

Python Summary

Strings

```
In [ ]: string = "hello world!"; len(string) # assign string to variable
# string[0] = "H" # doesn't work because strings are immutable meaning they cannot be changed
string[1 : len(string)] # return string starting from index 1 to len()-1
string.count("l") # count the occurrence of str in str
string.startswith("he") # check if string start with str
string.endswith("rld!") # check if string end with str
string.replace("o", "a") # replace all occurrence of str1 with str2
# concatenate a list of string with each other and return new one " " at beginning adds spaces between str
" ".join([string, "im new", "to programming", "happy to learn", "about Python"])
string.find("wor") # search a string in string when its found it returns the beginning index of string else return -1
string.index("llo") # same as find() but it throws ValueError exception when not found
string.partition(" ") # partition a string on 3 string base on separator " " mean space
string.split("l") # partition base on string, default sep character (including \n \r \t \f and spaces)
string.capitalize(); string.lower(); string.upper() # formatting string
string.islower(); string.isupper(); string.isdigit(); string.isalpha(); # check if string is
string.isalnum(); string.isnumeric(); string.isspace()
```

Common methods & function for objects work with indexing

```
obj[start:end:step] # slicing: element from start to end-1 index with number of steps between
obj[:] # default : all element of obj one by one
obj.append(element) # add element in the end of obj
obj.insert(index, element) # insert element at index
obj.reverse() # reverse obj (start..end) -> (end..start)
obj.count(element) # return the count of element in a obj
obj.remove(element) # remove the first occurrence of an element
obj.pop(index) # Return the index'th element and delete it, default -1
obj.index(element) # return index of the first occurrence of element
obj.index(value, start_index, end_index) # index of value in (start, end) sub-search
del(obj) # destruct the object
del(obj[index]) # delete element at index
del(obj[start_index:end_index]) # delete elements from index (start_index, end_index)
del(obj[start_index:]) # del elements from the index (start, N)
del(obj[:end_index]) # del elements from the index (0, index)
len(obj) # give the length of obj
min(obj) # min element of obj
max(obj) # max element of obj
sorted(obj) # sort the obj
type(obj) # return the type obj
obj.sort(reverse = True) # sort in reverse order (max -> min) default is false
element in obj # check if element in obj return true if found else false
element not in obj # check if element not in obj return true if not found else false
new_obj = copy.copy(obj) # returns a shallow copy of object (reference the same obj)
new_obj = deep.copy(obj) # return a copy of all element of an object (2 different obj)
```

Arrays

- Ordered collection of items (Dynamic) & Holds one data type

```
In [ ]: from array import array # Lib for using array
# array("<data-type>", [data...]) # types : i, f, d, b
A = array("i", [1, 2])
B = array("f", [0.55, 3.77])
C = array("i", range(0,10))
# A[index] = value # assign value to index
sorted(C) # only sorted is available
```

Lists []

- Ordered collection of items (Dynamic) & Mutable (can be modified after creation).

```
In [ ]: L = [] # empty list
L = [20] * 20; # [20,.....20] create list of 20's 20 time
L = list(range(13, 30, 2)) # initialise list with rang seq / list() Built a mutable sequence.
L = [14, "hello", 7.99, False] # list are dynamique can hold any type
L = [x ** 2 for x in range(3, 30) if x % 3 == 0] # initialise list with mul of 3
L[4] = 333 # L[index] = value
L[0:6] = [333] # from index (0 to 6) replace with 333
L.extend([9999]*3) # extend list by adding new list of [9999,9999,9999]
L + L # same as extend => concatenate L and L
L * 3 # same as extend => concatenate L, 3 times
```

```
L = [10, 20, [300, 400, [5000, 6000], 500], 30, 40]
L[2][2][1] # accessing the list inside list every [] represent a level L[2][2][1] = 6000
```

Sets {}

- Unordered collection of unique items (Dynamic/hashable items) & Mutable (can be modified after creation), set is not indexable.

```
In [ ]: # s = set() # to declare set use set()
s1 = {1, 2, 14.7, 'py', 'oop'} # or initialise it in order for interpreter to know its a set
# type(s); type(s1);
s1.add(5) # Add an element, has no effect if the element is already present.
s1.update({4, 5, 6}) # Update a set with the union of itself and others. add elements if not exists
s1.remove(4) # Remove a specific element from the set. Raises KeyError if the element is not present.
element = s1.pop() # Remove and return an arbitrary element from the set. Raises KeyError if the set is empty.
s1.discard(0) # same as remove but it doesn't Raise KeyError
s2 = s1.copy() # Create a shallow copy of the set.
s1.clear(); # Remove all elements from the set.
s1 = {1, 2, 3, 4}; s2 = {3, 4, 5, 6}; s1.difference_update(s2) # Remove all elements of s2 from s1.
A = {1, 2, 3}; B = {2, 3, 4}
A == B # check elements appearances in each set
A != B # check elements appearances in each set
A < B # check if A ⊂ B sous-ensemble strict
A <= B # check if A ⊆ B sous-ensemble
A > B # check if A ⊃ B sur-ensemble strict
A >= B # check if A ⊇ B sur-ensemble
A & B # intersection ∩ / returns common elements that are in A and B
A | B # union ∪ / returns element that are in A and B without duplication
A - B # difference - / subtract B from A => returns elements of A
B - A # difference - / subtract A from B => returns elements of B
A ^ B # difference symetrique Δ / return items that are not in B from A & not in A from B
```

Dictionary {key : value}

- Unordered collection of key-value pairs & Mutable (You can add, remove, and modify key-value pairs), dict is not indexable.

```
In [ ]: d = {}; d = dict() # auto knows its a dict empty dict
d = dict([[1, 2], [3, 4]]) #=> {1: 2, 3: 4} / [(key, value), (key, value)...] / [[key, value], [key, value]...]
d = {"key": "value", 1: "first", 2: "second"}; # doesnt not accept unhashble data as key (list, obj..)
d["key"] = "new value"; d[1] = "first key"; # d[key] = value
d = {'name': 'John', 'age': 30, 'city': ''}
d.items() # Accessing items as key-value pairs / return all (keys , values) of dict
d.update({'gender': 'Male'}) # Updating the dictionary with new key-value pairs if not exist
d.setdefault('country', 'USA') # Setting a default value for a key if it doesn't exist
d.pop('age') # Removing and returning the value associated with a specified key
d.popitem() # Removing and returning an arbitrary (key, value) pair
d.values() # Returning a values of all keys in the dictionary
d.get('age') # Getting the value associated with a key (returns None if the key is not present)
dc = d.copy() # Creating a shallow copy of the dictionary
keys = d.keys() # Returning a view of all keys in the dictionary dict_keys / dict_keys(['name', 'age', 'city'])
key_present = 'name' in d # Checking if a key is present in the dictionary (true, false)
# Creating a new dictionary with specified keys and a default value 'Unknown'
dn = dict.fromkeys(['name', 'age', 'city'], 'Unknown')
d.clear(); dc.clear(); # Clearing all items from the dictionary
```

Tuples ()

- Ordered, immutable sequence of elements. (static List)

```
In [ ]: t = tuple(); t = (); type(t); # to declare & initialise with empty tuple
t = (2,); t = tuple([2]); # for declaring 1 element in tuple
t = tuple(["hello", "world", "hello", 14, 7, 14, {1,2,3}, {"key": "value"}])
t = (1, 2, 3, 'a', 'b', 'c')
# t[index] = value # tuple' object does not support item assignment
""" tuples support only this methods because they are static """
t.count(1); t.index('a');
t + (3,4) # adds 3,4 add the end of tuple
t * 2 # concatenate tuple 2 times
t = (1, 2, 3, 4, 5)
a, b, c = t[0], t[1], t[2:] # Unpacking: c is a list of the remaining values
a, b, *c = t # Extended unpacking using '*' c takes the rest
a, b = 1, 2 # Simultaneous assignment
a, b = b, a # Swapping values
c = a, b # Creating a tuple
```

Shallow copy & deep copy

- shallow copy does not return new copy for objects / deep copy return a copy for all elements

```
In [ ]: import copy
# shallow copy
L = [1, 2, 3]; K = list(L);
K = L[:]
K = copy.copy(L)
# deep copy
K = copy.deepcopy(L) # works with other objects too
```

Files

- operation (r: read), (w: write), (a: append) work with 1 at the time

```
In [ ]: import os
pwd = os.getcwd() # get current work dir Pwd: C:\Users\lenovo\Desktop\Jupyter_Workspace
file = open('data_files/text.txt', 'r') # open file in r, w, a mode
# file.write('\nHello World!') # for writing data with (w or a) mod
print(file.read()) # read data with (r) mode
file.read(12) # Read up to 12 characters from the file
file.readline() # Read line from the file
file.readlines() # Read all remaining lines from the current cursor position until the end of the file
file.tell() # Return the current cursor position in the file
file.seek(6) # Change the position of the cursor to byte offset 6
file.read() # Read the entire content from the current cursor position to the end of the file
file.close() # Close the file
```

Exceptions

```
In [ ]: # example of exceptions: Exception - IOError - ZeroDivisionError - IndexError - ValueError
try:
    pass # Code that may raise an exception
except Exception as e: # Activated when an exception occurs
    pass # Handle the exception
else:
    pass # Executed if no exception is raised in the try block (optionnal)
finally:
    pass # Block that always executes, whether an exception occurred or not (optionnal)
```

Database Connectivity

- By Default Python support `SQLite` first import sqlite lib with

```
In [3]: import sqlite3
connection = sqlite3.connect('data_files/db.sqlite3') # Connect to a database
cursor = connection.cursor() # Create a cursor object to execute SQL commands
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY, username varchar(50), email varchar(50), password varchar(50), phone varchar(50)
    ) ''') # Create a table
# Insert data into the table
cursor.execute("INSERT INTO users (username, email, password, phone) VALUES (?, ?, ?, ?)"
    , ('john_doe', 'john@example.com', 'john', '0601020304'))
cursor.execute("INSERT INTO users (username, email, password, phone) VALUES (?, ?, ?, ?)",
    ('jane_doe', 'jane@example.com', 'jane', '0602030405'))
# Example 1: # cursor.rowcount / return number of rows effected / rowcount is a propertie
affected_rows = cursor.rowcount; print(f"Number of rows affected: {affected_rows}")
# Example 2: fetchone() used with a cursor to retrieve the next row of a query result set.
cursor.execute("SELECT * FROM users WHERE username='john_doe'"); one_row = cursor.fetchone()
print("Data fetched using fetchone():", one_row)
# Example 3: executemany() used to insert multiple rows multiple sets of parameters in a single call
data_to_insert = [('Alice', 'alice@example.com', 'pass123', '987654321'),
    ('Bob', 'bob@example.com', 'bobpass', '876543210'),
    ('Charlie', 'charlie@example.com', 'charliepass', '765432109')]
cursor.executemany("INSERT INTO users (username, email, password, phone) VALUES (?, ?, ?, ?)", data_to_insert)
affected_rows_many = cursor.rowcount
print(f"Number of rows affected by executemany(): {affected_rows_many}")
cursor.execute("SELECT * FROM users") # Fetch data from the table
users = cursor.fetchall()
for user in users: # Display the results
    print(user)
cursor.execute("DELETE FROM users") # empty the table
cursor.execute("DROP TABLE IF EXISTS users") # drop table users
connection.commit() # Commit the changes
cursor.close(); connection.close(); # Close the cursor and connection
```

```

Number of rows affected: 1
Data fetched using fetchone(): (1, 'john_doe', 'john@example.com', 'john', '0601020304')
Number of rows affected by executemany(): 3
(1, 'john_doe', 'john@example.com', 'john', '0601020304')
(2, 'jane_doe', 'jane@example.com', 'jane', '0602030405')
(3, 'Alice', 'alice@example.com', 'pass123', '987654321')
(4, 'Bob', 'bob@example.com', 'bobpass', '876543210')
(5, 'Charlie', 'charlie@example.com', 'charliepass', '765432109')

```

Overloading of function

- Unlike some other languages (like C++ or Java), Python doesn't support traditional function overloading where you can define multiple functions with the same name but different parameter types. However, you can achieve similar functionality by using **default parameter values or variable-length argument lists**.

```

In [1]: def add_numbers(a, b=0, c=0):
        return a + b + c
result1 = add_numbers(5); result2 = add_numbers(5, 3); result3 = add_numbers(5, 3, 2) # Test the function
print(result1, result2, result3)
def add_numbers(*args):
    return sum(args)
result1 = add_numbers(5); result2 = add_numbers(5, 3); result3 = add_numbers(5, 3, 2) # Test the function
print(result1, result2, result3)

```

5 8 10

5 8 10

OOP

- oop in python is adapted for dynamic typing

```

In [2]: class Point:
        obj_nb = 0      # static variable
        def __init__(self, x = 0, y = 0):      # constructor of class
            """ default & parameters create object
            x : public
            _x : protected
            __x : private
            """

            # variable self is self pointer
            self.__x = x
            self.__y = y
            self.__class__.obj_nb += 1

        def __del__(self):
            """ delete object """
            self.__class__.obj_nb -= 1

        def GetNumberObjs(self):      # Get static variable
            return self.__class__.obj_nb

        # attributes getters & setters
        @property
        def x(self):
            return self.__x

        @x.setter
        def x(self, value):
            self.__x = value

        @property
        def y(self):
            return self.__y

        @y.setter
        def y(self, value):
            self.__y = value

        def prompt(self):      # prompt the value of point
            print(f"P({self.__x},{self.__y})")

        # operators of class + - * / ...
        def __add__(self, p):      # Define operation p1 + p2
            return Point(self.__x + p.__x, self.__y + p.__y)

        def __sub__(self, p):      # Define operation p1 - p2
            return Point(self.__x - p.__x, self.__y - p.__y)

        def __str__(self): # Define toString
            return f"Value of Point is P({self.__x},{self.__y})"

```



```

p1 = Point(); p2 = Point(5,1); # create instance of class
p1.prompt() # print the values of point
print(p1.GetNumberObjs()) # get the static variable
print(f"P : x = {p1.x}, y = {p1.y}") # get attriutes with method of @property
p2.x = 5; p2.y = 7; # set attributes of @attr.setter
p2.prompt()
# operation on object
print(f"Add: {p1 + p2}"); print(f"Sub: {p1 - p2}"); print(f"P1: {p1.__str__()}\\nP2: {p2.__str__()}");

class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass # This method will be overridden in subclasses

class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

    def fetch(self):
        return f"{self.name} is fetching a ball."

class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"

    def scratch(self):
        return f"{self.name} is scratching."

# Create instances of subclasses
dog = Dog("Buddy"); cat = Cat("Whiskers");
# Call methods inherited from the base class
print(dog.name) # Access attribute from the base class
print(dog.speak()) # Call overridden method in the subclass
print(cat.name) # Access attribute from the base class
print(cat.speak()) # Call overridden method in the subclass
# Call methods specific to each subclass
print(dog.fetch()) # Call method unique to Dog class
print(cat.scratch()) # Call method unique to Cat class

```

```

P(0,0)
2
P : x = 0, y = 0
P(5,7)
Add: Value of Point is P(5,7)
Sub: Value of Point is P(-5,-7)
P1: Value of Point is P(0,0)
P2: Value of Point is P(5,7)
Buddy
Buddy says Woof!
Whiskers
Whiskers says Meow!
Buddy is fetching a ball.
Whiskers is scratching.

```

Lib Numpy (Matrix)

```

In [ ]: import numpy as np
x = np.array([[1,4,5,9],[2,5,9,7],[14,5,9,7]]) # matrix of (L1, L2, L3) L: line/row
y = np.random.randint(0, 100, (5,5)) # create matrix of (5,5) from values 0 =< n =<100

y.sum(); y.mean(); y.max(); y.min(); y.transpose();
print(y.shape) # shape (number of rows, number of cols)
print(y.ndim) # dimansion of matrix (1D: array, 2D: table-matrix, 3D)..
print(y.dtype) # data type

# prompt from values of matrix from [index_rows, index_cols] or [i, j]
print(y[-5:,]) # returns matrix starting from -5 of last row and all cols
print(y[0:2,2:4]) # from row (0 to 1) & col (2 to 3))

print(f"{y*2}") # multiply for every element with 2
print(f"{y+2}") # add for every element with 2
print(f"{y/2.0}") # divide for every element with 2

print(np.zeros((3,3))) # create matrix of (3,3) filled with 0's
print(np.ones((3,3))) # create matrix of (3,3) filled with 1's

print(np.random.rand(2)) # Create a 1x2 matrix filled with values between 0 and 1 (uniform distribution).
print(np.random.randn(2)) # Create a 1x2 matrix filled with values from a standard normal distribution.
print(np.random.rand(2,2)) # Create a 2x2 matrix filled with values between 0 and 1 (uniform distribution).
print(np.random.randn(5,3)) # Create a 5x3 matrix filled with values from a standard normal distribution.
print(np.random.randint(-100,1000,(5,5))) # Create a 5x5 matrix filled with integer values between -100 and 100.

```

```

print(np.arange(1,11,1)) # create an array stating from 1 to 10 with 1 step similar to for i in range(1,11,1)
print(np.arange(1,10.5,0.5)) # create an array stating from 1 to 10.5 with 0.5 step (1..10)
print(np.eye(5)) # create matrix of 5 x 5 with diag of 1's
print(np.eye(3)) # create matrix of 3 x 3 with diag of 1's

x = np.random.randint(0, 100, (3,3)); y = np.random.randint(0, 100, (3,3));
print(x); print(y);
# return the diagonal of matrix depending the starting col (matix, index_col)
print(np.diag(x, 0)); print(np.diag(x, 1)); print(np.diag(x, 2));
print(np.dot(x, y)) # Perform the dot:mutiplication of matrices A and B.
print(np.outer(x, y)) # Compute the outer product of vectors A and B to create a 2D array of shape (len(A), len(B)).
# The np.reshape() method in NumPy is used to change the shape of an array
# while keeping the same data. It takes an array T and a new shape as its arguments
T = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print(np.reshape(T, (2, 4))) # This will reshape the 1D array T into a 2D array with shape (2, 4).
# >>> np.reshape(a, (2, 3)) # C-like index ordering
# array([[0, 1, 2],
#        [3, 4, 5]])

# Concatenate matrices A and B along a specified axis (axis parameter should be specified).
# For example, to concatenate along rows (vertical stacking), use axis=0.
# To concatenate along columns (horizontal stacking), use axis=1.
result_concatenate = np.concatenate((x, y), 0) # 0-> y under x / 1-> y on the right of x
print(result_concatenate)

```

Lib Pandas

```

In [ ]: import pandas as pd
# df = pd.read_excel('your_file.xlsx') # Read data from an Excel file
# df = pd.read_csv("data_files\heart.csv",sep=';') # Read data from a CSV file
# df = pd.read_table('data_files\heart.txt', sep='\t', header=0) # Read data from file where data is formatted as table
sample_data = np.array([[1.1, 2, 3.3, 4], [2.7, 10, 5.4, 7], [5.3, 9, 1.5, 15]])
# df created as 3 row/index (a1,a2,a3) & 4 cols (A,B,C,D) & data (numpy matrix) in between like bellow
df = pd.DataFrame(sample_data, index = ['a1', 'a2', 'a3'], columns = ['A', 'B', 'C', 'D'])
# #      A      B      C      D
# # a1  1.1    2.0    3.3    4.0
# # a2  2.7   10.0    5.4    7.0
# # a3  5.3    9.0    1.5   15.0

print(df.index) # returns all indexes/row names
print(df.columns) # returns all cols names
print(df.info()) # return meta-data of dataframe
""" describe() renvoie un dataframe donnant des statistiques sur les valeurs (nombres de valeurs, moyenne, écart-type, ...),
mais uniquement sur les colonnes numériques (faire df.describe(include = 'all') pour avoir toutes les colonnes) """
print(df.describe()) # operations on numeric values
print(df.describe(include='all')) # opertion on all values

print(df['A']['a2']) # get intersection of (A, a2) = 2.7
# head(number_lines), tail(number_lines), default number of lines is 5
print(df.head(3)) # get the first 3 lines of data
print(df.tail(2)) # get the last 2 lines of data

""" indexing data frame for easy access: On peut accéder aux valeurs du DataFrame via des indices ou plages d'indice.
La structure se comporte alors comme une matrice. La cellule en haut et à gauche est de coordonnées (0,0).
Il y a différentes manières de le faire, l'utilisation de .iloc[,] constitue une des solutions les plus simples.
N'oublions pas que Shape permet d'obtenir les dimensions (lignes et colonnes)
du DataFrame / .iloc[range(start,end,step),range(start,end,step)] """
# # iloc[(rows, cols)]
print(df.iloc[2,2]) # get data from intersection of [row: 2, col: 2] = 1.5
print(df.iloc[1:,3:]) # get data from intersection of [rows(1..end) , col: (3..end)]
print(df.iloc[1:,[0,1,3]]) # get data from intersection of [rows(1..end) , col: (0,1,3)]

""" Restrictions avec les conditions - Les requêtes Nous pouvons isoler les sous-ensembles d'observations répondant
à des critères définis sur les champs. Nous utiliserons préférentiellement la méthode .loc[,] dans ce cadre."""
print(df.loc[df['A']==2.7,:]) # similar to: select * from dataframe where A = 2.7
print(df.loc[df['A']!=2.7,:]) # similar to: select * from dataframe where A != 2.7
print(df.loc[df['A']==2.7,:].value_counts()) # similar to: select count(*) from dataframe where A = 2.7
print(df.loc[df['A'].isin([1.1,5.3]),:]) # select * from dataframe where A in (1.1,5.3)
# # Des opérateurs logiques permettent de combiner les conditions: & pour ET, | pour OU, ~ pour la négation.
print(df.loc[(df['A']==2.7) | (df['B'] == 9),:]) # select * from dataframe where A = 2.7 or B = 9
print(df.loc[(df['A'] > 1) & (df['B'] >= 2) & (df['C'] > 4),:]) # select * from dataframe where A > 1 AND B >= 2 AND C > 4

```

Graphical User Interface (Tkinter)

- toolkit for creating desktop applications in Python. It provides a set of tools for building graphical user interfaces and is widely used due to its simplicity and ease of use.

```

In [ ]: # creates GUI window with three check buttons for options (Etudiant, Master BABD M1, Master MSI M1).
from tkinter import *
fenetre = Tk() # Create a Tkinter window

```

```

CheckVar1 = StringVar(); CheckVar2 = StringVar(); CheckVar3 = StringVar() # Create StringVar variables for check buttons
# Create check buttons with specified text and variables
bouton1 = Checkbutton(fenetre, text="Etudiant", variable=CheckVar1, onvalue="oui", offvalue="non").pack()
bouton2 = Checkbutton(fenetre, text="Master BABD M1", variable=CheckVar2, onvalue="oui", offvalue="non").pack()
bouton3 = Checkbutton(fenetre, text="Master MSI M1", variable=CheckVar3, onvalue="oui", offvalue="non").pack()
def fonction1(): # Define a function and create a button to display the selected options
    ch = "Tu es"
    if(CheckVar1.get()=="oui"):ch+=" Etudiant"
    if(CheckVar2.get()=="oui"):ch+=" En Master DSBD M1 "
    if(CheckVar3.get()=="oui"):ch+=" En Master MSI M1 "
    label2 = Label(fenetre, text=ch); label2.pack()
bouton4=Button(fenetre, text="Afficher", command=fonction1); bouton4.pack()
fenetre.mainloop() # Start the Tkinter event Loop

```

```

In [ ]: # window with three radio buttons ("Yes", "No", "Maybe") grouped together. Only one button can be selected at a time.
from tkinter import * # Import the Tkinter library
fenetre = Tk() # Create the main window
label = Label(fenetre, text="do you accept!"); label.pack()
value = StringVar() # Variable to store the selected value in the radio button group
# Create radio buttons
button1 = Radiobutton(fenetre, text="Yes", variable=value, value=1)
button2 = Radiobutton(fenetre, text="No", variable=value, value=2)
button3 = Radiobutton(fenetre, text="Maybe", variable=value, value=3)
button1.pack(); button2.pack(); button3.pack() # Display radio buttons in the window
fenetre.mainloop() # Start the main window loop

```

```

In [ ]: import tkinter as tk
from tkinter import messagebox
import sqlite3
conn = sqlite3.connect('data_files/app.sqlite3'); cursor = conn.cursor() # Create a connection to the SQLite database
cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT NOT NULL, age INTEGER)''')
conn.commit() # Create a table if it doesn't exist

def insert_data(): # Function to insert data into the database
    name = name_entry.get(); age = age_entry.get()
    if not name or not age.isdigit(): # Validate input
        messagebox.showerror("Error", "Invalid input"); return
    cursor.execute('INSERT INTO users (name, age) VALUES (?, ?)', (name, age)); conn.commit()
    messagebox.showinfo("Success", "Data inserted successfully"); fetch_data()

def fetch_data(): # Function to fetch data from the database
    cursor.execute('SELECT * FROM users'); data = cursor.fetchall()
    message = "ID\tName\tAge\n"
    for row in data: # Display data in a message box
        message += f"{row[0]}\t{row[1]}\t{row[2]}\n"
    messagebox.showinfo("Data", message)

def update_data(): # Function to update data in the database
    selected_id = id_entry.get(); name = name_entry.get(); age = age_entry.get()
    if not selected_id.isdigit() or not name or not age.isdigit(): # Validate input
        messagebox.showerror("Error", "Invalid input"); return
    cursor.execute('UPDATE users SET name=?, age=? WHERE id=?', (name, age, selected_id)); conn.commit()
    messagebox.showinfo("Success", "Data updated successfully"); fetch_data()

def delete_data(): # Function to delete data from the database
    selected_id = id_entry.get()
    if not selected_id.isdigit(): # Validate input
        messagebox.showerror("Error", "Invalid input"); return
    cursor.execute('DELETE FROM users WHERE id=?', (selected_id,)); conn.commit()
    messagebox.showinfo("Success", "Data deleted successfully"); fetch_data()

app = tk.Tk(); app.title("Database GUI") # Create the main application window
# Create and place widgets (labels, entry fields, buttons)
id_label = tk.Label(app, text="ID:"); id_label.grid(row=0, column=0, padx=10, pady=10)
id_entry = tk.Entry(app); id_entry.grid(row=0, column=1, padx=10, pady=10)
name_label = tk.Label(app, text="Name:"); name_label.grid(row=1, column=0, padx=10, pady=10)
name_entry = tk.Entry(app); name_entry.grid(row=1, column=1, padx=10, pady=10)
age_label = tk.Label(app, text="Age:"); age_label.grid(row=2, column=0, padx=10, pady=10)
age_entry = tk.Entry(app); age_entry.grid(row=2, column=1, padx=10, pady=10)
insert_button = tk.Button(app, text="Insert Data", command=insert_data)
insert_button.grid(row=3, column=0, columnspan=2, pady=10)
fetch_button = tk.Button(app, text="Fetch Data", command=fetch_data)
fetch_button.grid(row=4, column=0, columnspan=2, pady=10)
update_button = tk.Button(app, text="Update Data", command=update_data)
update_button.grid(row=5, column=0, columnspan=2, pady=10)
delete_button = tk.Button(app, text="Delete Data", command=delete_data)
delete_button.grid(row=6, column=0, columnspan=2, pady=10)
app.mainloop() # Start the Tkinter event Loop
conn.close() # Close the database connection when the application is closed

```

```

In [ ]: from tkinter import *
import sqlite3

```



```

def validate():
    name = entryName.get(); email = entryEmail.get(); age = entryAge.get() # Retrieving form data
    conn = sqlite3.connect('data_files/app.sqlite3'); cur = conn.cursor() # Connecting to the SQLite database
    # Creating the "students" table if it doesn't exist
    cur.execute("""CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY AUTOINCREMENT,
                                                         name TEXT NOT NULL, email TEXT NOT NULL, age INTEGER NOT NULL)"""); conn.commit()
    # Inserting data into the table
    cur.execute("INSERT INTO students (name, email, age) values (?, ?, ?)", (name, email, age))
    conn.commit(); conn.close()

def display_data():
    # Connecting to the SQLite database
    conn = sqlite3.connect('data_files/app.sqlite3'); cur = conn.cursor()
    result = cur.execute("SELECT * FROM students") # Retrieving and displaying data from the "students" table
    for row in result:
        print("ID:", row[0]); print("Name:", row[1]); print("Email:", row[2]); print("Age:", row[3]); print("-----")
    conn.close() # Closing the database connection

root = Tk(); root.geometry("600x400") # Creating the main window
# Creating a form to insert data
lblName = Label(root, text="Name : "); lblName.place(x=10, y=10)
entryName = Entry(root); entryName.place(x=100, y=10, width=200)
lblEmail = Label(root, text="Email"); lblEmail.place(x=10, y=40)
entryEmail = Entry(root); entryEmail.place(x=100, y=40, width=200)
lblAge = Label(root, text="Age"); lblAge.place(x=10, y=70)
entryAge = Entry(root); entryAge.place(x=100, y=70, width=200)
# buttons 1 for validation of input / button 2 for displaying data in terminal
btnValidate = Button(root, text="Validate", command=validate)
btnValidate.place(x=100, y=100, width=200, height=25)
btnDisplayData = Button(root, text="Display Data", command=display_data)
btnDisplayData.place(x=100, y=130, width=200, height=25)
root.mainloop() # Starting the main loop of the graphical interface
""" # example of output (displaying data)
ID: 1
Name: abdo
Email: a@gmail.com
Age: 14
----- """

```

Machine learning

- Types d'apprentissage
 - Apprentissage supervisé
 - Apprentissage non supervisé
- Types de prédiction de données
 - Régression : pour prédire des valeurs numériques
 - Classification : pour prédire des états ou catégories
 - Clustering : pour prédire des groupes d'éléments
- Types de solutions
 - Régression linéaire (univariée)
 - RL (univariée) avec descente de gradient
 - Régression polynomiale (no-relationnelle)
 - Régression logistique (pour classification)
 - Arbre de décision pour classification & régression
 - KNN: k Nearest Neighbors pour classification é régression (sklearn)

l'apprentissage supervisé traite des données étiquetées et vise à apprendre une cartographie des entrées aux sorties, tandis que

l'apprentissage non supervisé traite des données non étiquetées pour explorer des modèles, des structures ou des relations cachées au sein des données. Les deux paradigmes jouent un rôle crucial dans les applications d'apprentissage automatique.

Régression Lineare univariée

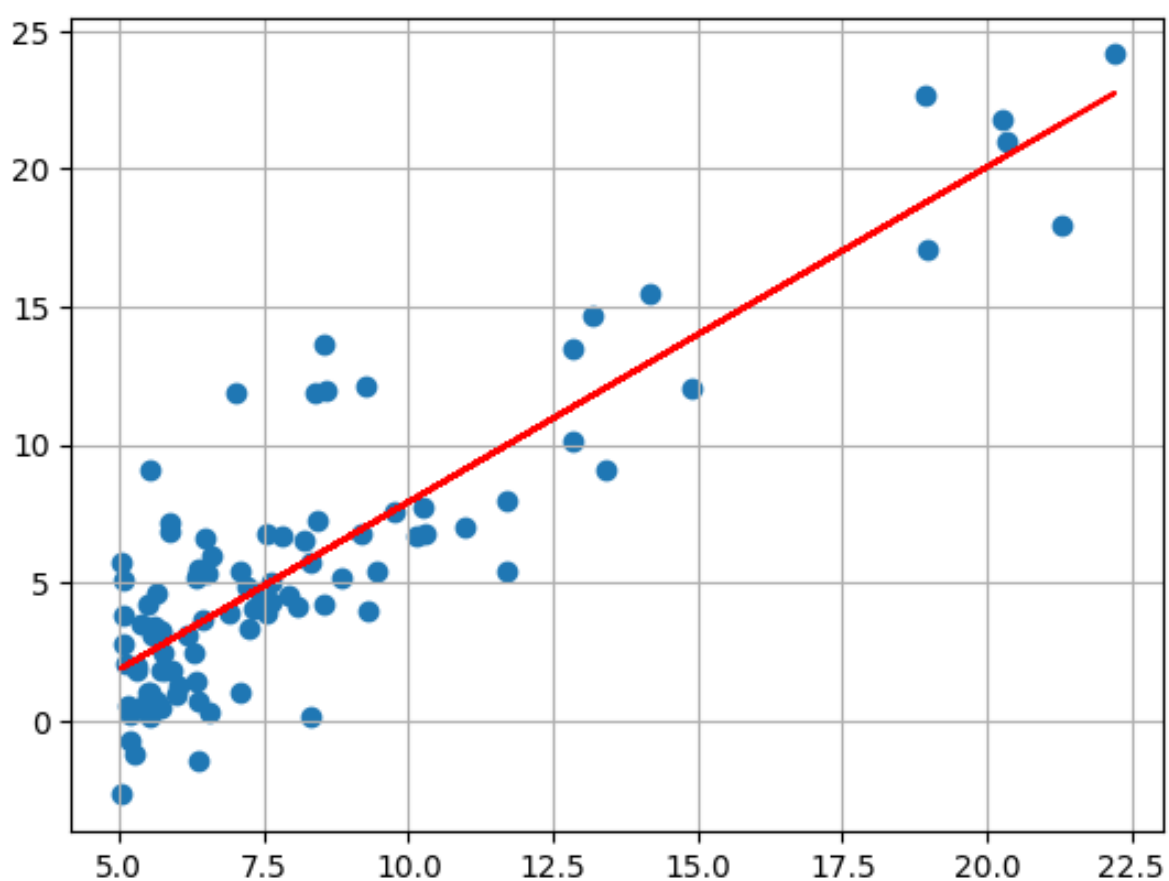
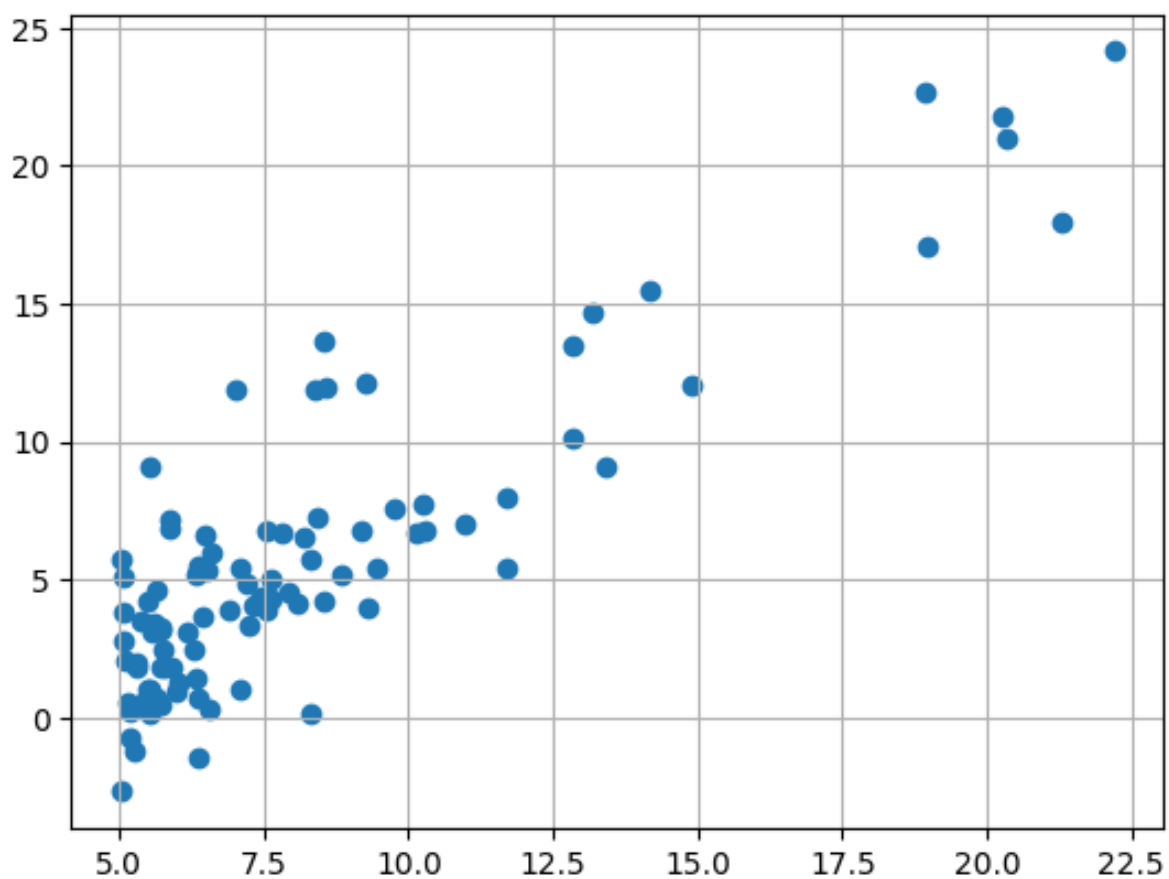
- Ce code lit un fichier CSV contenant un ensemble de données, effectue une régression linéaire univariée et visualise les résultats via des nuages de points et des lignes de régression.
- Le fonction de prédiction pour une régression linéaire univariée est comme suit : $H(x)=intercept+slope*x \rightarrow f(x) = A + Bx$
 - *slope*: représente la "pente" de la line de prédiction et
 - *intercept* représente le "point" d'intersection avec l'axe des ordonnées

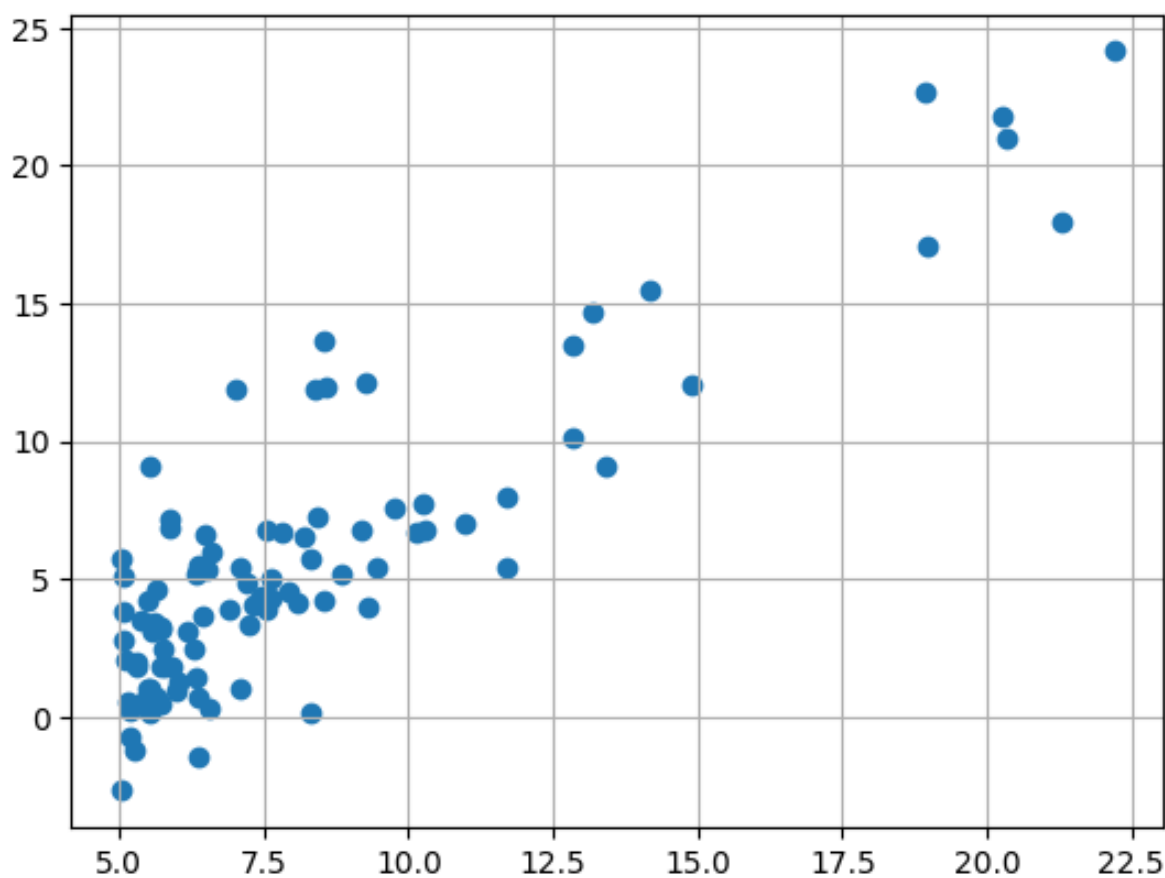
In [6]: `# Importation des bibliothèques nécessaires`
`import pandas as pd # Importation de la bibliothèque Pandas pour la manipulation des données`
`import matplotlib.pyplot as plt # Importation de Matplotlib pour le tracé`


```

from scipy import stats # Importation du module stats de SciPy pour les fonctions statistiques
# Lire l'ensemble de données dans un DataFrame
df = pd.read_csv("C:/Users/lenovo/Desktop/Jupyter_Workspace/ML/univariate_linear_regression_dataset.csv")
# Lire un fichier CSV dans un DataFrame Pandas. Ajustez le chemin du fichier en conséquence.
# Extraire la variable indépendante/prédictive (X) et la variable dépendante/cible (Y) du DataFrame  $f(x) = y$ 
X = df.iloc[:, 0] # Extraire la première colonne comme variable indépendante/prédictive (X)
Y = df.iloc[:, 1] # Extraire la deuxième colonne comme variable dépendante/cible (Y)
# Créer un nuage de points des données placement selon les valeurs
axes = plt.axes() # Créer un objet d'axes pour le tracé
axes.grid() # Ajouter des lignes de grille au tracé
plt.scatter(X, Y) # Nuage de points de X et Y
plt.show() # Afficher le tracé
# Calculer les statistiques de régression linéaire en utilisant scipy.stats.linregress
SL = stats.linregress(X, Y) # Effectuer une régression linéaire  $f(x) = A + Bx$ 
slope = SL.slope # Extraire B la pente de la ligne de régression
intercept = SL.intercept # Extraire A l'ordonnée à l'origine de la ligne de régression
coef_correlation = SL.rvalue # Extraire le coefficient de corrélation
# Définir une fonction pour prédire les valeurs de Y en fonction du modèle de régression linéaire
def predict(x): #  $f(x) = a + bx$ 
    return slope * x + intercept
# Définir une fonction de prédiction de régression linéaire simple
# Tracer les données originales avec la ligne de régression linéaire
axes = plt.axes() # Créer un autre objet d'axes pour le tracé
axes.grid() # Ajouter des lignes de grille au tracé / Grids for graphe
plt.scatter(X, Y) # Nuage de points de X et Y
fitLine = predict(X) # Valeurs prédites en utilisant le modèle de régression
plt.plot(X, fitLine, c='r') # Tracer la ligne de régression en rouge
plt.show() # Afficher le tracé avec la ligne de régression
# Afficher uniquement le nuage de points sans la ligne de régression linéaire
axes = plt.axes() # Créer un autre objet d'axes pour le tracé
axes.grid() # Ajouter des lignes de grille au tracé
plt.scatter(X, Y) # Nuage de points de X et Y
plt.show() # Afficher uniquement le nuage de points

```





Régression Lineaire univariée avec descente de gradient

- $X = \begin{pmatrix} x & 1 \end{pmatrix}$; $\theta = \begin{pmatrix} a \\ b \end{pmatrix}$ et $f(x) = \theta \cdot X = \begin{pmatrix} x & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = a \cdot x + b$
- Objectif de trouver θ qui donne une bonne prédiction d'une entrée x .



No description has been provided for this image



No description has been provided for this image

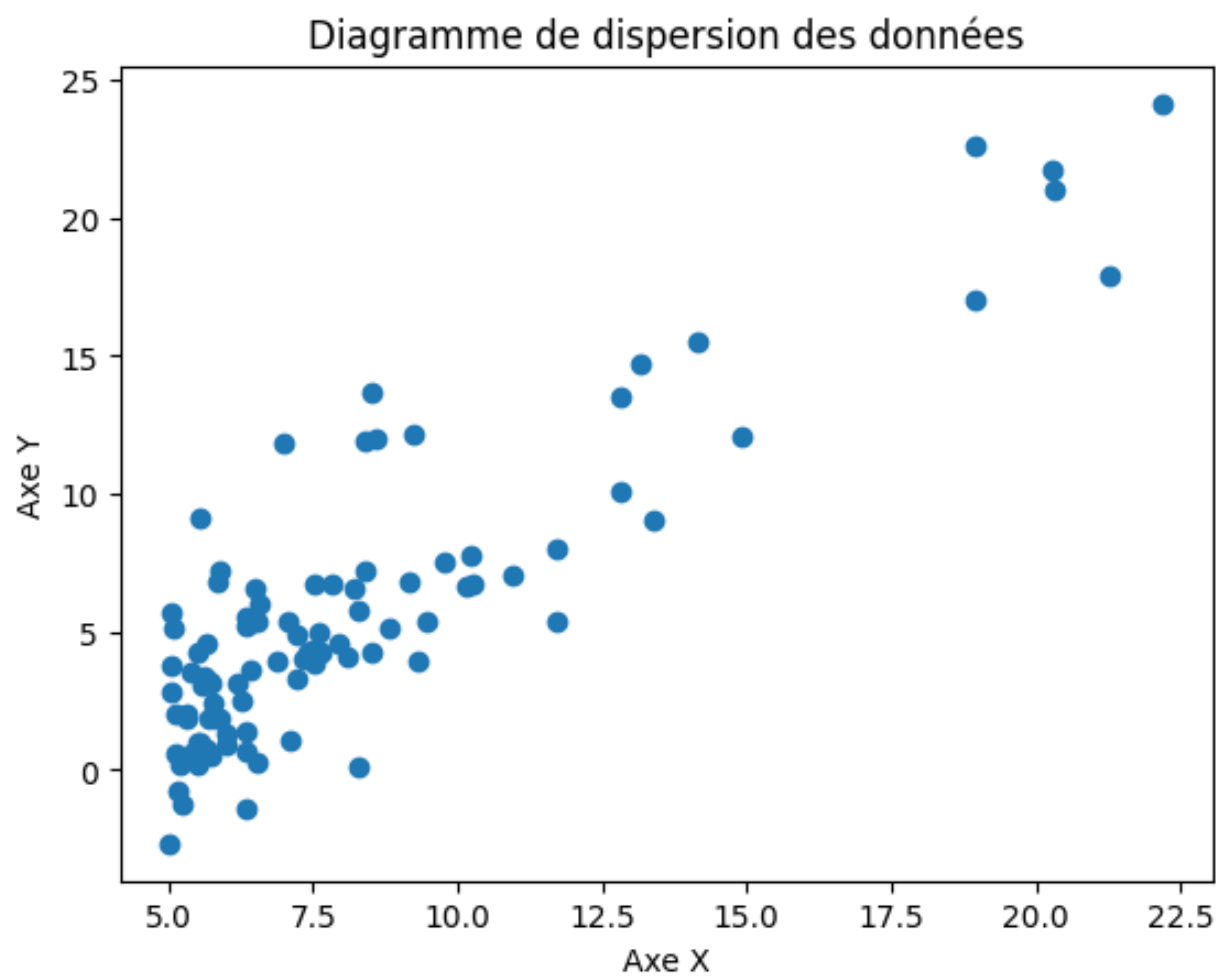
```
In [8]: # Importer Les bibliothèques nécessaires
import numpy as np # Importer NumPy pour Les opérations numériques
import pandas as pd # Importer Pandas pour La manipulation des données
import matplotlib.pyplot as plt # Importer Matplotlib pour Les graphiques

# Charger L'ensemble de données
df = pd.read_csv("data_files/univariate_linear_regression_dataset.csv")
# Extraire Les variables indépendantes et La variable dépendante f(x) = y
x = df.iloc[:, 0].values.reshape((-1, 1)) # Extraire La variable indépendante (variable explicative) et remodeler
y = df.iloc[:, 1].values.reshape((-1, 1)) # Extraire La variable dépendante (variable cible) et remodeler
""" `.values` les convertit en tableaux NumPy. Le `.reshape((-1, 1))`
est utilisé pour remodeler les tableaux en vecteurs de colonnes. """
# Tracer Le graphique des données originales
plt.scatter(x, y) # Créer un graphique de dispersion des données originales
plt.xlabel("Axe X") # Libellé de l'axe x
plt.ylabel("Axe Y") # Libellé de l'axe y
plt.title("Diagramme de dispersion des données") # Titre du graphique
plt.show() # Afficher Le graphique
# Ajouter une colonne de biais à X définie X = (x, 1)
# Ajouter une colonne de une's pour représenter Le biais dans Les variables explicatives
X = np.hstack((x, np.ones_like(x)))
# Initialiser theta avec des valeurs aléatoires, θ = (a, b)
theta = np.random.randn(2, 1) # Initialiser Les paramètres avec des valeurs aléatoires
# Définir La fonction hypothèse, f(X, θ) = X * θ = (x, 1)*(a, b) = ax + b
# Cette fonction définit L'hypothèse ou La fonction de prédiction pour La régression linéaire.
def hypothese(X, theta): # f(X, θ) = hypothese(X, theta)
    return X.dot(theta)
# Définir La fonction d'erreur quadratique moyenne mesure (Les prédictions du modèle correspondent aux valeurs réelles)
def erreur_quadratique_moyenne(X, y, theta):
    m = len(y)
    return (1 / (2 * m)) * np.sum((hypothese(X, theta) - y) ** 2, axis=0) # comme σ standard deviation
# Définir La fonction du gradient Il est utilisé pour mettre à jour Les coefficients.
def gradient(X, y, theta):
    m = len(y)
    return 1 / m * X.T.dot(hypothese(X, theta) - y)
""" Définir la fonction de descente de gradient, effectue l'optimisation, Il met à jour de manière itérative
les coefficients en utilisant le gradient de l'erreur et stocke les valeurs d'erreur dans l'historique des erreurs. """
def descente_gradient(X, y, theta, taux_apprentissage, n_iterations):
    historique_erreur = np.zeros(n_iterations)
    for i in range(n_iterations):
        theta = theta - taux_apprentissage * gradient(X, y, theta)
        historique_erreur[i] = erreur_quadratique_moyenne(X, y, theta)
    return theta, historique_erreur
# test algorithme en l'exécutant avec un nombre d'itérations et un taux d'apprentissage spécifiés.
n_iterations = 10000
taux_apprentissage = 0.01
theta_final, historique_erreur = descente_gradient(X, y, theta, taux_apprentissage, n_iterations)
```

```

# Afficher Le theta final
print("Theta Final:", theta_final)
# Tracer Les prédictions par rapport aux données originales
predictions = hypothese(X, theta_final)
plt.scatter(x, y, label='Données Originales')
plt.plot(x, predictions, color='red', label='Régression Linéaire')
plt.xlabel("Axe X")
plt.ylabel("Axe Y")
plt.title("Ajustement de la Régression Linéaire")
plt.legend()
plt.show()
""" Cette section trace l'historique de l'erreur quadratique moyenne au cours des itérations,
montrant comment l'erreur diminue à mesure que le modèle est entraîné. """
plt.figure()
plt.plot(range(n_iterations), historique_erreur)
plt.xlabel("Itérations")
plt.ylabel("Erreur Quadratique Moyenne")
plt.title("Convergence de la Descente de Gradient")
plt.show()
""" ce bloc teste le modèle entraîné en effectuant une prédiction pour une nouvelle entrée et imprime le résultat.
L'entrée [22.5, 1] est remodelée pour correspondre au format d'entrée attendu par le modèle."""
nouvelle_entree = np.array([22.5, 1]).reshape((1, -1)) # X = (x , 1) = (value, 1)
prediction = hypothese(nouvelle_entree, theta_final) # f(X, θ) = y
print("Prédiction pour [22.5, 1]:", prediction) # value of y

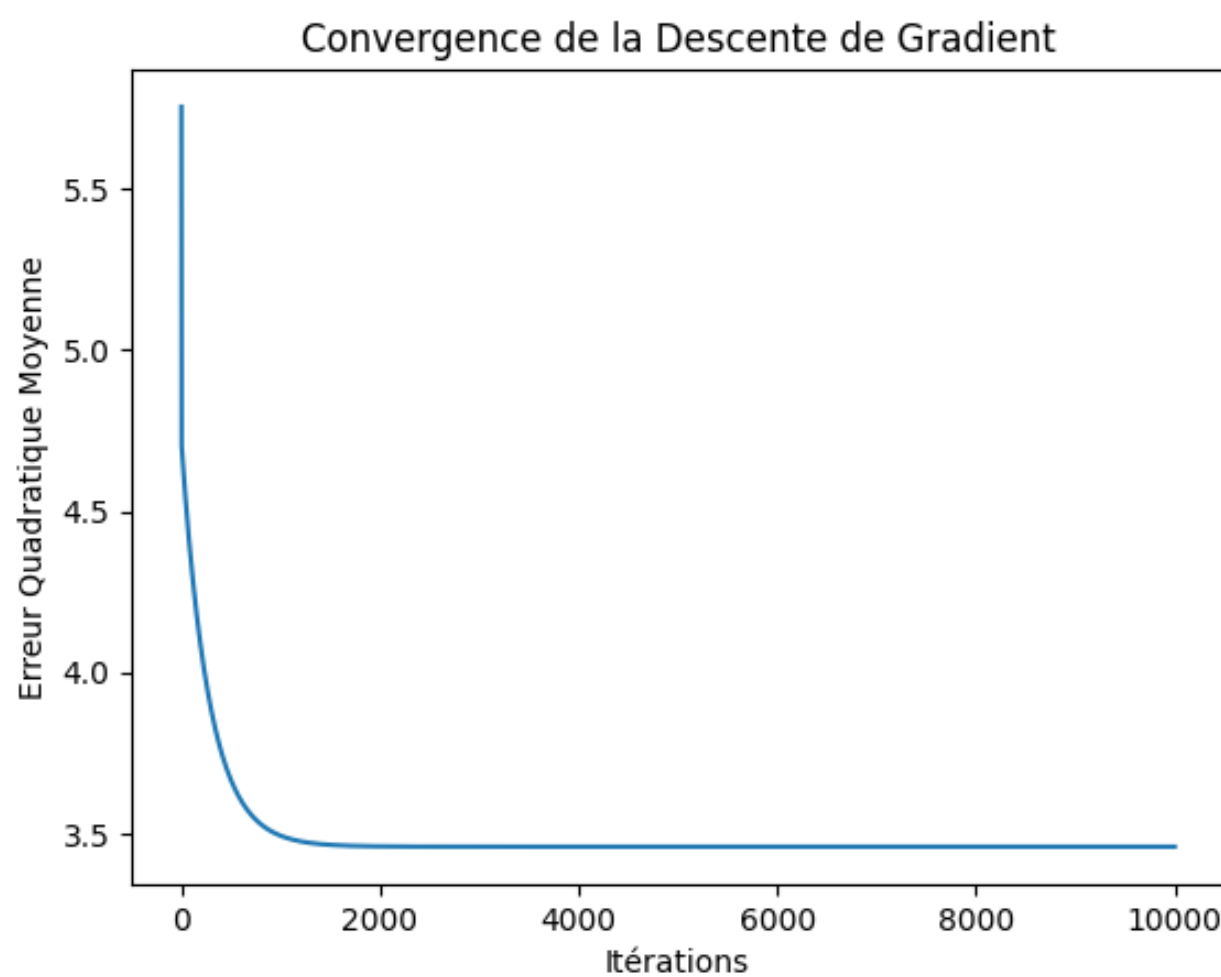
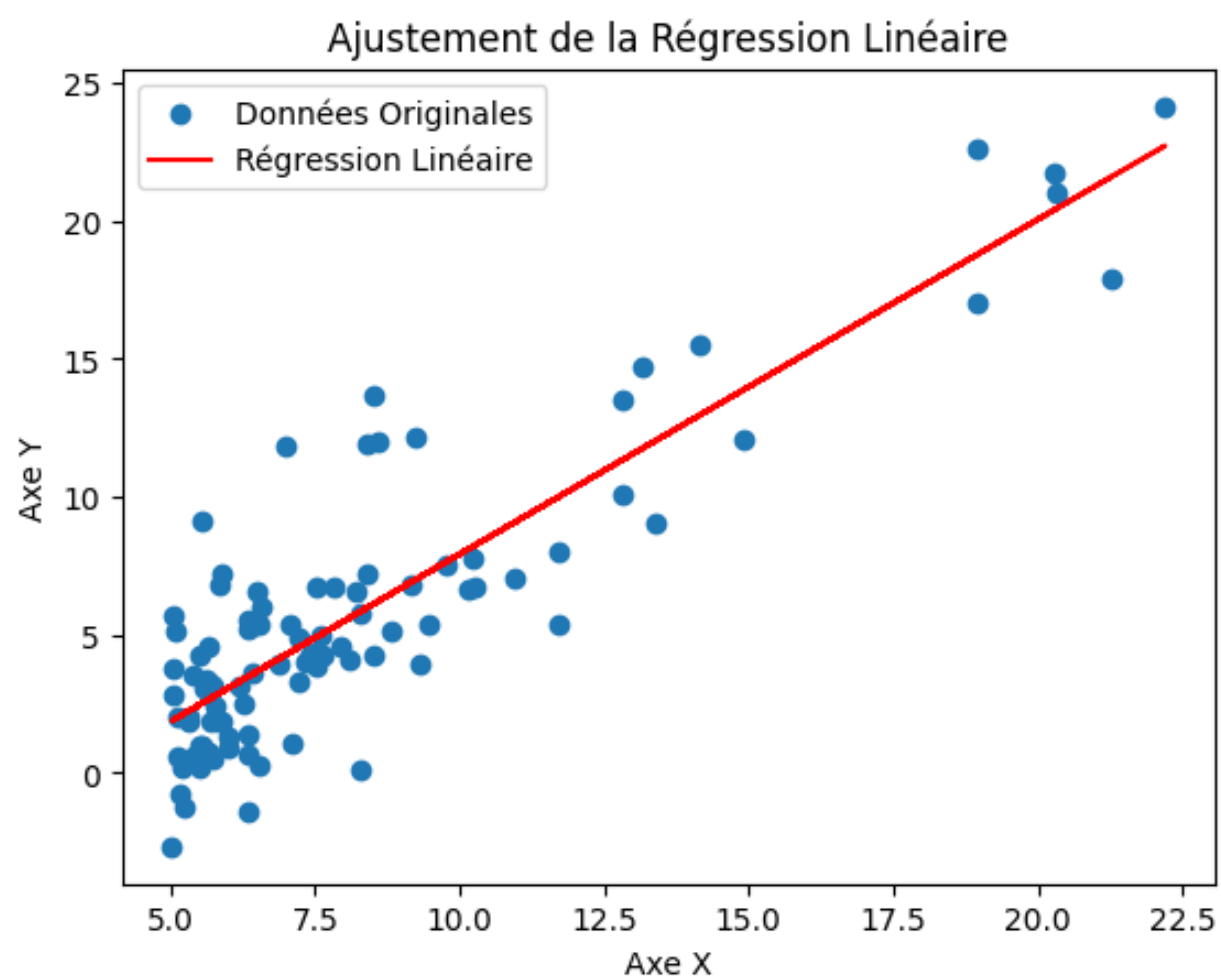
```



C:\Users\lenovo\AppData\Local\Temp\ipykernel_15516\676576063.py:47: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)

```
historique_erreur[i] = erreur_quadratique_moyenne(X, y, theta)
```

```
Theta Final: [[ 1.21354725]
 [-4.21150395]]
```



Prédiction pour [22.5, 1]: [[23.09330914]]

Régression polynomiale (Relation non linéaire entre prédictives & cible)

- L'objectif ici c'est de trouver un polynôme $P(X) \approx y$ la puissance de certaines variables indépendantes est supérieure à 1

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

```
In [15]: import numpy as np
import matplotlib.pyplot as plt

def afficher_pol(LPol): # Fonction pour afficher le polynôme
    L = LPol.copy() # Faire une copie de la liste des coefficients
    L.reverse() # Inverser la liste pour correspondre au format du polynôme
    L = [round(e, 2) for e in L] # Arrondir chaque coefficient à deux décimales
    for i in range(len(L)): # Afficher le polynôme dans un format lisible
        print('('+str(L[i])+'*X^'+str(i)+')', end=' + ')
    print()

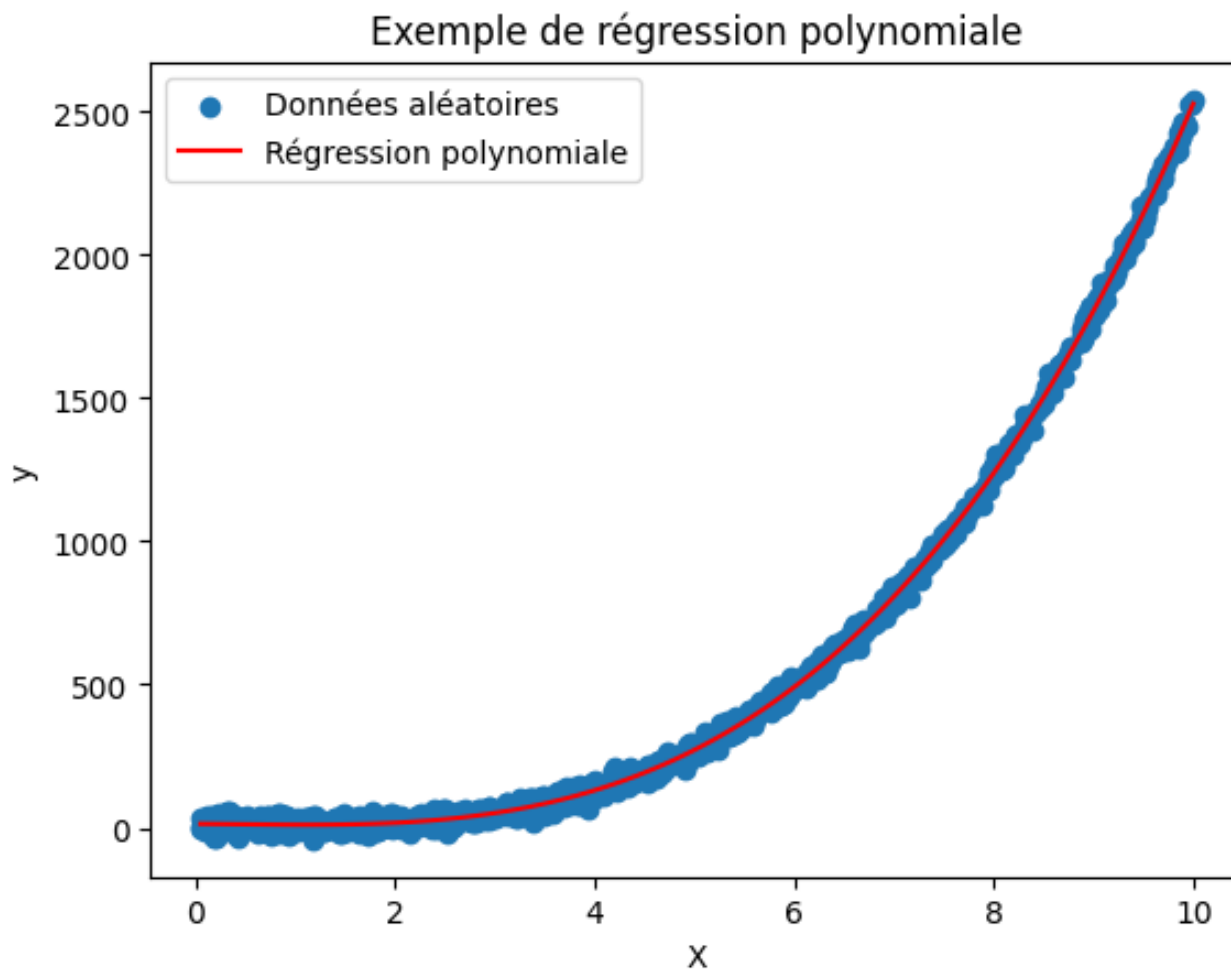
def Evaluer_pol(LPol, x): # Fonction pour évaluer le polynôme en un point donné
    s = 0
    L = LPol[:] # Faire une copie de la liste des coefficients
    L.reverse() # Inverser la liste pour correspondre au format du polynôme
    for i in range(len(LPol)): # Évaluer le polynôme au point donné x
        s += L[i]*(x**i)
    return s

# Générer des données aléatoires
np.random.seed(42)
```



```
x = 10 * np.random.rand(1000)
y = 3 * x**3 - 5 * x**2 + 2 * x + 8 + np.random.randn(1000) * 20
degre_polynome = 5
p = np.poly1d(np.polyfit(x, y, degre_polynome)) # Ajuster le polynôme aux données aléatoires
afficher_pol(list(p)) # Afficher le polynôme
les_x = np.linspace(min(x), max(x), 100) # Générer les valeurs de x pour le tracé
# Tracer les données aléatoires et la courbe de régression polynomiale
plt.scatter(x, y, label='Données aléatoires')
plt.plot(les_x, p(les_x), c='r', label='Régression polynomiale')
plt.xlabel('X'); plt.ylabel('y'); plt.title('Exemple de régression polynomiale'); plt.legend()
plt.show()
```

(13.82*X^0) + (-2.45*X^1) + (-4.16*X^2) + (3.2*X^3) + (-0.06*X^4) + (0.0*X^5) +



Régression Logistique (Sigmoid Function / sklearn)

No description has been provided for this image

- La fonction sigmoïde, souvent utilisée en régression logistique, est une fonction mathématique qui transforme n'importe quel nombre réel en une valeur comprise entre 0 et 1. Elle est appelée "sigmoïde" en raison de sa courbe en forme de S $data(x) = y \text{ in } [0,1]$. La fonction sigmoïde est représentée par la formule :

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

- Elle est particulièrement utile car elle comprime la sortie dans une plage pouvant être interprétée comme une probabilité, la rendant adaptée aux problèmes de **classification binaire**. (vrai, faux) / (spam, non) / (valide, non)...
- X: une observation (du **Training Set** ou du **Test Set**), cette variable est un vecteur contenant: [x0,..xn]
- xi : est une variable prédictive (feature) qui servira dans le calcul du modèle prédictif
- θ_i : est un **poids/paramètre** de la fonction hypothèse. Ce sont ces θ_i qu'on cherche à calculer pour obtenir notre fonction de prédiction.
- θ_0 : est une constante nommée le bias (biais)

No description has been provided for this image

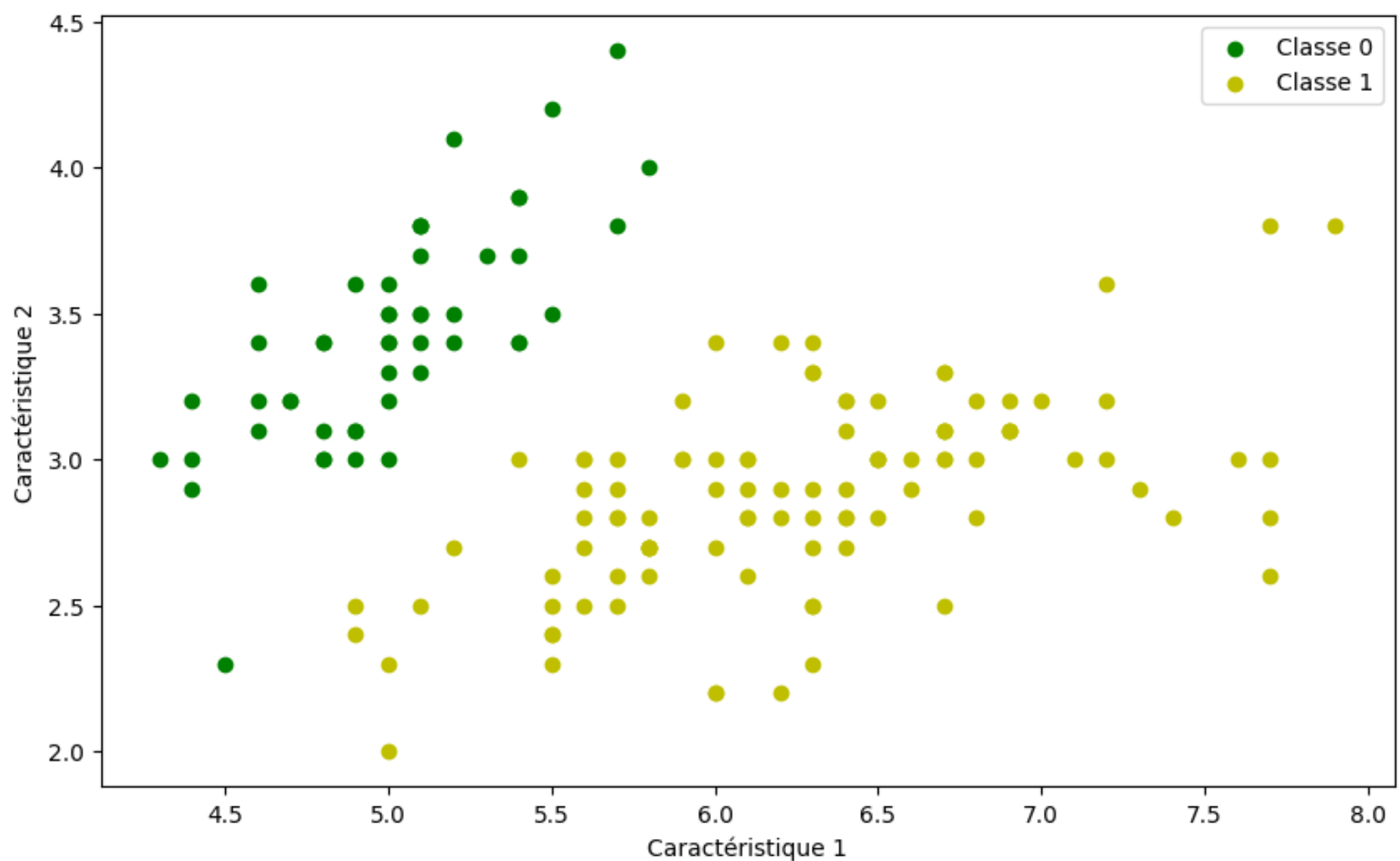
Ce code utilise le jeu de données Iris pour créer un problème de classification binaire en utilisant les deux premières caractéristiques. Il entraîne ensuite un modèle de régression logistique, évalue ses performances, et fait des prédictions sur un ensemble spécifié. La fonction de régression logistique est également affichée pour ces prédictions.

```
In [18]: import numpy as np # Importation de La bibliothèque NumPy
import matplotlib.pyplot as plt # Importation de La bibliothèque Matplotlib
from sklearn import datasets # Importation du module datasets de La bibliothèque scikit-Learn
from sklearn.linear_model import LogisticRegression # Importation du modèle de régression logistique de scikit-Learn
# fonctions pour séparer les données et faire la validation croisée
from sklearn.model_selection import train_test_split, cross_val_score
# Chargement du jeu de données Iris
iris = datasets.load_iris()
# Sélection des deux premières caractéristiques pour créer un problème de classification binaire
X = iris.data[:, :2]
# Re-étiquetage des fleurs pour avoir deux classes au lieu de trois pour classification binaire selection 2 species
```

```

y = []
for e in iris.target: # targets (setosa: 0, versicolor: 1 ,virginica: 2)
    if e == 0:
        y += [0]
    else: # species 1 & 2 est similaire
        y += [1]
# Conversion de la liste en un tableau NumPy
y = np.array(y)
# Affichage des points de données en utilisant Matplotlib
plt.figure(figsize=(10, 6)) # Taille de la figure(width, height)
# Points verts pour les fleurs ayant l'étiquette 0
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g', label='Classe 0')
# Points jaunes pour les fleurs ayant l'étiquette 1
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y', label='Classe 1')
# Ajout de la légende
plt.legend()
# Ajout des étiquettes des axes
plt.xlabel("Caractéristique 1")
plt.ylabel("Caractéristique 2")
# Affichage de la figure
plt.show()
# Création du modèle de régression logistique
model = LogisticRegression(C=1e20) # Tester avec 0.01
# Séparation des données en ensembles d'entraînement et de test
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Entraînement du modèle
model.fit(x_train, y_train)
# Calcul du score en utilisant la validation croisée
score = cross_val_score(model, x_train, y_train, cv=5, scoring='accuracy')
# Calcul du score sur l'ensemble de test et d'entraînement
score1 = model.score(x_test, y_test)
score2 = model.score(x_train, y_train)
# Affichage des scores
print("Score sur le test-set =", str(score1 * 100) + '%')
print("Score sur le train-set =", str(score2 * 100) + '%')
# Obtention des paramètres du modèle
theta0 = model.intercept_
theta1 = model.coef_[0][0]
theta2 = model.coef_[0][1]
# Iris à prédire
Iries_To_Predict = np.array([[5.5, 2.5], [7, 3], [3, 2], [5, 3]])
# Demande de prédiction
print("Prédictions pour Iries_To_Predict:")
print(model.predict(Iries_To_Predict))
# Affichage des résultats de la fonction de régression logistique
print("Résultats de la fonction de régression logistique pour Iries_To_Predict:")
print([model.predict_proba([[e[0], e[1]]]) for e in Iries_To_Predict])
# Fonction de régression logistique S(X)
def fct_reg_logistic(x1, x2):
    z = (theta0) + theta1 * x1 + (theta2 * x2)
    return z
# Affichage des résultats de la fonction de régression logistique pour Iries_To_Predict
print("Résultats de la fonction de régression logistique pour Iries_To_Predict:")
print([fct_reg_logistic(e[0], e[1]) for e in Iries_To_Predict])

```



Score sur le test-set = 100.0%
 Score sur le train-set = 100.0%
 Prédiction pour Iries_To_Predict:
 [1 1 0 0]
 Résultats de la fonction de régression logistique pour Iries_To_Predict:
 [array([[0., 1.]]), array([[0., 1.]]), array([[1.00000000e+00, 4.37888528e-73]]), array([[1.00000000e+00, 3.08160025e-19]])]
 Résultats de la fonction de régression logistique pour Iries_To_Predict:
 [array([93.91809461]), array([222.09084863]), array([-166.6119176]), array([-42.62366774])]

Les arbres de décision pour la classification

No description has been provided for this image

- Un arbre de décision représente la séquence (≠ d'ensemble) de questions à poser pour classer un nouvel exemple.
- Le but est de minimiser le nombre de questions à poser

Question : - comment quantifier les informations fournies par chaque attribut ? - comment partitionner l'attribut ?

Choix de l'attribut de partitionnement basé sur le Gain et l'Entropie (plus grand valeur).

En partitionnant les instances S en fonction d'un attribut, vous créez des sous-ensembles S_i .

1. Entropie de l'ensemble S (Entropie S) :

$$Entropie\ S = - \sum_{i=1}^c p_i \log_2(p_i)$$

2. Entropie du sous-ensemble S_i (Entropie S_i) :

$$Entropie\ S_i = - \sum_{j=1}^k p_{ij} \log_2(p_{ij})$$

3. Gain d'information (Gain S Attribut) :

$$Gain\ S\ Attribut = Entropie\ S - \sum_{i=1}^k \frac{|S_i|}{|S|} \cdot Entropie\ S_i$$

- Il représente la réduction de l'entropie après la partition des données selon l'attribut.

Choix de l'attribut de partitionnement basé sur l'indice de GINI (plus petite valeur).

Partitionnement de l'ensemble d'instances S en fonction d'un attribut {Attribut} :

Si vous partitionnez l'ensemble d'instances S en fonction d'un attribut \text{Attribut}, vous obtenez des sous-ensembles d'instances S_i où i varie de 1 au nombre k de valeurs distinctes de l'attribut \text{Attribut}.

1. Indice de Gini pour Mesurer le Désordre :

L'indice de Gini (Gini) est une mesure du désordre couramment utilisée dans la construction des arbres de décision. La formule de calcul de Gini est donnée par :

$$Gini(S) = 1 - \sum_{i=1}^k p_i^2$$

2. Choix d'un Attribut :

$$Gini_{Attribut} = \sum_{i=1}^k \frac{|S_i|}{|S|} \cdot Gini(S_i)$$

4. Formule Fournie :

$$U(S) = \sum_{i=1}^k \frac{|S_i|}{|S|} \cdot Gini(S_i)$$

Ce code utilise directement l'ensemble de données fourni, le divise en ensembles de formation et de test, entraîne les modèles d'arbre de décision en utilisant à la fois les critères d'indice de Gini et d'entropie et évalue leurs performances.

`confusion_matrix` lorsque nous construisons un modèle pour prédire quelque chose (comme 'oui' ou 'non'), nous voulons savoir dans quelle mesure il réussit. La matrice de confusion nous aide à comprendre cela:

- 1. **True Positive (TP):** You predicted 'yes,' and it was actually 'yes.'
- 2. **True Negative (TN):** You predicted 'no,' and it was actually 'no.'
- 3. **False Positive (FP):** You predicted 'yes,' but it was actually 'no.'
- 4. **False Negative (FN):** You predicted 'no,' but it was actually 'yes.'

Matrice de confusion :

	<i>Classé+</i>	<i>Classé-</i>
<i>Exemple+</i>	V _p	F _n
<i>Exemple-</i>	F _p	V _n

$$TauxD'erreur = 1 - \text{tauxderéussite}(accuracy) = \frac{F_p + F_n}{F_p + F_n + V_p + V_n}$$

```
In [1]: import numpy as np # Importation de La bibliothèque NumPy
import pandas as pd # Importation de La bibliothèque Pandas
from sklearn.metrics import confusion_matrix, accuracy_score # Importation des métriques pour évaluer Les performances
from sklearn.tree import DecisionTreeClassifier # Importation du modèle de classification par arbres de décision
# Jeu de données fourni
data = np.array([[1, 1, 2, 'oui'],[1, 4, 6, 'oui'],[1, 1, 3, 'oui'],[1, 5, 7, 'non'],[2, 2, 3, 'non'],
                 [2, 6, 5, 'non'],[2, 8, 9, 'non'],[2, 3, 3, 'non'],[2, 6, 9, 'non'],[2, 8, 9, 'non'],
                 [1, 8, 7, 'oui'],[1, 9, 6, 'oui'],[1, 0, 4, 'oui']])
# Séparation du jeu de données
X = data[:, :3] # variable
Y = data[:, 3] # cible
# Entraînement des modèles de classification avec arbres de décision en utilisant l'indice de Gini et l'entropie
clf_gini = DecisionTreeClassifier(criterion="gini")
clf_gini.fit(X[:8], Y[:8]) # training data set using gini
clf_entropy = DecisionTreeClassifier(criterion="entropy")
clf_entropy.fit(X[:8], Y[:8]) # training data set using entropy
# Test/prédicte des modèles avec les données restantes X's -> Y's (à partir de l'index 8)
print("Résultats en utilisant l'indice de Gini:")
y_pred_gini = clf_gini.predict(X[8:]) # predict(X) -> Y | Les resultats de prediction avec gini
# Matrice de confusion: [[true_negative false_positive] [false_negative true_positive]]
print("Matrice de confusion :", confusion_matrix(Y[8:], y_pred_gini))
print("Précision :", accuracy_score(Y[8:], y_pred_gini) * 100) # compare resultat de prediction avec les données de test
# Test des modèles avec les données restantes (à partir de l'index 8)
print("\nRésultats en utilisant l'entropie:")
y_pred_entropy = clf_entropy.predict(X[8:]) # predict(X) -> Y | Les resultats de prediction avec l'entropie
# Matrice de confusion: [[true_negative false_positive] [false_negative true_positive]]
print("Matrice de confusion :", confusion_matrix(Y[8:], y_pred_entropy))
print("Précision :", accuracy_score(Y[8:], y_pred_entropy) * 100) # compare resultat de prediction avec les données de test
```

Résultats en utilisant l'indice de Gini:
Matrice de confusion : [[2 0]
[2 1]]
Précision : 60.0

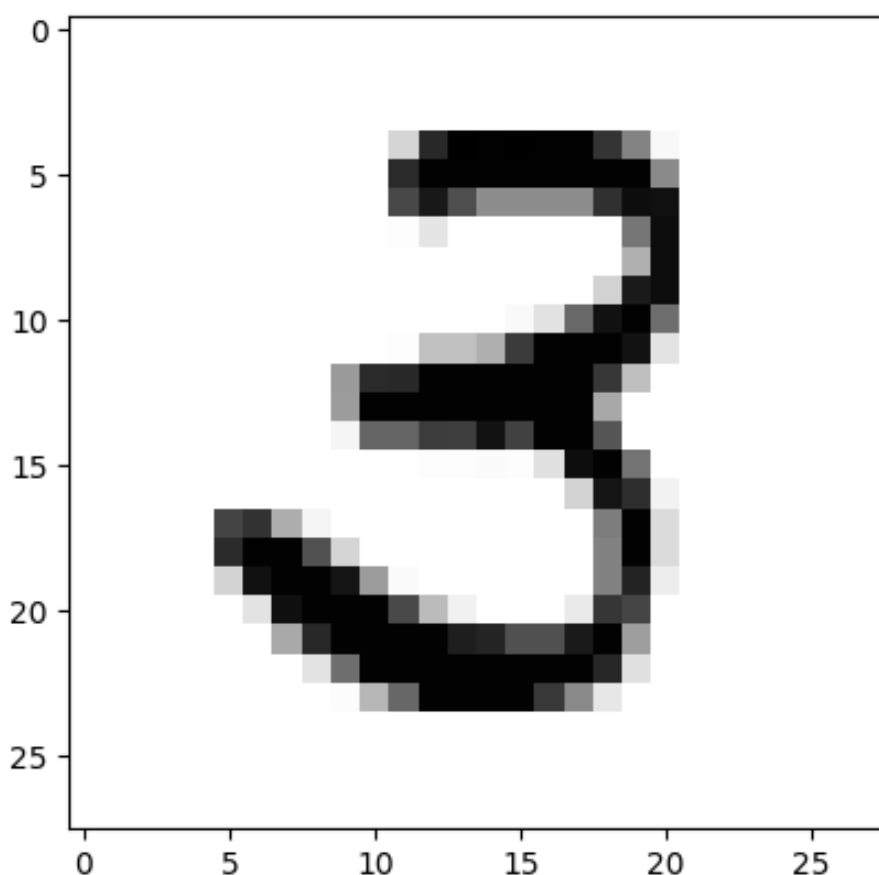
Résultats en utilisant l'entropie:
Matrice de confusion : [[2 0]
[1 2]]
Précision : 80.0

Application de Machine Learning sur la reconnaissance des nombres manuscrits

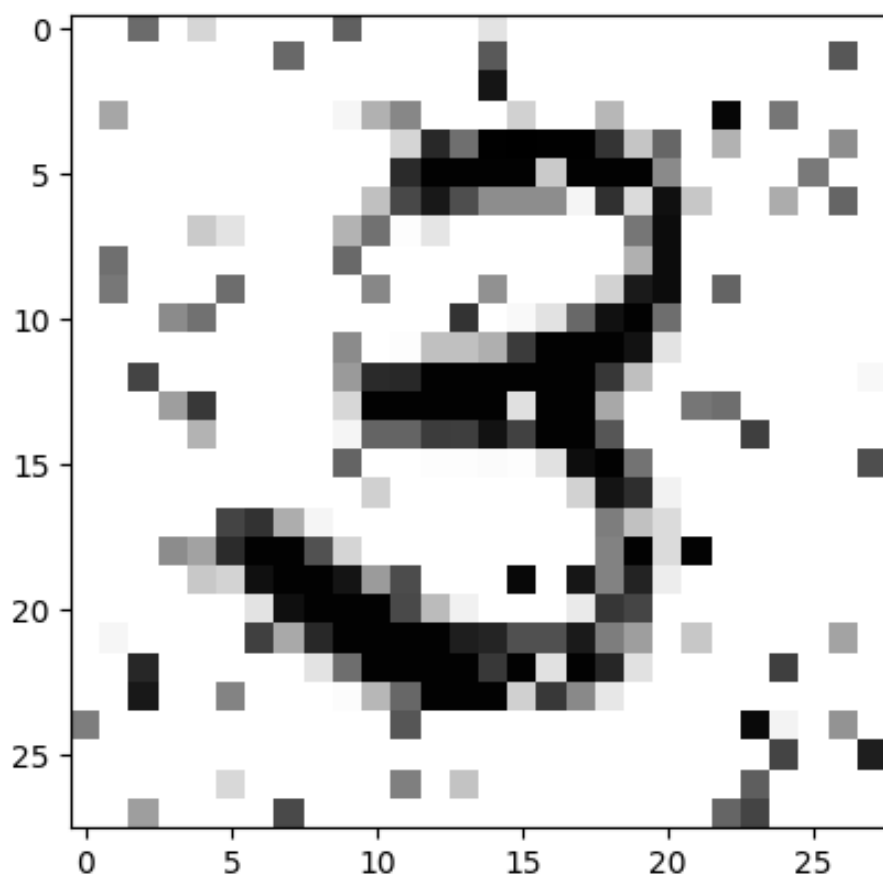
ce code illustre le processus de création, d'entraînement, d'évaluation, et de prédiction d'un modèle de classification d'arbres de décision sur un ensemble de données d'images de chiffres en niveaux de gris.

```
In [8]: import numpy as np # Importation de La bibliothèque NumPy
# Permet d'afficher Les graphiques directement dans Le carnet Jupyter
%matplotlib inline
import matplotlib.pyplot as plt # Importation de La bibliothèque Matplotlib pour La visualisation
import pandas # Importation de La bibliothèque Pandas pour La manipulation de données
from sklearn.tree import DecisionTreeClassifier # Importation du modèle de classification par arbres de décision
import os # Importation du module OS pour interagir avec Le système d'exploitation
# fonction pour diviser Le jeu de données en ensembles d'entraînement et de test
from sklearn.model_selection import train_test_split
from copy import deepcopy # Importation de La fonction deepcopy depuis Le module copy
# Chargement des données à partir d'un fichier CSV | 'as_matrix()' a été déprécié, on utilise 'values' à La place
data = pandas.read_csv("data_files\\images_chiffres_codees_niveau_de_gris.csv").values # jeu de données contient 2559 lignes
MonModele = DecisionTreeClassifier() # Création d'une instance du classificateur d'arbres de décision
x = data[:, 1:]; # variable
label = data[:, 0] # cible
# Séparation du jeu de données en ensembles d'entraînement et de test
x_train, x_test, y_train, y_test = train_test_split(x, label, test_size=0.2)
MonModele.fit(x_train, y_train) # Entraînement du modèle avec L'ensemble d'entraînement
# Évaluation de La précision du modèle sur L'ensemble de test
p = MonModele.predict(x_test) # Prédiction des étiquettes pour L'ensemble de test
# Calcul de La précision du modèle par comparaison de module et Les données de test
accuracy = MonModele.score(x_test, y_test)
print("Le pourcentage de réussite = " + str(accuracy * 100) + '%') # Affichage du pourcentage de réussite
# Affichage d'une image à partir de L'ensemble de test
image_index = 53 # Index de L'image à afficher
d = x_test[image_index] # Sélection de L'image du jeu de test
d2 = deepcopy(d) # Création d'une copie profonde de L'image
d2.shape = (28, 28) # Redimensionnement de L'image à 28x28 (supposant que c'est une image de 28x28 pixels)
plt.imshow(255 - d2, cmap='gray') # Affichage de L'image inversée en niveaux de gris
plt.show() # Affichage de L'image
# Ajout de bruit à une image et prédiction du modèle
Nombre_de_pixels_errones = 100 # Nombre de pixels à perturber
for i in range(Nombre_de_pixels_errones):
    position = np.random.randint(0, 784, 1)[0] # Sélection aléatoire d'une position de pixel
    bruit = np.random.randint(-200, 200, 1)[0] # Génération aléatoire d'une valeur de bruit
    d[position] += bruit # Ajout du bruit à La position sélectionnée
    d[position] = d[position] % 255 # Assure que La valeur reste dans La plage des niveaux de gris
# Prédiction du modèle pour L'image bruitée
print(MonModele.predict([d])) # Affichage de La prédiction du modèle pour L'image bruitée
d.shape = (28, 28) # Redimensionnement de L'image à 28x28
plt.imshow(255 - d, cmap='gray') # Affichage de L'image bruitée inversée en niveaux de gris
plt.show() # Affichage de L'image bruitée
```

Le pourcentage de réussite = 74.0234375%




[3]



KNN: k Nearest Neighbors (classification/supervised learning)

K-NN est un algorithme flexible qui peut être utilisé dans divers contextes d'apprentissage supervisé, basé sur le principe de proximité entre les points dans l'espace des caractéristiques (*Dis moi qui sont tes voisins, je te dirais qui tu es*).

 No description has been provided for this image

Les étapes de KNN:

1. Choisir la distance, la mesure de similarité entre les échantillons(instances)
2. Choisir (déterminer) la valeur de K ($k = n + 1$) impaire
3. Calculer la distance entre la nouvelle entrée et toutes les données de la base d'apprentissage.
4. Déterminer les classes des K plus proches voisins
5. Donner la prédiction pour la nouvelle entrée.

Algorithme de KNN:

1. Charger les données
2. Initialiser k au nombre de plus proches voisins choisi (*sous group des voisines de taille k*)
3. Pour chaque exemple dans les données:
 - Calculer la distance entre notre requête et l'observation itérative actuelle de la boucle depuis les données.
 - Ajouter la distance et l'indice de l'observation concernée à une collection ordonnée de données
4. Trier cette collection ordonnée contenant distances et indices de la plus petite distance à la plus grande (dans l'ordre croissant).
5. Sélectionner les k premières entrées de la collection de données triées (équivalent aux k plus proches voisins)
6. Obtenir les étiquettes des k entrées sélectionnées
7. Si régression, retourner la moyenne des k étiquettes
8. Si classification, retourner le mode (valeur la plus fréquente/commune) des k étiquettes

```
In [1]: # Importation de la classe CountVectorizer depuis Le module sklearn.feature_extraction.text
from sklearn.feature_extraction.text import CountVectorizer
texte = ["La vie est douce", "La vie est tranquille, est belle, est douce"] # Définition d'une liste de textes
vect = CountVectorizer(binary=True) # Détermine l'occurrence des mots (1 si existant / 0 sinon) avec binary=True
T = vect.fit_transform(texte) # Transformation des textes en une matrice d'occurrences binaires (0 ou 1)
dictionnaire_des_mots = vect.vocabulary_ # Récupération du dictionnaire des mots et de leurs indices dans le vocabulaire
print("dictionnaire_des_mots :", dictionnaire_des_mots)
liste_des_mots = list(dictionnaire_des_mots.keys()) # Création d'une liste contenant les mots du dictionnaire
print("liste_des_mots :", liste_des_mots)
Matrice_sparse_correspondante = T.toarray() # Conversion de la matrice creuse en une matrice dense (numpy array)
print("Matrice_sparse_correspondante: \n", Matrice_sparse_correspondante) # 'la' -> exist dans [text1, text2] -> [1,1]..
```

```
dictionnaire_des_mots : {'la': 3, 'vie': 5, 'est': 2, 'douce': 1, 'tranquille': 4, 'belle': 0}
liste_des_mots : ['la', 'vie', 'est', 'douce', 'tranquille', 'belle']
Matrice_sparse_correspondante:
[[0 1 1 1 0 1]
 [1 1 1 1 1 1]]
```

```
In [2]: import numpy as np # Importation de la bibliothèque NumPy
import pandas # Importation de la bibliothèque Pandas
from sklearn.model_selection import train_test_split # diviser les données en ensembles d'entraînement et de test
from sklearn.feature_extraction.text import CountVectorizer # classe CountVectorizer pour la vectorisation du texte
from sklearn.linear_model import LogisticRegression # Importation du modèle de régression logistique
spams = pandas.read_table("data_files/SMSspamCollection.txt", sep="\t", header=0) # Chargement des données
```

```
Score sur les données d'entraînement : 0.9976923076923077
Score sur les données de test : 0.9814593301435407
```

K-means est un algorithme de regroupement qui divise des points de données en k clusters en fonction de leur similarité. Il affecte itérativement des points au cluster dont le centre est le plus proche et met à jour ces centres jusqu'à convergence, fournissant une partition distincte des données.

1. Choisissez k objets pour former k clusters/groupes.
2. Affectez chaque objet O au cluster/groupe C_i de centre M_i , où $dist(O, M_i)$ est **minimal**.
3. Recalculer le barycentre M_i de chaque cluster.
4. Répétez les étapes 2 et 3 jusqu'à convergence.

[illegible]

```
c:\Users\lenovo\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
    super()._check_params_vs_input(X, default n_init=10)
```


Base R

Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Libraries

install.packages('dplyr')
Download and install a package from CRAN.
library(dplyr)
Load the package into the session, making all its functions available to use.
dplyr::select
Use a particular function from a package.
data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).
setwd('C://file/path')
Change the current working directory.
Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x) Return x sorted.	rev(x) Return x reversed.
table(x) See counts of values.	unique(x) See unique values.

Selecting Vector Elements

By Position

x[4]	The fourth element.
x[-4]	All but the fourth.
x[2:4]	Elements two to four.
x[-(2:4)]	All elements except two to four.
x[c(1, 5)]	Elements one and five.

By Value

x[x == 10]	Elements which are equal to 10.
x[x < 0]	All elements less than zero.
x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.

Named Vectors

x['apple']	Element with name 'apple'.
-------------------	----------------------------

Programming

For Loop

```
for (variable in sequence){  
  Do something  
}
```

Example

```
for (i in 1:4){  
  j <- i + 10  
  print(j)  
}
```

While Loop

```
while (condition){  
  Do something  
}
```

Example

```
while (i < 5){  
  print(i)  
  i <- i + 1  
}
```

If Statements

```
if (condition){  
  Do something  
} else {  
  Do something different  
}
```

Example

```
if (i > 3){  
  print('Yes')  
} else {  
  print('No')  
}
```

Functions

```
function_name <- function(var){  
  Do something  
  return(new_variable)  
}
```

Example

```
square <- function(x){  
  squared <- x*x  
  return(squared)  
}
```

Reading and Writing Data

Input	Ouput	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/write.table.
load('file.RData')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions	a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
	a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

<code>as.logical</code>	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
<code>as.numeric</code>	1, 0, 1	Integers or floating point numbers.
<code>as.character</code>	'1', '0', '1'	Character strings. Generally preferred to factors.
<code>as.factor</code>	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Maths Functions

<code>log(x)</code>	Natural log.	<code>sum(x)</code>	Sum.
<code>exp(x)</code>	Exponential.	<code>mean(x)</code>	Mean.
<code>max(x)</code>	Largest element.	<code>median(x)</code>	Median.
<code>min(x)</code>	Smallest element.	<code>quantile(x)</code>	Percentage quantiles.
<code>round(x, n)</code>	Round to n decimal places.	<code>rank(x)</code>	Rank of elements.
<code>signif(x, n)</code>	Round to n significant figures.	<code>var(x)</code>	The variance.
<code>cor(x, y)</code>	Correlation.	<code>sd(x)</code>	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```




The Environment

<code>ls()</code>	List all variables in the environment.
<code>rm(x)</code>	Remove x from the environment.
<code>rm(list = ls())</code>	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrixes

```
m <- matrix(x, nrow = 3, ncol = 3)
Create a matrix from x.
```

 <code>m[2,]</code> - Select a row	<code>t(m)</code> Transpose
 <code>m[, 1]</code> - Select a column	<code>m %*% n</code> Matrix Multiplication
 <code>m[2, 3]</code> - Select an element	<code>solve(m, n)</code> Find x in: $m \cdot x = n$

Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
A list is collection of elements which can be of different types.
```

<code>l[[2]]</code> Second element of l.	<code>l[1]</code> New list with only the first element.	<code>l\$x</code> Element named x.	<code>l['y']</code> New list with only element named y.
---	--	---------------------------------------	--



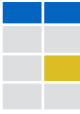
Also see the **dplyr** library.

Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
A special case of a list where all elements are the same length.
```

x	y
1	a
2	b
3	c

Matrix subsetting

<code>df[, 2]</code>	
<code>df[2,]</code>	
<code>df[2, 2]</code>	

List subsetting

<code>df\$x</code>		<code>df[[2]]</code>	
<i>Understanding a data frame</i>			
<code>View(df)</code>	See the full data frame.		
<code>head(df)</code>	See the first 6 rows.		

`nrow(df)`
Number of rows.

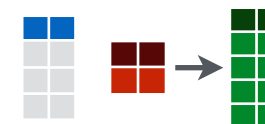
`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

Also see the **stringr** library.

<code>paste(x, y, sep = ' ')</code>	Join multiple vectors together.
<code>paste(x, collapse = ' ')</code>	Join elements of a vector together.
<code>grep(pattern, x)</code>	Find regular expression matches in x.
<code>gsub(pattern, replace, x)</code>	Replace matches in x with a string.
<code>toupper(x)</code>	Convert to uppercase.
<code>tolower(x)</code>	Convert to lowercase.
<code>nchar(x)</code>	Number of characters in a string.

Factors

<code>factor(x)</code> Turn a vector into a factor. Can set the levels of the factor and the order.	<code>cut(x, breaks = 4)</code> Turn a numeric vector into a factor but 'cutting' into sections.
--	---

Statistics

<code>lm(x ~ y, data=df)</code> Linear model.	<code>t.test(x, y)</code> Perform a t-test for difference between means.	<code>prop.test</code> Test for a difference between proportions.
<code>glm(x ~ y, data=df)</code> Generalised linear model.	<code>pairwise.t.test</code> Perform a t-test for paired data.	<code>aov</code> Analysis of variance.
<code>summary</code> Get more detailed information out a model.		

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	<code>rnorm</code>	<code>dnorm</code>	<code>pnorm</code>	<code>qnorm</code>
Poisson	<code>rpois</code>	<code>dpois</code>	<code>ppois</code>	<code>qpois</code>
Binomial	<code>rbinom</code>	<code>dbinom</code>	<code>pbinom</code>	<code>qbinom</code>
Uniform	<code>runif</code>	<code>dunif</code>	<code>punif</code>	<code>qunif</code>

Plotting

Also see the **ggplot2** library.

 <code>plot(x)</code> Values of x in order.	 <code>plot(x, y)</code> Values of x against y.	 <code>hist(x)</code> Histogram of x.
---	---	---

Dates

See the **lubridate** library.

R cheat sheet

1. Basics

Commands	<code>objects()</code>	List of objects in workspace
	<code>ls()</code>	Same
	<code>rm(object)</code>	Delete 'object'
Assignments	<code><-</code>	Assign value to a variable
	<code>=</code>	Same
Getting help	<code>help(fun)</code>	Display help file for function <code>fun()</code>
	<code>args(fun)</code>	List arguments of function <code>fun()</code>
Libraries / packages	<code>library(pkg)</code>	Open package (library) 'pkg'
	<code>library(help=pkg)</code>	Display description of package 'pkg'

2. Vectors and data types

Generating	<code>seq(-4,4,0.1)</code>	Sequence: -4.0, -3.9, -3.8, ..., 3.9, 4.0
	<code>2:7</code>	Same as <code>seq(2,7,1)</code>
	<code>c(5,7,9,1:3)</code>	Concatenation (vector): 5 7 9 1 2 3
	<code>rep(1,5)</code>	1 1 1 1 1
	<code>rep(4:6,1:3)</code>	4 5 5 6 6 6
	<code>gl(3,2,12)</code>	Factor with 3 levels, repeat each level in blocks of 2, up to length 12 (1 1 2 2 3 3 1 1 2 2 3 3)
Coercion	<code>as.numeric(x)</code>	Convert to numeric
	<code>as.character(x)</code>	Convert to text string
	<code>as.logical(x)</code>	Convert to logical
	<code>factor(x)</code>	Create factor from vector x
	<code>unlist(x)</code>	Convert list, result from <code>table()</code> etc. to vector

3. Data frames

Accessing data	<code>data.frame(height, weight)</code>	Collect vectors 'height' and 'weight' into data frame
	<code>dfr\$var</code>	Select vector 'var' in data frame 'dfr'
	<code>attach(dfr)</code>	Put data frame in search path
	<code>detach()</code>	- and remove it from the path
Editing	<code>dfr2 <- edit(dfr)</code>	open data frame 'dfr' in spreadsheet, write changed version into new data frame 'dfr2'
	<code>fix(dfr)</code>	open data frame 'dfr' in spreadsheet, changes will overwrite entries in 'dfr'
Summary	<code>dim(dfr)</code>	Number of rows and columns in data frame 'dfr', works also for matrices and arrays
	<code>summary(dfr)</code>	Summary statistics for each variable in 'dfr'

Modified from: P. Dalgaard (2002). Introductory Statistics with R. Springer, New York.

4. Input and export of data

General	<code>data(name)</code>	Built-in data set
	<code>read.table("file.txt")</code>	Read from external ASCII file
Arguments to <code>read.table()</code>	<code>header = TRUE</code>	First line has variable names
	<code>row.names = 1</code>	First column has row names
	<code>sep = ",", "</code>	Data are separated by commas
	<code>sep = "\t"</code>	Data are separated by tabs
	<code>dec = ",", "</code>	Decimal point is comma
	<code>na.strings = ".", "</code>	Missing value is dot
Variants of <code>read.table()</code>	<code>read.csv("file.csv")</code>	Comma separated
	<code>read.delim("file.txt")</code>	Tab delimited text file
Export	<code>write.table()</code>	see <code>help(write.table)</code> for details
Adding names	<code>names()</code>	Column names for data frame or list only
	<code>dimnames()</code>	Row and column names, also for matrix

5. Indexing / selection / sorting

Vectors	<code>x[1]</code>	First element
	<code>x[1:5]</code>	Subvector containing the first five elements
	<code>x[c(2,3,5)]</code>	Elements nos. 2, 3, and 5
	<code>x[y <= 30]</code>	Selection by logical expression
	<code>x[sex == "male"]</code>	Selection by factor variable
	<code>i <- c(2,3,5); x[i]</code>	Selection by numerical variable
	<code>k <- (y <= 30); x[k]</code>	Selection by logical variable
	<code>length(x)</code>	Returns length of vector x
Matrices, data frames	<code>m[4,]</code>	Fourth row
	<code>m[, 3]</code>	Third column
	<code>dfr[dfr\$var <= 30,]</code>	Partial data frame (not for matrices)
	<code>subset(dfr, var <= 30)</code>	Same, often simpler (not for matrices)
	<code>m[m[, 3] <= 30,]</code>	Partial matrix (also for data frames)
Sorting	<code>sort(c(7,9,10,6))</code>	Returns the sorted values: 6, 7, 9, 10
	<code>order(c(7,9,10,6))</code>	Returns the element number in order of ascending values: 4, 1, 2, 3
	<code>order(c(7,9,10,6), decreasing = TRUE)</code>	same, but in order of decreasing values: 3, 2, 1, 4
	<code>rank(c(7,9,10,6))</code>	Returns the ranks in order of ascending values: 2, 3, 4, 1

6. Missing values

Functions	<code>is.na(x)</code>	Logical vector. TRUE where x has NA
	<code>complete.cases(x1,x2,...)</code>	Neither missing in x1, nor x2, nor ...
Arguments to other functions	<code>na.rm =</code>	In statistical functions: Remove missing if TRUE, returns NA if FALSE
	<code>na.last =</code>	In 'sort' TRUE, FALSE and NA means "last", "first", and "discard"
	<code>na.action =</code>	in 'lm()', etc., values <code>na.fail</code> , <code>na.omit</code> , <code>na.exclude</code>
	<code>na.print =</code>	In 'summary()' and 'print()': How to represent NA in output
	<code>na.strings =</code>	In 'read.table()': Codes(s) for NA in input

7. Numerical functions

Mathematical	<code>log(x)</code>	Logarithm of x, natural logarithm
	<code>log(x, 10)</code>	Base10 logarithm of x
	<code>exp(x)</code>	Exponential function e^x
	<code>sin(x)</code>	Sine
	<code>cos(x)</code>	Cosine
	<code>tan(x)</code>	Tangent
	<code>asin(x)</code>	Arcsin (inverse sine)
	<code>min(x)</code>	Smallest value in vector
	<code>min(x1, x2, ...)</code>	minimum number over several vectors
	<code>max(x)</code>	Largest value in vector
	<code>range(x)</code>	Like <code>c(min(x), max(x))</code>
	<code>pmin(x1, x2, ...)</code>	Parallel (elementwise) minimum over multiple equally long vectors
	<code>length(x)</code>	Number of elements in vector
	<code>sum(x)</code>	Sum of values in vector
	<code>cumsum(x)</code>	Cumulative sum of values in vector
	<code>sum(complete.cases(x))</code>	Number of non-missing elements
Statistical	<code>mean(x)</code>	Average
	<code>median(x)</code>	Median
	<code>quantile(x, p)</code>	Quantiles: <code>median = quantile(x, 0.5)</code>
	<code>var(x)</code>	Variance
	<code>sd(x)</code>	Standard deviation
	<code>cor(x, y)</code>	Pearson correlation
	<code>cor(x, y, method = "spearman")</code>	Spearman rank correlation

8. Programming

Conditional execution	<code>if(p < 0.5) print("Hooray")</code>	Print "Hooray" if condition is true
	<code>if(p < 0.5) { print("Hooray") i = i + 1 }</code>	If condition is true, perform all commands within the curved brackets { }
	<code>if(p < 0.5) { print("Hooray") } else { i = i + 1 }</code>	Conditional execution with an alternative
Loop	<code>for(i in 1:10) { print(i) }</code>	Go through loop 10 times
	<code>i <- 1 while(i <= 10) { print(i) i = i + 1 }</code>	Same, but more complicated
User-defined function	<code>func<- function(a, b, doit = FALSE) { if(doit) {a + b} else 0 }</code>	Defines a function 'fun' that returns the sum of a and b if the argument 'doit' is set to TRUE, or zero, if 'doit' is FALSE

9. Operators

Arithmetic	<code>+</code>	Addition
	<code>-</code>	Subtraction
	<code>*</code>	Multiplication
	<code>/</code>	Division
	<code>^</code>	Raise to the power of
	<code>% / %</code>	Integer division: <code>5 %/ 3 = 1</code>
	<code>% %</code>	Remainder from integer division: <code>5 %% 3 = 2</code>
Logical or relational	<code>=</code>	Equal to
	<code>!=</code>	Not equal to
	<code><</code>	Less than
	<code>></code>	Greater than
	<code><=</code>	Less than or equal to
	<code>>=</code>	Greater than or equal to
	<code>is.na(x)</code>	Missing?
	<code>&</code>	Logical AND
	<code> </code>	Logical OR
	<code>!</code>	Logical NOT

10. Tabulation, grouping, recoding

General	<code>table(x)</code>	Frequency table of vector (factor) x
	<code>table(x, y)</code>	Crosstabulation of x and y
	<code>xtabs(~ x + y)</code>	Formula interface for crosstabulation: use <code>summary()</code> for chi-square test
	<code>factor(x)</code>	Convert vector to factor
	<code>cut(x, breaks)</code>	Groups from cutpoints for continuous variable, breaks is a vector of cutpoints
Arguments to <code>factor()</code>	<code>levels = c()</code>	Values of x to code. Use if some values are not present in data, or if the order would be wrong.
	<code>labels = c()</code>	Values associated with factor levels
	<code>exclude = c()</code>	Values to exclude. Default NA. Set to NULL to have missing values included as a level.
Arguments to <code>cut()</code>	<code>breaks = c()</code>	Cutpoints. Note values of x outside of 'breaks' gives NA. Can also be a single number, the number of cutpoints.
	<code>labels = c()</code>	Names for groups. Default is 1, 2, ...
Factor recoding	<code>levels(f) <- names</code>	New level names
	<code>factor(newcodes[f])</code>	Combining levels: 'newcodes', e.g., <code>c(1,1,1,2,3)</code> to amalgamate the first 3 of 5 groups of factor f

11. Manipulations of matrices and lists

Matrix algebra	<code>m1 % * % m2</code>	Matrix product
	<code>t(m)</code>	Matrix transpose
	<code>m[lower.tri(m)]</code>	Returns the values from the lower triangle of matrix m as a vector
	<code>diag(m)</code>	Returns the diagonal elements of matrix m
	<code>matrix(x, dim1, dim2)</code>	Fill the values of vector x into a new matrix with dim1 rows and dim2 columns,
Marginal operations etc.	<code>apply(m, dim, fun)</code>	Applies the function 'fun' to each row (dim = 1) or column (dim = 2) of matrix m
	<code>tapply(m, list(f1, f2), fun)</code>	Can be used to aggregate columns or rows within matrix m as defined by f1, f2, using the function 'fun' (e.g., mean, max)
	<code>split(x, f)</code>	Split vector, matrix or data frame by factor x. Different results for matrix and data frame! The result is a list with one object for each level of f.
	<code>sapply(list, fun)</code> <code>sapply(split(x, f), fun)</code>	applies the function 'fun' to each object in a list, e.g. as created by the split function

12. Statistical standard methods

Parametric tests, continuous data	<code>t.test</code>	One- and two-sample t-test
	<code>pairwise.t.test</code>	Pairwise comparison of means
	<code>cor.test</code>	Significance test for correlation coeff.
	<code>var.test</code>	Comparison of two variances (F-test)
	<code>lm(y ~ x)</code>	Regression analysis
	<code>lm(y ~ f)</code>	One-way analysis of variance
	<code>lm(y ~ x1 + x2 + x3)</code>	Multiple regression
	<code>lm(y ~ f1 * f2)</code>	Two-way analysis of variance
Non-parametric	<code>wilcox.test</code>	One- and two-sample Wilcoxon test
	<code>kruskal.test</code>	Kruskal-Wallis test
	<code>friedman.test</code>	Friedman's two-way analysis of variance
cor.test variant	<code>method = "spearman"</code>	Spearman rank correlation
Discrete response	<code>binom.test</code>	Binomial test (incl. sign test)
	<code>prop.test</code>	Comparison of proportions
	<code>fisher.test</code>	Exact test in 2 x 2 tables
	<code>chisq.test</code>	Chi-square test of independence
	<code>glm(y ~ x1+x2, binomial)</code>	Logistic regression

13. Statistical distributions

Normal distribution	<code>dnorm(x)</code>	Density function
	<code>pnorm(x)</code>	Cumulative distribution function $P(X \leq x)$
	<code>qnorm(p)</code>	p-quantile, returns x in: $P(X \leq x) = p$
	<code>rnorm(n)</code>	n random normally distributed numbers
Distributions	<code>pnorm(x, mean, sd)</code>	Normal
	<code>plnorm*x, mean, sd)</code>	Lognormal
	<code>pt(x, df)</code>	Student's t distribution
	<code>pf(x, n1, n2)</code>	F distribution
	<code>pchisq(x, df)</code>	Chi-square distribution
	<code>pbinom(x, n, p)</code>	Binomial
	<code>ppois(x, lambda)</code>	Poisson
	<code>punif(x, min, max)</code>	Uniform
	<code>pexp(x, rate)</code>	Exponential
	<code>pgamma(x, shape, scale)</code>	Gamma
	<code>pbeta(x, a, b)</code>	Beta

14. Models

Model formulas	~	As explained by
	+	Additive effects
	:	Interaction
	*	Main effects + interaction: a*b =a+b+a:b
	-1	Remove intercept
Linear models	lm.out <- lm(y ~ x)	Fit model and save results as 'lm.out'
	summary(lm.out)	Coefficients etc.
	anova(lm.out)	Analysis of variance table
	fitted(lm.out)	Fitted values
	resid(lm.out)	Residuals
	predict(lm.out,newdata)	Predictions for a new data frame
Other models	glm(y ~ x, binomial)	Logistic regression
	glm(y ~ x, poisson)	Poisson regression
	gam(y ~ s(x))	General additive model for non-linear regression with smoothing. Package: gam
	tree(y ~ x1+x2+x3)	Classification (y = factor) or regression (y = numeric) tree. Package: tree
Diagnostics	rstudent(lm.out)	Studentized residuals
	dfbetas(lm.out)	Change in standardized regression coefficients beta if observation removed
	dffits(lm.out)	Change in fit if observation removed
Survival analysis	S <- Surv(time,ev)	Create survival object. Package: survival
	survfit(S)	Kaplan-Meier estimate
	plot(survfit(S))	Survival curve
	survdif(S ~ g)	(Log-rank) test for equal survival curves
	coxph(S ~ x1 + x2)	Cox's proportional hazards model
Multivariate	dist()	Calculate Euclidean or other distances
	hclust()	Hierarchical cluster analysis
	kmeans()	k-means cluster analysis
	rda()	Perform principal component analysis PCA or redundancy analysis RDA. Package 'vegan'.
	cca()	Perform (canonical) correspondence analysis, CA /CCA. Package: 'vegan'
	diversity()	Calculate diversity indices. Pkg: 'vegan'

15. Graphics

Standard plots	<code>plot(x, y)</code>	Scatterplot (or other type of plot if x and y are not numeric vectors)
	<code>plot(f, y)</code>	Set of boxplots for each level of factor f
	<code>hist()</code>	Histogram
	<code>boxplot()</code>	Boxplot
	<code>barplot()</code>	Bar diagram
	<code>dotplot()</code>	Dot diagram
	<code>piechart()</code>	Pie chart
	<code>interaction.plot()</code>	Interaction plot (analysis of variance)
Plotting elements (adding to a plot)	<code>lines()</code>	Lines
	<code>abline()</code>	Regression line
	<code>points()</code>	Points
	<code>arrows()</code>	Arrows (NB: <code>angle = 90</code> for error bars)
	<code>box()</code>	Frame around plot
	<code>title()</code>	Title (above plot)
	<code>text()</code>	Text in plot
	<code>mtext()</code>	Text in margin
	<code>legend()</code>	List of symbols
Graphical pars.: arguments to <code>par()</code>	<code>pch</code>	Symbol (see below)
	<code>mfrow, mfcol</code>	Several plots in one (multiframe)
	<code>xlim, ylim</code>	Plot limits
	<code>lty, lwd</code>	Line type / width (see below)
	<code>col</code>	Color for lines or symbols (see below)

Point symbols (pch)

□ ○ △ + × ◇ ▽ ▨ ✱ ⋈ ⊕ ⊗ ⊞ ⊠ ⊡ ⊢ ⊣ ⊤ ⊥ ⊦ ⊧ ⊨ ⊩ ⊪ ⊫ ⊬ ⊭ ⊮ ⊯ ⊰ ⊱ ⊲ ⊳ ⊴ ⊵ ⊶ ⊷ ⊸ ⊹ ⊺ ⊻ ⊼ ⊽ ⊾ ⊿ ⊿ ⊿ ⊿ ⊿
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Colors (col)

- 1 black
- 2 red
- 3 green
- 4 blue
- 5 light blue
- 6 purple
- 7 yellow
- 8 grey

Line types (lty)