**Data Processing Using Python**

**Data Analysis & Exception Handling**

**Executing an array/matrix using function(the regular method)**

| # executing an array/matrix using function  import timeit # if %%timeit gives a usage error,you can import timeit  def make\_matrix(n\_rows,n\_cols):  return [list(range(n\_cols \*i , n\_cols\*(i+1))) for i in range(n\_rows)]  test\_data=make\_matrix(40,5)  #test\_data  time\_taken = timeit.timeit(lambda: make\_matrix(40, 5), number=1000)  print(time\_taken)  *# output* Output exceeds the size limit. Open the full output data in a text editor [[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, 27, 28, 29],  [30, 31, 32, 33, 34],  [35, 36, 37, 38, 39],  [40, 41, 42, 43, 44],  [45, 46, 47, 48, 49],  [50, 51, 52, 53, 54],  [55, 56, 57, 58, 59],  [60, 61, 62, 63, 64],  [65, 66, 67, 68, 69],  [70, 71, 72, 73, 74],  [75, 76, 77, 78, 79],  [80, 81, 82, 83, 84],  [85, 86, 87, 88, 89],  [90, 91, 92, 93, 94],  [95, 96, 97, 98, 99],  [100, 101, 102, 103, 104],  [105, 106, 107, 108, 109],  [110, 111, 112, 113, 114],  [115, 116, 117, 118, 119],  [120, 121, 122, 123, 124], ... [175, 176, 177, 178, 179],  [180, 181, 182, 183, 184],  [185, 186, 187, 188, 189],  [190, 191, 192, 193, 194],  [195, 196, 197, 198, 199]] |
| --- |

**Using %%timeit to calculate the execution time of your code**

%%time it is a jupyter notebook command used to calculate the execution time of the code inside one cell.

%%timeit

a = sum([sum(r) for r in test\_data])

#output

7.95 µs ± 135 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)

Here it takes much time for the execution..

So here comes using Numpy..

**Using Numpy**

*Numpy stands for Numerical Python.*  
*NumPy provides a* ***multi-dimensional array*** *object, ndarray, which is faster and more efficient than Python's built-in data structures such as lists and tuples.*  
*The* ***ndarray*** *object is used to store homogeneous data (elements of the same data type) in a fixed-size, contiguous block of memory.*  
*also provides a large collection of mathematical functions and operations that can be applied to arrays. These include mathematical operations such as* *addition, subtraction, multiplication, division, and exponentiation, as well as statistical functions,random number generation, linear algebra, Fourier analysis, and more.*

**Installing numpy first:**

pip install numpy

Re-execute the same code using Numpy

| %%timeit import numpy as np test\_np\_Array=np.array(test\_data) *#print(test\_np\_Array)* *#output* 13.6 µs ± 947 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)  summing=np.array(test\_data).sum() print(summing) *#output* 19900   arr=np.array([1,2,3,4,5]) print(arr) arr2d=np.array([[1,2,3],[4,5,6],[7,8,9]]) # be attentioned to the brackets here as the elements of the array should be put in single bracket print(arr2d) array\_zeros= np.zeros((2,3)) *# gives an array of zeros* print(array\_zeros) *#output 1 of arr* [1 2 3 4 5]  *#output 2 of arr2d* [[1 2 3]  [4 5 6]  [7 8 9]]  *#output 3 of array\_zeros* [[0. 0. 0.]  [0. 0. 0.]] |
| --- |

**Accessing specific columns and rows in the array**

| print(arr[3]) *# array scalar getting one element from numpy array* print(arr2d[0][2]) *# to access the element at row 0 and column 2 of the array* print(arr2d[0,2]) *# to access the element at row 0 and column 2 of the array; gives the same result*  *# to get specific rows and columns* Arr2d[1:,2:] # here we need from the first row and after but to begin from the third column and after    *#output 1* 4 *#output 2* 3 *#output 3* 3 *#output 4* array([[6],  [9]]) |
| --- |

**Slicing Rows/Columns**

Slicing allows you to subtract the data you need based on specific conditions.

Used in :

* Filtering the data: extract only the data needed based on specific criteria
* Data manipulation: after extracting the data filtered/needed,you can perform calculations on it
* Data visualisation: to visualize the extracted data.

#slicing the rows and columns

small\_array[1:2,0:]=0 # here it sets the second row to zero

small\_array

#slicing the rows and columns

small\_array[2:3,1: ]=5 # here it sets the third row starting from the second column to 5

Small\_array

#output 1

[[0.294665 0.53058676 0.19152079]

[0. 0. 0. ]

[0.6375209 5. 5. ]

[0.3578136 0.94568319 0.06004468]]

#output 2

[[0.294665 0.53058676 0.19152079]

[0. 0. 0. ]

[0.6375209 5. 5. ]

[0.3578136 0.94568319 0.06004468]]

**To get the size of the array (no. of rows x no. of columns)**

**To get the transpose of the array**

**To get the data type or specify the data type of the array**

| print(arr.shape) *# get the size of row* print(arr2d.shape) *# get the size of row and columns* print(arr.dtype) *# get the data type*  *# to specify the data type in the matrix*  *acomplex=np.array([1,2,3],dtype="complex")*  *print(acomplex)*  #output 1  (5,)  #output 2 (3, 3)  #output 3 Int32  #output 4; specifying the data type to be complex  [1.+0.j 2.+0.j 3.+0.j]  *# these code is for doing numpy transpose of matrix* arr2d\_transpose=np.transpose(arr2d) print(arr2d\_transpose) *# this also works, it is more like mathematical notation* arr2d\_transpose = arr2d.T *# option 2* print(arr2d\_transpose)  #output 1  [[1 4 7]  [2 5 8]  [3 6 9]]  #output 2; it gives the same result [[1 4 7]  [2 5 8]  [3 6 9]] |
| --- |

**Giving the range of numbers to be inserted in the matrix**

**using linspace() and arange() func.**

**Using random.seed()**

**linspace():**

*-giving the range of number inserted in the matrix using linspace(); start point,end point and the increment*

*-the end point here is included which is different here from using range()*

**arange():**

*-here it is like range() as the end point is not included*

| array=np.linspace(0,12,4,dtype=int) print(array) *#output 1* [ 0 4 8 12]  *# here endpoint is not included* array=np.arange(0,11,2) print(array) *#output 2* [ 0 2 4 6 8 10] |
| --- |

**Random.seed():**

*-used to seed the random number generator.*  
*# numpy.random.seed(seed=None)*  
*-seed is an optional integer value that is used to* ***initialise the random number generator****-By* ***setting the same seed value****, you can ensure that the random number generator produces* ***the same sequence of random numbers every time the code is run.****-This can be useful for testing and debugging, as it allows you to reproduce the same results.*

| np.random.seed(0) *# set random set for consistency* array\_random=np.random.rand(5,2) print(array\_random)  *# getting random integers specifying the number of random integers to be generated* np.random.seed(42) x = np.random.randint(0, 10, size=5) print(x) *#output 1; specifying the size of the array* [[0.5488135 0.71518937]  [0.60276338 0.54488318]  [0.4236548 0.64589411]  [0.43758721 0.891773 ]  [0.96366276 0.38344152]]  *#output 2; specifying the data type to be int. And no. of integers are 5* [6 3 7 4 6] |
| --- |

**Filtering the array based on specific conditions**

**Using boolean indexing**

| *# filtering the array using the logical operators and boolean indexing* x[:] >3 *#output* array([ True, False, True, True, True])  filtering=np.where(x>4,x,0) *# to filter arrays* print(filtering) *#output*  [6 0 7 0 6] |
| --- |

**Sorting the Array or the data set /Reshaping the data set using .reshape()**

*using reshape() if we have huge data set in an excel file in one row*, *so we divide it/simplify it by using the reshape func. to form a matrix*

| np.random.seed(17) small\_array=np.random.rand(12) *# here the size is specified to be 12x0 ; 12 rows only* print(small\_array) print(small\_array.shape) small\_array= small\_array.reshape(4,3) *# but now we want to reshape it with different size to make it easy to read* small\_array *#output 1* [0.294665 0.53058676 0.19152079 0.06790036 0.78698546 0.65633352  0.6375209 0.57560289 0.03906292 0.3578136 0.94568319 0.06004468] *#output 2* (12,)  *#output 3* array([[0.294665 , 0.53058676, 0.19152079],  [0.06790036, 0.78698546, 0.65633352],  [0.6375209 , 0.57560289, 0.03906292],  [0.3578136 , 0.94568319, 0.06004468]]) |
| --- |

**Before getting into data manipulation, we need first to have a copy of the original array in case we need it**

**Copying an Array**

| small\_array\_copy=small\_array.copy() Small\_array\_copy *#output* array([[0.294665 , 0.53058676, 0.19152079],  [0.06790036, 0.78698546, 0.65633352],  [0.6375209 , 0.57560289, 0.03906292],  [0.3578136 , 0.94568319, 0.06004468]])  *#the difference between fake and real copy:* *# creating a copy vs assign new name* my\_arr = np.arange(3) my\_arr\_not\_copy = my\_arr *# here it changes whenever there is any change in the array; called a shallow copy* my\_arr\_copy = my\_arr.copy() *# here it maintain the original copy however changes made to the original one; called a deep copy*  print(my\_arr, my\_arr\_not\_copy, my\_arr\_copy) *#my\_arr\_not\_copy[0] = 10* *#print(my\_arr, my\_arr\_not\_copy, my\_arr\_copy)* my\_arr[:]=5 print(my\_arr, my\_arr\_not\_copy, my\_arr\_copy)  *#output* [0 1 2] [0 1 2] [0 1 2] [5 5 5] [5 5 5] [0 1 2] # only the copy() func remains the same |
| --- |

**Saving an array in .npy file , .csv file as text**

**To retrieve the data of the array or the array itsef from a file**

| *# saving and retrieving an array in a .npy file* import numpy as np np.save('test.npy',array1) *# it will be saved in a file named: test.npy* *#to retrieve the array saved* retrive\_array=np.load('test.npy') Retrive\_array *#output*  array([[0.22199317, 0.87073231, 0.20671916, 0.91861091, 0.48841119],  [0.61174386, 0.76590786, 0.51841799, 0.2968005 , 0.18772123],  [0.08074127, 0.7384403 , 0.44130922, 0.15830987, 0.87993703],  [0.27408646, 0.41423502, 0.29607993, 0.62878791, 0.57983781]])  *# Saving and Retrieving as a text* *# to save the data of the array in a text file but .npy* np.savetxt('test.txt.npy',array1,delimiter=',',header='array1') *# the delimiter is used to separate between the data of the array*  *# to save as a text in .csv with header and datatype=float %f* np.savetxt('test.txt.csv',array1,fmt='%f',delimiter=',',header='Array1')   *# to retrieve the array which is saved in the text file .csv* retrivefromcsv=np.loadtxt('test.txt.csv',delimiter=',') retrivefromcsv *#output*  array([[0.22199317, 0.87073231, 0.20671916, 0.91861091, 0.48841119],  [0.61174386, 0.76590786, 0.51841799, 0.2968005 , 0.18772123],  [0.08074127, 0.7384403 , 0.44130922, 0.15830987, 0.87993703],  [0.27408646, 0.41423502, 0.29607993, 0.62878791, 0.57983781]])  *# save an array with int Data type and header* x = np.array([3,4,5]) np.savetxt("my\_data", x, fmt="%d", header="My numbers")   *# to make sure the file is saved* %ls -l \*.npy *#output* Volume in drive C is Windows-SSD  Volume Serial Number is 1854-1986   Directory of c:\Users\rovan\OneDrive\Desktop\WileyEdge\Wiley's Training\Python    Directory of c:\Users\rovan\OneDrive\Desktop\WileyEdge\Wiley's Training\Python  13/03/2023 15:10 288 test.npy 13/03/2023 15:14 504 test.txt.npy  2 File(s) 792 bytes  0 Dir(s) 425.592.500.224 bytes free |
| --- |

**Saving Objects of a class in a numpy file**

| # saving one object  class Myclass:  def \_\_init\_\_(self,value1,value2):  self.value1=value1  self.value2=value2 obj=Myclass(42,'rffe') *# here we want to save the object in numpy file so we have to convert it to array first* obj1=Myclass(50,'ds') np.array(obj,dtype=object) *#optional step: converting it to array* np.save('abc.npy',obj) *# saving it in numpy file* loaded\_obj=np.load('abc.npy',allow\_pickle=True) *# retrieving the data using allow\_pickle to be able to load the data from the file* loaded\_obj=loaded\_obj.item() *# we convert the array to object again so we can see the data using .item()* print(loaded\_obj.value1) *# the normal way of retrieving the attribute from an object/instance* loaded\_obj.value2 Loaded\_obj.\_\_dict\_\_ *#output* 42  {'value1': 42, 'value2': 'rffe'} |
| --- |

#jinesh's way to store more than one object

| class Myclass:  def \_\_init\_\_(self,value1,valye2):  self.value=value1  self.value2=valye2 obj1= Myclass(42,'asdhkasd') obj2=Myclass(123,'dsjkhkjhsdkhdaskjhd') obj3=Myclass(2123,'d422sjkhkjhsdkhdaskjhd') np.savez('mycustom.npz',obj1=obj1,obj2=obj2,obj3=obj3)  loaded\_temp\_obj=np.load('mycustom.npz',allow\_pickle=True) loaded\_obj1=loaded\_temp\_obj['obj2'].item() print(loaded\_obj1.\_\_dict\_\_) *#output* {'value': 123, 'value2': 'dsjkhkjhsdkhdaskjhd'} |
| --- |

**My Research/Work**

| *# to store more than one object* class Myclass:  def \_\_init\_\_(self,value1,value2):  self.value1=value1  self.value2=value2 obj=Myclass(42,'rffe') *# here we want to save the object in numpy file so we have to convert it to array first* obj1=Myclass(50,'ds') objects={'obj':obj,'obj1':obj1} *# create a dictionary* *#np.array(objects,dtype=object) # converting it to array* np.save('abc.npy',objects) *# saving it in numpy file* loaded\_obj=np.load('abc.npy',allow\_pickle=True) *# retrieving the data using allow\_pickle to be able to load the data from the file* loaded\_obj=loaded\_obj.item() *# we cnvert the array to object again so we can see the data using .item()* print(loaded\_obj['obj'].\_\_dict\_\_) print(loaded\_obj['obj1'].\_\_dict\_\_) *#output* {'value1': 42, 'value2': 'rffe'} {'value1': 50, 'value2': 'ds'} |
| --- |

**Mathematical operations used on the numerical arrays**

Summing,subtracting,multiplication,dot,sqrt,log,log2,log10,mean.std,cos,sin,tan,max,min,argmin(),argmax(),etc..

#print(small\_array\_copy)

print(np.sin(small\_array\_copy))

np.tan(small\_array\_copy)

np.cos(small\_array\_copy)

np.log(small\_array\_copy)

np.log2(small\_array\_copy)

np.log10(small\_array\_copy)

np.sqrt(small\_array\_copy)

#multiplication

np.random.seed(5)

array1=np.random.rand(4,5)

array2=np.random.rand(4,5)

print(array1);print(array2)

multiplication=array1\*array2

multiplication # here it will give the multiplication in the same array size; it multiplies each element with its spouse

#subtraction

subtr=array1-array2

subtr # it subtracts each element with each spouse

# dot

#dot=np.dot(array1,array2) # here it will give error because one of them needs to be transposed

dot=np.dot(array1,array2.T)

dot # here array1 size is 4x5 and array2 size is 5x4 so the result will be 4x4 array size

# maximum and minimum value in the array

# to get the max and min value in the array

maximum=np.max(array1)

print(maximum)

minimum=np.min(array1)

minimum

# argmin , argmax giving indexes

# to get the min and max value of the array using indexing

minimum\_indexing=array1.argmin()

print(minimum\_indexing)

maximum\_indexing=array1.argmax()

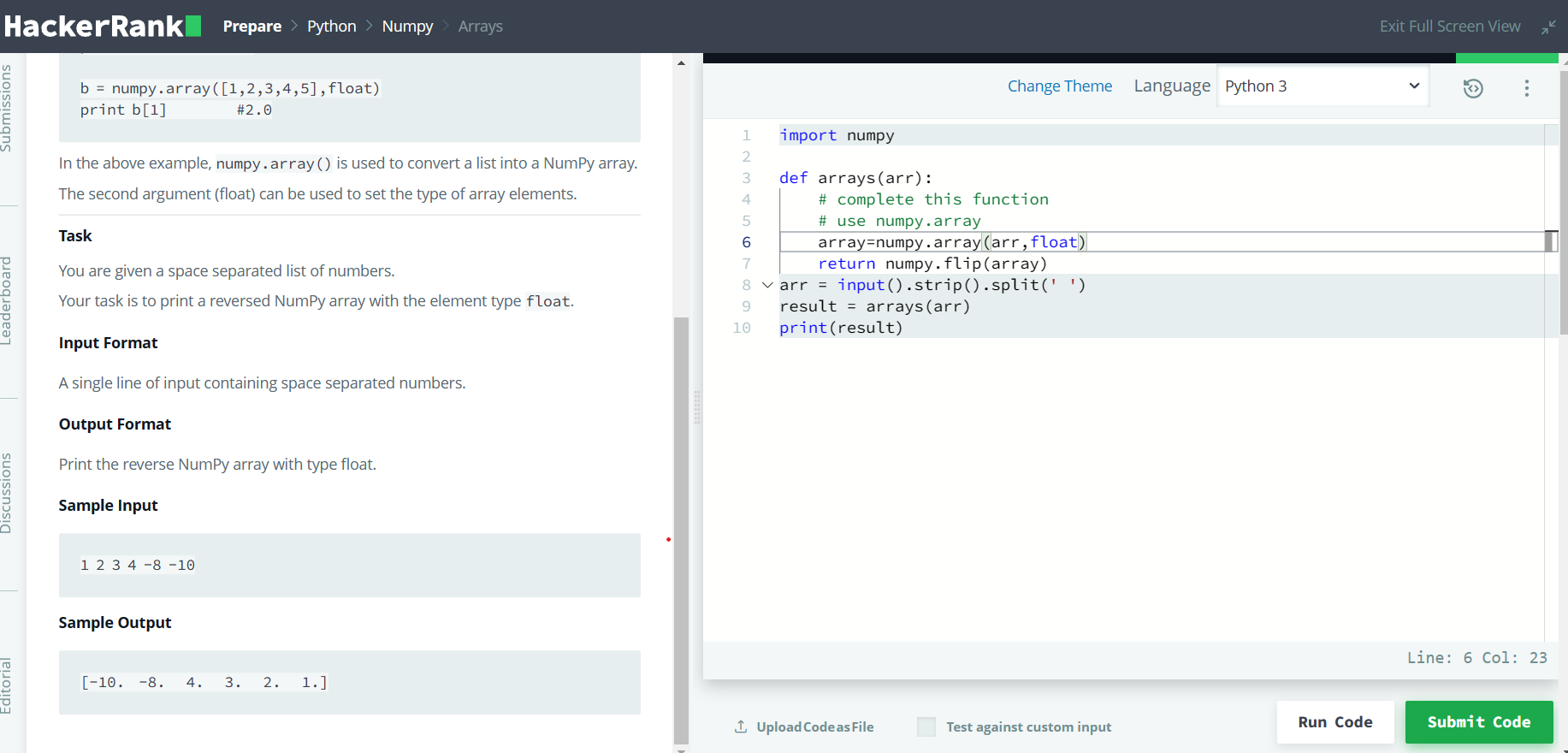
print(maximum\_indexing)

array1[:,0].argmin() # to get the min value in specific column or row

**My Research/work/practice**

**HackerRank**

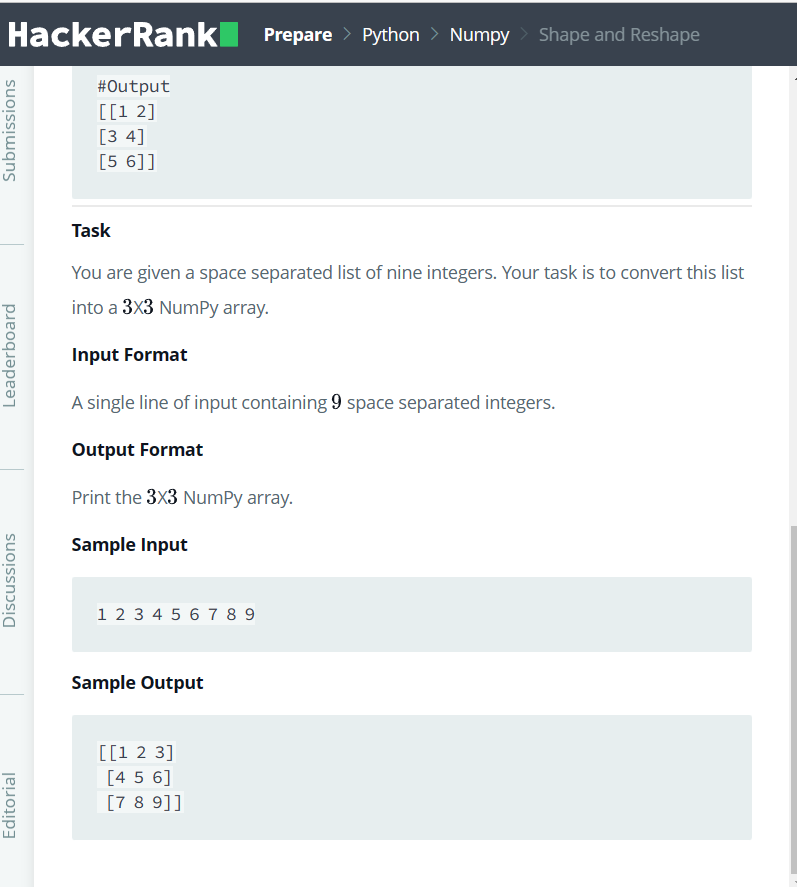
**Task: Reversing an array**

****

**Code: Using Numpy and .flip() func.**

| **import numpy  def arrays(arr):  *# complete this function*  *# use numpy.array*  array=numpy.array(arr,float)  return numpy.flip(array) arr = input().strip().split(' ') result = arrays(arr) print(result)** |
| --- |

**Practice 2: .shape() & .reshape()**

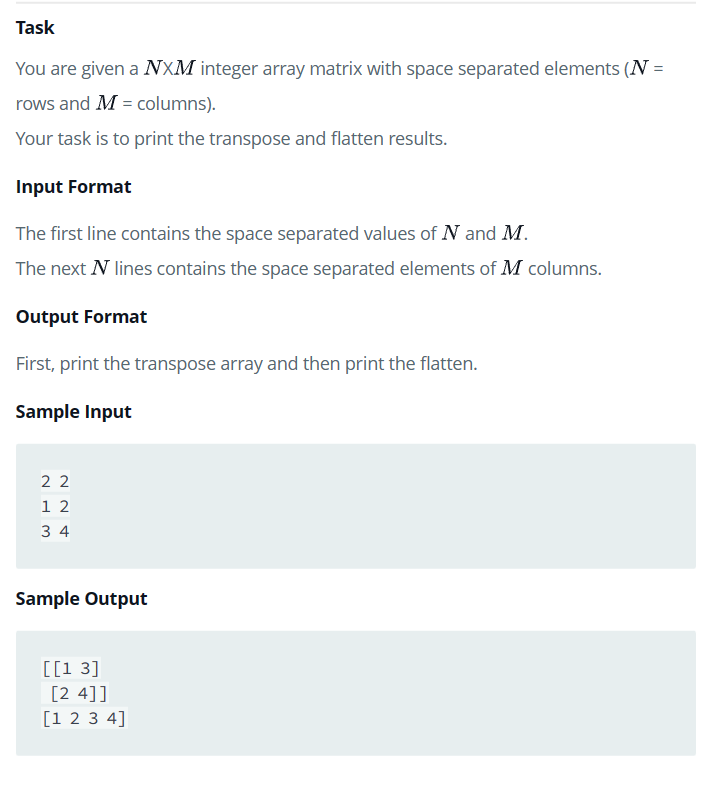
****

**Code:**

| import numpy arr=numpy.array([input().strip().split(' ')],int) *#print(arr)* reshaped\_array=arr.reshape(3,3) print(reshaped\_array) |
| --- |

**Practice 3: using Transpose() & .flatten()**

**flatten() func: is used to flatten the array to one dimension only.**

****

**Code:**

| **import numpy size=input().strip().split(' ') int\_list = [int(x) for x in size]  *#print(size)* ar = [] for i in range(int\_list[0]): # adding each row in a list in the ar list depending on the size input  row = list(input().strip().split(' '))  ar.append(row) *#print(ar)* arr=numpy.array(ar,int) arr.shape=(int\_list[0],int\_list[1]) transposing=arr.T flattening=arr.flatten() print(transposing) print(flattening) *# flatten the array to one dimension*** |
| --- |

**Pandas Library**

Pandas is used for tabular data like columns in spreadsheet or a SQL table

*PANDAS vs Numpy*  
*NumPy is mainly used for* ***numerical and scientific computations****. It provides a powerful* ***N-dimensional array*** *object,tools for integrating C/C++ and Fortran code, and functions for linear algebra, Fourier analysis, and random number generation.*

*NumPy arrays can only store elements of* ***the same data type****. If you try to create a NumPy array with elements of different data types, NumPy will automatically* ***convert all the elements to a single data type.***  
  
*Pandas, on the other hand, is built on top of NumPy and provides high-level* ***data manipulation*** *tools for working with structured data.*  
*It provides a* ***DataFrame object for handling tabular data****, as well as tools for reading and writing data from/to various data sources,data alignment, reshaping, slicing, indexing, grouping, and aggregating data, and handling missing data.*  
  
*In summary, NumPy is more focused on numerical computations, while Pandas is more focused on data manipulation and analysis.*

*Pandas can store non-homogeneous or mixed data, while NumPy cannot.*

| .pip install pandas |
| --- |

Creating Series of data

| import pandas as pd s=pd.Series([1,2.2,3,9.35]) S *#output* 0 1.00 1 2.20 2 3.00 3 9.35 dtype: float64  *# renaming the indexes* s=pd.Series([1,2.2,3,9.35],index=['a','b','c','d']) s *#output* a 1.00 b 2.20 c 3.00 d 9.35 dtype: float64 |
| --- |

Accessing data in Pandas Series

| *# accessing data using indexes* S['a']  #output  1.0  print(s.index)  print(s['a':'c'])  #output 1  Index(['a', 'b', 'c', 'd'], dtype='object')  #output 2  a 1.0  b 2.2  c 3.0  dtype: float64 |
| --- |

**Mathematical Operations between two tabular data sets using indexes**

| *# summin two tabular data by the indexes* import pandas as pd s1=pd.Series([1,2,3],index=['a','b','c']) s2=pd.Series([4,5,6],index=['a','b','d']) print(s1+s2) *# here it shows the uncommon indexes result is NaN ( not a number);represents undefined data or value/when data is missing* s1.add(s2,fill\_value=0)  *#output* a 5.0 b 7.0 c NaN d NaN dtype: float64  a 5.0 b 7.0 c 3.0 d 6.0 dtype: float64 |
| --- |

**Using Lambda**

| *#using Lambda to call the function immediately instead of writing def functions* *#lambda is a keyword that is used to define a small, anonymous function.* *# It is also known as an anonymous function or a lambda function. It is a way to create a function without a name and can be defined in a single line of code.* *# the Lambda syntax: lambda arguments: expression* *# Lambda can be used to filter,map or reduce a list of data in short period of time*   b=lambda x,y:x+y *# anonymous functions* result=b(6,5) print(result) *#output* 11   product=lambda x,y:x\*y result=product(3,2) print(result) *#output* 6  *#putting the lambda and the input in one line* print((lambda x:x if(x>10) else 10)(5)) *#output* 10   Trickyyyyyyyyyy print((lambda x:x\*10 if x>10 else (x\*5 if x<5 else x))(6)) *#output* 6 |
| --- |

**My Research/work**

**Filtering the data**

| *# filtering a data* *# Define a list of numbers* numbers = [1, 2, 3, 4, 5]  *# Use filter() and a lambda function to filter out the even numbers from the list* evens = list(filter(lambda x: x % 2 == 0, numbers)) *# it should be put in a list or tuple to be able to see the updated dataset*  *# Print the even numbers* print(evens) *#output* [2, 4] |
| --- |

**Using map()**

| using map() to perform operation on each value of the data set *# Define a list of numbers* numbers = [1, 2, 3, 4, 5]  *# Use map() and a lambda function to square each number in the list* squared = tuple(map(lambda x: x \*\* 2, numbers)) sq=list(map(lambda x: {x: 'a'} if(x%2 ==0) else x, numbers)) *# Print the squared list* print(squared) print(sq) *#output 1* (1, 4, 9, 16, 25)  *#output 2* [1, {2: 'a'}, 3, {4: 'a'}, 5] |
| --- |

**Using reduce()**

| *# using reduce()* from functools import reduce  *# Define a list of numbers* numbers = [1, 2, 3, 4, 5] *#print(numbers[ :2]) # gives [1,2]* *# Use reduce() and a lambda function to calculate the product of the numbers in the list* product = reduce(lambda x, y: x \* y, numbers) *# here we don;t need to put it in a list or tuple as it is reduced to one value*  *# Print the product* print(product) *#output* 120 |
| --- |

**Digital Ethics**

As a data analyst, digital ethics is an important consideration because data analysts are responsible for collecting, analyzing, and interpreting data that may have ethical implications.For example, when collecting data, data analysts must ensure that they are doing so in an ethical and legal manner, and that they are obtaining informed consent from participants when appropriate. Additionally, data analysts must ensure that the data they are collecting is relevant to the research question and that it is not biased in any way.  
When analyzing data, data analysts must be aware of the potential for bias and take steps to minimize it. They must also ensure that their analyses are accurate and that their conclusions are based on sound statistical principles.  
Finally, data analysts must be aware of the potential ethical implications of their work and consider the broader impact of their findings on society. They must be mindful of the potential consequences of their work and take steps to ensure that their analyses do not harm individuals or groups.  
In short, data analysts must be aware of the ethical implications of their work and take steps to ensure that they are conducting their work in an ethical and responsible manner.

**Filtering a tabular dataset using Pandas**

We have a dataset talking about the college score card in States in a csv file.

Consists of 10 columns: OPEID,INSTNM,CITY,STABBR,ZIP,INSTURL,REGION,LOCALE,ADM\_RATE,COSTT4\_A

| data=pd.read\_csv('from Jinesh\college\_scorecard\_2017-18.csv',dtype={'OPEID':str}) Data *# IT WILL SHOW THE DATASET TABLE*  *#sorting the data based on the city* data\_sorted = data.sort\_values(by='CITY', ascending=True) data\_sorted *# IT WILL SHOW THE DATASET ORDERED BY THE CITY alphabetic* |
| --- |



Accessing data



| *#if you want to show two columns data* data[['CITY','INSTNM']].head() *#Geting the unique data* unique\_=data['CITY'].unique() Unique\_ *#output* array(['Normal', 'Birmingham', 'Montgomery', ..., 'Cedar Hill',  'Lewis Center', 'Ivins'], dtype=object)  #accessing data based on two conditions  data[(data['STATE'] == 'TX') & (data['CITY'] == 'McAllen')]  #output |
| --- |

data\_renaming.query('STATE == "TX" and CITY =="McAllen"')

#the same output using .query()



Filtering the data

| *#filtering using loc and iloc(for indexing look up)* *# loc is used to access or modify DataFrame values by providing row and column labels as input. It takes two arguments:*  *#The first argument is the row labels or indices you want to select.* *#The second argument is the column labels or names you want to select.* *# loc can also be used to modify values in a DataFrame, not just access them.* *#allows you to access or modify specific rows and columns by integer-based indexing. iloc stands for "integer location".*   *#iloc is used to access or modify DataFrame values by providing row and column positions as input. It takes two arguments:*  *#The first argument is the row positions or indices you want to select.* *#The second argument is the column positions or indices you want to select.* *# Note that : iloc is different from loc in that it uses integer positions instead of label names.* filtering=data.loc[3,'CITY'] Filtering *#output* 'Huntsville'  #data.loc[3,3] # here it gives error as loc is not for indexing look up |
| --- |

| filtering=data.loc[3] *# to get the data for the fourth row for all columns* Filtering *#output* OPEID 00105500 INSTNM University of Alabama in Huntsville CITY Huntsville STABBR AL ZIP 35899 INSTURL www.uah.edu REGION 5 LOCALE 12.0 ADM\_RATE 0.8123 COSTT4\_A 22108.0 Name: 3, dtype: object |
| --- |

data.loc[data['CITY']=='Normal']



#loc enables you to modify the data as well

#data.loc[data['CITY'] == 'Normal'] = 'Germany'

data.loc[data['CITY'] == 'Germany']= 'Normal'

filtered\_df = df\_sorted.loc[df\_sorted['CITY'] =='Aberdeen']

filtered\_df

# Save the filtered data to a new file

filtered\_df.to\_csv('from Jinesh/filtered\_data.csv', index=False)

| *# filter data rows and columns* Accessing=data.loc[[3,10],['INSTNM','ZIP']] Accessing *# to access the row 3 and row 10 with columns INSTNM & ZIP*   *# iloc is for accessing rows and columns based on indexing* *# first arg: row index second arg" column index* data.iloc[3,3] *#output* 'AL' |
| --- |

Setting new index range for the dataset

| *# setting new range of index labels that instead of the raw starts from zero,it will start from 1 which makes more sense and also the excel files starting from 1 not zero* new\_index=pd.RangeIndex(1,len(data)+1) new\_index\_data=data.set\_index(new\_index).head() New\_index *#output* RangeIndex(start=1, stop=7059, step=1)  New\_index\_data *#output* |
| --- |

| new\_index\_data.loc[3, :].head() OPEID 02503400 INSTNM Amridge University CITY Montgomery STABBR AL ZIP 36117-3553 Name: 3, dtype: object  data.loc[3, :].head() *# here it will show with the old row label* OPEID 00105500 INSTNM University of Alabama in Huntsville CITY Huntsville STABBR AL ZIP 35899 Name: 3, dtype: object  new\_index\_data.iloc[3, :].head() *# here it will not change as iloc depends on the indexing not labels* OPEID 00105500 INSTNM University of Alabama in Huntsville CITY Huntsville STABBR AL ZIP 35899 Name: 4, dtype: object |
| --- |

# setting a column name to be the index

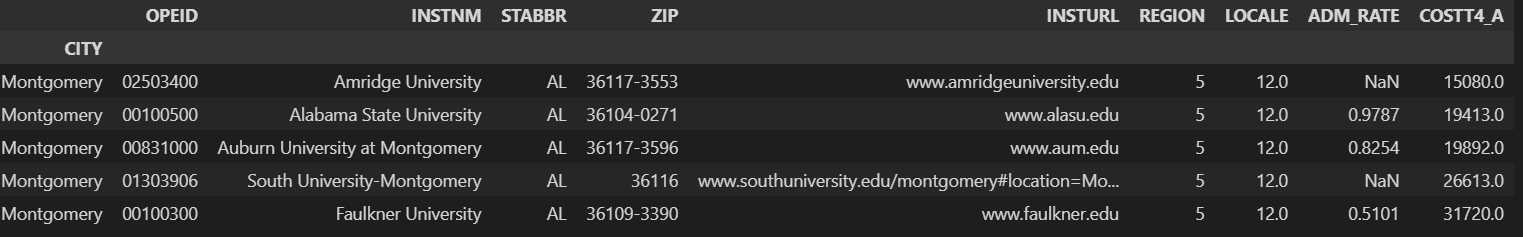
data\_city\_index=data.set\_index('CITY')

data\_city\_index.head()



data\_city\_index.loc[data\_city\_index.index=='Montgomery'].head()

#accessing the data using city name as an index



data.set\_index(['STATE','CITY']).loc['MN','Alexandria'] # setting the state and the city to be indexes labels which we can access from using loc

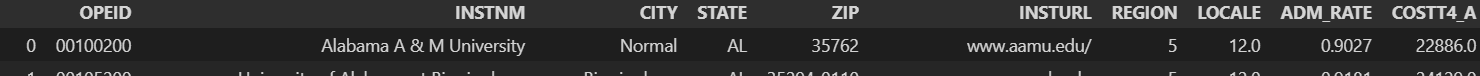


Dropping the index

data = data.reset\_index(drop=True)

Renaming columns

| data\_renaming = data.rename(columns={'STABBR':'STATE'})  data\_renaming |
| --- |



Drop Columns

# axis =0 or axis='index : means row

# axis=1 or axis='columns' : means column

# here we specify the type of the data we want to delete. is it column or row? using axis

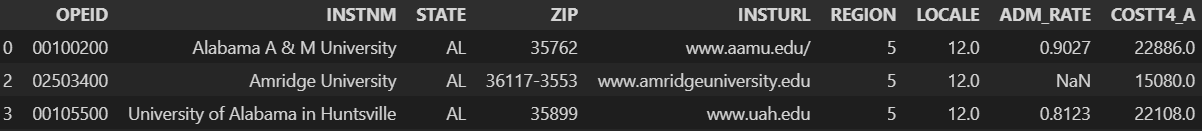
data\_new=data\_new.drop('CITY',axis='columns').head() # here it will drop the column in the copy only not in the original data

Data\_new # no CITY column here



data\_new = data\_new.drop(index=1)

data\_new.head() # here it will delete the row number 1



To show the details of the dataset

| *# to make sure that it has been dropped in the copied data* *# many ways to ensure that this column has been deleted* *# to show the dataset details* 'CITY' in data\_new *#output*  False data\_new.describe(include='all') *#output*  data\_new.info()  *#output*  <class 'pandas.core.frame.DataFrame'>  Int64Index: 4 entries, 0 to 4  Data columns (total 9 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 OPEID 4 non-null object  1 INSTNM 4 non-null object  2 STATE 4 non-null object  3 ZIP 4 non-null object  4 INSTURL 4 non-null object  5 REGION 4 non-null int64  6 LOCALE 4 non-null float64  7 ADM\_RATE 3 non-null float64  8 COSTT4\_A 4 non-null float64  dtypes: float64(3), int64(1), object(5)  memory usage: 320.0+ bytes  Data\_new.columns  *#output*  Index(['OPEID', 'INSTNM', 'STATE', 'ZIP', 'INSTURL', 'REGION', 'LOCALE',  'ADM\_RATE', 'COSTT4\_A'],  dtype='object') |
| --- |

Data Cleaning

*how to know from a .describe() function that the data needs cleaning?*  
  
*The describe() function can give some indications that the data needs cleaning. Here are a few things to look out for:*  
  
*-* ***Missing values:*** *If the count for a particular column is less than the number of rows in the dataset, then there are missing values that need to be dealt with.*  
  
*-* ***Outliers****: Look at the mean, standard deviation, and quartiles for each numerical column. If the values are too high or too low, or if there are huge differences between the mean and median, then there might be outliers that need to be examined.*  
  
*-* ***Unexpected values:*** *Check the minimum and maximum values for each numerical column. If there are values that seem impossible (e.g. negative age, or weight over 1000 kg), then the data might need cleaning.*  
  
*-* ***Inconsistent data types(using .info()):*** *Check the data types of each column. If there are columns that are supposed to be numerical but are described as "object" or "string", or vice versa, then the data might need cleaning.*  
  
*-* ***Duplicates:*** *Check the count for each column. If there are columns that have a low count, but are supposed to be unique, then there might be duplicates that need to be removed.*  
  
*#It's important to note that these indications don't necessarily mean that the data is dirty or unusable, but they do*   
suggest that the data needs further examination and potentially cleaning.

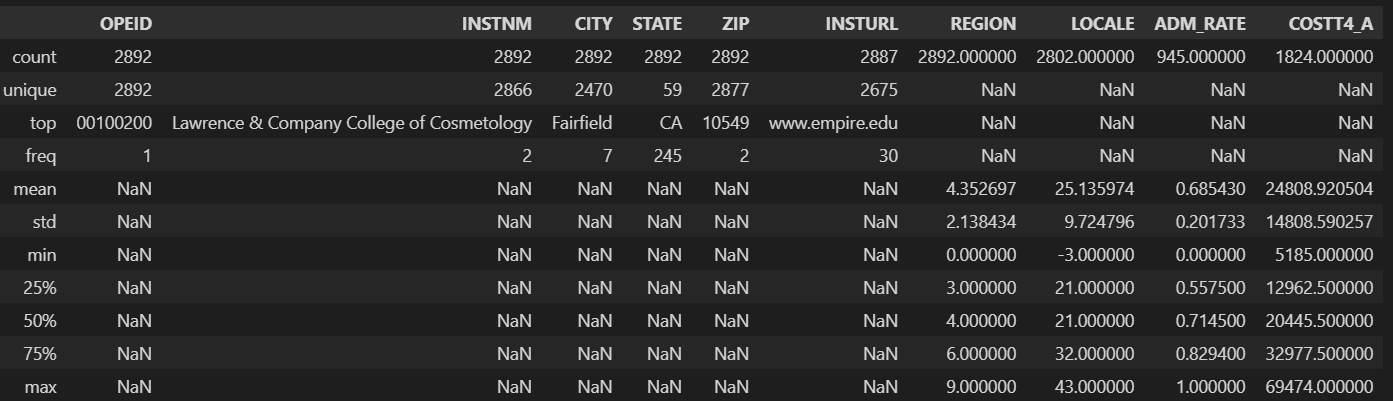
To Remove/drop Duplicates

# Remove the duplicate rows based on all columns

data\_dup=data\_renaming.drop\_duplicates(subset=['STATE','CITY']) # remove the duplicate in a seperate file or use inplace= parameter

data\_dup.describe(include='all')

#output



**#notice here the count now is 2892 instead of 7058 rows!!**

print('Deduplicated number of rows:', len(data\_dup))

#output

Deduplicated number of rows: 2892

To check the NaN values.the number of NaN values

print(data['INSTURL'].isna().sum()) # This will print the number of NaN values in a specific column.

#output

19

print(data.isna().sum()) # to see the number of NaN values in all the table columns

#output

OPEID 0

INSTNM 0

CITY 0

STATE 0

zipcode 0

INSTURL 19

REGION 0

LOCALE 444

ADM\_RATE 5039

COSTT4\_A 3486

dtype: int64

**To join two datasets based on common columns**

File2: AGI\_zipcode; adjusted gross income based on zipcodes of tax payers

Target: we want to join this dataset with the file we are on now based on the common columns of zipcodes and AGI

| *# reading a file named AGI\_zipcode..* import pandas as pd  *# read Excel file* data\_ZipCode = pd.read\_excel('from Jinesh\AGI\_zipcode\_2016.xlsx',dtype={'zipcode':str}) data\_ZipCode |
| --- |

Using left join based on the zip codes only

| *# we need to join both files using the ZIP* *#this code performs a left join between two DataFrames, data and data\_econ* data\_merged=data.merge(data\_ZipCode,left\_on='ZIP',right\_on='zipcode', how='left') *# it is like left join in SQL* data\_merged[['INSTNM','ZIP','AGI']].head(11) Data\_merged *#output* |
| --- |

**My Research/Work**

Inner join based on two common columns; zipcode and state

Here in our file column’s name is ZIP and in the AGI-zipcode column’s name is zipcode

So we have to rename one of them to match the other

data\_renaming= data\_renaming.rename(columns={'ZIP':'zipcode'})

# to merge two columns in common ; but make sure that the names are the same either use rename() to change one of them

merged\_data = pd.merge(data\_ZipCode, data\_renaming, on=['STATE', 'zipcode'], how='inner')

merged\_data

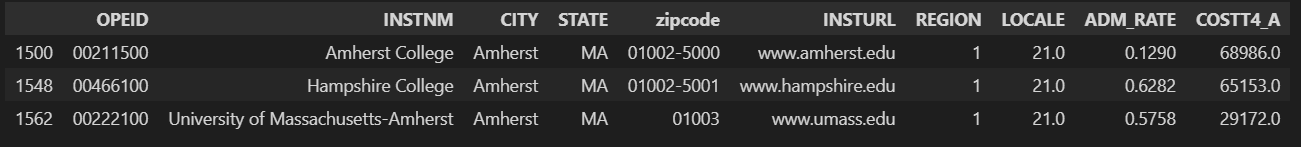
#output



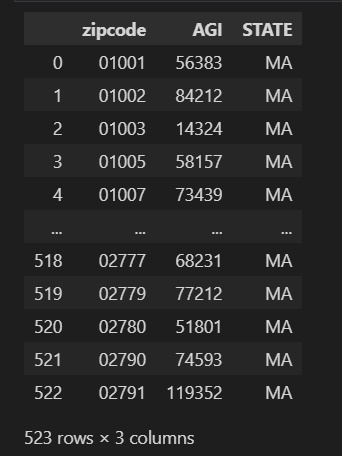
# to make sure that both are matched correctly

# check for one row with CITY/STATE in the first dataset and the second dataset

data[(data['STATE']=='MA') & (data['CITY']=='Amherst')]



data\_ZipCode[data\_ZipCode['STATE']=='MA']



**Quiz Notes**

1- The os module in Python provides a way of using operating system dependent functionality like reading or writing to the file system, creating and managing processes, and so on. It provides a portable way of using operating system dependent functionality like reading or writing to the file system, creating and managing processes, and so on. Some of the functions of the os module include file and directory handling, process management, environment variables, and file permissions.

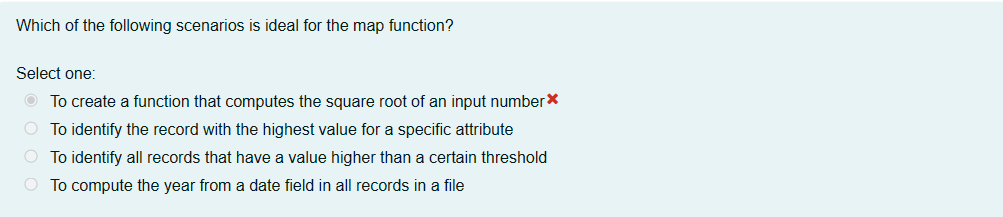
2- tzinfo attribute is used to provide information about the datetime used in the object

Ex:

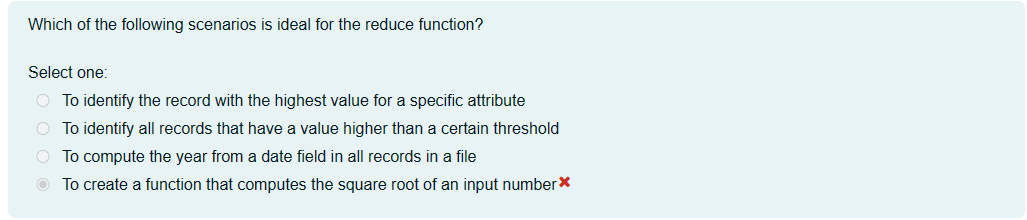
| import datetime import pytz  *# create a datetime object with timezone information* dt = datetime.datetime(2022, 3, 16, 12, 0, 0, tzinfo=pytz.UTC)  #dt = datetime.datetime(2022, 3, 16, 12, 0, 0, tzinfo=pytz.timezone('US/Eastern'))  *# print the datetime object with timezone information* print(dt)  *# convert the datetime object to a different timezone* new\_timezone = pytz.timezone('CET') dt\_new = dt.astimezone(new\_timezone)  *# print the new datetime object with the new timezone information* print(dt\_new) *#output* 2022-03-16 12:00:00+00:00 2022-03-16 13:00:00+01:00 |
| --- |

3- ‘a’ flag (stands for append flag) is used when opening python files that when writing any new content to the file ,it is added to the end of the file;without overwriting on the existed content

‘W’ flag to write a new content to the file with overwriting the previous content means that the previous content will be deleted and replaced with the new one written.

4- 

???

5- 

I think it is .. to identify the record with the highest value for a specific attribute

Example we had earlier:

# using reduce()

from functools import reduce

# Define a list of numbers

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

#print(numbers[ :2]) # gives [1,2]

# Use reduce() and a lambda function to calculate the product of the numbers in the list

product = reduce(lambda x, y: x \* y, numbers) # here we don;t need to put it in a list or tuple as it is reduced to one value

# Print the product

print(product)

# 120

6- we include multiple exception handling in the code if we have customised errors to let the python choose which is the best way to handle the code according to the exceptions given

Ex:

| Try:  num1 = int(input("Enter a number: "))  num2 = int(input("Enter another number: "))  result = num1 / num2  print("The result is:", result) except ZeroDivisionError: # if num2 was zero  print("Error: division by zero") except ValueError: # if num1 or num2 is anything other than int or float  print("Error: invalid input") Except: # anything else  print("An error occurred") |
| --- |
|  |

**Plots/Charts for data visualisation**

Types of plots & uses:

* **Line plot :** displays the data points in straight line between continuous variables ( such as time )

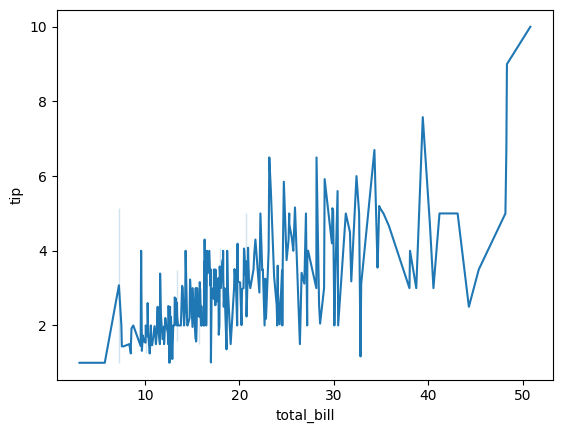
X axis: any cont. Variable like time

Y axis: the variable being measured

Efficient in small data points for easy identification

Uses: finance,economics where the time series data is analysed

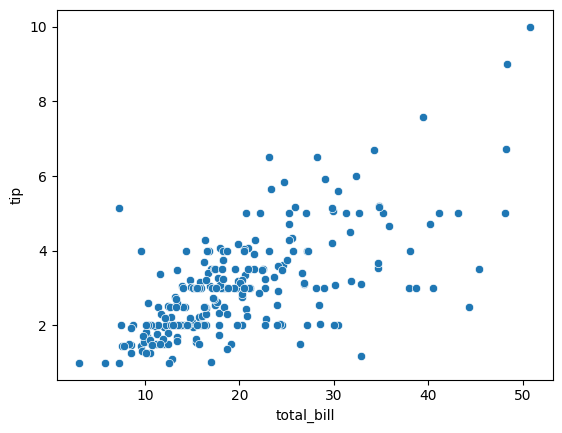
| sns.lineplot(x='total\_bill',y='tip',data=tips\_data) plt.show() |
| --- |



* **Scatter plot:** to find the relationship between two different variable and see if they are correlated or not

Examples: the relationship between height and weight, ice cream and temperature , study time and grades

| ***#scatterplot: shows relation between two continious variables; outliers and correlations can be detected using scatter plot* sns.scatterplot(x='total\_bill',y='tip',data=tips\_data) plt.show()** |
| --- |

****

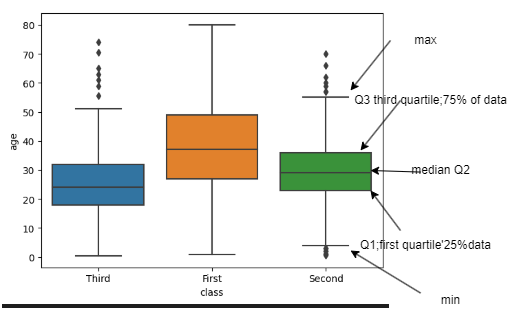
* **Count plot(Categorical Data):**

| ***#countplot: shows the number of occurences in each category in specific column* *# This plot is useful for quickly visualizing the distribution of categorical data in a dataset* sns.countplot(x='sex',data=tips\_data) *#The 'x' parameter specifies that the plot should display the count of each category in the 'sex' column of the 'tips\_data' dataset.* plt.show()** |
| --- |

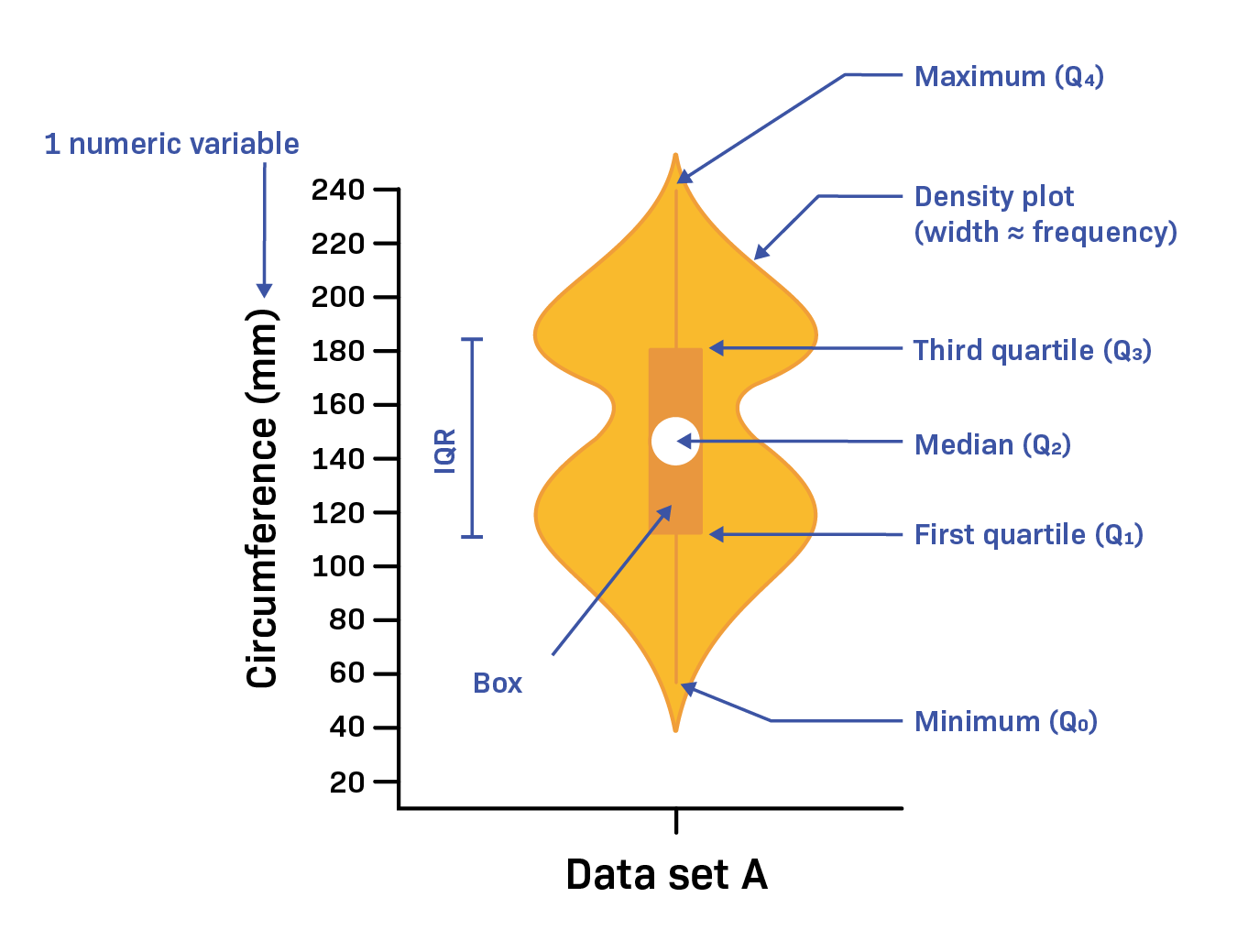
* **Pie chart (categorical data)** :
* **Box plot:**

| ***#try bar blot* titanic=pd.read\_csv("from Jinesh\seaborn-data-master\seaborn-data-master\\titanic.csv") Titanic** |
| --- |

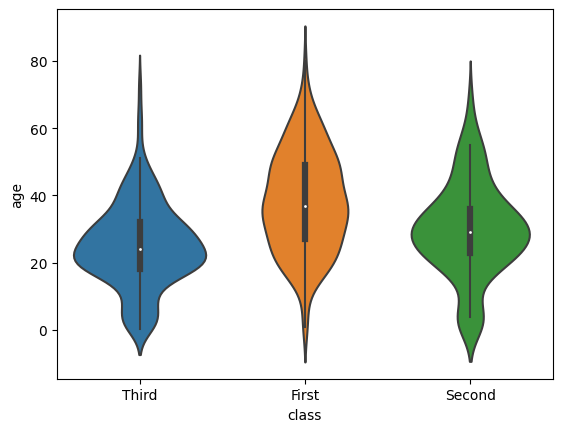
| ***#Boxplot displays the median, quartiles, and extreme values (outliers) of a dataset in a concise manner.* *# The box represents the middle 50% of the data, while the whiskers extend to the highest and lowest values that are not considered outliers.* *#Outliers are represented by individual points beyond the whiskers.* *# Boxplots are useful for detecting outliers and comparing the distribution of data between different groups or categories.* sns.boxplot(x="class",y="age",data=titanic) plt.show()** |
| --- |

****using draw.io

* **Violin plot**



| *# violin plot is like the boxplot but with more details like the width of the plot determins the density of the data in this area* *# the white point in the middle : median* *#the box shape is the first and third quartile* *# the line is the max and min* *# the width/density plot is the density of the data in that area* sns.violinplot(x="class",y="age",data=titanic) plt.show() |
| --- |



* **Joint plot:**

| *# joint plot: shows scatter plot and histogram for each variable allowing us to see the range and distribution of each variable separately.* *# kind=reg; it is the regression line to show the relationship between the two variables* import pandas as pd sns.jointplot(x="total\_bill",y="tip",data=tips\_data,kind='reg') plt.show() |
| --- |

**Flights data set on Pivot**

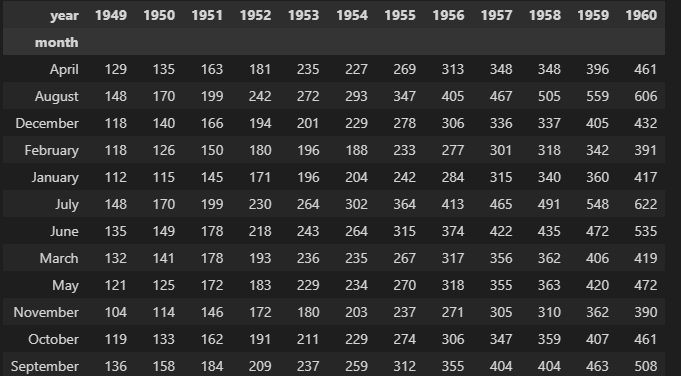
In order to be able to plot the heatmap , you convert the dataset in pivot shape

Flights dataset before pivoting

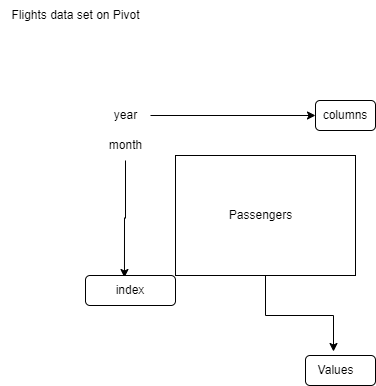


After pivoting

| *# here the pivot exists in two area on Pandas DataFRame and as a stand alone function so we better use the standalone function* flights\_pivot = pd.pivot(flights, index="month", columns="year", values="passengers") flights\_pivot |
| --- |



More explanation using draw.io:



Then now we can do the heatmap on the pivoted dataset

| *#heatmap is 2D graphical representation of data where the values are represented as colors.* *# here after the data is pivoted/rearranged, we can use the heatmap* sns.heatmap(flights\_pivot) sns.heatmap(flights\_pivot,cmap="YlGnBu") *# cmap is for the colors ; here it is from yellow(low values)..green..to blue(high values)* plt.show() |
| --- |

**Tokenization,Stopwords & Ngrams**

The sequence of using ngrams, tokenization, and stop words removal in NLTK data cleaning process is as follows:

Tokenization: This is the process of breaking down a text into smaller units called tokens. Tokenization helps in preparing the text for analysis by breaking it down into meaningful units. This is usually the first step in the data cleaning process.

Stopwords Removal: Stopwords are common words that do not carry much meaning in a text, such as "the", "and", "a", etc. These words can be removed from the text to reduce noise and improve the quality of analysis. Therefore, the next step in the data cleaning process is to remove stopwords.

Ngrams: Ngrams are contiguous sequences of n items (words, letters, etc.) in a text. Ngrams are useful in capturing contextual relationships between words. Therefore, ngrams are usually generated after tokenization and stopword removal.

So, the correct sequence for using these three techniques in data cleaning with NLTK is as follows:

Tokenization: Breaking down the text into smaller units.

Stopword Removal: Removing common words that do not add meaning to the text.

Ngrams: Generating contiguous sequences of words or letters in the text.

It is important to follow this sequence as tokenization should be done before stopword removal and ngrams as tokenization provides the basis for removing stopwords and generating ngrams.

**Logging and threading concepts**

* logging is used to record information about the execution of your program. This information can be used to debug problems and to monitor the performance of your program.
* threading when you want to run multiple tasks concurrently in your program to improve its performance. By creating multiple threads, you can make sure that long-running tasks do not block the execution of other parts of your program.

some examples of when you might use threading and logging:

When you want to download multiple files from the internet simultaneously in a Python program, you can use threading to create multiple threads, each downloading a different file.

When you want to process a large amount of data in a Python program, you can use threading to split the data into multiple parts and process each part in a separate thread.

When you want to monitor the performance of a long-running Python program, you can use logging to record information about the execution of the program, such as the start and end times of each task, and the amount of time each task takes to complete.

| import threading import time  def task1():  print("Starting task 1...")  time.sleep(5) *# Simulating a long-running task*  print("Task 1 completed.")  def task2():  print("Starting task 2...")  time.sleep(3) *# Simulating a long-running task*  print("Task 2 completed.")  *# Creating two threads for running the tasks* thread1 = threading.Thread(target=task1) thread2 = threading.Thread(target=task2)  *# Starting the threads to run the tasks concurrently* thread1.start() thread2.start()  *# Waiting for the threads to complete before exiting the program* thread1.join() thread2.join()  print("All tasks completed.")  *#output*  Starting task 1... Starting task 2... Task 2 completed. Task 1 completed. All tasks completed. |
| --- |

**Using concurrent futures module with ThreadPoolExecuter Class**

The concurrent.futures module provides a high-level interface for asynchronously executing functions using threads or processes. The ThreadPoolExecutor class specifically provides a thread pool that can be used to execute multiple tasks in parallel, which can be useful for IO-bound tasks where the bottleneck is waiting for input/output operations to complete.

Some examples of tasks that can benefit from being executed in a thread pool include:

Downloading multiple files from the internet concurrently

Processing large amounts of data from multiple sources in parallel

Running multiple simulations or calculations concurrently

By using a thread pool, these tasks can be executed more efficiently and with better performance than if they were executed sequentially.

| import logging import threading import time import concurrent.futures  def thread\_function(name):  logging.info("Thread %s:starting ",name)  *#these would basically sleep the program for 2 seconds*  time.sleep(20)  logging.info("Thread %s fineshing",name)  if \_\_name\_\_=='\_\_main\_\_':  format='%(asctime)s:%(message)s'  logging.basicConfig(format=format,level=logging.INFO,datefmt="%H:%M:%S")  with concurrent.futures.ThreadPoolExecutor(max\_workers=5) as executor:  executor.map(thread\_function,range(10))  logging.info('main thread end')  #output:  15:14:09:Thread 0:starting  15:14:09:Thread 1:starting  15:14:09:Thread 2:starting  15:14:09:Thread 3:starting  15:14:09:Thread 4:starting  15:14:29:Thread 0 fineshing  15:14:29:Thread 5:starting  15:14:29:Thread 1 fineshing  15:14:29:Thread 6:starting  15:14:29:Thread 2 fineshing  15:14:29:Thread 7:starting  15:14:29:Thread 3 fineshing  15:14:29:Thread 8:starting  15:14:29:Thread 4 fineshing  15:14:29:Thread 9:starting  15:14:49:Thread 5 fineshing  15:14:49:Thread 6 fineshing  15:14:49:Thread 7 fineshing  15:14:49:Thread 8 fineshing  15:14:49:Thread 9 fineshing  15:14:49:main thread end |
| --- |