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..

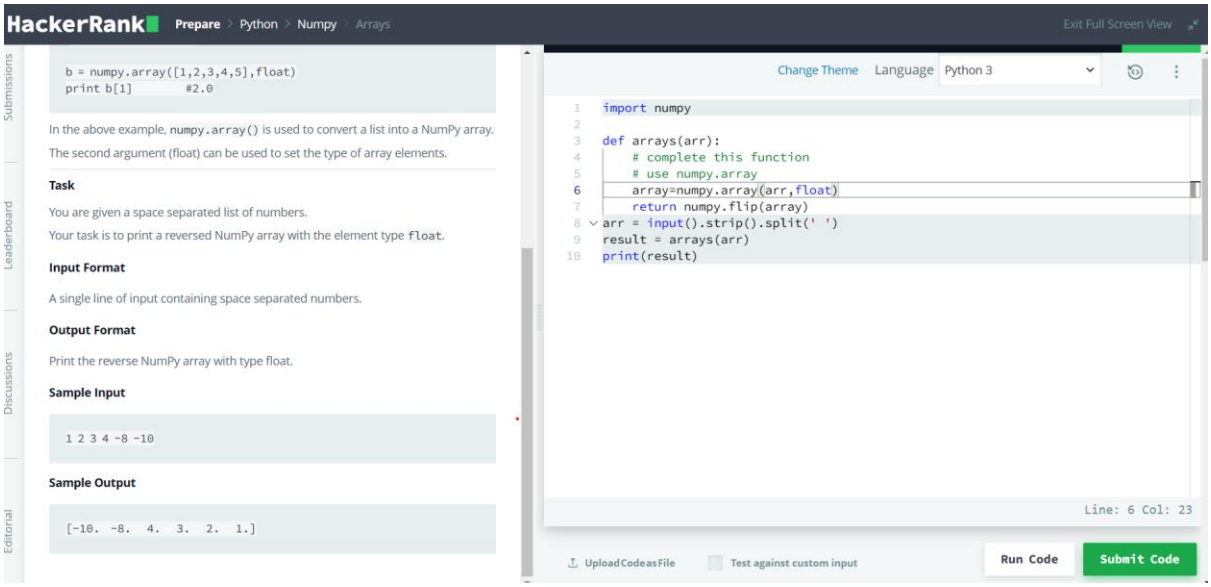
```
#print(small_array_copy)
print(np.sin(small_array_copy))
np.tan(small_array_copy)
np.cos(small_array_copy)
np.log(small_array_copy)
np.log2(small_array_copy)
np.log10(small_array_copy)
np.sqrt(small_array_copy)

#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

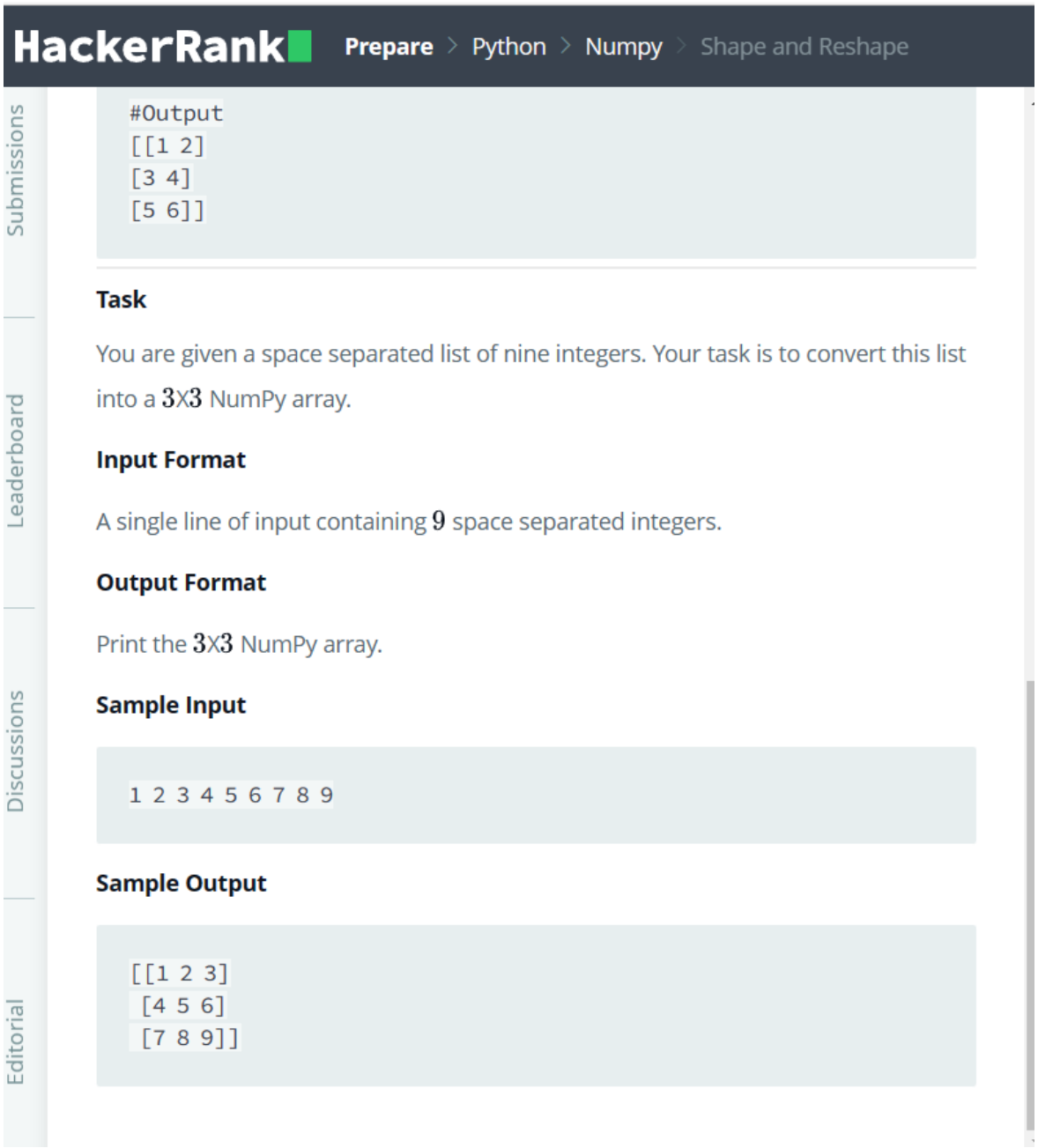
Task: Reversing an array



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.

Task

You are given a NXM integer array matrix with space separated elements (N = rows and M = columns).

Your task is to print the transpose and flatten results.

Input Format

The first line contains the space separated values of N and M.

The next N lines contains the space separated elements of M columns.

Output Format

First, print the transpose array and then print the flatten.

Sample Input

```
2 2
1 2
3 4
```

Sample Output

```
[[1 3]
 [2 4]]
[1 2 3 4]
```

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
```

	OPEID	INSTNM	CITY	STABBR	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
5550	04164400	Angeles Institute	ARTESIA	CA	90701-2713	www.angelesinstitute.edu/	8	21.0	NaN	NaN
3267	00346700	Presentation College	Aberdeen	SD	57401-1280	www.presentation.edu	4	33.0	0.9628	30978.0
3265	00346600	Northern State University	Aberdeen	SD	57401-7198	www.northern.edu	4	33.0	0.8783	19697.0
3724	00377900	Grays Harbor College	Aberdeen	WA	98520-7599	www.ghc.edu	8	41.0	NaN	12392.0
3434	00357100	Hardin-Simmons University	Abilene	TX	79698-0001	https://www.hsutx.edu	6	12.0	0.8177	36644.0
...	...	...	...	...	...	...	...	...	...	...
7053	00108163	Arizona State University at Yuma	Yuma	AZ	853656900	www.asu.edu/	6	NaN	NaN	NaN
2723	00813300	Zane State College	Zanesville	OH	43701-2626	www.zanestate.edu	3	32.0	NaN	12197.0
2753	00310002	Ohio University-Zanesville Campus	Zanesville	OH	43701	www.zanesville.ohiou.edu/	3	32.0	NaN	11074.0
4334	02189000	Mid-EastCTC-Adult Education	Zanesville	OH	43701	www.mideastadulted.org	3	41.0	NaN	NaN
5919	04190500	New York School of Esthetics & Day Spa	white plains	NY	10606-1207	www.newyorkschoolofesthetics.com	2	13.0	NaN	NaN

7058 rows × 10 columns

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
```

	OPEID	INSTNM	CITY	STATE	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
3497	00946607	Brightwood College-McAllen	McAllen	TX	78503-1625	www.brightwood.edu	6	12.0	NaN	NaN
3511	02306809	Platt College-STVT-McAllen	McAllen	TX	78501	www.stvt.edu	6	12.0	NaN	NaN
3553	02285900	UCAS University of Cosemtology Arts & Sciences...	McAllen	TX	78504	www.ucastx.com	6	12.0	NaN	NaN
3557	04178400	Vogue College of Cosmetology-McAllen	McAllen	TX	78501-1945	voguebeautycollege.com	6	12.0	NaN	NaN
4433	03103400	South Texas College	McAllen	TX	78502-9701	https://www.southtexascollege.edu	6	12.0	NaN	6787.0
6095	04213300	Southern Texas Careers Academy	McAllen	TX	78504-2569	stcacademy.edu/	6	12.0	NaN	NaN
6518	03128104	The College of Health Care Professions-McAllen...	McAllen	TX	78504-4398	www.chcp.edu/	6	12.0	NaN	NaN
6866	02218305	Southwest School of Business and Technical Car...	McAllen	TX	78501	www.sws.edu	6	NaN	NaN	NaN

```
data_renaming.query('STATE == "TX" and CITY =="McAllen"')
#the same output using .query()
```

	CITY	INSTNM
0	Normal	Alabama A & M University
1	Birmingham	University of Alabama at Birmingham
2	Montgomery	Amridge University
3	Huntsville	University of Alabama in Huntsville
4	Montgomery	Alabama State University

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']
```

	OPEID	INSTNM	CITY	STABBR	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
0	00100200	Alabama A & M University	Normal	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0
970	00169200	Illinois State University	Normal	IL	61790-1000	illinoisstate.edu/	3	22.0	0.8913	28197.0
4319	03083800	Heartland Community College	Normal	IL	61761-9446	www.heartland.edu	3	22.0	NaN	11877.0
5569	04160600	Paul Mitchell the School-Normal	Normal	IL	61761	paulmitchell.edu/normal/	3	22.0	NaN	NaN

```
#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
```

	INSTNM	ZIP
3	University of Alabama in Huntsville	35899
10	Birmingham Southern College	35254

```
# 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
```

	OPEID	INSTNM	CITY	STABBR	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
1	00100200	Alabama A & M University	Normal	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0
2	00105200	University of Alabama at Birmingham	Birmingham	AL	35294-0110	www.uab.edu	5	12.0	0.9181	24129.0
3	02503400	Amridge University	Montgomery	AL	36117-3553	www.amridgeuniversity.edu	5	12.0	NaN	15080.0
4	00105500	University of Alabama in Huntsville	Huntsville	AL	35899	www.uah.edu	5	12.0	0.8123	22108.0
5	00100500	Alabama State University	Montgomery	AL	36104-0271	www.alasu.edu	5	12.0	0.9787	19413.0

```
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()
```

CITY	OPEID	INSTNM	STABBR	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
Normal	00100200	Alabama A & M University	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0
Birmingham	00105200	University of Alabama at Birmingham	AL	35294-0110	www.uab.edu	5	12.0	0.9181	24129.0
Montgomery	02503400	Amridge University	AL	36117-3553	www.amridgeuniversity.edu	5	12.0	NaN	15080.0
Huntsville	00105500	University of Alabama in Huntsville	AL	35899	www.uah.edu	5	12.0	0.8123	22108.0
Montgomery	00100500	Alabama State University	AL	36104-0271	www.alasu.edu	5	12.0	0.9787	19413.0

```
data_city_index.loc[data_city_index.index=='Montgomery'].head()
#accessing the data using city name as an index
```

CITY	OPEID	INSTNM	STABBR	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
Montgomery	02503400	Amridge University	AL	36117-3553	www.amridgeuniversity.edu	5	12.0	NaN	15080.0
Montgomery	00100500	Alabama State University	AL	36104-0271	www.alasu.edu	5	12.0	0.9787	19413.0
Montgomery	00831000	Auburn University at Montgomery	AL	36117-3596	www.aum.edu	5	12.0	0.8254	19892.0
Montgomery	01303906	South University-Montgomery	AL	36116	www.southuniversity.edu/montgomery#location=Mo...	5	12.0	NaN	26613.0
Montgomery	00100300	Faulkner University	AL	36109-3390	www.faulkner.edu	5	12.0	0.5101	31720.0

```
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
```

STATE	CITY	OPEID	INSTNM	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
MN	Alexandria	00554400	Alexandria Technical & Community College	56308	www.alextech.edu	4	33.0	NaN	17933.0

Dropping the index

```
data = data.reset_index(drop=True)
```

Renaming columns

```
data_renaming = data.rename(columns={'STABBR':'STATE'})
data_renaming
```

	OPEID	INSTNM	CITY	STATE	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
0	00100200	Alabama A & M University	Normal	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0

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
```

	OPEID	INSTNM	STATE	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
0	00100200	Alabama A & M University	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0
1	00105200	University of Alabama at Birmingham	AL	35294-0110	www.uab.edu	5	12.0	0.9181	24129.0

```
data_new = data_new.drop(index=1)
data_new.head() # here it will delete the row number 1
```

	OPEID	INSTNM	STATE	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
0	00100200	Alabama A & M University	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0
2	02503400	Amridge University	AL	36117-3553	www.amridgeuniversity.edu	5	12.0	NaN	15080.0
3	00105500	University of Alabama in Huntsville	AL	35899	www.uah.edu	5	12.0	0.8123	22108.0

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
```

	OPEID	INSTNM	STATE	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
count	4	4	4	4	4	4.0	4.0	3.000000	4.000000
unique	4	4	1	4	4	NaN	NaN	NaN	NaN
top	00100200	Alabama A & M University	AL	35762	www.aamu.edu/	NaN	NaN	NaN	NaN
freq	1	1	4	1	1	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	5.0	12.0	0.897900	19871.750000
std	NaN	NaN	NaN	NaN	NaN	0.0	0.0	0.083304	3524.099731
min	NaN	NaN	NaN	NaN	NaN	5.0	12.0	0.812300	15080.000000
25%	NaN	NaN	NaN	NaN	NaN	5.0	12.0	0.857500	18329.750000
50%	NaN	NaN	NaN	NaN	NaN	5.0	12.0	0.902700	20760.500000
75%	NaN	NaN	NaN	NaN	NaN	5.0	12.0	0.940700	22302.500000
max	NaN	NaN	NaN	NaN	NaN	5.0	12.0	0.978700	22886.000000

```
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
```

	OPEID	INSTNM	CITY	STATE	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
count	2892	2892	2892	2892	2892	2887	2892.000000	2802.000000	945.000000	1824.000000
unique	2892	2866	2470	59	2877	2675	NaN	NaN	NaN	NaN
top	00100200	Lawrence & Company College of Cosmetology	Fairfield	CA	10549	www.empire.edu	NaN	NaN	NaN	NaN
freq	1	2	7	245	2	30	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	4.352697	25.135974	0.685430	24808.920504
std	NaN	NaN	NaN	NaN	NaN	NaN	2.138434	9.724796	0.201733	14808.590257
min	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	-3.000000	0.000000	5185.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	21.000000	0.557500	12962.500000
50%	NaN	NaN	NaN	NaN	NaN	NaN	4.000000	21.000000	0.714500	20445.500000
75%	NaN	NaN	NaN	NaN	NaN	NaN	6.000000	32.000000	0.829400	32977.500000
max	NaN	NaN	NaN	NaN	NaN	NaN	9.000000	43.000000	1.000000	69474.000000

```
#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
```

	zipcode	AGI	STATE
0	01001	56383	MA
1	01002	84212	MA
2	01003	14324	MA
3	01005	58157	MA
4	01007	73439	MA
...	...	...	...
29868	99922	41045	AK
29869	99925	52921	AK
29870	99926	42022	AK
29871	99929	49177	AK
29872	99999	75995	AL

29873 rows × 3 columns

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
```

	OPEID	INSTNM	CITY	STABBR	ZIP	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A	zipcode	AGI	STATE
0	00100200	Alabama A & M University	Normal	AL	35762	www.aamu.edu/	5	12.0	0.9027	22886.0	NaN	NaN	NaN
1	00105200	University of Alabama at Birmingham	Birmingham	AL	35294-0110	www.uab.edu	5	12.0	0.9181	24129.0	NaN	NaN	NaN
2	02503400	Amridge University	Montgomery	AL	36117-3553	www.amridgeuniversity.edu	5	12.0	NaN	15080.0	NaN	NaN	NaN
3	00105500	University of Alabama in Huntsville	Huntsville	AL	35899	www.uah.edu	5	12.0	0.8123	22108.0	NaN	NaN	NaN
4	00100500	Alabama State University	Montgomery	AL	36104-0271	www.alasu.edu	5	12.0	0.9787	19413.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7053	00108163	Arizona State University	Yuma	AZ	853656000	www.acu.edu/	6	NaN	NaN	NaN	NaN	NaN	NaN

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
```

	zipcode	AGI	STATE	OPEID	INSTNM	CITY	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
0	01003	14324	MA	00222100	University of Massachusetts-Amherst	Amherst	www.umass.edu	1	21.0	0.5758	29172.0
1	01013	40873	MA	00214000	College of Our Lady of the Elms	Chicopee	https://www.elms.edu	1	21.0	0.7949	44722.0
2	01040	43005	MA	00217000	Holyoke Community College	Holyoke	www.hcc.edu	1	21.0	NaN	12375.0
3	01056	60275	MA	03205400	Jolie Hair and Beauty Academy-Ludlow	Ludlow	JolieAcademy.com	1	21.0	NaN	NaN
4	01060	75646	MA	02274300	Conway School of Landscape Design	Northampton	www.csld.edu	1	21.0	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
3264	99508	54864	AK	01146200	University of Alaska Anchorage	Anchorage	www.uaa.alaska.edu	8	11.0	0.8283	19126.0
3265	99508	54864	AK	00106100	Alaska Pacific University	Anchorage	www.alaskapacific.edu	8	11.0	0.8600	32489.0
3266	99645	73807	AK	00884300	Alaska Bible College	Palmer	www.akbible.edu/	8	32.0	NaN	19560.0
3267	99669	74066	AK	04138600	Alaska Christian College	Soldotna	www.alaskacc.edu	8	41.0	NaN	19820.0
3268	99723	59308	AK	03461300	Ilisagvik College	Barrow	www.ilisagvik.edu	8	41.0	NaN	20600.0

3269 rows × 11 columns

```
# 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')]
```

	OPEID	INSTNM	CITY	STATE	zipcode	INSTURL	REGION	LOCALE	ADM_RATE	COSTT4_A
1500	00211500	Amherst College	Amherst	MA	01002-5000	www.amherst.edu	1	21.0	0.1290	68986.0
1548	00466100	Hampshire College	Amherst	MA	01002-5001	www.hampshire.edu	1	21.0	0.6282	65153.0
1562	00222100	University of Massachusetts-Amherst	Amherst	MA	01003	www.umass.edu	1	21.0	0.5758	29172.0

```
data_ZipCode[data_ZipCode['STATE']=='MA']
```

	zipcode	AGI	STATE
0	01001	56383	MA
1	01002	84212	MA
2	01003	14324	MA
3	01005	58157	MA
4	01007	73439	MA
...	...	...	...
518	02777	68231	MA
519	02779	77212	MA
520	02780	51801	MA
521	02790	74593	MA
522	02791	119352	MA

523 rows × 3 columns

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-

Which of the following scenarios is ideal for the map function?

Select one:

- To create a function that computes the square root of an input number ✖
- To identify the record with the highest value for a specific attribute
- To identify all records that have a value higher than a certain threshold
- To compute the year from a date field in all records in a file

???

5-

Which of the following scenarios is ideal for the reduce function?

Select one:

- To identify the record with the highest value for a specific attribute
- To identify all records that have a value higher than a certain threshold
- To compute the year from a date field in all records in a file
- To create a function that computes the square root of an input number ✖

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:

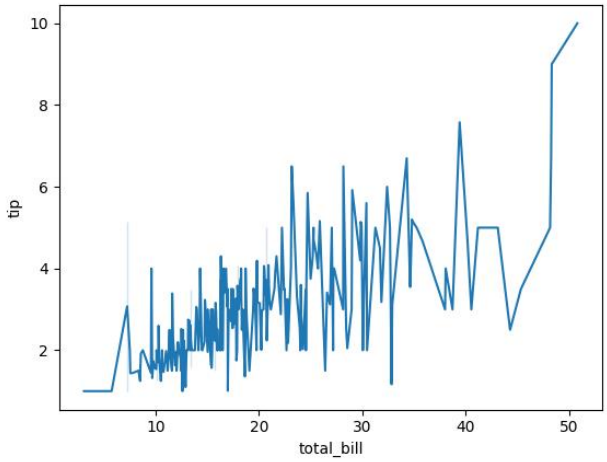
```
Try:
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2
    print("The result is:", result)
except ZeroDivisionError: # if num2 was zero
    print("Error: division by zero")
except ValueError: # if num1 or num2 is anything other than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

Plots/Charts for data visualisation

Types of plots & uses:

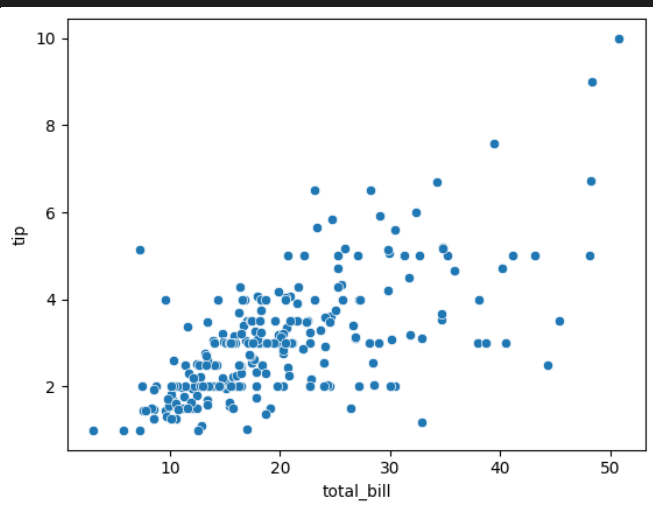
- **Line plot :** displays the data points in straight line between continuous variables (such as time)
  X axis: any cont. Variable like time
  Y axis: the variable being measured
  Efficient in small data points for easy identification
  Uses: finance,economics where the time series data is analysed

```
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

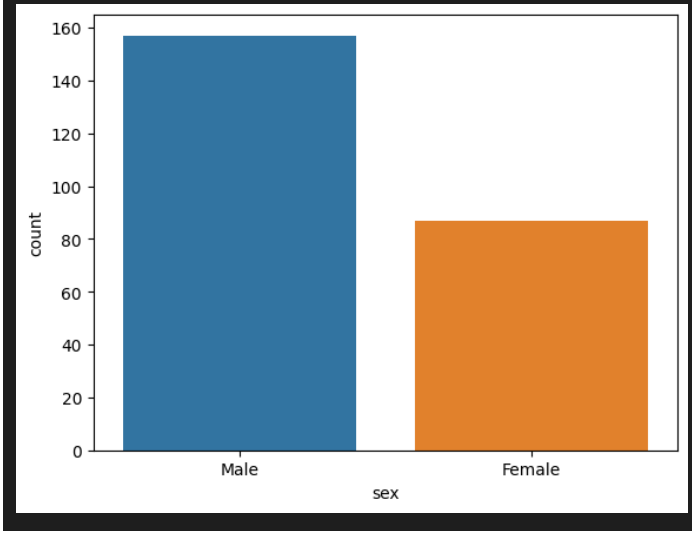
```
#scatterplot: shows relation between two continious variables; outliers and correlations can be detected using scatter plot
sns.scatterplot(x='total_bill',y='tip',data=tips_data)
plt.show()
```



- **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)** :

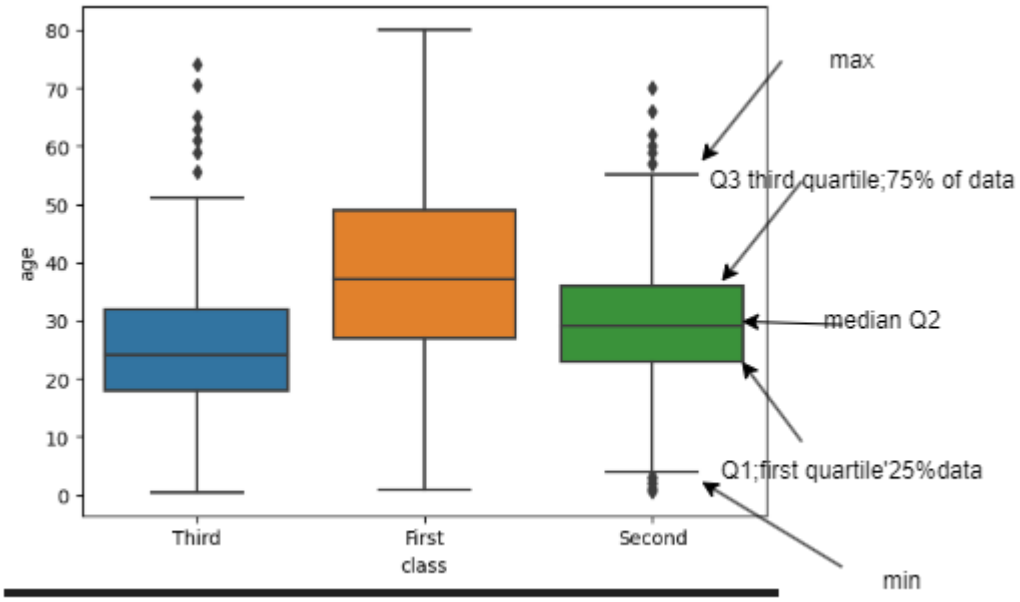
- **Box plot:**

```
#try bar blot
titanic=pd.read_csv("from Jinesh\seaborn-data-master\seaborn-data-
master\\titanic.csv")
Titanic
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

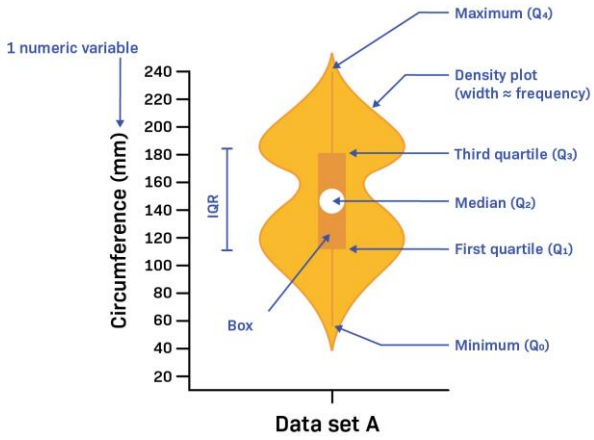
891 rows × 15 columns

```
#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()
```



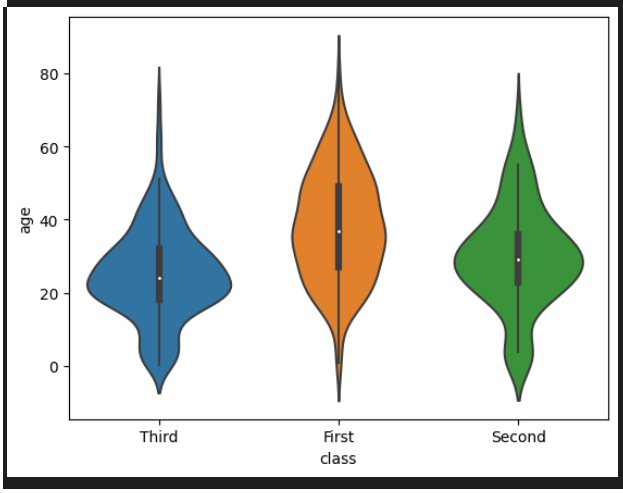
using draw.io

- **Violin plot**



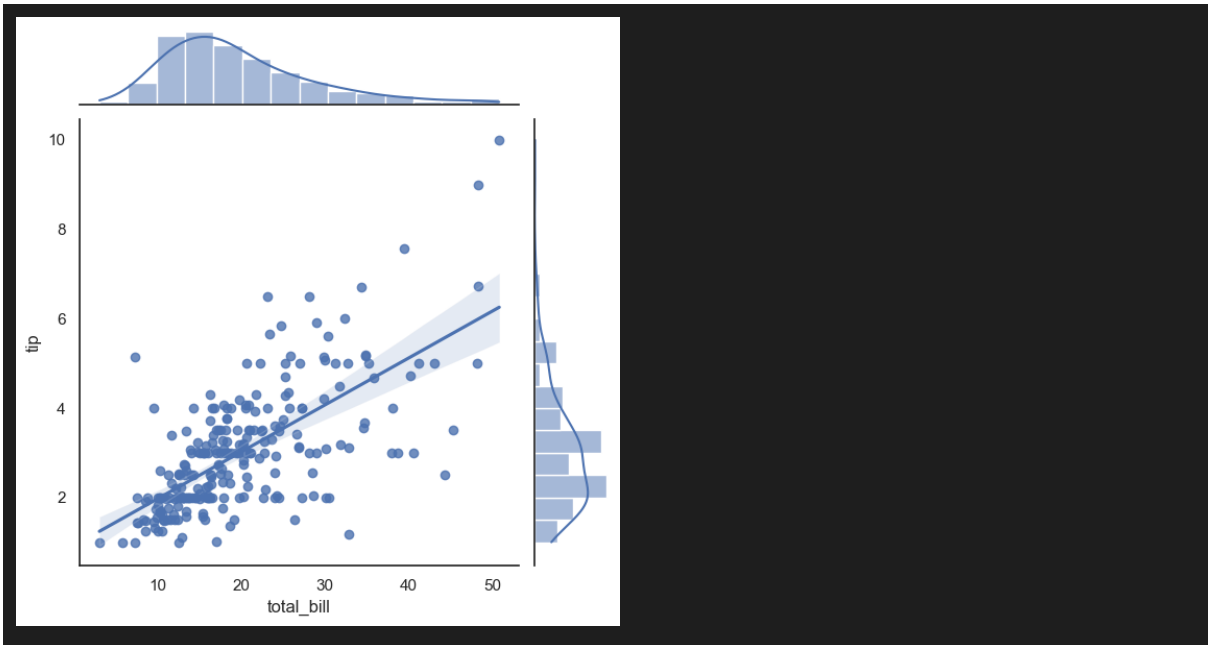

```
# violin plot is like the boxplot but with more details like the width of the plot determins the density of the data in this area
# the white point in the middle : median
```

```
#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

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121
...	...	...	...
139	1960	August	606
140	1960	September	508
141	1960	October	461
142	1960	November	390
143	1960	December	432

144 rows × 3 columns

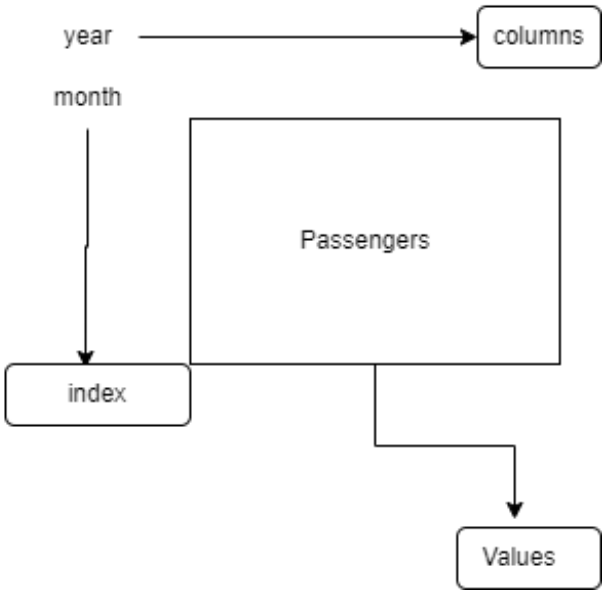
After pivoting

```
# 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
```

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
April	129	135	163	181	235	227	269	313	348	348	396	461
August	148	170	199	242	272	293	347	405	467	505	559	606
December	118	140	166	194	201	229	278	306	336	337	405	432
February	118	126	150	180	196	188	233	277	301	318	342	391
January	112	115	145	171	196	204	242	284	315	340	360	417
July	148	170	199	230	264	302	364	413	465	491	548	622
June	135	149	178	218	243	264	315	374	422	435	472	535
March	132	141	178	193	236	235	267	317	356	362	406	419
May	121	125	172	183	229	234	270	318	355	363	420	472
November	104	114	146	172	180	203	237	271	305	310	362	390
October	119	133	162	191	211	229	274	306	347	359	407	461
September	136	158	184	209	237	259	312	355	404	404	463	508

More explanation using draw.io:

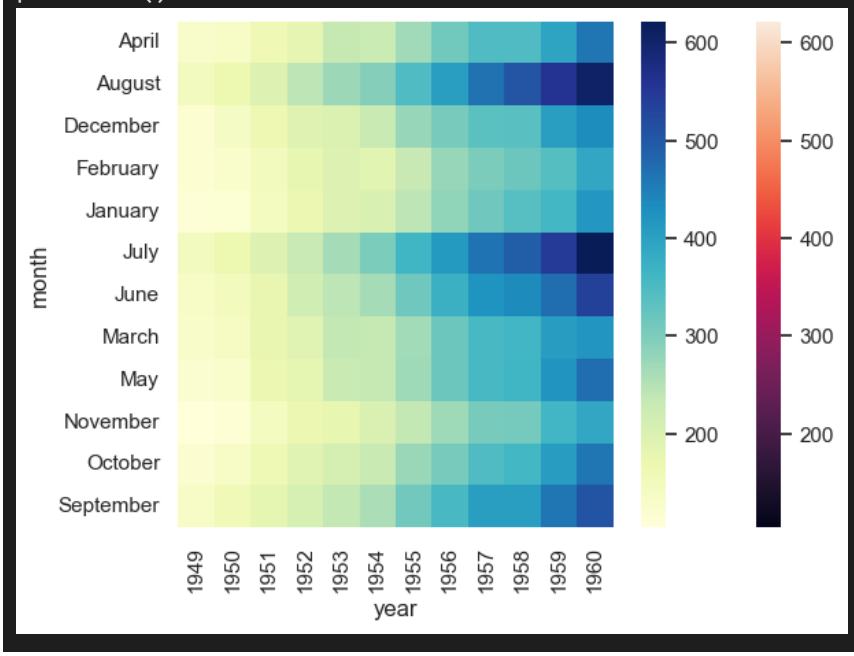
Flights data set on Pivot



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
```