "plot")
r + theme(panel.background = element_rect(fill = "blue"))

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

```
t <- ggplot(mpg, aes(cty, hwy)) + geom_point()
```

t + facet_grid(cols = vars(fl))
Facet into columns based on fl.

t + facet_grid(rows = vars(year))
Facet into rows based on year.

t + facet_grid(rows = vars(year), cols = vars(fl))
Facet into both rows and columns.

t + facet_wrap(vars(fl))
Wrap facets into a rectangular layout.

Set **scales** to let axis limits vary across facets.

t + facet_grid(rows = vars(drv), cols = vars(fl), scales = "free")
x and y axis limits adjust to individual facets:
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet label:

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl), labeller = label_bquote(alpha ^ .(fl)))

α^{c}	α^{d}	α^{e}	α^{p}	α^{r}

Labels and Legends

Use **labs()** to label the elements of your plot.

t + labs(**x** = "New x axis label", **y** = "New y axis label", **title** ="Add a title above the plot", **subtitle** = "Add a subtitle below title", **caption** = "Add a caption below plot", **alt** = "Add alt text to the plot", <AES> = "New <AES> legend title")

t + annotate(geom = "text", x = 8, y = 9, label = "A")
Places a geom with manually selected aesthetics.

p + guides(x = guide_axis(n.dodge = 2)) Avoid crowded or overlapping labels with guide_axis(n.dodge or angle).

n + guides(fill = "none") Set legend type for each aesthetic: colorbar, legend, or none (no legend).

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right".

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred):
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points):
t + xlim(0, 100) **+ ylim**(10, 20)

t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous**(limits = c(0, 100))

- **Package**

  A package is a collection of R functions, example data, precompiled code including R programs, annotated documentation, examples, test data, etc. R language-related packages are generally stored in the "library" directory under the installation directory. By default, some commonly used packages have been brought with the R language installation. Of course, we can also customize and add some packages to be used later.

  We can use the following function to view the installation directory of R packages:

```
> .libPaths()
[1] "/Library/Frameworks/R.framework/Versions/4.0/Resources/library"
>
```

Listing 153: Package 1

  We can use the following function to view installed packages:

```
library()
```

Listing 154: Package 2

  We can use the following function to view the packages loaded by the compilation environment:

```
> search()
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:utils" "package:datasets"
[7] "package:methods" "Autoloads" "package:base"
```

Listing 155: Package 3

  To install new packages, you can use the install.packages() function in the following format:

```
install.packages("The name of the package to be installed")
```

Listing 156: Package 4

  The above is the content of all R guides. We only list some of the frequently used content. As time goes by, some of them may no longer apply, welcome to update and correct.

Programming and Data Ressources - a first step

We wanted to collect some resources for you to give you some help if you do not know where to start to look and to find some useful programming resources on the web.

First full introduction to Python:

- https://youtu.be/rfscVS0vtbw (freeCodeCamp)

- https://youtu.be/kqtD5dpn9C8 (1 hour course intro in Python)
- https://youtu.be/8mAITcNt710 (Harvard CS50 – Full Computer Science University Course, includes partly C, Python, HTML, SQL,... generally for the computer science mindset)

However, the most useful thing to do when you are stuck with an error is to look on the internet on websites such as stackoverflow, github, reddit and many more, where probably your question has been asked already (at least the questions when you start programming).

5 OPEN DATA

Also we provide you with some websites for data sets, just to give you a start.
https://data.worldbank.org/ (World Bank Data)
https://www.census.gov/data.html (US Census Data, general source)
https://www.kaggle.com/datasets (great variety of data sets)
https://archive.ics.uci.edu/ml/index.php (Machine Learning Data)
https://trends.google.com/trends/explore (General trends, almost every topic)
https://healthdata.gov/ (Health Data)
https://www.ncei.noaa.gov/products (large provider of climate and weather information)
https://www.earthdata.nasa.gov/?_fsi=BqJ6IiI5 (NASA)
https://registry.opendata.aws/ (Amazon Web Service, Public Datasets)
https://www.aeaweb.org/resources/data (Economic Data, mainly US)
https://github.com/awesomedata/awesome-public-datasets (all topics)

Furthermore, every country (at least in Europe and North America) has a Federal Statistical Office, where you can get access to many kinds of data and services. Here is the Swiss one: https://www.bfs.admin.ch/bfs/en/home/services/ogd/portal.html

6 EMBEDDED DATA SETS (IN R AND PYTHON)

For R:
Sometimes the data sets are embedded in a package, so you have to install the package first.

```
install.packages("name_of_package") #do not forget the strings

data(“name_of_data”)  #to load the data into your environment
```

e.g.

```
data(“mtcars”) #Loading

head(mtcars, 6) #Print the first 6 rows
```

In the package *datasets*, you find already a big selection of available in-built data sets: https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html

Some very popular data sets in R that you will encounter in your studies with probability = 1:

- mtcars Motor Trend Car Road Tests
- Iris Edgar Anderson's Iris Data
- Titanic Survival of passengers on the Titanic
- JohnsonJohnson Quarterly Earnings per Johnson Johnson Share

For Python:

A curated collection of datasets for data analysis machine learning is provided by opendatasets. *Opendatasets* is a Python library for downloading datasets from online sources like Kaggle and Google Drive using a simple Python command.

https://pypi.org/project/opendatasets/ (website that gives information on how to install and import data e.g. from Kaggle and some further links for data sets)

Part III. Mathematics

Linear Algebra

7 OPERATIONS WITH VECTORS AND MATRICES

Given two matrices $\mathbf{A} \in \mathbb{R}^{p \times q}$ and $\mathbf{B} \in \mathbb{R}^{r \times s}$, the following operations can be defined.

- **Sum.** If $p = r$ and $q = s$, $\mathbf{A} + \mathbf{B} = \mathbf{C} \in \mathbb{R}^{p \times q}$, where each element of $\mathbf{C}$ is the sum of the corresponding elements in $\mathbf{A}$ and $\mathbf{B}$,

$$c_{ij} = a_{ij} + b_{ij}.$$

- **Transpose.** The transpose of $\mathbf{A} \in \mathbb{R}^{p \times q}$ is the matrix $\mathbf{A}^\top \in \mathbb{R}^{q \times p}$, where the i-th column of $\mathbf{A}$ corresponds to the i-th row of $\mathbf{A}^\top$, i.e. $(\mathbf{A}^\top)_{ji} = a_{ij}$

- **Scalar Multiplication.** Given a scalar α, the matrix $\alpha\mathbf{A}$ is obtained by multiplying each entry of $\mathbf{A}$ by α, i.e. $(\alpha\mathbf{A})_{ij} = \alpha \cdot a_{ij}$.

- **Matrix Multiplication.** The matrix multiplication $\mathbf{AB}$ is possible only if $q = r$. $\mathbf{AB} = \mathbf{C} \in \mathbb{R}^{p \times s}$, where $c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}$.

- **Power to an integer.** If $p = q$ and $n \in \mathbb{N}$, the n-th power of the matrix $\mathbf{A}$ is $\mathbf{A}^n = \mathbf{A} \times \cdots \times \mathbf{A}$, where the matrix $\mathbf{A}$ is multiplied by itself n times.

- **Kronecker product.** The Kronecker product of two matrices $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{(pr) \times (qs)}$ and is such that

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1m}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{n1}\mathbf{B} & \dots & \dots & a_{nm}\mathbf{B} \end{bmatrix}.$$

- **Inverse.** The inverse of the matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{-1}$ and is such that

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I},$$

  where $\mathbf{I}$ is the identity matrix. Only if $\mathbf{A}$ is quadratic, the inverse exists, but a square matrix does not necessarily has an inverse.

Note. *Note that these operations apply also for vectors, considering that a column vector $v \in \mathbb{R}^p$ corresponds to a matrix of dimension $p \times 1$.*

8 DETERMINANT AND TRACE

8.1 *Definitions*

The **determinant** of a squared matrix $\mathbf{A} \in \mathbb{R}^{p \times p}$ is a function which associates the matrix A to a scalar. It is normally denoted as $\det(\cdot)$.

The **trace** of a matrix $\mathbf{A} \in \mathbb{R}^{p \times p}$ is defined as $\mathrm{Tr}(\mathbf{A}) = \sum_{1 \leq i \leq p} a_{ii}$.

8.2 *Computation of the determinant*

- If $p = 1$, $\det(\mathbf{A}) = \mathbf{A}$;
- if $p = 2$, $\det(\mathbf{A}) = a_{11}a_{22} - a_{12}a_{21}$;
- if $p > 2$, the Leibniz formula for the determinant is applied.

8.3 *Properties*

The determinant has the following properties.

- $\det(\mathbf{A}) = \det(\mathbf{A}^\top)$,
- $\det(\mathbf{A}^{-1}) = 1/\det(\mathbf{A})$ if $\mathbf{A}$ is invertible,
- $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$,
- $\det(\alpha\mathbf{A}) = \alpha \det(\mathbf{A})$,
- $\det(\mathbf{A}) = \prod_{i=1}^{p} \lambda_i$, where λ_i is the i-th eigenvalue of the matrix $\mathbf{A}$ (see Section 9).

The trace has the following properties.

- $\mathrm{Tr}(\mathbf{A} + \mathbf{B}) = \mathrm{Tr}(\mathbf{A}) + \mathrm{Tr}(\mathbf{B})$,
- $\mathrm{Tr}(\mathbf{AB}) = \mathrm{Tr}(\mathbf{BA})$,
- $\mathrm{Tr}(\mathbf{B}^{-1}\mathbf{AB}) = \mathrm{Tr}(\mathbf{A})$,
- $\mathrm{Tr}(\mathbf{A}) = \sum_{i=1}^{p} \lambda_i$, where λ_i is the i-th eigenvalue of the matrix $\mathbf{A}$ (see Section 9).

Note. $\mathrm{Tr}(\mathbf{ABC}) = \mathrm{Tr}(\mathbf{CAA})$, but $\mathrm{Tr}(\mathbf{ABC}) \neq \mathrm{Tr}(\mathbf{BAC})$.

9 EIGENVECTORS AND EIGENVALUES

9.1 *Definition*

Let $\mathbf{A}$ be a $p \times p$ matrix and let $\lambda \in \mathbb{R}$. If λ is such that $\mathbf{v} \in \mathbb{R}^p$, $v \neq \mathbf{0}$ and

$$\mathbf{Av} = \lambda \mathbf{v},$$

then λ is called an eigenvalue of $\mathbf{A}$ and $\mathbf{v}$ is called an eigenvector of $\mathbf{A}$ associated to the eigenvalue λ.

9.2 *Quadratic forms and definite matrices*

Let $\mathbf{A} \in \mathbb{R}^{p \times p}$ be a symmetric matrix and $\mathbf{x} \in \mathbb{R}^p$, the **quadratic form** of $\mathbf{A}$ is defined as $\mathbf{x}^\top \mathbf{Ax}$.
Let λ_i be the i-th eigenvalue of $\mathbf{A}$, then

- if $\mathbf{x}^\top \mathbf{Ax} > 0 \ \forall \mathbf{x} \in \mathbb{R}^p$, $\mathbf{A}$ is called *definite positive*, and $\lambda_i > 0 \ \forall i = 1, \ldots, p$;
- if $\mathbf{x}^\top \mathbf{Ax} \geq 0 \ \forall \mathbf{x} \in \mathbb{R}^p$, $\mathbf{A}$ is called *semi-definite positive*, and $\lambda_i \geq 0 \ \forall i = 1, \ldots, p$;
- if $\mathbf{x}^\top \mathbf{Ax} < 0 \ \forall \mathbf{x} \in \mathbb{R}^p$, $\mathbf{A}$ is called *definite negative*, and $\lambda_i < 0 \ \forall i = 1, \ldots, p$;
- if $\mathbf{x}^\top \mathbf{Ax} \leq 0 \ \forall \mathbf{x} \in \mathbb{R}^p$, $\mathbf{A}$ is called *semi-definite negative*, and $\lambda_i \leq 0 \ \forall i = 1, \ldots, p$.

9.3 *Theorem: Spectral decomposition*

Theorem 9.1. *Let* $\mathbf{A}$ *be a real symmetric matrix. Then,* $\mathbf{A}$ *can be expressed as*

$$\mathbf{A} = \mathbf{E}\Lambda\mathbf{E}^\top = \sum_{i=1}^{p} \lambda_i \mathbf{e}_i \mathbf{e}_i^\top,$$

where

- $\mathbf{E}$ *is a* $p \times p$ *matrix and the i-th column of* $\mathbf{E}$, $\mathbf{e}_i$, *is the i-th eigenvector of* $\mathbf{A}$,
- Λ *is a* $p \times p$ *diagonal matrix and each diagonal element* λ_i *corresponds to the i-th eigenvalue of* $\mathbf{A}$ *(with possibly identical eigenvalues).*

9.4 *Theorem: Singular values decomposition*

Theorem 9.2. *Let* $\mathbf{A}$ *be a* $n \times p$ *matrix, then* $\mathbf{A}$ *can be expressed as*

$$\mathbf{A} = \mathbf{UDV}^\top,$$

where

- $\mathbf{U}$ *is a* $n \times n$ *matrix such that* $\mathbf{U}^\top \mathbf{U} = \mathbf{I}_n$;
- $\mathbf{D}$ *is a diagonal* $n \times p$ *matrix and the i-th diagonal element is called the i-th singular value of* $\mathbf{A}$;
- $\mathbf{V}$ *is a* $p \times p$ *matrix such that* $\mathbf{V}^\top \mathbf{V} = \mathbf{I}_p$.

9.5 *Differential calculus*

In this section, we report the main techniques to compute the derivatives taken with respect to a vector $\mathbf{x} \in \mathbb{R}^p$. The notation $\mathbf{x}'$ indicates the transpose of the vector $\mathbf{x}$.

Consider the scalar function $y = f\left(x_1, x_2, \ldots, x_p\right) = f(\mathbf{x})$. The **gradient** ∇_y, or vector of first derivatives, is defined as

$$\nabla_y = \partial y / \partial \mathbf{x} = \begin{bmatrix} \partial y / \partial x_1 \\ \partial y / \partial x_2 \\ \vdots \\ \partial y / \partial x_n \end{bmatrix}$$

The matrix of second derivatives is called the **Hessian matrix**, H_y. It is defined as

$$\begin{aligned} H_y = \frac{\partial^2 y}{\partial \mathbf{x} \partial \mathbf{x}'} &= \frac{\partial \left(\partial y_i / \partial \mathbf{x}\right)}{\partial \mathbf{x}'} \\ &= \begin{bmatrix} \frac{\partial^2 y}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 y}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 y}{\partial x_n \partial x_n}. \end{bmatrix} \end{aligned}$$

Consider a vector $\mathbf{y} \in \mathbb{R}^m$. The derivative of $\mathbf{y}$ with respect to $\mathbf{x}'$ is a matrix of dimension $m \times p$, called the **Jacobian matrix** of $\mathbf{y}$ with respect to $\mathbf{x}'$.

$$\begin{aligned} \partial \mathbf{y} / \partial \mathbf{x}' &= \left[\partial y_i / \partial \mathbf{x}_j\right]_{ij} \\ &= \begin{bmatrix} \partial y_1 / \partial x_1 & \ldots & \partial y_1 / \partial x_n \\ \vdots & \ddots & \vdots \\ \partial y_m / \partial x_1 & \ldots & \partial y_m / \partial x_n \end{bmatrix} = \left[\partial \mathbf{y}' / \partial \mathbf{x}\right]' \end{aligned}$$

Let's now focus on linear functions.

Let $\mathbf{c}$ be a vector in $\mathbb{R}^p$ and $z = \mathbf{c}'\mathbf{x}$, then

$$\partial z / \partial \mathbf{x} = \partial \mathbf{c}'\mathbf{x} / \partial \mathbf{x} = \partial \mathbf{x}'\mathbf{c} / \partial \mathbf{x} = \begin{bmatrix} \partial z / \partial x_1 \\ \vdots \\ \partial z / \partial x_p \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_p \end{bmatrix} = \mathbf{c}.$$

Let A be a matrix in $\mathbb{R}^{p \times p}$ and $\mathbf{z} = \mathrm{A}'\mathbf{x}$, then

$$\partial \mathbf{z}' / \partial \mathbf{x} = \partial \mathbf{x}' \mathrm{A} / \partial \mathbf{x} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_p \end{pmatrix} = \mathrm{A},$$

where $\mathbf{a}_j$ is the j^{th} column of A.

Let's now focus on the quadratic function $z = \mathbf{x}'\mathrm{A}\mathbf{x}$. Then

$$\begin{aligned} \frac{\partial z}{\partial \mathbf{x}} &= \frac{\partial \mathbf{x}'\mathrm{A}\mathbf{x}}{\partial \mathbf{x}} = \mathrm{A}'\mathbf{x} + \mathbf{A}\mathbf{x} = \left(\mathbf{A}' + \mathbf{A}\right)\mathbf{x}, \\ \frac{\partial^2 z}{\partial \mathbf{x} \partial \mathbf{x}'} &= \frac{\partial^2 \left(\mathbf{x}'\mathbf{A}\mathbf{x}\right)}{\partial \mathbf{x} \partial \mathbf{x}'} = \mathbf{A} + \mathbf{A}'. \end{aligned}$$

Note. *If* A *is symmetric,* $\dfrac{\partial \mathbf{x}'\mathrm{A}\mathbf{x}}{\partial \mathbf{x}} = 2\mathrm{A}\mathbf{x}$ *and* $\dfrac{\partial^2 \mathbf{x}'\mathbf{A}\mathbf{x}}{\partial \mathbf{x}\partial \mathbf{x}'} = 2\mathrm{A}$.

Furthermore,

$$
\begin{aligned}
&\partial \mathbf{x}'\mathrm{B}\mathbf{y}/\partial \mathrm{B} = \mathbf{x}\mathbf{y}' \\
&\partial \operatorname{tr}(\mathrm{A})/\partial \mathrm{A} = \mathrm{I} \\
&\partial |\mathrm{A}|/\partial \mathbf{A} = |\mathbf{A}| \left(\mathbf{A}'\right)^{-1} \\
&\partial \ln |\mathbf{A}|/\partial \mathrm{A} = \left(\mathrm{A}'\right)^{-1} \\
&\partial \mathrm{AB}/\partial \mathbf{x} = \mathrm{A}(\partial \mathrm{B}/\partial \mathbf{x}) + (\partial \mathrm{A}/\partial x)\mathrm{B} \\
&\partial \mathrm{A}^{-1}/\partial x = -\mathrm{A}^{-1}(\partial \mathrm{A}/\partial \mathbf{x})\mathrm{A}^{-1} \\
&\partial \operatorname{tr}\left(\mathrm{A}'\mathrm{A}\right)/\partial \mathrm{A} = \operatorname{tr}\left(\mathrm{A}\mathrm{A}'\right)/\partial \mathrm{A} = 2\mathrm{A} \\
&\partial \operatorname{tr}\left(\mathrm{A}'\mathrm{X}\mathrm{A}\right)/\partial \mathrm{A} = \left(\mathrm{X} + \mathrm{X}'\right)\mathrm{A}, \quad \mathrm{X} \text{ symmetric } \Rightarrow \partial \operatorname{tr}\left(\mathrm{A}'\mathrm{X}\mathrm{A}\right)/\partial \mathrm{A} = 2\mathrm{X}\mathrm{A} \\
&\partial \operatorname{tr}\left(\mathrm{A}\mathrm{X}\mathrm{A}'\right)/\partial \mathrm{A} = \mathrm{A}\left(\mathrm{X} + \mathrm{X}'\right), \quad \mathrm{A} \text{ symmetric } \Rightarrow \partial \operatorname{tr}\left(\mathrm{A}\mathrm{X}\mathrm{A}'\right)/\partial \mathrm{A} = 2\mathrm{A}\mathrm{X} \\
&\partial \operatorname{tr}(\mathrm{X}\mathrm{A}\mathrm{B})/\partial \mathrm{A} = \mathrm{X}'\mathrm{B}', \quad \text{with } \mathrm{A} \in \mathbb{R}^{M\times N}, \mathrm{X} \in \mathbb{R}^{P\times M} \text{ and } \mathrm{B} \in \mathbb{R}^{N\times P} \\
&\partial \operatorname{tr}\left(\mathrm{X}\mathrm{A}'\mathrm{B}\right)/\partial \mathrm{A} = \mathrm{B}\mathrm{X}, \quad \text{with } \mathrm{A} \in \mathbb{R}^{M\times N}, \mathrm{X} \in \mathbb{R}^{P\times N} \text{ and } \mathrm{B} \in \mathbb{R}^{M\times P} \\
&\partial \operatorname{tr}\left(\mathrm{X}\mathrm{A}\mathrm{A}'\mathrm{X}'\right)/\partial \mathrm{A} = 2\mathrm{X}'\mathrm{X}\mathrm{A} \\
&\partial \operatorname{vec}(\mathrm{A}\mathrm{X})/\partial \mathrm{A} = (\mathrm{I}_T \otimes \operatorname{vec} \mathrm{I}_M)\,\mathrm{X}', \quad \text{with } \mathrm{A} \in \mathbb{R}^{M\times N} \text{ and } \mathrm{X} \in \mathbb{R}^{N\times T} \\
&\partial \operatorname{vec}(\mathrm{X}\mathrm{A})/\partial \mathrm{A} = \mathrm{I}_N \otimes \operatorname{vec} \mathrm{X}', \quad \text{with } \mathrm{A} \in \mathbb{R}^{M\times N} \text{ and } \mathrm{X} \in \mathbb{R}^{K\times M}.
\end{aligned}
$$

10 SEQUENCES AND SERIES

10.1 *Geometric series*

Intuition. A geometric sequence is a sequence that is formed by multiplying the previous element by a ratio.

For $r \neq 1$, the *finite geometric series* is defined as:

$$\sum_{j=0}^{M} r^j = \frac{r^{M+1} - 1}{r - 1};$$

if $|r| < 1$, the *infinite geometric series* is decreasing and converges to

$$\sum_{j=0}^{\infty} r^j = \frac{1}{1 - r}.$$

11 EULER'S NUMBER

Euler's number, better known as the number e, is approximately equal to 2.71828 and can be characterized in different ways, such as

1. Limit. $\lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n = e$;

2. Maclaurin Series. $\sum_{n=0}^{\infty} \frac{1}{n!} = e$;

The **Euler's formula** states that

$$e^{i\theta} = \cos\theta + i\sin\theta.$$

12 INTEGRALS

In this section the most common antiderivatives are recalled, as well as main techniques of integration: integration by parts and integration by substitution.

Intuitively, the definite integral of a scalar function represents the area under its curve between two points.

Definition of integral

If you need a refresher on the definition of Riemann's integral, definite integral and indefinite integral, you can check out Mathematics I, Section 2.

12.1 *Antiderivatives of common functions*

Table 1 is a sum up of common antiderivatives. Note that $k \in \mathbb{N}$, while $a, b \in \mathbb{R}\backslash\{0\}$.

12.2 *Integration by parts*

The formula of integration by parts is

$$\int f(x) \cdot g(x)dx = F(x)g(x) - \int F(x)g'(x)dx, \tag{1}$$

where $F(x)$ is the antiderivative of $f(x)$.

Example 12.1. Consider the integral $\int \log(ax)dx$.

To solve this integral, one can apply the integration by parts method and notice that it can be rewritten as

$$\int 1 \cdot \log(ax)dx.$$

Considering the notation the formula 1, in this specific example we have $f(x) = 1$ and $g(x) = \log(ax)$. Hence, $F(x) = x$ and $g'(x) = a\frac{1}{ax} = \frac{1}{x}$.

By following the formula 1, we obtain

$$\begin{aligned}\int 1 \cdot log(ax)dx &= x\log(ax) - \int x\frac{1}{x}dx \\ &= x\log(ax) - \int 1dx = \\ &= x\log(ax) - x + c.\end{aligned}$$

Table 1: Most common Antiderivatives

Integral	Antiderivative	Conditions
$\int x^k dx$	$\frac{1}{k+1}x^{k+1} + c$	$k \neq -1$
$\int (f(x))^k f'(x) dx$	$\frac{1}{k+1}(f(x))^{k+1} + c$	$k \neq -1$
$\int \frac{1}{x} dx$	$\ln \lvert x \rvert + c$	
$\int \frac{1}{f(x)} f'(x) dx$	$\ln \lvert f(x) \rvert + c$	
$\int e^{ax} dx$	$\frac{1}{a}e^{ax} + c$	
$\int e^{f(x)} f'(x) dx$	$e^{f(x)} + c$	
$\int a^x dx$	$\frac{a^x}{\ln a} + c$	
$\int \cos(ax) dx$	$\frac{1}{a}\sin(ax) + c$	
$\int \sin(ax) dx$	$-\frac{1}{a}\cos(ax) + c$	
$\int \frac{1}{\cos^2(ax)} dx$	$\frac{1}{a}\tan(ax) + c$	
$\int \frac{1}{\sin^2(ax)} dx$	$-\frac{1}{a\tan(ax)} + c$	
$\int \frac{1}{ax^2+b} dx$	$-\frac{1}{\sqrt{ab}}\arctan\left(\frac{\sqrt{a}}{\sqrt{b}}x\right) + c$	
$\int \frac{1}{\sqrt{b-ax^2}} dx$	$\frac{1}{\sqrt{a}}\arcsin\left(\frac{\sqrt{a}}{\sqrt{b}}x\right) + c$	

12.3 *Integration by substitution*

The formula of integration by substitution comes from the chain rule, and is the following

$$\int_a^b f(g(x))g'(x)dx = \int_{g(a)}^{g(b)} f(u)du. \tag{2}$$

Example 12.2. Let's solve the integral $\int_a^b \left(\frac{x-l}{s}\right)^2 dx$.

This integral can be solved by substitution and defying

$$u = g(x) = \frac{x-l}{s},$$

from which we can compute

- $du = d\left(\frac{x-l}{s}\right) = dx/s \to dx = s\, du$,
- $g(a) = \frac{a-l}{s}$,
- $g(b) = \frac{b-l}{s}$.

Substituting these quantities in the original integral leads to

$$\begin{aligned}\int_a^b \left(\frac{x-l}{s}\right)^2 dx &= \int_{\frac{a-l}{s}}^{\frac{b-l}{s}} (u^2)s\, du = \\ &= s \int_{\frac{a-l}{s}}^{\frac{b-l}{s}} u^2 du = \\ &= s\left[\frac{u^3}{3}\right]_{\frac{a-l}{s}}^{\frac{b-l}{s}} = \\ &= \frac{s}{2}\left[\left(\frac{b-l}{s}\right)^3 - \left(\frac{a-l}{s}\right)^3\right].\end{aligned}$$

Help with checking calculus and plotting

To solve specific math problems or to find information on mathematical subjects and topics e.g. arithmetic, algebra, calculus, differential equations, statistics, Wolfram Alpha can be useful:
https://www.wolframalpha.com/
There is a lot to play around to visualise and plot equations, solve equations.

For statistics, Wolfram Alpha is able to compute all manner of descriptive and inferential statistical properties and to produce regression analyses and equation fitting. However, we think for learning new concepts and hand-written practice, you should rely on the materials given in class. Additionally, you can use R or Python to check your results or have an additional understanding.

To have a visual representation or intuition, you can also check GeoGebra https://www.geogebra.org/.

Specifically helpful for checking derivatives and integral computations:
https://www.integralrechner.de/
https://www.ableitungsrechner.net/

Part IV.
Probability

In this chapter, the main focus will be presenting a general and basic introduction to probabilities, especially some that will be used throughout the master courses. The goal is to present the main components of probability theory as well as useful properties for the master. However, it will not be as explanatory as a general class on the subject. This is why we advice you to either follow probability 1 or probability 2 given your own probability's knowledge.

13 SET THEORY

Let us start with sets of elements **A**, **B** and **C** belonging to the sample space **S**. It can also be called Omega, written Ω.

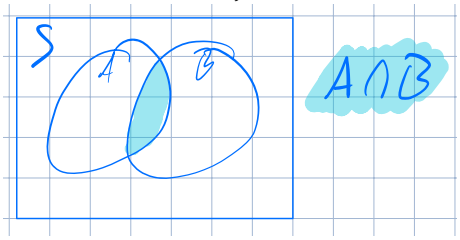
13.1 *Subset*

We say that **A** belongs to **B** if $\forall x \mid x \subset \mathbf{A} \Rightarrow x \subset \mathbf{B}$.
It is noted $\mathbf{A} \subset \mathbf{B}$.

- If $\mathbf{A} \subset \mathbf{B}$ and $\mathbf{B} \subset \mathbf{A}$, then $\mathbf{A} = \mathbf{B}$.
- If $\mathbf{A} \subset \mathbf{B}$ and $\mathbf{B} \subset \mathbf{C}$, then $\mathbf{A} \subset \mathbf{C}$.

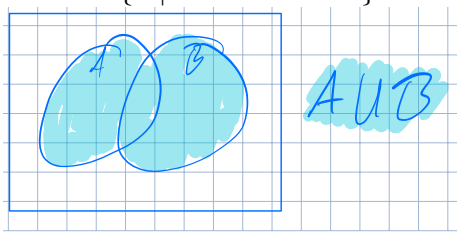
13.2 *Intersection*

We write the intersection between two sets:
$\mathbf{A} \cap \mathbf{B} = \{x \mid x \in \mathbf{A} \,\&\, x \in \mathbf{B}\}$.
If **A** and **B** are disjoint, then $\mathbf{A} \cap \mathbf{B} = \emptyset$.



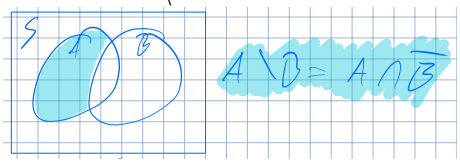
13.3 *Union*

We write the union between two sets:
$\mathbf{A} \cup \mathbf{B} = \{x \mid x \in \mathbf{A} \, or \, x \in \mathbf{B}\}$.



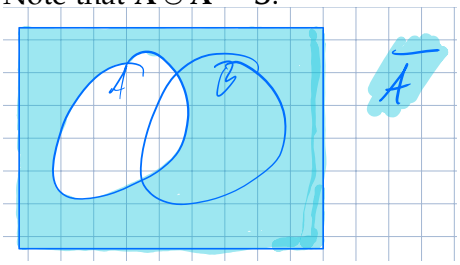
13.4 *Difference*

We write the difference between two sets:
$\mathbf{A} \backslash \mathbf{B} = \{x \mid x \in \mathbf{A} \,\&\, x \notin \mathbf{B}\}$.
Note that $\mathbf{A} \backslash \mathbf{B} = \mathbf{A} \cap \bar{\mathbf{B}}$.



13.5 *Complement*

We say that a set $\bar{\mathbf{A}}$ is the complement of $\mathbf{A}$. It is defined as follow: $\bar{\mathbf{A}} = \{x \mid x \in \mathbf{S} \,\&\, x \notin \mathbf{A}\}$.
Note that $\mathbf{A} \cup \bar{\mathbf{A}} = \mathbf{S}$.



13.6 *Properties*

- Commutative Laws
  $\mathbf{A} \cup \mathbf{B} = \mathbf{B} \cup \mathbf{A}$,
  $\mathbf{A} \cap \mathbf{B} = \mathbf{B} \cap \mathbf{A}$.
- Associative Laws
  $(\mathbf{A} \cup \mathbf{B}) \cup \mathbf{C} = \mathbf{A} \cup (\mathbf{B} \cup \mathbf{C})$,
  $(\mathbf{A} \cap \mathbf{B}) \cup \mathbf{C} = \mathbf{A} \cap (\mathbf{B} \cup \mathbf{C})$.

- Distributive laws
  $\mathbf{A} \cup (\mathbf{B} \cap \mathbf{C}) = (\mathbf{A} \cup \mathbf{B}) \cap (\mathbf{A} \cup \mathbf{C})$,
  $\mathbf{A} \cap (\mathbf{B} \cup \mathbf{C}) = (\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C})$.
- Morgan's Laws
  $\overline{\mathbf{A} \cup \mathbf{B}} = \bar{\mathbf{A}} \cap \bar{\mathbf{B}}$,
  $\overline{\mathbf{A} \cap \mathbf{B}} = \bar{\mathbf{A}} \cup \bar{\mathbf{B}}$.

Set Theory

If you need a refresher on this topic, you can check out Probability I lectures, Lecture 1 and Probability Theory: a concise course. [8]

14 COMBINATORICS

Combinatorics is the field of mathematics which is concerned by counting on finite subset. This can be particularly useful when doing basic probability.

14.1 *Factorial*

The factorial is the product of all natural numbers until a given n. It is noted: $n! = 1 * 2 * ... * (n-1) * n = \prod_{i=1}^{n} i$.
Note that $n! = n * (n-1)!$ and $0! = 1$.

14.2 *Binomial Coefficient*

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$, $n \,\&\, k \in \mathbb{N}$ and $0 \leq k \leq n$.

- $\binom{n}{0} = \binom{n}{n} = 1$,
- $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$,
- $\binom{n}{n-k} = \binom{n}{k}$

15 ENUMERATION

This section will explain how to count elements on a set by different way.

15.1 *simple arrangement*

Among n distinct elements, we choose k distinct elements. By classifying them in a particular order, we obtain an arrangement simple:
$A_k^n = \frac{n!}{(n-k)!}$.

15.2 *Arrangement with repetition*

Among n distinct elements, we choose k distinct or not elements. By classifying them in a particular order, we obtain an arrangement avec répétiton:
$\overline{A_k^n} = n^k$.

15.3 *simple permutation*

By classifying in a particular order n distinct elements, we obtain a simple permutation:
$P_n = n!$

15.4 *permutation with repetition*

By classifying in a particular order n elements of which n_1 are of type 1, n_2 of type 2,..., n_p of type p, we obtain a permutation with repetition:
$\overline{P}_{(n_1,\ldots,n_p)} = \frac{n!}{\prod_{i=1}^{p} n_i!}$.
Note that $\sum_{i=1}^{p} n_i = n$

15.5 *Simple combination*

Among n distinct elements, we choose k distinct elements without ordering them in any order:
$C_k^n = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!}$.

15.6 *combination with repetition*

Among n distinct elements, we choose k distinct or not elements without ordering them in any order:
$\overline{C_k^n} = \left(\begin{array}{c} n+k-1 \\ k \end{array} \right) = \frac{(n+k-1)!}{k!(n-1)!}$.

Combinatorics and Enumeration

If you need a refresher on this topic, you can check out the Formulaires et Tables CRM [3]. This book is available at the Uni mail Library.

16 SIMPLE PROBABILITIES

Let us start with some basic notion as already defined in the Set Theory section 13. Define Ω the universe; set of all possible outcomes associated with a given random event. Define **A**, **B** and **C**, which are different events belonging to **U**. The notation $P(\mathbf{A}) = p$ defines the probability of observing the event **A** with associated probability p.
You can find more information in the following book: A First Course in Probability. [7]

- $P(\mathbf{U}) = 1$: The probability of the universe is certain,
- $P(\emptyset) = 0$: The probability of the empty set is null,
- $0 \leq P(\mathbf{A}) \leq 1$: The probability of an event is always between $[0; 1]$,
- $P(\bar{\mathbf{A}}) = 1 - P(\mathbf{A})$,
- If $\mathbf{A} \subset \mathbf{B}$, then $P(\mathbf{A}) \leq P(\mathbf{B})$,
- $P(\mathbf{A} \cup \mathbf{B}) = P(\mathbf{A}) + P(\mathbf{B}) - P(\mathbf{A} \cap \mathbf{B})$: The probability of the union is the sum of the probabilities minus the probability of the intersection,
- If **A** and **B** are disjoints, then $P(\mathbf{A} \cup \mathbf{B}) = P(\mathbf{A}) + P(\mathbf{B})$,
- $P(\bar{\mathbf{A}} \cup \bar{\mathbf{B}}) = 1 - P(\mathbf{A} \cap \mathbf{B})$,
- $P(\bar{\mathbf{A}} \cap \bar{\mathbf{B}}) = 1 - P(\mathbf{A} \cup \mathbf{B})$,
- $P(\mathbf{A} \cap \bar{\mathbf{B}}) = P(\mathbf{A}) - P(\mathbf{A} \cap \mathbf{B})$.

16.1 *Conditional Probabilities*

We denote $P(\mathbf{B}|\mathbf{A})$ the probability of the event B given the event A, if $P(\mathbf{A}) \neq 0$

$P(\mathbf{B}|\mathbf{A}) = \frac{P(\mathbf{A} \cap \mathbf{B})}{P(\mathbf{A})}$.

Furthermore, we can write the probability of the intersection of two events as follow:
$\Rightarrow P(\mathbf{A} \cap \mathbf{B}) = P(\mathbf{A}) \cdot P(\mathbf{B}|\mathbf{A}) = P(\mathbf{B}) \cdot P(\mathbf{A}|\mathbf{B})$

16.2 *Independent Events*

If $P(\mathbf{A} \cap \mathbf{B}) = P(\mathbf{A}) \cdot P(\mathbf{B})$, then $\mathbf{A}$ and $\mathbf{B}$ are called independent events.
Furthermore, conditioning two independent events yields the unconditional probability.

$\Rightarrow P(\mathbf{B}|\mathbf{A}) = \frac{P(\mathbf{A} \cap \mathbf{B})}{P(\mathbf{A})} = \frac{P(\mathbf{A}) \cdot P(\mathbf{B})}{P(\mathbf{A})} = P(\mathbf{B})$.

16.3 *Theorem of total probabilities*

If $\bigcup_{i=1}^{n} \mathbf{B}_i = \mathbf{U}$, $\forall i, j \quad \mathbf{B}_i \cap \mathbf{B}_j = \emptyset$ and $\forall i \quad P(\mathbf{B}_i) \neq 0$, then
$P(\mathbf{A}) = P(\mathbf{A}|\mathbf{B}_1) \cdot P(\mathbf{B}_1) + \ldots + P(\mathbf{A}|\mathbf{B}_n) \cdot P(\mathbf{B}_n) = \sum_{i=1}^{n} P(\mathbf{A}|\mathbf{B}_i) \cdot P(\mathbf{B}_i)$.

16.4 *Bayes' Theorem*

If $\bigcup_{i=1}^{n} \mathbf{B}_i = \mathbf{U}$, $\forall i, j \quad \mathbf{B}_i \cap \mathbf{B}_j = \emptyset$ and $\forall i \quad P(\mathbf{B}_i) \neq 0$, then
$P(\mathbf{B}_i|\mathbf{A}) = \frac{P(\mathbf{A}|\mathbf{B}_i) \cdot P(\mathbf{B}_i)}{\sum_{j=1}^{n} P(\mathbf{A}|\mathbf{B}_j) \cdot P(\mathbf{B}_j)}$, if $P(\mathbf{A}) \neq 0$

Combinatorics and Enumeration

If you need a refresher on this topic, you can check out Probability I lectures, Lecture 2.

17 RANDOM VARIABLE

To start this chapter, let us define a random variable X as a function that can takes different outcomes/realizations $x_1, x_2, \ldots$ with different probabilities associated $p_1, p_2, \ldots$. Such function can be called discrete random variable if the set of different outcomes is finite or countable, or continuous if the set is infinite/uncountable.
Example 1: Discrete Random Variable
For example, if we roll a dice the set of all possible outcomes is as followed: $\{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5, x_6 = 6\}$. The associated probabilities are all the same: $\{p = \frac{1}{6}\}$. As we have a finite number of outcomes, this function is called discrete random variables.

Example 2: Continuous Random Variable
On the other hand, if we define a function X being the waiting time in minutes before being served at the restaurant. Assuming it won't take the server longer than 75 minutes to serve, the set of all possible outcomes is $]0; 75] = \{x : 0 < x \leq 75\}$. As there is an uncountable number of outcomes for this function, we say that it follows a continuous random variable.
Furthermore, the exact probability of observing one outcome in a continuous random variable

is always 0: $P(X = x) = 0$. We rather compute the probability that the server will arrive before a certain time: $P(X \leq x)$

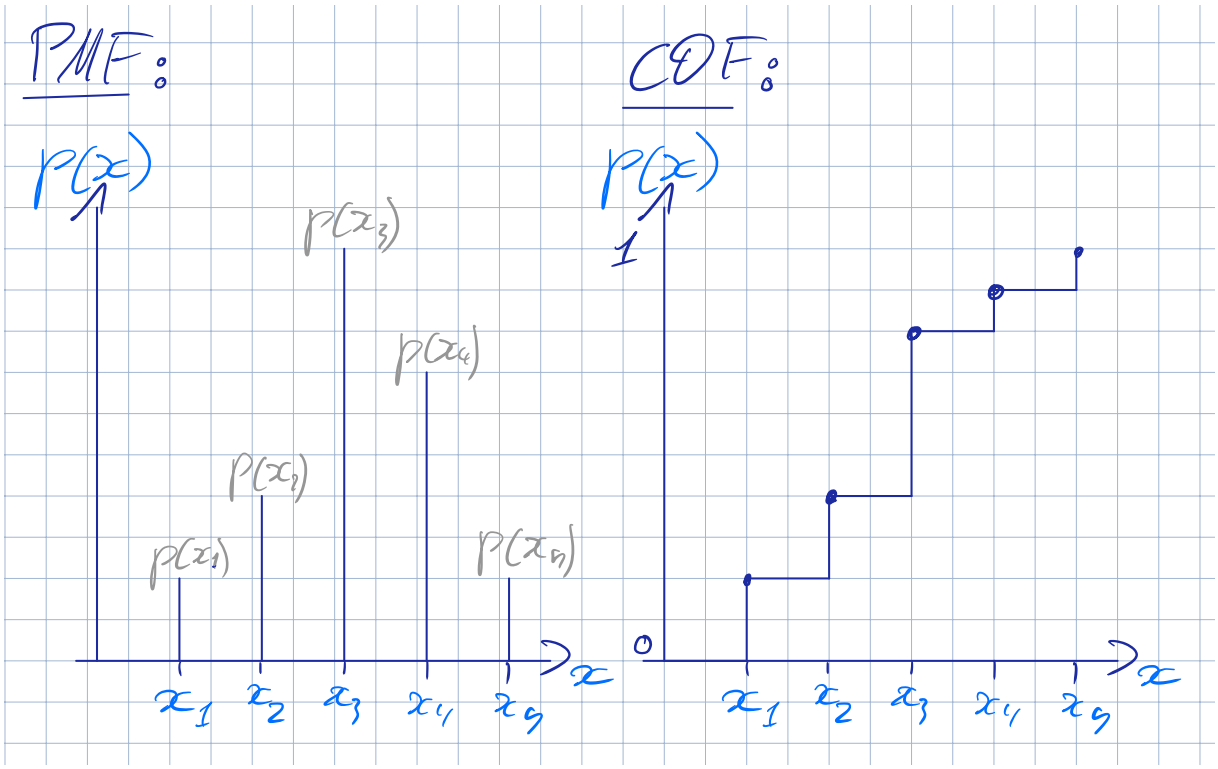
To formalize these concepts, we say that to characterize a random variable, one needs a cumulative distribution function function (**CDF**) and a probability mass function (**PMF**) for discrete variable or a probability density function (**PDF**) for a continuous one.

17.1 *Discrete Random Variable*

The PMF is defined as follow:
$P_a = P(a) = P(X = a)$.
All outcomes a have their according probability $P_a \geq 0$ and as there is a countable number n of outcome, one can sum their probability to obtain the universe.
$\sum_{i=1}^{n} P(X_i) = 1 = P(\Omega)$.

The CDF is a step function in the case of discrete random variable. It is defined as:
$\mathbf{F}_x(a) = P(X \leq a) = \sum_{x \leq a} p_x$.

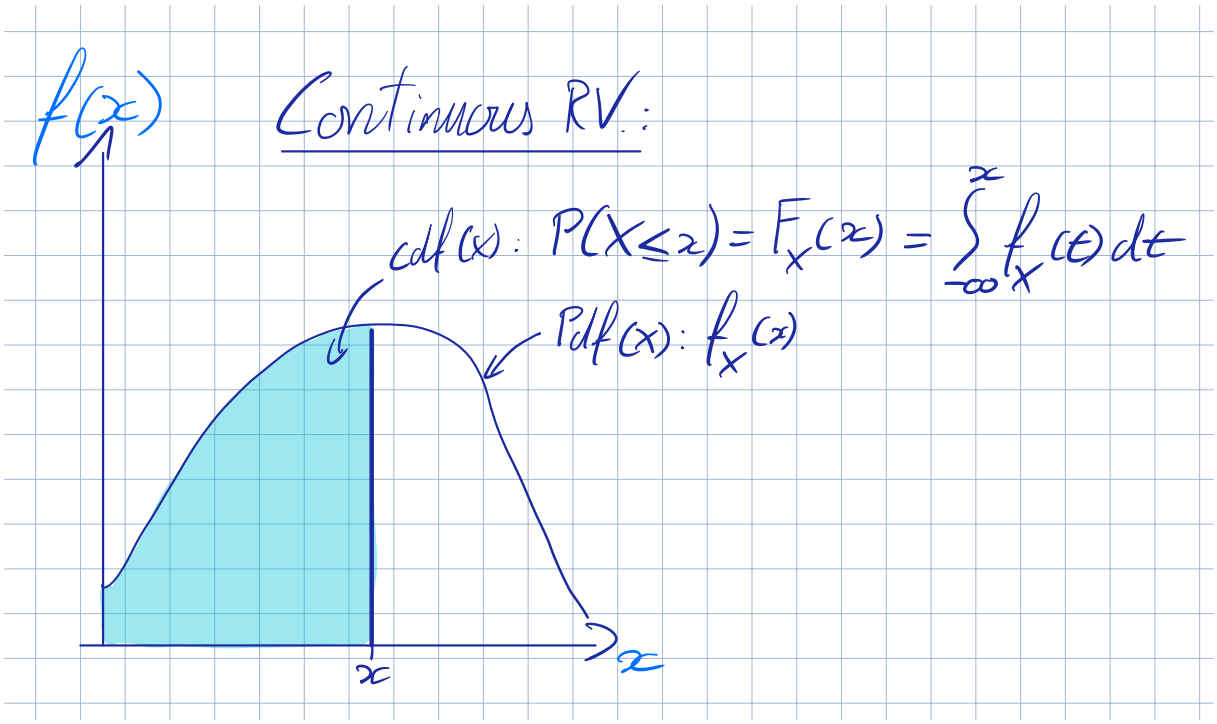
- $E(X) = \sum_{i=1}^{n} x_i \cdot p_i$
- $Var(X) = \sum_{i=1}^{n} p_i \cdot (x_i - E(X))^2$
  $\iff \sum_{i=1}^{n} p_i \cdot x_i^2 - E(X)^2$



17.2 *Continuous Random Variable*

To define the PDF, we need that: $f_X(x) \geq 0 \quad \forall x$ and $\int_{-\infty}^{\infty} f_X(x)dx = 1$.
As X can take an uncountable number of outcome, we cannot sum them all to define the CDF. Rather we take the area under the density function, namely: $\int_{-\infty}^{\infty} f_X(x)dx = F_X(x)$ defines the CDF.

- $F_X(x) = \int_{-\infty}^{x} f_X(t)dt = P(X \leq x)$
- $P(a \leq X \leq b) = \int_a^b f_X(x)dx$
- $E(X) = \int_{-\infty}^{\infty} x \cdot f_X(x)dx$
- $Var(X) = \int_{-\infty}^{\infty} f_X(x) \cdot (x - E(x))^2 dx =$
  $\int_{-\infty}^{\infty} x^2 \cdot f_X(x)dx - E(X)^2 = E(X^2) - E(X)^2$.



17.3 *Expected Values and Variance properties*

The expected value is as its name suggests the anticipated amount a random variable will take on average. It is denoted $E(X)$. The variance is a dispersion value to the mean. Consider X and Y two random variables, a any scalar or real number, then

- $E(X+Y) = E(X) + E(Y)$
- $E(a) = a$
- $E(aX) = aE(x)$
- $E(X+a) = E(X) + a$
- If X and Y are independents, then $E(X \cdot Y) = E(X) \cdot E(Y)$
- $Var(X) = E[(X - E(X))^2] = E(X^2) - E(X)^2$
- $Var(aX) = a^2 Var(X)$
- $Var(a) = 0$
- $Var(X+a) = Var(X)$
- $Var(X+Y) = Var(X) + Var(Y) - 2 \cdot Cov(X,Y)$ note that if X and Y are independent, their covariance equals 0.

- $\sqrt{Var(X)} = \sigma$ which is called the standard deviation
- $Cov(X,Y) = E[(X - E(X)) \cdot (Y - E(Y))]$

18 SOME DISCRETE PROBABILITY DISTRIBUTIONS

Here is a short list of representative distributions that you may see during the MASTAT courses. A more comprehensive list is either available in the Probability 1 slides or in A First course in Probability. [7]

18.1 *Binomial Distribution*

Denoted $X \sim \ Bin(n;p)$, with n being the number of trials and p the number of success.

- $f_X(x) = \binom{x}{p} \cdot p^x \cdot (1-p)^{n-x}$
- $E(X) = np$
- $Var(X) = np \cdot (1-p)$

Note that if $n = 1, \quad X \sim Bernouilli(p)$

18.2 *Poisson Distribution*

Denoted $X \sim \mathcal{P}(\lambda)$, with parameter λ.

- $f_X(x) = \frac{\lambda^x e^{-\lambda}}{x!}$
- $F_X(x) = e^{-\lambda} \sum_{i=o}^{x} \frac{\lambda^i}{i!}$
- $E(X) = \lambda$
- $Var(X) = \lambda$

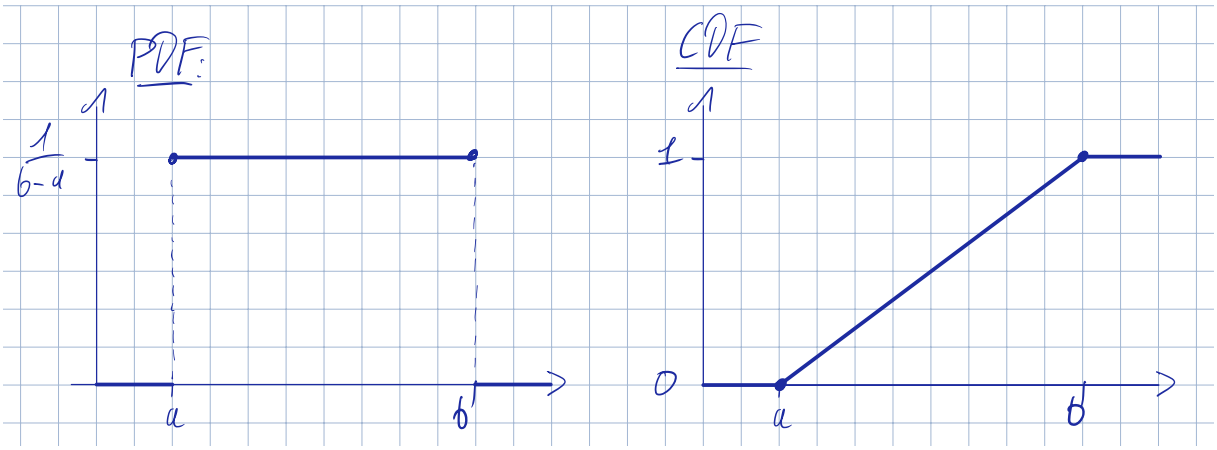
19 SOME CONTINUOUS PROBABILITY DISTRIBUTIONS

19.1 *Uniform Distribution*

Denoted $X \sim \mathcal{U}(a;b)$, with bounds $[a;b]$

- $f_X(x) = \begin{cases} \frac{1}{b-a} & \text{if} \quad a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$

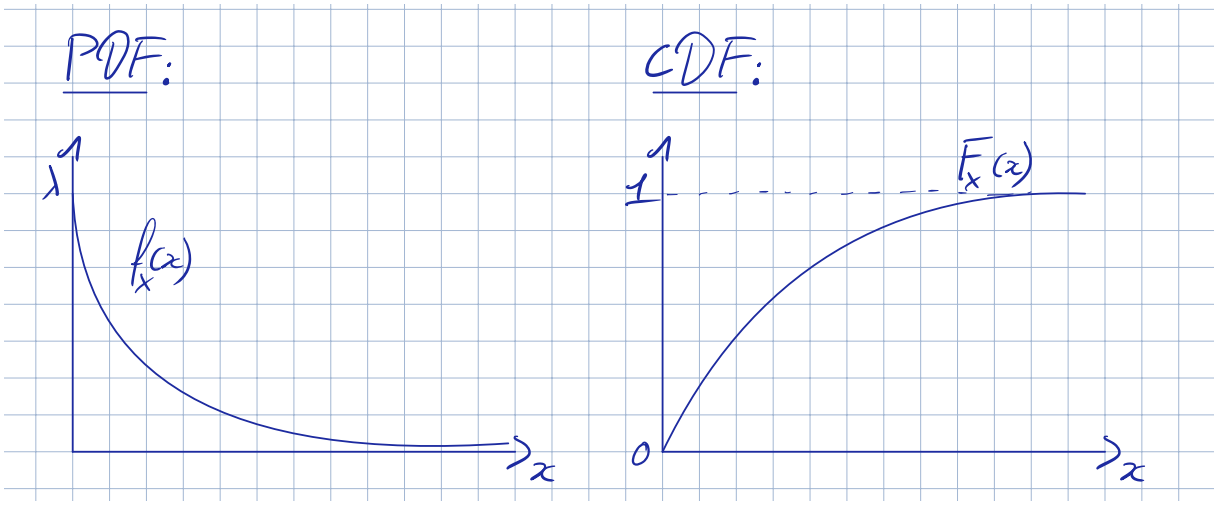
- $F_X(x) = \begin{cases} 0 & \text{if} \quad x < a \\ \frac{x-a}{b-a} & \text{if} \quad a \leq x \leq b \\ 1 & \text{if} \quad x > b \end{cases}$
- $E(X) = \frac{a+b}{2}$
- $Var(X) = \frac{(b-a)^2}{12}$



19.2 *Exponential Distribution*

Denoted $X \sim \mathcal{E}(\lambda)$, with parameter $\lambda > 0$

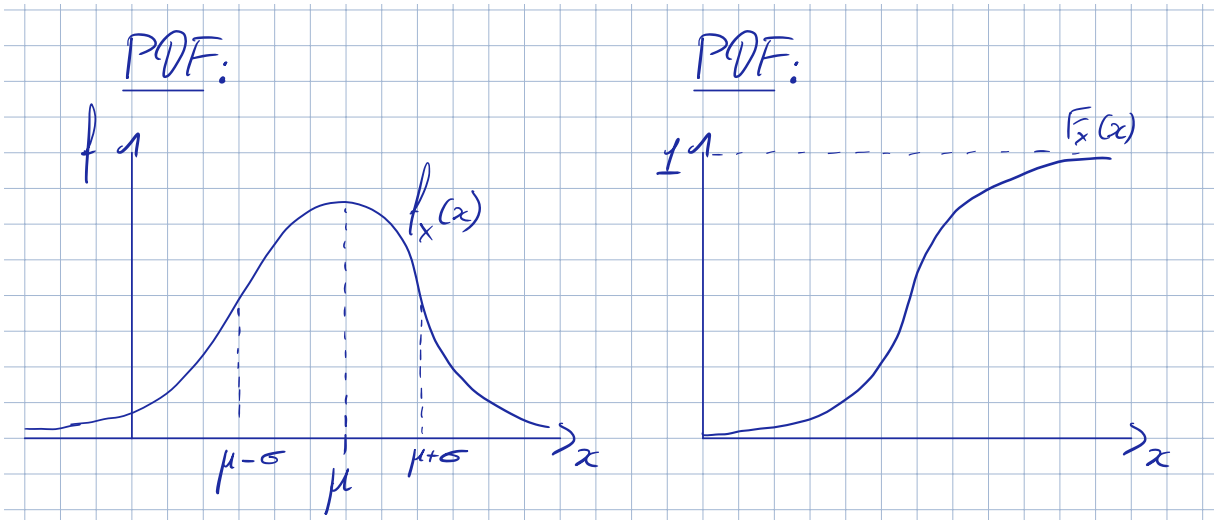
- $f_X(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
- $F_X(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
- $E(X) = \frac{1}{\lambda}$
- $Var(X) = \frac{1}{\lambda^2}$



19.3 *Normal Distribution*

Denoted $X \sim \mathcal{N}(\mu, \sigma^2)$, with parameter μ as its mean and σ^2 as its variance.

- $f_X(x) = \frac{1}{\sqrt{2 \cdot \sigma^2 \pi}} e^{-\frac{1}{2}(\frac{(x-\mu)}{\sigma})^2}$
- $F_X(x) = \int_{-\infty}^{x} f_X(t)dt$
- $E(X) = \mu$
- $Var(X) = \sigma^2$



Another useful normal distribution is the so-called log-normal distribution, written $X \sim ln\mathcal{N}(\mu, \sigma^2)$.
$E(X) = e^{\mu + \frac{sigma^2}{2}}$ and $Var(X) = (e^2 - 1)e^{2\mu + \sigma^2}$.

20 ASYMPTOTIC THEORY

This section mainly focuses on the basic of the asymptotic theory in probability, meanings what happens when $n \to \infty$. We will only speak about the two main elements of asymptotic theory, namely convergence in probability and convergence in distribution as you will be seeing other specifities later during the MASTAT.

20.1 *Convergence in Probability*

We say that T_n converges in probability to a constant c if and only if $lim_{n\to\infty} P(|T_n - c| > \epsilon) = 0, \forall \epsilon > 0$.
We denote that convergence in probability as $plim_{n\to\infty} T_n = c$ or $T_n \xrightarrow{p} c$.

T_n can also converge to a random variable T. This is the case if and only if $lim_{n\to\infty} P(|T_n - T| > \epsilon) = 0, \forall \epsilon > 0$.

20.2 *Convergence in Distribution*

Now consider T_n with $F_n(t)$ as its CDF. If $lim_{n\to\infty} |F_n(t) - F(t)| = 0$ at every point of continuity $F(t)$, then $T_n \xrightarrow{d} T \sim F(t)$.
Furthermore, $F(t)$ is called the limiting distribution of T_n.

20.3 *Properties of the plim operator*

Let $T_n \xrightarrow{p} c$ and $S_n \xrightarrow{p} d$ be two random sequences. Then:

- $T_n + S_n \xrightarrow{p} c + d$
- $T_n * S_n \xrightarrow{p} c * d$
- $T_n / S_n \xrightarrow{p} c/d$
- $plim\, g(T_n) = g(plim\, T_n)$ for any $g(\cdot)$ continuous

Let $T_n \xrightarrow{d} T$ and $S_n \xrightarrow{p} c$ be two random sequences. Then:

- $S_n * T_n \xrightarrow{d} c * T$ meaning that the limiting distribution of $S_n * T_{n)}$ is cT
- If $g(\cdot)$ is a continuous function, then $g(T_n) \xrightarrow{d} g(T)$
- If $S_n \xrightarrow{d} S$ and $plim\,(T_n - S_n = 0)$, then T_n and S_n have the same limiting distribution

Asymptotic Theory

If you need more information on Asymptotic Theory, we strongly recommend to follow Econometrics - Masters course as it focuses a lot on that.

21 LIMIT THEOREMS

Here is a brief presentation of the Weak Law of Large Number (WLLN) and the Central Limit Theorems (CLT), which are key theorems in Probabilities and Statistics. You will be using those a lot throughout the master, so it is best to be acquainted with them.

21.1 *Weak Law of Large Numbers*

The WLLN describes what happens when an experiment is conducted repeatedly n numbers of time. It states that the sample mean of the results will converge to the expected value.
Let $X_1, ..., X_n$ be a collection of i.i.d. random variables from a distribution F with mean μ and $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ their sample average, then:

$lim_{n\to\infty} P(|\bar{X}_n - \mu| > \epsilon) = 0$ or simply $\bar{X}_n \xrightarrow{p} \mu$.

21.2 *Central Limit Theorem*

The CLT states that for i.i.d random variables, their standardized mean converges to a normal distribution even though they were not normally distributed.
Let $X_1, ..., X_n$ a collection of i.i.d random variable with $E[X] = \mu$ and $Var[X] = \sigma^2 < \infty$. Let $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ be their sample mean, then:

$\sqrt{n}(\bar{X}_n - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2)$.

Limit Theorems

You will be seeing those theorems in a different classes throughout your courses at the MASTAT such as The statistical analysis of time series.

22 CONCENTRATION IN INEQUALITY

To end this probability chapter, we will present two famous inequalities in probability, namely the Markov's inequality and Chebyshev's inequality.

22.1 *Markov's Inequality*

let X be a non-negative random variable and $a > 0$, then:

$P(X > a) \leq \frac{E[x]}{a}$.

22.2 *Markov's Inequality*

Let X be a random variable having a finite non-zero variance σ^2 and finite mean μ. Let $k \in \mathbb{R}_+^*$, then:

$P(|X - \mu|) \geq k\sigma) \leq \frac{1}{k^2}$.

Part V.
Statictics

Introduction

As you are all master of statistics students, you must already have some notions of statistics. We would like to point out the following. It is always advisable to stay critical about the results of the functions, tables, data. Before proceeding with the analysis, always ask yourselves the following questions:

- Does the data look realistic? Are there any observations that don't make sense, for example age being -700 for a human?
- Always double check the most basic of the calculations.
- Always make sure the sample size is big enough. In the masters, you will learn about the asymptotic analysis, the asymptotic properties of the estimators as well as small-sample analysis.
- Does the data collection entail an estimation bias? For example, asking exclusively people at the gym how many hours per week they sport won't produce any reliable results for the whole population of your city!

Just for fun, try and uncover misleading statistics in the media!
All of that being said, let's proceed to the quick summary of the material.

22.3 *Quick summary*

This section regroups the basic notions and introduction into Statistics.
Population is the whole (often unobserved) group of people.
Sample is a smaller, representative group of the population.
Data are the actual measurements of the study.

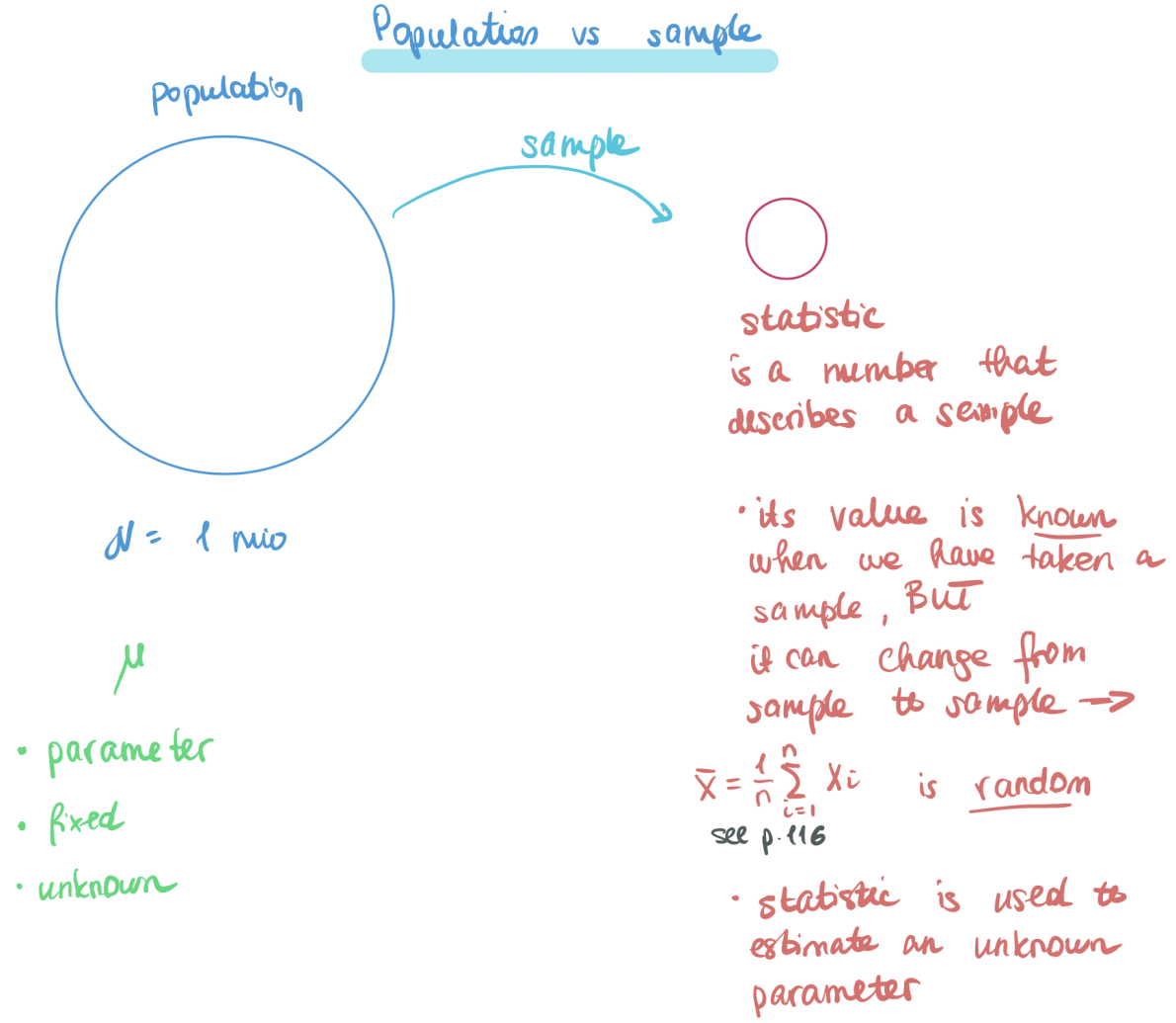
23 RANDOM VARIABLES

Talking about population and sample, we can define the following variables as fixed:

- population mean,
- population size,
- population variance, etc.

On the other hand, when we take a (random) sample from a population, we start dealing with random variables. The statistics summarize a sample, for example:

Figure 1: Population vs sample.



- sample mean,
- sample variance, etc

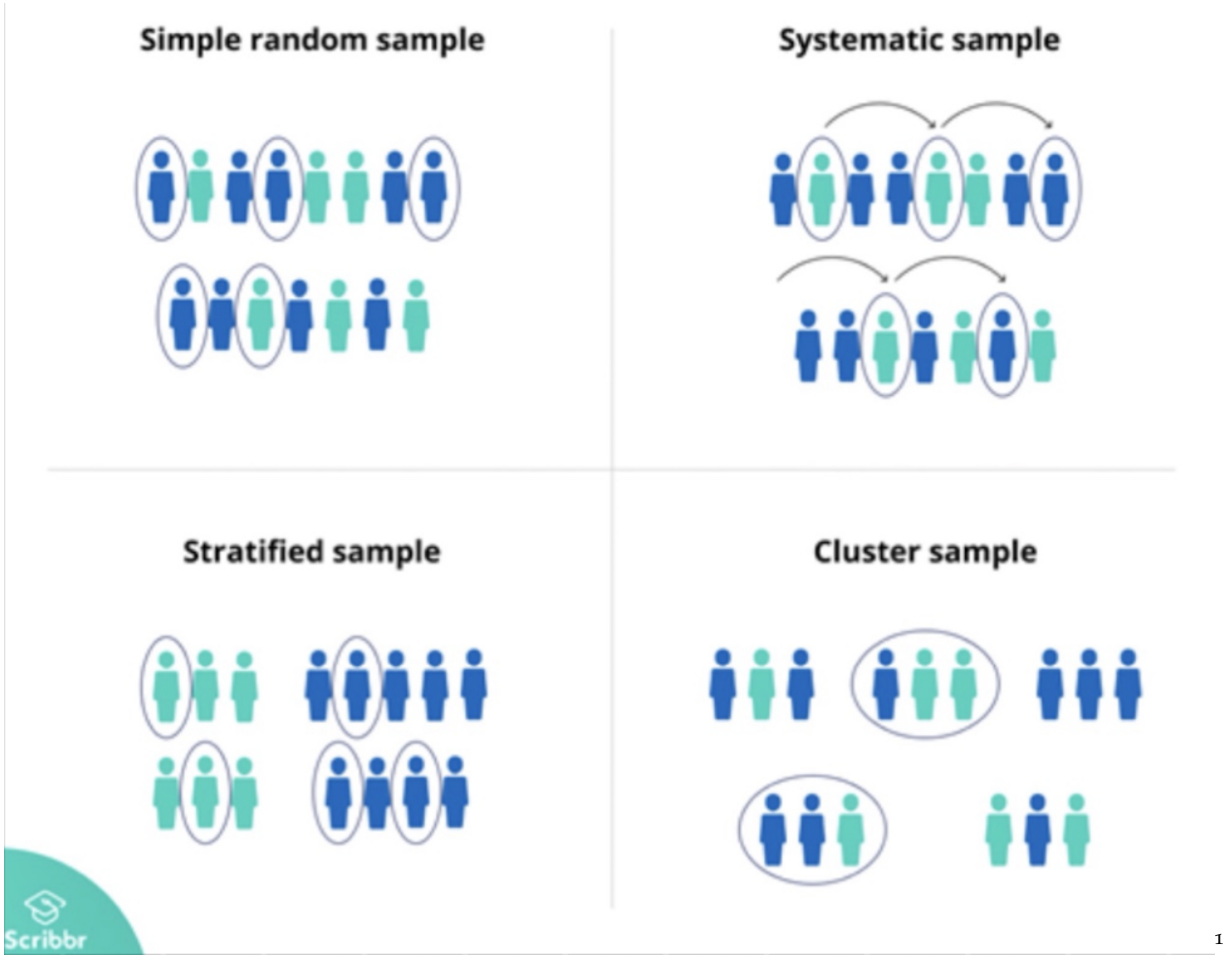
Why are they random? Because their value changes from sample to sample.

23.1 *Sampling schemes*

How do we sample then? There are different techniques, one of the most basic being a **simple random sampling (SRS)**. Then, every possible sample of a given size has every chance to be chosen.

There is also a **stratified random sampling**, where the idea is to divide the population into *strata*, or groups of individuals similar in some way.

Figure 2: Sampling schemes.



Sampling schemes

If you need a refresher on this topic, you can check out Statistics lectures, Preliminary Work Week 4 and Moore et al. book, pages 401 and on.

23.2 *Discrete random variables*

A discrete random variable is one that can take values from an ordered list. [5].

Endogenous variables are variables that are changed or determined by its relationship with other variables in the model.

Exogenous variables are the independent variables, sometimes called the outside forces.

Population and sample

If you need a refresher on this topic, you can check out Statistics lectures, Preliminary Work Week 1.

The **mean** is the arithmetic mean of the sum of all the observed values. Consider the sum of all the observations divided by the number of the observations.

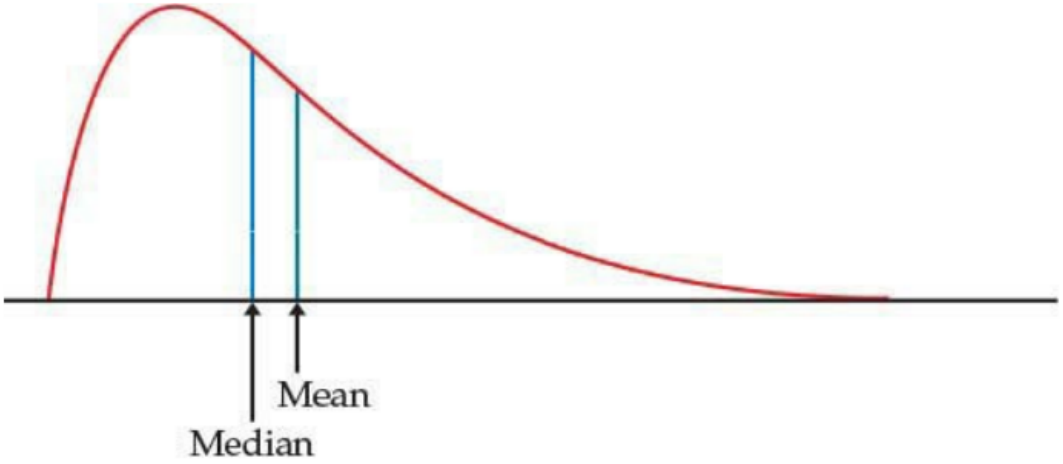
Median is another way to measure the center of the data. It truly represents the "middle" of the dataset. It is obtained by first sorting the data and then finding the middle value by using the rank:

$$r = p \cdot (n - 1) + 1, \tag{3}$$

where r is the rank, p the probability, and n the number of observations.

These images give a great intuition behind the mean and median and come from a fairly easy to read book you can turn to if you feel like you need to know more about statistics [5]:

Figure 3: The median and the mean.



(b)

FIGURE 1.21
(a) A symmetric Normal density curve with its mean and median marked. (b) A right-skewed density curve with its mean and median marked.

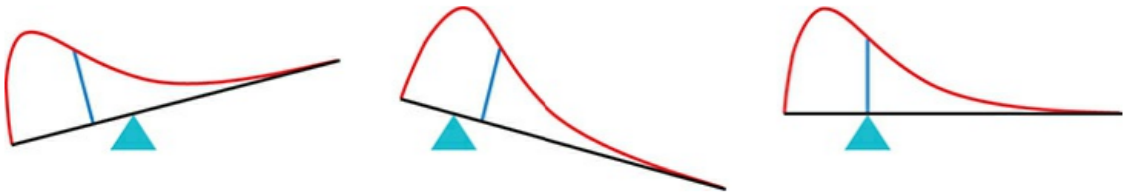


FIGURE 1.22
The mean of a density curve is the point at which it would balance.

Measures of centrality

If you feel you need more information about this topic, you can take a look at the Statistics lectures, Preliminary Work Week 2.

24 QUANTILES, PERCENTILES...

Quite a lot of confusion can be caused by the misunderstanding of the quantiles and percentiles. **Quantiles** actually divide the distribution into areas with a certain probability. For example, the **median** always is the 0.5 quantile: per definition, the median divides the probability distribution

into areas of equal probability. With each quantile, there is an associated **probability**, or area under the cumulative distribution function curve.

Here is a quick visual summary of the concept for the continuous random variables.

Figure 4: CDF of a continuous random variable.

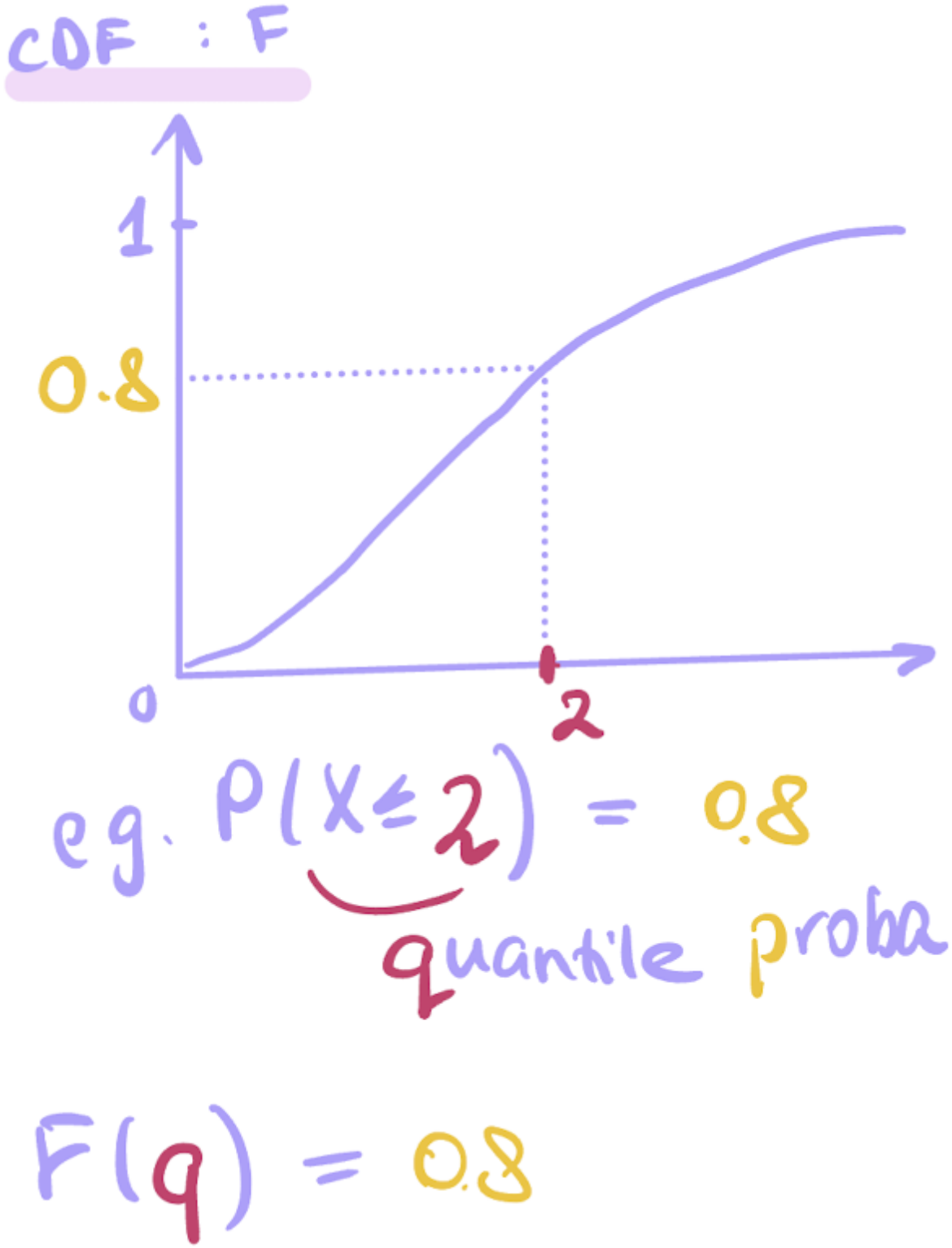
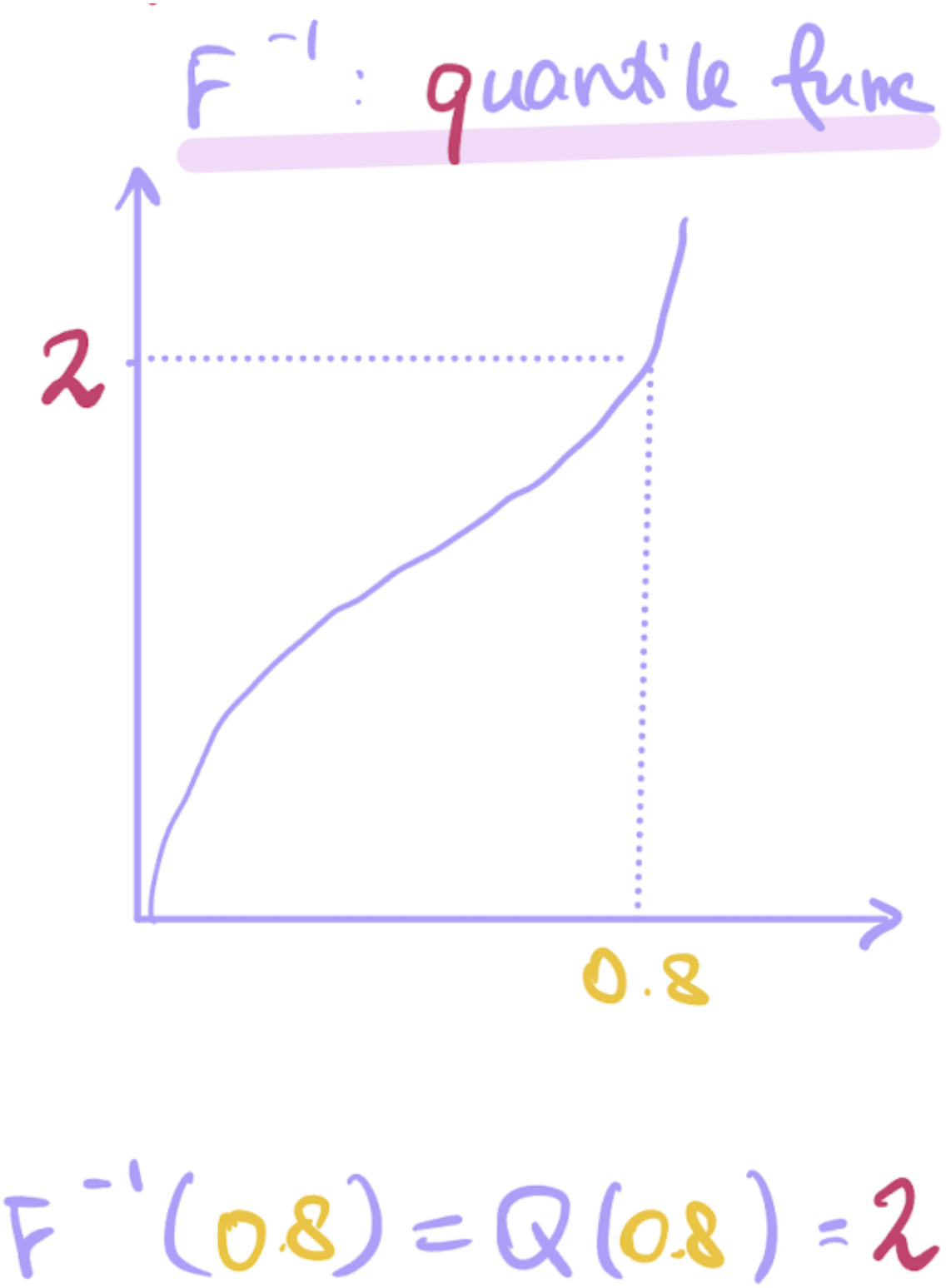
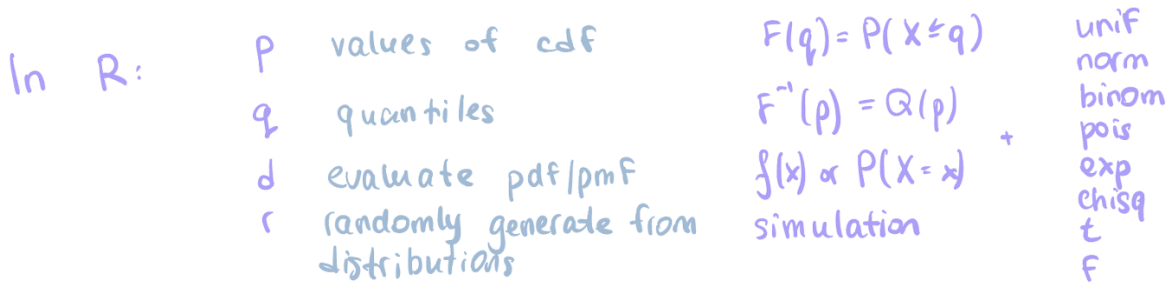


Figure 5: The quantile function.



When working with R, it is helpful to remember which command to use when you want to find out the quantiles, percentiles or simulate from a certain distribution.

Figure 6: Useful functions in R.



Quantiles

To find out how to compute the quantiles, median, and construct graphs based on those quantities, check out Statistics lectures, Preliminary Work Week 2.

25 GRAPHICAL REPRESENTATION

When working with data, the first thing you always want to do is represent the data graphically. Here is why:

- it lets you simplify data sets,
- shows the basic tendencies,
- helps you spot the outliers,
- and so on.

You can represent data graphically in many ways. The R Cheatsheet for the `ggplot2` package lets you always keep a clear summary of what graph you can use for which kind of data (one, two, or more variables, continuous or discrete data)[23]. Make sure you at least understand exactly how to build and read the following plots:

- histogram,
- boxplot,
- mosaic plot,

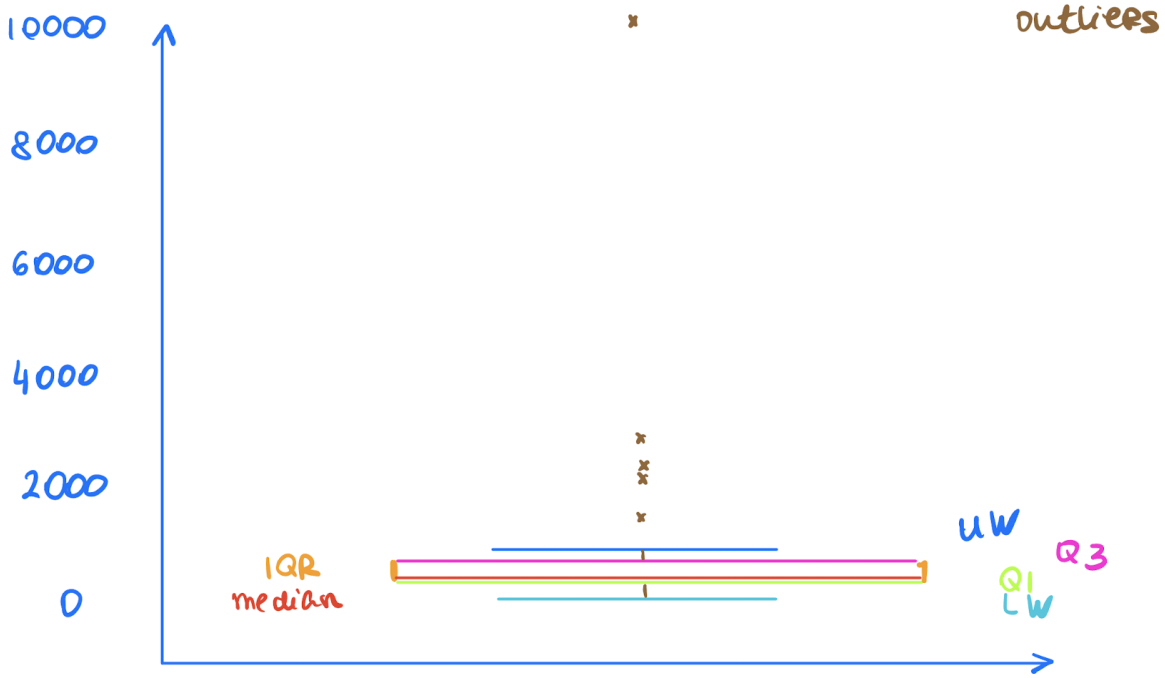
- density plot,
- QQ-plot.

In order to understand the working of R, it is sometimes useful to recreate the plots by hand. For example, Figure 7 represents a boxplot computed by hand using the ranks and the quantiles of the data. The same graph can be created using the `boxplot()` command.

The steps are the following.

1. Sort the data from smallest to largest observation.
2. Compute the Q1, Q2 and Q3 using formula specified in equation 3, where Q1 represents the 0.25 quantile, Q2 0.5 quantile and Q3 0.75 quantile.
3. Compute the IQR, Interquantile Range: $IQR = Q3 - Q1$.
4. Compute the LB, Lower Bound: $LB = Q1 - 1.5 \times IQR$.
5. Compute the UB, Upper Bound: $UB = Q3 + 1.5 \times IQR$.
6. To find the UW (Upper Whisker) and LW (Lower Whisker), find the data point closest to the UB and LB, respectively, but still within the bounds.
7. All data outside the LB and UB are the outliers.

Figure 7: Boxplot drawn by hand.



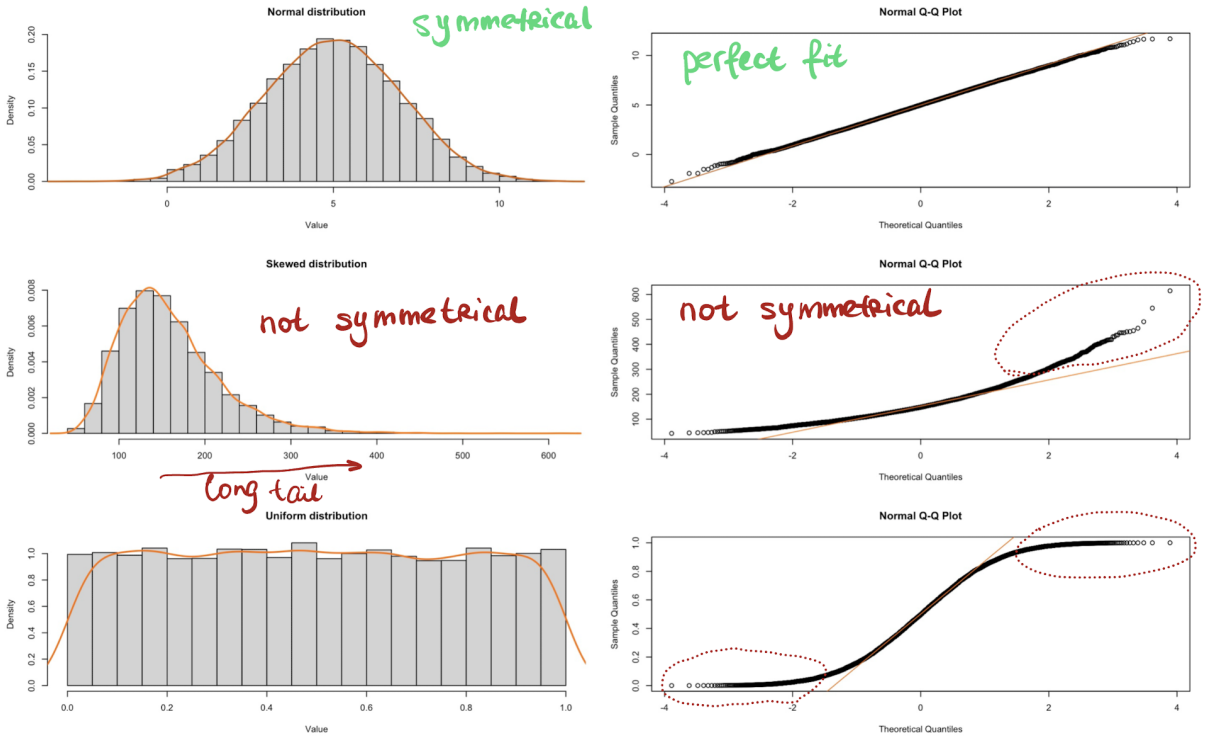
Graphical representation

For more information about the histograms, boxplots and contingency tables see Statistics lectures, Preliminary Work Weeks 2 and 3.

25.1 *Q-Q plot*

Q-Q plot helps you compare two probability distributions one against another. The following plots compare the simulated distributions to the theoretical quantiles of the normal distribution with the given parameters. The plots will give you an intuition behind the concept.

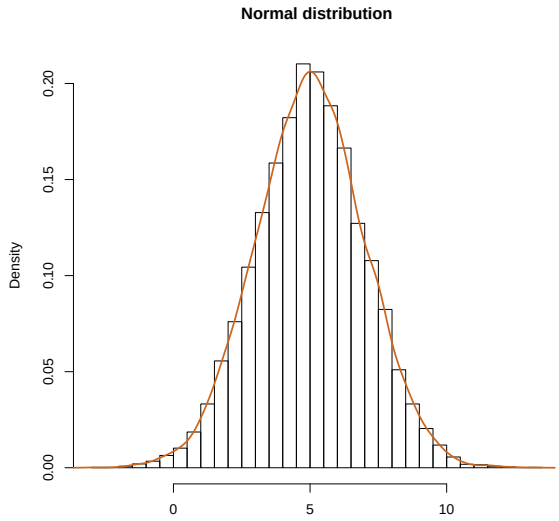
Figure 8: Distribution and QQ-plot.



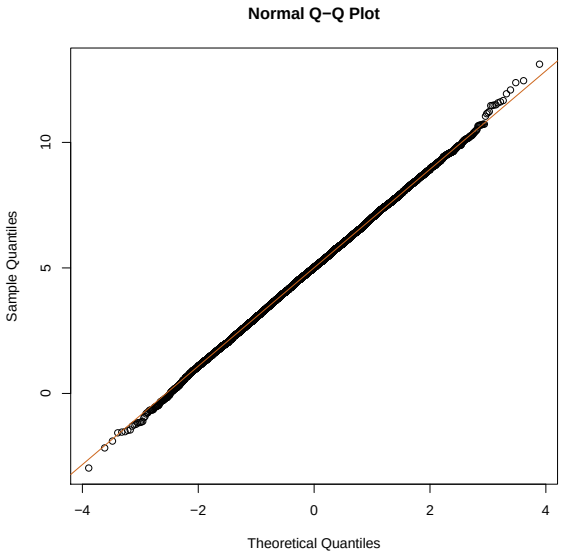
You can recreate the plots using the following `R` codes. The codes contain the function `density`, which computes the kernel density estimates. You can find more information about it here.

Here we simulate from the normal distribution with mean 5 and standard deviation of 2.

```r
# Normal distribution
norm.distr.sim=rnorm(10000, mean=5,
sd=2)
hist(norm.distr.sim,breaks = 30,
probability = T,
#histogram plot
xlab="",main = "Normal distribution")
# add a density plot
lines(density(norm.distr.sim),
 lwd = 2, # thickness of line
 col = "chocolate3")
```



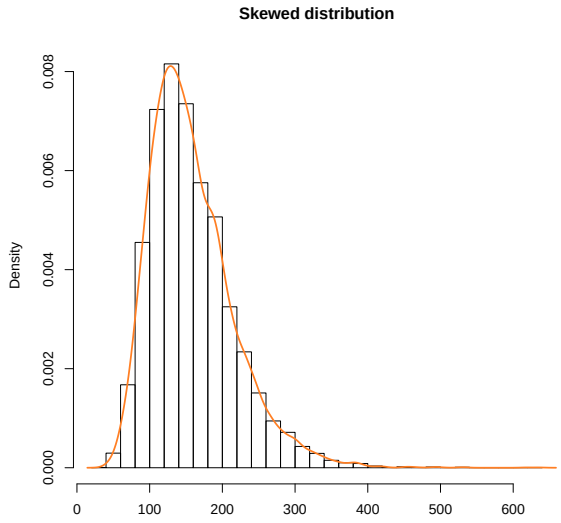

```r
# QQ plot of the normal distribution
qqnorm(norm.distr.sim)
qqline(norm.distr.sim, col="chocolate3")
```



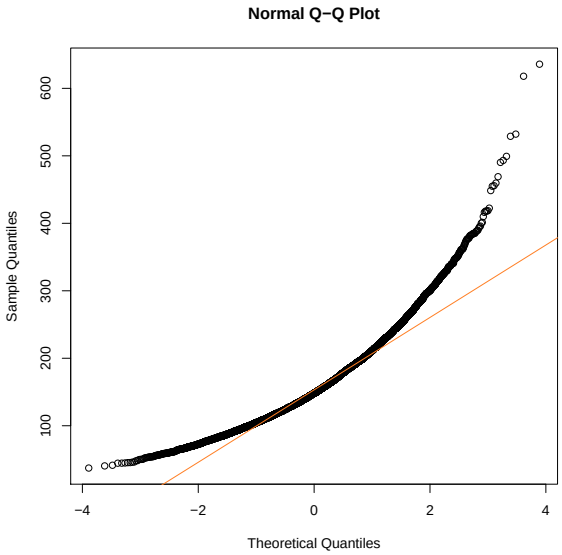
Here we simulate from a skewed (lognormal) distribution.

```r
# Simulate from a skewed distribution
skewed.distr <- rlnorm(10000, 5, 0.35)

#histogram plot
hist(skewed.distr,breaks = 30,
probability = T,xlab="",
main = "Skewed distribution")
# add a density plot
lines(density(skewed.distr),
 lwd = 2, # thickness of line
 col = "chocolate1")
```



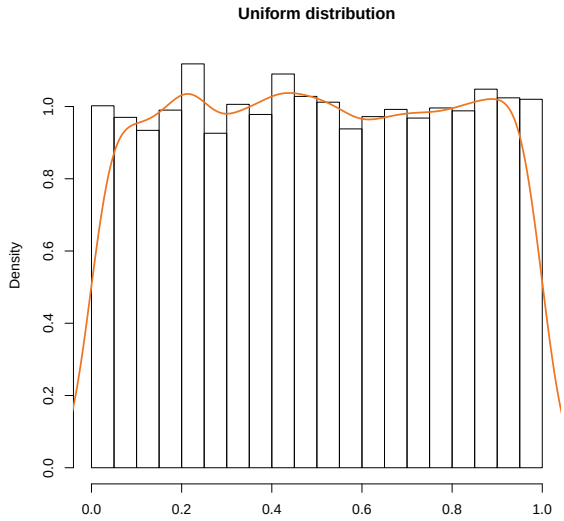

```r
# QQ plot of the skewed distribution
qqnorm(skewed.distr)
qqline(skewed.distr, col="chocolate1")
```



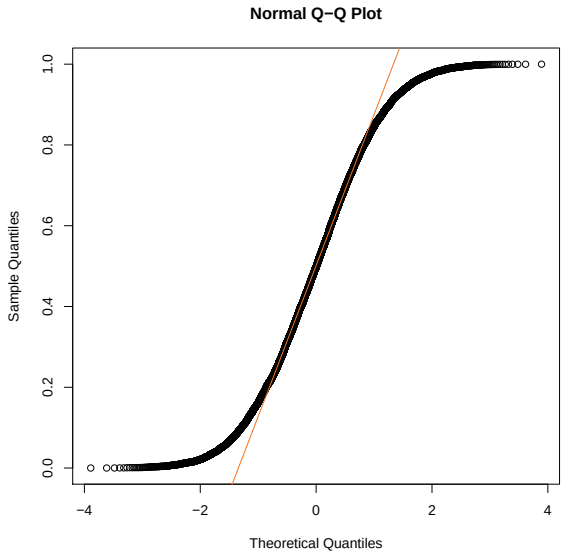
Here we simulate from a uniform distribution.

```r
# Simulate from uniform
skewed.distr2 <- runif(10000)

#histogram plot
hist(skewed.distr2,breaks = 30,
probability = T,xlab="",
main = "Uniform distribution")
# add a density plot
lines(density(skewed.distr2),
 lwd = 2, # thickness of line
 col = "chocolate2")
```




```r
# QQ plot of the skewed distribution
qqnorm(skewed.distr2)
qqline(skewed.distr2, col="chocolate2")
```



Q-Q plot

For more information about Q-Q plot, see Statistics lectures, Preliminary Work Week 3.

26 ESTIMATION

Often, the population parameters are unknown and you'll need to estimate them through sampling.

26.1 *Point estimation*

What is an estimator, really? Remember that it's a function of the sampled variables, $X_1, \ldots, X_n$ and is thus a *random* variable. Estimator is a gentleman and always wears a **hat**, for example, θ is a population parameter, while $\hat{\theta}$ denotes an estimator of the univariate parameter. Also, it is important to distinguish between the random variables $X_1, \ldots, X_n$, because each sample is random and thus the values change from one time to another, while small letters $x_1, \ldots, x_n$ denote the realizations and are thus known values. This is very important to distinguish. Here is an example in R, where we can simulate from a standard normal distribution many times - here, 3, just for the purpose of the illustration, and each time the values are different, while, as long as we store the simulation in a variable, it becomes a certain *realisation* of a random variable.

In the left-side section three observations of the random variable X are observed three times. In the section on the left, the recorded observations are represented three times. The values do not change as these observations $x_1, x_2 and x_3$ are not random anymore.

```
#simulation from standard normal
# X1,X2,X3 are all different:
rnorm(3, mean=0, sd=1)

## [1]  0.8890266  2.4347347 -0.7719102

rnorm(3, mean=0, sd=1)

## [1]  0.4152474 -1.1185879 -1.5135935

rnorm(3, mean=0, sd=1)

## [1]  0.1491298  1.6576958 -0.3087861
```

```
# realisations x1,x2,x3:
x=rnorm(3, mean=0, sd=1)
x

## [1] -1.03292099  0.01447077  0.72651270

x

## [1] -1.03292099  0.01447077  0.72651270

x

## [1] -1.03292099  0.01447077  0.72651270
```

The two estimators you'll see most often are the mean and the variance. Here we give the formulas without any further explanation.

Mean

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

A mean has a very natural intuitive meaning: it's an arithmetic mean of the sum of all the observations divided by the number of the observations. By noting $\bar{X}$ the sample mean and

μ the population mean and under some assumptions, we have that asymptotically (in other words, when the number of observations tends to infinity), the sample mean will converge to the population mean. Let $\bar{X} = \sum_{i=1}^{\infty} X_i$. We then have that $\mathrm{E}(\bar{X}) = \mu$ and $\mathrm{Var}(\bar{X}) = \sigma^2/n$. Applying Chebyshev's inequality to $\bar{X}$, we get

$$\Pr\left(\frac{|\bar{X} - \mu|}{\sigma/\sqrt{n}} \geq x\right) \leq \frac{1}{x^2},$$

so

$$\Pr\left(|\bar{X} - \mu| \geq \frac{x\sigma}{\sqrt{n}}\right) \leq \frac{1}{x^2}.$$

This holds if X is any positive number. Hence it holds when $x = \varepsilon\sqrt{n}/\sigma$, where ε is any positive number. That gives then

$$\Pr(|\bar{X} - \mu| \geq \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2 n} \to 0 \text{ as } n \to \infty,$$

or, in other words

$$\bar{X} \xrightarrow{p} \mu$$

, or $\bar{X}$ converges in probability to μ.

Variance

The variance is a measure of variability, the squared differences from the mean. It tells us the degree at which the data is spread. We can compute it by

$$\mathrm{Var} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2.$$

26.2 *Confidence intervals*

When you compute an estimator, an important part is to know by how much that estimator is expected to vary. Confidence intervals give important information about the accuracy of an estimate.

To put it into a context, imagine you draw a sample of 100 observations 100 times. Each time you draw a sample of 100, you compute the mean. Then, you plot these means using a histogram. Then, a 95% confidence interval will cover 95 of these means.

How to compute the confidence intervals?

You have 2 situations, namely, when σ is known and when σ is unknown. This distinguish will entail the important theoretical results and properties of the estimators. Putting it grossly, a confidence interval can be seen as

$$\textit{estimate} \pm \text{margin of error.}$$

26.2.1 *Confidence interval for a population mean, σ known*

$$\left[\bar{X} \pm z_{\left(1-\frac{\alpha}{2}\right)} \frac{\sigma}{\sqrt{n}}\right], \tag{4}$$

where $z = (x - \mu)/\sigma$ and $z_{\left(1-\frac{\alpha}{2}\right)}$ is the $1 - \frac{\alpha}{2}$ quantile of the standard normal distribution. In order to find out the value, you can either use a z-table or refer to statistical programs, such as R and remember the recap table in section 6.

26.2.2 *Confidence interval for a population mean, σ unknown*

When σ is unknown, you'll have to estimate it and thus will be dealing with more uncertainty, which will result in a wider interval, given all other parameters are constant.

$$\left[\bar{X} \pm t_{n-1,1-\frac{\alpha}{2}} \frac{s}{\sqrt{n}}\right], \tag{5}$$

where $t_{n-1,1-\frac{\alpha}{2}}$ is the $1 - \frac{\alpha}{2}$ quantile of the t-distribution with $n - 1$ degrees of freedom.

Estimation and confidence intervals

For more information about point estimation, confidence intervals, formulas for confidence intervals for proportion, σ^2, difference of means and the theoretical properties of the estimators, see Statistics lectures, Preliminary Work Weeks 5 and 6, as well as Moore et al. book, pages 356 and on.

27 COMPARING ESTIMATORS

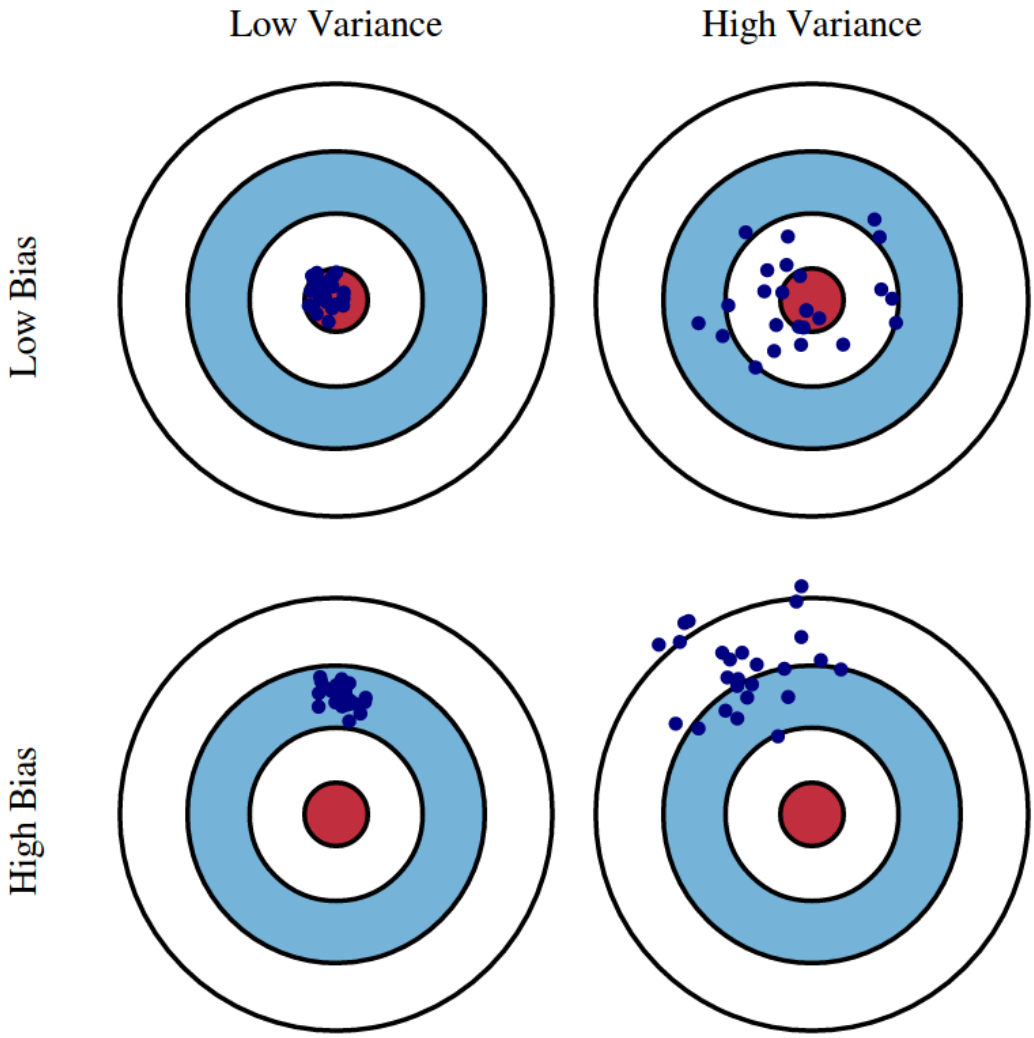
When comparing the estimators, two important notions are those of **accuracy** and **precision**. To get an intuition behind the concept, the following graphical representation might help: Figure 9.

Accuracy can be defined as the closeness or nearness of the measurements to the true or actual value of the quantity being measured.[1] The term accuracy is also used to as a ratio between the correct predictions and the total number of predictions in the classification problems.

The variance of the observations can be used as a measure of the accuracy. You can think about it in the following manner. If an estimator has a very big variance, the chances the true value is someone in there get higher, as the variance get bigger. Thus, the estimation becomes less and less accurate.

Precision is the term mostly associated with bias. Although, some sources say that bias is a systematic error whereas precision is a random error [1].

Figure 9: Bias and variance.



Let us go through both of them step by step.

27.1 *Bias*

When you draw a sample and compute an estimator, it is important to know how close to the true value that estimator is. When thinking of bias, you should directly think of the **expectation**. An **unbiased** estimator is the one that is equal to the true parameter.

When thinking about bias, you should immediately think of expectation. Bias is defined as follows:

$$\text{bias}\left(T_n, \theta\right) = E\left(T_n\right) - \theta, \tag{6}$$

where $T_n := T(X_1, \cdots, X_n)$ is an estimator of $X_1, \ldots, X_n$ and θ the true parameter.

If bias $(T_n, \theta) = 0$, we say that the estimator is unbiased. Bias $(T_n, \theta) < 0$ indicates underestimation of θ. Bias $(T_n, \theta) > 0$ indicates overestimation of θ.

27.2 *Variance*

The formal definition of the estimator's variance is the following.

$$\text{Var}(T_n) = E\left((T_n - E(T_n))^2\right) \tag{7}$$

.

In other words, variance expresses variability around the expectation of the estimator T_n.

27.3 *MSE: Mean Squared Error*

It goes without saying the the best estimator is the one that has both high accuracy and precision in figure 9. Let's formalize this.

Mean squared error is defined as the sum of bias squared plus variance.

$$\text{MSE}(T_n, \theta) = \text{Var}(T_n) + \text{bias}((T_n, \theta))^2, \tag{8}$$

27.4 *Efficiency*

When comparing two estimators, one can compare the MSE's for both estimators. The following is the definition of efficiency.

The efficiency of an estimator $T := T(X_1, \ldots, X_n)$ of θ with respect to an alternative estimator $U := U(X_1, \ldots, X_n)$ of θ is defined by:

$$\text{eff}(T, U) = \frac{\text{MSE}(U, \theta)}{\text{MSE}(T, \theta)} \tag{9}$$

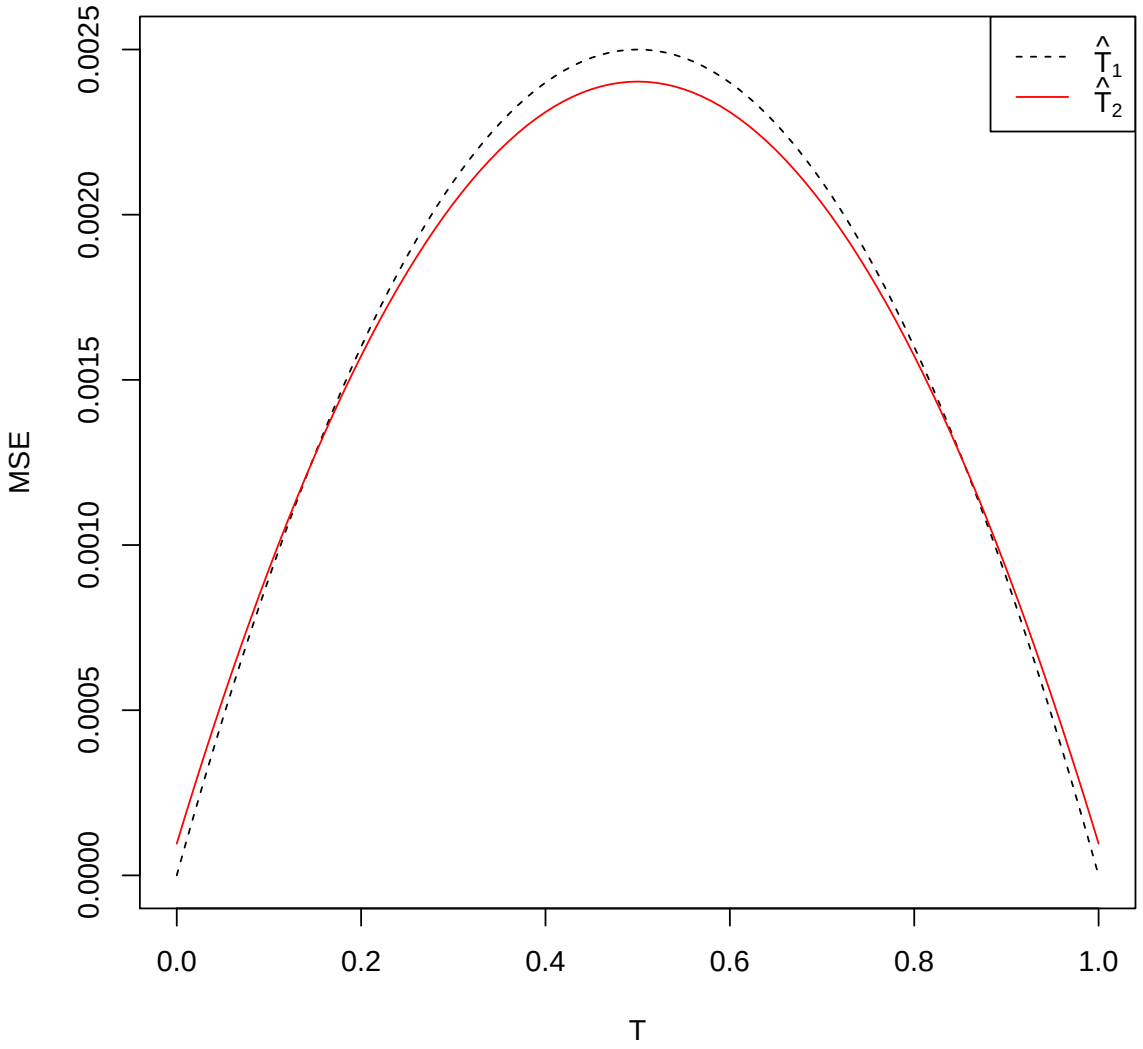
We are looking for an estimator with the **lowest** error, thus making:

- if $\text{eff}(T, U) < 1$, estimator U should be preferred,
- if $\text{eff}(T, U) > 1$, estimator T should be preferred.

This can be demonstrated graphically in the following way. MSE represents a curve as a function of parameter T. We can observe that the curve separates the MSE in two regions. Choosing the right estimator thus depends on many factors.

- when $T \in [0, 0.15)$ or $T \in (0.85, 1]$: $MSE(\hat{T^{(2)}}, T) > MSE(\hat{T^{(1)}}, T)$,
- when $T \in (0.15, 0.85)$: $MSE(\hat{T^{(1)}}, T) > MSE(\hat{T^{(2)}}, T)$.

Illustration for the MSE. Here T represents the proportion of the observed values.



27.5 *Consistency*

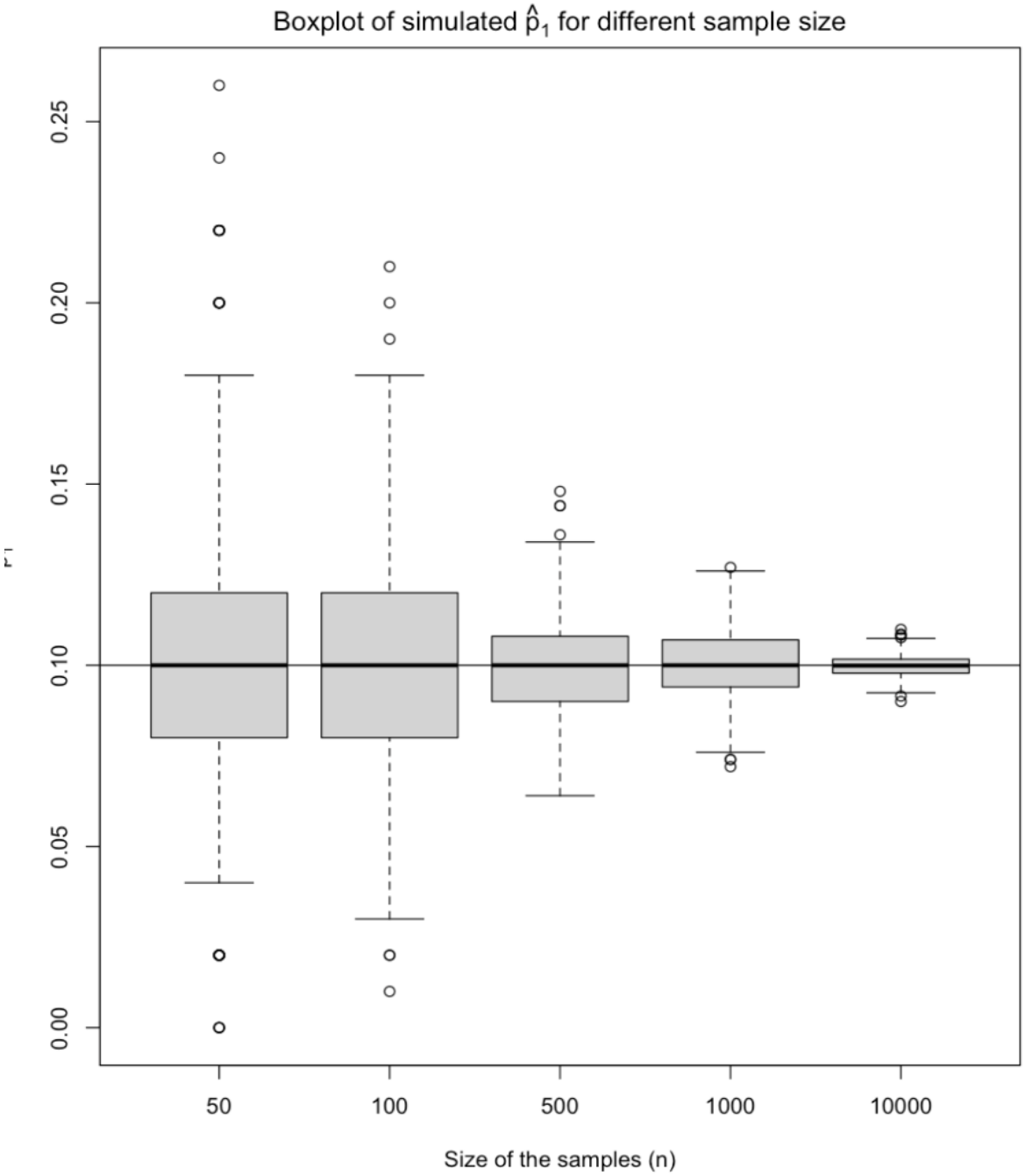
A consistent estimator "approaches" the true parameter θ as the sample size increases. Put more formally, an estimator $T(X_1, \ldots, X_n)$ is said to be consistent if $\text{Plim}_{n\to\infty} T(X_1, \ldots, X_n) = \theta$, that is, if $\forall \epsilon > 0$ [2]

$$P(|T(X_1, \ldots, X_n) - \theta| > \epsilon) \longrightarrow_{n\to\infty} 0.$$

2 See also Probability section for more information. This will come in useful during the first Time Series lecture.

Let's again illustrate this with an example. In figure 10, we are looking at simulations of a random variable following a Bernoulli distribution. On the $x - axis$ different n are represented. The true parameter p is represented by the black horizontal line.

Figure 10: Bias and consistency.



The boxplots show that $\hat{p}_1$ is an unbiased and consistent estimator. It is unbiased because $E(\hat{p}_1) = p$ for any n; it is consistent since the sampling distribution of $\hat{p}_1$ is becoming more concentrated at p as the sample size increases.

Bias, variance, MSE and efficiency

For more information about choosing the right estimator, see Statistics lectures, Preliminary Work Week 7, Probability and Statistical Learning lectures, Section 7, Subsections 3 and 4, Heumann and Shalabh [4] book chapter 9, Moore and McCabe [5] pages 210-212 for bias and variance, Rice [6] book p. 127 for MSE.

28 CONSTRUCTING ESTIMATORS

When constructing estimators, two main approaches exist. You will use both throughout the whole masters, make sure you are very familiar with the concepts as well as procedures to derive these estimators.

28.1 *Method of moments*

The method of moments works by matching the population moments to the empirical moments. The method is very intuitive and simple to implement. The procedure is as follows.

Method of moments implementation.

Step 1: compute the empirical moment of order k for the population

Step 2: replace the empirical moment with sample moment

The population moment of order k is defined as

$$\mu_k = E(X^k),$$

and the empirical sample moment as

$$\hat{\mu_k} = \frac{1}{n}\sum_{i=1}^{n} X_i^k.$$

For example, $X \sim Ber(p)$ (X following Bernoulli distribution), we know $E(X) = p$, we thus replace $E(X)$ by $\bar{X}$ and get $\bar{X} = p \Rightarrow \hat{p}_{MM} = \bar{X}$.

28.2 *Maximum likelihood*

The idea behind the maximum likelihood is to find a pdf that is **most plausible** based on the observations.

For the intuition, you can watch this video - by the way, the whole channel is great to gain intuition behind many statistical concepts.

The method is very intuitive and allows you to get the full exploit of the information at hand. The likelihood function can be used to summarize data. Whereas when working with distribution, we focus on the observations given the same function parameter, when working with the likelihood function, one focuses more in the probability of the sample we actually observed under various possible values of θ, the function parameter [2].

Let $f(\mathbf{x} \mid \theta)$ denote the joint pdf or pmf of the sample $\mathbf{X} = (X_1, \ldots, X_n)$. Then, given that $\mathbf{X} = \mathbf{x}$ is observed, the function of θ defined by

$$L(\theta \mid \mathbf{x}) = f(\mathbf{x} \mid \theta)$$

is called the likelihood function. Note the difference in the order of notation for θ and x.

While the likelihood function will be absolutely a necessary concept throughout the whole master's program, if you are particularly interested in the mathematical statistics and the deeper understanding of the purely statistical concepts, we recommend you to follow the Advanced Statistical Inference course, that will give you very deep insights and concepts you will probably not see anywhere else. You can also read [2] or [9].

Maximum likelihood computation.

Step 1: determine what pdf X follows

Step 2: determine the pdf for $X_1, \cdots, X_n$. Generally speaking, the likelihood is the product of pdf's of $X_1, \cdots, X_n$. If X are iid (independent identically distributed), then the likelihood function can be written as

$$\begin{aligned} &L\left(\theta \mid x_1, \ldots, x_n\right) = \\ &= \begin{cases} \prod_{i=1}^{n} f_\theta\left(x_i\right) = f_\theta\left(x_1\right) \cdots f_\theta\left(x_n\right) & X_i \text{ continuos} \\ \prod_{i=1}^{n} P_\theta\left(X_i = x_i\right) = P_\theta\left(X_1 = x_1\right) \cdots P_\theta\left(X_n = x_n\right) & X_i \text{ discrete} \end{cases} \end{aligned}$$

Step 3: take the log of the likelihood function. The reason we take the log is because it's easier to work with sums rather than products.

$$\begin{aligned} &\ell\left(\theta \mid x_1, \ldots, x_n\right) = \log\left(L\left(\theta \mid x_1, \ldots, x_n\right)\right) = \\ &= \begin{cases} \sum_{i=1}^{n} \log\left(f_\theta\left(x_i\right)\right) & X_i \text{ continous} \\ \sum_{i=1}^{n} \log\left(P_\theta\left(X_i = x_i\right)\right) & X_i \text{ discrete} \end{cases} \end{aligned}$$

Step 4: take the FOC, first order condition and solve it for 0.

$$\frac{\partial \ell\left(\theta \mid x_1, \ldots, x_n\right)}{\partial \theta} = 0 = \begin{cases} \sum_{i=1}^{n} \frac{\partial \log f_\theta(x_i)}{\partial \theta_i} & X_i \text{ continuous} \\ \sum_{i=1}^{n} \frac{\partial \log \theta_\theta(X_i = x_i)}{\partial \theta} & X_i \text{ discrete} \end{cases}$$

The function $\frac{\partial \ell(\theta \mid x_1, \ldots, x_n)}{\partial \theta}$ is called the **score** function.

Step 5: take the second derivative to make sure it's a maximum.

$$\frac{\partial^2 \ell\left(\theta \mid x_1, \ldots, x_n\right)}{\partial \theta^2} \stackrel{?}{=} 0$$

A quick reminder from calculus: when a function's slope (first derivative) is zero at **x**, and the second derivative at **x** is:

1. less than 0, it is a local maximum
2. greater than 0, it is a local minimum
3. equal to 0 , then the test fails (there may be other ways of finding out though).

Thus, for a maximum likelihood function we want the second derivative to be less than zero:

$$\frac{\partial^2 \ell\left(\theta \mid x_1, \ldots, x_n\right)}{\partial \theta^2} \stackrel{?}{<} 0$$

The result of the equation in *Step 4* is the estimator $\hat{\theta}_{MLE}$ that maximizes the likelihood function with respect to θ, thus finding an estimator with the most *alike* the theoretical pdf.

The maximum likelihood estimators have a lot of nice properties and the technique is general. The results are the following:

1. $\hat{\theta}_{ML}$ can be biased for θ.
2. $\hat{\theta}_{ML}$ is consistent for θ.

There is a general distributional asymptotic result:

$$\sqrt{n}\left(\hat{\theta}_{ML} - \theta\right) \sim_{n\to\infty} N\left(0, J^{-1}(\theta)\right),$$

where $J(\theta) = E\left(\left(\frac{\partial \log(f(\theta|x_1,\ldots,x_n))}{\partial \theta}\right)^2\right)$ is called the **Fisher information**.

This implies that $\hat{\theta}_{ML} \sim_{\text{approx}} N\left(\theta, \frac{1}{n} J^{-1}(\theta)\right)$ because $E\left(\hat{\theta}_{ML}\right) \simeq \theta$ and

$$\operatorname{Var}\left(\hat{\theta}_{ML}\right) = \frac{1}{n} \operatorname{Var}\left(\sqrt{n}\left(\hat{\theta}_{ML} - \theta\right)\right) \simeq \frac{1}{n} J^{-1}(\theta).$$

Method of moments and Maximum Likelihood

For more information about method of moments, maximum likelihood, examples of computations, asymptotics and the confidence intervals based on those estimators, see Statistics lectures, Preliminary Work Week 8, Probability and Statistical Learning lectures, Section 7, Subsection 1, Section 9, Heumann and Shalabh [4] book chapter 9.3, Advanced Statistical Inference course (advanced) and Casella and Berger [2] book chapter 6 (advanced).

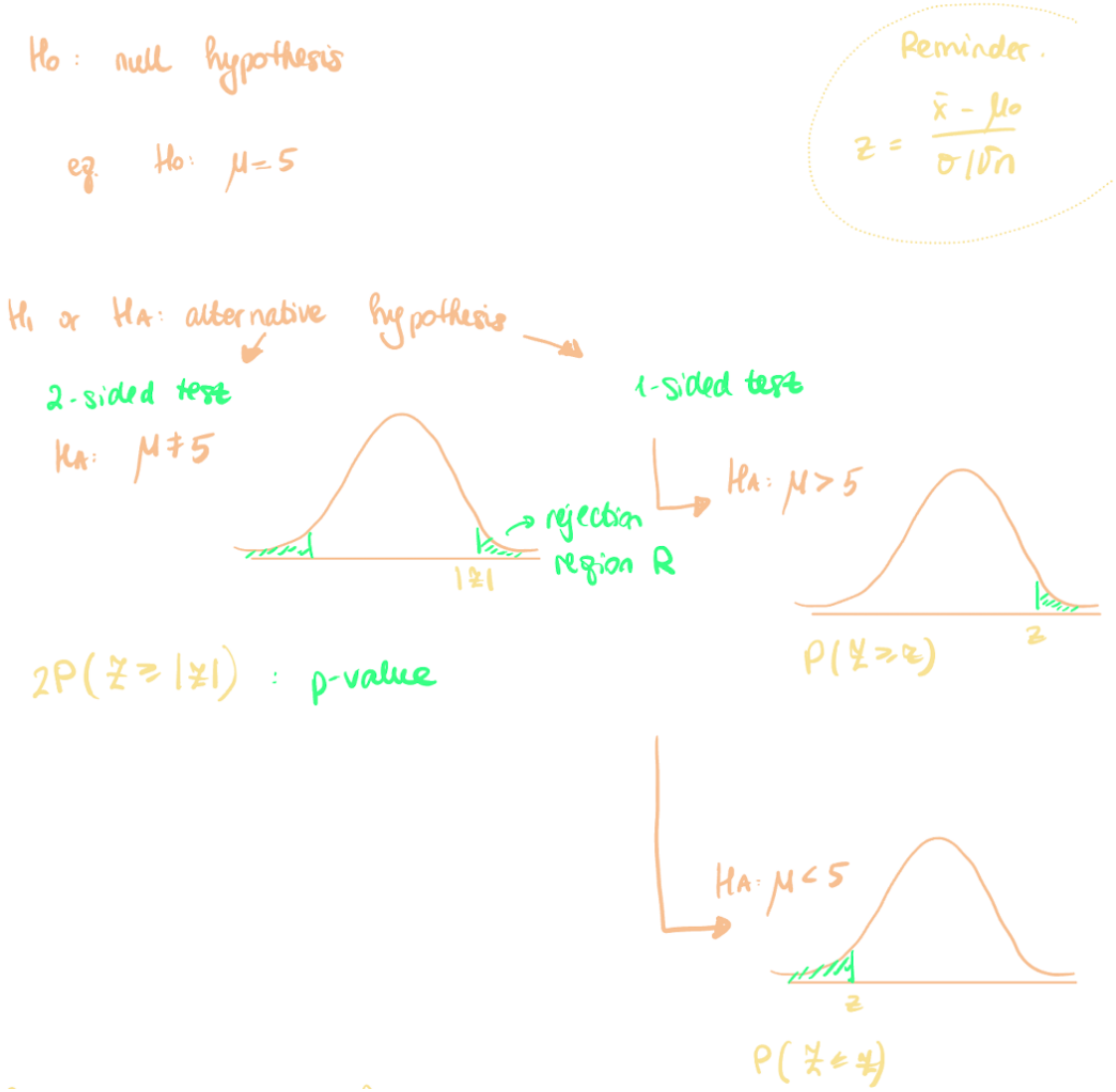
Hypothesis testing

When making decisions, a formal procedure needs to be implemented to compare the observed data with a hypothesis. This is exactly the idea behind hypothesis testing. It contains several "ingredients":

1. Random variable and its distribution,
2. Confidence level,
3. Test statistic.

Hypothesis testing can be summarized as in figure 11.

Figure 11: Hypothesis testing.

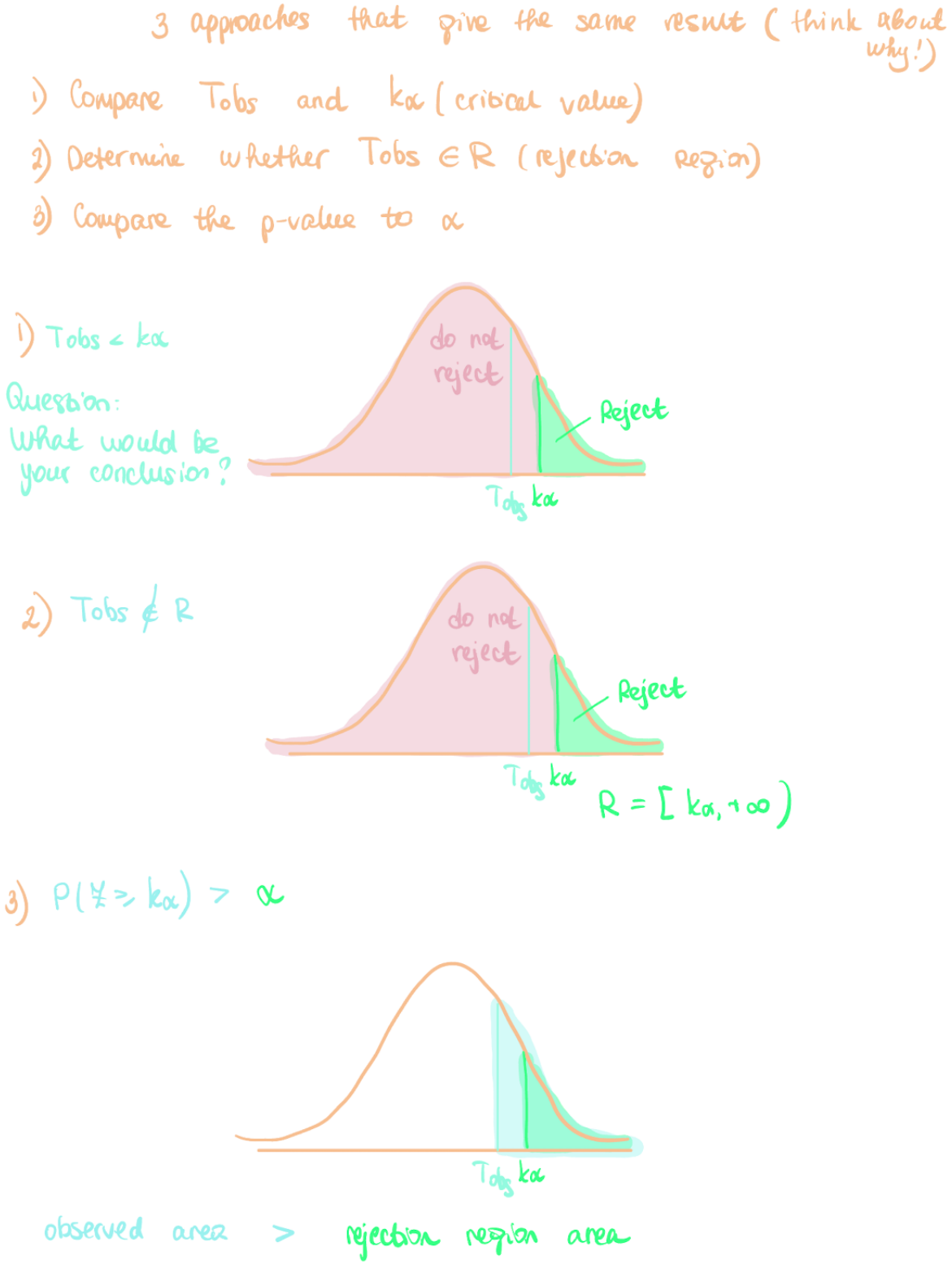


Now some definitions and tips:

- Null hypothesis, denoted H_0, is the assumed or specific value you want to test for,
- Alternative hypothesis is what you believe the value to be,
- The confidence level α is always set **before** the experiment, the most common values being $\alpha = 0.05, \alpha = 0.01, \alpha = 0.1$,
- As to the test statistic, you can take mean, proportion, difference of means or another statistic of your choice.

To determine whether to reject or not reject the H_0, you can use one of the following methods presented in figure 12. They all yield the same result.

Figure 12: Three approaches to hypothesis testing.



- In method 1, the observed value of the statistic, T_{obs}, is compared to the critical value for the specified level of α, k_α.
- In method 2, rejection region R is being constructed from k_α: $R = [k_\alpha, \infty)$.
- In method 3, $p - value$ (observed ared) is computed and compared to the rejection region area.

For the mean of the observations, assuming the normal distribution, known σ and n, $H_0 : \mu = \mu_0$ and $H_A : \mu > \mu_0$, the $p - value$ under H_0 can be computed as follows:

$$P_{H_0}(\bar{X} > \text{ t.obs }) = P_{H_0}\left(\frac{\bar{X} - \mu_0}{\sigma/\sqrt{n}} > \frac{t.obs - \mu_0}{\sigma/\sqrt{n}}\right) \tag{10}$$

Then, H_0 is rejected when $p - value \leq \alpha$. When $p - value > \alpha$, we do not have enough evidence to reject H_0.

29 TYPE I AND II ERROR

When dealing with hypothesis testing, different types of errors have to be dealt with. They can be summarized as follows.

	H_0 true	H_0 not true
Reject H_0	Type I error	Correct decision
Don't reject H_0	Correct decision	Type II error

We define probabilities associated with each situation

$$\alpha = P(\text{ reject } H_0 \mid H_0 \text{ true }) = P_{H_0}(\text{ reject } H_0)$$

and

$$\beta = P(\text{ do not reject } H_0 \mid H_0 \text{ not true }) = P_{H_A}(\text{ do not reject } H_0)\,.$$

The **power** is defined as $power = 1 - \beta = 1 - P_{H_A}(\text{ do not reject } H_0) = P_{H_A}(\text{ reject } H_0)$.
In order to increase the power of a test, following can be done:

- increase n,
- increase the distance between the two hypotheses ($(\mu - \mu_0) \uparrow$),
- increase α.

Figure ?? illustrates the power of a test.

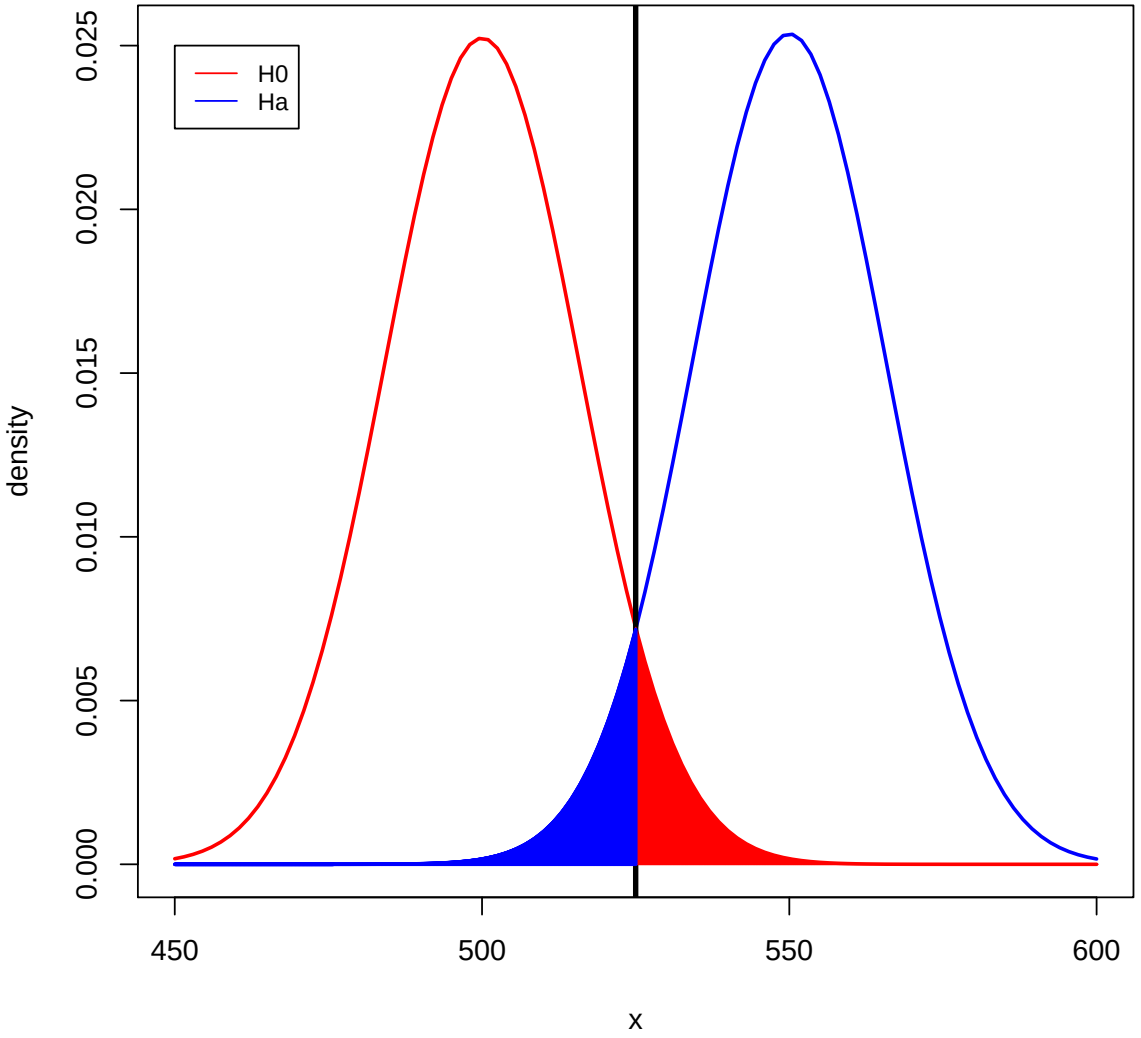


Figure 13: Power of the test.

30 COMPARING 2 GROUPS

When testing for location (mean), two groups can be compared. The idea stays the same, but the formula has to be adjusted.

Let $X_1, \ldots, X_{n_X}$ and $Y_1, \ldots, Y_{n_Y}$ two independent samples, with $E(X_i) = \mu_1$ and $E(Y_i) = \mu_2$. To test $H_0 : \mu_1 = \mu_2$ (or equivalently $H_0 : \mu_1 - \mu_2 = 0$), use

$$T = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{s_X^2}{n_X} + \frac{s_Y^2}{n_y}}} \sim_{\text{approx }, H_0} N(0,1),$$

where $s_X^2 = \frac{1}{n-1} \sum_{i=1}^{n_X} (X_i - \bar{X})^2$ and $s_Y^2 = \frac{1}{n-1} \sum_{j=1}^{n_Y} (Y_j - \bar{Y})^2$.

T can then be compared to k_α using Method 1 mentioned in section 28.2.

31 ANALYSIS OF VARIANCE (ANOVA)

When comparing more than 2 groups, it is useful to do analysis of variance test (ANOVA). A following example can be helpful. Here, we compare means of several (I) groups (samples), having J measures per group.

Model (one-way layout):

$$\underset{j\text{-measure },i\text{-group}}{Y_{ij}} = \mu_i + \varepsilon_{ij} = \underset{\text{mean effect}}{\mu} + \underset{\text{group effect}}{\alpha_i} + \underset{\text{random error}}{\varepsilon_{ij}}, \tag{11}$$

for $i = 1, \ldots, I$ et $j = 1, \ldots, J$.

We assume $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$ and require $\sum_{i=1}^{l} \alpha_i = 0$ (or $\alpha_1 = 0$, par ex.).

We test: $H_0 : \alpha_1 = \alpha_2 = \ldots = \alpha_I = 0$ (which implies $\mu_1 = \mu_2 = \ldots = \mu_I = \mu$) .

The analysis of variance is based on the following decomposition of Y_{ij}:

$$Y_{ij} = Y_{..} + (Y_{i.} - Y_{..}) + (Y_{ij} - Y_{i.})$$

with $Y_{i.} = \frac{1}{J} \sum_j Y_{ij}$ the group i mean, and $Y_{..} = \frac{1}{I.J} \sum_{i,j} Y_{ij}$ the global mean. We therefore have the following

$$\underbrace{\sum_{i,j} (Y_{ij} - Y_{..})^2}_{SS_T} = \underbrace{J \sum_i (Y_{i.} - Y_{..})^2}_{SS_B} + \underbrace{\sum_{i,j} (Y_{ij} - Y_{i.})^2}_{SS_W}, \tag{12}$$

where SS_T is the total sum of squares,
SS_B is the sum of squares between groups and
SS_W is the sum of squares within groups.

Hypothesis testing

For more information about hypothesis testing, see Statistics lectures, Preliminary Work Weeks 9-12, Probability and Statistical Learning lectures, Week 11, Subsection 2, Heumann and Shalabh [4] book chapter 10, Moore and McCabe [5] pages 660 and on, Rice [6] book chapter 9.

REFERENCES

[1] Ayyub, B. M. and McCuen, R. H. (2016). *Probability, statistics, and reliability for engineers and scientists*. CRC press.

[2] Casella, G. and Berger, R. L. (2021). *Statistical inference*. Cengage Learning.

[3] Commissions romandes de mathématique, d. p. e. d. c. (2006). *Formulaires et tables*. Editions du Tricorne.

[4] Heumann, C. and Shalabh, M. S. (2016). *Introduction to statistics and data analysis*. Springer.

[5] Moore, D. S. and McCabe, G. P. (1989). *Introduction to the Practice of Statistics*. WH Freeman/Times Books/Henry Holt & Co.

[6] Rice, J. A. (2006). *Mathematical statistics and data analysis*. Cengage Learning.

[7] Ross, S. (2014). *A First course in Probability*. Pearson New International Edition.

[8] Rozanov, Y. (1977). *Probability Theory: A Concise Course*. Dover Publications.

[9] Shao, J. (2003). *Mathematical statistics*. Springer Science & Business Media.