Python 3 Notes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Type | | | | | | | Integer: 0,2,7  Float: 1.3, 6.23  String: ‘hello’, “123”  Booleans: True, False  Set: {unique elements}  Dictionary: {key: values}  List: [ ]  Tuples: ( )  Immutable(cannot change value inside) | | | | | | | | | | | | | | | | | | | ('a',1,)  ('a',)  ('a')  t = (1,2,3,4)  (1,2) + (3,4)  ((1,2),) + (3,4) | | | | | | | | | | | | | -> tuple  -> tuple  -> str  t[10:] => ()  t[:10] => (1,2,3,4)  (1,2,3,4)  ((1,2), 3, 4) | |
| Arithmetic Operations | | | | | | | 1. Bracket ()  2. Exponent: \*\*  3. (Multiply: \*) | (Divide: /) | (Quotient: //) | (Modulus/: %) | | | | | | | | | | | | | | | | | | | | | | | | 4. (Plus: +) | (Minus: -)  5. (Greater: >) | (Less or equal: <=) | (Equals to: ==) | (Not equal to: !=) | | | | | | | | | |
| Commentating | | | | | | | # Comment | | | | | """  Long Comment  """ | | | | | | | | | | | | | | | | | | -when long comment used in function, help(fn) will output the comment inside | | | | | | | | | | |
| Variable name = ‘Sam’  num = 12  l = ['a', 'b', 'c'] | | | | | | | print(‘My name is {} and my age is {}’.format(name, num))  print(‘My name is {} and my age is {}’.format(Sam, 12))  print('My name is %s and my age is %s' %(name, num))  print(‘My name is ‘ + Name + ‘my age is ‘ + str(num))  print('My name is %r and my age is %s' %(name, num))  print(f'My name is {name!r} and my age is {num:1.0f}')  print('Floating point numbers: %5.2f' %(13.144))  print('a', 'b')  print(\*l, sep =' & ') | | | | | | | | | | | | | | | | | | | | | | | | My name is Sam and my age is 12  My name is 'Sam' and my age is 12  Floating point numbers: 13.14  # 5 is min no. of char including whitespace; 2 is no. of dp  a b #Using , automatically adds space  a & b & c  #\* print out all items, sep = separator | | | | | | | | | |
| Indexing | | | | | | | s = ’abcdef’ | | | | s[0] = ‘a’  s[-1] = ‘f’  s[:2] = ‘ab’  s[:-1:2] = ‘ace’ | | | | | | | s = [a,b,[c[d,e]]]  d = {‘k1’:[1,2,3]} | | | | | | | | | | | s[2][1][0] = ‘d’  d[‘k1] = [1,2,3] | | | | | | | | | | | |
| AND OR | | | | | | | and – both conditions true (evaluate all conditions)  or – at least 1 condition true (evaluate 1st condition first)  not 0, not None, not '' are all true | | | | | | | | | | | | | | (1>2) and (2>3)  (1<2) and (2<3)  (1<2) or (2>3) or (1==1)  20 or True  True or 20  20 and True  True and 20  False==0 and False == '' | | | | | | | | | | | | | | | | | False  True  True  20  True  True  20  False (True and False) | | |
|  | | | | | | | if 1<2:  print(‘hello’)  elif \_\_\_:  \_\_\_  else \_\_\_:  \_\_\_ | | | | | | #short form  print('hello') if 1<2 else: ... | | | | | | | | | seq = [1,2,3,4,5]  for item in seq:  print(item)  OR  for x in range (0,6):  print(x) | | | | | | | | | | | | | | | 1  2  3  4  5 | | | |
|  | | | | | | | i = 1  while i<4:  print('i is: ' + str(i)) | | | | | | | i is :1  i is :2  i is :3 | | | | | | | | | | | 1 > 1, 000  3,2,1>0,1,2  (1,2,3) + (1,2,3) | | | | | | | | | | | | | | | (False, 0)  (3,2,True,1,2)  (1,2,3,1,2,3) |
| x = 'Sam' list1 = [1,2,3]  list2 = ['a', 'b', 'c'] | | | | | | | break  continue  pass  for item in x:  if item == 'a':  continue  print('item')  for item in x:  if item == 'a':  break  print('item')  for item in enumerate(x):  print(item)  for item in zip(list1, list2):  print(item)  list1.extend(list2)  list1 | | | | | | | | | | | | | | | | break out of current loop  goes to top of nearest loop  does nothing (to not get error when loop is empty)  S  m  S  (0, 'S')  (1, 'a')  (2, 'm')  (1, 'a')  (2, 'b')  (3, 'c')  [1, 2, 3, 'a', 'b', 'c'] | | | | | | | | | | | | | | | | | |
| x = [1,2,3,4]  out = [] | | | | | | | for num in x:  out.append(num\*\*2)  print(out) | | | | | | | | x = [1,2,3,4]  out = [num\*\*2 for num in x]  print(out) | | | | | | | | | | | | | | | | | | [1,4,9,16] | | | | | | | |
| Function | | | | | | | def my\_func(parameter):  print(parameter)  def say\_hello(name='Default')  print(f'Hello {name}')  say\_hello() | | | | | | | | | | | | | | | | print --> value not stored/remembered  return --> can store in variable  --> default value if no arg passed  Hello Default | | | | | | | | | | | | | | | | | |
| def times2(var):  return var\*2  def myfunc(\*args):  return sum(args) \* 0.05  def myfunc(\*\*kwargs):  print(kwargs)  if 'fruit' in kwargs:  print('My fruit is {}'.format(kwargs['fruit']))  else:  print('I do not like fruit')  myfunc(fruit = 'apple', veggie = 'lettuce') | | | | | | | | | | | | | | | | t = lambda var: var\*2  --> can insert multiple numbers ('args' is dummy variable and in form of tuple)  --> only keywords  {'fruit': 'apple', 'veggie': 'lettuce')  My fruit is apple | | | | | | | | | | | | | | | | | |
| Built-in Functions seq = [1,2,3,4,5]  lst = [True, False, True]  # All iterables(can only print out items once) | | | | | | | | | list(map(lambda num: num\*2, seq))  list(filter(lambda num: num%2 ==0, seq)  reduce(lambda x,y:10\*x+y, seq)  all(lst)  any(lst)  list(zip([1,2,3], ['a', 'b', 'c'], ['+','-','\*'])) | | | | | | | | | | | | | | [2,4,6,8,10]  [2,4] --> filter will ensure condition is true  12345 ((((1\*10+2)\*10+3)\*10+4)\*10+5)  False  True  [(1, 'a', '+'), (2, 'b', '-'), (3, 'c', '\*')] | | | | | | | | | | | | | | | | | |
| Higher Order Fn | | | | | | | foo = lambda x: lambda a,b: a\*x+b  bar = foo(5)  bar(2,3)  def thrice(f):  return lambda x: f(f(f(x)))  add1 = lambda x: x + 1  thrice(thrice(add1))(0)  thrice(thrice)(add1)(0) | | | | | | | | | | | | | | | | 13 (2\*5+3)  9  27 | | | | | | | | | | | | | | | | | |
| Methods name= 'sAm', x='s a m'  lst = [1,2,3]  lst2 = [1,2,4] | | | | | | | name.upper()  name.lower()  name.title()  x.split() | | | | | | | | | | | | | | | | SAM  sam  Sam  ['s', 'a', 'm'] | | | | | | | | | | | | | | | | | |
| d.keys() --> return keys  d.items() --> return key and values  d.values() --> return values  t.count('a') --> count no. of times 'a' is in tuple  t.index('a') --> smallest index of 'a' in tuple  # List and dictionary are mutable => their original global values will be changed if passed into a function  lst2 > lst --> True  # Compare first element and so on until unequal element is found. If first N elements equal, but one list longer, longer list is greater | | | | | | | | | | | | | | | | list.pop() --> remove and return last value  list.pop(0) --> remove 1st value  list.append(('a', 'b')) --> add 'new' to end of list  # [1,2,3, ('a', 'b')]  list.extend(('a','b')) --> add iterable to end of list  # [1,2,3, 'a', 'b']  list.insert(obj, index) --> insert object at index  list.reverse() --> reverse elements in list  list.copy() --> copy list if dont want original list to be affected | | | | | | | | | | | | | | | | | |
| Sorting | | | | | | | Inplace  Stable  {-2, 4, 5, -11, 9, -10}  {-2, -11, -10, 4, 5, 9}  bubble sort (stable, inplace)  selection sort (inplace)  insertion sort (stable, inplace)  merge sort | | | | | | | | | | | | | | | | - No additional space required, elements are reindexed  - Relative index remain the same  Before sorting by negative integers first  After sorting, relative index same  - for i in range, compare i and i+1 (each iteration will decrease by 1 as largest value to end)  - find min value in unsorted list, place min value at end of sorted list, repeat  - for each i, find index to insert in sorted list  - | | | | | | | | | | | | | | | | | |
|  | | | | | | | 'x' in [1,2,3]  '1' in [1,2,3]  ord('o')  ord('1')  '12345' < '2345'  isinstance(obj, data type) | | | | | | | | | | | | | | | | False  True  111 (Unicode code point, alphanumeric character  49 have a number assigned to them)  True (only look at first alphanumeric value of str)  # Better than checking type as considers inheritance as well (subclass will be same as class in isinstance but not for type()) | | | | | | | | | | | | | | | | | |
| x = [(1,2), (3,4)] | | | | | | | for items in x:  print(item)  for (a,b) in x:  print(a)  for (a,b) in x:  print(a)  print(b) | | | | | | | | | | | | | | | | (1,2)  (3,4)  1  3  1  2  3  4 | | | | | | | | | | | | | | | | | |
| Truthy/Falsey value  - values which evaluate to true / false | | | | | | | False: [], '', (), {}, range(0), 0, 0.0, None, False  Truth: non-empty list, dict..., non-zero numbers, Truth | | | | | | | | | | | | | | | | However, 1 == True (True)  5 == True (False)  And 0 == False (True)  [] == False (False) | | | | | | | | | | | | | | | | | |
| Equivalence & Identity | | | | | | | Equivalence refers to same content  Identity refers to same memory space  x = (1,2)  y = (1,2)  x is y  x == y  z = x  z is x  a = (1,2,3)  a == (1,2,3)  a is (1,2,3) | | | | | | | | | | | | | | | | ==  is (Equivalence subset of identity)  False  True  True  True  False (Not stored in same space) | | | | | | | | | | | | | | | | | |
| Function in a function | | | | | | | def a(x,y):  def b(i):  return i\*x\*y  return b | | | | | | | | | | | | | | | | h = (3,5)  h(2) => 30  a(3,5)(2) =>30 | | | | | | | | | | | | | | | | | |
| Big O Notation | | | | | | | O(1)  O(log n)  O(n)  O(n\*log n)  O(n\*\*2)  O(n\*\*3)  O(2\*\*n) | | | | | | | | | | | | | | | | Time proportional to no. of operations (Time to execute once \* no. of times fn is called)  Space proportional to no. of pending operations | | | | | | | | | | | | | | | | | |
| Files | pwd  input = open('filename.txt', 'r')  input.read()  input.readlines()  input.seek(0)  input.write('Hello World')  input.close()  import os  os.getcwd()  os.listdir(param)  import shutil  shutil.move('file', 'new\_dir')  import sendtotrash  sendtotrash.sendtotrash('file')  import zipfile  comp\_file = zipfile.ZipFile('comp\_file.txt', 'w')  comp\_file.write('filename.txt', compress\_type = zipfile.ZIP\_DEFLATED)  comp\_file.close()  zip\_obj = zipfile.ZipFile('comp\_file.txt', 'r')  zip\_obj.extractall('extracted\_content')  shutil.make\_archive('newfile', 'zip', dir\_to\_zip)  shutil.unpack\_archive('filetounpack', 'newfile2', 'zip') | | | | | | | | | | | | | | | | | | present working directory  r: read | w: write | a: append | r+: reading & writing |  w+: writing & reading (overwrite existing file or create new file) | wb: write binary  # output everything in 1 line & cursor goes to end of file  # output lines as separate obj in list  # to get cursor back to start of file  # same as pwd  # param can be any directory, default is pwd  # move file to another directory  # creating a new empty zip file  # unzip file | | | | | | | | | | | | | | | | | | | | | |
| Object Oriented Programming  # Superclass : base class where other object inherit from  # Inheritance  # Subclass | | | | | | | class NameOfClass():    #Class object attribute  num = 0  def \_\_init\_\_(self, param1, param2):  self.param1 = param1  self.param2 = param2  self.area = self.param2 \*2  NameOfClass.num += 1  def some\_mtd(self):  # perform some action  print(f'Hello {self.param1}')  x = NameOfClass('Sam', 20)  x.attri OR NameOfClass.attri  x.param1  x.param2  x.area  x.some\_mtd()  class Name2(NameOfClass):  def \_\_init\_\_(self):  NameOfClass.\_\_init\_\_(self)  OR. super().\_\_init\_\_(self)  print('Attribute of Name2')  def some\_mtd(self):  print('Hi') | | | | | | | | | | | | | | | | --> naming convention follows camel casing: each word is capitalized  --> same attribute for all instance of this class  #NameOfClass.num --> 'random'  --> attribute of the class  --> track how many instances of class created  --> methods that can be called with this class  'random'  'Sam'  20  40  Hello Sam  --> Name2 inherit base class (NameOfClass)  - Name2 will inherit all mtd in NameOfClass  --> Name2 inherit all attribute of NameOfClass in addition to the print statement  --> overwrite method in base class for class Name2 | | | | | | | | | | | | | | | | | |
| Polymorphism  - different classes use the same mtd name  Niko = Dog('Niko')  Felix = Cat('Felix')  pet\_speak(Niko) => Niko say woof!  pet\_speak(Felix) => Felix say meow!  fido = Dog('Fido')  isis = Cat('Isis')  print(fido.speak()) => Fido say woof!  print(isis.speak()) => Isis say meow! | | | | | | | | | | | class Dog():  def \_\_init\_\_(self, name):  self.name = name  def speak(self):  return self.name + ' say woof!'  class Cat():  def \_\_init\_\_(self, name):  self.name = name  def speak(self):  return self.name + ' say meow!'  def pet\_speak(pet):  print(pet.speak()) | | | | | | | | | | | | | | | | | class Animal():  def \_\_init\_\_(self, name):  self.name = name  def speak(self):  raise NotImplementedError('Subclass  must implement this abstract method')  class Dog(Animal):  def speak(self):  return self.name + ' say woof!'  class Cat(Animal):  def speak(self):  return self.name + ' say meow!' | | | | | | | | | | | | |
| Special Methods / Dunder Mtds | | | | | | | | class Book():  def \_\_init\_\_(self, title, author, pages):  self.title = title  self.author = author  self.pages = pages  def \_\_str\_\_(self):  return f'{self.title} by {author}'  def \_\_len\_\_(self):  return self.pages | | | | | | | | | | | | | | | | | | | | b = Book(Python rocks, Jose, 200)  print(b) => Python rocks by Jose  str(b) => 'Python rocks by Jose'  len(b) => 200  --> return this whenever str of b required  - \_\_(function name)\_\_ allows predefined methods to be called on your class  - \_\_equals\_\_ for == | | | | | | | | | | | | |
| Errors | | | | | | | | try:  # Want to run this code that may have an error  result = 10 + '10'  except:  # Runs if code in try fails  print('You aren't adding correctly')  raise MyError("...")  else:  # Runs if code in try runs correctly  print('Added')  finally:  # Code here always run  print('I always run')  class MyError(Exception):  def \_\_init\_\_(self, msg):  super().\_\_init\_\_(msg) | | | | | | | | | | | | | | | | | | | | You aren't adding correctly  I always run  --> To target a specific error, (e.g.  except TypeError: )  - can have multiple except statements  - can raise own error but must define class  --> Code will always run even there is a break in code above. (Usually used to close or save a file)  - default would be subclass of Exception  - msg to be printed out beside error | | | | | | | | | | | | |
| Decorators  - fn that can change / extend behavior of other fn without permanently modifying the other fn | | | | | | | | | def decorator(original\_func):  def wrap\_func():  print('Before original function')  original\_func()  print('After original function')  return wrap\_func  def new\_func():  print('I want to be decorated') | | | | | | | | | | | | decorated\_func = decorator(new\_func)  decorated\_func() => Before original function  I want to be decorated  After original function  @decorator  def new\_func():  print('I want to be decorated')  new\_func()  --> same output as decorated func  --> comment out @decorator to remove if not needed | | | | | | | | | | | | | | | | | | | |
| Generator Fn  - instead of computing an entire sequence of values and hold in memory, it only generate 1 value and wait until next value is needed | | | | | | | | | def gencubes(n):  for num in range(n):  yield num\*\*3  for x in gencubes(3):  print(x)  def simple\_gen():  for x in range(3):  print(x)  g = simple\_gen()  print(next(g))  print(next(g))  print(next(g))  print(next(g))  s = 'ord'  next(s)  s\_iter = iter(s)  next(iter(s))  next(iter(s))  next(iter(s))  gencom = (i/2 for i in ... if ...)  for i in ...:  if ...:  yield i/2 | | | | | | | | | | | | -one example in range(), which just keeps track of last number and add the step size  - more memory efficient as no need to create entire list and iterate through it  0  1  8  --> will rmb the last called value  0  1  2  --> StopIteration error  works with for x in s:  print(x)  --> TypeError as str not iterable  o  r  d  --> generator comeprehension  --> equivalent to | | | | | | | | | | | | | | | | | | | |
| Built-In Modules  lst = list(range(0,20)) | | from collections import Counter  lst = [1,1,1,2,2,2,2,2,3,3,3,3]  Counter(lst)  Counter(lst).most\_common(n=..)  sum(c.values())  c.clear()  list(c)  set(c)  dict(c)  c.items()  Counter(dict(list\_of\_pairs))  c.most\_common()[:-n-1:-1]  c += Counter()  from collections import defaultdict  d = defaultdict(lambda: 0)    from collections import namedtuple  Dog = namedtuple('Dog',['age','breed','name'])  sam = Dog(age=2,breed='Lab',name='Sammy')  sam.age || sam[0]  import datetime  t = datetime.time(4,20,1)  print(t)  t.hour...t.microsecond  t.tzinfo  print(datetime.date.today())  from datetime import datetime, date  print(datetime(2021,2,5,10,47,05)  t.replace(hour = 10)  date1 = date(2021,2,5)  date2 = date(2020,2,5)  date1-date2  import random  random.randint(0,100)  random.seed(101)  random.choice(lst)  random.choice(population = lst, k = 5)  random.sample(population = lst, k = 5)  random.shuffle(lst)  import pdb  pdb.set\_trace()  q | | | | | | | | | | | | | | | | | -works with string as well  - technically a dict so normal dict mtds can be used  Counter({1: 3, 2: 5, 3: 4)  [(2: 5), (3: 4), (1: 3)] # n = no. of most common items  # total of all counts  # reset all counts  # list unique elements  # convert to a set  # convert to a regular dictionary  # convert to a list of (elem, cnt) pairs  # convert from a list of (elem, cnt) pairs  # n least common elements  # remove zero and negative counts  # assign default value to a key that is called even though not present in dict  # assign named indices to tuple values (almost likea creating a class)  2  04:20:01  4...0  None (timezone info)  2021-MM-DD  2021-02-05 10:47:05  datetime.time(10,20,1)  datetime.timedelta(365)  74 (0 and 100 inclusive)  # ensure following calls of randint will generate same output but not same int  16  [4,4,5,20,14] (k = no. of items picked with replacement)  [4,1,13,16,7] (without replacement)  # shuffle position of values in list  # python debugger  # find out value of variable at that pt of trace  # to quit trace | | | | | | | | | | | | | | | | | | | | | |
| Timing code | | import time, timeit  start\_time = time.time()  # code here  end\_time = time.time()  elapsed\_time = end\_time - start\_time  stmt = '''func\_one(100)'''  setup = '''#code for func\_one'''  timeit.timeit(stmt, setup, number = 100000) | | | | | | | | | | | | | | | | | # statement to be called  # number: number of time func\_one is called | | | | | | | | | | | | | | | | | | | | | |
| Regex (regular expression)  match = re.search(pattern, text) | | | | | | import re  r"(\d\d\d)-\d\d\d-\d\d\d\d"  r"(\d{3})-\d{3}-\d{4}"  re.search(pattern, text)  match.span()  match.start()  match.end()  re.findall(pattern, text)  match.group()  re.compile(r'(\d{3})-(...)-(...)') | | | | | | | | | | | | | - r (to inform python value inside string are identifiers)  # \d: digit || \w: alphanumeric || \s: white space || \D: non-digit || \W: non-alphanumeric || \S: non-whitespace  - () and - are exact identifiers that we are looking for  # +: >=1 || {3}: exactly 3 times || {2,4}: 2-4 times ||  (3,}: >= 3 || \\* : >= 0 || ? : 1 or 0  - text: where to search ,  pattern|pattern2: search for pattern or pattern2 ,  .pattern: search for \_pattern (e.g. .at: => cat, hat, sat)  ^: start with, $: end with, [^]: exclude things in bracket  [^]+ : exclude things and combine back to string  (12,17) #index of matches (only return first match)  12  17  # return pattern that you are searching for  # group the values and can call group(n) | | | | | | | | | | | | | | | | | | | | | |
| Web-Scrapping | | | Html  CSS  Javascript  CSS syntax  soup.select('div')  soup.select('#some\_id')  soup.select('.notice')  soup.select('div span')  soup.select('div > span')  import requests  res = requests.get('url')  import bs4  soup = bs4.BeautifulSoup(res.text, 'lxml')  search = soup.select(...)  search.text  url = search[0]['src']  link = requests.get('url')  file.write(link.content)  base\_url = 'http://abcde/page-{}.html'  base\_url.format('i') | | | | | | | | | | | | | | Basic structure and content  Design and Style  Interactive elements of webpage  All elements with the <div> tag  The HTML element containing the id attribute of some\_id  All the HTML elements with the CSS class named notice  # if class = some class -> soup.select('.some.class')  Elements named <span>  within an element named <div>  Any elements named <span> that are directly within an element named <div>, with no other element in between  #General steps to getting text off website  # To download image to computer  # acts like a dict object. src: source url  # link.content will be in binary form, so open file with 'wb'  # Going through different page of website  # http://abcde/page-i.html | | | | | | | | | | | | | | | | | | | | | | | |
| Images | | | from PIL import Image  img = Image.open('example.jpg')  img.show()  img.size  img.filename  img.format\_description  i2 = img.crop(x,y,w,h)  img.paste(im = i2, box = (x,y))  img.resize(h,w)  img.rotate(90, expand = True)  img.putalpha(val)  img.save('new.jpg') | | | | | | | | | | | | | | # pillow  # No need .show() for jupyter notebook  (1993,1257)  'example.jpg'  'JPEG (ISO 10918)'  # top left is (0,0) and (w,h) is coordinates of end pt  # Paste i2 over img, box is top left coord of i2  # resize image  # first arg is degree to rotate,  # without expand, new img will have old dimensions and # empty space will be filled black  # Make image transparent  # Save image | | | | | | | | | | | | | | | | | | | | | | | |
| Numpy arr = np.array([[5,10,15], [20,25,30], [35,40,45]]) | | | | | | | import numpy as np  my\_list = [1,2,3]  arr = np.array(my\_list)  arr  x = ([1,2,3]), [4,5,6])  x | | | | | | | | | | | | | | | | array([1,2,3])  array([[1,2,3],  [4,5,6]]) | | | | | | | | | | | | | | | | | |
| np.arange(0,10,2)  np.zeros(3)  np.ones(2)  np.zeros((2,3)) | | | | | | | | | | | | | | | | array([0,2,4,6,8])  array([0,0,0])  array([1,1])  array([[0,0,0],  [0,0,0]]) | | | | | | | | | | | | | | | | | |
| np.linspace(0,5,11)  -->(start, stop, no. of values)  np.eye(3)  -->identity matrix | | | | | | | | | | | | | | | | array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])  array([[1,0,0],  [0,1,0]  [0,0,1]]) | | | | | | | | | | | | | | | | | |
| np.random.rand(1,2)  -->(row, column)  np.random.randn(1,2)  np.random.randint(1,100,10)  -->(start inclusive, end exclusive, no. of values) | | | | | | | | | | | | | | | | -->random no. in uniform dist.  -->random no. in normal dist.  -->random int. in range | | | | | | | | | | | | | | | | | |
| arr.reshape(5,5)  arr.max()  arr.min()  arr.argmax()  arr.shape()  arr.dtype()  arr\_copy = arr.copy() | | | | | | | | | | | | | | | | -->reshape array into type of matrix  -->max value in array  -->min value  -->index of max value  -->tells you shape/dimension of matrix  -->tells you data type  -->when you dont want original data to be affected | | | | | | | | | | | | | | | | | |
| arr[1,2] OR arr[1][2]  arr[:2, 1:]  -->[row, column]  arr>10  arr[arr>10] | | | | | | | | | | | | | | | | 30  array([[10,15],  [25,30]])  array([[False, False, True],  [True, True, True],  [True, True, True]])  array([15,20,25,30,35,40,45]) | | | | | | | | | | | | | | | | | |
| np.sqrt(arr)  np.exp(arr)  np.sin(arr)  np.log(arr)  np.sum(arr)  np.isnan(var) | | | | | | | | | | | | | | | | \*\*  sine  logarithm  sum array  return boolean of var is null | | | | | | | | | | | | | | | | | |
| Pandaslabels = ['a', 'b', 'c']x = [10, 20, 30]d = {'a':10, 'b':20, 'c':30} | | | | | | | import pandas as pd  pd.Series(np.array(x)) OR  pd.Series(x)  pd.Series(data = x, index = labels)  OR  pd.Series(d) | | | | | | | | | | | | | | | | 0 10  1 20  2 30  dtype: int64  a 10  b 20  c 30  dtype:int64 | | | | | | | | | | | | | | | | | |
| ser1 = pd.Series([1,2,3], ['US', 'Germany', 'USSR']  ser2 = pd.series([1,2,5], ['US', 'Germany', 'Italy']  ser1 + ser2 | | | | | | | | | | | | | | | | | | | Germany 4.0  Italy NaN  US 2.0  USSR NaN  dtype: int64 | | | | | | | | | | | | | | |
| DataFrame | | | | | | | df = pd.DataFrame(randn(3,2), ['A', 'B','C'], ['X', 'Y']  -->(data, index, column) | | | | | | | | | | | | | | | | X Y  A 2.7 0.6  B. 0.7 1.9  C -2.0 0.1 | | | | | | | | | | | | | | | | | |
| df['X']  df['X'] > 0  df[ df['X'] > 0]  df['new'] = df['X'] + df['Y']  df.reset\_index()  df.set\_index('List')  df.idxmin() | | | | | | | | | | | | | | | --> get X column  --> get boolean for condition  --> get data which satisfy condition  --> insert new column in dataframe  --> change index to 0,1,2 | default is not permanent  --> change index  --> gives index of min value for each column | | | | | | | | | | | | | | | | | | |
| X Y  A 2.7 0.6  B. 0.7 1.9 C -2.0 0.1 | | | | | | | df.loc['A']  OR  df.iloc[0]  df.loc['B', 'Y'] | | | | | | | | | | | | | | | | X 2.7  Y 0.6  --> index of row  1.9 --> [row, column] | | | | | | | | | | | | | | | | | |
|  | | | | | | | df[ (df['X'] > 0) & (df['Y' > 1)]  df[ (df['X'] > 0) | (df['Y' > 1)] | | | | | | | | | | | | | | | | --> and condition  --> or condition | | | | | | | | | | | | | | | | | |
| Multilevel Index | | | | | | | arr = [np.array(['G1', 'G1', 'G1', 'G2', 'G2', 'G2']),  np.array(['1', '2', '3', '1', '2', '3'])]  df=pd.DataFrame(np.random.randn(6,2), index=arr, columns=['A', 'B'])  df.index.names=['Groups', 'Num']  df  df.loc['G1'].loc['1']  df.xs('1', level='Num') | | | | | | | | | | | | | | | | | | |  |  |  |  | | --- | --- | --- | --- | | Groups | Name | A | B | | G1 | 1 | 0.30 | 1.69 | | 2 | -1.70 | -1.15 | | 3 | -0.13 | 0.39 | | G2 | 1 | 0.16 | -1.20 | | 2 | 0.80 | 1.10 | | 3 | 0.63 | 0.75 |   A. 0.30  B 1.69  Name: 1, dtype:float64   |  |  |  | | --- | --- | --- | | Groups | A | B | | G1 | 0.30 | 1.69 | | G2 | 0.16 | -1.20 | | | | | | | | | | | | | | | | |
|  | | | | | | | df.drop('R')  df.drop('C', inplace=True, axis=1)  OR  df = df.drop('C', axis=1)df.dropna()  df.dropna(axis=1)  df.dropna(thresh=2)  df.fillna(value='...')  df['A'].fillna(value=df['A'].mean()) | | | | | | | | | | | | | --> drop row E, default is row drop not permanent  --> permanently drop column C  --> remove columns with null value  --> remove rows with >=2 null values  --> fill null values  --> for column A, fill null values with mean of column A | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | df.groupby('Groups')  df.groupby(by=['List'])  df['A'].mean() sum, std, count  df.describe()  df.info()  df.head(5)  df.transpose() | | | | | | | | | | | | | | | | --> compile into groups by column  --> compile into multiple groups by column  --> get average, sum, std deviation, count  --> get stats of df  --> get data type and null values  --> get first 5 rows of df  --> transpose df | | | | | | | | | | | | | | | | | |
|  | | | | | | | pd.concat([df1, df2, df3], axis = .., join = )  pd.merge(df1, df2, how = 'left', on = 'col')  df1.join(df2, how = 'outer') | | | | | | | | | | | | | | | | concat to stack data  --> axis = 0: concat along rows (default)  --> axis = 1: concat along columns  --> join = outer: union (default)  --> join = inner: intersection  merge to combine dataframes according to column  --> how = inner (default), outer, left, right  --> on = column name: combine according to this column  join same as merge but column = index only  --> how = left (default), inner, outer, right | | | | | | | | | | | | | | | | | |
|  | | | | | | | df['A'].unique()  len(df['A'].unique()])  OR  df['A'].nunique()  sorted(df['A'].unique())  df['A'].value\_counts()  df['A'].apply(lambda x: x\*2)  df.columns()  df.index()  df.sort\_values('A')  df.isnull()  df.pivot\_table(values = ['..'], index = ['..'], columns = ['..'] | | | | | | | | | | | | | | | | --> array of unique values in column A  --> no. of unique values in array  --> sort unique value in array  --> display no. of times value occurs  --> map own function onto df['A']  --> column names  --> shows index  --> sort df according to values in column A (index remain the same according to orginal)  --> boolean of null values  --> pivot unstructured data into dataframe according to index and column specified | | | | | | | | | | | | | | | | | |
|  | | | | | | | pd.read\_csv('file\_name.csv')  df.to\_csv('filename', index = False)  pd.read\_excel('..', sheetname = 'Sheet1')  df.to\_excel('..', sheetname = '..')  pd.read\_html('..html')  pd.read\_sql('..', con = ..) | | | | | | | | | | | | | | | | --> read csv file  --> change df to csv file (default index not saved)  --> read excel file  --> read html file  --> read sql file (con = connection) | | | | | | | | | | | | | | | | | |
|  | | | | | | | df.select\_dtypes(int)  pd.to\_datetime(df['A'])  df.corr()  df.corrwith(df2)  df.replace(inplace = '..', value = '..') | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | --> subset of df with datatype according to specified (int, float, obj, bool)  --> convert str to datetime obj  --> correlation btw columns  --> correlation btw dataframes  --> replace data in inplace (can be list) with data in value | | | |
| Matplotlib | | | | | | | import matplotlib.pyplot as plt  %matplotlib inline  plt.show()  plt.plot(x,y)  plt.xlabel('X axis')  plt.ylabel('Y axis')  plt.title('Title') | | | | | | | | | | | | | | | | --> for jupyter notebook  --> last line if not jupyter notebook  --> label x axis  --> label y axos  --> label title | | | | | | | | | | | | | | | | | |
| Multiple graphs in 1 canvas/var/folders/8b/n4frqn8n3f927hpnmlxv1dlr0000gp/T/com.microsoft.Word/Content.MSO/B0E25CA1.tmp | | | | | | | plt.subplot(1,2,1)  plt.plot(x,y,'r')  plt.subplot(1,2,2)  plt.plot(y,x,'b') | | | | | | | | | | | | | | | | --> dimensions of graph (height, width, plot no.)  --> (x,y, color(red))  --> (x,y, color(bluez)) | | | | | | | | | | | | | | | | | |
| fig, axes = plt.subplots(nrows = 1, ncols = 2)  axes[0].plot(x,y, label='..')  axes[1].plot(y,x) | | | | | | | | | | | | | | | | --> (1 row by 2 columns of graph)  --> can label graph | | | | | | | | | | | | | | | | | |
| 2 diagram overlay each other/var/folders/8b/n4frqn8n3f927hpnmlxv1dlr0000gp/T/com.microsoft.Word/Content.MSO/FDF162DB.tmp | | | | | | | fig = plt.figure()  axes1 = fig.add\_axes([0.1,0.2,0.8,0.8])  axes2 = fig.add\_axes([0.2,0.5,0.4,0.3])  axes1.plot(x,y, color = '..', linewidth = .., linestyle = '..', alpha = 0.5, marker = '..', markersize = .., markerfacecolor = 'yellow'  markeredgewidth = .., markeredgecolor = ..)  axes2.plot(y,x) | | | | | | | | | | | | | | | | --> creating a canvass  --> [10% from left, 20% from bot, 80% width, 80% height]  --> color (#RGB Hex code for more colors), linewidth: default 1, increase for thicker line  linestyle: '- -' (dashed), '-' (solid), '-.' (dashed dot) ':' (dotted)  alpha: transparency  marker: point marker. 'o', '+', '\*', '1', 'x'  markersize: size of marker  markerfacecolor: color of markers  markeredgewidth: width of marker edge  markeredgecolor: color of marker edge | | | | | | | | | | | | | | | | | |
| 2 graph on 1 diagram/var/folders/8b/n4frqn8n3f927hpnmlxv1dlr0000gp/T/com.microsoft.Word/Content.MSO/D90A6857.tmp | | | | | | | fig = plt.figure()  ax = fig.add\_axes([0,0,1,1])  ax.plot(x, x\*\*2, label = 'X Squared')  ax.plot(x, x\*\*3, label = 'X Cubed')  ax.legend(loc=0)  ax.set\_xlimit([0,2])  ax.set\_ylimit([0,1]) | | | | | | | | | | | | | | | | --> loc=location  0:best, 1:upper right, 2:upp left, 3:low left,  4:low right, 5:right, 6:center left, 7:center right  8:low center, 9: upp center, 10:center  --> OR loc = (0.1,0.1) : 10% left, 10% bot  --> set x limit(lower limit, upper limit) | | | | | | | | | | | | | | | | | |
|  | | | | | | | plt.tight\_layout()  plt.figure(figsize=(12,5), dpi = 100)  fig.savefig('file\_name') | | | | | | | | | | | | | | | | --> to spread out graph  --> figsize in inches (width, height)  --> save figure | | | | | | | | | | | | | | | | | |
| plt.scatter(x,y)  plt.hist()  plt.boxplot() | | | | | | | | | | | | | | | |  | | | | | | | | | | | | | | | | | |
| SeabornS = x = 'A', y = 'B', data = df | | | | | | | import seaborn as sns  sns.displot(df['A'])  sns.jointplot(S, kind = '..')  sns.pairplot(df, hue = 'C', palette = 'coolwarm')  sns.rugplot(df['A'])  sns.barplot(S, estimator = np.std)  sns.countplot(x = '..', data = df)  sns.boxplot(S)  sns.violinplot(S, split = True)  sns.stripplot(S, jitter = True)  sns.swarmplot(S)  sns.factorplot(S, kind = 'bar')  sns.lmplot(S, col = 'C', aspect = 0.6, size = 8)  sns.heatmap(df, annot = True, cmap = 'coolwarm', linecolor = 'red', linewidth = 3 )  sns.clustermap(df, standard\_scale = 1) | | | | | | | | | | | | | | | | --> default kind='hist', kind:type of plot  --> scatterplot, kind = hex, reg: regression line, kde  --> jointplot for all variables; hue: separate data using color; palette: type of color (matplotlib colormap)  --> horizontal dash plot: shows density of data  --> default is bar plot with estimator mean  --> bar plot with y = count  --> box and whisker plot  --> boxplot but for all data, split = True: split values according to category along center line  --> scatterplot when one variable is categorical, jitter = True: used when point overlap  --> combine strip & violin plot(not for large data)  --> general plot  --> linear fit; col: split into 2 graphs unlike hue where both plots in 1 diagram; aspect: ratio of width to height; size: absolute size of diagram  --> annot: annotate and label values; cmap: color  --> cluster rows & columns according to similarity, standard\_scale: normalise data | | | | | | | | | | | | | | | | | |
| g = sns.PairGrid(df)  g.map(plt.scatter)  g.map\_diag(sns.displot)  g.map\_upper(plt.scatter)  g.map\_lower(sns.kdeplot) | | | | | | | | | | | | | | | | --> provides more control than pairplot  --> diagonal graphs shows dist plot  --> upper of diagonal shows scatter plot  --> lower of diagonal shows kde plot | | | | | | | | | | | | | | | | | |
| g = sns.Facetgrid(data = df, col = 'A', row = 'B')  g.map(sns.displot, 'C', 'D') | | | | | | | | | | | | | | | | --> similar to Pairgrid but shows multiple graphs segregated according to rows and cols  --> 'D' if graphs required for more variables | | | | | | | | | | | | | | | | | |
| plt.figure(figsize=(12,3))  sns.set\_style('white')  sns.despine(left = True, bottom = True)  sns.set\_context('poster'), font\_scale = 3  df.plot(x = '..', y = '..'kind = '..', figsize = (.,.), cmap = '..', color = '..') | | | | | | | | | | | | | | | --> will override seaborn  --> style of background( ticks, blackgrid, whitegrid)  --> default remove top and right spine  --> sns has built in format(paper, notebook)  --> use matplotlib to plot | | | | | | | | | | | | | | | | | | |
| Plotly & Cufflinks | | | | | | | import cufflinks as cf  import chart\_studio.plotly as py  import plotly.graph\_objs as go  from plotly.offline import download\_plotlyjs, init\_notebook\_mode, plot, iplot  init\_notebook\_mode(connected=True)  cf.go\_offline() | | | | | | | | | | | | | | | | --> interactive plot | | | | | | | | | | | | | | | | | |
| df.iplot(kind = 'scatter', x = 'A', y = 'B', mode = 'markers', size = 20, colorscale = 'rdylbu', bins = 50)  df.scatter\_matrix() | | | | | | | | | | | | | | | | --> kind: scatter, bar, box, surface, hist, spread, bubble  mode = markers: so that data pts are not connected by lines  size: size of markers (can be another column so that plot has additional variable)  colorscale: redyellowblue  --> similar to pairplot | | | | | | | | | | | | | | | | | |
| Chororpleth Maps | | | | | | | data = dict(type = 'choropleth', locations = [..], locationmode = 'USA-states', colorscale = '..', reversescale = True, text = [..], z = [..], colorbar = {'title': 'Colorbar title goes here'}, marker = dict(line = dict(color = 'rgb(255,255,255), width = 1)  )  layout = dict(geo = (scope': 'usa', showlakes = True, lakecolor = Ygbl(85,173,240))),  choromap = go.Figure(data = [data], layout = layout)  iplot(choromap)  OR  layout = dict(title = '..', geo = dict(showframe = False, projection = {'type': 'Mercator'} | | | | | | | | | | | | | | | | --> type: type of map  locations: name of countries/states  locationmode: ISO-3(for short form of country name in location), USA-states, country names  colorscale: Greys, YlGnBu, Greens, YlOrRd, Bluered, RdBu, Reds, Blues, Picnic, Rainbow, Portland, Jet, Hot, Blackbody, Earth, Electric, Viridis, Cividis  reversescale = True: reverse colorscale  text: info displayed when hovering  z = "y-values"  colorbar: customise colorbar: title, len, lenmode  geo: map layout: world | usa | europe | asia | africa | north america | south america  showlakes: whether lakes are shown  lakecolor: customise color of lakes  marker: customise markers  --> showframe: whether frame is drawn  --> projection: how map is displayed equirectangular | mercator | orthographic | natural earth | kavrayskiy7 | miller | robinson | eckert4 | azimuthal equal area | azimuthal equidistant | conic equal area | conic conformal | conic equidistant | gnomonic | stereographic | mollweide | hammer | transverse mercator | albers usa | winkel tripel | aitoff | sinusoidal | | | | | | | | | | | | | | | | | |
| Machine Learning | | | | | Accurracy = correct predictions / total predictions  Recall = True +ve / (True +ve + False -ve)  Precision = True +ve / (True +ve + False +ve)  F1 score = 2 \*  F1 score best = 1 | | | | | | | | | | | | | | | | | | Confusion Matrix   |  |  |  |  | | --- | --- | --- | --- | |  | | Predictions | | | + | - | | Actual | + | True +ve | False -ve  Type II error | | - | False +ve  Type I error | True -ve | | | | | | | | | | | | | | | | | | |
| Mean Absolute Error(MAE) =  Mean Squared Error(MSE) =  Root Mean Squared Error(RMSE) = | | | | | | | | | | | | | | | | | | | | | | Bias Variance Trade-Off : by constantly adding complexity to model / model overfit, could lead to training data accuracy to increase, while test data accuracy to decrease | | | | | | | | | | | | | |
| General: Supervised Estimator  from sklearn.family import model  model.fit()  model.predict()  model.predict\_proba()  model.score() | | | | | | | | | | | | | | | | | | General: Unsupervised Estimator  from sklearn.family import model  model.fit()  model.transform()  model.fit\_transform()  model.predict()  model.score() | | | | | | | | | | | | | | | | | |
| Linear Regression | | | | X = df[['A', 'B', 'C',...]]  y = df['Y']  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test size = 0.3)  from sklearn.linear\_model import LinearRegression  lm = LinearRegression()  lm.fit(X\_train, y\_train)  coeff\_df = pd.DataFrame(lm.coef\_, X.columns, columns = [''Coefficient']  predictions = lm.predict(X\_test)  plt.scatter(y\_test, predictions)  sns.displot((y\_test - predicitions), bins = 50)  from sklearn import metrics  metrics.mean\_absolute\_error(y\_test, predictions)  metrics.mean\_squared\_error(y\_test, predictions)  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | --> test size: how much data to set aside for test  --> create instance of model  --> fit data to model  --> produce dataframe of coefficient for each x value  --> use test data to predict y value  --> ideally a straight line  --> ideally normal distribution | | | | | | | |
| Logistics Regressionuses logistic function to classify binary variable | | | | aa = pd.get\_dummies(df['A'], drop\_first = True)  df = df.drop(['A'], axis = 1)  df = pd.concat([df, aa], axis = 1)  OR  nameofcol = ['A']  df = pd.get\_dummies(df, columns = nameofcol, drop\_first = True)  from sklearn.linear\_model import LogisticRegression  logmodel = LogisticRegression()  train test split...fit...predict...  from sklearn.metrics import classification\_report, confusion\_matrix  print(classification\_report(y\_test, predictions))  print(confusion\_matrix(y\_test, predictions)) | | | | | | | | | | | | | | | | | | | | --> dtype 'object' and 'category' will be converted to binary values  drop\_first: remove 1 column as the other column can tell result (if female = 0.0, male confirm = 1.0), so no need for 2 columns  --> drop column with classes and combine column with binary value with df  --> to create list of the column  --> columns only accept list objects; change column A to dummies and drop column A  --> Precision, Recall, F1 score, Support  support is the number of occurence of the given class in your dataset | | | | | | | | | | | | | | | | |
| K Nearest Neighbourclassify variable according to data around itK = no. of nearest neighbour you want to take into account | | | | from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaled\_features = scaler.fit\_transform(df.drop('Target Class', axis = 1)  df\_feat = pd.DataFrame(scaled\_features, columns = df.columns[:-1])  train test split with (X = df\_feat, y = df['Target Class'])  from sklearn.neighbors import KNeighborsClassifier  error\_rate = []  for i in range(1,40):  knn = KNeighborsClassifier(n\_neighbors = i)  knn.fit(X\_train, y\_train)  pred\_i = knn.predict(X\_test)  error\_rate.append(np.mean(pred\_i != y\_test))  plt.plot(range(1,40), error\_rate)  knn = KNeighborsClassifier(N\_neighbors = ...)  fit...predict...error report... | | | | | | | | | | | | | | | | | | | | --> fit and standardise data (less "y" column)  --> dataframe of "X" value less "y" column  --> to find the best K value  get mean of False (0) and True (1), where  predictions != y\_test (higher value = higher  --> error rate)  --> choose value near min & surroundings don't fluctuate much  --> sub best K value in  --> can compare report with K=1 | | | | | | | | | | | | | | | | |
| Decision Trees & Random Forests Root = 1st Condition  Edge = Different outcome  Node = Condition  Leaf = Final Outcome  Random Forest combine output of multiple decision tree to generate final output | | | | | | | | | train test split...  from sklearn.tree import DecisionTreeClassifier  dtree = DecisionTreeClassifier  fit...predict...error report...  from sklearn.ensemble import RandomForestClassifier  rfe = RandomForestClassifier(n\_estimators = 200)  fit...predict...error report... | | | | | | | | | | | | | | | | |  | | | | | | | | | | | | | | |
| Support Vector Machines- non-probabilistic binary linear classifierrecognise pattern, - used for classification & regression analysis | | | | | | | Support vector = vector pts that margin line touches  can expand to non linear data using 'kernel trick' | | | | | | | | | | | | | | | | | | --> choose best line to categorise data (if data on left side of line: class A; if data on right side of line: class B) | | | | | | | | | | | | | | | |
| train test split...  from sklearn.svm import SVC  model = SVC()  fit...predict...error report  from sklearn.model\_selection import GridSearchCV  param\_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']}  grid = GridSearchCV(SVC(), param\_grid, refit = True, verbose = 3)  grid.fit(X\_train, y\_train)  grid.best\_params\_  predict...error report... | | | | | | | | | | | | | | | | | | | | | | | | | | | | --> dict of paramters to try  --> refit: use the best paramter found; verbose: messages to see what it is doing | | | | | |
| K Means Clustering- unsupervised learning;- divide data into distinct grps such that observations within each grp is similar;- assign data to cluster for which centroid is closest- K = no. of clusters- sum of squared error(SSE) = sum of squared dist. btw data in cluster & centroid- 'elbow method': k value = when SSE drastically | | | | | | | | | | | | | | | | from sklearn.cluster import KMeans  kmeans = KMeans(n\_clusters = ..)  kmeans.fit(df)  kmeans.cluster\_centers\_  kmeans.labels\_ | | | | | | | | | | | | | | | | | | | | --> coordinates of centroid  --> predicted "y" values | | | | |
| Principal Component Analysis- aka "general factor analysis"- determines several orthogonal(right-angled) lines of best fit to data set- transform higher dimensions of variables to lower dimension that still contain most info- principal component =. new variable that contain info in desc. order- find out which feature explains the most variance in data- can use new df for analysis | | | | | | | | | | scaler...fit\_transform...  from sklearn.decomposition import PCA pca = PCA(n\_components = 2)  x\_pca = pca.fit\_transform(scaled\_data)  x\_pca.shape  plt.scatter(x\_pca[:,0], x\_pca[:,1], c = df['Target'])  plt.xlabel('First PC')  plt.ylabel('Second PC')  pca.components\_  df\_comp = pd.DataFrame(pca\_components, columns = df.columns[:-1])  sns.heatmap(df\_comp, cmap = 'Plasma') | | | | | | | | | | | | | | | | | | | | | | --> how many variables you want  --> see that shape of matrix reduced to 2 principal component  --> 1st PC, 2nd PC, target class  --> does not corresponds to a single feature but a combination of all  --> new df  --> shows rs btw feature and PC | | | | | | | | |
| Recommenders Systems | | | | | | | - content-based: focus on attibutes of items & give recommendations based on similarity btw them  - collaborative filtering: based on knowledge of users' attitude to items to recommend items(based on others buying habits)  CF 1)memory based CF by computing cosine similarity  CF 2) model based CF by using singular value decomposition(SVD)  CF 1)a) user-item filtering = users who are similar to u also liked...  CF 1)b) item-item filtering = users who liked this item also liked... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Natural Language Processing | | | | | - bag of words = words that you are finding  - convert words to vector: 'Red house' -> (red, blue, house) -> (1,0,1) | 'Blue house' -> (0,1,1)  - cosine similarity: sin(A,B) = cos =  - TF - IDF : Term Frequency - Inverse Document Frequency  - TF(d,t) = no. of occurrence of term t in document d (importance of term within document)  - IDF() = log(N, ) where N = total no. of documents, = no. of documents with term t  (importance of term in corpus(grp of all documents))  - = TF(d,t) \* log( | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | import nltk  nltk.download()  messages = [line.rstrip() for line in open('sms/SMS')]  for mess\_no, message in enumerate(messages[:10]):  print(mess\_no, message)  print('\n')  messages = pd.read\_csv('sms/SMS', sep = '\t', names =  ['label', 'message']  import string  string.punctuation  from nltk.corpus import stopwords  def text\_process(mess)  nopunc = [char for char in mess if char not in  string.punctuation]  nopunc = ' '.join(nopunc)  return [word for word in nopunc.split() if word.lower() not  stopwords.words('english')  messages['message'].head(5).apply(text\_process)  from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | --> to download dataset  --> no. the items in data  --> to see the file and how data is stored  --> remove punctuation & change list to indv letters  --> join indv letter according to  ' ' (nothing in this case)  --> remove stopwords  --> test if function works | | | |
| 1) | | | | | | | bow = CountVectorizer(analyzer = text\_process)  print(len(bow.vocabulary\_))  X2 = bow.fit\_transform(messages['message'])  X2.nnz  tfidf = TfidfTransformer()  X = tfidf.fit\_transform(X2)  train test split...  from sklearn.naive\_bayes import MultinomialNB  model = MultinomialNB().fit(X\_train, messages['label'])  pred...error report... | | | | | | | | | | | | | | | | | | | | | | | | | | | --> default analyzer = 'word'  --> to see no. of unique words  --> fit & transform  --> no. of non-zero occurrence  --> fit & transform  --> train test split  --> fit | | | | | | |
| OR 2) | | | | | | | train test split...  from sklearn.pipeline import Pipeline  pipeline = Pipeline([  ('bow', CountVectorizer(analyser = text\_process)),  ('tfidf', TfidfTransformer()),  ('classifier', MultinomialNB())]  pipeline.fit(X\_train, y\_train)  pred...error report... | | | | | | | | | | | | | | | | | | | | | | | | | | |  | | | | | | |
| Neural Network & Deep Learning | | | | | | | 1. Perceptron Model  Input -> f(x) -> Output  insert 'weights' to affect inputs = w  'bias' as offset value / threshold = b | | | | | | | | | | | | | | | | If f(x) is sum | | | | | | | | | | | | | | | | | |
|  | | | | | | | 2. Neural Networks  - Multilayer perceptron model  - output of 1 perceptron become input to another perceptron  - Hidden layers are grp of perceptron btw input and output layers  - When >= 2 layers, neural network -> deep neural network | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | 3. Activation Function  - limit output value | | | | | | | | | | | a. step-up function (for classification problem) output 0 or 1  b. sigmoid function : output btw 0 & 1  c. hyperbolic tangent [tanh(z)] : output btw -1 & 1  d. rectified linear unit (ReLU) : if f(z) < 0, output = 0, if f(z) > 0, output = f(z) | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | 4. Multi-Class Classification Considerations  - Non-exclusive classes: data can have multiple classes assigned to it (e.g. photo with multiple tags)  - Mutually exclusive classes: 1 class per data pt (1 output node per class)  - one-hot encoding / dummy variable -> use matrix to represent diff. class | (1,0,0), (0,1,0)  - sigmoid function for non-exclusive classes (have cutoff values to determine which class) | (1,0,1)  classes independent of one another = can have multiple classes  - softmax function for mutually-exclusive classes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | 5. Cost Function & Gradient Descent  - cost / loss function is avg. of how far off predictions is from true value  quadratic cost function -> to find minimum cost/value in function  -> step-size to find min. cost = (grad = 0) / aka learning rate  - adaptive gradient descent -> larger steps initially, smaller steps as approach grad = 0, (symbol for grad) [e.g. Adam optimizer]  - cross-entropy loss function for classification problems (assume model predict prob. dist.) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | 6. BackPropagation  - partial derivative for weight  (can do for bias also)  - use grad to adjust w and b to minimise output of error vector  - Hadamard Pdt  - error vector  -backpropagate the error | | | | | | | | | | | | | | | | where L = last layer | w = weight | b = bias |  = activation function | C = cost function | x = 1st input (  where : rate of change of C wrt output activation | | | | | | | | | | | | | | | | | |
| Tensorflow | | | | | | | X = df[['feature 1', 'feature 2']].values  y = df['price'].values  from sklearn.preprocessing import MinMaxScaler  fit...transform...train test split...  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Activation | | | | | | | | | | | | | | | | | | | | | | | | | | --> tensorflow need numpy array  --> fit only for train data | transform & train test split for both train & test data | | | | | | | |
|  | | | | | | | model = Sequential([Dense(4, activation = 'relu'),  Dense(2, activation = 'relu')  OR Dense(1)])  model = Sequential()  model.add(Dense(4, activation = 'relu'))  model.add(Dense(2, activation = 'relu'))  model.add(Dense(1)) | | | | | | | | | | | | | | | | | | | | --> (no. of neurons/unit, activation function)  --> next layer  --> model: units usually decrease by half until last layer = 1 | | | | | | | | | | | | | |
| from tensorflow.keras.optimizers import Adam- can use optimizer = 'Adam', loss = 'mse' | | | | | | | model.compile(...)  model.fit(X\_train, y\_train, epochs = 250, validation\_data = (X\_test, y\_test), batch\_size = 128)  losses = pd.DataFrame(model.history.history)  losses.plot() | | | | | | | | | | | | | | | | # For a multi-class classification problem  model.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  # For a binary classification problem  model.compile(optimizer='rmsprop',  loss='binary\_crossentropy',  metrics=['accuracy'])  # For a mean squared error regression problem  model.compile(optimizer='rmsprop',  loss='mse')  --> epochs: how many time model go through data  validation data: evaluate data btw train & test; check for overfitting  batch size: Number of samples per gradient update | | | | | | | | | | | | | | | | | |
| (model.predict(X\_test) > 0.5).astype("int32") for binary classification | | | | | | | | | pred = model.predict(X\_test)  pred = pd.Series(pred.reshape(..,1))  true\_y = pd.DataFrame(y\_test, columns = ['True Y'])  pred\_df = pd.concat([true\_y, pred], axis = 1)  pred\_df.columns = ['True Y', 'Predictions']  sns.scatterplot(x='True Y',y='Predictions',data=pred\_df)  pred\_df['Error'] = pred\_df['True Y'] - pred\_df['Predictions']  sns.distplot(pred\_df['Error'],bins=50)  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score  explained\_variance\_score(y\_test,predictions) | | | | | | | | | | | | | | | | | | | | | | | | | | | | --> predict  --> reshape to (no. of data, 1)  --> change to DataFrame  --> concat  --> name columns  --> ideally straight line  --> ideally normally dist.  --> best score is 1.0 | | | |
|  | | | | | | | from tensorflow.kera.models import load\_model  model.save('model\_name.h5')  later\_model = load\_model('model\_name')  new\_data = scaler.transform(new\_data)  model.predict(new\_data) | | | | | | | | | | | | | | | | | | --> save model  --> load and use model  --> transform new data  --> predict new data | | | | | | | | | | | | | | | |
| TF Classification | | | | | | train test split...scaling...model...  model fit...model\_loss.plot... from tensorflow.keras.callbacks import EarlyStopping  early\_stop = EarlyStopping(monitor = 'val\_loss', mode = min, verbose = 1, patience = 25)  fit model again..plot... | | | | | | | | | | | | | | | | | | | --> last layer activation = 'sigmoid'  --> if overfit  --> monitor: validation loss; mode: minimise loss  patience: Number of epochs with no improvement after which training will be stopped  --> if still not aligned | | | | | | | | | | | | | | | |
|  | | from tensorflow.keras.layers import Dropout  model = Sequential()  model.add(Dense(units=30,activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(units=15,activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(units=1,activation='sigmoid'))  model.compile(loss='binary\_crossentropy', optimizer='adam')  model.fit(X\_train, y\_train, epochs=600, validation\_data=(X\_test, y\_test), verbose=1, callbacks=[early\_stop])  plot...predict...error report... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |