

# Machine Learning For Architects

Morteza Khorsand



## **Contentss**

1. Python (from biggening to advanced)
2. Linear Regression
3. Polynomial Regression
4. Locally weighted Regression
5. K-Neighbors Classifier (KNN)
6. Decision Tree
7. Random Forest
8. Naïve Bayse
9. Support Vector Machine (SVM)
10. Logistic Regression
11. Soft-max Classifier
12. Artificial Neural Network (ANN)
13. K Means
14. K Mode
15. K Means ++
16. K medoids
17. Dimensionality Reduction
18. Fuzzy Clustering
19. Hierarchical Clustering
20. DBSCAN
21. Anomaly Detection (PCA)
22. Advanced Optimization Methods
23. Regularization (Overfitting)
24. Covariance
25. Correlation

## variables

```
int
float
string

In [1]: a= 12
In [2]: b=12.3
In [3]: c="kitchen"
In [4]: num=2
print(num)
2
In [5]: odd_number = 13          #snake case
In [6]: oddNumber = 15          # camel case
In [7]: oddnumber= 17           #lower case
In [8]: OddNumber= 23           #upper camel case
In [ ]: """
1. start with alphabet
2. do not start with number
3. % , $ , * , & is not acceptable
4. keyword
""";
```

## Python keywords

```
In [9]: import keyword as kw
print(kw.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', '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']
```

## Arithmetic Operation

addition (+), subtraction (-), division (/), multiplication (\*), power , quotient , remainder

```
In [10]: num_1 = 2
num_2=5.
num_3=4.

In [11]: add= num_1 + num_2
print(add)
7.0

In [12]: mul=num_3 * num_1
print(mul)
8.0

In [13]: quo=num_2 // num_1
print(quo)
2.0

In [14]: re=num_3 % 1.5
print(re)
1.0

In [15]: po=num_1**2
print(po)
4

In [16]: var_1=(num_1 * num_2)**2+ num_3
var_2= num_1 * (num_2**2+ num_3)
#var_2=num_1 * (num_2+ num_3)**2
print(var_1 , var_2)
104.0 58.0
```

```
In [17]: str_1 = "mortezza"
str_2 = "ali"
str_3= str_1+ str_2
print(str_3)

print(str_1*str_2)
mortezzaali
-----
TypeError                                 Traceback (most recent call last)
Cell In[17], line 6
      3 str_3= str_1+ str_2
      4 print(str_3)
----> 6 print(str_1*str_2)

TypeError: can't multiply sequence by non-int of type 'str'
```

```
In [ ]: str_1 **2
str_1/2

In [18]: a="2"
print(type(a))
<class 'str'>

In [19]: str_1**2
```

```

-----
TypeError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 str_1**2

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'

In [20]: x = 12
x*13
print(x)

13

In [21]: #x=x+7
x+=7
print(x)

20

```

## comparison operator

> , < , == , !=

```

In [22]: number_1= 60      #assignment and declaration  int number1;
          # number1=60;

In [23]: number_1 > number_2
Out[23]: True

In [24]: number_1<= number_2
Out[24]: False

In [25]: number_1==number_2      #boolean
Out[25]: False

In [26]: number_1!= 10
Out[26]: True

```

## Data Structures

1. list
2. string
3. tuple
4. dictionary
5. set

### list

```

In [27]: number_list=[1,10,45,89,75,46]
name_list=["kitchen", "bedroom", "diningroom", "livingroom"]

```

```
In [28]: list_1= ["python" , 12. , 0 , "#" , "introduction to python for architects", "grasshopper", True , "def" ]
```

```
In [29]: print(list_1[-3])
```

grasshopper

```
In [30]: list_1
```

```
Out[30]: ['python',
12.0,
0,
 '#',
'introduction to python for architects',
'grasshopper',
True,
'def']
```

```
In [31]: list_1[-6]           #memory
```

```
Out[31]: 0
```

```
In [32]: list_1[1:5]
```

```
Out[32]: [12.0, 0, '#', 'introduction to python for architects']
```

```
In [33]: list_1[3:6]
```

```
Out[33]: ['#', 'introduction to python for architects', 'grasshopper']
```

```
In [34]: list_1[0:6:2]
```

```
Out[34]: ['python', 0, 'introduction to python for architects']
```

```
In [35]: list_1[-1 : -4:-1]
```

```

Out[35]: ['def', True, 'grasshopper']

In [36]: list_1[:]

Out[36]: ['python',
12.0,
0,
'',
'#',
'introduction to python for architects',
'grasshopper',
True,
'def']

In [37]: list_1[: : 2]

Out[37]: ['python', 0, 'introduction to python for architects', True]

In [38]: #exercise
#reverse list

num=[1,2,3,4,5]
num[: :-1]

Out[38]: [5, 4, 3, 2, 1]

In [39]: list_1[len(list_1)-1]

Out[39]: 'def'

In [40]: #print()    #built in function
type(list_1)
len(list_1)

Out[40]: 8

```

## Nested list

```

In [41]: buildings = ["livingroom", ["bedroom", "masterbedroom"], ["W.C", "bath"]]

In [42]: buildings[2]      #int a = 12
Out[42]: ['W.C', 'bath']

In [43]: buildings[1][1]
Out[43]: 'masterbedroom'

In [44]: buildings[2][0][2]
Out[44]: 'C'

In [45]: matrix = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
matrix[1][1]

matrix[1][-1]

Out[45]: 6

```

## updates in lists

```

In [46]: scores= [1,20,19,18,14,10,5,6,19,20,17,15,14,12,13,16,14]

In [47]: scores[0]= 18                      #mutable
print(scores)

[18, 20, 19, 18, 14, 10, 5, 6, 19, 20, 17, 15, 14, 12, 13, 16, 14]

In [48]: scores[1:6] = 14,15
scores

Out[48]: [18, 14, 15, 5, 6, 19, 20, 17, 15, 14, 12, 13, 16, 14]

In [49]: scores[1:3]=[]
scores

Out[49]: [18, 5, 6, 19, 20, 17, 15, 14, 12, 13, 16, 14]

In [50]: scores[1]=[]
scores

Out[50]: [18, [], 6, 19, 20, 17, 15, 14, 12, 13, 16, 14]

In [51]: scores[:] = []
scores

Out[51]: []

In [52]: numbers_1 = [1,2,3,8]
numbers_2= [4,5,6]
numbers_1 + numbers_2

Out[52]: [1, 2, 3, 8, 4, 5, 6]

In [53]: #numbers_1 + 2
type(numbers_1)

Out[53]: list

In [54]: numbers_1 *2
numbers_1 *2

Out[54]: [1, 2, 3, 8, 1, 2, 3, 8]

In [55]: print(len(numbers_1*2))
8

In [56]: #multiply each number in a list by 2
list_2= [4,7,10]

```

## list methods

```
In [57]: student_scores=[12,15,18,17,19,15,14,16,20,8,8,8]
print(id(student_scores))

student_scores.reverse()

print(id(student_scores))
1883366017024
1883366017024

In [58]: student_scores.append(0)
student_scores

student_scores.append(10)
student_scores

print(id(student_scores))
student_scores

1883366017024
Out[58]: [8, 8, 8, 20, 16, 14, 15, 19, 17, 18, 15, 12, 0, 10]

In [59]: student_scores.remove(8)
student_scores

Out[59]: [8, 8, 20, 16, 14, 15, 19, 17, 18, 15, 12, 0, 10]

In [60]: student_scores=[12,15,18,17,19,15,14,16,20,8]
t=student_scores.index(20)
print(t)

8

In [61]: student_scores.extend([12 , 13])      #student_scores+=[12 , 13]
student_scores

Out[61]: [12, 15, 18, 17, 19, 15, 14, 16, 20, 8, 12, 13]

In [62]: student_scores.sort()
student_scores

Out[62]: [8, 12, 12, 13, 14, 15, 15, 16, 17, 18, 19, 20]

In [63]: student_scores.sort(reverse= True)      #key= Len
student_scores

Out[63]: [20, 19, 18, 17, 16, 15, 15, 14, 13, 12, 12, 8]

In [64]: list_name= ["python" , "java", "kotlin", "csharp"]
list_name.sort(reverse= True)
list_name

Out[64]: ['python', 'kotlin', 'java', 'csharp']

In [65]: list_name2= ["python" , "java", "kotlin", "csharp" , 1,12,15]
list_name2.sort(reverse= True)
list_name2

-----
TypeError                                 Traceback (most recent call last)
Cell In[65], line 2
  1 list_name2= ["python" , "java", "kotlin", "csharp" , 1,12,15]
  ----> 2 list_name2.sort(reverse= True)
      3 list_name2
      ...
TypeError: '<' not supported between instances of 'str' and 'int'

In [66]: a=sorted(student_scores, reverse= True)
a

Out[66]: [20, 19, 18, 17, 16, 15, 15, 14, 13, 12, 12, 8]

In [67]: student_scores.insert(5 , 14)    #insert 14 in 0 index
student_scores

Out[67]: [20, 19, 18, 17, 16, 14, 15, 15, 14, 13, 12, 12, 8]

In [68]: student_scores.pop()        #remove   #pop has return
student_scores

print(id(student_scores))
student_scores

1883366076224
Out[68]: [20, 19, 18, 17, 16, 14, 15, 15, 14, 13, 12, 12]

In [69]: a=student_scores.copy()      #in python3
print(id(a))
1883366297472

In [70]: s= sum(student_scores)      #function
print(" sum of the scores are:",s)
sum of the scores are: 185

In [71]: ma= max(student_scores)
mi=min(student_scores)
mi

Out[71]: 12

In [72]: list_3=[]
list_3.append(10)
list_3

Out[72]: [10]

In [ ]:

#exercise create a list of scores remove the lowest and highest number in the list calculate the average of reminders (we need the average original list too) ----- specify a list and shows the second minimum of it ----- specify a list of numbers and calculate the sum of maximum and minimum numbers and append it an empty list -----
-
In [73]: a = ['x','y','z']
a.clear()
```

```
del(a)
a

-----
NameError
Cell In[73], line 6
  2 a.clear()
  4 del(a)
----> 6 a

NameError: name 'a' is not defined
```

## string

```
In [74]: seq_1= "python"
seq_2= "for architects"

print(id(seq_1) , id(seq_2))
1885447398960 1883366360176

In [75]: print(seq_1 + seq_2)
seqt= seq_1 + seq_2
print(seqt)

pythonfor architects
pythonfor architects

In [76]: seq_3 = "machine learning for architectural students"
s_7=seq_3[-2:-8:-1]

s_7
```

Out[76]: 'tnedut'

```
In [77]: s_3= seq_3[16]
print(s_3)
```

```
In [78]: s_4= seq_3[5:12]
print(s_4)

ne lear
```

```
In [79]: s_5= seq_3[1:15:2]
s_5
```

Out[79]: 'ahn eri'

```
In [80]: s_6=seq_3[-1 : 5: -1 ]
s_6
```

Out[80]: 'stneduts larutcetihcra rof gminrael e'

```
In [81]: seq_3[0] ="k"
```

```
-----
TypeError
Cell In[81], line 1
----> 1 seq_3[0] ="k"

TypeError: 'str' object does not support item assignment
```

```
In [82]: #methods
```

```
seq_3 = "machinelearning for architects"
a=seq_3.upper()                                #convert string to uppercase
print(seq_3.lower())                            #convert string to lowercase
print(seq_3.count("o"))                         #returns the number of times a specific value occurs in a string
print(seq_3.endswith("archItects"))              #returns true if the string ends with especific value or values
print(seq_3.find("e",12))                       #searching the string for a specific value and returns the position
print(seq_3.find("e", 5, 10))
print(seq_3.find("for"))
print(seq_3.index("e",5))
b=seq_3.replace("t", "H")                      #replace t with H
print(seq_3.split(sep="e"))
print(seq_3.split(sep="e", maxsplit= 2))
print(seq_3.title())                           #converts the first character of each word to uppercase

print(a)
print(b)

machinelearning for architects
0
False
26
6
16
25
['machin', 'l', 'arning for archit', 'cts']
['machinelearning for archi', 'ec', 's']
['machin', 'l', 'arning for architects']
Machinelearning For Architects
MACHINELEARNING FOR ARCHITECTS
machinelearning for archiHecHs
```

Memory	
type	int
value	343
id/address	0x1002

#

```
In [83]: print(id(seq_3))
1883366451488

In [84]: a=seq_3.upper()
print(id(a))
1883366333440

In [85]: print(int(input("please enter your number: ")))
please enter your number:
-----
ValueError                                Traceback (most recent call last)
Cell In[85], line 1
----> 1 print(int(input("please enter your number: ")))

ValueError: invalid literal for int() with base 10: ''
```

```
In [ ]: #triangle square
base= float(input("the base is: "))
height= float(input("the hight is: "))
squire= base* height/2
print("the squire is :", squire)
```

## for loop

```
In [ ]: #multiply by 2 manually
lst=[1,2,3]
lst[0]=lst[0]*2
lst[1]=lst[1]*2
lst[2]=lst[2]*2

print(lst)
```

for iterator in iterable: statement

```
In [86]: list1=[1,2,3,4,5,6,7,8,9,10]

for i in list1:
    print(i)

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

```
In [87]: list_num = [45,67,25,41,10,36,15,47,89,45,74,14,15,25]

list_2=[]

for num in list_num:
    a=num*2
    list_2.append(a)

print(list_2)
[90, 134, 50, 82, 20, 72, 30, 94, 178, 90, 148, 28, 30, 50]
```

```
In [88]: empty_list=[]

for j in list_num:
    empty_list.append(j*2)

empty_list
Out[88]: [1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1]
```

```
In [89]: #sort the length of char from biggest to shortest
list_name= ["python" , "java", "JavaScript" , "c_sharp" , "C++" , "SQL","MATLAB","Kotlin"]

length=[]

for char in list_name:
    length.append(len(char))

length.sort(reverse= True)

print(length)
[10, 7, 6, 6, 6, 4, 3, 3]
```

```
In [90]: list_2=[]
for i in list_name:
    a=i.upper()
    list_2.append(a)

list_2
```

```
Out[90]: ['PYTHON', 'JAVA', 'JAVASCRIPT', 'C_SHARP', 'C++', 'SQL', 'MATLAB', 'KOTLIN']
```

## range

```
range(start, end, step) for i in range(1,5,1): pass
```

```
In [91]: list_2=[]  
for x in range(0,10):  
    a=(x**2)  
    list_2.append(a)  
  
list_2
```

```
Out[91]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [92]: for i in range(10):  
    a=i**2  
    #print(a)  
  
    print(a)  
81
```

```
In [93]: #nested Loop  
  
numbers=[]  
for i in range(5):  
    for j in range(5):  
        numbers.append(i + j)  
  
print(numbers)
```

```
[0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
```

```
In [94]: list_total=[]  
list_1 = [1,2,3]  
list_2 = [4,5,6]  
  
for x in list_1:  
    for y in list_2:  
        list_total.append(x+y)  
print(list_total)
```

```
[5, 6, 7, 6, 7, 8, 7, 8, 9]
```

```
In [95]: string= "grasshopper and rhino"  
  
list_st=[]  
for i in range(len(string)):  
    list_st.append(string[i])  
print(list_st)
```

```
['g', 'r', 'a', 's', 's', 'h', 'o', 'p', 'p', 'e', 'r', ' ', 'a', 'n', 'd', ' ', 'r', 'h', 'i', 'n', 'o']
```

```
In [96]: list1=[10,12,14,15]  
  
for j in range(len(list1)):  
    print(list1[j])
```

```
10  
12  
14  
15
```

```
In [97]: for i in range(10):  
    pass
```

## from, import

```
In [98]: import math  
import random as rnd  
  
from math import sin, cos , pi  
#from random import randint  
  
rnd.randrange(1,10)
```

```
4
```

```
#exercise -01 5 x 1 = 5 x 2 = 10 5 x 3 = 15 5 x 4 = 20 5 x 5 = 25 5 x 6 = 30 5 x 7 = 35 5 x 8 = 40 5 x 9 = 45 5 x 10 = 50 #exercise 02 - performing sum of first 10 numbers #exercise 03 - show each item separately - lst = [[1,2,3], ["land", "sea", "sky"]] #exercise 04- combine one by one colors=["yellow", "black", "green", "purple"] fruits=["mango", "banana", "apple", "eggplant"] like: yellow mango #exercise 05 - count the length of the list by for loop - numbers = [12,3,56,67,89,90] #exercise 06 - sum of all numbers in a list - numbers = [12,3,56,67,89,90] #exercise 07 - find the multiples of 5 in a list - numbers = [2,5,6,10,15,20,25] #exercise 08 - multiples of 5 using range() between 0-100
```

## Conditional statement

```
if condition : statement
```

```
In [99]: num = 10  
  
if num > 0 :  
    print("the number is positive:" ,num)
```

```
the number is positive: 10
```

```
In [100]: number= int(input("enter the number: "))  
  
if number%2==0:  
    print("the number is even" )           #display~ print
```

```
enter the number: 45
```

```
In [102]: weight=float(input("your weight"))  
  
if weight >= 90:  
    print("your weight (weight) is overweight")  
    print("your weight is:" , weight)  
  
if weight <= 60:  
    print("your weight (weight) is underweight")  
  
if 60<weight<90:  
    print("your weight (weight) is normal")
```

```
your weight82  
your weight 82.0 is normal
```

```
In [103...]
weight=float(input("your wight"))

if weight >= 90:
    print("overwight")
elif weight <= 60:
    print("underwight")
elif 60<weight<90:
    print("normal")

your wight52
underwight

In [104...]
score= float(input("enter your score"))

if score >=18 :
    print(f" your number {score} is A")
elif score >=16:
    print(f" your number {score} is B")
elif score >=14:
    print(f" your number {score} is C")
else:
    print(f" your number {score} is D")

enter your score41
your number 41.0 is A

In [105...]
score= float(input("enter your score"))
if score >=18 :
    print(f" your number {score} is A")
if score >=16:
    print(f" your number {score} is B")
if score >=14:
    print(f" your number {score} is C")

enter your score18
your number 18.0 is A
your number 18.0 is B
your number 18.0 is C

In [107...]
number= int(input("enter number: "))

if number %2 ==0:
    print("the number is even")
elif number %2!=0:
    print("the number is odd")

enter number: 18
the number is even

In [109...]
number= int(input("enter number: "))
if number %2 ==0:
    print("the number is even")
else:
    print("the number is odd")

enter number: 12
the number is even

In [110...]
#even number smaller than 100

n= int(input("your number: "))
if n <= 100:
    if n%2==0:
        print(f" the number {n} is correct")

your number: 11

In [ ]:
#exercise
##even number smaller than 100 and coefficient of 3

In [111...]
height= float(input(" your hight(m) : "))
weight= float(input(" your weight(kg) : "))
BMI= weight / height**2

#<18.5 --- undewight
#18.5-24.9 --- normal
#25-29.9---- overweight
#else : obess

print(BMI)

your hight(m) :  120
your weight(kg) : 50
0.003472222222222222

In [112...]
#exercise
r="rock"
p="paper"
s="scissors"
player_1= input(f"{r} or {p} or {s}: ")
player_2= input(f"{r} or {p} or {s}: ")
if player_1 == r and player_2==p:
    print("player_2 won")
elif player_1==r and player_2==s:
    print("player_1 won")
elif player_1 ==player_2:
    print("that is a tie")
elif player_1==p and player_2==r:
    print("player_1 won")
elif player_1==p and player_2==s:
    print("player_2 won")
elif player_1==s and player_2==r:
    print("player_2 won")
elif player_1==s and player_2==p:
    print("player_1 won")
else:
    print("something is wrong")

rock or paper or scissors: r
rock or paper or scissors: p
something is wrong

In [113...]
even_numbers=[]
odd_numbers=[]
for i in range(100):
    if i%2==0:
        even_numbers.append(i)
    elif i%2!=0:
        odd_numbers.append(i)

print(even_numbers)
print(odd_numbers)

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]
```

```
In [114]: #how many 10 exist in the list
list_number_1=[12,15,10,14,17,10,78,98,96,85,74,12,10,14,78,54,123
             ,45,10,41,12,36,98,74,10,15,14,17,89,
             12, 10,45,78,10,98,639,47,10,78,10,78,96,85]
counter=0
for num in list_number_1:
    if num==10:
        counter+=1      #counter= counter+1      additional assignment

In [ ]: #even numbers in the list above
#odd numbers in the list above
# multiples of 3 in the list above

#home exercise # 01. Write a Python program that iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for multiples of five print "Buzz". For numbers that are multiples of three and five, print "FizzBuzz". Sample Output : fizzbuzz 1 2 fizz 4 buzz ----- #02. Write a Python program that accepts a string and calculates the number of digits and letters. Sample Data : Python 3.2
Expected Output : Letters 6 Digits 2 #guide: use isdigit() , isalpha() ----- #03. Write a Python program to check if a triangle is equilateral, isosceles or scalene. Note : An equilateral triangle is a triangle in which all three sides are equal. A scalene triangle is a triangle that has three unequal sides. An isosceles triangle is a triangle with (at least) two equal sides. Expected Output: Input lengths of the triangle sides: x: 6 y: 8 z: 12 Scalene triangle ----- #04 . Write a Python program to construct the following pattern, using a nested loop number. Expected Output: 1 22 333 4444 55555 666666 777777
88888888 999999999
```

## break and continue

```
In [115]: for i in range(1,11):
    if i == 5:
        break
    print(i)

1
2
3
4
5
6
7
8
9
10
Out[115]: 'when the condition is met, that iteration is skipped.\nBut we do not exit the loop. Hence,\nall the values except 5 are printed out.\n'

In [116]: for i in range(1,11):
    if i == 5:
        continue
    print(i)

1
2
3
4
6
7
8
9
10
Out[116]: 'when the condition is met, that iteration is skipped.\nBut we do not exit the loop. Hence, \nall the values except 5 are printed out.\n'

In [117]: for i in range(1,11):
    if i == 5:
        pass
    print(i)

1
2
3
4
5
6
7
8
9
10
Out[117]: 'when the condition is met, that iteration is skipped.\nBut we do not exit the loop. Hence, \nall the values except 5 are printed out.\n'

In [118]: for i in range(10):
    pass

a=12
b=13
In [119]: s = 'geeksforgeeks'
for letter in s:
    # break the Loop as soon it sees 'e'
    # or 's'
    if letter == 'e' or letter == 's':
        break
    print(letter)

g
```

## Logical Operators

Operator	Example	Meaning
and	a and b	Logical AND: True only if both the operands are True
or	a or b	Logical OR: True if at least one of the operands is True
not	not a	Logical NOT: True if the operand is False and vice-versa.

```
In [124]: number=int(input(" number"))
if number >3 and number%2==0:
    print(True)
else:
    print(False)
number=45
False

In [125]: x=10 #hard code
y=15
if (x>7 and y >8):
    print("thats true")
if (x>14 or y>14):
    print("accepted")
thats true
accepted

In [128]: number= int(input("enter your number"))

if (number%2==0 and number<100 and number%3==0):
    print(f"the number {number} has all the condition")
enter your number12
the number 12 has all the condition

In [129]: list_number=[12,45,78,100,25,13,10,78,95]
factor=[]
for num in list_number:
    if num%2==0 and num%3==0:
        factor.append(num)

factor
Out[129]: [12, 78, 78]

In [130]: e_list=[]
for num in list_number:
    if not num%2==0:
        e_list.append(num)
e_list
Out[130]: [45, 25, 13, 95]

In [131]: scored1=[]
scored2=[]
x=float(input("enter the number"))
#x<0 or x>20
if x <0 or x>20:
    print("wrong")
elif x<10:
    scored1.append(x)
    print(f"you failed and your score is {scored1}")
else:
    scored2.append(x)
    print(f"you passed and your score is{scored2}")

scored2
enter the number20
you passed and your score is[20.0]
Out[131]: [20.0]

In [132]: list1=[5,20,15,20,25,50,20]
for num in list1:
    if num==20:
        list1.remove(num)

list1
Out[132]: [5, 15, 25, 50]
```

## Membership operators

```
In [133]: a = [1, 2, 3, 4, 5]
print(5 in a)

10 in a
True
False
Out[133]: 

In [134]: message = 'Hello world'
print('H' in message)
print('python' not in message)
```

```
True  
True
```

## Identity operators

```
In [135...]  
x1 = 5          # identity, not just equality(==)  
y1 = 5  
x2 = 'Hello'  
y2 = 'Hello'  
x3 = [1,2,3]  
  
y3 = [1,2,3]  
  
s1="check if 'H' is present in message string"  
s2="check if 'H' is present in message string"  
  
print(x1 is y1)  
  
print(x2 is y2)  
  
print(x3 is y3)  
  
print(s1 == s2)  
  
print(s1 is s2)  
  
print(id(x3) , id(y3))  
  
True  
True  
False  
True  
False  
1883366074176 1883366896512
```

```
In [136...]  
print(id(s1))  
print(id(s2))  
1883366965488  
1883366981424
```

## Random Numbers

```
In [137...]  
#import random  
  
#https://www.aparat.com/v/V7Yki  
from random import random ,seed      #real numbers between 0 , 1  
  
#seed(5)  
#print(random()*10)  
  
  
seed(2)  
print(random())           #print(random.random())  
0.9560342718892494
```

```
In [138...]  
random_numbers=[]  
  
#seed(5)  
for _ in range(10):  
    random_numbers.append(random())  
  
print(random_numbers )  
[0.9478274870593494, 0.05655136772680869, 0.08487199515892163, 0.8354988781294496, 0.7359699890685233, 0.6697304014402209, 0.3081364575891442, 0.6059441656784624, 0.6068017336408379, 0.5812040171120031]
```

```
In [139...]  
from random import randint , seed      #integer numbers between (a, b)  
  
random_integers=[]  
#seed(2)  
for _ in range(10):  
    random_integers.append(randint(5,9))  
  
print(random_integers)  
  
#remove repetitive numbers  
u=[]  
for i in range(len(random_integers)):  
    if random_integers[i] not in u:  
        u.append(random_integers[i])  
u  
[6, 8, 8, 9, 7, 9, 8, 9, 7, 5]  
Out[139]: [6, 8, 9, 7, 5]
```

```
In [140...]  
list8=[8, 9, 7, 9, 5, 7, 5, 6, 5, 5]  
for i in range(len(list8)):  
    print(list8[i])
```

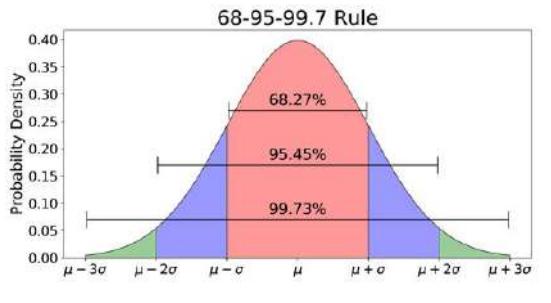
```
8  
9  
7  
9  
5  
7  
5  
6  
5  
5
```

```
In [141...]  
from random import randrange  
a=randrange(2,20,2)  
a
```

```
Out[141]: 2
```

```
In [142...]  
from random import uniform  
uniform(2,6)  
  
uniform(0 , 1)
```

```
Out[142]: 0.9318465539698684
```



```
In [143]: import random          # gaussian randoms with mu and sigma
guss_list=[]
random.seed(5)
for _ in range(12):
    guss_list.append(random.gauss(1 , 1))
guss_list
```

```
Out[143]: [-0.1788417512306717,
-0.14816868079880158,
1.6694689143859696,
-1.293918094631911,
0.8566161616995311,
-1.2568772673958667,
2.1009715963243587,
1.2028981117525226,
2.3563159867998544,
0.49581743802085077,
1.3981949121429393,
0.7141226372189547]
```

```
In [144]: from random import seed      #choose a sublist from a List
from random import choice
point_list=[[0,0,0], [2,3,6], [5,4,8], [1,2,-5], [8,7,4],[1,5,6]]
r_list=[]
for _ in range(3):
    r_list.append(choice(point_list))
r_list
```

```
Out[144]: [[1, 2, -5], [2, 3, 6], [1, 2, -5]]
```

```
In [145]: from random import seed      #choose a sublist from a List
from random import sample
point_list=[[0,0,0], [2,3,6], [5,4,8], [1,2,-5], [8,7,4],[1,5,6]]
seed(1)
print(sample(point_list,2))
```

```
[[2, 3, 6], [8, 7, 4]]
```

```
In [146]: import random
list_number=[1,2,3,4,5,6,7,8,9,10]
random.shuffle(list_number)
list_number
```

```
Out[146]: [8, 3, 1, 9, 6, 7, 4, 10, 5, 2]
```

```
In [147]: import random
computer_choice = random.randrange(0,3)
if computer_choice==0:
    computer_choice="rock"
elif computer_choice==1:
    computer_choice="paper"
elif computer_choice==2:
    computer_choice="scissors"
r="rock"
p="paper"
s="scissors"
player_name=input("enter your name to start the game:")
player_1=input("{player_name} {r} or {p} or {s}: ")
if player_1==r and computer_choice=="paper":
    print(f"computer choice is {computer_choice} and computer won")
elif player_1==r and computer_choice == "scissors":
    print(f"computer choice is {computer_choice} and {player_name} won")
elif player_1 ==computer_choice:
    print(f"computer choice is {computer_choice} and thats a tie")
elif player_1==p and computer_choice==r:
    print(f"computer choice is {computer_choice} and {player_name} won")
elif player_1==p and computer_choice==s:
    print(f"computer choice is {computer_choice} and computer won")
elif player_1==s and computer_choice==r:
    print(f"computer choice is {computer_choice} and computer won")
elif player_1==s and computer_choice==p:
    print(f"computer choice is {computer_choice} and {player_name} won")
else:
    print("something is wrong")
```

```
enter your name to start the game:
rock or paper or scissors:
something is wrong
```

```
In [149]: from random import randrange
counter=0
for _ in range(1000):
    a=randrange(1,7)
    if a%6==0:
        counter+=1
print(counter)

#exercise : how many 2
```

178

```
In [150]: #1M random numbers and specify how many tails a and how many head
heads=0
tails=0
from random import randint
```

```

for _ in range(1000000):
    a=randint(0,1)
    if a==0:
        heads+=1
    else:
        tails+=1
print(heads , tails)
500588 499412

```

## Tuple

```
In [151]: tuple_1= ('python' , 12. , 0 , "#" , "introduction to python for architects" , "grasshopper")
```

```
In [152]: import sys
tuple_2= "java" , 18 , "rhino" , 17.36 , "grasshopper"
sys.getsizeof(tuple_2)
```

```
Out[152]: 80
```

```
In [153]: list1=[ "java" , 18 , "rhino" , 17.36 , "grasshopper"]
```

```
list1[0]= "c++"
list1
```

```
import sys
sys.getsizeof(list1)
```

```
list1
```

```
Out[153]: ['c++', 18, 'rhino', 17.36, 'grasshopper']
```

```
In [154]: tuple_2[1] = 200      # tuple is immutable
```

```
-----
TypeError                                 Traceback (most recent call last)
Cell In[154], line 1
----> 1 tuple_2[1] = 200
      TypeError: 'tuple' object does not support item assignment
```

```
In [155]: b=tuple_1[3]
```

```
b
```

```
Out[155]: '#'
```

```
In [156]: tuple_1[1:3]
```

```
Out[156]: (12.0, 0)
```

```
In [157]: tuple_1[:]
```

```
Out[157]: ('python',
12.0,
0,
'#',
'introduction to python for architects',
'grasshopper')
```

```
In [158]: tuple_1[-1:-5:-1]
```

```
Out[158]: ('grasshopper', 'introduction to python for architects', '#', 0)
```

```
In [159]: tuple_3= (20,
```

```
a=12
```

```
In [160]: test_list = [5, 6, 7]
test_tup = (9, 10)
```

```
test_list.extend(test_tup)

#test_list+=test_tup
test_list
```

```
Out[160]: [5, 6, 7, 9, 10]
```

```
In [161]: test_list = [5, 6, 7]
list2=[7,10,15]
```

```
test_list.extend(list2)

test_list
```

```
Out[161]: [5, 6, 7, 7, 10, 15]
```

```
In [162]: test_list*list2
```

```
Out[162]: [5, 6, 7, 7, 10, 15, 7, 10, 15]
```

## Nested tuple

```
In [163]: tuple_4=(( "rhino" , "grasshopper") , ("3dmax" , "vray" , "corona") , ("revit" , "dynamo"))

tuple_4[0]
```

```
Out[163]: ('rhino', 'grasshopper')
```

```
In [164]: tuple_4[0][0][0]
```

```
'r'
```

## methods in tuple

```
In [165]: tuple_5= (1,2,3,12,1,45,87,96,36,12,10,10,52,45,45,2,1,96)
```

```
In [166]: t=tuple_5.count(1)
print(t)
```

```
3
```

```
In [167]: tuple_5.index(45,10)
```

```
Out[167]: 13
```

## built-in function in tuple

```
In [168... sum(tuple_5)
Out[168]: 556

In [169... min(tuple_5)
Out[169]: 1

In [170... max(tuple_5)
Out[170]: 96

In [171... len(tuple_5)
Out[171]: 18

In [172... sorted(tuple_5, reverse=True)
Out[172]: [96, 96, 87, 52, 45, 45, 45, 36, 12, 12, 10, 10, 3, 2, 2, 1, 1, 1]

In [173... print(tuple_5)
(1, 2, 3, 12, 1, 45, 87, 96, 36, 12, 10, 10, 52, 45, 45, 2, 1, 96)

#exercise tuple1 = ("Orange", [10, 20, 30], (5, 15, 25)) #access 20 ..... #Unpack the tuple into 4 variables tuple1 = (10, 20, 30, 40) ..... #input..... listt = [1, 2, 3] output.....[(1, 1), (2, 8), (3, 27)]
..... ## initializing list test_list = [5, 6, 7] # initializing tuple test_tup = (9, 10) output -----[5, 6, 7, 9, 10] ..... a=12 a,b = 12 , 13
```

## Set

```
In [174... set1={"python", "grasshopper", 12 , 45.36, "$" }
Out[174]: {'$': 45.36, 12, 'python', 'grasshopper'}

In [175... set1[1]
-----+-----+
TypeError                                 Traceback (most recent call last)
Cell In[175], line 1
----> 1 set1[1]

TypeError: 'set' object is not subscriptable

In [176... set1["python"] = "rhino"
-----+-----+
TypeError                                 Traceback (most recent call last)
Cell In[176], line 1
----> 1 set1["python"] = "rhino"

TypeError: 'set' object does not support item assignment

In [177... set2={1,2,3,5,5,6}
set2
Out[177]: {1, 2, 3, 5, 6}

In [178... list_1=[1,2,3,4,5,6,6,7,8,8,9,10]           #change List to set
list_2=[]
for i in (list_1):
    if i not in list_2:
        list_2.append(i)
list_2
Out[178]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [179... numbers= list(set(list_1))
numbers
Out[179]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [180... import time
start_time= time.time()
list3=[]
for i in range(50000):
    if i not in list3:
        list3.append(i)
end_time=time.time()
print(end_time-start_time)
25.31426763534546

In [181... start_time= time.time()
u={i for i in range(50000)}
numbers= list(set(u))
end_time=time.time()
print(end_time-start_time)
0.004988431930541992
```

## built-in methods in sets

```
In [182... set_new={1,5,6,9,8,7,14,78,96,52,10}
set_new.remove(14)
set_new
Out[182]: {1, 5, 6, 7, 8, 9, 10, 52, 78, 96}

In [183... set_new.add(13)
set_new
Out[183]: {1, 5, 6, 7, 8, 9, 10, 13, 52, 78, 96}

In [184... set_new.discard(10)          #remove a data if exist
set_new
Out[184]: {1, 5, 6, 7, 8, 9, 13, 52, 78, 96}
```

```
In [185... set_new.pop()
set_new
Out[185]: {1, 5, 6, 7, 8, 9, 13, 52, 78}

In [186... set_new.clear()
set_new
del(set_new)
set_new
-----
NameError: Traceback (most recent call last)
Cell In[186], line 7
  3 set_new
  5 del(set_new)
----> 7 set_new
NameError: name 'set_new' is not defined

In [187... M1 ={12,13,15,14,17,19}
M2={12,85,15,14,17,10,23}
M1.union(M2)      # M1|M2
Out[187]: {10, 12, 13, 14, 15, 17, 19, 23, 85}

In [188... M1.intersection(M2)    #M1&M2
Out[188]: {12, 14, 15, 17}
```

## Dictionaries

```
In [189... dic={0: "python" , 1: "grasshopper" , 2: "rhino" , 3: "machine"}
list_1= ["python" , "grasshopper","rhino","machine "]

In [190... print(list_1[0])
print(dic[0])
python
python

In [191... dic_2= {"car": "bmw" , "salary":12000, "education": "architecture" }

#key ---- value

In [192... dic_2["salary"]
12000
Out[192]: 12000

In [193... dic_2[1]
-----
KeyError: Traceback (most recent call last)
Cell In[193], line 1
----> 1 dic_2[1]
KeyError: 1

In [194... dic_2["car"]= "pride"
dic_2
Out[194]: {'car': 'pride', 'salary': 12000, 'education': 'architecture'}

In [195... car_class = {"brand": "Ford",
                      "model": "Mustang",
                      "year": 1964}

In [196... car_class * 2
-----
TypeError: Traceback (most recent call last)
Cell In[196], line 1
----> 1 car_class * 2
TypeError: unsupported operand type(s) for *: 'dict' and 'int'

In [197... "brand" in car_class
True
Out[197]: True

In [198... "factory" in car_class
Out[198]: False

In [199... "Ford" in car_class
Out[199]: False
```

## Nested Dictionary

```
In [200... season={
    "Spring":["March", "April" , "May"],
    "Summer":["Jun", "July", "August"],
    "Fall": ["September", "October", "November"],
    "Winter": ["December", "January", "February"]
}

season["Fall"][1]
'October'
Out[200]: 'October'
```

## Dictionary methods

```
In [201... print(dir(dict))
['__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__',
 '__hash__', '__init__', '__init_subclass__', '__ior__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
 '__ror__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update',
 'values']
```

```
In [202... Math_commands={0:[0,0,0], 1:[10,10,10] , 2:[20,20,20]}

In [203... a=Math_commands[0]
a

Out[203]: [0, 0, 0]

In [204... Math_commands.pop(0) #The pop() method removes the specified item from the dictionary.

Out[204]: [0, 0, 0]

In [205... Math_commands

Out[205]: {1: [10, 10, 10], 2: [20, 20, 20]}

In [206... Math_commands.clear()          #empty dictionary
Math_commands

Out[206]: {}

In [207... del(Math_commands)

In [208... Math_commands

NameError                                 Traceback (most recent call last)
Cell In[208], line 1
----> 1 Math_commands

NameError: name 'Math_commands' is not defined

In [209... #update method
dic_2={"car": "bmw" , "salary":12000, "education": "architecture" }

print(id(dic_2))
dic_2.update({"car":"samand"})

dic_2.update({"salary": 20000})

print(dic_2)

print(id(dic_2))

1883366527040
{'car': 'samand', 'salary': 20000, 'education': 'architecture'}
1883366527040

In [210... car_class = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}

car_class.update({"color": "White"})
car_class

Out[210]: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}
```

```
In [211... car_class2={
    "colour": "red",
    "door":2
}

In [212... car_class.update(car_class2)
car_class

Out[212]: {'brand': 'Ford',
'model': 'Mustang',
'year': 1964,
'color': 'White',
'colour': 'red',
'door': 2}

In [213... dict2={"colour": "blue" , "code":1245, "size":"large"}

In [214... dict2.values()

Out[214]: dict_values(['blue', 1245, 'large'])

In [215... dict2.keys()

Out[215]: dict_keys(['colour', 'code', 'size'])

In [216... dict2.items()

Out[216]: dict_items([('colour', 'blue'), ('code', 1245), ('size', 'large')])
```

## for in dictionaries

```
In [217... dic_3= {"name": "sara" , "age": 27, "education": "architecture" , "job":"manager" }

In [218... for i in dic_3:
    print(i)

name
age
education
job

In [219... for i in dic_3:      #name
    print(dic_3[i])

sara
27
architecture
manager

In [220... for value in dic_3.values():
    print(value)

sara
27
architecture
manager

In [221... for key in dic_3.keys():
    print(key)
```

```

name
age
education
job

In [222...]: for key,value in dic_3.items():
    print(key , value)

name sara
age 27
education architecture
job manager

```

## Data structures comparison

Set	Dictionary	Tuple	List	Sequence	
{ }	{ }	()	[]	" "	Sign
+	+	-	+	-	Mutable
-	Key	From 0 - n	From 0 - n	From 0 – n	Index
-	+	+	+	+	Repetitive Members
-	+	+	+	+	call

## zip

```

In [223...]: n_tuple=("morteza","maryam","mina")
f_tuple=("akbari","sharifi","karimi")

total=zip(n_tuple, f_tuple )

#print(tuple(total))
#print(list(total))
print(dict(total))

{'morteza': 'akbari', 'maryam': 'sharifi', 'mina': 'karimi'}

```

```

In [224...]: list_n=[1,2,3,4,5]
list_m=[4,5,6,9,7]
list_k= [6,9,8,7,5]

a=zip(list_n , list_m , list_k)
#print(tuple(a))
#print(list(a))

print(dict(a))

```

```

-----+-----+-----+-----+-----+-----+
ValueError                                Traceback (most recent call last)
Cell In[224], line 9
  5 a=zip(list_n , list_m , list_k)
  6 #print(tuple(a))
  7 #print(list(a))
----> 9 print(dict(a))

ValueError: dictionary update sequence element #0 has length 3; 2 is required

```

```

In [225...]: list_k=[1,2,3,4]
list_p=[4,5,6,9,7,10]

b=zip(list_k , list_p)
print(tuple(b))

((1, 4), (2, 5), (3, 6), (4, 9))

```

```

In [226...]: list(zip('abcdefg', range(3), range(4)))
Out[226]: [('a', 0, 0), ('b', 1, 1), ('c', 2, 2)]

```

```

In [227...]: list_n=[1,2,3,4,5]
list_m=[4,5,6,9,7]
for i,j in zip(list_n , list_m):
    print(i ,"-", j )

1 -- 4
2 -- 5
3 -- 6
4 -- 9
5 -- 7

```

```

In [228...]: list_n=[1,2,3,4,5]
list_m=[4,5,6,9,7]
list_k=[8,5,9,7,0]

for i,_ ,m in zip(list_n , list_m , list_k):
    print(i ,m)

1 8
2 5
3 9
4 7
5 0

```

```

In [229...]: list_n=[1,2,3,4,5]
list_m=[4,5,6,9,7]

list_factor=[]

for i,j in zip(list_n , list_m):
    list_factor.append(i*j)

list_factor

```

```

Out[229]: [4, 10, 18, 36, 35] #01-Convert two lists into a dictionary keys = ['Ten', 'Twenty', 'Thirty'] values = [10, 20, 30] output: {'Ten': 10, 'Twenty': 20, 'Thirty': 30} ----- #02- Merge two Python dictionaries into one
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30} dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50} output: {'Ten': 10, 'Twenty': 20, 'Thirty': 30, 'Fourty': 40, 'Fifty': 50} ----- #03 - Delete a list of keys from a dictionary sample_dict = {"name": "Kelly", "age": 25, "salary": 8000, "city": "New York"} # Keys to remove keys = ["name", "salary"] output: {'city': 'New York', 'age': 25} ----- #04- Check if a value exists in a dictionary sample_dict = {'a': 100, 'b': 200, 'c': 300} output: 200 present in a dict ----- #05 - Rename key of a dictionary Write a program to rename a key city to a location in the following dictionary. sample_dict = {"name": "Kelly", "age": 25, "salary": 8000, "city": "New York"} output: {'name': 'Kelly', 'age': 25, 'salary': 8000, 'location': 'New York'} ----- #06 - Change value of a key in a nested dictionary sample_dict = { 'emp1': {'name': 'Jhon', 'salary': 7500}, 'emp2': {'name': 'Emma', 'salary': 8000}, 'emp3': {'name': 'Brad', 'salary': 500 } output : {'emp1':
```

```
{'name': 'Jhon', 'salary': 7500}, {'name': 'Emma', 'salary': 8000}, {'name': 'Brad', 'salary': 8500} } ----- #07- Write a Python program to multiply all the values in a dictionary.  
my_dict = {'data1': 100, 'data2': -54, 'data3': 247} output : -1333800 ----- #08 Write a Python program to check if a dictionary is empty or not
```

```
In [230...]  
my_dict = {'data1': 100, 'data2': -54, 'data3': 247}  
val = 1  
for val in my_dict.values():  
    val += val  
  
print(val)  
294
```

## while

initial value while condition: statement

```
In [231...]  
num = 10  
while num>0:  
  
    print(num)  
    num-=1
```

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

```
In [232...]  
number=10  
while number<=20:  
    print(number*2)  
    number+=1
```

```
20  
22  
24  
26  
28  
30  
32  
34  
36  
38  
40
```

#danger infinitive loop n=10 while n>0: print(n)

```
In [233...]  
#even numbers between 1 to 50  
  
even_numbers=[]  
  
number=1  
while number<=50:  
    if number%2==0:  
        even_numbers.append(number)  
    number+=1  
  
print(even_numbers)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50]
```

```
In [234...]  
#exercise: factors of 5 between 1 to 1000
```

```
factors=[]  
  
num=1  
  
while num<=200:  
    factors.append(num*5)  
    num+=1  
  
print(factors)
```

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250, 255, 260, 265, 270, 275, 280, 285, 290, 295, 300, 305, 310, 315, 320, 325, 330, 335, 340, 345, 350, 355, 360, 365, 370, 375, 380, 385, 390, 395, 400, 405, 410, 415, 420, 425, 430, 435, 440, 445, 450, 455, 460, 465, 470, 475, 480, 485, 490, 495, 500, 505, 510, 515, 520, 525, 530, 535, 540, 545, 550, 555, 560, 565, 570, 575, 580, 585, 590, 595, 600, 605, 610, 615, 620, 625, 630, 635, 640, 645, 650, 655, 660, 665, 670, 675, 680, 685, 690, 695, 700, 705, 710, 715, 720, 725, 730, 735, 740, 745, 750, 755, 760, 765, 770, 775, 780, 785, 790, 795, 800, 805, 810, 815, 820, 825, 830, 835, 840, 845, 850, 855, 860, 865, 870, 875, 880, 885, 890, 895, 900, 905, 910, 915, 920, 925, 930, 935, 940, 945, 950, 955, 960, 965, 970, 975, 980, 985, 990, 995, 1000]
```

```
In [236...]  
password= int (input ("enter your password: "))  
  
while password != 1234:  
    print("the password {password} you entered is wrong")  
    password=int(input ("enter your password: "))  
if password==1234:  
    print("successful")
```

```
enter your password: 1234  
successful
```

```
In [238...]  
#create 20 random numbers between 0 to 100
```

```
import random  
  
a=[]  
random.seed(10)  
  
while len(a)<20:  
    rand=random.randrange(0,100)  
    a.append(rand)  
  
a
```

```
Out[238]: [73, 4, 54, 61, 73, 1, 26, 59, 62, 35, 83, 20, 4, 66, 62, 41, 9, 31, 95, 46]
```

```
In [239...]  
b=[]  
c=0  
while True:  
    c+=1  
    if c%6==0 and c%14==0:  
        break  
    b.append(c)  
  
print(b)  
print(c)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]  
42
```

```
In [240... import random
counter=0

while True:
    ran=random.random()+1
    counter+=ran
    if counter>=20:
        break

print(counter)
20.505253406323266

In [241... n = 5
while n > 0:n -= 1; print(n)

4
3
2
1
0
# while loop exercise #using loop to show 3 times "hello world" # Prints all letters except 'e' and 's' use :while , if , else , .... a = 'geeksforgeeks' # break the loop as soon it sees 'e' b = 'geeksforgeeks' # checks if list still contains any element a = [1, 2, 3, 4]

In [242... dictt={"a": 1 , "b": 2 , "c": 3 , "d":4 , "e":5 , "f":6 , "g":7 , "h": 8}
dictt["a"]

Out[242]: 1

In [243... i=0
a="geeksforgeeks"
while i <len(a):
    if a[i]=="e" or a[i]=="s":
        i+=1
        continue

    else:
        print(a[i])
        i+=1

g
k
f
o
r
g
k
#continue to special hight initial_height = 10 bounce_factor = 0.5 height = initial_height ----- #reverse counter ----- print 0 to 100 except 2 factors -----
----- using while loop chack if s exist in list a: a = ['foo', 'bar', 'baz', 'qux'] s = 'corge' ----- For each iteration, the program asks for user input and keeps repeating till the user inputs a non-numeric string. use isnumeric() method. -----
```

## List comprehension

```
In [244... number_list=[]
for i in range(10):
    number_list.append(i*2)

number_list
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Out[244]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [245... number_list2=[i*2 for i in range(10)]           #just for append

number_list2
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Out[245]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [246... num=[]
for i in range(10):
    if i%2==0:
        num.append(i)

num
[0, 2, 4, 6, 8]

Out[246]: [0, 2, 4, 6, 8]

In [247... num1=[i for i in range(10) if i%2==0]

num1
[0, 2, 4, 6, 8]

Out[247]: [0, 2, 4, 6, 8]

In [248... fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = []

for x in fruits:
    if "apple" == x:
        newlist.append(x)

newlist
['apple']

Out[248]: ['apple']

In [249... f=[x for x in fruits if "apple" == x]
f

Out[249]: ['apple']

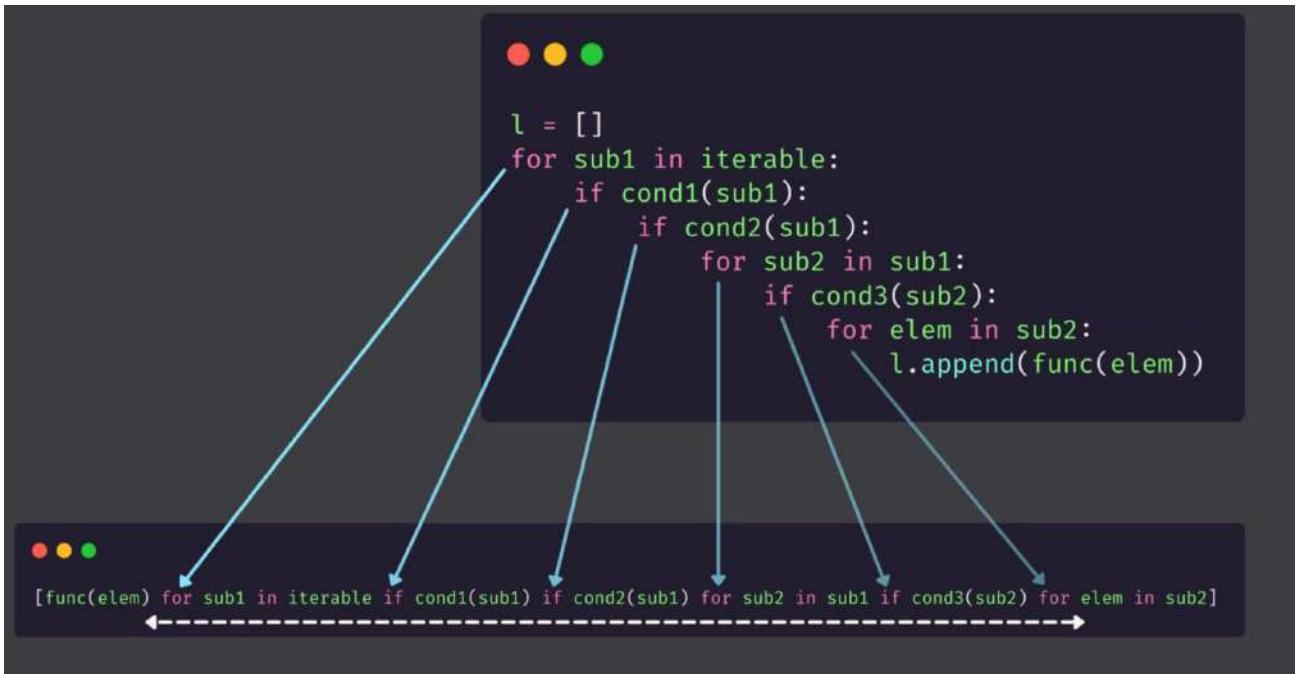
In [250... newList1 = [j for j in range(10) if j < 5]
newList1

Out[250]: [0, 1, 2, 3, 4]

In [251... newList=[]
for j in range(10):
    if j<5:
        newList.append(j)

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

Out[251]: [0, 1, 2, 3, 4]
```



```
In [252]: a=[]
for i in range(20):
    if i%2==0:
        a.append(i)
    else:
        a.append(-i)
print(a)
[0, -1, 2, -3, 4, -5, 6, -7, 8, -9, 10, -11, 12, -13, 14, -15, 16, -17, 18, -19]
```

```
In [253]: num3=[i if i%2==0 else -i for i in range(20) ]
print(num3)
[0, -1, 2, -3, 4, -5, 6, -7, 8, -9, 10, -11, 12, -13, 14, -15, 16, -17, 18, -19]
```

```
In [254]: import random
#exercise : change to List comprehension
random_numbers=[]
for i in range(10):
    random_numbers.append(random.randint(1,20))

number2=[random.randint(1,20) for i in range(10)]

random_numbers
number2
```

```
Out[254]: [12, 8, 11, 18, 15, 14, 16, 3, 19, 11]
```

## Exercise

```
In [255]: numbers = []
for i in range(1,1001):
    numbers.append(i)

#print(numbers , end="")

numbers2=[i for i in range(1,1001)]
#print(numbers2)
```

```
In [256]: list_8 = []
nums=[1,18,56,9,8,74,12,16,24,80,50,23,14,78,95]

for num in nums:
    if num%8==0:
        list_8.append(num)

list9=[num for num in nums if num%8==0]
```

```
In [257]: list_9 = []
nums=[1,18,56,9,8,74,12,16,24,80,50,23,14,78,95]

for num in nums:
    if num%2==0:
        list_9.append("even")
    else:
        list_9.append("odd")

nums3=["even" if num%2==0 else "odd" for num in nums]
```

```
In [258]: list_6=[]
for j in range(1,1000):
    if "6" in str(j):
        list_6.append(j)

print(list_6)

list7=[j for j in range(1,1000) if "6" in str(j)]
```

[6, 16, 26, 36, 46, 56, 66, 61, 62, 63, 64, 65, 66, 67, 68, 69, 76, 86, 96, 106, 116, 126, 136, 146, 156, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 176, 186, 196, 206, 216, 226, 236, 246, 256, 266, 261, 262, 263, 264, 265, 266, 267, 268, 269, 276, 286, 296, 306, 316, 326, 336, 346, 356, 366, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 376, 386, 396, 406, 416, 426, 436, 446, 456, 466, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 476, 486, 496, 506, 516, 526, 536, 546, 556, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 576, 586, 596, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 706, 716, 726, 736, 746, 756, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 776, 786, 796, 806, 816, 826, 836, 846, 856, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 876, 886, 896, 906, 916, 926, 936, 946, 956, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 976, 986, 996]

```

-----
NameError                                 Traceback (most recent call last)
Cell In[258], line 9
      4         list_6.append(j)
      5     print(list_6)
----> 6 list7=[j for j in rang(1,1000) if "6" in str(j)]
      7
      8 NameError: name 'rang' is not defined
In [259... string= "python in for student of architecture is important"
      counter=[]
      for char in string:
          if char== " " :
              counter.append(char)
      print(len(counter))
      a=len([char for char in string if char==" "])
      a
      7
Out[259]: 7

In [260... list1=["M","na","i","ke"]
list2=["y","me","s","lly"]

list3=[]
for i in range(len(list1)):
    list3.append(list1+list2)

list3=[list1[i]+list2[i] for i in range(len(list1))]
print(list3)
['My', 'name', 'is', 'kelly']

In [261... list1=["Mike", " ", "Ema", "kelley", " ", "brad"]
list2=[char for char in list1 if char!=" "]
print(list2)

list3=[]
for char in list1:
    if char!=" ":
        list3.append(char)
print(list3)
['Mike', 'Ema', 'kelley', 'brad']
['Mike', 'Ema', 'kelley', 'brad']

```

## Dictionary Comprehension

```

dictionary = {key: value for vars in iterable}
In [262... keys = ['a','b','c','d','e']
values = [1,2,3,4,5]

mydict={}
for k , v in zip(keys , values):
    mydict.update({k:v})

mydict
Out[262]: {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

In [263... keys = ['a','b','c','d','e']
values = [1,2,3,4,5]

myDict = {k:v for k,v in zip(keys, values)}

print (myDict)
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

In [264... mydict2={}
for x in [1,2,3,4,5]:
    mydict2.update({x:x**2})

mydict2
Out[264]: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

In [265... myDict2 = {x: x**2 for x in [1,2,3,4,5]}
print (myDict2)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

In [266... newdict = {x: x**3 for x in range(10) if x**3 % 4 == 0}
print(newdict)
{0: 0, 2: 8, 4: 64, 6: 216, 8: 512}

In [267... dict4={}
for i in range(20):
    if i%2==0:
        dict4.update({i : "even"})
    else:
        dict4.update({i : "odd"})

print(dict4 , end= "")
{0: 'even', 1: 'odd', 2: 'even', 3: 'odd', 4: 'even', 5: 'odd', 6: 'even', 7: 'odd', 8: 'even', 9: 'odd', 10: 'even', 11: 'odd', 12: 'even', 13: 'odd', 14: 'even', 15: 'odd', 16: 'even', 17: 'odd', 18: 'even', 19: 'odd'}

In [268... dictionary4={i:'even' if i%2==0 else "odd" for i in range(20)}
print(dictionary4 , end= "")
{0: 'even', 1: 'odd', 2: 'even', 3: 'odd', 4: 'even', 5: 'odd', 6: 'even', 7: 'odd', 8: 'even', 9: 'odd', 10: 'even', 11: 'odd', 12: 'even', 13: 'odd', 14: 'even', 15: 'odd', 16: 'even', 17: 'odd', 18: 'even', 19: 'odd'}
# square_dict = dict() for num in range(1, 11): square_dict[num] = num*num ~ update print(square_dict) ----- original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33} output('john': 33) -----
----- customer = {customer.discount for (customer, discount) in discount_dict.items()} if discount<30 ----- powers = {} for n in range(100): powers[n] = n ** print() Prints to the

```

## Built-in functions

`abs()` Returns the absolute value of a number  
`dir()` Returns a list of the specified object's properties and methods  
`float()` Returns a floating point number  
`help()` Executes the built-in help system  
`id()` Returns the id of an object  
`input()` Allowing user input  
`int()` Returns an integer number  
`len()` Returns the length of an object  
`list()` Returns a list  
`min()` Returns the smallest item in an iterable  
`pow()` Returns the value of x to the power of y  
`print()` Prints to the

standard output device range() Returns a sequence of numbers, starting from 0 and increments by 1 (by default) reversed() Returns a reversed iterator round() Rounds a numbers set() Returns a new set object sorted() Returns a sorted list str() Returns a string object sum() Sums the items of an iterator tuple() Returns a tuple type() Returns the type of an object zip() Returns an iterator, from two or more iterators

```
In [269... st=["kitchen" , "diningroom" , "livingroom"]
sorted(st , key=len , reverse=True)
Out[269]: ['diningroom' , 'livingroom' , 'kitchen']
```

## Function

```
def functionName (): #parameter statement return...
```

```
In [270... #even numbers
number=10
if number%2==0:
    print("even number")
even number

In [271... def even_number(number):
    if number%2==0:
        return "even_number"
    else:
        return "odd number"

even_number(20)           #argument
Out[271]: 'even_number'
```

```
In [272... def prime_number (x):          #prime number
    for i in range(2 , x):
        if x%i==0:
            return False
        else:
            return True

prime_number(30031)
```

```
Out[272]: True
```

```
In [273... def f (a, b):
    return (a**2 + b**2)

f(5,8)
Out[273]: 41
```

```
In [274... def gh12():
    print("hello")

gh12()
hello
```

```
In [275... def mean(list_5):
    return sum(list_5)/ len(list_5)

mean([1,2,3, 15,45,7,8,20])
#mean(5)
```

```
Out[275]: 12.625
```

```
In [279... def f(x):
    k=[]
    for i in range(len(x)):
        k.append(x[i]**2)
    return k

f([1,2,3,4,5,6])
#function(6)
```

```
Out[279]: [1, 4, 9, 16, 25, 36]
```

```
In [280... def function(x):
    return [x[i]**2 for i in range(len(x))]

function([1,2,3,4,5,6])
Out[280]: [1, 4, 9, 16, 25, 36]
```

```
In [281... def function(u):
    k=[]
    for i in range(len(u)):
        return u[i]**2

function([1,2,3,4,5,6])
```

```
Out[281]: 1
```

```
In [282... #exercise : write previous function as a list comprehension

def function(u):
    return [u[i] **2 for i in range(len(u))]

function([1,2,3,4,5,6])
Out[282]: [1, 4, 9, 16, 25, 36]
```

```
In [283... def h(x):
    return x+2
    return x*4

h(3)
Out[283]: 5
```

```
In [284... def g(y):
    return y+2 , y+3 , y*3

#g(3)[2]
a,...,_=g(3)
```

```
In [285... def math (num1 , num2):
    a=num1*3
    b=num2/4
    return a , b
```

```

math([2,6][0]
a,b=math(6,10)

In [286... def math (num1 , num2):
    a=num1*3
    b=num2/2
    return a , b

math([2,6] , [4,3])

-----
TypeError: Traceback (most recent call last)
Cell In[286], line 7
      3     b=num2/2
      4     return a , b
----> 7 math([2,6] , [4,3])

Cell In[286], line 3, in math(num1, num2)
      1 def math (num1, num2):
      2     a=num1*3
----> 3     b=num2/2
      4     return a , b

TypeError: unsupported operand type(s) for /: 'list' and 'int'

In [287... [1,2]*3
Out[287]: [1, 2, 1, 2, 1, 2]

In [288... def maximum (a,b,c,d):
    return max(a,b,c,d)

maximum(1,2,3,10)

Out[288]: 10

```

## a) function- argument

An argument is the value that is sent to the function when it is called.

```

In [289... #unpack
a=[1,2,3,4,5,6]
for i in a:
    print(i , end=" ")
1 2 3 4 5 6

In [290... print(*a) #asterisk
1 2 3 4 5 6

In [291... n=[10,20,32,44,58]
print(*n)
10 20 32 44 58

In [292... def f (a,b,c):
    print(a**2)
    print(b**2)
    print(c**2)

n=[1,2,3]
#f(n[0] , n[1] , n[2])
f(*n)           # *** unpack  and specify more than one
#f(n)
2
4
6

In [293... def f (a,b,c):
    print(a)
    print(b)
    print(c)

dic={"a": 7 , "b": 4 , "c": 5}
#f(dic["a"], dic["b"], dic["c"])

f(**dic)
7
4
5

In [294... a,b,c = 1,2,3
print(b)
2

In [295... a,b,c = 1,2,3,4,5

-----
ValueError: Traceback (most recent call last)
Cell In[295], line 1
----> 1 a,b,c = 1,2,3,4,5

ValueError: too many values to unpack (expected 3)

In [296... a,*b,c= 1,2,3,4,5      #Asterisk
print(*c)
3 4 5

In [297... *a,b,*c= [1,2,3,4,5]      #Asterisk
print(*a)
1 2 3

In [298... n,*c,d=[1,2,3,4,5]

```

```

print(c)
[2, 3, 4]

In [299... def f (a,b,c):
    print(a)
    print(b)
    print(c)

f(1,2,3)          #positional
1
2
3

In [300... def f (a,b,c):
    print(a)
    print(b)
    print(c)

f(a=1,b=2,c=3)      #keyword argument
1
2
3

In [301... def f (a,b,c):
    print(a)
    print(b)
    print(c)

f(b=2,c=3,a=1)
1
2
3

In [302... def f (a,b,c):
    print(a)
    print(b)
    print(c)

f(1,2,c=3)
1
2
3

In [303... def f (a,b,c):
    print(a)
    print(b)
    print(c)

f(1,b=2,3)
Cell In[303], line 6
  f(1,b=2,3)
^
SyntaxError: positional argument follows keyword argument

In [304... def f (a,b,c):
    print(a)
    print(b)
    print(c)

f(a=1,c=2,3)
Cell In[304], line 6
  f(a=1,c=2,3)
^
SyntaxError: positional argument follows keyword argument

```

## b) function- parameter

A parameter is the variable listed inside the parentheses in the function definition.

```

In [305... def f (a=1,b=2,c=3):
    print(a)
    print(b)
    print(c)

f()
1
2
3

In [306... def f (a=1,b=2,c=3):
    print(a)
    print(b)
    print(c)

f(a=5,b=2,c=4)
5
2
4

In [307... def f (a=1,b=2,c=3):
    print(a)
    print(b)
    print(c)

f(5,b=6,c=4)
5
6
4

In [308... def f (a,b,c=3):
    print(a)
    print(b)
    print(c)

f(5,8)
5
8
3

In [309... def power(a , b , c , d=3 , e = 2):
    return (a**2 + b**2 + c**2+d**2+e**2)
power(1,5)

```

```

-----  

TypeError: power() missing 1 required positional argument: 'c'  

-----  

Cell In[309], line 3  

  1 def power(a , b , c , d=3 , e = 2):  

  2     return (a**2 + b**2 + c**2+d**2+e**2)  

--> 3 power(1,5)  

-----  

TypeError: power() missing 1 required positional argument: 'c'  

In [310]:  

def summ(a=1 , b=2 ,c=5):  

    return (a+b+c)  

print(summ())  

print(".....")  

print(summ(4))  

print(".....")  

print(summ(5,4))  

print(".....")  

print(summ(c=10))  

8  

.....  

11  

.....  

14  

.....  

13  

-----  

In [311]:  

def f (a,b,*c):  

    print(a)  

    print(b)  

    print(c)  

#f(1,2,3)  

f(2,3,4,5)  

2  

3  

(4, 5)  

-----  

In [314]:  

def f (*c):  

    #print(a)  

    #print(b)  

    print(c)  

f(1,2,3,4,5,6,8)  

(1, 2, 3, 4, 5, 6, 8)  

-----  

In [313]:  

def maxi(*args):  

    return max(args)  

maxi(1,2,5,12,48,75,9,86,3,102)  

102  

Out[313]: 102  

-----  

In [315]:  

def mean(*args):  

    return sum(args)/len(args)  

mean (1,2,3,4,5,6,10)  

4.428571428571429  

Out[315]: 4.428571428571429  

-----  

In [316]:  

def number(*args):  

    return max(args)  

n=[12,10,5,9,8,7,4,14,7,0]  

number(n)  

14  

Out[316]: 14  

-----  

In [317]:  

def f (a,b,c):  

    print(a)  

    print(b)  

    print(c)  

f(a=7 , b=4, c=5)  

7  

4  

5  

-----  

In [318]:  

def f (**kwargs):  

    print(kwargs)  

f(a=2 , b=4 , c=5 ,d=6 , e=9)  

{'a': 2, 'b': 4, 'c': 5, 'd': 6, 'e': 9}  

-----  

In [319]:  

def h(*args , **kwargs):  

    pass  

h(1,2,3,4,5,a=8,b=9,d=12)  

-----  

In [320]:  

def p (a,b,*args , **kwargs):  

    pass  

p(14,-5,9,45,72, m=12.5,n=-32)  

-----  

In [321]:  

def function(a , b , * , c , d):           #after  

    return a+b+c+d  

function(5,7,c=10,d=6)  

28  

Out[321]: 28  

-----  

In [322]:  

def function(a , b , / , c , d):           #before  

    return a+b+c+d  

function(5,7,c=10,d=6)  

28  

Out[322]: 28  

-----  

In [323]:  

def function(a , b , / ,c,* , d,e):  

    return a+b+c+d+e  

function(5,7,c=10,d=6,e=5)

```

```

Out[323]: 33
In [324... #exercise
# function with two parameters and shows the summation
# A simple Python function to check whether x is even or odd
# function that gives an interger number and cast it to string
#sum(1, 2, 3, 4, 5) using *args

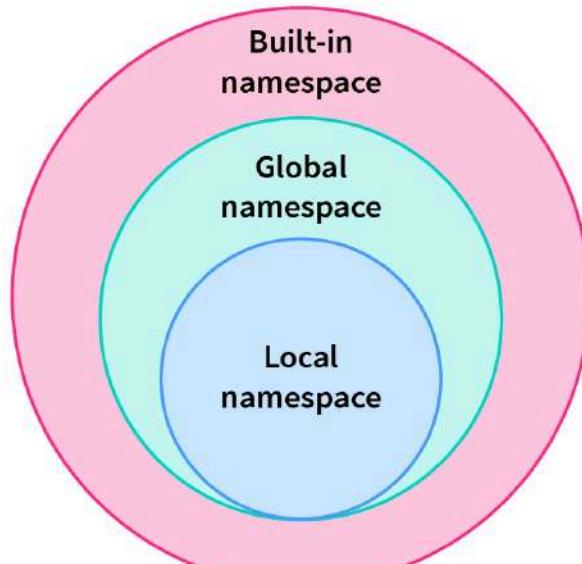
In [325... def summ(**kwargs):
    return sum(kwargs.values())
summ(a=1,b=2,c=3,d=7)
Out[325]: 13
In [326... def f(x):
    if type(x)== str:
        print("ok")
    else:
        return str(x)
f("12.5")
ok

c) function annotation

In [327... def t (x:"adad sahih vared shabvad", y:int , z:int)-> int:           #definition can be string      python3 and more
    return (x+y+z)
t(2,5,9)
Out[327]: 16
In [328... def t (x:"int or float", y:int , z=8):
    return (x+y+z)
t(2,5)
Out[328]: 15
In [329... def maxi(*args:int)-> float:
    return max(args)
maxi(1,2,5,8,9,12)
Out[329]: 12

```

## Name space



Type of Namespaces

**SCALER**  
*Topics*

```
In [ ]: dir(__builtin__)
In [331... def f(x):
    y=x**2
    z=x+5
    return y+z
print(f(10))
#.....
a=10
115
In [332... def f(x):
    global y,z
    y=x**2
    z=x+5
    return y+z
print(f(10))
print(y)
print(z)
#enclose
115
100
15
In [333... for i in range(10):
    a=i
print(a)
9
```

## Lambda

```
In [334... def f (x):
    return x+10
print(f(2))
a=lambda x : x+10
print(a(2))
12
12
In [335... f=lambda x,y: x*y
f(3,5)

def s (x, y):
    return x*y
In [336... power = lambda x: x ** 2
power(3)
Out[336]: 9
In [337... add = lambda x, y: x + y
add(3,5)
```

```
In [338]: def addd(x,y):
    return x+y

# Use if-else in Lambda Functions

# check if number is even or odd
result = lambda x : x if x %2==0 else f"{x} is odd"

# print for numbers
print(result(12))
print(result(11))

12
11 is odd
```

```
In [339]: def result(x):
    if x%2==0:
        return x
    else:
        print(f"{x} is odd")
```

```
In [340]: result = lambda x : x%2==0

# print for numbers
print(result(12))
print(result(11))

True
False
```

## Map

Python's map() is a built-in function that allows you to process and transform all the items in an iterable without using an explicit for loop

map(function , list)

```
In [341]: def f (*args):
    a=[]
    for i in args:
        a.append(i**2)
    return a

list_1 = [1,2,3,4,5]
f(*list_1)

Out[341]: [1, 4, 9, 16, 25]
```

```
In [342]: def f (x):
    return x**2

list_1 = [1,2,3,4,5]

map_number= map(f , list_1)

print(list(map_number))

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

```
In [343]: #function
def addition(n):
    return n + n

#list
numbers = (1, 2, 3, 4)

# We double all numbers using map()
result = map(addition, numbers)
print(list(result))

[2, 4, 6, 8]
```

```
In [344]: def func(n):
    return abs(n)

abso_numbers=list(map(func , range(-10, 0)))

abso_numbers

Out[344]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
In [345]: def up(word):
    return word.upper()

word="introduction to python"

# map() can listify the list of strings individually

a=tuple(map(up , word))
print(a)

('I', 'N', 'T', 'R', 'O', 'D', 'U', 'C', 'T', 'I', 'O', 'N', ' ', 'T', 'O', ' ', 'P', 'Y', 'T', 'H', 'O', 'N')
```

```
In [346]: sequences = [10,2,8,7,5,4,11]
```

```
def g(x):
    return x**2

squared_result = map (lambda x: x**2, sequences)

print(list(squared_result))

[100, 4, 64, 49, 25, 16, 121]
```

```
In [347]: list_1 = [1,2,3,4,5]

map_number= list(map(lambda x : x+5 , list_1))

map_number
```

```
Out[347]: [6, 7, 8, 9, 10]

In [348]: abso_numbers=list(map(lambda n : abs(n) , range(-10, 0)))

abso_numbers

Out[348]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
In [349]: numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]
```

```

result = map(lambda x, y: x + y, numbers1, numbers2)
print(list(result))
[5, 7, 9]

In [350... # List of strings
list_3 = ['satt', 'bat', 'cattt', 'ma']

test = list(map(len, list_3))
print(test)
[4, 3, 6, 2]

In [351... list_2=[]

for x in range(0,10):
    a=(x**2)
    list_2.append(a)

list_2
Out[351]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```

## Filter

Filter() is a built-in function in Python. The filter function can be applied to an iterable such as a list or a dictionary and create a new iterator. This new iterator can filter out certain specific elements based on the condition that you provide very efficiently.

filter(function , list)

```

In [352... def function (number):
    if number>=4:
        return True
    else:
        return False

list_2 = [0,2,6,9,4,1,-2,5,4,6,10]

filtered_number= filter(function , list_2)
print(list(set(filtered_number)))
[4, 5, 6, 9, 10]

In [353... def even(x):
    if x %2==0:
        return True
    else:
        return False

list_number=[1,2,3,4,5,6,7,8,9,10,10]

print(tuple(filter(even , list_number)))
(2, 4, 6, 8, 10, 10)

In [354... list_3 = [0,2,6,9,4,1,-2,5,4,6,10]

filtered_m= list(filter(lambda number:True if number>=4 else False, list_3))

print(set(filtered_m))
{4, 5, 6, 9, 10}

In [355... list_2 = [0,2,6,9,4,1,-2,5,4,6,10]

filtered_n= list(filter(lambda number:number>=4 , list_2))

filtered_n
[6, 9, 4, 5, 4, 6, 10]

Out[355]: [6, 9, 4, 5, 4, 6, 10]

```

```

In [356... seq = [0, 1, 2, 3, 5, 8, 13]

# result contains odd numbers of the list
result = filter(lambda x: x % 2 != 0, seq)
print(list(result))

# result contains even numbers of the list
result2 = filter(lambda x: x % 2 == 0, seq)
print(list(result2))
[1, 3, 5, 13]
[0, 2, 8]

```

```

In [357... while "":
    pass

-----
KeyboardInterrupt                                     Traceback (most recent call last)
Cell In[357], line 1
----> 1 while "":
      2     pass

KeyboardInterrupt:

```

```

In [358... result = filter(None,(1,0,6,-1,None, [], " ", True , False))      # returns all non-zero values
print(list(result))
[1, 6, -1, ' ', True]

```

#exercise multiplication of 3 numbers = (1,3,5,6,8,9,12,14) ----- filter even numbers seq = [10, 125, 248, 320, 54, 88, 132]

```

In [359... #exercise
#use Lambda
def mySum(x, y,z):
    return x+y+z

take 2 parameters from user and multiply them using lambda
# List1=[1,2,3,4,5,6,7,8,9]
#multiply the numbers of list by 3 using map and Lambda

#calculate the absolute of range(-10, 0) using Lambda function and map

#prices=[1000,250,360,1020,45,789,256,365,412,478,120,203,478, 25,156]
#we need prices more than 400 using filter and lambda

Cell In[359], line 5
-----^
SyntaxError: invalid syntax

```

```

In [360... result= lambda x,y : x*y

```

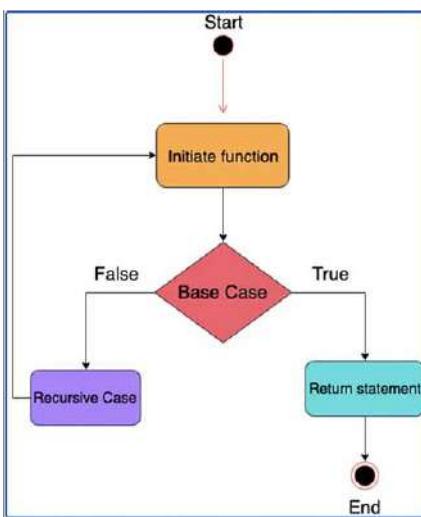
```

a=int(input("enter the first number"))
b=int(input("enter the first number"))

print(result(a , b))
enter the first numbers
enter the first numbers
30

```

## Recursive function



```

In [361]: #factorial
def f(number):
    if number==1:
        return 1
    return number * f(number-1)

f(3)

```

Out[361]: 6

```

In [362]: from time import sleep
def reverse_number(n):
    #condition
    if n==10:
        return n
    print(n)
    sleep(1)
    n+=1
    reverse_number(n)

reverse_number(0)

```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

```

In [363]: #fibonacci
# 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144...

def fib (n):
    if n==0 or n==1:
        return n
    return fib(n-1)+ fib(n-2)

fib(5)

```

Out[363]: 5

## Enumerate

```

In [364]: listt=[1,2,3,4,5,6]
print(list(enumerate(listt)))
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]

```

```

In [365]: numbers=[12,18,75,23,10]
for index,num in enumerate (numbers):
    print(index,"--",num)

0 -- 12
1 -- 18
2 -- 75
3 -- 23
4 -- 10

```

```

In [366]: # Python program to illustrate
# enumerate function in Loops
list_5 = ["Mahsa", "Sara", "Ashkan"]

# changing index and printing separately
for i, j in enumerate(list_5 , start=2):
    print (i,j)

2 Mahsa
3 Sara
4 Ashkan

```

#exercise using enumerate the list of some numbers and when reach to index 10 print(show the index and value) and berak use enumerate and Return values from iterable when their index is even. start from 2 [10,12,13,14,89,65,47,12,10,35,47,96,0,2,3,4,5,7,89]

```

In [367]: list1=[10,12,13,14,89,65,47,12,10,35,47,96,0,2,3,4,5,7,89]
for i,j in enumerate(list1):

```

```

    if i==10:
        print(i,j)
        break
10 47
In [368... list1=[10,12,13,14,89,65,47,12,10,35,47,96,0,2,3,4,5,7,89]
for i,j in enumerate(list1,start=2):
    if i%2==0:
        print(j)

10
13
89
47
10
47
0
3
5
89
In [369... a = [10,12,13,14,89,65,47,12,10,35,47,96,0,2,3,4,5,7,89]
for i, j in enumerate(a , 2):
    if i % 2 == 0 :
        print(i,j)

2 10
4 13
6 89
8 47
10 10
12 47
14 0
16 3
18 5
20 89

```

## The Walrus Operator :=

assign - return



```

In [370... import sys          #python 3.8 and higher
print(sys.version)
3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
In [371... w=5
print(w)
5
In [372... print(w=5)
-----
TypeError                                 Traceback (most recent call last)
Cell In[372], line 1
----> 1 print(w=5)

TypeError: 'w' is an invalid keyword argument for print()
In [373... print(w:=5)
5
In [374... a=[1,2,3,4,5]
n=len(a)
if n >3:
    print(True)
True
In [375... if (n:=len(a)) >3:           #better to use paracensis
    print(True)
True
In [376... n=5
print(f" the number you wrote is {n} ")
the number you wrote is 5
In [377... print(f" the number you wrote is {(n:=5)}")
the number you wrote is 5
In [378... number=[1,2,3,4,5]

dictionary={
    "le" : len(number),
    "su" : sum(number),
    "mean" : len(number) / sum(number)
}

In [379... number=[1,2,3,4,5]

dictionary={
    "le": (1:=len(number)),
    "su": (su:= sum(number)),
    "mean": 1/su
}
```

## Format

```

In [380... #format string
x= 12
y=10.3

```

```
#print("the first number is {}\nand the seconf number is {}" .format(x, y))
#print(f"the first number is {x}\nand the second number is {y}")
#print("the first number is:", x, "and the seconf number is",y )

In [381... print("the first number is {0}\n and the seconf number is {1}" .format(x, y))

the first number is 12
and the seconf number is 10.3

In [382... print("the first number is {0}\n and the seconf number is {1}\n and the third number is {2}" .format(*[1,2,3]))

the first number is 1
and the seconf number is 2
and the third number is 3

In [383... print("the number is {:.3}" .format(12.3694589))

print("the number is {:.2}" .format(0.5569))

print("the number is {:.2%}" .format(0.5))

the number is 12.4
the number is 0.56
the number is 50.00%
```

## input

```
In [ ]: """
x= int(input(" the first number"))
y=int(input("the seconfd number"))
z= int(input("the third numbrt"))
"""

"""

list1=input().split(",")
print(list1)

#print([int(x) for x in list1])

#list(map(lambda x : int(x) , list1))
"""


```

## DocString

```
#documentation string
x"""
this finction add two nubers
"""

#print function
print(x)

this finction add two nubers

In [384... def maximum (*args):
    """ Return the maximum of some numbers.           #START UPPERCASE - point at end
    parameters:
        args (int)

    Returns:
        max_int (int) : Returns the maximum of some numbers

    """
    return max(args)

#print(maximum.__doc__)
print(help(maximum))

Help on function maximum in module __main__:

maximum(*args)
    Return the maximum of some numbers.           #START UPPERCASE - point at end
    parameters:
        args (int)

    Returns:
        max_int (int) : Returns the maximum of some numbers

None
In [ ]: help(str)
```

## files

```
In [ ]: #f1= open(r"D:\txt1.txt" , "r")
#print(f1.read())

In [386... f2= open(r"D:\txt1.txt" , "w")

In [387... f3= open(r"D:\text2.txt" , "w")
print(f3.write("machine learning for architects"))
f3.close()

31
In [388... ff= open(r"D:\txt2.txt" , "w")

In [389... f4= open(r"D:\text2.txt" , "a")
print(f4.write("\nthis useful"))
f4.close()

12
In [ ]: with open(r"D:\text3.txt" , "w") as f5:
    f5.write("this is fun")
```

```
In [390...]: with open(r"D:\text3.txt" , "a") as f5:  
    f5.write("\nnot the end")
```

```
In [391...]: with open(r"D:\txt1.txt" , "r")as f6:  
    for line in f6:  
        print(line)
```

```
In [392...]: li=[1,2,3,4,5,6,7]  
with open(r"D:\text4.txt" , "w") as f7:  
    f7.writelines(str(li))
```

## PEP - Python Enhancement Proposal

```
In [ ]: #https://peps.python.org/pep-0000/
```

```
In [ ]: import this
```

```
In [ ]: #https://pep8.ir/
```

## Iterat

iteration -> for and while ..... iterable -> list - tuple - sequence ..... iterator -> Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `_iter_()` and `_next_()`.

```
In [393...]: list_1 = [1,2,3,4,5]  
  
print(dir(list_1))          # print("_iter_" in dir(List_1))  
  
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',  
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',  
'__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert',  
'pop', 'remove', 'reverse', 'sort']
```

```
In [394...]: list_1 = [1,2,3,4,5]  
  
list_1 = iter(list_1)          #you can use next method on it  
  
print(next(list_1))
```

```
1
```

```
In [395...]: list_number=[1,2,3,4,5,6]  
list_number= iter(list_number)  
  
while True:  
    try:  
        print(next(list_number))  
    except StopIteration:  
        break
```

```
1  
2  
3  
4  
5  
6
```

```
In [ ]: from itertools import count  
  
counter= count()  
for i in counter:  
    print(i)  
    if i==50:  
        break
```

```
In [ ]: #define an empty List  
a=list()  
for i in range(5):  
    a.append(i)
```

## Object Oriented Programming (OOP)

```
class Grey : morteza = Grey() #object morteza.nationality = "iranian" #attribute or properties morteza.language = "persian" morteza.job = "teacher" morteza.hight= 187  
class Architecture : project1.name = "Azadi Tower" project1.architect = "Amanat" project1.year= "1353" project1 = Architecture()  
#step 1 class Grey: #attribute or properties self.language = "persian" self.job = "teacher" self.hight= 187 morteza = Grey() #object #step2 class Grey: self.nationality = x self.language = y self.job = z  
self.hight= k morteza = Grey() #step3 class Grey: self.nationality = nationality self.language = language self.job = job self.hight= height morteza = Grey() #step4 class Grey: def data (self , nationality , language , job , hight): #method  
self.nationality = nationality self.language = language self.job = job self.hight= height morteza.data("iranian" , "persian" , "teacher" , "178")
```

```
In [397...]: class Architecture(object):  
    def f (self, name , architect , year):  
        self.name = name  
        self.architect = architect  
        self.year = year  
  
project1 = Architecture()  
  
project1.f("azadi" , " amanat" , "1353")
```

```
In [398...]: class Test :  
    def var(self , number1 , number2):  
        self.number1 = number1  
        self.number2 = number2  
    def add(self):  
        return self.number1 + self.number2  
    def mul(self):  
        return self.number1 * self.number2  
  
obj1= Test()  
  
obj1.var(2,3)  
  
print(obj1.add())  
  
print("-----")  
  
print(obj1.mul())
```

```
5  
-----  
6
```

```
In [399...]: class Language (object):  
    def Langname(self , name):  
        self.name = name  
    def display_name (self):  
        return self.name  
    def message_name (self):  
        print("welcome to " , self.name)
```

```

11=Language()
11.Langname("python")
print(11.display_name())
print("-----")

11.message_name()

python
-----
welcome to python

In [400]: s="pytohn"
          b="arct"
          print(type(s))
          s.upper()
<class 'str'>
'PYTOHN'
Out[400]: 

In [401]: class c :
             lst=[]
             def f (self , x):
                 self.lst.append(x)

obj1= c()
obj1.f(1)
obj1.f(2)
obj1.f(3)
print(obj1.lst)

[1, 2, 3]

In [402]: class list(object):
             """
             list() -> new empty list
             list(iterable) -> new list initialized from iterable's items
             """
             def append(self,number): # real signature unknown; restored from __doc__
                 """ L.append(object) -> None -- append object to end """
                 pass

             def clear(self): # real signature unknown; restored from __doc__
                 """ L.clear() -> None -- remove all items from L """
                 pass

             def copy(self): # real signature unknown; restored from __doc__
                 """ L.copy() -> list -- a shallow copy of L """
                 return []

             def count(self, value): # real signature unknown; restored from __doc__
                 """ L.count(value) -> integer -- return number of occurrences of value """
                 return 0

             def extend(self, iterable): # real signature unknown; restored from __doc__
                 """ L.extend(iterable) -> None -- extend list by appending elements from the iterable """
                 pass

             def index(self, value, start=None, stop=None): # real signature unknown; restored from __doc__
                 """
                 L.index(value, [start, [stop]]) -> integer -- return first index of value.
                 Raises ValueError if the value is not present.
                 """
                 return 0

             def insert(self, index, p_object): # real signature unknown; restored from __doc__
                 """ L.insert(index, object) -- insert object before index """
                 pass

             def pop(self, index=None): # real signature unknown; restored from __doc__
                 """
                 L.pop([index]) -> item -- remove and return item at index (default last).
                 Raises IndexError if list is empty or index is out of range.
                 """
                 pass

             def remove(self, value): # real signature unknown; restored from __doc__
                 """
                 L.remove(value) -> None -- remove first occurrence of value.
                 Raises ValueError if the value is not present.
                 """
                 pass

             def reverse(self): # real signature unknown; restored from __doc__
                 """ L.reverse() -- reverse *IN PLACE* """
                 pass

             def sort(self, key=None, reverse=False): # real signature unknown; restored from __doc__
                 """ L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE* """
                 pass

             def __add__(self, *args, **kwargs): # real signature unknown
                 """ Return self+value. """
                 pass

             def __contains__(self, *args, **kwargs): # real signature unknown
                 """ Return key in self. """
                 pass

             def __delitem__(self, *args, **kwargs): # real signature unknown
                 """ Delete self[key]. """
                 pass

             def __eq__(self, *args, **kwargs): # real signature unknown
                 """ Return self==value. """
                 pass

             def __getattribute__(self, *args, **kwargs): # real signature unknown
                 """ Return getattr(self, name). """
                 pass

             def __getitem__(self, y): # real signature unknown; restored from __doc__
                 """ x.__getitem__(y) <=> x[y] """
                 pass

             def __ge__(self, *args, **kwargs): # real signature unknown
                 """ Return self>=value. """
                 pass

             def __gt__(self, *args, **kwargs): # real signature unknown
                 """ Return self>value. """
                 pass

```

```

def __iadd__(self, *args, **kwargs): # real signature unknown
    """ Implement self+=value. """
    pass

def __imul__(self, *args, **kwargs): # real signature unknown
    """ Implement self*=value. """
    pass

def __init__(self, seq=()): # known special case of list.__init__
    """
    list() -> new empty list
    list(iterable) -> new list initialized from iterable's items
    # (copied from class doc)
    """
    pass

def __iter__(self, *args, **kwargs): # real signature unknown
    """ Implement iter(self). """
    pass

def __len__(self, *args, **kwargs): # real signature unknown
    """ Return len(self). """
    pass

def __le__(self, *args, **kwargs): # real signature unknown
    """ Return self<=value. """
    pass

def __lt__(self, *args, **kwargs): # real signature unknown
    """ Return self<value. """
    pass

def __mul__(self, *args, **kwargs): # real signature unknown
    """ Return self*n """
    pass

@staticmethod # known case of __new__
def __new__(*args, **kwargs): # real signature unknown
    """ Create and return a new object. See help(type) for accurate signature. """
    pass

def __ne__(self, *args, **kwargs): # real signature unknown
    """ Return self!=value. """
    pass

def __repr__(self, *args, **kwargs): # real signature unknown
    """ Return repr(self). """
    pass

def __reversed__(self): # real signature unknown; restored from __doc__
    """ L.__reversed__() -- return a reverse iterator over the list """
    pass

def __rmul__(self, *args, **kwargs): # real signature unknown
    """ Return self*n. """
    pass

def __setitem__(self, *args, **kwargs): # real signature unknown
    """ Set self[key] to value. """
    pass

def __sizeof__(self): # real signature unknown; restored from __doc__
    """ L.__sizeof__() -- size of L in memory, in bytes """
    pass

__hash__ = None

```

## kinds of method

1. Instance method (use self)

1.1 usual method

1.2 magic method(dunder method) : recall method with a sign

1. Class method

@classmethod - cls

1. Static method

@staticmethod

In [403...]

```

class Car:
    def __init__(self, name):           #constructor
        self.name = name

a = Car('benz')
print(a.name)
benz

```

In [407...]

```

a=12
a=int(12)
b=[1,2,3,4,5]
b=list([1,2,3,4,5])

c="python for architects"
c=str("python for architects")

#b.append(8)

```

In [408...]

```

class car (object):
    def __init__ ( self , name , number_of_doors , color):
        self.name= name
        self.number_of_doors = number_of_doors
        self.color= color

benz = car ("s1s" , 2 , "red" )
print(benz.color)
red

```

```
In [409...]
class circle:
    def __init__(self , r):
        self.r = r
    def area (self):
        return self.r *self.r *3.14
x=circle(8)
x.area()

Out[409]: 67.14

In [410...]
class Rectangle:
    def __init__(self , width , hight):
        self.width= width
        self.hight= hight

    def area(self):
        return self.width * self.hight

rec=Rectangle(5 , 4)
rec.area()

Out[410]: 20

In [411...]
class Book :
    def __init__(self , author , title):
        self.author = author
        self.title = title
    def info(self):
        return self.title + " : " + self.author

X= Book("khorsand" , "interactive architecture")
X.info()

Out[411]: 'interactive architecture : khorsand'

In [412...]
class Dog:

    def __init__(self, dogBreed,dogEyeColor):
        self.breed = dogBreed
        self.eyeColor = dogEyeColor

    tomita = Dog("Fox Terrier","brown")

print("This dog is a",tomita.breed,"and his eyes are",tomita.eyeColor)

This dog is a Fox Terrier and his eyes are brown

In [413...]
# __str__ return a string representation of its object. run with print

class Employee:

    def __init__(self, name , salary):
        self.name= name
        self.salary= salary

    def __str__(self):
        return 'name=' +self.name+' salary='+str(self.salary)

person_1 = Employee("Neda" , 12000)

print(person_1)

person_1
name=Neda salary=$12000
<__main__.Employee at 0x1b6fff618340>

Out[413]: 

In [414...]
# __ge__() method  :This method gets invoked when the >= operator is used and returns True or False

class Person:
    def __init__(self, age):
        self.age = age
    def __ge__(self, other):
        return self.age >= other.age

alice = Person(18)
bob = Person(17)
carl = Person(18)

print(alice >= bob)
print(alice >= carl)
print(bob >= alice)

True
True
False

In [415...]
class Address:
    def __init__(self , city, street , zipcode):
        self.city= city
        self.street = street
        self.zipcode= zipcode

    def __str__(self):
        lst=[]
        lst.append(f" {self.city} -- {self.street} -- {self.zipcode}")
        return " ".join(lst)

a= Address("Tehran" , "174east" , "1655916993")

print(a)

Tehran -- 174east -- 1655916993

In [416...]
text = ['Python', 'is', 'a', 'fun', 'programming', 'language']

# join elements of text with space
print('++'.join(text))
Python++is++a+fun++programming++language

In [417...]
class Robot(object):
    def __init__(self , n , y):
        self.name= n
        self.buildyear= y
```

```

    def __str__(self):
        return "name: " + self.name + ", buildyear: "+str(self.buildyear)

object2= Robot("rr" ,1980)
print(object2)

name: rr, buildyear: 1980

```

```
In [419...:
class Person:
    def __init__(self, name, height):
        self.name = name
        self.height = height

    def show(self):
        print(f'{self.name} is {self.height}')

p = Person('amir', 180)
p.show()

amir is 180

```

```
In [420...:
class Test :
    def __init__(self , a):
        self.a = a
    def __add__(self , b):
        return self.a + b.a

obj1=Test(1)
obj2=Test(2)

print(obj1+obj2)

3

```

```
In [421...:
class AB :
    def __init__(self , age):
        self.age = age

    def __gt__(self, other):
        if self.age > other.age :           # __lt__ "Lowerthan"
            #__eq__ "equalto"           <   ==
            return True
        else:   False

Ali=AB(22)
Sara=AB(38)

if (Ali>Sara):
    print("yes")
else: print("no")

```

```
In [422...:
class K:
    def __init__(self):
        self.lst=[45,89,12]

    def __str__(self):
        return str(self.lst)

    def __len__(self):
        return len(self.lst)

    def __getitem__(self , i):
        return self.lst[i]

object1=K()
len(object1)
#object1[2]

#print(object1)

3

```

```
Out[422]: 3

In [423...:
class Machine:
    model = "peugeot"           #class variable

    def __init__(self , t):      #instance variable
        self.t = t

m1= Machine("206")
m2=Machine("207")

print(m1.model)
print(m2.model)

print("-----")

print(m1.t)
print(m2.t)

peugeot
peugeot
-----
206
207

```

```
In [425...:
class Test:
    def __init__(self , num1:int, num2:int):
        self.num1=num1           #public
        self.__num2= num2         #private     #data hiding

ob=Test(2,5)

print(ob.num1)

#print(ob.num2)           # Calling the private member
# through name mangling

2

```

```
In [426...:
class S :
    def __init__(self , x):
        self.__x = x

    def f (self):
        b=self.__x**2
        return b

    def g (self , m):
        return (m+ self.__x)

ob= S(2)
ob.f()


```

```
#ob.g(3)
Out[426]: 4

In [427]:
class H :
    __x=3           #Private class variable
    def show(self):
        print(self.__x + 2)

ob=H()
ob.show()
ob.__H__X

5
3
Out[427]: 3

In [428]:
class ABC:
    def __f(self):      #private method
        print(1)

    def g(self):
        return (self.__f())

ob=ABC()
ob.g()
ob.__ABC__f()

1
1
```

## Docstring

```
In [429]:
class Rectangle:
    """
    mathematical calculation of a rectangle

    >>>rec=Rectangle(2,8)
    >>>rec.area()
    16

    method:
        area : calculate the area of an rectangle

    attribute:
        width: the width of a rectangle
        height: the height of the rectangle
    """
    def __init__(self , width , height):
        self.width= width
        self.height= height
    def area(self):
        return self.width * self.height

rec=Rectangle(5 , 4)
rec.area()

Out[429]: 20

In [430]:
print(help(Rectangle))
Help on class Rectangle in module __main__:

class Rectangle(builtins.object)
| Rectangle(width, height)
|
|     mathematical calculation of a rectangle
|
|     >>>rec=Rectangle(2,8)
|     >>>rec.area()
|     16
|
|     method:
|         area : calculate the area of an rectangle
|
|     attribute:
|         width: the width of a rectangle
|         height: the height of the rectangle
|
|     Methods defined here:
|
|     __init__(self, width, height)
|         Initialize self. See help(type(self)) for accurate signature.
|
|     area(self)
|
|     -----
|     Data descriptors defined here:
|
|     __dict__
|         dictionary for instance variables (if defined)
|
|     __weakref__
|         list of weak references to the object (if defined)

None
```

In [ ]: print(help(list))

## Setter and Getter

```
In [431]:
class Person:
    def __init__(self , name , age):
        self.name =name
        self.age=age

pr=Person("Ali" , 25)
pr.name

pr.name="Hassan"
print(pr.name)

Hassan

In [432]:
class Colour:
    def __init__(self , name , rgb):          #private , protect
        self.__name=name
        self.__rgb=rgb

obj1=Colour("blue" , 0x80b0ff)
```

```
#print(obj1.__name__)

print(obj1._Colour__name)

#change the name
obj1._Colour__name="red"
obj1._Colour__name

blue
'red'
Out[432]:
```

```
In [ ]: class Colour:
    def __init__(self, name, rgb):
        self.__name=name
        self.__rgb=rgb

    def setName(self, name):
        self.__name=name

    def getName(self):
        return self.__name

    def setRgb(self, rgb):
        self.__rgb=rgb

    def getRgb(self):
        return self.__rgb

obj2=Colour("black", 0x000000)
obj2.setName("red")
obj2.getName()
```

## Inheritance

```
In [433]: class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def eat(self):
        print(f'{self.name} is eating.')

class Cat(Animal):
    def sound(self):
        print(f'{self.name} says meow.')

obj3=Cat("Garfield", "persian")
print(obj3.name)
obj3.eat()
obj3.sound()
```

Garfield  
Garfield is eating.  
Garfield says meow.

```
In [434]: class Person:
    def __init__(self, firstname, lastname):
        self.firstname = firstname
        self.lastname = lastname

    def printname(self):
        print(self.firstname, self.lastname)

class Student(Person):
    def studentId(self, StudentNumber):
        print("the student StudentNumber {} is valid".format(StudentNumber))

class Teacher(Person):
    def teacherId(self, teacherId):
        print("the student teacherId {} is valid".format(teacherId))

obst=Student("ali", "akbari")
obst.printname()
obst.studentId(1256)
obst.teacherId(5897)
```

ali akbari  
the student StudentNumber 1256 is valid

```
-----
AttributeError                                 Traceback (most recent call last)
Cell In[434], line 23
  20 obst.printname()
  21 obst.studentId(1256)
--> 23 obst.teacherId(5897)
```

**AttributeError: 'Student' object has no attribute 'teacherId'**

## Override

```
In [435]: class Student2(Person):
    def __init__(self, firstname, lastname, age):           #the same name
        self.firstname=firstname
        self.lastname=lastname
        self.age=age

st2=Student2("john", "Nori", 20)
st2.firstname
st2.printname()
```

john Nori

```
In [436]: class Student3(Person):
    def printname(self):
        print(self.lastname)

st3=Student3("john", "Nori")
st3.firstname
st3.printname()
```

```
Nori
In [437...]
class student4(Person):
    def __init__(self , firstname, lastname, age):
        super().__init__(firstname , lastname)
        self.age=age
#proxy object- create temporary object from superclass

st4=student4("Mina" , "Cruse", 23)
st4.firstname
st4.age
st4.printname()

Mina Cruse
```

## Inheritance from builtin class

```
In [438...]
class Name(str):
    def __init__(self , name):
        self.name=name

obj3=Name("python")
print(obj3.upper())
obj3.index("t")

PYTHON
2
Out[438]:
```

## Multiple Inheritance

```
In [439...]
class Mammal:
    def mammal_info(self):
        print("Mammals can give direct birth.")

class WingedAnimal:
    def winged_animal_info(self):
        print("Winged animals can flap.")

class Bat(Mammal, WingedAnimal):
    pass

# create an object of Bat class
b1 = Bat()

b1.mammal_info()
b1.winged_animal_info()

Mammals can give direct birth.
Winged animals can flap.
```

```
In [440...]
class Mother():
    def Cooking(self):
        print("Mother Enjoys Cooking")

class Father():
    def Driving(self):
        print("Father Enjoys Driving")

class Child(Mother, Father):
    def Playing(self):
        print("Child Loves Playing")

c = Child()
c.Cooking()
c.Driving()
c.Playing()

Mother Enjoys Cooking
Father Enjoys Driving
Child Loves Playing
```

```
In [441...]
class Car():
    def Benz(self):
        print(" This is a Benz Car ")

class Bike():
    def Bmw(self):
        print(" This is a BMW Bike ")

class Bus():
    def Volvo(self):
        print(" This is a Volvo Bus ")

class Truck():
    def Eicher(self):
        print(" This is a Eicher Truck ")

class Plane():
    def Indigo(self):
        print(" This is a Indigo plane ")

class Transport(Car,Bike,Bus,Truck,Plane):
    def Main(self):
        print("This is the Main Class")
B=Transport()
B.Benz()
B.Bmw()
B.Volvo()
B.Eicher()
B.Indigo()
B.Main()

This is a Benz Car
This is a BMW Bike
This is a Volvo Bus
This is a Eicher Truck
This is a Indigo plane
This is the Main Class
```

## Polymorphism

```
In [442...]
10 + 15
"Artificial" + "Intelligence"

len([1,2,3,3,4])
len("machine learning")
len({"a":1 , "b":6})

def f (a, b):
```

```

        return a+b

print(f(15,12))
f("python" , " " "machine")
27
'python machine'

In [443...]
class Cat:
    def __init__(self , name , color):
        self.name= name
        self.color= color

    def info(self):
        print("the name of this cat is {} and the color is {}".format(self.name , self.color))

    def makesound(self):
        print(f" the sound is mew")



class Bird:
    def __init__(self , name , color):
        self.name= name
        self.color= color

    def info(self):
        print("the name of this bird is {} and the color is {}".format(self.name , self.color))

    def makesound(self):
        print(f" the sound is jikjik")

catobj=Cat("Galaxy" , "ocean blue")
birdobj=Bird("Fandogh" , "yellow")

for i in [catobj , birdobj]:
    i.makesound()

the sound is mew
the sound is jikjik

```

## Error handing

```

In [444...]
# 1) syntax ....           bitecode
for i range(10):
    pass

print("hi")

Cell In[444], line 3
  for i range(10):
  ^
SyntaxError: invalid syntax

In [ ]: #2) runtime      (expection)
#x= int(input("number: "))

#print(5 / x)

#-----
#print("r" + 4)
#-----
print(y + 6)

In [445...]
try:
    x= int(input("number: "))
    print(x)

except:
    print("Error!")

print("finish")
number: hg
Error!
finish

In [446...]
while True:
    try:
        n=int(input("enter your password: "))
        if n==1234:
            print("succesfull")
            break
    except:
        print("try again")



enter your password: 1235
enter your password: 1234
succesfull

In [447...]
try:
    x= int(input("number1: "))
    y= int(input("number2: "))
    z=x/y
    print(z)

except:
    print("ZeroDivisionError")

print("Hello World")
number1: 0
number2: 2
0.0
Hello World

In [448...]
try:
    x= int(input("number1: "))
    y= int(input("number2: "))
    z=x/y
    print(z)

```

```

except ZeroDivisionError:
    print("numbers can not be devided by zero")

except (ValueError,TypeError):
    print(" you should import just numbers")

#except TypeError:
#    #print("you can not concatinate string and int")

except:
    print("Error!!!")

number1: 5
number2: 0
numbers can not be devided by zero

```

```

In [449... try:
    x= int(input("number1: "))
    y= int(input("number2: "))
    z=x/y
    print(z)

except ZeroDivisionError :
    print("numbers can not be devided by zero")

except (ValueError, TypeError):
    print(" you should import just numbers")

except:
    print("Error!!!")

```

```

number1: 56
number2: 25
2.24

try:

    # code that might raise an exception

except SomeException:

    # code to handle the exception

else:

    # code to be executed if no exception occurs in the try block

```

```

In [450... def division(a, b, c):
    try:
        s = a + b
        result = s / c

    except ZeroDivisionError:
        print("Oops!! Cannot divide by zero")

    else:
        print(result)
        print("Division successful")

division(1, 2, 3)

```

```

1.0
Division successful

try:

    # code that might raise an exception

except SomeException:

    # code to handle the exception

else:

    # code to be executed if no exception occurs in the try block

finally:
    # code to be executed regardless of whether an exception occurred or not

```

```

In [451... try:
    def division(a,b,c):
        s=a+b
        result=s/c

except ZeroDivisionError:
    print("Cannot divide by zero!")

else:
    print("Division successful. Result:")

finally:
    print("This code will always be executed.")

division(1,2,3)

```

Division successful. Result:  
This code will always be executed.

## raise

```

In [452... def divide(x, y):
    if y == 0:
        raise ValueError ("oops!! Cannot divide by zero")
    return x / y

print(divide(10,5))
print(divide(10,0))

2.0

```

```

-----
ValueError                                Traceback (most recent call last)
Cell In[452], line 7
      4     return x / y
      5 print(divide(10,5))
----> 6 print(divide(10,0))

Cell In[452], line 3, in divide(x, y)
      1 def divide(x, y):
      2     if y == 0:
----> 3         raise ValueError ("oops!! Cannot divide by zero")
      4     return x / y

ValueError: oops!! Cannot divide by zero

```

## if sentence in one line(turnary operator)

```

In [453... number=int(input("number: "))

if number > 0 :
    print(number+1)

number: 10
11

In [454... num_=int(input("number: "))
if num_ > 0: print(num_+1)

number: 52
53

In [455... n=int(input("number: "))
if n> 0: print(n+1) ;print(n*3) ;print(n**2)

number: 23
24
69
529

In [456... list1=[]
m=int(input("number: "))
if m%2==0: list1.append(m)

print(list1)

number: 45
[]

In [457... weight=int(input("enter you weight: "))

if weight<50 : print("undeweight")
elif 50<=weight<80: print("fit")
elif weight>80: print("overweight")

enter you weight: 56
fit

In [458... age=int(input("enter you age: "))

if age>18:
    print(True)
else:
    print(False)

True if age>18 else False

enter you age: 25
True
True

Out[458]: True

In [459... age = 19

outcome = 'Go home.' if age < 18 else 'Welcome!'
print(outcome)

Welcome!

In [460... if x % 2 == 0:
    result = x * 2 + 10
else:
    result = x / 2 - 10

result = x * 2 + 10 if x % 2 == 0 else x / 2 - 10

```

## knapsack and greedyargorithm

```

In [ ]: names=["sofa" , "curtain" , "bookshelf" , "desk" , "bed" , "closet"]
values=[120,20,20,80,150,64]
sizes=[6,1,5,10,15,8]

class knapsack(object):
    def __init__(self , names , values , sizes) -> None:
        self.names=names
        self.values=values
        self.sizes=sizes

    def __str__(self) -> str:
        return f"the name of these objects are {self.names}\n the values are {self.values}\n and the sizes are {self.sizes} "

    def zipped(self):
        return list(zip(self.names , self.values , self.sizes))
    #output: [('sofa', 120, 6), ('curtain', 20, 1), ('bookshelf', 20, 5),
    #('desk', 80, 10), ('bed', 150, 15), ('closet', 64, 8)]

    #sort values
    def sortvalue(self):
        list1=self.zipped()
        return sorted(list1 , key=lambda x : x[1] , reverse=True)
    #([('bed', 150, 15), ('sofa', 120, 6), ('desk', 80, 10),
    #('closet', 64, 8), ('curtain', 20, 1), ('bookshelf', 20, 5)])

    def sortsize (self):
        list2=self.zipped()
        return sorted(list2 , key=lambda x : x[2])
    #([('curtain', 20, 1), ('bookshelf', 20, 5),
    #('sofa', 120, 6), ('closet', 64, 8), ('desk', 80, 10),
    #('bed', 150, 15)])

    def sort_value_size(self):
        list3=self.zipped()
        return sorted(list3 , key=lambda x : x[1]/x[2] , reverse=True)
    #([('sofa', 120, 6), ('curtain', 20, 1), ('bed', 150, 15),
    #('desk', 80, 10), ('closet', 64, 8), ('bookshelf', 20, 5)])

```

```

def greedyvalue(self , max_size):
    list4=self.sortvalue()
    total_value=0.0
    total_size=0.0
    result=[]
    for i in range(len(list4)):
        if total_size + list4[i][2] <=max_size:
            total_size+= list4[i][2]
            total_value+=list4[i][1]
            result.append(list4[i])
    return f"the objects that you can select based on value are {result}\n and the total size is {total_size}\n and the total values are {total_value} "

def greedysize(self , max_size):
    list5=self.sortsize()
    total_size=0.0
    total_value=0.0
    result=list()
    for j in range(len(list5)):
        if total_size + list5[j][2]<=max_size:
            total_size+= list5[j][2]
            total_value+=list5[j][1]
            result.append(list5[j])
    return f"the objects that you can select based on size are {result}\n and the total size is {total_size}\n and the total values are {total_value} "

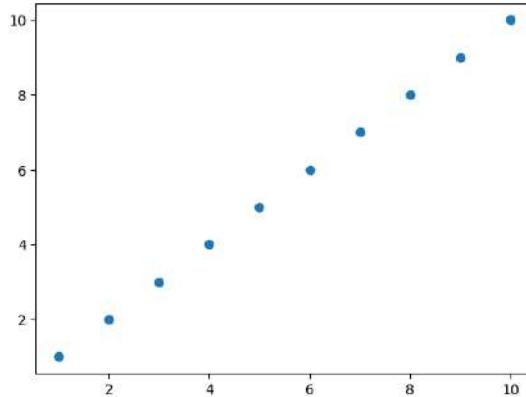
def greedyvaluesize(self , max_size):
    list6=self.sort_value_size()
    total_size=0.0
    total_value=0.0
    result=[]
    for z in range(len(list6)):
        if total_size + list6[z][2]<=max_size:
            total_size+= list6[z][2]
            total_value+=list6[z][1]
            result.append(list6[z])
    return f"the objects that you can select based on value to size are {result}\n and the total size is {total_size}\n and the total values are {total_value} "

obj=knapsack(names,values , sizes)
print(obj.greedyvalue(15))
print("*" *20)
print(obj.greedysize(15))
print("*" *20)
print(obj.greedyvaluesize(15))

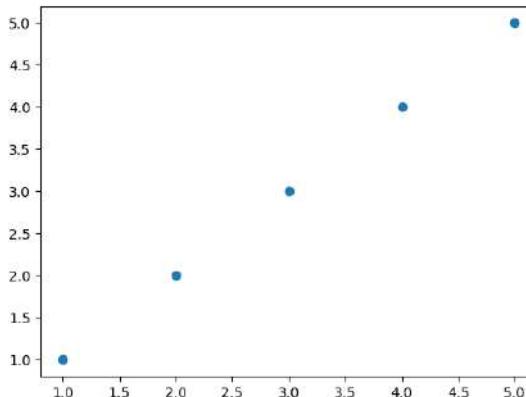
```

## Matplotlib

```
In [463...]: import matplotlib.pyplot as plt
X=[1,2,3,4,5,6,7,8,9,10]
y=[1,2,3,4,5,6,7,8,9,10]
plt.scatter(X, y)
plt.show()
```

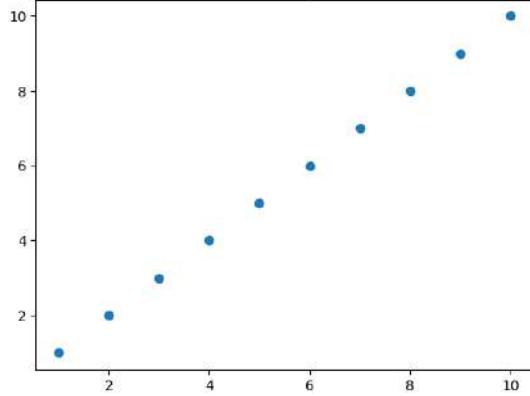


```
In [472]: plt.scatter(X[:5] , y[:5])
plt.show()
```



```
In [473]: #title
X2=[1,2,3,4,5,6,7,8,9,10]
y2=[1,2,3,4,5,6,7,8,9,10]
plt.scatter(X, y)
plt.title("simple equation graph\n x and y")
plt.show()
```

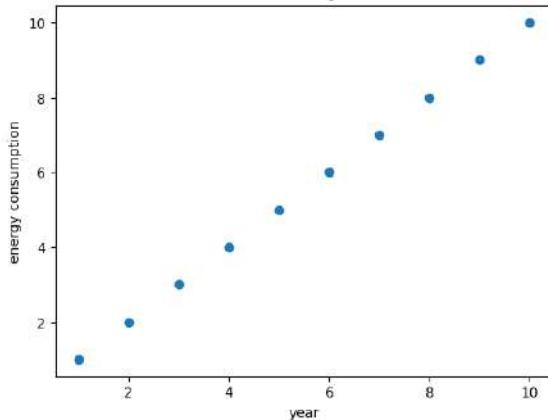
simple equation graph  
x and y



```
In [4]: #Label
X2=[1,2,3,4,5,6,7,8,9,10]
y2=[1,2,3,4,5,6,7,8,9,10]

plt.scatter(X2, y2)
plt.xlabel("year")
plt.ylabel("energy consumption")
plt.title("simple equation graph\n x and y")
plt.show()
```

simple equation graph  
x and y

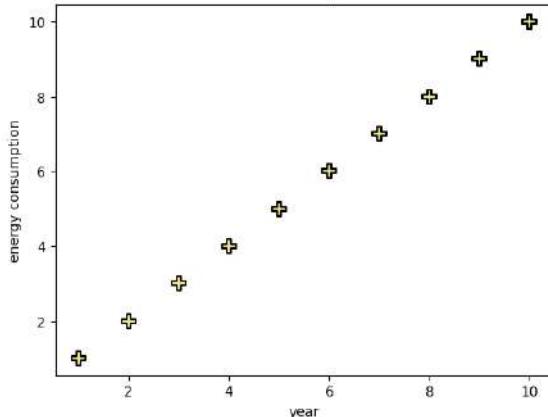


```
In [9]: #color, marker , ...
X2=[1,2,3,4,5,6,7,8,9,10]
y2=[1,2,3,4,5,6,7,8,9,10]

#color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.scatter(X2, y2,s=100 , c= "khaki" , marker="P", alpha = 1, linewidths=1.5 , edgecolors="black" )
plt.xlabel("year")
plt.ylabel("energy consumption")
plt.title("simple equation graph\n x and y")
plt.show()
```

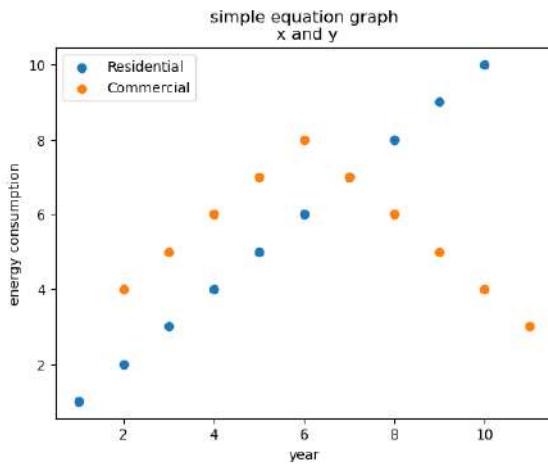
simple equation graph  
x and y



```
In [10]: #Legend
X3=[1,2,3,4,5,6,7,8,9,10]
y3=[1,2,3,4,5,6,7,8,9,10]

X4=[2,3,4,5,6,7,8,9,10,11]
y4=[4,5,6,7,8,7,6,5,4,3]

plt.scatter(X3, y3 , label="Residential")
plt.scatter(X4, y4, label= "Commercial")
plt.xlabel("year")
plt.ylabel("energy consumption")
plt.legend()
plt.title("simple equation graph\n x and y")
plt.savefig(r"D:\pic.png")
plt.show()
```

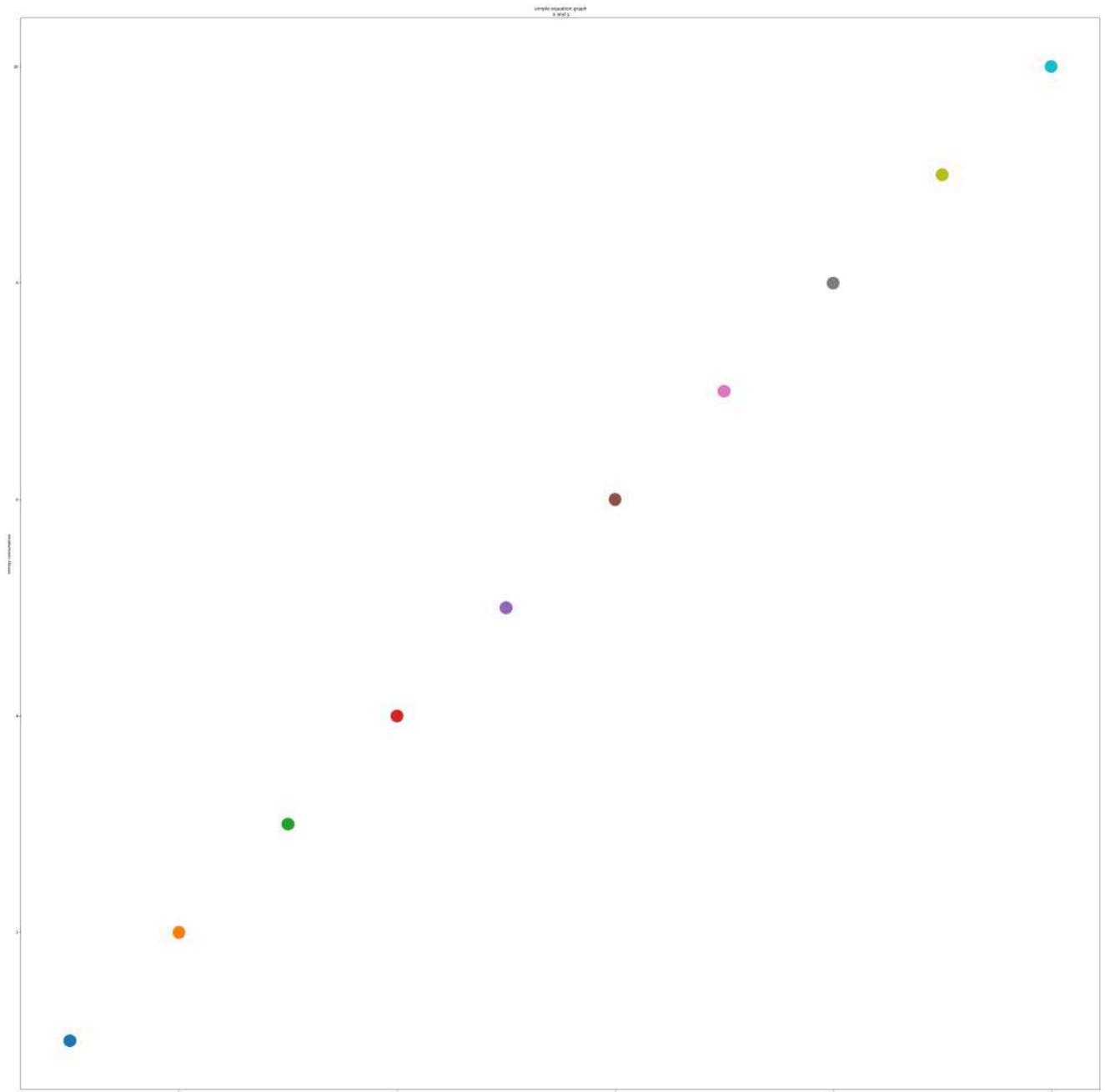


```
In [11]: #figure
X2=[1,2,3,4,5,6,7,8,9,10]
y2=[1,2,3,4,5,6,7,8,9,10]

color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(50,50) , dpi=75)
plt.scatter(X2, y2 ,s=1000 , color= color , marker="o", alpha = 1, linewidths=1.5 )
plt.xlabel("year")
plt.ylabel("energy consumption")
plt.title("simple equation graph\n x and y")

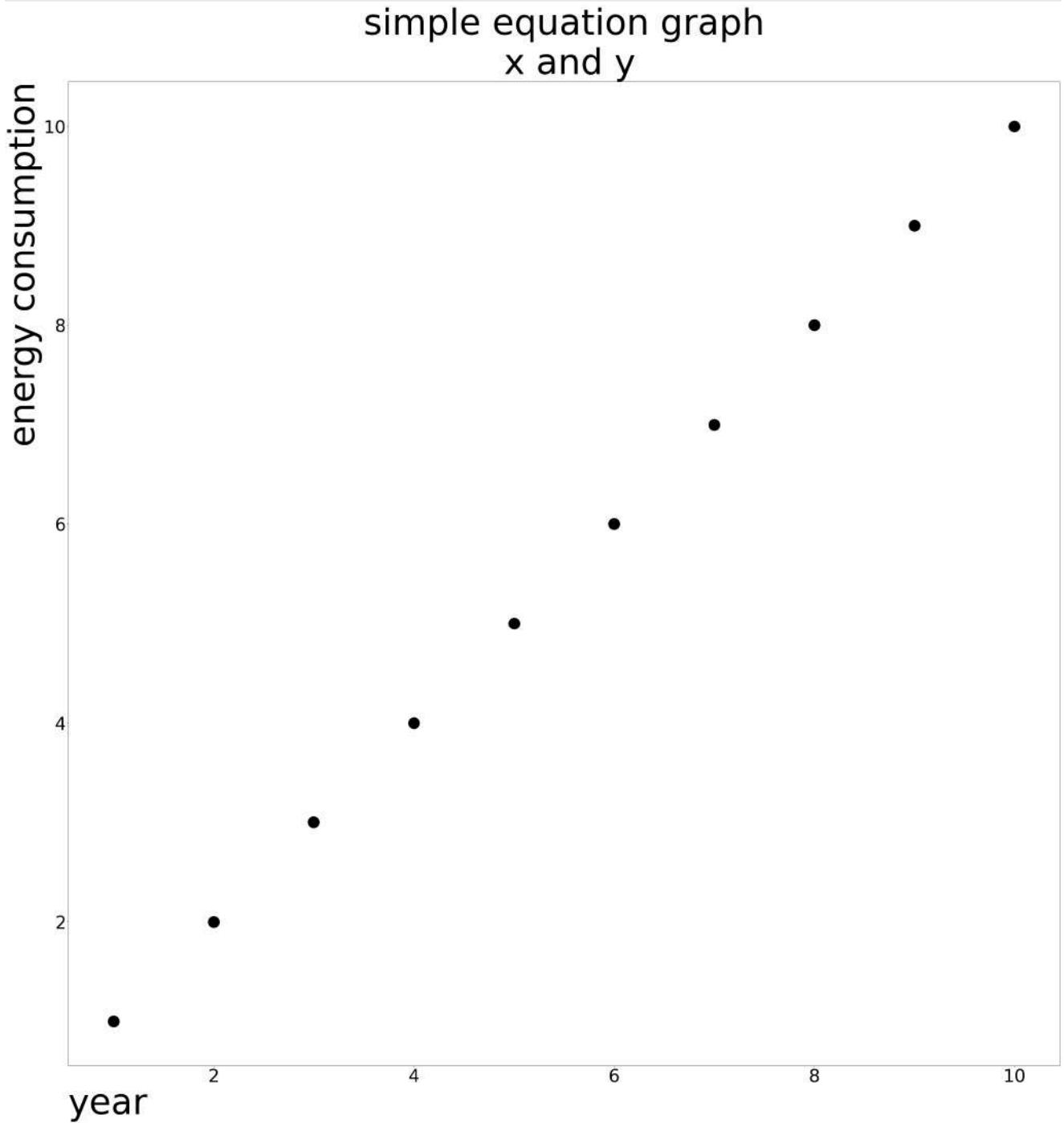
plt.show()
```



```
In [13]: #figure
X2=[1,2,3,4,5,6,7,8,9,10]
y2=[1,2,3,4,5,6,7,8,9,10]

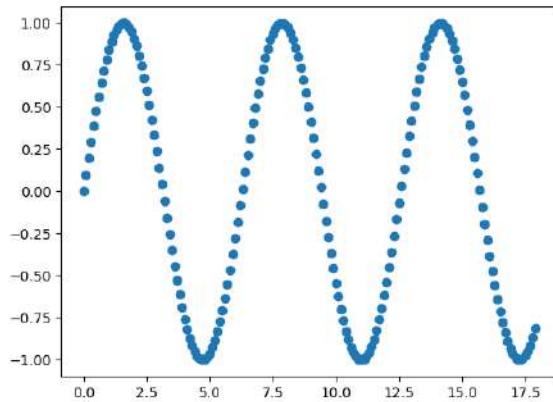
#color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

plt.figure(figsize=(50,50) , dpi=75 )
plt.scatter(X2, y2 ,s=1000 , c= "k" , marker="o" , alpha = 1, linewidths=1.5 )
plt.xlabel("year" ,fontsize=100 , loc="center")
plt.ylabel("energy consumption" ,fontsize=100 , loc = "top")
# Set the font size of xticks
plt.xticks(fontsize=50 )
plt.yticks(fontsize=50)
plt.title("simple equation graph\n x and y" , fontsize=100)
plt.show()
```



```
In [15]: from math import cos , sin
x=[i*0.1 for i in range(0, 180 )]
y=[sin(i) for i in x]

plt.scatter(x , y)
plt.show()
```

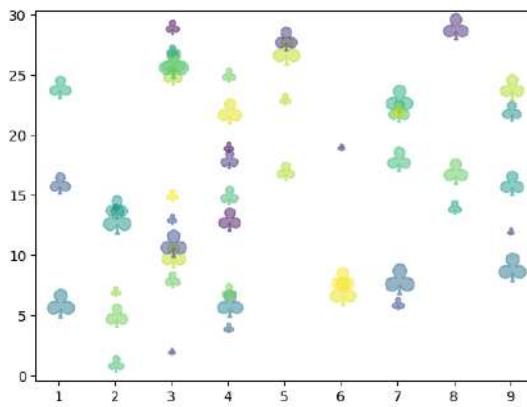


```
In [21]: import random

random.seed(28)
x = [random.randrange(1,10) for i in range(50)]
y = [random.randrange(1,30) for i in range(50)]

colors = [random.randrange(1,300) for i in range(50)]
area = [30*random.randrange(1,20) for i in range(50)]

plt.scatter(x, y, s=area,c=colors , alpha=0.5 , marker=r'$\clubsuit$')
plt.show()
```



```
In [ ]: #exercise
#day one, the age and speed of 13 cars:
x1 = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y1 = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x2 = [2,2,8,1,15,8,12,9,7,3,11,4,7,14,12]
y2 = [100,105,84,105,90,99,90,95,94,100,79,112,91,80,85]

print("-----")

# dataset-1
x3 = [89, 43, 36, 36, 95, 10,
       66, 34, 38, 20]

y3 = [21, 46, 3, 35, 67, 95,
      53, 72, 58, 10]

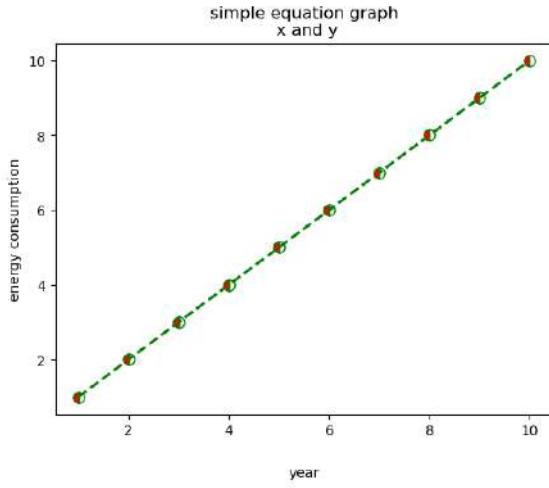
# dataset2
x4 = [26, 29, 48, 64, 6, 5,
       36, 66, 72, 40]

y4 = [26, 34, 90, 33, 38,
      28, 56, 2, 47, 15]
```

```
In [32]: #plot
import matplotlib.pyplot as plt

X2=[1,2,3,4,5,6,7,8,9,10]
Y2=[1,2,3,4,5,6,7,8,9,10]

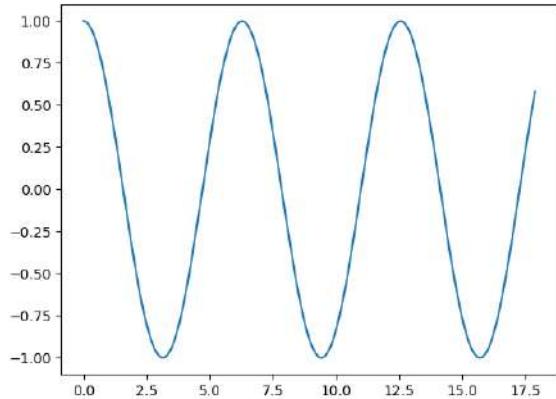
plt.plot(X2, Y2 , alpha = 1, color='green' , marker="o" ,markersize=8,
         fillstyle="left" ,markerfacecolor="red",
         linestyle="dashed" , linewidth=2 )
plt.xlabel("year" , labelpad=20)
plt.ylabel("energy consumption")
plt.title("simple equation graph\n x and y")
plt.show()
```



```
In [35]: #exercise
from math import cos

x=[i*0.1 for i in range(0, 180 )]
y=[cos(i) for i in x]

plt.plot(x , y)
#plt.xlim(0,5)
#plt.ylim(0,1)
plt.show()
```



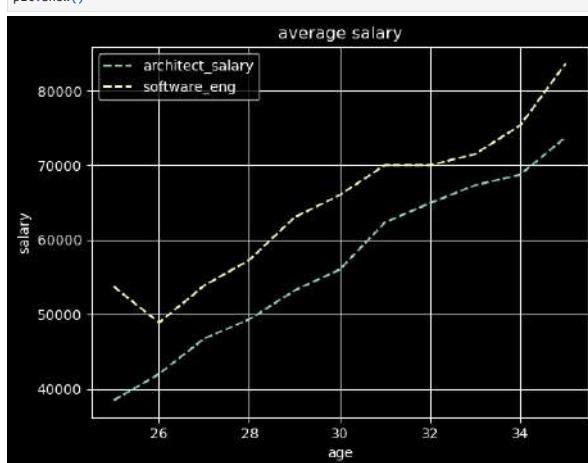
```
In [36]: print(plt.style.available)

['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

```
In [37]: age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83640]
```

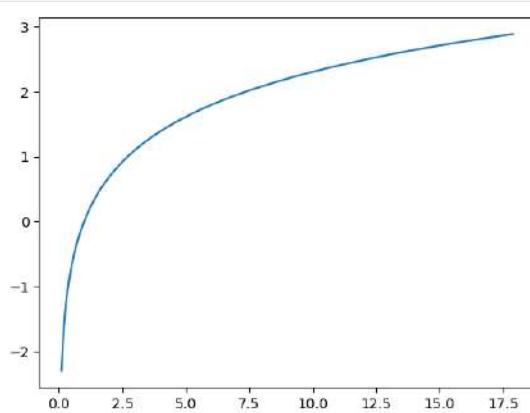
```
plt.style.use("dark_background")
plt.plot(age_average , architect_salary , linestyle="dashed" , label="architect_salary")
plt.plot(age_average , software_salary , linestyle="dashed" , label="software_eng")
plt.title("average salary")
plt.xlabel("age")
plt.ylabel("salary")
plt.legend()
plt.grid()

plt.show()
```



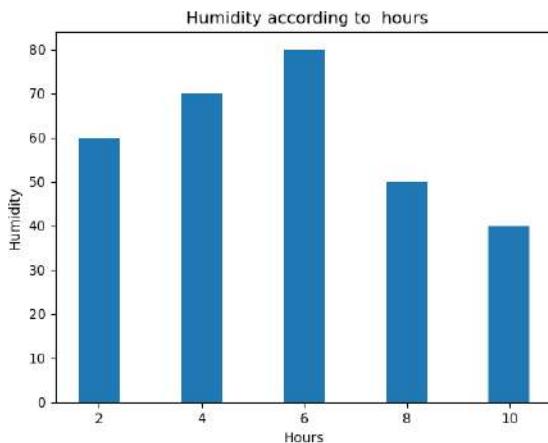
```
In [1]: import matplotlib.pyplot as plt
from math import log
x=[i*0.1 for i in range(1,180 )]
y=[log(j) for j in x]
```

```
#plt.xkcd()  
plt.plot(x , y)  
plt.show()
```



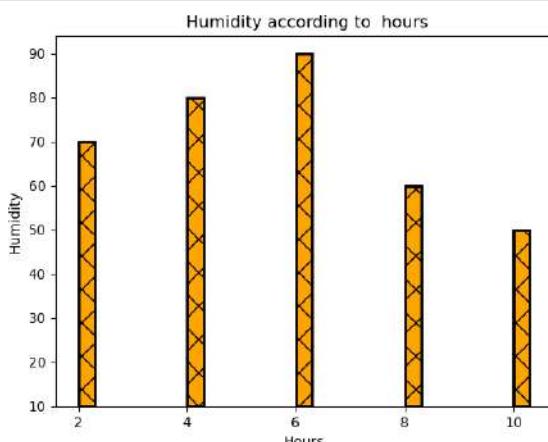
In [2]: #Bar chart

```
X=[2,4,6,8,10]  
y=[60,70,80,50,40]  
  
plt.bar(X, y)  
plt.xlabel("Hours")  
plt.ylabel("Humidity")  
plt.title(" Humidity according to hours")  
plt.show()
```



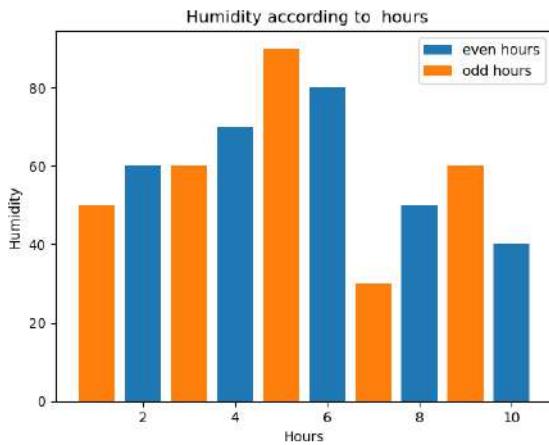
In [5]: import matplotlib.pyplot as plt

```
X=[2,4,6,8,10]  
y=[60,70,80,50,40]  
  
plt.bar(X, y , width=0.3 , bottom=10 , align="edge" , color="orange", edgecolor="k" , linewidth=2 , hatch='x' )  
plt.xlabel("Hours")  
plt.ylabel("Humidity")  
plt.title(" Humidity according to hours")  
plt.show()
```



In [6]: X=[2,4,6,8,10]  
y=[60,70,80,50,40]

```
X2=[1,3,5,7,9]  
y2=[50,60,90,30,60]  
  
plt.bar(X, y , label="even hours")  
plt.bar(X2, y2, label= "odd hours")  
plt.xlabel("Hours")  
plt.ylabel("Humidity")  
plt.legend()  
plt.title(" Humidity according to hours")  
plt.show()
```



```
In [9]: import matplotlib.pyplot as plt
import numpy as np
#bar
age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83640]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]

x_index=np.arange(len(age_average))
width=0.25

plt.bar(x_index , architect_salary , label="architect_salary" , color="m" , width=width )
plt.bar(x_index , software_salary , label="software_eng" , color="r" , width=width)
plt.bar(x_index , dentist_salary , label="dentist_salary" , color="c" , width=width)
plt.xticks( x_index ,age_average)
plt.title("average salary")

plt.legend()
plt.grid()

plt.show()

print(type(x_index))
```



```
In [15]: import matplotlib.pyplot as plt
import numpy as np
#bar
age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83640]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]

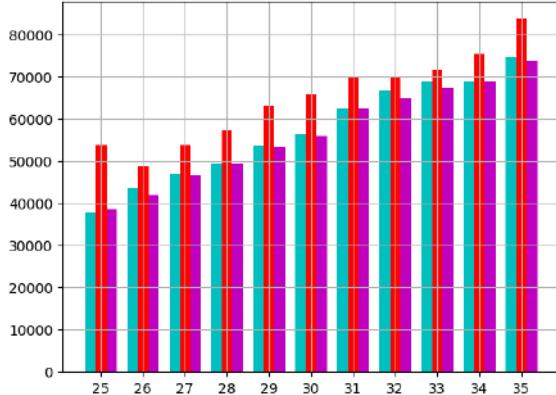
x_index=[i for i in range(len(age_average))]
x_arch=[i+0.25 for i in range(len(age_average))]
x_den=[i+0.25 for i in range(len(age_average))]

plt.bar(x_arch , architect_salary , label="architect_salary" , color="m" , width=0.25 )
plt.bar(x_index , software_salary , label="software_eng" , color="r" , width=0.25)
plt.bar(x_den , dentist_salary , label="dentist_salary" , color="c",width=0.25)
plt.xticks( x_index ,age_average)
plt.title("average salary")

# plt.legend()
plt.grid()

# plt.show()
```

average salary



```
In [17]: import matplotlib.pyplot as plt
import numpy as np
#bar
age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[5372,48876,53850,57287,63016,65999,70003,70000,71496,75370,83640]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68746,74583]

y_index=[i for i in range(len(age_average))]
y_arch=[i+0.25 for i in range(len(age_average))]
y_den=[i-0.25 for i in range(len(age_average))]
height=0.25

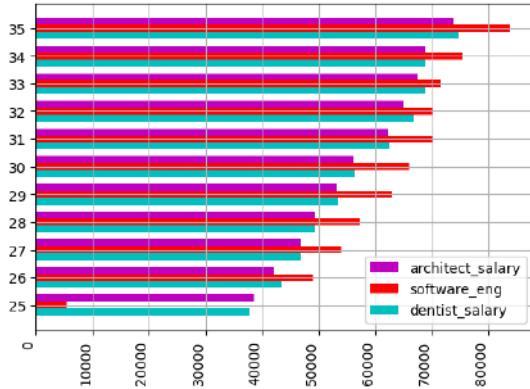
plt.barh(y_arch , architect_salary , label="architect_salary" , color="m" , height=height)
plt.barh(y_index , software_salary , label="software_eng" , color="r" , height=height)
plt.barh(y_den , dentist_salary , label="dentist_salary" , color="c" , height=height)
plt.yticks(y_index , age_average)

plt.gcf().autofmt_xdate(rotation=90) #gfc: get current format - change xaxis format

plt.title("average salary")

plt.legend()
plt.grid()
plt.show()
```

average salary



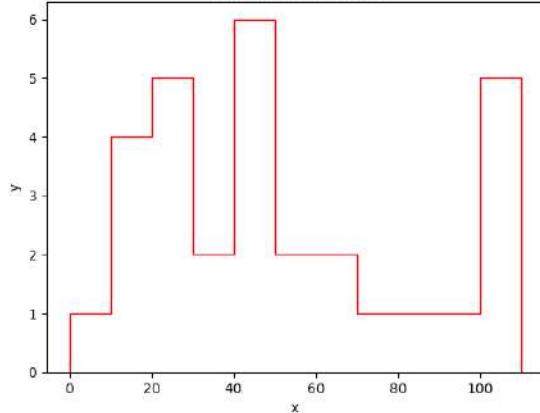
```
In [21]: #histogram
population_age=[22,55,62,42,42,99,21,22,45,102,110,106,113,16,17,66,43,73,82,24,31,15,2,104,48,49,51,17,36,28,101]
bins=[0,10,20,30,40,50,60,70,80,90,100,110]

plt.hist(population_age ,bins ,histtype="step",rwidth=0.8, color="red") #density=True :percent
#histtype : step , stepfilled

plt.xlabel("x")
plt.ylabel("y")

plt.title("population in village")
plt.show()
```

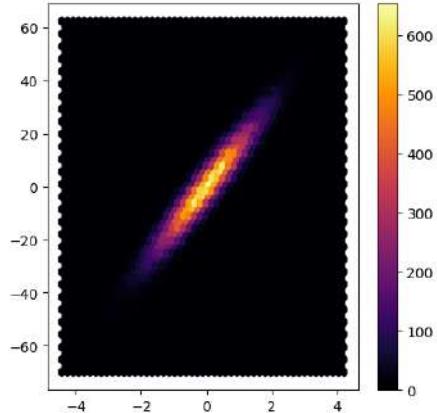
population in village



```
In [26]: #hexbin
import random
x=[random.gauss(0,1) for i in range(50000)]
y=[i**3 + random.gauss(0,1)*5 for i in x]

plt.figure(figsize=(5,5))
plt.hexbin( x,y ,gridsize=50 , cmap='inferno' )
plt.colorbar()

plt.show()
```



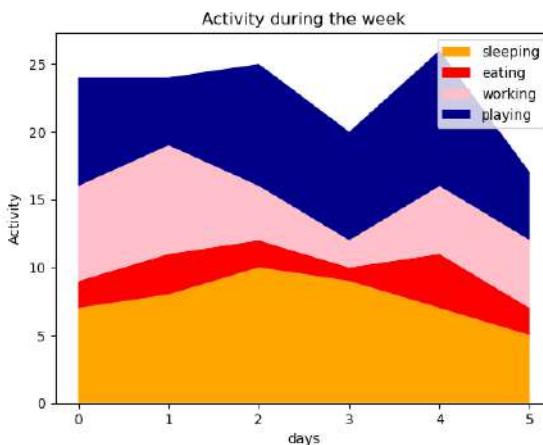
```
In [ ]: #exercise : hexbin
#random list between 1 and 10000
#cmap='plasma' with diffrent grid size
```

```
In [464...]: #stackplot

days=[i for i in range(6)]
#days=["monday" , "tuesday" , "wednesday" , "thursday" , "friday" , "saturday"]

sleeping=[7,8,10,9,7,5]
eating=[2,3,2,1,4,2]
working=[7,8,4,2,5,5]
playing=[8,5,9,8,10,5]

colors=["orange" , "red" , "pink" , "darkblue"]
plt.stackplot(days,sleeping,eating,working,playing, labels=["sleeping" , "eating" , "working" , "playing"] , colors=colors)
plt.xlabel("days")
plt.ylabel("Activity")
plt.title("Activity during the week")
#plt.gcf().autofmt_xdate(rotation=90)
plt.legend()
plt.show()
```

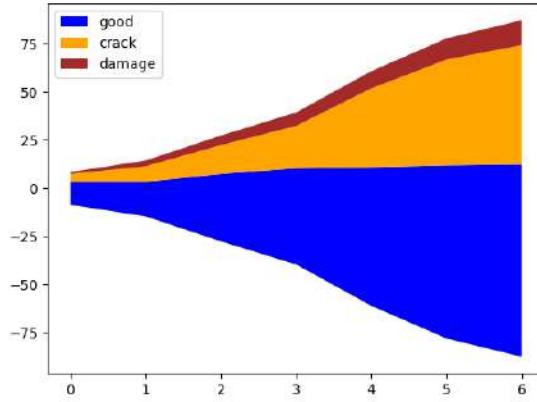


```
In [465...]: #exercise

# List of 7-days
days = [x for x in range(0, 7)]
# List of good appearance building
good = [12, 18, 35, 50, 72, 90, 100]
# List of buildings with cracks
crack = [4, 8, 15, 22, 41, 55, 62]
# List of damaged buildings
damage = [1, 3, 5, 7, 9, 11, 13]

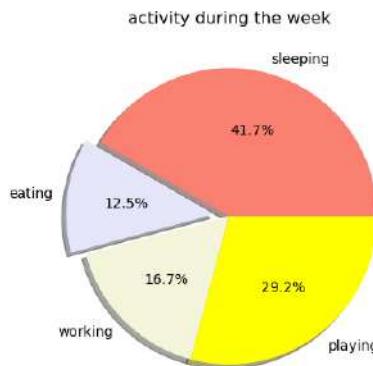
plt.stackplot(days, good, crack, damage,
             baseline ='sym',                      #baseline : {'zero', 'sym'}
             colors =[ 'blue' , 'orange' ,
                      'brown' ] , labels=[ "good" , "crack" , "damage"])

plt.legend(loc="upper left") # ~2
plt.show()
```

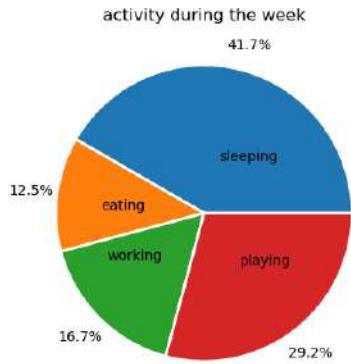


```
In [466]: #pie chart
slices=[10,3,4,7]
activities=["sleeping", "eating", "working", "playing"]
plt.pie(slices, labels= activities, colors=["salmon", "lavender", "beige","yellow"] ,
shadow=True , explode=(0,0.1,0,0) , autopct='%1.1f%%')

plt.title("activity during the week")
plt.show()
```



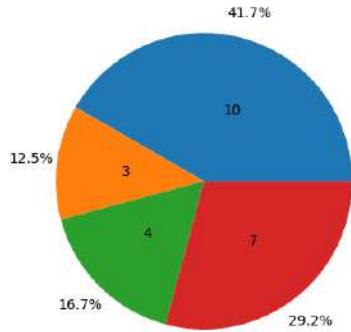
```
In [467]: #pie chart
slices=[10,3,4,7]
activities=["sleeping", "eating", "working", "playing"]
plt.pie(slices , labels= activities , autopct='%.1f%%' , pctdistance=1.18, labeldistance=0.4 ,
wedgeprops = {"linewidth": 2, "edgecolor": "white"})
plt.title("activity during the week")
plt.show()
```



```
In [468]: slices=[10,3,4,7]
activities=["sleeping", "eating", "working", "playing"]

plt.pie(slices , labels= slices , autopct='%.1f%%' , pctdistance=1.18, labeldistance=0.5 )
plt.title("activity during the week")
plt.show()
```

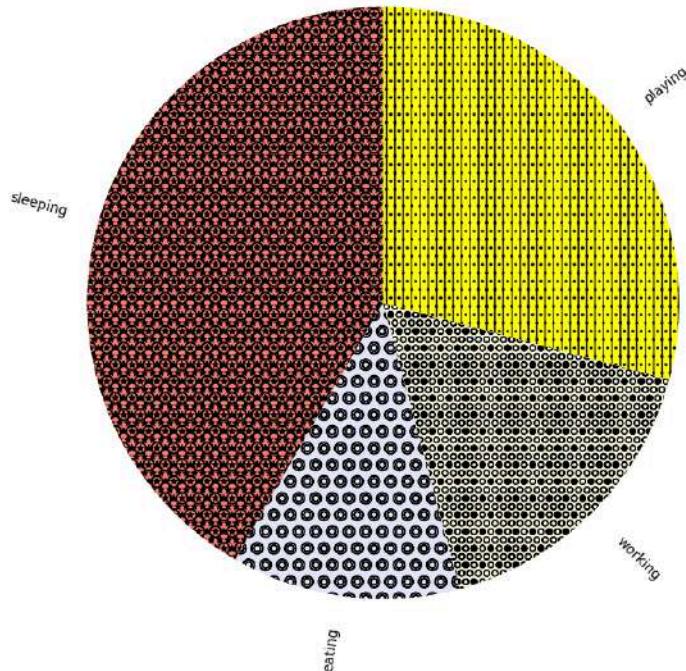
activity during the week



```
In [469]: slices=[10,3,4,7]
activities=["sleeping" , "eating" , "working" , "playing"]

plt.pie(slices ,labels=activities, colors=["salmon" , "lavender" , "beige","yellow"]
        ,hatch=['**0' , 'oO' , 'O.O' , '.||.|'] , radius=2 ,startangle=90 , rotatelabels = True)

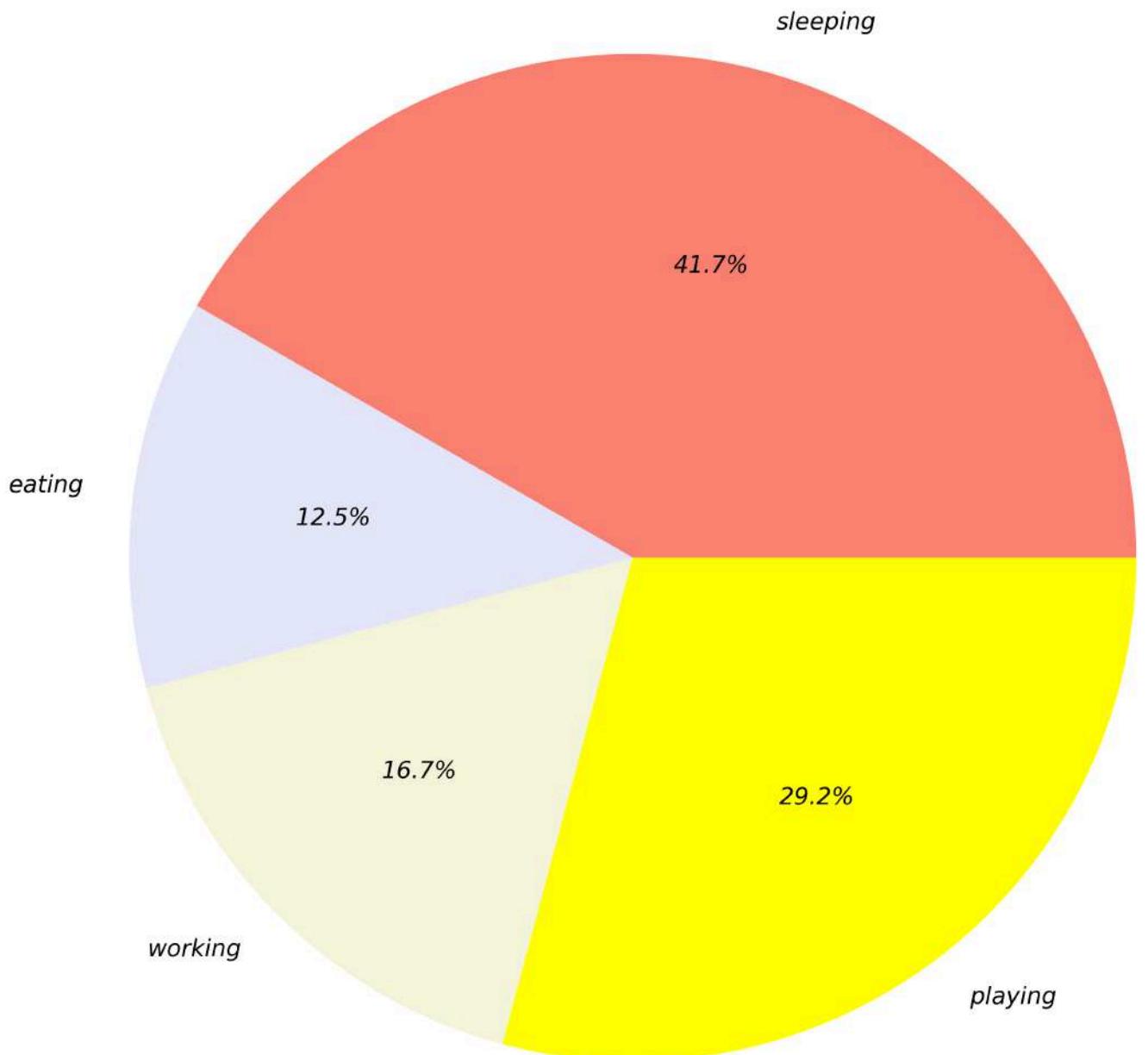
plt.show()
```



```
In [470]: slices=[10,3,4,7]
activities=["sleeping" , "eating" , "working" , "playing"]

plt.pie(slices ,labels=activities, colors=["salmon" , "lavender" , "beige","yellow"]
        , radius=10 , autopct='%1.1f%%' , textprops={'fontsize': 50 , "style":"italic"} )

plt.show()
```



```
In [471... #excercise
# Data
labels = ["G1", "G2", "G3", "G4", "G5"]      #counterClock = False
value = [12, 22, 16, 38, 12]
```

```
labels = ["G1", "G2", "G3", "G4", "G5"]
value = [0.1, 0.2, 0.1, 0.2, 0.1]           #normalize=False
```

```
import numpy as np
x,y=np.loadtxt("D:\\example.txt", delimiter=",", unpack=True)

plt.plot(x,y , label="load from file") plt.show()
```

```
In [472... help(plt.gcf().autofmt_xdate)
Help on method autofmt_xdate in module matplotlib.figure:

autofmt_xdate(bottom=0.2, rotation=30, ha='right', which='major') method of matplotlib.figure.Figure instance
    Date ticklabels often overlap, so it is useful to rotate them
    and right align them. Also, a common use case is a number of
    subplots with shared x-axis where the x-axis is date data. The
    ticklabels are often long, and it helps to rotate them on the
    bottom subplot and turn them off on other subplots, as well as
    turn off xlabel.

Parameters
-----
bottom : float, default: 0.2
    The bottom of the subplots for `subplots_adjust`.
rotation : float, default: 30 degrees
    The rotation angle of the xtick labels in degrees.
ha : {'left', 'center', 'right'}, default: 'right'
    The horizontal alignment of the xticklabels.
which : {'major', 'minor', 'both'}, default: 'major'
    Selects which ticklabels to rotate.
```

```
<Figure size 640x480 with 0 Axes>
age,arch,dent=np.loadtxt("D:\\example2.txt", delimiter=",", unpack=True , skiprows=1)

plt.plot(age , arch , color="r" , label="architecte_salary") plt.plot(age , dent , color="g" , label="dentist salary") plt.fill_between(age , arch , alpha=0.3) plt.legend() plt.show()

age,arch,dent=np.loadtxt("D:\\example2.txt", delimiter=",", unpack=True , skiprows=1)
```

```

plt.plot(age , arch , color="r" , label="architecure_salary") plt.plot(age , dent , color="g" , label="dentist salary") plt.fill_between(age , arch ,arch.mean(), alpha=0.3) #(length , first curve , second curve) plt.legend()
plt.show()

age,arch,dent=np.loadtxt(r"D:\example2.txt" , delimiter="," , unpack=True , skiprows=1)

plt.plot(age , arch , color="r" , label="architecure_salary") plt.plot(age , dent , color="g" , label="dentist salary") plt.fill_between(age , arch ,dent, alpha=0.3) #(length , first curve , second curve) plt.legend()
plt.show()

age,arch,dent=np.loadtxt(r"D:\example2.txt" , delimiter="," , unpack=True , skiprows=1)

plt.plot(age , arch , color="r" , label="architecure_salary") plt.plot(age , dent , color="g" , label="dentist salary") plt.fill_between(age , arch ,dent,where=arch>arch.mean() ,alpha=0.3 , color="orange") #(length ,
first curve , second curve) plt.legend() plt.show()

In [473... import matplotlib.pyplot as plt

In [474... age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83640]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]

fig , (ax1 ,ax2 ,ax3)=plt.subplots(nrows=3 , ncols=1)

# Set figure size
fig.set_size_inches(10, 8)

# Set figure title
fig.suptitle("My Figure Title")

# Set figure background color
fig.patch.set_facecolor('lightgray')

#-----
ax1.plot(age_average ,architect_salary , label="architect_salary" , color='red')
ax2.plot(age_average , software_salary , label="software salary")
ax3.plot(age_average , dentist_salary , label="dentist salary")

ax1.legend()
ax2.legend()
ax3.legend()

ax1.set_title("plot1")
ax1.set_xlabel("ages")
ax1.set_ylabel("median salary")

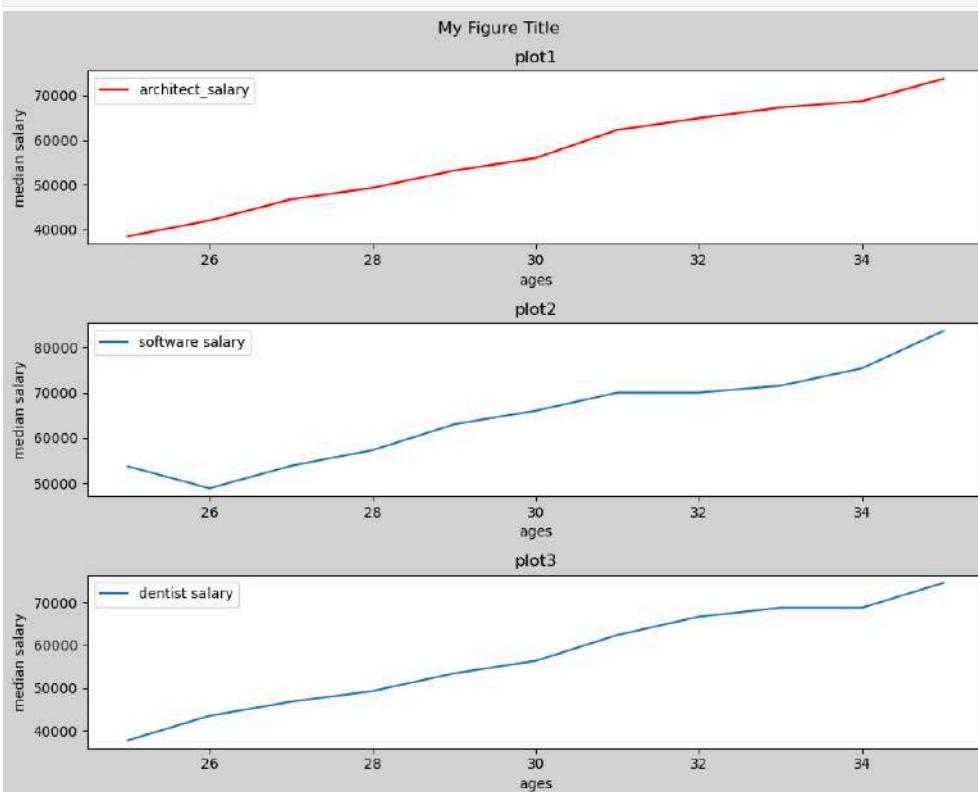
ax2.set_title("plot2")
ax2.set_xlabel("ages")
ax2.set_ylabel("median salary")

ax3.set_title("plot3")
ax3.set_xlabel("ages")
ax3.set_ylabel("median salary")

plt.tight_layout()
# Save the figure
fig.savefig(r"D:\my_figure.png" , dpi=300, bbox_inches='tight')

plt.show()

```



```

age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83640]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]

fig , (ax1 ,ax2 ,ax3)=plt.subplots(nrows=3 , ncols=1 , sharex=True )
#SHARE xticks

```

```

ax1.plot(age_average ,architect_salary , label="architect_salary" , color="orange" )
ax2.plot(age_average , software_salary , label="software salary", color="k")
ax3.plot(age_average , dentist_salary , label="dentist salary" ,color="purple")

ax1.legend()
ax2.legend()
ax3.legend()

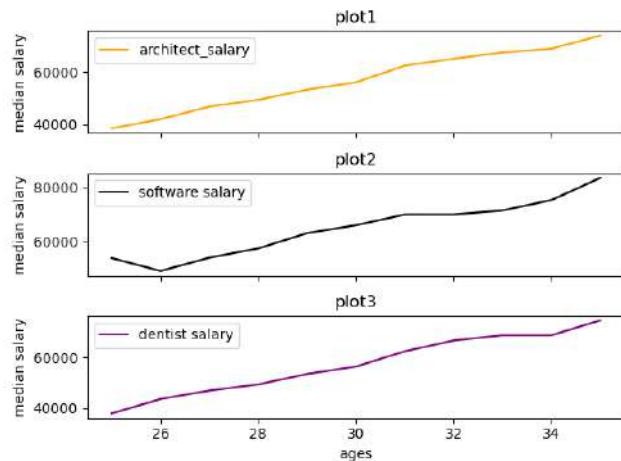
ax1.set_title("plot1")
ax1.set_ylabel("median salary")

ax2.set_title("plot2")
ax2.set_ylabel("median salary")

ax3.set_title("plot3")
ax3.set_xlabel("ages")
ax3.set_ylabel("median salary")

plt.tight_layout()
plt.show()

```



```

In [476...]
age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83648]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]

```

```

fig , (ax1 ,ax2 ,ax3)=plt.subplots(nrows=1 , ncols=3 , sharey=True )

ax1.plot(age_average ,architect_salary , label="architect_salary" , color="orange" )
ax2.plot(age_average , software_salary , label="software salary", color="k")
ax3.plot(age_average , dentist_salary , label="dentist salary" ,color="purple")

ax1.legend(fontsize=7)
ax2.legend(fontsize=7)
ax3.legend(fontsize=7)

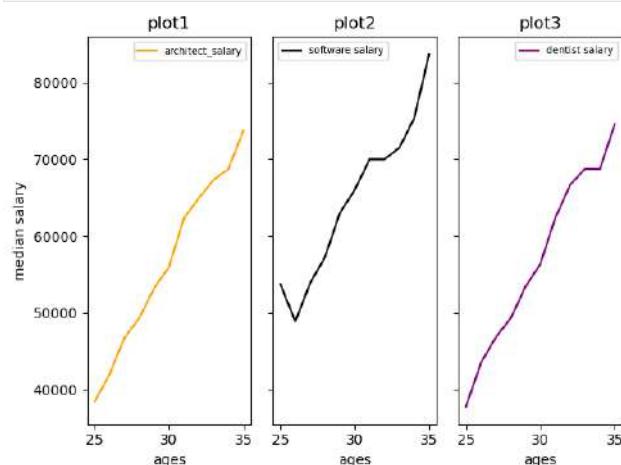
ax1.set_title("plot1")
ax1.set_xlabel("ages")
ax1.set_ylabel("median salary")

ax2.set_title("plot2")
ax2.set_xlabel("ages")

ax3.set_title("plot3")
ax3.set_xlabel("ages")

plt.tight_layout()          #Adjust the padding between and around subplots.
plt.show()

```



```

In [477...]
age_average=[25,26,27,28,29,30,31,32,33,34,35]
architect_salary=[38469,42000,46752,49320,53200,56000,62316,64928,67317,68749,73752]
software_salary=[53720,48876,53850,57287,63016,65999,70003,70000,71496,75370,83648]
dentist_salary=[37810,43515,46823,49293,53437,56373,62375,66674,68745,68746,74583]

```

```

fig1,(ax1 ,ax2)=plt.subplots(nrows=1 , ncols=2 , sharey=True )
fig2,ax =plt.subplots(nrows=1 , ncols=1)

ax1.plot(age_average ,architect_salary , label="architect_salary" , color="orange" )
ax2.plot(age_average , software_salary , label="software salary", color="k")
ax.plot(age_average , dentist_salary , label="dentist salary" ,color="purple")

```

```

plt.plot(age_average , dentist_salary , label="dentist salary" ,color="purple")
ax1.legend(fontsize=7)
ax2.legend(fontsize=7)
ax.legend(fontsize=7)

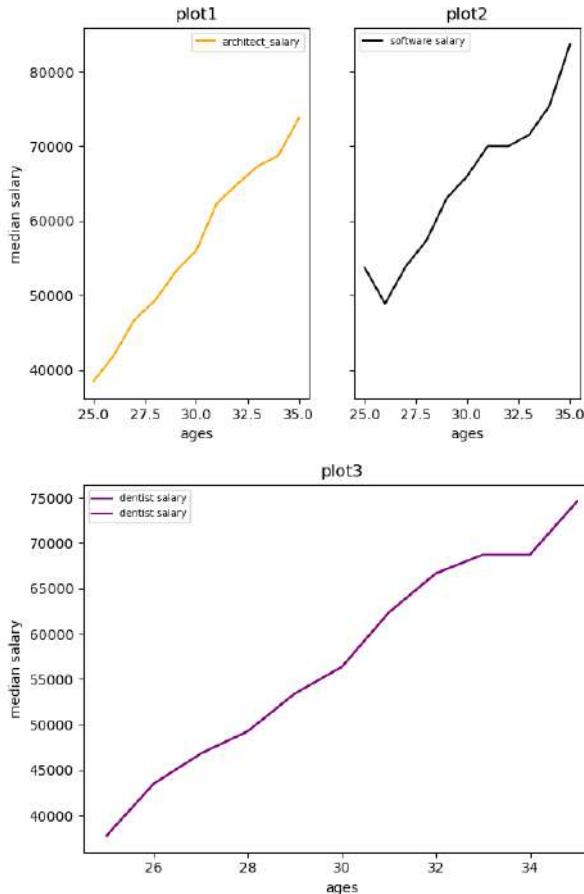
ax1.set_title("plot1")
ax1.set_xlabel("ages")
ax1.set_ylabel("median salary")

ax2.set_title("plot2")
ax2.set_xlabel("ages")

ax.set_title("plot3")
ax.set_xlabel("ages")
ax.set_ylabel("median salary")

plt.tight_layout(pad=1.8)           #Adjust the padding between and around subplots.
plt.show()

```



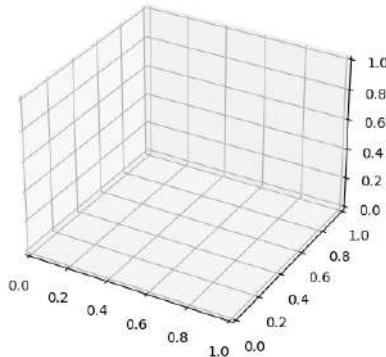
In [478...]

```

import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

plt.axes(projection='3d')
plt.show()

```



In [479...]

```

architect_salary=[38469, 42000, 46752, 49320, 53200, 56000, 62316, 64928, 67317, 68749, 73752]
software_salary=[53720, 48876, 53850, 57287, 63016, 65999, 70003, 70000, 71496, 75370, 83640]
dentist_salary=[37810, 43515, 46823, 49293, 53437, 56373, 62375, 66674, 68745, 68746, 74583]

fig = plt.figure(figsize=(10,10))
ax=plt.axes(projection='3d')

ax.scatter3D(architect_salary, software_salary, dentist_salary , color="red")
ax.plot3D(architect_salary, software_salary, dentist_salary, color="k")

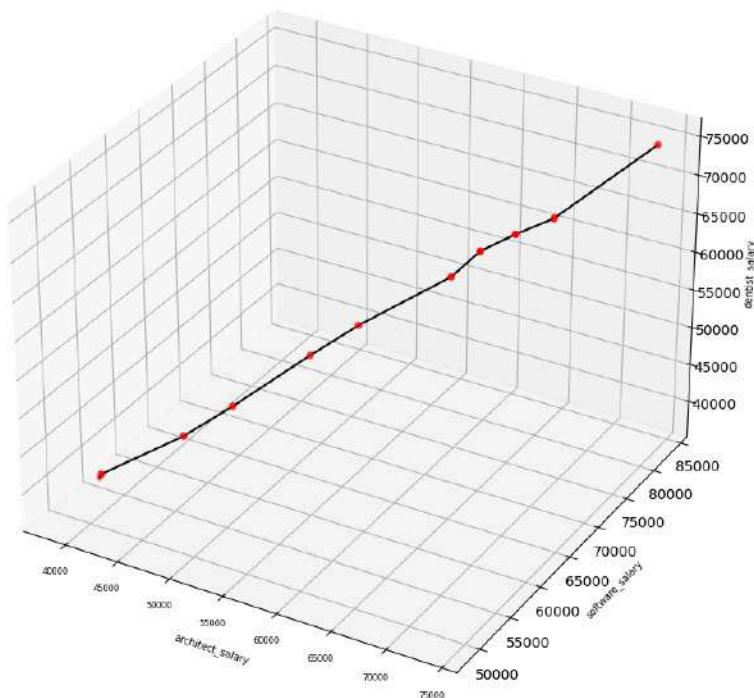
ax.set_title("3D plot")
ax.set_xlabel("architect_salary" , fontsize=7)
ax.set_ylabel("software_salary" , fontsize=7)
ax.set_zlabel("dentist_salary" , fontsize=7)

```

```
ax.tick_params(axis='x', labelsize=6)

plt.show()
```

3D plot



```
In [ ]: import random

ax=plt.axes(projection='3d')

x_vals = list(range(0, 100))
y_vals = list(range(0, 100))
z_vals= list(range(0, 100))

random.shuffle(x_vals)
random.shuffle(y_vals)
random.shuffle(z_vals)

ax.scatter3D(x_vals, y_vals, z_vals , color="orange")
plt.show()
```

## Pandas

```
In [482...]: import pandas as pd

buildingplaces=["kitechen" , "livigroom" , "bedroom" , "sittingroom" , "bath"]

df=pd.DataFrame(buildingplaces , index=[0,1,2,3,4] , columns=["labels"])
display(df)
```

labels
0 kitechen
1 livigroom
2 bedroom
3 sittingroom
4 bath

```
In [483...]: buildingplaces=[ "kitechen" , "livigroom" , "bedroom" , "sittingroom" , "bath"]

df=pd.DataFrame(buildingplaces)
display(df)
```

0
0 kitechen
1 livigroom
2 bedroom
3 sittingroom
4 bath

```
In [484...]: energy_consumption=[15,22,18,23,12,14,10,43,8,12,21,23]

columns=["January" , "February" , "March" , "April" , "May" , "June" ,
"July" , "August" , "September" , "October" , "November" , "December"]

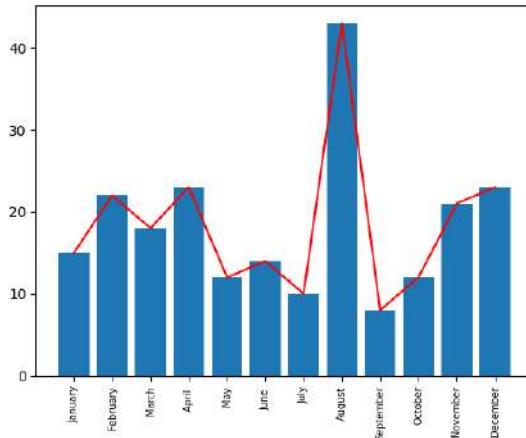
df2=pd.DataFrame(energy_consumption, columns=[ "month"])
df2
```

```
Out[484]:
```

	month
0	15
1	22
2	18
3	23
4	12
5	14
6	10
7	43
8	8
9	12
10	21
11	23

```
In [485...]
```

```
import matplotlib.pyplot as plt
plt.bar(columns , energy_consumption)
plt.plot(columns , energy_consumption, color="red")
plt.xticks(fontsize=7 , rotation=90 )
plt.show()
```



```
In [486...]
```

```
dataset={"windowSize": [120,150,170,210],
        "CeilingHeight": [3,3.20,4,6],
        "Insulation": [0,1,0,1],
        "EnergyConsumption (target)": [12,15,24,31]
       }
```

```
df3=pd.DataFrame(dataset)
display(df3)
```

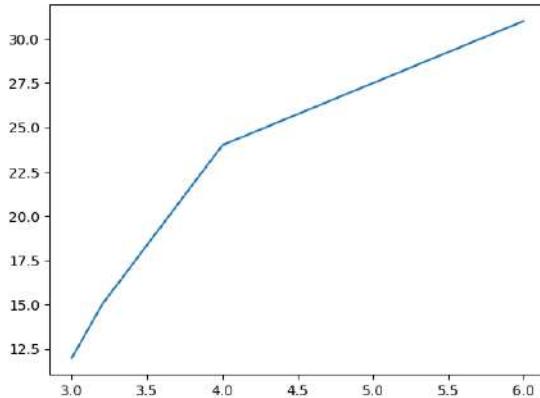
	windowSize	CeilingHeight	Insulation	EnergyConsumption (target)
0	120	3.0	0	12
1	150	3.2	1	15
2	170	4.0	0	24
3	210	6.0	1	31

```
In [487...]
```

```
print(type(df3))
<class 'pandas.core.frame.DataFrame'>
```

```
In [488...]
```

```
#plt.plot(dataset["windowSize"] ,dataset["CeilingHeight"])
#plt.plot(dataset["windowSize"] ,dataset["EnergyConsumption (target)"])
plt.plot(dataset["CeilingHeight"] ,dataset["EnergyConsumption (target)"])
plt.show()
```



```
In [489...]
```

```
print(df3.columns)
print("*"*20)
print(df3.index)
print("*"*20)
print(df3.values)
print("*"*20)
print(df3.value_counts)
```

```

Index(['windowSize', 'CeilingHeight', 'Insulation',
       'EnergyConsumption (target)'],
      dtype='object')
*****
RangeIndex(start=0, stop=4, step=1)
*****
[[120.   3.   0.  12. ]
 [150.   3.2  1.  15. ]
 [170.   4.   0.  24. ]
 [210.   6.   1.  31. ]]
*****
<bound method DataFrame.value_counts of    windowSize  CeilingHeight  Insulation  EnergyConsumption (target)
0          120           3.0          0            12
1          150           3.2          1            15
2          170           4.0          0            24
3          210           6.0          1            31>

```

## loc AND iloc

```

In [490... dataset={"windowSize": [120,150,170,210],
       "CeilingHeight": [3,3.2,4,6],
       "Insulation": [0,1,0,1],
       "EnergyConsumption (target)": [12,15,24,31]
     }

df2=pd.DataFrame(dataset , index=["sample1" , "sample2" , "sample3" , "sample4"])

display(df2)

display(df2.loc[["sample2"]])

```

	windowSize	CeilingHeight	Insulation	EnergyConsumption (target)
sample1	120	3.0	0	12
sample2	150	3.2	1	15
sample3	170	4.0	0	24
sample4	210	6.0	1	31

	windowSize	CeilingHeight	Insulation	EnergyConsumption (target)
sample2	150	3.2	1	15

```

In [491... dataset={"windowSize": [120,150,170,210],
       "CeilingHeight": [3,3.2,4,6],
       "Insulation": [0,1,0,1],
       "EnergyConsumption (target)": [12,15,24,31]
     }

df2=pd.DataFrame(dataset , index=["sample1" , "sample2" , "sample3" , "sample4"])

display(df2)

display(df2.loc[["sample2" , "sample3"]])

```

	windowSize	CeilingHeight	Insulation	EnergyConsumption (target)
sample1	120	3.0	0	12
sample2	150	3.2	1	15
sample3	170	4.0	0	24
sample4	210	6.0	1	31

	windowSize	CeilingHeight	Insulation	EnergyConsumption (target)
sample2	150	3.2	1	15
sample3	170	4.0	0	24

```

In [492... dataset={"windowSize": [120,150,170,210],
       "CeilingHeight": [3,3.2,4,6],
       "Insulation": [0,1,0,1],
       "EnergyConsumption (target)": [12,15,24,31]
     }

df2=pd.DataFrame(dataset , index=["sample1" , "sample2" , "sample3" , "sample4"])

#display(df2)

display(df2.loc["sample2":"sample3" , "windowSize":"Insulation"])

```

	windowSize	CeilingHeight	Insulation
sample2	150	3.2	1
sample3	170	4.0	0

```

In [493... dataset={"windowSize": [120,150,170,210],
       "CeilingHeight": [3,3.2,4,6],
       "Insulation": [0,1,0,1],
       "EnergyConsumption (target)": [12,15,24,31]
     }

df3=pd.DataFrame(dataset)

display(df3)
display(df3.loc[0:2 , "windowSize":"Insulation"])

display(df3.iloc[1:3])

```

	windowSize	CeilingHeight	Insulation	EnergyConsumption (target)
0	120	3.0	0	12
1	150	3.2	1	15
2	170	4.0	0	24
3	210	6.0	1	31

	windowSize	CeilingHeight	Insulation
0	120	3.0	0
1	150	3.2	1
2	170	4.0	0

windowSize	CeilingHeight	Insulation	EnergyConsumption	(target)
1	150	3.2	1	15
2	170	4.0	0	24

```
In [494]: display(df3["CeilingHeight"])
df3.CeilingHeight
0    3.0
1    3.2
2    4.0
3    6.0
Name: CeilingHeight, dtype: float64
Out[494]: 0    3.0
1    3.2
2    4.0
3    6.0
Name: CeilingHeight, dtype: float64
```

## csv : Comma Separated Values

```
In [ ]: table=pd.read_csv("D:\\example2.txt")
table
table.index=[1,2,3,4,5,6,7,8,9,10,11] #change columns table

#X1 : area # X2: area without opening # X3: overall height # X4:opening percentage # X5: maximum cooling temreture # X6:insulation # y: heating load dataset_energy= pd.read_csv("D:\\data_energy.csv") display(dataset_energy)

In [ ]: dataset_energy.shape

In [ ]: dataset_energy.count()

In [ ]: dataset_energy.head(8)

In [ ]: dataset_energy.tail(8)

In [ ]: dataset_energy.dtypes

In [ ]: dataset_energy.describe()

          #count : the number of not nulls
          #25%   : 25 % of the data are Lower than 83   after sorting
          #50%   : 50 % of the data are Lower than 116
          #75%   : 75 % of the data are Lower than 171.5
```

## Extract one column from DataFrame

```
In [ ]: dataset_energy["X2"]

In [ ]: dataset_energy.X2

In [ ]: dataset_energy.info()

In [ ]: dataset_energy[["X1" , "y"]]

In [ ]: #save file
newfile=dataset_energy[["X1" , "y"]]
newfile.to_csv(r"D:\\newfile.csv")

In [ ]: df=pd.read_csv(r"D:\\newfile.csv")
df

In [ ]:

newfile2=dataset_energy[["X1" , "y"]]
newfile2.to_csv(r"D:\\newfile2.csv", index=False)
df2=pd.read_csv(r"D:\\newfile2.csv") df2

In [ ]: #sort
dataset_energy.sort_values("y" , ascending=False)

In [ ]: #EXERCISE
import matplotlib.pyplot as plt

area=dataset_energy["X1"]
heating_load=dataset_energy["y"]

plt.scatter(area , heating_load , c="red" , marker="o")
plt.xlabel("area")
plt.ylabel("heating_load")
plt.show()

In [ ]: energy= dataset_energy["y"]
bins=[0,5,10,15,20,25,30]

plt.hist(energy ,bins,histtype="step", color="red" , linewidth=1.2)
plt.show()
```

## change column names

```
In [ ]: dataset_energy.columns=["a","b","c","d","e","f" , "h"]
dataset_energy

In [ ]: dataset_energy= pd.read_csv("D:\\data_energy.csv")
display(dataset_energy)

In [ ]: dataset_energy.rename(columns={"X1" : "area" , "y": "target"} , inplace=True)

In [ ]: dataset_energy

In [ ]: dataset_energy.drop("target" ,axis=1,inplace=True)
dataset_energy

In [ ]: dataset_energy= pd.read_csv("D:\\data_energy.csv" , usecols=["X1" , "X3" , "y"])
dataset_energy

In [ ]: #add new column to our dataframe

dataset_energy["Y2"] = [i for i in range(648)]
```

```

dataset_energy

In [ ]: dataset_energy["Y2"] = (dataset_energy.X1 + dataset_energy.X3)/2
dataset_energy

In [ ]: dataset_energy.loc[4:10]

In [ ]: dataset_energy.loc[4:10 , ["y" , "X1"]]

In [ ]: dataset_energy.iloc[4:10]

In [ ]: dataset_energy.iloc[4:10 , 0:3]

In [ ]: dataset_energy[dataset_energy.X1>120]

In [ ]: dataset_energy[dataset_energy.Y2==101.50]

In [ ]: dataset_energy[dataset_energy["X1"].isin([162])]
```

## NAN

```

In [ ]: df2=pd.read_csv(r"D:\energynan.csv")
df2

In [ ]: df2.describe()

In [ ]: df2.isnull()

In [ ]: df2.notnull()

In [ ]: df2.fillna(0)          #

In [ ]: df2.dropna()

In [ ]: df2.dropna(thresh=2)    #more than 2 nan remove

In [ ]: df2.dropna(how="all")   #if all items in a row is nan drop it

In [ ]: #exercise
df4=pd.read_csv(r"D:\tallest_buildings_global.csv")
display(df4)

In [ ]: df4.describe()

In [ ]: df4.isnull()

In [ ]: data5=df4.fillna(0)
data5

In [ ]: data5.drop("height_ft" , axis=1 , inplace= True)

data5

In [ ]: X=data5["height_m"]
y=data5.floors_above

plt.figure(figsize=(10,10) , dpi = 75)
plt.plot(X , y , color="k" , marker="o" , markersize=5 , markerfacecolor="red" , linestyle="dashed" )
plt.xlabel("height")
plt.ylabel("floor above")
plt.xticks(range(360,900 , 30) )
plt.savefig(r"D:\plot1.png")
plt.show()
```

```

In [ ]: unique_values = data5['country'].unique()
value_counts = data5['country'].value_counts()

In [ ]: plt.pie(slices , labels= activities , autopct='%1.1f%%' , pctdistance=1.18, labeldistance=0.4 ,
wedgeprops = {"linewidth": 2, "edgecolor": "white"})
plt.title("activity during the week")
plt.show()
```

```

In [ ]: import random
val=dict(value_counts)

slices=[i for i in val.values()]
labels=[j for j in val.keys()]
colors=[(random.random() , random.random() , random.random()) for z in range(len(slices))]

plt.pie(slices , labels= labels , colors=colors ,
shadow=True, autopct='%1.1f%%' , labeldistance=1.02 ,
pctdistance=0.5,
wedgeprops = {"linewidth": 2, "edgecolor": "white"},
textprops={'fontsize': 25 , "style":"italic"},
radius=5 ,startangle=90 , rotatelabels = True)
# Adding a circle in the center to create the donut shape
centre_circle = plt.Circle((0, 0), 1.2, color='white', linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```

val

```

In [ ]: unique_values

In [ ]: import matplotlib.pyplot as plt

# Assuming you have a dictionary 'val' with value counts
# Replace this with your actual dictionary
val = {'A': 10, 'B': 20, 'C': 15, 'D': 5}

slices = list(val.values())
labels = list(val.keys())

# Creating a donut-shaped pie chart
colors = plt.cm.Paired(range(len(slices)))
plt.pie(slices, labels=labels, autopct='%1.1f%%', startangle=90, counterclock=False,
labeldistance=1.1, colors=colors, wedgeprops=dict(width=0.4, edgecolor='w'))
```

```
# Adding a circle in the center to create the donut shape
centre_circle = plt.Circle((0, 0), 0.6, color='white', linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Donut-shaped Pie Chart')
plt.show()
```

# Machine learning

## linear Regression

Morteza khorsand



### Linear regression

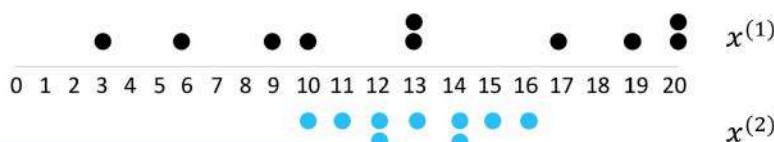
2

#### Mean

$$\mu = \bar{x} = \frac{x_1 + x_2 + x_3 + \dots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n (x_i)$$

$$x^{(1)} = [6, 20, 20, 9, 17, 13, 3, 10, 13, 19]$$

$$x^{(2)} = [10, 11, 13, 13, 16, 14, 12, 12, 15, 14]$$



$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

variance

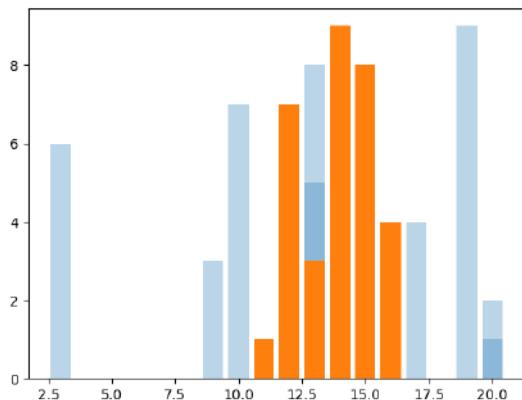
$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

standard deviation

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

```
In [1]: import matplotlib.pyplot as plt
x1=[6,20,20,9,17,13,3,10,13,19]
x2=[10,11,13,13,16,14,12,12,15,14]
y=[i for i in range(len(x1))]
```

```
plt.bar(x1 , y , alpha=0.3)
plt.bar(x2 , y )
plt.show()
```



```
In [2]: x1=[6,20,20,9,17,13,3,10,13,19]
x2=[10,11,13,13,16,14,12,12,15,14]

mu1= sum(x1)/len(x1)
mu2= sum(x2) / len(x2)

print(mu1 , "...." ,mu2)
```

```
13.0 .... 13.0
```

```
In [3]: #variance and standard deviation
from math import sqrt

counter=0

for i in range(len(x1)):
    counter+= (x1[i]- mu1)**2

var=counter/len(x1)
std=round(sqrt(var) , 2)

display(var , std)
```

```
32.4
5.69
```

```
In [4]: counter2 =0
for i in range(len(x2)):
    counter2+=(x2[i]- mu2)**2

var2=counter2 / len(x1)
sqrt2=round(sqrt(var2) , 2)

display(var2 , sqrt2)
```

```
3.0
1.73
```

### Linear Regression

$$Y = C + m X$$

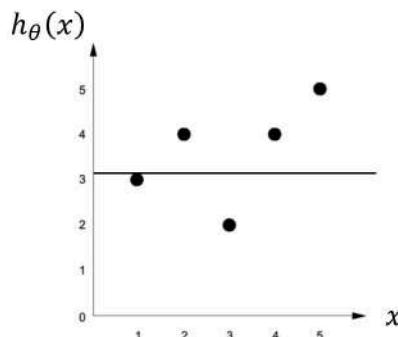
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$h_{\theta}(x) = 0 + x$$

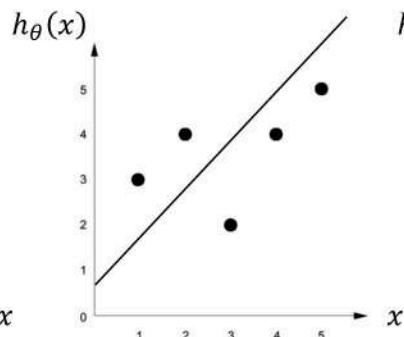
$$h_{\theta}(x) = 1 + 0.5 x$$

$$h_{\theta}(x) = 2 + (-2)x$$

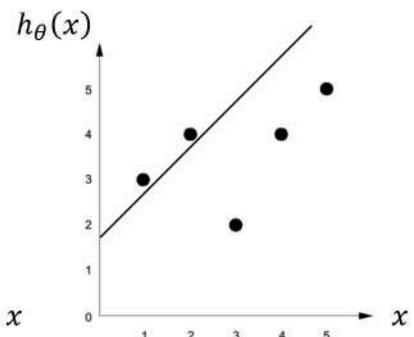
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



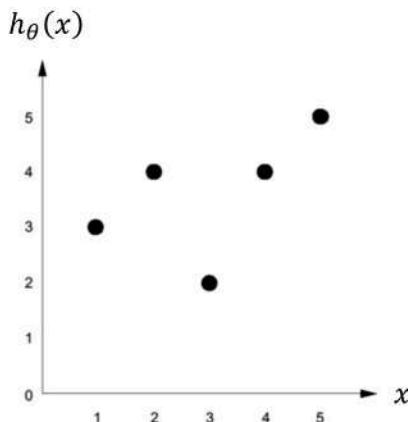
$$h_{\theta}(x) = 3.1 + 0x$$



$$h_{\theta}(x) = 0.75 + 1x$$



$$h_{\theta}(x) = 1.8 + 1x$$



X	y
1	3
2	4
3	2
4	4
5	5
$\bar{x} = 3$	$\bar{y} = 3.6$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$\theta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad \text{analytical method}$$



SCAN ME

<https://qrco.de/bdPcj1>

```
In [5]: X = [1, 2, 3, 4, 5]
y = [3, 4, 2, 4, 5]

x_bar = sum(X) / len(X)
y_bar = sum(y) / len(y)
print(f"y_bar={y_bar} and x_bar={x_bar}")

numerator=0
Denominator=0

for i in range(len(X)):
    numerator+= (X[i]- x_bar )*( y[i] - y_bar)
    Denominator+= (X[i]- x_bar ) **2
theta_1=numerator/Denominator
theta_0= y_bar- (theta_1*x_bar)
print(f" slope is {theta_1} , intercept is {theta_0}")

y_bar=3.6 and x_bar=3.0
slope is 0.4 , intercept is 2.4
```

## Model Evaluation

### COST FUNCTION

Mean Absolute Error

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

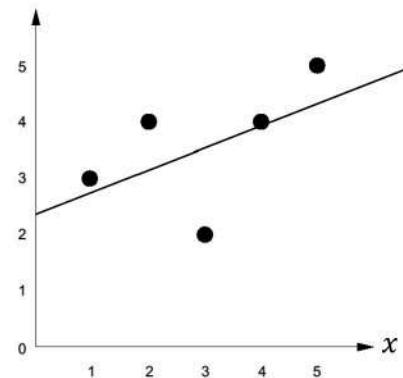
Mean Square Error

$$\text{MSE} = \frac{1}{2n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Root Mean Square Error

$$\text{RMSE} = \sqrt{\frac{1}{2n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

$$h_\theta(x)$$



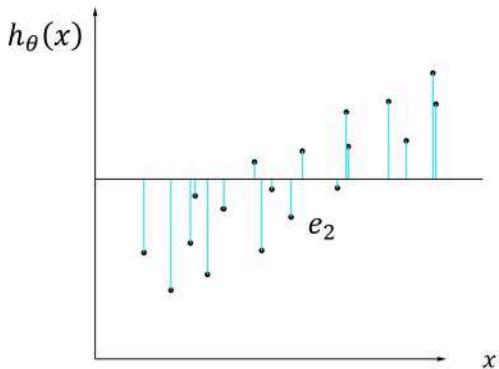
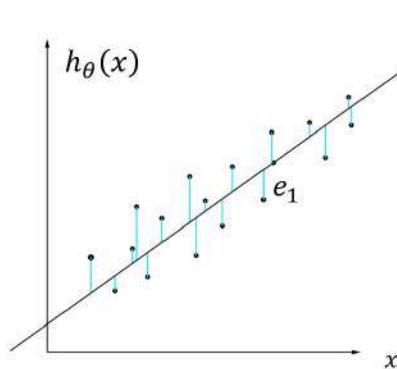
Relative Absolute Error – Residual Sum of Squre

$$\text{RAE} = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{\sum_{i=1}^n |y_i - \bar{y}|}$$

Relative Square Error

$$\text{RSE} = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} = \frac{e_1^2}{e_2^2}$$

$$R^2 = 1 - \text{RSE}$$



In [6]:

```
#y_hat
y_hat = []
for i in range(len(X)):
    y_hat.append(X[i]*theta_1 + theta_0)
print(y_hat)
```

```
[2.8, 3.2, 3.6, 4.0, 4.4]
In [7]: #calculating mean absolute error
MAE=0
for num in range(len(y)):
    MAE+= abs (y[i] - y_hat[i]) / len(y)
print(round(MAE,2))
0.6
```

```
In [8]: #calculating relative square error

numerator=0
Denominator=0
for i in range(len(y)):
    numerator+= (y[i]- y_hat[i]) **2
    Denominator+= (y[i] - y_bar) **2
rse= numerator/Denominator

r_square= 1-rse
print(f" the relative square error is {round(rse, 2)} , and R_squared is {round(r_square, 2)} ")
the relative square error is 0.69 , and R_squared is 0.31
```

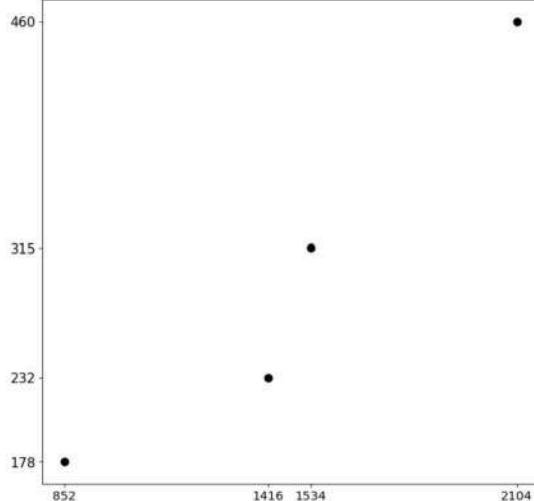
## Linear regression

8

SIZE (foot <sup>2</sup> )	PRICE (1000\$)
2104	460
1416	232
1534	315
852	178



$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



predict a house with 1200 foot square

```
X=[852, 1416, 1532, 2104]
y=[178, 232, 315, 460]

x_bar=sum(X) / len(X)
y_bar=sum(y)/ len(y)

display(f" X_bar is:{x_bar} , y_bar is:{y_bar}")

' X_bar is:1476.0 , y_bar is:296.25'
```

```
In [10]: numerator=0
Denominator=0
for i in range(len(X)):
    numerator+= (X[i]- x_bar )* (y[i] - y_bar)
    Denominator+= (X[i] - x_bar) **2
w=numerator/Denominator
b=y_bar- (w* x_bar)
print(f" slope is {w} , intercept is {b}")

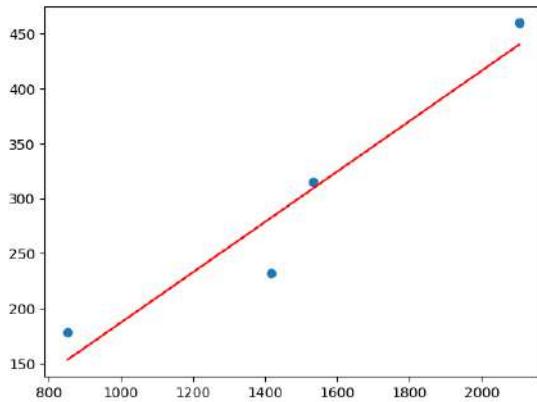
slope is 0.2296381006355035 , intercept is -42.695836538072285
```

```
In [11]: #y_hat = y_prediction
y_hat=[]
for i in range(len(X)):
    y_hat.append(X[i]* w + b)
print(y_hat)

[152.95582520341662, 282.471713961867, 309.1097336355908, 440.46272719912565]
```

```
In [12]: #plot
import matplotlib.pyplot as plt

plt.scatter(X ,y)
plt.plot(X ,y_hat , c= "red")
plt.show()
```



```
In [13]: y_predicted= w*1200+b
print(y_predicted)
232.8698422458813
```

```
In [14]: #calculating relative square error
numerator=0
Denominator=0
for i in range(len(y)):
    numerator+= (y[i]- y_hat[i] )**2
    Denominator+= (y[i] - y_bar) **2
rse= numerator/Denominator

r_squre= 1-rse

print(f' the relative square error is {round(rse , 2)} , and R_squre is {round(r_squre, 2)} ')
the relative square error is  0.08 , and R_squre is 0.92
```

## Linear regression

9

### Matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$A = [a_{ij}]_{m \times n}$$

Size (feet <sup>2</sup> )	#bedrooms	#floors	age	Price(1000\$)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$$X = \begin{bmatrix} 2104 & 5 & 1 & 45 \\ 1416 & 3 & 2 & 40 \\ 1534 & 3 & 2 & 30 \\ 852 & 2 & 1 & 36 \end{bmatrix}$$

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \leftarrow x_1^{(2)}$$

تعداد ویژگی‌ها  
ورودی‌ها در  $i$  امین نمونه آموزشی  
مقدار ویژگی زام در  $i$  امین نمونه آموزشی

$n$  □  
 $x^{(i)}$  □  
 $x_j^{(i)}$  □

## Main types

**Square matrix:** a matrix with the same number of rows and columns.

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

**Upper triangular matrix:** all entries, below the main diagonal, are zero.

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 0 & 5 & 8 \\ 0 & 0 & 9 \end{bmatrix}$$

**Lower triangular matrix:** all entries of  $A$  above the main diagonal are zero.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 3 & 6 & 9 \end{bmatrix}$$

**Diagonal matrix:** all the entries outside the main diagonal are all zero

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

**Scalar matrix:** a square matrix having a constant value for all the elements of the principal diagonal, and the other elements of the matrix are zero

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

**Identity matrix:** all the elements on the main diagonal are equal to 1 and all other elements are equal to 0.

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**zero matrix (null matrix)** : is a matrix all of whose entries are zero

$$O_{3 \times 2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

**transpose of a matrix:** an operator which flips a matrix over its diagonal

$$A = [1, 2, 3 \dots, n]_{1 \times n} \quad A^T = \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n \end{bmatrix}_{n \times 1} \quad A$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$A_{3 \times 4} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

$$A' = A_{4 \times 3}^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 3 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

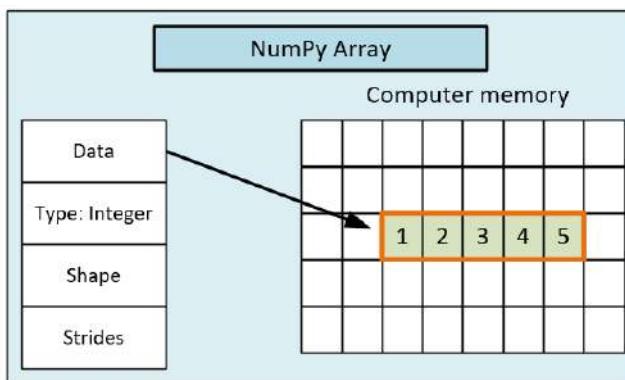
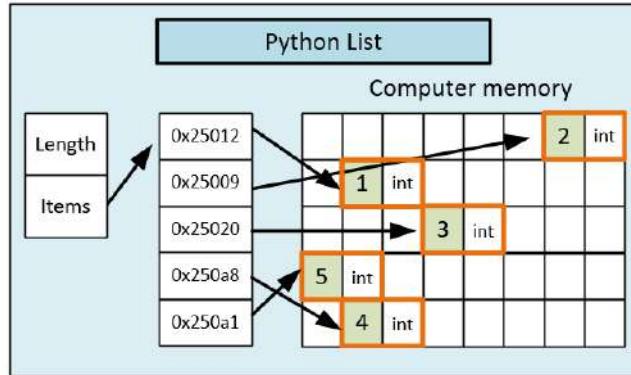
```
In [15]: import numpy as np
list_1 = [1,2,3,4,5]
matrix_1 = np.array(list_1)
```

```

print(matrix_1)
print(type(list_1) , type(matrix_1))
print(np.shape(matrix_1))
print(np.ndim(matrix_1))

[[1 2 3 4 5]
<class 'list'> <class 'numpy.ndarray'>
(5,)
1

```



$$\begin{bmatrix} [1, 2, 3] \\ [4, 5, 6] \\ [7, 8, 9] \end{bmatrix} = \begin{bmatrix} 1, 2, 3 \\ 4, 5, 6 \\ 7, 8, 9 \end{bmatrix}$$

2D Array    Matrix

**SCALER  
Topics**

```

#
In [16]: A = np.array ([[1,4,7],
                   [2,5,8],
                   [3,6,9]])
print(A)
print(np.shape(A))      #A.shape
print(A.ndim)

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

In [17]: ones=np.ones(shape=(3,4))
print(ones)

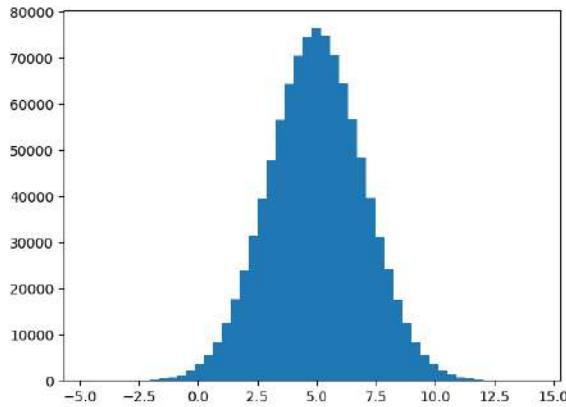
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

```

```
In [18]: oness=np.ones(shape=(3,3) , dtype=np.int32)
oness
Out[18]: array([[1, 1, 1],
   [1, 1, 1],
   [1, 1, 1]])
In [19]: #define a matrix all members=2
2*oness
Out[19]: array([[2, 2, 2],
   [2, 2, 2],
   [2, 2, 2]])
In [20]: eye=np.eye(3)      #identity matrix
print(eye)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
In [21]: print(3*eye )      #scalar matrix
[[3. 0. 0.]
 [0. 3. 0.]
 [0. 0. 3.]]
In [22]: print(np.zeros((3,4)))
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
In [23]: p= np.array([[3,3,5],
   [5,7,9]])
print(p)
print(*p*10)
e=np.transpose(p)      #p.transpose
print(e)
print(*e*10)
np.shape(e)
[[3 3 5]
 [5 7 9]]
*****
[[3 5]
 [3 7]
 [5 9]]
*****
(3, 2)
Out[23]: (3, 2)
In [24]: matrix1=np.array([[3,3,5],
   [5,7,9]])
matrix2=np.array([[1,4,7],
   [2,5,8]])
matrix1*matrix2
Out[24]: array([[ 4,  7, 12],
   [ 7, 12, 17]])
In [25]: v=np.arange(0,5,0.2)
v
Out[25]: array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4,
   2.6, 2.8, 3. , 3.2, 3.4, 3.6, 3.8, 4. , 4.2, 4.4, 4.6, 4.8])
In [26]: np.linspace(1,2 , num=11)      #Linear space
Out[26]: array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
In [27]: #random number in numpy
r1=np.random.rand(2,3)      #between[0,1)
r1
Out[27]: array([[0.84069328, 0.14171293, 0.01025492],
   [0.5869367 , 0.89171718, 0.771885 ]])
In [28]: r2=np.random.randn(2,3)      #normal distribution with(mu=0 , sigma=1)
r2
Out[28]: array([-0.39048991, 0.66925732, 2.75853643],
   [-0.7640673 , 0.90670924, -2.14909485])
In [29]: import matplotlib.pyplot as plt
r3=np.random.randn(1000000,1)

plt.hist(r3 , bins=50)
plt.show()
```

```
In [30]: r4=r3*2+5
plt.hist(r4 , bins=50)
plt.show()
```

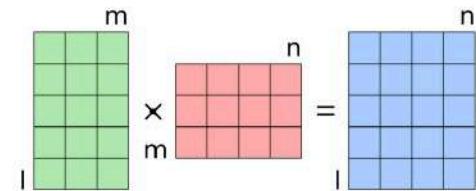


## Linear regression

19

### Multiplication

For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix, known as the **matrix product**, has the number of rows of the first and the number of columns of the second matrix. The product of matrices **A** and **B** is denoted as **AB**.



$$[\mathbf{AB}]_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \cdots + a_{i,n}b_{n,j} = \sum_{r=1}^n a_{i,r}b_{r,j},$$

?

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 7 & 8 \\ 0 & 9 \end{bmatrix} \quad \mathbf{A} \times \mathbf{B}$$

$$\mathbf{B} \times \mathbf{A}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 5 & -2 \\ 1 & 2 & -1 \\ 3 & 6 & -3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 3 & -1 \\ 0 & 3 & -1 \\ 0 & 9 & -3 \end{bmatrix} \quad \mathbf{A} \times \mathbf{B}$$

```
In [31]: a=np.array([[1,2,3],
[4,-5,6]])

b=np.array([[7,8],
[0,9]])

b@a
```

```
Out[31]: array([[ 39, -26,  69],
[ 36, -45,  54]])
```

```
In [32]: import numpy as np
c=np.array([[1,2,-2],
[1,2,-1],
[3,6,-3]])
```

```
d=np.array([[0,3,-1],
[0,3,-1],
[0,9,-3]])
```

```
c@d
```

```
Out[32]: array([[ 0,  0,  0],
[ 0,  0,  0],
[ 0,  0,  0]])
```

```
In [33]: j=np.array([[2,3],
[4,5],
[6,7]])

p=np.array([[2,1], [3,0], [7,5]])
```

```
print(j)
print(".....")
print(p)
```

```

[[2 3]
 [4 5]
 [6 7]]
.....
[[2 1]
 [3 0]
 [7 5]]]

In [34]: j@p
-----
ValueError                                Traceback (most recent call last)
Cell In[34], line 1
----> 1 j@p

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 2)

In [35]: print(j.shape)
print(p.shape)

(3, 2)
(3, 2)

In [36]: k = np.array([[2,3],[4,5],[6,7]])
m = np.array([[3,6,8],[7,0,1]])
#k@m
np.matmul(k,m)
np.dot(k,m)

Out[36]: array([[27, 12, 19],
 [47, 24, 37],
 [67, 36, 55]])

In [37]: A= np.array([[1,5,-2],
 [1,2,-1],
 [3,6,-3]])

B=np.array([[0,3,-1],
 [0,3, -1],
 [0,9,-3]])

A@B

Out[37]: array([[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]])

In [38]: A*B

Out[38]: array([[ 0, 15,  2],
 [ 0,  6,  1],
 [ 0, 54,  9]])

In [39]: A**2

Out[39]: array([[ 1, 25,  4],
 [ 1,  4,  1],
 [ 9, 36,  9]])

In [40]: k= np.array([[2,3],[4,5],[6,7]])
np.log(k)

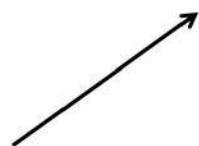
Out[40]: array([[0.69314718, 1.09861229],
 [1.38629436, 1.69943791],
 [1.79175947, 1.94591015]])

In [41]: A= np.array([[1,5,-2],
 [1,2,-1],
 [3,6,-3]])
np.abs(A)

Out[41]: array([[1, 5, 2],
 [1, 2, 1],
 [3, 6, 3]])

```

■ **Vector Representation**



Physics approach  
Length – Direction

$$\vec{v}$$

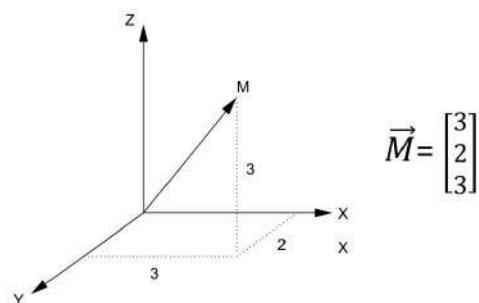
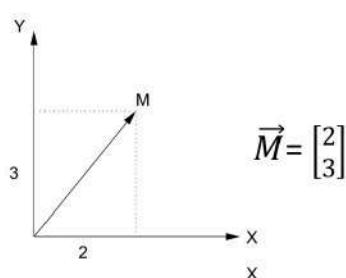
$$\begin{bmatrix} 4 \\ 5 \\ -1 \end{bmatrix}$$

Mathematic approach

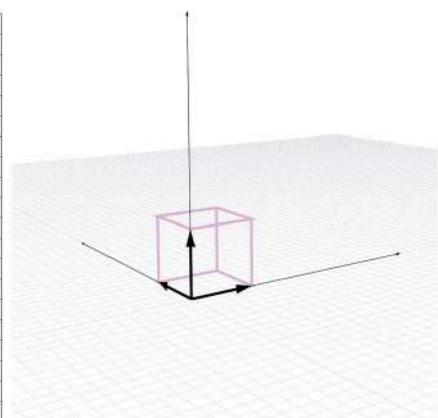
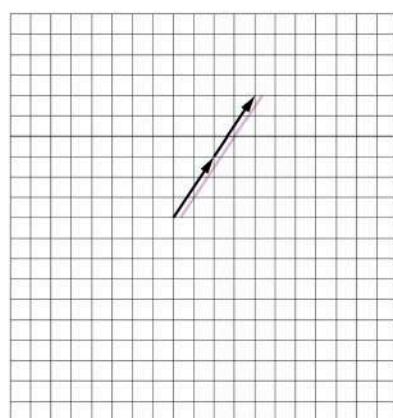
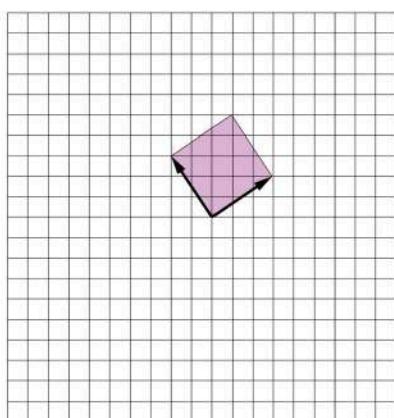
Computer science approach  
List of numbers

**Row vector:** A matrix with one row, sometimes used to represent a vector

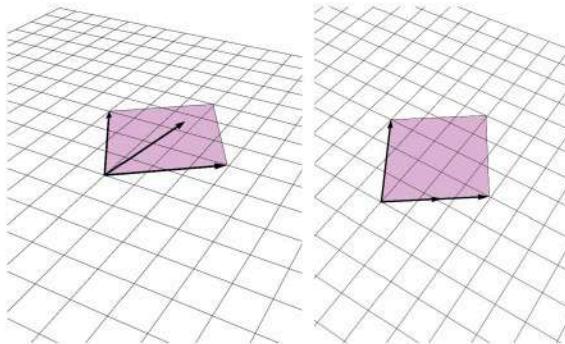
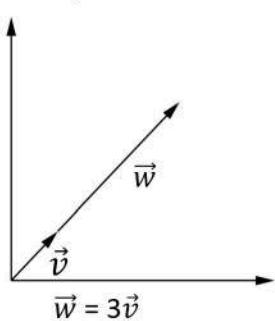
**Column vector:** A matrix with one column, sometimes used to represent a vector



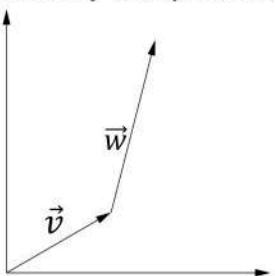
■ **Span:** all the available spaces is been created by vectors.



**Linearly dependent vector** :In the theory of vector spaces, a set of vectors is said to be linearly dependent if there is a nontrivial linear combination of the vectors that equals the zero



**Linearly independent**



### linear transformation

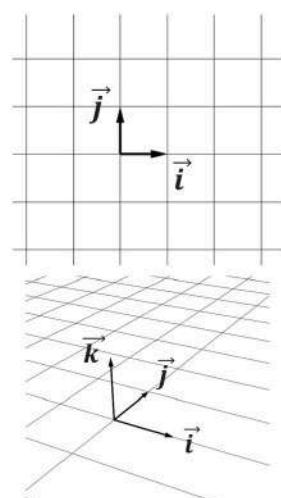
The matrix of a linear transformation is a matrix for which

$$T(\vec{x}) = A \vec{x}, \quad \text{for a vector } \vec{x} \text{ in the domain of } T.$$

This means that applying the transformation  $T$  to a vector is the same as multiplying by this matrix

**The standard unit vectors:** The standard unit vectors in two dimensions,  $i$  and  $j$  are length one vectors that point parallel to the  $x$ -axis and  $y$ -axis, respectively.

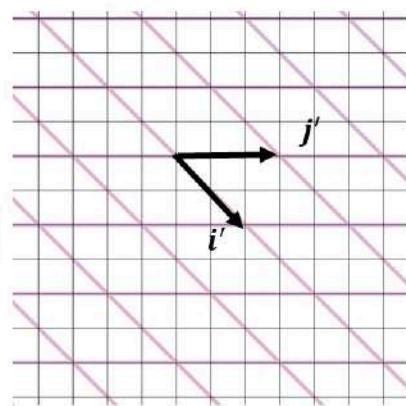
The standard unit vectors in three dimensions,  $i$ ,  $j$ , and  $k$  are length one vectors that point parallel to the  $x$ -axis,  $y$ -axis, and  $z$ -axis respectively.



■ **Changing the standard unit vectors**

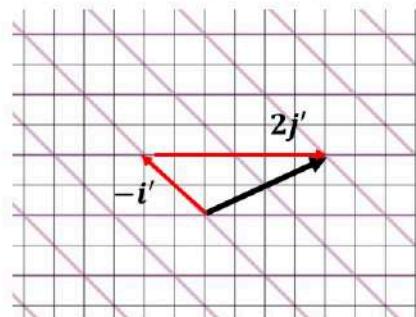
$$j' = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$i' = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$



① Specify  $\vec{v} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$  in new span

$$\begin{bmatrix} 2 & 3 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$



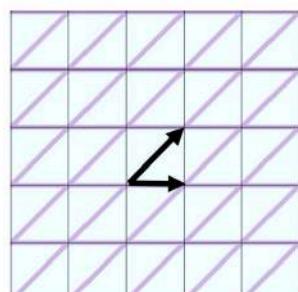
#

■ **Shear matrix:** In mathematics, a shear matrix or transvector is an elementary matrix that represents the addition of a multiple of one row or column to another

$$S = \begin{pmatrix} 1 & 0 & 0 & \lambda & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

① Specify  $\vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$  after shear transformation by  $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

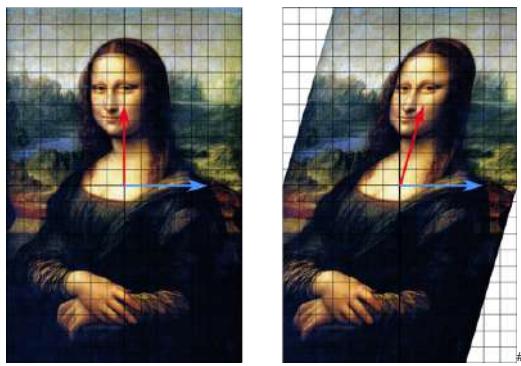
$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$



■ **Identity matrix**

② Specify  $\vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$  after transformation by  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$A \times I = I \times A = A$$



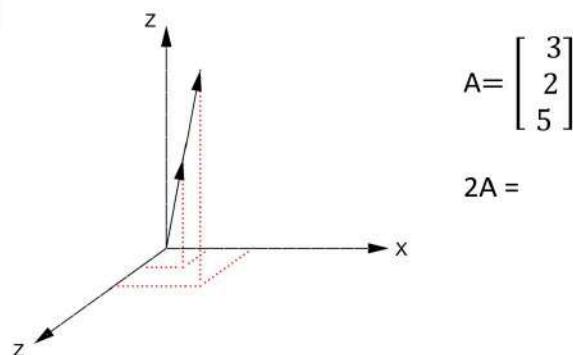
## Linear regression

20

### Scalar multiplication

matrix A is computed by multiplying every entry of A by k:

$$K \cdot A = A \cdot K = K [a_{ij}]_{m \times n} = [ka_{ij}]_{m \times n}$$



### Power of a Matrix

If  $A$  is a square matrix and  $k$  is a positive integer, the  $k$  power of  $A$  is given by  $A = A \times A \times \dots \times A$ , where there are  $k$  copies of matrix  $A$

$$A_{n \times n} \times A_{n \times n} = A^2$$

?

$$A = \begin{bmatrix} 1 & -2 & 2 \\ 0 & 2 & 0 \\ 1 & -1 & -3 \end{bmatrix} \quad A^2$$

$$A \times A^2 = A^3$$

```
In [42]: import numpy as np
In [43]: k= np.array([[2,3], [4,5] , [6,7]])
          np.shape(k) #k.shape
Out[43]: (3, 2)
In [44]: print(k)
2*k
[[2 3]
 [4 5]
 [6 7]]
Out[44]: array([[ 4,  6],
 [ 8, 10],
 [12, 14]])
In [45]: k**2
Out[45]: array([[ 4,  9],
 [16, 25],
 [36, 49]])
In [46]: 1/k
Out[46]: array([[ 0.5      ,  0.33333333],
 [ 0.25     ,  0.2      ],
 [ 0.16666667,  0.14285714]])
In [47]: A_mat=np.array([[1,-2,2],
 [0,2,0],
 [1,-1,-3]])
A_mat@A_mat
Out[47]: array([[ 3, -8, -4],
 [ 0,  4,  0],
 [-2, -1, 11]])
In [48]: A=np.array([[2,3,4]])
B= np.array([[5,6,-7]])
A*B      #np.multiply(j, p)
```

```

Out[48]: array([[ 10,  18, -28]])

In [49]: A_mat=np.array([[1,-2,2],
                      [0,2,0],
                      [1,-1,-3]])

In [50]: np.sum(A_mat)
Out[50]: 0

In [51]: np.sum(A_mat , axis=0)
Out[51]: array([ 2, -1, -1])

In [52]: B_mat=np.array([1.2,2.2,3.6])

In [53]: np.floor(B_mat)
Out[53]: array([1., 2., 3.])

In [54]: np.ceil(B_mat)
Out[54]: array([2., 3., 4.])

In [55]: np.prod(B_mat)
Out[55]: 9.504000000000001

In [56]: #broadcasting
import numpy as np

In [57]: d=np.array([[3,4,5],
                  [7,10,15],
                  [0,5,4]])

In [58]: 3*d
Out[58]: array([[ 6,  7,  8],
               [10, 13, 18],
               [ 3,  8,  7]])

In [59]: a=np.ones((3,3))
b=np.array([[4,2,7]])

In [60]: a+b
Out[60]: array([[5.,  3.,  8.],
               [5.,  3.,  8.],
               [5.,  3.,  8.]})

In [61]: s=np.array([[2,3],
                  [4,5]])
L=np.ones_like(s)      #np.ones((2,2))
L
Out[61]: array([[1, 1],
               [1, 1]])

In [62]: np.zeros_like(s)
Out[62]: array([[0, 0],
               [0, 0]])

In [63]: # Colon :
In [64]: import numpy as np
A=np.array([[8,1,6],           #Magic square
            [3,5,7],
            [4,9,2]]))

In [65]: A[2]
Out[65]: array([4, 9, 2])

In [66]: A[2,1]    #A [2][1]
Out[66]: 9

In [67]: A[A.flatten()]
A[[1,4,5,7]]
Out[67]: array([1, 5, 7, 9])

In [68]: A=np.array([[8,1,6],
                  [3,5,7],
                  [4,9,2]]))

A[:, 2]
Out[68]: array([6, 7, 2])

In [69]: A[0:2 ,0 :2 ]
Out[69]: array([[8, 1],
               [3, 5]])

In [70]: # CONCATINATE
In [71]: A=np.array([[8,1,6],           #Magic square
                  [3,5,7],
                  [4,9,2]]))

v=np.array([[9] , [7], [0]])
v=np.transpose(v)

In [72]: np.concatenate((A , v) , axis = 0)    #hstack and vstack == concatenate   np.stack(a, b): 3D matrix
Out[72]: array([[8, 1, 6],
               [3, 5, 7],
               [4, 9, 2],
               [9, 7, 0]])

In [73]: np.vstack((A, v))
Out[73]: array([[8, 1, 6],
               [3, 5, 7],
               [4, 9, 2],
               [9, 7, 0]])

```

```
In [74]: A=np.array([[8,1,6],           #Magic square
[3,5,7],
[4,9,2]])
p=np.array([1,4], [2], [3]))
np.concatenate((A, p) , axis=1)

Out[74]: array([[ 8,  1,  6,  4],
[ 3,  5,  7, -2],
[ 4,  9,  2,  3]])

In [75]: np.hstack((A, p))

Out[75]: array([[ 8,  1,  6,  4],
[ 3,  5,  7, -2],
[ 4,  9,  2,  3]])

In [76]: # FLATTEN

In [77]: #function in nural networks
B=np.array([[1,2],
[3,4],
[5,6]])

print(B)
B.shape

[[1 2]
 [3 4]
 [5 6]]
(3, 2)

Out[77]: (3, 2)

In [78]: B.flatten()

Out[78]: array([1, 2, 3, 4, 5, 6])

In [79]: C=B.ravel(order = "c")      #c Language
D=B.ravel(order="F")            #fortran Language
print(C)
print(D)

print(np.shape(C))

[1 2 3 4 5 6]
[1 3 5 2 4 6]
(6,)

In [80]: A=np.array([[8,1,6],           #Magic square
[3,5,7],
[4,9,2]]))

In [81]: A.max()      #np.max(A)

Out[81]: 9

In [82]: A.argmax()    # index of maximum number

Out[82]: 7

In [83]: A.max(axis=1)

Out[83]: array([8, 7, 9])

In [84]: A>3

Out[84]: array([[ True, False,  True],
[False,  True,  True],
[ True,  True, False]])

In [85]: A=A.ravel()

In [86]: np.where(A>3)      #shows the index

Out[86]: (array([0, 2, 4, 5, 6, 7], dtype=int64),)

In [87]: A[np.where(A>3)]

Out[87]: array([8, 6, 5, 7, 4, 9])

In [88]: A=np.array([[8,1,6],
[3,5,7],
[4,9,2]])           #shows the rows and columns
np.where(A>3)

Out[88]: (array([0, 0, 1, 1, 2, 2], dtype=int64),
array([0, 2, 1, 2, 0, 1], dtype=int64))

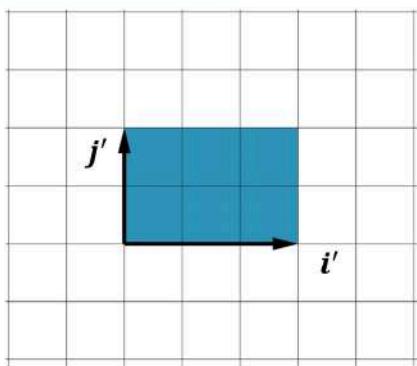
In [89]: r, c = np.where(A>3)

In [90]: A[r,c]

Out[90]: array([8, 6, 5, 7, 4, 9])
```

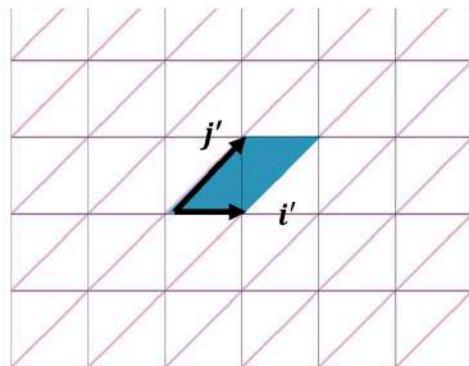
- **Determinant:** a scalar value that is a function of the entries of a square matrix.

Area between  $i, j$



$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

$$|A| = 6$$



$$B = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$|B| = 1$$

- **Laplace expansion**

$$|B| \text{ or } \det(B) = \sum_{j=1}^n (-1)^{i+j} B_{i,j} M_{i,j},$$

Where  $B_{i,j}$  is the entry of the  $i$ th row and  $j$ th column of  $B$ , and  $M_{i,j}$  is the determinant of the submatrix obtained by removing the  $i$ th row and the  $j$ th column of  $B$ . The term  $(-1)^{i+j} M_{i,j}$  is called the **cofactor** of  $B_{i,j}$  in  $B$

- **Cofactor**

$$\sum_{k=1}^n (-1)^{i+k} M_{ik} = \sum_{k=1}^n (-1)^{k+j} M_{kj}$$

In linear algebra, a minor of a matrix  $A$  is the determinant of some smaller square matrix, cut down from  $A$  by removing one or more of its rows and columns. Minors obtained by removing just one row and one column from square matrices (first minors) are required for calculating matrix cofactors, which in turn are useful for computing both the determinant and inverse of square matrices

$$A = \begin{bmatrix} 2 & 3 \\ 5 & 4 \end{bmatrix}$$

$$|A| = (2 \times 4) - (3 * 5)$$

$$B = \begin{bmatrix} 2 & 3 & 5 \\ 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

$$|B| = 2 \begin{vmatrix} 0 & 0 \\ 1 & 0 \end{vmatrix} - 3 \begin{vmatrix} 1 & 0 \\ 2 & 0 \end{vmatrix} + 5 \begin{vmatrix} 1 & 0 \\ 2 & 1 \end{vmatrix} = 2(0-0) - 3(0-0) + 5(1-0) = 5$$

## Properties of Determinants

### 1. All-zero Property

If all the elements of a row (or column) are zero, then the determinant is zero.

```
In [91]: B= np.array([[2,3,5],
                  [1,0,0],
                  [2,1,0]])
# np.linalg
print(round(np.linalg.det(B)))
```

5

## 2 .Reflection Property

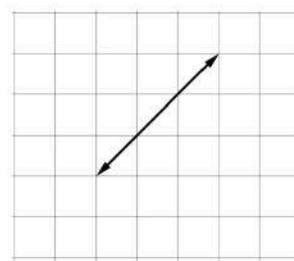
The determinant remains unaltered if its rows are changed into columns and the columns into rows. This is known as the property of reflection

$$|A| = |A'|$$

## 3. Proportionality (Repetition) Property

If all elements of a row (or column) are proportional (identical) to the elements of some other row (or column), then the determinant is zero.

$$|A| = \begin{vmatrix} a & ka & d \\ b & kb & e \\ c & kc & f \end{vmatrix} = 0$$



$$A = \begin{bmatrix} 2 & -1 \\ 2 & -1 \end{bmatrix}$$

$$|A| = 0$$

$$\text{area}=0$$

### ■ Matrix Inverse

The inverse of a square matrix A, sometimes called a reciprocal matrix, is a matrix  $A^{-1}$  ( $\check{A}$ ) such that:

$$A \cdot A^{-1} = A^{-1} \cdot A = I_n$$

A square matrix A has an inverse iff the determinant  $|A| \neq 0$

### ■ Calculate matrix inverse

#### Adjoint of a matrix

$$\text{adj}A = [(-1)^{j+i} |A_{ji}|]$$

$$A^{-1} = \frac{1}{|A|} \cdot \text{adj} A$$

$$A = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \quad A' = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \quad \text{adj}A = \begin{bmatrix} 1 & 4 & -2 \\ -2 & -5 & 4 \\ 1 & -2 & 1 \end{bmatrix} \quad |A| = 3 \quad A^{-1} = \begin{bmatrix} 1/3 & 4/3 & -2/3 \\ -2/3 & -5/3 & 4/3 \\ 1/3 & -2/3 & 1/3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 3 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 4 \end{bmatrix} \quad B^{-1} = \begin{bmatrix} 7 & -3 & -3 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

```
In [92]: import numpy as np
A= np.array([[1,0,2],
[2,1,0],
[3,2,1]])

display(np.around(np.linalg.inv(A), decimals=1))
```

```

array([[ 0.3,  1.3, -0.7],
       [-0.7, -1.7,  1.3],
       [ 0.3, -0.7,  0.3]])

In [93]: B=np.array([[1,3,3],
                  [1,4,3],
                  [1,3,4]])

print(np.linalg.det(B))
print(np.around(np.linalg.inv(B)))

1.0
[[ 7. -3. -3.]
 [-1.  1.  0.]
 [-1.  0.  1.]]

In [94]: D= np.array([[2,3,6],
                  [5,6,9]])

np.linalg.inv(D)

-----
LinAlgError                                 Traceback (most recent call last)
Cell In[94], line 4
      1 D= np.array([[2,3,6],
      2           [5,6,9]])
--> 4 np.linalg.inv(D)

File <__array_function__ internals>:180, in inv(*args, **kwargs)

File F:\Users\nikoo\anaconda3\lib\site-packages\numpy\linalg\linalg.py:547, in inv(a)
    545 a, wrap = _makearray(a)
    546 assert_stacked_2d(a)
--> 547 _assert_stacked_square(a)
    548 t, result_t = _commontype(a)
    550 signature = 'D->D' if isComplexType(t) else 'd->d'

File F:\Users\nikoo\anaconda3\lib\site-packages\numpy\linalg\linalg.py:204, in _assert_stacked_square(*arrays)
    202 m, n = a.shape[-2:]
    203 if m != n:
--> 204     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square

In [95]: dataset= np.array([[2104, 5, 1, 45],
                         [1416, 3, 2, 40],
                         [1532, 3, 2, 30],
                         [852, 2, 1, 36]])

display(np.linalg.det(dataset))

display(np.linalg.inv(dataset))

6211.999999999945
array([[-1.60978751e-03, -1.88345138e-02,  1.40051513e-02,
       1.12685126e-02],
      [ 1.06117193e+00,  8.21571153e+00, -6.03219575e+00,
       -5.42828348e+00],
      [-7.85576304e-02,  3.88087572e+00, -2.11654862e+00,
       -2.45009659e+00],
      [-1.86735351e-02, -1.18480361e-01,  6.24597553e-02,
       1.30714746e-01]]])

```

## Linear regression

28

### Linear regression equation

$$\theta_0x_0 + \theta_1x_1 + \theta_2x_2 + \dots + \theta_nx_n = y$$

$$4x_0 + 6x_1 + 5x_2 + 4x_3 = 10$$

$$\theta_{n \times n} \cdot X_{n \times 1} = y_{n \times 1}$$

$$|\theta| \neq 0$$

$$(\theta^{-1} \cdot \theta) \cdot X = \theta^{-1} \cdot Y$$

$$I_n \cdot X = \theta^{-1} \cdot Y$$

$$X = \theta^{-1} \cdot Y$$

?

$$\begin{cases} 2x_0 + 3x_1 + x_2 = 7 \\ x_0 + 2x_1 + 3x_2 = 9 \\ 3x_0 + x_1 + 2x_2 = 8 \end{cases}$$

$$\begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 9 \\ 8 \end{bmatrix}$$

$$|\theta| = 18$$

$$X = \theta^{-1} \cdot Y$$

$$\theta^{-1} = \frac{1}{18} \begin{bmatrix} 1 & -5 & 7 \\ 7 & 1 & -5 \\ -5 & 7 & 1 \end{bmatrix}$$

$$X = \frac{1}{18} \begin{bmatrix} 1 & -5 & 7 \\ 7 & 1 & -5 \\ -5 & 7 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 9 \\ 8 \end{bmatrix} = \frac{1}{18} \begin{bmatrix} 18 \\ 18 \\ 36 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

### Normal equation

In linear regression analysis, the normal equations are a system of equations whose solution is the Ordinary Least Squares (OLS) estimator of the regression coefficients

	Size (feet <sup>2</sup> )	#Bedrooms	#Floors	Age (years)	Price(1000\$)
$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$y$
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
	...	...	...	...	...

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$X\theta = y \quad X^{-1}X\theta = X^{-1}y \quad I\theta = X^{-1}y \quad \theta = X^{-1}y$$

X matrix is often noninvertible, so we rarely use this method

$$X\theta = y \quad X^T X \theta = X^T y \quad \text{معادله نرمال} \quad A\theta = B \quad A^{-1}A\theta = A^{-1}B \quad \theta = A^{-1}B$$

$$\theta = (X^T X)^{-1} X^T y$$

$$\theta = X^+ y \quad \begin{array}{l} \text{شبہ وارون} \\ \text{Pseudo inverse} \end{array}$$

Sometimes  $X^T X$  is not invertible

Reason:

1. Redundancy ( linearly dependent)
2. Having a lot of features.

Solution : removing some features

```
In [96]: #numpy.linalg.det
```

```
R= np.array([[8,1,6],
[3,5,7],
[4,9,2]])
```

```

In [96]: np.linalg.pinv(R)
Out[96]: array([[ 0.14722222, -0.14444444,  0.06388889],
   [-0.06111111,  0.02222222,  0.10555556],
   [-0.01944444,  0.18888889, -0.10277778]])

In [97]: v=[[2,3,4],
      [5,9,12],
      [4,6,8]]
      print(np.linalg.pinv(v))
      p=np.linalg.pinv(X)@y
[[ 0.6 -1.  1.2 ]
 [-0.12  0.24 -0.24]
 [-0.16  0.32 -0.32]]

LinAlgError: Traceback (most recent call last)
Cell In[97], line 7
  1 v=[[2,3,4],
  2   [5,9,12],
  3   [4,6,8]]
  4 print(np.linalg.pinv(v))
----> 7 p=np.linalg.pinv(X)@y

File <__array_function__ internals>:180, in pinv(*args, **kwargs)

File F:\Users\nikoo\anaconda3\lib\site-packages\numpy\linalg\linalg.py:1998, in pinv(a, rcond, hermitian)
1996     return wrap(res)
1997 a = a.conjugate()
-> 1998 u, s, vt = svd(a, full_matrices=False, hermitian=hermitian)
2000 # discard small singular values
2001 cutoff = rcond..., newaxis] * amax(s, axis=-1, keepdims=True)

File <__array_function__ internals>:180, in svd(*args, **kwargs)

File F:\Users\nikoo\anaconda3\lib\site-packages\numpy\linalg\linalg.py:1638, in svd(a, full_matrices, compute_uv, hermitian)
1635     s = abs(s)
1636     return sort(s)[..., ::-1]
-> 1638 _assert_stacked_2d(a)
1639 t, result_t = _commonType(a)
1641 extobj = get_linalg_error_extobj(_raise_linalgerror_svd_nonconvergence)

File F:\Users\nikoo\anaconda3\lib\site-packages\numpy\linalg\linalg.py:197, in _assert_stacked_2d(*arrays)
195 for a in arrays:
196     if a.ndim < 2:
-> 197         raise LinAlgError('%d-dimensional array given. Array must be '
198                         'at least two-dimensional'

LinAlgError: 1-dimensional array given. Array must be at least two-dimensional

In [98]: R=np.array([8,1,6])
        np.mean(R)
        np.std(R)      #R.std()
        np.var(R)
Out[98]: 8.666666666666666

In [99]: #import Libraries
        import numpy as np
        import matplotlib.pyplot as plt

In [100... X=np.array([[1,1], [1,2], [1,3], [1,4], [1,5]])
        y=np.array([[3],[4],[2],[4],[5]])
        y.shape
Out[100]: (5, 1)

In [101... # theta= inv(X) @
        theta= np.linalg.pinv(X)@y
        print(theta.shape)
        print(f" the slope is {theta[1]} and the intercept is {theta[0]}")
(2, 1)
        the slope is [0.4] and the intercept is [2.4]

In [102... def plot_hyp (X, y , theta):
        #Plot data
        plt.scatter(X[:, 1], y , s=25 , c="r", marker="o")
        h=X@theta
        plt.plot(X[:, 1], h , c="b", lw=2)
        plt.show()

plot_hyp (X, y , theta)


In [103... #cost function
def mse(X, y , theta):
    """ a vectorized implementation of sum of squared error"""
    predictions= X@theta
    errors= predictions - y
    return 0.5 * np.sum(errors**2)
a=mse(X , y , theta)
print(f" the cost function is: {a:.2f}")

the cost function is: 1.8

In [104... #calculating y_hat ans y bar
y_bar= sum(y / len(y))

```

```

y_hat= []
for i in range(len(X)):
    y_hat.append(X[i][1]*theta[1]+ theta[0])
print(y_hat)

[array([2.8]), array([3.2]), array([3.6]), array([4.]), array([4.4])]

```

```

In [105...]:
#rse
def rse(y , y_hat , y_bar):
    enumerator=0
    denominator=0
    for i in range(len(y)):
        enumerator+= (y[i]-y_hat[i])**2
        denominator+= (y[i]-y_bar)**2
    return enumerator/denominator
a=rse(y , y_hat , y_bar)
a

```

```
Out[105]: array([0.69230769])
```

```

In [106...]:
#cost function diagram
thetas= np.linspace(-2 , 3 , num=50)
J=[]
for z in thetas:
    cost=mse(X , y , np.array([[2.4],[z]]))      #intercept is constant to dipict 2d diagram
    J+=[cost]

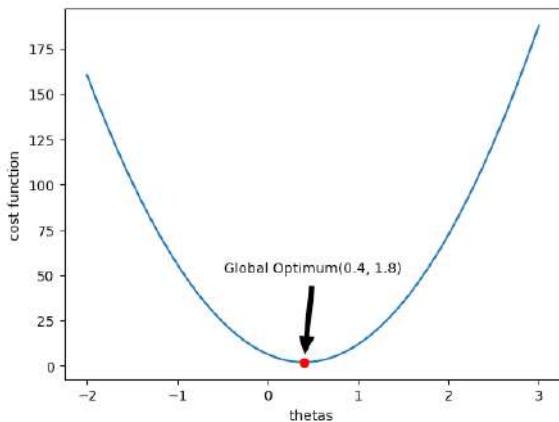
```

```

In [107...]:
plt.plot(thetas , J)
plt.plot(0.4,1.8 , "ro")
plt.xlabel("thetas ")
plt.ylabel("cost function")
plt.annotate("Global Optimum(0.4, 1.8)" , xy=(0.4, 1.8) , xytext=(0.5,50) ,
            arrowprops= dict(width =3, headwidth=10,facecolor="black", shrink=0.1),
            horizontalalignment="center",verticalalignment="bottom")

```

```
Out[107]: Text(0.5, 50, 'Global Optimum(0.4, 1.8)')
```



## Example

Data Set Information: We perform energy analysis using 12 different building shapes simulated in Ecotect. The buildings differ with respect to the glazing area, the glazing area distribution, and the orientation, amongst other parameters. We simulate various settings as functions of the afore-mentioned characteristics to obtain 768 building shapes. The dataset comprises 768 samples and 8 features, aiming to predict two real valued responses. It can also be used as a multi-class classification problem if the response is rounded to the nearest integer. Specifically: X1 Relative Compactness # - نرخ تراکم X2 Surface Area # مساحت سطح X3 Wall Area # مساحت دیوار X4 Roof Area # مساحت گلوبال X5 Overall Height # ارتفاع کلی X6 Orientation # ناحیه شفاف X7 Glazing Area # نوچ گلوبال X8 Glazing Area Distribution # نوزیر منطقه شفاف y1 Heating Load # بارگذاری y2 Cooling Load # مساحت سقف

مساحت دیوار =  $x_2$   
مساحت سقف =  $x_8$

وجود عایق =  $x_6$  کمینه دمای متوسط در 10 سال =  $x_5$  ناحیه شفاف =  $x_4$  ارتفاع کلی

```

In [108...]:
import numpy as np
import pandas as pd
data=pd.read_csv(r"D:\AI_Machinlearning\datasets\ENERGY\energy.csv")
data.tail(10)

```

```

Out[108]:
   X0   X1   X2   X3   X4   X5   X6   X7   X8     Y
758  1  0.66  759.5  318.5  220.5  3.5   4  0.4   5  14.92
759  1  0.66  759.5  318.5  220.5  3.5   5  0.4   5  15.16
760  1  0.64  784.0  343.0  220.5  3.5   2  0.4   5  17.69
761  1  0.64  784.0  343.0  220.5  3.5   3  0.4   5  18.19
762  1  0.64  784.0  343.0  220.5  3.5   4  0.4   5  18.16
763  1  0.64  784.0  343.0  220.5  3.5   5  0.4   5  17.88
764  1  0.62  808.5  367.5  220.5  3.5   2  0.4   5  16.54
765  1  0.62  808.5  367.5  220.5  3.5   3  0.4   5  16.44
766  1  0.62  808.5  367.5  220.5  3.5   4  0.4   5  16.48
767  1  0.62  808.5  367.5  220.5  3.5   5  0.4   5  16.64

```

```
In [109...]:
data.describe()
```

	X0	X1	X2	X3	X4	X5	X6	X7	X8	Y
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	1.0	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000	0.234375	2.81250	22.307201
std	0.0	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763	0.133221	1.55096	10.090196
min	1.0	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000	0.000000	0.00000	6.010000
25%	1.0	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000	0.100000	1.75000	12.992500
50%	1.0	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000	0.250000	3.00000	18.950000
75%	1.0	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000	0.400000	4.00000	31.667500
max	1.0	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000	0.400000	5.00000	43.100000

```

In [110...]:
X=np.array(data.drop(["Y"] , axis= 1))
y=np.array(data["Y"])
y=y.reshape((768,1))

```

```
In [111...  
print(X.shape)  
print(y.shape)  
(768, 9)  
(768, 1)  
In [112... theta= np.linalg.pinv(X)@y  
theta  
Out[112]: array([[ 8.40145212e+01],  
[-6.47739915e+01],  
[-6.26063386e-02],  
[ 3.61294044e-02],  
[-4.93678715e-02],  
[ 4.16993881e+00],  
[-2.33281250e-02],  
[ 1.99326802e+01],  
[ 2.03771772e-01]])  
In [113... y_hat= X@theta  
y_hat
```

```
Out[113]: array([[22.64720083],  
 [22.6238727 ],  
 [22.60054458],  
 [22.57721645],  
 [25.04182354],  
 [25.01849541],  
 [24.99516729],  
 [24.97183916],  
 [24.00424464],  
 [23.98091651],  
 [23.95758839],  
 [23.93426026],  
 [25.94651941],  
 [25.92319129],  
 [25.89986316],  
 [25.87653504],  
 [27.24105427],  
 [27.21772614],  
 [27.19439882],  
 [27.17106989],  
 [31.51544279],  
 [31.49211467],  
 [31.46878654],  
 [31.44545842],  
 [ 5.64803721],  
 [ 5.62470999],  
 [ 5.60138096],  
 [ 5.57805284],  
 [ 6.94257287],  
 [ 6.91924394],  
 [ 6.89591582],  
 [ 6.87258769],  
 [ 7.58936781],  
 [ 7.56603889],  
 [ 7.54271076],  
 [ 7.51938264],  
 [ 8.88390187],  
 [ 8.86057374],  
 [ 8.83724562],  
 [ 8.81391749],  
 [ 9.53869681],  
 [ 9.50736868],  
 [ 9.48404056],  
 [ 9.46071243],  
 [10.17749175],  
 [10.15416363],  
 [10.1388355 ],  
 [10.10750738],  
 [24.84424062],  
 [24.82091249],  
 [24.79758437],  
 [24.77425624],  
 [27.23886333],  
 [27.2155352 ],  
 [27.19220708],  
 [27.16887895],  
 [26.20128443],  
 [26.1779563 ],  
 [26.15462818],  
 [26.13130005],  
 [28.1435592 ],  
 [28.12023188],  
 [28.09690295],  
 [28.07357483],  
 [29.43809486],  
 [29.41476593],  
 [29.39143781],  
 [29.36818968],  
 [33.71248258],  
 [33.68915446],  
 [33.66582633],  
 [33.64249821],  
 [ 7.845077 ,  
 [ 7.82174888],  
 [ 7.79842075],  
 [ 7.77509263],  
 [ 9.13961186],  
 [ 9.11628373],  
 [ 9.09295561],  
 [ 9.06962748],  
 [ 9.7864068 ],  
 [ 9.73975055],  
 [ 9.71642243],  
 [11.08894166],  
 [11.05761353],  
 [11.03428541],  
 [11.01095728],  
 [11.7277366 ],  
 [11.70440847],  
 [11.68108035],  
 [11.65775222],  
 [12.37453154],  
 [12.35120342],  
 [12.32787529],  
 [12.30454717],  
 [25.04801239],  
 [25.02468426],  
 [25.00135614],  
 [24.97802801],  
 [27.4426351 ],  
 [27.41930697],  
 [27.39597885],  
 [27.37265072],  
 [26.4050562 ],  
 [26.38172808],  
 [26.35839995],  
 [26.33507183],  
 [28.34733097],  
 [28.32400285],  
 [28.30067472],  
 [28.2773466 ],  
 [29.64186583],  
 [29.6185377 ],  
 [29.59520958],  
 [29.57188145],  
 [33.91625435],  
 [33.89292623],  
 [33.8695981 ],  
 [33.84626998],  
 [ 8.04884877],  
 [ 8.02552065],  
 [ 8.00219252],  
 [ 7.9788644 ],  
 [ 9.34338363],  
 [ 9.32005551],  
 [ 9.29672738],  
 [ 9.27339926],
```

[ 9.99017857 ],  
[ 9.96685045 ],  
[ 9.94352232 ],  
[ 9.9201942 ],  
[ 11.28471343 ],  
[ 11.2613853 ],  
[ 11.23805718 ],  
[ 11.21472965 ],  
[ 11.93150837 ],  
[ 11.90818025 ],  
[ 11.88485212 ],  
[ 11.861524 ],  
[ 12.57830331 ],  
[ 12.55497519 ],  
[ 12.53164706 ],  
[ 12.50831894 ],  
[ 25.25178416 ],  
[ 25.22845603 ],  
[ 25.20512791 ],  
[ 25.18179978 ],  
[ 27.64648687 ],  
[ 27.62307874 ],  
[ 27.59975062 ],  
[ 27.57642249 ],  
[ 26.60882797 ],  
[ 26.58549985 ],  
[ 26.56217172 ],  
[ 26.5388436 ],  
[ 28.55110274 ],  
[ 28.52777462 ],  
[ 28.50444649 ],  
[ 28.48111837 ],  
[ 29.8456376 ],  
[ 29.82230948 ],  
[ 29.79898135 ],  
[ 29.77565323 ],  
[ 34.12002613 ],  
[ 34.096698 ],  
[ 34.07336988 ],  
[ 34.05004175 ],  
[ 8.25262055 ],  
[ 8.22929242 ],  
[ 8.2059643 ],  
[ 8.18263617 ],  
[ 9.5471554 ],  
[ 9.52382728 ],  
[ 9.50049915 ],  
[ 9.47717103 ],  
[ 10.19395034 ],  
[ 10.17062222 ],  
[ 10.14729409 ],  
[ 10.12396597 ],  
[ 11.4884852 ],  
[ 11.46515708 ],  
[ 11.44182895 ],  
[ 11.41850083 ],  
[ 12.13528014 ],  
[ 12.11195202 ],  
[ 12.08862389 ],  
[ 12.06529577 ],  
[ 12.78207509 ],  
[ 12.75874696 ],  
[ 12.73541884 ],  
[ 12.71209071 ],  
[ 25.45555593 ],  
[ 25.43222781 ],  
[ 25.40889968 ],  
[ 25.38557156 ],  
[ 27.85017864 ],  
[ 27.82685052 ],  
[ 27.80352239 ],  
[ 27.78019427 ],  
[ 26.81259974 ],  
[ 26.78927162 ],  
[ 26.76594349 ],  
[ 26.74261537 ],  
[ 28.75487452 ],  
[ 28.73154639 ],  
[ 28.70821287 ],  
[ 28.68489014 ],  
[ 30.04940937 ],  
[ 30.02608125 ],  
[ 30.00275312 ],  
[ 29.979425 ],  
[ 34.3237979 ],  
[ 34.30046977 ],  
[ 34.27714165 ],  
[ 34.25381352 ],  
[ 8.45639232 ],  
[ 8.43306419 ],  
[ 8.40973607 ],  
[ 8.38640794 ],  
[ 9.75092717 ],  
[ 9.72759905 ],  
[ 9.70427092 ],  
[ 9.6809428 ],  
[ 18.39772212 ],  
[ 18.37439399 ],  
[ 18.35106587 ],  
[ 18.32773774 ],  
[ 11.69225697 ],  
[ 11.66892885 ],  
[ 11.64560072 ],  
[ 11.6222726 ],  
[ 12.33905192 ],  
[ 12.31572379 ],  
[ 12.29239567 ],  
[ 12.26906754 ],  
[ 12.98548686 ],  
[ 12.96251873 ],  
[ 12.93919061 ],  
[ 12.91586248 ],  
[ 25.6593277 ],  
[ 25.635999958 ],  
[ 25.61267145 ],  
[ 25.58934333 ],  
[ 28.05395041 ],  
[ 28.03862229 ],  
[ 28.00729416 ],  
[ 27.98396684 ],  
[ 27.01637152 ],  
[ 26.99304339 ],  
[ 26.96971527 ],  
[ 26.94638714 ],  
[ 28.95864629 ],  
[ 28.93531816 ],  
[ 28.91199004 ],  
[ 28.88866191 ],

[30.25318114],  
[30.22985302],  
[30.20652489],  
[30.18319677],  
[34.52756967],  
[34.50424154],  
[34.48991342],  
[34.45758529],  
[8.66016409],  
[8.63683595],  
[8.61350784],  
[8.59017971],  
[9.95469895],  
[9.93137082],  
[9.9080427],  
[9.88471457],  
[10.60149389],  
[10.57816576],  
[10.55483764],  
[10.53150951],  
[11.89602874],  
[11.87270062],  
[11.84937249],  
[11.82604437],  
[12.54282369],  
[12.51949556],  
[12.49616744],  
[12.47283931],  
[13.18961863],  
[13.1662905],  
[13.14296238],  
[13.11963425],  
[27.83414264],  
[27.81081452],  
[27.78748639],  
[27.76415827],  
[30.22876535],  
[30.20543723],  
[30.1821091],  
[30.15878098],  
[29.19118646],  
[29.16785833],  
[29.14453021],  
[29.12120208],  
[31.13346123],  
[31.11013131],  
[31.08680498],  
[31.06347685],  
[32.42799668],  
[32.40466796],  
[32.38133983],  
[32.35801171],  
[36.70238461],  
[36.67905648],  
[36.655722836],  
[36.63240023],  
[18.83497903],  
[18.8116509],  
[18.78832278],  
[18.76499465],  
[12.12951389],  
[12.10618576],  
[12.08285764],  
[12.05952951],  
[12.77638883],  
[12.7529807],  
[12.72965258],  
[12.70632445],  
[14.07084368],  
[14.04751556],  
[14.02418743],  
[14.00085931],  
[14.71763863],  
[14.6943105],  
[14.67098238],  
[14.64765425],  
[15.36443357],  
[15.34110544],  
[15.31777732],  
[15.29444919],  
[28.03791441],  
[28.01458629],  
[27.99125816],  
[27.96793004],  
[30.43253712],  
[30.409289],  
[30.38588087],  
[30.362555275],  
[29.39495823],  
[29.3716301],  
[29.34830198],  
[29.32497385],  
[31.337233],  
[31.31390487],  
[31.29057675],  
[31.26724862],  
[32.63176786],  
[32.60843973],  
[32.58511161],  
[32.56178348],  
[36.90615638],  
[36.88282826],  
[36.85958013],  
[36.83617201],  
[11.0387508],  
[11.01542268],  
[18.99209455],  
[18.96876643],  
[12.33328566],  
[12.30995753],  
[12.28662941],  
[12.26330128],  
[12.9808086],  
[12.95675247],  
[12.93342435],  
[12.91089622],  
[14.27461546],  
[14.25128733],  
[14.22795921],  
[14.20463108],  
[14.92141104],  
[14.89808227],  
[14.87475415],  
[14.85142602],  
[15.56820534],  
[15.54487722],  
[15.52154909],  
[15.49822097],

[28.24168619],  
[28.21835806],  
[28.19502994],  
[28.17170181],  
[30.6363089 ],  
[30.61298077],  
[30.58965265],  
[30.56632452],  
[29.59873 ],  
[29.57540187],  
[29.55207375],  
[29.52874562],  
[31.54100477],  
[31.51767665],  
[31.49434852],  
[31.4710204 ],  
[32.83553963],  
[32.8122115 ],  
[32.78888338],  
[32.76555525],  
[37.10992815],  
[37.08660003],  
[37.0632719 ],  
[37.03994378],  
[11.24252257],  
[11.21919445],  
[11.19586632],  
[11.17253582 ],  
[12.53705743],  
[12.5137293 ],  
[12.49840118],  
[12.46707305],  
[13.18385237],  
[13.16852425],  
[13.13719612],  
[13.113865 ],  
[14.47838723],  
[14.4550591 ],  
[14.43173098],  
[14.40840285],  
[15.12518217],  
[15.10185405],  
[15.07852592],  
[15.0551978 ],  
[15.77197711],  
[15.74864899],  
[15.72532086],  
[15.70199274],  
[28.44545796],  
[28.42212983],  
[28.39880171],  
[28.37547358],  
[38.84808067],  
[38.81675254],  
[38.79342442],  
[38.77009629],  
[29.80250177],  
[29.77917365],  
[29.75584552],  
[29.7325174 ],  
[31.74477654],  
[31.72144842],  
[31.69812029],  
[31.67479217],  
[33.0393114 ],  
[33.01598328],  
[32.99265515],  
[32.96932703],  
[37.31369992],  
[37.2903718 ],  
[37.26704367],  
[37.24371555],  
[11.44629434],  
[11.42296622],  
[11.39963809],  
[11.37630997],  
[12.7408292 ],  
[12.71750108],  
[12.69417295],  
[12.67884483],  
[13.38762414],  
[13.36429602],  
[13.34096789],  
[13.31763977],  
[14.682159 ],  
[14.65883088],  
[14.63550275],  
[14.61217463],  
[15.32895394],  
[15.30562582],  
[15.28229769],  
[15.25896957],  
[15.97574888],  
[15.95242076],  
[15.92909263],  
[15.90576451],  
[28.64922973],  
[28.62590161],  
[28.60257348],  
[28.57924536],  
[31.04385244],  
[31.02052431],  
[38.99719619],  
[38.97386806],  
[38.00627354],  
[29.98294542],  
[29.95961729],  
[29.93628917],  
[31.94854852],  
[31.92522019],  
[31.90189207],  
[31.87856394],  
[31.24308317],  
[33.21975505],  
[33.19642692],  
[33.1730988 ],  
[37.5174717 ],  
[37.49414357],  
[37.47081545],  
[37.44748732],  
[11.65006612],  
[11.62673799],  
[11.60340987],  
[11.58008174],  
[12.944600097],  
[12.92127285],  
[12.89794472],  
[12.8746166 ],

[13.59139592],  
[13.56806779],  
[13.54473967],  
[13.52141154],  
[14.88593077],  
[14.86260265],  
[14.83927452],  
[14.8159464 ],  
[15.53272571],  
[15.50939759],  
[15.48606946],  
[15.46274134],  
[16.17952066],  
[16.15619253],  
[16.13286441],  
[16.10953628],  
[30.82404467],  
[30.80071654],  
[30.77738842],  
[30.75406029],  
[33.21866738],  
[33.19533925],  
[33.17201113],  
[33.148683 ],  
[32.18108848],  
[32.15776036],  
[32.13443223],  
[32.11110411],  
[34.12336325],  
[34.10003513],  
[34.076707 ],  
[34.05337888],  
[35.41789811],  
[35.39456999],  
[35.37124188],  
[35.34791374],  
[39.69228664],  
[39.66895851],  
[39.64563039],  
[39.62230226],  
[13.82488106],  
[13.80155293],  
[13.77822481],  
[13.75489668],  
[15.11941591],  
[15.09608779],  
[15.07275966],  
[15.04943154],  
[15.76621086],  
[15.74288273],  
[15.71955461],  
[15.69622648],  
[17.06874571],  
[17.03741759],  
[17.01408946],  
[16.99876134],  
[17.70754065],  
[17.68421253],  
[17.6688844 ],  
[17.63755628],  
[18.3543356 ],  
[18.33100747],  
[18.30767935],  
[18.28435122],  
[31.02781644],  
[31.00448832],  
[30.98116019],  
[30.95783207],  
[33.42243915],  
[33.39911103],  
[33.3757829 ],  
[33.35245478],  
[32.38486026],  
[32.36153213],  
[32.33820401],  
[32.31487588],  
[34.32713503],  
[34.3038069 ],  
[34.28847878],  
[34.25715065],  
[35.62166988],  
[35.59834176],  
[35.57501363],  
[35.55168551],  
[39.89605841],  
[39.87273028],  
[39.84940216],  
[39.82607403],  
[14.02865283],  
[14.0053247 ],  
[13.98199658],  
[13.95866845],  
[15.32318768],  
[15.29985956],  
[15.27653143],  
[15.25320331],  
[15.96998263],  
[15.9466545 ],  
[15.92332638],  
[15.89999825],  
[17.26451748],  
[17.24118936],  
[17.21786123],  
[17.19453311],  
[17.91131243],  
[17.8879843 ],  
[17.86465618],  
[17.84132805],  
[18.55810737],  
[18.53477924],  
[18.51145112],  
[18.48812299],  
[31.23158821],  
[31.20826009],  
[31.18493196],  
[31.16160384],  
[33.62621092],  
[33.6028828 ],  
[33.57955467],  
[33.55622655],  
[32.58863203],  
[32.5653039 ],  
[32.54197578],  
[32.51864765],  
[34.5309968 ],  
[34.50757867],  
[34.48425055],  
[34.46892242],

[35.82544166],  
[35.80211353],  
[35.77878541],  
[35.75545728],  
[40.09983018],  
[40.07650206],  
[40.05317393],  
[40.02984581],  
[14.2324246 ],  
[14.20909647],  
[14.18576835],  
[14.16244022],  
[15.52695946],  
[15.50363133],  
[15.48830321],  
[15.45697508],  
[16.1737544 ],  
[16.15042627],  
[16.12709815],  
[16.10377002],  
[17.46828926],  
[17.44496113],  
[17.42163301],  
[17.39830488],  
[18.1150842 ],  
[18.09175607],  
[18.06842795],  
[18.04509982],  
[18.76187914],  
[18.73855101],  
[18.71522289],  
[18.69189476],  
[31.43535999],  
[31.41203186],  
[31.38870374],  
[31.36537561],  
[33.82998269],  
[33.80665457],  
[33.78332644],  
[33.75999832],  
[32.7924038 ],  
[32.76907567],  
[32.745747551],  
[32.72241942],  
[34.73467857],  
[34.71135045],  
[34.68802232],  
[34.6646942 ],  
[36.02921343],  
[36.0058853 ],  
[35.98255718],  
[35.959222905],  
[48.30368195],  
[48.28027383],  
[48.2569457 ],  
[48.23361758],  
[14.43619637],  
[14.41286825],  
[14.38954012],  
[14.366212 ],  
[15.73073123],  
[15.7074031 ],  
[15.68407498],  
[15.66074685],  
[16.37752617],  
[16.35419805],  
[16.33086992],  
[16.3075418 ],  
[17.67206103],  
[17.6487329 ],  
[17.62540478],  
[17.60207665],  
[18.31885597],  
[18.29552784],  
[18.27219972],  
[18.24887159],  
[18.96565091],  
[18.94232279],  
[18.91899466],  
[18.89566654],  
[31.63913176],  
[31.61580363],  
[31.59247551],  
[31.56914738],  
[34.03375447],  
[34.01042634],  
[33.98709822],  
[33.96377009],  
[32.99617557],  
[32.97284745],  
[32.949511932],  
[32.9261912 ],  
[34.93845034],  
[34.91512222],  
[34.89179409],  
[34.86846597],  
[36.2329852 ],  
[36.20965707],  
[36.18632895],  
[36.16300082],  
[40.50737372],  
[40.4840456 ],  
[40.46871747],  
[40.43738935],  
[14.63996814],  
[14.61664002],  
[14.59331189],  
[14.56998377],  
[15.934583 ],  
[15.91117488],  
[15.88784675],  
[15.86451863],  
[16.58129794],  
[16.55796982],  
[16.53464169],  
[16.51131357],  
[17.8758328 ],  
[17.85250467],  
[17.82917655],  
[17.80584842],  
[18.52262774],  
[18.49929962],  
[18.47597149],  
[18.45264337],  
[19.16942268],  
[19.14609456],  
[19.12276643],  
[19.09943831]]])

```
In [114]: x_predict=np.array([[1,0.98,72,90,72,3,2,0.16,1]])
```

```
x_predict@theta
```

```
array([[31.58167359]])
```

```
In [115]: numerator=sum ((y - y_hat)**2)
```

```
Denominator=sum ((y - y.mean())**2)
```

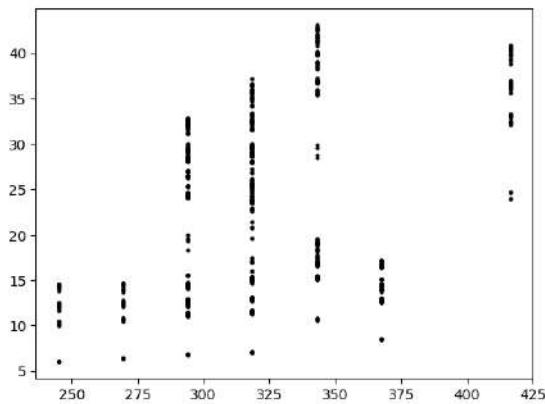
```
rse= numerator / Denominator
```

```
R_squared= 1 - rse
```

```
print(f" the accuracy of the algorhim is: {round(float(R_squared*100), 2)} ")
```

```
the accuracy of the algorhim is: 91.62
```

```
In [116]: import matplotlib.pyplot as plt  
plt.scatter(X[ : , 3] , y , c="black" , s=4)  
plt.show()
```



## Concrete compressive strength

```
In [117]: import pandas as pd  
concrete_data=pd.read_csv(r"D:\AI_Machinlearning\datasets\CONCRETE\Concrete_Data .csv")  
  
display(concrete_data.tail())
```

	X0	X1	X2	X3	X4	X5	X6	X7	X8	y
1025	1	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.28
1026	1	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.18
1027	1	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.70
1028	1	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.77
1029	1	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.40

```
In [118]: concrete_data.describe()
```

```
Out[118]:
```

	X0	X1	X2	X3	X4	X5	X6	X7	X8	y
count	1030.0	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	1.0	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	0.0	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	1.0	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	1.0	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	1.0	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	1.0	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	1.0	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

Number of instances (observations): 1030 Number of Attributes: 9 Attribute breakdown: 8 quantitative input variables, and 1 quantitative output variable Missing Attribute Values: None x1= Cement (component 1) -- quantitative -- kg in a m3 mixture -- Input Variable # سیمان x2=Blast Furnace Slag (component 2) -- quantitative -- kg in a m3 mixture -- Input Variable # سرباره کوره بلند x3=Fly Ash (component 3) -- quantitative -- kg in a m3 mixture -- Input Variable # چاکستر بازی x4=Water (component 4) -- quantitative -- kg in a m3 mixture -- Input Variable # آب x5=Superplasticizer (component 5) -- quantitative -- kg in a m3 mixture -- Input Variable # سنگدانه تیر کوشہ x6=Coarse Aggregate (component 6) -- quantitative -- kg in a m3 mixture -- Input Variable # سنگهای خشک x7=Fine Aggregate (component 7) -- quantitative -- kg in a m3 mixture -- Input Variable # سنگدانه گرد کوشہ x8=Age -- quantitative -- Day (1~365) -- Input Variable # سال y=Concrete compressive strength -- quantitative -- MPa -- Output Variable # مقاومت فشاری بتن

```
In [119]: X= np.array(concrete_data.drop(["y"] , axis= 1))  
y=np.array(concrete_data["y"])
```

```
print(X.shape)
```

```
print(y.shape)
```

```
(1030, 9)
```

```
(1030,)
```

```
In [120]: y=y.reshape(1030,1)
```

```
print(y.shape)
```

```
(1030, 1)
```

```
In [121]: theta= np.linalg.pinv(X)@y
```

```
theta
```

```
Out[121]: array([[-2.33312136e+01],  
[ 1.19804334e-01],  
[ 1.03865809e-01],  
[ 8.79343215e-02],  
[-1.49918419e-01],  
[ 2.92224595e-01],  
[ 1.80862148e-02],  
[ 2.01903511e-02],  
[ 1.14222068e-01]])
```

```
In [122]: y_hat= X@theta
```

```
y_hat
```

```

Out[122]: array([[53.46346329,
   [53.73475651],
   [56.81258594],
   ...,
   [26.46841169],
   [29.12237014],
   [31.89770807]])]

In [123... x_predict=np.array( [[1,470,0,118.4,153,1.8,700,500,28]])
x_predict@theta

Out[123]: array([[46.93047727]])

In [124... numerator=sum ((y - y_hat)**2)
Denominator=sum ((y - y.mean())**2)

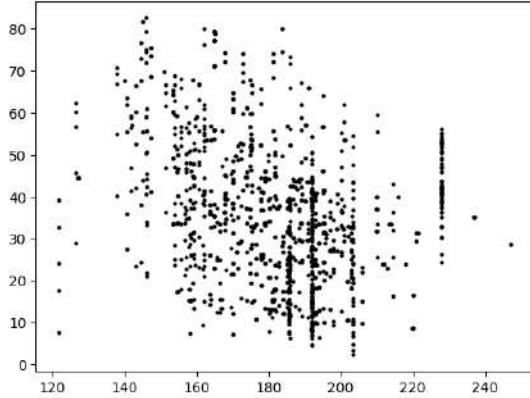
rse= numerator / Denominator

R_squared= 1 - rse

print(f" the accuracy of the algorhim is: {round(float(R_squared*100), 2)} ")
the accuracy of the algorhim is: 61.55

In [125... import matplotlib.pyplot as plt
plt.scatter(X[ :, 4] , y , c="black" , s=4)
plt.show()

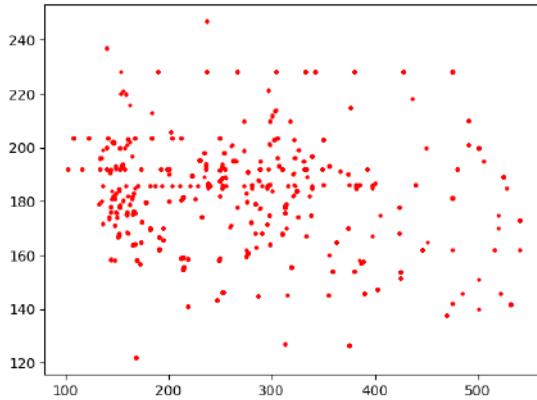
```



```

In [126... plt.scatter(X[ :, 1] , X[ :, 4] , c="red" , s=4)
plt.show()

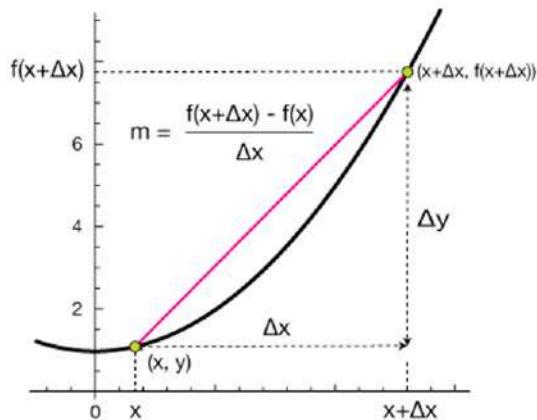
```



### Derivative (differential algebra)

$$Y = f(x)$$

$$y' = f'(x) = \frac{dy}{dx}$$



### Derivative

$f(x)$	$\frac{dy}{dx}$
a	0
5	
$2\sqrt{3}$	
$x^m$	$mx^{m-1}$
$x^3$	
$\theta^2$	

$f(x)$	$\frac{dy}{dx}$
$ax^m$	$amx^{m-1}$
$2x^3$	
$2x^3 + 4x^2 + 3x + 7$	
$u + g$	$u' + g'$
$(x^2 + 5) + (3x^3 + 3x)$	

## Linear regression

### Derivative

$f(x)$	$\frac{dy}{dx}$	$f(x)$	$\frac{dy}{dx}$
$u - g$	$u' - g'$	$\frac{u}{g}$	$\frac{u' \times g - g' \times u}{g^2}$
$(x^2+5) - (3x^3+3x)$		$\frac{(x^2-1)}{(x^5+3x)}$	
$u \times g$	$u' \times g + g' \times u$	$\frac{(3x^2+2)}{(x^3-3x)}$	
$(x^2-1)(x^5+3x)$		$(u)^m$	$m(u)^{m-1} (u')$
$(3x^2+2)(x^3-3x)$		$(2x^2-x)^2$	
		$(3x^3-2x)^6$	

## Linear regression

42

### Partial Derivative

$$z = f(x, y)$$

$$Z(x, y) = x^2 + 5xy^2 + 4y^3$$

$$z'_x = \frac{\partial z}{\partial x}$$

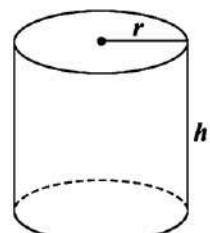
$$U(x, y) = \frac{(x+y)}{(x-y)}$$

$$z'_y = \frac{\partial z}{\partial y}$$

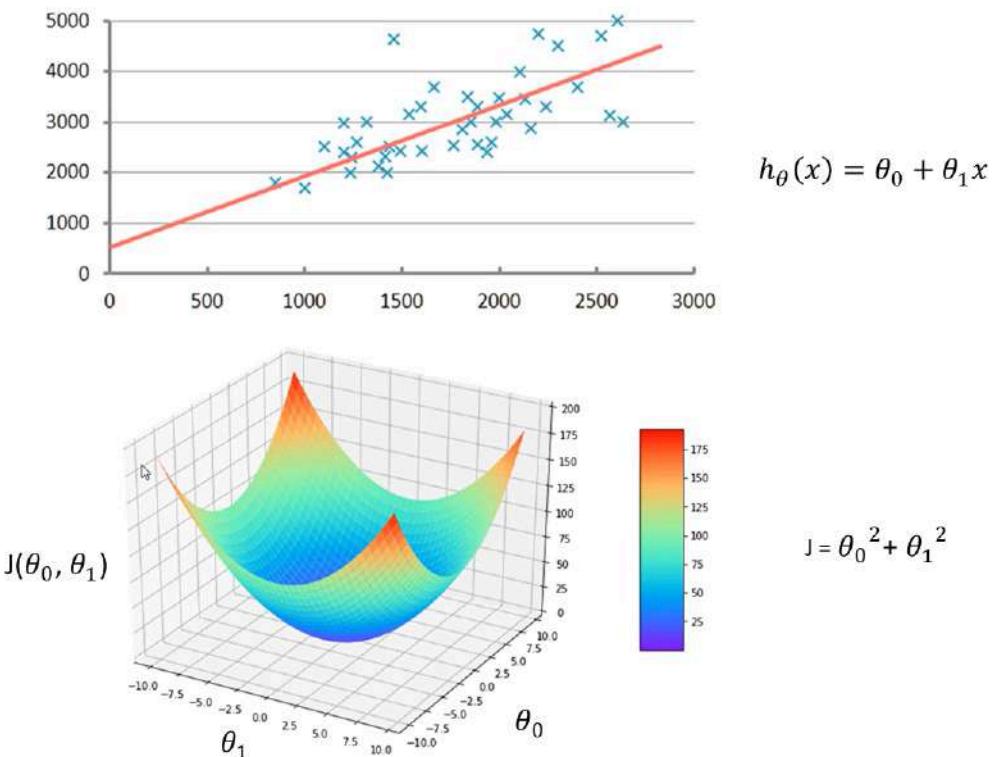
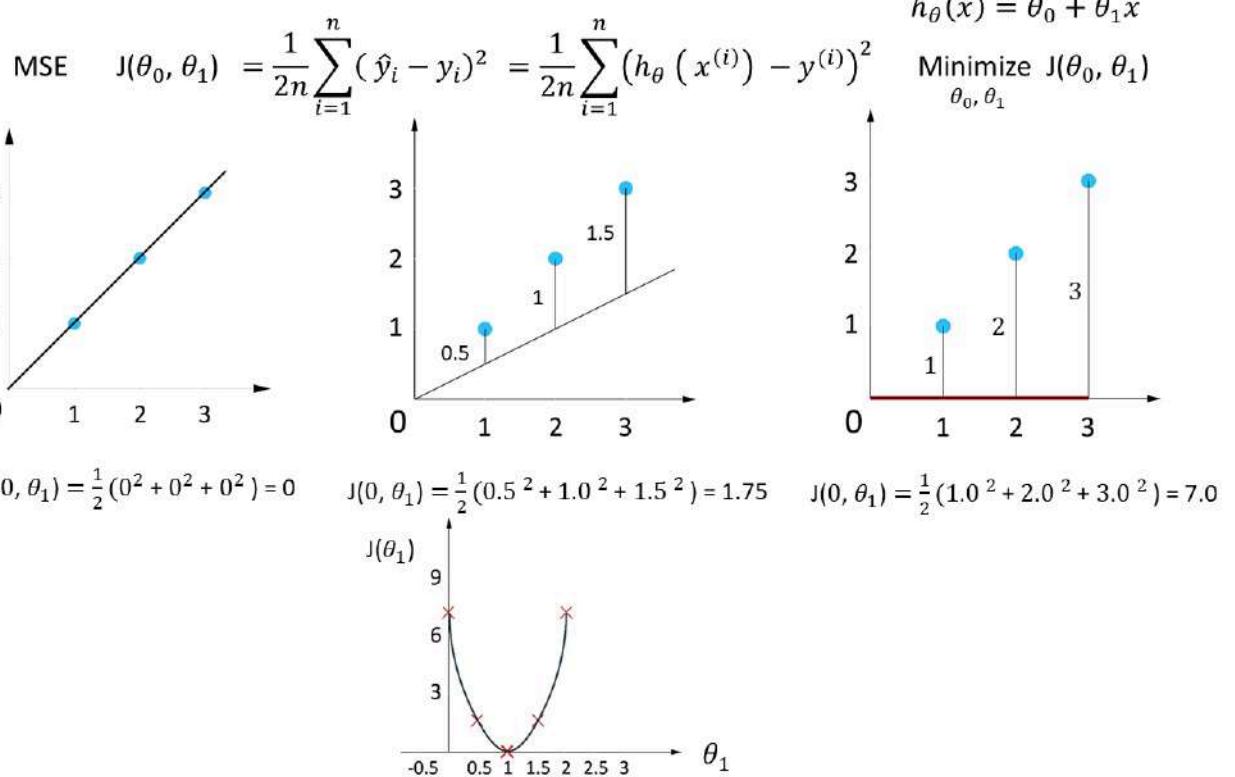
$$F(X, Y) = (2x^2 - y)^2$$

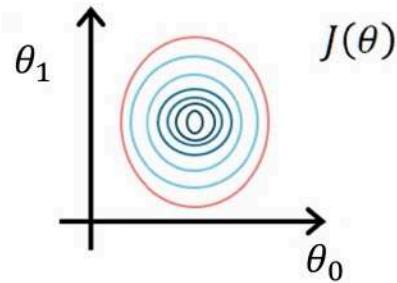
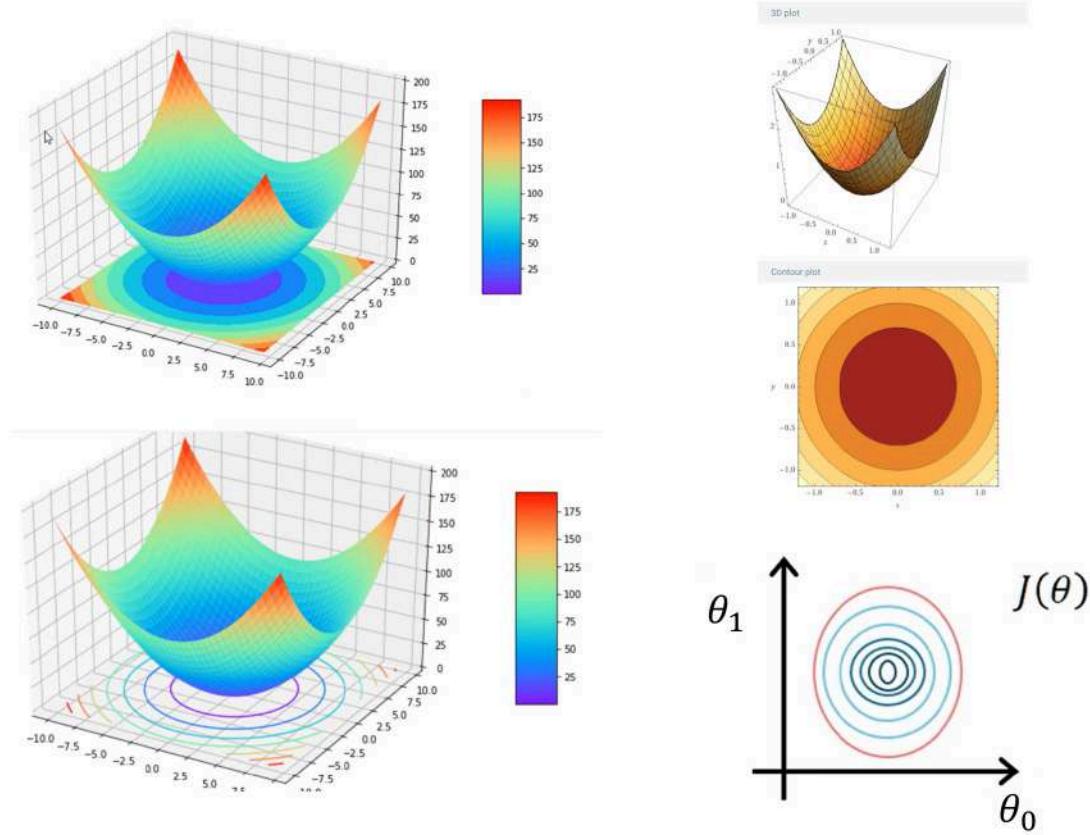
1.1.1

Consider a cylinder of height  $h$  and radius  $r$ . Calculate the volume changes of the cylinder in two cases. In the first case, only the radius of the cylinder is allowed to change, and in the second case, only the height of the cylinder is changed

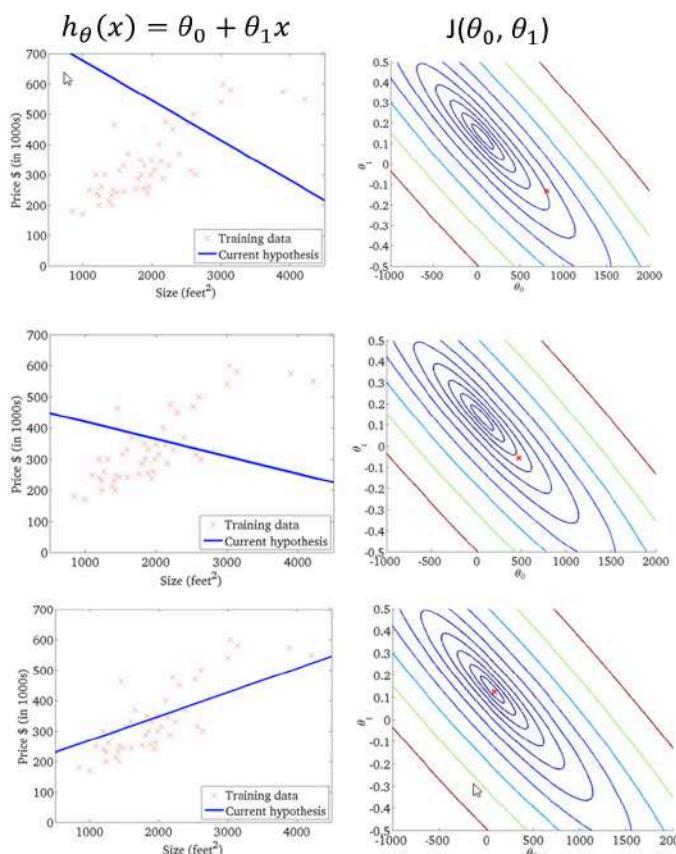


$$F(r, h) = \pi r^2 h$$

**COST FUNCTION**



## Linear regression



### Method

start with a random initial value for the parameters  $\theta_0, \theta_1$ .

change the parameters in such a way that the value of the cost function decreases.

We repeat the steps until we reach a minimum value for the cost function (convergence).

Hypothesis 
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameter  $\theta_0, \theta_1$

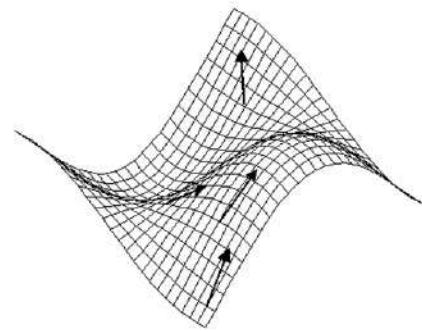
Cost function 
$$\frac{1}{2n} \sum_{i=1}^n (h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal Minimize  $J(\theta_0, \theta_1)$

## Gradient Descent

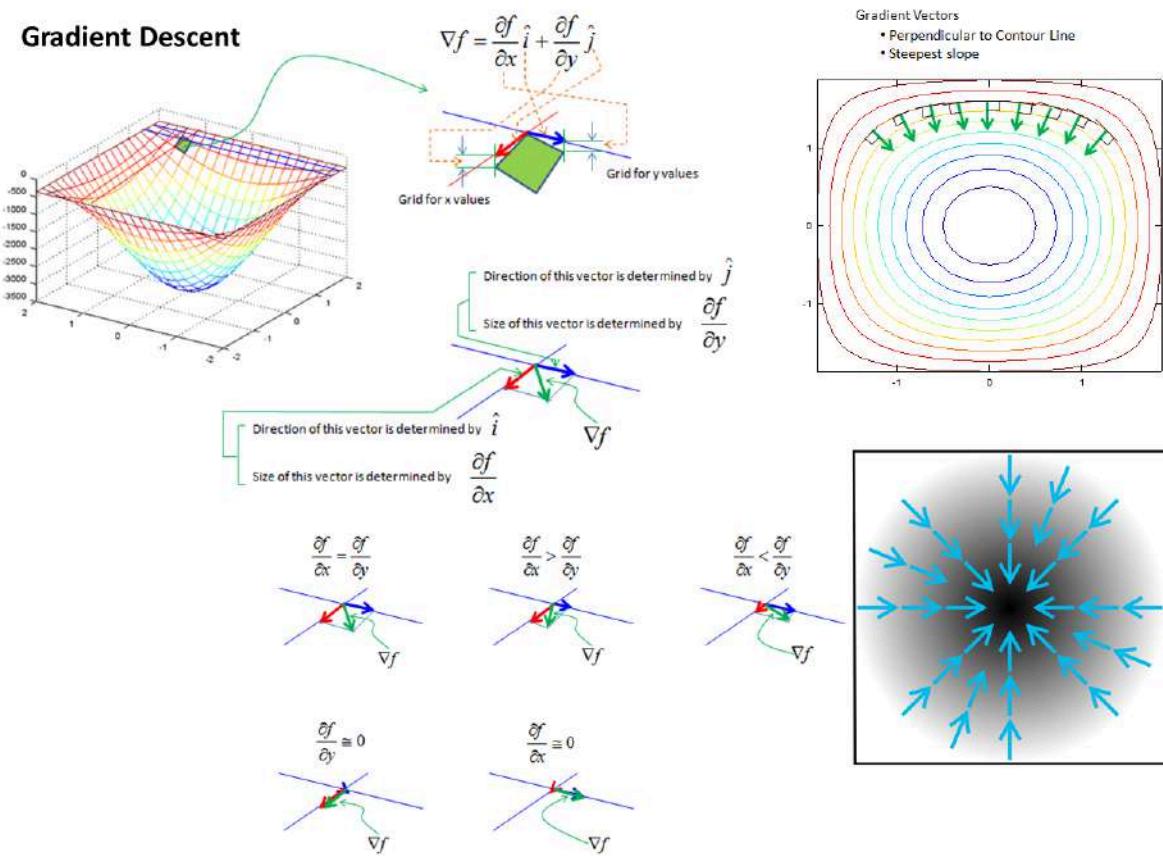
### gradient vector

The gradient vector can be interpreted as the "direction and rate of fastest increase". If the gradient of a function is non-zero at a point  $p$ , the direction of the gradient is the direction in which the function increases most quickly from  $p$ , and the magnitude of the gradient is the rate of increase in that direction, the greatest absolute directional derivative. Further, a point where the gradient is the zero vector is known as a stationary point. The gradient thus plays a fundamental role in optimization theory, where it is used to maximize a function by gradient ascent. (Wikipedia)



Gradient Vectors Shown at Several Points on the Surface of  $\cos(x) \sin(y)$

## Gradient Descent



### Gradient Descent

gradient vector: a vector whose value at a point  $p$  is the vector whose components are the partial derivatives of function  $f$  at  $p$ . shows with  $\nabla f$ .

$$\nabla f = \text{grad}(f)$$

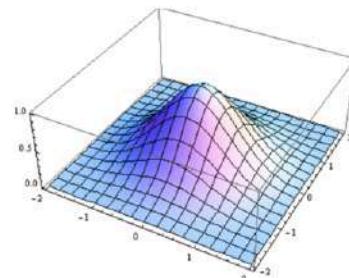
$$U = f(x_1 + x_2 + x_3 + \dots + x_n)$$

$$\nabla U = (u'_{x_1}, u'_{x_2}, u'_{x_3}, \dots, u'_{x_n})$$

?

$$Z = x^2 + 5xy^2 + 4y^3$$

Specify gradient vector at point(1,1)



?

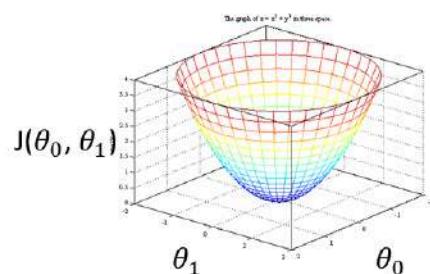
$$J(\theta_0, \theta_1) = x^2y$$

Specify gradient vector at point(2,3)

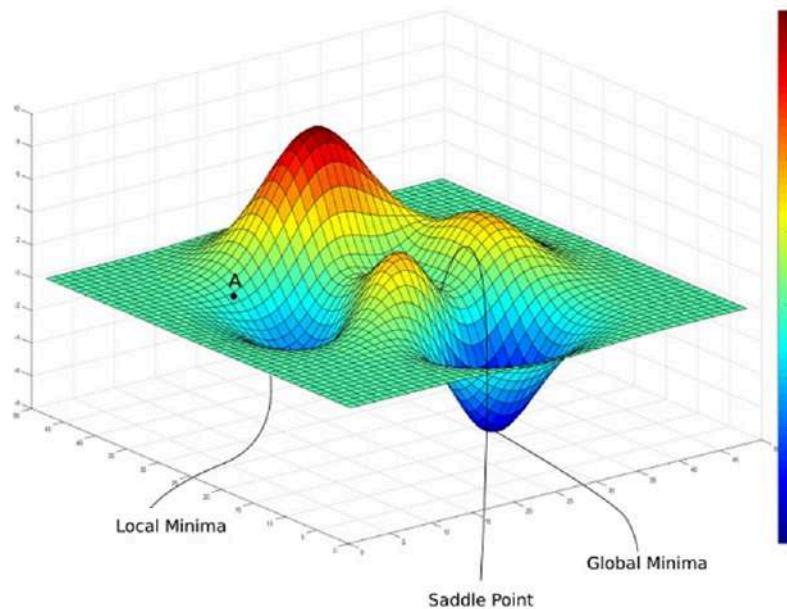
?

$$J = \theta_0^2 + \theta_1^2$$

Specify the gradient vector at point (0,0)



### Gradient Descent



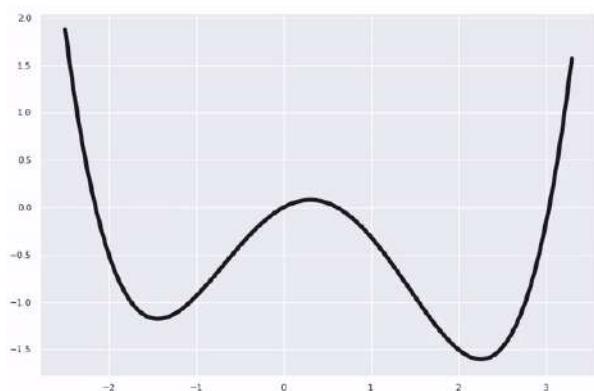
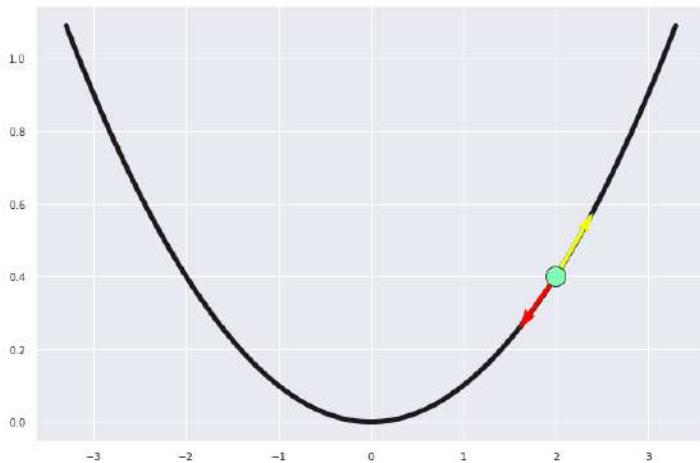
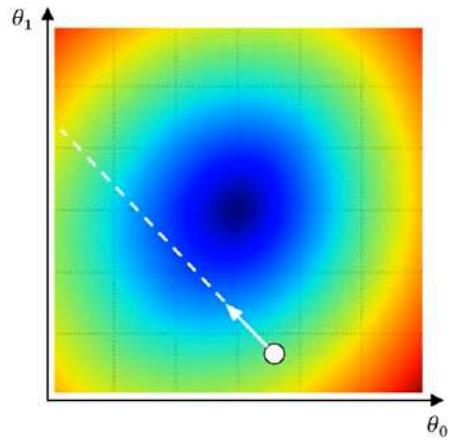
### Mathematic

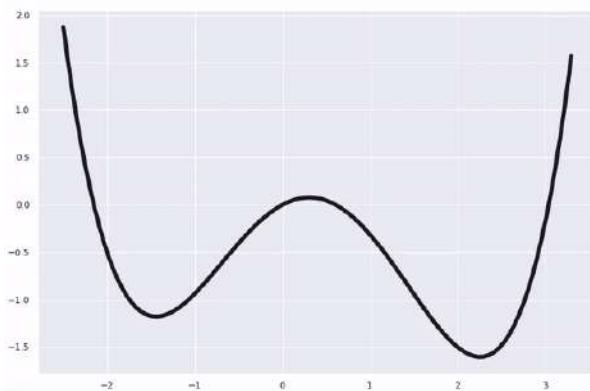
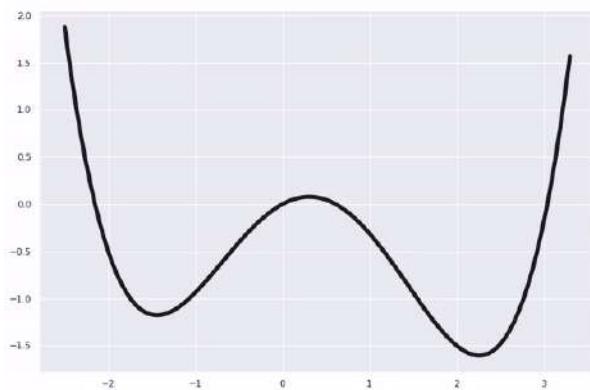
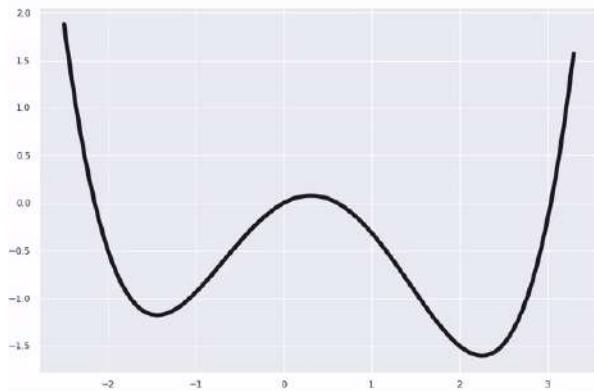
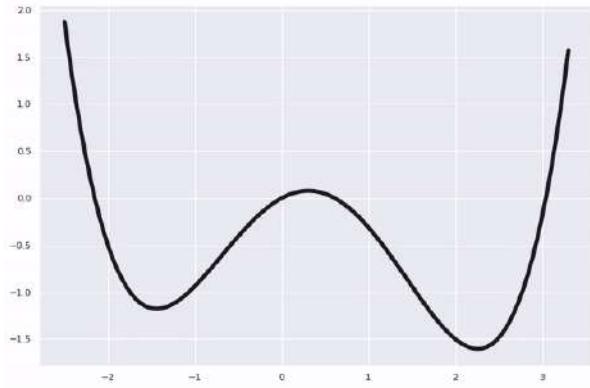
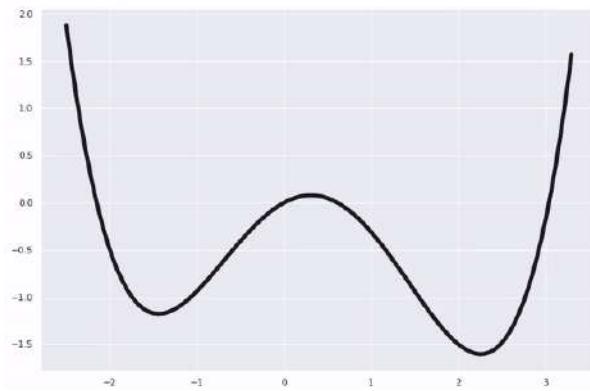
Repeat until convergence

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{for } j=0 \text{ and } i=0$$

$\alpha, \eta$  : Learning rate

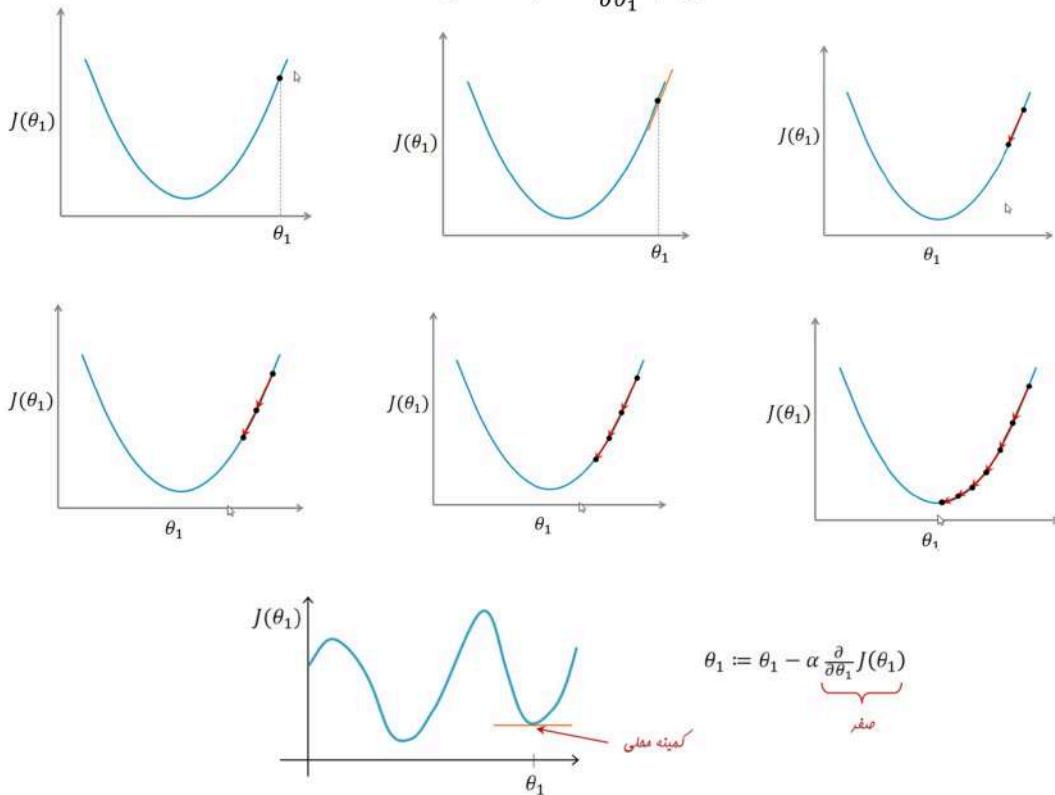
$$\begin{aligned} J(\theta_0, \theta_1) \\ \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \\ \theta^T = [\theta_0 \ \theta_1] \\ \nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \end{bmatrix} \\ -\nabla J = \begin{bmatrix} -\frac{\partial J}{\partial \theta_0} \\ -\frac{\partial J}{\partial \theta_1} \end{bmatrix} \\ \theta^{(new)} := \theta + \alpha (-\nabla J) \\ \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} + \alpha \begin{bmatrix} -\frac{\partial J}{\partial \theta_0} \\ -\frac{\partial J}{\partial \theta_1} \end{bmatrix} \\ \theta_j := \theta_j - \alpha \left( \frac{\partial J}{\partial \theta_j} \right) \end{aligned}$$





## Linear regression

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$



## Linear regression

59

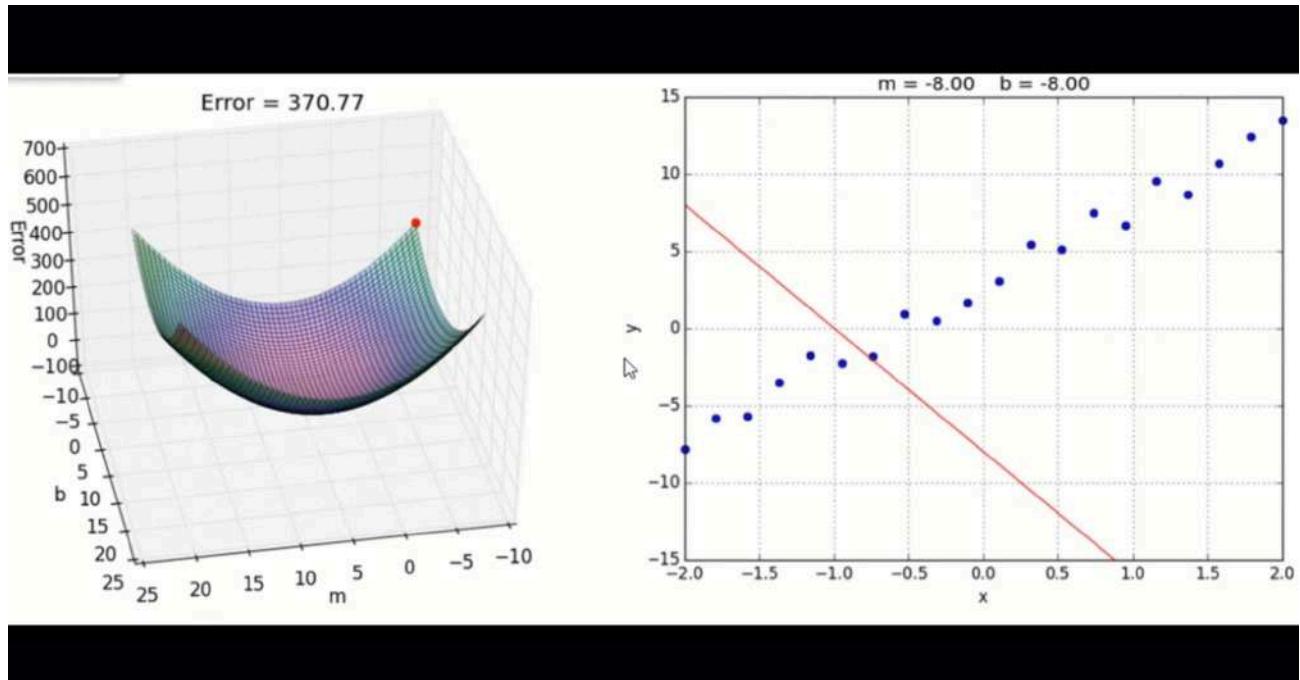
### Mathematic

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

$$J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^n (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^n (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

$$j=0 \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$j=1 \quad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$



## Linear regression

67

Gradient descent	Normal equation
Need to choose $\alpha$	Not to need choosing $\alpha$
Need to multiple repetition	Not to need repetition
Works good even for large amount of n	Because of calculating x transpose is slow for large n
N>=100	N<10000

```
In [127...]
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

def mean_squared_error(y_true, y_predicted):

    # Calculating the Loss or cost
    cost = np.sum((y_true-y_predicted)**2) / len(y_true)
    return cost

# Gradient Descent Function
# Here iterations, learning_rate, stopping_threshold
# are hyperparameters that can be tuned
def gradient_descent(x, y, iterations = 1000, learning_rate = 0.0001,
                     stopping_threshold = 1e-6):

    # Initializing weight, bias, Learning rate and iterations
    current_weight = 0.1
    current_bias = 0.01
    iterations = iterations
    learning_rate = learning_rate
    n = float(len(x))
```

```

costs = []
weights = []
previous_cost = None

# Estimation of optimal parameters
for i in range(iterations):

    # Making predictions
    y_predicted = (current_weight * x) + current_bias

    # Calculationg the current cost
    current_cost = mean_squared_error(y, y_predicted)

    # If the change in cost is less than or equal to
    # stopping_threshold we stop the gradient descent
    if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
        break

    previous_cost = current_cost

    costs.append(current_cost)
    weights.append(current_weight)

    # Calculating the gradients
    weight_derivative = (1/n) * sum(x * (y_predicted-y))
    bias_derivative = (1/n) * sum(y_predicted-y)

    # Updating weights and bias
    current_weight = current_weight - (learning_rate * weight_derivative)
    current_bias = current_bias - (learning_rate * bias_derivative)

    # Printing the parameters for each 1000th iteration
    print(f"Iteration {i+1}: Cost {current_cost}, Weight \
{current_weight}, Bias {current_bias}")

# Visualizing the weights and cost at for all iterations
plt.figure(figsize = (8,6))
plt.plot(weights, costs)
plt.scatter(weights, costs, marker='o', color='red')
plt.title("Cost vs Weights")
plt.ylabel("Cost")
plt.xlabel("Weight")
plt.show()

return current_weight, current_bias

```

```

def main():

    # Data
    X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.81320787,
      55.14218841, 52.21179669, 39.29956669, 48.10584169, 52.55001444,
      45.41973814, 54.35163488, 44.1640495 , 58.16847672, 56.72720806,
      48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])
    Y = np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.23092513,
      78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,
      55.16567719, 82.47884676, 62.00892325, 75.39287043, 81.43619216,
      60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319])

    # Estimating weight and bias using gradient descent
    estimated_weight, estimated_bias = gradient_descent(X, Y, iterations=2000)
    print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {estimated_bias}")

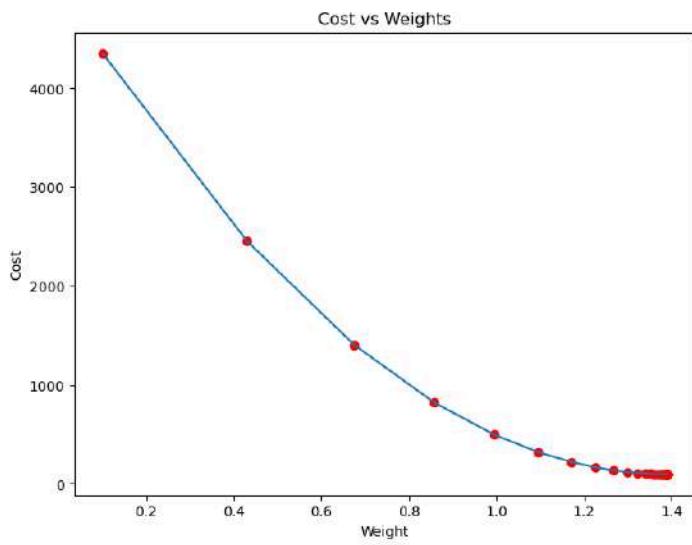
    # Making predictions using estimated parameters
    Y_pred = estimated_weight*X + estimated_bias

    # Plotting the regression Line
    plt.figure(figsize = (8,6))
    plt.scatter(X, Y, marker='o', color='red')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='blue',markerfacecolor='red',
             markersize=10,linestyle='dashed')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()

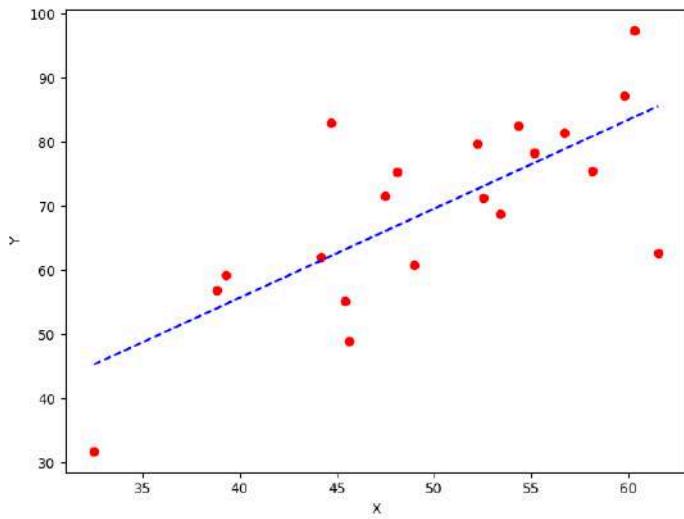
if __name__ == "__main__":
    main()

```

Iteration 1: Cost 4352.088931274409, Weight 0.42996645571281059, Bias 0.16442790653545  
Iteration 2: Cost 2455.5784375358007, Weight 0.6750652968436868, Bias 0.021237827524968778  
Iteration 3: Cost 1404.6759887342773, Weight 0.85774404534812643, Bias 0.024806284355954666  
Iteration 4: Cost 822.3454608270998, Weight 0.99349477515914172, Bias 0.02943739964189955  
Iteration 5: Cost 499.66199560480237, Weight 1.1702990021186792, Bias 0.030907169558405834  
Iteration 6: Cost 320.8526339760065, Weight 1.2263901314834122, Bias 0.032000313564632024  
Iteration 7: Cost 221.77411748836593, Weight 1.2681441186776101, Bias 0.032813099048499534  
Iteration 8: Cost 166.870852087989, Weight 1.2992256057414215, Bias 0.03341718698014148  
Iteration 9: Cost 136.44762129236017, Weight 1.322362534700248, Bias 0.03386592146217562  
Iteration 10: Cost 119.589371235687827, Weight 1.3395855725639687, Bias 0.03419981157887844  
Iteration 11: Cost 110.24780618877969, Weight 1.3524063381524187, Bias 0.03446016582543725  
Iteration 12: Cost 105.0714183050163, Weight 1.3619500773297226, Bias 0.034628940423534405  
Iteration 13: Cost 102.60385614622164, Weight 1.369054402624701, Bias 0.03476416206079033  
Iteration 14: Cost 100.61362700364388, Weight 1.3743428499281516, Bias 0.03486378485565827  
Iteration 15: Cost 99.73288565523464, Weight 1.3782795633256226, Bias 0.03493715490609268  
Iteration 16: Cost 99.24484539690684, Weight 1.3812100524618867, Bias 0.034990758550037325  
Iteration 17: Cost 98.9744103342946, Weight 1.3833915130657024, Bias 0.035029715173176455  
Iteration 18: Cost 98.8245555868343, Weight 1.3850154004087032, Bias 0.03505776862183525  
Iteration 19: Cost 98.74151731567352, Weight 1.386224232880339, Bias 0.03507770581361211  
Iteration 20: Cost 98.69550367410189, Weight 1.387124100488778, Bias 0.035091601293896345  
Iteration 21: Cost 98.67000627752948, Weight 1.3877939761292133, Bias 0.03510099935842256  
Iteration 22: Cost 98.65587249747278, Weight 1.3882296468475025, Bias 0.035107049570320285  
Iteration 23: Cost 98.64804817050401, Weight 1.3886638734397032, Bias 0.03511060765889062  
Iteration 24: Cost 98.64370966601487, Weight 1.3889402312333161, Bias 0.035112310624936714  
Iteration 25: Cost 98.6413054737348, Weight 1.38914596911464911, Bias 0.03511263264881291  
Iteration 26: Cost 98.63997312983149, Weight 1.3892991379580404, Bias 0.03511192670810939  
Iteration 27: Cost 98.63923472386213, Weight 1.3894131745814668, Bias 0.03511045557657794  
Iteration 28: Cost 98.638825433104, Weight 1.3894988013989816, Bias 0.03510841479889936  
Iteration 29: Cost 98.63859851306498, Weight 1.3895613041061574, Bias 0.03510595000665722  
Iteration 30: Cost 98.6384726490638, Weight 1.389608385285069, Bias 0.03510316958799966  
Iteration 31: Cost 98.63840278266409, Weight 1.3896434507814657, Bias 0.0351001542176487  
Iteration 32: Cost 98.63836394589366, Weight 1.3896695718611656, Bias 0.0350996394463888  
Iteration 33: Cost 98.6383423033937, Weight 1.3896890347597692, Bias 0.035099364348427625  
Iteration 34: Cost 98.63833018865651, Weight 1.3897035413401346, Bias 0.03509922611309702  
Iteration 35: Cost 98.63832335348422, Weight 1.38971434584608416, Bias 0.03508673660414244  
Iteration 36: Cost 98.63831944384746, Weight 1.3897224291653643, Bias 0.03508319339834686  
Iteration 37: Cost 98.63831756509264, Weight 1.3897284554507587, Bias 0.035079610223050714



Estimated Weight: 1.3897284554507587  
Estimated Bias: 0.035079610223050714



## Multiple Linear Regression

Size (feet <sup>2</sup> )	#bedrooms	#floors	age	Price(1000\$)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$$

In [128]

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

def train (X, y, iterations = 1000, learning_rate = 0.0001,
           stopping_threshold = 1e-6):

    m, n = X.shape

    # Initializing weight, bias, Learning rate and iterations
    current_weight = np.random.rand(n ,1)
    current_bias = np.zeros([1,1])

    iterations = iterations
    learning_rate = learning_rate

    costs = []
    weights = []
    biass=[]
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(iterations):

        # Making predictions
        prediction = (X@current_weight) + current_bias

        # Calculationg the current cost
        current_cost = np.sum(1/(m*2) * (prediction - y)**2)

        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
            break

        previous_cost = current_cost

        costs.append(current_cost)
        weights.append(current_weight)
        biass.append(current_bias)

        # Calculating the gradients
        weight_derivative = X.T@(prediction-y)
        bias_derivative = np.sum(prediction-y)

        # Updating weights and bias
        current_weight = current_weight - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

    return costs , weights
```

In [129]

```
X = np.array([[32.50234527],
              [53.42680403],
              [61.53035803],
              [47.47563963],
```

```

[59.81320787],
[55.14218841],
[52.21179669],
[39.2995669],
[48.18504169],
[52.55001444],
[45.41973014],
[54.35163488],
[44.1640495 ],
[58.16847072],
[56.72720806],
[48.95588857],
[44.68719623],
[60.29732685],
[45.61864377],
[38.81681754]]))

y = np.array([[31.70700585],
[68.77759598],
[62.5623823],
[71.54663223],
[87.23092513],
[78.21151827],
[79.64197305],
[59.17148932],
[75.3312423 ],
[71.30087989],
[55.16567715],
[82.47884676],
[62.00892325],
[75.39287043],
[81.43619216],
[60.72360244],
[82.89250373],
[97.37989686],
[48.84715332],
[56.87721319]])]

#train (X, y, iterations = 10, learning_rate = 0.00001, stopping_threshold = 1e-6)

weights= a[1]
costs=a[0]

print(weights)

print("*"*10)

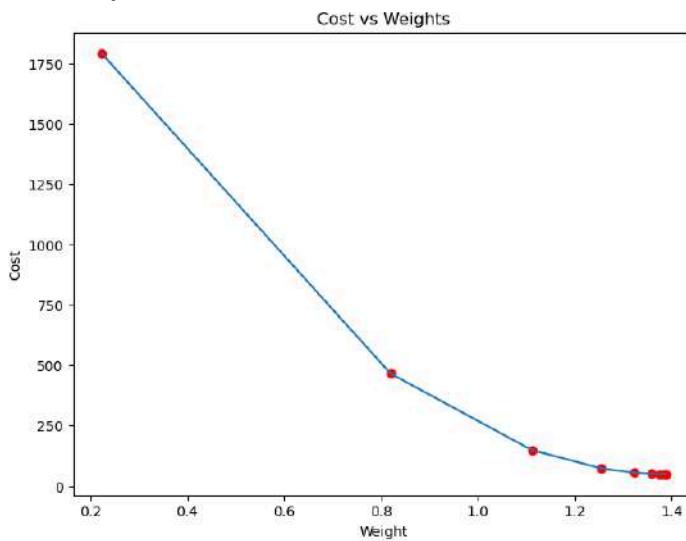
print(costs)

print("*"*10)
weights2 = [float(weights[i][0]) for i in range(len(weights))]
print(weights2)

#Visualizing the weights and cost at for all iterations
plt.figure(figsize = (8,6))
plt.plot(weights2,costs)
plt.scatter(weights2, costs, marker='o', color='red')
plt.title("Cost vs Weights")
plt.ylabel("Cost")
plt.xlabel("Weight")
plt.show()

[array([[0.22277729]]), array([[0.8194654]]), array([[1.11112092]]), array([[1.25367942]]), array([[1.32336073]]), array([[1.35742036]]), array([[1.37406845]]), array([[1.38220596]]),
array([[1.38618357]]), array([[1.38812786]])]
*****
[1791.1321355029274, 465.46597374352183, 148.74298646246558, 73.0727658797899, 54.99393181200671, 50.67460710865342, 49.64265069779575, 49.39609949320669, 49.33719423612124, 49.323120
6166852]
*****
[0.2227772935759561, 0.8194653995753225, 1.1111209173605427, 1.2536794208555224, 1.3233607285057407, 1.3574203592263152, 1.3740684546568185, 1.3822059580182762, 1.386183565301528, 1.3
81278551325336]

```



## Preprocessing

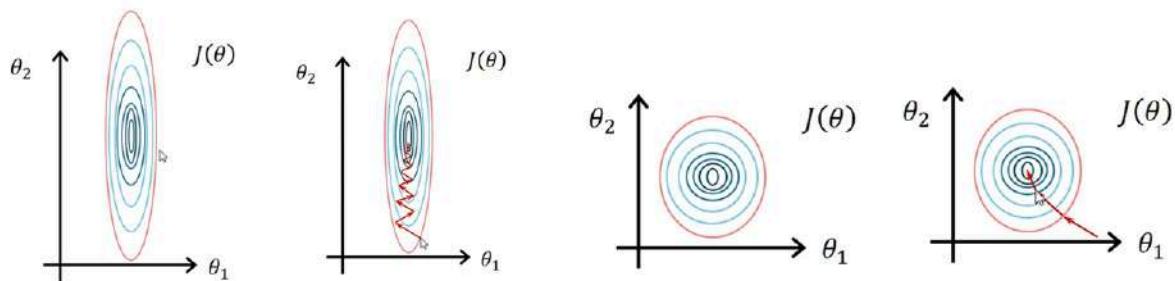
- Practical tricks in multivariate regression, feature scaling, learning rate determination

scaling (normalizing) in the preprocessing phase We put the attribute values in the same scale.

The purpose of normalization: increasing the speed of convergence in gradient descent

If  $x_2$  is too small,  $\theta_2$  must be too large to moderate its effect. It becomes much faster by normalizing the algorithms

$$\begin{aligned}x_1 &: \text{size } (30,2500) \\x_2 &: \# \text{rooms } (6) \\1000 &= \theta_1 x_1 + \theta_2 x_2\end{aligned}$$



z-score method (standardization)

transforms the info into distribution with a mean of 0 and a typical deviation of 1

$$x_j := \frac{x_j - \mu_j}{\sigma_j}$$

Min-Max feature scaling (normalization)

rescales the feature to [0,1]

$$x_j := \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j)}$$

Maximum absolute scaling

The maximum absolute scaling rescales each feature between -1 and 1

$$x_j := \frac{x_j}{|\max(x_j)|}$$

```
In [130]: import pandas as pd
concrete_data=pd.read_csv(r"D:\AI_MachineLearning\datasets\CONCRETE\regression\Concrete_Data .csv")

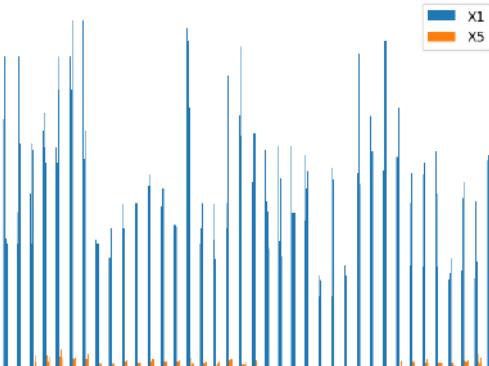
display(concrete_data.head())
```

```

      X1   X2   X3   X4   X5   X6   X7   X8   y
0  540.0  0.0  0.0  1620.  2.5  1040.0  676.0  28  79.99
1  540.0  0.0  0.0  1620.  2.5  1055.0  676.0  28  61.89
2  332.5  142.5  0.0  228.0  0.0  932.0  594.0  270  40.27
3  332.5  142.5  0.0  228.0  0.0  932.0  594.0  365  41.05
4  198.6  132.4  0.0  192.0  0.0  978.4  825.5  360  44.30

```

```
In [131]: import matplotlib.pyplot as plt
data=concrete_data[["X1" , "X5"]]
data.plot(kind = 'bar')
plt.axis('off')
plt.show()
```



```
In [132]: #standardization
X1=concrete_data.copy()
X1=concrete_data.drop("y" , axis=1)
for i in X1.columns:
    X1[i] = (X1[i] - X1[i].mean())/X1[i].std()
X1
```

```
Out[132]:
      X1   X2   X3   X4   X5   X6   X7   X8
0  2.476712 -0.856472 -0.846733 -0.916319 -0.620147  0.862735 -1.217079 -0.279597
1  2.476712 -0.856472 -0.846733 -0.916319 -0.620147  1.055651 -1.217079 -0.279597
2  0.491187  0.795140 -0.846733  2.174405 -1.038638 -0.526262 -2.239829  3.551340
3  0.491187  0.795140 -0.846733  2.174405 -1.038638 -0.526262 -2.239829  5.055221
4  -0.790075  0.678079 -0.846733  0.488555 -1.038638  0.070492  0.647569  4.976069
...
1025 -0.045623  0.487998  0.564271 -0.092126  0.451190 -1.322363 -0.065861 -0.279597
1026  0.392628 -0.856472  0.959602  0.675872  0.702285 -1.993711  0.496651 -0.279597
1027 -1.269472  0.759210  0.850222  0.521336 -0.017520 -1.035561  0.080068 -0.279597
1028 -1.168042  1.307430 -0.846733 -0.279443  0.852942  0.214537  0.191074 -0.279597
1029 -0.193939  0.308349  0.376762  0.891286  0.400971 -1.394385 -0.150675 -0.279597
```

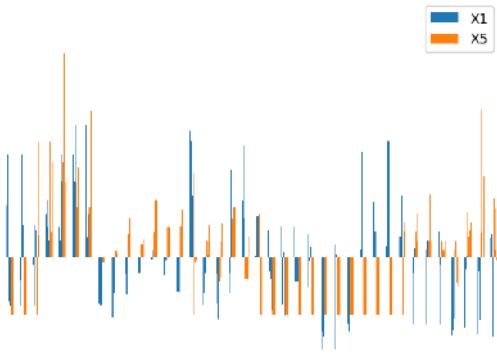
1030 rows × 8 columns

```
In [133]: X11=concrete_data.copy()
X11=concrete_data.drop("y" , axis=1)
X11 =(X11 - X11.mean(axis=0))/X11.std(axis=0)
X11
```

```
Out[133]:
      X1   X2   X3   X4   X5   X6   X7   X8
0  2.476712 -0.856472 -0.846733 -0.916319 -0.620147  0.862735 -1.217079 -0.279597
1  2.476712 -0.856472 -0.846733 -0.916319 -0.620147  1.055651 -1.217079 -0.279597
2  0.491187  0.795140 -0.846733  2.174405 -1.038638 -0.526262 -2.239829  3.551340
3  0.491187  0.795140 -0.846733  2.174405 -1.038638 -0.526262 -2.239829  5.055221
4  -0.790075  0.678079 -0.846733  0.488555 -1.038638  0.070492  0.647569  4.976069
...
1025 -0.045623  0.487998  0.564271 -0.092126  0.451190 -1.322363 -0.065861 -0.279597
1026  0.392628 -0.856472  0.959602  0.675872  0.702285 -1.993711  0.496651 -0.279597
1027 -1.269472  0.759210  0.850222  0.521336 -0.017520 -1.035561  0.080068 -0.279597
1028 -1.168042  1.307430 -0.846733 -0.279443  0.852942  0.214537  0.191074 -0.279597
1029 -0.193939  0.308349  0.376762  0.891286  0.400971 -1.394385 -0.150675 -0.279597
```

1030 rows × 8 columns

```
In [134]: data2=X1[["X1" , "X5"]]
data2.plot(kind = 'bar')
plt.axis('off')
plt.show()
```



```
In [135]: #maximum absolute scaling  
X2 = np.sqrt(data_norm())
```

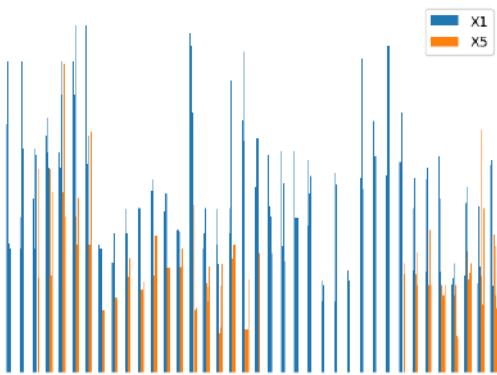
```
X2=concrete_data.copy()
X2=concrete_data.drop("y" , axis=1)
for column in X2.columns:
    X2[column]=X2[column]/X2[column].abs().max()
```

x2

Out[135]:	X1	X2	X3	X4	X5	X6	X7	X8
<b>0</b>	1.000000	0.000000	0.000000	0.655870	0.077640	0.908297	0.681040	0.076712
<b>1</b>	1.000000	0.000000	0.000000	0.655870	0.077640	0.921397	0.681040	0.076712
<b>2</b>	0.615741	0.396494	0.000000	0.923077	0.000000	0.813974	0.598428	0.739726
<b>3</b>	0.615741	0.396494	0.000000	0.923077	0.000000	0.813974	0.598428	1.000000
<b>4</b>	0.367778	0.368392	0.000000	0.777328	0.000000	0.854498	0.831654	0.986301
...	...	...	...	...	...	...	...	...
<b>1025</b>	0.511852	0.322760	0.451274	0.727126	0.276398	0.759913	0.774028	0.076712
<b>1026</b>	0.596667	0.000000	0.577711	0.793522	0.322981	0.714323	0.819464	0.076712
<b>1027</b>	0.275000	0.387869	0.542729	0.780162	0.189441	0.779389	0.785815	0.076712
<b>1028</b>	0.294630	0.519477	0.000000	0.710931	0.350932	0.864279	0.794781	0.076712
<b>1029</b>	0.483148	0.279633	0.391304	0.812146	0.267081	0.755022	0.767177	0.076712

1030 rows × 8 columns

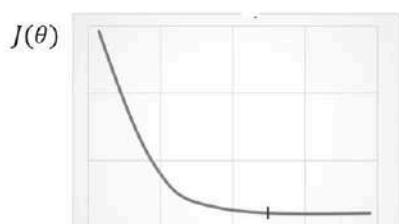
```
In [136]: data3=X2[['X1' , "X5"]]
data3.plot(kind = 'bar')
plt.axis('off')
plt.show()
```



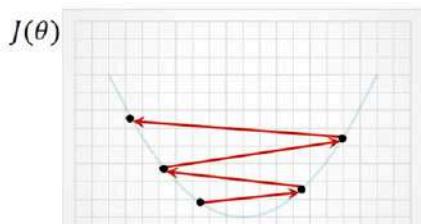
**Determine the value of  $\alpha$** 

Convergence test

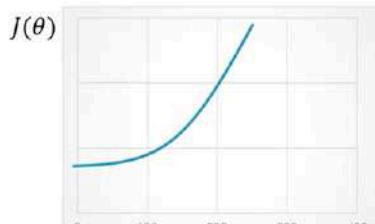
Convergence has occurred if the value of  $J(\theta)$  changes less than  $10^{-3}$  in one iteration.



Repeated times  
convergence



$\theta$   
divergence



Repeated times  
divergence

Try a number like  $1/1000$  and multiply by one number each time to reach the appropriate rate. It changes logarithmically.

**Test and Train**

## Linear regression

### linear regression with Sklearn

1. Import library (from `sklearn.linear_model import LinearRegression`)  
(from `scipy.stats import linregress`)
1. Make data (from `sklearn.model_selection import train_test_split`)
2. fit the data  
`reg= LinearRegression( )`  
`reg.fit(x_train , y_train)`
1. Predict new X  
`y_hat = reg.predict(x_test)`

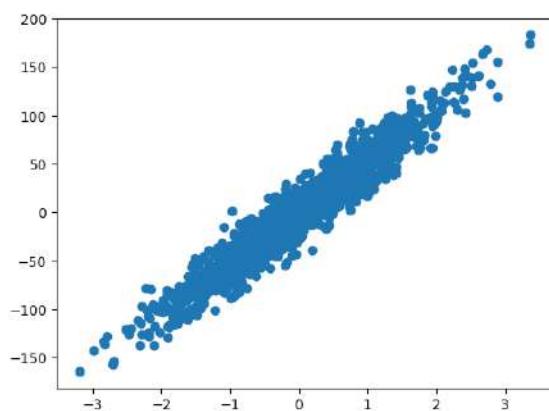
## Sklearn

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
```

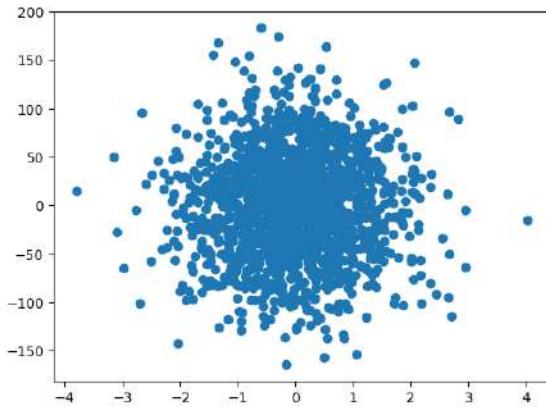
```
In [2]: X,y= make_regression(n_samples=1500,n_features=2 ,n_targets=1, random_state=20, noise=15 )
```

```
In [3]: print(X.shape , y.shape)
(1500, 2) (1500,)
```

```
In [4]: plt.scatter(X[:, 0] , y)
plt.show()
```

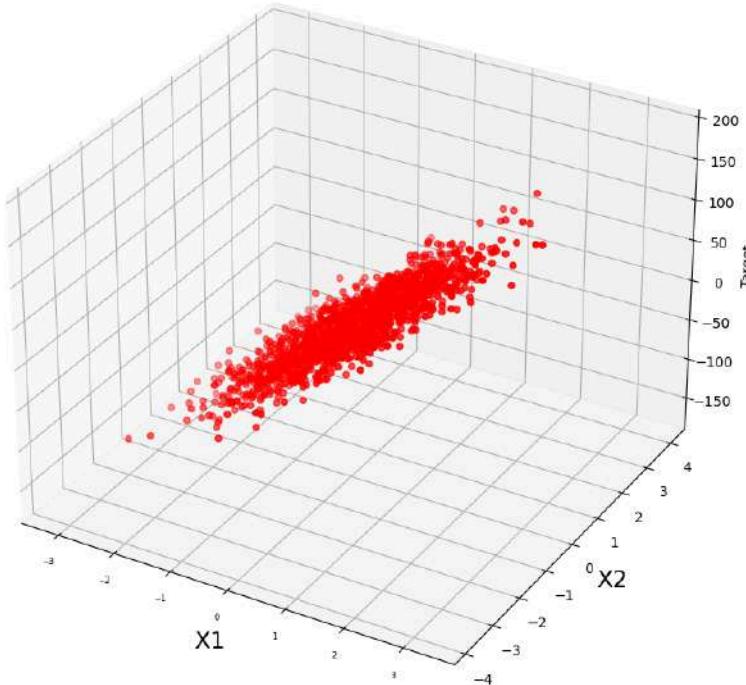


```
In [5]: plt.scatter(X[:, 1] , y)
plt.show()
```



```
In [6]: from mpl_toolkits import mplot3d
fig = plt.figure(figsize=(10,10))
ax=plt.axes(projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], y , color="red")
ax.set_title("3D plot")
ax.set_xlabel("X1" , fontsize=17)
ax.set_ylabel("X2" , fontsize=17)
ax.set_zlabel("Target" , fontsize=10)
ax.tick_params(axis='x', labelsize=6)
plt.show()
```

3D plot



```
In [7]: df=pd.DataFrame(X , columns=["X1", "X2"])
df
```

```
Out[7]:   X1      X2
 0 -0.596364  0.631029
 1 -0.646425  0.913473
 2  0.394063 -0.164301
 3  1.031434 -0.808968
 4 -1.502834 -0.763108
 ...
1495  1.257544  0.201318
1496  1.267289 -1.242577
1497  0.981740 -0.110902
1498 -0.120893  0.735879
1499 -0.093369  0.170153
```

1500 rows × 2 columns

```
In [8]: #add y column y
df["y"] = y
df
```

```
Dat[8]:
```

	X1	X2	y
0	-0.596364	0.631029	-37.905873
1	-0.646425	0.913473	-45.554255
2	0.394063	-0.164301	5.691576
3	1.031434	-0.808968	48.315317
4	-1.502834	-0.763108	-97.691419
...	...	...	...
1495	1.257544	0.201318	68.482958
1496	1.267289	-1.242577	58.539574
1497	0.981740	-0.110902	83.308583
1498	-0.120893	0.735879	-4.394360
1499	-0.093369	0.170153	-12.906895

1500 rows × 3 columns

```
In [9]: import sklearn
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
In [10]: #make data
X_train, X_test, y_train, y_test= sklearn.model_selection.train_test_split(X, y, test_size=0.2 , shuffle=False)

#fit data
reg=LinearRegression()
reg.fit(X_train ,y_train)
weights= reg.coef_
print(weights)
bias=reg.intercept_
print(bias)

#predict
y_new= reg.predict([[1.2,0.5]])
print(y_new)

#yhat
y_hat= reg.predict(X_test)
#print(y_hat)

#score
print(reg.score(X_test , y_test)*100)

#r2score
print(round(metrics.r2_score(y_hat , y_test)*100,2))

[52.16846205  2.99856868
 -0.7387902935417028
 [63.3626485]
 92.61484593280022
 91.73
```

```
In [11]: import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Assuming you have trained your Linear Regression model and stored it in 'model'

# Create a mesh grid of X1 and X2 values
X1 = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)
X2 = np.linspace(min(X[:, 1]), max(X[:, 1]), 100)
X1, X2 = np.meshgrid(X1, X2)

# Predict target values for each point on the grid
X_grid = np.column_stack((X1.ravel(), X2.ravel()))
y_pred = reg.predict(X_grid)
y_pred = y_pred.reshape(X1.shape)

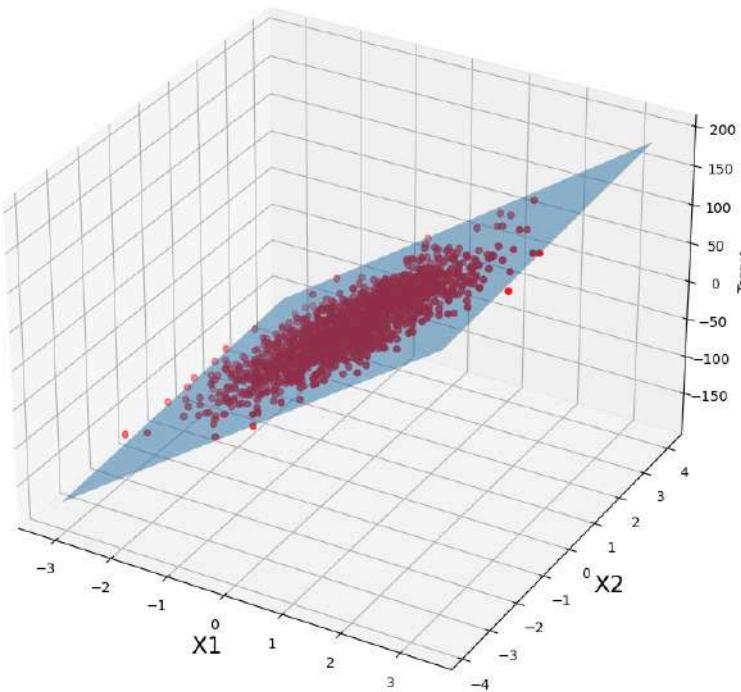
# Plot 3D scatter plot
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], y, color="red")

# Plot surface manifold
ax.plot_surface(X1, X2, y_pred, alpha=0.5)

# Set titles and labels
ax.set_title("3D plot with Surface Manifold")
ax.set_xlabel("X1", fontsize=17)
ax.set_ylabel("X2", fontsize=17)
ax.set_zlabel("Target", fontsize=10)

plt.show()
```

3D plot with Surface Manifold



```
In [148]:  
import sklearn  
from sklearn.linear_model import LinearRegression  
from sklearn import metrics  
from sklearn import preprocessing
```

```
In [149]:  
import numpy as np  
import pandas as pd  
data=pd.read_csv(r"D:\AI_Machinlearning\datasets\ENERGY\sklearn\en.csv")  
data.tail(10)
```

```
Out[149]:  
X1 X2 X3 X4 X5 X6 X7 X8 Y1  
758 0.66 759.5 318.5 220.5 3.5 4 0.4 5 14.92  
759 0.66 759.5 318.5 220.5 3.5 5 0.4 5 15.16  
760 0.64 784.0 343.0 220.5 3.5 2 0.4 5 17.69  
761 0.64 784.0 343.0 220.5 3.5 3 0.4 5 18.19  
762 0.64 784.0 343.0 220.5 3.5 4 0.4 5 18.16  
763 0.64 784.0 343.0 220.5 3.5 5 0.4 5 17.88  
764 0.62 808.5 367.5 220.5 3.5 2 0.4 5 16.54  
765 0.62 808.5 367.5 220.5 3.5 3 0.4 5 16.44  
766 0.62 808.5 367.5 220.5 3.5 4 0.4 5 16.48  
767 0.62 808.5 367.5 220.5 3.5 5 0.4 5 16.64
```

```
In [150]:  
X= np.array(data.drop(["Y1"] , axis= 1))  
y=np.array(data["Y1"])  
  
y=y.reshape((768,1))
```

```
In [151]:  
#normalize  
Xnormalized= preprocessing.normalize(X)
```

```
In [152]:  
#test and tran split  
X_train , X_test , y_train , y_tset = sklearn.model_selection.train_test_split(X, y , test_size= 0.2 , shuffle=False)  
linear= LinearRegression()  
  
linear.fit(X_train , y_train)  
  
theta= linear.coef_  
print(theta)  
  
print("-----")  
#predict new data  
y_new= linear.predict([[0.98,72,90,72,3,2,0.16,1]])  
print(y_new)  
  
print("-----")  
y_hat= linear.predict(X_test)  
#print(y_hat)  
  
print("-----")  
#accuracy  
from sklearn import metrics  
  
print(linear.score(X_test , y_tset)*100)  
  
print("-----")  
#r2score  
print(round(metrics.r2_score(y_hat , y_tset)*100,2))
```

```

[-6.45580164e+01 -3.99085024e+11 3.99085024e+11 7.98170049e+11
 4.24481963e+00 -2.24118177e-02 2.22191931e+01 2.98581083e-01]]
-----
[[6.4651774e+13]]
-----
91.2163594211527
-----
88.51

```

```

In [153]: #accuracy from the scratch

numerator=sum (( y_tset - y_hat)**2)
Denominator=sum (( y_tset - y_tset.mean())**2)

rse= numerator / Denominator

R_square= 1 - rse

print(f" the accuracy of the algorihm is: {round(float(R_square*100), 2)} ")
the accuracy of the algorihm is: 91.22

```

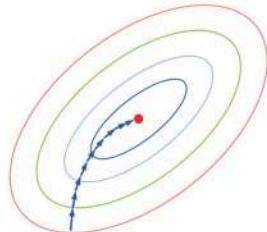
## Mini Batch Gradient Descent

### Linear regression

#### Full batch gradient descent

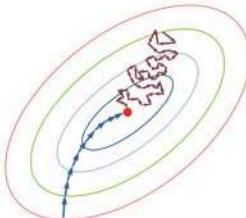
full training set at each step  
very slow on very large training data

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$



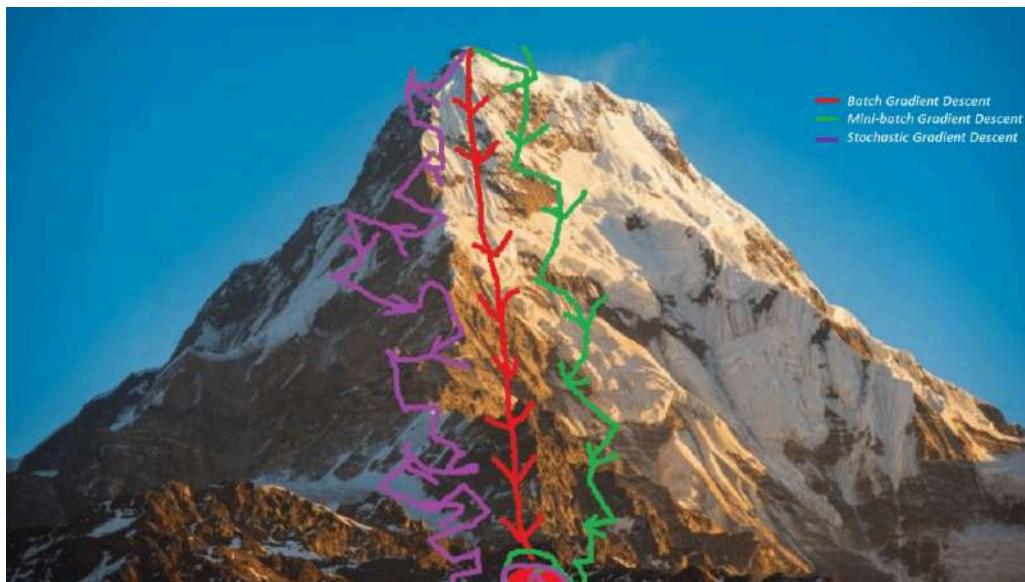
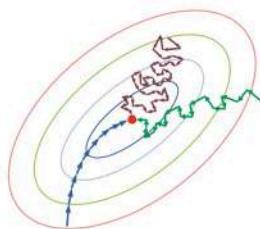
#### Stochastic gradient descent (SGD)

just one example at a time to take a single step.



#### Min batch gradient descent

use a batch of a fixed number of training examples.  
achieve the advantages of both the former variants



```

In [154]: # Importing Libraries
import numpy as np

```

```

import matplotlib.pyplot as plt

def train(X, y, batch_size=32, iterations=1000, learning_rate=0.0001, stopping_threshold=1e-6):
    m, n = X.shape

    # Initializing weight, bias, Learning rate and iterations
    current_weight = np.random.rand(n, 1)
    current_bias = np.zeros([1, 1])

    iterations = iterations
    learning_rate = learning_rate

    costs = []
    weights = []
    biases = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(iterations):
        # Shuffle the data
        indices = np.random.permutation(m)
        X_shuffled = X[indices]
        y_shuffled = y[indices]

        # Mini-batch gradient descent
        for j in range(0, m, batch_size):
            X_batch = X_shuffled[j:j+batch_size]
            y_batch = y_shuffled[j:j+batch_size]

            # Making predictions
            prediction = (X_batch @ current_weight) + current_bias

            # Calculationg the current cost
            current_cost = np.sum((1 / (2 * batch_size)) * (prediction - y_batch) ** 2)

            # Calculating the gradients
            weight_derivative = (1 / batch_size) * X_batch.T @ (prediction - y_batch)
            bias_derivative = (1 / batch_size) * np.sum(prediction - y_batch)

            # Updating weights and bias
            current_weight = current_weight - (learning_rate * weight_derivative)
            current_bias = current_bias - (learning_rate * bias_derivative)

            # Calculate cost over entire dataset for monitoring
            prediction = (X @ current_weight) + current_bias
            total_cost = np.sum((1 / (2 * m)) * (prediction - y) ** 2)

            # If the change in cost is less than or equal to
            # stopping_threshold we stop the gradient descent
            if previous_cost and abs(previous_cost - total_cost) <= stopping_threshold:
                break

            previous_cost = total_cost

            costs.append(total_cost)
            weights.append(current_weight)
            biases.append(current_bias)

    return costs, weights, biases

```

In [4]:

```

import numpy as np
a=np.array([[1,2,3,],
           [3,4,5,],
           [6,7,8]])
```

Out[4]:

```

array([3, 0, 1, 4, 2])
```

In [7]:

```

X = np.array([[32.50234527],
              [53.42680403],
              [61.5303503],
              [47.47563963],
              [59.81320787],
              [55.14218841],
              [52.21179669],
              [39.2995669],
              [48.10504169],
              [52.55001444],
              [45.41973914],
              [54.35163488],
              [44.1640495 ],
              [58.16847072],
              [56.72720806],
              [48.95588857],
              [44.68719623],
              [60.29732685],
              [45.61864377],
              [38.81681754]])
```

y = np.array([[31.70700585],
 [68.77759598],
 [62.5623823],
 [71.54663223],
 [87.23092513],
 [78.21151827],
 [79.64197305],
 [59.17148932],
 [75.3312423 ],
 [71.30087989],
 [55.16567715],
 [82.4784676],
 [62.00892325],
 [75.39287043],
 [81.43619216],
 [60.72360244],
 [82.89250373],
 [97.37989686],
 [48.84715332],
 [56.87721319]])

a=train (X, y, iterations = 500, learning\_rate = 0.00001, stopping\_threshold = 1e-6)

weights= a[1]

costs=a[0]

#print(weights)

#print("\*\*\*\*10")

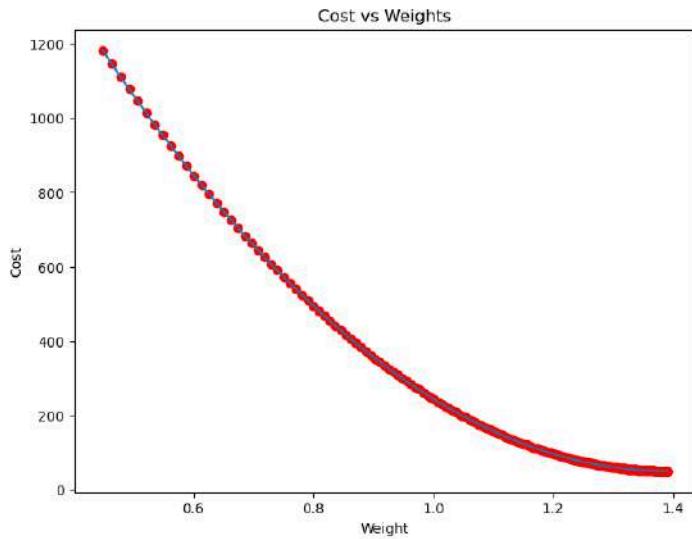
#print(costs)

#print("\*\*\*\*10")

weights2 = [float(weights[i][0]) for i in range(len(weights))]

```
#print(weights2)

#Visualizing the weights and cost at for all iterations
plt.figure(figsize = (8,6))
plt.plot(weights2 , costs)
plt.scatter(weights2, costs, marker='o', color='red')
plt.title("Cost vs Weights")
plt.ylabel("Cost")
plt.xlabel("Weight")
plt.show()
```



# Machine learning

## PolyNomial Regression

Morteza khorsand



### Polynomial regression

2

**Dot product** (inner product, projection product) :  
an algebraic operation that takes two equal-length sequences of numbers (usually coordinate vectors),  
and returns a single number.

$$A = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

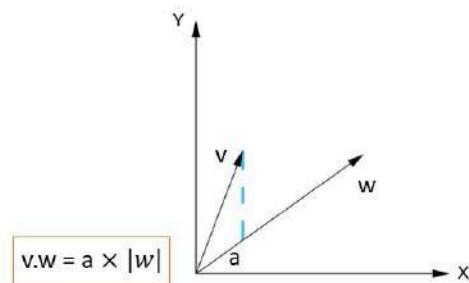
$$v = \begin{bmatrix} 4 \\ 2 \\ -1 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 \\ 6 \\ -7 \end{bmatrix}$$

$$w = \begin{bmatrix} 7 \\ -3 \\ 5 \end{bmatrix}$$

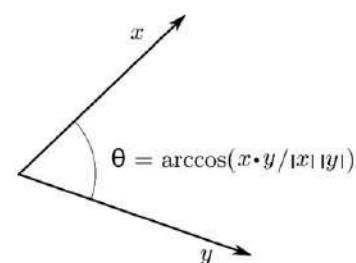
$$A \cdot B = 10 + 18 - 28 = 0$$

$$V \cdot W = 28 - 6 - 5 = 16$$



### Dot product application

$$a \cdot b = \|a\| \|b\| \cos \theta$$



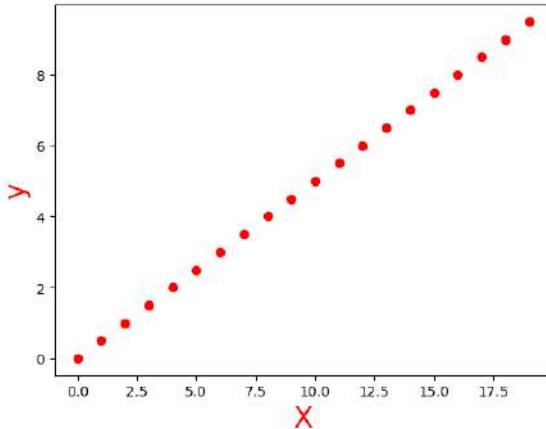
```
In [2]: M
1 import numpy as np
2 A= np.array([[2,3,4])
3 B=np.array([5,6,-7])
4
5
6 print(A.T@B)
7
8 AB=np.inner(A, B)
9 print(AB)

0
0
```

```
In [4]: M
1 A=np.linalg.norm(A)
2 B=np.linalg.norm(B)
3 A
4 B
5
6
```

Out[4]: 10.488088481701515

```
In [4]: M
1 import matplotlib.pyplot as plt
2 x= np.array([i for i in range(20)])
3 y=np.array([x*0.5])
4
5 plt.scatter(x, y , c="red")
6 plt.xlabel("X" , color="red" , fontsize=20)
7 plt.ylabel("y" , color="red" , fontsize=20)
8
9 plt.show()
```



```
In [5]: M
1 import pandas as pd
2 import sklearn
3 from sklearn.datasets import load_iris
4
5 iris=load_iris()
6
7 #print(iris.data)
8 #print(iris.feature_names)
9 #print(iris.target_names)
10 #print(iris.DESCR)
11
12
13 df=pd.DataFrame(iris.data , columns=iris.feature_names)
14 df.head()
```

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
In [7]: M
1 X0=df.iloc[:, 0] # X0=df[["sepal Length (cm)"]]
2 #display(X0)
3
4 X1=df.iloc[:, 1]
5
6
7
8
```

```
In [8]: M
1 relate=np.inner(X0, X1)
2 print(relate)

2673.43
```

```
In [9]: M
1 #normalize Length
2 X0norm = X0 / np.sqrt(np.sum(X0**2))
3 X1norm = X1 / np.sqrt(np.sum(X1**2))
```

```
In [10]: M
1 relate1=np.inner(X0norm, X1norm)
2 print(relate1)

0.9780132025219612
```

```
In [11]: X0=df.iloc[:, 0]
2 X0norm = X0 / np.sqrt(np.sum(X0**2))
3
4 X3= df.iloc[:, 3]
5 X3norm = X3 / np.sqrt(np.sum(X3**2))
6
7 relate2=np.inner(X1norm, X3norm)
8 #print(relate2*100)
9
10
11 print(relate2)

0.8088210631226513
```

```
In [10]: X1=df.iloc[:, 1]
2 X1norm = X1 / np.sqrt(np.sum(X1**2))
3
4 X2= df.iloc[:, 2]
5 X2norm = X2 / np.sqrt(np.sum(X2**2))
6
7 relate2=np.inner(X1norm, X2norm)
8 #print(relate2*100)
9
10
11 print(relate2)

0.8710973738895628
```

```
In [12]: Dataset={"width":np.linspace(1.5,2.2 , 20),
2 "length":((np.linspace(1.5,2.2 , 20)) *1.5 +0.2)}
3
4
5 df= pd.DataFrame(Dataset)
6 df
```

Out[12]:

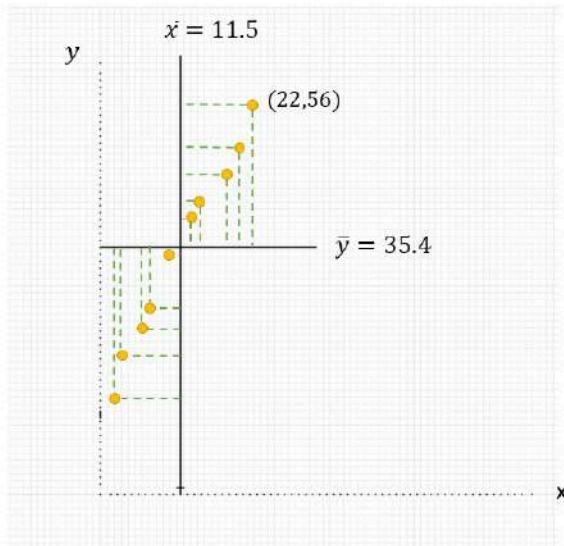
	width	length
0	1.500000	2.450000
1	1.536842	2.505263
2	1.573684	2.560526
3	1.610526	2.615789
4	1.647368	2.671053
5	1.684211	2.726316
6	1.721053	2.781579
7	1.757895	2.836842
8	1.794737	2.892105
9	1.831579	2.947368
10	1.868421	3.002632
11	1.905263	3.057895
12	1.942105	3.113158
13	1.978947	3.168421
14	2.015789	3.223684
15	2.052632	3.278947
16	2.089474	3.334211
17	2.126316	3.389474
18	2.163158	3.444737
19	2.200000	3.500000

```
In [13]: X1=df.iloc[:, 0]
2 X1norm = X1 / np.sqrt(np.sum(X1**2))
3
4 X2= df.iloc[:, 1]
5 X2norm = X2 / np.sqrt(np.sum(X2**2))
6
7
8 relate2=np.inner(X1 , X2)
9 print(round(relate2))
10
11 relatenorm2=np.inner(X1norm, X2norm)
12 print(round(relatenorm2))
13
14
15
```

111  
1

### Covariance

$$\text{Cov}(x, y) = \frac{\sum(x - \bar{x})(y - \bar{y})}{m}$$



<b>X : green area</b>	<b>y: users in min</b>
10	35
6	24
18	46
2	14
14	42
13	40
7	27
22	56
20	50
3	20
$\bar{x} = 11.5$	$\bar{y} = 35.4$

```
counter=0
for i in range(len(x1)):
    counter+= (x1[i] - m1) * (x2[i] - m2) /10
Print(counter) : 84
```

```
Np.cov(x1 , x2)
```

```
Matrix.T @ matrix
```

$$\text{COV}(X,Y)=E[(X-E(X))(Y-E(Y))]$$

```
In [16]: # واحد نهاده گردی
1 X= np.array([[10],[6],[18],[2],[14],[13],[7],[22],[20],[3]])
2 y=np.array([[35],[24],[46],[14],[42],[40],[27],[56],[50],[20]])
3
4
5
6
7 cov = np.sum((X - X.mean()) * (y - y.mean())) / (len(X))
8 cov
9
10
11
12
```

Out[16]: 86.8

## PolyNomial Regression

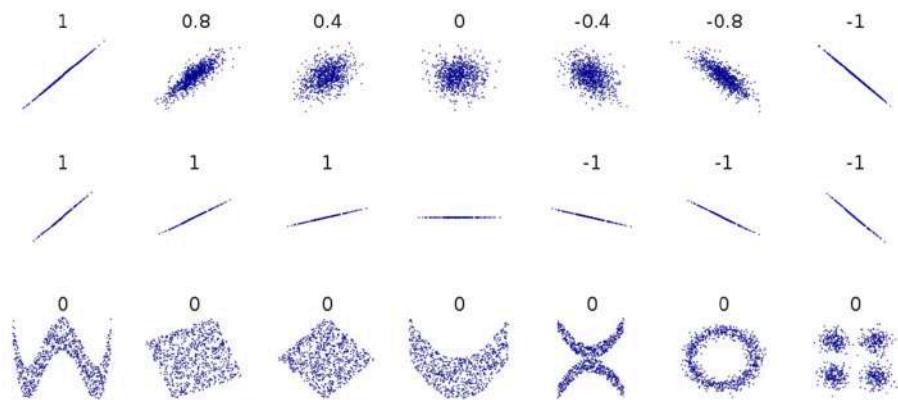
### Correlation

Parametric method

Pearson correlation

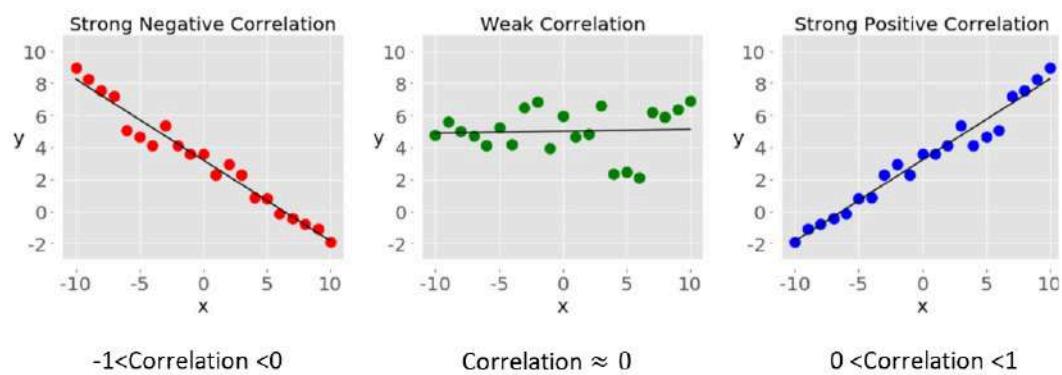
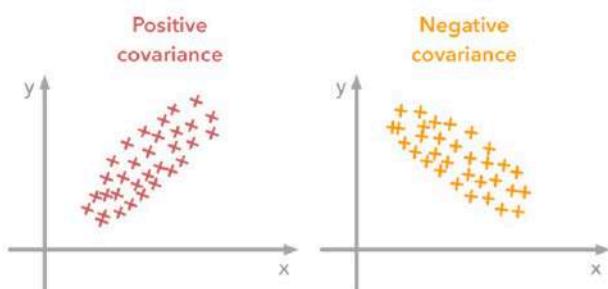
- Normal distribution

$$\frac{\text{covariance}(x_1, x_2)}{\text{std}(x_1) \text{ std}(x_2)}$$



## PolyNomial Regression

4



$$\rho(X, Y) = \text{corr}(X, Y) = \frac{(\mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))])}{\sqrt{(\mathbb{V}(X)\mathbb{V}(Y))}}$$

```
In [14]: M
1 import pandas as pd
2 concrete_data=pd.read_csv(r"D:\AI_Machinelearning\datasets\CONCRETE\regression\Concrete_Data .csv")
3
4
5 display(concrete_data.head())
6
7
8 print(type(concrete_data))
```

	X1	X2	X3	X4	X5	X6	X7	X8	y
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

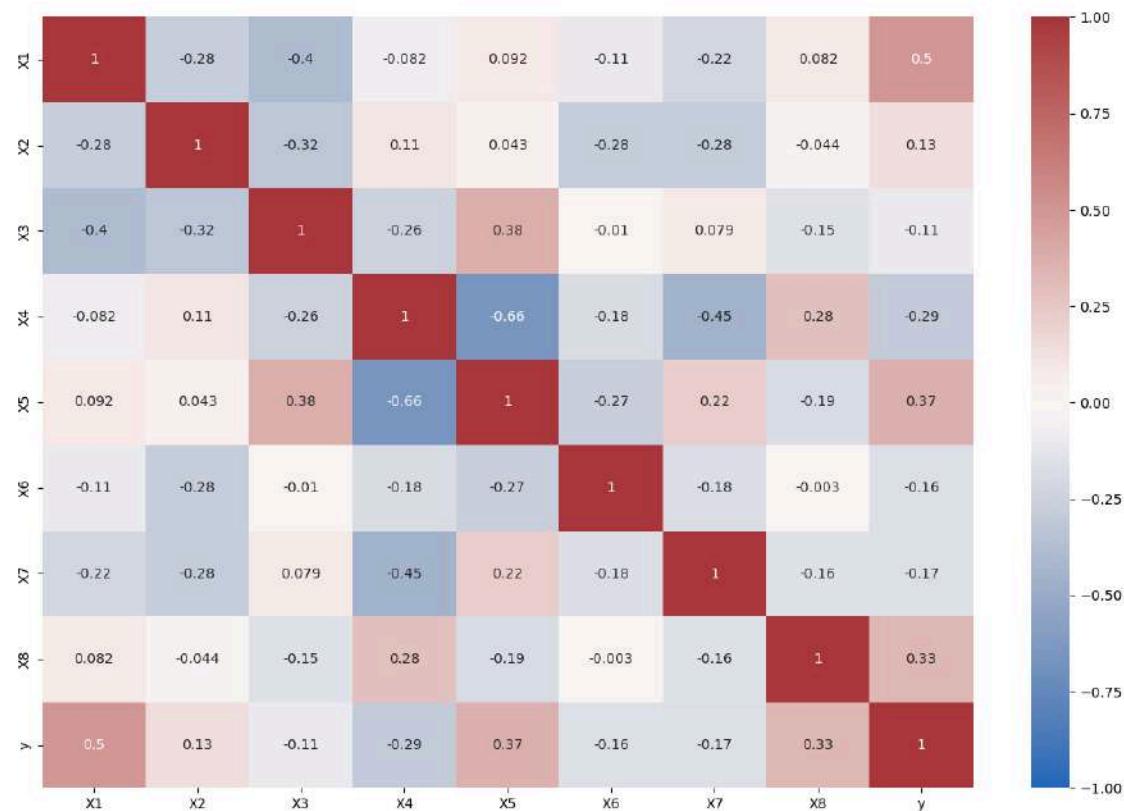
<class 'pandas.core.frame.DataFrame'>

```
1 x1= Cement (component 1) -- quantitative -- kg in a m3 mixture -- Input Variable # سیمان
2 x2=Blast Furnace Slag (component 2) -- quantitative -- kg in a m3 mixture -- Input Variable # سرباره کوره بلند
3 x3=Fly Ash (component 3) -- quantitative -- kg in a m3 mixture -- Input Variable # خاکستر بدی
4 x4=Water (component 4) -- quantitative -- kg in a m3 mixture -- Input Variable # آب
5 x5=Superplasticizer (component 5) -- quantitative -- kg in a m3 mixture -- Input Variable # فوق رون کننده
6 x6=Coarse Aggregate (component 6) -- quantitative -- kg in a m3 mixture -- Input Variable # سنگاله نیز گوش
7 x7=Fine Aggregate (component 7) -- quantitative -- kg in a m3 mixture -- Input Variable # سنگاله گرد گوش
8 x8=Age -- quantitative -- Day (1-365) -- Input Variable # سن
9
10 y=Concrete compressive strength -- quantitative -- MPa -- Output Variable # مقاومت فشاری بتن
11 مقارنت فشاری بتن # مقارنت فشاری بتن
```

```
In [15]: M
1 correlation1= concrete_data.corr()
2 correlation1
```

	X1	X2	X3	X4	X5	X6	X7	X8	y
X1	1.000000	-0.275216	-0.397467	-0.081587	0.092386	-0.109349	-0.222718	0.081946	0.497832
X2	-0.275216	1.000000	-0.323580	0.107252	0.043270	-0.283999	-0.281603	-0.044246	0.134829
X3	-0.397467	-0.323580	1.000000	-0.256984	0.377503	-0.009961	0.079108	-0.154371	-0.105755
X4	-0.081587	0.107252	-0.256984	1.000000	-0.657533	-0.182294	-0.450661	0.277618	-0.289633
X5	0.092386	0.043270	0.377503	-0.657533	1.000000	-0.265999	0.222691	-0.192700	0.366079
X6	-0.109349	-0.283999	-0.009961	-0.182294	-0.265999	1.000000	-0.178481	-0.003016	-0.164935
X7	-0.222718	-0.281603	0.079108	-0.450661	0.222691	-0.178481	1.000000	-0.156095	-0.167241
X8	0.081946	-0.044246	-0.154371	0.277618	-0.192700	-0.003016	-0.156095	1.000000	0.328873
y	0.497832	0.134829	-0.105755	-0.289633	0.366079	-0.164935	-0.167241	0.328873	1.000000

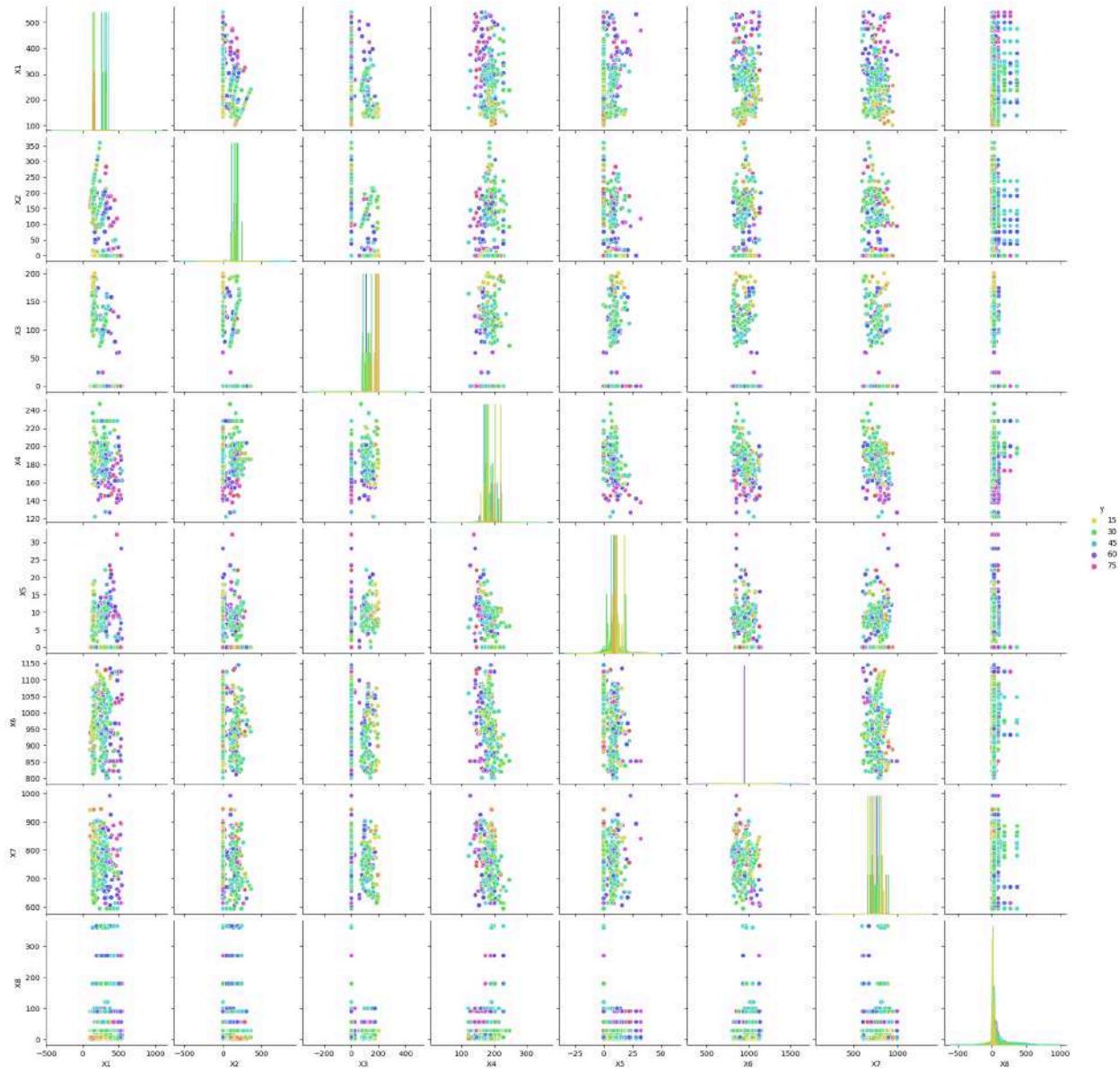
```
In [16]: M
1 import seaborn as sb
2 import matplotlib.pyplot as plt
3 plt.figure(figsize=(15,10))
4 sb.heatmap(correlation1 , vmin=-1 , vmax=1 , annot=True , cmap="vlag")
5 plt.show()
6
```



```
In [17]: 1 correlation1.X1
```

```
Out[17]: X1    1.000000
X2   -0.275216
X3   -0.397467
X4   -0.081587
X5    0.092386
X6   -0.109349
X7   -0.222718
X8    0.081946
y     0.497832
Name: X1, dtype: float64
```

```
In [18]: 1 sb.pairplot(concrete_data , hue="y" , palette="hls")
2 plt.show()
```



```
1 Specifically:
2 X1 Relative Compactness      تراکم نسبی -
3 X2 Surface Area             مساحت سطح #
4 X3 Wall Area                مساحت بیوار #
5 X4 Roof Area                مساحت سقف #
6 X5 Overall Height          ارتفاع کل #
7 X6 Orientation              چیت گیری ساختمان #
8 X7 Glazing Area             نافه شفاف #
9 X8 Glazing Area Distribution توزیع نافه شفاف #
10 y1 Heating Load            بار گرمایی #
11
```

```
In [19]: 1 dataenergy=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\sklearn\en.csv")
2 dataenergy.info()

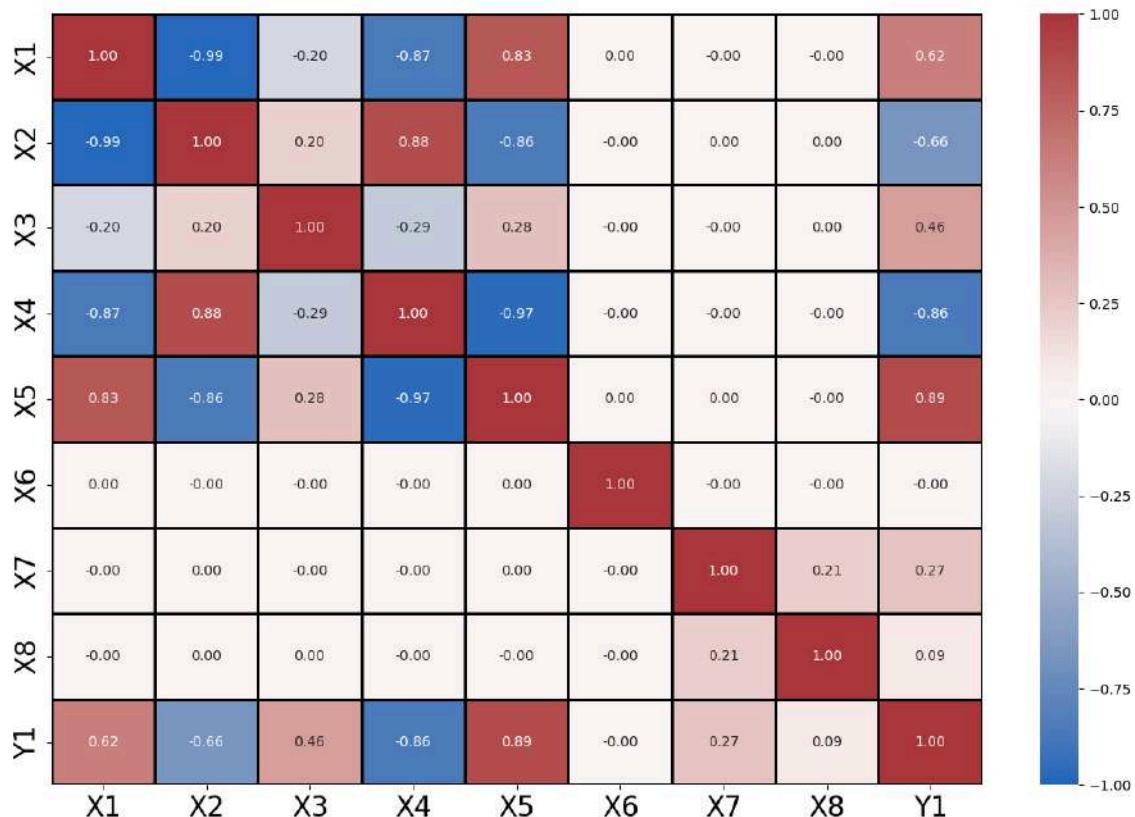
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   X1        768 non-null    float64
 1   X2        768 non-null    float64
 2   X3        768 non-null    float64
 3   X4        768 non-null    float64
 4   X5        768 non-null    float64
 5   X6        768 non-null    int64  
 6   X7        768 non-null    float64
 7   X8        768 non-null    int64  
 8   Y1        768 non-null    float64
dtypes: float64(7), int64(2)
memory usage: 54.1 KB
```

```
In [20]: 1 correlation2=dataenergy.corr()
2 correlation2
```

```
Out[20]:
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
X1	1.000000e+00	-9.919015e-01	-2.037817e-01	-8.688234e-01	8.277473e-01	4.678592e-17	-2.960552e-15	-7.107006e-16	0.622272
X2	-9.919015e-01	1.000000e+00	1.955016e-01	8.807195e-01	-8.581477e-01	-3.459372e-17	3.636925e-15	2.438409e-15	-0.658120
X3	-2.037817e-01	1.955016e-01	1.000000e+00	-2.923165e-01	2.809757e-01	-2.429498e-17	-8.567455e-17	2.067384e-16	0.455671
X4	-8.688234e-01	8.807195e-01	-2.923165e-01	1.000000e+00	-9.725122e-01	-5.830058e-17	-1.759011e-15	-1.078071e-15	-0.861828
X5	8.277473e-01	-8.581477e-01	2.809757e-01	-9.725122e-01	1.000000e+00	4.492205e-17	1.489134e-17	-2.920613e-17	0.889431
X6	4.678592e-17	-3.459372e-17	-2.429498e-17	-5.830058e-17	4.492205e-17	1.000000e+00	-9.406007e-16	-2.549352e-16	-0.002587
X7	-2.960552e-15	3.636925e-15	-8.567455e-17	-1.759011e-15	1.489134e-17	-9.406007e-16	1.000000e+00	2.129642e-01	0.269841
X8	-7.107006e-16	2.438409e-15	2.067384e-16	-1.078071e-15	-2.920613e-17	-2.549352e-16	2.129642e-01	1.000000e+00	0.087368
Y1	6.222722e-01	-6.581202e-01	4.556712e-01	-8.618283e-01	8.894307e-01	-2.586534e-03	2.698410e-01	8.736759e-02	1.000000

```
In [21]: 1 plt.figure(figsize=(15,10))
2 plt.xticks(fontsize=20)
3 plt.yticks(fontsize=20)
4 sb.heatmap(correlation2 , vmin=-1 , vmax=1 , annot=True , cmap="vlag" , linewidths=2 , fmt=".2f")
5 plt.show()
6
```



```
In [6]: 1 dataenergy2=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\data_energywb.csv")
2 dataenergy2.head()
```

```
NameError: name 'pd' is not defined
Cell In[6], line 1
----> 1 dataenergy2=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\data_energywb.csv")
2 dataenergy2.head()
```

```
NameError: name 'pd' is not defined
```

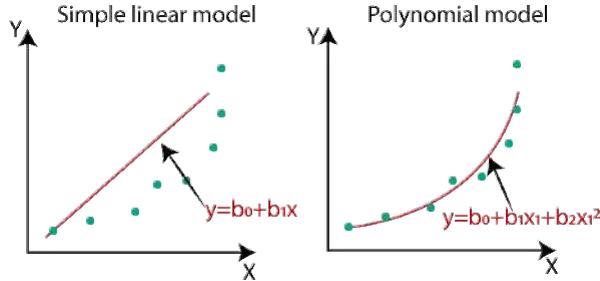
```
1 y      X6      X5      X4      X3      X2      X1
2 Heating Load  Insulation  MaximumCooling Temperature  Opening Percentage Overall Height  Area Without Opening  Area
3
```

```
In [5]: M
1 correlation3=dataenergy2.corr()
2 plt.figure(figsize=(15,10))
3 sb.heatmap(correlation3 , vmin=-1 , vmax=1 , annot=True ,cmap="RdGy", fmt=".2f" , linecolor="k")
4 plt.show()

-----
NameError                                 Traceback (most recent call last)
Cell In[5], line 1
----> 1 correlation3=dataenergy2.corr()
      2 plt.figure(figsize=(15,10))
      3 sb.heatmap(correlation3 , vmin=-1 , vmax=1 , annot=True ,cmap="RdGy", fmt=".2f" , linecolor="k")

NameError: name 'dataenergy2' is not defined
```

## Polynomial regression



Simple  
Linear  
Regression

$$y = b_0 + b_1 x_1$$

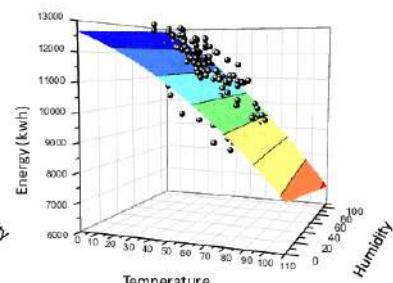
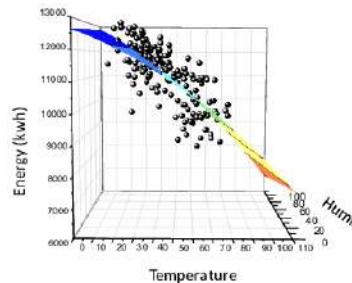
Multiple  
Linear  
Regression

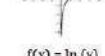
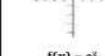
$$y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$$

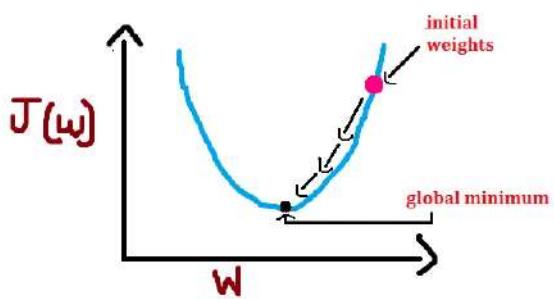
Polynomial  
Linear  
Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \dots + b_n x_1^n$$

$x$	$y$
3	3.45
5	6.9
6	8.79
8	12.91
10	17.48
12	22.49
15	30.85



Constant	Linear	Absolute Value	Quadratic
 $f(x) = c$	 $f(x) = x$	 $f(x) =  x $	 $f(x) = x^2$
Square Root	Cubic	Cube Root	Reciprocal/Inverse/Rational
 $f(x) = \sqrt{x}$	 $f(x) = x^3$	 $f(x) = \sqrt[3]{x}$	 $f(x) = \frac{1}{x}$
Rational	Logarithmic	Exponential	Greatest Integer (Step Function)
 $f(x) = \frac{1}{x^2}$	 $f(x) = \ln(x)$	 $f(x) = e^x$	 $f(x) = [[x]]$
Trigonometric Functions →			
	 $f(x) = \sin(x)$	 $f(x) = \cos(x)$	 $f(x) = \tan(x)$



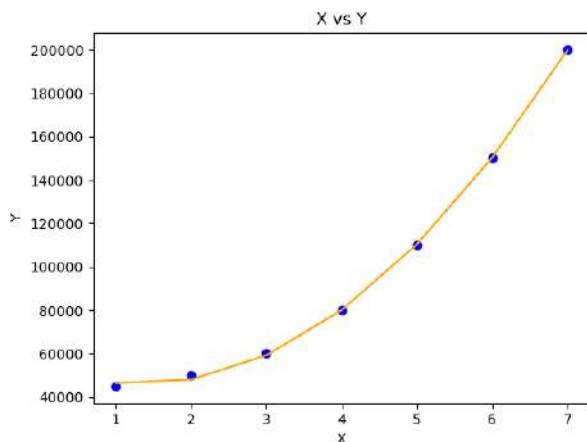
In [24]:

```
1 # Importing Libraries
2
3 import numpy as np
4 import math
5 import matplotlib.pyplot as plt
6
7
8 class PolynomialRegression:
9
10     def __init__(self, degree, learning_rate, iterations, stopping_threshold):
11         self.degree = degree
12         self.learning_rate = learning_rate
13         self.iterations = iterations
14         self.stopping_threshold = stopping_threshold
15
16     def normalize(self, X, method=None):
17         """
18             zscore:
19                 It is a special case of the normal distribution where
20                 the mean (average) is 0 and the standard deviation is 1.
21
22             maxabs:
23                 Rescale each feature between -1 and 1.
24
25             mimmax:
26                 The Min-Max Scaling (Normalization)
27                 technique works by transforming the original
28                 data into a new range, typically between 0 and 1.
29             """
30
31         if method is None or method == "zscore":
32             return (X - X.mean(axis=0)) / X.std(axis=0)
33         elif method == "maxabs":
34             return X / np.abs(X.max(axis=0))
35         elif method == "mimmax":
36             return (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
37
38     # Transform method
39     def transform(self, X):
40         X_norm = self.normalize(X)
41         num_samples, num_features = X_norm.shape
42         X_transform = np.empty((num_samples, 0))
43         for i in range(1, self.degree + 1):
44             X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
45         return X_transform
46
47     # model training
48     def fit(self, X, y):
49         Xtansnorm = self.transform(X)
50         m, n = Xtansnorm.shape
51
52         # weight initialization
53         current_weight = np.random.rand(n ,1)
54         current_bias = np.zeros([1,1])
55
56         costs = []
57         weights = []
58         biases = []
59         previous_cost = None
60
61         for i in range(self.iterations):
62             # Making predictions
63             prediction = (Xtansnorm @ current_weight) + current_bias
64             # Calculating the current cost
65             current_cost = np.sum((1 / (2 * m)) * np.square(prediction - y))
66
67             # If the change in cost is less than or equal to
68             # stopping_threshold we stop the gradient descent
69             if previous_cost is not None and abs(previous_cost - current_cost) <= self.stopping_threshold:
70                 break
71
72             previous_cost = current_cost
73             costs.append(current_cost)
74             weights.append(current_weight)
75             biases.append(current_bias)
76
77             # Calculating the gradients
78             weight_derivative = Xtansnorm.T @ (prediction - y)/m
79             bias_derivative = np.sum(prediction - y)/m
80
81             # Updating weights and bias
82             current_weight -= self.learning_rate * weight_derivative
83             current_bias -= self.learning_rate * bias_derivative
84
85         return costs, weights[-1], biases[-1]
86
87
88     # predict
89     # predict
90     def predict(self, X , y):
91         _, predicted_weights, predicted_bias = self.fit(X, y) # Assuming y is defined somewhere
92         X_transform = self.transform(X)
93         return (X_transform @ predicted_weights) + predicted_bias
94
95
96
```

## Example1

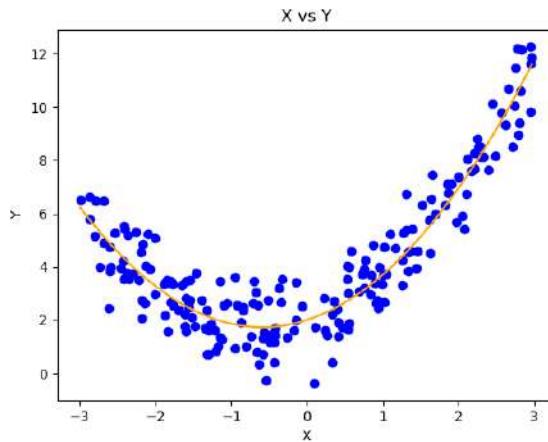
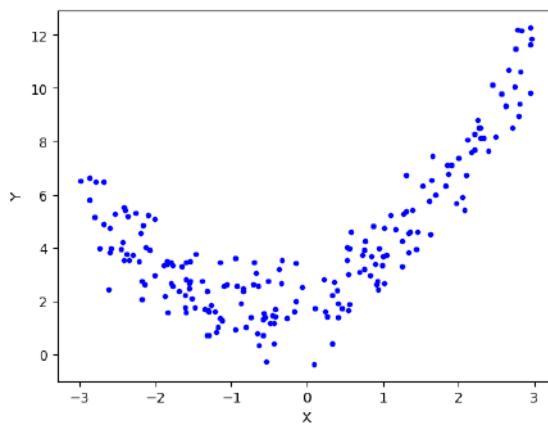
In [25]:

```
1 #Driver code
2
3 def main() :
4     # Create dataset
5     X = np.array( [ [1], [2], [3], [4], [5], [6], [7] ] )
6     Y = np.array( [[45000], [50000], [60000], [80000], [110000], [150000], [200000]] )
7
8     # model training
9
10    model = PolynomialRegression(degree=2, learning_rate=0.01, iterations=1500, stopping_threshold=1e-6)
11
12    model.fit( X, Y )
13    # Prediction on training set
14    Y_pred = model.predict(X,Y)
15    # Visualization
16    plt.scatter( X, Y, color = 'blue' )
17    plt.plot( X, Y_pred, color = 'orange' )
18    plt.title( 'X vs Y' )
19    plt.xlabel( 'X' )
20    plt.ylabel( 'Y' )
21    plt.show()
22
23
24
25 if __name__ == "__main__":
26     main()
```



## Example2

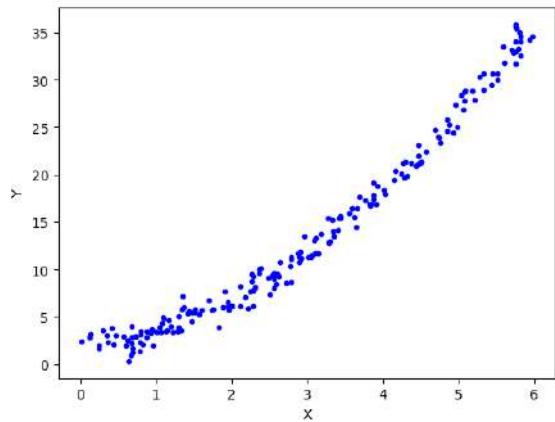
```
In [26]: M
1 def main() :
2
3     # Create dataset
4     X = 6 * np.random.rand(200, 1) - 3
5     y = 0.8 * X**2 + 0.9*X + 2 + np.random.randn(200, 1)
6
7
8     # Get indices that would sort X
9     sorted_indices = np.argsort(X[:, 0])
10
11    # Sort X and y based on the sorted indices
12    X_sorted = X[sorted_indices]
13    y_sorted = y[sorted_indices]
14
15
16    plt.plot(X, y, 'b.')
17    plt.xlabel("X")
18    plt.ylabel("Y")
19    plt.show()
20
21
22
23    # model training
24
25    model = PolynomialRegression(degree=2, learning_rate=0.01, iterations=1000, stopping_threshold=1e-6)
26
27    model.fit( X_sorted, y_sorted )
28    # Prediction on training set
29    Y_pred = model.predict(X_sorted,y_sorted)
30
31    # Visualization
32    plt.scatter( X_sorted, y_sorted, color = 'blue' )
33    plt.plot( X_sorted, Y_pred, color = 'orange' )
34    plt.title( 'X vs Y' )
35    plt.xlabel( 'X' )
36    plt.ylabel( 'Y' )
37    plt.show()
38
39
40 if __name__ == "__main__":
41     main()
```



### Example3

In [27]:

```
1 # Create dataset
2 X = 6 * np.random.rand(200, 1)
3 y = 0.8 * X**2 + 0.9*X + 2 + np.random.randn(200, 1)
4
5
6 plt.plot(X, y, 'b.')
7 plt.xlabel("X")
8 plt.ylabel("Y")
9 plt.show()
```

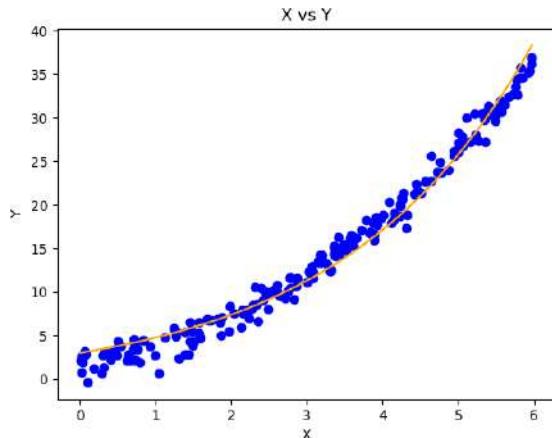
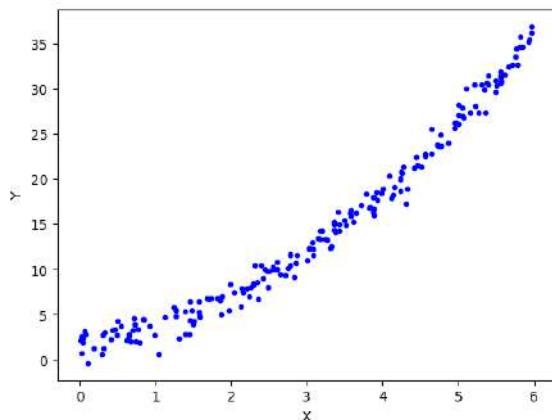


In [28]:

```
 1 # Importing Libraries
 2
 3 import numpy as np
 4 import math
 5 import matplotlib.pyplot as plt
 6 from scipy.special import iv
 7
 8 class PolynomialRegression:
 9
10     def __init__(self, degree, learning_rate, iterations, stopping_threshold):
11         self.degree = degree
12         self.learning_rate = learning_rate
13         self.iterations = iterations
14         self.stopping_threshold = stopping_threshold
15
16     def normalize(self, X, method=None):
17         """
18             zscore:
19                 It is a special case of the normal distribution where
20                 the mean (average) is 0 and the standard deviation is 1.
21
22             maxabs:
23                 Rescale each feature between -1 and 1.
24
25             mimmax:
26                 The Min-Max Scaling (Normalization)
27                 technique works by transforming the original
28                 data into a new range, typically between 0 and 1.
29             """
30
31         if method is None or method == "zscore":
32             return (X - X.mean(axis=0)) / X.std(axis=0)
33         elif method == "maxabs":
34             return X / np.abs(X.max(axis=0))
35         elif method == "mimmax":
36             return (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
37
38     # Transform method
39     def transform(self, X):
40         X_norm = self.normalize(X)
41         num_samples, num_features = X_norm.shape
42         X_transform = np.ones((num_samples, 1))
43         for i in range(1, self.degree + 1):
44             X_transform = np.concatenate((X_transform, np.power(i, X_norm)), axis=1)
45         return X_transform
46
47     # model training
48     def fit(self, X, y):
49         Xtansnorm = self.transform(X)
50         m, n = Xtansnorm.shape
51
52         # weight initialization
53         current_weight = np.random.rand(n ,1)
54         current_bias = np.zeros([1,1])
55
56         costs = []
57         weights = []
58         biases = []
59         previous_cost = None
60
61         for i in range(self.iterations):
62             # Making predictions
63             prediction = (Xtansnorm @ current_weight) + current_bias
64             # Calculating the current cost
65             current_cost = np.sum((1 / (2 * m)) * np.square(prediction - y))
66
67             # If the change in cost is less than or equal to
68             # stopping_threshold we stop the gradient descent
69             if previous_cost is not None and abs(previous_cost - current_cost) <= self.stopping_threshold:
70                 break
71
72             previous_cost = current_cost
73             costs.append(current_cost)
74             weights.append(current_weight)
75             biases.append(current_bias)
76
77             # Calculating the gradients
78             weight_derivative = Xtansnorm.T @ (prediction - y)/m
79             bias_derivative = np.sum(prediction - y)/m
80
81             # Updating weights and bias
82             current_weight -= self.learning_rate * weight_derivative
83             current_bias -= self.learning_rate * bias_derivative
84
85         return costs, weights[-1], biases[-1]
86
87
88     # predict
89     # predict
90     def predict(self, X , y):
91         _, predicted_weights, predicted_bias = self.fit(X, y) # Assuming y is defined somewhere
92         X_transform = self.transform(X)
93         return (X_transform @ predicted_weights) + predicted_bias
94
95
```

In [29]:

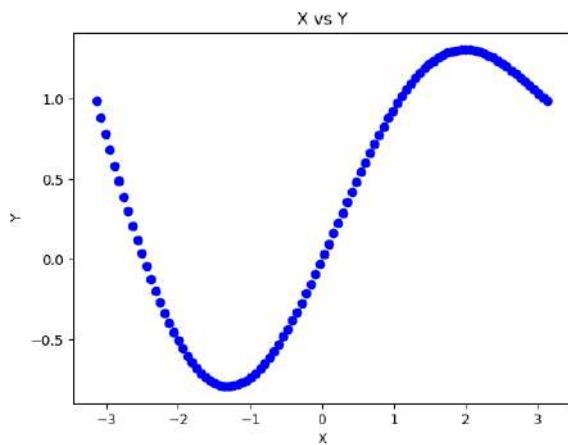
```
1 def main() :
2
3     # Create dataset
4     X = 6 * np.random.rand(200, 1)
5     y = 0.8 * X**2 + 0.9*X + 2 + np.random.randn(200, 1)
6
7
8     # Get indices that would sort X
9     sorted_indices = np.argsort(X[:, 0])
10
11    # Sort X and y based on the sorted indices
12    X_sorted = X[sorted_indices]
13    y_sorted = y[sorted_indices]
14
15
16    plt.plot(X, y, 'b.')
17    plt.xlabel("X")
18    plt.ylabel("Y")
19    plt.show()
20
21
22
23    # model training
24
25    model = PolynomialRegression(degree=2, learning_rate=0.01, iterations=1000, stopping_threshold=1e-6)
26    model.fit( X_sorted, y_sorted )
27    # Prediction on training set
28    Y_pred = model.predict(X_sorted,y_sorted)
29    # Visualization
30    plt.scatter( X_sorted, y_sorted, color = 'blue' )
31    plt.plot( X_sorted, Y_pred, color = 'orange' )
32    plt.title( 'X vs Y' )
33    plt.xlabel( 'X' )
34    plt.ylabel( 'Y' )
35    plt.show()
36
37
38
39 if __name__ == "__main__":
40     main()
```



#### Example4

In [30]:

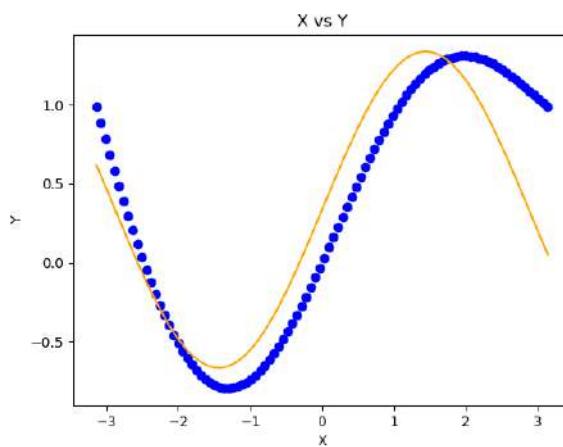
```
1 # Generate x values using linspace
2 x = np.array([np.linspace(-np.pi, np.pi,100)]).T
3 # Generate y values using a polynomial function with a sin term
4 y = np.array(np.sin(x) + (0.1 * np.power(x, 2)))
5
6 # Visualization
7 plt.scatter( x, y, color = 'blue' )
8 plt.title( 'X vs Y' )
9 plt.xlabel( 'X' )
10 plt.ylabel( 'Y' )
11 plt.show()
12
```



In [31]:

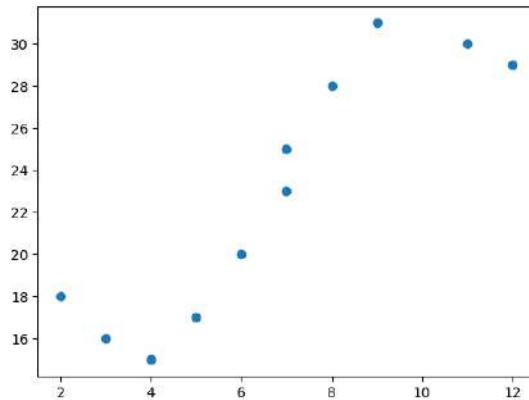
```
1 # Importing Libraries
2
3 import numpy as np
4 import math
5 import matplotlib.pyplot as plt
6 from scipy.special import iv
7
8 class PolynomialRegression:
9
10     def __init__(self, degree, learning_rate, iterations, stopping_threshold):
11         self.degree = degree
12         self.learning_rate = learning_rate
13         self.iterations = iterations
14         self.stopping_threshold = stopping_threshold
15
16     def normalize(self, X, method=None):
17         """
18             zscore:
19                 It is a special case of the normal distribution where
20                 the mean (average) is 0 and the standard deviation is 1.
21
22             maxabs:
23                 Rescale each feature between -1 and 1.
24
25             mimmax:
26                 The Min-Max Scaling (Normalization)
27                 technique works by transforming the original
28                 data into a new range, typically between 0 and 1.
29             """
30
31         if method is None or method == "zscore":
32             return (X - X.mean(axis=0)) / X.std(axis=0)
33         elif method == "maxabs":
34             return X / np.abs(X.max(axis=0))
35         elif method == "mimmax":
36             return (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
37
38     # Transform method
39     def transform(self, X):
40         X_norm = self.normalize(X)
41         num_samples, num_features = X_norm.shape
42         X_transform = np.empty((num_samples, 0))
43         for i in range(1, self.degree + 1):
44             X_transform = np.concatenate((X_transform, np.sin(2*i*X_norm)), axis=1)
45         return X_transform
46
47     # model training
48     def fit(self, X, y):
49         Xtansnorm = self.transform(X)
50         m, n = Xtansnorm.shape
51
52         # weight initialization
53         current_weight = np.random.rand(n ,1)
54         current_bias = np.zeros([1,1])
55
56         costs = []
57         weights = []
58         biases = []
59         previous_cost = None
60
61         for i in range(self.iterations):
62             # Making predictions
63             prediction = (Xtansnorm @ current_weight) + current_bias
64             # Calculating the current cost
65             current_cost = np.sum((1 / (2 * m)) * np.square(prediction - y))
66
67             # If the change in cost is less than or equal to
68             # stopping_threshold we stop the gradient descent
69             if previous_cost is not None and abs(previous_cost - current_cost) <= self.stopping_threshold:
70                 break
71
72             previous_cost = current_cost
73             costs.append(current_cost)
74             weights.append(current_weight)
75             biases.append(current_bias)
76
77             # Calculating the gradients
78             weight_derivative = Xtansnorm.T @ (prediction - y)/m
79             bias_derivative = np.sum(prediction - y)/m
80
81             # Updating weights and bias
82             current_weight -= self.learning_rate * weight_derivative
83             current_bias -= self.learning_rate * bias_derivative
84
85         return costs, weights[-1], biases[-1]
86
87
88     # predict
89     # predict
90     def predict(self, X , y):
91         _, predicted_weights, predicted_bias = self.fit(X, y) # Assuming y is defined somewhere
92         X_transform = self.transform(X)
93         return (X_transform @ predicted_weights) + predicted_bias
94
```

```
In [32]: M
1 def main() :
2
3     # Create dataset
4
5     # Generate x values using linspace
6     x = np.array([np.linspace(-np.pi, np.pi, 100)]).T
7
8     # Generate y values using a polynomial function with a sin term
9     y = np.array(np.sin(x) + (0.1 * np.power(x, 2)))
10
11    # model training
12    model = PolynomialRegression(degree=1, learning_rate=0.01, iterations=1000, stopping_threshold=1e-6)
13    model.fit(x, y)
14
15    # Prediction on training set
16    Y_pred = model.predict(x, y)
17
18    # Visualization
19    plt.scatter( x, y, color = 'blue' )
20    plt.plot(x, Y_pred, color = 'orange' )
21    plt.title( 'X vs Y' )
22    plt.xlabel( 'X' )
23    plt.ylabel( 'Y' )
24    plt.show()
25
26 if __name__ == "__main__":
27     main()
28
29
```



## sklearn

```
In [33]: M
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 #define predictor and response variables
5 x = np.array([2, 3, 4, 5, 6, 7, 7, 8, 9, 11, 12])
6 y = np.array([18, 16, 15, 17, 20, 23, 25, 28, 31, 30, 29])
7
8 #create scatterplot to visualize relationship between x and y
9 plt.scatter(x, y)
10 plt.show()
```

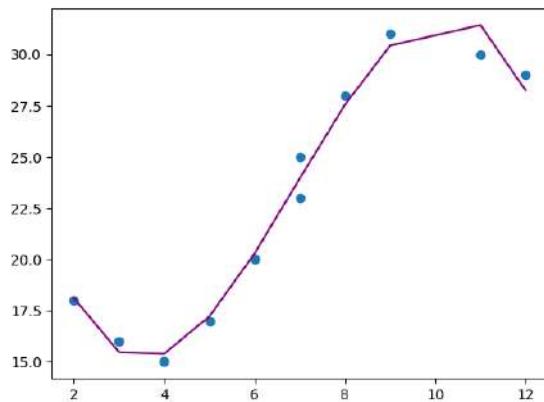


```
In [34]: 
1 import numpy as np
2 from sklearn.preprocessing import PolynomialFeatures
3 from sklearn.linear_model import LinearRegression
4
5
6
7 poly = PolynomialFeatures(degree=3 , include_bias=True)
8 poly_features = poly.fit_transform(x.reshape(-1, 1))
9
10 #poly = PolynomialFeatures(interaction_only=True)
11 #poly.fit_transform(X)
12
13
14 #fit polynomial regression model
15 model = LinearRegression()
16 model.fit(poly_features, y)
17
18
19 print(model.intercept_, model.coef_)
20
21
22
```

33.62640037531699 [ 0. -11.83877127 2.25592957 -0.10889554]

```
In [35]: 
1 #use model to make predictions on response variable
2 y_predicted = model.predict(poly_features)
3
4 #create scatterplot of x vs. y
5 plt.scatter(x, y)
6
7 #add line to show fitted polynomial regression model
8 plt.plot(x, y_predicted, color='purple')
```

Out[35]: [<matplotlib.lines.Line2D at 0x2506d71a0b0>]



## Example 2

```
In [36]: 
1 import pandas as pd
2 dataenergy=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\sklearn\en.csv")
3 dataenergy.head()
```

Out[36]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84

```
In [37]: 
1 from sklearn.preprocessing import PolynomialFeatures
2 from sklearn.linear_model import LinearRegression
3 from sklearn.model_selection import train_test_split
4
5 X=dataenergy.iloc[:, :-1]
6 y=dataenergy.iloc[:, -1]
7
8
9
10
11 poly = PolynomialFeatures(degree = 3)
12 X = poly.fit_transform(X)
13
14
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
16
17 model = LinearRegression()
18 model.fit(X_train, y_train)
19 expected_y = y_test
20 predicted_y = model.predict(X_test)
21
22
23 print(model.score(X_train,y_train)*100)
24 print(model.score(X_test,y_test)*100)
25
26
27
```

99.42489511803178  
99.27169979762127

## Save the best model

```
In [38]: 1 with open(r"D:\text3.txt", "w") as f5:
2     f5.write("this is fun")
3
4 with open(r"D:\text3.txt", "r") as f6:
5     for line in f6:
6         print(line)
7
8
```

this is fun

```
In [39]: 1 import pandas as pd
2
3 dataenergy=pd.read_csv(r"D:\AI_Machinlearning\datasets\ENERGY\sklearn\en.csv")
4 dataenergy.head()
5
```

Out[39]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84

```
In [40]: 1 import pickle
2 from sklearn.preprocessing import PolynomialFeatures
3 from sklearn.linear_model import LinearRegression
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import normalize
6
7
8 X = dataenergy.iloc[:, :-1]
9 y = dataenergy.iloc[:, -1]
10 X_normalized = normalize(X)
11
12
13 poly = PolynomialFeatures(degree=2)
14 X_poly = poly.fit_transform(X_normalized)
15
16
17 best_acc = 0
18 for i in range(3):
19     X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.25, random_state=12)
20     model = LinearRegression()
21     model.fit(X_train, y_train)
22     acc = model.score(X_train, y_train)
23     if acc > best_acc:
24         best_acc = acc
25         with open(r"D:\bestmodel.pickle", "wb") as f:
26             pickle.dump(model, f)
27
28
29
30
31
32
33
34
35
```

```
In [15]: 1 # Load the best model
2 with open(r"D:\bestmodel.pickle", "rb") as f:
3     best_model = pickle.load(f)
4
5
6 #print(best_model.intercept_)
7 # print("*" * 20)
8 #print(best_model.coef_)
9 # print("*" * 20)
10 #print(best_model.predict(X_test))
11 #print(best_model.score(X_test,y_test))
```

0.9851272372442831

Type Markdown and LaTeX:

In [ ]:

1

# Machine learning

Locally weighted regression

Morteza khorsand



## Locally weighted regression

### ■ Parametric model

In a parametric model, you know exactly which model you are going to fit in with the data, for example, linear regression line.

- fixed set of parameters
- To predict new data we do not need historical dataset.

### ■ Non-parametric model

Nonparametric machine learning algorithms are those which do not make specific assumptions about the type of the mapping function.

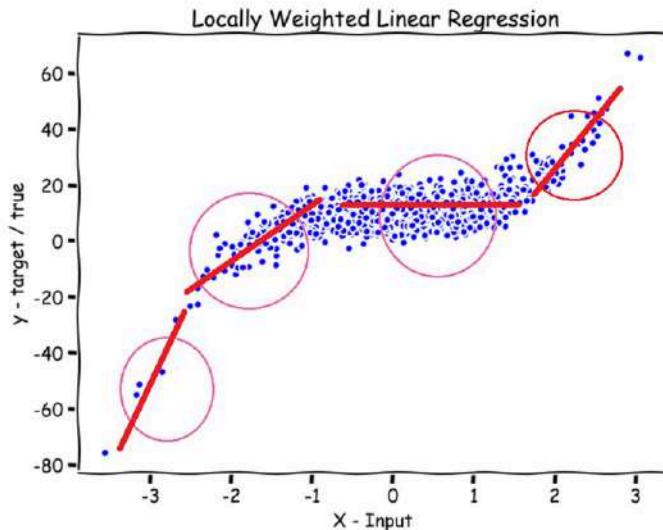
Number of parameters increased in order to number of samples

- To predict new samples we need historical dataset.

## Locally weighted regression

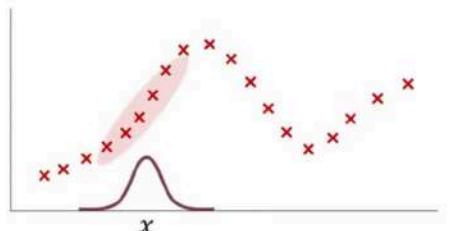
### Locally weighted regression (LWR)

a memory-based method that performs a regression around a point of interest using only training data that are "local" to that point.

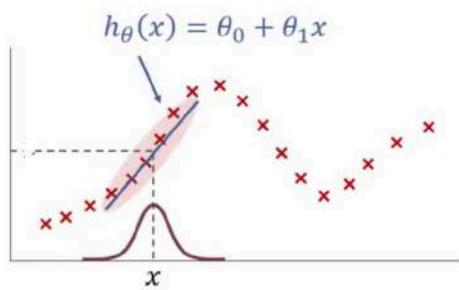


## Locally weighted regression

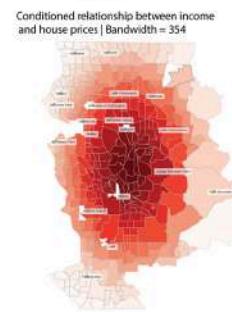
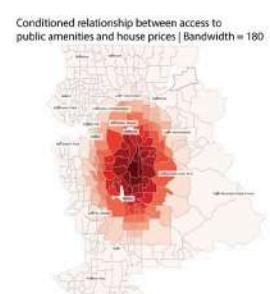
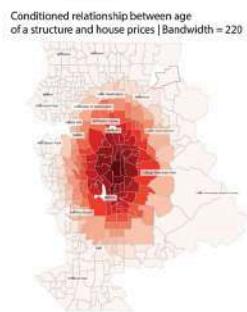
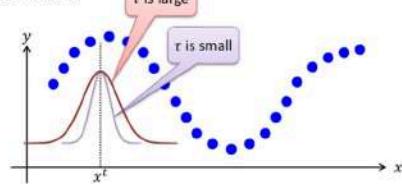
### Locally weighted regression



$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$



- Bandwidth parameter  $\tau$ :

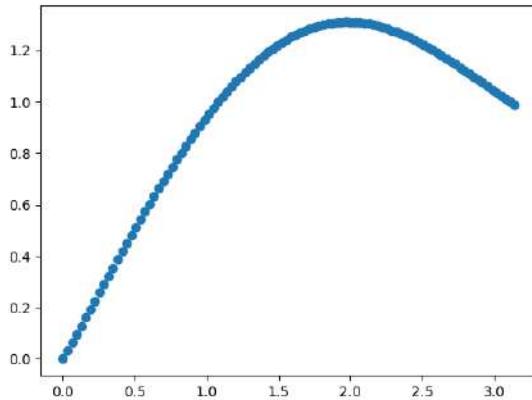


```
In [1]: import matplotlib.pyplot as plt
import numpy as np

X = np.array([np.linspace(0, np.pi, 100)]).T
y = np.array(np.sin(X) + (0.1 * np.power(X, 2)))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
plt.scatter(X, y)
plt.show()
```



```
In [2]: import matplotlib.pyplot as plt
import numpy as np

def kernel(Inputs, X, tau):
    m, n = X.shape
    weights = np.eye(m)
    for i in range(m):
        dis = Inputs - X[i]
        weights[i, i] = np.exp(-(dis @ dis.T) / (2 * tau ** 2))
    return weights

def fit(X, y, Inputs, tau):
    m, n = X.shape
    weights = kernel(Inputs, X, tau)
    theta = (np.linalg.inv(X.T @ weights @ X)) @ X.T @ weights @ y
    return theta

def predict(X, y, Inputs, tau):
    predictions = []
    for sample in Inputs:
        theta = fit(X, y, np.array([sample]), tau)
        prediction = np.dot(sample, theta)
        predictions.append(prediction)
    return predictions

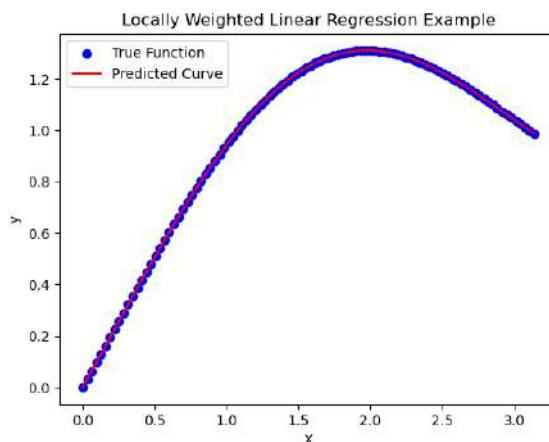
# Choose points for prediction
Inputs = np.linspace(0, np.pi, 100)

# Set bandwidth parameter
tau = 0.01

# Predict output for the chosen points
predictions = predict(X, y, Inputs, tau)

# Flatten predictions to remove extra dimensions
predictions = np.squeeze(predictions)

# Plot the true function and the predicted curve
plt.scatter(X, y, color='blue', label='True Function')
plt.plot(Inputs, predictions, color='red', label='Predicted Curve')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Locally Weighted Linear Regression Example')
plt.legend()
plt.show()
```



```
In [24]: import pandas as pd
data=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\energy.csv")

X=np.array(data.drop(["Y"] , axis= 1))
y=np.array(data["Y"])

y=y.reshape((768,1))

from sklearn.model_selection import train_test_split

X_train , X_test , y_train , y_test= train_test_split(X , y)
```

```
In [28]: # Choose points for prediction
Inputs = X_test

# Set bandwidth parameter
tau = 100

# Predict output for the chosen points
predictions = predict(X_train, y_train, X_test[0], tau)
```

```

Out[28]: [array([[ -5.63670689e+03],
   [ 3.07381495e+03],
   [ 2.28822281e+00],
   [ 2.89793595e+00],
   [ 5.57568983e+00],
   [ 5.48136969e+00],
   [-3.01529113e-02],
   [ 3.11790049e-01],
   [ 3.3673412e-01]]),
array([[-2.32217177e-03],
   [ 1.23900939e-03],
   [ 1.33030507e-00],
   [ 8.39191800e-01],
   [ 1.42275514e+00],
   [ 4.49489125e+00],
   [-2.47218341e-02],
   [ 2.55672076e+01],
   [ 2.76112477e-01]]),
array([[ 3.94132892e-04],
   [-4.62424896e+04],
   [ 4.57950812e-01],
   [-6.15984325e+01],
   [-1.49480209e-02],
   [ 4.02868731e-03],
   [ 5.06488578e-01],
   [ 1.00640191e+04],
   [ 4.83586713e-01]]),
array([-8470.18714428],
   [-1804.92451891],
   [ -45.1978867 ],
   [ 74.61136641],
   [ 98.08840924],
   [ 1544.52924951],
   [ -25.05239058],
   [ 8727.81316238],
   [ 82.79788702]]),
array([-1.08547562e-05],
   [ 5.38928996e-04],
   [ 2.84025860e+01],
   [ 8.00982794e-01],
   [ 1.22875761e+02],
   [ 7.69994506e+02],
   [-5.75359254e+00],
   [ 4.47293521e-03],
   [ 4.60166358e-01]]),
array([[ 1.55563522e-04],
   [-8.77966681e-03],
   [-1.23541404e+01],
   [ -8.60031669e-01],
   [-3.71030113e+00],
   [ 3.83210036e+01],
   [-2.12123206e-01],
   [ 2.18130017e+02],
   [ 2.35240168e+00]]),
array([-1.70851211e+04],
   [ 9.30176738e+03],
   [ 1.61140791e+01],
   [-5.97029727e-01],
   [-2.06369123e+00],
   [ 1.09604651e+01],
   [-6.03543424e-02],
   [ 6.23522485e+01],
   [ 6.73193818e-01]]),
array([[ 1.15405882e-02],
   [-6.64067243e-01],
   [-1.31393006e-02],
   [-8.27292622e-02],
   [-1.91896334e-01],
   [ 5.48238819e-01],
   [-3.01313478e-03],
   [ 3.11815814e+00],
   [ 3.36798874e-02]]),
array([[ 2.41751879e-04],
   [-1.34190437e+04],
   [-2.29815388e+01],
   [ 1.95578833e+00],
   [ 2.65272610e+00],
   [ 2.19117329e+01],
   [-1.20906603e-01],
   [ 1.24681222e+02],
   [ 1.34553006e+00]]])

```

In [ ]:

# Machine learning

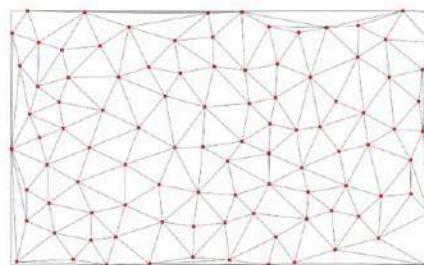
KNeighborsClassifier

Morteza khorsand

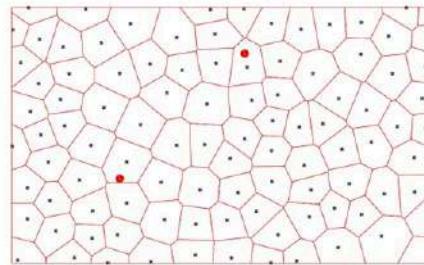
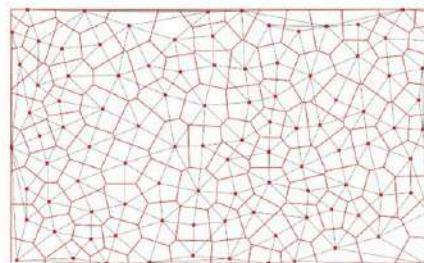


## KNN

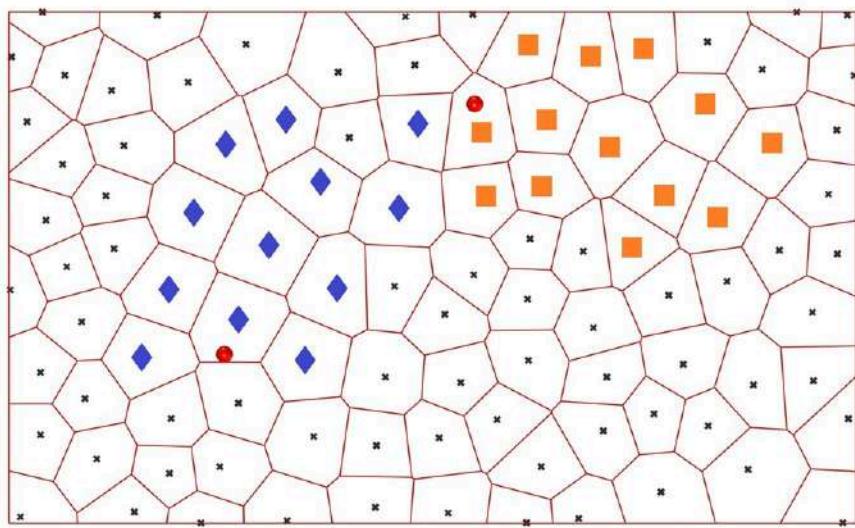
- The Delaunay triangulation



- Voronoi diagram

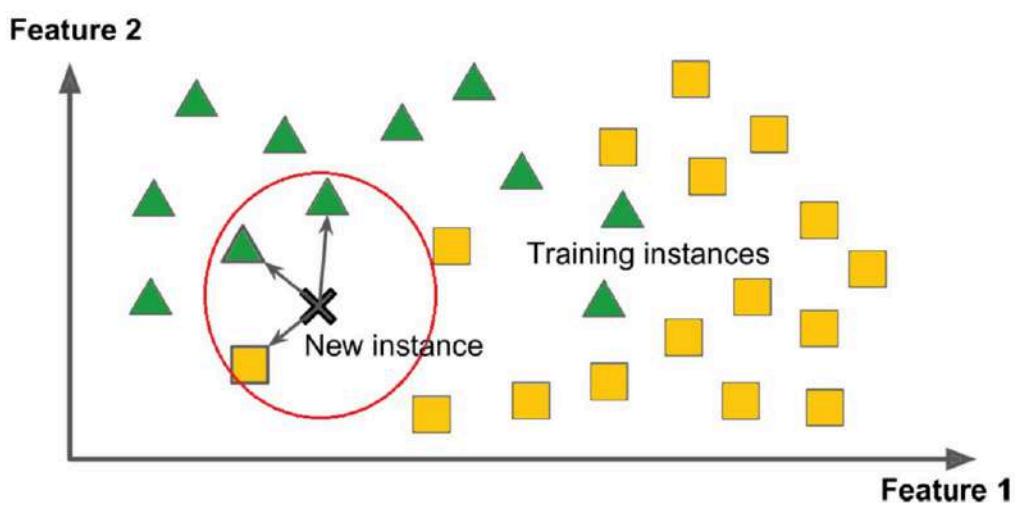


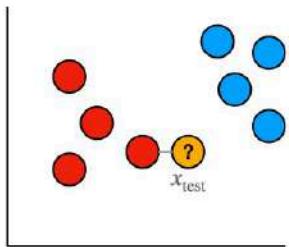
## KNN



## KNN

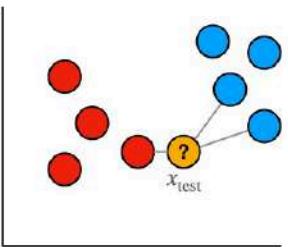
■ KNN





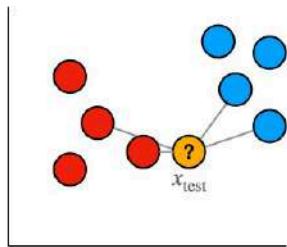
$k = 1$

Nearest point is **red**, so  
 $x_{\text{test}}$  classified as **red**



$k = 3$

Nearest points are {**red**,  
**blue**, **blue**} so  $x_{\text{test}}$   
classified as **blue**



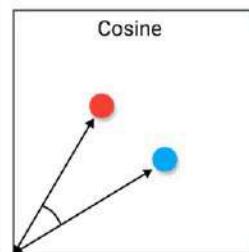
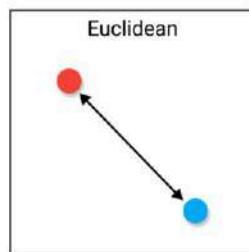
$k = 4$

Nearest points are {**red**,  
**red**, **blue**, **blue**} so  
classification of  $x_{\text{test}}$  is  
not properly defined

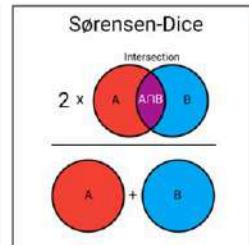
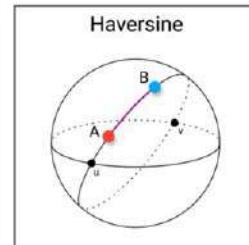
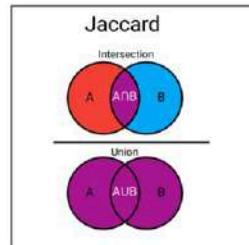
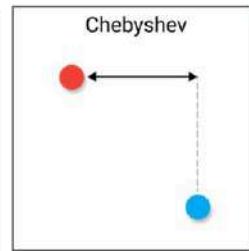
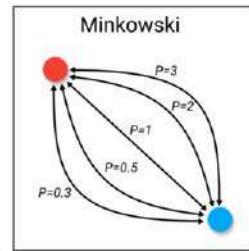
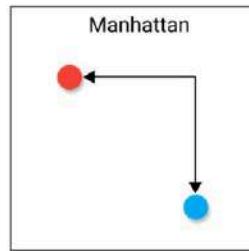
## KNN

### Distance

1. Euclidean distance can be used if features are similar or we want to find the distance between two data points.
2. When we have high dimensions then Manhattan is preferred.
3. In the case of Categorical features Hamming Distance is used.
4. Cosine similarity is used when we are concerned about the direction of vectors, not magnitude.

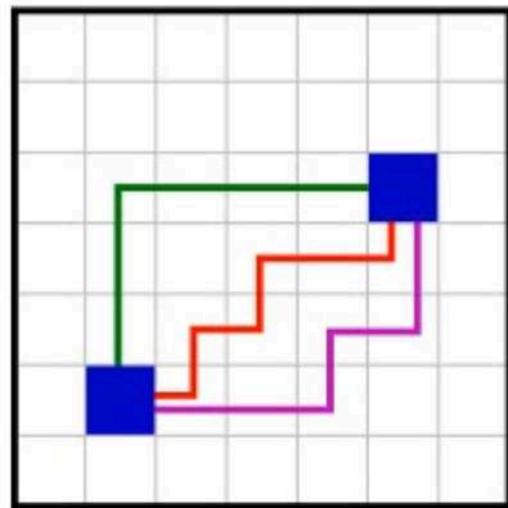


Hamming	
A	1 0 1 1 0 0
B	1 1 1 0 0 0



## KNN

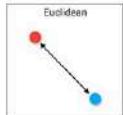
### Manhattan Distance(L1)



## KNN

### Euclidean Distance(L2)

$$\|A - B\|_2 = \sqrt{\sum_{i=1}^k (A_i - B_i)^2}$$



$$\|X\|_2 = (\sum_{i=1}^n |x_i|^2)^{\frac{1}{2}} = \sqrt{x_1^2 + x_2^2 + x_3^2 + \dots + x_n^2}$$

### Manhattan Distance(L1) - City Block Distance or Taxicab Distance

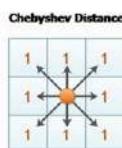
$$\|(A) - (B)\|_1 = \sum_{i=1}^k |A_i - B_i|$$



$$\|X\|_1 = (\sum_{i=1}^n |x_i|) = |x_1| + |x_2| + |x_3| + \dots + |x_n|$$

### Chebyshev Distance - Chessboard Distance or L infinity Distance or Maximum value distance.

$$\|(A) - (B)\|_\infty = \max\{|A_i, B_i|\}$$



$$\|X\|_\infty = \max|x_i| \quad x = \left\| \begin{bmatrix} 2 \\ -1 \\ 4 \\ -2.5 \end{bmatrix} \right\|_\infty = 4$$

### Minkowski Distance - p-norm vector

$$d((A) - (B)) = \sqrt[p]{\sum_{i=1}^k |A_i - B_i|^p}$$

$$\|X\|_p = (\sum_{i=1}^n |x_i|^p)^{\frac{1}{p}} = \sqrt[p]{|x_1|^p + |x_2|^p + |x_3|^p + \dots + |x_n|^p}$$

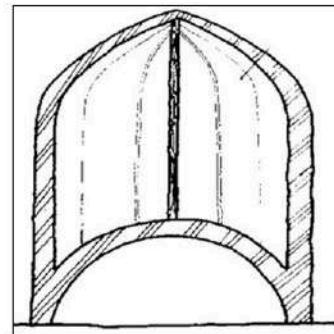
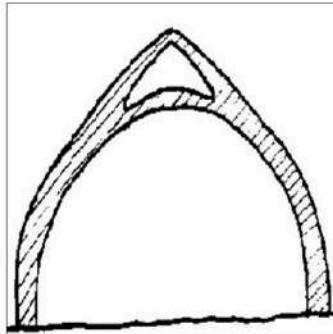
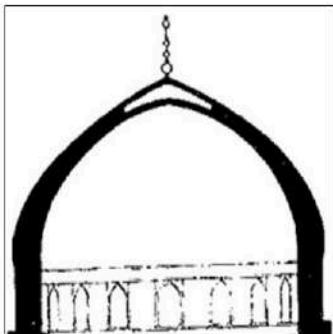
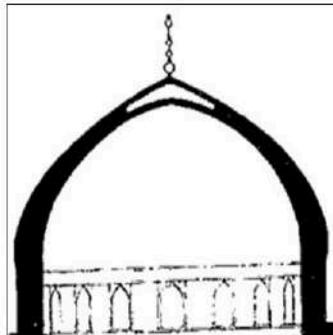
```
In [1]: import numpy as np
a=np.array([1,2,3,4])
b=np.array([2,1,0,5])

print(np.linalg.norm(a , 1))
print(np.linalg.norm(a , 2))
```

```
#https://pixspy.com/
```

```
10.0  
5.477225575051661
```

## KNN



```
In [2]: from sklearn import datasets  
from sklearn.model_selection import train_test_split  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.colors import ListedColormap
```

```
class KNeighborsClassifier(object):  
    def __init__(self, k):  
        self.k = k  
  
    def Euclidean(self, point, data):  
        return np.sqrt(np.sum((point - data)**2 , axis=1) )  
  
    def Manhattan(self, point, data):  
        return np.sum(np.abs(point - data) ,axis=1)  
  
    def Chebyshev(self, point, data):  
        return np.max(np.abs(point - data) ,axis=1)  
  
    def fit(self, X, y):  
        self.X_train = X  
        self.y_train = y  
  
    def most_common(self, List):  
        return max(set(List), key=List.count)  
  
    def predict(self, X_test):  
        neighbors = []  
        for x in X_test:  
            distances = self.Chebyshev(x, self.X_train)  
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]  
            neighbors.append(y_sorted[:self.k])  
        return list(map(self.most_common, neighbors))  
  
    def evaluate(self, X_test, y_test):  
        y_pred = np.array(self.predict(X_test))  
        accuracy = np.mean(y_pred == y_test)  
        return accuracy
```

## Iris

```
In [3]: iris = datasets.load_iris()  
X, y = iris.data, iris.target  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)  
cmap = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])  
plt.figure()  
plt.scatter(X[:,2],X[:,3], c=y, cmap=cmap, edgecolor='k', s=20)  
plt.show()
```

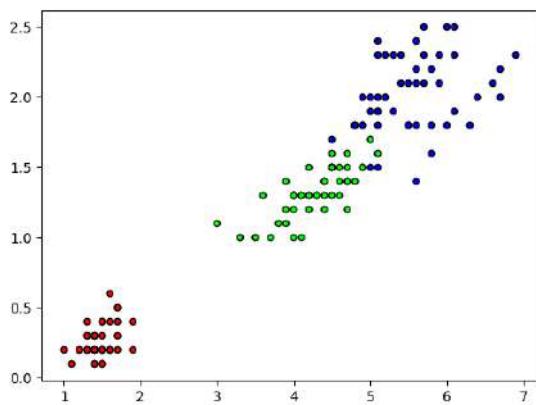
```

clf = KNeighborsClassifier(3)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
acc = np.sum(predictions == y_test) / len(y_test)

print("Predictions: ", predictions)
print("Accuracy: ", acc*100)

for i in range(len(predictions)):
    print(f"the predicted y are: {predictions[i]} and the real y are: {y_test[i]}")

```



Predictions: [1, 1, 2, 0, 1, 0, 0, 1, 2, 1, 0, 2, 1, 0, 1, 2, 0, 2, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0]  
Accuracy: 100.0

the predicted y are: 1 and the real y are: 1  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 0 and the real y are: 0  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 1 and the real y are: 1  
the predicted y are: 2 and the real y are: 2  
the predicted y are: 0 and the real y are: 0

## Mnist

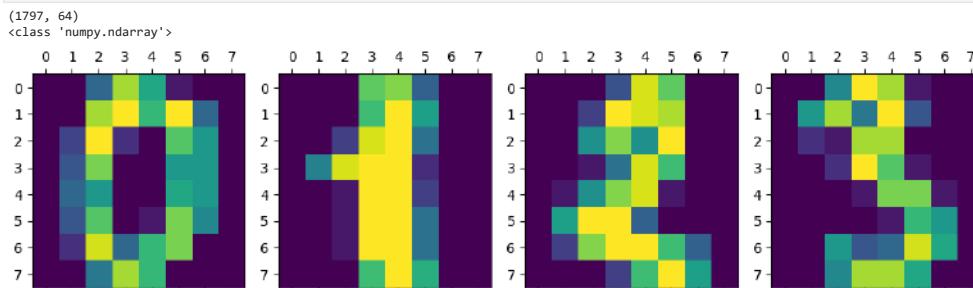
### Visualize data

```

In [2]: from sklearn.datasets import load_digits
mnist = load_digits()
print(mnist.data.shape)
#print(mnist.feature_names)
import matplotlib.pyplot as plt

print(type(mnist.images))
#plt.gray()
fig , (ax1 ,ax2 ,ax3 , ax4)=plt.subplots(nrows=1 , ncols=4)
fig.set_size_inches(10, 8)
ax1.matshow(mnist.images[0])
ax2.matshow(mnist.images[1])
ax3.matshow(mnist.images[2])
ax4.matshow(mnist.images[3])
plt.tight_layout()
plt.show()

```



```

In [5]: from sklearn.model_selection import train_test_split
import numpy as np

mnist = load_digits()
X = mnist.data
y = mnist.target

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print(np.unique(y_train, return_counts=True))
print(np.unique(y_test, return_counts=True))

(1257, 64) (1257, 64)
(540, 64) (540, 64)
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([119, 126, 124, 137, 120, 125, 124, 129, 126, 127], dtype=int64))
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([59, 56, 53, 46, 61, 57, 57, 50, 48, 53], dtype=int64))

```

```
In [6]: for k in range(3, 22, 2):
    model = KNeighborsClassifier(k)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    acc = model.evaluate(X_test, y_test)
    print("K = "+str(k)+" Accuracy: "+str(acc))

K = 3; Accuracy: 0.9629629629629629
K = 5; Accuracy: 0.9611111111111111
K = 7; Accuracy: 0.9611111111111111
K = 9; Accuracy: 0.9555555555555556
K = 11; Accuracy: 0.9555555555555556
K = 13; Accuracy: 0.9537037037037037
K = 15; Accuracy: 0.9537037037037037
K = 17; Accuracy: 0.9481481481481482
K = 19; Accuracy: 0.9444444444444444
K = 21; Accuracy: 0.9333333333333333
```

## Sklearn

```
In [7]: from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [8]: iris = datasets.load_iris()
print("feature_names\n", iris.feature_names, "\n")
print("target_names\n", iris.target_names, "\n")

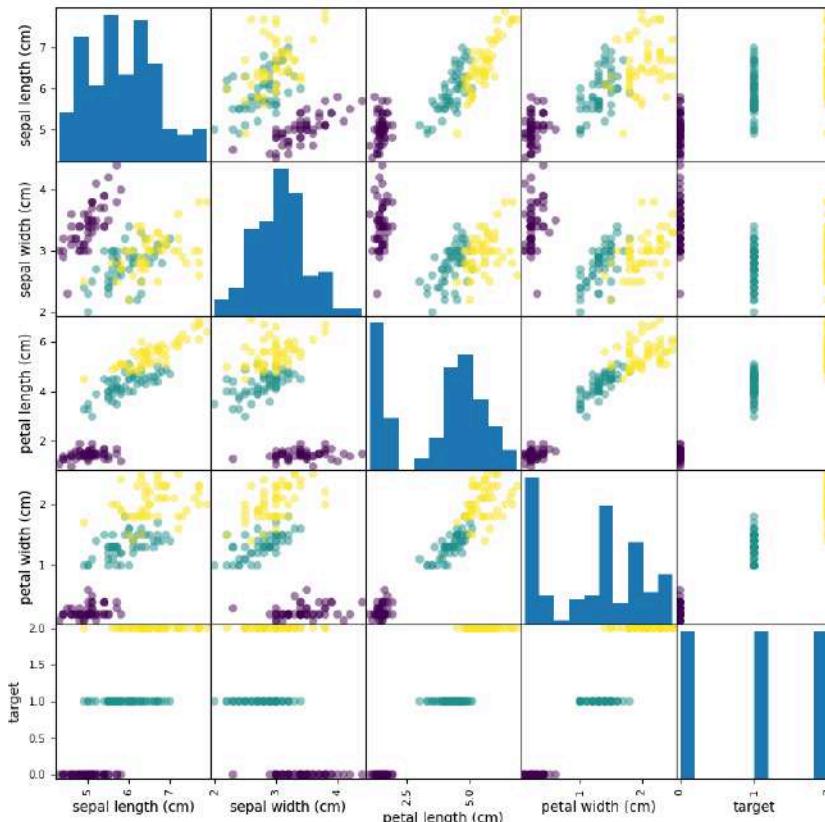
feature_names
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

target_names
['setosa' 'versicolor' 'virginica']
```

```
In [9]: df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
df.tail()
```

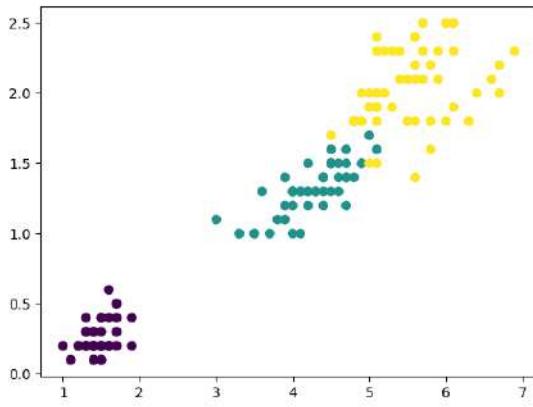
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

```
In [10]: pd.plotting.scatter_matrix(df, c=iris.target, figsize=[10,10], s=150)
plt.show()
```



```
In [11]: X=iris.data[:, [2,3]]
y=iris.target

plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



```
In [12]: from sklearn.neighbors import KNeighborsClassifier
In [13]: knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
In [14]: X=iris.data
y=iris.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.25, random_state=123)
knn.fit(X_train, y_train)
y_predict=knn.predict(X_test)

In [15]: print("KNN score after fitting: ", knn.score(X_test, y_test))
KNN score after fitting: 0.9736842105263158

In [16]: from sklearn.metrics import accuracy_score
acc=accuracy_score(y_test, y_predict)
acc
Out[16]: 0.9736842105263158

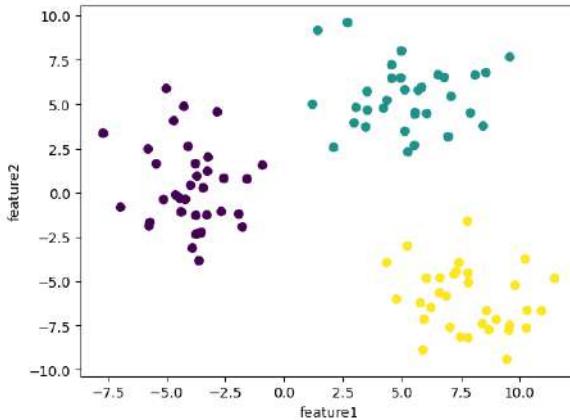
In [20]: for i in range(len(y_predict)):
    print(f"the predicted y are: {y_predict[i]} and the real y are: {y_test[i]}")
the predicted y are: 2 and the real y are: 1
the predicted y are: 2 and the real y are: 2
the predicted y are: 2 and the real y are: 2
the predicted y are: 1 and the real y are: 1
the predicted y are: 0 and the real y are: 0
the predicted y are: 2 and the real y are: 2
the predicted y are: 1 and the real y are: 1
the predicted y are: 0 and the real y are: 0
the predicted y are: 0 and the real y are: 0
the predicted y are: 1 and the real y are: 1
the predicted y are: 2 and the real y are: 2
the predicted y are: 0 and the real y are: 0
the predicted y are: 0 and the real y are: 0
the predicted y are: 2 and the real y are: 2
the predicted y are: 2 and the real y are: 2
the predicted y are: 0 and the real y are: 0
the predicted y are: 1 and the real y are: 1
the predicted y are: 0 and the real y are: 0
the predicted y are: 0 and the real y are: 0
the predicted y are: 2 and the real y are: 2
the predicted y are: 0 and the real y are: 0
the predicted y are: 2 and the real y are: 2
the predicted y are: 0 and the real y are: 0
the predicted y are: 0 and the real y are: 0
the predicted y are: 0 and the real y are: 0
the predicted y are: 1 and the real y are: 1
the predicted y are: 1 and the real y are: 1
the predicted y are: 2 and the real y are: 2
the predicted y are: 0 and the real y are: 0
```

## Example 2

```
In [31]: import numpy as np
from sklearn.datasets import make_blobs

X, y=make_blobs(n_samples=100, n_features=2, centers=3, cluster_std=2, random_state=26)

# #visualize
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel("feature1")
plt.ylabel("feature2")
plt.show()
```



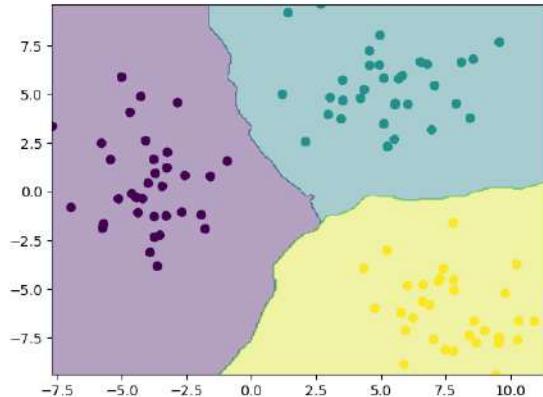
```
In [32]: from sklearn.neighbors import KNeighborsClassifier
#k=std n
clf=KNeighborsClassifier(n_neighbors=5 , metric="euclidean" )
clf.fit(X, y)
#clf.predict([[7.5,-7.5]])
clf.predict([[2,2]])
#clf.predict([[2,8]])
#clf.predict([[1.8,9]])
Out[32]: array([1])
```

```
In [33]: x_min , x_max=X[:,0].min() , X[:,0].max()
y_min , y_max=X[:,1].min() , X[:,1].max()

x_test=np.linspace(x_min , x_max , 300)
y_test=np.linspace(y_min , y_max , 300)

xx, yy=np.meshgrid(x_test , y_test)
x_test=np.c_[xx.ravel() , yy.ravel()]
pred=clf.predict(x_test).reshape(xx.shape)

plt.contourf(xx , yy , pred, alpha=0.4)
plt.scatter(X[:,0] , X[:,1] , c=y)
plt.show()
```



```
In [ ]: class sklearn.neighbors.KNeighborsClassifier(
    n_neighbors=5,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    metric='minkowski',
    p=2,
    n_jobs=None)

n_neighborsint, default=5
Number of neighbors to use by default for kneighbors queries.

weights{'uniform', 'distance'}, default='uniform'
Weight function used in prediction. Possible values:
'uniform' : uniform weights. All points in each neighborhood are weighted equally.
'distance' : weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

algorithm{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'

'ball_tree' will use BallTree
'kd_tree' will use KDTree
'brute' will use a brute-force search.
'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.

leaf_sizeint, default=30
Leaf size passed to Balltree or KDTree.
This can affect the speed of the construction and query,
as well as the memory required to store the tree.
The optimal value depends on the nature of the problem.

metricstr or callable, default='minkowski'
Metric to use for distance computation.
Default is "minkowski", which results in the standard Euclidean distance when p = 2.

pfloat, default=2
```

```

Power parameter for the Minkowski metric.
When p = 1, this is equivalent to using manhattan_distance (11),
and euclidean_distance (12) for p = 2.

```

```

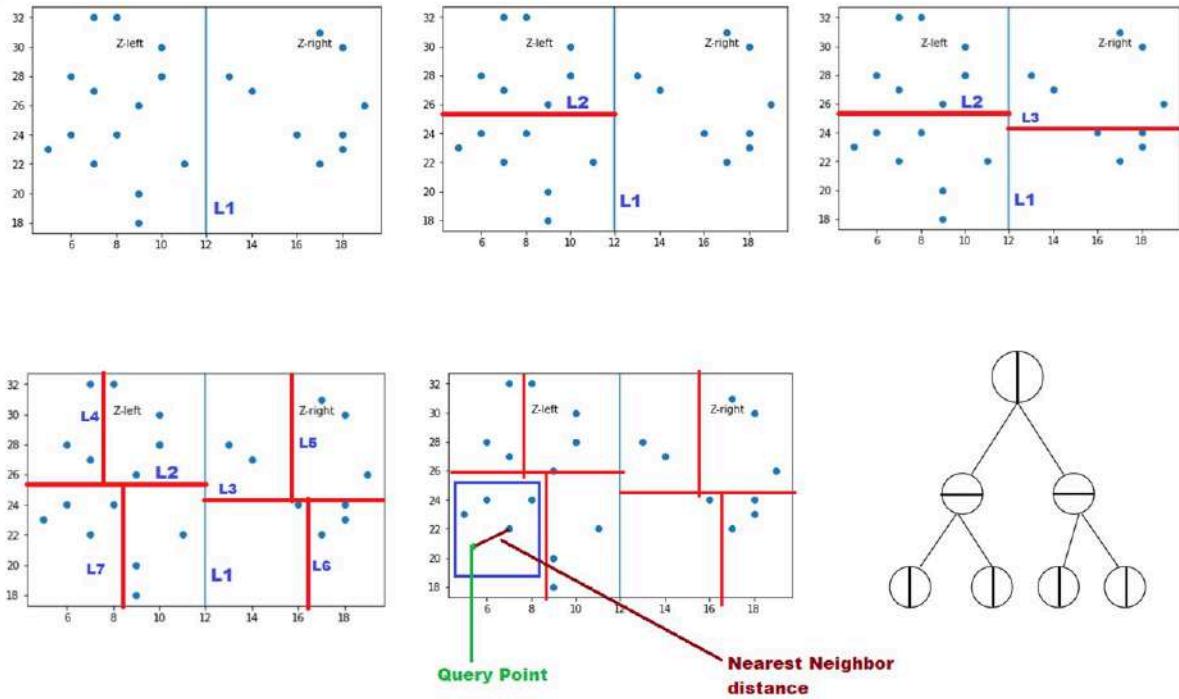
n_jobsint, default=None
The number of parallel jobs to run for neighbors search.
None means 1 .
-1 means using all processors

```

## KNN

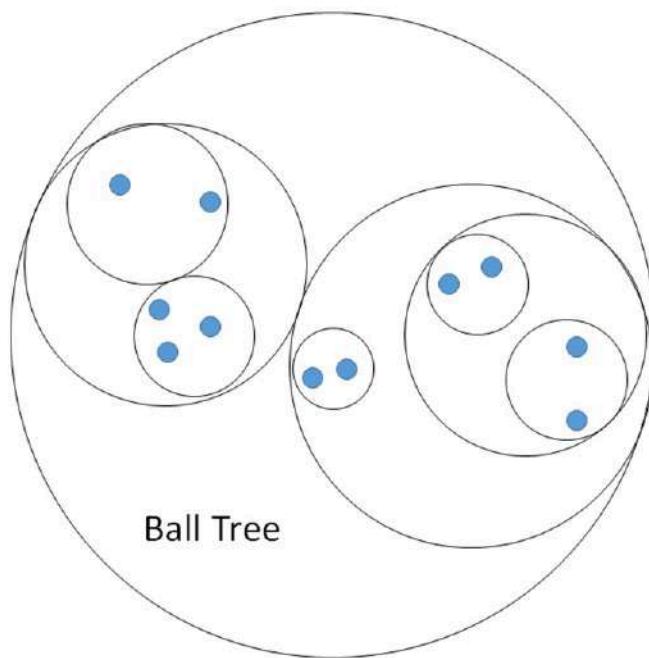
### Space Partitioning

#### KDTree



## Space Partitioning

### Ball Tree



<https://varshasaini.in/kd-tree-and-ball-tree-knn-algorithm/>

```
In [43]: import math
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=math.inf, weights="distance", algorithm="ball_tree")
```

```
In [44]: X=iris.data
y=iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.25, random_state=123)

knn.fit(X_train, y_train)
y_predict=knn.predict(X_test)
```

```
In [45]: from sklearn.metrics import accuracy_score
acc=accuracy_score(y_test, y_predict)
acc
```

```
Out[45]: 1.0
```

# Machine learning

## Decision Tree

Morteza khorsand



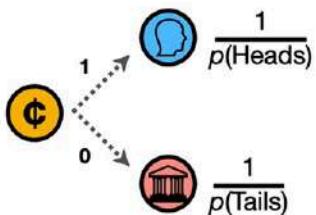
### Decision Tree

Attributes				Classes
Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

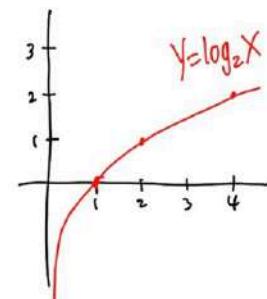
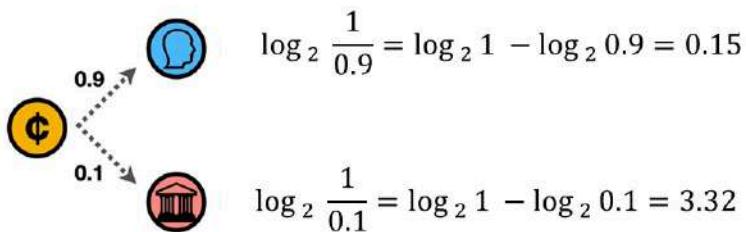
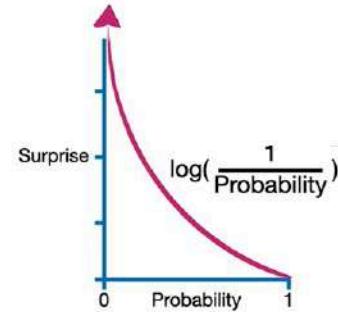
## Decision Tree

### Entropy

Surprise



$$\text{Surprise} = \log_2 \frac{1}{p}$$



## Decision Tree

### Entropy

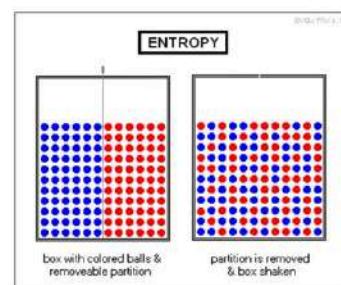
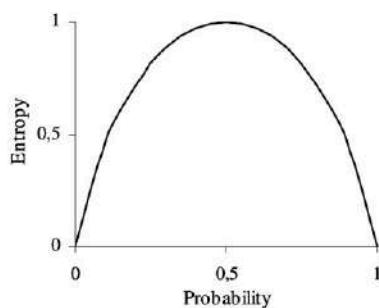
	Heads	Tails
Probability $p(x)$ :	0.9	0.1
Surprise:	0.15	3.32

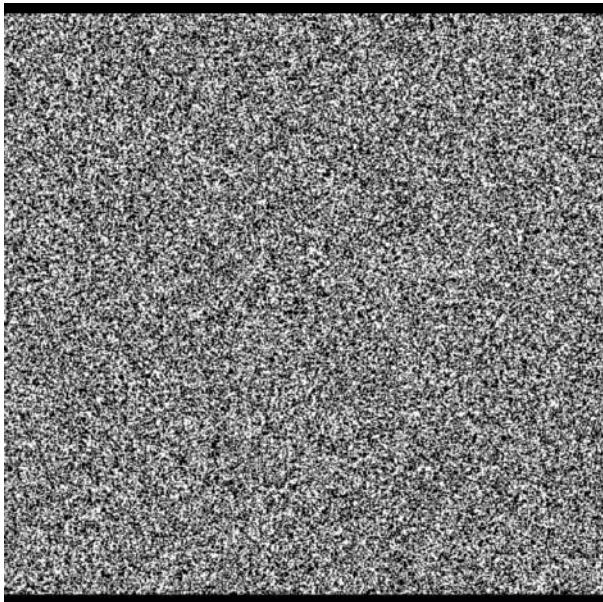
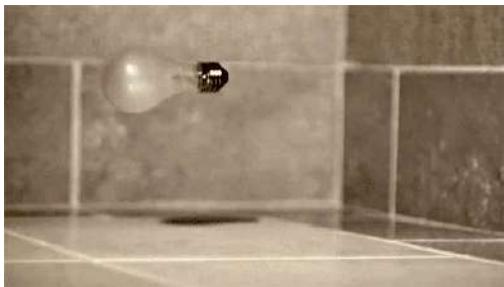
$$0.9 \times 0.15 + 0.1 \times 3.32 = 0.47$$

$$\text{Entropy} = \sum_{i=1}^n \log_2 \frac{1}{p} \times p$$

$$\sum_{i=1}^n \log_2 \frac{1}{p} \times p = \sum_{i=1}^n p \times \log_2 1 - \log_2 p$$

$$- \sum_{i=1}^n \log_2 p \times p$$



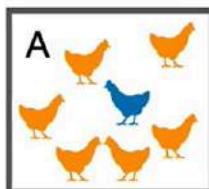


## Decision Tree

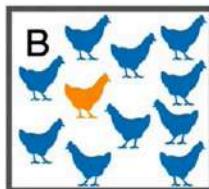
### Entropy

Entropy quantifies similarities and differences.

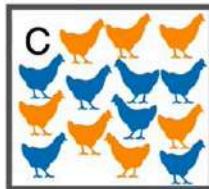
Surprise



$$\text{Entropy} = \log_2 \frac{1}{6/7} \times \frac{6}{7} + \log_2 \frac{1}{1/7} \times \frac{1}{7} = 0.59$$



$$\text{Entropy} = \log_2 \frac{1}{1/11} \times \frac{1}{11} + \log_2 \frac{1}{10/11} \times \frac{10}{11} = 0.44$$



$$\text{Entropy} = \log_2 \frac{1}{7/14} \times \frac{7}{14} + \log_2 \frac{1}{7/14} \times \frac{7}{14} = 1$$

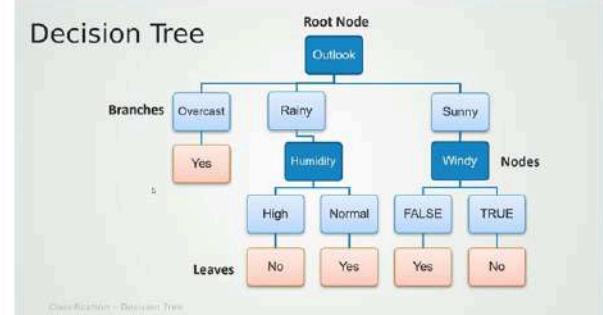
In a closed system, Entropy will always increase with time."

If you see the randomness increasing(in a closed system), you'll know that time is moving forward.

## Decision Tree

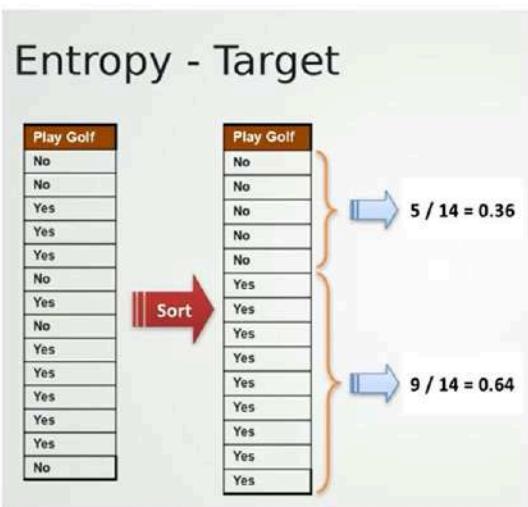
Attributes					Classes
Outlook	Temperature	Humidity	Windy		Play Golf
Rainy	Hot	High	FALSE	No	
Rainy	Hot	High	TRUE	No	
Overcast	Hot	High	FALSE	Yes	
Sunny	Mild	High	FALSE	Yes	
Sunny	Cool	Normal	FALSE	Yes	
Sunny	Cool	Normal	TRUE	No	
Overcast	Cool	Normal	TRUE	Yes	
Rainy	Mild	High	FALSE	No	
Rainy	Cool	Normal	FALSE	Yes	
Sunny	Mild	Normal	FALSE	Yes	
Rainy	Mild	Normal	TRUE	Yes	
Overcast	Mild	High	TRUE	Yes	
Overcast	Hot	Normal	FALSE	Yes	
Sunny	Mild	High	TRUE	No	

### Decision Tree



$$\text{Entropy}(T, X) = \sum p(x) \times E(x)$$

## Decision Tree



$$\text{Entropy} = \sum_{i=1}^n \log_2 \frac{1}{p_i} \times p_i$$

$$\log_2 \frac{1}{0.36} \times 0.36 + \log_2 \frac{1}{0.64} \times 0.64 = 0.94$$

```
In [1]: import numpy as np
from scipy.stats import entropy

y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1])

# Calculate probabilities of each class
unique, counts = np.unique(y, return_counts=True)
probabilities = counts / len(y)

# Calculate entropy using the probabilities
shannon_entropy = entropy(probabilities, base=2)
print(f'Entropy: {shannon_entropy}')

Entropy: 0.940285958670631
```

## Decision Tree

$$\text{Entropy}(T, X) = \sum p(x) \times E(x)$$

		Play Golf		Total
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
		14		Total

$$\text{Entropy}(T, X) = [\frac{5}{14} * ((\frac{3}{5} * \log_2 \frac{1}{3}) + \frac{2}{5} * \log_2 \frac{1}{2})] + [\frac{4}{14} * ((\frac{4}{4} * \log_2 \frac{1}{1}) + \frac{0}{4} * \log_2 \frac{1}{0})] + [\frac{5}{14} * ((\frac{2}{5} * \log_2 \frac{1}{2}) + \frac{3}{5} * \log_2 \frac{1}{3})] = 0.693$$

		Play Golf		Total
		Yes	No	
Temp	Hot	2	2	4
	Mild	4	2	6
	Cool	3	1	4
		14		Total

$$\text{Entropy}(T, X) = [\frac{4}{14} * ((\frac{2}{4} * \log_2 \frac{1}{2}) + \frac{2}{4} * \log_2 \frac{1}{2})] + [\frac{6}{14} * ((\frac{4}{6} * \log_2 \frac{1}{4}) + \frac{2}{6} * \log_2 \frac{1}{2})] + [\frac{4}{14} * ((\frac{3}{4} * \log_2 \frac{1}{3}) + \frac{1}{4} * \log_2 \frac{1}{1})] = 0.911$$

		Play Golf		Total
		Yes	No	
Humidity	High	3	4	7
	Normal	6	1	7
		14		Total

$$\text{Entropy}(T, X) = [\frac{7}{14} * ((\frac{3}{7} * \log_2 \frac{1}{3}) + \frac{4}{7} * \log_2 \frac{1}{4})] + [\frac{7}{14} * ((\frac{6}{7} * \log_2 \frac{1}{6}) + \frac{1}{7} * \log_2 \frac{1}{1})] = 0.788$$

		Play Golf		Total
		Yes	No	
Windy	False	6	2	8
	True	3	3	6
		14		Total

$$\text{Entropy}(T, X) = [\frac{8}{14} * ((\frac{6}{8} * \log_2 \frac{1}{6}) + \frac{2}{8} * \log_2 \frac{1}{2})] + [\frac{6}{14} * ((\frac{3}{6} * \log_2 \frac{1}{3}) + \frac{3}{6} * \log_2 \frac{1}{3})] = 0.892$$

## Decision Tree

### Information Gain

$$\text{Gain} = E(T) - E(T, X)$$

↓ Parent      ↓ Children

$$G = 0.94 - 0.693 = 0.247$$

Gain = 0.247		Play Golf		Total
		Yes	No	
Outlook	Sunny	3	2	
	Overcast	4	0	
	Rainy	2	3	

$$G = 0.94 - 0.911 = 0.029$$

Gain = 0.029		Play Golf		Total
		Yes	No	
Temp	Hot	2	2	
	Mild	4	2	
	Cool	3	1	

$$G = 0.94 - 0.788 = 0.152$$

Gain = 0.152		Play Golf		Total
		Yes	No	
Humidity	High	3	4	
	Normal	6	1	
		14		Total

$$G = 0.94 - 0.892 = 0.048$$

Gain = 0.048		Play Golf		Total
		Yes	No	
Windy	False	6	2	
	True	3	3	
		14		Total

In [3]: `!pip install info-gain`

`^C`

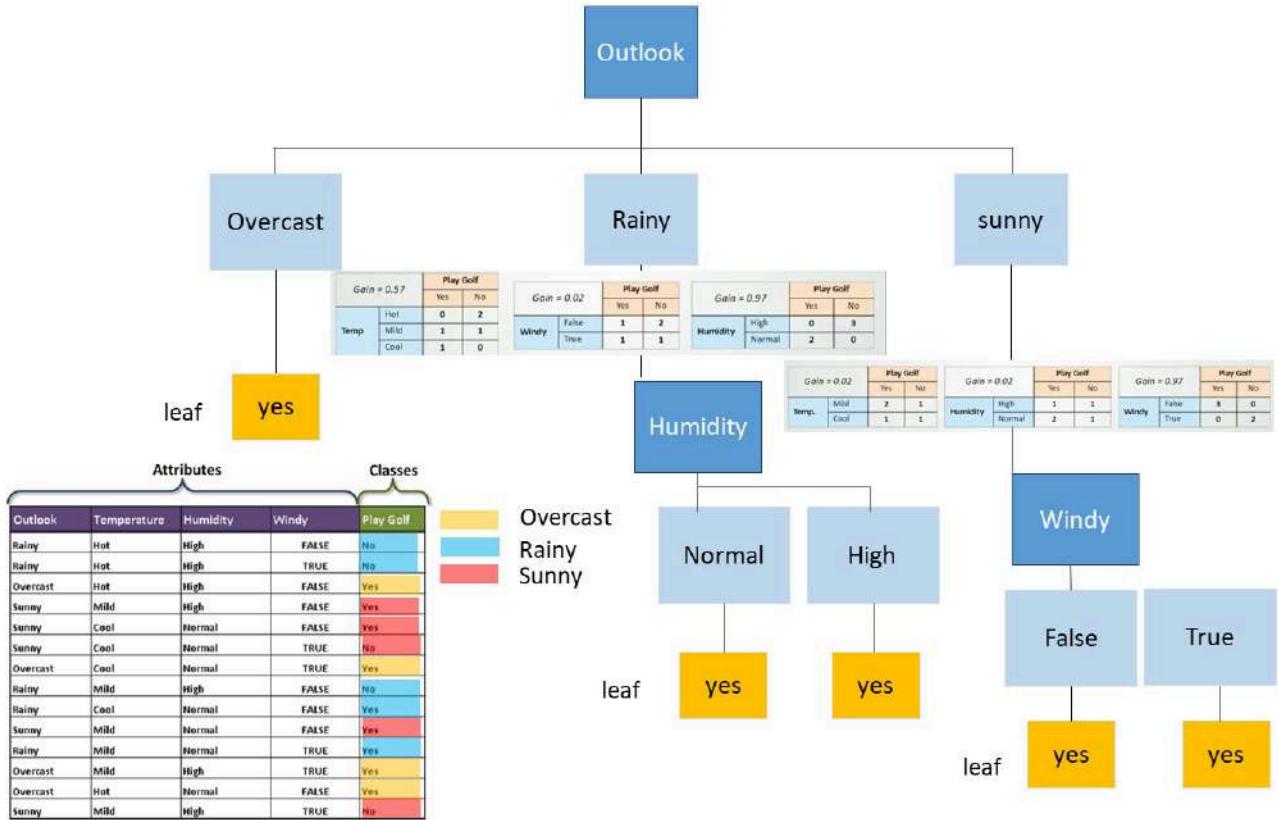
In [4]: `from info_gain import info_gain`

```
In [5]: y = [1,1,1,0,0,1,1,1,1,1,0,0,0]
outlook = ["sunny", "sunny", "sunny", "sunny", "sunny",
           "overcast", "overcast", "overcast", "overcast",
           "rainy ", "rainy ", "rainy ", "rainy ", "rainy "]

gainoutlook = info_gain.info_gain(outlook, y)
gainoutlook

Out[5]: 0.25072806734968767
```

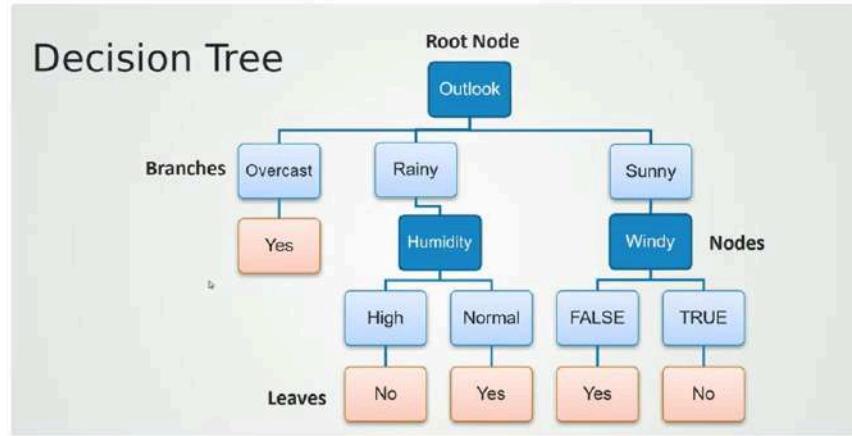
## Decision Tree



Attributes				Classes
Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	No
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Overcast  
Rainy  
Sunny

## Decision Tree



### Rules

R1 = if (outlook == overcast) then YES  
R2 = if (outlook == Rainy & (Humidity == High) then No  
R3= if (outlook== Rainy) & (Humidity == Normal) then YES  
R4= if (outlook == Sunny ) & (Windy == False) then YES  
R5 = if(outlook == Sunny) & (Windy == True) then NO

```
In [6]: import numpy as np
#area , bedroom , parking , Warehouse
X=np.array([[120,2,1,0],
            [100,1,0,0],
            [60,1,1,0],
            [150,3,1,1]])

y=[1,0,0,1]

In [7]: import scipy as sp
print(sp.stats.entropy(X , axis=1 , base = 2))
print(sp.stats.entropy(y , base = 2))
[0.18782449  0.08013605  0.23785057  0.24981672]
1.0

In [8]: from info_gain import info_gain
gain1=info_gain.info_gain(X[:, 0], y)
gain2=info_gain.info_gain(X[:, 1], y)
gain3=info_gain.info_gain(X[:, 2], y)
gain4=info_gain.info_gain(X[:, 3], y)

print("area gain is {0} , the bedroom gain is {1} , the parking gain is {2} and the warehouse gain is {3}".format(gain1 , gain2 , gain3 ,gain4))
area gain is 0.6931471805599453 , the bedroom gain is 0.6931471805599452 , the parking gain is 0.21576155433883565 and the warehouse gain is 0.21576155433883565

In [9]: from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(X , y)

Out[9]: DecisionTreeClassifier()

In [10]: clf.predict([[160 ,2,1,1]])
Out[10]: array([1])

Requirement already satisfied: info-gain in f:\users\nikoo\anaconda3\lib\site-packages (1.0.1)
Requirement already satisfied: info-gain in f:\users\nikoo\anaconda3\lib\site-packages (1.0.1)
WARNING: Ignoring invalid distribution -illow (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (f:\users\nikoo\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (f:\users\nikoo\anaconda3\lib\site-packages)
```

## Decision Tree

### Missing values

Numeric

- average
- median

Categorical

- Most common values
- Missing value

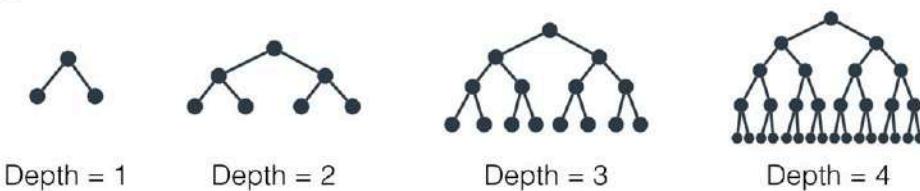
### Over fitting

- pruning

**Max Depth : int, default=None**

The maximum depth of the tree.

If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.



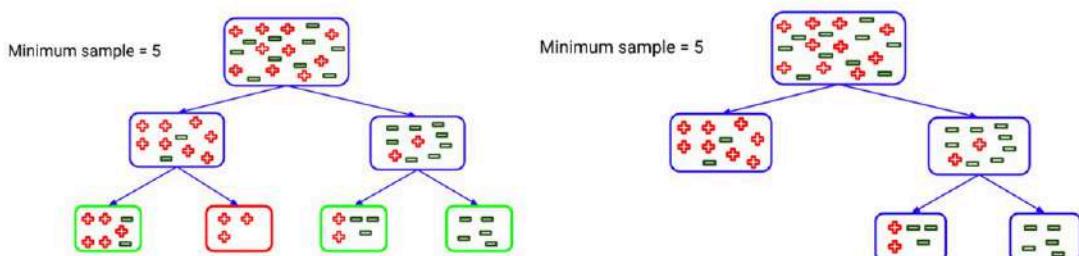
## Decision Tree

### min\_samples\_leaf int or float, default=1

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model.

If int, then consider min\_samples\_leaf as the minimum number.

If float, then min\_samples\_leaf is a fraction and ceil(min\_samples\_leaf \* n\_samples) are the minimum number of samples for each node.



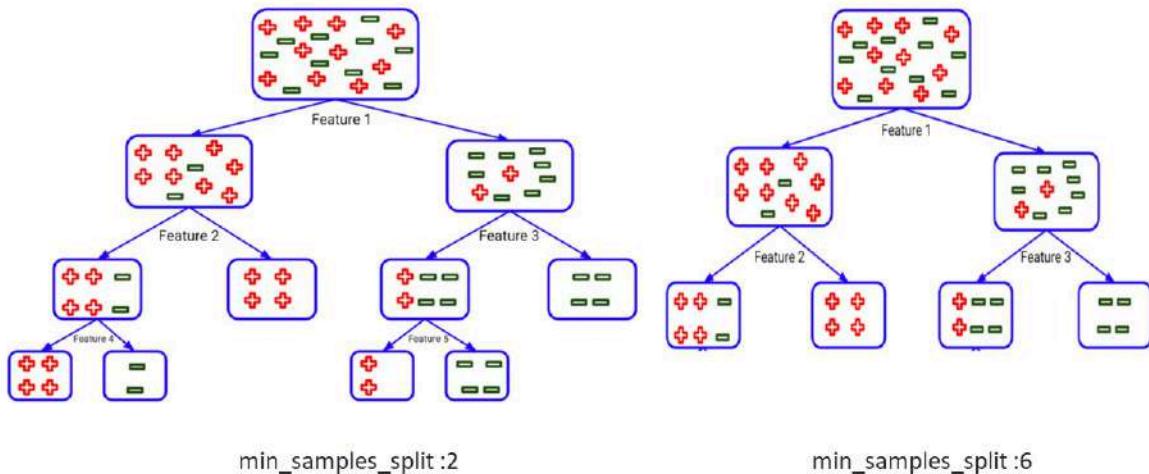
## Decision Tree

### min\_samples\_split : int or float, default=2

The minimum number of samples required to split an internal node:

- If int, then consider min\_samples\_split as the minimum number.

- If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.



## Decision Tree

### Gini Impurity

$$Gi(x) = 1 - \sum_i^n p_i^2$$

### Entropy - Target

Play Golf
No
No
Yes
Yes
Yes
No
Yes
No
Yes
No

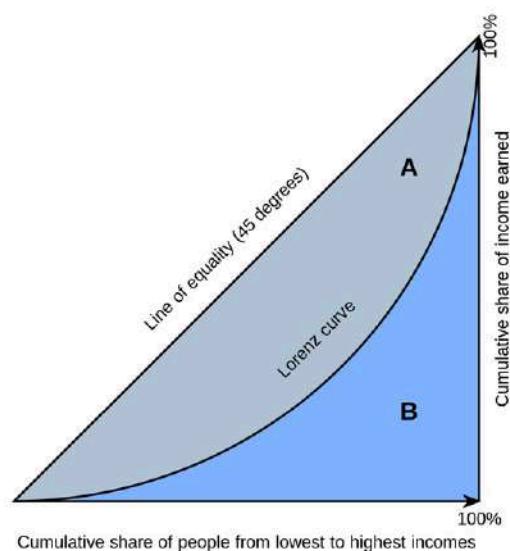
Sort →

Play Golf
No
Yes
No

$5 / 14 = 0.36$

$9 / 14 = 0.64$

$$Gi(x) = 1 - \left( \left( \frac{9}{14} \right)^2 + \left( \frac{5}{14} \right)^2 \right) = 0.4592$$



## Decision Tree

		Play Golf		Total
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

$$G_i(x) = 1 - \sum_i^n p_i^2$$

$$Gini_{Sunny} = 1 - \left( \left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 \right) = 0.48$$

$$Gini_{Overcast} = 1 - \left( \left(\frac{4}{4}\right)^2 + \left(\frac{0}{4}\right)^2 \right) = 0$$

$$Gini_{Rainy} = 1 - \left( \left(\frac{2}{5}\right)^2 + \left(\frac{3}{5}\right)^2 \right) = 0.48$$

Weighted average Gini for "Outlook":

$$Gini_{outlook} = \frac{5}{14} \times Gini_{Sunny} + \frac{4}{14} \times Gini_{Overcast} + \frac{5}{14} \times Gini_{Rainy} = 0.3428$$

		Play Golf		Total
		Yes	No	
Temp	Hot	2	2	4
	Mild	4	2	6
	Cool	3	1	4
				14

$$Gini_{Sunny} = 1 - \left( \left(\frac{2}{4}\right)^2 + \left(\frac{2}{4}\right)^2 \right) = 0.5$$

$$Gini_{Overcast} = 1 - \left( \left(\frac{4}{6}\right)^2 + \left(\frac{2}{6}\right)^2 \right) = 0.444$$

$$Gini_{Rainy} = 1 - \left( \left(\frac{3}{7}\right)^2 + \left(\frac{4}{7}\right)^2 \right) = 0.375$$

$$Gini_{temp} = \frac{4}{14} \times Gini_{hot} + \frac{6}{14} \times Gini_{mild} + \frac{4}{14} \times Gini_{cool} = 0.440$$

$$Gini_{High} = 1 - \left( \left(\frac{3}{7}\right)^2 + \left(\frac{4}{7}\right)^2 \right) = 0.489$$

$$Gini_{Normal} = 1 - \left( \left(\frac{6}{7}\right)^2 + \left(\frac{1}{7}\right)^2 \right) = 0.244$$

$$Gini_{temp} = \frac{7}{14} \times Gini_{high} + \frac{7}{14} \times Gini_{normal} = 0.367$$

$$Gini_{windy} = 0.428$$

		Play Golf		Total
		Yes	No	
Humidity	High	3	4	7
	Normal	6	1	7
				14

		Play Golf		Total
		Yes	No	
Windy	False	6	2	8
	True	3	3	6
				14

### Gini Gain

$$Gain = Gi(T) - Gi(T, X)$$

↓ Parent      ↓ Children

$$G = 0.4592 - 0.3428 = 0.1164$$

Gain = 0.1164		Play Golf		Total
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

$$G = 0.4592 - 0.440 = 0.0192$$

Gain = 0.0192		Play Golf		Total
		Yes	No	
Temp	Hot	2	2	4
	Mild	4	2	6
	Cool	3	1	4
				14

$$G = 0.4592 - 0.367 = 0.092$$

Gain = 0.092		Play Golf		Total
		Yes	No	
Humidity	High	3	4	7
	Normal	6	1	7
				14

$$G = 0.4592 - 0.428 = 0.0312$$

Gain = 0.0312		Play Golf		Total
		Yes	No	
Windy	False	6	2	8
	True	3	3	6
				14

## Decision Tree

Random Forest    Ensemble learning

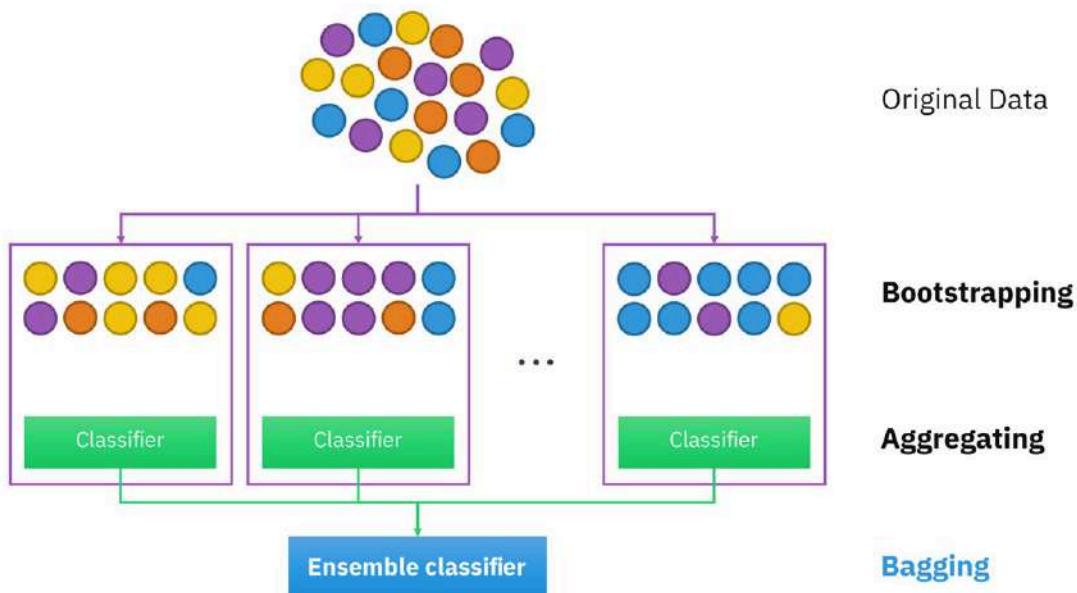
- Boot strapped

Original Dataset					Bootstrapped Dataset					
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease	
No	No	No	125	No	2	Yes	Yes	Yes	180	Yes
Yes	Yes	Yes	180	Yes	1	No	No	No	125	No
Yes	Yes	No	210	No	4	Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes	4	Yes	No	Yes	167	Yes

Out of bag dataset

Typically about 1/3 of the original data dose not end up in the bootstrapped dataset

## Decision Tree

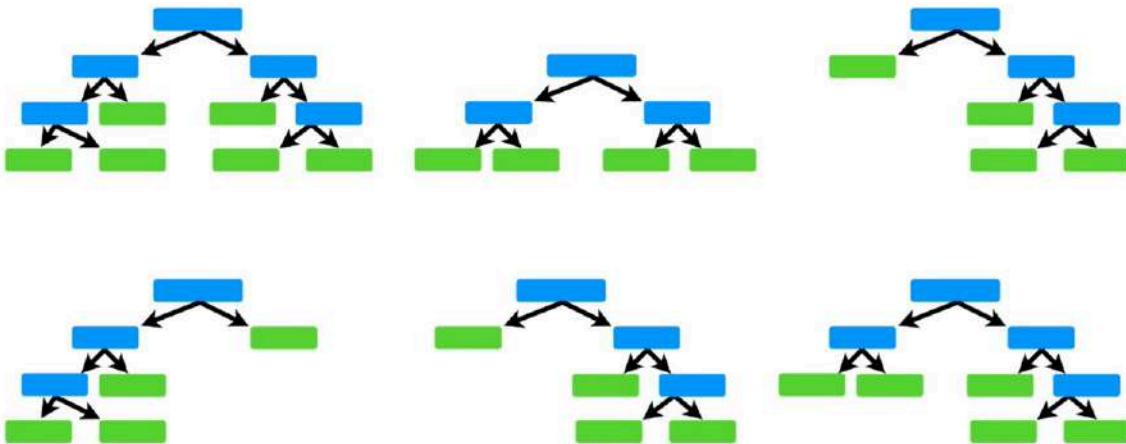


## Decision Tree

### Random Forest

#### Bootstrapped

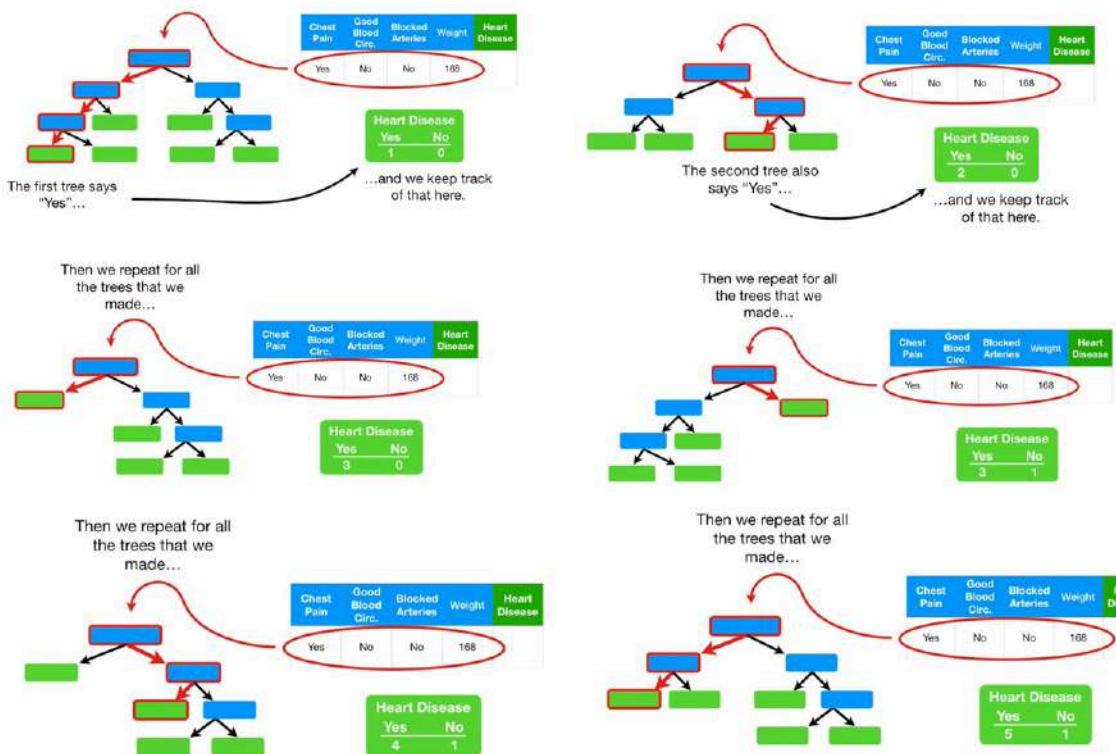
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes



## Decision Tree

### Random Forest

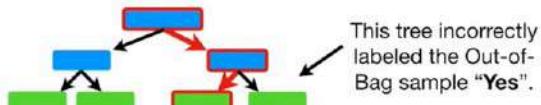
#### Boot strapped



## Decision Tree

### Random Forest

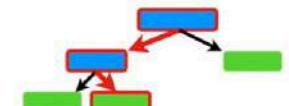
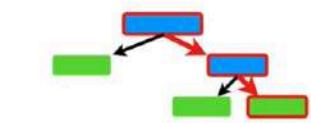
- Bagging = bootstrapping + aggregate



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Since the label with the most votes wins, it is the label that we assign this Out-of-Bag sample.



$$\text{Out of bag error} = \frac{\text{incorrect out of bag samples}}{\text{correct out of bag samples}}$$

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

### Example 1

```
In [1]: from sklearn.datasets import load_iris  
data= load_iris()  
X=data.data  
y= data.target
```

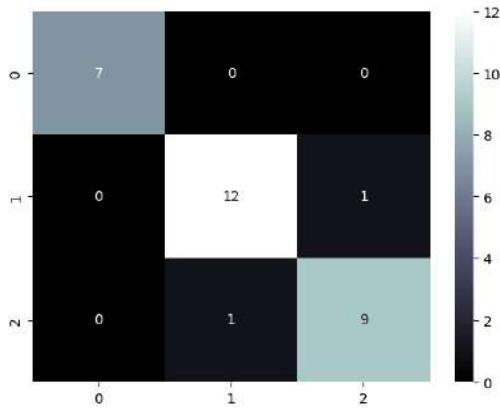
```
In [2]: from sklearn.preprocessing import StandardScaler  
sc=StandardScaler()  
Xnorm= sc.fit_transform(X)  
  
from sklearn.model_selection import train_test_split  
X_train , X_test , y_train , y_test= train_test_split(Xnorm , y , random_state= 142 , test_size= .2)
```

```
In [5]: from sklearn.tree import DecisionTreeClassifier  
clf2= DecisionTreeClassifier(criterion= "gini" , splitter='best' , max_depth=4)  
clf2.fit(X_train , y_train)
```

```
Out[5]: DecisionTreeClassifier(max_depth=4)
```

```
In [6]: predict= clf2.predict(X_test )  
print(clf2.score(X_test , y_test))  
print(clf2.score(X_train , y_train))  
0.9333333333333333  
0.9916666666666667
```

```
In [7]: from sklearn.metrics import confusion_matrix  
import seaborn as sb  
import matplotlib.pyplot as plt  
  
cf=confusion_matrix(predict , y_test)  
sb.heatmap(cf , annot=True , cmap= "bone")  
plt.show()
```

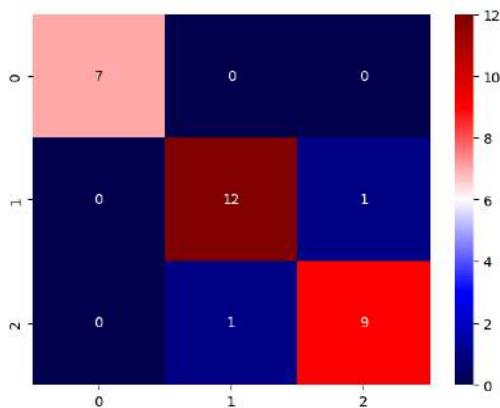


```
In [13]: #random forest
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100 , criterion="entropy",min_samples_split=8, bootstrap= True )
rf.fit(X_train , y_train)
```

```
Out[13]: RandomForestClassifier
RandomForestClassifier(criterion='entropy', min_samples_split=8)
```

```
In [14]: predict_test=rf.predict(X_test)
score_test = rf.score(X_test , y_test)
score_train= rf.score(X_train , y_train)
print(f'the test accuracy is: {score_test} and the train accuracy is: {score_train}')
the test accuracy is: 0.9333333333333333 and the train accuracy is: 0.9833333333333333
```

```
In [15]: from sklearn.metrics import confusion_matrix
cm= confusion_matrix(predict_test , y_test)
sb.heatmap(cm , annot=True , cmap="seismic")
plt.show()
```



## Example 2

```
In [22]: import pandas as pd
import numpy as np
df=pd.read_csv(r"D:\AI_Machinelearning\datasets\shows.csv")
df
```

```
Out[22]:   Age  Experience  Rank  Nationality  Go
  0    36          10     9      UK    NO
  1    42          12     4      USA    NO
  2    23           4     6       N    NO
  3    52           4     4      USA    NO
  4    43          21     8      USA   YES
  5    44          14     5      UK    NO
  6    66           3     7       N   YES
  7    35          14     9      UK   YES
  8    52          13     7       N   YES
  9    35           5     9       N   YES
 10   24            3     5      USA    NO
 11   18            3     7      UK   YES
 12   45            9     9      UK   YES
```

```
In [23]: X=np.array(df.iloc[:, :-1])
y=np.array(df.iloc[:, -1])
```

```
In [24]: df.replace({"UK" : 0 , "USA": 1 , "N": 2} , inplace = True )
df
```

```
Out[24]:
```

	Age	Experience	Rank	Nationality	Go
0	36	10	9		0 NO
1	42	12	4		1 NO
2	23	4	6		2 NO
3	52	4	4		1 NO
4	43	21	8		1 YES
5	44	14	5		0 NO
6	66	3	7		2 YES
7	35	14	9		0 YES
8	52	13	7		2 YES
9	35	5	9		2 YES
10	24	3	5		1 NO
11	18	3	7		0 YES
12	45	9	9		0 YES

```
In [25]: df["Go"] = df["Go"].map({"NO":0 , "YES":1})  
df  
  
X=np.array(df.iloc[ : , :-1])  
y=np.array(df.iloc[ : , -1])
```

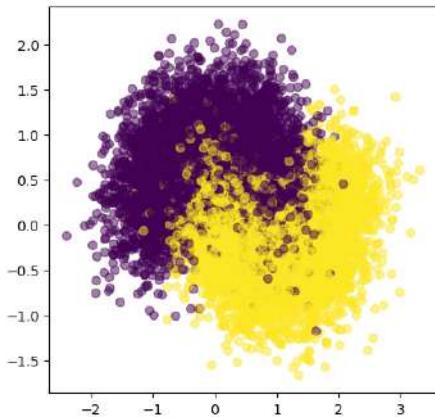
```
In [26]: from sklearn.tree import DecisionTreeClassifier  
  
clf3= DecisionTreeClassifier(criterion= "gini" , min_samples_split= 2 )  
clf3.fit(X , y)
```

```
Out[26]:
```

```
In [27]: clf3.predict(np.array([[40,10,3,0]]))  
Out[27]: array([0], dtype=int64)
```

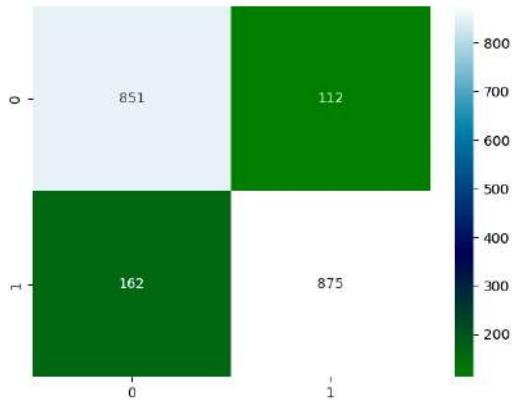
## Example 3

```
In [28]: #Libraries  
from sklearn.datasets import make_moons  
from sklearn.model_selection import train_test_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy_score  
  
X, y = make_moons(n_samples=10000, noise=0.4, random_state=42)  
  
plt.figure(figsize=(5,5))  
plt.scatter(X[ : ,0] , X[ : ,1], c=y , alpha=0.5)  
plt.show()
```



```
In [47]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2 , random_state=120)  
  
#Library  
from sklearn.tree import DecisionTreeClassifier  
dcl=DecisionTreeClassifier(criterion= "gini" , max_depth= 3 , min_samples_leaf=3)  
dcl.fit(X_train , y_train)  
predict_test=dcl.predict(X_test)  
predict_train=dcl.predict(X_train)  
  
print(f" the test accuracy is: {accuracy_score(predict_test , y_test)} and the train score is: {accuracy_score(predict_train , y_train)}")  
the test accuracy is: 0.8695 and the train score is: 0.856
```

```
In [30]: from sklearn.metrics import confusion_matrix  
  
conf = confusion_matrix(predict_test , y_test)  
sb.heatmap(conf , cmap="ocean" , annot=True, fmt= "0g" )  
plt.show()
```

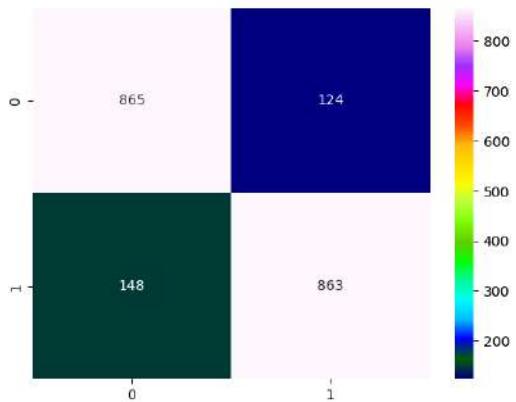


```
In [35]: from sklearn.ensemble import RandomForestClassifier
clfr = RandomForestClassifier(n_estimators=50, bootstrap=False, criterion="gini", max_depth=6, min_samples_leaf=3)
clfr.fit(X_train, y_train)
predict_test = clfr.predict(X_test)
predict_train = clfr.predict(X_train)

print(f"train accuracy is {accuracy_score(predict_train, y_train)}, test accuracy is {accuracy_score(predict_test, y_test)}")
```

```
train accuracy is 0.868625 , test accuracy is 0.864
```

```
In [36]: from sklearn.metrics import confusion_matrix
confr = confusion_matrix(predict_test, y_test)
sb.heatmap(confr, cmap="gist_ncar", annot=True, fmt="0g")
plt.show()
```

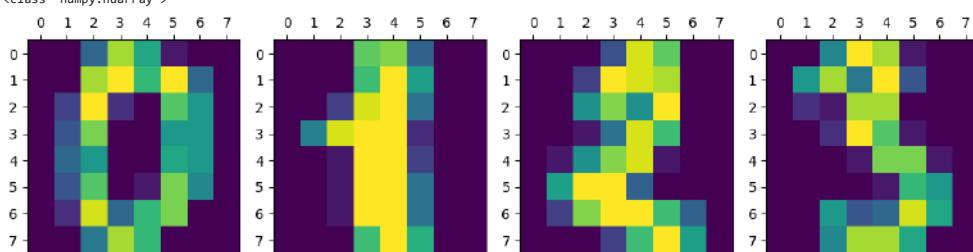


## Example 4

```
In [48]: from sklearn.datasets import load_digits
mnist = load_digits()
print(mnist.data.shape)
#print(mnist.feature_names)

import matplotlib.pyplot as plt

print(type(mnist.images))
#plt.gray()
fig, (ax1, ax2, ax3, ax4)=plt.subplots(nrows=1, ncols=4)
fig.set_size_inches(10, 8)
ax1.matshow(mnist.images[0])
ax2.matshow(mnist.images[1])
ax3.matshow(mnist.images[2])
ax4.matshow(mnist.images[3])
plt.tight_layout()
plt.show()
```



```
In [49]: X=mnist.data
y=mnist.target

#split train test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1563, test_size=0.2)
```

```
In [50]: #decision tree
from sklearn.tree import DecisionTreeClassifier
```

```
tr=DecisionTreeClassifier(criterion="entropy", max_depth=8, min_samples_split=10, min_samples_leaf=2)
tr.fit(X_train, y_train)
```

```
Out[50]: DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_leaf=2,
min_samples_split=10)
```

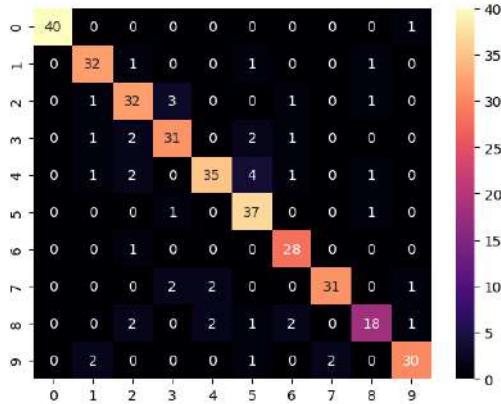
```
In [51]: #prediction
predict_test=tr.predict(X_test)
predict_train=tr.predict(X_train)

#accuracy
from sklearn.metrics import accuracy_score
print(f" train accuracy is: {accuracy_score(predict_train, y_train)}, test accuracy is: {accuracy_score(predict_test, y_test)}")

train accuracy is: 0.9471120389700766 , test accuracy is: 0.8722222222222222
```

```
In [52]: #confusion matrix
from sklearn.metrics import confusion_matrix

cnfm=confusion_matrix(predict_test, y_test)
sb.heatmap(cnfm, annot=True, cmap="magma")
plt.show()
```



```
In [53]: #random forest
from sklearn.ensemble import RandomForestClassifier
rndf=RandomForestClassifier(n_estimators=100, criterion="entropy", max_depth=8,
min_samples_split=10, min_samples_leaf=2, bootstrap=True)

#fit
rndf.fit(X_train, y_train)

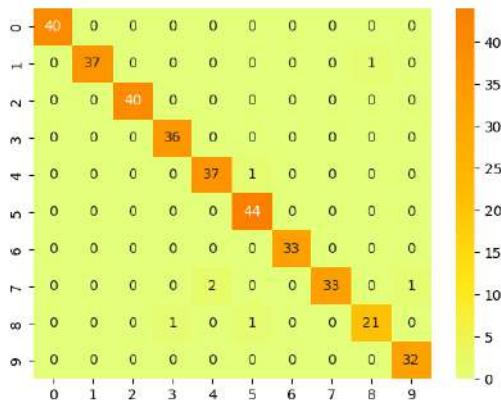
#predict
predict_test=rndf.predict(X_test)
predict_train=rndf.predict(X_train)

#accuracy
print(" train_accuracy is: {} and the test_accuracy is: {}".format(accuracy_score(predict_train, y_train), accuracy_score(predict_test, y_test)))

train_accuracy is: 0.9979123173277662 and the test_accuracy is: 0.9805555555555555
```

```
In [54]: #confusion matrix
from sklearn.metrics import confusion_matrix

cnfm2=confusion_matrix(predict_test, y_test)
sb.heatmap(cnfm2, annot=True, cmap="Wistia")
plt.show()
```



## Example 5

```
In [55]: import pandas as pd
from sklearn.datasets import load_wine

wine=load_wine()

df=pd.DataFrame(wine.data, columns=wine.feature_names)

df["target"] = wine.target

df
```

```
Out[55]:
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavonoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target		
0	14.23	1.71	2.43		15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04		3.92	1065.0	0
1	13.20	1.78	2.14		11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05		3.40	1050.0	0
2	13.16	2.36	2.67		18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03		3.17	1185.0	0
3	14.37	1.95	2.50		16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86		3.45	1480.0	0
4	13.24	2.59	2.87		21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04		2.93	735.0	0
...	...	...	...		...	...	...	...	...	...	...	...		...	...	...
173	13.71	5.65	2.45		20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64		1.74	740.0	2
174	13.40	3.91	2.48		23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70		1.56	750.0	2
175	13.27	4.28	2.26		20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59		1.56	835.0	2
176	13.17	2.59	2.37		20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60		1.62	840.0	2
177	14.13	4.10	2.74		24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61		1.60	560.0	2

178 rows × 14 columns

```
In [56]:
```

```
x=wine.data
y=wine.target

#normalize
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_norm= sc.fit_transform(X)

#train test split
from sklearn.model_selection import train_test_split
X_train , X_test, y_train , y_test = train_test_split(X_norm , y , random_state= 145 , test_size= 0.2)

#decision tree
from sklearn.tree import DecisionTreeClassifier
dc=DecisionTreeClassifier(criterion= "entropy" , max_depth= 8 , min_samples_split= 8 , min_samples_leaf=2)
dc.fit(X_train , y_train)
```

```
Out[56]:
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_leaf=2,
min_samples_split=8)
```

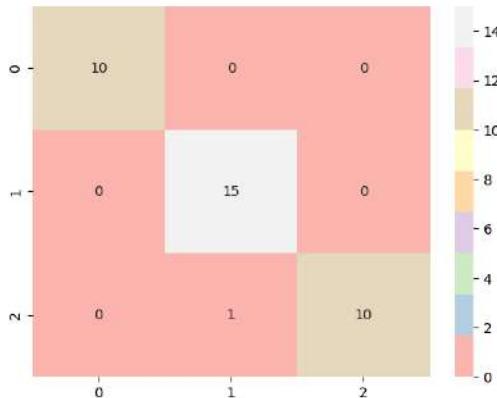
```
In [57]:
```

```
predict_train = dc.predict(X_train)
predict_test=dc.predict(X_test)

#accuracy
print(f" train accuracy is: {accuracy_score(predict_train , y_train)} and test accuracy is: { accuracy_score(predict_test , y_test)}")
train accuracy is: 0.9929577464788732 and test accuracy is: 0.9722222222222222
```

```
In [58]:
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix
cn=confusion_matrix(predict_test , y_test)
sb.heatmap(cn , annot=True , cmap="Pastel1")
plt.show()
```



```
In [59]:
```

```
#random forest
from sklearn.ensemble import RandomForestClassifier
wfr=RandomForestClassifier(n_estimators=100 , criterion="entropy" , max_depth= 8 , min_samples_split= 8 , min_samples_leaf=2)
wfr.fit(X_train , y_train)
```

```
Out[59]:
```

```
RandomForestClassifier(criterion='entropy', max_depth=8, min_samples_leaf=2,
min_samples_split=8)
```

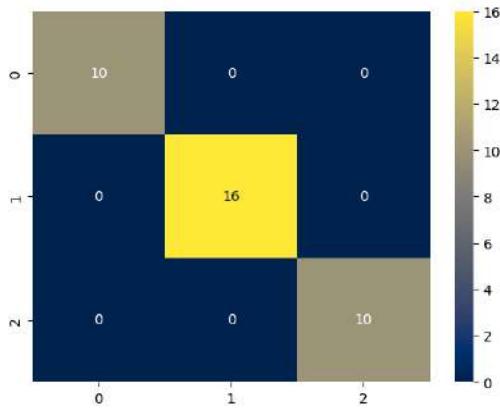
```
In [60]:
```

```
predict_test= wfr.predict(X_test)
predict_train=wfr.predict(X_train)

#score
print(f" train accuracy is: {accuracy_score(predict_train , y_train)} , test_accuracy is: {accuracy_score(predict_test , y_test)}")
train accuracy is: 1.0 , test_accuracy is: 1.0
```

```
In [61]:
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix
cn=confusion_matrix(predict_test , y_test)
sb.heatmap(cn , annot=True , cmap="cividis")
plt.show()
```



## Example 6

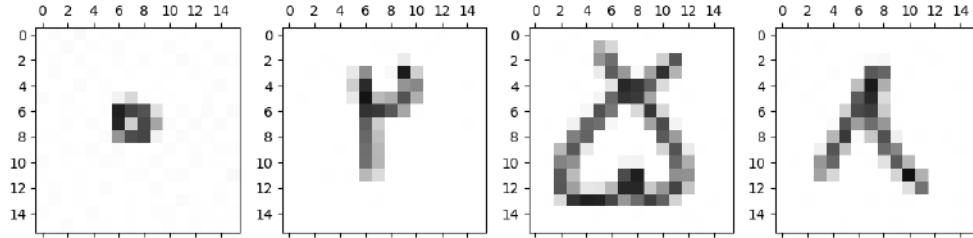
```
In [62]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import glob
from PIL import Image

data_dir= r'D:\AI_Machinelearning\datasets\persianDigits\dataset'
trn_fnames=glob.glob(f"{data_dir}/*/*.jpg")
```

```
In [63]: import matplotlib.pyplot as plt

# plt.gray()
fig , (ax1 ,ax2 ,ax3 , ax4)=plt.subplots(nrows=1 , ncols=4)
fig.set_size_inches(10, 8)
ax1.matshow(plt.imread(trn_fnames[150])) #plt.imshow
ax2.matshow(plt.imread(trn_fnames[520]))
ax3.matshow(plt.imread(trn_fnames[1300]))
ax4.matshow(plt.imread(trn_fnames[1800]))

plt.tight_layout()
plt.show()
```



```
In [64]: def load_images_and_labels(folder, target_size=(16, 16)):
    images = []
    labels = []
    for label in os.listdir(folder):
        label_path = os.path.join(folder, label)
        if os.path.isdir(label_path):
            for filename in os.listdir(label_path):
                img_path = os.path.join(label_path, filename)
                if os.path.isfile(img_path):
                    img = Image.open(img_path).convert('L') # Convert to grayscale
                    img = img.resize(target_size) # Resize image
                    img_array = np.array(img).flatten() # Flatten image to 1D array
                    images.append(img_array)
                    labels.append(label) # Add the Label
    return np.array(images), np.array(labels)

def save_images_and_labels_to_csv(X, y, csv_file):
    # Convert the Numpy arrays to a DataFrame
    df = pd.DataFrame(X)
    df['label'] = y # Add the Labels column

    # Save DataFrame to CSV
    df.to_csv(csv_file, index=False)
    return df

folder_path = r'D:\AI_Machinelearning\datasets\persianDigits\dataset'
target_size = (16, 16) # target size

# Load images and Labels
X, y = load_images_and_labels(folder_path, target_size)

# Specify the path where you want to save the CSV file
csv_file = 'image_data.csv'

# Save to CSV
df = save_images_and_labels_to_csv(X, y, csv_file)
```

df

```
Out[64]:
```

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255	label
0	255	255	255	255	255	255	255	255	255	255	...	255	255	255	255	255	255	255	255	255	0
1	255	255	254	255	255	255	255	255	255	254	...	255	255	255	255	255	255	255	255	255	0
2	255	255	254	255	255	255	254	254	255	255	...	255	255	255	255	255	254	255	255	255	0
3	255	255	252	255	254	254	255	255	254	255	...	255	255	255	255	255	254	255	255	253	0
4	255	253	255	255	250	255	253	255	255	254	...	254	254	255	255	253	254	255	255	254	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2050	255	255	255	255	255	255	255	255	255	255	...	255	255	255	255	255	254	255	255	255	9
2051	255	255	254	255	254	255	255	255	255	255	...	255	255	255	254	255	255	255	255	254	9
2052	255	255	255	255	255	255	255	255	255	255	...	254	255	255	254	255	255	254	255	255	9
2053	255	255	255	255	255	255	244	255	255	255	...	255	255	253	249	255	254	255	255	255	9
2054	255	241	255	255	255	255	250	255	253	255	...	255	255	253	255	255	255	249	255	254	9

2055 rows × 257 columns

```
In [65]: X=np.array(df.iloc[:, :-1])
X.shape
```

```
Out[65]: (2055, 256)
```

```
In [74]: #pca
```

```
class Pca:
    def __init__(self , X , variance):
        self.X = X
        self.variance = variance

    def train(self):
        m,n = self.X.shape
        mu=self.X .mean(axis=0)
        std=self.X .std(axis=0)
        X= (self.X-mu) /std

        sigma = self.X.T @ self.X
        u, s, _ = np.linalg.svd(sigma)

        last_k=0
        for k in range(n+1):
            total_var = np.sum(s[ : k]) / np.sum(s)
            if total_var >= self.variance:
                last_k = k
                break

        u_reduced = u[ : , : last_k]
        x_proj = X@u_reduced

        X_recovered= ((x_proj @ u_reduced.T) + mu ) * std

        return sigma , u , s , u_reduced , x_proj , X_recovered , k

model= Pca(X , 0.8)
_,_, s , _ , x_proj , _, k =model.train()
```

```
In [75]: import matplotlib.pyplot as plt
import numpy as np
```

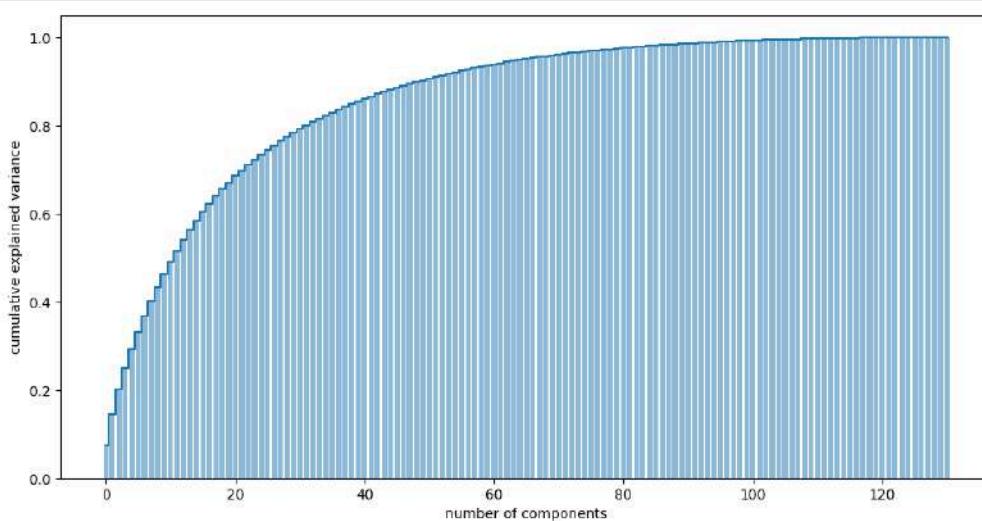
```
s_normalized = np.cumsum(s) / np.sum(s)
plt.figure(figsize=(12, 6))

# Bar plot
plt.bar(range(len(s_normalized)),
        s_normalized,
        alpha=0.5,
        align='center')

# Step plot
plt.step(range(len(s_normalized)),
        s_normalized,
        where='mid')

plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

plt.show()
```



```
In [76]: #normalize
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X=sc.fit_transform(x_proj)
```

```
#SPLIT
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test=train_test_split(X , y , random_state=45 , test_size=0.2)

In [83]: #decision tree
from sklearn.tree import DecisionTreeClassifier

digi=DecisionTreeClassifier(criterion="entropy" , max_depth=None , min_samples_split=10 , min_samples_leaf=2 )
digi.fit(X_train , y_train)

Out[83]: DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', min_samples_leaf=2,
                      min_samples_split=10)

In [84]: predict_train= digi.predict(X_train)
predict_test=digi.predict(X_test)

#accuracy
from sklearn.metrics import accuracy_score

print(f" the train accuracy is: {accuracy_score(predict_train , y_train)} and the test accuracy is: {accuracy_score(predict_test , y_test)}" )
the train accuracy is: 0.871654501216545 and the test accuracy is: 0.6472019464720195

In [85]: #random forest
from sklearn.ensemble import RandomForestClassifier

accuracy_train=[]
accuracy_test=[]
digr=RandomForestClassifier(n_estimators=100,
                           criterion="entropy" ,
                           max_depth=30 ,
                           min_samples_split=6 ,
                           min_samples_leaf=2 ,
                           bootstrap=True)

digr.fit(X_train , y_train)

Out[85]: RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=30, min_samples_leaf=2,
                      min_samples_split=6)

In [86]: predict_train= digr.predict(X_train)
predict_test=digr.predict(X_test)

#accuracy
from sklearn.metrics import accuracy_score

print(f" the train accuracy is: {accuracy_score(predict_train , y_train)} and the test accuracy is: {accuracy_score(predict_test , y_test)}" )
the train accuracy is: 0.9975669099756691 and the test accuracy is: 0.8467153284671532

In [87]: #confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sb

cn=confusion_matrix(predict_test , y_test)
sb.heatmap(cn , annot=True , cmap="magma")
plt.show()



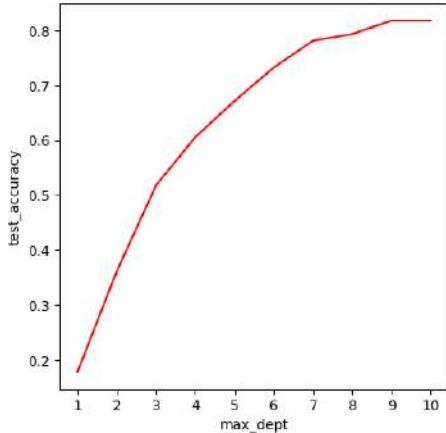

|   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|---|----|----|----|----|----|----|----|----|----|----|
| 0 | 39 | 1  | 0  | 1  | 0  | 2  | 0  | 1  | 1  | 0  |
| 1 | 0  | 36 | 0  | 0  | 1  | 0  | 1  | 0  | 0  | 2  |
| 2 | 3  | 1  | 69 | 6  | 1  | 0  | 2  | 0  | 0  | 4  |
| 3 | 1  | 0  | 0  | 32 | 0  | 0  | 0  | 0  | 0  | 0  |
| 4 | 0  | 0  | 0  | 7  | 29 | 1  | 1  | 0  | 0  | 1  |
| 5 | 3  | 0  | 0  | 2  | 0  | 27 | 2  | 0  | 0  | 0  |
| 6 | 0  | 0  | 0  | 0  | 2  | 0  | 23 | 0  | 0  | 2  |
| 7 | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 29 | 1  | 1  |
| 8 | 0  | 0  | 0  | 0  | 2  | 1  | 2  | 0  | 35 | 0  |
| 9 | 0  | 0  | 0  | 1  | 1  | 0  | 3  | 0  | 1  | 29 |
|   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |



In [88]: accuracy_train=[]
accuracy_test=[]
for i in range(1,11):
    digr=RandomForestClassifier(n_estimators=100,
                               criterion="entropy",
                               max_depth=i ,
                               min_samples_split=6 ,
                               min_samples_leaf=2 ,
                               bootstrap=True)

    digr.fit(X_train , y_train)
    predict_train= digr.predict(X_train)
    predict_test=digr.predict(X_test)
    acc_train = accuracy_score(predict_train , y_train)
    acc_test= accuracy_score(predict_test , y_test)
    accuracy_train.append(acc_train)
    accuracy_test.append(acc_test)

In [89]: plt.figure(figsize=(5,5))
plt.plot([i for i in range(1,11)] , accuracy_test , c="red" )
plt.xlabel("max_dept")
plt.ylabel("test_accuracy")
plt.xticks([i for i in range(1,11)])
plt.show()
```



```
In [90]: #without pca and normalization
X=np.array(df.iloc[:, :-1])
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=.2)

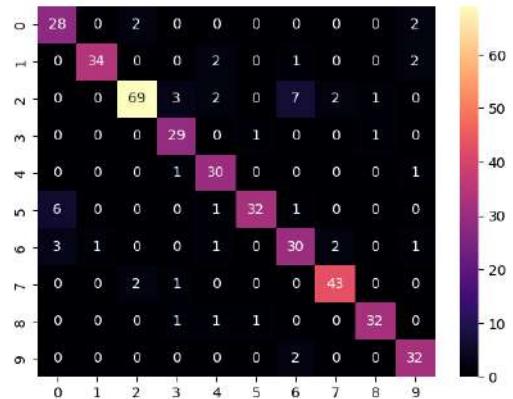
from sklearn.ensemble import RandomForestClassifier
rn=RandomForestClassifier(n_estimators=1000, criterion="entropy")
rn.fit(X_train, y_train)
prdict=rn.predict(X_test)

rn.score(X_test, y_test)
```

Out[90]: 0.8734793187347932

```
In [91]: #confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sb

cn=confusion_matrix(prdict, y_test)
sb.heatmap(cn, annot=True, cmap="magma")
plt.show()
```



# Machine learning

Naïve Bayes

Morteza khorsand



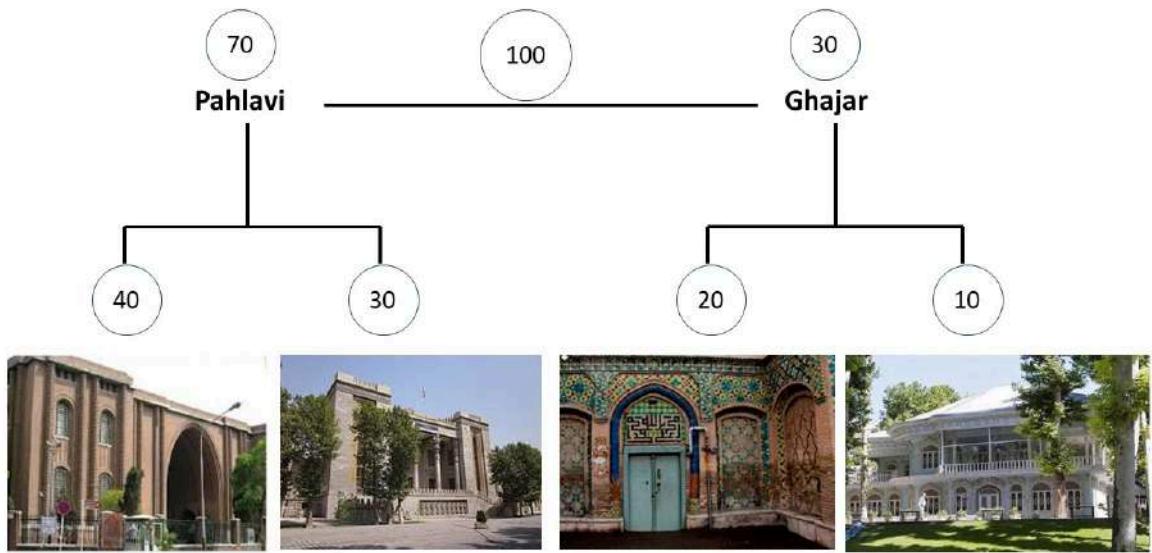
Naïve Bayes

Probability



## Naïve Bayes

### Naïve Bayes



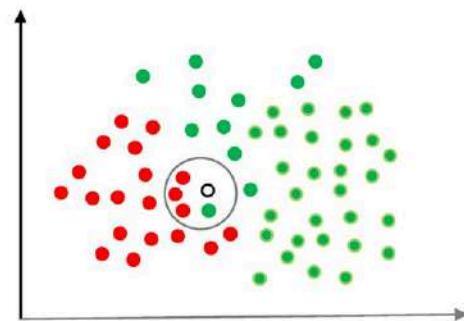
## Naïve Bayes

### Naïve Bayes GaussianNB

Posterior probability      Prior      Likelihood

$$P(Y|X) = \frac{P(Y) P(X|Y)}{P(X)}$$

Predictor prior



$$P(Y|X_1, X_2, \dots, X_n) = \frac{P(X_1, X_2, \dots, X_n|Y) P(Y)}{P(X_1, X_2, \dots, X_n)}$$

$$\hat{y} = \operatorname{argmax} P(Y) \prod_{i=1}^n P(X_i | Y)$$

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

### Example 1

```
In [1]: 1 from sklearn.datasets import load_iris
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 iris=load_iris()
6 df=pd.DataFrame(iris.data , columns=iris.feature_names)
7 df[["target"]]=iris.target
8
9 df
```

Out[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

```
In [4]: 1 X=df.iloc[:, :4]
2 y=df.iloc[:, -1]
3
4 from sklearn.model_selection import train_test_split
5
6 X_train , X_test , y_train, y_test = train_test_split(X , y ,test_size=0.2, random_state=12 )
```

```
In [5]: 1 from sklearn.naive_bayes import GaussianNB
2
3 model=GaussianNB()
4 model.fit(X_train , y_train)
```

Out[5]:

```
+ GaussianNB
GaussianNB()
```

```
In [9]: 1 y_hat=model.predict(X_test)
2
3
4
5 for i in range(len(y_hat)):
6     print(" the y_predicted is {{list(y_hat)[i]}} and the y_real is {{list(y_train)[i]}}")
```

(30,) the y\_predicted is 0 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 1  
the y\_predicted is 0 and the y\_real is 2  
the y\_predicted is 1 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 2  
the y\_predicted is 2 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 1  
the y\_predicted is 0 and the y\_real is 1  
the y\_predicted is 2 and the y\_real is 0  
the y\_predicted is 0 and the y\_real is 1  
the y\_predicted is 1 and the y\_real is 1  
the y\_predicted is 0 and the y\_real is 0  
the y\_predicted is 0 and the y\_real is 0  
the y\_predicted is 1 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 0  
the y\_predicted is 1 and the y\_real is 0  
the y\_predicted is 0 and the y\_real is 2  
the y\_predicted is 1 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 1  
the y\_predicted is 1 and the y\_real is 1  
the y\_predicted is 0 and the y\_real is 0  
the y\_predicted is 2 and the y\_real is 2  
the y\_predicted is 2 and the y\_real is 1  
the y\_predicted is 0 and the y\_real is 2  
the y\_predicted is 0 and the y\_real is 2  
the y\_predicted is 0 and the y\_real is 0

```
In [7]: 1 from sklearn.metrics import accuracy_score
2
3
4 acc=accuracy_score(y_test , y_hat)
5
6 acc
```

Out[7]: 0.9666666666666667

## Example 2

```
In [10]: 1 import numpy as np
2 from sklearn.datasets import make_blobs
3
4 X, y=make_blobs(n_samples=10000 ,n_features=10, centers=4 ,cluster_std=2,random_state=18 )
5
6
7
8 #visualize
9 import matplotlib.pyplot as plt
10 plt.scatter(X[:, :0] ,X[:, :1] , c=y)
11 plt.xlabel("feature1")
12 plt.ylabel("feature2")
13 plt.show()
```

```
In [11]: 1 from sklearn.model_selection import train_test_split
2 X_train , X_test , y_train , y_test= train_test_split(X , y , test_size=0.3 , random_state=25)
3
4
```

```
In [12]: 1 from sklearn.naive_bayes import GaussianNB
2
3 clf=GaussianNB()
4 clf.fit(X_train , y_train)
```

```
Out[12]: GaussianNB()
GaussianNB()
```

```
In [13]: 1 y_hat=clf.predict(X_test)
```

```
In [14]: 1 from sklearn.metrics import accuracy_score
2
3 acc=accuracy_score(y_hat , y_test)
4
5 print("the accuracy of train is: {0}%\n ".format(acc*100))
```

the accuracy of train is: 100.0%

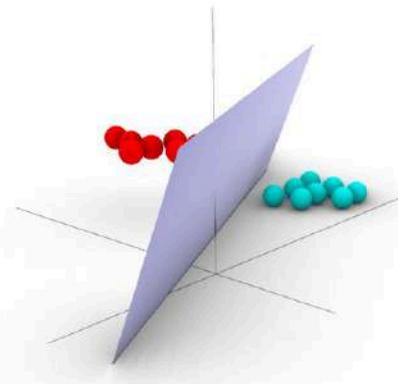
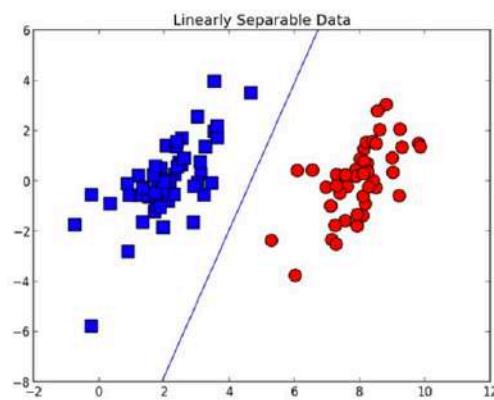
# Machine learning

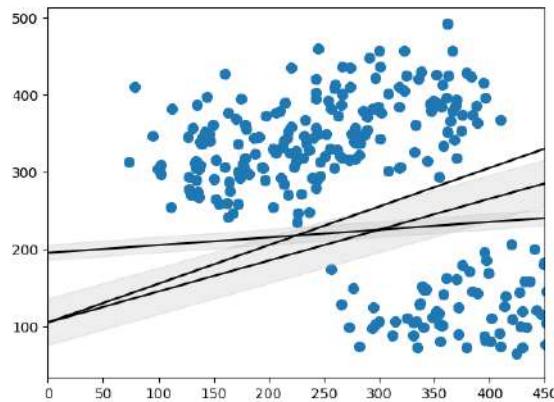
Support Vector Machine

Morteza khorsand



## Support Vector Machine

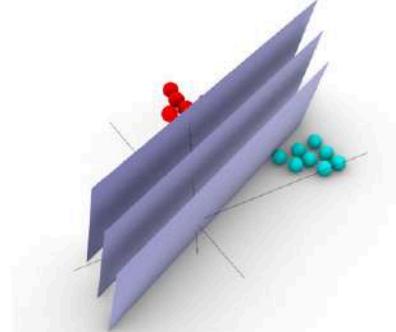
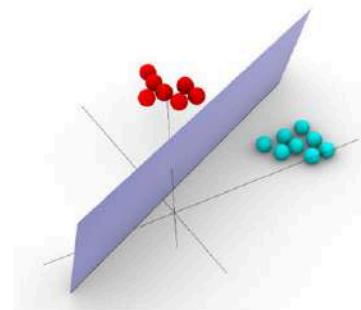
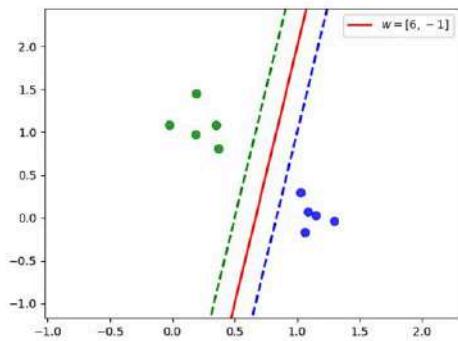




## Support Vector Machine

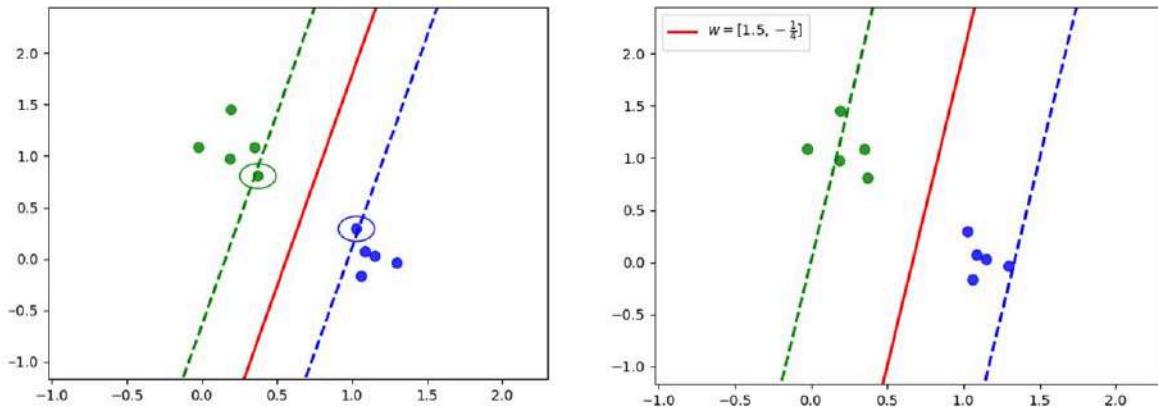
### Support Vector Machine - SVM

Wide Street The goal is to find the widest street that separates classes.

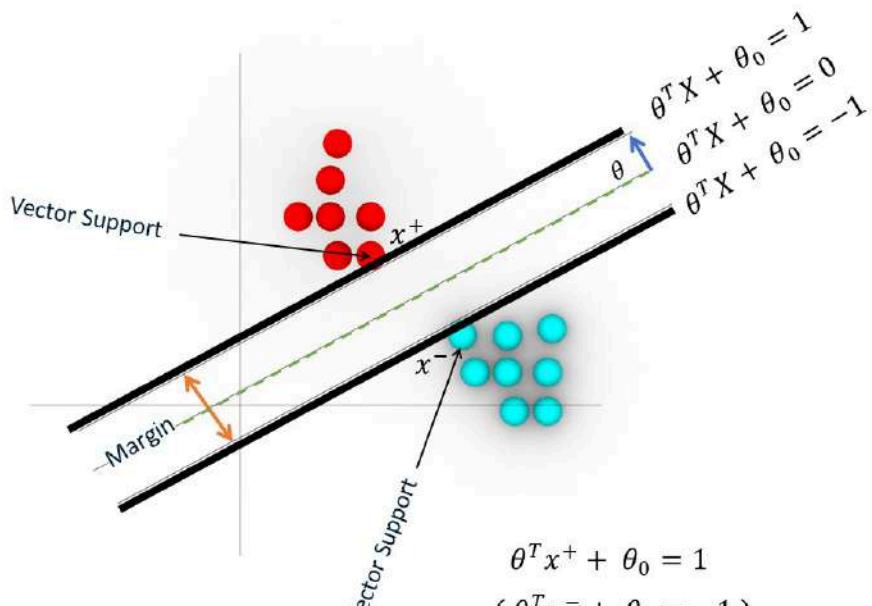


## Support Vector Machine

■ Hard Margin      ■ Soft Margin



## Support Vector Machine



$$\theta^T X + \theta_0 \geq 1 \quad \text{for } y = +1$$

$$\theta^T X + \theta_0 \leq -1 \quad \text{for } y = -1$$

$$y^i (\theta^T X + \theta_0) \geq 1$$

Condition

$$M = \frac{2}{\|\theta\|}$$

## Support Vector Machine

■ State 1

$$y^i (\theta^T X + \theta_0) \geq 1 \quad y \in \{1, -1\} \quad \text{Condition}$$

$$\text{Minimize} \quad \frac{1}{2} \|\theta\|^2 \quad \text{Objective Function}$$

■ State 2

$$!y^i (\theta^T X + \theta_0) \geq 1 \quad y \in \{1, -1\}$$

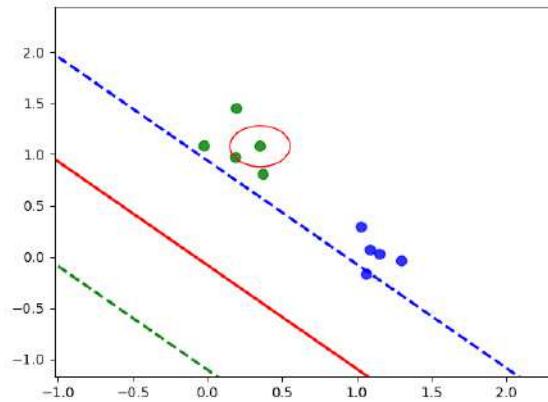
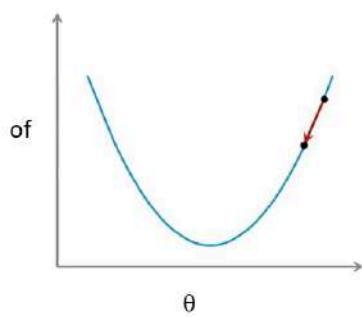
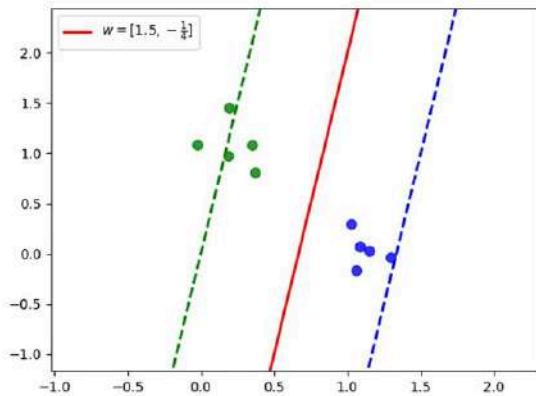
$$y^i \theta^T X + y^i \theta_0$$

$$\boxed{\frac{\partial}{\partial \theta} = \theta}$$

$$\theta^{(new)} := \theta + \alpha (-\nabla J) \quad M = \frac{2}{\|\theta\|}$$

$$\boxed{\frac{\partial}{\partial \theta} = y^i X}$$

$$\boxed{\frac{\partial}{\partial \theta_0} = y^i}$$



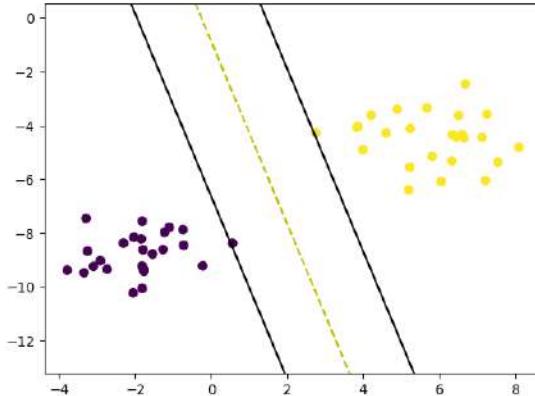
In [1]:

```

1 import numpy as np
2
3 class SVM:
4
5     def __init__(self, learning_rate=0.001, n_iters=500):
6         self.lr = learning_rate
7         self.n_iters = n_iters
8         self.w = None
9         self.b = None
10
11    def fit(self, X, y):
12        n_samples, n_features = X.shape
13
14        y_ = np.where(y <= 0, -1, 1)
15
16        # init weights
17        self.w = np.zeros(n_features)
18        self.b = 0
19
20        for _ in range(self.n_iters):
21            for idx, x_i in enumerate(X):
22                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
23
24                if condition:
25                    self.w -= self.lr * (2*1/n_samples * self.w)
26                else:
27                    self.w -= self.lr * (2 * 1/n_samples * self.w - np.dot(x_i, y_[idx]))
28                    self.b -= self.lr * y_[idx]
29
30    def predict(self, X):
31        approx = np.dot(X, self.w) - self.b
32        return np.sign(approx)
33
34
35 # Testing
36 if __name__ == "__main__":
37     # Imports
38     from sklearn.model_selection import train_test_split
39     from sklearn import datasets
40     import matplotlib.pyplot as plt
41
42     X, y = datasets.make_blobs(
43         n_samples=50, n_features=2, centers=2, cluster_std=1.05, random_state=40)
44
45     y = np.where(y == 0, -1, 1)
46
47     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
48
49     clf = SVM()
50     clf.fit(X_train, y_train)
51     predictions = clf.predict(X_test)
52
53     def accuracy(y_true, y_pred):
54         accuracy = np.sum(y_true == y_pred) / len(y_true)
55         return accuracy
56
57     print("SVM classification accuracy", accuracy(y_test, predictions))
58
59     def visualize_svm():
60         def get_hyperplane_value(x, w, b, offset):
61             return (-w[0] * x + b + offset) / w[1]
62
63         fig = plt.figure()
64         ax = fig.add_subplot(1, 1, 1)
65         plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)
66
67         x0_1 = np.amin(X[:, 0])
68         x0_2 = np.amax(X[:, 0])
69
70         x1_1 = get_hyperplane_value(x0_1, clf.w, clf.b, 0)
71         x1_2 = get_hyperplane_value(x0_2, clf.w, clf.b, 0)
72
73         x1_1_m = get_hyperplane_value(x0_1, clf.w, clf.b, -1)
74         x1_2_m = get_hyperplane_value(x0_2, clf.w, clf.b, -1)
75
76         x1_1_p = get_hyperplane_value(x0_1, clf.w, clf.b, 1)
77         x1_2_p = get_hyperplane_value(x0_2, clf.w, clf.b, 1)
78
79         ax.plot([x0_1, x0_2], [x1_1, x1_2], "y--")
80         ax.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "k")
81         ax.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "k")
82
83         x1_min = np.amin(X[:, 1])
84         x1_max = np.amax(X[:, 1])
85         ax.set_xlim([x1_min - 3, x1_max + 3])
86
87         plt.show()
88
89     visualize_svm()
90

```

SVM classification accuracy 1.0

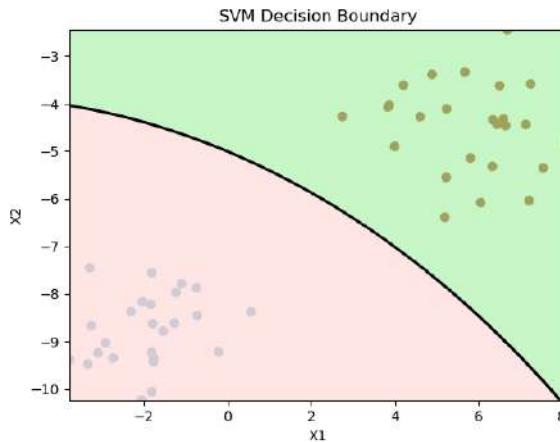


```

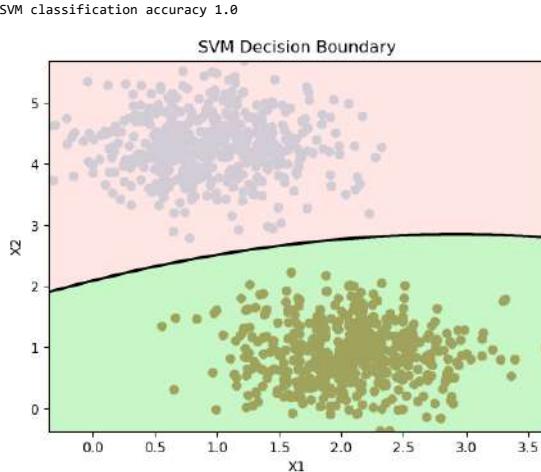
In [2]: M
  1 import numpy as np
  2 from sklearn.model_selection import train_test_split
  3 from sklearn import datasets
  4 import matplotlib.pyplot as plt
  5
  6 class SVM:
  7     def __init__(self, learning_rate=0.001, n_iters=1000, degree=2):
  8         self.lr = learning_rate
  9         self.degree = degree
 10         self.n_iters = n_iters
 11         self.w = None
 12         self.b = None
 13
 14     def normalize(self, X):
 15         return (X - X.mean(axis=0)) / X.std(axis=0)
 16
 17     # Transform method
 18     def transform(self, X):
 19         X_norm = self.normalize(X)
 20         num_samples, num_features = X_norm.shape
 21         X_transform = np.empty((num_samples, 0))
 22         for i in range(1, self.degree + 1):
 23             X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
 24         return X_transform
 25
 26     def fit(self, X, y):
 27         Xtansnorm = self.transform(X)
 28         n_samples, n_features = Xtansnorm.shape
 29
 30         y_ = np.where(y <= 0, -1, 1)
 31
 32         # init weights
 33         self.w = np.zeros(n_features)
 34         self.b = 0
 35
 36         for _ in range(self.n_iters):
 37             for idx, x_i in enumerate(Xtansnorm):
 38                 condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
 39                 if condition:
 40                     self.w -= self.lr * (2 * 1/n_samples * self.w)
 41                 else:
 42                     self.w -= self.lr * (2 * 1/n_samples * self.w - np.dot(x_i, y_[idx]))
 43                     self.b -= self.lr * y_[idx]
 44
 45     def predict(self, X):
 46         X_transformed = self.transform(X)
 47         approx = np.dot(X_transformed, self.w) - self.b
 48         return np.sign(approx)
 49
 50
 51
 52     def get_hyperplane_values(x, w, b, degree):
 53         # Calculate the value of the decision boundary for each degree
 54         values = []
 55         for d in range(degree + 1):
 56             if d == 0:
 57                 value = (-b / w[0]) - (w[1] / w[0]) * x ** (d)
 58             else:
 59                 value += (w[1] / w[0]) * x ** (d)
 60         return value
 61
 62
 63
 64
 65
 66
 67     def visualize_svm(clf, X, y):
 68         # Plot the data points
 69         plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
 70
 71         # Increase the resolution of the grid
 72         x0_min, x0_max = np.min(X[:, 0]), np.max(X[:, 0])
 73         x1_min, x1_max = np.min(X[:, 1]), np.max(X[:, 1])
 74         xx, yy = np.meshgrid(np.linspace(x0_min, x0_max, 1000), np.linspace(x1_min, x1_max, 1000))
 75         Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
 76         Z = Z.reshape(xx.shape)
 77
 78         # Plot the decision boundary
 79         plt.contour(xx, yy, Z, levels=[-1, 0, 1], colors=['#FFCCCB', '#90EE90'], alpha=0.5)
 80         plt.contour(xx, yy, Z, colors='k', levels=[0], linestyles=['-'], linewidths=2)
 81
 82         # Set plot labels and show
 83         plt.xlabel('X1')
 84         plt.ylabel('X2')
 85         plt.title('SVM Decision Boundary')
 86         plt.show()
 87
 88
 89
 90
 91
 92     # Testing
 93     if __name__ == "__main__":
 94         X, y = datasets.make_blobs(
 95             n_samples=50, n_features=2, centers=2, cluster_std=1.05, random_state=40)
 96         y = np.where(y == 0, -1, 1)
 97
 98         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
 99
100         clf = SVM()
101         clf.fit(X_train, y_train)
102         predictions = clf.predict(X_test)
103
104     def accuracy(y_true, y_pred):
105         accuracy = np.sum(y_true == y_pred) / len(y_true)
106         return accuracy
107
108     print("SVM classification accuracy", accuracy(y_test, predictions))
109
110     visualize_svm(clf, X, y)
111
112
113

```

SVM classification accuracy 1.0



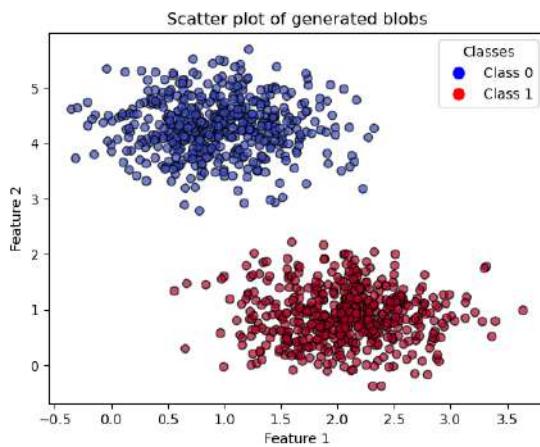
```
In [3]: 1 import numpy as np
2 from sklearn.datasets import make_blobs
3 import matplotlib.pyplot as plt
4 #normalize
5 from sklearn.preprocessing import StandardScaler
6
7
8 # Testing
9 if __name__ == "__main__":
10     X, y = make_blobs(n_samples=1000, n_features=2, centers=2, random_state=0, cluster_std=0.5)
11     y = np.where(y == 0, -1, 1)
12
13     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
14
15
16
17     sc= StandardScaler()
18     X_train= sc.fit_transform(X_train)
19     X_test= sc.transform(X_test)
20
21
22
23     clf = SVC()
24     clf.fit(X_train, y_train)
25     predictions = clf.predict(X_test)
26
27
28     def accuracy(y_true, y_pred):
29         accuracy = np.sum(y_true == y_pred) / len(y_true)
30         return accuracy
31
32     print("SVM classification accuracy", accuracy(y_test, predictions))
33
34     visualize_svm(clf, X, y)
35
36
37
38
```



## Sklearn

Example 1

```
In [4]: 1 import numpy as np
2 from sklearn.datasets import make_blobs
3 import matplotlib.pyplot as plt
4
5 # Generate data
6 X, y = make_blobs(n_samples=1000, n_features=2, centers=2, random_state=0, cluster_std=0.5)
7
8 # Create scatter plot
9 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
10
11 # Add Labels and title
12 plt.xlabel('Feature 1')
13 plt.ylabel('Feature 2')
14 plt.title('Scatter plot of generated blobs')
15
16 # Add legend
17 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
18                   plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')],
19                   title='Classes')
20
21 # Show plot
22 plt.show()
23
24
```



```
In [5]: 1 #train and test split
2 from sklearn.model_selection import train_test_split
3
4 X_train , X_test, y_train , y_test=train_test_split(X , y)
5
6 #normalize
7 from sklearn.preprocessing import StandardScaler
8
9 sc= StandardScaler()
10 x_train= sc.fit_transform(X_train)
11 x_test=sc.transform(X_test)
```

```
In [6]: 1 from sklearn.svm import SVC
2
3 model=SVC(kernel="linear")
4
5 model.fit(x_train , y_train)
6
7
```

Out[6]:

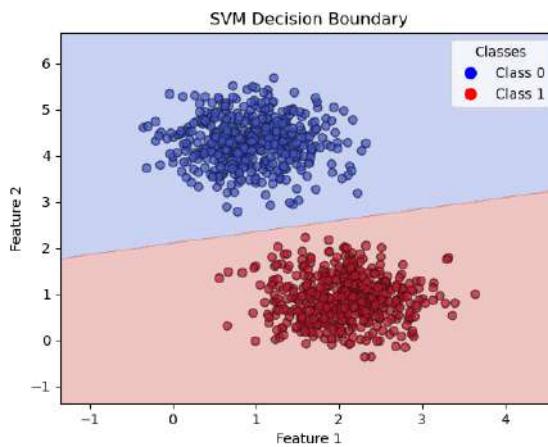
```
SVC
SVC(kernel='linear')
```

```
In [7]: 1 #prediction
2 y_predict=model.predict(x_test)
3
4
5 for i in range(len(y_predict)):
6     if y_predict[i] != y_test[i]:
7         print(f"The predicted y is: {y_predict[i]} and the real y is: {y_test[i]}")
8
9
10
```

```
In [8]: 1 #evaluate
2 from sklearn.metrics import accuracy_score
3
4 print(accuracy_score(y_predict , y_test))
```

1.0

```
In [9]: 1 # Create a mesh grid to plot the decision boundary
2 h = .02 # step size in the mesh
3 x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
4 y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
5 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
6
7 # Plot decision boundary
8 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
9
10 # Scale the mesh grid points to match the training data scale
11 Z = model.predict(sc.transform(np.c_[xx.ravel(), yy.ravel()]))
12 Z = Z.reshape(xx.shape)
13
14 # Plot the decision boundary
15 plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
16
17 plt.xlabel('Feature 1')
18 plt.ylabel('Feature 2')
19 plt.title('SVM Decision Boundary')
20 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
21             plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')],
22             title='Classes')
23 plt.show()
24
```



Example 2

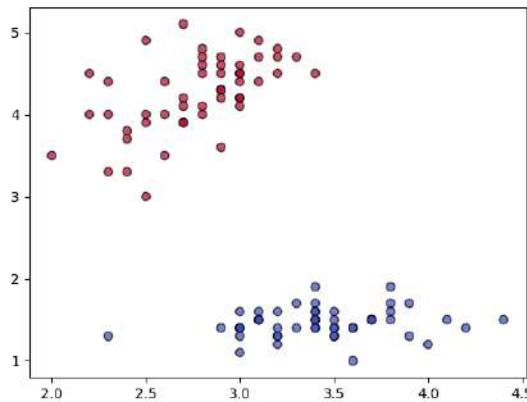
```
In [10]: 1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.svm import SVC
6 from sklearn.datasets import make_moons
7
8 # Generate data
9 X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
10
11
```

Example 3

```
In [11]: 1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import load_iris
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.svm import SVC
7
8
9 # Load the Iris dataset
10 data = load_iris()
11 X = data.data[:, [1,2]]
12 y = data.target
13
14
```

```
In [12]: 1 # Select only classes 0 and 1 for binary classification
2 binary_mask = y < 2
3 X = X[binary_mask]
4 y = y[binary_mask]
5
6
7
```

```
In [14]: 1 # Plot decision boundary
2 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
3 plt.show()
```



```
In [15]: 1 # Train and test split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
3
4
```

```
In [16]: 1 # Normalize
2 sc = StandardScaler()
3 X_train = sc.fit_transform(X_train)
4 X_test = sc.transform(X_test)
```

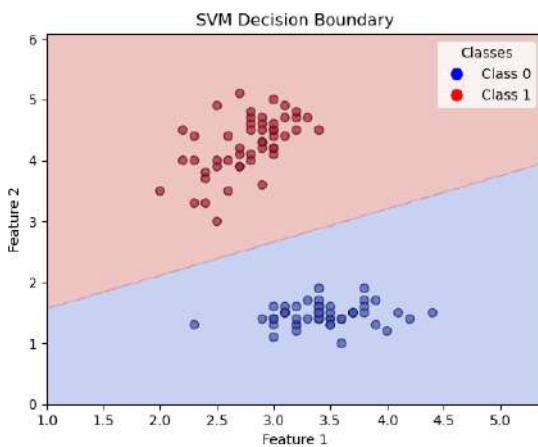
```
In [17]: 1 # Train SVM model
2 model = SVC(kernel="linear")
3 model.fit(X_train, y_train)
```

Out[17]: `SVC  
SVC(kernel='linear')`

```
In [18]: 1 model.score(X_test, y_test)
```

Out[18]: 1.0

```
In [19]: 1 # Create a mesh grid to plot the decision boundary
2 h = .02 # step size in the mesh
3 x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
4 y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
5 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
6
7 # Plot decision boundary
8 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
9
10 # Scale the mesh grid points to match the training data scale
11 Z = model.predict(sc.transform(np.c_[xx.ravel(), yy.ravel()]))
12 Z = Z.reshape(xx.shape)
13
14 # Plot the decision boundary
15 plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
16
17 plt.xlabel('Feature 1')
18 plt.ylabel('Feature 2')
19 plt.title('SVM Decision Boundary')
20 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
21             plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')],
22             title='Classes')
23 plt.show()
```

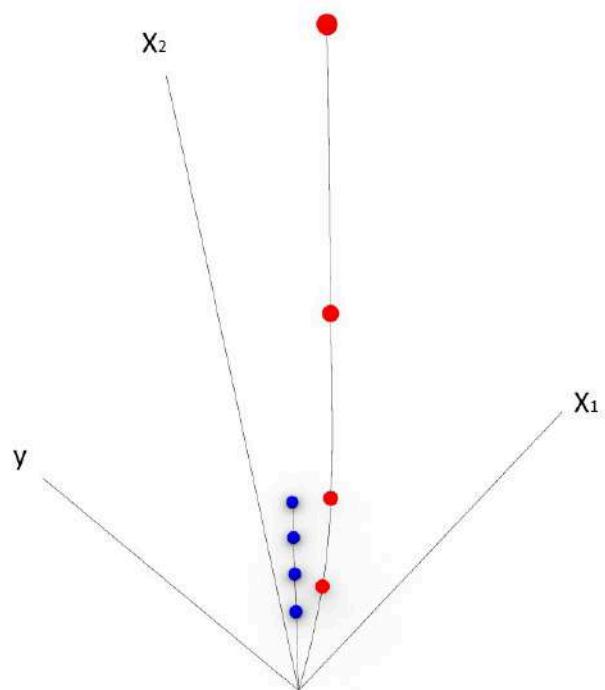


## Kernel Trick

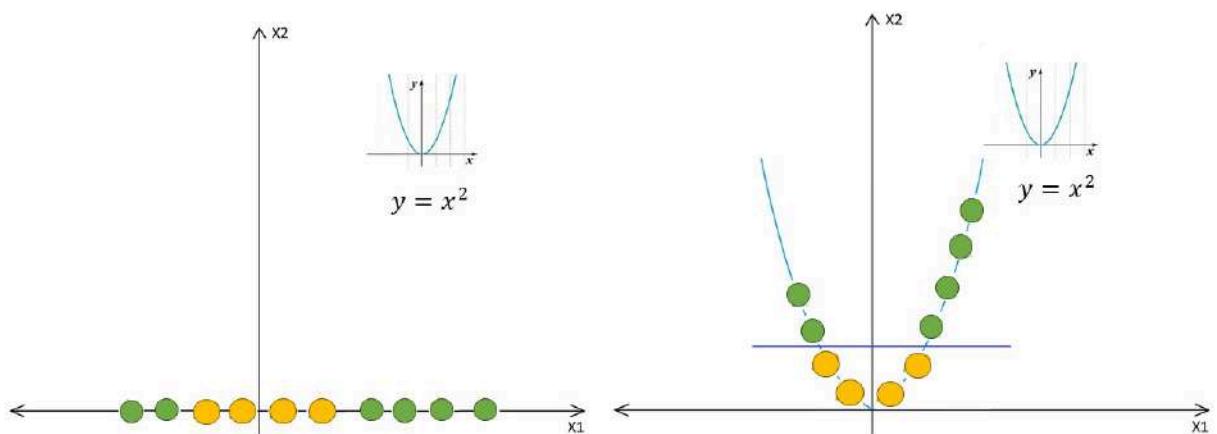
## Support Vector Machine

x1	x2	y
1	1	1
2	4	2
3	9	3
4	16	4
5	25	5

$$\Phi: x \rightarrow \varphi(x)$$

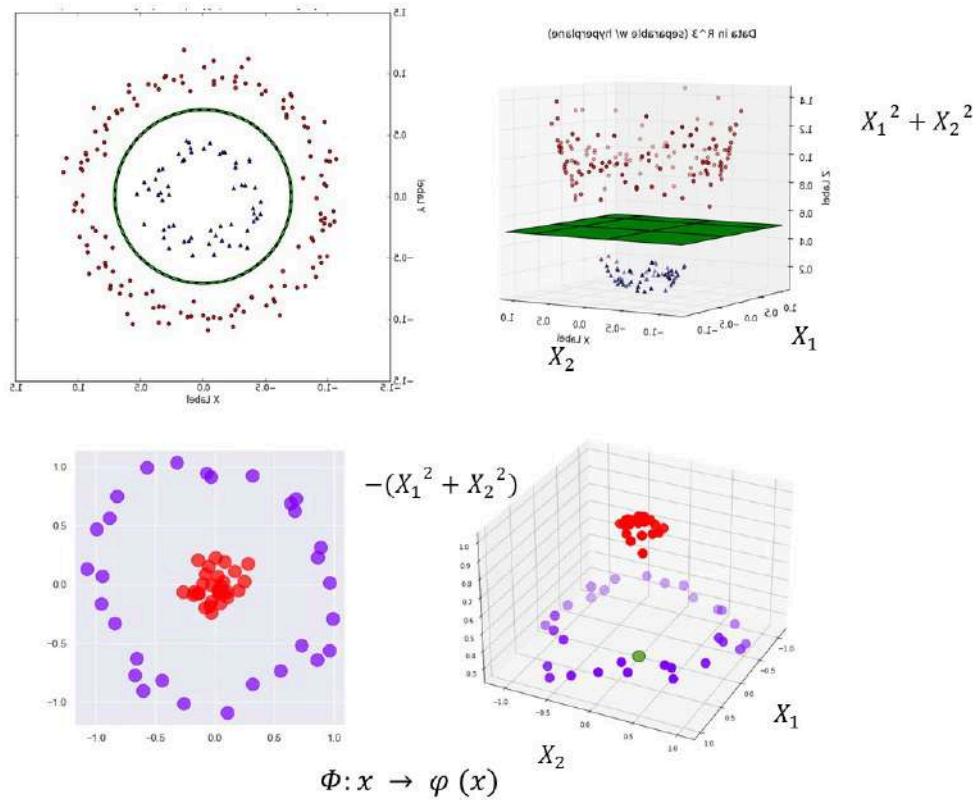


## Support Vector Machine

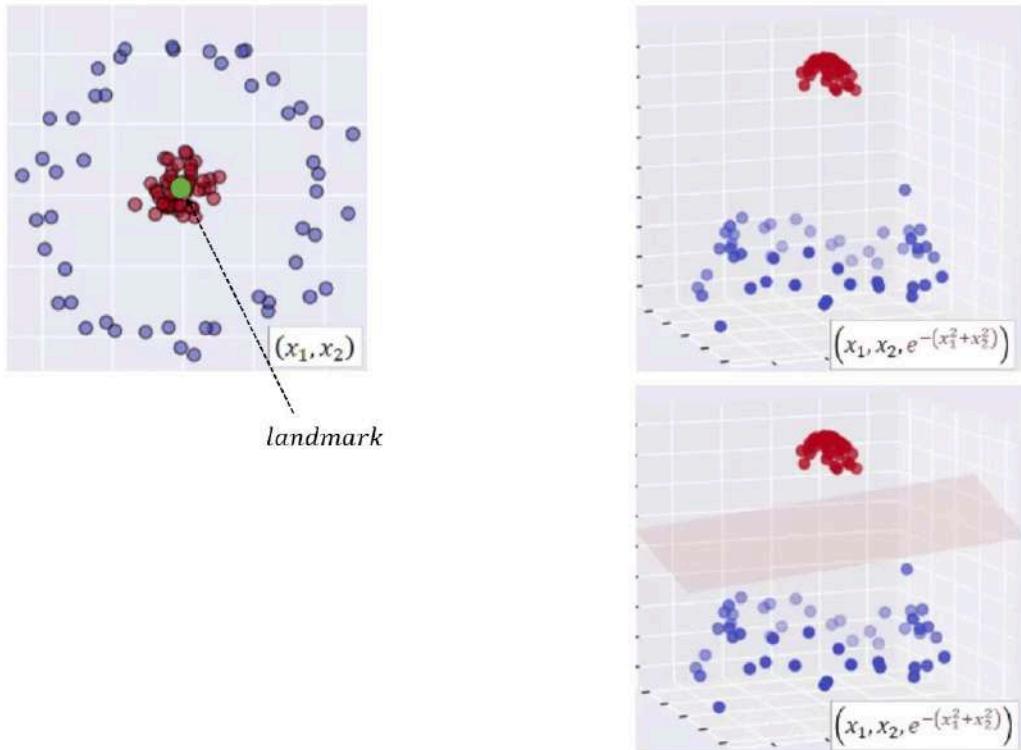


x1	X2 = ( $x^2$ )	y
1	1	0
2	4	0
3	9	1
4	16	1
5	25	1

## Support Vector Machine



## Support Vector Machine



## Support Vector Machine

	X <sub>1</sub>	X <sub>2</sub>	y
$x^{(1)}$	1	1	0
$x^{(2)}$	2	4	0
$x^{(3)}$	3	9	1

$$\Phi: x \rightarrow \varphi(x)$$

	X <sub>1</sub>	X <sub>2</sub>	$X_1^2$	$X_2^2$	$X_1 * X_2$	y
$x^{(1)}$	1	1	1	1	1	0
$x^{(2)}$	2	4	4	16	8	0
$x^{(3)}$	3	9	9	64	27	1

## Support Vector Machine

■ kernel trick

■ Type of Kernels

	X <sub>1</sub>	X <sub>2</sub>	y
$x^{(1)}$	1	1	0
$x^{(2)}$	2	4	0
$x^{(3)}$	3	9	1

1. Linear Kernel
2. Polynomial Kernel
3. RBF Kernel  
(Radial Basis Function)

$k(x^{(1)}, x^{(2)}) = (\text{sim})^d$   
 $k(x^{(1)}, x^{(2)}) = (\text{e})^{-\gamma \text{sim}}$

$$\begin{bmatrix} x^{(1)} \cdot x^{(1)} & x^{(1)} \cdot x^{(2)} & x^{(1)} \cdot x^{(3)} \\ x^{(2)} \cdot x^{(1)} & x^{(2)} \cdot x^{(2)} & x^{(2)} \cdot x^{(3)} \\ x^{(3)} \cdot x^{(1)} & x^{(3)} \cdot x^{(2)} & x^{(3)} \cdot x^{(3)} \end{bmatrix}_{m \times m}$$

$$\Phi: x \rightarrow \varphi(x)$$

$$\begin{bmatrix} k(x^{(1)}, x^{(1)}) & k(x^{(1)}, x^{(2)}) & k(x^{(1)}, x^{(3)}) \\ k(x^{(2)}, x^{(1)}) & k(x^{(2)}, x^{(2)}) & k(x^{(2)}, x^{(3)}) \\ k(x^{(3)}, x^{(1)}) & k(x^{(3)}, x^{(2)}) & k(x^{(3)}, x^{(3)}) \end{bmatrix}_{m \times m} \quad \text{Gram matrix}$$

$$k(x^{(i)}, x^{(j)})$$

## Support Vector Machine

### Polynomial Kernel parameters

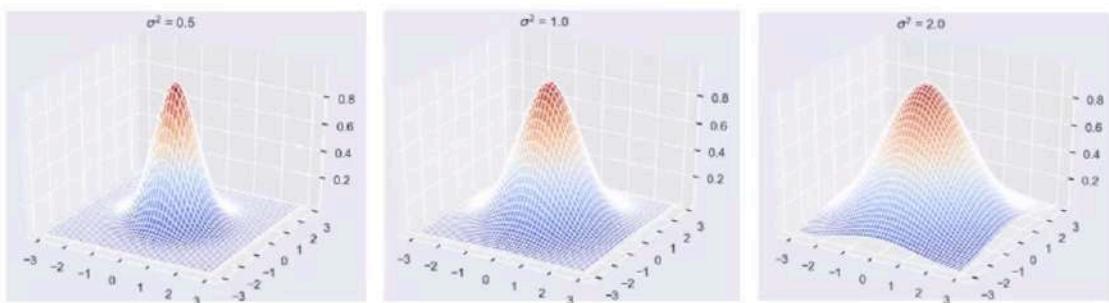
$d, \gamma$

### RBF Kernel parameters

$C, \gamma$

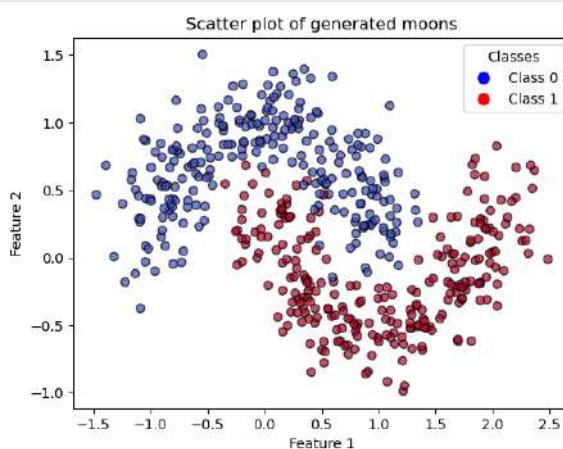
(e)  $-\gamma sim$

$$\text{Minimize } \frac{1}{2} \|\theta\|^2 + C \varepsilon$$



```
In [20]: 1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.svm import SVC
6 from sklearn.datasets import make_moons
7
8 # Generate data
9 X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
```

```
In [21]: 1 # Create scatter plot
2 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
3 plt.xlabel('Feature 1')
4 plt.ylabel('Feature 2')
5 plt.title('Scatter plot of generated moons')
6 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
7                   plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')],
8             title='Classes')
9 plt.show()
```



```
In [22]: 1 # Train and test split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
3
4 # Normalize
5 sc = StandardScaler()
6 X_train = sc.fit_transform(X_train)
7 X_test = sc.transform(X_test)
```

```
In [23]: 1 # Train SVM model
2 model = SVC(kernel="poly", C=1, degree=3, gamma=1)
3 model.fit(X_train, y_train)
```

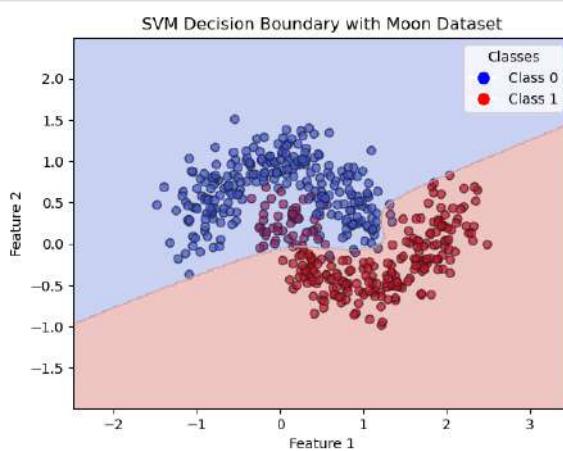
```
Out[23]: SVC(C=1, gamma=1, kernel='poly')
```

```
In [24]: 1 y_predict=model.predict(X_test)
```

```
In [25]: 1 #evaluate
2 from sklearn.metrics import accuracy_score
3
4 print(accuracy_score(y_predict, y_test))
```

0.87

```
In [26]: 1 # Create a mesh grid to plot the decision boundary
2 h = .02 # step size in the mesh
3 x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
4 y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
5 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
6
7 # Plot decision boundary
8 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
9
10 # Scale the mesh grid points to match the training data scale
11 Z = model.predict(sc.transform(np.c_[xx.ravel(), yy.ravel()]))
12 Z = Z.reshape(xx.shape)
13
14 # Plot the decision boundary
15 plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
16
17 plt.xlabel('Feature 1')
18 plt.ylabel('Feature 2')
19 plt.title('SVM Decision Boundary with Moon Dataset')
20 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
21             plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')],
22             title='Classes')
23 plt.show()
```



```
In [27]: 1 # Train SVM model
2 model = SVC(kernel="rbf", C=1, gamma=1)
3 model.fit(X_train, y_train)
```

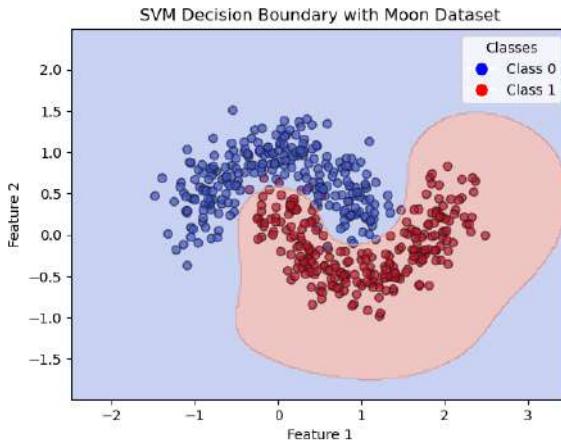
```
Out[27]: SVC(C=1, gamma=1)
```

```
In [28]: 1 y_predict=model.predict(X_test)
```

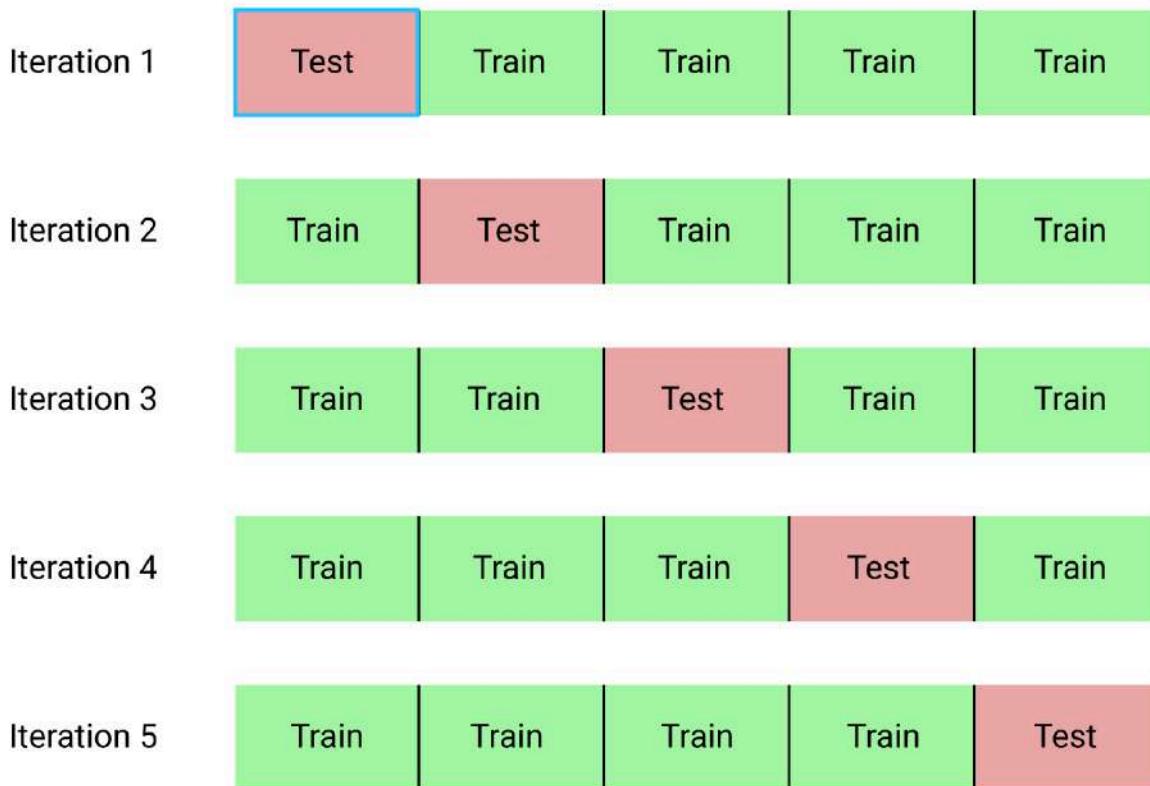
```
In [29]: 1 #evaluate
2 from sklearn.metrics import accuracy_score
3
4 print(accuracy_score(y_predict, y_test))
```

0.97

```
In [30]: 
1 # Create a mesh grid to plot the decision boundary
2 h = .02 # step size in the mesh
3 x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
4 y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
5 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
6
7 # Plot decision boundary
8 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
9
10 # Scale the mesh grid points to match the training data scale
11 Z = model.predict(sc.transform(np.c_[xx.ravel(), yy.ravel()]))
12 Z = Z.reshape(xx.shape)
13
14 # Plot the decision boundary
15 plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
16
17 plt.xlabel('Feature 1')
18 plt.ylabel('Feature 2')
19 plt.title('SVM Decision Boundary with Moon Dataset')
20 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
21             plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')],
22             title='Classes')
23 plt.show()
```

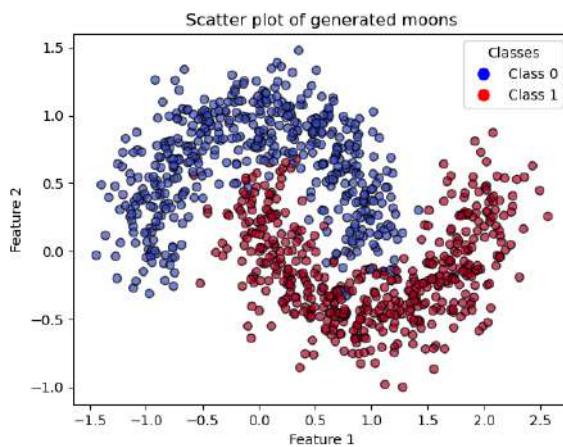


## K-FOLD



```
In [31]: 
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.svm import SVC
5 from sklearn.datasets import make_moons
6
7 # Generate data
8 X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
```

```
In [32]: 
1 # Create scatter plot
2 plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k', alpha=0.7)
3 plt.xlabel('Feature 1')
4 plt.ylabel('Feature 2')
5 plt.title('Scatter plot of generated moons')
6 plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=10, label='Class 0'),
7                  plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='r', markersize=10, label='Class 1')], title='Classes')
8
9 plt.show()
```



```
In [33]: 
1 from sklearn.model_selection import cross_val_score
2
3 # Train SVM model
4 model = SVC(kernel="rbf", C=1, gamma=1)
5
6 cv_scores=cross_val_score(model , X ,y , cv=5)
7
8 print(cv_scores)
9
10 print(round(np.mean(cv_scores) , 2))
11
12
```

[0.97 0.95 0.99 0.975 0.965]  
0.97

## Example

```
In [34]: 
1 import pandas as pd
2 from sklearn.datasets import load_wine
3
4 wine=load_wine()
5
6 df=pd.DataFrame(wine. data , columns= wine.feature_names)
7
8
9 df[ "target" ] = wine.target
10
11 df
12
```

Out[34]:

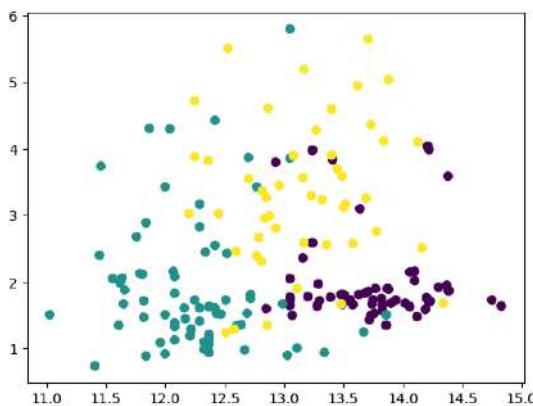
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0	2
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750.0	2
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835.0	2
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840.0	2
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560.0	2

178 rows × 14 columns

```
In [51]: 
1 X= wine. data
2 y=wine.target
3
```

In [52]:  1 #visualize

```
1 #visualize
2
3 import matplotlib.pyplot as plt
4
5 plt.scatter(X[:, 0] , X[:,1] , c=y )
6 plt.show()
7
```



In [56]: ►

```
1 from sklearn.svm import SVC
2 from sklearn.model_selection import cross_val_score
3
4 # Train SVM model
5 model = SVC(kernel="poly" , gamma=1 , C=1)
6
7 cv_scores=cross_val_score(model , X , y , cv=5)
8
9 print(cv_scores)
10
11 print(round(np.mean(cv_scores) , 2))
```

```
[0.97222222 0.86111111 0.97222222 1. 1.  
0.96
```

In [57]: ►

```
1 from sklearn.datasets import load_breast_cancer  
2  
3 data=load_breast_cancer()  
4  
5 df=pd.DataFrame(data.data , columns=data.feature_names)  
6  
7  
8 df[ "target"] =data.target  
9  
0 df
```

Out[57]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613
3	11.42	20.38	77.58	386.1	0.14250	0.26390	0.24140	0.10520	0.2597	0.09744	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871

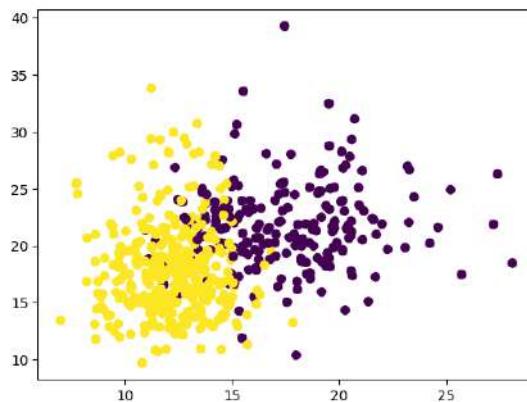
569 rows x 31 columns

T- [58]: N

```
1 X= data.data  
2 y=data.target
```

```
In [59]:
```

```
1 import matplotlib.pyplot as plt
2
3 plt.scatter(X[:, 0] , X[:,1] , c=y )
4 plt.show()
```



```
In [61]:
```

```
1 from sklearn.svm import SVC
2 from sklearn.model_selection import cross_val_score
3
4 # Train SVM model
5 model = SVC(kernel="poly" ,degree=2 , gamma=1 , C=1)
6
7 cv_scores=cross_val_score(model , X ,y , cv=5)
8
9 print(cv_scores)
10
11 print(round(np.mean(cv_scores) , 2))
```

```
[0.94736842 0.93859649 0.98245614 0.96491228 0.98230088]
```

```
0.96
```

```
In [64]:
```

```
1 from sklearn.datasets import load_iris
2
3 iris= load_iris()
4
5 df=pd.DataFrame(iris.data , columns= iris.feature_names)
6
7 df[["target"]]= iris.target
8
9 df.head(10)
```

```
Out[64]:
```

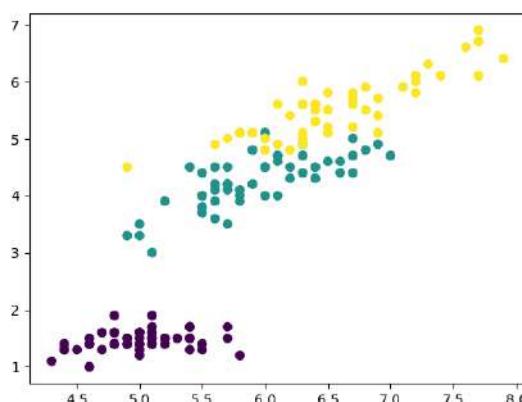
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0

```
In [66]:
```

```
1 X=iris.data
2 y=iris.target
3
```

```
In [70]:
```

```
1 plt.scatter(X[:, 0] , X[:, 2] , c=y)
2 plt.show()
```



```
In [72]:
```

```
1 from sklearn.model_selection import cross_val_score
2 from sklearn.svm import SVC
3
```

```
In [79]: 1 model=SVC(kernel="linear" , gamma=1 , C=1)
2 print(np.mean(cv_scores))
0.9631268436578171
```

# Machine learning

logistic Regression

Morteza khorsand



## Logistic regression

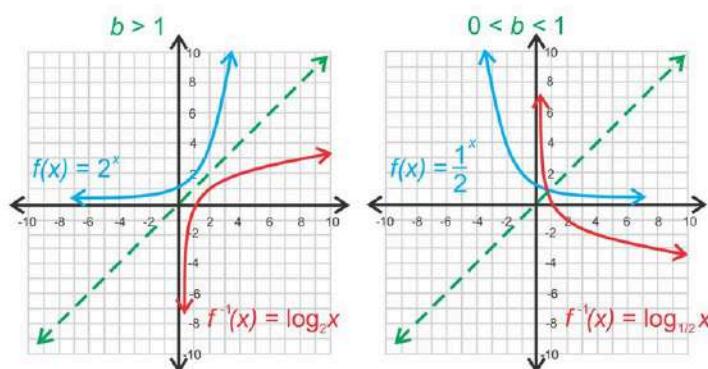
### Exponentiation

Exponentiation is a mathematical operation, written as  $b^n$ , involving two numbers, the base  $b$ , and the exponent or power  $n$ , and pronounced as "b (raised) to the (power of) n". When  $n$  is a positive integer, exponentiation corresponds to repeated multiplication of the base:  $b^n$  is the product of multiplying  $n$  bases.

$$y = b^n (b \times b \times b \times \dots \times b)$$

### logarithm

In mathematics, the logarithm is the inverse function of exponentiation. That means the logarithm of a given number  $x$  is the exponent to which another fixed number, the base  $b$ , must be raised, to produce that number  $x$ .



$$y^{-1} : n = \log_b y$$

### Logarithms and Exponents

*The log is the exponent*

$$a^x = y \Leftrightarrow \log_b y = x$$

base

$a > 0, a \neq 1, y \neq 0$

Example:  
 $2^5 = 32$  means  $\log_2 32 = 5$



### Logarithmic Identities

$$x = \log_2 64$$

$$\log_b 1 = 0$$

$$x = \log_{\sqrt{5}} 125$$

$$\log_b b = 1$$

$$\log_b MN = \log_b M + \log_b N$$

$$\frac{4}{21} = \log_{a^3\sqrt{a}} X$$

$$\log_b \frac{M}{N} = \log_b M - \log_b N$$

$$\log_b m^n = n \log_b m$$



$$\ln x = \log_e x$$

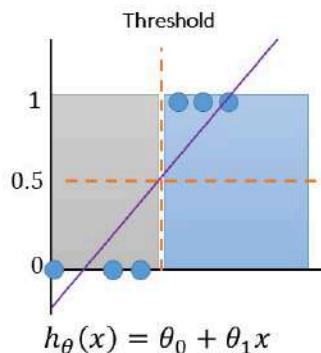
$$\log_3 243 = \log_3(9 \cdot 27) = \log_3 9 + \log_3 27 = 2 + 3 = 5$$

$$e = (1 + \frac{1}{n})^n = 2.7182 \dots$$

$$\log_2 16 = \log_2 \frac{64}{4} = \log_2 64 - \log_2 4 = 6 - 2 = 4$$

$$\log_2 64 = \log_2 (2^6) = 6 \log_2 2 = 6$$

X	y
0	0
0.3	0
0.4	0
0.6	1
0.7	1
1	1
$\bar{x} = 5.0$	$\bar{y} = 5.0$



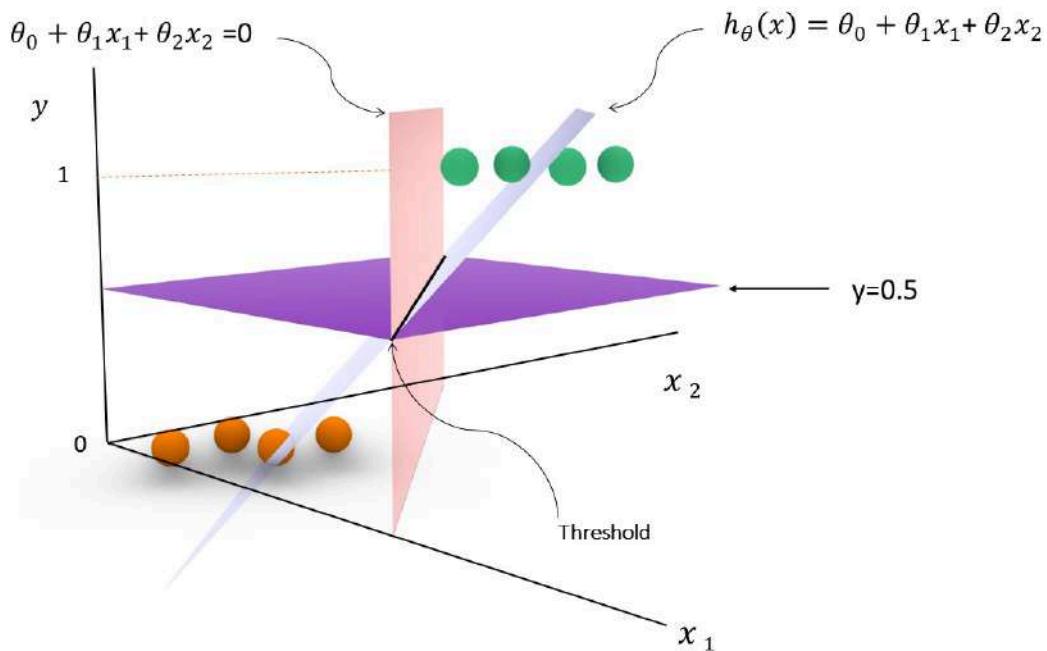
```

1 enumerator= 0
2 denominator=0
3
4 x=[0,0.3,0.4,0.6,0.7,1]
5 y=[0,0,0,1,1,1]
6
7 x_bar= sum(x)/ len(x)
8 y_bar= sum(y) / len(y)
9 for i in range(len(x)):
10     enumerator+= (x[i]- x_bar) * (y[i] - y_bar)
11     denominator+= (x[i] - x_bar)**2
12
13 theta1= enumerator / denominator
14 theta0= y_bar - theta1 * x_bar
15 print (f" x_bar:{x_bar} ----- y_bar:{y_bar} ----- theta1:{round(theta1 , 1)} -----theta0:{round(theta0, 1)}")
16
17 x_bar:0.5 ----- y_bar:0.5 ----- theta1:1.3 -----theta0:-0.2

```

$$\theta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

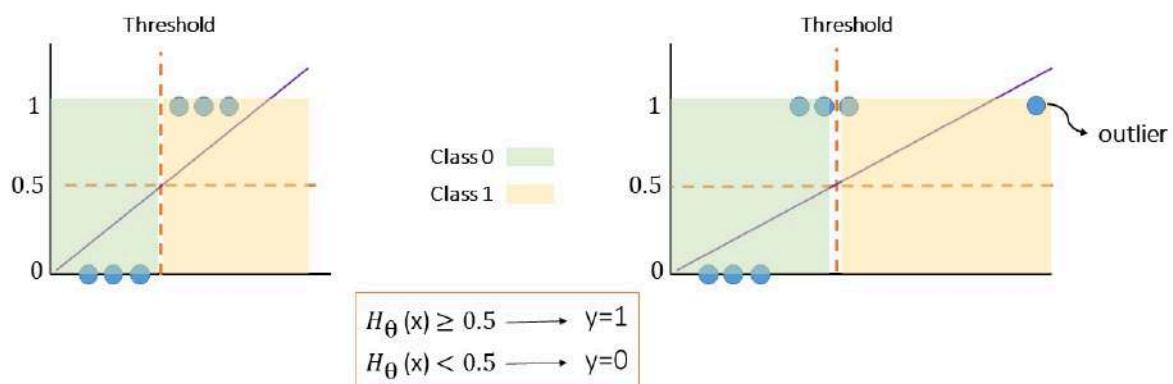
## Logistic regression



## Logistic regression

### Binary classification

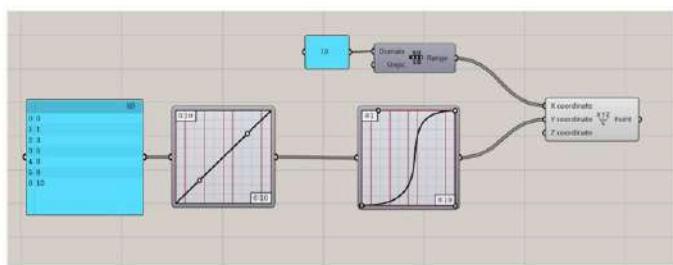
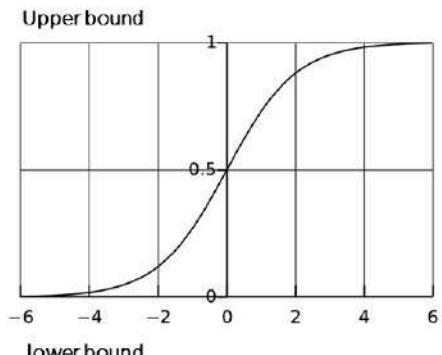
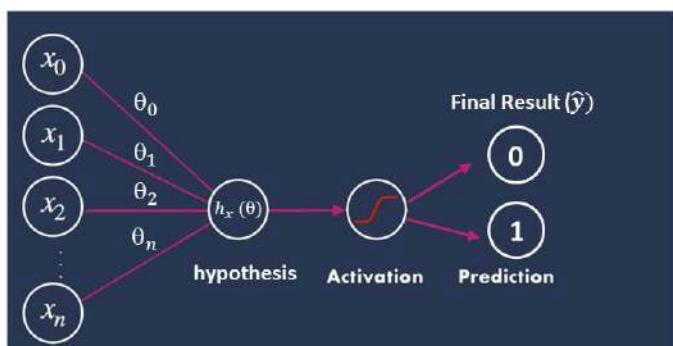
Application	Observation	0	1
Medical Diagnosis	Patient	Healthy	Diseased
Email Analysis	Email	Not Spam	Spam
Financial Data Analysis	Transaction	Not Fraud	Fraud
Marketing	Website visitor	Won't Buy	Will Buy
Image Classification	Image	Hotdog	Not Hotdog



## Logistic regression

### Sigmoid function

$$h_{\theta}(x) = g(\theta^T @ X) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)$$



$$g(y) = \frac{1}{1 + e^{-y}}$$

$\theta^T @ X > 0$  Class A

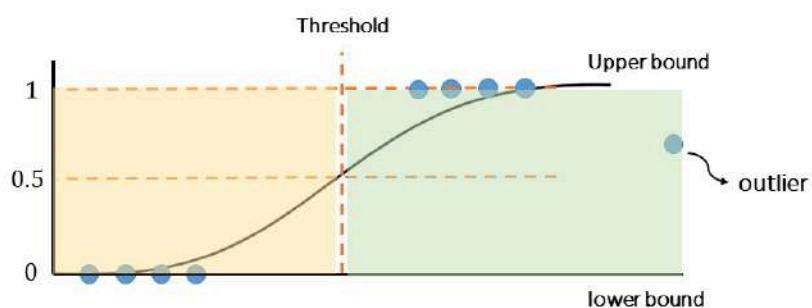
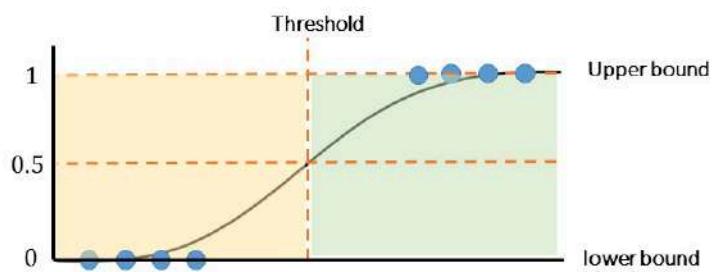
$\theta^T @ X < 0$  Class B

$\theta^T @ X = 0$  Threshold

## Logistic regression

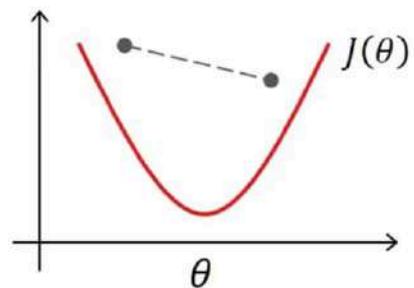
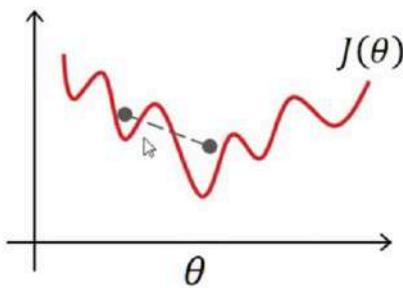
### Sigmoid function

#### Single-variate logistic regression



## Logistic regression

### Why we do not use MSE

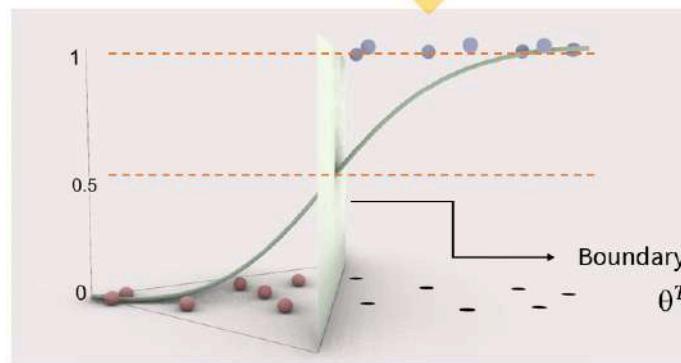
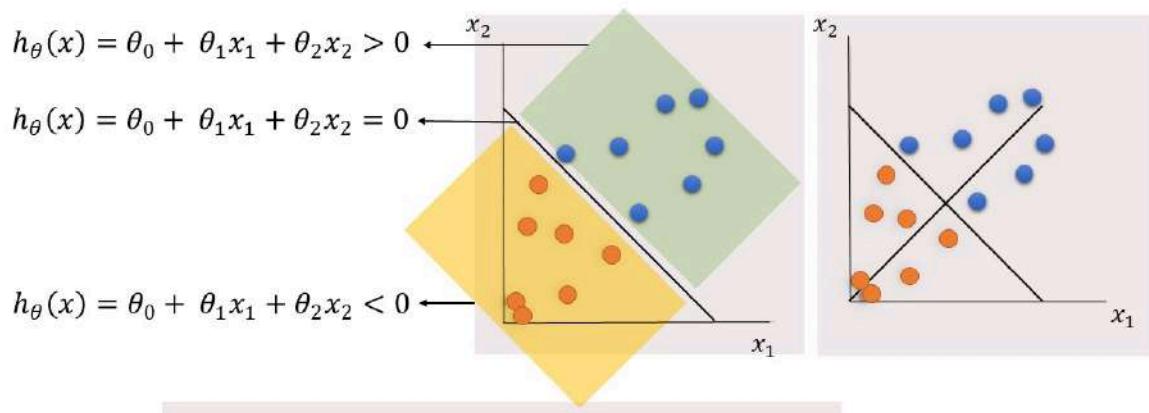


$$\frac{1}{m} \sum (g(\theta^T @ X) - y)^2$$

$$\frac{1}{m} \sum (\theta^T @ X - y)^2$$

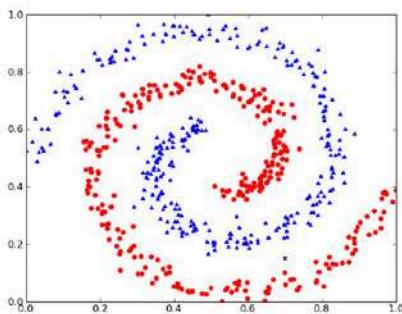
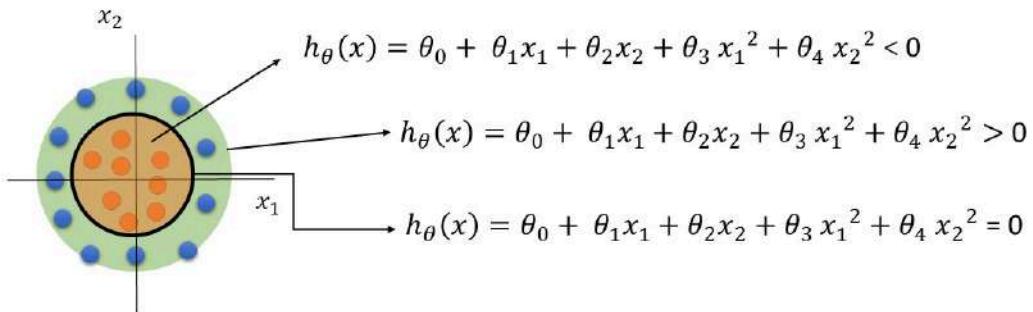
## Logistic regression

### Binary classification - double-variate logistic regression



## Logistic regression

### Binary classification - double-variate logistic regression



## Logistic regression

### Bernoulli distribution

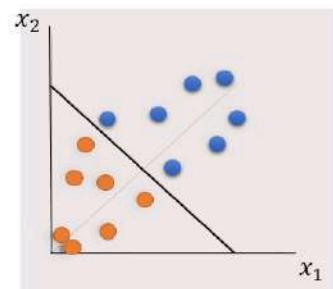
discrete probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability q=1-p

$$Y \sim Ber(p) \longrightarrow p(Y = y) = \begin{cases} p & y=1 \\ 1-p & y=0 \end{cases}$$

$$p(Y = y) = p^y (1-p)^{1-y} \quad y = 0, 1$$

$$Y \sim Ber(p) \longrightarrow p(Y = y | x, \theta) = \begin{cases} h_\theta(x) & y=1 \\ 1 - h_\theta(x) & y=0 \end{cases}$$

$$p(Y = y | x, \theta) = h_\theta(x)^y (1 - h_\theta(x))^{1-y} \quad y=0,1$$



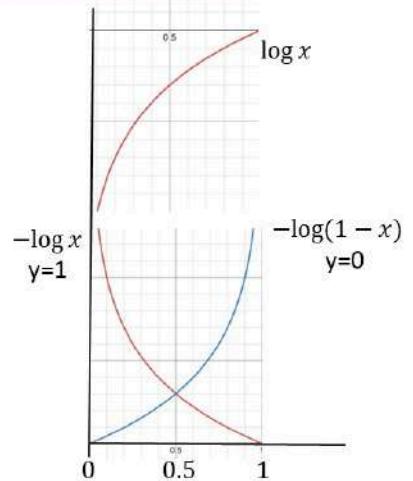
## Logistic regression

### Maximum Likelihood Estimation (MLE)

In statistics, maximum likelihood estimation (MLE) is a method of estimating the parameters of an assumed probability distribution, given some observed data. This is achieved by maximizing a likelihood function so that, under the assumed statistical model, the observed data is most probable.

$$L(\theta) = p(Y = y | x, \theta) = \prod_{i=1}^m p(y^{(i)} | x^{(i)}, \theta) = \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

$$\begin{aligned}\log L(\theta) &= \log \left[ \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \right] \\ &= \sum_{i=1}^m \log \left( h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \right) \\ &= \sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\end{aligned}$$



Binary Cross-Entropy Loss or the Log Loss function

$$J(\theta) = -L(\theta)$$

## Logistic regression

### Derivative of loglikelihood

$$J=0 \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$J=1 \quad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

$$J=0 \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \sum_{i=1}^m \left( \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)}} - y^{(i)} \right)$$

$$J=1 \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \sum_{i=1}^m \left( \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)}} - y^{(i)} \right) \cdot x^{(i)}$$

### Vectorized

$$\nabla J(\theta_1) = X^T (g(\theta^T @ X) - y)$$

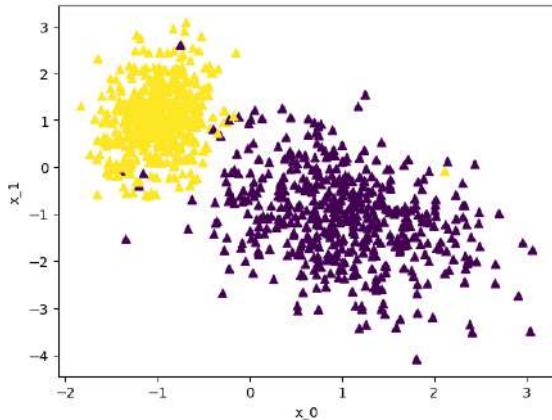
$$\nabla J(\theta_0) = \sum (g(\theta^T @ X) - y)$$

```
In [1]: M
1 import numpy as np
2
3 class LogisticRegression:
4     def __init__(self, X, y, iteration, learning_rate, stopping_threshold, degree):
5         self.X = X
6         self.y = y.reshape(-1, 1) # Reshape y to be a column vector
7         self.iteration = iteration
8         self.learning_rate = learning_rate
9         self.stopping_threshold = stopping_threshold
10        self.degree = degree
11
12    def normalize(self, X):
13        mean = X.mean(axis=0)
14        std = X.std(axis=0)
15        std[std == 0] = 1 # To avoid division by zero, set zero std to one
16        return (X - mean) / std
17
18    def transform(self, X):
19        X_norm = self.normalize(X)
20        num_samples, num_features = X_norm.shape
21        X_transform = np.empty((num_samples, 0))
22        for i in range(1, self.degree + 1):
23            X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
24        return X_transform
25
26    def Sigmoid(self, Z):
27        return 1. / (1. + np.exp(-Z))
28
29    def train(self):
30        Xtansnorm = self.transform(self.X)
31        m, n = Xtansnorm.shape
32
33        # Weight initialization
34        current_weight = np.random.rand(n, 1)
35        current_bias = np.zeros((1, 1))
36
37        costs = []
38        previous_cost = None
39
40        for i in range(self.iteration):
41            # Making predictions
42            Z = Xtansnorm @ current_weight + current_bias
43            y_hat = self.Sigmoid(Z)
44
45            # Clipping the predictions to avoid Log(0) error
46            y_hat = np.clip(y_hat, 1e-15, 1 - 1e-15)
47
48            # Calculating the current cost
49            loss = -np.mean(self.y * np.log(y_hat) + (1 - self.y) * np.log(1 - y_hat))
50
51            # If the change in cost is less than or equal to stopping_threshold we stop the gradient descent
52            if previous_cost is not None and abs(previous_cost - loss) <= self.stopping_threshold:
53                break
54
55            previous_cost = loss
56            costs.append(loss)
57
58            # Calculating the gradients
59            weight_derivative = Xtansnorm.T @ (y_hat - self.y) / m
60            bias_derivative = np.sum(y_hat - self.y) / m
61
62            # Updating weights and bias
63            current_weight -= self.learning_rate * weight_derivative
64            current_bias -= self.learning_rate * bias_derivative
65
66        return costs, current_weight, current_bias
67
68    def predict(self, X):
69        _, w, b = self.train()
70        Xtansnorm = self.transform(X)
71        Z = Xtansnorm @ w + b
72        y_hat = self.Sigmoid(Z)
73        predictions = (y_hat >= 0.5).astype(int)
74        return predictions
75
76    def plot_decision_boundary(self):
77        # Define the x and y limits of the plot
78        x_min, x_max = self.X[:, 0].min() - 1, self.X[:, 0].max() + 1
79        y_min, y_max = self.X[:, 1].min() - 1, self.X[:, 1].max() + 1
80        xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
81        grid = np.c_[xx.ravel(), yy.ravel()]
82
83        # Generate predictions over the grid
84        probs = self.predict(grid).reshape(xx.shape)
85
86        plt.contourf(xx, yy, probs, alpha=0.8, cmap=plt.cm.Spectral)
87        plt.scatter(self.X[:, 0], self.X[:, 1], c=self.y.ravel(), s=40, edgecolors='k', cmap=plt.cm.Spectral)
88        plt.xlim(x_min, x_max)
89        plt.ylim(y_min, y_max)
90        plt.xlabel('Feature 1')
91        plt.ylabel('Feature 2')
92        plt.title('Logistic Regression Decision Boundary')
93        plt.show()
94
95
96
97
98
99
```

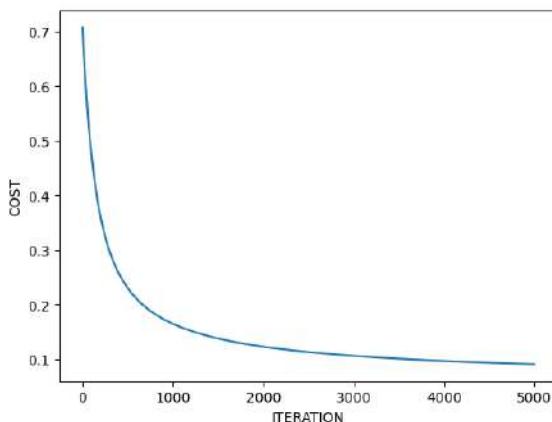
## Exercise 1

```
In [2]: M
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 #create dataset
5 from sklearn.datasets import make_classification
6
7 X,y = make_classification(n_samples=1000, n_features= 2,
8                           random_state=153,n_clusters_per_class=1 , n_redundant=0)
9
10 from sklearn.model_selection import train_test_split
11
12 X_train , X_test , y_train , y_test = train_test_split(X , y , test_size=0.2 , random_state=2)
```

```
In [3]: 1 plt.scatter(X[ :,0] , X[ :,1] ,c= y , marker= "^^" )
2 plt.xlabel("x_0")
3 plt.ylabel("x_1")
4 plt.show()
```

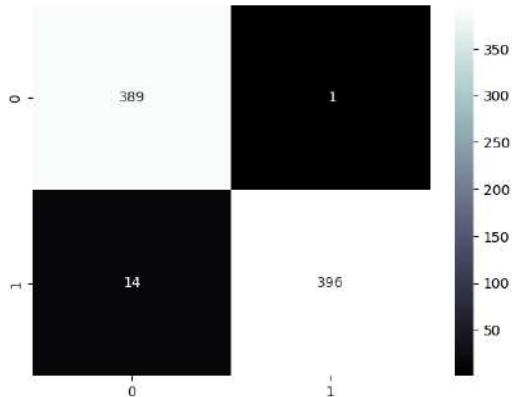


```
In [4]: 1 model= LogisticRegression(X_train , y_train ,5000,0.01, 10e-12,1 )
2
3 costs, weights, biases = model.train()
4
5 import matplotlib.pyplot as plt
6 plt.plot([i for i in range (len(costs))] , costs)
7 plt.xlabel("ITERATION")
8 plt.ylabel ("COST")
9 plt.show()
10
11
12
```

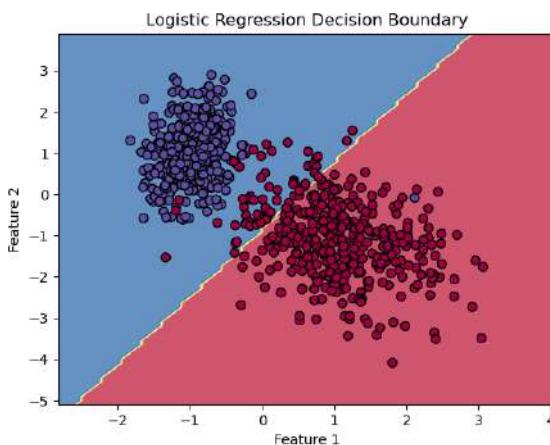


```
In [5]: 1 prediction= model.predict(X_test)
2
3 from sklearn.metrics import accuracy_score , confusion_matrix
4 import seaborn as sb
5
6 accuracy= accuracy_score(prediction , y_test)
7 print("Accuracy test:", accuracy)
8
9 predict_train= model.predict(X_train)
10 accuracy_train= accuracy_score(predict_train , y_train)
11 print("Accuracy train:", accuracy_train)
12
13 cm1=confusion_matrix(prediction , y_test)
14 cm2=confusion_matrix(predict_train , y_train)
15 #sb.heatmap(cm1 , annot=True , cmap="bone" , fmt='2g' , )
16 sb.heatmap(cm2 , annot=True , cmap="bone" , fmt='2g' , )
17
18 plt.show()
19
20
21
```

Accuracy test: 0.99  
Accuracy train: 0.98125



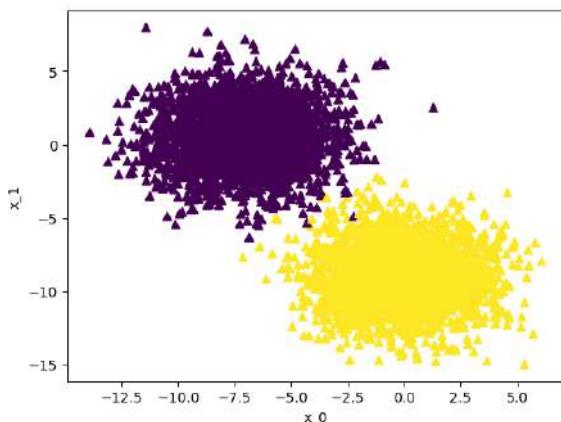
```
In [6]: 1 model.plot_decision_boundary()
```



## Exercise 2

```
In [7]: 1 from sklearn.datasets import make_blobs
2
3 X,y = make_blobs(n_samples=5000 , n_features= 2,
4                   random_state=153, centers=2, cluster_std=2, )
5
6 from sklearn.model_selection import train_test_split
7
8 X_train , X_test , y_train , y_test = train_test_split(X , y , test_size=0.2 , random_state=2)
```

```
In [8]: 1 plt.scatter(X[ :,0] , X[ :,1] ,c= y , marker= "^^" )
2 plt.xlabel("x_0")
3 plt.ylabel("x_1")
4 plt.show()
```



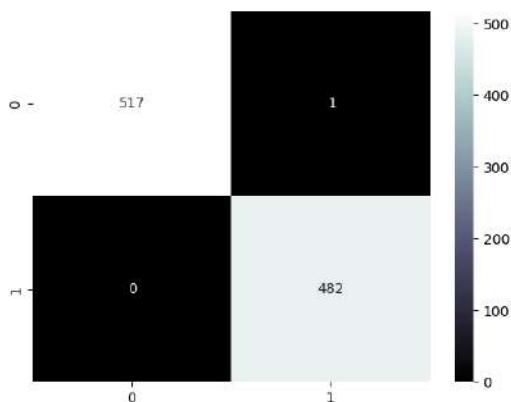
```
In [9]: 1 import matplotlib.pyplot as plt
2 import numpy as np
3 import seaborn as sb
4 from sklearn.metrics import accuracy_score
5
```

```
In [10]: 1 model2= LogisticRegression(X_train , y_train , 5000 , learning_rate= 0.01 , stopping_threshold=1e-6 , degree=1)
2 costs , weight , bias=model2.train()
```

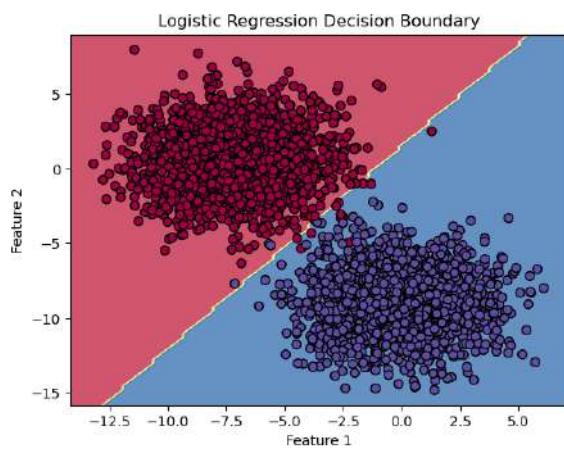
```
In [11]: 1 trainpredict=model2.predict(X_train)
2 testpredict=model2.predict(X_test)
3
4 #accuracy
5 print(f" train_accuracy is: {accuracy_score(trainpredict , y_train)} and test_accuracy is: {accuracy_score(testpredict,y_test)}")
```

train\_accuracy is: 0.9975 and test\_accuracy is: 0.999

```
In [12]: 1 from sklearn.metrics import confusion_matrix
2
3 m=confusion_matrix(testpredict , y_test )
4 sb.heatmap(m , annot= True , cmap="bone" , fmt="2g")
5 plt.show()
```

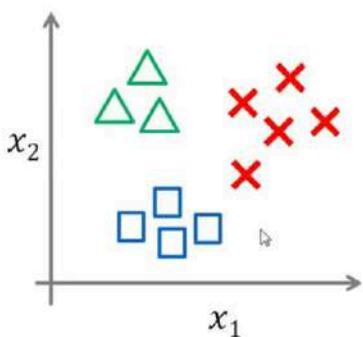


```
In [13]: 1 model2.plot_decision_boundary()
```

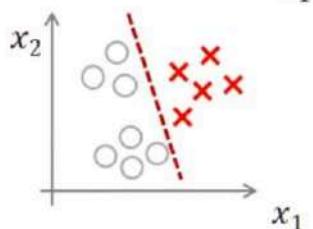
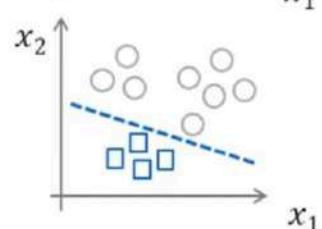
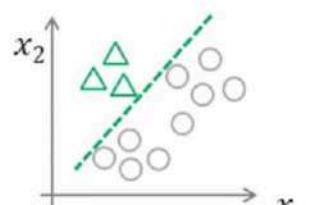


## Logistic regression

### one-vs-rest



$$y = \operatorname{argmax} h_{\theta}^{(i)} (x)$$



In [7]:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class LogisticRegression:
5     def __init__(self, X, y, iteration, learning_rate, stopping_threshold, degree):
6         self.X = X
7         self.y = y
8         self.iteration = iteration
9         self.learning_rate = learning_rate
10        self.classes = np.unique(y)
11        self.classifiers = {}
12        self.stopping_threshold = stopping_threshold
13        self.degree = degree
14        self.classes = np.unique(y)
15        self.classifiers = {}
16
17    def normalize(self, X):
18        mean = X.mean(axis=0)
19        std = X.std(axis=0)
20        std[std == 0] = 1           # To avoid division by zero, set zero std to one
21        return (X - mean) / std
22
23    def transform(self, X):
24        X_norm = self.normalize(X)
25        num_samples, num_features = X_norm.shape
26        X_transform = np.empty((num_samples, 0))
27        for i in range(1, self.degree + 1):
28            X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
29        return X_transform
30
31    def sigmoid(self, Z):
32        return 1. / (1. + np.exp(-Z))
33
34    def train_binary_classifier(self, X, y):
35        Xtansnorm = self.transform(X)
36        m, n = Xtansnorm.shape
37
38        # Weight initialization
39        current_weight = np.random.rand(n, 1)
40        current_bias = np.zeros((1, 1))
41
42        costs = []
43        previous_cost = None
44
45        for i in range(self.iteration):
46            # Making predictions
47            Z = Xtansnorm @ current_weight + current_bias
48            y_hat = self.sigmoid(Z)
49
50            # Clipping the predictions to avoid Log(0) error
51            y_hat = np.clip(y_hat, 1e-15, 1 - 1e-15)
52
53            # Calculating the current cost
54            loss = - np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
55
56            # If the change in cost is less than or equal to stopping_threshold we stop the gradient descent
57            if previous_cost is not None and abs(previous_cost - loss) <= self.stopping_threshold:
58                break
59
60            previous_cost = loss
61            costs.append(loss)
62
63            # Calculating the gradients
64            weight_derivative = Xtansnorm.T @ (y_hat - y) / m
65            bias_derivative = np.sum(y_hat - y) / m
66
67            # Updating weights and bias
68            current_weight -= self.learning_rate * weight_derivative
69            current_bias -= self.learning_rate * bias_derivative
70
71        return current_weight, current_bias, costs
72
73    def train(self):
74        for cls in self.classes:
75            y_binary = (self.y == cls).astype(int).reshape(-1, 1)          # کلاس مناسب با وای رو مشخص میکنه
76            weight, bias, costs = self.train_binary_classifier(self.X, y_binary)  # کلاس سدی دو تابی رو انجام میده
77            self.classifiers[cls] = (weight, bias, costs)                  # در دیکشنری بالا ذخیره میکنه
78
79    def predict_proba(self, X):
80        Xtansnorm = self.transform(X)
81        probabilities = np.zeros((Xtansnorm.shape[0], len(self.classes)))
82
83        for i, cls in enumerate(self.classes):
84            weight, bias, _ = self.classifiers[cls]
85            Z = Xtansnorm @ weight + bias
86            probabilities[:, i] = self.sigmoid(Z).ravel()
87
88        return probabilities
89
90    def predict(self, X):
91        probabilities = self.predict_proba(X)
92        predictions = np.argmax(probabilities, axis=1)
93        return self.classes[predictions]
94
95
96
97
98
99
100
101

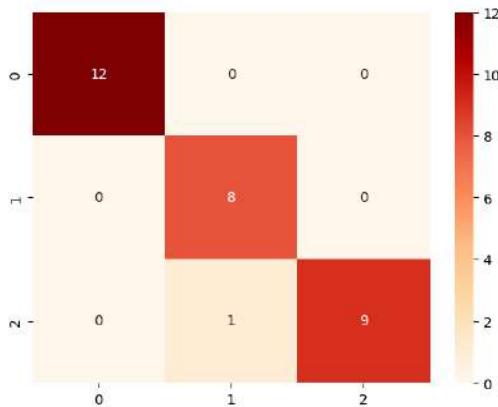
```

### Exercise 3

```
In [8]: M
1 from sklearn.datasets import load_iris
2
3 from sklearn.model_selection import train_test_split
4
5 iris = load_iris()
6 X = iris.data
7 y = iris.target
8
9
10 X_train , X_test , y_train , y_test = train_test_split(X , y , random_state= 12 , test_size=0.2)
11
12
13 model = LogisticRegression(X_train, y_train, iteration=1000, learning_rate=0.03, stopping_threshold=1e-6, degree=2)
14
15 # Train the model
16 model.train()
17
18 # Make predictions
19 predict_test = model.predict(X_test)
20 predict_train= model.predict(X_train)
21
22 #score
23 from sklearn.metrics import accuracy_score
24
25 print(f" train accuracy is: {accuracy_score(predict_train , y_train)} and test accuracy is: {accuracy_score(predict_test , y_test)}")
26
```

train accuracy is: 0.958333333333334 and test accuracy is: 0.9666666666666666

```
In [9]: M
1 from sklearn.metrics import confusion_matrix
2 import seaborn as sb
3 import matplotlib.pyplot as plt
4
5 cnf1=confusion_matrix(predict_test , y_test)
6 sb.heatmap(cnf1 , annot=True , cmap="OrRd")
7 plt.show()
```



## Exercise 4 mnist

```
In [10]: M
1 from sklearn.datasets import load_digits
2
3 mnist=load_digits()
4 X= mnist.data
5 y=mnist.target
6
7 # split
8 from sklearn.model_selection import train_test_split
9
10 X_train , X_test , y_train , y_test = train_test_split(X , y , random_state= 1456 , test_size=0.3)
11
12 #train
13 model2 = LogisticRegression(X_train , y_train,iteration=1000, learning_rate=0.03, stopping_threshold=1e-6, degree=1 )
14 model2.train()
15
16 # Make predictions
17 predict_test = model2.predict(X_test)
18 predict_train= model2.predict(X_train)
19
20 #score
21 from sklearn.metrics import accuracy_score
22
23 print(f" train accuracy is: {accuracy_score(predict_train , y_train)} and test accuracy is: {accuracy_score(predict_test , y_test)}")
24
25
```

train accuracy is: 0.9562450278440732 and test accuracy is: 0.9351851851851852

```
In [11]: 1 probability = np.max((model2.predict_proba(X_test)*100) , axis=1)
2
3 predict= model2.predict(X_test)
4
5 result=np.round(list(zip(predict,probability )))
6
7 import pandas as pd
8
9 df=pd.DataFrame(result , columns=["prediction" , "probability"] )
10 df
```

Out[11]:

	prediction	probability
0	1.0	59.0
1	3.0	98.0
2	8.0	86.0
3	1.0	64.0
4	3.0	84.0
...	...	...
535	9.0	22.0
536	1.0	97.0
537	1.0	81.0
538	9.0	54.0
539	0.0	91.0

540 rows × 2 columns

## Sklearn

```
In [1]: 1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
4 from sklearn.datasets import load_digits
5 from sklearn.preprocessing import MaxAbsScaler
6 from sklearn.model_selection import train_test_split
7 from sklearn.linear_model import LogisticRegression
8
9 # Load dataset
10 mnist = load_digits()
11 X = mnist.data
12 y = mnist.target
13
14 # Split data
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=78)
16
17 # Normalize data
18 scaler = MaxAbsScaler()
19 X_train = scaler.fit_transform(X_train)
20 X_test = scaler.transform(X_test)
21
22 # Create and train model
23 model2 = LogisticRegression(max_iter=1000, solver='lbfgs', multi_class='auto') #multi_class='multinomial'
24 model2.fit(X_train, y_train)
25
26 # Test accuracy
27 y_predtest = model2.predict(X_test)
28 accuracytest = accuracy_score(y_test, y_predtest)
29 print(f'Accuracy test: {accuracytest:.4f}')
30
31 # Train accuracy
32 y_predtrain = model2.predict(X_train)
33 accuracytrain = accuracy_score(y_train, y_predtrain)
34 print(f'Accuracy train: {accuracytrain:.4f}')
35
36 # Display confusion matrix
37 conf_matrix = confusion_matrix(y_test, y_predtest)
38 print('Confusion Matrix:')
39 print(conf_matrix)
40
41
```

Accuracy test: 0.9694  
Accuracy train: 0.9854  
Confusion Matrix:  
[[24 0 0 0 0 0 0 0 0 0]  
 [ 0 31 0 0 0 0 0 1 0]  
 [ 0 0 26 0 0 0 0 0 0 0]  
 [ 0 0 0 38 0 0 0 0 1 0]  
 [ 0 1 0 0 46 0 0 0 0 1]  
 [ 0 0 1 0 0 29 0 0 0 1]  
 [ 0 0 0 0 0 0 42 0 0 0]  
 [ 0 0 0 0 0 0 0 44 0 2]  
 [ 0 2 1 0 0 0 0 0 30 0]  
 [ 0 0 0 0 0 0 0 0 0 39]]

## Exercise 5

```
In [14]: 1 from sklearn.datasets import load_digits
2
3 mnist= load_digits()
4 X_ = mnist.data
5 y=mnist.target
6
7 #normalize
8 from sklearn.preprocessing import StandardScaler
9 sc=StandardScaler()
10 X=sc.fit_transform(X_)
11
12 #split
13 from sklearn.model_selection import train_test_split
14 X_train , X_test , y_train , y_test = train_test_split(X_ , y , random_state=2 , test_size=0.2)
15
16 #import logistic regression
17 from sklearn.linear_model import LogisticRegression
18 model3= LogisticRegression(max_iter=10000, solver='lbfgs')
19 model3.fit(X_train , y_train)
20
21
```

```
Out[14]: LogisticRegression
LogisticRegression(max_iter=10000)
```

```
In [15]: 1 #predict
2 test_prediction=model3.predict(X_test)
3 train_prediction = model3.predict(X_train)
4
5 #accuracy
6 from sklearn.metrics import accuracy_score
7
8 acctrain=accuracy_score(train_prediction , y_train)
9 print("acc train is:" , acctrain)
10 print("*****")
11 acctest=accuracy_score(test_prediction , y_test)
12 print("acc test is:" , acctest)
13
```

```
acc train is: 0.9986082115518441
*****
acc test is: 0.95
```

```
In [16]: 1 #confusion matrix
2 from sklearn.metrics import confusion_matrix
3
4 cnfm3= confusion_matrix(test_prediction , y_test)
5 display(cnfm3)
```

```
array([[32, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 41, 0, 0, 0, 1, 1, 0, 0, 0],
       [0, 0, 31, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 35, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 30, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 41, 0, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 33, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 39, 0, 1],
       [0, 1, 0, 0, 3, 0, 1, 1, 36, 1],
       [0, 1, 0, 0, 1, 0, 0, 0, 24]], dtype=int64)
```

## Exercise 6

```
In [17]: 1 import pandas as pd
2 from sklearn.datasets import load_wine
3
4 wine=load_wine()
5
6 df=pd.DataFrame(wine.data , columns=wine.feature_names)
7 df[["target"]]= wine.target
8 df
9
10
11
```

```
Out[17]:
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0	2
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750.0	2
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835.0	2
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840.0	2
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560.0	2

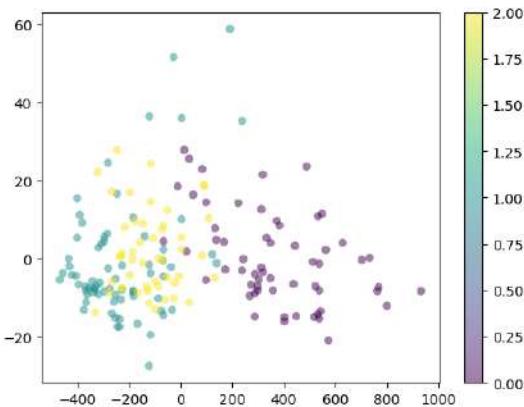
178 rows × 14 columns

```
In [18]: 1 #Create x and y
2 X=df.iloc[:, :-1]
3 y=df.iloc[:, -1]
```

```
In [19]: 1 #decrease dataset to 2dimension
2 from sklearn.decomposition import PCA
3 pca= PCA(n_components=2)
4 X_proj = pca.fit_transform(X)
5
6 print("Shape of original data: {}".format(X.shape))
7 print("Shape of projected data: {}".format(X_proj.shape))

Shape of original data: (178, 13)
Shape of projected data: (178, 2)
```

```
In [20]: 1 import matplotlib.pyplot as plt
2 plt.scatter(X_proj[:, 0], X_proj[:, 1],
3             c=y, edgecolor='none', alpha=0.5,
4             cmap=plt.colormaps.get_cmap('viridis'))
5 plt.colorbar()
6 plt.show()
7
8
```



```
In [21]: 1 #split
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X_proj, y, test_size= 0.2 , random_state=54698)
4
5
6 #X_proj
7 from sklearn.linear_model import LogisticRegression
8 clf= LogisticRegression(n_jobs=-1)
10 clf.fit(X_train , y_train)
11
12
```

```
Out[21]: LogisticRegression
LogisticRegression(n_jobs=-1)
```

```
In [22]: 1 train_predict= clf.predict(X_train)
2 test_predict=clf.predict(X_test)
3
4 #accuracy
5 from sklearn.metrics import accuracy_score
6 print("the train accuracy is: {} and the test accuracy is {}".format((accuracy_score(train_predict , y_train)) , (accuracy_score(test_predict , y_test)) ))
```

the train accuracy is: 0.7112676056338029 and the test accuracy is 0.6111111111111112

## Exercise 7

```
In [23]: 1 from sklearn.datasets import load_iris
2
3 iris= load_iris()
4
5 X_= iris.data
6 y=iris.target
7
8 from sklearn.preprocessing import StandardScaler
9 from sklearn.model_selection import train_test_split
10
11 sc=StandardScaler()
12 X=sc.fit_transform(X_)
13
14 X_train , X_test , y_train, y_test = train_test_split(X , y , random_state= 12 , test_size=0.2)
```

```
In [24]: 1 from sklearn.linear_model import LogisticRegression
2
3 clf1= LogisticRegression()
4 clf1.fit(X_train , y_train)
5
```

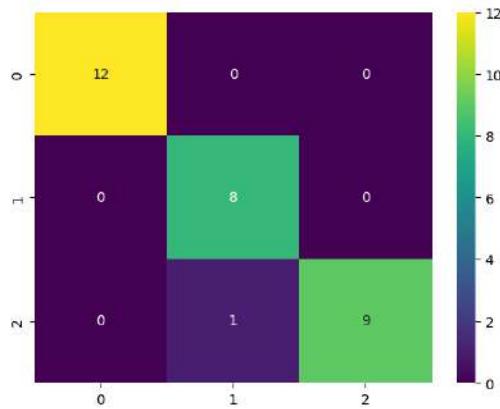
```
Out[24]: LogisticRegression
LogisticRegression()
```

```
In [25]: 1 predict_test= clf1.predict(X_test)
2 predict_train=clf1.predict(X_train)
3
```

```
In [26]: 1 #score
2 from sklearn.metrics import accuracy_score
3 acc_test=accuracy_score(predict_test , y_test)
4 acc_train=accuracy_score(predict_train , y_train)
5
6 print(f" the train accuracy is:{acc_train} and the test accuracy is: {acc_test}")
7
8
```

the train accuracy is:0.975 and the test accuracy is: 0.9666666666666667

```
In [27]: 1 from sklearn.metrics import confusion_matrix
2 import seaborn as sb
3 import matplotlib.pyplot as plt
4
5 iriscn= confusion_matrix(predict_test , y_test)
6 sb.heatmap(iriscn , annot=True , cmap='viridis')
7 plt.show()
8
```



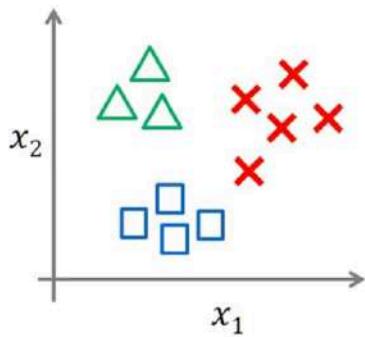
# Machine learning

Soft-Max classifier  
Morteza khorsand



## Soft-Max Classifier

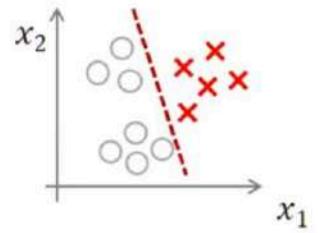
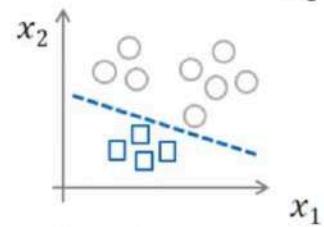
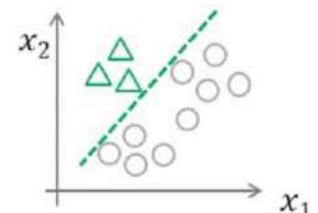
### ■ Logistic regression



$$h^{(1)}(x) = g\left((\theta^{(1)})^T x\right)$$

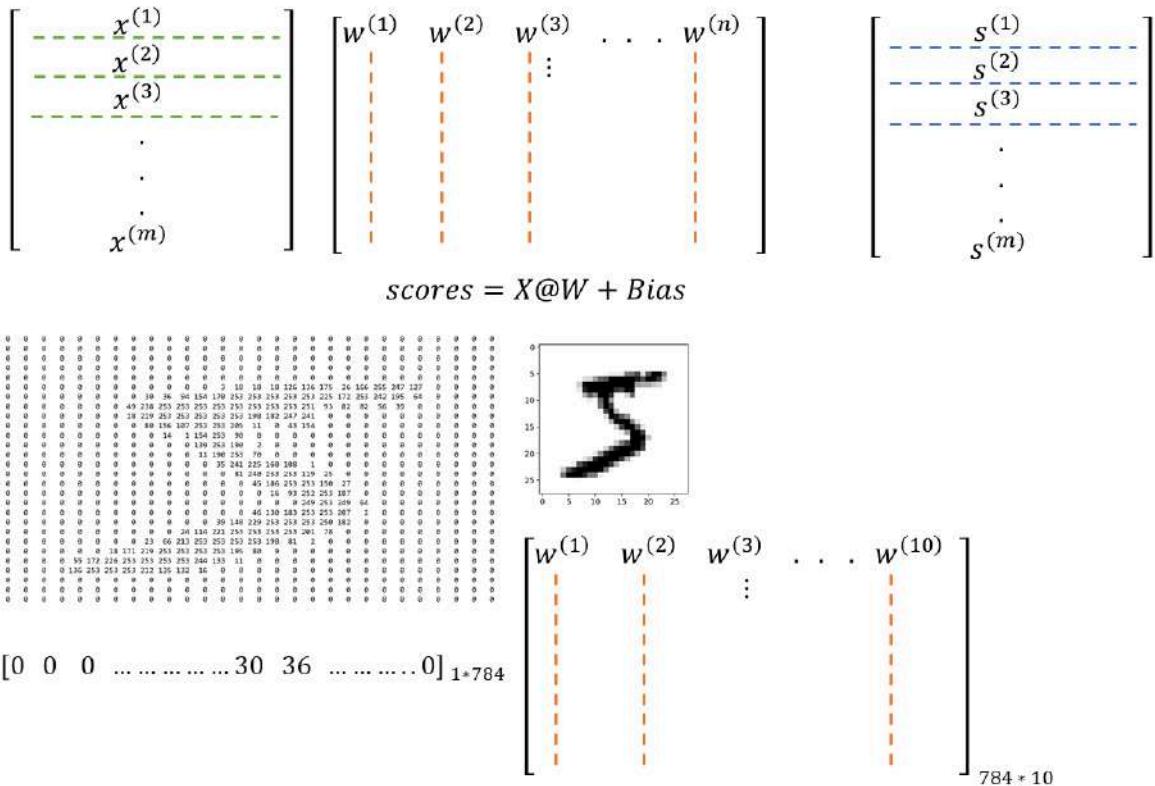
$$h^{(2)}(x) = g\left((\theta^{(2)})^T x\right)$$

$$h^{(3)}(x) = g\left((\theta^{(3)})^T x\right)$$



$$y = \operatorname{argmax} h_{\theta}^{(i)} (x)$$

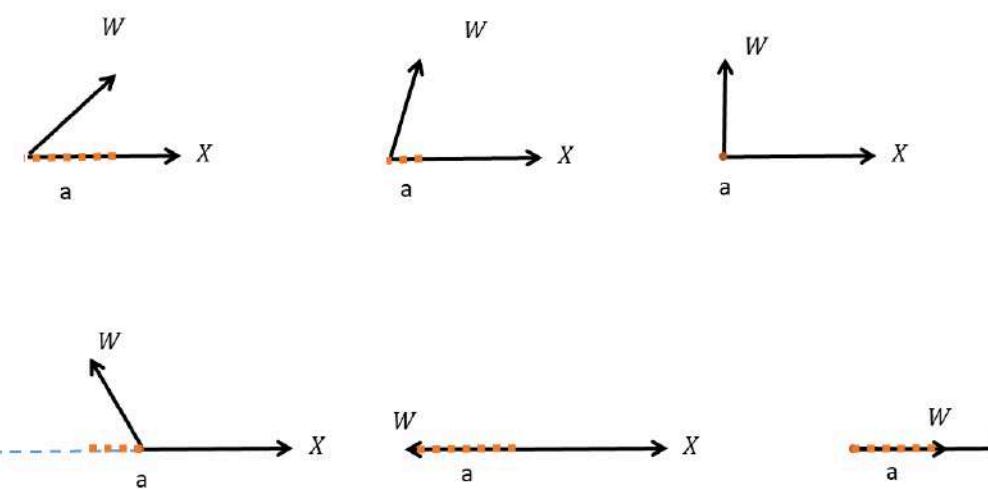
■ Batch processing



■ Scalar product - Inner product

$$x \cdot w = \|x\| \|w\| \cos \theta$$

$$X \cdot W = a \times |x|$$



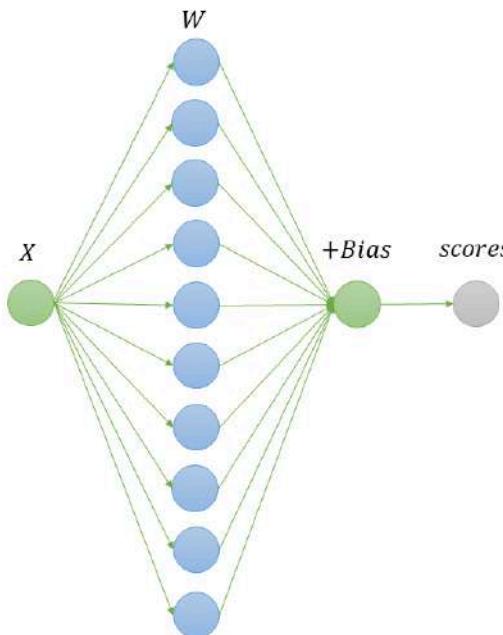
```
In [ ]: 
1 import numpy as np
2
3 X= np.array([[10 , 0 ]])
4
5 w1=np.array([[0],
6             [10]])
7
8 w2=np.array([[5],
9             [5 ]])
10
11 w3=np.array([[-10],
12              [0]])
13
14 w4=np.array([[1],
15              [6]])
16
17 w5=np.array([[10],
18              [0]])
```

```
In [ ]: 
1 print(np.dot(X ,w1))
2 print(np.dot(X ,w2))
3 print(np.dot(X ,w3))
4 print(np.dot(X ,w4))
5 print(np.dot(X ,w5))
```

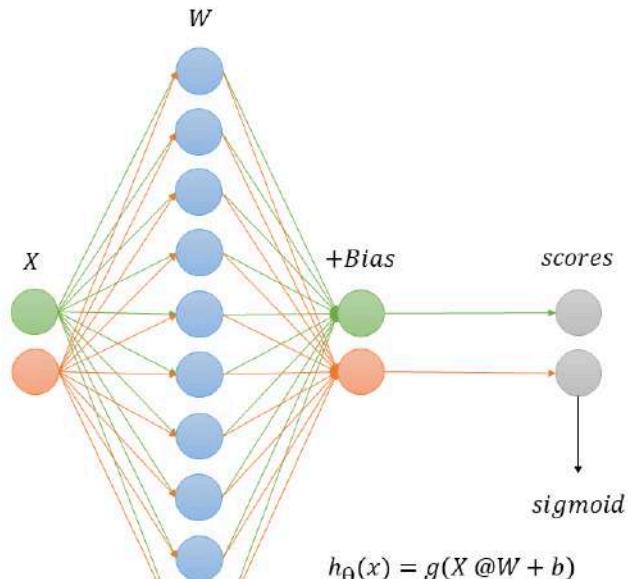
## Soft-Max Classifier

4

*one sample*

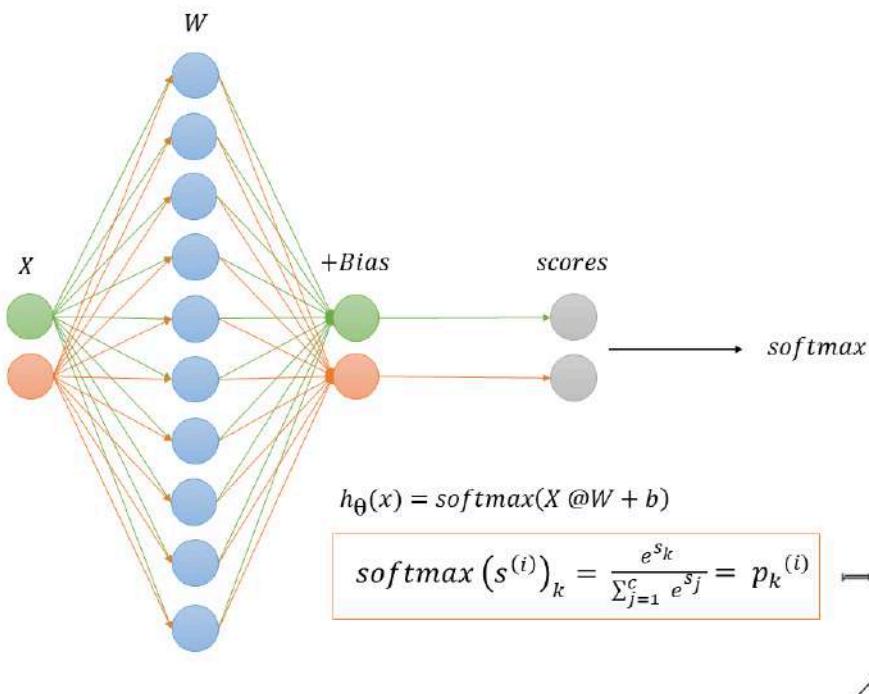


*two samples*



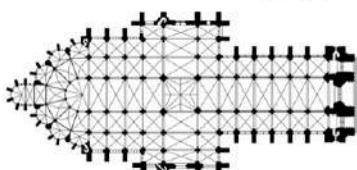
### ■ Softmax Function

normalized exponential function



### ■ Softmax loss function – cross-entropy loss function

20\*10



Amiens Cathedral floorplan. Gothic cathedrals built in France during the 13th century

```
def softmax_loss(scores, y):
    #softmax loss implementation
    #y in ended as a one-hot vector
    p = softmax(scores)
    return -np.sum(y*np.log(p))
```

$$l_i = -\log p(Y = y^{(i)} | X = x^{(i)}) = -\log \left( \frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^c e^{s_j}} \right)$$

$$l_i = \sum_{k=1}^c -y_k \log p_k \quad Y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_c \end{bmatrix} \longrightarrow H(p) = \sum_{i=1}^c -p_i \log p_i \quad \text{Entropy } H$$

style	scores	$e^{s_k}$	$\frac{e^{s_k}}{\sum_{j=1}^c e^{s_j}}$	one-hot encode	- Log (softmax)
Gothic	1.16	$e^{1.16} = 3.18$	0.05	1	3
Baroque	3.91	$e^{3.91} = 49.89$	0.90	0	0
Modern	-3.44	$e^{-3.44} = 0.03$	0.0005	0	0

■ Total Cost function

$$(w, b) = \frac{1}{m} \sum_{i=1}^m l_i + \lambda R(w)$$

$$(w, b) = \frac{1}{m} \sum_{i=1}^m (-\log p_y^{(i)}) + \lambda \|w\|_2^2$$

$$(w, b) = \frac{1}{m} \sum_{i=1}^m \left( -\log \left( \frac{e^{s_{y(i)}}}{\sum_{j=1}^c e^{s_j}} \right) \right) + \lambda \|w\|_2^2$$

$$(w, b) = \frac{1}{m} \sum_{i=1}^m \left( -\log \left( \frac{e^{\left( (w^{(i)})^T x^{(i)} + b^{(i)} \right)}}{\sum_{j=1}^c e^{\left( (w^{(j)})^T x^{(j)} + b^{(j)} \right)}} \right) \right) + \lambda \|w\|_2^2$$

■ Cost function derivative

$$l_i = -\log p_y^{(i)} \quad P_k = \left( \frac{e^{s_{y(i)}}}{\sum_{j=1}^c e^{s_j}} \right) \quad S_k = (w^{(k)})^T x^{(i)} + b^{(k)}$$

$$\frac{\partial l_i}{\partial w^k} = \frac{\partial l_i}{\partial p_y^{(i)}} \times \frac{\partial p_y^{(i)}}{\partial s_k} \times \frac{\partial s_k}{\partial p_w^k}$$

$$K=y^{(i)} \longrightarrow \frac{\partial l_i}{\partial w^k} = \left( -\frac{1}{p_y^{(i)}} \right) \cdot p_k \cdot (1-p_k) \cdot x^{(i)} = (p_k - 1) \cdot x^{(i)}$$

$$K \neq y^{(i)} \longrightarrow \frac{\partial l_i}{\partial w^k} = \left( -\frac{1}{p_y^{(i)}} \right) \cdot p_y^{(i)} \cdot (-p_k) \cdot x^{(i)} = (p_k) \cdot x^{(i)}$$

$$K=y^{(i)} \quad W\text{- derivative} = \frac{1}{m} (X.T @ (\text{softmax(scores)} - \text{yencode}))$$

$$K \neq y^{(i)} \quad \text{bias - derivative} = \frac{1}{m} (\text{sum}(\text{softmax(scores)} - \text{yencode}))$$

```
In [1]: 1 import numpy as np
2 y = np.array([0, 1, 2, 2])
3
4 X = np.array([[0.1, 0.5],
5               [1.1, 2.3],
6               [-1.1, -2.3],
7               [-1.5, -2.5]])
8
9
10 c= np.unique(y).shape[0]      #NUMBER OF CLASSES
11 m,n=X.shape                 #number of features
12
13
14
```

```
In [2]: 1 #create functions
2
3
4 #scores
5 def scores(w , b , x):
6     scores = x@w + b
7     return scores
8
9
10 #softmax
11 def softmax(scores):
12     return np.exp(scores) / np.sum(np.exp(scores), axis = 1, keepdims = True)
13
14
15 #one hot
16 def one_hot_encode(y):
17     n_class = np.unique(y).shape[0]
18     y_encode = np.zeros((y.shape[0], n_class))
19     for idx, val in enumerate(y):
20         y_encode[idx, val] = 1.0
21
22     return y_encode
23
24
25
26 #cross entropy
27 def cross_entropy_cost(y_hot, smax):
28     return np.mean(-np.sum(y_hot * np.log(smax), axis = 1))
29
30
31 #Loss = -np.sum(np.Log(probs[range(m), y])) / m      #instead of one hot-vector
```

```
In [3]: 1 def one_hot_encode(y):
2     n_class = np.unique(y).shape[0]
3     y_encode = np.zeros((y.shape[0], n_class))
4     for idx, val in enumerate(y):
5         y_encode[idx, val] = 1.0
6
7     return y_encode
8
9 y_encode=one_hot_encode(y)
10 y_encode
```

Out[3]: array([[1., 0., 0.],
 [0., 1., 0.],
 [0., 0., 1.],
 [0., 0., 1.]])

```
In [4]: 1 def train(x, y, iterations = 100, learning_rate = 0.0001 , stopping_threshold = 1e-6):
2
3     n=X.shape[1]
4     m=X.shape[0]
5
6     ##Initializing weight, bias, learning rate and iterations
7     current_weight = 0.01 * np.random.randn(n,c)
8     current_bias = np.zeros(c)
9
10    iterations = iterations
11    learning_rate = learning_rate
12
13
14    costs = []
15    weights = []
16    previous_cost = None
17
18    for i in range(iterations):
19
20        # Making predictions
21        scores = (X.dot(current_weight)) + current_bias
22
23
24        smax= softmax(scores)
25
26
27        y_hot_vector= one_hot_encode(y)      # initial y [0,1,2,2]
28
29        # Calculationg the current cost
30        current_cost= cross_entropy_cost(y_hot_vector, smax)
31
32
33
34        # If the change in cost is Less than or equal to
35        # stopping_threshold we stop the gradient descent
36        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
37            break
38
39
40        previous_cost = current_cost
41
42        costs.append(current_cost)
43        weights.append(current_weight)
44
45        # Calculating the gradients
46        #dscores = np.copy(scores)
47        #dscores-=1.0
48
49        weight_derivative = (1/m)*np.dot(X.T, (smax-y_hot_vector))
50        bias_derivative= (1/m)*np.sum(smax - y_hot_vector)
51
52
53
54
55        # Updating weights and bias
56        current_weight = current_weight - (learning_rate * weight_derivative)
57        current_bias = current_bias - (learning_rate * bias_derivative)
58
59        # Printing the parameters for each 1000th iteration
60        #print(f"Iteration {i+1}: Cost {current_cost}, Weight \
61        #{current_weight}, Bias {current_bias}")
62
63    return current_weight, current_bias
```

```
In [5]: 1 train(X , y)
```

```
Out[5]: (array([[ 0.00503519,  0.00798346,  0.00140098],
   [-0.00663693,  0.00778321, -0.00442581]]),
 array([-4.16333634e-20, -4.16333634e-20, -4.16333634e-20]))
```

In [6]:

```
1 import numpy as np
2
3 class Softmax:
4     def __init__(self, learning_rate=0.0001, iterations=100, stopping_threshold=1e-6):
5         self.learning_rate = learning_rate
6         self.iterations = iterations
7         self.stopping_threshold = stopping_threshold
8         self.costs = []
9         self.weights = []
10        self.current_weight = None
11        self.current_bias = None
12
13    def fit(self, X, y):
14        n_samples = X.shape[0]
15        n_features = X.shape[1]
16
17        # Initialize weights and bias
18        self.current_weight = np.zeros(n_features)
19        self.current_bias = 0
20
21        previous_cost = None
22
23        for i in range(self.iterations):
24            # Make predictions
25            scores = (X.dot(self.current_weight)) + self.current_bias
26            smax = self.softmax(scores)
27
28            # One hot encode y
29            y_hot_vector = self.one_hot_encode(y)
30
31            # Calculate the current cost
32            current_cost = self.cross_entropy_cost(y_encode, smax)
33
34            # If the change in cost is less than or equal to stopping_threshold we stop the gradient descent
35            if previous_cost and abs(previous_cost - current_cost) <= self.stopping_threshold:
36                break
37
38            previous_cost = current_cost
39
40            self.costs.append(current_cost)
41            self.weights.append(self.current_weight.copy())
42
43            # Derivatives
44            weight_derivative = (1 / n_samples) * np.dot(X.T, smax - y_hot_vector)
45            bias_derivative = (1 / n_samples) * np.sum(smax - y_hot_vector)
46
47            # Update weights and bias
48            self.current_weight = self.current_weight - (self.learning_rate * weight_derivative)
49            self.current_bias = self.current_bias - (self.learning_rate * bias_derivative)
50
51        return self.current_weight, self.current_bias
52
53    def cross_entropy_cost(self, y_encode, smax):
54        return np.mean(-np.sum(y_encode * np.log(smax), axis=1))
55
56    def softmax(self, z):
57        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True)) # numerical stability
58        return exp_z / np.sum(exp_z, axis=1, keepdims=True)
59
60    def one_hot_encode(self, y):
61        n_classes = np.unique(y).shape[0]
62        y_encode = np.zeros((y.shape[0], n_classes))
63        for idx, val in enumerate(y):
64            y_encode[idx, val] = 1.0
65
66        return y_encode
```

In [7]:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class SoftmaxRegression:
5     def __init__(self, X, y, iteration, learning_rate, stopping_threshold, degree):
6         self.X = X
7         self.y = y
8         self.iteration = iteration
9         self.learning_rate = learning_rate
10        self.classes = np.unique(y)
11        self.classifiers = {}
12        self.stopping_threshold = stopping_threshold
13        self.degree = degree
14        self.current_weight = None
15        self.current_bias = None
16
17    def normalize(self, X):
18        mean = X.mean(axis=0)
19        std = X.std(axis=0)
20        std[std == 0] = 1           # To avoid division by zero, set zero std to one
21        return (X - mean) / std
22
23    def transform(self, X):
24        X_norm = self.normalize(X)
25        num_samples, num_features = X_norm.shape
26        X_transform = np.empty((num_samples, 0))
27        for i in range(1, self.degree + 1):
28            X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
29        return X_transform
30
31    def softmax(self, Z):
32        exp_Z = np.exp(Z - np.max(Z, axis=1, keepdims=True))      # Stability improvement
33        return exp_Z / np.sum(exp_Z, axis=1, keepdims=True)
34
35 #def softmax(self, z):
36 #    return np.exp(z) / np.sum(np.exp(z), axis = 1, keepdims = True)
37
38    def one_hot_encode(self, y):
39        n_classes = len(self.classes)
40        y_encoded = np.zeros((y.shape[0], n_classes))
41        for idx, val in enumerate(y):
42            y_encoded[idx, val] = 1.0
43        return y_encoded
44
45    def cross_entropy_cost(self, y_encoded, y_hat):
46        return -np.mean(np.sum(y_encoded * np.log(y_hat), axis=1))
47
48
49    def fit (self):
50        Xtansnorm = self.transform(self.X)
51        m, n = Xtansnorm.shape
52        k = len(self.classes)
53
54        # Initialize weights and biases
55        self.current_weight = np.random.rand(n, k)
56        self.current_bias = np.zeros((1, k))
57
58        y_encoded = self.one_hot_encode(self.y)
59
60        costs = []
61        previous_cost = None
62
63        for i in range(self.iteration):
64            # Forward propagation
65            Z = Xtansnorm @ self.current_weight + self.current_bias
66            y_hat = self.softmax(Z)
67
68            # Calculate cost
69            loss = self.cross_entropy_cost(y_encoded, y_hat)
70
71            # If the change in cost is less than or equal to stopping_threshold, stop the gradient descent
72            if previous_cost is not None and abs(previous_cost - loss) <= self.stopping_threshold:
73                break
74
75            previous_cost = loss
76            costs.append(loss)
77
78            # Backward propagation (compute gradients)
79            weight_derivative = Xtansnorm.T @ (y_hat - y_encoded) / m
80            bias_derivative = np.sum(y_hat - y_encoded, axis=0, keepdims=True) / m
81
82            # Update weights and biases
83            self.current_weight -= self.learning_rate * weight_derivative
84            self.current_bias -= self.learning_rate * bias_derivative
85
86        return self.current_weight, self.current_bias, costs
87
88    def predict_proba(self, X):
89        Xtansnorm = self.transform(X)
90        Z = Xtansnorm @ self.current_weight + self.current_bias
91        probabilities = self.softmax(Z)
92        return probabilities
93
94    def predict(self, X):
95        probabilities = self.predict_proba(X)
96        predictions = np.argmax(probabilities, axis=1)
97        return self.classes[predictions]
98

```

```
In [8]: 
1 X=np.array([[100,100,100,60,3,95,100,100,
2           100,100,67,14,1,100,100,100,
3           100,57,35,78,8,100,100,100,
4           100,90,91,89,13,100,100,100,
5           100,100,100,88,13,100,100,100,
6           100,100,100,85,16,100,100,100,
7           100,100,100,85,16,100,100,100,
8           100,100,100,95,9,100,100,100],
9           [100,100,78,26,67,100,100,100,
10          100,84,19,30,69,100,100,100,
11          100,31,71,46,65,100,100,100,
12          100,100,100,51,58,100,100,100,
13          100,100,100,58,51,100,100,100,
14          100,100,100,64,46,100,100,100,
15          100,100,100,73,39,100,100,100,
16          100,100,100,85,44,100,100,100],
17          [100,98,51,34,42,82,100,100,
18          100,72,24,84,51,53,100,100,
19          100,64,49,100,61,51,100,100,
20          100,100,100,28,75,99,100,
21          100,100,100,58,51,100,100,100,
22          100,100,100,64,46,100,100,100,
23          100,100,100,73,39,100,100,100,
24          100,100,100,85,44,100,100,100],
25          [100,98,51,34,42,82,100,100,
26          100,72,24,84,51,53,100,100,
27          100,64,49,100,61,51,100,100,
28          100,100,100,28,75,99,100,
29          100,100,100,58,51,100,100,100,
30          100,100,100,64,46,100,100,100,
31          100,100,100,73,39,12,83,100,100,
32          100,49,8,19,56,93,100,100,
33          [100,75,13,15,58,100,100,100,
34          100,51,75,84,33,68,100,100,
35          100,100,100,57,52,100,100,
36          100,89,71,39,12,83,100,100,
37          100,49,8,19,56,93,100,100,
38          100,100,100,79,44,31,100,100,
39          100,100,100,96,23,100,100,
40          100,45,96,87,37,47,100,100],
41          [100,100,100,38,33,45,100,100,
42          100,99,46,39,67,45,100,100,
43          100,50,30,93,64,49,100,100,
44          53,16,62,63,36,33,43,80,
45          58,41,49,49,18,26,65,92,
46          100,100,100,100,58,55,100,100,
47          100,100,100,100,58,55,100,100,
48          100,100,100,100,61,51,100,100]]))
49
50 y=np.array([0,0,1,2,3])
51
```

```
In [9]: 
1 import numpy as np
2 from PIL import Image
3
4 # Your numpy array X
5 X1 = np.array([[100,75,13,15,58,100,100,100,
6           100,51,75,84,33,68,100,100,
7           100,100,100,57,52,100,100,
8           100,89,71,39,12,83,100,100,
9           100,49,8,19,56,93,100,100,
10          100,100,100,79,44,31,100,100,
11          100,100,100,96,23,100,100,
12          100,45,96,87,37,47,100,100]])
13
14 # Reshape X to create an 8x8 image
15 X_reshaped = X1.reshape((8, 8))
16
17 # Convert the numpy array to an image (grayscale)
18 image = Image.fromarray(X_reshaped)
19
20 # Show the image
21 image.show()
22
```

```
In [10]: 
1 clf1=SoftmaxRegression(X, y , iteration=100, learning_rate=10e-3, stopping_threshold = 10e-6, degree=1)
2 a=clf1.fit()
3 clf1.predict(X)
4
```

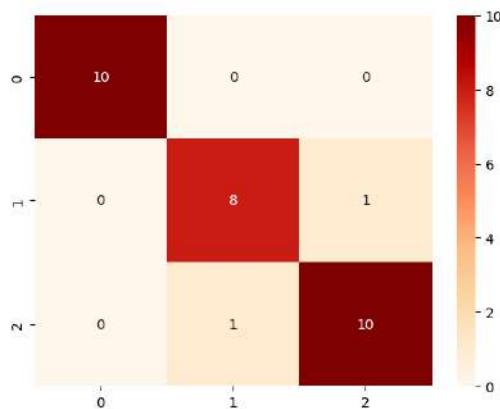
Out[10]: array([0, 0, 1, 2, 3])

```
In [11]: 
1 import numpy as np
2 from sklearn.datasets import load_iris
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score
5
6
7
8 # Load dataset (example using the iris dataset)
9 data = load_iris()
10 X = data.data
11 y = data.target
12
13 # Split the dataset into training and test sets
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
15
16
```

```
In [12]: 
1 clf1=SoftmaxRegression(X_train, y_train , iteration=1000, learning_rate=10e-3, stopping_threshold = 10e-6, degree=1)
2 a=clf1.fit()
3 y_pred= clf1.predict(X_test)
4
5 accuracy_score(y_pred , y_test)
6
```

Out[12]: 0.9333333333333333

```
In [13]: 1 from sklearn.metrics import confusion_matrix
2 import seaborn as sb
3 import matplotlib.pyplot as plt
4
5 cnf1=confusion_matrix(y_pred , y_test)
6 sb.heatmap(cnf1 , annot=True , cmap="OrRd")
7 plt.show()
```



```
In [14]: 1 from sklearn.datasets import load_digits
2
3 mnist=load_digits()
4 X= mnist.data
5 y=mnist.target
6
7 # split
8 from sklearn.model_selection import train_test_split
9
10 X_train , X_test , y_train , y_test = train_test_split(X , y , random_state= 1456 , test_size=0.3)
11
12 #train
13 clf2 = SoftmaxRegression(X_train , y_train ,iteration=1000, learning_rate=0.03, stopping_threshold=1e-6, degree=1 )
14 clf2.fit()
15
16 # Make predictions
17 predict_test = clf2.predict(X_test)
18 predict_train= clf2.predict(X_train)
19
20 #score
21 from sklearn.metrics import accuracy_score
22
23 print(f" train accuracy is: {accuracy_score(predict_train , y_train)} and test accuracy is: {accuracy_score(predict_test , y_test)}")
```

```
train accuracy is: 0.9594272076372315 and test accuracy is: 0.95
```

## Sklearn

In [15]:

```
1 import numpy as np
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.datasets import load_iris
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.metrics import accuracy_score
7
8 # Load dataset (example using the iris dataset)
9 data = load_iris()
10 X = data.data
11 y = data.target
12
13 # Split the dataset into training and test sets
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
15
16 # Standardize the features (important for gradient-based algorithms)
17 scaler = StandardScaler()
18 X_train = scaler.fit_transform(X_train)
19 X_test = scaler.transform(X_test)
20
21 # Initialize and fit the LogisticRegression model with softmax
22 model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
23 #solver='lbfgs' is often used for multiclass classification problems.
24 model.fit(X_train, y_train)
25
26 # Make predictions
27 y_pred = model.predict(X_test)
28
29 # Calculate accuracy
30 accuracy = accuracy_score(y_test, y_pred)
31 print(f'Accuracy: {accuracy:.2f}')
32
33 # Predict probabilities (softmax output)
34 y_prob = model.predict_proba(X_test)
35 print('Predicted probabilities:')
36 print(y_prob)
37
```

Accuracy: 1.00  
Predicted probabilities:  
[[1.14475073e-02 8.76014266e-01 1.12538227e-01]  
[9.64369004e-01 3.56305853e-02 4.10786879e-07]  
[3.76768338e-08 2.88064673e-03 9.97119316e-01]  
[1.32028749e-02 7.595970668e-01 2.27200057e-01]  
[1.88736913e-03 7.52191840e-01 2.45928791e-01]  
[9.32163787e-01 6.78346862e-02 1.52679556e-06]  
[8.92013671e-02 8.78517730e-01 3.22809028e-02]  
[8.42008232e-05 6.42658457e-02 9.35649953e-01]  
[7.39930569e-04 5.77409899e-01 4.21856170e-01]  
[3.02188127e-02 9.25802172e-01 4.39790155e-02]  
[1.18150194e-03 2.105002066e-01 7.88318292e-01]  
[9.49649175e-01 5.03503361e-02 4.88813101e-07]  
[9.60117238e-01 3.98825066e-02 2.55213774e-07]  
[9.51984539e-01 4.89949984e-02 4.70254636e-07]  
[9.89916261e-01 1.00836487e-02 9.07712609e-08]  
[1.91396506e-02 7.24696375e-01 2.56163974e-01]  
[4.03061386e-05 3.18044910e-02 9.68155203e-01]  
[2.66553386e-02 3.94977581e-01 3.83676808e-02]  
[2.43092297e-02 8.48118670e-01 1.27572100e-01]  
[3.09840180e-05 3.24467710e-02 9.67522245e-01]  
[9.696346503e-01 3.03649341e-02 4.62997876e-07]  
[3.64812070e-03 3.80157455e-01 6.16194424e-01]  
[9.63826294e-01 3.61727421e-02 9.63634789e-07]  
[5.26934090e-05 4.84670344e-02 9.51480272e-01]  
[8.40723306e-05 9.00892760e-02 9.09826652e-01]  
[6.24209964e-05 4.91054840e-02 9.50832095e-01]  
[2.52241948e-05 8.48626965e-02 9.15112079e-01]  
[2.67317844e-05 2.41572334e-02 9.75816035e-01]  
[9.35691444e-01 6.43071156e-02 1.44036235e-06]  
[9.50631939e-01 4.93672098e-02 8.51266279e-07]]

# Machine learning

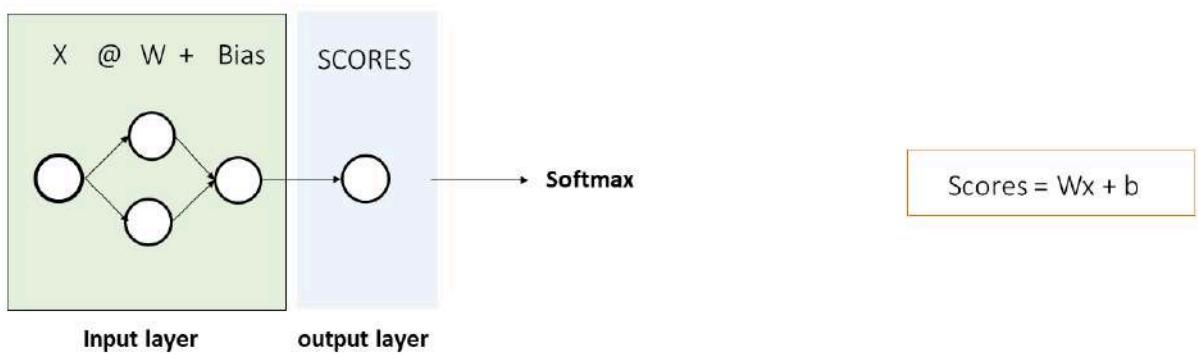
## Artificial neural networks (ANN)

Morteza khorsand

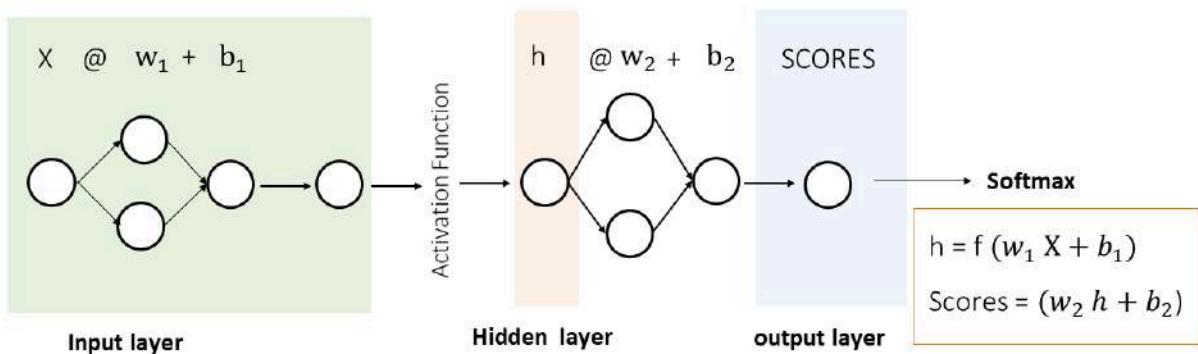


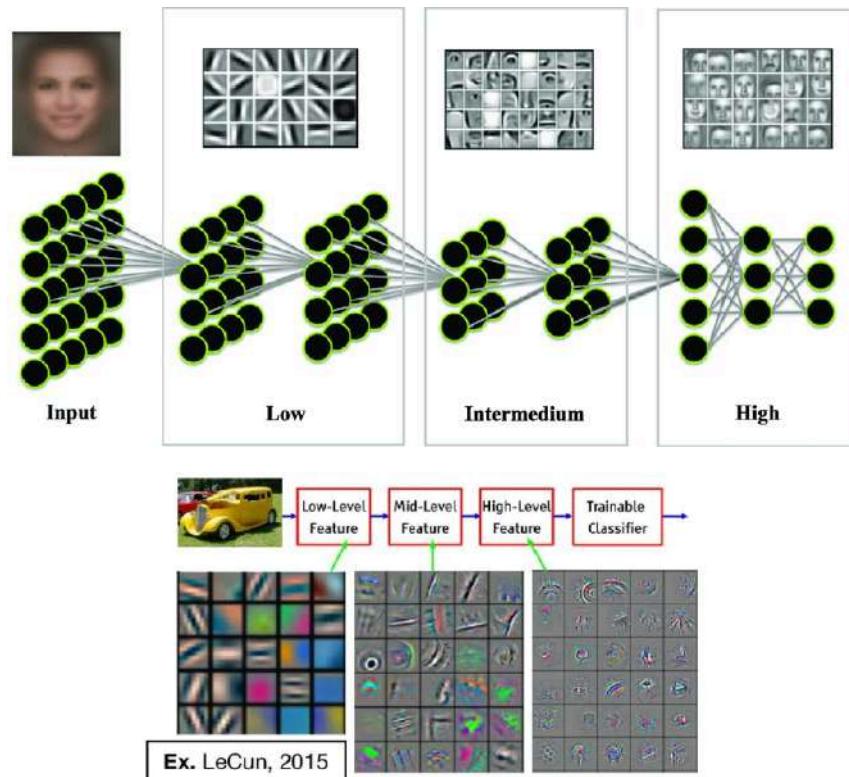
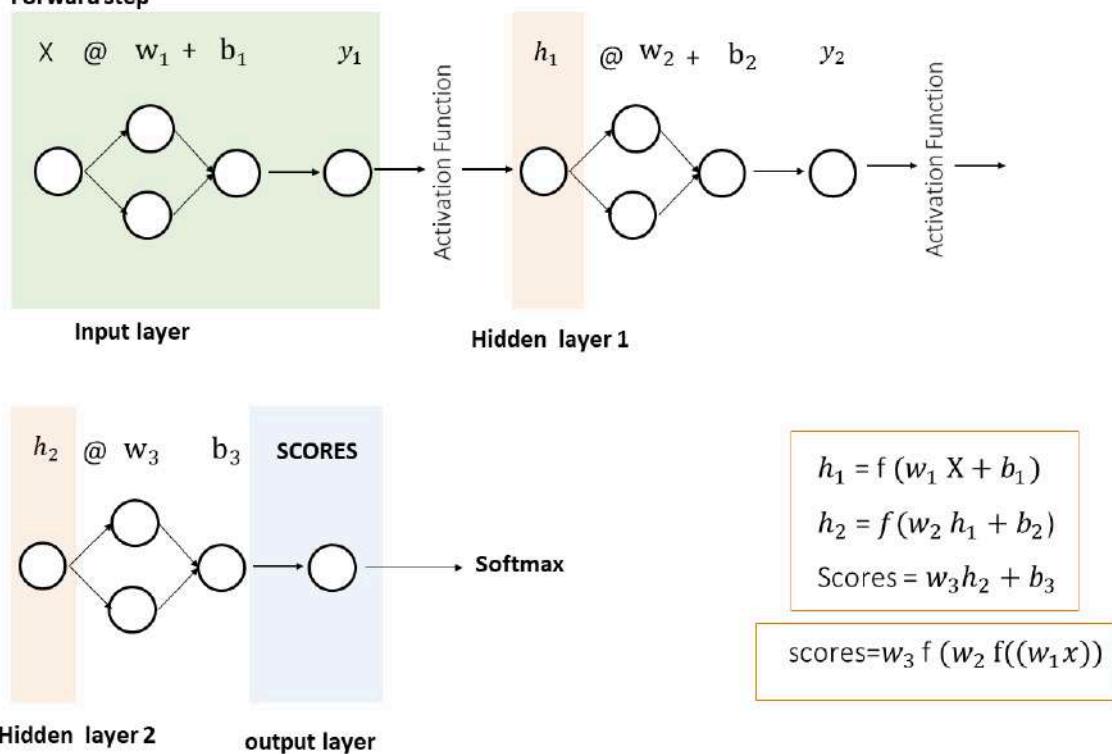
### Artificial neural networks (ANN)

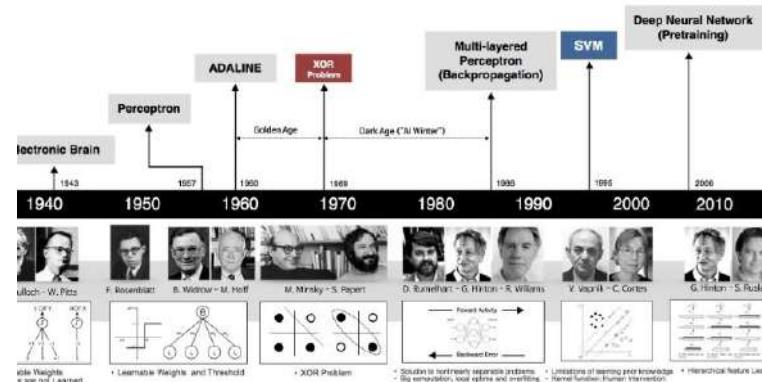
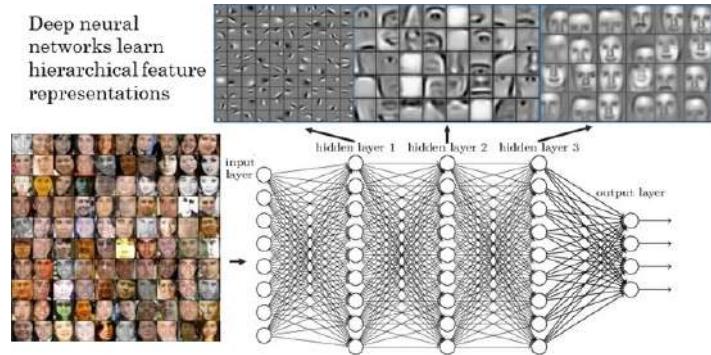
2



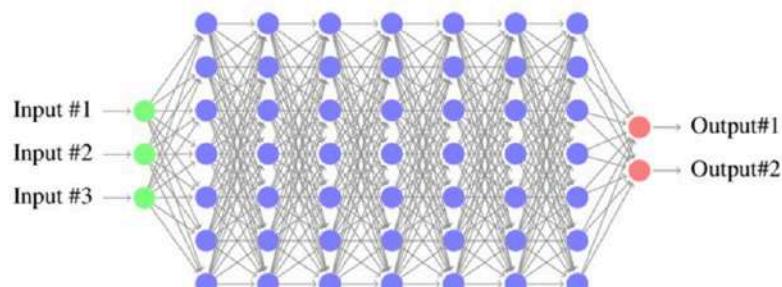
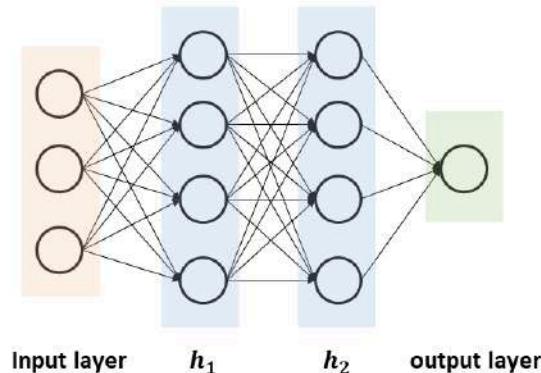
### Multi-layer perceptron(MLP) - Two layers neural network



**Three layers neural network – fully connected network****Forward step**

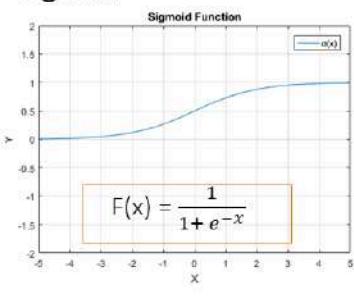


## Artificial neural networks (ANN)

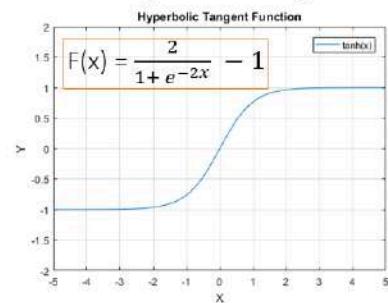


## ■ Activation functions

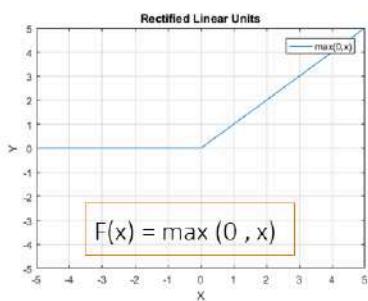
### Sigmoid



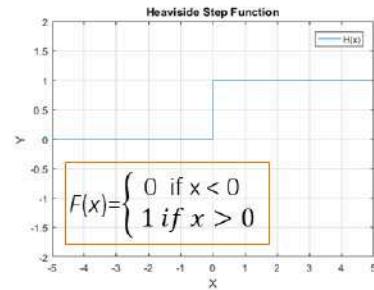
### tanh Hyperbolic Tangent



### Relu (Rectified Linear Unit)



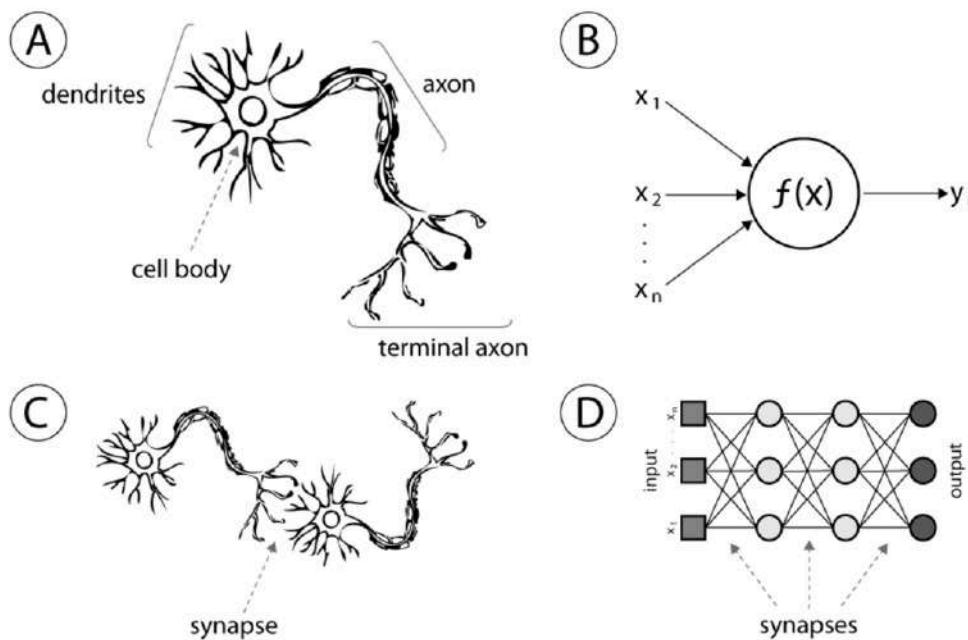
### Threshold Function



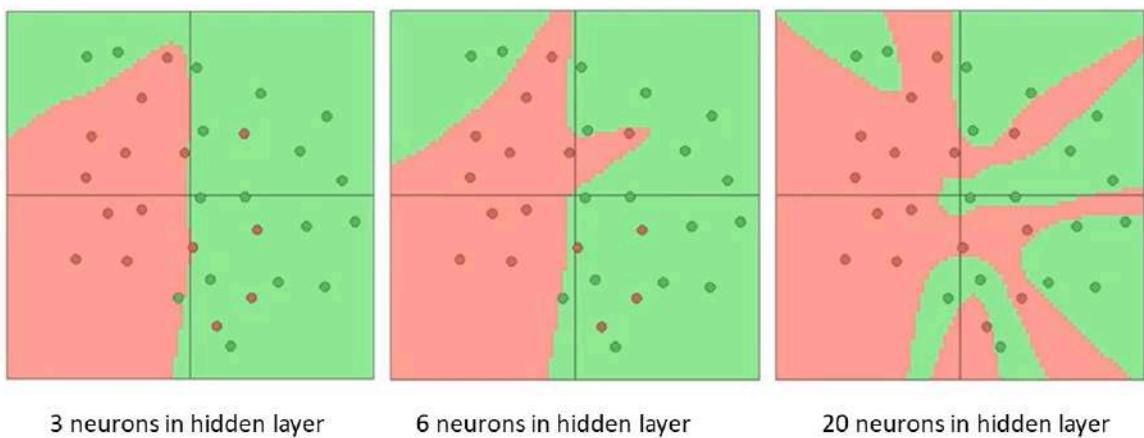
## ■ Activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

### ■ Neural network



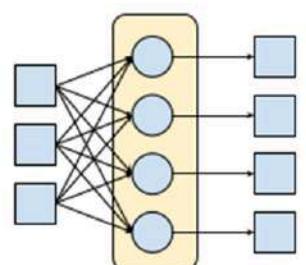
### ■ Neural network



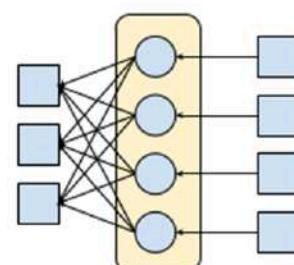
### Backpropagation

$$l_i = \frac{1}{m} \sum_{i=1}^m (-\log (\frac{e^{s_{y(i)}}}{\sum_{j=1}^c e^{s_j}})) + \lambda \|w\|_2^2$$

forward propagation

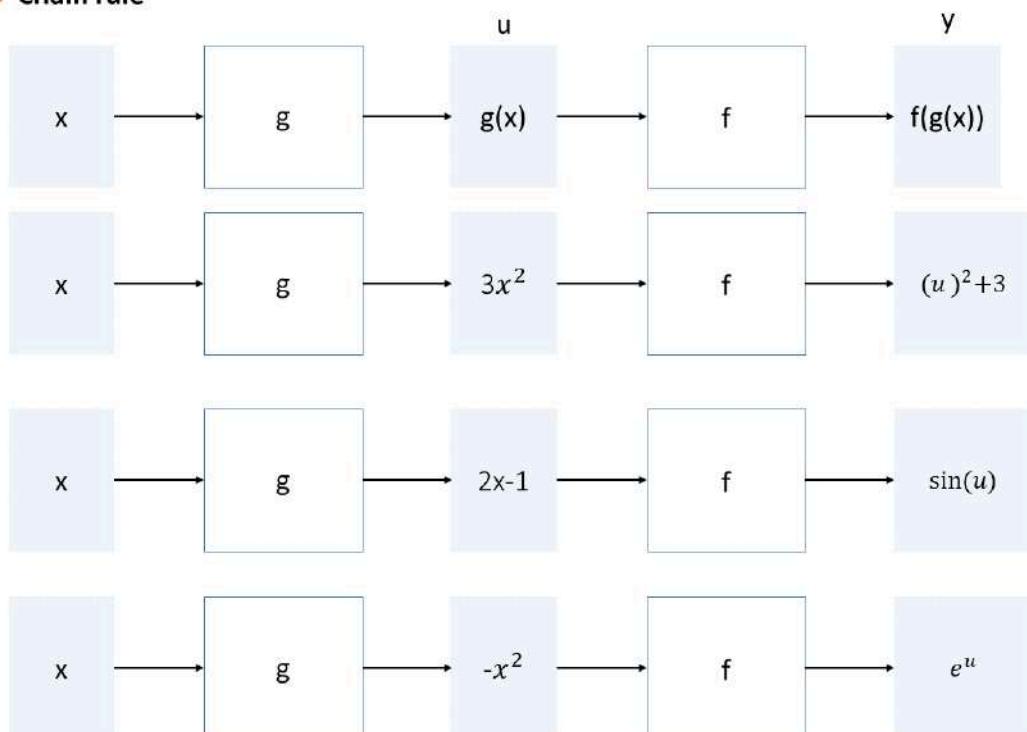


values of input      hidden layer      values of output

back propagation  
(original)

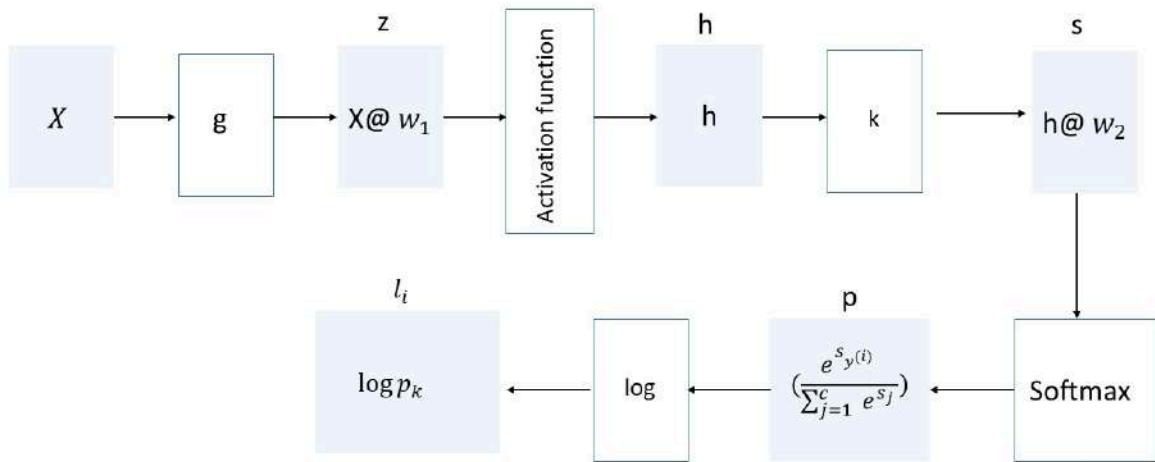
gradient of input      hidden layer      gradient of output

### Chain rule



$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

### Chain rule



$$\frac{dl_i}{dw_2} = \frac{dl_i}{dp} \cdot \frac{dp}{ds} \cdot \frac{ds}{dw_2}$$

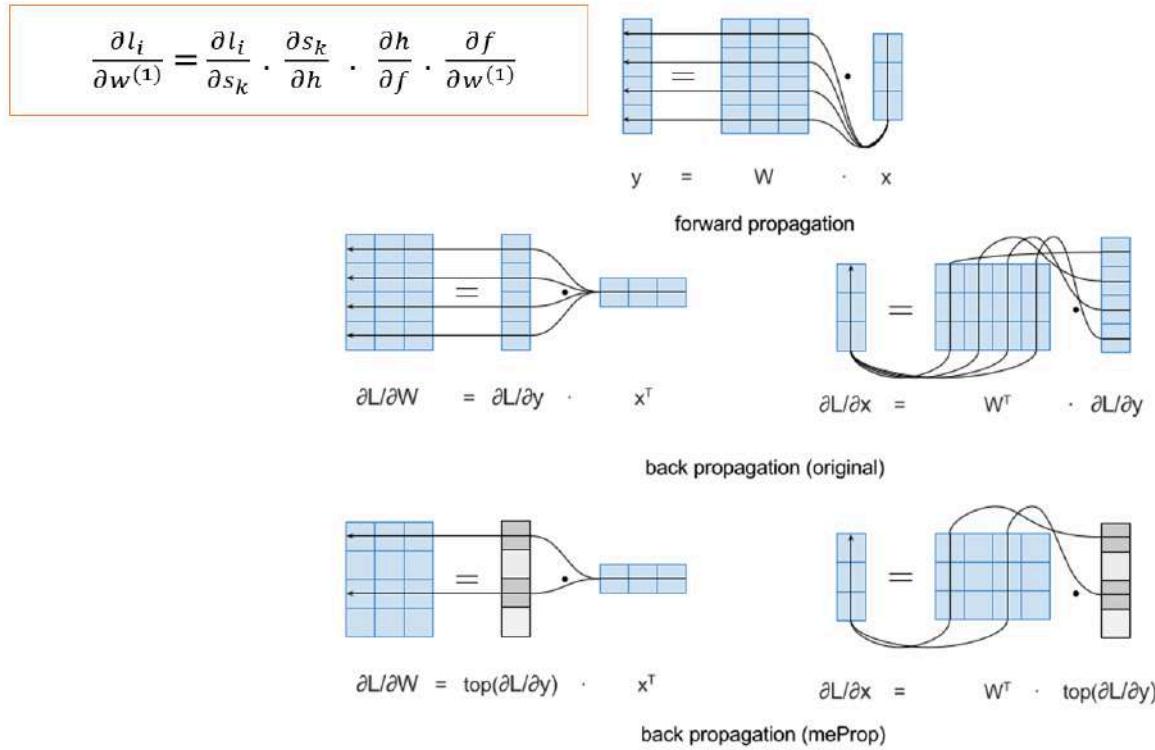
$$(-\frac{1}{p}) \cdot p(1-p) \cdot h = (p-1) \cdot h \quad h.T @ dscores$$

$$\frac{dl_i}{dw_1} = \frac{dl_i}{dp} \cdot \frac{dp}{ds} \cdot \frac{ds}{dh} \cdot \frac{dh}{dz} \cdot \frac{dz}{dw_1}$$

$$(-\frac{1}{p}) \cdot p(1-p) \cdot w_2 \cdot 1 \cdot x = (p-1) \cdot w_2 \cdot x$$

X.T @ dscores @ w2.T

### Backpropagation



```

In [1]: 
 1 import numpy as np
 2 import seaborn as sns
 3 import matplotlib.pyplot as plt
 4 from sklearn.manifold import TSNE
 5 from sklearn.metrics import confusion_matrix
 6 #from utils import softmax
 7
 8
 9 def plot_random_samples(X_train, y_train, classes=None, samples_per_class=10, shape=(28, 28), figsize=(5, 4), show_titles=False):
10     num_classes = len(classes)
11
12     plt.figure(figsize=figsize)
13     for y, label in enumerate(classes):
14         idxs = np.flatnonzero(y_train == y)
15         idxs = np.random.choice(idxs, samples_per_class, replace=False)
16         for i, idx in enumerate(idxs):
17             plt_idx = i * num_classes + y + 1
18             plt.subplot(samples_per_class, num_classes, plt_idx)
19             plt.imshow(X_train[idx].reshape(shape), cmap=plt.cm.Greys)
20             plt.axis('off')
21             if i == 0 and show_titles:
22                 plt.title(label)
23     plt.show()
24
25 def plot_sample(X, y, idx=None, annot=False, shape=(28, 28)):
26     if idx is None:
27         idx = np.random.randint(0, X.shape[0] + 1)
28
29     x = X[idx].reshape(shape)
30
31     figsize = (shape[0] // 2, shape[1] // 2)
32     plt.figure(figsize=figsize)
33     sns.heatmap(x, annot=annot, cmap=plt.cm.Greys, cbar=False)
34     plt.title(y[idx])
35     plt.xticks([])
36     plt.yticks([])
37     plt.show()
38
39
40 def plot_tsne(X, y):
41     plt.figure(figsize=(14, 8))
42     X_embedded = TSNE(n_components=2).fit_transform(X)
43
44     cmap = plt.cm.Spectral
45     for c in range(10):
46         l = np.flatnonzero(c == y)
47         plt.scatter(X_embedded[l, 0], X_embedded[l, 1], cmap=cmap, alpha=0.5, label="%d" %c)
48
49     plt.xticks([])
50     plt.yticks([])
51     plt.legend(loc='best')
52     plt.show()
53
54
55 def predict_and_plot(probs, x, y, mu, classes, shape=(28, 28)):
56     plt.figure(figsize=(10, 4))
57
58     # plot the digit
59     plt.subplot(1, 2, 1)
60     x = x + mu
61     plt.imshow(x.reshape(shape), interpolation='nearest', cmap=plt.cm.Greys)
62     plt.xticks([])
63     plt.yticks([])
64     plt.title(classes[np.argmax(probs)])
65
66     # plot top 5 predictions
67     idx = np.argsort(probs)[5:]
68     tick_label = [classes[i] for i in idx]
69     color = ['g' if i == y else 'r' for i in idx]
70
71     plt.subplot(1, 2, 2)
72     plt.barh(range(5), width=probs[idx], tick_label=tick_label, color=color)
73     plt.xlim(0, 1)
74     plt.title("Top 5 predictions")
75     plt.show()
76
77
78 def plot_confusion_matrix(y_true, y_pred, normalize=False, figsize=(12, 12)):
79     cm = confusion_matrix(y_true, y_pred, labels=range(10))
80     plt.figure(figsize=figsize)
81     annot = cm/cm.sum(axis=1) if normalize else True
82     sns.heatmap(cm, annot=annot, cmap=plt.cm.Blues, cbar=False)
83     plt.title("Confusion Matrix")
84     plt.show()

```

```

In [2]: 
 1 import pandas as pd
 2 data=pd.read_csv("mnist_784.csv")
 3 data.head(5)
 4

```

```

Out[2]: 
   pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  pixel9  pixel10 ...  pixel176  pixel177  pixel178  pixel179  pixel1780  pixel1781  pixel1782  pixel1783  pixel1784  class
0       0       0       0       0       0       0       0       0       0       0 ...       0       0       0       0       0       0       0       0       0       0       0       5
1       0       0       0       0       0       0       0       0       0       0 ...       0       0       0       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0       0 ...       0       0       0       0       0       0       0       0       0       0       0       4
3       0       0       0       0       0       0       0       0       0       0 ...       0       0       0       0       0       0       0       0       0       0       0       1
4       0       0       0       0       0       0       0       0       0       0 ...       0       0       0       0       0       0       0       0       0       0       0       0

```

5 rows × 785 columns

```

In [3]: 
 1 X=np.array(data.drop(["class"], axis=1))
 2 y=np.array(data["class"])
 3
 4 y=y.reshape((-1,1))
 5
 6 X.shape

```

Out[3]: (70000, 784)

```
In [4]: #normalize  
1 Xmax, Xmin = X.max(), X.min()  
2 X = (X - Xmin)/(Xmax - Xmin)  
3  
4
```

```
In [5]: M 1 import sklearn.model_selection
2 X_train , X_test , y_train , y_test = sklearn.model_selection.train_test_split(X, y , test_size= 0.3)
3
4
5
6 print('Training data shape:      ', X_train.shape)
7 print('Training labels shape:     ', y_train.shape)
8 #print('Validation data shape:    ', X_valid.shape)
9 #print('Validation Labels shape:', y_valid.shape)
10 print('Test data shape:          ', X_test.shape)
11 print('Test labels shape:         ', y_test.shape)
```

```
Training data shape: (49000, 784)
Training labels shape: (49000, 1)
Test data shape: (21000, 784)
Test labels shape: (21000, 1)
```

## Visualize data

```
In [6]: 1 classes = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
In [7]: 1 plot_random_samples(X_train, y_train, classes, samples_per_class=10)
```

```
In [9]: 1 plot_sample(X_train, y_train, annot=True, idx=None)
```

[6]



```

In [ ]: M 1 import numpy as np
2
3 class TwoLayerNeuralNetwork:
4
5     def __init__(self, num_features=784, num_hiddens=20, num_classes=10):
6         self.num_features = num_features
7         self.num_hiddens = num_hiddens
8         self.num_classes = num_classes
9
10        # Random initialization: create random weights, set all biases to zero
11        self.params = {}
12        self.params['W1'] = np.random.randn(num_features, num_hiddens) * 0.001
13        self.params['W2'] = np.random.randn(num_hiddens, num_classes) * 0.001
14        self.params['b1'] = np.zeros((num_hiddens,))
15        self.params['b2'] = np.zeros((num_classes,))
16
17    def softmax(self, x):
18        x = x - np.max(x, axis=1, keepdims=True)
19        exp_x = np.exp(x)
20        return exp_x / np.sum(exp_x, axis=1, keepdims=True)
21
22    def softmax_loss(self, scores, y, mode='train'):
23        m = scores.shape[0]
24        probs = self.softmax(scores)
25        loss = -np.sum(np.log(probs[range(m), y])) / m
26
27        if mode != 'train':
28            return loss
29
30        # Backward
31        dscores = probs
32        dscores[range(m), y] -= 1.0
33        dscores /= m
34
35        return loss, dscores
36
37    def forward(self, X):
38        W1, b1 = self.params['W1'], self.params['b1']
39        W2, b2 = self.params['W2'], self.params['b2']
40
41        # Forward step
42        h_in = X @ W1 + b1      # Hidden Layer input
43        h = np.maximum(0, h_in)   # Hidden Layer output (using ReLU)
44        scores = h @ W2 + b2    # Neural net output
45
46        return scores, h, h_in
47
48    def train_step(self, X, y):
49        scores, h, h_in = self.forward(X)
50
51        # Compute Loss
52        loss, dscores = self.softmax_loss(scores, y)
53
54        # Backward step
55        db2 = dscores.sum(axis=0)
56        dw2 = h.T @ dscores
57
58        dh = dscores @ self.params['W2'].T
59        dh[h_in < 0] = 0.0
60
61        db1 = dh.sum(axis=0)
62        dw1 = X.T @ dh
63
64        gradient = {'W1': dw1, 'b1': db1, 'W2': dw2, 'b2': db2}
65
66        return loss, gradient
67
68    def train(self, X_train, y_train, X_valid, y_valid, batch_size=50,
69              alpha=0.001, lmbda=0.0001, num_epochs=10):
70
71        m, n = X_train.shape
72        num_batches = m // batch_size
73
74        report = "{:3d}: training loss = {:.2f} | validation loss = {:.2f}"
75
76        losses = []
77        for epoch in range(num_epochs):
78            train_loss = 0.0
79
80            for _ in range(num_batches):
81                # Select a random mini-batch
82                batch_idx = np.random.choice(m, batch_size, replace=False)
83                X_batch, y_batch = X_train[batch_idx], y_train[batch_idx]
84
85                # Train on mini-batch
86                data_loss, gradient = self.train_step(X_batch, y_batch)
87                reg_loss = 0.5 * (np.sum(self.params['W1'] ** 2) + np.sum(self.params['W2'] ** 2))
88                train_loss += (data_loss + lmbda * reg_loss)
89                losses.append(data_loss + lmbda * reg_loss)
90
91                # Regularization
92                gradient['W1'] += lmbda * self.params['W1']
93                gradient['W2'] += lmbda * self.params['W2']
94
95                # Update parameters
96                for p in self.params:
97                    self.params[p] = self.params[p] - alpha * gradient[p]
98
99                # Report training loss and validation loss
100               train_loss /= num_batches
101               valid_loss = self.softmax_loss(self.forward(X_valid)[0], y_valid, mode='test')
102               print(report.format(epoch + 1, train_loss, valid_loss))
103
104        return losses
105
106    def predict(self, X):
107        """Predict labels for input data."""
108        scores = self.forward(X)[0]
109        return np.argmax(scores, axis=1)
110
111    def predict_proba(self, X):
112        """Predict probabilities of classes for each input data."""
113        scores = self.forward(X)[0]
114        return self.softmax(scores)
115
116    def accuracy(self, y_pred, y_true):
117        return 100. * np.mean(y_pred == y_true)

```

```

119

In [ ]: 1 params = {}
2 params['W1'] = np.random.randn(784, 20) * 0.001
3 params['W2'] = np.random.randn(20, 10) * 0.001
4 params['b1'] = np.zeros((20,))
5 params['b2'] = np.zeros((10,))
6
7 params

In [ ]: 1 mlp = TwoLayerNeuralNetwork(num_hiddens=10)
2 mlp.train_step(X_train, y_train)[0]

In [ ]: 1 train_acc = accuracy(mlp.predict(X_train), y_train)
2 print("Training accuracy = {:.2f}%".format(train_acc))
3
4 test_acc = accuracy(mlp.predict(X_test), y_test)
5 print("Validation accuracy = {:.2f}%".format(test_acc))

plt.plot(losses) plt.xlabel('Epoch') plt.ylabel('Loss') plt.xticks(range(0, 10001, 1000), range(0, 11)) plt.show()

In [ ]: 1 test_acc = accuracy(predict(w, b, X_test), y_test)
2 print('Test accuracy = {:.2f}%'.format(train_acc))

In [ ]: 1 for i in range(10):
2     idx = np.random.choice(len(y_test))
3     probs = mlp.predict_proba(X_test[idx].reshape((1, -1)))[0]
4     predict_and_plot(probs, X_test[idx], y_test[idx], mu, classes)

In [ ]: 1 count = 10
2 y_pred = mlp.predict(X_test)
3 idx = np.flatnonzero(y_pred != y_test)
4 sample_idx = np.random.choice(idx, count)
5
6 for i in sample_idx:
7     probs = mlp.predict_proba(X_test[i].reshape((1, -1)))[0]
8     predict_and_plot(probs, X_test[i], y_test[i], mu, classes)

In [ ]: 1 y_pred = mlp.predict(X_test)
2 plot_confusion_matrix(y_test, y_pred, normalize=True)

In [ ]: 1 W = mlp.params['W1']
2 print(W.shape)

In [ ]: 1 for i in range(20):
2     plt.subplot(4, 5, i+1)
3     plt.imshow(W[:, i].reshape((28, 28)), cmap=plt.cm.coolwarm)
4     plt.xticks([])
5     plt.yticks([])
6     plt.show()

```

## Neural networks in scikit learn

```

In [12]: 1 from sklearn.neural_network import MLPClassifier

In [ ]: 1 model = MLPClassifier(hidden_layer_sizes=(20,), learning_rate='adaptive', alpha=0.1, max_iter=50, verbose=1)
2 model.fit(X_train, y_train);

In [14]: 1 train_acc = model.score(X_train, y_train)
2 print("Train accuracy = {:.2f}%".format(train_acc * 100))
3
4 test_acc = model.score(X_test, y_test)
5 print("Test accuracy = {:.2f}%".format(test_acc * 100))

Train accuracy = 97.31%
Test accuracy = 95.87%

```

## example2

```

In [15]: 1 from sklearn.datasets import load_iris
2
3 from sklearn.model_selection import train_test_split
4
5 iris = load_iris()
6 X = iris.data
7 y = iris.target
8
9
10
11 X_train , X_test , y_train, y_test = train_test_split(X , y , random_state= 12 , test_size=0.2)
12
13

In [22]: 1 model2 = MLPClassifier(hidden_layer_sizes=(20,), learning_rate='adaptive', alpha=0.2, max_iter=1000, verbose=0)
2 model2.fit(X_train, y_train);

F:\Users\nikoo\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10
00) reached and the optimization hasn't converged yet.
warnings.warn(

```

```
In [23]: 1 train_acc = model2.score(X_train, y_train)
2 print("Train accuracy = {:.2f}%".format(train_acc * 100))
3
4 test_acc = model2.score(X_test, y_test)
5 print("Test accuracy = {:.2f}%".format(test_acc * 100))
```

Train accuracy = 98.33%  
Test accuracy = 96.67%

### example3

```
In [24]: 1 from sklearn.datasets import load_digits
2 mnist=load_digits()
3 X= mnist.data
4 y=mnist.target
5
6 # split
7 from sklearn.model_selection import train_test_split
8
9 X_train , X_test , y_train , y_test = train_test_split(X , y , random_state= 1456 , test_size=0.3)
```

```
In [ ]: 1 model3 = MLPClassifier(hidden_layer_sizes=(20,), learning_rate='adaptive', alpha=0.1, max_iter=100, verbose=1)
2 model3.fit(X_train, y_train);
```

```
In [28]: 1 train_acc = model3.score(X_train, y_train)
2 print("Train accuracy = {:.2f}%".format(train_acc * 100))
3
4 test_acc = model3.score(X_test, y_test)
5 print("Test accuracy = {:.2f}%".format(test_acc * 100))
```

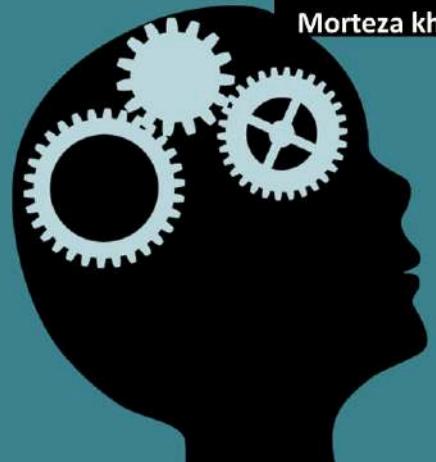
Train accuracy = 99.60%  
Test accuracy = 93.52%

```
In [ ]: 1
```

# Machine learning

K-MEANS – K-MODE –  
KMEANS ++ - k-medoids

Morteza khorsand



## K - MEANS

2

### Clustering K-Means

How does the K-Means Algorithm Work?

Step-1: Select random K points or centroids as the center of clusters.

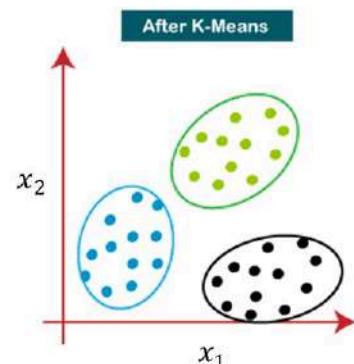
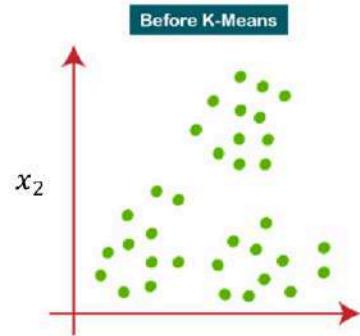
Step-2: Assign each data point to its closest centroid, which will form the predefined K clusters.

Step-3: Calculate the variance and place a new centroid of each cluster.

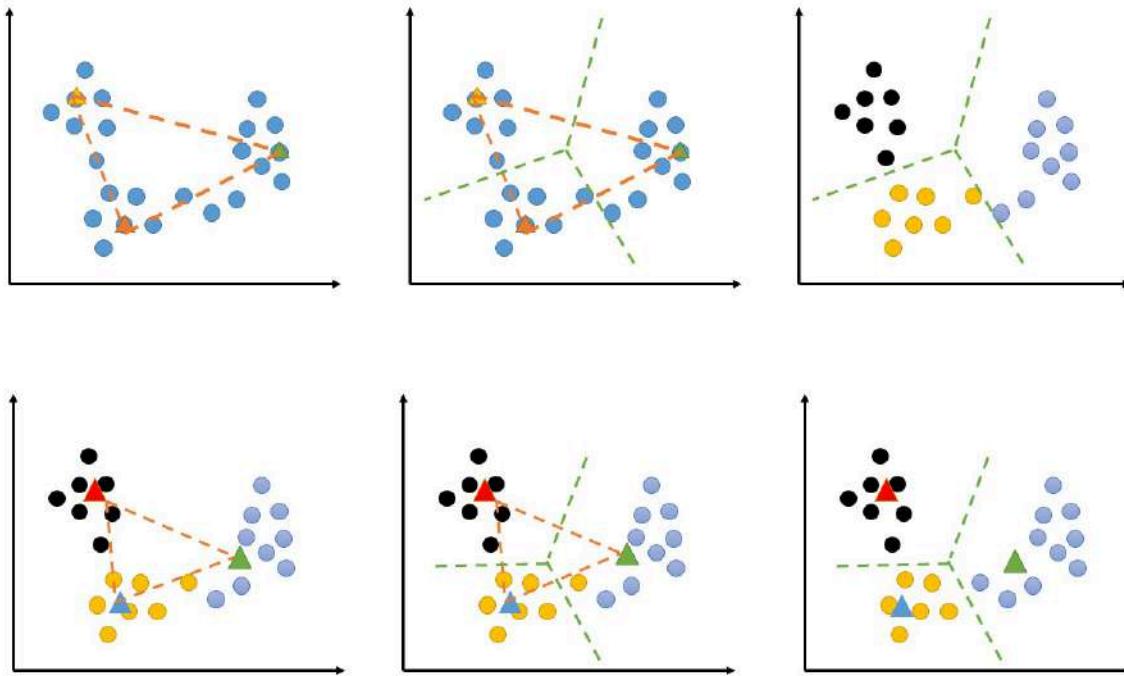
Step-4: Repeat the third step, which means reassigning each data point to the new closest centroid of each cluster.

Step-5: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-6: The model is ready.



## K - MEANS



```
In [1]: #import Libraries for plot
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.cluster._kmeans")

import os
os.environ['OMP_NUM_THREADS'] = '4'
%matplotlib inline

import math
import scipy
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import ipywidgets as widgets

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# ignore warnings
import warnings
warnings.filterwarnings('ignore')

# matplotlib setup
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['figure.dpi'] = 120
plt.rcParams['image.cmap'] = 'Spectral'

def plot_kmeans(X, initial_centroids, plot=True):
    m = X.shape[0]
    K = initial_centroids.shape[0]
    centroids = np.array(initial_centroids)
    old_cluster_ids = np.zeros((m,))

    while True:
        # find closest center to each data
        cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - centroids, axis=1)) for i in range(m)])

        if plot:
            plot_clusters(X, cluster_ids, centroids, 'Assigning Data to Clusters')

        # update cluster centers
        for k in range(K):
            centroids[k] = np.mean(X[cluster_ids==k], axis=0)

        if plot:
            plot_clusters(X, cluster_ids, centroids, 'Updating Cluster Centers')

        # stop
        if np.all(cluster_ids == old_cluster_ids):
            return cluster_ids, centroids
        else:
            old_cluster_ids = cluster_ids

def plot_clusters(X, cluster_ids, centroids, title):
    K = centroids.shape[0]

    plt.figure()
    plt.scatter(X[:, 0], X[:, 1], s=50, c=cluster_ids, edgecolors='k', alpha=0.6)
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='^', s=200, c=range(K), edgecolors='k')
    plt.title(title)
```

```

plt.axis('equal')
plt.show()

from sklearn.datasets import make_blobs
from sklearn.metrics.pairwise import euclidean_distances

def plot_kmeans_interactive(min_clusters=1, max_clusters=6):
    with warnings.catch_warnings():
        warnings.filterwarnings('ignore')

    X, y = make_blobs(n_samples=300, centers=4, random_state=0, cluster_std=0.6)

    def kmeans_step(frame, K):
        rng = np.random.RandomState(2)
        cluster_ids = np.zeros(X.shape[0])
        centroids = rng.randn(K, 2)

        nsteps = frame // 3
        for i in range(nsteps + 1):
            old_centroids = centroids

            if i < nsteps or frame % 3 > 0:
                dist = euclidean_distances(X, centroids)
                cluster_ids = dist.argmin(1)

            if i < nsteps or frame % 3 > 1:
                centroids = np.array([X[cluster_ids==k].mean(0) for k in range(K)])
                nans = np.isnan(centroids)
                centroids[nans] = old_centroids[nans]

    # plot data
    c = cluster_ids if frame > 0 else 'w'
    plt.scatter(X[:, 0], X[:, 1], c=c, s=50, edgecolors='k', vmin=0, vmax=K - 1, alpha=0.6);

    # plot centroids
    plt.scatter(old_centroids[:, 0], old_centroids[:, 1], marker='o', c=range(K), s=200)
    plt.scatter(old_centroids[:, 0], old_centroids[:, 1], marker='o', c='black', s=50)

    # plot new centers if third frame
    if frame % 3 == 2:
        for i in range(K):
            plt.annotate("", xy=centroids[i], xytext=old_centroids[i],
                        arrowprops=dict(arrowstyle='->', linewidth=1, color='k'))
        plt.scatter(centroids[:, 0], centroids[:, 1], marker='o', c=range(K), s=200)
        plt.scatter(centroids[:, 0], centroids[:, 1], marker='o', c='black', s=50)

    plt.xlim(-4, 4)
    plt.ylim(-2, 10)

    if frame % 3 == 1:
        plt.title("Assign data to nearest centroid", size=14)
    elif frame % 3 == 2:
        plt.title("Update centroids to cluster means", size=14)
    else:
        plt.title(" ", size=14)

    frame = widgets.IntSlider(value=0, min=0, max=50, step=1, desc='frame')
    K = widgets.IntSlider(value=4, min=min_clusters, max=max_clusters, desc='K')

    return widgets.interact(kmeans_step, frame=frame, K=K)

```

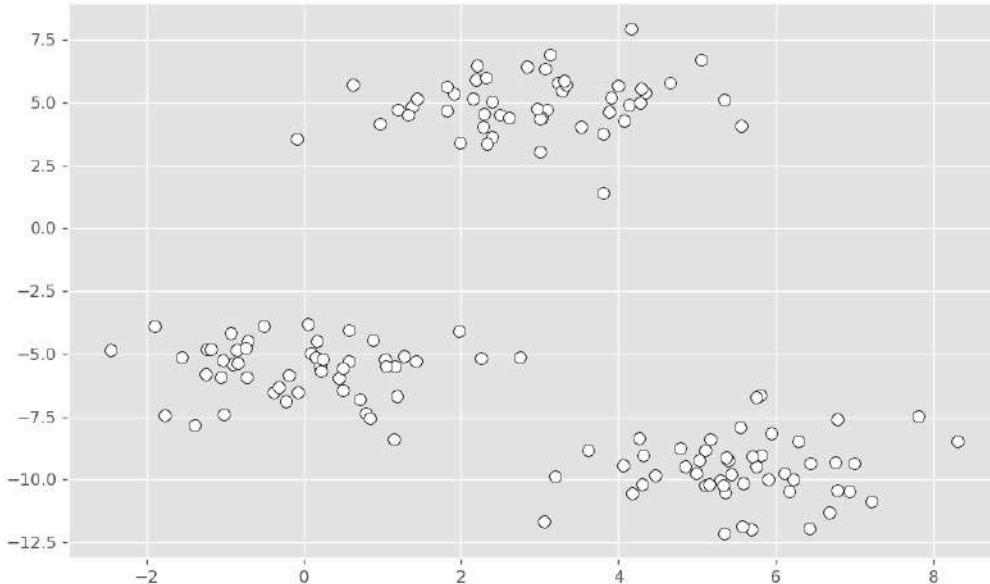
In [2]:

```

# create random data
X, y = make_blobs(n_samples=150, centers=3, cluster_std=1.2, random_state=10)

# plot data
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w');

```



In [3]:

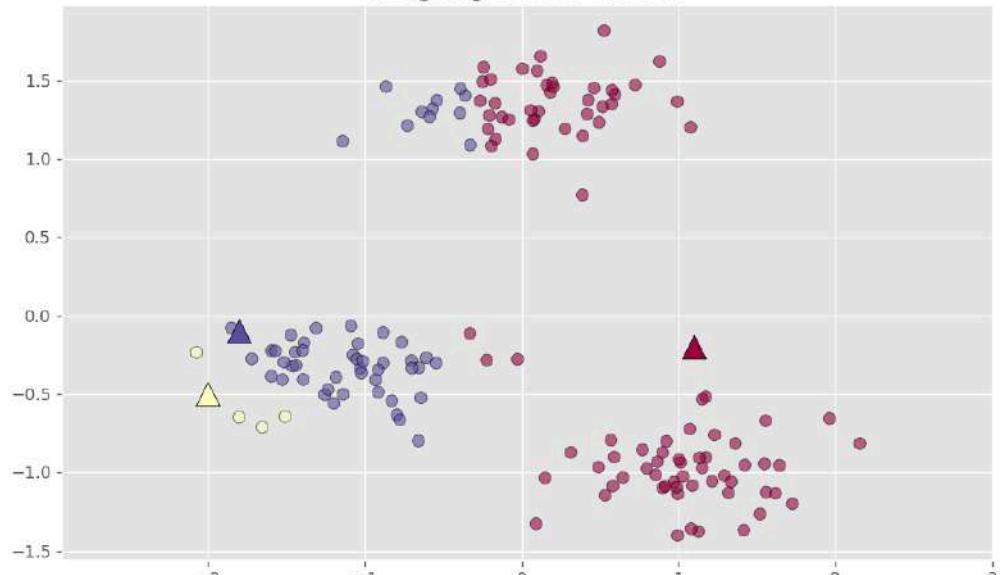
```

# Normalize X
mu = X.mean(axis=0)
std = X.std(axis=0)
X = (X - mu) / std

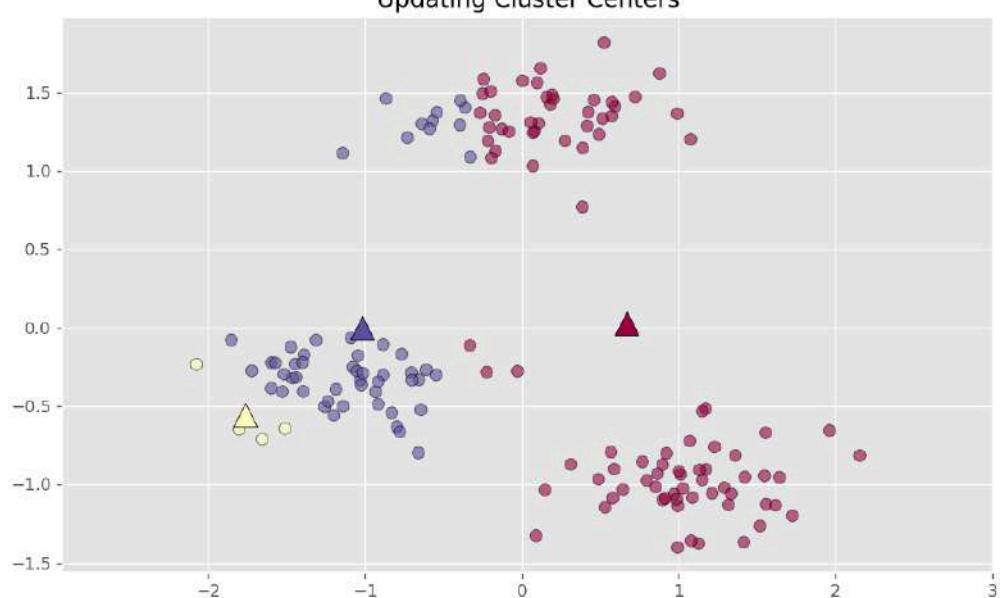
# plot kmeans steps
initial_centroids = np.array([[1.1, -0.2], [-2.0, -0.5], [-1.8, -0.1]])
cluster_ids, centroids = plot_kmeans(X, initial_centroids)

```

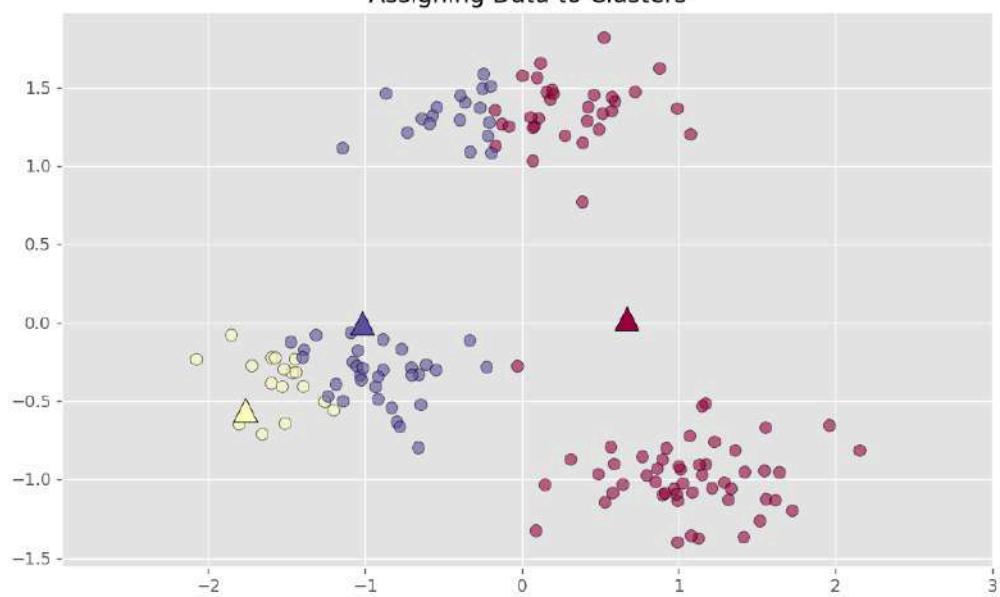
Assigning Data to Clusters



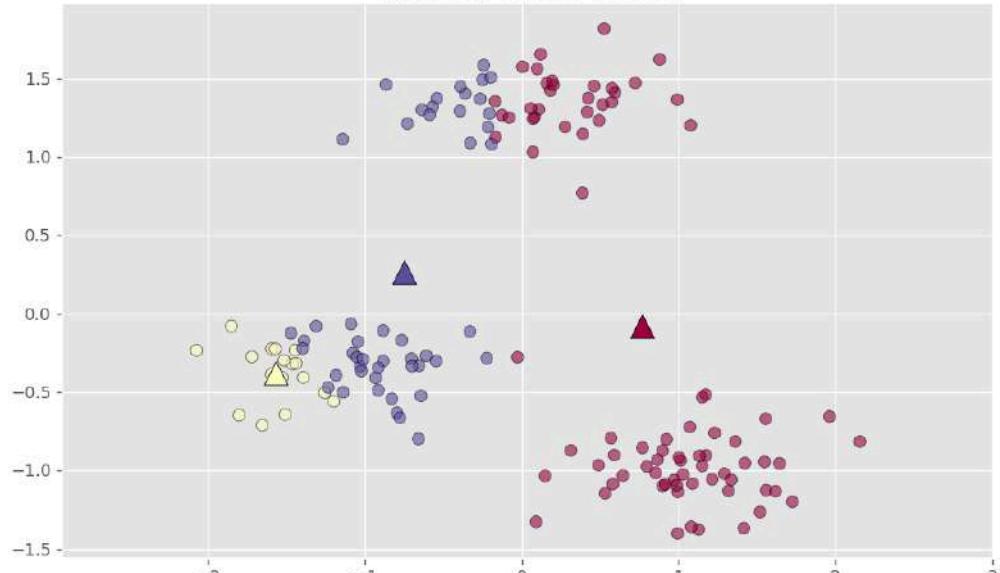
Updating Cluster Centers



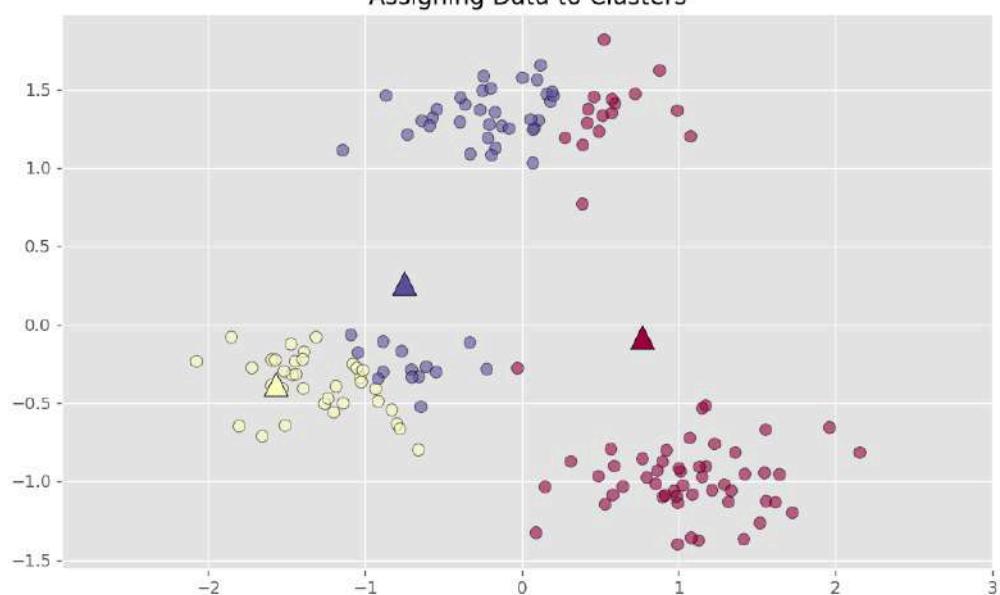
Assigning Data to Clusters



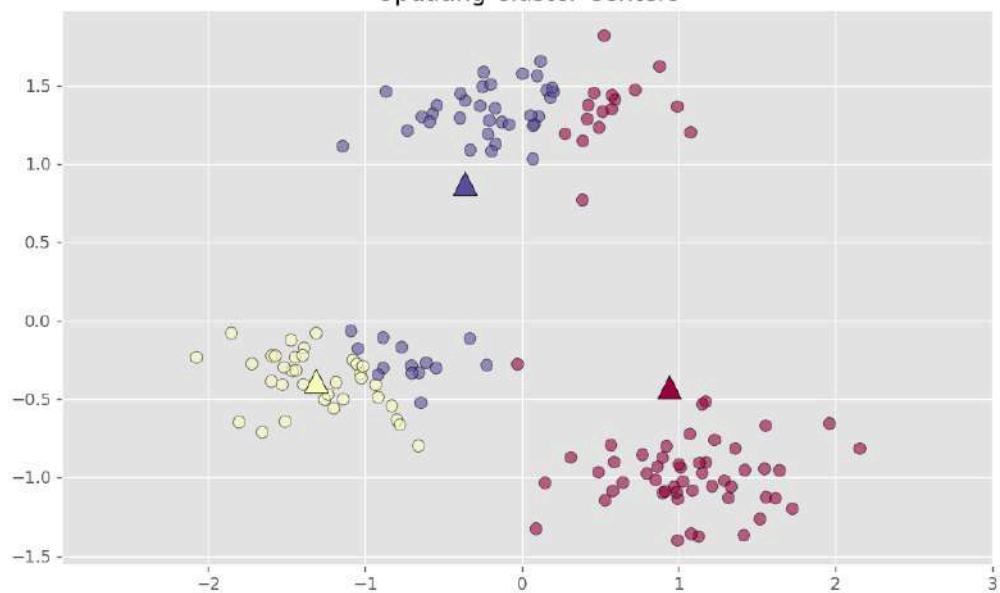
Updating Cluster Centers



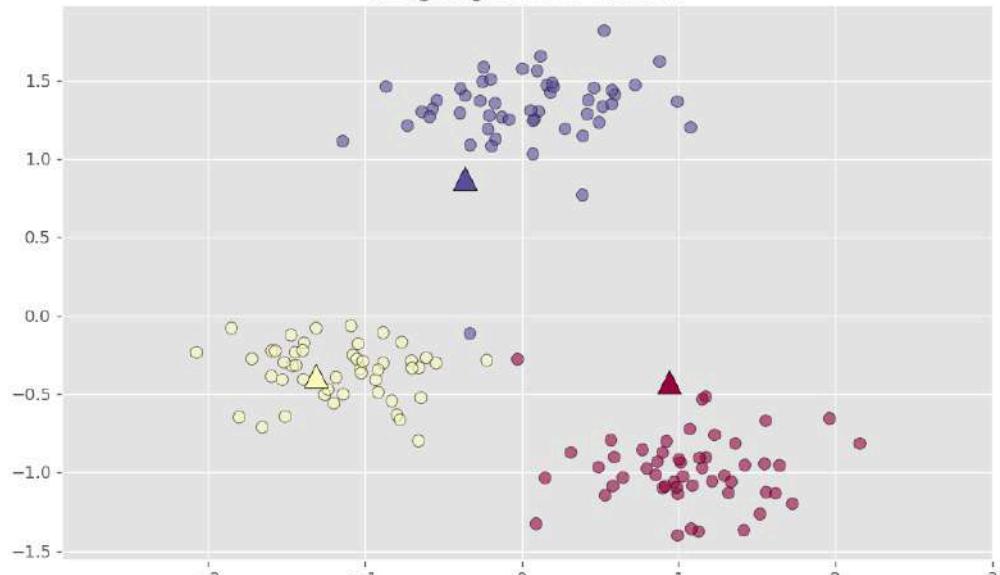
Assigning Data to Clusters



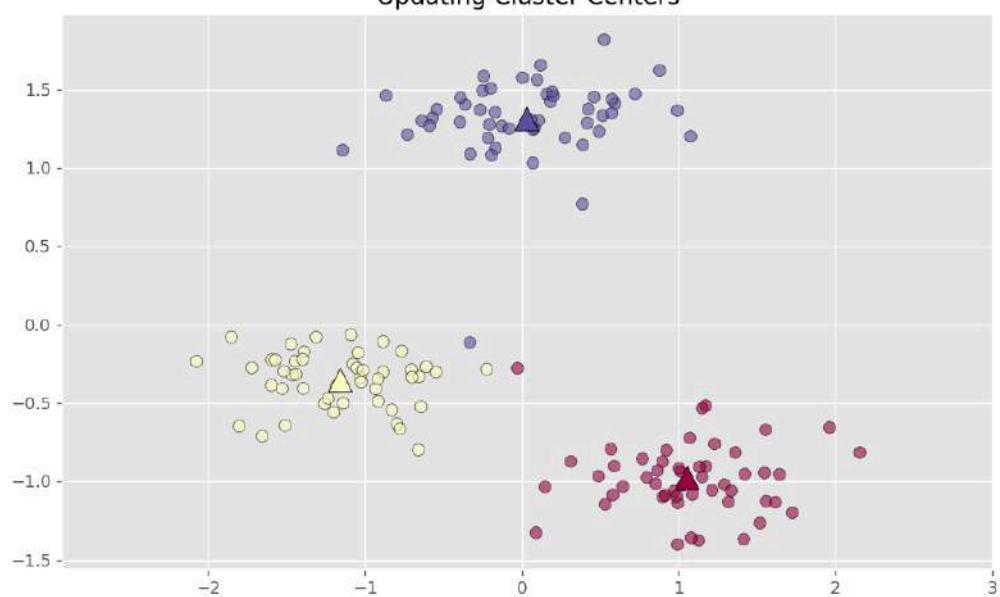
Updating Cluster Centers



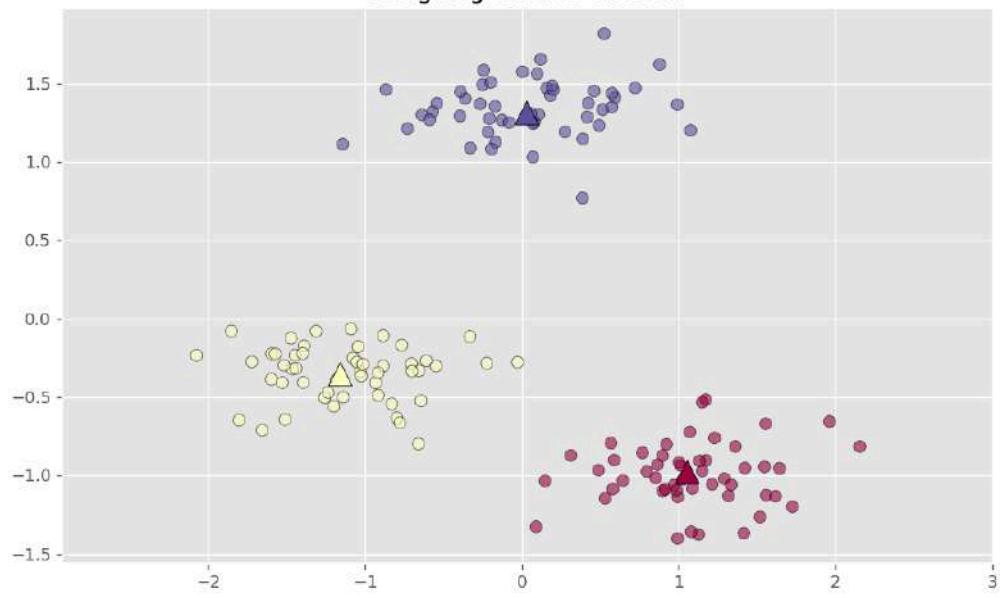
Assigning Data to Clusters



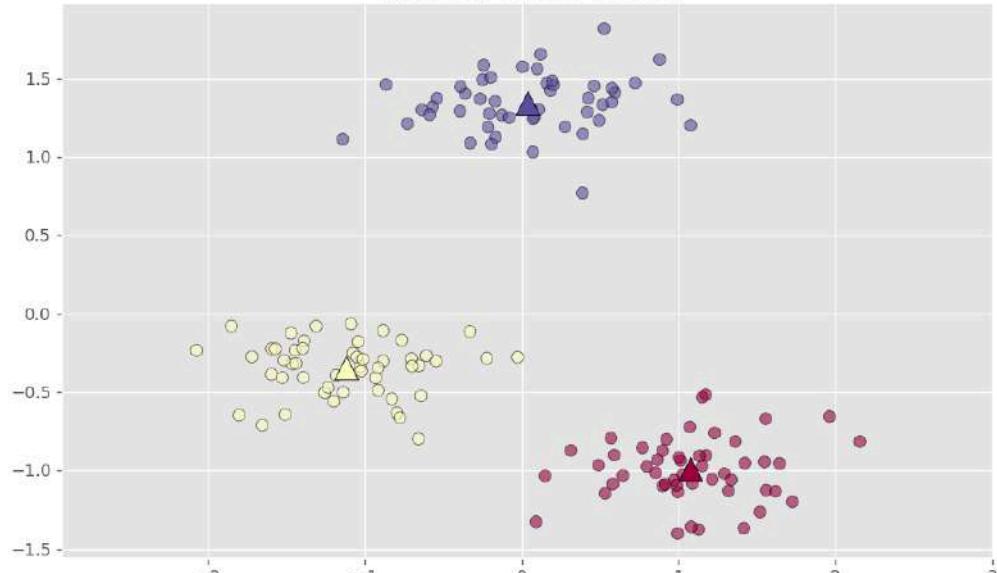
Updating Cluster Centers



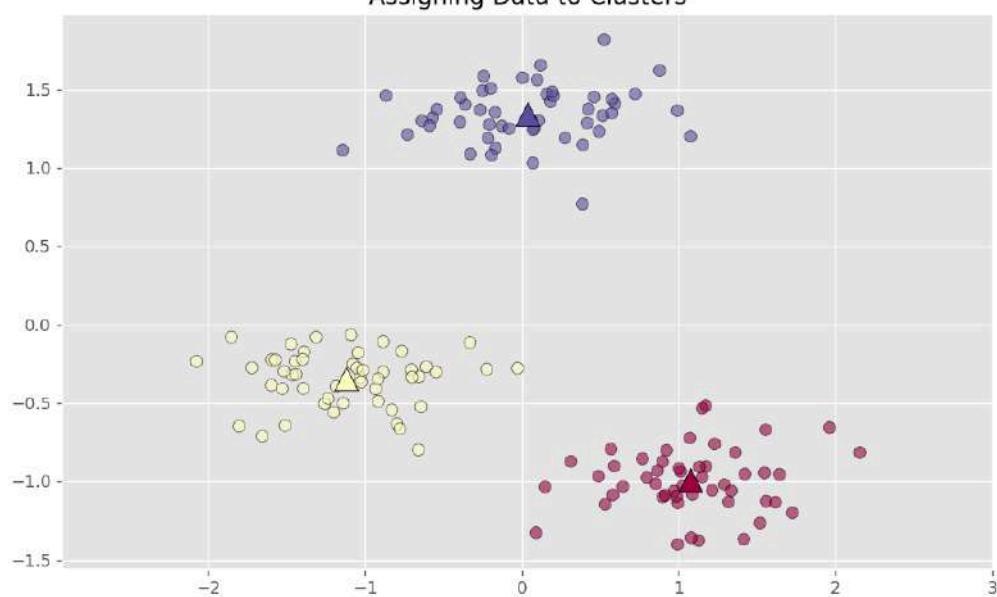
Assigning Data to Clusters



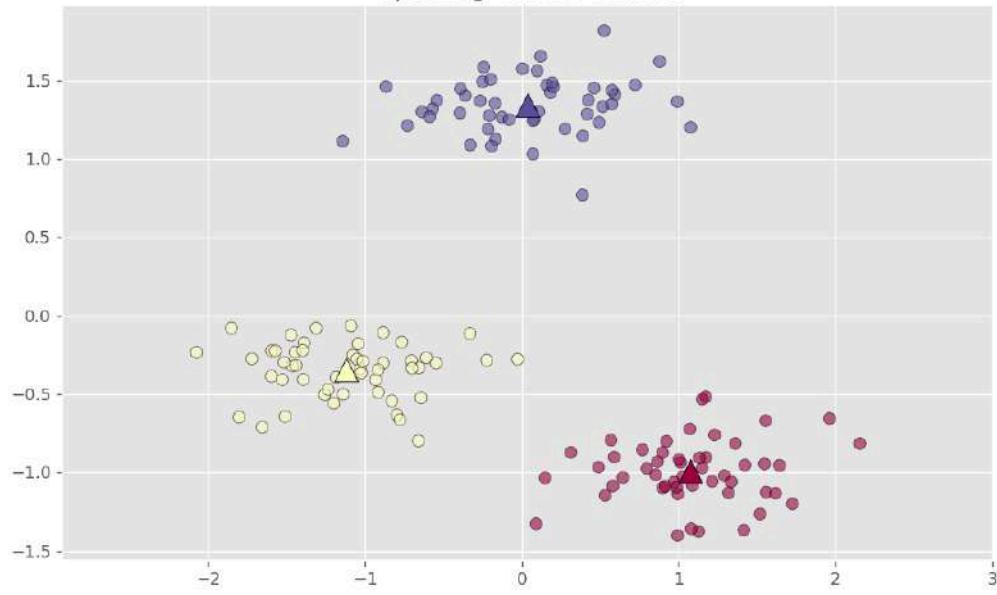
Updating Cluster Centers



Assigning Data to Clusters



Updating Cluster Centers



```
In [4]: plot_kmeans_interactive();  
interactive(children=(IntSlider(value=0, description='frame', max=50), IntSlider(value=4, description='K', max...
```

## K - MEANS

### Pseudo code

randomly initiate K cluster centroid ( $\mu_1, \mu_2, \dots, \mu_k \in R^n$ )

Repeat

{

for i = 1 to m

$c^{(i)} = argmin \|x^{(i)} - \mu_k\|$

for k = 1 to k

$\mu_k$  = average of points assigned to cluster K

}

Centroids = np.random.random ((k, n))

While True:

for i in range(m):

$c[i] = np.argmin(np.linalg.norm(x[i] - centroids, axis = 1)$

for k in range(k):

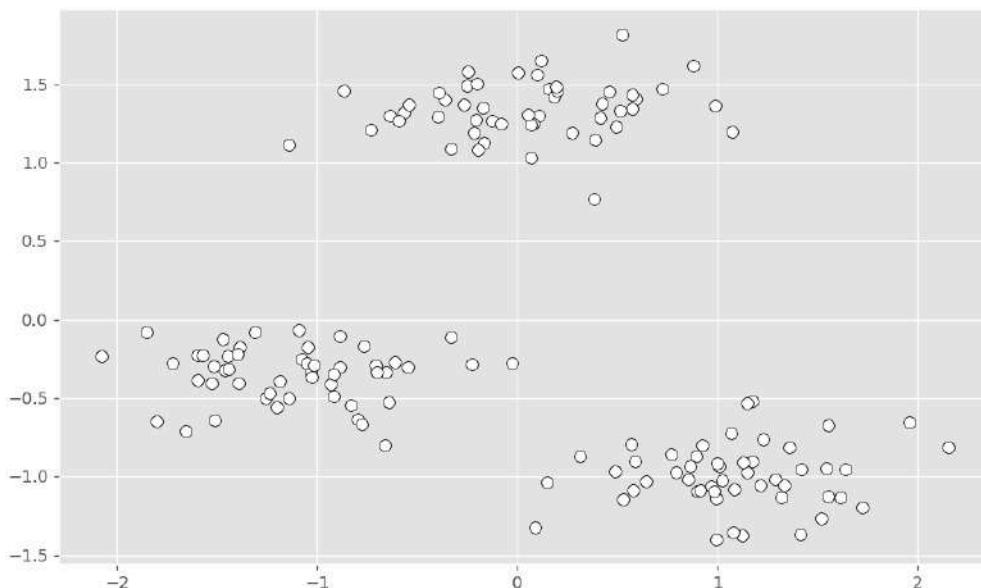
centroids[k] = np.mean(X[c==k], axis = 0)

Termination condition

```
In [5]: #import Libraries and data
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
# create random data
X, y = make_blobs(n_samples=150, centers=3, cluster_std=1.2, random_state=10)

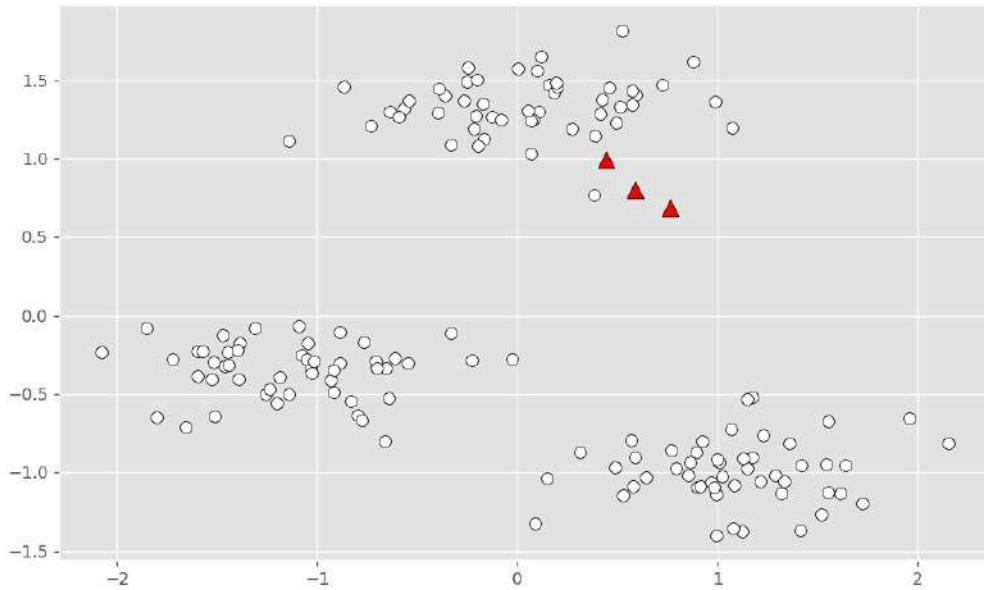
# Normalize X
mu = X.mean(axis=0)
std = X.std(axis=0)
X = (X - mu) / std

# plot data
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w')
plt.show()
```



```
In [6]: m, n = X.shape
K=3
initial_centroids= np.random.rand(K,n)

#plot centeroids
plt.scatter(initial_centroids[:, 0], initial_centroids[:, 1], edgecolors='k', s=100, c='red' , marker = "^")
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w' )
plt.show()
```



```
In [7]: centroids=initial_centroids.copy()
print(centroids)
```

```
[[0.4462534  0.99433332]
 [0.59087501 0.80390867]
 [0.76481699 0.68522131]]
```

```
In [7]: cluster_ids=np.array([np.argmin(np.linalg.norm(X[i] - centroids , axis = 1)) for i in range(m)])
```

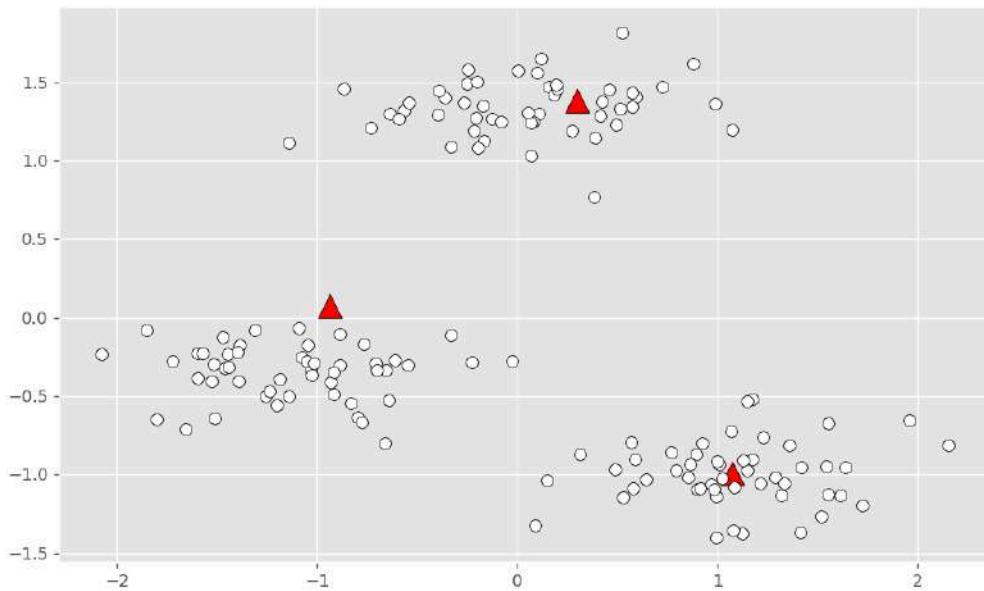
```
print(cluster_ids)
```

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

```
In [8]: for k in range(K):
    centroids[k] = np.mean(X[cluster_ids==k], axis=0) # يعني همه اونایی که مطلاً اندیشور صفر است
    centroids
```

```
Out[8]: array([[ 1.07765271, -0.9922514 ],
 [-0.93411721,  0.07734374],
 [ 0.30116671,  1.38603736]])
```

```
In [9]: plt.scatter(centroids[:, 0], centroids[:, 1], edgecolors='k', s=200, c='red' , marker = "^")
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w' )
plt.show()
```



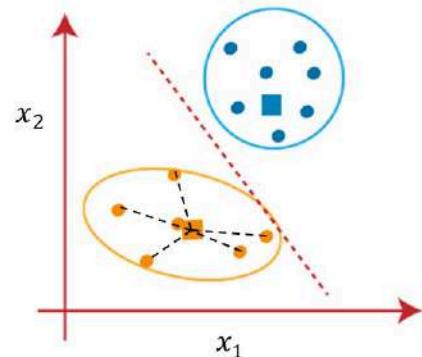
$\mu_k$  : center of clusters

$c^{(i)}$  : index of cluster assigned to  $x^{(i)}$

$\mu_{c^{(i)}}$  : center of cluster assigned to  $x^{(i)}$

Goal function :  $J(c^{(1)}, c^{(2)}, c^{(3)} \dots c^{(m)}, \mu_1, \mu_2, \mu_3 \dots \mu_k)$

$$\text{WCSS} = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_k\|^2$$



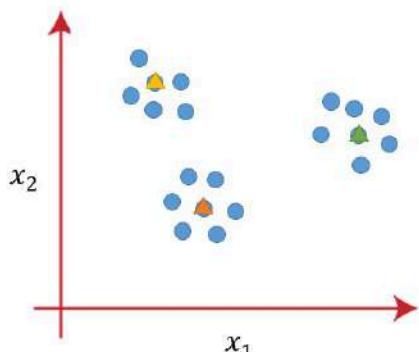
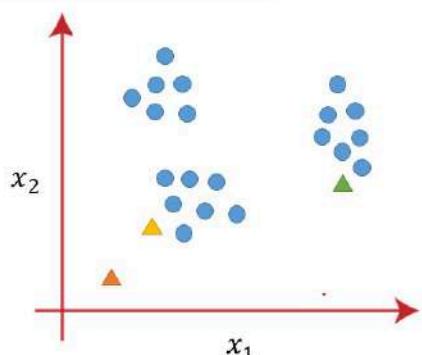
$$\text{WCSS} = \sum_{\text{Pi in Cluster1}} \text{distance(Pi, C}_1)^2 + \sum_{\text{Pi in Cluster2}} \text{distance(Pi, C}_2)^2 + \sum_{\text{Pi in Cluster3}} \text{distance(Pi, C}_3)^2$$

$$\text{WCSS} = \frac{1}{m} \sum_{j=1}^k \sum_{i=1}^m \|x^{(i)} - \mu_k\|^2$$

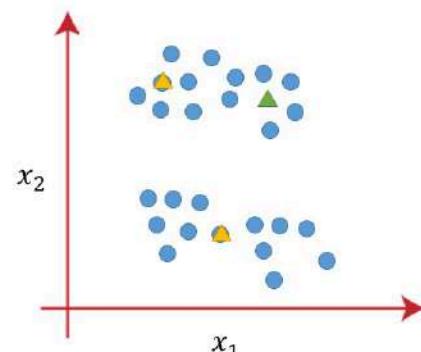
### Problem

Select from existing data

C = np.random.permutation(x) [: k]



Global Optimum



Local Optimum

- To avoid local optimum calculate the algorithm many times.

For t = 1 to max

{

Randomly initialize cluster centroids  $\mu_1, \mu_2, \mu_3 \dots \mu_k$

Run k-means to get  $c^{(1)}, c^{(2)}, c^{(3)} \dots c^{(m)}, \mu_1, \mu_2, \mu_3 \dots \mu_k$

Compute cost function :  $J(c^{(1)}, c^{(2)}, c^{(3)} \dots c^{(m)}, \mu_1, \mu_2, \mu_3 \dots \mu_k)$

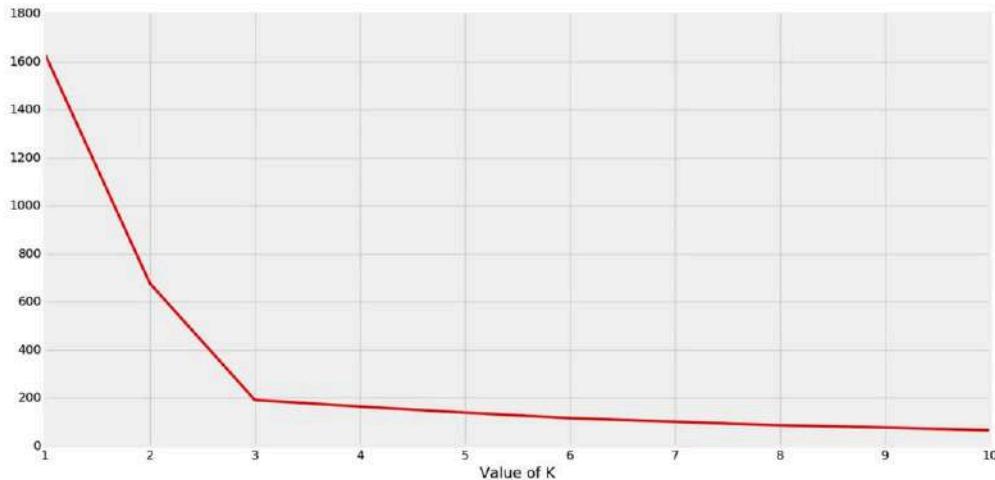
}

Pick cluster with minimum cost

## K - MEANS

### number of clusters

#### Elbow method



```
In [8]: def kmeans(X, initial_centroids):
    m, n = X.shape
    centroids = initial_centroids
    K, _ = initial_centroids.shape
    initial_cluster_ids = np.zeros((m,))
    cost=0

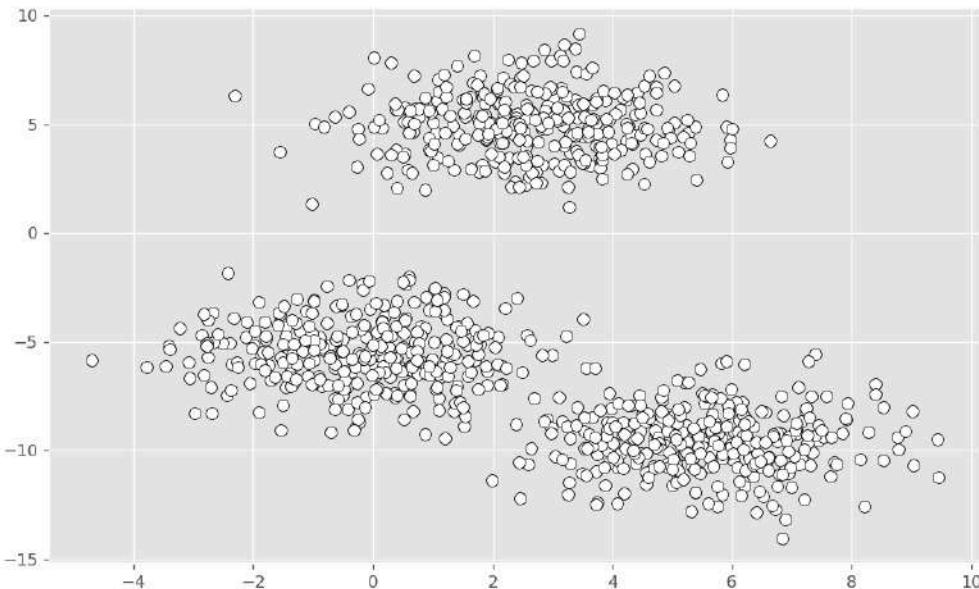
    while True:
        cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - centroids, axis=1)) for i in range(m)])

        # update cluster centers
        for j in range(K):
            centroids[j] = np.mean(X[cluster_ids==j], axis=0) # میانگین بگیر بدار به عنوان مرکز

        # stop
        if np.all(cluster_ids == initial_cluster_ids):
            for z in range(K):
                cost+=1/m * (np.linalg.norm(X[cluster_ids==z] - centroids[z])**2) # هر داده را از مرکز خودش کم کن
            return cost , initial_cluster_ids , centroids
        else:
            initial_cluster_ids = cluster_ids
```

```
In [9]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
```

```
In [10]: X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.5, random_state=10)
# plot data
plt.scatter(X[:, 0], X[:, 1], edgecolors='k', s=50, c='w');
```



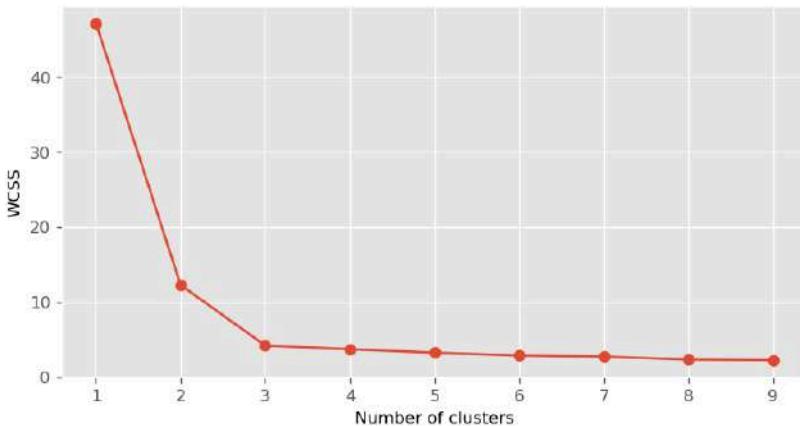
```
In [11]: costs=[]
best_centeroids=[]
final_ids=[]

for i in range(1 , 10):
    initial_centeroids= np.random.permutation(X[ : i])
    A=kmeans(X , initial_centeroids)
    costs.append(A[0])
    final_ids.append(A[1])
    best_centeroids.append(A[2])

print(costs)
print("-----")
print(best_centeroids[2])
print("-----")
print(final_ids[2])

plt.figure(figsize=(8, 4))
plt.plot(range(1, 10), costs, marker='o')
plt.xticks(range(1, 10))
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

[47.029062199797174, 12.21527718761187, 4.167450222191264, 3.6894595293854398, 3.2242865833613026, 2.843484126290323, 2.6948171538429073, 2.3008401012827138, 2.2250155257616107]
[-----]
[-0.13936238 -5.54381213]
[ 2.61164305  4.95788186]
[ 5.5198791 -9.57304198]
[-----]
[2 1 0 0 1 0 0 0 0 2 1 2 2 0 0 2 2 2 0 0 1 1 2 1 2 2 2 2 2 2 1 1
2 1 1 0 1 1 2 1 0 0 2 2 2 0 1 0 1 0 0 2 1 1 0 0 2 1 0 2 1 0 1 1 0 0 2
0 0 2 2 2 0 0 0 1 0 1 2 1 0 0 1 1 2 2 1 0 2 1 2 2 1 1 0 1 1 0 2 0 1 2 0 0
1 2 1 0 2 1 0 0 1 2 2 2 2 2 2 1 0 1 1 1 0 1 0 0 0 2 0 0 2 1 0 2 1 2 0 1
1 0 1 2 2 0 0 1 0 0 1 0 1 0 2 2 1 1 1 2 0 1 0 0 0 2 2 0 2 2 2 2 0 0 2 0 1
1 0 0 1 0 0 2 2 1 2 1 1 0 0 0 2 1 0 2 2 1 1 0 1 1 1 1 1 2 2 2 1 0 1 0
2 2 0 1 0 1 2 0 1 2 1 2 0 0 2 1 0 0 1 0 1 2 2 2 1 0 0 0 1 0 1 2 1 1 1 0 1
1 2 2 1 0 1 2 2 2 1 2 2 1 0 2 0 2 1 1 2 2 2 1 0 1 1 2 2 2 2 0 0 1 1 1 0 1
0 2 2 0 0 2 0 2 2 2 0 0 2 1 0 2 0 0 0 1 1 2 0 2 0 1 0 2 0 1 0 2
0 2 2 2 0 2 2 2 1 0 1 2 1 1 1 2 1 2 2 2 1 0 0 2 1 0 0 0 2 0 2 1 0 0 1 0
0 0 2 1 0 0 2 2 0 2 2 2 1 0 1 1 1 2 0 0 0 1 2 0 0 0 1 2 0 0 0 2 1 0 0 2 1
0 1 0 2 2 0 0 0 2 0 2 2 1 0 2 0 2 0 2 1 1 1 0 2 0 2 2 1 0 1 0 2 1 2 0 2 1
1 1 2 2 0 2 0 1 1 0 2 2 2 0 1 0 2 0 0 1 0 1 2 2 2 0 1 0 2 2 2 0 2 0 2 2 1
1 0 0 2 2 0 2 1 1 1 2 0 0 2 0 2 0 2 2 2 0 0 2 2 2 0 0 1 2 1 1 2 2 0 1
0 0 0 1 0 1 0 0 1 2 1 0 2 0 2 0 1 1 1 1 0 0 0 0 0 1 1 0 1 2 0 1 1 2 0 0 1 1 2
2 1 2 1 1 2 2 0 2 1 2 1 1 0 2 2 0 2 0 0 0 1 1 1 0 1 2 1 2 1 0 2 2 2 1 1 1
1 1 1 2 0 0 0 0 1 1 2 1 1 2 2 0 1 0 1 0 1 0 1 2 1 2 0 0 2 2 0 1 0 0 2 1
2 2 0 1 1 1 2 1 2 1 0 1 2 1 1 2 2 0 0 1 0 0 1 2 0 0 2 1 2 1 2 1 2 1 2 0 0 0 0
1 0 0 0 2 2 1 1 1 1 2 0 1 0 2 0 0 1 2 2 2 1 0 2 2 2 2 2 2 1 2 0 1 0 2
1 0 0 0 0 2 1 2 0 2 1 0 1 0 1 1 1 1 2 1 0 2 2 2 2 2 2 1 2 0 1 2 2 0 1
2 1 0 0 2 1 2 1 2 1 2 1 2 1 0 1 1 1 2 0 1 2 0 2 2 0 1 0 1 2 1 1 0
1 0 1 0 1 1 1 2 0 2 0 2 0 1 0 1 0 2 1 0 1 2 0 0 1 2 1 1 2 2 2 1 0
1 2 0 1 0 1 2 1 2 1 2 1 2 1 0 0 2 1 2 1 2 0 1 0 2 0 2 0 2 0 2 0 2 2 2 1
2 2 0 2 0 2 2 0 0 1 0 1 2 0 1 1 2 0 0 2 0 2 2 2 1 0 2 2 2 2 1 0 2 0 0 2 1
1 2 2 2 1 1 0 0 1 1 0 0 2 2 1 1 0 2 2 0 2 1 1 2 0 1 1 2 2 1 2 2 2 0 0
0 2 1 2 1 1 2 1 2 2 0 1 0 2 2 1 2 0 1 0 0 2 1 1 0 1 1 1 0 1 2 0 0
1 2 2 0 1 2 2 1 0 2 2 0 0 1 2 0 1 2 1 2 0 1 0 1 1 1 0 1 2 0 2 1 1
```



```
In [13]: import numpy as np

class KMeansCustom:
    def __init__(self, n_clusters, max_iter=300):
```

```

    self.n_clusters = n_clusters
    self.max_iter = max_iter
    self.centroids = None
    self.cluster_ids = None

    def fit(self, X, initial_centroids=None):
        m = X.shape[0]
        if initial_centroids is None:
            indices = np.random.choice(m, self.n_clusters, replace=False)
            self.centroids = X[indices]
        else:
            self.centroids = initial_centroids

        self.cluster_ids = np.zeros(m)
        cost = 0

        for _ in range(self.max_iter):
            new_cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - self.centroids, axis=1)) for i in range(m)])

            # update cluster centers
            for j in range(self.n_clusters):
                if np.any(new_cluster_ids == j):
                    self.centroids[j] = np.mean(X[new_cluster_ids == j], axis=0)

            # stop
            if np.all(new_cluster_ids == self.cluster_ids):
                for z in range(self.n_clusters):
                    cost += (1/m) * np.sum(np.linalg.norm(X[new_cluster_ids == z] - self.centroids[z], axis=1) ** 2)
                self.cluster_ids = new_cluster_ids
                return cost, self.cluster_ids, self.centroids
            else:
                self.cluster_ids = new_cluster_ids

        # If it reaches here, it means max_iter was reached without convergence
        for z in range(self.n_clusters):
            cost += (1/m) * np.sum(np.linalg.norm(X[self.cluster_ids == z] - self.centroids[z], axis=1) ** 2)
        return cost, self.cluster_ids, self.centroids

    def predict(self, X):
        return np.array([np.argmin(np.linalg.norm(x - self.centroids, axis=1)) for x in X])

```

## Example1

In [2]: `#the first two one(MinMaxScaler,StandardScaler) can scale and fit the data , they are classes.  
#the third one(scale) scale the data, that is a method.`

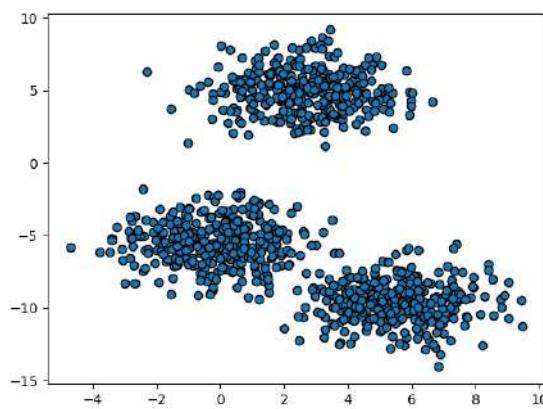
```

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import scale
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import pandas as pd

```

In [ ]:

In [3]: `X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.5, random_state=10)`  
`plt.scatter(X[:, 0], X[:, 1], edgecolor="k")`  
`plt.show()`



In [4]: `X.shape`

Dut[4]: `(1000, 2)`

In [17]: `X1_normalized= scale(X[:, 0] , axis= 0 , with_mean= True , with_std= True)`  
`X2_normalized= scale(X[:, 1] , axis= 0 , with_mean= True , with_std= True)`

In [18]: `d = {'first_value': X1_normalized, 'second_value':X2_normalized}`  
`df=pd.DataFrame(d)`  
`df`

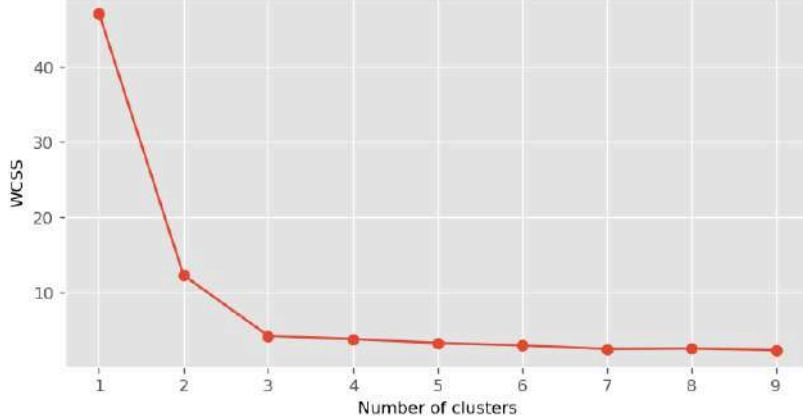
	first_value	second_value
0	0.541071	-1.070662
1	-0.173577	0.981308
2	-0.983675	-0.465678
3	-0.904900	-0.067413
4	-0.567329	1.491656
...	...	...
995	-1.446594	0.053757
996	1.100001	-0.660024
997	1.224302	1.304300
998	-0.868375	1.108034
999	-0.439211	-0.174590

1000 rows x 2 columns

In [19]: `model1= KMeansCustom(3)`  
`model1.fit(X)`

```
In [20]: costs = []
for i in range(1 , 10):
    model2=KMeansCustom(i)
    cost,_,_ =model2.fit(X)
    costs.append(cost)
```

```
In [21]: plt.figure(figsize=(8, 4))
plt.plot(range(1, 10), costs, marker='o')
plt.xticks(range(1, 10))
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



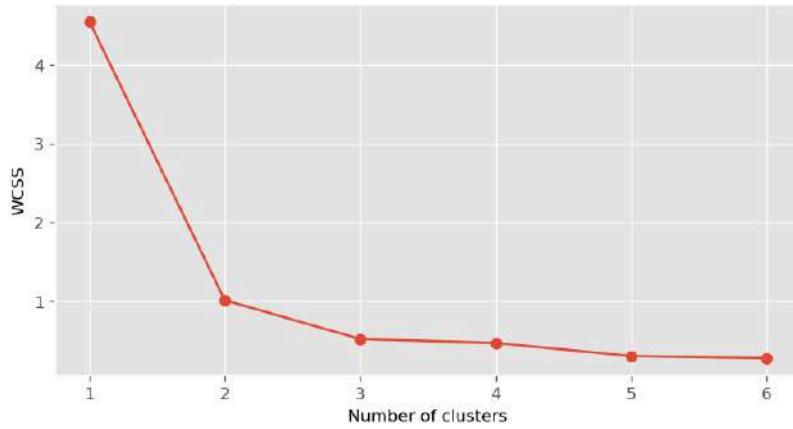
## Example2

```
In [22]: from sklearn.datasets import load_iris  
iris=load_iris()
```

```
In [23]: X,y = iris.data , iris.target
```

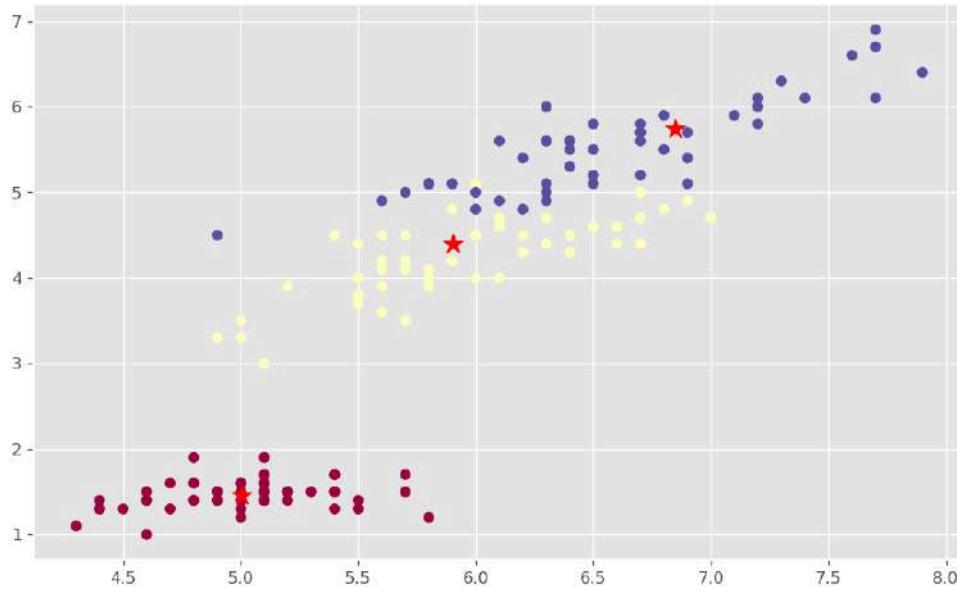
```
In [24]: costs=[]
for k in range(1, 7):
    model3=KMeansCustom(k)
    cost,_,_ =model3.fit(X)
    costs.append(cost)
```

```
In [25]: plt.figure(figsize=(8, 4))
plt.plot(range(1, 7), costs, marker='o')
plt.xticks(range(1, 7))
plt.xlabel('Number of clusters')
plt.ylabel("WCSS")
plt.show()
```



```
In [26]: model3 = KMeansCustom(3)
cost, cluster_ids, centroids = model3.fit(X)
```

```
In [27]: plt.scatter(iris.data[:, 0], iris.data[:, 2], c=y)
plt.scatter(centroids[:, 0], centroids[:, 2], marker="*", c="red", s=150)
plt.show()
```



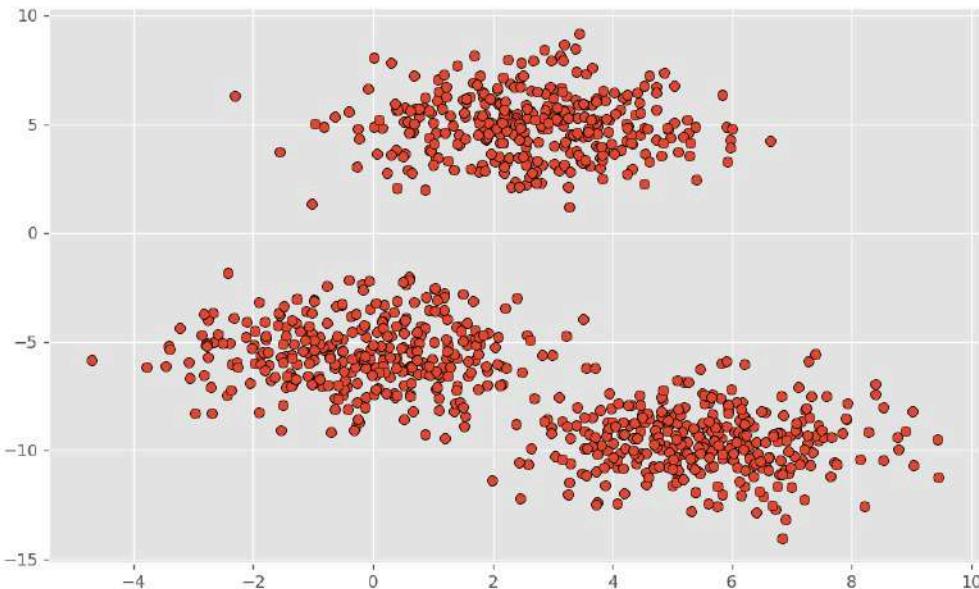
## Sklearn

```
In [28]: from sklearn.cluster import KMeans
import pandas as pd
```

```
from sklearn.preprocessing import scale
import matplotlib.pyplot as plt
```

```
In [29]: X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.5, random_state=10)
```

```
plt.scatter(X[:, 0], X[:, 1], edgecolor="k")
plt.show()
```



```
In [30]: X1_normalized= scale(X[:, 0] , axis= 0 , with_mean= True , with_std= True)
X2_normalized= scale(X[:, 1] , axis= 0 , with_mean= True , with_std= True)
```

```
In [31]: d = {"first_value": X1_normalized, "second_value":X2_normalized}
df=pd.DataFrame(d)
df
```

```
Out[31]: first_value    second_value
```

0	0.541071	-1.070662
1	-0.173577	0.981308
2	-0.983675	-0.465678
3	-0.904900	-0.067413
4	-0.567329	1.491656
...	...	...
995	-1.446594	0.053757
996	1.100001	-0.660024
997	1.224302	1.304300
998	-0.868375	1.108034
999	-0.439211	-0.174590

1000 rows × 2 columns

```
In [32]: km= KMeans(n_init="auto" )
km.fit(df[["first_value","second_value"]])
km.inertia_
```

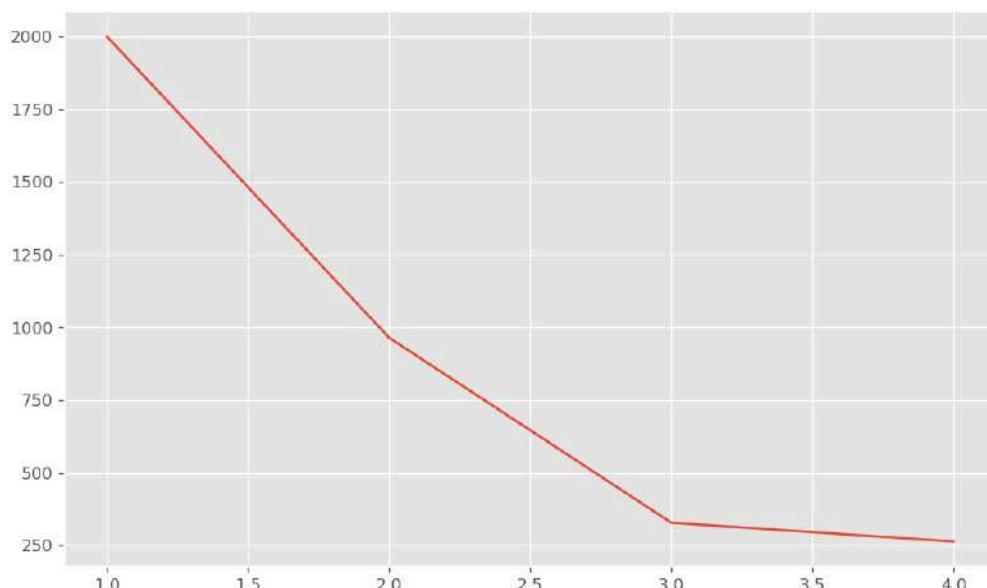
```
Out[32]: 116.56796234014519
```

```
In [33]: wcss = []
for i in range(1,5):
    km= KMeans(n_clusters= i )
    km.fit(df[["first_value","second_value"]])
    wcss.append(km.inertia_)

wcss
```

```
Out[33]: [1999.999999999995, 964.9482561631145, 329.0090988040914, 264.24847175654315]
```

```
In [34]: plt.plot(range(1, 5),wcss)
plt.show()
```



```
In [35]: km= KMeans(n_clusters=3,n_init='auto')
y_predict=km.fit_predict (df[["first_value","second_value"]])

df[ "predict"] = y_predict
df
```

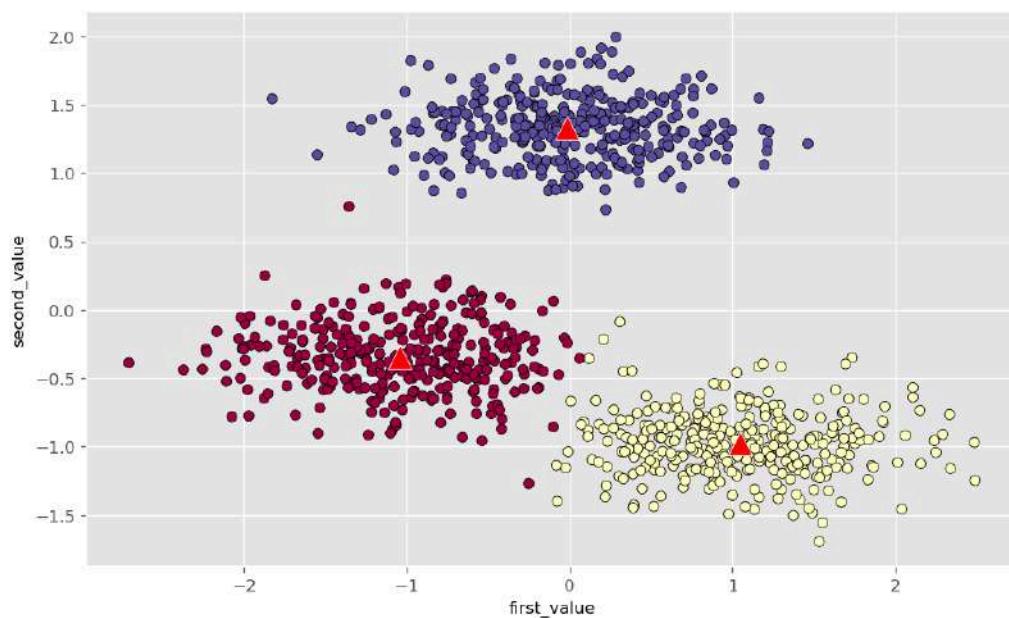
Dut[35]:	first_value	second_value	predict
<b>0</b>	0.541071	-1.070662	1
<b>1</b>	-0.173577	0.981308	2
<b>2</b>	-0.983675	-0.465678	0
<b>3</b>	-0.904900	-0.067413	0
<b>4</b>	-0.567329	1.491656	2
...	...	...	...
<b>995</b>	-1.446594	0.053757	0
<b>996</b>	1.100001	-0.660024	1
<b>997</b>	1.224302	1.304300	2
<b>998</b>	-0.868375	1.108034	2
<b>999</b>	-0.439211	-0.174590	0

1000 rows × 3 columns

```
In [36]: km.cluster_centers_
Out[36]: array([[-0.04184561, -0.34361576],
               [ 1.04554222, -0.97308184],
               [-0.0162935 ,  1.32842067]])
```

```
In [37]: plt.scatter(df["first_value"], df["second_value"], c=df["predict"], edgecolor="k")
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], marker = "x", s=200, c="red", edgecolors="white")
plt.xlabel("first_value")
plt.ylabel("second_value")
```

```
Out[37]: Text(0, 0.5, 'second_value')
```

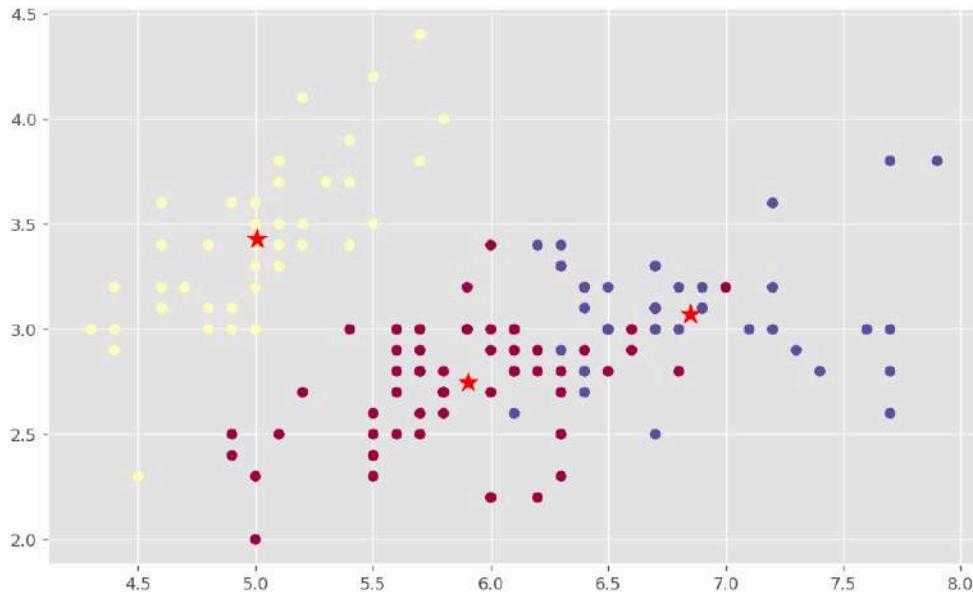


```
In [38]: from sklearn.datasets import load_iris
```

```
In [39]: iris=load_iris()
```

```
In [40]: kmn=KMeans(n_clusters=3)
kmn.fit(iris.data)
labels=kmn.predict(iris.data)
```

```
In [41]: plt.scatter(iris.data[ :, 0 ] , iris.data[ :, 1 ] , c= labels)
plt.scatter(centroids[ :, 0 ] , centroids[ :, 1 ] , marker= "*" , c= "red" , s=150)
plt.show()
```



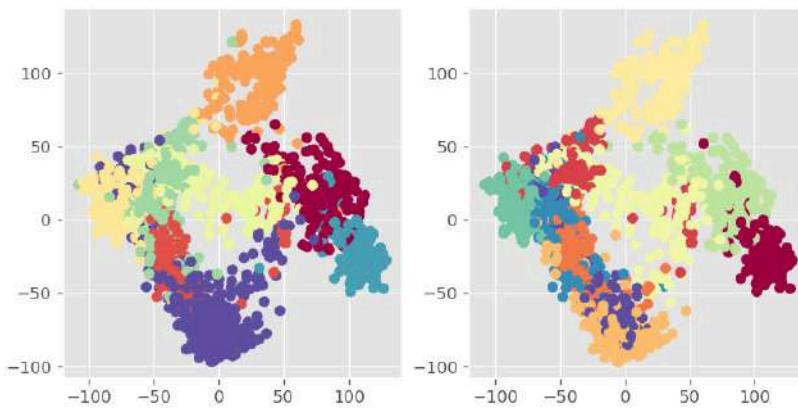
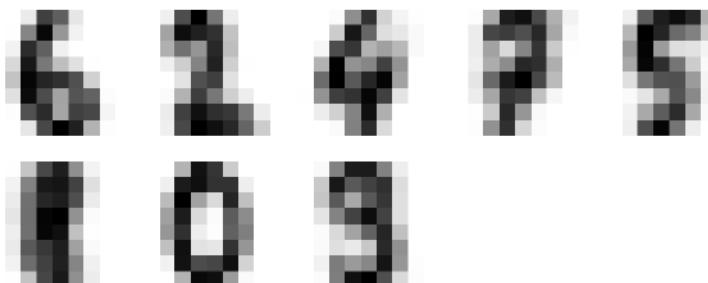
```
In [42]: # Load digits dataset
from sklearn.datasets import load_digits
digits = load_digits()

# cluster digits to 10 clusters
kmeans = KMeans(n_init=3)
cluster_ids = kmeans.fit_predict(digits.data)
print(kmeans.cluster_centers_.shape)
```

```
In [43]: # visualize the cluster centers
fig = plt.figure(figsize=(8, 3))
for i in range(8):
    ax = fig.add_subplot(2, 5, 1 + i)
    ax.imshow(kmeans.cluster_centers_[i].reshape((8, 8)), cmap=plt.cm.binary)
    ax.grid(False)
    ax.axis('off')

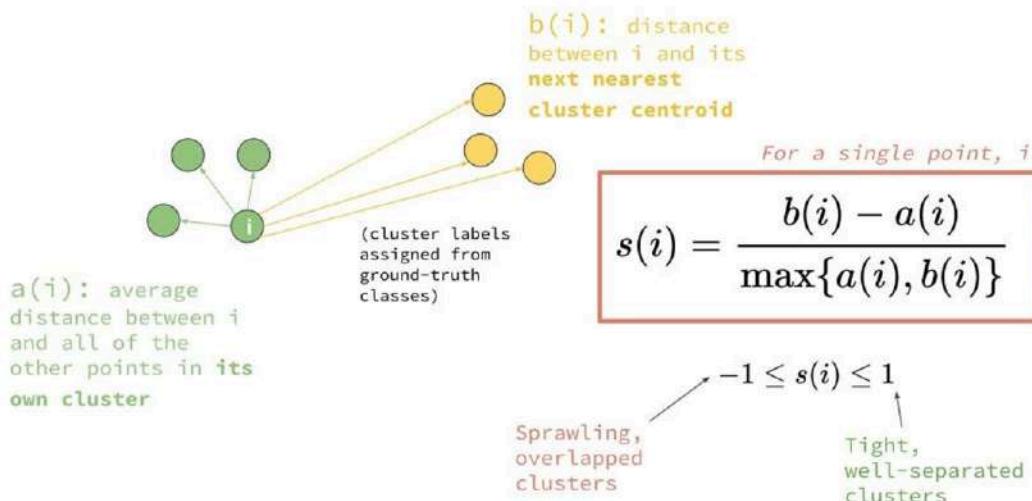
# visualize the projected data
from sklearn.manifold import Isomap
X_iso = Isomap(n_neighbors=10).fit_transform(digits.data)

fig, ax = plt.subplots(1, 2, figsize=(8, 4))
ax[0].scatter(X_iso[:, 0], X_iso[:, 1], c=cluster_ids)
ax[1].scatter(X_iso[:, 0], X_iso[:, 1], c=digits.target);
```



## K - MEANS

### Cluster accuracy



```
sklearn.metrics.silhouette_score(X, labels, *, metric='euclidean', sample_size=None, random_state=None,)
```

```
In [45]: import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.cluster._kmeans")

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

X, _ = make_blobs(random_state=42)
kmeans = KMeans(n_clusters=2)
labels=kmeans.fit_predict(X)
silhouette_score(X, labels, metric="euclidean", sample_size=X.shape[0])
```

Out[45]: 0.7020937832636734

### Example

```
In [46]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

In [47]: df=pd.read_csv(r"D:\AI_Machinelearning\machine learning\jozavat\kmeans\facebook.csv")
df.drop(['status_id', "status_published"] , axis=1 , inplace=True)
df
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	video	529	512	262	432	92	3	1	1	0
1	photo	150	0	0	150	0	0	0	0	0
2	video	227	236	57	204	21	1	1	0	0
3	photo	111	0	0	111	0	0	0	0	0
4	photo	213	0	0	204	9	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
7045	photo	89	0	0	89	0	0	0	0	0
7046	photo	16	0	0	14	1	0	1	0	0
7047	photo	2	0	0	1	1	0	0	0	0
7048	photo	351	12	22	349	2	0	0	0	0
7049	photo	17	0	0	17	0	0	0	0	0

7050 rows × 10 columns

```
In [48]: df["status_type"].unique()
array(['video', 'photo', 'link', 'status'], dtype=object)

In [49]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   status_type    7050 non-null   object  
 1   num_reactions  7050 non-null   int64  
 2   num_comments   7050 non-null   int64  
 3   num_shares     7050 non-null   int64  
 4   num_likes      7050 non-null   int64  
 5   num_loves      7050 non-null   int64  
 6   num_wows       7050 non-null   int64  
 7   num_hahas      7050 non-null   int64  
 8   num_sads       7050 non-null   int64  
 9   num_angrys     7050 non-null   int64  
dtypes: int64(9), object(1)
memory usage: 550.9+ KB
```

In [50]: `df.describe().T.style.background_gradient()`

	count	mean	std	min	25%	50%	75%	max
<code>num_reactions</code>	7050.000000	230.117163	462.625309	0.000000	17.000000	59.500000	219.000000	4710.000000
<code>num_comments</code>	7050.000000	224.356028	889.636820	0.000000	0.000000	4.000000	23.000000	20990.000000
<code>num_shares</code>	7050.000000	40.022553	131.599965	0.000000	0.000000	0.000000	4.000000	3424.000000
<code>num_likes</code>	7050.000000	215.043121	449.472357	0.000000	17.000000	58.000000	184.750000	4710.000000
<code>num_loves</code>	7050.000000	12.728652	39.972930	0.000000	0.000000	0.000000	3.000000	657.000000
<code>num_wows</code>	7050.000000	1.289362	8.719650	0.000000	0.000000	0.000000	0.000000	278.000000
<code>num_hahas</code>	7050.000000	0.696454	3.957183	0.000000	0.000000	0.000000	0.000000	157.000000
<code>num_sads</code>	7050.000000	0.243688	1.597156	0.000000	0.000000	0.000000	0.000000	51.000000
<code>num_angrys</code>	7050.000000	0.113191	0.726812	0.000000	0.000000	0.000000	0.000000	31.000000

In [51]: `X = df`

```
In [52]: from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X['status_type'] = le.fit_transform(X['status_type'])

"""
another method

df.replace({"video": 1, "photo": 0, ...}, inplace=True)
df

"""

label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print("Label Mapping:", label_mapping)

df
```

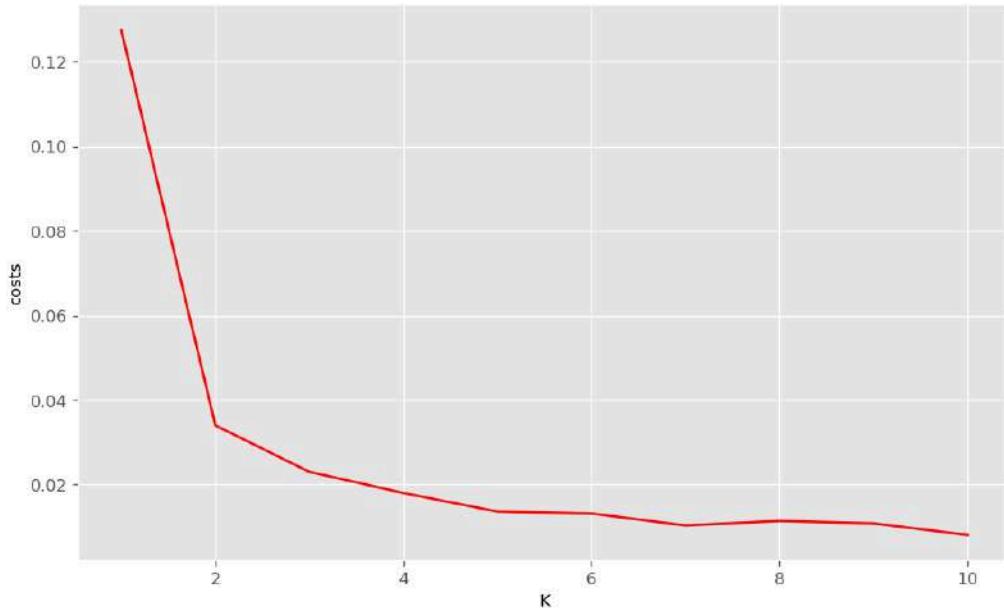
	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
<b>0</b>	3	529	512	262	432	92	3	1	1	0
<b>1</b>	1	150	0	0	150	0	0	0	0	0
<b>2</b>	3	227	236	57	204	21	1	1	0	0
<b>3</b>	1	111	0	0	111	0	0	0	0	0
<b>4</b>	1	213	0	0	204	9	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
<b>7045</b>	1	89	0	0	89	0	0	0	0	0
<b>7046</b>	1	16	0	0	14	1	0	1	0	0
<b>7047</b>	1	2	0	0	1	1	0	0	0	0
<b>7048</b>	1	351	12	22	349	2	0	0	0	0
<b>7049</b>	1	17	0	0	17	0	0	0	0	0

7050 rows × 10 columns

```
In [53]: from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
X = ms.fit_transform(X)
```

```
In [54]: costs=[]
for i in range(1,11):
    model1= KMeansCustom(i)
    cost, _, _ = model1.fit(X)
    costs.append(cost)

plt.plot([i for i in range(1,11)] , costs , c="red")
plt.xlabel("K")
plt.ylabel("costs")
plt.show()
```



```
In [75]: model1 = KMeansCustom(4)
_, _, _ = model1.fit(X)

from sklearn.metrics import silhouette_score
silhouette_score(X, labels, metric="12")
0.775850346329405

Out[75]:
```

```
In [76]: df["labels"] = labels
df
```

```
Out[76]:   status_type  num_reactions  num_comments  num_shares  num_likes  num_loves  num_wows  num_hahas  num_sads  num_angrys  labels
0               3        529           512         262       432        92        3        1        1        0        2
1               1        150            0           0       150        0        0        0        0        0        0
2               3        227           236          57       204       21        1        1        0        0        2
3               1        111            0           0       111        0        0        0        0        0        0
4               1        213            0           0       204        9        0        0        0        0        0
...
7045              1        89            0           0        89        0        0        0        0        0        0
7046              1        16            0           0       14        1        0        1        0        0        0
7047              1         2            0           0        1        1        0        0        0        0        0
7048              1        351           12          22       349        2        0        0        0        0        0
7049              1        17            0           0       17        0        0        0        0        0        0
7050 rows × 11 columns
```

```
In [77]: def print_status_for_clusters(df, labels, status_type):
    unique_labels = df[labels].unique()
    for label in unique_labels:
        status_types = df[df[labels] == label][status_type].unique()
        print(f"Cluster {label} contains status types: {status_types}")

# Print status types for each cluster label
print_status_for_clusters(df, 'labels', 'status_type')

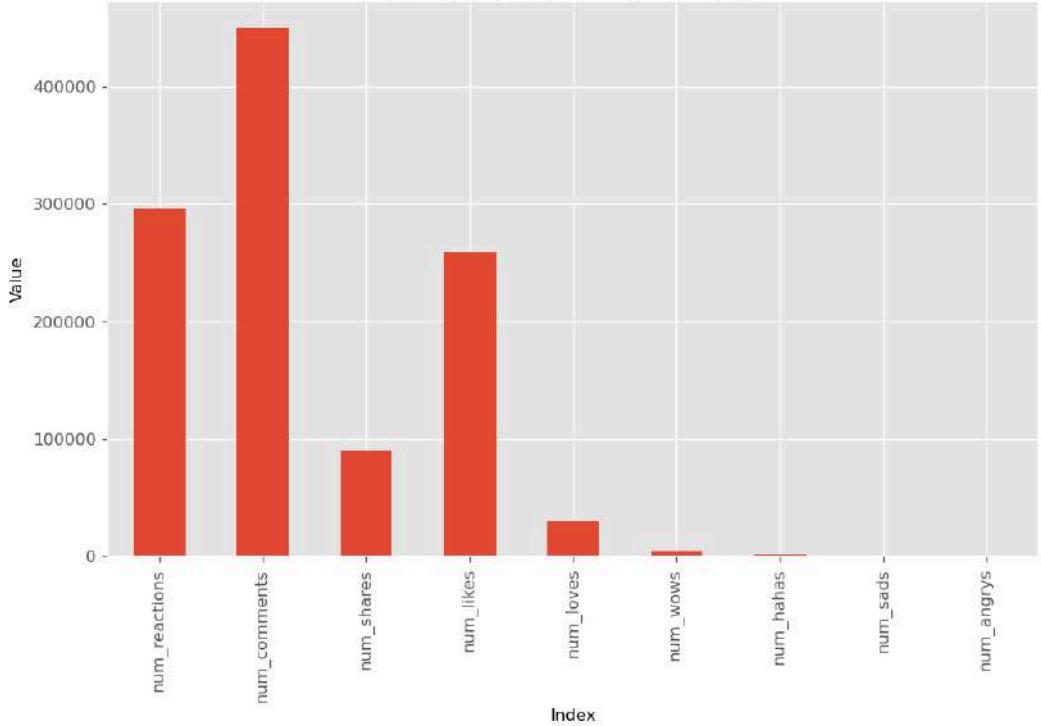
Cluster 2 contains status types: [3 2]
Cluster 0 contains status types: [1 0]
Cluster 1 contains status types: [3]
Cluster 3 contains status types: [1 2 0]
```

```
In [78]: df_label_2 = df[df['labels'] == 1]

df_features_label_2 = df_label_2.drop(columns=['labels', 'status_type']).sum()

# Plot the bar chart
df_features_label_2.plot(kind='bar', figsize=(10, 6))
plt.title('Bar Plot of Features with Label 2')
plt.xlabel('Index')
plt.ylabel('Value')
plt.show()
```

### Bar Plot of Features with Label 2

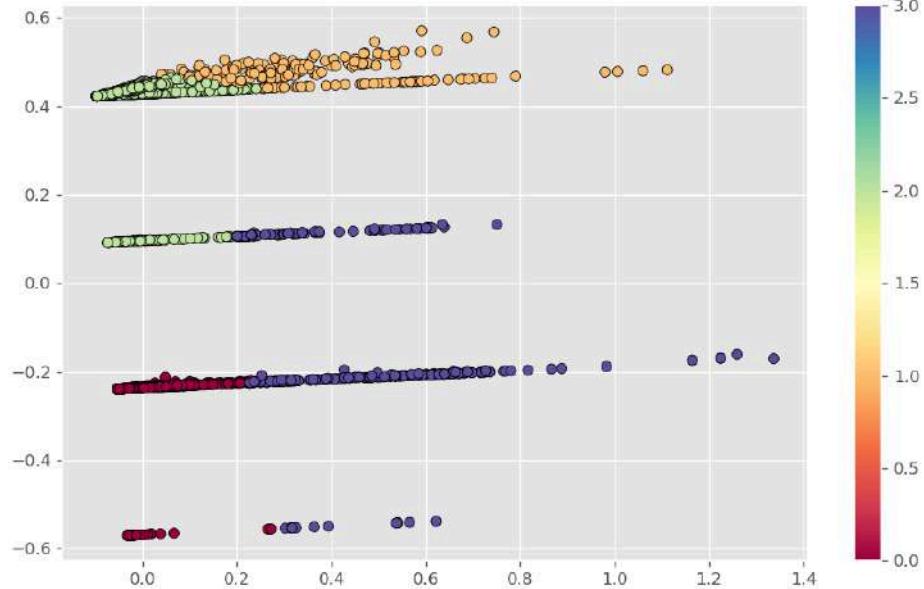


```
In [79]: from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_proj = pca.fit_transform(X)
```

```
X_proj.shape
# plot projected data

plt.scatter(X_proj[:, 1], X_proj[:, 0],
            c=labels, edgecolor='k', alpha=1)

plt.colorbar()
plt.show()
```



```
In [80]: #using sklearn
from sklearn.cluster import KMeans
model2 = KMeans(n_clusters=2, n_init=10)

cluster_ids = model2.fit_predict(X)
centers = model2.cluster_centers_

#accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X, cluster_ids, metric="l2")
```

```
Out[80]: 0.764095933166648
```

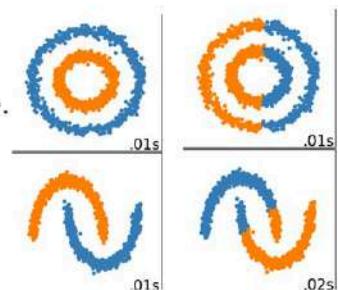
## K - MEANS

### Weakness

- Often terminates at local optimum



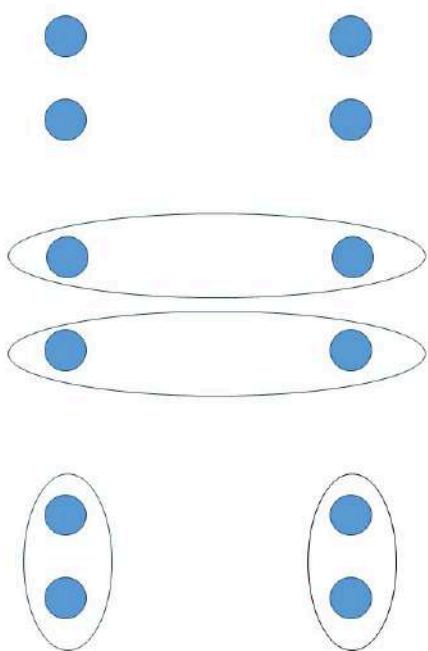
- Applicable to only objects in a continuous n-dimensional space.
- Sensitive to noisy data and outliers.
- Sensitive to initial clusters.
- Not suitable to discover clusters with non-convex shape



## kmeans++

## K - MEANS

### Kmeans ++

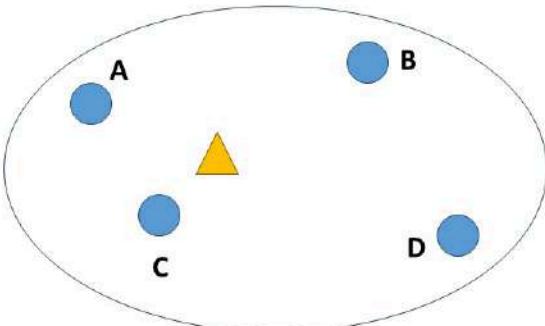


- Choose one center uniformly at random among the data points.
- For each data point  $x$  not chosen yet, compute  $D(x)$ , the distance between  $x$  and the nearest center that has already been chosen.
- Choose one new data point at random as a new center, using a weighted probability distribution where a point  $x$  is chosen with probability proportional to  $D(x)^2$ .
- Repeat Steps 2 and 3 until  $k$  centers have been chosen.
- Now that the initial centers have been chosen, proceed using standard  $k$ -means clustering

new method almost always performed at least as well as vanilla  $k$ -means in both speed and error.

## K - MEANS

### Kmeans ++



	dist	dist <sup>2</sup>	p
A	2	4	$\frac{4}{30} = 0.13$
B	3	9	$\frac{9}{30} = 0.30$
C	1	1	$\frac{1}{30} = 0.03$
D	4	16	$\frac{16}{30} = 0.54$
		30	1

```
In [82]: import numpy as np

class KMeansPlusPlus:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.centroids = None
        self.cluster_ids = None
    def _initialize_centroids(self, X):
        m = X.shape[0]
        self.centroids = np.zeros((self.n_clusters, X.shape[1]))

        # Initialize the first centroid randomly
        self.centroids[0] = X[np.random.choice(m)]

        for i in range(1, self.n_clusters):
            # Compute the distance from each point to the closest centroid
            distances = np.min(np.linalg.norm(X[:, np.newaxis] - self.centroids[:, :i], axis=2), axis=1)
            # Square the distances
            distances_squared = distances ** 2
            # Choose the next centroid with probability proportional to the distance squared
            probabilities = distances_squared / np.sum(distances_squared)
            chosen_index = np.random.choice(m, p=probabilities)
            self.centroids[i] = X[chosen_index]

    def fit(self, X, initial_centroids=None):
        m = X.shape[0]
        if initial_centroids is None:
            self._initialize_centroids(X)
        else:
            self.centroids = initial_centroids

        self.cluster_ids = np.zeros(m)
        cost = 0
        iter_count = 0

        while True:
            new_cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - self.centroids, axis=1)) for i in range(m)])

            # Update cluster centers
            for j in range(self.n_clusters):
                if np.any(new_cluster_ids == j):
                    self.centroids[j] = np.mean(X[new_cluster_ids == j], axis=0)

            # Stop if convergence or max_iter reached
            if np.all(new_cluster_ids == self.cluster_ids) or iter_count >= self.max_iter:
                cost += (1/m) * np.sum(np.linalg.norm(X[new_cluster_ids == z] - self.centroids[z], axis=1) ** 2)
                self.cluster_ids = new_cluster_ids
                return cost, self.cluster_ids, self.centroids
            else:
                self.cluster_ids = new_cluster_ids

            iter_count += 1

            # If it reaches here, it means max_iter was reached without convergence
            for z in range(self.n_clusters):
                cost += (1/m) * np.sum(np.linalg.norm(X[self.cluster_ids == z] - self.centroids[z], axis=1) ** 2)
            return cost, self.cluster_ids, self.centroids

    def predict(self, X):
        return np.array([np.argmin(np.linalg.norm(x - self.centroids, axis=1)) for x in X])
```

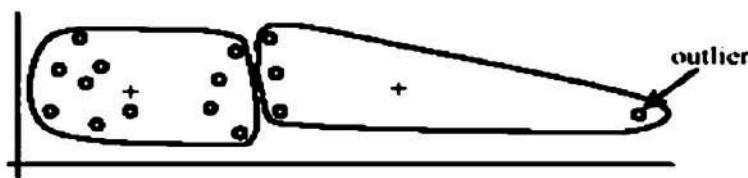
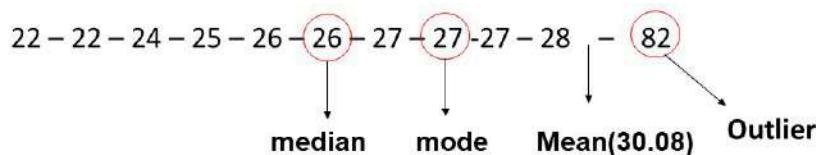
```
In [83]: model3=KMeansPlusPlus(2)
_, labels , _=model3.fit(X)
```

```
#accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X , labels , metric="12")
Out[83]:
```

## Kmedoids

### K - MEANS

#### k-medoids



(A): Undesirable clusters



(B): Ideal clusters

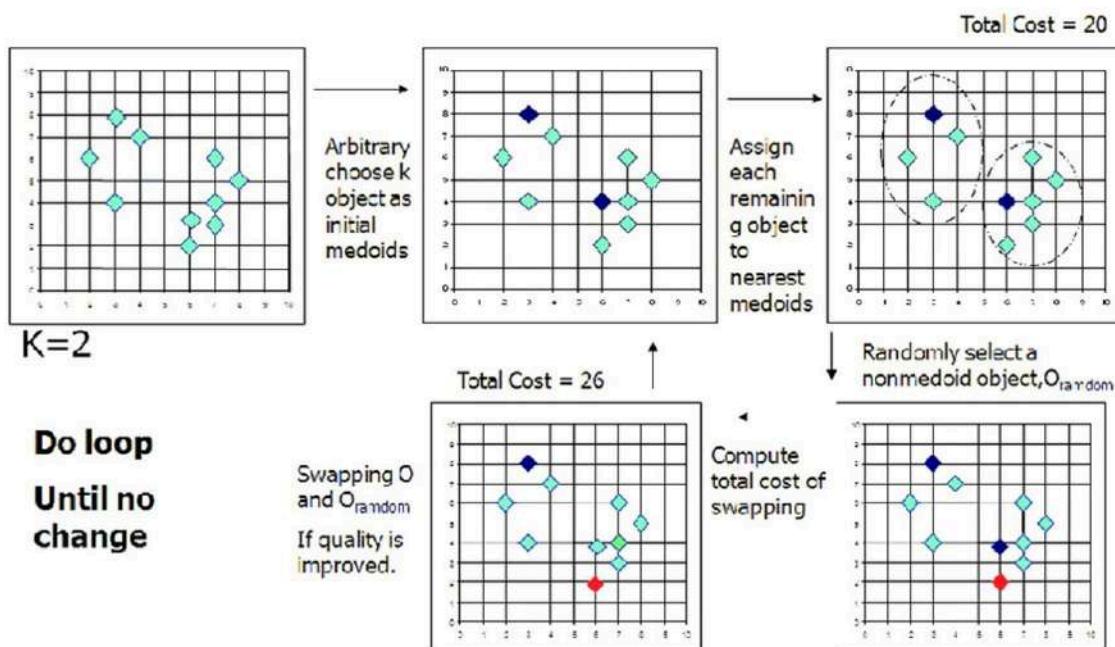
## K - MEANS

### PAM (Partitioning Around Medoids)

- 1.(BUILD) Initialize: select  $k$  of the  $n$  data points as the medoids to minimize the cost
- 2.Associate each data point to the closest medoid.
- 3.(SWAP) While the cost of the configuration decreases:
  - 1.For each medoid  $m$ , and for each non-medoid data point  $o$ :
    - 1.Consider the swap of  $m$  and  $o$ , and compute the cost change
    - 2.If the cost change is the current best, remember this  $m$  and  $o$  combination
  - 2.Perform the best swap of  $m_{best}$  and  $o_{best}$ , if it decreases the cost function. Otherwise, the algorithm terminates.

## K - MEANS

### A Typical K-Medoids Algorithm (PAM)



```
In [85]: import numpy as np

class KMedoids:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
```

```

self.medoids = None
self.cluster_ids = None

def _initialize_medoids(self, X):
    m = X.shape[0]
    indices = np.random.choice(m, self.n_clusters, replace=False)
    self.medoids = X[indices]

def fit(self, X):
    m = X.shape[0]
    self._initialize_medoids(X)

    self.cluster_ids = np.zeros(m, dtype=int)
    cost = 0
    iter_count = 0

    while True:
        # Assign each point to the nearest medoid
        new_cluster_ids = np.array([np.argmin([np.linalg.norm(X[i] - medoid) for medoid in self.medoids]) for i in range(m)])

        # Update medoids
        new_medoids = np.copy(self.medoids)
        for j in range(self.n_clusters):
            cluster_points = X[new_cluster_ids == j]
            if len(cluster_points) > 0:
                medoid_costs = np.sum(np.linalg.norm(cluster_points[:, np.newaxis] - cluster_points, axis=2), axis=0)
                new_medoids[j] = cluster_points[np.argmin(medoid_costs)]

        # Stop if convergence or max_iter reached
        if np.array_equal(new_medoids, self.medoids) or iter_count >= self.max_iter:
            cost = np.sum([np.sum(np.linalg.norm(X[new_cluster_ids == j] - self.medoids[j], axis=1)) for j in range(self.n_clusters)])
            self.cluster_ids = new_cluster_ids
            self.medoids = new_medoids
            return cost, self.cluster_ids, self.medoids
        else:
            self.cluster_ids = new_cluster_ids
            self.medoids = new_medoids

        iter_count += 1

    # If it reaches here, it means max_iter was reached without convergence
    cost = np.sum([np.sum(np.linalg.norm(X[self.cluster_ids == j] - self.medoids[j], axis=1)) for j in range(self.n_clusters)])
    return cost, self.cluster_ids, self.medoids

def predict(self, X):
    return np.array([np.argmin([np.linalg.norm(x - medoid) for medoid in self.medoids]) for x in X])

```

```
In [87]: model4=KMedoids(2)
_, labelss, _ =model4.fit(X)

#accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X , labelss , metric="12")
```

```
Out[87]: 0.7608485051559389
```

```
!pip install scikit-learn-extra
classsklearn_extra.cluster.KMedoids(n_clusters=8, metric='euclidean', method='pam', max_iter=300, random_state=None)
```

```
In [88]: from sklearn_extra.cluster import KMedoids
model4= KMedoids(n_clusters=4 , method='pam')
labelss4=model4.fit_predict(X)
```

```
In [89]: #accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X , labelss4 , metric="12")
```

```
Out[89]: 0.7388213514786055
```

## K - MEANS

### ■ Hamming Distance

A	3	2	7	4	9	5	8	1
Hamming Distance(A, B) = 4								
B	1	2	7	3	9	5	1	8
Hamming Distance(A, B) = 2								
A	"Geeks"	"for"	"Geeks"	"Java"	"C++"	"Swift"	"R"	"Python"
B	"Geeks"	"for"	"Geeks"	"C++"	"C"	"Swift"	"R"	"Python"

### ■ Mode

You can have more than one mode

1, 3, 3, 3, 5, 6, 6, 9, 9, 9

There are two modes

3 9



## K - MEANS

### ■ kmodes (Categorical data)

How KModes Algorithm Work

- Pick K observations at random and use them as leaders/clusters
- Calculate the dissimilarities and assign each observation to its closest cluster
- Define new modes for the clusters
- Repeat 2–3 steps until there are no re-assignment required

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

## K - MEANS

clusters(K)=3

**Step 1:** Pick K observations at random and use them as leaders/clusters

Leaders			
person	hair color	eye color	skin color
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

## K - MEANS

**Step 2:** Calculate the dissimilarities(no. of mismatches) and assign each observation to its closest cluster

Comparing leader/Cluster P1 to the observation P1 gives 0 dissimilarities.

Comparing leader/cluster P1 to the observation P2 gives 3(1+1+1) dissimilarities.

Leaders			
person	hair color	eye color	skin color
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

## K - MEANS

calculate all the dissimilarities and put them in a matrix as shown below and assign the observations to their closest cluster(cluster that has the least dissimilarity)

Dissimilarity matrix

	Cluster 1 (P1)	Cluster 2 (P7)	Cluster 3 (P8)	Cluster
P1	0 ✓	2	2	Cluster 1
P2	3 ✓	3	3	Cluster 1
P3	3	1 ✓	3	Cluster 2
P4	3	3	1 ✓	Cluster 3
P5	1 ✓	2	2	Cluster 1
P6	3	3	2 ✓	Cluster 3
P7	2	0 ✓	2	Cluster 2
P8	2	2	0 ✓	Cluster 3

After step 2, the observations P1, P2, P5 are assigned to cluster 1; P3, P7 are assigned to Cluster 2; and P4, P6, P8 are assigned to cluster 3.

## K - MEANS

### Step 3: Define new modes for the clusters

Mode is simply the most observed value.

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

$$j = \begin{cases} 1 & \text{if } x^{(i)} = \mu_k \\ 0 & \text{if } x^{(i)} \neq \mu_k \end{cases}$$

Cluster 1 observations(P1, P2, P5) has brunette as the most observed hair color, amber as the most observed eye color, and fair as the most observed skin color.

new leaders after the update.

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

## K - MEANS

Repeat steps 2–4

Comparing Cluster 1 to the observation P1 gives 1 dissimilarity.

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

After obtaining the new leaders, again calculate the dissimilarities between the observations and the newly obtained leaders.

## K - MEANS

Repeat steps 2–4

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing Cluster 1 to the observation P2 gives 2 dissimilarities.

## K - MEANS

	Cluster 1	Cluster 2	Cluster 3	Cluster
P1	1 ✓	2	3	Cluster 1
P2	2 ✓	3	2	Cluster 1
P3	3	1 ✓	2	Cluster 2
P4	3	3	0 ✓	Cluster 3
P5	0 ✓	2	3	Cluster 1
P6	3	3	1 ✓	Cluster 3
P7	2	0 ✓	3	Cluster 2
P8	2	2	1 ✓	Cluster 3

The observations P1, P2, P5 are assigned to Cluster 1; P3, P7 are assigned to Cluster 2; and P4, P6, P8 are assigned to Cluster 3.

## kmodes (Categorical data)

How KModes Algorithm Work

Pick K observations at random and use them as leaders/clusters Calculate the dissimilarities and assign each observation to its closest cluster Define new modes for the clusters Repeat 2-3 steps until there are no re-assignment required

```
In [91]: import numpy as np
import pandas as pd

def kmodescustom(X, initial_centroids):
    m, _ = X.shape
    centroids = initial_centroids
    K, _ = initial_centroids.shape
    initial_cluster_ids = np.zeros((m,))
    cost = 0

    while True:
        # Calculate the Hamming distance and assign clusters
        cluster_ids = np.array([np.argmin([np.sum(X[i] != centroid) for centroid in centroids]) for i in range(m)])

        # Update cluster centers with mode
        for j in range(K):
            if np.any(cluster_ids == j): # Check if there are any points assigned to cluster j
                centroids[j] = pd.DataFrame(X[cluster_ids == j]).mode().iloc[0].values

        # Stop if cluster assignments do not change
        if np.all(cluster_ids == initial_cluster_ids):
            for z in range(K):
                if np.any(cluster_ids == z): # Check if there are any points assigned to cluster z
                    cost += np.sum(np.sum(X[cluster_ids == z] != centroids[z], axis=1))
            return cost, cluster_ids, centroids
        else:
            initial_cluster_ids = cluster_ids
```

```
In [92]: # Example usage
X = np.array([
    ['green', 'M'],
    ['red', 'L'],
    ['blue', 'S'],
    ['green', 'L'],
    ['red', 'M'],
    ['blue', 'M']
])
initial_centroids = np.array([
    ['green', 'L'],
    ['red', 'M']
])

cost, cluster_ids, centroids = kmodescustom(X, initial_centroids)
print("Cost:", cost)
print("Cluster IDs:", cluster_ids)
print("Centroids:", centroids)
```

```
Cost: 4
Cluster IDs: [0 0 1 0 1 1]
Centroids: [['green' 'L']
 ['blue' 'M']]
```

```
In [ ]: !pip install kmodes
```

```
In [93]: import pandas as pd
import numpy as np
```

```

# Create toy dataset
hair_color = np.array(['blonde', 'brunette', 'red', 'black', 'black', 'black', 'red', 'black'])
eye_color = np.array(['amber', 'gray', 'green', 'hazel', 'amber', 'gray', 'green', 'hazel'])
skin_color = np.array(['brown', 'brown', 'brown', 'brown', 'fair', 'brown', 'fair', 'fair'])
person = ['P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8']
data = pd.DataFrame({ 'person':person, 'hair_color':hair_color, 'eye_color':eye_color, 'skin_color':skin_color})
data = data.set_index('person')
print(data)

  hair_color eye_color skin_color
person
P1      blonde    amber     fair
P2    brunette    gray    brown
P3        red    green    brown
P4       black   hazel    brown
P5  brunette    amber     fair
P6       black    gray    brown
P7       red    green     fair
P8       black   hazel     fair

```

```

In [99]: # Elbow curve to find optimal K
from kmodes.kmodes import KModes
cost = []
K = range(1,5)
for num_clusters in list(K):
    kmode = KModes(n_clusters=num_clusters, init = "random", n_init = 5, verbose=1)
    kmode.fit_predict(data)
    cost.append(kmode.cost_)

plt.plot(K, cost, 'bx-')
plt.xlabel('No. of clusters')
plt.ylabel('Cost')
plt.title('Elbow Method For Optimal k')
plt.show()

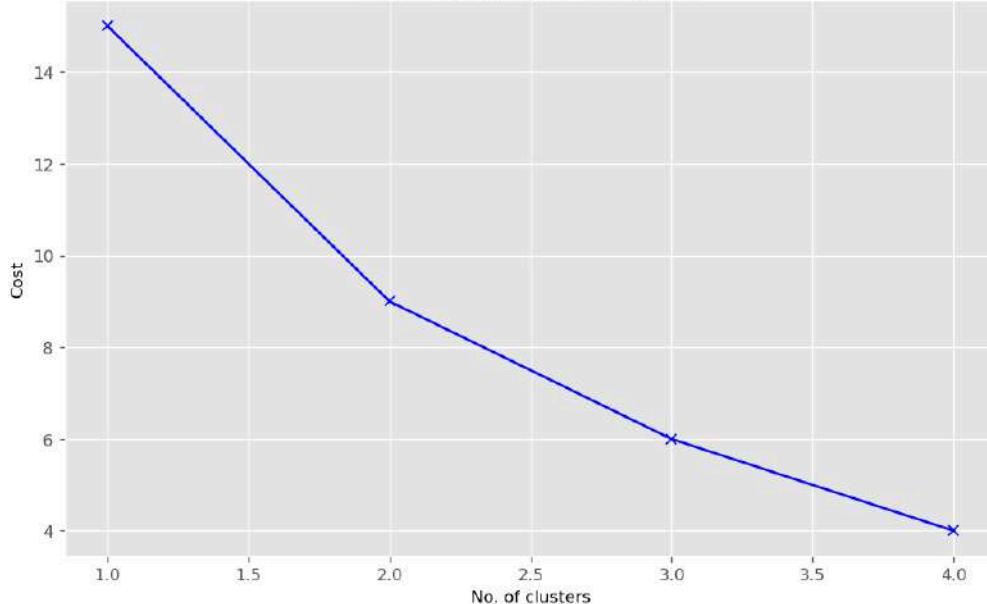
```

```

Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 15.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 15.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 15.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 15.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 15.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 9.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 9.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 9.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 1, cost: 9.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 1, cost: 9.0
Run 5, iteration: 2/100, moves: 1, cost: 9.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 2, cost: 6.0
Run 1, iteration: 2/100, moves: 0, cost: 6.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 1, cost: 10.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 2, cost: 8.0
Run 3, iteration: 2/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 6.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 6.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 2, cost: 6.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 6.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 1, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 4.0
Best run was number 1

```

## Elbow Method For Optimal k



1 - age (numeric) 2 - job : type of job (categorical: "admin.", "blue-collar", "entrepreneur", "housemaid", "management", "retired", "self-employed", "services", "student", "technician", "unemployed", "unknown") 3 - marital : marital status (categorical: "divorced", "married", "single", "unknown"; note: "divorced" means divorced or widowed) 4 - education (categorical: "basic.4y", "basic.6y", "high.school", "illiterate", "professional.course", "university.degree", "unknown") 5 - default: has credit in default? (categorical: "no", "yes", "unknown") 6 - housing: has housing loan? (categorical: "no", "yes", "unknown") 7 - loan: has personal loan? (categorical: "no", "yes", "unknown") # related with the last contact of the current campaign: 8 - contact: contact communication type (categorical: "cellular", "telephone") 9 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec") 10 - day\_of\_week: last contact day of the week (categorical: "mon", "tue", "wed", "thu", "fri") 11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y="no"). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model. # other attributes: 12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact) 13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted) 14 - previous: number of contacts performed before this campaign and for this client (numeric) 15 - poutcome: outcome of the previous marketing campaign (categorical: "failure", "nonexistent", "success") # social and economic context attributes 16 - emp.var.rate: employment variation rate - quarterly indicator (numeric) 17 - cons.price.idx: consumer price index - monthly indicator (numeric) 18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric) 19 - euribor3m: euribor 3 month rate - daily indicator (numeric) 20 - nr.employed: number of employees - quarterly indicator (numeric) Output variable (desired target): 21 - y - has the client subscribed a term deposit? (binary: "yes", "no")

```
In [102]: bank = pd.read_csv(r"D:\AI_Machinelearning\machine learning\jozavat\kmeans\bank.csv", delimiter=";")
```

```
Out[102]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul	329	5	-1	0	unknown	no
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may	153	1	-1	0	unknown	no
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug	151	11	-1	0	unknown	no
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb	129	4	211	3	other	no
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr	345	2	249	7	other	no

4521 rows x 17 columns

```
In [103]: bank.columns
```

```
Out[103]: Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

```
In [104]: bank.isnull().sum().sum()
```

```
Out[104]: 0
```

```
In [105]: from sklearn import preprocessing
le = preprocessing.LabelEncoder()
bank = bank.apply(le.fit_transform)
bank.head()
```

```
Out[105]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	11	10	1	0	0	1475	0	0	0	18	10	75	0	0	0	3	0
1	14	7	1	1	0	2030	1	1	0	10	8	216	0	228	4	0	0
2	16	4	2	2	0	1303	1	0	0	15	0	181	0	219	1	0	0
3	11	4	1	2	0	1352	1	1	2	2	6	195	3	0	0	3	0
4	40	1	1	1	0	274	1	0	2	4	8	222	0	0	0	3	0

```
In [106]: import numpy as np
from kmodes.kmodes import KModes

km_cao = KModes(n_clusters=2)
fitClusters_cao = km_cao.fit_predict(bank)
```

```
In [107]: # Predicted Clusters
fitClusters_cao
```

```
Out[107]: array([0, 0, 0, ..., 0, 0, 0], dtype=uint16)
```

```
In [108]: clusterCentroidsDF = pd.DataFrame(km_cao.cluster_centroids_)
clusterCentroidsDF.columns = bank.columns
```

```
# Mode of the clusters
clusterCentroidsDF
```

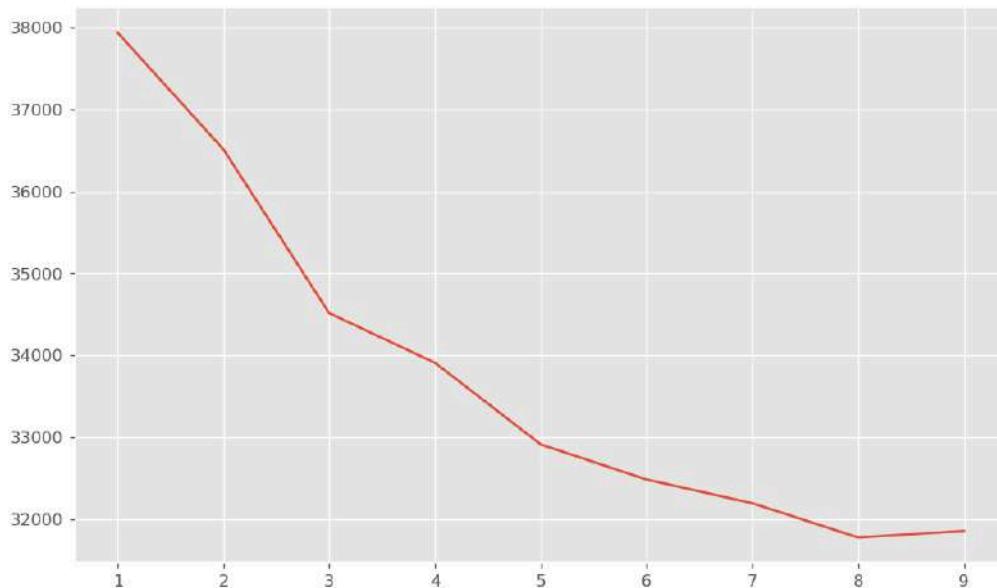
```
Out[108]:   age job marital education default balance housing loan contact day month duration campaign pdays previous poutcome y
0    15    4       1        1     0    274      1     0     0    17     8    119      0     0     0    3  0
1    11    1       2        0     0    274      0     0     2    19     6    25      1     0     0    3  0
```

```
In [109... cost = []
for num_clusters in list(range(1,10)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(bank)
    cost.append(kmode.cost_)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 37936.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 159, cost: 36513.0
Run 1, iteration: 2/100, moves: 0, cost: 36513.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 806, cost: 34515.0
Run 1, iteration: 2/100, moves: 23, cost: 34515.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 919, cost: 33910.0
Run 1, iteration: 2/100, moves: 18, cost: 33910.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1180, cost: 32917.0
Run 1, iteration: 2/100, moves: 207, cost: 32914.0
Run 1, iteration: 3/100, moves: 1, cost: 32914.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1299, cost: 32487.0
Run 1, iteration: 2/100, moves: 297, cost: 32486.0
Run 1, iteration: 3/100, moves: 1, cost: 32486.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1298, cost: 32209.0
Run 1, iteration: 2/100, moves: 300, cost: 32197.0
Run 1, iteration: 3/100, moves: 0, cost: 32197.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1356, cost: 31863.0
Run 1, iteration: 2/100, moves: 365, cost: 31779.0
Run 1, iteration: 3/100, moves: 58, cost: 31779.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 1319, cost: 31861.0
Run 1, iteration: 2/100, moves: 254, cost: 31859.0
Run 1, iteration: 3/100, moves: 3, cost: 31859.0
```

```
In [110... y = np.array([i for i in range(1,10)])
plt.plot(y,cost)
```

```
Out[110]: [
```



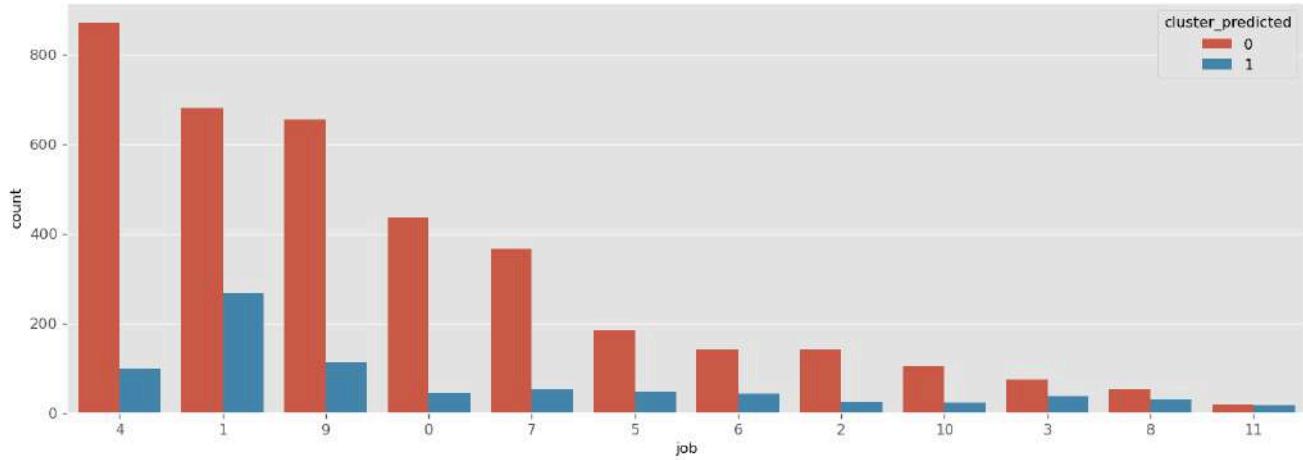
```
In [111... bank_cust = bank.reset_index()
clustersDf = pd.DataFrame(fitClusters_cao)
clustersDf.columns = ['cluster_predicted']
combinedDf = pd.concat([bank_cust, clustersDf], axis = 1).reset_index()
combinedDf = combinedDf.drop(['index', 'level_0'], axis = 1)
```

```
In [112... combinedDf.head()
```

```
Out[112]:   age job marital education default balance housing loan contact day month duration campaign pdays previous poutcome y  cluster_predicted
0    11    10      1        0     0    1475      0     0     0    18    10     75      0     0     0    3  0          0
1    14     7      1        1     0    2030      1     1     0    10     8    216      0    228     4     0  0          0
2    16     4      2        2     0    1303      1     0     0    15     0    181      0    219     1     0  0          0
3    11     4      1        2     0    1352      1     1     2    2  6    195      3     0     0    3  0          0
4    40     1      1        1     0    274      1     0     2    4  8    222      0     0     0    3  0          0
```

```
In [113... import seaborn as sns
plt.subplots(figsize = (15,5))
```

```
sns.countplot(x=combinedDF['job'],order=combinedDF['job'].value_counts().index,hue=combinedDF['cluster_predicted'])  
plt.show()
```



```
In [114]:  
import numpy as np  
from kmodes.kmodes import KModes  
  
# random categorical data  
data = np.random.choice(20, (100, 10))  
  
km = KModes(n_clusters=4, init='Huang', n_init=5, verbose=1)  
  
clusters = km.fit_predict(data)  
  
# Print the cluster centroids  
print(km.cluster_centroids_)
```

```
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 1, iteration: 1/100, moves: 18, cost: 807.0  
Run 1, iteration: 2/100, moves: 8, cost: 801.0  
Run 1, iteration: 3/100, moves: 0, cost: 801.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 2, iteration: 1/100, moves: 24, cost: 810.0  
Run 2, iteration: 2/100, moves: 0, cost: 810.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 3, iteration: 1/100, moves: 19, cost: 801.0  
Run 3, iteration: 2/100, moves: 14, cost: 796.0  
Run 3, iteration: 3/100, moves: 0, cost: 796.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 4, iteration: 1/100, moves: 18, cost: 786.0  
Run 4, iteration: 2/100, moves: 7, cost: 786.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 5, iteration: 1/100, moves: 28, cost: 795.0  
Run 5, iteration: 2/100, moves: 5, cost: 793.0  
Run 5, iteration: 3/100, moves: 0, cost: 793.0  
Best run was number 4  
[[ 0  8  0  0 13  5 17 15  4 16]  
[ 1 11  2  2 11 19  9  0  9 19]  
[ 9 19 16  7  4  3 11 13 11  1]  
[ 6 10  1  1 13  9  1  4  4]]
```

```
In [115]:  
# Building the model with 3 clusters  
kmode = KModes(n_clusters=3, init = "random", n_init = 5, verbose=1)  
clusters = kmode.fit_predict(data)
```

```
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 1, iteration: 1/100, moves: 26, cost: 823.0  
Run 1, iteration: 2/100, moves: 0, cost: 823.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 2, iteration: 1/100, moves: 26, cost: 825.0  
Run 2, iteration: 2/100, moves: 3, cost: 825.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 3, iteration: 1/100, moves: 18, cost: 819.0  
Run 3, iteration: 2/100, moves: 5, cost: 816.0  
Run 3, iteration: 3/100, moves: 2, cost: 816.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 4, iteration: 1/100, moves: 23, cost: 815.0  
Run 4, iteration: 2/100, moves: 3, cost: 815.0  
Init: initializing centroids  
Init: initializing clusters  
Starting iterations...  
Run 5, iteration: 1/100, moves: 24, cost: 818.0  
Run 5, iteration: 2/100, moves: 4, cost: 818.0  
Best run was number 4  
array([0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 1, 1, 0, 0, 1, 0, 1, 1, 2, 2, 0, 1,  
2, 2, 0, 0, 0, 2, 2, 1, 1, 2, 0, 0, 0, 2, 1, 1, 2, 2, 2, 2, 0,  
1, 2, 2, 0, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,  
1, 1, 0, 0, 1, 1, 2, 0, 1, 2, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,  
2, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1], dtype=uint16)
```

# Machine learning

## Dimensionality reduction

Morteza khorsand



### Dimensionality reduction

2

#### Dimensionality reduction

Dimensionality reduction refers to techniques that reduce the number of input variables in a dataset.

#### curse of dimensionality

The curse of dimensionality basically refers to the difficulties a machine learning algorithm faces when working with data in higher dimensions.

High dimensionality might mean hundreds, thousands, or even millions of input variables.

#### Goal

Visualize Data

Data compression

- Fewer features mean less complexity.
- You will need less storage space because you have fewer data.
- Fewer features require less computation time.
- Model accuracy improves due to less misleading data.

## Dimensionality reduction

### EigenVector and Eigenvalue

$$\begin{bmatrix} 1 & 2 \\ 8 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 5 \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$A \cdot V = \lambda V$$

$$\det(A - \lambda I) = 0$$

$$\begin{bmatrix} 1 & 2 \\ 8 & 1 \end{bmatrix} \cdot \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 1-\lambda & 2 \\ 8 & 1-\lambda \end{bmatrix} \rightarrow (1-\lambda)^2 - 16 = 0 \rightarrow$$

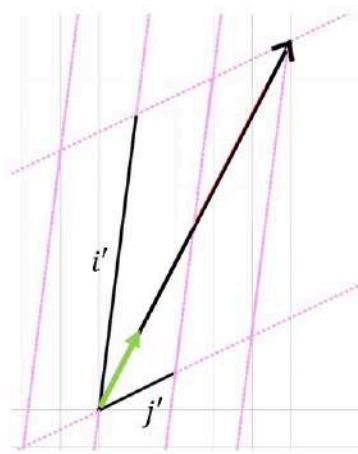
$$1 + \lambda^2 - 2\lambda - 16 = 0$$

$$\lambda^2 - 2\lambda - 15 = 0 \rightarrow \begin{cases} \lambda = 5 \\ \lambda = -3 \end{cases}$$

$$\lambda = 5 \rightarrow A \cdot V = \lambda V$$

$$\begin{bmatrix} 1 & 2 \\ 8 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = 5 \begin{bmatrix} X \\ Y \end{bmatrix} \rightarrow X + 2Y = 5X \\ 8X + Y = 5Y \\ \rightarrow 2Y = X$$

$$V^T = \begin{bmatrix} 1 & 2 \\ -1 & 2 \end{bmatrix}$$



The sum of eigenvalues is equal to the sum of the diagonal elements of the matrix (trace).

The product of eigenvalues is equal to the determinant of the matrix.

A matrix and its transpose have the same eigenvalues

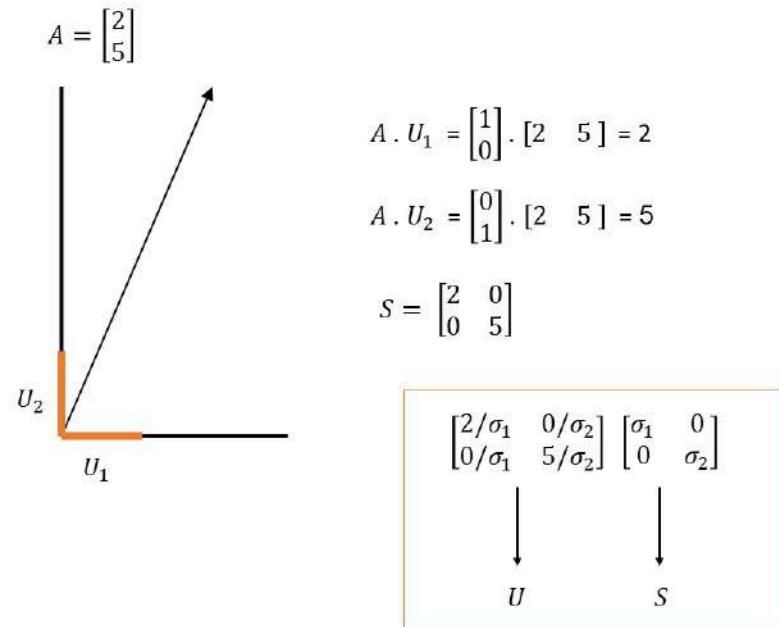
```
In [1]: import numpy as np
a=np.array([[1,2],[8,1]])

In [2]: eigenvalues, eigenvectors=np.linalg.eig(a)
eigenvalues
Out[2]: array([ 5., -3.])

In [3]: eigenvectors
Out[3]: array([[ 0.4472136 , -0.4472136 ],
   [ 0.89442719,  0.89442719]])
```

## Dimensionality reduction

### Dot Product and similarity



## Dimensionality reduction

### Covariance matrix

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \text{var}(X) & \text{cov}(X, Y) \\ \text{cov}(X, Y) & \text{var}(Y) \end{bmatrix}$$

Compute the Mean Vector  $X$

$$\bar{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Center the Data:

$$\begin{bmatrix} 1 - 3 & 2 - 4 \\ 3 - 3 & 4 - 4 \\ 5 - 3 & 6 - 4 \end{bmatrix} = \begin{bmatrix} -2 & -2 \\ 0 & 0 \\ 2 & 2 \end{bmatrix}$$

Compute the Covariance Matrix:

$$\Sigma = \frac{1}{m-1} \tilde{X}^T \tilde{X} = \frac{1}{2} \begin{bmatrix} -2 & 0 & 2 \\ -2 & 0 & 2 \end{bmatrix} \begin{bmatrix} -2 & -2 \\ 0 & 0 \\ 2 & 2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 8 & 8 \\ 8 & 8 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$$

```
In [4]: X = np.array([[1,2],  
                   [3,4],  
                   [5,6]])  
  
np.cov(X.T)
```

```
Out[4]: array([[4., 4.],  
   [4., 4.]])
```

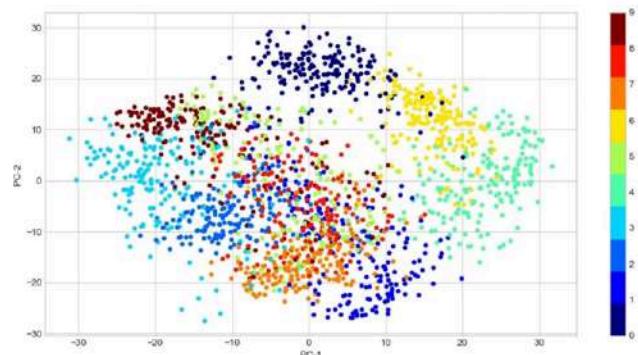
## Dimensionality reduction

3

### Dimensionality Reduction Techniques

#### Principal component analysis (PCA)

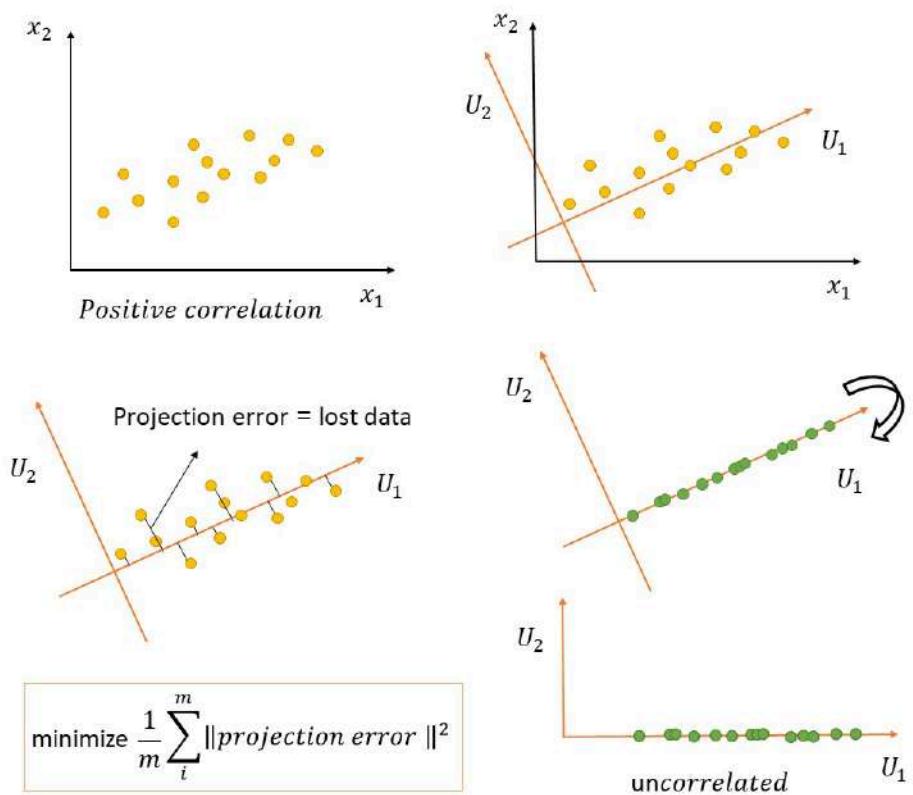
- Missing value ratio.
- Backward feature elimination.
- Forward feature selection.
- Random forest.
- Factor analysis.
- Independent component analysis (ICA).
- Low variance filter.



## Dimensionality reduction

### PCA

X <sub>1</sub>	X <sub>2</sub>
1	1.5
1.25	1.6
1.36	1.7
2	1.8
2.01	1.8
2.99	1.9
3	2
3.1	2.01
3.2	2.03
4	2.03
4.1	2.5
4.3	1.87
4.8	2.3
5	2.2
R=4	R=1



$$\text{minimize } \frac{1}{m} \sum_i^m \|\text{projection error}\|^2$$

```
In [5]: import numpy as np  
import pandas as pd  
X1 = np.array([1,1.25,1.36,2,2.01,2.99,3,3.1,3.2,4,4.1,4.3,4.8,5])
```

```
X2=np.array([1.5,1.6,1.7,1.8,1.8,1.9 ,2,2.01,2.03,2.03,2.5,1.87,2.3,2.2])

df = pd.DataFrame({'X1': X1, 'X2': X2})
display(df)

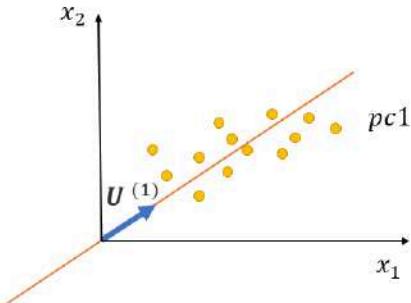
df.corr()

   X1  X2
0  1.00 1.50
1  1.25 1.60
2  1.36 1.70
3  2.00 1.80
4  2.01 1.80
5  2.99 1.90
6  3.00 2.00
7  3.10 2.01
8  3.20 2.03
9  4.00 2.03
10 4.10 2.50
11 4.30 1.87
12 4.80 2.30
13 5.00 2.20

Out[5]:      X1      X2
X1  1.000000  0.848639
X2  0.848639  1.000000
```

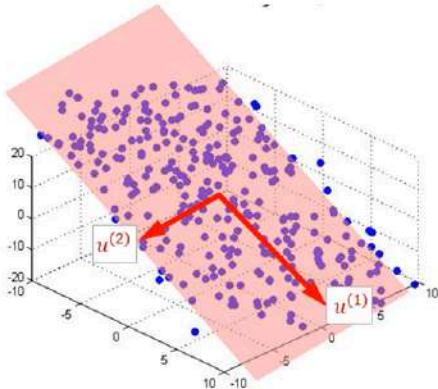
## Dimensionality reduction

5



### Reduction dimension from 2 to 1

Finding a direction like  $U^{(1)} \in \mathbb{R}^2$  to minimize So that by depicting the points in that direction, the sum of the squares of the error is minimized.



### Dimension reduction from n to k ( $k \leq n$ )

finding k orthogonal vectors such as  $U^{(1)}, U^{(2)}, U^{(3)}, \dots, U^{(k)} \in \mathbb{R}^n$  so that by depicting the points in those directions, the sum of squared errors is minimized.

## PCA Implementation

```
In [6]: import sys
import numpy as np
np.set_printoptions(precision=2)
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [7]: X = np.array([[1, 1, 1, 0, 0],
[2, 2, 2, 0, 0],
[1, 1, 1, 0, 0],
[5, 5, 5, 0, 0],
[1, 1, 0, 2, 2],
[0, 0, 0, 3, 3],
[0, 0, 0, 1, 1]])
```

```
In [8]: data= pd.DataFrame(X)
data

display(data.corr())
```

	0	1	2	3	4
0	1.000000	1.000000	0.977965	-0.524628	-0.524628
1	1.000000	1.000000	0.977965	-0.524628	-0.524628
2	0.977965	0.977965	1.000000	-0.588069	-0.588069
3	-0.524628	-0.524628	-0.588069	1.000000	1.000000
4	-0.524628	-0.524628	-0.588069	1.000000	1.000000

## Dimensionality reduction

6

### PCA

#### Decorrelation

- 1. rotate the samples to be aligned with axes
  - 2. shift data samples so they have zero mean
- Reduce dimension

### Pre processing

- Remove average (zero means)
- subtract the average data of the first column from all the data of the first column and so on.

$$\mu_j = \frac{1}{m} \sum_i^m x^{(i)}_j \quad x^{(i)}_j = x^{(i)}_j - \mu_j$$

$$\begin{aligned} \text{mu} &= X.\text{mean}(\text{axis} = 0) \\ s_j &= X.\text{std}(\text{axis}=0) \\ X_{\text{norm}} &= (X - \text{mu}) / s \end{aligned}$$

- Scaling (if needed)

$$x^{(i)}_j = \frac{x^{(i)}_j - \mu_j}{s_j}$$

# STEP 1: Zero-center data preprocessing

```
In [9]: #preprocessing
mu = X.mean (axis = 0)
std=X.std(axis=0)
print (" the mean of X is{mu}\n the standard deviation is{std}")

print("-----")
X_norm =(X - mu)
print(X_norm )

the mean of X is[1.43 1.43 1.29 0.86 0.86]
the standard deviation is[1.59 1.59 1.67 1.12 1.12]
-----
[[ -0.43 -0.43 -0.29 -0.86 -0.86]
 [ 0.57 0.57 0.71 -0.86 -0.86]
 [ -0.43 -0.43 -0.29 -0.86 -0.86]
 [ 3.57 3.57 3.71 -0.86 -0.86]
 [ -0.43 -0.43 -1.29 1.14 1.14]
 [ -1.43 -1.43 -1.29 2.14 2.14]
 [ -1.43 -1.43 -1.29 0.14 0.14]]
```

### Dimension reduction from n to k

#### 1. Covariance matrix

$$\Sigma = \frac{1}{m} X^T X = \frac{1}{m} \sum_{i=1}^m x^{(i)} (x^{(i)})^T$$

#### 2. Covariance matrix decomposition (SVD : singular value decomposition)

$$\Sigma = U \times S \times V \quad [U \times S \times V] = \text{svd}(\Sigma)$$

$$U = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ U^{(1)} & U^{(2)} & U^{(3)} & \vdots & U^{(n)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{n \times n} \quad S = \begin{bmatrix} s_{11} & 0 & 0 & \cdot & 0 \\ 0 & s_{22} & 0 & \cdot & 0 \\ \cdot & \cdot & s_{33} & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & \cdot & s_{nn} \end{bmatrix} \quad s_{11} > s_{22} > s_{33} \dots s_{nn}$$

#### 3. Select first k matrices from U matrix

$$U_{reduced} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ U^{(1)} & U^{(2)} & U^{(3)} & \vdots & U^{(k)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{n \times k}$$

# STEP 2: Compute covariance matrix

```
In [10]: m=X.shape[0]
sigma=(1/m)* X.T@X
print(sigma)

[[4.57 4.57 4.43 0.29 0.29]
 [4.57 4.57 4.43 0.29 0.29]
 [4.43 4.43 4.43 0. 0. ]
 [0.29 0.29 0. 2. 2. ]
 [0.29 0.29 0. 2. 2. ]]
```

# STEP 3: Singular Value Decomposition

```
In [11]: #decomposition (svd)
U, S, V = np.linalg.svd(sigma)

print(U)
print("-----")
print(S)

[[-5.81e-01 -4.61e-03 4.03e-01 -7.07e-01 1.08e-16]
 [-5.81e-01 -4.61e-03 4.03e-01 7.07e-01 -9.53e-17]
 [-5.67e-01 9.62e-02 -8.18e-01 -9.22e-15 -1.31e-17]
 [-3.50e-02 -7.04e-01 -5.85e-02 -6.14e-16 -7.07e-01]
 [-3.50e-02 -7.04e-01 -5.85e-02 -7.25e-16 7.07e-01]]
-----
```

```
[1.35e+01 4.00e+00 6.69e-02 5.87e-16 6.06e-48]
```

```
In [12]: U_reduced = U[ :, :2 ]
```

```
print(U_reduced)

print(np.linalg.norm(U_reduced , axis=0))
print(round(np.dot(U[0], U[1])))

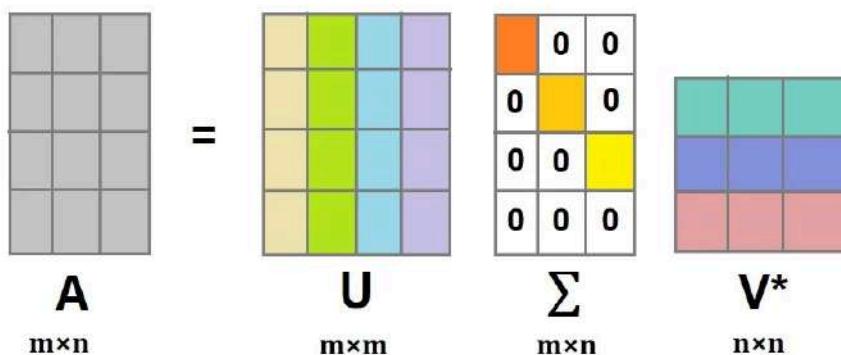
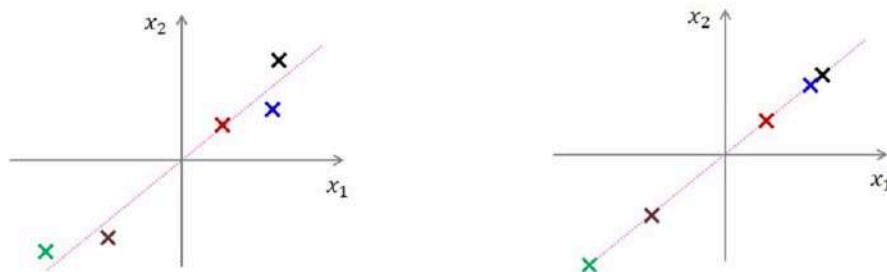
[[-0.58 0. ]
 [-0.58 0. ]
 [-0.57 0.1 ]
 [-0.03 -0.7 ]
 [-0.03 -0.7 ]]
[1. 1.]
0
```

## 4. Project data on new vectors

$$[U_{reduced}]^T \text{ } k \times n \times x^{(i)}_{n \times 1} = [X_{proj}] \text{ } k \times 1$$

5. We can revert the data back to its initial space to see how much data is lost.

$$X_{recovered} = X_{proj} \times u^T_{reduced} + means$$



```
In [13]: X_proj = X_norm @ U_reduced
print(X_proj)
[[ 0.72  1.18]
 [-1.01  1.27]
 [ 0.72  1.18]
 [-6.2   1.53]
 [ 1.15 -1.73]
 [ 2.24 -3.13]
 [ 2.38 -0.31]]
```

# STEP 5: Recover

```
In [14]: X_approx = (X_proj @ U_reduced.T + mu)
print(np.round(X_approx))
[[ 1.  1. -0. -0.]
 [ 2.  2. -0. -0.]
 [ 1.  1.  1. -0.]
 [ 5.  5. -0. -0.]
 [ 1.  1.  2.  2.]
 [ 0.  0. -0.  3.]
 [ 0.  0. -0.  1. 1.]]
```

```
In [15]: reconstructed = np.matmul(U[:, :2], np.diag(S[2])) .dot(V[2:, :])
reconstructed
```

```
Out[15]: array([[ 4.56e+00,  4.56e+00,  4.45e+00,  2.87e-01,  2.87e-01],
 [ 4.56e+00,  4.56e+00,  4.45e+00,  2.87e-01,  2.87e-01],
 [ 4.45e+00,  4.45e+00,  4.38e+00, -3.20e-03, -3.20e-03],
 [ 2.87e-01,  2.87e-01, -3.20e-03,  2.00e+00,  2.00e+00],
 [ 2.87e-01,  2.87e-01, -3.20e-03,  2.00e+00,  2.00e+00]])
```

```
In [16]: import numpy as np
from PIL import Image

img= Image.open(r"D:\AI_Machinlearning\datasets\building\pca\41.jpg").convert("L")
img
```

Out[16]:



```
In [17]: X= np.array(img)

X.shape
```

Out[17]: (600, 450)

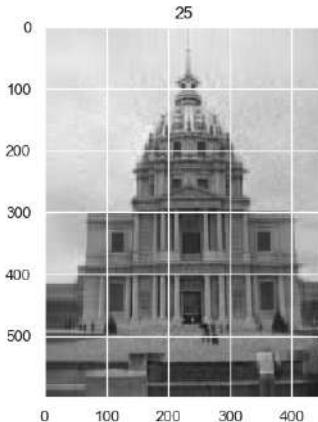
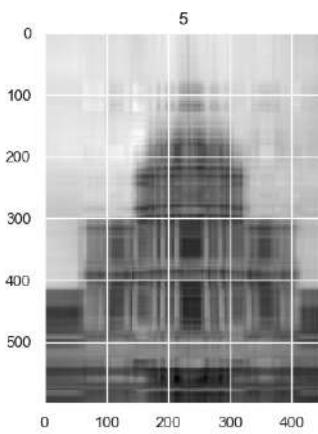
```
In [31]: import numpy as np
import cv2
import matplotlib.pyplot as plt

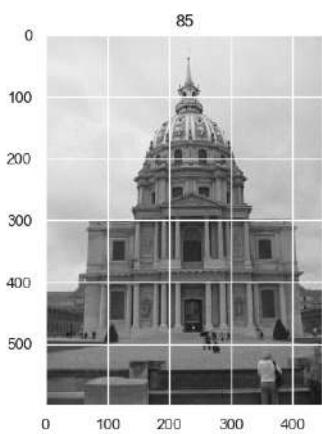
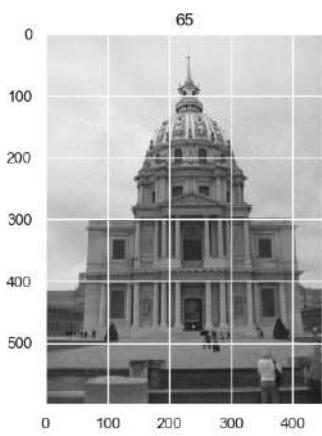
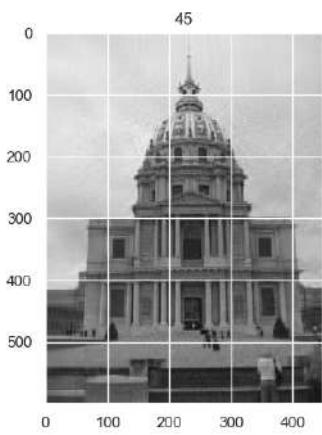
img = cv2.imread(r"D:\AI_Machinlearning\datasets\building\pca\41.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

u, s, v = np.linalg.svd(gray)

s_matrix = np.diag(s[:2])

for i in range(5, 100, 20):
    reconstructed_image = np.matmul(u[:, :i], np.diag(s[:i])).dot(v[:i, :])
    plt.imshow(reconstructed_image, cmap='gray')
    plt.title("{} ".format(i))
    plt.show()
```





### Select "k"

1. The mean of the sum of squared projection errors.

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01$$

```
K=0
Repeat
{
K=k+1
Try Pca(X) with K components
```

Compute  $U_{reduced}$ ,  $Z^{(1)}, Z^{(2)}, Z^{(3)}, \dots, Z^{(n)}$ ,  $x_{approx}^{(1)}, x_{approx}^{(2)}, \dots, x_{approx}^{(m)}$

$$\text{Until } \frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01$$

# Choosing number of principal components

```
In [19]: m, n = X.shape
X = X - X.mean(axis=0)
sigma = (X.T @ X) / m
U, S, V = np.linalg.svd(sigma)

for k in range(n+1):
    total_var = np.sum(S[:k]) / np.sum(S)
    if total_var >= 0.99:
        print(k)
        break
```

72

### PCA class

```
In [32]: class Pca:
    def __init__(self, X, variance):
        self.X = X
        self.variance = variance

    def train(self):
        m, n = self.X.shape
        mu = self.X.mean(axis=0)
        std = self.X.std(axis=0)
        X = (self.X - mu) / std

        sigma = self.X.T @ self.X
        U, S, _ = np.linalg.svd(sigma)

        last_k = 0
        for k in range(n+1):
            total_var = np.sum(S[:k]) / np.sum(S)
            if total_var >= self.variance:
                last_k = k
                break

        U_reduced = U[:, :last_k]
        X_recovered = ((X @ U_reduced.T) + mu) * std

        return sigma, U, S, U_reduced, X_reduced, X_recovered, k
```

### Example

```
In [21]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

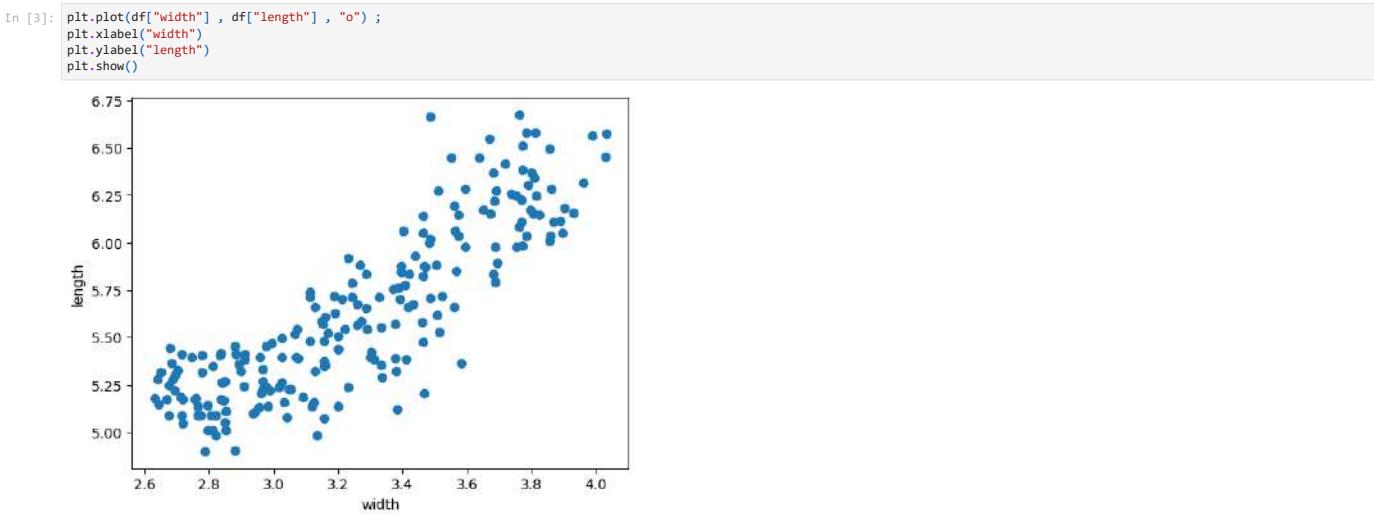
In [2]: import pandas as pd
seed = pd.read_csv(r"D:\AI_Machinelearning\datasets\plant\01\seed\width-vs-length.csv")
data = np.array(seed)
d = {'width': data[:, 0], 'length': data[:, 1]}
```

```
df=pd.DataFrame(d)
df
```

Out[2]:

	width	length
0	3.333	5.554
1	3.337	5.291
2	3.379	5.324
3	3.562	5.658
4	3.312	5.386
...	...	...
204	2.981	5.137
205	2.795	5.140
206	3.232	5.236
207	2.836	5.175
208	2.974	5.243

209 rows × 2 columns



In [4]:

```
df.corr()
```

Out[4]:

	width	length
width	1.000000	0.860441
length	0.860441	1.000000

In [ ]:

```
model= Pca(data , 0.99)

sigma , u , s , u_reduced , x_proj , X_recoverd , k =model.train()

print(sigma)
print("-----")
print(u)
print("-----")
print(s)
print("-----")
print(u_reduced)
print("-----")
print((np.linalg.norm(u_reduced)))
print("-----")
print((round(np.dot(u[0] , u[1]))))
print("-----")
print(x_proj[: 10])
print("-----")
print(X_recoverd[: 10])
```

## Sklearn

In [5]:

```
from sklearn.decomposition import PCA
```

In [6]:

```
pca = PCA()
pca.fit(data)          #project
```

In [7]:

```
U = pca.components_    # Principal Components (directions)
S = pca.explained_variance_ # importance of each direction (variances)

print("1st Principal Component: {} ({:.2f})".format(U[0], S[0]))
print("2nd Principal Component: {} ({:.2f})".format(U[1], S[1]))
```

1st Principal Component: [0.63917819 0.76905867] (0.32)
2nd Principal Component: [-0.76905867 0.63917819] (0.02)

In [8]:

```
pca.explained_variance_ratio_
```

Out[8]:

```
array([0.93210095, 0.06789905])
```

In [9]:

```
print(np.linalg.norm(U[0]))
print(np.linalg.norm(U[1]))
```

1.0  
1.0

In [10]:

```
print(np.dot(U[0] , U[1]))
```

0.0

In [11]:

```
mean=pca.mean_
mean
# plot data
c = transformed[:, 0]           # colors
plt.scatter(data[:, 0], data[:, 1], s=50, c=c, cmap='viridis', alpha=0.5)
```

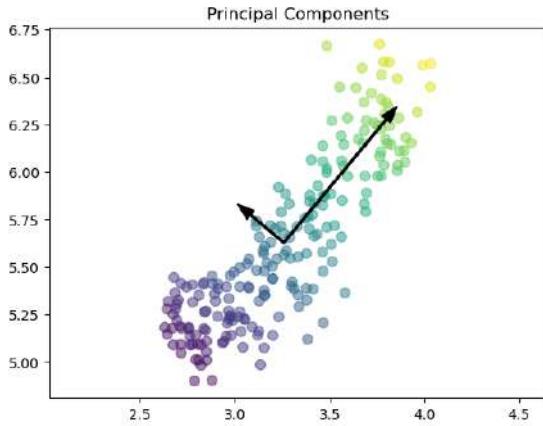
```

plt.arrow(mean[0] , mean[1], 1.5 * np.sqrt(S[0]) * U[0, 0],
          1.5 * np.sqrt(S[0]) * U[0, 1],
          width=.01, head_width=.06, color='k')

plt.arrow( mean[0] , mean[1], 1.5 * np.sqrt(S[1]) * U[1, 0],
          1.5 * np.sqrt(S[1]) * U[1, 1],
          width=.01, head_width=.06, color='k')

plt.title("Principal Components")
plt.axis('equal')
plt.show()

```



```

In [12]: pca = PCA(.93)           # keep 93% of variance
X_proj = pca.fit_transform(data)

print(data.shape)
print(X_proj.shape)

(209, 2)
(209, 1)

```

```

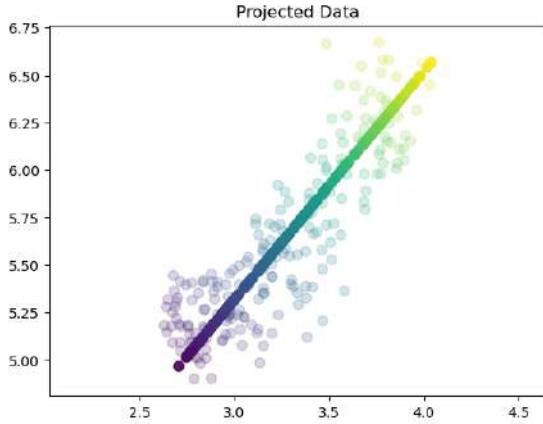
In [13]: X_approx = pca.inverse_transform(X_proj)

# plot original data
plt.scatter(data[:, 0], data[:, 1], s=50, c=c, cmap='viridis', alpha=0.2)

# plot projected data
plt.scatter(X_approx[:, 0], X_approx[:, 1], s=50, c=c, cmap='viridis', alpha=0.9)

plt.title("Projected Data")
plt.axis('equal')
plt.show()

```



```

In [14]: plt.figure(figsize=(12, 6))

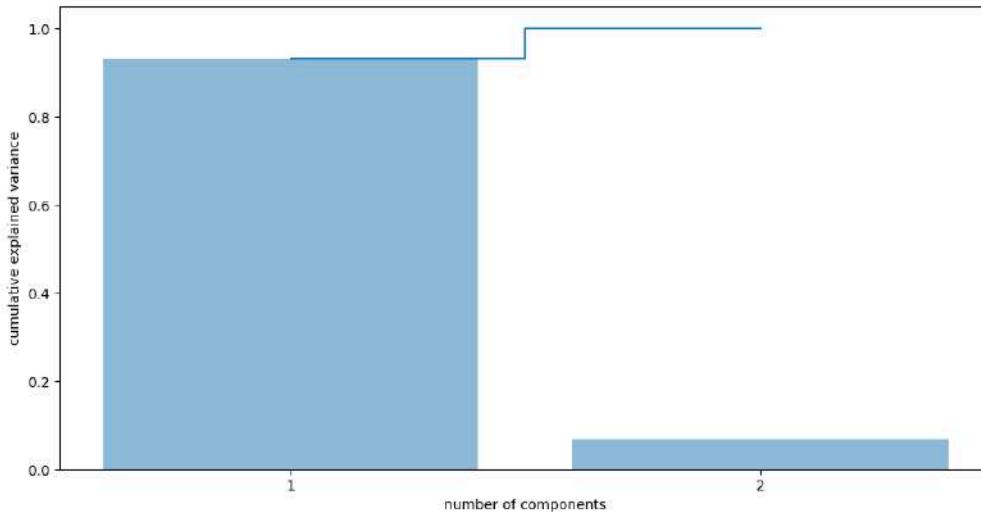
pca = PCA().fit(data) # Notice

plt.bar(range(len(pca.explained_variance_ratio_)),
        pca.explained_variance_ratio_,
        alpha=0.5,
        align='center')

plt.step(range(len(pca.explained_variance_ratio_)),
         np.cumsum(pca.explained_variance_ratio_),
         where='mid')

plt.xticks([0, 1], [1, 2])
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');

```



```
In [15]: total_var = np.cumsum(pca.explained_variance_ratio_)

for i in [0, 1]:
    print("Components: {}, total explained variance: {:.2f}".format(i, total_var[i]))

Components: 0, total explained variance: 0.93
Components: 1, total explained variance: 1.00
```

```
In [16]: from sklearn.datasets import load_digits

mnist = load_digits()
X, y = mnist.data, mnist.target
```

```
In [17]: import matplotlib.pyplot as plt

# select 100 faces randomly
X_samples = np.random.permutation(X)[:10]

fig, axes = plt.subplots(1, 10, figsize=(12, 12))
fig.subplots_adjust(left=0.2, bottom=0.2, right=0.8, top=0.8, hspace=0.01, wspace=0.01)

for i, ax in enumerate(axes.flat):
    ax.imshow(X_samples[i].reshape((8, 8)))
    ax.set_xticks(())
    ax.set_yticks(())
plt.show()
```



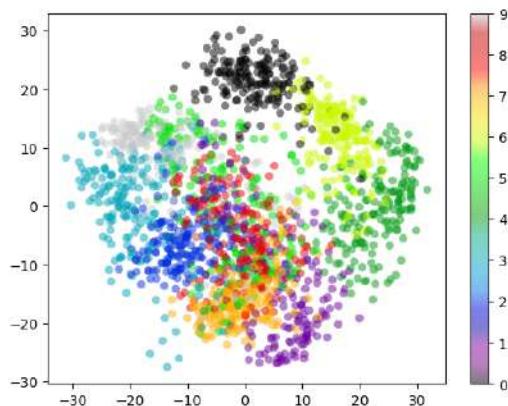
```
In [18]: from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_proj = pca.fit_transform(X) # project from 64 to 2 dimensions

print("Shape of original data: {}".format(X.shape))
print("Shape of projected data: {}".format(X_proj.shape))

Shape of original data: (1797, 64)
Shape of projected data: (1797, 2)
```

```
In [19]: plt.scatter(X_proj[:, 0], X_proj[:, 1],
                  c=y, edgecolor='none', alpha=0.5,
                  cmap=plt.colormaps.get_cmap('nipy_spectral'))
plt.colorbar()
plt.show()
```



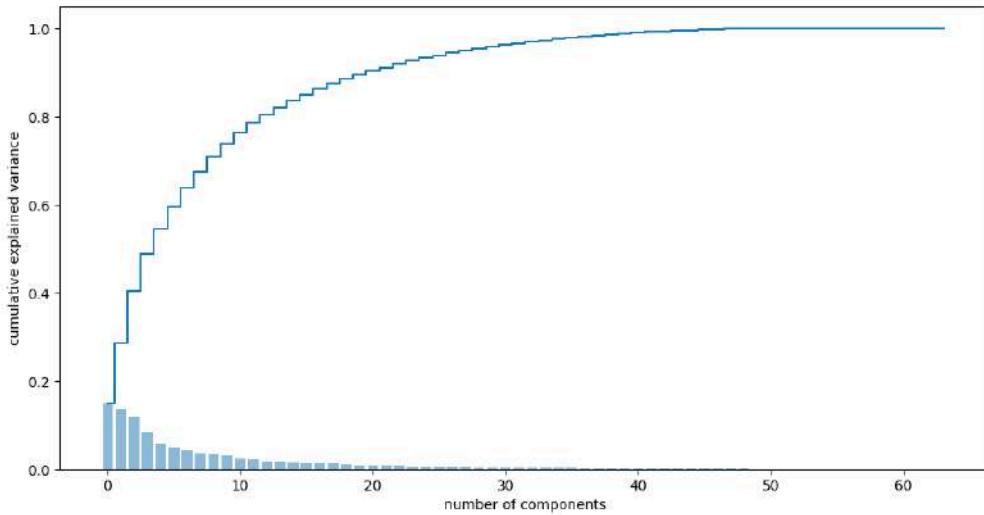
```
In [20]: plt.figure(figsize=(12, 6))

pca = PCA().fit(X) # Notice

plt.bar(range(len(pca.explained_variance_ratio_)),
        pca.explained_variance_ratio_,
        alpha=0.5,
        align='center')

plt.step(range(len(pca.explained_variance_ratio_)),
         np.cumsum(pca.explained_variance_ratio_),
         where='mid')

plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



In [21]:

```
total_var = np.cumsum(pca.explained_variance_ratio_)

for i in [0, 1, 2, 3, 4, 5, 8, 12, 20, 27, 36, 40, 50]:
    print("Components: {:2d}, total explained variance: {:.2f}".format(i, total_var[i]))
```

Components: 0, total explained variance: 0.15  
 Components: 1, total explained variance: 0.29  
 Components: 2, total explained variance: 0.40  
 Components: 3, total explained variance: 0.49  
 Components: 4, total explained variance: 0.54  
 Components: 5, total explained variance: 0.59  
 Components: 8, total explained variance: 0.71  
 Components: 12, total explained variance: 0.80  
 Components: 20, total explained variance: 0.90  
 Components: 27, total explained variance: 0.95  
 Components: 36, total explained variance: 0.98  
 Components: 40, total explained variance: 0.99  
 Components: 50, total explained variance: 1.00

## SECOM

In [3]:

```
import numpy as np
data = []
with open(r"D:\AI_Machinlearning\datasets\secom\secom.data") as f:
    for line in f:
        data.append([float(x) for x in line.split(' ')])
```

SECOM = np.array(data)

print(SECOM.shape)

(1567, 590)

In [5]:

```
import pandas as pd
df = pd.DataFrame(data=SECOM)
df.head(10)
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	580	581	582	583	584	585	586	587	588	589
0	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	0.0162	...	Nan	Nan	0.5005	0.0118	0.0035	2.3630	Nan	Nan	Nan	Nan
1	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	-0.0005	...	0.0060	208.2045	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201	0.0060	208.2045
2	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	0.0041	...	0.0148	82.8602	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484	0.0148	82.8602
3	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	-0.0124	...	0.0044	73.8432	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149	0.0044	73.8432
4	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	-0.0031	...	Nan	Nan	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149	0.0044	73.8432
5	2946.25	2432.84	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5287	0.0167	...	0.0052	44.0077	0.4949	0.0189	0.0044	3.8276	0.0342	0.0151	0.0052	44.0077
6	3030.27	2430.12	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.5816	-0.0270	...	Nan	Nan	0.5010	0.0143	0.0042	2.8515	0.0342	0.0151	0.0052	44.0077
7	3058.88	2690.15	2248.9000	1004.4692	0.7884	100.0	106.2400	0.1185	1.5153	0.0157	...	0.0063	95.0310	0.4984	0.0106	0.0034	2.1261	0.0204	0.0194	0.0063	95.0310
8	2967.68	2600.47	2248.9000	1004.4692	0.7884	100.0	106.2400	0.1185	1.5358	0.0111	...	0.0045	111.6525	0.4993	0.0172	0.0046	3.4456	0.0111	0.0124	0.0045	111.6525
9	3016.11	2428.37	2248.9000	1004.4692	0.7884	100.0	106.2400	0.1185	1.5381	0.0159	...	0.0073	90.2294	0.4967	0.0152	0.0038	3.0687	0.0212	0.0191	0.0073	90.2294

10 rows × 590 columns

In [6]:

df.isnull().sum().sum()

41951

In [7]:

df.fillna(df.mean(), axis=0, inplace=True)

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	580	581	582	583	584	585	586	587	588	589
0	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	0.016200	...	0.005396	97.934373	0.5005	0.0118	0.0035	2.3630	0.021458	0.016475	0.005283	99.670066
1	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	-0.000500	...	0.006000	208.204500	0.5019	0.0223	0.0055	4.4447	0.009600	0.020100	0.006000	208.204500
2	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.443600	0.004100	...	0.014800	82.860200	0.4958	0.0157	0.0039	3.1745	0.058400	0.048400	0.014800	82.860200
3	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.488200	-0.012400	...	0.004400	73.843200	0.4990	0.0103	0.0025	2.0544	0.020200	0.014900	0.004400	73.843200
4	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.503100	-0.003100	...	0.005396	97.934373	0.4800	0.4766	0.1045	99.3032	0.020200	0.014900	0.004400	73.843200
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1562	2899.41	2464.36	2179.7333	3085.3781	1.4843	100.0	82.2467	0.1248	1.342400	-0.004500	...	0.004700	203.172000	0.4988	0.0143	0.0039	2.8669	0.006800	0.013800	0.004700	203.172000
1563	3052.31	2522.55	2198.5667	1124.6595	0.8763	100.0	98.4689	0.1205	1.433300	-0.006100	...	0.005396	97.934373	0.4975	0.0131	0.0036	2.6238	0.006800	0.013800	0.004700	203.172000
1564	2978.81	2379.78	2206.3000	1110.4967	0.8236	100.0	99.4122	0.1208	1.462862	-0.000841	...	0.002500	43.523100	0.4987	0.0153	0.0041	3.0590	0.019700	0.008600	0.002500	43.523100
1565	2894.92	2532.01	2177.0333	1183.7287	1.5726	100.0	98.7978	0.1213	1.462200	-0.007200	...	0.007500	93.494100	0.5004	0.0178	0.0038	3.5662	0.026200	0.024500	0.007500	93.494100
1566	2944.92	2450.76	2195.4444	2914.1792	1.5978	100.0	85.1011	0.1235	1.462862	-0.000841	...	0.004500	137.784400	0.4987	0.0181	0.0040	3.6275	0.011700	0.016200	0.004500	137.784400

1567 rows × 590 columns

In [8]:

df.isnull().sum().sum()

```
Dut[8]: 0
```

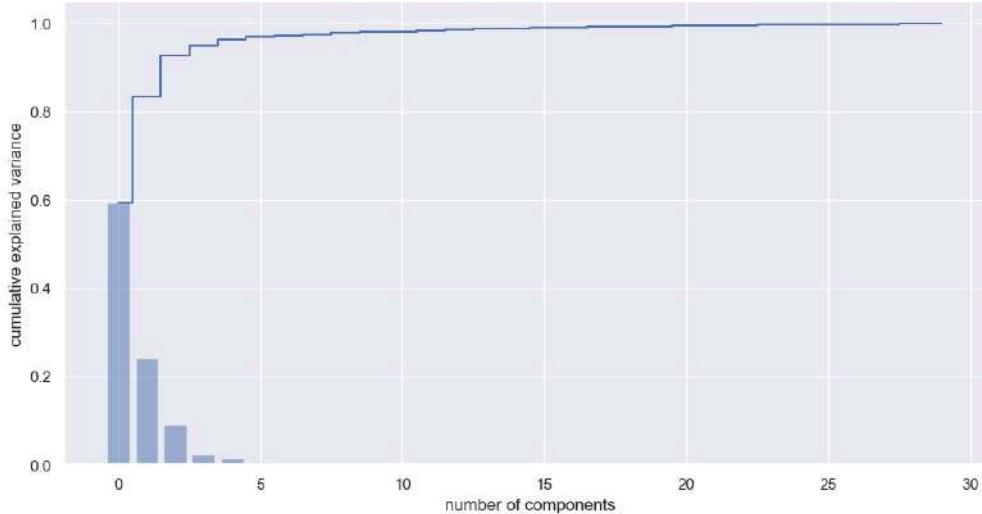
```
In [10]: import matplotlib.pyplot as plt
K = 30
plt.figure(figsize=(12, 6))

pca = PCA().fit(df.fillna(0))

plt.bar(range(K),
        pca.explained_variance_ratio_[:K],
        alpha=0.5,
        align='center')

plt.step(range(K),
         np.cumsum(pca.explained_variance_ratio_[:K]),
         where='mid')

plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



```
In [11]: total_var = np.cumsum(pca.explained_variance_ratio_[K:])

for i in [0, 1, 2, 3, 4, 5, 10, 15, 20, 29]:
    print("Components: {:2d}, total explained variance: {:.2f}".format(i, total_var[i]))

Components:  0, total explained variance: 0.59
Components:  1, total explained variance: 0.83
Components:  2, total explained variance: 0.93
Components:  3, total explained variance: 0.95
Components:  4, total explained variance: 0.96
Components:  5, total explained variance: 0.97
Components: 10, total explained variance: 0.98
Components: 15, total explained variance: 0.99
Components: 20, total explained variance: 0.99
Components: 29, total explained variance: 1.00
```

```
In [38]: import ipywidgets as widgets
from ipywidgets import interact

from sklearn.decomposition import PCA

# use seaborn plotting style defaults
import seaborn as sns; sns.set()
```

## Face detection

```
In [13]: from sklearn.datasets import fetch_olivetti_faces

faces = fetch_olivetti_faces()
X, y = faces['data'], faces['target']
print(X.shape)

(400, 4096)
```

```
In [14]: # select 100 faces randomly
X_samples = np.random.permutation(X)[:100]

fig, axes = plt.subplots(10, 10, figsize=(12, 12))
fig.subplots_adjust(left=0.2, bottom=0.2, right=0.8, top=0.8, hspace=0.01, wspace=0.01)

for i, ax in enumerate(axes.flat):
    ax.imshow(X_samples[i].reshape((64, 64)), cmap='gray')
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```



```
In [15]: def update_pca_face_plot(i, n_components):
    pca = PCA(n_components).fit(X)
    im = pca.inverse_transform(pca.transform(X[i:i+1]))

    fig = plt.figure(figsize=(12, 6))
    plt.subplot(121)
    plt.imshow(im.reshape((64, 64)), cmap='gray')
    total_var = pca.explained_variance_ratio_.sum()
    plt.title('Approximated Data {:.2f}'.format(total_var))
    plt.axis('off')

    plt.subplot(122)
    plt.imshow(X[i].reshape((64, 64)), cmap='gray')
    plt.title('Original Data')
    plt.axis('off')
    plt.show()

idx = widgets.IntSlider(value=32, min=0, max=400, desc='data')
widgets.interact(update_pca_face_plot, i=idx, n_components=[10, 50, 100, 150, 200, 300, 400]);
interact(children=(IntSlider(value=32, description='i', max=400), Dropdown(description='n_components', opti...
```

```
In [16]: nside=5
X_samples = np.random.permutation(X)[::nside ** 2]

def plot_faces(n_components):
    global X_samples

    fig = plt.figure(figsize=(12, 6))
    pca = PCA(n_components).fit(X)
    X_proj = pca.inverse_transform(pca.transform(X_samples))
    X_proj = np.reshape(X_proj, (nside, nside, 64, 64))
    total_var = pca.explained_variance_ratio_.sum()

    plt.subplot(121)
    im = np.vstack([np.hstack([X_proj[i, j] for j in range(nside)]) for i in range(nside)])
    plt.imshow(im, cmap='gray')
    plt.axis('off')
    plt.title("={0}, variance = {:.2f}".format(n_components, total_var), size=18)

    plt.subplot(122)
    X_org = np.reshape(X_samples, (nside, nside, 64, 64))
    im = np.vstack([np.hstack([X_org[i, j] for j in range(nside)]) for i in range(nside)])
    plt.imshow(im, cmap='gray')
    plt.axis('off')
    plt.title("Original Faces", size=18)

interact(plot_faces, n_components=[10, 20, 30, 40, 50, 100, 150, 200]);
interactive(children=(Dropdown(description='n_components', options=(10, 20, 30, 40, 50, 100, 150, 200), value=...
```

```
In [17]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import glob
from PIL import Image

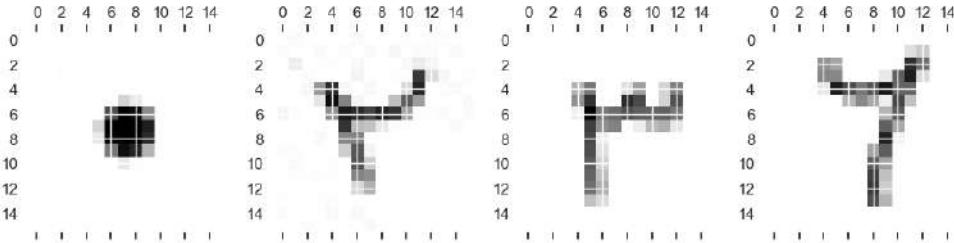
In [22]: data_dir= r'D:\AI_Machinlearning\datasets\persianDigits'

In [23]: trn_fnames=glob.glob(f'{data_dir}/*/*.jpg')

In [24]: import matplotlib.pyplot as plt

#plt.gray()
fig , (ax1 ,ax2 ,ax3 , ax4)=plt.subplots(nrows=1 , ncols=4)
fig.set_size_inches(10, 8)
ax1.matshow(plt.imread(trn_fnames[0]))           #plt.imshow
ax2.matshow(plt.imread(trn_fnames[500]))
ax3.matshow(plt.imread(trn_fnames[1200]))
ax4.matshow(plt.imread(trn_fnames[2000]))

plt.tight_layout()
plt.show()
```



```
In [26]: def load_images_and_labels(folder, target_size=(16, 16)):
    images = []
    labels = []
    for label in os.listdir(folder):
        label_path = os.path.join(folder, label)
        if os.path.isdir(label_path):
            for filename in os.listdir(label_path):
                img_path = os.path.join(label_path, filename)
                if os.path.isfile(img_path):
                    img = Image.open(img_path).convert('L') # Convert to grayscale
                    img = img.resize(target_size) # Resize image
                    img_array = np.array(img).flatten() # Flatten image to 1D array
                    images.append(img_array)
                    labels.append(label) # Add the label
    return np.array(images), np.array(labels)

def save_images_and_labels_to_csv(X, y, csv_file):
    # Convert the Numpy arrays to a DataFrame
    df = pd.DataFrame(X)
    df['label'] = y # Add the Labels column

    # Save DataFrame to CSV
    df.to_csv(csv_file, index=False)
    return df

folder_path = r'D:\AI_Machinlearning\datasets\persianDigits'
target_size = (16, 16) # target size

# Load images and Labels
X, y = load_images_and_labels(folder_path, target_size)

# Specify the path where you want to save the CSV file
csv_file = 'image_data.csv'

# Save to CSV
df = save_images_and_labels_to_csv(X, y, csv_file)
```

df

```
Out[26]:   0   1   2   3   4   5   6   7   8   9   ... 247  248  249  250  251  252  253  254  255  label
  0  255  255  255  255  255  255  255  255  255  ... 254  255  255  255  255  255  254  255  254  0
  1  255  255  254  255  255  255  254  254  255  ... 255  255  255  255  255  254  255  255  255  0
  2  255  255  255  255  255  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  0
  3  255  255  255  255  255  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  0
  4  255  254  255  255  255  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  0
  ...
  2555 255  255  255  255  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  9
  2556 255  255  255  254  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  9
  2557 255  255  255  254  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  9
  2558 255  255  254  255  255  255  255  255  ... 255  255  255  255  255  255  255  255  255  9
  2559 254  255  255  255  255  254  255  255  ... 255  255  255  255  255  255  255  255  255  9
```

2560 rows × 257 columns

```
In [27]: X=np.array(df.iloc[:, :-1])
X.shape
```

Out[27]: (2560, 256)

```
In [33]: model=Pca(X , 0.80)

sigma , u , s , u_reduced , x_proj , X_recovred , k =model.train()

print("sigma" , sigma)
print("-----")
print("u" , u)
print("-----")
print("s" , s)
print("-----")
print("u_reduced" , u_reduced)
print("-----")
print("u_reduced norm " , (np.linalg.norm(u_reduced[0])))
print("-----")
print((round(np.dot(u[0], u[1]))))
print("-----")
print(" x_proj" , x_proj[: 10])
print("-----")
print(" X_recovred" , X_recovred[: 10])

print(k)
```

```

sigma [[ 204 138 219 ... 191 95 25]
 [138 246 21 ... 240 7 149]
 [219 21 223 ... 112 149 47]
 ...
 [191 240 112 ... 111 57 188]
 [ 95 7 149 ... 57 168 7]
 [ 25 149 47 ... 188 7 229]]
u [[ -0.06224348  0.00016111  0.00078766 ... -0.05455833  0.07438076
 -0.05450439]
 [-0.06362314 -0.0392145 -0.06820786 ... -0.03068183  0.04483247
  0.03816265]
 [-0.06335127 -0.05946045  0.00891003 ...  0.04593194 -0.01929073
 -0.02897158]
 ...
 [-0.06310405  0.04649026  0.00779145 ...  0.01840761 -0.00654749
  0.04261371]
 [-0.06117475  0.01910422  0.02258302 ...  0.00951285 -0.00615244
  0.05182392]
 [-0.05936795  0.03028897 -0.03454679 ... -0.08188718 -0.09580759
 -0.04213891]]
s [[ 3.26241465e+04 2.29718978e+03 2.29293669e+03 2.26479759e+03
 2.24661886e+03 2.21882492e+03 2.20824444e+03 2.19973932e+03
 2.19975644e+03 2.15923457e+03 2.149493904e+03 2.11929801e+03
 2.08249939e+03 2.07912361e+03 2.06985406e+03 2.05518530e+03
 2.02635398e+03 2.02157516e+03 2.00255435e+03 1.99786060e+03
 1.99589929e+03 1.97137397e+03 1.95309572e+03 1.94696946e+03
 1.93131567e+03 1.91955445e+03 1.90663941e+03 1.89400751e+03
 1.88983358e+03 1.88686808e+03 1.847011157e+03 1.84556762e+03
 1.83764294e+03 1.82780459e+03 1.82553935e+03 1.80581353e+03
 1.80063254e+03 1.77984122e+03 1.75933802e+03 1.75631812e+03
 1.73924331e+03 1.72661434e+03 1.71456409e+03 1.70757100e+03
 1.69382974e+03 1.68898109e+03 1.68673463e+03 1.66208990e+03
 1.65278936e+03 1.64480889e+03 1.64035757e+03 1.61664330e+03
 1.61485180e+03 1.61247590e+03 1.59579546e+03 1.59115392e+03
 1.57774602e+03 1.564093506e+03 1.56368326e+03 1.55859711e+03
 1.54043632e+03 1.53775421e+03 1.52173421e+03 1.52887884e+03
 1.51910768e+03 1.50170572e+03 1.49203597e+03 1.47858400e+03
 1.47575891e+03 1.47295652e+03 1.45384372e+03 1.44806180e+03
 1.44561286e+03 1.41546327e+03 1.40574573e+03 1.38913327e+03
 1.38609963e+03 1.38486380e+03 1.38231338e+03 1.38224015e+03
 1.36678551e+03 1.35048175e+03 1.35026517e+03 1.33616412e+03
 1.31911589e+03 1.31542251e+03 1.31257508e+03 1.30961229e+03
 1.28757227e+03 1.28546148e+03 1.24943662e+03 1.24327649e+03
 1.24206974e+03 1.23280687e+03 1.21785785e+03 1.21209223e+03
 1.20387548e+03 1.19147719e+03 1.18874405e+03 1.17938083e+03
 1.17549488e+03 1.15039710e+03 1.139828258e+03 1.13640069e+03
 1.13590563e+03 1.13514709e+03 1.12591988e+03 1.11857801e+03
 1.11526046e+03 1.10060968e+03 1.09428866e+03 1.08365816e+03
 1.07241645e+03 1.059808185e+03 1.049922119e+03 1.04779434e+03
 1.04420486e+03 1.04032793e+03 1.02851080e+03 1.01468942e+03
 1.006464668e+03 1.00233546e+03 9.96931281e+02 9.89907514e+02
 9.84284665e+02 9.77647170e+02 9.50959199e+02 9.46881500e+02
 9.39762114e+02 9.29087164e+02 9.29017858e+02 9.23207057e+02
 9.19208940e+02 9.04264233e+02 8.91468524e+02 8.87496887e+02
 8.78663133e+02 8.736233237e+02 8.66223987e+02 8.60245651e+02
 8.47927361e+02 8.304323495e+02 8.24576549e+02 8.16584393e+02
 8.09948496e+02 8.05056519e+02 8.02469869e+02 7.98766286e+02
 7.82388903e+02 7.78718560e+02 7.72677485e+02 7.66685783e+02
 7.51479231e+02 7.46788844e+02 7.41053448e+02 7.35231048e+02
 7.31502179e+02 7.18183163e+02 7.147293937e+02 7.09485347e+02
 6.94222850e+02 6.87396664e+02 6.84498110e+02 6.78697325e+02
 6.64037665e+02 6.61313980e+02 6.617490854e+02 6.49804611e+02
 6.38402770e+02 6.34286827e+02 6.25605871e+02 6.23159499e+02
 6.05997269e+02 5.96495725e+02 5.91054597e+02 5.83014696e+02
 5.81713167e+02 5.72130158e+02 5.67530450e+02 5.65258505e+02
 5.56582862e+02 5.509651183e+02 5.40722259e+02 5.28269621e+02
 5.22096741e+02 5.12723619e+02 5.09588157e+02 4.97519186e+02
 4.93278726e+02 4.90143838e+02 4.81467243e+02 4.61186878e+02
 4.51093743e+02 4.49017266e+02 4.33954835e+02 4.33073771e+02
 4.21129985e+02 4.19687614e+02 4.13420880e+02 4.12916385e+02
 4.07874857e+02 3.900845183e+02 3.89381430e+02 3.87459269e+02
 3.78374512e+02 3.69288618e+02 3.46536930e+02 3.44725828e+02
 3.34845205e+02 3.32984538e+02 3.28447556e+02 3.13371052e+02
 3.12779038e+02 3.08087844e+02 2.89831685e+02 2.82868684e+02
 2.79017453e+02 2.77217859e+02 2.70716878e+02 2.55841215e+02
 2.53506253e+02 2.48831768e+02 2.38203375e+02 2.34244569e+02
 2.30170144e+02 2.25567832e+02 2.14179256e+02 2.02394187e+02
 1.93320984e+02 1.88796617e+02 1.83427317e+02 1.76129503e+02
 1.74278070e+02 1.72424267e+02 1.67043435e+02 1.51562323e+02
 1.49944441e+02 1.44086575e+02 1.29412758e+02 1.15311904e+02
 1.06449772e+02 1.03435690e+02 1.01028693e+02 9.49562656e+01
 8.84090640e+01 8.48734658e+01 7.55147542e+01 6.45710688e+01
 5.54051576e+01 4.510658847e+01 4.43758842e+01 3.80169430e+01
 2.38051635e+01 1.41655851e+01 7.50465507e+00 3.27650609e+00]
-----
u_reduced [[ -0.06224348  0.00016111  0.00078766 ...  0.0818698  0.01711341
 -0.01856192]
 [-0.06362314 -0.0392145 -0.06820786 ... -0.08451196 -0.03141886
  0.03857799]
 [-0.06335127 -0.05946045  0.00891003 ...  0.10738772 0.02101321
  0.02002478]
 ...
 [-0.06310405  0.04649026  0.00779145 ...  0.02098157 0.01376036
  0.08373767]
 [-0.06117475  0.01910422  0.02258302 ... -0.09709907 0.0548442
  0.10020372]
 [-0.05936795  0.03028897 -0.03454679 ... -0.14587906 0.02980519
  0.07388915]
 -----
u_reduced norm 0.7072225035224439
-----
0
-----
x_proj [[ -2.79617003  0.75242872 -0.61720206 ... 0.57433265 -0.04634815
 -0.61488742]
 [-3.94971879  0.81015755 -0.38508633 ... 0.29478535 -0.10671366
  0.06692359]
 [-4.13162035  0.84063788 -0.31089603 ... 0.1769393 0.05500876
  -0.10551137]
 ...
 [-2.60033774  0.99903774 -0.0361055 ... 0.44915437 -0.20774736
 -0.21451459]
 [-3.26412022  1.08370937 -0.33061882 ... 0.56680343 -0.43881399
  -0.17046816]
 [-2.28825201  1.10334816 -0.41513317 ... 0.20462311 -0.20209296
  -0.42603138]
 -----
x_recorderd [[ 228.7952362 437.75710499 1079.70303447 ... 1075.68370163
 1146.90573936 766.69417431]
 [ 228.89723031 438.09689188 1080.10953585 ... 1076.52106668
 1147.1752757 767.7528158]
 [ 229.00398866 438.14470084 1080.57886871 ... 1077.24022189
 1147.32745639 768.80405542]
 ...
 [ 228.76673779 437.68834476 1080.69731045 ... 1074.52924224
 1146.3533521 766.81705242]

```

```
[ 228.36257966 437.73896924 1080.40853183 ... 1075.79243507  
1146.91794756 767.20910954]  
[ 228.69425513 437.84826681 1080.45681447 ... 1074.93888446  
1147.2898576 767.08155677]]  
131
```

```
In [34]: df2 = pd.DataFrame(x_proj)  
df2["label"] = df.label  
df2
```

```
Out[34]:
```

	0	1	2	3	4	5	6	7	8	9	...	122	123	124	125	126	127	128	129	130	label
0	-2.796170	0.752429	-0.617220	-0.065920	-0.370488	0.213836	-0.907861	0.117052	0.489013	-0.263297	...	-1.244164	-0.976403	0.190746	-0.679698	-0.412507	-1.067522	0.574333	-0.046348	-0.614807	
1	-3.949719	0.810158	-0.385086	-0.461747	-0.749410	-0.094150	-0.579426	-0.343208	0.125834	-0.412178	...	-0.953738	-0.210647	0.140832	-0.220414	-0.333146	-0.164211	0.294785	-0.106714	0.066924	
2	-4.131620	0.840638	-0.310896	-0.065565	-0.749688	-0.465985	-0.614853	-0.211090	0.025683	-0.514596	...	-0.885561	0.062644	0.413249	-0.124577	-0.256175	-0.185641	0.176939	0.055009	-0.105511	
3	-2.855952	1.099970	-0.428147	-0.919423	-0.537525	0.241710	-0.910745	-0.049219	0.335865	-0.253700	...	-1.164072	-0.706850	-0.031255	-0.505632	-0.139579	-0.435080	0.446797	-0.316689	-0.296080	
4	-2.785128	1.034643	-0.155661	-0.787821	-0.417497	0.170134	-0.830211	0.060533	0.034681	-0.426010	...	-1.118470	-0.584962	-0.135654	-0.281771	-0.647432	-0.542592	0.416489	-0.101441	-0.086814	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...		
2555	-0.308123	0.205913	-0.838279	-1.602649	-0.374211	1.345595	0.394755	1.081843	0.181838	0.576562	...	1.110765	0.023080	0.767828	-0.034205	-0.217455	-1.035106	0.059068	0.261638	-0.165427	
2556	-2.734907	-0.175156	-0.433892	-0.569481	0.130652	0.869747	-0.279304	0.233973	-0.415603	-0.604034	...	0.416308	0.126700	0.233796	-0.188648	-0.417229	-0.289025	0.295044	-0.310089	-0.118742	
2557	-2.167324	0.951553	-0.568938	-0.346288	-0.489408	-0.456093	-0.215305	0.060365	0.235212	0.107251	...	-0.011637	-0.180718	-0.700979	-0.028229	-0.146190	0.429170	0.248354	0.235952	-0.099510	
2558	-0.872908	-0.094146	-1.094882	0.543091	0.617593	-1.346038	-0.087232	0.556069	0.691806	0.591371	...	0.612839	-0.447612	1.352365	-0.758657	-0.045461	0.454588	-0.291121	1.643131	-0.487802	
2559	-0.736626	0.368936	-0.037597	-0.806961	-0.242308	0.732926	0.732694	2.114147	-1.677166	-0.795577	...	0.956711	-1.275453	0.975928	-0.316283	-1.049855	-0.592598	0.540646	-0.655274	0.811524	

2560 rows × 132 columns

```
In [35]: X = np.array(df2.iloc[:, :-1])  
y = np.array(df2.iloc[:, -1])  
  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
Xnorm = sc.fit_transform(X)  
  
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(Xnorm, y, random_state=120, test_size=20)
```

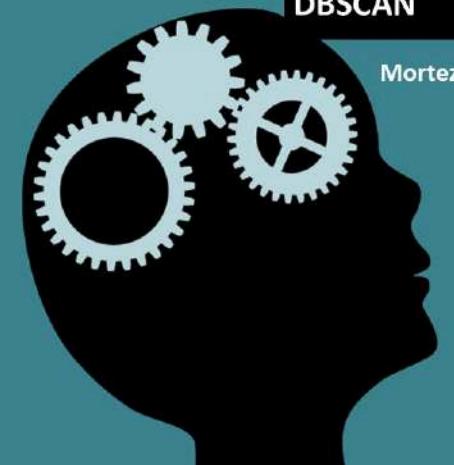
```
In [36]: import math  
from sklearn.neighbors import KNeighborsClassifier  
  
knn = KNeighborsClassifier()  
knn.fit(X_train, y_train)  
  
y_predict = knn.predict(X_test)
```

```
In [37]: #train  
for k in range(3, 16, 2):  
    model = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2, weights="distance", algorithm="kd_tree")  
    model.fit(X_train, y_train)  
    acc = model.score(X_test, y_test)  
    print("K = " + str(k) + "; Accuracy: " + str(acc))  
  
K = 3; Accuracy: 0.85  
K = 5; Accuracy: 0.8  
K = 7; Accuracy: 0.75  
K = 9; Accuracy: 0.75  
K = 11; Accuracy: 0.75  
K = 13; Accuracy: 0.85  
K = 15; Accuracy: 0.85
```

# Machine learning

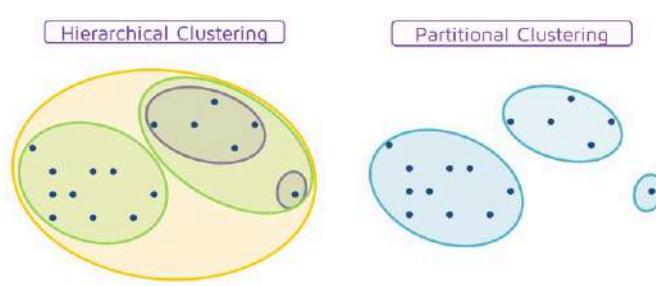
Fuzzy Clustering  
Hierarchical clustering  
DBSCAN

Morteza khorsand



## Hierarchical clustering

### Hierarchical clustering

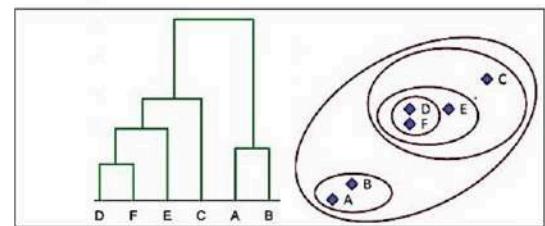
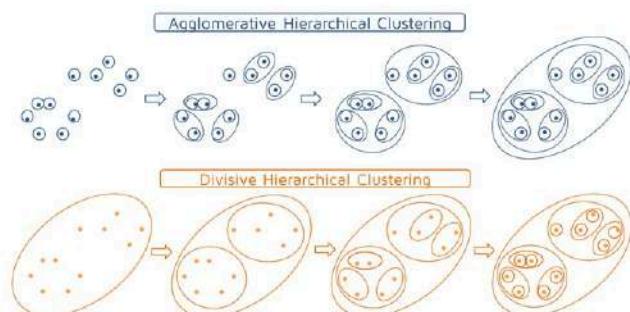


#### Agglomerative (bottom-up):

Start with each sample being a single cluster.  
Eventually all samples belong to the same cluster.

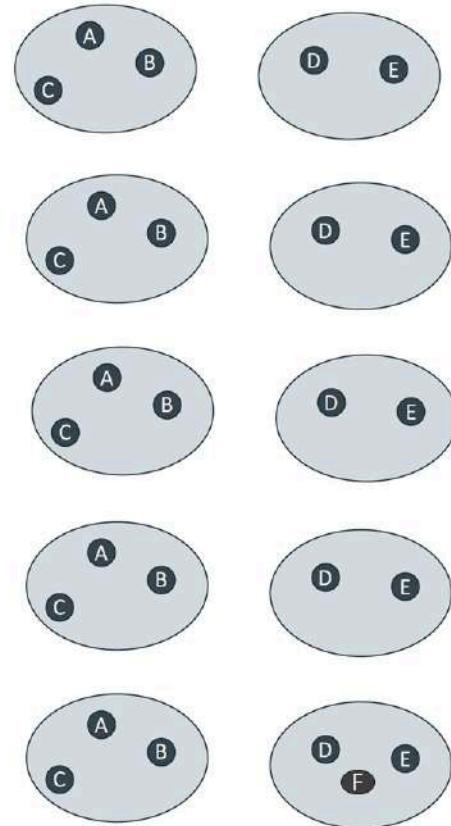
#### Divisive (top-down):

Start with all samples belong to the same cluster.  
Eventually each sample forms a cluster on its own.



## Hierarchical clustering

- **Single linkage**: smallest distance between an element in one cluster and an element in the other.
- **Complete linkage** : largest distance between an element in one cluster and an element in the other.
- **Average linkage** : avg distance between an element in one cluster and an element in the other.
- **Centroid(ward) linkage** : distance between the centroids of two clusters.
- **Medoid linkage** : distance between the medoids of two clusters.



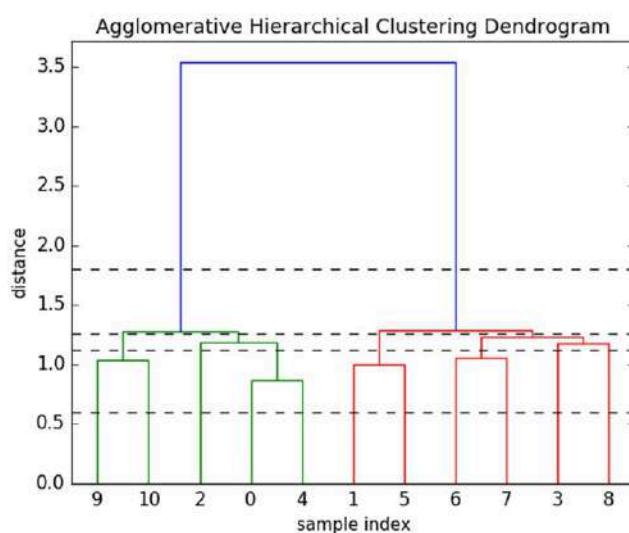
## Hierarchical clustering

Decompose data objects into several levels of nested partitioning (tree of clusters), called a **dendrogram**.

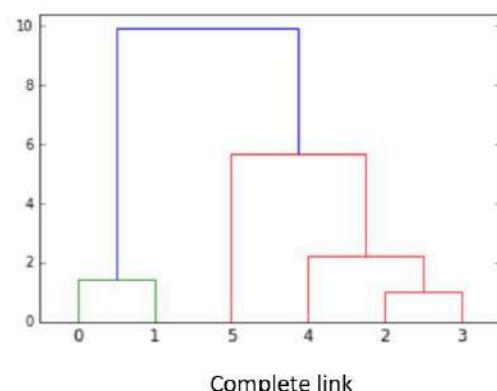
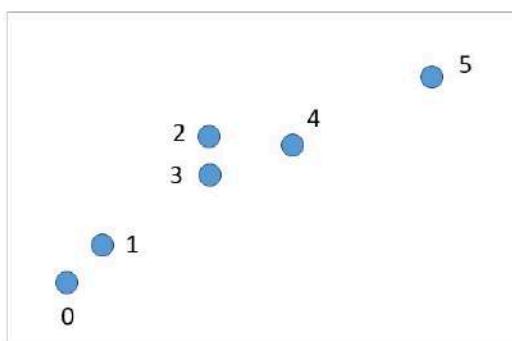
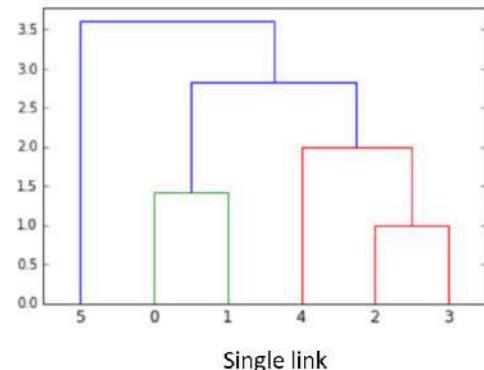
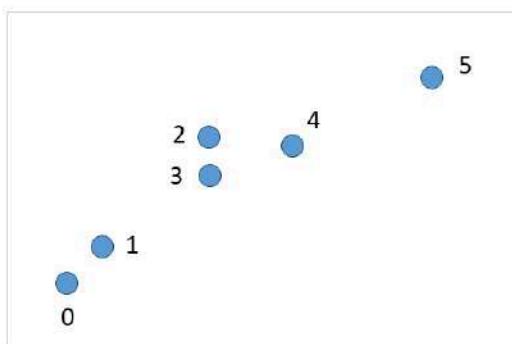
- Shows the inner structure of each cluster.

Does not require the number of clusters  $k$  in advance.

Clustering obtained by cutting the dendrogram at a desired level.

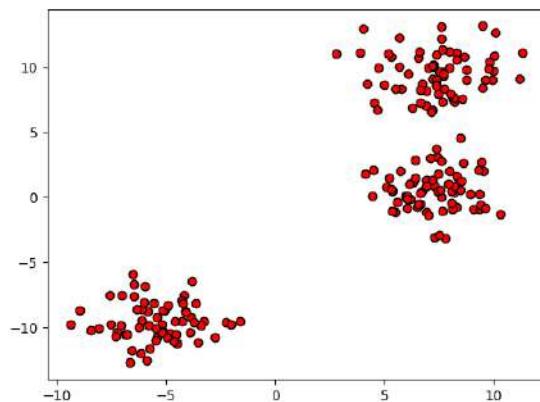


## Hierarchical clustering

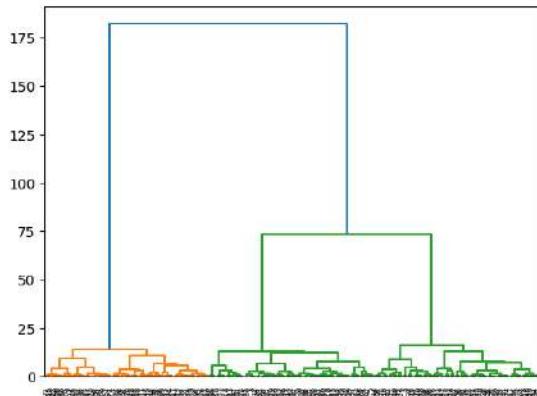


### Example1

```
In [1]:  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import AgglomerativeClustering  
  
#make dataset  
from sklearn.datasets import make_blobs  
  
X, y = make_blobs(n_samples=200, random_state=8, cluster_std=1.5)  
plt.scatter(X[:, 0], X[:, 1], c="red", edgecolors="k")  
plt.show()
```



```
In [2]:  
#dendrogram  
  
import scipy.cluster.hierarchy as sch  
  
dendrogram=sch.dendrogram(sch.linkage(X, method="ward", metric="euclidean"))  
  
#Linkage('ward', 'complete', 'average', 'single'), default='ward'
```



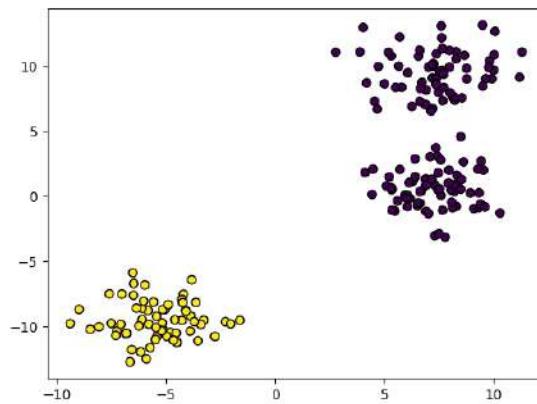
```
In [3]: cluster1 =AgglomerativeClustering( n_clusters=2, metric='euclidean' , linkage='ward')
cluster1.fit(X)
```

```
Out[3]: AgglomerativeClustering
AgglomerativeClustering(metric='euclidean')
```

```
In [4]: print(cluster1.labels_)
print(20 *("**"))
print(cluster1.n_leaves_)

[0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0
1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 0 1 0 0
1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 1
1 0 0 1 0 1 1 0 1 0 0 0 0 1]
*****
200
```

```
In [5]: plt.scatter(X[:, 0] , X[:, 1] , c= cluster1.labels_ , edgecolors="k" )
plt.show()
```



```
In [6]: #accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X , cluster1.labels_)
```

```
Out[6]: 0.7406863778705479
```

```
In [7]: cluster2 =AgglomerativeClustering( n_clusters=8, metric='euclidean' , linkage='ward')
cluster2.fit(X)
```

```
Out[7]: AgglomerativeClustering
AgglomerativeClustering(metric='euclidean', n_clusters=8)
```

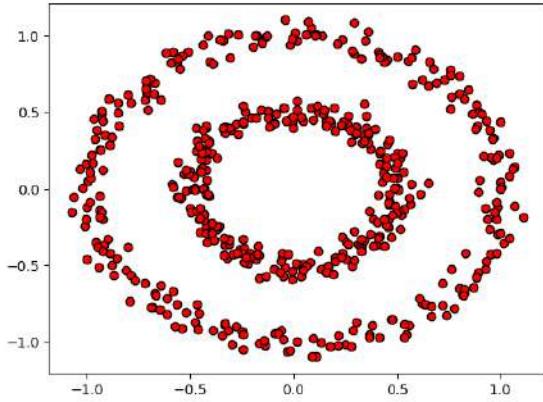
```
In [8]: labels2= cluster2.labels_
```

```
In [9]: #accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X , labels2)
```

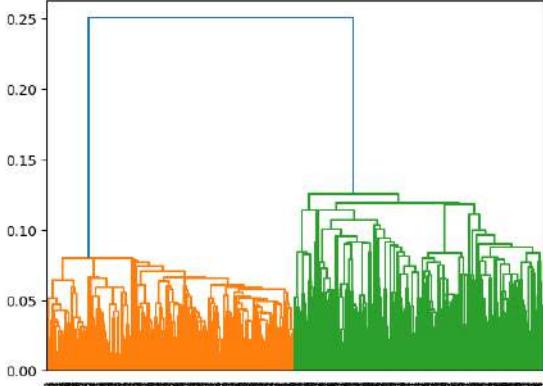
```
Out[9]: 0.3221762651982705
```

## Example 2

```
In [10]: from sklearn.datasets import make_circles
X , y = make_circles(500 , noise=0.05 , factor=0.5)
plt.scatter(X[:, 0] , X[:, 1] , c= "red" , edgecolors="k" )
plt.show()
```



```
In [11]: import scipy.cluster.hierarchy as sch  
dendrogram=sch.dendrogram(sch.linkage(X , method="single" , metric="euclidean"))
```

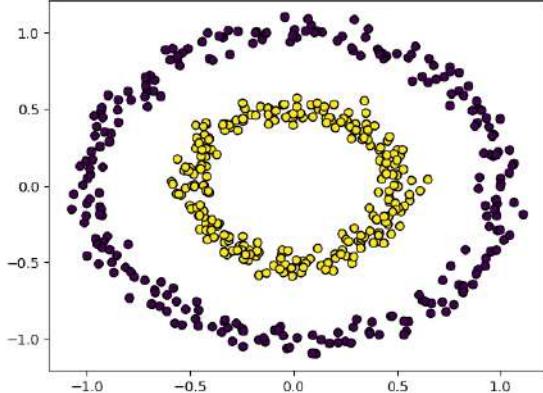


```
In [12]: cluster3=AgglomerativeClustering(n_clusters=2, linkage="single", metric="euclidean")
cluster3.fit(X)
```

```
Dut[12]: AgglomerativeClustering  
AgglomerativeClustering(linkage='single', metric='euclidean')
```

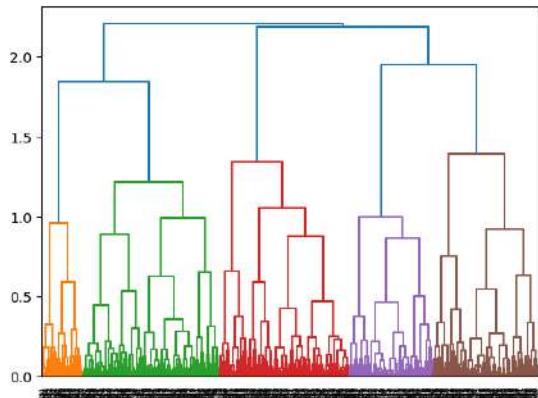
```
In [13]: cluster3.labels_
```

```
In [14]: plt.scatter(X[:, 0] , X[:, 1] , c= cluster3.labels_ , edgecolors="k" )
plt.show()
```



```
In [15]: import scipy.cluster.hierarchy as sch

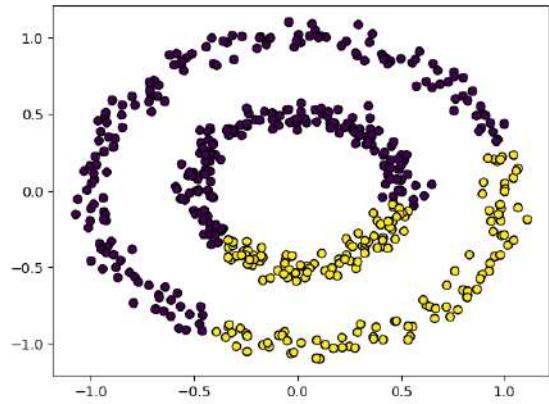
dendrogram=sch.dendrogram(sch.linkage(X , method="complete" , metric="euclidean"))
```

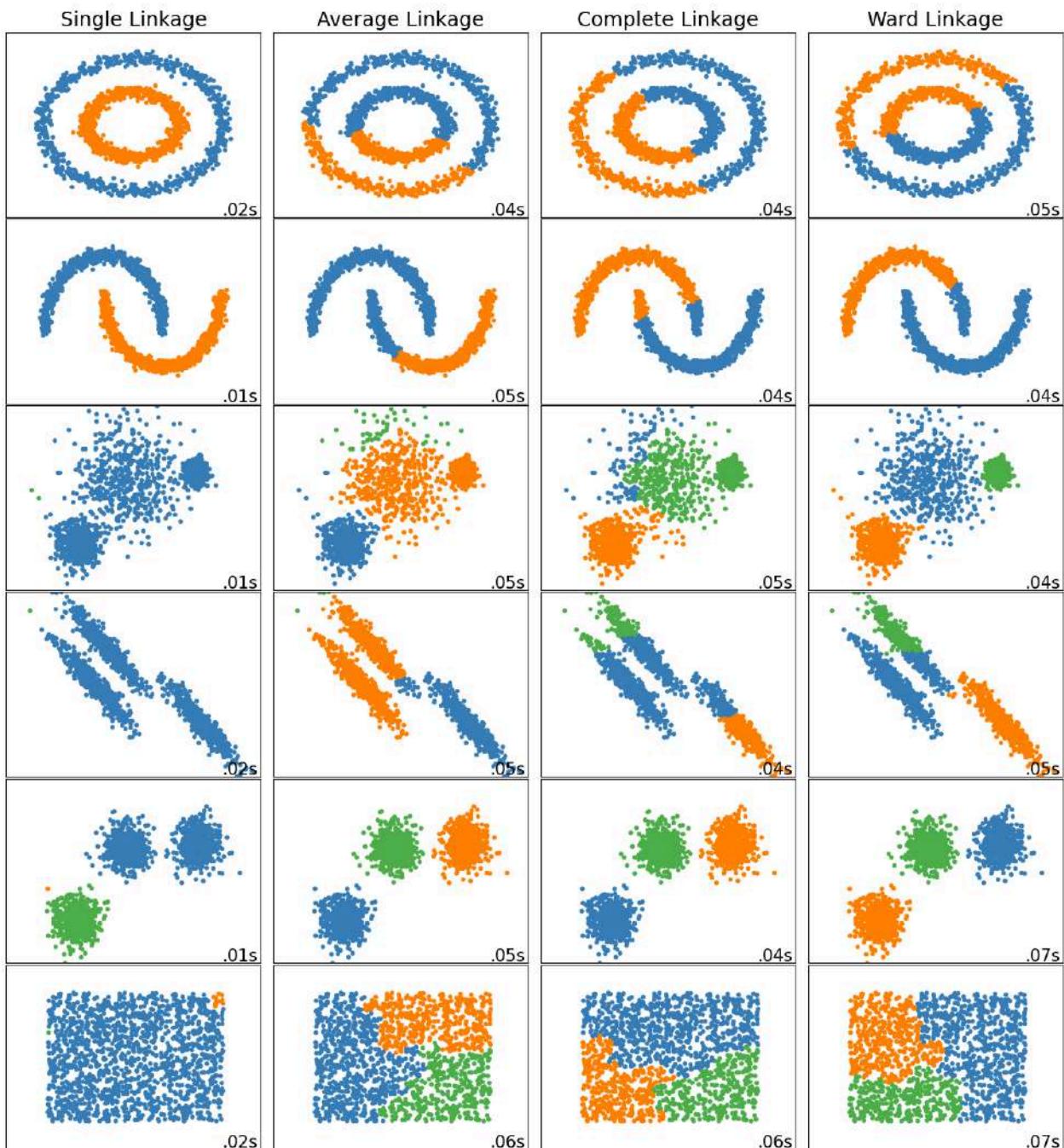


```
In [16]: cluster4=AgglomerativeClustering(n_clusters=2, linkage="complete" , metric="euclidean" )
cluster4.fit(X)
```

```
Out[16]: AgglomerativeClustering
AgglomerativeClustering(linkage='complete', metric='euclidean')
```

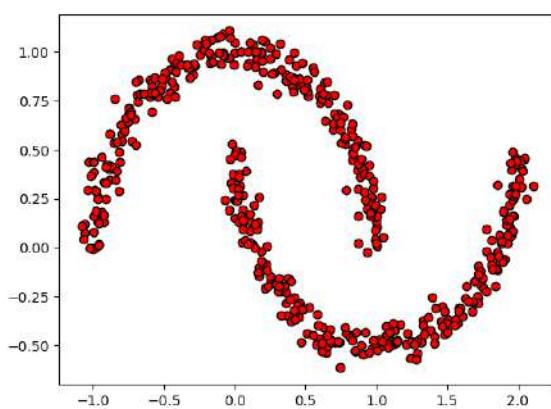
```
In [17]: plt.scatter(X[:, 0] , X[:, 1] , c= cluster4.labels_ , edgecolors="k" )
plt.show()
```



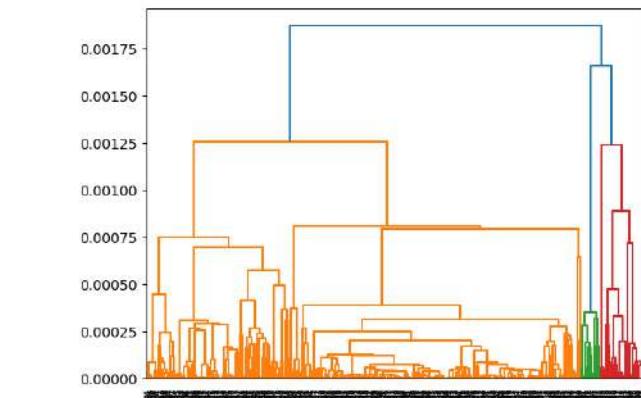


### Exemple 3

```
In [18]: from sklearn.datasets import make_moons
X , y = make_moons(500 , noise=0.05 )
plt.scatter(X[:, 0 ] , X[:, 1 ] , c= "red" , edgecolors="k" )
plt.show()
```



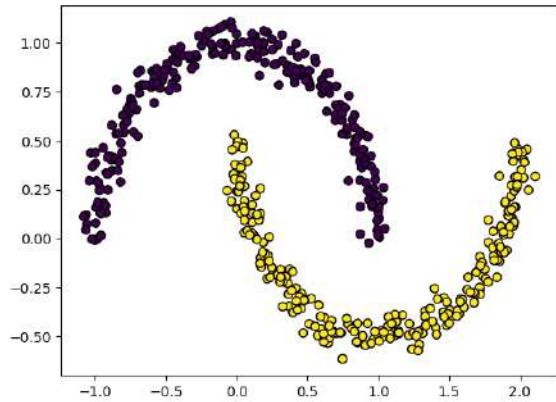
```
In [19]: from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
dendrogram=sch.dendrogram(sch.linkage(X , method="single" , metric="cosine"))
```



```
In [20]: cluster5=AgglomerativeClustering(n_clusters=2, linkage="single" , metric="manhattan" )
cluster5.fit(X)
```

```
Out[20]: AgglomerativeClustering
AgglomerativeClustering(linkage='single', metric='manhattan')
```

```
In [21]: plt.scatter(X[:, 0] , X[:, 1] , c = cluster5.labels_ , edgecolors="k" )
plt.show()
```



## Fuzzy Clustering

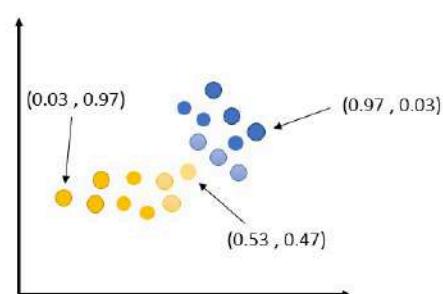
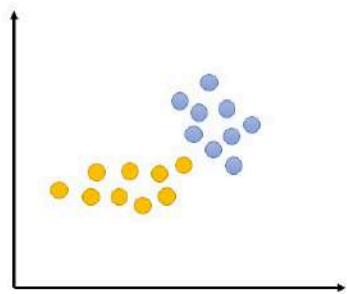
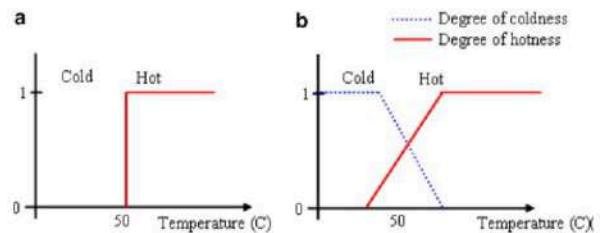
### ■ Hard Clustering

Every object belongs to exactly one cluster.

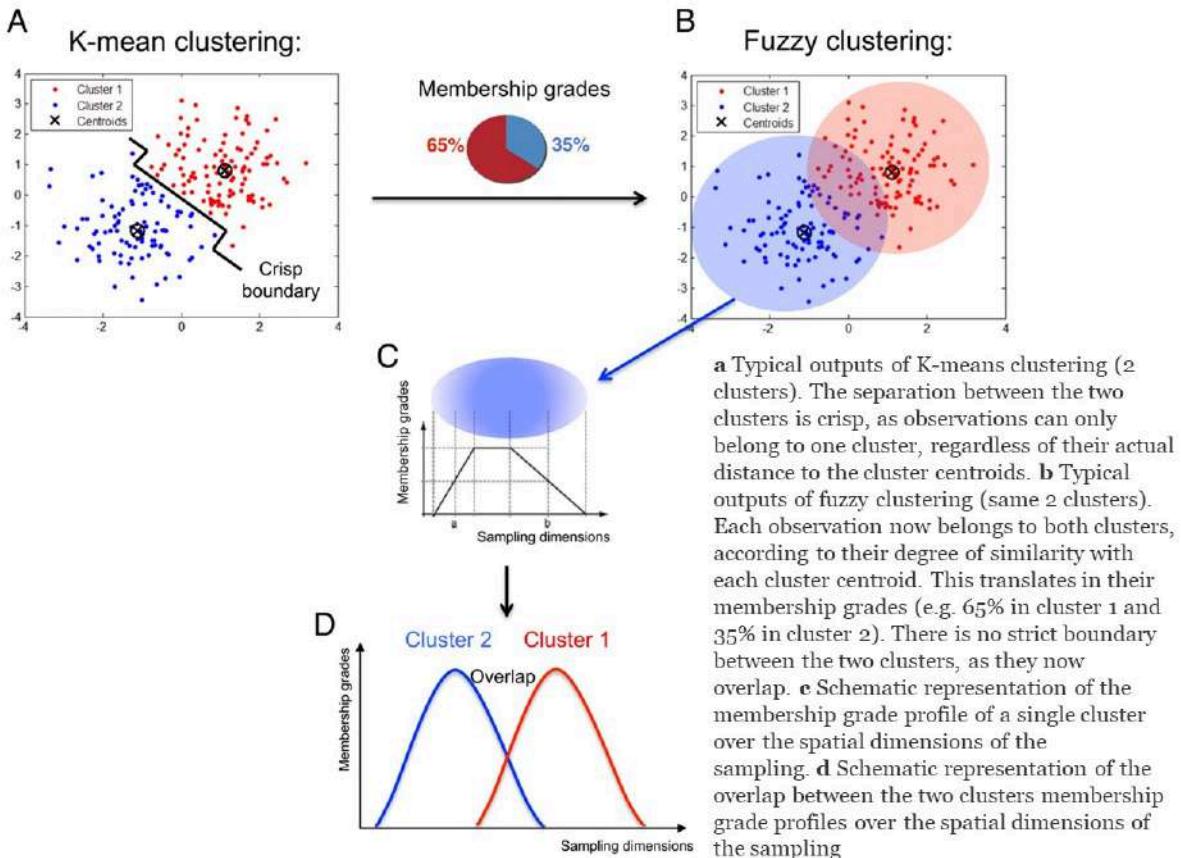
### ■ Soft Clustering

Objects may belong to several clusters with fractional degree of membership in each.

#### ▪ Fuzzy Clustering



## Fuzzy Clustering



## Fuzzy Clustering

**Step 1:** Specify the number of clusters and assign randomly points as center of clusters

**Step 2:** For each point compute its coefficients according to each cluster

partitioned matrix

$$U_{i,j} = \frac{1}{\sum \left( \frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

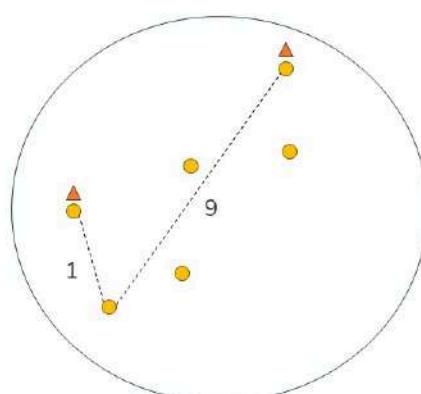
$$\begin{aligned} m &: \text{fuzzifier } 1 < m < \infty \\ 0 &\leq U_{i,j} \leq 1 \end{aligned}$$

$m=2$

For the first sample

$$U_{1,1} = \frac{1}{\left(\frac{1}{1}\right)^{\frac{2}{2-1}} + \left(\frac{1}{9}\right)^{\frac{2}{2-1}}} = 98.78$$

$$U_{1,2} = \frac{1}{\left(\frac{9}{1}\right)^{\frac{2}{2-1}} + \left(\frac{9}{9}\right)^{\frac{2}{2-1}}} = 1.22$$

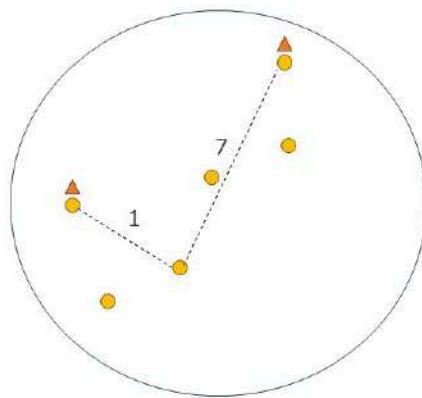


## Fuzzy Clustering

For the second sample

$$U_{2,1} = \frac{1}{\left(\frac{1}{1}\right)^{\frac{2}{2-1}} + \left(\frac{1}{7}\right)^{\frac{2}{2-1}}} = 98.00$$

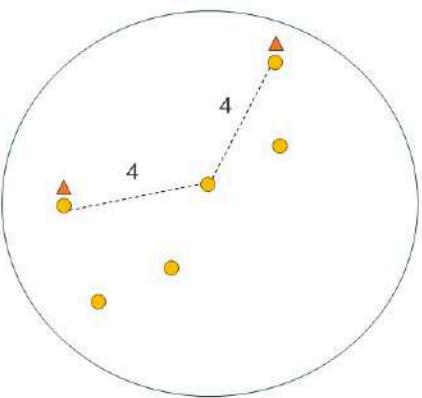
$$U_{2,2} = \frac{1}{\left(\frac{7}{1}\right)^{\frac{2}{2-1}} + \left(\frac{7}{7}\right)^{\frac{2}{2-1}}} = 2.00$$



For the next sample

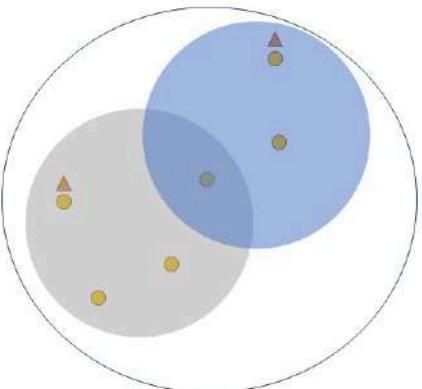
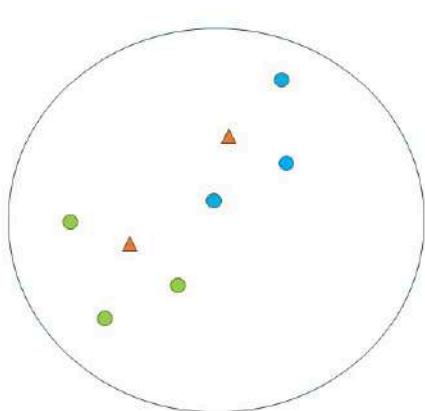
$$U_{3,1} = \frac{1}{\left(\frac{4}{4}\right)^{\frac{2}{2-1}} + \left(\frac{4}{4}\right)^{\frac{2}{2-1}}} = 50.00$$

$$U_{3,2} = \frac{1}{\left(\frac{4}{4}\right)^{\frac{2}{2-1}} + \left(\frac{4}{4}\right)^{\frac{2}{2-1}}} = 50.00$$



## Fuzzy Clustering

First Step result



Step 3: Compute the centroid of each cluster

$$c_j = \frac{\sum U_{i,j}^m \times x_i}{\sum U_{i,j}^m}$$

$$c_1 = \frac{(98.78)^2 \times x_1 + (98)^2 \times x_1 + (50)^2 \times x_1 + (100)^2 \times x_1}{(98.78)^2 + (98)^2 + (50)^2 + (100)^2}$$

Step 4: repeat 2,3 until convergence

!pip install scikit-fuzzy

```
skfuzzy.cmeans(data(n,m), c, m, error, maxiter, init=None, seed=None)
```

## Fuzzy C-Means has a known problem with high dimensionality datasets

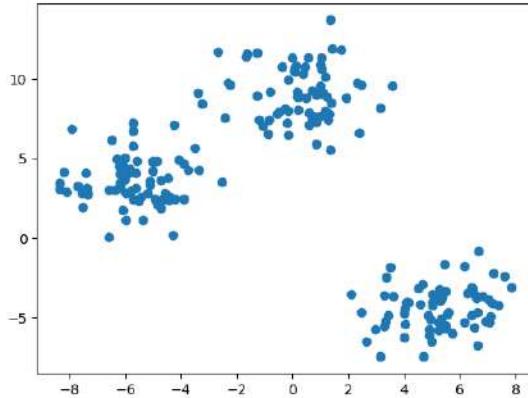
### Example1

```
In [22]: import numpy as np
import skfuzzy as fuzz

import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate some example data
X, y = make_blobs(n_samples=200, random_state=23, n_features=2, centers=3, cluster_std=1.5)

plt.scatter(X[:, 0], X[:, 1])
plt.show()
```



```
In [23]: n_clusters = 3

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X.T, n_clusters, m=2, error=0.005, maxiter=1000)

cluster_membership = np.argmax(u, axis=0)

print('Cluster Centers:', cntr)
print('Cluster Membership:', cluster_membership)

#d :Final Euclidian distance matrix.
# jm : Objective function history.
# p : Number of iterations run.
#fpc: Final fuzzy partition coefficient.
```

Cluster Centers: [[-5.65658531 3.54901721  
[ 0.28811043 9.14126178]  
[ 5.2370357 -4.41419487]]  
Cluster Membership: [0 0 2 0 2 0 1 2 2 0 1 1 2 1 2 0 0 2 2 1 2 1 2 0 1 0 1 0 1 2 0 1 2 2 0 0 2  
1 0 2 0 1 0 2 0 2 1 0 2 1 0 1 2 0 0 0 1 2 0 0 1 0 2 2 1 2 2 1 0 0 1 0  
1 1 0 2 1 0 1 0 2 2 1 1 2 0 0 2 2 2 2 0 2 0 1 2 1 0 1 0 2 1 0 2 2 1 1 2 2  
1 0 1 2 2 2 1 1 0 0 1 1 2 0 1 0 1 1 0 1 2 2 0 1 2 1 1 2 0 0 1 2 2 2 2 0  
0 2 2 2 2 0 1 0 1 1 0 1 1 2 2 2 1 0 1 0 0 0 1 0 1 0 2 1 1 1 2 0 2 2  
1 0 1 2 0 0 2 0 1 2 1 2 0 2 2]

```
In [24]: import pandas as pd
df=pd.DataFrame(u.T * 100, columns= ["cluster1", "cluster2", "cluster3"])
df["result"] = cluster_membership

df
```

```
Out[24]:   cluster1  cluster2  cluster3  result
0      3.974824   4.459909   91.565267      2
1     98.014302   1.380887   0.604811      0
2     80.230982  10.396826   9.372191      0
3     1.447260   1.356444  97.196296      2
4     96.694985   2.237523   1.067491      0
...
195   11.712200  86.036020   2.251780      1
196   1.908551   1.915376  96.176073      2
197   88.513995   9.484707   2.001298      0
198   2.165748   1.612178  96.222075      2
199   2.660115   1.942071  95.397814      2
```

200 rows × 4 columns

```
In [25]: # Plot the data points with colors based on their cluster membership
plt.figure(figsize=(10, 8))

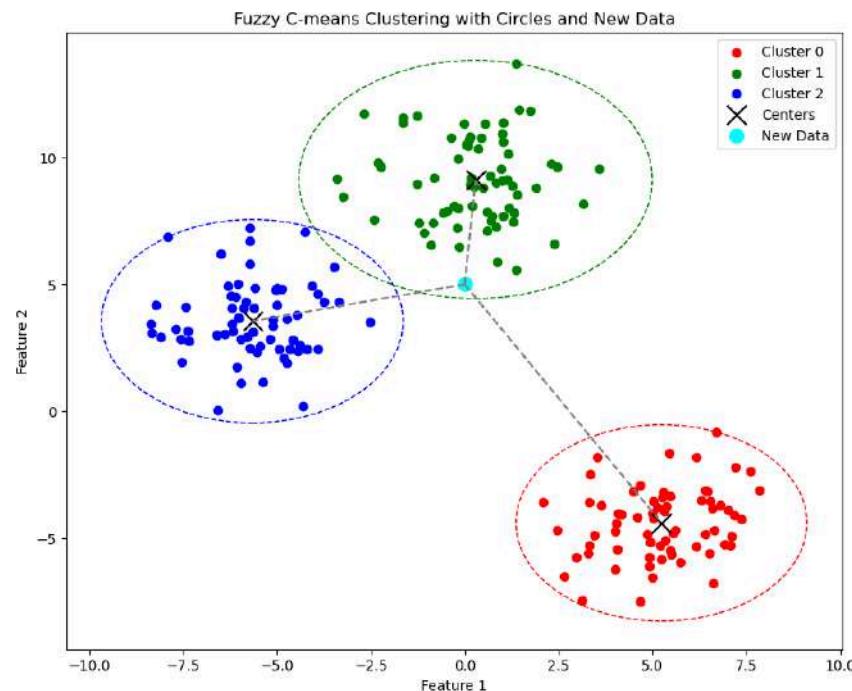
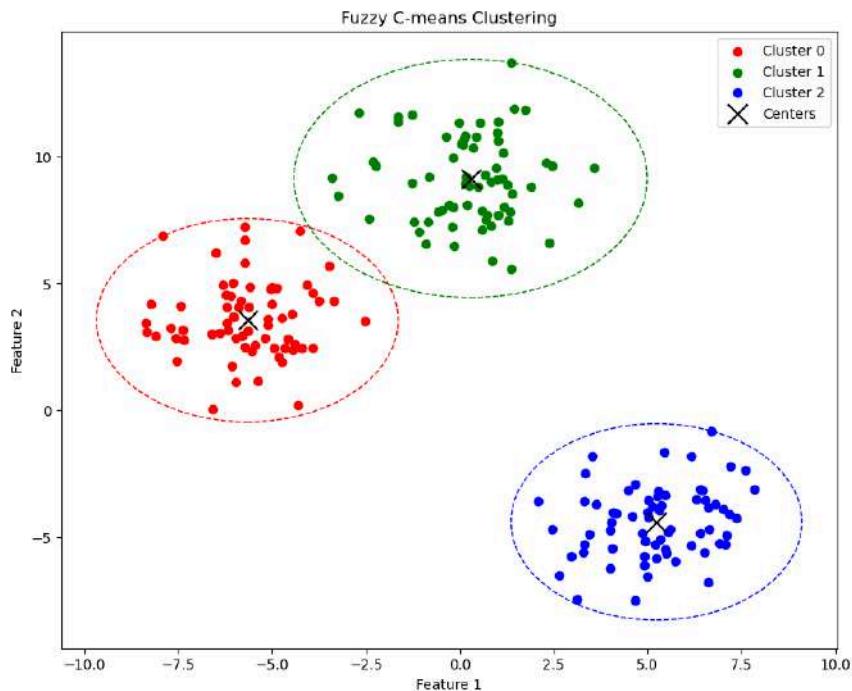
colors = ['r', 'g', 'b']
for i in range(n_clusters):
    plt.scatter(X[cluster_membership == i, 0], X[cluster_membership == i, 1], label=f'Cluster {i}', color=colors[i])

# Plot the cluster centers
plt.scatter(cntr[:, 0], cntr[:, 1], marker='x', s=200, c='black', label='Centers')

# Draw circles around clusters
for i in range(n_clusters):
    cluster_points = X[cluster_membership == i]
    radius = np.max(np.sqrt(np.sum((cluster_points - cntr[i])**2, axis=1)))
    circle = plt.Circle((cntr[i, 0], cntr[i, 1]), radius, color=colors[i], fill=False, linestyle='--')

plt.title('Fuzzy C-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

```
plt.legend()  
plt.show()
```

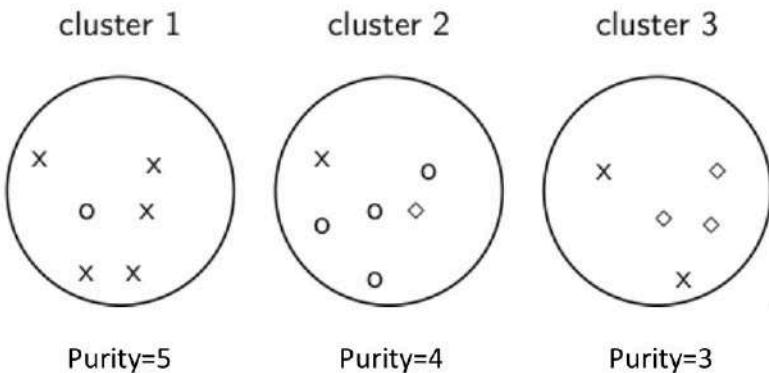


```
In [26]: def predict_new_data(new_data, cntr, m=2):  
    """  
    Predict the cluster membership and membership values for new data points.  
  
    Parameters:  
    new_data (ndarray): New data points to predict.  
    cntr (ndarray): Cluster centers.  
    m (float): Fuzziness parameter.  
  
    Returns:  
    ndarray: Membership values for each new data point.  
    """  
    # Calculate the distance between new data points and cluster centers  
    dis = np.linalg.norm(new_data - cntr, axis=1)  
  
    cluster_id = np.argmin(dis)  
  
    # Calculate the membership values  
    u_new = 1.0 / np.sum((dis[:, np.newaxis] / dis[np.newaxis, :]) ** (2 / (m - 1)), axis=1)  
  
    return u_new, cluster_id
```

```
In [27]: new_data=np.array([[0,5]])  
predict_new_data(new_data, cntr, m=2)  
Out[27]: (array([0.30555449, 0.60465845, 0.08978707]), 1)
```

## Purity

External accuracy : Purity (homogeneity\_score)



$$\text{Purity is } (1/17) \times (5 + 4 + 3) \approx 0.71$$

$$0 \leq \text{Purity} \leq 1$$

## External accuracy

```
In [28]: #accuracy score
from sklearn.metrics import homogeneity_score
homogeneity_score(cluster_membership, y)

Out[28]: 0.9999999999999998
```

## Internal accuracy

```
In [29]: from sklearn.metrics import silhouette_score
silhouette_score(X, cluster_membership, metric="euclidean")

Out[29]: 0.7223682972716525
```

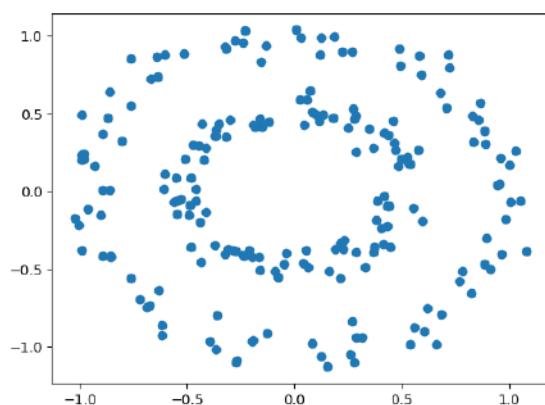
## Example3

```
In [30]: import numpy as np
import skfuzzy as fuzz

import matplotlib.pyplot as plt
from sklearn.datasets import make_circles

# Generate some example data
X, y = make_circles(n_samples=200, random_state=12, noise=0.07, factor=0.5)

plt.scatter(X[:, 0], X[:, 1])
plt.show()
```



```
In [31]: n_clusters = 2
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X.T, n_clusters, m=2, error=0.005, maxiter=1000)
```

```

cluster_membership = np.argmax(u, axis=0)

print('Cluster Centers:', cntr)
print('Cluster Membership:', cluster_membership)
Cluster Centers: [[ 0.23910063  0.39214814]
 [ -0.23610324 -0.38154764]]
Cluster Membership: [0 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 1 0 1
 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1
 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 1 0 1 0 0 0 1 1
 1 0 0 1 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0 0 0 0
 1 0 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 0 0
 1 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0]
```

```
In [32]: import pandas as pd
df2=pd.DataFrame(u.T * 100, columns= ["cluster1" , "cluster2" ])
df2["result"] = cluster_membership
df2
```

```
Out[32]:
   cluster1  cluster2  result
0    87.264488  12.735512      0
1    11.809759  88.190241      1
2     5.233871  94.766129      1
3    84.885424  15.114576      0
4    10.895230  89.104770      1
...
195   75.763900  24.236100      0
196   84.440832  15.559168      0
197   65.865269  34.134731      0
198   11.777823  88.222177      1
199   19.346912  80.653088      1
```

200 rows × 3 columns

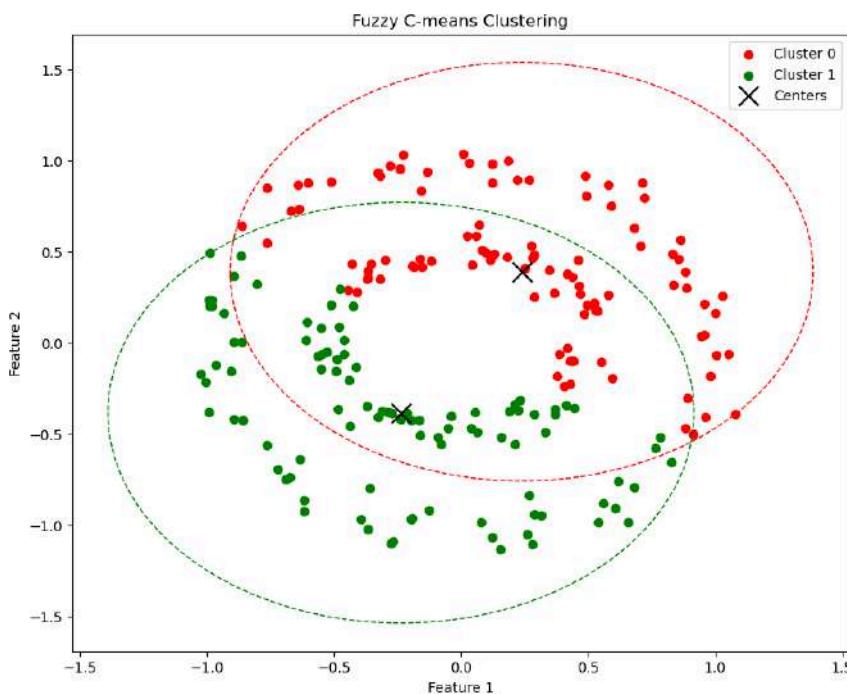
```
# Plot the data points with colors based on their cluster membership
plt.figure(figsize=(10, 8))

colors = ['r', 'g', 'b']
for i in range(n_clusters):
    plt.scatter(X[cluster_membership == i, 0], X[cluster_membership == i, 1], label=f'Cluster {i}', color=colors[i])

# Plot the cluster centers
plt.scatter(cntr[:, 0], cntr[:, 1], marker='x', s=200, c='black', label='Centers')

# Draw circles around clusters
for i in range(n_clusters):
    cluster_points = X[cluster_membership == i]
    radius = np.max(np.sqrt(np.sum((cluster_points - cntr[i])**2, axis=1)))
    circle = plt.Circle((cntr[i, 0], cntr[i, 1]), radius, color=colors[i], fill=False, linestyle='--')
    plt.gca().add_patch(circle)

plt.title('Fuzzy C-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



```
In [34]: #External accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(cluster_membership, y)
```

Out[34]: 0.0002885582471889403

```
In [35]: #internal accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X, cluster_membership , metric="euclidean" )
```

Out[35]: 0.3503716916612892

### Example3

```
In [36]: from sklearn.datasets import load_iris  
iris=load_iris()  
X , y =iris.data , iris.target
```

```
In [37]: n_clusters = 3

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X.T, n_clusters, m=2,error=0.005, maxiter=1000)

cluster_membership = np.argmax(u, axis=0)

print('Cluster Centers:', cntr)
print('Cluster Membership:', cluster_membership)
```

```
In [38]: import pandas as pd  
df3=pd.DataFrame(np.random.randint(0, 100, size=(100, 3)), columns= ["cluster1" , "cluster2" , "cluster3" ])  
df3["result"] = cluster_membership  
  
df3
```

Dut[38]:	cluster1	cluster2	cluster3	result
0	99.662364	0.107191	0.230445	0
1	97.583729	0.750176	1.666096	0
2	97.981598	0.641704	1.376698	0
3	96.740724	1.011253	2.248023	0
4	99.446999	0.176789	0.376212	0
...	...	...	...	...
145	1.123635	88.277317	10.599047	1
146	2.579648	46.772269	50.648083	2
147	1.207297	83.229659	15.563044	1
148	2.155860	78.984947	18.859193	1
149	2.693124	39.231299	58.075577	2

150 rows × 4 columns

```
In [39]: #External accuracy
from sklearn.metrics import homogeneity_score

homogeneity_score(cluster_membership, y)
```

Out[39]: 0.7542594807722846

```
In [40]: #internal accuracy
from sklearn.metrics import silhouette_score
silhouette_score(X, cluster_membership , metric="euclidean" )
```

Out[40]: 0.549517512647162

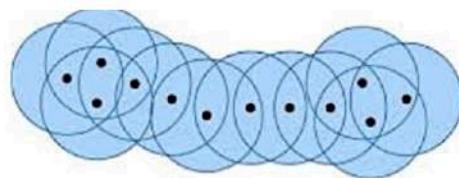
## DBSCAN

## DBSCAN

### DBSCAN

#### Major features:

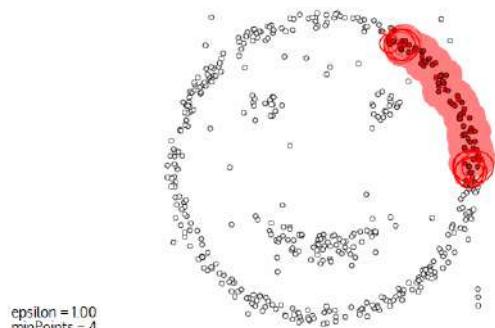
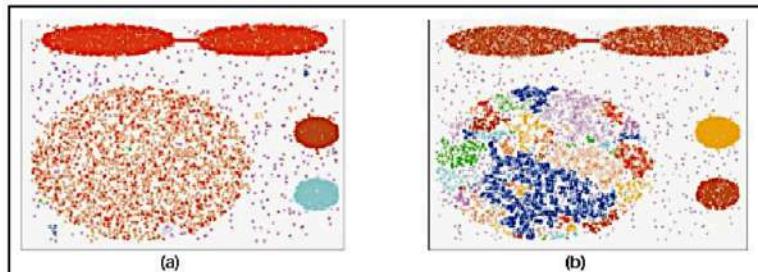
- Discover clusters of arbitrary shape
- Handle noise
- One scan
- Need density parameters
- Several interesting studies:



#### Parameters of DBSCAN:

- Eps: Maximum radius of the neighborhood.
- MinPts: Minimum number of points in an Eps neighbourhood of that point.

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.



Restart

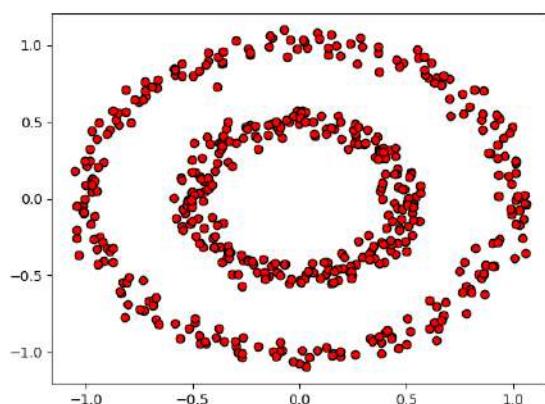
Pause

```
In [41]: from sklearn.cluster import DBSCAN
import numpy as np
import matplotlib.pyplot as plt

#make dataset
from sklearn.datasets import make_circles

X , y = make_circles(500 , noise=0.05 , factor=0.5)

plt.scatter(X[:, 0] , X[:, 1] , c= "red" , edgecolors="k" )
plt.show()
```

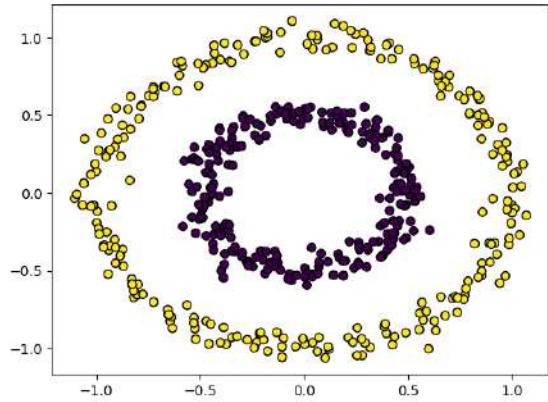


```
In [42]: DBclustering = DBSCAN(eps=0.2,min_samples=3 , metric='euclidean')
DBclustering.fit(X)
```

Out[42]: DBSCAN  
DBSCAN(eps=0.2, min\_samples=3)

```
In [43]: DBclustering.labels_
```

```
In [4]: plt.scatter(X[:, 0], X[:, 1], c=DBclustering.labels_, edgecolors="k")
plt.show()
```

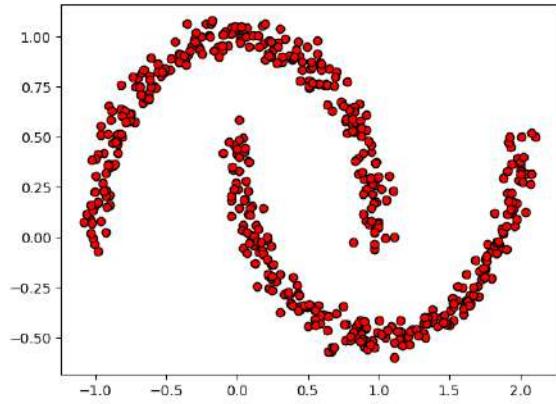


## Example

```
In [5]: #make dataset
    from sklearn.datasets import make_moons

    X , y = make_moons(500 , noise=0.05)

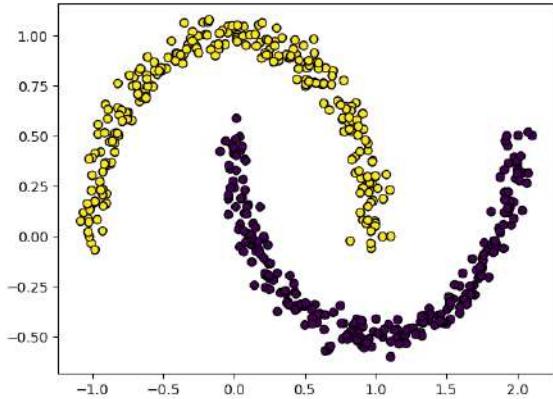
    plt.scatter(X[:, 0] , X[:, 1] , c = "red" , edgecolors="k")
    plt.show()
```



```
In [6]: DBclustering = DBSCAN(eps=0.2, min_samples=2 , metric='euclidean' )  
DBclustering.fit(X)
```

Out[6]: DBSCAN

```
In [7]: plt.scatter(X[:, 0] , X[:, 1] , c = DBclustering.labels_ , edgecolors="k" )
plt.show()
```



```
In [8]: #accuracy score
from sklearn.metrics import homogeneity_score
homogeneity_score(DBclustering.labels_, y)
```

```
Out[8]: 1.0
```

# Machine learning

## Anomaly Detection( Outliers)

Morteza khorsand



### Anomaly Detection (Outliers)

2

#### ■ Statistical population

a set of similar items or events which is of interest for some question or experiment

#### ■ sub population

A subset of a population that shares one or more additional properties is called a *sub population*.

#### population mean

The population mean, or population expected value, is a measure of the central tendency.

#### ■ Non- parametric methods

a statistical method in which the data are not assumed to come from prescribed models that are determined by a small number of parameters. examples of such as linear regression model. data can be collected from a sample that does not follow a specific distribution.

#### ■ parametric methods

assumes that sample data comes from a population that can be adequately modeled by a probability distribution that has a fixed set of parameters

## Anomaly Detection (Outliers)

4

- Gaussian Mixture Model (GMM) for Anomaly Detection (Z-score method)
  - normal distribution

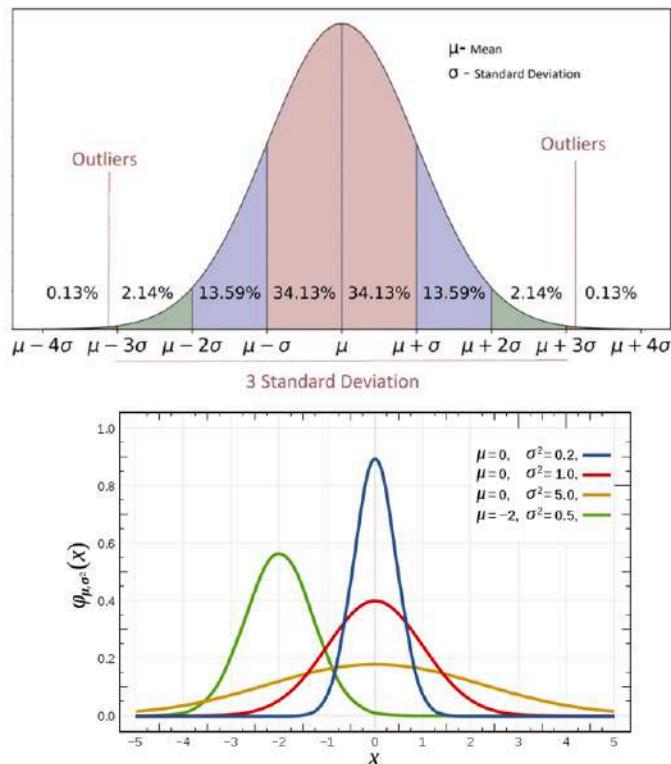
$$x \sim N(\mu, \sigma^2)$$

$$p(x; \mu, \sigma^2) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$p(\mu - \sigma < x < \mu + \sigma) = 0.6827$$

$$p(\mu - 2\sigma < x < \mu + 2\sigma) = 0.9544$$

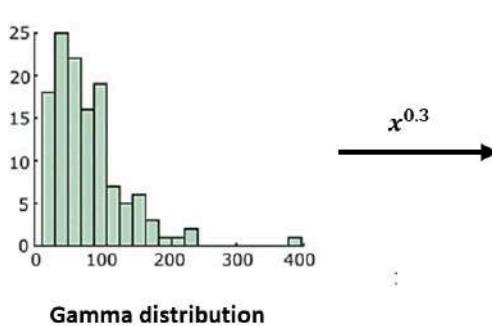
$$p(\mu - 3\sigma < x < \mu + 3\sigma) = 0.9973$$



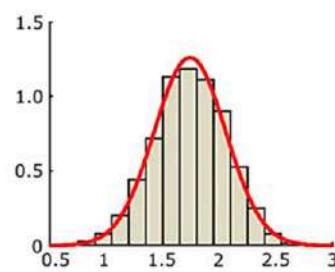
## Anomaly Detection (Outliers)

5

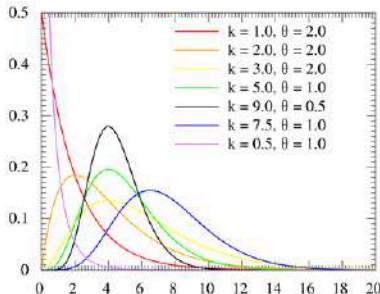
- Other distributions



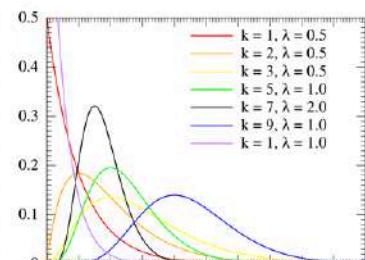
Gamma distribution



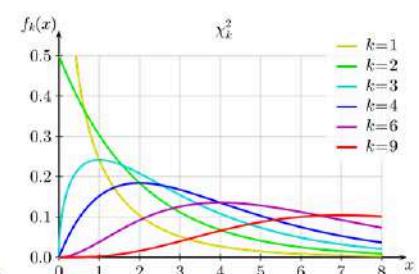
Normal distribution



Gamma distribution



Erlang distribution



Chi-squared distribution

In [1]: %matplotlib nbagg

```
import warnings
import pandas as pd
import numpy as np
import scipy as sp
```

```

import matplotlib.pyplot as plt
import ipywidgets as widgets

from mpl_toolkits.mplot3d import Axes3D
from numpy.linalg import pinv
from scipy.stats import multivariate_normal

import seaborn as sns; sns.set()

plt.rcParams['figure.figsize'] = (4, 4)
np.set_printoptions(precision=2)
warnings.filterwarnings('ignore')

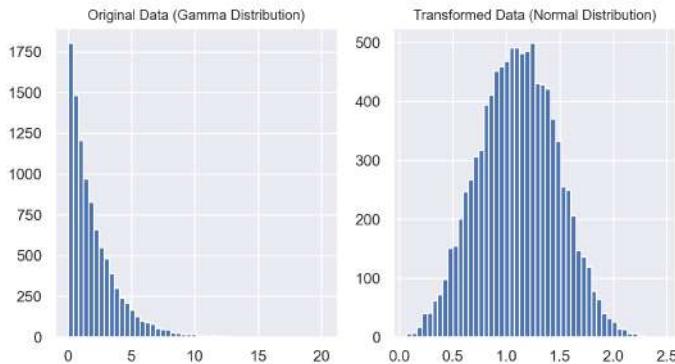
In [2]: X1 = np.random.gamma(1,2, size=(10000, 1))

plt.figure(figsize=(8, 4))
plt.subplot(121)
plt.hist(X1, bins=50)
plt.title('Original Data (Gamma Distribution)', size=10)

plt.subplot(122)
plt.hist(X1 ** 0.3, bins=50)
plt.title('Transformed Data (Normal Distribution)', size=10)

plt.show()

```

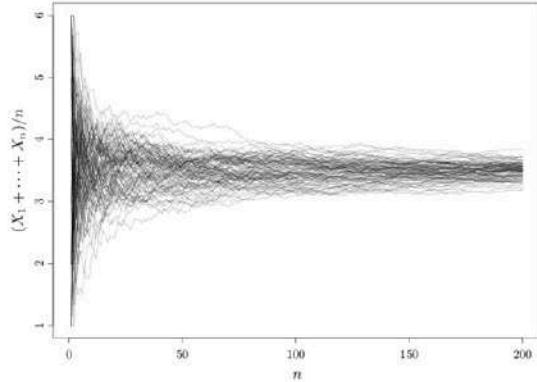


## Anomaly Detection (Outliers)

### Law of large numbers

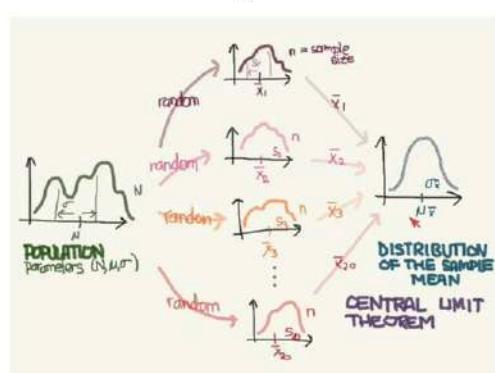
According to the law, the average of the results obtained from a large number of trials should be close to the expected value and tends to become closer to the expected value as more trials are performed

$$\lim_{n \rightarrow \infty} \sum \frac{x_i}{n} = \mu$$



### Central Limit Theorem (CLT)

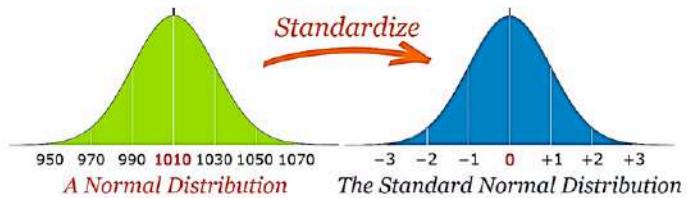
in many situations, for identically distributed independent samples, the standardized sample mean tends towards the standard normal distribution even if the original variables themselves are not normally distributed.



### Standard normal distribution

$$x \sim N(\mu, \sigma) \\ \downarrow \\ Z \sim N(0, 1)$$

$$p(z; \mu = 0, \sigma^2 = 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(z)^2}{2}}$$



### Example

$x \sim N(\mu = 10, \sigma = 2)$  calculate  $P(x < 13)$ ?

z	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6935	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9655	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

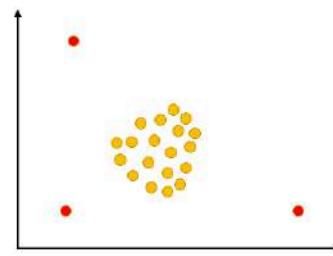
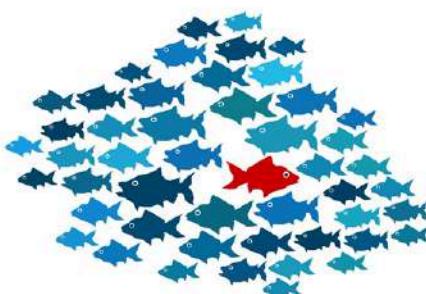
## Anomaly Detection (Outliers)

### Outliers(anomaly detection) – one class classification

Outliers are those data points that are *significantly* different from the rest of the dataset.

Fraud detection

Network Security



### Reasons of Occurrence of Outliers:

- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (data extraction or experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation errors)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

## Parameter evaluation

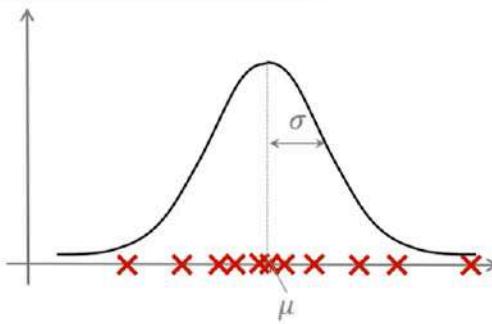
data set  $\{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$p(x; \mu, \sigma^2) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$$

- if  $p(x) < \varepsilon$ , then  $x$  can be detected as an **anomalous data**.
- if  $p(x) > \varepsilon$ , then  $x$  can be regarded as a **normal data**



```
In [3]: import numpy as np
np.set_printoptions(precision=2)

import numpy as np
def read_dataset(fname, delimiter=','):
    return np.genfromtxt(fname, delimiter=delimiter)

X = read_dataset(r"D:/AI_Machinelearning/machine_learning/jozavat/anomaly/onedimensional.csv")
print(X[:10])
(307.)
[13.05 13.41 14.2  14.91 13.58 13.92 12.82 15.68 16.16 12.67]
```

```
In [4]: mu=X.mean()
print(mu)
Sigma=X.std()
print(Sigma)

14.112252768729643
1.3537642702832688
```

```
In [5]: z=(X-mu) / Sigma
print(z.shape)
print(np.around(z[:10] , 3 ))
```

```
(307. -0.79  0.52  0.06  0.59 -0.4 -0.14 -0.95  1.16  1.51 -1.07)
```

```
In [6]: from scipy.stats import norm

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(111)

p_z = norm().pdf(z)      #probability density function
print(p_z)

outlier =z [p_z < 0.001]
print(outlier.shape)

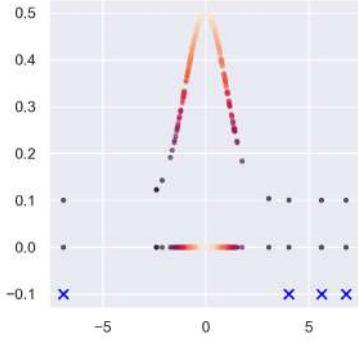
plt.scatter(z , p_z +0.1, s=8, alpha=0.6 , c=p_z)

a=np.array([0 for i in range(307)])
plt.scatter( z , a , s= 8 , alpha = 0.6 , c=p_z)

b=a=np.array([0 for i in range(outlier.shape[0])])
plt.scatter(outlier, b-.1, s=50, marker='x', c='blue')

plt.show()

np.argwhere(p_z < 0.001)
```



```
[2.93e-01 3.49e-01 3.98e-01 3.35e-01 3.69e-01 3.95e-01 2.53e-01 2.05e-01
1.27e-01 2.25e-01 3.97e-01 3.99e-01 3.46e-01 3.47e-01 3.97e-01 3.99e-01
3.96e-01 3.86e-01 3.75e-01 1.28e-01 3.66e-01 3.97e-01 3.26e-01 3.97e-01
2.65e-01 2.12e-01 3.74e-01 3.49e-01 3.75e-01 3.84e-01 2.31e-01 3.97e-01
2.77e-01 2.65e-01 1.47e-01 3.33e-01 1.90e-01 3.99e-01 3.80e-01 3.98e-01
3.81e-01 3.05e-01 3.82e-01 3.98e-01 3.61e-01 3.91e-01 1.51e-01 3.99e-01
3.91e-01 2.51e-01 3.26e-01 3.78e-01 3.87e-01 3.99e-01 3.79e-01 3.99e-01
3.99e-01 3.96e-01 3.54e-01 3.98e-01 3.27e-01 3.97e-01 2.71e-01 3.92e-01
3.97e-01 3.26e-01 3.81e-01 3.82e-01 2.02e-01 2.77e-01 3.71e-01 3.81e-01
3.96e-01 2.07e-01 3.93e-01 3.96e-01 3.43e-01 3.99e-01 1.53e-01 3.57e-01
3.87e-01 2.54e-01 3.78e-01 2.56e-01 2.28e-01 3.92e-01 3.74e-01 3.19e-01
3.91e-01 3.98e-01 3.24e-01 3.45e-01 3.93e-01 1.78e-01 3.94e-01 1.78e-01
1.83e-01 3.63e-01 3.62e-01 3.71e-01 1.63e-01 3.47e-01 3.01e-01 2.56e-01
2.44e-01 3.85e-01 2.01e-01 3.50e-01 2.94e-01 4.26e-02 3.94e-01 2.67e-01
2.80e-01 3.99e-01 3.98e-01 1.53e-01 3.47e-01 3.01e-01 3.92e-01 2.31e-01
2.69e-01 3.98e-01 3.89e-01 3.75e-01 3.80e-01 3.72e-01 2.98e-01 3.31e-01
1.47e-01 3.94e-01 3.42e-01 3.98e-01 3.48e-01 2.98e-01 2.29e-01 3.98e-01
3.42e-01 3.47e-01 3.70e-01 3.02e-01 3.79e-01 3.97e-01 1.23e-01 3.75e-01
3.68e-01 2.17e-01 2.91e-01 2.22e-02 2.73e-01 3.97e-01 2.09e-01 2.04e-01
3.20e-01 3.97e-01 3.56e-01 3.07e-01 3.93e-01 3.66e-01 2.82e-01 3.92e-01
3.70e-01 1.56e-01 3.78e-01 3.93e-01 3.73e-01 3.99e-01 3.99e-01 3.99e-01
1.69e-01 3.97e-01 2.32e-02 3.94e-01 1.74e-01 2.94e-01 2.65e-01 3.66e-01
3.69e-01 3.97e-01 2.66e-01 1.64e-01 3.92e-01 3.99e-01 2.20e-01 3.92e-01
3.83e-01 3.97e-01 3.01e-01 2.97e-01 3.35e-01 3.97e-01 3.82e-01 2.00e-01
3.46e-01 3.99e-01 3.45e-01 2.32e-01 3.29e-01 3.96e-01 2.96e-01 3.72e-01
2.29e-01 3.93e-01 3.98e-01 2.60e-01 3.69e-01 3.46e-01 3.80e-01 2.63e-01
3.81e-01 3.98e-01 3.75e-01 3.21e-01 3.99e-01 3.86e-01 3.69e-01 3.92e-01
3.96e-01 3.32e-01 3.86e-01 3.53e-01 3.98e-01 3.90e-01 8.34e-02 1.48e-01
3.39e-01 2.96e-01 9.14e-02 2.64e-01 3.73e-01 3.80e-01 2.22e-01 2.27e-01
3.90e-01 2.87e-01 3.12e-01 3.42e-01 2.53e-01 3.88e-01 3.70e-01 1.89e-01
3.98e-01 2.73e-01 3.81e-01 3.06e-01 3.90e-01 2.89e-01 3.80e-01 3.26e-01
3.24e-01 1.37e-01 3.91e-01 1.54e-01 3.85e-01 3.41e-01 2.99e-01 3.73e-01
2.93e-01 3.99e-01 3.22e-01 3.96e-01 2.55e-01 3.14e-01 3.46e-01 2.96e-01
3.78e-01 3.71e-01 3.31e-01 3.89e-01 3.95e-01 3.64e-01 3.51e-01 3.87e-01
3.86e-01 3.99e-01 2.70e-01 3.11e-01 2.31e-01 3.92e-01 3.28e-01 1.07e-01
3.43e-01 1.52e-01 3.99e-01 3.26e-01 3.57e-01 3.38e-01 2.93e-01 3.87e-01
2.93e-01 3.98e-01 3.90e-01 3.99e-01 3.79e-01 3.99e-01 3.76e-01 3.81e-01
3.95e-01 3.78e-01 3.98e-01 1.24e-01 2.98e-01 5.38e-08 1.92e-01 1.13e-04
3.25e-11 3.64e-03 1.66e-11]
```

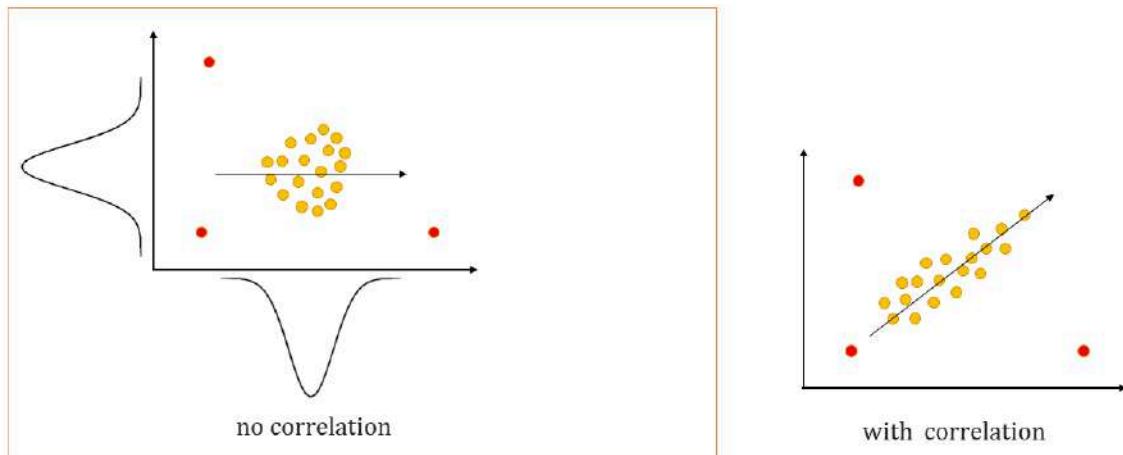
(4,)

```
Out[6]: array([[301],
 [303],
 [304],
 [306]], dtype=int64)
```

### Algorithm

data set  $\{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$   $x^{(i)} \in R^n$

- Features follow a normal distribution.
- Features do not follow a normal distribution
- correlation of  $x_1, x_2$  are zero. ( covariance matrix is diagonal)
- correlation of  $x_1, x_2, \dots, x_n$  are not zero

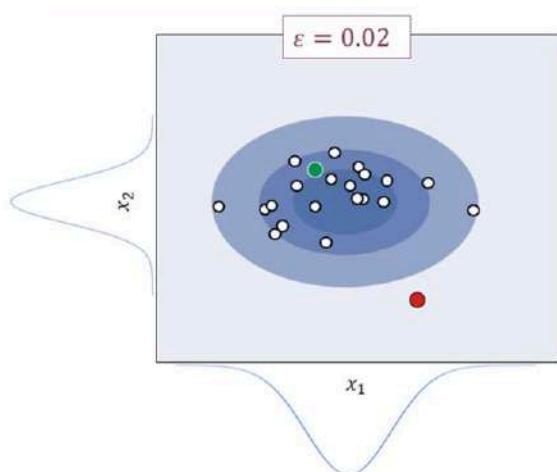


### Algorithm

If  $x_1, x_2, \dots, x_j$  are independent and  $x_j \sim N(\mu_j, \sigma_j^2)$ :

$$P(x) : p(x_1; \mu_1, \sigma_1^2) p(x_2; \mu_2, \sigma_2^2) \dots p(x_n; \mu_n, \sigma_n^2) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

$$\prod_{j=1}^n \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}}$$

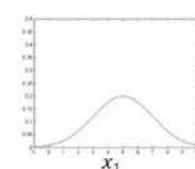


$$\mu_1 = 5, \sigma_1 = 2$$

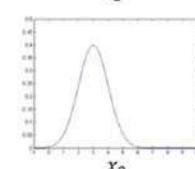
$$\mu_2 = 3, \sigma_2 = 1$$

$$p(x_{test}^{(1)}) = 0.0426$$

$$p(x_{test}^{(2)}) = 0.0021$$



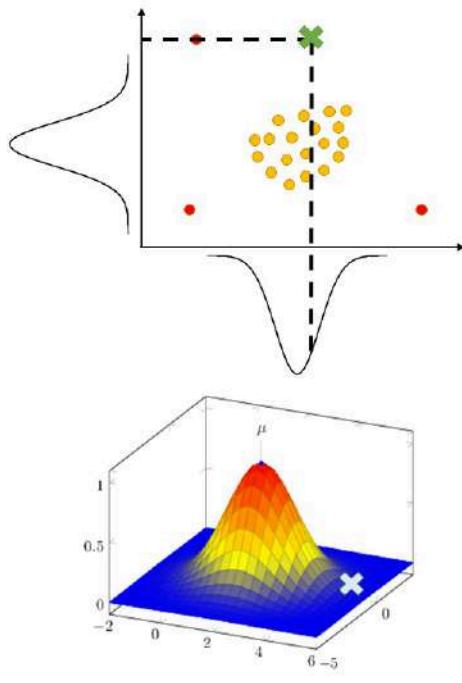
$$p(x_1; \mu_1, \sigma_1^2)$$



$$p(x_2; \mu_2, \sigma_2^2)$$

## Anomaly Detection (Outliers)

### Problem



multivariate normal - multinomial

$$P(x; \mu, \Sigma) = \frac{1}{|\Sigma|^{\frac{1}{2}} (2\pi)^{\frac{n}{2}}} e^{-\frac{(x-\mu)^T \Sigma^{-1} (x-\mu)}{2}}$$

Parameters :  $\mu \in R^n$ ,  $\Sigma \in R^{n \times n}$

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

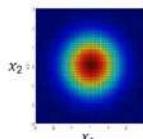
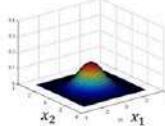
$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

$$\begin{bmatrix} \text{Var}(x_1) & \dots & \text{Cov}(x_n, x_1) \\ \vdots & \ddots & \vdots \\ \text{Cov}(x_n, x_1) & \dots & \text{Var}(x_n) \end{bmatrix}$$

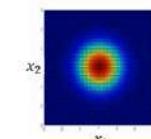
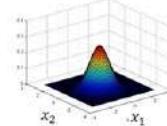
## Anomaly Detection (Outliers)

13

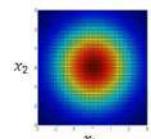
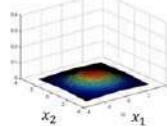
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



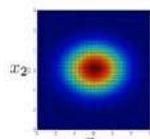
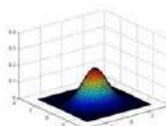
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$



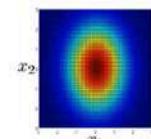
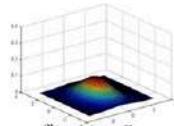
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$



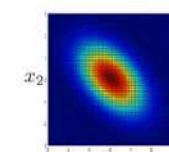
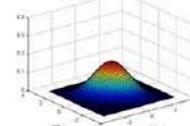
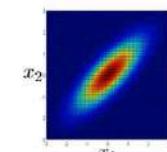
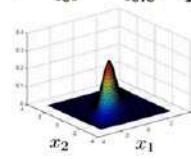
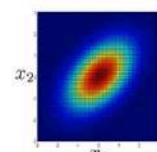
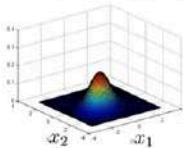
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$$



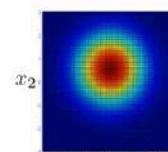
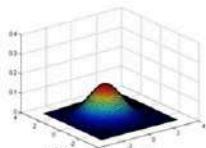
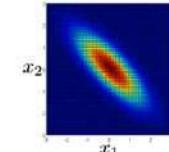
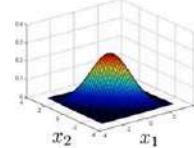
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$



```
In [7]: import numpy as np
def read_dataset(fname, delimiter=','):
    return np.genfromtxt(fname, delimiter=delimiter)

X = read_dataset(r"D:\AI_Machinelearning\machine_learning\jozavat\anomaly\tr_server_data.csv")
```

```

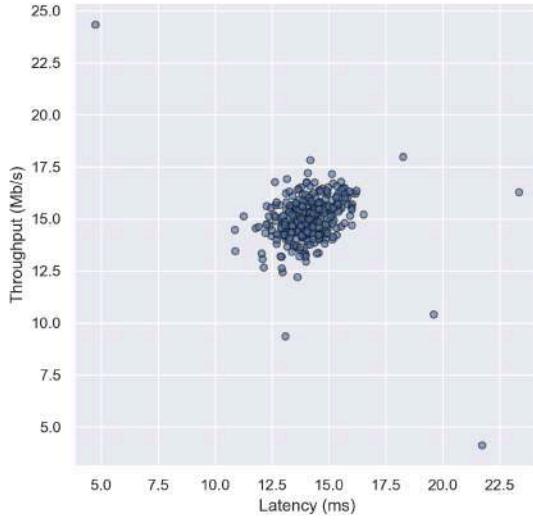
print(X.shape)
print(X[:10])

(307, 2)
[[13.05 14.74]
 [13.41 13.76]
 [14.2 15.85]
 [14.91 16.17]
 [13.58 14.04]
 [13.92 13.41]
 [12.82 14.22]
 [15.68 15.89]
 [16.16 16.2 ]
 [12.67 14.9 ]]

In [8]: import matplotlib.pyplot as plt
plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], s=25, edgecolors='k', cmap='coolwarm', alpha=0.6)
plt.xlabel('Latency (ms)')
plt.ylabel('Throughput (Mb/s)')

plt.show()

```



```

In [9]: np.set_printoptions(precision=2)

mu = np.mean(X, axis=0)
m=X.shape[0]

Sigma= np.cov(X.T)      # sigma= ((X-mu).T @ (X-mu)) * 1/m

print('mu =\n{}'.format(mu))
print('Sigma =\n{}'.format(Sigma))

mu =
[14.11 15.  ]
Sigma =
[[ 1.84 -0.23]
 [-0.23  1.72]]

In [10]: # defining the probabilistic model and computing
# the probability of observing each data in training data
from scipy.stats import multivariate_normal

p = multivariate_normal(mean=mu, cov=Sigma).pdf(X)

# print the probabilities for the first 10 data
print(p.shape)
print(p[:10])

(307,)
[0.06 0.05 0.07 0.05 0.06 0.04 0.04 0.03 0.02 0.05]

In [11]: X = read_dataset(r"D:\AI_Machinelearning\machine_learning\jozavat\anomaly\tr_server_data.csv")
p = multivariate_normal(mean=mu, cov=Sigma).pdf(X)

plt.figure(figsize=(9, 3))

plt.subplot(131)
EPSILON = 1e-3
plt.scatter(X[p >= EPSILON, 0], X[p >= EPSILON, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter(X[p < EPSILON, 0], X[p < EPSILON, 1], s=50, marker='x', c='r', edgecolors='k')
plt.title('$\epsilon$=0.00001$')

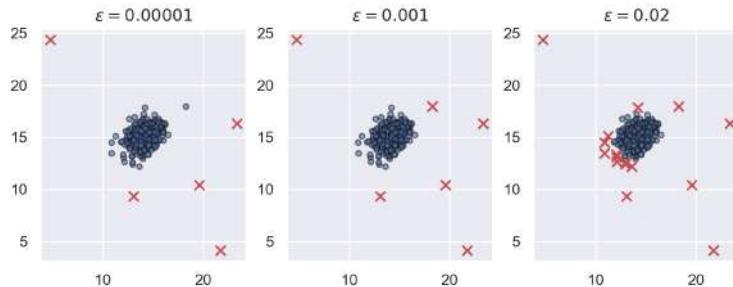
plt.show()

plt.subplot(132)
EPSILON = 1e-3
plt.scatter(X[p >= EPSILON, 0], X[p >= EPSILON, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter(X[p < EPSILON, 0], X[p < EPSILON, 1], s=50, marker='x', c='r', edgecolors='k')
plt.title('$\epsilon$=0.001$')

plt.subplot(133)
EPSILON = 1e-2
plt.scatter(X[p >= EPSILON, 0], X[p >= EPSILON, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter(X[p < EPSILON, 0], X[p < EPSILON, 1], s=50, marker='x', c='r', edgecolors='k')
plt.title('$\epsilon$=0.02$')

plt.show()

```



## Anomaly detection as an algorithm

### Anomaly Detection (Outliers)

#### Train - Test - Validation

Normal :  $y = 0$

Anomalous :  $y = 1$

*data set*       $\{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\} \quad x^{(i)} \in R^n$

*validation data*       $\{(x_{cv}^{(1)}, y_{cv}^{(1)}), (x_{cv}^{(2)}, y_{cv}^{(2)}), \dots (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})\}$

*test data*       $\{(x_{test}^{(1)}, y_{test}^{(1)}), (x_{test}^{(2)}, y_{test}^{(2)}) \dots (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})\}$



## Anomaly Detection (Outliers)

### Criteria evaluation

#### Confusion Matrix

- True positive

- False positive

- True negative

- False negative

- Precision

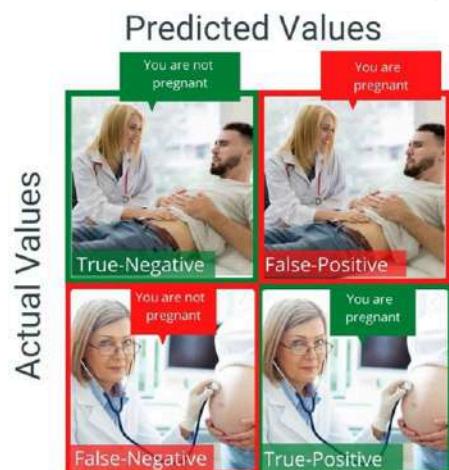
- Recall

- $F_1$  score

#### Errors hypothesis testing

Error type I:  $H_0$  is true but we reject it.      **False-positive**

Error type II:  $H_0$  is false but we accept it.      **False-negative**



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

## Anomaly Detection (Outliers)

18

### Criteria evaluation

- Precision

$$0 \leq \frac{\text{ture positive}}{\text{ture positive} + \text{false positive}} \leq 1$$

- Recall

$$0 \leq \frac{\text{ture positive}}{\text{ture positive} + \text{false negative}} \leq 1$$

- $F_1$  score

$$F_1 = 2 * \frac{\text{precision} \times \text{recal}}{\text{precision} + \text{recal}}$$

```
In [12]: import numpy as np
def read_dataset(fname, delimiter=','):
    return np.genfromtxt(fname, delimiter=delimiter)

X_train = read_dataset(r"D:\AI_Machinelearning\machine_learning\jozavat\anomaly\test_train_vald\train.csv") #train dataset
print(X_train.shape)
```

```

print(X_train[:5])
print("-----")
data_validation = read_dataset(r"D:\AI_Machinlearning\machine_learning\jozavat\anomaly\test train vald\validation.csv") # validation dataset
print(data_validation.shape)
print(data_validation[:5])
print("-----")
data_test = read_dataset(r"D:\AI_Machinlearning\machine_learning\jozavat\anomaly\test train vald\test.csv") #test
print(data_test.shape)
print(data_test[-1:-6:-1])
(245, 2)
[[13.05 14.74]
 [13.41 13.76]
 [14.2 15.85]
 [14.91 16.17]
 [13.58 14.04]]
-----

```

```

(31, 3)
[[13.03 14.25 0. ]
 [14.53 15.77 0. ]
 [13.25 16.32 0. ]
 [13.24 15.34 0. ]
 [12.13 12.67 0. ]]
-----

```

```

(31, 3)
[[ 4.75 24.35 0. ]
 [18.26 17.98 1. ]
 [23.34 16.3 1. ]
 [19.58 10.41 1. ]
 [21.73 4.13 0. ]]

```

```
In [13]: Xvalid=data_validation[:, :2]
```

```
yValid= data_validation[:, -1]
```

```
#import multivariate_normal
from scipy.stats import multivariate_normal
muv= np.mean(Xvalid , axis=0)
Sigmav= np.cov(Xvalid.T)
```

```
#create pdf
p_valid = multivariate_normal(mean=muv, cov=Sigmav).pdf(Xvalid)
```

```
p_valid
```

```
Out[13]: array([9.66e-02, 8.03e-02, 2.83e-02, 7.72e-02, 1.97e-02, 7.15e-02,
 8.91e-03, 1.08e-01, 9.58e-02, 8.69e-02, 1.27e-01, 3.16e-02,
 1.16e-01, 1.11e-01, 1.36e-01, 3.11e-02, 1.00e-01, 1.20e-01,
 4.49e-02, 7.80e-02, 8.12e-02, 6.26e-02, 1.32e-01, 5.96e-02,
 7.14e-02, 4.76e-02, 1.37e-01, 1.27e-01, 2.22e-02, 5.39e-05,
 4.39e-02])
```

```
In [14]: #create some epsilons
```

```
stepsize = (max(p_valid) - min(p_valid)) / 1000;
```

```
epsilons = np.arange(min(p_valid), max(p_valid), stepsize)
```

```
In [15]: from sklearn.metrics import precision_score , recall_score , f1_score
```

```
best_epsilon = 0
```

```
best_f1 = 0
```

```
for epsilon in np.nditer(epsilons):
    #predict outlier
    y_pred = (p_valid < epsilon)
    f1= f1_score(y_true=yValid , y_pred=y_pred)
    if f1>best_f1: best_f1=f1 ; best_epsilon=epsilon
```

```
print(f" the best epsilon is {best_epsilon} and the best f1 is {best_f1}")
```

```
the best epsilon is 0.022173060113477056 and the best f1 is 0.5714285714285715
```

```
In [16]: #Train _dataset
```

```
mu_train = np.mean(X_train, axis=0)
Sigma_train = np.cov(X_train.T)
```

```
p_train = multivariate_normal(mean=mu_train, cov=Sigma_train).pdf(X_train)
```

```
In [17]: import matplotlib.pyplot as plt
plt.figure(figsize=(8, 4))
```

```
plt.subplot(121)
plt.scatter(Xvalid [p_valid >= best_epsilon, 0], Xvalid [p_valid >= best_epsilon, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter(Xvalid [p_valid < best_epsilon, 0], Xvalid [p_valid < best_epsilon, 1], s=50, marker='x', c='r', edgecolors='k')
```

```
plt.title('Validation Data ($\epsilon$={:.5g})'.format(best_epsilon))

plt.subplot(122)
plt.scatter(X_train[p_train >= best_epsilon, 0], X_train[p_train >= best_epsilon, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter(X_train[p_train < best_epsilon, 0], X_train[p_train < best_epsilon, 1], s=50, marker='x', c='r', edgecolors='k')
plt.title('Training Data ($\epsilon$={:.5g})'.format(best_epsilon))

plt.show()
```



```
from scrach
```

```
In [18]: import numpy as np
```

```

def select_threshold(p_valid, y_valid):
    best_epsilon = 0
    best_f1 = 0

    stepsize = (max(p_valid) - min(p_valid)) / 1000
    epsilon = np.arange(min(p_valid), max(p_valid), stepsize)

    for epsilon in np.nditer(epsilon):
        # Predict outliers
        y_pred = (p_valid < epsilon)

        # Calculate TP, FP and FN
        tp = np.sum((y_pred == 1) & (y_valid == 1)) * 1.0
        fp = np.sum((y_pred == 1) & (y_valid == 0)) * 1.0
        fn = np.sum((y_pred == 0) & (y_valid == 1)) * 1.0

        # Calculate Precision, Recall and F1-score with checks
        if tp + fp == 0:
            precision = 0      # or np.nan
        else:
            precision = tp / (tp + fp)

        if tp + fn == 0:
            recall = 0         # or np.nan
        else:
            recall = tp / (tp + fn)

        if precision + recall == 0:
            f1 = 0             # or np.nan
        else:
            f1 = (2 * precision * recall) / (precision + recall)

        if f1 > best_f1:
            best_f1 = f1
            best_epsilon = epsilon

    return best_f1, best_epsilon

```

```

In [19]: from scipy.stats import multivariate_normal
# STEP 1: estimate parameters mu and sigma from X_val
mu_validation = np.mean((data_validation[:, 0:2]), axis=0)
Sigma_val = np.cov((data_validation[:, 0:2]).T)

# STEP 2: calculate probabilities
p_val = multivariate_normal(mean=mu_validation, cov=Sigma_val).pdf(data_validation[:, 0:2])

# STEP 3: choose best value for epsilon
f1, eps = select_threshold(p_val, data_validation[:, 0:2])
print("f1 = {:.2g}, epsilon = {}".format(f1, eps))

```

f1 = 0.57, epsilon = 0.022173060113477056

```
In [20]: np.argwhere(p_val<0.022173060113477056)
```

```
Out[20]: array([[ 4],
   [ 6],
   [28],
   [29]], dtype=int64)
```

```

In [21]: import matplotlib.pyplot as plt
plt.figure(figsize=(8, 4))

plt.subplot(121)
plt.scatter((data_validation[:, 0:2])[p_val >= eps, 0], (data_validation[:, 0:2])[p_val >= eps, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter((data_validation[:, 0:2])[p_val < eps, 0], (data_validation[:, 0:2])[p_val < eps, 1], s=50, marker='x', c='r', edgecolors='k')
plt.title('Validation Data ($\epsilon$={:.5g})'.format(eps))

plt.subplot(122)
mu = np.mean(X_train, axis=0)
sigma = np.cov(X_train.T)
p = multivariate_normal(mean=mu, cov=sigma).pdf(X_train)
plt.scatter(X_train[p >= eps, 0], X_train[p >= eps, 1], s=15, marker='o', c='b', edgecolors='k', alpha=0.6)
plt.scatter(X_train[p < eps, 0], X_train[p < eps, 1], s=50, marker='x', c='r', edgecolors='k')
plt.title('Training Data ($\epsilon$={:.5g})'.format(eps))

plt.show()

```



outlier detection: The training data contains outliers which are defined as observations that are far from the others. Outlier detection estimators thus try to fit the regions where the training data is the most concentrated, ignoring the deviant observations.

novelty detection: The training data is not polluted by outliers and we are interested in detecting whether a new observation is an outlier. In this context an outlier is also called a novelty.

```

In [22]: X = read_dataset(r"D:\AI_Machinelearning\machine_learning\jozavat\anomaly\tr_server_data.csv")
print(X.shape)
print(X[:10])

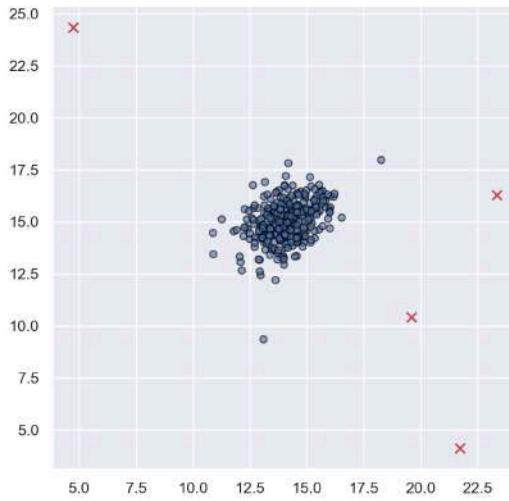
(307, 2)
[[13.05 14.74]
 [13.41 13.76]
 [14.2 15.85]
 [14.91 16.17]
 [13.58 14.04]
 [13.92 13.41]
 [12.82 14.22]
 [15.68 15.89]
 [16.16 16.2 ]
 [12.67 14.9 ]]

```

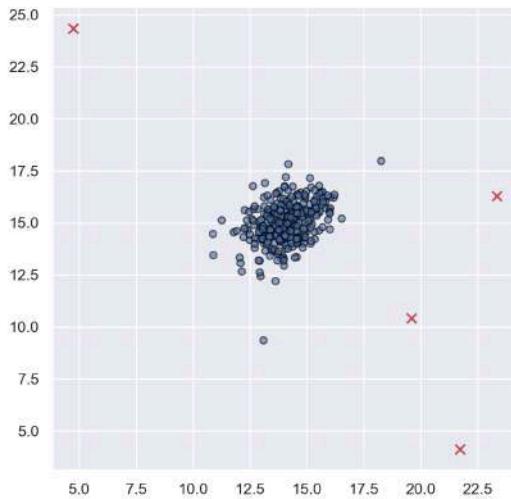
```

In [23]: from sklearn.covariance import EllipticEnvelope
anomaly=EllipticEnvelope(contamination=.01)
anomaly.fit(X)

```



```
Dut[26]: array([[301],  
   [303],  
   [304],  
   [306]], dtype=int64)  
  
In [27]: from sklearn.neighbors import LocalOutlierFactor  
  
clf = LocalOutlierFactor(n_neighbors=10, contamination=0.01)  
y_pred = clf.fit_predict(X)  
  
In [28]: fig = plt.figure(figsize=(6, 6))  
ax = fig.add_subplot(111)  
  
normal = X[y_pred== 1]  
outlier = X[y_pred== -1]  
ax.scatter(normal[:, 0], normal[:, 1], s=25, marker='o', c='b', edgecolors = "black",cmap='coolwarm', alpha=0.6)  
ax.scatter(outlier[:, 0], outlier[:, 1], s=50, marker='x', c='r', cmap='coolwarm')  
plt.show()  
  
np.argwhere(y_pred== -1)
```



```
Out[28]: array([301,
 [303,
 [304],
 [306]], dtype=int64)
In [29]: from sklearn.svm import OneClassSVM
CLF2 = OneClassSVM(nu=0.01, kernel="rbf", degree=3, gamma=0.001)
CLF2.fit(X)

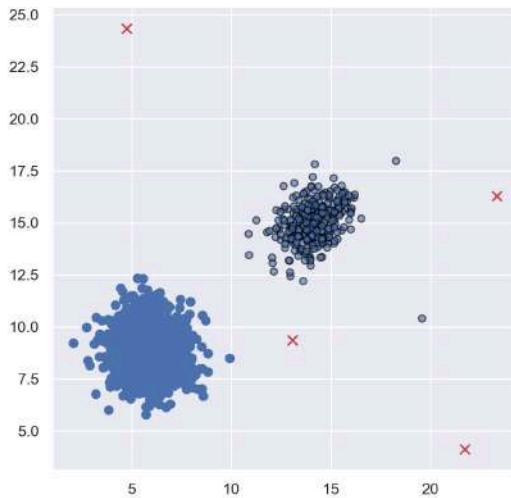
Out[29]: OneClassSVM(gamma=0.001, nu=0.01)
```

```
In [30]: predict2=CLF2.predict(X)

In [31]: fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111)

normal = X[predict2== 1]
outlier = X[predict2== -1]
ax.scatter(normal[:, 0], normal[:, 1], s=25, marker='o', c='b', edgecolors = "black",cmap='coolwarm', alpha=0.6)
ax.scatter(outlier[:, 0], outlier[:, 1], s=50, marker='x', c='r', cmap='coolwarm')
plt.show()

np.argwhere(predict2== -1)
```



```
Out[31]: array([300,
 [301,
 [304],
 [306]], dtype=int64)
```

# Machine learning

## Advanced Optimization Methods

Morteza khorsand



### Advanced Optimization Methods

#### Methods

The minimize function supports the following methods:

- `minimize(method='Nelder-Mead')`
- `minimize(method='Powell')`
- `minimize(method='CG')`
- `minimize(method='BFGS')`
- `minimize(method='Newton-CG')`
- `minimize(method='L-BFGS-B')`
- `minimize(method='TNC')`
- `minimize(method='COBYLA')`
- `minimize(method='SLSQP')`
- `minimize(method='trust-constr')`
- `minimize(method='dogleg')`
- `minimize(method='trust-ncg')`
- `minimize(method='trust-krylov')`
- `minimize(method='trust-exact')`

Example1

## Advanced Optimization Methods

$$F(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 1)^2$$

- Create a main function

```
def F(params : x1 , x2):  
    return (params[0]- 3 ) ** 2 + (params[1] - 1) ** 2
```

- Create a Gradient function

$$\frac{\partial F}{\partial x_1} = 2(x_1 - 3)$$

$$\frac{\partial F}{\partial x_2} = 2(x_2 - 1)$$

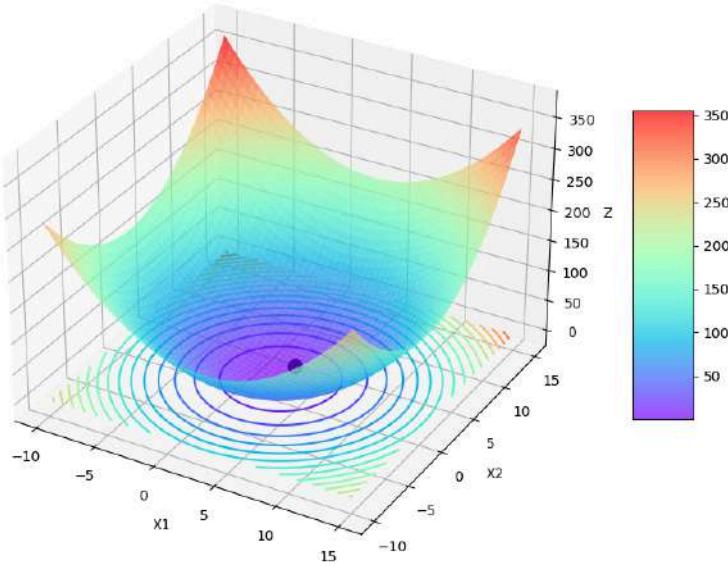
```
def grad_F(params):  
    return np.array ([2*(params[0]-3) , 2*(params[1] -1)] )
```

- from scipy.optimize import minimize

```
minimize(F, x0=[0,0] , method='BFGS', jac=grad_F)
```

## Advanced Optimization Methods

```
In [1]: from scipy.optimize import minimize  
import numpy as np  
  
In [2]: def F (params : "x1 , x2"):  
    return (params[0]- 3 ) ** 2 + (params[1] - 1) ** 2  
  
In [3]: def grad_F(params):  
    return np.array ([2*(params[0]-3) , 2*(params[1] -1)] )  
  
In [7]: result =minimize(F, x0=[0,0] , method='cg' , jac=grad_F)  
  
Out[7]: message: Optimization terminated successfully.  
success: True  
status: 0  
fun: 4.9545506515086484e-14  
x: [ 3.000e+00  1.000e+00]  
nit: 1  
jac: [ 4.372e-07  1.557e-07]  
nfev: 9  
njev: 3  
  
In [5]: optimal_params = result.x  
optimal_params  
  
Out[5]: array([3., 1.])  
  
In [6]: Z_min = F(optimal_params)  
Z_min  
  
Out[6]: 0.0  
  
In [8]: import matplotlib.pyplot as plt  
from mpl_toolkits.mplot3d import Axes3D  
import numpy as np  
  
t0 = np.linspace(-10, 15, 100)  
t1 = np.linspace(-10, 15, 100)  
T0, T1 = np.meshgrid(t0, t1)  
Z = (T0-3)**2 + (T1-1)**2  
  
fig = plt.figure(figsize=(12, 8))  
ax = fig.add_subplot(111, projection='3d')  
  
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow , alpha=0.7)  
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)  
  
# Plot the minimum point  
ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point' , s=100)  
ax.set_xlabel('X1')  
ax.set_ylabel('X2')  
ax.set_zlabel('Z')  
fig.colorbar(surf, shrink=0.5, aspect=5)  
plt.show()
```



## Example2

### Advanced Optimization Methods

■  $F(x_1, x_2) = \frac{x_1^2}{2} + x_1x_2 + x_2^2 - 2x_2$

- Create a main function

```
def f(params):
    return params[0]**2/2 + params[0]*params[1] + params[1]**2 - 2*params[1]
```

- Create a Gradient function

$$\frac{\partial F}{\partial x_1} = x_1 + x_2$$

$$\frac{\partial F}{\partial x_2} = x_1 + 2x_2 - 2$$

```
def grad(params):
    return np.array([params[0]+params[1], params[0] + 2 * params[1] - 2])
```

- `minimize(f, x0=[0, 0], method='Newton-CG', jac=grad )`

```
In [9]: def f(params):
    return params[0]**2/2 + params[0]*params[1] + params[1]**2 - 2*params[1]

def grad(params):
    return np.array([params[0]+params[1], params[0] + 2 * params[1] - 2])

result2= minimize(f, x0=[0, 0], method='Newton-CG', jac=grad )
```

```
Dut[9]: message: Optimization terminated successfully.
success: True
status: 0
fun: -2.000e+00 2.000e+00
nit: 3
jac: [ 0.000e+00  0.000e+00]
nfev: 3
njev: 7
nhev: 0
```

```
In [12]: optimal_params2 = result2.x
Z_min2 = f(optimal_params)
```

```
optimal_params2
Z_min2
```

```
Out[12]: 6.5
```

```
In [13]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

t0 = np.linspace(-10, 15, 100)
t1 = np.linspace(-10, 15, 100)

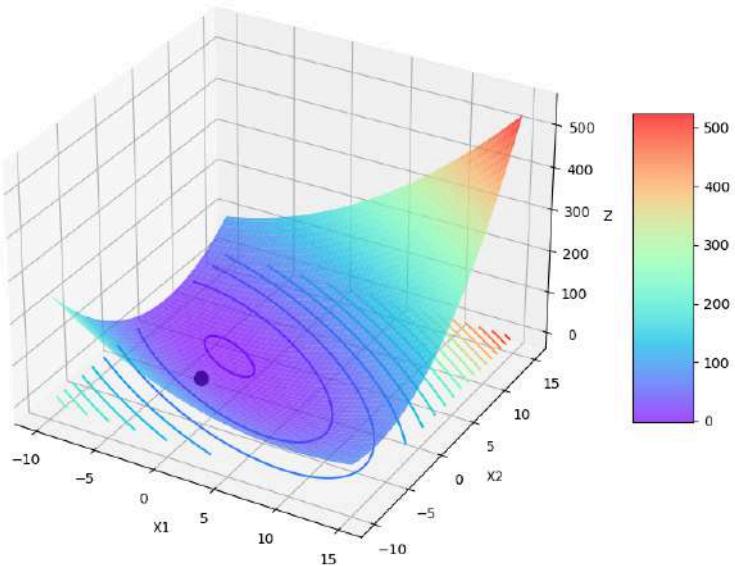
T0, T1 = np.meshgrid(t0, t1)

Z = T0**2/2 + T0*T1 + T1**2 - 2*T1

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.7)
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

# Plot the minimum point
ax.scatter(optimal_params2[0], optimal_params2[1], Z_min2, color='k', label='Minimum Point', s=100)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Z')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```



### Example3

## Advanced Optimization Methods

$$F(x_1, x_2) = x_1^4 - 2x_1^2x_2 + x_1^2 + x_2^2 - 2x_1 + 1$$

- Create a main function

```
def F(params):
    return params[0]**4 - 2*params[0]**2*params[1] + params[0]**2 + params[1]**2 - 2*params[0] + 1
```

- Create a Gradient function

$$\frac{\partial F}{\partial x_1} = 4x_1^3 - 4x_1x_2 + 2x_1 - 2$$
$$\frac{\partial F}{\partial x_2} = -2x_1^2 + 2x_2$$

```
def grad_F(params):
    dF_dx1 = 4*params[0]**3 - 4*params[0]*params[1] + 2*params[1] - 2
    dF_dx2 = -2*params[0]**2 + 2*params[1]
    return np.array([dF_dx1, dF_dx2])
```

- minimize(F, x0=[0, 0], method='BFGS', jac=grad )

```
In [14]: import numpy as np
from scipy.optimize import minimize

# Define the function
def F(params):
    return params[0]**4 - 2*params[0]**2*params[1] + params[0]**2 + params[1]**2 - 2*params[0] + 1

# Define the gradient function
def grad_F(params):
    dF_dx1 = 4*params[0]**3 - 4*params[0]*params[1] + 2*params[1] - 2
    dF_dx2 = -2*params[0]**2 + 2*params[1]
    return np.array([dF_dx1, dF_dx2])

initial_params=np.array([0,0])

# Perform optimization using the BFGS method
result3 = minimize(F, initial_params, method='BFGS', jac=grad_F)
print(result3)
```

```
message: Optimization terminated successfully.
success: True
status: 0
fun: 2.424727085781342e-12
x: [ 1.000e+00  1.000e+00]
nit: 11
jac: [ 2.392e-06  2.138e-06]
hess_inv: [[ 1.632e-01  2.444e-01]
           [ 2.444e-01  9.834e-01]]
nfev: 14
njev: 14
```

```
In [15]: optimal_params3 = result3.x
Z_min3 = F(optimal_params)
Z_min3
```

```
Out[15]: 68.0
```

```
In [16]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

t0 = np.linspace(-10, 10, 100)
t1 = np.linspace(-10, 10, 100)

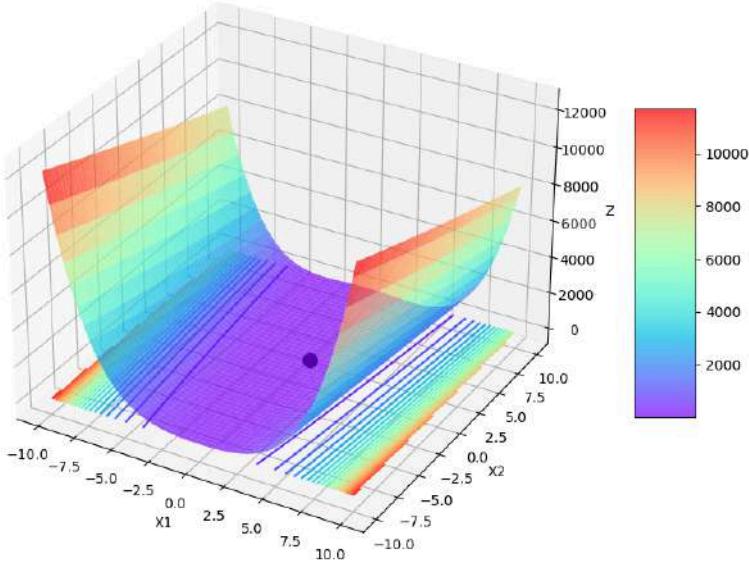
T0, T1 = np.meshgrid(t0, t1)

Z = T0**4 - 2*T0**2*T1 + T0**2 + T1**2 - 2*T0 + 1

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.7)
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

# Plot the minimum point
ax.scatter(optimal_params3[0], optimal_params3[1], Z_min3, color='k', label='Minimum Point', s=100)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Z')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```



## Example4

### Advanced Optimization Methods

■  $F(t_1, t_2) = t_1^3 - 3t_1 t_2^2$

- Create a main function

```
def F_new(params):
    return params[0] **3 - 3*params[0]*params[1]**2
```

- Create a Gradient function

$$\frac{\partial F}{\partial t_1} = 3 t_1^2 - 3 t_2^2$$

$$\frac{\partial F}{\partial t_2} = -6 x_1 x_2$$

```
def grad_F_new(params):
    dF_dt1 = 3*params[0]**2 - 3*params[1]**2
    dF_dt2 = -6*params[0]*params[1]
    return np.array([dF_dt1, dF_dt2])
```

- `minimize(F_new, x0=[0, 0], method='dogleg', jac=grad_F_new)`

```
In [17]: import numpy as np
from scipy.optimize import minimize

# Define the function
def F_new(params):
    return params[0] **3 - 3*params[0]*params[1]**2

# Define the gradient function
def grad_F_new(params):
    dF_dt1 = 3*params[0]**2 - 3*params[1]**2
    dF_dt2 = -6*params[0]*params[1]
    return np.array([dF_dt1, dF_dt2])

# Perform optimization using the BFGS method
result4 = minimize(F_new, x0=[0, 0], method='Newton-CG', jac=grad_F_new)
print(result4)
```

```

message: Optimization terminated successfully.
success: True
status: 0
  fun: [ 0.000e+00  0.000e+00]
  nit: 1
  jac: [ 0.000e+00 -0.000e+00]
  nfev: 1
  njev: 1
  nhev: 0

```

```
In [18]: optimal_params4 = result4.x
Z_min4 = F_new(optimal_params)
Z_min4
```

```
Out[18]: 18.0
```

```

In [19]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

t0 = np.linspace(-10, 10, 100)
t1 = np.linspace(-10, 10, 100)

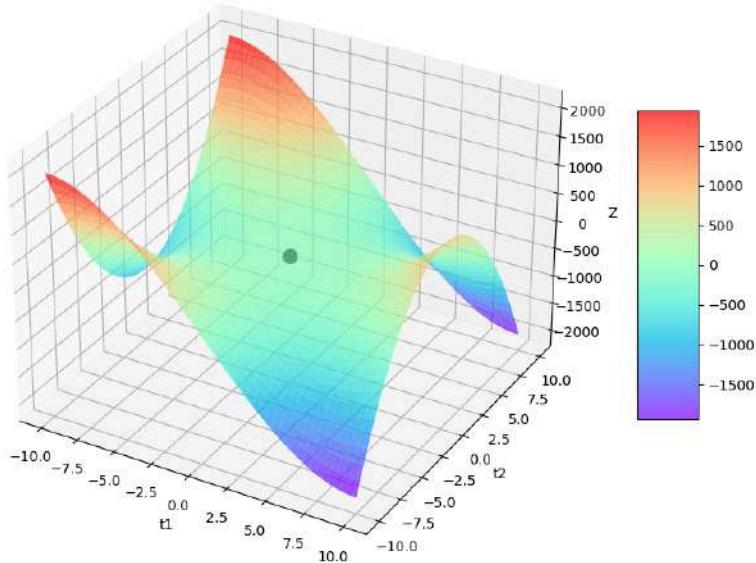
T0, T1 = np.meshgrid(t0, t1)

Z = T0 **3 - 3*T0*T1**2

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.7)
#cset = ax.contour(T0, T1, Z, zdir='z', offset=0, cmap=plt.cm.rainbow)

# Plot the minimum point
ax.scatter(optimal_params4[0], optimal_params4[0], Z_min4, color='k', label='Minimum Point', s=100)
ax.set_xlabel('t1')
ax.set_ylabel('t2')
ax.set_zlabel('Z')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()

```



## Example5

## Advanced Optimization Methods

$$F(x_1, x_2, x_3, x_4) = 100(x_2 - x_1^2)^2 + (1 - x_1^2)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10(x_2 + x_4 - 2)^2 + 0.1(x_2 - x_4)^2$$

- Create a main function

```
def F(params):  
    term1 = 100 * (params[1] - params[0]**2)**2  
    term2 = (1 - params[0])**2  
    term3 = 90 * (params[3] - params[2]**2)**2  
    term4 = (1 - params[2])**2  
    term5 = 10 * (params[1] + params[3] - 2)**2  
    term6 = 0.1 * (params[1] - params[3])**2  
    return term1 + term2 + term3 + term4 + term5 + term6
```

- Create a Gradient function

$$\frac{\partial F}{\partial x_1} = -400x_1(-x_1^2 + x_2) + 2x_1 - 2$$
$$\frac{\partial F}{\partial x_2} = -200x_1^2 + 220.2x_2 + 19.8x_4 - 40$$
$$\frac{\partial F}{\partial x_3} = -360x_3(-x_3^2 + x_4) + 2x_3 - 2$$
$$\frac{\partial F}{\partial x_4} = 19.8x_2 - 180x_3^2 + 200.2x_4 - 40$$

```
minimize(F, x0=[0, 0, 0, 0], method='TNC', jac=grad_F)
```

```
In [20]: def F(params):  
    term1 = 100 * (params[1] - params[0]**2)**2  
    term2 = (1 - params[0])**2  
    term3 = 90 * (params[3] - params[2]**2)**2  
    term4 = (1 - params[2])**2  
    term5 = 10 * (params[1] + params[3] - 2)**2  
    term6 = 0.1 * (params[1] - params[3])**2  
    return term1 + term2 + term3 + term4 + term5 + term6  
  
def grad_F (params):  
    df_dx1 = -400*params[0]*(-params[0]**2 + params[1]) + 2 *params[0] - 2  
    df_dx2 = -200*params[0]**2 + 220.2 * params[1] + 19.8 *params[3]-40  
    df_dx3=-360* params[2]*(-params[2]**2 + params[3]) + 2 *params[2] - 2  
    df_dx4= 19.8*params[1] - 180*params[2]**2 + 200.2*params[3] - 40  
    return np.array([df_dx1, df_dx2, df_dx3,df_dx4] )  
  
result5 = minimize(F, x0=[0, 0, 0, 0], method='TNC', jac=grad_F)  
print(result5)  
  
message: Converged (|f_n-f_(n-1)| ~ 0)  
success: True  
status: 1  
fun: 38.001747302741094  
x: [ 2.017e-03  1.223e-04  2.017e-03  2.100e-01]  
nit: 2  
jac: [-1.996e+00 -3.582e+01 -2.148e+00 -3.580e+02]  
nfev: 86  
  
In [21]: optimal_params5 = result5.x  
Z_min5 = F(optimal_params5)  
Z_min5  
Out[21]: 38.001747302741094
```

### Example6

```
In [22]: from IPython.display import Image  
  
# Path to the image file  
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\9.jpg"  
Image(filename=image_path, width=400, height=300)  
  
Out[22]: Rosenbrock (x1, x2) = (1 - x22) + 100 (x2 - x12)2  
  
In [23]: def rosenbrock(x):  
    return (1 - x[0])**2 + 100 * (x[1] - x[0]**2)**2  
  
def rosenbrock_gradient(x):  
    dfdx0 = -2 * (1 - x[0]) - 400 * (x[1] - x[0]**2) * x[0]  
    dfdx1 = 200 * (x[1] - x[0]**2)  
    return np.array([dfdx0, dfdx1])  
  
# Initial guess  
x0 = [0, 0]  
  
# Minimize using Newton-CG method with explicit gradient  
result6 = minimize(rosenbrock, x0, method='Newton-CG', jac=rosenbrock_gradient, options={'disp': True})  
print(result6.x)
```

```
Optimization terminated successfully.
    Current function value: 0.000000
    Iterations: 31
    Function evaluations: 49
    Gradient evaluations: 112
    Hessian evaluations: 0
[0.99999991 0.99999981]
```

```
In [24]: optimal_params6 = result.x
Z_min6 = rosenbrock(optimal_params5)
Z_min6
```

```
Out[24]: 0.9959714124143151
```

```
In [25]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

t0 = np.linspace(-10, 10, 100)
t1 = np.linspace(-10, 10, 100)

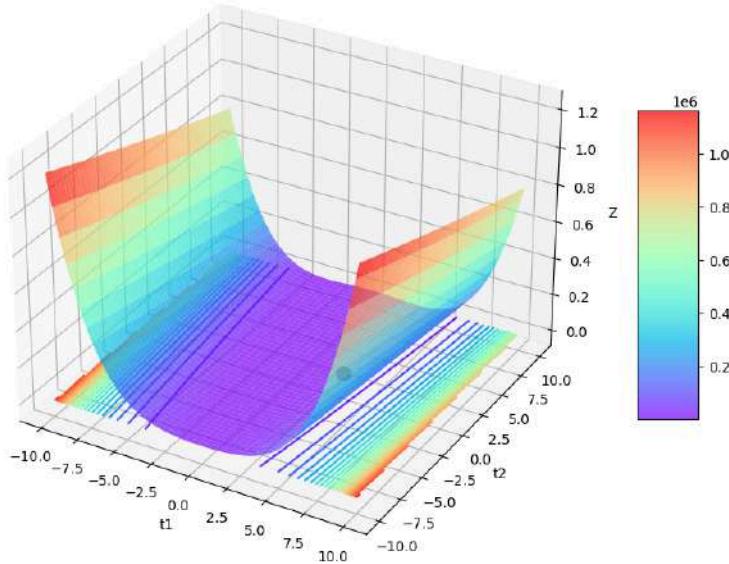
T0, T1 = np.meshgrid(t0, t1)

Z = (1 - T0)**2 + 100 * (T1 - T0**2)**2

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.7)
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

# Plot the minimum point
ax.scatter(optimal_params6[0], optimal_params6[1], Z_min6, color='k', label='Minimum Point', s=100)
ax.set_xlabel('t1')
ax.set_ylabel('t2')
ax.set_zlabel('Z')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```



## Example7

```
In [26]: from IPython.display import Image
# Path to the image file
image_path = r"..\..\..\MachineLearning\machine_learning\jozavat\advanced optimization methods\advanced\10.jpg"
Image(filename=image_path, width=250, height=150)
```

```
Out[26]: J(\theta_0, \theta_1) = (\theta_0)^2+(\theta_1)^2
```

```
In [27]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def J(params):
    return params[0]**2 + params[1]**2

# Perform optimization
result_new = minimize(J, x0=[0, 0], method='cg')
print(result_new)

# Optimal parameters
optimal_params = result_new.x
Z_min = J(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(-100, 100, 100)
t1 = np.linspace(-100, 100, 100)
T0, T1 = np.meshgrid(t0, t1)
Z = T0**2 + T1**2

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.8)

# Plot contours
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

# Plot the minimum point
```

```

ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point', s=100)

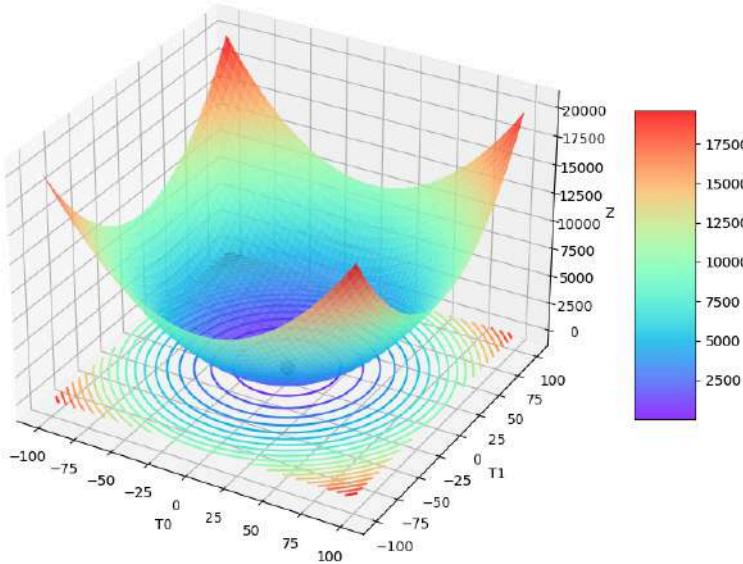
# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

# Show plot
plt.show()

message: Optimization terminated successfully.
success: True
status: 0
    fun: 0.0
    x: [ 0.000e+00  0.000e+00]
    nit: 0
    jac: [ 1.490e-08  1.490e-08]
    nfev: 3
    njev: 1

```



## Example8

```

In [28]: from IPython.display import Image

# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\12.jpg"
Image(filename=image_path, width=250, height=150)

```

Out[28]:  $J(\theta_0, \theta_1) = 2(\theta_0) \times (\theta_1)^3$

```

In [32]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def J(params):
    return 2*params[0] * params[1]**3

# Perform optimization using the BFGS method
result8 = minimize(J, x0=[0, 0], method='L-BFGS-B')
print(result8)

# Optimal parameters
optimal_params = result8.x
Z_min = J(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(-1, 1, 100)
t1 = np.linspace(-1, 1, 100)
T0, T1 = np.meshgrid(t0, t1)
Z = 2*T0 * T1**3

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.8)

# Plot contours
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

# Plot the minimum point
ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point', s=100)

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

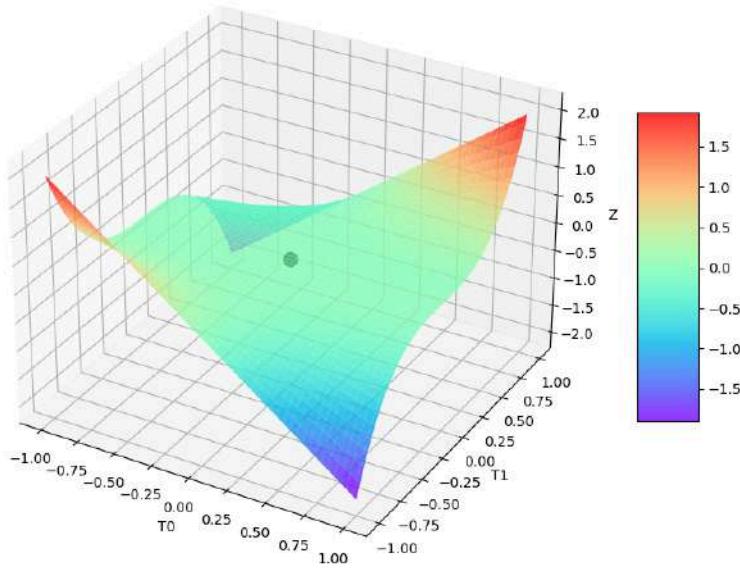
# Show plot
plt.show()

```

```

message: CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=PGTOL
success: True
status: 0
fun: 0.0
  x: [ 0.000e+00  0.000e+00]
nit: 0
jac: [ 0.000e+00  0.000e+00]
nfev: 3
njev: 1
hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>

```



## Example9

```
In [33]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\13.jpg"
Image(filename=image_path, width=250, height=150)
```

```
Out[33]: J(theta_0, theta_1) = 3(theta_0)^2 + (theta_1)^2
```

```
In [34]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def J(params):
    return 3*params[0]**2 + params[1]**2

# Perform optimization
result8 = minimize(J, x0=[0, 0], method='SLSQP')
print(result8)

# Optimal parameters
optimal_params = result8.x
Z_min8 = J(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(-1, 1, 100)
t1 = np.linspace(-1, 1, 100)
T0, T1 = np.meshgrid(t0, t1)
Z = 3*T0**2 + T1**2

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.8)

# Plot contours
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

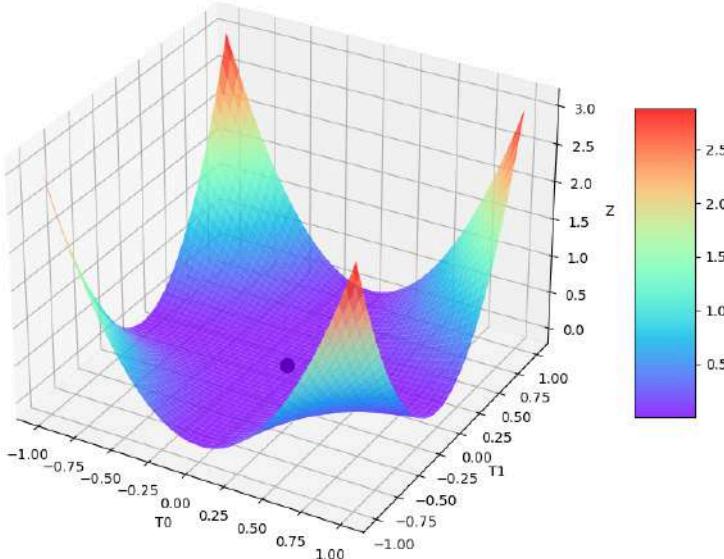
# Plot the minimum point
ax.scatter(optimal_params[0], optimal_params[1], Z_min8, color='k', label='Minimum Point', s=100)

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

# Show plot
plt.show()
```

message: Optimization terminated successfully  
success: True  
status: 0  
fun: 0.0  
 x: [ 0.000e+00 0.000e+00]  
nit: 1  
jac: [ 0.000e+00 0.000e+00]  
nfev: 3  
njev: 1



## Example10

```
In [35]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\14.jpg"
Image(filename=image_path, width=450, height=300)
```

```
Out[35]:  $J(\theta_0, \theta_1) = 3\theta_0^3 + 2\theta_0\theta_1^3 + \theta_0^2\theta_1^2$ 
```

```
In [36]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def F_new(params):
    x0, x1 = params
    return 3*x0**2 * x1**2 + 2*x0*x1**3 + x0**2*x1**2

# Define the Jacobian (gradient) of the function
def jac_F_new(params):
    x0, x1 = params
    dfdx0 = 6*x0*x1**2 + 2*x1**3 + 2*x0*x1**2
    dfdx1 = 6*x0**2*x1 + 6*x0*x1**2 + 2*x0**2*x1
    return np.array([dfdx0, dfdx1])

# Perform optimization using the BFGS method with the gradient
result_new = minimize(F_new, x0=[0, 0], method='TNC', jac=jac_F_new)
print(result_new)

# Optimal parameters
optimal_params = result_new.x
Z_min = F_new(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(-1, 1, 100)
t1 = np.linspace(-1, 1, 100)
T0, T1 = np.meshgrid(t0, t1)
Z = 3*T0**2 * T1**2 + 2*T0*T1**3 + T0**2 * T1**2

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.8)

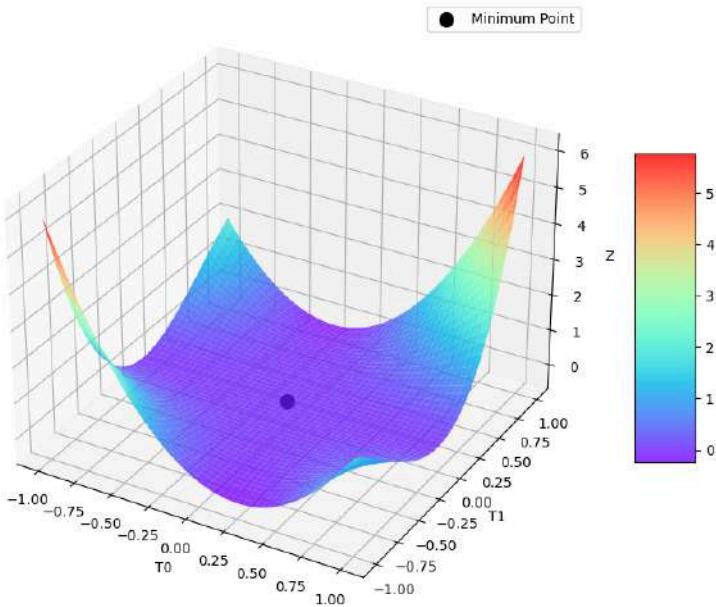
# Plot the minimum point
ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point', s=100)

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

# Show plot
plt.legend()
plt.show()

message: Local minimum reached (|pg| ~ 0)
success: True
status: 0
fun: 0.0
  x: [ 0.000e+00  0.000e+00]
  nit: 0
  jac: [ 0.000e+00  0.000e+00]
  nfev: 1
```



## Example11

```
In [37]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\17.jpg"
Image(filename=image_path, width=400, height=300)
```

```
Out[37]: J(x0,x1) = x1 sin(x0) - x0 cos(x1)
```

```
In [38]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def F_new(params):
    x0, x1 = params
    return x1 * np.sin(x0) - x0 * np.cos(x1)

# Define the Jacobian (gradient) of the function
def jac_F_new(params):
    x0, x1 = params
    dfdx0 = x1 * np.cos(x0) - np.cos(x1)
    dfdx1 = np.sin(x0) + x0 * np.sin(x1)
    return np.array([dfdx0, dfdx1])

# Perform optimization using the L-BFGS-B method with the gradient
try:
    result_new = minimize(F_new, x0=[1, 1], method='L-BFGS-B', jac=jac_F_new, options={'maxiter': 10000})
    print(result_new)
except Exception as e:
    print("Optimization failed:", e)

# Optimal parameters
optimal_params = result_new.x
Z_min = F_new(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(5, 15, 100)
t1 = np.linspace(5, 15, 100)
T0, T1 = np.meshgrid(t0, t1)
Z = T1 * np.sin(T0) - T0 * np.cos(T1)

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.8)

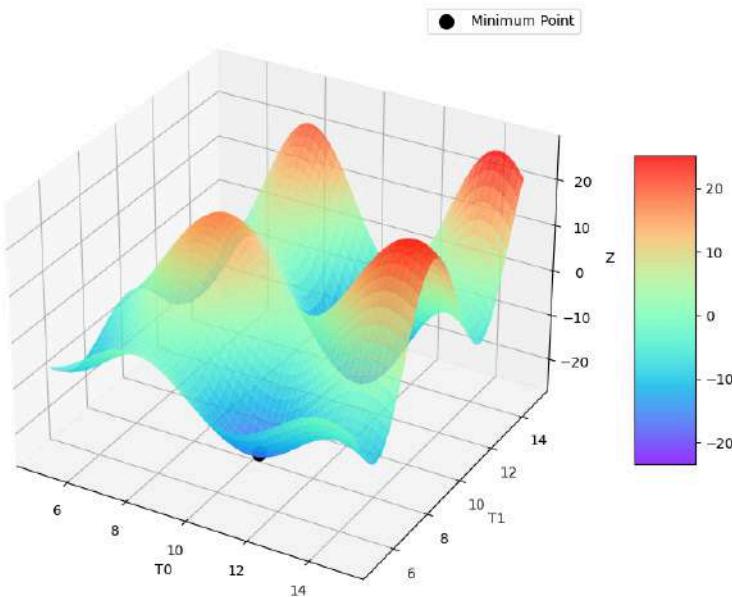
# Plot the minimum point if optimization succeeded
if result_new.success:
    ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point', s=100)
else:
    print("Could not plot minimum point because optimization did not succeed.")

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

# Show plot
plt.legend()
plt.show()

message: CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
success: True
status: 0
fun: -17.402246733958783
x: [ 1.115e+01  6.372e+00]
nit: 11
jac: [-7.400e-06 -4.197e-06]
nfev: 18
njev: 18
hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
```



## Example12

```
In [39]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\18.jpg"
Image(filename=image_path, width=300, height=200)

Out[39]: J(x0,x1) = sin(x0) cos(x1)
```

```
In [41]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def J(params):
    x0, x1 = params
    return np.sin(x0) * np.cos(x1)

# Define the Jacobian (gradient) of the function
def jac_J(params):
    x0, x1 = params
    dfdx0 = np.cos(x0) * np.cos(x1)
    dfdx1 = -np.sin(x0) * np.sin(x1)
    return np.array([dfdx0, dfdx1])

# Perform optimization using the trust-exact method with the gradient
try:
    result12 = minimize(J, x0=[1, 1], method='cg', jac=jac_J, options={'maxiter': 10000})
    print(result12)
except Exception as e:
    print("Optimization failed:", e)

# Optimal parameters
optimal_params = result12.x
Z_min12 = J(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(0, 4, 50)
t1 = np.linspace(0, 4, 50)
T0, T1 = np.meshgrid(t0, t1)
Z = np.sin(T0) * np.cos(T1)

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap="RdYlGn", alpha=0.2)

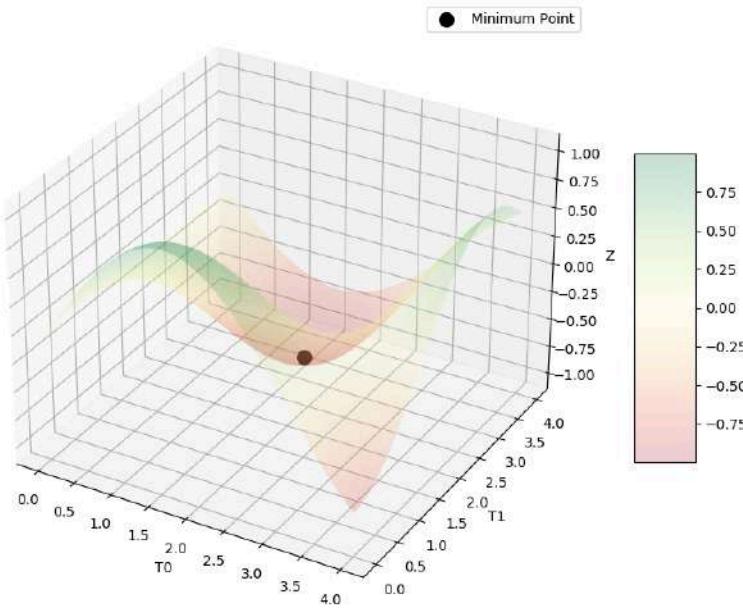
# Plot the minimum point if optimization succeeded
if result12.success:
    ax.scatter(optimal_params[0], optimal_params[1], Z_min12, color='k', label='Minimum Point', s=100)
else:
    print("Could not plot minimum point because optimization did not succeed.")

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

# Show plot
plt.legend()
plt.show()

message: Optimization terminated successfully.
success: True
status: 0
  fun: -1.0
    x: [ 1.571e+00  3.142e+00]
   nit: 5
   jac: [ 4.156e-10  1.032e-09]
  nfev: 10
  njev: 10
```



## Example13

```
In [42]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\19.jpg"
Image(filename=image_path, width=300, height=200)

Out[42]: J(x₀,x₁) = sin(x₀) sin(x₁)
```

```
In [45]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def F_new(params):
    x0, x1 = params
    return np.sin(x0) * np.sin(x1)

# Define the Jacobian (gradient) of the function
def jac_F_new(params):
    x0, x1 = params
    dfdx0 = np.cos(x0) * np.sin(x1)
    dfdx1 = np.sin(x0) * np.cos(x1)
    return np.array([dfdx0, dfdx1])

# Perform optimization using the L-BFGS-B method with the gradient
try:
    result_new = minimize(F_new, x0=[2, 1.5], method='Newton-CG', jac=jac_F_new, options={'maxiter': 10000})
    print(result_new)
except Exception as e:
    print("Optimization failed:", e)

# Optimal parameters
optimal_params = result_new.x
Z_min = F_new(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(0, 6, 50)
t1 = np.linspace(0, 4, 50)
T0, T1 = np.meshgrid(t0, t1)
Z = np.sin(T0) * np.sin(T1)

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.5)

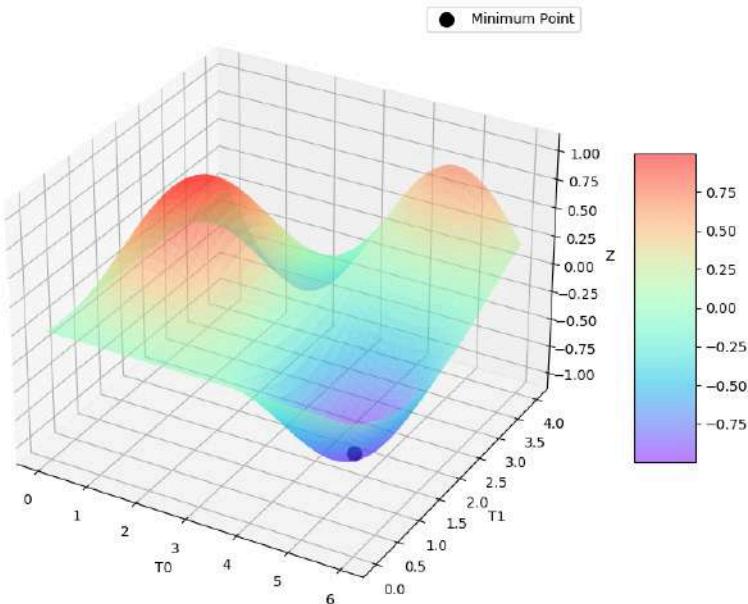
# Plot the minimum point if optimization succeeded
if result_new.success:
    ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point', s=100)
else:
    print("Could not plot minimum point because optimization did not succeed.")

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('T0')
ax.set_ylabel('T1')
ax.set_zlabel('Z')

# Show plot
plt.legend()
plt.show()

message: Optimization terminated successfully.
success: True
status: 0
fun: -1.0
x: [ 4.712e+00  1.571e+00]
nit: 5
jac: [ 2.854e-08  3.646e-08]
nfev: 8
njev: 17
nhev: 0
```



## Example14

```
In [46]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinelearning\machine learning\jozavat\advanced optimization methods\advanced\20.png"
Image(filename=image_path, width=300, height=200)
```

```
Out[46]:  $J(x_0, x_1) = \sin \sqrt{x_0^2 + x_1^2}$ 
```

```
In [47]: import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import minimize

def function(params):
    return np.sin(np.sqrt(params[0]**2 + params[1]**2))

# Use a different optimization method
result = minimize(function, x0=[0, 0], method='Nelder-Mead')
print(result)

# Create a grid of points
x = np.linspace(-10, 10, 400)
y = np.linspace(-10, 10, 400)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.viridis, edgecolor='none')

# Add the result of the optimization as a point on the surface
ax.scatter(result.x[0], result.x[1], function(result.x), color='r', s=100, label='Minimum')

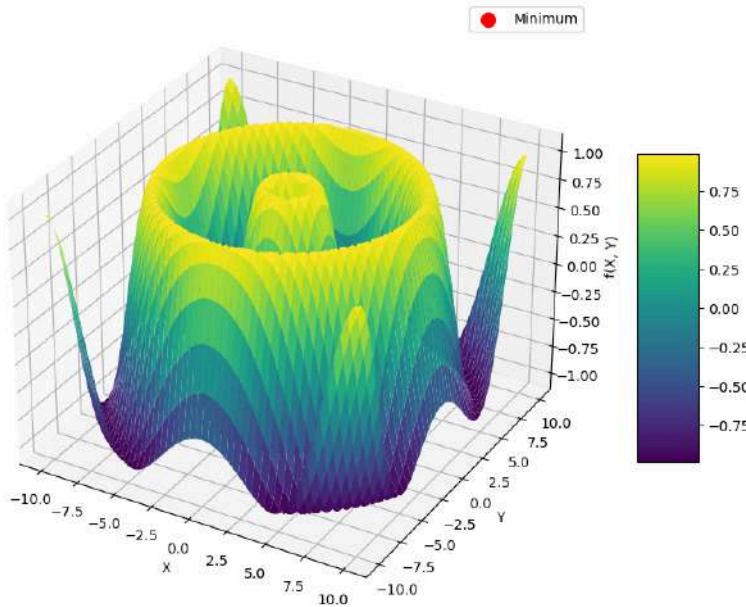
# Add Labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('f(X, Y)')
ax.set_title('3D Surface plot of f(X, Y) = sin(sqrt(X0**2 + X1**2))')
ax.legend()

# Add a color bar which maps values to colors
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

message: Optimization terminated successfully.  
success: True  
status: 0  
fun: 0.0  
x: [ 0.000e+00 0.000e+00]  
nit: 5  
nfev: 11  
final\_simplex: (array([[ 0.000e+00, 0.000e+00],  
[ 4.785e-05, 1.758e-05],  
[ 1.172e-05, 8.594e-05]]), array([ 0.000e+00, 5.098e-05, 8.673e-05]))

3D Surface plot of  $f(X, Y) = \sin(\sqrt{X_0^2 + X_1^2})$



## Example15

```
In [48]: from IPython.display import Image
# Path to the image file
image_path = r"D:\AI_Machinlearning\machine learning\jozavat\advanced optimization methods\advanced\22.jpg"
Image(filename=image_path, width=300, height=200)

Out[48]: J(\theta_0, \theta_1) = \sin(\theta_0) + \cos(\theta_1)
```

```
In [51]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the function
def F_new(param):
    return np.sin(param[0]) + np.cos(param[1])

# Perform optimization using the BFGS method
result_new = minimize(F_new, x0=[0,0], method='BFGS')
print(result_new)

# Optimal parameters
optimal_params = result_new.x
Z_min = F_new(optimal_params)

# Generate meshgrid for plotting
t0 = np.linspace(-10, 1, 100)
t1 = np.linspace(-10, 1, 100)
T0, T1 = np.meshgrid(t0, t1)
Z = np.sin(T0) + np.cos(T1)

# Plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow, alpha=0.9)

# Plot contours
cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)

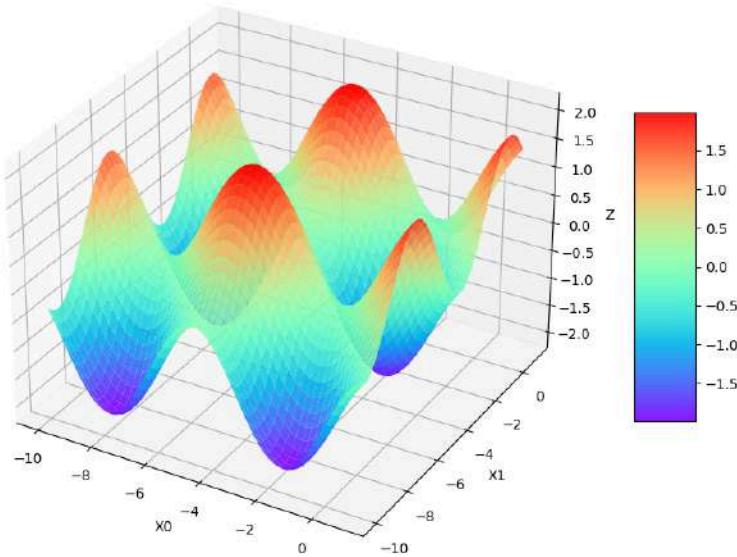
# Plot the minimum point
ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='k', label='Minimum Point', s=100)

# Color bar
fig.colorbar(surf, shrink=0.5, aspect=5)

# Axes Labels
ax.set_xlabel('X0')
ax.set_ylabel('X1')
ax.set_zlabel('Z')

# Show plot
plt.show()

message: Optimization terminated successfully.
success: True
status: 0
fun: -8.548717289613705e-15
x: [-1.571e+00  1.310e-07]
nit: 4
jac: [ 0.000e+00 -1.416e-07]
hess_inv: [[ 1.000e+00 -7.105e-05]
           [-7.105e-05  1.000e+00]]
nfev: 18
njev: 6
```



## Example16

```
In [54]: import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the cost function
def cost_function(params, X, y, Lambda):
    m = len(y)
    weights = np.array(params[:-1])
    bias = params[-1]
    predictions = np.dot(X, weights) + bias
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (Lambda / (2 * m)) * np.sum(weights ** 2)
    return cost

# Define the gradient of the cost function
def gradient(params, X, y, Lambda):
    m = len(y)
    weights = np.array(params[:-1])
    bias = params[-1]
    predictions = np.dot(X, weights) + bias
    weight_derivative = (1 / m) * (np.dot(X.T, (predictions - y)) + Lambda * weights)
    bias_derivative = (1 / m) * np.sum(predictions - y)
    return np.append(weight_derivative, bias_derivative)

# Example data (replace with your actual data)
X = np.array([[1], [2], [3], [4]]) # Single feature for 2D plot
y = np.array([2, 3, 4, 5])
Lambda = 0.1

# Initial guess for weights and bias
initial_params = np.zeros(X.shape[1] + 1)

# Record the optimization path
history = []

def callback(params):
    history.append(params)

# Perform optimization using the BFGS method
result = minimize(cost_function, initial_params, args=(X, y, Lambda), method='BFGS', jac=gradient, callback=callback)

print(result)

# Convert history to a numpy array for easy indexing
history = np.array(history)

# Plotting the cost function in 3D
w0_vals = np.linspace(-10, 10, 100)
b_vals = np.linspace(-10, 10, 100)
w0, b = np.meshgrid(w0_vals, b_vals)

# Calculate cost values
Z = np.array([[cost_function([w0, b], X, y, Lambda) for w0 in w0_vals] for b in b_vals])

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot surface
surf = ax.plot_surface(w0, b, Z, cmap=plt.cm.rainbow, alpha=0.7)

# Plot contours
cset = ax.contour(w0, b, Z, 20, zdir='z', offset=np.min(Z), cmap=plt.cm.rainbow)

# Plot the optimization path
path_costs = np.array([cost_function(params, X, y, Lambda) for params in history])
ax.plot(history[:, 0], history[:, 1], path_costs, 'o', label='Optimization Path')

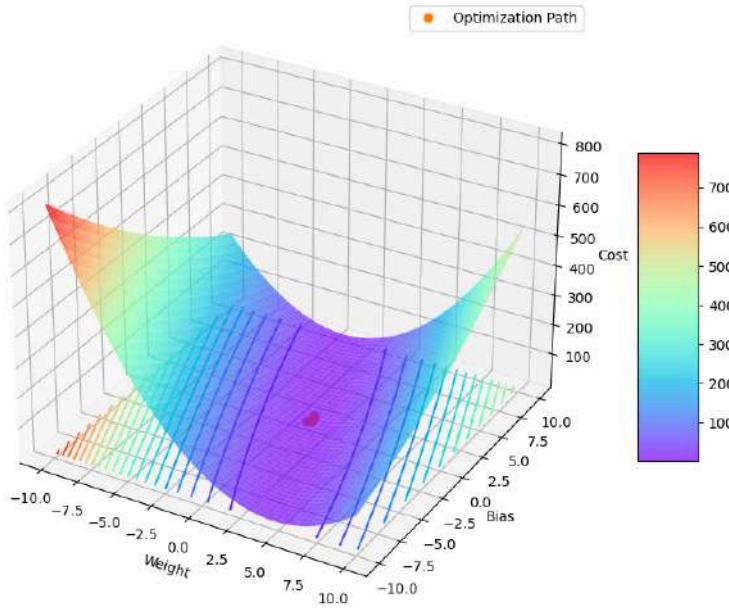
fig.colorbar(surf, shrink=0.5, aspect=5)
ax.set_title('Cost Function Surface and Optimization Path')
ax.set_xlabel('Weight')
ax.set_ylabel('Bias')
ax.set_zlabel('Cost')
plt.legend()
plt.show()
```

```

message: Optimization terminated successfully.
success: True
status: 0
fun: 0.012254901960784314
x: [ 9.804e-01  1.049e+00]
nit: 7
jac: [-5.899e-11  1.096e-10]
hess_inv: [[ 7.842e-01 -1.960e+00]
           [-1.960e+00  5.901e+00]]
nfev: 8
njev: 8

```

Cost Function Surface and Optimization Path



```
In [53]: np.zeros(5 + 1)
Out[53]: array([0., 0., 0., 0., 0.])
```

## LinearRegression with Advanced Optimization Methods

```

In [55]:
import numpy as np
from scipy.optimize import minimize
from sklearn.preprocessing import StandardScaler

class LinearRegression:
    def __init__(self, Lambda=0):
        self.Lambda = Lambda
        self.weights = None
        self.bias = None
        self.scaler = StandardScaler()

    def cost_function(self, params, X, y):
        m = len(y)
        weights = np.array(params[:-1])
        bias = params[-1]
        predictions = np.dot(X, weights) + bias
        cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (self.Lambda / (2 * m)) * np.sum(weights ** 2)
        return cost

    def gradient(self, params, X, y):
        m = len(y)
        weights = np.array(params[:-1])
        bias = params[-1]
        predictions = np.dot(X, weights) + bias
        error = predictions - y.flatten() # Ensure y is 1D
        gradient_weights = (1 / m) * np.dot(X.T, error) + (self.Lambda / m) * weights
        gradient_bias = (1 / m) * np.sum(error)
        gradient = np.append(gradient_weights, gradient_bias)
        return gradient

    def fit(self, X, y, max_iter=2000, method='L-BFGS-B'):
        # Standardize the data
        X = self.scaler.fit_transform(X)
        y = y.flatten() # Ensure y is 1D

        m, n = X.shape
        initial_params = np.zeros(n + 1) # Initial weights and bias

        # Use minimize from scipy.optimize to minimize the cost function
        result = minimize(self.cost_function, initial_params, args=(X, y), jac=self.gradient, method=method, options={'maxiter': max_iter})

        self.weights = result.x[:-1]
        self.bias = result.x[-1]

        if not result.success:
            print("Optimization failed:", result.message)
        else:
            print("Optimization succeeded.")
            print(result.x)

        return result

    def predict(self):
        X = self.scaler.transform(X)
        return np.dot(X, self.weights) + self.bias

```

```
In [56]:
import numpy as np
import pandas as pd
data=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\sklearn\en.csv")
data.tail(10)
```

```
Out[56]:
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
758	0.66	759.5	318.5	220.5	3.5	4	0.4	5	14.92
759	0.66	759.5	318.5	220.5	3.5	5	0.4	5	15.16
760	0.64	784.0	343.0	220.5	3.5	2	0.4	5	17.69
761	0.64	784.0	343.0	220.5	3.5	3	0.4	5	18.19
762	0.64	784.0	343.0	220.5	3.5	4	0.4	5	18.16
763	0.64	784.0	343.0	220.5	3.5	5	0.4	5	17.88
764	0.62	808.5	367.5	220.5	3.5	2	0.4	5	16.54
765	0.62	808.5	367.5	220.5	3.5	3	0.4	5	16.44
766	0.62	808.5	367.5	220.5	3.5	4	0.4	5	16.48
767	0.62	808.5	367.5	220.5	3.5	5	0.4	5	16.64

```
In [57]: X = np.array(data.drop(["Y1"], axis=1))
y = np.array(data["Y1"])
y = y.reshape(-1, 1)
```

```
In [58]: # train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2123)
```

```
In [59]: model = LinearRegression(Lambda=0.1)
result = model.fit(X_train, y_train)

# predict
predictions = model.predict(X_test)

Optimization succeeded.
[-6.20239963 -3.21669825  0.7314846 -3.4900771   7.82823374 -0.05395602
 2.56781995  0.40623813 22.48795044]
```

```
In [60]: predict_test = model.predict(X_test)
predict_train = model.predict(X_train)

from sklearn.metrics import r2_score
print(f" the test accuracy is: {r2_score(predict_test, y_test)*100} and the train accuracy is:{r2_score(predict_train, y_train)*100}")

the test accuracy is: 91.78929394341161 and the train accuracy is:90.67924737645198
```

## PolynomialRegression with Advanced Optimization Methods

```
In [61]: import numpy as np
from scipy.optimize import minimize

class PolynomialRegression:

    def __init__(self, degree, iterations, Lambda=0.01):
        self.degree = degree
        self.iterations = iterations
        self.Lambda = Lambda
        self.costs = []
        self.weights = None
        self.bias = None

    def normalize(self, X, method=None):
        if method is None or method == "zscore":
            return (X - X.mean(axis=0)) / X.std(axis=0)
        elif method == "maxabs":
            return X / np.abs(X).max(axis=0)
        elif method == "minmax":
            return (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

    def transform(self, X):
        X_norm = self.normalize(X)
        num_samples, num_features = X_norm.shape
        X_transform = np.empty((num_samples, 0))
        for i in range(1, self.degree + 1):
            X_transform = np.concatenate((X_transform, np.sin(2 * i * X_norm)), axis=1)
        return X_transform

    def cost_function(self, params, X, y):
        m = len(y)
        weights = np.array(params[:-1]).reshape(-1, 1)
        bias = params[-1]
        predictions = np.dot(X, weights) + bias
        cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (self.Lambda / (2 * m)) * np.sum(weights ** 2)
        return cost

    def gradient(self, params, X, y):
        m = len(y)
        weights = np.array(params[:-1]).reshape(-1, 1)
        bias = params[-1]
        predictions = np.dot(X, weights) + bias
        error = predictions - y
        weight_gradient = (1 / m) * np.dot(X.T, error) + (self.Lambda / m) * weights
        bias_gradient = (1 / m) * np.sum(error)
        gradient = np.append(weight_gradient.flatten(), bias_gradient)
        return gradient

    def fit(self, X, y):
        X_transform = self.transform(X)
        m, n = X_transform.shape

        initial_params = np.zeros(n + 1) # Initial weights and bias

        # Use minimize from scipy.optimize to minimize the cost function
        result = minimize(self.cost_function, initial_params, args=(X_transform, y), jac=self.gradient, method='L-BFGS-B', options={'maxiter': self.iterations})

        self.weights = result.x[:-1].reshape(-1, 1)
        self.bias = result.x[-1]

        if not result.success:
            print("Optimization failed:", result.message)
        else:
            print("Optimization succeeded.")

        return result

    def predict(self, X):
        X_transform = self.transform(X)
        return np.dot(X_transform, self.weights) + self.bias

    def get_weights(self):
        return self.weights
```

```

def get_bias(self):
    return self.bias

def get_costs(self):
    return self.costs

In [62]: model2 = PolynomialRegression(degree=2, iterations=1000, Lambda=0.1)
result2 = model.fit(X_train, y_train)

# To predict
predictions = model.predict(X_test)

Optimization succeeded.
[-6.20239963 -3.21669825  0.7314846 -3.4900771  7.82823374 -0.05395602
 2.56781995  0.40623813 22.48795044]

In [63]: from sklearn.metrics import r2_score
print(f" the test accuracy is: {r2_score(predictions , y_test)*100} ")

the test accuracy is: 91.78929394341161

```

## LogisticRegression with Advanced Optimization Methods

```

import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt

class LogisticRegression:
    def __init__(self, degree, iterations, Lambda=0.01):
        self.degree = degree
        self.iterations = iterations
        self.Lambda = Lambda
        self.classes = None
        self.classifiers = {}

    def normalize(self, X):
        mean = X.mean(axis=0)
        std = X.std(axis=0)
        std[std == 0] = 1 # To avoid division by zero, set zero std to one
        return (X - mean) / std

    def transform(self, X):
        X_norm = self.normalize(X)
        num_samples, num_features = X_norm.shape
        X_transform = np.empty((num_samples, 0))
        for i in range(1, self.degree + 1):
            X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
        return X_transform

    def sigmoid(self, Z):
        return 1. / (1. + np.exp(-Z))

    def cost_function(self, params, X, y):
        m = len(y)
        weights = np.array(params[:-1]).reshape(-1, 1)
        bias = params[-1]
        Z = np.dot(X, weights) + bias
        y_hat = self.sigmoid(Z)
        y_hat = np.clip(y_hat, 1e-15, 1 - 1e-15)
        cost = - np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
        cost += (self.Lambda / (2 * m)) * np.sum(weights ** 2)
        return cost

    def gradient(self, params, X, y):
        m = len(y)
        weights = np.array(params[:-1]).reshape(-1, 1)
        bias = params[-1]
        Z = np.dot(X, weights) + bias
        y_hat = self.sigmoid(Z)
        error = y_hat - y
        weight_gradient = (np.dot(X.T, error) + self.Lambda * weights) / m
        bias_gradient = np.sum(error) / m
        gradient = np.append(weight_gradient.flatten(), bias_gradient)
        return gradient

    def train_binary_classifier(self, X, y):
        X_transform = self.transform(X)
        m, n = X_transform.shape

        initial_params = np.zeros(n + 1) # Initial weights and bias

        # Use minimize from scipy.optimize to minimize the cost function
        result = minimize(self.cost_function, initial_params, args=(X_transform, y), jac=self.gradient, method='L-BFGS-B', options={'maxiter': self.iterations})

        weights = result.x[:-1].reshape(-1, 1)
        bias = result.x[-1]

        if not result.success:
            print("Optimization failed:", result.message)
        else:
            print("Optimization succeeded.")

        return weights, bias

    def fit(self, X, y):
        self.classes = np.unique(y)
        for cls in self.classes:
            y_binary = (y == cls).astype(int).reshape(-1, 1)
            weights, bias = self.train_binary_classifier(X, y_binary)
            self.classifiers[cls] = (weights, bias)

    def predict_proba(self, X):
        X_transform = self.transform(X)
        probabilities = np.zeros((X_transform.shape[0], len(self.classes)))

        for i, cls in enumerate(self.classes):
            weights, bias = self.classifiers[cls]
            Z = np.dot(X_transform, weights) + bias
            probabilities[:, i] = self.sigmoid(Z).ravel()

        return probabilities

    def predict(self, X):
        probabilities = self.predict_proba(X)
        predictions = np.argmax(probabilities, axis=1)
        return self.classes[predictions]

    def get_weights(self):
        return {cls: self.classifiers[cls][0] for cls in self.classes}

    def get_biases(self):
        return {cls: self.classifiers[cls][1] for cls in self.classes}

In [65]: from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

```

```
iris = load_iris()
X = iris.data
y = iris.target

X_train , X_test , y_train, y_test = train_test_split(X , y , random_state= 12 , test_size=0.2)

model = LogisticRegression(degree=2, iterations=1000, Lambda=0.1)
result = model.fit(X_train, y_train)

# Make predictions
predict_test = model.predict(X_test)
predict_train= model.predict(X_train)

#score
from sklearn.metrics import accuracy_score

print(f" train accuracy is: {accuracy_score(predict_train , y_train)*100} and test accuracy is: {accuracy_score(predict_test , y_test)*100}")

Optimization succeeded.
Optimization succeeded.
Optimization succeeded.
train accuracy is: 97.5 and test accuracy is: 93.33333333333333
```

# Machine learning

## Regularization (Overfitting)

Morteza khorsand



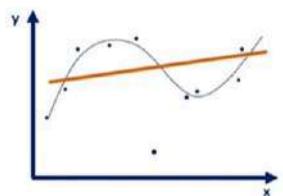
### Over Fitting

2

#### Regularization

Ockham's razor (also spelled Occam's razor) is the idea that, in trying to understand something, getting unnecessary information out of the way is the fastest way to the truth or to the best explanation.

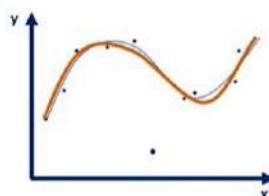
An **underfitted** model



Doesn't capture any logic

- High loss
- Low accuracy

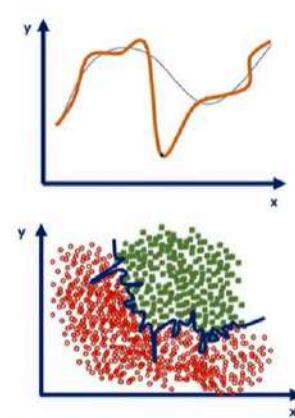
A **good** model



Captures the underlying logic of the dataset

- Low loss
- High accuracy

An **overfitted** model

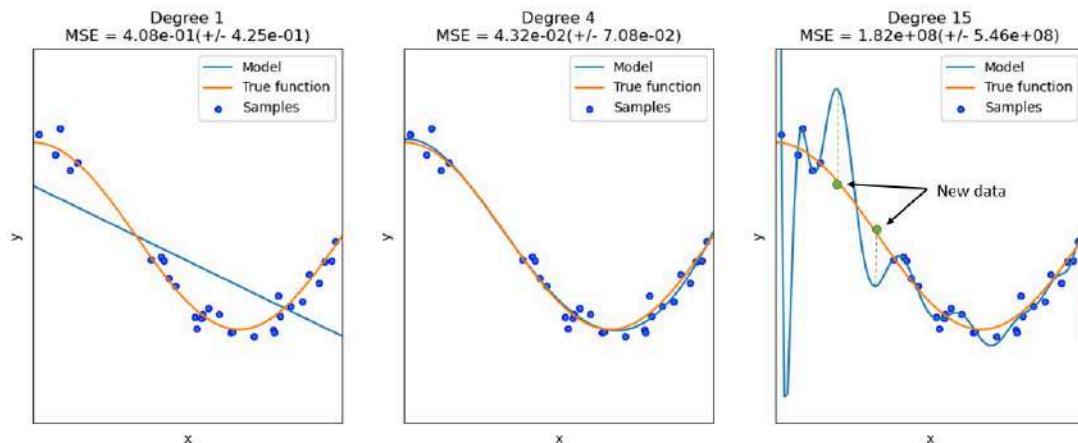


Captures all the noise, thus "missed the point"

- Low loss
- Low accuracy

### Overfitting

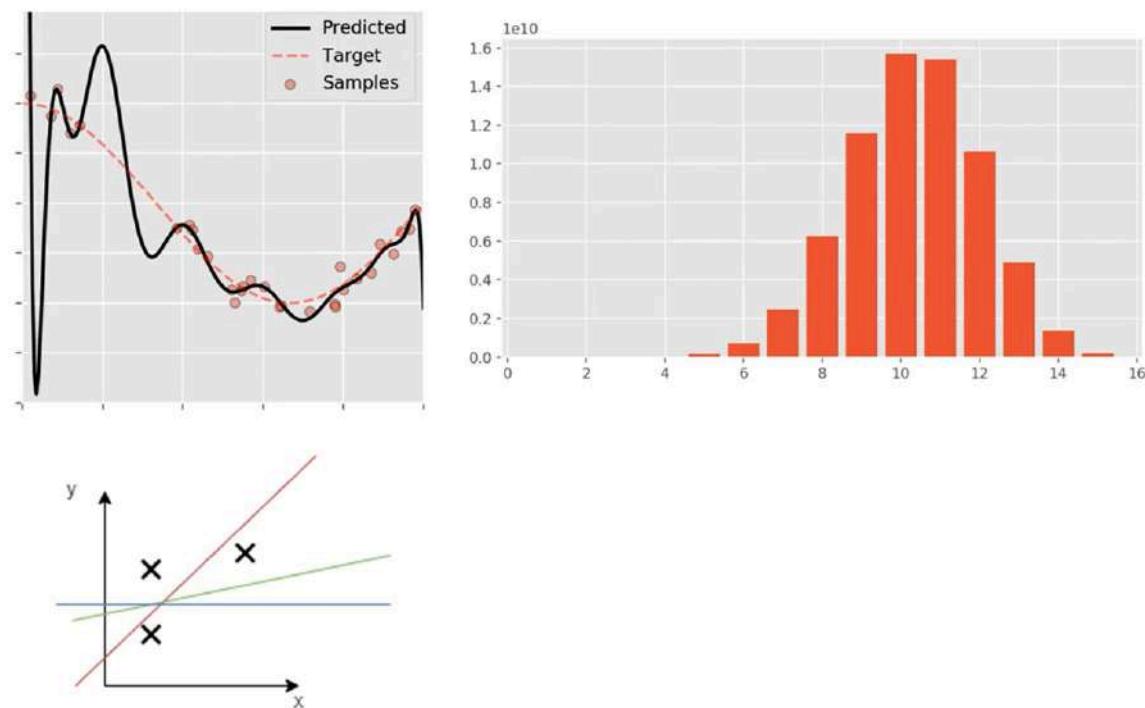
Occam's razor: Entities are not to be multiplied without necessity



*K-fold cross-validation* is one of the most popular techniques commonly used to detect overfitting.

We split the data points into  $k$  equally sized subsets in  $K$ -folds cross-validation, called "folds." One split subsets act as the testing set, and the remaining folds will train the model.

### Overfitting



## Over Fitting

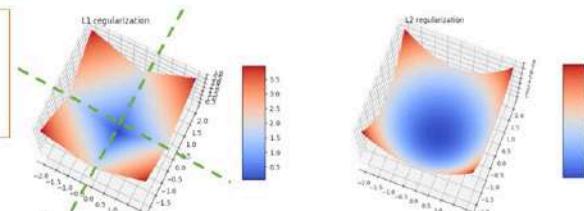
### ■ Regularization ( shrinkage , weight decay)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(x^{(i)}, y^{(i)}) + \lambda R(\theta)$$

$\lambda$  : hyper parameter from 0 to  $+\infty$

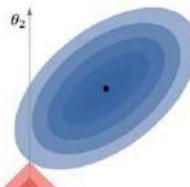
$R(\theta)$  : excluded from  $\theta_0$

*optimum points*



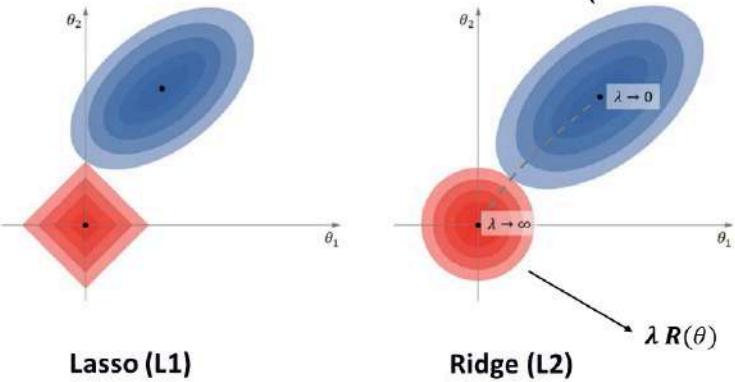
### ■ Lasso (L1)

$$R(\theta) = \sum_j |\theta_j| = \|\theta\|_1$$



### ■ Ridge (L2)

$$R(\theta) = \sum_{j=1}^n \theta_j^2 = \|\theta\|_2^2$$

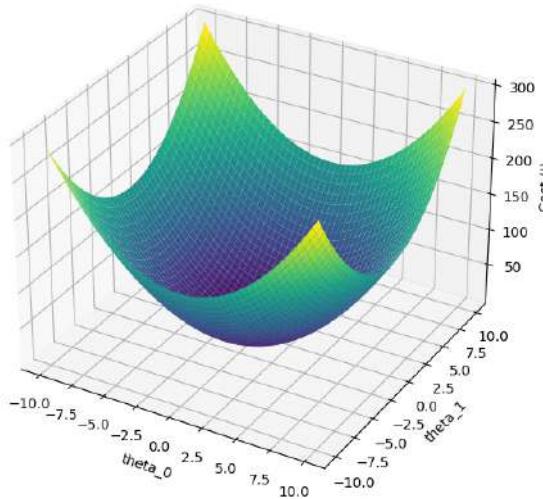


Lasso (L1)

Ridge (L2)

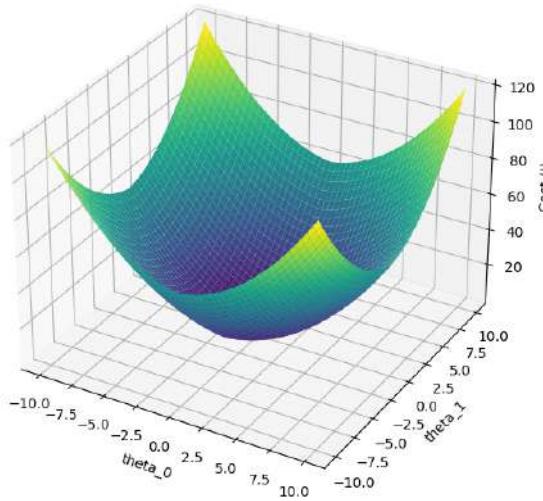
```
In [1]: M
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Generate data for visualization
5 theta_0 = np.linspace(-10, 10, 400)
6 theta_1 = np.linspace(-10, 10, 400)
7 theta_0, theta_1 = np.meshgrid(theta_0, theta_1)
8
9 # Mean Squared Error (MSE) component
10 mse = (theta_0**2 + theta_1**2) / 2
11
12 # Ridge Regularization component (with Lambda = 1 for simplicity)
13 ridge_reg = theta_0**2 + theta_1**2
14
15 # Combined cost function
16 J = mse + ridge_reg
17
18 # Plotting the combined surface
19 fig = plt.figure(figsize=(10, 7))
20 ax = fig.add_subplot(111, projection='3d')
21 ax.plot_surface(theta_0, theta_1, J, cmap='viridis')
22
23 ax.set_xlabel('theta_0')
24 ax.set_ylabel('theta_1')
25 ax.set_zlabel('Cost (J)')
26 ax.set_title('Combined MSE and Ridge Regularization Cost Function')
27
28 plt.show()
29
```

Combined MSE and Ridge Regularization Cost Function



```
In [2]: M
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Generate data for visualization
5 theta_0 = np.linspace(-10, 10, 400)
6 theta_1 = np.linspace(-10, 10, 400)
7 theta_0, theta_1 = np.meshgrid(theta_0, theta_1)
8
9 # Mean Squared Error (MSE) component
10 mse = (theta_0**2 + theta_1**2) / 2
11
12 # Lasso Regularization component (with Lambda = 1 for simplicity)
13 lasso_reg = np.abs(theta_0) + np.abs(theta_1)
14
15 # Combined cost function
16 J = mse + lasso_reg
17
18 # Plotting the combined surface
19 fig = plt.figure(figsize=(10, 7))
20 ax = fig.add_subplot(111, projection='3d')
21 ax.plot_surface(theta_0, theta_1, J, cmap='viridis')
22
23 ax.set_xlabel('theta_0')
24 ax.set_ylabel('theta_1')
25 ax.set_zlabel('Cost (J)')
26 ax.set_title('Combined MSE and Lasso Regularization Cost Function')
27
28 plt.show()
29
```

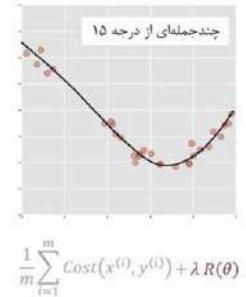
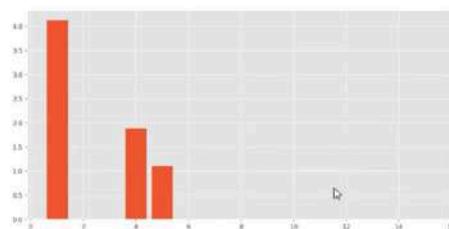
Combined MSE and Lasso Regularization Cost Function



## Over Fitting

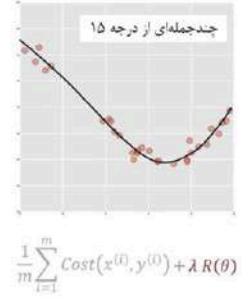
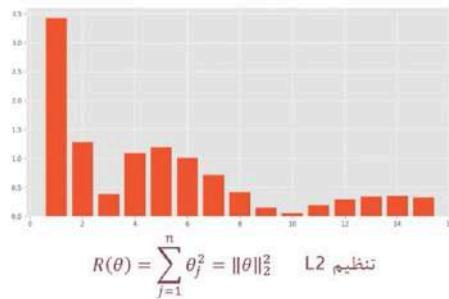
### ■ Lasso (L1)

$$R(\theta) = \sum_{j=1}^n |\theta_j| = \|\theta\|_1$$



### ■ Ridge (L2)

$$R(\theta) = \sum_{j=1}^n \theta_j^2 = \|\theta\|_2^2$$



## Over Fitting

### ■ Cost function in polynomial regression

$$J(\theta) = \frac{1}{2} \left[ \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

$$J(\theta) = \frac{1}{2} \left[ \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right]$$

Goal : minimize  $J(\theta)$    Gradient descent    $\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$

$$\theta_0 := \theta_0 - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \quad j = 0$$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \lambda \theta_j \quad j = (1, 2, 3, \dots, n)$$

## Over Fitting

### Gradient descent

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \lambda \theta_j \quad j = (1, 2, 3, \dots, n)$$

$$\theta_j := \theta_j (1 - \alpha \lambda) - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

### Normal equation with regularization

$$\theta = (X^T X)^{-1} X^T y$$

$$\theta = X^+ y$$

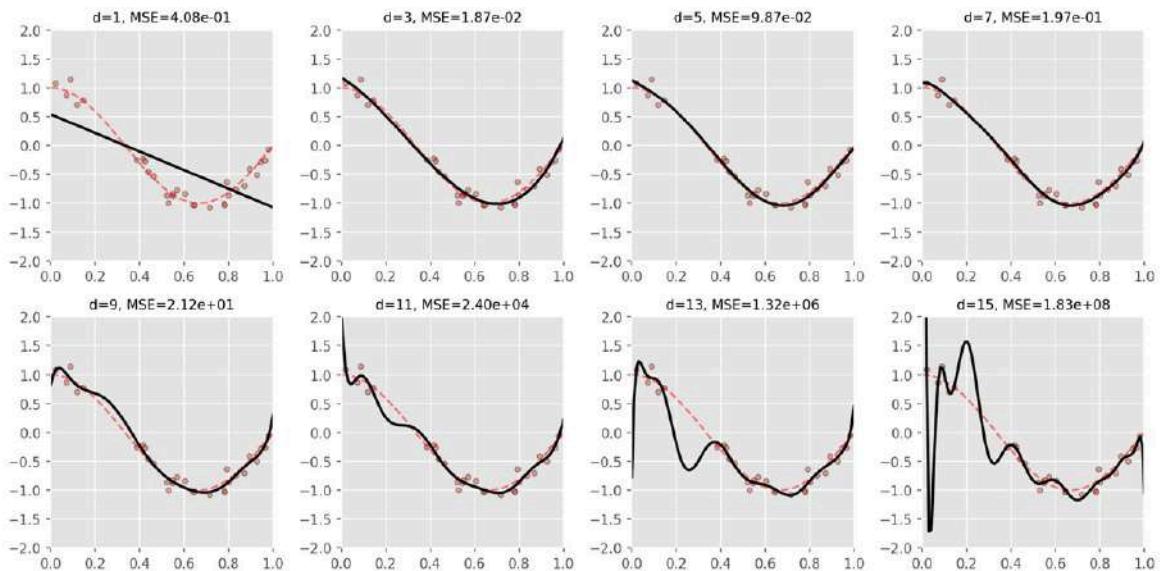
$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

Tikhonov regularization

$$\theta = \left( X^T X + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} \right) X^T y$$

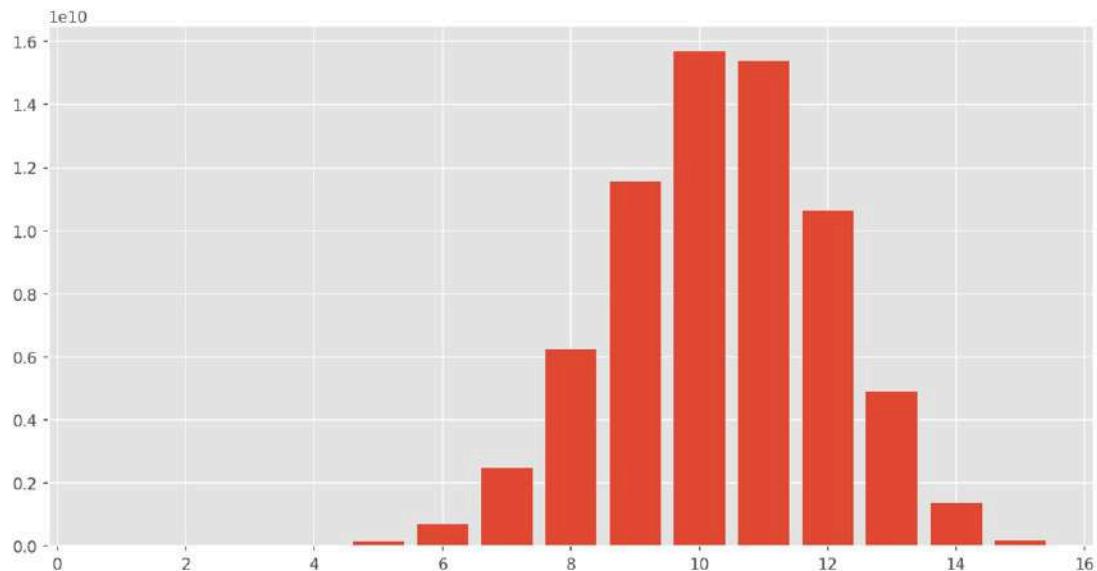
## Over Fitting

### degrees = [1, 3, 5, 7, 9, 11, 13, 15]



## Over Fitting

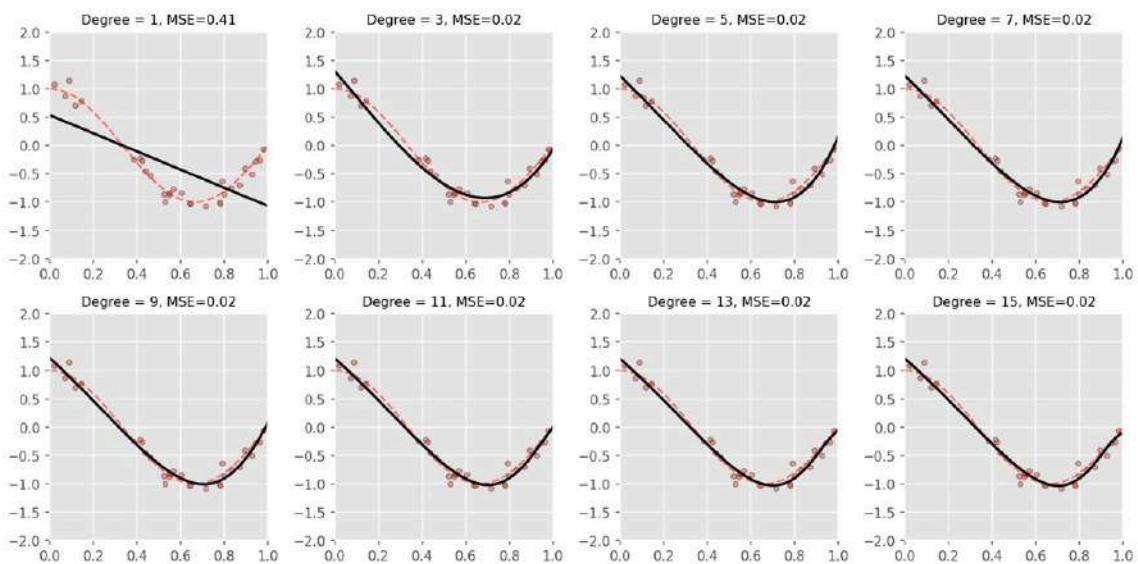
- Coefficients without regularization



## Over Fitting

- L2-Regularizarion (Ridge)

$$\lambda R(\theta) = \lambda \sum_{j=1}^n \theta_j^2 = \lambda \|\theta\|_2^2 \quad \lambda = 1e-2$$

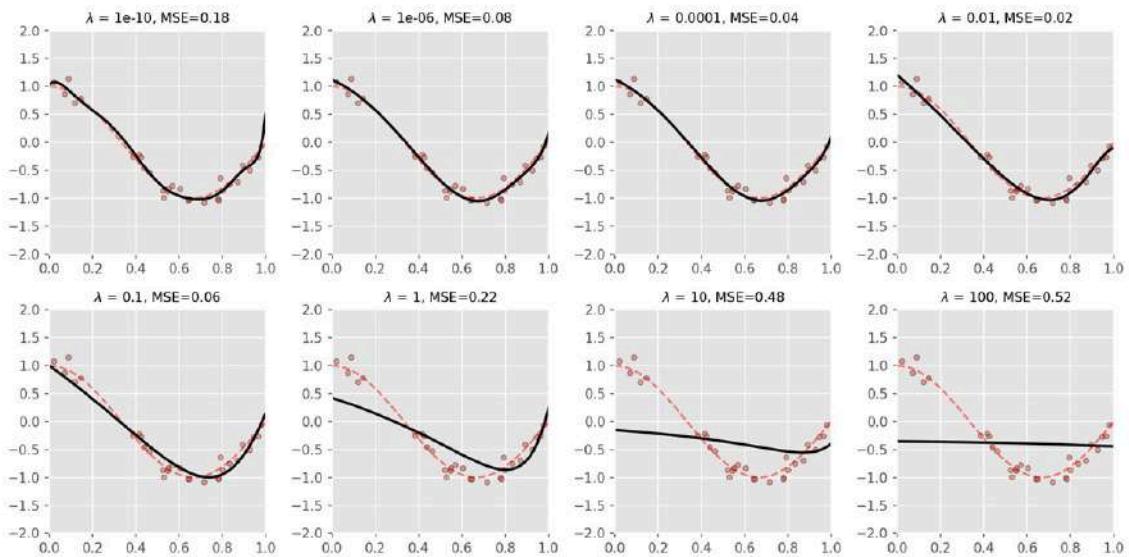


## Over Fitting

### L2-Regularization (Ridge)

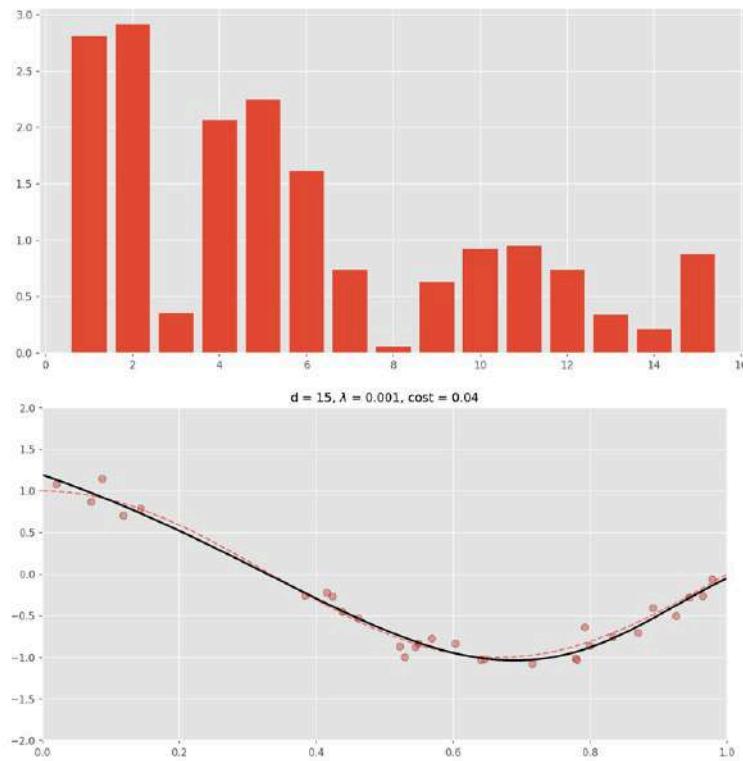
$$\lambda R(\theta) = \lambda \sum_{j=1}^n \theta_j^2 = \lambda \|\theta\|_2^2$$

$\lambda = [1e-10, 1e-6, 1e-4, 1e-2, 1e-1, 1, 10, 100]$



## Over Fitting

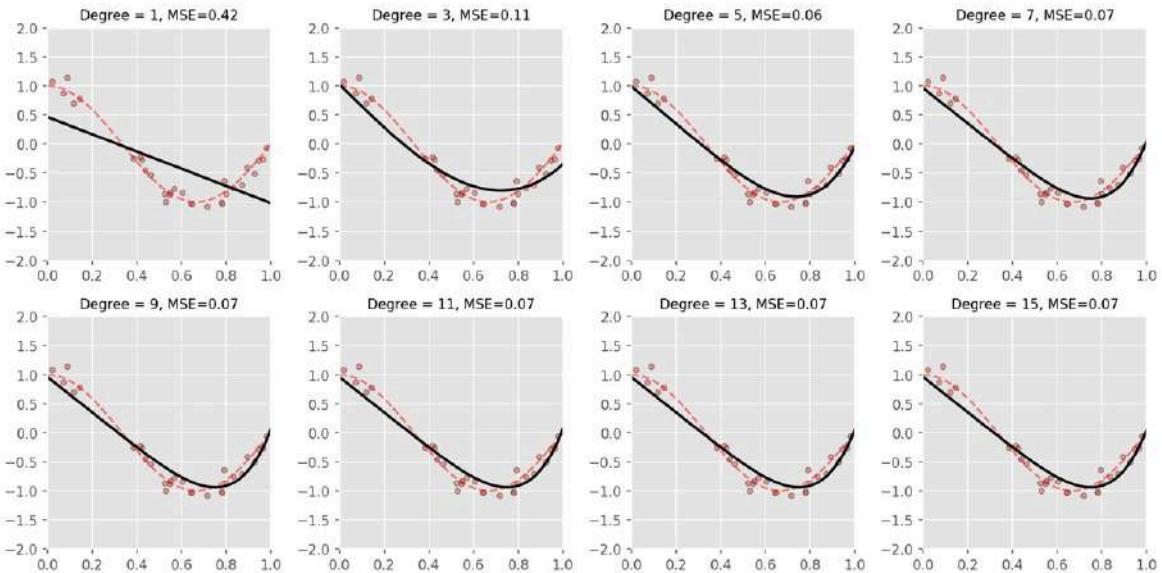
### Coefficients with L2 regularization



## Over Fitting

### L1-Regularization (Lasso)

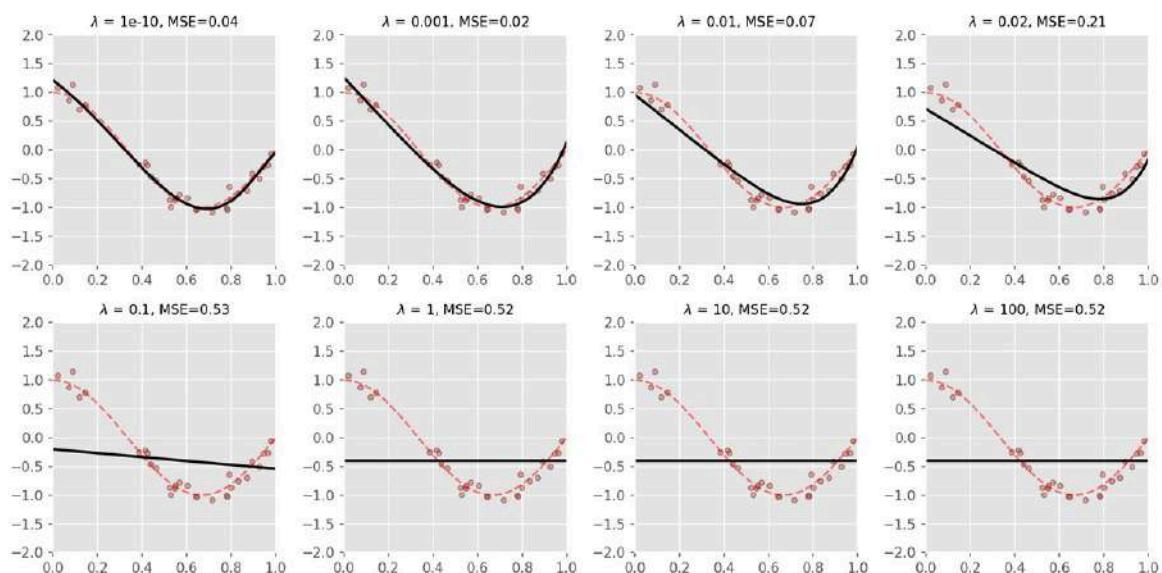
$$\lambda R(\theta) = \lambda \sum_{j=1}^n |\theta_j| = \lambda \|\theta\|_1 \quad \lambda = 1e-2$$



## Over Fitting

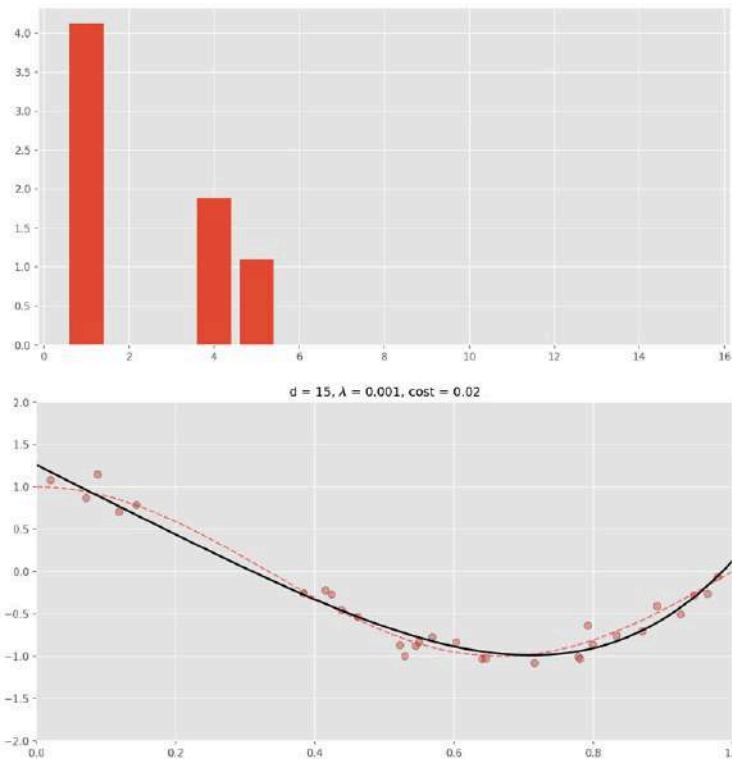
### L1-Regularization (Lasso)

$$\lambda R(\theta) = \lambda \sum_{j=1}^n |\theta_j| = \lambda \|\theta\|_1 \quad \lambda = [1e-10, 1e-6, 1e-4, 1e-2, 1e-1, 1, 10, 100]$$



## Over Fitting

### Coefficients with L1 regularization



## Over Fitting

10

### Regularization of logistic regression

$$J(\theta) = -\sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} + \frac{1}{2} \lambda \sum_{j=1}^n \theta_j^2 \quad j = (1, 2, 3, \dots, n)$$

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

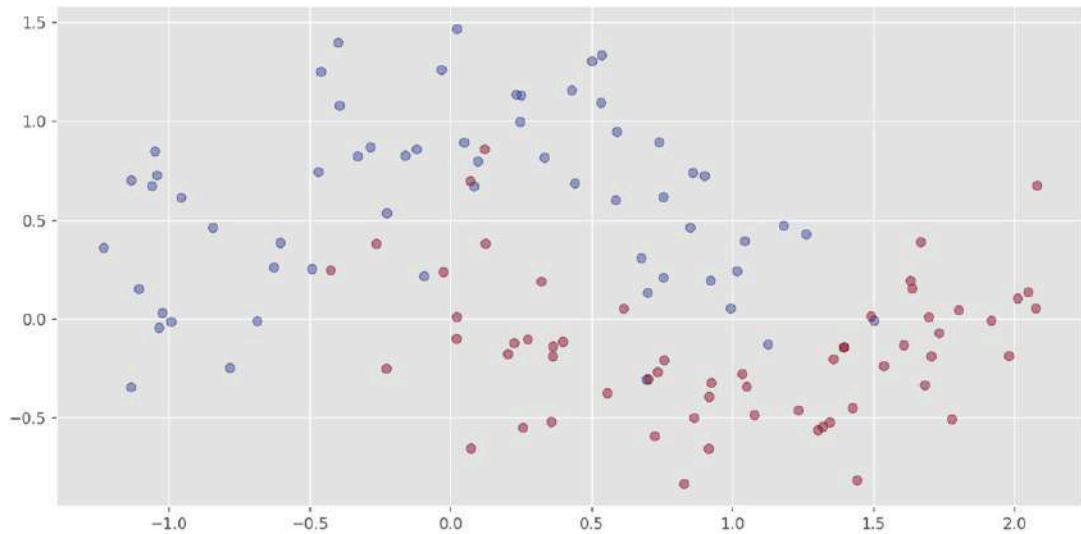
$$\theta_0 := \theta_0 - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \quad j = 0$$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \lambda \theta_j \quad j = (1, 2, 3, \dots, n)$$

$$h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$$

## Over Fitting

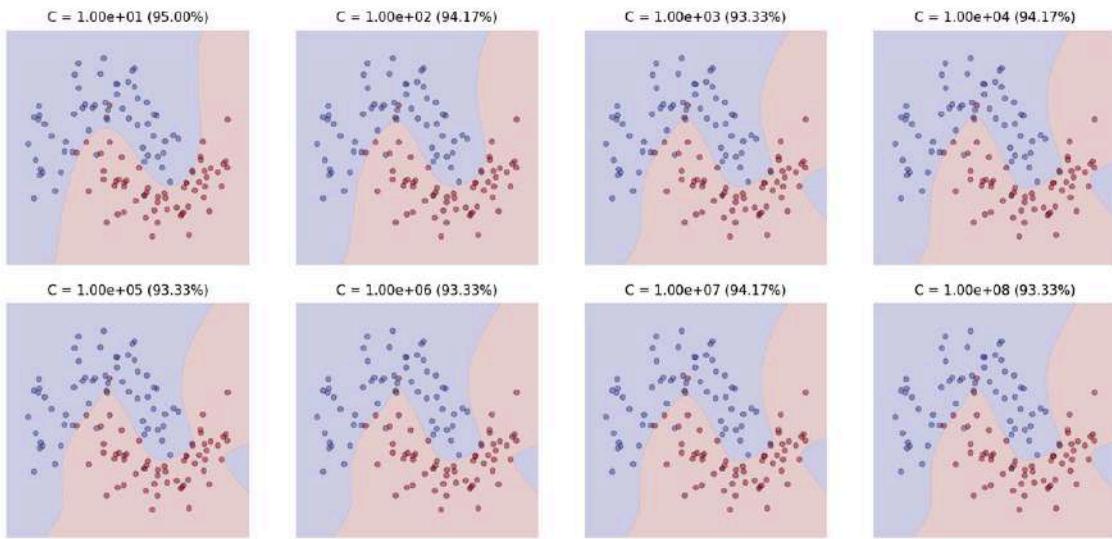
### Classification with Regularization



## Over Fitting

### Classification with Regularization

```
degree = 7  
coeffs = [1e1, 1e2, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8]
```



In [21]:

```
1 import numpy as np
2 from sklearn.metrics import r2_score
3
4 class RegularizedLinearRegression:
5     def __init__(self, iterations=100000, learning_rate=0.0001, stopping_threshold=1e-6, Lambda=0.01):
6         self.iterations = iterations
7         self.learning_rate = learning_rate
8         self.stopping_threshold = stopping_threshold
9         self.Lambda = Lambda
10        self.costs = []
11        self.weights = None
12        self.bias = None
13
14    def normalize(self, X):
15        """ Normalizes the feature matrix X """
16        mean = np.mean(X, axis=0)
17        std = np.std(X, axis=0)
18        return (X - mean) / std
19
20    def train(self, X, y):
21        X = self.normalize(X) # Normalize the features
22        m, n = X.shape
23        self.weights = np.random.rand(n, 1)
24        self.bias = np.zeros([1, 1])
25        previous_cost = None
26
27        for i in range(self.iterations):
28            # Making predictions
29            prediction = np.dot(X, self.weights) + self.bias
30
31            # Calculating the current cost with L2 regularization
32            current_cost = (1 / (2 * m)) * (np.sum((prediction - y) ** 2) + \
33                                         (self.Lambda / (2 * m)) * np.sum(self.weights ** 2))
34
35            # If the change in cost is less than or equal to stopping_threshold, stop gradient descent
36            if previous_cost and abs(previous_cost - current_cost) <= self.stopping_threshold:
37                break
38
39            previous_cost = current_cost
40            self.costs.append(current_cost)
41
42            # Calculating the gradients with L2 regularization
43            weight_derivative = (1 / m) * (np.dot(X.T, (prediction - y)) + self.Lambda * self.weights)
44            bias_derivative = (1 / m) * np.sum(prediction - y)
45
46            # Updating weights and bias
47            self.weights -= self.learning_rate * weight_derivative
48            self.bias -= self.learning_rate * bias_derivative
49
50
51    def predict(self, X):
52        X = self.normalize(X) # Normalize the features for prediction
53        return np.dot(X, self.weights) + self.bias
54
55    def get_weights(self):
56        return self.weights
57
58    def get_bias(self):
59        return self.bias
60
61    def get_costs(self):
62        return self.costs
63
64
65
66
```

## Example1

```
In [ ]: 1 X = np.array([[32.50234527],
2 [53.42680403],
3 [61.53035803],
4 [47.47563963],
5 [59.81320787],
6 [55.14218841],
7 [52.21179669],
8 [39.29956669],
9 [48.10504169],
10 [52.55001444],
11 [45.41973014],
12 [54.35163488],
13 [44.1640495 ],
14 [58.16847072],
15 [56.72720806],
16 [48.95588857],
17 [44.68719623],
18 [60.29732685],
19 [45.61864377],
20 [38.81681754]])
21
22 y = np.array([[31.70700585],
23 [68.77759598],
24 [62.5623823],
25 [71.54663223],
26 [87.23092513],
27 [78.21151827],
28 [79.64197305],
29 [59.17148932],
30 [75.3312423 ],
31 [71.30087989],
32 [55.16567715],
33 [82.47884676],
34 [62.00892325],
35 [75.39287043],
36 [81.43619216],
37 [60.72360244],
38 [82.89250373],
39 [97.37989686],
40 [48.8471532],
41 [56.87721319]])
42
43
44
45 model =RegularizedLinearRegression(iterations = 10, learning_rate = 0.1, stopping_threshold = 1e-6 , Lambda=0.1)
46 model.train(X , y)
```

```
In [ ]: 1 costs=model.get_costs()
2 iteration = [i for i in range(len(costs))]
3
4
5
6 #Visualizing the weights and cost at for all iterations
7 plt.figure(figsize = (8,6))
8 plt.plot(iteration, costs)
9 plt.scatter(iteration, costs, marker='o', color='red')
10 plt.title("Cost vs iteration")
11 plt.ylabel("Cost")
12 plt.xlabel("iteration")
13 plt.show()
```

## Example2

```
In [17]: 1 import numpy as np
2 import pandas as pd
3 data=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\sklearn\en.csv")
4 data.tail(10)
```

Out[17]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
758	0.66	759.5	318.5	220.5	3.5	4	0.4	5	14.92
759	0.66	759.5	318.5	220.5	3.5	5	0.4	5	15.16
760	0.64	784.0	343.0	220.5	3.5	2	0.4	5	17.69
761	0.64	784.0	343.0	220.5	3.5	3	0.4	5	18.19
762	0.64	784.0	343.0	220.5	3.5	4	0.4	5	18.16
763	0.64	784.0	343.0	220.5	3.5	5	0.4	5	17.88
764	0.62	808.5	367.5	220.5	3.5	2	0.4	5	16.54
765	0.62	808.5	367.5	220.5	3.5	3	0.4	5	16.44
766	0.62	808.5	367.5	220.5	3.5	4	0.4	5	16.48
767	0.62	808.5	367.5	220.5	3.5	5	0.4	5	16.64

```
In [18]: 1 X= np.array(data.drop([ "Y1"] , axis=1))
2 y=np.array(data["Y1"])
3
4 y=y.reshape(-1 , 1)
5
6
```

```
In [19]: 1 #train test split
2 from sklearn.model_selection import train_test_split
3
4 X_train , X_test , y_train , y_test= train_test_split(X , y , test_size=0.2 , random_state= 123)
```

```
In [22]: 1 model2 = RegularizedLinearRegression( iterations=1000, learning_rate=0.005, stopping_threshold=1e-6, Lambda=0.1)
2 model2.train(X_train , y_train)
```

```
In [23]: 1 predict_test=model2.predict(X_test)
2
3 #accuracy
4 from sklearn.metrics import r2_score
5
6 print(round(r2_score(predict_test , y_test)*100,2))
7
8
9
```

87.67

```
In [ ]: 1 costs=model2.get_costs()
2 iteration = [i for i in range(len(costs))]
3
4
5
6 #Visualizing the weights and cost at for all iterations
7 plt.figure(figsize = (8,6))
8 plt.plot(iteration, costs)
9 plt.title("Cost vs iteration")
10 plt.xlabel("Cost")
11 plt.ylabel("iteration")
12 plt.show()
```

## RegularizedNormal\_equation

```
In [3]: 1 import numpy as np
2
3 def ridge_regression_normal_equation(X, y, Lambda):
4
5     m, n = X.shape
6     I = np.eye(n)      # Identity matrix of shape (n, n)
7     I[0, 0] = 0        # Exclude the bias term from regularization if applicable
8
9     # Normal equation with regularization
10    theta = np.linalg.inv(X.T @ X + Lambda * I) @ X.T @ y
11    return theta
12
13
14
15
```

```
In [4]: 1 import numpy as np
2 import pandas as pd
3 data=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\energy.csv")
4 data.tail(10)
```

Out[4]:

	X0	X1	X2	X3	X4	X5	X6	X7	X8	Y
758	1	0.66	759.5	318.5	220.5	3.5	4	0.4	5	14.92
759	1	0.66	759.5	318.5	220.5	3.5	5	0.4	5	15.16
760	1	0.64	784.0	343.0	220.5	3.5	2	0.4	5	17.69
761	1	0.64	784.0	343.0	220.5	3.5	3	0.4	5	18.19
762	1	0.64	784.0	343.0	220.5	3.5	4	0.4	5	18.16
763	1	0.64	784.0	343.0	220.5	3.5	5	0.4	5	17.88
764	1	0.62	808.5	367.5	220.5	3.5	2	0.4	5	16.54
765	1	0.62	808.5	367.5	220.5	3.5	3	0.4	5	16.44
766	1	0.62	808.5	367.5	220.5	3.5	4	0.4	5	16.48
767	1	0.62	808.5	367.5	220.5	3.5	5	0.4	5	16.64

```
In [5]: 1 X= np.array(data.drop(["Y" , "X0"] , axis= 1))
2 y=np.array(data["Y"])
3
4 y=y.reshape((768,1))
5
```

```
In [6]: 1 from sklearn.preprocessing import StandardScaler
2
3 sc= StandardScaler()
4 X_= sc.fit_transform(X)
5
6 #SLIT
7 from sklearn.model_selection import train_test_split
8 X_train , X_test , y_train , y_test = train_test_split(X , y , random_state=123 , test_size=0.2)
9
10 theta= ridge_regression_normal_equation(X_train , y_train , 0.01)
11 theta
```

Out[6]: array([[-2.18372008e+01],
 [-2.21614098e-03],
 [ 2.77825243e-02],
 [-1.49994288e-02],
 [ 5.60455306e+00],
 [-1.13046991e-01],
 [ 2.02708145e+01],
 [ 2.19828003e-01]])

```
In [7]: 1 y_hat= X_test@theta
2
3 from sklearn.metrics import r2_score
4
5 print(round(r2_score(y_hat , y_test )*100 , 2))
6
7
8
```

90.23

## RegularizedPolynomialRegression

```
In [8]: M
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class RegularizedPolynomialRegression:
5
6     def __init__(self, degree, learning_rate, iterations, stopping_threshold, Lambda=0.01):
7         self.degree = degree
8         self.learning_rate = learning_rate
9         self.iterations = iterations
10        self.stopping_threshold = stopping_threshold
11        self.Lambda = Lambda
12        self.costs = []
13        self.weights = None
14        self.bias = None
15
16    def normalize(self, X, method=None):
17        """
18            zscore:
19                It is a special case of the normal distribution where
20                    the mean (average) is 0 and the standard deviation is 1.
21
22            maxabs:
23                Rescale each feature between -1 and 1.
24
25            mimmax:
26                The Min-Max Scaling (Normalization)
27                technique works by transforming the original
28                data into a new range, typically between 0 and 1.
29        """
30        if method is None or method == "zscore":
31            return (X - X.mean(axis=0)) / X.std(axis=0)
32        elif method == "maxabs":
33            return X / np.abs(X.max(axis=0))
34        elif method == "mimmax":
35            return (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
36
37    # Transform method
38    def transform(self, X):
39        X_norm = self.normalize(X)
40        num_samples, num_features = X_norm.shape
41        X_transform = np.empty((num_samples, 0))
42        for i in range(1, self.degree + 1):
43            X_transform = np.concatenate((X_transform, np.sin(2 * i * X_norm)), axis=1)
44        return X_transform
45
46    # model training
47    def fit(self, X, y):
48        X_transform = self.transform(X)
49        m, n = X_transform.shape
50
51        # weight initialization
52        self.weights = np.random.rand(n, 1)
53        self.bias = np.zeros([1, 1])
54
55        previous_cost = None
56
57        for i in range(self.iterations):
58            # Making predictions
59            prediction = (X_transform @ self.weights) + self.bias
60
61            # Calculating the current cost with L2 regularization
62            current_cost = np.sum((1 / (2 * m)) * np.square(prediction - y)) + \
63                            (self.Lambda / (2 * m)) * np.sum(self.weights ** 2)
64
65            # If the change in cost is less than or equal to
66            # stopping_threshold we stop the gradient descent
67            if previous_cost is not None and abs(previous_cost - current_cost) <= self.stopping_threshold:
68                break
69
70            previous_cost = current_cost
71            self.costs.append(current_cost)
72
73            # Calculating the gradients with L2 regularization
74            weight_derivative = (X_transform.T @ (prediction - y)) + self.Lambda * self.weights) / m
75            bias_derivative = np.sum(prediction - y) / m
76
77            # Updating weights and bias
78            self.weights -= self.learning_rate * weight_derivative
79            self.bias -= self.learning_rate * bias_derivative
80
81
82    # Predict method
83    def predict(self, X):
84        X_transform = self.transform(X)
85        return (X_transform @ self.weights) + self.bias
86
87    # Methods to get weights, bias, and costs
88    def get_weights(self):
89        return self.weights
90
91    def get_bias(self):
92        return self.bias
93
94    def get_costs(self):
95        return self.costs
```

```
In [9]: M
1 import pandas as pd
2 dataenergy=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\sklearn\en.csv")
3 dataenergy.head()
```

Out[9]:

	X1	X2	X3	X4	X5	X6	X7	X8	y1
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84

```
In [11]: 1 X=np.array(dataenergy.drop(["y1"] , axis = 1))
2 y=np.array(dataenergy["y1"])
3
4 y=y.reshape(-1,1)

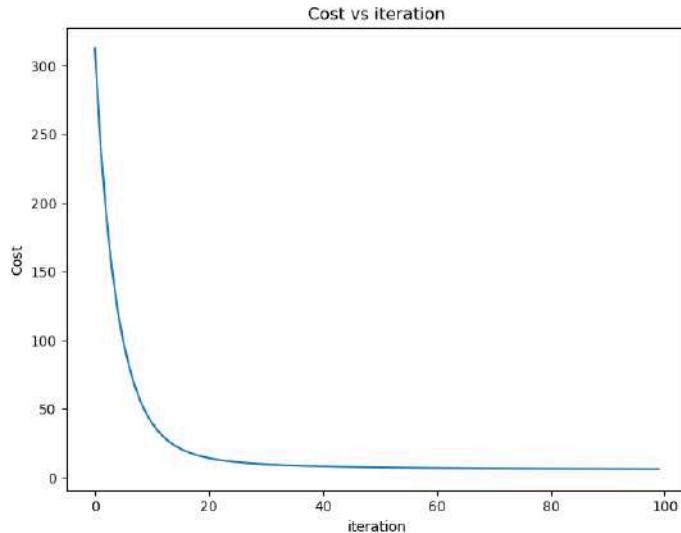
In [12]: 1 #train test split
2 from sklearn.model_selection import train_test_split
3
4 X_train , X_test , y_train , y_test= train_test_split(X , y , test_size=0.2 , random_state= 123)

In [13]: 1 model3=RegularizedPolynomialRegression(2, 0.1,100, 1e-12, Lambda=0.02)
2 model3.fit(X_train , y_train)

In [14]: 1 predict_test= model3.predict(X_test)
2
3 #accuracy
4 from sklearn.metrics import r2_score
5
6 print(round(r2_score(predict_test , y_test)*100,2))
7

83.17

In [15]: 1 costs=model3.get_costs()
2 iteration = [i for i in range(len(costs))]
3
4
5
6 #Visualizing the weights and cost at for all iterations
7 plt.figure(figsize = (8,6))
8 plt.plot(iteration, costs)
9 plt.title("Cost vs iteration")
10 plt.ylabel("Cost")
11 plt.xlabel("iteration")
12 plt.show()
```



## SVM

In [16]:

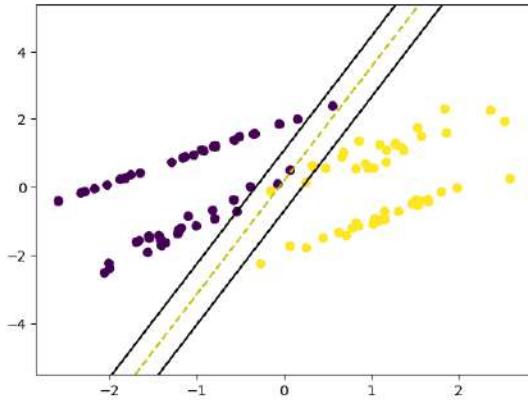
```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class RegularizedSVM:
5     def __init__(self, learning_rate=0.001, n_iters=1000, degree=1, Lambda=0.01):
6         self.lr = learning_rate
7         self.degree = degree
8         self.n_iters = n_iters
9         self.Lambda = Lambda
10        self.w = None
11        self.b = None
12        self.costs = []
13
14    def normalize(self, X):
15        return (X - X.mean(axis=0)) / X.std(axis=0)
16
17    # Transform method
18    def transform(self, X):
19        X_norm = self.normalize(X)
20        num_samples, num_features = X_norm.shape
21        X_transform = np.empty((num_samples, 0))
22        for i in range(1, self.degree + 1):
23            X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
24        return X_transform
25
26    def fit(self, X, y):
27        Xtansnorm = self.transform(X)
28        n_samples, n_features = Xtansnorm.shape
29
30        y_ = np.where(y <= 0, -1, 1)
31
32        # Initialize weights
33        self.w = np.zeros(n_features)
34        self.b = 0
35
36        for _ in range(self.n_iters):
37            cost = 0
38            for idx, x_i in enumerate(Xtansnorm):
39                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
40                if condition:
41                    self.w -= self.lr * (2 * self.Lambda * self.w / n_samples)
42                else:
43                    self.w -= self.lr * (2 * self.Lambda * self.w / n_samples - np.dot(x_i, y_[idx]))
44                    self.b -= self.lr * y_[idx]
45                    cost += 1 - y_[idx] * (np.dot(x_i, self.w) - self.b) # Hinge loss
46
47            self.costs.append(cost)
48
49    def predict(self, X):
50        X_transformed = self.transform(X)
51        approx = np.dot(X_transformed, self.w) - self.b
52        return np.sign(approx)
53
54    def get_weights(self):
55        return self.w
56
57    def get_bias(self):
58        return self.b
59
60    def get_costs(self):
61        return self.costs
62
63    def visualize_svm(self, X, y):
64        def get_hyperplane_value(x, w, b, offset):
65            return (-w[0] * x + b + offset) / w[1]
66
67        fig = plt.figure()
68        ax = fig.add_subplot(1, 1, 1)
69        plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)
70
71        x0_1 = np.amin(X[:, 0])
72        x0_2 = np.amax(X[:, 0])
73
74        x1_1 = get_hyperplane_value(x0_1, self.w, self.b, 0)
75        x1_2 = get_hyperplane_value(x0_2, self.w, self.b, 0)
76
77        x1_1_m = get_hyperplane_value(x0_1, self.w, self.b, -1)
78        x1_2_m = get_hyperplane_value(x0_2, self.w, self.b, -1)
79
80        x1_1_p = get_hyperplane_value(x0_1, self.w, self.b, 1)
81        x1_2_p = get_hyperplane_value(x0_2, self.w, self.b, 1)
82
83        ax.plot([x0_1, x0_2], [x1_1, x1_2], "y--")
84        ax.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "k")
85        ax.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "k")
86
87        x1_min = np.amin(X[:, 1])
88        x1_max = np.amax(X[:, 1])
89        ax.set_xlim([x1_min - 3, x1_max + 3])
90
91        plt.show()
92
93
94
95

```

In [17]:

```
1 # Example usage
2 # Assuming X and y are already defined as your dataset
3 #
4 # Example usage:
5 # Load a dataset (e.g., from sklearn)
6 from sklearn.datasets import make_classification
7
8 # Generate a synthetic dataset
9 X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)
10
11 # Initialize and train the SVM model
12 model = RegularizedSVM(learning_rate=0.001, n_iters=1000, degree=2, Lambda=0.01)
13 model.fit(X, y)
14
15 # Predict on the same data (or new data)
16 predictions = model.predict(X)
17
18
19 # Get model parameters
20 weights = model.get_weights()
21 bias = model.get_bias()
22 costs = model.get_costs()
23
24 model.visualize_svm(X, y)
```



## RegularizedLogisticRegression

In [18]:

```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3
 4 class RegularizedLogisticRegression:
 5     def __init__(self, X, y, iteration, learning_rate, stopping_threshold, degree, Lambda=0.01):
 6         self.X = X
 7         self.y = y
 8         self.iteration = iteration
 9         self.learning_rate = learning_rate
10         self.stopping_threshold = stopping_threshold
11         self.degree = degree
12         self.Lambda = Lambda
13         self.classes = np.unique(y)
14         self.classifiers = {}
15
16     def normalize(self, X):
17         mean = X.mean(axis=0)
18         std = X.std(axis=0)
19         std[std == 0] = 1 # To avoid division by zero, set zero std to one
20         return (X - mean) / std
21
22     def transform(self, X):
23         X_norm = self.normalize(X)
24         num_samples, num_features = X_norm.shape
25         X_transform = np.empty((num_samples, 0))
26         for i in range(1, self.degree + 1):
27             X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
28         return X_transform
29
30     def sigmoid(self, Z):
31         return 1. / (1. + np.exp(-Z))
32
33     def train_binary_classifier(self, X, y):
34         Xtansnorm = self.transform(X)
35         m, n = Xtansnorm.shape
36
37         # Weight initialization
38         current_weight = np.random.rand(n, 1)
39         current_bias = np.zeros((1, 1))
40
41         costs = []
42         previous_cost = None
43
44         for i in range(self.iteration):
45             # Making predictions
46             Z = Xtansnorm @ current_weight + current_bias
47             y_hat = self.sigmoid(Z)
48
49             # Clipping the predictions to avoid Log(0) error
50             y_hat = np.clip(y_hat, 1e-15, 1 - 1e-15)
51
52             # Calculating the current cost with L2 regularization
53             loss = - np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
54             loss += (self.Lambda / (2 * m)) * np.sum(np.square(current_weight))
55
56             # If the change in cost is less than or equal to stopping threshold we stop the gradient descent
57             if previous_cost is not None and abs(previous_cost - loss) <= self.stopping_threshold:
58                 break
59
60             previous_cost = loss
61             costs.append(loss)
62
63             # Calculating the gradients with L2 regularization
64             weight_derivative = (Xtansnorm.T @ (y_hat - y)) / m + (self.Lambda / m) * current_weight
65             bias_derivative = np.sum(y_hat - y) / m
66
67             # Updating weights and bias
68             current_weight -= self.learning_rate * weight_derivative
69             current_bias -= self.learning_rate * bias_derivative
70
71         return current_weight, current_bias, costs
72
73     def train(self):
74         for cls in self.classes:
75             y_binary = (self.y == cls).astype(int).reshape(-1, 1)
76             weight, bias, costs = self.train_binary_classifier(self.X, y_binary)
77             self.classifiers[cls] = (weight, bias, costs)
78
79     def predict_proba(self, X):
80         Xtansnorm = self.transform(X)
81         probabilities = np.zeros((Xtansnorm.shape[0], len(self.classes)))
82
83         for i, cls in enumerate(self.classes):
84             weight, bias, _ = self.classifiers[cls]
85             Z = Xtansnorm @ weight + bias
86             probabilities[:, i] = self.sigmoid(Z).ravel()
87
88         return probabilities
89
90     def predict(self, X):
91         probabilities = self.predict_proba(X)
92         predictions = np.argmax(probabilities, axis=1)
93         return self.classes[predictions]
94
95     def get_weights(self):
96         return {cls: self.classifiers[cls][0] for cls in self.classes}
97
98     def get_biases(self):
99         return {cls: self.classifiers[cls][1] for cls in self.classes}
```

```
In [19]: M
1 from sklearn.datasets import load_iris
2
3 from sklearn.model_selection import train_test_split
4
5 iris = load_iris()
6 X = iris.data
7 y = iris.target
8
9
10
11 X_train , X_test , y_train, y_test = train_test_split(X , y , random_state= 12 , test_size=0.2)
12
13
14 model = RegularizedLogisticRegression(X_train, y_train, iteration=1000, learning_rate=0.03, stopping_threshold=1e-6, degree=2, Lambda=0.01)
15
16 # Train the model
17 model.train()
18
19 # Make predictions
20 predict_test = model.predict(X_test)
21 predict_train= model.predict(X_train)
22
23 #score
24 from sklearn.metrics import accuracy_score
25
26 print(f" train accuracy is: {accuracy_score(predict_train , y_train)} and test accuracy is: {accuracy_score(predict_test , y_test)}")

```

◀ ▶

train accuracy is: 0.95 and test accuracy is: 0.9666666666666667

## sklearn

```
In [20]: M
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.metrics import accuracy_score, confusion_matrix
4 from sklearn.datasets import load_digits
5 from sklearn.preprocessing import MaxAbsScaler
6 from sklearn.model_selection import train_test_split
7 from sklearn.linear_model import LogisticRegression
8
9 # Load dataset
10 mnist = load_digits()
11 X = mnist.data
12 y = mnist.target
13
14 # Split data
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=78)
16
17 # Normalize data
18 scaler = MaxAbsScaler()
19 X_train = scaler.fit_transform(X_train)
20 X_test = scaler.transform(X_test)
21
22 # Create and train model
23 model2 = LogisticRegression(max_iter=1000, solver='lbfgs', multi_class='auto', penalty="l2" )
24 model2.fit(X_train, y_train)
25
26 # Test accuracy
27 y_predtest = model2.predict(X_test)
28 accuracytest = accuracy_score(y_test, y_predtest)
29 print(f'Accuracy test: {accuracytest:.4f}')
30
31 # Train accuracy
32 y_predtrain = model2.predict(X_train)
33 accuracytrain = accuracy_score(y_train, y_predtrain)
34 print(f'Accuracy train: {accuracytrain:.4f}')
35
36 # Display confusion matrix
37 conf_matrix = confusion_matrix(y_test, y_predtest)
38 print('Confusion Matrix:')
39 print(conf_matrix)
40
```

Accuracy test: 0.9694  
Accuracy train: 0.9854  
Confusion Matrix:  
[[24 0 0 0 0 0 0 0 0]  
 [ 0 31 0 0 0 0 0 1 0]  
 [ 0 0 26 0 0 0 0 0 0]  
 [ 0 0 0 38 0 0 0 1 0]  
 [ 0 1 0 0 46 0 0 0 1]  
 [ 0 0 1 0 0 29 0 0 1]  
 [ 0 0 0 0 0 0 42 0 0]  
 [ 0 0 0 0 0 0 0 44 0]  
 [ 0 2 1 0 0 0 0 0 30]  
 [ 0 0 0 0 0 0 0 0 39]]

```
In [21]: M
1 import pandas as pd
2 from sklearn.datasets import load_wine
3
4 wine=load_wine()
5
6 df=pd.DataFrame(wine.data , columns=wine.feature_names)
7 df[ "target" ]= wine.target
8
9
```

```
In [22]: M
1 #Create x and y
2 X= df.iloc[:, :-1]
3 y=df.iloc[:, -1]
4
```

```
In [23]: 1 #split
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2 , random_state=54698)
4
5
6
7 #X_proj
8 from sklearn.linear_model import LogisticRegression
9 clf= LogisticRegression(n_jobs=-1 , penalty = "l2" ,solver="newton-cg" , C=1 ,max_iter=1500 )
10 clf.fit(X_train , y_train)
11
```

```
Out[23]: LogisticRegression
LogisticRegression(C=1, max_iter=1500, n_jobs=-1, solver='newton-cg')
```

```
In [24]: 1 train_predict= clf.predict(X_train)
2 test_predict=clf.predict(X_test)
3
4 #accuracy
5 from sklearn.metrics import accuracy_score
6 print("the train accuracy is: {0} and the test accuracy is {1}".format((accuracy_score(train_predict , y_train)) , (accuracy_score(test_predict , y_test)) ))
```

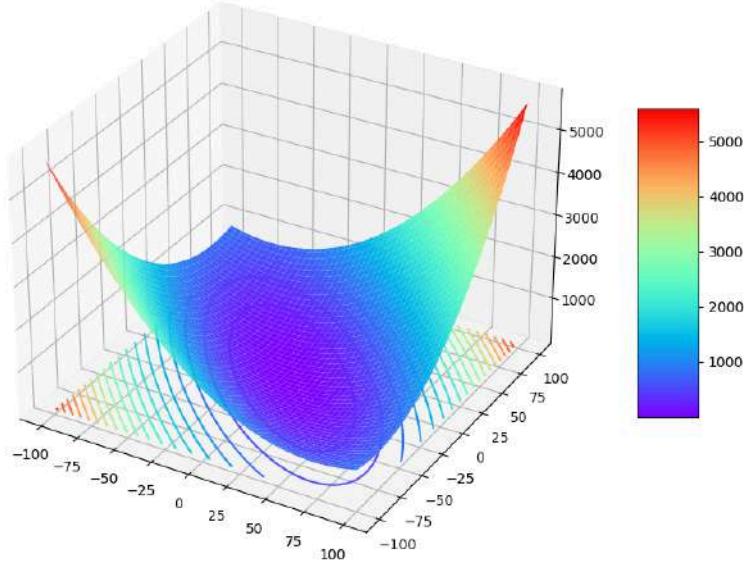
the train accuracy is: 1.0 and the test accuracy is 0.9722222222222222

## Advanced Optimization Methods

```
In [25]: 1 import numpy as np
2 from scipy.optimize import minimize
3
4 # Define the cost function
5 def cost_function(params, X, y, Lambda):
6     weights = np.array(params[:-1])
7     bias = params[-1]
8     predictions = np.dot(X, weights) + bias
9     m = len(y)
10    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (Lambda / (2 * m)) * np.sum(weights ** 2)
11    return cost
12
13 # Define the gradient of the cost function
14 def gradient(params, X, y, Lambda):
15     m = len(y)
16     weights = params[:-1]
17     bias = params[-1]
18     predictions = np.dot(X, weights) + bias
19     weight_derivative = (1 / m) * (np.dot(X.T, (predictions - y)) + Lambda * weights)
20     bias_derivative = (1 / m) * np.sum(predictions - y)
21     return np.append(weight_derivative, bias_derivative)
22
23 # Example data (replace with your actual data)
24 X = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])
25 y = np.array([2, 3, 4, 5])
26 Lambda = 0.1
27
28 # Initial guess for weights and bias
29 initial_params = np.zeros(X.shape[1] + 1)
30
31 # Perform optimization using the BFGS method
32 result = minimize(cost_function, initial_params, args=(X, y, Lambda), method='BFGS', jac=gradient)
33
34 print(result)
```

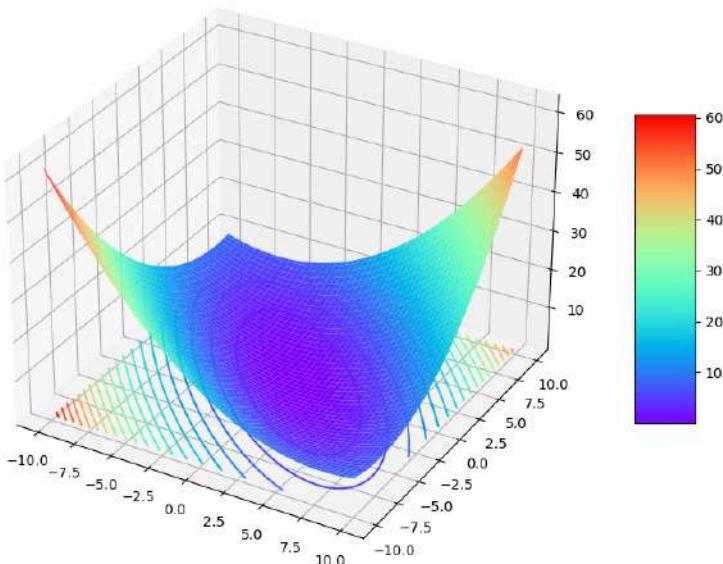
```
message: Optimization terminated successfully.
success: True
status: 0
  fun: 0.0061881188190417505
    x: [ 4.951e-01  4.950e-01  5.297e-01]
   nit: 11
    jac: [ 2.540e-07 -2.358e-07  3.302e-07]
hess_inv: [[ 1.969e+01 -1.927e+01  1.821e+01]
           [-1.927e+01  1.964e+01 -2.055e+01]
           [ 1.821e+01 -2.055e+01  2.741e+01]]
  nfev: 13
  njev: 13
```

```
In [32]: M
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4
5
6 # Assuming X, y, and Lambda are already defined
7 # For demonstration purposes, I'll define X, y, and Lambda here:
8 np.random.seed(0)
9 X = np.random.rand(100, 2)
10 y = np.random.rand(100)
11 Lambda = 0.01
12
13 # Generate a range of values for w0 and w1
14 x0_vals = np.linspace(-100, 100, 100)
15 x1_vals = np.linspace(-100, 100, 100)
16 X0, X1 = np.meshgrid(x0_vals, x1_vals)
17
18 # Calculate cost values
19 Z = np.array([[cost_function([w0, w1, 0], X, y, Lambda) for w0 in x0_vals] for w1 in x1_vals])
20
21 # Plotting
22 fig = plt.figure(figsize=(12, 8))
23 ax = fig.add_subplot(111, projection='3d')
24
25 surf = ax.plot_surface(X0, X1, Z, cmap=plt.cm.rainbow)
26 cset = ax.contour(X0, X1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)
27
28 fig.colorbar(surf, shrink=0.5, aspect=5)
29 plt.show()
30
```



```
In [ ]: M 1
```

```
In [26]: 
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4
5 def cost_function(params, X, y):
6     weights = np.array(params[:-1])
7     bias = params[-1]
8     predictions = np.dot(X, weights) + bias
9     m = len(y)
10    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
11    return cost
12
13 # Assuming X and y are already defined
14 # For demonstration purposes, I'll define X and y here:
15 np.random.seed(0)
16 X = np.random.rand(100, 2)
17 y = np.random.rand(100)
18
19 # Generate a range of values for w0 and w1
20 x0_vals = np.linspace(-10, 10, 100)
21 x1_vals = np.linspace(-10, 10, 100)
22 X0, X1 = np.meshgrid(x0_vals, x1_vals)
23
24 # Calculate cost values
25 Z = np.array([[cost_function([w0, w1, 0], X, y) for w0 in x0_vals] for w1 in x1_vals])
26
27 # Plotting
28 fig = plt.figure(figsize=(12, 8))
29 ax = fig.add_subplot(111, projection='3d')
30
31 surf = ax.plot_surface(X0, X1, Z, cmap=plt.cm.rainbow)
32 cset = ax.contour(X0, X1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)
33
34 fig.colorbar(surf, shrink=0.5, aspect=5)
35 plt.show()
36
```



```
In [11]: 
1 import numpy as np
2 from scipy.optimize import minimize
3
4 def quadratic_objective(x, Q, b):
5     return 0.5 * np.dot(x, np.dot(Q, x)) - np.dot(b, x)
6
7 # Define a positive definite matrix Q and a vector b
8 Q = np.array([[2, 1], [1, 2]])
9 b = np.array([1, 1])
10
11 # Minimize the quadratic objective function using CG method
12 result = minimize(quadratic_objective, x0=[0, 0], args=(Q, b), method='CG')
13 print(result.x)
14
```

[0.33333333 0.33333333]

```
In [14]: 
1 # Minimize the quadratic objective function using L-BFGS method
2 result = minimize(quadratic_objective, x0=[0, 0], args=(Q, b), method='L-BFGS-B')
3 print(result.x)
4
```

[0.33333333 0.33333333]

```
In [16]: 1 import numpy as np
2 from scipy.optimize import minimize
3
4 def quadratic_objective(x, Q, b):
5     return 0.5 * np.dot(x, np.dot(Q, x)) - np.dot(b, x)
6
7 def quadratic_gradient(x, Q, b):
8     return np.dot(Q, x) - b
9
10 def quadratic_hessian(x, Q, b):
11     return Q
12
13 # Define a positive definite matrix Q and a vector b
14 Q = np.array([[2, 1], [1, 2]])
15 b = np.array([1, 1])
16
17 # Initial guess
18 x0 = np.zeros_like(b)
19
20 # Minimize using trust-exact method with explicit gradient and Hessian
21 result = minimize(quadratic_objective, x0, args=(Q, b), method='trust-exact', jac=quadratic_gradient, hess=quadratic_hessian)
22 print(result.x)
23
```

[0.33333333 0.33333333]

```
In [ ]: 1 !pip install pyswarm
```

```
In [17]: 1 from pyswarm import pso
2
3 def rastrigin(x, A=10):
4     n = len(x)
5     return A * n + np.sum(x**2 - A * np.cos(2 * np.pi * x))
6
7 lb = [-5.12, -5.12]
8 ub = [5.12, 5.12]
9
10 x_opt, f_opt = pso(rastrigin, lb, ub)
11 print("Optimal solution:", x_opt)
12 print("Optimal value:", f_opt)
13
```

Stopping search: Swarm best objective change less than 1e-08  
Optimal solution: [1.69689681e-06 6.37172549e-06]  
Optimal value: 8.625761438452173e-09

```
In [ ]: 1
```

```
In [18]: 1 def F(x):
2     return sin(x) + cos(x)
```

```
In [ ]: 1
```

In [35]:

```

1 import numpy as np
2 from scipy.optimize import minimize
3 import matplotlib.pyplot as plt
4
5 # Define the cost function
6 def cost_function(params, X, y, Lambda):
7     m = len(y)
8     weights = np.array(params[:-1])
9     bias = params[-1]
10    predictions = np.dot(X, weights) + bias
11    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (Lambda / (2 * m)) * np.sum(weights ** 2)
12    return cost
13
14 # Define the gradient of the cost function
15 def gradient(params, X, y, Lambda):
16     m = len(y)
17     weights = np.array(params[:-1])
18     bias = params[-1]
19     predictions = np.dot(X, weights) + bias
20     weight_derivative = (1 / m) * (np.dot(X.T, (predictions - y)) + Lambda * weights)
21     bias_derivative = (1 / m) * np.sum(predictions - y)
22     return np.append(weight_derivative, bias_derivative)
23
24 # Example data (replace with your actual data)
25 X = np.array([[1], [2], [3], [4]]) # Single feature for 2D plot
26 y = np.array([2, 3, 4, 5])
27 Lambda = 0.1
28
29 # Initial guess for weights and bias
30 initial_params = np.zeros(X.shape[1] + 1)
31
32 # Record the optimization path
33 history = []
34
35 def callback(params):
36     history.append(params)
37
38 # Perform optimization using the BFGS method
39 result = minimize(cost_function, initial_params, args=(X, y, Lambda), method='BFGS', jac=gradient, callback=callback)
40
41 print(result)
42
43 # Convert history to a numpy array for easy indexing
44 history = np.array(history)
45
46 # Plotting the contour of the cost function and the optimization path
47 w0_vals = np.linspace(-3, 3, 100)
48 b_vals = np.linspace(-3, 3, 100)
49 w0, B = np.meshgrid(w0_vals, b_vals)
50
51 # Calculate cost values
52 Z = np.array([[cost_function([w0, b], X, y, Lambda) for w0 in w0_vals] for b in b_vals])
53
54 plt.figure(figsize=(10, 6))
55 cp = plt.contourf(w0, B, Z, levels=50, cmap='viridis')
56 plt.colorbar(cp)
57 plt.plot(history[:, 0], history[:, 1], 'ro-', label='Optimization Path')
58 plt.scatter(result.x[0], result.x[1], c='red', marker='x', label='Optimized Point')
59 plt.title('Cost Function Contour with Optimization Path')
60 plt.xlabel('Weight')
61 plt.ylabel('Bias')
62 plt.legend()
63 plt.show()
64
65

```

message: Optimization terminated successfully.

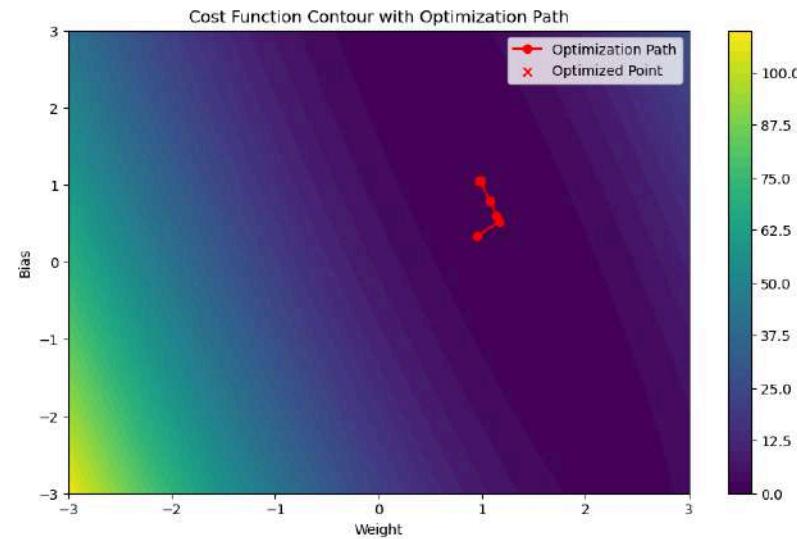
success: True

status: 0

```

        fun: 0.012254901960784314
        x: [ 9.804e-01  1.049e+00]
        nit: 7
        jac: [-5.899e-11  1.096e-10]
      hess_inv: [[ 7.842e-01 -1.960e+00]
                  [-1.960e+00  5.901e+00]]
      nfev: 8
      njev: 8

```

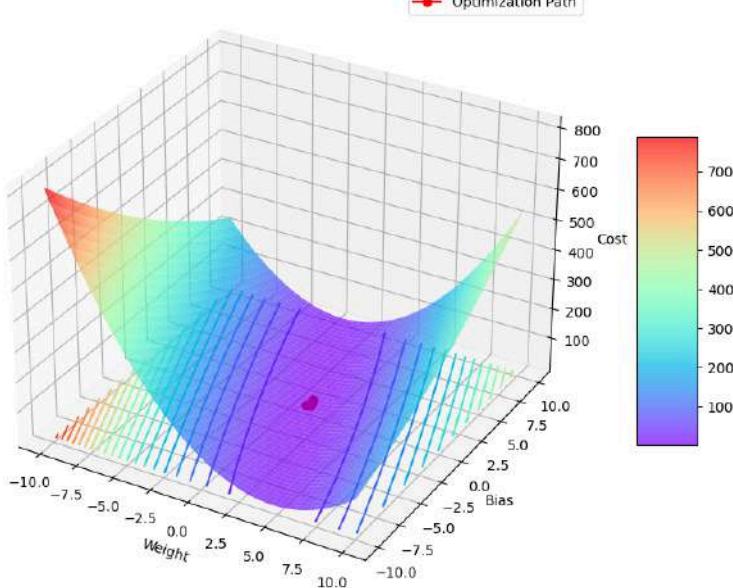


In [36]:

```
 1 import numpy as np
 2 from scipy.optimize import minimize
 3 import matplotlib.pyplot as plt
 4 from mpl_toolkits.mplot3d import Axes3D
 5
 6 # Define the cost function
 7 def cost_function(params, X, y, Lambda):
 8     m = len(y)
 9     weights = np.array(params[:-1])
10     bias = params[-1]
11     predictions = np.dot(X, weights) + bias
12     cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (Lambda / (2 * m)) * np.sum(weights ** 2)
13     return cost
14
15 # Define the gradient of the cost function
16 def gradient(params, X, y, Lambda):
17     m = len(y)
18     weights = np.array(params[:-1])
19     bias = params[-1]
20     predictions = np.dot(X, weights) + bias
21     weight_derivative = (1 / m) * (np.dot(X.T, (predictions - y)) + Lambda * weights)
22     bias_derivative = (1 / m) * np.sum(predictions - y)
23     return np.append(weight_derivative, bias_derivative)
24
25 # Example data (replace with your actual data)
26 X = np.array([[1], [2], [3], [4]]) # Single feature for 2D plot
27 y = np.array([2, 3, 4, 5])
28 Lambda = 0.1
29
30 # Initial guess for weights and bias
31 initial_params = np.zeros(X.shape[1] + 1)
32
33 # Record the optimization path
34 history = []
35
36 def callback(params):
37     history.append(params)
38
39 # Perform optimization using the BFGS method
40 result = minimize(cost_function, initial_params, args=(X, y, Lambda), method='BFGS', jac=gradient, callback=callback)
41
42 print(result)
43
44 # Convert history to a numpy array for easy indexing
45 history = np.array(history)
46
47 # Plotting the cost function in 3D
48 w0_vals = np.linspace(-10, 10, 100)
49 b_vals = np.linspace(-10, 10, 100)
50 W0, B = np.meshgrid(w0_vals, b_vals)
51
52 # Calculate cost values
53 Z = np.array([[cost_function([w0, b], X, y, Lambda) for w0 in w0_vals] for b in b_vals])
54
55 fig = plt.figure(figsize=(12, 8))
56 ax = fig.add_subplot(111, projection='3d')
57
58 # Plot surface
59 surf = ax.plot_surface(W0, B, Z, cmap=plt.cm.rainbow, alpha=0.7)
60
61 # Plot contours
62 cset = ax.contour(W0, B, Z, 20, zdir='z', offset=np.min(Z), cmap=plt.cm.rainbow)
63
64 # Plot the optimization path
65 path_costs = np.array([cost_function(params, X, y, Lambda) for params in history])
66 ax.plot(history[:, 0], history[:, 1], path_costs, 'ro-', label='Optimization Path')
67
68 fig.colorbar(surf, shrink=0.5, aspect=5)
69 ax.set_title('Cost Function Surface and Optimization Path')
70 ax.set_xlabel('Weight')
71 ax.set_ylabel('Bias')
72 ax.set_zlabel('Cost')
73 plt.legend()
74 plt.show()
```

```
message: Optimization terminated successfully.
success: True
status: 0
    fun: 0.012254901960784314
    x: [ 9.804e-01  1.049e+00]
    nit: 7
    jac: [-5.899e-11  1.006e-10]
hess_inv: [[ 7.842e-01 -1.960e+00]
            [-1.960e+00  5.901e+00]]
    nfev: 8
    njev: 8
```

### Cost Function Surface and Optimization Path

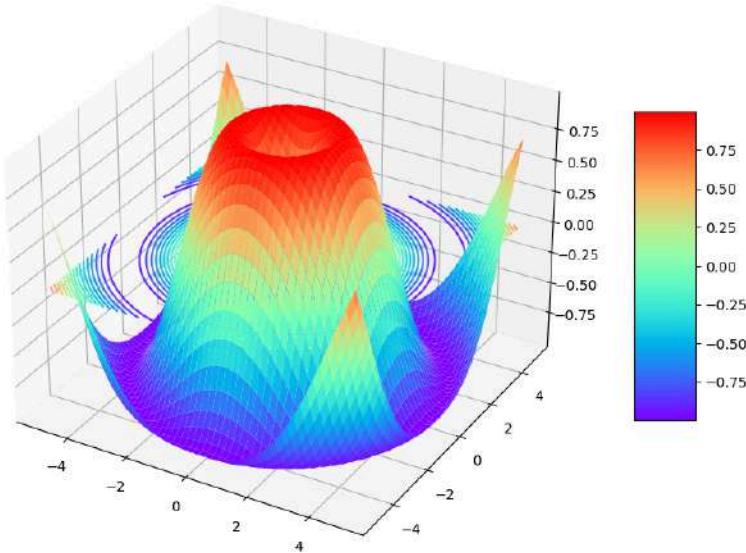


```
In [ ]: M 1
```

```
In [37]: M 1
from scipy.optimize import minimize
import numpy as np
3
4 def function(params):
    return np.sin(np.sqrt(params[0]**2 + params[1]**2))
5
# Use a different optimization method
6 result = minimize(function, x0=[0,0], method='Nelder-Mead')
7 print(result)
8
9
10
11
12
```

```
message: Optimization terminated successfully.
success: True
status: 0
fun: 0.0
x: [ 0.000e+00  0.000e+00]
nit: 5
nfev: 11
final_simplex: (array([[ 0.000e+00,  0.000e+00],
       [ 4.785e-05,  1.758e-05],
       [ 1.172e-05,  8.594e-05]]), array([ 0.000e+00,  5.098e-05,  8.673e-05]))
```

```
In [13]: M
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.mplot3d import Axes3D
3 import numpy as np
4
5 # Assuming T0, T1, and Z are already defined
6 # For demonstration purposes, I'll define T0, T1, and Z here:
7 T0, T1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
8 Z = np.sin(np.sqrt(T0**2 + T1**2))
9
10 fig = plt.figure(figsize=(12, 8))
11 ax = fig.add_subplot(111, projection='3d')
12
13 surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow)
14 cset = ax.contour(T0, T1, Z, 20, zdir='z', offset=0, cmap=plt.cm.rainbow)
15
16 fig.colorbar(surf, shrink=0.5, aspect=5)
17 plt.show()
18
```



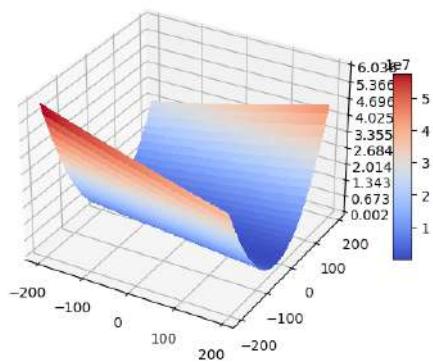
```
In [ ]: M
1 Z = Y.*sin(X) - X.*cos(Y)
```

```
In [ ]: M
1 z= sin(x).*cos(y);
```

```
In [ ]: M
1 %matplotlib inline
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.mplot3d import Axes3D
6
7 from scipy.optimize import minimize # for optimization
```

In [5]:

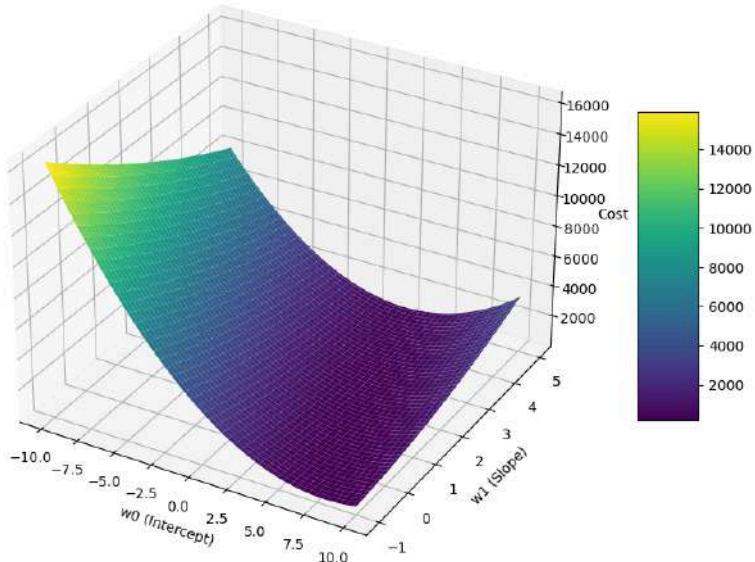
```
 1 import matplotlib.pyplot as plt
 2 from matplotlib import cm
 3 from matplotlib.ticker import LinearLocator
 4 import numpy as np
 5
 6 fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
 7
 8 # Make data.
 9 start, end, step = -200, 200, 5
10 w1, w2 = np.arange(start, end, step), np.arange(start, end, step)
11
12
13
14
15
16
17
18
19 #b=np.arange(-100, 100)
20 w1, w2 = np.meshgrid(w1, w2)
21 cost=(1/3)*((150-(3*w1+10*w2))**2+(500-(5*w1+50*w2))**2+(625-(4*w1+25*w2))**2)
22 #cost=(1/3)*((150-3*w1+10*w2+b)**2+(500-5*w1+50*w2+b)**2+(625-4*w1+25*w2+b)**2)
23 # Plot the surface.
24 surf = ax.plot_surface(w1, w2, cost, cmap=cm.coolwarm,
25                         linewidth=0, antialiased=False)
26
27
28 ax.xaxis.set_major_locator(LinearLocator(10))
29
30 # Add a color bar which maps values to colors.
31 fig.colorbar(surf, shrink=0.5, aspect=10)
32
33 plt.show()
```



In [3]:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4
5 # Define the cost function for Linear regression
6 def cost_function(params, X, y):
7     m = len(y)
8     weights = np.array(params[:-1]) # weights
9     bias = params[-1] # bias
10    predictions = np.dot(X, weights) + bias
11    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
12    return cost
13
14 # Generate synthetic data for demonstration
15 np.random.seed(0)
16 X = 2 * np.random.rand(100, 1)
17 y = 4 + 3 * X + np.random.randn(100, 1)
18
19 # Since we are including bias separately, no need to add intercept term in X
20 X_b = X # without intercept column
21
22 # Generate a range of values for w0 (intercept) and w1 (slope)
23 w0_vals = np.linspace(-10, 10, 100)
24 w1_vals = np.linspace(-1, 5, 100)
25 W0, W1 = np.meshgrid(w0_vals, w1_vals)
26
27 # Calculate cost values
28 Z = np.array([[cost_function([w1, w0], X_b, y) for w0 in w0_vals] for w1 in w1_vals])
29
30 # Plotting
31 fig = plt.figure(figsize=(12, 8))
32 ax = fig.add_subplot(111, projection='3d')
33
34 # Plot the surface
35 surf = ax.plot_surface(W0, W1, Z, cmap=plt.cm.viridis, edgecolor='none')
36
37 # Add Labels and title
38 ax.set_xlabel('w0 (Intercept)')
39 ax.set_ylabel('w1 (Slope)')
40 ax.set_zlabel('Cost')
41 ax.set_title('3D Surface Plot of Linear Regression Cost Function')
42
43 # Add a color bar which maps values to colors
44 fig.colorbar(surf, shrink=0.5, aspect=5)
45
46 plt.show()
47
48
```

3D Surface Plot of Linear Regression Cost Function



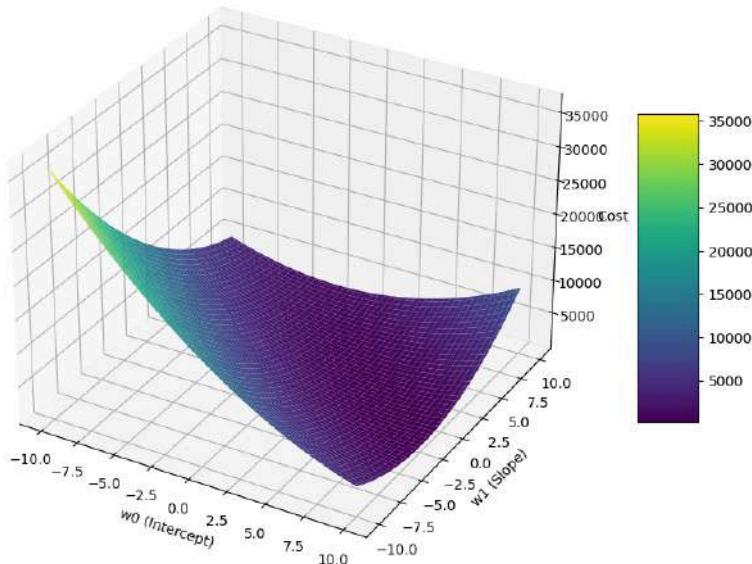
In [1]:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4
5 # Define the cost function for Linear regression
6 def cost_function(params, X, y):
7     m = len(y)
8     weights = np.array(params) # weights (intercept and slope)
9     predictions = np.dot(X, weights)
10    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
11    return cost
12
13 # Generate synthetic data for demonstration
14 np.random.seed(0)
15 X = 2 * np.random.rand(100, 1)
16 y = 4 + 3 * X + np.random.randn(100, 1)
17
18 # Add a column of ones to X for the bias term
19 X_b = np.c_[np.ones((100, 1)), X] # Bias term added here
20
21 # Generate a range of values for w0 (intercept) and w1 (slope)
22 w0_vals = np.linspace(-10, 10, 100)
23
24
25 w1_vals = np.linspace(-10, 10, 100)
26 W0, W1 = np.meshgrid(w0_vals, w1_vals)
27
28 # Calculate cost values
29 Z = np.array([[cost_function([w0, w1], X_b, y) for w0 in w0_vals] for w1 in w1_vals])
30
31 # Plotting
32 fig = plt.figure(figsize=(12, 8))
33 ax = fig.add_subplot(111, projection='3d')
34
35 # Plot the surface
36 surf = ax.plot_surface(W0, W1, Z, cmap=plt.cm.viridis, edgecolor='none')
37
38 # Add Labels and title
39 ax.set_xlabel('w0 (Intercept)')
40 ax.set_ylabel('w1 (Slope)')
41 ax.set_zlabel('Cost')
42 ax.set_title('3D Surface Plot of Linear Regression Cost Function')
43
44 # Add a color bar which maps values to colors
45 fig.colorbar(surf, shrink=0.5, aspect=5)
46
47 plt.show()
48

```

3D Surface Plot of Linear Regression Cost Function

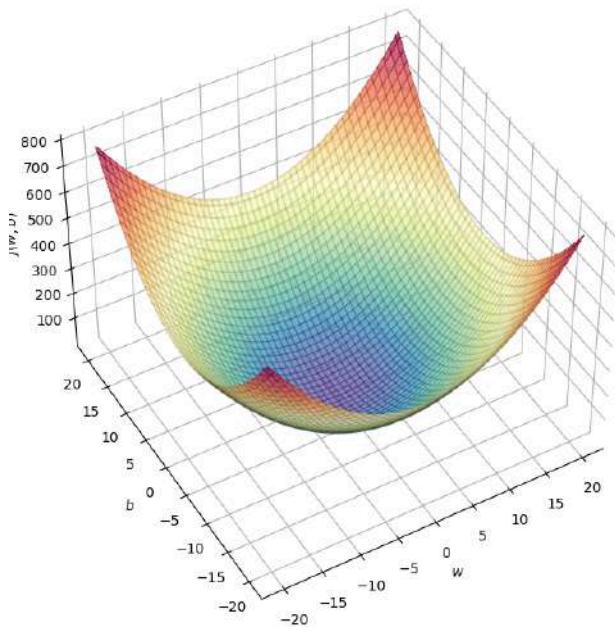


```

In [2]: M 1 # Figure and plot - 3D projection
2 def soup_bowl():
3     """ Create figure and plot with a 3D projection"""
4     fig = plt.figure(figsize=(8,8))
5
6     #Plot configuration
7     ax = fig.add_subplot(111, projection='3d')
8     ax.xaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
9     ax.yaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
10    ax.zaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
11    ax.zaxis.set_rotate_label(False)
12    ax.view_init(45, -120)
13
14    #Useful. Linespaces to give values to the parameters w and b
15    w = np.linspace(-20, 20, 100)
16    b = np.linspace(-20, 20, 100)
17
18    #Get the z value for a bowl-shaped cost function
19    z=np.zeros((len(w), len(b)))
20    j=0
21    for x in w:
22        i=0
23        for y in b:
24            z[i,j] = x**2 + y**2
25            i+=1
26        j+=1
27
28    #Meshgrid used for plotting 3D functions
29    W, B = np.meshgrid(w, b)
30
31    #Create the 3D surface plot of the bowl-shaped cost function
32    ax.plot_surface(W, B, z, cmap = "Spectral_r", alpha=0.7, antialiased=False)
33    ax.plot_wireframe(W, B, z, color='k', alpha=0.1)
34    ax.set_xlabel("$w$")
35    ax.set_ylabel("$b$")
36    ax.set_zlabel("$J(w,b)$", rotation=90)
37    ax.set_title("$J(w,b)$\n [You can rotate this figure]", size=15)
38
39    plt.show()
40
41
42 soup_bowl()

```

$J(w, b)$   
[You can rotate this figure]



```
In [ ]: M 1 minimize(J, x0=[10, -3], method='CG')
```

```
In [ ]: M 1 minimize(J, x0=np.random.randn(2), method='CG')
```

```
In [ ]: M 1 result = minimize(J, x0=np.random.randn(2), method='CG')
```

```
In [ ]: M 1 result.success
```

```
In [ ]: M 1 result.x # solution
```

```
In [ ]: M 1 result.fun # the minimum value of the cost function
```

```
In [ ]: M 1 minimize(J, x0=[0, 0], method='BFGS')
```

```
In [ ]: M 1 def gradients(theta):
2         return np.array([2 * (theta[0] - 5), 2 * (theta[1] - 5)])
```

```
In [ ]: M 1 gradients([5, 5])
```

```

In [ ]: 1 minimize(J, x0=np.random.randn(2), method='CG', jac=gradients)

In [ ]: 1 y_train.shape

In [ ]: 1

In [ ]: 1
import numpy as np
from scipy.optimize import minimize
from sklearn.metrics import r2_score
4
5 class RegularizedlinearRegression:
6     def __init__(self, iterations=100000, learning_rate=0.0001, stopping_threshold=1e-6, Lambda=0.01):
7         self.iterations = iterations
8         self.learning_rate = learning_rate
9         self.stopping_threshold = stopping_threshold
10        self.Lambda = Lambda
11        self.costs = []
12        self.weights = None
13        self.bias = None
14
15    def normalize(self, X):
16        """ Normalizes the feature matrix X """
17        mean = np.mean(X, axis=0)
18        std = np.std(X, axis=0)
19        std[std == 0] = 1 # To avoid division by zero, set zero std to one
20        return (X - mean) / std
21
22    def cost_function(self, params, X, y, Lambda):
23        """ Computes the cost with L2 regularization """
24        m = X.shape[0]
25        weights = params[:-1].reshape(-1, 1)
26        bias = params[-1]
27        predictions = np.dot(X, weights) + bias
28        cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2) + (Lambda / (2 * m)) * np.sum(weights ** 2)
29        return cost
30
31    def fit(self, X, y):
32        X = self.normalize(X) # Normalize the features
33        m, n = X.shape
34        initial_params = np.random.rand(n + 1) # Initialize weights and bias
35
36        # Objective function to minimize
37        def objective_function(params):
38            return self.cost_function(params, X, y, self.Lambda)
39
40        # Optimize using scipy's minimize function
41        result = minimize(objective_function, initial_params, method='BFGS', options={'maxiter': self.iterations})
42
43        # Extract weights and bias from the result
44        self.weights = result.x[:-1].reshape(-1, 1)
45        self.bias = result.x[-1]
46        self.costs = result.fun
47
48    def predict(self, X):
49        X = self.normalize(X) # Normalize the features for prediction
50        return np.dot(X, self.weights) + self.bias
51
52    def get_weights(self):
53        return self.weights
54
55    def get_bias(self):
56        return self.bias
57
58    def get_costs(self):
59        return self.costs
60
61 # Example usage
62 if __name__ == "__main__":
63     # Create some example data
64     np.random.seed(42)
65     X_train = np.random.rand(100, 3)
66     y_train = np.dot(X_train, np.array([3, 5, 2])) + np.random.randn(100, 1)
67
68     model = RegularizedlinearRegression(iterations=10000, learning_rate=0.01, stopping_threshold=1e-6, Lambda=0.1)
69     model.fit(X_train, y_train)
70     predictions = model.predict(X_train)
71
72     print("Weights:", model.get_weights())
73     print("Bias:", model.get_bias())
74     print("Costs:", model.get_costs())
75
76     # Evaluate the model
77     #r2 = r2_score(y_train, predictions)
78     #print("R2 Score:", r2)
79 #
80
81

```

```
In [ ]: 1
```

```
In [ ]: 1
```

In [ ]:

```
1 import numpy as np
2 from scipy.optimize import minimize
3 from sklearn.metrics import r2_score
4
5 class RegularizedLinearRegression:
6     def __init__(self, iterations=100000, learning_rate=0.0001, Lambda=0.01):
7         self.iterations = iterations
8         self.learning_rate = learning_rate
9
10        self.Lambda = Lambda
11        self.costs = []
12        self.weights = None
13        self.bias = None
14
15    def normalize(self, X):
16        """ Normalizes the feature matrix X """
17        mean = np.mean(X, axis=0)
18        std = np.std(X, axis=0)
19        return (X - mean) / std
20
21    def cost_function(self, X, y, Lambda):
22        prediction = np.dot(X, weights) + bias
23        error = prediction - y
24        cost = np.sum((1 / (2 * len(y))) * (error ** 2)) + (Lambda / (2 * len(y))) * np.sum(weights ** 2)
25        return cost
26
27    def gradient_function(self, X, y, Lambda):
28        #weights = np.random.rand(n, 1)
29        #bias = np.zeros([1, 1])
30        prediction = np.dot(X, weights) + bias
31        error = prediction - y
32        weight_gradients = (X.T @ error + Lambda * weights) / len(y)
33        bias_gradient = np.sum(error) / len(y)
34        return np.concatenate((weight_gradients.flatten(), [bias_gradient]))
35
36    def train(self, X, y):
37        X = self.normalize(X) # Normalize the features
38        m, n = X.shape
39
40        self.weights = np.random.rand(n, 1)
41        self.bias = np.zeros([1, 1])
42        initial_params = np.concatenate((self.weights, self.bias), axis=0)
43
44        # Minimize the cost function using scipy.optimize.minimize
45        result = minimize(fun=self.cost_function, x0=initial_params, args=(X, y, self.Lambda),
46                           jac=self.gradient_function, method='BFGS', options={'maxiter': self.iterations})
47
48        # Retrieve optimized parameters
49        optimized_params = result.x
50        self.weights = optimized_params[:-1].reshape(-1, 1)
51        self.bias = optimized_params[-1]
52
53        # Compute costs
54        self.costs = result.fun
55
56    def predict(self, X):
57        X = self.normalize(X) # Normalize the features for prediction
58        return np.dot(X, self.weights) + self.bias
59
60    def get_weights(self):
61        return self.weights
62
63    def get_bias(self):
64        return self.bias
65
66    def get_costs(self):
67        return self.costs
68
69    # Example usage:
70    # Assuming X_train, y_train are your training data
71    X_train = np.array([[1, 2], [2, 3], [3, 4]])
72    y_train = np.array([3, 4, 5])
73
74    model = RegularizedLinearRegression(iterations=100000, learning_rate=0.0001, Lambda=0.01)
75    model.train(X_train, y_train)
76
77    # Get weights, bias, and costs
78    print("Weights:")
79    print(model.get_weights())
80    print("Bias:", model.get_bias())
81    print("Costs:", model.get_costs())
82
83    # Evaluate the model
84    predictions = model.predict(X_train)
85    r2 = r2_score(y_train, predictions)
86    print("R2 Score:", r2)
```

```
In [ ]: M
 1 import numpy as np
 2 from sklearn.metrics import r2_score
 3
 4 class RegularizedLinearRegression:
 5     def __init__(self, iterations=100000, learning_rate=0.0001, stopping_threshold=1e-6, Lambda=0.01):
 6         self.iterations = iterations
 7         self.learning_rate = learning_rate
 8         self.stopping_threshold = stopping_threshold
 9         self.Lambda = Lambda
10         self.costs = []
11         self.weights = None
12         self.bias = None
13
14     def normalize(self, X):
15         """ Normalizes the feature matrix X """
16         mean = np.mean(X, axis=0)
17         std = np.std(X, axis=0)
18         return (X - mean) / std
19
20     def train(self, X, y):
21         X = self.normalize(X) # Normalize the features
22         m, n = X.shape
23         self.weights = np.random.rand(n, 1)
24         self.bias = np.zeros([1, 1])
25         previous_cost = None
26
27         for i in range(self.iterations):
28             # Making predictions
29             prediction = np.dot(X, self.weights) + self.bias
30
31             # Calculating the current cost with L2 regularization
32             current_cost = (1 / (2 * m)) * (np.sum((prediction - y) ** 2) + \
33                                         (self.Lambda / (2 * m)) * np.sum(self.weights ** 2))
34
35             # If the change in cost is less than or equal to stopping_threshold, stop gradient descent
36             if previous_cost and abs(previous_cost - current_cost) <= self.stopping_threshold:
37                 break
38
39             previous_cost = current_cost
40             self.costs.append(current_cost)
41
42             # Calculating the gradients with L2 regularization
43             weight_derivative = (1 / m) * (np.dot(X.T, (prediction - y)) + self.Lambda * self.weights)
44             bias_derivative = (1 / m) * np.sum(prediction - y)
45
46             # Updating weights and bias
47             self.weights -= self.learning_rate * weight_derivative
48             self.bias -= self.learning_rate * bias_derivative
49
50
51     def predict(self, X):
52         X = self.normalize(X) # Normalize the features for prediction
53         return np.dot(X, self.weights) + self.bias
54
55     def get_weights(self):
56         return self.weights
57
58     def get_bias(self):
59         return self.bias
60
61     def get_costs(self):
62         return self.costs
63
64
65
66
```

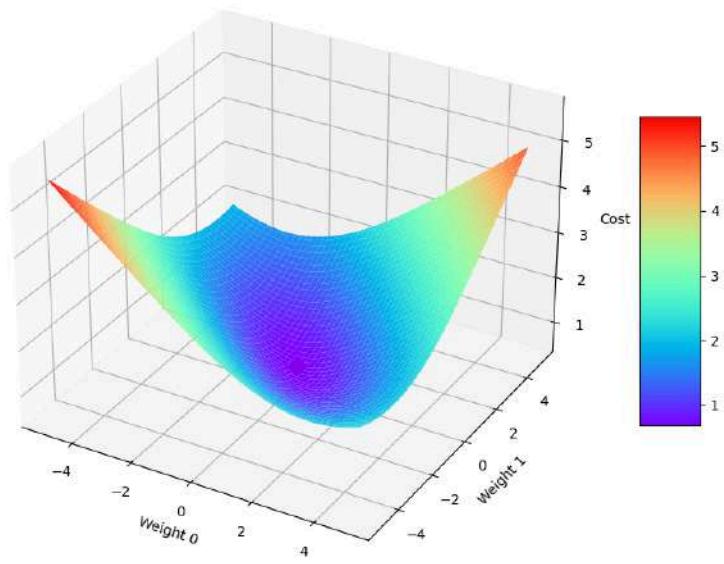
In [153]:

```

1 import numpy as np
2 from scipy.optimize import minimize
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.mplot3d import Axes3D
5
6 # Define the sigmoid function
7 def sigmoid(z):
8     return 1 / (1 + np.exp(-z))
9
10 # Define the cost function with L2 regularization
11 def cost_function(params, X, y, Lambda):
12     m = len(y)
13     weights = params[:-1]
14     bias = params[-1]
15     y_hat = sigmoid(np.dot(X, weights) + bias)
16     loss = -np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
17     loss += (Lambda / (2 * m)) * np.sum(np.square(weights))
18     return loss
19
20 # Define the gradient of the cost function
21 def gradient(params, X, y, Lambda):
22     m = len(y)
23     weights = params[:-1]
24     bias = params[-1]
25     y_hat = sigmoid(np.dot(X, weights) + bias)
26     weight_derivative = (np.dot(X.T, (y_hat - y)) / m) + (Lambda / m) * weights
27     bias_derivative = np.sum(y_hat - y) / m
28     return np.append(weight_derivative, bias_derivative)
29
30 # Generate synthetic data for demonstration
31 np.random.seed(0)
32 X = 2 * np.random.rand(100, 2) # 100 samples, 2 features
33 y = (np.random.rand(100) > 0.5).astype(int) # Binary target
34
35 # Regularization parameter
36 Lambda = 1.0
37
38 # Perform optimization using the BFGS method
39 initial_params = np.zeros(X.shape[1] + 1) # Initialize weights and bias to zero
40 result = minimize(cost_function, initial_params, args=(X, y, Lambda), method='BFGS', jac=gradient)
41 print(result)
42
43 # Optimal parameters
44 optimal_params = result.x
45 Z_min = cost_function(optimal_params, X, y, Lambda)
46
47 # Generate meshgrid for plotting
48 t0 = np.linspace(-5, 5, 100)
49 t1 = np.linspace(-5, 5, 100)
50 T0, T1 = np.meshgrid(t0, t1)
51
52 # Compute the cost function over the meshgrid
53 Z = np.array([[cost_function([w0, w1, optimal_params[-1]], X, y, Lambda) for w0 in t0] for w1 in t1]])
54
55 # Plotting
56 fig = plt.figure(figsize=(12, 8))
57 ax = fig.add_subplot(111, projection='3d')
58
59 # Plot the surface
60 surf = ax.plot_surface(T0, T1, Z, cmap=plt.cm.rainbow)
61
62 # Plot the minimum point if optimization succeeded
63 if result.success:
64     ax.scatter(optimal_params[0], optimal_params[1], Z_min, color='r', label='Minimum Point', s=100)
65 else:
66     print("Could not plot minimum point because optimization did not succeed.")
67
68 # Color bar
69 fig.colorbar(surf, shrink=0.5, aspect=5)
70
71 # Axes Labels
72 ax.set_xlabel('Weight 0')
73 ax.set_ylabel('Weight 1')
74 ax.set_zlabel('Cost')
75
76 # Show plot
77 plt.show()
78

```

message: Optimization terminated successfully.  
success: True  
status: 0  
status: 0  
fun: 0.6859657632436597  
x: [-5.434e-02 3.914e-01 -3.252e-01]  
nit: 13  
jac: [ 6.071e-06 4.850e-06 2.644e-06]  
hess\_inv: [[ 1.038e+01 8.687e-01 -1.183e+01]  
[ 8.687e-01 9.589e+00 -1.010e+01]  
[-1.183e+01 -1.010e+01 2.614e+01]]  
nfev: 14  
njev: 14



## EXERCISE 1 - Water Potability

ph: pH of 1. water (0 to 14). Hardness: Capacity of water to precipitate soap in mg/L. Solids: Total dissolved solids in ppm. Chloramines: Amount of Chloramines in ppm. Sulfate: Amount of Sulfates dissolved in mg/L. Conductivity: Electrical conductivity of water in  $\mu\text{S}/\text{cm}$ . Organic\_carbon: Amount of organic carbon in ppm. Trihalomethanes: Amount of Trihalomethanes in  $\mu\text{g}/\text{L}$ . Turbidity: Measure of light emitting property of water in NTU. Potability: Indicates if water is safe for human consumption. Potable : 1 and Not potable : 0

```
In [1]: #import Libraries
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sb
import pandas as pd
```

```
In [3]: df=pd.read_csv(r"D:\AI_Machinelearning\datasets\ENERGY\water_potability.csv")
df
```

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

```
In [3]: df.describe()
```

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

```
In [4]: df.describe().T.style.background_gradient(subset=['mean','std','50%','count'], cmap='YlOrRd')
```

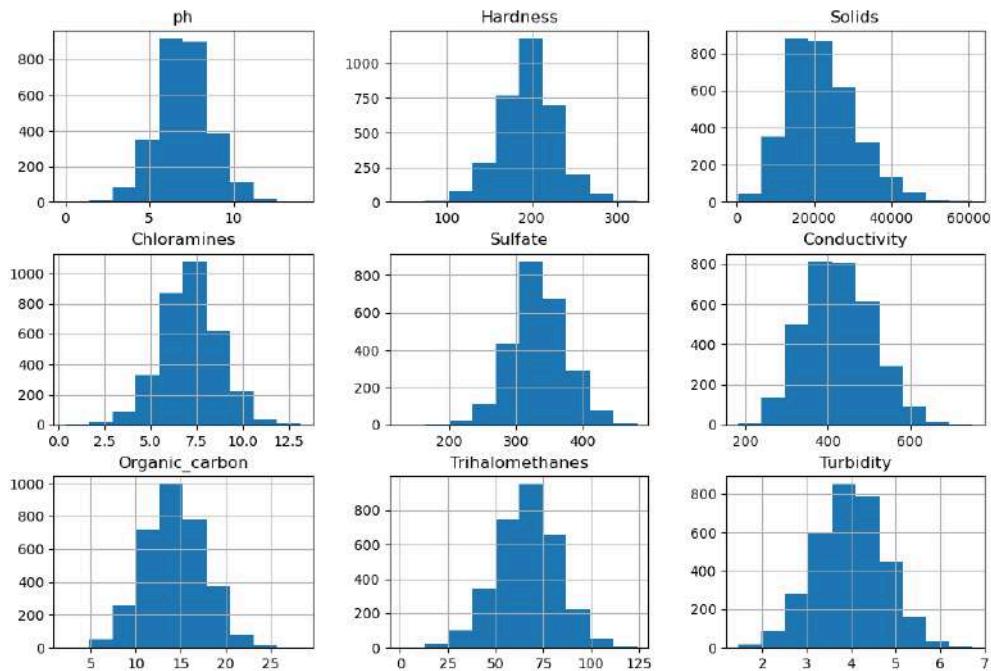
Out[4]:

	count	mean	std	min	25%	50%	75%	max
ph	2785.000000	7.080795	1.594320	0.000000	6.093092	7.036752	8.062066	14.000000
Hardness	3276.000000	196.369496	32.879761	47.432000	176.850538	196.967627	216.667456	323.124000
Solids	3276.000000	22014.092526	8768.570828	320.942611	15666.690297	20927.833607	27332.762127	61227.196008
Chloramines	3276.000000	7.122277	1.583085	0.352000	6.127421	7.130299	8.114887	13.127000
Sulfate	2495.000000	333.775777	41.416840	129.000000	307.699498	333.073546	359.950170	481.030642
Conductivity	3276.000000	426.205111	80.824064	181.483754	365.734414	421.884968	481.792304	753.342620
Organic_carbon	3276.000000	14.284970	3.308162	2.200000	12.065801	14.218338	16.557652	28.300000
Trihalomethanes	3114.000000	66.396293	16.175008	0.738000	55.844536	66.622485	77.337473	124.000000
Turbidity	3276.000000	3.966786	0.780382	1.450000	3.439711	3.955028	4.500320	6.739000
Potability	3276.000000	0.390110	0.487849	0.000000	0.000000	0.000000	1.000000	1.000000

```
In [5]: df.drop('Potability', axis=1).hist(figsize=(12,8))
```

Out[5]:

```
array([[<Axes: title={'center': 'ph'}>,
       <Axes: title={'center': 'Hardness'}>,
       <Axes: title={'center': 'Solids'}>],
      [<Axes: title={'center': 'Chloramines'}>,
       <Axes: title={'center': 'Sulfate'}>,
       <Axes: title={'center': 'Conductivity'}>],
      [<Axes: title={'center': 'Organic_carbon'}>,
       <Axes: title={'center': 'Trihalomethanes'}>,
       <Axes: title={'center': 'Turbidity'}>],
      [<Axes: title={'center': 'Potability'}>]], dtype=object)
```



In [6]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ph          2785 non-null    float64
 1   Hardness     3276 non-null    float64
 2   Solids      3276 non-null    float64
 3   Chloramines  3276 non-null    float64
 4   Sulfate      2495 non-null    float64
 5   Conductivity 3276 non-null    float64
 6   Organic_carbon 3276 non-null  float64
 7   Trihalomethanes 3114 non-null float64
 8   Turbidity    3276 non-null    float64
 9   Potability   3276 non-null    int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

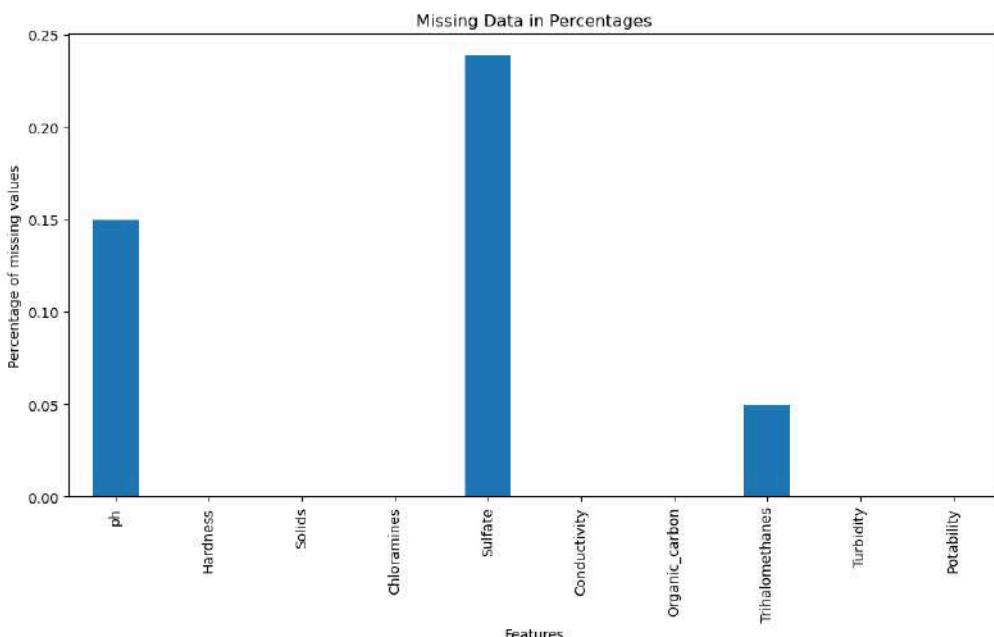
In [7]: `df.isnull().sum().sum()`

Out[7]: 1434

In [8]: `df.isnull().mean()*100`

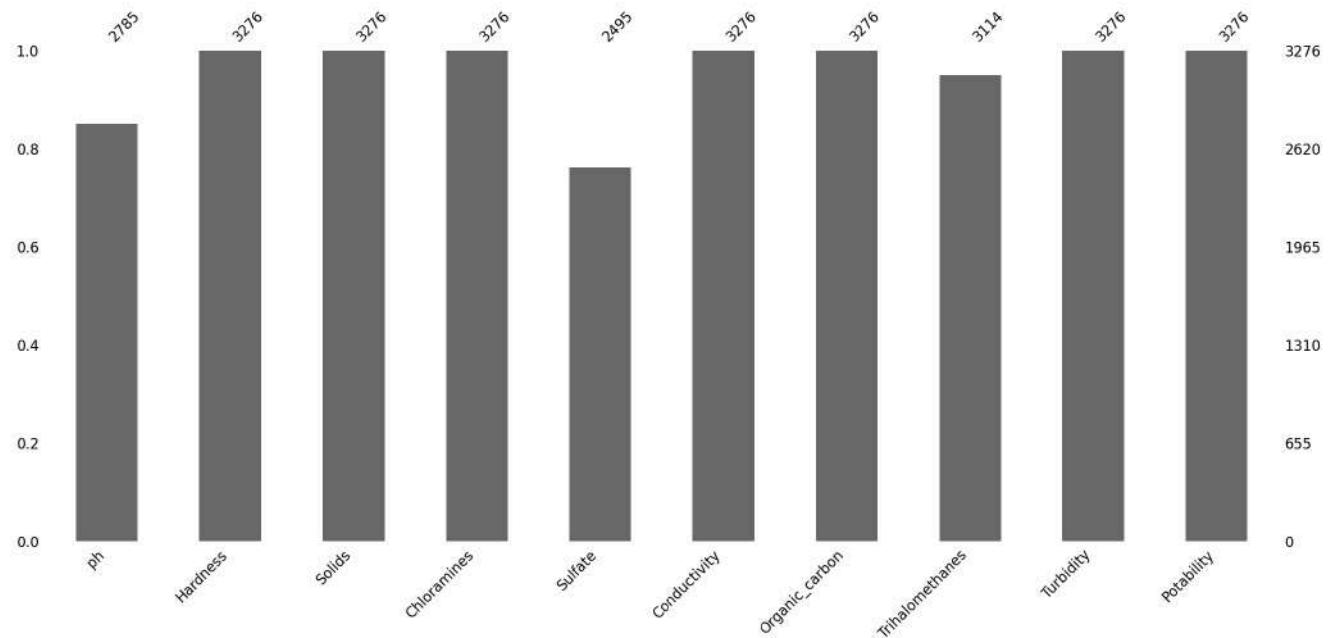
```
Out[8]: ph           14.987790
Hardness       0.000000
Solids         0.000000
Chloramines    0.000000
Sulfate        23.840049
Conductivity   0.000000
Organic_carbon 0.000000
Trihalomethanes 4.945855
Turbidity      0.000000
Potability     0.000000
dtype: float64
```

In [9]: `df.isnull().mean().plot.bar(figsize=(12,6))
plt.ylabel('Percentage of missing values')
plt.xlabel('Features')
plt.title('Missing Data in Percentages');`

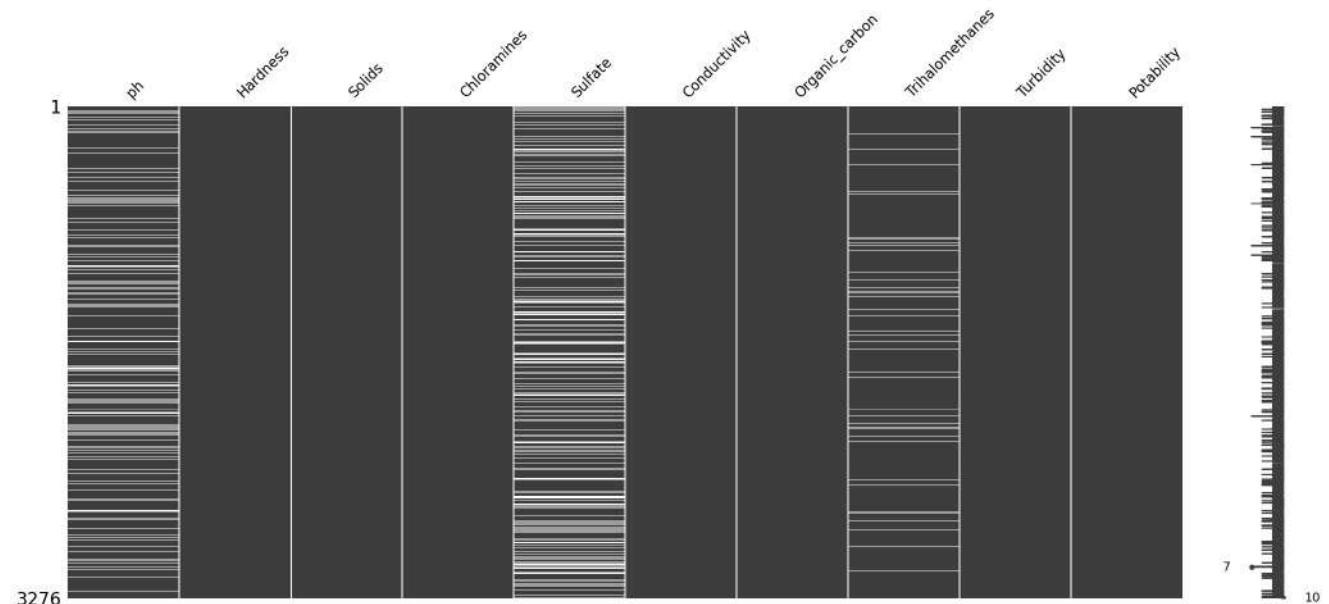


```
In [1]: pip install missingno
```

```
In [10]: #Another method to show the missing values by using missingno Library  
import missingno as msno  
msno.bar(df);
```



```
In [11]: msno.matrix(df);
```



```
In [12]: df.dropna(thresh=2) #df.dropna(how="all")
```

```
df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 3276 entries, 0 to 3275  
Data columns (total 10 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --  
 0   ph          2785 non-null    float64  
 1   Hardness    3276 non-null    float64  
 2   Solids     3276 non-null    float64  
 3   Chloramines 3276 non-null    float64  
 4   Sulfate     2495 non-null    float64  
 5   Conductivity 3276 non-null    float64  
 6   Organic_carbon 3276 non-null    float64  
 7   Trihalomethanes 3114 non-null    float64  
 8   Turbidity    3276 non-null    float64  
 9   Potability    3276 non-null    int64  
dtypes: float64(9), int64(1)  
memory usage: 256.1 KB
```

```
In [3]: df.fillna(df.mean(), axis=0, inplace=True)  
display(df)
```

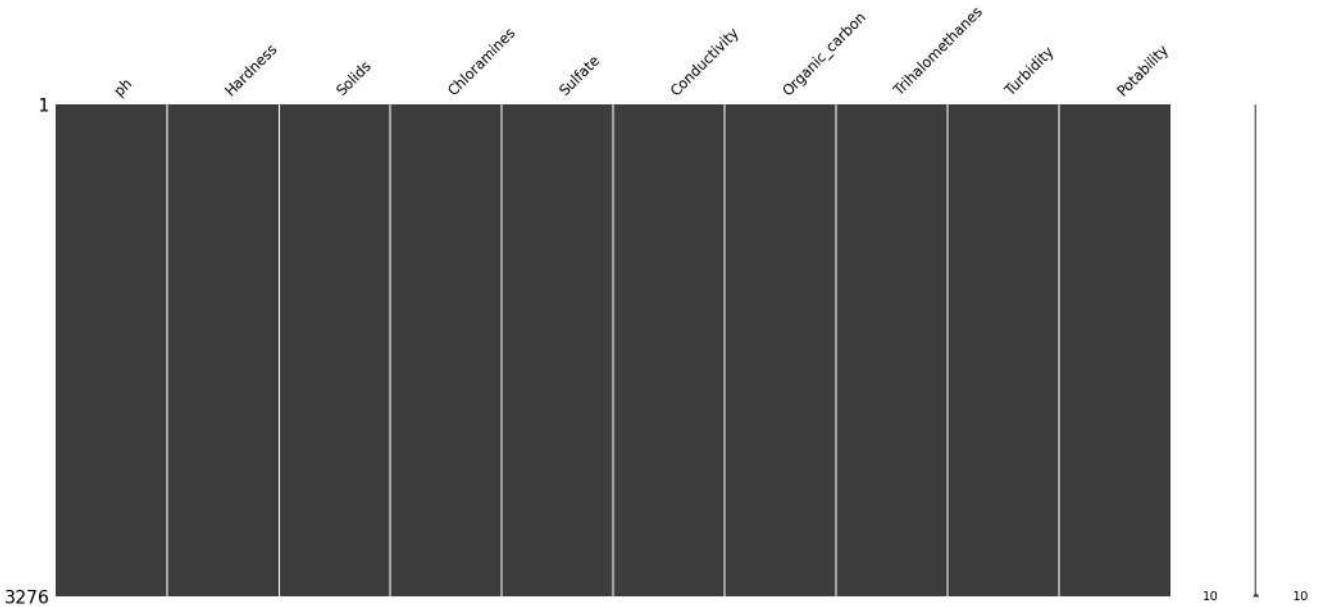
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

In [14]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   ph          3276 non-null   float64
 1   Hardness    3276 non-null   float64
 2   Solids     3276 non-null   float64
 3   Chloramines 3276 non-null   float64
 4   Sulfate     3276 non-null   float64
 5   Conductivity 3276 non-null   float64
 6   Organic_carbon 3276 non-null   float64
 7   Trihalomethanes 3276 non-null   float64
 8   Turbidity   3276 non-null   float64
 9   Potability   3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

In [15]: `msno.matrix(df);`

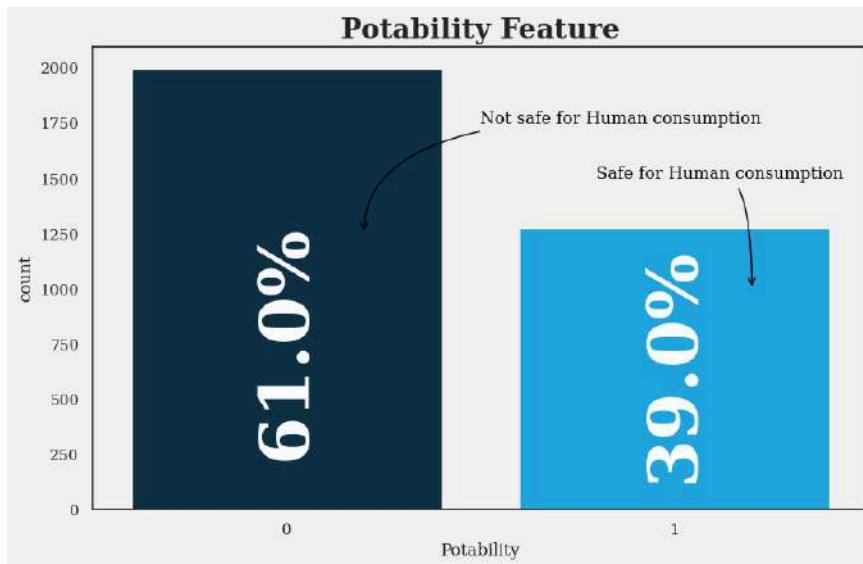


In [16]: `#Import plotting Libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
colors = ['#003344', '#00b2ff']
sns.set(palette=colors, font='Serif', style='white', rc={'axes.facecolor':'#f1f1f1', 'figure.facecolor':'#f1f1f1'})
sns.palplot(colors)`

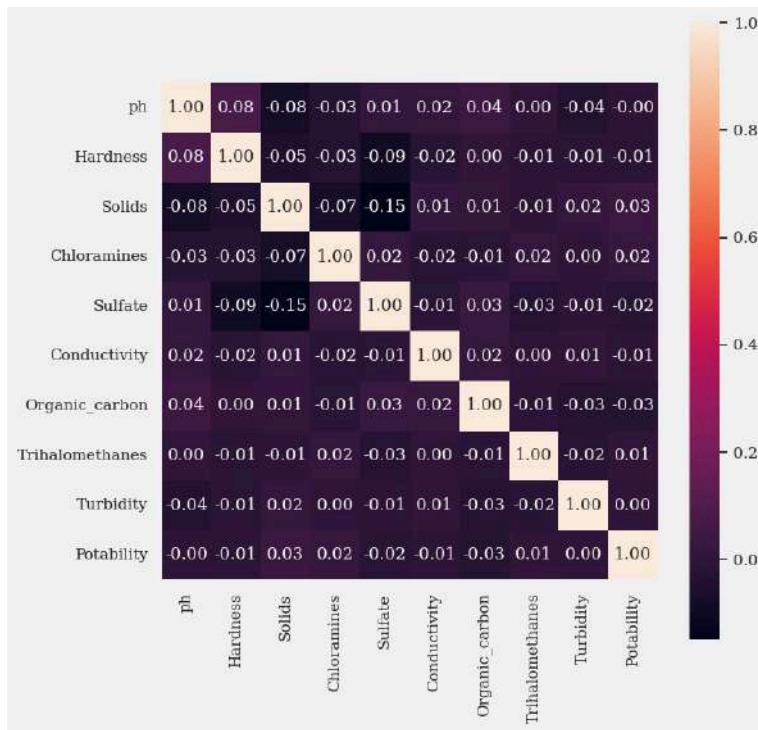


In [17]: `#Lets check the Target features first
fig = plt.figure(figsize=(10,6))
ax=sns.countplot(data=df, x='Potability')
for i in ax.patches:
 ax.text(i.get_x() + i.get_width()/2, i.get_height()/7, s=f'{np.round(i.get_height() / len(df) * 100,0)}%', ha='center', size=50, weight='bold', rotation=90, color='white')
plt.title("Potability Feature", size=20, weight='bold')
plt.annotate(text="Not safe for Human consumption", xytext=(0.5,1750),xy=(0.2,1250), arrowprops =dict(arrowstyle="->", color='black', connectionstyle="angle3,angleA=0,angleB=90"), color='red')
plt.annotate(text="Safe for Human consumption", xytext=(0.8,1500),xy=(1.2,1000), arrowprops =dict(arrowstyle="->", color='black', connectionstyle="angle3,angleA=0,angleB=90"), color='green')`

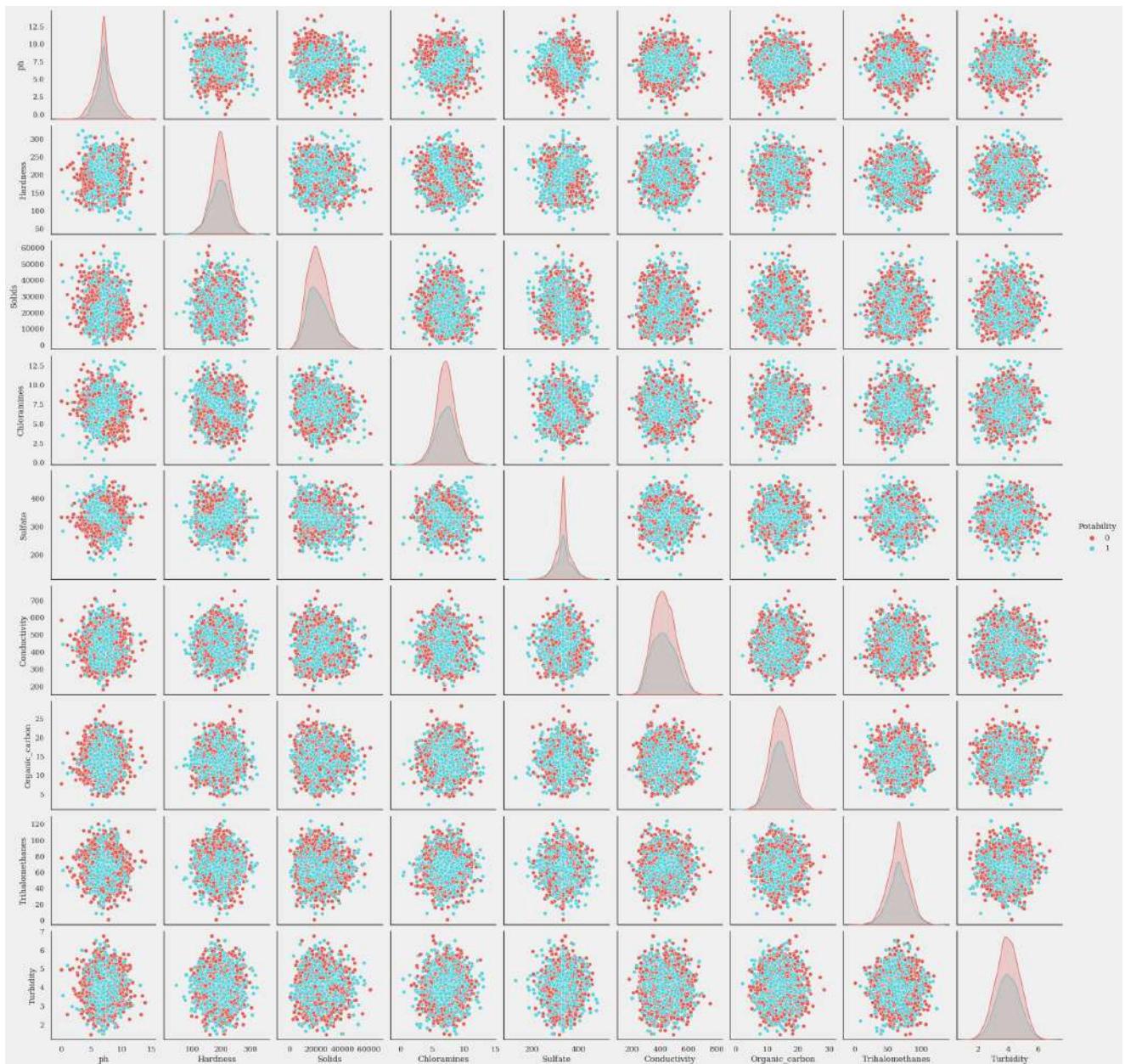
Out[17]: `Text(0.8, 1500, 'Safe for Human consumption')`



```
In [18]: fig=plt.figure(figsize=(8,8))
sns.heatmap(df.corr(), annot=True, fmt='0.2f', square=True)
Out[18]: <Axes: >
```

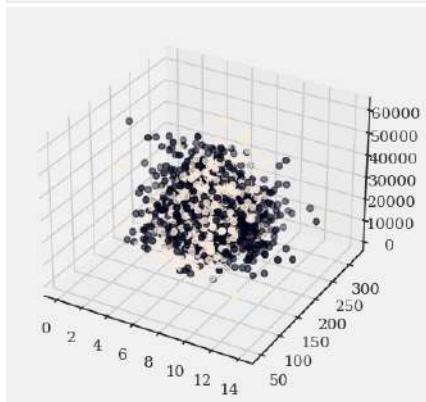


```
In [19]: sb.pairplot(df , hue="Potability" , palette="hls")
plt.show()
```



```
In [20]: X=df.iloc[:, :-1]
y=df.iloc[:, -1]
```

```
In [21]: ax=plt.axes(projection='3d')
ax.scatter3D(X.iloc[:, 0], X.iloc[:, 1], X.iloc[:, 2], c=y)
plt.show()
```



## KNN

```
In [4]: # from sklearn.model_selection import train_test_split
```

```
class KNeighborsClassifier(object):
    def __init__(self, k):
        self.k = k

    def Euclidean(self, point, data):
        return np.sqrt(np.sum((point - data)**2, axis=1))

    def Manhattan(self, point, data):
```

```

    return np.sum(np.abs(point - data) ,axis=1)

def Chebyshev(self, point, data):
    return np.max(np.abs(point - data) ,axis=1)

def fit(self, X, y):
    self.X_train = X
    self.y_train = y

def most_common(self, List):
    return max(set(List), key=list.count)

def predict(self, X_test):
    neighbors = []
    for x in X_test:
        distances = self.Chebyshev(x, self.X_train)
        y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
        neighbors.append(y_sorted[:self.k])
    return list(map(self.most_common, neighbors))

def evaluate(self, X_test, y_test):
    y_pred = np.array(self.predict(X_test))
    accuracy = np.mean(y_pred == y_test)
    return accuracy

```

In [5]: `X=np.array(df.iloc[:, :-1])  
y=np.array(df.iloc[:, -1])`

`X.shape  
y.shape`

Out[5]: (3276,)

In [6]: `from sklearn.model_selection import train_test_split  
X_train ,X_test , y_train , y_test = train_test_split(X , y , test_size=0.2 , random_state=12)`

```

accuracy=[]
for k in range(3, 30,2):
    model = KNeighborsClassifier(k)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    acc = model.evaluate(X_test, y_test)
    accuracy.append(acc)
    print("K = "+str(k)+" ; Accuracy: "+str(acc))

```

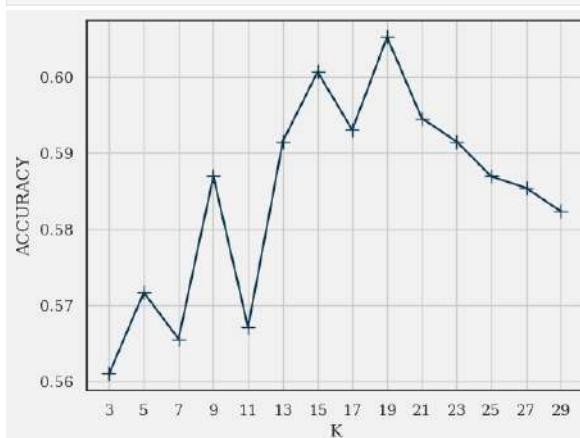
K = 3; Accuracy: 0.5609756097560976  
K = 5; Accuracy: 0.5716463414634146  
K = 7; Accuracy: 0.5655487804878049  
K = 9; Accuracy: 0.586890243902439  
K = 11; Accuracy: 0.5670731707317073  
K = 13; Accuracy: 0.5914634146341463  
K = 15; Accuracy: 0.600609756097561  
K = 17; Accuracy: 0.5929878048780488  
K = 19; Accuracy: 0.6051829268292683  
K = 21; Accuracy: 0.5945121951219512  
K = 23; Accuracy: 0.5914634146341463  
K = 25; Accuracy: 0.586890243902439  
K = 27; Accuracy: 0.5853658536585366  
K = 29; Accuracy: 0.5823170731707317

In [8]: `predict=model.predict(X_test)`

In [9]: `from sklearn.metrics import confusion_matrix  
cnf=confusion_matrix(predict , y_test)  
cnf`

Out[9]: `array([[355, 234],  
 [ 40, 27]], dtype=int64)`

In [25]: `plt.plot([i for i in range(3,30,2)] , accuracy , marker="+" , markersize=10)  
plt.xticks([i for i in range(3,30,2)])  
plt.grid()  
plt.xlabel("K")  
plt.ylabel("ACCURACY")  
plt.show()`



In [26]: `#Normalize  
from sklearn.preprocessing import StandardScaler  
  
sc= StandardScaler()  
x_traintnormal= sc.fit_transform(X_train)  
x_testnormal=sc.transform(X_test)  
  
#train  
for k in range(3, 30,2):  
 model = KNeighborsClassifier(k)  
 model.fit(x_traintnormal, y_train)  
 pred = model.predict(x_testnormal)  
 acc = model.evaluate(x_testnormal, y_test)  
 print("K = "+str(k)+" ; Accuracy: "+str(acc))`

```

K = 3; Accuracy: 0.6082317073170732
K = 5; Accuracy: 0.635670731707317
K = 7; Accuracy: 0.625
K = 9; Accuracy: 0.6417682926829268
K = 11; Accuracy: 0.6371951219512195
K = 13; Accuracy: 0.6478658536585366
K = 15; Accuracy: 0.6318975609756098
K = 17; Accuracy: 0.6318975609756098
K = 19; Accuracy: 0.6448170731707317
K = 21; Accuracy: 0.6432926829268293
K = 23; Accuracy: 0.6585365853658537
K = 25; Accuracy: 0.649390243902439
K = 27; Accuracy: 0.6402439024390244
K = 29; Accuracy: 0.6387195121951219

```

## sklearn knn

```

In [27]: import math
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7 , metric= 'minkowski' , p=2 ,
                           weights="distance" , algorithm="kd_tree" , leaf_size=20 )

In [28]: knn.fit(x_trainnormal, y_train)
y_predict=knn.predict(x_testnormal)

In [29]: from sklearn.metrics import accuracy_score
acc=accuracy_score(y_test , y_predict)
acc

Out[29]: 0.6402439024390244

```

## kfold

```

In [30]: from sklearn.model_selection import cross_val_score

# Train SVM model
model = KNeighborsClassifier (n_neighbors=13 , metric= 'minkowski' , p=math.inf , weights="distance" ,algorithm="ball_tree")

cv_scores=cross_val_score(model ,X ,y , cv=5)

print(cv_scores)

print(round(np.mean(cv_scores) , 2))

[0.55487805 0.5648855 0.58473282 0.56641221 0.5648855 ]
0.57

```

## Naive\_bayes

```

In [31]: from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_trainnormal , y_train)

Out[31]: GaussianNB()
GaussianNB()

In [32]: y_hat=model.predict(x_testnormal)

In [33]: from sklearn.metrics import accuracy_score

acc=accuracy_score(y_test , y_hat)
acc

Out[33]: 0.5899390243902439

```

## SVM

```

In [34]: X=np.array(df.iloc[:, :-1])
y=np.array(df.iloc[:, -1])

#NORMALIZE
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_normal= sc.fit_transform(X)

In [35]: import numpy as np

class SVM:

    def __init__(self, learning_rate=0.001, n_iters=500):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.w = None
        self.b = 0

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)

        # init weights
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2*1/n_samples * self.w - np.dot(x_i, y_[idx]))
                else:
                    self.w -= self.lr * (2 *1/n_samples * self.w - np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]

    def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.sign(approx)

# Testing

```

```

if __name__ == "__main__":
    # Imports
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt

    X, y = x_normal, y
    y = np.where(y == 0, -1, 1)

    X_train, X_test, y_train, y_test = train_test_split(x_normal, y, test_size=0.2, random_state=123)

    clf = SVM()
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    print("SVM classification accuracy", accuracy(y_test, predictions))

def visualize_svm():
    def get_hyperplane_value(x, w, b, offset):
        return (-w[0] * x + b + offset) / w[1]

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    plt.scatter(X[:, 0], X[:, 1], c=y)

    x0_1 = np.amin(X[:, 0])
    x0_2 = np.amax(X[:, 0])

    x1_1 = get_hyperplane_value(x0_1, clf.w, clf.b, 0)
    x1_2 = get_hyperplane_value(x0_2, clf.w, clf.b, 0)

    x1_1_m = get_hyperplane_value(x0_1, clf.w, clf.b, -1)
    x1_2_m = get_hyperplane_value(x0_2, clf.w, clf.b, -1)

    x1_1_p = get_hyperplane_value(x0_1, clf.w, clf.b, 1)
    x1_2_p = get_hyperplane_value(x0_2, clf.w, clf.b, 1)

    ax.plot([x0_1, x0_2], [x1_1, x1_2], "y-")
    ax.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "k")
    ax.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "k")

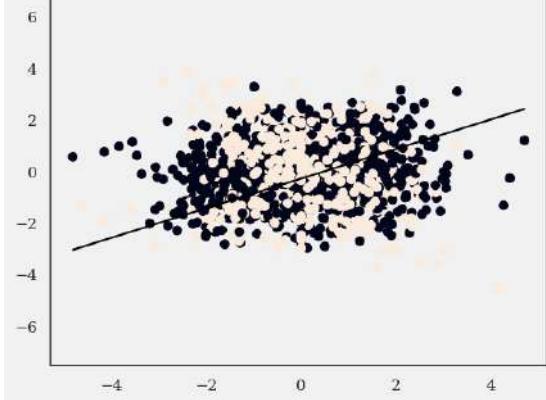
    x1_min = np.amin(X[:, 1])
    x1_max = np.amax(X[:, 1])
    ax.set_xlim([x1_min - 3, x1_max + 3])

    plt.show()

visualize_svm()

```

SVM classification accuracy 0.600609756097561



## kernel trick

```

In [36]: from sklearn.svm import SVC

X=np.array(df.iloc[:, :-1])
y=np.array(df.iloc[:, -1])

# Train and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

```

In [37]: # Train SVM model
model = SVC(kernel="rbf", C=.5, gamma=0.5)
model.fit(X_train, y_train)

y_predict=model.predict(X_test)

#accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_predict, y_test))

0.6448170731707317

```

```

In [38]: # Train SVM model
model = SVC(kernel="poly", C=1, gamma=1, degree=2)
model.fit(X_train, y_train)

y_predict=model.predict(X_test)

#accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_predict, y_test))

0.6768292682926829

```

## kfold

```

In [39]: from sklearn.model_selection import cross_val_score

```

```
# Train SVM model
model = SVC(kernel="rbf" ,C=1, gamma=1 , degree=2)
cv_scores=cross_val_score(model , X ,y , cv=5)
print(cv_scores)
print(round(np.mean(cv_scores) , 2))
[0.6097561 0.61068702 0.61068702 0.60916031 0.60916031]
0.61
```

## Exercise 2 Persian Digits

```
In [40]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import glob
from PIL import Image

In [41]: data_dir= r"D:/AI_Machinelearning/datasets/numbers/"
os.listdir(data_dir)

Out[41]: ['test', 'train']

In [42]: train_dir=f'{data_dir}train'
train_dir

Out[42]: 'D:/AI_Machinelearning/datasets/numbers/train'

In [43]: os.listdir(train_dir)

Out[43]: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

In [44]: test_dir=f'{data_dir}test'

In [45]: trn_fnames=glob.glob(f"{train_dir}/*/*.jpg")

In [46]: plt.imread(trn_fnames[0])
```



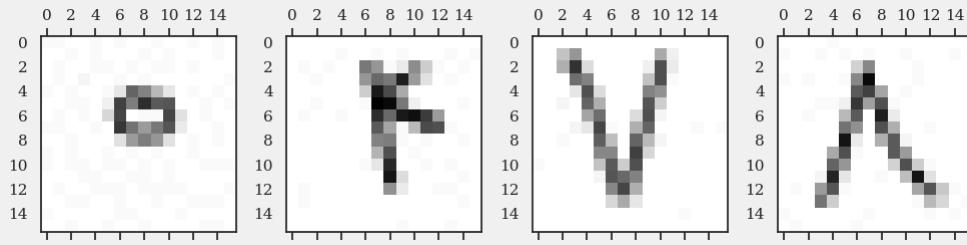


```
[255, 255, 255],  
[253, 253, 253],  
[255, 255, 255],  
[255, 255, 255],  
[253, 253, 253],  
[255, 255, 255],  
[255, 255, 255],  
[253, 253, 253],  
[255, 255, 255],  
[255, 255, 255],  
[255, 255, 255],  
[254, 254, 254],  
[253, 253, 253],  
[255, 255, 255]]], dtype=uint8)
```

```
In [47]: import matplotlib.pyplot as plt
```

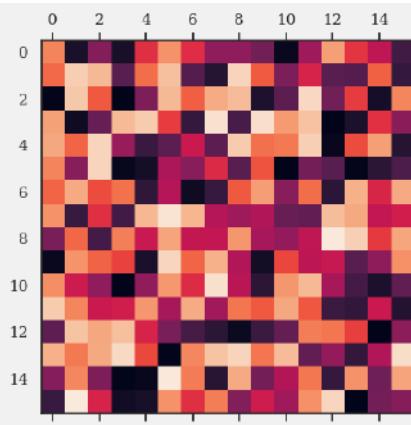
```
#plt.gray()  
fig , (ax1 ,ax2 ,ax3 , ax4)=plt.subplots(nrows=1 , ncols=4)  
fig.set_size_inches(10, 8)  
ax1.matshow(plt.imread(trn_fnames[0])) #plt.imshow  
ax2.matshow(plt.imread(trn_fnames[350]))  
ax3.matshow(plt.imread(trn_fnames[550]))  
ax4.matshow(plt.imread(trn_fnames[600]))
```

```
plt.tight_layout()  
plt.show()
```



```
In [48]: #create noise
```

```
a=np.random.rand(16,16)  
#plt.gray()  
plt.matshow(a)  
plt.show()
```



```
In [49]: def load_images_and_labels(folder, target_size=(16, 16)):
```

```
    images = []  
    labels = []  
    for label in os.listdir(folder):  
        label_path = os.path.join(folder, label)  
        if os.path.isdir(label_path):  
            for filename in os.listdir(label_path):  
                img_path = os.path.join(label_path, filename)  
                if os.path.isfile(img_path):  
                    img = Image.open(img_path).convert('L') # Convert to grayscale  
                    img = img.resize(target_size) # Resize image  
                    img_array = np.array(img).flatten() # Flatten image to 1D array  
                    images.append(img_array)  
                    labels.append(label) # Add the label  
    return np.array(images), np.array(labels)
```

```
def save_images_and_labels_to_csv(X, y, csv_file):  
    # Convert the Numpy arrays to a DataFrame  
    df = pd.DataFrame(X)  
    df['label'] = y # Add the Labels column  
  
    # Save DataFrame to CSV  
    df.to_csv(csv_file, index=False)  
    return df
```

```
In [50]: folder_path = r'D:\AI_Machinlearning\datasets\persianDigits\dataset'  
target_size = (16, 16) # target size
```

```
In [51]: # Load images and labels  
X, y = load_images_and_labels(folder_path, target_size)  
  
# Specify the path where you want to save the CSV file  
csv_file = 'image_data.csv'  
  
# Save to CSV  
df = save_images_and_labels_to_csv(X, y, csv_file)
```

```
df
```

```
Out[51]:
```

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255	label
0	255	255	255	255	255	255	255	255	255	255	...	255	255	255	255	255	255	255	255	255	0
1	255	255	254	255	255	255	255	255	254	255	...	255	255	255	255	255	255	255	255	255	0
2	255	255	254	255	255	255	254	254	255	255	...	255	255	255	255	255	254	255	255	255	0
3	255	255	252	255	254	254	255	255	254	255	...	255	255	255	255	254	255	255	253	253	0
4	255	253	255	255	250	255	253	255	255	254	...	254	254	255	255	253	254	255	255	254	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2050	255	255	255	255	255	255	255	255	255	255	...	255	255	255	255	254	255	255	255	255	9
2051	255	255	254	255	254	255	255	255	255	255	...	255	255	255	254	255	255	255	255	254	9
2052	255	255	255	255	255	255	255	255	255	255	...	254	255	255	254	255	255	254	255	255	9
2053	255	255	255	255	255	255	244	255	255	255	...	255	255	253	249	255	254	255	255	255	9
2054	255	241	255	255	255	255	250	255	253	255	...	255	255	253	255	255	255	249	255	254	9

2055 rows × 257 columns

```
In [52]: from sklearn.model_selection import train_test_split
X= np.array(df.drop("label" , axis=1))
y=np.array(df["label"])
X_train , X_test , y_train , y_test= train_test_split(X , y , test_size=0.2 )
```

```
In [53]: import math
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4 , metric= 'minkowski' , p=2 , weights="distance" ,algorithm="kd_tree" )
```

```
In [54]: knn.fit(X_train, y_train)
y_predict=knn.predict(X_test)
```

```
In [55]: from sklearn.metrics import accuracy_score
acc=accuracy_score(y_test , y_predict)
acc
```

Out[55]: 0.8491484184914841

```
In [56]: #train
for k in range(3, 16,2):
    model = KNeighborsClassifier(n_neighbors=k , metric= 'minkowski' , p=2 , weights="distance" ,algorithm="kd_tree" )
    model.fit(X_train, y_train)
    acc = model.score(X_test , y_test)
    print("K = "+str(k)+" ; Accuracy: "+str(acc))
```

K = 3; Accuracy: 0.8369829683698297  
K = 5; Accuracy: 0.8540145985401459  
K = 7; Accuracy: 0.8199513381995134  
K = 9; Accuracy: 0.8199513381995134  
K = 11; Accuracy: 0.79808535279805353  
K = 13; Accuracy: 0.7907542579075426  
K = 15; Accuracy: 0.7761557177615572

```
In [57]: # from sklearn.model_selection import train_test_split
class KNeighborsClassifier(object):
    def __init__(self, k):
        self.k = k

    def Euclidean(self, point, data):
        return np.sqrt(np.sum((point - data)**2 , axis=1) )

    def Manhattan(self, point, data):
        return np.sum(np.abs(point - data) ,axis=1)

    def Chebyshev(self, point, data):
        return np.max(np.abs(point - data) ,axis=1)

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def most_common(self, List):
        return max(set(List), key=list.count)

    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.Euclidean(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(self.most_common, neighbors))

    def evaluate(self, X_test, y_test):
        y_pred = np.array(self.predict(X_test))
        accuracy = np.mean(y_pred == y_test)
        return accuracy
```

```
In [58]: from sklearn.model_selection import train_test_split
X= np.array(df.drop("label" , axis=1))
y=np.array(df["label"])
X_train , X_test , y_train , y_test= train_test_split(X , y , test_size=0.20 , random_state=123)

#normalize
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_traintnormal= sc.fit_transform(X_train)
x_testnormal=sc.transform(X_test)
```

```
#train
for k in range(3, 16,2):
    model = KNeighborsClassifier(k)
    model.fit(x_traintnormal, y_train)
    pred = model.predict(x_testnormal)
    acc = model.evaluate(x_testnormal, y_test)
    print("K = "+str(k)+" ; Accuracy: "+str(acc))
```

```
K = 3; Accuracy: 0.7980535279805353
K = 5; Accuracy: 0.805352798053528
K = 7; Accuracy: 0.8004866180048662
K = 9; Accuracy: 0.7688564476885644
K = 11; Accuracy: 0.7615571776155717
K = 13; Accuracy: 0.7372262773722628
K = 15; Accuracy: 0.7250608272506083
```

```
In [59]: from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
X= np.array(df.drop("label" , axis=1))
y=np.array(df[ "label" ])

X_train , X_test , y_train , y_test=train_test_split(X, y , test_size=0.20, random_state=123)

# Train SVM model
model = SVC(kernel="poly" , degree=5 ,C=1, gamma=1)
model.fit(X_train, y_train)

y_predict=model.predict(X_test)

#accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_predict , y_test))

0.880778588077859
```

## Exercise3 Concrete Slump and Strength

```
In [60]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

In [61]: df=pd.read_csv(r"D:\AI_MachineLearning\datasets\CONCRETE\ConcreteSlump\slump_test.data" , usecols=lambda column: column != 'No')
df
```

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	Compressive Strength (28-day)(Mpa)
0	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
1	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
2	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
3	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
4	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82
...	...	...	...	...	...	...	...	...	...	...
98	248.3	101.0	239.1	168.9	7.7	954.2	640.6	0.0	20.0	49.97
99	248.0	101.0	239.9	169.1	7.7	949.9	644.1	2.0	20.0	50.23
100	258.8	88.0	239.6	175.3	7.6	938.9	646.0	0.0	20.0	50.50
101	297.1	40.9	239.9	194.0	7.5	908.9	651.8	27.5	67.0	49.17
102	348.7	0.1	223.1	208.5	9.6	786.2	758.1	29.0	78.0	48.77

103 rows × 10 columns

```
In [62]: df2=df.rename(columns={"Compressive Strength (28-day)(Mpa)" : "target"})
```

```
In [63]: df2
```

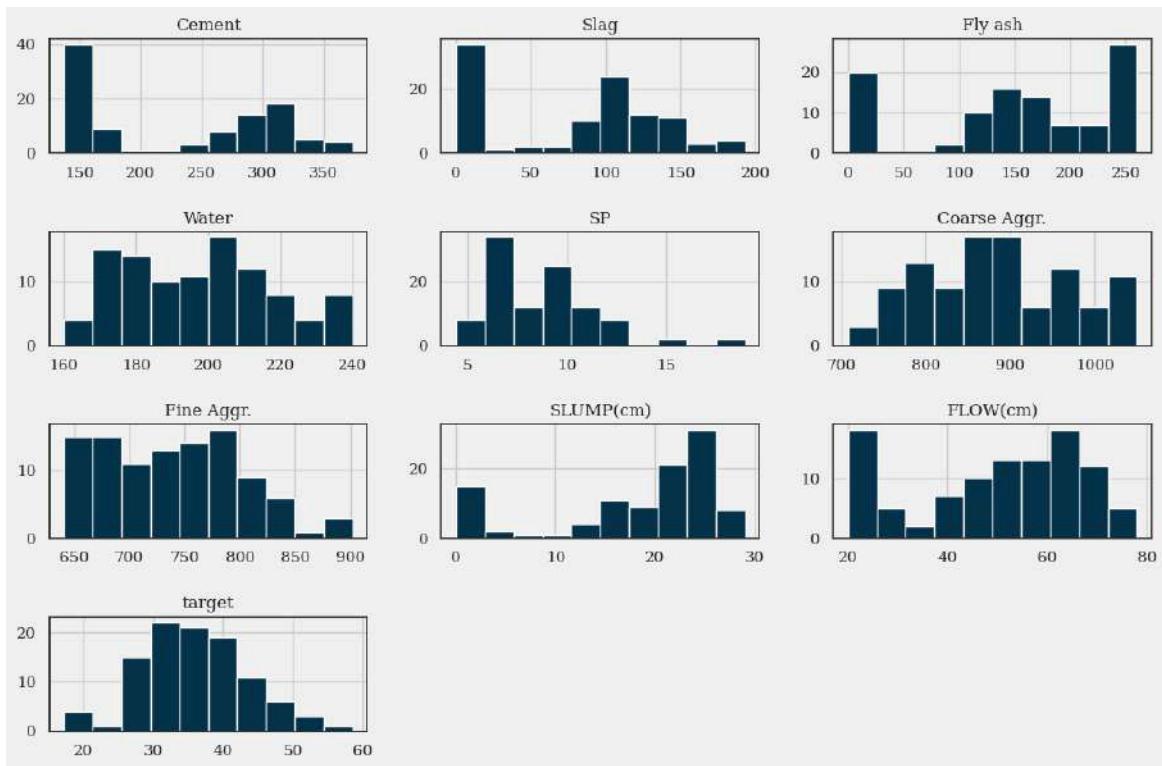
	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	target
0	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
1	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
2	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
3	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
4	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82
...	...	...	...	...	...	...	...	...	...	...
98	248.3	101.0	239.1	168.9	7.7	954.2	640.6	0.0	20.0	49.97
99	248.0	101.0	239.9	169.1	7.7	949.9	644.1	2.0	20.0	50.23
100	258.8	88.0	239.6	175.3	7.6	938.9	646.0	0.0	20.0	50.50
101	297.1	40.9	239.9	194.0	7.5	908.9	651.8	27.5	67.0	49.17
102	348.7	0.1	223.1	208.5	9.6	786.2	758.1	29.0	78.0	48.77

103 rows × 10 columns

```
In [64]: df2.describe().T.style.background_gradient(cmap='YlOrRd')
```

	count	mean	std	min	25%	50%	75%	max
Cement	103.000000	229.894175	78.877230	137.000000	152.000000	248.000000	303.900000	374.000000
Slag	103.000000	77.973786	60.461363	0.000000	0.050000	100.000000	125.000000	193.000000
Fly ash	103.000000	149.014563	85.418080	0.000000	115.500000	164.000000	235.950000	260.000000
Water	103.000000	197.167961	20.208158	160.000000	180.000000	196.000000	209.500000	240.000000
SP	103.000000	8.539806	2.807530	4.400000	6.000000	8.000000	10.000000	19.000000
Coarse Aggr.	103.000000	883.978641	88.391393	708.000000	819.500000	879.000000	952.800000	1049.900000
Fine Aggr.	103.000000	739.604854	63.342117	640.600000	684.500000	742.700000	788.000000	902.000000
SLUMP(cm)	103.000000	18.048544	8.750844	0.000000	14.500000	21.500000	24.000000	29.000000
FLOW(cm)	103.000000	49.610680	17.568610	20.000000	38.500000	54.000000	63.750000	78.000000
target	103.000000	36.039417	7.838232	17.190000	30.900000	35.520000	41.205000	58.530000

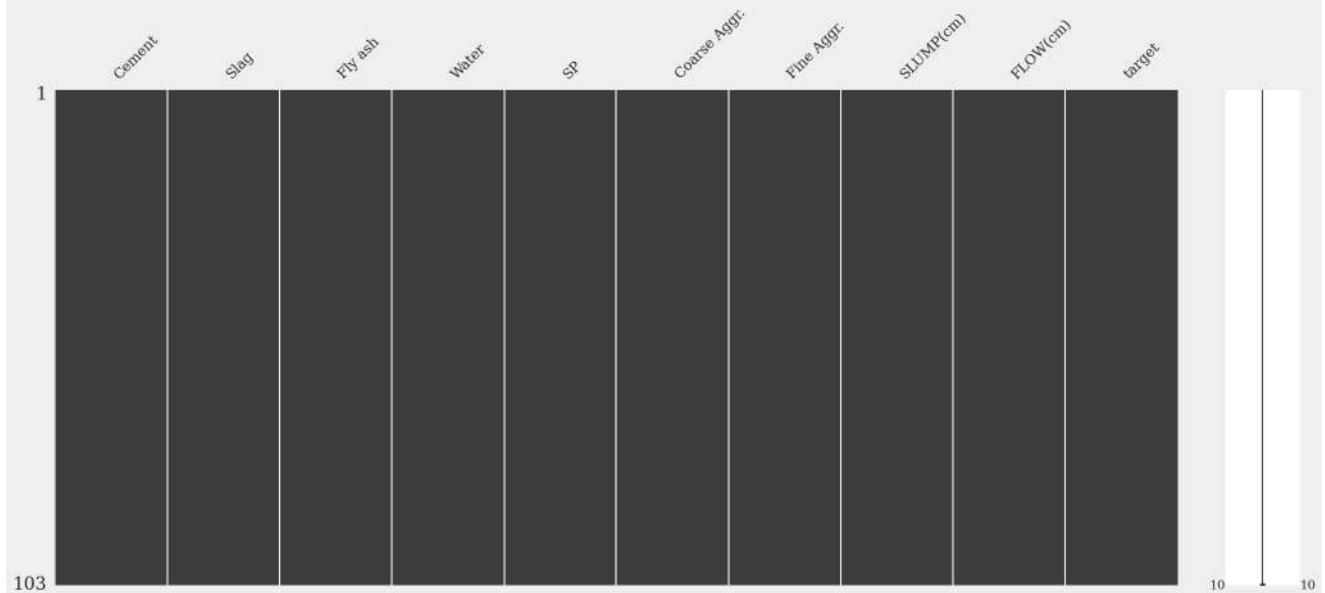
```
In [65]: df2.hist(figsize=(12,8) );
plt.tight_layout(pad=1.8)
```



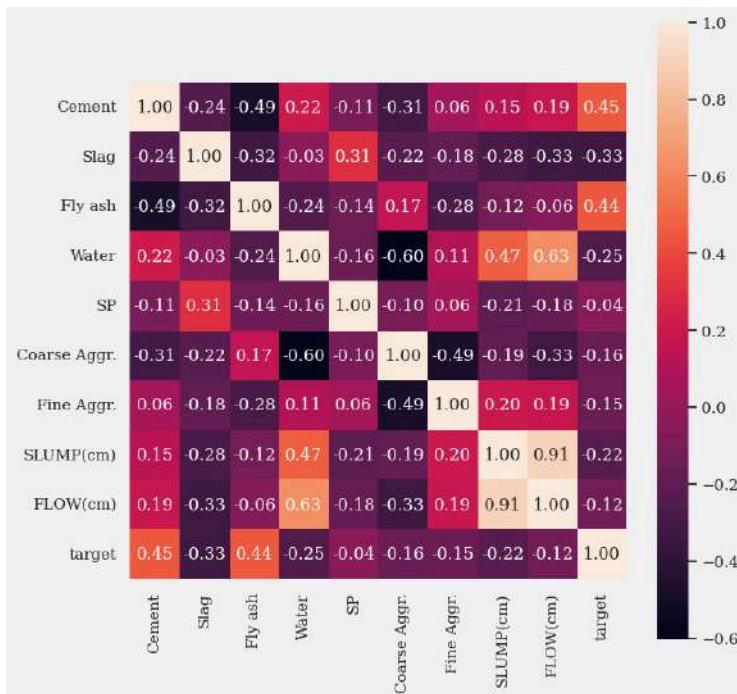
In [66]: `df2.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Cement       103 non-null    float64
 1   Slag         103 non-null    float64
 2   Fly ash     103 non-null    float64
 3   Water        103 non-null    float64
 4   SP           103 non-null    float64
 5   Coarse Aggr. 103 non-null    float64
 6   Fine Aggr.  103 non-null    float64
 7   SLUMP(cm)   103 non-null    float64
 8   FLOW(cm)    103 non-null    float64
 9   target       103 non-null    float64
dtypes: float64(10)
memory usage: 8.2 KB
```

In [67]: `import missingno as msno`  
`msno.matrix(df2);`



In [68]: `import seaborn as sns`  
`fig=plt.figure(figsize=(8,8))`  
`sns.heatmap(df2.corr(), annot=True, fmt='0.2f', square=True)`  
`plt.show()`



```
In [69]: df.columns
Out[69]: Index(['Cement', 'Slag', 'Fly ash', 'Water', 'SP', 'Coarse Aggr.',
   'Fine Aggr.', 'SLUMP(cm)', 'FLOW(cm)',
   'Compressive Strength (28-day)(Mpa)'],
  dtype='object')
```

```
In [70]: #df3=df[['Cement', 'Slag', 'Fly ash', 'Water', 'SP', 'Coarse Aggr.',
   'Fine Aggr.', 'FLOW(cm)', 'Compressive Strength (28-day)(Mpa)', 'SLUMP(cm)']]
#df3
```

**Cell In[70], line 2**  
 'Fine Aggr.', 'FLOW(cm)', 'Compressive Strength (28-day)(Mpa)', 'SLUMP(cm)']]  
 ^  
**IndentationError:** unexpected indent

```
In [71]: X=np.array(df2.iloc[:, :-1])
y=np.array(df2.iloc[:, -1])
y=y.reshape(-1, 1)
```

## Normal Equation

```
In [72]: #import Libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#train test split
X_train , X_test, y_train = train_test_split(X , y , test_size=0.2 , random_state=42)
```

```
In [73]: theta= np.linalg.pinv(X_train)@y_train
theta
```

```
Out[73]: array([[ 0.10147096],
   [ 0.02885457],
   [ 0.09286486],
   [-0.08264368],
   [ 0.22955971],
   [-0.00225078],
   [ 0.01830386],
   [-0.23467266],
   [ 0.07459518]])
```

```
In [74]: predict_train= X_train @theta
predict_test=X_test @theta
```

```
In [75]: #cost function train
numerator=sum ((y_train - predict_train)**2)
Denominator=sum ((y_train - y_train.mean())**2)

rsetr= numerator / Denominator

R_squaredtr= 1 - rsetr

print(f" the accuracy of the algorhithm is: {round(float(R_squaredtr*100), 2)} ")
the accuracy of the algorhithm is: 90.25
```

```
In [76]: #cost function test
numerator=sum ((y_test - predict_test)**2)
Denominator=sum ((y_test - y_test.mean())**2)

rsets= numerator / Denominator

R_squaredts= 1 - rsets

print(f" the accuracy of the algorhithm is: {round(float(R_squaredts*100), 2)} ")
the accuracy of the algorhithm is: 91.19
```

## sklearn

```
In [18]: from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(X_train , y_train)

print(reg.score(X_train , y_train))
```

```
print(reg.score(X_test , y_test))
0.9072349701566343
0.9119795179382083
```

## Gradient Descent

```
In [25]: # Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

def train(X, y, batch_size=32, iterations=10000, learning_rate=0.001): #stopping_threshold=1e-6
    m, n = X.shape

    # Initializing weight, bias, Learning rate and iterations
    current_weight = np.random.randn(n, 1)
    current_bias = np.zeros([1, 1])

    iterations = iterations
    learning_rate = learning_rate

    costs = []
    weights = []
    biases = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(iterations):
        # Shuffle the data
        indices = np.random.permutation(m)
        X_shuffled = X[indices]
        y_shuffled = y[indices]

        # Mini-batch gradient descent
        for j in range(0, m, batch_size):
            X_batch = X_shuffled[j:j+batch_size]
            y_batch = y_shuffled[j:j+batch_size]

            # Making predictions
            prediction = (X_batch @ current_weight) + current_bias

            # Calculating the current cost
            current_cost = np.sum((1 / (2 * batch_size)) * (prediction - y_batch) ** 2)

            # Calculating the gradients
            weight_derivative = (1 / batch_size) * X_batch.T @ (prediction - y_batch)
            bias_derivative = (1 / batch_size) * np.sum(prediction - y_batch)

            # Updating weights and bias
            current_weight = current_weight - (learning_rate * weight_derivative)
            current_bias = current_bias - (learning_rate * bias_derivative)

            # Calculate cost over entire dataset for monitoring
            prediction = (X @ current_weight) + current_bias
            total_cost = np.sum((1 / (2 * m)) * (prediction - y) ** 2)

            # If the change in cost is less than or equal to
            # stopping_threshold we stop the gradient descent
            if previous_cost and abs(previous_cost - total_cost) <= stopping_threshold:
                break

            previous_cost = total_cost

            costs.append(total_cost)
            weights.append(current_weight)
            biases.append(current_bias)

    return costs, weights, biases

X= np.array(df2.iloc[:, :-1])
y=np.array(df2.iloc[:, -1])
y=y.reshape(-1 , 1)

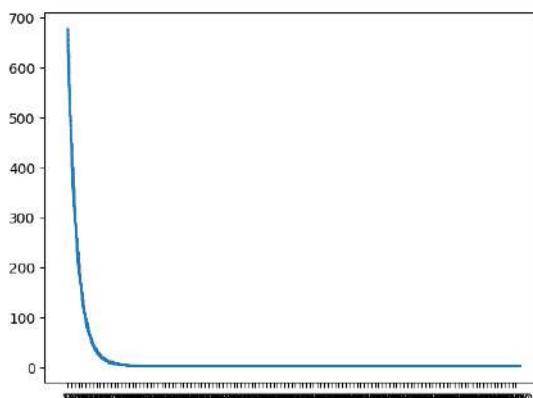
if __name__ == "__main__":
    # Assuming df is already defined
    X = np.array(df.iloc[:, :-1])
    y = np.array(df.iloc[:, -1])
    y = y.reshape(-1, 1)

    # Standardize the features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    costs, weights, biases = train(X_train, y_train)
```

```
In [26]: plt.plot([i for i in range(len(costs))] , costs , marker="+", markersize=2 )
plt.xticks([i for i in range(0,len(costs)) , 80])
plt.show()
```



```
In [27]: #prediction
final_weight = weights[-1]
final_bias = biases[-1]
```

```

predictionstrain = (X_train @ final_weight) + final_bias
predictionstest = (X_test @ final_weight) + final_bias

In [28]: #cost function test
numerator=sum ((y_train - predictionstrain)**2)
Denominator=sum ((y_train - y_train.mean())**2)

rsets= numerator / Denominator

R_squaredts= 1 - rsets

print(f" the accuracy of the algorihm is: {round(float(R_squaredts*100), 2)} ")
the accuracy of the algorihm is: 90.62

In [29]: #cost function test
numerator=sum ((y_test - predictionstest)**2)
Denominator=sum ((y_test - y_test.mean())**2)

rsets= numerator / Denominator

R_squaredts= 1 - rsets

print(f" the accuracy of the algorihm is: {round(float(R_squaredts*100), 2)} ")
the accuracy of the algorihm is: 91.27

```

## polynomial

```

In [30]: # Importing Libraries
import numpy as np
import math
import matplotlib.pyplot as plt

class PolynomialRegression:

    def __init__(self, degree, learning_rate, iterations , stopping_threshold):
        self.degree = degree
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.stopping_threshold = stopping_threshold

    def normalize(self, X, method=None):
        """
        zscore:
        It is a special case of the normal distribution where
        the mean (average) is 0 and the standard deviation is 1.

        maxabs:
        Rescale each feature between -1 and 1.

        minmax:
        The Min-Max Scaling (Normalization)
        technique works by transforming the original
        data into a new range, typically between 0 and 1.
        """
        if method is None or method == "zscore":
            return (X - X.mean(axis=0)) / X.std(axis=0)
        elif method == "maxabs":
            return X / np.abs(X.max(axis=0))
        elif method == "minmax":
            return (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

    # Transform method
    def transform(self, X):
        X_norm = self.normalize(X)
        num_samples, num_features = X_norm.shape
        X_transform = np.empty((num_samples, 0))
        for i in range(1, self.degree + 1):
            X_transform = np.concatenate((X_transform, np.power(X_norm, i)), axis=1)
        return X_transform

    # model training
    def fit(self, X, y):
        Xtansnorm = self.transform(X)
        m, n = Xtansnorm.shape

        # weight initialization
        current_weight = np.random.rand(n ,1)
        current_bias = np.zeros([1,1])

        costs = []
        weights = []
        biases = []
        previous_cost = None

        for i in range(self.iterations):
            # Making predictions
            prediction = (Xtansnorm @ current_weight) + current_bias
            # Calculating the current cost
            current_cost = np.sum((1 / (2 * m)) * np.square(prediction - y))

            # If the change in cost is Less than or equal to
            # stopping_threshold we stop the gradient descent
            #if previous_cost is not None and abs(previous_cost - current_cost) <= self.stopping_threshold:
            #    break

            previous_cost = current_cost
            costs.append(current_cost)
            weights.append(current_weight)
            biases.append(current_bias)

            # Calculating the gradients
            weight_derivative = Xtansnorm.T @ (prediction - y)/m
            bias_derivative = np.sum(prediction - y)/m

            # Updating weights and bias
            current_weight -= self.learning_rate * weight_derivative
            current_bias -= self.learning_rate * bias_derivative

        return costs, weights[-1], biases[-1]

    # predict
    def predict(self, X , y):
        _, predicted_weights, predicted_bias = self.fit(X, y) # Assuming y is defined somewhere
        X_transform = self.transform(X)
        return (X_transform @ predicted_weights) + predicted_bias

```

```
In [31]: from sklearn.model_selection import train_test_split

def main():
    # Assuming df is already defined
    X = np.array(df2.iloc[:, :-1])
    y = np.array(df2.iloc[:, -1])
    y = y.reshape(-1, 1)

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Model training
    model = PolynomialRegression(degree=3, learning_rate=0.01, iterations=10000, stopping_threshold=1e-6)
    model.fit(X_train, y_train)

    # Prediction on test set
    y_pred = model.predict(X_test, y_test)

    # Visualization
    plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual')
    plt.scatter(X_test[:, 0], y_pred, color='orange', label='Predicted')
    plt.title('X vs Y')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.show()

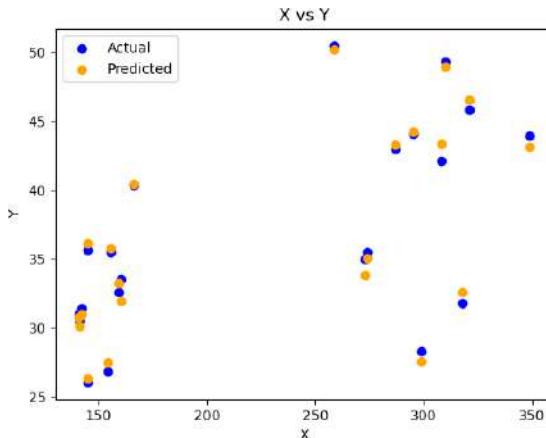
    return y_test, y_pred

# Call main function and show result
if __name__ == "__main__":
    y_test, predictions = main()

    # R-squared calculation
    numerator = np.sum((y_test - predictions)**2)
    denominator = np.sum((y_test - y_test.mean())**2)

    r_squared = 1 - (numerator / denominator)

    print(f"The accuracy of the algorithm is: {round(float(r_squared * 100), 2)}%")
```



The accuracy of the algorithm is: 99.05%

## sklearn

```
In [33]: import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

X=df2.iloc[:, :-1]
y=df2.iloc[:, -1]

poly = PolynomialFeatures(degree=3 )
X = poly.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

model = LinearRegression()
model.fit(X_train, y_train)
expected_y = y_test
predicted_y = model.predict(X_test)

print(model.score(X_train,y_train)*100)
print(model.score(X_test,y_test)*100)

100.0
92.95369647941108
```

## Fashion Mnist

0 T-shirt/top 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Sandal 6 Shirt 7 Sneaker 8 Bag 9 Ankle boot

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

In [4]: mnist_train=pd.read_csv(r"D:\AI_Machinelearning\datasets\fashion-mnist-master\mnist_train.csv")
mnist_test=pd.read_csv(r"D:\AI_Machinelearning\datasets\fashion-mnist-master\mnist_test.csv")

In [5]: dftrain=pd.DataFrame(mnist_train)
dftrain
```

```
Out[5]:
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel75	pixel76	pixel77	pixel78	pixel79	pixel80	pixel81	pixel82	pixel83	pixel84
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	0	0	0	0	0
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	1	0	0	0	0
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59995	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59996	1	0	0	0	0	0	0	0	0	0	...	73	0	0	0	0	0	0	0	0	0
59997	8	0	0	0	0	0	0	0	0	0	...	160	162	163	135	94	0	0	0	0	0
59998	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59999	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

60000 rows x 785 columns

```
In [37]: dftrain["label"].unique()
```

```
Out[37]: array([2, 9, 6, 0, 3, 4, 5, 8, 7, 1], dtype=int64)
```

```
In [40]: dftrain.isnull().sum().sum()
```

```
Out[40]: 0
```

```
In [6]: X_train , y_train = np.array(dftrain.drop("label" , axis=1)) ,np.array(dftrain["label"] )
```

```
In [9]: dftest=pd.DataFrame(mnist_test)
```

```
Out[9]:
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel75	pixel76	pixel77	pixel78	pixel79	pixel80	pixel81	pixel82	pixel83	pixel84
0	0	0	0	0	0	0	0	0	9	8	...	103	87	56	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	34	0	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	14	53	99	...	0	0	0	0	63	53	31	0	0	0
3	2	0	0	0	0	0	0	0	0	0	...	137	126	140	0	133	224	222	56	0	0
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0	0	0	0	0	0	0	0	0	0	...	32	23	14	20	0	0	1	0	0	0
9996	6	0	0	0	0	0	0	0	0	0	...	0	0	0	2	52	23	28	0	0	0
9997	8	0	0	0	0	0	0	0	0	0	...	175	172	172	182	199	222	42	0	1	0
9998	8	0	1	3	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
9999	1	0	0	0	0	0	0	0	140	119	...	111	95	75	44	1	0	0	0	0	0

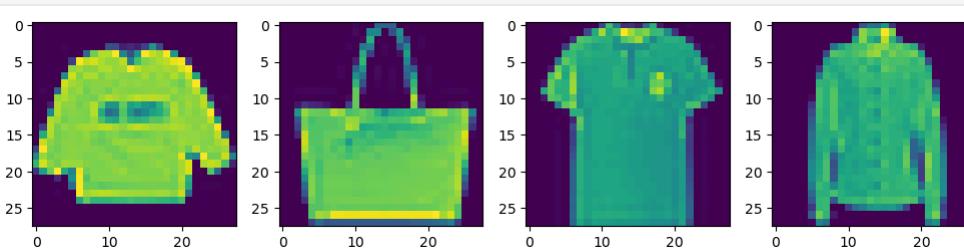
10000 rows x 785 columns

```
In [10]: X_test , y_test= np.array(dftest.drop("label" , axis=1)) , np.array(dftest["label"] )
```

```
In [44]: import matplotlib.pyplot as plt
```

```
#plt.gray()
fig , (ax1 ,ax2 ,ax3 , ax4)=plt.subplots(nrows=1 , ncols=4)
fig.set_size_inches(10, 8)
ax1.imshow((X_train[0].reshape(28,28)))
ax2.imshow((X_train[2000].reshape(28,28)))
ax3.imshow((X_train[30000].reshape(28,28)))
ax4.imshow((X_train[50000].reshape(28,28)))

plt.tight_layout()
plt.show()
```



## knn

```
In [15]: xxtr=X_train[:2000]
yytr=y_train[:2000]
```

```
In [16]: xxts=X_test[:500]
yyts=y_test[: 500]
```

```
In [17]: # normalizing the predictors
from sklearn.preprocessing import MinMaxScaler
normalize=MinMaxScaler()
X_normtr=normalize.fit_transform(xxtr)
X_normts=normalize.transform(xxts)
```

```
In [18]: from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
```

```
In [19]: knn.fit(X_normtr,yytr)
```

```
Out[19]: KNeighborsClassifier()
KNeighborsClassifier()
```

```
In [20]: y_pred=knn.predict(X_normts)
y_pred
```

```
Out[20]: array([0, 1, 2, 2, 4, 6, 8, 6, 5, 0, 3, 2, 4, 6, 8, 5, 6, 3, 6, 2, 4, 4,
   2, 1, 5, 9, 8, 6, 4, 1, 9, 7, 7, 8, 1, 0, 9, 8, 0, 8, 2, 6, 2, 2,
   2, 0, 3, 3, 2, 3, 2, 4, 9, 3, 0, 9, 9, 4, 2, 0, 4, 9, 6, 6, 1, 1,
   6, 9, 7, 2, 7, 3, 4, 6, 5, 7, 1, 6, 1, 3, 9, 6, 1, 4, 4, 8, 9, 4,
   1, 6, 3, 4, 4, 2, 0, 4, 7, 7, 3, 9, 3, 9, 6, 2, 2, 3, 2, 2, 7,
   5, 5, 4, 4, 7, 5, 0, 2, 7, 3, 6, 5, 4, 4, 7, 0, 5, 5, 0, 3, 1, 7,
   9, 4, 9, 6, 4, 6, 0, 4, 3, 3, 3, 4, 2, 6, 3, 0, 1, 3, 2, 3, 3, 1,
   9, 3, 3, 3, 9, 5, 6, 7, 7, 3, 6, 4, 0, 0, 7, 2, 2, 8, 9, 0, 2, 4,
   2, 8, 7, 9, 7, 1, 3, 9, 1, 7, 5, 6, 0, 7, 4, 9, 1, 6, 0, 0, 6, 4,
   2, 9, 6, 2, 2, 5, 5, 8, 6, 4, 1, 9, 2, 2, 7, 1, 9, 5, 9, 4, 2, 8,
   5, 9, 9, 3, 2, 4, 5, 9, 8, 1, 9, 5, 0, 2, 9, 7, 4, 0, 9, 2, 1, 5,
   7, 7, 0, 2, 4, 5, 3, 3, 8, 1, 6, 4, 4, 8, 4, 9, 2, 3, 3, 2, 4, 4,
   4, 9, 3, 9, 2, 8, 6, 8, 8, 2, 7, 2, 5, 5, 9, 7, 8, 2, 4, 8, 3, 8,
   3, 1, 4, 2, 6, 1, 8, 1, 3, 0, 3, 1, 7, 0, 7, 2, 0, 9, 6, 2, 7, 4,
   7, 7, 1, 0, 6, 0, 7, 8, 7, 0, 9, 2, 0, 2, 7, 0, 8, 5, 8, 7, 7, 2,
   1, 2, 9, 7, 6, 6, 1, 6, 3, 8, 0, 5, 5, 0, 9, 2, 9, 8, 0, 6, 9, 6,
   4, 3, 0, 2, 0, 8, 5, 9, 5, 2, 2, 0, 7, 1, 2, 7, 9, 6, 2, 0, 2,
   2, 2, 0, 7, 2, 9, 8, 3, 9, 7, 5, 1, 4, 4, 9, 5, 5, 0, 3, 6, 7, 4,
   2, 2, 0, 5, 0, 2, 0, 2, 3, 5, 7, 0, 5, 0, 9, 0, 5, 6, 7, 7, 5, 6,
   4, 4, 0, 4, 0, 9, 8, 4, 1, 0, 0, 2, 4, 9, 4, 3, 1, 2, 5, 8, 3, 8,
   2, 7, 1, 2, 0, 7, 8, 7, 6, 5, 1, 0, 3, 0, 5, 8, 4, 9, 7, 5, 8,
   3, 0, 0, 2, 7, 1, 5, 2, 7, 5, 9, 0, 0, 7, 0, 2, 7, 6, 8, 6, 3, 7,
   3, 1, 4, 9, 7, 0, 8, 2, 2, 6, 8, 1, 7, 1, 9, 3], dtype=int64)
```

```
In [21]: from sklearn.metrics import accuracy_score
```

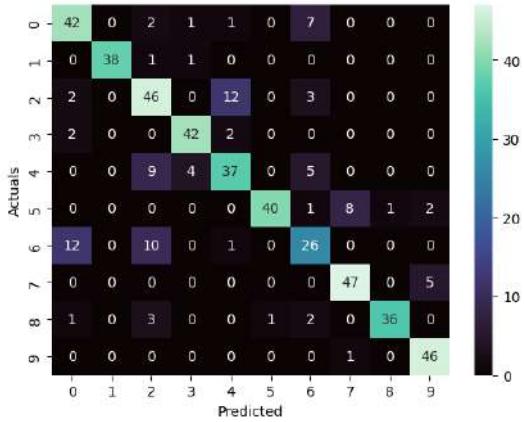
```
acc=accuracy_score(yyts , y_pred)*100
acc
```

```
Out[21]: 80.0
```

```
In [52]: from sklearn.metrics import confusion_matrix
```

```
# creating confusion matrix for this training set
sns.heatmap(confusion_matrix(yyts,y_pred), annot=True, cmap='mako', fmt='.5g')
plt.xlabel('Predicted')
plt.ylabel('Actuals')
```

```
Out[52]: Text(50.72222222222214, 0.5, 'Actuals')
```



## naive\_bayes

```
In [11]: from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_trainnormal=sc.fit_transform(X_train)
x_testnormal=sc.transform(X_test)
```

```
In [12]: from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_trainnormal , y_train)
```

```
Out[12]: GaussianNB()
GaussianNB()
```

```
In [13]: y_hat=model.predict(x_testnormal)
```

```
In [14]: from sklearn.metrics import accuracy_score
acc=accuracy_score(y_test , y_hat)
acc
```

```
Out[14]: 0.5791
```

## Exercise 5

```
In [4]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [5]: df=pd.read_csv(r"D:/AI_Machinlearning/datasets/building/housePrice.csv")
df
```

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)
0	63	1	True	True	True	Shahran	1.850000e+09	61666.67
1	60	1	True	True	True	Shahran	1.850000e+09	61666.67
2	79	2	True	True	True	Pardis	5.500000e+08	18333.33
3	95	2	True	True	True	Shahrake Qods	9.025000e+08	30083.33
4	123	2	True	True	True	Shahrake Gharb	7.000000e+09	233333.33
...	...	...	...	...	...	...	...	...
3474	86	2	True	True	True	Southern Janatabad	3.500000e+09	116666.67
3475	83	2	True	True	True	Niavaran	6.800000e+09	226666.67
3476	75	2	False	False	False	Parand	3.650000e+08	12166.67
3477	105	2	True	True	True	Dorous	5.600000e+09	186666.67
3478	82	2	False	True	True	Parand	3.600000e+08	12000.00

3479 rows × 8 columns

In [6]: `df.sample(5)`

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)
563	145	2	True	True	True	Zaferanieh	1.100000e+10	366666.67
761	140	3	True	True	True	Ozgol	6.300000e+09	210000.00
2707	61	1	False	True	False	Shahr-e-Ziba	1.490000e+09	49666.67
1502	85	2	True	True	True	Dorous	4.200000e+09	140000.00
1456	155	2	True	True	True	Pardis	1.900000e+09	63333.33

In [7]: `df.dtypes`

```
Out[7]: Area          object
Room         int64
Parking       bool
Warehouse     bool
Elevator      bool
Address       object
Price        float64
Price(USD)   float64
dtype: object
```

In [8]: `df['Area'] = pd.to_numeric(df['Area'], errors='coerce') #df.Area , df.iloc[:, 0]`  
df

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)
0	63.0	1	True	True	True	Shahran	1.850000e+09	61666.67
1	60.0	1	True	True	True	Shahran	1.850000e+09	61666.67
2	79.0	2	True	True	True	Pardis	5.500000e+08	18333.33
3	95.0	2	True	True	True	Shahrake Qods	9.025000e+08	30083.33
4	123.0	2	True	True	True	Shahrake Gharb	7.000000e+09	233333.33
...	...	...	...	...	...	...	...	...
3474	86.0	2	True	True	True	Southern Janatabad	3.500000e+09	116666.67
3475	83.0	2	True	True	True	Niavaran	6.800000e+09	226666.67
3476	75.0	2	False	False	False	Parand	3.650000e+08	12166.67
3477	105.0	2	True	True	True	Dorous	5.600000e+09	186666.67
3478	82.0	2	False	True	True	Parand	3.600000e+08	12000.00

3479 rows × 8 columns

In [9]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3479 entries, 0 to 3478
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
 ---  -- 
 0   Area      3479 non-null   float64
 1   Room      3479 non-null   int64  
 2   Parking    3479 non-null   bool    
 3   Warehouse  3479 non-null   bool    
 4   Elevator   3479 non-null   bool    
 5   Address    3456 non-null   object  
 6   Price      3479 non-null   float64
 7   Price(USD) 3479 non-null   float64
dtypes: bool(3), float64(3), int64(1), object(1)
memory usage: 146.2+ KB
```

In [10]: `df.isnull().sum()`

```
Out[10]: Area      6
Room      0
Parking    0
Warehouse  0
Elevator   0
Address    23
Price      0
Price(USD) 0
dtype: int64
```

In [12]: `!pip install ydata_profiling`

^C

In [13]: `from ydata_profiling import ProfileReport`

```
In [14]: # Create a report by pandas profiling
report = ProfileReport(
    df,
    title="House Price (Tehran, Iran)",
    dataset={
        "description": "This profiling report has been generated for the purpose of analyzing and exploring House Price dataset.",
        "author": "Morteza Khorsand",
    },
    variables={
        "descriptions": {
            "Area": "Area in square meters",
            "Room": "Number of bedrooms",
            "Parking": "Has Parking or not",
            "Elevator": "Has elevator or not",
        }
    }
)
```



# Overview

Dataset statistics

Number of variables	8
Number of observations	3479
Missing cells	29
Missing cells (%)	0.1%
Duplicate rows	161
Duplicate rows (%)	4.6%
Total size in memory	146.2 KiB
Average record size in memory	43.0 B

Variable types

Numeric	4
Boolean	3
Text	1

Dataset

Description	This profiling report has been generated for the purpose of analyzing and exploring House Price dataset.
Author	Morteza Khorsand
URL	()

Variable descriptions

Area	Area in square meters
------	-----------------------

# Overview

Dataset statistics

Number of variables	8
Number of observations	3479
Missing cells	29
Missing cells (%)	0.1%
Duplicate rows	161
Duplicate rows (%)	4.6%
Total size in memory	146.2 KiB
Average record size in memory	43.0 B

Variable types

Numeric	4
Boolean	3
Text	1

Dataset

Description	This profiling report has been generated for the purpose of analyzing and exploring House Price dataset.
Author	Morteza Khorsand
URL	()

Variable descriptions

Area	Area in square meters
------	-----------------------

Out[14]:

```
In [8]: # Drop the missing values
df = df.dropna()
df.shape
```

Out[8]: (3450, 8)

```
In [9]: # Find duplicates based on all columns
df.duplicated().sum()
```

Out[9]: 208

```
In [10]: # Drop the duplicated rows
df = df.drop_duplicates()
df.shape
```

Out[10]: (3242, 8)

```
In [11]: # Reset dataframe index
df.reset_index(drop = True, inplace = True)
```

df

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)
0	63.0	1	True	True	True	Shahran	1.850000e+09	61666.67
1	60.0	1	True	True	True	Shahran	1.850000e+09	61666.67
2	79.0	2	True	True	True	Pardis	5.500000e+08	18333.33
3	95.0	2	True	True	True	Shahrake Qods	9.025000e+08	30083.33
4	123.0	2	True	True	True	Shahrake Gharb	7.000000e+09	233333.33
...	...	...	...	...	...	...	...	...
3237	63.0	1	True	True	False	Feiz Garden	1.890000e+09	63000.00
3238	86.0	2	True	True	True	Southern Janatabad	3.500000e+09	116666.67
3239	83.0	2	True	True	True	Niavaran	6.800000e+09	226666.67
3240	105.0	2	True	True	True	Dorous	5.600000e+09	186666.67
3241	82.0	2	False	True	True	Parand	3.600000e+08	12000.00

3242 rows × 8 columns

```
In [12]: df.replace({True : 1 , False : 0} , inplace= True)
df
```

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)
0	63.0	1	1	1	1	Shahran	1.850000e+09	61666.67
1	60.0	1	1	1	1	Shahran	1.850000e+09	61666.67
2	79.0	2	1	1	1	Pardis	5.500000e+08	18333.33
3	95.0	2	1	1	1	Shahrake Qods	9.025000e+08	30083.33
4	123.0	2	1	1	1	Shahrake Gharb	7.000000e+09	233333.33
...	...	...	...	...	...	...	...	...
3237	63.0	1	1	1	0	Feiz Garden	1.890000e+09	63000.00
3238	86.0	2	1	1	1	Southern Janatabad	3.500000e+09	116666.67
3239	83.0	2	1	1	1	Niavaran	6.800000e+09	226666.67
3240	105.0	2	1	1	1	Dorous	5.600000e+09	186666.67
3241	82.0	2	0	1	1	Parand	3.600000e+08	12000.00

3242 rows × 8 columns

```
In [13]: df.Address.unique()
```

```
Out[13]: array(['Shahran', 'Pardis', 'Shahrake Qods', 'Shahrake Gharb',
   'North Program Organization', 'Andisheh', 'West Ferdows Boulevard',
   'Narmak', 'Saadat Abad', 'Zafar', 'Islamshahr', 'Pirouzi',
   'Shahrake Shahid Bagheri', 'Moniriye', 'Velenjak', 'Amirieh',
   'Southern Janatabad', 'Salsabil', 'Zargandeh', 'Feiz Garden',
   'Water Organization', 'ShahrAra', 'Gisha', 'Ray', 'Abbasabad',
   'Ostad Moein', 'Farmanieh', 'Parand', 'Punak', 'Qasr-od-Dash',
   'Agdasieh', 'Pakdasht', 'Railway', 'Central Janatabad',
   'East Ferdows Boulevard', 'Pakdash', 'KhatunAbad', 'Sattarkhan',
   'Baghestan', 'Shahryar', 'Northern Janatabad', 'Daryan No',
   'Southern Program Organization', 'Rudhen', 'West Pars', 'Afsarieh',
   'Marzdaran', 'Dorous', 'Sadeghieh', 'Chahardangeh', 'Baqershahr',
   'Jeyhoon', 'Lavizan', 'Shams Abad', 'Fatemi',
   'Keshavarz Boulevard', 'Kahrizak', 'Qarchak',
   'Northren Jamalzadeh', 'Azarbajian', 'Bahar',
   'Persian Gulf Martyrs Lake', 'Beryanak', 'Heshmatieh',
   'Elm-o-Sanat', 'Golestan', 'Shahr-e-Ziba', 'Pasdaran',
   'Chardivari', 'Gheitarieh', 'Kamranieh', 'Gholhak', 'Heravi',
   'Hashemi', 'Dekhake Olimpic', 'Damavand', 'Republic', 'Zafaranieh',
   'Qazvin Imamzadeh Hassan', 'Niavaran', 'Valiasr', 'Qalandari',
   'Amir Bahador', 'Ekhtiarieh', 'Ekbatan', 'Absard', 'Haft Tir',
   'Mahallati', 'Ozgol', 'Tajrish', 'Abazar', 'Kohosar', 'Hekmat',
   'Parastan', 'Lavasan', 'Majidieh', 'Southern Chitgan', 'Karimkhan',
   'Si Metri Ji', 'Karoon', 'Northern Chitgan', 'East Pars', 'Kook',
   'Ain force', 'Sohanak', 'Komeil', 'Azadshahr', 'Zibadasht',
   'Amirabad', 'Dezashib', 'Elahieh', 'Mirdamad', 'Razi', 'Jordan',
   'Mahmoudieh', 'Shahedshahr', 'Yaftabadi', 'Mehrzan', 'Nasim Shahr',
   'Tenant', 'Chardangeh', 'Fallah', 'Eskandari', 'Shahrake Naft',
   'Ajudaniye', 'Tehransan', 'Nawab', 'Yousef Abad',
   'Northern Suhraward', 'Villa', 'Hakimiye', 'Nezamabad',
   'Garden of Saba', 'Terasht', 'Azari', 'Shahrake Apadana', 'Araj',
   'Vahidieh', 'Malard', 'Shahrake Azadi', 'Darband', 'Vanak',
   'Tehran Now', 'Darabadi', 'Eram', 'Atabak', 'Sabalan', 'Sabashahr',
   'Shahrake Madaen', 'Waterfall', 'Ahang', 'Salehabad', 'Pishva',
   'Enghelab', 'Islamshahr Elahieh', 'Ray - Montazeri',
   'Firoozkooh Kuhsar', 'Ghoba', 'Mehrabad', 'Southern Suhraward',
   'Abuzar', 'Dolatabad', 'Hor Square', 'Taslihat', 'Kazemabad',
   'Robat Karim', 'Ray - Pilgosh', 'Ghiyamdasht', 'Telecommunication',
   'Mirza Shirazi', 'Gandhi', 'Argentina', 'Seyed Khandan',
   'Shahrake Quds', 'Safadasht', 'Khademabad Garden', 'Hassan Abad',
   'Chidz', 'Khavaran', 'Boloorsazi', 'Mehrabad River River',
   'Varamin - Beheshti', 'Shoosh', 'Thirteen November', 'Darakeh',
   'Aliabad South', 'Alborz Complex', 'Firoozkooh', 'Vahidiye',
   'Shababad', 'Nazabad', 'Javadiyeh', 'Yakhchiabad'], dtype=object)
```

```
In [14]: len(df.Address.unique())
```

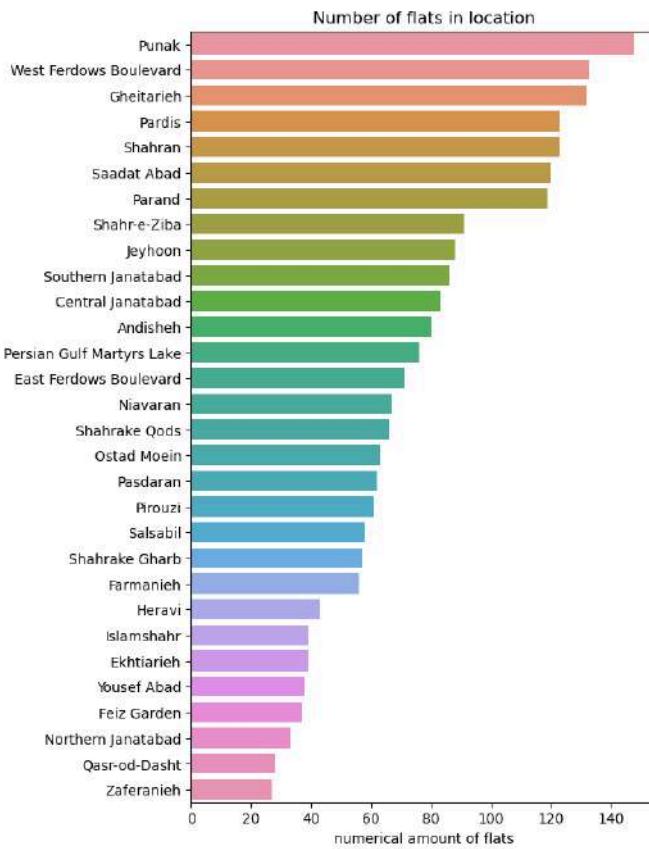
192

Out[14]:

```
In [15]: df3 = df['Address'].value_counts().copy()
df3 = df3[:30]
df3
```

```
Out[15]: Punak          148
West Ferdows Boulevard    133
Gheitarieh            132
Pardis                 123
Shahran                123
Saadat Abad             120
Parand                  119
Shahr-e-Ziba            91
Jeyhoon                 88
Southern Janatabad        86
Central Janatabad          83
Andisheh                80
Persian Gulf Martyrs Lake   76
East Ferdows Boulevard      71
Niavaran                 67
Shahrake Qods             66
Ostad Moein                63
Pasdaran                  62
Pirouzi                   61
Salsabil                  58
Shahrake Gharb              57
Farmanieh                  56
Heravi                     43
Islamshahr                  39
Ekhtiarieh                  39
Yousef Abad                  38
Feiz Garden                  37
Northern Janatabad             33
Qasr-od-Dasht                28
Zaferanieh                  27
Name: Address, dtype: int64
```

```
In [16]: fig, ax = plt.subplots(figsize=(6,10))
sns.barplot(x=df3.values, y=df3.index,ax=ax)
plt.xlabel('numerical amount of flats')
plt.title('Number of flats in location')
plt.show()
```



```
In [17]: df.sort_values('Price', ascending=False)[['Address']].head(20)
```

Out[17]:	Address
1606	Zaferanieh
1704	Abazar
405	Lavasan
770	Ekhtiarieh
1249	Niavarans
1593	Zafar
2855	Dorous
2908	Tajrish
782	Mahmoudieh
2242	Aqdasieh
2050	Niavarans
2536	Gandhi
1751	Elahieh
1321	North Program Organization
1534	Vanak
1191	Mirdamad
414	Niavarans
437	Farmarieh
406	Lavasan
547	Niavarans

```
In [18]: Area_new_feature = []

for i in df['Area']:
    if i <= 30:
        Area_new_feature.append("Small")

    elif (i > 30 and i<=90):
        Area_new_feature.append("UnderMean")

    elif (i>90 and i <= 120):
        Area_new_feature.append("UpperMean")

    elif i > 120:
        Area_new_feature.append("High")
    else:
        Area_new_feature.append(np.nan)

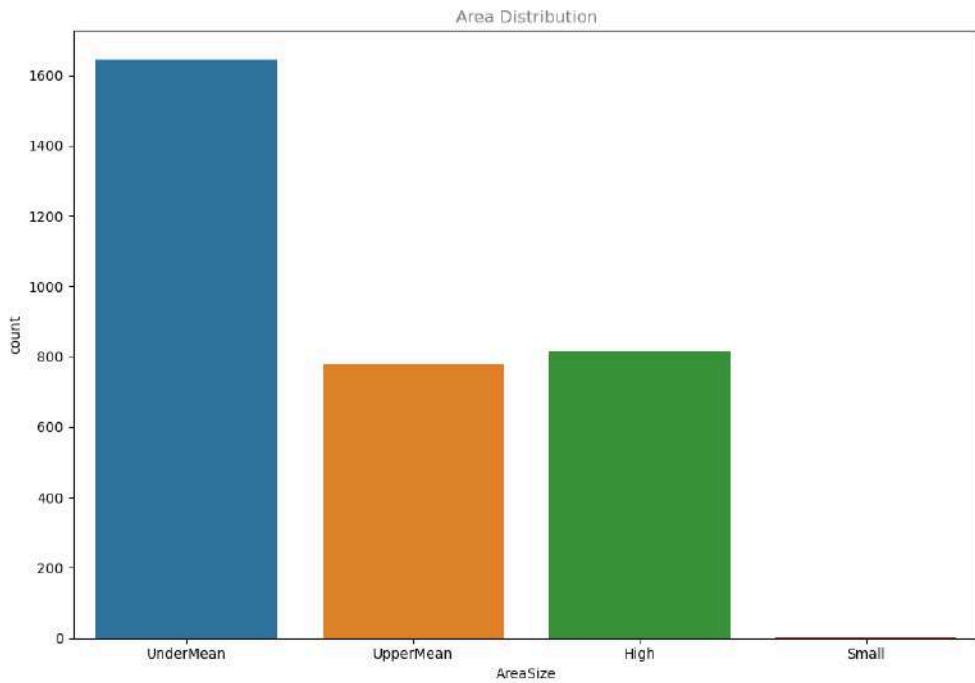
df["AreaSize"] = Area_new_feature

df
```

Out[18]:	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)	AreaSize
0	63.0	1	1	1	1	Shahran	1.850000e+09	61666.67	UnderMean
1	60.0	1	1	1	1	Shahran	1.850000e+09	61666.67	UnderMean
2	79.0	2	1	1	1	Pardis	5.500000e+08	18333.33	UnderMean
3	95.0	2	1	1	1	Shahrake Qods	9.025000e+08	30083.33	UpperMean
4	123.0	2	1	1	1	Shahrake Gharb	7.000000e+09	233333.33	High
...	...	...	...	...	...	...	...	...	...
3237	63.0	1	1	1	0	Feiz Garden	1.890000e+09	63000.00	UnderMean
3238	86.0	2	1	1	1	Southern Janatabad	3.500000e+09	116666.67	UnderMean
3239	83.0	2	1	1	1	Niavarans	6.800000e+09	226666.67	UnderMean
3240	105.0	2	1	1	1	Dorous	5.600000e+09	186666.67	UpperMean
3241	82.0	2	0	1	1	Parand	3.600000e+08	12000.00	UnderMean

3242 rows × 9 columns

```
In [19]: plt.figure(figsize=(10,7))
sns.countplot(x="AreaSize" , data= df)
plt.title("Area Distribution" , color='gray')
plt.tight_layout()
```



```
In [20]: Price_new_feature = []
H25 = df['Price'].describe()[4]
H50 = df['Price'].describe()[5]
H75 = df['Price'].describe()[6]

for i in df['Price']:
    if i <= H25:
        Price_new_feature.append("Cheap")
    elif (i > H25 and i <= H50):
        Price_new_feature.append("UnderMean")
    elif (i > H50 and i <= H75):
        Price_new_feature.append("UpperMean")
    elif i > H75:
        Price_new_feature.append("Expensive")
    else:
        Price_new_feature.append(np.nan)

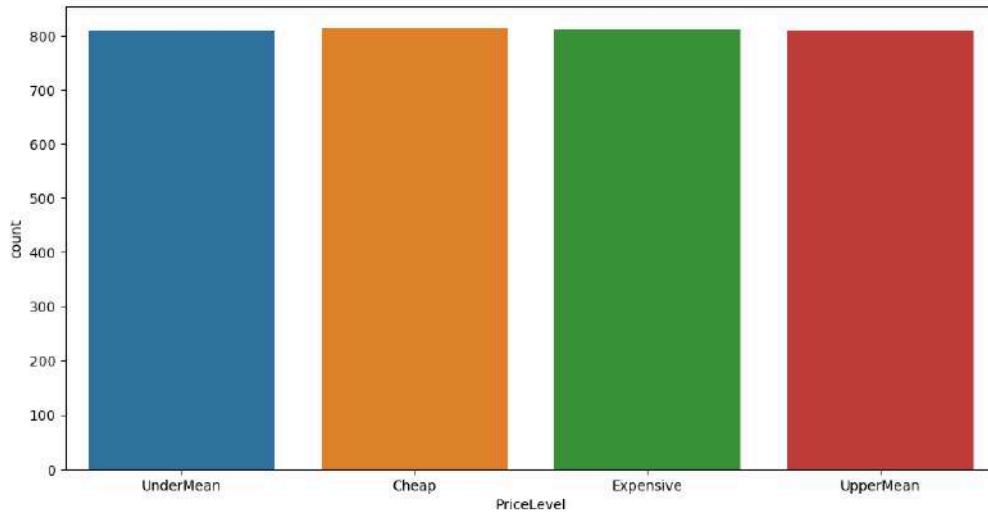
df["PriceLevel"] = Price_new_feature
```

```
Out[20]:
```

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)	AreaSize	PriceLevel
0	63.0	1	1	1	1	Shahran	1.850000e+09	61666.67	UnderMean	UnderMean
1	60.0	1	1	1	1	Shahran	1.850000e+09	61666.67	UnderMean	UnderMean
2	79.0	2	1	1	1	Pardis	5.500000e+08	18333.33	UnderMean	Cheap
3	95.0	2	1	1	1	Shahrake Qods	9.025000e+08	30083.33	UpperMean	Cheap
4	123.0	2	1	1	1	Shahrake Gharb	7.000000e+09	233333.33	High	Expensive
...	...	...	...	...	...	...	...	...	...	...
3237	63.0	1	1	1	0	Feiz Garden	1.890000e+09	63000.00	UnderMean	UnderMean
3238	86.0	2	1	1	1	Southern Janatabad	3.500000e+09	116666.67	UnderMean	UpperMean
3239	83.0	2	1	1	1	Niavaran	6.800000e+09	226666.67	UnderMean	Expensive
3240	105.0	2	1	1	1	Dorous	5.600000e+09	186666.67	UpperMean	UpperMean
3241	82.0	2	0	1	1	Parand	3.600000e+08	12000.00	UnderMean	Cheap

3242 rows × 10 columns

```
In [19]: plt.figure(figsize=(12,6))
sns.countplot(x='PriceLevel', data = df )
Out[19]: <Axes: xlabel='PriceLevel', ylabel='count'>
```



Thats Your Turn ...

"Classification and regression algorithms are ready for you."

## KMeans

```
In [65]: import numpy as np

class KMeansCustom:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.centroids = None
        self.cluster_ids = None

    def fit(self, X, initial_centroids=None):
        m = X.shape[0]
        if initial_centroids is None:
            indices = np.random.choice(m, self.n_clusters, replace=False)
            self.centroids = X[indices]
        else:
            self.centroids = initial_centroids

        self.cluster_ids = np.zeros(m)
        cost = 0

        for _ in range(self.max_iter):
            new_cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - self.centroids, axis=1)) for i in range(m)])

            # Update cluster centers
            for j in range(self.n_clusters):
                if np.any(new_cluster_ids == j):
                    self.centroids[j] = np.mean(X[new_cluster_ids == j], axis=0)

            # Stop
            if np.all(new_cluster_ids == self.cluster_ids):
                for z in range(self.n_clusters):
                    cost += (1/m) * np.sum(np.linalg.norm(X[new_cluster_ids == z] - self.centroids[z], axis=1) ** 2)
                self.cluster_ids = new_cluster_ids
                return cost, self.cluster_ids, self.centroids

            else:
                self.cluster_ids = new_cluster_ids

        # If it reaches here, it means max_iter was reached without convergence
        for z in range(self.n_clusters):
            cost += (1/m) * np.sum(np.linalg.norm(X[self.cluster_ids == z] - self.centroids[z], axis=1) ** 2)
        return cost, self.cluster_ids, self.centroids

    def predict(self, X):
        return np.array([np.argmin(np.linalg.norm(x - self.centroids, axis=1)) for x in X])
```

## kmeans++

```
In [66]: import numpy as np

class KMeansPlusPlus:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.centroids = None
        self.cluster_ids = None

    def _initialize_centroids(self, X):
        m = X.shape[0]
        self.centroids = np.zeros((self.n_clusters, X.shape[1]))

        # Initialize the first centroid randomly
        self.centroids[0] = X[np.random.choice(m)]

        for i in range(1, self.n_clusters):
            # Compute the distance from each point to the closest centroid
            distances = np.min(np.linalg.norm(X[:, np.newaxis] - self.centroids[:i], axis=2), axis=1)
            # Square the distances
            distances_squared = distances ** 2
            # Choose the next centroid with probability proportional to the distance squared
            probabilities = distances_squared / np.sum(distances_squared)
            chosen_index = np.random.choice(m, p=probabilities)
            self.centroids[i] = X[chosen_index]

    def fit(self, X, initial_centroids=None):
        m = X.shape[0]
        if initial_centroids is None:
            self._initialize_centroids(X)
        else:
            self.centroids = initial_centroids

        self.cluster_ids = np.zeros(m)
        cost = 0
        iter_count = 0

        while True:
            new_cluster_ids = np.array([np.argmin(np.linalg.norm(X[i] - self.centroids, axis=1)) for i in range(m)])

            # Update cluster centers
            for j in range(self.n_clusters):
                if np.any(new_cluster_ids == j):
                    self.centroids[j] = np.mean(X[new_cluster_ids == j], axis=0)

            # Stop if convergence or max_iter reached
            if np.all(new_cluster_ids == self.cluster_ids) or iter_count >= self.max_iter:
                for z in range(self.n_clusters):
                    cost += (1/m) * np.sum(np.linalg.norm(X[new_cluster_ids == z] - self.centroids[z], axis=1) ** 2)
                self.cluster_ids = new_cluster_ids
                return cost, self.cluster_ids, self.centroids

            else:
                self.cluster_ids = new_cluster_ids

            iter_count += 1

        # If it reaches here, it means max_iter was reached without convergence
        for z in range(self.n_clusters):
            cost += (1/m) * np.sum(np.linalg.norm(X[self.cluster_ids == z] - self.centroids[z], axis=1) ** 2)
        return cost, self.cluster_ids, self.centroids

    def predict(self, X):
        return np.array([np.argmin(np.linalg.norm(x - self.centroids, axis=1)) for x in X])
```

## KMedoids

```
In [67]: # import numpy as np

class KMedoidsCustom:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
```

```

    self.max_iter = max_iter
    self.medoids = None
    self.cluster_ids = None

    def _initialize_medoids(self, X):
        m = X.shape[0]
        indices = np.random.choice(m, self.n_clusters, replace=False)
        self.medoids = X[indices]

    def fit(self, X):
        m = X.shape[0]
        self._initialize_medoids(X)

        self.cluster_ids = np.zeros(m, dtype=int)
        cost = 0
        iter_count = 0

        while True:
            # Assign each point to the nearest medoid
            new_cluster_ids = np.array([np.argmin([np.linalg.norm(X[i] - medoid) for medoid in self.medoids]) for i in range(m)])

            # Update medoids
            new_medoids = np.copy(self.medoids)
            for j in range(self.n_clusters):
                cluster_points = X[new_cluster_ids == j]
                if len(cluster_points) > 0:
                    medoid_costs = np.sum(np.linalg.norm(cluster_points[:, np.newaxis] - cluster_points, axis=2), axis=0)
                    new_medoids[j] = cluster_points[np.argmin(medoid_costs)]

            # Stop if convergence or max_iter reached
            if np.array_equal(new_medoids, self.medoids) or iter_count >= self.max_iter:
                cost = np.sum([np.sum(np.linalg.norm(X[new_cluster_ids == j] - self.medoids[j], axis=1)) for j in range(self.n_clusters)])
                self.cluster_ids = new_cluster_ids
                self.medoids = new_medoids
                return cost, self.cluster_ids, self.medoids
            else:
                self.cluster_ids = new_cluster_ids
                self.medoids = new_medoids
                iter_count += 1

            # If it reaches here, it means max_iter was reached without convergence
            cost = np.sum([np.sum(np.linalg.norm(X[self.cluster_ids == j] - self.medoids[j], axis=1)) for j in range(self.n_clusters)])
        return cost, self.cluster_ids, self.medoids

    def predict(self, X):
        return np.array([np.argmin([np.linalg.norm(x - medoid) for medoid in self.medoids]) for x in X])

```

```

In [7]: import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.cluster._kmeans")

from sklearn.cluster import AgglomerativeClustering
import skfuzzy as fuzz
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

```

In [ ]:

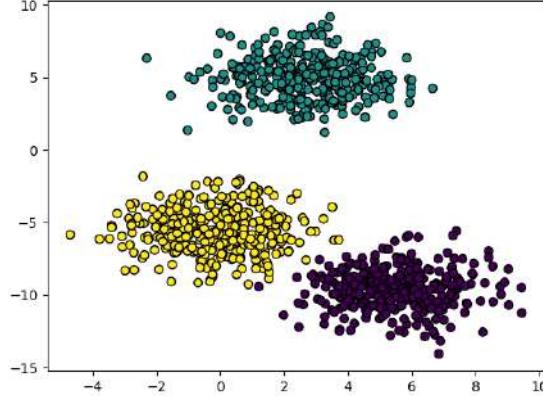
## Example1

```

In [8]: from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.5, random_state=10)

plt.scatter(X[:, 0], X[:, 1], edgecolor="k", c=y)
plt.show()

```

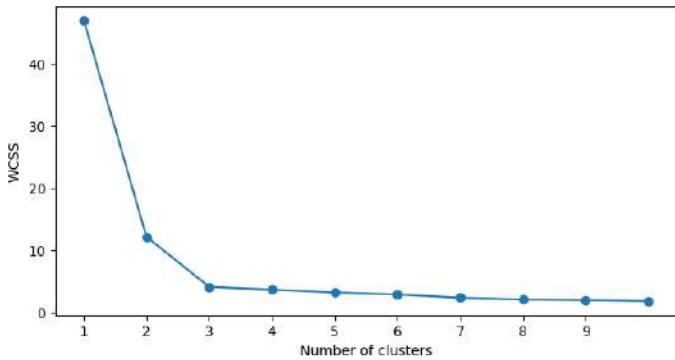


```

In [9]: costs=[]
for i in range(1, 11):
    modell = KMeansCustom(i)
    cost, _, _ = modell.fit(X)
    costs.append(cost)

plt.figure(figsize=(8, 4))
plt.plot(range(1, 11), costs, marker='o')
plt.xticks(range(1, 10))
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

```



## KMeansCustom

```
In [10]: model1 = KMeansCustom(n_clusters=3)
cost , labels , centroids= model1.fit(X)
```

```
In [11]: #external accuracy score
from sklearn.metrics import homogeneity_score
homogeneity_score(labels, y)
```

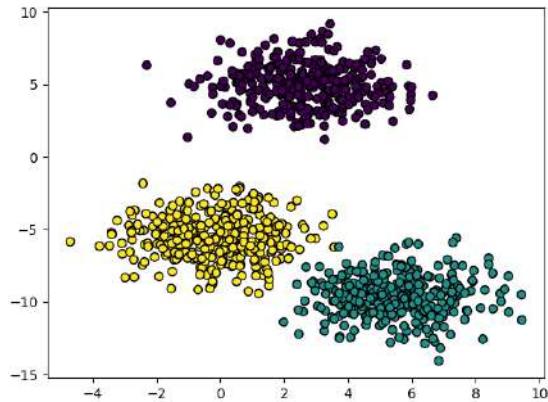
```
Out[11]: 0.9777410259568494
```

```
In [12]: #internal accuracy score
from sklearn.metrics import silhouette_score
```

```
silhouette_score(X , labels)
```

```
Out[12]: 0.6753782140623351
```

```
In [13]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels)
plt.show()
```



## kmeans++

```
In [14]: #kmeans++
model2= KMeansPlusPlus(3)
cost2 , labels2 ,centroids2= model2.fit(X)
```

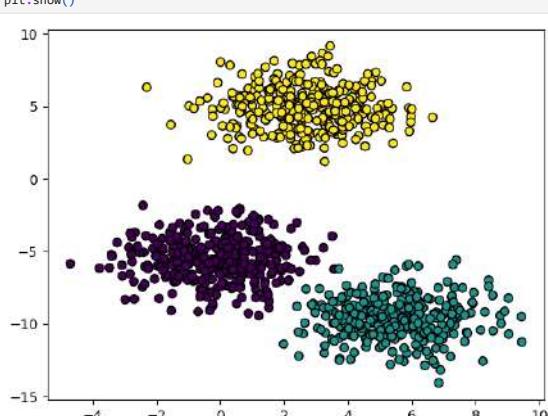
```
In [15]: #external accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(labels2 , y)
```

```
Out[15]: 0.9777410259568494
```

```
In [16]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels)
```

```
Out[16]: 0.6753782140623351
```

```
In [17]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels2)
plt.show()
```



## kmeans sklearn

```
In [18]: #kmeans sklearn(++)
from sklearn.cluster import KMeans
model3 = KMeans(n_clusters=3 , n_init='auto')
model3.fit(X)

Out[18]: KMeans
KMeans(n_clusters=3, n_init='auto')

In [19]: #cost
cost=model3.inertia_

#labels
labels3=model3.labels_

#centeroids
centeroids=model3.cluster_centers_

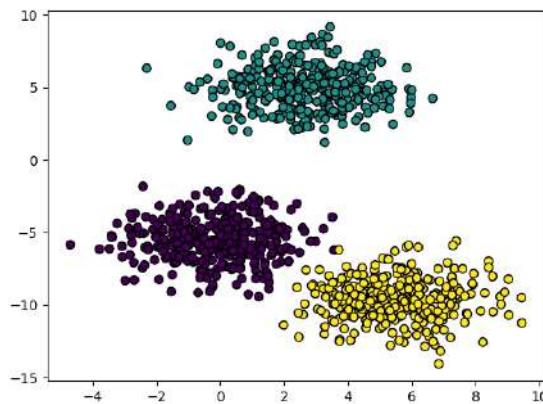
In [20]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels3)

Out[20]: 0.6753782140623351

In [21]: #external score
from sklearn.metrics import homogeneity_score
homogeneity_score(y , labels3)

Out[21]: 0.9777410259568494

In [22]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels3)
plt.show()
```



## kmmedoidCustom

```
In [23]: model4=KMedoidsCustom(n_clusters=3)
costs4 , labels4 , centeroids4 = model4.fit(X)

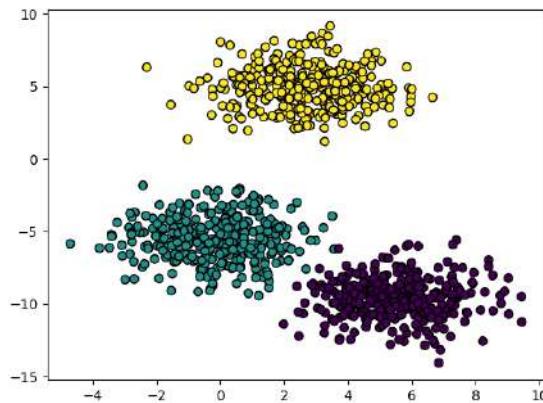
In [24]: #external accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(y , labels4)

Out[24]: 0.9826720995072343

In [25]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels4)

Out[25]: 0.6753999073686944

In [26]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels4)
plt.show()
```



## kmmedoid skextra

```
In [27]: from sklearn_extra.cluster import KMedoids

In [28]: model5= KMedoids(n_clusters=3 , method='pam')
model5.fit(X)
```

```

Out[28]: v KMedoids
KMedoids(method='pam', n_clusters=3)

In [29]: labels5 = model5.labels_

In [30]: #external accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(y , labels5)

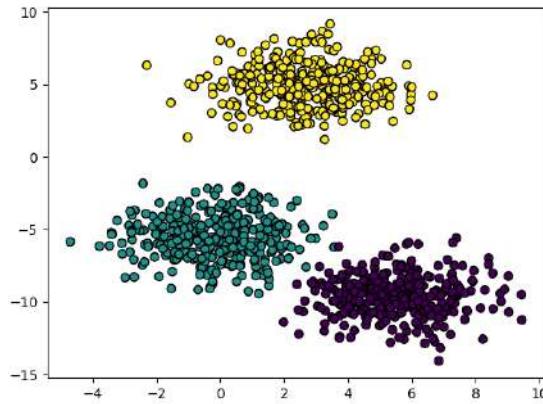
Out[30]: 0.9826720995072343

In [31]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels5)

Out[31]: 0.6753999073686944

In [32]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels5)
plt.show()

```



## hierarchical clustering

```

In [33]: from sklearn.cluster import AgglomerativeClustering

In [34]: #dendrogram
import scipy.cluster.hierarchy as sch
dendrogram=sch.dendrogram(sch.linkage(X , method="ward" , metric="euclidean"))

In [35]: model6 =AgglomerativeClustering(n_clusters=3, metric='euclidean' , linkage='ward')
model6.fit(X)

Out[35]: v AgglomerativeClustering
AgglomerativeClustering(metric='euclidean', n_clusters=3)

In [36]: labels6= model6.labels_

In [37]: #external accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(y , labels6)

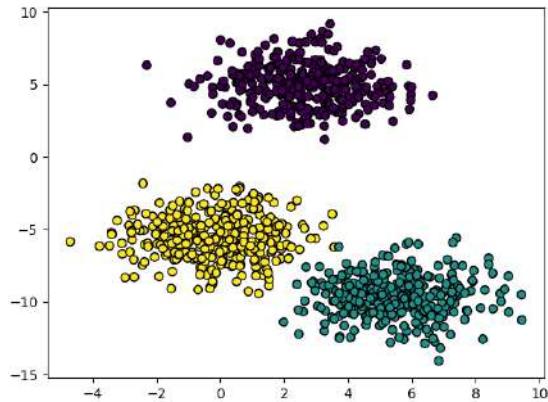
Out[37]: 0.9826720995072343

In [38]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels6)

Out[38]: 0.6753448956733746

In [39]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels6)
plt.show()

```



## fuzzy clustering

```
In [40]: import skfuzzy as fuzz
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X.T, 3, m=2, error=0.005, maxiter=1000)
labels7 = np.argmax(u, axis=0)
```

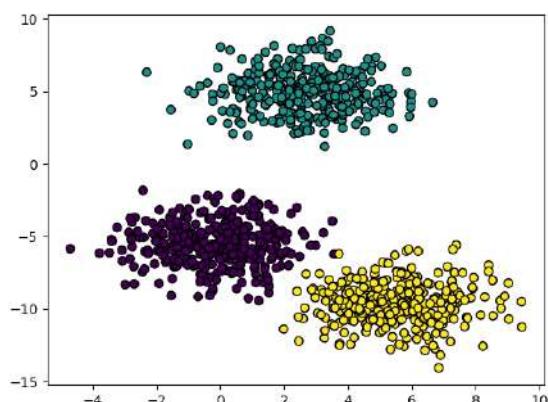
```
In [41]: #external accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(y , labels7)
```

```
Out[41]: 0.9777410259568494
```

```
In [42]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels7)
```

```
Out[42]: 0.6753782140623351
```

```
In [43]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels7)
plt.show()
```



## DBscan

```
In [59]: from sklearn.cluster import DBSCAN
model8 = DBSCAN(eps=0.5,min_samples=3 , metric='euclidean' , leaf_size=5)
model8.fit(X)
```

```
Out[59]: DBSCAN(leaf_size=5, min_samples=3)
```

```
In [60]: labels8= model8.labels_
```

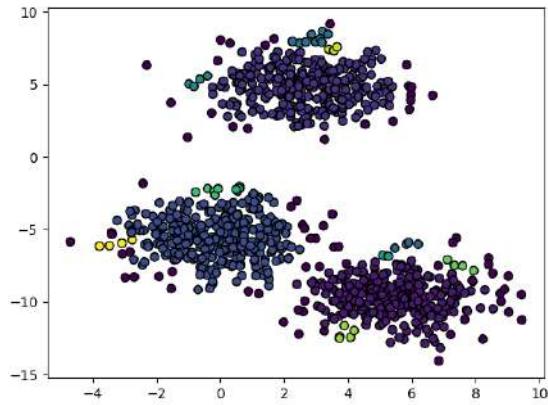
```
In [61]: #external accuracy
from sklearn.metrics import homogeneity_score
homogeneity_score(y , labels8)
```

```
Out[61]: 0.9212843281343638
```

```
In [62]: #internal score
from sklearn.metrics import silhouette_score
silhouette_score(X , labels8)
```

```
Out[62]: 0.020537564922775024
```

```
In [63]: #clustering result
plt.scatter(X[:, 0], X[:, 1], edgecolor="k" , c=labels8)
plt.show()
```

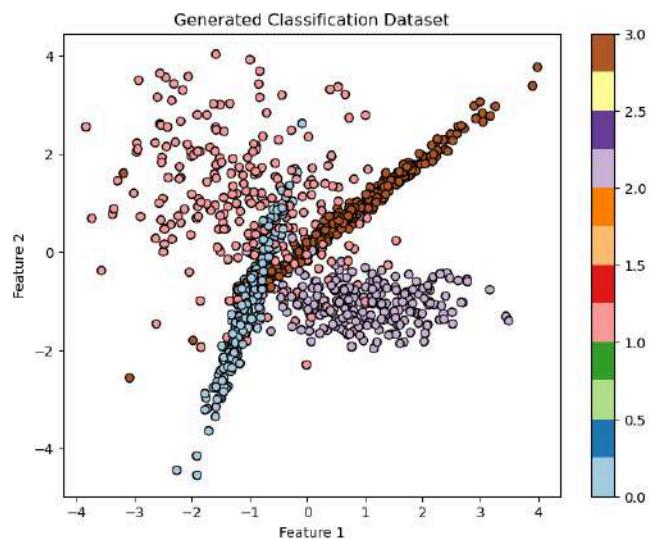


## Example 2

```
In [64]: import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

# Generate data
X, y = make_classification(n_samples=1000,
                           n_features=2,
                           n_clusters=4,
                           n_clusters_per_class=1,
                           n_informative=2,           # ALL features are informative
                           n_redundant=0,             # No redundant features
                           random_state=42)

# Plotting the dataset
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired)
plt.title('Generated Classification Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar()
plt.show()
```



```
In [68]: model1 = KMeansCustom(4)
cost1, labels1, centroids1 = model1.fit(X)
```

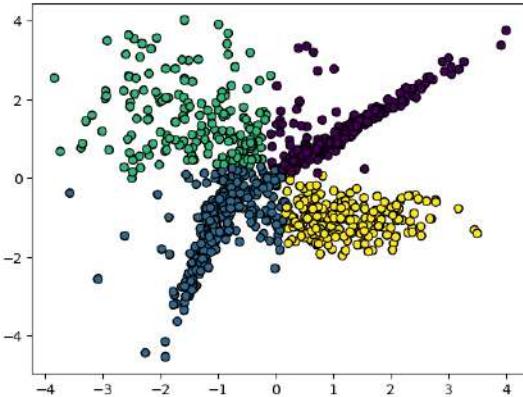
```
In [71]: #internal
from sklearn.metrics import silhouette_score
silhouette_score(X, labels1)
```

```
Out[71]: 0.4408896029342354
```

```
In [72]: #EXTERNAL
from sklearn.metrics import homogeneity_score
homogeneity_score(labels1, y)
```

```
Out[72]: 0.5404865089395575
```

```
In [77]: import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=labels1, edgecolors="k")
plt.show()
```



```
In [78]: from sklearn.cluster import KMeans  
model2 = KMeans(n_clusters=3 ,n_init="auto" )  
model2.fit(X)
```

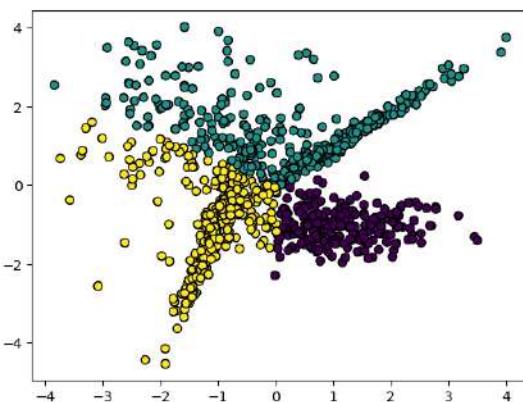
```
Out[78]: KMeans  
KMeans(n_clusters=3, n_init='auto')
```

```
In [79]: labels2= model2.labels_
```

```
In [80]: #external  
from sklearn.metrics import homogeneity_score  
homogeneity_score(labels2 , y)
```

```
Out[80]: 0.4939842723297336
```

```
In [81]: plt.scatter(X[:, 0] ,X[:, 1] , c=labels2 , edgecolors="k")  
plt.show()
```



```
In [88]: from sklearn.cluster import AgglomerativeClustering  
model3= AgglomerativeClustering(n_clusters=4 , linkage="ward")  
model3.fit(X)
```

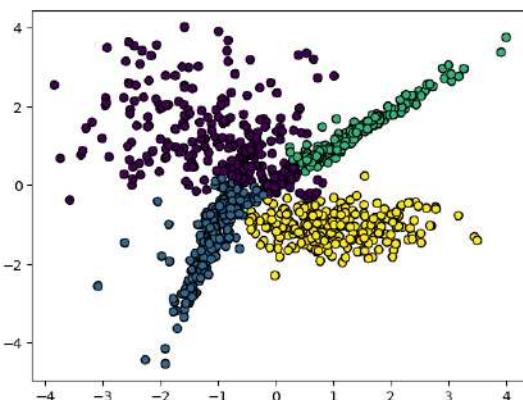
```
Out[88]: AgglomerativeClustering  
AgglomerativeClustering(n_clusters=4)
```

```
In [89]: labels3= model3.labels_
```

```
In [90]: homogeneity_score(labels3 , y)
```

```
Out[90]: 0.5670642011018312
```

```
In [91]: plt.scatter(X[:, 0] ,X[:, 1] , c=labels3 , edgecolors="k")  
plt.show()
```



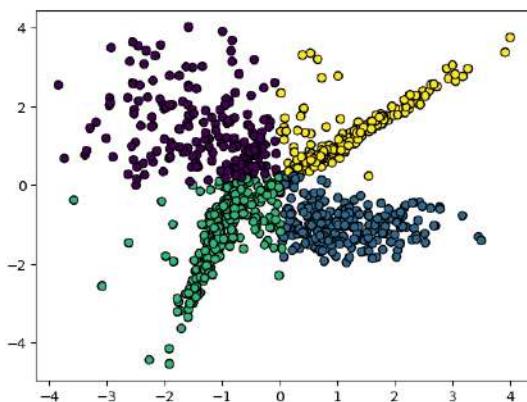
```
In [96]: import skfuzzy as fuzz  
cnt , u , _ , _ , fpc = fuzz.cluster.cmeans(X.T , 4 , m=2 , error=0.05 , maxiter=5000)  
labels4= np.argmax( u , axis=0)
```

```
In [97]: fpc
```

```

Out[97]: 0.6095276494456421
In [98]: homogeneity_score(y , labels4)
Out[98]: 0.5153687553869889
In [99]: plt.scatter(X[:, 0] ,X[:, 1] , c=labels4 , edgecolors="k")
plt.show()

```



## Example2

```

In [5]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv("D:\AI_Machinlearning\machine_learning\jozavat\kmeans\facebook.csv")
df.drop(["status_id" , "status_published"] , axis=1 , inplace=True)
df

```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	video	529	512	262	432	92	3	1	1	0
1	photo	150	0	0	150	0	0	0	0	0
2	video	227	236	57	204	21	1	1	0	0
3	photo	111	0	0	111	0	0	0	0	0
4	photo	213	0	0	204	9	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
7045	photo	89	0	0	89	0	0	0	0	0
7046	photo	16	0	0	14	1	0	1	0	0
7047	photo	2	0	0	1	1	0	0	0	0
7048	photo	351	12	22	349	2	0	0	0	0
7049	photo	17	0	0	17	0	0	0	0	0

7050 rows × 10 columns

```

In [3]: df.isnull().sum().sum()
Out[3]: 0

In [8]: #CREATE X
X=np.array(df)

In [ ]: #normalize
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X= sc.fit_transform(X)

In [ ]: costs=[]
for i in range(1,11):
    model1= KMeansCustom(i)
    cost , _, _ = model1.fit(X)
    costs.append(cost)

In [ ]: import matplotlib.pyplot as plt
plt.plot(costs , [i for i in range(len(costs))])
plt.xlabel("iteration")
plt.ylabel("costs")

plt.show()

```

```
In [ ]:
```