* Build-in functions
  + Map(function\_name , list\_name)
    - def names(text):  
       return f"- {text} -"  
        
      words = ["amira" , "aya" , "yoka" , "meroo"]  
      words\_map = map(names , words)  
      for name in words:  
       print(name)  
       # - amira -  
       # - aya -  
       # - yoka -  
       # - meroo -
  + filter()
    - def number\_filter(num):  
       if num > 10:  
       return num  
      numbers = [10 , 20, 3, 50 , 11 , 5]  
      list\_filter = filter(number\_filter , numbers)  
      for num in list\_filter:  
       print(f"{num}" , end = " ") #20 50 11
    - def number\_filter(num):  
       if num == 0:  
       return True  
      numbers = [10 , 0, 3, 0 , 11 , 5]  
      list\_filter = filter(number\_filter , numbers)  
      for num in list\_filter:  
       print(f"{num}" , end = " ") # 0 0
  + reduce()
    - from functools import reduce  
      def sumAll(num1 , num2):  
       return num1 + num2  
      list\_reduce = [1 , 5, 10 , 10 , 20, 15] #61

# (((((1+5) + 10) + 20 ) + 15)  
result = reduce(sumAll , list\_reduce)  
print(result)

* + enumerate(iterator , start)
    - mySkills = ["Html" , "Css" , "JS" , "C" , "C++"]  
      mySkillsCounter = enumerate(mySkills , 10)  
      for counter in mySkillsCounter:  
       print(counter)  
      # (10, 'Html')  
      # (11, 'Css')  
      # (12, 'JS')  
      # (13, 'C')  
      # (14, 'C++')
  + Print()
  + Reversed()
  + Id(variable) => place of variable in memory
  + Type()
  + Help(“keywords”)
  + Reshape(metrics , (height , width)
  + Dir(module\_name)
  + All(list\_name)
  + Any(list\_name)
  + Bin(number) => convert decimal to binary
  + Sum(alterable , start)
  + Round(number , num\_of\_digits)
  + Range(start , end , step)
    - Range(10) => from 0 to 10
  + Sep = “ ”
    - print("Hello" , "meroo" , "how" , "old" , "are you" , sep = " & ")  
      # Hello & meroo & how & old & are you
  + end = “\n”
    - print("amira" , end = " ")  
      print("hassan") #amira hassan  
      print("sobhi")#sobhi
  + abs()
  + pow(base , exp , mod)
  + min()
* Variables
  + X -> without datatype
  + Global x
    - x = 5
    - def num():
    - global x
    - x = 14
    - num()
    - print(x) #14
  + Var\_name.isidentifier() #check on variable name if it valid or no
    - a = "aa--nn"  
      print(a.isidentifier()) # false
  + Var\_name.isalpha()
    - a = "21432ddvdx"  
      print(a.isalpha()) # false
  + Var\_name.isalnum()
    - a = "fdg5546cs\*&cdsvv"  
      print(a.isalnum()) # false
* Data types
  + Complex
    - complexNumber = 3 + 5j  
      print(type(complexNumber)) #<class 'complex'>  
      print("realNumber {}" . format(complexNumber.real)) #realNumber 3.0  
      print("imagNumber {}" . format(complexNumber.imag)) #imagNumber 5.0
  + Numbers
    - Int
    - Float
    - complex
  + Strings
    - “”” -> to print more than one line
    - Methods
      * Len(str)
      * Str.upper()
      * Str.lower()
      * Str.islower()
      * Str.isspace()
      * Str.expandtabs(2)
      * Str.replace(“old\_word” , “new\_word” , limit if you want)
      * Split Text
        + str = "i love my name"  
          print(str.split()) # ['i', 'love', 'my', 'name']  
          print(str.split( " ", 2)) # ['i', 'love', 'my name']  
          print(str.rsplit(" " , 2)) # ['i love', 'my', 'name']
        + str = """first  
          second third  
          """  
          print(str.splitlines()) # ['first', 'second third']
      * put the text in the middle
        + str = "amira"  
          print(str.center(10)) # amira   
          print(str.center(10,"#")) # ##amira###
      * Str.count(“word that you wanna search it” , start , end)
      * Str.swapcase() #replce lowercase with uppercase and vice versa
      * Search
        + Str.startwith(“u”,start , end)
        + Str.endwith(“u”,start , end)
        + Str.find(“u” , start , end) # if u not found it will return -1
        + Str.index(“u” , start , end) # if u not found it will return error
      * Put spaces || characters
        + str = "Meroo"  
          print(str.rjust(10 , "\*")) #\*\*\*\*\*Meroo  
          print(str.ljust(10 , "\*")) #Meroo\*\*\*\*\*
      * Remove spaces
        + str = " mer ooo "
        + Str.strip() # mer ooo.
        + Str.lstrip() # mer ooo .
        + Str.rstrip() # mer ooo.
      * Remove characters
        + str = "\*\*\*mer\*\*ooo\*\*\*"
        + Str.strip(“\*”) #mer\*\*ooo
        + Str.lstrip(“\*”) #mer\*\*ooo\*\*\*
        + Str.rstrip(“\*”) #\*\*\*mer\*\*ooo
      * Str.title()
      * Str.istitle()
        + str = "info 3d cycle remove 2am"
        + # Info 3D Cycle Remove 2Am
      * Str.capitalize()
        + # Info 3d cycle remove 2am
      * Str.zfill(length of num)
        + a , b , c, d = "1" , "12" , "123" , "1234"  
          print(a.zfill(3)) #001  
          print(b.zfill(3)) #012  
          print(c.zfill(3)) #123  
          print(d.zfill(3)) #1234
  + Boolean
    - Methods
      * Bool(value)
  + Lists (mutable)
    - Writing method
      * numbers = [3 , 5, 9,7]
        + print(numbers[1:3]) [5,9]
        + print(numbers[1]) 5
        + print(numbers[-1]) 7
        + print(numbers[-2::]) [9,7]
        + print(numbers[::3]) [3,7]
      * numbers = [3 , 5, [1 , 2], 7 , 8]
    - functions
      * .append(value) to add a value to the end of the list
      * .insert(index,value you want it)
      * .extend([value,value,…]) to add more than one value
      * Sum(list\_name)
      * L1 = sorted(list\_name)
      * .sort()
      * .reverse()
      * ‘’.join(list\_name) to concatenate the list
      * .remove(item)
      * Del list\_name[1]
      * .pop() to return last element and use it if I want
      * .pop(2) 2 -> the index that I want remove & return it
      * .clear()
      * .copy()
      * .count(item)
      * .index(item)
  + Tuple (immutable)
    - Tuples = (2 , ‘c’, 7,6)
    - To convert tuple to list
      * Var = list(tubles\_name)
    - Concatenate
      * +
    - tu = ("one" , "two" , 5 , "three")  
      a , b , \_ , c = tu  
      print(a) #one  
      print(b) #two  
      print(c) #three
  + Set (is an unordered collection of items and unindexed)
    - Writing methods
      * Myset = {10 , 20 , 30 , 20 , 10}
    - Function
      * Myset.union(anotherSet , set , …) = myset | anotherSet
      * Myset.add(value)
      * .discard(value you wonna remove it without error)
      * Myset.update(Myset1) |=
        + بيدمجهم مع بعض بس مش بياخد المتكرر
      * Myset.difference(anotherSet) = myset – anotherset
        + اللى موجود فى الاولى ومش موجود فى التانيه
      * Myset.symmetric\_difference (Myset1) ^
        + اللى مش موجوده فى الاتنين
      * Myset.intersection (Myset1) &
      * Myset.intersection\_update(Myset1) &=
      * Myset.difference\_update(Myset1) -=
        + هيعمل لناتج الديفيرينس ابديت ويحطها فى الاولى
      * Myset.symmetric\_difference\_update(Myset1) ^=
      * Set1.issuperset(set2)
        + set2 part of set1
      * set1.isupset(set2)
        + set1 part of set2
      * set1.disjoint(set2) (منفصلينْ)
  + Dictionaries
    - Writing methods
      * grades = {
      * "arabic" : 50,
      * "English" : 70,
      * "german" : 90
      * }
      * Loop
        + for k in grades:
        + for x,y in dept.items():
      * Print(‘german’ in grades)
      * Print(grades.keys())
      * Print(grades.values())
      * Print(grades.items())
    - functions
      * Grades[‘islamic’] = 80
      * Del grades[‘English’]
      * Del grades
      * Grades1 = grades2.copy()
      * Grades.pop[key]
      * Dic.popitem()
      * Grades.get(‘math’ , ‘not exist’) return the value and if not exist return another msg
      * Grades1.update(grades2) grades1 + grades2
      * Dic.keys()
      * Dic.values()
      * Dic.items()
      * Dic.setdefault(key , value)
      * a = ("one" , "two", "three")  
        b = "value"  
        print(dict.fromkeys(a , b)) #{'one': 'value', 'two': 'value', 'three': 'value'}
  + map
* operations
  + Arithmetic
    - =, +, -, \*, \*\*(exponent), /, //(floor divide), %
  + Comparison
    - ==, !=, >, <, >=, <=
  + Logical
    - And, or, not
  + Bitwise
    - &, |, ^, ~, <<, >>
    - Ex
      * 10 -> 8 + 2
      * 75 -> 64 + 8 + 2 + 1
      * 10 & 75 = 10(8 + 2). بياخد المشترك
      * 10 | 75 = 75(64 + 8 + 2 + 1). كائنى بعمل اتحاد
      * 10 ^ 75 = 65(64 + 1)هياخد اللى مش مشترك
      * 10 << 3 = 80 (01010 000) بياخد 3 اصفار من اليمين ويحطهم فى الشمال
      * 10(1010) >> 3 = 1
        + 0000 1010
        + 0100 0001
      * 75 >> 3 = 9
        + 0100 1011
        + 0111 1001
  + Assignment
  + Membership
    - In, not in
      * If x in numbers:
      * If x not in numbers:
  + Identity
* Conditions
  + If
  + If … else
  + Nested if
    - If … elif … elif … else
  + Turnary
    - age = 18  
      age\_user = int(input("enter your age : "))  
      print("Done, your age is valid " if age\_user > age else "Sorry, your age is not valid")
* Loops
  + For
    - For I in “Egypt”:
    - For I,x in enumerate(“Egypt”)
    - For in range(start, condition, step)
      * For I in range(2,7,1)
    - format
      * num = 7
      * for i in range(1,11):
      * print("{}\*{}={}" . format (i,num , i\*num))
  + While
* Function
  + Func with args
    - Def function\_name (a,b): return ()
  + Fun with list
    - def sum(\*nums): #tuble
    - result = 0
    - count = 0
    - for i in nums:
    - result = result + i
    - count = count + 1
    - return (result , count)
    - result\_sum = sum(2 , 4, 6, 3) #list
    - print(result\_sum[0])
    - print(result\_sum[1])
  + func with dictionary (kwargs)
    - def grades(\*\*skills):  
       print(type(skills)) #dict  
       for language , precentage in skills.items():  
       print(f"{language} => {precentage}")  
        
      # grades(html = "50%" , css = "60%" , js = "90%")  
      languages = {  
       "html" :"50%",  
       "css" : "60%",  
       "js" : "90%"  
      }  
      grades(\*\*languages)
* Lambda
  + hello = lambda name , age : f"the name is {name} & the age is {age}"  
    print(hello("meroo" , 20))
* Recursive
  + def main\_word(word):  
     if len(word) == 1:  
     return word  
     if word[0] == word[1]:  
     return main\_word(word[1:])  
     return word[0] + main\_word(word[1:])  
      
    print(main\_word("wwworrlddd"))
* Module
  + Random
    - import random  
      print(random.randint(10 , 50))
    - import random as rn  
      print(rn.random()) # from 0 to 1  
      print(rn.randint(10 , 100)) # integer number from 10 to 100
    - a = random.randint(2 ,10, size = 6) #[4 9 6 8 2 6]
    - print(rn.uniform(10 , 100)) # decimal number from 10 to 100
    - a = random.uniform(2 ,10 , 5) # 5.24139675 5.84049289 3.83104411 8.26042491 5.57731417]
    - ]
    - print(rn.randrange(100)) # integer number from 0 to 100
    - print(rn.randrange(0,100,2)) # integer number from 0 to 100 (odd)
    - print(rn.choice([4,6,7])) # choice random number
    - print(rn.sample(range(200) , 10) # choice 10 nums from 0 to 199
    - print(rn.shuffle(a) # items in list a will be shuffle
    - print(rn.uniform(2 , 10)) #3.5383602557941733
    - a = random.random((2 , 10))
    - print(a) #[[0.64261358 0.55698388 0.59304565 0.16059278 0.36594248 0.76777121
    - 0.04232465 0.81228508 0.15808882 0.4843274
    - [0.59338917 0.20505942 0.58565385 0.9934994 0.99986773 0.96498948
    - 0.17749675 0.58416055 0.27800513 0.80648592]]
  + import sys
    - import sys  
      print(sys.path)  
      sys.path.append(r"D:\Games")  
      print(sys.path)
  + alias
    - import test as tt
* extern Package => pip
  + in cmd
    - pip –version
    - pip list
    - to install package
      * pip install termcolor
      * pip install pyfiglet
        + termcolor & pyfiglet => are names of packages
    - To upgrade the library
      * pip install pip –upgrade
  + import
    - import pyfiglet
      * pyfiglet.figlet\_format(“meroo”)
    - import datetime
      * print the current date & time
        + datetime.datetime.now()
        + datetime.datetime.now().year
        + datetime.datetime.now().month
        + datetime.datetime.now().day
        + datetime.datetime.now().time()

datetime.datetime.now().time().hour

datetime.datetime.now().time().minute

datetime.datetime.now().time().second

datetime.datetime.now().time().microsecond

* + - * + datetime.datetime.min
        + datetime.datetime.max
        + datetime.datetime(year , month , day , hour , minute , second)
      * formate the date
        + print(my\_date.strftime("%a - %b - %y")) #Sun - Jul - 03  
          print(my\_date.strftime("%A - %B - %Y")) #Sunday - July - 2003
    - import PIL
      * from PIL import Image
        + from PIL import Image  
          myImage = Image.open("C:\learnPython\game.jpg")  
          myImage.show()  
          box = (0,0,500,800)  
          crop\_image = myImage.crop(box)  
          crop\_image.show()
        + .crop(left , top , width , height)
    - Import pylint => to improve the code
      * In terminal
        + Pylint.exe file\_path
* Exception
  + Raise exception(“”)
  + Try: except: else: finally:
  + Type of errors
    - ZeroDivisionError
    - NameError
    - ValueError
  + Ex
    - Try:
      * Print(10 / 0)
      * Print(x)
      * Int(“Hello”)
    - Except ZeroDivisionError:
      * Print(“you can not divide by 0”)
    - Except NameError:
      * Print(“the value is not exist”)
    - Except ValueError:
      * Print(“you can not convert word hello to integer”)
    - Except:
      * Print(“exist error here”)
* Regural
  + Website
    - Pythex
    - Regx101.com
  + A screenshot of a computer

    Description automatically generated
* Doc
  + def doc\_func(name):  
     *"""  
     info about the function  
     return name of person  
     to explain documentation  
     """* return name  
    print(doc\_func.\_\_doc\_\_)  
    # info about the function  
    # return name of person  
    # to explain documentation
* Iterable & iterator
  + Iterable => string, list , … except Number(Int & float)
  + Iter(iterable) = iterator
  + Next(iterator)
  + myString = "Meroo"   
    myIterator = iter(myString)  
    print(next(myIterator)) #M  
    print(next(myIterator)) #e  
    print(next(myIterator)) #r
* generator
  + def myGenerator():  
     yield 1  
     yield 2  
     yield 3  
     yield 4  
     yield 5  
      
    print(next(myGenerator())) #1  
    print(next(myGenerator())) #1  
    print("#" \* 10) ##########  
    gen = myGenerator()  
    print(next(gen)) #1  
    print(next(gen)) #2  
    print("hello") #hello  
    print(next(gen)) #3  
    for g in gen:  
     print(g, end=" ") # 4 5
* Decorator
  + No parameter
  + def myDecorator(func):  
     def nestedFunc():  
     print("before")  
     func()  
     print("after")  
     return nestedFunc  
    @myDecorator  
    def sayHello():  
     print("Hello")  
      
    sayHello()
  + with parameter
  + def myDecorator(func):  
     def nestedFunc(num1 , num2):  
     print("before")  
     func(num1 , num2)  
     print("after")  
     return nestedFunc  
      
    @myDecorator  
    def sum(num1 , num2):  
     print(num1 + num2)  
      
    sum(2 , 5)
* Zip
  + list = [1 , 2, 3, 4, 5]  
    list1 = ["A" , "B" , "C"]  
    print(zip(list , list1)) #<zip object at 0x000001DEC5A47E40>  
    for item in zip(list , list1):  
     print(item)  
    # (1, 'A')  
    # (2, 'B')  
    # (3, 'C')
* file
  + mode
    - r -> read
    - a -> append
    - w -> rite
    - x -> create
  + paths
    - to know directory name
      * import os  
        print(os.path.dirname(os.path.abspath(\_\_file\_\_))) #C:\learnPython\learn.py
    - to remove directory
      * import os  
        os.remove("moraa.txt")
    - absolute path => Starting is from root
      * if you wanna know absolute path
        + import os  
          print(os.path.abspath(\_\_file\_\_)) # C:\learnPython\learn.py\main.py
    - relative path
      * to use it => you should know the Current Working Directory
        + import os  
          print(os.getcwd()) #C:\learnPython\learn.py
  + function
    - write
      * f = open('Desktop/python.txt' , 'w')
      * f.write("hello, my name is merOoo")
      * f.close()
      * f.writelines(list\_name)
    - read
      * f = open('Desktop/python.txt' , 'r')
      * f.read()
      * f.readline()
      * f.readlines()
      * f.seek(5) => skept 5 characters then print
        + myFile = open("moraa.txt" , "r")  
          myFile.seek(5)  
          print(myFile.readline())
    - append
      * f = open('Desktop/python.txt' , 'a')
      * f.write(' and my age is 20 years')
      * f.writelines(myList)
      * f.truncate(5) => cut the text
        + myFile = open("moraa.txt" , "a")  
          print(myFile.truncate(5)) #5  
          print(open("moraa.txt" , "r").readline())
      * f.tell() => place of curser
        + calc the new line with 2byte => (\r\n)
        + Return number of characters
      * f.close()
  + build-in-functions
    - name & mode & encoding
      * myFile = open("moraa.txt" , "r")  
        print(myFile)   
        #C:\learnPython\learn.py\main.py <\_io.TextIOWrapper name='moraa.txt' mode='r' encoding='cp1252'>  
        print(myFile.name) #moraa.txt  
        print(myFile.mode) #r  
        print(myFile.encoding) #cp1252
* Statistics
  + Import statistics as st
  + A = st.Mean([2 , 6 , …])
  + A = St.harmonic\_mean([2 6 , …])
  + A = st.median([2,6,…])
  + A = st.media\_low([4,5,6,7]) #5
  + A = st.media\_high([4,5,6,7]) #5
  + A = st.mode([2,3,2]) #2
  + Standard deviation
    - A = st.stdev([3.2,6,…])
  + A = st.variance([6,7,…])
* Numpy
  + import numpy as np
    - [.ipynb\_checkpoints\numpy-checkpoint.ipynb](.ipynb_checkpoints/numpy-checkpoint.ipynb)
    - SIN
      * import math as ma
      * a = ma.sin(30 \* (np.pi/180)) #or
        + a = ma.sin(np.deg2rad(30)) #or
        + a = ma.sin(ma.radians(30))
      * print(a) #0.49999999999999994
    - print(np.mod(21 , 5)) #1
    - METRICS
      * Array & shape
        + a = np.array( [[1,2,3],[4,5,6]] )
        + print(a)

[ [1 2 3]

[4 5 6] ]

* + - * + Print(a.shape)

(2 , 3)

* + - * With range
        + a = np.array([range(i , i+4) for i in [2,4,6]])
        + print(a)
        + [[2 3 4 5]
        + [4 5 6 7]
        + [6 7 8 9]]
      * Empty
        + A = empty((2,4))

[[0.00e+000 0.00e+000 0.00e+000 0.00e+000]

[0.00e+000 7.35e-321 0.00e+000 0.00e+000]]

* + - * Zeros, Ones, Identical, diagonal & Full
        + A = ones((4 , 6))
        + A = zeros((2 , 5))
        + A = eye(6)
        + a = diag(array([1 ,3 , 5, 7, 9]))

[[1 0 0 0 0]

[0 3 0 0 0]

[0 0 5 0 0]

[0 0 0 7 0]

[0 0 0 0 9]]

* + - * + a = diag(array([1 ,3 , 5, 7, 9]) , k = 2)

[[0 0 1 0 0 0 0]

[0 0 0 3 0 0 0]

[0 0 0 0 5 0 0]

[0 0 0 0 0 7 0]

[0 0 0 0 0 0 9]

[0 0 0 0 0 0 0]

[0 0 0 0 0 0 0]]

NOTE -> it will increase by 2 cols & 2 rows

* + - * + A = full((3 , 5) , 20)
        + Arrange & reshape
        + A = arange(10)

[0 1 2 3 4 5 6 7 8 9]

* + - * + A = arrange(10 , 20)

[10 11 12 13 14 15 16 17 18 19]

* + - * + A = arrange(10 , 20 , 2)

[10 12 14 16 18]

* + - * + a = arange(5 , 25).reshape(5, 4)

[[ 5 6 7 8]

[ 9 10 11 12]

[13 14 15 16]

[17 18 19 20]

[21 22 23 24]]

NOTE -> from 5 to 25 is 20 numbers so that we need matricx is rows \* cols = 20

* + - * + A = arange(27).reshape(3,3,3)

[[[ 0 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]]]

* + - * + c = linspace(0 , 100 , 5)

[0. 25. 50. 75. 100.]

* pandas
  + <Pandas.ipynb>
  + Read\_csv(‘D:\\2.csv’ , index\_col = ‘column\_name’)
  + Head(5)
    - Appear first 5 rows
  + To Write
    - To.csv(‘D:\\2.csv’)
    - To\_excel(‘D:\\1.xls’)
  + To read
    - Read\_excel(‘D:\\1.xls’)
    - Read\_csv(‘D:\\2.csv’)
  + Set\_option(‘display.width’ , 100) #width => 1000
  + Set\_option(‘precision’ , 3) #12.3456545 => 12.345
  + Series
    - Values
    - Index
    - Keys
    - Descripe()
  + agg
  + Index
  + Plot()
    - Kind
      * Line
      * Pie
      * Bar
      * Barh
      * Hist
      * Box
      * Kde
      * Area
  + DataFrame(array\_name , index= , columns= )
    - T
    - Keys()
    - Values
    - Stack()
    - Unstack()
    - Iloc[ , ] #index
    - Loc[ , ] #keys
    - Sort\_values([‘’] , ascending = )
    - Max()
    - Min()
    - Mean()
    - Std()
    - Sum()
    - Prod()
    - Mean()
      * Df1[‘key\_name’].mean()
      * Mean(axis = ‘columns’)
    - Descripe()
    - Corr() #correlation coefficient
    - Skew()
    - Query()
    - Merge(df1 , df2 , left\_on = “” , right\_on = “”).drop(“” , axis = )
    - Merge(df1 , df2 , how = “right | outer | left”)
    - Set\_index(‘’)
    - Groupby(‘key’).sum()
    - Aggregate()
    - Transform()
    - MultiIndex()
      * From\_tuples()
* OOP
  + Magic Methods
    - Class
      * Object\_name = class\_name()
      * Object\_name.\_\_class\_\_ => return class\_name
    - class learn:  
       def \_\_init\_\_(self):  
       self.list = ["css" , "html" , "js" , "react"]  
       def \_\_str\_\_(self):  
       return f"{self.list}"  
       def \_\_len\_\_(self):  
       return len(self.list)  
      test = learn()  
      print(test) #['css', 'html', 'js', 'react']  
      print(len(test)) #4
    - @classmethod & @staticmethod
      * class Member:  
         counter = 0  
         def \_\_init\_\_(self , first\_name , last\_name):  
         self.fname = first\_name  
         self.lname = last\_name  
         def info\_person(self):  
         return f"Hello {self.fname} {self.lname}"  
         @classmethod  
         def number\_users(cls):  
         cls.counter += 1  
         @staticmethod  
         def number\_users\_with\_static():  
         Member.counter +=1  
          
        person1 = Member("amira" , "hassan")  
        print(person1.info\_person())  
        print(Member.number\_users())  
        print(person1.counter)  
        print(person1.number\_users())
  + Inheritance
    - class Food:  
       def \_\_init\_\_(self , name):  
       self.fname = name  
       print(f"{self.fname} is from Parent Class")  
        
      class Apple(Food):  
       def \_\_init\_\_(self , name):  
       super().\_\_init\_\_(name)  
       print(f"{self.fname} is from derived class")  
        
      a1 = Apple("Yellow")  
      print(a1.fname)  
        
      # Yellow is from Parent Class  
      # Yellow is from derived class  
      # Yellow
* Projects
  + Calc average of subjects
    - math = int(input ("math : "))
    - science = int(input("science : "))
    - computer = int(input("computer : "))
    - total = math + science + computer
    - avg = total / 3
    - print(avg)
* Escape Sequences Characters
  + \b => Back Space
  + \n => new line
  + \r => carriage return
  + \t => Horizontal Tab
  + \xhh => Character Hex Value
    - Print(“\x4F\x73”) # Os
* Concatenate
  + +
* Notes
  + Iterable
    - String , list , …. Except Integer & Float
  + The variables will have the same id if they have the same values -> because they have the same reference
    - x = 5  
      y = 5  
      print('x : ',id(x)) #x : 140732599473064  
      print('y : ' , id(y)) #y : 140732599473064  
        
      y = 8  
      print('\nx : ',id(x)) #x : 140732599473064  
      print('y : ' , id(y)) #y : 140722110391304  
        
      print('\nx : ',x) #5  
      print('y : ' , y) #8
* Run Time Error
  + - x = 5  
      def ff(d):  
       x = x + 1  
       print(x)  
      ff(x)
* Tuples
  + - t = ('c',)  
      s = ('c')  
        
      print('t : ',type(t)) #t : <class 'tuple'>  
      print('s : ',type(s)) #s : <class 'str'>

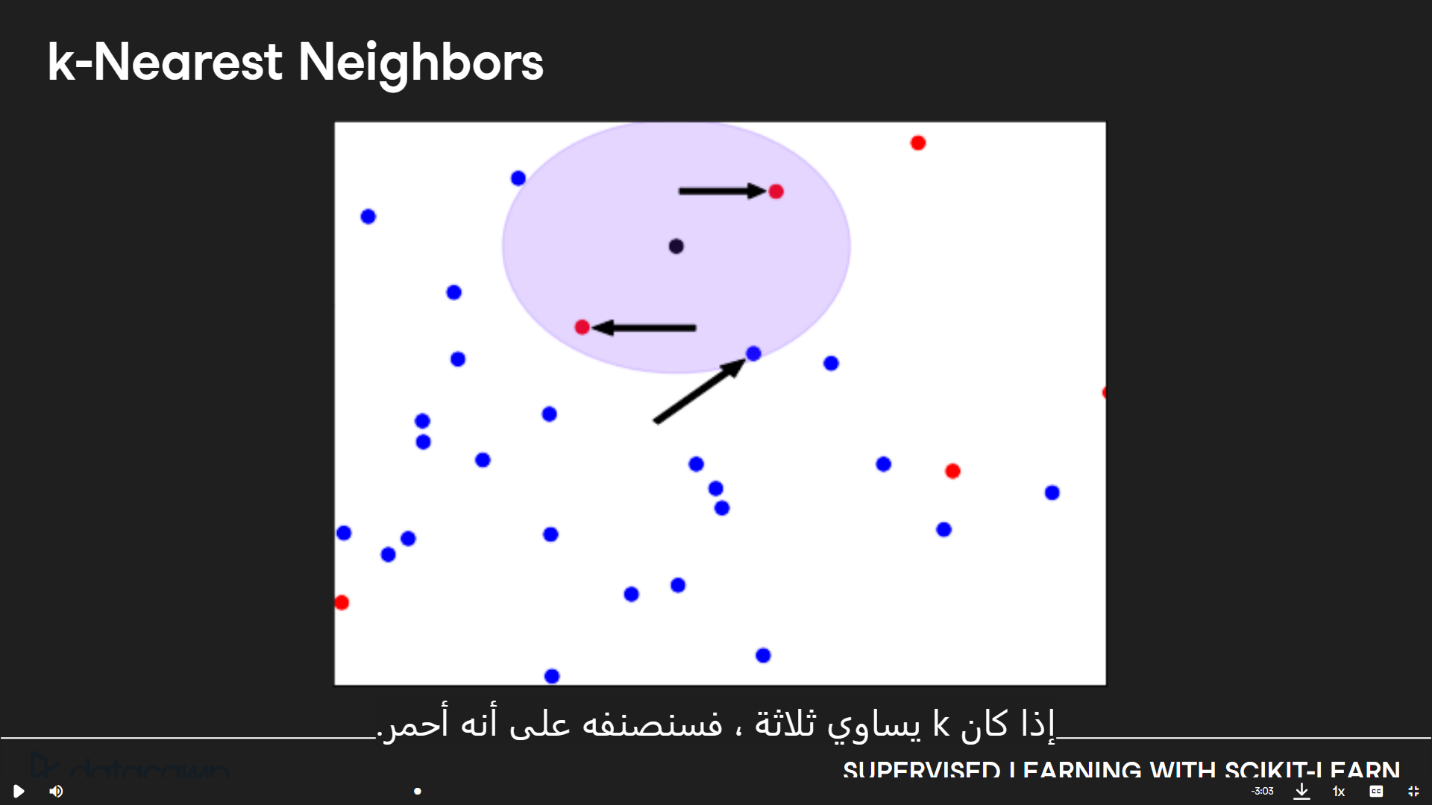
* + i = 5  
    t = 5,  
      
    print('t : ',type(i)) #t : <class 'int'>  
    print('s : ', type(t)) #s : <class 'tuple'>
* swap
  + x = 4  
    y = 6  
    (x,y) = (y,x)  
    print('x : ' , x) # 6  
    print('y : ' , y) # 4
* Alias
  + l = [1 , 3, 4]  
    l1 = [2 , 6 , 7]  
    l = l1  
    l[0] = 1000  
    print(l1) # [1000, 6, 7]
* Cloning a List
  + l = [1 , 3, 4]  
    l1 = [2 , 6 , 7]  
    l = l1[:]  
    l[0] = 1000  
    print(l1) # [2, 6, 7]
* remove\_dups
  + def remove\_dups(l1 , l2):  
     l1\_copy = l1[:]  
     for i in l1\_copy:  
     if i in l2:  
     l1.remove(i)  
      
    l1 = [1 , 2, 3, 5]  
    l2 = [2 , 3]  
    remove\_dups(l1 , l2)  
    print(l1)
* all data in python is Object
* a, b, c = 7 , 3 , 5
* to concatenate number with string
  + age = 20  
    name = "meroo"  
    print("my name is %s and my age is %d" %(name , age))  
    print("my name is {} and my age is {}".format(name , age))
  + num = 35.5720  
    name = "meroo hassan"  
    print("my name is {:.5s} and the num is {:.2f}".format(name , num))
  + num = 243476985758790  
    print("the num is {:\_d}".format(num)) #the num is 243\_476\_985\_758\_790
  + a , b , c = "one" , "two" , "three"  
    print("hello {1} {0} {2}".format(a , b , c)) #hello two one three

< Questions >

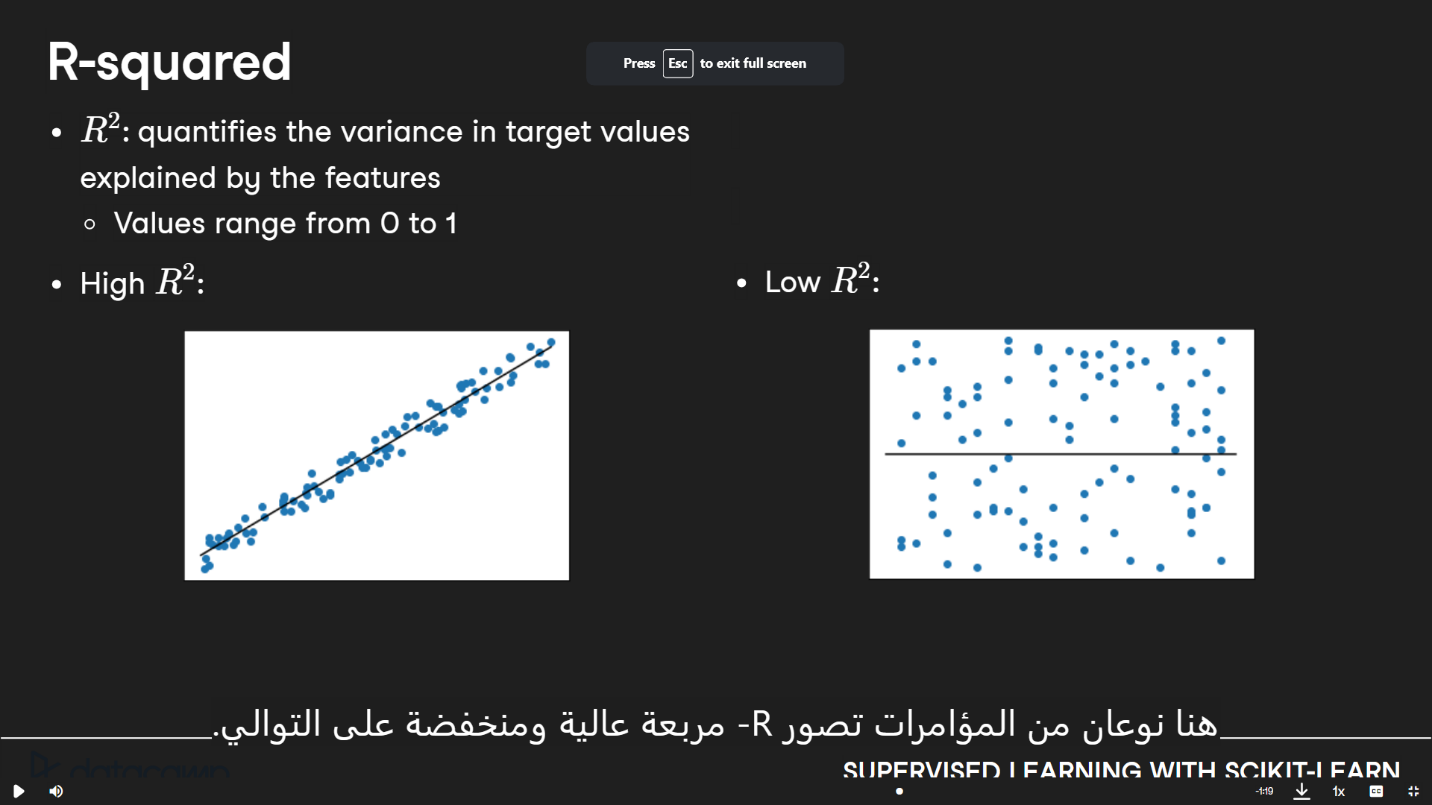
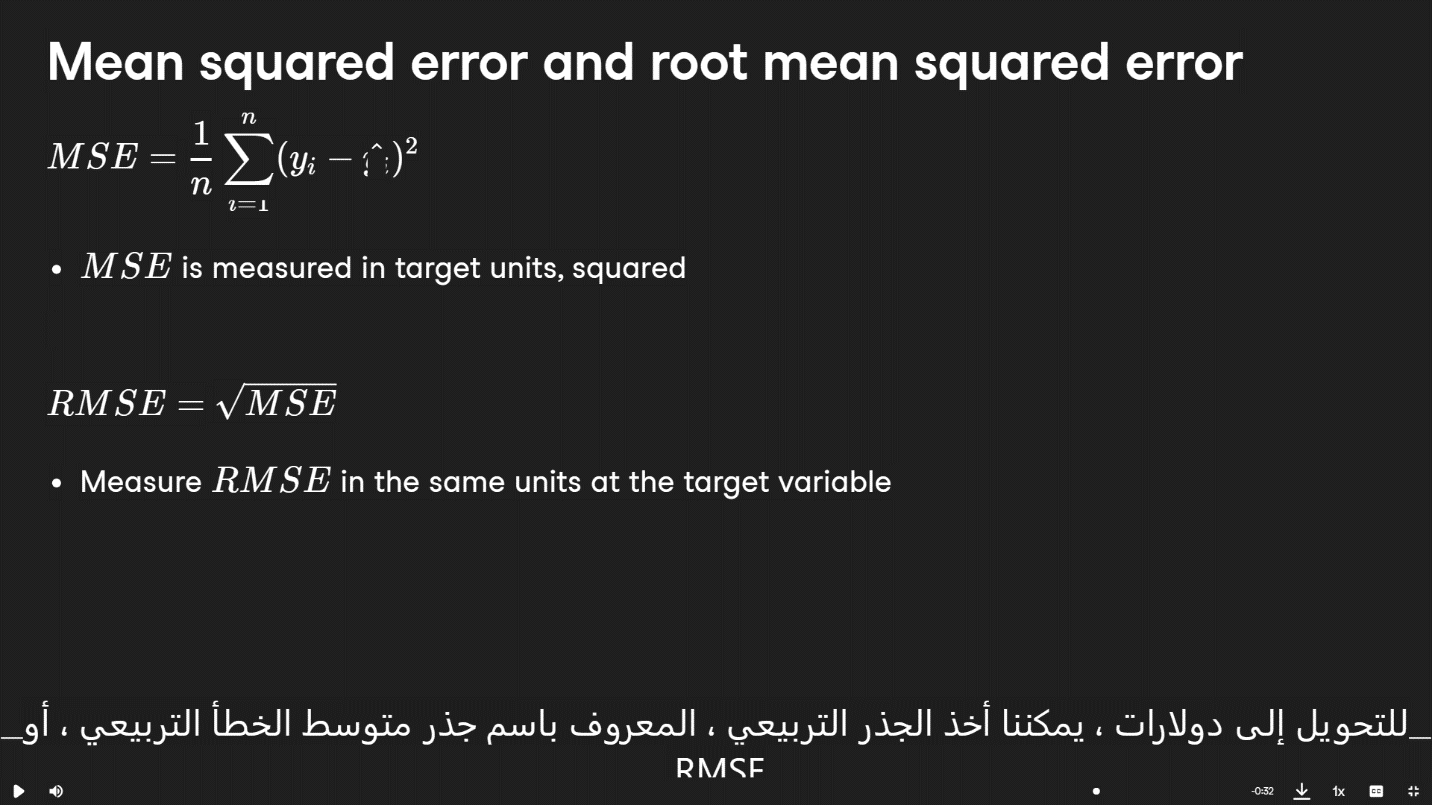
* HackerRank
* Is\_leapYear
  + def is\_leap(year):  
     leap = False  
     if year%4==0 and year%400==0 and year%100!=0:  
     leap = True  
     return leap  
      
    year = int(input())  
    print(is\_leap(year))
* print\_numbers\_without\_space
  + def loop(n):  
     str\_num = ""  
     for i in range(1, n + 1):  
     str\_num = str\_num + str(i)  
     return str\_num  
      
    n = int(input())  
    print(loop(n))

***scikit-learn syntax***

Course #1 (Supervised with scikit-learn)

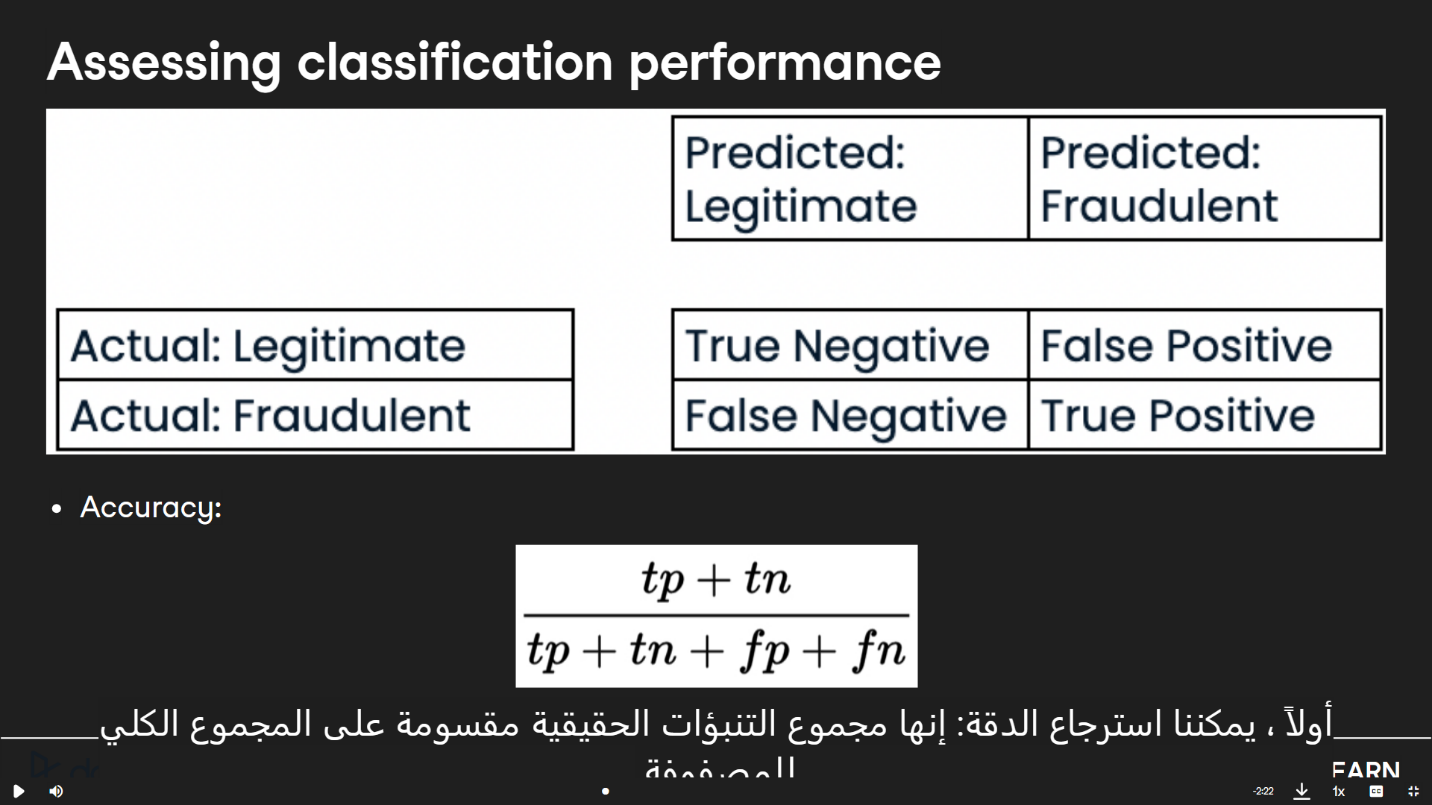
1. Classification
2. Requirements
   * No missing values
   * Data in numeric format
   * Data stored in pandas DataFrame or Numpy array
3. Syntax
   * From sklearn.module import Model
   * model = Model() #instance
   * model.fit(X,y) #
   * predictions = model.predict(X\_new)
     + علشان اعرف اصنف كل عنصر يعنى لو دخلتله داتا وعايزه اعرف هى اسبام ولا لا فهيطلعلى ارااى عبارة عن صفر وواحد
4. Classification Labels of unseen data
   * Build a model
   * Model learns from the labeled data we pass to it
   * Pass unlabeled data to the model as input
   * Model predicts the labels of the unseen data
5. K-Nearest Neighbors -> KNN (first algo)
   * Predicting the label of any data point by looking at k
   * 
   * Syntax
     + From sklearn.neighbors import KNeighborsClassifier
     + X = churn\_df[[“total\_day\_charge” , “total\_eve\_charge”]].values
     + Y = churn\_df[“churn”].values
     + Print(X.shape , y.shape)
       - # (3333 , 2) , (3333,)
     + Knn = KNeighborsClassifier(n\_neighbors = 15) # k = 15
     + Knn.fit(X,y)
6. Measuring model performance
   * Accuracy = (correct predictions) / (total observations)
   * Syntax
     + Sklearn.model\_selection import train\_test\_split
     + X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 21, stratify = y) # 0.3 -> 30 % for test
     + Knn = KNeighborsClassifier(n\_neighbors = 6)
     + Knn.fit(X\_train , y\_train)
     + Print(knn.score(X\_test, y\_test))
   * Complexity
     + Larger k = less complex model = can cause underfitting
     + Smaller k = more complex model = can lead to overfiting
   * Syntax
     + Train\_accuracies = {}
     + Test\_accuracies = {}
     + Neighbors = np.arange(1, 26)
     + For neighbor in neighbors:
     + Knn = KNeighborsClassifier(n\_neighbors = neighbor)
     + Knn.fit(X\_train , y\_train)
     + Train\_accuracies[neighbor] = knn.score(X\_train , y\_train)
     + Test\_accuracies[neighbor] = knn.score(X\_test, y\_test)
   * Plotting our result
     + Plt.figure(figsize=(8,6))
     + Plt.title(“KNN: Varying Number of Neighbors”)
     + Pplt.plot(neighbors, train\_accuracies.values(), label=”Training Accracy”)
     + Plt.plot(neighbors, test\_accuracies.values(), label = “Testing Accuracy”)
     + Plt.legend()
     + Plt.xlabel(“Number of Neighbors”)
     + Plt.ylabel(“Accuracy”)
     + Plt.show()

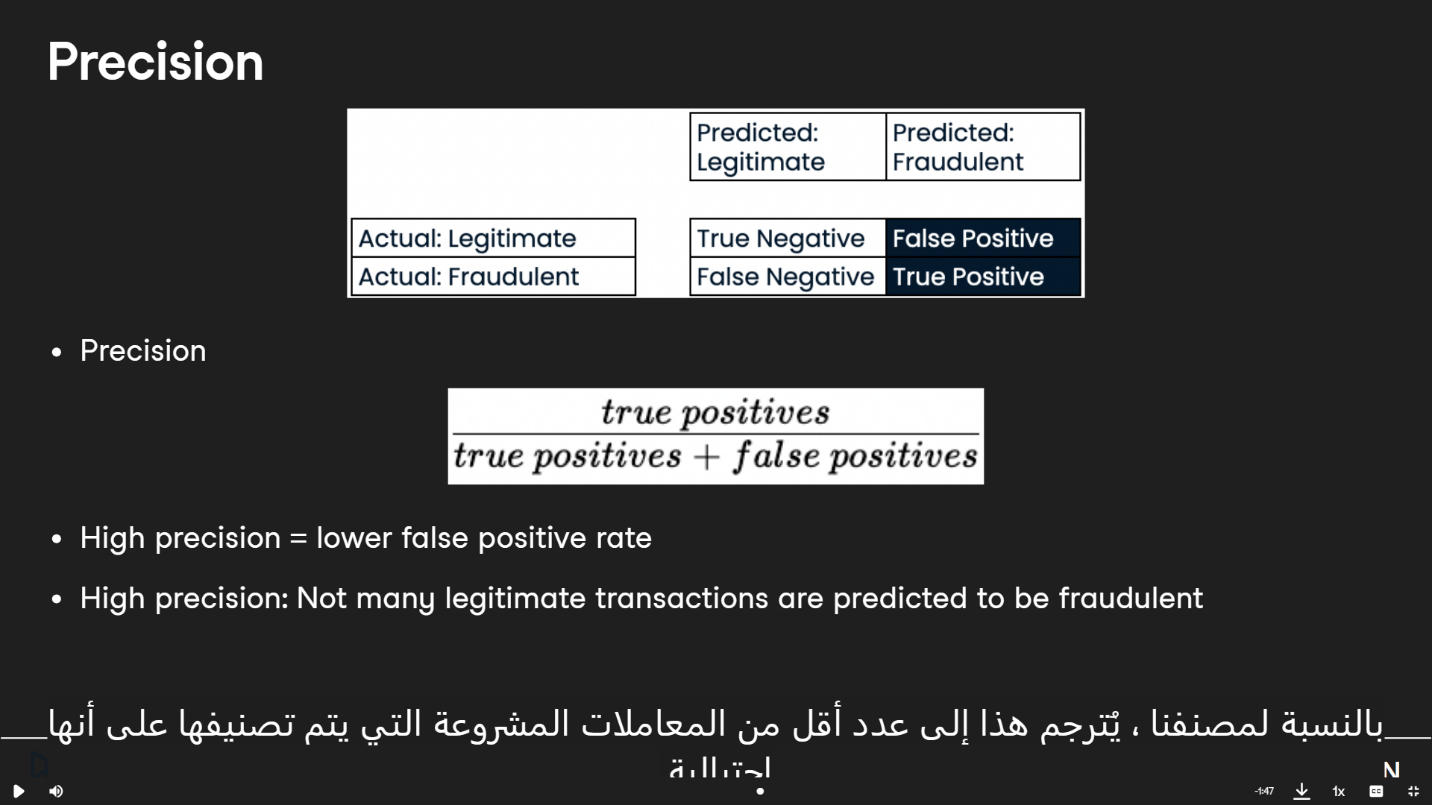
Regression

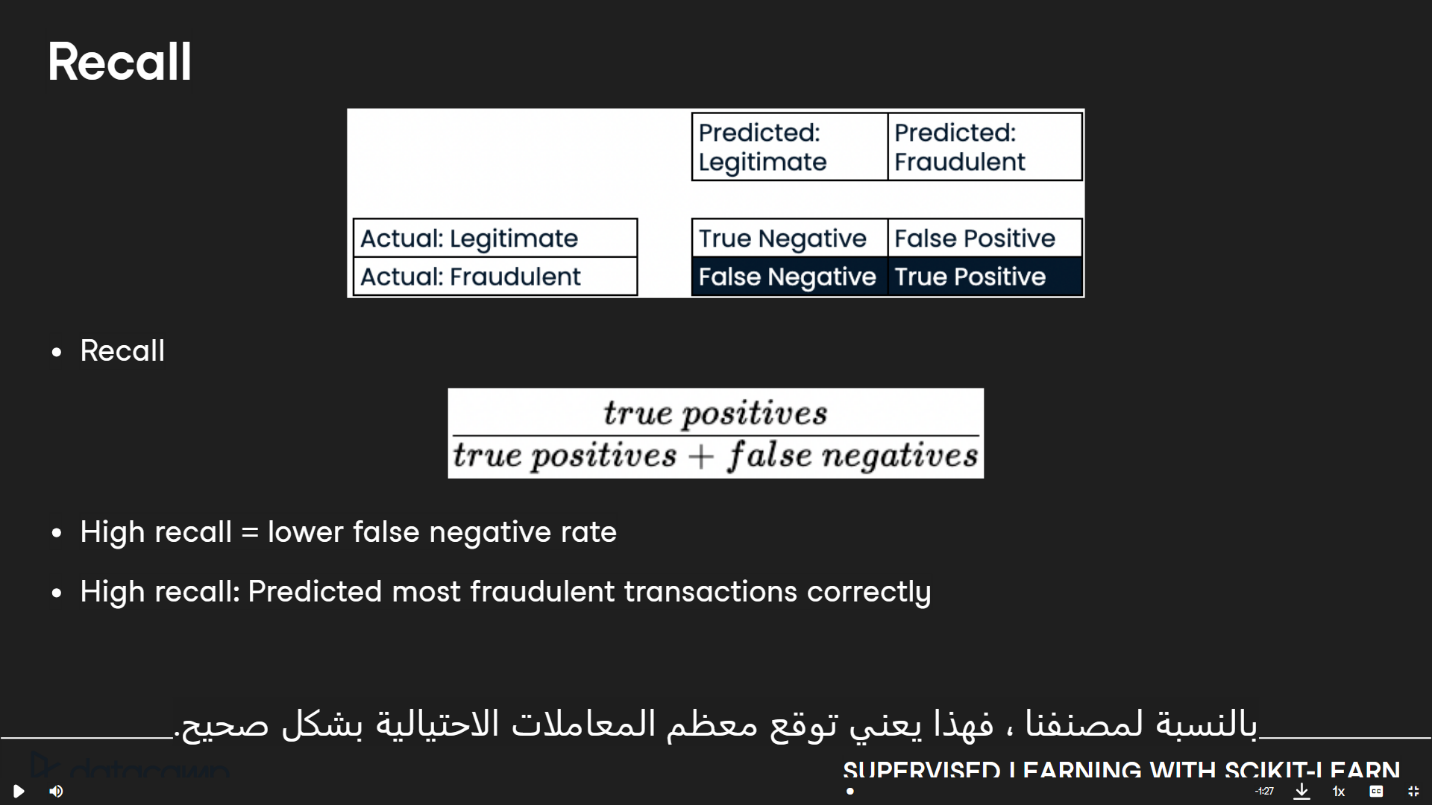
* Introduction
  + Import pandas as pd
  + Diabetes\_df = pd.read\_csv(“diabetes.csv”)
  + Print(diabetes\_df.head())
* Creating feature and target arrays
  + X = diabetes\_df.drop(“glucose” , axis = 1).values
  + Y = diabetes\_df[“glucose”].values
  + Print(type(x) , type(y))
* Making predictions from a single feature
  + X\_bmi = X[:, 3] # to calculate mass from table in 4th column
  + Print(y.shape, X\_bmi.shape)
    - (752,) (752,)
  + X\_bmi = X\_bmi.reshape(-1, 1) #to convert one dimensional to two dimensional
  + Print(X\_bmi.shape)
    - (752, 1)
* Plotting glucose vs. body mass index
  + Import matplotlib.pyplot as plt
  + Plt.scatter(X\_bmi, y) #draw the relationship by points
  + Plt.ylaabel(“Blood Glucose )
  + Plt.xlabel(“Body Mass Index”)
  + Plt.show()
* Fitting a regression model
  + From sklearn.linear\_model import LinearRegression
  + Reg = LinearRegression()
  + Reg.fit(X\_bmi, y)
  + Predictions = reg.predict(X\_bmi)
  + Plt.scatter(X\_bmi, y)
  + Plt.plot(X\_bmi, prediction)
* R-squared
* 
* Reg\_all.score(X\_test, y\_test) # to return the variance
* Mean squared error and root mean squared error
  + 
  + Syntax
    - From sklearn.metrics import mean\_squared\_error
    - Mean\_squared\_error(y\_test, y\_pred, squared= False)
* Cross -validation in scikit-learn
  + From sklearn.model\_selection import cross\_val\_score, KFold
  + Kf = KFold(n\_splits=6, shuffle = True, random\_state = 42)
    - N\_splits => means that the dataset will be divided into 6 folds
    - Shuffle=> is set to **True**, the dataset is shuffled before it is split into folds
  + Reg = LinearRegression()
  + Cv\_results = cross\_val\_score(reg, X, y, cv=kf)
    - to perform k-fold cross-validation on a linear regression model (**reg**)
    - **cv=kf:** The **cv** parameter specifies the cross-validation splitting strategy, and in this case, it's set to the **KFold** object (**kf**) created earlier. It means the dataset will be split into the specified number of folds (6 in this case), and each fold will be used as a validation set exactly once.
    - **cv\_results:** After running **cross\_val\_score**, **cv\_results** will contain an array of scores, where each score corresponds to the performance of the model on a different fold of the data.
* 
* 
  + Rridge
    - which is a type of linear regression that includes regularization

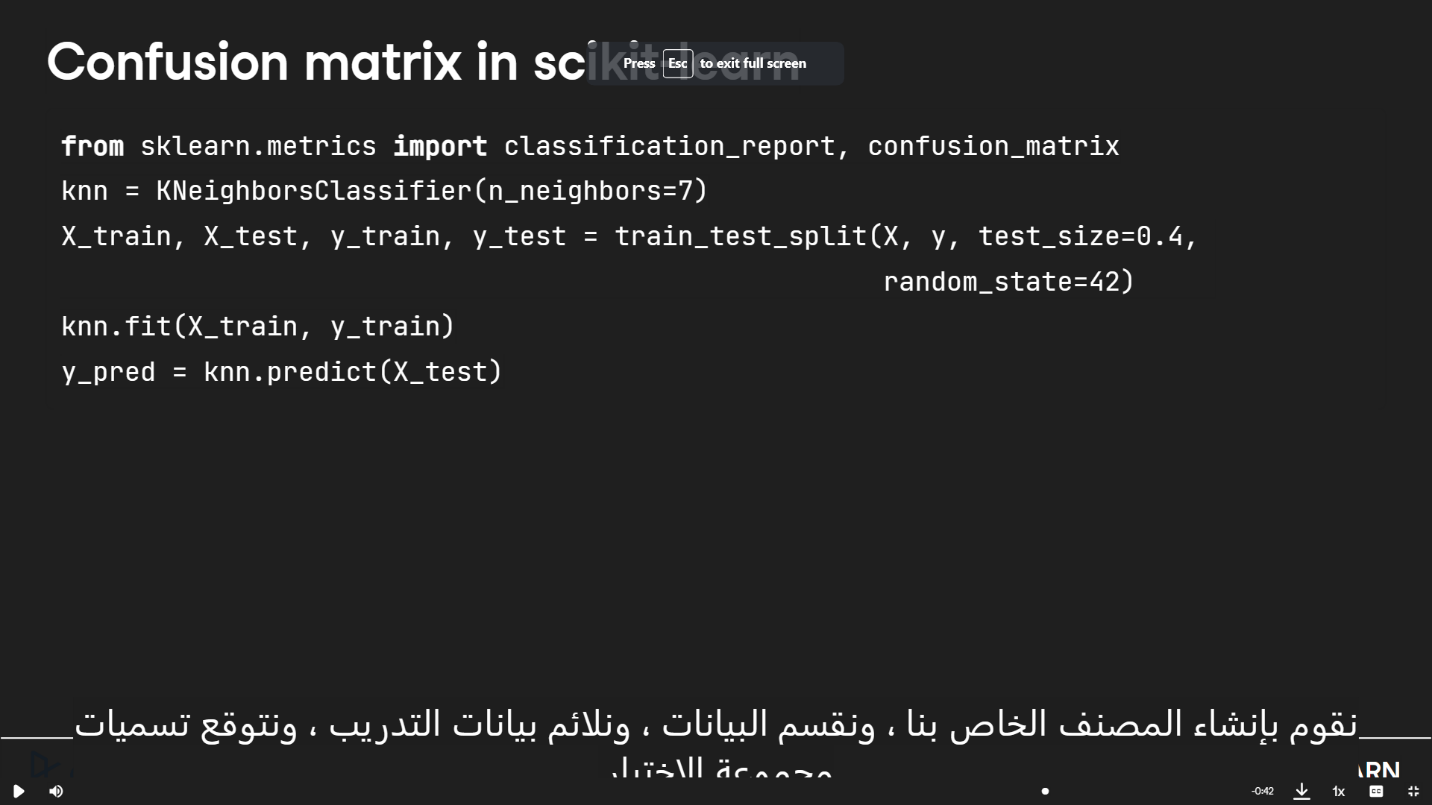
Fine-Tuning Your Model

* How good is your model?
  + Confusion metrics

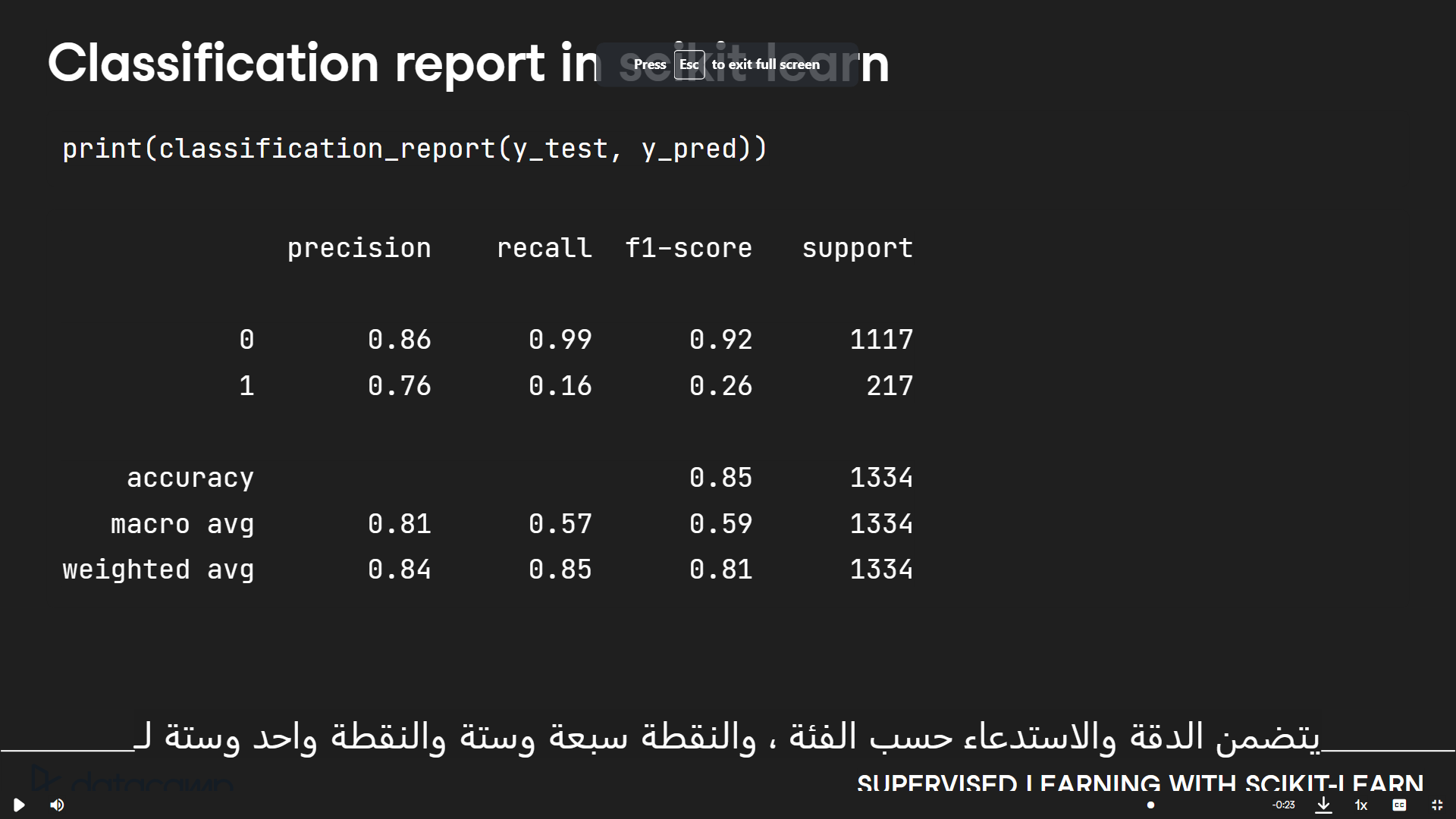








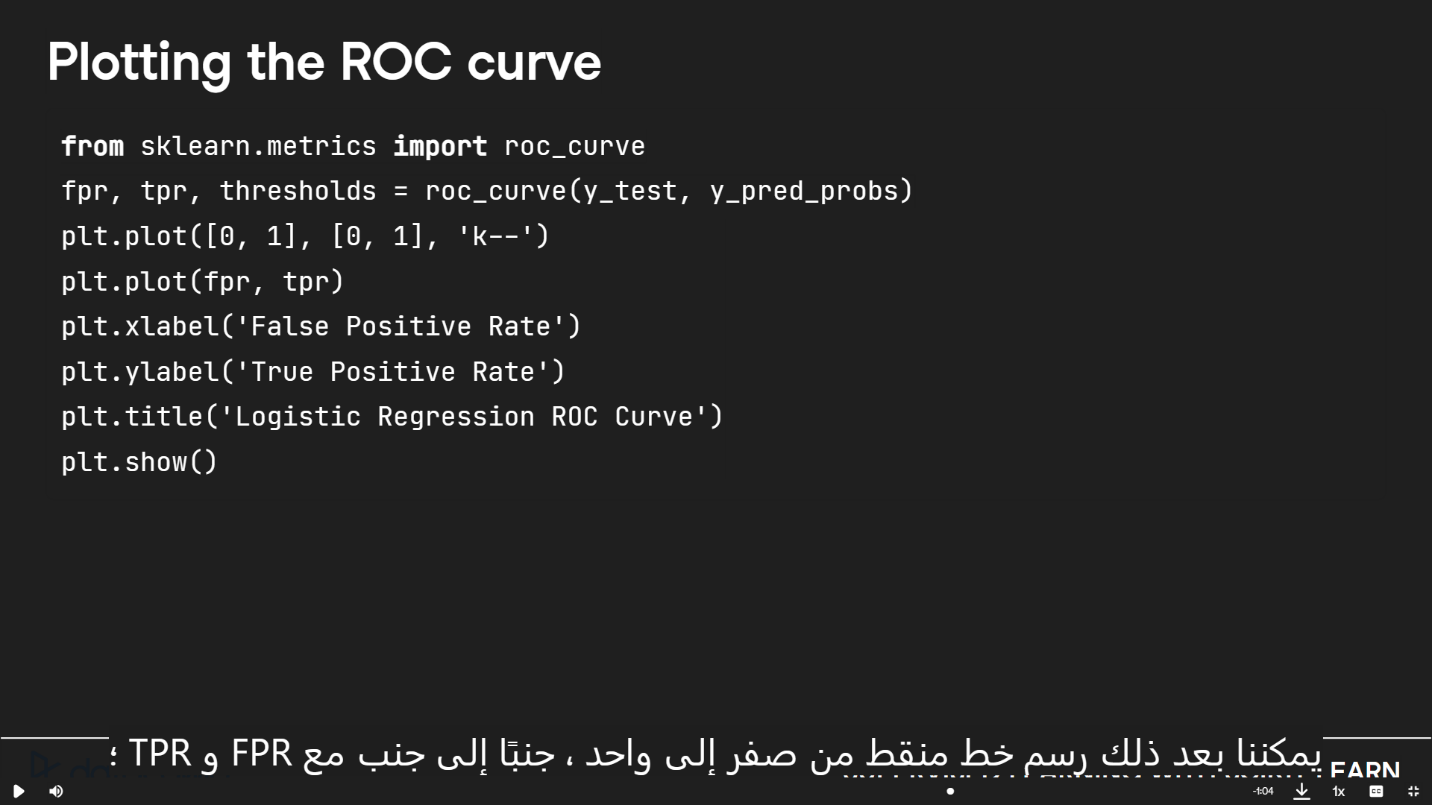
* Print(confusion\_matrix(y\_test, y\_pred))
  + - # [[1106 11]
    - [183 34]
      * 1106 is True Negative



* Logistic regression for binary classification



* + Predicting probabilities
    - Y\_pred\_probs = logreg.predict\_proba(X\_test)[:, 1]
    - Print(y\_pred\_probs[0])



* + Roc curve
    - From sklearn.metrics import roc\_auc\_score
    - Print(roc\_auc\_score(y\_test, y\_pred\_probs))
* Hyperparameter tuning





